CAHN

Cahn/D1-06-Architecture

Table Of Contents:

- 20-D1-06 Architecture
- CAHN Architecture
- CAHN Overview
  - BRSKI Mapping
  - Interfaces and processes
    - Policy oriented methods
    - Networking methods.
    - Pledge/device methods.
- Policy Introduction
  - Verifiable Credential Technology
- Continuous Assurance
- Claim Flows
- Claim Cascade Design
  - Signing of VCs
  - Receipt of VCs
  - Verification of VCs
  - Trust Engine
  - Trusted VC to Inference Environment
  - Querying the system
  - Databots
  - System Diagram
- claim-schemas
- VOLT architecture
- Continuous Assurance
  - CA Command Protocol
    - CA Protocol Authentication
  - Continuous Assurance Commands
  - Continuous Assurance Use Cases
    - Network Owner Trusts Manufacturer
    - Network Owner Trusts Device
    - Device Has Been Purchased
    - Device No Major Vulnerabilities
- Continuous Assurance Demonstrator
- VC Syntax
- NIST technical specification
- BRSKI overview
- Terminology
- Stages Overview
- 0 - Factory Provisioning
- A - Onboarding Discovery
  - A.1 BRSKI SSID Name Wildcard Match
  - A.2 802.11u eap.arpa
  - Onboarding Process
- B - Discover Registrar
  - Discover Registrar (Full Options)
- C - Request Voucher Registrar
  - C - Request Voucher Overview (Basic)
  - 3 - Request Voucher Overview (Advanced Policy)
- D - Enrol the Device
- E - Join the Network
- F Continuous Assurance of the Network
- Appendix: Key References
  - Footnotes

- Certificate Lifecycle
    - Cert relationships
    - iDevID
    - Domain
    - Registrar
    - Radius
        - Use of Radius certificate
    - Router
        - Use of router certificate
    - LDevID
        - Use of LDevID certificate
- Binary artifact API
    - Voucher binary array
        - copy_binary_array
        - compare_binary_array
        - free_binary_array_content
        - free_binary_array
        - Buffer linked list definition
        - init_array_list
        - free_array_list
        - push_array_list
- BRSKI API
    - BRSKI Core API
        - sign_pledge_voucher_request
        - sign_voucher_request
        - voucher_req_fn
        - sign_masa_pledge_voucher
        - verify_masa_pledge_voucher
- BRSKI Usage
    - Voucher Library Usage
    - Example CMakeLists.txt file
    - BRSKI Tool usage
        - Exporting a pledge voucher request
        - Sending a pledge-voucher request to the registrar
        - Starting the registrar
        - Starting the MASA
- Voucher artifact API
    - Voucher attributes
    - Voucher creation/manipulation API
        - init_voucher
        - free_voucher
        - set_attr_bool_voucher
        - set_attr_time_voucher
        - set_attr_enum_voucher
        - set_attr_str_voucher
        - set_attr_array_voucher
        - set_attr_voucher
        - clear_attr_voucher
        - is_attr_voucher_nonempty
        - get_attr_bool_voucher
        - get_attr_time_voucher
        - get_attr_enum_voucher
        - get_attr_str_voucher
        - get_attr_array_voucher
    - Voucher serialization and deserialization API
        - serialize_voucher
        - deserialize_voucher
    - Voucher CMS signing and verification API

    - sign_eccms_voucher
    - sign_rsacms_voucher
    - sign_cms_voucher
    - verify_cms_voucher
- RFC 8995
- Bootstrapping Remote Secure Key Infrastructure (BRSKI)
  - Abstract
  - Status of This Memo
  - Copyright Notice
  - Table of Contents
  - 1. Introduction
    - 1.1. Prior Bootstrapping Approaches
    - 1.2. Terminology
    - 1.3. Scope of Solution
      - 1.3.1. Support Environment
      - 1.3.2. Constrained Environments
      - 1.3.3. Network Access Controls
      - 1.3.4. Bootstrapping is Not Booting
    - 1.4. Leveraging the New Key Infrastructure / Next Steps
    - 1.5. Requirements for Autonomic Networking Infrastructure (ANI) Devices
  - 2. Architectural Overview
    - 2.1. Behavior of a Pledge
    - 2.2. Secure Imprinting Using Vouchers
    - 2.3. Initial Device Identifier
      - 2.3.1. Identification of the Pledge
      - 2.3.2. MASA URI Extension
    - 2.4. Protocol Flow
    - 2.5. Architectural Components
      - 2.5.1. Pledge
      - 2.5.2. Join Proxy
      - 2.5.3. Domain Registrar
      - 2.5.4. Manufacturer Service
      - 2.5.5. Public Key Infrastructure (PKI)
    - 2.6. Certificate Time Validation
      - 2.6.1. Lack of Real-Time Clock
      - 2.6.2. Infinite Lifetime of IDevID
    - 2.7. Cloud Registrar
    - 2.8. Determining the MASA to Contact
  - 3. Voucher-Request Artifact
    - 3.1. Nonceless Voucher-Requests
    - 3.2. Tree Diagram
    - 3.3. Examples
    - 3.4. YANG Module
  - 4. Proxying Details (Pledge -- Proxy -- Registrar)
    - 4.1. Pledge Discovery of Proxy
      - 4.1.1. Proxy GRASP Announcements
    - 4.2. CoAP Connection to Registrar
    - 4.3. Proxy Discovery and Communication of Registrar
  - 5. Protocol Details (Pledge -- Registrar -- MASA)
    - 5.1. BRSKI-EST TLS Establishment Details
    - 5.2. Pledge Requests Voucher from the Registrar
    - 5.3. Registrar Authorization of Pledge
    - 5.4. BRSKI-MASA TLS Establishment Details
      - 5.4.1. MASA Authentication of Customer Registrar
    - 5.5. Registrar Requests Voucher from MASA
      - 5.5.1. MASA Renewal of Expired Vouchers
      - 5.5.2. MASA Pinning of Registrar
      - 5.5.3. MASA Check of the Voucher-Request Signature

        - 5.5.4. MASA Verification of the Domain Registrar
        - 5.5.5. MASA Verification of the Pledge 'prior-signed-voucher-request'
        - 5.5.6. MASA Nonce Handling
    - 5.6. MASA and Registrar Voucher Response
        - 5.6.1. Pledge Voucher Verification
        - 5.6.2. Pledge Authentication of Provisional TLS Connection
    - 5.7. Pledge BRSKI Status Telemetry
    - 5.8. Registrar Audit-Log Request
        - 5.8.1. MASA Audit-Log Response
        - 5.8.2. Calculation of domainID
        - 5.8.3. Registrar Audit-Log Verification
    - 5.9. EST Integration for PKI Bootstrapping
        - 5.9.1. EST Distribution of CA Certificates
        - 5.9.2. EST CSR Attributes
        - 5.9.3. EST Client Certificate Request
        - 5.9.4. Enrollment Status Telemetry
        - 5.9.5. Multiple Certificates
        - 5.9.6. EST over CoAP
- 6. Clarification of Transfer-Encoding
- 7. Reduced Security Operational Modes
    - 7.1. Trust Model
    - 7.2. Pledge Security Reductions
    - 7.3. Registrar Security Reductions
    - 7.4. MASA Security Reductions
        - 7.4.1. Issuing Nonceless Vouchers
        - 7.4.2. Trusting Owners on First Use
        - 7.4.3. Updating or Extending Voucher Trust Anchors
- 8. IANA Considerations
    - 8.1. The IETF XML Registry
    - 8.2. YANG Module Names Registry
    - 8.3. BRSKI Well-Known Considerations
        - 8.3.1. BRSKI .well-known Registration
        - 8.3.2. BRSKI .well-known Registry
    - 8.4. PKIX Registry
    - 8.5. Pledge BRSKI Status Telemetry
    - 8.6. DNS Service Names
    - 8.7. GRASP Objective Names
- 9. Applicability to the Autonomic Control Plane (ACP)
    - 9.1. Operational Requirements
        - 9.1.1. MASA Operational Requirements
        - 9.1.2. Domain Owner Operational Requirements
        - 9.1.3. Device Operational Requirements
- 10. Privacy Considerations
    - 10.1. MASA Audit-Log
    - 10.2. What BRSKI-EST Reveals
    - 10.3. What BRSKI-MASA Reveals to the Manufacturer
    - 10.4. Manufacturers and Used or Stolen Equipment
    - 10.5. Manufacturers and Grey Market Equipment
    - 10.6. Some Mitigations for Meddling by Manufacturers
    - 10.7. Death of a Manufacturer
- 11. Security Considerations
    - 11.1. Denial of Service (DoS) against MASA
    - 11.2. DomainID Must Be Resistant to Second-Preimage Attacks
    - 11.3. Availability of Good Random Numbers
    - 11.4. Freshness in Voucher-Requests
    - 11.5. Trusting Manufacturers
    - 11.6. Manufacturer Maintenance of Trust Anchors
        - 11.6.1. Compromise of Manufacturer IDevID Signing Keys

        - 11.6.2. Compromise of MASA Signing Keys
        - 11.6.3. Compromise of MASA Web Service
      - 11.7. YANG Module Security Considerations
    - 12. References
      - 12.1. Normative References
      - 12.2. Informative References
    - Appendix A. IPv4 and Non-ANI Operations
    - A.1. IPv4 Link-Local Addresses
    - A.2. Use of DHCPv4
    - Appendix B. mDNS / DNS-SD Proxy Discovery Options
    - Appendix C. Example Vouchers
    - C.1. Keys Involved
      - C.1.1. Manufacturer Certification Authority for IDevID Signatures
      - C.1.2. MASA Key Pair for Voucher Signatures
      - C.1.3. Registrar Certification Authority
      - C.1.4. Registrar Key Pair
      - C.1.5. Pledge Key Pair
    - C.2. Example Process
      - C.2.1. Pledge to Registrar
      - C.2.2. Registrar to MASA
      - C.2.3. MASA to Registrar
      - Acknowledgements
    - Authors' Addresses
  - BRSKI over wifi
  - BRSKI-AE: Alternative Enrollment Protocols in BRSKI
    - Abstract
    - Status of This Memo
    - Copyright Notice
    - Table of Contents
    - 1. Introduction
      - 1.1. Motivation
      - 1.2. Supported Environment
      - 1.3. List of Application Examples
    - 2. Terminology
    - 3. Requirements and Mapping to Solutions
      - 3.1. Basic Requirements
      - 3.2. Solution Options for Proof-of-possession
      - 3.3. Solution Options for Proof-of-identity
    - 4. Adaptations to BRSKI
      - 4.1. Architecture
      - 4.2. Message Exchange
      - 4.3. Enhancements to Addressing Scheme
      - 4.4. Domain Registrar Support of Alternative Enrollment Protocols
    - 5. Instantiation to Existing Enrollment Protocols
      - 5.1. BRSKI-EST-fullCMC: Instantiation to EST (informative)
      - 5.2. BRSKI-CMP: Instantiation to CMP (normative if CMP is chosen)
    - 6. IANA Considerations
    - 7. Security Considerations
    - 8. Acknowledgments
    - 9. References
      - 9.1. Normative References
      - 9.2. Informative References
    - Appendix A. Using EST for Certificate Enrollment
    - Appendix B. Application Examples
      - B.1. Rolling Stock
      - B.2. Building Automation
      - B.3. Substation Automation
      - B.4. Electric Vehicle Charging Infrastructure

        - B.5. Infrastructure Isolation Policy
        - B.6. Sites with Insufficient Level of Operational Security
    - Appendix C. History of Changes TBD RFC Editor: please delete
    - Authors' Addresses
- network integration architecture
- wifi integration method
- lora integration method
- satellite integration method
- 20 5G integration method
- Physical Architecture
    - 5G Network
    - WIFI Network

20-D1-06 Architecture

CAHN Architecture

The nature and ambition of CAHN means it needs to integrate with many different systems, both physical and logical. It necessarily touches many different network types at a low level of detail. As a consequence the documented CAHN architecture is extensive, with critical touch points, with many different networking protocols, the interface of which needs fully defining.

Many of the core concepts in CAHN reflect critical security requirements that have been actively evolving in the NIST trusted onboarding activities. The final NIST Trusted onboarding specification has yet to be published. We have been actively evolving the primary documentation directly on the NIST documents, event though it reflects CAHN activities. Similarly for the BRSKI family of specifications. BRSKI is core the CAHN vision. The practical delivery of CAHN, means actively implementing and evolving the BRSKI specification. And, the same is true of the Volt public documentation. Volt is an NQM product, but defined by an open spec. In order to reduce duplication of effort, we directly include the completed or in progress specifications for the destination technical documents. Hence the CAHN specification is an aggregation of technical architecture documents, which have their final "home" in different organisation and specifications but all of which represent the aggregate efforts of the CAHN initiatives.

The cornerstone of the CAHN specification is the "policy frameworks", this is the area of active innovation that represents the critical commonality between the different networking protocols.

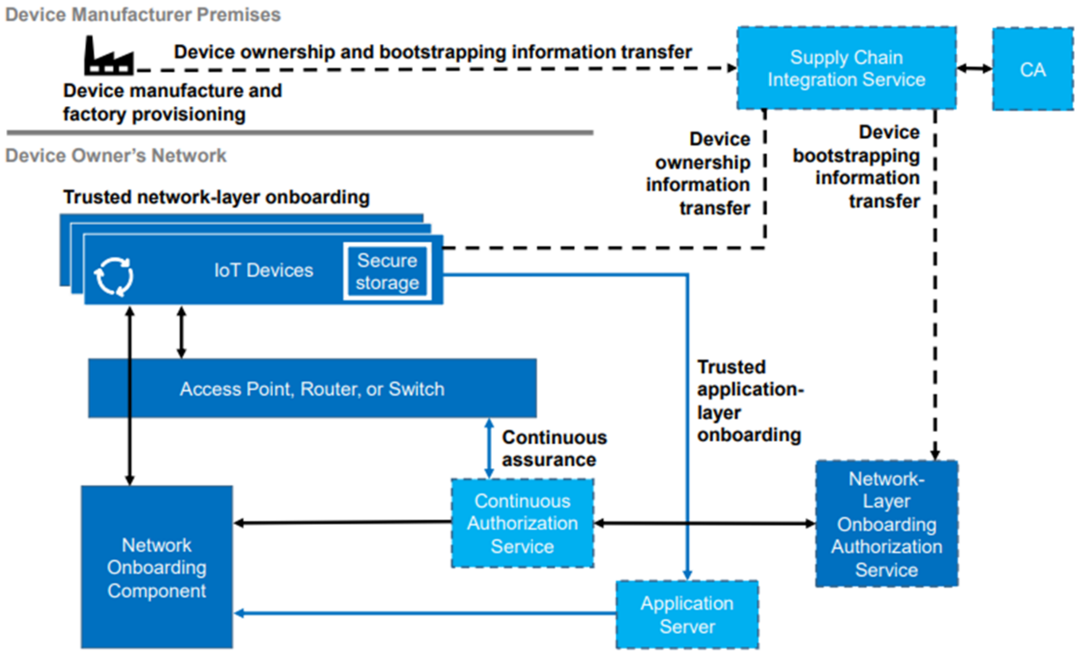
The critical sections of the architecture document are as follows.

- **Architecture overview:** conceptual high level architecture showing how the different parts fit together
- Policy frameworks: detailed design of the verifiable credential based policy core.
- **Integration architecture:** detailed design and specification of the volt system, used the communications and security glue to bind the different pieces together
- **NIST reference architecture:** the complete (close to final) NIST architecture for trusted on boarding and continuous assurance
- **BRSKI specification:** define the IETF method of joining a security domain and the evolving mappings for different networking protocols.
- **Device security architecture**: defines low level user equipment security requirements and specifications, including mappings to various legislative domains
- Networking interface architecture: the detailed binding mappings for each target networking type. (Note this is delivered in full in milstone2)
- **Physical architecture:** detailed architecture documents for the physical deployment architecture at the test sites. Define how the physical networking element fit together at an implementation level.

CAHN Overview

CAHN is delivered as a physical embodiment of the NIST trusted onboarding conceptual architecture

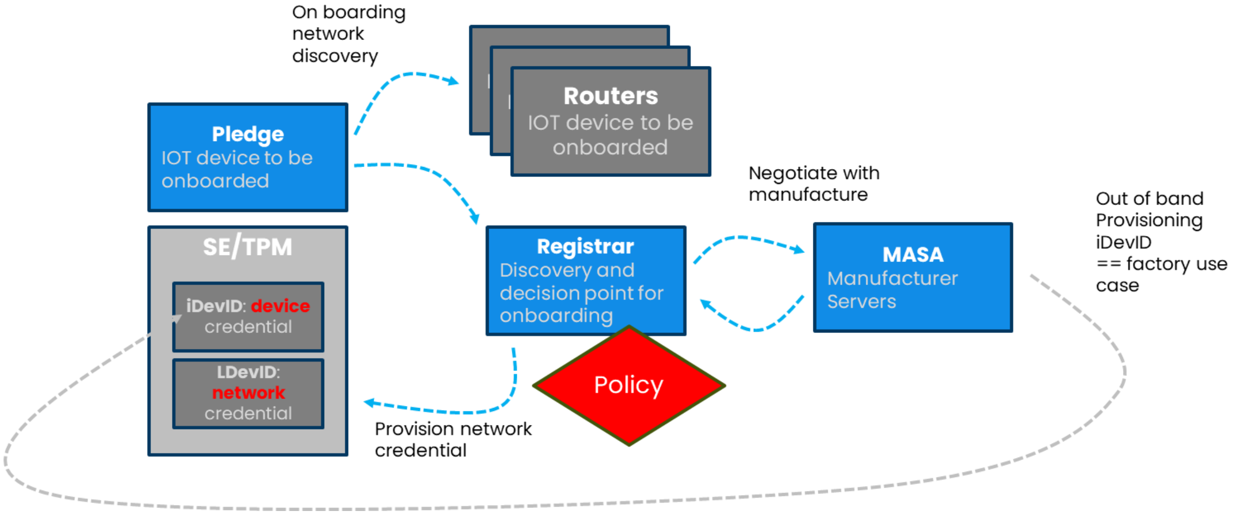
This is defined in full in the originating specific documents, but repeated below


Device Manufacturer Premises
Device ownership and bootstrapping information transfer
Device manufacture and factory provisioning
Supply Chain Integration Service
CA
Device Owner's Network
Device ownership information transfer
Device bootstrapping information transfer
Trusted network-layer onboarding
IoT Devices
Secure storage
Access Point, Router, or Switch
Trusted application-layer onboarding
Continuous assurance
Continuous Authorization Service
Network-Layer Onboarding Authorization Service
Network Onboarding Component
Application Server

The continuous authorisation service is the lynchpin of this target architecture; it provides a concrete method of delivering a zero trust architecture.

BRSKI Mapping

The following schematic shows how this abstract architecture maps onto the BRSKI variation on a WIFI network


On boarding network discovery
Routers
IOT device to be onboarded
Pledge
IOT device to be onboarded
Negotiate with manufacture
Out of band Provisioning iDevID == factory use case
SE/TPM
iDevID: device credential
LDevID: network credential
Registrar
Discovery and decision point for onboarding
MASA
Manufacturer Servers
Policy
Provision network credential

Within the above we have the following critical elements

- Pledge or end user device which is being onboarded to the network
- Registrar: which represents the network owner, it is the logical thing to which the pledge must "belong" before networking credentials can be provisioned
- Routers: are the physical networking device to which the pledge will attach
- MASA: which represents the pledge, manufactures enterprise/cloud services services which are part of the onboarding process.

The process of onboarding goes through the following abstracted phases

0. The assumed precondition: each device has been provisioned with a unique "birthing certificate"; a secure cryptographic identity which is unique to each device
1. Discover onboarding network: the pledge must search for and find candidate onboarding networks, which can be used to start the bootstrap process
2. Discover registrar: once physical onboarding network has been found the device must discover one or more registrars with which they can negotiate
3. Initiate onboarding: the pledge initiates the negotiation to join the network, via the registrar. Optionally this may involve further negotiation with the manufacturer (and other sites)
4. Generate network credential at the registrar and simultaneously notify the "routers" and pass the credential to the pledge
5. Store credential: the pledge should store the credential, ideally in secure storage
6. Attempt to attach: the pledge should attempt to attach to the operational network, and the router should evaluated the request
7. SUCCESS: the pledge is connected to the network
8. Continuous assurance: the registrar continuously evaluates the security posture of the pledge, ejecting it if necessary.

Interfaces And Processes

To scale this solution and adapt it for multiple network types we need to define a number of interoperable interfaces and data structures, as well as implementing some key algorithms.

These are summarised conceptually below and find their detailed specification in the documents to follow

Policy Oriented Methods

- **Policy data formats:** these are the interoperable data structures for sharing data with the policy engine and the outside world. Concretely a family of verifiable credential data structures that embody our policy use cases.
- **Policy interchange protocols**: these are a set of methods for ingesting security relevant data into the policy engine. The nature of verifiable credentials means this can be expansive. Predominately however these will be generic REST interfaces
- **Policy query engine:** the algorithmic core of the policy engine that can operate on ingested VCs and perform structure queries that result in appropriate security actions.

Networking Methods.

For each networking type (5G, Wifi etc) we need to define the following

- **Onboarding discovery:** a method of discovering the onboarding network and candidate registrar
- **Authentication/authorisation hooks**: for each networking type we need a mechanism to hook into the authentication and authorisation flows in order to insert the continuous assurance policy enforcement.
- **Authentication/authorisation hooks**: for each networking type we need a mechanism to hook into the authentication and authorisation flows in order to insert the continuous assurance policy enforcement.
- **Continuous assurance hook**: to complement the authorisation hooks, we need an asynchronous interface that allows the policy engine to notify the network(s) when a mitigation action is needed
- **Credential provisioning **: for each networking type we need a mechanism negotiate/provision the appropriate bearer level networking credential onto the pledge device

Pledge/Device Methods.

For each end device for each networking type (5G, Wifi etc) we need to define the following

- **Onboarding client:** the local client that will do the client side negotiation of the protocols
- **(optional)SE/TPM interface**: optionally for each client we need a method to store security sensitive information (e.g credentials) in a tamper proof form

The detailed CAHN architecture makes these conceptual requirements concrete

Policy Introduction

Continuous Assurance for Heterogeneous Networks (CAHN) is a new technology that underpins network-of-networks, with novel identity models and zero-trust security. [1]

CAHN, delivers continuous assurance, by building on some innovative work that has been developed within NIST as part of the trusted network-layer onboarding program [2]

CAHN will develop new architectures working across networks. It will use advanced notions of identity and distributed credentials, combined with dynamic (AI) reasoning to dynamically infer trustworthiness and assurance. We will work with many different use case and endpoints, with use cases that include Digital Secure by Design hardware silicon to protect against memory vulnerabilities developed with ARM and University of Cambridge. [3][4]

The policy framework is the cornerstone of this approach and is documented in full.

Verifiable Credential Technology

CAHN will be a system which can take in information from many sources and through many channels through the use of Verifiable Credential (VC) technology. With traditional secure communication, such as online banking, the channel by which you communicate information is secured (e.g. the https connection made to your online banking provider of choice, established via standard PKI technology). Using verifiable credentials you instead secure the information you are communicating itself by using PKI technology to sign the packets of information. The advantages of this approach are:

- the information can come via unsecured or out-of-band mechanisms and the information's content can still be tamper-proof and it's source may be cryptographically attested to and verified.

- the provenance of the information may be attested to and tracked, VCs themselves may be signed and attested to as VC recursively, which may be utilised as a method to attest to the mechanism and routes by which the information came into the system.

- By storing the data as VCs then the provenance of each piece of information's origin is maintained, and can be retrospectively reasoned about post receipt of the information. By virtue of this one can also introspect on the VCs issuers to gain insights into the issuers, for example the trustworthiness of those issuers, or the kinds of information they provide, and can model their behaviour to spot unusual activity.

- users consuming data from a system which embodies it's data as VCs can choose which VC issuers and routes to trust, and the time period / conditions in which they trust them. Each user may therefore have a different view of the same data by trusting different VCs, the data host does therefore not impose their own world view of concept of trust onto system users, they gain the autonomy to decide for themselves what information they believe.

- Use of VCs facilitates multiple decentralised signing authorities, rather than relying on one central authority to secure the route via which data enters your system. This mitigates the risks of a central authority being compromised and decouples you from relying on a single providers services, freeing data providers being reliant on a single central source. This also facilitates self-signing of information, supporting the self-sovereign identity model, which gives individuals full ownership and control of their digital data and identity.

- Because the information itself is signed distributed storing of data becomes much easier, as it can be re-communicated and distributed between data lakes later, while maintaining the security and provenance of the information.

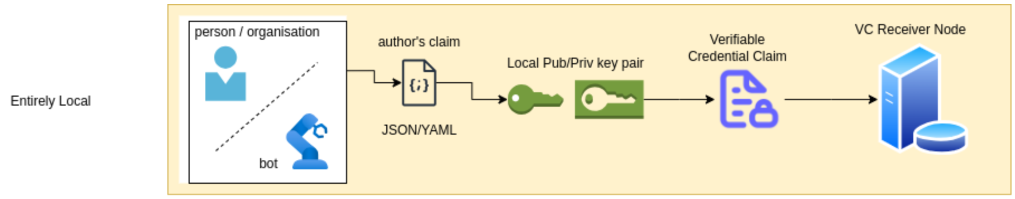
Continuous Assurance

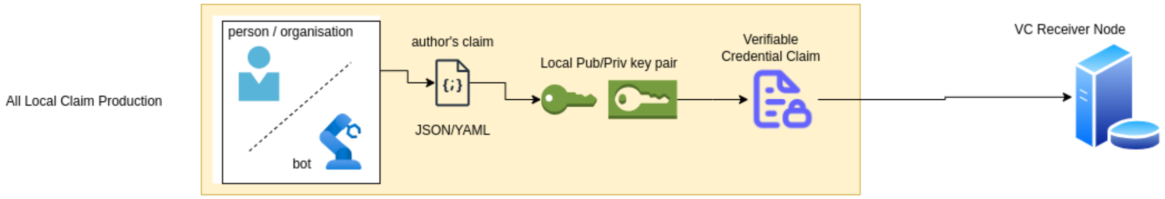
Another piece of our solution is continuous assurance, that is, *continuously* examining the claims made to the system to see if we still believe those piece of information and their sources, and *assuring* that the data and its sources are still trustworthy. Rather than relying on a one-off assurance event the system will continuously apply multiple multiple analytical techniques to infer the trustworthiness and validity of the information claimed to the system.

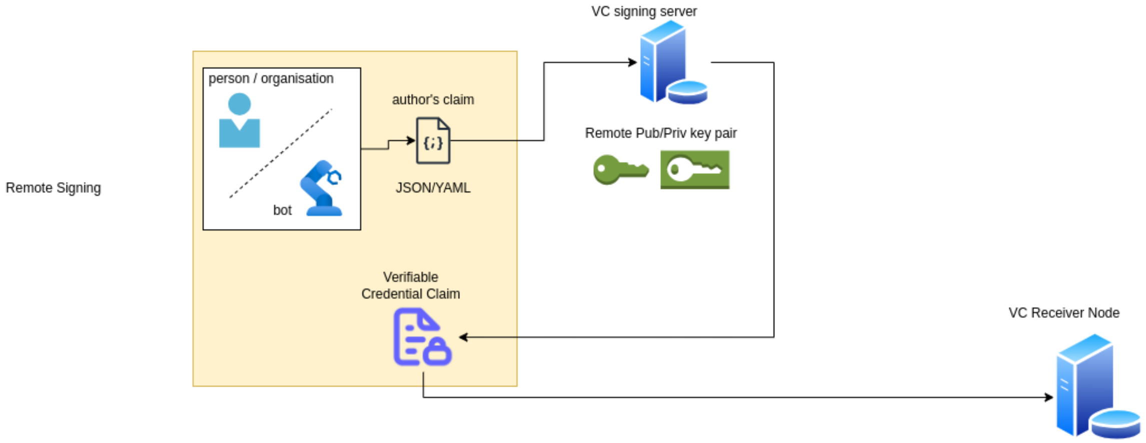
This can be configured to work differently for each data-consuming user in the system, rather than being a top down authoritative approach of prescribing to users what facts, issuers or routes they should trust, we give the freedom to users to configure their own thresholds and metrics for what information, routes of data provenance and issuers they should trust.

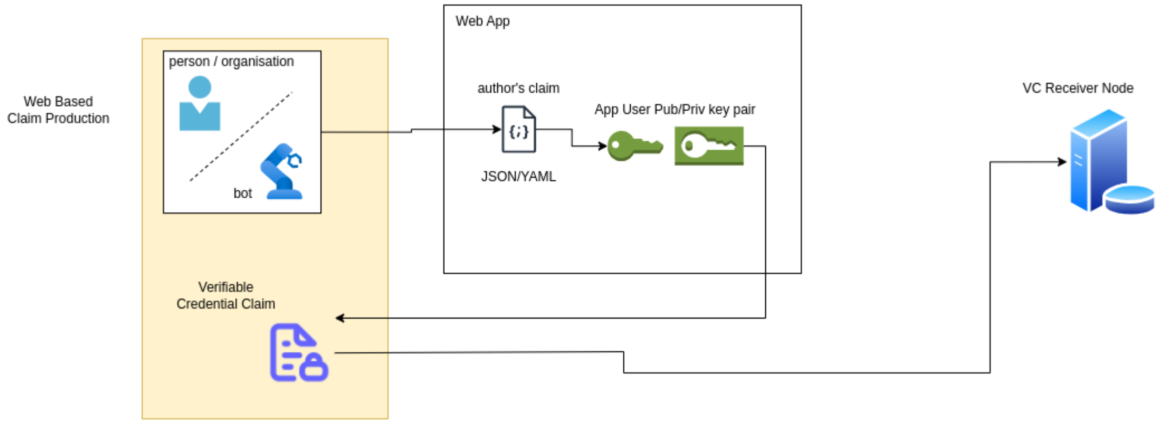
Claim Flows

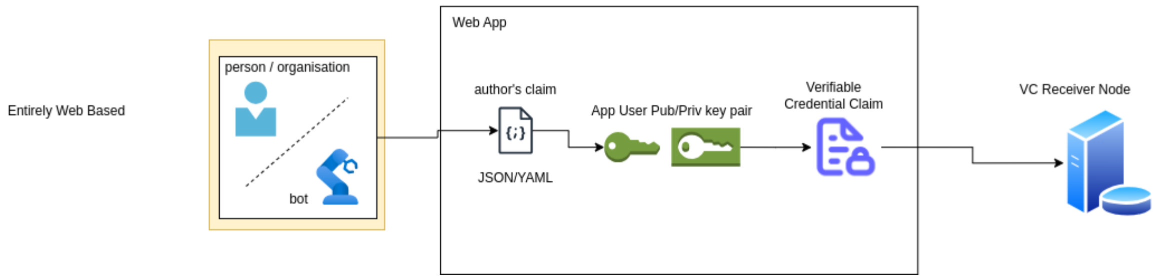
Every piece of information communicated to the system will be claimed in the form of a verifiable credential, and signed by the private key of the user making the claim (henceforth called the claim issuer). Where this signing takes place is completely optional, for example below we've outlined several different schemes by which the VC may be produced, signed and send to a node set up to receive the VC, with differing levels of locality, from everything performed on a local machine, including a node on the local machine, to a user authoring and signing the claim entirely through a web interface.


Entirely Local
person / organisation
bot
author's claim
JSON/YAML
Local Pub/Priv key pair
Verifiable
Credential Claim
VC Receiver Node

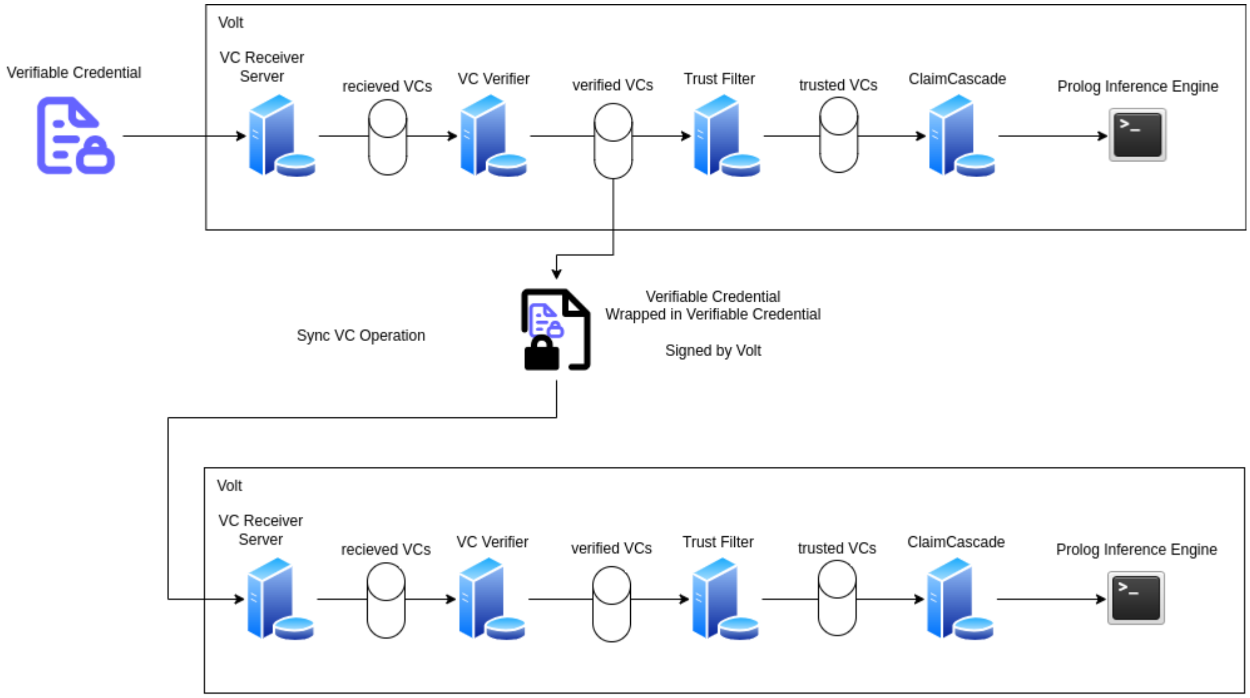

All Local Claim Production
person / organisation
bot
author's claim
JSON/YAML
Local Pub/Priv key pair
Verifiable
Credential Claim
VC Receiver Node


VC signing server
Remote Signing
person / organisation
bot
author's claim
JSON/YAML
Remote Pub/Priv key pair
Verifiable
Credential Claim
VC Receiver Node

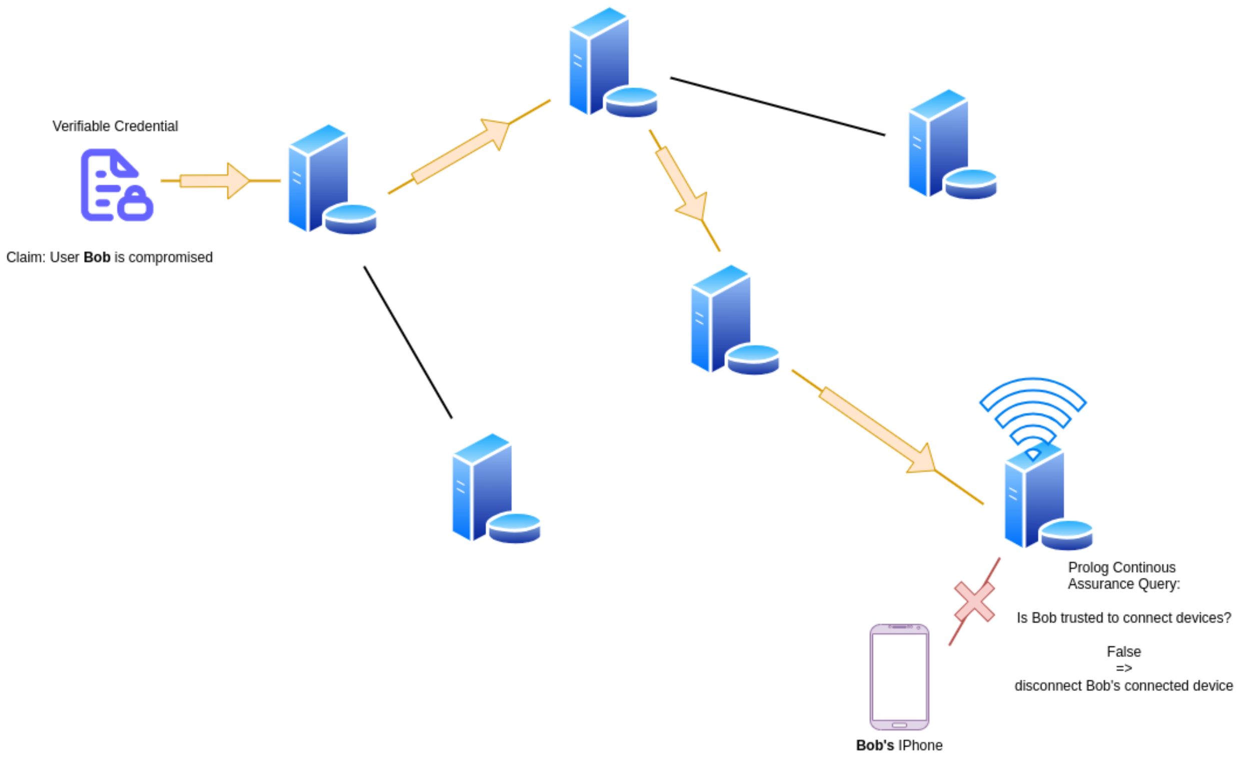

Web App
Web Based
Claim Production
person / organisation
bot
author's claim
JSON/YAML
App User Pub/Priv key pair
VC Receiver Node
Verifiable
Credential Claim


Web App
Entirely Web Based
person / organisation
bot
author's claim
JSON/YAML
App User Pub/Priv key pair
Verifiable
Credential Claim
VC Receiver Node

The VC may be communicated to the a node of the system via a number of different routes, for the simplest example lets assume a REST API endpoint is present on the node which may receive VCs. The VC will be verified to check that it was signed by the private key of the user and that the information contained within has not been tampered with. Next the VC may optionally be propagated to other nodes on the system (encapsulated within a VC signed by the node which received the VC). The next step is that the VC is parsed by the trust engine to determine whether the VC is trusted to make inferences upon, this trust engine may be differently configured on different nodes of the system, if the trust engine determines that the VC is trusted it is passed to the next stage of the system. The next stage of the system parses the information in the verifiable credential and translates it into the appropriate Prolog representation, and updates the current Prolog inferencing environment of that node, such that the encapsulated information may be used to perform inferences.



In this way each node of the system can pass verified information to each other node of the system, and the route by which it got to that node of the network (the provenance of that claim) may be cryptographically attested to by the nesting of VCs.


Verifiable Credential
Claim: User **Bob** is compromised
Prolog Continous
Assurance Query:
Is Bob trusted to connect devices?
False
=>
disconnect Bob's connected device
Bob's IPhone

Claim Cascade Design

Signing Of VCs

The claim cascade system is designed such that users or automated agents may sign claims with their private key (stored locally or stored on a remote server which for signs the claims on the user's behalf), the only requirement for the VC to be verifiable is that public key paired with the private key which was used to perform the signing will need to be securely shared in a trusted way with the system, such that the public key can be tied to a particular user's identity.

This will be achieved by utilising an authentication server which can be sent a request to generate a VC binding a public key to a user's identity, this request will contain the public key and a users credentials, in the most simple case, an email address. These credentials will then be verified, for example in the case of email; the user will receive an email which contains a link they must click to prove their identity, once that is completed the authentication server will sign a VC with it's own private key, which is trusted bt the system, binding the user's identity to that public key. This can be submitted to the system by the user to allow them to submit further VCs signed with their private key.

Receipt Of VCs

VCs will be submitted to a particular Claim Cascade node (running on a router / access point for example) through a HTTPS REST API endpoint. The VC will then be stored in an append only return as a received VC on that node using an Event Sourcing pattern. This means that once received a record of all the VCs received will be maintained and if our trust basis or verification basis changes one could regenerate the entire state of the system from the received VCs.

Verification Of VCs

Once a new VC has been received it will be stored on the node and a verifier process will start, which runs on that VC to verify that the VC was signed by the issuer (the VC issuer is specified in the VC) and has not been tampered with, this requires the verifier to have access to the public key of the issuer, which requires it to have been communicated through a trusted method to the node, or for the node to have access to, and trust, a server which has a record of the pubic key of the issuer. Once a VC is verified it may be passed onto the next stage of the process, which is the trust engine, it may also at this stage be signed by the node's public key as a VC and communicated to other nodes on the system, so they can process the information claimed within.

Trust Engine

The verification step above merely checks that the VC was signed by the issuer contained within and that it has not been altered since signing, it doesn't say anything about whether the issuer of that VC is trusted to be making claims to this node. The trust engine's purpose is to resolve whether we should trust the information contained within the VC and use the information contained within to perform inference. The details of the algorithm to use to determine if we trust the VC issuer is still in development and discussion but the idea is to use the following information to infer whether the issuer is to be trusted:

- which other users trust or distrust the issuer, and the trust we have in those users
- which claims this issuer has submitted in the past

Trusted VC To Inference Environment

The VCs which are trusted are then passed to a utility which generates the code in Prolog for the information contained within the VC. The VCs which may be submitted here come in 4 types:

- schema - this defines the schema for a fact which may be submitted, for example, for a fact that a device has a vulnerability the schema defines how that fact is structured, i.e. what fields are associated with this fact and any constraints on the values which may be entered for each field.
- fact - this defines a fact which should be added to the inferencing environment to be reasoned about, the fact structure should match a schema submitted to the system, for example a fact that a device has a vulnerability
- retraction - a statement that a user retracts a previously submitted fact, used to retract falsely submitted information
- rule - this defines a rule, which defines the logic used for inference, for example a rule may be that a device is not allowed to connect to this wifi router if it has a vulnerability with a score higher than a certain threshold.

Using these building blocks one may build any general inferencing system.

Querying The System

When an external process needs to query the system, this query will take the form of a VC, the verification step will check the query is from the the VC issuer, and the trust engine will check that the VC issuer has the rights to perform that query and access the data required to fulfil the query.

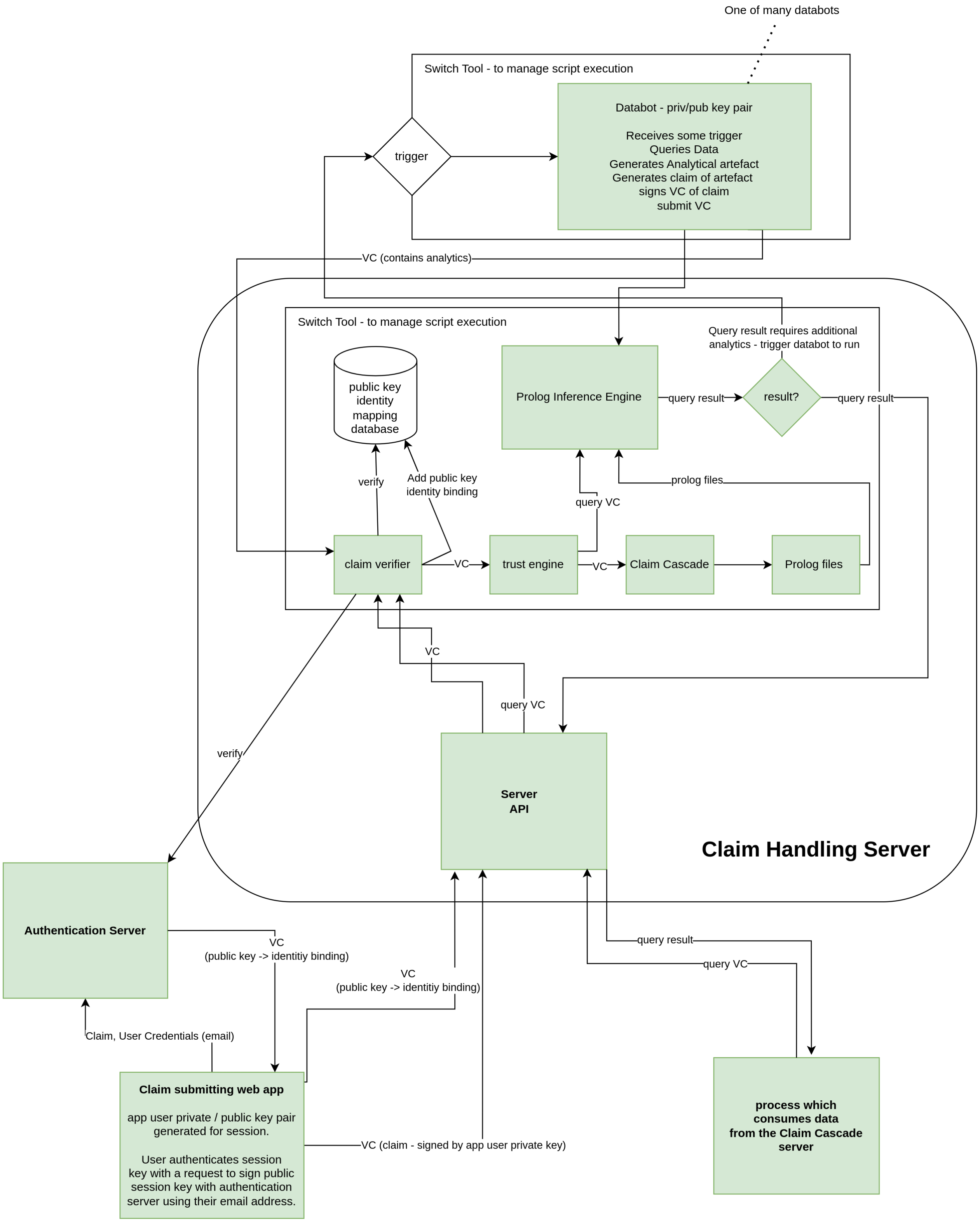
Databots

In order to perform more complex or numerical analytics which cannot be simply performed with a prolog query, for example generating some computed analytical artifact by analysing the data about device behaviour the analytics will be performed by a process set up to run on configurable inputs by a configurable trigger, which we call a databot. Upon a trigger being triggered, the trigger will receive any inputs which need to be configured at run-time and the process will run for those inputs, retrieve any data required for the analytics from the system by means of a query VC submitted to the system, and once it's computation is complete, send its output data into the system by means of a VC. These triggers can be triggered at query time if the information for a query result is not already present in the system, or is in need of update.

System Diagram

Below is a diagram showing the components of the system. The components in the Claim Handling System box will be installed on a node of the system, inside a router or network element. The databots may be installed locally on the node, or run on a remote server. The authentication server, likewise could sit locally on the node, or run on a remote server.

Any process or app which wants to submit VCs to the system must generate a key pair and have it's public key bound to an identity which is verified through the authentication server. The process may submit VCs signed with it's private key and once the public key has been bound, those VC which have been submitted, or will be submitted, will be verified as being issued by their issuer and may proceed to be entered into the knowledge base held in the inference engine, depending on if the issuer is trusted by the trust engine or not.


One of many databots
Switch Tool - to manage script execution
trigger
Databot - priv/pub key pair
Receives some trigger
Queries Data
Generates Analytical artefact
Generates claim of artefact
signs VC of claim
submit VC
VC (contains analytics)
Switch Tool - to manage script execution
Query result requires additional analytics - trigger databot to run
public key identity mapping database
Prolog Inference Engine
query result
result?
query result
verify
Add public key identity binding
prolog files
query VC
claim verifier
VC
trust engine
VC
Claim Cascade
Prolog files
VC
query VC
verify
Server API
Claim Handling Server
Authentication Server
VC
(public key -> identitiy binding)
VC
(public key -> identitiy binding)
query result
query VC
Claim, User Credentials (email)
Claim submitting web app
app user private / public key pair generated for session.
User authenticates session key with a request to sign public session key with authentication server using their email address.
VC (claim - signed by app user private key)
process which consumes data from the Claim Cascade server

Claim-Schemas

Although the policy framework is fully extensible, in order to fulfil the concrete policies use cases, we need very specific interoperable claims to be defined.

Each of these claims is defined as fully specified verifiable credential.

The current draft claims can be found at Schemas

VOLT Architecture

The VOLT system and accompanying SDK provide the implementation level glue that is used to integrate the diverse systems with CAHM

The VOLT provides the following key capabilities

VOLTs implementation and specificaiton has been enhanced considerably inorder to accomodate the current CAHN requirments. We anticipate some further additions as the system evolves.

VOLT provides the following key capabilities

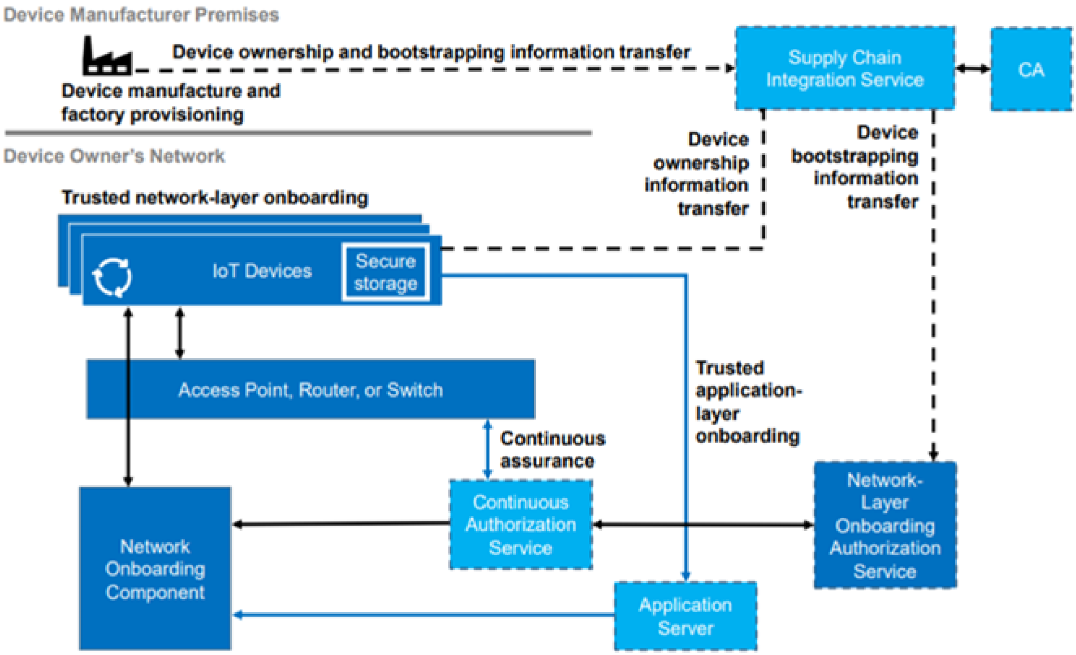
- Identities - and managed rotating keys
- Verifiable credential creation and attestation
- Discovery
- Service publishing
- Proxy services
- Authentication
- Authorisation
- Transport and marshalling

The full specification can be found here

30-tdxvolt docs.pdf

Continuous Assurance

The continuous assurance process is the mechanism by which the `network` continuously monitors the security posture of the connected `device` and responds appropriately.


Device Manufacturer Premises
Device ownership and bootstrapping information transfer
Device manufacture and factory provisioning
Supply Chain Integration Service
CA
Device Owner's Network
Device ownership information transfer
Device bootstrapping information transfer
Trusted network-layer onboarding
IoT Devices
Secure storage
Access Point, Router, or Switch
Trusted application-layer onboarding
Continuous assurance
Continuous Authorization Service
Network-Layer Onboarding Authorization Service
Network Onboarding Component
Application Server

CA Command Protocol

Within the BRSKI architecture, evaluation of network policy happens at the `registrar`.

A many-to-one relationship exists between the `router` and the `registrar`. This connection is initiated by the router, which finds the registrar using the same mechanism the device uses (GRASP or mDNS). The connection is a simple bi-directional TLS session.

CA Protocol Authentication

The authentication method between the router and registrar is intentionally undefined. This is an implementation detail of the setup. For the purposes of this demonstrator, we shall use the notion of "common ownership." The steps for this are:

1. A one-off event: `registrar` generates a public/private key and self-signs a certificate.
2. A one-off event: `router` generates a public-private key pair and self-signs a certificate.

3. A one-off event: a VC is generated to the effect that person@email.com trusts the `registrar` (public key thereof). VC is signed by the private key of the person's DID: **Trust**(person@email.com,`registrar`+) : `person-`.
4. A one-off event: a VC is generated to the effect that person@email.com trusts the `router` (public key thereof). VC is signed by the private key of the person's DID **Trust**(person@email.com,`router`+) : `person-`.
5. The router attempts to connect to the registrar using mutually authenticated TLS. The router passes the router-signed VC as the initiation parameter. The connection is ACCEPTED IF the same person is the signatory, the same person is in both VCs, and the router and registrar public keys match the certificate evidence presented in the mutually authenticated TLS.

Continuous Assurance Commands

There are two primary commands that the `router` must be able to process from the `registrar`:

- Trust device
- Revoke trust on device

We trust the device by sending the following VC on the CA protocol:

- **Trust**(`device+`,`registrar`+) : `registrar-`

We revoke trust by sending the following VC:

- **Revoke**(`device+`,`registrar`+) : `registrar-`

When we revoke trust on the device, we have a number of options:

1. Remove iDevID and LDevID pair from the registrar and reboot wifi
2. Change the VLAN allocation of the device to a constrained network

In this demonstrator, we shall use version 1; specifically, we shall remove the iDevID/LDevID from the permitted list of devices. At an implementation level, this means removing the record from the RADIUS server.

Continuous Assurance Use Cases

The following lists out the different continuous assurance use cases we want to implement and defines the method by which it is implemented.

Network Owner Trusts Manufacturer

The owner of the network specifically trusts the manufacturer.

The following VC is inserted into the registrar:

- **Trust**(person@email.com,`manufacturer`+) : `person-`

Note: If the network owner does not trust the manufacturer, the registrar should NOT forward the voucher request to the manufacturer. This would create information leakage.

Network Owner Trusts Device

The owner of the network specifically trusts the device being onboarded.

The following VC is inserted into the registrar:

- **Trust**(person@email.com,`device`+) : `person-`

Device Has Been Purchased

The device has been purchased by the person.

In the simple case, the manufacturer (MASA) records the purchase.

If the device is owned by the user, then we can infer the user trusts the device.

- **Purchase**(`device+`, person@email.com,) : `manufacturer-` => **Trust**(person@email.com,`device`+) : `person-`

Or more specifically, only where the person trusts the manufacturer:

- **Purchase**(`device+`, person@email.com,) : `manufacturer-` &&
- **Trust**(person@email.com,`manufacturer+`) : `manufacturer-` => **Trust**(person@email.com,`device`+) : `person-`

Device No Major Vulnerabilities

The device is trusted if vulnerabilities are below a threshold.

For each device connected to the network, we need a type identifier. There are a number of options for this, some examples are:

1. A purchase invoice declaring type
2. A device owner declaration
3. An iDevID with a custom attribute
4. An intercepted MUD statement

For the moment we will just consider 1 & 2.

The purchase invoice could have the following form:

- **Purchase**(`device+`, person@email.com, `device-type`) : `manufacturer-`

The device owner declaring could have the following form:

- **OwnerDeclaration**(`device+`, `device-type`) : `person-`

From either declaration, given that the registrar has a list of accepted iDevIDs (the iDevID and the issued LDevID need to be stored at the registrar to implement the RADIUS permission), it is possible to uniquely identify the `device-type`.

The registrar should cache a list of SBOM declarations. These map the device type to the SBOM statement.

- **SBOM**(`device+`, `device-type`, SBOM) : `manufacturer-`

The registrar should intermittently run the CVE function against the declared SBOMs. There should be a configurable CVE threshold function. If the threshold is not met, then a **revoke** command should be sent from the registrar to the routers.

Continuous Assurance Demonstrator

At any moment in time, we should be able to see:

1. All VCs that have been received by the registrar (and time received)
2. Any VCs created through inference (and time inferred)
3. Any VCs issued through the command interface (and time sent)

We should be able to delete any VC. We should be able to manually add new VCs and have a batch of templates to draw from.

VC Syntax

In order to efficiently describe verifiable credentials, we shall use the following simplified syntax.

The notation that we propose is as follows:

```
{ Relation (entity1@, entity2+) } [signatory-]
```

using the normal shorthand expression, where:

- **Relation**: is the relation holding between one or more entities;
- **entity1@**: identifies an entity using an indirect expression, such as a URI;
- **entity2+**: identifies an entity using a direct public key reference;
- **signatory-**: identifies the private key of the entity acting as a signatory of the expression;

and `A -> B` is used to denote message passing, e.g.

```
signatory -> entity2 : { Relation (entity1@, entity2+) } [signatory-]
```

At an implementation level, each statement is encapsulated as a W3C Verifiable Credential (VC).

The schemas used are formally defined in the following section: xxx

NIST Technical Specification

The NIST Trusted Internet of Things (IoT) Device Network-Layer Onboarding and Lifecycle Management a major new initiative define best practice and interoperable specifications relating to "continuous netrwork security"

NQM have invested considerable effort in developing these public spacing specifications, and continue to do so as the work evolves. The NIST work is an integral part of the CAHN vision.

The publications in their current draft version are attached as follows

- Overview nist-sp-1800-36-draft.pdf

- Vol A Executive Summary nist-sp-1800-36a-draft.pdf
- Volume B: Approach, Architecture, and Security Characteristics nist-sp-1800-36b-draft.pdf
- Volume C: How-To Guides nist-sp-1800-36c-draft.pdf
- Volume D: Functional Demonstrations nist-sp-1800-36d-draft.pdf
- Volume E: Risk and Compliance Management nist-sp-1800-36e-draft.pdf

BRSKI Overview

BRSKI defines and open IETF specification that securely on boards a device to a "security domain"

Joining security domain is manifest, within the provisioning of a an appropriately signed certificate on the target device.

The joining of a security domain is then dovetailed wiht the physical provisioning of network credentials on a device. This can be different for each network type.

The initial work in this area has focussed on the onboarding onto WIFI networks, specifically using EAP-TLS as the network provisioning method. As the development matures, these protocols will be asbtract3ed and refined to work over many network types.

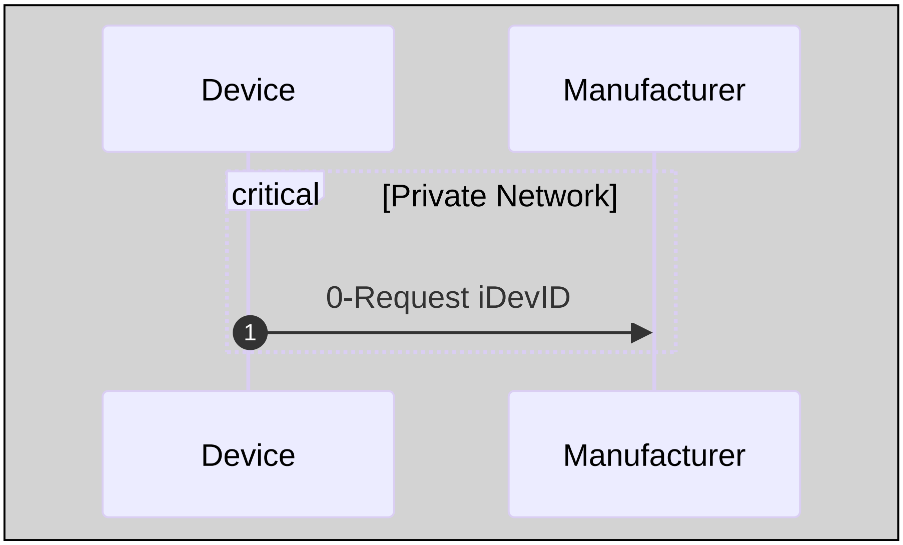
Terminology

We will use the term `device` interchangeably with the term `pledge` for easier reading.

- `device`: The device wishing to onboard to the network.
- `registrar`: The principal decision-making entity in the onboarding process.
- `proxy`: A proxy to the registrar - useful if the registrar is located externally or shared between networks.
- `MASA`: The Manufacturer Authority, which issues manufacturer assurances (like iDevID) and can attest to the validity of these assurances.
- `router`: The physical router to which a device is connecting; the router may potentially host many networks.
- `onboarding network`: A constrained network used to bootstrap the onboarding process.
- `target network`: The operational network to which the device intends to attach.
- `iDevID`: A unique device certificate issued by the manufacturer.
- `LDevID`: A certificate used by the device to gain network access (EAP certificate).

Stages Overview

- O - Factory Provisioning
- A - Discover Onboarding Network
- B - Discover Registrar
- C - Request Voucher
- D - Enrol the Device
- E - Join the Network
- F - Continuous Assurance of the Network



0 - Factory Provisioning

The demonstrated factory provisioning flow is as follows:

- 0.1 - Device is pre-provisioned with the manufacturer's CA and URI for the manufacturer's servers.
- 0.2 - Device generates a unique public/private key pair.
- 0.3 - Device requests an iDevID from the manufacturer.
- 0.4 - The returned iDevID is installed on the device.

A - Onboarding Discovery

There are two methods for discovering potential onboarding networks:

- A.1 - Search for public Wi-Fi networks matching a particular SSID wildcard name.
- A.2 - Search for Wi-Fi networks advertising a particular realm.

A.1 BRSKI SSID Name Wildcard Match

The device will search for all SSIDs matching the wildcard as specified in [1].

The device will iterate round-robin across successful pattern matches in strength order.

Every time a device finds a viable match, it will connect to the onboarding network and attempt to discover the registrar.

A.2 802.11u Eap.Arpa

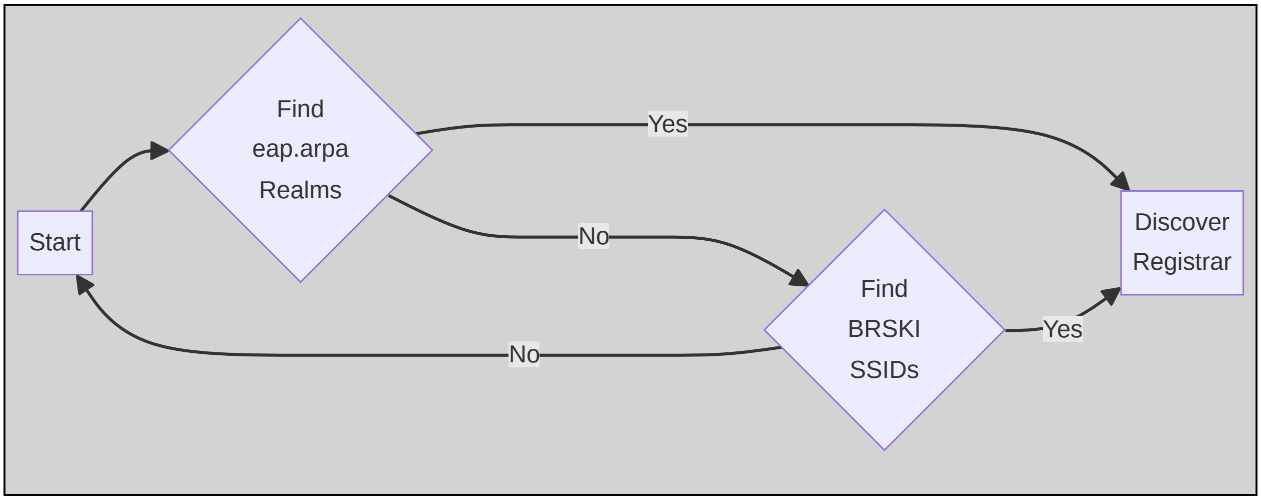
The device will search for all networks supporting the `eap.arpa` realm.

The device will iterate round-robin across successful pattern matches in strength order.

Every time a device finds a viable match, it will connect to the onboarding network and attempt to discover the registrar.

Onboarding Process

The device will prefer onboarding networks with realm support over BRSKI SSID matches.



B - Discover Registrar

When the device discovers a candidate onboarding network, it will attempt to discover the registrar.

If the registrar is non-discoverable, this onboarding network will be temporarily marked as failed, and the onboarding process will proceed to the next candidate onboarding network.

The device operates either in IPV6 or IPV4 mode. The options for Registrar discovery are slightly different in each case.

For the purposes of the NIST Build 5 demonstrator, we shall use the mDNS method of directly discovering the registrar as outlined in Appendix A [2].

> Discovery of the registrar **MAY** also be performed with DNS-based Service Discovery by searching for the service "_brski-registrar._tcp.example.com". In this case, the domain "example.com" is discovered as described in [RFC6763], Section 11 (Appendix A.2 of this document suggests the use of DHCP parameters).

Specifically:

- Device obtains an IP address via DHCP as per A.2 [2].
- Device listens for service announcement `"_brski-registrar._tcp.example.com"` as per Appendix B [2].
- Device secures IP address of candidate Registrar.
- Device attempts to initiate voucher request.

Discover Registrar (Full Options)

TBD: Outline the full list of methods for discovering the registrar.

C - Request Voucher Registrar

Preconditions: Before we initiate the Request Voucher, we assume the following conditions are met:

- `device` is provisioned with a valid `iDevID`.
- `device` has connected to a candidate `onboarding network`.
- `device` has a valid IP address on the `onboarding network`.
- `device` has discovered the IP address of a candidate `registrar` (or a `proxy`).

Post Conditions (Success): If the voucher request is successful:

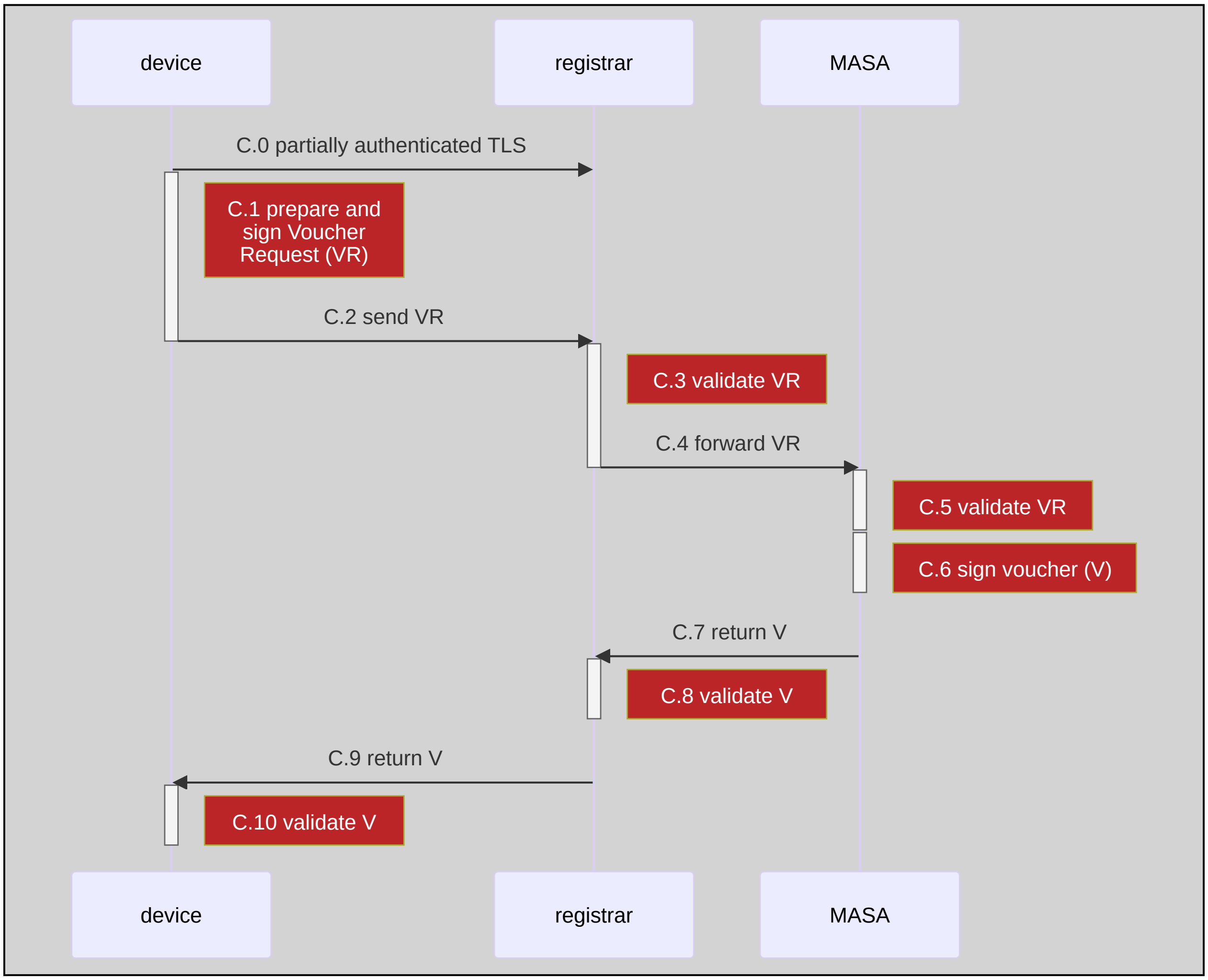
- `device` is in possession of a valid `voucher`.
- where the tests that need to pass are
  - voucher has not been revoked, which requires:
    - behaviour of IoT device bound by MUD descriptor
    - No requests to blacklisted domains / IP addresses
    - Manufacturer is trusted
    - Device is trusted or is owned by a user who is trusted to connect devices
    - SBOM vulnerability score lower than set threshold

C - Request Voucher Overview (Basic)

The complete flow of the voucher request process is as follows:

- C.0 `device` creates a partially authenticated TLS connection with registrar.
- C.1 `device` constructs `voucher request` construct request and signs it with `iDevID` private key.
- C.2 `device` sends `voucher request` to `registrar`.
- C.3 `registrar` validates `voucher request`.
- C.4 `registrar` forwards `voucher request` to `MASA`.
- C.5 `MASA` validates `voucher request`.
- C.6 `MASA` signs `voucher`.
- C.7 `MASA` returns `voucher` to `registrar`.

- C.8 `registrar` validates `voucher`.
- C.9 `registrar` returns voucher to `device`.
- C.10. `device` validates `voucher`.



3 - Request Voucher Overview (Advanced Policy)

Validation processes exist at stages:

- C.1 - `device` constructs `voucher request` construct request and signs it with `iDevID` private key.
- C.3 - `registrar` validates `voucher request`.
- C.5 - `MASA` validates `voucher request`.
- C.8 - `registrar` validates `voucher`.
- C.10 - `device` validates `voucher`.

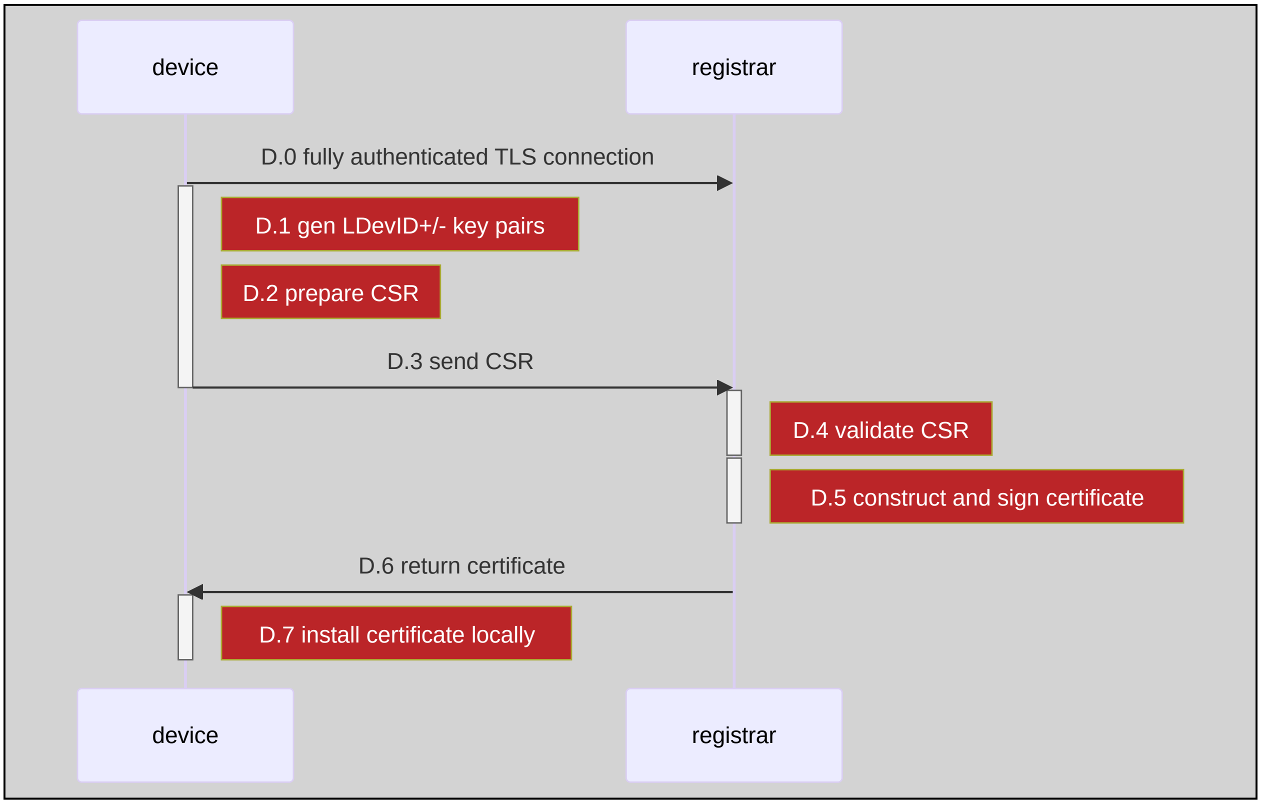
At each of these stages, there is the option to evaluate and enforce a policy decision.

C.3 and C.8 are validation and policy enforcement points implemented at the registrar and therefore ideal for implementing the core networking policy.

D - Enrol The Device

Enrolling the device consists of the following steps:

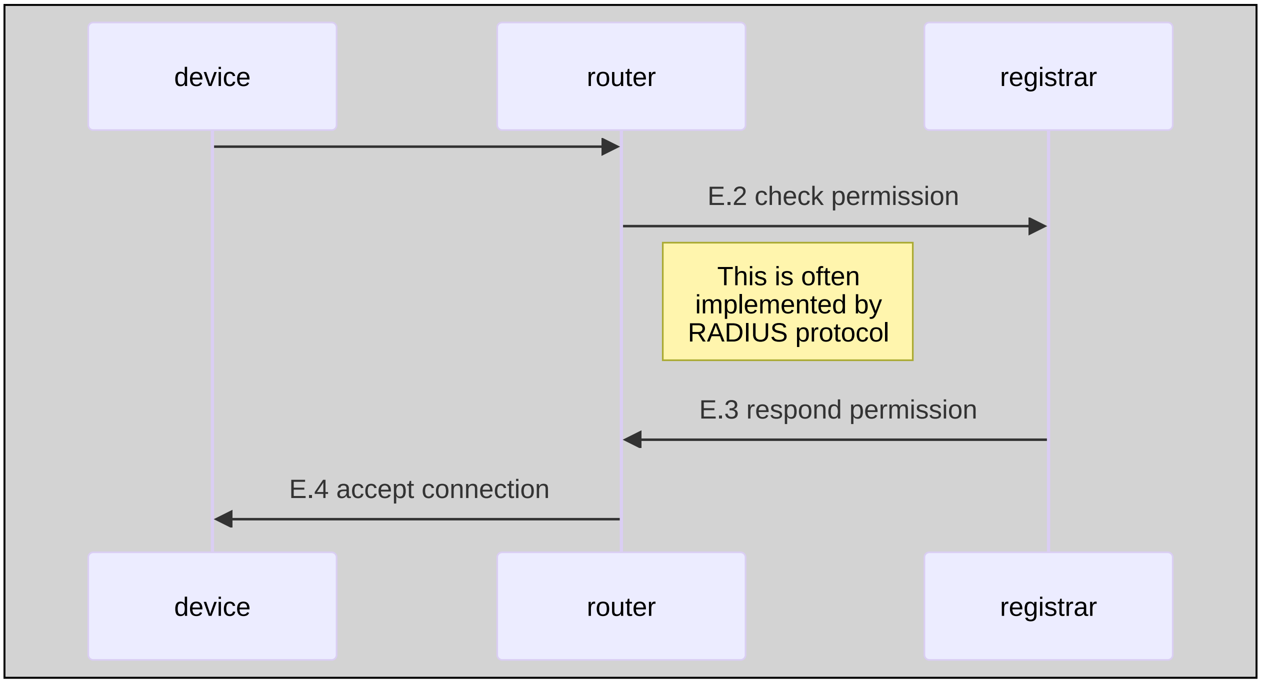
- D.0 - Fully authenticate the TLS connection, using iDevID (using the pinned cert in the voucher response).
- D.1 - Generate LDevID public/private key pair.
- D.2 - Device constructs the CSR request for enrolment, which includes the iDeviD and is signed by iDeviD.
- D.3 - Device sends the CSR to the registrar (over the authenticated TLS session).
- D.4 - The registrar validates the CSR request.
- D.5 - The registrar constructs the certificate response (LDevID).
- D.6- The registrar returns the certificate to the device.
- D.7 - The device saves the LDevID (network credentials) locally ready to attach to the network.


device
registrar
D.0 fully authenticated TLS connection
D.1 gen LDevID+/- key pairs
D.2 prepare CSR
D.3 send CSR
D.4 validate CSR
D.5 construct and sign certificate
D.6 return certificate
D.7 install certificate locally
device
registrar

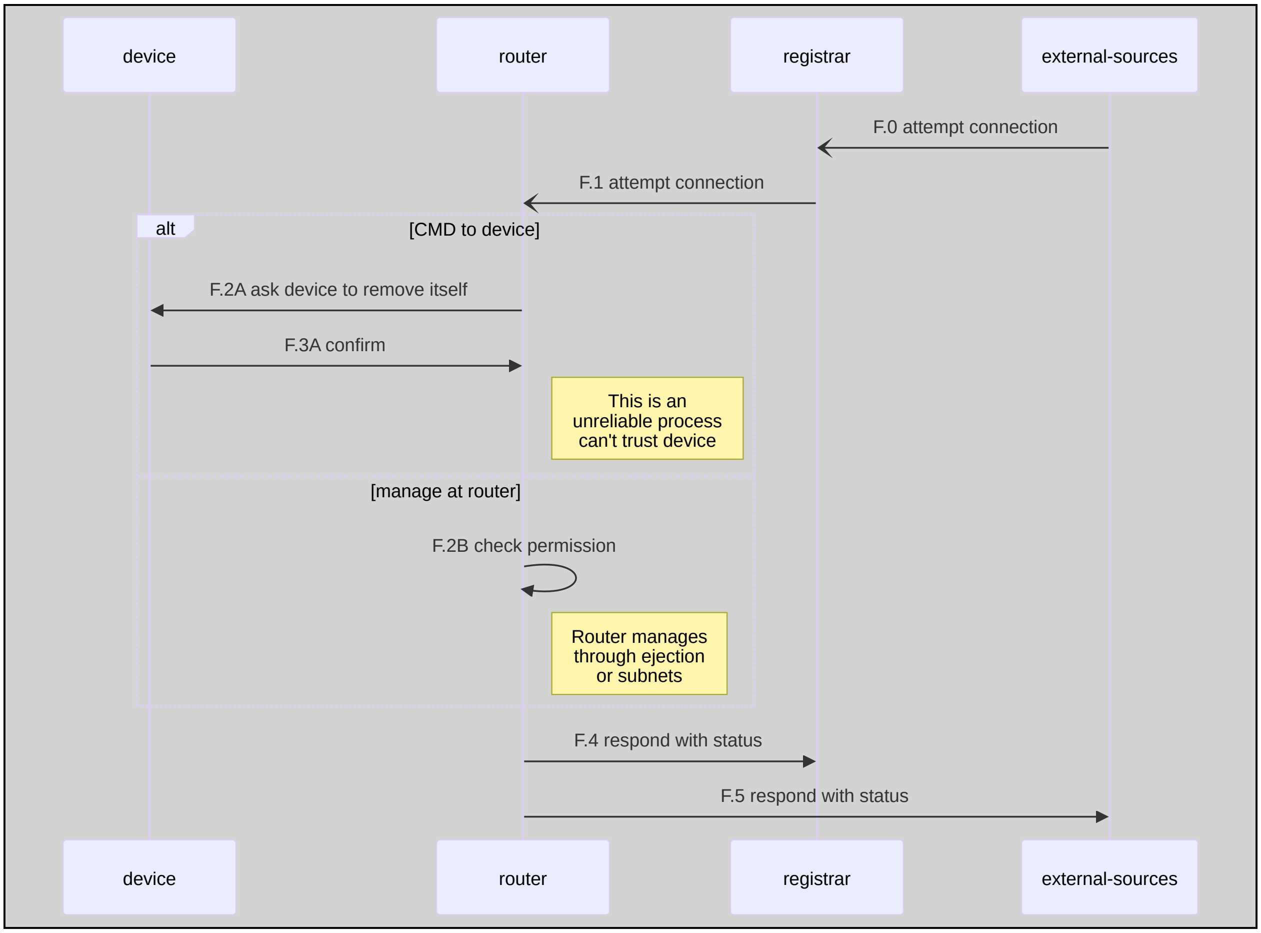
E - Join The Network

Joining the network can be triggered by the device as soon as the device is in possession of a valid LDevID (or other network credential).

The router will receive the network connection request. It may confer with the registrar to check that the device is adequately permissioned to join the network. Typically, this may be performed through the RADIUS interface.



F Continuous Assurance Of The Network



For the full details of the continuous assurance process, see the reference document Continuous Assurance.

Appendix: Key References

Footnotes

1. https://ftp.kaist.ac.kr/ietf/draft-friel-brski-over-802dot11-00.txt ↩

2. https://datatracker.ietf.org/doc/rfc8995/ ↩ ↩[2] ↩[3]

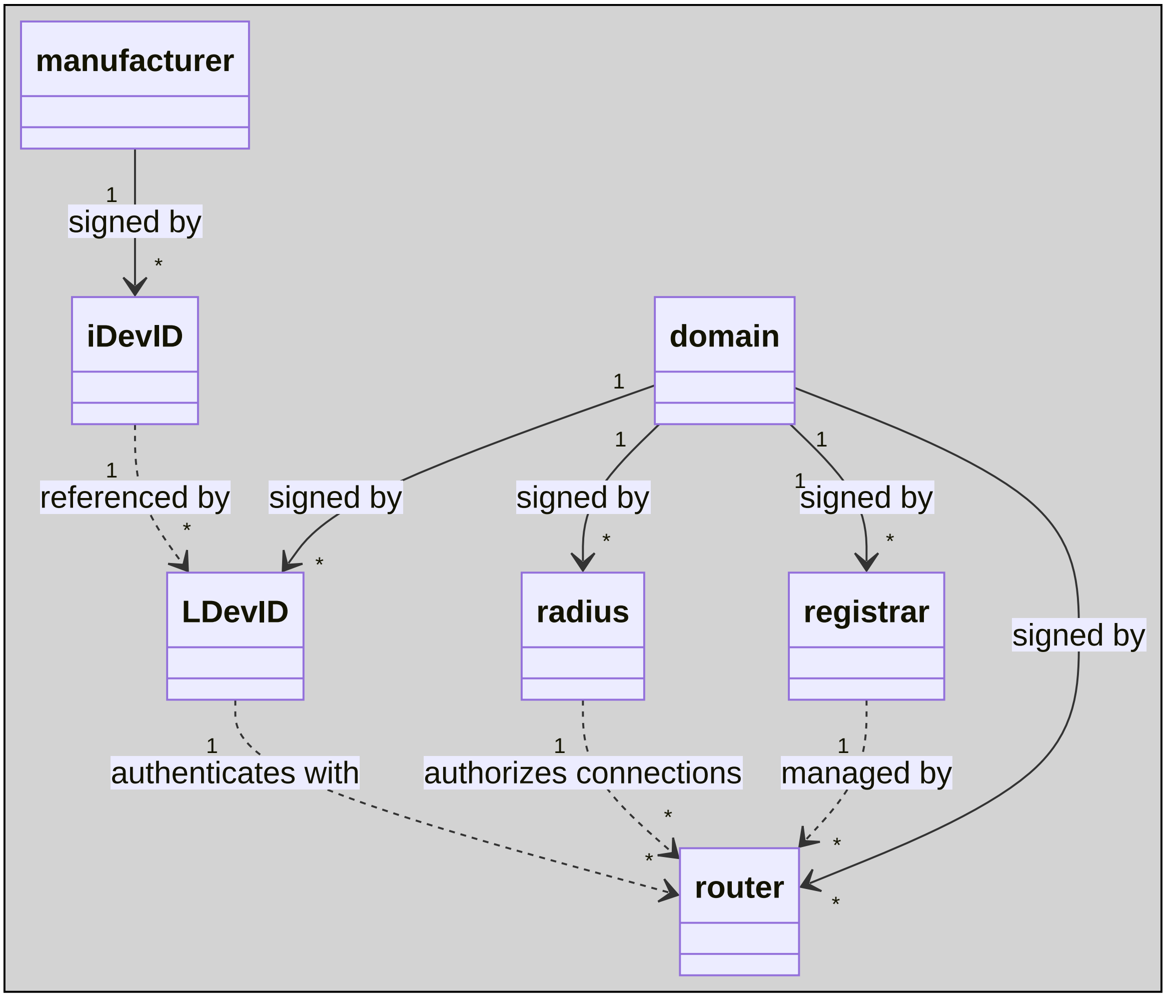
Certificate Lifecycle

Certificates are used throughout the BRSKI-WIFI onboarding process.

It is helpful to clearly distinguish how these are used in a practical implementation and outline how they are structured

Cert Relationships

- `iDevID`: is the unique identifier for the device, and is typically only created once
- `LDevID`: are created when a device is onboarded and represents the evidence that the device has joined the logical domain
- `domain`: is the logical domain to which the device is onboarded, and implicitly to which the network also belongs
- `registrar`: is the intermediate mechanism by which a device joins a logical domain.
- `radius`: is not mentioned directly by the BRSKI specification, but is used in the implementation. Logically the radius and the registrar are in a 1:1 relationship
- `manufacturer`: the originating manufacture of the device
- `router`: each router must have an end point certificate,



IDevID

The iDevID sits on the device (pledge).

It is typically created as a one off process. It should not change during the devices lifecycle

It typically is installed by the manufacture as a privileged process.

The method of installing a manufacturer created iDevID is covered in depth in the factory use case. The basic flow however is as follows.

Creation

1. [@ DEVICE] create a public private key pair
2. [@ DEVICE] create a CSR
3. [TO MANUFACTURER] send CSR
4. [@ MANUFACTURER] validate CSR

5. [@ MANUFACTURER] create and sign certificate with manufacture key
6. [TO DEVICE] return signed iDevID certificate
7. [@ DEVICE] locally install iDevID certificate

X509 Attribute	Description/use

`Subject`	Name of device (optional)
	CN="serial number"
	OU="model name"

`Subject Key Identifier`	Public key of the `device+`

`Issuer`	Name of the manufacture
	CN="Manufacturer ltd"
	OU="www.manufacture.com"

`Authority Key Identifier`	Public key of the `manufacturer+`
(signed by)	Private key of the `manufacturer-`

Domain

The domain represents the root logical ownership of the ***network*** it is broadest sense. It is therefore an organisational perimeter, which can own network routing devices and onboarded devices.

The root domain CA is typically a self signed certificate

Practically for the purposes of the demo, the domain is collocated with the registrar

Creation

1. [@ DOMAIN] create a public private key pair
2. [@ DOMAIN] self sign with domain private key

X509 Attribute	Description/use

`Subject`	Name of the owner
	CN="name of owner"
	OU="www.manufacture.com" OR email@address

`Subject Key Identifier`	Public key of the `domain+`

`Issuer`	Name of the manufacture
	CN="name of owner"
	OU="www.manufacture.com" OR email@address

`Authority Key Identifier`	Public key of the `domain+`
(signed by)	Private key of the `domain-`

Registrar

According to the specification, the registrar

> is element of the network domain that it will belong to and that will perform its bootstrap

It is best to conceive of it as the part of the network (domain) which provides a method the device of logically joining the network (domain)

The `registrar` is signed by the `domain`

Creation Practically, for this build, the domain and registrar are co-located, so the creation process can be simplified.

1. [@ REGISTRAR] create a public private key pair
2. [@ REGISTRAR] create CSR
3. [@ DOMAIN] sign CSR with domain private key and create registrar X509 In a real deployment, where the registrar and domain are not in a 1:1 relationship, we need to consider how the registrars certificates are deployed.

This could end up looking very like the BRSKI provisioning process

The process should be in infrequent setup process

X509 Attribute	Description/use
`Subject`	C = IE, CN = registrar-tls-meta
`Subject Key Identifier`	Public key of the `registrar+`
`Issuer`	C = IE, CN = registrar-tls-ca
`Authority Key Identifier`	Public key of the `domain+`
(signed by)	Private key of the `domain-`

Radius

The radius server is an implementation detail of the router.

It is not needed or referenced in the BRSKI definition; it is useful in a practical implementation.

Many routers, use a RADIUS server to abstract the authentication process

Specially the implementation of EAP-TLS on HostAPD in the Raspberry Pi (See implementation notes)

Creation

The creation process for the RADIUS certificate is identical to the creation process for the registrar. It just refers to a different subject (the radius public key)

X509 Attribute	Description/use
`Subject`	C = IE, CN = registrar-tls-ca
`Subject Key Identifier`	Public key of the `radius+`
`Issuer`	C = IE, CN = registrar-tls-ca
`Authority Key Identifier`	Public key of the `domain+`
(signed by)	Private key of the `domain-`

Use Of Radius Certificate

The radius certificate is used to setup EAP on the hostapd (the router)

In operational mode hostapd will accept a presented EAP certificate (LDevID), if it signed by the same root (the domain), as long as that LDevID has not been revoked.

Router

The router certificate not needed or referenced in the baseline BRSKI definition. It is used in Build 5, to secure the connection between the router and the registrar. Specifically it is used to implement the continuous assurance command server.

Many routers can be connected to a single registrar.

Creation

For the purposes of the Build 5 demonstrator, we assume the router certificate to have been created and provisioned before time.

Interestingly however, a router joining a network, is not dissimilar to an IOT device joining a network.

It just happens that after joining, the router (over and above a normal IOT device), can physically extend or bridge the network.

A later demonstrator should show how live router provisioning can be performed using BRSKI provisioning

Essentially its just an iDevID

X509 Attribute	Description/use

`Subject`	Name of router(optional)
	CN="serial number"
	OU="model name"

`Subject Key Identifier`	Public key of the `router+`

`Issuer`	Name of the manufacture
	CN="Manufacturer ltd"
	OU="www.manufacture.com"

`Authority Key Identifier`	Public key of the `manufacturer+`
(signed by)	Private key of the `manufacturer-`

Use Of Router Certificate

The router certificate us used to authorise the connection between the router and the registrar.

If the router and registrar are signed by the same domain, then the connection is deemed authorised.

LDevID

The LDevID is created when a device is successfully enrolled

It is used as a cryptographic artefact to prove that the device (iDevID logically belongs to the domain)

If the device can prove it belongs to the domain this is one of the primary dimensions of authorisation, to allow a device physical operational access to the network

Creation

The LDevID creation process is formally defined in the BRSKI specification and is summarised in protocol overview section 4.

There are a few specific additions needed to get EAP working

Simplified

1. [@ DEVICE] creates mutually authenticated TLS with registrar using iDevID

2. [-> REGISTRAR] send the enrol command (optimally repeating iDevID)

3. [@ REGISTRAR] extracts iDevID (from TLS session or enrol command) and constructs a signing request for LDevID

4. [@ REGISTRAR] add the SSID identifier to the CSR

5. [@ REGISTRAR] signs the LDevID with the registrar- private key (where in turn the registrar has been signed with the domain-)

6. [-> DEVICE] send certificate back to device

7. [@ DEVICE] install LDevID

X509 Attribute	Description/use

`Subject`	Name of connection (optional)
	CN="SSID of network"
	OU="model name"

`Subject Key Identifier`	Public key of the `iDevID+`
`Issuer`	Name of the registrar
`Authority Key Identifier`	Public key of the `registrar+`
(signed by)	Private key of the `registrar-`

Use Of LDevID Certificate

LDevID is created in the EST enrol stage of the BRSKI flow LDevID is signed by registrar

LDeviD is presented by the device as its EAP-TLS certificate when attempting to connect to the operational network.

Binary Artifact API

The binary array APi defines the helper functions and structure to encode binary arrays and lists of binary arrays. The below structure and functions are used in the voucher and `BRSKI` protocol API as inputs and outputs.

Voucher Binary Array

The array API defines a structure to encode binary arrays:

```
struct BinaryArray {
  uint8_t *array;
  size_t length;
};
```

If `array == NULL` and `length == 0` the array is considered to be emtpy.

`Copy_binary_array`

Copies a binary arrays to a destination.

```
int copy_binary_array(struct BinaryArray *const dst,
                      const struct BinaryArray *src);
```

Parameters:

- `dst` - The destination binary array and
- `src` - The source binary array.

Return: `0` on success or `-1` on failure.

`Compare_binary_array`

Compare two binary arrays.

```c
int compare_binary_array(const struct BinaryArray *src,
                         const struct BinaryArray *dst);
```

Parameters:

- `src` - The source binary array and
- `dst` - The destination binary array.

Return: `1` if arrays are equal, `0` otherwise or `-1` on failure.

`Free_binary_array_content`

Frees a binary array content, i.e., frees the `array` element of the `struct BinaryArray`.

```c
void free_binary_array_content(struct BinaryArray *arr);
```

Parameters:

- `arr` - The binary array

`Free_binary_array`

Frees a binary array structure and its content.

```c
void free_binary_array(struct BinaryArray *arr);
```

Parameters:

- `arr` - The binary array

Buffer Linked List Definition

The `struct BinaryArrayList` is an array list that holds a pointer to a heap allocated array, the length and a generic flags integer.

```c
struct BinaryArrayList {
  uint8_t *arr;        /**< The array (heap allocated) */
  size_t length;       /**< The array length (heap allocated) */
  int flags;            /**< The generic array flags */
  struct dl_list list; /**< List definition */
};
```

Parameters:

- `arr` - pointer to the heap allocated array,
- `length` - the array length,
- `flags` - the generic array flags and
- `list` - the structure containg the previous and next element of the linked list.

`Init_array_list`

Initializes the array list.

```c
struct BinaryArrayList *init_array_list(void);
```

Return: Initialised array list or `NULL` on failure.

`Free_array_list`

Frees the array list and all of its elements.

```c
void free_array_list(struct BinaryArrayList *arr_list);
```

Parameters:

- `arr_list` - The array list to free.

`Push_array_list`

Pushes a heap allocated array into the list and assigns the flags.

```c
int push_buffer_list(struct BinaryArrayList *arr_list,
                     uint8_t *const arr,
                     const size_t length,
                     const int flags);
```

Parameters:

- `arr_list` - The array list structure,
- `arr` - The array pointer to insert,
- `length` - The array length and
- `flags` - The array flags.

Return: `0` on success or `-1` on failure.

BRSKI API

This `BRSKI` API implements the bootstrapping functionalities that allows a pledge to discover or being discovered by an element of the network domain (registrar) that it will belong to and that will perform its bootstrap.

The logical elements of the bootstrapping framework are described in RFC8995:

```
                                           +------------------------+
   +--------------Drop-Ship----------------| Vendor Service         |
   |                                       +------------------------+
   |                                       | M anufacturer|         |
   |                                       | A uthorized  |Ownership|
   |                                       | S igning     |Tracker  |
   |                                       | A uthority   |         |
   |                                       +--------------+---------+
   |                                                      ^
   |                                                      |  BRSKI-
   V                                                      |   MASA
+-------+     ............................................|...
|       |     .                                           |  .
|       |     .  +------------+       +-----------+       |  .
|       |     .  |            |       |           |       |  .
|Pledge |     .  |   Join     |       | Domain    <-------+  .
|       |     .  |   Proxy    |       | Registrar |          .
|       <-------->............<-------> (PKI RA)  |          .
|       |        |        BRSKI-EST   |           |          .
|       |     .  |            |       +-----+-----+          .
|IDevID |     .  +------------+             | e.g., RFC 7030 .
|       |     .           +-----------------+----------+     .
|       |     .           | Key Infrastructure         |     .
|       |     .           | (e.g., PKI CA)             |     .
+-------+     .           |                            |     .
              .           +----------------------------+     .
              .                                              .
              ................................................
                            "Domain" Components
```

The API details the functions to allows implementing the below state description for a pledge:

1. Discover a communication channel to a registrar.

2. Identify itself. This is done by presenting an X.509 IDevID credential to the discovered registrar (via the proxy) in a TLS handshake. (The registrar credentials are only provisionally accepted at this time.)
3. Request to join the discovered registrar. A unique nonce is included, ensuring that any responses can be associated with this particular bootstrapping attempt.
4. Imprint on the registrar. This requires verification of the manufacturer-service-provided (MASA) voucher. A voucher contains sufficient information for the pledge to complete authentication of a registrar.
5. Enroll. After imprint, an authenticated TLS (HTTPS) connection exists between the pledge and registrar. EST [RFC7030] can then be used to obtain a domain certificate from a registrar.

BRSKI Core API

`Sign_pledge_voucher_request`

Signs a pledge voucher request using CMS with a private key (type detected automatically) and output as binary array (`DER` format).

```
__must_free_binary_array struct BinaryArray *
sign_pledge_voucher_request(const struct tm *created_on,
                            const char *serial_number,
                            const struct BinaryArray *nonce,
                            const struct BinaryArray
*registrar_tls_cert,
                            const struct BinaryArray
*pledge_sign_cert,
                            const struct BinaryArray *pledge_sign_key,
                            const struct BinaryArrayList
*additional_pledge_certs);
```

Parameters:

- `created_on` - Time when the pledge is created,
- `serial_number` - The serial number string of the pledge,
- `nonce` - Random/pseudo-random nonce (`NULL` for empty),
- `registrar_tls_cert` - The first certificate in the TLS server "certificate_list" sequence presented by the registrar to the pledge (`DER` format),
- `pledge_sign_cert` - The certificate buffer (`DER` format) corresponding to the signig private key,
- `pledge_sign_key` - The private key buffer (`DER` format) for signing the pledge-voucher request and
- `additional_pledge_certs` - The list of additional pledge certificates (`DER` format) to append to CMS (`NULL` for empty).

Return: The signed pledge-voucher CMS structure as bianry array (`DER` format) or `NULL` on failure.

`Sign_voucher_request`

Signs a voucher request using CMS with a private key (type detected automatically) and output as binary array (`DER` format).

```
__must_free_binary_array struct BinaryArray *
sign_voucher_request(const struct BinaryArray
*pledge_voucher_request_cms,
                     const struct tm *created_on, const char
*serial_number,
                     const struct BinaryArray *idevid_issuer,
                     const struct BinaryArray *registrar_tls_cert,
                     const struct BinaryArray *registrar_sign_cert,
                     const struct BinaryArray *registrar_sign_key,
                     const struct BinaryArrayList
*pledge_verify_certs,
                     const struct BinaryArrayList
*pledge_verify_store,
                     const struct BinaryArrayList
*additional_registrar_certs);
```

Parameters:

- `pledge_voucher_request_cms` - The signed pledge-voucher request CMS structure as binary array (`DER` format),
- `created_on` - Time when the voucher request is created,
- `serial_number` - The serial number string from the idevid certificate,
- `idevid_issuer` - The idevid issuer from the idevid certificate,
- `registrar_tls_cert` - The first certificate in the TLS server "certificate_list" sequence presented by the registrar to the pledge (`DER` format),
- `registrar_sign_cert` - The certificate buffer (`DER` format) corresponding to the signing private key,
- `registrar_sign_key` - The private key buffer (`DER` format) for signing the voucher request,
- `pledge_verify_certs` - The list of intermediate certificate buffers (`DER` format) to verify the pledge-voucher request (`NULL` for empty),
- `pledge_verify_store` - The list of trusted certificate buffers (`DER` format) to verify the pledge-voucher request (`NULL` for empty). The lists' flags are described in verify_cms_voucher function and
- `additional_registrar_certs` - The list of additional registrar certificate buffers (`DER` format) to append to CMS (`NULL` for empty).

Return: The signed CMS structure as binary array (`DER` format) or `NULL` on failure.

`Voucher_req_fn`

Callback function definition to find a pledge serial number in a user defined database and output a pinned domain certificate (DER format).

```
typedef int (*voucher_req_fn)(
    const char *serial_number,
    const struct BinaryArrayList *additional_registrar_certs, const
void *user_ctx,
    struct BinaryArray *pinned_domain_cert);
```

Parameters:

- `serial_number` - The serial number string from the idevid certificate,
- `additional_registrar_certs` - The list of additional registrar certificates (`DER` format) appended to the voucher request CMS,
- `user_ctx` - The callback function user context and
- `voucher_req_fn` - The output pinned domain certificate (`DER` format) for the pledge.

Return: `0` on success or `-1` on failure.

`Sign_masa_pledge_voucher`

Signs a MASA voucher request using CMS with a private key (type detected automatically) and output as binary array (DER format).

```
__must_free_binary_array struct BinaryArray
*sign_masa_pledge_voucher(const struct BinaryArray
*voucher_request_cms,
                         const struct tm *expires_on, const
voucher_req_fn cb,
                         void *user_ctx,
                         const struct BinaryArray *masa_sign_cert,
                         const struct BinaryArray *masa_sign_key,
                         const struct BinaryArrayList
*registrar_verify_certs,
                         const struct BinaryArrayList
*registrar_verify_store,
                         const struct BinaryArrayList
*pledge_verify_certs,
                         const struct BinaryArrayList
*pledge_verify_store,
                             const struct BinaryArrayList
*additional_masa_certs);
```

Parameters:

- `voucher_request_cms` - The signed pledge voucher request CMS structure as binary array (`DER` format),
- `expires_on` - Time when the new voucher will expire,
- `voucher_req_fn` - The callback function to output pinned domain certificate (`DER` format),
- `user_ctx` - The callback function user context (`NULL` for empty),
- `masa_sign_cert` - The certificate buffer (`DER` format) corresponding to the signing private key,
- `masa_sign_key` - The private key buffer (`DER` format) for signing the MASA voucher request,
- `registrar_verify_certs` - The list of intermediate certificate buffers (`DER` format) to verify the voucher request from registrar (`NULL` for empty),
- `registrar_verify_store` - The list of trusted certificate buffers (`DER` format) to verify the voucher request from registrar (`NULL` for empty). The lists' flags are described in verify_cms_voucher function,
- `pledge_verify_certs` - The list of intermediate certificate buffers (`DER` format) to verify the pledge-voucher request (`NULL` for empty),
- `pledge_verify_store` - The list of trusted certificate buffers (`DER` format) to verify the pledge-voucher request (`NULL` for empty). The lists' flags are described in verify_cms_voucher function and
- `additional_masa_certs` - The list of additional MASA certificate buffers (`DER` format) to append to CMS (`NULL` for empty).

Return: The signed CMS structure as binary array (`DER` format) or `NULL` on failure.

`Verify_masa_pledge_voucher`

Verifies a MASA pledge voucher and outputs a pinned domain certificate (`DER` format) and the CMS appended list of certificates.

```
int verify_masa_pledge_voucher(
    const struct BinaryArray *masa_pledge_voucher_cms, const char
*serial_number,
    const struct BinaryArray *nonce,
    const struct BinaryArray *registrar_tls_cert,
    const struct BinaryArrayList *domain_store,
    const struct BinaryArrayList *pledge_verify_certs,
    const struct BinaryArrayList *pledge_verify_store,
    struct BinaryArrayList **pledge_out_certs,
    struct BinaryArray *const pinned_domain_cert);
```

Parameters:

- `masa_pledge_voucher_cms` - The signed MASA pledge voucher CMS structure as binary array (`DER` format),
- `serial_number` - The serial number string from the idevid certificate,

- `nonce` - Random/pseudo-random nonce from the pledge voucher request (`NULL` for empty),
- `registrar_tls_cert` - The first certificate in the TLS server "certificate_list" sequence presented by the registrar to the pledge (`DER` format),
- `domain_store` - The list of trusted certificate buffers (`DER` format) to verify the pinned domain certificate (`NULL` for empty). The lists' flags are described in verify_cms_voucher function,
- `pledge_verify_certs` - The list of intermediate certificate buffers (`DER` format) to verify the masa pledge voucher (`NULL` for empty),
- `pledge_verify_store` - The list of trusted certificate buffers (`DER` format) to verify the masa pledge voucher (`NULL` for empty). The lists' flags are described in verify_cms_voucher function,
- `pledge_out_certs` - The list of output certificate buffers (`NULL` for empty) from the MASA pledge CMS structure and
- `pinned_domain_cert` - The output pinned domain certificate buffer (`DER` format)

Return: `0` on success or `-1` on failure.

BRSKI Usage

Voucher Library Usage

Example CMakeLists.Txt File

Below is an example `cmake` config file to import the voucher library into a `cmake` project and use it with OpenSSL. The path for the static voucher library file `libvoucher.a` is set by the `LIBVOUCHER_LIBRARY` variable and the voucher library include path is set by `LIBVOUCHER_INCLUDE_PATH` variable.

```
cmake_minimum_required(VERSION 3.1...3.26)

project(
  LibTest
  VERSION 1.0
  LANGUAGES C)

find_package(OpenSSL 3 MODULE REQUIRED COMPONENTS Crypto SSL)
message("Found OpenSSL ${OPENSSL_VERSION} crypto library")
add_library(OpenSSL3::Crypto ALIAS OpenSSL::Crypto)
add_library(OpenSSL3::SSL ALIAS OpenSSL::SSL)
set(LIBOPENSSL3_INCLUDE_PATH "${OPENSSL_INCLUDE_DIR}")

set(LIBVOUCHER_INCLUDE_PATH "${CMAKE_SOURCE_DIR}/include/voucher")
set(LIBVOUCHER_LIB_PATH "${CMAKE_SOURCE_DIR}/lib")
set(LIBVOUCHER_LIBRARY "${LIBVOUCHER_LIB_PATH}/libvoucher.a")

add_library(Voucher::Voucher STATIC IMPORTED)
set_target_properties(Voucher::Voucher PROPERTIES
  IMPORTED_LOCATION "${LIBVOUCHER_LIBRARY}"
  INTERFACE_INCLUDE_DIRECTORIES "${LIBVOUCHER_INCLUDE_PATH}"
)

add_executable(libtest libtest.c)
target_link_libraries(libtest PRIVATE Voucher::Voucher
OpenSSL::Crypto)
```

BRSKI Tool Usage

The BRSKI `generate_test_certs` test creates a `test-config.ini` file (located at `${CMAKE_BINARY_DIR}/tests/brski/test-config.ini`), which has some pregenerated example certificates for running the MASA, registrar, and pledge on localhost.

Exporting A Pledge Voucher Request

To export a pledge voucher request to a `CMS` SMIME file `out.cms` use the command `epvr` as following:

```
$ brski -c config.ini -o out.cms epvr
```

where the example `config.ini` file is defined as follows:

```
[pledge]
createdOn = "1973-11-29T21:33:09Z"
serialNumber = "12345"
nonce = "some-nonce-value-in-base64"
idevidKeyPath = ""
idevidCertPath = ""
idevidCACertPath = ""
cmsSignKeyPath = "/absolute_path_to/pledge-cms.key"
cmsSignCertPath = "/absolute_path_to/pledge-cms.crt"
cmsAdditionalCertPath = ""
cmsVerifyCertPath = ""
cmsVerifyStorePath = ""

[registrar]
bindAddress = ""
port = 0
tlsKeyPath = "/absolute_path_to/registrar-tls.key"
tlsCertPath = "/absolute_path_to/registrar-tls.crt"
tlsCACertPath = ""
cmsSignCertPath = ""
cmsSignKeyPath = ""
cmsAdditionalCertPath = ""
cmsVerifyCertPath = ""
cmsVerifyStorePath = ""

[masa]
```

```
bindAddress = ""
port = 0
expiresOn = ""
ldevidCAKeyPath = ""
ldevidCACertPath = ""
tlsKeyPath = ""
tlsCertPath = ""
tlsCACertPath = ""
cmsSignKeyPath = ""
cmsSignCertPath = ""
cmsAdditionalCertPath = ""
cmsVerifyCertPath = ""
cmsVerifyStorePath = ""
```

Sending A Pledge-Voucher Request To The Registrar

To send a pledge-voucher request to a registrar use the command `preq` as following:

```
$ brski -c config.ini preq
```

where the example `config.ini` file is defined as follows:

```
[pledge]
createdOn = "1973-11-29T21:33:09Z"
serialNumber = "idev-serial12345"
nonce = "some-nonce-value-in-base64"
idevidKeyPath = "/absolute_path_to/idevid.key"
idevidCertPath = "/absolute_path_to/idevid.crt"
idevidCACertPath = "/absolute_path_to/idevid-ca.crt"
cmsSignKeyPath = "/absolute_path_to/pledge-cms.key"
cmsSignCertPath = "/absolute_path_to/pledge-cms.crt"
cmsAdditionalCertPath = ""
cmsVerifyCertPath = ""
cmsVerifyStorePath = ""

[registrar]
bindAddress = "https://registrar-address.com"
port = 12345
tlsKeyPath = ""
```

```
tlsCertPath = "/absolute_path_to/registrar-tls.crt"
tlsCACertPath = "/absolute_path_to/registrar-tls-ca.crt"
cmsSignKeyPath = ""
cmsSignCertPath = ""
cmsAdditionalCertPath = ""
cmsVerifyCertPath = ""
cmsVerifyStorePath = ""

[masa]
bindAddress = ""
port = 0
expiresOn = ""
ldevidCAKeyPath = ""
ldevidCACertPath = ""
tlsKeyPath = ""
tlsCertPath = ""
tlsCACertPath = ""
cmsSignKeyPath = ""
cmsSignCertPath = ""
cmsAdditionalCertPath = ""
cmsVerifyCertPath = ""
cmsVerifyStorePath = ""
```

Starting The Registrar

To start the registrar server on port `12345` use the command `registrar` as following:

```
$ brski -c config.ini registrar
```

where the example `config.ini` file is defined as follows:

```
[pledge]
createdOn = ""
serialNumber = ""
nonce = ""
idevidKeyPath = ""
idevidCertPath = "/absolute_path_to/idevid.crt"
idevidCACertPath = "/absolute_path_to/idevid-ca.crt"
cmsSignKeyPath = ""
```

```
cmsSignCertPath = ""
cmsAdditionalCertPath = ""
cmsVerifyCertPath = ""
cmsVerifyStorePath = ""

[registrar]
bindAddress = "127.0.0.1"
port = 12345
tlsKeyPath = ""
tlsCertPath = "/absolute_path_to/registrar-tls.crt"
tlsCACertPath = "/absolute_path_to/registrar-tls-ca.crt"
cmsSignKeyPath = "/absolute_path_to/registrar-cms.key"
cmsSignCertPath = "/absolute_path_to/registrar-cms.crt"
cmsAdditionalCertPath = ""
cmsVerifyCertPath = ""
cmsVerifyStorePath = ""

[masa]
bindAddress = "https://masa-address.com"
port = 12346
expiresOn = ""
ldevidCAKeyPath = ""
ldevidCACertPath = "/absolute_path_to/ldevid-ca.crt"
tlsKeyPath = ""
tlsCertPath = "/absolute_path_to/masa-tls.crt"
tlsCACertPath = "/absolute_path_to/masa-tls-ca.crt"
cmsSignKeyPath = ""
cmsSignCertPath = ""
cmsAdditionalCertPath = ""
cmsVerifyCertPath = ""
cmsVerifyStorePath = ""
```

Starting The MASA

To start the MASA server on port `12346` use the command `masa` as following:

```
$ brski -c config.ini masa
```

where the example `config.ini` file is defined as follows:

```
[pledge]
createdOn = ""
serialNumber = ""
nonce = ""
idevidKeyPath = ""
idevidCertPath = "/absolute_path_to/idevid.crt"
idevidCACertPath = "/absolute_path_to/idevid-ca.crt"
cmsSignKeyPath = ""
cmsSignCertPath = ""
cmsAdditionalCertPath = ""
cmsVerifyCertPath = ""
cmsVerifyStorePath = ""

[registrar]
bindAddress = ""
port = 12345
tlsKeyPath = ""
tlsCertPath = "/absolute_path_to/registrar-tls.crt"
tlsCACertPath = "/absolute_path_to/registrar-tls-ca.crt"
cmsSignKeyPath = ""
cmsSignCertPath = ""
cmsAdditionalCertPath = ""
cmsVerifyCertPath = ""
cmsVerifyStorePath = ""

[masa]
bindAddress = "127.0.0.1"
port = 12346
expiresOn = "1973-11-29T21:33:09Z"
ldevidCAKeyPath = "/absolute_path_to/ldevid-ca.key"
ldevidCACertPath = "/absolute_path_to/ldevid-ca.crt"
tlsKeyPath = "/absolute_path_to/masa-tls.key"
tlsCertPath = "/absolute_path_to/masa-tls.crt"
tlsCACertPath = "/absolute_path_to/masa-tls-ca.crt"
cmsSignKeyPath = "/absolute_path_to/masa-cms.key"
cmsSignCertPath = "/absolute_path_to/masa-cms.crt"
cmsAdditionalCertPath = ""
cmsVerifyCertPath = ""
cmsVerifyStorePath = ""
```

For detailed example of `config.ini` files and certificates pleasce check the `test` folder.

Voucher Artifact API

The voucher artifact is a JSON RFC8259 document that conforms with a data model described by YANG RFC7950, is encoded using the rules defined in RFC8259, and is signed using (by default) a CMS structure RFC5652.

```
module: ietf-voucher
   yang-data voucher-artifact:
      +---- voucher
         +---- created-on                       yang:date-and-time
         +---- expires-on?                      yang:date-and-time
         +---- assertion                        enumeration
         +---- serial-number                    string
         +---- idevid-issuer?                   binary
         +---- pinned-domain-cert               binary
         +---- domain-cert-revocation-checks?   boolean
         +---- nonce?                           binary
         +---- last-renewal-date?               yang:date-and-time
         +-- prior-signed-voucher-request?      binary
         +-- proximity-registrar-cert?          binary
```

Voucher Attributes

The voucher artifact attributes are define by the enum below:

```
enum VoucherAttributes {
  ATTR_CREATED_ON = 0,
  ATTR_EXPIRES_ON,
  ATTR_ASSERTION,
  ATTR_SERIAL_NUMBER,
  ATTR_IDEVID_ISSUER,
  ATTR_PINNED_DOMAIN_CERT,
  ATTR_DOMAIN_CERT_REVOCATION_CHECKS,
  ATTR_NONCE,
  ATTR_LAST_RENEWAL_DATE,
  ATTR_PRIOR_SIGNED_VOUCHER_REQUEST,
  ATTR_PROXIMITY_REGISTRAR_CERT
};
```

Voucher Creation/Manipulation API

`Init_voucher`

Initialises an empty voucher structure.

```
__must_free_voucher struct Voucher *init_voucher(void);
```

Return: Pointer to an allocated voucher or NULL on failure.

`Free_voucher`

Frees an allocated voucher structure.

```
void free_voucher(struct Voucher *voucher);
```

Parameters:

- `voucher` - The allocated voucher structure.

`Set_attr_bool_voucher`

Sets the value for a voucher bool attribute.

```
int set_attr_bool_voucher(struct Voucher *voucher,
                          const enum VoucherAttributes attr,
                          const bool value);
```

Parameters:

- `voucher` - The allocated voucher structure,
- `attr` - The voucher attribute corresponding to the `bool` value and

- `value` - The `bool` attribute value.

Return: `0` on success or `-1` on failure.

`Set_attr_time_voucher`

Sets the value for a voucher time attribute.

```
int set_attr_time_voucher(struct Voucher *voucher,
                          const enum VoucherAttributes attr,
                          const struct tm *value);
```

Parameters:

- `voucher` - The allocated voucher structure,
- `attr` - The voucher attribute corresponding to the `struct tm` value and
- `value` - The `struct tm` attribute value.

Return: `0` on success or `-1` on failure.

`Set_attr_enum_voucher`

Sets the value for a voucher enum attribute.

```
int set_attr_enum_voucher(struct Voucher *voucher,
                          const enum VoucherAttributes attr,
                          const int value);
```

Parameters:

- `voucher` - The allocated voucher structure,
- `attr` - The enum voucher attribute and
- `value` - The enum attribute value.

Return: `0` on success or `-1` on failure.

The enum attribute API sets the value for the assertion attribute with one of the following values as described in RFC8995:

```
enum VoucherAssertions {
  VOUCHER_ASSERTION_NONE = 0,
  VOUCHER_ASSERTION_VERIFIED = 1,
  VOUCHER_ASSERTION_LOGGED = 2,
  VOUCHER_ASSERTION_PROXIMITY = 3
};
```

`Set_attr_str_voucher`

Sets the value for a voucher string attribute.

```
int set_attr_str_voucher(struct Voucher *voucher,
                         const enum VoucherAttributes attr,
                         const char *value);
```

Parameters:

- `voucher` - The allocated voucher structure,
- `attr` - The string voucher attribute name and
- `value` - The string attribute value.

Return: `0` on success or `-1` on failure.

`Set_attr_array_voucher`

Sets the value for a voucher array attribute.

```
int set_attr_array_voucher(struct Voucher *voucher,
                           const enum VoucherAttributes attr,
                           const struct BinaryArray *value);
```

Parameters:

- `voucher` - The allocated voucher structure,
- `attr` - The array voucher attribute name and
- `value` - The array attribute value.

Return: `0` on success or `-1` on failure.

`Set_attr_voucher`

Sets the value for a voucher attribute.

```
int set_attr_voucher(struct Voucher *voucher,
                     const enum VoucherAttributes attr,
                     ...);
```

Parameters:

- `voucher` - The allocated voucher structure,
- `attr` - The array voucher attribute name and
- `__VA_ARGS__` - The variable list of attribute values:
  - `ATTR_CREATED_ON` => `struct tm *`
  - `ATTR_EXPIRES_ON` => `struct tm *`
  - `ATTR_LAST_RENEWAL_DATE` => `struct tm *`
  - `ATTR_ASSERTION` => `enum VoucherAssertions`
  - `ATTR_SERIAL_NUMBER` => `char *`
  - `ATTR_IDEVID_ISSUER` => `struct BinaryArray *`
  - `ATTR_PINNED_DOMAIN_CERT` => `struct BinaryArray *`
  - `ATTR_NONCE` => `struct BinaryArray *`
  - `ATTR_PRIOR_SIGNED_VOUCHER_REQUEST` => `struct BinaryArray *`
  - `ATTR_PROXIMITY_REGISTRAR_CERT` => `struct BinaryArray *`
  - `ATTR_DOMAIN_CERT_REVOCATION_CHECKS` => `bool`

Return: `0` on success or `-1` on failure.

`Clear_attr_voucher`

Clears a voucher attribute.

```
int clear_attr_voucher(struct Voucher *voucher,
                       const enum VoucherAttributes attr);
```

Parameters:

- `voucher` - The allocated voucher structure and
- `attr` - The attribute name

Return: `0` on success or `-1` on failure.

`Is_attr_voucher_nonempty`

Checks if a voucher attribute is non empty.

```
bool is_attr_voucher_nonempty(const struct Voucher *voucher,
                              const enum VoucherAttributes attr);
```

Parameters:

- `voucher` - The allocated voucher structure and
- `attr` - The attribute name.

Return: `true` if non empty or `false` otherwise.

`Get_attr_bool_voucher`

Gets the pointer to the value for a voucher bool attribute.

```
const bool *get_attr_bool_voucher(const struct Voucher *voucher,
                                  const enum VoucherAttributes attr);
```

Parameters:

- `voucher` - The allocated voucher structure and
- `attr` - The bool voucher attribute.

Return: Pointer to the `bool` value or `NULL` on failure.

`Get_attr_time_voucher`

Gets the pointer to the value for a voucher time attribute.

```
const struct tm *get_attr_time_voucher(struct Voucher *voucher,
                                       const enum VoucherAttributes
attr);
```

Parameters:

- `voucher` - The allocated voucher structure and
- `attr` - The time voucher attribute.

Return: Pointer to the time value or `NULL` on failure.

`Get_attr_enum_voucher`

Gets the pointer to the value for a voucher enum attribute.

```
const int *get_attr_enum_voucher(struct Voucher *voucher,
                                 const enum VoucherAttributes attr);
```

Parameters:

- `voucher` - The allocated voucher structure and
- `attr` - The enum voucher attribute.

Return: Pointer to the enum value or `NULL` on failure.

`Get_attr_str_voucher`

Gets the pointer to the value for a voucher string attribute.

```
const char *const *get_attr_str_voucher(struct Voucher *voucher,
                                        const enum VoucherAttributes
attr);
```

Parameters:

- `voucher` - The allocated voucher structure and
- `attr` - The string voucher attribute name.

Return: Pointer to the string value or `NULL` on failure.

Example:

```
const char *const *serial_number = get_attr_str_voucher(voucher,
ATTR_SERIAL_NUMBER);
if (strcmp(*serial_number, "12345")) {}
```

`Get_attr_array_voucher`

Gets the pointer to the value for a voucher array attribute.

```
const struct BinaryArray * get_attr_array_voucher(struct Voucher
*voucher,
                                                         const enum
VoucherAttributes attr);
```

Parameters:

- `voucher` - The allocated voucher structure and
- `attr` - The array voucher attribute name.

Return: Pointer to the array value or `NULL` on failure.

Voucher Serialization And Deserialization API

`Serialize_voucher`

Serializes a voucher to a string.

```
__must_free char *serialize_voucher(const struct Voucher *voucher);
```

Parameters:

- `voucher` - The allocated voucher structure.

Return: Serialized voucher to string or `NULL` on failure.

Example:

```
struct Voucher *voucher = init_voucher();

set_attr_enum_voucher(voucher, ATTR_ASSERTION,
VOUCHER_ASSERTION_PROXIMITY);

char *serialized = serialize_voucher(voucher);
```

```c
/* ... */

free(serialized);
free_voucher(voucher);
```

`Deserialize_voucher`

Deserializes a json string buffer to a voucher structure.

```c
struct Voucher *deserialize_voucher(const uint8_t *json, const size_t
length);
```

Paramaters:

- `json` - The json buffer and
- `length` - The json buffer length.

Return: Voucher structure or `NULL` on failure.

Example:

```c
struct Voucher *voucher = deserialize_voucher(json, json_length);

/* ... */

free_voucher(voucher);
```

Voucher CMS Signing And Verification API

`Sign_eccms_voucher`

Signs a voucher using CMS with an Elliptic Curve private key and output to a binary buffer (`DER` format).

```
__must_free_binary_array struct BinaryArray *sign_eccms_voucher(struct
Voucher *voucher,
                                     const struct BinaryArray *cert,
                                     const struct BinaryArray *key,
                                     const struct BinaryArrayList
*certs);
```

Parameters:

- `voucher` - The allocated voucher structure,
- `cert` - The certificate buffer (`DER` format) correspoding to the private key,
- `key` - The Elliptic Curve private key buffer (`DER` format) of the certificate and
- `certs` - The `struct BinaryArrayList` of additional certificate buffers (`DER` format) to be included in the CMS (`NULL` if none).

Return: The signed CMS structure in binary (`DER` format) or `NULL` on failure.

`Sign_rsacms_voucher`

Signs a voucher using CMS with a RSA private key and output to a binary buffer (`DER` format).

```
__must_free_binary_array struct BinaryArray
*sign_rsacms_voucher(struct Voucher *voucher,
                                      const struct BinaryArray *cert,
                                      const struct BinaryArray *key,
                                      const struct BinaryArrayList
*certs);
```

Parameters:

- `voucher` - The allocated voucher structure,
- `cert` - The certificate buffer (`DER` format) correspoding to the private key,
- `key` - The RSA private key buffer (`DER` format) of the certificate and
- `certs` - The `struct BinaryArrayList` of additional certificate buffers (`DER` format) to be included in the CMS (`NULL` if none)

Return: The signed CMS structure in binary (`DER` format) or `NULL` on failure.

`Sign_cms_voucher`

Signs a voucher using CMS with a private key (detected automatically) and output as binary array (`DER` format).

```
__must_free_binary_array struct BinaryArray *sign_cms_voucher(struct
Voucher *voucher,
                                   const struct BinaryArray *cert,
                                   const struct BinaryArray *key,
                                   const struct BinaryArrayList
*certs);
```

Parameters:

- `voucher` - The allocated voucher structure,
- `cert` - The certificate buffer (`DER` format) correspoding to the private key,
- `key` - The private key buffer (`DER` format) of the certificate and
- `certs` - The list of additional certificate buffers (`DER` format) to be included in the CMS (`NULL` if none)

Return: The signed CMS structure as binary array (`DER` format) or `NULL` on failure.

`Verify_cms_voucher`

Verifies a CMS binary buffer and extracts the voucher structure, and the list of included certificates.

```
__must_free_voucher struct Voucher *verify_cms_voucher(const struct
BinaryArray *cms,
                                   const struct BinaryArrayList
*certs,
                                   const struct BinaryArrayList
*store,
                                   struct BinaryArrayList
**out_certs);
```

Parameters:

- `cms` - The CMS binary buffer string (`DER` format),
- `certs` - The list of additional certificate buffers (`DER` format),
- `store` - The list of trusted certificate for store (`DER` format). The list's flags is encoded with the following enum:

  ```
  enum CRYPTO_CERTIFICATE_TYPE {
    CRYPTO_CERTIFICATE_VALID = 0,
    CRYPTO_CERTIFICATE_CRL,
  };
  ```

  where `CRYPTO_CERTIFICATE_VALID` denotes a standard certificate buffer and `CRYPTO_CERTIFICATE_CRL` denotes a certificate revocation type buffer, and
- `out_certs` - The output list of certificates (`NULL` for empty) from the CMS structure.

Return: The verified voucher structrure or `NULL` on failure.

Example:

```
struct BinaryArrayList *out_certs = NULL;
struct Voucher *voucher = verify_cms_voucher(cms, certs, store,
&out_certs);
struct BinaryArrayList *cert = NULL;

dl_list_for_each(el, &out_certs->list, struct BinaryArrayList, list) {
  uint8_t cert_array = cert->buf;
  uint8_t cert_length = cert->length;
  /* ... */
}

/* ... */

free_voucher(voucher);
free_buffer_list(out_certs);
```

RFC 8995

Bootstrapping Remote Secure Key Infrastructure (BRSKI)

Abstract

This document specifies automated bootstrapping of an Autonomic Control Plane. To do this, a Secure Key Infrastructure is bootstrapped. This is done using manufacturer-installed X.509 certificates, in combination with a manufacturer's authorizing service, both online and offline. We call this process the Bootstrapping Remote Secure Key Infrastructure (BRSKI) protocol. Bootstrapping a new device can occur when using a routable address and a cloud service, only link-local connectivity, or limited/disconnected networks. Support for deployment models with less stringent security requirements is included. Bootstrapping is complete when the cryptographic identity of the new key infrastructure is successfully deployed to the device. The established secure connection can be used to deploy a locally issued certificate to the device as well.

Status Of This Memo

This is an Internet Standards Track document.

This document is a product of the Internet Engineering Task Force (IETF). It represents the consensus of the IETF community. It has received public review and has been approved for publication by the Internet Engineering Steering Group (IESG). Further information on Internet Standards is available in Section 2 of RFC 7841.

Information about the current status of this document, any errata, and how to provide feedback on it may be obtained at https://www.rfc-editor.org/info/rfc8995.

Copyright Notice

▲

Table Of Contents

- 1. Introduction
    - 1.1. Prior Bootstrapping Approaches
    - 1.2. Terminology
    - 1.3. Scope of Solution
        - 1.3.1. Support Environment
        - 1.3.2. Constrained Environments
        - 1.3.3. Network Access Controls
        - 1.3.4. Bootstrapping is Not Booting
    - 1.4. Leveraging the New Key Infrastructure / Next Steps
    - 1.5. Requirements for Autonomic Networking Infrastructure (ANI) Devices
- 2. Architectural Overview
    - 2.1. Behavior of a Pledge
    - 2.2. Secure Imprinting Using Vouchers
    - 2.3. Initial Device Identifier
        - 2.3.1. Identification of the Pledge
        - 2.3.2. MASA URI Extension
    - 2.4. Protocol Flow
    - 2.5. Architectural Components
        - 2.5.1. Pledge
        - 2.5.2. Join Proxy
        - 2.5.3. Domain Registrar
        - 2.5.4. Manufacturer Service
        - 2.5.5. Public Key Infrastructure (PKI)
    - 2.6. Certificate Time Validation
        - 2.6.1. Lack of Real-Time Clock
        - 2.6.2. Infinite Lifetime of IDevID
    - 2.7. Cloud Registrar
    - 2.8. Determining the MASA to Contact
- 3. Voucher-Request Artifact
    - 3.1. Nonceless Voucher-Requests
    - 3.2. Tree Diagram
    - 3.3. Examples
    - 3.4. YANG Module
- 4. Proxying Details (Pledge -- Proxy -- Registrar)
    - 4.1. Pledge Discovery of Proxy
        - 4.1.1. Proxy GRASP Announcements
    - 4.2. CoAP Connection to Registrar
    - 4.3. Proxy Discovery and Communication of Registrar
- 5. Protocol Details (Pledge -- Registrar -- MASA)

    - 5.1. BRSKI-EST TLS Establishment Details
    - 5.2. Pledge Requests Voucher from the Registrar
    - 5.3. Registrar Authorization of Pledge
    - 5.4. BRSKI-MASA TLS Establishment Details
        - 5.4.1. MASA Authentication of Customer Registrar
    - 5.5. Registrar Requests Voucher from MASA
        - 5.5.1. MASA Renewal of Expired Vouchers
        - 5.5.2. MASA Pinning of Registrar
        - 5.5.3. MASA Check of the Voucher-Request Signature
        - 5.5.4. MASA Verification of the Domain Registrar
        - 5.5.5. MASA Verification of the Pledge 'prior-signed-voucher-request'
        - 5.5.6. MASA Nonce Handling
    - 5.6. MASA and Registrar Voucher Response
        - 5.6.1. Pledge Voucher Verification
        - 5.6.2. Pledge Authentication of Provisional TLS Connection
    - 5.7. Pledge BRSKI Status Telemetry
    - 5.8. Registrar Audit-Log Request
        - 5.8.1. MASA Audit-Log Response
        - 5.8.2. Calculation of domainID
        - 5.8.3. Registrar Audit-Log Verification
    - 5.9. EST Integration for PKI Bootstrapping
        - 5.9.1. EST Distribution of CA Certificates
        - 5.9.2. EST CSR Attributes
        - 5.9.3. EST Client Certificate Request
        - 5.9.4. Enrollment Status Telemetry
        - 5.9.5. Multiple Certificates
        - 5.9.6. EST over CoAP
- 6. Clarification of Transfer-Encoding
- 7. Reduced Security Operational Modes
    - 7.1. Trust Model
    - 7.2. Pledge Security Reductions
    - 7.3. Registrar Security Reductions
    - 7.4. MASA Security Reductions
        - 7.4.1. Issuing Nonceless Vouchers
        - 7.4.2. Trusting Owners on First Use
        - 7.4.3. Updating or Extending Voucher Trust Anchors
- 8. IANA Considerations
    - 8.1. The IETF XML Registry
    - 8.2. YANG Module Names Registry
    - 8.3. BRSKI Well-Known Considerations
        - 8.3.1. BRSKI .well-known Registration
        - 8.3.2. BRSKI .well-known Registry
    - 8.4. PKIX Registry
    - 8.5. Pledge BRSKI Status Telemetry
    - 8.6. DNS Service Names
    - 8.7. GRASP Objective Names
- 9. Applicability to the Autonomic Control Plane (ACP)
    - 9.1. Operational Requirements

        - 9.1.1. MASA Operational Requirements
        - 9.1.2. Domain Owner Operational Requirements
        - 9.1.3. Device Operational Requirements
- 10. Privacy Considerations
    - 10.1. MASA Audit-Log
    - 10.2. What BRSKI-EST Reveals
    - 10.3. What BRSKI-MASA Reveals to the Manufacturer
    - 10.4. Manufacturers and Used or Stolen Equipment
    - 10.5. Manufacturers and Grey Market Equipment
    - 10.6. Some Mitigations for Meddling by Manufacturers
    - 10.7. Death of a Manufacturer
- 11. Security Considerations
    - 11.1. Denial of Service (DoS) against MASA
    - 11.2. DomainID Must Be Resistant to Second-Preimage Attacks
    - 11.3. Availability of Good Random Numbers
    - 11.4. Freshness in Voucher-Requests
    - 11.5. Trusting Manufacturers
    - 11.6. Manufacturer Maintenance of Trust Anchors
        - 11.6.1. Compromise of Manufacturer IDevID Signing Keys
        - 11.6.2. Compromise of MASA Signing Keys
        - 11.6.3. Compromise of MASA Web Service
    - 11.7. YANG Module Security Considerations
- 12. References
    - 12.1. Normative References
    - 12.2. Informative References
- Appendix A. IPv4 and Non-ANI Operations
    - A.1. IPv4 Link-Local Addresses
    - A.2. Use of DHCPv4
- Appendix B. mDNS / DNS-SD Proxy Discovery Options
- Appendix C. Example Vouchers
    - C.1. Keys Involved
        - C.1.1. Manufacturer Certification Authority for IDevID Signatures
        - C.1.2. MASA Key Pair for Voucher Signatures
        - C.1.3. Registrar Certification Authority
        - C.1.4. Registrar Key Pair
        - C.1.5. Pledge Key Pair
    - C.2. Example Process
        - C.2.1. Pledge to Registrar
        - C.2.2. Registrar to MASA
        - C.2.3. MASA to Registrar
        - Acknowledgements
- Authors' Addresses

1. Introduction

The Bootstrapping Remote Secure Key Infrastructure (BRSKI) protocol provides a solution for secure zero-touch (automated) bootstrap of new (unconfigured) devices that are called "pledges" in this document. Pledges have an Initial Device Identifier (IDevID) installed in them at the factory.

"BRSKI", pronounced like "brewski", is a colloquial term for beer in Canada and parts of the Midwestern United States [brewski].

This document primarily provides for the needs of the ISP and enterprise-focused Autonomic Networking Integrated Model and Approach (ANIMA) Autonomic Control Plane (ACP) [RFC8994]. This bootstrap process satisfies the requirement of making all operations secure by default per Section 3.3 of [RFC7575]. Other users of the BRSKI protocol will need to provide separate applicability statements that include privacy and security considerations appropriate to that deployment. Section 9 explains the detailed applicability for this ACP usage.

The BRSKI protocol requires a significant amount of communication between manufacturer and owner: in its default modes, it provides a cryptographic transfer of control to the initial owner. In its strongest modes, it leverages sales channel information to identify the owner in advance. Resale of devices is possible, provided that the manufacturer is willing to authorize the transfer. Mechanisms to enable transfers of ownership without manufacturer authorization are not included in this version of the protocol, but it could be designed into future versions.

This document describes how a pledge discovers (or are discovered by) an element of the network domain that it will belong to and that will perform its bootstrap. This element (device) is called the "registrar". Before any other operation, the pledge and registrar need to establish mutual trust:

1. Registrar authenticating the pledge: "Who is this device? What is its identity?"
2. Registrar authorizing the pledge: "Is it mine? Do I want it? What are the chances it has been compromised?"
3. Pledge authenticating the registrar: "What is this registrar's identity?"
4. Pledge authorizing the registrar: "Should I join this network?"

This document details protocols and messages to answer the above questions. It uses a TLS connection and a PKIX-shaped (X.509v3) certificate (an IEEE 802.1AR IDevID [IDevID]) of the pledge to answer points 1 and 2. It uses a new artifact called a "voucher" that the registrar receives from a Manufacturer Authorized Signing Authority (MASA) and passes it to the pledge to answer points 3 and 4.

A proxy provides very limited connectivity between the pledge and the registrar.

The syntactic details of vouchers are described in detail in [RFC8366]. This document details automated protocol mechanisms to obtain vouchers, including the definition of a "voucher-request" message that is a minor extension to the voucher format (see Section 3) as defined by [RFC8366].

BRSKI results in the pledge storing an X.509 root certificate sufficient for verifying the registrar identity. In the process, a TLS connection is established that can be directly used for Enrollment over Secure Transport (EST). In effect, BRSKI provides an automated mechanism for "Bootstrap Distribution of CA Certificates" described in [RFC7030], Section 4.1.1, wherein the pledge "**MUST** [...] engage a human user to authorize the CA certificate using out-of-band data". With BRSKI, the pledge now can automate this process using the voucher. Integration with a complete EST enrollment is optional but trivial.

BRSKI is agile enough to support bootstrapping alternative key infrastructures, such as a symmetric key solution, but no such system is described in this document.

1.1. Prior Bootstrapping Approaches

To literally "pull yourself up by the bootstraps" is an impossible action. Similarly, the secure establishment of a key infrastructure without external help is also an impossibility. Today, it is commonly accepted that the initial connections between nodes are insecure, until key distribution is complete, or that domain-specific keying material (often pre-shared keys, including mechanisms like Subscriber Identification Module (SIM) cards) is pre-provisioned on each new device in a costly and non-scalable manner. Existing automated mechanisms are known as non-secured "Trust on First Use (TOFU)" [RFC7435], "resurrecting duckling" [Stajano99theresurrecting], or "pre-staging".

Another prior approach has been to try and minimize user actions during bootstrapping, but not eliminate all user actions. The original EST protocol [RFC7030] does reduce user actions during bootstrapping but does not provide solutions for how the following protocol steps can be made autonomic (not involving user actions):

- using the Implicit Trust Anchor (TA) [RFC7030] database to authenticate an owner-specific service (not an autonomic solution because the URL must be securely distributed),
- engaging a human user to authorize the CA certificate using out-of-band data (not an autonomic solution because the human user is involved),
- using a configured Explicit TA database (not an autonomic solution because the distribution of an explicit TA database is not autonomic), and
- using a certificate-less TLS mutual authentication method (not an autonomic solution because the distribution of symmetric key material is not autonomic).

These "touch" methods do not meet the requirements for zero-touch.

There are "call home" technologies where the pledge first establishes a connection to a well-known manufacturer service using a common client-server authentication model. After mutual authentication, appropriate credentials to authenticate the target domain are transferred to the pledge. This creates several problems and limitations:

- the pledge requires real-time connectivity to the manufacturer service,
- the domain identity is exposed to the manufacturer service (this is a privacy concern), and
- the manufacturer is responsible for making the authorization decisions (this is a liability concern).

BRSKI addresses these issues by defining extensions to the EST protocol for the automated distribution of vouchers.

1.2. Terminology

The key words "**MUST**", "**MUST NOT**", "**REQUIRED**", "**SHALL**", "**SHALL NOT**", "**SHOULD**", "**SHOULD NOT**", "**RECOMMENDED**", "**NOT RECOMMENDED**", "**MAY**", and "**OPTIONAL**" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

The following terms are defined for clarity:

- ANI:

  The Autonomic Networking Infrastructure as defined by [RFC8993]. Section 9 details specific requirements for pledges, proxies, and registrars when they are part of an ANI.

- Circuit Proxy:

  A stateful implementation of the Join Proxy. This is the assumed type of proxy.

- drop-ship:

  The physical distribution of equipment containing the "factory default" configuration to a final destination. In zero-touch scenarios, there is no staging or preconfiguration during drop-ship.

- Domain:

  The set of entities that share a common local trust anchor. This includes the proxy, registrar, domain CA, management components, and any existing entity that is already a member of the domain.

- Domain CA:

  The domain Certification Authority (CA) provides certification functionalities to the domain. At a minimum, it provides certification functionalities to a registrar and manages the private key that defines the domain. Optionally, it certifies all elements.

- domainID:

  The domain IDentity is a unique value based upon the registrar's CA certificate. Section 5.8.2 specifies how it is calculated.

- enrollment:

  The process where a device presents key material to a network and acquires a network-specific identity. For example, when a certificate signing request is presented to a CA, and a certificate is obtained in response.

- IDevID:

  An Initial Device Identifier X.509 certificate installed by the vendor on new equipment. This is a term from 802.1AR [IDevID].

- imprint:

  The process where a device obtains the cryptographic key material to identify and trust future interactions with a network. This term is taken from Konrad Lorenz's work in biology with new ducklings: during a critical period, the duckling would assume that anything that looks like a mother duck is in fact their mother. An equivalent for a device is to obtain the fingerprint of the network's root CA certificate. A device that imprints on an attacker suffers a similar fate to a duckling that imprints on a hungry wolf. Securely imprinting is a primary focus of this document [imprinting]. The analogy to Lorenz's work was first noted in [Stajano99theresurrecting].

- IPIP Proxy:

  A stateless proxy alternative.

- Join Proxy:

  A domain entity that helps the pledge join the domain. A Join Proxy facilitates communication for devices that find themselves in an environment where they are not provided connectivity until after they are validated as members of the domain. For simplicity, this document sometimes uses the term of "proxy" to indicate the Join Proxy. The pledge is unaware that they are communicating with a proxy rather than directly with a registrar.

- Join Registrar (and Coordinator):

  A representative of the domain that is configured, perhaps autonomically, to decide whether a new device is allowed to join the domain. The administrator of the domain interfaces with a "Join Registrar (and Coordinator)" to control this process. Typically, a Join Registrar is "inside" its domain. For simplicity, this document often refers to this as just "registrar". Within [RFC8993], it is referred to as the "Join Registrar Autonomic Service Agent (ASA)". Other communities use the abbreviation "JRC".

- LDevID:

  A Local Device Identifier X.509 certificate installed by the owner of the equipment. This is a term from 802.1AR [IDevID].

- manufacturer:

  The term manufacturer is used throughout this document as the entity that created the device. This is typically the original equipment manufacturer (OEM), but in more complex situations, it could be a value added retailer (VAR), or possibly even a systems integrator. In general, a goal of BRSKI is to eliminate small distinctions between different sales channels. The reason for this is that it permits a single device, with a uniform firmware load, to be shipped directly to all customers. This eliminates costs for the manufacturer. This also reduces the number of products supported in the field, increasing the chance that firmware will be more up to date.

- MASA Audit-Log:

  An anonymized list of previous owners maintained by the MASA on a per-device (per-pledge) basis, as described in Section 5.8.1.

- MASA Service:

  A third-party MASA service on the global Internet. The MASA signs vouchers. It also provides a repository for audit-log information of privacy-protected bootstrapping events. It does not track ownership.

- nonced:

  A voucher (or request) that contains a nonce (the normal case).

- nonceless:

  A voucher (or request) that does not contain a nonce and either relies upon accurate clocks for expiration or does not expire.

- offline:

  When an architectural component cannot perform real-time communications with a peer, due to either network connectivity or the peer being turned off, the operation is said to be occurring offline.

- Ownership Tracker:

  An Ownership Tracker service on the global Internet. The Ownership Tracker uses business processes to accurately track ownership of all devices shipped against domains that have purchased them. Although optional, this component allows vendors to provide additional value in cases where their sales and distribution channels allow for accurate tracking of such ownership. Tracking information about ownership is indicated in vouchers, as described in [RFC8366].

- Pledge:

  The prospective (unconfigured) device, which has an identity installed at the factory.

- (Public) Key Infrastructure:

  The collection of systems and processes that sustains the activities of a public key system. The registrar acts as a "Registration Authority"; see [RFC5280] and Section 7 of [RFC5272].

- TOFU:

  Trust on First Use. Used similarly to how it is described in [RFC7435]. This is where a pledge device makes no security decisions but rather simply trusts the first registrar it is contacted by. This is also known as the "resurrecting duckling" model.

- Voucher:

  A signed artifact from the MASA that indicates the cryptographic identity of the registrar it should trust to a pledge. There are different types of vouchers depending on how that trust is asserted. Multiple voucher types are defined in [RFC8366].

1.3. Scope Of Solution

1.3.1. Support Environment

This solution (BRSKI) can support large router platforms with multi-gigabit inter-connections, mounted in controlled access data centers. But this solution is not exclusive to large equipment: it is intended to scale to thousands of devices located in hostile environments, such as ISP-provided Customer Premises Equipment (CPE) devices that are drop-shipped to the end user. The situation where an order is fulfilled from a distributed warehouse from a common stock and shipped directly to the target location at the request of a domain owner is explicitly supported. That stock ("SKU") could be provided to a number of potential domain owners, and the eventual domain owner will not know a priori which device will go to which location.

The bootstrapping process can take minutes to complete depending on the network infrastructure and device processing speed. The network communication itself is not optimized for speed; for privacy reasons, the discovery process allows for the pledge to avoid announcing its presence through broadcasting.

Nomadic or mobile devices often need to acquire credentials to access the network at the new location. An example of this is mobile phone roaming among network operators, or even between cell towers. This is usually called "handoff". BRSKI does not provide a low-latency handoff, which is usually a requirement in such situations. For these solutions, BRSKI can be used to create a relationship (an LDevID) with the "home" domain owner. The resulting credentials are then used to provide credentials more appropriate for a low-latency handoff.

1.3.2. Constrained Environments

Questions have been posed as to whether this solution is suitable in general for Internet of Things (IoT) networks. This depends on the capabilities of the devices in question. The terminology of [RFC7228] is best used to describe the boundaries.

The solution described in this document is aimed in general at non-constrained (i.e., Class 2+ [RFC7228]) devices operating on a non-challenged network. The entire solution as described here is not intended to be usable as is by constrained devices operating on challenged networks (such as 802.15.4 Low-Power and Lossy Networks (LLNs)).

Specifically, there are protocol aspects described here that might result in congestion collapse or energy exhaustion of intermediate battery-powered routers in an LLN. Those types of networks should not use this solution. These limitations are predominately related to the large credential and key sizes required for device authentication. Defining symmetric key techniques that meet the operational requirements is out of scope, but the underlying protocol operations (TLS handshake and signing structures) have sufficient algorithm agility to support such techniques when defined.

The imprint protocol described here could, however, be used by non-energy constrained devices joining a non-constrained network (for instance, smart light bulbs are usually mains powered and use 802.11 wireless technology). It could also be used by non-constrained devices across a non-energy constrained, but challenged, network (such as 802.15.4). The certificate contents, and the process by which the four questions above are resolved, do apply to constrained devices. It is simply the actual on-the-wire imprint protocol that could be inappropriate.

1.3.3. Network Access Controls

This document presumes that network access control has already occurred, is not required, or is integrated by the proxy and registrar in such a way that the device itself does not need to be aware of the details. Although the use of an X.509 IDevID is consistent with IEEE 802.1AR [IDevID], and allows for alignment with 802.1X network access control methods, its use here is for pledge authentication rather than network access control. Integrating this protocol with network access control, perhaps as an Extensible Authentication Protocol (EAP) method (see [RFC3748]), is out of scope for this document.

1.3.4. Bootstrapping is Not Booting

This document describes "bootstrapping" as the protocol used to obtain a local trust anchor. It is expected that this trust anchor, along with any additional configuration information subsequently installed, is persisted on the device across system restarts ("booting"). Bootstrapping occurs only infrequently such as when a device is transferred to a new owner or has been reset to factory default settings.

1.4. Leveraging The New Key Infrastructure / Next Steps

As a result of the protocol described herein, bootstrapped devices have the domain CA trust anchor in common. An end-entity (EE) certificate has optionally been issued from the domain CA. This makes it possible to securely deploy functionalities across the domain; for example:

- Device management
- Routing authentication
- Service discovery

The major intended benefit is the ability to use the credentials deployed by this protocol to secure the Autonomic Control Plane (ACP) [RFC8994].

1.5. Requirements For Autonomic Networking Infrastructure (ANI) Devices

The BRSKI protocol can be used in a number of environments. Some of the options in this document are the result of requirements that are out of the ANI scope. This section defines the base requirements for ANI devices.

For devices that intend to become part of an ANI [RFC8993] that includes an Autonomic Control Plane [RFC8994], the BRSKI protocol **MUST** be implemented.

The pledge must perform discovery of the proxy as described in Section 4.1 using the Discovery Unsolicited Link-Local (DULL) [RFC8990] M_FLOOD announcements of the GeneRic Autonomic Signaling Protocol (GRASP).

Upon successfully validating a voucher artifact, a status telemetry **MUST** be returned; see Section 5.7.

An ANIMA ANI pledge **MUST** implement the EST automation extensions described in Section 5.9. They supplement the EST [RFC7030] to better support automated devices that do not have an end user.

The ANI Join Registrar ASA **MUST** support all the BRSKI and above-listed EST operations.

All ANI devices **SHOULD** support the BRSKI proxy function, using Circuit Proxies over the Autonomic Control Plane (ACP) (see Section 4.3).

2. Architectural Overview

The logical elements of the bootstrapping framework are described in this section. Figure 1 provides a simplified overview of the components.

```
                                           +------------------------+
   +--------------Drop-Ship----------------| Vendor Service         |
   |                                       +------------------------+
   |                                       | M anufacturer|         |
   |                                       | A uthorized  |Ownership|
   |                                       | S igning     |Tracker  |
   |                                       | A uthority   |         |
   |                                       +--------------+---------+
   |                                                      ^
   |                                                      |  BRSKI-
   V                                                      |   MASA
+-------+     ............................................|...
|       |     .                                           |  .
|       |     .  +------------+       +-----------+       |  .
|       |     .  |            |       |           |       |  .
|Pledge |     .  |   Join     |       | Domain    \<-------+  .
|       |     .  |   Proxy    |       | Registrar |          .
|       \<-------->............\<-------> (PKI RA)  |          .
|       |        |        BRSKI-EST   |           |          .
|       |     .  |            |       +-----+-----+          .
|IDevID |     .  +------------+             | e.g., RFC 7030 .
|       |     .           +-----------------+----------+     .
|       |     .           | Key Infrastructure         |     .
|       |     .           | (e.g., PKI CA)             |     .
+-------+     .           |                            |     .
              .           +----------------------------+     .
              .                                              .
              ................................................
                            "Domain" Components
```

Figure 1: Architecture Overview

We assume a multivendor network. In such an environment, there could be a manufacturer service for each manufacturer that supports devices following this document's specification, or an integrator could provide a generic service authorized by multiple manufacturers. It is unlikely that an integrator could provide ownership tracking services for multiple manufacturers due to the required sales channel integrations necessary to track ownership.

The domain is the managed network infrastructure with a key infrastructure that the pledge is joining. The domain provides initial device connectivity sufficient for bootstrapping through a proxy. The domain registrar authenticates the pledge, makes authorization decisions, and distributes vouchers obtained from the manufacturer service. Optionally, the registrar also acts as a PKI CA.

2.1. Behavior Of A Pledge

The pledge goes through a series of steps, which are outlined here at a high level.

```
               ------------
              /  Factory   \
              \  default   /
               -----+------
                    |
             +------v-------+
             | (1) Discover |
+------------>              |
|            +------+-------+
|                   |
|            +------v-------+
|            | (2) Identify |
^------------+              |
| rejected   +------+-------+
|                   |
|            +------v-------+
|            | (3) Request  |
|            |     Join     |
|            +------+-------+
|                   |
|            +------v-------+
|            | (4) Imprint  |
^------------+              |
| Bad MASA   +------+-------+
| response          |  send Voucher Status Telemetry
|            +------v-------+
|            | (5) Enroll   |\<---+ (non-error HTTP codes)
^------------+              |\___/ (e.g., 202 "Retry-After")
| Enroll     +------+-------+
| failure           |
|              -----v------
|             /  Enrolled  \
^------------+             |
 Factory      \------------/
 reset
```

Figure 2: Pledge State Diagram

State descriptions for the pledge are as follows:

1. Discover a communication channel to a registrar.
2. Identify itself. This is done by presenting an X.509 IDevID credential to the discovered registrar (via the proxy) in a TLS handshake. (The registrar credentials are only provisionally accepted at this time.)
3. Request to join the discovered registrar. A unique nonce is included, ensuring that any responses can be associated with this particular bootstrapping attempt.
4. Imprint on the registrar. This requires verification of the manufacturer-service-provided voucher. A voucher contains sufficient information for the pledge to complete authentication of a registrar. This document details this step in depth.
5. Enroll. After imprint, an authenticated TLS (HTTPS) connection exists between the pledge and registrar. EST [RFC7030] can then be used to obtain a domain certificate from a registrar.

The pledge is now a member of, and can be managed by, the domain and will only repeat the discovery aspects of bootstrapping if it is returned to factory default settings.

This specification details integration with EST enrollment so that pledges can optionally obtain a locally issued certificate, although any Representational State Transfer (REST) (see [REST]) interface could be integrated in future work.

2.2. Secure Imprinting Using Vouchers

A voucher is a cryptographically protected artifact (using a digital signature) to the pledge device authorizing a zero-touch imprint on the registrar domain.

The format and cryptographic mechanism of vouchers is described in detail in [RFC8366].

Vouchers provide a flexible mechanism to secure imprinting: the pledge device only imprints when a voucher can be validated. At the lowest security levels, the MASA can indiscriminately issue vouchers and log claims of ownership by domains. At the highest security levels, issuance of vouchers can be integrated with complex sales channel integrations that are beyond the scope of this document. The sales channel integration would verify actual (legal) ownership of the pledge by the domain. This provides the flexibility for a number of use cases via a single common protocol mechanism on the pledge and registrar devices that are to be widely deployed in the field. The MASA services have the flexibility to either leverage the currently defined claim mechanisms or experiment with higher or lower security levels.

Vouchers provide a signed but non-encrypted communication channel among the pledge, the MASA, and the registrar. The registrar maintains control over the transport and policy decisions, allowing the local security policy of the domain network to be enforced.

2.3. Initial Device Identifier

Pledge authentication and pledge voucher-request signing is via a PKIX-shaped certificate installed during the manufacturing process. This is the 802.1AR IDevID, and it provides a basis for authenticating the pledge during the protocol exchanges described here. There is no requirement for a common root PKI hierarchy. Each device manufacturer can generate its own root certificate. Specifically, the IDevID enables:

- Uniquely identifying the pledge by the Distinguished Name (DN) and subjectAltName (SAN) parameters in the IDevID. The unique identification of a pledge in the voucher objects are derived from those parameters as described below. Section 10.3 discusses privacy implications of the identifier.
- Providing a cryptographic authentication of the pledge to the registrar (see Section 5.3).
- Securing auto-discovery of the pledge's MASA by the registrar (see Section 2.8).
- Signing of a voucher-request by the pledge's IDevID (see Section 3).
- Providing a cryptographic authentication of the pledge to the MASA (see Section 5.5.5).

Sections 7.2.13 (2009 edition) and 8.10.3 (2018 edition) of [IDevID] discuss keyUsage and extendedKeyUsage extensions in the IDevID certificate. [IDevID] acknowledges that adding restrictions in the certificate limits applicability of these long-lived certificates. This specification emphasizes this point and therefore RECOMMENDS that no key usage restrictions be included. This is consistent with [RFC5280], Section 4.2.1.3, which does not require key usage restrictions for end-entity certificates.

2.3.1. Identification of the Pledge

In the context of BRSKI, pledges have a 1:1 relationship with a "serial-number". This serial-number is used both in the serial-number field of a voucher or voucher-requests (see Section 3) and in local policies on the registrar or MASA (see Section 5).

There is a (certificate) serialNumber field defined in [RFC5280], Section 4.1.2.2. In ASN.1, this is referred to as the CertificateSerialNumber. This field is NOT relevant to this specification. Do not confuse this field with the serial-number defined by this document, or by [IDevID] and [RFC4519], Section 2.31.

The device serial number is defined in Appendix A.1 of [RFC5280] as the X520SerialNumber, with the OID tag id-at-serialNumber.

The device *serialNumber* field (X520SerialNumber) is used as follows by the pledge to build the **serial-number** that is placed in the voucher-request. In order to build it, the fields need to be converted into a serial-number of "type string".

An example of a printable form of the serialNumber field is provided in [RFC4519], Section 2.31 ("WI-3005"). That section further provides equality and syntax attributes.

Due to the reality of existing device identity provisioning processes, some manufacturers have stored serial-numbers in other fields. Registrars **SHOULD** be configurable, on a per-manufacturer basis, to look for serial-number equivalents in other fields.

As explained in Section 5.5, the registrar **MUST** again extract the serialNumber itself from the pledge's TLS certificate. It can consult the serial-number in the pledge request if there is any possible confusion about the source of the serial-number.

2.3.2. MASA URI Extension

This document defines a new PKIX non-critical certificate extension to carry the MASA URI. This extension is intended to be used in the IDevID certificate. The URI is represented as described in Section 7.4 of [RFC5280].

The URI provides the authority information. The BRSKI "/.well-known" tree [RFC8615] is described in Section 5.

A complete URI **MAY** be in this extension, including the "scheme", "authority", and "path". The complete URI will typically be used in diagnostic or experimental situations. Typically (and in consideration to constrained systems), this **SHOULD** be reduced to only the "authority", in which case a scheme of "https://" (see [RFC7230], Section 2.7.3) and a "path" of "/.well-known/brski" is to be assumed.

The registrar can assume that only the "authority" is present in the extension, if there are no slash ("/") characters in the extension.

Section 7.4 of [RFC5280] calls out various schemes that **MUST** be supported, including the Lightweight Directory Access Protocol (LDAP), HTTP, and FTP. However, the registrar **MUST** use HTTPS for the BRSKI-MASA connection.

The new extension is identified as follows:

```
\<CODE BEGINS>
MASAURLExtnModule-2016 { iso(1) identified-organization(3) dod(6)
internet(1) security(5) mechanisms(5) pkix(7)
id-mod(0) id-mod-MASAURLExtn2016(96) }

DEFINITIONS IMPLICIT TAGS ::= BEGIN

-- EXPORTS ALL --

IMPORTS
EXTENSION
FROM PKIX-CommonTypes-2009
  { iso(1) identified-organization(3) dod(6) internet(1)
    security(5) mechanisms(5) pkix(7) id-mod(0)
    id-mod-pkixCommon-02(57) }

id-pe FROM PKIX1Explicit-2009
  { iso(1) identified-organization(3) dod(6) internet(1)
     security(5) mechanisms(5) pkix(7) id-mod(0)
     id-mod-pkix1-explicit-02(51) } ;

MASACertExtensions EXTENSION ::= { ext-MASAURL, ... }
ext-MASAURL EXTENSION ::= { SYNTAX MASAURLSyntax
IDENTIFIED BY id-pe-masa-url }

id-pe-masa-url OBJECT IDENTIFIER ::= { id-pe 32 }

MASAURLSyntax ::= IA5String

END
```

```
\<CODE ENDS>
```

Figure 3: MASAURL ASN.1 Module

The choice of id-pe is based on guidance found in Section 4.2.2 of [RFC5280]: "These extensions may be used to direct applications to on-line information about the issuer or the subject". The MASA URL is precisely that: online information about the particular subject.

2.4. Protocol Flow

A representative flow is shown in Figure 4.

```
+--------+         +---------+    +------------+     +------------+
| Pledge |         | Circuit |    | Domain     |     | Vendor     |
|        |         | Join    |    | Registrar  |     | Service    |
|        |         | Proxy   |    |  (JRC)     |     | (MASA)     |
+--------+         +---------+    +------------+     +------------+
  |                     |                   |           Internet |
[discover]              |                   |                    |
  |\<-RFC 4862 IPv6 addr |                   |                    |
  |\<-RFC 3927 IPv4 addr | Appendix A        |  Legend            |
  |-++++++++++++++++++->|                   | C - Circuit        |
  | optional: mDNS query| Appendix B        |     Join Proxy     |
  | RFCs 6763/6762 (+)  |                   | P - Provisional TLS|
  |\<-++++++++++++++++++-|                   |     Connection     |
  | GRASP M_FLOOD       |                   |                    |
  |   periodic broadcast|                   |                    |
[identity]              |                   |                    |
  |\<------------------->C\<----------------->|                    |
  |         TLS via the Join Proxy          |                    |
  |\<--Registrar TLS server authentication---|                    |
[PROVISIONAL accept of server cert]         |                    |
  P---X.509 client authentication---------->|                    |
[request join]                              |                    |
  P---Voucher-Request(w/nonce for voucher)->|                    |
  P                  /-------------------   |                    |
  P                  |                 [accept device?]          |
  P                  |                 [contact vendor]          |
  P                  |                      |--Pledge ID-------->|
  P                  |                      |--Domain ID-------->|
  P                  |                     |--optional:nonce--->|
  P              optional:                 |     [extract DomainID]
  P        can occur in advance            |     [update audit-log]
  P            if nonceless                |                    |
  P                  |                     |\<- voucher ---------|
  P                  \------------------   | w/nonce if provided|
  P\<------voucher--------------------------|                    |
[imprint]                                  |                    |
  |-------voucher status telemetry--------->|                    |
  |                                        |\<-device audit-log--|
  |                            [verify audit-log and voucher]   |
  |\<--------------------------------------->|                    |
[enroll]                                   |                    |
  | Continue with enrollment using now     |                    |
  | bidirectionally authenticated TLS      |                    |
  | session per RFC 7030.                  |                    |
[enrolled]                                 |                    |
```

Figure 4: Protocol Time Sequence Diagram

On initial bootstrap, a new device (the pledge) uses a local service auto-discovery (the GeneRic Autonomic Signaling Protocol (GRASP) or Multicast DNS (mDNS)) to locate a Join Proxy. The Join Proxy connects the pledge to a local registrar (the JRC).

Having found a candidate registrar, the fledgling pledge sends some information about itself to the registrar, including its serial number in the form of a voucher-request and its IDevID certificate as part of the TLS session.

The registrar can determine whether it expected such a device to appear and locates a MASA. The location of the MASA is usually found in an extension in the IDevID. Having determined that the MASA is suitable, the entire information from the initial voucher-request (including the device's serial number) is transmitted over the Internet in a TLS-protected channel to the manufacturer, along with information about the registrar/owner.

The manufacturer can then apply policy based on the provided information, as well as other sources of information (such as sales records), to decide whether to approve the claim by the registrar to own the device; if the claim is accepted, a voucher is issued that directs the device to accept its new owner.

The voucher is returned to the registrar, but not immediately to the device -- the registrar has an opportunity to examine the voucher, the MASA's audit-logs, and other sources of information to determine whether the device has been tampered with and whether the bootstrap should be accepted.

No filtering of information is possible in the signed voucher, so this is a binary yes-or-no decision. After the registrar has applied any local policy to the voucher, if it accepts the voucher, then the voucher is returned to the pledge for imprinting.

The voucher also includes a trust anchor that the pledge uses to represent the owner. This is used to successfully bootstrap from an environment where only the manufacturer has built-in trust by the device to an environment where the owner now has a PKI footprint on the device.

When BRSKI is followed with EST, this single footprint is further leveraged into the full owner's PKI and an LDevID for the device. Subsequent reporting steps provide flows of information to indicate success/failure of the process.

2.5. Architectural Components

2.5.1. Pledge

The pledge is the device that is attempting to join. It is assumed that the pledge talks to the Join Proxy using link-local network connectivity. In most cases, the pledge has no other connectivity until the pledge completes the enrollment process and receives some kind of network credential.

2.5.2. Join Proxy

The Join Proxy provides HTTPS connectivity between the pledge and the registrar. A Circuit Proxy mechanism is described in Section 4. Additional mechanisms, including a Constrained Application Protocol (CoAP) mechanism and a stateless IP in IP (IPIP) mechanism, are the subject of future work.

2.5.3. Domain Registrar

The domain's registrar operates as the BRSKI-MASA client when requesting vouchers from the MASA (see Section 5.4). The registrar operates as the BRSKI-EST server when pledges request vouchers (see Section 5.1). The registrar operates as the BRSKI-EST server "Registration Authority" if the pledge requests an end-entity certificate over the BRSKI-EST connection (see Section 5.9).

The registrar uses an Implicit Trust Anchor database for authenticating the BRSKI-MASA connection's MASA TLS server certificate. Configuration or distribution of trust anchors is out of scope for this specification.

The registrar uses a different Implicit Trust Anchor database for authenticating the BRSKI-EST connection's pledge TLS Client Certificate. Configuration or distribution of the BRSKI-EST client trust anchors is out of scope of this specification. Note that the trust anchors in / excluded from the database will affect which manufacturers' devices are acceptable to the registrar as pledges, and they can also be used to limit the set of MASAs that are trusted for enrollment.

2.5.4. Manufacturer Service

The manufacturer service provides two logically separate functions: the MASA as described in Sections 5.5 and 5.6 and an ownership tracking/auditing function as described in Sections 5.7 and 5.8.

2.5.5. Public Key Infrastructure (PKI)

The Public Key Infrastructure (PKI) administers certificates for the domain of concern, providing the trust anchor(s) for it and allowing enrollment of pledges with domain certificates.

The voucher provides a method for the distribution of a single PKI trust anchor (as the "pinned-domain-cert"). A distribution of the full set of current trust anchors is possible using the optional EST integration.

The domain's registrar acts as a Registration Authority [RFC5272], requesting certificates for pledges from the PKI.

The expectations of the PKI are unchanged from EST [RFC7030]. This document does not place any additional architectural requirements on the PKI.

2.6. Certificate Time Validation

2.6.1. Lack of Real-Time Clock

When bootstrapping, many devices do not have knowledge of the current time. Mechanisms such as Network Time Protocols cannot be secured until bootstrapping is complete. Therefore, bootstrapping is defined with a framework that does not require knowledge of the current time. A pledge **MAY** ignore all time stamps in the voucher and in the certificate validity periods if it does not know the current time.

The pledge is exposed to dates in the following five places: registrar certificate notBefore, registrar certificate notAfter, voucher created-on, and voucher expires-on. Additionally, Cryptographic Message Syntax (CMS) signatures contain a signingTime.

A pledge with a real-time clock in which it has confidence **MUST** check the above time fields in all certificates and signatures that it processes.

If the voucher contains a nonce, then the pledge **MUST** confirm the nonce matches the original pledge voucher-request. This ensures the voucher is fresh. See Section 5.2.

2.6.2. Infinite Lifetime of IDevID

Long-lived pledge certificates "**SHOULD** be assigned the GeneralizedTime value of 99991231235959Z" for the notAfter field as explained in [RFC5280].

Some deployed IDevID management systems are not compliant with the 802.1AR requirement for infinite lifetimes and are put in typical <= 3 year certificate lifetimes. Registrars **SHOULD** be configurable on a per-manufacturer basis to ignore pledge lifetimes when the pledge does not follow the recommendations in [RFC5280].

2.7. Cloud Registrar

There exist operationally open networks wherein devices gain unauthenticated access to the Internet at large. In these use cases, the management domain for the device needs to be discovered within the larger Internet. The case where a device can boot and get access to a larger Internet is less likely within the ANIMA ACP scope but may be more important in the future. In the ANIMA ACP scope, new devices will be quarantined behind a Join Proxy.

Additionally, there are some greenfield situations involving an entirely new installation where a device may have some kind of management uplink that it can use (such as via a 3G network, for instance). In such a future situation, the device might use this management interface to learn that it should configure itself to become the local registrar.

In order to support these scenarios, the pledge **MAY** contact a well-known URI of a cloud registrar if a local registrar cannot be discovered or if the pledge's target use cases do not include a local registrar.

If the pledge uses a well-known URI for contacting a cloud registrar, a manufacturer-assigned Implicit Trust Anchor database (see [RFC7030]) **MUST** be used to authenticate that service as described in [RFC6125]. The use of a DNS-ID for validation is appropriate, and it may include wildcard components on the left-mode side. This is consistent with the human-user configuration of an EST server URI in [RFC7030], which also depends on [RFC6125].

2.8. Determining The MASA To Contact

The registrar needs to be able to contact a MASA that is trusted by the pledge in order to obtain vouchers.

The device's IDevID will normally contain the MASA URL as detailed in Section 2.3. This is the **RECOMMENDED** mechanism.

In some cases, it can be operationally difficult to ensure the necessary X.509 extensions are in the pledge's IDevID due to the difficulty of aligning current pledge manufacturing with software releases and development; thus, as a final fallback, the registrar **MAY** be manually configured or distributed with a MASA URL for each manufacturer. Note that the registrar can only select the configured MASA URL based on the trust anchor -- so manufacturers can only leverage this approach if they ensure a single MASA URL works for all pledges associated with each trust anchor.

3. Voucher-Request Artifact

Voucher-requests are how vouchers are requested. The semantics of the voucher-request are described below, in the YANG module.

A pledge forms the "pledge voucher-request", signs it with its IDevID, and submits it to the registrar.

In turn, the registrar forms the "registrar voucher-request", signs it with its registrar key pair, and submits it to the MASA.

The "proximity-registrar-cert" leaf is used in the pledge voucher-requests. This provides a method for the pledge to assert the registrar's proximity.

This network proximity results from the following properties in the ACP context: the pledge is connected to the Join Proxy (Section 4) using a link-local IPv6 connection. While the Join Proxy does not participate in any meaningful sense in the cryptography of the TLS connection (such as via a Channel Binding), the registrar can observe that the connection is via the private ACP (ULA) address of the Join Proxy, and it cannot come from outside the ACP. The pledge must therefore be at most one IPv6 link-local hop away from an existing node on the ACP.

Other users of BRSKI will need to define other kinds of assertions if the network proximity described above does not match their needs.

The "prior-signed-voucher-request" leaf is used in registrar voucher-requests. If present, it is the signed pledge voucher-request artifact. This provides a method for the registrar to forward the pledge's signed request to the MASA. This completes transmission of the signed proximity-registrar-cert leaf.

Unless otherwise signaled (outside the voucher-request artifact), the signing structure is as defined for vouchers; see [RFC8366].

3.1. Nonceless Voucher-Requests

A registrar **MAY** also retrieve nonceless vouchers by sending nonceless voucher-requests to the MASA in order to obtain vouchers for use when the registrar does not have connectivity to the MASA. No prior-signed-voucher-request leaf would be included. The registrar will also need to know the serial number of the pledge. This document does not provide a mechanism for the registrar to learn that in an automated fashion. Typically, this will be done via the scanning of a bar code or QR code on packaging, or via some sales channel integration.

3.2. Tree Diagram

The following tree diagram illustrates a high-level view of a voucher-request document. The voucher-request builds upon the voucher artifact described in [RFC8366]. The tree diagram is described in [RFC8340]. Each node in the diagram is fully described by the YANG module in Section 3.4. Please review the YANG module for a detailed description of the voucher-request format.

```
module: ietf-voucher-request

 grouping voucher-request-grouping
  +-- voucher
     +-- created-on?                       yang:date-and-time
     +-- expires-on?                       yang:date-and-time
     +-- assertion?                        enumeration
     +-- serial-number                     string
     +-- idevid-issuer?                    binary
     +-- pinned-domain-cert?               binary
     +-- domain-cert-revocation-checks?    boolean
     +-- nonce?                            binary
     +-- last-renewal-date?                yang:date-and-time
     +-- prior-signed-voucher-request?     binary
     +-- proximity-registrar-cert?         binary
```

Figure 5: YANG Tree Diagram for a Voucher-Request

3.3. Examples

This section provides voucher-request examples for illustration purposes. These examples show JSON prior to CMS wrapping. JSON encoding rules specify that any binary content be base64 encoded ([RFC4648], Section 4). The contents of the (base64) encoded certificates have been elided to save space. For detailed examples, see Appendix C.2. These examples conform to the encoding rules defined in [RFC7951].

- Example (1):

  The following example illustrates a pledge voucher-request. The assertion leaf is indicated as "proximity", and the registrar's TLS server certificate is included in the proximity-registrar-cert leaf. See Section 5.2.

```
{
    "ietf-voucher-request:voucher": {
        "assertion": "proximity",
        "nonce": "62a2e7693d82fcda2624de58fb6722e5",
        "serial-number" : "JADA123456789",
        "created-on": "2017-01-01T00:00:00.000Z",
        "proximity-registrar-cert": "base64encodedvalue=="
    }
}
```

Figure 6: JSON Representation of an Example Voucher-Request

- Example (2):

  The following example illustrates a registrar voucher-request. The prior-signed-voucher-request leaf is populated with the pledge's voucher-request (such as the prior example). The pledge's voucher-request is a binary CMS-signed object. In the JSON encoding used here, it must be base64 encoded. The nonce and assertion have been carried forward from the pledge request to the registrar request. The serial-number is extracted from the pledge's Client Certificate from the TLS connection. See Section 5.5.

```
{
    "ietf-voucher-request:voucher": {
        "assertion" : "proximity",
        "nonce": "62a2e7693d82fcda2624de58fb6722e5",
        "created-on": "2017-01-01T00:00:02.000Z",
        "idevid-issuer": "base64encodedvalue==",
        "serial-number": "JADA123456789",
        "prior-signed-voucher-request": "base64encodedvalue=="
    }
}
```

Figure 7: JSON Representation of an Example Prior-Signed Voucher-Request

- Example (3):

  The following example illustrates a registrar voucher-request. The prior-signed-voucher-request leaf is not populated with the pledge's voucher-request nor is the nonce leaf. This form might be used by a registrar requesting a voucher when the pledge cannot communicate with the registrar (such as when it is powered down or still in packaging) and therefore cannot submit a nonce. This scenario is most useful when the registrar is aware that it will not be able to reach the MASA during deployment. See Section 5.5.

```
{
    "ietf-voucher-request:voucher": {
        "created-on":    "2017-01-01T00:00:02.000Z",
        "idevid-issuer": "base64encodedvalue==",
        "serial-number": "JADA123456789"
    }
}
```

Figure 8: JSON Representation of an Offline Voucher-Request

3.4. YANG Module

Following is a YANG module [RFC7950] that formally extends a voucher [RFC8366] into a voucher-request. This YANG module references [ITU.X690].

```
\<CODE BEGINS> file "ietf-voucher-request@2021-05-20.yang"

module ietf-voucher-request {
  yang-version 1.1;
  namespace "urn:ietf:params:xml:ns:yang:ietf-voucher-request";
  prefix vcr;

  import ietf-restconf {
    prefix rc;
    description
      "This import statement is only present to access
       the yang-data extension defined in RFC 8040.";
    reference
      "RFC 8040: RESTCONF Protocol";
  }
  import ietf-voucher {
    prefix vch;
    description
      "This module defines the format for a voucher,
       which is produced by a pledge's manufacturer or
       delegate (MASA) to securely assign a pledge to
       an 'owner', so that the pledge may establish a secure
       connection to the owner's network infrastructure.";
    reference
      "RFC 8366: A Voucher Artifact for
       Bootstrapping Protocols";
  }

  organization
    "IETF ANIMA Working Group";
  contact
    "WG Web:   \<https://datatracker.ietf.org/wg/anima/>
     WG List:  \<mailto:anima@ietf.org>
     Author:   Kent Watsen
               \<mailto:kent+ietf@watsen.net>
     Author:   Michael H. Behringer
               \<mailto:Michael.H.Behringer@gmail.com>
```

```
   Author:   Toerless Eckert
             \<mailto:tte+ietf@cs.fau.de>
   Author:   Max Pritikin
             \<mailto:pritikin@cisco.com>
   Author:   Michael Richardson
             \<mailto:mcr+ietf@sandelman.ca>";
description
  "This module defines the format for a voucher-request.
   It is a superset of the voucher itself.
   It provides content to the MASA for consideration
   during a voucher-request.

   The key words 'MUST', 'MUST NOT', 'REQUIRED', 'SHALL', 'SHALL
   NOT', 'SHOULD', 'SHOULD NOT', 'RECOMMENDED', 'NOT RECOMMENDED',
   'MAY', and 'OPTIONAL' in this document are to be interpreted as
   described in BCP 14 (RFC 2119) (RFC 8174) when, and only when,
   they appear in all capitals, as shown here.

   Copyright (c) 2021 IETF Trust and the persons identified as
   authors of the code. All rights reserved.

   Redistribution and use in source and binary forms, with or
   without modification, is permitted pursuant to, and subject
   to the license terms contained in, the Simplified BSD License
   set forth in Section 4.c of the IETF Trust's Legal Provisions
   Relating to IETF Documents
   (https://trustee.ietf.org/license-info).

   This version of this YANG module is part of RFC 8995; see the
   RFC itself for full legal notices.";

revision 2021-05-20 {
  description
    "Initial version";
  reference
    "RFC 8995: Bootstrapping Remote Secure Key Infrastructure
     (BRSKI)";
}

// Top-level statement
rc:yang-data voucher-request-artifact {
  uses voucher-request-grouping;
```

```
}

// Grouping defined for future usage

grouping voucher-request-grouping {
  description
    "Grouping to allow reuse/extensions in future work.";
  uses vch:voucher-artifact-grouping {
    refine "voucher/created-on" {
      mandatory false;
    }
    refine "voucher/pinned-domain-cert" {
      mandatory false;
      description
        "A pinned-domain-cert field is not valid in a
         voucher-request, and any occurrence MUST be ignored.";
    }
    refine "voucher/last-renewal-date" {
      description
        "A last-renewal-date field is not valid in a
         voucher-request, and any occurrence MUST be ignored.";
    }
    refine "voucher/domain-cert-revocation-checks" {
      description
        "The domain-cert-revocation-checks field is not valid in a
         voucher-request, and any occurrence MUST be ignored.";
    }
    refine "voucher/assertion" {
      mandatory false;
      description
        "Any assertion included in registrar voucher-requests
         SHOULD be ignored by the MASA.";
    }
    augment "voucher" {
      description
        "Adds leaf nodes appropriate for requesting vouchers.";
      leaf prior-signed-voucher-request {
        type binary;
        description
          "If it is necessary to change a voucher, or re-sign and
           forward a voucher that was previously provided along a
           protocol path, then the previously signed voucher SHOULD
```

```
             be included in this field.

             For example, a pledge might sign a voucher-request
             with a proximity-registrar-cert, and the registrar
             then includes it as the prior-signed-voucher-request
             field.  This is a simple mechanism for a chain of
             trusted parties to change a voucher-request, while
             maintaining the prior signature information.

             The registrar and MASA MAY examine the prior-signed
             voucher information for the
             purposes of policy decisions.  For example, this
             information could be useful to a MASA to determine
             that both the pledge and registrar agree on proximity
             assertions.  The MASA SHOULD remove all
             prior-signed-voucher-request information when
             signing a voucher for imprinting so as to minimize
             the final voucher size.";
        }
        leaf proximity-registrar-cert {
          type binary;
          description
            "An X.509 v3 certificate structure, as specified by
             RFC 5280, Section 4, encoded using the ASN.1
             distinguished encoding rules (DER), as specified
             in ITU X.690.

             The first certificate in the registrar TLS server
             certificate_list sequence (the end-entity TLS
             certificate; see RFC 8446) presented by the registrar
             to the pledge.  This MUST be populated in a pledge's
             voucher-request when a proximity assertion is
             requested.";
          reference
            "ITU X.690: Information Technology - ASN.1 encoding
             rules: Specification of Basic Encoding Rules (BER),
             Canonical Encoding Rules (CER) and Distinguished
             Encoding Rules (DER)
             RFC 5280: Internet X.509 Public Key Infrastructure
             Certificate and Certificate Revocation List (CRL)
             Profile
             RFC 8446: The Transport Layer Security (TLS)
             Protocol Version 1.3";
        }
      }
    }
  }
}

\<CODE ENDS>
```

Figure 9: YANG Module for Voucher-Request

4. Proxying Details (Pledge -- Proxy -- Registrar)

This section is normative for uses with an ANIMA ACP. The use of the GRASP mechanism is part of the ACP. Other users of BRSKI will need to define an equivalent proxy mechanism and an equivalent mechanism to configure the proxy.

The role of the proxy is to facilitate communications. The proxy forwards packets between the pledge and a registrar that has been provisioned to the proxy via full GRASP ACP discovery.

This section defines a stateful proxy mechanism that is referred to as a "circuit" proxy. This is a form of Application Level Gateway (see [RFC2663], Section 2.9).

The proxy does not terminate the TLS handshake: it passes streams of bytes onward without examination. A proxy **MUST NOT** assume any specific TLS version. Please see [RFC8446], Section 9.3 for details on TLS invariants.

A registrar can directly provide the proxy announcements described below, in which case the announced port can point directly to the registrar itself. In this scenario, the pledge is unaware that there is no proxying occurring. This is useful for registrars that are servicing pledges on directly connected networks.

As a result of the proxy discovery process in Section 4.1.1, the port number exposed by the proxy does not need to be well known or require an IANA allocation.

During the discovery of the registrar by the Join Proxy, the Join Proxy will also learn which kinds of proxy mechanisms are available. This will allow the Join Proxy to use the lowest impact mechanism that the Join Proxy and registrar have in common.

In order to permit the proxy functionality to be implemented on the maximum variety of devices, the chosen mechanism should use the minimum amount of state on the proxy device. While many devices in the ANIMA target space will be rather large routers, the proxy function is likely to be implemented in the control-plane CPU of such a device, with available capabilities for the proxy function similar to many class 2 IoT devices.

The document [ANIMA-STATE] provides a more extensive analysis and background of the alternative proxy methods.

4.1. Pledge Discovery Of Proxy

The result of discovery is a logical communication with a registrar, through a proxy. The proxy is transparent to the pledge. The communication between the pledge and Join Proxy is over IPv6 link-local addresses.

To discover the proxy, the pledge performs the following actions:

1. **MUST**: Obtain a local address using IPv6 methods as described in "IPv6 Stateless Address Autoconfiguration" [RFC4862]. Use of temporary addresses [RFC8981] is encouraged. To limit pervasive monitoring [RFC7258], a new temporary address **MAY** use a short lifetime (that is, set TEMP_PREFERRED_LIFETIME to be short). Pledges will generally prefer use of IPv6 link-local addresses, and discovery of the proxy will be by link-local mechanisms. IPv4 methods are described in Appendix A.
2. **MUST**: Listen for GRASP M_FLOOD [RFC8990] announcements of the objective: "AN_Proxy". See Section 4.1.1 for the details of the objective. The pledge **MAY** listen concurrently for other sources of information; see Appendix B.

Once a proxy is discovered, the pledge communicates with a registrar through the proxy using the bootstrapping protocol defined in Section 5.

While the GRASP M_FLOOD mechanism is passive for the pledge, the non-normative other methods (mDNS and IPv4 methods) described in Appendix B are active. The pledge **SHOULD** run those methods in parallel with listening for the M_FLOOD. The active methods **SHOULD** back off by doubling to a maximum of one hour to avoid overloading the network with discovery attempts. Detection of physical link status change (Ethernet carrier, for instance) **SHOULD** reset the back-off timers.

The pledge could discover more than one proxy on a given physical interface. The pledge can have a multitude of physical interfaces as well: a Layer 2/3 Ethernet switch may have hundreds of physical ports.

Each possible proxy offer **SHOULD** be attempted up to the point where a valid voucher is received: while there are many ways in which the attempt may fail, it does not succeed until the voucher has been validated.

The connection attempts via a single proxy **SHOULD** exponentially back off to a maximum of one hour to avoid overloading the network infrastructure. The back-off timer for each **MUST** be independent of other connection attempts.

Connection attempts **SHOULD** be run in parallel to avoid head-of-queue problems wherein an attacker running a fake proxy or registrar could intentionally perform protocol actions slowly. Connection attempts to different proxies **SHOULD** be sent with an interval of 3 to 5s. The pledge **SHOULD** continue to listen for additional GRASP M_FLOOD messages during the connection attempts.

Each connection attempt through a distinct Join Proxy **MUST** have a unique nonce in the voucher-request.

Once a connection to a registrar is established (e.g., establishment of a TLS session key), there are expectations of more timely responses; see Section 5.2.

Once all discovered services are attempted (assuming that none succeeded), the device **MUST** return to listening for GRASP M_FLOOD. It **SHOULD** periodically retry any manufacturer-specific mechanisms. The pledge **MAY** prioritize selection order as appropriate for the anticipated environment.

4.1.1. Proxy GRASP Announcements

A proxy uses the DULL GRASP M_FLOOD mechanism to announce itself. This announcement can be within the same message as the ACP announcement detailed in [RFC8994].

The formal Concise Data Definition Language (CDDL) [RFC8610] definition is:

```
\<CODE BEGINS> file "proxygrasp.cddl"

flood-message = [M_FLOOD, session-id, initiator, ttl,
                 +[objective, (locator-option / [])]]

objective = ["AN_Proxy", objective-flags, loop-count,
                                       objective-value]

ttl             = 180000     ; 180,000 ms (3 minutes)
initiator = ACP address to contact registrar
objective-flags   = sync-only  ; as in the GRASP spec
sync-only         =  4         ; M_FLOOD only requires
                               ; synchronization
loop-count        =  1         ; one hop only
objective-value   =  any       ; none

locator-option    = [ O_IPv6_LOCATOR, ipv6-address,
                    transport-proto, port-number ]
ipv6-address      = the v6 LL of the Proxy
$transport-proto /= IPPROTO_TCP   ; note that this can be any value
                                 ; from the IANA protocol registry,
                                 ; as per RFC 8990, Section 2.9.5.1,
                                 ; Note 3.
port-number      = selected by Proxy

\<CODE ENDS>
```

Figure 10: CDDL Definition of Proxy Discovery Message

Here is an example M_FLOOD announcing a proxy at fe80::1, on TCP port 4443.

```
[M_FLOOD, 12340815, h'fe800000000000000000000000000001', 180000,
            [["AN_Proxy", 4, 1, ""],
             [O_IPv6_LOCATOR,
              h'fe800000000000000000000000000001', IPPROTO_TCP,
4443]]]
```

Figure 11: Example of Proxy Discovery Message

On a small network, the registrar **MAY** include the GRASP M_FLOOD announcements to locally connected networks.

The $transport-proto above indicates the method that the pledge-proxy-registrar will use. The TCP method described here is mandatory, and other proxy methods, such as CoAP methods not defined in this document, are optional. Other methods **MUST NOT** be enabled unless the Join Registrar ASA indicates support for them in its own announcement.

4.2. CoAP Connection To Registrar

The use of CoAP to connect from pledge to registrar is out of scope for this document and is described in future work. See [ANIMA-CONSTRAINED-VOUCHER].

4.3. Proxy Discovery And Communication Of Registrar

The registrar **SHOULD** announce itself so that proxies can find it and determine what kind of connections can be terminated.

The registrar announces itself using GRASP M_FLOOD messages, with the "AN_join_registrar" objective, within the ACP instance. A registrar may announce any convenient port number, including use of stock port 443. ANI proxies **MUST** support GRASP discovery of registrars.

The M_FLOOD is formatted as follows:

```
[M_FLOOD, 51804321, h'fda379a6f6ee00000200000064000001', 180000,
            [["AN_join_registrar", 4, 255, "EST-TLS"],
             [O_IPv6_LOCATOR,
              h'fda379a6f6ee00000200000064000001', IPPROTO_TCP,
8443]]]
```

Figure 12: An Example of a Registrar Announcement Message

The formal CDDL definition is:

```
\<CODE BEGINS> file "jrcgrasp.cddl"

flood-message = [M_FLOOD, session-id, initiator, ttl,
                 +[objective, (locator-option / [])]]

objective = ["AN_join_registrar", objective-flags, loop-count,
                                       objective-value]

initiator = ACP address to contact registrar
objective-flags = sync-only  ; as in the GRASP spec
sync-only =  4               ; M_FLOOD only requires
                             ; synchronization
```

```
loop-count      = 255         ; mandatory maximum
objective-value = text        ; name of the (list of) supported
                              ; protocols: "EST-TLS" for RFC 7030.

\<CODE ENDS>
```

Figure 13: CDDL Definition for Registrar Announcement Message

The M_FLOOD message **MUST** be sent periodically. The default period **SHOULD** be 60 seconds, and the value **SHOULD** be operator configurable but **SHOULD NOT** be smaller than 60 seconds. The frequency of sending **MUST** be such that the aggregate amount of periodic M_FLOODs from all flooding sources causes only negligible traffic across the ACP.

Here are some examples of locators for illustrative purposes. Only the first one ($transport-protocol = 6, TCP) is defined in this document and is mandatory to implement.

```
locator1  = [O_IPv6_LOCATOR, fd45:1345::6789, 6,  443]
locator2  = [O_IPv6_LOCATOR, fd45:1345::6789, 17, 5683]
locator3  = [O_IPv6_LOCATOR, fe80::1234, 41, nil]
```

A protocol of 6 indicates that TCP proxying on the indicated port is desired.

Registrars **MUST** announce the set of protocols that they support, and they **MUST** support TCP traffic.

Registrars **MUST** accept HTTPS/EST traffic on the TCP ports indicated.

Registrars **MUST** support the ANI TLS Circuit Proxy and therefore BRSKI across HTTPS/TLS native across the ACP.

In the ANI, the ACP-secured instance of GRASP [RFC8990] **MUST** be used for discovery of ANI registrar ACP addresses and ports by ANI proxies. Therefore, the TCP leg of the proxy connection between the ANI proxy and ANI registrar also runs across the ACP.

5. Protocol Details (Pledge -- Registrar -- MASA)

The pledge **MUST** initiate BRSKI after boot if it is unconfigured. The pledge **MUST NOT** automatically initiate BRSKI if it has been configured or is in the process of being configured.

BRSKI is described as extensions to EST [RFC7030]. The goal of these extensions is to reduce the number of TLS connections and crypto operations required on the pledge. The registrar implements the BRSKI REST interface within the "/.well-known/brski" URI tree and implements the existing EST URIs as described in EST [RFC7030], Section 3.2.2. The communication channel between the pledge and the registrar is referred to as "BRSKI-EST" (see Figure 1).

The communication channel between the registrar and MASA is a new communication channel, similar to EST, within the newly registered "/.well-known/brski" tree. For clarity, this channel is referred to as "BRSKI-MASA" (see Figure 1).

The MASA URI is "https://" authority "/.well-known/brski".

BRSKI uses existing CMS message formats for existing EST operations. BRSKI uses JSON [RFC8259] for all new operations defined here and for voucher formats. In all places where a binary value must be carried in a JSON string, a base64 format ([RFC4648], Section 4) is to be used, as per [RFC7951], Section 6.6.

While EST ([RFC7030], Section 3.2) does not insist upon use of HTTP persistent connections ([RFC7230], Section 6.3), BRSKI-EST connections **SHOULD** use persistent connections. The intention of this guidance is to ensure the provisional TLS state occurs only once, and that the subsequent resolution of the provision state is not subject to a Man-in-the-Middle (MITM) attack during a critical phase.

If non-persistent connections are used, then both the pledge and the registrar **MUST** remember the certificates that have been seen and also sent for the first connection. They **MUST** check each subsequent connection for the same certificates, and each end **MUST** use the same certificates as well. This places a difficult restriction on rolling certificates on the registrar.

Summarized automation extensions for the BRSKI-EST flow are:

- The pledge either attempts concurrent connections via each discovered proxy or times out quickly and tries connections in series, as explained at the end of Section 5.1.
- The pledge provisionally accepts the registrar certificate during the TLS handshake as detailed in Section 5.1.
- The pledge requests a voucher using the new REST calls described below. This voucher is then validated.
- The pledge completes authentication of the server certificate as detailed in Section 5.6.1. This moves the BRSKI-EST TLS connection out of the provisional state.
- Mandatory bootstrap steps conclude with voucher status telemetry (see Section 5.7).

The BRSKI-EST TLS connection can now be used for EST enrollment.

The extensions for a registrar (equivalent to an EST server) are:

- Client authentication is automated using IDevID as per the EST certificate-based client authentication. The subject field's DN encoding **MUST** include the "serialNumber" attribute with the device's unique serial number as explained in Section 2.3.1.
- The registrar requests and validates the voucher from the MASA.
- The registrar forwards the voucher to the pledge when requested.
- The registrar performs log verifications (described in Section 5.8.3) in addition to local authorization checks before accepting optional pledge device enrollment requests.

5.1. BRSKI-EST TLS Establishment Details

The pledge establishes the TLS connection with the registrar through the Circuit Proxy (see Section 4), but the TLS handshake is with the registrar. The BRSKI-EST pledge is the TLS client, and the BRSKI-EST registrar is the TLS server. All security associations established are between the pledge and the registrar regardless of proxy operations.

Use of TLS 1.3 (or newer) is encouraged. TLS 1.2 or newer is **REQUIRED** on the pledge side. TLS 1.3 (or newer) **SHOULD** be available on the registrar server interface, and the registrar client interface, but TLS 1.2 **MAY** be used. TLS 1.3 (or newer) **SHOULD** be available on the MASA server interface, but TLS 1.2 **MAY** be used.

Establishment of the BRSKI-EST TLS connection is as specified in "Bootstrap Distribution of CA Certificates" (Section 4.1.1) of [RFC7030], wherein the client is authenticated with the IDevID certificate, and the EST server (the registrar) is provisionally authenticated with an unverified server certificate. Configuration or distribution of the trust anchor database used for validating the IDevID certificate is out of scope of this specification. Note that the trust anchors in / excluded from the database will affect which manufacturers' devices are acceptable to the registrar as pledges and can also be used to limit the set of MASAs that are trusted for enrollment.

The signature in the certificate **MUST** be validated even if a signing key cannot (yet) be validated. The certificate (or chain) **MUST** be retained for later validation.

A self-signed certificate for the registrar is acceptable as the voucher can validate it upon successful enrollment.

The pledge performs input validation of all data received until a voucher is verified as specified in Section 5.6.1 and the TLS connection leaves the provisional state. Until these operations are complete, the pledge could be communicating with an attacker.

The pledge code needs to be written with the assumption that all data is being transmitted at this point to an unauthenticated peer, and that received data, while inside a TLS connection, **MUST** be considered untrusted. This particularly applies to HTTP headers and CMS structures that make up the voucher.

A pledge that can connect to multiple registrars concurrently **SHOULD** do so. Some devices may be unable to do so for lack of threading, or resource issues. Concurrent connections defeat attempts by a malicious proxy from causing a TCP Slowloris-like attack (see [slowloris]).

A pledge that cannot maintain as many connections as there are eligible proxies will need to rotate among the various choices, terminating connections that do not appear to be making progress. If no connection is making progress after 5 seconds, then the pledge **SHOULD** drop the oldest connection and go on to a different proxy: the proxy that has been communicated with least recently. If there were no other proxies discovered, the pledge **MAY** continue to wait, as long as it is concurrently listening for new proxy announcements.

5.2. Pledge Requests Voucher From The Registrar

When the pledge bootstraps, it makes a request for a voucher from a registrar.

This is done with an HTTPS POST using the operation path value of "/.well-known/brski/requestvoucher".

The pledge voucher-request Content-Type is as follows.

- application/voucher-cms+json:

  [RFC8366] defines a "YANG-defined JSON document that has been signed using a Cryptographic Message Syntax (CMS) structure", and the voucher-request described in Section 3 is created in the same way. The media type is the same as defined in [RFC8366]. This is also used for the pledge voucher-request. The pledge **MUST** sign the request using the credentials in Section 2.3.

Registrar implementations **SHOULD** anticipate future media types but, of course, will simply fail the request if those types are not yet known.

The pledge **SHOULD** include an "Accept" header field (see [RFC7231], Section 5.3.2) indicating the acceptable media type for the voucher response. The "application/voucher-cms+json" media type is defined in [RFC8366], but constrained voucher formats are expected in the future. Registrars and MASA are expected to be flexible in what they accept.

The pledge populates the voucher-request fields as follows:

- created-on:

Pledges that have a real-time clock are **RECOMMENDED** to populate this field with the current date and time in yang:date-and-time format. This provides additional information to the MASA. Pledges that have no real-time clocks **MAY** omit this field.

- nonce:

  The pledge voucher-request **MUST** contain a cryptographically strong random or pseudo-random number nonce (see [RFC4086], Section 6.2). As the nonce is usually generated very early in the boot sequence, there is a concern that the same nonce might be generated across multiple boots, or after a factory reset. Different nonces **MUST** be generated for each bootstrapping attempt, whether in series or concurrently. The freshness of this nonce mitigates against the lack of a real-time clock as explained in Section 2.6.1.

- assertion:

  The pledge indicates support for the mechanism described in this document, by putting the value "proximity" in the voucher-request, and **MUST** include the proximity-registrar-cert field (below).

- proximity-registrar-cert:

  In a pledge voucher-request, this is the first certificate in the TLS server "certificate_list" sequence (see [RFC8446], Section 4.4.2) presented by the registrar to the pledge. That is, it is the end-entity certificate. This **MUST** be populated in a pledge voucher-request.

- serial-number:

  The serial number of the pledge is included in the voucher-request from the pledge. This value is included as a sanity check only, but it is not to be forwarded by the registrar as described in Section 5.5.

All other fields **MAY** be omitted in the pledge voucher-request.

See an example JSON payload of a pledge voucher-request in Section 3.3, Example 1.

The registrar confirms that the assertion is "proximity" and that pinned proximity-registrar-cert is the registrar's certificate. If this validation fails, then there is an on-path attacker (MITM), and the connection **MUST** be closed after the returning of an HTTP 401 error code.

5.3. Registrar Authorization Of Pledge

In a fully automated network, all devices must be securely identified and authorized to join the domain.

A registrar accepts or declines a request to join the domain, based on the authenticated identity presented. For different networks, examples of automated acceptance may include the allowance of:

- any device of a specific type (as determined by the X.509 IDevID),
- any device from a specific vendor (as determined by the X.509 IDevID),
- a specific device from a vendor (as determined by the X.509 IDevID) against a domain acceptlist. (The mechanism for checking a shared acceptlist potentially used by multiple registrars is out of scope.)

If validation fails, the registrar **SHOULD** respond with the HTTP 404 error code. If the voucher-request is in an unknown format, then an HTTP 406 error code is more appropriate. A situation that could be resolved with administrative action (such as adding a vendor to an acceptlist) **MAY** be responded to with a 403 HTTP error code.

If authorization is successful, the registrar obtains a voucher from the MASA service (see Section 5.5) and returns that MASA-signed voucher to the pledge as described in Section 5.6.

5.4. BRSKI-MASA TLS Establishment Details

The BRSKI-MASA TLS connection is a "normal" TLS connection appropriate for HTTPS REST interfaces. The registrar initiates the connection and uses the MASA URL that is obtained as described in Section 2.8. The mechanisms in [RFC6125] **SHOULD** be used in authentication of the MASA using a DNS-ID that matches that which is found in the IDevID. Registrars **MAY** include a mechanism to override the MASA URL on a manufacturer-by-manufacturer basis, and within that override, it is appropriate to provide alternate anchors. This will typically be used by some vendors to establish explicit (or private) trust anchors for validating their MASA that is part of a sales channel integration.

Use of TLS 1.3 (or newer) is encouraged. TLS 1.2 or newer is **REQUIRED**. TLS 1.3 (or newer) **SHOULD** be available.

As described in [RFC7030], the MASA and the registrars **SHOULD** be prepared to support TLS Client Certificate authentication and/or HTTP Basic, Digest, or Salted Challenge Response Authentication Mechanism (SCRAM) authentication. This connection **MAY** also have no client authentication at all.

Registrars **SHOULD** permit trust anchors to be preconfigured on a per-vendor (MASA) basis. Registrars **SHOULD** include the ability to configure a TLS Client Certificate on a per-MASA basis, or to use no Client Certificate. Registrars **SHOULD** also permit HTTP Basic and Digest authentication to be configured.

The authentication of the BRSKI-MASA connection does not change the voucher-request process, as voucher-requests are already signed by the registrar. Instead, this authentication provides access control to the audit-log as described in Section 5.8.

Implementers are advised that contacting the MASA establishes a secured API connection with a web service, and that there are a number of authentication models being explored within the industry. Registrars are **RECOMMENDED** to fail gracefully and generate useful administrative notifications or logs in the advent of unexpected HTTP 401 (Unauthorized) responses from the MASA.

5.4.1. MASA Authentication of Customer Registrar

Providing per-customer options requires the customer's registrar to be uniquely identified. This can be done by any stateless method that HTTPS supports such as HTTP Basic or Digest authentication (that is using a password), but the use of TLS Client Certificate authentication is **RECOMMENDED**.

Stateful methods involving API tokens, or HTTP Cookies, are not recommended.

It is expected that the setup and configuration of per-customer Client Certificates is done as part of a sales ordering process.

The use of public PKI (i.e., WebPKI) end-entity certificates to identify the registrar is reasonable, and if done universally, this would permit a MASA to identify a customer's registrar simply by a Fully Qualified Domain Name (FQDN).

The use of DANE records in DNSSEC-signed zones would also permit use of a FQDN to identify customer registrars.

A third (and simplest, but least flexible) mechanism would be for the MASA to simply store the registrar's certificate pinned in a database.

A MASA without any supply-chain integration can simply accept registrars without any authentication or on a blind TOFU basis as described in Section 7.4.2.

This document does not make a specific recommendation on how the MASA authenticates the registrar as there are likely different tradeoffs in different environments and product values. Even within the ANIMA ACP applicability, there is a significant difference between supply-chain logistics for $100 CPE devices and $100,000 core routers.

5.5. Registrar Requests Voucher From MASA

When a registrar receives a pledge voucher-request, it in turn submits a registrar voucher-request to the MASA service via an HTTPS interface [RFC7231].

This is done with an HTTP POST using the operation path value of "/.well-known/brski/requestvoucher".

The voucher media type "application/voucher-cms+json" is defined in [RFC8366] and is also used for the registrar voucher-request. It is a JSON document that has been signed using a CMS structure. The registrar **MUST** sign the registrar voucher-request.

MASA implementations **SHOULD** anticipate future media ntypes but, of course, will simply fail the request if those types are not yet known.

The voucher-request CMS object includes some number of certificates that are input to the MASA as it populates the pinned-domain-cert. As [RFC8366] is quite flexible in what may be put into the pinned-domain-cert, the MASA needs some signal as to what certificate would be effective to populate the field with: it may range from the end-entity certificate that the registrar uses to the entire private Enterprise CA certificate. More-specific certificates result in a tighter binding of the voucher to the domain, while less-specific certificates result in more flexibility in how the domain is represented by certificates.

A registrar that is seeking a nonceless voucher for later offline use benefits from a less-specific certificate, as it permits the actual key pair used by a future registrar to be determined by the pinned CA.

In some cases, a less-specific certificate, such as a public WebPKI CA, could be too open and could permit any entity issued a certificate by that authority to assume ownership of a device that has a voucher pinned. Future work may provide a solution to pin both a certificate and a name that would reduce such risk of malicious ownership assertions.

The registrar **SHOULD** request a voucher with the most specificity consistent with the mode that it is operating in. In order to do this, when the registrar prepares the CMS structure for the signed voucher-request, it **SHOULD** include only certificates that are a part of the chain that it wishes the MASA to pin. This **MAY** be as small as only the end-entity certificate (with id-kp-cmcRA set) that it uses as its TLS server certificate, or it **MAY** be the entire chain, including the domain CA.

The registrar **SHOULD** include an "Accept" header field (see [RFC7231], Section 5.3.2) indicating the response media types that are acceptable. This list **SHOULD** be the entire list presented to the registrar in the pledge's original request (see Section 5.2), but it **MAY** be a subset. The MASA is expected to be flexible in what it accepts.

The registrar populates the voucher-request fields as follows:

- created-on:

  The registrar **SHOULD** populate this field with the current date and time when the voucher-request is formed. This field provides additional information to the MASA.

- nonce:

  This value, if present, is copied from the pledge voucher-request. The registrar voucher-request **MAY** omit the nonce as per Section 3.1.

- serial-number:

  The serial number of the pledge the registrar would like a voucher for. The registrar determines this value by parsing the authenticated pledge IDevID certificate; see Section 2.3. The registrar **MUST** verify that the serial-number field it parsed matches the serial-number field the pledge provided in its voucher-request. This provides a sanity check useful for detecting error conditions and logging. The registrar **MUST NOT** simply copy the serial-number field from a pledge voucher-request as that field is claimed but not certified.

- idevid-issuer:

  The Issuer value from the pledge IDevID certificate is included to ensure unique interpretation of the serial-number. In the case of a nonceless (offline) voucher-request, an appropriate value needs to be configured from the same out-of-band source as the serial-number.

- prior-signed-voucher-request:

  The signed pledge voucher-request **SHOULD** be included in the registrar voucher-request. The entire CMS-signed structure is to be included and base64 encoded for transport in the JSON structure.

A nonceless registrar voucher-request **MAY** be submitted to the MASA. Doing so allows the registrar to request a voucher when the pledge is offline, or when the registrar anticipates not being able to connect to the MASA while the pledge is being deployed. Some use cases require the registrar to learn the appropriate IDevID serialNumber field and appropriate "Accept" header field values from the physical device labeling or from the sales channel (which is out of scope for this document).

All other fields **MAY** be omitted in the registrar voucher-request.

The proximity-registrar-cert field **MUST NOT** be present in the registrar voucher-request.

See example JSON payloads of registrar voucher-requests in Section 3.3, Examples 2 through 4.

The MASA verifies that the registrar voucher-request is internally consistent but does not necessarily authenticate the registrar certificate since the registrar **MAY** be unknown to the MASA in advance. The MASA performs the actions and validation checks described in the following subsections before issuing a voucher.

5.5.1. MASA Renewal of Expired Vouchers

As described in [RFC8366], vouchers are normally short lived to avoid revocation issues. If the request is for a previous (expired) voucher using the same registrar (that is, a registrar with the same domain CA), then the request for a renewed voucher **SHOULD** be automatically authorized. The MASA has sufficient information to determine this by examining the request, the registrar authentication, and the existing audit-log. The issuance of a renewed voucher is logged as detailed in Section 5.6.

To inform the MASA that existing vouchers are not to be renewed, one can update or revoke the registrar credentials used to authorize the request (see Sections 5.5.4 and 5.5.3). More flexible methods will likely involve sales channel integration and authorizations (details are out of scope of this document).

5.5.2. MASA Pinning of Registrar

A certificate chain is extracted from the registrar's signed CMS container. This chain may be as short as a single end-entity certificate, up to the entire registrar certificate chain, including the domain CA certificate, as specified in Section 5.5.

If the domain's CA is unknown to the MASA, then it is considered a temporary trust anchor for the rest of the steps in this section. The intention is not to authenticate the message as having come from a fully validated origin but to establish the consistency of the domain PKI.

The MASA **MAY** use the certificate in the chain that is farthest from the end-entity certificate of the registrar, as determined by MASA policy. A MASA **MAY** have a local policy in which it only pins the end-entity certificate. This is consistent with [RFC8366]. Details of the policy will typically depend upon the degree of supply-chain integration and the mechanism used by the registrar to authenticate. Such a policy would also determine how the MASA will respond to a request for a nonceless voucher.

5.5.3. MASA Check of the Voucher-Request Signature

As described in Section 5.5.2, the MASA has extracted the registrar's domain CA. This is used to validate the CMS signature [RFC5652] on the voucher-request.

Normal PKIX revocation checking is assumed during voucher-request signature validation. This CA certificate **MAY** have Certificate Revocation List (CRL) distribution points or Online Certificate Status Protocol (OCSP) information [RFC6960]. If they are present, the MASA **MUST** be able to reach the relevant servers belonging to the registrar's domain CA to perform the revocation checks.

The use of OCSP Stapling is preferred.

5.5.4. MASA Verification of the Domain Registrar

The MASA **MUST** verify that the registrar voucher-request is signed by a registrar. This is confirmed by verifying that the id-kp-cmcRA extended key usage extension field (as detailed in EST [RFC7030], Section 3.6.1) exists in the certificate of the entity that signed the registrar voucher-request. This verification is only a consistency check to ensure that the unauthenticated domain CA intended the voucher-request signer to be a registrar. Performing this check provides value to the domain PKI by assuring the domain administrator that the MASA service will only respect claims from authorized registration authorities of the domain.

Even when a domain CA is authenticated to the MASA, and there is strong sales channel integration to understand who the legitimate owner is, the above id-kp-cmcRA check prevents arbitrary end-entity certificates (such as an LDevID certificate) from having vouchers issued against them.

Other cases of inappropriate voucher issuance are detected by examination of the audit-log.

If a nonceless voucher-request is submitted, the MASA **MUST** authenticate the registrar either as described in EST (see Sections 3.2.3 and 3.3.2 of [RFC7030]) or by validating the registrar's certificate used to sign the registrar voucher-request using a configured trust anchor. Any of these methods reduce the risk of DDoS attacks and provide an authenticated identity as an input to sales channel integration and authorizations (details are out of scope of this document).

In the nonced case, validation of the registrar's identity (via TLS Client Certificate or HTTP authentication) **MAY** be omitted if the MASA knows that the device policy is to accept audit-only vouchers.

5.5.5. MASA Verification of the Pledge 'prior-signed-voucher-request'

The MASA **MAY** verify that the registrar voucher-request includes the prior-signed-voucher-request field. If so, the prior-signed-voucher-request **MUST** include a proximity-registrar-cert that is consistent with the certificate used to sign the registrar voucher-request. Additionally, the voucher-request serial-number leaf **MUST** match the pledge serial-number that the MASA extracts from the signing certificate of the prior-signed-voucher-request. The consistency check described above entails checking that the proximity-registrar-cert Subject Public Key Info (SPKI) Fingerprint exists within the registrar voucher-request CMS signature's certificate chain. This is substantially the same as the pin validation described in [RFC7469], Section 2.6.

If these checks succeed, the MASA updates the voucher and audit-log assertion leafs with the "proximity" assertion, as defined by [RFC8366], Section 5.3.

5.5.6. MASA Nonce Handling

The MASA does not verify the nonce itself. If the registrar voucher-request contains a nonce, and the prior-signed-voucher-request exists, then the MASA **MUST** verify that the nonce is consistent. (Recall from above that the voucher-request might not contain a nonce; see Sections 5.5 and 5.5.4.)

The MASA populates the audit-log with the nonce that was verified. If a nonceless voucher is issued, then the audit-log is to be populated with the JSON value "null".

5.6. MASA And Registrar Voucher Response

The MASA voucher response to the registrar is forwarded without changes to the pledge; therefore, this section applies to both the MASA and the registrar. The HTTP signaling described applies to both the MASA and registrar responses.

When a voucher-request arrives at the registrar, if it has a cached response from the MASA for the corresponding registrar voucher-request, that cached response can be used according to local policy; otherwise, the registrar constructs a new registrar voucher-request and sends it to the MASA.

Registrar evaluation of the voucher itself is purely for transparency and audit purposes to further inform log verification (see Section 5.8.3); therefore, a registrar could accept future voucher formats that are opaque to the registrar.

If the voucher-request is successful, the server (a MASA responding to a registrar or a registrar responding to a pledge) response **MUST** contain an HTTP 200 response code. The server **MUST** answer with a suitable 4xx or 5xx HTTP [RFC7230] error code when a problem occurs. In this case, the response data from the MASA **MUST** be a plain text human-readable (UTF-8) error message containing explanatory information describing why the request was rejected.

The registrar **MAY** respond with an HTTP 202 ("the request has been accepted for processing, but the processing has not been completed") as described in EST [RFC7030], Section 4.2.3, wherein the client "**MUST** wait at least the specified "retry-after" time before repeating the same request" (also see [RFC7231], Section 6.6.4). The pledge is **RECOMMENDED** to provide local feedback (blinked LED, etc.) during this wait cycle if mechanisms for this are available. To prevent an attacker registrar from significantly delaying bootstrapping, the pledge **MUST** limit the Retry-After time to 60 seconds. Ideally, the pledge would keep track of the appropriate Retry-After header field values for any number of outstanding registrars, but this would involve a state table on the pledge. Instead, the pledge **MAY** ignore the exact Retry-After value in favor of a single hard-coded value (a registrar that is unable to complete the transaction after the first 60 seconds has another chance a minute later). A pledge **SHOULD** be willing to maintain a 202 retry-state for up to 4 days, which is longer than a long weekend, after which time the enrollment attempt fails, and the pledge returns to Discovery state. This allows time for an alert to get from the registrar to a human operator who can make a decision as to whether or not to proceed with the enrollment.

A pledge that retries a request after receiving a 202 message **MUST** resend the same voucher-request. It **MUST NOT** sign a new voucher-request each time, and in particular, it **MUST NOT** change the nonce value.

In order to avoid infinite redirect loops, which a malicious registrar might do in order to keep the pledge from discovering the correct registrar, the pledge **MUST NOT** follow more than one redirection (3xx code) to another web origin. EST supports redirection but requires user input; this change allows the pledge to follow a single redirection without a user interaction.

A 403 (Forbidden) response is appropriate if the voucher-request is not signed correctly or is stale or if the pledge has another outstanding voucher that cannot be overridden.

A 404 (Not Found) response is appropriate when the request is for a device that is not known to the MASA.

A 406 (Not Acceptable) response is appropriate if a voucher of the desired type or that uses the desired algorithms (as indicated by the "Accept" header fields and algorithms used in the signature) cannot be issued as such because the MASA knows the pledge cannot process that type. The registrar **SHOULD** use this response if it determines the pledge is unacceptable due to inventory control, MASA audit-logs, or any other reason.

A 415 (Unsupported Media Type) response is appropriate for a request that has a voucher-request or "Accept" value that is not understood.

The voucher response format is as indicated in the submitted "Accept" header fields or based on the MASA's prior understanding of proper format for this pledge. Only the "application/voucher-cms+json" media type [RFC8366] is defined at this time. The syntactic details of vouchers are described in detail in [RFC8366]. Figure 14 shows a sample of the contents of a voucher.

```
{
  "ietf-voucher:voucher": {
    "nonce": "62a2e7693d82fcda2624de58fb6722e5",
    "assertion": "logged",
    "pinned-domain-cert": "base64encodedvalue==",
    "serial-number": "JADA123456789"
  }
}
```

Figure 14: An Example Voucher

The MASA populates the voucher fields as follows:

- nonce:

  The nonce from the pledge if available. See Section 5.5.6.

- assertion:

  The method used to verify the relationship between the pledge and registrar. See Section 5.5.5.

- pinned-domain-cert:

  A certificate; see Section 5.5.2. This figure is illustrative; for an example, see Appendix C.2 where an end-entity certificate is used.

- serial-number:

  The serial-number as provided in the voucher-request. Also see Section 5.5.5.

- domain-cert-revocation-checks:

  Set as appropriate for the pledge's capabilities and as documented in [RFC8366]. The MASA **MAY** set this field to "false" since setting it to "true" would require that revocation information be available to the pledge, and this document does not make normative requirements for [RFC6961], Section 4.4.2.1 of [RFC8446], or equivalent integrations.

- expires-on:

  This is set for nonceless vouchers. The MASA ensures the voucher lifetime is consistent with any revocation or pinned-domain-cert consistency checks the pledge might perform. See Section 2.6.1. There are three times to consider: (a) a configured voucher lifetime in the MASA, (b) the expiry time for the registrar's certificate, and (c) any CRL lifetime. The expires-on field **SHOULD** be before the earliest of these three values. Typically, (b) will be some significant time in the future, but (c) will typically be short (on the order of a week or less). The **RECOMMENDED** period for (a) is on the order of 20 minutes, so it will typically determine the life span of the resulting voucher. 20 minutes is sufficient time to reach the post-provisional state in the pledge, at which point there is an established trust relationship between the pledge and registrar. The subsequent operations can take as long as required from that point onwards. The lifetime of the voucher has no impact on the life span of the ownership relationship.

Whenever a voucher is issued, the MASA **MUST** update the audit-log sufficiently to generate the response as described in Section 5.8.1. The internal state requirements to maintain the audit-log are out of scope.

5.6.1. Pledge Voucher Verification

The pledge **MUST** verify the voucher signature using the manufacturer-installed trust anchor(s) associated with the manufacturer's MASA (this is likely included in the pledge's firmware). Management of the manufacturer-installed trust anchor(s) is out of scope of this document; this protocol does not update this trust anchor(s).

The pledge **MUST** verify that the serial-number field of the signed voucher matches the pledge's own serial-number.

The pledge **MUST** verify the nonce information in the voucher. If present, the nonce in the voucher must match the nonce the pledge submitted to the registrar; vouchers with no nonce can also be accepted (according to local policy; see Section 7.2).

The pledge **MUST** be prepared to parse and fail gracefully from a voucher response that does not contain a pinned-domain-cert field. Such a thing indicates a failure to enroll in this domain, and the pledge **MUST** attempt joining with other available Join Proxies.

The pledge **MUST** be prepared to ignore additional fields that it does not recognize.

5.6.2. Pledge Authentication of Provisional TLS Connection

Following the process described in [RFC8366], the pledge should consider the public key from the pinned-domain-cert as the sole temporary trust anchor.

The pledge then evaluates the TLS server certificate chain that it received when the TLS connection was formed using this trust anchor. It is possible that the public key in the pinned-domain-cert directly matches the public key in the end-entity certificate provided by the TLS server.

If a registrar's credentials cannot be verified using the pinned-domain-cert trust anchor from the voucher, then the TLS connection is discarded, and the pledge abandons attempts to bootstrap with this discovered registrar. The pledge **SHOULD** send voucher status telemetry (described below) before closing the TLS connection. The pledge **MUST** attempt to enroll using any other proxies it has found. It **SHOULD** return to the same proxy again after unsuccessful attempts with other proxies. Attempts should be made at repeated intervals according to the back-off timer described earlier. Attempts **SHOULD** be repeated as failure may be the result of a temporary inconsistency (an inconsistently rolled registrar key, or some other misconfiguration). The inconsistency could also be the result of an active MITM attack on the EST connection.

The registrar **MUST** use a certificate that chains to the pinned-domain-cert as its TLS server certificate.

The pledge's PKIX path validation of a registrar certificate's validity period information is as described in Section 2.6.1. Once the PKIX path validation is successful, the TLS connection is no longer provisional.

The pinned-domain-cert **MAY** be installed as a trust anchor for future operations such as enrollment (e.g., as recommended per [RFC7030]) or trust anchor management or raw protocols that do not need full PKI-based key management. It can be used to authenticate any dynamically discovered EST server that contains the id-kp-cmcRA extended key usage extension as detailed in EST (see [RFC7030], Section 3.6.1); but to reduce system complexity, the pledge **SHOULD** avoid additional discovery operations. Instead, the pledge **SHOULD** communicate directly with the registrar as the EST server. The pinned-domain-cert is not a complete distribution of the CA certificate response, as described in [RFC7030], Section 4.1.3, which is an additional justification for the recommendation to proceed with EST key management operations. Once a full CA certificate response is obtained, it is more authoritative for the domain than the limited pinned-domain-cert response.

5.7. Pledge BRSKI Status Telemetry

The domain is expected to provide indications to the system administrators concerning device life-cycle status. To facilitate this, it needs telemetry information concerning the device's status.

The pledge **MUST** indicate its pledge status regarding the voucher. It does this by sending a status message to the registrar.

The posted data media type: application/json

The client sends an HTTP POST to the server at the URI ".well-known/brski/voucher_status".

The format and semantics described below are for version 1. A version field is included to permit significant changes to this feedback in the future. A registrar that receives a status message with a version larger than it knows about **SHOULD** log the contents and alert a human.

The status field indicates if the voucher was acceptable. Boolean values are acceptable, where "true" indicates the voucher was acceptable.

If the voucher was not acceptable, the Reason string indicates why. In a failure case, this message may be sent to an unauthenticated, potentially malicious registrar; therefore, the Reason string **SHOULD NOT** provide information beneficial to an attacker. The operational benefit of this telemetry information is balanced against the operational costs of not recording that a voucher was ignored by a client that the registrar expected was going to continue joining the domain.

The reason-context attribute is an arbitrary JSON object (literal value or hash of values) that provides additional information specific to this pledge. The contents of this field are not subject to standardization.

The version and status fields **MUST** be present. The Reason field **SHOULD** be present whenever the status field is false. The Reason-Context field is optional. In the case of a SUCCESS, the Reason string **MAY** be omitted.

The keys to this JSON object are case sensitive and **MUST** be lowercase. Figure 16 shows an example JSON.

```
\<CODE BEGINS> file "voucherstatus.cddl"

voucherstatus-post = {
    "version": uint,
    "status": bool,
    ? "reason": text,
    ? "reason-context" : { $$arbitrary-map }
  }
}

\<CODE ENDS>
```

Figure 15: CDDL for Voucher Status POST

```
{
    "version": 1,
    "status":false,
    "reason":"Informative human-readable message",
    "reason-context": { "additional" : "JSON" }
}
```

Figure 16: Example Status Telemetry

The server **SHOULD** respond with an HTTP 200 but **MAY** simply fail with an HTTP 404 error. The client ignores any response. The server **SHOULD** capture this telemetry information within the server logs.

Additional standard JSON fields in this POST **MAY** be added; see Section 8.5. A server that sees unknown fields should log them, but otherwise ignore them.

5.8. Registrar Audit-Log Request

After receiving the pledge status telemetry (see Section 5.7), the registrar **SHOULD** request the MASA audit-log from the MASA service.

This is done with an HTTP POST using the operation path value of "/.well-known/brski/requestauditlog".

The registrar **SHOULD** HTTP POST the same registrar voucher-request as it did when requesting a voucher (using the same Content-Type). It is posted to the /requestauditlog URI instead. The idevid-issuer and serial-number informs the MASA which log is requested, so the appropriate log can be prepared for the response. Using the same media type and message minimizes cryptographic and message operations, although it results in additional network traffic. The relying MASA implementation **MAY** leverage internal state to associate this request with the original, and by now already validated, voucher-request so as to avoid an extra crypto validation.

A registrar **MAY** request logs at future times. If the registrar generates a new request, then the MASA is forced to perform the additional cryptographic operations to verify the new request.

A MASA that receives a request for a device that does not exist, or for which the requesting owner was never an owner, returns an HTTP 404 ("Not found") code.

It is reasonable for a registrar, that the MASA does not believe to be the current owner, to request the audit-log. There are probably reasons for this, which are hard to predict in advance. For instance, such a registrar may not be aware that the device has been resold; it may be that the device has been resold inappropriately, and this is how the original owner will learn of the occurrence. It is also possible that the device legitimately spends time in two different networks.

Rather than returning the audit-log as a response to the POST (with a return code 200), the MASA **MAY** instead return a 201 ("Created") response ([RFC7231], Sections 6.3.2 and 7.1), with the URL to the prepared (and idempotent, therefore cachable) audit response in the "Location" header field.

In order to avoid enumeration of device audit-logs, a MASA that returns URLs **SHOULD** take care to make the returned URL unguessable. [W3C.capability-urls] provides very good additional guidance. For instance, rather than returning URLs containing a database number such as https://example.com/auditlog/1234 or the Extended Unique Identifier (EUI) of the device such https://example.com/auditlog/10-00-00-11-22-33, the MASA **SHOULD** return a randomly generated value (a "slug" in web parlance). The value is used to find the relevant database entry.

A MASA that returns a code 200 **MAY** also include a "Location" header for future reference by the registrar.

5.8.1. MASA Audit-Log Response

A log data file is returned consisting of all log entries associated with the device selected by the IDevID presented in the request. The audit-log may be abridged by removal of old or repeated values as explained below. The returned data is in JSON format [RFC8259], and the Content-Type **SHOULD** be "application/json".

The following CDDL [RFC8610] explains the structure of the JSON format audit-log response:

```
\<CODE BEGINS> file "auditlog.cddl"

audit-log-response = {
  "version": uint,
  "events": [ + event ]
  "truncation": {
    ? "nonced duplicates": uint,
    ? "nonceless duplicates": uint,
    ? "arbitrary": uint,
  }
}

event = {
  "date": text,
  "domainID": text,
  "nonce": text / null,
  "assertion": "verified" / "logged" / "proximity",
  ? "truncated": uint,
}

\<CODE ENDS>
```

Figure 17: CDDL for Audit-Log Response

An example:

```
{
  "version":"1",
  "events":[
    {
        "date":"2019-05-15T17:25:55.644-04:00",
        "domainID":"BduJhdHPpfhQLyponf48JzXSGZ8=",
        "nonce":"VOUFT-WwrEv0NuAQEHoV7Q",
        "assertion":"proximity",
        "truncated":"0"
    },
    {
        "date":"2017-05-15T17:25:55.644-04:00",
        "domainID":"BduJhdHPpfhQLyponf48JzXSGZ8=",
        "nonce":"f4G6Vi1t8nKo/FieCVgpBg==",
        "assertion":"proximity"
    }
  ],
    "truncation": {
        "nonced duplicates": "0",
        "nonceless duplicates": "1",
        "arbitrary": "2"
     }
}
```

Figure 18: Example of an Audit-Log Response

The domainID is a binary SubjectKeyIdentifier value calculated according to Section 5.8.2. It is encoded once in base64 in order to be transported in this JSON container.

The date is formatted per [RFC3339], which is consistent with typical JavaScript usage of JSON.

The truncation structure **MAY** be omitted if all values are zero. Any counter missing from the truncation structure is assumed to be zero.

The nonce is a string, as provided in the voucher-request, and is used in the voucher. If no nonce was placed in the resulting voucher, then a value of null **SHOULD** be used in preference to omitting the entry. While the nonce is often created as a base64-encoded random series of bytes, this should not be assumed.

Distribution of a large log is less than ideal. This structure can be optimized as follows: nonced or nonceless entries for the same domainID **MAY** be abridged from the log leaving only the single most recent nonced or nonceless entry for that domainID. In the case of truncation, the "event" truncation value **SHOULD** contain a count of the number of events for this domainID that were omitted. The log **SHOULD NOT** be further reduced, but an operational situation could exist where maintaining the full log is not possible. In such situations, the log **MAY** be arbitrarily abridged for length, with the number of removed entries indicated as "arbitrary".

If the truncation count exceeds 1024, then the MASA **MAY** use this value without further incrementing it.

A log where duplicate entries for the same domain have been omitted ("nonced duplicates" and/or "nonceless duplicates") could still be acceptable for informed decisions. A log that has had "arbitrary" truncations is less acceptable, but manufacturer transparency is better than hidden truncations.

A registrar that sees a version value greater than 1 indicates an audit-log format that has been enhanced with additional information. No information will be removed in future versions; should an incompatible change be desired in the future, then a new HTTP endpoint will be used.

This document specifies a simple log format as provided by the MASA service to the registrar. This format could be improved by distributed consensus technologies that integrate vouchers with technologies such as block-chain or hash trees or optimized logging approaches. Doing so is out of the scope of this document but is an anticipated improvement for future work. As such, the registrar **SHOULD** anticipate new kinds of responses and **SHOULD** provide operator controls to indicate how to process unknown responses.

5.8.2. Calculation of domainID

The domainID is a binary value (a BIT STRING) that uniquely identifies a registrar by the pinned-domain-cert.

If the pinned-domain-cert certificate includes the SubjectKeyIdentifier ([RFC5280], Section 4.2.1.2), then it is used as the domainID. If not, the SPKI Fingerprint as described in [RFC7469], Section 2.4 is used. This value needs to be calculated by both the MASA (to populate the audit-log) and the registrar (to recognize itself in the audit-log).

[RFC5280], Section 4.2.1.2 does not mandate that the SubjectKeyIdentifier extension be present in non-CA certificates. It is **RECOMMENDED** that registrar certificates (even if self-signed) always include the SubjectKeyIdentifier to be used as a domainID.

The domainID is determined from the certificate chain associated with the pinned-domain-cert and is used to update the audit-log.

5.8.3. Registrar Audit-Log Verification

Each time the MASA issues a voucher, it appends details of the assignment to an internal audit-log for that device. The internal audit-log is processed when responding to requests for details as described in Section 5.8. The contents of the audit-log can express a variety of trust levels, and this section explains what kind of trust a registrar can derive from the entries.

While the audit-log provides a list of vouchers that were issued by the MASA, the vouchers are issued in response to voucher-requests, and it is the content of the voucher-requests that determines how meaningful the audit-log entries are.

A registrar **SHOULD** use the log information to make an informed decision regarding the continued bootstrapping of the pledge. The exact policy is out of scope of this document as it depends on the security requirements within the registrar domain. Equipment that is purchased preowned can be expected to have an extensive history. The following discussion is provided to help explain the value of each log element:

- date:

  The date field provides the registrar an opportunity to divide the log around known events such as the purchase date. Depending on the context known to the registrar or administrator, events before/after certain dates can have different levels of importance. For example, for equipment that is expected to be new, and thus has no history, it would be a surprise to find prior entries.

- domainID:

  If the log includes an unexpected domainID, then the pledge could have imprinted on an unexpected domain. The registrar can be expected to use a variety of techniques to define "unexpected" ranging from acceptlists of prior domains to anomaly detection (e.g., "this device was previously bound to a different domain than any other device deployed"). Log entries can also be compared against local history logs in search of discrepancies (e.g., "this device was re-deployed some number of times internally, but the external audit-log shows additional re-deployments our internal logs are unaware of").

- nonce:

  Nonceless entries mean the logged domainID could theoretically trigger a reset of the pledge and then take over management by using the existing nonceless voucher.

- assertion:

  The assertion leaf in the voucher and audit-log indicates why the MASA issued the voucher. A "verified" entry means that the MASA issued the associated voucher as a result of positive verification of ownership. However, this entry does not indicate whether or not the pledge was actually deployed in the prior domain. A "logged" assertion informs the registrar that the prior vouchers were issued with minimal verification. A "proximity" assertion assures the registrar that the pledge was truly communicating with the prior domain and thus provides assurance that the prior domain really has deployed the pledge.

A relatively simple policy is to acceptlist known (internal or external) domainIDs and require all vouchers to have a nonce. An alternative is to require that all nonceless vouchers be from a subset (e.g., only internal) of domainIDs. If the policy is violated, a simple action is to revoke any locally issued credentials for the pledge in question or to refuse to forward the voucher. The registrar **MUST** then refuse any EST actions and **SHOULD** inform a human via a log. A registrar **MAY** be configured to ignore (i.e., override the above policy) the history of the device, but it is **RECOMMENDED** that this only be configured if hardware-assisted (i.e., Transport Performance Metrics (TPM) anchored) Network Endpoint Assessment (NEA) [RFC5209] is supported.

5.9. EST Integration For PKI Bootstrapping

The pledge **SHOULD** follow the BRSKI operations with EST enrollment operations including "CA Certificates Request", "CSR Attributes Request", and "Client Certificate Request" or "Server-Side Key Generation", etc. This is a relatively seamless integration since BRSKI API calls provide an automated alternative to the manual bootstrapping method described in [RFC7030]. As noted above, use of HTTP-persistent connections simplifies the pledge state machine.

Although EST allows clients to obtain multiple certificates by sending multiple Certificate Signing Requests (CSRs), BRSKI does not support this mechanism directly. This is because BRSKI pledges **MUST** use the CSR Attributes request ([RFC7030], Section 4.5). The registrar **MUST** validate the CSR against the expected attributes. This implies that client requests will "look the same" and therefore result in a single logical certificate being issued even if the client were to make multiple requests. Registrars **MAY** contain more complex logic, but doing so is out of scope of this specification. BRSKI does not signal any enhancement or restriction to this capability.

5.9.1. EST Distribution of CA Certificates

The pledge **SHOULD** request the full EST Distribution of CA certificate messages; see [RFC7030], Section 4.1.

This ensures that the pledge has the complete set of current CA certificates beyond the pinned-domain-cert (see Section 5.6.2 for a discussion of the limitations inherent in having a single certificate instead of a full CA certificate response). Although these limitations are acceptable during initial bootstrapping, they are not appropriate for ongoing PKIX end-entity certificate validation.

5.9.2. EST CSR Attributes

Automated bootstrapping occurs without local administrative configuration of the pledge. In some deployments, it is plausible that the pledge generates a certificate request containing only identity information known to the pledge (essentially the X.509 IDevID information) and ultimately receives a certificate containing domain-specific identity information. Conceptually, the CA has complete control over all fields issued in the end-entity certificate. Realistically, this is operationally difficult with the current status of PKI CA deployments, where the CSR is submitted to the CA via a number of non-standard protocols. Even with all standardized protocols used, it could operationally be problematic to expect that service-specific certificate fields can be created by a CA that is likely operated by a group that has no insight into different network services/protocols used. For example, the CA could even be outsourced.

To alleviate these operational difficulties, the pledge **MUST** request the EST "CSR Attributes" from the EST server, and the EST server needs to be able to reply with the attributes necessary for use of the certificate in its intended protocols/services. This approach allows for minimal CA integrations, and instead, the local infrastructure (EST server) informs the pledge of the proper fields to include in the generated CSR (such as rfc822Name). This approach is beneficial to automated bootstrapping in the widest number of environments.

In networks using the BRSKI enrolled certificate to authenticate the ACP, the EST CSR Attributes **MUST** include the ACP domain information fields defined in [RFC8994], Section 6.2.2.

The registrar **MUST** also confirm that the resulting CSR is formatted as indicated before forwarding the request to a CA. If the registrar is communicating with the CA using a protocol such as full Certificate Management over CMS (CMC), which provides mechanisms to override the CSR Attributes, then these mechanisms **MAY** be used even if the client ignores the guidance for the CSR Attributes.

5.9.3. EST Client Certificate Request

The pledge **MUST** request a new Client Certificate; see [RFC7030], Section 4.2.

5.9.4. Enrollment Status Telemetry

For automated bootstrapping of devices, the administrative elements that provide bootstrapping also provide indications to the system administrators concerning device life-cycle status. This might include information concerning attempted bootstrapping messages seen by the client. The MASA provides logs and the status of credential enrollment. Since an end user is assumed per [RFC7030], a final success indication back to the server is not included. This is insufficient for automated use cases.

The client **MUST** send an indicator to the registrar about its enrollment status. It does this by using an HTTP POST of a JSON dictionary with the attributes described below to the new EST endpoint at "/.well-known/brski/enrollstatus".

When indicating a successful enrollment, the client **SHOULD** first re-establish the EST TLS session using the newly obtained credentials. TLS 1.3 supports doing this in-band, but TLS 1.2 does not. The client **SHOULD** therefore always close the existing TLS connection and start a new one, using the same Join Proxy.

In the case of a failed enrollment, the client **MUST** send the telemetry information over the same TLS connection that was used for the enrollment attempt, with a Reason string indicating why the most recent enrollment failed. (For failed attempts, the TLS connection is the most reliable way to correlate server-side information with what the client provides.)

The version and status fields **MUST** be present. The Reason field **SHOULD** be present whenever the status field is false. In the case of a SUCCESS, the Reason string **MAY** be omitted.

The reason-context attribute is an arbitrary JSON object (literal value or hash of values) that provides additional information specific to the failure to unroll from this pledge. The contents of this field are not subject to standardization. This is represented by the group-socket "$$arbitrary-map" in the CDDL.

```
\<CODE BEGINS> file "enrollstatus.cddl"

enrollstatus-post = {
    "version": uint,
    "status": bool,
    ? "reason": text,
    ? "reason-context" : { $$arbitrary-map }
  }
}

\<CODE ENDS>
```

Figure 19: CDDL for Enrollment Status POST

An example status report can be seen below. It is sent with the media type: application/json

```
{
    "version": 1,
    "status":true,
    "reason":"Informative human readable message",
    "reason-context": { "additional" : "JSON" }
}
```

Figure 20: Example of Enrollment Status POST

The server **SHOULD** respond with an HTTP 200 but **MAY** simply fail with an HTTP 404 error.

Within the server logs, the server **MUST** capture if this message was received over a TLS session with a matching Client Certificate.

5.9.5. Multiple Certificates

Pledges that require multiple certificates could establish direct EST connections to the registrar.

5.9.6. EST over CoAP

This document describes extensions to EST for the purpose of bootstrapping remote key infrastructures. Bootstrapping is relevant for CoAP enrollment discussions as well. The definition of EST and BRSKI over CoAP is not discussed within this document beyond ensuring proxy support for CoAP operations. Instead, it is anticipated that a definition of CoAP mappings will occur in subsequent documents such as [ACE-COAP-EST] and that CoAP mappings for BRSKI will be discussed either there or in future work.

6. Clarification Of Transfer-Encoding

[RFC7030] defines endpoints to include a "Content-Transfer-Encoding" heading and payloads to be base64-encoded DER [RFC4648].

When used within BRSKI, the original EST endpoints remain base64 encoded [RFC7030] (as clarified by [RFC8951]), but the new BRSKI endpoints that send and receive binary artifacts (specifically, "/.well-known/brski/requestvoucher") are binary. That is, no encoding is used.

In the BRSKI context, the EST "Content-Transfer-Encoding" header field **SHOULD** be ignored if present. This header field does not need to be included.

7. Reduced Security Operational Modes

A common requirement of bootstrapping is to support less secure operational modes for support-specific use cases. This section suggests a range of mechanisms that would alter the security assurance of BRSKI to accommodate alternative deployment architectures and mitigate life-cycle management issues identified in Section 10. They are presented here as informative (non-normative) design guidance for future standardization activities. Section 9 provides standardization applicability statements for the ANIMA ACP. Other users would expect that subsets of these mechanisms could be profiled with accompanying applicability statements similar to the one described in Section 9.

This section is considered non-normative in the generality of the protocol. Use of the suggested mechanisms here **MUST** be detailed in specific profiles of BRSKI, such as in Section 9.

7.1. Trust Model

This section explains the trust relationships detailed in Section 2.4:

```
+--------+         +---------+    +------------+     +------------+
| Pledge |         | Join    |    | Domain     |     |Manufacturer|
|        |         | Proxy   |    | Registrar  |     | Service    |
|        |         |         |    |            |     | (Internet) |
+--------+         +---------+    +------------+     +------------+
```

Figure 21: Elements of BRSKI Trust Model

- Pledge:

  The pledge could be compromised and provide an attack vector for malware. The entity is trusted to only imprint using secure methods described in this document. Additional endpoint assessment techniques are **RECOMMENDED** but are out of scope of this document.

- Join Proxy:

  Provides proxy functionalities but is not involved in security considerations.

- Registrar:

  When interacting with a MASA, a registrar makes all decisions. For Ownership Audit Vouchers (see [RFC8366]), the registrar is provided an opportunity to accept MASA decisions.

- Vendor Service, MASA:

  This form of manufacturer service is trusted to accurately log all claim attempts and to provide authoritative log information to registrars. The MASA does not know which devices are associated with which domains. These claims could be strengthened by using cryptographic log techniques to provide append only, cryptographic assured, publicly auditable logs.

- Vendor Service, Ownership Validation:

  This form of manufacturer service is trusted to accurately know which device is owned by which domain.

7.2. Pledge Security Reductions

The following is a list of alternative behaviors that the pledge can be programmed to implement. These behaviors are not mutually exclusive, nor are they dependent upon each other. Some of these methods enable offline and emergency (touch-based) deployment use cases. Normative language is used as these behaviors are referenced in later sections in a normative fashion.

1. The pledge **MUST** accept nonceless vouchers. This allows for a use case where the registrar cannot connect to the MASA at the deployment time. Logging and validity periods address the security considerations of supporting these use cases.
2. Many devices already support "trust on first use" for physical interfaces such as console ports. This document does not change that reality. Devices supporting this protocol **MUST NOT** support "trust on first use" on network interfaces. This is because "trust on first use" over network interfaces would undermine the logging based security protections provided by this specification.
3. The pledge **MAY** have an operational mode where it skips voucher validation one time, for example, if a physical button is depressed during the bootstrapping operation. This can be useful if the manufacturer service is unavailable. This behavior **SHOULD** be available via local configuration or physical presence methods (such as use of a serial/craft console) to ensure new entities can always be deployed even when autonomic methods fail. This allows for unsecured imprint.
4. A craft/serial console could include a command such as "est-enroll [2001:db8:0:1]:443" that begins the EST process from the point after the voucher is validated. This process **SHOULD** include server certificate verification using an on-screen fingerprint.

It is **RECOMMENDED** that "trust on first use" or any method of skipping voucher validation (including use of a craft serial console) only be available if hardware-assisted Network Endpoint Assessment (NEA) [RFC5209] is supported. This recommendation ensures that domain network monitoring can detect inappropriate use of offline or emergency deployment procedures when voucher-based bootstrapping is not used.

7.3. Registrar Security Reductions

A registrar can choose to accept devices using less secure methods. They **MUST NOT** be the default behavior. These methods may be acceptable in situations where threat models indicate that low security is adequate. This includes situations where security decisions are being made by the local administrator:

1. A registrar **MAY** choose to accept all devices, or all devices of a particular type. The administrator could make this choice in cases where it is operationally difficult to configure the registrar with the unique identifier of each new device expected.
2. A registrar **MAY** choose to accept devices that claim a unique identity without the benefit of authenticating that claimed identity. This could occur when the pledge does not include an X.509 IDevID factory-installed credential. New entities without an X.509 IDevID credential **MAY** form the request per Section 5.2 using the format per Section 5.5 to ensure the pledge's serial number information is provided to the registrar (this includes the IDevID AuthorityKeyIdentifier value, which would be statically configured on the pledge). The pledge **MAY** refuse to provide a TLS Client Certificate (as one is not available). The pledge **SHOULD** support HTTP-based or certificate-less TLS authentication as described in EST [RFC7030], Section 3.3.2. A registrar **MUST NOT** accept unauthenticated new entities unless it has been configured to do so by an administrator that has verified that only expected new entities can communicate with a registrar (presumably via a physically secured perimeter.)
3. A registrar **MAY** submit a nonceless voucher-request to the MASA service (by not including a nonce in the voucher-request). The resulting vouchers can then be stored by the registrar until they are needed during bootstrapping operations. This is for use cases where the target network is protected by an air gap and therefore cannot contact the MASA service during pledge deployment.
4. A registrar **MAY** ignore unrecognized nonceless log entries. This could occur when used equipment is purchased with a valid history of being deployed in air gap networks that required offline vouchers.
5. A registrar **MAY** accept voucher formats of future types that cannot be parsed by the registrar. This reduces the registrar's visibility into the exact voucher contents but does not change the protocol operations.

7.4. MASA Security Reductions

Lower security modes chosen by the MASA service affect all device deployments unless the lower security behavior is tied to specific device identities. The modes described below can be applied to specific devices via knowledge of what devices were sold. They can also be bound to specific customers (independent of the device identity) by authenticating the customer's registrar.

7.4.1. Issuing Nonceless Vouchers

A MASA has the option of not including a nonce in the voucher and/or not requiring one to be present in the voucher-request. This results in distribution of a voucher that may never expire and, in effect, makes the specified domain an always trusted entity to the pledge during any subsequent bootstrapping attempts. The log information captures when a nonceless voucher is issued so that the registrar can make appropriate security decisions when a pledge joins the domain. Nonceless vouchers are useful to support use cases where registrars might not be online during actual device deployment.

While a nonceless voucher may include an expiry date, a typical use for a nonceless voucher is for it to be long lived. If the device can be trusted to have an accurate clock (the MASA will know), then a nonceless voucher CAN be issued with a limited lifetime.

A more typical case for a nonceless voucher is for use with offline onboarding scenarios where it is not possible to pass a fresh voucher-request to the MASA. The use of a long-lived voucher also eliminates concern about the availability of the MASA many years in the future. Thus, many nonceless vouchers will have no expiry dates.

Thus, the long-lived nonceless voucher does not require proof that the device is online. Issuing such a thing is only accepted when the registrar is authenticated by the MASA and the MASA is authorized to provide this functionality to this customer. The MASA is **RECOMMENDED** to use this functionality only in concert with an enhanced level of ownership tracking, the details of which are out of scope for this document.

If the pledge device is known to have a real-time clock that is set from the factory, use of a voucher validity period is **RECOMMENDED**.

7.4.2. Trusting Owners on First Use

A MASA has the option of not verifying ownership before responding with a voucher. This is expected to be a common operational model because doing so relieves the manufacturer providing MASA services from having to track ownership during shipping and throughout the supply chain, and it allows for a very low overhead MASA service. A registrar uses the audit-log information as an in-depth defense strategy to ensure that this does not occur unexpectedly (for example, when purchasing new equipment, the registrar would throw an error if any audit-log information is reported). The MASA **SHOULD** verify the prior-signed-voucher-request information for pledges that support that functionality. This provides a proof-of-proximity check that reduces the need for ownership verification. The proof-of-proximity comes from the assumption that the pledge and Join Proxy are on the same link-local connection.

A MASA that practices TOFU for registrar identity may wish to annotate the origin of the connection by IP address or netblock and restrict future use of that identity from other locations. A MASA that does this **SHOULD** take care to not create nuisance situations for itself when a customer has multiple registrars or uses outgoing IPv4-to-IPv4 NAT (NAT44) connections that change frequently.

7.4.3. Updating or Extending Voucher Trust Anchors

This section deals with two problems: A MASA that is no longer available due to a failed business and a MASA that is uncooperative to a secondary sale.

A manufacturer could offer a management mechanism that allows the list of voucher verification trust anchors to be extended. [YANG-KEYSTORE] describes one such interface that could be implemented using YANG. Pretty much any configuration mechanism used today could be extended to provide the needed additional update. A manufacturer could even decide to install the domain CA trust anchors received during the EST "cacerts" step as voucher verification anchors. Some additional signals will be needed to clearly identify which keys have voucher validation authority from among those signed by the domain CA. This is future work.

With the above change to the list of anchors, vouchers can be issued by an alternate MASA. This could be the previous owner (the seller) or some other trusted third party who is mediating the sale. If it is a third party, the seller would need to take steps to introduce the third-party configuration to the device prior to disconnection. The third party (e.g., a wholesaler of used equipment) could, however, use a mechanism described in Section 7.2 to take control of the device after receiving it physically. This would permit the third party to act as the MASA for future onboarding actions. As the IDevID certificate probably cannot be replaced, the new owner's registrar would have to support an override of the MASA URL.

To be useful for resale or other transfers of ownership, one of two situations will need to occur. The simplest is that the device is not put through any kind of factory default/reset before going through onboarding again. Some other secure, physical signal would be needed to initiate it. This is most suitable for redeploying a device within the same enterprise. This would entail having previous configuration in the system until entirely replaced by the new owner, and it represents some level of risk.

For the second scenario, there would need to be two levels of factory reset. One would take the system back entirely to manufacturer state, including removing any added trust anchors, and the other (more commonly used) one would just restore the configuration back to a known default without erasing trust anchors. This weaker factory reset might leave valuable credentials on the device, and this may be unacceptable to some owners.

As a third option, the manufacturer's trust anchors could be entirely overwritten with local trust anchors. A factory default would never restore those anchors. This option comes with a lot of power but is also a lot of responsibility: if access to the private part of the new anchors are lost, the manufacturer may be unable to help.

8. IANA Considerations

Per this document, IANA has completed the following actions.

8.1. The IETF XML Registry

This document registers a URI in the "IETF XML Registry" [RFC3688]. IANA has registered the following:

- URI:

  urn:ietf:params:xml:ns:yang:ietf-voucher-request

- Registrant Contact:

  The ANIMA WG of the IETF.

- XML:

  N/A; the requested URI is an XML namespace.

8.2. YANG Module Names Registry

This document registers a YANG module in the "YANG Module Names" registry [RFC6020]. IANA has registered the following:

- Name:

  ietf-voucher-request

- Namespace:

  urn:ietf:params:xml:ns:yang:ietf-voucher-request

- Prefix:

  vch

- Reference:

  RFC 8995

8.3. BRSKI Well-Known Considerations

8.3.1. BRSKI .well-known Registration

To the "Well-Known URIs" registry at https://www.iana.org/assignments/well-known-uris/, this document registers the well-known name "brski" with the following filled-in template from [RFC8615]:

- URI Suffix:

  brski

- Change Controller:

  IETF

IANA has changed the registration of "est" to now only include [RFC7030] and no longer this document. Earlier draft versions of this document used "/.well-known/est" rather than "/.well-known/brski".

8.3.2. BRSKI .well-known Registry

IANA has created a new registry entitled: "BRSKI Well-Known URIs". The registry has three columns: URI, Description, and Reference. New items can be added using the Specification Required [RFC8126] process. The initial contents of this registry are:

URI	Description	Reference
requestvoucher	pledge to registrar, and from registrar to MASA	RFC 8995
voucher_status	pledge to registrar	RFC 8995
requestauditlog	registrar to MASA	RFC 8995
enrollstatus	pledge to registrar	RFC 8995

8.4. PKIX Registry

IANA has registered the following:

a number for id-mod-MASAURLExtn2016(96) from the pkix(7) id-mod(0) Registry.

IANA has assigned a number from the id-pe registry (Structure of Management Information (SMI) Security for PKIX Certificate Extension) for id-pe-masa-url with the value 32, resulting in an OID of 1.3.6.1.5.5.7.1.32.

8.5. Pledge BRSKI Status Telemetry

IANA has created a new registry entitled "BRSKI Parameters" and has created, within that registry, a table called: "Pledge BRSKI Status Telemetry Attributes". New items can be added using the Specification Required process. The following items are in the initial registration, with this document (see Section 5.7) as the reference:

- version
- Status
- Reason

- reason-context

8.6. DNS Service Names

IANA has registered the following service names:

- Service Name:

  brski-proxy

- Transport Protocol(s):

  tcp

- Assignee:

  IESG <iesg@ietf.org>

- Contact:

  IESG <iesg@ietf.org>

- Description:

  The Bootstrapping Remote Secure Key Infrastructure Proxy

- Reference:

  RFC 8995

- Service Name:

  brski-registrar

- Transport Protocol(s):

  tcp

- Assignee:

  IESG <iesg@ietf.org>

- Contact:

  IESG <iesg@ietf.org>

- Description:

  The Bootstrapping Remote Secure Key Infrastructure Registrar

- Reference:

  RFC 8995

8.7. GRASP Objective Names

IANA has registered the following GRASP Objective Names:

IANA has registered the value "AN_Proxy" (without quotes) to the "GRASP Objective Names" table in the GRASP Parameter registry. The specification for this value is Section 4.1.1 of this document.

The IANA has registered the value "AN_join_registrar" (without quotes) to the "GRASP Objective Names" table in the GRASP Parameter registry. The specification for this value is Section 4.3 of this document.

9. Applicability To The Autonomic Control Plane (ACP)

This document provides a solution to the requirements for secure bootstrapping as defined in "Using an Autonomic Control Plane for Stable Connectivity of Network Operations, Administration, and Maintenance (OAM)" [RFC8368], "A Reference Model for Autonomic Networking" [RFC8993], and specifically "An Autonomic Control Plane (ACP)" [RFC8994]; see Sections 3.2 ("Secure Bootstrap over an Unconfigured Network") and 6.2 ("ACP Domain, Certificate, and Network").

The protocol described in this document has appeal in a number of other non-ANIMA use cases. Such uses of the protocol will be deployed into other environments with different tradeoffs of privacy, security, reliability, and autonomy from manufacturers. As such, those use cases will need to provide their own applicability statements and will need to address unique privacy and security considerations for the environments in which they are used.

The ACP that is bootstrapped by the BRSKI protocol is typically used in medium to large Internet service provider organizations. Equivalent enterprises that have significant Layer 3 router connectivity also will find significant benefit, particularly if the enterprise has many sites. (A network consisting of primarily Layer 2 is not excluded, but the adjacencies that the ACP will create and maintain will not reflect the topology until all devices participate in the ACP.)

In the ACP, the Join Proxy is found to be proximal because communication between the pledge and the Join Proxy is exclusively on IPv6 link-local addresses. The proximity of the Join Proxy to the registrar is validated by the registrar using ANI ACP IPv6 ULAs. ULAs are not routable over the Internet, so as long as the Join Proxy is operating correctly, the proximity assertion is satisfied. Other uses of BRSKI will need similar analysis if they use proximity assertions.

As specified in the ANIMA charter, this work "focuses on professionally-managed networks." Such a network has an operator and can do things like install, configure, and operate the registrar function. The operator makes purchasing decisions and is aware of what manufacturers it expects to see on its network.

Such an operator is also capable of performing bootstrapping of a device using a serial console (craft console). The zero-touch mechanism presented in this and the ACP document [RFC8994] represents a significant efficiency: in particular, it reduces the need to put senior experts on airplanes to configure devices in person.

As the technology evolves, there is recognition that not every situation may work out, and occasionally a human may still have to visit. Given this, some mechanisms are presented in Section 7.2. The manufacturer **MUST** provide at least one of the one-touch mechanisms described that permit enrollment to proceed without the availability of any manufacturer server (such as the MASA).

The BRSKI protocol is going into environments where there have already been quite a number of vendor proprietary management systems. Those are not expected to go away quickly but rather to leverage the secure credentials that are provisioned by BRSKI. The connectivity requirements of the said management systems are provided by the ACP.

9.1. Operational Requirements

This section collects operational requirements based upon the three roles involved in BRSKI: the MASA, the (domain) owner, and the device. It should be recognized that the manufacturer may be involved in two roles, as it creates the software/firmware for the device and may also be the operator of the MASA.

The requirements in this section are presented using BCP 14 language [RFC2119] [RFC8174]. These do not represent new normative statements, just a review of a few such things in one place by role. They also apply specifically to the ANIMA ACP use case. Other use cases likely have similar, but **MAY** have different, requirements.

9.1.1. MASA Operational Requirements

The manufacturer **MUST** arrange for an online service called the MASA to be available. It **MUST** be available at the URL that is encoded in the IDevID certificate extensions described in Section 2.3.2.

The online service **MUST** have access to a private key with which to sign voucher artifacts [RFC8366]. The public key, certificate, or certificate chain **MUST** be built into the device as part of the firmware.

It is **RECOMMENDED** that the manufacturer arrange for this signing key (or keys) to be escrowed according to typical software source code escrow practices [softwareescrow].

The MASA accepts voucher-requests from domain owners according to an operational practice appropriate for the device. This can range from any domain owner (first-come first-served, on a TOFU-like basis), to full sales channel integration where domain owners need to be positively identified by TLS pinned Client Certificates or an HTTP authentication process. The MASA creates signed voucher artifacts according to its internally defined policies.

The MASA **MUST** operate an audit-log for devices that is accessible. The audit-log is designed to be easily cacheable, and the MASA **MAY** find it useful to put this content on a Content Delivery Network (CDN).

9.1.2. Domain Owner Operational Requirements

The domain owner **MUST** operate an EST [RFC7030] server with the extensions described in this document. This is the JRC or registrar. This JRC/EST server **MUST** announce itself using GRASP within the ACP. This EST server will typically reside with the Network Operations Center for the organization.

The domain owner **MAY** operate an internal CA that is separate from the EST server, or it **MAY** combine all activities into a single device. The determination of the architecture depends upon the scale and resiliency requirements of the organization. Multiple JRC instances **MAY** be announced into the ACP from multiple locations to achieve an appropriate level of redundancy.

In order to recognize which devices and which manufacturers are welcome on the domain owner's network, the domain owner **SHOULD** maintain an acceptlist of manufacturers. This **MAY** extend to integration with purchasing departments to know the serial numbers of devices.

The domain owner **SHOULD** use the resulting overlay ACP network to manage devices, replacing legacy out-of-band mechanisms.

The domain owner **SHOULD** operate one or more EST servers that can be used to renew the domain certificates (LDevIDs), which are deployed to devices. These servers **MAY** be the same as the JRC or **MAY** be a distinct set of devices, as appropriate for resiliency.

The organization **MUST** take appropriate precautions against loss of access to the CA private key. Hardware security modules and/or secret splitting are appropriate.

9.1.3. Device Operational Requirements

Devices **MUST** come with built-in trust anchors that permit the device to validate vouchers from the MASA.

Devices **MUST** come with (unique, per-device) IDevID certificates that include their serial numbers and the MASA URL extension.

Devices are expected to find Join Proxies using GRASP, and then connect to the JRC using the protocol described in this document.

Once a domain owner has been validated with the voucher, devices are expected to enroll into the domain using EST. Devices are then expected to form ACPs using IPsec over IPv6 link-local addresses as described in [RFC8994].

Once a device has been enrolled, it **SHOULD** listen for the address of the JRC using GRASP, and it **SHOULD** enable itself as a Join Proxy and announce itself on all links/interfaces using GRASP DULL.

Devices are expected to renew their certificates before they expire.

10. Privacy Considerations

10.1. MASA Audit-Log

The MASA audit-log includes the domainID for each domain a voucher has been issued to. This information is closely related to the actual domain identity. A MASA may need additional defenses against Denial-of-Service attacks (Section 11.1), and this may involve collecting additional (unspecified here) information. This could provide sufficient information for the MASA service to build a detailed understanding of the devices that have been provisioned within a domain.

There are a number of design choices that mitigate this risk. The domain can maintain some privacy since it has not necessarily been authenticated and is not authoritatively bound to the supply chain.

Additionally, the domainID captures only the unauthenticated subject key identifier of the domain. A privacy-sensitive domain could theoretically generate a new domainID for each device being deployed. Similarly, a privacy-sensitive domain would likely purchase devices that support proximity assertions from a manufacturer that does not require sales channel integrations. This would result in a significant level of privacy while maintaining the security characteristics provided by the registrar-based audit-log inspection.

10.2. What BRSKI-EST Reveals

During the provisional phase of the BRSKI-EST connection between the pledge and the registrar, each party reveals its certificates to each other. For the pledge, this includes the serialNumber attribute, the MASA URL, and the identity that signed the IDevID certificate.

TLS 1.2 reveals the certificate identities to on-path observers, including the Join Proxy.

TLS 1.3 reveals the certificate identities only to the end parties, but as the connection is provisional; an on-path attacker (MITM) can see the certificates. This includes not just malicious attackers but also registrars that are visible to the pledge but are not part of the intended domain.

The certificate of the registrar is rather arbitrary from the point of view of the BRSKI protocol. As no validations [RFC6125] are expected to be done, the contents could be easily pseudonymized. Any device that can see a Join Proxy would be able to connect to the registrar and learn the identity of the network in question. Even if the contents of the certificate are pseudonymized, it would be possible to correlate different connections in different locations that belong to the same entity. This is unlikely to present a significant privacy concern to ANIMA ACP uses of BRSKI, but it may be a concern to other users of BRSKI.

The certificate of the pledge could be revealed by a malicious Join Proxy that performed a MITM attack on the provisional TLS connection. Such an attacker would be able to reveal the identity of the pledge to third parties if it chose to do so.

Research into a mechanism to do multistep, multiparty authenticated key agreement, incorporating some kind of zero-knowledge proof, would be valuable. Such a mechanism would ideally avoid disclosing identities until the pledge, registrar, and MASA agree to the transaction. Such a mechanism would need to discover the location of the MASA without knowing the identity of the pledge or the identity of the MASA. This part of the problem may be unsolvable.

10.3. What BRSKI-MASA Reveals To The Manufacturer

With consumer-oriented devices, the "call-home" mechanism in IoT devices raises significant privacy concerns. See [livingwithIoT] and [IoTstrangeThings] for exemplars. The ACP usage of BRSKI is not targeted at individual usage of IoT devices but rather at the enterprise and ISP creation of networks in a zero-touch fashion where the "call-home" represents a different class of privacy and life-cycle management concerns.

It needs to be reiterated that the BRSKI-MASA mechanism only occurs once during the commissioning of the device. It is well defined, and although encrypted with TLS, it could in theory be made auditable as the contents are well defined. This connection does not occur when the device powers on or is restarted for normal routines. (It is conceivable, but remarkably unusual, that a device could be forced to go through a full factory reset during an exceptional firmware update situation, after which enrollment would have to be repeated, and a new connection would occur.)

The BRSKI call-home mechanism is mediated via the owner's registrar, and the information that is transmitted is directly auditable by the device owner. This is in stark contrast to many "call-home" protocols where the device autonomously calls home and uses an undocumented protocol.

While the contents of the signed part of the pledge voucher-request cannot be changed, they are not encrypted at the registrar. The ability to audit the messages by the owner of the network is a mechanism to defend against exfiltration of data by a nefarious pledge. Both are, to reiterate, encrypted by TLS while in transit.

The BRSKI-MASA exchange reveals the following information to the manufacturer:

- the identity of the device being enrolled. This is revealed by transmission of a signed voucher-request containing the serial-number. The manufacturer can usually link the serial number to a device model.
- an identity of the domain owner in the form of the domain trust anchor. However, this is not a global PKI-anchored name within the WebPKI, so this identity could be pseudonymous. If there is sales channel integration, then the MASA will have authenticated the domain owner, via either a pinned certificate or perhaps another HTTP authentication method, as per Section 5.5.4.
- the time the device is activated.
- the IP address of the domain owner's registrar. For ISPs and enterprises, the IP address provides very clear geolocation of the owner. No amount of IP address privacy extensions [RFC8981] can do anything about this, as a simple whois lookup

likely identifies the ISP or enterprise from the upper bits anyway. A passive attacker who observes the connection definitely may conclude that the given enterprise/ISP is a customer of the particular equipment vendor. The precise model that is being enrolled will remain private.

Based upon the above information, the manufacturer is able to track a specific device from pseudonymous domain identity to the next pseudonymous domain identity. If there is sales-channel integration, then the identities are not pseudonymous.

The manufacturer knows the IP address of the registrar, but it cannot see the IP address of the device itself. The manufacturer cannot track the device to a detailed physical or network location, only to the location of the registrar. That is likely to be at the enterprise or ISP's headquarters.

The above situation is to be distinguished from a residential/individual person who registers a device from a manufacturer. Individuals do not tend to have multiple offices, and their registrar is likely on the same network as the device. A manufacturer that sells switching/routing products to enterprises should hardly be surprised if additional purchases of switching/routing products are made. Deviations from a historical trend or an established baseline would, however, be notable.

The situation is not improved by the enterprise/ISP using anonymization services such as Tor [Dingledine], as a TLS 1.2 connection will reveal the ClientCertificate used, clearly identifying the enterprise/ISP involved. TLS 1.3 is better in this regard, but an active attacker can still discover the parties involved by performing a MITM attack on the first attempt (breaking/killing it with a TCP reset (RST)), and then letting subsequent connection pass through.

A manufacturer could attempt to mix the BRSKI-MASA traffic in with general traffic on their site by hosting the MASA behind the same (set) of load balancers that the company's normal marketing site is hosted behind. This makes a lot of sense from a straight capacity planning point of view as the same set of services (and the same set of Distributed Denial-of-Service mitigations) may be used. Unfortunately, as the BRSKI-MASA connections include TLS ClientCertificate exchanges, this may easily be observed in TLS 1.2, and a traffic analysis may reveal it even in TLS 1.3. This does not make such a plan irrelevant. There may be other organizational reasons to keep the marketing site (which is often subject to frequent redesigns, outsourcing, etc.) separate from the MASA, which may need to operate reliably for decades.

10.4. Manufacturers And Used Or Stolen Equipment

As explained above, the manufacturer receives information each time a device that is in factory-default mode does a zero-touch bootstrap and attempts to enroll into a domain owner's registrar.

The manufacturer is therefore in a position to decline to issue a voucher if it detects that the new owner is not the same as the previous owner.

1. This can be seen as a feature if the equipment is believed to have been stolen. If the legitimate owner notifies the manufacturer of the theft, then when the new owner brings the device up, if they use the zero-touch mechanism, the new (illegitimate) owner reveals their location and identity.
2. In the case of used equipment, the initial owner could inform the manufacturer of the sale, or the manufacturer may just permit resales unless told otherwise. In which case, the transfer of ownership simply occurs.
3. A manufacturer could, however, decide not to issue a new voucher in response to a transfer of ownership. This is essentially the same as the stolen case, with the manufacturer having decided that the sale was not legitimate.
4. There is a fourth case, if the manufacturer is providing protection against stolen devices. The manufacturer then has a responsibility to protect the legitimate owner against fraudulent claims that the equipment was stolen. In the absence of such manufacturer protection, such a claim would cause the manufacturer to refuse to issue a new voucher. Should the device go through a deep factory reset (for instance, replacement of a damaged main board component), the device would not bootstrap.
5. Finally, there is a fifth case: the manufacturer has decided to end-of-line the device, or the owner has not paid a yearly support amount, and the manufacturer refuses to issue new vouchers at that point. This last case is not new to the industry: many license systems are already deployed that have a significantly worse effect.

This section has outlined five situations in which a manufacturer could use the voucher system to enforce what are clearly license terms. A manufacturer that attempted to enforce license terms via vouchers would find it rather ineffective as the terms would only be enforced when the device is enrolled, and this is not (to repeat) a daily or even monthly occurrence.

10.5. Manufacturers And Grey Market Equipment

Manufacturers of devices often sell different products into different regional markets. Which product is available in which market can be driven by price differentials, support issues (some markets may require manuals and tech support to be done in the local language), and government export regulation (such as whether strong crypto is permitted to be exported or permitted to be used in a particular market). When a domain owner obtains a device from a different market (they can be new) and transfers it to a different location, this is called a Grey Market.

A manufacturer could decide not to issue a voucher to an enterprise/ISP based upon their location. There are a number of ways that this could be determined: from the geolocation of the registrar, from sales channel knowledge about the customer, and from what products are available or unavailable in that market. If the device has a GPS, the coordinates of the device could even be placed into an extension of the voucher.

The above actions are not illegal, and not new. Many manufacturers have shipped crypto-weak (exportable) versions of firmware as the default on equipment for decades. The first task of an enterprise/ISP has always been to login to a manufacturer system, show one's "entitlement" (country information, proof that support payments have been made), and receive either a new updated firmware or a license key that will activate the correct firmware.

BRSKI permits the above process to be automated (in an autonomic fashion) and therefore perhaps encourages this kind of differentiation by reducing the cost of doing it.

An issue that manufacturers will need to deal with in the above automated process is when a device is shipped to one country with one set of rules (or laws or entitlements), but the domain registry is in another one. Which rules apply is something that will have to be worked out: the manufacturer could believe they are dealing with Grey Market equipment when they are simply dealing with a global enterprise.

10.6. Some Mitigations For Meddling By Manufacturers

The most obvious mitigation is not to buy the product. Pick manufacturers that are up front about their policies and who do not change them gratuitously.

Section 7.4.3 describes some ways in which a manufacturer could provide a mechanism to manage the trust anchors and built-in certificates (IDevID) as an extension. There are a variety of mechanisms, and some may take a substantial amount of work to get exactly correct. These mechanisms do not change the flow of the protocol described here but rather allow the starting trust assumptions to be changed. This is an area for future standardization work.

Replacement of the voucher validation anchors (usually pointing to the original manufacturer's MASA) with those of the new owner permits the new owner to issue vouchers to subsequent owners. This would be done by having the selling (old) owner run a MASA.

The BRSKI protocol depends upon a trust anchor and an identity on the device. Management of these entities facilitates a few new operational modes without making any changes to the BRSKI protocol. Those modes include: offline modes where the domain owner operates an internal MASA for all devices, resell modes where the first domain owner becomes the MASA for the next (resold-to) domain owner, and services where an aggregator acquires a large variety of devices and then acts as a pseudonymized MASA for a variety of devices from a variety of manufacturers.

Although replacement of the IDevID is not required for all modes described above, a manufacturer could support such a thing. Some may wish to consider replacement of the IDevID as an indication that the device's warranty is terminated. For others, the privacy requirements of some deployments might consider this a standard operating practice.

As discussed at the end of Section 5.8.1, new work could be done to use a distributed consensus technology for the audit-log. This would permit the audit-log to continue to be useful, even when there is a chain of MASA due to changes of ownership.

10.7. Death Of A Manufacturer

A common concern has been that a manufacturer could go out of business, leaving owners of devices unable to get new vouchers for existing products. Said products might have been previously deployed but need to be reinitialized, used, or kept in a warehouse as long-term spares.

The MASA was named the Manufacturer *Authorized* Signing Authority to emphasize that it need not be the manufacturer itself that performs this. It is anticipated that specialist service providers will come to exist that deal with the creation of vouchers in much the same way that many companies have outsourced email, advertising, and janitorial services.

Further, it is expected that as part of any service agreement, the manufacturer would arrange to escrow appropriate private keys such that a MASA service could be provided by a third party. This has routinely been done for source code for decades.

11. Security Considerations

This document details a protocol for bootstrapping that balances operational concerns against security concerns. As detailed in the introduction, and touched on again in Section 7, the protocol allows for reduced security modes. These attempt to deliver additional control to the local administrator and owner in cases where less security provides operational benefits. This section goes into more detail about a variety of specific considerations.

To facilitate logging and administrative oversight, in addition to triggering registrar verification of MASA logs, the pledge reports on the voucher parsing status to the registrar. In the case of a failure, this information is informative to a potentially malicious registrar. This is mandated anyway because of the operational benefits of an informed administrator in cases where the failure is indicative of a problem. The registrar is **RECOMMENDED** to verify MASA logs if voucher status telemetry is not received.

To facilitate truly limited clients, EST requires that the client **MUST** support a client authentication model (see [RFC7030], Section 3.3.2); Section 7 updates these requirements by stating that the registrar **MAY** choose to accept devices that fail cryptographic authentication. This reflects current (poor) practices in shipping devices without a cryptographic identity that are **NOT RECOMMENDED**.

During the provisional period of the connection, the pledge **MUST** treat all HTTP header and content data as untrusted data. HTTP libraries are regularly exposed to non-secured HTTP traffic: mature libraries should not have any problems.

Pledges might chose to engage in protocol operations with multiple discovered registrars in parallel. As noted above, they will only do so with distinct nonce values, but the end result could be multiple vouchers issued from the MASA if all registrars attempt to claim the device. This is not a failure, and the pledge chooses whichever voucher to accept based on internal logic. The registrars verifying log information will see multiple entries and take this into account for their analytic purposes.

11.1. Denial Of Service (DoS) Against MASA

There are use cases where the MASA could be unavailable or uncooperative to the registrar. They include active DoS attacks, planned and unplanned network partitions, changes to MASA policy, or other instances where MASA policy rejects a claim. These introduce an operational risk to the registrar owner in that MASA behavior might limit the ability to bootstrap a pledge device. For example, this might be an issue during disaster recovery. This risk can be mitigated by registrars that request and maintain long-term copies of "nonceless" vouchers. In that way, they are guaranteed to be able to bootstrap their devices.

The issuance of nonceless vouchers themselves creates a security concern. If the registrar of a previous domain can intercept protocol communications, then it can use a previously issued nonceless voucher to establish management control of a pledge device even after having sold it. This risk is mitigated by recording the issuance of such vouchers in the MASA audit-log that is verified by the subsequent registrar and by pledges only bootstrapping when in a factory default state. This reflects a balance between enabling MASA independence during future bootstrapping and the security of bootstrapping itself. Registrar control over requesting and auditing nonceless vouchers allows device owners to choose an appropriate balance.

The MASA is exposed to DoS attacks wherein attackers claim an unbounded number of devices. Ensuring a registrar is representative of a valid manufacturer customer, even without validating ownership of specific pledge devices, helps to mitigate this. Pledge signatures on the pledge voucher-request, as forwarded by the registrar in the prior-signed-voucher-request field of the registrar voucher-request, significantly reduce this risk by ensuring the MASA can confirm proximity between the pledge and the registrar making the request. Supply-chain integration ("know your customer") is an additional step that MASA providers and device vendors can explore.

11.2. DomainID Must Be Resistant To Second-Preimage Attacks

The domainID is used as the reference in the audit-log to the domain. The domainID is expected to be calculated by a hash that is resistant to a second-preimage attack. Such an attack would allow a second registrar to create audit-log entries that are fake.

11.3. Availability Of Good Random Numbers

The nonce used by the pledge in the voucher-request **SHOULD** be generated by a Strong Cryptographic Sequence ([RFC4086], Section 6.2). TLS has a similar requirement.

In particular, implementations should pay attention to the advance in [RFC4086]; see Sections 3 and, in particular, 3.4. The random seed used by a device at boot **MUST** be unique across all devices and all bootstraps. Resetting a device to factory default state does not obviate this requirement.

11.4. Freshness In Voucher-Requests

A concern has been raised that the pledge voucher-request should contain some content (a nonce) provided by the registrar and/or MASA in order for those actors to verify that the pledge voucher-request is fresh.

There are a number of operational problems with getting a nonce from the MASA to the pledge. It is somewhat easier to collect a random value from the registrar, but as the registrar is not yet vouched for, such a registrar nonce has little value. There are privacy and logistical challenges to addressing these operational issues, so if such a thing were to be considered, it would have to provide some clear value. This section examines the impacts of not having a fresh pledge voucher-request.

Because the registrar authenticates the pledge, a full MITM attack is not possible, despite the provisional TLS authentication by the pledge (see Section 5.) Instead, we examine the case of a fake registrar (Rm) that communicates with the pledge in parallel or in close-time proximity with the intended registrar. (This scenario is intentionally supported as described in Section 4.1.)

The fake registrar (Rm) can obtain a voucher signed by the MASA either directly or through arbitrary intermediaries. Assuming that the MASA accepts the registrar voucher-request (because either the Rm is collaborating with a legitimate registrar according to supply-chain information or the MASA is in audit-log only mode), then a voucher linking the pledge to the registrar Rm is issued.

Such a voucher, when passed back to the pledge, would link the pledge to registrar Rm and permit the pledge to end the provisional state. It now trusts the Rm and, if it has any security vulnerabilities leverageable by an Rm with full administrative control, can be assumed to be a threat against the intended registrar.

This flow is mitigated by the intended registrar verifying the audit-logs available from the MASA as described in Section 5.8. The Rm might chose to collect a voucher-request but wait until after the intended registrar completes the authorization process before submitting it. This pledge voucher-request would be "stale" in that it has a nonce that no longer matches the internal state of the pledge. In order to successfully use any resulting voucher, the Rm would need to remove the stale nonce or anticipate the pledge's future nonce state. Reducing the possibility of this is why the pledge is mandated to generate a strong random or pseudo-random number nonce.

Additionally, in order to successfully use the resulting voucher, the Rm would have to attack the pledge and return it to a bootstrapping-enabled state. This would require wiping the pledge of current configuration and triggering a rebootstrapping of the pledge. This is no more likely than simply taking control of the pledge directly, but if this is a consideration, it is **RECOMMENDED** that the target network take the following steps:

- Ongoing network monitoring for unexpected bootstrapping attempts by pledges.
- Retrieval and examination of MASA log information upon the occurrence of any such unexpected events. The Rm will be listed in the logs along with nonce information for analysis.

11.5. Trusting Manufacturers

The BRSKI extensions to EST permit a new pledge to be completely configured with domain-specific trust anchors. The link from built-in manufacturer-provided trust anchors to domain-specific trust anchors is mediated by the signed voucher artifact.

If the manufacturer's IDevID signing key is not properly validated, then there is a risk that the network will accept a pledge that should not be a member of the network. As the address of the manufacturer's MASA is provided in the IDevID using the extension from Section 2.3, the malicious pledge will have no problem collaborating with its MASA to produce a completely valid voucher.

BRSKI does not, however, fundamentally change the trust model from domain owner to manufacturer. Assuming that the pledge used its IDevID with EST [RFC7030] and BRSKI, the domain (registrar) still needs to trust the manufacturer.

Establishing this trust between domain and manufacturer is outside the scope of BRSKI. There are a number of mechanisms that can be adopted including:

- Manually configuring each manufacturer's trust anchor.
- A TOFU mechanism. A human would be queried upon seeing a manufacturer's trust anchor for the first time, and then the trust anchor would be installed to the trusted store. There are risks with this; even if the key to name mapping is validated using something like the WebPKI, there remains the possibility that the name is a look alike: e.g., dem0.example. vs. demO.example.
- scanning the trust anchor from a QR code that came with the packaging (this is really a manual TOFU mechanism).
- some sales integration processing where trust anchors are provided as part of the sales process, probably included in a digital packing "slip", or a sales invoice.
- consortium membership, where all manufacturers of a particular device category (e.g, a light bulb or a cable modem) are signed by a CA specifically for this. This is done by CableLabs today. It is used for authentication and authorization as part of [docsisroot] and [TR069].

The existing WebPKI provides a reasonable anchor between manufacturer name and public key. It authenticates the key. It does not provide a reasonable authorization for the manufacturer, so it is not directly usable on its own.

11.6. Manufacturer Maintenance Of Trust Anchors

BRSKI depends upon the manufacturer building in trust anchors to the pledge device. The voucher artifact that is signed by the MASA will be validated by the pledge using that anchor. This implies that the manufacturer needs to maintain access to a signing key that the pledge can validate.

The manufacturer will need to maintain the ability to make signatures that can be validated for the lifetime that the device could be onboarded. Whether this onboarding lifetime is less than the device lifetime depends upon how the device is used. An inventory of devices kept in a warehouse as spares might not be onboarded for many decades.

There are good cryptographic hygiene reasons why a manufacturer would not want to maintain access to a private key for many decades. A manufacturer in that situation can leverage a long-term CA anchor, built-in to the pledge, and then a certificate chain may be incorporated using the normal CMS certificate set. This may increase the size of the voucher artifacts, but that is not a significant issue in non-constrained environments.

There are a few other operational variations that manufacturers could consider. For instance, there is no reason that every device need have the same set of trust anchors preinstalled. Devices built in different factories, or on different days, or in any other consideration, could have different trust anchors built in, and the record of which batch the device is in would be recorded in the asset database. The manufacturer would then know which anchor to sign an artifact against.

Aside from the concern about long-term access to private keys, a major limiting factor for the shelf life of many devices will be the age of the cryptographic algorithms included. A device produced in 2019 will have hardware and software capable of validating algorithms common in 2019 and will have no defense against attacks (both quantum and von Neumann brute-force attacks) that have not yet been invented. This concern is orthogonal to the concern about access to private keys, but this concern likely dominates and limits the life span of a device in a warehouse. If any update to the firmware to support new cryptographic mechanisms were possible (while the device was in a warehouse), updates to trust anchors would also be done at the same time.

The set of standard operating procedures for maintaining high-value private keys is well documented. For instance, the WebPKI provides a number of options for audits in [cabforumaudit], and the DNSSEC root operations are well documented in [dnssecroot].

It is not clear if manufacturers will take this level of precaution, or how strong the economic incentives are to maintain an appropriate level of security.

The next section examines the risk due to a compromised manufacturer IDevID signing key. This is followed by examination of the risk due to a compromised MASA key. The third section below examines the situation where a MASA web server itself is under attacker control, but the MASA signing key itself is safe in a not-directly connected hardware module.

11.6.1. Compromise of Manufacturer IDevID Signing Keys

An attacker that has access to the key that the manufacturer uses to sign IDevID certificates can create counterfeit devices. Such devices can claim to be from a particular manufacturer but can be entirely different devices: Trojan horses in effect.

As the attacker controls the MASA URL in the certificate, the registrar can be convinced to talk to the attacker's MASA. The registrar does not need to be in any kind of promiscuous mode to be vulnerable.

In addition to creating fake devices, the attacker may also be able to issue revocations for existing certificates if the IDevID certificate process relies upon CRL lists that are distributed.

There does not otherwise seem to be any risk from this compromise to devices that are already deployed or that are sitting locally in boxes waiting for deployment (local spares). The issue is that operators will be unable to trust devices that have been in an uncontrolled warehouse as they do not know if those are real devices.

11.6.2. Compromise of MASA Signing Keys

There are two periods of time in which to consider: when the MASA key has fallen into the hands of an attacker and after the MASA recognizes that the key has been compromised.

11.6.2.1. Attacker Opportunities with a Compromised MASA Key

An attacker that has access to the MASA signing key could create vouchers. These vouchers could be for existing deployed devices or for devices that are still in a warehouse. In order to exploit these vouchers, two things need to occur: the device has to go through a factory default boot cycle, and the registrar has to be convinced to contact the attacker's MASA.

If the attacker controls a registrar that is visible to the device, then there is no difficulty in delivery of the false voucher. A possible practical example of an attack like this would be in a data center, at an ISP peering point (whether a public IX or a private peering point). In such a situation, there are already cables attached to the equipment that lead to other devices (the peers at the IX), and through those links, the false voucher could be delivered. The difficult part would be to put the device through a factory reset. This might be accomplished through social engineering of data center staff. Most locked cages have ventilation holes, and possibly a long "paperclip" could reach through to depress a factory reset button. Once such a piece of ISP equipment has been compromised, it could be used to compromise equipment that it was connected to (through long haul links even), assuming that those pieces of equipment could also be forced through a factory reset.

The above scenario seems rather unlikely as it requires some element of physical access; but if there was a remote exploit that did not cause a direct breach, but rather a fault that resulted in a factory reset, this could provide a reasonable path.

The above deals with ANI uses of BRSKI. For cases where IEEE 802.11 or 802.15.4 is involved, the need to connect directly to the device is eliminated, but the need to do a factory reset is not. Physical possession of the device is not required as above, provided that there is some way to force a factory reset. With some consumer devices that have low overall implementation quality, end users might be familiar with the need to reset the device regularly.

The authors are unable to come up with an attack scenario where a compromised voucher signature enables an attacker to introduce a compromised pledge into an existing operator's network. This is the case because the operator controls the communication between registrar and MASA, and there is no opportunity to introduce the fake voucher through that conduit.

11.6.2.2. Risks after Key Compromise is Known

Once the operator of the MASA realizes that the voucher signing key has been compromised, it has to do a few things.

First, it **MUST** issue a firmware update to all devices that had that key as a trust anchor, such that they will no longer trust vouchers from that key. This will affect devices in the field that are operating, but those devices, being in operation, are not performing onboarding operations, so this is not a critical patch.

Devices in boxes (in warehouses) are vulnerable and remain vulnerable until patched. An operator would be prudent to unbox the devices, onboard them in a safe environment, and then perform firmware updates. This does not have to be done by the end-operator; it could be done by a distributor that stores the spares. A recommended practice for high-value devices (which typically have a <4hr service window) may be to validate the device operation on a regular basis anyway.

If the onboarding process includes attestations about firmware versions, then through that process, the operator would be advised to upgrade the firmware before going into production. Unfortunately, this does not help against situations where the attacker operates their own registrar (as listed above).

The need for short-lived vouchers is explained in [RFC8366], Section 6.1. The nonce guarantees freshness, and the short-lived nature of the voucher means that the window to deliver a fake voucher is very short. A nonceless, long-lived voucher would be the only option for the attacker, and devices in the warehouse would be vulnerable to such a thing.

A key operational recommendation is for manufacturers to sign nonceless, long-lived vouchers with a different key than what is used to sign short-lived vouchers. That key needs significantly better protection. If both keys come from a common trust-anchor (the manufacturer's CA), then a compromise of the manufacturer's CA would compromise both keys. Such a compromise of the manufacturer's CA likely compromises all keys outlined in this section.

11.6.3. Compromise of MASA Web Service

An attacker that takes over the MASA web service can inflict a number of attacks. The most obvious one is simply to take the database listing of customers and devices and sell the data to other attackers who will now know where to find potentially vulnerable devices.

The second most obvious thing that the attacker can do is to kill the service, or make it operate unreliably, making customers frustrated. This could have a serious effect on the ability to deploy new services by customers and would be a significant issue during disaster recovery.

While the compromise of the MASA web service may lead to the compromise of the MASA voucher signing key, if the signing occurs offboard (such as in a hardware signing module (HSM)), then the key may well be safe, but control over it resides with the attacker.

Such an attacker can issue vouchers for any device presently in service. Said device still needs to be convinced to go through a factory reset process before an attack.

If the attacker has access to a key that is trusted for long-lived nonceless vouchers, then they could issue vouchers for devices that are not yet in service. This attack may be very hard to verify as it would involve doing firmware updates on every device in warehouses (a potentially ruinously expensive process); a manufacturer might be reluctant to admit this possibility.

11.7. YANG Module Security Considerations

As described in Section 7.4 (Security Considerations) of [RFC8366], the YANG module specified in this document defines the schema for data that is subsequently encapsulated by a CMS signed-data content type, as described in Section 5 of [RFC5652]. As such, all of the YANG-modeled data is protected from modification.

The use of YANG to define data structures, via the "yang-data" statement, is relatively new and distinct from the traditional use of YANG to define an API accessed by network management protocols such as NETCONF [RFC6241] and RESTCONF [RFC8040]. For this reason, these guidelines do not follow the template described by Section 3.7 of [RFC8407].

12. References

12.1. Normative References

- [IDevID]

IEEE, "IEEE Standard for Local and metropolitan area networks - Secure Device Identity", IEEE 802.1AR, <https://1.ieee802.org/security/802-1ar>.

- [ITU.X690]

ITU-T, "Information Technology - ASN.1 encoding rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER)", ITU-T Recommendation X.690, ISO/IEC 8825-1:2015, August 2015, <https://www.itu.int/rec/T-REC-X.690>.

- [REST]

Fielding, R.F., "Architectural Styles and the Design of Network-based Software Architectures", 2000, <http://www.ics.uci.edu/~fielding/pubs/dissertation/fielding_dissertation.pdf>.

- [RFC2119]

Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

- [RFC3339]

Klyne, G. and C. Newman, "Date and Time on the Internet: Timestamps", RFC 3339, DOI 10.17487/RFC3339, July 2002, <https://www.rfc-editor.org/info/rfc3339>.

- [RFC3688]

Mealling, M., "The IETF XML Registry", BCP 81, RFC 3688, DOI 10.17487/RFC3688, January 2004, <https://www.rfc-editor.org/info/rfc3688>.

- [RFC3748]

Aboba, B., Blunk, L., Vollbrecht, J., Carlson, J., and H. Levkowetz, Ed., "Extensible Authentication Protocol (EAP)", RFC 3748, DOI 10.17487/RFC3748, June 2004, <https://www.rfc-editor.org/info/rfc3748>.

- [RFC3927]

Cheshire, S., Aboba, B., and E. Guttman, "Dynamic Configuration of IPv4 Link-Local Addresses", RFC 3927, DOI 10.17487/RFC3927, May 2005, <https://www.rfc-editor.org/info/rfc3927>.

- [RFC4086]

Eastlake 3rd, D., Schiller, J., and S. Crocker, "Randomness Requirements for Security", BCP 106, RFC 4086, DOI 10.17487/RFC4086, June 2005, <https://www.rfc-editor.org/info/rfc4086>.

- [RFC4519]

Sciberras, A., Ed., "Lightweight Directory Access Protocol (LDAP): Schema for User Applications", RFC 4519, DOI 10.17487/RFC4519, June 2006, <https://www.rfc-editor.org/info/rfc4519>.

- [RFC4648]

Josefsson, S., "The Base16, Base32, and Base64 Data Encodings", RFC 4648, DOI 10.17487/RFC4648, October 2006, <https://www.rfc-editor.org/info/rfc4648>.

- [RFC4862]

  Thomson, S., Narten, T., and T. Jinmei, "IPv6 Stateless Address Autoconfiguration", RFC 4862, DOI 10.17487/RFC4862, September 2007, <https://www.rfc-editor.org/info/rfc4862>.

- [RFC5272]

  Schaad, J. and M. Myers, "Certificate Management over CMS (CMC)", RFC 5272, DOI 10.17487/RFC5272, June 2008, <https://www.rfc-editor.org/info/rfc5272>.

- [RFC5280]

  Cooper, D., Santesson, S., Farrell, S., Boeyen, S., Housley, R., and W. Polk, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 5280, DOI 10.17487/RFC5280, May 2008, <https://www.rfc-editor.org/info/rfc5280>.

- [RFC5652]

  Housley, R., "Cryptographic Message Syntax (CMS)", STD 70, RFC 5652, DOI 10.17487/RFC5652, September 2009, <https://www.rfc-editor.org/info/rfc5652>.

- [RFC6020]

  Bjorklund, M., Ed., "YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF)", RFC 6020, DOI 10.17487/RFC6020, October 2010, <https://www.rfc-editor.org/info/rfc6020>.

- [RFC6125]

  Saint-Andre, P. and J. Hodges, "Representation and Verification of Domain-Based Application Service Identity within Internet Public Key Infrastructure Using X.509 (PKIX) Certificates in the Context of Transport Layer Security (TLS)", RFC 6125, DOI 10.17487/RFC6125, March 2011, <https://www.rfc-editor.org/info/rfc6125>.

- [RFC6241]

  Enns, R., Ed., Bjorklund, M., Ed., Schoenwaelder, J., Ed., and A. Bierman, Ed., "Network Configuration Protocol (NETCONF)", RFC 6241, DOI 10.17487/RFC6241, June 2011, <https://www.rfc-editor.org/info/rfc6241>.

- [RFC6762]

  Cheshire, S. and M. Krochmal, "Multicast DNS", RFC 6762, DOI 10.17487/RFC6762, February 2013, <https://www.rfc-editor.org/info/rfc6762>.

- [RFC6763]

  Cheshire, S. and M. Krochmal, "DNS-Based Service Discovery", RFC 6763, DOI 10.17487/RFC6763, February 2013, <https://www.rfc-editor.org/info/rfc6763>.

- [RFC7030]

Pritikin, M., Ed., Yee, P., Ed., and D. Harkins, Ed., "Enrollment over Secure Transport", RFC 7030, DOI 10.17487/RFC7030, October 2013, <https://www.rfc-editor.org/info/rfc7030>.

- [RFC7230]

  Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing", RFC 7230, DOI 10.17487/RFC7230, June 2014, <https://www.rfc-editor.org/info/rfc7230>.

- [RFC7231]

  Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content", RFC 7231, DOI 10.17487/RFC7231, June 2014, <https://www.rfc-editor.org/info/rfc7231>.

- [RFC7469]

  Evans, C., Palmer, C., and R. Sleevi, "Public Key Pinning Extension for HTTP", RFC 7469, DOI 10.17487/RFC7469, April 2015, <https://www.rfc-editor.org/info/rfc7469>.

- [RFC7950]

  Bjorklund, M., Ed., "The YANG 1.1 Data Modeling Language", RFC 7950, DOI 10.17487/RFC7950, August 2016, <https://www.rfc-editor.org/info/rfc7950>.

- [RFC7951]

  Lhotka, L., "JSON Encoding of Data Modeled with YANG", RFC 7951, DOI 10.17487/RFC7951, August 2016, <https://www.rfc-editor.org/info/rfc7951>.

- [RFC8040]

  Bierman, A., Bjorklund, M., and K. Watsen, "RESTCONF Protocol", RFC 8040, DOI 10.17487/RFC8040, January 2017, <https://www.rfc-editor.org/info/rfc8040>.

- [RFC8174]

  Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

- [RFC8259]

  Bray, T., Ed., "The JavaScript Object Notation (JSON) Data Interchange Format", STD 90, RFC 8259, DOI 10.17487/RFC8259, December 2017, <https://www.rfc-editor.org/info/rfc8259>.

- [RFC8366]

  Watsen, K., Richardson, M., Pritikin, M., and T. Eckert, "A Voucher Artifact for Bootstrapping Protocols", RFC 8366, DOI 10.17487/RFC8366, May 2018, <https://www.rfc-editor.org/info/rfc8366>.

- [RFC8368]

  Eckert, T., Ed. and M. Behringer, "Using an Autonomic Control Plane for Stable Connectivity of Network Operations, Administration, and Maintenance (OAM)", RFC 8368, DOI 10.17487/RFC8368, May 2018, <https://www.rfc-

editor.org/info/rfc8368>.

- [RFC8407]

  Bierman, A., "Guidelines for Authors and Reviewers of Documents Containing YANG Data Models", BCP 216, RFC 8407, DOI 10.17487/RFC8407, October 2018, <https://www.rfc-editor.org/info/rfc8407>.

- [RFC8446]

  Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", RFC 8446, DOI 10.17487/RFC8446, August 2018, <https://www.rfc-editor.org/info/rfc8446>.

- [RFC8610]

  Birkholz, H., Vigano, C., and C. Bormann, "Concise Data Definition Language (CDDL): A Notational Convention to Express Concise Binary Object Representation (CBOR) and JSON Data Structures", RFC 8610, DOI 10.17487/RFC8610, June 2019, <https://www.rfc-editor.org/info/rfc8610>.

- [RFC8951]

  Richardson, M., Werner, T., and W. Pan, "Clarification of Enrollment over Secure Transport (EST): Transfer Encodings and ASN.1", RFC 8951, DOI 10.17487/RFC8951, November 2020, <https://www.rfc-editor.org/info/rfc8951>.

- [RFC8981]

  Gont, F., Krishnan, S., Narten, T., and R. Draves, "Temporary Address Extensions for Stateless Address Autoconfiguration in IPv6", RFC 8981, DOI 10.17487/RFC8981, February 2021, <https://www.rfc-editor.org/info/rfc8981>.

- [RFC8990]

  Bormann, C., Carpenter, B., Ed., and B. Liu, Ed., "GeneRic Autonomic Signaling Protocol (GRASP)", RFC 8990, DOI 10.17487/RFC8990, May 2021, <https://www.rfc-editor.org/rfc/rfc8990>.

- [RFC8994]

  Eckert, T., Ed., Behringer, M., Ed., and S. Bjarnason, "An Autonomic Control Plane (ACP)", RFC 8994, DOI 10.17487/RFC8994, May 2021, <https://www.rfc-editor.org/rfc/rfc8994>.

12.2. Informative References

- [ACE-COAP-EST]

  van der Stok, P., Kampanakis, P., Richardson, M., and S. Raza, "EST over secure CoAP (EST-coaps)", Work in Progress, Internet-Draft, draft-ietf-ace-coap-est-18, 6 January 2020, <https://tools.ietf.org/html/draft-ietf-ace-coap-est-18>.

- [ANIMA-CONSTRAINED-VOUCHER]

  Richardson, M., van der Stok, P., Kampanakis, P., and E. Dijk, "Constrained Voucher Artifacts for Bootstrapping Protocols", Work in Progress, Internet-Draft, draft-ietf-anima-constrained-voucher-10, 21 February 2021, <https://tools.ietf.org/html/draft-ietf-anima-constrained-voucher-10>.

- [ANIMA-STATE]

  Richardson, M., "Considerations for stateful vs stateless join router in ANIMA bootstrap", Work in Progress, Internet-Draft, draft-richardson-anima-state-for-joinrouter-03, 22 September 2020, <https://tools.ietf.org/html/draft-richardson-anima-state-for-joinrouter-03>.

- [brewski]

  Urban Dictionary, "brewski", March 2003, <https://www.urbandictionary.com/define.php?term=brewski>.

- [cabforumaudit]

  CA/Browser Forum, "Information for Auditors and Assessors", August 2019, <https://cabforum.org/information-for-auditors-and-assessors/>.

- [Dingledine]

  Dingledine, R., Mathewson, N., and P. Syverson, "Tor: The Second-Generation Onion Router", August 2004, <https://svn-archive.torproject.org/svn/projects/design-paper/tor-design.pdf>.

- [dnssecroot]

  "DNSSEC Practice Statement for the Root Zone ZSK Operator", December 2017, <https://www.iana.org/dnssec/procedures/zsk-operator/dps-zsk-operator-v2.1.pdf>.

- [docsisroot]

  "CableLabs Digital Certificate Issuance Service", February 2018, <https://www.cablelabs.com/resources/digital-certificate-issuance-service/>.

- [imprinting]

  Wikipedia, "Imprinting (psychology)", January 2021, <https://en.wikipedia.org/w/index.php?title=Imprinting_(psychology)&=999211441>.

- [IoTstrangeThings]

  ESET, "IoT of toys stranger than fiction: Cybersecurity and data privacy update", March 2017, <https://www.welivesecurity.com/2017/03/03/internet-of-things-security-privacy-iot-update/>.

- [livingwithIoT]

  Silicon Republic, "What is it actually like to live in a house filled with IoT devices?", February 2018, <https://www.siliconrepublic.com/machines/iot-smart-devices-reality>.

- [minerva]

  Richardson, M., "Minerva reference implementation for BRSKI", 2020, <https://minerva.sandelman.ca/>.

- [minervagithub]

"ANIMA Minerva toolkit", <https://github.com/ANIMAgus-minerva>.

- [openssl]

  OpenSSL, "OpenSSL X509 Utility", September 2019, <https://www.openssl.org/docs/man1.1.1/man1/openssl-x509.html/>.

- [RFC2131]

  Droms, R., "Dynamic Host Configuration Protocol", RFC 2131, DOI 10.17487/RFC2131, March 1997, <https://www.rfc-editor.org/info/rfc2131>.

- [RFC2663]

  Srisuresh, P. and M. Holdrege, "IP Network Address Translator (NAT) Terminology and Considerations", RFC 2663, DOI 10.17487/RFC2663, August 1999, <https://www.rfc-editor.org/info/rfc2663>.

- [RFC5209]

  Sangster, P., Khosravi, H., Mani, M., Narayan, K., and J. Tardo, "Network Endpoint Assessment (NEA): Overview and Requirements", RFC 5209, DOI 10.17487/RFC5209, June 2008, <https://www.rfc-editor.org/info/rfc5209>.

- [RFC6960]

  Santesson, S., Myers, M., Ankney, R., Malpani, A., Galperin, S., and C. Adams, "X.509 Internet Public Key Infrastructure Online Certificate Status Protocol - OCSP", RFC 6960, DOI 10.17487/RFC6960, June 2013, <https://www.rfc-editor.org/info/rfc6960>.

- [RFC6961]

  Pettersen, Y., "The Transport Layer Security (TLS) Multiple Certificate Status Request Extension", RFC 6961, DOI 10.17487/RFC6961, June 2013, <https://www.rfc-editor.org/info/rfc6961>.

- [RFC7228]

  Bormann, C., Ersue, M., and A. Keranen, "Terminology for Constrained-Node Networks", RFC 7228, DOI 10.17487/RFC7228, May 2014, <https://www.rfc-editor.org/info/rfc7228>.

- [RFC7258]

  Farrell, S. and H. Tschofenig, "Pervasive Monitoring Is an Attack", BCP 188, RFC 7258, DOI 10.17487/RFC7258, May 2014, <https://www.rfc-editor.org/info/rfc7258>.

- [RFC7435]

  Dukhovni, V., "Opportunistic Security: Some Protection Most of the Time", RFC 7435, DOI 10.17487/RFC7435, December 2014, <https://www.rfc-editor.org/info/rfc7435>.

- [RFC7575]

  Behringer, M., Pritikin, M., Bjarnason, S., Clemm, A., Carpenter, B., Jiang, S., and L. Ciavaglia, "Autonomic Networking: Definitions and Design Goals", RFC 7575, DOI 10.17487/RFC7575, June 2015, <https://www.rfc-editor.org/info/rfc7575>.

- [RFC8126]

  Cotton, M., Leiba, B., and T. Narten, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 8126, DOI 10.17487/RFC8126, June 2017, <https://www.rfc-editor.org/info/rfc8126>.

- [RFC8340]

  Bjorklund, M. and L. Berger, Ed., "YANG Tree Diagrams", BCP 215, RFC 8340, DOI 10.17487/RFC8340, March 2018, <https://www.rfc-editor.org/info/rfc8340>.

- [RFC8615]

  Nottingham, M., "Well-Known Uniform Resource Identifiers (URIs)", RFC 8615, DOI 10.17487/RFC8615, May 2019, <https://www.rfc-editor.org/info/rfc8615>.

- [RFC8993]

  Behringer, M., Ed., Carpenter, B., Eckert, T., Ciavaglia, L., and J. Nobre, "A Reference Model for Autonomic Networking", RFC 8993, DOI 10.17487/RFC8993, May 2021, <https://www.rfc-editor.org/info/rfc8993>.

- [slowloris]

  Wikipedia, "Slowloris (computer security)", January 2021, <https://en.wikipedia.org/w/index.php?title=Slowloris_(computer_security)&oldid=1001473290/>.

- [softwareescrow]

  Wikipedia, "Source code escrow", March 2020, <https://en.wikipedia.org/w/index.php?title=Source_code_escrow&oldid=948073074>.

- [Stajano99theresurrecting]

  Stajano, F. and R. Anderson, "The Resurrecting Duckling: Security Issues for Ad-hoc Wireless Networks", 1999, <https://www.cl.cam.ac.uk/~fms27/papers/1999-StajanoAnd-duckling.pdf>.

- [TR069]

  Broadband Forum, "CPE WAN Management Protocol", TR-069, Issue 1, Amendment 6, March 2018, <https://www.broadband-forum.org/download/TR-069_Amendment-6.pdf>.

- [W3C.capability-urls]

  Tennison, J., "Good Practices for Capability URLs", W3C First Public Working Draft, World Wide Web Consortium WD WD-capability-urls-20140218, February 2014, <https://www.w3.org/TR/2014/WD-capability-urls>.

- [YANG-KEYSTORE]

  Watsen, K., "A YANG Data Model for a Keystore", Work in Progress, Internet-Draft, draft-ietf-netconf-keystore-22, 18 May 2021, <https://tools.ietf.org/html/draft-ietf-netconf-keystore-22>.

Appendix A. IPv4 And Non-ANI Operations

The specification of BRSKI in Section 4 intentionally covers only the mechanisms for an IPv6 pledge using link-local addresses. This section describes non-normative extensions that can be used in other environments.

A.1. IPv4 Link-Local Addresses

Instead of an IPv6 link-local address, an IPv4 address may be generated using "Dynamic Configuration of IPv4 Link-Local Addresses" [RFC3927].

In the case where an IPv4 link-local address is formed, the bootstrap process would continue, as in an IPv6 case, by looking for a (circuit) proxy.

A.2. Use Of DHCPv4

The pledge **MAY** obtain an IP address via DHCP ([RFC2131]. The DHCP-provided parameters for the Domain Name System can be used to perform DNS operations if all local discovery attempts fail.

Appendix B. MDNS / DNS-SD Proxy Discovery Options

Pledge discovery of the proxy (Section 4.1) **MAY** be performed with DNS-based Service Discovery [RFC6763] over Multicast DNS [RFC6762] to discover the proxy at "_brski-proxy._tcp.local.".

Proxy discovery of the registrar (Section 4.3) **MAY** be performed with DNS-based Service Discovery over Multicast DNS to discover registrars by searching for the service "_brski-registrar._tcp.local.".

To prevent unacceptable levels of network traffic, when using mDNS, the congestion avoidance mechanisms specified in [RFC6762], Section 7 **MUST** be followed. The pledge **SHOULD** listen for an unsolicited broadcast response as described in [RFC6762]. This allows devices to avoid announcing their presence via mDNS broadcasts and instead silently join a network by watching for periodic unsolicited broadcast responses.

Discovery of the registrar **MAY** also be performed with DNS-based Service Discovery by searching for the service "_brski-registrar._tcp.example.com". In this case, the domain "example.com" is discovered as described in [RFC6763], Section 11 (Appendix A.2 of this document suggests the use of DHCP parameters).

If no local proxy or registrar service is located using the GRASP mechanisms or the above-mentioned DNS-based Service Discovery methods, the pledge **MAY** contact a well-known manufacturer-provided bootstrapping server by performing a DNS lookup using a well-known URI such as "brski-registrar.manufacturer.example.com". The details of the URI are manufacturer specific. Manufacturers that leverage this method on the pledge are responsible for providing the registrar service. Also see Section 2.7.

The current DNS services returned during each query are maintained until bootstrapping is completed. If bootstrapping fails and the pledge returns to the Discovery state, it picks up where it left off and continues attempting bootstrapping. For example, if the first Multicast DNS _bootstrapks._tcp.local response doesn't work, then the second and third responses are tried. If these fail, the pledge moves on to normal DNS-based Service Discovery.

Appendix C. Example Vouchers

Three entities are involved in a voucher: the MASA issues (signs) it, the registrar's public key is mentioned in it, and the pledge validates it. In order to provide reproducible examples, the public and private keys for an example MASA and registrar are listed first.

The keys come from an open source reference implementation of BRSKI, called "Minerva" [minerva]. It is available on GitHub [minervagithub]. The keys presented here are used in the unit and integration tests. The MASA code is called "highway", the registrar code is called "fountain", and the example client is called "reach".

The public key components of each are presented as base64 certificates and are decoded by openssl's x509 utility so that the extensions can be seen. This was version 1.1.1c of the library and utility of [openssl].

C.1. Keys Involved

The manufacturer has a CA that signs the pledge's IDevID. In addition, the Manufacturer's signing authority (the MASA) signs the vouchers, and that certificate must distributed to the devices at manufacturing time so that vouchers can be validated.

C.1.1. Manufacturer Certification Authority For IDevID Signatures

This private key is the CA that signs IDevID certificates:

```
\<CODE BEGINS> file "vendor.key"

-----BEGIN EC PRIVATE KEY-----
MIGkAgEBBDCAYkoLW1IEA5SKKhMMdkTK7sJxk5ybKqYq9Yr5aR7tNwqXyLGS7z8G
8S4w/UJ58BqgBwYFK4EEACKhZANiAAQu5/yktJbFLjMC87h7b+yTreFuF8GwewKH
L4mS0r0dVAQubqDUQcTrjvpXrXCpTojiLCzgp8fzkcUDkZ9LD/M90LDipiLNIOkP
juF8QkoAbT8pMrY83MS8y76wZ7AalNQ=
-----END EC PRIVATE KEY-----

\<CODE ENDS>
```

This public key validates IDevID certificates:

file: examples/vendor.key

```
\<CODE BEGINS> file "vendor.cert"

Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number: 1216069925 (0x487bc125)
        Signature Algorithm: ecdsa-with-SHA256
        Issuer: CN = highway-test.example.com CA
        Validity
            Not Before: Apr 13 20:34:24 2021 GMT
            Not After : Apr 13 20:34:24 2023 GMT
        Subject: CN = highway-test.example.com CA
        Subject Public Key Info:
            Public Key Algorithm: id-ecPublicKey
                Public-Key: (384 bit)
                pub:
                    04:2e:e7:fc:a4:b4:96:c5:2e:33:02:f3:b8:7b:6f:
                    ec:93:ad:e1:6e:17:c1:b0:7b:02:87:2f:89:92:d2:
                    bd:1d:54:04:2e:6e:a0:d4:41:c4:eb:8e:fa:57:ad:
                    70:a9:4e:88:e2:2c:2c:e0:a7:c7:f3:91:c5:03:91:
                    9f:4b:0f:f3:3d:d0:b0:e2:a6:22:cd:20:e9:0f:8e:
                    e1:7c:42:4a:00:6d:3f:29:32:b6:3c:dc:c4:bc:cb:
                    be:b0:67:b0:1a:94:d4
                ASN1 OID: secp384r1
                NIST CURVE: P-384
        X509v3 extensions:
            X509v3 Basic Constraints: critical
                CA:TRUE
            X509v3 Key Usage: critical
                Certificate Sign, CRL Sign
            X509v3 Subject Key Identifier:
                5E:0C:A9:52:5A:8C:DF:A9:0F:03:14:E9:96:F1:80:76:
                8C:53:8A:08
            X509v3 Authority Key Identifier:
                keyid:5E:0C:A9:52:5A:8C:DF:A9:0F:03:14:E9:96:F1:
                80:76:8C:53:8A:08

    Signature Algorithm: ecdsa-with-SHA256
```

```
         30:64:02:30:60:37:a0:66:89:80:27:e1:0d:e5:43:9a:62:f1:
         02:bc:0f:72:6d:a9:e9:cb:84:a5:c6:44:d3:41:9e:5d:ce:7d:
         46:16:6e:15:de:f7:cc:e8:3e:61:f9:03:7c:20:c4:b7:02:30:
         7f:e9:f3:12:bb:06:c6:24:00:2b:41:aa:21:6b:d8:25:ed:81:
         07:11:ef:66:8f:06:bf:c8:be:f0:58:74:24:45:39:4d:04:fc:
         31:69:6f:cf:db:fe:61:7b:c3:24:31:ff
-----BEGIN CERTIFICATE-----
MIIB3TCCAWSgAwIBAgIESHvBJTAKBggqhkjOPQQDAjAmMSQwIgYDVQQDDBtoaWdo
d2F5LXRlc3QuZXhhbXBsZS5jb20gQ0EwHhcNMjEwNDEzMjAzNDI0WhcNMjMwNDEz
MjAzNDI0WjAmMSQwIgYDVQQDDBtoaWdod2F5LXRlc3QuZXhhbXBsZS5jb20gQ0Ew
djAQBgcqhkjOPQIBBgUrgQQAIgNiAAQu5/yktJbFLjMC87h7b+yTreFuF8GwewKH
L4mS0r0dVAQubqDUQcTrjvpXrXCpTojiLCzgp8fzkcUDkZ9LD/M90LDipiLNIOkP
juF8QkoAbT8pMrY83MS8y76wZ7AalNSjYzBhMA8GA1UdEwEB/wQFMAMBAf8wDgYD
VR0PAQH/BAQDAgEGMB0GA1UdDgQWBBReDKlSWozfqQ8DFOmW8YB2jFOKCDAfBgNV
HSMEGDAWgBReDKlSWozfqQ8DFOmW8YB2jFOKCDAKBggqhkjOPQQDAgNnADBkAjBg
N6BmiYAn4Q3lQ5pi8QK8D3JtqenLhKXGRNNBnl3OfUYWbhXe98zoPmH5A3wgxLcC
MH/p8xK7BsYkACtBqiFr2CXtgQcR72aPBr/IvvBYdCRFOU0E/DFpb8/b/mF7wyQx
/w==
-----END CERTIFICATE-----


\<CODE ENDS>
```

C.1.2. MASA Key Pair For Voucher Signatures

The MASA is the Manufacturer Authorized Signing Authority. This key pair signs vouchers. An example TLS certificate (see Section 5.4) HTTP authentication is not provided as it is a common form.

This private key signs the vouchers that are presented below:

```
\<CODE BEGINS> file "masa.key"

-----BEGIN EC PRIVATE KEY-----
MHcCAQEEIFhdd0eDdzip67kXx72K+KHGJQYJHNy8pkiLJ6CcvxMGoAoGCCqGSM49
AwEHoUQDQgAEqgQVo0S54kT4yfkbBxumdHOcHrpsqbOpMKmiMln3oB1HAW25MJV+
gqi4tMFfSJ0iEwt8kszfWXK4rLgJS2mnpQ==
-----END EC PRIVATE KEY-----

\<CODE ENDS>
```

This public key validates vouchers, and it has been signed by the CA above:

file: examples/masa.key

```
\<CODE BEGINS> file "masa.cert"

Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number: 193399345 (0xb870a31)
        Signature Algorithm: ecdsa-with-SHA256
        Issuer: CN = highway-test.example.com CA
        Validity
            Not Before: Apr 13 21:40:16 2021 GMT
            Not After : Apr 13 21:40:16 2023 GMT
        Subject: CN = highway-test.example.com MASA
        Subject Public Key Info:
            Public Key Algorithm: id-ecPublicKey
                Public-Key: (256 bit)
                pub:
                    04:aa:04:15:a3:44:b9:e2:44:f8:c9:f9:1b:07:1b:
                    a6:74:73:9c:1e:ba:6c:a9:b3:a9:30:a9:a2:32:59:
                    f7:a0:1d:47:01:6d:b9:30:95:7e:82:a8:b8:b4:c1:
                    5f:48:9d:22:13:0b:7c:92:cc:df:59:72:b8:ac:b8:
                    09:4b:69:a7:a5
                ASN1 OID: prime256v1
                NIST CURVE: P-256
        X509v3 extensions:
            X509v3 Basic Constraints: critical
                CA:FALSE
    Signature Algorithm: ecdsa-with-SHA256
         30:66:02:31:00:ae:cb:61:2d:d4:5c:8d:6e:86:aa:0b:06:1d:
         c6:d3:60:ba:32:73:36:25:d3:23:85:49:87:1c:ce:94:23:79:
         1a:9e:41:55:24:1d:15:22:a1:48:bb:0a:c0:ab:3c:13:73:02:
         31:00:86:3c:67:b3:95:a2:e2:e5:f9:ad:f9:1d:9c:c1:34:32:
         78:f5:cf:ea:d5:47:03:9f:00:bf:d0:59:cb:51:c2:98:04:81:
         24:8a:51:13:50:b1:75:b2:2f:9d:a8:b4:f4:b9
-----BEGIN CERTIFICATE-----
MIIBcDCB9qADAgECAgQLhwoxMAoGCCqGSM49BAMCMCYxJDAiBgNVBAMMG2hpZ2h3
YXktdGVzdC5leGFtcGxlLmNvbSBDQTAeFw0yMTA0MTMyMTQwMTZaFw0yMzA0MTMy
MTQwMTZaMCgxJjAkBgNVBAMMHWhpZ2h3YXktdGVzdC5leGFtcGxlLmNvbSBNQVNB
MFkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDQgAEqgQVo0S54kT4yfkbBxumdHOcHrps
```

```
qbOpMKmiMln3oB1HAW25MJV+gqi4tMFfSJ0iEwt8kszfWXK4rLgJS2mnpaMQMA4w
DAYDVR0TAQH/BAIwADAKBggqhkjOPQQDAgNpADBmAjEArsthLdRcjW6GqgsGHcbT
YLoyczYl0yOFSYcczpQjeRqeQVUkHRUioUi7CsCrPBNzAjEAhjxns5Wi4uX5rfkd
nME0Mnj1z+rVRwOfAL/QWctRwpgEgSSKURNQsXWyL52otPS5
-----END CERTIFICATE-----

\<CODE ENDS>
```

C.1.3. Registrar Certification Authority

This CA enrolls the pledge once it is authorized, and it also signs the registrar's certificate.

```
\<CODE BEGINS> file "ownerca_secp384r1.key"

-----BEGIN EC PRIVATE KEY-----
MIGkAgEBBDCHnLI0MSOLf8XndiZqoZdqblcPR5YSoPGhPOuFxWy1gFi9HbWv8b/R
EGdRgGEVSjKgBwYFK4EEACKhZANiAAQbf1m6F8MavGaNjGzgw/oxcQ9l9iKRvbdW
gAfb37h6pUVNeYpGlxlZljGxj2l9Mr48yD5bY7VG9qjVb5v5wPPTuRQ/ckdRpHbd
0vC/9cqPMAF/+MJf0/UgA0SLi/IHbLQ=
-----END EC PRIVATE KEY-----

\<CODE ENDS>
```

The public key is indicated in a pledge voucher-request to show proximity.

file: examples/ownerca_secp384r1.key

```
\<CODE BEGINS> file "ownerca_secp384r1.cert"

Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number: 694879833 (0x296b0659)
        Signature Algorithm: ecdsa-with-SHA256
        Issuer: DC = ca, DC = sandelman,
         CN = fountain-test.example.com Unstrung Fountain Root CA
        Validity
            Not Before: Feb 25 21:31:45 2020 GMT
            Not After : Feb 24 21:31:45 2022 GMT
        Subject: DC = ca, DC = sandelman,
         CN = fountain-test.example.com Unstrung Fountain Root CA
        Subject Public Key Info:
            Public Key Algorithm: id-ecPublicKey
                Public-Key: (384 bit)
                pub:
                    04:1b:7f:59:ba:17:c3:1a:bc:66:8d:8c:6c:e0:c3:
                    fa:31:71:0f:65:f6:22:91:bd:b7:56:80:07:db:df:
                    b8:7a:a5:45:4d:79:8a:46:97:19:59:96:31:b1:8f:
                    69:7d:32:be:3c:c8:3e:5b:63:b5:46:f6:a8:d5:6f:
                    9b:f9:c0:f3:d3:b9:14:3f:72:47:51:a4:76:dd:d2:
                    f0:bf:f5:ca:8f:30:01:7f:f8:c2:5f:d3:f5:20:03:
                    44:8b:8b:f2:07:6c:b4
                ASN1 OID: secp384r1
                NIST CURVE: P-384
        X509v3 extensions:
            X509v3 Basic Constraints: critical
                CA:TRUE
            X509v3 Key Usage: critical
                Certificate Sign, CRL Sign
            X509v3 Subject Key Identifier:
                B9:A5:F6:CB:11:E1:07:A4:49:2C:A7:08:C6:7C:10:BC:
                87:B3:74:26
            X509v3 Authority Key Identifier:
                keyid:B9:A5:F6:CB:11:E1:07:A4:49:2C:A7:08:C6:7C:
                10:BC:87:B3:74:26

    Signature Algorithm: ecdsa-with-SHA256
         30:64:02:30:20:83:06:ce:8d:98:a4:54:7a:66:4c:4a:3a:70:
         c2:52:36:5a:52:8d:59:7d:20:9b:2a:69:14:58:87:38:d8:55:
         79:dd:fd:29:38:95:1e:91:93:76:b4:f5:66:29:44:b4:02:30:
         6f:38:f9:af:12:ed:30:d5:85:29:7c:b1:16:58:bd:67:91:43:
         c4:0d:30:f9:d8:1c:ac:2f:06:dd:bc:d5:06:42:2c:84:a2:04:
         ea:02:a4:5f:17:51:26:fb:d9:2f:d2:5c
```

```
-----BEGIN CERTIFICATE-----
MIICazCCAfKgAwIBAgIEKWsGWTAKBggqhkjOPQQDAjBtMRIwEAYKCZImiZPyLGQB
GRYCY2ExGTAXBgoJkiaJk/IsZAEZFglzYW5kZWxtYW4xPDA6BgNVBAMMM2ZvdW50
YWluLXRlc3QuZXhhbXBsZS5jb20gVW5zdHJ1bmcgRm91bnRhaW4gUm9vdCBDQTAe
Fw0yMDAyMjUyMTMxNDVaFw0yMjAyMjQyMTMxNDVaMG0xEjAQBgoJkiaJk/IsZAEZ
FgJjYTEZMBcGCgmSJomT8ixkARkWCXNhbmRlbG1hbjE8MDoGA1UEAwwzZm91bnRh
aW4tdGVzdC5leGFtcGxlLmNvbSBVbnN0cnVuZyBGb3VudGFpbiBSb290IENBMHYw
EAYHKoZIzj0CAQYFK4EEACIDYgAEG39ZuhfDGrxmjYxs4MP6MXEPZfYikb23VoAH
```

```
29+4eqVFTXmKRpcZWZYxsY9pfTK+PMg+W2O1Rvao1W+b+cDz07kUP3JHUaR23dLw
v/XKjzABf/jCX9P1IANEi4vyB2y0o2MwYTAPBgNVHRMBAf8EBTADAQH/MA4GA1Ud
DwEB/wQEAwIBBjAdBgNVHQ4EFgQUuaX2yxHhB6RJLKcIxnwQvIezdCYwHwYDVR0j
BBgwFoAUuaX2yxHhB6RJLKcIxnwQvIezdCYwCgYIKoZIzj0EAwIDZwAwZAIwIIMG
zo2YpFR6ZkxKOnDCUjZaUo1ZfSCbKmkUWIc42FV53f0pOJUekZN2tPVmKUS0AjBv
OPmvEu0w1YUpfLEWWL1nkUPEDTD52BysLwbdvNUGQiyEogTqAqRfF1Em+9kv0lw=
-----END CERTIFICATE-----

\<CODE ENDS>
```

C.1.4. Registrar Key Pair

The registrar is the representative of the domain owner. This key signs registrar voucher-requests and terminates the TLS connection from the pledge.

```
\<CODE BEGINS> file "jrc_prime256v1.key"

-----BEGIN EC PRIVATE KEY-----
MHcCAQEEIFZodk+PC5Mu24+ra0sbOjKzan+dW5rvDAR7YuJUOC1YoAoGCCqGSM49
AwEHoUQDQgAElmVQcjS6n+Xd5l/28IFv6UiegQwSBztGj5dkK2MAjQIPV8l8lH+E
jLIOYdbJiI0VtEIf1/Jqt+TOBfinTNOLOg==
-----END EC PRIVATE KEY-----

\<CODE ENDS>
```

The public key is indicated in a pledge voucher-request to show proximity.

```
\<CODE BEGINS> file "jrc_prime256v1.cert"

Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number: 1066965842 (0x3f989b52)
        Signature Algorithm: ecdsa-with-SHA256
        Issuer: DC = ca, DC = sandelman,
         CN = fountain-test.example.com Unstrung Fountain Root CA
        Validity
            Not Before: Feb 25 21:31:54 2020 GMT
            Not After : Feb 24 21:31:54 2022 GMT
        Subject: DC = ca, DC = sandelman,
         CN = fountain-test.example.com
        Subject Public Key Info:
            Public Key Algorithm: id-ecPublicKey
                Public-Key: (256 bit)
                pub:
                    04:96:65:50:72:34:ba:9f:e5:dd:e6:5f:f6:f0:81:
                    6f:e9:48:9e:81:0c:12:07:3b:46:8f:97:64:2b:63:
                    00:8d:02:0f:57:c9:7c:94:7f:84:8c:b2:0e:61:d6:
                    c9:88:8d:15:b4:42:1f:d7:f2:6a:b7:e4:ce:05:f8:
                    a7:4c:d3:8b:3a
                ASN1 OID: prime256v1
                NIST CURVE: P-256
        X509v3 extensions:
            X509v3 Extended Key Usage: critical
                CMC Registration Authority
            X509v3 Key Usage: critical
                Digital Signature
    Signature Algorithm: ecdsa-with-SHA256
         30:65:02:30:66:4f:60:4c:55:48:1e:96:07:f8:dd:1f:b9:c8:
         12:8d:45:36:87:9b:23:c0:bc:bb:f1:cb:3d:26:15:56:6f:5f:
         1f:bf:d5:1c:0e:6a:09:af:1b:76:97:99:19:23:fd:7e:02:31:
         00:bc:ac:c3:41:b0:ba:0d:af:52:f9:9c:6e:7a:7f:00:1d:23:
         c8:62:01:61:bc:4b:c5:c0:47:99:35:0a:0c:77:61:44:01:4a:
         07:52:70:57:00:75:ff:be:07:0e:98:cb:e5
-----BEGIN CERTIFICATE-----
MIIB/DCCAYKgAwIBAgIEP5ibUjAKBggqhkjOPQQDAjBtMRIwEAYKCZImiZPyLGQB
GRYCY2ExGTAXBgoJkiaJk/IsZAEZFglzYW5kZWxtYW4xPDA6BgNVBAMMM2ZvdW50
YWluLXRlc3QuZXhhbXBsZS5jb20gVW5zdHJ1bmcgRm91bnRhaW4gUm9vdCBDQTAe
Fw0yMDAyMjUyMTMxNTRaFw0yMjAyMjQyMTMxNTRaMFMxEjAQBgoJkiaJk/IsZAEZ
FgJjYTEZMBcGCgmSJomT8ixkARkWCXNhbmRlbG1hbjEiMCAGA1UEAwwZZm91bnRh
aW4tdGVzdC5leGFtcGxlLmNvbTBZMBMGByqGSM49AgEGCCqGSM49AwEHA0IABJZl
UHI0up/l3eZf9vCBb+lInoEMEgc7Ro+XZCtjAI0CD1fJfJR/hIyyDmHWyYiNFbRC
H9fyarfkzgX4p0zTizqjKjAoMBYGA1UdJQEB/wQMMAoGCCsGAQUFBwMcMA4GA1Ud
DwEB/wQEAwIHgDAKBggqhkjOPQQDAgNoADBlAjBmT2BMVUgelgf43R+5yBKNRTaH
myPAvLvxyz0mFVZvXx+/1RwOagmvG3aXmRkj/X4CMQC8rMNBsLoNr1L5nG56fwAd
I8hiAWG8S8XAR5k1Cgx3YUQBSgdScFcAdf++Bw6Yy+U=
-----END CERTIFICATE-----

\<CODE ENDS>
```

C.1.5. Pledge Key Pair

The pledge has an IDevID key pair built in at manufacturing time:

```
\<CODE BEGINS> file "idevid_00-D0-E5-F2-00-02.key"

-----BEGIN EC PRIVATE KEY-----
MHcCAQEEIBHNh6r8QRevRuo+tEmBJeFjQKf6bpFA/9NGoltv+9sNoAoGCCqGSM49
AwEHoUQDQgAEA6N1Q4ezfMAKmoecrfb0OBMc1AyEH+BATkF58FsTSyBxs0SbSWLx
FjDOuwB9gLGn2TsTUJumJ6VPw5Z/TP4hJw==
-----END EC PRIVATE KEY-----

\<CODE ENDS>
```

The certificate is used by the registrar to find the MASA.

```
\<CODE BEGINS> file "idevid_00-D0-E5-F2-00-02.cert"

Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number: 521731815 (0x1f18fee7)
        Signature Algorithm: ecdsa-with-SHA256
        Issuer: CN = highway-test.example.com CA
        Validity
            Not Before: Apr 27 18:29:30 2021 GMT
            Not After : Dec 31 00:00:00 2999 GMT
        Subject: serialNumber = 00-D0-E5-F2-00-02
        Subject Public Key Info:
            Public Key Algorithm: id-ecPublicKey
                Public-Key: (256 bit)
                pub:
                    04:03:a3:75:43:87:b3:7c:c0:0a:9a:87:9c:ad:f6:
                    f4:38:13:1c:d4:0c:84:1f:e0:40:4e:41:79:f0:5b:
                    13:4b:20:71:b3:44:9b:49:62:f1:16:30:ce:bb:00:
                    7d:80:b1:a7:d9:3b:13:50:9b:a6:27:a5:4f:c3:96:
                    7f:4c:fe:21:27
                ASN1 OID: prime256v1
                NIST CURVE: P-256
        X509v3 extensions:
            X509v3 Subject Key Identifier:
                45:88:CC:96:96:00:64:37:B0:BA:23:65:64:64:54:08:
                06:6C:56:AD
            X509v3 Basic Constraints:
                CA:FALSE
            1.3.6.1.5.5.7.1.32:
                ..highway-test.example.com:9443
    Signature Algorithm: ecdsa-with-SHA256
         30:65:02:30:62:2a:db:be:34:f7:1b:cb:85:de:26:8e:43:00:
         f9:0d:88:c8:77:a8:dd:3c:08:40:54:bc:ec:3d:b6:dc:70:2b:
         c3:7f:ca:19:21:9a:a0:ab:c5:51:8e:aa:df:36:de:8b:02:31:
         00:b2:5d:59:f8:47:c7:ed:03:97:a8:c0:c7:a8:81:fa:a8:86:
         ed:67:64:37:51:7a:6e:9c:a3:82:4d:6d:ad:bc:f3:35:9e:9d:
         6a:a2:6d:7f:7f:25:1c:03:ef:f0:ba:9b:71
-----BEGIN CERTIFICATE-----
MIIBrzCCATWgAwIBAgIEHxj+5zAKBggqhkjOPQQDAjAmMSQwIgYDVQQDDBtoaWdo
d2F5LXRlc3QuZXhhbXBsZS5jb20gQ0EwIBcNMjEwNDI3MTgyOTMwWhgPMjk5OTEy
MzEwMDAwMDBaMBwxGjAYBgNVBAUTETAwLUQwLUU1LUYyLTAwLTAyMFkwEwYHKoZI
zj0CAQYIKoZIzj0DAQcDQgAEA6N1Q4ezfMAKmoecrfb0OBMc1AyEH+BATkF58FsT
SyBxs0SbSWLxFjDOuwB9gLGn2TsTUJumJ6VPw5Z/TP4hJ6NZMFcwHQYDVR0OBBYE
FEWIzJaWAGQ3sLojZWRkVAgGbFatMAkGA1UdEwQCMAAwKwYIKwYBBQUHASAEHxYd
aGlnaHdheS10ZXN0LmV4YW1wbGUuY29tOjk0NDMwCgYIKoZIzj0EAwIDaAAwZQIw
YirbvjT3G8uF3iaOQwD5DYjId6jdPAhAVLzsPbbccCvDf8oZIZqgq8VRjqrfNt6L
AjEAsl1Z+EfH7QOXqMDHqIH6qIbtZ2Q3UXpunKOCTW2tvPM1np1qom1/fyUcA+/w
uptx
-----END CERTIFICATE-----



\<CODE ENDS>
```

C.2. Example Process

The JSON examples below are wrapped at 60 columns. This results in strings that have newlines in them, which makes them invalid JSON as is. The strings would otherwise be too long, so they need to be unwrapped before processing.

For readability, the output of the asn1parse has been truncated at 68 columns rather than wrapped.

C.2.1. Pledge To Registrar

As described in Section 5.2, the pledge will sign a pledge voucher-request containing the registrar's public key in the proximity-registrar-cert field. The base64 has been wrapped at 60 characters for presentation reasons.

```
\<CODE BEGINS> file "vr_00-D0-E5-F2-00-02.b64"

MIIGcAYJKoZIhvcNAQcCoIIGYTCCBl0CAQExDTALBglghkgBZQMEAgEwggOJBgkqhkiG
9w0BBwGgggN6BIIDdnsiaWV0Zi12b3VjaGVyLXJlcXVlc3Q6dm91Y2hlciI6eyJhc3Nl
cnRpb24iOiJwcm94aW1pdHkiLCJjcmVhdGVkLW9uIjoiMjAyMS0wNC0xM1QxNzo0Mzoy
My43NDctMDQ6MDAiLCJzZXJpYWwtbnVtYmVyIjoiMDAtRDAtRTUtRjItMDAtMDIiLCJu
b25jZSI6Ii1fWEU5eks5cThMbDFxeWxNdExLZWciLCJwcm94aW1pdHktcmVnaXN0cmFy
LWNlcnQiOiJNSUlCL0RDQ0FZS2dBd0lCQWdJRVA1aWJVakFLQmdncWhrak9QUVFEQWpC
dE1SSXdFQVlLQ1pJbWlaUHlMR1FCR1JZQ1kyRXhHVEFYQmdvSmtpYUprL0lzWkFFWkZn
bHpZVzVrVWld4dFlXNHhQREE2QmdOVkJBTU1NMlp2ZFc1MFlXbHVMWFJsYzNRdVpYaGhi
WEJzWlM1amIyMGdWVzV6ZEhKMWJtY2dSbTkxYm5SaGFXNGdVbTl2ZENRFFUQWVGdzB5
TURBeU1qVXlNVE14TlRSYUZ3MHlNakF5TWpReU1UTXhOVFJhTUZNeEVqQVFCZ29Ka2lh
SmsvSXNaQUVaRmdKallURVpNQmNHQ2dtU0pvbVQ4aXhrQVJrV0NYTmhibVJsYkcxaGJq
RWlNQ0FHQTFVRUF3d1pabTkxYm5SaGFXNHRkR1Z6ZEM1bGVHRnRjR3hsTG1OdmJUQlpN
Qk1HQnlxR1NNNDlBZ0VHQ0NxR1NNNDlBd0VIQTBJQUJKWmxVSEkwdXAvbDNlWmY5dkNC
YitsSW5vRU1FZ2M3Um8rWFpDdGpBSTBDRDFmSmZKUi9oSXl5RG1IV3lZaU5GYlJDSDlm
eWFyZmt6Z1g0cDB6VGl6cWpLakFvTUJZR0ExVWRKUUVCL3dRTU1Bb0dDQ3NHQVFVRkJ3
TWNNQTRHQTFVZER3RUIvd1FFQXdJSGdEQUtCZ2dxaGtqT1BRUURBZ05vQURCbEFqQm1U
MkJNVlVnZWxnZjQzUis1eUJLTlJUYUhteVBBdkx2eHl6MG1GVlp2WHgrLzFSd09hZ212
RzNhWG1Sa2ovWDRDTVFDOHJNTkJzTG9OcjFMNW5HNTZmd0FkSThoaUFXRzhTOFhBUjVr
MUNNeDNZVVFCU2dkU2NGY0FkZisrQnc2WXkrVT0ifX2gggGyMIIBrjCCATWgAwIBAgIE
DYOv2TAKBggqhkjOPQQDAjAmMSQwIgYDVQQDDBtoaWdod2F5LXRlc3QuZXhhbXBsZS5j
b20gQ0EwIBcNMjEwNDEzMjAzNzM5WhgPMjk5OTEyMzEwMDAwMDBaMBwxGjAYBgNVBAUM
ETAwLUQwLUU1LUYyLTAwLTAyMFkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDQgAEA6N1Q4ez
fMAKmoecrfb0OBMc1AyEH+BATkF58FsTSyBxs0SbSWLxFjDOuwB9gLGn2TsTUJumJ6VP
w5Z/TP4hJ6NZMFcwHQYDVR0OBBYEFEWIzJaWAGQ3sLojZWRkVAgGbFatMAkGA1UdEwQC
MAAwKwYIKwYBBQUHASAEHxYdaGlnaHdheS10ZXN0LmV4YW1wbGUuY29tOjk0NDMwCgYI
KoZIzj0EAwIDZwAwZAIwTmlG8sXkKGNbwbKQcYMapFbmSbnHHURFUoFuRqvbgYX7FlXp
BczfwF2kllNuujigAjAow1kc4r55EmiH+OMEXjBNlWlBSZC5QuJjEf0Jsmxssc+pucjO
J4ShqnexMEy7bjAxggEEMIIBAAIBATAuMCYxJDAiBgNVBAMMG2hpZ2h3YXktdGVzdC5l
eGFtcGxlLmNvbSBDQQIEDYOv2TALBglghkgBZQMEAgGgaTAYBgkqhkiG9w0BCQMxCwYJ
KoZIhvcNAQcBMBwGCSqGSIb3DQEJBTEPFw0yMTA0MTMyMTQzMjNaMC8GCSqGSIb3DQEJ
BDEiBCBJwhyYibIjeqeR3bOaLURzMlGrc3F2X+kvJ1errtoCtTAKBggqhkjOPQQDAgRH
MEUCIQCmYuCE61HFQXH/E16GDOCsVquDtgr+Q/6/Du/9QkzA7gIgf7MFhAIPW2PNwRa2
vZFQAKXUbimkiHKzXBA8md0VHbU=

\<CODE ENDS>
```

The ASN1 decoding of the artifact:

file: examples/vr_00-D0-E5-F2-00-02.b64

```
    0:d=0  hl=4 l=1648 cons: SEQUENCE
    4:d=1  hl=2 l=   9 prim: OBJECT            :pkcs7-signedData
   15:d=1  hl=4 l=1633 cons: cont [ 0 ]
   19:d=2  hl=4 l=1629 cons: SEQUENCE
   23:d=3  hl=2 l=   1 prim: INTEGER           :01
   26:d=3  hl=2 l=  13 cons: SET
   28:d=4  hl=2 l=  11 cons: SEQUENCE
   30:d=5  hl=2 l=   9 prim: OBJECT            :sha256
   41:d=3  hl=4 l= 905 cons: SEQUENCE
   45:d=4  hl=2 l=   9 prim: OBJECT            :pkcs7-data
   56:d=4  hl=4 l= 890 cons: cont [ 0 ]
   60:d=5  hl=4 l= 886 prim: OCTET STRING      :{"ietf-voucher-
request:v
  950:d=3  hl=4 l= 434 cons: cont [ 0 ]
  954:d=4  hl=4 l= 430 cons: SEQUENCE
  958:d=5  hl=4 l= 309 cons: SEQUENCE
  962:d=6  hl=2 l=   3 cons: cont [ 0 ]
  964:d=7  hl=2 l=   1 prim: INTEGER           :02
  967:d=6  hl=2 l=   4 prim: INTEGER           :0D83AFD9
  973:d=6  hl=2 l=  10 cons: SEQUENCE
  975:d=7  hl=2 l=   8 prim: OBJECT            :ecdsa-with-SHA256
  985:d=6  hl=2 l=  38 cons: SEQUENCE
  987:d=7  hl=2 l=  36 cons: SET
  989:d=8  hl=2 l=  34 cons: SEQUENCE
  991:d=9  hl=2 l=   3 prim: OBJECT            :commonName
  996:d=9  hl=2 l=  27 prim: UTF8STRING        :highway-
test.example.com
 1025:d=6  hl=2 l=  32 cons: SEQUENCE
 1027:d=7  hl=2 l=  13 prim: UTCTIME           :210413203739Z
 1042:d=7  hl=2 l=  15 prim: GENERALIZEDTIME   :29991231000000Z
 1059:d=6  hl=2 l=  28 cons: SEQUENCE
 1061:d=7  hl=2 l=  26 cons: SET
 1063:d=8  hl=2 l=  24 cons: SEQUENCE
 1065:d=9  hl=2 l=   3 prim: OBJECT            :serialNumber
 1070:d=9  hl=2 l=  17 prim: UTF8STRING        :00-D0-E5-F2-00-02
 1089:d=6  hl=2 l=  89 cons: SEQUENCE
 1091:d=7  hl=2 l=  19 cons: SEQUENCE
 1093:d=8  hl=2 l=   7 prim: OBJECT            :id-ecPublicKey
 1102:d=8  hl=2 l=   8 prim: OBJECT            :prime256v1
 1112:d=7  hl=2 l=  66 prim: BIT STRING
 1180:d=6  hl=2 l=  89 cons: cont [ 3 ]
 1182:d=7  hl=2 l=  87 cons: SEQUENCE
 1184:d=8  hl=2 l=  29 cons: SEQUENCE
 1186:d=9  hl=2 l=   3 prim: OBJECT            :X509v3 Subject Key
Ident
 1191:d=9  hl=2 l=  22 prim: OCTET STRING      [HEX
DUMP]:04144588CC9696
 1215:d=8  hl=2 l=   9 cons: SEQUENCE
 1217:d=9  hl=2 l=   3 prim: OBJECT            :X509v3 Basic
Constraints
 1222:d=9  hl=2 l=   2 prim: OCTET STRING      [HEX DUMP]:3000
 1226:d=8  hl=2 l=  43 cons: SEQUENCE
 1228:d=9  hl=2 l=   8 prim: OBJECT            :1.3.6.1.5.5.7.1.32
 1238:d=9  hl=2 l=  31 prim: OCTET STRING      [HEX
DUMP]:161D6869676877
 1271:d=5  hl=2 l=  10 cons: SEQUENCE
 1273:d=6  hl=2 l=   8 prim: OBJECT            :ecdsa-with-SHA256
 1283:d=5  hl=2 l= 103 prim: BIT STRING
 1388:d=3  hl=4 l= 260 cons: SET
 1392:d=4  hl=4 l= 256 cons: SEQUENCE
 1396:d=5  hl=2 l=   1 prim: INTEGER           :01
 1399:d=5  hl=2 l=  46 cons: SEQUENCE
 1401:d=6  hl=2 l=  38 cons: SEQUENCE
 1403:d=7  hl=2 l=  36 cons: SET
 1405:d=8  hl=2 l=  34 cons: SEQUENCE
 1407:d=9  hl=2 l=   3 prim: OBJECT            :commonName
 1412:d=9  hl=2 l=  27 prim: UTF8STRING        :highway-
test.example.com
 1441:d=6  hl=2 l=   4 prim: INTEGER           :0D83AFD9
 1447:d=5  hl=2 l=  11 cons: SEQUENCE
 1449:d=6  hl=2 l=   9 prim: OBJECT            :sha256
 1460:d=5  hl=2 l= 105 cons: cont [ 0 ]
 1462:d=6  hl=2 l=  24 cons: SEQUENCE
 1464:d=7  hl=2 l=   9 prim: OBJECT            :contentType
 1475:d=7  hl=2 l=  11 cons: SET
 1477:d=8  hl=2 l=   9 prim: OBJECT            :pkcs7-data
 1488:d=6  hl=2 l=  28 cons: SEQUENCE
 1490:d=7  hl=2 l=   9 prim: OBJECT            :signingTime
 1501:d=7  hl=2 l=  15 cons: SET
 1503:d=8  hl=2 l=  13 prim: UTCTIME           :210413214323Z
 1518:d=6  hl=2 l=  47 cons: SEQUENCE
 1520:d=7  hl=2 l=   9 prim: OBJECT            :messageDigest
 1531:d=7  hl=2 l=  34 cons: SET
 1533:d=8  hl=2 l=  32 prim: OCTET STRING      [HEX
DUMP]:49C21C9889B223
 1567:d=5  hl=2 l=  10 cons: SEQUENCE
 1569:d=6  hl=2 l=   8 prim: OBJECT            :ecdsa-with-SHA256
 1579:d=5  hl=2 l=  71 prim: OCTET STRING      [HEX
DUMP]:3045022100A662
```

The JSON contained in the voucher-request:

```
{"ietf-voucher-request:voucher":{"assertion":"proximity","cr
eated-on":"2021-04-13T17:43:23.747-04:00","serial-number":"0
0-D0-E5-F2-00-02","nonce":"-_XE9zK9q8Ll1qylMtLKeg","proximit
y-registrar-cert":"MIIB/DCCAYKgAwIBAgIEP5ibUjAKBggqhkjOPQQDA
jBtMRIwEAYKCZImiZPyLGQBGRYCY2ExGTAXBgoJkiaJk/IsZAEZFglzYW5kZ
WxtYW4xPDA6BgNVBAMMM2ZvdW50YWluLXRlc3QuZXhhbXBsZS5jb20gVW5zd
HJ1bmcgRm91bnRhaW4gUm9vdCBDQTAeFw0yMDAyMjUyMTMxNTRaFw0yMjAyM
jQyMTMxNTRaMFMxEjAQBgoJkiaJk/IsZAEZFgJjYTEZMBcGCgmSJomT8ixkA
RkWCXNhbmRlbG1hbjEiMCAGA1UEAwwZZm91bnRhaW4tdGVzdC5leGFtcGxlL
mNvbTBZMBMGByqGSM49AgEGCCqGSM49AwEHA0IABJZlUHI0up/l3eZf9vCBb
+lInoEMEgc7Ro+XZCtjAI0CD1fJfJR/hIyyDmHWyYiNFbRCH9fyarfkzgX4p
0zTizqjKjAoMBYGA1UdJQEB/wQMMAoGCCsGAQUFBwMcMA4GA1UdDwEB/wQEA
wIHgDAKBggqhkjOPQQDAgNoADBlAjBmT2BMVUgelgf43R+5yBKNRTaHmyPAv
Lvxyz0mFVZvXx+/1RwOagmvG3aXmRkj/X4CMQC8rMNBsLoNr1L5nG56fwAdI
8hiAWG8S8XAR5k1Cgx3YUQBSgdScFcAdf++Bw6Yy+U="}}
```

C.2.2. Registrar To MASA

As described in Section 5.5, the registrar will sign a registrar voucher-request and will include the pledge's voucher-request in the prior-signed-voucher-request.

```
\<CODE BEGINS> file "parboiled_vr_00-D0-E5-F2-00-02.b64"

MIIPYwYJKoZIhvcNAQcCoIIPVDCCD1ACAQExDTALBglghkgBZQMEAgEwggl4BgkqhkiG
9w0BBwGgggl pBIIJZXsiaWV0Zi12b3VjaGVyLXJlcXVlc3Q6dm91Y2hlciI6eyJhc3Nl
cnRpb24iOiJwcm94aW1pdHkiLCJjcmVhdGVkLW9uIjoiMjAyMS0wNC0xM1QyMTo0Mzoy
My43ODdaIiwic2VyaWFsLW51bWJlciI6IjAwLUQwLUU1LUYyLTAwLTAyIiwibm9uY2Ui
OiItX1hFOXpLOXE4TGwxcXlsTXRMS2VnIiwicHJpb3Itc2lnbmVkLXZvdWNoZXItcmVx
dWVzdCI6Ik1JSUdjQVlKS29aSWh2Y05BUWNDb0lJR1lUQ0NCbDBDQVFFeERUQUxCZ2xn
```

```
aGtnQlpRTUVBZ0V3Z2dPSkJna3Foa2lHOXcwQkJ3R2dnZ042QklJRGRuc2lhV1YwWmkx
MmIzVmphR1Z5TFhKbGNYVmxjM1E2ZG05MVkyaGxjaUk2ZXlKaGMzTmxjblJwYjI0aU9p
SndjbTk0YVcxcGRIa2lMQ0pqY21WaGRHVmtMVzl1SWpvaU1qQXlNUzB3TkMweE0xUXhO
em8wTXpveU15NDNORGN0TURRNk1EQWlMQ0p6WlhKcFlXd3RiblZ0WW1WeUlqb2lNREF0
UkRBdFJUVXRSakl0TURBdE1ESWlMQ0p1YjI1alpTSTZJaTFmV0VVNWVrczVjVGhNYkRG
eGVXeE5kRXhMWldjaUxDSndjbTk0YVcxcGRIa3RjbVZuYVhOMGNtRnlMV05sY25RaU9p
Sk5TVWxDTDBSRFEwRlpTMmRCZDBsQ1FXZEpSVkExYVdKVmFrRkxRbWRuY1docmFrOVFV
VkZFUVdwQ2RFMVNTWGRGUVZsTFExcEpiV2xhVUhsTVIxRkNSMUpaUTFreVJYYaEhWRUZZ
UW1kdlNtdHBZVXByTDBseldrRkZXa1puYkhwWlZ6VnJXbGQ0ZEZsWE5IaFFSUUyUW1k
T1ZrSkJUVTFOTWxwMlpGYzFNRmxYYkhWTVdGSnNZek5SZFZwWWFHaGxXRUp6V2xNMWFt
SXlNR2RXVnpwWNlpFaEtNV0p0WTJkU2JUa3hZbTVTTVRYUdGWE5HZFZiVGwyWkVOQ1JGRlVR
V1ZHZHpCNVRVUkJlVTFxVlhsTlZFMTRUbFJTWVVaM01IbE5ha0Y1VFdwUmVVMVVVWGhP
VkZKaFRVWk5lRVZxUVZGQ1oyOUthMmxoU21zdlNYTmFRVVZhUm1kS2FsbFVSVnBOUW1O
SFEyZHRVMHB2YlZRNGFYaHJRVkpyVjBOWVRtaGliVkpzWWtjeGFHSnFSV2xOUTBGSFFU
RlZSVUYzZDFwYWJUa3hZbTVTTVTYUdGWE5IUmtSMVo2WkVNMWJHVkhhSblJqUjNoc1RHMU9k
bUpVWWxwTlFrRWUhRbmx4UjFOTk5EbEJaMFZZIUTBOeFIxTk5ORGxCZDBWSVFUQkpRVUpL
V214VlNFa3dkWEF2YkROOWFdtWTVka05DWWw0c1NXNXZTVTFGWjJNM1Z0T0hJWFJnQkVZRWR3
QlNUQkRSREZtU21aS1VpOW9TWGw1UkcxSVYzbFphVlTVHWWxKRFNEbG1lV0Z5Wm10Nlox
ZzBjREI2VkdsNmNXcExha0Z2VFVKWlIwRXhWV1JLVVVWWQ0wzZFJUVTFCYjBkRFEzTkhR
VkZWUmtKM1RXk5RVFJIUVRGVlpFUjNSVUl2ZDFGR1RlFYZEpTR2RFUVV0Q1oyZHhhR3Rx
VDFCUlVVUkJaMDV2UVVSQ2JFRnFFRbTFVTWtKTlZsVm5aV3huWmpSelVpczFlVUpMVGxK
VVlVaHRlVkJCZGt4MmVIbDZNRHFHVmxwMldIZ3JMekZTZDA5aFoyMTJSek5oV0cxU2Ey
b3ZXRFJEVFZGRE9ISk5Ua0p6VEc5T2NqRk1OVzVJTlRabWQwRmtTVGhvYVVGWFJ6aFRP
RmhCVWpWck1VTm5lRE5aVlZGQ1UyZGtVMk5HWTBCR2ExcHpc3JRbmMyV1hrclZUMGlmWDJn
Z2dHeU1JSUJyakNDQVRXZ0F3SUJBZ0lFRFlPdjJUQUtCZ2dxaGtqT1BRUURBakFtTVNR
d0lnWURWUVFEREJ0b2FXZG9kMkY1TFhSbGMzUXVaWGhoYlhCc1pTNWpiMjBnUTBBFd0lC
Y05NakV3TkRFek1qQXpOek01V2hnUE1qazVPVEV5TXpFd01EQXdNREJhTUJ3eEdqQVlC
Z05WQkFVTUVVQXdMVVF3TFVVMUxVWXlMVEF3TFRBeU1Ga3dFd1lIS29aSXpqMENBUVlJ
S29aSXpqMERBUWNEUWdBRUE2TjFRNGV6Zk1BS21vZWNyZmIwT0JNYzFBeUVIK0JBVGtG
NThGc1RTeUJ4czBTYlNXTHhGakRPdXdCOWdMR24yVHNUVUp1bUo2VlB3NVovVFA0aEo2
TlpNRmN3SFFZRFZSME9CQllFRkVXSXpKYVdBR1Ezc0xvalpXUmtWQWdHYkZhdE1Ba0dB
MVVkRXdRQ01BQXdLd1lJS3dZQkJRVUhBU0FFFSHhZZGFHbG5hSGRoZVMxMFpYTjBMbVY0
WVcxd2JHVXVZMjl0T2prME5ETXdDZ1lJS29aSXpqMEVBd0lEWndBd1pBSXdUbWxHOHNY
a0tHTmJ3YktRY1lNYXBGYm1TYm5ISFVSRlVvRnVScXZiZ1lYN0ZsWHBCY3pmd0Yya2xs
TnV1amlnQWpBb3cxa2M0cjU1RW1pSCtPTUVYakJObFdsQlNaQzVRdUpqRWYwSnNNteHNz
YytwdWNqT0o0U2hxbmV4TUV5N2JqQXhnZ0VFTUlJQkFBSUJBVEF1TUNZeEpEQWlCZ05W
QkFNTUcyaHBaMmgzWVhrdGRHRHVnpkQzVsZUdGdGNHeGxMbU52YlNCRFFRSUVEWU92MlRB
TEJnbGdoa2dCWlFNRUFnR2dhVEFZQmdrcWhraUc5dzBCQ1FNeEN3WUpLb1pJaHZjTkFR
Y0JNQndHQ1NxR1NJYjNEUUVKQlFFUEZ3MHlNVEEwTVRNeU1UUXpNak5hTUM4R0NTcUdT
SWIzRFFFSkJERWlCQ0JKd2h5WWliSWplcWVSM2JPYUxVUnpNbEdyYzNGMlgra3ZKMWVy
cnRvQ3RUQUtCZ2dxaGtqT1BRUURBZ1JJTUVVQ0lRQ21ZdUNFNjFIRlFYSC9FMTZHRE9D
c1ZxdUR0Z3IrUS82L0R1LzlRa3pBN2dJZ2Y3TUZoQUlQVzJQTndSYTJ2WkZRQUtYVWJp
```

```
bWtpSEt6WEJBOG1kMFZIYlU9In19oIIEbzCCAfwwggGCoAMCAQICBD+Ym1IwCgYIKoZI
zj0EAwIwbTESMBAGCgmSJomT8ixkARkWAmNhMRkwFwYKCZImiZPyLGQBGRYJc2FuZGVs
bWFuMTwwOgYDVQQDDDNmb3VudGFpbi10ZXN0LmV4YW1wbGUuY29tIFVuc3RydW5nIEZv
dW50YWluIFJvb3QgQ0EwHhcNMjAwMjI1MjEzMTU0WhcNMjIwMjI0MjEzMTU0WjBTMRIw
EAYKCZImiZPyLGQBGRYCY2ExGTAXBgoJkiaJk/IsZAEZFglzYW5kZWxtYW4xIjAgBgNV
BAMMGWZvdW50YWluLXRlc3QuZXhhbXBsZS5jb20wWTATBgcqhkjOPQIBBggqhkjOPQMB
BwNCAASWZVByNLqf5d3mX/bwgW/pSJ6BDBIHO0aPl2QrYwCNAg9XyXyUf4SMsg5h1smI
jRW0Qh/X8mq35M4F+KdM04s6oyowKDAWBgNVHSUBAf8EDDAKBggrBgEFBQcDHDAOBgNV
HQ8BAf8EBAMCB4AwCgYIKoZIzj0EAwIDaAAwZQIwZk9gTFVIHpYH+N0fucgSjUU2h5sj
wLy78cs9JhVWb18fv9UcDmoJrxt2l5kZI/1+AjEAvKzDQbC6Da9S+Zxuen8AHSPIYgFh
vEvFwEeZNQoMd2FEAUoHUnBXAHX/vgcOmMvlMIICazCCAfKgAwIBAgIEKWsGWTAKBggq
hkjOPQQDAjBtMRIwEAYKCZImiZPyLGQBGRYCY2ExGTAXBgoJkiaJk/IsZAEZFglzYW5k
ZWxtYW4xPDA6BgNVBAMMM2ZvdW50YWluLXRlc3QuZXhhbXBsZS5jb20gVW5zdHJ1bmcg
Rm91bnRhaW4gUm9vdCBDQTAeFw0yMDAyMjUyMTMxNDVaFw0yMjAyMjQyMTMxNDVaMG0x
EjAQBgoJkiaJk/IsZAEZFgJjYTEZMBcGCgmSJomT8ixkARkWCXNhbmRlbG1hbjE8MDoG
A1UEAwwzZm91bnRhaW4tdGVzdC5leGFtcGxlLmNvbSBVbnN0cnVuZyBGb3VudGFpbiBS
b290IENBMHYwEAYHKoZIzj0CAQYFK4EEACIDYgAEG39ZuhfDGrxmjYxs4MP6MXEPZfYi
kb23VoAH29+4eqVFTXmKRpcZWZYxsY9pfTK+PMg+W2O1Rvao1W+b+cDz07kUP3JHUaR2
3dLwv/XKjzABf/jCX9P1IANEi4vyB2y0o2MwYTAPBgNVHRMBAf8EBTADAQH/MA4GA1Ud
DwEB/wQEAwIBBjAdBgNVHQ4EFgQUuaX2yxHhB6RJLKcIxnwQvIezdCYwHwYDVR0jBBgw
FoAUuaX2yxHhB6RJLKcIxnwQvIezdCYwCgYIKoZIzj0EAwIDZwAwZAIwIIMGzo2YpFR6
ZkxKOnDCUjZaUo1ZfSCbKmkUWIc42FV53f0pOJUekZN2tPVmKUS0AjBvOPmvEu0w1YUp
fLEWWL1nkUPEDTD52BysLwbdvNUGQiyEogTqAqRfF1Em+9kv0lwxggFLMIIBRwIBATB1
MG0xEjAQBgoJkiaJk/IsZAEZFgJjYTEZMBcGCgmSJomT8ixkARkWCXNhbmRlbG1hbjE8
MDoGA1UEAwwzZm91bnRhaW4tdGVzdC5leGFtcGxlLmNvbSBVbnN0cnVuZyBGb3VudGFp
biBSb290IENBAgQ/mJtSMAsGCWCGSAFlAwQCAaBpMBgGCSqGSIb3DQEJAzELBgkqhkiG
9w0BBwEwHAYJKoZIhvcNAQkFMQ8XDTIxMDQxMzIxNDMyM1owLwYJKoZIhvcNAQkEMSIE
IEnOrdWjlG70K74IhCJ7UXi+wPS+r2C8DFEqjabGP+G8MAoGCCqGSM49BAMCBEcwRQIh
AMhO3M+tSWb2wKTBOXPArN+XvjSzAhaQA/uLj3qhPwi/AiBDDthf6mjMuirqXE0yjMif
C2UY9oNUFF9Nl0wEQpBBAA==

\<CODE ENDS>
```

The ASN1 decoding of the artifact:

file: examples/parboiled_vr_00_D0-E5-02-00-2D.b64

```
    0:d=0  hl=4 l=3939 cons: SEQUENCE
    4:d=1  hl=2 l=   9 prim: OBJECT            :pkcs7-signedData
   15:d=1  hl=4 l=3924 cons: cont [ 0 ]
   19:d=2  hl=4 l=3920 cons: SEQUENCE
   23:d=3  hl=2 l=   1 prim: INTEGER           :01
   26:d=3  hl=2 l=  13 cons: SET
   28:d=4  hl=2 l=  11 cons: SEQUENCE
   30:d=5  hl=2 l=   9 prim: OBJECT            :sha256
   41:d=3  hl=4 l=2424 cons: SEQUENCE
   45:d=4  hl=2 l=   9 prim: OBJECT            :pkcs7-data
   56:d=4  hl=4 l=2409 cons: cont [ 0 ]
   60:d=5  hl=4 l=2405 prim: OCTET STRING      :{"ietf-voucher-
request:v
 2469:d=3  hl=4 l=1135 cons: cont [ 0 ]
 2473:d=4  hl=4 l= 508 cons: SEQUENCE
 2477:d=5  hl=4 l= 386 cons: SEQUENCE
 2481:d=6  hl=2 l=   3 cons: cont [ 0 ]
 2483:d=7  hl=2 l=   1 prim: INTEGER           :02
 2486:d=6  hl=2 l=   4 prim: INTEGER           :3F989B52
 2492:d=6  hl=2 l=  10 cons: SEQUENCE
 2494:d=7  hl=2 l=   8 prim: OBJECT            :ecdsa-with-SHA256
 2504:d=6  hl=2 l= 109 cons: SEQUENCE
 2506:d=7  hl=2 l=  18 cons: SET
 2508:d=8  hl=2 l=  16 cons: SEQUENCE
 2510:d=9  hl=2 l=  10 prim: OBJECT            :domainComponent
 2522:d=9  hl=2 l=   2 prim: IA5STRING         :ca
 2526:d=7  hl=2 l=  25 cons: SET
 2528:d=8  hl=2 l=  23 cons: SEQUENCE
 2530:d=9  hl=2 l=  10 prim: OBJECT            :domainComponent
 2542:d=9  hl=2 l=   9 prim: IA5STRING         :sandelman
 2553:d=7  hl=2 l=  60 cons: SET
 2555:d=8  hl=2 l=  58 cons: SEQUENCE
 2557:d=9  hl=2 l=   3 prim: OBJECT            :commonName
 2562:d=9  hl=2 l=  51 prim: UTF8STRING        :fountain-
test.example.co
 2615:d=6  hl=2 l=  30 cons: SEQUENCE
 2617:d=7  hl=2 l=  13 prim: UTCTIME           :200225213154Z
 2632:d=7  hl=2 l=  13 prim: UTCTIME           :220224213154Z
 2647:d=6  hl=2 l=  83 cons: SEQUENCE
 2649:d=7  hl=2 l=  18 cons: SET
 2651:d=8  hl=2 l=  16 cons: SEQUENCE
 2653:d=9  hl=2 l=  10 prim: OBJECT            :domainComponent
 2665:d=9  hl=2 l=   2 prim: IA5STRING         :ca
 2669:d=7  hl=2 l=  25 cons: SET
 2671:d=8  hl=2 l=  23 cons: SEQUENCE
 2673:d=9  hl=2 l=  10 prim: OBJECT            :domainComponent
 2685:d=9  hl=2 l=   9 prim: IA5STRING         :sandelman
 2696:d=7  hl=2 l=  34 cons: SET
 2698:d=8  hl=2 l=  32 cons: SEQUENCE
 2700:d=9  hl=2 l=   3 prim: OBJECT            :commonName
 2705:d=9  hl=2 l=  25 prim: UTF8STRING        :fountain-
test.example.co
 2732:d=6  hl=2 l=  89 cons: SEQUENCE
 2734:d=7  hl=2 l=  19 cons: SEQUENCE
 2736:d=8  hl=2 l=   7 prim: OBJECT            :id-ecPublicKey
 2745:d=8  hl=2 l=   8 prim: OBJECT            :prime256v1
 2755:d=7  hl=2 l=  66 prim: BIT STRING
 2823:d=6  hl=2 l=  42 cons: cont [ 3 ]
 2825:d=7  hl=2 l=  40 cons: SEQUENCE
 2827:d=8  hl=2 l=  22 cons: SEQUENCE
 2829:d=9  hl=2 l=   3 prim: OBJECT            :X509v3 Extended Key
Usag
 2834:d=9  hl=2 l=   1 prim: BOOLEAN           :255
 2837:d=9  hl=2 l=  12 prim: OCTET STRING      [HEX
DUMP]:300A06082B0601
 2851:d=8  hl=2 l=  14 cons: SEQUENCE
 2853:d=9  hl=2 l=   3 prim: OBJECT            :X509v3 Key Usage
 2858:d=9  hl=2 l=   1 prim: BOOLEAN           :255
 2861:d=9  hl=2 l=   4 prim: OCTET STRING      [HEX DUMP]:03020780
 2867:d=5  hl=2 l=  10 cons: SEQUENCE
 2869:d=6  hl=2 l=   8 prim: OBJECT            :ecdsa-with-SHA256
 2879:d=5  hl=2 l= 104 prim: BIT STRING
 2985:d=4  hl=4 l= 619 cons: SEQUENCE
 2989:d=5  hl=4 l= 498 cons: SEQUENCE
 2993:d=6  hl=2 l=   3 cons: cont [ 0 ]
 2995:d=7  hl=2 l=   1 prim: INTEGER           :02
 2998:d=6  hl=2 l=   4 prim: INTEGER           :296B0659
 3004:d=6  hl=2 l=  10 cons: SEQUENCE
 3006:d=7  hl=2 l=   8 prim: OBJECT            :ecdsa-with-SHA256
 3016:d=6  hl=2 l= 109 cons: SEQUENCE
 3018:d=7  hl=2 l=  18 cons: SET
 3020:d=8  hl=2 l=  16 cons: SEQUENCE
 3022:d=9  hl=2 l=  10 prim: OBJECT            :domainComponent
 3034:d=9  hl=2 l=   2 prim: IA5STRING         :ca
 3038:d=7  hl=2 l=  25 cons: SET
 3040:d=8  hl=2 l=  23 cons: SEQUENCE
 3042:d=9  hl=2 l=  10 prim: OBJECT            :domainComponent
 3054:d=9  hl=2 l=   9 prim: IA5STRING         :sandelman
 3065:d=7  hl=2 l=  60 cons: SET
 3067:d=8  hl=2 l=  58 cons: SEQUENCE
 3069:d=9  hl=2 l=   3 prim: OBJECT            :commonName
 3074:d=9  hl=2 l=  51 prim: UTF8STRING        :fountain-
test.example.co
 3127:d=6  hl=2 l=  30 cons: SEQUENCE
 3129:d=7  hl=2 l=  13 prim: UTCTIME           :200225213145Z
 3144:d=7  hl=2 l=  13 prim: UTCTIME           :220224213145Z
 3159:d=6  hl=2 l= 109 cons: SEQUENCE
 3161:d=7  hl=2 l=  18 cons: SET
 3163:d=8  hl=2 l=  16 cons: SEQUENCE
 3165:d=9  hl=2 l=  10 prim: OBJECT            :domainComponent
 3177:d=9  hl=2 l=   2 prim: IA5STRING         :ca
 3181:d=7  hl=2 l=  25 cons: SET
 3183:d=8  hl=2 l=  23 cons: SEQUENCE
 3185:d=9  hl=2 l=  10 prim: OBJECT            :domainComponent
 3197:d=9  hl=2 l=   9 prim: IA5STRING         :sandelman
 3208:d=7  hl=2 l=  60 cons: SET
 3210:d=8  hl=2 l=  58 cons: SEQUENCE
 3212:d=9  hl=2 l=   3 prim: OBJECT            :commonName
 3217:d=9  hl=2 l=  51 prim: UTF8STRING        :fountain-
test.example.co
 3270:d=6  hl=2 l= 118 cons: SEQUENCE
 3272:d=7  hl=2 l=  16 cons: SEQUENCE
 3274:d=8  hl=2 l=   7 prim: OBJECT            :id-ecPublicKey
 3283:d=8  hl=2 l=   5 prim: OBJECT            :secp384r1
 3290:d=7  hl=2 l=  98 prim: BIT STRING
 3390:d=6  hl=2 l=  99 cons: cont [ 3 ]
 3392:d=7  hl=2 l=  97 cons: SEQUENCE
 3394:d=8  hl=2 l=  15 cons: SEQUENCE
 3396:d=9  hl=2 l=   3 prim: OBJECT            :X509v3 Basic
Constraints
 3401:d=9  hl=2 l=   1 prim: BOOLEAN           :255
 3404:d=9  hl=2 l=   5 prim: OCTET STRING      [HEX DUMP]:30030101FF
 3411:d=8  hl=2 l=  14 cons: SEQUENCE
 3413:d=9  hl=2 l=   3 prim: OBJECT            :X509v3 Key Usage
 3418:d=9  hl=2 l=   1 prim: BOOLEAN           :255
 3421:d=9  hl=2 l=   4 prim: OCTET STRING      [HEX DUMP]:03020106
 3427:d=8  hl=2 l=  29 cons: SEQUENCE
 3429:d=9  hl=2 l=   3 prim: OBJECT            :X509v3 Subject Key
Ident
 3434:d=9  hl=2 l=  22 prim: OCTET STRING      [HEX
DUMP]:0414B9A5F6CB11
 3458:d=8  hl=2 l=  31 cons: SEQUENCE
 3460:d=9  hl=2 l=   3 prim: OBJECT            :X509v3 Authority Key
Ide
 3465:d=9  hl=2 l=  24 prim: OCTET STRING      [HEX
DUMP]:30168014B9A5F6
 3491:d=5  hl=2 l=  10 cons: SEQUENCE
 3493:d=6  hl=2 l=   8 prim: OBJECT            :ecdsa-with-SHA256
 3503:d=5  hl=2 l= 103 prim: BIT STRING
 3608:d=3  hl=4 l= 331 cons: SET
 3612:d=4  hl=4 l= 327 cons: SEQUENCE
 3616:d=5  hl=2 l=   1 prim: INTEGER           :01
 3619:d=5  hl=2 l= 117 cons: SEQUENCE
 3621:d=6  hl=2 l= 109 cons: SEQUENCE
 3623:d=7  hl=2 l=  18 cons: SET
 3625:d=8  hl=2 l=  16 cons: SEQUENCE
 3627:d=9  hl=2 l=  10 prim: OBJECT            :domainComponent
 3639:d=9  hl=2 l=   2 prim: IA5STRING         :ca
 3643:d=7  hl=2 l=  25 cons: SET
 3645:d=8  hl=2 l=  23 cons: SEQUENCE
 3647:d=9  hl=2 l=  10 prim: OBJECT            :domainComponent
 3659:d=9  hl=2 l=   9 prim: IA5STRING         :sandelman
 3670:d=7  hl=2 l=  60 cons: SET
 3672:d=8  hl=2 l=  58 cons: SEQUENCE
 3674:d=9  hl=2 l=   3 prim: OBJECT            :commonName
 3679:d=9  hl=2 l=  51 prim: UTF8STRING        :fountain-
test.example.co
 3732:d=6  hl=2 l=   4 prim: INTEGER           :3F989B52
 3738:d=5  hl=2 l=  11 cons: SEQUENCE
 3740:d=6  hl=2 l=   9 prim: OBJECT            :sha256
 3751:d=5  hl=2 l= 105 cons: cont [ 0 ]
 3753:d=6  hl=2 l=  24 cons: SEQUENCE
 3755:d=7  hl=2 l=   9 prim: OBJECT            :contentType
 3766:d=7  hl=2 l=  11 cons: SET
 3768:d=8  hl=2 l=   9 prim: OBJECT            :pkcs7-data
 3779:d=6  hl=2 l=  28 cons: SEQUENCE
 3781:d=7  hl=2 l=   9 prim: OBJECT            :signingTime
 3792:d=7  hl=2 l=  15 cons: SET
 3794:d=8  hl=2 l=  13 prim: UTCTIME           :210413214323Z
 3809:d=6  hl=2 l=  47 cons: SEQUENCE
 3811:d=7  hl=2 l=   9 prim: OBJECT            :messageDigest
 3822:d=7  hl=2 l=  34 cons: SET
 3824:d=8  hl=2 l=  32 prim: OCTET STRING      [HEX
DUMP]:49CEADD5A3946E
 3858:d=5  hl=2 l=  10 cons: SEQUENCE
 3860:d=6  hl=2 l=   8 prim: OBJECT            :ecdsa-with-SHA256
 3870:d=5  hl=2 l=  71 prim: OCTET STRING      [HEX
DUMP]:3045022100C84E
```

The JSON contained in the voucher-request. Note that the previous voucher-request is in the prior-signed-voucher-request attribute.

```
{"ietf-voucher-request:voucher":{"assertion":"proximity","cr
eated-on":"2021-04-13T21:43:23.787Z","serial-number":"00-D0-
E5-F2-00-02","nonce":"-_XE9zK9q8Ll1qylMtLKeg","prior-signed-
voucher-request":"MIIGcAYJKoZIhvcNAQcCoIIGYTCCBl0CAQExDTALBg
lghkgBZQMEAgEwggOJBgkqhkiG9w0BBwGgggN6BIIDdnsiaWV0Zi12b3VjaG
VyLXJlcXVlc3Q6dm91Y2hlciI6eyJhc3NlcnRpb24iOiJwcm94aW1pdHkiLC
JjcmVhdGVkLW9uIjoiMjAyMS0wNC0xM1QxNzo0MzoyMy43NDctMDQ6MDAiLC
JzZXJpYWwtbnVtYmVyIjoiMDAtRDAtRTUtRjItMDAtMDIiLCJub25jZSI6Ii
1fWEU5eks5cThMbDFxeWxNdExLZWciLCJwcm94aW1pdHktcmVnaXN0cmFyLW
NlcnQiOiJNSUlCL0RDQ0FZS2dBd0lCQWdJRVA1aWJVakFLQmdncWhrak9QUV
FEQWpCdE1SSXdFQVlLQ1pJbWlaUHlMR1FCR1JZQ1kyRXhHVEFYQmdvSmtpYU
prL0lzWkFFWkZnbHpZVzVrWld4dFlXNHhQREE2QmdOVkJBTU1NMlp2ZFc1MF
lXbHVMWFJsYzNRdVpYaGhiWEJzWlM1amIyMGdWVzV6ZEhKMWJtY2dSbTkxYm
5SaGFXNGdVbTl2ZENCRFFUQWVGdzB5TURBeU1qVXlNVE14TlRSYUZ3MHlNak
F5TWpReU1UTXhOVFJhTUZNeEVqQVFCZ29Ka2lhSmsvSXNaQUVaRmdKallURV
pNQmNHQ2dtU0pvbVQ4aXhrQVJrV0NYTmhibVJsYkcxaGJqRWlNQ0FHQTFVRU
F3d1pabTkxYm5SaGFXNHRkR1Z6ZEM1bGVHRnRjR3hsTG1OdmJUQlpNQk1HQn
lxR1NNNDlBZ0VHQ0NxR1NNNDlBd0VIQTBJQUJKWmxVSEkwdXAvbDNlWmY5dk
NCYitsSW5vRU1FZ2M3Um8rWFpDdGpBSTBDRDFmSmZKUi9oSXl5RG1IV3lZaU
5GYlJDSDlmeWFyZmt6Z1g0cDB6VGl6cWpLakFvTUJZR0ExVWRKUUVCL3dRTU
1Bb0dDQ3NHQVFVRkJ3TWNNQTRHQTFVZER3RUIvd1FFQXdJSGdEQUtCZ2dxaG
tqT1BRUURBZ05vQURCbEFqQm1UMkJNVlVnZWxnZjQzUis1eUJLTlJUYUhteV
BBdkx2eHl6MG1GVlp2WHgrLzFSd09hZ212RzNhWG1Sa2ovWDRDTVFDOHJNTk
JzTG9OcjFMNW5HNTZmd0FkSThoaUFXRzhTOFhBUjVrMUNneDNZVVFCU2dkU2
NGY0FkZisrQnc2WXkrVT0ifX2gggGyMIIBrjCCATWgAwIBAgIEDYOv2TAKBg
gqhkjOPQQDAjAmMSQwIgYDVQQDDBtoaWdod2F5LXRlc3QuZXhhbXBsZS5jb2
0gQ0EwIBcNMjEwNDEzMjAzNzM5WhgPMjk5OTEyMzEwMDAwMDBaMBwxGjAYBg
NVBAUMETAwLUQwLUU1LUYyLTAwLTAyMFkwEwYHKoZIzj0CAQYIKoZIzj0DAQ
cDQgAEA6N1Q4ezfMAKmoecrfb0OBMc1AyEH+BATkF58FsTSyBxs0SbSWLxFj
DOuwB9gLGn2TsTUJumJ6VPw5Z/TP4hJ6NZMFcwHQYDVR0OBBYEFEWIzJaWAG
Q3sLojZWRkVAgGbFatMAkGA1UdEwQCMAAwKwYIKwYBBQUHASAEHxYdaGlnaH
dheS10ZXN0LmV4YW1wbGUuY29tOjk0NDMwCgYIKoZIzj0EAwIDZwAwZAIwTm
lG8sXkKGNbwbKQcYMapFbmSbnHHURFUoFuRqvbgYX7FlXpBczfwF2kllNuuj
igAjAow1kc4r55EmiH+OMEXjBNlWlBSZC5QuJjEf0Jsmxssc+pucjOJ4Shqn
exMEy7bjAxggEEMIIBAAIBATAuMCYxJDAiBgNVBAMMG2hpZ2h3YXktdGVzdC
5leGFtcGxlLmNvbSBDQQIEDYOv2TALBglghkgBZQMEAgGgaTAYBgkqhkiG9w
0BCQMxCwYJKoZIhvcNAQcBMBwGCSqGSIb3DQEJBTEPFw0yMTA0MTMyMTQzMj
NaMC8GCSqGSIb3DQEJBDEiBCBJwhyYibIjeqeR3bOaLURzMlGrc3F2X+kvJ1
errtoCtTAKBggqhkjOPQQDAgRHMEUCIQCmYuCE61HFQXH/E16GDOCsVquDtg
r+Q/6/Du/9QkzA7gIgf7MFhAIPW2PNwRa2vZFQAKXUbimkiHKzXBA8md0VHb
U="}}
```

C.2.3. MASA To Registrar

The MASA will return a voucher to the registrar, which is to be relayed to the pledge.

```
\<CODE BEGINS> file "voucher_00-D0-E5-F2-00-02.b64"

MIIGIgYJKoZIhvcNAQcCoIIGEzCCBg8CAQExDTALBglghkgBZQMEAgEwggN4BgkqhkiG
9w0BBwGgggNpBIIDZXsiaWV0Zi12b3VjaGVyOnZvdWNoZXIiOnsiYXNzZXJ0aW9uIjoi
bG9nZ2VkIiwiY3JlYXRlZC1vbiI6IjIwMjEtMDQtMTNUMTc6NDM6MjQuNTg5LTA0OjAw
Iiwic2VyaWFsLW51bWJlciI6IjAwLUQwLUU1LUYyLTAwLTAyIiwibm9uY2UiOiItX1hF
OXpLOXE4TGwxcXlsTXRMS2VnIiwicGlubmVkLWRvbWFpbi1jZXJ0IjoiTUlJQi9EQ0NB
WUtnQXdJQkFnSUVQNWliVWpBS0JnZ3Foa2pPUFFREFqQnRNUkl3RUFZS0NaSW1pWlB5
TEdRQkdSWUNZMkV4R1RBWEJnb0praWFKay9Jc1pBRVpGZ2x6WVc1a1pXeHRZVzR4UERB
NkJnTlZCQU1NTTJadmRXNTBZV2x1TFhSbGMzUXVaWGhoYlhCc1pTNWpiMjBnVlc1emRI
SjFibWNnUm05MWJuUmhhVzRnVW05dmRDQkRRVEFlRncweU1EQXlNalV5TVRNeE5UUmFG
dzB5TWpBeU1qUXlNVE14TlRSYU1GTXhFakFRQmdvSmtpYUprL0lzWkFFWkZnSmpZVEVa
TUJjR0NnbVNKb21UOGl4a0FSa1dDWE5oYm1SbGJHMWhiakVpTUNBR0ExVUVBd3daWm05
MWJuUmhhVzR0ZEdWemRDNWxlR0Z0Y0d4bExtTnZiVEJaTUJNR0J5cUdTTTQ5QWdFR0ND
cUdTTTQ5QXdFSEEwSUFCSlpsVUhJMHVwL2wzZVpmOXZDQmIrbElub0VNRWdjN1JvK1ha
Q3RqQUkwQ0QxZkpmSlIvaEl5eURtSFd5WWlORmJSQ0g5ZnlhcmZremdYNHAwelRpenFq
S2pBb01CWUdBMVVkSlFFQi93UU1NQW9HQ0NzR0FRVUZCd01jTUE0R0ExVWREd0VCL3dR
RUF3SUhnREFLQmdncWhrak9QUVFEQWdOb0FEQmxBakJtVDJCTVZVZ2VsZ2Y0M1IrNXlC
S05SVGFIbXlQQXZMdnh5ejBtRlZadlh4Ky8xUndPYWdtdkczYVhtUmtqL1g0Q01RQzhy
TU5Cc0xvTnIxTDVuRzU2ZndBZEk4aGlBV0c4UzhYQVI1azFDZ3gzWVVRlNnZFNjRmNB
ZGYrK0J3Nll5K1U9In19oIIBdDCCAXAwgfagAwIBAgIEC4cKMTAKBggqhkjOPQQDAjAm
MSQwIgYDVQQDDBtoaWdod2F5LXRlc3QuZXhhbXBsZS5jb20gQ0EwHhcNMjEwNDEzMjE0
MDE2WhcNMjMwNDEzMjE0MDE2WjAoMSYwJAYDVQQDDB1oaWdod2F5LXRlc3QuZXhhbXBs
ZS5jb20gTUFTQTBZMBMGByqGSM49AgEGCCqGSM49AwEHA0IABKoEFaNEueJE+Mn5Gwcb
pnRznB66bKmzqTCpojJZ96AdRwFtuTCVfoKouLTBX0idIhMLfJLM31lyuKy4CUtpp6Wj
EDAOMAwGA1UdEwEB/wQCMAAwCgYIKoZIzj0EAwIDaQAwZgIxAK7LYS3UXI1uhqoLBh3G
```

```
02C6MnM2JdMjhUmHHM6UI3kankFVJB0VIqFIuwrAqzwTcwIxAIY8Z7OVouLl+a35HZzB
NDJ49c/q1UcDnwC/0FnLUcKYBIEkilETULF1si+dqLT0uTGCAQUwggEBAgEBMC4wJjEk
MCIGA1UEAwwbaGlnaHdheS10ZXN0LmV4YW1wbGUuY29tIENBAgQLhwoxMAsGCWCGSAFl
AwQCAaBpMBgGCSqGSIb3DQEJAzELBgkqhkiG9w0BBwEwHAYJKoZIhvcNAQkFMQ8XDTIx
MDQxMzIxNDMyNFowLwYJKoZIhvcNAQkEMSIEIFUUjg4WYVO+MpX122Qfk/7zm/G6/B59
HD/xrVR0lGIjMAoGCCqGSM49BAMCBEgwRgIhAOhUfxbH2dwpB2BrTDcsYSjRkCCk/WE6
Mdt+y4z5KD9IAiEAphwdIUb40A0noNIUpH7N2lTyAFZgyn1lNHTteY9DmYI=

\<CODE ENDS>
```

The ASN1 decoding of the artifact:

file: examples/voucher_00-D0-E5-F2-00-02.b64

```
    0:d=0  hl=4 l=1570 cons: SEQUENCE
    4:d=1  hl=2 l=   9 prim: OBJECT            :pkcs7-signedData
   15:d=1  hl=4 l=1555 cons: cont [ 0 ]
   19:d=2  hl=4 l=1551 cons: SEQUENCE
   23:d=3  hl=2 l=   1 prim: INTEGER           :01
   26:d=3  hl=2 l=  13 cons: SET
   28:d=4  hl=2 l=  11 cons: SEQUENCE
   30:d=5  hl=2 l=   9 prim: OBJECT            :sha256
   41:d=3  hl=4 l= 888 cons: SEQUENCE
   45:d=4  hl=2 l=   9 prim: OBJECT            :pkcs7-data
   56:d=4  hl=4 l= 873 cons: cont [ 0 ]
   60:d=5  hl=4 l= 869 prim: OCTET STRING      :{"ietf-
voucher:voucher":
  933:d=3  hl=4 l= 372 cons: cont [ 0 ]
  937:d=4  hl=4 l= 368 cons: SEQUENCE
  941:d=5  hl=3 l= 246 cons: SEQUENCE
  944:d=6  hl=2 l=   3 cons: cont [ 0 ]
  946:d=7  hl=2 l=   1 prim: INTEGER           :02
  949:d=6  hl=2 l=   4 prim: INTEGER           :0B870A31
  955:d=6  hl=2 l=  10 cons: SEQUENCE
  957:d=7  hl=2 l=   8 prim: OBJECT            :ecdsa-with-SHA256
  967:d=6  hl=2 l=  38 cons: SEQUENCE
  969:d=7  hl=2 l=  36 cons: SET
  971:d=8  hl=2 l=  34 cons: SEQUENCE
  973:d=9  hl=2 l=   3 prim: OBJECT            :commonName
  978:d=9  hl=2 l=  27 prim: UTF8STRING        :highway-
test.example.com
```

```
 1007:d=6  hl=2 l=  30 cons: SEQUENCE
 1009:d=7  hl=2 l=  13 prim: UTCTIME           :210413214016Z
 1024:d=7  hl=2 l=  13 prim: UTCTIME           :230413214016Z
 1039:d=6  hl=2 l=  40 cons: SEQUENCE
 1041:d=7  hl=2 l=  38 cons: SET
 1043:d=8  hl=2 l=  36 cons: SEQUENCE
 1045:d=9  hl=2 l=   3 prim: OBJECT            :commonName
 1050:d=9  hl=2 l=  29 prim: UTF8STRING        :highway-
test.example.com
 1081:d=6  hl=2 l=  89 cons: SEQUENCE
 1083:d=7  hl=2 l=  19 cons: SEQUENCE
 1085:d=8  hl=2 l=   7 prim: OBJECT            :id-ecPublicKey
 1094:d=8  hl=2 l=   8 prim: OBJECT            :prime256v1
 1104:d=7  hl=2 l=  66 prim: BIT STRING
 1172:d=6  hl=2 l=  16 cons: cont [ 3 ]
 1174:d=7  hl=2 l=  14 cons: SEQUENCE
 1176:d=8  hl=2 l=  12 cons: SEQUENCE
 1178:d=9  hl=2 l=   3 prim: OBJECT            :X509v3 Basic
Constraints
 1183:d=9  hl=2 l=   1 prim: BOOLEAN           :255
 1186:d=9  hl=2 l=   2 prim: OCTET STRING      [HEX DUMP]:3000
 1190:d=5  hl=2 l=  10 cons: SEQUENCE
 1192:d=6  hl=2 l=   8 prim: OBJECT            :ecdsa-with-SHA256
 1202:d=5  hl=2 l= 105 prim: BIT STRING
 1309:d=3  hl=4 l= 261 cons: SET
 1313:d=4  hl=4 l= 257 cons: SEQUENCE
 1317:d=5  hl=2 l=   1 prim: INTEGER           :01
 1320:d=5  hl=2 l=  46 cons: SEQUENCE
 1322:d=6  hl=2 l=  38 cons: SEQUENCE
 1324:d=7  hl=2 l=  36 cons: SET
 1326:d=8  hl=2 l=  34 cons: SEQUENCE
 1328:d=9  hl=2 l=   3 prim: OBJECT            :commonName
 1333:d=9  hl=2 l=  27 prim: UTF8STRING        :highway-
test.example.com
 1362:d=6  hl=2 l=   4 prim: INTEGER           :0B870A31
 1368:d=5  hl=2 l=  11 cons: SEQUENCE
 1370:d=6  hl=2 l=   9 prim: OBJECT            :sha256
 1381:d=5  hl=2 l= 105 cons: cont [ 0 ]
 1383:d=6  hl=2 l=  24 cons: SEQUENCE
 1385:d=7  hl=2 l=   9 prim: OBJECT            :contentType
 1396:d=7  hl=2 l=  11 cons: SET
 1398:d=8  hl=2 l=   9 prim: OBJECT            :pkcs7-data
 1409:d=6  hl=2 l=  28 cons: SEQUENCE
 1411:d=7  hl=2 l=   9 prim: OBJECT            :signingTime
 1422:d=7  hl=2 l=  15 cons: SET
 1424:d=8  hl=2 l=  13 prim: UTCTIME           :210413214324Z
 1439:d=6  hl=2 l=  47 cons: SEQUENCE
 1441:d=7  hl=2 l=   9 prim: OBJECT            :messageDigest
 1452:d=7  hl=2 l=  34 cons: SET
 1454:d=8  hl=2 l=  32 prim: OCTET STRING      [HEX
DUMP]:55148E0E166153
 1488:d=5  hl=2 l=  10 cons: SEQUENCE
 1490:d=6  hl=2 l=   8 prim: OBJECT            :ecdsa-with-SHA256
 1500:d=5  hl=2 l=  72 prim: OCTET STRING      [HEX
DUMP]:3046022100E854
```

Acknowledgements

We would like to thank the various reviewers for their input, in particular William Atwood, Brian Carpenter, Fuyu Eleven, Eliot Lear, Sergey Kasatkin, Anoop Kumar, Tom Petch, Markus Stenberg, Peter van der Stok, and Thomas Werner.

Significant reviews were done by Jari Arkko, Christian Huitema, and Russ Housley.

Henk Birkholz contributed the CDDL for the audit-log response.

This document started its life as a two-page idea from Steinthor Bjarnason.

In addition, significant review comments were provided by many IESG members, including Adam Roach, Alexey Melnikov, Alissa Cooper, Benjamin Kaduk, Éric Vyncke, Roman Danyliw, and Magnus Westerlund.

Authors' Addresses

Max Pritikin

Cisco

Email: pritikin@cisco.com

Michael C. Richardson

Sandelman Software Works

Email: mcr+ietf@sandelman.ca

URI: http://www.sandelman.ca/

Toerless Eckert

Futurewei Technologies Inc. USA

2330 Central Expy

Santa Clara, CA 95050

United States of America

Email: tte+ietf@cs.fau.de

Michael H. Behringer

Email: Michael.H.Behringer@gmail.com

Kent Watsen

Watsen Networks

Email: kent+ietf@watsen.net

BRSKI Over Wifi

```



                          BRSKI over IEEE 802.11
                    draft-friel-brski-over-802dot11-01

Abstract

   This document outlines the challenges associated with implementing
   Bootstrapping Remote Secure Key Infrastructures over IEEE 802.11
and
   IEEE 802.1x networks.  Multiple options are presented for
discovering
   and authenticating to the correct IEEE 802.11 SSID.  This initial
   draft is a discussion document and no final recommendations are
made
   on the recommended approaches to take.

Status of This Memo

   This Internet-Draft is submitted in full conformance with the
   provisions of BCP 78 and BCP 79.

   Internet-Drafts are working documents of the Internet Engineering
   Task Force (IETF).  Note that other groups may also distribute
   working documents as Internet-Drafts.  The list of current
Internet-
   Drafts is at https://datatracker.ietf.org/drafts/current/.

   Internet-Drafts are draft documents valid for a maximum of six
months
   and may be updated, replaced, or obsoleted by other documents at
any
   time.  It is inappropriate to use Internet-Drafts as reference
   material or to cite them other than as "work in progress."

   This Internet-Draft will expire on January 3, 2019.
```

```
Table of Contents
```

```
   1.  Introduction  . . . . . . . . . . . . . . . . . . . . . . . .
3
     1.1.  Terminology . . . . . . . . . . . . . . . . . . . . . . .
4
   2.  Discovery and Authentication Design Considerations  . . . . .
5
     2.1.  Incorrect SSID Discovery  . . . . . . . . . . . . . . . .
5
       2.1.1.  Leveraging BRSKI MASA . . . . . . . . . . . . . . . .
```

```
5
       2.1.2.  Relying on the Network Administrator  . . . . . . . . .
6
       2.1.3.  Requiring the Network to Demonstrate Knowledge of
               Device  . . . . . . . . . . . . . . . . . . . . . . . .
6
     2.2.  IEEE 802.11 Authentication Mechanisms . . . . . . . . . . .
6
       2.2.1.  IP Address Assignment Considerations  . . . . . . . . .
7
     2.3.  Client and Server Implementations . . . . . . . . . . . . .
8
   3.  Potential SSID Discovery Mechanisms . . . . . . . . . . . . . .
8
     3.1.  Well-known BRSKI SSID . . . . . . . . . . . . . . . . . . .
8
     3.2.  IEEE 802.11aq . . . . . . . . . . . . . . . . . . . . . . .
9
     3.3.  IEEE 802.11 Vendor Specific Information Element . . . . . .
10
     3.4.  Reusing Existing IEEE 802.11u Elements  . . . . . . . . . .
10
     3.5.  IEEE 802.11u Interworking Information - Internet  . . . . .
11
     3.6.  Define New IEEE 802.11u Extensions  . . . . . . . . . . . .
12
     3.7.  Wi-Fi Protected Setup . . . . . . . . . . . . . . . . . . .
12
     3.8.  Define and Advertise a BRSKI-specific AKM in RSNE . . . . .
12
     3.9.  Wi-Fi Device Provisioning Profile . . . . . . . . . . . . .
13
   4.  Potential Authentication Options  . . . . . . . . . . . . . . .
13
     4.1.  Unauthenticated Pre-BRSKI and EAP-TLS Post-BRSKI  . . . . .
14
     4.2.  PSK or SAE Pre-BRSKI and EAP-TLS Post-BRSKI . . . . . . . .
15
     4.3.  MAC Address Bypass Pre-BRSKI and EAP-TLS Post-BRSKI . . . .
15
     4.4.  EAP-TLS Pre-BRSKI and EAP-TLS Post-BRSKI  . . . . . . . . .
15
```

```
     4.5.  New TEAP BRSKI mechanism  . . . . . . . . . . . . . . . . .
16
     4.6.  New IEEE 802.11 Authentication Algorithm for BRSKI and
           EAP-TLS Post-BRSKI  . . . . . . . . . . . . . . . . . . . .
18
     4.7.  New IEEE 802.1X EAPOL-Announcements to encapsulate BRSKI
           and EAP-TLS Post-BRSKI  . . . . . . . . . . . . . . . . . .
19
   5.  IANA Considerations . . . . . . . . . . . . . . . . . . . . . .
20
   6.  Security Considerations . . . . . . . . . . . . . . . . . . . .
20
   7.  Informative References  . . . . . . . . . . . . . . . . . . . .
20
   Appendix A.  IEEE 802.11 Primer . . . . . . . . . . . . . . . . . .
21
     A.1.  IEEE 802.11i  . . . . . . . . . . . . . . . . . . . . . . .
21
     A.2.  IEEE 802.11u  . . . . . . . . . . . . . . . . . . . . . . .
22
   Authors' Addresses  . . . . . . . . . . . . . . . . . . . . . . . .
23
```

1. Introduction

   Bootstrapping Remote Secure Key Infrastructures (BRSKI)
   [I-D.ietf-anima-bootstrapping-keyinfra] describes how a device can

```
   bootstrap against a local network using an Initial Device Identity
   X.509 [IEEE802.1AR] IDevID certificate that is pre-installed by the
   vendor on the device in order to obtain an [IEEE802.1AR] LDevID.
The
   BRSKI flow assumes the device can obtain an IP address, and thus
   assumes the device has already connected to the local network.
   Further, the draft states that BRSKI use of IDevIDs:

      allows for alignment with [IEEE802.1X] network access control
      methods, its use here is for Pledge authentication rather than
      network access control.  Integrating this protocol with network
      access control, perhaps as an Extensible Authentication Protocol
      (EAP) method (see [RFC3748], is out-of-scope.

   The draft does not describe any mechanisms for how an [IEEE802.11]
   enabled device would discover and select a suitable [IEEE802.11]
SSID
   when multiple SSIDs are available.  A typical deployment scenario
   could involve a device begin deployed in a location were twenty or
   more SSIDs are being broadcast, for example, in a multi-tenanted
   building or campus where multiple independent organizations operate
   [IEEE802.11] networks.

   In order to reduce the administrative overhead of installing new
   devices, it is desirable that the device will automatically
discover
   and connect to the correct SSID without the installer having to
   manually provision any network information or credentials on the
   device.  It is also desirable that the device does not discover,
   connect to, and automatically enroll with the wrong network as this
   could result in a device that is owned by one organization
connecting
   to the network of a different organization in a multi-tenanted
   building or campus.

   Additionally, as noted above, the BRSKI draft does not describe how
   BRSKI could potentially align with [IEEE802.1X] authentication
   mechanisms.

   This document outlines multiple different potential mechanisms that
   would enable a bootstrapping device to choose between different
   available [IEEE802.11] SSIDs in order to execute the BRSKI flow.
   This document also outlines several options for how [IEEE802.11]
   networks enforcing [IEEE802.1X] authentication could enable the
BRSKI
   flow, and describes the required device behaviour.

   This document presents both [IEEE802.11] mechanisms and Wi-Fi
   Alliance (WFA) mechanisms.  An important consideration when



Friel, et al.            Expires January 3, 2019                [Page
3]
```

```
Internet-Draft                  BRSKI-WIFI                       July
2018


   determining what the most appropriate solution to device onboarding
   should be is what bodies need to be involved in standardisation
   efforts: IETF, IEEE and/or WFA.

1.1.  Terminology

   IEEE 802.11u: an amendment to the IEEE 802.11-2007 standard to add
   features that improve interworking with external networks.

   ANI: Autonomic Networking Infrastructure

   ANQP: Access Network Query Protocol

   AP: IEEE 802.11 Access Point

   CA: Certificate Authority

   EAP: Extensible Authentication Protocol

   EST: Enrollment over Secure Transport

   HotSpot 2.0 / HS2.0: An element of the Wi-Fi Alliance Passpoint
   certificatoin program that enables cell phones to automatically
   discover capabilities and enroll into IEEE 802.11 guest networks
   (hotspots).

   IE: Information Element

   IDevID: Initial Device Identifier

   LDevID: Locally Significant Device Identifier

   OI: Organization Identifier

   MASA: BRSKI Manufacturer Authorized Signing Authority service

   SSID: IEEE 802.11 Service Set Identifier

   STA: IEEE 802.11 station

   WFA: Wi-Fi Alliance

   WLC: Wireless LAN Controller

   WPA/WPA2: Wi-Fi Protected Access / Wi-Fi Protected Access version 2

   WPS: Wi-Fi Protected Setup



Friel, et al.            Expires January 3, 2019                [Page
4]
```

```
Internet-Draft                  BRSKI-WIFI                         July
2018


2.  Discovery and Authentication Design Considerations

2.1.  Incorrect SSID Discovery
```

As will be seen in the following sections, there are several discovery scenarios where the device can choose an incorrect SSID and attempt to join the wrong network. For example, the device is being deployed by one organization in a multi-tenant building, and chooses to connect to the SSID of a neighbor organization. The device is dependent upon the incorrect network rejecting its BRSKI enrollment attempt. It is possible that the device could end up enrolled with the wrong network.

2.1.1. Leveraging BRSKI MASA

2.1.1.1. Prevention

BRSKI allows optional sales channel integration which could be used to ensure only the "correct" network can claim the device. In theory, this could be achieved if the BRSKI MASA service has explicit knowledge of the network where every single device will be deployed. After connecting to the incorrect SSID and possibly authenticating to the network, the device would present network TLS information in its voucher-request, and the MASA server would have to reject the request based on this network TLS information and not issue a voucher. The device could then reject that SSID and attempt to bootstrap against the next available SSID.

This could possibly be acheieved via sales channel integration, where devices are tracked through the supply chain all the way from manufacturer factory to target deployment network operator. In practice, this approach may be challenging to deploy as it may be extremely difficult to implement this tightly coupled sales channel integration and ensure that the MASA actually has accurate deployment network information.

```
   An alternative to sales channel integration is to provide the
device
   owners with a, possibly authenticated, interface or API to the MASA
   service whereby they would have to explicitly claim devices prior
to
   the MASA issuing vouchers for that device.  There are similar
   problems with this approach, as there could be a complex sales and
   channel partner chain between the MASA service operator and the
   device operator who owns and deploys the device.  This could make
   exposure of APIs by the MASA operator to the device operator
   untenable.
```

```
2.1.1.2.  Detection

   If a device connects to the wrong network, the correct network
   operator could detect this after the fact by integration with MASA
   and checking audit logs for the device.  The MASA audit logs should
   indicate all networks that have been issued vouchers for a specific
   device.  This mechanism also relies on the correct network operater
   having a list, bill or materials, or similar of all device
identities
   that should be connecting to their network in order to check MASA
   logs for devices that have not come online, but are known to be
   physically deployed.

2.1.2.  Relying on the Network Administrator
```

```
   An obvious mechanism is to rely on network administrators to be
good
   citizens and explicitly reject devices that attempt to bootstrap
   against the wrong network.  This is not guaranteed to work for two
   main reasons:

   o  Some network administrators will configure an open policy on
their
      network.  Any device that attempts to connect to the network
will
      be automatically granted access.

   o  Some network administrators will be bad actors and will
      intentionally attempt to onboard devices that they do not own
but
      that are in range of their networks.

2.1.3.  Requiring the Network to Demonstrate Knowledge of Device

   Protocols such as the WFA Device Provisioning Profile [DPP] require
   that a network provisoining entity demonstrate knowledge of device
   information such as the device's bootstrapping public key prior to
   the device attempting to connect to the network.  This gives a
higher
   level of confidence to the device that it is connecting to the
   correct SSID.  These mechanisms could leverage a key that is
printed
   on the device label, or included in a sales channel bill of
   materials.  The security of these types of key distribution
   mechanisms relies on keeping the device label or bill of materials
   content from being compromised prior to device installation.

2.2.  IEEE 802.11 Authentication Mechanisms

   [IEEE802.11i] allows an SSID to advertise different authentication
   mechanisms via the AKM Suite list in the RSNE.  A very brief
   introduction to [IEEE802.11i] is given in the appendices.  An SSID
   could advertise PSK or [IEEE802.1X] authentication mechanisms.
When
   a network operator needs to enforce two different authentication
   mechanisms, one for pre-BRSKI devices and one for post-BRSKI
devices,
   the operator has two options:

   o  configure two SSIDs with the same SSID string value, each one
      advertising a different authentication mechanism

   o  configure two different SSIDs, each with its own SSID string
      value, with each one advertising a different authentication
      mechanism

   If devices have to be flexible enough to handle both options, then
   this adds complexity to the device firmware and internal state
   machines.  Similarly, if network infrastructure (APs, WLCs, AAAs)
   potentially needs to support both options, then this adds
complexity
   to network infrastructure configuration flexibility, software and
   state machines.  Consideration must be given to the practicalities
of
   implementation for both devices and network infrastructure when
   designing the final bootstrap mechanism and aligning [IEEE802.11],
   [IEEE802.1X] and BRSKI protocol interactions.

   Devices should be flexible enough to handle potential options
defined
   by any final draft.  When discovering a pre-BRSKI SSID, the device
   should also discover the authentication mechanism enforced by the
   SSID that is advertising BRSKI support.  If the device supports the
   authentication mechanism being advertised, then the device can
   connect to the SSID in order to initiate the BRSKI flow.  For
   example, the device may support [IEEE802.1X] as a pre-BRSKI
   authentication mechanism, but may not support PSK as a pre-BRSKI
   authentication mechanism.

   Once the device has completed the BRKSI flow and has obtained an
   LDevID, a mechanism is needed to tell the device which SSID to use
   for post-BRSKI network access.  This may be a different SSID to the
   pre-BRSKI SSID.  The mechanism by which the post-BRSKI SSID is
   advertised to the device is out-of-scope of this version of this
   document.

2.2.1.  IP Address Assignment Considerations

   If a device has to perform two different authentications, one for
   pre-BRSKI and one for post-BRSKI, network policy will typically
   assign the device to different VLANs for these different stages,
and
   may assign the device different IP addresses depending on which
   network segment the device is assigned to.  This could be true even
   if a single SSID is used for both pre-BRSKI and post-BRSKI
   connections.  Therefore, the bootstrapping device may need to
   completely reset its network connection and network software stack,
```

```
   and obtain a new IP address between pre-BRSKI and post-BRSKI
   connections.

2.3.  Client and Server Implementations
```

```
   When evaluating all possible SSID discovery mechanism and
   authentication mechanisms outlined in this document, consideration
   must be given to the complexity of the required client and server
   implementation and state machines.  Consideration must also be
given
   to the network operator configuration complexity if multiple
   permutations and combinations of SSID discovery and network
   authentication mechanisms are possible.

3.  Potential SSID Discovery Mechanisms

   This section outlines multiple different mechanisms that could
   potentially be leveraged that would enable a bootstrapping device
to
   choose between multiple different available [IEEE802.11] SSIDs.  As
   noted previously, this draft does not make any final
recommendations.

   The discovery options outlined in this document include:

   o  Well-known BRSKI SSID

   o  [IEEE802.11aq]

   o  [IEEE802.11] Vendor Specific Information Element

   o  Reusing Existing [IEEE802.11u] Elements

   o  [IEEE802.11u] Interworking Information - Internet

   o  Define New [IEEE802.11u] Extensions

   o  Wi-Fi Protected Setup

   o  Define and Advertise a BRSKI-specific AKM in RSNE

   o  Wi-Fi Device Provisioning Profile

   These mechanisms are described in more detail in the following
   sections.

3.1.  Well-known BRSKI SSID
```

```
   A standardized naming convention for SSIDs offering BRSKI services
is
   defined such as:




Friel, et al.            Expires January 3, 2019               [Page
8]
```

```
Internet-Draft                   BRSKI-WIFI                       July
2018


   o  BRSKI%ssidname

   Where:

   o  BRSKI: is a well-known prefix string of characters.  This prefix
      string would be baked into device firmware.

   o  %: is a well known delimiter character.  This delimiter
character
      would be baked into device firmware.

   o  ssidname: is the freeform SSID name that the network operator
      defines.

   Device manufacturers would bake the well-known prefix string and
   character delimiter into device firmware.  Network operators
   configuring SSIDs which offer BRSKI services would have to ensure
   that the SSID of those networks begins with this prefix.  On
   bootstrap, the device would scan all available SSIDs and look for
   ones with this given prefix.

   If multiple SSIDs are available with this prefix, then the device
   could simply round robin through these SSIDs and attempt to start
the
```

BRSKI flow on each one in turn until it succeeds.

This mechanism suffers from the limitations outlined in Section 2.1 - it does nothing to prevent a device enrolling against an incorrect network.

Another issue with defining a specific naming convention for the SSID is that this may require network operators to have to deploy a new SSID. In general, network operators attempt to keep the number of unique SSIDs deployed to a minimum as each deployed SSID eats up a percentage of available air time and network capacity. A good discussion of SSID overhead and an SSID overhead [calculator] is available.

3.2. IEEE 802.11aq

[IEEE802.11aq] is currently being worked by the IEEE, but is not yet finalized, and is not yet supported by any vendors in shipping product. [IEEE802.11aq] defines new elements that can be included in [IEEE802.11] Beacon, Probe Request and Probe Response frames, and defines new elements for ANQP frames.

The extensions allow an AP to broadcast support for backend services, where allowed services are those registered in the [IANA] Service Name and Transport Protocol Port Number Registry. The services can be advertised in [IEEE802.11] elements that include either:

```
   o  SHA256 hashes of the registered service names

   o  a bloom filter of the SHA256 hashes of the registered service
      names

   Bloom filters simply serve to reduce the size of Beacon and Probe
   Response frames when a large number of services are advertised.  If
a
   bloom filter is used by the AP, and a device discovers a potential
   service match in the bloom filter, then the device can query the AP
   for the full list of service name hashes using newly defined ANQP
   elements.

   If BRSKI were to leverage [IEEE802.11aq], then the [IEEE802.11aq]
   specification would need to be pushed and supported, and a BRSKI
   service would need to be defined in [IANA].

   This mechanism suffers from the limitations outlined in Section 2.1
-
   it does nothing to prevent a device enrolling against an incorrect
   network.

3.3.  IEEE 802.11 Vendor Specific Information Element

   [IEEE802.11] defines Information Element (IE) number 221 for
carrying
   Vendor Specific information.  The purpose of this document is to
   define an SSID discovery mechanism that can be used across all
   devices and vendors, so use of this IE is not an appropriate long
   term solution.

3.4.  Reusing Existing IEEE 802.11u Elements

   [IEEE802.11u] defines mechanisms for interworking.  An introduction
   to [IEEE802.11u] is given in the appendices.  Existing IEs in
   [IEEE802.11u] include:

   o  Roaming Consortium IE

   o  NAI Realm IE
```

These existing IEs could be used to advertise a well-known, logical service that devices implicitly know to look for.

In the case of NAI Realm, a well-known service name such as "_bootstrapks" could be defined and advertised in the NAI Realm IE. In the case of Roaming Consortium, a well-known Organization Identifier (OI) could be defined and advertised in the Roaming Consortium IE.

Device manufacturers would bake the well-known NAI Realm or Roaming Consortium OI into device firmware. Network operators configuring SSIDs which offer BRSKI services would have to ensure that the SSID offered this NAI Realm or OI. On bootstrap, the device would scan all available SSIDs and use ANQP to query for NAI Realms or Roaming Consortium OI looking for a match.

The key concept with this proposal is that BRSKI uses a well-known NAI Realm name or Roaming Consortium OI more as a logical service advertisement rather than as a backhaul internet provider advertisement. This is conceptually very similar to what [IEEE802.11aq] is attempting to achieve.

Leveraging NAI Realm or Roaming Consortium would not require any [IEEE802.11] specification changes, and could possibly be defined by this IETF draft. Note that the authors are not aware of any currently defined IETF or IANA namespaces that define NAI Realms or

```
   OIs.

   Additionally (or alternatively...) as NAI Realm includes
advertising
   the EAP mechanism required, if a new EAP-BRSKI were to be defined,
   then this could be advertised.  Devices could then scan for an NAI
   Realm that enforced EAP-BRSKI, and ignore the realm name.

   This mechanism suffers from the limitations outlined in Section 2.1
-
   it does nothing to prevent a device enrolling against an incorrect
   network.

   Additionally, as the IEEE is attempting to standardize logical
   service advertisement via [IEEE802.11aq], [IEEE802.11aq] would seem
   to be the more appropriate option than overloading an existing IE.
   However, it is worth noting that configuration of these IEs is
   supported today by WLCs, and this mechanism may be suitable for
   demonstrations or proof-of-concepts.

3.5.  IEEE 802.11u Interworking Information - Internet

   It is possible that an SSID may be configured to provide
unrestricted
   and unauthenticated internet access.  This could be advertised in
the
   Interworking Information IE by including:

   o  internet bit = 1

   o  ASRA bit = 0

   If such a network were discovered, a device could attempt to use
the
   BRSKI well-known vendor cloud Registrar.  Possibly this could be a




Friel, et al.            Expires January 3, 2019               [Page
11]
```

```
   default fall back mechanism that a device could use when
determining
   which SSID to use.

3.6.  Define New IEEE 802.11u Extensions

   Of the various elements currently defined by [IEEE802.11u] for
   potentially advertising BRSKI, NAI Realm and Roaming Consortium IE
   are the two existing options that are a closest fit, as outlined
   above.  Another possibility that has been suggested in the IETF
   mailers is defining an extension to [IEEE802.11u] specifically for
   advertising BRSKI service capability.  Any extensions should be
   included in Beacon and Probe Response frames so that devices can
   discover BRSKI capability without the additional overhead of having
   to explicitly query using ANQP.

   [IEEE802.11aq] appears to be the proposed mechanism for generically
   advertising any service capability, provided that service is
   registered with [IANA].  It is probably a better approach to
   encourage adoption of [IEEE802.11aq] and register a service name
for
   BRSKI with [IANA] rather than attempt to define a completely new
   BRSKI-specific [IEEE802.11u] extension.

3.7.  Wi-Fi Protected Setup

   Wi-Fi Protected Setup (WPS) only works with Wi-Fi Protected Access
   (WPA) and WPA2 when in Personal Mode.  WPS does not work when the
   network is in Enterprise Mode enforcing [IEEE802.1X]
authentication.
   WPS is intended for consumer networks and does not address the
   security requirements of enterprise or IoT deployments.

3.8.  Define and Advertise a BRSKI-specific AKM in RSNE

   [IEEE802.11i] introduced the RSNE element which allows an SSID to
   advertise multiple authentication mechanisms.  A new Authentication
   and Key Management (AKM) Suite could be defined that indicates the
   STA can use BRSKI mechanisms to authenticate against the SSID.  The
   authentication handshake could be an [IEEE802.1X] handshake,
possibly
   leveraging an EAP-BRSKI mechanism, the key thing here is that a new
   AKM is defined and advertised to indicate the specific BRSKI-
capable
   EAP method that is supported by [IEEE802.1X], as opposed to the
   current [IEEE802.1X] AKMs which give no indication of the supported
   EAP mechanisms.  It is clear that such method would limit the SSID
to
   BRSKI-supporting clients.  This would require an additional SSID
   specifically for BRSKI clients.
```

```
3.9.  Wi-Fi Device Provisioning Profile

   The [DPP] specification defines how an entity that is already
trusted
   by a network can assist an untrusted entity in enrolling with the
   network.  The description below assumes the [IEEE802.11] network is
   in infrastructure mode.  DPP introduces multiple key roles
including:

   o  Configurator: A logical entity that is already trusted by the
      network that has capabilities to enroll and provision devices
      called Enrollees.  A Configurator may be a STA or an AP.

   o  Enrollee: A logical entity that is being provisioned by a
      Configurator.  An Enrollee may be a STA or an AP.

   o  Initiator: A logical entity that initiates the DPP
Authentication
      Protocol.  The Initiator may be the Configurator or the
Enrollee.

   o  Responder: A logical entity that responds to the Initiator of
the
      DPP Authentication Protocol.  The Responder may be the
      Configurator or the Enrollee.

   In order to support a plug and play model for installation of
   devices, where the device is simply powered up for the first time
and
   automatically discovers the network without the need for a helper
or
   supervising application, for example an application running on a
   smart cell phone or tablet that performs the role of Configurator,
   then this implies that the AP must perform the role of the
   Configurator and the device or STA performs the role of Enrollee.
   Note that the AP may simply proxy DPP messages through to a backend
   WLC, but from the perspective of the device, the AP is the
   Configurator.

   The DPP specification also mandates that the Initiator must be
   bootstrapped the bootstrapping public key of the Responder.  For
   BRSKI purposes, the DPP bootstrapping public key will be the
   [IEEE802.1AR] IDevID of the device.  As the boostrapping device
   cannot know in advance the bootstrapping public key of a specific
   operators network, this implies that the Configurator must take on
   the role of the Initiator.  Therefore, the AP must take on the
roles
   of both the Configurator and the Initiator.

   More details to be added...

4.  Potential Authentication Options
```

```
   When the bootstrapping device determines which SSID to connect to,
   there are multiple potential options available for how the device
```

```
   authenticates with the network while bootstrapping.  Several
options
   are outlined in this section.  This list is not exhaustive.

   At a high level, authentication can generally be split into two
   phases using two different credentials:

   o  Pre-BRSKI: The device can use its [IEEE802.1AR] IDevID to
connect
      to the network while executing the BRSKI flow

   o  Post-BRSKI: The device can use its [IEEE802.1AR] LDevID to
connect
      to the network after completing BRSKI enrollment

   The authentication options outlined in this document include:

   o  Unauthenticated Pre-BRSKI and EAP-TLS Post-BRSKI

   o  PSK or SAE Pre-BRSKI and EAP-TLS Post-BRSKI

   o  MAC Address Bypass Pre-BRSKI and EAP-TLS Post-BRSKI

   o  EAP-TLS Pre-BRSKI and EAP-TLS Post-BRSKI

   o  New TEAP BRSKI mechanism
   o  New [IEEE802.11] Authentication Algorithm for BRSKI and EAP-TLS
      Post-BRSKI

   o  New [IEEE802.1X] EAPOL-Announcements to encapsulate BRSKI prior
to
      EAP-TLS Post-BRSKI

   These mechanisms are described in more detail in the following
   sections.  Note that any mechanisms leveraging [IEEE802.1X] are
   [IEEE802.11] MAC layer authentication mechanisms and therefore the
   SSID must advertise WPA2 capability.

   When evaluating the multiple authentication options outlined below,
   care and consideration must be given to the complexity of the
   software state machine required in both devices and services for
   implementation.

4.1.  Unauthenticated Pre-BRSKI and EAP-TLS Post-BRSKI

   The device connects to an unauthenticated network pre-BRSKI.  The
   device connects to a network enforcing EAP-TLS post-BRSKI.  The
   device uses its LDevID as the post-BRSKI EAP-TLS credential.

   To be completed..
```

```
4.2.  PSK or SAE Pre-BRSKI and EAP-TLS Post-BRSKI

   The device connects to a network enforcing PSK pre-BRSKI.  The
   mechanism by which the PSK is provisioned on the device for pre-
BRSKI
   authentication is out-of-scope of this version of this document.
The
   device connects to a network enforcing EAP-TLS post-BRSKI.  The
   device uses the LDevID obtained via BRSKI as the post-BRSKI EAP-TLS
   credential.

   When the device connects to the post-BRSKI network that is
enforcing
   EAP-TLS, the device uses its LDevID as its credential.  The device
   should verify the certificate presented by the server during that
   EAP-TLS exchange against the trusted CA list it obtained during
   BRSKI.

   If the [IEEE802.1X] network enforces a tunneled EAP method, for
   example [RFC7170], where the device must present an additional
   credential such as a password, the mechanism by which that
additional
   credential is provisioned on the device for post-BRSKI
authentication
   is out-of-scope of this version of this document.  NAI Realm may be
   used to advertise the EAP methods being enforced by an SSID.  It is
   to be determined if guidelines should be provided on use of NAI
Realm
   for advertising EAP method in order to streamline BRSKI.

4.3.  MAC Address Bypass Pre-BRSKI and EAP-TLS Post-BRSKI

   Many AAA server state machine logic allows for the network to
   fallback to MAC Address Bypass (MAB) when initial authentication
   against the network fails.  If the device does not present a valid
   credential to the network, then the network will check if the
   device's MAC address is whitelisted.  If it is, then the network
may
   grant the device access to a network segment that will allow it to
   complete the BRSKI flow and get provisioned with an LDevID.  Once
the
   device has an LDevID, it can then reauthenticate against the
network
   using its EAP-TLS and its LDevID.
```

4.4. EAP-TLS Pre-BRSKI and EAP-TLS Post-BRSKI

The device connects to a network enforcing EAP-TLS pre-BRSKI. The device uses its IDevID as the pre-BRSKI EAP-TLS credential. The device connects to a network enforcing EAP-TLS post-BRSKI. The device uses its LDevID as the post-BRSKI EAP-TLS credential.

When the device connects to a pre-BRSKI network that is enforcing EAP-TLS, the device uses its IDevID as its credential. The deivce should not attempt to verify the certificate presented by the server during that EAP-TLS exchange, as it has not yet discovered the local domain trusted CA list.

When the device connects to the post-BRSKI network that is enforcing EAP-TLS, the device uses its LDevID as its credential. The device should verify the certificate presented by the server during that EAP-TLS exchange against the trusted CA list it obtained during BRSKI.

Again, if the post-BRSKI network enforces a tunneled EAP method, the mechanism by which that second credential is provisioned on the device is out-of-scope of this version of this document.

4.5. New TEAP BRSKI mechanism

New TEAP TLVs are defined to transport BRSKI messages inside an

```
outer
   EAP TLS tunnel such as TEAP [RFC7170].  [I-D.lear-eap-teap-brski]
   outlines a proposal for how BRSKI messages could be transported
   inside TEAP TLVs.  At a high level, this enables the device to
obtain
   an LDevID during the Layer 2 authentication stage.  This has
multiple
   advantages including:

   o  avoids the need for the device to potentially connect to two
      different SSIDs during bootstrap

   o  the device only needs to handle one authentication mechanism
      during bootstrap

   o  the device only needs to obtain one IP address, which it obtains
      after BRSKI is complete

   o  avoids the need for the device to have to disconnect from the
      network, reset its network stack, and reconnect to the network

   o  potentially simplifies network policy configuration

   There are two suboptions to choose from when tunneling BRSKI
messages
   inside TEAP:

   o  define new TLVs for transporting BRSKI messages inside the TEAP
      tunnel

   o  define a new EAP BRSKI method type that is tunneled within the
      outer TEAP method

   This section assumes that new TLVs are defined for transporting
BRSKI
   messages inside the TEAP tunnel and that a new EAP BRSKI method
type
   is not defined.

   The device discovers and connects to a network enforcing TEAP.  A
   high level TEAP with BRSKI extensions flow would look something
like:
   o  Device starts the EAP flow by sending the EAP TLS ClientHello
      message

   o  EAP server replies and includes CertificateRequest message, and
      may specify certificate_authorities in the message

   o  if the device has an LDevID and the LDevID issuing CA is allowed
      by the certificate_authorities list (i.e. the issuing CA is
      explicitly included in the list, or else the list is empty) then
      the device uses its LDevID to establish the TLS tunnel

   o  if the device does not have an LDevID, or
certificate_authorities
      prevents it using its LDevID, then the device uses its IDevID to
      establish the TLS tunnel

   o  if certificate_authorities prevents the device from using its
      IDevID (and its LDevID if it has one) then the device fails to
      connect

   The EAP server continues with TLS tunnel establishment:

   o  if the device certificate is invalid or expired, then the EAP
      server fails the connection request.

   o  if the device certificate is valid but is not allowed due to a
      configured policy on the EAP server, then the EAP server fails
the
      connection request
   o  if the device certificate is accepted, then the EAP server
      establishes the TLS tunnel and starts the tunneled EAP-BRSKI
      procedures

   At this stage, the EAP server has some policy decisions to make:

   o  if network policy indicates that the device certificate is
      sufficient to grant network access, whether it is an LDevID or
an
      IDevID, then the EAP server simply initiates the Crypto-Binding
      TLV and 'Success' Result TLV exchange.  The device can now
obtain
      an IP address and connect to the network.

   o  the EAP server may instruct the device to initialise a full
BRSKI
      flow.  Typically, the EAP server will instruct the device to
      initialize a BRSKI flow when it presents an IDevID, however, the
      EAP server may instruct the device to initialize a BRSKI flow
even
      if it presented a valid LDevID.  The device sends all BRSKI
      messages, for example 'requestvoucher', inside the TLS tunnel
      using new TEAP TLVs.  Assuming the BRSKI flow completes
      successfully and the device is issued an LDevID, the EAP server
```

```
      completes the exchange by initiating the Crypto-Binding TLV and
      'Success' Result TLV exchange.

   Once the EAP flow has successfully completed, then:
```

```
   o  network policy will automatically assign the device to the
correct
      network segment

   o  the device obtains an IP address

   o  the device can access production service

   It is assumed that the device will automatically handle LDevID
   certificate reenrolment via standard EST [RFC7030] outside the
   context of the EAP tunnel.

   An item to be considered here is what information is included in
   Beacon or Probe Response frames to explicitly indicate that
   [IEEE802.1X] authentication using TEAP supporting BRSKI extensions
is
   allowed.  Currently, the RSNE included in Beacon and Probe Response
   frames can only indicate [IEEE802.1X] support.

4.6.  New IEEE 802.11 Authentication Algorithm for BRSKI and EAP-TLS
      Post-BRSKI

   [IEEE802.11] supports multiple authentication algorithms in its
   Authentication frame including:

   o  Open System

   o  Shared Key

   o  Fast BSS Transition

   o  Simultaneous Authentication of Equals

   Shared Key authentication is used to indicate that the legacy WEP
   authentication mechanism is to be used.  Simultaneous
Authentication
   of Equals is used to indicate that the Dragonfly-based shared
   passphrase authentication mechanism introduced in [IEEE802.11s] is
to
   be used.  One thing that these two methods have in common is that a
   series of handshake data exchanges occur between the device and the
   AP as elements inside Authentication frames, and these
Authentication
   exchanges happen prior to [IEEE802.11] Association.

   It would be possible to define a new Authentication Algorithm and
   define new elements to encapsulate BRSKI messages inside
   Authentication frames.  For example, new elements could be defined
to
```

```
   encapsulate BRSKI requestvoucher, voucher and voucher telemetry
JSON
   messages.  The full BRSKI flow completes and the device gets issued
   an LDevID prior to associating with an SSID, and prior to doing
full
   [IEEE802.1X] authentication using its LDevID.

   The high level flow would be something like:

   o  SSID Beacon / Probe Response indicates in RSNE that it supports
      BRSKI based Authentication Algorithm

   o  SSIDs could also advertise that they support both BRSKI based
      Authentication and [IEEE802.1X]

   o  device discovers SSID via suitable mechanism

   o  device completes BRSKI by sending new elements inside
      Authentication frames and obtains an LDevID

   o  device associates with the AP
```

- device completes [IEEE802.1X] authentication using its LDevID as credential for EAP-TLS or TEAP

4.7. New IEEE 802.1X EAPOL-Announcements to encapsulate BRSKI and EAP-TLS Post-BRSKI

[IEEE802.1X] defines multiple EAPOL packet types, including EAPOL-Announcement and EAPOL-Announcement-Req messages. EAPOL-Annoncement and EAPOL-Announcement-Req messages can include multiple TLVs. EAPOL-Annoncement messages can be sent prior to starting any EAP authentication flow. New TLVs could be defined to encapsulate BRSKI messages inside EAPOL-Announcement and EAPOL-Announcement-Req TLVs. For example, new TLVs could be defined to encapsulate BRSKI requestvoucher, voucher and voucher telemetry JSON messages. The full BRSKI flow could complete inside EAPOL-Announcement exchanges prior to sending EAPOL-Start or EAPOL-EAP messages.

The high level flow would be something like:

- SSID Beacon / Probe Response indicates somehow in RSNE that it supports [IEEE802.1X] including BRSKI extensions.

- device connects to SSID and completes standard Open System Authentication and Association

- device starts [IEEE802.1X] EAPOL flow and uses new EAPOL-Announcement frames to encapsulate and complete BRSKI flow to obtain an LDevID

```
   o  device completes [IEEE802.1X] authentication using its LDevID as
      credential for EAP-TLS or TEAP

5.  IANA Considerations

   [[ TODO ]]

6.  Security Considerations

   [[ TODO ]]

7.  Informative References

   [calculator]
              Revolution Wi-Fi, "SSID Overhead Calculator", n.d.,
              <http://www.revolutionwifi.net/revolutionwifi/p/
              ssid-overhead-calculator.html>.

   [DPP]      Wi-Fi Alliance, "Wi-Fi Device Provisioning Protocol",
              n.d., <https://www.wi-fi.org/file/wi-fi-device-
              provisioning-protocol-dpp-draft-technical-specification-
              v0023>.

   [I-D.ietf-anima-bootstrapping-keyinfra]
              Pritikin, M., Richardson, M., Behringer, M., Bjarnason,
              S., and K. Watsen, "Bootstrapping Remote Secure Key
              Infrastructures (BRSKI)", draft-ietf-anima-
bootstrapping-
              keyinfra-16 (work in progress), June 2018.

   [I-D.lear-eap-teap-brski]
              Lear, E., Friel, O., and N. Cam-Winget, "Bootstrapping
Key
              Infrastructure over EAP", draft-lear-eap-teap-brski-00
              (work in progress), June 2018.

   [IANA]     Internet Assigned Numbers Authority, "Service Name and
              Transport Protocol Port Number Registry", n.d.,
              <https://www.iana.org/assignments/service-names-port-
              numbers/service-names-port-numbers.xhtml>.

   [IEEE802.11]
              IEEE, ., "Wireless LAN Medium Access Control (MAC) and
              Physical Layer (PHY) Specifications", 2016.

   [IEEE802.11aq]
              IEEE, ., "802.11 Amendment 5 Pre-Association Discovery",
              2017.
```

```
   [IEEE802.11i]
              IEEE, ., "802.11 Amendment 6 Medium Access Control (MAC)
              Security Enhancements", 2004.

   [IEEE802.11s]
              IEEE, ., "802.11 Amendment 10 Mesh Networking", 2011.

   [IEEE802.11u]
              IEEE, ., "802.11 Amendment 9 Interworking with External
              Networks", 2011.

   [IEEE802.1AR]
              IEEE, ., "Secure Device Identity", 2017.

   [IEEE802.1X]
              IEEE, ., "Port-Based Network Access Control", 2010.
```

[RFC3748] Aboba, B., Blunk, L., Vollbrecht, J., Carlson, J., and H. Levkowetz, Ed., "Extensible Authentication Protocol (EAP)", RFC 3748, DOI 10.17487/RFC3748, June 2004, <https://www.rfc-editor.org/info/rfc3748>.

[RFC4282] Aboba, B., Beadles, M., Arkko, J., and P. Eronen, "The Network Access Identifier", RFC 4282, DOI 10.17487/RFC4282, December 2005, <https://www.rfc-editor.org/info/rfc4282>.

[RFC7030] Pritikin, M., Ed., Yee, P., Ed., and D. Harkins, Ed., "Enrollment over Secure Transport", RFC 7030, DOI 10.17487/RFC7030, October 2013, <https://www.rfc-editor.org/info/rfc7030>.

[RFC7170] Zhou, H., Cam-Winget, N., Salowey, J., and S. Hanna, "Tunnel Extensible Authentication Protocol (TEAP) Version 1", RFC 7170, DOI 10.17487/RFC7170, May 2014, <https://www.rfc-editor.org/info/rfc7170>.

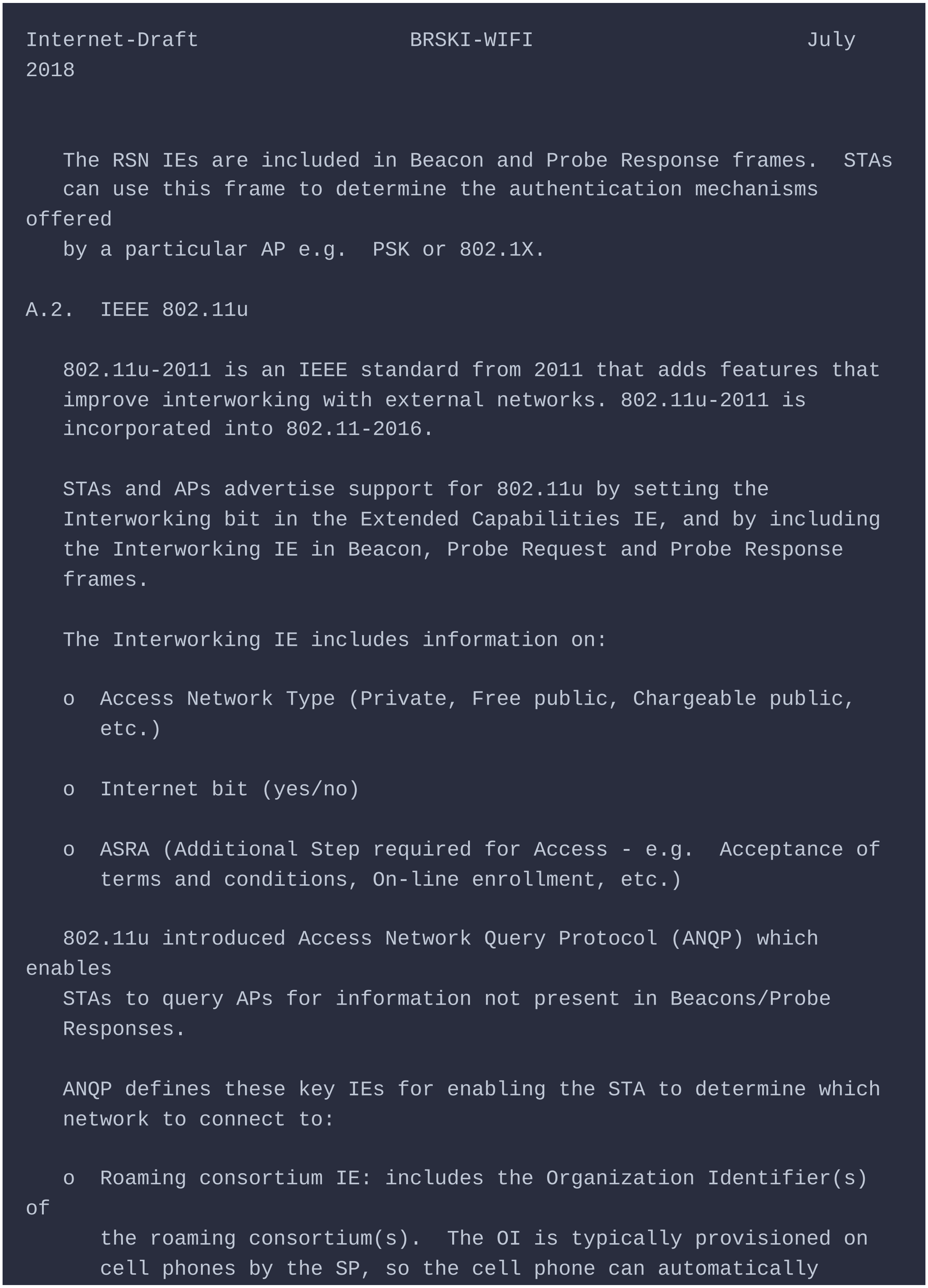
Appendix A. IEEE 802.11 Primer

A.1. IEEE 802.11i

802.11i-2004 is an IEEE standard from 2004 that improves connection security. 802.11i-2004 is incorporated into 802.11-2014. 802.11i defines the Robust Security Network IE which includes information on:

o Pairwise Cipher Suites (WEP-40, WEP-104, CCMP-128, etc.)

o Authentication and Key Management Suites (PSK, 802.1X, etc.)



The RSN IEs are included in Beacon and Probe Response frames. STAs can use this frame to determine the authentication mechanisms offered by a particular AP e.g. PSK or 802.1X.

A.2. IEEE 802.11u

802.11u-2011 is an IEEE standard from 2011 that adds features that improve interworking with external networks. 802.11u-2011 is incorporated into 802.11-2016.

STAs and APs advertise support for 802.11u by setting the Interworking bit in the Extended Capabilities IE, and by including the Interworking IE in Beacon, Probe Request and Probe Response frames.

The Interworking IE includes information on:

- o Access Network Type (Private, Free public, Chargeable public, etc.)

- o Internet bit (yes/no)

- o ASRA (Additional Step required for Access - e.g. Acceptance of terms and conditions, On-line enrollment, etc.)

802.11u introduced Access Network Query Protocol (ANQP) which enables STAs to query APs for information not present in Beacons/Probe Responses.

ANQP defines these key IEs for enabling the STA to determine which network to connect to:

- o Roaming consortium IE: includes the Organization Identifier(s) of the roaming consortium(s). The OI is typically provisioned on cell phones by the SP, so the cell phone can automatically detect 802.11 networks that provide access to its SP's consortium.

o 3GPP Cellular Network IE: includes the Mobile Country Code (MCC) and Mobile Network Code (MNC) of the SP the AP provides access to.

o Network Access Identifier Realm IE: includes [RFC4282] realm names that the AP provides access to (e.g. wifi.service-provider.com). The NAI Realm IE also includes info on the EAP type required to access that realm e.g. EAP-TLS.

o Domain name IE: the domain name(s) of the local AP operator. Its purpose is to enable a STA to connect to a domain operator that may have a roaming agreement with STA's Service Provider.

STAs can use one or more of the above IEs to make a suitable decision on which SSID to pick.

HotSpot 2.0 is an example of a specification built on top of 802.11u and defines 10 additional ANQP elements using the standard vendor extensions mechanisms defined in 802.11. It also defines a HS2.0 Indication element that is included in Beacons and Probe Responses so that STAs can immediately tell if an SSID supports HS2.0.

Authors' Addresses

Owen Friel
Cisco

Email: ofriel@cisco.com

Eliot Lear
Cisco

Email: lear@cisco.com

Max Pritikin
Cisco

Email: pritikin@cisco.com

Michael Richardson
Sandelman Software Works

Email: mcr+ietf@sandelman.ca

Document type	Replaced Internet-Draft (individual) Expired & archivedThis document is an Internet-Draft (I-D). Anyone may submit an I-D to the IETF. This I-D is not endorsed by the IETF and has no formal standing in the IETF standards process.
Select version	0001
Compare versions	draft-friel-brski-over-802dot11-01 draft-friel-brski-over-802dot11-00 draft-friel-brski-over-802dot11-00 draft-friel-brski-over-802dot11-01 draft-friel-brski-over-802dot11-00 draft-friel-brski-over-802dot11-01Side-by-side Inline
Authors	Owen Friel , Eliot Lear , Max Pritikin , Michael Richardson Email authors
Replaced by	draft-friel-anima-brski-over-802dot11
RFC stream	(None)
Intended RFC status	(None)
Other formats	txt xml pdf bibtex bibxml

BRSKI-AE: Alternative Enrollment Protocols In BRSKI

Abstract

This document enhances Bootstrapping Remote Secure Key Infrastructure (BRSKI, RFC 8995) to allow employing alternative enrollment protocols, such as CMP.

Using self-contained signed objects, the origin of enrollment requests and responses can be authenticated independently of message transfer. This supports end-to-end security and asynchronous operation of certificate enrollment and provides flexibility where to authenticate and authorize certification requests.

Status Of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at https://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on 8 October 2022.

Copyright Notice

Table Of Contents

- 1. Introduction
  - 1.1. Motivation
  - 1.2. Supported Environment
  - 1.3. List of Application Examples
- 2. Terminology
- 3. Requirements and Mapping to Solutions
  - 3.1. Basic Requirements
  - 3.2. Solution Options for Proof-of-possession
  - 3.3. Solution Options for Proof-of-identity
- 4. Adaptations to BRSKI
  - 4.1. Architecture
  - 4.2. Message Exchange
  - 4.3. Enhancements to Addressing Scheme
  - 4.4. Domain Registrar Support of Alternative Enrollment Protocols
- 5. Instantiation to Existing Enrollment Protocols
  - 5.1. BRSKI-EST-fullCMC: Instantiation to EST (informative)
  - 5.2. BRSKI-CMP: Instantiation to CMP (normative if CMP is chosen)
- 6. IANA Considerations
- 7. Security Considerations
- 8. Acknowledgments
- 9. References
  - 9.1. Normative References
  - 9.2. Informative References
- Appendix A. Using EST for Certificate Enrollment
- Appendix B. Application Examples
  - B.1. Rolling Stock
  - B.2. Building Automation
  - B.3. Substation Automation
  - B.4. Electric Vehicle Charging Infrastructure
  - B.5. Infrastructure Isolation Policy
  - B.6. Sites with Insufficient Level of Operational Security
- Appendix C. History of Changes TBD RFC Editor: please delete
- Authors' Addresses

1. Introduction

1.1. Motivation

BRSKI, as defined in [RFC8995], specifies a solution for secure automated zero-touch bootstrapping of new devices, so-called pledges. This includes the discovery of the registrar in the target domain, time synchronization, and the exchange of security information necessary to establish mutual trust between pledges and the target domain.

A pledge gains trust in the target domain via the domain registrar as follows. It obtains security information about the domain, specifically a domain certificate to be trusted, by requesting a voucher object defined in [RFC8366]. Such a voucher is a self-contained signed object originating from a Manufacturer Authorized Signing Authority (MASA). Therefore, the voucher may be provided in online mode (synchronously) or offline mode (asynchronously). The pledge can authenticate the voucher because it is shipped with a trust anchor of its manufacturer such that it can validate signatures (including related certificates) by the MASA.

Trust by the target domain in a pledge is established by providing the pledge with a domain-specific LDevID certificate. The certification request of the pledge is signed using its IDevID secret and can be validated by the target domain using the trust anchor of the pledge manufacturer, which needs to pre-installed in the domain.

For enrolling devices with LDevID certificates, BRSKI typically utilizes Enrollment over Secure Transport (EST) [RFC7030]. EST has its specific characteristics, detailed in Appendix A. In particular, it requires online or on-site availability of the RA for performing the data origin authentication and final authorization decision on the certification request. This type of enrollment can be called 'synchronous enrollment'. For various reasons, it may be preferable to use alternative enrollment protocols such as the Certificate Management Protocol (CMP) [RFC4210] profiled in [I-D.ietf-lamps-lightweight-cmp-profile] or Certificate Management over CMS (CMC) [RFC5272]. that are more flexible and independent of the transfer mechanism because they represent certification request messages as authenticated self-contained objects.

Depending on the application scenario, the required RA/CA components may not be part of the registrar. They even may not be available on-site but rather be provided by remote backend systems. The registrar or its deployment site may not have an online connection with them or the connectivity may be intermittent. This may be due to security requirements for operating the backend systems or due to site deployments where on-site or always-online operation may be not feasible or too costly. In such scenarios, the authentication and authorization of certification requests will not or can not be performed on-site at enrollment time. In this document, enrollment that is not performed in a (time-wise) consistent way is called 'asynchronous enrollment'. Asynchronous enrollment requires a store-and-forward transfer of certification requests along with the information needed for authenticating the requester. This allows offline processing the request.

Application scenarios may also involve network segmentation, which is utilized in industrial systems to separate domains with different security needs. Such scenarios lead to similar requirements if the TLS connection carrying the requester authentication is terminated and thus request messages need to be forwarded on further channels before the registrar/RA can authorize the certification request. In order to preserve the requester authentication, authentication information needs to be retained and ideally bound directly to the certification request.

There are basically two approaches for forwarding certification requests along with requester authentication information:

- A trusted component (e.g., a local RA) in the target domain is needed that forwards the certification request combined with the validated identity of the requester (e,g., its IDevID certificate) and an indication of successful verification of the proof-of-possession (of the corresponding private key) in a way preventing changes to the combined information. When connectivity is available, the trusted component forwards the certification request together with the requester information (authentication and proof-of-possession) for further processing. This approach offers only hop-by-hop security. The backend PKI must rely on the local pledge authentication result provided by the local RA when performing the authorization of the certification request. In BRSKI, the EST server is such a trusted component, being co-located with the registrar in the target domain.
- Involved components use authenticated self-contained objects for the enrollment, directly binding the certification request and the requester authentication in a cryptographic way. This approach supports end-to-end security, without the need to trust in intermediate domain components. Manipulation of the request and the requester identity information can be detected during the validation of the self-contained signed object.

Focus of this document is the support of alternative enrollment protocols that allow using authenticated self-contained objects for device credential bootstrapping. This enhancement of BRSKI is named BRSKI-AE, where AE stands for alternative enrollment protocols and for asynchronous enrollment. This specification carries over the main characteristics of BRSKI, namely that the pledge obtains trust anchor information for authenticating the domain registrar and other target domain components as well as a domain-specific X.509 device certificate (the LDevID certificate) along with the corresponding private key (the LDevID secret) and certificate chain.

The goals are to enhance BRSKI to

- support alternative enrollment protocols,
- support end-to-end security for enrollment, and
- make it applicable to scenarios involving asynchronous enrollment.

This is achieved by

- extending the well-known URI approach with an additional path element indicating the enrollment protocol being used, and
- defining a certificate waiting indication and handling, for the case that the certifying component is (temporarily) not available.

This specification can be applied to both synchronous and asynchronous enrollment.

In contrast to BRSKI, this specification supports offering multiple enrollment protocols on the infrastructure side, which enables pledges and their developers to pick the preferred one.

1.2. Supported Environment

BRSKI-AE is intended to be used in domains that may have limited support of on-site PKI services and comprises application scenarios like the following.

- There are requirements or implementation restrictions that do not allow using EST for enrolling an LDevID certificate.
- Pledges and/or the target domain already have an established certificate management approach different from EST that shall be reused (e.g., in brownfield installations).
- There is no registration authority available on site in the target domain. Connectivity to an off-site RA is intermittent or entirely offline. A store-and-forward mechanism is used for communicating with the off-site services.
- Authoritative actions of a local RA are limited and may not be sufficient for authorizing certification requests by pledges. Final authorization is done by an RA residing in the operator domain.

1.3. List Of Application Examples

Bootstrapping can be handled in various ways, depending on the application domains. The informative Appendix B provides illustrative examples from various industrial control system environments and operational setups. They motivate the support of alternative enrollment protocols, based on the following examples of operational environments:

- Rolling stock
- Building automation
- Electrical substation automation
- Electric vehicle charging infrastructures
- Infrastructure isolation policy
- Sites with insufficient level of operational security

2. Terminology

The key words "**MUST**", "**MUST NOT**", "**REQUIRED**", "**SHALL**", "**SHALL NOT**", "**SHOULD**", "**SHOULD NOT**", "**RECOMMENDED**", "**NOT RECOMMENDED**", "**MAY**", and "**OPTIONAL**" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

This document relies on the terminology defined in [RFC8995] and [IEEE.802.1AR_2009]. The following terms are defined in addition:

- EE:

  End entity, in the BRSKI context called pledge. It is the entity that is bootstrapped to the target domain. It holds a public-private key pair, for which it requests a public-key certificate. An identifier for the EE is given as the subject name of the certificate.

- RA:

  Registration authority, an optional system component to which a CA delegates certificate management functions such as authenticating requesters and performing authorization checks on certification requests.

- CA:

  Certification authority, issues certificates and provides certificate status information.

- target domain:

  The set of entities that share a common local trust anchor, independent of where the entities are deployed.

- site:

  Describes the locality where an entity, e.g., pledge, registrar, RA, CA, is deployed. Different sites can belong to the same target domain.

- on-site:

  Describes a component or service or functionality available in the target deployment site.

- off-site:

  Describes a component or service or functionality available in an operator site different from the target deployment site. This may be a central site or a cloud service, to which only a temporary connection is available.

- asynchronous communication:

  Describes a time-wise interrupted communication between a pledge (EE) and a registrar or PKI component.

- synchronous communication:

  Describes a time-wise uninterrupted communication between a pledge (EE) and a registrar or PKI component.

- authenticated self-contained object:

  Describes in this context an object that is cryptographically bound to the IDevID certificate of a pledge. The binding is assumed to be provided through a digital signature of the actual object using the IDevID secret.

3. Requirements And Mapping To Solutions

3.1. Basic Requirements

There were two main drivers for the definition of BRSKI-AE:

- The solution architecture may already use or require a certificate management protocol other than EST. Therefore, this other protocol should be usable for requesting LDevID certificates.
- The domain registrar may not be the (final) point that authenticates and authorizes certification requests and the pledge may not have a direct connection to it. Therefore, certification requests should be self-contained signed objects.

Based on the intended target environment described in Section 1.2 and the application examples described in Appendix B, the following requirements are derived to support authenticated self-contained objects as containers carrying certification requests.

At least the following properties are required:

- proof-of-possession: demonstrates access to the private key corresponding to the public key contained in a certification request. This is typically achieved by a self-signature using the corresponding private key.
- proof-of-identity: provides data origin authentication of the certification request. This typically is achieved by a signature using the IDevID secret of the pledge.

The rest of this section gives an incomplete list of solution examples, based on existing technology described in IETF documents:

3.2. Solution Options For Proof-Of-Possession

Certification request objects: Certification requests are data structures protecting only the integrity of the contained data and providing proof-of-possession for a (locally generated) private key. Examples for certification request data structures are:

- PKCS#10 [RFC2986]. This certification request structure is self-signed to protect its integrity and prove possession of the private key that corresponds to the public key included in the request.
- CRMF [RFC4211]. Also this certificate request message format supports integrity protection and proof-of-possession, typically by a self-signature generated over (part of) the structure with the private key corresponding to the included public key. CRMF also supports further proof-of-possession methods for types of keys that do not support any signature algorithm.

The integrity protection of certification request fields includes the public key because it is part of the data signed by the corresponding private key. Yet note that for the above examples this is not sufficient to provide data origin authentication, i.e., proof-of-identity. This extra property can be achieved by an additional binding to the IDevID of the pledge. This binding to source authentication supports the authorization decision for the certification request. The binding of data origin authentication to the certification request may be delegated to the protocol used for certificate management.

3.3. Solution Options For Proof-Of-Identity

The certification request should be bound to an existing authenticated credential (here, the IDevID certificate) to enable a proof of identity and, based on it, an authorization of the certification request. The binding may be achieved through security options in an underlying transport protocol such as TLS if the authorization of the certification request is (completely) done at the next communication hop. This binding can also be done in a transport-independent way by wrapping the certification request with signature employing an existing IDevID. the BRSKI context, this will be the IDevID. This requirement is addressed by existing enrollment protocols in various ways, such as:

- EST [RFC7030] utilizes PKCS#10 to encode the certification request. The Certificate Signing Request (CSR) optionally provides a binding to the underlying TLS session by including the tls-unique value in the self-signed PKCS#10 structure. The tls-unique value results from the TLS handshake. Since the TLS handshake includes client authentication and the pledge utilizes its IDevID for it, the proof-of-identity is provided by such a binding to the TLS session. This can be supported using the EST /simpleenroll endpoint. Note that the binding of the TLS handshake to the CSR is optional in EST. As an alternative to binding to the underlying TLS authentication in the transport layer, [RFC7030] sketches wrapping the CSR with a Full PKI Request message using an existing certificate.
- SCEP [RFC8894] supports using a shared secret (passphrase) or an existing certificate to protect CSRs based on SCEP Secure Message Objects using CMS wrapping ([RFC5652]). Note that the wrapping using an existing IDevID in SCEP is referred to as renewal. Thus SCEP does not rely on the security of the underlying transfer.
- CMP [RFC4210] supports using a shared secret (passphrase) or an existing certificate, which may be an IDevID credential, to authenticate certification requests via the PKIProtection structure in a PKIMessage. The certification request is typically encoded utilizing CRMF, while PKCS#10 is supported as an alternative. Thus CMP does not rely on the security of the underlying transfer protocol.
- CMC [RFC5272] also supports utilizing a shared secret (passphrase) or an existing certificate to protect certification requests, which can be either in CRMF or PKCS#10 structure. The proof-of-identity can be provided as part of a FullCMCRequest, based on CMS [RFC5652] and signed with an existing IDevID secret. Thus CMC does not rely on the security of the underlying transfer protocol.

4. Adaptations To BRSKI

In order to support alternative enrollment protocols, asynchronous enrollment, and more general system architectures, BRSKI-AE lifts some restrictions of BRSKI [RFC8995]. This way, authenticated self-contained objects such as those described in Section 3 above can be used for certificate enrollment.

The enhancements needed are kept to a minimum in order to ensure reuse of already defined architecture elements and interactions. In general, the communication follows the BRSKI model and utilizes the existing BRSKI architecture elements. In particular, the pledge initiates communication with the domain registrar and interacts with the MASA as usual.

4.1. Architecture

The key element of BRSKI-AE is that the authorization of a certification request **MUST** be performed based on an authenticated self-contained object. The certification request is bound in a self-contained way to a proof-of-origin based on the IDevID. Consequently, the authentication and authorization of the certification request **MAY** be done by the domain registrar and/or by other domain components. These components may be offline or reside in some central backend of the domain operator (off-site) as described in Section 1.2. The registrar and other on-site domain components may have no or only temporary (intermittent) connectivity to them. The certification request **MAY** also be piggybacked on another protocol.

This leads to generalizations in the placement and enhancements of the logical elements as shown in Figure 1.

```
                                           +------------------------+
   +--------------Drop-Ship--------------->| Vendor Service         |
   |                                       +------------------------+
   |                                       | M anufacturer|         |
   |                                       | A uthorized  |Ownership|
   |                                       | S igning     |Tracker  |
   |                                       | A uthority   |         |
   |                                       +--------------+---------+
   |                                                      ^
   |                                                      |
   V                                                      |
+--------+     .........................................  |
|        |     .                                       .  | BRSKI-
|        |     .  +------------+       +------------+  .  | MASA
| Pledge |     .  |   Join     |       | Domain     \<-----+
|        |     .  |   Proxy    |       | Registrar/ |  .
|        \<-------->............\<-------> Enrollment |  .
|        |     .  |       BRSKI-AE    | Proxy/LRA  |  .
| IDevID |     .  |            |       +------^-----+  .
|        |     .  +------------+              |        .
|        |     .                              |        .
+--------+     ...............................|.........
                on-site "domain" components   |
                                              | e.g., RFC 4210,
                                              |       RFC 7030, ...
 .............................................|.....................
 . +---------------------------+     +--------v------------------+ .
 . | Public-Key Infrastructure \<-----+ Registration Authority    | .
 . | PKI CA                    +-----> PKI RA                    | .
 . +---------------------------+     +---------------------------+ .
 ...................................................................
         off-site or central "domain" components
```

Figure 1: Architecture Overview Using Off-site PKI Components

The architecture overview in Figure 1 has the same logical elements as BRSKI, but with more flexible placement of the authentication and authorization checks on certification requests. Depending on the application scenario, the registrar **MAY** still do all of these checks (as is the case in BRSKI), or part of them, or none of them.

The following list describes the on-site components in the target domain of the pledge shown in Figure 1.

- Join Proxy: same functionality as described in BRSKI [RFC8995].

- Domain Registrar / Enrollment Proxy / LRA: in BRSKI-AE, the domain registrar has mostly the same functionality as in BRSKI, namely to facilitate the communication of the pledge with the MASA and the PKI. Yet in contrast to BRSKI, the registrar offers different enrollment protocols and **MAY** act as a local registration authority (LRA) or simply as an enrollment proxy. In such cases, the domain registrar forwards the certification request to some off-site RA component, which performs at least part of the authorization. This also covers the case that the registrar has only intermittent connection and forwards the certification request to the RA upon re-established connectivity.

  Note: To support alternative enrollment protocols, the URI scheme for addressing the domain registrar is generalized (see Section 4.3).

The following list describes the components provided by the vendor or manufacturer outside the target domain.

- MASA: general functionality as described in BRSKI [RFC8995]. The voucher exchange with the MASA via the domain registrar is performed as described in BRSKI.

  Note: The interaction with the MASA may be synchronous (voucher request with nonce) or asynchronous (voucher request without nonce).

- Ownership tracker: as defined in BRSKI.

The following list describes the target domain components that can optionally be operated in the off-site backend of the target domain.

- PKI RA: Performs certificate management functions for the domain as a centralized public-key infrastructure for the domain operator. As far as not already done by the domain registrar, it performs the final validation and authorization of certification requests.
- PKI CA: Performs certificate generation by signing the certificate structure requested in already authenticated and authorized certification requests.

Based on the diagram in Section 2.1 of BRSKI [RFC8995] and the architectural changes, the original protocol flow is divided into three phases showing commonalities and differences to the original approach as follows.

- Discovery phase: same as in BRSKI steps (1) and (2)
- Voucher exchange phase: same as in BRSKI steps (3) and (4).
- Enrollment phase: step (5) is changed to employing an alternative enrollment protocol that uses authenticated self-contained objects.

4.2. Message Exchange

The behavior of a pledge described in Section 2.1 of BRSKI [RFC8995] is kept with one exception. After finishing the Imprint step (4), the Enroll step (5) **MUST** be performed with an enrollment protocol utilizing authenticated self-contained objects. Section 5 discusses selected suitable enrollment protocols and options applicable.

```
[
 Cannot render SVG graphics - please view
 https://raw.githubusercontent.com/anima-wg/anima-brski-ae/main/o.png
]
```

Figure 2: BRSKI-AE Abstract Protocol Overview

Pledge - registrar discovery and voucher exchange

The discovery phase and voucher exchange are applied as specified in [RFC8995].

Registrar - MASA voucher exchange

This voucher exchange is performed as specified in [RFC8995].

Pledge - registrar - RA/CA certificate enrollment

As stated in Section 3, the enrollment **MUST** be performed using an authenticated self-contained object providing not only proof-of-possession but also proof-of-identity (source authentication).

```
+--------+                        +------------+       +------------+
| Pledge |                        | Domain     |       | Operator   |
|        |                        | Registrar  |       | RA/CA      |
|        |                        |  (JRC)     |       | (PKI)      |
+--------+                        +------------+       +------------+
 /-->                                      |                       |
[Optional request of CA certificates]      |                       |
 |---------- CA Certs Request ------------>|                       |
 |                 [if connection to operator domain is available] |
 |                                         |-- CA Certs Request -->|
 |                                         |\<- CA Certs Response --|
 |\<--------- CA Certs Response ------------|                       |
 /-->                                      |                       |
[Optional request of attributes to include in Certificate Request] |
 |---------- Attribute Request ----------->|                       |
 |                 [if connection to operator domain is available] |
 |                                         |- Attribute Request -->|
 |                                         |\<- Attribute Response -|
 |\<--------- Attribute Response -----------|                       |
 /-->                                      |                       |
[Mandatory certificate request]            |                       |
 |---------- Certificate Request --------->|                       |
 |                 [if connection to operator domain is available] |
 |                                         |-Certificate Request ->|
 |                                         |\<- Certificate Resp. --|
 |\<--------- Certificate Response ---------|                       |
 /-->                                      |                       |
[Optional certificate confirmation]        |                       |
 |---------- Certificate Confirm --------->|                       |
 |                 [if connection to operator domain is available] |
 |                                        |-Certificate Confirm ->|
 |                                        |\<---- PKI Confirm -----|
 |\<--------- PKI/Registrar Confirm --------|                        |
```

Figure 3: Certificate Enrollment

The following list provides an abstract description of the flow depicted in Figure 3.

- CA Certs Request: The pledge optionally requests the latest relevant CA certificates. This ensures that the pledge has the complete set of current CA certificates beyond the pinned-domain-cert (which is contained in the voucher and may be just the domain registrar certificate).
- CA Certs Response: It **MUST** contain the current root CA certificate, which typically is the LDevID trust anchor, and any additional certificates that the pledge may need to validate certificates.
- Attribute Request: Typically, the automated bootstrapping occurs without local administrative configuration of the pledge. Nevertheless, there are cases in which the pledge may also include additional attributes specific to the target domain into the certification request. To get these attributes in advance, the attribute request can be used.
- Attribute Response: It **MUST** contain the attributes to be included in the subsequent certification request.
- Certificate Request: This certification request **MUST** contain the authenticated self-contained object ensuring both proof-of-possession of the corresponding private key and proof-of-identity of the requester.
- Certificate Response: The certification response message **MUST** contain on success the requested certificate and **MAY** include further information, like certificates of intermediate CAs.
- Certificate Confirm: An optional confirmation sent after the requested certificate has been received and validated. It contains a positive or negative confirmation by the pledge whether the certificate was successfully enrolled and fits its needs.
- PKI/Registrar Confirm: An acknowledgment by the PKI or registrar that **MUST** be sent on reception of the Cert Confirm.

The generic messages described above may be implemented using various enrollment protocols supporting authenticated self-contained objects, as described in Section 3. Examples are available in Section 5.

Pledge - registrar - enrollment status telemetry

The enrollment status telemetry is performed as specified in [RFC8995]. In BRSKI this is described as part of the enrollment phase, but due to the generalization on the enrollment protocol described in this document it fits better as a separate step here.

4.3. Enhancements To Addressing Scheme

BRSKI-AE provides generalizations to the addressing scheme defined in BRSKI [RFC8995] to accommodate alternative enrollment protocols that use authenticated self-contained objects for certification requests. As this is supported by various existing enrollment protocols, they can be directly employed (see also Section 5).

The addressing scheme in BRSKI for certification requests and the related CA certificates and CSR attributes retrieval functions uses the definition from EST [RFC7030]; here on the example of simple enrollment: "/.well-known/est/simpleenroll". This approach is generalized to the following notation: "/.well-known/<enrollment-protocol>/<request>" in which <enrollment-protocol> refers to a certificate enrollment protocol. Note that enrollment is considered here a message sequence that contains at least a certification request and a certification response. The following conventions are used in order to provide maximal compatibility to BRSKI:

- <enrollment-protocol>: **MUST** reference the protocol being used, which **MAY** be CMP, CMC, SCEP, EST [RFC7030] as in BRSKI, or a newly defined approach.

Note: additional endpoints (well-known URIs) at the registrar may need to be defined by the enrollment protocol being used.

- <request>: if present, the <request> path component **MUST** describe, depending on the enrollment protocol being used, the operation requested. Enrollment protocols are expected to define their request endpoints, as done by existing protocols (see also Section 5).

4.4. Domain Registrar Support Of Alternative Enrollment Protocols

Well-known URIs for various endpoints on the domain registrar are already defined as part of the base BRSKI specification or indirectly by EST. In addition, alternative enrollment endpoints **MAY** be supported at the registrar. The pledge will recognize whether its preferred enrollment option is supported by the domain registrar by sending a request to its preferred enrollment endpoint and evaluating the HTTP response status code.

The following list of endpoints provides an illustrative example for a domain registrar supporting several options for EST as well as for CMP to be used in BRSKI-AE. The listing contains the supported endpoints to which the pledge may connect for bootstrapping. This includes the voucher handling as well as the enrollment endpoints. The CMP related enrollment endpoints are defined as well-known URIs in CMP Updates [I-D.ietf-lamps-cmp-updates] and the Lightweight CMP profile [I-D.ietf-lamps-lightweight-cmp-profile].

```
\</brski/voucherrequest>,ct=voucher-cms+json
\</brski/voucher_status>,ct=json
\</brski/enrollstatus>,ct=json
\</est/cacerts>;ct=pkcs7-mime
\</est/fullcmc>;ct=pkcs7-mime
\</est/csrattrs>;ct=pkcs7-mime
\</cmp/initialization>;ct=pkixcmp
\</cmp/p10>;ct=pkixcmp
\</cmp/getcacerts>;ct=pkixcmp
\</cmp/getcertreqtemplate>;ct=pkixcmp
```

5. Instantiation To Existing Enrollment Protocols

This section maps the requirements to support proof-of-possession and proof-of-identity to selected existing enrollment protocols handles provides further aspects of instantiating them in BRSKI-AE.

5.1. BRSKI-EST-FullCMC: Instantiation To EST (Informative)

When using EST [RFC7030], the following aspects and constraints need to be considered and the given extra requirements need to be fulfilled, which adapt Section 5.9.3 of BRSKI [RFC8995]:

- proof-of-possession is provided typically by using the specified PKCS#10 structure in the request. Together with Full PKI requests, also CRMF can be used.

- proof-of-identity needs to be achieved by signing the certification request object using the Full PKI Request option (including the /fullcmc endpoint). This provides sufficient information for the RA to authenticate the pledge as the origin of the request and to make an authorization decision on the received certification request. Note: EST references CMC [RFC5272] for the definition of the Full PKI Request. For proof-of-identity, the signature of the SignedData of the Full PKI Request is performed using the IDevID secret of the pledge.

  Note: In this case the binding to the underlying TLS connection is not necessary.

- When the RA is temporarily not available, as per Section 4.2.3 of [RFC7030], an HTTP status code 202 should be returned by the registrar, and the pledge will repeat the initial Full PKI Request

5.2. BRSKI-CMP: Instantiation To CMP (Normative If CMP Is Chosen)

Note: Instead of referring to CMP as specified in [RFC4210] and [I-D.ietf-lamps-cmp-updates], this document refers to the Lightweight CMP Profile [I-D.ietf-lamps-lightweight-cmp-profile] because the subset of CMP defined there is sufficient for the functionality needed here.

When using CMP, the following specific implementation requirements apply (cf. Figure 3).

- CA Certs Request
  - Requesting CA certificates over CMP is **OPTIONAL**. If supported, it **SHALL** be implemented as specified in Section 4.3.1 of [I-D.ietf-lamps-lightweight-cmp-profile].
- Attribute Request
  - Requesting certificate request attributes over CMP is **OPTIONAL**. If supported, it **SHALL** be implemented as specified in Section 4.3.3 of [I-D.ietf-lamps-lightweight-cmp-profile]. Note that alternatively the registrar **MAY** modify the contents of requested certificate contents as specified in Section 5.2.3.2 of [I-D.ietf-lamps-lightweight-cmp-profile].
- Certificate Request
  - Proof-of-possession **SHALL** be provided as defined in Section 4.1.1 (based on CRMF) or Section 4.1.4 (based on PKCS#10) of the Lightweight CMP Profile [I-D.ietf-lamps-lightweight-cmp-profile]. The `caPubs` field of certificate response messages **SHOULD NOT** be used.
  - Proof-of-identity **SHALL** be provided by using signature-based protection of the certification request message as outlined in Section 3.2. of [I-D.ietf-lamps-lightweight-cmp-profile] using the IDevID secret.
- Certificate Confirm
  - Explicit confirmation of new certificates to the RA **MAY** be used as specified in Section 4.1.1 of the Lightweight CMP Profile [I-D.ietf-lamps-lightweight-cmp-profile]. Note that independently of certificate confirmation within CMP, enrollment status telemetry with the registrar will be performed as described in Section 5.9.4 of BRSKI [RFC8995].
- If delayed delivery of responses (for instance, to support asynchronous enrollment) within CMP is needed, it **SHALL** be performed as specified in Sections 4.4 and 5.1.2 of [I-D.ietf-lamps-lightweight-cmp-profile].

6. IANA Considerations

This document does not require IANA actions.

7. Security Considerations

The security considerations as laid out in BRSKI [RFC8995] apply for the discovery and voucher exchange as well as for the status exchange information.

The security considerations as laid out in the Lightweight CMP Profile [I-D.ietf-lamps-lightweight-cmp-profile] apply as far as CMP is used.

8. Acknowledgments

We would like to thank Brian E. Carpenter, Michael Richardson, and Giorgio Romanenghi for their input and discussion on use cases and call flows.

9. References

9.1. Normative References

- [I-D.ietf-lamps-cmp-updates]

  Brockhaus, H., Oheimb, D. V., and J. Gray, "Certificate Management Protocol (CMP) Updates", Work in Progress, Internet-Draft, draft-ietf-lamps-cmp-updates-17, 12 January 2022, <https://www.ietf.org/archive/id/draft-ietf-lamps-cmp-updates-17.txt>.

- [I-D.ietf-lamps-lightweight-cmp-profile]

  Brockhaus, H., Oheimb, D. V., and S. Fries, "Lightweight Certificate Management Protocol (CMP) Profile", Work in Progress, Internet-Draft, draft-ietf-lamps-lightweight-cmp-profile-10, 1 February 2022, <https://www.ietf.org/archive/id/draft-ietf-lamps-lightweight-cmp-profile-10.txt>.

- [IEEE.802.1AR_2009]

  IEEE, "IEEE Standard for Local and metropolitan area networks - Secure Device Identity", IEEE 802.1AR-2009, DOI 10.1109/ieeestd.2009.5367679, 28 December 2009, <http://ieeexplore.ieee.org/servlet/opac?punumber=5367676>.

- [RFC2119]

  Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

- [RFC4210]

  Adams, C., Farrell, S., Kause, T., and T. Mononen, "Internet X.509 Public Key Infrastructure Certificate Management Protocol (CMP)", RFC 4210, DOI 10.17487/RFC4210, September 2005, <https://www.rfc-editor.org/info/rfc4210>.

- [RFC8174]

  Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

- [RFC8366]

  Watsen, K., Richardson, M., Pritikin, M., and T. Eckert, "A Voucher Artifact for Bootstrapping Protocols", RFC 8366, DOI 10.17487/RFC8366, May 2018, <https://www.rfc-editor.org/info/rfc8366>.

- [RFC8995]

  Pritikin, M., Richardson, M., Eckert, T., Behringer, M., and K. Watsen, "Bootstrapping Remote Secure Key Infrastructure (BRSKI)", RFC 8995, DOI 10.17487/RFC8995, May 2021, <https://www.rfc-editor.org/info/rfc8995>.

9.2. Informative References

- [IEC-62351-9]

  International Electrotechnical Commission, "IEC 62351 - Power systems management and associated information exchange - Data and communications security - Part 9: Cyber security key management for power system equipment", IEC 62351-9, May 2017.

- [ISO-IEC-15118-2]

  International Standardization Organization / International Electrotechnical Commission, "ISO/IEC 15118-2 Road vehicles - Vehicle-to-Grid Communication Interface - Part 2: Network and application protocol requirements", ISO/IEC 15118-2, April 2014.

- [NERC-CIP-005-5]

  North American Reliability Council, "Cyber Security - Electronic Security Perimeter", CIP 005-5, December 2013.

- [OCPP]

  Open Charge Alliance, "Open Charge Point Protocol 2.0.1 (Draft)", December 2019.

- [RFC2986]

  Nystrom, M. and B. Kaliski, "PKCS #10: Certification Request Syntax Specification Version 1.7", RFC 2986, DOI 10.17487/RFC2986, November 2000, <https://www.rfc-editor.org/info/rfc2986>.

- [RFC4211]

  Schaad, J., "Internet X.509 Public Key Infrastructure Certificate Request Message Format (CRMF)", RFC 4211, DOI 10.17487/RFC4211, September 2005, <https://www.rfc-editor.org/info/rfc4211>.

- [RFC5272]

Schaad, J. and M. Myers, "Certificate Management over CMS (CMC)", RFC 5272, DOI 10.17487/RFC5272, June 2008, <https://www.rfc-editor.org/info/rfc5272>.

- [RFC5652]

  Housley, R., "Cryptographic Message Syntax (CMS)", STD 70, RFC 5652, DOI 10.17487/RFC5652, September 2009, <https://www.rfc-editor.org/info/rfc5652>.

- [RFC5929]

  Altman, J., Williams, N., and L. Zhu, "Channel Bindings for TLS", RFC 5929, DOI 10.17487/RFC5929, July 2010, <https://www.rfc-editor.org/info/rfc5929>.

- [RFC7030]

  Pritikin, M., Ed., Yee, P., Ed., and D. Harkins, Ed., "Enrollment over Secure Transport", RFC 7030, DOI 10.17487/RFC7030, October 2013, <https://www.rfc-editor.org/info/rfc7030>.

- [RFC8894]

  Gutmann, P., "Simple Certificate Enrolment Protocol", RFC 8894, DOI 10.17487/RFC8894, September 2020, <https://www.rfc-editor.org/info/rfc8894>.

- [UNISIG-Subset-137]

  UNISIG, "Subset-137; ERTMS/ETCS On-line Key Management FFFIS; V1.0.0", December 2015, <https://www.era.europa.eu/sites/default/files/filesystem/ertms/ccs_tsi_annex_a_-_mandatory_specifications/set_of_specifications_3_etcs_b3_r2_gsm-r_b1/index083_-_subset-137_v100.pdf>. http://www.kmc-subset137.eu/index.php/download/

Appendix A. Using EST For Certificate Enrollment

When using EST with BRSKI, pledges interact via TLS with the domain registrar, which acts both as EST server and as registration authority (RA). The TLS connection is mutually authenticated, where the pledge uses its IDevID certificate issued by its manufacturer.

In order to provide a strong proof-of-origin of the certification request, EST has the option to include in the certification request the so-called tls-unique value [RFC5929] of the underlying TLS channel. This binding of the proof-of-identity of the TLS client, which is supposed to be the certificate requester, to the proof-of-possession for the private key is conceptually non-trivial and requires specific support by TLS implementations.

The registrar terminates the security association with the pledge at TLS level and thus the binding between the certification request and the authentication of the pledge. The EST server uses the authenticated pledge identity provided by the IDevID for checking the authorization of the pledge for the given certification request before issuing to the pledge a domain-specific certificate (LDevID certificate). This approach typically requires online or on-site availability of the RA for performing the final authorization decision for the certification request.

Using EST for BRSKI has the advantage that the mutually authenticated TLS connection established between the pledge and the registrar can be reused for protecting the message exchange needed for enrolling the LDevID certificate. This strongly simplifies the implementation of the enrollment message exchange.

Yet the use of TLS has the limitation that this cannot provide auditability nor end-to-end security for the certificate enrollment request because the TLS session is transient and terminates at the registrar. This is a problem in particular if the enrollment is done via multiple hops, part of which may not even be network-based.

A further limitation of using EST as the certificate enrollment protocol is that due to using PKCS#10 structures in enrollment requests, the only possible proof-of-possession method is a self-signature, which excludes requesting certificates for key types that do not support signing.

Appendix B. Application Examples

This informative annex provides some detail to the application examples listed in Section 1.3.

B.1. Rolling Stock

Rolling stock or railroad cars contain a variety of sensors, actuators, and controllers, which communicate within the railroad car but also exchange information between railroad cars building a train, with track-side equipment, and/or possibly with backend systems. These devices are typically unaware of backend system connectivity. Managing certificates may be done during maintenance cycles of the railroad car, but can already be prepared during operation. Preparation will include generating certification requests, which are collected and later forwarded for processing, once the railroad car is connected to the operator backend. The authorization of the certification request is then done based on the operator's asset/inventory information in the backend.

UNISIG has included a CMP profile for enrollment of TLS certificates of on-board and track-side components in the Subset-137 specifying the ETRAM/ETCS on-line key management for train control systems [UNISIG-Subset-137].

B.2. Building Automation

In building automation scenarios, a detached building or the basement of a building may be equipped with sensors, actuators, and controllers that are connected with each other in a local network but with only limited or no connectivity to a central building management system. This problem may occur during installation time but also during operation. In such a situation a service technician collects the necessary data and transfers it between the local network and the central building management system, e.g., using a laptop or a mobile phone. This data may comprise parameters and settings required in the operational phase of the sensors/actuators, like a component certificate issued by the operator to authenticate against other components and services.

The collected data may be provided by a domain registrar already existing in the local network. In this case connectivity to the backend PKI may be facilitated by the service technician's laptop. Alternatively, the data can also be collected from the pledges directly and provided to a domain registrar deployed in a different network as preparation for the operational phase. In this case, connectivity to the domain registrar may also be facilitated by the service technician's laptop.

B.3. Substation Automation

In electrical substation automation scenarios, a control center typically hosts PKI services to issue certificates for Intelligent Electronic Devices (IEDs) operated in a substation. Communication between the substation and control center is performed through a proxy/gateway/DMZ, which terminates protocol flows. Note that [NERC-CIP-005-5] requires inspection of protocols at the boundary of a security perimeter (the substation in this case). In addition, security management in substation automation assumes central support of several enrollment protocols in order to support the various capabilities of IEDs from different vendors. The IEC standard IEC62351-9 [IEC-62351-9] specifies mandatory support of two enrollment protocols: SCEP [RFC8894] and EST [RFC7030] for the infrastructure side, while the IED must only support one of the two.

B.4. Electric Vehicle Charging Infrastructure

For electric vehicle charging infrastructure, protocols have been defined for the interaction between the electric vehicle and the charging point (e.g., ISO 15118-2 [ISO-IEC-15118-2]) as well as between the charging point and the charging point operator (e.g. OCPP [OCPP]). Depending on the authentication model, unilateral or mutual authentication is required. In both cases the charging point uses an X.509 certificate to authenticate itself in TLS connections between the electric vehicle and the charging point. The management of this certificate depends, among others, on the selected backend connectivity protocol. In the case of OCPP, this protocol is meant to be the only communication protocol between the charging point and the backend, carrying all information to control the charging operations and maintain the charging point itself. This means that the certificate management needs to be handled in-band of OCPP. This requires the ability to encapsulate the certificate management messages in a transport-independent way. Authenticated self-containment will support this by allowing the transport without a separate enrollment protocol, binding the messages to the identity of the communicating endpoints.

B.5. Infrastructure Isolation Policy

This refers to any case in which network infrastructure is normally isolated from the Internet as a matter of policy, most likely for security reasons. In such a case, limited access to external PKI services will be allowed in carefully controlled short periods of time, for example when a batch of new devices is deployed, and forbidden or prevented at other times.

B.6. Sites With Insufficient Level Of Operational Security

The registration authority performing (at least part of) the authorization of a certification request is a critical PKI component and therefore requires higher operational security than components utilizing the issued certificates for their security features. CAs may also demand higher security in the registration procedures. Especially the CA/Browser forum currently increases the security requirements in the certificate issuance procedures for publicly trusted certificates. In case the on-site components of the target domain cannot be operated securely enough for the needs of a registration authority, this service should be transferred to an off-site backend component that has a sufficient level of security.

Appendix C. History Of Changes TBD RFC Editor: Please Delete

From IETF draft 06 -> IETF draft 06:

- Renamed the repo and files from anima-brski-async-enroll to anima-brski-ae
- Added graphics for abstract protocol overview as suggested by Toerless Eckert
- Balanced (sub-)sections and their headers
- Added details on CMP instance, now called BRSKI-CMP

From IETF draft 04 -> IETF draft 05:

- David von Oheimb became the editor.
- Streamline wording, consolidate terminology, improve grammar, etc.
- Shift the emphasis towards supporting alternative enrollment protocols.
- Update the title accordingly - preliminary change to be approved.
- Move comments on EST and detailed application examples to informative annex.
- Move the remaining text of section 3 as two new sub-sections of section 1.

From IETF draft 03 -> IETF draft 04:

- Moved UC2 related parts defining the pledge in responder mode to a separate document. This required changes and adaptations in several sections. Main changes concerned the removal of the subsection for UC2 as well as the removal of the YANG model related text as it is not applicable in UC1.
- Updated references to the Lightweight CMP Profile.
- Added David von Oheimb as co-author.

From IETF draft 02 -> IETF draft 03:

- Housekeeping, deleted open issue regarding YANG voucher-request in UC2 as voucher-request was enhanced with additional leaf.
- Included open issues in YANG model in UC2 regarding assertion value agent-proximity and CSR encapsulation using SZTP sub module).

From IETF draft 01 -> IETF draft 02:

- Defined call flow and objects for interactions in UC2. Object format based on draft for JOSE signed voucher artifacts and aligned the remaining objects with this approach in UC2 .
- Terminology change: issue #2 pledge-agent -> registrar-agent to better underline agent relation.
- Terminology change: issue #3 PULL/PUSH -> pledge-initiator-mode and pledge-responder-mode to better address the pledge operation.
- Communication approach between pledge and registrar-agent changed by removing TLS-PSK (former section TLS establishment) and associated references to other drafts in favor of relying on higher layer exchange of signed data objects. These data objects are included also in the pledge-voucher-request and lead to an extension of the YANG module for the voucher-request (issue #12).
- Details on trust relationship between registrar-agent and registrar (issue #4, #5, #9) included in UC2.
- Recommendation regarding short-lived certificates for registrar-agent authentication towards registrar (issue #7) in the security considerations.
- Introduction of reference to agent signing certificate using SKID in agent signed data (issue #11).
- Enhanced objects in exchanges between pledge and registrar-agent to allow the registrar to verify agent-proximity to the pledge (issue #1) in UC2.
- Details on trust relationship between registrar-agent and pledge (issue #5) included in UC2.
- Split of use case 2 call flow into sub sections in UC2.

From IETF draft 00 -> IETF draft 01:

- Update of scope in Section 1.2 to include in which the pledge acts as a server. This is one main motivation for use case 2.
- Rework of use case 2 to consider the transport between the pledge and the pledge-agent. Addressed is the TLS channel establishment between the pledge-agent and the pledge as well as the endpoint definition on the pledge.
- First description of exchanged object types (needs more work)

- Clarification in discovery options for enrollment endpoints at the domain registrar based on well-known endpoints in Section 4.4 do not result in additional /.well-known URIs. Update of the illustrative example. Note that the change to /brski for the voucher related endpoints has been taken over in the BRSKI main document.
- Updated references.
- Included Thomas Werner as additional author for the document.

From individual version 03 -> IETF draft 00:

- Inclusion of discovery options of enrollment endpoints at the domain registrar based on well-known endpoints in Section 4.4 as replacement of section 5.1.3 in the individual draft. This is intended to support both use cases in the document. An illustrative example is provided.
- Missing details provided for the description and call flow in pledge-agent use case UC2, e.g. to accommodate distribution of CA certificates.
- Updated CMP example in Section 5 to use Lightweight CMP instead of CMP, as the draft already provides the necessary /.well-known endpoints.
- Requirements discussion moved to separate section in Section 3. Shortened description of proof of identity binding and mapping to existing protocols.
- Removal of copied call flows for voucher exchange and registrar discovery flow from [RFC8995] in Section 4 to avoid doubling or text or inconsistencies.
- Reworked abstract and introduction to be more crisp regarding the targeted solution. Several structural changes in the document to have a better distinction between requirements, use case description, and solution description as separate sections. History moved to appendix.

From individual version 02 -> 03:

- Update of terminology from self-contained to authenticated self-contained object to be consistent in the wording and to underline the protection of the object with an existing credential. Note that the naming of this object may be discussed. An alternative name may be attestation object.
- Simplification of the architecture approach for the initial use case having an offsite PKI.
- Introduction of a new use case utilizing authenticated self-contain objects to onboard a pledge using a commissioning tool containing a pledge-agent. This requires additional changes in the BRSKI call flow sequence and led to changes in the introduction, the application example,and also in the related BRSKI-AE call flow.
- Update of provided examples of the addressing approach used in BRSKI to allow for support of multiple enrollment protocols in Section 4.3.

From individual version 01 -> 02:

- Update of introduction text to clearly relate to the usage of IDevID and LDevID.
- Definition of the addressing approach used in BRSKI to allow for support of multiple enrollment protocols in Section 4.3. This section also contains a first discussion of an optional discovery mechanism to address situations in which the registrar supports more than one enrollment approach. Discovery should avoid that the pledge performs a trial and error of enrollment protocols.
- Update of description of architecture elements and changes to BRSKI in Section 4.1.
- Enhanced consideration of existing enrollment protocols in the context of mapping the requirements to existing solutions in Section 3 and in Section 5.

From individual version 00 -> 01:

- Update of examples, specifically for building automation as well as two new application use cases in Appendix B.
- Deletion of asynchronous interaction with MASA to not complicate the use case. Note that the voucher exchange can already be handled in an asynchronous manner and is therefore not considered further. This resulted in removal of the alternative

path the MASA in Figure 1 and the associated description in Section 4.1.

- Enhancement of description of architecture elements and changes to BRSKI in Section 4.1.
- Consideration of existing enrollment protocols in the context of mapping the requirements to existing solutions in Section 3.
- New section starting Section 5 with the mapping to existing enrollment protocols by collecting boundary conditions.

Authors' Addresses

David von Oheimb (editor)

Siemens AG

Otto-Hahn-Ring 6

81739 Munich

Germany

Email: david.von.oheimb@siemens.com

URI: https://www.siemens.com/

Steffen Fries

Siemens AG

Otto-Hahn-Ring 6

81739 Munich

Germany

Email: steffen.fries@siemens.com

URI: https://www.siemens.com/

Hendrik Brockhaus

Siemens AG

Otto-Hahn-Ring 6

81739 Munich

Germany

Email: hendrik.brockhaus@siemens.com

URI: https://www.siemens.com/

Eliot Lear

Cisco Systems

Richtistrasse 7

CH-8304 Wallisellen

Switzerland

Phone: [+41 44 878 9200](tel:+41 44 878 9200)

Email: lear@cisco.com

Network Integration Architecture

This section will be finalised as part of D2.04

"D2.04 - Integration methods - outlines practical method of integration with various bearer technologies"

These items are placeholders for work in progress

The substance of this section will reconcile the target network and interface/API where the CAHN policy framework is integrated. This allows us to define the universal authorisation and authentication methods across multiple networks.

Wifi Integration Method

The WIFI integration method is addresses through the BRSKI over IEEE 802.11 specification, futher refinements of which have been produced as part of the NIST delviery.

Lora Integration Method

The LORA integration method will be defined by identifying the precise LoRaWAN API interface points with the CAHN policy framework

Satellite Integration Method

The satellited integration method will be defined by identifying the precise Starlink API interface points with the CAHN policy framework

20 5G Integration Method

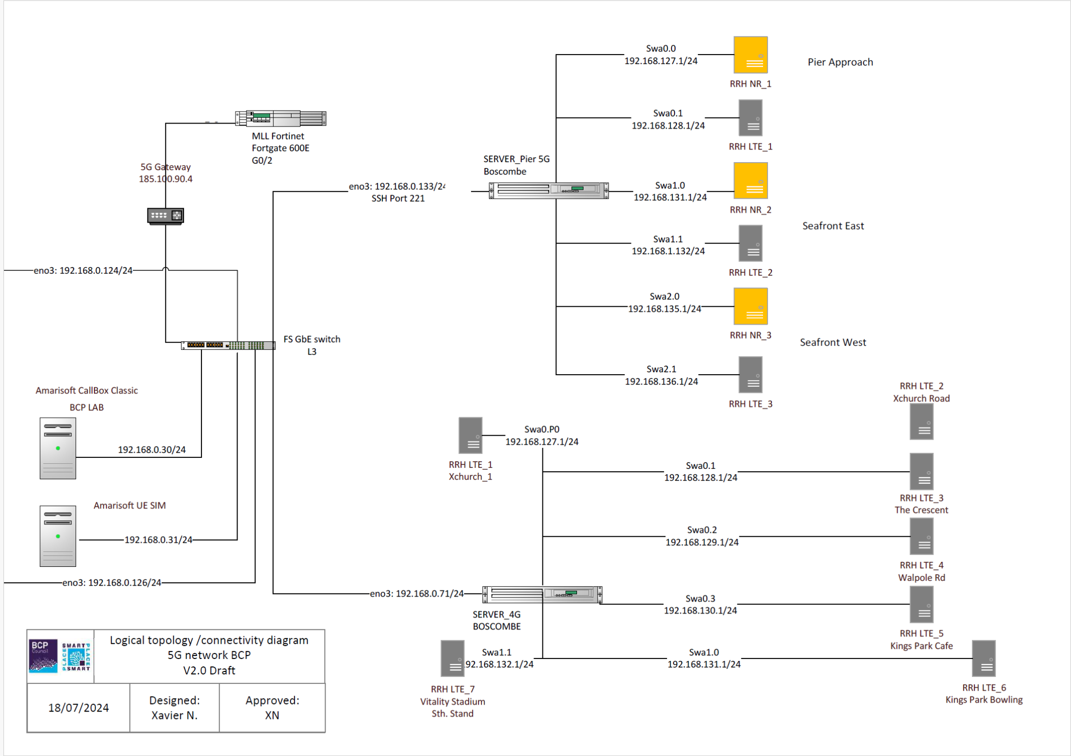
The 5G integration method will be defined by identifying the precise 5G Core CRAN/ORAN API interface points with the CAHN policy framework

Physical Architecture

5G Network

[5G Architecture.pdf](5G Private Network\5G Architecture.pdf)

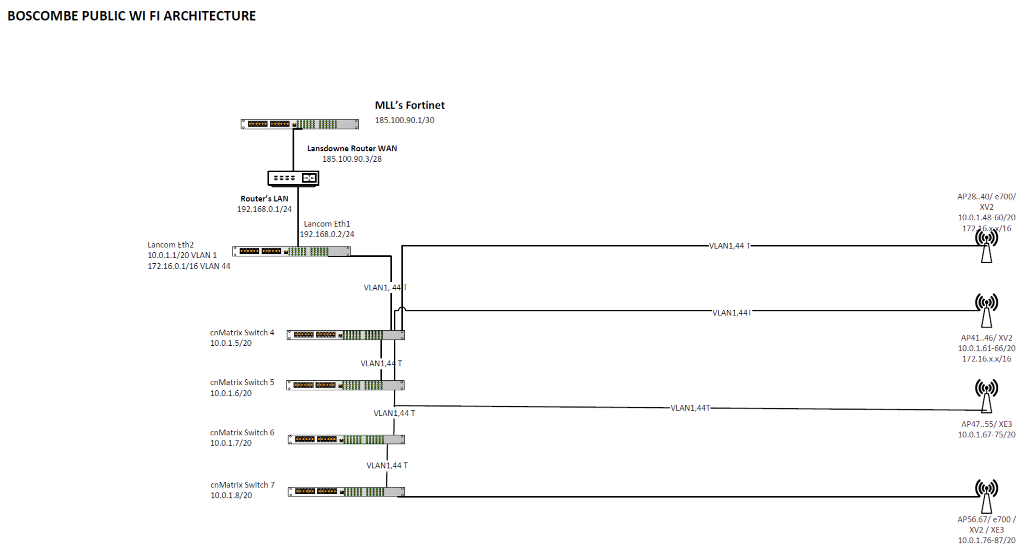
[5G Architecture.vsdx](5G Private Network\5G Architecture.vsdx)



WIFI Network

[Boscombe Network As Built.vsdx](Cambium - Public Wi Fi\Boscombe Network As Built.vsdx)

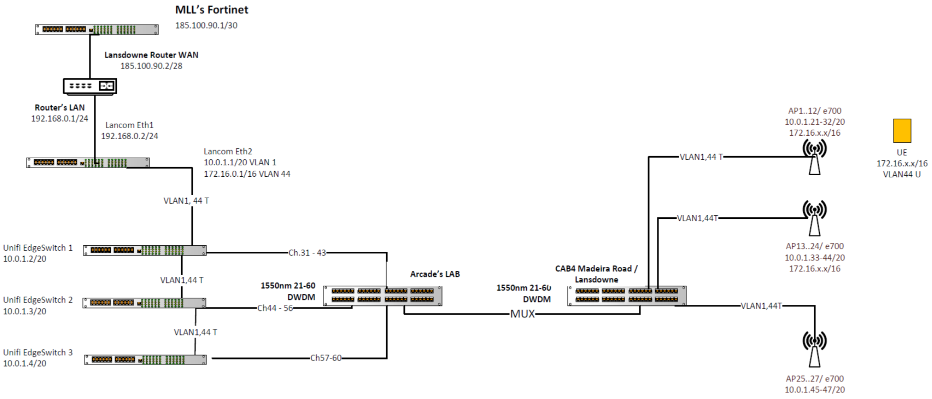
[Boscombe Wi Fi Architecture.pdf](Cambium - Public Wi Fi\Boscombe Wi Fi Architecture.pdf)



[Lansdowne Network As Built.vsdx](Cambium - Public Wi Fi\Lansdowne Network As Built.vsdx)

[Lansdowne WiFi architecture.pdf](Cambium - Public Wi Fi\Lansdowne WiFi architecture.pdf)

LANSDOWNE PUBLIC WI FI ARCHITECTURE



tdxVolt documentation

Skip to Content

tdx Volt

putting you in charge
☀ ☾
Toggle sidebar

- ## Getting Started
  - Welcome
  - Quick Start
- ## Concepts
  - Fundamentals
  - Identity
  - DID registry
  - Verifiable credentials
  - Policy
  - Resource
  - File
  - Database
  - Wire
  - Relay
  - Key strategy
- ## How to...
  - Create a Volt
  - Start a Volt
  - Connect to a Volt
  - Upload files
  - Download files
  - Publish to wire
  - Subscribe to wire
  - Create a database
  - Execute SQL
  - Import data
  - Establish a connect stream
  - Approve authenticate request
- ## Clients
  - Command line
  - Native / C++
  - Web
  - NodeJS

- ## Reference
    - Best practice
    - Battery
    - Configuration
    - Connection
    - Connect stream
    - Logging
    - PKCS#11
- ## API
    - Discovery API
    - File API
    - SqliteDatabase API
    - SqliteServer API
    - SSI API
    - Sync API
    - Relay API
    - Volt API
    - Wire API
- ## Utilities
    - protoDbSync
    - sqliteServer
    - wireTransform
- ## FAQ
    - Questions
- ## Coming soon
    - Roadmap
- ☀☾

NodeJS client

This document illustrates how to use the nodeJS client library to communicate with a Volt via grpc.

Install packages

Two packages are required by nodeJS **tdx Volt** clients, `@tdxvolt/volt-client-grpc` and `@grpc/grpc-js`.

To install the `@tdxvolt/volt-client-grpc` and `@grpc/grpc-js` packages:

```
npm install @tdxvolt/volt-client-grpc @grpc/grpc-js
```

Terminal window

Configuration

In order to be able to initialise a connection to a Volt, you will need to obtain a client configuration.

How your web application obtains or stores the client configuration is not discussed here, needless to say you need to be careful not to expose keys or other sensitive data. For the purposes of this document, we assume Bob's configuration is stored in a local file.

Example client connection object, obtained from the **fusebox** identity dialog:

```
const BobsConfig = {  client_name: "Bob",  credential: {
client_id: "25d050b9-5270-441e-920b-de07578394a9",    key:
"-----BEGIN ENCRYPTED PRIVATE KEY-----
\nMIIFHDBOBgkqhkiG9w0BBQ0wQTApBgkqhkiG9w0BBQwwHAQIRalkrxf/Lt0
-----END ENCRYPTED PRIVATE KEY-----\n",  },  volt: {    id:
"9a5944eb-1942-406e-a553-39b4220cbf94",    display_name:
"Bob's Volt",    address: "192.168.1.194:50393",
public_key:      "-----BEGIN PUBLIC KEY-----
\nMIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAshZ/IqKr1y0TeMg
-----END PUBLIC KEY-----\n",    fingerprint:
"5M1aCpnijWbmibq748AHnyG1qgpHMFmLi5UUeaGBGo8t",    ca_pem:
"-----BEGIN CERTIFICATE-----
\nMIIDojCCAoqgAwIBAgIEG/pk5zANBgkqhkiG9w0BAQsFADBxMQswCQYDVQQ
-----END CERTIFICATE-----\n",  },};
```

Connection

Begin by creating a **tdx Volt** client instance:

```
import grpc from "@grpc/grpc-js";import { VoltClient } from
"@tdxvolt/volt-client-grpc";
const client = new VoltClient(grpc);
```

Initialise the **tdx Volt** instance to enable Bob to connect to the Volt:

```
const configPath = "./volt-config.json";await
client.initialise(configPath);
```

API call

Once initialised successfully, you can issue any API call. In the example below, we retrieve the immediate descendants of the Volt root resource:

```
client  .initialise(configPath)  .then(() => {    return
client.GetResourceDescendants({      resource_id:
client.config.volt.id,      depth: 1,    });  })
.then((response) => {    console.log(response.descendant);
})  .catch((err) => {    console.error(err.message);  })
```

Refer to the API documentation for details of the method names and corresponding request and response messages. Each API method expects a JSON object representing the request as input, and returns a JSON object as indicated by the response message type. The JSON representation of the protobuf message format is intuitive.

If you see deprecation warnings along the lines of `Setting the TLS ServerName to an IP address is not permitted by RFC 6066` you can safely ignore them for the time being. See also this github issue.

Unary calls

All unary calls return a promise that will either resolve with a JSON response object matching that defined by the API protobuf, or reject if there is a problem with the grpc transport or an error status on the response object.

Streaming calls

The promise model does not fit well with streaming calls since they are long-lived.

For this reason, all streaming calls (client streaming, server streaming, and bi-directional streaming) return a `call` object that is used to manage the call.

The `call` object emits events as interactions with the underlying websocket occur, and there are 3 events of interest:

- “error” emitted when an error occurs on the call
- “data” emitted when a response is received from the Volt
- “end” emitted when the call ends

Server streaming calls

The example below shows execution of an SQL statement on a database, the `data` event will be emitted for each row of data in the result set.

The `end` event is emitted when the call ends.

```
const call = client.SqlExecute(  {    database_id:
"fc17c5a1-a858-45c6-804b-7e16af7c968d",    statement:
"SELECT * FROM NETFLIX LIMIT 100",  });
call.on("error", (err) => console.error(err.message));
call.on("data", (response) => console.log(response));
call.on("end", () => console.log("complete"));
```

Client streaming calls

Client streaming calls have a similar syntax to server streaming calls, although the `data` event should only be emitted once rather than multiple times.

Clients can use the `send` method on the returned `call` object to send requests to the server (see bi-direction example below).

Clients use the `end` method of the returned `call` object to indicate to the server that the call has ended.

Bi-directional streaming calls

As expected, bi-directional calls are a combination of client and server streaming calls, whereby the `data` event is emitted for each response from the server, and the client can send multiple requests using the `call` object `send` method.

```
const sub = voltApi.SubscribeWire({ wireId });
sub.on("error", (err) => console.error(err.message));
sub.on("data", (response) => console.log(response.chunk));
sub.on("end", () => console.log("subscription ended"));
//// ...//
// Some time later, ask the server to end the
subscriptionsub.send({ stop: true });
```

Skip to Content

tdx Volt

putting you in charge

☀☾

Toggle sidebar

- **Getting Started**
  - Welcome
  - Quick Start
- **Concepts**
  - Fundamentals
  - Identity

  - DID registry
  - Verifiable credentials
  - Policy
  - Resource
  - File
  - Database
  - Wire
  - Relay
  - Key strategy

- ## How to...

  - Create a Volt
  - Start a Volt
  - Connect to a Volt
  - Upload files
  - Download files
  - Publish to wire
  - Subscribe to wire
  - Create a database
  - Execute SQL
  - Import data
  - Establish a connect stream
  - Approve authenticate request

- ## Clients

  - Command line
  - Native / C++
  - Web
  - NodeJS

- ## Reference

  - Best practice
  - Battery
  - Configuration
  - Connection
  - Connect stream
  - Logging
  - PKCS#11

- ## API

  - Discovery API
  - File API
  - SqliteDatabase API
  - SqliteServer API
  - SSI API
  - Sync API
  - Relay API
  - Volt API
  - Wire API

- ## Utilities

  - protoDbSync
  - sqliteServer
  - wireTransform

- **FAQ**
    - Questions
- **Coming soon**
    - Roadmap
- ☀☾

Web client

There are two means of accessing the **tdx Volt** from web applications, via the HTTP interface or using the websocket API.

The websocket API is recommended because it maps well to the grpc streaming modes.

HTTP interface

The HTTP interface is a work in progress. The **tdx Volt** API is accessible using pure REST requests, but a javascript library that packages this up is not fully implemented yet.

Websocket API

The websocket API is currently more polished than the HTTP interface.

Begin by installing the `@tdxvolt/volt-client-web` package:

```
npm install @tdxvolt/volt-client-web
```

Terminal window

Configuration

In order to be able to initialise a connection to a Volt, you will need to obtain a client configuration.

How your web application obtains or stores the client configuration is not discussed here, needless to say you need to be careful not to expose keys or other sensitive data. For the purposes of this document, we assume Bob's configuration is obtained via some out-of-band means and stored in the browser `localStorage`. It holds Bob's key in encrypted form. The web application will then prompt Bob to enter the key passphrase in order to unlock it.

Example client connection object, obtained from the **fusebox** identity dialog:

```
const BobsConfig = {  client_name: "Bob",  credential: {
client_id: "25d050b9-5270-441e-920b-de07578394a9",    key:
"-----BEGIN ENCRYPTED PRIVATE KEY-----
\nMIIFHDBOBgkqhkiG9w0BBQ0wQTApBgkqhkiG9w0BBQwwHAQIRalkrxf/Lt0
-----END ENCRYPTED PRIVATE KEY-----\n",  },  volt: {    id:
"9a5944eb-1942-406e-a553-39b4220cbf94",    display_name:
"Bob's Volt",    address: "192.168.1.194:50393",
public_key:      "-----BEGIN PUBLIC KEY-----
\nMIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAshZ/IqKr1y0TeMg
-----END PUBLIC KEY-----\n",    fingerprint:
"5M1aCpnijWbmibq748AHnyG1qgpHMFmLi5UUeaGBGo8t",    ca_pem:
"-----BEGIN CERTIFICATE-----
\nMIIDojCCAoqgAwIBAgIEG/pk5zANBgkqhkiG9w0BAQsFADBxMQswCQYDVQC
-----END CERTIFICATE-----\n",  },};
```

Connection

Create a **tdx Volt** instance to enable Bob to connect to the Volt:

```
import { VoltClient } from "@tdxvolt/volt-client-web";
const client = new VoltClient(window.WebSocket, BobsConfig);
```

Now attempt to initialise the **tdx Volt** connection:

```
await client.initialise();
```

If initialisation fails, an error will be thrown.

API call

Once initialised successfully, you can issue any API call.

Refer to the API documentation for details of the method names and corresponding request and response messages. Each API method expects a JSON object representing the request(s) as input, and returns JSON objects as indicated by the response message type. The JSON representation of the protobuf message format is intuitive.

In the example below, we retrieve the resource associated with Bob's home folder:

```
client  .initialise()  .then(() => {    return
client.GetResource({ resourceId: config.credential.client_id
});  })  .then((response) => {
console.log(response.resource);  })  .catch((err) => {
console.error(err.message);  });
```

Unary calls

All unary calls return a promise that will either resolve with a JSON response object matching that defined by the API protobuf, or reject if there is a problem with the grpc transport or an error status on the response object.

Streaming calls

The promise model does not fit well with streaming calls since they are long-lived.

For this reason, all streaming calls (client streaming, server streaming, and bi-directional streaming) return a `call` object that is used to manage the call.

The `call` object emits events as interactions with the underlying websocket occur, and there are 3 events of interest:

- "error" emitted when an error occurs on the call
- "data" emitted when a response is received from the Volt
- "end" emitted when the call ends

Server streaming calls

The example below shows execution of an SQL statement on a database, the `data` event will be emitted for each row of data in the result set.

The `end` event is emitted when the call ends.

```
const call = client.SqlExecute(  {    database_id:
"fc17c5a1-a858-45c6-804b-7e16af7c968d",    statement:
"SELECT * FROM NETFLIX LIMIT 100",  });
call.on("error", (err) => console.error(err.message));
call.on("data", (response) => console.log(response));
call.on("end", () => console.log("complete"));
```

Client streaming calls

Client streaming calls have a similar syntax to server streaming calls, although the `data` event should only be emitted once rather than multiple times.

Clients can use the `send` method on the returned `call` object to send requests to the server (see bi-direction example below).

Clients use the `end` method of the returned `call` object to indicate to the server that the call has ended.

Bi-directional streaming calls

As expected, bi-directional calls are a combination of client and server streaming calls, whereby the `data` event is emitted for each response from the server, and the client can send multiple requests using the `call` object `send` method.

```
const sub = voltApi.SubscribeWire({ wireId });
sub.on("error", (err) => console.error(err.message));
sub.on("data", (response) => console.log(response.chunk));
sub.on("end", () => console.log("subscription ended"));
//// ...//
// Some time later, ask the server to end the
subscriptionsub.send({ stop: true });
```

Skip to Content

tdx Volt

putting you in charge
☀☾
Toggle sidebar

- ## Getting Started
  - Welcome
  - Quick Start
- ## Concepts
  - Fundamentals
  - Identity
  - DID registry
  - Verifiable credentials
  - Policy
  - Resource
  - File
  - Database
  - Wire
  - Relay
  - Key strategy
- ## How to...
  - Create a Volt
  - Start a Volt
  - Connect to a Volt
  - Upload files
  - Download files
  - Publish to wire
  - Subscribe to wire
  - Create a database
  - Execute SQL
  - Import data
  - Establish a connect stream
  - Approve authenticate request

- **Clients**
  - Command line
  - Native / C++
  - Web
  - NodeJS
- **Reference**
  - Best practice
  - Battery
  - Configuration
  - Connection
  - Connect stream
  - Logging
  - PKCS#11
- **API**
  - Discovery API
  - File API
  - SqliteDatabase API
  - SqliteServer API
  - SSI API
  - Sync API
  - Relay API
  - Volt API
  - Wire API
- **Utilities**
  - protoDbSync
  - sqliteServer
  - wireTransform
- **FAQ**
  - Questions
- **Coming soon**
  - Roadmap
- ☀☾

Command line interface

Although not strictly a client library as such, the **tdx Volt** command line interface (CLI) exposes several aspects of the core **tdx Volt** API for use in shell scripts and other terminal-based processes. This, in combination with the **tdx Volt** `wire` functionality, provides opportunities for useful and succinct workflows.

The **tdx Volt** CLI can also act as a server, and can be daemonised to run one or more Volts in the background.

The **tdx Volt** CLI supports several commands, and the arguments and syntax of each command varies. The basic format to invoke a command is:

```
./volt <command> [options]
```

Help is available for each command using the `-h` or `--help` switch, for example:

```
./volt create --help
```

Terminal window

tdx Volt management commands

The commands in this section relate to **tdx Volt** management tasks, such as creating and running Volts.

battery options

There are some options that are common to the **tdx Volt** management commands, including those to do with configuration of the Battery in which the **tdx Volt** is contained.

The Battery name can be configured using the `-b` switch. If this switch is omitted, the default Battery is used.

For example, to create a **tdx Volt** in a Battery named `Household`:

```
./volt create Alice -b Household
```

Terminal window

Use the `--bp` switch to specify an encryption passphrase and create a secure Battery:

```
./volt create Alice -b Household --bp foobar
```

Terminal window

As above, but using `--bp .` to request a passphrase prompt:

```
./volt create Alice -b Household --bp .
```

Terminal window

Note that once an Battery is encrypted, the passphrase will be required for any subsequent **tdx Volt** command. For example to start a **tdx Volt** that is stored in a passphrase-protected Battery:

```
./volt run 449a3385-f380-41f7-bd0a-e60caaa403cb -bp foobar
```

create command

The command to create a Volt.

In the following example, the **tdx Volt** will be created in the default Battery, with no passphrase protection or encryption on either the Battery storage or the **tdx Volt** itself.

```
./volt create "Alice's laptop"
```

Terminal window

To avoid having to copy and paste the UUID when subsequently referring to the **tdx Volt**, you can give it an alias using the `-a` switch:

```
./volt create "Alice's laptop" -a alice
```

Terminal window

The alias can be substituted for the UUID, by prefixing it with `@`:

```
./volt run @alice
```

Terminal window

Create a secure Volt

Use the `-p` switch to specify a passphrase that will be used to encrypt the **tdx Volt** root key and storage.

By default, the **tdx Volt** will auto-generate a new key at creation and encrypt it with the given passphrase. The **tdx Volt will not** store the passphrase so you must remember it.

It is recommended to use a phrase rather than a single word for the passphrase, for example "I like cheese".

```
./volt create "Alice's laptop" -p secret
```

Terminal window

Use a period with the passphrase switch (`-p .`) to force the CLI to prompt for the passphrase rather than include it explicitly in the command line.

```
./volt create "Alice's laptop" -p .
```

Terminal window

There is currently no way to recover or reset the **tdx Volt** passphrase so if you lose it you will not be able to access the Volt.

Create a tdx Volt using a fixed host

By default when a **tdx Volt** is created its certificate is bound to the current (or first) ipv4 network interface address. If you would like to bind a **tdx Volt** to a specific IP address or domain name use the `host` and `port` command line switches.

```
./volt create "Alice's server" --host aliceserver.com --port
40725
```

Terminal window

Create a tdx Volt with a file-based key

By default the **tdx Volt** key is stored with the **tdx Volt** configuration in the Battery. If the Battery is encrypted with a strong passphrase this is a fairly safe option.

Alternatively you can create a **tdx Volt** using a key stored on the local file system.

```
./volt create "Alice's laptop" -k /path/to/key/file
```

Terminal window

If `/path/to/key/file` does not exist it will be created.

If the key is encrypted (recommended) you can specify or prompt for the passphrase using the `-p` switch as described above.

```
./volt create "Alice's laptop" -k /path/to/key/file -p .
```

Terminal window

See the key strategy section for more information.

For full usage information:

```
./volt create --help
```

Terminal window

run command

Use the `run` command to start a Volt.

This command can be used to daemonise an instance of a Volt, using an `init.d` script or similar.

To run a **tdx Volt** that is in the default Battery and is not passphrase-protected:

```
./volt run f902f0f7-b9b7-4d2a-b05a-2d4e76a16ded
```

Terminal window

To run a passphrase-protected Volt, use the `-p .` option to prompt for a passphrase:

```
./volt run f902f0f7-b9b7-4d2a-b05a-2d4e76a16ded -p .
```

Terminal window

If the **tdx Volt** is not in the default Battery, specify the path to the Battery using the `-l` option:

```
./volt run f902f0f7-b9b7-4d2a-b05a-2d4e76a16ded -p . -b
Household
```

Terminal window

For full usage information:

```
./volt run --help
```

Terminal window

config command

The `config` command can be used to show the available **tdx Volt** configurations, or see the detail of a particular **tdx Volt** configuration.

To list all Volts in the default Battery:

```
./volt config
```

Terminal window

To list all Volts in a given Battery:

```
./volt config -b Household
```

Terminal window

To view the full configuration for a given Volt, specify the **tdx Volt** id or alias:

```
./volt config @alice
```

Terminal window

For full usage information:

```
./volt config --help
```

Terminal window

Client commands

The commands in this section are classed as **tdx Volt** client commands, and as such they require client credentials in order to authenticate with the target Volt.

There are two ways to specify the client credentials, either via a well-known file named `volt.config.json`, or using a custom file name and passing it via the command line switch `-c`.

See the connection section for more details about client credentials.

The following examples assume that the client credentials have been placed in the `volt.config.json` file.

download command

Download a file or folder from the **tdx Volt** to the local disk:

```
./volt download ./shares/images volt-downloads
```

Terminal window

In the example above, the `.` prefix of `./shares/images` indicates that the path is relative the ‘home’ folder of the client.

To specify a path relative to the root of the **tdx Volt** itself, use a `/` prefix:

```
./volt download /documents/reports volt-downloads
```

Terminal window

For full usage information:

```
./volt download --help
```

Terminal window

upload command

Upload a file or folder to the Volt.

```
./volt upload ~/Documents/images ./shares
```

Terminal window

The `.` prefix on `./shares` indicates the target resource is relative to the ‘home’ folder of the client.

To upload to a resource relative to the root of the **tdx Volt** itself, use a `/` prefix:

```
./volt upload ~/Documents/images /documents/images
```

Terminal window

For full usage information:

```
./volt upload --help
```

Terminal window

list command

List resources on the **tdx Volt** (similar to `ls` or `dir` shell commands).

```
./volt list ./shares
```

Terminal window

For full usage information:

```
./volt list --help
```

Terminal window

wire command

Subscribe or publish to wire resources.

publish

To publish to a wire, specify the wire identifier or alias:

```
./volt wire @sensor-feed
```

Terminal window

The wire data is read from `STDIN`. This enables flows such as:

```
curl www.google.com | ./volt wire bcc778cd-04b0-4e5e-803c-
dcca143790e1cat somefile | ./volt wire bcc778cd-04b0-4e5e-
803c-dcca143790e1
```

Terminal window

subscribe

To subscribe to a wire, add the `-s` switch to the command line:

```
./volt wire 9c797959-eafe-4d2a-8005-1e5ef66c29ac -s
```

Terminal window

All wire data will be written to STDOUT. This enables flows such as:

```
./volt wire -s bcc778cd-04b0-4e5e-803c-dcca143790e1 | wire-
data.log
```

Terminal window

For full usage information:

```
./volt wire --help
```

Terminal window

Utility commands

logger command

Split data arriving on a wire or `STDIN` into files on the local disk.

This command was developed for use with the Protobuf Database Sync utility, but it may be useful in other scenarios.

The `logger` command listens for incoming data on a Volt wire or `STDIN`, splitting the input into multiple files of a configured maximum size in a configured folder. It makes no attempt to interpret the incoming data.

If a header is given as part of the **tdx Volt** Logger configuration it will populate and write a `ProtobufSyncConfigurationHeader` message to the start of each file it creates.

Once a file reaches the configured size (`logFileSize`) it will be moved to the ingestion folder, where it will get picked up by the **protoDbSync** process if one is running. The logger will then create a new file to continue logging the incoming data.

This enables scenarios such as:

```
# Continuously pipe data from a producer, splitting it into
log files.> fs20PacketProducer | volt logger -c
fs20.logger.json
```

Terminal window

```
# Continuously pipe data from a tdx Volt Wire (grpc stream),
splitting it into log files.> volt logger -c
zwave.logger.json -w d1cc1560-9b58-4bdf-a2cf-7d65a2db0c01
```

Terminal window

```
# Load data from a URL.> curl https://somedata.endpoint |
volt logger -c snapshot.logger.json
```

Terminal window

An example logger configuration when reading from `STDIN` is shown below.

```
{  "logger": {    "headerId": "tcpdump-logger",
"headers": [      {        "messageName": "TCPDumpPacket",
"messageProto": "syntax = \"proto3\";\n\npackage
tranforms;\n\nmessage TCPDumpPacket {\n  string timestamp =
1;\n  string source_mac_address = 2;\n  string
source_manufacturer_id = 3;\n  bool is_broadcast = 4;\n
bool is_arp = 5;\n  string target_mac_address = 6;\n  string
target_manufacturer_id = 7;\n  string ether_type = 8;\n
string unknown_1 = 9;\n  int32 length = 10;\n  string
source_address = 11;\n  string target_address = 12;\n
string payload = 13;\n}\n",        "name": "header0",
"tableName": "tcpdump"      }    ],    "logFileExtension":
"pdat",    "logFilePath": "./logs",    "logFilePrefix":
"tcpdump-log-",    "logFileSize": 64000  }}
```

If reading from a wire, the configuration should include details of a standard client connection, as shown below. The credentials given should have `subscribe` access to the source wire.

```
{  "logger": {    "headerId": "tcpdump-logger",
"headers": [      {        "messageName": "TCPDumpPacket",
"messageProto": "syntax = \"proto3\";\n\npackage
tranforms;\n\nmessage TCPDumpPacket {\n  string timestamp =
1;\n  string source_mac_address = 2;\n  string
source_manufacturer_id = 3;\n  bool is_broadcast = 4;\n
bool is_arp = 5;\n  string target_mac_address = 6;\n  string
target_manufacturer_id = 7;\n  string ether_type = 8;\n
string unknown_1 = 9;\n  int32 length = 10;\n  string
source_address = 11;\n  string target_address = 12;\n
string payload = 13;\n}\n",        "name": "header0",
"tableName": "tcpdump"      }    ],    "logFileExtension":
"pdat",    "logFilePath": "./logs",    "logFilePrefix":
"tcpdump-log-",    "logFileSize": 64000  },  "client_name":
"CLI",  "crypto": {    "cert": "-----BEGIN CERTIFICATE-----
\nMIIDWzCCAkOgAwIBAgIEQ7J4qTANBgkqhkiG9w0BAQsFADBxMQswCQYDVQQ
-----END CERTIFICATE-----\n",    "client_id": "576d3f07-0aab-
4ced-b10f-085220c2b1ff",    "key": "-----BEGIN PRIVATE KEY--
---
\nMIIEvQIBADANBgkqhkiG9w0BAQEFAASCBKcwggSjAgEAAoIBAQCmVwOOkfI
-----END PRIVATE KEY-----\n"  },  "volt": {    "ca_pem": "--
---BEGIN CERTIFICATE-----
\nMIIDojCCAoqgAwIBAgIENzfzHjANBgkqhkiG9w0BAQsFADBxMQswCQYDVQQ
-----END CERTIFICATE-----\n",    "challenge_code":
"w6uP8Tcg6K2QR905Rms8iXTlksL6OD1KOWBxTK7wxPI=",    "id":
"3803c239-ffc6-40a3-b85c-1da798945c3c",    "address":
"192.168.1.139:40265"  }}
```

STDIN logger performance

Reading asynchronously from `STDIN` is surprisingly difficult. The method used by the logger command involves a background thread that reads data from `STDIN` one byte at a time, and when the accumulated data reaches the configured log size the data is flushed to disk.

On its own this approach causes problems whereby a flush might occur in the middle of an incoming packet, thus breaking the protobuf binary format.

To guard against this, the Volt logger uses a second thread thread which is responsible for flushing the buffer, and will only do so after 50ms of inactivity on `STDIN` (the interval is configurable via the `flushIdleInterval` property in the configuration file). As a consequence, the produced log file size may slightly exceed that configured in `logFileSize`.

The flushing thread described above helps reduce the frequency of split packets, but they can still occur due to network latency or other timing blips on the incoming data. Fortunately the **protoDbSync** utility can recover from these split packets the majority of the time.

Support for more commands in the Volt CLI is coming soon...

Skip to Content

tdx Volt

putting you in charge
☀☾
Toggle sidebar

- ## Getting Started

  - Welcome
  - Quick Start

- ## Concepts

  - Fundamentals
  - Identity
  - DID registry
  - Verifiable credentials
  - Policy
  - Resource
  - File
  - Database
  - Wire
  - Relay
  - Key strategy

- ## How to...

  - Create a Volt
  - Start a Volt
  - Connect to a Volt
  - Upload files
  - Download files
  - Publish to wire
  - Subscribe to wire
  - Create a database
  - Execute SQL
  - Import data
  - Establish a connect stream
  - Approve authenticate request

- ## Clients

  - Command line
  - Native / C++
  - Web
  - NodeJS

- ## Reference

  - Best practice
  - Battery
  - Configuration
  - Connection
  - Connect stream
  - Logging
  - PKCS#11

- ## API

  - Discovery API
  - File API
  - SqliteDatabase API
  - SqliteServer API
  - SSI API
  - Sync API
  - Relay API
  - Volt API
  - Wire API

- ## Utilities

  - protoDbSync
  - sqliteServer
  - wireTransform

- ## FAQ

  - Questions

- ## Coming soon

  - Roadmap
- ☀ ☾

C++ client

The C++ API is in a constant state of flux at the moment, documentation will appear in the near future.

Coming soon…

Skip to Content

tdx Volt

putting you in charge

☀ ☾

Toggle sidebar

- ## Getting Started

  - Welcome
  - Quick Start

- ## Concepts

  - Fundamentals
  - Identity
  - DID registry
  - Verifiable credentials
  - Policy
  - Resource

- File
- Database
- Wire
- Relay
- Key strategy

How to...

- Create a Volt
- Start a Volt
- Connect to a Volt
- Upload files
- Download files
- Publish to wire
- Subscribe to wire
- Create a database
- Execute SQL
- Import data
- Establish a connect stream
- Approve authenticate request

Clients

- Command line
- Native / C++
- Web
- NodeJS

Reference

- Best practice
- Battery
- Configuration
- Connection
- Connect stream
- Logging
- PKCS#11

API

- Discovery API
- File API
- SqliteDatabase API
- SqliteServer API
- SSI API
- Sync API
- Relay API
- Volt API
- Wire API

Utilities

- protoDbSync
- sqliteServer
- wireTransform

FAQ

- Questions

- **Coming soon**
    - Roadmap
- ☀ ☾

FAQ

Frequently asked questions and trouble-shooting tips are shown below.

Errors

I get `Volt IP invalid` error when starting a tdx Volt

This usually means there is no network connection.

I get `invalid argument` errors when attempting to bind or connect to a tdx Volt

This usually means target **tdx Volt** has no knowledge of your public key and you didn't provide any further credentials or the challenge code. Add the correct `challenge_code` property to the `volt` configuration section.

Linked Volt connections and Relays are broken

Currently Relay and Linked Volt connections are based on IP address. If one of the Volts has recently changed IP address it will cause any connections to it to fail. This will change when the universal identity resolution mechanism is in place, in the mean time you can work around it by deleting the 'shared with me on xxxxx' folder and re-discovering the Volt.

Skip to Content

tdx Volt

putting you in charge
☀ ☾
Toggle sidebar

- **Getting Started**
    - Welcome
    - Quick Start
- **Concepts**
    - Fundamentals
    - Identity
    - DID registry
    - Verifiable credentials
    - Policy
    - Resource
    - File
    - Database
    - Wire
    - Relay
    - Key strategy

- ## How to...
  - Create a Volt
  - Start a Volt
  - Connect to a Volt
  - Upload files
  - Download files
  - Publish to wire
  - Subscribe to wire
  - Create a database
  - Execute SQL
  - Import data
  - Establish a connect stream
  - Approve authenticate request
- ## Clients
  - Command line
  - Native / C++
  - Web
  - NodeJS
- ## Reference
  - Best practice
  - Battery
  - Configuration
  - Connection
  - Connect stream
  - Logging
  - PKCS#11
- ## API
  - Discovery API
  - File API
  - SqliteDatabase API
  - SqliteServer API
  - SSI API
  - Sync API
  - Relay API
  - Volt API
  - Wire API
- ## Utilities
  - protoDbSync
  - sqliteServer
  - wireTransform
- ## FAQ
  - Questions
- ## Coming soon
  - Roadmap
- ☀🌙

Connect to a tdx Volt

The **tdx Volt** command line interface (CLI) and all client libraries make use of a JSON object stored in a local file to persist the details of a client connection to the Volt.

A full description of the client connection JSON format and how to obtain or create one can be found in the connection section.

api

The VoltAPI.Authenticate API enables clients to authenticate on a Volt and obtain credentials for subsequent API calls.

cli

Use the `auth` command.

```
./volt auth -help
```

Terminal window

To authenticate on a Volt with the DID `did:volt:a06e10d1-3fa4-445a-948f-7e7c5ee36262` use the following command:

```
./volt auth did:volt:a06e10d1-3fa4-445a-948f-7e7c5ee36262
"Bob"
```

Terminal window

By default, the configuration details will be stored in a file called `volt.config.json` in the current working directory. You can specify a different file using the `-c` switch:

```
./volt auth did:volt:a06e10d1-3fa4-445a-948f-7e7c5ee36262
"Bob" -c bob.config.json
```

Terminal window

You can use the discovery URL of the Volt instead of the DID:

```
./volt auth https://acme.com "Bob" -c bob.config.json
```

Terminal window

javascript

Use the Authenticate API, the response will contain the generated credentials on success.

grpc

It’s not normally necessary to explicitly use the `Authenticate` API when using the grpc client library, as the library will automatically authenticate if required when `initialise` is called.

It’s worth noting that the `initialise` call will not return until either a ‘permit’ or ‘deny’ decision is received from the Volt. This means that the `initialise` call will block if there is a ‘prompt’ decision, which will occur if the Volt is configured to require user interaction for authentication.

```
import { VoltClient } from "@tdxvolt/volt-client-
grpc";import grpc from "@grpc/grpc-js";
// This is the path where we'd like to store the generated
configuration.const configPath = "./volt.config.json";
const voltConfig = {};
// This is the name we'd like to use for our
client.voltConfig.client_name = "Alice";
// This is the DID of the Volt we'd like to connect
to.voltConfig.volt = "did:volt:a06e10d1-3fa4-445a-948f-
7e7c5ee36262";
// Alternatively, you can use the URL of the Volt instead of
the DID.voltConfig.volt = "http://192.168.1.195:49824";
const voltClient = new VoltClient(grpc);
voltClient  .initialise(configPath, voltConfig)
.then((response) => {    console.log(response);  })
.catch((err) => {    console.error(err.message);  })
.finally(() => {    console.log("finished");  });
```

Once the client has been initialised, the configuration will be stored in the file specified by `configPath`. The next time the client is initialised, the configuration will be read from this file.

If you'd rather manage the persistence of the configuration yourself, you can pass a configuration object directly to the `initialise` method rather than a file path:

```
import { VoltClient } from "@tdxvolt/volt-client-
grpc";import grpc from "@grpc/grpc-js";
// Initialise or load the configuration here.const
voltConfig = {};
// This is the name we'd like to use for our
client.voltConfig.client_name = "Alice 2";
// This is the DID of the Volt we'd like to connect
to.voltConfig.volt = "did:volt:a06e10d1-3fa4-445a-948f-
7e7c5ee36262";
const voltClient = new VoltClient(grpc);
voltClient  .initialise(voltConfig)  .then((response) => {
// Persist the configuration here.    console.log(response);
})  .catch((err) => {    console.error(err.message);  })
.finally(() => {    console.log("finished");  });
```

web

```
  const sub = client.Authenticate({    client_name: "Bob",
"public_key": "<PEM encoded key>"  });
```

C++

```
examples coming soon...
```

Skip to Content

tdx Volt

putting you in charge
☀☾
Toggle sidebar

- ## Getting Started
  - Welcome
  - Quick Start
- ## Concepts

  - Fundamentals
  - Identity
  - DID registry
  - Verifiable credentials
  - Policy
  - Resource
  - File
  - Database
  - Wire
  - Relay
  - Key strategy

- ## How to...

  - Create a Volt
  - Start a Volt
  - Connect to a Volt
  - Upload files
  - Download files
  - Publish to wire
  - Subscribe to wire
  - Create a database
  - Execute SQL
  - Import data
  - Establish a connect stream
  - Approve authenticate request

- ## Clients

  - Command line
  - Native / C++
  - Web
  - NodeJS

- ## Reference

  - Best practice
  - Battery
  - Configuration
  - Connection
  - Connect stream
  - Logging
  - PKCS#11

- ## API

  - Discovery API
  - File API
  - SqliteDatabase API
  - SqliteServer API
  - SSI API
  - Sync API
  - Relay API
  - Volt API
  - Wire API

- ## Utilities

  - protoDbSync
  - sqliteServer
  - wireTransform

- **FAQ**
    - Questions
- **Coming soon**
    - Roadmap
- ☀🌙

Establish a connect stream

api

The VoltAPI.Connect API enables clients to establish a Volt **connect stream**. Note that clients will not need to call this API directly as it is encapsulated in the various ‘Connection’ classes in the client libraries.

cli

The CLI automatically uses **connect streams** to implement certain commands, in particular the wire publish and subscribe commands. This means that when you subscribe to a wire using the CLI, the subscription will automatically be restarted if the connection to the Volt is lost and subsequently re-established.

fusebox

The fusebox always establishes a **connect stream** to the target Volt, and this is used to reflect the connection status to the user and to receive notification of changes to resources that are in view.

javascript [grpc]

Use the `VoltClient.connect` method to establish a Volt **connect stream**, and register to receive `connected` events for notifications.

```
import grpc from "@grpc/grpc-js";import { VoltClient } from
"@tdxvolt/volt-client-grpc";
const client = new VoltClient(grpc);
const config = "./volt.config.json";
// Register to receive connection
events.client.on("connected", (connected) => {  if
(connected) {    console.log("connected");  } else {
console.log("disconnected");  }});
// Register to receive error events.client.on("error", (err)
=> {  console.log("error: %s", err.message);});
client  .initialise("connect-example", config)  .then(() =>
{    return client.connect();  })  .then(() => {
console.log("client waiting for events");  })  .catch((err)
=> {    console.log("failure in client [%s]", err.message);
process.exit(1);  });
```

Javascript [web]

This is a work in progress.

```
import { VoltClient } from "@tdxvolt/volt-client-web";import
{ config } from "./client-configuration.js";
let client;
try {  client = new VoltClient(WebSocket, config);
  await client.initialise();  console.log("client
initialised");
  const connectStream = client.connect(    {      hello: {},
},    (err, resp) => {      if (err) {
console.log("received error on connect: " + err.message);
} else {        console.log(JSON.stringify(resp));      }
}  );  await connectStream.responsePromise;} catch (err) {
console.log("failure in connect example [%s]",
err.message);} finally {  console.log("finished");
client.close();}
```

C++

```
#include <volt_client/volt_api_client.h>
...
tdx::volt_client::VoltConnectCallbacks
voltCallbacks;voltCallbacks.onConnection = [](bool
connected) {  // Perform connect/disconnect logic here.
std::cout << "connected: " << connected << std::endl;};
// Create and initialise the Volt client connection with the
configuration// details and the key passphrase.auto volt =
std::make_unique<tdx::volt_client::VoltConnection>();if
((result = volt->initialise(config, passphrase)) !=
tdx::error_code::ok) {  std::cout << "failure initialising
connection: " << result << std::endl;  return result;}
// Wait for the Volt connection to initialise successfully
(this// may fail if the Volt is offline or
unreachable).result = tdx::error_code::notInitialised;while
(!result.empty()) {  if ((result = volt-
>connect(voltCallbacks)) != tdx::error_code::ok) {
std::cout << "failure connecting to Volt: " << result <<
std::endl;    std::cout << "retrying in 10 seconds......" <<
std::endl;
std::this_thread::sleep_for(std::chrono::milliseconds(10000))
;  }}
// Once the initial connection is successfully established,
the `onConnection`// callback will be called, including for
subsequent disconnection/connection// events.
```

Skip to Content

tdx Volt

putting you in charge

☀ ☾

Toggle sidebar

- ## Getting Started
  - Welcome
  - Quick Start
- ## Concepts
  - Fundamentals
  - Identity
  - DID registry
  - Verifiable credentials

- Policy
- Resource
- File
- Database
- Wire
- Relay
- Key strategy

How to...

- Create a Volt
- Start a Volt
- Connect to a Volt
- Upload files
- Download files
- Publish to wire
- Subscribe to wire
- Create a database
- Execute SQL
- Import data
- Establish a connect stream
- Approve authenticate request

Clients

- Command line
- Native / C++
- Web
- NodeJS

Reference

- Best practice
- Battery
- Configuration
- Connection
- Connect stream
- Logging
- PKCS#11

API

- Discovery API
- File API
- SqliteDatabase API
- SqliteServer API
- SSI API
- Sync API
- Relay API
- Volt API
- Wire API

Utilities

- protoDbSync
- sqliteServer
- wireTransform

FAQ

Subscribe to wire

Error handling is omitted from example code for brevity.

api

The WireAPI.SubscribeWire API enables clients to subscribe to a wire.

cli

Use the `wire` command.

```
./volt wire --help
```

Terminal window

Subscribe to a wire using the cli `wire` command along with the `-s` or `--subscribe` switch:

```
./volt wire <wire id> -s
```

Terminal window

All wire data will be written to STDOUT. This enables flows such as:

```
./volt wire @demo-wire -s | wire-data.log
```

Terminal window

fusebox

Navigate to the wire in the resource explorer and use the ‘subscribe’ button on the bottom panel, which contains a ‘headphones’ icon.

javascript

Use the SubscribeWire API, the callback will be called every time new data arrives.

grpc

```
import grpc from "@grpc/grpc-js";import { VoltClient } from
"@tdxvolt/volt-client-grpc";
const client = new VoltClient(grpc);
const configPath = "./volt.config.json";
client  .initialise(configPath)  .then(() => {    return new
Promise((resolve, reject) => {      const sub =
client.SubscribeWire({ wire_id: "@wire-demo" });
      sub.on("error", (err) => {      reject(err);      });
      sub.on("data", (response) => {
console.log(Buffer.from(response.chunk,
"base64").toString());      });
      sub.on("end", () => {      resolve();      });    });
})  .catch((err) => {    console.error("failure in
subscribe-wire [%s]", err.message);  });
```

web

```
  const sub = client.SubscribeWire({ wireId });
sub.on("error", (err) => {    console.error(err);  });
  sub.on("data", (response) => {
console.log(atob(response.chunk));  });
  sub.on("end", () => {    console.log("subscription
ended");  });
```

C++

```
examples coming soon...
```

Skip to Content

tdx Volt

putting you in charge

☀☾

Toggle sidebar

- ## Getting Started
  - Welcome
  - Quick Start
- ## Concepts
  - Fundamentals
  - Identity
  - DID registry
  - Verifiable credentials
  - Policy
  - Resource
  - File
  - Database
  - Wire
  - Relay
  - Key strategy
- ## How to...
  - Create a Volt
  - Start a Volt
  - Connect to a Volt
  - Upload files
  - Download files
  - Publish to wire
  - Subscribe to wire
  - Create a database
  - Execute SQL
  - Import data
  - Establish a connect stream
  - Approve authenticate request
- ## Clients
  - Command line

  - Native / C++
  - Web
  - NodeJS

- ## Reference

  - Best practice
  - Battery
  - Configuration
  - Connection
  - Connect stream
  - Logging
  - PKCS#11

- ## API

  - Discovery API
  - File API
  - SqliteDatabase API
  - SqliteServer API
  - SSI API
  - Sync API
  - Relay API
  - Volt API
  - Wire API

- ## Utilities

  - protoDbSync
  - sqliteServer
  - wireTransform

- ## FAQ

  - Questions

- ## Coming soon

  - Roadmap
- ☀☾

Start a Volt

A **tdx Volt** can be started using either the command-line interface (CLI) or the **fusebox**.

cli

Use the `run` cli command:

```
./volt run --help
```

Terminal window

The most simple way to start a **tdx Volt** is using the `id`:

```
./volt run did:volt:449a3385-f380-41f7-bd0a-e60caaa403cb
```

Terminal window

If the **tdx Volt** has been given an alias you can use this instead of the GUID:

```
./volt run @alice
```

Terminal window

If the **tdx Volt** is secure you will also need to specify the passphrase, using the `-p` switch:

```
./volt run did:volt:449a3385-f380-41f7-bd0a-e60caaa403cb -p foobar
```

Terminal window

To avoid specifying the passphrase on the command line, you can force a prompt using `-p .`:

```
./volt run did:volt:449a3385-f380-41f7-bd0a-e60caaa403cb -p .
```

Terminal window

See the CLI section for more details about starting a **tdx Volt**.

fusebox

The **fusebox** can be used to run local **tdx Volt**s.

Simply click on the name of the **tdx Volt** in the slide-out menu panel on the left-hand side:

Skip to Content

tdx Volt

putting you in charge
☀☾
Toggle sidebar

- **Getting Started**
  - Welcome
  - Quick Start
- **Concepts**
  - Fundamentals
  - Identity
  - DID registry
  - Verifiable credentials
  - Policy
  - Resource
  - File
  - Database
  - Wire
  - Relay
  - Key strategy
- **How to...**
  - Create a Volt
  - Start a Volt

- Connect to a Volt
- Upload files
- Download files
- Publish to wire
- Subscribe to wire
- Create a database
- Execute SQL
- Import data
- Establish a connect stream
- Approve authenticate request

Clients

- Command line
- Native / C++
- Web
- NodeJS

Reference

- Best practice
- Battery
- Configuration
- Connection
- Connect stream
- Logging
- PKCS#11

API

- Discovery API
- File API
- SqliteDatabase API
- SqliteServer API
- SSI API
- Sync API
- Relay API
- Volt API
- Wire API

Utilities

- protoDbSync
- sqliteServer
- wireTransform

FAQ

- Questions

Coming soon

- Roadmap

- ☀🌙

Approve authenticate request

When a client attempts to authenticate on a **tdx Volt**, the policy engine will determine if it has any rules that relate to the public key or credentials presented in the authentication request.

If the policy does not find any rules applicable to the presented credentials, it will prompt the owner of the **tdx Volt** to approve or reject the request.

Managing authentication requests

The **fusebox** can be used to manage authentication requests, using the ‘sessions’ button, highlighted below:

Clicking on the button highlighted above will display the list of pending requests on the **tdx Volt**.

You can use the ‘egg timer’ button to restrict the list to only show pending requests.

Clicking on a request in the list will prompt you to either permit or deny the request.

Skip to Content

tdx Volt

putting you in charge

☀ ☾

Toggle sidebar

- ## Getting Started

  - Welcome
  - Quick Start

- ## Concepts

  - Fundamentals
  - Identity
  - DID registry
  - Verifiable credentials
  - Policy
  - Resource
  - File
  - Database
  - Wire
  - Relay
  - Key strategy

- ## How to...

  - Create a Volt
  - Start a Volt
  - Connect to a Volt
  - Upload files
  - Download files
  - Publish to wire
  - Subscribe to wire
  - Create a database
  - Execute SQL

  - Import data
  - Establish a connect stream
  - Approve authenticate request

- **Clients**
  - Command line
  - Native / C++
  - Web
  - NodeJS

- **Reference**
  - Best practice
  - Battery
  - Configuration
  - Connection
  - Connect stream
  - Logging
  - PKCS#11

- **API**
  - Discovery API
  - File API
  - SqliteDatabase API
  - SqliteServer API
  - SSI API
  - Sync API
  - Relay API
  - Volt API
  - Wire API

- **Utilities**
  - protoDbSync
  - sqliteServer
  - wireTransform

- **FAQ**
  - Questions

- **Coming soon**
  - Roadmap
- ☀☾

Publish to wire

api

The WireAPI.PublishWire API enables clients to publish to a wire.

cli

Use the `wire` command.

```
./volt wire --help
```

Terminal window

Publish to a wire using the cli `wire` command:

```
./volt wire <wire id>
```

Terminal window

The wire data is read from STDIN. This enables flows such as:

```
curl www.google.com | ./volt wire @demo-wirecat somefile |
./volt wire @demo-wire./volt wire @demo-wire < anotherfile
```

Terminal window

fusebox

Navigate to the wire in the resource explorer and use the ‘publish’ button on the bottom panel, which contains a ‘microphone’ icon.

javascript

Use the PublishWire API write data to the wire.

web

```
const pub = voltApi.PublishWire({ wire_id: "@demo-wire" });
pub.on("error", (err) => console.error(err.message));
pub.on("data", (response) => console.log(response));
pub.on("end", () => console.log("publication ended"));
pub.send({ chunk: "hello" });pub.send({ chunk: "world" });
```

C++

```
  volt_client::WirePublishCallbacks publishCallbacks;
volt_client::WirePublishClient* publishRpc = nullptr;
  publishCallbacks.onEnd = [](bool serverSide) {    //
Publish call ended.  };
  publishCallbacks.onResponse =      []
(tdx::volt_api::volt::v1::PublishResponse const& response) {
if (response.has_status() &&
response.status().message() != error_code::ok) {          //
Publication ended with an error.        } else {          //
Publication ended OK.        }      };
  publishCallbacks.onDestroyed = []
(tdx::grpc::CallClientBase* rpcClient) {      // Publication
RPC has died.  };
  // Start the publish RPC.  if ((result = volt()-
>serviceApi()->publishWire(           publishCallbacks,
&publishRpc)) != error_code::ok) {    // Failed to start
wire publish RPC    return result;  }
  // Send the initial request containing the wire id.
tdx::volt_api::volt::v1::PublishRequest startReq;
startReq.set_wire_id("@demo-wire");  publishRpc-
>send(startReq);
```

Skip to Content

tdx Volt

putting you in charge
☀☾
Toggle sidebar

- **Getting Started**
  - Welcome
  - Quick Start
- **Concepts**
  - Fundamentals
  - Identity
  - DID registry
  - Verifiable credentials
  - Policy
  - Resource
  - File
  - Database
  - Wire
  - Relay
  - Key strategy
- **How to...**
  - Create a Volt
  - Start a Volt
  - Connect to a Volt
  - Upload files
  - Download files
  - Publish to wire
  - Subscribe to wire
  - Create a database
  - Execute SQL
  - Import data
  - Establish a connect stream
  - Approve authenticate request
- **Clients**
  - Command line
  - Native / C++
  - Web
  - NodeJS
- **Reference**
  - Best practice
  - Battery
  - Configuration
  - Connection
  - Connect stream
  - Logging
  - PKCS#11
- **API**

  - Discovery API
  - File API
  - SqliteDatabase API
  - SqliteServer API
  - SSI API
  - Sync API
  - Relay API
  - Volt API
  - Wire API

- **Utilities**

  - protoDbSync
  - sqliteServer
  - wireTransform

- **FAQ**

  - Questions

- **Coming soon**

  - Roadmap
- ☀ ☾

Execute SQL

api

The SqliteDatabaseAPI.Execute API enables clients to execute SQL against databases.

cli

Use the `db` command.

```
./volt db --help
```

Terminal window

To execute a statement against a database resource with alias `db-demo`:

```
./volt db @db-demo "select * from tcpdump limit 10"
```

Terminal window

You can also write to the database if you have the appropriate permissions:

```
./volt db @db-demo "create table foobar (id integer primary
key, key text, value text)"
```

Terminal window

```
./volt db @db-demo "insert into foobar (key,value) values
('hello','world')"
```

Terminal window

```
./volt db @db-demo "select * from foobar"
```

Terminal window

fusebox

The **fusebox** has a built-in SQL terminal. Select the database resource and then click the `view` button on the toolbar.

javascript

grpc

The following example runs an SQL SELECT statement against a database resource with alias `db-demo`. It assumes the client configuration is stored in the `volt.config.json` file in the current working directory.

```
import grpc from "@grpc/grpc-js";import { VoltClient } from
"@tdxvolt/volt-client-grpc";
const client = new VoltClient(grpc);
const configPath = "./volt.config.json";
client  .initialise(configPath)  .then(() => {    return
client      .SqlExecuteJSON({        database_id: "@db-
demo",        statement: "select * from tcpdump limit 10",
})      .then((response) => {
console.log(JSON.stringify(response, null, 2));      });  })
.catch((err) => {    console.error("failure in database-
execute [%s]", err.message);  });
```

web

The following example runs a parameterised query against the resource with alias `@query-example`, substituting the value `%cage%` into the first parameter. It assumes the configuration is store in `localStorage`.

```
import { VoltClient } from "@tdxvolt/volt-client-web";
const configJSON = localStorage.getItem("config");const
config = JSON.parse(configJSON);
const client = new VoltClient(WebSocket, config);
client  .initialise()  .then(() => {    return
client.SqlExecuteJSON({      database_id: "@query-example",
parameter: [{ string: "%cage%" }],    });  })  .then((rows)
=> {    console.log(JSON.stringify(rows, null, 2));  })
.catch((err) => {    console.log("failure [%s]",
err.message);  })  .finally(() => {
console.log("finished");    client.close();  });
```

C++

```
  volt_client::SqliteDatabaseExecuteCallbacks
executeCallbacks_;
volt_client::SqliteDatabaseExecuteClient* executeRpc_ =
nullptr;
  tdx::volt_api::data::v1::SqlExecuteRequest req;
req.set_database_id(resource_.id());
req.set_statement("select * from fs20Actuator");
req.set_page_size(100);
  executeCallbacks_.onResponse =      [this]
(tdx::volt_api::data::v1::SqlExecuteResponse const&
response) {        // Handle response here.      };
  executeCallbacks_.onEnd = [this](bool serverSide) {    if
(serverSide) {      // The server ended the execution rpc -
this means it has no more data to      // send (i.e. the
cursor is at then end).    }  };
  executeCallbacks_.onError = [this](std::string err) {
// Handle errors here.  };
  if ((result = api_->executeDatabase(req,
executeCallbacks_, &executeRpc_)) != error_code::ok) {    //
Handle failure starting call.  }
```

Skip to Content

tdx Volt

putting you in charge
☀ ☾
Toggle sidebar

- ## Getting Started
  - Welcome
  - Quick Start
- ## Concepts
  - Fundamentals
  - Identity
  - DID registry
  - Verifiable credentials
  - Policy
  - Resource
  - File
  - Database
  - Wire
  - Relay
  - Key strategy
- ## How to...
  - Create a Volt
  - Start a Volt
  - Connect to a Volt
  - Upload files
  - Download files
  - Publish to wire
  - Subscribe to wire
  - Create a database
  - Execute SQL
  - Import data
  - Establish a connect stream
  - Approve authenticate request
- ## Clients
  - Command line
  - Native / C++
  - Web
  - NodeJS
- ## Reference
  - Best practice
  - Battery
  - Configuration
  - Connection
  - Connect stream
  - Logging
  - PKCS#11

- **API**
  - Discovery API
  - File API
  - SqliteDatabase API
  - SqliteServer API
  - SSI API
  - Sync API
  - Relay API
  - Volt API
  - Wire API
- **Utilities**
  - protoDbSync
  - sqliteServer
  - wireTransform
- **FAQ**
  - Questions
- **Coming soon**
  - Roadmap
- ☀ ☾

Create a Volt

A **tdx Volt** can be created using either the command-line interface (CLI) or the **fusebox**.

cli

Use the `create` cli command:

```
./volt create --help
```

Terminal window

The most simple way to create a **tdx Volt** from the command line is shown below.

In this example, the **tdx Volt** will be created in the default Battery, with no password protection or encryption on either the Battery storage or the **tdx Volt** itself.

```
./volt create "Alice's laptop"
```

Terminal window

Using a YubiKey Hardware Security Module (HSM)

Use the `hardware` key strategy to create a **tdx Volt** that uses a YubiKey HSM to store the root key.

The `-k` switch specifies the PKCS#11 engine type, module path and slot number that identifies the YubiKey HSM.

The `-k` switch takes the form of a semicolon-separated string with the following format:

```
<engine id>;<module path>;<slot number>
```

For example:

```
pkcs11;/opt/homebrew/Cellar/opensc/0.24.0/lib/opensc-
pkcs11.so;01
```

You will need to adjust the path to the `opensc-pkcs11.so` library to match your installation.

An example command line to create a **tdx Volt** using a YubiKey HSM on macOS that has installed the OpenSC software using Homebrew is shown below. Here we are using the default PIN for the YubiKey HSM, you should change this to your own PIN using the `-p` switch.

```
./volt create "Alice" -s hardware -k
"pkcs11;/opt/homebrew/Cellar/opensc/0.24.0/lib/opensc-
pkcs11.so;01" -p 123456
```

Terminal window

See the PKCS#11 reference for more information about configuring the **tdx Volt** to use a PKCS#11 HSM.

Using an encrypted root key

Use the `p` switch to specify a password that will be used to encrypt the **tdx Volt** storage.

By default, the **tdx Volt** will auto-generate a new key at creation and encrypt it with the given password. The **tdx Volt will not** store the password so you must remember it.

```
./volt create "Alice's laptop" -p secret
```

Terminal window

Use a period `.` to force the CLI to prompt for the password rather than include it explicitly in the command line.

```
./volt create "Alice's laptop" -p .> enter Volt passphrase:
_
```

Terminal window

There is currently no way to recover or reset the **tdx Volt** password so if you lose it you will not be able to access the Volt.

Using a file-based key

By default the **tdx Volt** key is stored with the **tdx Volt** configuration. If the Battery and **tdx Volt** itself are encrypted this is a fairly safe option.

Alternatively you can create a **tdx Volt** using a key stored on the local file system.

```
./volt create "Alice's laptop" -k /path/to/key/file
```

Terminal window

If `/path/to/key/file` does not exist it will be created.

If the key is encrypted (recommended) you can specify or prompt for the password using the `p` switch as described above.

```
./volt create "Alice's laptop" -k /path/to/key/file -p .
```

Terminal window

It's possible to use this option to create a **tdx Volt** using a key stored on an encrypted USB stick for example. Of course the key file must be available to the **tdx Volt** when it is started.

Create a tdx Volt using a fixed host

By default when a **tdx Volt** is created its certificate is bound to the current (or first) ipv4 network interface address. If you would like to bind a **tdx Volt** to a specific IP address or domain name use the `host` and `port` command line switches.

```
./volt create "Alice's server" --host aliceserver.com --port
40725
```

Terminal window

Create a tdx Volt with a Relay

Use the `relay-address` and `relay-challenge` switches to create a **tdx Volt** with a pre-configured relay. This is useful in scenarios where the **tdx Volt** is created on remote devices, as it enables the configuration of a remote connection to the **tdx Volt** via the **fusebox** to complete the commissioning.

Note that you almost certainly want to use the `-k` switch to specify the file in which the **tdx Volt** key will be written. This is because you will need the key to be able to configure the remote **fusebox** connection.

```
./volt create --name "Alice's NAS" -k ./alice.key --relay-
address https://tdxvolt.com --relay-challenge letmein
```

Terminal window

When the **tdx Volt** is first run it will create and configure the Relay connection and issue a bind request to the Relay Volt. The Relay Volt owner/administrator may need to approve the bind request before the binding is complete.

fusebox

Creating a **tdx Volt** using the **fusebox** is straightfoward. Use the ‘new Volt’ button on the slide-out menu panel:

Then fill out the form:

Skip to Content

tdx Volt

putting you in charge

☀☾

Toggle sidebar

- ## Getting Started

  - Welcome
  - Quick Start

- ## Concepts

  - Fundamentals
  - Identity
  - DID registry
  - Verifiable credentials
  - Policy
  - Resource
  - File
  - Database
  - Wire
  - Relay

  - Key strategy

- **How to...**
  - Create a Volt
  - Start a Volt
  - Connect to a Volt
  - Upload files
  - Download files
  - Publish to wire
  - Subscribe to wire
  - Create a database
  - Execute SQL
  - Import data
  - Establish a connect stream
  - Approve authenticate request

- **Clients**
  - Command line
  - Native / C++
  - Web
  - NodeJS

- **Reference**
  - Best practice
  - Battery
  - Configuration
  - Connection
  - Connect stream
  - Logging
  - PKCS#11

- **API**
  - Discovery API
  - File API
  - SqliteDatabase API
  - SqliteServer API
  - SSI API
  - Sync API
  - Relay API
  - Volt API
  - Wire API

- **Utilities**
  - protoDbSync
  - sqliteServer
  - wireTransform

- **FAQ**
  - Questions

- **Coming soon**

- Roadmap
- ☀🌙

Import data

api

The SqliteDatabaseAPI.ImportCSV API enables clients to import CSV file(s) into a database.

cli

Coming soon…

fusebox

Use the ‘Import from file’ button on the database terminal toolbar, which is visible when you select a database in the **fusebox** and click the ‘View’ button.

Alternatively, you can drag a CSV file using the OS file manager and drop it onto the ‘SQL statement’ edit field at the bottom of the SQL Terminal window (below the import button highlighted above).

javascript

Use the ImportCSV API.

```
examples coming soon...
```

C++

```
examples coming soon...
```

Skip to Content

tdx Volt

putting you in charge
☀🌙
Toggle sidebar

- ## Getting Started
  - Welcome
  - Quick Start
- ## Concepts
  - Fundamentals
  - Identity
  - DID registry
  - Verifiable credentials

- Policy
- Resource
- File
- Database
- Wire
- Relay
- Key strategy

How to...

- Create a Volt
- Start a Volt
- Connect to a Volt
- Upload files
- Download files
- Publish to wire
- Subscribe to wire
- Create a database
- Execute SQL
- Import data
- Establish a connect stream
- Approve authenticate request

Clients

- Command line
- Native / C++
- Web
- NodeJS

Reference

- Best practice
- Battery
- Configuration
- Connection
- Connect stream
- Logging
- PKCS#11

API

- Discovery API
- File API
- SqliteDatabase API
- SqliteServer API
- SSI API
- Sync API
- Relay API
- Volt API
- Wire API

Utilities

- protoDbSync
- sqliteServer
- wireTransform

FAQ

  - Questions

- ## Coming soon
  - Roadmap
- ☀☾

Download file or folder

api

The FileAPI.DownloadFile API enables a file to be downloaded from a resource on the **tdx Volt** to the local disk.

Both the **fusebox** and the **cli** make use of this endpoint to also support downloading folders or entire trees of files.

cli

Use the `download` cli command.

```
./volt download --help
```

Terminal window

Download a file or folder:

```
./volt download ./share/pictures ./shared-images
```

Terminal window

fusebox

Use the ‘download’ button on the folder toolbar.

javascript

grpc

```
const response = await client.DownloadFileSync({
resource_id: "@download-demo",});
console.log(Buffer.from(response.buffer,
"base64").toString());
```

web

Use the DownloadFile API to stream the resource contents, writing each chunk received to memory or local disk.

```
let contents = "";
const rpc = voltApi.DownloadFile({ resource_id: resource.id
});
rpc.on("error", (err) => {  console.log("Error downloading
file: " + err.message);});
rpc.on("data", (payload) => {  contents +=
atob(payload.block || "");});
rpc.on("end", () => {  console.log("download finished");
console.log(contents);});
```

C++

```
auto result = voltApi->downloadFileSync("@download-demo",
"c:/temp/download-demo", cancelCB);
```

Skip to Content

tdx Volt

putting you in charge

☀☾

Toggle sidebar

- ## Getting Started
  - Welcome
  - Quick Start
- ## Concepts
  - Fundamentals
  - Identity
  - DID registry
  - Verifiable credentials
  - Policy
  - Resource
  - File
  - Database
  - Wire
  - Relay
  - Key strategy
- ## How to...
  - Create a Volt
  - Start a Volt
  - Connect to a Volt
  - Upload files
  - Download files
  - Publish to wire
  - Subscribe to wire
  - Create a database
  - Execute SQL
  - Import data
  - Establish a connect stream
  - Approve authenticate request
- ## Clients
  - Command line
  - Native / C++
  - Web
  - NodeJS
- ## Reference
  - Best practice

    - Battery
    - Configuration
    - Connection
    - Connect stream
    - Logging
    - PKCS#11

- ## API

    - Discovery API
    - File API
    - SqliteDatabase API
    - SqliteServer API
    - SSI API
    - Sync API
    - Relay API
    - Volt API
    - Wire API

- ## Utilities

    - protoDbSync
    - sqliteServer
    - wireTransform

- ## FAQ

    - Questions

- ## Coming soon

    - Roadmap

- ☀ ☾

Create database

api

The SqliteServerAPI.CreateDatabase API enables clients to create databases.

cli

Coming soon…

fusebox

Use the ‘New’ button on the folder toolbar.

javascript

Use the CreateDatabase API.

```
examples coming soon...
```

C++

```
examples coming soon...
```

Skip to Content

tdx Volt

putting you in charge

☀ ☾

Toggle sidebar

- ## Getting Started
  - Welcome
  - Quick Start
- ## Concepts
  - Fundamentals
  - Identity
  - DID registry
  - Verifiable credentials
  - Policy
  - Resource
  - File
  - Database
  - Wire
  - Relay
  - Key strategy
- ## How to...
  - Create a Volt
  - Start a Volt
  - Connect to a Volt
  - Upload files
  - Download files
  - Publish to wire
  - Subscribe to wire
  - Create a database
  - Execute SQL
  - Import data
  - Establish a connect stream
  - Approve authenticate request
- ## Clients
  - Command line
  - Native / C++
  - Web
  - NodeJS
- ## Reference
  - Best practice
  - Battery

  - Configuration
  - Connection
  - Connect stream
  - Logging
  - PKCS#11

- **API**
  - Discovery API
  - File API
  - SqliteDatabase API
  - SqliteServer API
  - SSI API
  - Sync API
  - Relay API
  - Volt API
  - Wire API

- **Utilities**
  - protoDbSync
  - sqliteServer
  - wireTransform

- **FAQ**
  - Questions

- **Coming soon**
  - Roadmap
- ☀🌙

Upload file or folder

api

The FileAPI.UploadFile API enables a file to be uploaded from the local disk to a resource on the Volt.

Both the **fusebox** and the **cli** make use of this endpoint to also support uploading folders or entire trees of files.

cli

Use the `upload` cli command:

```
./volt upload --help
```

Terminal window

Upload the local folder `~/Pictures` to the **tdx Volt** `./share/pictures` folder resource.

```
./volt upload ~/Pictures ./share/pictures
```

Terminal window

fusebox

Use the ‘upload’ button on the folder toolbar, or drag and drop files from the OS native file explorer.

javascript

Open the file and read it in chunks, passing each chunk to the UploadFile API.

```
examples coming soon...
```

C++

```
auto result = voltApi->uploadFileSync(resourceId,
sourceFile.toStdString(), cancelCB);
```

Skip to Content

tdx Volt

putting you in charge

☀☾

Toggle sidebar

- ## Getting Started
  - Welcome
  - Quick Start
- ## Concepts
  - Fundamentals
  - Identity
  - DID registry
  - Verifiable credentials
  - Policy
  - Resource
  - File
  - Database
  - Wire
  - Relay
  - Key strategy
- ## How to...
  - Create a Volt
  - Start a Volt
  - Connect to a Volt
  - Upload files
  - Download files
  - Publish to wire
  - Subscribe to wire
  - Create a database
  - Execute SQL
  - Import data
  - Establish a connect stream

  - Approve authenticate request

- **Clients**
  - Command line
  - Native / C++
  - Web
  - NodeJS

- **Reference**
  - Best practice
  - Battery
  - Configuration
  - Connection
  - Connect stream
  - Logging
  - PKCS#11

- **API**
  - Discovery API
  - File API
  - SqliteDatabase API
  - SqliteServer API
  - SSI API
  - Sync API
  - Relay API
  - Volt API
  - Wire API

- **Utilities**
  - protoDbSync
  - sqliteServer
  - wireTransform

- **FAQ**
  - Questions

- **Coming soon**
  - Roadmap
- ☀☾

Protobuf Data Synchronisation

This utility provides a means of synchronising arbitrary protobuf data streams to multiple SQLITE databases.

The basic concept is that the app monitors a configured folder for incoming files.

When a new file arrives, the process inspects the file to determine the types of protobuf message it contains (see file format details below).

The process ingests the file and builds SQL `CREATE TABLE IF NOT EXISTS` and `INSERT` statements from the contents.

The protoDbSync ingestor uses reflection to translate the protobuf definition to the equivalent SQL schema and statements.

It then uses the **tdx Volt** API to run the SQL on any number of configured remote databases. A separate sync status is kept for each target database.

File Format

The file format used is essentially the de-facto standard for protobuf data, which is the length-prefixed binary format. This can easily be generated from most protobuf client library auto-generated stubs, for example `encodeDelimited` in Javascript and `SerializeDelimitedToOStream` in C++.

Note that currently only basic protobuf `message` structures are supported - messages with sub-message types or those that make use of `oneof` or `optional` are not currently supported.

Most of the time you will not need to manually create files in this format, you can use the Volt Logger command to do it for you.

Each file must have a header present which is a serialisation of a `ProtobufSyncConfigurationHeader` message, as defined here. The serialisation of the header **must** include a length prefix.

This header contains a configuration entry (an instance of ProtobufSyncConfiguration) for each message type that will appear in the file.

For example, consider a data producer that is creating two types of message, tcp packets and udp packets, which are defined by the protobuf message types `TcpPacket` and `UdpPacket` respectively. In this scenario the `ProtobufSyncConfigurationHeader` will contain two `ProtobufSyncConfiguration` instances, the first describing `TcpPacket` and the second describing `UdpPacket`.

Each `ProtobufSyncConfiguration` instance contains the actual protobuf that describes the data type. The main fields of the header are `message_proto`, which is the protobuf definition of the target messages (as a string), and `table_name`, which is the name of the table into which the data should be inserted. See ProtobufSyncConfiguration for full details.

As well as the header format described above, each message in the file must be wrapped in a ProtobufSyncWrapper instance and serialised to the file using a length prefix.

Logger integration

The **tdx Volt** CLI has a ‘logger’ command which can automatically create **protoDbSync** compatible files, taking input from STDIN or a wire subscription.

To be clear, the **tdx Volt** Logger will blindly write whatever data is on STDIN, splitting the input into multiple files of a configured maximum size in a configured folder. It makes no attempt to interpret the incoming data.

If a header is given as part of the **tdx Volt** Logger configuration it will populate and write a `ProtobufSyncConfigurationHeader` message to the start of each file it creates.

In order to create files that are compatible with the **protoDbSync** utility, data producers must output data in protobuf binary format as serialised instances of the `ProtobufSyncWrapper` message type.

The idea is to eliminate the need for the producer process to know about the nitty-gritty of the sync file format.

So an arbitrary process can write data (in protobuf binary format for the database sync scenario) to STDIN, and the Volt Logger will create **protoDbSync** compatible files in a configured folder and pipe the data from STDIN into them up to a configured log size.

See the ‘logger’ command documentation for more details.

Configuration

The process can sync to multiple remote databases. The configuration takes the form of a list of target Volt configurations. See the Appendix for a full working example.

The basic structure of the configuration file is as follows:

```
{  "extension": "<file extension>",  "rootFolder": "<folder
to watch>",  "targets": {    "<target-database-id-1>":
{<volt connection object>}    "<target-database-id-2>":
{<volt connection object>}    ...    "<target-database-id-
n>": {<volt connection object>}  }}
```

The configuration is made up of following properties:

- `extension` - optional file extension to match when watching for incoming data (`pdat` is the default)
- `rootFolder` - the folder the process will watch for incoming files.
- `targets` - the list of target database configurations, each entry defines the **tdx Volt** connection details.
- `target-database-id` - this is the id of the target database resource (`12439d48-4c0a-4417-822a-e3db70a9d4d4` in the example below), and contains the details of the **tdx Volt** on which the database is hosted. This is essentially the normal **tdx Volt** client configuration data, optionally with the addition of the `sync` property.
- `sync` - optional configuration for the sync process.
- `sync.fullInitialSync` - the process will send **all** the data it has seen when it first encounters a new database configuration. The default is ‘true’.

The example below shows a configuration file for a single **tdx Volt** target, namely resource `12439d48-4c0a-4417-822a-e3db70a9d4d4` on Volt `Local RPi`:

```
{  "extension": "pdat",  "rootFolder":
"/home/pi/dev/tdxvolt/tdxvolt-core/release/bin/fs20logs",
"targets": {    "12439d48-4c0a-4417-822a-e3db70a9d4d4": {
"client_name": "RPi proto db sync",      "credential": {
"key": "<--REDACTED-->"      },      "volt": {
"ca_pem": "-----BEGIN CERTIFICATE-----
\nMIIDnDCCAoSgAwIBAgIECZZpTjANBgkqhkiG9w0BAQsFADBuMQswCQYDVQC
-----END CERTIFICATE-----\n",        "id": "ab0d5c6e-237a-
45a5-9c27-215dfd6da0b0"      },      "relay_url":
"https://tdxvolt.com",      "sync": {
"fullInitialSync": true      }    }  }}
```

Sync algorithm

Possibly ‘sync’ is a misnomer as currently it is essentially a one-way upload. The main objective is to only send any given data file once.

It does this using a set of sub-folders that are dynamically created and managed off of the `rootFolder` property, an example of which is shown below.

```
.├── fs20logs│   ├── 12439d48-4c0a-4417-822a-e3db70a9d4d4│
│   ├── sync-2021-12-10T06:31:27.724Z.pdat│   │   ├──
archive│   │   │   ├── sync-2021-12-01T14:06:38.421Z.pdat│
│   │   ├── sync-2021-12-01T14:30:30.498Z.pdat│   │   │
└── sync-2021-12-01T18:22:59.301Z.pdat│   │   ├── duplicate│
│   └── error│   ├── archive│   │   ├── sync-2021-12-
01T14:06:38.421Z.pdat│   │   ├── sync-2021-12-
01T14:30:30.498Z.pdat│   │   └── sync-2021-12-
01T18:22:59.301Z.pdat│   ├── sync-2021-12-
10T06:31:27.724Z.pdat│   └── sync-2021-12-
10T06:47:57.196Z.pdat.lock
```

For example, considering the above example configuration file, the following directory structure is used:

- **fs20logs** - the root folder, which is essentially the ‘pending’ folder - incoming data should be placed here
- **12439d48-4c0a-4417-822a-e3db70a9d4d4** - the target-specific ‘pending’ folder, a sub-

folder like this will be created for each target configured. When a new file arrives in the root folder, it is copied into this folder for each target configured.
- **archive** - the files that are successfully processed are moved here. Each target also has an ‘archive’ folder to keep track of the sync status of that target.
- **duplicate** - any new file that already exists in **archive** are skipped and moved here
- **error** - any files that fail to transmit are moved here

The process that is creating the data (producer) should only place a file matching the pattern in the root folder when it is complete and ready for transmission. The example producer achieves this by writing to a `<timestamp>.pdat.lock` file, and then renames the file to remove the `.lock` when it has reached the desired size. If you’re using the Volt logger command it will do this for you.

File size

The particular maximum file size isn’t mandated, but bear in mind that by default each file is transmitted as a single GRPC call in the form of a bulk SQL INSERT statement. There doesn’t seem to be any hard data on this, but it seems the optimal protobuf message size is around 64 kB, although I’ve used 1 MB without any apparent issues.

Appendix

Configuration file example

This example shows a sync to 3 databases on 3 different Volts.

```
{  "extension": "pdat",  "rootFolder":
"/home/pi/dev/tdxvolt/tdxvolt-core/release/bin/fs20logs",
"targets": {    "05847828-799a-4cc5-975b-03f2908d1443": {
"client_name": "protoDbSync",      "credential": {
"cert": "-----BEGIN CERTIFICATE-----
\nMIIDWDCCAkCgAwIBAgIERpkVdzANBgkqhkiG9w0BAQsFADBuMQswCQYDVQ
-----END CERTIFICATE-----\n",        "client_id": "de5e0d80-
2eb8-4e7f-a4d0-d080a20cedf5",        "key": "<--REDACTED-->"
},      "sync": {        "fullInitialSync": true      },
"volt": {        "ca_pem": "-----BEGIN CERTIFICATE-----
\nMIIDnDCCAoSgAwIBAgIEdxBiBjANBgkqhkiG9w0BAQsFADBuMQswCQYDVQ
-----END CERTIFICATE-----\n",        "challenge_code":
"w6uP8Tcg6K2QR905Rms8iXTlksL6OD1KOWBxTK7wxPI=",        "id":
"f8297180-bccd-4c49-b4e5-0cb4e5738d15",        "address":
"192.168.1.178:33325"      }    },    "6fd1f237-681c-4b58-
8d10-39238ac82db2": {      "client_name": "ProtoDbSync",
"credential": {        "cert": "-----BEGIN CERTIFICATE-----
\nMIIDWDCCAkCgAwIBAgIEUNJ8szANBgkqhkiG9w0BAQsFADBuMQswCQYDVQ
-----END CERTIFICATE-----\n",        "client_id": "b5ddca19-
1db3-4aeb-af37-2d05986bbbe9",        "key": "<--REDACTED-->"
},      "sync": {        "fullInitialSync": true      },
"volt": {        "ca_pem": "-----BEGIN CERTIFICATE-----
\nMIIDnDCCAoSgAwIBAgIET8kUKTANBgkqhkiG9w0BAQsFADBuMQswCQYDVQ
-----END CERTIFICATE-----\n",        "challenge_code":
"w6uP8Tcg6K2QR905Rms8iXTlksL6OD1KOWBxTK7wxPI=",        "id":
"6fa2dd6b-02fa-47de-b488-da2d8139ce4c",        "address":
"tdxvolt.com:40725"      }    },    "f3a968df-58c2-44d7-b39c-
97a5fa78ba90": {      "client_name": "ProtoDbSync",
"credential": {        "cert": "-----BEGIN CERTIFICATE-----
\nMIIDWDCCAkCgAwIBAgIEL7VkHjANBgkqhkiG9w0BAQsFADBuMQswCQYDVQ
-----END CERTIFICATE-----\n",        "client_id": "e1afc20b-
2ba7-4103-8f35-26c3cdc51bce",        "key": "<--REDACTED-->"
},      "sync": {        "fullInitialSync": true      },
"volt": {        "ca_pem": "-----BEGIN CERTIFICATE-----
\nMIIDnDCCAoSgAwIBAgIELyHuBTANBgkqhkiG9w0BAQsFADBuMQswCQYDVQ
-----END CERTIFICATE-----\n",        "challenge_code":
"w6uP8Tcg6K2QR905Rms8iXTlksL6OD1KOWBxTK7wxPI=",        "id":
"78263765-730b-4514-8fd9-1c762eb81aa2",        "address":
"192.168.1.69:59969"      }    }  }}
```

Skip to Content

tdx Volt

putting you in charge
☀☾
Toggle sidebar

- ## Getting Started
  - Welcome
  - Quick Start
- ## Concepts
  - Fundamentals
  - Identity
  - DID registry
  - Verifiable credentials
  - Policy
  - Resource
  - File
  - Database
  - Wire
  - Relay
  - Key strategy
- ## How to...
  - Create a Volt
  - Start a Volt
  - Connect to a Volt
  - Upload files
  - Download files
  - Publish to wire
  - Subscribe to wire
  - Create a database
  - Execute SQL
  - Import data
  - Establish a connect stream
  - Approve authenticate request
- ## Clients
  - Command line
  - Native / C++
  - Web
  - NodeJS
- ## Reference
  - Best practice
  - Battery
  - Configuration
  - Connection
  - Connect stream
  - Logging
  - PKCS#11

- ## API
  - Discovery API
  - File API
  - SqliteDatabase API
  - SqliteServer API
  - SSI API
  - Sync API
  - Relay API
  - Volt API
  - Wire API
- ## Utilities
  - protoDbSync
  - sqliteServer
  - wireTransform
- ## FAQ
  - Questions
- ## Coming soon
  - Roadmap
- ☀☾

Stand-alone SQLite server

The **tdx Volt** has built-in support for arbitrary Sqlite databases via the SqliteServerAPI and SqliteDatabaseAPI APIs.

This utility provides the same implementation as the built-in support in a stand-alone executable.

This demonstrates how a 3rd party could provide a database implementation to all **tdx Volt** clients, either in the form of an augmented Sqlite implementation, or perhaps an implementation of a completely different database.

Since the \`sqliteServer\` binary implements the SqliteServerAPI and SqliteDatabaseAPI APIs, it is interchangeable with the built-in support in any scenario.

Usage

The `sqliteServer` connects to the **tdx Volt** like any other client. In order to connect you will need a **tdx Volt** configuration, as outlined here. Assuming you have a configuration in a file called `db.config.json` for example, you can then run the `sqliteServer` using:

```
./sqliteServer --config ./db.config.json
```

Terminal window

When the server first starts, it will create a resource on the **tdx Volt** that represents it's internal grpc server, and registers that as a service with the **tdx Volt**.

Clients can then use DiscoverServices to obtain the details of all running SqliteServers and choose which one to connect to.

In order for the standalone server to be able to create databases in any given folder, it will need \`create in\` permission on that folder, which you can assign via the **fusebox**.

Skip to Content

tdx Volt

putting you in charge

☀☾

Toggle sidebar

- **Getting Started**
  - Welcome
  - Quick Start
- **Concepts**
  - Fundamentals
  - Identity
  - DID registry
  - Verifiable credentials
  - Policy
  - Resource
  - File
  - Database
  - Wire
  - Relay
  - Key strategy
- **How to...**
  - Create a Volt
  - Start a Volt
  - Connect to a Volt
  - Upload files
  - Download files
  - Publish to wire
  - Subscribe to wire
  - Create a database
  - Execute SQL
  - Import data
  - Establish a connect stream
  - Approve authenticate request
- **Clients**
  - Command line
  - Native / C++
  - Web
  - NodeJS
- **Reference**
  - Best practice
  - Battery
  - Configuration
  - Connection
  - Connect stream
  - Logging
  - PKCS#11

- **API**
  - Discovery API
  - File API
  - SqliteDatabase API
  - SqliteServer API
  - SSI API
  - Sync API
  - Relay API
  - Volt API
  - Wire API
- **Utilities**
  - protoDbSync
  - sqliteServer
  - wireTransform
- **FAQ**
  - Questions
- **Coming soon**
  - Roadmap
- ☀☾

Wire Transform

The `wireTransform` utility is a demonstration of a generic wire 'transform' concept, in which the input is taken from a wire subscription, transformed into a different format, and either published onto a different wire, or written to `STDOUT`.

Note that the source and target wires can be located on different Volts and located anywhere on the internet.

Currently the only supported transform is from `tcpdump` text output to a protobuf format matching the definition below, and wrapped in a ProtobufSyncWrapper message suitable for input in to the protoDbSync utility.

```
syntax = "proto3";
package tranforms;
message TCPDumpPacket {  string timestamp = 1;  string
source_mac_address = 2;  string source_manufacturer_id = 3;
bool is_broadcast = 4;  bool is_arp = 5;  string
target_mac_address = 6;  string target_manufacturer_id = 7;
string ether_type = 8;  string unknown_1 = 9;  int32 length
= 10;  string source_address = 11;  string target_address =
12;  string payload = 13;}
```

Usage

Begin by dumping the `tcpdump` command output to a wire:

```
tcpdump <filter> | volt wire -w @tcpdump-text
```

Terminal window

The `wireTransform` utility is then used (potentially on a different machine from that running the `tcpdump` command) to transform the incoming text format `tcpdump` output (on wire `@tcpdump-text`) into a binary protobuf format, and then publish this transformed output onto the `@tcpdump-binary` wire:

```
./wireTransform --source-config=source.config.json -s
@tcpdump-text --target-config=target.config.json -t
@tcpdump-binary
```

Terminal window

If both the source and target wires are on the same Volt you can omit the `--target-config` option.

To write the transformed output to `STDOUT` rather than another wire, simply omit the `-t` switch:

```
./wireTransform --source-config=source.config.json -s
@tcpdump-text
```

Terminal window

The format of the configuration files is a plain Volt client connection format as described here.

protoDbSync integration

The `wireTransform` utility transforms textual `tcpdump` data into a format compatible with the protoDbSync utility.

In order to make the output available to `protoDbSync`, use the volt logger command. In the example below, we run `wireTransform` so that it writes the transformed data to `STDOUT`, and then redirect that output into the `volt logger` command.

```
./wireTransform --source-config=source.config.json -s
@tcpdump-text | ./volt logger -c tcpdump.logger.json
```

The `volt logger` command will generate the appropriate format files required by the `protoDbSync` utility, according to the configuration contained in the `tcpdump.logger.json` file, an example of which is shown below.

```
{  "logger": {    "headerId": "tcpdump-logger",
"headers": [      {        "messageName": "TCPDumpPacket",
"messageProto": "syntax = \"proto3\";\n\npackage
tranforms;\n\nmessage TCPDumpPacket {\n  string timestamp =
1;\n  string source_mac_address = 2;\n  string
source_manufacturer_id = 3;\n  bool is_broadcast = 4;\n
bool is_arp = 5;\n  string target_mac_address = 6;\n  string
target_manufacturer_id = 7;\n  string ether_type = 8;\n
string unknown_1 = 9;\n  int32 length = 10;\n  string
source_address = 11;\n  string target_address = 12;\n
string payload = 13;\n}\n",        "name": "header0",
"tableName": "tcpdump"      }    ],    "logFileExtension":
"pdat",    "logFilePath": "./logs",    "logFilePrefix":
"tcpdump-log-",    "logFileSize": 64000  }}
```

See the protoDbSync utility for more details.

Skip to Content

tdx Volt

putting you in charge
☀☾
Toggle sidebar

- **Getting Started**
  - Welcome

- Quick Start

- ## Concepts

- Fundamentals
- Identity
- DID registry
- Verifiable credentials
- Policy
- Resource
- File
- Database
- Wire
- Relay
- Key strategy

- ## How to...

- Create a Volt
- Start a Volt
- Connect to a Volt
- Upload files
- Download files
- Publish to wire
- Subscribe to wire
- Create a database
- Execute SQL
- Import data
- Establish a connect stream
- Approve authenticate request

- ## Clients

- Command line
- Native / C++
- Web
- NodeJS

- ## Reference

- Best practice
- Battery
- Configuration
- Connection
- Connect stream
- Logging
- PKCS#11

- ## API

- Discovery API
- File API
- SqliteDatabase API
- SqliteServer API
- SSI API
- Sync API
- Relay API
- Volt API
- Wire API

- ## Utilities
  - protoDbSync
  - sqliteServer
  - wireTransform
- ## FAQ
  - Questions
- ## Coming soon
  - Roadmap
- ☀☾

Quick Start

To get started quickly, use the section below that corresponds to your operating system to download the **tdx Volt** installer, then follow the install instructions.

Windows

Click on the link to download the package.

Windows (64bit)

Once downloaded, use the File Explorer to locate the downloaded file named `voltUp.exe`, and then double-click on it and follow the on-screen instructions.

You may be prompted to confirm that you wish to run an application from an unknown publisher, click **run** to continue.

MacOS - M1, M2, M3

Click on the link to download the package.

macOS (Apple Silicon)

Once downloaded, open a terminal window and navigate to the downloaded file.

For example, if you downloaded the file to your `Downloads` folder:

```
cd ~/Downloads
```

Terminal window

It is necessary to change some of the package attributes to allow it to be executed:

```
chmod +x ./voltUp && xattr -d com.apple.quarantine ./voltUp
```

Terminal window

These steps are necessary because the package isn't currently signed with an Apple developer certificate. This will be fixed in an upcoming release.

You can then run the installer directly from the terminal and follow the on-screen instructions:

```
./voltUp
```

Terminal window

MacOS - Intel

Click on the link to download the package.

macOS (Intel)

Once downloaded, open a terminal window and navigate to the downloaded file.

For example, if you downloaded the file to your `Downloads` folder:

```
cd ~/Downloads
```

Terminal window

It is necessary to change some of the package attributes to allow it to be executed:

```
chmod +x ./voltUp && xattr -d com.apple.quarantine ./voltUp
```

Terminal window

These steps are necessary because the package isn't currently signed with an Apple developer certificate. This will be fixed in an upcoming release.

You can then run the installer directly from the terminal and follow the on-screen instructions:

```
./voltUp
```

Terminal window

Ubuntu

Click on the link to download the package.

Ubuntu (20.04 - x86_64)

Ubuntu (24.04 - x86_64)

Ubuntu (23.04 - arm64)

Once downloaded, open a terminal window and navigate to the downloaded file.

For example, if you downloaded the file to your `Downloads` folder:

```
cd ~/Downloads
```

Terminal window

It is necessary to change some of the package attributes to allow it to be executed:

```
chmod +x ./voltUp
```

Terminal window

You can then run the installer directly from the terminal and follow the on-screen instructions:

```
./voltUp
```

Terminal window

Android

Android APK Android

A pre-release version of a **tdx Volt** client application is available for Android. This is a self-contained APK that must be side-loaded onto your device.

This is a pre-release version of the Android client. It is not yet available on the Google Play store.

To install the APK, download it to your device and then open it using the Android package installer. You may need to enable the installation of apps from unknown sources in your device settings.

SDK packages

For headless systems such as routers, Raspberry Pis and cloud servers, it might be desirable to install the platform without using a GUI-based installer. In this case, the SDK is available as a set of packages that can be installed via the command line.

See the links below for the available packages:

- Debian arm64
- MacOS (Apple silicon)
- MacOS (Intel)
- Omnia Turris
- Raspberry Pi buster/32 bit
- Ubuntu 20.04/x86_64
- Ubuntu 24.04/x86_64
- Ubuntu 23.04/arm64
- Windows

With the exception of Windows, the SDK is currently distributed in the form of a self-extracting archive. To install the SDK, navigate to the downloaded file in a terminal and execute it, then follow the prompts. Note you may need to add 'execute' permissions to the downloaded file in order to be able to run it, see the `chmod` command in the example below.

The Windows installation is an NSI package. To install it, download the package and then double-click on the downloaded file and follow the on-screen instructions.

Use the `--help` switch to see the full instructions. For example, assuming the downloaded SDK is called `pi-buster.sh` and is in the folder `~/Downloads`:

```
cd ~/Downloadschmod +x ./pi-buster.sh./pi-buster.sh --help
```

Terminal window

Typically, it makes sense to first create a folder to receive the extracted SDK, and then run the extraction using the `--prefix` option:

```
mkdir voltSDK./pi-buster.sh --prefix=./voltSDK
```

Terminal window

Run

There are two tools that help with managing and interacting with a **tdx Volt**. The most straightforward (and recommended for beginners) option is via the **fusebox**, a graphical user interface (GUI).

The alternative is using the **volt** command line interface (CLI). This is a powerful tool that can operate in both 'client' and 'server' modes. It is useful when peforming tasks at the script level, or for running a **tdx Volt** on a limited resource or headless/embedded device.

Both of the above binaries are located in the installation folder.

It is possible to run a **tdx Volt** on a headless device using the command line (or as a daemon) and connect to it via the **fusebox** from another machine on the network (similar to remote desktop applications).

fusebox

You can launch the **fusebox** from the command line or using the usual operating system file explorer or finder.

Assuming you installed the **tdx Volt** in the `~/Downloads/volt` folder, you can launch the **fusebox** from the command line as follows:

```
cd ~/Downloads/volt./fusebox
```

Terminal window

If you're running Ubuntu 18.04 you may see an error about `libstdc++6` and `GLIBCXX`.

This indicates an updated version of `libstdc++6` is required. You can install this using the command below:

```
sudo add-apt-repository ppa:ubuntu-toolchain-r/test && sudo
apt upgrade libstdc++6
```

Terminal window

When the **fusebox** first starts it will prompt for you to create and confirm a password.

This password can not currently be recovered or reset. Future versions may provide the ability to pre-rotate your key.

After creating the password, you can go ahead and create your first **tdx Volt** by clicking on the big **create Volt** button:

Enter a name for the Volt, ignore the other options and click **create** (you may need to scroll down to see the **create** button):

You have now created your first **tdx Volt** - click on the **tdx Volt** in the left column to navigate to the **tdx Volt** browser.

Command line

You can also create and start volts using the command line tool.

Volts created and run via the command line are fully functional in the same way as those created via the **fusebox**. However there are currently limited management options exposed by the command line interface. The options available will be increased in the near future.

show CLI `create` options

Assuming you installed the **tdx Volt** in the `~/Downloads/volt` folder, you can view the available options for the `create` command as follows:

```
cd ~/Downloads/volt./volt create --help
```

Terminal window

create a Volt

To create a **tdx Volt** from the command line, it's as simple as:

```
./volt create "Alice's MacBook"
```

Terminal window

It's advisable to specify an alias when creating a **tdx Volt** from the command line. This will make it easier to refer to the **tdx Volt** in future commands.

```
./volt create "Alice's MacBook" --alias alice
```

Terminal window

start a Volt

You can then use the alias to start the **tdx Volt**:

```
./volt run @alice
```

Terminal window

list Volts

You can list the available **tdx Volts** on the system using the `config` command:

```
./volt config
```

Terminal window

See the CLI documentation for more details of the command line interface.

Next steps

You now have a **tdx Volt** running on your machine. You can use the **fusebox** to manage the **tdx Volt** and create additional **tdx Volts**.

Learn about uploading files to your **tdx Volt** in the uploading files guide.

Skip to Content

tdx Volt

putting you in charge
☀☾
Toggle sidebar

- ## Getting Started
  - Welcome
  - Quick Start
- ## Concepts
  - Fundamentals
  - Identity
  - DID registry
  - Verifiable credentials
  - Policy
  - Resource
  - File
  - Database
  - Wire
  - Relay
  - Key strategy
- ## How to...
  - Create a Volt
  - Start a Volt
  - Connect to a Volt
  - Upload files
  - Download files
  - Publish to wire
  - Subscribe to wire
  - Create a database
  - Execute SQL
  - Import data
  - Establish a connect stream
  - Approve authenticate request

- **Clients**
  - Command line
  - Native / C++
  - Web
  - NodeJS
- **Reference**
  - Best practice
  - Battery
  - Configuration
  - Connection
  - Connect stream
  - Logging
  - PKCS#11
- **API**
  - Discovery API
  - File API
  - SqliteDatabase API
  - SqliteServer API
  - SSI API
  - Sync API
  - Relay API
  - Volt API
  - Wire API
- **Utilities**
  - protoDbSync
  - sqliteServer
  - wireTransform
- **FAQ**
  - Questions
- **Coming soon**
  - Roadmap
- ☀☾

Wire

A Wire is a type of resource that enables clients to publish and subscribe to opaque streams of data.

Authentication and authorisation

Clients wishing to publish or subscribe to a wire must authenticate with the **tdx Volt** in the normal way.

In order to publish to a wire, the authenticated identity will require `tdx:resource-publish` permission, subscribing to a wire requires the `tdx:resource-subscribe` permission. These permissions can be set from the **fusebox** ‘share’ panel.

Format and Transforms

By default the wire stream is raw binary. It is anticipated that clients will use the resource 'kind' field to indicate the format of data that is published on any given wire. For example, a wire that streams an FS20 feed might indicate this by using a resource 'kind' of `tdx:wire fs20:feed`.

Some scenarios may require transformation of wire data, for example conversion of text data (e.g. `tcpdump` output) into a serialised protobuf serialisation format.

In the following scenario, output of a `tcpdump` filter is being piped (published) into a wire. This process is running on the network router.

```
tcpdump <filter> | ./volt wire -w <tcpdump-id>
```

On a different machine another process is subscribed to the `tcpdump-id` wire and transforms the text data it contains into a raw protobuf data stream, and re-publishes it onto another wire:

```
./volt wire -w <tcpdump-id> -s | tcpdump-transform | ./volt
wire -w <transformed-wire-id>
```

On a third, or multiple other machines, a process subscribes to the transformed wire and pushes it into a database synchronisation file cache:

```
./volt wire -w <transformed-wire-id> -s | ./volt logger -c
tcpdump.logger.json
```

See the protoDbSync utility for a more detailed analysis of this approach.

Skip to Content

tdx Volt

putting you in charge
☀ ☾
Toggle sidebar

- **Getting Started**
    - Welcome
    - Quick Start
- **Concepts**
    - Fundamentals
    - Identity
    - DID registry
    - Verifiable credentials
    - Policy
    - Resource
    - File
    - Database
    - Wire
    - Relay
    - Key strategy
- **How to...**
    - Create a Volt
    - Start a Volt

  - Connect to a Volt
  - Upload files
  - Download files
  - Publish to wire
  - Subscribe to wire
  - Create a database
  - Execute SQL
  - Import data
  - Establish a connect stream
  - Approve authenticate request

- **Clients**

  - Command line
  - Native / C++
  - Web
  - NodeJS

- **Reference**

  - Best practice
  - Battery
  - Configuration
  - Connection
  - Connect stream
  - Logging
  - PKCS#11

- **API**

  - Discovery API
  - File API
  - SqliteDatabase API
  - SqliteServer API
  - SSI API
  - Sync API
  - Relay API
  - Volt API
  - Wire API

- **Utilities**

  - protoDbSync
  - sqliteServer
  - wireTransform

- **FAQ**

  - Questions

- **Coming soon**

  - Roadmap
- ☀☾

Verifiable credentials

The **tdx Volt** supports the W3C Verifiable Credentials data model for issuing, presenting, and verifying credentials. Verifiable credentials are a way to express and exchange credentials in a way that is cryptographically secure and privacy-preserving.

What is a verifiable credential?

A verifiable credential is a tamper-evident credential that has cryptographic proof of its authenticity. It is a digital representation of a credential that can be used to prove claims about a subject. Verifiable credentials are issued by an issuer to a holder, and can be presented to a verifier to prove a claim.

A verifiable credential consists of three main parts:

- **Issuer**: The entity that issues the credential.
- **Holder**: The entity that holds the credential.
- **Subject**: The entity that the credential is about.

Within the **tdx Volt** architecture, the **Issuer** is typically (although not limited to) a **tdx Volt** instance, the **Holder** is the **tdx Volt** instance or identity that holds the credential, and the **Subject** is the identity that the credential is about.

Usage within the tdx Volt

The **tdx Volt** uses verifiable credentials to represent claims about the identity of the holder. These claims can be used to prove the identity of the holder to other entities in the **tdx Volt** ecosystem, and to define rules and policies that govern the access permissions and behavior of the holder to the various resources and services within the **tdx Volt**.

The identities and claims within the verifiable credential leverage the Decentralized Identifiers (DIDs) and DID Registries to provide a secure and decentralized way to manage and verify the identity of the holder.

Example credential

Below is an example of a Verifiable Credential, issued by the DVLA, that contains a claim that the subject is over 18:

```
{  "@context": [
"https://www.w3.org/2018/credentials/v1",
"https://tdxvolt.com/credentials/v1"  ],
"credentialSubject": {    "id": "did:volt:0847915a-6bdf-
478c-9d82-f96df58c7856",    "isOver18": true  },  "id":
"http://192.168.1.195:62597/credential/c590d690-e665-428e-
8cbc-d47a1b405f06",  "issuanceDate": "2024-05-30T16:12:43",
"issuer": {    "id": "did:volt:0847915a-6bdf-478c-9d82-
f96df58c7856",    "name": "DVLA"  },  "proof": {
"created": "2024-05-30T15:12:43Z",    "jws":
"eyJhbGciOiJFZERTQSIsImI2NCI6IGZhbHNlfQ..JeByn3x1XpcQ3OmookNy
-
V4v079xpOoqBE8Nkf0298vRmYGstMvqFyAR661k46SgBMsWDGXZJcxvCvW9bL
,    "proofPurpose": "assertionMethod",    "type":
"Ed25519Signature2018",    "verificationMethod":
"did:volt:0847915a-6bdf-478c-9d82-f96df58c7856#key-1"  },
"type": ["VerifiableCredential", "AgeVerification"]}
```

Let’s break down the parts of the Verifiable Credential shown above:

- **`@context`**

  The context in which the credential is issued. This is a list of URIs that define the terms used in the credential.

- **`credentialSubject`**

  The subject of the credential, along with the claims made about the subject.

In this case the subject is the DID `did:volt:0847915a-6bdf-478c-9d82-f96df58c7856`, and the credential contains a claim that the subject is over 18.

- **`id`**

  The identifier of the credential. This is a URI that uniquely identifies the credential.

- **`issuanceDate`**

  The date and time the credential was issued.

- **`issuer`**

  The entity that issued the credential. This includes the ID of the issuer and the name of the issuer.

  In this case the issuer is the DID `did:volt:0847915a-6bdf-478c-9d82-f96df58c7856`, and the name of the issuer is `DVLA`.

- **`proof`**

  The cryptographic proof of the authenticity of the credential.

- **`type`**

  The type of the credential. This is a list of URIs that define the type of the credential.

Verifying a credential

In order for a verifier to trust the authenticity of the credential, a verifier must be able to verify the cryptographic proof provided in the `proof` section. This proof is generated using the private key of the issuer, and can be verified using the public key of the issuer.

Within the **tdx Volt** ecosystem, peers and entities make use of the DID Registry to resolve the DIDs of the subject and issuer, and to acquire their public keys in order to verify the proof and securely communicate.

Policy-based access control

Policy rules within the **tdx Volt** can be created to allow or deny access to a resource based on the presence of a verifiable credential that meets certain criteria.

For example, a policy rule could be created to allow access to a resource only if the holder of the credential is over 18, and the credential is issued by a trusted issuer such as the DVLA. This policy rule would be evaluated at runtime, and the access decision would be made based on the verifiable credential presented by the holder.

Non-DID subjects

Verifiable Credential subjects do not necessarily have to reference a DID. For example, the following Verifiable Credential is issued by `coreid.com` and contains an email address and public key as the subject:

```
{  "@context": [
"https://www.w3.org/2018/credentials/v1",
"https://tdxvolt.com/credentials/v1"  ],
"credentialSubject": {    "email": "alice@example.com",
"id": "alice@example.com",    "keyId":
"BG4EZ1fgP7FoEyRycniTZKPneddSLxv8wkhvHv5nXcdo",
"publicKey": "-----BEGIN PUBLIC KEY-----
\nMCowBQYDK2VwAyEAotWPJoA/nQ6gaIbsu77u5vQ/auhV717aEN1o/0ZJoic
-----END PUBLIC KEY-----\n"  },  "id":
"https://coreid.com/credential/d662fbfa-986d-4505-9367-
6e2c10b01809",  "issuanceDate": "2024-05-21T07:00:50",
"issuer": {    "id": "did:volt:bed919ab-6081-40e7-9677-
88d1cd37a0c0",    "name": "coreid.com"  },  "proof": {
"created": "2024-05-21T06:00:50Z",    "jws":
"eyJhbGciOiJFZERTQSIsImI2NCI6IGZhbHNlfQ..TWCHI2_LG8xYbx6-
EYBEnBLf2D_6pdFowzH2fdxuaw2z3Im8gOTv3hL3l-
Urz3_3rZt5hG_iMB7Hjzy-R-4ECA",    "proofPurpose":
"assertionMethod",    "type": "Ed25519Signature2018",
"verificationMethod": "did:volt:bed919ab-6081-40e7-9677-
88d1cd37a0c0#key-1"  },  "type": [
"VerifiableCredential",    "VoltIdentityCredential",
"VoltEmailCredential"  ]}
```

The credential was issued by https://coreid.com after verifying that the holder of the public key also has control over the email address `alice@example.com`.

This credential could be used to prove the identity of the holder in a variety of scenarios, such as authenticating to a service or a web application.

The flow would involve the browser or client creating a key pair using the web crypto API, and then presenting the public key and email address to the `coreid.com` identity provider service, along with a signature. The service would then send an email to the address containing a verification link, which the holder would then click and return to the service. The service would then issue the credential to the holder.

Skip to Content

tdx Volt

putting you in charge

☀☾

Toggle sidebar

- ## Getting Started
  - Welcome
  - Quick Start

- ## Concepts
  - Fundamentals
  - Identity
  - DID registry
  - Verifiable credentials
  - Policy
  - Resource
  - File
  - Database
  - Wire
  - Relay
  - Key strategy

- ## How to...
  - Create a Volt
  - Start a Volt
  - Connect to a Volt
  - Upload files
  - Download files
  - Publish to wire
  - Subscribe to wire
  - Create a database
  - Execute SQL
  - Import data
  - Establish a connect stream
  - Approve authenticate request
- ## Clients
  - Command line
  - Native / C++
  - Web
  - NodeJS
- ## Reference
  - Best practice
  - Battery
  - Configuration
  - Connection
  - Connect stream
  - Logging
  - PKCS#11
- ## API
  - Discovery API
  - File API
  - SqliteDatabase API
  - SqliteServer API
  - SSI API
  - Sync API
  - Relay API
  - Volt API
  - Wire API
- ## Utilities
  - protoDbSync
  - sqliteServer
  - wireTransform
- ## FAQ
  - Questions
- ## Coming soon
  - Roadmap
- ☀☾

Database

The **tdx Volt** offers built-in support for Sqlite database resources. This enables the creation of an arbitrary number of databases, each governed by the **tdx Volt** policy.

Encryption

As with the **tdx Volt** metadata store, database resources support encryption at rest. The key is derived from the **tdx Volt** root key, but can be configured.

Policy

The policy can be applied to database resources as either read or write permission to the entire database. Read permission enforces that only `SELECT` statements can be executed on the database, whereas write permission permits execution of any SQL statement.

Audit

Database resources can be configured to audit reads and/or writes.

Locks

Sqlite does not support multi-threaded/mult-process access very well, however the **tdx Volt** database API implementation essentially acts as a gatekeeper of the underlying database and as such is able to marshal access.

The implementation utilises the write-ahead logging mode of Sqlite, which supports unlimited concurrent 'read' clients along with a single 'write' client. The **tdx Volt** SQLite grpc server exposes this functionality through a simple protobuf interface.

In summary:

- The **tdx Volt** SQLite server supports multiple clients executing `SELECT` statements concurrently.
- A `SELECT` statement will never block, even if a write statement is executing.
- A single client can successfully execute any statement other than `SELECT`, (e.g. `INSERT`, `UPDATE`, `DELETE` etc).
- A non-`SELECT` statement will succeeed even if there are in-flight `SELECT` statements running for other clients.
- If two or more clients attempt to write concurrently, the server will block the clients until the write-lock is free and then complete each pending statement. This is an improvement over the standard SQLite interface, which will create an `SQL_BUSY` error in this scenario.

Stand-alone server

The **tdx Volt** core has built-in support for SQLite databases as described above. There is also a stand-alone version of the server available as a utility. This enables configurations whereby the database server is running on a different machine from the **tdx Volt** itself.

The **tdx Volt** offers clients the ability to register services for consumption by other clients and it is anticipated that support for other types of databases will be gradually increased as the need arises, both as built-in services and stand-alone servers.

☀☾
Toggle sidebar

- ## Getting Started
  - Welcome
  - Quick Start
- ## Concepts
  - Fundamentals
  - Identity
  - DID registry
  - Verifiable credentials
  - Policy
  - Resource
  - File
  - Database
  - Wire
  - Relay
  - Key strategy
- ## How to...
  - Create a Volt
  - Start a Volt
  - Connect to a Volt
  - Upload files
  - Download files
  - Publish to wire
  - Subscribe to wire
  - Create a database
  - Execute SQL
  - Import data
  - Establish a connect stream
  - Approve authenticate request
- ## Clients
  - Command line
  - Native / C++
  - Web
  - NodeJS
- ## Reference
  - Best practice
  - Battery
  - Configuration
  - Connection
  - Connect stream
  - Logging
  - PKCS#11
- ## API
  - Discovery API
  - File API
  - SqliteDatabase API

  - SqliteServer API
  - SSI API
  - Sync API
  - Relay API
  - Volt API
  - Wire API

- **Utilities**
  - protoDbSync
  - sqliteServer
  - wireTransform

- **FAQ**
  - Questions

- **Coming soon**
  - Roadmap
- ☀☾

Resource

One of the main functions of the **tdx Volt** is the storage of resource metadata.

Almost all the entities exposed by the various **tdx Volt** APIs are represented internally by a resource in the metadata store, including identities, files, folders, databases, wires, 3rd party services and so on.

The metadata store is implemented as an SqlCipher database, a secure, encrypted-at-rest version of Sqlite.

Full details of the resource schema can be found in the protobuf definition

Kinds

Resources are classified using a simple text-based, free-form taxonomy, represented by the `kind` property.

This is implemented as an ordered list of strings, where each entry in the list describes a ‘kind’.

There is a limited set of reserved kinds for use by the system.

All resources must have a top-level ‘kind’ that matches one of the reserved system ‘kinds’.

Most of the reserved ‘kinds’ are self-explanatory. They are listed in the table below.

Kind	Description
tdx:cloud-connection	Represents a connection to a cloud-based tunnel.
tdx:database	A database.
tdx:file	A file.
tdx:folder	A folder.
tdx:group	A group of identities.
tdx:home-folder	The home folder of a given identity.
tdx:http-proxy	An HTTP forward proxy resource.
tdx:http-server	An HTTP server resource.
tdx:identity	An identity resource.
tdx:service	The top-level service kind.
tdx:sqlite-database	A sub-kind of Database, representing an Sqlite database.
tdx:sqlite-server	SqliteServer
tdx:symbolic-link	A file or folder that is linked directly to the local file system.
tdx:volt-link	Used by the **fusebox** to store links to other Volts
tdx:web-view	Used by the **fusebox** to display web pages. Experimental, macOS only.
tdx:wire	A wire resource.

A resource can reflect multiple 'kinds', for example:

- An SQLite database is represented by `tdx:database, tdx:sqlite-database`.
- When a new identity is created in a Volt, the underlying resource is classified using the 'kinds' `tdx:identity, tdx:folder, tdx:home-folder`.

Clients can augment the default resource 'kind' with custom kinds that suit their domain, but they must not use the `tdx:` prefix. For example `tdx:wire, fs20:feed`.

It is anticipated that the `kind` property of a resource will often reflect a 'type' hierarchy along the lines of generic -> specific, but this isn't enforced in any way and there will be use cases where this is not applicable.

Service description

The service description of a resource describes a grpc server, and is only applicable to resources that expose a grpc server.

A single grpc server can expose multiple services, where each service is described by the protobuf definition language.

All resources that expose a grpc server have the `tdx:service` kind.

Within the Volt, each resource is hosted by one and only one grpc server.

All built-in resources are hosted by the **tdx Volt** itself, which exposes a single grpc server that implements the following APIs: `tdx.volt_api.volt.v1.VoltAPI`, `tdx.volt_api.volt.v1.FileAPI`, `tdx.volt_api.volt.v1.WireAPI`, `tdx.volt_api.data.v1.SqliteServerAPI`, `tdx.volt_api.data.v1.SqliteDatabaseAPI`.

If a client registers a service with the Volt, the details of where the service is running and the APIs it supports are supplied by the client as part of the registration process.

The `service_api` field within the resource can be useful to discover services that expose a given interface. For example, the DiscoverServices API can be used to find all servers that expose the `tdx.volt_api.data.v1.SqliteDatabaseAPI` service.

To better illustrate this concept, consider the following:

- the **tdx Volt** exposes the `tdx.volt_api.data.v1.SqliteServerAPI`, among others.
- when a database is created via the **fusebox**, a 'use **tdx Volt** database server' checkbox can be used to indicate if the database should be hosted by the **tdx Volt** or if the **fusebox** should try and discover other services that support the

`tdx.volt_api.data.v1.SqliteServerAPI`.
- selecting 'use **tdx Volt** database server' means that any operations on that database are handled by the **tdx Volt** grpc server
- however, the SQLite Server utility also exposes `tdx.volt_api.data.v1.SqliteServerAPI`
- when the Sqlite Server utility connects to the **tdx Volt** it registers it's server with the Volt
- if a database is then created with 'use **tdx Volt** database server' **not** selected, the **fusebox** will ask the Sqlite Server utility to create the database
- any operations on that database are handled by the utility and **not** the Volt. If the Sqlite Server utility is not online then it will not be possible to interact with that database resource.

Attributes

The resource schema is fixed as described by the protobuf definition.

However, clients can use the `attributes` field of the resource to associate an arbitrary number of ResourceAttributes with any given resource.

A resource attribute is an advanced form of name-value pair. Each attribute is identified by a unique identifier and is of a specified data type. A single attribute instance can take multiple values.

For example, http proxy forwarder resources will have the following attributes attached when they are created via the **fusebox**:

Attribute Id	Data Type	Description	Example
tdx:http-proxy-domain	string	The sub-domain to proxy on.	"docs"
tdx:http-proxy-host	string	The host to proxy to.	"localhost"
tdx:http-proxy-port	integer	The port to proxy to.	3000

Clients can add attributes using identifiers that match their domain.

Attribute identifiers with the `tdx:` prefix are reserved.

The GetResources API can be used to find resources based on the associated attributes.

Resource attributes are a good fit for the **tdx Volt** policy engine, which uses attribute-based rules rather than the traditional role-based approach. This means that policy rules can be defined for arbitrary attributes.

Ownership

Each resource is assigned an owner at creation time, which by default is the currently authenticated identity that issued the command.

The resource owner can perform any operation on the resource.

By default the **tdx Volt** policy states that the **Volt owner** can also perform any operation on **any** resource, irrespective of the owner.

This differs from the standard TDX, which has no concept of an over-arching owner, and by default resources are only accessible to their owners. The **tdx Volt** operates in a peer-to-peer model. If Alice grants Bob permission to use her Volt, and Bob uploads some data to Alice's Volt, he does so in the knowledge that Alice has full access to that data. If Bob wanted to restrict the access Alice has to his data, he should create it on his own **tdx Volt** and invite Alice to connect, with the appropriate policy rules in place.

Hierarchy

Resources can be organised hierarchically.

All resources are descended from the **tdx Volt** root resource.

When a client creates a resource, they indicate the parent resource that should contain the new resource. If a client does not specify a parent resource, the authenticated identity's Home folder will be used.

Skip to Content

tdx Volt

putting you in charge

☀☾

Toggle sidebar

- ## Getting Started
  - Welcome
  - Quick Start
- ## Concepts
  - Fundamentals
  - Identity
  - DID registry
  - Verifiable credentials
  - Policy
  - Resource
  - File
  - Database
  - Wire
  - Relay
  - Key strategy
- ## How to...
  - Create a Volt
  - Start a Volt
  - Connect to a Volt
  - Upload files
  - Download files
  - Publish to wire
  - Subscribe to wire
  - Create a database
  - Execute SQL
  - Import data
  - Establish a connect stream
  - Approve authenticate request
- ## Clients
  - Command line
  - Native / C++
  - Web
  - NodeJS
- ## Reference
  - Best practice
  - Battery
  - Configuration
  - Connection

  - Connect stream
  - Logging
  - PKCS#11

- ## API

  - Discovery API
  - File API
  - SqliteDatabase API
  - SqliteServer API
  - SSI API
  - Sync API
  - Relay API
  - Volt API
  - Wire API

- ## Utilities

  - protoDbSync
  - sqliteServer
  - wireTransform

- ## FAQ

  - Questions

- ## Coming soon

  - Roadmap
- ☀☾

Fundamentals

The **tdx Volt** is comprised of 4 central facets:

- security policy
- identity management
- resource management (service, file and data sharing)
- service registration and discovery

Security policy

The security policy is attribute-based and borrows heavily from the XACML standard in terms of functionality.

It provides an extremely versatile and extensible framework for controlling who can access what and when.

For example it is possible to express rules such as ‘Nick can access my geolocation service and view my current location between 9am - 5pm Monday to Friday’.

Learn more about the security policy .

Identity Management

Identities are centred around asymetrical cryptography in the form of public/private key pairs. They form the bedrock of the **tdx Volt** infrastructure. Security policy rules are expressed in terms of permitting or denying resource access to one or more identities.

Learn more about **tdx Volt** identities.

Resource Management

A resource is the fundamental entity in the **tdx Volt**. Various kinds of resource exist out of the box, including services, folders, files, and databases. Custom resource types are also supported.

A simple, clean hierarchical taxonomy is used to classify resources.

A single file or entire folder hierarchies can easily be uploaded or linked to the **tdx Volt** and made available from anywhere, given the correct authentication and authorisation as determined by the security policy.

Databases can quickly be created and data ingested using a drag-and-drop interface. This too can then be made available for reading or writing anywhere using the appropriate security policy rules.

SQL databases currently have built-in support in **tdx Volt**, but other flavours will be coming soon. The **tdx Volt** is not opinionated about the database system or format - it just sees databases as another service.

Learn more about **tdx Volt** resources.

Service Registration and Discovery

One of the main functions of the **tdx Volt** is to allow services to be registered, shared, discovered and accessed by others.

Once an identity has successfully bound and connected to the **tdx Volt** it can register a service for others to discover, and/or discover and utilise services that others have registered. This is all strictly governed by the security policy imposed by the **tdx Volt** owner.

Skip to Content

tdx Volt

putting you in charge
☀☾
Toggle sidebar

- ## Getting Started

  - Welcome
  - Quick Start

- ## Concepts

  - Fundamentals
  - Identity
  - DID registry
  - Verifiable credentials
  - Policy
  - Resource
  - File
  - Database
  - Wire
  - Relay
  - Key strategy

How to...

- Create a Volt
- Start a Volt
- Connect to a Volt
- Upload files
- Download files
- Publish to wire
- Subscribe to wire
- Create a database
- Execute SQL
- Import data
- Establish a connect stream
- Approve authenticate request

Clients

- Command line
- Native / C++
- Web
- NodeJS

Reference

- Best practice
- Battery
- Configuration
- Connection
- Connect stream
- Logging
- PKCS#11

API

- Discovery API
- File API
- SqliteDatabase API
- SqliteServer API
- SSI API
- Sync API
- Relay API
- Volt API
- Wire API

Utilities

- protoDbSync
- sqliteServer
- wireTransform

FAQ

- Questions

Coming soon

- Roadmap

- ☀🌙

Identity

The following explores in depth how identities are represented in **tdx Volt** and describes the interactions with other elements in the architecture.

The **tdx Volt** identity model is based on the principles of self-sovereign identity (SSI), which is a model for managing digital identities in a way that is secure, private, and decentralized.

Public key cryptography

At the heart of the identity management (and **tdx Volt** itself) is public key cryptography.

All identities known to a **tdx Volt** are associated with one or more public/private key pairs. By default, the **tdx Volt** never stores or even sees the private portion of the key, which must be kept private by the identity. The **tdx Volt** has a copy of the public key(s).

In order to perform any interaction with **tdx Volt**, an identity must prove that it is in possesion of the private key corresponding to one of the public keys the **tdx Volt** is holding for that identity.

This proof is usually achieved by the identity signing some message or claim with its private key, which **tdx Volt** can then verify using the public portion of the key. Another method is via an x509 certificate-backed TLS connection, which includes the key verification as part of the handshake.

The **tdx Volt** uses the Edwards-curve Digital Signature Algorithm (EdDSA) by default. This is a modern, secure, and efficient algorithm that is well-suited to the requirements of the **tdx Volt**.

Decentralized Identifiers (DIDs)

Internally, an identity is represented by an immutable unique identifier rather than its public key. This is known as a ‘decentralised identifier’ or DID. This level of indirection ensures that an identity can easily periodically change keys as a security precaution or in case of compromise. This also allows for an identity to utilise more than one key pair which might be useful in certain scenarios to limit exposure.

A decentralized identfier, or DID, is a new type of identifier that enables verifiable, self-sovereign digital identity. DIDs are fully under the control of the DID subject, independent of any centralized registry, identity provider, or certificate authority. DIDs are URIs that relate a DID subject to means for trustable interactions with that subject.

For more information, see the W3C Decentralized Identifiers (DIDs) specification.

Each **tdx Volt** maintains a database or registry of DIDs and their associated documents, which in turn contains their public key(s). This registry is used to look up the public key of an identity when it is presented to the **tdx Volt**. See the DID Registry section for more information.

Authentication

The core authentication mechanism is based on standard public key infrastructure (PKI) technology.

The **tdx Volt** acts as a certificate authority (CA), issuing verifiable credentials or certificates to clients that request access.

Any client wanting to access resources or services on the **tdx Volt** must present a credential that has been issued by the **tdx Volt** CA, or by some other **tdx Volt** that is trusted.

Each **tdx Volt** maintains a list of public keys that it knows about, and each public key has a mapping to a unique identity within the **tdx Volt**. This is known as a DID Registry.

There are varying degrees of trust implemented, for example a **tdx Volt** may be configured to trust a public key to connect to and use services, or it may trust a public key to sign additional public keys for access to the Volt. The latter is achieved in combination with Verifiable Credentials.

This provides a flexible and scalable mechanism of establishing trust, whereby a **tdx Volt** can decide, on a case-by-case basis, to trust any client that presents a credential signed by its own key or that of another trusted identity.

The following notes relate to the core **tdx Volt** management interface as well as any service registered with the **tdx Volt** by other applications/clients.

The Authenticate step

For the purposes of this discussion, a client is, for example, an application that wants to access some resource or service.

When a client first starts it must obtain a credential from the **tdx Volt** in order to be able to connect to any service or resource. It does this by sending an authentication request to the Volt.

The authentication request includes the public key of the client, and a signature. Upon receipt of this request, the **tdx Volt** can infer that the client is in possession of the corresponding private key.

When submitting an authentication request, a client may also supply one or more additional certificates or Verifiable Credentials that can be used to help the Volt decide whether or not to issue a credential. For example, a Samsung device may submit a certificate that chains to the Samsung CA. If the **tdx Volt** has been configured to permit the Samsung CA to authenticate clients, it might automatically issue the credential.

Once in possession of a credential, the client can connect to the Volt. As part of the default TLS mechanism, the client provides one or more root certificates that it trusts. If the **tdx Volt** server does not present a certificate that chains to one of these root certificates the client will abort the call.

For example, Bob (the client) wants to access a resource on Alice’s **tdx Volt**.

- Alice’s **tdx Volt** starts its grpc service using a certificate issued by her **tdx Volt** CA
- Bob issues a call to Alice’s **tdx Volt** service. As part of the call Bob mandates that the server must present a credential issued by the **tdx Volt** CA. Bob also sends his own credential to the server, which has been issued by Alice’s **tdx Volt** CA.
- Alice’s **tdx Volt** receives Bob’s request and verifies that a credential was presented, and that it chains to the **tdx Volt** CA.
- Alice’s **tdx Volt** extracts the public key from Bob’s credential and looks it up in the DID Registry. If no matching identity is found (or the identity has been revoked) the call is rejected.
- Alice’s **tdxVolt** can now provide this identifier to the policy engine, which will determine if Bob has permission to access the resource or service he has requested.

Token authentication

Some grpc platforms do not support client certificate inspection from server implementations. In these scenarios, the client presents a certificate and grpc will enforce and verify the certificate requirement, but the actual service implementation has no visibility of the client certificate that was presented.

This makes it impossible for the service to identify the client without further information provided with the call.

This isn’t a problem for the **tdx Volt** itself which is implemented in C++ and can access the client certificate. However grpc is platform agnostic and third-party services that are registered with the **tdx Volt** may be implemented in any language. These third-party services need to be able to identify the client in order to determine if they have permission to access the service, among other things.

To address this the **tdx Volt** supports additional authentication by way of a JSON Web Token (**JWT**) that is sent as part of the grpc call metadata, which is available on all platforms.

The JWT contains a claim that identifies the client, which is the resource id that was assigned to the client at the authentication stage. The JWT is then signed by the client private key.

Upon receipt of the token, a server can decode it to extract the resource id and then verify the signature using the client public key, which will be stored alongside the resource id in the **tdx Volt** DID Registry.

This is conceptually very similar to an x509 certificate in the sense that it is binding a public key to some identifier. The key difference is that the use of a client certificate (as used, for example, in TLS) requires the client to be in possession of the private key, whereas a token does not. Tokens are typically obtained through some authentication step that requires the private key or similar verification, and is then used subsequently for repeated calls to an API. It is therefore possible that if a token is misplaced or stolen it can be abused.

There are ways of limiting the vulnerability of tokens such as adding additional claims for expiry and such. In some situations a token is preferable to full- blown PKI, such as when securely storing a private key is impossible or impractical. As such, **tdx Volt** services can support a combination of authentication mechanisms tailored to suit their individual requirements.

Client Authentication	Server Authentication	Supported	Notes
certificate	certificate	yes	This is the default configuration in which mutual certificate authentication is used
certificate + token	certificate	yes	The client provides both a certificate and JWT.
token	certificate	yes	The client only presents a JWT with no client certificate. This is useful in scenarios where storing a private key is impractical
none	certificate	no	The client must always provide some authentication
certificate	none	no	The server must always present a certificate.
token	none	no	The server must always present a certificate.
none	none	no	Insecure

Skip to Content

tdx Volt

putting you in charge

☀☾

Toggle sidebar

- **Getting Started**

  - Welcome
  - Quick Start

- **Concepts**

  - Fundamentals
  - Identity
  - DID registry
  - Verifiable credentials
  - Policy
  - Resource
  - File
  - Database

- Wire
- Relay
- Key strategy

How to...

- Create a Volt
- Start a Volt
- Connect to a Volt
- Upload files
- Download files
- Publish to wire
- Subscribe to wire
- Create a database
- Execute SQL
- Import data
- Establish a connect stream
- Approve authenticate request

Clients

- Command line
- Native / C++
- Web
- NodeJS

Reference

- Best practice
- Battery
- Configuration
- Connection
- Connect stream
- Logging
- PKCS#11

API

- Discovery API
- File API
- SqliteDatabase API
- SqliteServer API
- SSI API
- Sync API
- Relay API
- Volt API
- Wire API

Utilities

- protoDbSync
- sqliteServer
- wireTransform

FAQ

- Questions

- **Coming soon**
  - Roadmap
- ☀☾

Key strategy

Various strategies are available for securing the **tdx Volt** root key.

Hardware

This strategy enables the use of a PKCS#11-compliant hardware security module (HSM) to secure the **tdx Volt** key. The encryption key is never exposed outside the HSM, providing a high level of security.

For more information about configuring the **tdx Volt** to use a PKCS#11 HSM, see the PKCS#11 reference.

Battery

In this key strategy, the **tdx Volt** key is encrypted using AES-256 in CBC mode with the passphrase given to the Battery by the owner. The encrypted key is then stored in the Battery storage.

A Battery stores configurations details required to locate and start a Volt. A passphrase is specified when the Battery is created. The Battery storage is encrypted by a key derived from the passphrase using PBKDF2-HMAC-SHA512.

Password

The **tdx Volt** key is encrypted using AES-256 in CBC mode with the passphrase assigned to the **tdx Volt** by its owner.

The difference between the 'Battery' and 'Password' strategies is that the 'Battery' strategy means that all Volts contained in the Battery will have their key encrypted by the same passphrase. The 'Password' strategy uses a passphrase unique to each **tdx Volt** to encrypt the key.

File

The ‘File’ key strategy indicates that the key is stored somewhere on a local file system. This can include, for example, a removable encrypted drive. The key can also be encrypted using a passphrase.

Skip to Content

tdx Volt

putting you in charge
☀☾
Toggle sidebar

- **Getting Started**
  - Welcome

- Quick Start

Concepts

- Fundamentals
- Identity
- DID registry
- Verifiable credentials
- Policy
- Resource
- File
- Database
- Wire
- Relay
- Key strategy

How to...

- Create a Volt
- Start a Volt
- Connect to a Volt
- Upload files
- Download files
- Publish to wire
- Subscribe to wire
- Create a database
- Execute SQL
- Import data
- Establish a connect stream
- Approve authenticate request

Clients

- Command line
- Native / C++
- Web
- NodeJS

Reference

- Best practice
- Battery
- Configuration
- Connection
- Connect stream
- Logging
- PKCS#11

API

- Discovery API
- File API
- SqliteDatabase API
- SqliteServer API
- SSI API
- Sync API
- Relay API
- Volt API
- Wire API

- ## Utilities
    - protoDbSync
    - sqliteServer
    - wireTransform
- ## FAQ
    - Questions
- ## Coming soon
    - Roadmap
- ☀☾

File

The **tdx Volt** has comprehensive built-in file support.

Files can be uploaded from the local file system to a **tdx Volt** running anywhere in the world. The **CLI** and **fusebox** also support recursive uploading of entire sub-trees of folders and files.

Policy and file modes

The **tdx Volt** supports 3 modes of file storage:

- **standard** - in this mode a **tdx Volt** resource is created for each file and folder that is uploaded. The file contents are then encrypted and stored locally in the Volt.
- **mirrored** - the mirrored mode is similar to standard mode in that a **tdx Volt** resource is created for each file and folder, however the contents of the file are not uploaded to the Volt. Instead the resource stores a path to the original file on the local disk.
- **linked** - linked mode is a continuation of mirrored mode. A single resource is created on the **tdx Volt** that reflects the ‘root’ folder that was uploaded, and the entire sub-tree is still exposed as descendants of the root folder but implementation wise they are loaded at runtime rather than being cached as resources in the Volt.

Standard mode exactly reflects the local file system as a snapshot at the time of uploading. The resource hierarchy will match that of the local file system, and policy rules can be applied at any level of the hierarchy, e.g. sharing a sub-folder several levels deep in the file tree. If a file is added or deleted or modified on the local file system this will not be reflected in the **tdx Volt** until a sync is run.

Mirrored mode also reflects the local file system as a snapshot in time, however when a file is read or downloaded it will fetch the data from the local disk rather than from the Volt. Policy rules can still be applied at any level in the hierarchy. As a result of this the file contents will always appear in sync with the local disk, but file additions and deletions will not be reflected in the **tdx Volt** resource tree until a sync is run.

Linked mode on the other hand totally reflects the local file system at all times. Additions, deletions and updates will immediately be reflected in the Volt. However it is only possible to apply policy rules at the level of the root folder, i.e. a subject can either see the entire sub-tree or not. It’s a trade-off between performance, flexibility of sharing and staleness of data.

Note that 'mirrored' and 'linked' modes are only applicable to the file system local to the Volt, i.e. Alice can upload a folder tree to Bob's **tdx Volt** using standard mode, but she cannot do so using mirrored or linked mode because the files would be unavailable when Alice is offline or unreachable. In this scenario, Alice would mirror or link the folder to her own **tdx Volt** and add a share for Bob.

Encryption

Files uploaded to the **tdx Volt** are encrypted at rest using a key derived from the **tdx Volt** root key.

Sync

The **tdx Volt** file upload and download commands can run in an optimised mode such that they will not upload or download a file if it has not changed since the last operation, and the upload command implements a ‘watch’ function that can upload files as they change on the local disk.

Skip to Content

tdx Volt

putting you in charge

☀ ☾

Toggle sidebar

- **Getting Started**
  - Welcome
  - Quick Start
- **Concepts**
  - Fundamentals
  - Identity
  - DID registry
  - Verifiable credentials
  - Policy
  - Resource
  - File
  - Database
  - Wire
  - Relay
  - Key strategy
- **How to...**
  - Create a Volt
  - Start a Volt
  - Connect to a Volt
  - Upload files
  - Download files
  - Publish to wire
  - Subscribe to wire
  - Create a database
  - Execute SQL
  - Import data
  - Establish a connect stream
  - Approve authenticate request
- **Clients**
  - Command line

  - Native / C++
  - Web
  - NodeJS

- **Reference**
  - Best practice
  - Battery
  - Configuration
  - Connection
  - Connect stream
  - Logging
  - PKCS#11

- **API**
  - Discovery API
  - File API
  - SqliteDatabase API
  - SqliteServer API
  - SSI API
  - Sync API
  - Relay API
  - Volt API
  - Wire API

- **Utilities**
  - protoDbSync
  - sqliteServer
  - wireTransform

- **FAQ**
  - Questions

- **Coming soon**
  - Roadmap
- ☀🌙

DID registry

A DID Registry is a service that stores and retrieves DIDs and DID documents. It is a key component of the **tdx Volt** decentralized identity ecosystem, as it allows users to manage their DIDs and DID documents in a consistent, reliable, secure and decentralized way.

What is a DID?

A decentralized identfier, or DID, is a new type of identifier that enables verifiable, self-sovereign digital identity. DIDs are fully under the control of the DID subject, independent of any centralized registry, identity provider, or certificate authority.

DIDs are presented as URIs that are resolvable to a DID document, which is a JSON object that contains cryptographic material, authentication suites, and service endpoints.

Below is an example DID document for the DID `did:volt:48c7e0bb-9c62-4db0-be2f-`

`d2f46528ccdb`.

```
{  "@context": ["https://www.w3.org/ns/did/v1",
"https://tdxvolt.com/ns/did/v1"],  "authentication":
["did:volt:48c7e0bb-9c62-4db0-be2f-d2f46528ccdb#key-1"],
"controller": "did:volt:ee300134-69c7-41b7-8736-
13959174d90d",  "id": "did:volt:48c7e0bb-9c62-4db0-be2f-
d2f46528ccdb",  "verificationMethod": [    {      "id":
"did:volt:48c7e0bb-9c62-4db0-be2f-d2f46528ccdb#key-1",
"publicKeyPem": "-----BEGIN PUBLIC KEY-----
\nMCowBQYDK2VwAyEAmrsi5oMFVYWGt3mA6kvxpMIjMOLiylaTUQJelDsslQg
-----END PUBLIC KEY-----\n",      "type":
"Ed25519Signature2018"    }  ]}
```

DID documents do not contain any sensitive or personal information about the DID subject, but rather contain cryptographic material that can be used to verify the identity of the DID subject. It simply binds an opaque identifier to a set of cryptographic keys and services.

For more information, see the W3C Decentralized Identifiers (DIDs) specification .

DID Registry

The DID registry is a database that stores and retrieves DIDs and DID documents.

The **tdx Volt** ecosystem is designed to be decentralized and self-sovereign, so each **tdx Volt** instance maintains its own DID registry. This allows each **tdx Volt** instance to have complete control over its own identity and the identities of other peers that it interacts with. As well as this, any **tdx Volt** instance can be configured to use one or more other DID registries to store and resolve DIDs.

In order for two **tdx Volt** instances, or any two entities in the **tdx Volt** ecosystem, to communicate with each other, they must be able to resolve each other’s DIDs. This is necessary in order to acquire the public key or the other peer and thereby verify their identity and encrypt the communication between the two peers.

To resolve a given DID, the **tdx Volt** or client will begin by querying the local DID registry, and then querying any other DID registries that are configured. If the DID is not found in any of the configured DID registries, the resolution fails.

Hence in order for two peers to verify each other’s identity and encrypt/decrypt data, they must have a common DID registry that they can use to resolve each other’s DIDs.

Currently a number of DID registries are publicly available, and can be used by any **tdx Volt** instance to resolve DIDs.

- coreid.com - a public DID registry operated by the **tdx Volt** community
- tdxvolt.com - a public DID registry and relay operated by nquiringminds Ltd in the UK
- tdxid.com - a development DID registry based in the nquiringminds London office

These DID registries are designed to be highly available and secure and are typically operated by trusted third parties, such as those run by the community or a foundation of some sort.

The idea is that over time a network of DID registries will emerge, each operated by different entities. Some may interoperate with each other, some may not, some may be public, some may be private. Each **tdx Volt** instance can choose which DID registries to use, and can even run its own DID registry if it wishes. This will allow for a highly decentralized and self-sovereign identity ecosystem that is not dependent on any single entity or organization, and enables a high degree of flexibility and interoperability between different **tdx Volt** instances.

Resolution API

Applications and services of the **tdx Volt** platform can resolve and register DIDs using the various APIs.

The DID Registry resolution is exposed as an HTTP endpoint, as well as the grpc ResolveDID and SearchDIDRegistry APIs.

The HTTP endpoint is a simple GET request that takes a DID as a query parameter and returns the DID document as a JSON object - example.

The grpc API is a more advanced API that allows for more complex queries and operations on the DID registry.

Registration API

In order to register a DID in the DID registry, use the Authenticate API.

There is no HTTP endpoint for registering a DID directly.

It is also possible for the client must first create a DID document and then submit it to the DID registry using the RegisterDIDDocument API.

The DID registry will validate the DID document and store it in the database. The DID document will then be available for resolution by other **tdx Volt** instances and clients.

Synchronisation

DID registries can elect to synchronise with each other in order to maintain a consistent view of the DID registry across several **tdx Volt** instances. This also provides a level of fault tolerance and availability, including the ability to resolve DIDs when offline or only partially connected to the network or intranet.

In this configuration, each **tdx Volt** instance maintains its own copy of the DID registry, and the DID registry is designed to be eventually consistent across all **tdx Volt** instances.

When a new DID is registered in the DID registry, the **tdx Volt** instance that registered the DID will broadcast the new DID to all other **tdx Volt** instances in the network. Each **tdx Volt** instance will then update its local copy of the DID registry to include the new DID.

Skip to Content

tdx Volt

putting you in charge
☀ ☾
Toggle sidebar

- **Getting Started**
  - Welcome
  - Quick Start
- **Concepts**
  - Fundamentals
  - Identity
  - DID registry
  - Verifiable credentials
  - Policy
  - Resource
  - File
  - Database
  - Wire
  - Relay
  - Key strategy

- ## How to...
  - Create a Volt
  - Start a Volt
  - Connect to a Volt
  - Upload files
  - Download files
  - Publish to wire
  - Subscribe to wire
  - Create a database
  - Execute SQL
  - Import data
  - Establish a connect stream
  - Approve authenticate request
- ## Clients
  - Command line
  - Native / C++
  - Web
  - NodeJS
- ## Reference
  - Best practice
  - Battery
  - Configuration
  - Connection
  - Connect stream
  - Logging
  - PKCS#11
- ## API
  - Discovery API
  - File API
  - SqliteDatabase API
  - SqliteServer API
  - SSI API
  - Sync API
  - Relay API
  - Volt API
  - Wire API
- ## Utilities
  - protoDbSync
  - sqliteServer
  - wireTransform
- ## FAQ
  - Questions
- ## Coming soon
  - Roadmap
- ☀☾

Relay

A **tdx Volt** Relay enables Volts to bypass firewall and NAT systems and make data and services available to clients on the wider internet.

A Relay is implemented as ‘tunnel’ or bi-directional grpc stream that is initiated from the ‘client’ to the Relay Volt. Once a session is established, the Relay can then proxy grpc calls over the bi-direction byte stream.

For example, consider Alice’s Volt, which is behind a firewall, wishes to invoke functions on a service exposed by Bob’s Volt, which is itself behind a firewall. If both Alice and Bob establish a session with a third Relay Volt, then Alice is able to send the invocation request to Bob via the intermediate Relay Volt.

A Volt may establish connections to many Relay Volts concurrently.

Obviously for the Relay **tdx Volt** to be reachable it will need to be visible on the public internet. However there may be scenarios where it’s desireable to enable Relay on a **tdx Volt** that is not on the public internet, for example to take advantage of the discovery capabilities it brings (see below).

Encryption

The Relay **tdx Volt** has no visibility of the payload, other than the identity fingerprint of the recipient.

All data that passes through the Relay is encrypted before it enters the Relay using the intended recipients public key. When the payload is received it is decrypted and processed before the response is encrypted using the originating callers public key and sent back via the Relay.

Authentication

By default the Relay will require both ends of the ‘pipe’ to be authenticated. This means that both Alice and Bob will need to have bound to the Relay Volt.

A Relay can also run in ‘open’ mode, which means that any client can use the Relay with no authentation required. Note that this only applies to the establishment of the Relay session, authentication and policy will be applied as normal by the **tdx Volt** that is the target of any invocation, and the Relay payload is encrypted no matter what authentication mode.

Discovery

The Relay **tdx Volt** provides a discovery function that allows any **tdx Volt** connected to the Relay to discover the configuration of any other **tdx Volt** connected to the Relay. A **tdx Volt** will only participate in this discovery function if it has set ‘discoverable’ on in the **tdx Volt** settings.

Configuration

The Relay feature is optional and can be switched on and off from the ‘settings’ in the **fusebox** app.

The **tdx Volt** may need to be restarted for any Relay configuration change to take effect.

☀🌙
Toggle sidebar

- ## Getting Started
  - Welcome
  - Quick Start
- ## Concepts
  - Fundamentals
  - Identity
  - DID registry
  - Verifiable credentials
  - Policy
  - Resource
  - File
  - Database
  - Wire
  - Relay
  - Key strategy
- ## How to...
  - Create a Volt
  - Start a Volt
  - Connect to a Volt
  - Upload files
  - Download files
  - Publish to wire
  - Subscribe to wire
  - Create a database
  - Execute SQL
  - Import data
  - Establish a connect stream
  - Approve authenticate request
- ## Clients
  - Command line
  - Native / C++
  - Web
  - NodeJS
- ## Reference
  - Best practice
  - Battery
  - Configuration
  - Connection
  - Connect stream
  - Logging
  - PKCS#11
- ## API
  - Discovery API
  - File API
  - SqliteDatabase API

  - SqliteServer API
  - SSI API
  - Sync API
  - Relay API
  - Volt API
  - Wire API

- **Utilities**

  - protoDbSync
  - sqliteServer
  - wireTransform

- **FAQ**

  - Questions

- **Coming soon**

  - Roadmap
- ☀☾

Policy

The **tdx Volt** security policy is based on XACML.

XACML is an attribute-based policy language. Policy rules are defined in terms of the attributes, and in the case of the **tdx Volt** the 4 main attribute types are subject (the identity), resource, action and environment.

This document won't go into full details of how XACML works, please refer to the XACML standard.

SSI integration

Self-sovereign identity (SSI) is a method of managing digital identities in a way that is independent of any central authority. For more information, see SSI.

This provides a powerful and flexible way to define policy rules that are based on the identity of the subject, and the issuers of the credentials that the subject holds.

For example, a policy rule could be defined that permits a subject to perform an action on a resource if they hold a credential that was issued by a specific issuer, such as a government department.

For more information on how the **tdx Volt** integrates with SSI, see the Verifiable Credentials, Decentralized Identifiers (DIDs) and DID Registry pages.

Implementation

The **tdx Volt** policy engine is a stand-alone library that implements large parts of the XACML standard, as well as the multiple decision and hierarchical resource profiles, with no hardwiring to the **tdx Volt** infrastructure.

Within the **tdx Volt** core there are implementations of Policy Information Points (PIPs) for subject, resource and environment.

Hierarchy

The policy engine supports the hierarchical nature of the **tdx Volt** resource tree via the multiple decision and hierarchical resource profiles.

This allows rules that are applied to a parent resource to be inherited by its descendants, and greatly simplifies the policy rules required to protect the Volt.

Persistence

The **tdx Volt** uses JSON to persist policies rather than XML, but the underlying semantics are the same.

Note that a ‘root’ policy set is created when the **tdx Volt** first boots. This contains the general rules pertaining to **tdx Volt** ownership, resource ownership and so on, and this is persisted in the **tdx Volt** database.

Resource sharing rules that are added as a result of calls to SaveAccess are dynamically included in the policy at runtime.

Examples

The example below shows a policy that permits the **tdx Volt** owner to perform any action, irrespective of the target resource:

```
{  "id": "volt-owner",  "ruleCombiningAlgorithm": "first-
applicable",  "rules": [    {      "description": "permit
**tdx Volt** owner to perform any action",      "effect":
"permit"    }  ],  "target": {    "allOf": [      {
"anyOf": [          {            "allOf": [            {
"attributeId": "tdx:action",                "category":
"action",                "dataType": "string",
"functionId": "string-regexp-match",                "value":
".*"              },              {
"attributeId": "tdx:identityId",              "category":
"subject",                "dataType": "string",
"functionId": "string-equal",                "value":
"449a3385-f380-41f7-bd0a-e60caaa403cb"              }
]          }        ]      }    ]  }}
```

This example shows a rule that permits subjects (identities) to perform any action on any resource they own. It makes use of a ‘condition’ to dynamically interrogate the resource PIP to establish the owning identity. It then compares this to the currently authenticated identity and if the two match, it permits the subject to perform any action:

```
{  "id": "resource-owner",  "ruleCombiningAlgorithm":
"first-applicable",  "rules": [    {      "condition": {
"args": [          {            "attributeId":
"tdx:identityId",            "category": "subject",
"dataType": "string",            "typeName":
"AttributeDesignator"          },          {
"attributeId": "tdx:resourceOwner",            "category":
"resource",            "dataType": "string",
"typeName": "AttributeDesignator"          }        ],
"functionId": "string-equal",        "typeName": "Apply"
},      "description": "permit resource owner
create/delete/read/write access",      "effect": "permit"
}  ],  "target": {    "allOf": [      {        "anyOf": [
{            "allOf": [              {
"attributeId": "tdx:action",                "category":
"action",                "dataType": "string",
"functionId": "string-regexp-match",                "value":
".*"              }            ]          }        ]      }
]  }}
```

Skip to Content

tdx Volt

putting you in charge

☀☾

Toggle sidebar

- ## Getting Started
  - Welcome
  - Quick Start
- ## Concepts
  - Fundamentals
  - Identity
  - DID registry
  - Verifiable credentials
  - Policy
  - Resource
  - File
  - Database
  - Wire
  - Relay
  - Key strategy
- ## How to...
  - Create a Volt
  - Start a Volt
  - Connect to a Volt
  - Upload files
  - Download files
  - Publish to wire
  - Subscribe to wire
  - Create a database
  - Execute SQL
  - Import data
  - Establish a connect stream
  - Approve authenticate request
- ## Clients
  - Command line
  - Native / C++
  - Web
  - NodeJS
- ## Reference
  - Best practice
  - Battery
  - Configuration
  - Connection
  - Connect stream
  - Logging
  - PKCS#11
- ## API
  - Discovery API
  - File API

  - SqliteDatabase API
  - SqliteServer API
  - SSI API
  - Sync API
  - Relay API
  - Volt API
  - Wire API

- ## Utilities

  - protoDbSync
  - sqliteServer
  - wireTransform

- ## FAQ

  - Questions

- ## Coming soon

  - Roadmap
- ☀☾

Contents

Contents

DiscoveryAPI

Top

The public Volt discovery service.

This is exposed by the Volt battery over an INSECURE grpc channel.

This insecurity is ameliorated by the fact that discovered Volts return signatures of their challenge code and owner credential.

Note that only Volts that have explicitly set 'discoverable' in the Volt settings will be discovered.

Discover()

Request: DiscoverRequest
Response: DiscoverResponse

DiscoverRequest

Field	Type	Description
require_relay	bool	Set to require that only Volts that expose a Relay be included in the response.

DiscoverResponse

Field	Type	Description
status	Status	
endpoint	SignedEndpoint repeated	A list of endpoints that match the request criteria.

SignedEndpoint

Field	Type	Description
endpoint	VoltEndpoint	The discovered Volt endpoint information.
challenge_signature	string	The discovered Volt's challenge code, signed by the Volt private key and base64 encoded. If a client knows the Volt challenge code by some out-of-band means, it can use the Volt public key (contained in the endpoint information above) to determine that the discovered Volt also knows the same challenge code.

Status

Field	Type	Description
code	int32	A simple error code that can be easily handled by the client. Mirrors the grpc StatusCode enum, 0 => OK
message	string	A developer-facing human-readable error message in English. It should both explain the error and offer an actionable resolution to it.
description	string	Long form error description.

Scalar Value Types

.proto Type	Notes	C++	Java	Python	Go	C#	PHI
double		double	double	float	float64	double	float
float		float	float	float	float32	float	float
	Uses variable-length encoding. Inefficient						

.proto Type	Notes	C++	Java	Python	Go	C#	PH
int32	for encoding negative numbers – if your field is likely to have negative values, use sint32 instead.	int32	int	int	int32	int	integer
int64	Uses variable-length encoding. Inefficient for encoding negative numbers – if your field is likely to have negative values, use sint64 instead.	int64	long	int/long	int64	long	integer/s
uint32	Uses variable-length encoding.	uint32	int	int/long	uint32	uint	integer
uint64	Uses variable-length encoding.	uint64	long	int/long	uint64	ulong	integer/s
sint32	Uses variable-length encoding. Signed int value. These more efficiently encode negative numbers than regular int32s.	int32	int	int	int32	int	integer
sint64	Uses variable-length encoding. Signed int value. These more efficiently encode negative numbers than regular int64s.	int64	long	int/long	int64	long	integer/s
	Always						

.proto Type	Notes	C++	Java	Python	Go	C#	PH
fixed32	Always four bytes. More efficient than uint32 if values are often greater than 2^28.	uint32	int	int	uint32	uint	integer
fixed64	Always eight bytes. More efficient than uint64 if values are often greater than 2^56.	uint64	long	int/long	uint64	ulong	integer/s
sfixed32	Always four bytes.	int32	int	int	int32	int	integer
sfixed64	Always eight bytes.	int64	long	int/long	int64	long	integer/s
bool		bool	boolean	boolean	bool	bool	boolean
string	A string must always contain UTF-8 encoded or 7-bit ASCII text.	string	String	str/unicode	string	string	string
bytes	May contain any arbitrary sequence of bytes.	string	ByteString	str	[]byte	ByteString	string

Skip to Content

tdx Volt

putting you in charge
☀🌙
Toggle sidebar

- **Getting Started**
  - Welcome
  - Quick Start

- **Concepts**

  - Fundamentals
  - Identity
  - DID registry
  - Verifiable credentials
  - Policy
  - Resource
  - File
  - Database
  - Wire
  - Relay
  - Key strategy

- **How to...**

  - Create a Volt
  - Start a Volt
  - Connect to a Volt
  - Upload files
  - Download files
  - Publish to wire
  - Subscribe to wire
  - Create a database
  - Execute SQL
  - Import data
  - Establish a connect stream
  - Approve authenticate request

- **Clients**

  - Command line
  - Native / C++
  - Web
  - NodeJS

- **Reference**

  - Best practice
  - Battery
  - Configuration
  - Connection
  - Connect stream
  - Logging
  - PKCS#11

- **API**

  - Discovery API
  - File API
  - SqliteDatabase API
  - SqliteServer API
  - SSI API
  - Sync API
  - Relay API
  - Volt API
  - Wire API

- **Utilities**

  - protoDbSync

    - sqliteServer
    - wireTransform

- ## FAQ
    - Questions

- ## Coming soon
    - Roadmap
- ☀☾

Contents

Contents

SqliteDatabaseAPI

Top

The Sqlite Database API exposes functions that enable clients to manipulate data in a given Volt database.

Use the Sqlite Server API to create the database resource.

BulkUpdate()

Execute multiple SQL statements in a single transaction via a single RPC.

Request: SqlBulkUpdateRequest
Response: SqlBulkUpdateResponse

Execute()

Execute a single SQL statement. Any valid SQL is accepted. In order to execute non-SELECT statements, the caller must have the "write" permission.

Request: streaming SqlExecuteRequest
Response: streaming SqlExecuteResponse

ImportCSV()

Import CSV data into a table.

The import handler will inspect the incoming CSV data, infer the columns and data types required, and create and populate the SQL table.

The CSV must contain a header row containing the column names and at least one row of data so that the types can be inferred.

The importer assumes all data in any given CSV file relates to a single table.

The data can either be streamed in chunks or retrieved from an existing resource.

If the data is streamed in chunks, the client must call the Close() method on the stream to indicate that it has finished.

Request: streaming SqlImportCSVRequest
Response: streaming SqlImportCSVResponse

SqlBulkUpdateRequest

Field	Type	Description
database_id	string	The id of the database to update.
statement	string repeated	The SQL statements to execute. The statements will be executed within a transaction and will be committed if all statements succeed. Bear in mind the maximum size limit of a single message is 64MB, and around 1MB seems to be optimal in terms of performance.

SqlBulkUpdateResponse

Field	Type	Description
status	tdx.volt_api.volt.v1.Status	Details of any error that occurred on the call.

SqlExecuteEnd

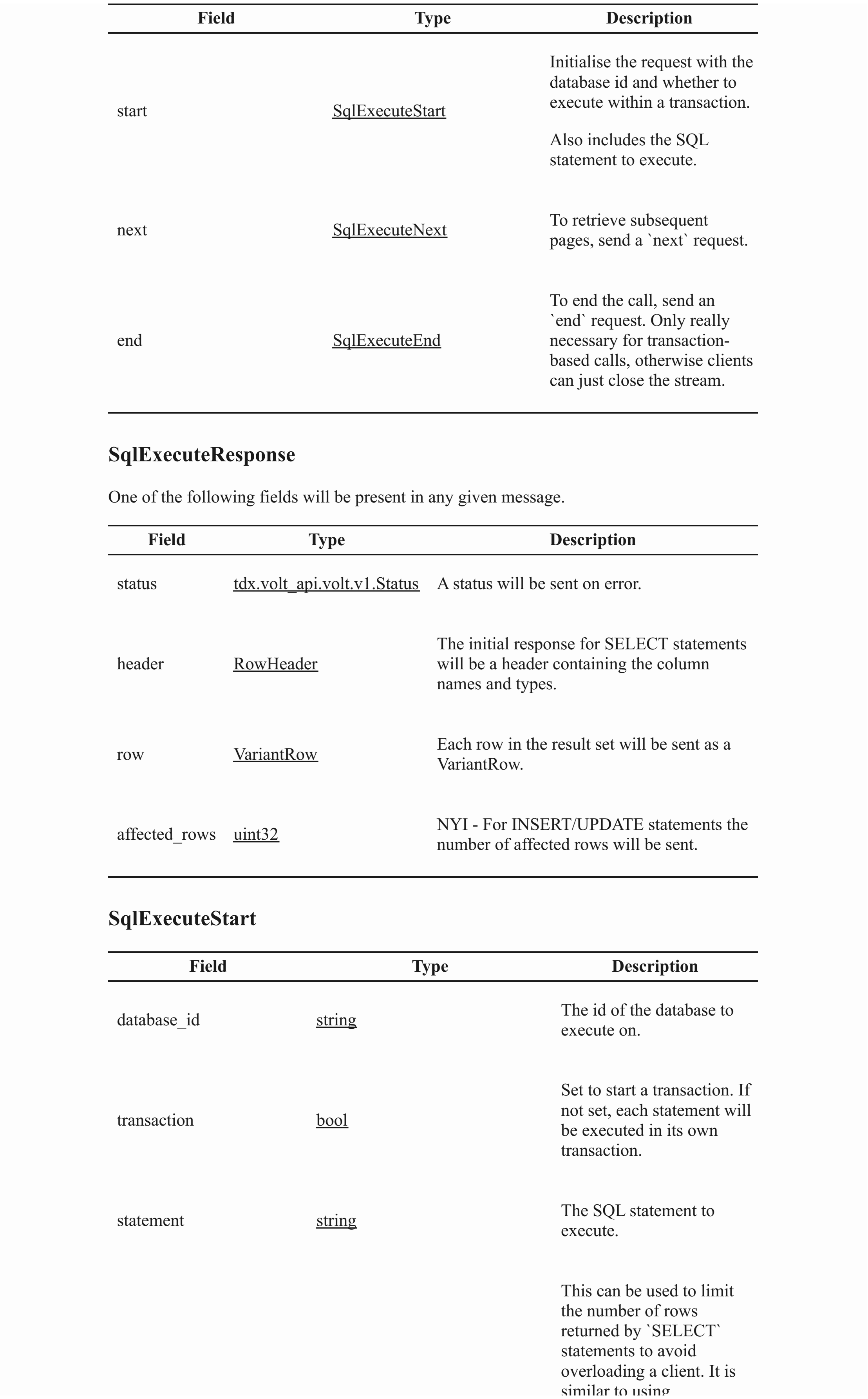
Ends the call.

Field	Type	Description
commit_transaction	bool	Set to commit the transaction. If not set, the transaction will be rolled back.

SqlExecuteNext

Intentionally empty.

SqlExecuteRequest



Field	Type	Description
start	SqlExecuteStart	Initialise the request with the database id and whether to execute within a transaction. Also includes the SQL statement to execute.
next	SqlExecuteNext	To retrieve subsequent pages, send a `next` request.
end	SqlExecuteEnd	To end the call, send an `end` request. Only really necessary for transaction-based calls, otherwise clients can just close the stream.

SqlExecuteResponse

One of the following fields will be present in any given message.

Field	Type	Description
status	tdx.volt_api.volt.v1.Status	A status will be sent on error.
header	RowHeader	The initial response for SELECT statements will be a header containing the column names and types.
row	VariantRow	Each row in the result set will be sent as a VariantRow.
affected_rows	uint32	NYI - For INSERT/UPDATE statements the number of affected rows will be sent.

SqlExecuteStart

Field	Type	Description
database_id	string	The id of the database to execute on.
transaction	bool	Set to start a transaction. If not set, each statement will be executed in its own transaction.
statement	string	The SQL statement to execute.
		This can be used to limit the number of rows returned by `SELECT` statements to avoid overloading a client. It is similar to using

Field	Type	Description
page_size	uint32	'LIMIT/OFFSET' clauses on the 'SELECT' statement, but this method is easier to manage and the entire result set will be prepared on the Volt, and the client can then page through it by sending successive messages.
can_cancel	bool	Set to indicate that the query will be cancelled if the client disconnects. This is useful for long running queries, where it might be desirable for the client to be able to cancel the request before all the data is received. However this requires a worker thread to be allocated to the query until it completes, which may affect performance and limit the number of concurrent queries that can be executed due to file handle limitations. Only applicable to `SELECT` statements, ignored otherwise.
parameter	SqlExecuteStart.ParameterEntry repeated	The values to set for each parameter defined in a database view. This field is only relevant to 'query' databases, i.e. those with kind `tdx:sqlite-view`. The values in the map should be keyed by parameter name. A query may have no parameters defined, in which case leave this field empty.

SqlExecuteStart.ParameterEntry

Field	Type	Description
key	string	
value	tdx.volt_api.volt.v1.AttributeValue	

SqlImportCSVConfiguration

Field	Type	Description
database_id	string	The target database resource id, which must already exist, and the authenticated account must have write access to the resource.
table_name	string	The target table name.
create_table	bool	Not yet implemented - flag indicating that data is to be inserted into an existing table.
source_resource_id	string	Optional - when not sending the data via this RPC, this should contain the id of a resource that contains the CSV data. If set, the authenticated account must have read access to the resource.
progress_interval	uint32	Optional - the interval to receive progress updates, in number of rows. E.g. set to 1000 to receive progress updates every 1000 rows. Set to 0 to disable progress updates.
schema_scan_limit	uint32	The number of rows scanned to infer the schema. Set to 0 to scan the entire file.

SqlImportCSVRequest

Each message must contain one and only one of the following fields.

Field	Type	Description
configuration	SqlImportCSVConfiguration	The initial request must contain the configuration parameters for the import.
upload_complete	bool	Set to indicate that the upload is complete.
block	bytes	Subsequent requests may contain the data to be imported, unless importing from an existing resource that contains the CSV data. It is recommended that the block size is less than 1MB.
abort	bool	Set to abort the import.

SqlImportCSVResponse

Field	Type	Description
status	tdx.volt_api.volt.v1.Status	The status will contain any error info.
row_count	int64	The number of rows processed - this will be sent after every `progress_interval` rows.
total_rows	int64	The total number of rows to be imported.

Status

Field	Type	Description
code	int32	A simple error code that can be easily handled by the client. Mirrors the grpc StatusCode enum, 0 => OK
message	string	A developer-facing human-readable error message in English. It should both explain the error and offer an actionable resolution to it.
description	string	Long form error description.

Column

Field	Type	Description
name	string	
description	string	
type	DataType	

RowHeader

Field	Type	Description
column	Column repeated	

Schema

Field	Type	Description
name	string	
description	string	
column	Column repeated	

Variant

Field	Type	Description
text	string	
integer	int64	
real	double	
blob	bytes	
null	bool	

VariantRow

Field	Type	Description
column	Variant repeated	

DataType

Just reflect SQLite types for now.

Name	Number	Description
DATA_TYPE_UNKNOWN	0	
DATA_TYPE_TEXT	1	
DATA_TYPE_INTEGER	2	
DATA_TYPE_REAL	3	
DATA_TYPE_BLOB	4	
DATA_TYPE_NULL	5	

Access

Field	Type	Description
id	string	
resource_id	string	The resource being accessed.
resource_name	string	A human-readable short identifier of the resource.
resource_owner	string	The identity that owns the resource.
resource_kind	string repeated	The kind of resource.
identity_did	string	The identity attempting access.
credential_lookup	string	The JSON path array for looking up verifiable credentials.
identity_name	string	A human-readable short identifier of the subject.
identity_kind	string repeated	The kind of identity.
access	string	Requested access.
extra	string	Optional extra data.
decision	PolicyDecision	Assigned decision.
recursive	bool	
request_time	int64	Time at which the request was made.
decision_time	int64	Time at which the decision was taken.
request_count	uint32	Counter of number times this access was requested.

AttributeValue

Attribute value will be one of the following fields, depending on the data type.

Field	Type	Description
string	string	
integer	int64	
real	double	
boolean	bool	
bytes	bytes	

Identity

A Volt identity encompasses a Resource and a set of identity aliases.

Field	Type	Description
resource	Resource	
alias	IdentityAlias repeated	

IdentityAlias

Field	Type	Description
id	uint32	The alias id.
identity_did	string	The corresponding identity id.
alias	string	The actual alias, e.g. a common name or key fingerprint.
public_key	string	This will only be populated if alias_type == tdx:public-key
private_key	string	This will only be populated if alias_type == tdx:public-key, and the key is stored in the Volt.
alias_type	string	The alias type, for example public key, email, phone number etc.
issuer_id	string	The identity that issued this alias.
authenticate	PolicyDecision	Indicates if this alias has an authenticate policy decision assigned.
description	string	Optional description of this alias.

MethodDescription

Internal use only.

Field	Type	Description
path	string	
client_streaming	bool	
server_streaming	bool	

ProtoFile

Describes a single protobuf file for use in ServiceDescription.

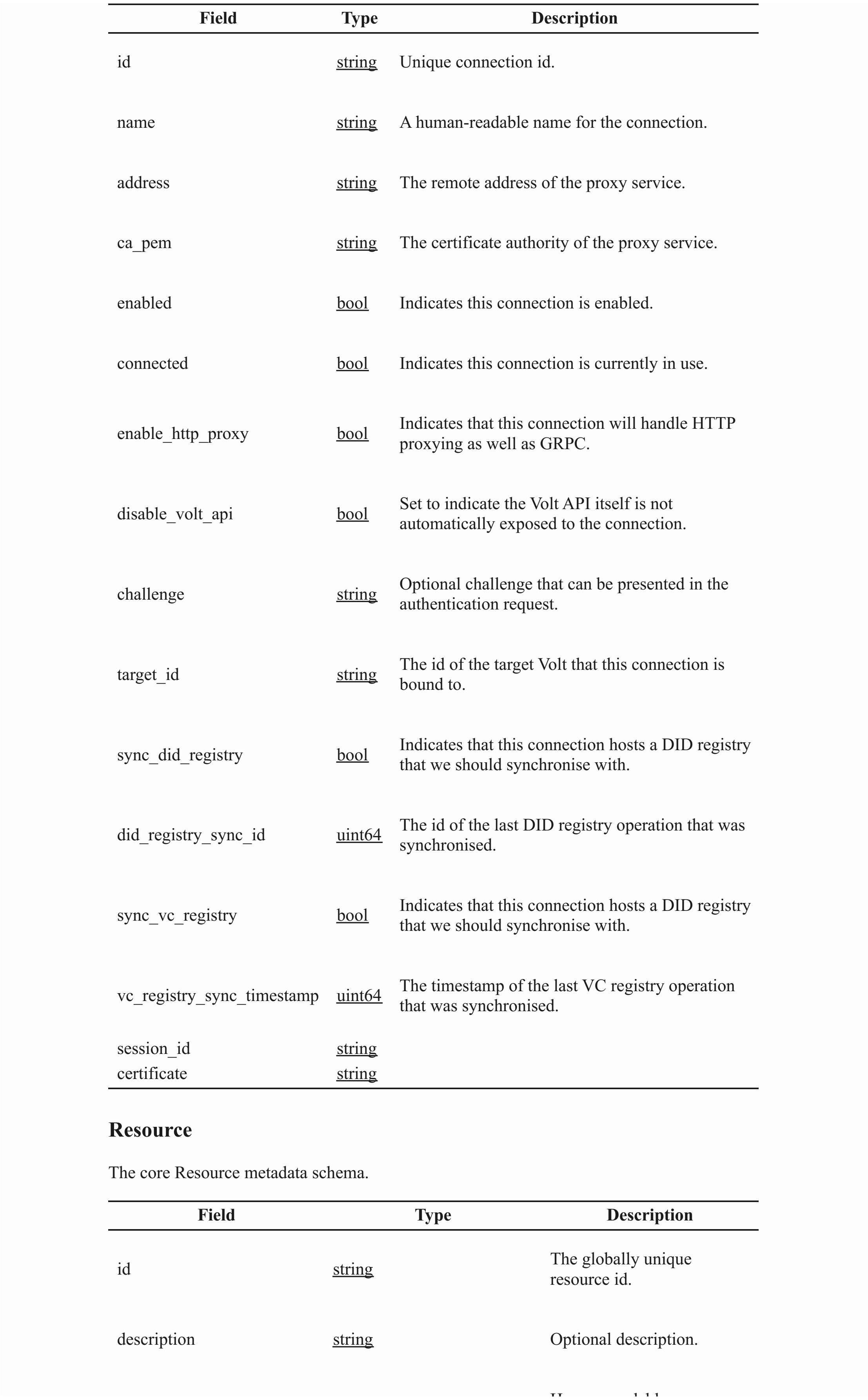
Field	Type	Description
file_path	string	The path name of the proto file, relative to the 'root' of the namespace, e.g. "tdx/volt_api/volt/v1/volt.proto".
protobuf	string	The actual protobuf file contents.
service_name	string repeated	Optional - the service(s) contained in this protobuf file, if omitted here they will be loaded dynamically from the protobuf.

ProxyConnection

Represents an outbound connection from a Volt to a remote service that will act as a proxy for that Volt.

This enables Volts to bypass firewall and NATs.

Example - connection from a Volt to a Relay Volt running on the public internet, such as tdxvolt.com



Field	Type	Description
id	string	Unique connection id.
name	string	A human-readable name for the connection.
address	string	The remote address of the proxy service.
ca_pem	string	The certificate authority of the proxy service.
enabled	bool	Indicates this connection is enabled.
connected	bool	Indicates this connection is currently in use.
enable_http_proxy	bool	Indicates that this connection will handle HTTP proxying as well as GRPC.
disable_volt_api	bool	Set to indicate the Volt API itself is not automatically exposed to the connection.
challenge	string	Optional challenge that can be presented in the authentication request.
target_id	string	The id of the target Volt that this connection is bound to.
sync_did_registry	bool	Indicates that this connection hosts a DID registry that we should synchronise with.
did_registry_sync_id	uint64	The id of the last DID registry operation that was synchronised.
sync_vc_registry	bool	Indicates that this connection hosts a DID registry that we should synchronise with.
vc_registry_sync_timestamp	uint64	The timestamp of the last VC registry operation that was synchronised.
session_id	string	
certificate	string	

Resource

The core Resource metadata schema.

Field	Type	Description
id	string	The globally unique resource id.
description	string	Optional description.

Field	Type	Description
name	string	Human-readable resource name.
share_mode	ShareMode	Not in use.
volt_id	string	The id of the Volt that hosts this resource.
service_description	ServiceDescription	Optional description of any services exposed by this resource.
attribute	ResourceAttribute repeated	Attributes assigned to the resource.
platform_version	Version	The version of the platform.
version	uint64	The resource version.
owner	string	The identity of the resource owner.
created	uint64	Creation timestamp, milliseconds since epoch.
modified	uint64	Last modification timestamp, milliseconds since epoch.
status	ResourceStatus	Not in use.
kind	string repeated	The taxonomy of the resource.
online_status	OnlineStatus	The online status. For most kinds of resource this indicates that the server hosting the resource is online, the exception being identity resources, in which case the status reflects whether or not the identity has a live connection. All built-in resources are hosted by the Volt itself and are therefore always online when the Volt is running. Resources hosted by external servers are online if the server itself is online and has registered the resource as online using `setServiceStatus`.

Field	Type	Description
size	uint64	The size of the resource store in bytes.
store	string	The path to the resource store.
alias	string repeated	Alias(es) that can be used to refer to the resource rather than the id. Each alias must be unique to the Volt, this is enforced by the API. No format restrictions are currently applied to alias, but this may change in future, for the time being it makes sense to stick to alphanumeric characters and '_' or '-'.
content_hash	string	The hash of the resource content contained in the store.
child	Resource repeated	Not yet supported.

ResourceAttribute

A resource attribute enables storing arbitrary data associated with a resource.

Field	Type	Description
id	uint32	
attribute_id	string	
resource_id	string	
data_type	AttributeDataType	
value	AttributeValue repeated	

ServiceDescription

Describes a Volt service.

Field	Type	Description
host_type	ServiceHostType	The configuration used by the host of this service.
host_client_id	string	The identity of the client that is exposing the service. For example, if a third party is exposing a database service via a Volt, it will first authenticate and obtain a client DID and credentials in order to be able to create service resource(s). Any resources that are owned by this client will be marked as online if the client itself is online, i.e. has a live connection to the Volt.

Field	Type	Description
		This will be empty if the service is a built-in Volt service.
host_service_id	string	The id of the resource that holds the protobuf definition for this resource. For example, if a third party is exposing a database service via a Volt, it will create a service resource that holds details of the protobuf methods exposed by the service. For built-in services, i.e. those hosted by the Volt, this will set to the Volt id.
host_address	string	The address of the grpc server hosting this service. Only relevant to grpc-hosted services.
host_ca_pem	string	The certificate authority (chain) that signed the service server certificate. This is only relevant to grpc-hosted services.
host_public_key	string	The public key of the service host, which is used to encrypt payloads. This may change as the service comes and goes online.
host_connection_id	string	The connection id currently used to host this service.
host_session_id	string	Internal use only.
discoverable	DiscoveryMode	The discovery mode.
ping_timestamp	int64	The ping timestamp of the server hosting this service.
proto_file	ProtoFile repeated	The protobuf definitions of the APIs exposed by this service.
service_api	string repeated	The fully qualified names of the protobuf services, for example tdx.volt_api.webcam.v1.WebcamControlAPI.
method	MethodDescription repeated	Internal use only.

Session

Field	Type	Description
id	string	
identity_did	string	
identity_name	string	
ip	string	
created	uint64	
modified	uint64	
expires	uint64	
credential	SessionCredential repeated	
origin	string	
status	SessionStatus	

SessionCredential

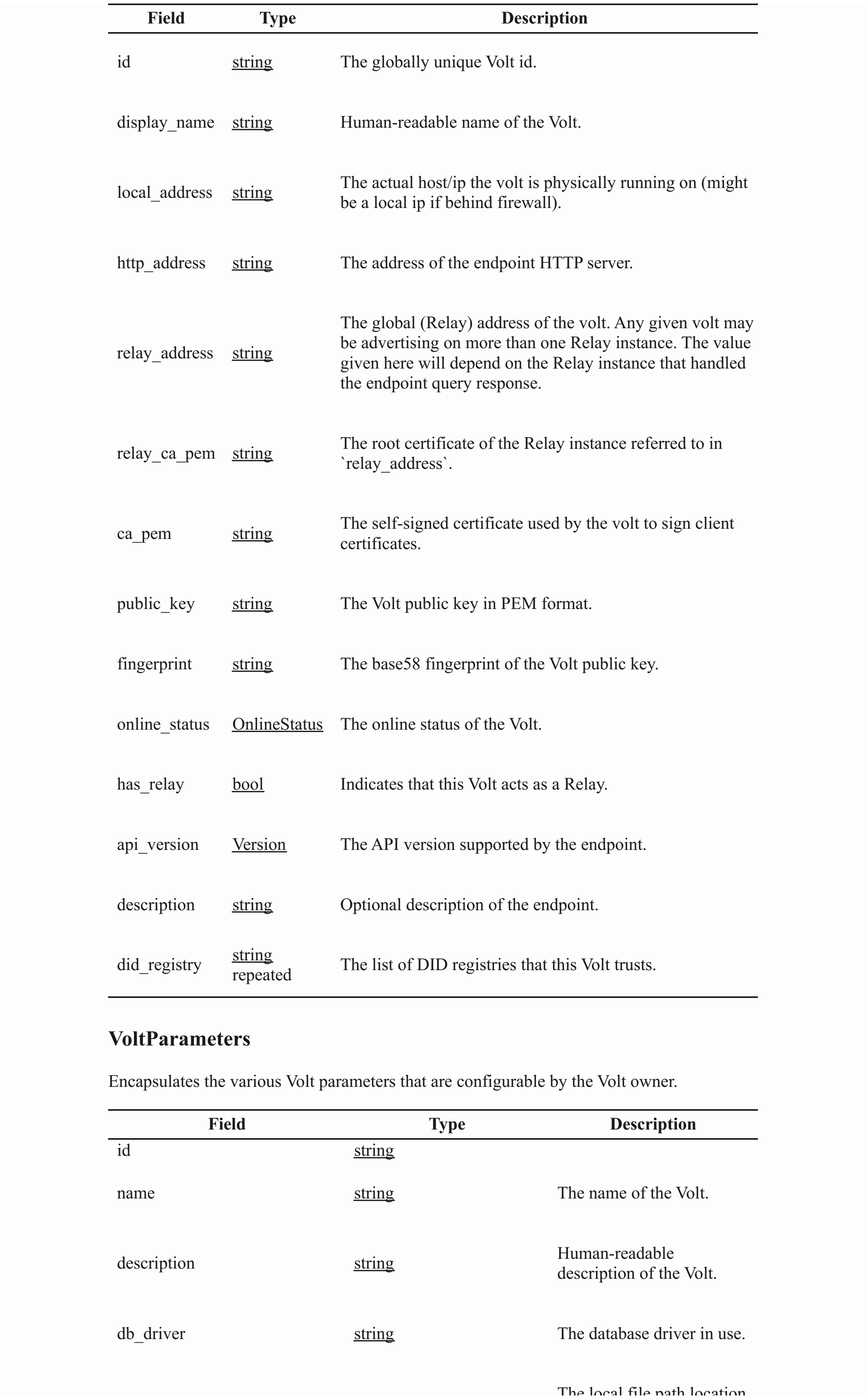
Field	Type	Description
id	uint32	The alias id.
session_id	string	The corresponding session id.
credential_type	string	The credential type, for example public key, verifiable credential, challenge etc.
description	string	Optional description of this credential.
vc_id	string	The id of the verifiable credential, if the credential type is volt:vc-claim.
vc_json	string	The verifiable credential in JSON format, if the credential type is volt:vc-claim.
vc_subject_id	string	The subject id extracted from the `vc_json` field.
vc_issuer_id	string	The issuer id extracted from the `vc_json` field.
vc_type	string	The comma-separated type(s) extracted from the `vc_json` field.
challenge	string	The challenge string, if the credential type is volt:challenge.
key_fingerprint	string	The key fingerprint, if the credential type is volt:public-key.
public_key	string	The PEM-encoded public key, if the credential type is volt:public-key.
private_key	string	Optional PEM-encoded private key, if the credential type is volt:public-key. Only used for ephemeral REST-base sessions created dynamically after OTP authentication.
extra	string	Type-specific extra data stored with the credential.
extra_2	string	More type-specific data stored with the credential.

Version

Using `major` and `minor` here upsets the GNU C Library, so we add a `version_` prefix.

Field	Type	Description
version_major	uint32	
version_minor	uint32	
version_patch	uint32	

VoltEndpoint



Field	Type	Description
id	string	The globally unique Volt id.
display_name	string	Human-readable name of the Volt.
local_address	string	The actual host/ip the volt is physically running on (might be a local ip if behind firewall).
http_address	string	The address of the endpoint HTTP server.
relay_address	string	The global (Relay) address of the volt. Any given volt may be advertising on more than one Relay instance. The value given here will depend on the Relay instance that handled the endpoint query response.
relay_ca_pem	string	The root certificate of the Relay instance referred to in \`relay_address\`.
ca_pem	string	The self-signed certificate used by the volt to sign client certificates.
public_key	string	The Volt public key in PEM format.
fingerprint	string	The base58 fingerprint of the Volt public key.
online_status	OnlineStatus	The online status of the Volt.
has_relay	bool	Indicates that this Volt acts as a Relay.
api_version	Version	The API version supported by the endpoint.
description	string	Optional description of the endpoint.
did_registry	string repeated	The list of DID registries that this Volt trusts.

VoltParameters

Encapsulates the various Volt parameters that are configurable by the Volt owner.

Field	Type	Description
id	string	
name	string	The name of the Volt.
description	string	Human-readable description of the Volt.
db_driver	string	The database driver in use.
		The local file path location

Field	Type	Description
location	string	The local file path location of the Volt storage.
key_strategy	string	The key strategy in use, this determines how the root key is stored.
key_id	string	The identifier for the key, the semantics depend on the key strategy in use.
ca_pem	string	The Volt certificate authority.
cert_pem	string	The Volt API server certificate.
fixed_host	string	Optional hostname of the Volt if using DNS or a static IP address, e.g. tdxvolt.com
grpc_port	int32	Port to use for hosting the Volt management service.
http_port	int32	Port to use for hosting the Volt grpc service.
http_key_path	string	The Volt http server key file path.
http_cert_path	string	The Volt http server certificate file path.
http_ca_path	string	The Volt http server certificate authority chain file path.
discoverable	bool	Indicates the Volt will be discoverable by clients using the discovery api.
authenticate_challenge	string	Optional challenge code that can be used aid in the process of authenticating clients.
require_authenticate_challenge	bool	Indicates that clients must present the correct challenge code in order to be able to authenticate.
confirm_stop	bool	Internal use only.
auto_start	bool	Internal use only.

Field	Type	Description
enable_messaging	bool	Internal use only.
has_relay	bool	Set to indicate this Volt acts as a Relay. This means this Volt can act as a proxy for other Volts (or in fact any client) that connect to it.
relay_open	bool	Set to run the Relay open to any client, i.e. clients can utilise the Relay without first authenticating.
enable_http_server	bool	Determines if the Volt HTTP server is enabled.
http_server_secure	bool	Determines whether the HTTP server employs TLS.
enable_http_forwarding	bool	Determines whether the HTTP server supports forwarding.
enable_http_api	bool	Determines if the Volt REST API is exposed via the HTTP server.
enable_websocket_api	bool	Determines if the Volt Websocket API is exposed via the HTTP server.
address	string	The hostname:port at which the Volt API is currently running.
encrypt_file_store	bool	Set to indicate the Volt file store is encrypted.
connection_id	string	This is a unique connection id. Indicates that these parameters refer to a connection to a remote Volt rather than a local Volt.
relay_ca_pem	string	The certificate authority of the Relay if this is a remote connection via a Relay.
		Optional override of the http address, rather than using the default of fixed_host:http_port

Field	Type	Description
http_address_override	string	fixed_host:http_port. This is useful if the Volt is behind a firewall or NAT, and the http server is listening on a different port from 80 or 443 but this is hidden by the proxy. For example, if the `fixed_host` is `coreid.com` and http server is listening on 2115, but the proxy is forwarding 443 to 2115, then the http_address_override would be set to `https://coreid.com`.
alias	string	An optional alias that can be used to refer to the Volt rather than the `id` field. This alias must be unique within the scope of the Battery in which the Volt is stored.
version	Version	The runtime version this Volt is running.
approve_on_challenge	bool	If set, indicates that any client that provides the correct challenge during authentication will automatically be approved to access the Volt.
approve_on_did	bool	If set, indicates that any client that proves ownership of a DID known to the Volt will automatically be approved to access the Volt.
enable_did_registry	bool	If set, indicates that clients can register DIDs with this Volt.
did_registry	string repeated	Zero or more URLs of trusted peer DID registries.
enable_outbound_smtp	bool	If set, enables outbound SMTP.
outbound_smtp_host	string	The SMTP host to use for sending emails.
outbound_smtp_port	uint32	The SMTP port to use for sending emails.

sending emails.

Field	Type	Description
outbound_smtp_user	string	The SMTP username to use for sending emails.
outbound_smtp_password	string	The SMTP password to use for sending emails.
enable_anonymous_create	bool	If set, enables sessions that authenticate using credentials rather than a DID to create resources in the 'anonymous' system folder.
catch_all_auth_decision	PolicyDecision	The decision to apply to all authentication requests that do not match any other policy. The default is PROMPT.
enable_policy_cache	bool	If set, enables caching of policy decisions.
enable_terminal	bool	If set, enables the terminal API.
start_time	uint64	The time at which the Volt was started.

AttributeDataType

Attribute data types.

Name	Number	Description
ATTRIBUTE_DATA_TYPE_UNKNOWN	0	
ATTRIBUTE_DATA_TYPE_STRING	1	
ATTRIBUTE_DATA_TYPE_INTEGER	2	
ATTRIBUTE_DATA_TYPE_REAL	3	
ATTRIBUTE_DATA_TYPE_BOOLEAN	4	
ATTRIBUTE_DATA_TYPE_BYTES	5	
ATTRIBUTE_DATA_TYPE_IDENTITY	100	
ATTRIBUTE_DATA_TYPE_RESOURCE	101	

DiscoveryMode

Name	Number	Description
DISCOVERY_MODE_UNKNOWN	0	
DISCOVERY_MODE_TRUSTED	1	Only local identities with explicit policy PERMIT can discover.
DISCOVERY_MODE_PUBLIC	2	Any bound local identity can discover.
DISCOVERY_MODE_TRUSTED_GLOBAL	3	Only identities with explicit policy PERMIT can discover, and the service will be available to local and non-local (Relayed) clients.
DISCOVERY_MODE_PUBLIC_GLOBAL	4	Any bound identity can discover, and the service will be available to local and non-local (Relayed) clients.

OnlineStatus

Name	Number	Description
ONLINE_STATUS_UNKNOWN	0	
ONLINE_STATUS_ONLINE	1	
ONLINE_STATUS_OFFLINE	2	

PolicyDecision

@todo currently this must align with AuthorisationDecision enum in policy library, but some of the values are irrelevant outside of the public API so we need a public-facing enum and some translation.

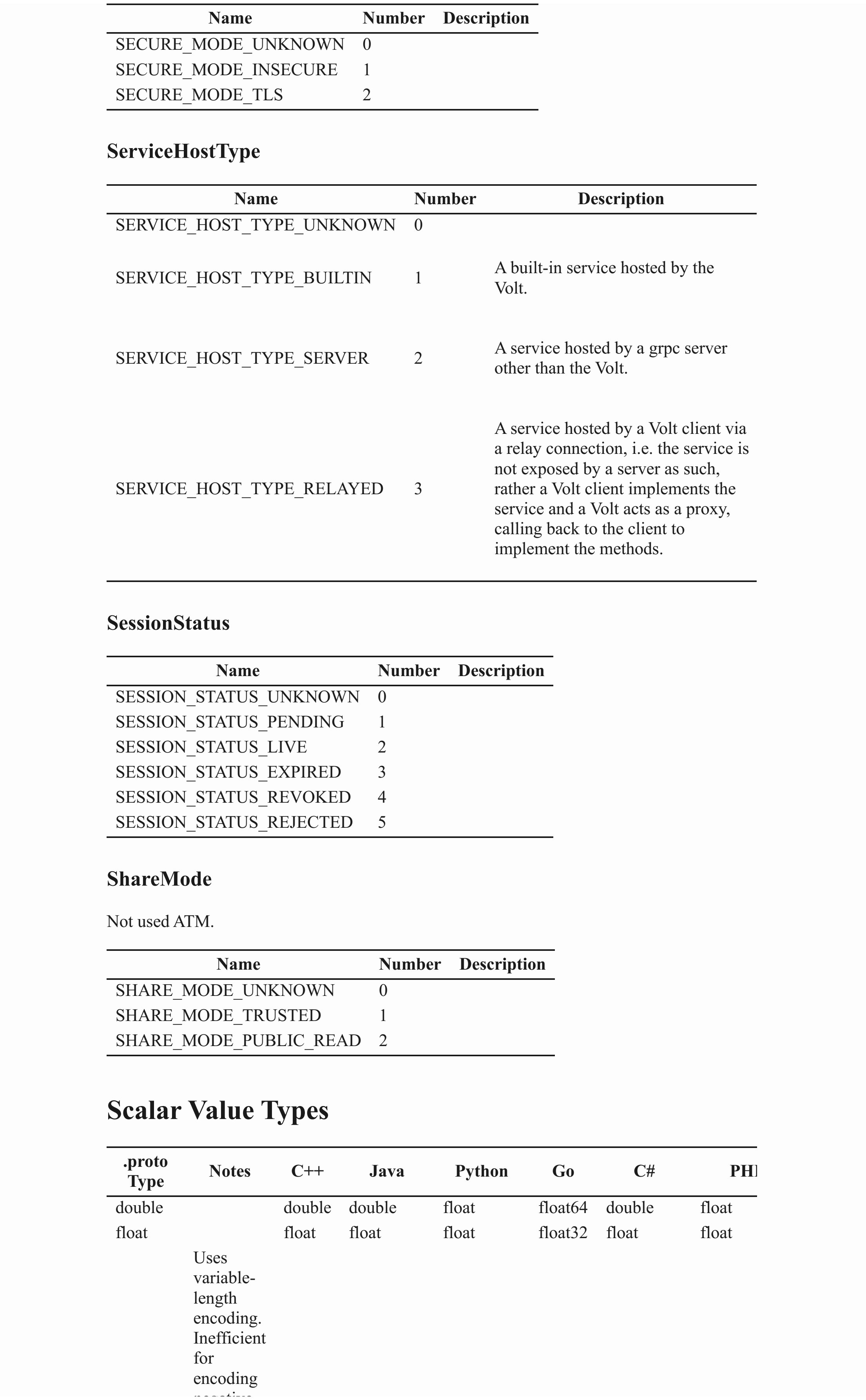
Name	Number	Description
POLICY_DECISION_UNKNOWN	0	
POLICY_DECISION_PROMPT	1	
POLICY_DECISION_PERMIT	2	
POLICY_DECISION_DENY	3	
POLICY_DECISION_INDETERMINATE	4	
POLICY_DECISION_NOT_APPLICABLE	5	
POLICY_DECISION_APPLICABLE	6	
POLICY_DECISION_PENDING	7	

ResourceStatus

Not used ATM.

Name	Number	Description
RESOURCE_STATUS_UNKNOWN	0	
RESOURCE_STATUS_LIVE	1	
RESOURCE_STATUS_INACTIVE	2	
RESOURCE_STATUS_DELETED	999	

SecureMode



Name	Number	Description
SECURE_MODE_UNKNOWN	0	
SECURE_MODE_INSECURE	1	
SECURE_MODE_TLS	2	

ServiceHostType

Name	Number	Description
SERVICE_HOST_TYPE_UNKNOWN	0	
SERVICE_HOST_TYPE_BUILTIN	1	A built-in service hosted by the Volt.
SERVICE_HOST_TYPE_SERVER	2	A service hosted by a grpc server other than the Volt.
SERVICE_HOST_TYPE_RELAYED	3	A service hosted by a Volt client via a relay connection, i.e. the service is not exposed by a server as such, rather a Volt client implements the service and a Volt acts as a proxy, calling back to the client to implement the methods.

SessionStatus

Name	Number	Description
SESSION_STATUS_UNKNOWN	0	
SESSION_STATUS_PENDING	1	
SESSION_STATUS_LIVE	2	
SESSION_STATUS_EXPIRED	3	
SESSION_STATUS_REVOKED	4	
SESSION_STATUS_REJECTED	5	

ShareMode

Not used ATM.

Name	Number	Description
SHARE_MODE_UNKNOWN	0	
SHARE_MODE_TRUSTED	1	
SHARE_MODE_PUBLIC_READ	2	

Scalar Value Types

.proto Type	Notes	C++	Java	Python	Go	C#	PH
double		double	double	float	float64	double	float
float		float	float	float	float32	float	float
	Uses variable-length encoding. Inefficient for encoding						

proto Type	Notes	C++	Java	Python	Go	C#	PH
int32	negative numbers – if your field is likely to have negative values, use sint32 instead.	int32	int	int	int32	int	integer
int64	Uses variable-length encoding. Inefficient for encoding negative numbers – if your field is likely to have negative values, use sint64 instead.	int64	long	int/long	int64	long	integer/s
uint32	Uses variable-length encoding.	uint32	int	int/long	uint32	uint	integer
uint64	Uses variable-length encoding.	uint64	long	int/long	uint64	ulong	integer/s
sint32	Uses variable-length encoding. Signed int value. These more efficiently encode negative numbers than regular int32s.	int32	int	int	int32	int	integer
sint64	Uses variable-length encoding. Signed int value. These more efficiently encode negative numbers than regular int64s.	int64	long	int/long	int64	long	integer/s
	Always four bytes. M						

.proto Type	Notes	C++	Java	Python	Go	C#	PH
fixed32	More efficient than uint32 if values are often greater than 2^28.	uint32	int	int	uint32	uint	integer
fixed64	Always eight bytes. More efficient than uint64 if values are often greater than 2^56.	uint64	long	int/long	uint64	ulong	integer/s
sfixed32	Always four bytes.	int32	int	int	int32	int	integer
sfixed64	Always eight bytes.	int64	long	int/long	int64	long	integer/s
bool		bool	boolean	boolean	bool	bool	boolean
string	A string must always contain UTF-8 encoded or 7-bit ASCII text.	string	String	str/unicode	string	string	string
bytes	May contain any arbitrary sequence of bytes.	string	ByteString	str	[]byte	ByteString	string

Skip to Content

tdx Volt

putting you in charge

☀☾

Toggle sidebar

- ## Getting Started
  - Welcome
  - Quick Start
- ## Concepts
  - Fundamentals

  - Identity
  - DID registry
  - Verifiable credentials
  - Policy
  - Resource
  - File
  - Database
  - Wire
  - Relay
  - Key strategy

- **How to...**
  - Create a Volt
  - Start a Volt
  - Connect to a Volt
  - Upload files
  - Download files
  - Publish to wire
  - Subscribe to wire
  - Create a database
  - Execute SQL
  - Import data
  - Establish a connect stream
  - Approve authenticate request

- **Clients**
  - Command line
  - Native / C++
  - Web
  - NodeJS

- **Reference**
  - Best practice
  - Battery
  - Configuration
  - Connection
  - Connect stream
  - Logging
  - PKCS#11

- **API**
  - Discovery API
  - File API
  - SqliteDatabase API
  - SqliteServer API
  - SSI API
  - Sync API
  - Relay API
  - Volt API
  - Wire API

- **Utilities**
  - protoDbSync
  - sqliteServer
  - wireTransform

- ## FAQ
  - Questions
- ## Coming soon
  - Roadmap
- ☀🌙

Contents

Contents

WireAPI

Top

The Wire API allows clients to subscribe and publish to Volt wire resources.

PublishWire()

Establishes a client-streaming call to the wire resource.

Request: streaming PublishWireRequest
Response: streaming PublishWireResponse

SubscribeWire()

Establishes a bi-directional streaming call to the wire resource.

Although we're only really interested in receiving data from the wire, a bi-directional stream is required so that we can gracefully stop the subscription.

Request: streaming SubscribeWireRequest
Response: streaming SubscribeWireResponse

PublishWireRequest

Because the publish RPC is streaming, the policy will not be checked until the first message arrives. This can mean that the `PublishWire` call appears to succeed but then fails after the first attempt to publish a message.

To avoid this, clients can immediately send a message with the `wire_id` set and no `chunk` set. This will establish that the correct permissions are in place and fail fast if not.

Field	Type	Description
wire_id	string	Only necessary in the first payload.
chunk	bytes	The chunk of data to publish.
do_not_persist	bool	Whether to persist the chunk. This is only valid if the wire is configured to persist messages, i.e. the `volt:wire-persist` attribute is true.

PublishWireResponse

Field	Type	Description
status	Status	Details of any error that occurred on the call.

SubscribeWireRequest

The request must include one of the following fields.

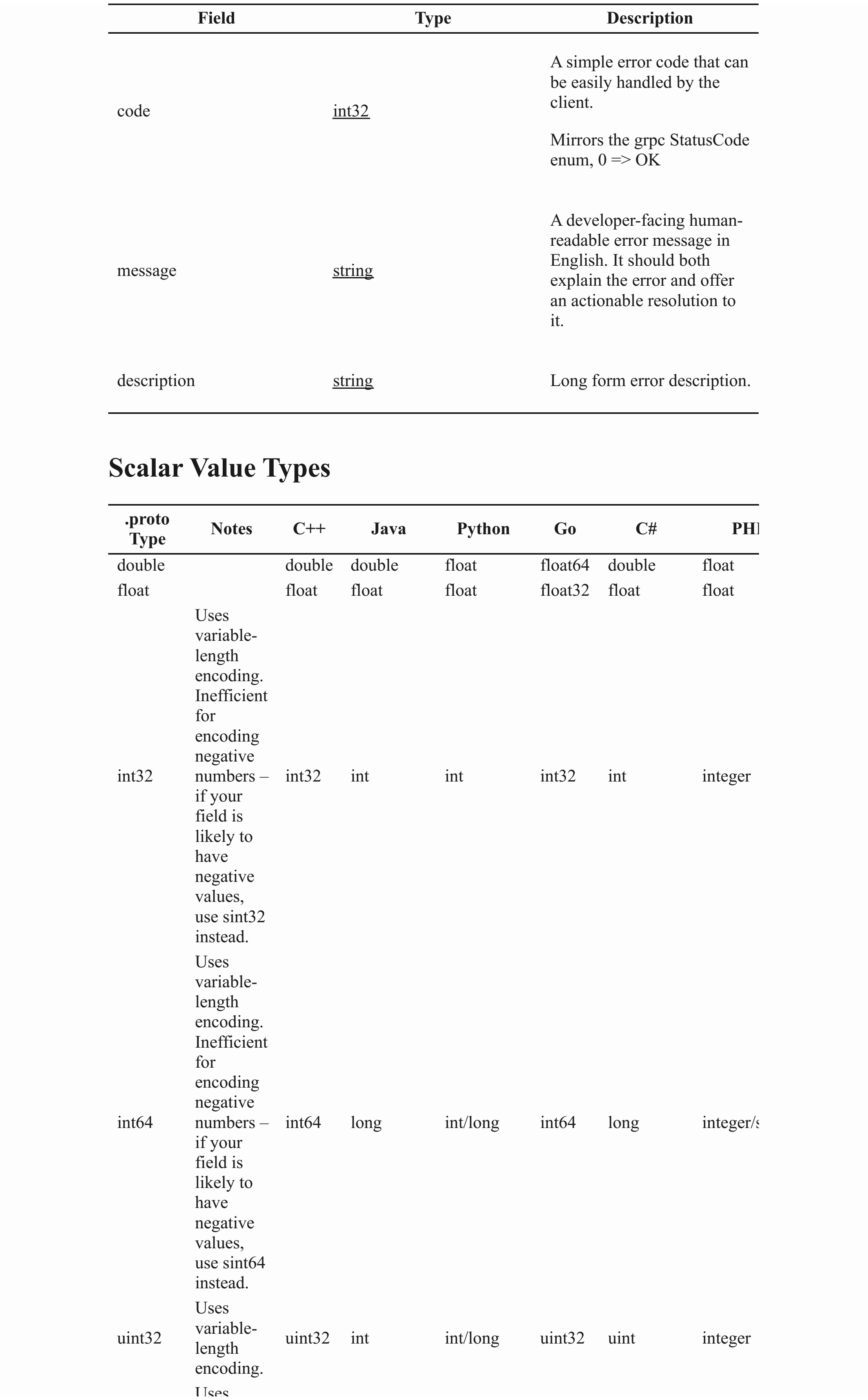
Field	Type	Description
wire_id	string	The wire id, only required for the first message.
stop	bool	Request to stop the subscription.

SubscribeWireResponse

One of the following fields will be present in the response.

Field	Type	Description
status	Status	Details of any error that occured on the call.
chunk	bytes	Data received from the wire.

Status



Field	Type	Description
code	int32	A simple error code that can be easily handled by the client. Mirrors the grpc StatusCode enum, 0 => OK
message	string	A developer-facing human-readable error message in English. It should both explain the error and offer an actionable resolution to it.
description	string	Long form error description.

Scalar Value Types

.proto Type	Notes	C++	Java	Python	Go	C#	PH
double		double	double	float	float64	double	float
float		float	float	float	float32	float	float
int32	Uses variable-length encoding. Inefficient for encoding negative numbers – if your field is likely to have negative values, use sint32 instead.	int32	int	int	int32	int	integer
int64	Uses variable-length encoding. Inefficient for encoding negative numbers – if your field is likely to have negative values, use sint64 instead.	int64	long	int/long	int64	long	integer/
uint32	Uses variable-length encoding.	uint32	int	int/long	uint32	uint	integer
	Uses						

.proto Type	Notes	C++	Java	Python	Go	C#	PH
uint64	Uses variable-length encoding.	uint64	long	int/long	uint64	ulong	integer/s
sint32	Uses variable-length encoding. Signed int value. These more efficiently encode negative numbers than regular int32s.	int32	int	int	int32	int	integer
sint64	Uses variable-length encoding. Signed int value. These more efficiently encode negative numbers than regular int64s.	int64	long	int/long	int64	long	integer/s
fixed32	Always four bytes. More efficient than uint32 if values are often greater than 2^28.	uint32	int	int	uint32	uint	integer
fixed64	Always eight bytes. More efficient than uint64 if values are often greater than 2^56.	uint64	long	int/long	uint64	ulong	integer/s
sfixed32	Always four bytes.	int32	int	int	int32	int	integer
sfixed64	Always eight bytes.	int64	long	int/long	int64	long	integer/s
bool		bool	boolean	boolean	bool	bool	boolean
	A string must always						

.proto Type	Notes	C++	Java	Python	Go	C#	PH
string	contain UTF-8 encoded or 7-bit ASCII text.	string	String	str/unicode	string	string	string
bytes	May contain any arbitrary sequence of bytes.	string	ByteString	str	[]byte	ByteString	string

Skip to Content

tdx Volt

putting you in charge

☀ ☾

Toggle sidebar

- ## Getting Started
  - Welcome
  - Quick Start
- ## Concepts
  - Fundamentals
  - Identity
  - DID registry
  - Verifiable credentials
  - Policy
  - Resource
  - File
  - Database
  - Wire
  - Relay
  - Key strategy
- ## How to...
  - Create a Volt
  - Start a Volt
  - Connect to a Volt
  - Upload files
  - Download files
  - Publish to wire
  - Subscribe to wire
  - Create a database
  - Execute SQL
  - Import data
  - Establish a connect stream
  - Approve authenticate request
- ## Clients
  - Command line

  - Native / C++
  - Web
  - NodeJS

- ## Reference

  - Best practice
  - Battery
  - Configuration
  - Connection
  - Connect stream
  - Logging
  - PKCS#11

- ## API

  - Discovery API
  - File API
  - SqliteDatabase API
  - SqliteServer API
  - SSI API
  - Sync API
  - Relay API
  - Volt API
  - Wire API

- ## Utilities

  - protoDbSync
  - sqliteServer
  - wireTransform

- ## FAQ

  - Questions

- ## Coming soon

  - Roadmap
- ☀☾

Contents

Contents

SqliteServerAPI

Top

The Sqlite Server API provides database management functions.

Use the VoltAPI DeleteResource function to delete a database.

CreateDatabase()

Create a new database resource.

Request: CreateDatabaseRequest
Response: CreateDatabaseResponse

CreateDatabaseRequest

Field	Type	Description
name	string	The name of the database to create.
create_in_parent_id	string	Optional - the id of the folder resource in which to create the database. If omitted, the callers home folder is used.
encrypted	bool	Set to encrypt the database. The encryption key will be generated by the server, the internal Volt server uses the root volt key.
read_audit	bool	Set to audit all SELECT database operations.
write_audit	bool	Set to audit all INSERT, UPDATE, and DELETE database operations.
discoverable	tdx.volt_api.volt.v1.DiscoveryMode	The discovery mode of the underlying Volt resource.
alias	string repeated	Alias(es) that can be used to refer to the database rather than the id. Each alias must be unique to the Volt, this is enforced by the API. No format restrictions are currently applied to alias, but this may change in future, for the time being it makes sense to stick to alphanumeric characters and '_' or '-'.
description	string	Optional description of the database.

CreateDatabaseResponse

Field	Type	Description
status	tdx.volt_api.volt.v1.Status	Any error message will be returned here.
resource	tdx.volt_api.volt.v1.Resource	The new database resource on success.

Status

Field	Type	Description
code	int32	A simple error code that can be easily handled by the client. Mirrors the grpc StatusCode enum, 0 => OK
message	string	A developer-facing human-readable error message in English. It should both explain the error and offer an actionable resolution to it.
description	string	Long form error description.

Access

Field	Type	Description
id	string	
resource_id	string	The resource being accessed.
resource_name	string	A human-readable short identifier of the resource.
resource_owner	string	The identity that owns the resource.
resource_kind	string repeated	The kind of resource.
identity_did	string	The identity attempting access.
credential_lookup	string	The JSON path array for looking up verifiable credentials.
identity_name	string	A human-readable short identifier of the subject.
identity_kind	string repeated	The kind of identity.
access	string	Requested access.
extra	string	Optional extra data.
decision	PolicyDecision	Assigned decision.
recursive	bool	
request_time	int64	Time at which the request was made.
decision_time	int64	Time at which the decision was taken.
request_count	uint32	Counter of number times this access was requested.

AttributeValue

Attribute value will be one of the following fields, depending on the data type.

Field	Type	Description
string	string	
integer	int64	
real	double	
boolean	bool	
bytes	bytes	

Identity

A Volt identity encompasses a Resource and a set of identity aliases.

Field	Type	Description
resource	Resource	
alias	IdentityAlias repeated	

IdentityAlias

Field	Type	Description
id	uint32	The alias id.
identity_did	string	The corresponding identity id.
alias	string	The actual alias, e.g. a common name or key fingerprint.
public_key	string	This will only be populated if alias_type == tdx:public-key
private_key	string	This will only be populated if alias_type == tdx:public-key, and the key is stored in the Volt.
alias_type	string	The alias type, for example public key, email, phone number etc.
issuer_id	string	The identity that issued this alias.
authenticate	PolicyDecision	Indicates if this alias has an authenticate policy decision assigned.
description	string	Optional description of this alias.

MethodDescription

Internal use only.

Field	Type	Description
path	string	
client_streaming	bool	
server_streaming	bool	

ProtoFile

Describes a single protobuf file for use in ServiceDescription.

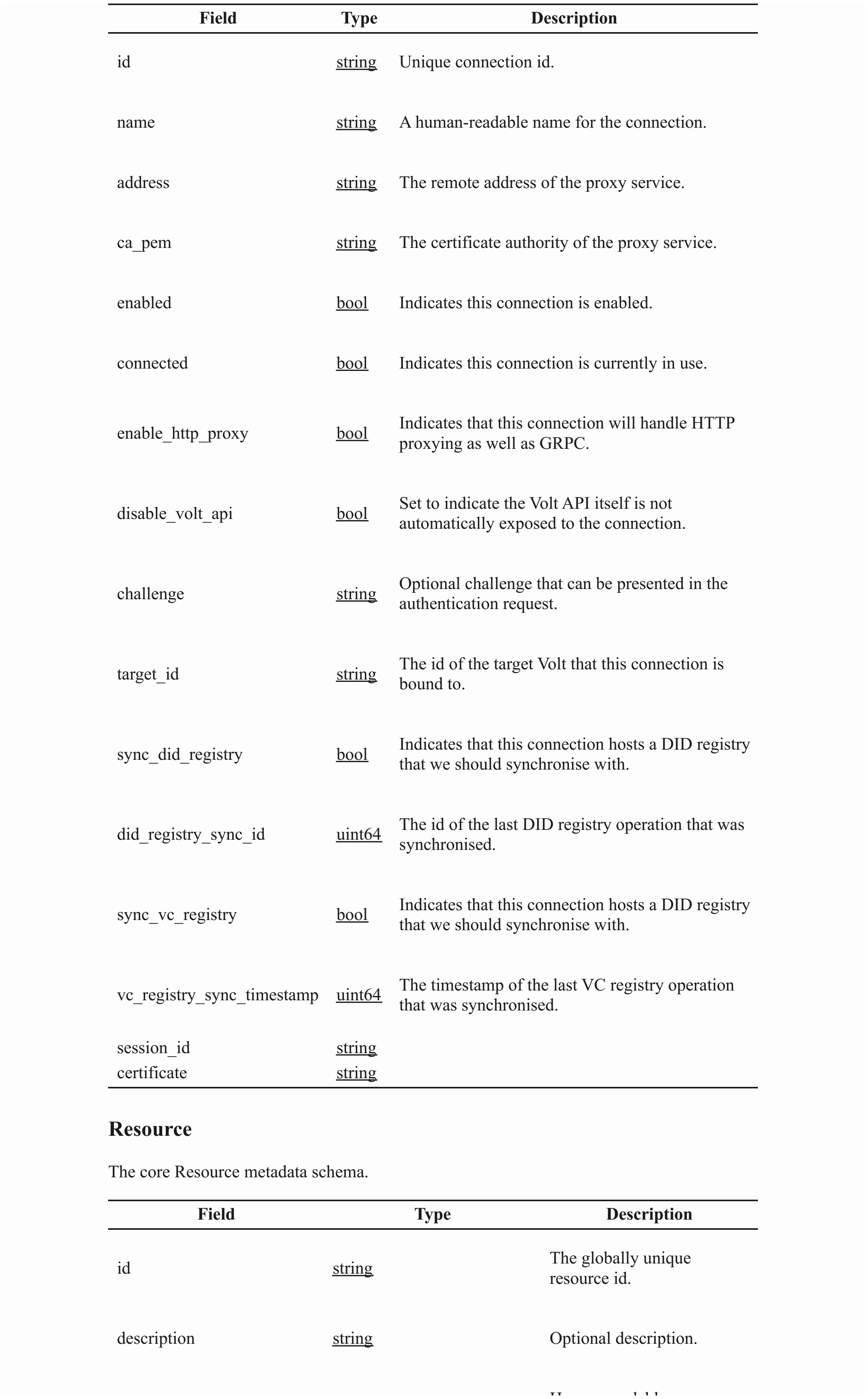
Field	Type	Description
file_path	string	The path name of the proto file, relative to the 'root' of the namespace, e.g. "tdx/volt_api/volt/v1/volt.proto".
protobuf	string	The actual protobuf file contents.
service_name	string repeated	Optional - the service(s) contained in this protobuf file, if omitted here they will be loaded dynamically from the protobuf.

ProxyConnection

Represents an outbound connection from a Volt to a remote service that will act as a proxy for that Volt.

This enables Volts to bypass firewall and NATs.

Example - connection from a Volt to a Relay Volt running on the public internet, such as tdxvolt.com



Field	Type	Description
id	string	Unique connection id.
name	string	A human-readable name for the connection.
address	string	The remote address of the proxy service.
ca_pem	string	The certificate authority of the proxy service.
enabled	bool	Indicates this connection is enabled.
connected	bool	Indicates this connection is currently in use.
enable_http_proxy	bool	Indicates that this connection will handle HTTP proxying as well as GRPC.
disable_volt_api	bool	Set to indicate the Volt API itself is not automatically exposed to the connection.
challenge	string	Optional challenge that can be presented in the authentication request.
target_id	string	The id of the target Volt that this connection is bound to.
sync_did_registry	bool	Indicates that this connection hosts a DID registry that we should synchronise with.
did_registry_sync_id	uint64	The id of the last DID registry operation that was synchronised.
sync_vc_registry	bool	Indicates that this connection hosts a DID registry that we should synchronise with.
vc_registry_sync_timestamp	uint64	The timestamp of the last VC registry operation that was synchronised.
session_id	string	
certificate	string	

Resource

The core Resource metadata schema.

Field	Type	Description
id	string	The globally unique resource id.
description	string	Optional description.

Field	Type	Description
name	string	Human-readable resource name.
share_mode	ShareMode	Not in use.
volt_id	string	The id of the Volt that hosts this resource.
service_description	ServiceDescription	Optional description of any services exposed by this resource.
attribute	ResourceAttribute repeated	Attributes assigned to the resource.
platform_version	Version	The version of the platform.
version	uint64	The resource version.
owner	string	The identity of the resource owner.
created	uint64	Creation timestamp, milliseconds since epoch.
modified	uint64	Last modification timestamp, milliseconds since epoch.
status	ResourceStatus	Not in use.
kind	string repeated	The taxonomy of the resource.
online_status	OnlineStatus	The online status. For most kinds of resource this indicates that the server hosting the resource is online, the exception being identity resources, in which case the status reflects whether or not the identity has a live connection. All built-in resources are hosted by the Volt itself and are therefore always online when the Volt is running. Resources hosted by external servers are online if the server itself is online and has registered the resource as online using \`setServiceStatus\`.

Field	Type	Description
size	uint64	The size of the resource store in bytes.
store	string	The path to the resource store.
alias	string repeated	Alias(es) that can be used to refer to the resource rather than the id. Each alias must be unique to the Volt, this is enforced by the API. No format restrictions are currently applied to alias, but this may change in future, for the time being it makes sense to stick to alphanumeric characters and '_' or '-'.
content_hash	string	The hash of the resource content contained in the store.
child	Resource repeated	Not yet supported.

ResourceAttribute

A resource attribute enables storing arbitrary data associated with a resource.

Field	Type	Description
id	uint32	
attribute_id	string	
resource_id	string	
data_type	AttributeDataType	
value	AttributeValue repeated	

ServiceDescription

Describes a Volt service.

Field	Type	Description
host_type	ServiceHostType	The configuration used by the host of this service.
host_client_id	string	The identity of the client that is exposing the service. For example, if a third party is exposing a database service via a Volt, it will first authenticate and obtain a client DID and credentials in order to be able to create service resource(s). Any resources that are owned by this client will be marked as online if the client itself is online, i.e. has a live connection to the Volt.

Field	Type	Description
		This will be empty if the service is a built-in Volt service.
host_service_id	string	The id of the resource that holds the protobuf definition for this resource. For example, if a third party is exposing a database service via a Volt, it will create a service resource that holds details of the protobuf methods exposed by the service. For built-in services, i.e. those hosted by the Volt, this will set to the Volt id.
host_address	string	The address of the grpc server hosting this service. Only relevant to grpc-hosted services.
host_ca_pem	string	The certificate authority (chain) that signed the service server certificate. This is only relevant to grpc-hosted services.
host_public_key	string	The public key of the service host, which is used to encrypt payloads. This may change as the service comes and goes online.
host_connection_id	string	The connection id currently used to host this service.
host_session_id	string	Internal use only.
discoverable	DiscoveryMode	The discovery mode.
ping_timestamp	int64	The ping timestamp of the server hosting this service.
proto_file	ProtoFile repeated	The protobuf definitions of the APIs exposed by this service.
service_api	string repeated	The fully qualified names of the protobuf services, for example tdx.volt_api.webcam.v1.WebcamControlAPI.
method	MethodDescription repeated	Internal use only.

Session

Field	Type	Description
id	string	
identity_did	string	
identity_name	string	
ip	string	
created	uint64	
modified	uint64	
expires	uint64	
credential	SessionCredential repeated	
origin	string	
status	SessionStatus	

SessionCredential

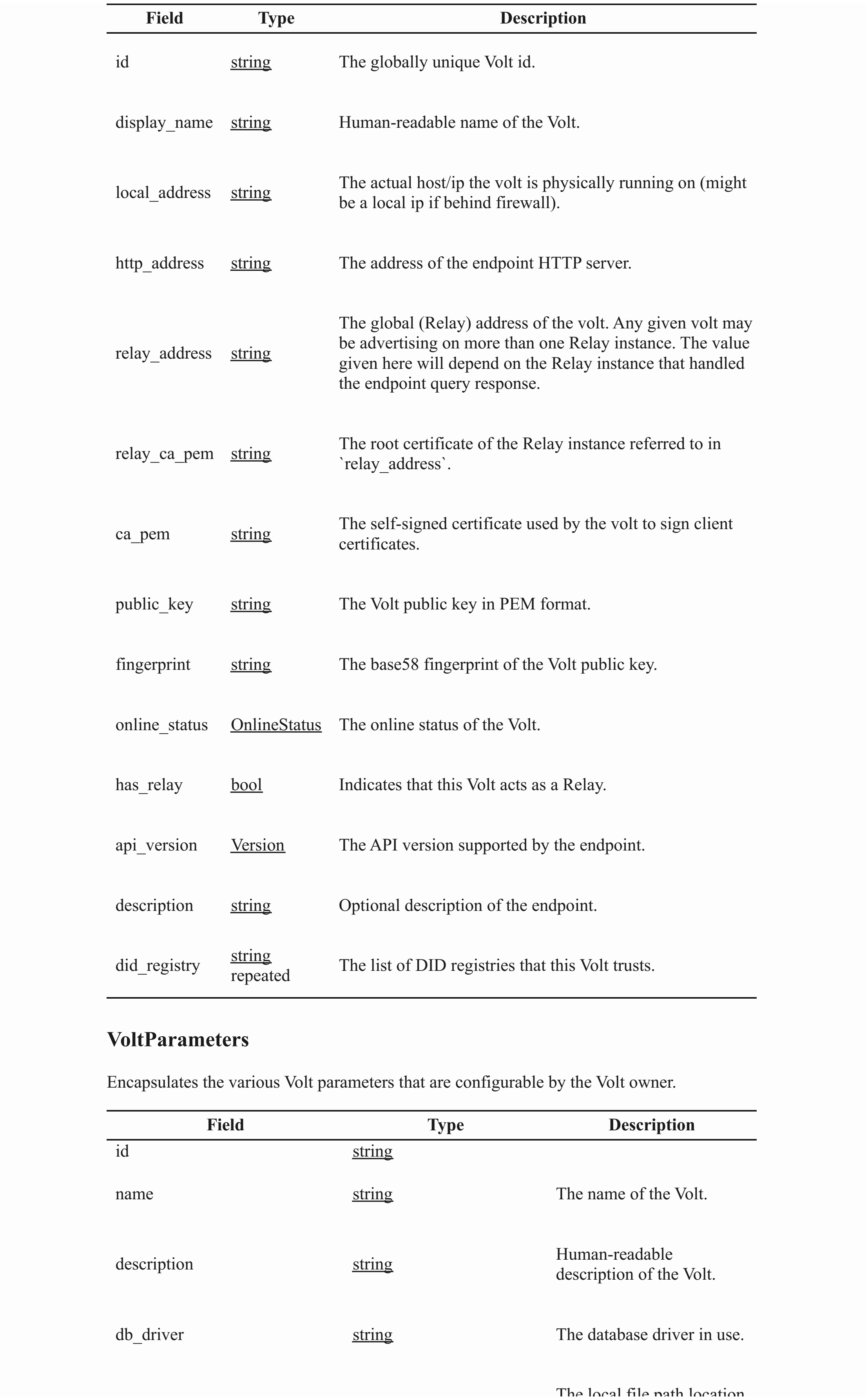
Field	Type	Description
id	uint32	The alias id.
session_id	string	The corresponding session id.
credential_type	string	The credential type, for example public key, verifiable credential, challenge etc.
description	string	Optional description of this credential.
vc_id	string	The id of the verifiable credential, if the credential type is volt:vc-claim.
vc_json	string	The verifiable credential in JSON format, if the credential type is volt:vc-claim.
vc_subject_id	string	The subject id extracted from the `vc_json` field.
vc_issuer_id	string	The issuer id extracted from the `vc_json` field.
vc_type	string	The comma-separated type(s) extracted from the `vc_json` field.
challenge	string	The challenge string, if the credential type is volt:challenge.
key_fingerprint	string	The key fingerprint, if the credential type is volt:public-key.
public_key	string	The PEM-encoded public key, if the credential type is volt:public-key.
private_key	string	Optional PEM-encoded private key, if the credential type is volt:public-key. Only used for ephemeral REST-base sessions created dynamically after OTP authentication.
extra	string	Type-specific extra data stored with the credential.
extra_2	string	More type-specific data stored with the credential.

Version

Using `major` and `minor` here upsets the GNU C Library, so we add a `version_` prefix.

Field	Type	Description
version_major	uint32	
version_minor	uint32	
version_patch	uint32	

VoltEndpoint



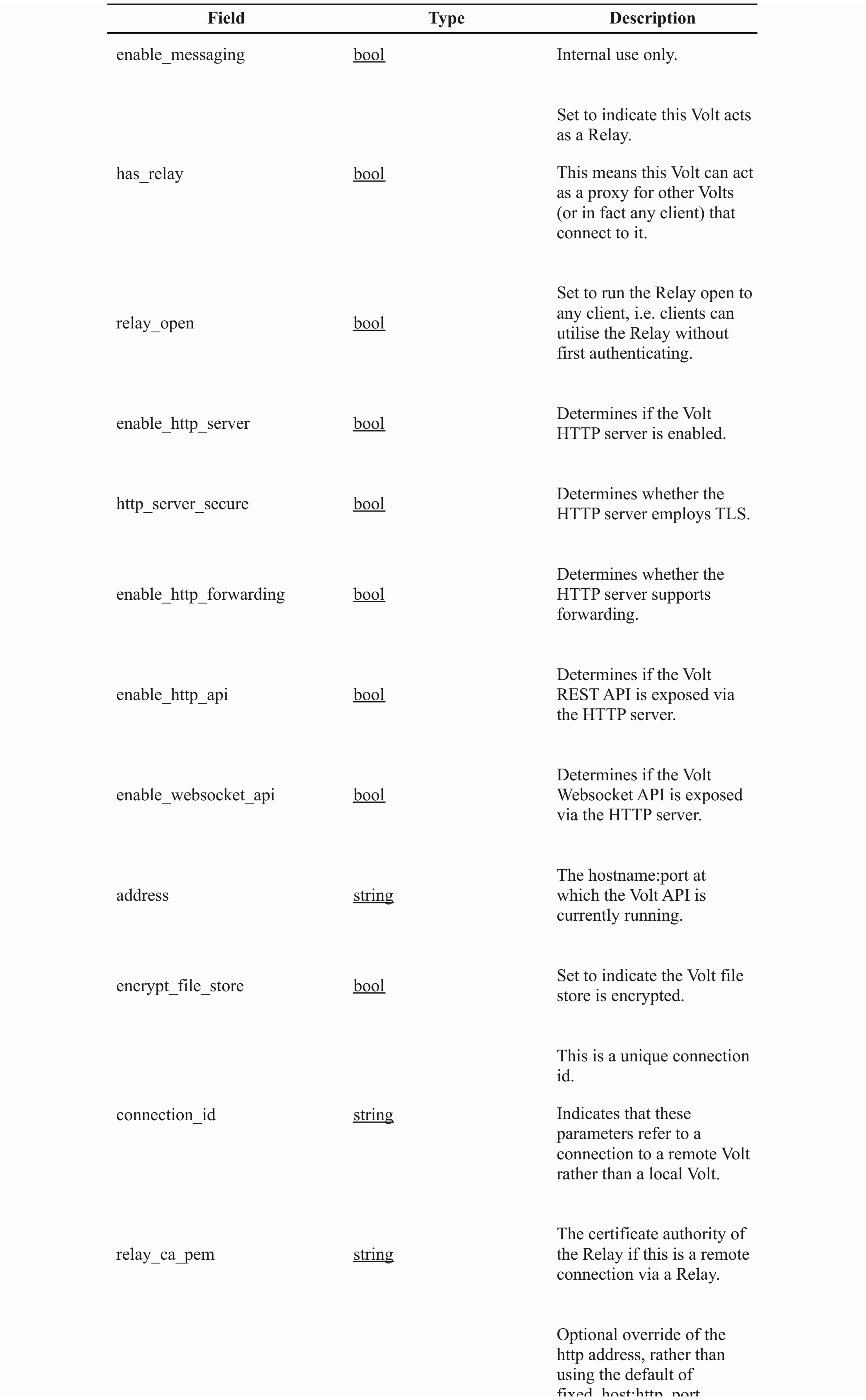
Field	Type	Description
id	string	The globally unique Volt id.
display_name	string	Human-readable name of the Volt.
local_address	string	The actual host/ip the volt is physically running on (might be a local ip if behind firewall).
http_address	string	The address of the endpoint HTTP server.
relay_address	string	The global (Relay) address of the volt. Any given volt may be advertising on more than one Relay instance. The value given here will depend on the Relay instance that handled the endpoint query response.
relay_ca_pem	string	The root certificate of the Relay instance referred to in \`relay_address\`.
ca_pem	string	The self-signed certificate used by the volt to sign client certificates.
public_key	string	The Volt public key in PEM format.
fingerprint	string	The base58 fingerprint of the Volt public key.
online_status	OnlineStatus	The online status of the Volt.
has_relay	bool	Indicates that this Volt acts as a Relay.
api_version	Version	The API version supported by the endpoint.
description	string	Optional description of the endpoint.
did_registry	string repeated	The list of DID registries that this Volt trusts.

VoltParameters

Encapsulates the various Volt parameters that are configurable by the Volt owner.

Field	Type	Description
id	string	
name	string	The name of the Volt.
description	string	Human-readable description of the Volt.
db_driver	string	The database driver in use.
		The local file path location

Field	Type	Description
location	string	The local file path location of the Volt storage.
key_strategy	string	The key strategy in use, this determines how the root key is stored.
key_id	string	The identifier for the key, the semantics depend on the key strategy in use.
ca_pem	string	The Volt certificate authority.
cert_pem	string	The Volt API server certificate.
fixed_host	string	Optional hostname of the Volt if using DNS or a static IP address, e.g. tdxvolt.com
grpc_port	int32	Port to use for hosting the Volt management service.
http_port	int32	Port to use for hosting the Volt grpc service.
http_key_path	string	The Volt http server key file path.
http_cert_path	string	The Volt http server certificate file path.
http_ca_path	string	The Volt http server certificate authority chain file path.
discoverable	bool	Indicates the Volt will be discoverable by clients using the discovery api.
authenticate_challenge	string	Optional challenge code that can be used aid in the process of authenticating clients.
require_authenticate_challenge	bool	Indicates that clients must present the correct challenge code in order to be able to authenticate.
confirm_stop	bool	Internal use only.
auto_start	bool	Internal use only.



Field	Type	Description
enable_messaging	bool	Internal use only.
has_relay	bool	Set to indicate this Volt acts as a Relay. This means this Volt can act as a proxy for other Volts (or in fact any client) that connect to it.
relay_open	bool	Set to run the Relay open to any client, i.e. clients can utilise the Relay without first authenticating.
enable_http_server	bool	Determines if the Volt HTTP server is enabled.
http_server_secure	bool	Determines whether the HTTP server employs TLS.
enable_http_forwarding	bool	Determines whether the HTTP server supports forwarding.
enable_http_api	bool	Determines if the Volt REST API is exposed via the HTTP server.
enable_websocket_api	bool	Determines if the Volt Websocket API is exposed via the HTTP server.
address	string	The hostname:port at which the Volt API is currently running.
encrypt_file_store	bool	Set to indicate the Volt file store is encrypted.
connection_id	string	This is a unique connection id. Indicates that these parameters refer to a connection to a remote Volt rather than a local Volt.
relay_ca_pem	string	The certificate authority of the Relay if this is a remote connection via a Relay.
		Optional override of the http address, rather than using the default of fixed_host:http_port

Field	Type	Description
http_address_override	string	fixed_host:http_port. This is useful if the Volt is behind a firewall or NAT, and the http server is listening on a different port from 80 or 443 but this is hidden by the proxy. For example, if the `fixed_host` is `coreid.com` and http server is listening on 2115, but the proxy is forwarding 443 to 2115, then the http_address_override would be set to `https://coreid.com`.
alias	string	An optional alias that can be used to refer to the Volt rather than the `id` field. This alias must be unique within the scope of the Battery in which the Volt is stored.
version	Version	The runtime version this Volt is running.
approve_on_challenge	bool	If set, indicates that any client that provides the correct challenge during authentication will automatically be approved to access the Volt.
approve_on_did	bool	If set, indicates that any client that proves ownership of a DID known to the Volt will automatically be approved to access the Volt.
enable_did_registry	bool	If set, indicates that clients can register DIDs with this Volt.
did_registry	string repeated	Zero or more URLs of trusted peer DID registries.
enable_outbound_smtp	bool	If set, enables outbound SMTP.
outbound_smtp_host	string	The SMTP host to use for sending emails.
outbound_smtp_port	uint32	The SMTP port to use for sending emails

sending emails.

Field	Type	Description
outbound_smtp_user	string	The SMTP username to use for sending emails.
outbound_smtp_password	string	The SMTP password to use for sending emails.
enable_anonymous_create	bool	If set, enables sessions that authenticate using credentials rather than a DID to create resources in the 'anonymous' system folder.
catch_all_auth_decision	PolicyDecision	The decision to apply to all authentication requests that do not match any other policy. The default is PROMPT.
enable_policy_cache	bool	If set, enables caching of policy decisions.
enable_terminal	bool	If set, enables the terminal API.
start_time	uint64	The time at which the Volt was started.

AttributeDataType

Attribute data types.

Name	Number	Description
ATTRIBUTE_DATA_TYPE_UNKNOWN	0	
ATTRIBUTE_DATA_TYPE_STRING	1	
ATTRIBUTE_DATA_TYPE_INTEGER	2	
ATTRIBUTE_DATA_TYPE_REAL	3	
ATTRIBUTE_DATA_TYPE_BOOLEAN	4	
ATTRIBUTE_DATA_TYPE_BYTES	5	
ATTRIBUTE_DATA_TYPE_IDENTITY	100	
ATTRIBUTE_DATA_TYPE_RESOURCE	101	

DiscoveryMode

Name	Number	Description
DISCOVERY_MODE_UNKNOWN	0	
DISCOVERY_MODE_TRUSTED	1	Only local identities with explicit policy PERMIT can discover.
DISCOVERY_MODE_PUBLIC	2	Any bound local identity can discover.
DISCOVERY_MODE_TRUSTED_GLOBAL	3	Only identities with explicit policy PERMIT can discover, and the service will be available to local and non-local (Relayed) clients.
DISCOVERY_MODE_PUBLIC_GLOBAL	4	Any bound identity can discover, and the service will be available to local and non-local (Relayed) clients.

OnlineStatus

Name	Number	Description
ONLINE_STATUS_UNKNOWN	0	
ONLINE_STATUS_ONLINE	1	
ONLINE_STATUS_OFFLINE	2	

PolicyDecision

@todo currently this must align with AuthorisationDecision enum in policy library, but some of the values are irrelevant outside of the public API so we need a public-facing enum and some translation.

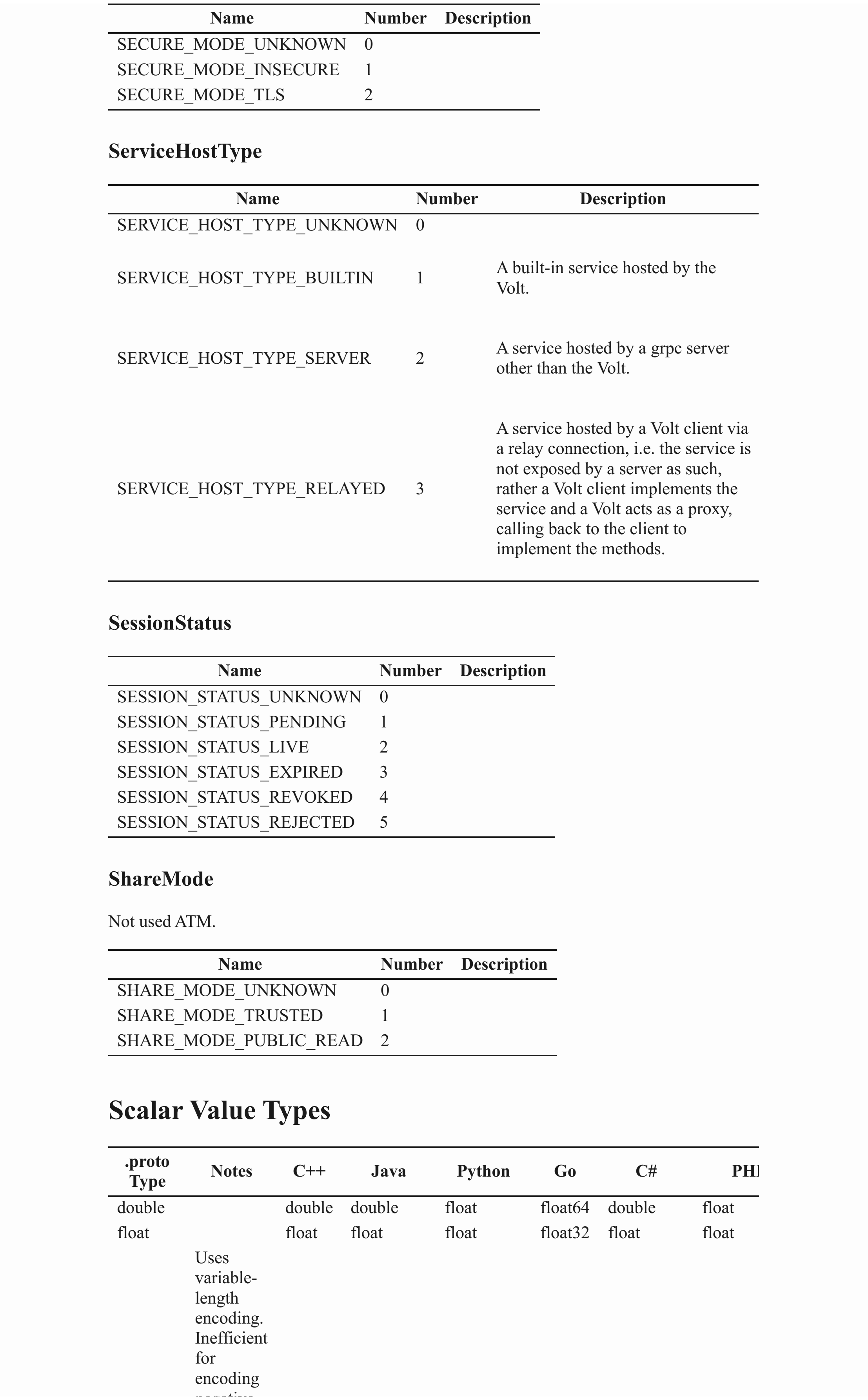
Name	Number	Description
POLICY_DECISION_UNKNOWN	0	
POLICY_DECISION_PROMPT	1	
POLICY_DECISION_PERMIT	2	
POLICY_DECISION_DENY	3	
POLICY_DECISION_INDETERMINATE	4	
POLICY_DECISION_NOT_APPLICABLE	5	
POLICY_DECISION_APPLICABLE	6	
POLICY_DECISION_PENDING	7	

ResourceStatus

Not used ATM.

Name	Number	Description
RESOURCE_STATUS_UNKNOWN	0	
RESOURCE_STATUS_LIVE	1	
RESOURCE_STATUS_INACTIVE	2	
RESOURCE_STATUS_DELETED	999	

SecureMode



Name	Number	Description
SECURE_MODE_UNKNOWN	0	
SECURE_MODE_INSECURE	1	
SECURE_MODE_TLS	2	

ServiceHostType

Name	Number	Description
SERVICE_HOST_TYPE_UNKNOWN	0	
SERVICE_HOST_TYPE_BUILTIN	1	A built-in service hosted by the Volt.
SERVICE_HOST_TYPE_SERVER	2	A service hosted by a grpc server other than the Volt.
SERVICE_HOST_TYPE_RELAYED	3	A service hosted by a Volt client via a relay connection, i.e. the service is not exposed by a server as such, rather a Volt client implements the service and a Volt acts as a proxy, calling back to the client to implement the methods.

SessionStatus

Name	Number	Description
SESSION_STATUS_UNKNOWN	0	
SESSION_STATUS_PENDING	1	
SESSION_STATUS_LIVE	2	
SESSION_STATUS_EXPIRED	3	
SESSION_STATUS_REVOKED	4	
SESSION_STATUS_REJECTED	5	

ShareMode

Not used ATM.

Name	Number	Description
SHARE_MODE_UNKNOWN	0	
SHARE_MODE_TRUSTED	1	
SHARE_MODE_PUBLIC_READ	2	

Scalar Value Types

.proto Type	Notes	C++	Java	Python	Go	C#	PH
double		double	double	float	float64	double	float
float		float	float	float	float32	float	float
	Uses variable-length encoding. Inefficient for encoding						

proto Type	Notes	C++	Java	Python	Go	C#	PH
int32	negative numbers – if your field is likely to have negative values, use sint32 instead.	int32	int	int	int32	int	integer
int64	Uses variable-length encoding. Inefficient for encoding negative numbers – if your field is likely to have negative values, use sint64 instead.	int64	long	int/long	int64	long	integer/s
uint32	Uses variable-length encoding.	uint32	int	int/long	uint32	uint	integer
uint64	Uses variable-length encoding.	uint64	long	int/long	uint64	ulong	integer/s
sint32	Uses variable-length encoding. Signed int value. These more efficiently encode negative numbers than regular int32s.	int32	int	int	int32	int	integer
sint64	Uses variable-length encoding. Signed int value. These more efficiently encode negative numbers than regular int64s.	int64	long	int/long	int64	long	integer/s
	Always four bytes.						

.proto Type	Notes	C++	Java	Python	Go	C#	PHP
fixed32	More efficient than uint32 if values are often greater than 2^28.	uint32	int	int	uint32	uint	integer
fixed64	Always eight bytes. More efficient than uint64 if values are often greater than 2^56.	uint64	long	int/long	uint64	ulong	integer/s
sfixed32	Always four bytes.	int32	int	int	int32	int	integer
sfixed64	Always eight bytes.	int64	long	int/long	int64	long	integer/s
bool		bool	boolean	boolean	bool	bool	boolean
string	A string must always contain UTF-8 encoded or 7-bit ASCII text.	string	String	str/unicode	string	string	string
bytes	May contain any arbitrary sequence of bytes.	string	ByteString	str	[]byte	ByteString	string

Skip to Content

tdx Volt

putting you in charge

☀ ☾

Toggle sidebar

- ## Getting Started
  - Welcome
  - Quick Start
- ## Concepts
  - Fundamentals

- Identity
- DID registry
- Verifiable credentials
- Policy
- Resource
- File
- Database
- Wire
- Relay
- Key strategy

How to...

- Create a Volt
- Start a Volt
- Connect to a Volt
- Upload files
- Download files
- Publish to wire
- Subscribe to wire
- Create a database
- Execute SQL
- Import data
- Establish a connect stream
- Approve authenticate request

Clients

- Command line
- Native / C++
- Web
- NodeJS

Reference

- Best practice
- Battery
- Configuration
- Connection
- Connect stream
- Logging
- PKCS#11

API

- Discovery API
- File API
- SqliteDatabase API
- SqliteServer API
- SSI API
- Sync API
- Relay API
- Volt API
- Wire API

Utilities

- protoDbSync
- sqliteServer
- wireTransform

- ## FAQ
  - Questions
- ## Coming soon
  - Roadmap
- ☀☾

Contents

Contents

VoltAPI

Top

The top-level volt management service.

With the exception of `Authenticate`, all RPCs on this service must submit either a valid client certificate signed by the volt CA, or a signed JWT.

A client certificate can be obtained from the `Authenticate` method.

All methods include a `tdx.volt_api.volt.v1.Status` in the response. If the status contains a non-OK error code the client should assume the remainder of the message is invalid, unless otherwise indicated in the response documentation.

Authenticate()

Issues a request to authenticate on the volt.

All clients must successfully authenticate in order to gain any kind of access.

A client certificate is optional for this RPC, if omitted a JWT must be provided in the call metadata.

If the client plans to register one or more services with the Volt, it should use the 'host' field of the request so that the returned certificate has the appropriate SAN extension in place. Note that if the client IP address changes it will be necessary to re-authenticate the client and obtain a new certificate.

If the authenticate decision is 'permit', the response contains various details that should be persisted by the client, including the unique identifier assigned by the Volt and a signed client certificate along with the Volt CA certificate.

Request: AuthenticateRequest
Response: AuthenticateResponse

CanAccessResource()

Determine if an identity can perform a specific action on a resource.

Third party services can use this to interrogate the Volt policy and determine if a client has permission to perform a certain action on a resource.

Request: CanAccessResourceRequest
Response: CanAccessResourceResponse

CheckCompatibility()

Request: CheckCompatibilityRequest
Response: CheckCompatibilityResponse

Connect()

Creates a long-lived, bi-directional connection to the Volt.

The connection stream serves several purposes, including remote invocations via a Relay, Volt event notifications, pings and service registration management.

A connection stream is required in order for a client to be able to register services with the Volt.

When the stream is closed, any services registered on it will be set to offline.

Request: streaming ConnectRequest
Response: streaming ConnectResponse

CopyResource()

Copy a resource from one folder to another.

The resource metadata, attributes and store are copied.

The shares attributed to the resource are **not** currently copied.

n.b. Copy resources between Volts is not supported by this API. This is achieved by creating an API instance for the source and target Volts, getting the resource from the source and creating it on the target.

Request: CopyResourceRequest
Response: CopyResourceResponse

DeleteAccess()

Remove a custom access rule, such as a file share.

Request: DeleteAccessRequest
Response: DeleteAccessResponse

DeleteResource()

Delete a resource from this volt.

Request: DeleteResourceRequest
Response: DeleteResourceResponse

DiscoverServices()

Discover services running on this volt.

Lookup is done via the exposed serviceAPI, e.g. tdx.volt_api.data.v1.SqliteServerAPI.

Request: DiscoverServicesRequest
Response: DiscoverServicesResponse

GetAccess()

Get access rule details.

Only rules in which the authenticated identity participates will be retrieved, in addition to any rules that target a resource that is owned by the authenticated identity.

If the authenticated identity is the Volt root, all rules will be retrieved that match the criteria.

Request: GetAccessRequest
Response: GetAccessResponse

GetIdentities()

Retrieve identities matching the given criteria.

Only identities that the authenticated identity has read access to will be retrieved.

Request: GetIdentitiesRequest
Response: GetIdentitiesResponse

GetIdentity()

Get details of a specific identity.

The identity will only be retrieved if the authenticated identity has read access to it.

Request: GetIdentityRequest
Response: GetIdentityResponse

GetOneTimeToken()

Gets a one-time token that can be used as a temporary authentication token, for example create an file download link that expires after a certain time.

Request: GetOneTimeTokenRequest
Response: GetOneTimeTokenResponse

GetParameters()

Retrieve the Volt parameters.

This is a privileged API call and requires Volt root access.

Request: GetParametersRequest
Response: GetParametersResponse

GetPolicy()

Retrieve the active Volt policy.

This is a privileged API call and requires Volt root access.

Request: GetPolicyRequest
Response: GetPolicyResponse

GetResource()

Get resource from this volt.

Request: GetResourceRequest
Response: GetResourceResponse

GetResources()

Get resources from this volt.

Request: GetResourcesRequest
Response: GetResourcesResponse

GetResourceAncestors()

Get ancestors of a resource.

Request: GetResourceAncestorsRequest
Response: GetResourceAncestorsResponse

GetResourceDescendants()

Get descendants of a resource.

Request: GetResourceDescendantsRequest
Response: GetResourceDescendantsResponse

GetSessions()

Get sessions.

Request: GetSessionsRequest
Response: GetSessionsResponse

Invoke()

Invoke a method.

This is primarily for use by Relay connections when proxying invocations.

Request: streaming InvokeRequest
Response: streaming InvokeResponse

MoveResource()

Move a resource from one folder to another.

Request: MoveResourceRequest
Response: MoveResourceResponse

RequestAccess()

Request access to a resource.

The subject of the access is assumed to be the identity of the currently authenticated peer.

Request: RequestAccessRequest
Response: RequestAccessResponse

SaveAccess()

Create or update an access rule.

Request: SaveAccessRequest
Response: SaveAccessResponse

SaveHttpFileServer()

Create or update a static file HTTP server.

Request: SaveResourceRequest
Response: SaveResourceResponse

SaveHttpApiServer()

Create or update an HTTP REST API server.

Request: SaveResourceRequest
Response: SaveResourceResponse

SaveIdentity()

Create or update an identity.

Request: SaveIdentityRequest

Response: SaveIdentityResponse

SaveMirroredLink()

Create or update a mirrored link resource.

Request: SaveResourceRequest
Response: SaveResourceResponse

SaveParameters()

Update Volt parameters.

This is a privileged call that requires Volt root access.

Request: SaveParametersRequest
Response: SaveParametersResponse

SaveResource()

Create or update resource in this volt.

Request: SaveResourceRequest
Response: SaveResourceResponse

SaveSymbolicLink()

Create or update a symbolic link resource.

Request: SaveResourceRequest
Response: SaveResourceResponse

SaveSession()

Save a session.

Request: SaveSessionRequest
Response: SaveSessionResponse

SetAccessRequestDecision()

Set Volt access request decision.

This is a privileged call that requires Volt root access.

Request: SetAccessRequestDecisionRequest
Response: SetAccessRequestDecisionResponse

SetPolicy()

Request: SetPolicyRequest
Response: SetPolicyResponse

SetServiceStatus()

Set the status of a Volt service.

Request: SetServiceStatusRequest
Response: SetServiceStatusResponse

Shutdown()

Used to shutdown remote Volts.

This is a privileged call that requires Volt root access.

Request: ShutdownRequest
Response: ShutdownResponse

SignVerify()

Sign or verify an arbitrary message using the Volt key.

Request: SignVerifyRequest
Response: SignVerifyResponse

AuthenticateRequest

Describes a request to authenticate on a Volt.

Present one of the `public_key`, `did_public_key`, `did`, or `did_document` fields.

Field	Type	Description
public_key	string	The client public key in PEM format. The Volt will create a session that is bound to this public key.
did_public_key	string	The client public key - must be in PEM format. If the client DID is lost or unknown for some reason, providing the public key here will allow the Volt to match it with the previously registered DID. Note this is only valid when a DID has previously been registered using this public key.
did	string	An existing DID owned by the client. The JWT presented with the authenticate call must be signed by the private key corresponding to this DID.
did_document	string	If the client doesn't have an existing DID, a DID document can be provided here. The Volt will register the DID on behalf of the client. The JWT presented with the authenticate call must be signed by the private key corresponding to this document.
did_document_signature	string	A base64-encoded signature of the DID document, only required if `did_document` is provided above.

is provided above.

Field	Type	Description
client_name	string	A human-readable name of the entity requesting to authenticate.
challenge	string	The volt challenge code, signed by the private key component of the `public_key` field above, and base64 encoded. This is optional.
host	string	The host name to add as a SAN to the issued certificate. This is optional, if you don't intend to host services with the certificate this can be omitted.
verifiable_presentation	VerifiablePresentation repeated	Optional verifiable credentials describing the client.
purge_aliases	bool	Reserved for internal use.
session_name	string	Optional additional name to differentiate between multiple sessions for a given client.

AuthenticateResponse

Field	Type	Description
status	Status	Details of any error that occurred on the call.
session_id	string	
identity_did	string	The identity id assigned to this authentication.
cert	string	A certificate issued by the volt CA, binding the request public key to the identity. Only valid for PERMIT authenticate decisions.
chain	string	The volt CA chain. This is used by the client in subsequent API calls to secure the connection.
decision	PolicyDecision	The authenticate decision.
request_time	int64	Reserved for internal use.
decision_time	int64	Reserved for internal use.

CanAccessResourceRequest

Field	Type	Description
token	string	The subject of the access request. This is optional, and if omitted will default to the authenticated account.
cert	string	The subject of the access request. This is optional, and if omitted will default to the authenticated account.
resource_id	string	The resource in question.
access	string	The type of access that is required.

CanAccessResourceResponse

Field	Type	Description
status	Status	Details of any error that occurred on the call.
identity_did	string	The identity subject that the access relates to.
resource_id	string	The resource the access relates to.
access	string	The access type.
decision	PolicyDecision	The policy decision for this access request.

CheckCompatibilityRequest

Field	Type	Description
version	Version	The client platform API version.

CheckCompatibilityResponse

Field	Type	Description
status	Status	Details of any error that occurred on the call.
version	Version	The target Volt API version.

ConnectAcknowledge

Field	Type	Description
connection_id	string	A unique identifier for this connection.
timestamp	uint64	The current time on the Volt.
ping_interval	uint32	The interval at which the target will send ping requests to the client.

ConnectAuthRequest

Field	Type	Description
session	Session	
context	string	
challenge	string	
timestamp	uint64	

ConnectDIDRegistryUpdate

Field	Type	Description
update	DIDRegistryUpdate	

ConnectEvent

Field	Type	Description
connect_auth_request	ConnectAuthRequest	An authentication request event.
connect_resource	ConnectResource	A resource event notification, such as updated or deleted.
did_registry_update	ConnectDIDRegistryUpdate	A DID registry entry update.

ConnectGoodbye

Field	Type	Description
status	Status	Details of any error that occurred on the call.
ended	bool	Set if the connection was ended gracefully, as opposed to errored.

ConnectHello

All fields in this message are optional, send an empty message if necessary.

Field	Type	Description
address	string	Optionally specify the grpc server address of the client. Only used if the client is a Volt (or service).
online_services	bool	Set to automatically make **all** services owned by the calling identity online.
subscribe_resource_events	bool	Set to receive notification of resource events.
relay_id	string	Set this if you are connecting to a relay, i.e. the Volt you are sending to this message is a relay. This indicates that you are happy to receive method invocations from clients of the Relay. This will usually be set to the `id` of your Volt, but in theory any client could receive remote invocation requests in this way.
relay_name	string	A friendly name to present to Relay clients.
relay_description	string	

Field	Type	Description
relay_ca_pem	string	The certificate authority to present to Relay clients.
relay_discoverable	bool	Set to indicate the connection is discoverable to other Relay clients.
relay_http_address	string	Optionally specify the address of the HTTP server to use for HTTP proxying. If set, a relay will forward HTTP requests to this address from the subdomain that matches the client's DID.
subscribe_auth_requests	bool	Set to receive notification of authentication requests.
accept_invocation	bool	Set to indicate the connection will accept method invocation requests.
subscribe_did_registry_updates	bool	Set to subscribe to DID registry updates from the peer.
ping_interval	uint32	The interval at which the client will send ping requests to the target.
timestamp	uint64	The current time on the client.
did_registry	string repeated	Optionally specify the DID registries supported.

ConnectPing

A connection ping message - intentionally empty.

ConnectRelay

Field	Type	Description
connected	bool	Indicates the Relay connection status.

ConnectRequest

One of the following payloads will be present in a given ConnectRequest message.

Field	Type	Description
hello	ConnectHello	A ConnectHello message is the first message a client sends to the target upon successfully starting the call.
goodbye	ConnectGoodbye	Indicates the client is closing the connection.
ping	ConnectPing	Clients should periodically send ping requests to the target to confirm the stream is still live.
invoke_response	InvokeResponse	Send invocation responses back to the caller. n.b. The 'request' and 'response' semantics are inverted with remote invocations because the Relayed connection is established by a request **from** the 'target' Volt to the Relay. Hence any invocation on behalf of a client of the Relay involves sending a **response** back down the Relay connection to the 'target' Volt.
http_response	HttpResponse	Send HTTP response back to the originating request. n.b. The 'request' and 'response' semantics are inverted with HTTP proxying for the same reason as `invoke_response` above.

ConnectResource

Field	Type	Description
event	ConnectResourceEvent	The type of resource event that has occurred.
resource	Resource	Details of the resource.

ConnectResponse

One of the following payloads will be present in a given ConnectResponse message.

Field	Type	Description
status	Status	Details of any error that occurred on the call.
acknowledge	ConnectAcknowledge	Server response to initial handshake.
goodbye	ConnectGoodbye	Indicates the server is ending the connection.
evt	ConnectEvent	Notifies clients of various events on the Volt.
ping	ConnectPing	Periodic ping response.
invoke_request	InvokeRequest	Send an invocation request to a remote target. n.b. The 'request' and 'response' semantics are inverted for remote invocations because the Relayed connection is established by a request **from** the 'target' Volt to the Relay. Hence any invocation on behalf of a client of the Relay involves sending a **response** back down the Relay connection to the 'target' Volt.
http_request	HttpRequest	Send an HTTP request to a remote target. n.b. The 'request' and 'response' semantics are inverted for HTTP requests for the same reason as `invoke_request` above.

CopyResourceRequest

Field	Type	Description
resource_id	string	The id of the resource to copy.
to_resource_id	string	The id of the resource to receive the new copy.
mode	CopyResourceMode	The copy mode to use.
recursive	bool	Indicates if all descendants of the resource should be copied too. Only relevant for COPY_RESOURCE_MODE_COPY mode.
from_resource_id	string	The parent resource to link from, only relevant for COPY_RESOURCE_MODE_LINK mode.

CopyResourceResponse

Field	Type	Description
status	Status	Details of any error that occurred on the call.

DeleteAccessRequest

Field	Type	Description
id	string	The id of the access rule to delete.

DeleteAccessResponse

Field	Type	Description
status	Status	Details of any error that occurred on the call.

DeleteResourceRequest

Field	Type	Description
resource_id	string	The resource to delete.
recursive	bool	Set to indicate all descendant resources should also be deleted. If this is not set and the resource has descendants, the call will fail.
parent_id	string	Set to attempt to unlink the resource from this parent resource, rather than completely delete it. The resource will be removed as a descendant from the `parent_id` resource. If the resource is has more than one parent, it will not be removed from those other parents. Note that if the resource's only parent is `parent_id` it will be removed from that parent and deleted as normal.

DeleteResourceResponse

Field	Type	Description
status	Status	Details of any error that occurred on the call.

DiscoverServicesRequest

Field	Type	Description
service_api	string repeated	List the service APIs that should be discovered. The response will include services that match **any of** the terms given. Use of '*' to indicate wildcards is supported.
include_offline	bool	Set to indicate that offline services should be included in the response.
include_attributes	bool	Set to include the service attributes in the response.
include_protobuf	bool	Set to include the service protobuf in the response.

DiscoverServicesResponse

Field	Type	Description
status	Status	Details of any error that occurred on the call.
resource	Resource repeated	The services that were discovered.

GetAccessRequest

Field	Type	Description
resource_id	string	The resource id that is the target of the access rule. If omitted, all resources will be considered.
identity_did	string	The identity id that is the subject of the access rule. If omitted, all identities will be considered.
access	string	The type of access to retrieve, if omitted all access will be considered.
decision	PolicyDecision	The type of decision, if omitted all decisions will be considered.

GetAccessResponse

Field	Type	Description
status	Status	Details of any error that occurred on the call.
access	Access repeated	The access rules that match the criteria.

GetIdentitiesRequest

Field	Type	Description
name	string	Optional name of the identity. Use '*' to perform a wildcard search. If omitted all identities will be retrieved.
alias	IdentityAlias	The identity alias criteria, if omitted all identities will be considered.

GetIdentitiesResponse

Field	Type	Description
status	Status	Details of any error that occurred on the call.
identity	Identity repeated	The identity list that matched the criteria.

GetIdentityRequest

One of `identity_did` or `fingerprint` must be populated.

Field	Type	Description
identity_did	string	The id of the identity to retrieve.
fingerprint	string	The public key fingerprint of the identity to retrieve.

GetIdentityResponse

Field	Type	Description
status	Status	Details of any error that occurred on the call.
identity	Identity	The identity details.

GetOneTimeTokenRequest

Field	Type	Description
ttl	int32	Optional token TTL, in seconds. Default is 10 seconds.

GetOneTimeTokenResponse

Field	Type	Description
status	Status	Details of any error that occurred on the call.
token	string	The one-time token.

GetParametersRequest

This an empty message.

GetParametersResponse

Field	Type	Description
status	Status	Details of any error that occurred on the call.
parameters	VoltParameters	The retrieved Volt parameters.

GetPolicyRequest

This an empty message.

Field	Type	Description
custom_policy	bool	Set to only retrieve the custom policy document.

GetPolicyResponse

Field	Type	Description
status	Status	Details of any error that occurred on the call.
policy	string	The live policy in JSON format.

GetResourceAncestorsRequest

Field	Type	Description
resource_id	string	The resource id whose ancestors will be retrieved.
include_resource_id	bool	Optional - if set the resource_id resource will be included in the set of resources returned.
depth	int32	Optional - restrict the depth search, e.g. for immediate parents depth = 1
ancestor_kind	string	Optional - only match ancestors of the given kind.
ancestor_id	string	Optional - can be used to determine if a resource is an ancestor.
include_attributes	bool	Set to include the service attributes in the response.
include_protobuf	bool	Set to include the service protobuf in the response.

GetResourceAncestorsResponse

Field	Type	Description
status	Status	Details of any error that occurred on the call.
ancestor	Resource repeated	The retrieved ancestors.

GetResourceDescendantsRequest

Field	Type	Description
resource_id	string	The resource id whose descendants will be retrieved.
include_resource_id	bool	Optional - if set, the fully populated `resource_id` resource (i.e. the parent) will be included in the set of resources returned.
depth	int32	Optional - restrict the depth search, e.g. for immediate children depth = 1
descendant_kind	string repeated	Optional - only match descendants of the given kind. If multiple kinds are given, resources matching **any of** the kinds will be included.
descendant_id	string	Optional - can be used to determine if a resource is a descendant.
parent_id	string	Optional - restrict to descendants of a given parent, for use in multi-parent hierarchies.
name	string	Restrict to a specific named resource.
modified_since	uint64	Only retrieve resources that have been modified after the given timestamp. Default is 0, which means all resources will be returned.
include_attributes	bool	Set to include the service attributes in the response.
include_protobuf	bool	Set to include the service protobuf in the response.

GetResourceDescendantsResponse

Field	Type	Description
status	Status	Details of any error that occurred on the call.
descendant	Resource repeated	The retrieved descendants.

GetResourceRequest

Field	Type	Description
resource_id	string	The id of the resource to retrieve.
include_attributes	bool	Set to include the resource attributes in the response.
include_protobuf	bool	Set to include service description protobuf in the response, if applicable.

GetResourceResponse

Field	Type	Description
status	Status	Details of any error that occurred on the call.
resource	Resource	The retrieved resource metadata.

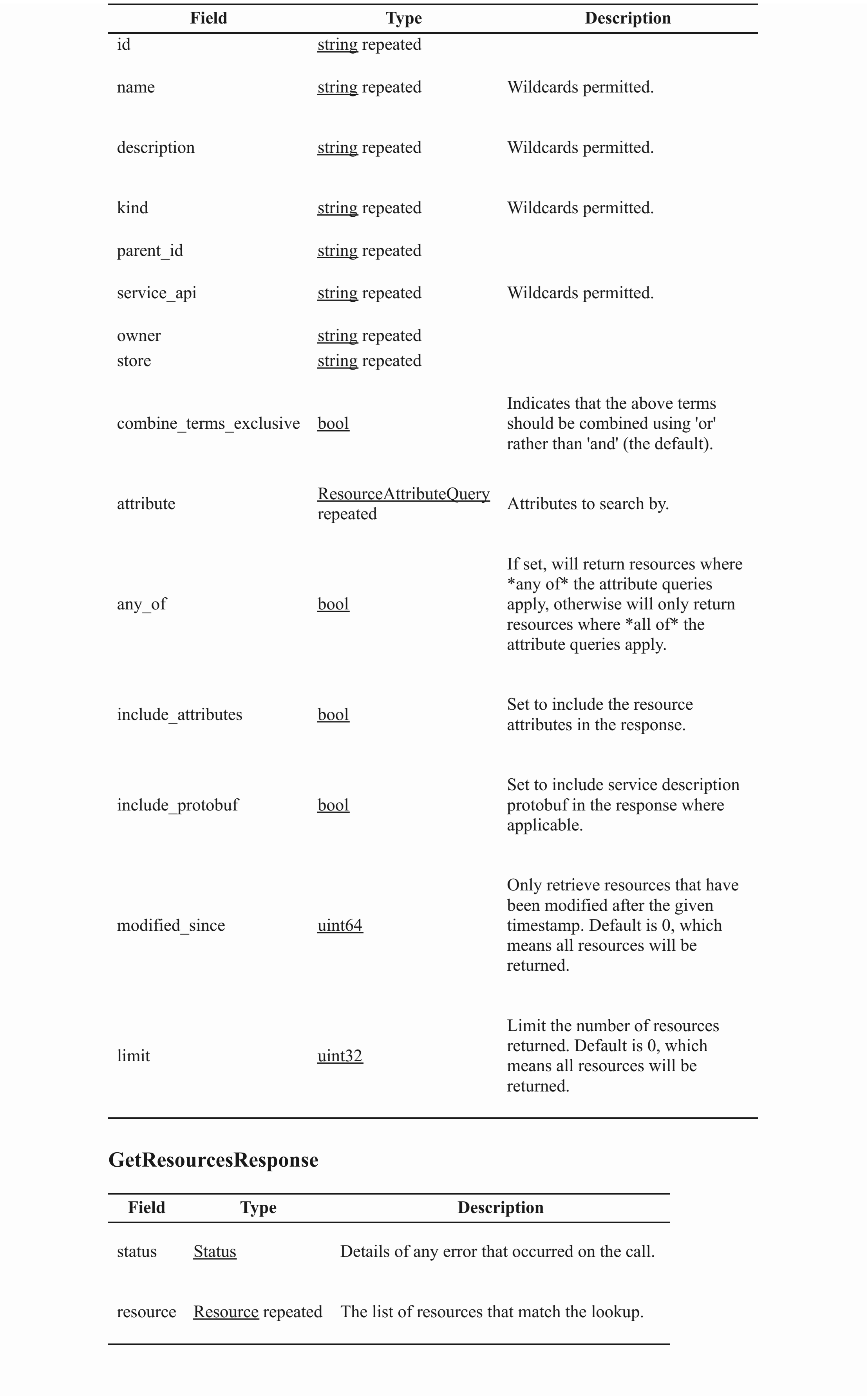
GetResourcesRequest

By default, the lookup is performed by combining the criteria below in the form:

(id = id[0] or id = id[1]) and (name = name[0] or name = name[1]) etc...

To combine using 'or' rather than 'and', set the `combine_terms_exclusive` flag.

Also see note below regarding attributes.



Field	Type	Description
id	string repeated	
name	string repeated	Wildcards permitted.
description	string repeated	Wildcards permitted.
kind	string repeated	Wildcards permitted.
parent_id	string repeated	
service_api	string repeated	Wildcards permitted.
owner	string repeated	
store	string repeated	
combine_terms_exclusive	bool	Indicates that the above terms should be combined using 'or' rather than 'and' (the default).
attribute	ResourceAttributeQuery repeated	Attributes to search by.
any_of	bool	If set, will return resources where *any of* the attribute queries apply, otherwise will only return resources where *all of* the attribute queries apply.
include_attributes	bool	Set to include the resource attributes in the response.
include_protobuf	bool	Set to include service description protobuf in the response where applicable.
modified_since	uint64	Only retrieve resources that have been modified after the given timestamp. Default is 0, which means all resources will be returned.
limit	uint32	Limit the number of resources returned. Default is 0, which means all resources will be returned.

GetResourcesResponse

Field	Type	Description
status	Status	Details of any error that occurred on the call.
resource	Resource repeated	The list of resources that match the lookup.

GetSessionsRequest

Field	Type	Description
id	string	
identity_did	string	
identity_name	string	
status	SessionStatus	

GetSessionsResponse

Field	Type	Description
status	Status	Details of any error that occurred on the call.
session	Session repeated	

InvokeRequest

Field	Type	Description
invoke_id	uint64	Client-assigned identifier for the request. Will be used to match responses and any subsequent requests.
token	string	Optional client token to use for the invocation. This is for use by the websocket proxy and is only necessary on the first request of the rpc.
target_did	string repeated	The DID of the target at each hop of the path to the service. This is used by Relays to route the request.
iv	bytes	The initialisation vector for the request payload encryption. This should be a random 16 byte value, that is different for each request.
payload	bytes	A serialised and encrypted instance of \`RemoteResponse\` in pure binary format.
json_payload	bytes	A serialised and encrypted instance of \`RemoteResponse\` as serialised JSON.
client_end	bool	Indicates the client has ended the invocation.
hop_index	uint32	Reserved for internal use.
target_service_id	string	Reserved for internal use.

InvokeRequestKeyExchange

Field	Type	Description
nonce	bytes	
encryption_key	bytes	
signature	bytes	

InvokeResponse

Field	Type	Description
invoke_id	uint64	The invocation id to match the originating request.
key_exchange	InvokeRequestKeyExchange	
iv	bytes	The initialisation vector for the response payload encryption. This will be a random 16 byte value, that is different for each response.
payload	bytes	A serialised and encrypted instance of `RemoteRequest` in pure binary format.
json_payload	bytes	A serialised and encrypted instance of `RemoteRequest` as serialised JSON.
status	Status	Details of any error that occurred on the call.
server_end	bool	Indicates the server has ended the invocation.

MoveResourceRequest

Field	Type	Description
resource_id	string	The id of the resource to move.
from_resource_id	string	The resource the parent folder to move the resource from.
to_resource_id	string	The target folder to move the resource into.

MoveResourceResponse

Field	Type	Description
status	Status	Details of any error that occurred on the call.

RequestAccessRequest

Field	Type	Description
resource_id	string	The target resource id.
access	string	The type of access requested.

RequestAccessResponse

Field	Type	Description
status	Status	Details of any error that occurred on the call.
resource_id	string	The resource being accessed.
identity_did	string	The identity attempting access.
access	string	Requested access.
decision	PolicyDecision	Assigned decision.
request_time	int64	Time at which the request was made.
decision_time	int64	Time at which the decision was taken.
request_count	int32	Counter of number times this access was requested.

ResourceAttributeQuery

Field	Type	Description
attribute_id	string	
data_type	AttributeDataType	
value	AttributeValue	

SaveAccessRequest

Field	Type	Description
access	Access	Omit `id` if creating new access.

SaveAccessResponse

Field	Type	Description
status	Status	Details of any error that occurred on the call.

SaveIdentityRequest

Field	Type	Description
identity	Identity	Details of the identity to save.
create	bool	Set to indicate this is a new identity.
delete_alias	IdentityAlias repeated	The list of aliases that should be removed. For example, this allows a simple form of key rotation whereby an existing public key alias is replaced by a new one while still maintaining the same root identity id.
create_in_parent_id	string	Reserved for system use.
purge_aliases	bool	Set to indicate the identity aliases should be purged before saving the identity. If the `identity` field contains aliases they will be saved after the purge. If the `identity` field does not contain aliases this effectively deletes all aliases for this identity. This allows you to selectively update aliases if required, i.e. don't set this flag and include a single alias in the update.
did_document	string	
did_update_signature	string	The signature of the identity did document, if present.

SaveIdentityResponse

Field	Type	Description
status	Status	Details of any error that occurred on the call.
identity	Identity	The updated identity details.

SaveParametersRequest

Field	Type	Description
parameters	VoltParameters	The updated parameters.
key_passphrase	string	The current root key passphrase. Only necessary if changes are being made to the Volt key.
new_key_passphrase	string	The new root key passphrase. Only necessary if changes are being made to the Volt key.

SaveParametersResponse

Field	Type	Description
status	Status	Details of any errors that occurred on the call.
parameters	VoltParameters	The updated Volt parameters.
reconnect	bool	If set, the client will need to reconnect (usually because the key has changed).

SaveResourceRequest

Field	Type	Description
resource	Resource	Details of the resource to save.
create	bool	Set to indicate this is a new resource.
create_in_parent_id	string	The id of the folder resource in which a new resource should be created. If omitted, the home folder of the currently authenticated identity will be used.
purge_attributes	bool	Set to indicate the resource attributes should be purged before saving the resource. If the `resource` field contains attributes they will be saved after the purge. If the `resource` field does not contain attributes this effectively deletes all attributes for this resource. This allows you to selectively update attributes if required, i.e. don't set this flag and include a single attribute in the update.

SaveResourceResponse

Field	Type	Description
status	Status	Details of any error that occurred on the call.
resource	Resource	The updated resource.

SaveSessionRequest

Field	Type	Description
session	Session	

SaveSessionResponse

Field	Type	Description
status	Status	Details of any error that occurred on the call.
session	Session	

SetAccessRequestDecisionRequest

Field	Type	Description
id	string	The id of the access request.
decision	PolicyDecision	The decision to save against the access request.

SetAccessRequestDecisionResponse

Field	Type	Description
status	Status	Details of any error that occurred on the call.

SetPolicyRequest

Field	Type	Description
custom_policy	string	

SetPolicyResponse

Field	Type	Description
status	Status	Details of any error that occurred on the call.

SetServiceStatusRequest

Field	Type	Description
service	Resource	The service description details.

SetServiceStatusResponse

Field	Type	Description
status	Status	Details of any error that occurred on the call.
resource	Resource	The updated service resource details.

ShutdownRequest

This message is emtpy.

ShutdownResponse

Field	Type	Description
status	Status	Details of any error that occurred on the call.

SignVerifyRequest

Field	Type	Description
verify	bool	Set to indicate this is a request to verify rather than sign.
encode	bool	Set to indicate the signature should be base64 encoded in the response. Only valid when signing.
message	string	The message to sign. Only valid when signing.
digest_raw	bytes	The digest in raw binary form. Only valid if verifying.
digest_encoded	string	The digest encoded using base64. Only valid if verifying.

SignVerifyResponse

Note that if verification was successful the response will be empty (there is no error and no digest is returned).

Field	Type	Description
status	Status	Details of any error that occurred on the call.
digest	bytes	The signature in raw binary form.
digest_encoded	string	The signature encoded using base64.

ConnectResourceEvent

Name	Number	Description
CONNECT_RESOURCE_EVENT_UNKNOWN	0	
CONNECT_RESOURCE_EVENT_CREATE	1	
CONNECT_RESOURCE_EVENT_UPDATE	2	
CONNECT_RESOURCE_EVENT_DELETE	3	
CONNECT_RESOURCE_EVENT_CREATE_CHILD	4	
CONNECT_RESOURCE_EVENT_DELETE_CHILD	5	
CONNECT_RESOURCE_EVENT_DATABASE_WRITE	6	

CopyResourceMode

Name	Number	Description
COPY_RESOURCE_MODE_UNKNOWN	0	
COPY_RESOURCE_MODE_COPY	1	
COPY_RESOURCE_MODE_LINK	2	

Status

Field	Type	Description
code	int32	A simple error code that can be easily handled by the client. Mirrors the grpc StatusCode enum, 0 => OK
message	string	A developer-facing human-readable error message in English. It should both explain the error and offer an actionable resolution to it.
description	string	Long form error description.

Access

Field	Type	Description
id	string	
resource_id	string	The resource being accessed.
resource_name	string	A human-readable short identifier of the resource.
resource_owner	string	The identity that owns the resource.
resource_kind	string repeated	The kind of resource.
identity_did	string	The identity attempting access.
credential_lookup	string	The JSON path array for looking up verifiable credentials.
identity_name	string	A human-readable short identifier of the subject.
identity_kind	string repeated	The kind of identity.
access	string	Requested access.
extra	string	Optional extra data.
decision	PolicyDecision	Assigned decision.
recursive	bool	
request_time	int64	Time at which the request was made.
decision_time	int64	Time at which the decision was taken.
request_count	uint32	Counter of number times this access was requested.

AttributeValue

Attribute value will be one of the following fields, depending on the data type.

Field	Type	Description
string	string	
integer	int64	
real	double	
boolean	bool	
bytes	bytes	

Identity

A Volt identity encompasses a Resource and a set of identity aliases.

Field	Type	Description
resource	Resource	
alias	IdentityAlias repeated	

IdentityAlias

Field	Type	Description
id	uint32	The alias id.
identity_did	string	The corresponding identity id.
alias	string	The actual alias, e.g. a common name or key fingerprint.
public_key	string	This will only be populated if alias_type == tdx:public-key
private_key	string	This will only be populated if alias_type == tdx:public-key, and the key is stored in the Volt.
alias_type	string	The alias type, for example public key, email, phone number etc.
issuer_id	string	The identity that issued this alias.
authenticate	PolicyDecision	Indicates if this alias has an authenticate policy decision assigned.
description	string	Optional description of this alias.

MethodDescription

Internal use only.

Field	Type	Description
path	string	
client_streaming	bool	
server_streaming	bool	

ProtoFile

Describes a single protobuf file for use in ServiceDescription.

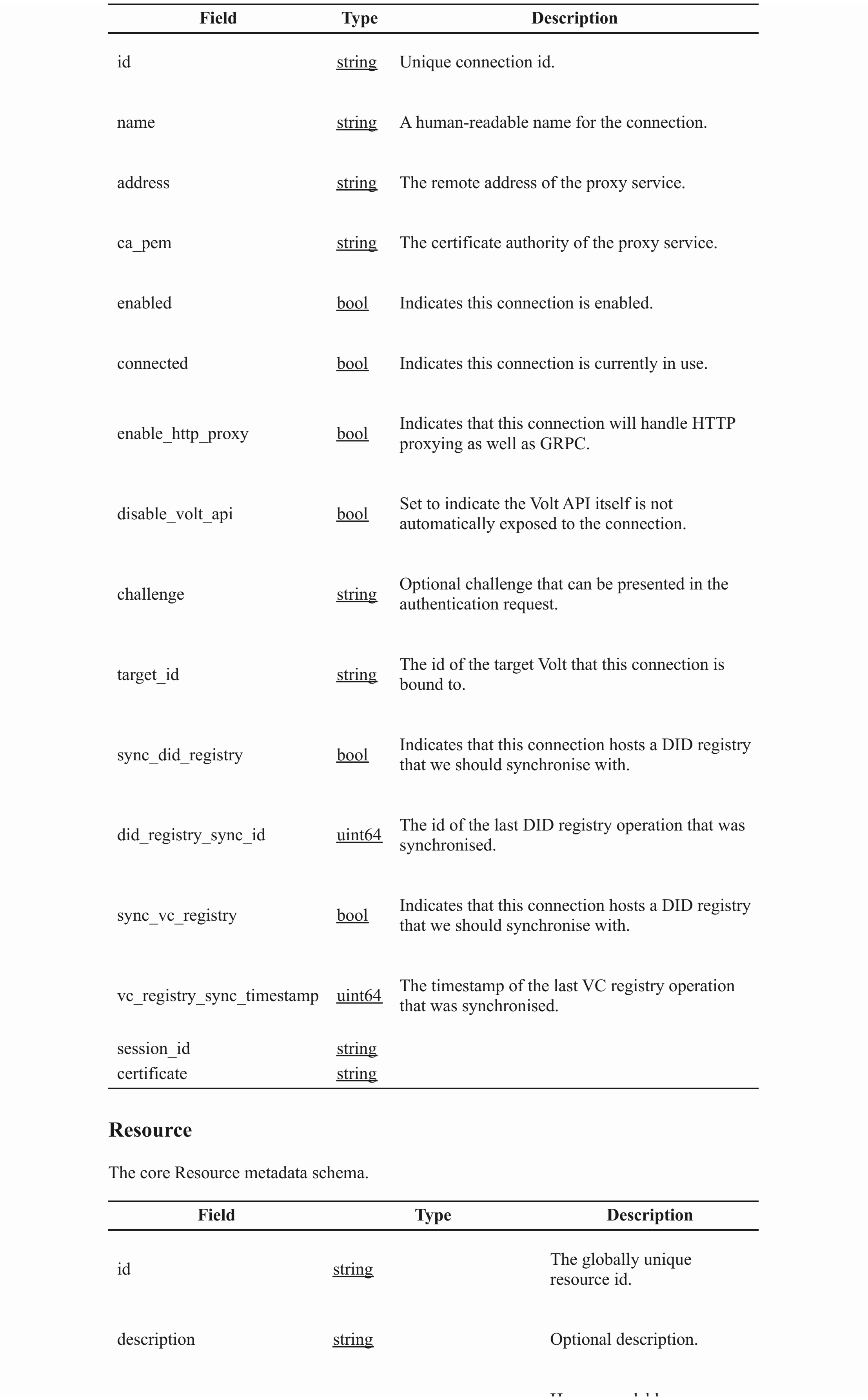
Field	Type	Description
file_path	string	The path name of the proto file, relative to the 'root' of the namespace, e.g. "tdx/volt_api/volt/v1/volt.proto".
protobuf	string	The actual protobuf file contents.
service_name	string repeated	Optional - the service(s) contained in this protobuf file, if omitted here they will be loaded dynamically from the protobuf.

ProxyConnection

Represents an outbound connection from a Volt to a remote service that will act as a proxy for that Volt.

This enables Volts to bypass firewall and NATs.

Example - connection from a Volt to a Relay Volt running on the public internet, such as tdxvolt.com



Field	Type	Description
id	string	Unique connection id.
name	string	A human-readable name for the connection.
address	string	The remote address of the proxy service.
ca_pem	string	The certificate authority of the proxy service.
enabled	bool	Indicates this connection is enabled.
connected	bool	Indicates this connection is currently in use.
enable_http_proxy	bool	Indicates that this connection will handle HTTP proxying as well as GRPC.
disable_volt_api	bool	Set to indicate the Volt API itself is not automatically exposed to the connection.
challenge	string	Optional challenge that can be presented in the authentication request.
target_id	string	The id of the target Volt that this connection is bound to.
sync_did_registry	bool	Indicates that this connection hosts a DID registry that we should synchronise with.
did_registry_sync_id	uint64	The id of the last DID registry operation that was synchronised.
sync_vc_registry	bool	Indicates that this connection hosts a DID registry that we should synchronise with.
vc_registry_sync_timestamp	uint64	The timestamp of the last VC registry operation that was synchronised.
session_id	string	
certificate	string	

Resource

The core Resource metadata schema.

Field	Type	Description
id	string	The globally unique resource id.
description	string	Optional description.

Field	Type	Description
name	string	Human-readable resource name.
share_mode	ShareMode	Not in use.
volt_id	string	The id of the Volt that hosts this resource.
service_description	ServiceDescription	Optional description of any services exposed by this resource.
attribute	ResourceAttribute repeated	Attributes assigned to the resource.
platform_version	Version	The version of the platform.
version	uint64	The resource version.
owner	string	The identity of the resource owner.
created	uint64	Creation timestamp, milliseconds since epoch.
modified	uint64	Last modification timestamp, milliseconds since epoch.
status	ResourceStatus	Not in use.
kind	string repeated	The taxonomy of the resource.
online_status	OnlineStatus	The online status. For most kinds of resource this indicates that the server hosting the resource is online, the exception being identity resources, in which case the status reflects whether or not the identity has a live connection. All built-in resources are hosted by the Volt itself and are therefore always online when the Volt is running. Resources hosted by external servers are online if the server itself is online and has registered the resource as online using `setServiceStatus`.

Field	Type	Description
size	uint64	The size of the resource store in bytes.
store	string	The path to the resource store.
alias	string repeated	Alias(es) that can be used to refer to the resource rather than the id. Each alias must be unique to the Volt, this is enforced by the API. No format restrictions are currently applied to alias, but this may change in future, for the time being it makes sense to stick to alphanumeric characters and '_' or '-'.
content_hash	string	The hash of the resource content contained in the store.
child	Resource repeated	Not yet supported.

ResourceAttribute

A resource attribute enables storing arbitrary data associated with a resource.

Field	Type	Description
id	uint32	
attribute_id	string	
resource_id	string	
data_type	AttributeDataType	
value	AttributeValue repeated	

ServiceDescription

Describes a Volt service.

Field	Type	Description
host_type	ServiceHostType	The configuration used by the host of this service.
host_client_id	string	The identity of the client that is exposing the service. For example, if a third party is exposing a database service via a Volt, it will first authenticate and obtain a client DID and credentials in order to be able to create service resource(s). Any resources that are owned by this client will be marked as online if the client itself is online, i.e. has a live connection to the Volt.

Field	Type	Description
		This will be empty if the service is a built-in Volt service.
host_service_id	string	The id of the resource that holds the protobuf definition for this resource. For example, if a third party is exposing a database service via a Volt, it will create a service resource that holds details of the protobuf methods exposed by the service. For built-in services, i.e. those hosted by the Volt, this will set to the Volt id.
host_address	string	The address of the grpc server hosting this service. Only relevant to grpc-hosted services.
host_ca_pem	string	The certificate authority (chain) that signed the service server certificate. This is only relevant to grpc-hosted services.
host_public_key	string	The public key of the service host, which is used to encrypt payloads. This may change as the service comes and goes online.
host_connection_id	string	The connection id currently used to host this service.
host_session_id	string	Internal use only.
discoverable	DiscoveryMode	The discovery mode.
ping_timestamp	int64	The ping timestamp of the server hosting this service.
proto_file	ProtoFile repeated	The protobuf definitions of the APIs exposed by this service.
service_api	string repeated	The fully qualified names of the protobuf services, for example tdx.volt_api.webcam.v1.WebcamControlAPI.
method	MethodDescription repeated	Internal use only.

Session

Field	Type	Description
id	string	
identity_did	string	
identity_name	string	
ip	string	
created	uint64	
modified	uint64	
expires	uint64	
credential	SessionCredential repeated	
origin	string	
status	SessionStatus	

SessionCredential

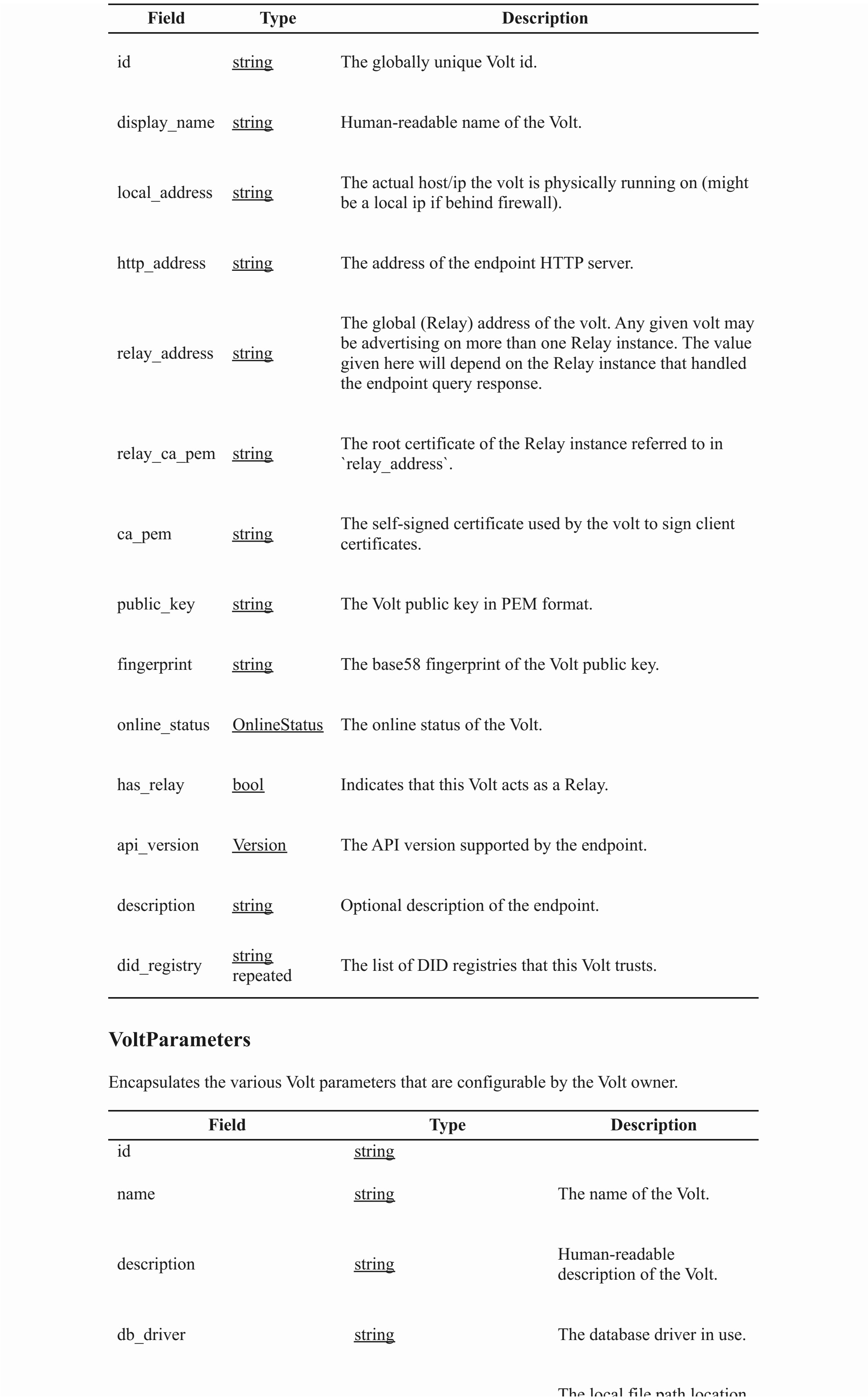
Field	Type	Description
id	uint32	The alias id.
session_id	string	The corresponding session id.
credential_type	string	The credential type, for example public key, verifiable credential, challenge etc.
description	string	Optional description of this credential.
vc_id	string	The id of the verifiable credential, if the credential type is volt:vc-claim.
vc_json	string	The verifiable credential in JSON format, if the credential type is volt:vc-claim.
vc_subject_id	string	The subject id extracted from the `vc_json` field.
vc_issuer_id	string	The issuer id extracted from the `vc_json` field.
vc_type	string	The comma-separated type(s) extracted from the `vc_json` field.
challenge	string	The challenge string, if the credential type is volt:challenge.
key_fingerprint	string	The key fingerprint, if the credential type is volt:public-key.
public_key	string	The PEM-encoded public key, if the credential type is volt:public-key.
private_key	string	Optional PEM-encoded private key, if the credential type is volt:public-key. Only used for ephemeral REST-base sessions created dynamically after OTP authentication.
extra	string	Type-specific extra data stored with the credential.
extra_2	string	More type-specific data stored with the credential.

Version

Using `major` and `minor` here upsets the GNU C Library, so we add a `version_` prefix.

Field	Type	Description
version_major	uint32	
version_minor	uint32	
version_patch	uint32	

VoltEndpoint



Field	Type	Description
id	string	The globally unique Volt id.
display_name	string	Human-readable name of the Volt.
local_address	string	The actual host/ip the volt is physically running on (might be a local ip if behind firewall).
http_address	string	The address of the endpoint HTTP server.
relay_address	string	The global (Relay) address of the volt. Any given volt may be advertising on more than one Relay instance. The value given here will depend on the Relay instance that handled the endpoint query response.
relay_ca_pem	string	The root certificate of the Relay instance referred to in `relay_address`.
ca_pem	string	The self-signed certificate used by the volt to sign client certificates.
public_key	string	The Volt public key in PEM format.
fingerprint	string	The base58 fingerprint of the Volt public key.
online_status	OnlineStatus	The online status of the Volt.
has_relay	bool	Indicates that this Volt acts as a Relay.
api_version	Version	The API version supported by the endpoint.
description	string	Optional description of the endpoint.
did_registry	string repeated	The list of DID registries that this Volt trusts.

VoltParameters

Encapsulates the various Volt parameters that are configurable by the Volt owner.

Field	Type	Description
id	string	
name	string	The name of the Volt.
description	string	Human-readable description of the Volt.
db_driver	string	The database driver in use.
		The local file path location

Field	Type	Description
location	string	The local file path location of the Volt storage.
key_strategy	string	The key strategy in use, this determines how the root key is stored.
key_id	string	The identifier for the key, the semantics depend on the key strategy in use.
ca_pem	string	The Volt certificate authority.
cert_pem	string	The Volt API server certificate.
fixed_host	string	Optional hostname of the Volt if using DNS or a static IP address, e.g. tdxvolt.com
grpc_port	int32	Port to use for hosting the Volt management service.
http_port	int32	Port to use for hosting the Volt grpc service.
http_key_path	string	The Volt http server key file path.
http_cert_path	string	The Volt http server certificate file path.
http_ca_path	string	The Volt http server certificate authority chain file path.
discoverable	bool	Indicates the Volt will be discoverable by clients using the discovery api.
authenticate_challenge	string	Optional challenge code that can be used aid in the process of authenticating clients.
require_authenticate_challenge	bool	Indicates that clients must present the correct challenge code in order to be able to authenticate.
confirm_stop	bool	Internal use only.
auto_start	bool	Internal use only.

Field	Type	Description
enable_messaging	bool	Internal use only.
has_relay	bool	Set to indicate this Volt acts as a Relay. This means this Volt can act as a proxy for other Volts (or in fact any client) that connect to it.
relay_open	bool	Set to run the Relay open to any client, i.e. clients can utilise the Relay without first authenticating.
enable_http_server	bool	Determines if the Volt HTTP server is enabled.
http_server_secure	bool	Determines whether the HTTP server employs TLS.
enable_http_forwarding	bool	Determines whether the HTTP server supports forwarding.
enable_http_api	bool	Determines if the Volt REST API is exposed via the HTTP server.
enable_websocket_api	bool	Determines if the Volt Websocket API is exposed via the HTTP server.
address	string	The hostname:port at which the Volt API is currently running.
encrypt_file_store	bool	Set to indicate the Volt file store is encrypted.
connection_id	string	This is a unique connection id. Indicates that these parameters refer to a connection to a remote Volt rather than a local Volt.
relay_ca_pem	string	The certificate authority of the Relay if this is a remote connection via a Relay.
		Optional override of the http address, rather than using the default of fixed_host:http_port

Field	Type	Description
http_address_override	string	fixed_host:http_port. This is useful if the Volt is behind a firewall or NAT, and the http server is listening on a different port from 80 or 443 but this is hidden by the proxy. For example, if the `fixed_host` is `coreid.com` and http server is listening on 2115, but the proxy is forwarding 443 to 2115, then the http_address_override would be set to `https://coreid.com`.
alias	string	An optional alias that can be used to refer to the Volt rather than the `id` field. This alias must be unique within the scope of the Battery in which the Volt is stored.
version	Version	The runtime version this Volt is running.
approve_on_challenge	bool	If set, indicates that any client that provides the correct challenge during authentication will automatically be approved to access the Volt.
approve_on_did	bool	If set, indicates that any client that proves ownership of a DID known to the Volt will automatically be approved to access the Volt.
enable_did_registry	bool	If set, indicates that clients can register DIDs with this Volt.
did_registry	string repeated	Zero or more URLs of trusted peer DID registries.
enable_outbound_smtp	bool	If set, enables outbound SMTP.
outbound_smtp_host	string	The SMTP host to use for sending emails.
outbound_smtp_port	uint32	The SMTP port to use for sending emails.

Field	Type	Description
		sending emails.
outbound_smtp_user	string	The SMTP username to use for sending emails.
outbound_smtp_password	string	The SMTP password to use for sending emails.
enable_anonymous_create	bool	If set, enables sessions that authenticate using credentials rather than a DID to create resources in the 'anonymous' system folder.
catch_all_auth_decision	PolicyDecision	The decision to apply to all authentication requests that do not match any other policy. The default is PROMPT.
enable_policy_cache	bool	If set, enables caching of policy decisions.
enable_terminal	bool	If set, enables the terminal API.
start_time	uint64	The time at which the Volt was started.

AttributeDataType

Attribute data types.

Name	Number	Description
ATTRIBUTE_DATA_TYPE_UNKNOWN	0	
ATTRIBUTE_DATA_TYPE_STRING	1	
ATTRIBUTE_DATA_TYPE_INTEGER	2	
ATTRIBUTE_DATA_TYPE_REAL	3	
ATTRIBUTE_DATA_TYPE_BOOLEAN	4	
ATTRIBUTE_DATA_TYPE_BYTES	5	
ATTRIBUTE_DATA_TYPE_IDENTITY	100	
ATTRIBUTE_DATA_TYPE_RESOURCE	101	

DiscoveryMode

Name	Number	Description
DISCOVERY_MODE_UNKNOWN	0	
DISCOVERY_MODE_TRUSTED	1	Only local identities with explicit policy PERMIT can discover.
DISCOVERY_MODE_PUBLIC	2	Any bound local identity can discover.
DISCOVERY_MODE_TRUSTED_GLOBAL	3	Only identities with explicit policy PERMIT can discover, and the service will be available to local and non-local (Relayed) clients.
DISCOVERY_MODE_PUBLIC_GLOBAL	4	Any bound identity can discover, and the service will be available to local and non-local (Relayed) clients.

OnlineStatus

Name	Number	Description
ONLINE_STATUS_UNKNOWN	0	
ONLINE_STATUS_ONLINE	1	
ONLINE_STATUS_OFFLINE	2	

PolicyDecision

@todo currently this must align with AuthorisationDecision enum in policy library, but some of the values are irrelevant outside of the public API so we need a public-facing enum and some translation.

Name	Number	Description
POLICY_DECISION_UNKNOWN	0	
POLICY_DECISION_PROMPT	1	
POLICY_DECISION_PERMIT	2	
POLICY_DECISION_DENY	3	
POLICY_DECISION_INDETERMINATE	4	
POLICY_DECISION_NOT_APPLICABLE	5	
POLICY_DECISION_APPLICABLE	6	
POLICY_DECISION_PENDING	7	

ResourceStatus

Not used ATM.

Name	Number	Description
RESOURCE_STATUS_UNKNOWN	0	
RESOURCE_STATUS_LIVE	1	
RESOURCE_STATUS_INACTIVE	2	
RESOURCE_STATUS_DELETED	999	

SecureMode

Name	Number	Description
SECURE_MODE_UNKNOWN	0	
SECURE_MODE_INSECURE	1	
SECURE_MODE_TLS	2	

ServiceHostType

Name	Number	Description
SERVICE_HOST_TYPE_UNKNOWN	0	
SERVICE_HOST_TYPE_BUILTIN	1	A built-in service hosted by the Volt.
SERVICE_HOST_TYPE_SERVER	2	A service hosted by a grpc server other than the Volt.
SERVICE_HOST_TYPE_RELAYED	3	A service hosted by a Volt client via a relay connection, i.e. the service is not exposed by a server as such, rather a Volt client implements the service and a Volt acts as a proxy, calling back to the client to implement the methods.

SessionStatus

Name	Number	Description
SESSION_STATUS_UNKNOWN	0	
SESSION_STATUS_PENDING	1	
SESSION_STATUS_LIVE	2	
SESSION_STATUS_EXPIRED	3	
SESSION_STATUS_REVOKED	4	
SESSION_STATUS_REJECTED	5	

ShareMode

Not used ATM.

Name	Number	Description
SHARE_MODE_UNKNOWN	0	
SHARE_MODE_TRUSTED	1	
SHARE_MODE_PUBLIC_READ	2	

HttpInvoke

Field	Type	Description
host	string	
port	int32	
method	string	
url	string	
version	string	
headers	HttpInvoke.HeadersEntry repeated	
body	bytes	

HttpInvoke.HeadersEntry

Field	Type	Description
key	string	
value	string	

HttpPayload

Field	Type	Description
chunk	bytes	
end	bool	
error	int32	

HttpRequest

Field	Type	Description
id	uint64	
http_invoke	HttpInvoke	
http_payload	HttpPayload	

HttpResponse

Field	Type	Description
id	uint64	
http_payload	HttpPayload	

MethodEnd

Field	Type	Description
id	uint64	
ended	bool	
error	string	
error_code	int32	

MethodInvoke

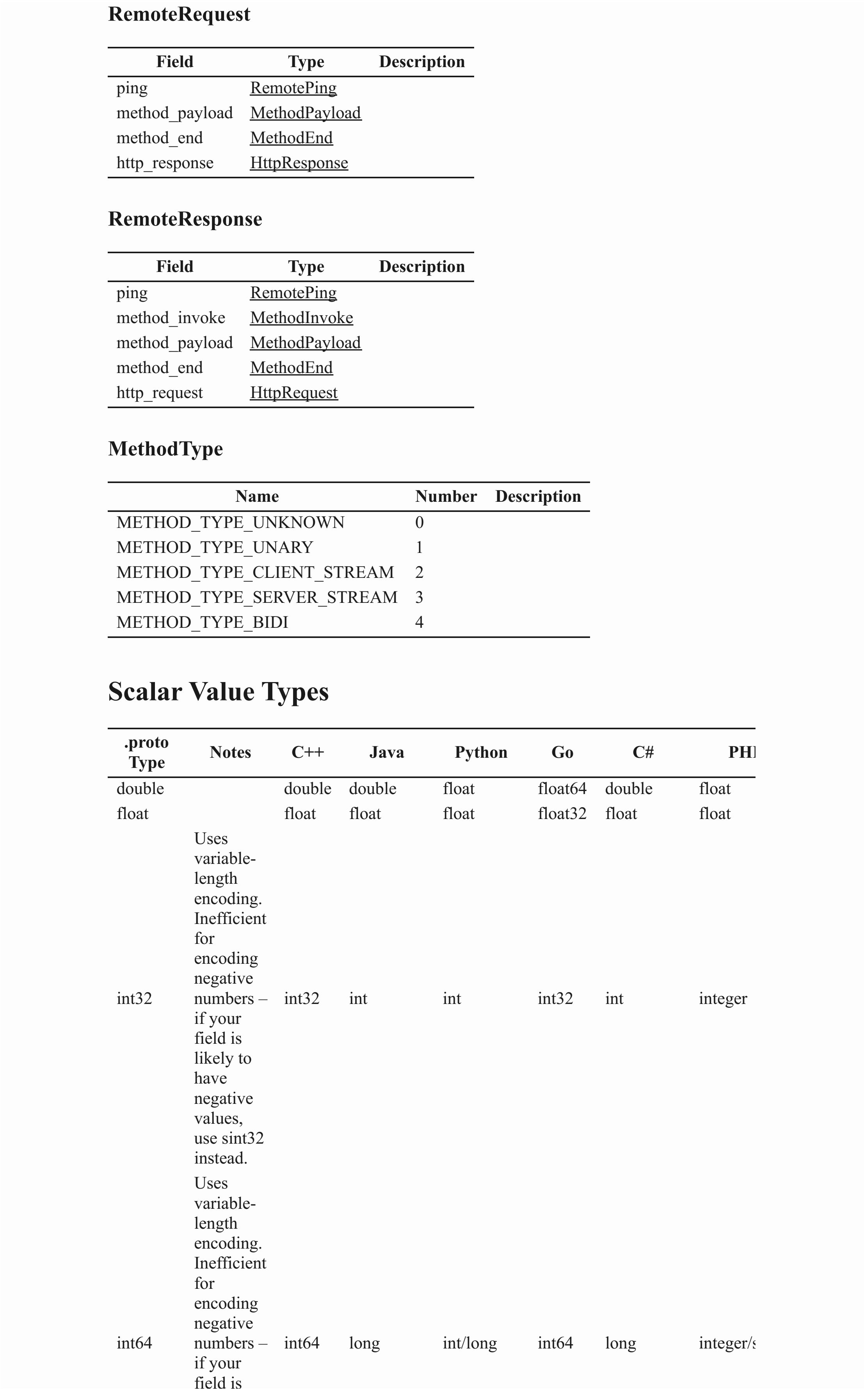
Field	Type	Description
id	uint64	
service_id	string	
method_name	string	
method_type	MethodType	
request	bytes	
json_request	string	

MethodPayload

Field	Type	Description
id	uint64	
payload	bytes	
json_payload	string	

RemotePing

Field	Type	Description
timestamp	uint64	



RemoteRequest

Field	Type	Description
ping	RemotePing	
method_payload	MethodPayload	
method_end	MethodEnd	
http_response	HttpResponse	

RemoteResponse

Field	Type	Description
ping	RemotePing	
method_invoke	MethodInvoke	
method_payload	MethodPayload	
method_end	MethodEnd	
http_request	HttpRequest	

MethodType

Name	Number	Description
METHOD_TYPE_UNKNOWN	0	
METHOD_TYPE_UNARY	1	
METHOD_TYPE_CLIENT_STREAM	2	
METHOD_TYPE_SERVER_STREAM	3	
METHOD_TYPE_BIDI	4	

Scalar Value Types

.proto Type	Notes	C++	Java	Python	Go	C#	PH
double		double	double	float	float64	double	float
float		float	float	float	float32	float	float
int32	Uses variable-length encoding. Inefficient for encoding negative numbers – if your field is likely to have negative values, use sint32 instead.	int32	int	int	int32	int	integer
int64	Uses variable-length encoding. Inefficient for encoding negative numbers – if your field is	int64	long	int/long	int64	long	integer/s

.proto Type	Notes	C++	Java	Python	Go	C#	PH
	likely to have negative values, use sint64 instead.						
uint32	Uses variable-length encoding.	uint32	int	int/long	uint32	uint	integer
uint64	Uses variable-length encoding.	uint64	long	int/long	uint64	ulong	integer/s
sint32	Uses variable-length encoding. Signed int value. These more efficiently encode negative numbers than regular int32s.	int32	int	int	int32	int	integer
sint64	Uses variable-length encoding. Signed int value. These more efficiently encode negative numbers than regular int64s.	int64	long	int/long	int64	long	integer/s
fixed32	Always four bytes. More efficient than uint32 if values are often greater than 2^28.	uint32	int	int	uint32	uint	integer
fixed64	Always eight bytes. More efficient than uint64 if values are often greater than 2^56	uint64	long	int/long	uint64	ulong	integer/s

.proto Type	Notes	C++	Java	Python	Go	C#	PH
sfixed32	Always four bytes.	int32	int	int	int32	int	integer
sfixed64	Always eight bytes.	int64	long	int/long	int64	long	integer/s
bool		bool	boolean	boolean	bool	bool	boolean
string	A string must always contain UTF-8 encoded or 7-bit ASCII text.	string	String	str/unicode	string	string	string
bytes	May contain any arbitrary sequence of bytes.	string	ByteString	str	[]byte	ByteString	string

Skip to Content

tdx Volt

putting you in charge

☀🌙

Toggle sidebar

- ## Getting Started
  - Welcome
  - Quick Start
- ## Concepts
  - Fundamentals
  - Identity
  - DID registry
  - Verifiable credentials
  - Policy
  - Resource
  - File
  - Database
  - Wire
  - Relay
  - Key strategy
- ## How to...
  - Create a Volt
  - Start a Volt
  - Connect to a Volt
  - Upload files
  - Download files
  - Publish to wire

  - Subscribe to wire
  - Create a database
  - Execute SQL
  - Import data
  - Establish a connect stream
  - Approve authenticate request

- ## Clients

  - Command line
  - Native / C++
  - Web
  - NodeJS

- ## Reference

  - Best practice
  - Battery
  - Configuration
  - Connection
  - Connect stream
  - Logging
  - PKCS#11

- ## API

  - Discovery API
  - File API
  - SqliteDatabase API
  - SqliteServer API
  - SSI API
  - Sync API
  - Relay API
  - Volt API
  - Wire API

- ## Utilities

  - protoDbSync
  - sqliteServer
  - wireTransform

- ## FAQ

  - Questions

- ## Coming soon

  - Roadmap

- ☀☾

Contents

Contents

ProtobufSyncConfiguration

Describes a single message type.

A set of one or more of these messages is specified in `ProtobufSyncConfigurationHeader`.

Field	Type	Description
id	string	Optional id to associate with this configuration. This can be used in the `header_id` field of `ProtobufSyncWrapper` above to reference the configuration. If omitted the numerical index of the configuration in `ProtobufSyncConfigurationHeader` will be used instead.
message_proto	string	The actual protobuf definition text. Copy and paste the source protobuf definition from the `.proto` file. Only simple protobuf structures are currently supported, e.g. no imports from other packages etc.
message_name	string	The name of the message within `message_proto` above that represents the data to be sync'd, e.g. `TCPDumpPacket`.
table_name	string	The name of the table within the target database into which the message data for this type should be written.

ProtobufSyncConfigurationHeader

This message is written at the beginning of every file to be ingested using the `protoDbSync` utility.

It contains a `header` entry for each message type that may appear in the file.

If the `volt logger` command is used, it will create this header automatically based on the configuration it's given.

Field	Type	Description
id	string	This should ideally be a persistent UUID, at minimum it must be unique within the set of types of file any given instance of `protoDbSync` is processing in a given folder. It is used to match up orphaned or split packets that might occur when receiving data from a wire, for example, if a log file is rotated midway through a packet arriving on the wire. This id should persist for the life time of the set of data it describes, i.e. if a wire publication is stopped and restarted at some later point, the same id should be used if possible.
configuration	ProtobufSyncConfiguration repeated	The set of possible configurations that can appear in any given protobuf sync data file. A serialised instance of this message must appear at the top of each data file. Each subsequent serialised message in the data file must be an instance of `ProtobufSyncWrapper`, and the `header_lookup` field refers to an entry in this list.
maximum_message_size	int32	Optional maximum size of the serialised messages, this doesn't need to be exact and the default is 64K if omitted.

ProtobufSyncWrapper

Wraps arbitrary protobuf messages, with an index into the `ProtobufSyncConfigurationHeader` to indicate the specific message type this message wraps.

Field	Type	Description
header_index	uint32	The index number of the header for this message type in the Volt logger configuration file.
header_id	string	The name of the header for this message type, will be used to lookup against the `id` field in `ProtobufSyncConfiguration`. This will incur an overhead in terms of the packet size, but might be preferrable if volume is low or managing the header index is difficult.
payload	bytes	The message payload, in serialised protobuf binary format. n.b. the serialisation should **not** be length-prefixed.

Scalar Value Types

.proto Type	Notes	C++	Java	Python	Go	C#	PH
double		double	double	float	float64	double	float
float		float	float	float	float32	float	float
int32	Uses variable-length encoding. Inefficient for encoding negative numbers – if your field is likely to have negative values, use sint32 instead.	int32	int	int	int32	int	integer
int64	Uses variable-length encoding. Inefficient for encoding negative numbers – if your field is likely to have negative	int64	long	int/long	int64	long	integer/s

.proto Type	Notes	C++	Java	Python	Go	C#	PH
	values, use sint64 instead.						
uint32	Uses variable-length encoding.	uint32	int	int/long	uint32	uint	integer
uint64	Uses variable-length encoding.	uint64	long	int/long	uint64	ulong	integer/s
sint32	Uses variable-length encoding. Signed int value. These more efficiently encode negative numbers than regular int32s.	int32	int	int	int32	int	integer
sint64	Uses variable-length encoding. Signed int value. These more efficiently encode negative numbers than regular int64s.	int64	long	int/long	int64	long	integer/s
fixed32	Always four bytes. More efficient than uint32 if values are often greater than 2^28.	uint32	int	int	uint32	uint	integer
fixed64	Always eight bytes. More efficient than uint64 if values are often greater than 2^56.	uint64	long	int/long	uint64	ulong	integer/s
sfixed32	Always four bytes.	int32	int	int	int32	int	integer

.proto Type	Notes	C++	Java	Python	Go	C#	PH
sfixed64	Always eight bytes.	int64	long	int/long	int64	long	integer/s
bool		bool	boolean	boolean	bool	bool	boolean
string	A string must always contain UTF-8 encoded or 7-bit ASCII text.	string	String	str/unicode	string	string	string
bytes	May contain any arbitrary sequence of bytes.	string	ByteString	str	[]byte	ByteString	string

Skip to Content

tdx Volt

putting you in charge

☀ ☾

Toggle sidebar

- **Getting Started**
  - Welcome
  - Quick Start
- **Concepts**
  - Fundamentals
  - Identity
  - DID registry
  - Verifiable credentials
  - Policy
  - Resource
  - File
  - Database
  - Wire
  - Relay
  - Key strategy
- **How to...**
  - Create a Volt
  - Start a Volt
  - Connect to a Volt
  - Upload files
  - Download files
  - Publish to wire
  - Subscribe to wire
  - Create a database
  - Execute SQL
  - Import data

- Establish a connect stream
- Approve authenticate request

- ## Clients
  - Command line
  - Native / C++
  - Web
  - NodeJS

- ## Reference
  - Best practice
  - Battery
  - Configuration
  - Connection
  - Connect stream
  - Logging
  - PKCS#11

- ## API
  - Discovery API
  - File API
  - SqliteDatabase API
  - SqliteServer API
  - SSI API
  - Sync API
  - Relay API
  - Volt API
  - Wire API

- ## Utilities
  - protoDbSync
  - sqliteServer
  - wireTransform

- ## FAQ
  - Questions

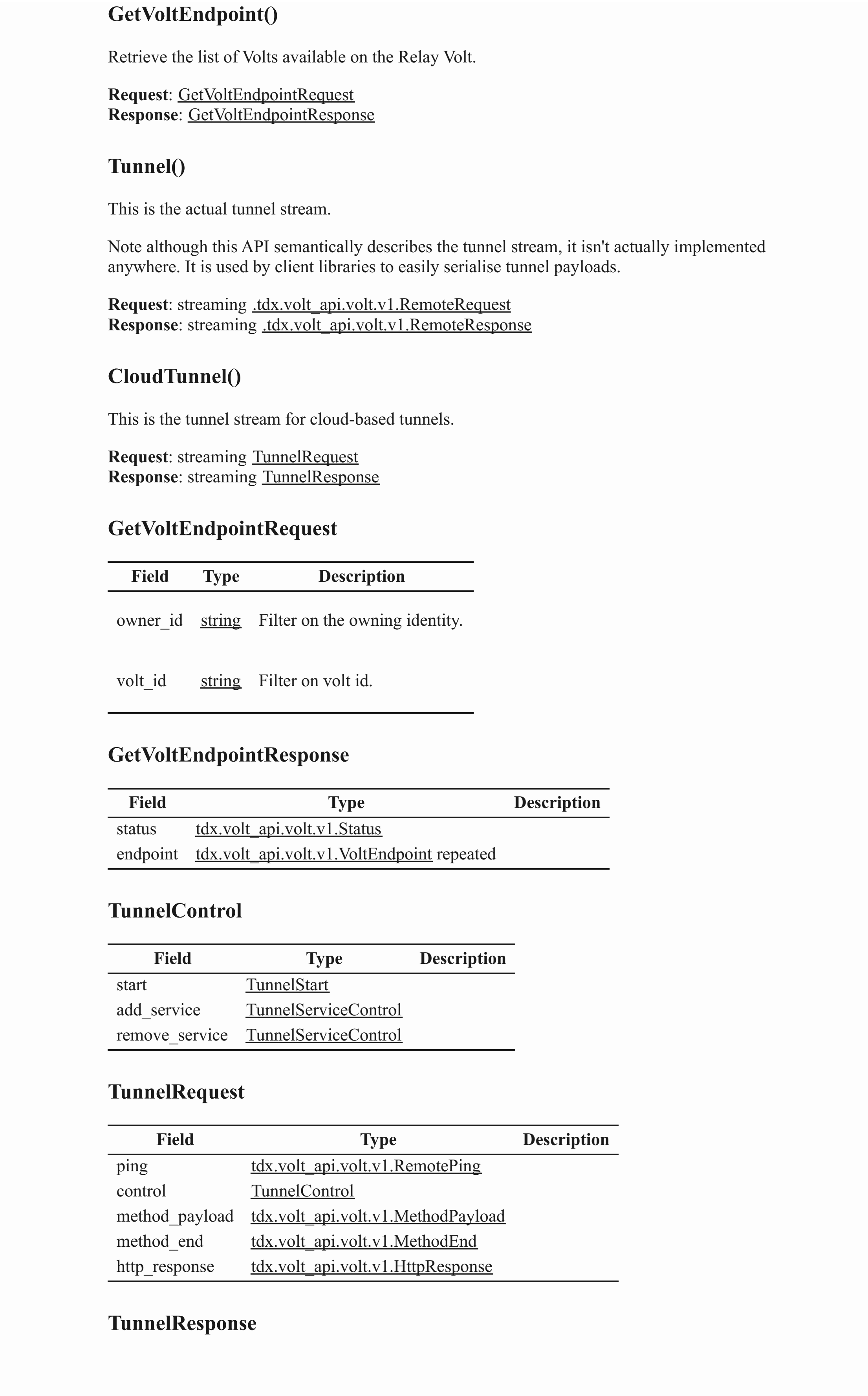
- ## Coming soon
  - Roadmap
- ☀☾

Contents

Contents

RelayAPI

Top



GetVoltEndpoint()

Retrieve the list of Volts available on the Relay Volt.

Request: GetVoltEndpointRequest
Response: GetVoltEndpointResponse

Tunnel()

This is the actual tunnel stream.

Note although this API semantically describes the tunnel stream, it isn't actually implemented anywhere. It is used by client libraries to easily serialise tunnel payloads.

Request: streaming .tdx.volt_api.volt.v1.RemoteRequest
Response: streaming .tdx.volt_api.volt.v1.RemoteResponse

CloudTunnel()

This is the tunnel stream for cloud-based tunnels.

Request: streaming TunnelRequest
Response: streaming TunnelResponse

GetVoltEndpointRequest

Field	Type	Description
owner_id	string	Filter on the owning identity.
volt_id	string	Filter on volt id.

GetVoltEndpointResponse

Field	Type	Description
status	tdx.volt_api.volt.v1.Status	
endpoint	tdx.volt_api.volt.v1.VoltEndpoint repeated	

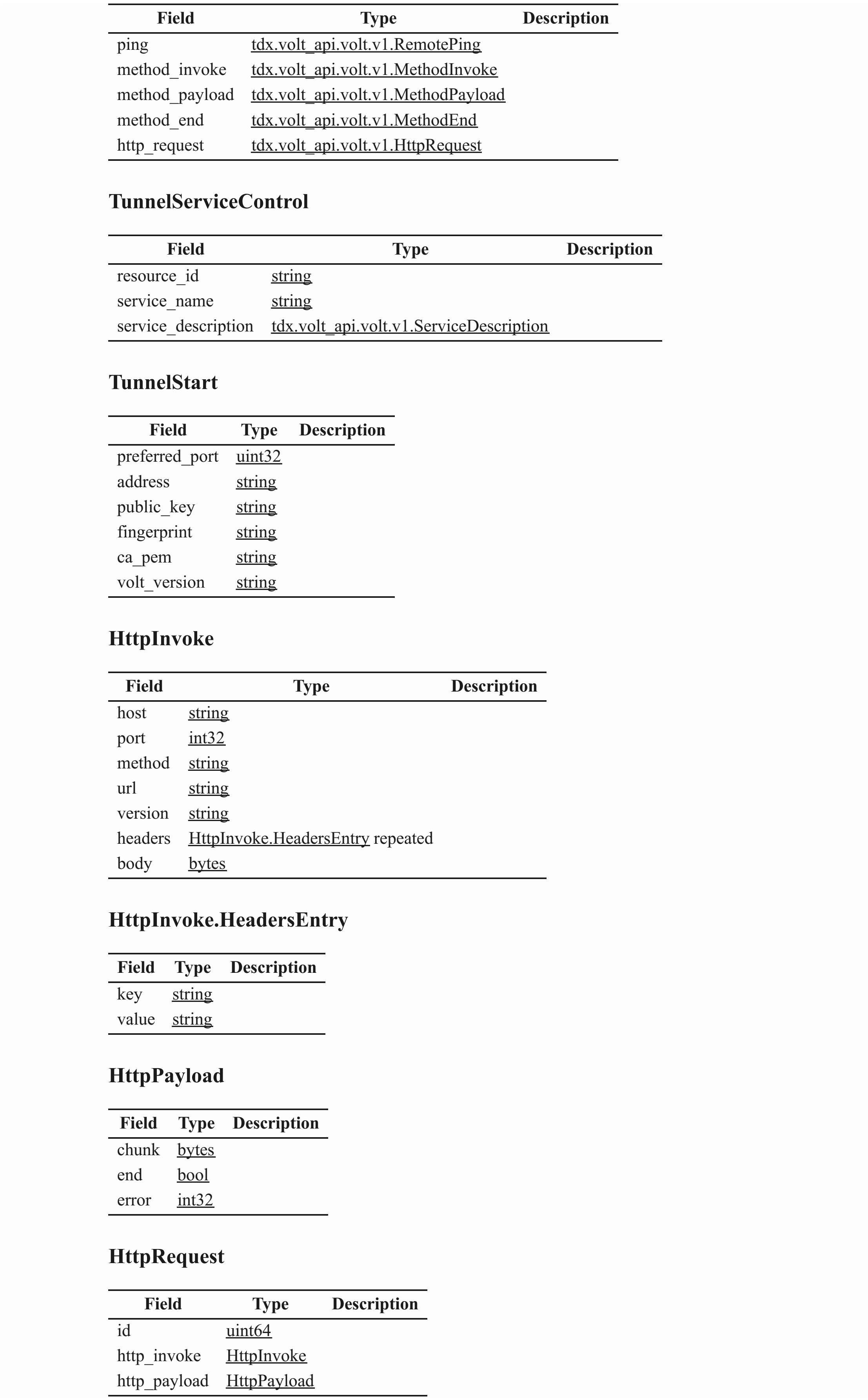
TunnelControl

Field	Type	Description
start	TunnelStart	
add_service	TunnelServiceControl	
remove_service	TunnelServiceControl	

TunnelRequest

Field	Type	Description
ping	tdx.volt_api.volt.v1.RemotePing	
control	TunnelControl	
method_payload	tdx.volt_api.volt.v1.MethodPayload	
method_end	tdx.volt_api.volt.v1.MethodEnd	
http_response	tdx.volt_api.volt.v1.HttpResponse	

TunnelResponse



Field	Type	Description
ping	tdx.volt_api.volt.v1.RemotePing	
method_invoke	tdx.volt_api.volt.v1.MethodInvoke	
method_payload	tdx.volt_api.volt.v1.MethodPayload	
method_end	tdx.volt_api.volt.v1.MethodEnd	
http_request	tdx.volt_api.volt.v1.HttpRequest	

TunnelServiceControl

Field	Type	Description
resource_id	string	
service_name	string	
service_description	tdx.volt_api.volt.v1.ServiceDescription	

TunnelStart

Field	Type	Description
preferred_port	uint32	
address	string	
public_key	string	
fingerprint	string	
ca_pem	string	
volt_version	string	

HttpInvoke

Field	Type	Description
host	string	
port	int32	
method	string	
url	string	
version	string	
headers	HttpInvoke.HeadersEntry repeated	
body	bytes	

HttpInvoke.HeadersEntry

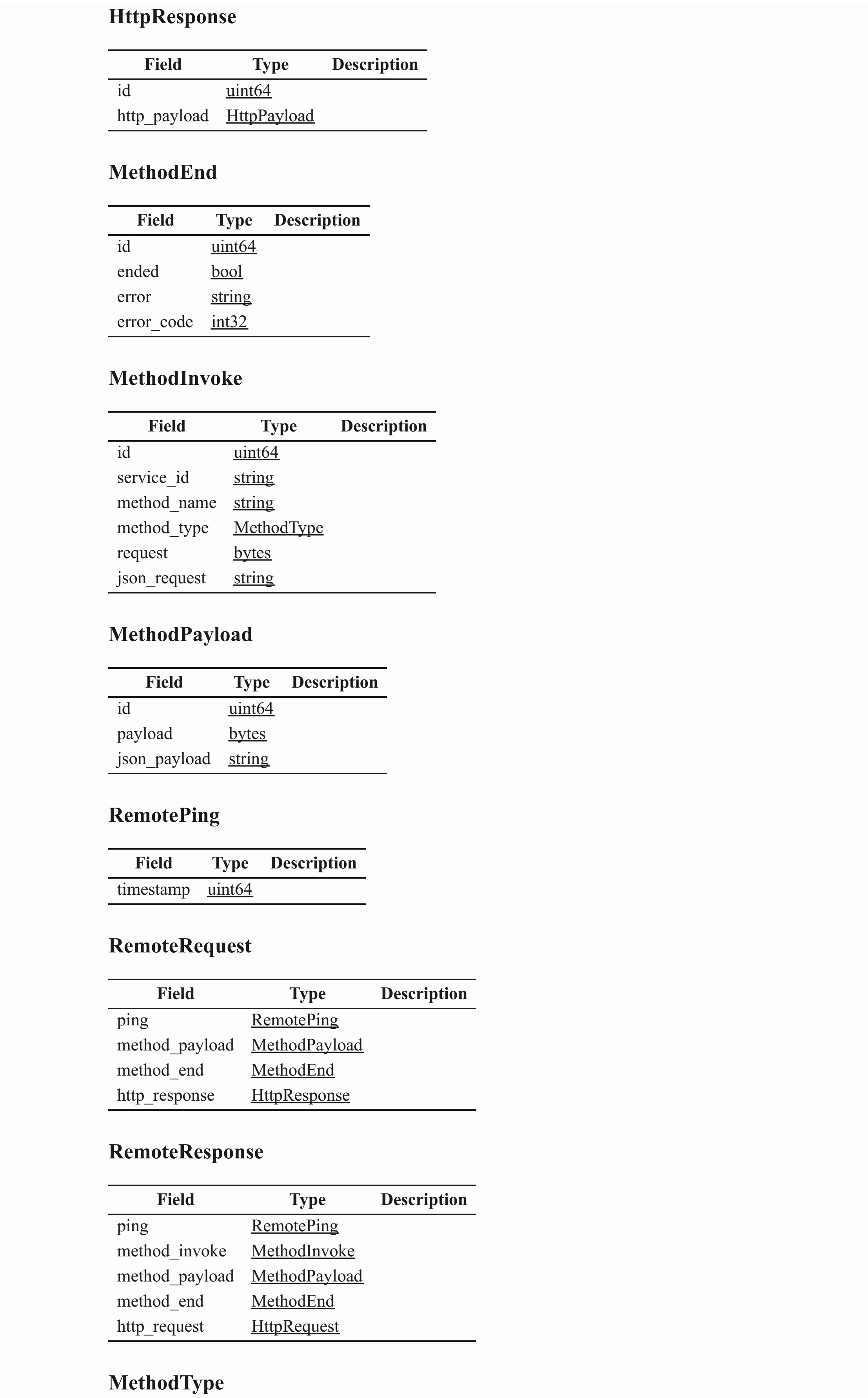
Field	Type	Description
key	string	
value	string	

HttpPayload

Field	Type	Description
chunk	bytes	
end	bool	
error	int32	

HttpRequest

Field	Type	Description
id	uint64	
http_invoke	HttpInvoke	
http_payload	HttpPayload	



HttpResponse

Field	Type	Description
id	uint64	
http_payload	HttpPayload	

MethodEnd

Field	Type	Description
id	uint64	
ended	bool	
error	string	
error_code	int32	

MethodInvoke

Field	Type	Description
id	uint64	
service_id	string	
method_name	string	
method_type	MethodType	
request	bytes	
json_request	string	

MethodPayload

Field	Type	Description
id	uint64	
payload	bytes	
json_payload	string	

RemotePing

Field	Type	Description
timestamp	uint64	

RemoteRequest

Field	Type	Description
ping	RemotePing	
method_payload	MethodPayload	
method_end	MethodEnd	
http_response	HttpResponse	

RemoteResponse

Field	Type	Description
ping	RemotePing	
method_invoke	MethodInvoke	
method_payload	MethodPayload	
method_end	MethodEnd	
http_request	HttpRequest	

MethodType

Name	Number	Description
METHOD_TYPE_UNKNOWN	0	
METHOD_TYPE_UNARY	1	
METHOD_TYPE_CLIENT_STREAM	2	
METHOD_TYPE_SERVER_STREAM	3	
METHOD_TYPE_BIDI	4	

Status

Field	Type	Description
code	int32	A simple error code that can be easily handled by the client. Mirrors the grpc StatusCode enum, 0 => OK
message	string	A developer-facing human-readable error message in English. It should both explain the error and offer an actionable resolution to it.
description	string	Long form error description.

Access

Field	Type	Description
id	string	
resource_id	string	The resource being accessed.
resource_name	string	A human-readable short identifier of the resource.
resource_owner	string	The identity that owns the resource.
resource_kind	string repeated	The kind of resource.
identity_did	string	The identity attempting access.
credential_lookup	string	The JSON path array for looking up verifiable credentials.
identity_name	string	A human-readable short identifier of the subject.
identity_kind	string repeated	The kind of identity.
access	string	Requested access.
extra	string	Optional extra data.
decision	PolicyDecision	Assigned decision.
recursive	bool	
request_time	int64	Time at which the request was made.
decision_time	int64	Time at which the decision was taken.
request_count	uint32	Counter of number times this access was requested.

AttributeValue

Attribute value will be one of the following fields, depending on the data type.

Field	Type	Description
string	string	
integer	int64	
real	double	
boolean	bool	
bytes	bytes	

Identity

A Volt identity encompasses a Resource and a set of identity aliases.

Field	Type	Description
resource	Resource	
alias	IdentityAlias repeated	

IdentityAlias

Field	Type	Description
id	uint32	The alias id.
identity_did	string	The corresponding identity id.
alias	string	The actual alias, e.g. a common name or key fingerprint.
public_key	string	This will only be populated if alias_type == tdx:public-key
private_key	string	This will only be populated if alias_type == tdx:public-key, and the key is stored in the Volt.
alias_type	string	The alias type, for example public key, email, phone number etc.
issuer_id	string	The identity that issued this alias.
authenticate	PolicyDecision	Indicates if this alias has an authenticate policy decision assigned.
description	string	Optional description of this alias.

MethodDescription

Internal use only.

Field	Type	Description
path	string	
client_streaming	bool	
server_streaming	bool	

ProtoFile

Describes a single protobuf file for use in ServiceDescription.

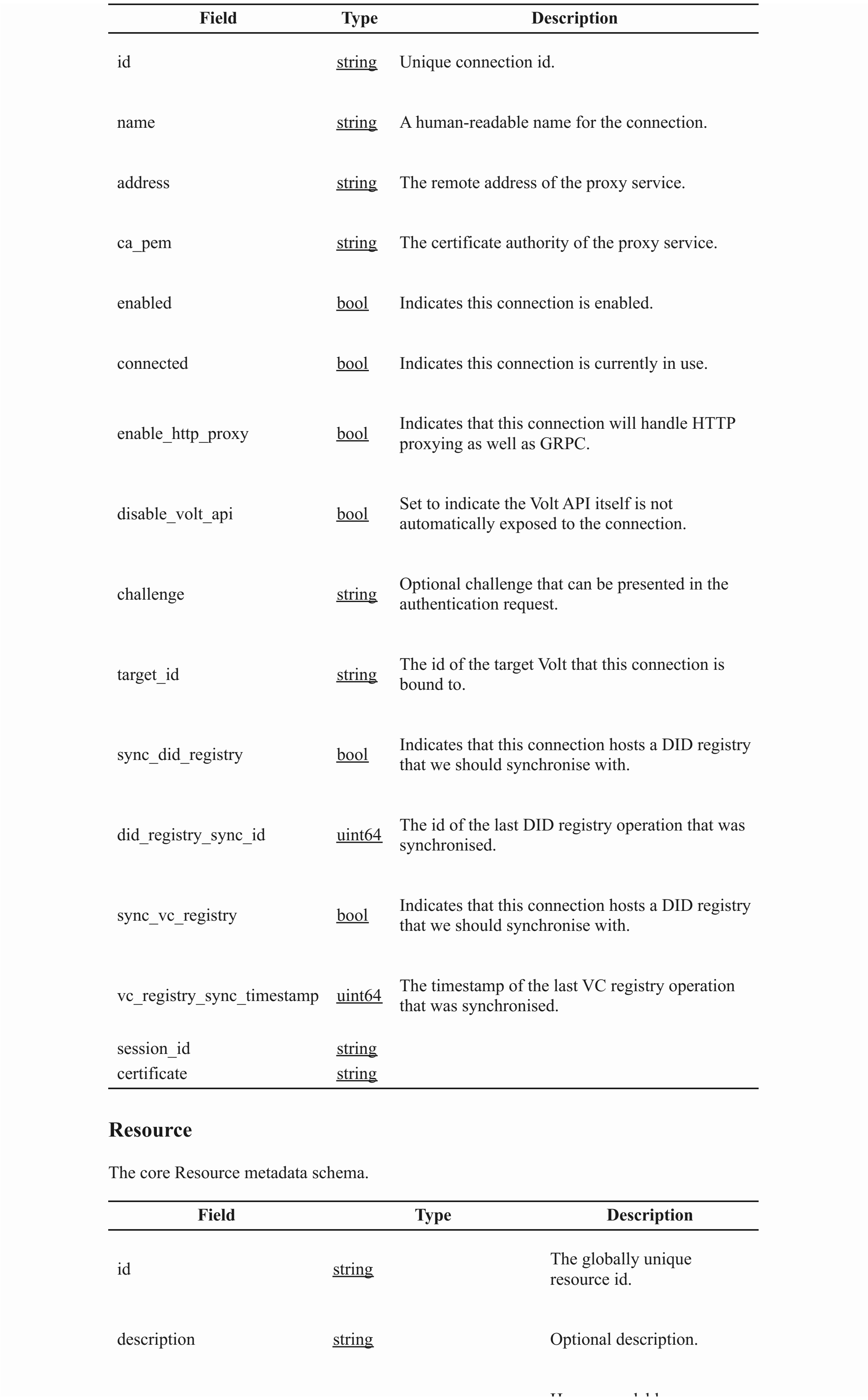
Field	Type	Description
file_path	string	The path name of the proto file, relative to the 'root' of the namespace, e.g. "tdx/volt_api/volt/v1/volt.proto".
protobuf	string	The actual protobuf file contents.
service_name	string repeated	Optional - the service(s) contained in this protobuf file, if omitted here they will be loaded dynamically from the protobuf.

ProxyConnection

Represents an outbound connection from a Volt to a remote service that will act as a proxy for that Volt.

This enables Volts to bypass firewall and NATs.

Example - connection from a Volt to a Relay Volt running on the public internet, such as tdxvolt.com



Field	Type	Description
id	string	Unique connection id.
name	string	A human-readable name for the connection.
address	string	The remote address of the proxy service.
ca_pem	string	The certificate authority of the proxy service.
enabled	bool	Indicates this connection is enabled.
connected	bool	Indicates this connection is currently in use.
enable_http_proxy	bool	Indicates that this connection will handle HTTP proxying as well as GRPC.
disable_volt_api	bool	Set to indicate the Volt API itself is not automatically exposed to the connection.
challenge	string	Optional challenge that can be presented in the authentication request.
target_id	string	The id of the target Volt that this connection is bound to.
sync_did_registry	bool	Indicates that this connection hosts a DID registry that we should synchronise with.
did_registry_sync_id	uint64	The id of the last DID registry operation that was synchronised.
sync_vc_registry	bool	Indicates that this connection hosts a DID registry that we should synchronise with.
vc_registry_sync_timestamp	uint64	The timestamp of the last VC registry operation that was synchronised.
session_id	string	
certificate	string	

Resource

The core Resource metadata schema.

Field	Type	Description
id	string	The globally unique resource id.
description	string	Optional description.

Field	Type	Description
name	string	Human-readable resource name.
share_mode	ShareMode	Not in use.
volt_id	string	The id of the Volt that hosts this resource.
service_description	ServiceDescription	Optional description of any services exposed by this resource.
attribute	ResourceAttribute repeated	Attributes assigned to the resource.
platform_version	Version	The version of the platform.
version	uint64	The resource version.
owner	string	The identity of the resource owner.
created	uint64	Creation timestamp, milliseconds since epoch.
modified	uint64	Last modification timestamp, milliseconds since epoch.
status	ResourceStatus	Not in use.
kind	string repeated	The taxonomy of the resource.
online_status	OnlineStatus	The online status. For most kinds of resource this indicates that the server hosting the resource is online, the exception being identity resources, in which case the status reflects whether or not the identity has a live connection. All built-in resources are hosted by the Volt itself and are therefore always online when the Volt is running. Resources hosted by external servers are online if the server itself is online and has registered the resource as online using `setServiceStatus`.

Field	Type	Description
size	uint64	The size of the resource store in bytes.
store	string	The path to the resource store.
alias	string repeated	Alias(es) that can be used to refer to the resource rather than the id. Each alias must be unique to the Volt, this is enforced by the API. No format restrictions are currently applied to alias, but this may change in future, for the time being it makes sense to stick to alphanumeric characters and '_' or '-'.
content_hash	string	The hash of the resource content contained in the store.
child	Resource repeated	Not yet supported.

ResourceAttribute

A resource attribute enables storing arbitrary data associated with a resource.

Field	Type	Description
id	uint32	
attribute_id	string	
resource_id	string	
data_type	AttributeDataType	
value	AttributeValue repeated	

ServiceDescription

Describes a Volt service.

Field	Type	Description
host_type	ServiceHostType	The configuration used by the host of this service.
host_client_id	string	The identity of the client that is exposing the service. For example, if a third party is exposing a database service via a Volt, it will first authenticate and obtain a client DID and credentials in order to be able to create service resource(s). Any resources that are owned by this client will be marked as online if the client itself is online, i.e. has a live connection to the Volt.

Field	Type	Description
		This will be empty if the service is a built-in Volt service.
host_service_id	string	The id of the resource that holds the protobuf definition for this resource. For example, if a third party is exposing a database service via a Volt, it will create a service resource that holds details of the protobuf methods exposed by the service. For built-in services, i.e. those hosted by the Volt, this will set to the Volt id.
host_address	string	The address of the grpc server hosting this service. Only relevant to grpc-hosted services.
host_ca_pem	string	The certificate authority (chain) that signed the service server certificate. This is only relevant to grpc-hosted services.
host_public_key	string	The public key of the service host, which is used to encrypt payloads. This may change as the service comes and goes online.
host_connection_id	string	The connection id currently used to host this service.
host_session_id	string	Internal use only.
discoverable	DiscoveryMode	The discovery mode.
ping_timestamp	int64	The ping timestamp of the server hosting this service.
proto_file	ProtoFile repeated	The protobuf definitions of the APIs exposed by this service.
service_api	string repeated	The fully qualified names of the protobuf services, for example tdx.volt_api.webcam.v1.WebcamControlAPI.
method	MethodDescription repeated	Internal use only.

Session

Field	Type	Description
id	string	
identity_did	string	
identity_name	string	
ip	string	
created	uint64	
modified	uint64	
expires	uint64	
credential	SessionCredential repeated	
origin	string	
status	SessionStatus	

SessionCredential

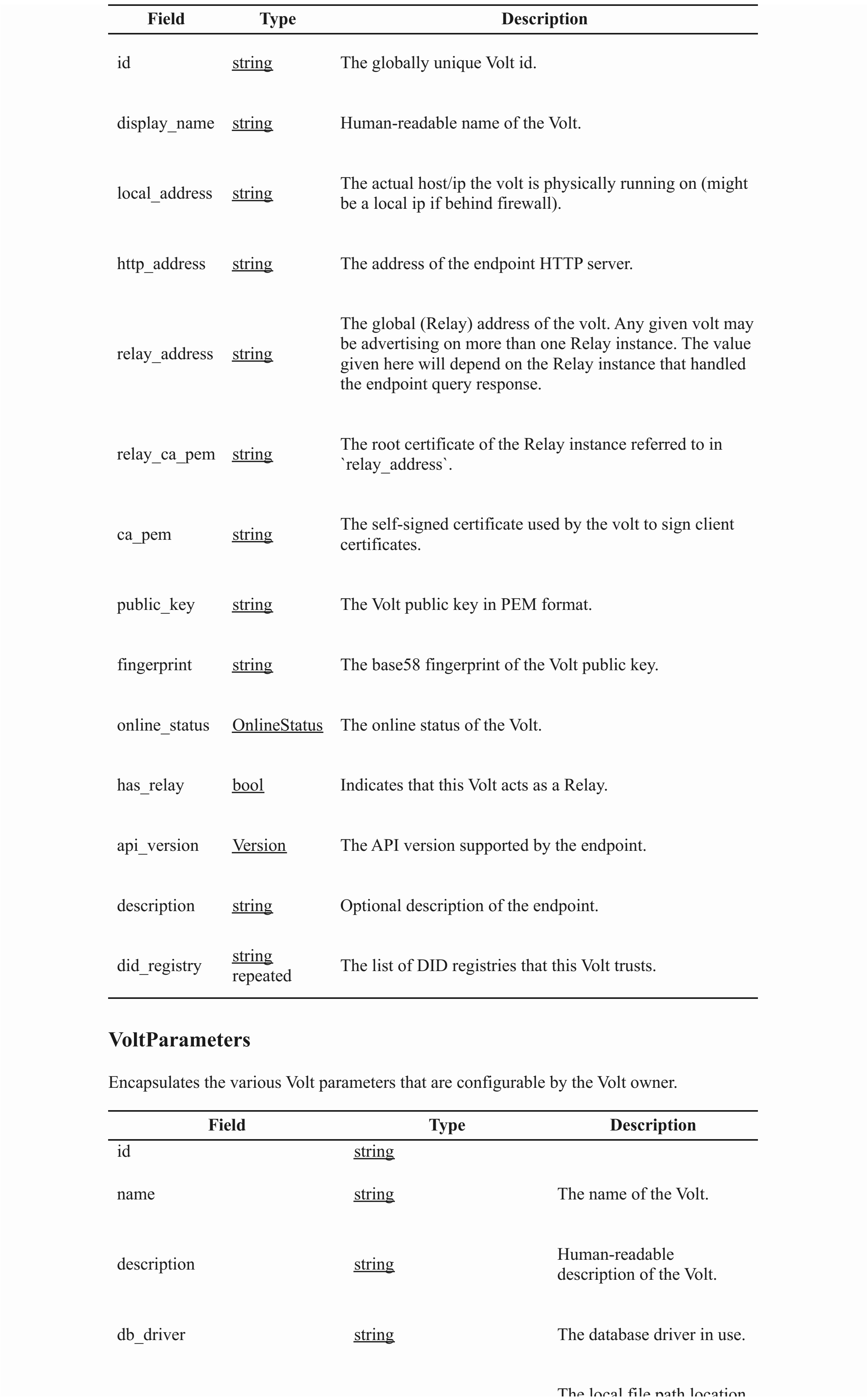
Field	Type	Description
id	uint32	The alias id.
session_id	string	The corresponding session id.
credential_type	string	The credential type, for example public key, verifiable credential, challenge etc.
description	string	Optional description of this credential.
vc_id	string	The id of the verifiable credential, if the credential type is volt:vc-claim.
vc_json	string	The verifiable credential in JSON format, if the credential type is volt:vc-claim.
vc_subject_id	string	The subject id extracted from the `vc_json` field.
vc_issuer_id	string	The issuer id extracted from the `vc_json` field.
vc_type	string	The comma-separated type(s) extracted from the `vc_json` field.
challenge	string	The challenge string, if the credential type is volt:challenge.
key_fingerprint	string	The key fingerprint, if the credential type is volt:public-key.
public_key	string	The PEM-encoded public key, if the credential type is volt:public-key.
private_key	string	Optional PEM-encoded private key, if the credential type is volt:public-key. Only used for ephemeral REST-base sessions created dynamically after OTP authentication.
extra	string	Type-specific extra data stored with the credential.
extra_2	string	More type-specific data stored with the credential.

Version

Using `major` and `minor` here upsets the GNU C Library, so we add a `version_` prefix.

Field	Type	Description
version_major	uint32	
version_minor	uint32	
version_patch	uint32	

VoltEndpoint



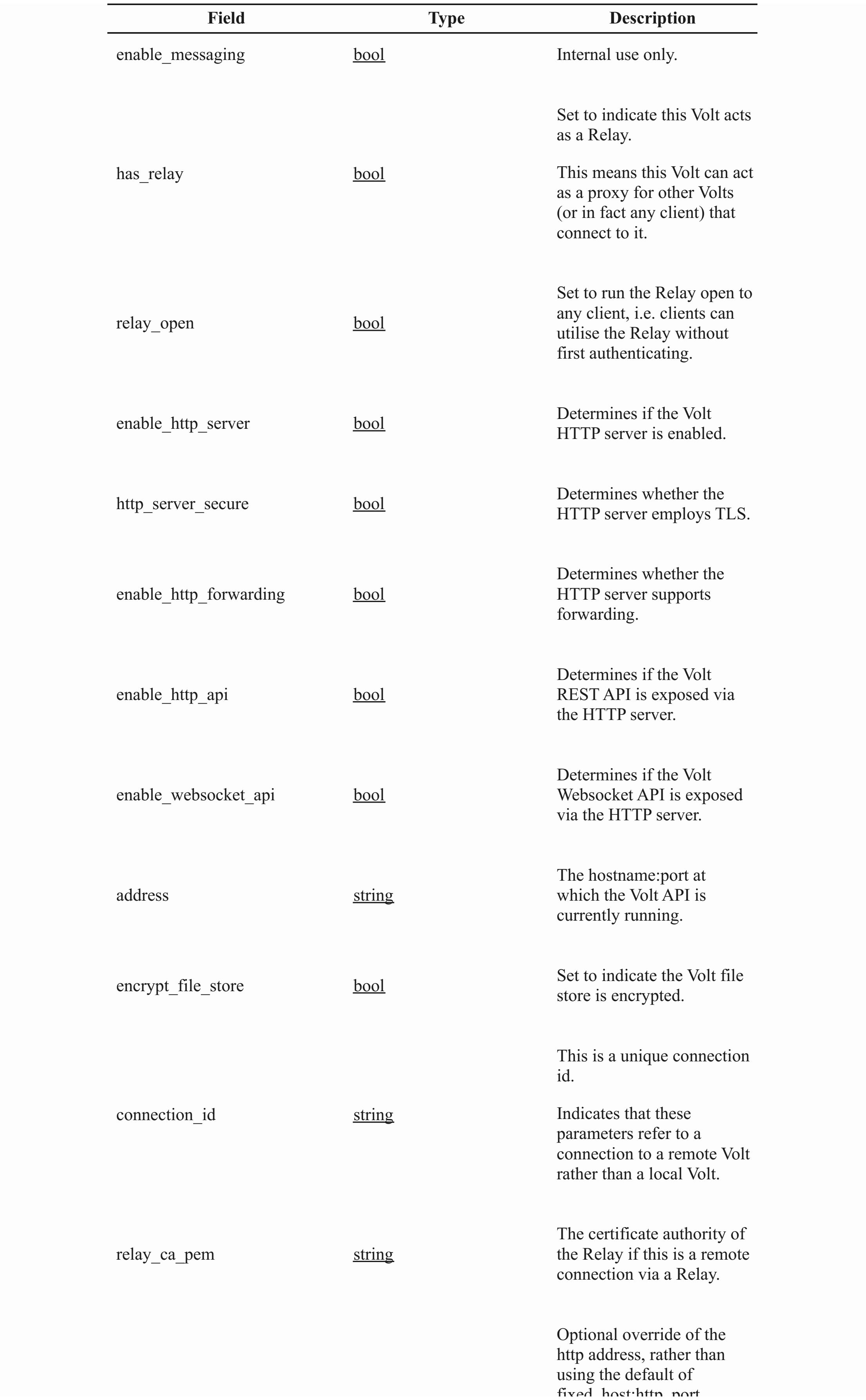
Field	Type	Description
id	string	The globally unique Volt id.
display_name	string	Human-readable name of the Volt.
local_address	string	The actual host/ip the volt is physically running on (might be a local ip if behind firewall).
http_address	string	The address of the endpoint HTTP server.
relay_address	string	The global (Relay) address of the volt. Any given volt may be advertising on more than one Relay instance. The value given here will depend on the Relay instance that handled the endpoint query response.
relay_ca_pem	string	The root certificate of the Relay instance referred to in \`relay_address\`.
ca_pem	string	The self-signed certificate used by the volt to sign client certificates.
public_key	string	The Volt public key in PEM format.
fingerprint	string	The base58 fingerprint of the Volt public key.
online_status	OnlineStatus	The online status of the Volt.
has_relay	bool	Indicates that this Volt acts as a Relay.
api_version	Version	The API version supported by the endpoint.
description	string	Optional description of the endpoint.
did_registry	string repeated	The list of DID registries that this Volt trusts.

VoltParameters

Encapsulates the various Volt parameters that are configurable by the Volt owner.

Field	Type	Description
id	string	
name	string	The name of the Volt.
description	string	Human-readable description of the Volt.
db_driver	string	The database driver in use.
		The local file path location

Field	Type	Description
location	string	The local file path location of the Volt storage.
key_strategy	string	The key strategy in use, this determines how the root key is stored.
key_id	string	The identifier for the key, the semantics depend on the key strategy in use.
ca_pem	string	The Volt certificate authority.
cert_pem	string	The Volt API server certificate.
fixed_host	string	Optional hostname of the Volt if using DNS or a static IP address, e.g. tdxvolt.com
grpc_port	int32	Port to use for hosting the Volt management service.
http_port	int32	Port to use for hosting the Volt grpc service.
http_key_path	string	The Volt http server key file path.
http_cert_path	string	The Volt http server certificate file path.
http_ca_path	string	The Volt http server certificate authority chain file path.
discoverable	bool	Indicates the Volt will be discoverable by clients using the discovery api.
authenticate_challenge	string	Optional challenge code that can be used aid in the process of authenticating clients.
require_authenticate_challenge	bool	Indicates that clients must present the correct challenge code in order to be able to authenticate.
confirm_stop	bool	Internal use only.
auto_start	bool	Internal use only.



Field	Type	Description
enable_messaging	bool	Internal use only.
has_relay	bool	Set to indicate this Volt acts as a Relay. This means this Volt can act as a proxy for other Volts (or in fact any client) that connect to it.
relay_open	bool	Set to run the Relay open to any client, i.e. clients can utilise the Relay without first authenticating.
enable_http_server	bool	Determines if the Volt HTTP server is enabled.
http_server_secure	bool	Determines whether the HTTP server employs TLS.
enable_http_forwarding	bool	Determines whether the HTTP server supports forwarding.
enable_http_api	bool	Determines if the Volt REST API is exposed via the HTTP server.
enable_websocket_api	bool	Determines if the Volt Websocket API is exposed via the HTTP server.
address	string	The hostname:port at which the Volt API is currently running.
encrypt_file_store	bool	Set to indicate the Volt file store is encrypted.
connection_id	string	This is a unique connection id. Indicates that these parameters refer to a connection to a remote Volt rather than a local Volt.
relay_ca_pem	string	The certificate authority of the Relay if this is a remote connection via a Relay.
		Optional override of the http address, rather than using the default of fixed_host:http_port

Field	Type	Description
http_address_override	string	fixed_host:http_port. This is useful if the Volt is behind a firewall or NAT, and the http server is listening on a different port from 80 or 443 but this is hidden by the proxy. For example, if the `fixed_host` is `coreid.com` and http server is listening on 2115, but the proxy is forwarding 443 to 2115, then the http_address_override would be set to `https://coreid.com`.
alias	string	An optional alias that can be used to refer to the Volt rather than the `id` field. This alias must be unique within the scope of the Battery in which the Volt is stored.
version	Version	The runtime version this Volt is running.
approve_on_challenge	bool	If set, indicates that any client that provides the correct challenge during authentication will automatically be approved to access the Volt.
approve_on_did	bool	If set, indicates that any client that proves ownership of a DID known to the Volt will automatically be approved to access the Volt.
enable_did_registry	bool	If set, indicates that clients can register DIDs with this Volt.
did_registry	string repeated	Zero or more URLs of trusted peer DID registries.
enable_outbound_smtp	bool	If set, enables outbound SMTP.
outbound_smtp_host	string	The SMTP host to use for sending emails.
outbound_smtp_port	uint32	The SMTP port to use for sending emails.

sending emails.

Field	Type	Description
outbound_smtp_user	string	The SMTP username to use for sending emails.
outbound_smtp_password	string	The SMTP password to use for sending emails.
enable_anonymous_create	bool	If set, enables sessions that authenticate using credentials rather than a DID to create resources in the 'anonymous' system folder.
catch_all_auth_decision	PolicyDecision	The decision to apply to all authentication requests that do not match any other policy. The default is PROMPT.
enable_policy_cache	bool	If set, enables caching of policy decisions.
enable_terminal	bool	If set, enables the terminal API.
start_time	uint64	The time at which the Volt was started.

AttributeDataType

Attribute data types.

Name	Number	Description
ATTRIBUTE_DATA_TYPE_UNKNOWN	0	
ATTRIBUTE_DATA_TYPE_STRING	1	
ATTRIBUTE_DATA_TYPE_INTEGER	2	
ATTRIBUTE_DATA_TYPE_REAL	3	
ATTRIBUTE_DATA_TYPE_BOOLEAN	4	
ATTRIBUTE_DATA_TYPE_BYTES	5	
ATTRIBUTE_DATA_TYPE_IDENTITY	100	
ATTRIBUTE_DATA_TYPE_RESOURCE	101	

DiscoveryMode

Name	Number	Description
DISCOVERY_MODE_UNKNOWN	0	
DISCOVERY_MODE_TRUSTED	1	Only local identities with explicit policy PERMIT can discover.
DISCOVERY_MODE_PUBLIC	2	Any bound local identity can discover.
DISCOVERY_MODE_TRUSTED_GLOBAL	3	Only identities with explicit policy PERMIT can discover, and the service will be available to local and non-local (Relayed) clients.
DISCOVERY_MODE_PUBLIC_GLOBAL	4	Any bound identity can discover, and the service will be available to local and non-local (Relayed) clients.

OnlineStatus

Name	Number	Description
ONLINE_STATUS_UNKNOWN	0	
ONLINE_STATUS_ONLINE	1	
ONLINE_STATUS_OFFLINE	2	

PolicyDecision

@todo currently this must align with AuthorisationDecision enum in policy library, but some of the values are irrelevant outside of the public API so we need a public-facing enum and some translation.

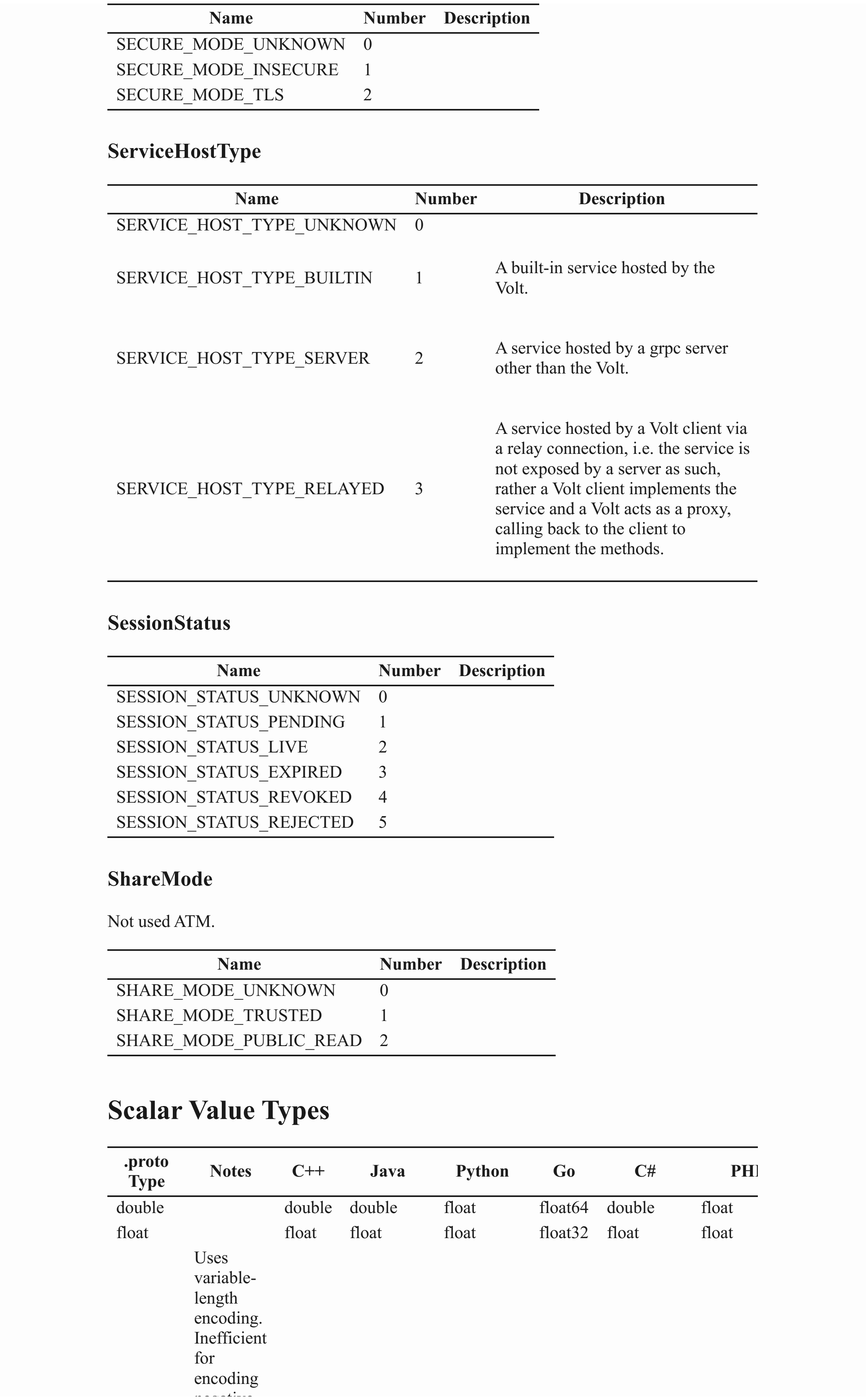
Name	Number	Description
POLICY_DECISION_UNKNOWN	0	
POLICY_DECISION_PROMPT	1	
POLICY_DECISION_PERMIT	2	
POLICY_DECISION_DENY	3	
POLICY_DECISION_INDETERMINATE	4	
POLICY_DECISION_NOT_APPLICABLE	5	
POLICY_DECISION_APPLICABLE	6	
POLICY_DECISION_PENDING	7	

ResourceStatus

Not used ATM.

Name	Number	Description
RESOURCE_STATUS_UNKNOWN	0	
RESOURCE_STATUS_LIVE	1	
RESOURCE_STATUS_INACTIVE	2	
RESOURCE_STATUS_DELETED	999	

SecureMode



Name	Number	Description
SECURE_MODE_UNKNOWN	0	
SECURE_MODE_INSECURE	1	
SECURE_MODE_TLS	2	

ServiceHostType

Name	Number	Description
SERVICE_HOST_TYPE_UNKNOWN	0	
SERVICE_HOST_TYPE_BUILTIN	1	A built-in service hosted by the Volt.
SERVICE_HOST_TYPE_SERVER	2	A service hosted by a grpc server other than the Volt.
SERVICE_HOST_TYPE_RELAYED	3	A service hosted by a Volt client via a relay connection, i.e. the service is not exposed by a server as such, rather a Volt client implements the service and a Volt acts as a proxy, calling back to the client to implement the methods.

SessionStatus

Name	Number	Description
SESSION_STATUS_UNKNOWN	0	
SESSION_STATUS_PENDING	1	
SESSION_STATUS_LIVE	2	
SESSION_STATUS_EXPIRED	3	
SESSION_STATUS_REVOKED	4	
SESSION_STATUS_REJECTED	5	

ShareMode

Not used ATM.

Name	Number	Description
SHARE_MODE_UNKNOWN	0	
SHARE_MODE_TRUSTED	1	
SHARE_MODE_PUBLIC_READ	2	

Scalar Value Types

.proto Type	Notes	C++	Java	Python	Go	C#	PHI
double		double	double	float	float64	double	float
float		float	float	float	float32	float	float
	Uses variable-length encoding. Inefficient for encoding						

proto Type	Notes	C++	Java	Python	Go	C#	PH
int32	negative numbers – if your field is likely to have negative values, use sint32 instead.	int32	int	in	int32	int	integ
int64	Uses variable-length encoding. Inefficient for encoding negative numbers – if your field is likely to have negative values, use sint64 instead.	int64	long	int/long	int64	long	integer/s
uint32	Uses variable-length encoding.	uint32	int	int/long	uint32	uint	integer
uint64	Uses variable-length encoding.	uint64	long	int/long	uint64	ulong	integer/s
sint32	Uses variable-length encoding. Signed int value. These more efficiently encode negative numbers than regular int32s.	int32	int	int	int32	int	integer
sint64	Uses variable-length encoding. Signed int value. These more efficiently encode negative numbers than regular int64s.	int64	long	int/long	int64	long	integer/s
	Always four bytes.						

.proto Type	Notes	C++	Java	Python	Go	C#	PH
fixed32	More efficient than uint32 if values are often greater than 2^28.	uint32	int	int	uint32	uint	integer
fixed64	Always eight bytes. More efficient than uint64 if values are often greater than 2^56.	uint64	long	int/long	uint64	ulong	integer/s
sfixed32	Always four bytes.	int32	int	int	int32	int	integer
sfixed64	Always eight bytes.	int64	long	int/long	int64	long	integer/s
bool		bool	boolean	boolean	bool	bool	boolean
string	A string must always contain UTF-8 encoded or 7-bit ASCII text.	string	String	str/unicode	string	string	string
bytes	May contain any arbitrary sequence of bytes.	string	ByteString	str	[]byte	ByteString	string

Skip to Content

tdx Volt

putting you in charge

☀☾

Toggle sidebar

- Getting Started
  - Welcome
  - Quick Start
- Concepts
  - Fundamentals

- Identity
- DID registry
- Verifiable credentials
- Policy
- Resource
- File
- Database
- Wire
- Relay
- Key strategy

- ## How to...

- Create a Volt
- Start a Volt
- Connect to a Volt
- Upload files
- Download files
- Publish to wire
- Subscribe to wire
- Create a database
- Execute SQL
- Import data
- Establish a connect stream
- Approve authenticate request

- ## Clients

- Command line
- Native / C++
- Web
- NodeJS

- ## Reference

- Best practice
- Battery
- Configuration
- Connection
- Connect stream
- Logging
- PKCS#11

- ## API

- Discovery API
- File API
- SqliteDatabase API
- SqliteServer API
- SSI API
- Sync API
- Relay API
- Volt API
- Wire API

- ## Utilities

- protoDbSync
- sqliteServer
- wireTransform

- ## FAQ
    - Questions
- ## Coming soon
    - Roadmap
- ☀☾

Contents

Contents

SsiAPI

Top

DeleteDID()

Delete a DID document or all DID documents originating from a given Volt.

Requires `volt:delete-did` API privilege.

Request: DeleteDIDRequest
Response: DeleteDIDResponse

GetDIDRegistryUpdates()

Get all updates to the DID registry since the specified timestamp.

Internal use only.

Request: GetDIDRegistryUpdatesRequest
Response: GetDIDRegistryUpdatesResponse

ImportCredential()

Import a verifiable credential.

Request: ImportCredentialRequest
Response: ImportCredentialResponse

ParseCredential()

Parse a verifiable credential from a URL or a verifiable presentation.

Request: ParseCredentialRequest
Response: ParseCredentialResponse

ResolveDID()

Resolve a DID to a DID document.

Request: ResolveDIDRequest
Response: ResolveDIDResponse

RegisterDIDDocument()

Register a DID document.

This is intended for use by the DID registry when synchronising with other registries.

To register a new DID document, it is recommended to use the VoltAPI Authenticate method.

Request: RegisterDIDDocumentRequest
Response: RegisterDIDDocumentResponse

SaveCredential()

Save a verifiable credential.

Request: SaveCredentialRequest
Response: SaveCredentialResponse

SearchDIDRegistry()

Search the DID registry.

Request: SearchDIDRegistryRequest
Response: SearchDIDRegistryResponse

DeleteDIDRequest

Field	Type	Description
did	string	The id of the identity to delete.
origin_volt	string	Force delete of all DIDs owned by the specified Volt.
key_passphrase	string	Optional passphrase of the DID controller key. Required if the key is encrypted.

DeleteDIDResponse

Field	Type	Description
status	Status	Details of any error that occurred on the call.

GetDIDRegistryUpdatesRequest

Field	Type	Description
since_id	uint64	The id of the last update received. All updates since this id will be returned, limited to the maximum number of updates specified in the request. To fully synchronise, clients should continue calling this method until the response contains no updates. If this is the first call, then this should be set to 0.
max_updates	uint32	The maximum number of updates to return, defaults to 1000.
origin_volt	string	Filter by origin volt.

GetDIDRegistryUpdatesResponse

Field	Type	Description
status	Status	Details of any error that occurred on the call.
update	DIDRegistryUpdate repeated	The updates.

ImportCredentialRequest

Field	Type	Description
id	string	Optional id of existing credential to import into. If not specified, a new credential will be created.
json	string	The JSON representation of the credential.
create_in_parent_id	string	Optional id of the folder resource to save the credential in. This is ignored if the id field is specified.
description	string	Optional description of the credential.

ImportCredentialResponse

Field	Type	Description
status	Status	Details of any error that occurred on the call.
id	string	The id assigned to the credential.

ParseCredentialRequest

Field	Type	Description
verifiable_presentation	VerifiablePresentation	A presentation of the verifiable credential. The presentation doesn't need to be signed, but if it is, the signature will be verified using the public key provided in the request.
url	string	A URL to a verifiable credential. Not yet implemented.
presentation_public_key	string	Optional public key to use to verify the presentation signature.

ParseCredentialResponse

Field	Type	Description
status	Status	Details of any error that occurred on the call.
verifiable_credential	VerifiableCredential	The parsed credential details.

RegisterDIDDocumentRequest

Field	Type	Description
create	bool	
did_update	DIDRegistryUpdate	The DID document to save.
update_signature	string	When updating an existing DID document, it is necessary to include a signature of the document JSON, signed by the DID document's current owner.

RegisterDIDDocumentResponse

Field	Type	Description
status	Status	Details of any error that occurred on the call.
did_document	DIDRegistryUpdate	

ResolveDIDRequest

Field	Type	Description
did	string	The DID to resolve.
include_registries	bool	Set to search all known registries rather than just the local registry.

ResolveDIDResponse

Field	Type	Description
status	Status	Details of any error that occurred on the call.
did_document	string	The JSON representation of the DID document.
update_signature	string	The signature used to save this version of the DID document.
origin_volt	string	The DID of the Volt that saved this version of the DID document.
description	string	Optional description associated with the DID document.

SaveCredentialRequest

Field	Type	Description
description	string	A human-readable description of the credential.
verifiable_credential	VerifiableCredential	Details of the credential to save.
create_in_parent_id	string	Optional id of the folder resource to save the credential in.

SaveCredentialResponse

Field	Type	Description
status	Status	Details of any error that occurred on the call.
id	string	The id assigned to the credential.
json	string	The JSON representation of the credential.

SearchDIDRegistryRequest

Field	Type	Description
did_filter	string	Filter by DID itself. This is a prefix match, so searching for 'did:volt:123' will match 'did:volt:1234'. You can also exclude the 'did:volt:' prefix.
description_filter	string	Filter by the (optional) description attached to the DID document. This will match any DID document whose description contains the specified string, for example 'John' will match 'John Smith', 'Elton John' and 'Jasper Johns-Frederick'.
origin_filter	string	Filter by the origin Volt. This will match any DID document whose origin Volt matches exactly the given DID.
page_number	uint32	The page number to retrieve, defaults to 1.
page_size	uint32	The number of results per page, defaults to 100.

SearchDIDRegistryResponse

Field	Type	Description
status	Status	Details of any error that occurred on the call.
did_document	DIDRegistryUpdate repeated	The DID documents that matched the search criteria.

DIDRegistryUpdate

Field	Type	Description
id	uint64	The id of the update. Reserved for internal use.
did	string	The id of the identity.
operation	string	The type of update, either "add", "update" or "delete".
document	string	The DID document contained in the update.
hash	string	The hash of DID document. This is for internal use in comparisons.
update_signature	string	The signature of the update.
timestamp	uint64	The timestamp of this update.
vector_clock	DIDRegistryUpdate.VectorClockEntry repeated	The vector clocks for this DID. The vector clocks are a map of the peer ID to the id of the last update received for this DID.
origin_volt	string	The id of the Volt that first created this DID.
description	string	Optional description of this document, this is not part of the DID document or signature.

DIDRegistryUpdate.VectorClockEntry

Field	Type	Description
key	string	
value	uint64	

VerifiableCredential

Field	Type	Description
id	string	The credential id. Leave empty when creating a new credential.
status	string	The credential status, either "pending", "verified", or "revoked".
type	string repeated	The credential types.
issuer_id	string	The DID of the issuer.
subject_json	string	The JSON of the credential subject.
json	string	The full JSON of the credential.

VerifiablePresentation

Field	Type	Description
credential_json	string	The credential JSON.
signature	string	A signature of the credential JSON. The signature should usually be that of the credential subject.

Status

Field	Type	Description
code	int32	A simple error code that can be easily handled by the client. Mirrors the grpc StatusCode enum, 0 => OK
message	string	A developer-facing human-readable error message in English. It should both explain the error and offer an actionable resolution to it.
description	string	Long form error description.

Scalar Value Types

.proto Type	Notes	C++	Java	Python	Go	C#	PHI
double		double	double	float	float64	double	float
float		float	float	float	float32	float	float
	Uses variable-						

.proto Type	Notes	C++	Java	Python	Go	C#	PH
int32	length encoding. Inefficient for encoding negative numbers – if your field is likely to have negative values, use sint32 instead.	int32	int	int	int32	int	integer
int64	Uses variable-length encoding. Inefficient for encoding negative numbers – if your field is likely to have negative values, use sint64 instead.	int64	long	int/long	int64	long	integer/s
uint32	Uses variable-length encoding.	uint32	int	int/long	uint32	uint	integer
uint64	Uses variable-length encoding.	uint64	long	int/long	uint64	ulong	integer/s
sint32	Uses variable-length encoding. Signed int value. These more efficiently encode negative numbers than regular int32s.	int32	int	int	int32	int	integer
sint64	Uses variable-length encoding. Signed int value. These more efficiently encode negative numbers than	int64	long	int/long	int64	long	integer/s

regular int64s.

.proto Type	Notes	C++	Java	Python	Go	C#	PHI
fixed32	Always four bytes. More efficient than uint32 if values are often greater than 2^28.	uint32	int	int	uint32	uint	integer
fixed64	Always eight bytes. More efficient than uint64 if values are often greater than 2^56.	uint64	long	int/long	uint64	ulong	integer/s
sfixed32	Always four bytes.	int32	int	int	int32	int	integer
sfixed64	Always eight bytes.	int64	long	int/long	int64	long	integer/s
bool		bool	boolean	boolean	bool	bool	boolean
string	A string must always contain UTF-8 encoded or 7-bit ASCII text.	string	String	str/unicode	string	string	string
bytes	May contain any arbitrary sequence of bytes.	string	ByteString	str	[]byte	ByteString	string

Skip to Content

tdx Volt

putting you in charge
☀ ☾
Toggle sidebar

- ## Getting Started
  - Welcome
  - Quick Start

- **Concepts**

  - Fundamentals
  - Identity
  - DID registry
  - Verifiable credentials
  - Policy
  - Resource
  - File
  - Database
  - Wire
  - Relay
  - Key strategy

- **How to...**

  - Create a Volt
  - Start a Volt
  - Connect to a Volt
  - Upload files
  - Download files
  - Publish to wire
  - Subscribe to wire
  - Create a database
  - Execute SQL
  - Import data
  - Establish a connect stream
  - Approve authenticate request

- **Clients**

  - Command line
  - Native / C++
  - Web
  - NodeJS

- **Reference**

  - Best practice
  - Battery
  - Configuration
  - Connection
  - Connect stream
  - Logging
  - PKCS#11

- **API**

  - Discovery API
  - File API
  - SqliteDatabase API
  - SqliteServer API
  - SSI API
  - Sync API
  - Relay API
  - Volt API
  - Wire API

- **Utilities**

  - protoDbSync

  - sqliteServer
  - wireTransform

- **FAQ**
  - Questions

- **Coming soon**
  - Roadmap
- ☀☾

Contents

Contents

FileAPI

Top

The File API exposes basic file management functions.

DownloadFile()

Download from file resource.

Request: DownloadFileRequest
Response: streaming DownloadFileResponse

GetFile()

Get file resource metadata.

Request: GetFileRequest
Response: GetFileResponse

GetFileContent()

Get the content of a file.

Note this rpc will fail if the size of the file content is greater than 64MB, in which case use DownloadFile instead.

Request: GetFileContentRequest
Response: GetFileContentResponse

GetFileDescendants()

Get the file resource metadata of all descendants of a given file resource.

Request: GetFileDescendantsRequest
Response: GetFileDescendantsResponse

SetFileContent()

Set the content of a file.

Note this rpc will fail if the size of the file content is greater than 64MB, in which case use UploadFile instead.

Request: SetFileContentRequest
Response: SetFileContentResponse

UploadFile()

Upload data to a file resource.

Request: streaming UploadFileRequest
Response: streaming UploadFileResponse

DownloadFileRequest

Field	Type	Description
resource_id	string	The id of the resource to download.
file_path	string	This is required for linked folders, and represents the relative path to the source file from the base folder.

DownloadFileResponse

The response stream will contain one or more of the following messages.

Each response message will contain one of the following fields.

Field	Type	Description
block	bytes	A chunk of file data.
status	Status	A status will be sent when the file is completely downloaded, or if an error occurs.

GetFileContentRequest

Field	Type	Description
resource_id	string	The resource id of the base folder.
file_path	string	Optional - the path to the file, relative to the base folder. Required for linked folders.

GetFileContentResponse

Field	Type	Description
status	Status	Details of any error that occurred on the call.
content	bytes	The file content.

GetFileDescendantsRequest

Field	Type	Description
resource_id	string	The resource id of the base folder.
file_path	string	For linked files, this is the relative path to the 'parent' file from the base folder.
extension	string	Optional - only match descendants of the given kind.
descendant_id	string	Optional - can be used to determine if a resource is a descendant.

GetFileDescendantsResponse

Field	Type	Description
status	Status	Details of any error that occurred on the call.
descendant	File repeated	The list of descendants.

GetFileRequest

Field	Type	Description
resource_id	string	The resource id of the base folder.
file_path	string	Optional - the path to the file, relative to the base folder. Required for linked folders.

GetFileResponse

Field	Type	Description
status	Status	Details of any error that occurred on the call.
file	File	The file metadata.

SetFileContentRequest

Field	Type	Description
resource_id	string	The resource id of the base folder.
file_path	string	Optional - the path to the file, relative to the base folder. Required for linked folders.
store_name	string	Optional store name to use. If specified, this will be used to extract the extension to set as a resource 'kind', e.g. tdx:ext:json. If not specified, the extension will be taken from the name of the target resource.
content	bytes	

SetFileContentResponse

Field	Type	Description
status	Status	Details of any error that occurred on the call.

UploadFileRequest

One of the following fields must be present.

Field	Type	Description
start	UploadFileStart	Describes the file upload, should only be sent as first message.
block	bytes	The next block of data.

UploadFileResponse

Field	Type	Description
status	Status	Details of any error that occurred on the call.
back_off	bool	Reserved for internal use.

UploadFileStart

Field	Type	Description
resource_id	string	The resource to upload to, required.
store_name	string	Optional store name to use. If specified, this will be used to extract the extension to set as a resource 'kind', e.g. tdx:ext:json. If not specified, the extension will be taken from the name of the target resource.
streaming_mode	bool	Optionally set streaming mode. When in streaming mode, data is written directly to the resource. Otherwise, data is written to a temporary file and then copied over once the upload completes successfully.
truncate	bool	Optional, truncates the file prior to beginning the upload. This is only really relevant if 'streaming_mode' is set.
eager_flush	bool	Optional, buffer will flush after each write.

Status

Field	Type	Description
code	int32	A simple error code that can be easily handled by the client. Mirrors the grpc StatusCode enum, 0 => OK
message	string	A developer-facing human-readable error message in English. It should both explain the error and offer an actionable resolution to it.
description	string	Long form error description.

File

Field	Type	Description
file_path	string	
absolute_path	string	
file_name	string	
extension	string	
size	uint64	
media_type	string	
is_directory	bool	
modified	uint64	
resource_id	string	
owner_resource_id	string	

Scalar Value Types

.proto Type	Notes	C++	Java	Python	Go	C#	PH
double		double	double	float	float64	double	float
float		float	float	float	float32	float	float
int32	Uses variable-length encoding. Inefficient for encoding negative numbers – if your field is likely to have negative values, use sint32 instead.	int32	int	int	int32	int	integer
int64	Uses variable-length encoding. Inefficient for encoding negative numbers – if your field is likely to have negative values, use sint64 instead.	int64	long	int/long	int64	long	integer/
uint32	Uses variable-length encoding.	uint32	int	int/long	uint32	uint	integer
uint64	Uses variable-length encoding.	uint64	long	int/long	uint64	ulong	integer/
	Uses variable-length						

.proto Type	Notes	C++	Java	Python	Go	C#	PH
sint32	encoding. Signed int value. These more efficiently encode negative numbers than regular int32s.	int32	int	int	int32	int	integer
sint64	Uses variable-length encoding. Signed int value. These more efficiently encode negative numbers than regular int64s.	int64	long	int/long	int64	long	integer/s
fixed32	Always four bytes. More efficient than uint32 if values are often greater than 2^28.	uint32	int	int	uint32	uint	integer
fixed64	Always eight bytes. More efficient than uint64 if values are often greater than 2^56.	uint64	long	int/long	uint64	ulong	integer/s
sfixed32	Always four bytes.	int32	int	int	int32	int	integer
sfixed64	Always eight bytes.	int64	long	int/long	int64	long	integer/s
bool		bool	boolean	boolean	bool	bool	boolean
string	A string must always contain UTF-8 encoded or 7-bit ASCII text.	string	String	str/unicode	string	string	string

.proto Type	Notes	C++	Java	Python	Go	C#	PH
bytes	May contain any arbitrary sequence of bytes.	string	ByteString	str	[]byte	ByteString	string

Skip to Content

tdx Volt

putting you in charge

☀☾

Toggle sidebar

- ## Getting Started
  - Welcome
  - Quick Start
- ## Concepts
  - Fundamentals
  - Identity
  - DID registry
  - Verifiable credentials
  - Policy
  - Resource
  - File
  - Database
  - Wire
  - Relay
  - Key strategy
- ## How to...
  - Create a Volt
  - Start a Volt
  - Connect to a Volt
  - Upload files
  - Download files
  - Publish to wire
  - Subscribe to wire
  - Create a database
  - Execute SQL
  - Import data
  - Establish a connect stream
  - Approve authenticate request
- ## Clients
  - Command line
  - Native / C++
  - Web
  - NodeJS
- ## Reference

- Best practice
- Battery
- Configuration
- Connection
- Connect stream
- Logging
- PKCS#11

- **API**
  - Discovery API
  - File API
  - SqliteDatabase API
  - SqliteServer API
  - SSI API
  - Sync API
  - Relay API
  - Volt API
  - Wire API

- **Utilities**
  - protoDbSync
  - sqliteServer
  - wireTransform

- **FAQ**
  - Questions

- **Coming soon**
  - Roadmap
- ☀☾

Roadmap

cli

Surfacing of more sections of the **tdx Volt** API are upcoming, specifically:

- database support - ability to create databases and execute SQL

fusebox

- Support multiple selection (and hence bulk operations) in resource folder

core

- Generic verifiable credential support
- Resource aliases

Skip to Content

tdx Volt

putting you in charge
☀ ☾
Toggle sidebar

- ## Getting Started
  - Welcome
  - Quick Start
- ## Concepts
  - Fundamentals
  - Identity
  - DID registry
  - Verifiable credentials
  - Policy
  - Resource
  - File
  - Database
  - Wire
  - Relay
  - Key strategy
- ## How to...
  - Create a Volt
  - Start a Volt
  - Connect to a Volt
  - Upload files
  - Download files
  - Publish to wire
  - Subscribe to wire
  - Create a database
  - Execute SQL
  - Import data
  - Establish a connect stream
  - Approve authenticate request
- ## Clients
  - Command line
  - Native / C++
  - Web
  - NodeJS
- ## Reference
  - Best practice
  - Battery
  - Configuration
  - Connection
  - Connect stream
  - Logging
  - PKCS#11
- ## API

  - Discovery API
  - File API
  - SqliteDatabase API
  - SqliteServer API
  - SSI API
  - Sync API
  - Relay API
  - Volt API
  - Wire API

- **Utilities**
  - protoDbSync
  - sqliteServer
  - wireTransform

- **FAQ**
  - Questions

- **Coming soon**
  - Roadmap
- ☀ ☾

Introduction

tdx Volt is a platform that provides the ability to **securely** share services, structured data, files, and analytics in a peer-to-peer, decentralised fashion.

A **tdx Volt** gives you complete control over access to all of your services, data and resources.

Features

Some of the features include:

- True end-to-end encryption between peers - no intermediate server.
- Data is encrypted at rest.
- Securely access the services and data in your **tdx Volt** from anywhere.
- Protect your data using a security policy that gives you fine-grained, complete control over who can access your data.
- Connect to friends and colleagues and share services, files and data.
- Client libraries for Javascript (Web and NodeJS), Python and C++.
- Run on many platforms, including Windows, MacOS (Intel and Apple silicon), Linux, RPi, Omnia Turris.

Get Started

OR

Learn More

Skip to Content

tdx Volt

putting you in charge
☀☾
Toggle sidebar

- **Getting Started**
  - Welcome
  - Quick Start
- **Concepts**
  - Fundamentals
  - Identity
  - DID registry
  - Verifiable credentials
  - Policy
  - Resource
  - File
  - Database
  - Wire
  - Relay
  - Key strategy
- **How to...**
  - Create a Volt
  - Start a Volt
  - Connect to a Volt
  - Upload files
  - Download files
  - Publish to wire
  - Subscribe to wire
  - Create a database
  - Execute SQL
  - Import data
  - Establish a connect stream
  - Approve authenticate request
- **Clients**
  - Command line
  - Native / C++
  - Web
  - NodeJS
- **Reference**
  - Best practice
  - Battery
  - Configuration
  - Connection
  - Connect stream
  - Logging
  - PKCS#11
- **API**
  - Discovery API

  - File API
  - SqliteDatabase API
  - SqliteServer API
  - SSI API
  - Sync API
  - Relay API
  - Volt API
  - Wire API

- **Utilities**
  - protoDbSync
  - sqliteServer
  - wireTransform

- **FAQ**
  - Questions

- **Coming soon**
  - Roadmap
- ☀☾

Connect stream

A client can choose to establish what's known as a **connect stream** with the Volt. For clients that are simply consuming services, this isn't always necessary. However if a client wishes to register a service with Volt for consumption by other clients, a **connect stream** is required to ensure that the Volt can reliably determine when services are online and reachable.

A **connect stream** can also be useful in consumer scenarios, because it makes things like reconnection and retries easier to manage. For example, if a client wishes to set up a reliable subscription to a Volt wire, the 'connected' event of a **connect stream** can be used to signal the start of the subscription. If the **connect stream** drops for whatever reason, a 'disconnected' event will be sent and the client library will automatically attempt periodic reconnection. Once the **connect stream** is re-established, the 'connect' event is fired again and the client can restart the subscription.

Service registration

Clients wishing to register a service(s) with the Volt must first establish a **connect stream**. The Volt uses the **connect stream** to control the lifetime of the services. If the **connect stream** is dropped or explicitly closed, the Volt will move all the services registered by that client offline.

Both client libraries provide interfaces that make **connect stream** creation and management relatively straight forward. See the How to establish a **connect stream** section for more information.

Remote invocation

As well as service registration management, **connect streams** are used internally by the Volt to support the Relay concept. When a Volt establishes a Relay connection to another Volt, this is done using a **connect stream**. See the InvokeRequest and InvokeResponse message types for more details.

This concept could be extended to enable any client to support remote invocation of arbitrary functions, i.e. it is not restricted to Volt endpoints. For example, a web client could establish a

connect stream using the Javascript Web API , and upon receipt of a ConnectResponse message with an InvokeRequest payload respond with an appropriate InvokeResponse.

Skip to Content

tdx Volt

putting you in charge

☀☾

Toggle sidebar

- **Getting Started**
  - Welcome
  - Quick Start
- **Concepts**
  - Fundamentals
  - Identity
  - DID registry
  - Verifiable credentials
  - Policy
  - Resource
  - File
  - Database
  - Wire
  - Relay
  - Key strategy
- **How to...**
  - Create a Volt
  - Start a Volt
  - Connect to a Volt
  - Upload files
  - Download files
  - Publish to wire
  - Subscribe to wire
  - Create a database
  - Execute SQL
  - Import data
  - Establish a connect stream
  - Approve authenticate request
- **Clients**
  - Command line
  - Native / C++
  - Web
  - NodeJS
- **Reference**
  - Best practice
  - Battery
  - Configuration
  - Connection

  - Connect stream
  - Logging
  - PKCS#11

- **API**

  - Discovery API
  - File API
  - SqliteDatabase API
  - SqliteServer API
  - SSI API
  - Sync API
  - Relay API
  - Volt API
  - Wire API

- **Utilities**

  - protoDbSync
  - sqliteServer
  - wireTransform

- **FAQ**

  - Questions

- **Coming soon**

  - Roadmap

- ☀☾

Battery

A Battery is primarily a convenience structure for storing **tdx Volt** configurations and potentially grouping together two or more Volts.

For example, a router may have a Battery that contains a **tdx Volt** for each member of the household, or an enterprise server may use multiple Batteries as a means of grouping various subscribers or similar.

Storage

A Battery stores the Volt configuration of each **tdx Volt** that it contains.

As well as the **tdx Volt** configurations, the Battery also stores details of the key strategy used by each Volt.

The Battery storage is implemented as an SqlCipher database, a secure, encrypted-at-rest version of Sqlite.

Battery key strategy

If a **tdx Volt** is using the ‘Battery’ key strategy, the **tdx Volt** key will be stored along with the **tdx Volt** configuration in the Battery storage.

If the Battery is password protected, the key will be stored in the encrypted Battery database, which in turn is protected by the Battery password.

Battery password protection

The user can choose whether to protect the Battery with a passphrase or not.

It is recommended to do so, but is not strictly necessary if the Volts it contains will be using key strategies other than the ‘Battery’ strategy.

The Battery password is used to encrypt the storage database. It is not stored anywhere and relies on the user remembering it. It is not currently possible to change or reset the Battery password without losing all data.

Location

All files relating to a given Battery are located in a named directory, which can be specified by the user at creation time.

A default Battery is automatically created in a sub-folder called `TDXVolt` of the `Documents` folder, which is dependent on the current user and operating system.

Skip to Content

tdx Volt

putting you in charge

☀☾

Toggle sidebar

- **Getting Started**
  - Welcome
  - Quick Start
- **Concepts**
  - Fundamentals
  - Identity
  - DID registry
  - Verifiable credentials
  - Policy
  - Resource
  - File
  - Database
  - Wire
  - Relay
  - Key strategy
- **How to...**
  - Create a Volt
  - Start a Volt
  - Connect to a Volt
  - Upload files
  - Download files
  - Publish to wire
  - Subscribe to wire
  - Create a database
  - Execute SQL
  - Import data
  - Establish a connect stream

  - Approve authenticate request

- **Clients**
  - Command line
  - Native / C++
  - Web
  - NodeJS
- **Reference**
  - Best practice
  - Battery
  - Configuration
  - Connection
  - Connect stream
  - Logging
  - PKCS#11
- **API**
  - Discovery API
  - File API
  - SqliteDatabase API
  - SqliteServer API
  - SSI API
  - Sync API
  - Relay API
  - Volt API
  - Wire API
- **Utilities**
  - protoDbSync
  - sqliteServer
  - wireTransform
- **FAQ**
  - Questions
- **Coming soon**
  - Roadmap
- ☀☾

PKCS#11

The **tdx Volt** can be configured to use a PKCS#11-compliant hardware security module (HSM) to secure the **tdx Volt** key.

This page will describe the steps involved in using a YubiKey 5 hardware security module to secure the **tdx Volt** key.

This example uses the Personal Identity Verification (PIV) application on the YubiKey. PIV is a standard for smart cards used for secure authentication.

This document is intended as a quick-start guide. You should read the YubiKey documentation for more detailed information about how to securely configure your YubiKey.

Prerequisites

- A YubiKey 5 hardware security module
- The YubiKey PIN (the default is `123456`)
- The YubiKey management software (`ykman`) installed on your computer
- You may also need to install the OpenSC software to enable PKCS#11 support, depending on your operating system

Installation

The YubiKey needs to be configured with a PIV key pair and a certificate. The key pair will be used to secure the **tdx Volt** key.

We will use the `ykman` command-line tool to create the key pair and configure the YubiKey. The `ykman` tool is available for Windows, macOS, and Linux, see the details here.

There is also a graphical user interface available for Windows and macOS, see Yubico website for more information.

Generate a key pair

The following command will generate a key pair on the YubiKey. The private key will be stored in slot `9a` on the YubiKey, and the public key will be written to a file on the local file system called `yubi-public.pem`.

Slot 9a is typically used for the PIV authentication key, see the Appendix below for more information.

The following command will generate the key pair:

```
ykman piv keys generate 9a ./yubi-public.pem
```

Terminal window

We also need to generate a certificate for the key pair. The following command will generate a self-signed certificate and store it in the slot on the device. Replace the `Alice` with your own name or the subject of the certificate holder.

```
ykman piv certificates generate -s "CN=Alice" 9a ./yubi-public.pem
```

Terminal window

See the YubiKey documentation for full details of how to configure the key pair generation, including algorithm selection and key size etc.

Configure the tdx Volt

To create a **tdx Volt** that is secured by the YubiKey, you need to specify the PKCS#11 key strategy when creating the **tdx Volt**. See the Create a Volt page for more information.

Appendix

You will need to know the slot number of the PIV key pair on your YubiKey.

Slot `9a` is typically used for the PIV authentication key. You can confirm that the key pair is available by running the `pkcs11-tool` command, which is part of the OpenSC software:

```
pkcs11-tool -v -O
```

Terminal window

You should see output similar to the following, look for the `PIV AUTH pubkey` label and note the `ID` field value:

```
Using slot 0 with a present token (0x0)Public Key Object;
RSA 2048 bits  label:      PIV AUTH pubkey  ID:        01
Usage:      encrypt, verify, verifyRecover, wrap  Access:
none
```

Terminal window

If this isn't the case, you will need to use the slot number shown rather than `01` when configuring your **tdx Volt**.

Skip to Content

tdx Volt

putting you in charge

☀ ☾

Toggle sidebar

- **Getting Started**
  - Welcome
  - Quick Start
- **Concepts**
  - Fundamentals
  - Identity
  - DID registry
  - Verifiable credentials
  - Policy
  - Resource
  - File
  - Database
  - Wire
  - Relay
  - Key strategy
- **How to...**
  - Create a Volt
  - Start a Volt
  - Connect to a Volt
  - Upload files
  - Download files
  - Publish to wire
  - Subscribe to wire
  - Create a database
  - Execute SQL
  - Import data
  - Establish a connect stream
  - Approve authenticate request
- **Clients**
  - Command line
  - Native / C++
  - Web

  - NodeJS

- ## Reference

  - Best practice
  - Battery
  - Configuration
  - Connection
  - Connect stream
  - Logging
  - PKCS#11

- ## API

  - Discovery API
  - File API
  - SqliteDatabase API
  - SqliteServer API
  - SSI API
  - Sync API
  - Relay API
  - Volt API
  - Wire API

- ## Utilities

  - protoDbSync
  - sqliteServer
  - wireTransform

- ## FAQ

  - Questions

- ## Coming soon

  - Roadmap
- ☀☾

Configuration

The **tdx Volt** configuration is used by clients that want to connect to a **tdx Volt** and access resources or services.

It contains all the information necessary for a client to identify and locate the **tdx Volt** management API and establish a connection.

The full definition of the **tdx Volt** configuration is show in the Appendix below.

Note that the client will also need to supply some credentials in order to identify itself to the Volt. This is described in Volt Connection.

The **tdx Volt** configuration does not contain any sensitive information that may compromise its security. It is safe to distribute the **tdx Volt** configuration, either publicly or only to those you would like to connect to your Volt. Just because somebody possesses your **tdx Volt** configuration does not mean they can access the Volt.

Obtaining a tdx Volt configuration

There are several methods of obtaining the configuration document for a given Volt.

Out of band

A likely scenario is that you will obtain a **tdx Volt** configuration via some out-of-band means such as email.

If Alice is happy for Bob to connect to her **tdx Volt** she will email Bob her **tdx Volt** configuration.

fusebox

The easiest option if you are owner of the **tdx Volt** is to use the **fusebox** application. On the main **tdx Volt** screen there is a 'configuration' field in the right-hand side bar - see the image below.

If you click on the 'copy' icon at the right-hand edge the **tdx Volt** configuration will be copied to the clipboard.

Command line

Another method of obtaining a **tdx Volt** configuration is via the command line interface. The `config` command will list all the configured Volts:

```
./volt config
```

Terminal window

Once you know the `id` of the Volt, you can obtain the full **tdx Volt** configuration by specifying the `id` to the `config` command:

```
./volt config -i <volt id>
```

Terminal window

If the **tdx Volt** has an alias, you can use that instead of the `id`:

```
./volt config -i @<alias>
```

Terminal window

Web portal

This is under review and may be removed shortly.

If you would like to connect to a remote **tdx Volt** via a cloud tunnel, you can copy the configuration from the **TDX Cloud** web portal.

Select the **tdx Volt** in the drop-down list at the top of the page and then click on the 'Command' button, followed by 'copy configuration' menu item.

Peer to Peer discovery

This functionality is experimental and subject to change.

All Batteries implement and expose the tdx.api.volt.v1.DiscoveryAPI service. This can be used to discover all Volts currently running on the Battery.

The **fusebox** also implements a rudimentary discovery function in the form of scanning all IP addresses on the local network, looking for instances of the DisoveryAPI.

Remote discovery

All Relay Volt connections will implement and expose the `tdx.api.relay.v1.RelayAPI` service. The endpoint GetVoltEndpoint can be used to list all Volts that the currently authenticated client has access to.

Examples

A minimal example of a **tdx Volt** configuration is shown below.

```
{  "id": "5160cbc4-8fd5-4f92-a0c9-589183cf822e",  "address":
"192.168.1.71:65247",  "ca_pem": "-----BEGIN CERTIFICATE----
-
\r\nMIIDtzCCAp+gAwIBAgIGAXslHHq0MA0GCSqGSIb3DQEBCwUAMH0xGzAZE
-----END CERTIFICATE-----\r\n"}
```

A full **tdx Volt** configuration is show below, note that many of the fields are included for convenience:

```
{  "id": "5160cbc4-8fd5-4f92-a0c9-589183cf822e",
"display_name": "Local 1",  "address": "192.168.1.71:65247",
"public_key": "-----BEGIN PUBLIC KEY-----
\r\nMIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAzBHtrgPpsFHdF
-----END PUBLIC KEY-----\r\n",  "fingerprint":
"EGWTGHuqxhnGDQb29GhfUhJXXfhjnGZ2rUhMVR9DGyUH",  "ca_pem":
"-----BEGIN CERTIFICATE-----
\r\nMIIDtzCCAp+gAwIBAgIGAXslHHq0MA0GCSqGSIb3DQEBCwUAMH0xGzAZE
-----END CERTIFICATE-----\r\n",  "owner_credential":
"eyJAY29udGV4dCI6WyJodHRwczovL3d3dy53My5vcmcvMjAxOC9jcmVkZW5
}
```

Property description

In addition to the JSONSchema given in the Appendix , the properties contained in the **tdx Volt** configuration are described below.

Depending on the scenario, some of the properties may not be applicable and can be omitted.

Properties marked **[required]** are applicable in all scenarios.

id [required]

The globally unique identifier of the Volt.

This property must be present in the **tdx Volt** configuration, unless you are connecting to a remote **tdx Volt** via a Relay Volt (i.e. not peer-to-peer). More on that later.

display_name

The human-readable display name of the Volt. This is non-unique, and is only for reference purposes, i.e. it is not required in order to be able to connect to the Volt.

address

The address at which the **tdx Volt** is running. This will be in the form of `host:port`, where `host` can be either an IP address or a DNS resolvable domain name.

ca_pem [required]

The PEM encoded signing certificate used by the Volt. This must be present in the **tdx Volt** configuration and is used to encrypt all communication with the Volt.

challenge_code

If present it represents a challenge code that can be signed and presented when attempting to bind to this Volt. This is required as part of the initial bind flow to demonstrate to the **tdx Volt** that you have some secret, pre-shared information. You do not need to do this if the **tdx Volt** already knows your public key, or you have other verifiable credentials you can present these in the bind request.

public_key

The public key of the Volt. This is useful for identifying the **tdx Volt** but is not required in the **tdx Volt** configuration in order to be able to connect to the **tdx Volt** (it can be inferred from the signing certificate).

fingerprint

This is a hash of the public key in base58 format, useful as a shortcut means of comparing and lookup up keys.

owner_credential

A base64 encoded JWT signed by the **tdx Volt** key and containing a verifiable credential stating the owner of the **tdx Volt** in the form of a DID.

Appendix

Volt configuration definition

The JSONSchema describing the **tdx Volt** configuration is as follows:

```
{  "$schema": "https://json-schema.org/draft/2019-
09/schema",  "$id": "https://tdxvolt.com/schemas/volt-
configuration",  "type": "object",  "title": "Volt
configuration schema",  "required": ["id", "ca_pem"],
"properties": {    "id": {      "type": "string",
"title": "The id of the Volt.",      "examples": ["1947660b-
fbc0-4345-aec7-03b147d4e417"]    },    "display_name": {
"type": "string",      "title": "A human-readable name of
the Volt.",      "examples": ["macBook (intel)"]    },
"address": {      "type": "string",      "title": "The
address that can be used to connect to the Volt within the
local network.",      "examples": ["192.168.1.69:50908"]
},    "public_key": {      "type": "string",      "title":
"The Volt public key, in PEM format. This is optional and
will be inferred from the ca_pem.",      "examples": [
"-----BEGIN PUBLIC KEY-----
\nMIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEA435moWiFRnDK/ul
-----END PUBLIC KEY-----\n"      ]    },    "fingerprint": {
"type": "string",      "title": "The base58 encoded
fingerprint of the Volt public key. This is optional.",
"examples": ["EPhBiNvb5RAvM1FjzrrrKYP7ggMBaJ5wM7KMTJjBfPaM"]
},    "ca_pem": {      "type": "string",      "title": "The
certificate authority used by the Volt, in PEM-encoded
format.",      "examples": [        "-----BEGIN CERTIFICATE--
---
\nMIIDojCCAoqgAwIBAgIEBoqCUTANBgkqhkiG9w0BAQsFADBxMQswCQYDVQC
-----END CERTIFICATE-----\n"      ]    },
"challenge_code": {      "type": "string",      "title":
"The SHA256 digest of the Volt challenge code. Optional.",
"examples": ["w6uP8Tcg6K2QR905Rms8iXTlksL6OD1KOWBxTK7wxPI="]
}  },  "examples": [    {      "id": "1947660b-fbc0-4345-
aec7-03b147d4e417",      "display_name": "macBook (intel)",
"address": "192.168.1.69:50908",      "public_key": "-----
BEGIN PUBLIC KEY-----
\nMIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEA435moWiFRnDK/ul
-----END PUBLIC KEY-----\n",      "fingerprint":
"EPhBiNvb5RAvM1FjzrrrKYP7ggMBaJ5wM7KMTJjBfPaM",
"ca_pem": "-----BEGIN CERTIFICATE-----
\nMIIDojCCAoqgAwIBAgIEBoqCUTANBgkqhkiG9w0BAQsFADBxMQswCQYDVQC
-----END CERTIFICATE-----\n",      "challenge_code":
"w6uP8Tcg6K2QR905Rms8iXTlksL6OD1KOWBxTK7wxPI="    }  ]}
```

Skip to Content

tdx Volt

putting you in charge

☀☾

Toggle sidebar

- **Getting Started**
  - Welcome
  - Quick Start
- **Concepts**
  - Fundamentals
  - Identity
  - DID registry
  - Verifiable credentials
  - Policy
  - Resource
  - File

- Database
- Wire
- Relay
- Key strategy

How to...

- Create a Volt
- Start a Volt
- Connect to a Volt
- Upload files
- Download files
- Publish to wire
- Subscribe to wire
- Create a database
- Execute SQL
- Import data
- Establish a connect stream
- Approve authenticate request

Clients

- Command line
- Native / C++
- Web
- NodeJS

Reference

- Best practice
- Battery
- Configuration
- Connection
- Connect stream
- Logging
- PKCS#11

API

- Discovery API
- File API
- SqliteDatabase API
- SqliteServer API
- SSI API
- Sync API
- Relay API
- Volt API
- Wire API

Utilities

- protoDbSync
- sqliteServer
- wireTransform

FAQ

- Questions

- **Coming soon**
  - Roadmap
- ☀☾

Connection

This document describes the information needed by a client to be able to establish a connection to a Volt, how that information is persisted in a configuration file by the **tdx Volt** client libraries, and how to use a client configuration file to establish a connection to a **tdx Volt** using the **tdx Volt** CLI.

Client connection information

Two pieces of information are required for a client to be able to connect to a **tdx Volt** to access resources or services:

- **The client credentials.** The client credentials identify the client. As far as the target **tdx Volt** is concerned, this is primarily made up of the client's public key. However, in order to be able to prove possession of the key to the target Volt, the private key will be required by the client library to initiate the TLS handshake or sign a JWT. The client credentials are stored in the `credential` property, described below.

- **The target tdx Volt configuration.** The **tdx Volt** configuration identifies the target Volt. It includes a unique identitifier, address and public key. See the Volt configuration reference section for a full description and details of how to obtain a **tdx Volt** configuration. The Volt configuration is stored in the `volt` property, described below.

Note that only the public portion of the client key is sent to the Volt, the private key is never seen by the Volt.

The basic structure of a client configuration is shown below:

```
{  "client_name": "Alice",  "credential": {    "key": "<PEM-
encoded private key>"  },  "volt": {    <paste a Volt
configuration object here>  }}
```

Client configuration file

All the current **tdx Volt** client libraries support using a file to store a client configuration in the JSON format described here.

It is not obligatory for clients or applications to use this format. The configuration details can be specified as a plain object and stored in whatever method suits.

Obtain a client configuration

If you have a connection to the target **tdx Volt** configured in the **fusebox**, you can use this to quickly obtain a client configuration file.

Select the identity you want to use for the client in the Explorer pane on the left side of the **fusebox**. Then use the 'copy to clipboard' button next to the 'client configuration' detail in the right-hand panel, as highlighted in the image below:

Alternatively, you can use the 'copy client configuration' button on the 'metadata' dialog of the identity.

Populating the client key

Note that unless the full client key is stored in the **tdx Volt**, it will be necessary to manually populate the `key` property of the JSON that is copied to the clipboard using the method described above.

The example below shows the configuration copied from an identity that doesn't have the key stored in the **tdx Volt**. As you can see, the `key` property has a placeholder (`<**** INSERT PEM-FORMAT KEY HERE ****>`) that must be replaced with the actual PEM-format private key corresponding to this client.

```
{  "client_name": "Local",  "credential": {    "client_id":
"65b8a083-554e-442e-a62b-c9cefbe208a4",    "key": "<****
INSERT PEM-FORMAT KEY HERE ****>"  },  "volt": {    "id":
"1947660b-fbc0-4345-aec7-03b147d4e417",    "display_name":
"macBook (intel)",    "address": "192.168.1.69:50908",
"public_key": "-----BEGIN PUBLIC KEY-----
\nMIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEA435moWiFRnDK/uL
-----END PUBLIC KEY-----\n",    "fingerprint":
"EPhBiNvb5RAvM1FjzrrrKYP7ggMBaJ5wM7KMTJjBfPaM",    "ca_pem":
"-----BEGIN CERTIFICATE-----
\nMIIDojCCAoqgAwIBAgIEBoqCUTANBgkqhkiG9w0BAQsFADBxMQswCQYDVQQ
-----END CERTIFICATE-----\n",    "challenge_code":
"w6uP8Tcg6K2QR905Rms8iXTlksL6OD1KOWBxTK7wxPI="  }}
```

Create client configuration

If you want to create a new client on a **tdx Volt** you can manually create one using the CLI.

The first step in creating a client configuration file is to obtain the configuration details of the **tdx Volt** you wish to connect to.

Get the Volt configuration details

In general, the details of how to go about this are given in the Volt configuration section.

The client libraries support a couple of other methods that attempt to automatically resolve or discover the Volt configuration.

The first of these methods is the use of Volt DID, or decentralised identifier, which is registered on any Volt cloud portal. A full description of decentralised identifiers is out of scope for this document, but more information can be found here.

The Volt DID can be specified in the client configuration in one of two ways, the first of which is a shortcut using the `volt` property as a string:

```
{  "client_name": "Alice",  "credentials": { ... }  "volt" :
"did:tdx:349970a5-9f3a-4ac6-aef3-75881e7b87e7"}
```

The second is to add a `did` property to the `volt` object:

```
{  "client_name": "Alice",  "credentials": { ... }  "volt" :
{    "did": "did:tdx:349970a5-9f3a-4ac6-aef3-75881e7b87e7"
}}
```

Another method of automatically acquiring the Volt configuration is using a discovery URL.

Similar to the DID examples above, the Volt discovery URL can be specified in the client configuration in one of two ways, the first of which is a shortcut using the `volt` property as a string:

```
{  "client_name": "Alice",  "credentials": { ... }  "volt" :
"https://tdxvolt.com"}
```

Or using a `http_address` property on the `volt` object:

```
{  "client_name": "Alice",  "credentials": { ... }  "volt" :
{    "http_address": "https://tdxvolt.com"  }}
```

Note that the target Volt will need to have its HTTP server enabled for the `http_address` resolution to work.

Creating the configuration file

Once you have obtained the target **tdx Volt** configuration, the next step is to create the client configuration file to store the **tdx Volt** configuration alongside your client credentials.

Use your favourite text editor to create the configuration file, e.g.

```
# If using nano, specify the `-w` switch to prevent wrapping
as this can corrupt PEM encoded data.nano -w my.config.json
```

Terminal window

Create a minimal configuration using the following:

```
{  "client_name": "<enter a friendly name for this client>",
"volt": <paste the Volt configuration obtained above here>}
```

An example of a **tdx Volt** configuration for connection to a local (P2P) **tdx Volt** is shown below.

```
{  "client_name": "connection demo",  "volt": {    "id":
"449a3385-f380-41f7-bd0a-e60caaa403cb",    "address":
"192.168.1.194:58913",    "ca_pem": "-----BEGIN CERTIFICATE-
----
\nMIIDojCCAoqgAwIBAgIEJdz3cjANBgkqhkiG9w0BAQsFADBxMQswCQYDVQQ
-----END CERTIFICATE-----\n",    "challenge_code":
"w6uP8Tcg6K2QR905Rms8iXTlksL6OD1KOWBxTK7wxPI="  }}
```

Note that in the example above the `credential` section has not been specified, and therefore will be auto-generated by the client library and a key will be created and stored in the file. If you already have a key that you wish to use for the connection you should place it in the `key` property of the `credential` section.

You can now use the **tdx Volt** CLI to help complete the configuration file:

```
./volt list . -c my.config.json
```

Terminal window

Chances are that the above command will result in errors along the lines of `failure binding to Volt: policy decision pending`. This means that the owner of the Volt you are trying to connect to needs to approve your request to connect to the Volt. However, if you examine the `my.config.json` file you should see that the `credential` section has been created along with an auto-generated key:

```
cat my.config.json
```

Terminal window

```
{  "client_name": "connection demo",  "credential": {
"cert": "",    "client_id": "",    "key": "-----BEGIN
PRIVATE KEY-----
\nMIIEvgIBADANBgkqhkiG9w0BAQEFAASCBKgwggSkAgEAAoIBAQDwHswi1C8
-----END PRIVATE KEY-----\n"  },  "volt": {    "ca_pem": "--
---BEGIN CERTIFICATE-----
\nMIIDojCCAoqgAwIBAgIEBoqCUTANBgkqhkiG9w0BAQsFADBxMQswCQYDVQQ
-----END CERTIFICATE-----\n",    "challenge_code":
"w6uP8Tcg6K2QR905Rms8iXTlksL6OD1KOWBxTK7wxPI=",    "id":
"1947660b-fbc0-4345-aec7-03b147d4e417",    "address":
"192.168.1.69:50908"  }}
```

In the output above, the `client_id` and `cert` properties of the `credential` object are blank. These will be populated once the Volt owner approves the binding request.

Test the connection

Assuming you have created a client configuration file, you can now use it to connect to a Volt.

For example, an initial client configuration file named `client.config.json` is show below.

This file indicates that the target **tdx Volt** has id `449a3385-f380-41f7-bd0a-e60caaa403cb` and is running locally at the address `192.168.1.194:58913`.

```
{  "client_name": "connection demo",  "volt": {    "id":
"449a3385-f380-41f7-bd0a-e60caaa403cb",    "address":
"192.168.1.194:58913",    "ca_pem": "-----BEGIN CERTIFICATE-
----
\nMIIDojCCAoqgAwIBAgIEJdz3cjANBgkqhkiG9w0BAQsFADBxMQswCQYDVQQ
-----END CERTIFICATE-----\n",    "challenge_code":
"w6uP8Tcg6K2QR905Rms8iXTlksL6OD1KOWBxTK7wxPI="  }}
```

We can use the Volt CLI to issue a request to the Volt. Here we ask the **tdx Volt** to list all the resources in the clients ‘Home’ folder (indicated by ’.’). This may yield no results if you have only just bound to the Volt.

```
./volt list . -c client.config.json
```

Terminal window

Now try uploading a file to the home folder.

```
./volt upload path/to/some/file . -c client.config.json
```

Terminal window

And then list the resources again:

```
./volt list . -c client.config.json
```

Terminal window

The **tdx Volt** CLI will look for a client configuration file named `volt.config.json` if none is specified on the command line. So if you use this name to store your configuration details there is no need to specify the `-c client.config.json` parameter.

Relay connections

The discussion so far has related to peer-to-peer connections. However in many scenarios it will be necessary to connect to a Volt that is not on the same local network as the client, and may not be accessible via the wider internet because it is behind a firewall.

In order to be able to connect to remote Volts in these scenarios you can utilise the concept of a Relay Volt, which is described in more detail here .

The first step is to establish the configuration of the Relay Volt you would like to use.

It is then a case of adding another Volt configuration object describing the Relay Volt as a sub-property of the target `volt` configuration. This `relay` property takes the same format as the standard Volt configuration:

```
{  "client_name": "your friendly name",  "volt": {    "id":
"<volt id>",    "ca_pem": "<volt CA certificate>",
"challenge_code": "<volt challenge code>",    "relay": {
"id": "<Relay volt id>",      "ca_pem": "<Relay volt CA
certificate>",    }  }}
```

You can use the same discovery options as described above for obtaining a Volt configuration, for example, via a DID lookup or a HTTP discovery:

```
{  "client_name": "your friendly name",  "volt": {    ...
"relay": "did:tdx:349970a5-9f3a-4ac6-aef3-75881e7b87e7"  }}
{  "client_name": "your friendly name",  "volt": {    ...
"relay": "https://cloud.tdxvolt.com"  },}
```

The addition of the `relay` property is the only addition that is required to a standard client configuration to force the client libraries to connect via a Relay Volt. However if the configuration has previously been used to bind locally, it may be necessary to delete the `credential.cert` property to force the client library to initialise the Relay parameters correctly.

Appendix

Client configuration definition

The JSONSchema definition of the Volt client configuration object is shown below.

Note that the `client_name` and `crytpo` properties describe the client and its credentials, and the remainder of the document (the `volt` property) is simply an instance of a **tdx Volt** configuration object.

```
{  "$schema": "https://json-schema.org/draft/2019-
09/schema",  "$id": "https://tdxvolt.com/schemas/volt-
client-connection",  "type": "object",  "title": "Volt
client connection schema",  "required": ["client_name",
"credential", "volt"],  "properties": {    "client_name": {
"type": "string",      "title": "A human-readable name of
the client.",      "examples": ["Alice"]    },
"credential": {      "type": "object",      "title": "The
cryptographic credentials identifying the client.",
"required": ["client_id", "key"],      "properties": {
"client_id": {          "type": "string",          "title":
"The UUID assigned to the client by the Volt during the
binding phase.",          "examples": ["c65ab887-35d2-4955-
b777-cbc9fba32dd1"]        },        "key": {
"type": "string",          "title": "The PEM-encoded key of
the client. This can be encrypted.",          "examples": [
"-----BEGIN ENCRYPTED PRIVATE KEY-----
\nMIIFHDBOBgkqhkiG9w0BBQ0wQTApBgkqhkiG9w0BBQwwHAQInAzvld0a3a4
-----END ENCRYPTED PRIVATE KEY-----\n"          ]        }
}    },    "volt": {      "type": "object",      "title":
"The Volt configuration information.",      "properties": {
"$ref": "https://tdxvolt.com/schemas/volt-configuration"
}    }  },  "examples": [    {      "client_name": "CLI",
"credential": {        "client_id": "c65ab887-35d2-4955-
b777-cbc9fba32dd1",        "key": "-----BEGIN ENCRYPTED
PRIVATE KEY-----
\nMIIFHDBOBgkqhkiG9w0BBQ0wQTApBgkqhkiG9w0BBQwwHAQInAzvld0a3a4
-----END ENCRYPTED PRIVATE KEY-----\n"      },      "volt": {
"id": "1947660b-fbc0-4345-aec7-03b147d4e417",
"display_name": "macBook (intel)",        "address":
"192.168.1.69:50908",        "public_key": "-----BEGIN
PUBLIC KEY-----
\nMIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEA435moWiFRnDK/uL
-----END PUBLIC KEY-----\n",        "fingerprint":
"EPhBiNvb5RAvM1FjzrrrKYP7ggMBaJ5wM7KMTJjBfPaM",
"ca_pem": "-----BEGIN CERTIFICATE-----
\nMIIDojCCAoqgAwIBAgIEBoqCUTANBgkqhkiG9w0BAQsFADBxMQswCQYDVQQ
-----END CERTIFICATE-----\n",        "challenge_code":
"w6uP8Tcg6K2QR905Rms8iXTlksL6OD1KOWBxTK7wxPI="      }    }
]}
```

- ## Concepts

  - Fundamentals
  - Identity
  - DID registry
  - Verifiable credentials
  - Policy
  - Resource
  - File
  - Database
  - Wire
  - Relay
  - Key strategy

- ## How to...

  - Create a Volt
  - Start a Volt
  - Connect to a Volt
  - Upload files
  - Download files
  - Publish to wire
  - Subscribe to wire
  - Create a database
  - Execute SQL
  - Import data
  - Establish a connect stream
  - Approve authenticate request

- ## Clients

  - Command line
  - Native / C++
  - Web
  - NodeJS

- ## Reference

  - Best practice
  - Battery
  - Configuration
  - Connection
  - Connect stream
  - Logging
  - PKCS#11

- ## API

  - Discovery API
  - File API
  - SqliteDatabase API
  - SqliteServer API
  - SSI API
  - Sync API
  - Relay API
  - Volt API
  - Wire API

- ## Utilities

  - protoDbSync

- sqliteServer
- wireTransform

- **FAQ**
  - Questions

- **Coming soon**
  - Roadmap
- ☀☾

Best practice

The following sections give some recommendations for best practices when commissioning and using the **tdx Volt**.

tdx Volt Key Strategy

The ‘insecure’ Battery mode along with the Battery key strategy is intended for development and testing environments only.

At a minimum, the Battery should be secured with a password. In this configuration the Battery database is encrypted at rest. As such, any **tdx Volt**s that are configured to use the Battery key strategy at least have their key stored in an encrypted database.

However, it is recommended to use the **tdx Volt** pkcs#11 or local file key storage. This enables the key to be encrypted and stored in a file on disk, including a secure removable storage medium.

An added benefit of the ‘local file’ key strategy is that it makes it much easier to establish a remote connection to your **tdx Volt** via the **fusebox**, which requires the root key to be available in order to be able to configure the connection.

Secure the Volt key

Related to the above, and in line with least privilege practices, it is recommended to **not** use the **tdx Volt** root key for applications or scripts, or for anything other than securing your **tdx Volt**. Instead, create a separate identity for each use case or scenario and only share the data required to complete the task at hand.

For example, when provisioning the `protoDbSync` utility, create a new identity called `protoDbSync client` and copy the configuration into the required configuration file, rather than using the root **tdx Volt** key.

Skip to Content

tdx Volt

putting you in charge
☀☾
Toggle sidebar

- **Getting Started**

- Welcome
- Quick Start

Concepts

- Fundamentals
- Identity
- DID registry
- Verifiable credentials
- Policy
- Resource
- File
- Database
- Wire
- Relay
- Key strategy

How to...

- Create a Volt
- Start a Volt
- Connect to a Volt
- Upload files
- Download files
- Publish to wire
- Subscribe to wire
- Create a database
- Execute SQL
- Import data
- Establish a connect stream
- Approve authenticate request

Clients

- Command line
- Native / C++
- Web
- NodeJS

Reference

- Best practice
- Battery
- Configuration
- Connection
- Connect stream
- Logging
- PKCS#11

API

- Discovery API
- File API
- SqliteDatabase API
- SqliteServer API
- SSI API
- Sync API
- Relay API
- Volt API
- Wire API

- ## Utilities
  - protoDbSync
  - sqliteServer
  - wireTransform
- ## FAQ
  - Questions
- ## Coming soon
  - Roadmap
- ☀☾

Logging

All Volt components expose logging capabilities which can produce extensive diagnostic information for use in troubleshooting and debugging scenarios.

Environment variables

The logging mode and verbosity is controlled via two environment variables.

The `TDXVOLT_LOG_LEVEL` environment variable configures the overall logging mode, and can take either `ERROR` or `DEBUG` values. The default value, if no environment variable is found, is dependent on the build mode - see below.

The `TDXVOLT_LOG_DEBUG` environment variable can be used to control the verbosity of output when using the `DEBUG` log level. This variable can take various values as indicated in the table below:

value	description
all	All components emit output.
client	C++ client library components.
core	The Volt core components.
crypto	All cryptographic-related output.
database	Output related to the Volt meta-database. This can be quite noisy.
fusebox	Fusebox-specific debug output.
grpc-client	Client-side grpc diagnostics.
grpc-server	Server-side grpc diagnostics.
perfomance	Performance and timing data across the platform.
policy	Policy evaluation diagnostics, this can be quite noisy.
qml	Diagostics originating from QML code. This is only used by the fusebox.

The `TDXVOLT_LOG_DEBUG` variable can be used to pick and mix output from various debug components. A component can be omitted from the output by prefixing the name with a minus sign.

```
# Request all DEBUG information apart from policy and
database output.export TDXVOLT_LOG_DEBUG=all,-policy,-
database
```

Terminal window

```
# Request crypto debug output.export
TDXVOLT_LOG_DEBUG=crypto
```

Terminal window

If the `TDXVOLT_LOG_DEBUG` variable is unset and `TDXVOLT_LOG_LEVEL` is set to `DEBUG` then by default **all** component output is generated, which is equivalent to `TDXVOLT_LOG_DEBUG=all`.

Note that the `TDXVOLT_LOG_DEBUG` variable only affects `DEBUG` output. The `ERROR` output will be generated for all components irrespective of the value of `TDXVOLT_LOG_DEBUG`.

Release builds

The default log level for release builds is `ERROR`, and the output is written to a file in the `~/Documents/tdxVolt/logs` folder, or equivalent depending on the OS.

The log output is never written to STDOUT when using release builds.

The log level can be increased to `DEBUG` via the `TDXVOLT_LOG_LEVEL` environment variable, and similarly the `TDXVOLT_LOG_DEBUG` variable can be used to configure the verbosity of the various components, as outlined above.

Debug builds

The default log level for debug builds is `DEBUG`, and the output is written to STDOUT, as well as a file in the `~/Documents/tdxVolt/logs` folder, or equivalent depending on the OS.

The default verbosity level `all`, so all components will output debug information.

GRPC

It is possible to enable grpc logging in addition to that of the Volt. This can be done using the `GRPC_VERBOSITY` and `GRPC_TRACE` environment variables, see this document for more information.

IoTSAF

Docs

Table Of Contents:

- intro
- How to do this
- Result:
- nist-brski
  - Devices
- Mermaid charts
- Project Management
- Project Docs
- Project Plan
- Internal
- Reporting
- Actions
- IoT Security Assurance Framework
  - Notices, Disclaimer, Terms of Use, Copyright and Trademarks and Licensing
  - Notices
  - Terms of Use
  - Disclaimer
  - Copyright, Trademarks and Licensing
- Acknowledgements
  - Acknowledgements
    - Peer Reviewers
    - Editors
- Introduction
  - 1.1 Introduction
- intended-audience
  - 1.2 Intended Audience
- scope
  - 1.3 Scope
    - 1.3.1 Key Issues for IoT Security
- Security Requirements
    - 1.3.2 The Supply Chain of Trust
- iotsf-resources-that-support-the-framework
  - 1.4 IoTSF Resources that support the Framework
    - 1.4.1 Changes from Release 2.1 of the Framework
- the-process
  - 2.1 The Process
    - 2.1.1 Risk Assessment
- assurance-class
  - 2.2 Assurance Class
    - 2.2.1 Determining Security Goals – An Example
- using-the-assurance-questionare
  - 2.3 Using the Assurance Questionnaire
    - 2.3.1 Assessment Methodology
    - 2.3.2 Keywords
    - 2.3.3 Assurance Requirements Completion Responsibilities
    - 2.3.4 Evidence
- assurance-terminology-and-applicability
  - 2.4 Assurance Terminology and Applicability
    - 2.4.1 Terminology
    - 2.4.2 Level of Assurance
- 2.4.8 Authentication And Authorisation
- Requirement 2.4.8.1
- Requirement 2.4.8.10
- Requirement 2.4.8.3
- Requirement 2.4.8.12
- Requirement 2.4.8.8
- Requirement 2.4.8.14
- Requirement 2.4.8.18
- Requirement 2.4.8.13
- Requirement 2.4.8.9
- Requirement 2.4.8.6
- Requirement 2.4.8.16
- Requirement 2.4.8.15
- Requirement 2.4.8.4
- Requirement 2.4.8.5
- Requirement 2.4.8.7
- Requirement 2.4.8.2
- Requirement 2.4.8.11
- Requirement 2.4.8.17
- 2.4.3 Business Process
- Requirement 2.4.3.29
- Requirement 2.4.3.1
- Requirement 2.4.3.19
- Requirement 2.4.3.23
- Requirement 2.4.3.5.1
- Requirement 2.4.3.6
- Requirement 2.4.3.15
- Requirement 2.4.3.8
- Requirement 2.4.3.26
- Requirement 2.4.3.16

- Requirement 2.4.3.5
- Requirement 2.4.3.22
- Requirement 2.4.3.17
- Requirement 2.4.3.14
- Requirement 2.4.3.13
- Requirement 2.4.3.25
- Requirement 2.4.3.9.1
- Requirement 2.4.3.4
- Requirement 2.4.3.24
- Requirement 2.4.3.7
- Requirement 2.4.3.21
- Requirement 2.4.3.18
- Requirement 2.4.3.3
- Requirement 2.4.3.28
- Requirement 2.4.3.22.1
- Requirement 2.4.3.12
- Requirement 2.4.3.10
- Requirement 2.4.3.9
- Requirement 2.4.3.20
- Requirement 2.4.3.27
- Requirement 2.4.3.11
- Requirement 2.4.3.2
- 2.4.13 Cloud and Network Elements
- Requirement 2.4.13.5
- Requirement 2.4.13.17
- Requirement 2.4.13.19
- Requirement 2.4.13.15
- Requirement 2.4.13.20
- Requirement 2.4.13.6
- Requirement 2.4.13.4
- Requirement 2.4.13.26
- Requirement 2.4.13.16
- Requirement 2.4.13.3
- Requirement 2.4.13.29
- Requirement 2.4.13.2
- Requirement 2.4.13.18
- Requirement 2.4.13.33
- Requirement 2.4.13.35
- Requirement 2.4.13.9
- Requirement 2.4.13.1
- Requirement 2.4.13.25
- Requirement 2.4.13.12
- Requirement 2.4.13.30
- Requirement 2.4.13.23
- Requirement 2.4.13.32
- Requirement 2.4.13.7
- Requirement 2.4.13.11
- Requirement 2.4.13.36
- Requirement 2.4.13.14
- Requirement 2.4.13.31
- Requirement 2.4.13.27
- Requirement 2.4.13.28
- Requirement 2.4.13.13
- Requirement 2.4.13.21
- Requirement 2.4.13.8
- Requirement 2.4.13.34
- Requirement 2.4.13.24
- Requirement 2.4.13.10
- Requirement 2.4.13.22
- 2.4.15 Configuration
- Requirement 2.4.15.1
- Requirement 2.4.15.3
- Requirement 2.4.15.2
- 2.4.4 Device Hardware
- Requirement 2.4.4.13
- Requirement 2.4.4.11
- Requirement 2.4.4.2
- Requirement 2.4.4.8
- Requirement 2.4.4.17
- Requirement 2.4.4.6
- Requirement 2.4.4.15
- Requirement 2.4.4.3
- Requirement 2.4.4.9
- Requirement 2.4.4.14
- Requirement 2.4.4.16
- Requirement 2.4.4.7
- Requirement 2.4.4.12
- Requirement 2.4.4.1
- Requirement 2.4.4.5
- Requirement 2.4.4.10
- Requirement 2.4.4.4
- 2.4.7 Device Interfaces
- Requirement 2.4.7.18
- Requirement 2.4.7.5

- Requirement 2.4.7.9
- Requirement 2.4.7.13
- Requirement 2.4.7.2
- Requirement 2.4.7.23
- Requirement 2.4.7.15
- Requirement 2.4.7.12
- Requirement 2.4.7.7
- Requirement 2.4.7.11
- Requirement 2.4.7.1
- Requirement 2.4.7.20
- Requirement 2.4.7.24
- Requirement 2.4.7.16
- Requirement 2.4.7.6
- Requirement 2.4.7.8
- Requirement 2.4.7.21
- Requirement 2.4.7.10
- Requirement 2.4.7.4
- Requirement 2.4.7.17
- Requirement 2.4.7.25
- Requirement 2.4.7.14
- Requirement 2.4.7.3
- Requirement 2.4.7.22
- Requirement 2.4.7.19
- 2.4.6 Device Operating System
- Requirement 2.4.6.1
- Requirement 2.4.6.9
- Requirement 2.4.6.5
- Requirement 2.4.6.14
- Requirement 2.4.6.8
- Requirement 2.4.6.13
- Requirement 2.4.6.12
- Requirement 2.4.6.10
- Requirement 2.4.6.4
- Requirement 2.4.6.6
- Requirement 2.4.6.3
- Requirement 2.4.6.7
- Requirement 2.4.6.11
- Requirement 2.4.6.15
- Requirement 2.4.6.2
- 2.4.16 Device Ownership Transfer
- Requirement 2.4.16.6
- Requirement 2.4.16.7
- Requirement 2.4.16.5
- Requirement 2.4.16.3
- Requirement 2.4.16.4
- Requirement 2.4.16.1
- Requirement 2.4.16.2
- 2.4.5 Device Software
- Requirement 2.4.5.31
- Requirement 2.4.5.11
- Requirement 2.4.5.40
- Requirement 2.4.5.30
- Requirement 2.4.5.25
- Requirement 2.4.5.19
- Requirement 2.4.5.2
- Requirement 2.4.5.5
- Requirement 2.4.5.20
- Requirement 2.4.5.39
- Requirement 2.4.5.36
- Requirement 2.4.5.12
- Requirement 2.4.5.10
- Requirement 2.4.5.18
- Requirement 2.4.5.27
- Requirement 2.4.5.6
- Requirement 2.4.5.38
- Requirement 2.4.5.15
- Requirement 2.4.5.4
- Requirement 2.4.5.34
- Requirement 2.4.5.23
- Requirement 2.4.5.1
- Requirement 2.4.5.29
- Requirement 2.4.5.24
- Requirement 2.4.5.13
- Requirement 2.4.5.3
- Requirement 2.4.5.33
- Requirement 2.4.5.28
- Requirement 2.4.5.37
- Requirement 2.4.5.17
- Requirement 2.4.5.16
- Requirement 2.4.5.14
- Requirement 2.4.5.8
- Requirement 2.4.5.26
- Requirement 2.4.5.21
- Requirement 2.4.5.9

- Requirement 2.4.5.41
- Requirement 2.4.5.32
- Requirement 2.4.5.7
- Requirement 2.4.5.22
- Requirement 2.4.5.35
- 2.4.9 Encryption and Key Management for Hardware
- Requirement 2.4.9.1
- Requirement 2.4.9.6
- Requirement 2.4.9.2
- Requirement 2.4.9.3
- Requirement 2.4.9.10
- Requirement 2.4.9.4
- Requirement 2.4.9.7
- Requirement 2.4.9.9
- Requirement 2.4.9.11
- Requirement 2.4.9.8
- Requirement 2.4.9.5
- 2.4.17 Infrastructure Management
- Requirement
- Requirement 2.4.16.3
- Requirement 2.4.16.10
- Requirement 2.4.16.5
- Requirement 2.4.16.2
- Requirement 2.4.16.9
- Requirement 2.4.16.8
- Requirement 2.4.16.6
- Requirement 2.4.17.1
- Requirement 2.4.16.7
- Requirement 2.4.16.4
- 2.4.11 Mobile Application
- Requirement 2.4.11.3
- Requirement 2.4.11.1
- Requirement 2.4.11.10
- Requirement 2.4.11.11
- Requirement 2.4.11.7
- Requirement 2.4.11.2
- Requirement 2.4.11.9
- Requirement 2.4.11.12
- Requirement 2.4.11.5
- Requirement 2.4.11.13
- Requirement 2.4.11.4
- Requirement 2.4.11.8
- Requirement 2.4.11.6
- 2.4.12 Data Protection and Privacy
- Requirement 2.4.12.4
- Requirement 2.4.12.6
- Requirement 2.4.12.3
- Requirement 2.4.12.10
- Requirement 2.4.12.9
- Requirement 2.4.12.7
- Requirement 2.4.12.13
- Requirement 2.4.12.8
- Requirement 2.4.12.2
- Requirement 2.4.12.14
- Requirement 2.4.12.11
- Requirement 2.4.12.12
- Requirement 2.4.12.1
- Requirement 2.4.12.5
- Requirement 2.4.12.15
- Product Security and Telecommunications Infrastructure
- 2.4.14 Secure Supply Chain and Production
- Requirement 2.4.14.10
- Requirement 2.4.14.9
- Requirement 2.4.14.8
- Requirement 2.4.14.22
- Requirement 2.4.14.15
- Requirement 2.4.14.17
- Requirement 2.4.14.4
- Requirement 2.4.14.2
- Requirement 2.4.14.18
- Requirement 2.4.14.20
- Requirement 2.4.14.24
- Requirement 2.4.14.25
- Requirement 2.4.14.21
- Requirement 2.4.14.5
- Requirement 2.4.14.7
- Requirement 2.4.14.14
- Requirement 2.4.14.13
- Requirement 2.4.14.12
- Requirement 2.4.14.11
- Requirement 2.4.14.19
- Requirement 2.4.14.16
- Requirement 2.4.14.23
- Requirement 2.4.14.1

- Requirement 2.4.14.6
- Requirement 2.4.14.3
- 2.4.10 Web User Interface
- Requirement 2.4.10.6.1
- Requirement 2.4.10.11
- Requirement 2.4.10.13
- Requirement 2.4.10.2
- Requirement 2.4.10.5
- Requirement 2.4.10.16
- Requirement 2.4.10.6
- Requirement 2.4.10.1
- Requirement 2.4.10.7
- Requirement 2.4.10.18
- Requirement 2.4.10.15
- Requirement 2.4.10.3
- Requirement 2.4.10.4
- Requirement 2.4.10.14
- Requirement 2.4.10.10
- Requirement 2.4.10.9
- Requirement 2.4.10.17
- Requirement 2.4.10.19
- Requirement 2.4.10.12
- Requirement 2.4.10.8
- 3.1 References & Standards
- 3.2 Definitions and Abbreviations
    - 3.2.1 Definitions
    - 3.2.2 Acronyms
- Risk-Assessment-Steps
  - 1 Risk Assessment Steps
- Security-Objectives-and-Requirements
  - 2 Security Objectives and Requirements
- Security-Requirements-Design-and-Implementation
  - 3 Security Requirements Design and Implementation
- Appendix B Introduction to Supply Chain Security Requirements
- B1-Motivation
  - B1 Motivation
- B2-Definition-of-Terms
  - B2 Definition of Terms
- B3-Approach
  - B2 Approach

Intro

test edit

How To Do This

```
import ExternalContent from '@site/src/components/externalContent.js';

<ExternalContent link="https://raw.githubusercontent.com/nqminds/nist-brski/main/README.md"/>
```

Result:

Nist-Brski

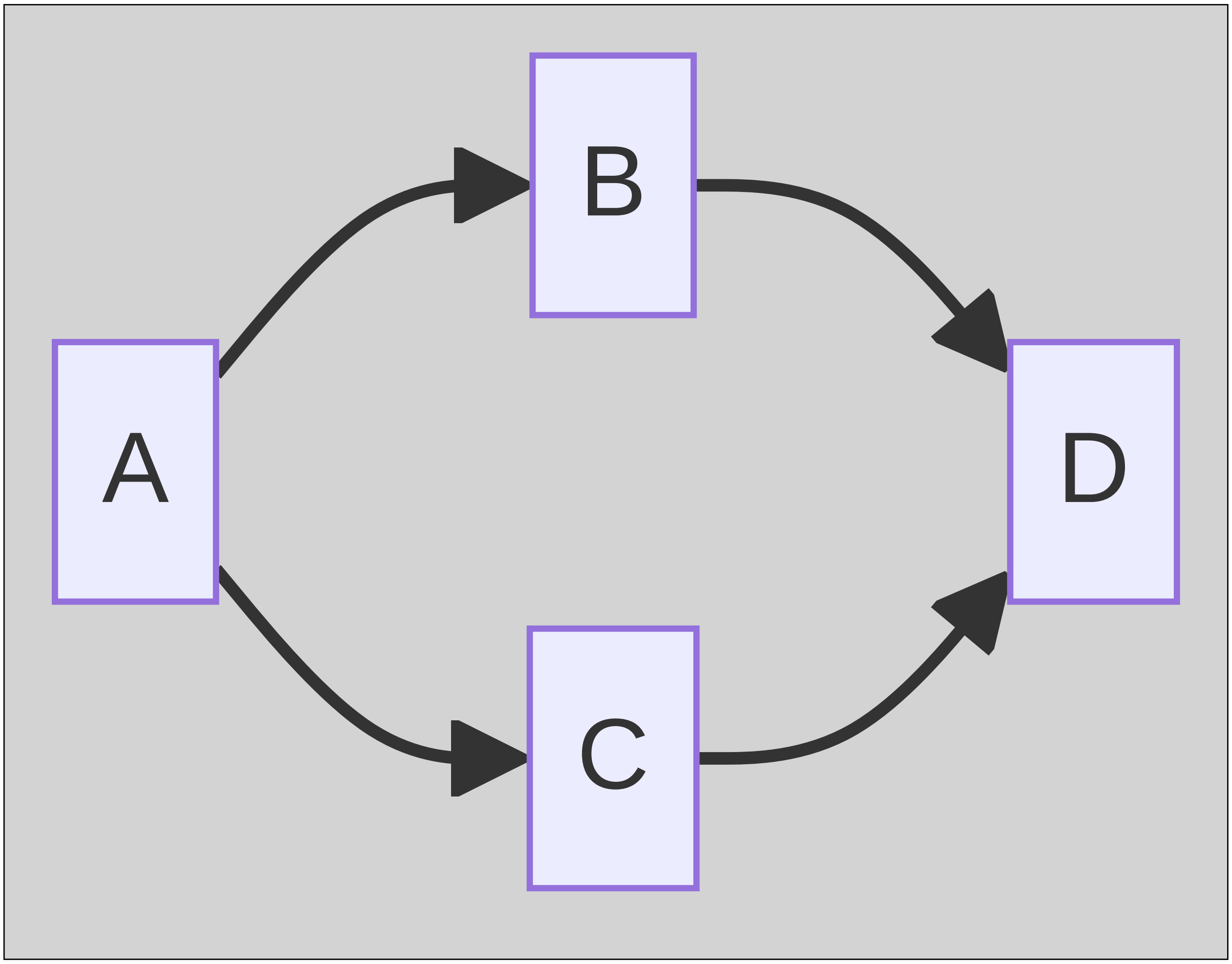
BRSKI demo for NIST

Devices

On the office network, I have set up the following devices:

- `nqm-britannic-brski.local`
  - **Server**, has 2x USB WiFi AP
  - Admins: `alois`, `alexandru`
- `nqm-benign-brski.local`
  - Admins: `alois`, `alexandru`
- `nqm-biddable-brski.local`
  - Admins: `alois`, `alexandru`

Mermaid Charts



Project Management

Project Docs

Project Plan

Internal

Reporting

Actions

☑ Create actions page

IoT Security Assurance Framework

Release 3.0, November 2021

Notices, Disclaimer, Terms Of Use, Copyright And Trademarks And Licensing

Notices

Documents published by the IoT Security Foundation ("IoTSF") are subject to regular review and may be updated or subject to change at any time. The current status of IoTSF publications, including this document, can be seen on the public website at: https://iotsecurityfoundation.org.

Terms Of Use

The role of IoTSF in providing this document is to promote contemporary best practices in IoT security for the benefit of society. In providing this document, IoTSF does not certify, endorse or affirm any third parties based upon using content provided by those third parties and does not verify any declarations made by users. In making this document available, no provision of service is constituted or rendered by IoTSF to any recipient or user of this document or to any third party.

Disclaimer

IoT security (like any aspect of information security) is not absolute and can never be guaranteed. New vulnerabilities are constantly being discovered, which means there is a need to monitor, maintain and review both policy and practice as they relate to specific use cases and operating environments on a regular basis. IoTSF is a non-profit organisation which publishes IoT security best practice guidance materials. Materials published by IoTSF include contributions from security practitioners, researchers, industrially experienced staff and other relevant sources from IoTSF membership and partners. IoTSF has a multi-stage process designed to develop contemporary best practice with a quality assurance peer review prior to publication. While IoTSF provides information in good faith and makes every effort to supply correct, current and high-quality guidance, IoTSF provides all materials (including this document) solely on an 'as is' basis without any express or implied warranties, undertakings or guarantees. The contents of this document are provided for general information only and do not purport to be comprehensive. No representation, warranty, assurance or undertaking (whether express or implied) is or will be made, and no responsibility or liability to a recipient or user of this document or to any third party is or will be accepted by IoTSF or any of its members (or any of their respective officers, employees or agents), in connection with this document or any use of it, including in relation to the adequacy, accuracy, completeness or timeliness of this document or its contents. Any such responsibility or liability is expressly disclaimed. Nothing in this document excludes any liability for: (i) death or personal injury caused by negligence; or (ii) fraud or fraudulent misrepresentation. By accepting or using this document, the recipient or user agrees to be bound by this disclaimer. This disclaimer is governed by English law.

Copyright, Trademarks And Licensing

Acknowledgements

Acknowledgements

We wish to acknowledge significant contributions from IoTSF members to this document:

- Abhay Soorya, Gemserv Ltd
- Alex Margulis, Intel Corp
- Andrew Bott, Secure Thingz Ltd
- Arun Sambordaran, Gemserv Ltd
- Chris Hills, Phaedrus Systems Ltd
- Chris Shire, Infineon Technologies Ltd
- Graham Markall, Embecosm Ltd
- Ian Phillips, Roke Manor Research Ltd
- Ian Poyner, IoTSF
- Isaac Dangana, Red Alert Labs Ltd
- Jan Krueger, Intel Corp
- Jeremy Bennett, Embecosm Ltd
- John Moor, IoT Security Foundation
- Lokesh Johri, Tantiv 4
- Mark Beaumont, Roke Manor Research Ltd
- Michael Richardson, Sandelman Software Works
- Nick Hayes, Thinkstream Ltd
- Pamela Gupta, Outsecure Inc
- Peter Burgers, DisplayLink Ltd
- Richard Marshall, Xitex Ltd
- Richard Storer, MathEmbedded Ltd
- Robert Dobson, Device Authority Ltd
- Roger Shepherd, Chipless Ltd
- Sean Gulliford, Gemserv Ltd
- Trevor Hall, Synaptics / DisplayLink Ltd

Peer Reviewers

- Andrew Bott, Secure Thingz Ltd
- Jeff Day, BT Plc
- James Willison, Unified Security Ltd
- Plus others – you know who you are!

Editors

- Trevor Hall, Synaptics Chair Assurance Framework WG
- Ian Poyner, IoTSF
- Amyas Phillips, Ambotec Ltd - Chair Supply Chain WG
- Richard Marshall, Xitex Ltd
- Graham Markall

Introduction

1.1 Introduction

The IoT Security Foundation (IoTSF) was established to address the challenges of IoT security in an increasingly connected world. It has a specific mission ***"to help secure the Internet of Things, in order to aid its adoption and maximise its benefits. To do this IoTSF will promote knowledge and clear best practice in appropriate security to those who specify, make and use IoT products and systems"***.

In more concise terms for vendors, operators, and end-users: ***"Build Secure, Buy Secure, Be Secure"***.

This IoT Security Assurance Framework ('Framework') leads its user through a structured process of questioning and evidence gathering. This ensures suitable security mechanisms and practices are implemented. It was previously published as the IoT Security Compliance Framework up until Release 2.1, and this version remains fully backward compatible with the same sections and requirement numbering. The terminology better reflects the risk-based system and is better aligned with how governments and international bodies are approaching IoT security.

The Framework is intended to help all companies make high-quality, informed security choices by guiding them through a comprehensive requirement checklist and evidence gathering process. The evidence gathered during the process can be used to declare conformance with best practice to customers and other stakeholders.

Providing good security capability requires decisions upfront in design and use – often referred to as ***secure by design***. In most cases, addressing the security of a product at the design stage is proven to be lower cost, and requiring less effort than trying to "put security" into or around a product after it has been created (which may not even be possible). Decisions need to be made to address use-case, business model, liability level and risk management in addition to technical concerns such as architecture, design features, implementation, testing, configuration and maintenance.

Throughout this document, and others published by the IoTSF, reference is made to "best practice" or "best practice security engineering". These best practices are derived from the combined expertise of the IoTSF members, used and tested within their own companies, and from the publications and guidance of other relevant organisations. Wherever possible, reference is made to existing standards and best practice materials to avoid unnecessary duplication. A list of external reference materials and related bodies is included at the end of this document in the section References and Abbreviations.

Intended-Audience

1.2 Intended Audience

The Framework can be used internally in an organisation as a pre-compliance tool to self-assess or self-certify against, or by a third-party auditor. It can also be used ‘in part’, as a procurement mechanism to help specify security requirements of a supplier contract. The Framework is aimed at the following stakeholders:

- For **Managers** in organisations that provide IoT products, technology and or services. It gives a comprehensive overview of the management process needed to adopt best practice. It will be useful for executive, programme, and project managers, by enabling them to ask the right questions and assess the answers.
- For **Developers and Engineers, Logistics and Manufacturing Staff**, it provides detailed requirements to use in their daily work and in project reviews to validate the use of best practice by different functions (e.g. hardware and software development, logistics etc.). Documentary evidence may be assembled using this Framework as a guide or by completing the Assurance Questionnaire (see below 1.4 IoTSF Resources That Support The Framework). In this way, documentary evidence will be compiled to demonstrate assurance both at development gates, and with third parties such as auditors or customers.
- For **Supply Chain Managers**, the structure can be used to guide the auditing of security practices. It may therefore be applied within a producer organisation (as described above); and inspected by a customer of the producer.
- For **Trusted Third Parties** as part of an audit or certification process.

Scope

1.3 Scope

The scope of this document includes (but is not limited to):

- Business processes
- The “Things” in IoT, i.e. network connected products and/or devices
- Aggregation points such as gateways and hubs that form part of the connectivity
- Networking including wired, and radio connections, cloud and server elements

1.3.1 Key Issues For IoT Security

The key compliance requirements can be summarised as follows:

Security Requirements

The following table outlines key security requirements and associated actions:

Key Requirement	Action Required	Framework Reference
Management governance	There must be a named executive responsible for product security, and privacy of customer information.	2.4.3, 2.4.11
Engineered for security	The hardware and software must be designed with attention to security threats.	2.4.4, 2.4.5, 2.4.6, 2.4.7
Fit for purpose cryptography	These functions should be from the best practice industry standards.	2.4.8, 2.4.9
Secure network framework and applications	Precautions have been taken to secure Apps, web interfaces, and server software.	2.4.12, 2.4.13
Secure production processes and supply chain	Making sure the security of the product is not compromised in the manufacturing process or in the end customer delivery and installation.	2.4.10, 2.4.12, 2.4.13
Safe and secure for the customer	The product is safe and secure "out of the box" and in its day-to-day use. The configuration and control should guide the person managing the device into maintaining security and provide for software updates, vulnerability disclosure policy, and life cycle management.	2.4.14

1.3.2 The Supply Chain Of Trust

All end-use products are constructed using a set of component parts, typically sourced from a variety of suppliers. These parts may be electronic or mechanical components, software modules or packages, including open source. Many of these parts will be procured from third party suppliers. It is important that all parts, together with the supply chain logistics, be subject to a security review/audit.

The final IoT product can then be provided with its own evidence of security assessment, together with the component parts documents, as a complete package of auditable evidence. This will help users to assess how the product conforms to the overall ***"supply chain of trust"*** [ref 36].

Iotsf-Resources-That-Support-The-Framework

1.4 IoTSF Resources That Support The Framework

The IoTSF provides a number of resources to foster security best practice:

- **This Framework** document [ref 19] is a structured list of security requirements intended to aid the evidence gathering process to guide an organisation through assurance.
- The **Assurance Questionnaire** is a companion audit and assessment tool to the Framework to aid the setting of security objectives and thereafter the collection of documentation and evidence. The Assurance Questionnaire is available to IoTSF members only for free.
- Additional **Best Practice Guidelines** are provided by the Foundation to help understanding of the most important topics [ref 45].
- Further resources including guides, documents, articles and blogs can be found on the IoTSF website.

All IoTSF publications are maintained and reviewed on a regular basis to keep them current – which is a crucial attribute, given the dynamic nature of cyber security.

This is the latest public release and user feedback is welcome as part of its maintenance and evolution for addressing new security threats. You can send feedback and suggestions to improve the Framework by emailing contact@iotsecurityfoundation.org with a subject line of **“Assurance Framework Feedback”**.

1.4.1 Changes From Release 2.1 Of The Framework

Release 2.1 of the Framework was restricted to consumer class products. This Release 3.0 of the Framework includes expanded mapping to standards that have emerged since release 2.1 was published and introduced additional sub sections. New items for this release:

- Change of name from “Compliance Framework” to “Assurance Framework”
- Updated requirements mapping to ETSI standard EN 303 645
- Added new requirements mapping for NIST standard 8259A
- Expanded the Supply Chain section’s requirements

The Assurance Applicability (requirements) elements detailed in section 2.4 and the numbering have been maintained where possible from prior releases of the Framework to maintain consistency.

The-Process

2.1 The Process

The Framework sets out a comprehensive set of security requirements for aspects of the organisation and product. A response to each requirement needs to be recorded, with supporting statements or evidence. The Assurance Questionnaire is available to IoTSF Members to facilitate evidence collation. For requirements deemed “not applicable”, an explanation must be provided as to why. Any alternative countermeasures to reduce any security risk should also be listed.

The assurance process breaks down into a number of steps:

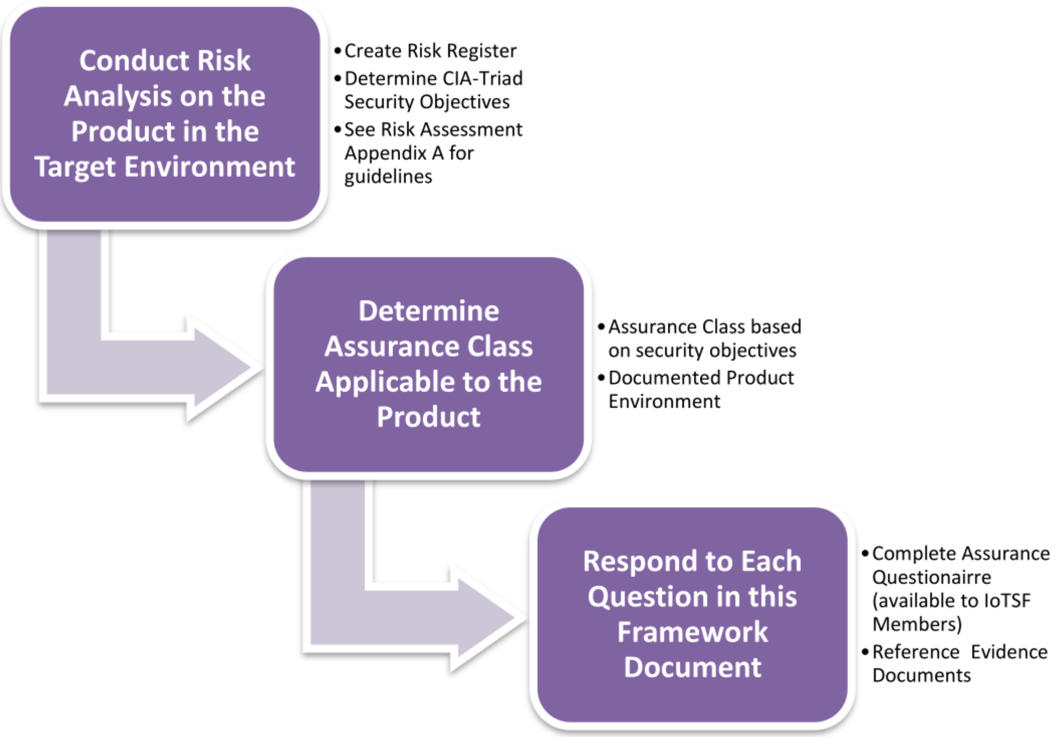

Conduct Risk Analysis on the Product in the Target Environment
•Create Risk Register
•Determine CIA-Triad Security Objectives
•See Risk Assessment Appendix A for guidelines
Determine Assurance Class Applicable to the Product
•Assurance Class based on security objectives
•Documented Product Environment
Respond to Each Question in this Framework Document
•Complete Assurance Questionairre (available to IoTSF Members)
•Reference Evidence Documents

Figure 1 Assurance process steps

2.1.1 Risk Assessment

In security terms, ***context is everything*** - each application differs in use-case and operating environment. It is the responsibility of the Framework user to determine their risk appetite within their stated usage environment and therefore the specific assurance class (section 2.2) of the security measures applied.

To achieve this, **a comprehensive risk assessment is a pre-requisite to using the Framework**. The risk assessment process will help determine the assurance class for the product/service. Section 2.2 has more details on assurance classes and how they relate to the Confidentiality, Integrity and Availability, otherwise known as the CIA Triad [ref 46] model, commonly used by security professionals. Generally, the highest possible assurance class should be adopted, considering not just the immediate context of the product, but also the potential hazards to the system(s) that the product/service may eventually be used in.

A basic outline of the risk assessment process can be found in Appendix A. Risk management techniques can also be found in publications from organisations such as NCSC, ENISA and NIST [ref 40, 41 and 42].

Assurance-Class

2.2 Assurance Class

Determining the security objectives across the full diversity of IoT-class applications is a subjective endeavour. Even within vertical sectors such as consumer and enterprise, the security measures and strength of controls will vary depending on the actual use case. In making the Framework more practical across a range of applications, this version has adopted a risk-based approach derived from the commonly used CIA Triad [ref 46]. Whilst it is not a perfect model, its simplicity is its strength, and good security practice can be derived from the core principles.

Depending on the market and application into which the product is intended to be used, a risk assessment may require a higher assurance class to mitigate the determined level of risk. Consider the following example: a fictional case of a Wi-Fi relay box used in a remote monitoring station, where the threat to the enterprise operation is considered low, could be assessed under Assurance Class 1 requirements. However, when deployed into a hospital, with higher threat dependencies, it could be assessed to be under Assurance Class 4 requirements. A further example is provided in section 2.2.1.

In order to apply an appropriate level of security assurance to a product, the requirements in the Framework are classified using the following assurance classes:

- *Class 0: where compromise to the data generated or loss of control is likely to result in little discernible impact on an individual or organisation*
- *Class 1: where compromise to the data generated or loss of control is likely to result in no more than limited impact on an individual or organisation (requirements in ETSI. DCMS, NCSC CoP demand Class 1 at a minimum)*
- *Class 2: in addition to class 1, the device is designed to resist attacks on availability that would have significant impact on an individual or organisation or impact many individuals. For example, by limiting operations of an infrastructure to which it is connected*
- *Class 3: in addition to class 2, the device is designed to protect sensitive data including Personally identifiable information (PII)*
- *Class 4: in addition to class 3, where compromise to the data generated or loss of control have the potential to affect critical infrastructure or cause personal injury*

For each assurance class, indicative levels of confidentiality, integrity and availability are shown in **Table 1** below.

Security Objective

Assurance Class	Confidentiality	Integrity	Availability
Class 0	Basic	Basic	Basic
Class 1	Basic	Medium	Medium
Class 2	Medium	Medium	High
Class 3	High	Medium	High
Class 4	High	High	High

Table 1: Assurance Class Security Objectives

The definitions of the levels of confidentiality, integrity, and availability are as follows:

- Confidentiality
  - Basic – devices or services processing public information
  - Medium – devices or services processing sensitive information, including Personally Identifiable Information, whose compromise would have limited impact on an individual or organisation
  - High – devices or services processing very sensitive information, including sensitive personal data whose compromise would have significant impact on an individual or organisation
- Integrity

  - Basic – devices or services whose compromise could have a minor or negligible impact on an individual or organisation
  - Medium – devices or services whose compromise could have limited impact on an individual or organisation
  - High – devices or services whose compromise could have a significant or catastrophic impact on an individual or organisation
- Availability
  - Basic – devices or services whose lack of availability would cause minor disruption
  - Medium – devices or services whose lack of availability would have limited impact on an individual or organisation
  - High – devices or services whose lack of availability would have significant impact to an individual or organisation, or impacts many individuals

  [ref 11, 12, 13 & 14 were used as the basis of the above definitions]

Please Note: The Framework Assurance Class is provided for guidance only. A supplier may know of application specific concerns that would change the class values. Requirements deemed "not applicable" must be supported by credible evidence to explain the case.

2.2.1 Determining Security Goals – An Example

To illustrate via a practical example, consider the security features required by a connected thermostat used in a commercial greenhouse. The Assurance Class selection for the device might be determined in the following way:

- Confidentiality is Basic: the underlying assumption is that the thermostat does not store sensitive, confidential, or personally identifiable information
- Integrity is Medium: for a thermostat in a commercial greenhouse, poor data integrity could have a business/financial impact
- Availability is Medium: the thermostat in a commercial greenhouse setting is likely to be part of an environmental control system. As such an individual sensor failure will have little impact, yet a denial- of-service attack across multiple sensors carries a greater commercial risk

In this case, the thermostat may be classified in the following way:

Security Objective

Assurance Class	Confidentiality	Integrity	Availability
Class 1	Basic	Medium	Medium

Table 2: Example of Assurance Class Security Objectives

Using-The-Assurance-Questionare

2.3 Using The Assurance Questionnaire

It is anticipated that assurance with the Framework will become an integral part of an organisation's security process and will provide the supporting evidence for business assurance. An accompanying audit and assessment tool (available to IoTSF Members), the Assurance Questionnaire, may be used at various stages in the product lifecycle. Firstly, by identifying the need for security at the concept stage; secondly listing evidence gathered; to finally signing off security requirements for production release.

The evidence gathering process can only commence after establishing the Assurance Class described in section 2.2. This is done using a risk assessment (see Appendix A).

Once the Assurance Class is determined, the applicable requirements are automatically derived by the accompanying Assurance Questionnaire tool as either mandatory (M) or advisory (A). The Assurance Questionnaire could also be used to optimise the product design and establish if a change would allow a lower Assurance Class to be selected. For example, by not collecting or processing sensitive personal data or perhaps providing automatic failover to alternative services for customers to maintain service availability.

2.3.1 Assessment Methodology

The assessment method is determined by the context i.e. Business (process) or System (technical) and the Class.This determines both the type of assessment e.g. physical testing or document review, along with the degree of rigour from Self-Assessment for lower Classes to full third-party audit for high classes.

2.3.2 Keywords

To improve the usability of this document the requirements in sections 2.4.3 to 2.4.16 have been categorised using the keywords defined in the **Table 3** below.

Primary Keyword	Description	Secondary keyword	Description
System	The requirement is applicable to the technical elements of the device/ product or service	Software	The requirement is directly applicable to the software of the device or service
		Hardware	The requirement is directly applicable to the electronics of the device/service hardware (PCB, processor, components etc.)
		Physical	The requirement is directly applicable to mechanical aspects of the device such as the casing, form factor etc.
Business	A business requirement not directly related to the operational function of the device/ product or service	Process	A flow of activities that indirectly contributes to the security characteristics of a device or service
		Policy	The instructions and guidelines that indirectly contribute to the security characteristics of a device or service
		Responsibility	A role or responsibility that indirectly contributes to the security characteristics of a device or service

Table 3: Keyword Categories

Please Note: the terms Device and Product are interchangeable in this document

2.3.3 Assurance Requirements Completion Responsibilities

The Assurance requirements completion will be addressed by a variety of roles in an organisation. These roles cannot be prescribed exactly as every organisation is different, but each section of requirements may require the attention of Managers and other specialist staff as suggested in **Table 4** below. Responsibility for any individual requirement may be determined by use of the associated keywords, which can be selected by filter, for users of the Assurance Questionnaire.

Section	Topic	Topic Audience & Typical Responsibilities
2.4.3	Business Security Processes, Policies and Responsibilities	Management responsible for governance of a business developing and deploying IoT Devices.
2.4.4	Device Hardware & Physical Security	Design and Production staff responsible for hardware and mechanical quality.
2.4.5	Device Software	Device application quality management by Software Architects, Product Owners and Release Managers.
2.4.6	Device Operating System	Management and Design staff responsible for selection of a third- party operating system or assessing the quality of 'in-house' developed software.
2.4.7	Device Wired and Wireless Interfaces	Design and Production staff responsible for device communications security.
2.4.8	Authentication and Authorisation	Design and Production staff responsible for security of the IoT systems interfaces and foundations of authentication.
2.4.9	Encryption and Key Management for Hardware	Design and Production staff responsible for security of the IoT systems hardware key management and encryption.
2.4.10	Web User Interface	Design and Production staff responsible for security of the IoT Product or Services' Web Systems.
2.4.11	Mobile Application	Design and Production staff responsible for security of the IoT Product or Services' Mobile Application.
2.4.12	Privacy	Management and staff responsible for Data Protection and Privacy regulatory compliance.
2.4.13	Cloud and Network Elements	Design and Production staff responsible for security of the IoT Product or Services' Cloud or Network Systems.
2.4.14	Secure Supply Chain and Production	Management, Design and Production staff responsible for security of the IoT Product or Services' Supply Chain.
2.4.15	Configuration	Design and Production staff responsible for security of the device and IoT Services configurations.
2.4.16	Device Ownership Transfer	Management, Design and Production staff responsible for a products and services' Supply Chain.

Table 4: Assurance Responsibilities

Relevant requirements should be shown as "addressed" and a reference made to the applicable evidence for the product design.

The accompanying Assurance Questionnaire allows for entries, against each relevant requirement, of either the evidence gathered to prove assurance or a link to that evidence. The evidence may be compiled from a number of sources and people. Evidence should be verified by the person responsible for completion of the Framework and such verification should be recorded.

An example of completed Assurance Questionnaire fragment on Business Processes for a high-risk Class 3 device is shown Figure 1 below.

ReqNo	Requirement	Required Assessment Method	Evidence Type	Pre-Assurance	Evidence	Responsability
2.4.3.1	There is a person or role, typically a board level executive, who takes ownership of and is responsible for product, service and business level security and makes and monitors the security policy	SA Document review + TP Inquiry	Organisation al Chart and Job role description/documentation and Proof of Competence (certification/attestation)		URL or reference to document with Third party attestation	CIO name
2.4.3.2	There is a person or role, who takes ownership for adherence to this compliance framework process.	SA Document review + TP Inquiry	Organisation al Chart and Job role description/documentation and Proof of Competence (certification/attestation)		URL or reference to document with Third party attestation	CIO name
2.4.3.4	The company follows industry standard cyber security recommenda tions (e.g. UK Cyber Essentials, NIST Cyber Security Framework, ISO27000 etc.).	SA Document review + TP Inquiry	Organisation al Chart and Job role description/documentation and Proof of Competence (certification/attestation)		URL or reference to document with Third party attestation	CIO name

Figure 2: Assurance Questionnaire Partially Completed Example

2.3.4 Evidence

This Framework offers a comprehensive set of security requirements (see section 2.4 under Assurance Applicability) and should be used with the products or services design documentation including the Risk Register. Evidence of the mitigations made to address each risk line item must also be recorded. Users of the Framework should therefore create their own records and IoTSF members are encouraged to use the Assurance Questionnaire for the recording process.

Such records should be kept safe and secure, we recommend having back-up copies. They could be useful in the case of real-world threats to the product, but also as evidence for any business assurance regimes used in the organisation. The record keeper should enable access, for auditing, to any referenced evidence and supporting documents. URLs especially should be checked to ensure they will remain accessible at least for the life of the product plus any warranty period. Attention should also be paid to maintaining any tools or applications needed to view the evidence material.

An organisation procuring products, systems and services from a supplier, which declares it has used the Framework, may request an audit of the evidence assembled, using either internal resources or a Trusted Third Party ("T3P"). A T3P might be used in situations where the documented evidence would expose sensitive information such as intellectual property or commercial aspects.

Assurance-Terminology-And-Applicability

2.4 Assurance Terminology And Applicability

2.4.1 Terminology

The following terms "must", "must not", "required", "shall", "shall not", "should", "should not", "recommended", "may" and "optional" are used in accordance with the definitions in RFC2119 [ref 25].

2.4.2 Level Of Assurance

The applicability levels are defined as follows

Mandatory	This requirement shall be met, as it is vital to meet the security objectives of the product.
Advisory	This requirement should be met unless there are sound product reasons (e.g. economic viability, hardware complexity). The reasons for deviating from the requirement and alternative countermeasures to reduce any security risk should be documented.

For example in the following tables, where it shows “M of 2 and above” assurance class, this means that the requirement is mandatory for the stated level and all higher levels i.e. 2, 3 & 4.

2.4.8 Authentication And Authorisation

This section's intended audience is for those personnel who are responsible for the security of the IoT systems interfaces and authentication processes. Guidance is available from the IoTSF Best Practice Guides [ref 44] regarding Credential Management (part F).

Requirement 2.4.8.1

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword	SecondaryKeyword
2.4.8.1	The product contains a unique and tamper-resistant device identifier.			
E.g.: the chip serial number or other unique silicon identifier, for example to bind code and data to a specific device hardware. This is to mitigate threats from cloning and also to ensure authentication may be done assuredly using the device identifier e.g. using a device certificate containing the device identifier.	Mandatory for all classes	System	Hardware	

Requirement 2.4.8.10

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword	SecondaryKeyword
2.4.8.10	The access control privileges are defined, justified and documented.	Mandatory for Class 1 and above	Business	Process

Requirement 2.4.8.3

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword	SecondaryKeyword
2.4.8.3	Where a user interface password is used for login authentication, the factory issued or reset password is randomly unique for every device in the product family. If a password-less authentication is used the same principles of uniqueness apply.	Mandatory for all classes	System	Software

Requirement 2.4.8.12

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword	SecondaryKeyword
2.4.8.12	The product allows the factory issued or OEM login accounts to be disabled or erased or renamed when installed or commissioned.	Advisory for all classes	System	Software

Requirement 2.4.8.8

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword	SecondaryKeyword
2.4.8.8	The product securely stores any passwords using an industry standard cryptographic algorithm, compliant with an industry standard.	Mandatory for Class 1 and above	System	Software

Requirement 2.4.8.14

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword	SecondaryKeyword
2.4.8.14	If the product has a password recovery or reset mechanism, an assessment has been made to confirm that this mechanism cannot readily be abused by an unauthorised party.	Mandatory for Class 1 and above	Business	Process

Requirement 2.4.8.18

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword	SecondaryKeyword
2.4.8.18	Devices are provided with a RoT-backed unique authenticable logical identity.	Mandatory for Class 1 and above	System	Software

Requirement 2.4.8.13

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword	SecondaryKeyword
2.4.8.13	The product supports having any or all of the factory default user login passwords altered when installed or commissioned.	Mandatory for all classes	Business	Process

Requirement 2.4.8.9

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword	SecondaryKeyword
2.4.8.9	The product supports access control measures to the root/highest privilege account to restrict access to sensitive information or system processes.	Mandatory for Class 1 and above	System	Software

Requirement 2.4.8.6

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword	SecondaryKeyword
2.4.8.6	Password entry follows industry standard practice on password length, characters from the groupings and special characters.	Mandatory for all classes	System	Software

Requirement 2.4.8.16

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword	SecondaryKeyword
2.4.8.16	The product allows an authorised and complete factory reset of all of the device’s authorisation information.	Advisory for all classes	System	Software

Requirement 2.4.8.15

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword	SecondaryKeyword
2.4.8.15	Where passwords are entered on a user interface, the actual pass phrase is obscured by default.	Mandatory for Class 1 and above	System	Software

Requirement 2.4.8.4

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword	SecondaryKeyword
2.4.8.4	The product does not accept the use of null or blank passwords.	Mandatory for all classes	System	Software

Requirement 2.4.8.5

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword	SecondaryKeyword
2.4.8.5	The product will not allow new passwords containing the user account name with which the user account is associated.	Mandatory for all classes	System	Software

Requirement 2.4.8.7

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword	SecondaryKeyword
2.4.8.7	The product has defence against brute force repeated login attempts, such as exponentially increasing retry attempt delays.	Mandatory for Class 1 and above	System	Software

Requirement 2.4.8.2

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword	SecondaryKeyword
2.4.8.2	Where the product has a secure source of time there is a method of validating its integrity.	Mandatory for Class 1 and above	System	Software

Requirement 2.4.8.11

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword	SecondaryKeyword
2.4.8.11	The product only allows controlled user account access; access using anonymous or guest user accounts is not supported without justification.	Mandatory for Class 1 and above	System	Software

Requirement 2.4.8.17

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword	SecondaryKeyword
2.4.8.17	Where the product has the ability to remotely recover from attack, it should rely on a known good state, to enable safe recovery and updating of the device, but should limit access to sensitive assets until the devices is in a known secure condition.	Mandatory for Class 1 and above	System	Software

2.4.3 Business Process

This section's intended audience is those personnel who are responsible for governance of a business developing and deploying IoT Devices. There must be named executive(s) responsible for product security, and privacy of customer information. There are several classes of requirements, which have been identified by a keyword. Each class should be allocated to a specified person or persons for the product being assessed. Further guidance is available from the IoTSF Best Practice Guidelines [ref 44]. The applicability of each requirement is defined as Advisory or Mandatory for the assessed risk level of any device, the default is Advisory.

Requirement 2.4.3.29

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword	SecondaryKeyword
2.4.3.29				
The organisation retains an enduring competency to revisit and act upon such information during product upgrades or in the event of a potential vulnerability being identified.				
(Key security design information and risk analysis is retained over the whole lifecycle of the product or service.)	Mandatory for all classes	business	process	

Requirement 2.4.3.1

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword	SecondaryKeyword
2.4.3.1	There is a person or role, accountable to the Board, who takes ownership of and is responsible for product, service and business level security, and mandates and monitors the security policy.	Mandatory for all classes	business	responsibility

Requirement 2.4.3.19

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword	SecondaryKeyword
2.4.3.19	Whilst overall accountability for the product or service remains with the person in 2.4.3.1, responsibility can be delegated for each domain involved in any system or device update process, e.g. new binary code to add features or correct vulnerabilities.	Mandatory for Class 2 and above	business	responsibility

Requirement 2.4.3.23

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword	SecondaryKeyword
2.4.3.23	The security update policy for devices with a constrained power source shall be assessed to balance the needs of maintaining the integrity and availability of the device.	Mandatory for Class 2 and above	business	policy

Requirement 2.4.3.5.1

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword	SecondaryKeyword
2.4.3.5.1	The third party policy shall be publicly available and include contact information for reporting issues and information on timelines to acknowledge and provide status updates.	Mandatory for all classes	business	policy

Requirement 2.4.3.6

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword	SecondaryKeyword
2.4.3.6	A policy has been established for addressing risks that could impact security and affect or involve technology or components incorporated into the product or service provided.			
At a minimum this should include a threat model, risk analysis and security requirements for the product and its supply chain through its whole stated supported life. This should be maintained, communicated, prioritised and addressed internally as part of product development throughout the product support period.	Mandatory for Class 2 and above	business	policy	

Requirement 2.4.3.15

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword	SecondaryKeyword
2.4.3.15	Intentionally left blank to maintain requirement numbering		business	

Requirement 2.4.3.8

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword	SecondaryKeyword
2.4.3.8	A process is in place for consistent briefing of senior executives in the event of the identification of a vulnerability or a security breach, especially those executives who may deal with the media or make public announcements.	Mandatory for all classes	business	process

Requirement 2.4.3.26

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword	SecondaryKeyword
2.4.3.26	As part of the security policy, define a process for maintaining a central inventory of third party components and services, and their suppliers, for each product.	Mandatory for all classes	business	policy

Requirement 2.4.3.16

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword	SecondaryKeyword
2.4.3.16	As part of the Security Policy, develop security advisory notification steps.	Mandatory for all classes	business	process

Requirement 2.4.3.5

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword	SecondaryKeyword
2.4.3.5	A policy has been established for interacting with both internal and third party security researcher(s) on the products or services.	Mandatory for all classes	business	policy

Requirement 2.4.3.22

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword	SecondaryKeyword
2.4.3.22	Where remote update is supported, there is an established process/plan for validating "updates" and updating devices on an on-going or remedial basis.	Mandatory for Class 2 and above	business	process

Requirement 2.4.3.17

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword	SecondaryKeyword
2.4.3.17	The Security Policy shall be compliant with ISO 30111 or similar standard.	Mandatory for Class 3 and above	business	policy

Requirement 2.4.3.14

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword	SecondaryKeyword
2.4.3.14	As part of the Security Policy, publish the organisation's conflict resolution process for Vulnerability Disclosures.	Mandatory for Class 1 and above	business	process

Requirement 2.4.3.13

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword	SecondaryKeyword
2.4.3.13	As part of the Security Policy, develop a conflict resolution process for Vulnerability Disclosures.	Mandatory for all classes	business	process

Requirement 2.4.3.25

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword	SecondaryKeyword
2.4.3.25	Where a remote software upgrade can be supported by the device, there should be a transparent and auditable policy with a schedule of actions of an appropriate priority, to fix any vulnerabilities in a timely manner.	Mandatory for Class 2 and above	business	policy

Requirement 2.4.3.9.1

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword	SecondaryKeyword
2.4.3.9.1	There is a minimum support period during which security updates will be made available to all stakeholders.	Mandatory for all classes	business	process

Requirement 2.4.3.4

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword	SecondaryKeyword
2.4.3.4	The company follows industry standard cyber security recommendations.	Mandatory for all classes	business	policy

Requirement 2.4.3.24

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword	SecondaryKeyword
2.4.3.24	There is a named owner responsible for assessing third party (including open-sourced) supplied components (hardware and software) used in the product	Mandatory for Class 2 and above	business	responsibility

Requirement 2.4.3.7

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword	SecondaryKeyword
2.4.3.7	Processes and plans are in place based upon the IoTSF “Vulnerability Disclosure Guidelines” [ref 19], or a similar recognised process, to deal with the identification of a security vulnerability or compromise when they occur.	Mandatory for all classes	business	policy

Requirement 2.4.3.21

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword	SecondaryKeyword
2.4.3.21	There is a point of contact for third party suppliers and open source communities to raise security issues.	Mandatory for Class 1 and above	business	process

Requirement 2.4.3.18

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword	SecondaryKeyword
2.4.3.18	Where the a device may be used in real-time or high-availability systems, a procedure must be defined for notifying operators of connected components and system management of impending downtime for updates. In such real time or high availability system the end user should be able to decide whether to automatically install updates or to chose to manually install an update at a time of their choosing (or to ignore an update).	Mandatory for Class 2 and above	business	process

Requirement 2.4.3.3

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword	SecondaryKeyword
2.4.3.3	Intentionally left blank to maintain requirement numbering	-		

Requirement 2.4.3.28

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword	SecondaryKeyword
2.4.3.28	As part of the procurement policy, a supplier should be awarded a higher score where they demonstrate that they implement secure design in accordance with industry implementation standards or guidelines.	Mandatory for all classes	business	policy

Requirement 2.4.3.22.1

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword	SecondaryKeyword
2.4.3.22.1	Users must have the ability to disable updating.	Mandatory for Class 1 and above	business	process

Requirement 2.4.3.12

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword	SecondaryKeyword
2.4.3.12	As part of the Security Policy, provide a dedicated security email address and/or secure online page for Vulnerability Disclosure communications.	Mandatory for all classes	business	policy

Requirement 2.4.3.10

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword	SecondaryKeyword
2.4.3.10	A security threat and risk assessment shall have been carried out using a standard methodology appropriate to IoT products and services, to determine the risks and evolving threats before a design is started -this should cover the entire system being assessed.	Mandatory for Class 1 and above	business	process

Requirement 2.4.3.9

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword	SecondaryKeyword
2.4.3.9	There is a secure notification process based upon the IoTSF “Vulnerability Disclosure Guidelines” [ref 19], ISO/IEC 29147, or a similar recognised process, for notifying partners/users of any security updates, and what vulnerability is addressed by the update.			
Mandatory for all classes	business	process		

Requirement 2.4.3.20

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword	SecondaryKeyword
2.4.3.20	Responsibility is allocated for control, logging and auditing of the update process.	Mandatory for Class 2 and above	business	process

Requirement 2.4.3.27

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword	SecondaryKeyword
2.4.3.27	As part of the security policy, define how security requirements on third party components and services (including open-source) will be established and assessed.	Mandatory for all classes	business	policy

Requirement 2.4.3.11

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword	SecondaryKeyword
2.4.3.11	As part of the Security Policy, include a specific contact and web page for Vulnerability Disclosure reporting.	Mandatory for all classes	business	policy

Requirement 2.4.3.2

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword	SecondaryKeyword
2.4.3.2	There is a person or role, who takes ownership for adherence to this assurance framework process.	Mandatory for all classes	business	responsibility

2.4.13 Cloud And Network Elements

This section's intended audience is for those personnel who are responsible for the security of the IoT Product or Services Cloud or Network Systems.

Requirement 2.4.13.5

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword
2.4.13.5	The Product Manufacturer or Service Provider has a process to monitor the relevant security advisories to ensure all the product related web servers use protocols with no publicly known weaknesses.	Mandatory for Class 1 and above	Business

Requirement 2.4.13.17

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword
2.4.13.17	All the related servers and network elements support access control measures to restrict access to sensitive information or system processes to privileged accounts.	Mandatory for Class 1 and above	System

Requirement 2.4.13.19

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword
2.4.13.19	If run as a cloud service, the service meets industry standard cloud security principles.	Advisory for all classes	System

Requirement 2.4.13.15

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword
2.4.13.15	Brute force attacks are impeded by introducing escalating delays following failed user account login attempts, and/or a maximum permissible number of consecutive failed attempts.	Mandatory for Class 1 and above	System

Requirement 2.4.13.20

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword
2.4.13.20	Where a Product or Services includes any safety critical or life-impacting functionality, the services infrastructure shall incorporate protection against DDOS attacks, such as dropping of traffic or sink-holing.	Mandatory for Class 2 and above	System

Requirement 2.4.13.6

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword
2.4.13.6	The product related web servers support appropriately secure TLS/DTLS ciphers and disable/remove support for deprecated ciphers.	Advisory for all classes	System

Requirement 2.4.13.4

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword
2.4.13.4	All the product related web servers’ TLS certificate(s) are signed by trusted certificate authorities; are within their validity period; and processes are in place for their renewal.	Mandatory for Class 1 and above	System

Requirement 2.4.13.26

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword
2.4.13.26	Product-related cloud services bind API keys to specific IoT applications and are not installed on non-authorised devices.	Mandatory for Class 2 and above	System

Requirement 2.4.13.16

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword
2.4.13.16	All the related servers and network elements store any passwords using a cryptographic implementation using industry standard cryptographic algorithms.	Mandatory for Class 1 and above	System

Requirement 2.4.13.3

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword
2.4.13.3	All product related web servers have their webserver HTTP trace and trace methods disabled.	Mandatory for Class 1 and above	System

Requirement 2.4.13.29

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword
2.4.13.29	Product-related cloud service databases are encrypted during storage.	Mandatory for Class 1 and above	System

Requirement 2.4.13.2

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword
2.4.13.2	Any product related web servers have their webserver identification options (e.g. Apache or Linux) switched off.	Mandatory for Class 1 and above	System

Requirement 2.4.13.18

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword
2.4.13.18	All the related servers and network elements prevent anonymous/guest access except for read only access to public information.	Mandatory for Class 1 and above	System

Requirement 2.4.13.33

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword
2.4.13.33	Product-related cloud services monitor for compliance with connection policies and report out-of-compliance connection attempts.	Mandatory for Class 2 and above	System

Requirement 2.4.13.35

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword
2.4.13.35	Any personal data communicated between the mobile app and the device shall be encrypted. Where the data includes sensitive personal data then the encryption must be appropriately secure.	Mandatory for Class 2 and above	System

Requirement 2.4.13.9

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword
2.4.13.9	Where a product related to a webserver encrypts communications using TLS and requests a client certificate, the server(s) only establishes a connection if the client certificate and its chain of trust are valid.	Mandatory for Class 1 and above	System

Requirement 2.4.13.1

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword
2.4.13.1	All the product related cloud and network elements have the latest operating system(s) security updates implemented and processes are in place to keep them updated.	Mandatory for Class 2 and above	Business

Requirement 2.4.13.25

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword
2.4.13.25	Where device identity and/or configuration registries (e.g., "thing shadows") are implemented to "on-board" devices within a cloud service, the registries are configured to restrict access to only authorized administrators.	Mandatory for Class 1 and above	System

Requirement 2.4.13.12

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword
2.4.13.12	Intentionally left blank to maintain requirement numbering	-	

Requirement 2.4.13.30

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword
2.4.13.30	Product-related cloud service databases restrict read/write access to only authorized individuals, devices and services.	Mandatory for Class 1 and above	System

Requirement 2.4.13.23

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword
2.4.13.23	If run as a cloud service, the cloud service TCP based communications (such as MQTT connections) are encrypted and authenticated using the latest TLS standard.	Mandatory for Class 1 and above	System

Requirement 2.4.13.32

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword
2.4.13.32	When implemented as a cloud service, all remote access to cloud services is via secure means (e.g. SSH).	Mandatory for Class 1 and above	System

Requirement 2.4.13.7

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword
2.4.13.7	The product related web servers have repeated renegotiation of TLS connections disabled.	Mandatory for Class 1 and above	System

Requirement 2.4.13.11

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword
2.4.13.11	All the related servers and network elements prevent the use of null or blank passwords.	Mandatory for Class 1 and above	System

Requirement 2.4.13.36

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword
2.4.13.36	Subject to user permission, telemetry data from the device should be analysed for anomalous behaviour to detect malfunctioning or malicious activity.	Mandatory for Class 2 and above	System

Requirement 2.4.13.14

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword
2.4.13.14	All the related servers and network elements enforce passwords that follows industry good practice.	Mandatory for Class 1 and above	System

Requirement 2.4.13.31

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword
2.4.13.31	Product-related cloud services are designed using a defence-in-depth architecture consisting of Virtual Private Clouds (VPCs), firewalled access, and cloud-based monitoring.	Mandatory for Class 1 and above	System

Requirement 2.4.13.27

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword
2.4.13.27	Product-related cloud services API keys are not hard-coded into devices or applications.	Mandatory for all classes	System

Requirement 2.4.13.28

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword
2.4.13.28	If run as a cloud service, privileged roles are defined and implemented for any gateway/service that can configure devices.	Mandatory for Class 2 and above	System

Requirement 2.4.13.13

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword
2.4.13.13	Intentionally left blank to maintain requirement numbering	-	

Requirement 2.4.13.21

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword
2.4.13.21	Where a Product or Service includes any safety critical or life-impacting functionality, the services infrastructure shall incorporate redundancy to ensure service continuity and availability.	Mandatory for Class 1 and above	System

Requirement 2.4.13.8

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword
2.4.13.8	The related servers have unused IP ports disabled.	Mandatory for Class 1 and above	System

Requirement 2.4.13.34

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword
2.4.13.34	IoT edge devices should connect to cloud services using secure hardware and services (e.g. TLS using private keys stored in secure hardware).	Mandatory for Class 1 and above	System

Requirement 2.4.13.24

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword
2.4.13.24	If run as a cloud service, UDP-based communications are encrypted using the latest Datagram Transport Layer Security (DTLS).	Mandatory for Class 1 and above	System

Requirement 2.4.13.10

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword
2.4.13.10	Where a product related to a webserver encrypts communications using TLS, certificate pinning is implemented.	Advisory for all classes	System

Requirement 2.4.13.22

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword
2.4.13.22	Input data validation should be maintained in accordance with industry best practice methods.	Mandatory for Class 1 and above	System

2.4.15 Configuration

This section's intended audience is for those personnel who are responsible for the security of the device and IoT Services configurations.

Requirement 2.4.15.1

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword	SecondaryKeyword
2.4.15.1	The configuration of the device and any related web services is secure and tamper resistant i.e. sensitive configuration parameters should only be changeable by authorised people (evidence should list the parameters and who is authorised to change e.g. Owners / Guests). Sensitive parameters include cryptographic configuration settings.	Mandatory for Class 1 and above	Business	Process

Requirement 2.4.15.3

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword	SecondaryKeyword
2.4.15.3	The manufacturer should provide users with guidance on how to check whether their device is securely set up.	Mandatory for Class 1 and above	Business	Process

Requirement 2.4.15.2

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword	SecondaryKeyword
2.4.15.2	Updates to configuration should be provisioned securely and just-in-time, maintaining consistency . Irrelevant components of the configuration must be removed at the same time.	Mandatory for Class 1 and above	Business	Process

2.4.4 Device Hardware

This section's intended audience is those personnel who are responsible for hardware and mechanical quality. Guidance is available from the IoTSF [ref 44] regarding Physical Security (part B) Secure Boot (part C) and Secure Operating Systems (part D).

Requirement 2.4.4.13

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword	SecondaryKeyword
2.4.4.13	In production devices the microcontroller/ microprocessor(s) shall not allow the firmware to be read out of the products non-volatile [FLASH] memory. Where a separate non-volatile memory device is used the contents shall be encrypted.	Mandatory for Class 1 and above	System	Hardware

Requirement 2.4.4.11

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword	SecondaryKeyword
2.4.4.11	Tamper Evident measures have been used to identify any interference to the assembly to the end user.	Mandatory for Class 2 and above	System	Hardware

Requirement 2.4.4.2

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword	SecondaryKeyword
2.4.4.2	The product’s processor system has an irrevocable “Trusted Root Hardware Secure Boot”.	Mandatory for Class 2 and above	System	Hardware

Requirement 2.4.4.8

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword	SecondaryKeyword
2.4.4.8	The hardware incorporates physical, electrical & logical protection against reverse engineering. The level of protection must be determined by the risk assessment.	Mandatory for Class 3 and above	System	Hardware

Requirement 2.4.4.17

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword	SecondaryKeyword
2.4.4.17	The product shall have a hardware source for generating true random numbers.	Mandatory for Class 2 and above	System	Hardware

Requirement 2.4.4.6

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword	SecondaryKeyword
2.4.4.6	The hardware incorporates protection against tampering and this has been enabled.			
The level of tamper protection must be determined by the risk assessment.	Mandatory for Class 1 and above	System	Hardware	

Requirement 2.4.4.15

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword	SecondaryKeyword
2.4.4.15	Where a production device has a CPU watchdog, it is enabled and will reset the device in the event of any unauthorised attempts to pause or suspend the CPU's execution.	Mandatory for Class 1 and above	System	Hardware

Requirement 2.4.4.3

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword	SecondaryKeyword
2.4.4.3	The product’s processor boot process provides an appropriate level of trustworthiness by using a hardware root of trust (RoT) to verify trusted boot or measured boot methods. This may be referred to as 'secure boot', but absolute security cannot be assured.	Mandatory for Class 3 and above	System	Hardware

Requirement 2.4.4.9

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword	SecondaryKeyword
2.4.4.9	All communications port(s) which are not used as part of the product’s normal operation are not physically accessible or only communicate with authorised and authenticated entities.	Mandatory for Class 1 and above	System	Hardware Physical Software

Requirement 2.4.4.14

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword	SecondaryKeyword
2.4.4.14	Where the product's credential/key storage is external to its processor, the storage and processor shall be cryptographically paired to prevent the credential/key storage being used by unauthorised software.	Mandatory for Class 1 and above	System	Hardware

Requirement 2.4.4.16

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword	SecondaryKeyword
2.4.4.16	Where the product has a hardware source for generating true random numbers, it is used for all relevant cryptographic operations including nonce, initialisation vector and key generation algorithms.	Mandatory for Class 1 and above	System	Hardware Software

Requirement 2.4.4.7

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword	SecondaryKeyword
2.4.4.7	The hardware incorporates physical, electrical and logical protection against tampering to reduce the attack surface. The level of protection must be determined by the risk assessment.	Mandatory for Class 2 and above	System	Hardware Physical

Requirement 2.4.4.12

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword	SecondaryKeyword
2.4.4.12	Intentionally left blank to maintain requirement numbering	-		

Requirement 2.4.4.1

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword	SecondaryKeyword
2.4.4.1	The product’s processor system has an irrevocable hardware Secure Boot process.	Mandatory for all classes	System	Hardware

Requirement 2.4.4.5

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword	SecondaryKeyword
2.4.4.5	Any debug interface only communicates with authorised and authenticated entities on the production devices.(note: 2.4.4.6-8 should be considered as advisory)			
The functionality of any interface should be minimised to its essential task(s).	Mandatory for Class 1 and above	System	Hardware Software	

Requirement 2.4.4.10

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword	SecondaryKeyword
2.4.4.10	All the product’s development test points are securely disabled or removed wherever possible in production devices.	Mandatory for Class 2 and above	System	Hardware Physical

Requirement 2.4.4.4

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword	SecondaryKeyword
2.4.4.4	The Secure Boot process is enabled by default.	Mandatory for all classes	System	Hardware

2.4.7 Device Interfaces

This section's intended audience is for those personnel who are responsible for device security. Guidance is available from the IoTSF Best Practice Guidelines [ref 44] regarding Credential Management (part F) and Network Connections (part H).

Requirement 2.4.7.18

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword
2.4.7.18	The product only initialises and enables the communications interfaces, network protocols, application protocols and network services necessary for the product’s operation.	Mandatory for Class 1 and above	System

Requirement 2.4.7.5

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword
2.4.7.5	If a potential unauthorised change is detected (e.g.: an access fails authentication or integrity checks), the device should alert the user/administrator to the issue and should not connect to wider networks than those necessary to perform the alerting function.		
Failed attempts should be logged, but without providing any information about the failure to the initiator.	Mandatory for Class 1 and above	System	

Requirement 2.4.7.9

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword
2.4.7.9	Where a wireless interface has an initial pairing process, the passkeys are changed from the factory issued, or reset password prior to providing normal service.	Mandatory for all classes	Business

Requirement 2.4.7.13

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword
2.4.7.13	Where a TCP protocol, such as MQTT, is used, it is protected by a TLS connection with no known vulnerabilities.	Mandatory for Class 1 and above	System

Requirement 2.4.7.2

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword
2.4.7.2	The network component and firewall (if applicable) configuration has been reviewed and documented for the required/defined secure behaviour.	Mandatory for Class 1 and above	Business

Requirement 2.4.7.23

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword
2.4.7.23	Protocol anonymity features are enabled in protocols (e.g., Bluetooth) to limit location tracking capabilities.	Advisory for all classes	System

Requirement 2.4.7.15

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword
2.4.7.15	Where cryptographic suites are used such as TLS, all cipher suites shall be listed and validated against the current security recommendations such as NIST 800-131A [ref 2] or OWASP. Where insecure ciphers suites are identified they shall be removed from the product.	Mandatory for Class 1 and above	Business

Requirement 2.4.7.12

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword
2.4.7.12	All network communications keys are stored securely, in accordance with industry standards.	Mandatory for Class 1 and above	System

Requirement 2.4.7.7

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword
2.4.7.7	If a connection requires a password or passcode or passkey for connection authentication, the factory issued or reset password is unique to each device.	Mandatory for all classes	Business

Requirement 2.4.7.11

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword
2.4.7.11	Where WPA-2 WPS is used it has a unique, random key per device and enforces exponentially increasing retry attempt delays.	Mandatory for Class 1 and above	System

Requirement 2.4.7.1

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword
2.4.7.1	The product prevents unauthorised connections to it or other devices the product is connected to.	Mandatory for Class 1 and above	System

Requirement 2.4.7.20

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword
2.4.7.20	Post product launch, communications protocols should be reviewed throughout the product life cycle against publicly known vulnerabilities and changed to the most secure versions available if appropriate.	Mandatory for Class 1 and above	Business

Requirement 2.4.7.24

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword
2.4.7.24	As far as reasonably possible, devices should remain operating and locally functional in the case of a loss of network connection.		
Mandatory for Class 1 and above	System		

Requirement 2.4.7.16

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword
2.4.7.16	All use of cryptography by the product, such as TLS cipher suites, shall be listed and validated against the import/export requirements for the territories where the product is to be sold and/or shipped.	Mandatory for Class 1 and above	Business

Requirement 2.4.7.6

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword
2.4.7.6	All the product's unused ports (or interfaces) are closed and only the necessary ones are active.	Mandatory for Class 1 and above	Business

Requirement 2.4.7.8

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword
2.4.7.8	Where using initial pairing process, a Strong Authentication shall be used, requiring physical interaction with the device or possession of a shared secret.	Mandatory for Class 1 and above	System

Requirement 2.4.7.21

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword
2.4.7.21	If a factory reset is made, the device should warn that secure operation may be compromised until updated.	Mandatory for Class 1 and above	System

Requirement 2.4.7.10

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword
2.4.7.10	For any Wi-Fi connection, WPA-2 AES [ref 51] or a similar strength encryption has been used. Migration to the latest standard should be planned.(e.g. WPA3).		
Older insecure protocols such as WEP, WPA/WPA2 (Auto), WPA-TKIP and WPA-2 TKIP/AES (Mixed Mode) are disabled.	Mandatory for Class 1 and above	System	

Requirement 2.4.7.4

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword
2.4.7.4	Devices support only the versions of application layer protocols that have been reviewed and evaluated against publicly known vulnerabilities.	Mandatory for Class 1 and above	Business

Requirement 2.4.7.17

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword
2.4.7.17	Where there is a loss of communications or availability it shall not compromise the local integrity of the device.	Mandatory for Class 1 and above	System

Requirement 2.4.7.25

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword
2.4.7.25	Following restoration of power or network connection, devices should be able to return to a network in a sensible state and in an orderly fashion, rather than in a massive scale reconnect, which collectively could overwhelm a network.	Mandatory for Class 1 and above	System

Requirement 2.4.7.14

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword
2.4.7.14	Where a UDP protocol is used, such as CoAP, it is protected by a DTLS connection with no known vulnerabilities.	Mandatory for Class 1 and above	System

Requirement 2.4.7.3

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword
2.4.7.3	To prevent bridging of security domains within products with network interfaces, forwarding functions should be blocked by default.	Mandatory for Class 1 and above	System

Requirement 2.4.7.22

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword
2.4.7.22	Where RF communications are enabled (e.g., ZigBee, etc.) antenna power is configured to limit ability of mapping assets to limit attacks such as WAR-Driving.	Advisory for all classes	System

Requirement 2.4.7.19

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword
2.4.7.19	Communications protocols should be latest versions with no publicly known vulnerabilities and/or appropriate for the product.	Mandatory for Class 1 and above	Business

2.4.6 Device Operating System

This section's intended audience are the personnel responsible for the selection of a third-party Operating System or assessing the quality of 'in-house' developed schedulers and control sequencers quality. The term Operating System (OS) is below used for sake of brevity to imply all such options. Guidance is available from the IoTSF [ref 44] regarding Secure Operating Systems (part D).

Requirement 2.4.6.1

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword
2.4.6.1	The OS is implemented with relevant security updates prior to release.		
Mandatory for Class 2 and above	Business		

Requirement 2.4.6.9

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword
2.4.6.9	All software is operated at the least privilege level possible and only has access to the resources needed as controlled through appropriate access control mechanisms.	Mandatory for Class 1 and above	System

Requirement 2.4.6.5

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword
2.4.6.5	Security parameters and passwords should not be hard-coded into source code or stored in a local file. If passwords absolutely must be stored in a local file, then the password file(s) are owned by, and are only accessible to and writable by, the Device's OS most privileged account and are obfuscated.	Mandatory for Class 1 and above	System

Requirement 2.4.6.14

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword
2.4.6.14	The Product OS should be reviewed for known security vulnerabilities particularly in the field of cryptography prior to each update and after release. Cryptographic algorithms, primitives, libraries and protocols should be updateable to address any vulnerabilities.	Mandatory for Class 1 and above	System

Requirement 2.4.6.8

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword
2.4.6.8	All of the product’s OS kernel and services or functions are disabled by default unless specifically required.		
Essential kernel, services or functions are prevented from being called by unauthorised external product level interfaces and applications.	Mandatory for Class 1 and above	System	

Requirement 2.4.6.13

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword
2.4.6.13	The product’s OS kernel is designed such that each component runs with the least security privilege required (e.g. a microkernel architecture), and the minimum functionality needed (2.4.6.6 - 2.4.6.8 requires non-essential components are disabled or removed).	Mandatory for Class 2 and above	System

Requirement 2.4.6.12

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword
2.4.6.12	The OS implements a separation architecture to separate trusted from untrusted applications.	Mandatory for Class 2 and above	System

Requirement 2.4.6.10

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword
2.4.6.10	All the applicable security features supported by the OS are enabled.	Mandatory for Class 1 and above	System

Requirement 2.4.6.4

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword
2.4.6.4	Files, directories and persistent data are set to minimum access privileges required to correctly function.	Mandatory for Class 1 and above	System

Requirement 2.4.6.6

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword
2.4.6.6	All OS non-essential services have been removed from the product's software, image or file systems.	Mandatory for Class 1 and above	System

Requirement 2.4.6.3

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword
2.4.6.3	All unnecessary accounts or logins have been disabled or eliminated from the software at the end of the software development process, e.g. development or debug accounts and tools.	Mandatory for Class 1 and above	System

Requirement 2.4.6.7

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword
2.4.6.7	All OS command line access to the most privileged accounts has been removed from the OS.	Mandatory for Class 1 and above	System

Requirement 2.4.6.11

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword
2.4.6.11	The OS is separated from the application(s) and is only accessible via defined secure interfaces.	Mandatory for Class 1 and above	System

Requirement 2.4.6.15

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword
2.4.6.15	As per 2.4.10.5, the user interface is protected by an automatic session idle logout timeout function.	Mandatory for Class 1 and above	System

Requirement 2.4.6.2

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword
2.4.6.2	Intentionally left blank to maintain requirement numbering	-	

2.4.16 Device Ownership Transfer

This section's intended audience is for those personnel who are responsible for Data Protection and Device Ownership management.

Requirement 2.4.16.6

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword	SecondaryKeyword
2.4.16.6	The device manufacturer ensures that the exposed identity of the device cannot be linked by unauthorised actors to the end user, to ensure anonymity and comply with relevant local data privacy laws e.g. GDPR [ref 14] in the EU.	Mandatory for Class 1 and above	Business	Policy

Requirement 2.4.16.7

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword	SecondaryKeyword
2.4.16.7	Where transfer of a device to a new end user is supported, user settings and confidential user data on the device should be reliably erasable by triggering a user reset function. This is so the new user can be confident in the device state and also so the previous user can be confident their data has been unrecoverably erased to maintain confidentiality (see alongside 2.4.12.13 and 2.4.12.11).	Mandatory for Class 1 and above	Business	Policy

Requirement 2.4.16.5

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword	SecondaryKeyword
2.4.16.5	The device registration with the Service Provider shall use a secure connection.	Mandatory for Class 1 and above	Business	Process

Requirement 2.4.16.3

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword	SecondaryKeyword
2.4.16.3	The Service Provider should not have the ability to do a reverse lookup of device ownership from the device identity.	Mandatory for Class 1 and above	Business	Process

Requirement 2.4.16.4

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword	SecondaryKeyword
2.4.16.4	If ownership change is required/allowed, the device must have an irrevocable method of decommissioning and recommissioning.	Mandatory for Class 1 and above	System	Software

Requirement 2.4.16.1

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword	SecondaryKeyword
2.4.16.1	Where a device may have its ownership transferred to a different owner, the supplier or manufacturer of any devices and/or services shall provide information about how the device(s) removal and/or disposal or replacement shall be carried out to maintain the end user’s privacy and security, including deletion of all Personal Information from the device and any associated services.			
This option must be available when a transfer of ownership occurs or when an end user wishes to delete their Personal Information from the service or device.	Mandatory for Class 1 and above	Business	Process	

Requirement 2.4.16.2

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword	SecondaryKeyword
2.4.16.2	Where a device User wishes to dispose of the device or end the service, the supplier or manufacturer of any devices and/or services shall provide information about how the device(s) removal and/or disposal or replacement shall be carried out to maintain the end user's privacy and security, including secure erasure of all Personal Information from the device and deletion of personal information from any associated services (other than that required for legitimate reasons such as billing).			
A clear confirmation is provided to the user.				
Examples of a user include a renter of accommodation, a vehicle or medical aids.	Mandatory for Class 1 and above	Business	Process	

2.4.5 Device Software

This section's intended audience is for those personnel who are responsible for device application quality e.g. Software Architects, Product Owners, and Release Managers. Guidance is available from the IoTSF [ref 44] regarding Secure Operating Systems (part D), Credential Management (part F), and Software Updates (part J).

Requirement 2.4.5.31

ReqNo	Requirement	AssuranceClassAndApplicability
2.4.5.31	Withdrawn as duplicate requirement	

Requirement 2.4.5.11

ReqNo	Requirement	AssuranceClassAndApplicability
2.4.5.11	Development software versions have any debug functionality switched off if the software is operated on the product outside of the product vendor’s trusted environment.	Mandatory for Class 2 and above

Requirement 2.4.5.40

ReqNo	Requirement	AssuranceClassAndApplicability
2.4.5.40	Hard-coded critical/ security parameters in device software source code shall not be used; if needed these should be injected in a separate (secure) process.	Mandatory for all classes

Requirement 2.4.5.30

ReqNo	Requirement	AssuranceClassAndApplicability
2.4.5.30	An update to a device must be authenticated before it is installed. Where the update fails authentication, the device should, if possible, revert to the last known good (current stable) configuration/software image which was stored on the device.	
Mandatory for all classes		

Requirement 2.4.5.25

ReqNo	Requirement	AssuranceClassAndApplicability
2.4.5.25	Support for partially installing updates is provided for devices whose on-time is insufficient for the complete installation of a whole update (constrained devices).	Advisory for all classes

Requirement 2.4.5.19

ReqNo	Requirement	AssuranceClassAndApplicability
2.4.5.19	Where present, production software signing keys are under access control.	Mandatory for all classes

Requirement 2.4.5.2

ReqNo	Requirement	AssuranceClassAndApplicability
2.4.5.2	Where remote software updates can be supported by the device, the software images must be digitally signed by an appropriate signing authority - e.g. manufacturer/supplier or public. The Signing Authority should be clearly identified.	Mandatory for all classes

Requirement 2.4.5.5

ReqNo	Requirement	AssuranceClassAndApplicability
2.4.5.5	If the product has any virtual port(s) that are not required for normal operation, they are only allowed to communicate with authorised and authenticated entities or are securely disabled when shipped. When a port is initialised or used for field diagnostics, the port input commands are deactivated and the output provides no information which could compromise the device, such as credentials, memory address or function names.	Mandatory for Class 2 and above

Requirement 2.4.5.20

ReqNo	Requirement	AssuranceClassAndApplicability
2.4.5.20	The production software signing keys are stored and secured in a storage device compliant to FIPS-140-2/FIPS-140-3 level 2, or equivalent or higher standard.	Mandatory for Class 1 and above

Requirement 2.4.5.39

ReqNo	Requirement	AssuranceClassAndApplicability
2.4.5.39	IoT devices must allow software updates to maintain security over the product lifetime.	Mandatory for Class 2 and above

Requirement 2.4.5.36

ReqNo	Requirement	AssuranceClassAndApplicability
2.4.5.36	Updates should be provided for a period appropriate to the device, and this period shall be made clear to a user when supplying the device. Updates should, where possible, be configurable to be automatically or manually installed. The supply chain partners should inform the user that an update is required.	Mandatory for all classes

Requirement 2.4.5.12

ReqNo	Requirement	AssuranceClassAndApplicability
2.4.5.12	Steps have been taken to protect the product's software from sensitive information leakage, including at network interfaces during initialisation, and side-channel attacks.	Mandatory for Class 3 and above

Requirement 2.4.5.10

ReqNo	Requirement	AssuranceClassAndApplicability
2.4.5.10	Production software images shall be compiled in such a way that all unnecessary debug and symbolic information is removed, to prevent accidental release of superfluous data.	Mandatory for Class 1 and above

Requirement 2.4.5.18

ReqNo	Requirement	AssuranceClassAndApplicability
2.4.5.18	The build environment and toolchain used to create the software is under configuration management and version control, and its integrity is validated regularly.	Mandatory for Class 2 and above

Requirement 2.4.5.27

ReqNo	Requirement	AssuranceClassAndApplicability
2.4.5.27	Where real-time expectations of performance are present, update mechanisms must not interfere with meeting these expectations (e.g. by running update processes at low priority, or notifying the user of the priority and duration of the update and with the option of postponing or disabling the update).	Mandatory for all classes

Requirement 2.4.5.6

ReqNo	Requirement	AssuranceClassAndApplicability
2.4.5.6	To prevent the stalling or disruption of the device’s software operation, watchdog timers are present, and cannot be disabled.	Mandatory for Class 1 and above

Requirement 2.4.5.38

ReqNo	Requirement	AssuranceClassAndApplicability
2.4.5.38	Maintenance changes should trigger full security regression testing.	Mandatory for Class 2 and above

Requirement 2.4.5.15

ReqNo	Requirement	AssuranceClassAndApplicability
2.4.5.15	The software must be architected to identify and ring fence sensitive software components, including cryptographic processes, to aid inspection, review and test. The access from other software components must be controlled and restricted to known and acceptable operations. For example security related processes should be executed at higher privilege levels in the application processor hardware.	Mandatory for Class 1 and above

Requirement 2.4.5.4

ReqNo	Requirement	AssuranceClassAndApplicability
2.4.5.4	If remote software upgrade is supported by a device, software images shall be encrypted or transferred over an encrypted channel.	Mandatory for Class 2 and above

Requirement 2.4.5.34

ReqNo	Requirement	AssuranceClassAndApplicability
2.4.5.34	Any caches which potentially store sensitive material are cleared flushed after memory locations containing sensitive material have been sanitised.	Mandatory for Class 3 and above

Requirement 2.4.5.23

ReqNo	Requirement	AssuranceClassAndApplicability
2.4.5.23	All inputs and outputs are checked for validity e.g. use “Fuzzing” tests to check for acceptable responses or output for both expected (valid) and unexpected (invalid) input stimuli.	Mandatory for Class 2 and above

Requirement 2.4.5.1

ReqNo	Requirement	AssuranceClassAndApplicability
2.4.5.1	The product has measures to prevent unauthorised and unauthenticated software, configurations and files being loaded onto it. If the product is intended to allow un-authenticated software, such software should only be run with limited permissions and/or sandbox.	Mandatory for all classes

Requirement 2.4.5.29

ReqNo	Requirement	AssuranceClassAndApplicability
2.4.5.29	Where a device cannot verify authenticity of updates itself (e.g. due to no cryptographic capabilities), only a local update by a physically present user is permitted and is their responsibility.	Mandatory for all classes

Requirement 2.4.5.24

ReqNo	Requirement	AssuranceClassAndApplicability
2.4.5.24	The software has been designed to meet the safety requirements identified in the risk assessment; for example in the case of unexpected invalid inputs, or erroneous software operation, the product does not become dangerous, or compromise security of other connected systems.	Mandatory for Class 2 and above

Requirement 2.4.5.13

ReqNo	Requirement	AssuranceClassAndApplicability
2.4.5.13	The product’s software source code follows the basic good practice of a Language subset coding standard.	Mandatory for Class 2 and above

Requirement 2.4.5.3

ReqNo	Requirement	AssuranceClassAndApplicability
2.4.5.3	Where updates are supported, the software update package has its digital signature, signing certificate and signing certificate chain verified by the device before the update process begins.	Mandatory for all classes

Requirement 2.4.5.33

ReqNo	Requirement	AssuranceClassAndApplicability
2.4.5.33	Memory locations used to store sensitive material (e.g. cryptographic keys, passwords/passphrases, etc.) are sanitised as soon as possible after they are no longer needed. These can include but are not limited to locations on the heap, the stack, and statically-allocated storage [ref 47].	Mandatory for Class 2 and above

Requirement 2.4.5.28

ReqNo	Requirement	AssuranceClassAndApplicability
2.4.5.28	Where a device doesn’t support secure boot, upon a firmware update the user data and credentials should be re-initialised.	Mandatory for all classes

Requirement 2.4.5.37

ReqNo	Requirement	AssuranceClassAndApplicability
2.4.5.37	The device manufacturer should ensure that shared libraries (e.g. Clib or Crypto libraries) that deliver network and security functionalities have been reviewed or evaluated (note that the actual review or evaluation does not have to be conducted by the manufacturer if it has been conducted by another reputable organisation or government entity). Cryptography libraries should be re-reviewed for known security vulnerabilities on each update of the device.	Mandatory for Class 2 and above

Requirement 2.4.5.17

ReqNo	Requirement	AssuranceClassAndApplicability
2.4.5.17	The build environment and toolchain used to compile the application is run on a build system with controlled and auditable access.	Mandatory for Class 2 and above

Requirement 2.4.5.16

ReqNo	Requirement	AssuranceClassAndApplicability
2.4.5.16	Software source code is developed, tested and maintained following defined repeatable processes.	Mandatory for Class 1 and above

Requirement 2.4.5.14

ReqNo	Requirement	AssuranceClassAndApplicability
2.4.5.14	The product’s software source code follows the basic good practice of static vulnerability analysis [ref 37] by the developer.	Mandatory for Class 2 and above

Requirement 2.4.5.8

ReqNo	Requirement	AssuranceClassAndApplicability
2.4.5.8	The product has protection against unauthorised reversion of the software to an earlier and potentially less secure version.	
Only authorised entities can restore the software to an earlier secure version.	Mandatory for Class 2 and above	

Requirement 2.4.5.26

ReqNo	Requirement	AssuranceClassAndApplicability
2.4.5.26	Support for partially downloading updates is provided for devices whose network access is limited or sporadic.	Advisory for all classes

Requirement 2.4.5.21

ReqNo	Requirement	AssuranceClassAndApplicability
2.4.5.21	Where the device software communicates with a product related webserver or application over TCP/IP or UDP/IP, the device software uses certificate pinning or public/private key equivalent, where appropriate.	Mandatory for Class 2 and above

Requirement 2.4.5.9

ReqNo	Requirement	AssuranceClassAndApplicability
2.4.5.9	There are measures to prevent the installation of non-production (e.g. development or debug) software onto production devices.	Mandatory for Class 1 and above

Requirement 2.4.5.41

ReqNo	Requirement	AssuranceClassAndApplicability
2.4.5.41	Where the device is capable, it should check after initialization, and then periodically, whether security updates are available, either autonomously or as part of the support service.	
Otherwise, the support service should push updates to the device.	Mandatory for Class 1 and above	

Requirement 2.4.5.32

ReqNo	Requirement	AssuranceClassAndApplicability
2.4.5.32	There is secure provisioning of cryptographic keys for updates during manufacture in accordance with industry standards.	Mandatory for Class 1 and above

Requirement 2.4.5.7

ReqNo	Requirement	AssuranceClassAndApplicability
2.4.5.7	The product’s software signing root of trust is stored in tamper-resistant memory.	Mandatory for Class 1 and above

Requirement 2.4.5.22

ReqNo	Requirement	AssuranceClassAndApplicability
2.4.5.22	For a device with no possibility of a software update, the conditions for and period of replacement support should be clear.	
A replacement strategy must be communicated to the user, including a schedule for when the device should be replaced or isolated.	Mandatory for all classes	

Requirement 2.4.5.35

ReqNo	Requirement	AssuranceClassAndApplicability
2.4.5.35	An end-of-life policy shall be published which explicitly states the minimum length of time for which a device will receive software updates and the reasons for the length of the support period. The need for each update should be made clear to users and an update should be easy to implement.	
At the end of the support period, the device should reduce the risk of a latent vulnerability being exploited. This could be by indicating an error condition to the user or curtailing functionality. This action should be clearly communicated to the user during the procurement stage.	Mandatory for all classes	

2.4.9 Encryption And Key Management For Hardware

This section's intended audience is for those personnel who are responsible for the security of the IoT systems hardware key management and encryption. Guidance is available from the IoTSF [ref 44] regarding Encryption (Part G).

Requirement 2.4.9.1

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword	SecondaryKeyword
2.4.9.1	Intentionally left blank to maintain requirement numbering	-		

Requirement 2.4.9.6

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword	SecondaryKeyword
2.4.9.6	All the product related cryptographic functions are sufficiently secure for the lifecycle of the product, or cryptographic algorithms and primitives should be updateable ("cryptoagility")".	Mandatory for Class 1 and above	Business	Process

Requirement 2.4.9.2

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword	SecondaryKeyword
2.4.9.2	If present, a true random number generator source has been validated for true randomness.	Mandatory for Class 2 and above	System	Hardware

Requirement 2.4.9.3

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword	SecondaryKeyword
2.4.9.3	There is a process for secure provisioning of security parameters and keys that includes random and individual (unique) generation, distribution, update, revocation and destruction.	Mandatory for Class 2 and above	Business	Process

Requirement 2.4.9.10

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword	SecondaryKeyword
2.4.9.10	All key lengths are sufficient for the level of assurance required.	Mandatory for Class 2 and above	Business	Policy

Requirement 2.4.9.4

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword	SecondaryKeyword
2.4.9.4	There is a secure method of key insertion that protects keys against copying.	Mandatory for Class 1 and above	System	Software

Requirement 2.4.9.7

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword	SecondaryKeyword
2.4.9.7	The product stores all sensitive unencrypted parameters (e.g. keys) in a secure, tamper-resistant location.	Mandatory for Class 1 and above	System	Hardware

Requirement 2.4.9.9

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword	SecondaryKeyword
2.4.9.9	In device manufacture, all asymmetric encryption private keys that are unique to each device are secured. They must be truly randomly internally generated or securely programmed into each device.	Mandatory for Class 2 and above	Business	Process

Requirement 2.4.9.11

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword	SecondaryKeyword
2.4.9.11	In systems with many layered sub devices, key management should follow best practice.	Mandatory for all classes	Business	Policy

Requirement 2.4.9.8

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword	SecondaryKeyword
2.4.9.8	The cryptographic key chain used for signing production software is different from that used for any other test, development or other software images or support requirement.	Advisory for all classes	System	Software

Requirement 2.4.9.5

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword	SecondaryKeyword
2.4.9.5	All the product related cryptographic functions have no publicly known unmitigated weaknesses in the algorithms or implementation, for example MD5 and SHA-1 are not used.	Mandatory for Class 1 and above	Business	Process

2.4.17 Infrastructure Management

This schema defines the structure for capturing and organizing the security requirements and standards compliance information as specified in the IoT Security Assurance Framework.

Requirement

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword	SecondaryKeyword

Requirement 2.4.16.3

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword	SecondaryKeyword
2.4.16.3	An Asset management policy for security related equipment	Mandatory for Class 1 and above	Business	Policy

Requirement 2.4.16.10

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword	SecondaryKeyword
2.4.16.10	HR Security Policy	Mandatory for Class 1 and above	Business	Policy

Requirement 2.4.16.5

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword	SecondaryKeyword
2.4.16.5	Data Backup processes for critical and secret data	Mandatory for Class 1 and above	Business	Process

Requirement 2.4.16.2

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword	SecondaryKeyword
2.4.16.2	A document final release process	Mandatory for Class 1 and above	Business	Process

Requirement 2.4.16.9

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword	SecondaryKeyword
2.4.16.9	Data destruction	Mandatory for Class 1 and above	Business	Policy

Requirement 2.4.16.8

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword	SecondaryKeyword
2.4.16.8	Security Risk Assesment	Mandatory for Class 1 and above	Business	Process

Requirement 2.4.16.6

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword	SecondaryKeyword
2.4.16.6	Access Control both Physical and for code repositories, and build artifacts	Mandatory for Class 1 and above	Business	Process

Requirement 2.4.17.1

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword	SecondaryKeyword
2.4.17.1	A documented Software Development Lifecycle,(SDLC)	Mandatory for Class 1 and above	Business	Process

Requirement 2.4.16.7

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword	SecondaryKeyword
2.4.16.7	Secure Assets and Key Management	Mandatory for Class 1 and above	Business	Policy

Requirement 2.4.16.4

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword	SecondaryKeyword
2.4.16.4	is there a defined Document Management classification process and management plan for material that contains security related informaton	Mandatory for Class 1 and above	Business	Process

2.4.11 Mobile Application

This section's intended audience is for those personnel who are responsible for the security of the IoT Product or Services Mobile Application. Guidance is available from the IoTSF [ref 44] regarding Application Security (part E) and Credential Management (part F).

Requirement 2.4.11.3

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword
2.4.11.3	The mobile application ensures that any related databases or files are either tamper resistant or restricted in their access. Upon detection of tampering of the databases or files, they are re-initialised.	Mandatory for Class 1 and above	System

Requirement 2.4.11.1

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword
2.4.11.1	Where an application’s user interface password is used for login authentication, the initial password or factory reset password is unique to each device in the product family.	Mandatory for all classes	System

Requirement 2.4.11.10

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword
2.4.11.10	Mobile Apps should be developed using best practice secure coding techniques and server frameworks.	Mandatory for Class 1 and above	System

Requirement 2.4.11.11

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword
2.4.11.11	App interface should provide a simple method (one to two clicks) to initiate any security update to the end device.	Mandatory for Class 1 and above	System

Requirement 2.4.11.7

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword
2.4.11.7	All data being transferred over interfaces should be validated where appropriate. This could include checking the data type, length, format, range, authenticity, origin and frequency.	Mandatory for Class 1 and above	System

Requirement 2.4.11.2

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword
2.4.11.2	Password entry follows industry standard practice.	Mandatory for all classes	System

Requirement 2.4.11.9

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword
2.4.11.9	All application inputs and outputs are validated using for example an allowed-list containing authorised origins of data and valid attributes of such data.	Mandatory for Class 1 and above	System

Requirement 2.4.11.12

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword
2.4.11.12	Access to device functionality via a network/web browser interface in the initialized state should only be permitted after successful Authentication using current best practice secure cryptographic modules.	Mandatory for Class 1 and above	System

Requirement 2.4.11.5

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword
2.4.11.5	The product securely stores any passwords using an industry standard cryptographic algorithm.	Mandatory for Class 1 and above	System

Requirement 2.4.11.13

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword
2.4.11.13	Any personal data communicated between the mobile app and the device shall be encrypted. Where the data includes sensitive personal data then the encryption must be appropriately secure.	Mandatory for Class 1 and above	System

Requirement 2.4.11.4

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword
2.4.11.4	Where the application communicates with a product related remote server(s), or device, it does so over a secure connection.	Mandatory for Class 1 and above	System

Requirement 2.4.11.8

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword
2.4.11.8	Secure Administration Interfaces; It is important that configuration management functionality is accessible only by authorised operators and administrators. Enforce Strong Authentication over administration interfaces, for example, by using certificates.	Mandatory for Class 1 and above	System

Requirement 2.4.11.6

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword
2.4.11.6	Where passwords are entered on a user interface, the actual pass phrase is obscured by default to prevent the capture of passwords.	Mandatory for Class 1 and above	System

2.4.12 Data Protection And Privacy

This section's intended audience is for those personnel who are responsible for Data Protection and Privacy regulatory compliance.

Requirement 2.4.12.4

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword
2.4.12.4	The product/service ensures that Personal Information is anonymised whenever possible and in particular in any reporting.	Mandatory for Class 1 and above	Business

Requirement 2.4.12.6

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword
2.4.12.6	There is a method or methods for the product owner to be informed about what Personal Information is collected, why, where it will be stored and processed, and by whom and for what purposes. This includes sensing capabilities, such as sound or video recording, biometrics, location, etc.	Mandatory for Class 1 and above	Business

Requirement 2.4.12.3

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword
2.4.12.3	The product/service ensures that only authorised personnel have access to personal data of users.	Mandatory for Class 1 and above	Business

Requirement 2.4.12.10

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword
2.4.12.10	The supplier or manufacturer of any devices or devices shall provide clear information about how the device(s) should be set up to maintain the end user's privacy and security.	Mandatory for all classes	Business

Requirement 2.4.12.9

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword
2.4.12.9	The supplier or manufacturer of any device shall provide documented information to end users about how the device(s) functions within the end user’s network may affect their privacy.	Advisory for all classes	Business

Requirement 2.4.12.7

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword
2.4.12.7	There is a method or methods for each user to check/verify what Personal Information is collected.	Mandatory for Class 1 and above	Business

Requirement 2.4.12.13

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword
2.4.12.13	Security of devices and services should be designed with usability in mind (reducing user decision points that may have a detrimental impact on privacy and security).	Mandatory for Class 1 and above	System

Requirement 2.4.12.8

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword
2.4.12.8	The product / service can be made compliant with the local and/or regional Personal Information protection legislation where the product is to be sold. For example GDPR [ref 14].	Mandatory for Class 1 and above	Business

Requirement 2.4.12.2

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword
2.4.12.2	The product/service ensures that all Personal Information is encrypted for confidentiality (both when stored and if communicated out of the device) and only accessible after successful authentication and authorisation.		
Note: authentication only proves who you are, but authorisation confirms if you are allowed access to the PI.			
The cryptography must be of sufficient strength to protect the Personal Information for however long it is expected to be retained (or remain confidential).	Mandatory for Class 3 and above	Business	

Requirement 2.4.12.14

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword
2.4.12.14	The product or service only records audio/visual/or any other data in accordance with the authorisation of the user (e.g., no passive recording without explicit authorisation).	Mandatory for Class 1 and above	System

Requirement 2.4.12.11

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword
2.4.12.11	The supplier or manufacturer of any devices and/or services shall provide information about how the device(s) removal and/or disposal or replacement shall be carried out to maintain the end user’s privacy and security, including deletion of all personal information from the device and any associated services.	Mandatory for Class 1 and above	Business

Requirement 2.4.12.12

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword
2.4.12.12	The supplier or manufacturer of any devices or services shall provide clear information about the end user’s responsibilities to maintain the devices and/or services privacy and security.	Mandatory for Class 1 and above	Business

Requirement 2.4.12.1

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword
2.4.12.1	The product/service stores the minimum amount of Personal Information from users required for the operation of the service.	Mandatory for Class 1 and above	Business

Requirement 2.4.12.5

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword
2.4.12.5	The Product Manufacturer or Service Provider shall ensure that a data retention policy is in place and documented for users.	Mandatory for Class 1 and above	Business

Requirement 2.4.12.15

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword
2.4.12.15	The supplier or manufacturer performs a privacy impact assessment (PIA) to identify Personally Identifiable Information (PII) and design approaches for safeguarding user privacy compliant with the legal requirements of the user's location (e.g. GDPR). This should extend to data gathered via Web APIs from third party platform suppliers.	Advisory for all classes	Business

Product Security And Telecommunications Infrastructure

Schema defining the necessary details of manufacturers, importers, and products under the Product Security and Telecommunications Infrastructure Act requirements. This includes identification, compliance, and security update commitments.

2.4.14 Secure Supply Chain And Production

This section's intended audience is for those personnel who are responsible for the security of the IoT Product or Services' Supply Chain and Production.

Requirement 2.4.14.10

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword	SecondaryKeyword
2.4.14.10	An authorised actor in physical possession of a device can discover and authenticate its RoT-backed logical identity e.g. for inspection, verification of devices being onboarded (this may need electrical connection).	Mandatory for Class 2 and above	Business	Process

Requirement 2.4.14.9

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword	SecondaryKeyword
2.4.14.9	In manufacture, all encryption keys that are unique to each device are either securely and truly randomly internally generated or securely programmed into each device in accordance with industry standard FIPS140-2 [ref 5] or equivalent. Any secret key programmed into a product at manufacture is unique to that individual device, i.e. no global secret key is shared between multiple devices, unless this is required by a licensing authority.	Mandatory for Class 2 and above	Business	Process

Requirement 2.4.14.8

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword	SecondaryKeyword
2.4.14.8	An auditable manifest of all libraries used within the product (open source, etc.) is maintained to inform vulnerability management throughout the device lifecycle and whole of the support period.	Advisory for all classes	Business	Process

Requirement 2.4.14.22

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword	SecondaryKeyword
2.4.14.22	The OEM retains authorisation of secure production control methods to prevent a third party manufacturer (CEM etc.) from producing overproduction and/or unauthorised devices.			
Mandatory for Class 2 and above	Business	Process		

Requirement 2.4.14.15

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword	SecondaryKeyword
2.4.14.15	Production assets are encrypted during transport to the intended production facility, area or system, or delivered via private channel. Examples of production assets include firmware images, device certificate CA keys, onboarding credentials, production tools and manufacturing files.	Mandatory for Class 2 and above	Business	Process

Requirement 2.4.14.17

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword	SecondaryKeyword
2.4.14.17	Steps have been taken to prevent inauthentic devices from being programmed with confidential firmware images and configuration data. This is to prevent IP theft and reverse engineering.	Mandatory for Class 2 and above	Business	Process

Requirement 2.4.14.4

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword	SecondaryKeyword
2.4.14.4	The production system for a device has a process to ensure that any devices with duplicate serial numbers are not shipped and are either reprogrammed or destroyed.	Mandatory for Class 1 and above	Business	Process

Requirement 2.4.14.2

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword	SecondaryKeyword
2.4.14.2	Any hardware design files, software source code and final production software images with full descriptive annotations are stored encrypted in off-site locations or by a 3rd party Escrow service.	Advisory for all classes	Business	Process

Requirement 2.4.14.18

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword	SecondaryKeyword
2.4.14.18	Steps have been taken to prevent inauthentic devices from being signed into certificate chains of trust or otherwise onboarded. For example, a policy or checklist describing which devices may be onboarded exists and is followed.	Mandatory for Class 2 and above	Business	Process

Requirement 2.4.14.20

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword	SecondaryKeyword
2.4.14.20	If time critical delivery of products is needed, availability of production resources accessed in real time over the Internet is assured, by providing them with alternative access channels not susceptible to DOS attacks.	Mandatory for all classes	Business	Process

Requirement 2.4.14.24

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword	SecondaryKeyword
2.4.14.24	An end of life disposal process shall be provided to ensure that retired devices are permanently disconnected from their cloud services and that any confidential user data is securely erased from both the device and the cloud services.			

| Mandatory for Class 1 and above | Business | Process

Requirement 2.4.14.25

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword	SecondaryKeyword
2.4.14.25	Where contractual supply arrangements and software licence agreements allow, a software bill of materials (SBOM) shall be available and notified (URL) to customers with product documentation.	Mandatory for Class 2 and above	Business	Process

Requirement 2.4.14.21

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword	SecondaryKeyword
2.4.14.21	Operators of production servers, computers and network equipment keep their software up to date and monitor them for signs of compromise e.g. unusual activity.	Mandatory for Class 2 and above	Business	Process

Requirement 2.4.14.5

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword	SecondaryKeyword
2.4.14.5	Where a product includes a trusted Secure Boot process, the entire production test and any related calibration is executed with the processor system operating in its secured boot, authenticated software mode.	Advisory for all classes	Business	Process

Requirement 2.4.14.7

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword	SecondaryKeyword
2.4.14.7	A cryptographic protected ownership proof shall be transferred along the supply chain and extended if a new owner is added in the chain. This process shall be based on open standards such as Enhanced Privacy ID, Certificates per definition in ISO 20008/20009 [ref 42].	Mandatory for Class 1 and above	Business	Process

Requirement 2.4.14.14

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword	SecondaryKeyword
2.4.14.14	Procedures for proper disposal of scrap product exist at manufacturing facilities, and compliance is monitored. This is to prevent scrap entering grey markets.	Mandatory for Class 2 and above	Business	Process

Requirement 2.4.14.13

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword	SecondaryKeyword
2.4.14.13	Products ship with information (documents or URL) about their operations and normal behaviour e.g. domains contacted, volume of messaging, Manufacturer Usage Description (MUD).	Mandatory for Class 2 and above	Business	Process

Requirement 2.4.14.12

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword	SecondaryKeyword
2.4.14.12	IoT devices' RoT-backed logical identity is used to identify them in logs of their physical chain of custody. This is to facilitate tracking through the supply chain.	Mandatory for Class 2 and above	Business	Process

Requirement 2.4.14.11

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword	SecondaryKeyword
2.4.14.11	Devices are shipped with readily-accessible physical identifiers derived from their RoT-backed IDs. This is to facilitate both tracking through the supply chain and for the user to identify the device-type/model and SKU throughout the support period.	Mandatory for Class 1 and above	Business	Process

Requirement 2.4.14.19

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword	SecondaryKeyword
2.4.14.19	Device certificate signing keys and other onboarding credentials are secured against unauthorised access. For example, they may be stored encrypted and managed or created by an HSM and delivered by the secure signing process.	Mandatory for Class 2 and above	Business	Process

Requirement 2.4.14.16

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword	SecondaryKeyword
2.4.14.16	Device firmware images and configuration data are secured against unauthorised modification in manufacturing environments, including during programming. If IP protection is required then the images and data need to be protected against unauthorised access.	Mandatory for Class 2 and above	Business	Process

Requirement 2.4.14.23

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword	SecondaryKeyword
2.4.14.23				
The supplier or manufacturer of any devices and/or services shall provide information about how the device(s) removal and/or disposal or replacement shall be carried out to maintain the end user’s privacy and security, including deletion of all personal information from the device and any associated services.	Mandatory for Class 2 and above	Business	Process	

Requirement 2.4.14.1

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword	SecondaryKeyword
2.4.14.1	Ensure the entire production test and calibration software used during manufacture is removed or secured before the product is dispatched from the factory. This is to prevent alteration of the product post manufacture when using authorised production software, for example hacking of the RF characteristics for greater RF ERP. Where such functionality is required in a service centre, it shall be removed upon completion of any servicing activities.	Mandatory for Class 2 and above	System	Software

Requirement 2.4.14.6

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword	SecondaryKeyword
2.4.14.6	A securely controlled area and process shall be used for device provisioning where the production facility is untrusted.	Advisory for all classes	Business	Process

Requirement 2.4.14.3

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword	SecondaryKeyword
2.4.14.3	In manufacture, all the devices are logged by the product vendor, utilizing unique tamper resistant identifiers such as serial number so that cloned or duplicated devices can be identified and either disabled or prevented from being used with the system.	Mandatory for Class 1 and above	Business	Process

2.4.10 Web User Interface

This section's intended audience is for those personnel who are responsible for the security of the IoT Product or Services Web Systems. Guidance is available from the IoTSF [ref 44] regarding Application Security (part E), and Credential Management (part F).

Requirement 2.4.10.6.1

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword
2.4.10.6.1	Strong passwords are required, and a random salt value is incorporated with the password.	Mandatory for Class 1 and above	System

Requirement 2.4.10.11

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword
2.4.10.11	Sanitise input in Web applications by using URL encoding or HTML encoding to wrap data and treat it as literal text rather than executable script.	Mandatory for Class 1 and above	System

Requirement 2.4.10.13

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword
2.4.10.13	Administration Interfaces are accessible only by authorised operators. Mutual Authentication is used over administration interfaces, for example, by using certificates.	Mandatory for Class 1 and above	System

Requirement 2.4.10.2

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword
2.4.10.2	Where the product or service provides a web browser based interface, access to any restricted/administrator area or functionality shall require authentication.	Mandatory for Class 1 and above	System

Requirement 2.4.10.5

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword
2.4.10.5	The web user interface is protected by an automatic session idle logout timeout function.	Mandatory for Class 1 and above	System

Requirement 2.4.10.16

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword
2.4.10.16	Web Interfaces should be developed using best practice secure coding techniques and server frameworks.	Mandatory for Class 1 and above	Business

Requirement 2.4.10.6

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword
2.4.10.6	User passwords are not stored in plain text.	Mandatory for all classes	System

Requirement 2.4.10.1

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword
2.4.10.1	Where the product or service provides a web based user interface, Authentication is secured using current best practice cryptography.	Mandatory for Class 1 and above	System

Requirement 2.4.10.7

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword
2.4.10.7	Where passwords are entered on a user interface, the actual pass phrase is obscured by default to prevent the capture of passwords.	Mandatory for Class 1 and above	System

Requirement 2.4.10.18

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword
2.4.10.18	Web interface should provide a simple method (one to two clicks) to initiate any security update to the end device	Mandatory for all classes	Business

Requirement 2.4.10.15

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword
2.4.10.15	All inputs and outputs are checked for validity. Tests to include both expected (valid) and unexpected (invalid) input stimuli.	Mandatory for Class 1 and above	Business

Requirement 2.4.10.3

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword
2.4.10.3	Where the product or service provides a web based management interface, Authentication is secured using current best practice cryptography.	Mandatory for Class 1 and above	System

Requirement 2.4.10.4

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword
2.4.10.4	Where a web user interface password is used for login authentication, the initial password or factory reset password is unique for every device in the product family.	Mandatory for all classes	System

Requirement 2.4.10.14

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword
2.4.10.14	Reduce the lifetime of sessions to mitigate the risk of session hijacking and replay attacks. (For example to reduce the time an attacker has to capture a session cookie and use it to access an application).	Mandatory for Class 1 and above	System

Requirement 2.4.10.10

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword
2.4.10.10	All data being transferred over interfaces should be validated where appropriate. This could include checking the data type, length, format, range, authenticity, origin and frequency.	Mandatory for Class 1 and above	System

Requirement 2.4.10.9

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword
2.4.10.9	A vulnerability assessment has been performed before deployment, and is repeated periodically throughout the lifecycle of the service or product.	Mandatory for Class 1 and above	Business

Requirement 2.4.10.17

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword
2.4.10.17	Password entry follows industry standard practice.	Mandatory for all classes	Business

Requirement 2.4.10.19

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword
2.4.10.19	Any personal data communicated between the web interface and the device shall be encrypted. Where the data includes sensitive personal data then the encryption must be appropriately secure.	Mandatory for all classes	Business

Requirement 2.4.10.12

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword
2.4.10.12	All inputs and outputs are validated using for example an allow list (formerly 'whitelist') containing authorised origins of data and valid attributes of such data.	Mandatory for Class 1 and above	System

Requirement 2.4.10.8

ReqNo	Requirement	AssuranceClassAndApplicability	PrimaryKeyword
2.4.10.8	The web user interface shall follow good practice guidelines.	Mandatory for Class 1 and above	Business

3.1 References & Standards

The following organisations, publications and/or standards have been used for the source of references in this document:

- 3GPP (3rd Generation Partnership Project)
- CSA (Cloud Security Alliance)
- DoD (US Department of Defense)
- ENISA (European Union Agency for Network and Information Security)
- ETSI (European Telecommunications Standards Institute)
- EU (European Union)
- FIPS (US Federal Information Processing Standard)
- GSMA (GSM Association)
- IETF (Internet Engineering Task Force)
- IoTSF (Internet of Things Security Foundation
- ISO (International Standard Organisation)
- JTAG (Joint Test Action Group)
- NCSC (UK National Cyber Security Centre)
- NIST (US National Institute of Standards and Technology)
- OWASP (Open Web Application Security Project)

The following references are used in this document:

1. NIST Special Publication SP800-57 Part 3 Revision 1" NIST Special Publication 800 – 57 Part 3 Revision 1 Recommendation for Key Management Part 3: Application – Specific Key Management Guidance" January 2015 http://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-57Pt3r1.pdf http://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-57Pt3r1.pdf

2. NIST Special Publication 800-131A Revision 1 "Transitions: Recommendation for Transitioning the Use of Cryptographic Algorithms and Key Lengths" November 2015

3. NIST Special Publication 800-90A Revision 1 "Recommendation for Random Number Generation Using Deterministic Random Bit Generators" June 2015 https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-90Ar1.pdf

4. Special Publication 800-22 Revision 1a "A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications" April 2010 https://tsapps.nist.gov/publication/get_pdf.cfm?pub_id=906762

5. FIPS PUB 140-2, Security Requirements for Cryptographic Modules, May 2001. http://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.140-2.pdf

6. Common Criteria for Information Technology Security Evaluation Part 1: Introduction and general model September 2012 Version 3.1 CCMB-2012-09-001 CCMB-2012-09-003 https://www.commoncriteriaportal.org/files/ccfiles/CCPART1V3.1R4_marked_changes.pdf

7. Common Criteria for Information Technology Security Evaluation Part 2: Security functional components September 2012 Version 3.1 Revision 4 CCMB-2012-09-002 https://www.commoncriteriaportal.org/files/ccfiles/CCPART2V3.1R4.pdf

8. Common Criteria for Information Technology Security Evaluation Part 3: Security assurance components September 2012 Version 3.1 Revision 4 https://www.commoncriteriaportal.org/files/ccfiles/CCPART3V3.1R4.pdf

9. Draft Framework for Cyber-Physical Systems; NIST; October 2016 https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.1500-201.pdf

10. UK Government advice on Password Guidance, Simplifying your approach, CESG and CPNI Sept 2015: https://assets.publishing.service.gov.uk/government/uploads/system/uploads/attachment_data/file/458857/Password_guidance_-_simplifying_your_approach.pdf

11. DoDI-8500.2 IA Controls: http://www.dote.osd.mil/tempguide/index.html

12. NIST Guide to Protecting the Confidentiality of Personally Identifiable Information (PII), Special Publication 800-122, NIST, April 2010: http://csrc.nist.gov/publications/nistpubs/800-122/sp800-122.pdf

13. Key definitions of the Data Protection Act, ICO: https://ico.org.uk/for-organisations/guide-to-data-protection/key-definitions

14. Overview of the General Data Protection Regulations (GDPR), ICO: https://ico.org.uk/for-organisations/data-protection-reform/overview-of-the-gdpr

15. TS-0003 Annex J (normative): List of Privacy Attributes and Clause 11 Privacy Protection Architecture using Privacy Policy Manager (PPM) https://www.onem2m.org/technicalpublished-specifications

16. Example of IoT application ID registry and possible privacy profile registry

17. https://www.onem2m.org/images/ppt/TP-2017-0200-AppID_Registry_A_Foundation_for_Trusted_Interoperability.pdf 3GPP TS33.117. Catalogue of general security assurance requirements produced by ESTI https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=2928

18. Cloud Security Alliance, Cloud Security Alliance is a not-for-profit organization promoting best practices for security assurance within Cloud Computing https://cloudsecurityalliance.org

19. IoTSF Vulnerability Disclosure Guidelines can be found https://iotsecurityfoundation.org/best-practice-guidelines

20. NIST National Institute of Standards and Technology www.nist.gov

21. NIST Cyber Security Framework https://www.nist.gov/cyberframework

22. Octave, programming language https://www.gnu.org/software/octave/

23. UK Cyber Essentials: UK government-backed, industry supported scheme to help organisations protect themselves against common cyber-attacks https://www.cyberaware.gov.uk/cyberessentials

24. UK Government Cloud Security Principles is for consumers and providers using cloud services https://www.gov.uk/government/publications/cloud-service-security-principles/cloud-service-security-principles

25. IETF – RFC2119 “Key words for use in RFCs to Indicate Requirement Levels” https://www.ietf.org/rfc/rfc2119.txt

26. NIST SP800-63b Revision 1” NIST Special Publication 800-63B Digital Identity Guidelines Authentication and Lifecycle Management” June 2017 https://pages.nist.gov/800-63-3/sp800-63b.html

27. ENISA “Algorithms, Key Sizes and Parameters Report – 2013” https://www.enisa.europa.eu/publications/algorithms-key-sizes-and-parameters-report

28. IETF RFC7525 “Recommendations for Secure Use of Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS)” https://tools.ietf.org/html/rfc7525

29. SSL Labs “SSL-and-TLS-Deployment-Best-Practices” 31 March 2017 https://github.com/ssllabs/research/wiki/SSL-and-TLS-Deployment-Best-Practices

30. OWASP “Transport Layer Protection Cheat Sheet” https://www.owasp.org/index.php/Transport_Layer_Protection_Cheat_Sheet

31. OWASP Certificate and Public Key Pinning https://www.owasp.org/index.php/Certificate_and_Public_Key_Pinning

32. NIST Special Publication 800-53, Revision 4, “Security and Privacy Controls for Federal Information Systems and Organizations” – SC-5 Denial of Service Protection https://nvd.nist.gov/800-53/Rev4/control/SC-5

33. NIST 800-53, Revision 4, “Security Controls and Assessment Procedures for Federal Information Systems and Organizations” - SI10 Information Input Validation https://nvd.nist.gov/800-53/Rev4/control/SI-10

34. NIST Special Publication 800–167 “Guide to Application Whitelisting” http://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-167.pdf

35. NIST SP 800-37 Rev. 1 “Guide for Applying the Risk Management Framework to Federal Information Systems: a Security Life Cycle Approach Risk Management Framework” https://csrc.nist.gov/publications/detail/sp/800-37/rev-1/final or Octave from ENISA

36. Operationally Critical Threat, Asset, and Vulnerability Evaluation (OCTAVE), an approach for managing information security risks. https://resources.sei.cmu.edu/library/asset-view.cfm?assetid=51546

37. Supply Chain of Trust by Hayden Povey of Secure Thingz and the IoTSF http://www.newelectronics.co.uk/article-images/152099/P18-19.pdf

38. Static Code Analysis Tools https://samate.nist.gov/index.php/Source_Code_Security_Analyzers.html

39. Bluetooth Numeric Comparison https://csrc.nist.gov/publications/detail/sp/800-121/rev-1/archive/2012-06-11 page 14

40. UK Government Cyber security risk assessment guidance https://www.ncsc.gov.uk/guidance/risk-management-collection

41. NIST Special Publication 800-30 guidance for conducting risk assessments https://www.nist.gov/publications/guide-conducting-risk-assessments

42. EU ENISA guidance of Cyber Security Risk Management https://www.enisa.europa.eu/topics/threat-risk-management/risk-management

43. Security Policy ISO/IEC Standards for Vulnerability Disclosures ISO/IEC 29147 and ISO/IEC 30111 http://standards.iso.org/ittf/PubliclyAvailableStandards/c045170_ISO_IEC_29147_2014.zip and https://www.iso.org/standard/53231.html

44. Enhanced Privacy standard for Anonymous Signatures ISO/IEC20008 https://www.iso.org/standard/57018.html

45. IoTSF Best Practice Guidelines for Connected Consumer Products V1.1 https://www.iotsecurityfoundation.org/best-practice-guidelines/#ConnectedConsumerProducts includes at time of publication individual guidelines for the following topics:

A. Classification of data

B. Physical security

C. Device secure boot

D. Secure operating system

E. Application security

F. Credential management

G. Encryption

H. Network connections

J. Securing software updates

K. Logging

L. Software update policy

46. CIA Triad has no original source, but for more info visit: https://www.techrepublic.com/blog/it-security/the-cia-triad

47. Examples of security vulnerability advisory programs: https://www.us-cert.gov/report and https://ics-cert.us-cert.gov/ICS-CERT-Vulnerability-Disclosure-Policy

48. Example of memory sanitisation:

SEI CERT C Coding Standard Recommendation MEM03-C: “Clear sensitive information stored in reusable resources” https://wiki.sei.cmu.edu/confluence/display/c/MEM03-C.+Clear+sensitive+information+stored+in+reusable+resources

ISO/IEC TR 24772:2013 “Information technology -- Programming languages -- Guidance to avoiding vulnerabilities in programming languages through language selection and use” “Sensitive Information Uncleared Before Use” https://www.iso.org/standard/61457.html

Other references:

MITRE CWE-226 “Sensitive Information Uncleared Before Release” https://cwe.mitre.org/data/definitions/226.html

CWE-244 “Improper Clearing of Heap Memory Before Release ('Heap Inspection')” https://cwe.mitre.org/data/definitions/244.html

49. NCSC password guidance https://www.ncsc.gov.uk/guidance/password-collection

50. Privacy Impact Assessment advice can be found at https://ico.org.uk/for-organisations/guide-to-the-general-data-protection-regulation-gdpr/accountability-and-governance/data-protection-impact-assessments/ and https://nvlpubs.nist.gov/nistpubs/legacy/sp/nistspecialpublication800-122.pdf

51. NCSC guidance on TLS management https://www.ncsc.gov.uk/guidance/tls-external-facing-services

52. WPA - Wi-Fi Protected Access is the name given to wireless security standard IEEE 802.11i-2004 https://standards.ieee.org/standard/802_11i-2004.html

53. The ETSI Technical Committee on Cybersecurity EN 303 645 version 2.1.1 “CYBER; Cyber Security for Consumer Internet of Things: Baseline Requirements” June 2020, , a standard for cybersecurity in the Internet of Things that establishes a security baseline for internet-connected consumer products and provides a basis for future IoT certification schemes. https://www.etsi.org/deliver/etsi_en/303600_303699/303645/02.01.01_60/en_303645v020101p.pdf

54. NIST 8259A “IoT Device Cybersecurity Capability Core Baseline” May 2020 https://nvlpubs.nist.gov/nistpubs/ir/2020/NIST.IR.8259A.pdf

3.2 Definitions And Abbreviations

For the purposes of the present document, the following abbreviations apply.

3.2.1 Definitions

Anonymity	In case of market requirements, an anonymous identity is required during ownership transfer. EU data privacy Privacy Regulations may apply.
Application	Applications (also called end-user programs) are software programs designed to perform a group of coordina tasks that may vary by installation or model. Examples of IoT applications include a web browser, sensor mar actuator controller. This contrasts with system software, which executes the operating software of the main pr device.
Authentication	Authentication is the process of recognising an identity. It is the mechanism of associating an incoming reque identifying credentials. The credentials provided are checked with those in the device or within an authenticat
Authentication	Authentication is the process of recognising an identity. It is the mechanism of associating an incoming reque identifying credentials. The credentials provided are checked with those in the device or within an authenticat
Boot	The initial process used by the device when turned on that prepares the system for operation (normally conta Boot steps).
Consumer	An end user, and not necessarily a purchaser, in the distribution chain of a good or service who make person and/or service.
Deployment	The placing of the product into customer trial or service.
Encrypted	Data secured using a recognised algorithm and protected keys, so as to be meaningful, only if decoded, and those with access to the relevant algorithm and keys.
Enterprise	An organisation in business for commercial or not-for-profit purposes that share information technology resou
Firmware	Computer programs and data stored in hardware – typically in read only memory(ROM) or programmable rea (PROM) – such that the programs and data cannot be dynamically written or modified during execution of the
IoT Product Class	Class of network products that all implement a common set of IoTSF defined functions for that particular IoT p
Interactive Account	Interactive accounts include non-personal accounts such as root, admin, service, batch, super user or privileg permit system configuration changes.
Mutual Authentication	Mutual authentication refers to a security process or technology in which two entities in a communications link and integrity of each other before any sensitive data is sent over the connection. In a network, the client authenticates the server and vice-versa. It is a default mode of authentication in some SSH (see https://tools.ietf.org/html/rfc4250) and optional in others, such as TLS (see https://tools.ietf.org/html

Nonce	Nonce is an abbreviation of the term "number used once”. It is often a random or pseudo-random number issu authentication protocol to ensure that old communications messages cannot be reused in replay attacks.
Operating System	An operating system (OS) is system software that manages device hardware and software resources and pro services for software programs.
On boarding	The method to register a device into its service or solution to enable device registration [ref 16], configuration
Ownership Transfer	In case a device is transferred through a supply chain and changes owner, this method ensures a reliable and ownership.
Personal Information	Personal Information is defined by the EU General Data Protection Regulation (GDPR): https://ec.europa.eu/i topic/data-protection_en. ‘personal data’ means any information relating to an identified or identifiable natural person (‘data subject’). A person is one who can be identified,directly or indirectly, in particular by reference to an identifier such as a na identification number, location data, an online identifier or to one or more factors specific to the physical, phys mental, economic, cultural or social identity of that natural person. Other jurisdictions may have different definitions.
Secure Boot	Process that ensures a device only starts software that is trusted by the OEM.
Secure Protocol	The method of exchanging information that ensures protection and reliability of the data (usually though crypt techniques).
Software	Unless otherwise explicitly stated, for the purposes of this document the term software also includes any firmw product.
Strong Authentication	A procedure based on the use of two or more of the following elements, categorised as knowledge, ownershi i) Something only the user or device knows, e.g. static password, code, personal identification number; ii)Something only the user or device possesses, e.g. token, smart card, mobile phone; iii) Something the user or device is, e.g. biometric characteristic, such as a fingerprint. In addition, the elements selected must be mutually independent, i.e. the breach of one does not compromise least one of the elements should be non-reusable and non-replicable (except for inherence), and not capable surreptitiously stolen via the internet. The strong authentication procedure should be designed in such a way confidentiality of the authentication data defined other examples include NIST Special Publication 800-63B se European Central Bank: Recommendations For The Security Of Internet Payments http://www.ecb.europa.eu/pub/pdf/other/recommendationssecurityinternetpaymentsoutcomeofpcfinalversiona 95e6bba1ef875877ad3c35cf3b12399c
Supply Chain of Trust	Where an IoT system uses device or service components with more than one source, all sources demonstrate relevant requirements of this framework. This will lead to the Devices and services in an IoT system exhibiting attributes: - Engender robust Root of Trust and secure identities - Safeguard application code at source Inhibit grey-manufacturing and protect IP - Ensure only valid applications are programmed - Integrate robust key structures for ownership delegation - Enable lifecycle updates and patching

Tamper Evident	The enclosure of the product has measures to ensure that any unauthorised attempt to open it leaves evidenc example, labelling across a product's enclosure joint that fragments when the joint is disturbed.
Tamper Resistant	The enclosure of the product has measures to prevent its unauthorised opening. Typically, with specialist fast features that require the use of specialist tooling, unique to the product.

3.2.2 Acronyms

CoAP	Constrained Application Protocol
DDoS	Distributed Denial of Service
DTLS	Datagram Transport Layer Security
EAL	Evaluation Assurance Level
ERP	Effective Radiated Power
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
IP	Internet Protocol
MD	Message Digest
MQTT	Message Queue Telemetry Transport - ISO standard ISO/IEC PRF 20922
OEM	Original Equipment Manufacturer
PRNG	Pseudo Random Number Generator
ROT	Root Of Trust
SHA	Secure Hash Algorithm
SSH	Secure Socket Shell
TRNG	True Random Number Generator
TBC	To Be Confirmed
TBD	To Be Determined
TCP	Transmission Control Protocol
TLS	Transport Layer Security
T3P	Trusted Third Party
UDP	User Datagram Protocol
URL	Uniform Resource Locator
WPS	Wi-Fi Protected Setup

Risk-Assessment-Steps

1 Risk Assessment Steps

The core of the security process is to understand what is being protected and from what or whom. It is also important to identify what is not being protected. There are many ways to accomplish this procedure, but it is recommended to use well-known, best practice, risk management standards [ref 39, 40 and 41]. Risk management techniques can also be found in several common business training publications. An outline of the Risk Assessment process used in this document can be seen in the flow diagram and bullet list below:

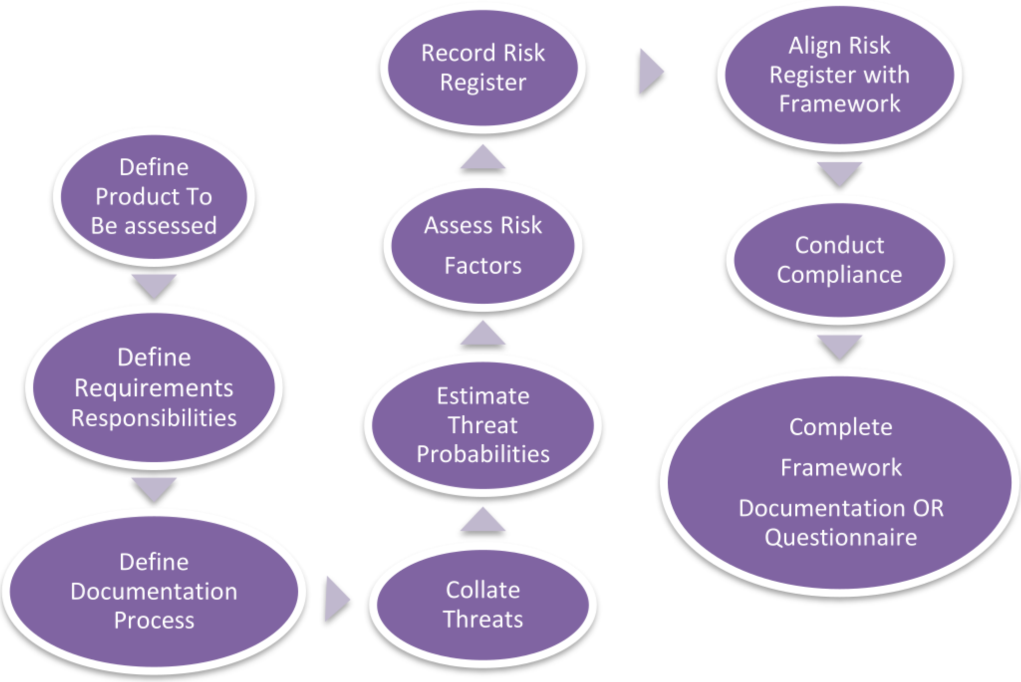

Define Product To Be assessed
Define Requirements Responsibilities
Define Documentation Process
Collate Threats
Estimate Threat Probabilities
Assess Risk Factors
Record Risk Register
Align Risk Register with Framework
Conduct Compliance
Complete Framework Documentation OR Questionnaire

Figure 3 Outline risk assessment process steps

- Create a list of security risks to the product
  - Use brainstorming techniques, mind mapping or other Group Creativity techniques.
  - Generate a list covering both business and technical threats:
    - E.g. “Brand Image damage due to adverse publicity”, “cost of product recall”, “product exposes users Wi-Fi credentials”
    - Safety aspects of the product that affect users if the security is compromised
    - The Framework can be used to support the creation of the list of risks by considering the Assurance Class criteria
- Assess the “probability” of each item on the Risk List happening
- Assess the “Cost” (impact in terms of the detectability and recovery) of each item on the Risk List – if it happens
- Multiply the Cost by the Probability, this gives a “Risk Factor”
- Order list by “Risk Factor”. This could be a percentage or simply Probability x Impact number

This list becomes the “Risk Register” document and may then be used to guide and justify the work needed to address product security. The aim of the work is to reduce the risk “probability” factor to an acceptable level.

Threat Description	Probability (0-100%)	Impact/Cost to company of threat happening (0-5)	Risk Factor
Compromise of Encryption and Key Management	5%	5	(0.05*5) = 0.25
Web User Interface subversion	90%	4	(0.9*4) =3.6

Threat Description	Probability (0-100%)	Impact/Cost to company of threat happening (0-5)	Risk Factor
Mobile Application hacked	15%	2	(0.15*2) = 0.3
Leakage of Private personal data	15%	5	(0.15*5) = 0.75

Table 5

This is showing the biggest risk is the web User Interface, so the priority should be on mitigating this to reduce the probability.

Security-Objectives-And-Requirements

2 Security Objectives And Requirements

The next step is to identify the security objectives and security non-objectives for the product. Items with high risk factors that need mitigation by design are usually considered as security objectives and items with low risk factors for which investment in mitigation is not justified are considered as non-objectives. Each objective must clearly state the asset that needs protection and relevant threats. Any excluded objectives should also be stated and explained, to make clear that they have been considered.

Security requirements are then derived from the security objectives. The main difference between those two is that security objectives specify what needs to be protected and security requirements are the means to achieve the required protection. The Security requirements document is a major milestone in the product development life cycle and should be ready before design is started.

Security-Requirements-Design-And-Implementation

3 Security Requirements Design And Implementation

The Security requirements document feeds the design and validation teams. Design methodology of security features is not different from the general design methodology of regular functional requirements. However, this is not true for validation methodology. The aim of the functional requirements validation is to verify that a system is able to do properly what it was designed to do. Security validation shall also try to simulate illegal or unexpected scenarios (e.g. writing to reserved bits in a register or applying an incorrect power up sequence) and verify that a system behaviour is predictable and security assets are not compromised.

The Risk Register should be maintained throughout the project, and the threat probabilities reassessed given the mitigations put in place to reduce the Risk Factor to an Acceptable level.

What is Acceptable? This is a qualitative assessment that needs to be made by the product owner against risk to reputation, customer expectation and cost of rectification in case of a security failure.

Appendix B Introduction To Supply Chain Security Requirements

The core of the security process is to understand what is being protected and from what or whom. It is also important to identify what is not being protected

B1-Motivation

B1 Motivation

IT systems, including IoT systems, can be compromised by cyber-attacks in their supply chain. Components compromised in the supply chain open the way for a variety of exploits when deployed into operational environments. Supply chain attacks are extremely cost effective from attackers' points of view. IT assets coming from development, manufacturing and distribution environments are often trusted implicitly by downstream users, despite weak or unknown security controls in those environments. Furthermore, a successful compromise of a single well-chosen IT vendor environment can fan out to the vendor's entire customer base as products and software updates are deployed. It is no coincidence that many of the most notorious cyber attacks have been supply chain attacks.

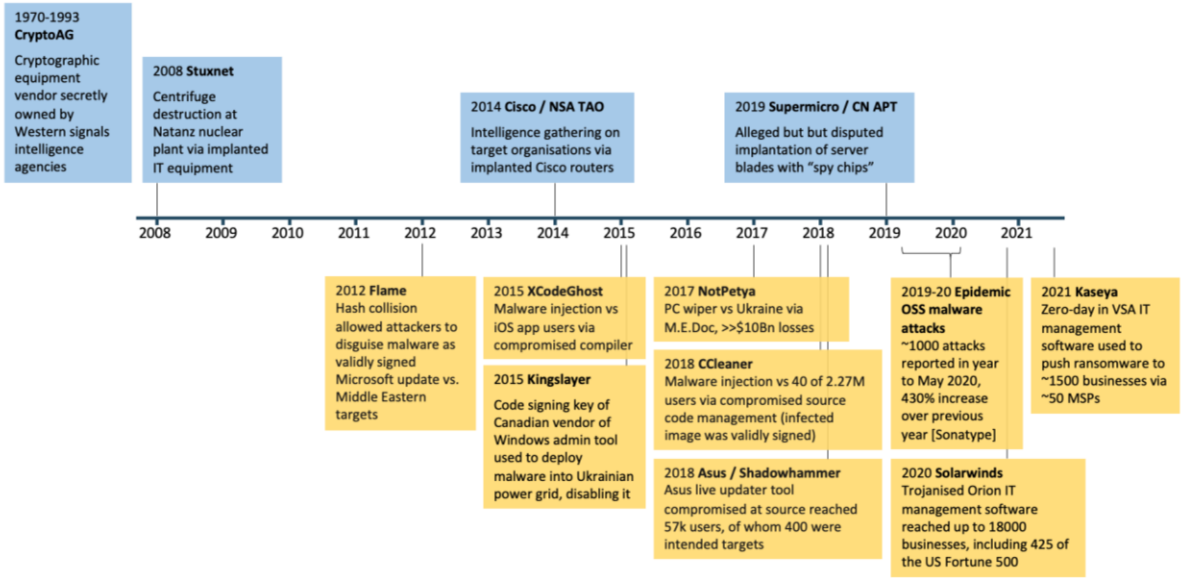

1970-1993 CryptoAG
Cryptographic equipment vendor secretly owned by Western signals intelligence agencies
2008 Stuxnet
Centrifuge destruction at Natanz nuclear plant via implanted IT equipment
2014 Cisco / NSA TAO
Intelligence gathering on target organisations via implanted Cisco routers
2019 Supermicro / CN APT
Alleged but but disputed implantation of server blades with "spy chips"
2008
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2020
2021
2012 Flame
Hash collision allowed attackers to disguise malware as validly signed Microsoft update vs. Middle Eastern targets
2015 XCodeGhost
Malware injection vs iOS app users via compromised compiler
2015 Kingslayer
Code signing key of Canadian vendor of Windows admin tool used to deploy malware into Ukrainian power grid, disabling it
2017 NotPetya
PC wiper vs Ukraine via M.E.Doc, >>$10Bn losses
2018 CCleaner
Malware injection vs 40 of 2.27M users via compromised source code management (infected image was validly signed)
2018 Asus / Shadowhammer
Asus live updater tool compromised at source reached 57k users, of whom 400 were intended targets
2019-20 Epidemic OSS malware attacks
~1000 attacks reported in year to May 2020, 430% increase over previous year [Sonatype]
2020 Solarwinds
Trojanised Orion IT management software reached up to 18000 businesses, including 425 of the US Fortune 500
2021 Kaseya
Zero-day in VSA IT management software used to push ransomware to ~1500 businesses via ~50 MSPs

Figure 4 Timeline of well-known supply chain attacks

In recent years the ICT security literature has increasingly recognised the problem of protecting both software and hardware assets in the supply chain and has developed a variety of recommendations in response. However, while many of these recommendations are applicable to IoT devices, interpreting them requires a detailed understanding of the IoT supply chain. There is also a need for IoT-specific security recommendations to accommodate IoT device supply chains' unique characteristics.

An IoTSF working group was formed in April 2020 to supply both these needs with an expanded and updated set of security requirements concerning smart devices' supply chains. The group received contributions from 43 experts representing 34 organisations resulting in 29 specific, implementable recommendations. These have been mapped into this edition of the Framework in 5 pre-existing and 24 new requirements.

B2-Definition-Of-Terms

B2 Definition Of Terms

The job of an IoT device supply chain is to deliver devices into an application in a known, trustworthy, and trusted state. As well as delivering hardware and software, an IoT device supply chain must establish trust relationships. This characteristic is not shared by ICT supply chains in general.

Each component of an IoT device is the product of a preceding design and production process. It is more accurate to think of the supply “chain” as a supply “network”. Anyone in the supply network with access to unprotected assets becomes part of the trust base of that device. Producers of key components such as embedded operating systems, cryptographic libraries and ICs carry a significant burden of trust and must demonstrate that they deserve it. But, as the designer of the production process, it is the device OEM who chooses whom to trust and is responsible for securing it overall.

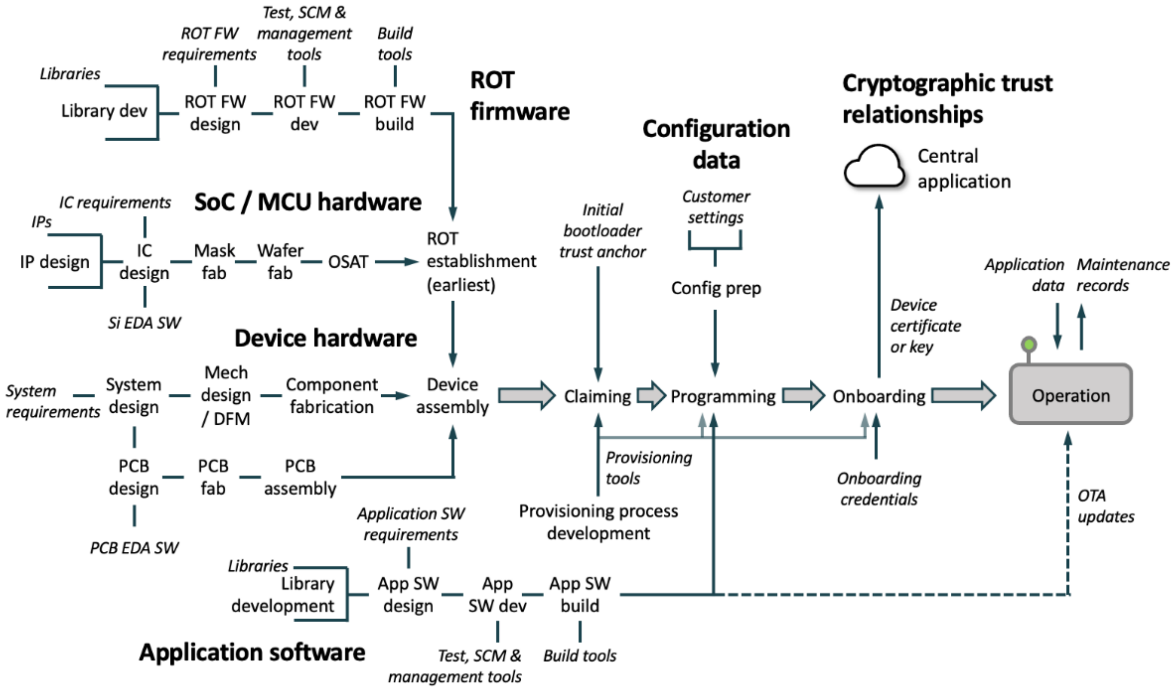


Figure 5 Overview of a typical IoT device supply network

The supply network is comprised of four basic types of operation: hardware assembly, which progressively integrates components and subassemblies into complete devices, programming, which installs logical assets onto those devices, personalisation, which generates a unique identity for each device, and on boarding, which places those devices into trust relationships with other systems. Programming, personalisation and on boarding together comprise the provisioning process, by which hardware is put into a functioning state.

While device hardware is undoubtedly important, it isn’t likely to be attacked in the supply chain. In any case by far the biggest hardware determinant of devices’ behaviour is the processor IC, the design and manufacture of which is outside of device OEMs’ control. For most OEMs the main hardware risk is the use by Contract Electronics Manufacturers (CEMs) of grey market parts, which have been known to include manufacturing discards, recycled parts and counterfeits1. Much more vulnerable to cyber-attacks are the various provisioning operations (Table 6).

Operation	Description
Programming	Programming is always performed via a programming interface exposed by the target. Programming operations place software and configuration assets onto devices. These can include assets such as: - software images and server certificates, which are the same for every device - manufacturing data and customer-specific settings, which change per batch - identity secrets and device certificates, which are individually personalised for each device.

Operation	Description
	Device operators rely on the authenticity and integrity of all these assets - and, in the case of identity secrets, also their confidentiality. Device OEMs and ODMs on their part often have an interest in maintaining the confidentiality of their software IP. Secure programming is rarely as straightforward as installing a binary image. Sometimes binaries are rebuilt per device to check for a specific IC hardware ID, as a defence against cloning. In other cases, configuration data is installed as late as possible in production, or even deferred into distribution. Device identities might be generated externally and programmed individually. Programmed assets must be protected not just in the programming environment but on the target IC. Because of this, ICs entering a secure programming environment must be authentically what they are believed to be, and they must be configured to prevent unauthorised readout or modification of assets before they leave.
RoT Establishment	With no identity or correspondent software already present, ICs fresh off the wafer typically expose a hardware-level programming interface. This channel is necessarily unencrypted and unauthenticated. The first programming step, **RoT establishment**, must therefore take place in a secure facility. RoTs once established can expose secure interfaces for provisioning subsequent assets. Examples of this pattern include secure boot managers which can detect and install new valid software images and secure programming interfaces. Both are often found as features of RoTs installed by IC vendors.
Claiming	An OEM making use of a secure boot manager established by the IC vendor must **claim** it by programming it with a trust anchor with which to validate the next software in the boot chain. Like ROT establishment, this is a special case of programming. Claiming is a key moment in the life of an IoT device because whoever installs that trust anchor chooses what software runs and thereby takes full control of the behaviour of the device.
Personalisation	Every connected device requires a unique, authenticable identity. Ideally devices should generate asymmetric identity key pairs internally, so the private key need never be exposed externally. Most modern microcontroller RoTs are able to generate high quality key pairs. Older or smaller microcontrollers may lack robust sources of high-quality entropy. Their private keys must be generated externally. Ideally this is done as close to the target device as possible to limit the potential exposure of those keys. The provisioning tool is an ideal place to accomplish this. Personalisation can also include serial numbers and other public identifiers.
Onboarding	IoT devices are useless until they are connected into larger applications. Those applications need to be told which devices to trust and how to authenticate them. There are various ways of doing this, but all involve telling the central application to trust devices which can prove possession of specified secret keys. This is called **on boarding**. The act of on boarding is a major trust decision. When a device operator makes a decision to trust an IoT device they're making a decision to trust it, and the supply chain that delivered it to them, including everyone who has had access to the device and its components. For a device with a RoT those components include **I.** The initial bootloader, on which the operator is relying to ensure only properly signed code runs, **II.** The RoT runtime services, on which they are relying to provide unimpeachable security services, and **III.** The embedded software developed by the device OEM or ODM, which the operator is expecting to behave exactly according to specification. Device operators unfortunately are not usually in a position to determine for themselves whether an IoT device has been provisioned into a known, trusted, functional initial state. Instead they must rely on

Operation	Description
	someone else’s assurances. Someone they trust, often the OEM, needs to assert “this device is in a known trusted state”. Where devices are identified using asymmetric (private and public) keys this is accomplished by on boarding the public key to central services. This can be done in several ways. The simplest method is to take a copy of each devices’ public key on the production line and upload it to the central service. The copy should be taken when the device is fully provisioned, but before it leaves the trusted manufacturing environment. A more powerful and flexible method is to sign each device’s public key into a certificate chain on the production line and load that certificate chain back into the device. The device can later deliver its public key to the central service itself, as part of a TLS handshake. Central services can on board that key on the authority of any Certificate Authority (CA) certificate in the chain. Because this allows large volumes of devices to be on boarded in a single operation it is convenient for device operators to have their devices signed into their own certificate chain of trust. In each case, whether keys are on boarded directly to the central service from the production line or signed into certificate chains of trust, it is essential that only trusted parties perform that operation. The fewer entities involved the better. Signing devices into chains of trust offers a distinct advantage over other on boarding methods in this respect, because the CA keys can be stored in an onsite HSM or secure element, or offsite in a secure facility, where they can be used without ever being exposed in manufacturing environments. It is important to note that the private keys of all the CAs in the chain of trust must be similarly protected, because an attacker gaining the use of any of them gains the ability to on board any device they choose [2].

Table 6: Provisioning operations

To reach a known functional initial state, devices must be provisioned with many software and data assets and into many trust relationships, often in a sequence of provisioning steps that begins with a blank IC and ends with a fully functional and secured device. Each step may be performed by a different actor, each provisioning the device into an intermediate state. The process may begin upstream of the OEM, with IC vendors provisioning naked dies, and it may extend to as late as immediately before devices’ live deployment, with installers commissioning devices on site.

IoT OEMs already design provisioning sequences and create or specify provisioning tools (Figure 3) for each step of those sequences, as part of their device development. Because manufacturing environments have generally been assumed secure it has been rare to give further consideration to protecting these tools and processes against deliberate attack. In essence though security is just another design goal. OEMs can use their control of this process to allocate key steps to more-trusted suppliers. Alternatively, they can engineer provisioning tools to keep assets out of harm’s way in untrusted environments.

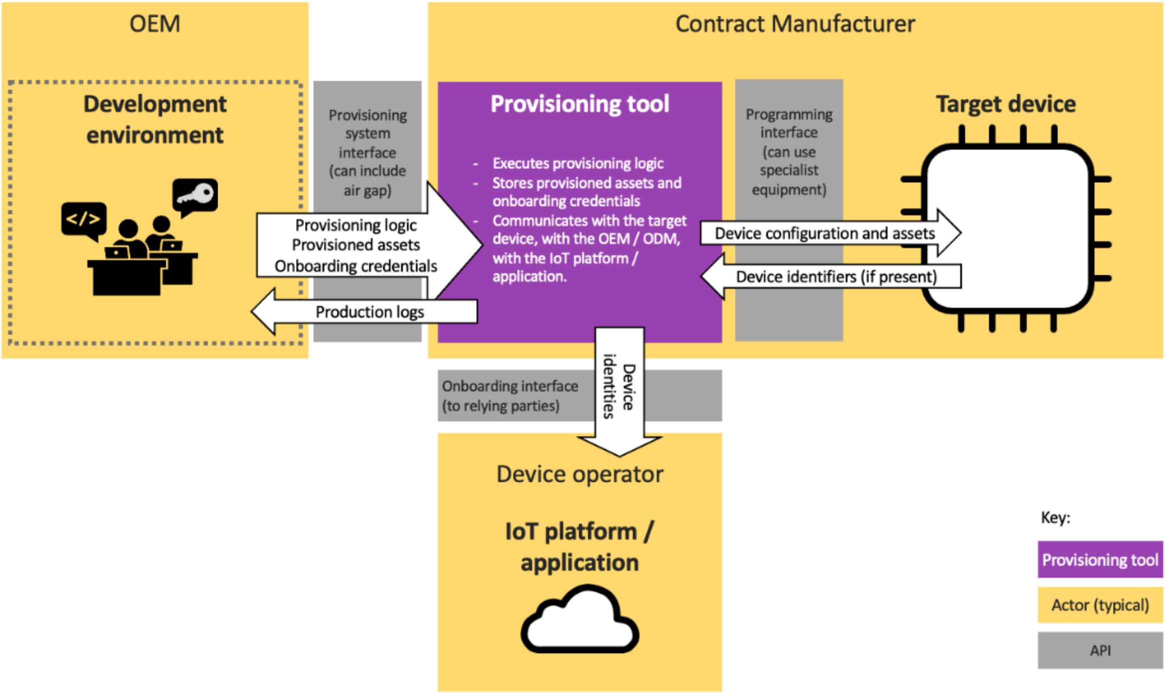

OEM
Development environment
Contract Manufacturer
Provisioning system interface (can include air gap)
Provisioning tool
- Executes provisioning logic
- Stores provisioned assets and onboarding credentials
- Communicates with the target device, with the OEM / ODM, with the IoT platform / application.
Programming interface (can use specialist equipment)
Target device
Provisioning logic
Provisioned assets
Onboarding credentials
Production logs
Device configuration and assets
Device identifiers (if present)
Onboarding interface (to relying parties)
Device identities
Device operator
IoT platform / application
Key:
Provisioning tool
Actor (typical)
API

Figure 6 Generic provisioning tool

1. 2015, Rob Wood, NCC Group, Secure Device Manufacturing: Supply Chain Security Resilience

2. 2021, Michael Richardson, IETF, A Taxonomy of operational security considerations for manufacturer installed keys and Trust Anchors

B3-Approach

B2 Approach

Submissions were invited from representatives of IoT users and vendors and categorised into lists of actors, principles, attacks, references, characteristics, assets, objectives, mitigations, and definitions. Using these inputs as an initial guide the working group developed the general characterisation of IoT device supply chains outlined above before proceeding to a threat analysis using the method of attack trees 3 . Security recommendations were developed to address these threats. In parallel, the group surveyed a range of standards and literature for known attacks and existing advice. Both were used to check the completeness of the ab initio analysis4 before the recommendations were mapped into the Framework.

This Appendix (B) was created from a white paper generated by the IoTSF Supply Chain Working Group Our thanks go to

Editor and chair

- Amyas Phillips, Ambotec Consulting

Working group members

- Amit Rao, Trusted Objects
- Anjana Priya, Microchip
- Michael Richardson, Sandelman Software Works
- Prof. Paul Dorey, CSO Confidential
- Rob Brown, Jitsuin

Contributors

- Alagar Gandhi, FCA
- Alexandru Suditu, OMV Petrom
- Andrew Frame, Secure Thingz / IAR Systems
- Angela Mison, University of South Wales

[3] 1999, Bruce Schneier, Dr Dobb’s Journal, Attack Trees (see https://www.schneier.com/academic/archives/1999/12/attack_trees.html)

[4] A full bibliography is not provided here, however special attention was given to associating actionable recommendations to the principles proposed in ENISA’s 2020 publication “Guidelines for Securing the Internet of Things: Secure Supply Chain for IoT”.

NIST SPECIAL PUBLICATION 1800-36A

Trusted Internet of Things (IoT) Device Network-Layer Onboarding and Lifecycle Management:

Enhancing Internet Protocol-Based IoT Device and Network Security

Volume A:
Executive Summary

Michael Fagan
Jeffrey Marron
Paul Watrobski
Murugiah Souppaya
National Cybersecurity Center of Excellence
Information Technology Laboratory

Blaine Mulugeta
Susan Symington
The MITRE Corporation
McLean, Virginia

Dan Harkins
Aruba, a Hewlett Packard Enterprise company
San Jose, California

William Barker
Dakota Consulting
Silver Spring, Maryland

Michael Richardson
Sandelman Software Works
Ottawa, Ontario

May 2024

DRAFT

This publication is available free of charge from
https://www.nccoe.nist.gov/projects/trusted-iot-device-network-layer-onboarding-and-lifecycle-management

Executive Summary

Establishing trust between a network and an Internet of Things (IoT) device (as defined in NIST Internal Report 8425) prior to providing the device with the credentials it needs to join the network is crucial for mitigating the risk of potential attacks. There are two possibilities for attack. One happens when a device is convinced to join an unauthorized network, which would take control of the device. The other occurs when a network is infiltrated by a malicious device. Trust is achieved by attesting and verifying the identity and posture of the device and the network before providing the device with its network credentials—a process known as *network-layer onboarding*. In addition, scalable, automated mechanisms are needed to safely manage IoT devices throughout their lifecycles, such as safeguards that verify the security posture of a device before the device is permitted to execute certain operations. In this practice guide, the National Cybersecurity Center of Excellence (NCCoE) applies standards, best practices, and commercially available technology to demonstrate various mechanisms for trusted network-layer onboarding of IoT devices in Internet Protocol based environments. This guide shows how to provide network credentials to IoT devices in a trusted manner and maintain a secure device posture throughout the device lifecycle, thereby enhancing IoT security in alignment with the IoT Cybersecurity Improvement Act of 2020.

CHALLENGE

With 40 billion IoT devices expected to be connected worldwide by 2025, it is unrealistic to onboard or manage these devices by manually interacting with each device. In addition, providing local network credentials at the time of manufacture requires the manufacturer to customize network-layer onboarding on a build-to-order basis, which prevents the manufacturer from taking full advantage of the economies of scale that could result from building identical devices for its customers.

There is a need to have a scalable, automated mechanism to securely manage IoT devices throughout their lifecycles and, in particular, a trusted mechanism for providing IoT devices with their network credentials and access policy at the time of deployment on the network. It is easy for a network to falsely identify itself, yet many IoT devices onboard to networks without verifying the network's identity and ensuring that it is their intended target network. Also, many IoT devices lack user interfaces, making it cumbersome to manually input network credentials. Wi-Fi is sometimes used to provide credentials over an open (i.e., unencrypted) network, but this onboarding method risks credential disclosure. Most home networks use a single password shared among all devices, so access is controlled only by the device's possession of the password and does not consider a unique device identity or whether the device belongs on the network. This method also increases the risk of exposing credentials to unauthorized parties. Providing unique credentials to each device is more secure, but providing unique credentials manually would be resource-intensive and error-prone, would risk credential disclosure, and cannot be performed at scale.

Once a device is connected to the network, if it becomes compromised, it can pose a security risk to both the network and other connected devices. Not keeping such a device current with the most recent software and firmware updates may make it more susceptible to compromise. The device could also be attacked through receipt of malicious payloads. Once compromised, it may be used to attack other devices on the network.

OUTCOME

The outcome of this project is development of example trusted onboarding solutions, demonstration that they support various scenarios, and publication of the findings in this practice guide, a NIST Special Publication (SP) 1800 that is composed of multiple volumes targeting different audiences.

This practice guide can help IoT device users:

Understand how to onboard their IoT devices in a trusted manner to:

- **Ensure that their network is not put at risk** as new IoT devices are added to it
- **Safeguard their IoT devices** from being taken over by unauthorized networks
- **Provide IoT devices with unique credentials** for network access
- **Provide, renew, and replace device network credentials** in a secure manner
- **Support ongoing protection of IoT devices** throughout their lifecycles

This practice guide can help manufacturers and vendors of semiconductors, secure storage components, IoT devices, and network onboarding equipment:

Understand the desired security properties for supporting trusted network-layer onboarding and explore their options with respect to recommended practices for:

- **Providing unique credentials into secure storage on IoT devices at the time of manufacture to mitigate supply chain risks** (i.e., *device credentials*)
- **Installing onboarding software onto IoT devices**
- **Providing IoT device purchasers with information needed to onboard the IoT devices to their networks** (i.e., *device bootstrapping information*)
- **Integrating support for network-layer onboarding with additional security capabilities** to provide ongoing protection throughout the device lifecycle

SOLUTION

The NCCoE recommends the use of trusted network-layer onboarding to provide scalable, automated, trusted ways to provide IoT devices with unique network credentials and manage devices throughout their lifecycles to ensure that they remain secure. The NCCoE is collaborating with technology providers and other stakeholders to implement example trusted network-layer onboarding solutions for IoT devices that:

- provide each device with unique network credentials,
- enable the device and the network to mutually authenticate,
- send devices their credentials over an encrypted channel,
- do not provide any person with access to the credentials, and

- can be performed repeatedly throughout the device lifecycle.

The capabilities demonstrated include:

- trusted network-layer onboarding of IoT devices,
- repeated trusted network-layer onboarding of devices to the same or a different network,
- trusted application-layer onboarding (i.e., automatic establishment of an encrypted connection between an IoT device and a trusted application service after the IoT device has performed trusted network-layer onboarding and used its credentials to connect to the network), and
- software-based methods to provide device credentials in the factory and transfer device bootstrapping information from device manufacturer to device purchaser.

Future capabilities may include demonstrating the integration of trusted network-layer onboarding with zero trust-inspired [Note: See NIST SP 800-207] mechanisms such as ongoing device authorization, renewal of device network credentials, device attestation to ensure that only trusted IoT devices are permitted to be onboarded, device lifecycle management, and enforcement of device communications intent.

This demonstration follows an agile methodology of building implementations (i.e., *builds*) iteratively and incrementally, starting with network-layer onboarding and gradually integrating additional capabilities that improve device and network security throughout a managed device lifecycle. This includes factory builds that simulate activities performed to securely provide device credentials during the manufacturing process, and five network-layer onboarding builds that demonstrate the Wi-Fi Easy Connect, Bootstrapping Remote Secure Key Infrastructure (BRSKI), and Thread Commissioning protocols. These builds also demonstrate both streamlined and independent trusted application-layer onboarding approaches, along with policy-based continuous assurance and authorization. The example implementations use technologies and capabilities from our project collaborators (listed below).

Collaborators

Aruba, a Hewlett Packard Enterprise company
CableLabs
Cisco
Foundries.io
Kudelski IoT
NquiringMinds
NXP Semiconductors
Open Connectivity Foundation (OCF)
Sandelman Software Works
SEALSQ, a subsidiary of WISeKey
Silicon Labs

While the NCCoE uses a suite of commercial products, services, and proof-of-concept technologies to address this challenge, this guide does not endorse these particular products, services, and technologies, nor does it guarantee compliance with any regulatory initiatives. Your organization's information security experts should identify the products and services that will best integrate with your existing tools, IT and IoT system infrastructure, and operations. Your organization can adopt these solutions or one that adheres to these guidelines in whole, or you can use this guide as a starting point for tailoring and implementing parts of a solution.

HOW TO USE THIS GUIDE

Depending on your role in your organization, you might use this guide in different ways:

Business decision makers, such as chief information security, product security, and technology officers, can use this part of the guide, *NIST SP 1800-36A: Executive Summary*, to understand the project's challenges and outcomes, as well as our solution approach.

Technology, security, and privacy program managers who are concerned with how to identify, understand, assess, and mitigate risk can use *NIST SP 1800-36B: Approach, Architecture, and Security Characteristics*. This part of the guide describes the architecture and different implementations. Also, *NIST SP 1800-36E: Risk and Compliance Management,* maps components of the trusted onboarding reference architecture to security characteristics in broadly applicable, well-known cybersecurity guidelines and practices.

IT professionals who want to implement an approach like this can make use of *NIST SP 1800-36C: How-To Guides*. It provides product installation, configuration, and integration instructions for building example implementations, allowing them to be replicated in whole or in part. They can also use *NIST SP 1800-36D*: *Functional Demonstrations,* which provides the use cases that have been defined to showcase trusted network-layer onboarding and lifecycle management security capabilities and the results of demonstrating these capabilities with each of the example implementations. These use cases may be helpful when developing requirements for systems being developed.

SHARE YOUR FEEDBACK

You can view or download the preliminary draft guide at https://www.nccoe.nist.gov/projects/building-blocks/iot-network-layer-onboarding. NIST is adopting an agile process to publish this content. Each volume is being made available as soon as possible rather than delaying release until all volumes are completed.

Help the NCCoE make this guide better by sharing your thoughts with us as you read the guide. As example implementations continue to be developed, you can adopt this solution for your own organization. If you do, please share your experience and advice with us. We recognize that technical solutions alone will not fully enable the benefits of our solution, so we encourage organizations to share lessons learned and recommended practices for transforming the processes associated with implementing this guide.

To provide comments, join the community of interest, or learn more by arranging a demonstration of these example implementations, contact the NCCoE at iot-onboarding@nist.gov.

COLLABORATORS

Collaborators participating in this project submitted their capabilities in response to an open call in the Federal Register for all sources of relevant security capabilities from academia and industry (vendors and integrators). Those respondents with relevant capabilities or product components signed a Cooperative Research and Development Agreement (CRADA) to collaborate with NIST in a consortium to build this example solution.

Certain commercial entities, equipment, products, or materials may be identified by name or company logo or other insignia in order to acknowledge their participation in this collaboration or to describe an experimental procedure or concept adequately. Such identification is not intended to imply special status or relationship with NIST or recommendation or endorsement by NIST or the NCCoE; neither is it intended to imply that the entities, equipment, products, or materials are necessarily the best available for the purpose.

NIST SPECIAL PUBLICATION 1800-36B

Trusted Internet of Things (IoT) Device Network-Layer Onboarding and Lifecycle Management:

Enhancing Internet Protocol-Based IoT Device and Network Security

Volume B:
Approach, Architecture, and Security Characteristics

Michael Fagan
Jeffrey Marron
Paul Watrobski
Murugiah Souppaya
National Cybersecurity Center of Excellence
Information Technology Laboratory

William Barker
Dakota Consulting
Silver Spring, Maryland

Chelsea Deane
Joshua Klosterman
Charlie Rearick
Blaine Mulugeta
Susan Symington
The MITRE Corporation
McLean, Virginia

Dan Harkins
Danny Jump
Aruba, a Hewlett Packard Enterprise Company
San Jose, California

Andy Dolan
Kyle Haefner
Craig Pratt
Darshak Thakore
CableLabs
Louisville, Colorado

Peter Romness
Cisco
San Jose, California

Tyler Baker
David Griego
Foundries.io
London, United Kingdom

Brecht Wyseur
Kudelski IoT
Cheseaux-sur-Lausanne,
Switzerland

Alexandru Mereacre
Nick Allott
Ashley Setter
NquiringMinds
Southampton, United Kingdom

Julien Delplancke
NXP Semiconductors
Mougins, France

Michael Richardson
Sandelman Software Works
Ontario, Canada

Steve Clark
SEALSQ, a subsidiary of WISeKey
Geneva, Switzerland

Mike Dow
Steve Egerter
Silicon Labs
Austin, Texas

May 2024

DRAFT

This publication is available free of charge from
https://www.nccoe.nist.gov/projects/trusted-iot-device-network-layer-onboarding-and-lifecycle-management

DISCLAIMER

Certain commercial entities, equipment, products, or materials may be identified by name or company logo or other insignia in order to acknowledge their participation in this collaboration or to describe an experimental procedure or concept adequately. Such identification is not intended to imply special status or relationship with NIST or recommendation or endorsement by NIST or NCCoE; neither is it intended to imply that the entities, equipment, products, or materials are necessarily the best available for the purpose.

National Institute of Standards and Technology Special Publication 1800-36B, Natl. Inst. Stand. Technol. Spec. Publ. 1800-36B, 114 pages, May 2024, CODEN: NSPUE2

FEEDBACK

You can improve this guide by contributing feedback regarding which aspects of it you find helpful as well as suggestions on how it might be improved. Should we provide guidance summaries that target specific audiences? What trusted IoT device onboarding protocols and related features are most important to you? Is there some content that is not included in this document that we should cover? Are we missing anything in terms of technologies or use cases? In what areas would it be most helpful for us to focus our future related efforts? For example, should we consider implementing builds that onboard devices supporting Matter and/or the Fast Identity Online (FIDO) Alliance application onboarding protocol? Should we implement builds that integrate security mechanisms such as lifecycle management, supply chain management, attestation, or behavioral analysis? As you review and adopt this solution for your own organization, we ask you and your colleagues to share your experience and advice with us.

Comments on this publication may be submitted to: iot-onboarding@nist.gov.

Public comment period: May 31, 2024 through July 30, 2024

All comments are subject to release under the Freedom of Information Act.

National Cybersecurity Center of Excellence
National Institute of Standards and Technology
100 Bureau Drive
Mailstop 2002
Gaithersburg, MD 20899
Email: nccoe@nist.gov

NATIONAL CYBERSECURITY CENTER OF EXCELLENCE

The National Cybersecurity Center of Excellence (NCCoE), a part of the National Institute of Standards and Technology (NIST), is a collaborative hub where industry organizations, government agencies, and academic institutions work together to address businesses' most pressing cybersecurity issues. This public-private partnership enables the creation of practical cybersecurity solutions for specific industries, as well as for broad, cross-sector technology challenges. Through consortia under Cooperative Research and Development Agreements (CRADAs), including technology partners—from Fortune 50 market leaders to smaller companies specializing in information technology security—the NCCoE applies standards and best practices to develop modular, adaptable example cybersecurity solutions using commercially available technology. The NCCoE documents these example solutions in the NIST Special Publication 1800 series, which maps capabilities to the NIST Cybersecurity Framework and details the steps needed for another entity to re-create the example solution. The NCCoE was established in 2012 by NIST in partnership with the State of Maryland and Montgomery County, Maryland.

To learn more about the NCCoE, visit https://www.nccoe.nist.gov/. To learn more about NIST, visit https://www.nist.gov.

NIST CYBERSECURITY PRACTICE GUIDES

NIST Cybersecurity Practice Guides (Special Publication 1800 series) target specific cybersecurity challenges in the public and private sectors. They are practical, user-friendly guides that facilitate the adoption of standards-based approaches to cybersecurity. They show members of the information security community how to implement example solutions that help them align with relevant standards and best practices, and provide users with the materials lists, configuration files, and other information they need to implement a similar approach.

The documents in this series describe example implementations of cybersecurity practices that businesses and other organizations may voluntarily adopt. These documents do not describe regulations or mandatory practices, nor do they carry statutory authority.

KEYWORDS

application-layer onboarding; bootstrapping; Internet of Things (IoT); Manufacturer Usage Description (MUD); network-layer onboarding; onboarding; Wi-Fi Easy Connect.

ACKNOWLEDGMENTS

We are grateful to the following individuals for their generous contributions of expertise and time.

Name	Organization
Amogh Guruprasad Deshmukh	Aruba, a Hewlett Packard Enterprise company
Bart Brinkman	Cisco

Name	Organization
Eliot Lear	Cisco
George Grey	Foundries.io
David Griego	Foundries.io
Fabien Gremaud	Kudelski IoT
Faith Ryan	The MITRE Corporation
Toby Ealden	NquiringMinds
John Manslow	NquiringMinds
Antony McCaigue	NquiringMinds
Alexandru Mereacre	NquiringMinds
Loic Cavaille	NXP Semiconductors
Mihai Chelalau	NXP Semiconductors
Julien Delplancke	NXP Semiconductors
Anda-Alexandra Dorneanu	NXP Semiconductors
Todd Nuzum	NXP Semiconductors
Nicusor Penisoara	NXP Semiconductors
Laurentiu Tudor	NXP Semiconductors
Karen Scarfone	Scarfone Cybersecurity
Pedro Fuentes	SEALSQ, a subsidiary of WISeKey
Gweltas Radenac	SEALSQ, a subsidiary of WISeKey
Kalvin Yang	SEALSQ, a subsidiary of WISeKey

The Technology Partners/Collaborators who participated in this build submitted their capabilities in response to a notice in the Federal Register. Respondents with relevant capabilities or product components were invited to sign a Cooperative Research and Development Agreement (CRADA) with NIST, allowing them to participate in a consortium to build this example solution. We worked with:

Technology Collaborators

Aruba, a Hewlett Packard Enterprise company
CableLabs
Cisco
Foundries.io
Kudelski IoT
NquiringMinds
NXP Semiconductors
Open Connectivity Foundation (OCF)
Sandelman Software Works
SEALSQ, a subsidiary of WISeKey
Silicon Labs

DOCUMENT CONVENTIONS

The terms “shall” and “shall not” indicate requirements to be followed strictly to conform to the publication and from which no deviation is permitted. The terms “should” and “should not” indicate that among several possibilities, one is recommended as particularly suitable without mentioning or excluding others, or that a certain course of action is preferred but not necessarily required, or that (in the negative form) a certain possibility or course of action is discouraged but not prohibited. The terms “may” and “need not” indicate a course of action permissible within the limits of the publication. The terms “can” and “cannot” indicate a possibility and capability, whether material, physical, or causal.

CALL FOR PATENT CLAIMS

This public review includes a call for information on essential patent claims (claims whose use would be required for compliance with the guidance or requirements in this Information Technology Laboratory (ITL) draft publication). Such guidance and/or requirements may be directly stated in this ITL Publication or by reference to another publication. This call also includes disclosure, where known, of the existence of pending U.S. or foreign patent applications relating to this ITL draft publication and of any relevant unexpired U.S. or foreign patents.

ITL may require from the patent holder, or a party authorized to make assurances on its behalf, in written or electronic form, either:

a) assurance in the form of a general disclaimer to the effect that such party does not hold and does not currently intend holding any essential patent claim(s); or

b) assurance that a license to such essential patent claim(s) will be made available to applicants desiring to utilize the license for the purpose of complying with the guidance or requirements in this ITL draft publication either:

1. under reasonable terms and conditions that are demonstrably free of any unfair discrimination; or
2. without compensation and under reasonable terms and conditions that are demonstrably free of any unfair discrimination.

Such assurance shall indicate that the patent holder (or third party authorized to make assurances on its behalf) will include in any documents transferring ownership of patents subject to the assurance, provisions sufficient to ensure that the commitments in the assurance are binding on the transferee, and that the transferee will similarly include appropriate provisions in the event of future transfers with the goal of binding each successor-in-interest.

The assurance shall also indicate that it is intended to be binding on successors-in-interest regardless of whether such provisions are included in the relevant transfer documents.

Such statements should be addressed to: iot-onboarding@nist.gov.

Contents

1 Summary 1

1.1 Challenge 1

1.2 Solution..2

1.3 Benefits..3

2 How to Use This Guide 3

2.1 Typographic Conventions 5

3 Approach 5

3.1 Audience 7

3.2 Scope 8

3.3 Assumptions and Definitions..8

3.3.1 Credential Types 8

3.3.2 Integrating Security Enhancements 10

3.3.3 Device Limitations 12

3.3.4 Specifications Are Still Improving 12

3.4 Collaborators and Their Contributions..12

3.4.1 Aruba, a Hewlett Packard Enterprise Company 14

3.4.2 CableLabs.. 16

3.4.3 Cisco.. 17

3.4.4 Foundries.io 17

3.4.5 Kudelski IoT.. 18

3.4.6 NquiringMinds 18

3.4.7 NXP Semiconductors 20

3.4.8 Open Connectivity Foundation (OCF).. 21

3.4.9 Sandelman Software Works 21

3.4.10 SEALSQ, a subsidiary of WISeKey 22

3.4.11 VaultIC408 23

3.4.12 Silicon Labs.. 23

4 Reference Architecture 25

4.1 Device Manufacture and Factory Provisioning Process 26

4.2 Device Ownership and Bootstrapping Information Transfer Process 28

4.3 Trusted Network-Layer Onboarding Process 31

4.4 Trusted Application-Layer Onboarding Process 32
4.5 Continuous Verification 35
5 Laboratory Physical Architecture 37
5.1 Shared Environment 40
5.1.1 Domain Controller 40
5.1.2 Jumpbox 40
5.2 Build 1 (Wi-Fi Easy Connect, Aruba/HPE) Physical Architecture 41
5.2.1 Wi-Fi Easy Connect Factory Provisioning Build Physical Architecture 42
5.3 Build 2 (Wi-Fi Easy Connect, CableLabs, OCF) Physical Architecture 43
5.4 Build 3 (BRSKI, Sandelman Software Works) Physical Architecture 44
5.5 Build 4 (Thread, Silicon Labs, Kudelski IoT) Physical Architecture 46
5.6 Build 5 (BRSKI, NquiringMinds) Physical Architecture 47
5.6.1 BRSKI Factory Provisioning Build Physical Architecture 48
6 General Findings 49
6.1 Wi-Fi Easy Connect 49
6.1.1 Mutual Authentication 50
6.1.2 Mutual Authorization 50
6.1.3 Secure Storage 50
6.2 BRSKI 50
6.2.1 Reliance on the Device Manufacturer 51
6.2.2 Mutual Authentication 51
6.2.3 Mutual Authorization 51
6.2.4 Secure Storage 51
6.3 Thread 51
6.4 Application-Layer Onboarding 52
6.4.1 Independent Application-Layer Onboarding 52
6.4.2 Streamline Application-Layer Onboarding 52
7 Additional Build Considerations 53
7.1 Network Authentication 53
7.2 Device Communications Intent 53
7.3 Network Segmentation 54
7.4 Integration with a Lifecycle Management Service 54
7.5 Network Credential Renewal 54

7.6 Integration with Supply Chain Management Tools 54

7.7 Attestation 54

7.8 Mutual Attestation 54

7.9 Behavioral Analysis 55

7.10 Device Trustworthiness Scale 55

7.11 Resource Constrained Systems 55

Appendix A List of Acronyms 56

Appendix B Glossary 59

Appendix C Build 1 (Wi-Fi Easy Connect, Aruba/HPE) 60

C.1 Technologies 60

C.2 Build 1 Architecture 62

C.2.1 Build 1 Logical Architecture 62

C.2.2 Build 1 Physical Architecture 64

Appendix D Build 2 (Wi-Fi Easy Connect, CableLabs, OCF) 65

D.1 Technologies 65

D.2 Build 2 Architecture 67

D.2.1 Build 2 Logical Architecture 67

D.2.2 Build 2 Physical Architecture 70

Appendix E Build 3 (BRSKI, Sandelman Software Works) 71

E.1 Technologies 71

E.2 Build 3 Architecture 73

E.2.1 Build 3 Logical Architecture 73

E.2.2 Build 3 Physical Architecture 75

Appendix F Build 4 (Thread, Silicon Labs-Thread, Kudelski KeySTREAM) 76

F.1 Technologies 76

F.2 Build 4 Architecture 78

F.2.1 Build 4 Logical Architecture 78

F.2.2 Build 4 Physical Architecture 83

Appendix G Build 5 (BRSKI over Wi-Fi, NquiringMinds) 84

G.1 Technologies 84

G.2 Build 5 Architecture 86

G.2.1 Build 5 Logical Architecture 86
G.2.2 Build 5 Physical Architecture 89
Appendix H Factory Provisioning Process 90
H.1 Factory Provisioning Process 90
H.1.1 Device Birth Credential Provisioning Methods 90
H.2 Factory Provisioning Builds – General Provisioning Process 92
H.3 BRSKI Factory Provisioning Builds (NquiringMinds and SEALSQ) 93
H.3.1 BRSKI Factory Provisioning Build Technologies 93
H.3.2 BRSKI Factory Provisioning Build Logical Architectures 95
H.3.3 BRSKI Factory Provisioning Build Physical Architectures 98
H.4 Wi-Fi Easy Connect Factory Provisioning Build (SEALSQ and Aruba/HPE) 98
H.4.1 Wi-Fi Easy Connect Factory Provisioning Build Technologies 98
H.4.2 Wi-Fi Easy Connect Factory Provisioning Build Logical Architecture 99
H.4.3 Wi-Fi Easy Connect Factory Provisioning Build Physical Architecture 100
Appendix I References 101

List of Figures

Figure 3-1 Aruba/HPE DPP Onboarding Components 16
Figure 3-2 Components for Onboarding an IoT Device that Communicates Using Thread to AWS IoT 24
Figure 4-1 Trusted IoT Device Network-Layer Onboarding and Lifecycle Management Logical Reference Architecture 25
Figure 4-2 IoT Device Manufacture and Factory Provisioning Process 27
Figure 4-3 Device Ownership and Bootstrapping Information Transfer Process 29
Figure 4-4 Trusted Network-Layer Onboarding Process 31
Figure 4-5 Trusted Streamlined Application-Layer Onboarding Process 33
Figure 4-6 Continuous Verification 35
Figure 5-1 NCCoE IoT Onboarding Laboratory Physical Architecture 38
Figure 5-2 Physical Architecture of Build 1 42
Figure 5-3 Physical Architecture of Wi-Fi Easy Connect Factory Provisioning Build 43
Figure 5-4 Physical Architecture of Build 2 44
Figure 5-5 Physical Architecture of Build 3 45
Figure 5-6 Physical Architecture of Build 4 47

Figure 5-7 Physical Architecture of Build 5 48

Figure 5-8 Physical Architecture of BRSKI Factory Provisioning Build....................49

Figure C-1 Logical Architecture of Build 1 63

Figure D-1 Logical Architecture of Build 2....................68

Figure E-1 Logical Architecture of Build 3 73

Figure F-1 Logical Architecture of Build 4: Device Preparation 80

Figure F-2 Logical Architecture of Build 4: Connection to the OpenThread Network 81

Figure F-3 Logical Architecture of Build 4: Application-Layer Onboarding using the Kudelski keySTREAM Service....................82

Figure G-1 Logical Architecture of Build 5....................87

Figure H-1 Logical Architecture of the First Version of the BRSKI Factory Provisioning Build 97

Figure H-2 Logical Architecture of the Second Version of the BRSKI Factory Provisioning Build....................98

Figure H-3 Logical Architecture of the Wi-Fi Easy Connect Factory Provisioning Build....................100

List of Tables

Table 3-1 Capabilities and Components Provided by Each Technology Partner/Collaborator 13

Table 5-1 Build 1 Products and Technologies....................40

Table C-1 Build 1 Products and Technologies....................60

Table E-1 Build 3 Products and Technologies....................71

Table F-1 Build 4 Products and Technologies....................76

Table G-1 Build 5 Products and Technologies 84

1 Summary

IoT devices are typically connected to a network. As with any other device needing to communicate on a network securely, an IoT device needs credentials that are specific to that network to help ensure that only authorized devices can connect to and use the network. A typical commercially available, mass-produced IoT device cannot be pre-provisioned with local network credentials by the manufacturer during the manufacturing process. Instead, the local network credentials will be provisioned to the device at the time of its deployment. This practice guide is focused on trusted methods of providing IoT devices with the network-layer credentials and policy they need to join a network upon deployment, a process known as *network-layer onboarding.*

Establishing trust between a network and an IoT device (as defined in NIST Internal Report 8425) prior to providing the device with the credentials it needs to join the network is crucial for mitigating the risk of potential attacks. There are two possibilities for attack. One is where a device is convinced to join an unauthorized network, which would take control of the device. The other is where a network is infiltrated by a malicious device. Trust is achieved by attesting and verifying the identity and posture of the device and the network before providing the device with its network credentials—a process known as *network-layer onboarding*. In addition, scalable, automated mechanisms are needed to safely manage IoT devices throughout their lifecycles, such as safeguards that verify the security posture of a device before the device is permitted to execute certain operations.

In this practice guide, the National Cybersecurity Center of Excellence (NCCoE) applies standards, best practices, and commercially available technology to demonstrate various mechanisms for trusted network-layer onboarding of IoT devices. This guide shows how to provide network credentials to IoT devices in a trusted manner and maintain a secure device posture throughout the device lifecycle.

1.1 Challenge

With 40 billion IoT devices expected to be connected worldwide by 2025 [1], it is unrealistic to onboard or manage these devices by visiting each device and performing a manual action. While it is possible for devices to be securely provided with their local network credentials at the time of manufacture, this requires the manufacturer to customize network-layer onboarding on a build-to-order basis, which prevents the manufacturer from taking full advantage of the economies of scale that could result from building identical devices for all its customers.

The industry lacks scalable, automatic mechanisms to safely manage IoT devices throughout their lifecycles and lacks a trusted mechanism for providing IoT devices with their network credentials and policy at the time of deployment on the network. It is easy for a network to falsely identify itself, yet many IoT devices onboard to networks without verifying the network’s identity and ensuring that it is their intended target network. Also, many IoT devices lack user interfaces, making it cumbersome to manually input network credentials. Wi-Fi is sometimes used to provide credentials over an open (i.e., unencrypted) network, but this onboarding method risks credential disclosure. Most home networks use a single password shared among all devices, so access is controlled only by the device’s possession of the password and does not consider a unique device identity or whether the device belongs on the network. This method also increases the risk of exposing credentials to unauthorized parties. Providing unique credentials to each device is more secure, but doing so manually would be resource-intensive and error-prone, would risk credential disclosure, and cannot be performed at scale.

Once a device is connected to the network, if it becomes compromised, it can pose a security risk to both the network and other connected devices. Not keeping such a device current with the most recent software and firmware updates may make it more susceptible to compromise. The device could also be attacked through the receipt of malicious payloads. Once compromised, it may be used to attack other devices on the network.

1.2 Solution

We need scalable, automated, trusted mechanisms to safely manage IoT devices throughout their lifecycles to ensure that they remain secure, starting with secure ways to provision devices with their network credentials, i.e., beginning with network-layer onboarding. Onboarding is a particularly vulnerable point in the device lifecycle because if it is not performed in a secure manner, then both the device and the network are at risk. Networks are at risk of having unauthorized devices connect to them, and devices are at risk of being taken over by networks that are not authorized to onboard or control them.

The NCCoE has adopted the trusted network-layer onboarding approach to promote automated, trusted ways to provide IoT devices with unique network credentials and manage devices throughout their lifecycles to ensure that they remain secure. The NCCoE is collaborating with CRADA consortium technology providers in a phased approach to develop example implementations of trusted network-layer onboarding solutions. We define a *trusted network-layer onboarding solution* to be a mechanism for provisioning network credentials to a device that:

- provides each device with unique network credentials,
- enables the device and the network to mutually authenticate,
- sends devices their network credentials over an encrypted channel,
- does not provide any person with access to the network credentials, and
- can be performed repeatedly throughout the device lifecycle to enable:
  - the device's network credentials to be securely managed and replaced as needed, and
  - the device to be securely onboarded to other networks after being repurposed or resold.

The use cases designed to be demonstrated by this project's implementations include:

- trusted network-layer onboarding of IoT devices
- repeated trusted network-layer onboarding of devices to the same or a different network
- automatic establishment of an encrypted connection between an IoT device and a trusted application service (i.e., *trusted application-layer onboarding*) after the IoT device has performed trusted network-layer onboarding and used its credentials to connect to the network
- policy-based ongoing device authorization
- software-based methods to provision device birth credentials in the factory

- mechanisms for IoT device manufacturers to provide IoT device purchasers with information needed to onboard the IoT devices to their networks (i.e., *device bootstrapping information*)

1.3 Benefits

This practice guide can benefit both IoT device users and IoT device manufacturers. The guide can help IoT device users understand how to onboard IoT devices to their networks in a trusted manner to:

- Ensure that their network is not put at risk as IoT devices are added to it
- Safeguard their IoT devices from being taken over by unauthorized networks
- Provide IoT devices with unique credentials for network access
- Provide, renew, and replace device network credentials in a secure manner
- Ensure that IoT devices can automatically and securely perform application-layer onboarding after performing trusted network-layer onboarding and connecting to a network
- Support ongoing protection of IoT devices throughout their lifecycles

This guide can help IoT device manufacturers, as well as manufacturers and vendors of semiconductors, secure storage components, and network onboarding equipment, understand the desired security properties for supporting trusted network-layer onboarding and demonstrate mechanisms for:

- Placing unique credentials into secure storage on IoT devices at time of manufacture (i.e., *device birth credentials*)
- Installing onboarding software onto IoT devices
- Providing IoT device purchasers with information needed to onboard the IoT devices to their networks (i.e., *device bootstrapping information*)
- Integrating support for network-layer onboarding with additional security capabilities to provide ongoing protection throughout the device lifecycle

2 How to Use This Guide

This NIST Cybersecurity Practice Guide demonstrates a standards-based reference design for implementing trusted IoT device network-layer onboarding and lifecycle management and describes various example implementations of this reference design. Each of these implementations, which are known as *builds,* is standards-based and is designed to help provide assurance that networks are not put at risk as new IoT devices are added to them and help safeguard IoT devices from connecting to unauthorized networks. The reference design described in this practice guide is modular and can be deployed in whole or in part, enabling organizations to incorporate trusted IoT device network-layer onboarding and lifecycle management into their legacy environments according to goals that they have prioritized based on risk, cost, and resources.

NIST is adopting an agile process to publish this content. Each volume is being made available as soon as possible rather than delaying release until all volumes are completed.

This guide contains five volumes:

- NIST Special Publication (SP) 1800-36A: *Executive Summary* – why we wrote this guide, the challenge we address, why it could be important to your organization, and our approach to solving this challenge
- NIST SP 1800-36B: *Approach, Architecture, and Security Characteristics* – what we built and why **(you are here)**
- NIST SP 1800-36C: *How-To Guides* – instructions for building the example implementations, including all the security-relevant details that would allow you to replicate all or parts of this project
- NIST SP 1800-36D: *Functional Demonstrations* – use cases that have been defined to showcase trusted IoT device network-layer onboarding and lifecycle management security capabilities, and the results of demonstrating these use cases with each of the example implementations
- NIST SP 1800-36E: *Risk and Compliance Management* – risk analysis and mapping of trusted IoT device network-layer onboarding and lifecycle management security characteristics to cybersecurity standards and recommended practices

Depending on your role in your organization, you might use this guide in different ways:

Business decision makers, including chief security and technology officers, will be interested in the *Executive Summary, NIST SP 1800-36A*, which describes the following topics:

- challenges that enterprises face in migrating to the use of trusted IoT device network-layer onboarding
- example solutions built at the NCCoE
- benefits of adopting the example solution

Technology or security program managers who are concerned with how to identify, understand, assess, and mitigate risk will be interested in *NIST SP 1800-36B*, which describes what we did and why.

Also, Section 4 of *NIST SP 1800-36E* will be of particular interest. Section 4, *Mappings*, maps logical components of the general trusted IoT device network-layer onboarding and lifecycle management reference design to security characteristics listed in various cybersecurity standards and recommended practices documents, including *Framework for Improving Critical Infrastructure Cybersecurity* (NIST Cybersecurity Framework) and *Security and Privacy Controls for Information Systems and Organizations* (NIST SP 800-53).

You might share the *Executive Summary, NIST SP 1800-36A*, with your leadership team members to help them understand the importance of using standards-based implementations for trusted IoT device network-layer onboarding and lifecycle management.

IT professionals who want to implement similar solutions will find all volumes of the practice guide useful. You can use the how-to portion of the guide, *NIST SP 1800-36C*, to replicate all or parts of the builds created in our lab. The how-to portion of the guide provides specific product installation, configuration, and integration instructions for implementing the example solution. We do not re-create the product manufacturers' documentation, which is generally widely available. Rather, we show how we incorporated the products together in our environment to create an example solution. Also, you can use *Functional Demonstrations, NIST SP 1800-36D*, which provides the use cases that have been defined to showcase trusted IoT device network-layer onboarding and lifecycle management security capabilities and the results of demonstrating these use cases with each of the example implementations. Finally, *NIST SP 1800-36E* will be helpful in explaining the security functionality that the components of each build provide.

This guide assumes that IT professionals have experience implementing security products within the enterprise. While we have used a suite of commercial products to address this challenge, this guide does not endorse these particular products. Your organization can adopt this solution or one that adheres to these guidelines in whole, or you can use this guide as a starting point for tailoring and implementing parts of a trusted IoT device network-layer onboarding and lifecycle management solution. Your organization's security experts should identify the products that will best integrate with your existing tools and IT system infrastructure. We hope that you will seek products that are congruent with applicable standards and recommended practices.

A NIST Cybersecurity Practice Guide does not describe "the" solution, but example solutions. We seek feedback on the publication's contents and welcome your input. Comments, suggestions, and success stories will improve subsequent versions of this guide. Please contribute your thoughts to iot-onboarding@nist.gov.

2.1 Typographic Conventions

The following table presents typographic conventions used in this volume.

Typeface/Symbol	Meaning	Example
Italics	file names and path names; references to documents that are not hyperlinks; new terms; and placeholders	For language use and style guidance, see the *NCCoE Style Guide*.
Bold	names of menus, options, command buttons, and fields	Choose **File** > **Edit**.
`Monospace`	command-line input, onscreen computer output, sample code examples, and status codes	`mkdir`
`Monospace Bold`	command-line user input contrasted with computer output	**`service sshd start`**
blue text	link to other parts of the document, a web URL, or an email address	All publications from NIST's NCCoE are available at https://www.nccoe.nist.gov.

3 Approach

This project builds on the document-based research presented in the NIST Draft Cybersecurity White Paper, *Trusted Internet of Things (IoT) Device Network-Layer Onboarding and Lifecycle Management* [2]. That paper describes key security and other characteristics of a trusted network-layer onboarding solution as well as the integration of onboarding with related technologies such as device attestation, device communications intent [3][4], and application-layer onboarding. The security and other attributes of the onboarding process that are cataloged and defined in that paper can provide assurance that the network is not put at risk as new IoT devices are added to it and also that IoT devices are safeguarded from being taken over by unauthorized networks.

To kick off this project, the NCCoE published a Federal Register Notice [5] inviting technology providers to participate in demonstrating approaches to deploying trusted IoT device network-layer onboarding and lifecycle management in home and enterprise networks, with the objective of showing how trusted IoT device network-layer onboarding can practically and effectively enhance the overall security of IoT devices and, by extension, the security of the networks to which they connect. The Federal Register Notice invited technology providers to provide products and/or expertise to compose prototypes. Components sought included network onboarding components and IoT devices that support trusted network-layer onboarding protocols; authorization services; supply chain integration services; access points, routers, or switches; components that support device communications intent management; attestation services; controllers or application services; IoT device lifecycle management services; and asset management services. Cooperative Research and Development Agreements (CRADAs) were established with qualified respondents, and teams of collaborators were assembled to build a variety of implementations.

NIST is following an agile methodology of building implementations iteratively and incrementally, starting with network-layer onboarding and gradually integrating additional capabilities that improve device and network security throughout a managed device lifecycle. The project team began by designing a general, protocol-agnostic reference architecture for trusted network-layer onboarding (see Section 4) and establishing a laboratory infrastructure at the NCCoE to host implementations (see Section 5).

Five build teams were established to implement trusted network-layer onboarding prototypes, and a sixth build team was established to demonstrate multiple builds for factory provisioning activities performed by an IoT device manufacturer to enable devices to support trusted network-layer onboarding. Each of the build teams fleshed out the initial architectures of their example implementations. They then used technologies, capabilities, and components from project collaborators to begin creating the builds:

- Build 1 (Wi-Fi Easy Connect, Aruba/HPE) uses components from Aruba, a Hewlett Packard Enterprise company, to support trusted network-layer onboarding using the Wi-Fi Alliance's Wi-Fi Easy Connect Specification, Version 2.0 [6] and independent (see Section 3.3.2) application-layer onboarding to the Aruba User Experience Insight (UXI) cloud.
- Build 2 (Wi-Fi Easy Connect, CableLabs, OCF) uses components from CableLabs to support trusted network-layer onboarding using the Wi-Fi Easy Connect protocol that allows provisioning of per-device credentials and policy management for each device. Build 2 also uses components from the Open Connectivity Foundation (OCF) to support streamlined (see Section 3.3.2) trusted application-layer onboarding to the OCF security domain.
- Build 3 (BRSKI, Sandelman Software Works) uses components from Sandelman Software Works to support trusted network-layer onboarding using the Bootstrapping Remote Secure Key Infrastructure (BRSKI) [7] protocol and an independent, third-party Manufacturer Authorized Signing Authority (MASA).

- Build 4 (Thread [8], Silicon Labs, Kudelski IoT) uses components from Silicon Labs to support connection to an OpenThread [9] network using pre-shared credentials and components from Kudelski IoT to support trusted application-layer onboarding to the Amazon Web Services (AWS) IoT core.
- Build 5 (BRSKI over Wi-Fi, NquiringMinds) uses components from NquiringMinds to support trusted network-layer onboarding using the BRSKI protocol over 802.11 [10]. Additional components from NquiringMinds support ongoing, policy-based, continuous assurance and authorization, as well as device communications intent enforcement.
- The BRSKI Factory Provisioning Build uses components from NquiringMinds to implement the factory provisioning flows. The build is implemented on Raspberry Pi devices, where the IoT secure element is an integrated Infineon Optiga™ SLB 9670 TPM 2.0. The device certificate authority (CA) is externally hosted on NquiringMinds servers. This build demonstrates activities for provisioning IoT devices with their initial (i.e., birth—see Section 3.3) credentials for use with the BRSKI protocol and for making device bootstrapping information available to device owners.
- The Wi-Fi Easy Connect Factory Provisioning Build uses Raspberry Pi devices and code from Aruba and secure storage elements, code, and a CA from SEALSQ, a subsidiary of WISeKey. This build demonstrates activities for provisioning IoT devices with their birth credentials for use with the Wi-Fi Easy Connect protocol and for making device bootstrapping information available to device owners.

Each build team documented the architecture and design of its build (see Appendix C, Appendix D, Appendix E, Appendix F, Appendix G, and Appendix H). As each build progressed, its team also documented the steps taken to install and configure each component of the build (see NIST SP 1800-36C).

The project team then designed a set of use case scenarios designed to showcase the builds' security capabilities. Each build team conducted a functional demonstration of its build by running the build through the defined scenarios and documenting the results (see NIST SP 1800-36D).

The project team also conducted a risk assessment and a security characteristic analysis and documented the results, including mappings of the security capabilities of the reference solution to both the *Framework for Improving Critical Infrastructure Cybersecurity* (NIST Cybersecurity Framework) [11] and Security and Privacy Controls for Information Systems and Organizations (*NIST SP 800-53 Rev. 5*) (see NIST SP 1800-36E).

Finally, the NCCoE worked with industry and standards-developing organization collaborators to distill their findings and consider potential enhancements to future support for trusted IoT device network-layer onboarding (see Section 6 and Section 7).

3.1 Audience

The intended audience for this practice guide includes:

- IoT device manufacturers, integrators, and vendors
- Semiconductor manufacturers and vendors
- Secure storage manufacturers

- Network equipment manufacturers
- IoT device owners and users
- Owners and administrators of networks (both home and enterprise) to which IoT devices connect
- Service providers (internet service providers/cable operators and application platform providers)

3.2 Scope

This project focuses on the trusted network-layer onboarding of IoT devices in both home and enterprise environments. Enterprise, consumer, and industrial use cases for trusted IoT device network-layer onboarding are all considered to be in scope at this time. The project encompasses trusted network-layer onboarding of IoT devices deployed across different Internet Protocol (IP) based environments using wired, Wi-Fi, and broadband networking technologies. The project addresses the onboarding of IP-based devices in the initial phase and will consider using technologies such as Zigbee or Bluetooth in future phases of this project.

The project's scope also includes security technologies that can be integrated with and enhanced by the trusted network-layer onboarding mechanism to protect the device and its network throughout the device's lifecycle. Examples of these technologies include supply chain management, device attestation, trusted application-layer onboarding, device communications intent enforcement, device lifecycle management, asset management, the dynamic assignment of devices to various network segments, and ongoing device authorization. Aspects of these technologies that are relevant to their integration with network-layer onboarding are within scope. Demonstration of the general capabilities of these technologies independent of onboarding is not within the project's scope. For example, demonstrating a policy that requires device attestation to be performed before the device will be permitted to be onboarded would be within scope. However, the details and general operation of the device attestation mechanism would be out of scope.

3.3 Assumptions and Definitions

This project is guided by a variety of assumptions, which are categorized by subsection below.

3.3.1 Credential Types

There are several different credentials that may be related to any given IoT device, which makes it important to be clear about which credential is being referred to. Two types of IoT device credentials are involved in the network-layer onboarding process: birth credentials and network credentials. Birth credentials are installed onto the device before it is released into the supply chain; trusted network-layer onboarding solutions leverage birth credentials to authenticate devices and securely provision them with their network credentials. If supported by the device and the application service provider, application-layer credentials may be provisioned to the device after the device performs network-layer onboarding and connects to the network, during the application-layer onboarding process. These different types of IoT device credentials are defined as follows:

- **Birth Credential**: In order to participate in trusted network-layer onboarding, devices must be equipped with a birth credential, which is sometimes also referred to as a device *birth identity* or *birth certificate.* A birth credential is a unique, authoritative credential that is generated or installed into secure storage on the IoT device during the pre-market phase of the device's lifecycle, i.e., before the device is released for sale. A manufacturer, integrator, or vendor typically generates or installs the birth credential onto an IoT device in the form of an Initial Device Identifier (IDevID) [12] and/or a public/private key pair.

  Birth credentials:

  - are permanent, and their value is independent of context;
  - enable the trusted network-layer onboarding process while keeping the device manufacturing process efficient; and
  - include a unique identity and a secret and can range from simple raw public and private keys to X.509 certificates that are signed by a trusted authority.

- **Network Credential:** A network credential is the credential that is provisioned to an IoT device during network-layer onboarding. The network credential enables the device to connect to the local network securely. A device's network credential may be changed repeatedly, as needed, by subsequent invocation of the trusted network-layer onboarding process.

Additional types of credentials that may also be associated with an IoT device are:

- **Application-Layer Credential:** An application-layer credential is a credential that is provisioned to an IoT device during application-layer onboarding. After an IoT device has performed network-layer onboarding and connected to a network, it may be provisioned with one or more application-layer credentials during the application-layer onboarding process. Each application-layer credential is specific to a given application and is typically unique to the device, and it may be replaced repeatedly over the course of the device's lifetime.
- **User Credential:** An IoT device that permits authorized users to access it and restricts access only to authorized users will have one or more user credentials associated with it. These credentials are what the users present to the IoT device in order to gain access to it. The user credential is not relevant during network-layer onboarding and is generally not of interest within the scope of this project. We include it in this list only for completeness. Many IoT devices may not even have user credentials associated with them.

In order to perform network- and application-layer onboarding, the device being onboarded must already have been provisioned with birth credentials. A pre-provisioned, unique, authoritative birth credential is essential for enabling the IoT device to be identified and authenticated as part of the trusted network-layer onboarding process, no matter what network the device is being onboarded to or how many times it is onboarded. The value of the birth credential is independent of context, whereas the network credential that is provisioned during network-layer onboarding is significant only with respect to the network to which the IoT device will connect. Each application-layer credential that is provisioned during application-layer onboarding is specific to a given application, and each user credential is specific to a given user. A given IoT device only ever has one birth credential over the course of its lifetime, and the value of this birth credential remains unchanged. However, that IoT device may have any number of network, application-layer, and user credentials at any given point in time, and these credentials may be replaced repeatedly over the course of the device's lifetime.

3.3.2 Integrating Security Enhancements

Integrating trusted network-layer IoT device onboarding with additional security mechanisms and technologies can help increase trust in both the IoT device and the network to which it connects. Examples of such security mechanism integrations demonstrated in this project include:

- **Trusted Application-Layer Onboarding:** When supported, application-layer onboarding can be performed automatically after a device has connected to its local network. Trusted application-layer onboarding enables a device to be securely provisioned with the application-layer credentials it needs to establish a secure association with a trusted application service. In many cases, a network's IoT devices will be so numerous that manually onboarding devices at the application layer would not be practical; in addition, dependence on manual application-layer onboarding would leave the devices vulnerable to accidental or malicious misconfiguration. So, application-layer onboarding, like network-layer onboarding, is fundamental to ensuring the overall security posture of each IoT device.

  As part of the application-layer onboarding process, devices and the application services with which they interact perform mutual authentication and establish an encrypted channel over which the application service can download application-layer credentials and software to the device and the device can provide information to the application service, as appropriate. Application-layer onboarding is useful for ensuring that IoT devices are executing the most up-to-date versions of their intended applications. It can also be used to establish a secure association between a device and a trusted lifecycle management service, which will ensure that the IoT device continues to be patched and updated with the latest firmware and software, thereby enabling the device to remain trusted throughout its lifecycle.

  Network-layer onboarding cannot be performed until after network-layer bootstrapping information has been introduced to the device and the network. This network-layer bootstrapping information enables the device and the network to mutually authenticate and establish a secure channel. Analogously, application-layer onboarding cannot be performed until after application-layer bootstrapping information has been introduced to the device and the application servers with which they will onboard. This application-layer bootstrapping information enables the device and the application server to mutually authenticate and establish a secure channel.

  - *Streamlined Application-Layer Onboarding*—One potential mechanism for introducing this application-layer bootstrapping information to the device and the application server is to use the network-layer onboarding process. The secure channel that is established during network-layer onboarding can serve as the mechanism for exchanging application-layer bootstrapping information between the device and the application server. By safeguarding the integrity and confidentiality of the application-layer bootstrapping information as it is conveyed between the device and the application server, the trusted network-layer onboarding mechanism helps to ensure that information that the device and the application server use to authenticate each other is truly secret and known only to them, thereby establishing a firm foundation for their secure association. In this way, trusted network-layer onboarding can provide a secure foundation for trusted application-layer onboarding. We call an application-layer onboarding process that uses network-layer onboarding to exchange application-layer bootstrapping information *streamlined* application-layer onboarding.

- *Independent Application-Layer Onboarding*—An alternative mechanism for introducing application-layer bootstrapping information to the device is to provide this information to the device during the manufacturing process. During manufacturing, the IoT device can be provisioned with software and associated bootstrapping information that enables the device to mutually authenticate with an application-layer service after it has connected to the network. This mechanism for performing application-layer onboarding does not rely on the network-layer onboarding process to provide application-layer bootstrapping information to the device. All that is required is that the device have connectivity to the application-layer onboarding service after it has connected to the network. We call an application-layer onboarding process that does not rely on network-layer onboarding to exchange application-layer bootstrapping information *independent* application-layer onboarding.

- **Segmentation:** Upon connection to the network, a device may be assigned to a particular local network segment to prevent it from communicating with other network components, as determined by enterprise policy. The device can be protected from other local network components that meet or do not meet certain policy criteria. Similarly, other local network components may be protected from the device if it meets or fails to meet certain policy criteria. A trusted network-layer onboarding mechanism may be used to convey information about the device that can be used to determine to which network segment it should be assigned upon connection. By conveying this information in a manner that protects its integrity and confidentiality, the trusted network-layer onboarding mechanism helps to increase assurance that the device will be assigned to the appropriate network segment. Post-onboarding, if a device becomes untrustworthy, for example because it is found to have software that has a known vulnerability or misconfiguration, or because it is behaving in a suspicious manner, the device may be dynamically assigned to a different network segment as a means of quarantining it, or its network-layer credential can be revoked or deleted.

- **Ongoing Device Authorization:** Once a device has been network-layer onboarded in a trusted manner and has possibly performed application-layer onboarding as well, it is important that as the device continues to operate on the network, it maintains a secure posture throughout its lifecycle. Ensuring the ongoing security of the device is important for keeping the device from being corrupted and for protecting the network from a potentially harmful device. Even though a device is authenticated and authorized prior to being onboarded, it is recommended that the device be subject to ongoing policy-based authentication and authorization as it continues to operate on the network. This may include monitoring device behavior and constraining communications to and from the device as needed in accordance with policy. In this manner, an ongoing device authorization service can ensure that the device and its operations continue to be authorized throughout the device's tenure on the network.

- **Device Communications Intent Enforcement:** Network-layer onboarding protocols can be used to securely transmit device communications intent information from the device to the network (i.e., to transmit this information in encrypted form with integrity protections). After the device has securely connected to the network, the network can use this device communications intent information to ensure that the device sends and receives traffic only from authorized locations. Secure conveyance of device communications intent information, combined with enforcement of it, ensures that IoT devices are constrained to sending and receiving only those communications that are explicitly required for each device to fulfill its purpose.

- **Additional Security Mechanisms:** Although not demonstrated in the implementations that have been built in this project so far, numerous additional security mechanisms can potentially be integrated with network-layer onboarding, beginning at device boot-up and extending through all phases of the device lifecycle. Examples of such mechanisms include integration with supply chain management tools, device attestation, automated lifecycle management, mutual attestation, and centralized asset management. Overall, application of these and other security protections can create a dependency chain of protections. This chain is based on a hardware root of trust as its foundation and extends up to support the security of the trusted network-layer onboarding process. The trusted network-layer onboarding process in turn may enable additional capabilities and provide a foundation that makes them more secure, thereby helping to ensure the ongoing security of the device and, by extension, the network.

3.3.3 Device Limitations

The security capabilities that any onboarding solution will be able to support will depend in part on the hardware, processing power, cryptographic modules, secure storage capacity, battery life, human interface (if any), and other capabilities of the IoT devices themselves, such as whether they support verification of firmware at boot time, attestation, application-layer onboarding, and device communications intent enforcement; what onboarding and other protocols they support; and whether they are supported by supply-chain tools. The more capable the device, the more security capabilities it should be able to support and the more robustly it should be able to support them. Depending on both device and onboarding solution capabilities, different levels of assurance may be provided.

3.3.4 Specifications Are Still Improving

Ideally, trusted network-layer onboarding solutions selected for widespread implementation and use will be openly available and standards-based. Some potential solution specifications are still being improved. In the meantime, their instability may be a limiting factor in deploying operational implementations of the proposed capabilities. For example, the details of running BRSKI over Wi-Fi are not fully specified at this time.

3.4 Collaborators and Their Contributions

Organizations participating in this project submitted their capabilities in response to an open call in the Federal Register for all sources of relevant security capabilities from academia and industry (vendors and integrators). Listed below are the respondents with relevant capabilities or product components (identified as “Technology Partners/Collaborators” herein) who signed a CRADA to collaborate with NIST in a consortium to build example trusted IoT device network-layer onboarding solution.

Technology Collaborators

Aruba, a Hewlett Packard Enterprise company

CableLabs

Cisco

Foundries.io

Kudelski IoT

NquiringMinds

NXP Semiconductors

Open Connectivity Foundation (OCF)

Sandelman Software Works

SEALSQ, a subsidiary of WISeKey

Silicon Labs

Table 3-1 summarizes the capabilities and components provided, or planned to be provided, by each partner/collaborator.

Table 3-1 Capabilities and Components Provided by Each Technology Partner/Collaborator

Collaborator	Security Capability or Component Provided
Aruba	Infrastructure for trusted network-layer onboarding using the Wi-Fi Easy Connect protocol and application-layer onboarding to the UXI cloud. IoT devices for use with both Wi-Fi Easy Connect network-layer onboarding and application-layer onboarding. The UXI Dashboard provides for an "always-on" remote technician with near real-time data insights into network and application performance.
CableLabs	Infrastructure for trusted network-layer onboarding using the Wi-Fi Easy Connect protocol. IoT devices for use with both Wi-Fi Easy Connect network-layer onboarding and application-layer onboarding to the OCF security domain.
Cisco	Networking components to support various builds.
Foundries.io	Factory software for providing birth credentials into secure storage on IoT devices and for transferring device bootstrapping information from device manufacturer to device purchaser.
Kudelski IoT	Infrastructure for trusted application-layer onboarding of a device to the AWS IoT core. The service comes with a cloud platform and a software agent that enables secure provisioning of AWS credentials into the secure storage of IoT devices.
NquiringMinds	Infrastructure for trusted network-layer onboarding using BRSKI over 802.11. Service that performs ongoing monitoring of connected devices to ensure their continued authorization (i.e., continuous authorization service), as well as device communications intent enforcement.
NXP Semiconductors	IoT devices with secure storage for use with both Wi-Fi Easy Connect and BRSKI network-layer onboarding. Service for provisioning credentials into secure storage of IoT devices.
Open Connectivity Foundation (OCF)	Infrastructure for trusted application-layer onboarding to the OCF security domain using IoTivity, an open-source software framework that implements the OCF specification.
Sandelman Software Works	Infrastructure for trusted network-layer onboarding using BRSKI. IoT devices for use with BRSKI network-layer onboarding.
SEALSQ, a subsidiary of WISeKey	Secure storage elements, code, and software that simulates factory provisioning of birth credentials to those secure elements on IoT devices in support of both Wi-Fi Easy Connect and BRSKI network-layer onboarding; certificate authority for signing device certificates.
Silicon Labs	Infrastructure for connection to a Thread network that has access to other networks for application-layer onboarding. IoT device with secure storage for use with Thread network connection and application-layer onboarding using Kudelski IoT.

Each of these technology partners and collaborators has described the relevant products and capabilities it brings to this trusted onboarding effort in the following subsections. The NCCoE does not certify or validate products or services. We demonstrate the capabilities that can be achieved by using participants' contributed technology.

3.4.1 Aruba, a Hewlett Packard Enterprise Company

Aruba, a Hewlett Packard Enterprise (HPE) company, provides secure, intelligent edge-to-cloud networking solutions that use artificial intelligence (AI) to automate the network, while harnessing data to drive powerful business outcomes. With Aruba ESP (Edge Services Platform) and as-a-service options as part of the HPE GreenLake family, Aruba takes a cloud-native approach to helping customers meet their connectivity, security, and financial requirements across campus, branch, data center, and remote worker environments, covering all aspects of wired, wireless local area networking (LAN), and wide area networking (WAN). Aruba ESP provides unified solutions for connectivity, visibility, and control throughout the IT-IoT workflow, with the objective of helping organizations accelerate IoT-driven digital transformation with greater ease, efficiency, and security. To learn more, visit Aruba at https://www.arubanetworks.com/.

3.4.1.1 Device Provisioning Protocol

Device Provisioning Protocol (DPP), certified under the Wi-Fi Alliance (WFA) as "Easy Connect," is a standard developed by Aruba that allows IoT devices to be easily provisioned onto a secure network. DPP improves security by leveraging Wi-Fi Protected Access 3 (WPA3) to provide device-specific credentials, enhance certificate handling, and support robust, secure, and scalable provisioning of IoT devices in any commercial, industrial, government, or consumer application. Aruba implements DPP through a combination of on-premises hardware and cloud-based services as shown in Table 3-1.

3.4.1.2 Aruba Access Point (AP)

From their unique vantage as ceiling furniture, Aruba Wi-Fi 6 APs have an unobstructed overhead view of all nearby devices. Built-in Bluetooth Low Energy (BLE) and Zigbee 802.15.4 IoT radios, as well as a flexible USB port, provide IoT device connectivity that allows organizations to address a broad range of IoT applications with infrastructure already in place, eliminating the cost of gateways and IoT overlay networks while enhancing IoT security.

Aruba's APs enable a DPP network through an existing Service Set Identifier (SSID) enforcing DPP access control and advertising the Configurator Connectivity Information Element (IE) to attract unprovisioned clients (i.e., clients that have not yet been onboarded). Paired with Aruba's cloud management service "Central", the APs implement the DPP protocol. The AP performs the DPP network introduction protocol (Connector exchange) with provisioned clients and assigns network roles.

3.4.1.3 Aruba Central

Aruba Central is a cloud-based networking solution with AI-powered insights, workflow automation, and edge-to-cloud security that empowers IT teams to manage and optimize campus, branch, remote, data center, and IoT networks from a single point of visibility and control. Built on a cloud-native, microservices architecture, Aruba Central is designed to simplify IT and IoT operations, improve agility, and reduce costs by unifying management of all network infrastructure.

Aruba's "Central" Cloud DPP service exposes and controls many centralized functions to enable a seamless integrated end-to-end solution and act as a DPP service orchestrator. The cloud based DPP service selects an AP to authenticate unprovisioned enrollees (in the event that multiple APs receive the client *chirps*). The DPP cloud service holds the Configurator signing key and generates Connectors for enrollees authenticated through an AP.

3.4.1.4 IoT Operations

Available within Aruba Central, the IoT Operations service extends network administrators' view into IoT devices and applications connected to the network. Organizations can gain critical visibility into previously invisible IoT devices, as well as reduce costs and complexity associated with deploying IoT applications. IoT Operations comprises three core elements:

- IoT Dashboard, which provides a granular view of devices connected to Aruba APs, as well as IoT connectors and applications in use.
- IoT App Store, a repository of click-and-go IoT applications that interface with IoT devices and their data.
- IoT Connector, which provisions multiple applications to be computed at the edge for agile IoT application support.

3.4.1.5 Client Insights

Part of Aruba Central, AI-powered Client Insights automatically identifies each endpoint connecting to the network with up to 99% accuracy. Client Insights discovers and classifies all connected endpoints—including IoT devices—using built-in machine learning and dynamic profiling techniques, helping organizations better understand what's on their networks, automate access privileges, and monitor the behavior of each endpoint's traffic flows to more rapidly spot attacks and act.

3.4.1.6 Cloud Auth

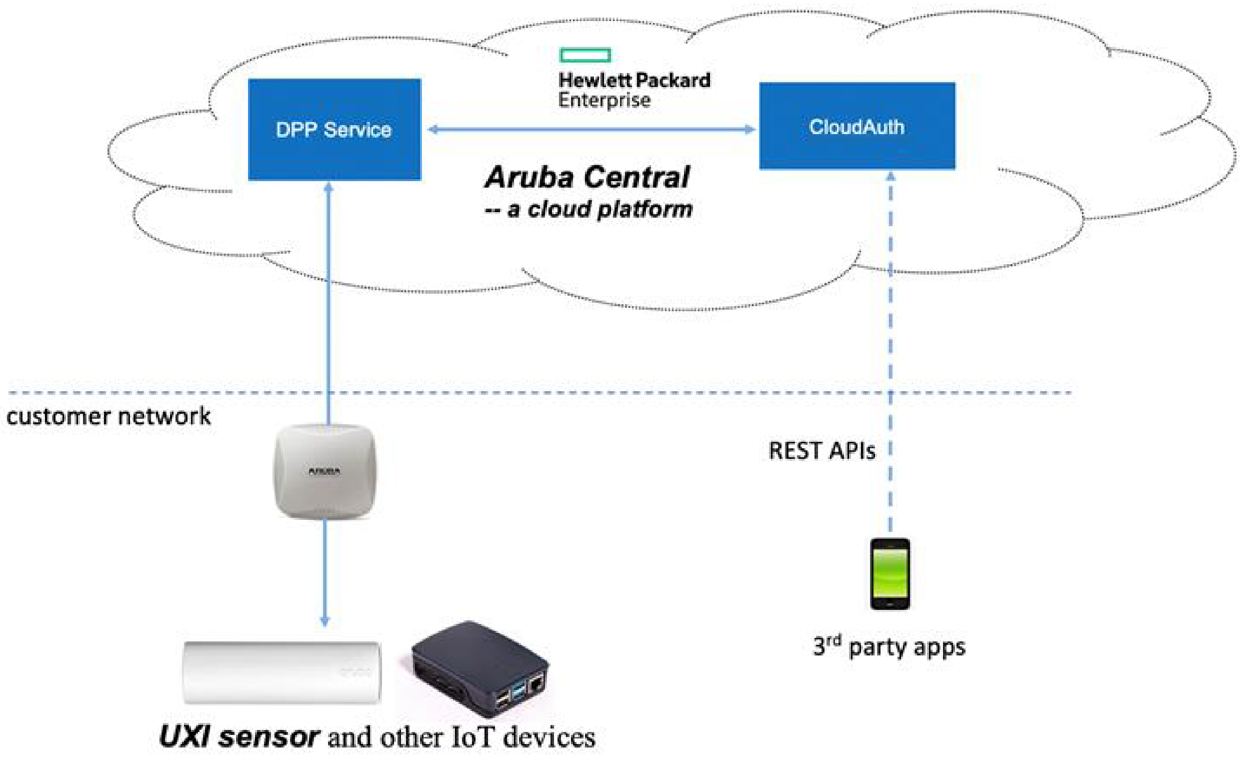
Cloud-native network access control (NAC) solution Cloud Auth delivers time-saving workflows to configure and manage onboarding, authorization, and authentication policies for wired and wireless networks. Cloud Auth integrates with an organization's existing cloud identity store, such as Google Workspace or Azure Active Directory, to authenticate IoT device information and assign the right level of network access.

Cloud Auth operates as the DPP Authorization server and is the repository for trusted DPP Uniform Resource Identifiers (URIs) of unprovisioned enrollees. It maintains role information for each unprovisioned DPP URI and provisioned devices based on unique per-device credential (public key extracted from Connector). Representational State Transfer (RESTful) application programming interfaces (APIs) provide extensible capabilities to support third parties, making an easy path for integration and collaborative deployments.

3.4.1.7 UXI Sensor: DPP Enrollee

User Experience Insight (UXI) sensors continuously monitor end-user experience on customer networks and provide a simple-to-use cloud-based dashboard to assess networks and applications. The UXI sensor is onboarded in a zero-touch experience using DPP. Once network-layer onboarding is complete, the UXI sensor performs application-layer onboarding to the Aruba cloud to download a customer-specific profile. This profile enables the UXI sensor to perform continuous network testing and monitoring, and to troubleshoot network issues that it finds.

Figure 3-1 Aruba/HPE DPP Onboarding Components



3.4.2 CableLabs

CableLabs is an innovation lab for future-forward research and development (R&D)—a global meeting of minds dedicated to building and orchestrating emergent technologies. By convening peers and experts to share knowledge, CableLabs' objective is to energize the industry ecosystem for speed and scale. Its research facilitates solutions with the goal of making connectivity faster, easier, and more secure, and its conferences and events offer neutral meeting points to gain consensus.

As part of this project, CableLabs has provided the reference platform for its Custom Connectivity architecture for the purpose of demonstrating trusted network-layer onboarding of Wi-Fi devices using a variety of credentials. The following components are part of the reference platform.

3.4.2.1 Platform Controller

The controller provides interfaces and messaging for managing service deployment groups, access points with the deployment groups, registration and lifecycle of user services, and the secure onboarding and lifecycle management of users' Wi-Fi devices. The controller also exposes APIs for integration with third-party systems for the purpose of integrating various business flows (e.g., integration with manufacturing process for device management).

3.4.2.2 Custom Connectivity Gateway Agent

The Gateway Agent is a software component that resides on the Wi-Fi AP and gateway. It connects with the controller to coordinate the Wi-Fi and routing capabilities on the gateway. Specifically, it enforces the policies and configuration from the controller by managing the lifecycle of the Wi-Fi Extended Service Set/Basic Service Set (ESS/BSS) on the AP, authentication and credentials of the client devices that connect to the AP, and service management and routing rules for various devices. It also manages secure onboarding capabilities like Easy Connect, simple onboarding using a per-device pre-shared key (PSK), etc. The Gateway agent is provided in the form of an operational Raspberry Pi-based Gateway that also includes hostapd for Wi-Fi/DPP and open-vswitch for the creation of trust domains and routing.

3.4.2.3 Reference Clients

Three Raspberry Pi-based reference clients are provided. The reference clients have support for WFA Easy Connect-based onboarding as well as support for different Wi-Fi credentials, including per-device PSK and 802.1x certificates. One of the reference clients also has support for OCF-based streamlined application-layer onboarding.

3.4.3 Cisco

Cisco Systems, or Cisco, delivers collaboration, enterprise, and industrial networking and security solutions. The company’s cybersecurity team, Cisco Secure, is one of the largest cloud and network security providers in the world. Cisco’s Talos Intelligence Group, the largest commercial threat intelligence team in the world, is comprised of world-class threat researchers, analysts, and engineers, and supported by unrivaled telemetry and sophisticated systems. The group feeds rapid and actionable threat intelligence to Cisco customers, products, and services to help identify new threats quickly and defend against them. Cisco solutions are built to work together and integrate into your environment, using the “network as a sensor” and “network as an enforcer” approach to both make your team more efficient and keep your enterprise secure. Learn more about Cisco at https://www.cisco.com/go/secure.

3.4.3.1 Cisco Catalyst Switch

A Cisco Catalyst switch is provided to support network connectivity and network segmentation capabilities.

3.4.4 Foundries.io

Foundries.io helps organizations bring secure IoT and edge devices to market faster. The FoundriesFactory cloud platform offers DevOps teams a secure Linux-based firmware/operating system (OS) platform with device and fleet management services for connected devices, based on a fixed no-royalty subscription model. Product development teams gain enhanced security from boot to cloud while reducing the cost of developing, deploying, and updating devices across their installed lifetime. The open-source platform interfaces to any cloud and offers Foundries.io customers maximum flexibility for hardware configuration, so organizations can focus on their intellectual property, applications, and value add. For more information, please visit https://foundries.io/.

3.4.4.1 FoundriesFactory

FoundriesFactory is a cloud-based software platform provided by Foundries.io that offers a complete development and deployment environment for creating secure IoT devices. It provides a set of tools and services that enable developers to create, test, and deploy custom firmware images, as well as manage the lifecycle of their IoT devices.

Customizable components include open-source secure boot software, the open-source Linux microPlatform (LmP) distribution built with Yocto and designed for secure managed IoT and edge products, secure Over the Air (OTA) update facilities, and a Docker runtime for managing containerized applications and services. The platform is cross architecture (x86, Arm, and RISC-V) and enables secure connections to public and private cloud services.

Leveraging open standards and open software, FoundriesFactory is designed to simplify and accelerate the process of developing, deploying, and managing IoT and edge devices at scale, while also ensuring that they are secure and up to date over the product lifetime.

3.4.5 Kudelski IoT

Kudelski IoT is the Internet of Things division of Kudelski Group and provides end-to-end IoT solutions, IoT product design, and full-lifecycle services to IoT semiconductor and device manufacturers, ecosystem creators, and end-user companies. These solutions and services leverage the group’s 30+ years of innovation in digital business model creation; hardware, software, and ecosystem design and testing; state-of-the-art security lifecycle management technologies and services; and managed operation of complex systems.

3.4.5.1 Kudelski IoT keySTREAM™

Kudelski IoT keySTREAM is a device-to-cloud, end-to-end solution for securing all the key assets of an IoT ecosystem during its entire lifecycle. The system provides each device with a unique, immutable, unclonable identity that forms the foundation for critical IoT security functions like in-factory or in-field provisioning, data encryption, authentication, and secure firmware updates, as well as allowing companies to revoke network access for vulnerable devices if necessary. This ensures that the entire lifecycle of the device and its data can be managed.

In this project, keySTREAM is used to enable trusted application-layer onboarding. It manages the attestation of devices, ownership, and provisioning of application credentials.

3.4.6 NquiringMinds

NquiringMinds provides intelligent trusted systems, combining AI-powered analytics with cyber security fundamentals. tdx Volt is the NquiringMinds general-purpose zero-trust services infrastructure platform, upon which it has built Cyber tdx, a cognitively enhanced cyber defense service designed for IoT. Both products are the latest iteration of the TDX product family. NquiringMinds is a UK company. Since 2010, it has been deploying its solutions into smart cities, health care, industrial, agricultural, financial technology, defense, and security sectors.

NquiringMinds collaborates within the open-standards and open-source community. It focuses on the principle of continuous assurance: the ability to continually reassess security risk by intelligently reasoning across the hard and soft information sources available. NquiringMinds' primary contributions to this project, described in the subsections below, are being made available as open source.

3.4.6.1 NquiringMinds' BRSKI Protocol Implementation

NquiringMinds has open sourced their software implementation of IETF's Bootstrapping Remote Secure Key Infrastructure (BRSKI) protocol, which provides a solution for secure zero-touch (automated) bootstrap of new (unconfigured) devices. This implementation includes the necessary adaptations for BRKSI to work with Wi-Fi networks.

The open source BRSKI implementation is available under an Apache 2.0 license at: https://github.com/nqminds/brski

3.4.6.2 TrustNetZ

NquiringMinds has open sourced the TrustNetZ (Zero Trust Networking) software stack which sits on top of their BRSKI implementation. TrustNetZ embodies the network onboarding and lifecycle management concepts into an easy to replicate demonstrator which includes the IoT device, the router, the router onboarding, the registrar, the manufacturer, the manufacturer provisioning, policy enforcement and continuous assurance servers.

This software also encapsulates NquiringMinds' continuous assurance capability, enhancing the security of the network by continually assessing whether connected IoT devices meet the policy requirements of the network. The software also includes a flexible, verifiable credential-based policy framework, which can rapidly be adapted to model different security and business model scenarios. The implementation models networking onboarding flows with EAP-TLS Wi-Fi certificates.

The open source TrustNetZ implementation is available under an Apache 2.0 license at: https://github.com/nqminds/trustnetz

3.4.6.3 edgeSEC

edgeSEC is an open-source, OpenWrt-based implementation of an intelligent secure router. It implements, on an open stack, the key components needed to implement both trusted onboarding and continuous assurance of devices. It contains an implementation of the Internet Engineering Task Force (IETF) BRSKI protocols, with the necessary adaptations for wireless onboarding, fully integrated into an open operational router. It additionally implements device communications intent constraints (IETF Manufacturer Usage Description [MUD]) and behavior monitoring (IoTSF ManySecured) that support some of the more enhanced trusted onboarding use cases. EdgeSEC additionally provides the platform for an asynchronous control plane for the continuous management of multiple routers and a general-purpose policy evaluation point, which can be used to demonstrate the breadth of onboarding and monitoring use cases that can be supported.

EdgeSEC is not directly used in the build that was demonstrated for this project, but it contains critical pieces of code that have been adapted in a simplified manner for the TrustNetZ implementation.

The open source edgeSEC implementation is available under an Apache 2.0 license at: https://github.com/nqminds/edgesec

3.4.6.4 tdx Volt

tdx Volt is NquiringMinds' zero-trust infrastructure platform. It encapsulates identity management, credential management, service discovery, and smart policy evaluation. This platform is designed to simplify the end-to-end demonstration of the trusted onboarding process and provides tools for use on the IoT device, the router, applications, and clouds. Tdx Volt is used by the TrustNetZ demonstrator as a verifiable credential issuer and verifier.

Tdx Volt is an NquiringMinds' product. Documented working implementation are available at: https://docs.tdxvolt.com/en/introduction

3.4.6.5 Reference Hardware

For demonstration purposes the NquiringMinds components can be deployed using the following hardware:

Compute hosts: Raspberry Pi 4

https://www.raspberrypi.com/products/raspberry-pi-4-model-b/. The Raspberry Pis are used to host the IoT client device, the router, and all additional compute services. Other Raspberry Pi models are also likely to work but have not been tested.

TPM/Secure Element

The secure storage for the IoT device (used in network-layer onboarding and factory provisioning) is provided by an Infineon Optiga™ SLB 9670 TPM 2.0, integrated through a Geeek Pi TPM hat. https://www.infineon.com/dgdl/Infineon-OPTIGA_SLx_9670_TPM_2.0_Pi_4-ApplicationNotes-v07_19-EN.pdf?fileId=5546d4626c1f3dc3016c3d19f43972eb.

A working version of the code is also available utilizing the SEALSQ Secure element https://www.sealsq.com/semiconductors/vaultic-secure-elements/vaultic-40x.

3.4.7 NXP Semiconductors

NXP Semiconductors focuses on secure connectivity solutions for embedded applications, NXP is impacting the automotive, industrial, and IoT, mobile, and communication infrastructure markets. Built on more than 60 years of combined experience and expertise, the company has approximately 31,000 employees in more than 30 countries. Find out more at https://www.nxp.com/.

3.4.7.1 EdgeLock SE050 secure element

The EdgeLock SE050 secure element (SE) product family offers strong protection against the latest attack scenarios and an extended feature set for a broad range of IoT use cases. This ready-to-use secure element for IoT devices provides a root of trust at the silicon level and delivers real end-to-end security – from edge to cloud – with a comprehensive software package for integration into any type of device.

3.4.7.2 EdgeLock 2GO

EdgeLock 2GO is the NXP service platform designed for easy and secure deployment and management of IoT devices. This flexible IoT service platform lets the device manufacturers and service providers choose the appropriate options to optimize costs while benefiting from an advanced level of device security. The EdgeLock 2GO service provisions the cryptographic keys and certificates into the hardware root of trust of the IoT devices and simplifies the onboarding of the devices to the cloud.

3.4.7.3 i.MX 8M family

The i.MX 8M family of applications processors based on Arm® Cortex®-A53 and Cortex-M4 cores provide advanced audio, voice, and video processing for applications that scale from consumer home audio to industrial building automation and mobile computers. It includes support for secure boot, secure debug, and lifecycle management, as well as integrated cryptographic accelerators. The development boards and Linux Board Support Package enablement provide out-of-the-box integration with an external SE050 secure element.

3.4.8 Open Connectivity Foundation (OCF)

OCF is a standards-developing organization that has had contributions and participation from over 450+ member organizations representing the full spectrum of the IoT ecosystem, from chip makers to consumer electronics manufacturers, silicon enablement software platform and service providers, and network operators. The OCF specification is an International Organization for Standardization/International Electrotechnical Commission (ISO/IEC) internationally recognized standard that was built in tandem with an open-source reference implementation called IoTivity. Additionally, OCF provides an in-depth testing and certification program.

3.4.8.1 IoTivity

OCF has contributed open-source code from IoTivity that demonstrates the advantage of secure network-layer onboarding and implements the WFA's Easy Connect to power a seamless bootstrapping of secure and trusted application-layer onboarding of IoT devices with minimal user interaction.

This code includes the interaction layer, called the OCF Diplomat, which handles secure communication between the DPP-enabled access point and the OCF application layer. The OCF onboarding tool (OBT) is used to configure and provision devices with operational credentials. The OCF reference implementation of a basic lamp is used to demonstrate both network- and application-layer onboarding and to show that once onboarded and provisioned, the OBT can securely interact with the lamp.

3.4.9 Sandelman Software Works

Sandelman Software Works (SSW) provides consulting and software design services in the areas of systems and network security. A complete stack company, SSW provides consulting and design services from the hardware driver level up to Internet Protocol Security (IPsec), Transport Layer Security (TLS), and cloud database optimization. SSW has been involved with the IETF since the 1990s, now dealing with the difficult problem of providing security for IoT systems. SSW leads standardization efforts through a combination of running code and rough consensus.

3.4.9.1 Minerva Highway IoT Network-Layer Onboarding and Lifecycle Management System

The Highway component is a cloud-native component operated by the device manufacturer (or its authorized designate). It provides the Request for Comments (RFC) 8995 [7] specified Manufacturer Authorized Signing Authority (MASA) for the BRSKI onboarding mechanism.

Highway is an asset manager for IoT devices. In its asset database it maintains an inventory of devices that have been manufactured, what type they are, and who the current owner of the device is (if it has been sold). Highway does this by taking control of the complete identity lifecycle of the device. It can aid in provisioning new device identity certificates (IDevIDs) by collecting Certificate Signing Requests and returning certificates, or by generating the new identities itself. This is consistent with Section 4.1.2.1 (On-device private key generation) and Section 4.1.2.2 (Off-device private key generation) of https://www.ietf.org/archive/id/draft-irtf-t2trg-taxonomy-manufacturer-anchors-00.html.

Highway can act as a standalone three-level private-public key infrastructure (PKI). Integrations with Automatic Certificate Management Environment (RFC 8555) allow it to provision certificates from an external PKI using the DNS-01 challenge in Section 8.4 of https://www.rfc-editor.org/rfc/rfc8555.html#section-8.4. Hardware integrations allow for the private key operations to be moved out of the main CPU. However, the needs of a busy production line in a factory would require continuous access to the hardware offload.

In practice, customers put the subordinate CA into Highway, which it needs to sign new IDevIDs, and put the trust anchor private CA into a hardware security module (HSM).

Highway provides a BRSKI-MASA interface running on a public TCP/HTTPS port (usually 443 or 9443). This service requires access to the private key associated to the anchor that has been “baked into” the Pledge device during manufacturing. The Highway instance that speaks to the world in this way does not have to be the same instance that signs IDevID certificates, and there are significant security advantages to separating them. Both instances do need access to the same database servers, and there are a variety of database replication techniques that can be used to improve resilience and security.

As IDevIDs do not expire, Highway does not presently include any mechanism to revoke IDevIDs, nor does it provide Certificate Revocation Lists (CRLs) or Online Certificate Status Protocol (OCSP). It is unclear how those mechanisms can work in practice.

Highway supports two models. In the Sales Integration model, the intended owner is known in advance. This model requires customer-specific integrations, which often occur at the database level through views or other SQL tools. In the trust on first use (TOFU) model, the first customer to claim a product becomes its owner.

3.4.10 SEALSQ, a subsidiary of WISeKey

WISeKey International Holding Ltd. (WISeKey) is a cybersecurity company that deploys digital identity ecosystems and secures IoT solution platforms. It operates as a Swiss-based holding company through several operational subsidiaries, each dedicated to specific aspects of its technology portfolio.

SEALSQ is the subsidiary of the group that focuses on designing and selling secure microcontrollers, PKI, and identity provisioning services while developing post-quantum technology hardware and software products. SEALSQ products and solutions are used across a variety of applications today, from multi-factor authentication devices, home automation systems, and network infrastructure, to automotive, industrial automation, and control systems.

3.4.11 VaultIC408

The VaultIC408 secure element combines hardware-based key storage with cryptographic accelerators to provide a wide array of cryptographic features including identity, authentication, encryption, key agreement, and data integrity. It protects against hardware attacks such as micro-probing and side channels.

The fundamental cryptography of the VaultIC family includes NIST-recommended algorithms and key lengths. Each of these algorithms, Elliptic Curve Cryptography (ECC), Rivest-Shamir-Adleman (RSA), and Advanced Encryption Standard (AES), is implemented on-chip and uses on-chip storage of the secret key material so the secrets are always protected in the secure hardware.

The secure storage and cryptographic acceleration support use cases like network and IoT end node security, platform security, secure boot, secure firmware download, secure communication or TLS, data confidentiality, encryption key storage, and data integrity.

3.4.11.1 INeS Certificate Management System (CMS)

SEALSQ's portfolio includes INeS, a managed PKI-as-a-service solution. INeS leverages the WISeKey Webtrust-accredited trust services platform, a Matter approved Product Attestation Authority (PAA), and custom CAs. These PKI technologies support large-scale IoT deployments, where IoT endpoints will require certificates to establish their identities. The INeS CMS platform provides a secure, scalable, and manageable trust model.

INeS CMS provides certificate management, CA management, public cloud integration and automation, role-based access control (RBAC), and APIs for custom implementations.

3.4.12 Silicon Labs

Silicon Labs provides products in the area of secure, intelligent wireless technology for a more connected world. Securing IoT is challenging. It's also mission critical. The challenge of protecting connected devices against frequently surfacing IoT security vulnerabilities follows device makers throughout the entire product lifecycle. Protecting products in a connected world is a necessity as customer data and modern online business models are increasingly targets for costly hacks and corporate brand damage. To stay secure, device makers need an underlying security platform in the hardware, software, network, and cloud. Silicon Labs offers security products with features that address escalating IoT threats, with the goal of reducing the risk of IoT ecosystem security breaches and the compromise of intellectual property and revenue loss from counterfeiting.

For this project, Silicon Labs has provided a host platform for the OpenThread border router (OTBR), a Thread radio transceiver, and an IoT device to be onboarded to the AWS cloud service and that communicates using the Thread wireless protocol.

3.4.12.1 OpenThread Border Router Platform

A Raspberry Pi serves as host platform for the OTBR. The OTBR forms a Thread network and acts as a bridge between the Thread network and the public internet, allowing the IoT device that communicates using the Thread wireless protocol and that is to be onboarded communicate with cloud services. The OTBR’s connection to the internet can be made through either Wi-Fi or ethernet. Connection to the SLWSTK6023A (see Section 3.4.12.2) is made through a USB serial port.

3.4.12.2 SLWSTK6023A Thread Radio Transceiver

The SLWSTK6023A (Wireless starter kit) acts as a Thread radio transceiver or radio coprocessor (RCP). This allows the OTBR host platform to form and communicate with a Thread network.

3.4.12.3 xG24-DK2601B Thread “End” Device

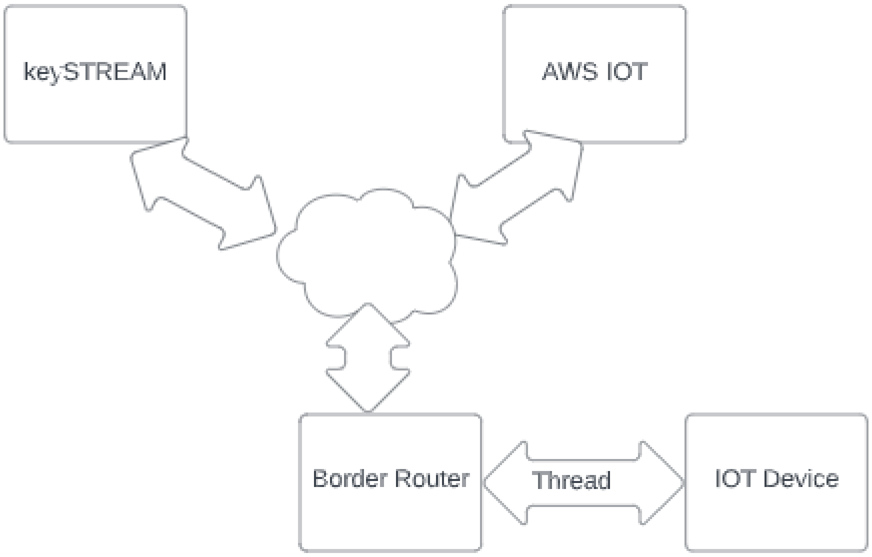
The xG24-DK2601B is the IoT device that is to be onboarded to the cloud service (AWS). It communicates using the Thread wireless protocol. Communication is bridged between the Thread network and the internet by the OTBR.

3.4.12.4 Kudelski IoT keySTREAM™

The Kudelski IoT keySTREAM solution is described more fully in Section 3.4.5.1. It is a cloud service capable of verifying the hardware-based secure identity certificate chain associated with the xG24-DK2601B component described in Section 3.4.12.3 and delivering a new certificate chain that can be refreshed or revoked as needed to assist with lifecycle management. The certificate chain is used to authenticate the xG24-DK2601B device to the cloud service (AWS).

Figure 3-2 shows the relationships among the components provided by Silicon Labs and Kudelski that support the trusted application-layer onboarding of an IoT device that communicates via the Thread protocol to AWS IoT.

Figure 3-2 Components for Onboarding an IoT Device that Communicates Using Thread to AWS IoT

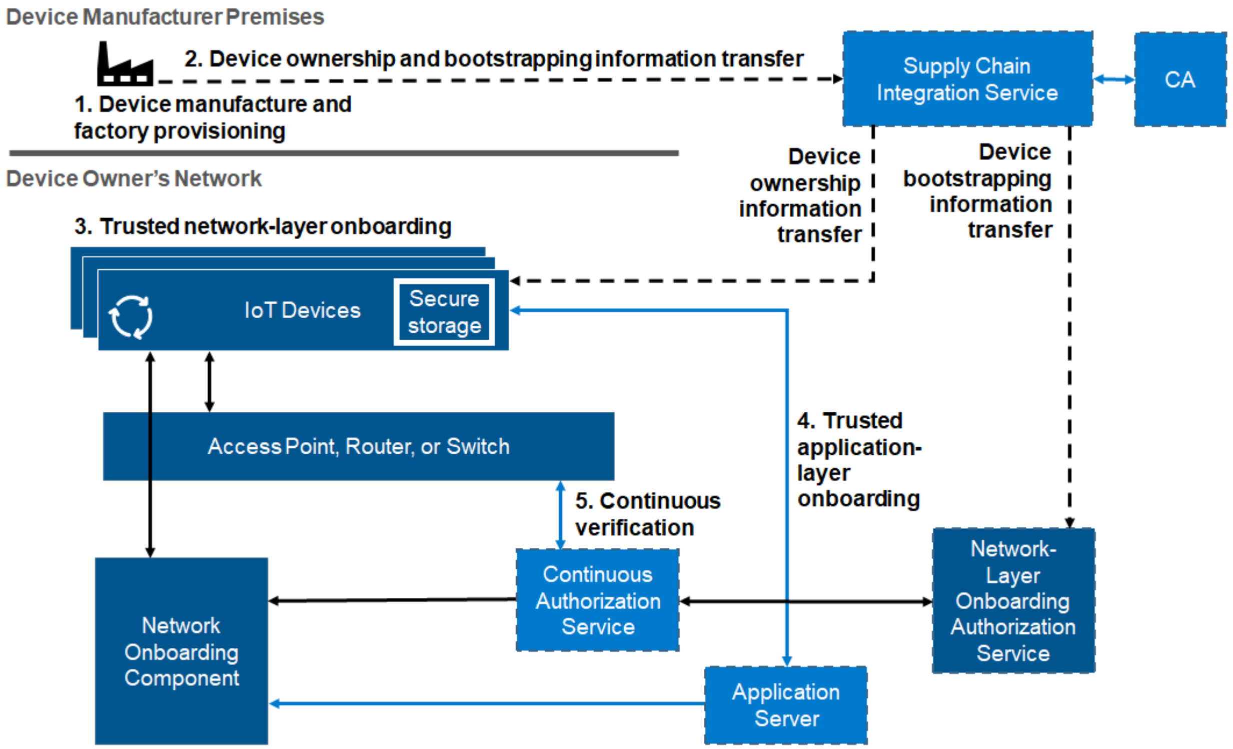


4 Reference Architecture

Figure 4-1 depicts the reference architecture to demonstrate trusted IoT device network-layer onboarding and lifecycle management used throughout this Practice Guide. This architecture shows a high-level, protocol-agnostic, and generic approach to trusted network-layer onboarding. It represents the basic components and processes, regardless of the network-layer onboarding protocol used and the particular device lifecycle management activities supported.

When implementing this architecture, an organization can follow different steps and use different components. The exact steps that are performed may not be in the same order as the steps in the logical reference architecture, and they may use components that do not have a one-to-one correspondence with the logical components in the logical reference architecture. In Appendices C, D, E, F and G we present the architectures for builds 1, 2, 3, 4 and 5, each of which is an instantiation of this logical reference architecture. Those build-specific architectures are more detailed and are described in terms of specific collaborator components and trusted network-layer onboarding protocols.

Figure 4-1 Trusted IoT Device Network-Layer Onboarding and Lifecycle Management Logical Reference Architecture


Device Manufacturer Premises
2. Device ownership and bootstrapping information transfer
1. Device manufacture and factory provisioning
Supply Chain Integration Service
CA
Device Owner's Network
Device ownership information transfer
Device bootstrapping information transfer
3. Trusted network-layer onboarding
IoT Devices
Secure storage
Access Point, Router, or Switch
4. Trusted application-layer onboarding
5. Continuous verification
Network Onboarding Component
Continuous Authorization Service
Network-Layer Onboarding Authorization Service
Application Server

There are five high-level processes to carry out this architecture, as labeled in Figure 4-1. These five processes are as follows:

1. **Device manufacture and factory provisioning** – the activities that the IoT device manufacturer performs to prepare the IoT device so that it is capable of network- and application-layer onboarding (Figure 4-2, Section 4.1).

2. **Device ownership and bootstrapping information transfer** – the transfer of IoT device ownership and bootstrapping information from the manufacturer to the device and/or the device's owner that enables the owner or an entity authorized by the owner to onboard the device securely. The component in Figure 4-1 labeled "Supply Chain Integration Service" represents the mechanism used to accomplish this information transfer (Figure 4-3, Section 4.2).

3. **Trusted network-layer onboarding** – the interactions that occur between the network-layer onboarding component and the IoT device to mutually authenticate, confirm authorization, establish a secure channel, and provision the device with its network credentials (Figure 4-4, Section 4.3).

4. **Trusted application-layer onboarding** – the interactions that occur between a trusted application server and the IoT device to mutually authenticate, establish a secure channel, and provision the device with application-layer credentials (Figure 4-5, Section 4.4).

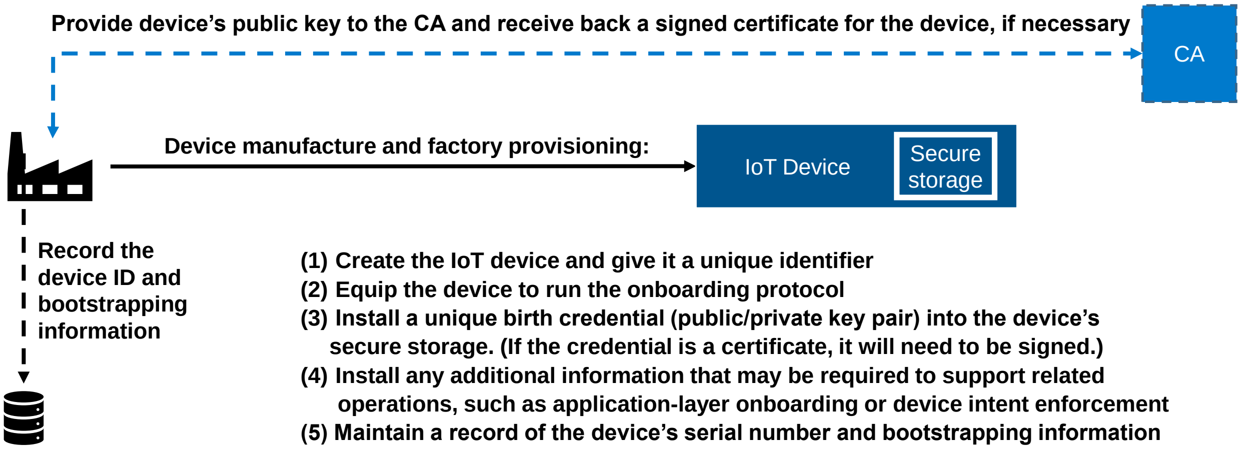
5. **Continuous verification** – ongoing, policy-based verification and authorization checks on the IoT device to support device lifecycle monitoring and control (Figure 4-6, Section 4.5).

Figure 4-1 uses two colors. The dark-blue components are central to supporting trusted network-layer onboarding itself. The light-blue components support the other aspects of the architecture. Each of the five processes is explained in more detail in the subsections below.

4.1 Device Manufacture and Factory Provisioning Process

Figure 4-2 depicts the device manufacture and factory provisioning process in more detail. As shown in Figure 4-2, the manufacturer is responsible for creating the IoT device and provisioning it with the necessary hardware, software, and birth credentials so that it is capable of network-layer onboarding. The IoT device should be manufactured with a secure root of trust as a best practice, possibly as part of a secure manufacturing process, particularly when outsourced. Visibility and control over the provisioning process and manufacturing supply chain, particularly for outsourced manufacturing, is critical in order to mitigate the risk of compromise in the supply chain, which could lead to the introduction of compromised devices. The CA component is shown in light blue in Figure 4-2 because its use is optional and depends on the type of credential that is being provisioned to the device (i.e., whether it is an 802.1AR certificate).

Figure 4-2 IoT Device Manufacture and Factory Provisioning Process


Provide device's public key to the CA and receive back a signed certificate for the device, if necessary
CA
Device manufacture and factory provisioning:
IoT Device
Secure storage
Record the device ID and bootstrapping information
(1) Create the IoT device and give it a unique identifier
(2) Equip the device to run the onboarding protocol
(3) Install a unique birth credential (public/private key pair) into the device's secure storage. (If the credential is a certificate, it will need to be signed.)
(4) Install any additional information that may be required to support related operations, such as application-layer onboarding or device intent enforcement
(5) Maintain a record of the device's serial number and bootstrapping information

At a high level, the steps that the manufacturer or an integrator performs as part of this preparation process, as shown in Figure 4-2, are as follows:

1. Create the IoT device and assign it a unique identifier (e.g., a serial number). Equip the device with secure storage.
2. Equip the device to run a specific network-layer onboarding protocol (e.g., Wi-Fi Easy Connect, BRSKI, Thread Mesh Commissioning Protocol (MeshCoP) [8]). This step includes ensuring that the device has the software/firmware needed to run the onboarding protocol as well as any additional information that may be required.
3. Generate or install the device's unique birth credential into the device's secure storage. [Note: using a secure element that has the ability to autonomously generate private/public root key pairs is inherently more secure than performing credential injection, which has the potential to expose the private key.] The birth credential includes information that must be kept secret (i.e., the device's private key) because it is what enables the device's identity to be authenticated. The contents of the birth credential will depend on what network-layer onboarding protocol the device supports. For example:
    a. If the device runs the Wi-Fi Easy Connect protocol, its birth credential will take the form of a unique private key, which has an associated DPP URI that includes the corresponding public key and possibly additional information such as Wi-Fi channel and serial number.
    b. If the device runs the BRSKI protocol, its birth credential takes the form of an 802.1AR certificate that gets installed as the device's IDevID and corresponding private key. The IDevID includes the device's public key, the location of the MASA, and trust anchors that can be used to verify vouchers signed by the MASA. The 802.1AR certificate needs to be signed by a trusted signing authority prior to installation, as shown in Figure 4-2.
4. Install any additional information that may be required to support related capabilities that are enabled by network-layer onboarding. The specific contents of the information that gets

installed on the device will vary according to what capabilities it is intended to support. For example, if the device supports:

   a. **streamlined application-layer onboarding** (see Section 3.3.2), then the bootstrapping information that is required to enable the device and a trusted application server to find and mutually authenticate each other and establish a secure association will be stored on the device. This is so it can be sent to the network during network-layer onboarding and used to automatically perform application-layer onboarding after the device has securely connected to the network. The Wi-Fi Easy Connect protocol, for example, can include such application-layer bootstrapping information as third-party information in its protocol exchange with the network, and Build 2 (i.e., the Wi-Fi Easy Connect, CableLabs, OCF build) demonstrates use of this mechanism to support streamlined application-layer onboarding.

      Note, however, that a device may still be capable of performing independent [see Section 3.3.2] application-layer onboarding even if the application-layer onboarding information is not exchanged as part of the network-layer onboarding protocol. The application that is installed on the device, i.e., the application that the device executes to fulfill its purpose, may include application-layer bootstrapping information that enables it to perform application-layer onboarding when it begins executing. Build 1 (i.e., the Wi-Fi Easy Connect, Aruba/HPE build) demonstrates independent application-layer onboarding.

   b. **device communications intent**, then the URI required to enable the network to locate the device’s intent information may be stored on the device so that it can be sent to the network during network-layer onboarding. After the device has securely connected to the network, the network can use this device communications intent information to ensure that the device sends and receives traffic only from authorized locations.

5. Maintain a record of the device’s serial number (or other uniquely identifying information) and the device’s bootstrapping information. The manufacturer will take note of the device’s ID and its bootstrapping information and store these. Eventually, when the device is sold, the manufacturer will need to provide the device’s owner with its bootstrapping information. The contents of the device’s bootstrapping information will depend on what network-layer onboarding protocol the device supports. For example:

   a. If the device runs the Wi-Fi Easy Connect protocol, its bootstrapping information is the DPP URI that is associated with its private key.

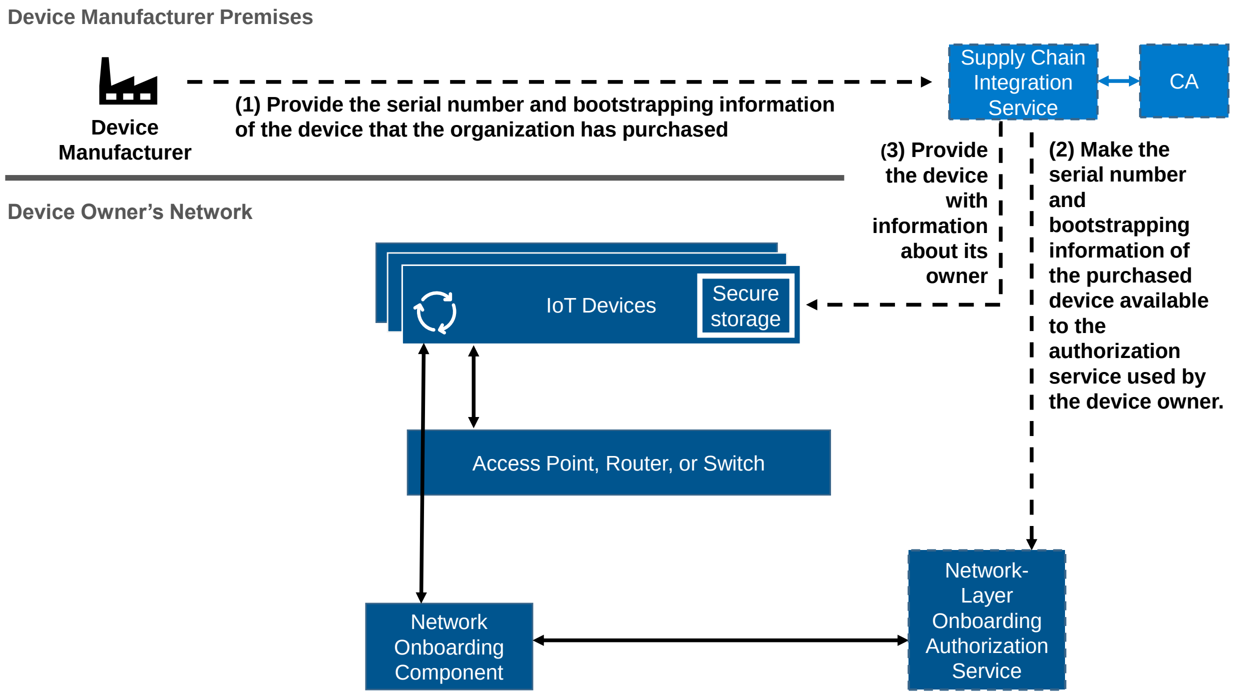
   b. If the device runs the BRSKI protocol, its bootstrapping information is its 802.1AR certificate.

4.2 Device Ownership and Bootstrapping Information Transfer Process

Figure 4-3 depicts the activities that are performed to transfer device bootstrapping information from the device manufacturer to the device owner, as well as to transfer device ownership information to the device itself, if appropriate. A high-level summary of these activities is described in the steps labeled A, B, and C.

The figure uses two colors. The dark-blue components are those used in the network-layer onboarding process. They are the same components as those depicted in the trusted network-layer onboarding process diagram provided in Figure 4-4. The light-blue components and their accompanying steps depict the portion of the diagram that is specific to device ownership and bootstrapping information transfer activities.

Figure 4-3 Device Ownership and Bootstrapping Information Transfer Process


Device Manufacturer Premises
Device Manufacturer
(1) Provide the serial number and bootstrapping information of the device that the organization has purchased
Supply Chain Integration Service
CA
(3) Provide the device with information about its owner
(2) Make the serial number and bootstrapping information of the purchased device available to the authorization service used by the device owner.
Device Owner's Network
IoT Devices
Secure storage
Access Point, Router, or Switch
Network Onboarding Component
Network-Layer Onboarding Authorization Service

These steps are as follows:

1. The device manufacturer makes the device serial number, bootstrapping information, and ownership information available so that the organization or individual who has purchased the device will have the device's serial number and bootstrapping information, and the device itself can be informed of who its owner is. In Figure 4-3, the manufacturer is shown sending this information to the supply chain integration service, which ensures that the necessary information ultimately reaches the device owner's authorization service as well as the device itself, if appropriate. (This description of the process is deliberately simple in order to enable it to be general enough that it applies to a variety of network-layer onboarding protocols.) In reality, the supply chain integration service mechanism for forwarding this bootstrapping information from the manufacturer to the owner may take many forms. For example, when BRSKI is used, the manufacturer sends the device serial number and bootstrapping information to a MASA that both the device and its owner trust. When other network-layer onboarding protocols are used, the device manufacturer may provide the device owner with this bootstrapping information directly by uploading this information to the owner's portion of a

trusted cloud. Such a mechanism is useful for the case in which the owner is a large enterprise that has made a bulk purchase of many IoT devices. In this case, the manufacturer can upload the information for hundreds or thousands of IoT devices to the supply chain integration service at once. We call this the enterprise use case. Alternatively, the device manufacturer may provide this information to the device owner indirectly by including it on or in the packaging of an IoT device that is sold at retail. We call this the consumer use case.

The contents of the device bootstrapping information will also vary according to the network-layer onboarding protocol that the device supports. For example, if the device supports the Wi-Fi Easy Connect network-layer onboarding protocol, the bootstrapping information will consist of the device's DPP URI. If the device supports the BRSKI network-layer onboarding protocol, bootstrapping information will consist of the device's IDevID (i.e., its 802.1AR certificate).

2. The supply chain integration service forwards the device serial number and bootstrapping information to an authorization service that has connectivity to the network-layer onboarding component that will onboard the device (i.e., to a network-layer onboarding component that belongs either to the device owner or to an entity that the device owner has authorized to onboard the device). The network-layer onboarding component will use the device's bootstrapping information to authenticate the device and verify that it is expected and authorized to be onboarded to the network. Again, this forwarding may take many forms, e.g., enterprise use case or consumer use case, and use a variety of different mechanisms within each use case type, e.g., information moved from one location to another in the device owner's portion of a trusted cloud, information transferred via a standardized protocol operating between the MASA and the onboarding network's domain registrar, or information scanned from a QR code on device packaging using a mobile app. In the case in which BRSKI is used, a certificate authority is consulted to help validate the signature of the 802.1AR certificate that comprises the device bootstrapping information.

3. The supply chain integration service may also provide the device with information about who its owner is. Knowing who its owner is enables the device to ensure that the network that is trying to onboard it is authorized to do so, because it is assumed that if a network owns a device, it is authorized to onboard it. The mechanisms for providing the device with assurance that the network that is trying to onboard it is authorized to do so can take a variety of forms, depending on the network-layer onboarding protocol being used. For example, if the Wi-Fi Easy Connect protocol is being used, then if an entity is in possession of the device's public key, that entity is assumed to be authorized to onboard the device. If BRSKI is being used, the device will be provided with a signed voucher verifying that the network that is trying to onboard the device is authorized to do so. The voucher is signed by the MASA. Because the device manufacturer has installed trust anchors for the MASA onto the device, the device trusts the MASA. It is also able to verify the MASA's signature.

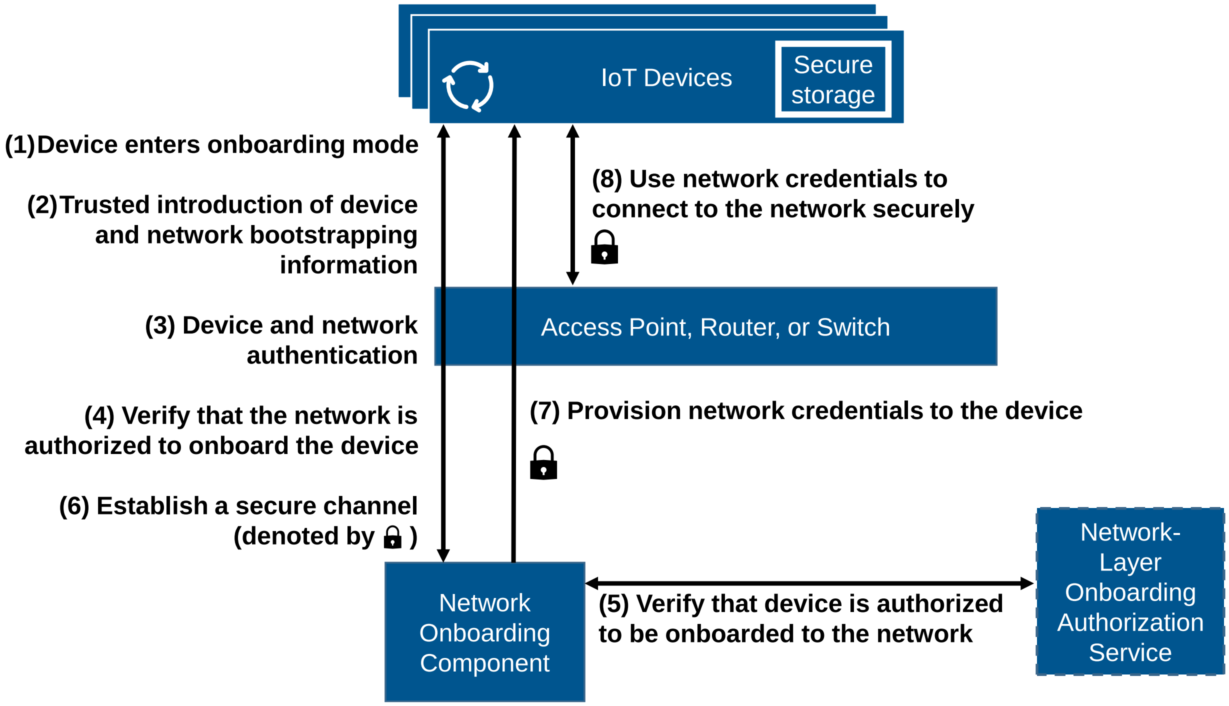
   (Note: In this document, for the sake of simplicity, we often refer to the network that is authorized to onboard a device as the device owner's network. In reality, it may not always be the case that the device's owner also owns the network to which the device is being onboarded. While it is assumed that a network that owns a device is authorized to onboard it, and the device and the onboarding network are often owned by the same entity, common ownership is not a requirement. The network that is onboarding a device does not have to be the owner of that device. The network owner may permit devices that it does not own to be onboarded to the network. In order for such a device to be onboarded, the network owner must be in possession of the device's bootstrapping information. By accepting the bootstrapping information, the network owner is implicitly authorizing the device to be onboarded to its network. Conversely, a device may permit itself to be onboarded to a network that is not owned by the device's owner. A device owner that wants to authorize a network to onboard the device needs to ensure that the device trusts the onboarding network. The specific mechanism for accomplishing this will vary according to the network-layer onboarding protocol being used. When the Wi-Fi Easy Connect protocol is being used, simply providing the network with the device's public key is sufficient to authorize the network to onboard the device. When BRSKI is being used, the voucher that the MASA provides to the device must authorize the network to onboard it.)

Authentication of the network by the device may also take a variety of forms. These may range from simply trusting the person who is onboarding the device to onboard it to the correct network, to providing the IoT device with the network's public key.

4.3 Trusted Network-Layer Onboarding Process

Figure 4-4 depicts the trusted network-layer onboarding process in more detail. It shows the interactions that occur between the network-layer onboarding component and the IoT device to mutually authenticate, confirm that the device is authorized to be onboarded to the network, confirm that the network is authorized to onboard the device, establish a secure channel, and provision the device with its network credentials.

Figure 4-4 Trusted Network-Layer Onboarding Process


IoT Devices
Secure storage
(1)Device enters onboarding mode
(2)Trusted introduction of device and network bootstrapping information
(3) Device and network authentication
(4) Verify that the network is authorized to onboard the device
(6) Establish a secure channel (denoted by 🔒)
(8) Use network credentials to connect to the network securely
Access Point, Router, or Switch
(7) Provision network credentials to the device
Network Onboarding Component
(5) Verify that device is authorized to be onboarded to the network
Network-Layer Onboarding Authorization Service

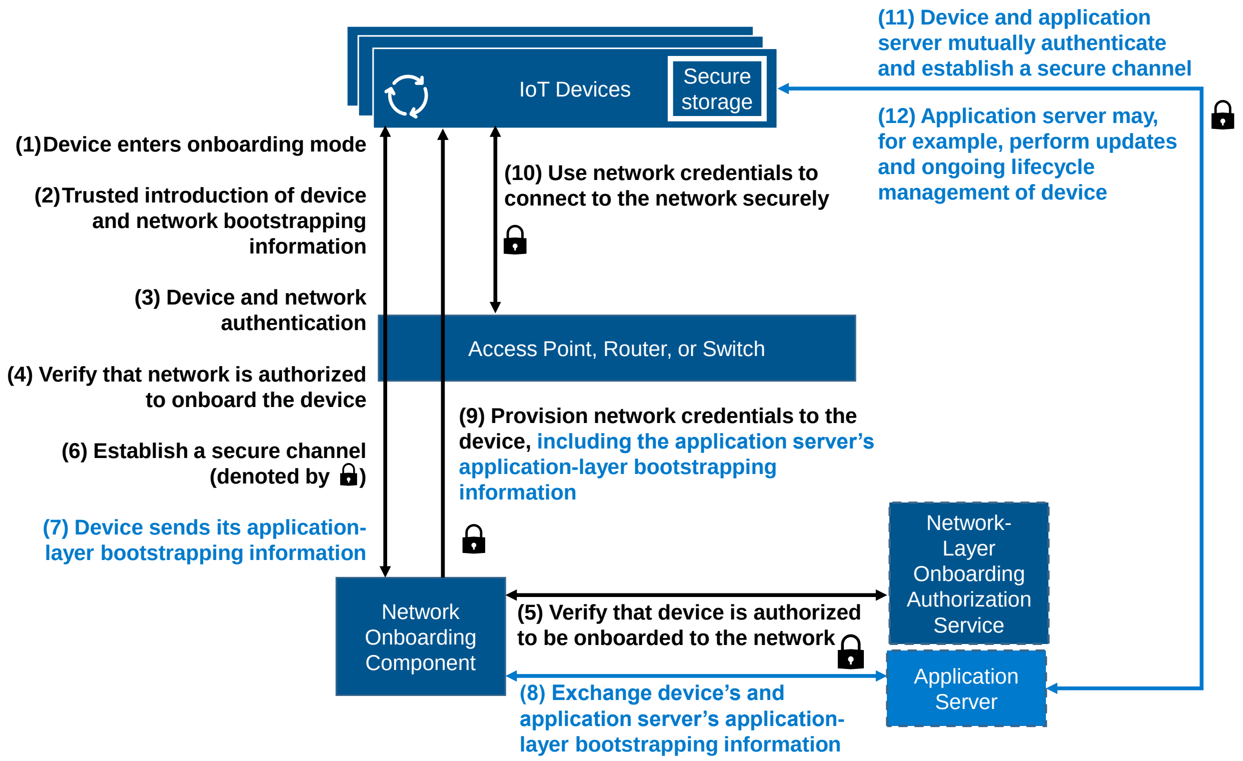
The numbered arrows in the diagram are intended to provide a high-level summary of the network-layer onboarding steps. These steps are assumed to occur after any device bootstrapping information and ownership transfer activities (as described in the previous section) that may need to be performed. The steps of the trusted network-layer onboarding process are as follows:

1. The IoT device to be onboarded is placed in onboarding mode, i.e., it is put into a state such that it is actively listening for and/or sending initial onboarding protocol messages.
2. Any required device bootstrapping information that has not already been provided to the network and any required network bootstrapping information that has not already been provided to the device are introduced in a trusted manner.
3. Using the device and network bootstrapping information that has been provided, the network authenticates the identity of the IoT device (e.g., by ensuring that the IoT device is in possession of the private key that corresponds with the public key for the device that was provided as part of the device's bootstrapping information), and the IoT device authenticates the identity of the network (e.g., by ensuring that the network is in possession of the private key that corresponds with the public key for the network that was provided as part of the network's bootstrapping information).
4. The device verifies that the network is authorized to onboard it. For example, the device may verify that it and the network are owned by the same entity, and therefore, assume that the network is authorized to onboard it.
5. The network onboarding component consults the network-layer onboarding authorization service to verify that the device is authorized to be onboarded to the network. For example, the network-layer authorization service can confirm that the device is owned by the network and is on the list of devices authorized to be onboarded.
6. A secure (i.e., encrypted) channel is established between the network onboarding component and the device.
7. The network onboarding component uses the secure channel that it has established with the device to confidentially send the device its unique network credentials.
8. The device uses its newly provisioned network credentials to establish secure connectivity to the network. The access point, router, or switch validates the device's credentials in this step. The mechanism it uses to do so varies depending on the implementation and is not depicted in Figure 4-4.

4.4 Trusted Application-Layer Onboarding Process

Figure 4-5 depicts the trusted application-layer onboarding process as enabled by the streamlined application-layer onboarding mechanism. As defined in Section 3.3.2, streamlined application-layer onboarding occurs after network-layer onboarding and depends upon and is enabled by it. The figure uses two colors. The dark-blue components are those used in the network-layer onboarding process. They and their accompanying steps (written in black font) are identical to those found in the trusted network-layer onboarding process diagram provided in Figure 4-4. The light-blue component and its accompanying steps (written in light-blue font) depict the portion of the diagram that is specific to streamlined application-layer onboarding.

Figure 4-5 Trusted Streamlined Application-Layer Onboarding Process


IoT Devices
Secure storage
(11) Device and application server mutually authenticate and establish a secure channel
(12) Application server may, for example, perform updates and ongoing lifecycle management of device
(1)Device enters onboarding mode
(2)Trusted introduction of device and network bootstrapping information
(3) Device and network authentication
(4) Verify that network is authorized to onboard the device
(6) Establish a secure channel (denoted by 🔒)
(7) Device sends its application-layer bootstrapping information
(10) Use network credentials to connect to the network securely
Access Point, Router, or Switch
(9) Provision network credentials to the device, including the application server's application-layer bootstrapping information
Network-Layer Onboarding Authorization Service
Network Onboarding Component
(5) Verify that device is authorized to be onboarded to the network
(8) Exchange device's and application server's application-layer bootstrapping information
Application Server

As is the case with Figure 4-4, the steps in this diagram are assumed to occur after any device ownership and bootstrapping information transfer activities that may need to be performed. Steps 1-6 in this figure are identical to Steps 1-6 in the trusted network-layer onboarding diagram of Figure 4-4, but steps 7 and 8 are different. With the completion of steps 1-6 in Figure 4-5, a secure channel has been established between the IoT device and the network-layer onboarding component. However, the device does not get provisioned with its network-layer credentials until step 9. To support streamlined application-layer onboarding, additional steps are required. Steps 1-12 are as follows:

1. The IoT device to be onboarded is placed in onboarding mode, i.e., it is put into a state such that it is actively listening for and/or sending initial onboarding protocol messages.

2. Any required device bootstrapping information that has not already been provided to the network and any required network bootstrapping information that has not already been provided to the device are introduced in a trusted manner.

3. Using the device and network bootstrapping information that has been provided, the network authenticates the identity of the IoT device (e.g., by ensuring that the IoT device is in possession of the private key that corresponds with the public key for the device that was provided as part of the device's bootstrapping information), and the IoT device authenticates the identity of the network (e.g., by ensuring that the network is in possession of the private key that corresponds

with the public key for the network that was provided as part of the network's bootstrapping information).

4. The device verifies that the network is authorized to onboard it. For example, the device may verify that it and the network are owned by the same entity, and therefore, assume that the network is authorized to onboard it.

5. The network onboarding component consults the network-layer onboarding authorization service to verify that the device is authorized to be onboarded to the network. For example, the network-layer authorization service can confirm that the device is owned by the network and is on the list of devices authorized to be onboarded.

6. A secure (i.e., encrypted) channel is established between the network onboarding component and the device.

7. The device sends its application-layer bootstrapping information to the network onboarding component. Just as the network required the trusted introduction of device network-layer bootstrapping information in order to enable the network to authenticate the device and ensure that the device was authorized to be network-layer onboarded, the application server requires the trusted introduction of device application-layer bootstrapping information to enable the application server to authenticate the device at the application layer and ensure that the device is authorized to be application-layer onboarded. Because this application-layer bootstrapping information is being sent over a secure channel, its integrity and confidentiality are ensured.

8. The network onboarding component forwards the device's application-layer bootstrapping information to the application server. In response, the application server provides its application-layer bootstrapping information to the network-layer onboarding component for eventual forwarding to the IoT device. The IoT device needs the application server's bootstrapping information to enable the device to authenticate the application server and ensure that it is authorized to application-layer onboard the device.

9. The network onboarding component uses the secure channel that it has established with the IoT device to confidentially send the device its unique network credentials. Along with these network credentials, the network onboarding component also sends the IoT device the application server's bootstrapping information. Because the application server's bootstrapping information is being sent over a secure channel, its integrity and confidentiality are ensured.z

10. The device uses its newly provisioned network credentials to establish secure connectivity to the network.

11. Using the device and application server application-layer bootstrapping information that has already been exchanged in a trusted manner, the application server authenticates the identity of the IoT device and the IoT device authenticates the identity of the application server. Then they establish a secure (i.e., encrypted) channel.

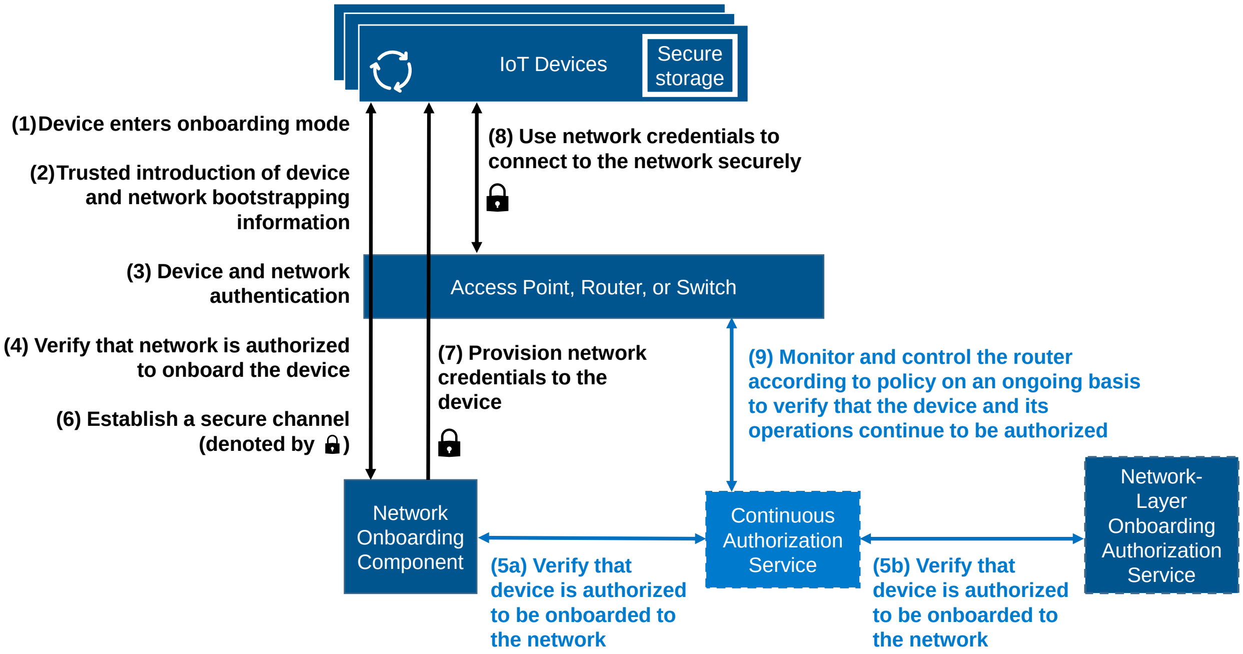
12. The application server application layer onboards the IoT device. This application-layer onboarding process may take a variety of forms. For example, the application server may download an application to the device for the device to execute. It may associate the device

with a trusted lifecycle management service that performs ongoing updates of the IoT device to patch it as needed to ensure that the device remains compliant with policy.

4.5 Continuous Verification

Figure 4-6 depicts the steps that are performed to support continuous verification. The figure uses two colors. The light-blue component and its accompanying steps (written in light-blue font) depict the portion of the diagram that is specific to continuous authorization. The dark-blue components are those used in the network-layer onboarding process. They and their accompanying steps (written in black font) are identical to those found in the trusted network-layer onboarding process diagram provided in Figure 4-4, except for step 5, *Verify that device is authorized to be onboarded to the network*.

Figure 4-6 Continuous Verification


IoT Devices
Secure storage
(1)Device enters onboarding mode
(2)Trusted introduction of device and network bootstrapping information
(3) Device and network authentication
(4) Verify that network is authorized to onboard the device
(6) Establish a secure channel (denoted by 🔒)
(8) Use network credentials to connect to the network securely
Access Point, Router, or Switch
(7) Provision network credentials to the device
(9) Monitor and control the router according to policy on an ongoing basis to verify that the device and its operations continue to be authorized
Network Onboarding Component
(5a) Verify that device is authorized to be onboarded to the network
Continuous Authorization Service
(5b) Verify that device is authorized to be onboarded to the network
Network-Layer Onboarding Authorization Service

When continuous verification is being supported, step 5 is broken into two separate steps, as shown in Figure 4-6. Instead of the network onboarding component directly contacting the network-layer onboarding authorization service to see if the device is owned by the network and on the list of devices authorized to be onboarded (as shown in the trusted network-layer onboarding architecture depicted in Figure 4-4), a set of other enterprise policies may also be applied to determine if the device is authorized to be onboarded. The application of these policies is represented by the insertion of the Continuous Authorization Service (CAS) component in the middle of the exchange between the network onboarding component and the network-layer onboarding authorization service.

For example, the CAS may have received external threat information indicating that certain device types have a vulnerability. If so, when the CAS receives a request from the network-layer onboarding component to verify that a device of this type is authorized to be onboarded to the network (Step 5a), it would immediately respond to the network-layer onboarding component that the device is not authorized to be onboarded to the network. If the CAS has not received any such threat information about the device and it checks all its policies and determines that the device should be permitted to be onboarded, it will forward the request to the network-layer onboarding authorization service (Step 5b) and receive a response (Step 5b) that it will forward to the network onboarding component (Step 5a).

As depicted by Step 9, the CAS also continues to operate after the device connects to the network and executes its application. The CAS performs asynchronous calls to the network router to monitor the device on an ongoing basis, providing policy-based verification and authorization checks on the device throughout its lifecycle.

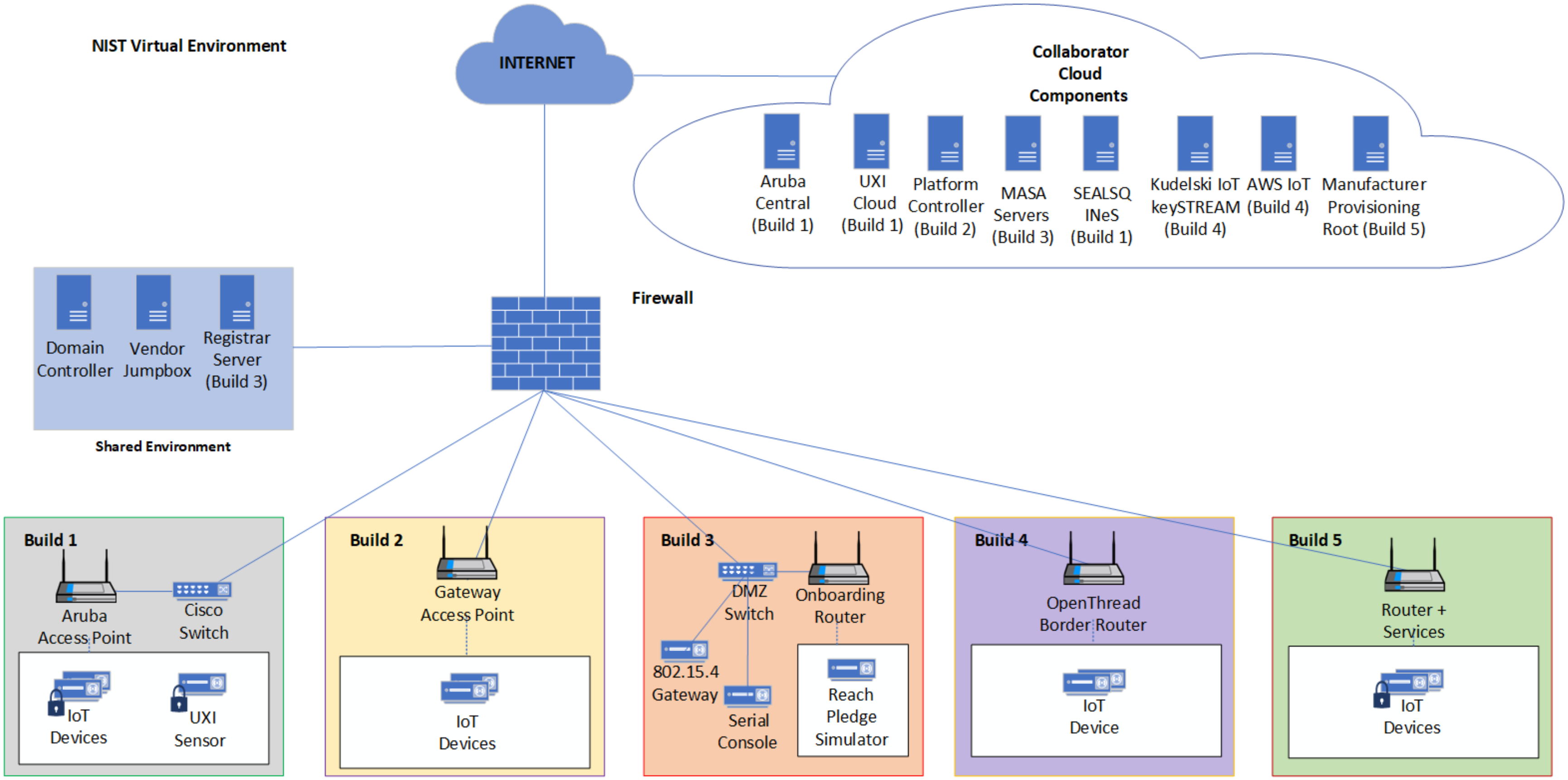
5 Laboratory Physical Architecture

Figure 5-1 depicts the high-level physical architecture of the NCCoE IoT Onboarding laboratory environment in which the five trusted IoT device network-layer onboarding project builds, and the factory provisioning builds are being implemented. The NCCoE provides virtual machine (VM) resources and physical infrastructure for the IoT Onboarding lab. As depicted, the NCCoE IoT Onboarding laboratory hosts collaborator hardware and software for the builds. The NCCoE also provides connectivity from the IoT Onboarding lab to the NIST Data Center, which provides connectivity to the internet and public IP spaces (both IPv4 and IPv6). Access to and from the NCCoE network is protected by a firewall.

Access to and from the IoT Onboarding lab is protected by a pfSense firewall, represented by the brick box icon in Figure 5-1. This firewall has both IPv4 and IPv6 (dual stack) configured. The IoT Onboarding lab network infrastructure includes a shared virtual environment that houses a domain controller and a vendor jumpbox. These components are used across builds where applicable. It also contains five independent virtual LANs, each of which houses a different trusted network-layer onboarding build.

The IoT Onboarding laboratory network has access to cloud components and services provided by the collaborators, all of which are available via the internet. These components and services include Aruba Central and the UXI Cloud (Build 1), SEALSQ INeS (Build 1), Platform Controller (Build 2), a MASA server (Build 3), Kudelski IoT keySTREAM application-layer onboarding service and AWS IoT (Build 4), and a Manufacturer Provisioning Root (Build 5).

1435 **Figure 5-1 NCCoE IoT Onboarding Laboratory Physical Architecture**



All five network-layer onboarding laboratory environments, as depicted in the diagram, have been installed:

- The Build 1 (i.e., the Wi-Fi Easy Connect, Aruba/HPE build) network infrastructure within the NCCoE lab consists of two components: the Aruba Access Point and the Cisco Switch. Build 1 also requires support from Aruba Central for network-layer onboarding and the UXI Cloud for application-layer onboarding. These components are in the cloud and accessed via the internet. The IoT devices that are onboarded using Build 1 include the UXI Sensor and the Raspberry Pi.
- The Build 2 (i.e., the Wi-Fi Easy Connect, CableLabs, OCF build) network infrastructure within the NCCoE lab consists of a single component: the Gateway Access Point. Build 2 requires support from the Platform Controller, which also hosts the IoTivity Cloud Service. The IoT devices that are onboarded using Build 2 include three Raspberry Pis.
- The Build 3 (i.e., the BRSKI, Sandelman Software Works build) network infrastructure components within the NCCoE lab include a Wi-Fi capable home router (including Join Proxy), a DMZ switch (for management), and an ESP32A Xtensa board acting as a Wi-Fi IoT device, as well as an nRF52840 board acting as an IEEE 802.15.4 device. A management system on a BeagleBone Green serves as a serial console. A registrar server has been deployed as a virtual appliance on the NCCoE private cloud system. Build 3 also requires support from a MASA server which is accessed via the internet. In addition, a Raspberry Pi 3 provides an ethernet/802.15.4 gateway, as well as a test platform.
- The Build 4 (i.e., the Thread, Silicon Labs, Kudelski IoT build) network infrastructure components within the NCCoE lab include an Open Thread Border Router, which is implemented using a Raspberry Pi, and a Silicon Labs Gecko Wireless Starter Kit, which acts as an 802.15.4 antenna. Build 4 also requires support from the Kudelski IoT keySTREAM service, which is in the cloud and accessed via the internet. The IoT device that is onboarded in Build 4 is the Silicon Labs Dev Kit (BRD2601A) with an EFR32MG24 System-on-Chip (SoC). The application service to which it onboards is AWS IoT.
- The Build 5 (i.e., the BRSKI over Wi-Fi, NquiringMinds build) includes 2 Raspberry Pi 4Bs running a Linux operating system. One Raspberry Pi acts as the pledge (or IoT Device) with an Infineon TPM connected. The other acts as the router, registrar and MASA all in one device. This build uses the open source TrustNetZ distribution, from which the entire build can be replicated easily. The TrustNetZ distribution includes source code for the IoT device, the router, the access point, the network onboarding component, the policy engine, the manufacturer services, the registrar and a demo application server. TrustNetZ makes use of NquiringMinds tdx Volt to issue and validate verifiable credentials.
- The BRSKI factory provisioning build is deployed in the Build 5 environment. The IoT device in this build is a Raspberry Pi equipped with an Infineon Optiga SLB 9670 TPM 2.0, which gets provisioned with birth credentials (i.e., a public/private key pair and an IDevID). The BRSKI factory provisioning build also uses an external certificate authority hosted on the premises of NquiringMinds to provide the device certificate signing service.
- The Wi-Fi Easy Connect factory provisioning build is deployed in the Build 1 environment. Its IoT devices are Raspberry Pis equipped with a SEALSQ VaultIC Secure Element, which gets provisioned with a DPP URI. The Secure Element can also be provisioned with an IDevID certificate signed by the SEALSQ INeS certification authority, which is independent of the DPP URI. Code for performing the factory provisioning is stored on an SD card.

Information regarding the physical architecture of all builds, their related collaborators' cloud components, and the shared environment, as well as the baseline software running on these physical architectures, are described in the subsections below. Table 5-1 summarizes the builds that were implemented and provides links to the appendices where each is described in detail.

Table 5-1 Build 1 Products and Technologies

Build	Network-Layer Protocols	Build Champions	Link to Details
Onboarding Builds			
Build 1	Wi-Fi Easy Connect	Aruba/HPE	Appendix C
Build 2	Wi-Fi Easy Connect	CableLabs and OCF	Appendix D
Build 3	BRSKI	Sandelman Software Works	Appendix E
Build 4	Thread	Silicon Labs and Kudelski IoT	Appendix F
Build 5	BRSKI over Wi-Fi	NquiringMinds	Appendix G
Factory Provisioning Builds			
BRSKI with Build 5	BRSKI over WIFI	SEALSQ and NquiringMinds	Appendix H.3
Wi-Fi Easy Connect with Build 1	Wi-Fi Easy Connect	SEALSQ and Aruba/HPE	Appendix H.4

5.1 Shared Environment

The NCCoE IoT Onboarding laboratory contains a shared environment to host several baseline services in support of the builds. These baseline services supported configuration and integration work in each of the builds and allowed collaborators to work together throughout the build process. This shared environment is contained in its own network segment, with access to/from the rest of the lab environment closely controlled. In addition, each of the systems in the shared environment is hardened with baseline configurations.

5.1.1 Domain Controller

The Domain Controller provides Active Directory and Domain Name System (DNS) services supporting network access and access control in the lab. It runs on Windows Server 2019.

5.1.2 Jumpbox

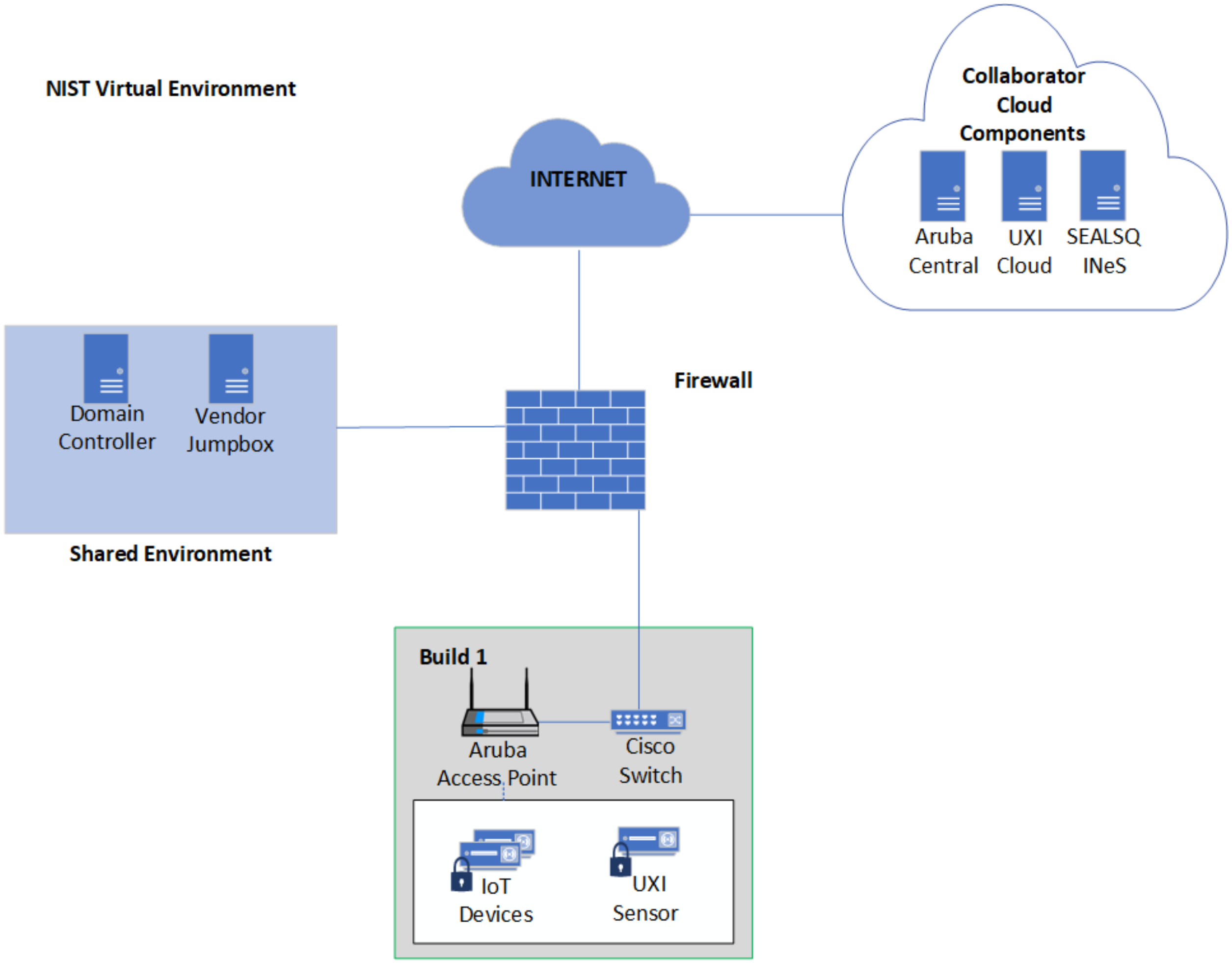
The jumpbox provides secure remote access and management to authorized collaborators on each of the builds. It runs on Windows Server 2019.

5.2 Build 1 (Wi-Fi Easy Connect, Aruba/HPE) Physical Architecture

Figure 5-2 is a view of the high-level physical architecture of Build 1 in the NCCoE IoT Onboarding laboratory. The build components include an Aruba Wireless Access Point, Aruba Central, UXI Cloud, a Cisco Catalyst switch, a SEALSQ INeS CMS CA, and the IoT devices to be onboarded, which include both a Raspberry Pi and a UXI sensor. Most of these components are described in Section 3.4.1 and Section 3.4.3.

- The Aruba Access Point acts as the DPP Configurator and relies on the Aruba Central cloud service for authentication and management purposes.
- Aruba Central ties together the IoT Operations, Client Insights, and Cloud Auth services to support the network-layer onboarding operations of the build. It also provides an API to support the device ownership and bootstrapping information transfer process.
- The Cisco Catalyst Switch provides Power-over-Ethernet and network connectivity to the Aruba Access Point.
- The UXI Sensor acts as an IoT device and onboards to the network via Wi-Fi Easy Connect. After network-layer onboarding, it performs independent (see Section 3.3.2) application-layer onboarding. Once it has application-layer onboarded and is operational on the network, it does passive and active monitoring of applications and services and will report outages, disruptions, and quality of service issues.
- UXI Cloud is an HPE cloud service that the UXI sensor contacts as part of the application-layer onboarding process. The UXI sensor downloads a customer-specific configuration from the UXI Cloud so that the UXI sensor can learn about the customer networks and services it needs to monitor.
- The Raspberry Pi acts as an IoT device and onboards to the network via Wi-Fi Easy Connect.
- SEALSQ Certificate Authority has been integrated with Build 1 to sign network credentials that are issued to IoT devices.

Figure 5-2 Physical Architecture of Build 1

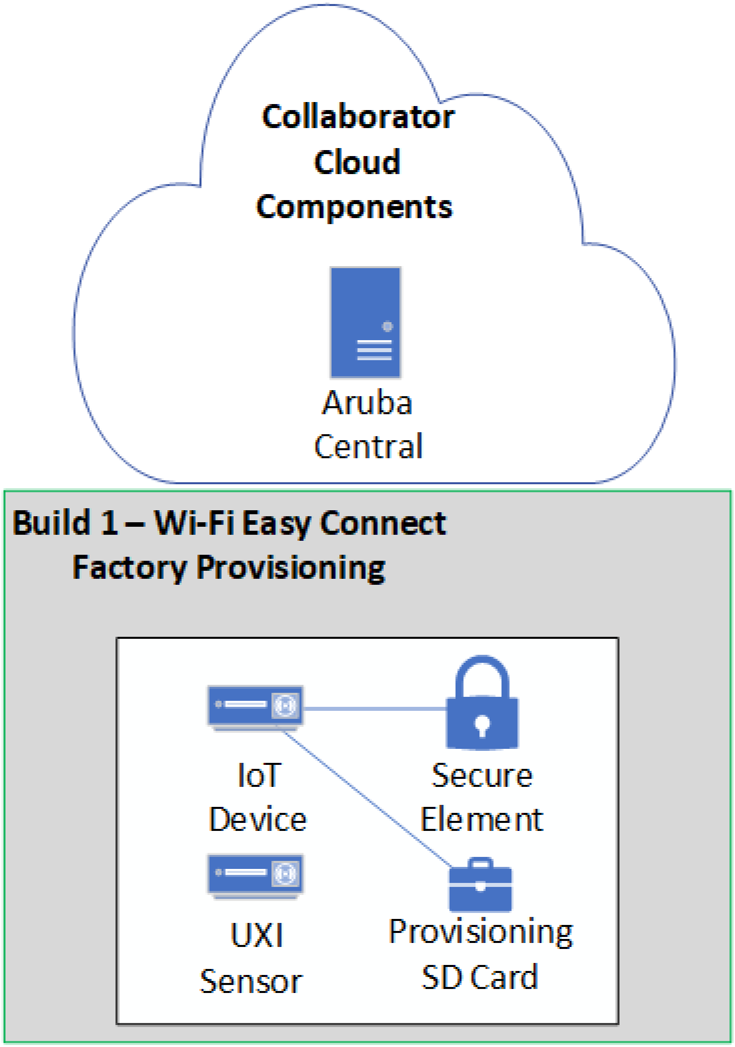


5.2.1 Wi-Fi Easy Connect Factory Provisioning Build Physical Architecture

Figure 5-3 is a view of the high-level physical architecture of the Wi-Fi Easy Connect Factory Provisioning Build in the NCCoE IoT Onboarding laboratory. The build components include the IoT device, an SD card with factory provisioning code on it, and a Secure Element. See Appendix H.4 for additional details on the Wi-Fi Easy Connect Factory Provisioning Build.

- A UXI sensor.
- The IoT Device is a Raspberry Pi.
- The Secure Element is a SEALSQ VaultIC Secure Element and is interfaced with the Raspberry Pi. The Secure Element both generates and stores the key material necessary to support the DPP URI during the Factory Provisioning Process.
- An SD card with factory provisioning code.
- Aruba Central provides an API to ingest the DPP URI in support of the device ownership and bootstrapping information transfer process.

Figure 5-3 Physical Architecture of Wi-Fi Easy Connect Factory Provisioning Build

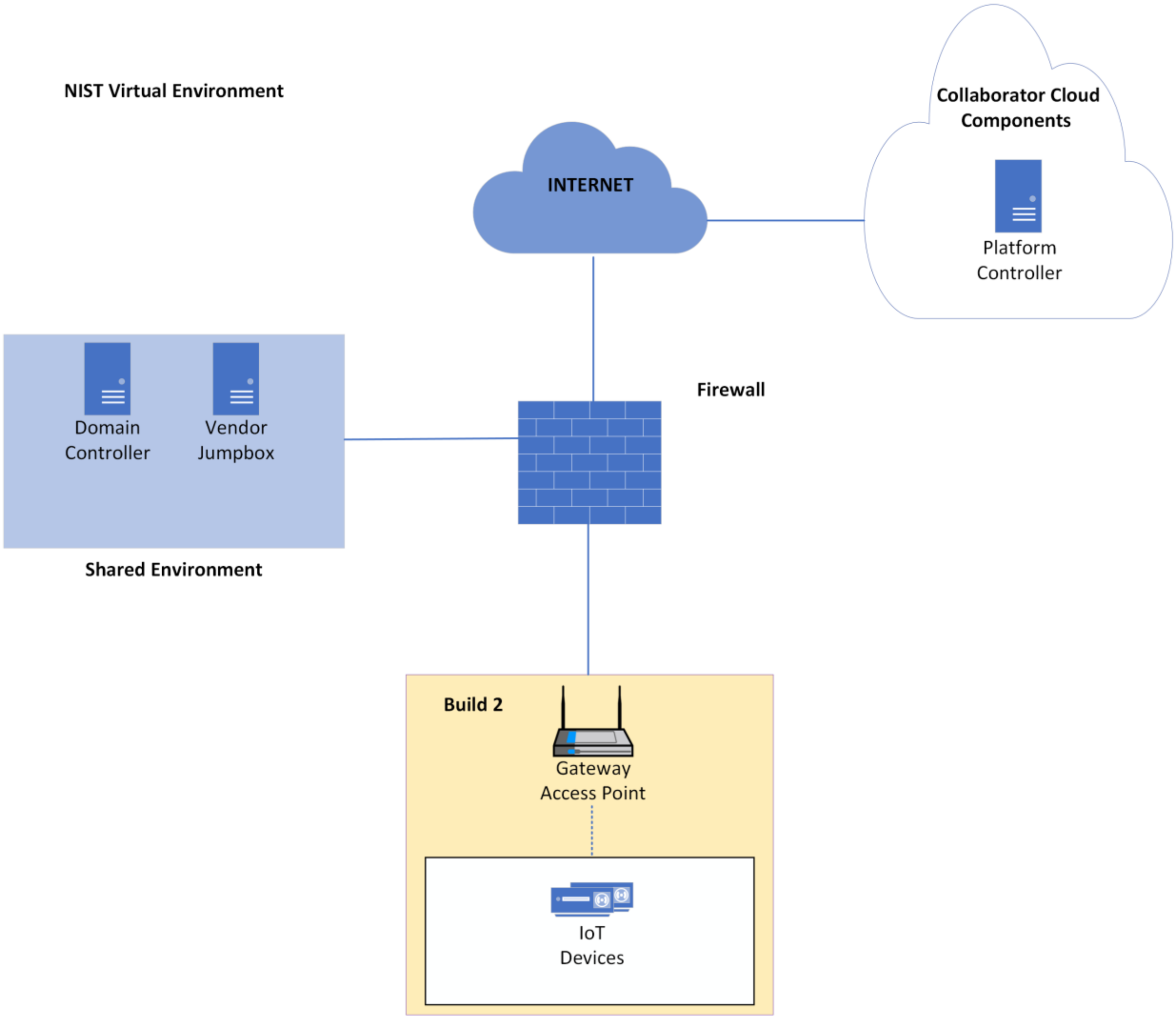

Collaborator Cloud Components
Aruba Central
Build 1 – Wi-Fi Easy Connect Factory Provisioning
IoT Device
Secure Element
UXI Sensor
Provisioning SD Card

5.3 Build 2 (Wi-Fi Easy Connect, CableLabs, OCF) Physical Architecture

Figure 5-3 is a view of the high-level physical architecture of Build 2 in the NCCoE IoT Onboarding laboratory. The Build 2 components include the Gateway Access Point, three IoT devices, and the Platform Controller, which hosts the application-layer IoTivity service.

- The Gateway Access Point acts as the Custom Connectivity Gateway Agent described in Section 3.4.2.2 and controls all network-layer onboarding activity within the network. It also hosts OCF IoTivity functions, such as the OCF OBT and the OCF Diplomat.
- The Platform Controller described in Section 3.4.2.1 provides management capabilities for the Custom Connectivity Gateway Agent. It also hosts the application-layer IoTivity service for the IoT devices as described in Section 3.4.8.1.
- The IoT devices serve as reference clients, as described in Section 3.4.2.3. They run OCF reference implementations. The IoT devices are onboarded to the network and complete both application-layer and network-layer onboarding.

Figure 5-4 Physical Architecture of Build 2



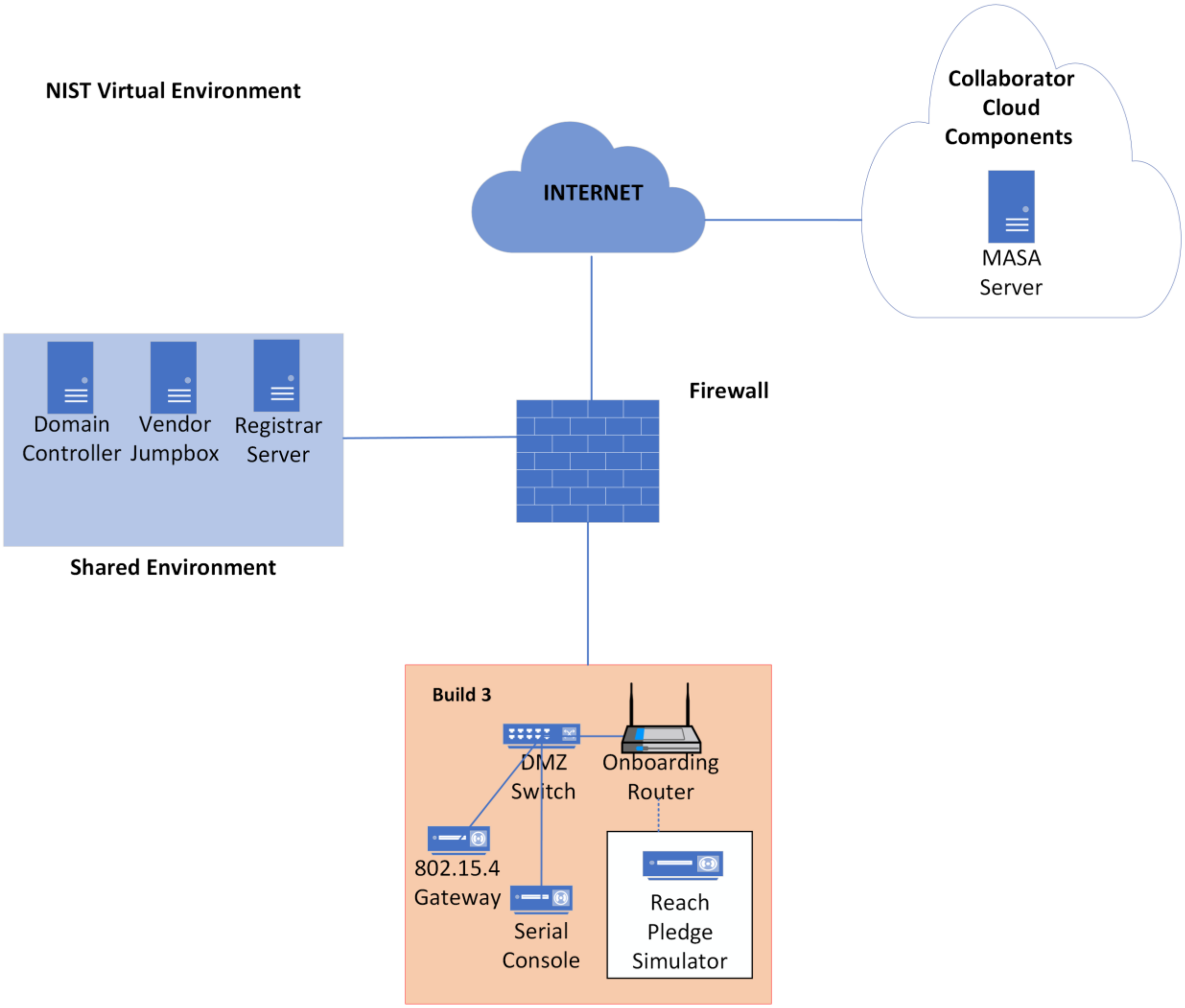
5.4 Build 3 (BRSKI, Sandelman Software Works) Physical Architecture

Figure 5-4 is a view of the high-level physical architecture of Build 3 in the NCCoE IoT Onboarding laboratory. The Build 3 components include the onboarding router, a Registrar Server, a MASA server, a DMZ switch, IoT devices, a serial console, and an 802.15.4 gateway.

- The onboarding router is a Turris MOX router running OpenWRT. The onboarding router quarantines the IoT devices until they complete the BRSKI onboarding process.
- The owner's Registrar Server hosts the Minerva Fountain Join Registrar Coordinator application running in a virtual machine. The Registrar Server determines whether or not a device meets the criteria to join the network.
- The MASA server for this build is a Minerva Highway MASA server as outlined in Section 3.4.9.1. The role of the MASA server is to receive the voucher-request from the Registrar Server and confirm that the Registrar Server has the right to own the device.

- The DMZ switch is a basic Netgear switch that segments the build from the rest of the lab.
- The IoT devices include an ESP32 Xtensa device with Wi-Fi that will be tested with FreeRTOS and RIOT-OS, a Raspberry Pi 3 running Raspbian 11, and an nRF52840 with an 802.15.4 radio that is running RIOT-OS. The IoT devices are currently not used in the build but will serve as clients to be onboarded onto the network in a future implementation of the build.
- The Sandelman Software Works Reach Pledge Simulator is the device that is onboarded to the network in the current build.
- The serial console is a BeagleBone Green with an attached USB hub. The serial console is used to access the IoT devices for diagnostic purposes. It also provides power and power control for USB-powered devices.
- The 802.15.4 gateway is integrated into the Raspberry Pi 3 via an OpenMote daughter card. This gateway will serve to onboard one of the IoT devices in a future implementation of this build.

Figure 5-5 Physical Architecture of Build 3

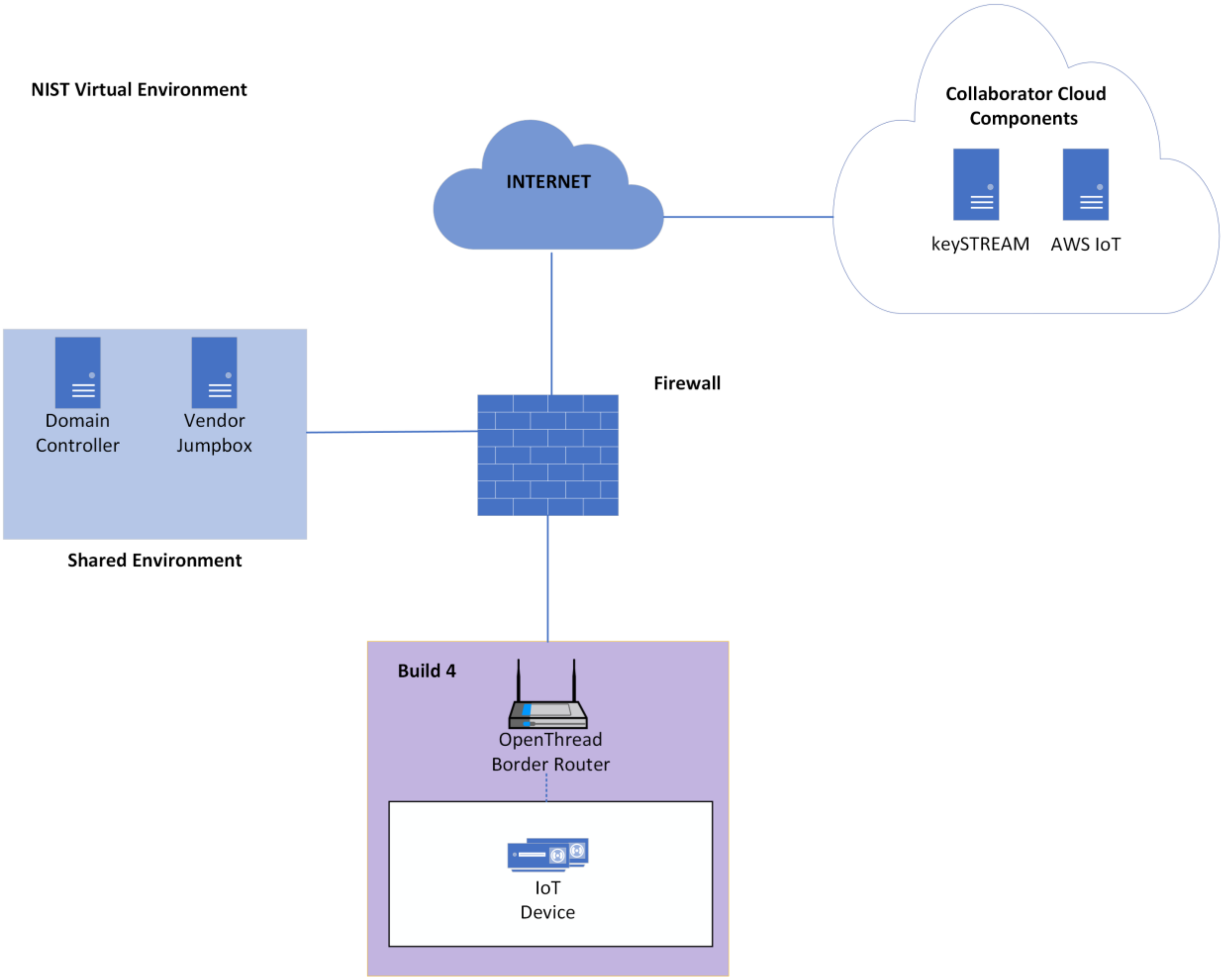


5.5 Build 4 (Thread, Silicon Labs, Kudelski IoT) Physical Architecture

Figure 5-6 is a view of the high-level physical architecture of Build 4 in the NCCoE IoT Onboarding laboratory. The Build 4 components include a keySTREAM server, an AWS IoT server, an OpenThread Border Router, and a Thread IoT device.

- The keySTREAM server described in Section 3.4.5.1 is the application layer onboarding service provided by Kudelski IoT. The IoT device will authenticate to keySTREAM using a Silicon Labs chip birth certificate and private key and leveraging Silicon Labs' Secure Engine in the EFR32MG24 chipset ("Secure Vault(TM) High" which is security certified Platform Security Architecture (PSA)/Security Evaluation Standard for IoT Platforms (SESIP) Level 3 to protect that birth identity with Secure Boot, Secure Debug, and physically unclonable function (PUF) wrapped key storage and hardware tamper protection).
- The AWS IoT server provides the MQTT test client for the trusted application-layer onboarding. The Proof of Possession Certificate is provisioned for the device using a registration code from the AWS server.
- The OpenThread Border Router is run on a Raspberry Pi 3B and serves as the Thread Commissioner and Leader. It communicates with the IoT device by means of a Silicon Labs Gecko Wireless Devkit which serves as the 802.15.4 antenna for the build.
- The IoT Device in this build is a Silicon Labs Thunderboard (BRD2601A) containing the EFR32MG24Bx 15.4 SoC with Secure Vault (TM) High running the Thread protocol. It serves as the child node on the Thread network and is onboarded onto AWS IoT Core using credentials provisioned from the Kudelski keySTREAM service.

Figure 5-6 Physical Architecture of Build 4



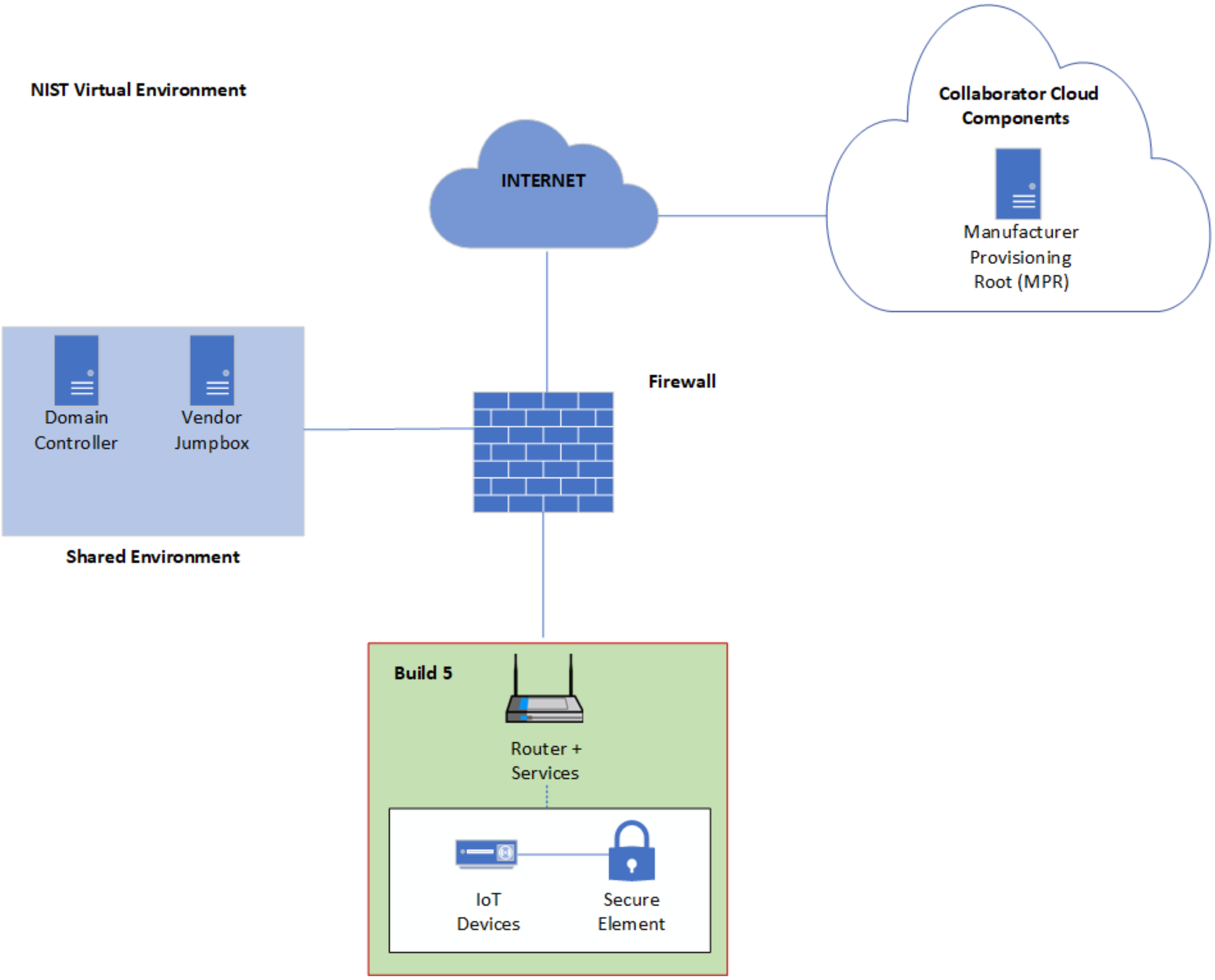
5.6 Build 5 (BRSKI, NquiringMinds) Physical Architecture

Figure 5-6 is a view of the high-level physical architecture of Build 5 in the NCCoE IoT Onboarding laboratory. The Build 5 components include a MASA, Registrar, Router Access Point, an IoT Device, and a Secure Element:

- A Raspberry Pi 4B serves as the MASA, Registrar and Router Access Point for the local network. The role of the MASA is to receive the voucher-request from the Registrar and confirm that the Registrar has the right to own the device. The registrar self-signs credentials, namely the Local Device Identifier (LDevID), issued to the IoT devices. The pledge (IoT device) gets its IDevID certificate for device identity from the Manufacturer Provisioning Root (MPR) server during the factory provisioning process, it can be assumed to be present on the device at the point of onboarding. The Registrar determines whether or not a device meets the criteria to join the network. The router access point runs an open and closed BRSKI network, the closed BRSKI network may only be accessed through secure onboarding, which is performed via the open network. The registrar leverages a local tdx Volt instance to sign and verify verifiable credentials.
- Raspberry Pi 4Bs act as IoT Devices (pledges) for this build.

- The Secure Element is an Infineon Optiga SLB 9670 TPM 2.0 Secure Element, and both generates and stores the key material necessary to support the IDevID certificate during the Factory Provisioning Process, as well as the onboarding process to request the voucher from the MASA via the registrar and the request to the registrar to sign the LDevID. The system can also be configured to use a SEALSQ VaultIC408 secure element. See Appendix H.3 for additional details on the BRSKI factory provisioning builds.

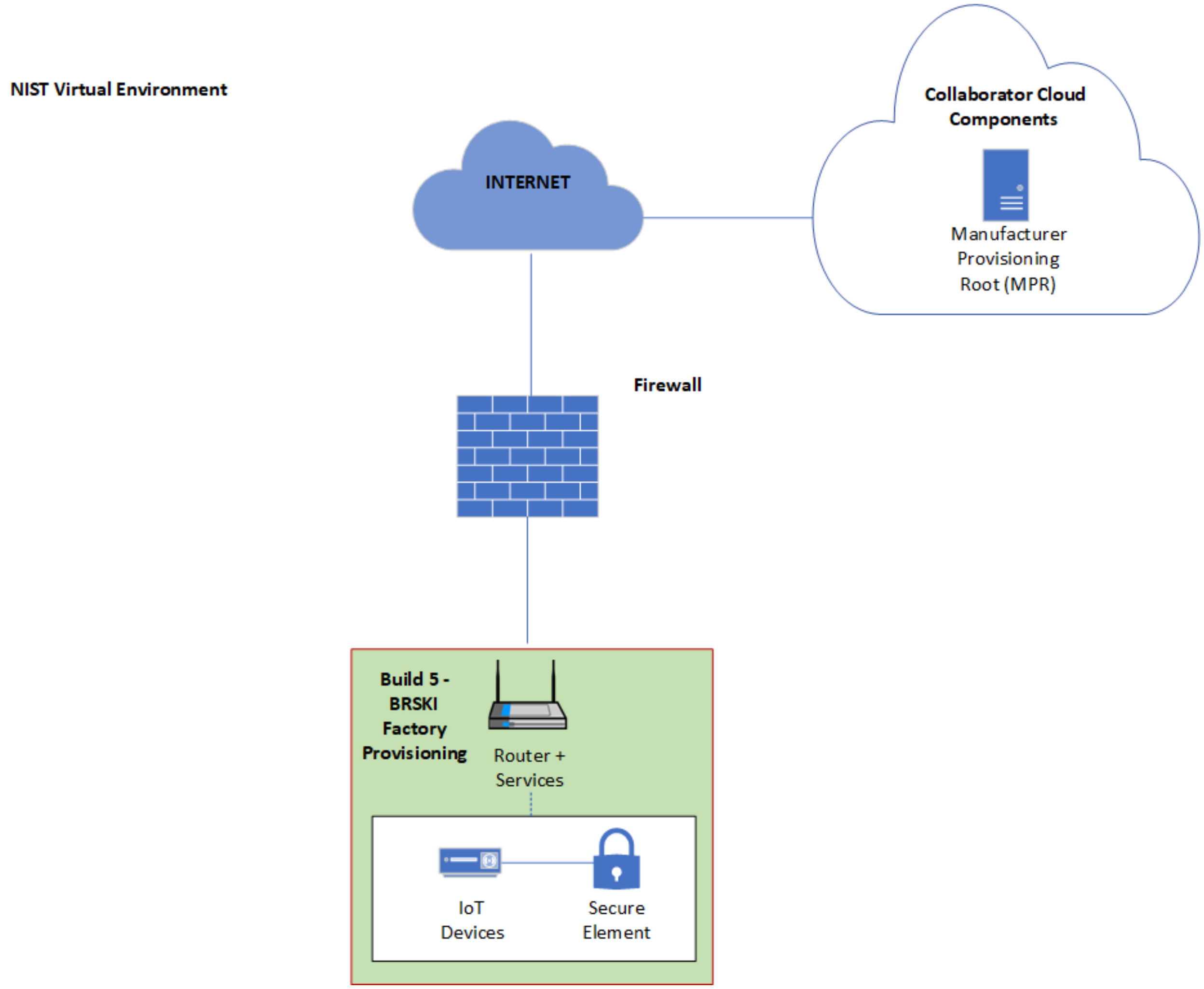
Figure 5-7 Physical Architecture of Build 5



5.6.1 BRSKI Factory Provisioning Build Physical Architecture

Figure 5-8 is a view of the high-level physical architecture of the BRSKI Factory Provisioning Build in the NCCoE IoT Onboarding laboratory. This build uses the same IoT device as Build 5: a Raspberry Pi integrated with an Infineon Optiga SLB 9670 TPM 2.0 Secure Element. The factory provisioning code is hosted on an SD card. When a provisioning event is triggered the IoT device will attempt a connection to a Manufacturer Provisioning Root (MPR) server that sits in the cloud and acts as the certification authority. It signs the IDevID (X.509) certificate, which is then passed back to the IoT device for installation. As in Build 5, the Router + Services hosts a MASA, which is given device identity information in order to verify voucher requests during the BRKSI process. See Appendix H.3 for additional details on the BRSKI factory provisioning builds.

Figure 5-8 Physical Architecture of BRSKI Factory Provisioning Build


NIST Virtual Environment
INTERNET
Collaborator Cloud Components
Manufacturer Provisioning Root (MPR)
Firewall
Build 5 - BRSKI Factory Provisioning
Router + Services
IoT Devices
Secure Element

6 General Findings

6.1 Wi-Fi Easy Connect

The Wi-Fi Easy Connect solution that was demonstrated in Build 1 and Build 2 supports trusted network-layer onboarding in a manner that is secure, efficient, and flexible enough to meet the needs of various use cases. It is simple enough to be used by consumers, who typically do not have specialized technical knowledge. In addition, to meet the needs of enterprises, it may be used to onboard a large number of devices quickly. Builds 1 and 2 are implementations of this protocol, and they are interoperable: IoT devices that were provisioned for use with Build 1 were able to be onboarded onto the network using Build 2, and IoT devices that were provisioned for use with Build 2 were able to be onboarded onto the network using Build 1.

6.1.1 Mutual Authentication

Although DPP is designed to support authentication of the network by the IoT device as well as authentication of the device by the network, the Wi-Fi Easy Connect solutions that were demonstrated in builds 1 and 2 do not demonstrate mutual authentication at the network layer. They only support authentication of the device. In order to authenticate the network, the device needs to be provided with the DPP URI for the network configurator, which means that the device has to have a functional user interface so that the DPP URI can be input into it. The devices being used in builds 1 and 2 do not have user interfaces.

6.1.2 Mutual Authorization

When using DPP, device authorization is based on possession of the device's DPP URI. When the device is acquired, its DPP URI is provided to the device owner. A trusted administrator of the owner's network is assumed to approve addition of the device's DPP URI to the database or cloud service where the DPP URIs of authorized devices are stored. During the onboarding process, the fact that the owning network is in possession of the device's DPP URI indicates to the network that the device is authorized to join it.

DPP supports network authorization using the Resurrecting Duckling security model [13]. Although the device cannot cryptographically verify that the network is authorized to onboard it, the fact that the network possesses the device's public key is understood by the device to implicitly authorize the network to onboard the device. The assumption is that an unauthorized network would not have possession of the device and so would not be able to obtain the device's public key. While this assurance of authorization is not cryptographic, it does provide some level of assurance that the "wrong" network won't onboard it.

6.1.3 Secure Storage

The UXI sensor used in Build 1 has a TPM where the device's birth credential and private key are stored, providing a secure root of trust. However, the lack of secure storage on some of the other IoT devices (e.g., the Raspberry Pis) used to demonstrate onboarding in Build 2 is a current weakness. Ensuring that the confidentiality of a device's birth, network, and other credentials is protected while stored on the device is an essential aspect of ensuring the security of the network-layer onboarding process, the device, and the network itself. To fully demonstrate trusted network-layer onboarding, devices with secure storage should be used in the future whenever possible.

6.2 BRSKI

The BRSKI solution that is demonstrated in Build 3 supports trusted network-layer onboarding in a manner that is secure, efficient, and able to meet the needs of enterprises. It may be used to onboard a large number of devices quickly onto a wired network. This BRSKI build is based on IETF RFC 8995 [7]. The build has a reliance on the manufacturer to provision keys for the onboarding device and has a reliance on a cloud-based service for the MASA server. The BRSKI solution that is demonstrated in Build 5 provides similar trusted functionality for onboarding devices onto a Wi-Fi network. This BRSKI build is based on an IETF individual draft describing how to run BRSKI over IEEE 802.11 [10].

6.2.1 Reliance on the Device Manufacturer

Organizations implementing BRSKI (whether wired or over Wi-Fi) should be aware of the reliance that they will have on the IoT device manufacturer in properly and securely provisioning their devices. If keys become compromised, attackers may be able to onboard their own devices to the network, revoke certificates to prevent legitimate devices from being onboarded, or onboard devices belonging to others onto the attacker's network using the attacker's MASA. These concerns are addressed in depth in RFC 8995 section 11.6. If a device manufacturer goes out of business or otherwise shuts down their MASA servers, the onboarding services for their devices will no longer function.

During operation, onboarding services may become temporarily unavailable for a number of reasons. In the case of a DoS attack on the MASA, server maintenance, or other outage on the part of the manufacturer, an organization will not be able to access the MASA. These concerns are addressed in depth in RFC 8995 section 11.1.

6.2.2 Mutual Authentication

BRSKI supports authentication of the IoT device by the network as well as authentication of the network by the IoT device. The Registrar authenticates the device when it receives the IDevID from the device. The MASA confirms that the Registrar is the legitimate owner or authorized onboarder of the device and issues a voucher. The device is able to authenticate the network using the voucher that it receives back from the MASA. This process is explained in depth in RFC 8995 section 11.5.

6.2.3 Mutual Authorization

BRSKI authorization for the IoT device is done via the voucher that is returned to the Registrar from the MASA. The voucher states which network the IoT device is authorized to join. The Registrar determines the level of access the IoT device has to the network.

6.2.4 Secure Storage

Build 5 uses a Secure Element attached to the IoT devices (e.g., Raspberry Pi devices) to store the IDevID after it is generated during the factory provisioning process (see Appendix H.3 for more details), however the LDevID is not stored on the Secure Element after network-layer onboarding is completed. The lack of secure storage on the IoT devices (e.g., the Raspberry Pi devices) used to demonstrate onboarding in Build 3 is a current weakness. Ensuring that the confidentiality of a device's birth, network, and other credentials is protected while stored on the device is an essential aspect of ensuring the security of the network-layer onboarding process, the device, and the network itself. To fully demonstrate trusted network-layer onboarding, devices with secure storage should be used in the future whenever possible.

6.3 Thread

We do not have any findings with respect to trusted network-layer onboarding using the Thread commissioning protocol. Build 4 demonstrated the connection of an IoT device to a Thread network, but not trusted onboarding of the Thread network credentials to the device. In Build 4, a passphrase is generated on the IoT device and then a person is required to enter this passphrase into the OpenThread

Border Router's (OTBR) web interface. This passphrase serves as a pre-shared key that the device uses to join the Thread network. Due to the fact that a person must be privy to this passphrase in order to provide it to the OTBR, this network-layer onboarding process is not considered to be trusted, according to the definition of trusted network-layer onboarding that we provided in Section 1.2.

After connecting to the Thread network using the passphrase, the Build 4 device was successfully able to gain access to the public IP network via a border router. This enabled the IoT device that was communicating using the Thread wireless protocol to communicate with cloud services and use them to successfully perform trusted application-layer onboarding to the AWS IoT Core.

6.4 Application-Layer Onboarding

We successfully demonstrated both:

- streamlined application-layer onboarding (to the OCF security domain in Build 2) and
- independent application-layer onboarding (to the UXI cloud in Build 1 and to the AWS IoT Core using the Kudelski keySTREAM service in Build 4).

6.4.1 Independent Application-Layer Onboarding

Support for independent application-layer onboarding requires the device manufacturer to pre-provision the device with software to support application-layer onboarding to the specific application service (e.g., the UXI cloud or the AWS IoT Core) desired. The Kudelski keySTREAM service supports the application-layer onboarding provided in Build 4. KeySTREAM is a device security management service that runs as a SaaS platform on the Amazon cloud. Build 4 relies on an integration that has been performed between Silicon Labs and Kudelski keySTREAM. KeySTREAM has integrated software libraries with the Silicon Lab EFR32MG24 (MG24) IoT device's secure vault to enable the private signing key that is associated with an application-layer certificate to be stored into the secure vault using security controls that are available on the MG24. This integration ensures that application-layer credentials can be provisioned into the vault securely such that no key material is misused or exposed.

Because the device is prepared for application-layer onboarding on behalf of a specific, pre-defined customer in Build 4 and this ownership information is sealed into device firmware, the device is permanently identified as being owned by that customer.

6.4.2 Streamline Application-Layer Onboarding

Support for streamlined application-layer onboarding does not necessarily present such a burden on the device manufacturer to provision application-layer onboarding software and/or credentials to the device at manufacturing time. If desired, the manufacturer could pre-install application-layer bootstrapping information onto the device at manufacturing time, as must be done in the independent application-layer onboarding case. Alternatively, the device manufacturer may simply ensure that the device has the capability to generate one-time application-layer bootstrapping information at runtime and use the secure exchanges inherent in trusted network-layer onboarding to support application-layer onboarding.

7 Additional Build Considerations

The Trusted Internet of Things (IoT) Device Network-Layer Onboarding and Lifecycle Management project is now complete, so no additions or changes to the existing builds are planned as part of this project effort. As trusted network-layer onboarding is increasingly adopted, however, others may wish to continue implementation efforts to develop new build capabilities or enhance existing ones, so it is worth noting potential areas of further work. Various ways in which individual builds could be enhanced are noted in the appendices that detail each build's technologies and architectures. For example, some builds could be enhanced by the addition of architectural components that they have not yet implemented, such as secure device storage; the use of an independent, third-party certificate signing authority; support for network-layer onboarding using Thread MeshCoP; support for application-layer onboarding; and support (or enhanced support) for ongoing device authorization. In addition to adding components to support these capabilities, future work could potentially involve demonstration of application-layer onboarding using the FIDO Alliance's FIDO Device Onboard (FDO) specification and/or the Connectivity Standards Alliance (CSA) MATTER specification. Other future work could involve integrating additional security mechanisms with network-layer onboarding, beginning at device boot-up and extending through all phases of the device lifecycle, to further protect the device and, by extension, the network. For example, future builds could include the capability to demonstrate the integration of trusted network-layer onboarding with zero trust-inspired capabilities such as those described in the following subsections. In addition, the scope of implementation efforts could potentially be expanded beyond the current focus on IP-based networks. While this project's goal has been to tackle what is currently implementable, the subsections that follow briefly discuss areas that could potentially be addressed by others in the future.

7.1 Network Authentication

Future builds could be designed to demonstrate network authentication in addition to device authentication as part of the network-layer onboarding process. Network authentication enables the device to verify the identity of the network that will be taking control of it prior to permitting itself to be onboarded.

7.2 Device Communications Intent

Future builds could be designed to demonstrate the use of network-layer onboarding protocols to securely transmit device communications intent information from the device to the network (i.e., to transmit this information in encrypted form with integrity protections). Secure conveyance of device communications intent information, combined with enforcement of it, would enable the build to ensure that IoT devices are constrained to sending and receiving only those communications that are explicitly required for each device to fulfill its purpose. Build 5 currently enforces device communications intent as part of its continuous assurance process. Build 5 determines device communications intent information (e.g., the device's MUD file URL) based on device type rather than conveying this information from the device to the network during onboarding.

7.3 Network Segmentation

Future builds could demonstrate the ability of the onboarding network to dynamically assign each new device that is permitted to join the network to a specific subnetwork. The router may have multiple network segments configured to which an onboarded device may be dynamically assigned. The decision regarding which segment (subnetwork) to which to assign the device could potentially be based on the device's DHCP fingerprint, other markers of the device's type, or some indication of the device's trustworthiness, subject to organizational policy.

7.4 Integration with a Lifecycle Management Service

Future builds could demonstrate trusted network-layer onboarding of a device, followed by streamlined trusted application-layer onboarding of that device to a lifecycle management application service. Such a capability would ensure that, once connected to the local network, the IoT device would automatically and securely establish an association with a trusted lifecycle management service that is designed to keep the device updated and patched on an ongoing basis.

7.5 Network Credential Renewal

Some devices may be provisioned with network credentials that are X.509 certificates and that will, therefore, eventually expire. Future build efforts could explore and demonstrate potential ways of renewing such credentials without having to reprovision the credentials to the devices.

7.6 Integration with Supply Chain Management Tools

Future work could include definition of an open, scalable supply chain integration service that can provide additional assurance of device provenance and trustworthiness automatically as part of the onboarding process. The supply chain integration service could be integrated with the authorization service to ensure that only devices whose provenance meets specific criteria and that reach a threshold level of trustworthiness will be onboarded or authorized.

7.7 Attestation

Future builds could integrate device attestation capabilities with network-layer onboarding to ensure that only IoT devices that meet specific attestation criteria are permitted to be onboarded. In addition to considering the attestation of each device as a whole, future attestation work could also focus on attestation of individual device components, so that detailed attestation could be performed for each board, integrated circuit, and software program that comprises a device.

7.8 Mutual Attestation

Future builds could implement mutual attestation of the device and its application services. In one direction, device attestation could be used to enable a high-value application service to determine whether a device should be given permission to access it. In the other direction, attestation of the application service could be used to enable the device to determine whether it should give the application service permission to access and update the device.

7.9 Behavioral Analysis

Future builds could integrate artificial intelligence (AI) and machine learning (ML)-based tools that are designed to analyze device behavior to spot anomalies or other potential signs of compromise. Any device that is flagged as a potential threat by these tools could have its network credentials invalidated to effectively evict it from the network, be quarantined, or have its interaction with other devices restricted in some way.

7.10 Device Trustworthiness Scale

Future efforts could incorporate the concept of a device trustworthiness scale in which information regarding device capabilities, secure firmware updates, the existence (or not) of a secure element for private key protection, type and version of each of the software components that comprise the device, etc., would be used as input parameters to calculate each device's trustworthiness value. Calculating such a value would essentially provide the equivalent of a background check. A history for the device could be maintained, including information about whether it has ever been compromised, if it has a known vulnerability, etc. Such a trustworthiness value could be provided as an onboarding token or integrated into the authorization service so permission to onboard to the network, or to access certain resources once joined, could be granted or denied based on historical data and trustworthiness measures.

7.11 Resource Constrained Systems

At present, onboarding solutions for technologies such as Zigbee, Z-Wave, and BLE use their own proprietary mechanisms or depend on gateways. In the future, efforts could be expanded to include onboarding in highly resource-constrained systems and non-IP systems without using gateways. Future work could include trying to perform trusted onboarding in these smaller microcontroller-constrained spaces in a standardized way with the goal of bringing more commonality across various solutions without having to rely on IP gateways.

1848 Appendix A List of Acronyms

AAA	Authentication, Authorization, and Accounting
ACL	Access Control List
AES	Advanced Encryption Standard
AI	Artificial Intelligence
AP	Access Point
API	Application Programming Interface
AWS	Amazon Web Services
BLE	Bluetooth Low Energy
BRSKI	Bootstrapping Remote Secure Key Infrastructure
BSS	Basic Service Set
CA	Certificate Authority
CAS	Continuous Authorization Service
CMS	Certificate Management System
CPU	Central Processing Unit
CRADA	Cooperative Research and Development Agreement
CRL	Certificate Revocation List
DHCP	Dynamic Host Configuration Protocol
DMZ	Demilitarized Zone
DNS	Domain Name System
DPP	Device Provisioning Protocol
DTLS	Datagram Transport Layer Security
ECC	Elliptic Curve Cryptography
ESP	(Aruba) Edge Services Platform
ESS	Extended Service Set
EST	Enrollment over Secure Transport
HPE	Hewlett Packard Enterprise
HSM	Hardware Security Module
HTTPS	Hypertext Transfer Protocol Secure

IDevID	Initial Device Identifier
IE	Information Element
IEC	International Electrotechnical Commission
IETF	Internet Engineering Task Force
IoT	Internet of Things
IP	Internet Protocol
IPsec	Internet Protocol Security
ISO	International Organization for Standardization
LAN	Local Area Network, Local Area Networking
LDevID	Local Device Identifier
LmP	Linux microPlatform
MASA	Manufacturer Authorized Signing Authority
MeshCoP	Thread Mesh Commissioning Protocol
ML	Machine Learning
mPKI	Managed Public Key Infrastructure
MUD	Manufacturer Usage Description
NAC	Network Access Control
NCCoE	National Cybersecurity Center of Excellence
NIST	National Institute of Standards and Technology
OBT	Onboarding Tool
OCF	Open Connectivity Foundation
OCSP	Online Certificate Status Protocol
OS	Operating System
OTA	Over the Air
OTBR	OpenThread Border Router
PKI	Public Key Infrastructure
PSK	Pre-Shared Key
R&D	Research & Development
RBAC	Role-Based Access Control

RCP	Radio Coprocessor
RESTful	Representational State Transfer
RFC	Request for Comments
RoT	Root of Trust
RSA	Rivest-Shamir-Adleman (public-key cryptosystem)
SaaS	Software as a Service
SE	Secure Element
SEF	Secure Element Factory
SoC	System-on-Chip
SP	Special Publication
SSID	Service Set Identifier
SSW	Sandelman Software Works
TCP	Transmission Control Protocol
TLS	Transport Layer Security
TOFU	Trust On First Use
TPM	Trusted Platform Module
URI	Uniform Resource Identifier
UXI	(Aruba) User Experience Insight
VM	Virtual Machine
WAN	Wide Area Network, Wide Area Networking
WFA	Wi-Fi Alliance
WPA2	Wi-Fi Protected Access 2
WPA3	Wi-Fi Protected Access 3

Appendix B Glossary

Application-Layer Bootstrapping Information Information that the device and an application-layer service must have in order for them to mutually authenticate and use a trusted application-layer onboarding protocol to onboard a device at the application layer. There is application-layer bootstrapping information about the device that the network must be in possession of, and application-layer bootstrapping information about the application service that the device must be in possession of. A typical example of application-layer bootstrapping information that the device must have is the public key that corresponds to the trusted application service's private key.

Application-Layer Onboarding The process of providing IoT devices with the application-layer credentials they need to establish a secure (i.e., encrypted) association with a trusted application service. This document defines two types of application-layer onboarding: independent and streamlined.

Independent Application-Layer Onboarding An application-layer onboarding process that does not rely on use of the network-layer onboarding process to transfer application-layer bootstrapping information between the device and the application service.

Network-Layer Bootstrapping Information Information that the device and the network must have in order for them to use a trusted network-layer onboarding protocol to onboard a device. There is network-layer bootstrapping information about the device that the network must be in possession of, and network-layer bootstrapping information about the network that the device must be in possession of. A typical example of device bootstrapping information that the network must have is the public key that corresponds with the device's private key.

Network-Layer Onboarding The process of providing IoT devices with the network-layer credentials and policy they need to join a network upon deployment.

Streamlined Application-Layer Onboarding An application-layer onboarding process that uses the network-layer onboarding protocol to securely transfer application-layer bootstrapping information between the device and the application service.

Trusted Network-Layer Onboarding A network-layer onboarding process that meets the following criteria:

- provides each device with unique network credentials,
- enables the device and the network to mutually authenticate,
- sends devices their network credentials over an encrypted channel,
- does not provide any person with access to the network credentials, and
- can be performed repeatedly throughout the device lifecycle to enable:
  - the device's network credentials to be securely managed and replaced as needed, and
  - the device to be securely onboarded to other networks after being repurposed or resold.

Appendix C Build 1 (Wi-Fi Easy Connect, Aruba/HPE)

C.1 Technologies

Build 1 is an implementation of network-layer onboarding that uses the Wi-Fi Easy Connect protocol. The onboarding infrastructure and related technology components for Build 1 have been provided by Aruba/HPE. IoT devices that were onboarded using Build 1 were provided by Aruba/HPE and CableLabs. The CA used for signing credentials issued to IoT devices was provided by SEALSQ, a subsidiary of WISeKey. For more information on these collaborators and the products and technologies that they contributed to this project overall, see Section 3.4.

Build 1 network onboarding infrastructure components within the NCCoE lab consist of the Aruba Access Point. Build 1 also requires support from Aruba Central and the UXI Cloud, which are accessed via the internet. IoT devices that can be network-layer onboarded using Build 1 include the Aruba/HPE UXI sensor and CableLabs Raspberry Pi. The UXI sensor also includes the Aruba UXI Application, which enables it to use independent (see Section 3.3.2) application-layer onboarding to be onboarded at the application layer as well, providing that the network to which the UXI sensor is onboarded has connectivity to the UXI Cloud via the internet. The Build 1 implementation supports the provisioning of all three types of network credentials defined in DPP:

- Connector for DPP-based network access
- Password/passphrase/PSK for WPA3/WPA2 network access
- X.509 certificates for 802.1X network access

Build 1 has been integrated with the SEALSQ CA on SEALSQ INeS CMS to enable Build 1 to obtain signed certificates from this CA when Build 1 is onboarding devices and issuing credentials for 802.1X network access. When issuing credentials for DPP and WPA3/WPA2-based network access, the configurator does not need to use a CA.

Table C-1 lists the technologies used in Build 1. It lists the products used to instantiate each component of the reference architecture and describes the security function that the component provides. The components listed are logical. They may be combined in physical form, e.g., a single piece of hardware may house a network onboarding component, a router, and a wireless access point.

Table C-1 Build 1 Products and Technologies

Component	Product	Function
Network-Layer Onboarding Component (Wi-Fi Easy Connect Configurator)	Aruba Access Point with support from Aruba Central	Runs the Wi-Fi Easy Connect network-layer onboarding protocol to interact with the IoT device to perform one-way or mutual authentication, establish a secure channel, and securely provide local network credentials to the device. If the network credential that is being provided to the device is a certificate, the onboarding component will interact with a certificate authority to sign the certificate. The configurator deployed in Build 1 supports DPP 2.0, but it is also backward compatible with DPP 1.0.

Component	Product	Function
Access Point, Router, or Switch	Aruba Access Point	Wireless access point that also serves as a router. It may get configured with per-device access control lists (ACLs) and policy when devices are onboarded.
Supply Chain Integration Service	Aruba Central	The device manufacturer provides device bootstrapping information to the HPE Cloud via the REST API that is documented in the DPP specification. Once the device is transferred to an owner, the HPE Cloud provides the device bootstrapping information (i.e., the device's DPP URI) to the device owner's private tenancy within the HPE Cloud.
Authorization Service	Cloud Auth (on Aruba Central)	The authorization service provides the configurator and router with the information needed to determine if the device is authorized to be onboarded to the network and, if so, whether it should be assigned any special roles or be subject to any specific access controls. It provides device authorization, role-based access control, and policy enforcement.
Build-Specific IoT Device	Aruba UXI Sensor	The IoT device that is used to demonstrate both trusted network-layer onboarding and trusted application-layer onboarding. It runs the Wi-Fi Easy Connect network-layer onboarding protocol supported by the build to securely receive its network credentials. It also has an application that enables it to perform independent (see Section 3.3.2) application-layer onboarding.
Generic IoT Device	Raspberry Pi	The IoT device that is used to demonstrate only trusted network-layer onboarding.
Secure Storage	Aruba UXI Sensor Trusted Platform Module (TPM)	Storage on the IoT device that is designed to be protected from unauthorized access and capable of detecting attempts to hack or modify its contents. Used to store and process private keys, credentials, and other information that must be kept confidential.
Certificate Authority (CA)	SEALSQ INeS CMS CA	Issues and signs certificates as needed. These certificates can be used by the device to connect to any 802.1a-based network.
Application-Layer Onboarding Service	UXI Application and UXI Cloud	After connecting to the network, the device downloads its application-layer credentials from the UXI cloud and uses them to authenticate to the UXI application, with which it interacts.

Component	Product	Function
Ongoing Device Authorization	N/A – Not intended for inclusion in this build	Performs activities designed to provide an ongoing assessment of the device’s trustworthiness and authorization to access network resources. For example, it may perform behavioral analysis or device attestation and use the results to determine whether the device should be granted access to certain high-value resources, assigned to a particular network segment, or other action taken.
Manufacturer Factory Provisioning Process	N/A (Not implemented at the time of publication)	Manufactures the IoT device. Creates, signs, and installs the device’s unique identity and other birth credentials into secure storage. Installs information the device requires for application-layer onboarding (if applicable). May populate a manufacturer database with information regarding devices that are created and, when the devices are sold, may record what entity owns them.

C.2 Build 1 Architecture

C.2.1 Build 1 Logical Architecture

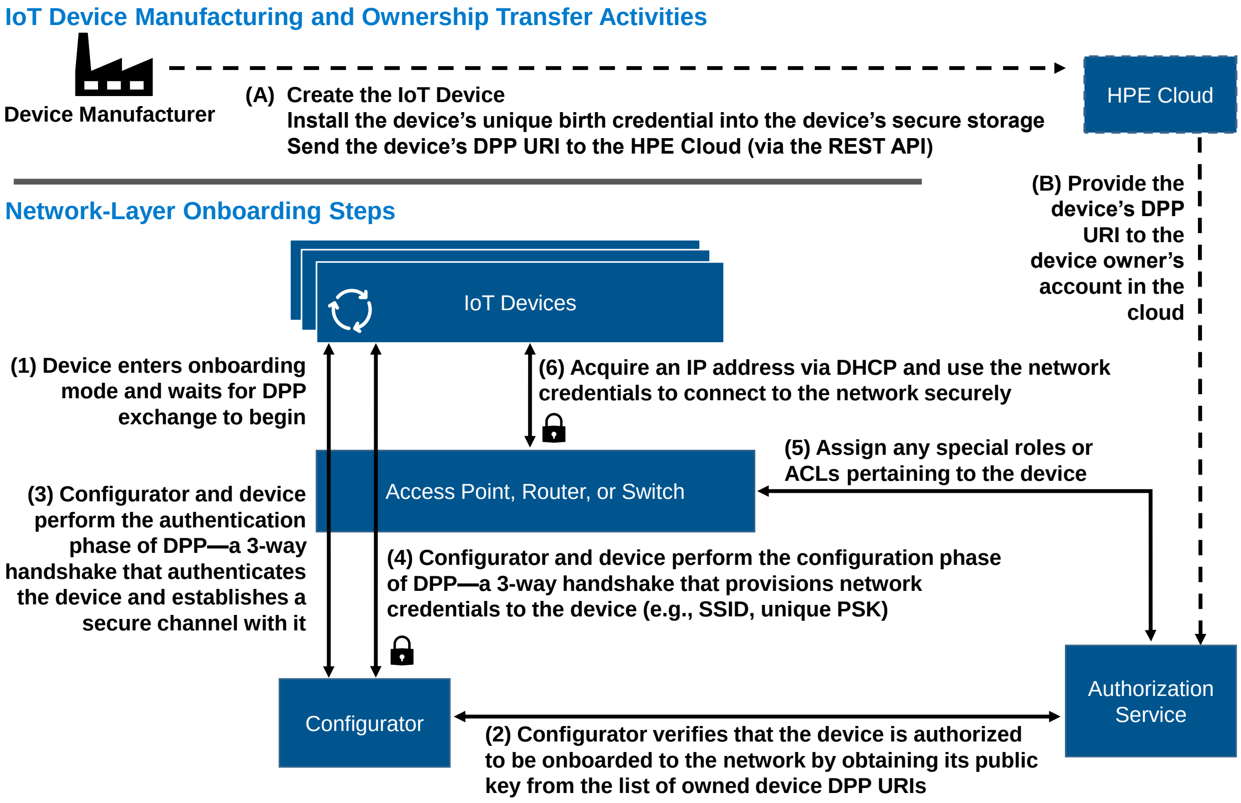
The network-layer onboarding steps that are performed in Build 1 are depicted in Figure C-1. These steps are broken into two main parts: those required to transfer device bootstrapping information from the device manufacturer to the device owner’s authorization service (labeled with letters) and those required to perform network-layer onboarding of the device (labeled with numbers).

The device manufacturer:

1. Creates the device and installs a unique birth credential into secure storage on the device. Then the manufacturer sends the device’s bootstrapping information, which takes the form of a DPP URI, to Aruba Central in the HPE cloud. The device manufacturer interfaces with the HPE cloud via a REST API.

2. When the device is purchased, the device’s DPP URI is sent to the HPE cloud account of the device’s owner. The device owner’s cloud account contains the DPP URIs for all devices that it owns.

Figure C-1 Logical Architecture of Build 1


IoT Device Manufacturing and Ownership Transfer Activities
Device Manufacturer
(A) Create the IoT Device
Install the device's unique birth credential into the device's secure storage
Send the device's DPP URI to the HPE Cloud (via the REST API)
HPE Cloud
(B) Provide the device's DPP URI to the device owner's account in the cloud
Network-Layer Onboarding Steps
IoT Devices
(1) Device enters onboarding mode and waits for DPP exchange to begin
(6) Acquire an IP address via DHCP and use the network credentials to connect to the network securely
(5) Assign any special roles or ACLs pertaining to the device
Access Point, Router, or Switch
(3) Configurator and device perform the authentication phase of DPP—a 3-way handshake that authenticates the device and establishes a secure channel with it
(4) Configurator and device perform the configuration phase of DPP—a 3-way handshake that provisions network credentials to the device (e.g., SSID, unique PSK)
Configurator
Authorization Service
(2) Configurator verifies that the device is authorized to be onboarded to the network by obtaining its public key from the list of owned device DPP URIs

After obtaining the device, the device owner provisions the device with its network credentials by performing the following network-layer onboarding steps:

1. The owner puts the device into onboarding mode. The device waits for the DPP exchange to begin. This exchange includes the device issuing a discovery message, which the owner's configurator hears. The discovery message is secured such that it can only be decoded by an entity that possesses the device's DPP URI.

2. The configurator consults the list of DPP URIs of all owned devices to decode the discovery message and verify that the device is owned by the network owner and is therefore assumed to be authorized to be onboarded to the network.

3. Assuming the configurator finds the device's DPP URI, the configurator and the device perform the authentication phase of DPP, which is a three-way handshake that authenticates the device and establishes a secure (encrypted) channel with it.

4. The configurator and the device use this secure channel to perform the configuration phase of DPP, which is a three-way handshake that provisions network credentials to the device, along with any other information that may be needed, such as the network SSID.

5. The router or switch consults the owner's authentication, authorization, and accounting (AAA) service to determine if the device should be assigned any special roles or if any special ACL entries should be made for the device. If so, these are configured on the router or switch.

6. The device uses Dynamic Host Configuration Protocol (DHCP) to acquire an IP address and then uses its newly provisioned network credentials to connect to the network securely.

This completes the network-layer onboarding process.

After the device is network-layer onboarded and connects to the network, it automatically performs independent (see Section 3.3.2) application-layer onboarding. The application-layer onboarding steps are not depicted in Figure C-1. During the application-layer onboarding process, the IoT device, which is a UXI sensor, authenticates itself to the UXI cloud using its manufacturing certificate and pulls its application-layer credentials from the UXI cloud. In addition, if a firmware update is relevant, this also happens. The UXI sensor contacts the UXI cloud service to download a customer-specific configuration that tells it what to monitor on the customer's network. The UXI sensor then conducts the network performance monitoring functions it is designed to perform and uploads the data it collects to the UXI application dashboard.

C.2.2 Build 1 Physical Architecture

Section 5.2 describes the physical architecture of Build 1.

Appendix D Build 2 (Wi-Fi Easy Connect, CableLabs, OCF)

D.1 Technologies

Build 2 is an implementation of network-layer onboarding that uses the Wi-Fi Easy Connect protocol. Build 2 also supports streamlined (see Section 3.3.2) application-layer onboarding to the OCF security domain. The network-layer onboarding infrastructure for Build 2 is provided by CableLabs and the application-layer onboarding infrastructure is provided by OCF. IoT devices that were network-layer onboarded using Build 2 were provided by Aruba/HPE and OCF. Only the IoT devices provided by OCF were capable of being both network-layer onboarded and streamlined application-layer onboarded. For more information on these collaborators and the products and technologies that they contributed to this project overall, see Section 3.4.

Build 2 onboarding infrastructure components consist of the CableLabs Custom Connectivity Gateway Agent, which runs on the Gateway Access Point, and the Platform Controller. IoT devices onboarded by Build 2 include the Aruba UXI Sensor and CableLabs Raspberry Pi.

Table D-1 lists the technologies used in Build 2. It lists the products used to instantiate each logical build component and the security function that the component provides. The components listed are logical. They may be combined in physical form, e.g., a single piece of hardware may house a network onboarding component, a router, and a wireless access point.

Table D-1 Build 2 Products and Technologies

Component	Product	Function
Network-Layer Onboarding Component (Configurator)	CableLabs Custom Connectivity Gateway Agent with support from CableLabs Platform Controller	Runs the Wi-Fi Easy Connect network-layer onboarding protocol to interact with the IoT device to perform one-way or mutual authentication, establish a secure channel, and securely provide local network credentials to the device. It also securely conveys application-layer bootstrapping information to the device as part of the Wi-Fi Easy Connect protocol to support application-layer onboarding. The network-layer onboarding component deployed in Build 2 supports DPP 2.0, but it is also backward compatible with DPP 1.0.
Access Point, Router, or Switch	Raspberry Pi (running Custom Connectivity Gateway Agent)	The access point includes a configurator that runs the Wi-Fi Easy Connect Protocol. It also serves as a router that: 1) routes all traffic exchanged between IoT devices and the rest of the network, and 2) assigns each IoT device to a local network segment appropriate to the device's trust level (optional).

Component	Product	Function
Supply Chain Integration Service	CableLabs Platform Controller/IoTivity Cloud Service	The device manufacturer provides device bootstrapping information (i.e., the DPP URI) to the CableLabs Web Server. There are several potential mechanisms for sending the DPP URI to the CableLabs Web Server. The manufacturer can send the device's DPP URI to the Web Server directly, via an API. The API used is not the REST API that is documented in the DPP specification. However, the API is published and was made available to manufacturers wanting to onboard their IoT devices using Build 2. Once the device is transferred to an owner, the CableLabs Web Server provides the device's DPP URI to the device owner's authorization service, which is part of the owner's configurator.
Authorization Service	CableLabs Platform Controller	The authorization service provides the configurator and router with the information needed to determine if the device is authorized to be onboarded to the network and, if so, whether it should be assigned any special roles, assigned to any specific network segments, or be subject to any specific access controls.
Build-Specific IoT Device	Raspberry Pi (Bulb) Raspberry Pi (switch)	The IoT devices that are used to demonstrate both trusted network-layer onboarding and trusted application-layer onboarding. They run the Wi-Fi Easy Connect network-layer onboarding protocol to securely receive their network credentials. They also support application-layer onboarding of the device to the OCF environment by conveying the device's application-layer bootstrapping information as part of the network-layer onboarding protocol.
Generic IoT Device	Aruba UXI Sensor	The IoT device that is used to demonstrate only trusted network-layer onboarding.
Secure Storage	N/A (IoT device is not equipped with secure storage)	Storage designed to be protected from unauthorized access and capable of detecting attempts to hack or modify its contents. Used to store and process private keys and other information that must be kept confidential.
Certificate Authority	N/A (Not implemented at the time of publication)	Issues and signs certificates as needed.
Application-Layer Onboarding Service	OCF Diplomat and OCF OBT within IoTivity	After connecting to the network, the OCF Diplomat authenticates the devices, establishes secure channels with them, and sends them access control lists that control which bulbs each switch is authorized to turn on and off.

Component	Product	Function
Ongoing Device Authorization	N/A – Not intended for inclusion in this build	Performs activities designed to provide ongoing assessment of the device's trustworthiness and authorization to access network resources. For example, it may perform behavioral analysis or device attestation and use the results to determine whether the device should be granted access to certain high-value resources, assigned to a particular network segment, or other action taken.
Manufacturer Factory Provisioning Process	N/A (Not yet implemented)	Manufactures the IoT device. Creates, signs, and installs the device's unique identity and other birth credentials into secure storage. Installs information the device requires for application-layer onboarding (if applicable). May populate a manufacturer database with information regarding devices that are created and, when the devices are sold, may record what entity owns them.

D.2 Build 2 Architecture

D.2.1 Build 2 Logical Architecture

The network-layer onboarding steps that are performed in Build 2 are depicted in Figure D-1. These steps are broken into two main parts: those required to transfer device bootstrapping information from the device manufacturer to the device owner's authorization service (labeled with letters) and those required to perform network-layer onboarding of the device (labeled with numbers).

The device manufacturer:

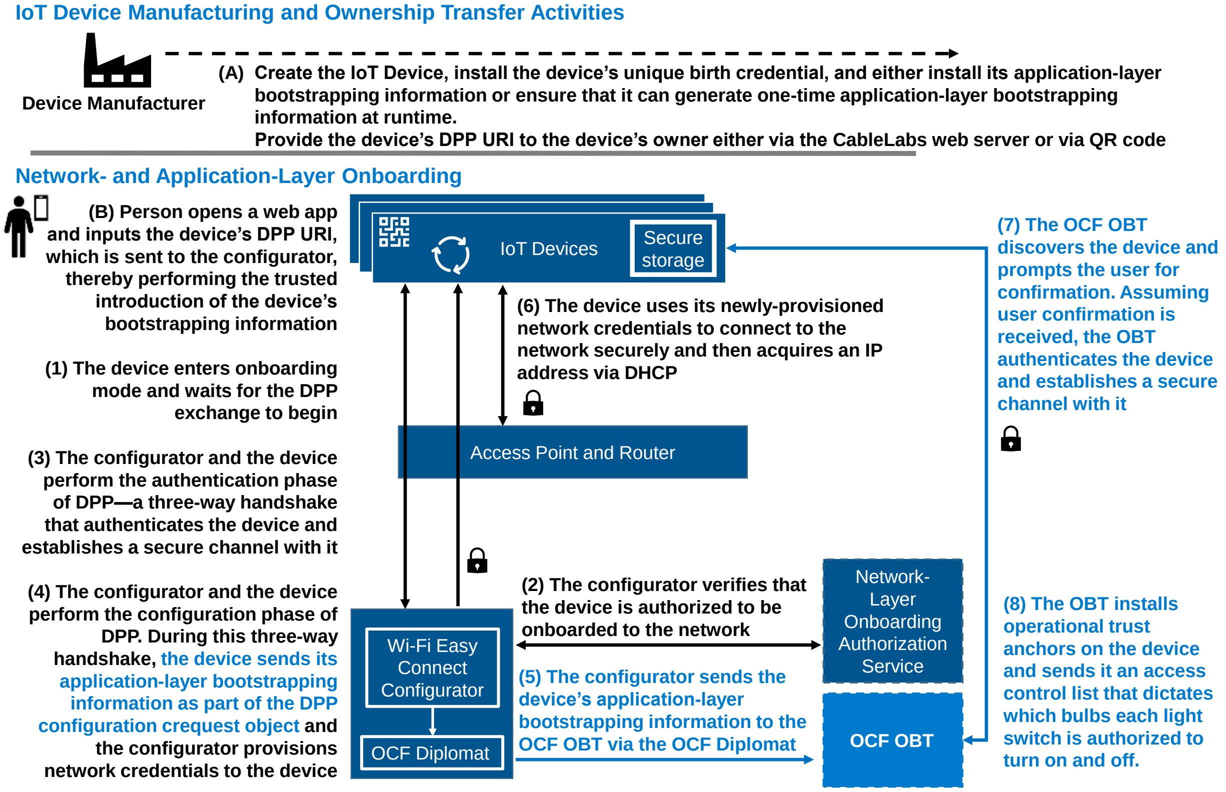
1. Creates the device and installs a unique birth credential into secure storage on the device. Because the device created for use in Build 2 will also perform application-layer onboarding into the OCF security domain, as part of the manufacturing process the manufacturer also either installs application-layer bootstrapping information onto the device or ensures that the device has the capability to generate one-time application-layer bootstrapping information at runtime. Then the manufacturer makes the device's network-layer bootstrapping information, which takes the form of a DPP URI, available to the device's owner.

   Build 2 supports several mechanisms whereby the manufacturer can make the device's network-layer bootstrapping information (i.e., its DPP URI) available to the device owner. The device's DPP URI can be uploaded directly to a device owner's cloud account or web server via API (as might come in handy when onboarding many enterprise devices at one time). Alternatively, the DPP URI can be manually entered into a local web portal that runs a configuration webpage that a device on the same Wi-Fi network can connect to for purposes of scanning a QR code or typing in the DPP URI. A DPP URI that is to be entered manually could, for example, be emailed to the owner or encoded into a QR code and printed on the device chassis, in device documentation, or on device packaging. Table D-1 depicts the case in which the manufacturer provides the device's DPP URI to the owner for manual entry. When the owner receives the device's DPP URI, the owner may optionally add the device's DPP URI to a list of

DPP URIs for devices that it owns that is maintained as part of the owner's authorization service. Such a list would enable the owner's network to determine if a device is authorized to be onboarded to it.

2. The person onboarding the device opens a web application and enters the device's DPP URI. The web application then sends the DPP URI to the Wi-Fi Easy Connect configurator, e.g., through a web request. (Note: Although the laboratory implementation of Build 2 requires the user to enter the DPP URI via a web page, an implementation designed for operational use would typically require the user to provide the DPP URI by scanning a QR code into a network operator-provided app that is logged into the user's account.)

Figure D-1 Logical Architecture of Build 2


IoT Device Manufacturing and Ownership Transfer Activities
Device Manufacturer
(A) Create the IoT Device, install the device's unique birth credential, and either install its application-layer bootstrapping information or ensure that it can generate one-time application-layer bootstrapping information at runtime.
Provide the device's DPP URI to the device's owner either via the CableLabs web server or via QR code
Network- and Application-Layer Onboarding
(B) Person opens a web app and inputs the device's DPP URI, which is sent to the configurator, thereby performing the trusted introduction of the device's bootstrapping information
(1) The device enters onboarding mode and waits for the DPP exchange to begin
(3) The configurator and the device perform the authentication phase of DPP—a three-way handshake that authenticates the device and establishes a secure channel with it
(4) The configurator and the device perform the configuration phase of DPP. During this three-way handshake, the device sends its application-layer bootstrapping information as part of the DPP configuration crequest object and the configurator provisions network credentials to the device
IoT Devices
Secure storage
(6) The device uses its newly-provisioned network credentials to connect to the network securely and then acquires an IP address via DHCP
Access Point and Router
(2) The configurator verifies that the device is authorized to be onboarded to the network
Wi-Fi Easy Connect Configurator
OCF Diplomat
(5) The configurator sends the device's application-layer bootstrapping information to the OCF OBT via the OCF Diplomat
Network-Layer Onboarding Authorization Service
OCF OBT
(7) The OCF OBT discovers the device and prompts the user for confirmation. Assuming user confirmation is received, the OBT authenticates the device and establishes a secure channel with it
(8) The OBT installs operational trust anchors on the device and sends it an access control list that dictates which bulbs each light switch is authorized to turn on and off.

After ensuring that the device's network-layer bootstrapping information (i.e., its DPP URI) has been uploaded to the configurator, the device owner performs both trusted network-layer onboarding and streamlined application-layer onboarding to the OCF security domain by performing the steps depicted in Figure D-1. In this diagram, the components that relate to network-layer onboarding are depicted in dark blue and their associated steps are written in black font. The components and steps that are related to application-layer onboarding are depicted in light blue. The steps are as follows:

1. The owner puts the device into onboarding mode. The device waits for the DPP exchange to begin. This exchange includes the device issuing a discovery message, which the owner's configurator hears. The discovery message is secured such that it can only be decoded by an entity that possesses the device's DPP URI.

2. Optionally, if such a list is being maintained, the configurator consults the list of DPP URIs of all owned devices to verify that the device is owned by the network owner and is, therefore, assumed to be authorized to be onboarded to the network. (If the device is being onboarded by an enterprise, the enterprise would likely maintain such a list; however, if the device is being onboarded to a home network, this step might be omitted.)

3. Assuming the configurator finds the device's DPP URI, the configurator and the device perform the authentication phase of DPP, which is a three-way handshake that authenticates the device and establishes a secure (encrypted) channel with it.

4. The configurator and the device use this secure channel to perform the configuration phase of DPP, which is a three-way handshake that provisions network credentials to the device, along with any other information that may be needed, such as the network SSID. In particular, as part of the three-way handshake in the Build 2 demonstration, the device sends its application-layer bootstrapping information to the configurator as part of the DPP configuration request object.

5. The configurator receives the device's application-layer bootstrapping information and forwards it to the OCF Diplomat. The purpose of the OCF Diplomat is to provide a bridge between the network and application layers. It accomplishes this by parsing the org.openconnectivity fields of the DPP request object, which contains the UUID of the device and the application-layer bootstrapping credentials, and sending these to the OCF OBT as part of a notification that the OBT has a new device to onboard. The Diplomat and the OBT use a subscribe and notify mechanism to ensure that the OBT will receive the onboarding request even if the OBT is unreachable for a period of time (e.g., the OBT is out of the home).

6. The device uses its newly provisioned network credentials to connect to the network securely and then uses DHCP to acquire an IP address. This completes the network-layer onboarding process.

7. The OBT implements a filtered discovery mechanism using the UUID provided from the OCF Diplomat to discover the new device on the network. Once it discovers the device, before proceeding, the OBT may optionally prompt the user for confirmation that they want to perform application-layer onboarding to the OCF security domain. This prompting may be accomplished, for example, by sending a confirmation request to an OCF app on the user's mobile device. Assuming the user responds affirmatively, the OBT uses the application-layer bootstrapping information to authenticate the device and take ownership of it by setting up a Datagram Transport Layer Security (DTLS) connection with the device.

8. The OBT then installs operational trust anchors and access control lists onto the device. For example, in the access control list, each light bulb may have an access control entry dictating which light switches are authorized to turn it on and off. This completes the application-layer onboarding process.

Note that, at this time, the application-layer bootstrapping information is provided unilaterally in the Build 2 application-layer onboarding demonstration. The application-layer bootstrapping information of the device is provided to the OCF Diplomat, enabling the OBT to authenticate the device. In a future version of this process, the application-layer bootstrapping information could be provided bidirectionally, meaning that the OCF Diplomat could also send the OCF operational root of trust to the IoT device as part of the DPP configuration response frame. Exchanging application-layer bootstrapping information bilaterally in this way would enable the secure channel set up as part of the network-layer onboarding process to support establishment of a mutually authenticated session between the device and the OBT.

In the Build 2 demonstration, two IoT devices, a switch and a light bulb, are onboarded at both the network and application layers. Each of these devices sends the OCF Diplomat its application-layer bootstrapping information over the secure network-layer onboarding channel during the network-layer onboarding process. Immediately after they complete the network-layer onboarding process and connect to the network, the OCF Diplomat provides their application-layer bootstrapping information to the OBT. The OBT then uses the provided application-layer bootstrapping information to discover, authenticate, and onboard each device. Because the devices have no way to authenticate the identity of the OBT in the current implementation, the devices are configured to trust the OBT upon first use.

After the OBT authenticates the devices, it establishes secure channels with them and provisions them access control lists that control which bulbs each switch is authorized to turn on and off. To demonstrate that the application onboarding was successful, Build 2 demonstrates that the switch is able to control only those bulbs that the OCF OBT has authorized it to.

D.2.2 Build 2 Physical Architecture

Section 5.3 describes the physical architecture of Build 2.

Appendix E Build 3 (BRSKI, Sandelman Software Works)

E.1 Technologies

Build 3 is an implementation of network-layer onboarding that uses the BRSKI protocol. Build 3 does not support application-layer onboarding. The network-layer onboarding infrastructure and related technology components for Build 3 were provided by Sandelman Software Works. The Raspberry Pi, ESP32, and Nordic NRF IoT devices that will be onboarded in a future implementation of Build 3 were also provided by Sandelman Software Works, as was the Sandelman Software Works Reach Pledge Simulator, which is the device that is onboarded in the current build. The IoT devices do not have secure storage, but future plans are to integrate them with secure storage elements. Build 3 issues private PKI certificates as network credentials at this time, but future plans are to integrate Build 3 with a third-party private CA from which it can obtain signed certificates. For more information on Sandelman Software Works and the products and technologies that it contributed to this project overall, see Section 3.4.

Onboarding Build 3 infrastructure components consist of Raspberry Pi, Nordic NRF, ESP32, Sandelman Software Works Minerva Fountain Join Registrar/Coordinator, Sandelman Software Works Minerva. Highway, Sandelman Software Works Reach Pledge Simulator, and a Minerva Fountain internal CA.

Table E-1 lists the technologies used in Build 3. It lists the products used to instantiate each logical build component and the security function that the component provides. The components are logical. They may be combined in physical form, e.g., a single piece of hardware may house both a network onboarding component and a router and/or wireless access point.

Table E-1 Build 3 Products and Technologies

Component	Product	Function
Network-Layer Onboarding Component (BRSKI Domain Registrar)	Sandelman Software Works Minerva Fountain Registrar	Runs the BRSKI protocol. It authenticates the IoT device, receives a voucher-request from the IoT device, and passes the request to the MASA. It also receives a voucher from the MASA, verifies it, and passes it to the IoT device. Assuming the IoT device finds the voucher to be valid and determines that the network is authorized to onboard it, the Domain Registrar provisions network credentials to the IoT device using EST.
Access Point, Router, or Switch	Turris MOX router running OpenWRT	The Onboarding Router segments the onboarding device from the rest of the network until the BRSKI onboarding is complete

Component	Product	Function
Supply Chain Integration Service (Manufacturer Authorized Signing Authority—MASA)	Minerva Highway, which is a MASA provided by Sandelman Software Works	The device manufacturer provides device bootstrapping information (e.g., the device's X.509 certificate) and device ownership information to the MASA. The MASA creates and signs a voucher saying who the owner of the device is and provides this voucher to the IoT device via the Domain Registrar so that the device can verify that the network that is trying to onboard it is authorized to do so.
Authorization Service	Minerva Highway, which is a MASA provided by Sandelman Software Works	As described in the previous row.
IoT Device (Pledge)	Sandelman Software Works Reach Pledge Simulator	The device that is used to demonstrate trusted network-layer onboarding by joining the network.
Secure Storage	N/A (The IoT devices and the Sandelman Software Works Reach Pledge Simulator do not include secure storage)	Storage on the IoT device that is designed to be protected from unauthorized access and capable of detecting attempts to hack or modify its contents. Used to store and process private keys, credentials, and other information that must be kept confidential.
Certificate Authority	N/A (self-signed certificates were used)	Issues and signs certificates as needed.
Application-Layer Onboarding Service	None. Not supported in this build.	After connecting to the network, the device mutually authenticates with a trusted application service and interacts with it at the application layer.
Ongoing Device Authorization	N/A – Not intended for inclusion in this build	Performs activities designed to provide an ongoing assessment of the device's trustworthiness and authorization to access network resources. For example, it may perform behavioral analysis or device attestation and use the results to determine whether the device should be granted access to certain high-value resources, assigned to a particular network segment, or other action taken.
Manufacturer Factory Provisioning Process	N/A (Not implemented at the time of publication)	Manufactures the IoT device. Creates, signs, and installs the device's unique identity and other birth credentials into secure storage. Installs information the device requires for application-layer onboarding (if applicable). May populate a manufacturer database with information regarding devices that are created and, when the devices are sold, may record what entity owns them.

E.2 Build 3 Architecture

E.2.1 Build 3 Logical Architecture

The network-layer onboarding steps that are performed in Build 3 are depicted in Figure E-1. These steps are broken into two main parts: those required to transfer device bootstrapping information from the device manufacturer to the device owner's authorization service (labeled with letters) and those required to perform network-layer onboarding of the device (labeled with numbers). These steps are described in greater detail in IETF RFC 8995.

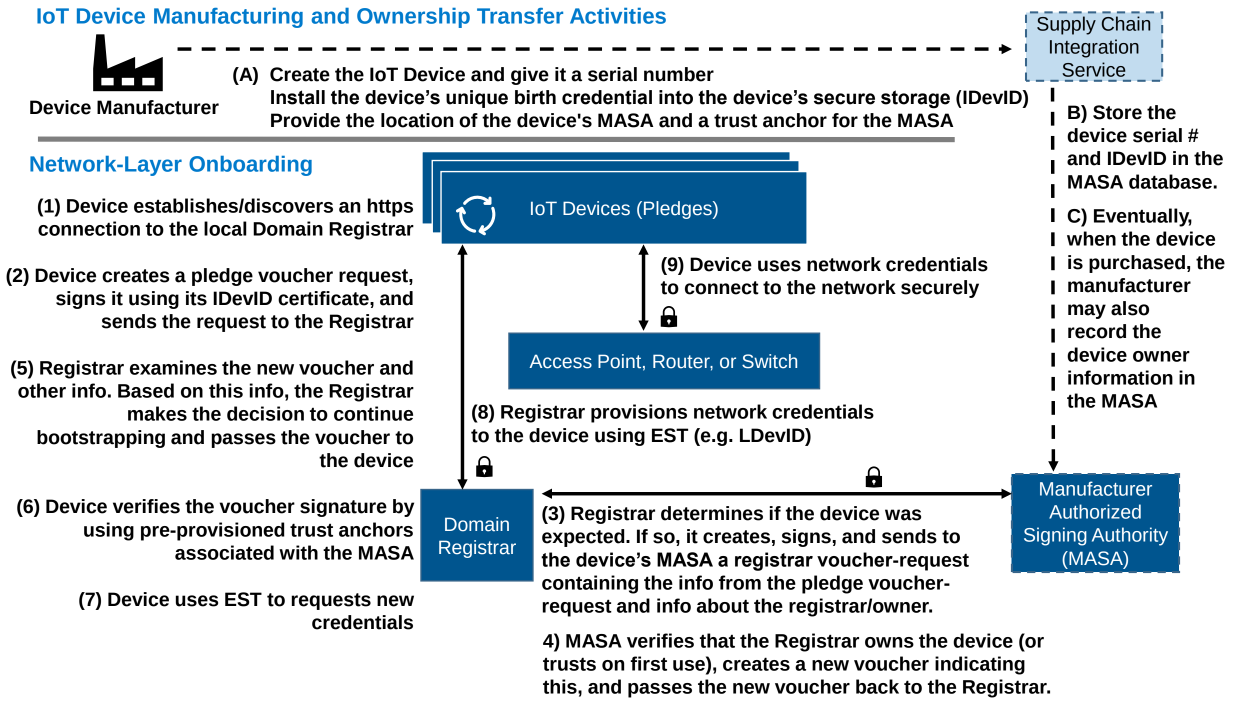
The device manufacturer:

1. Creates the device and installs a unique serial number and birth credential into secure storage on the device. This unique birth credential takes the form of a private key and its associated 802.1AR certificate, e.g., the device's IDevID. As part of this factory-installed certificate process, the location of the device's MASA is provided in an extension to the IDevID. The device is also provided with trust anchors for the MASA entity that will sign the returned vouchers.

2. Stores information about the device, such as its serial number and its IDevID, in the MASA's database.

3. Eventually, when the device is sold, the MASA may also record the device ownership information in its database.

Figure E-1 Logical Architecture of Build 3


IoT Device Manufacturing and Ownership Transfer Activities
Device Manufacturer
(A) Create the IoT Device and give it a serial number
Install the device's unique birth credential into the device's secure storage (IDevID)
Provide the location of the device's MASA and a trust anchor for the MASA
Supply Chain Integration Service
B) Store the device serial # and IDevID in the MASA database.
C) Eventually, when the device is purchased, the manufacturer may also record the device owner information in the MASA
Network-Layer Onboarding
(1) Device establishes/discovers an https connection to the local Domain Registrar
(2) Device creates a pledge voucher request, signs it using its IDevID certificate, and sends the request to the Registrar
(5) Registrar examines the new voucher and other info. Based on this info, the Registrar makes the decision to continue bootstrapping and passes the voucher to the device
(6) Device verifies the voucher signature by using pre-provisioned trust anchors associated with the MASA
(7) Device uses EST to requests new credentials
IoT Devices (Pledges)
(9) Device uses network credentials to connect to the network securely
Access Point, Router, or Switch
(8) Registrar provisions network credentials to the device using EST (e.g. LDevID)
Domain Registrar
(3) Registrar determines if the device was expected. If so, it creates, signs, and sends to the device's MASA a registrar voucher-request containing the info from the pledge voucher-request and info about the registrar/owner.
4) MASA verifies that the Registrar owns the device (or trusts on first use), creates a new voucher indicating this, and passes the new voucher back to the Registrar.
Manufacturer Authorized Signing Authority (MASA)

After obtaining the device, the device owner provisions the device with its network credentials by performing the following network-layer onboarding steps:

1. The owner puts the device into onboarding mode. The device establishes an https connection to the local Domain Registrar. Trust in the Domain Registrar is provisional. (In a standard implementation, the device would use link-local network connectivity to locate a join proxy, and the join proxy would provide the device with https connectivity to the local Domain Registrar. The Build 3 implementation, however, does not support discovery at this time. To overcome this code limitation, the IoT device has been pre-provided with the address of the local Domain Registrar, to which it connects directly.)

2. The device creates a pledge voucher-request that includes the device serial number, signs this request with its IDevID certificate (i.e., its birth credential), and sends this signed request to the Registrar.

3. The Registrar receives the pledge voucher-request and considers whether the manufacturer is known to it and whether devices of that type are welcome. If so, the Registrar forms a registrar voucher-request that includes all the information from the pledge voucher-request along with information about the registrar/owner. The Registrar signs this registrar voucher-request. It locates the MASA that the IoT device is known to trust (e.g., the MASA that is identified in the device’s IDevID extension) and sends the registrar voucher-request to the MASA.

4. The MASA consults the information that it has stored and applies policy to determine whether or not to approve the Registrar’s claim that it owns and/or is authorized to onboard the device. (For example, the MASA may consult sales records for the device to verify device ownership, or it may be configured to trust that the first registrar that contacts it on behalf of a given device is in fact the device owner.) Assuming the MASA decides to approve the Registrar’s claim to own and/or be authorized to onboard the device, the MASA creates a voucher that directs the device to accept its new owner/authorized network, signs this voucher, and sends it back to the Registrar.

5. The Registrar receives this voucher, examines it along with other related information (such as security posture, remote attestation results, and/or expected device serial numbers), and determines whether it trusts the voucher. Assuming it trusts the voucher, the Registrar passes the voucher to the device.

6. The device uses its factory-provisioned MASA trust anchors to verify the voucher signature, thereby ensuring that the voucher can be trusted. The voucher also validates the Registrar and represents the intended owner, ending the provisional aspect of the EST connection.

7. The device uses Enrollment over Secure Transport (EST) to request new credentials.

8. The Registrar provisions network credentials to the device using EST. These network credentials get stored into secure storage on the device, e.g., as an LDevID.

9. The device uses its newly provisioned network credentials to connect to the network securely.

This completes the trusted network-layer onboarding process for Build 3.

E.2.2 Build 3 Physical Architecture

Section 5.4 describes the physical architecture of Build 3.

Appendix F Build 4 (Thread, Silicon Labs-Thread, Kudelski KeySTREAM)

F.1 Technologies

Build 4 is an implementation of network-layer connection to an OpenThread network, followed by use of the Kudelski IoT keySTREAM Service to perform independent (see Section 3.3.2) application-layer onboarding of the device to a particular customer's tenancy in the AWS IoT Core. To join the network, the joining device generates and displays a pre-shared key that the owner enters on the commissioner, through a web interface, for authentication. The network-layer infrastructure for Build 4 was provided by Silicon Labs. The application-layer onboarding infrastructure for Build 4 was provided by Kudelski IoT. IoT devices that were onboarded using Build 4 were provided by Silicon Labs. For more information on these collaborators and the products and technologies that they contributed to this project overall, see Section 3.4.

Build 4 network infrastructure components within the NCCoE lab consist of a Thread border router (which is implemented using a Raspberry Pi) and a Silicon Labs Gecko Wireless Starter Kit. Build 4 also requires support from the Kudelski IoT keySTREAM service to perform application-layer onboarding. The keySTREAM service comes as a SaaS platform that is running in the cloud (accessible via the internet), and a software library (KTA – Kudelski Trusted Agent) that is integrated in the IoT device software stack. The KTA integrates with the Silicon Labs' Hardware Root of Trust (Secure Vault). The IoT device that is connected to the network and application-layer onboarded using Build 4 is the Silicon Labs Thunderboard (BRD2601A) with EFR32MG24x with Secure Vault(TM) High which is security certified to PSA/SESIP Level 3.

Table F-1 lists the technologies used in Build 4. It lists the products used to instantiate each logical build component and the security function that the component provides. The components are logical. They may be combined in physical form, e.g., a single piece of hardware may house a network onboarding component, a router, and a wireless access point.

Table F-1 Build 4 Products and Technologies

Component	Product	Function
Network-Layer Onboarding Component (Thread Protocol Component)	SLWSTK6023A Thread Radio Transceiver (Wireless starter kit);	The SLWSTK6023A acts as a Thread radio transceiver or radio coprocessor (RCP), allowing the open thread boarder router host platform to form and communicate with a Thread network. If the Thread MeshCoP were running on this device, it would provision the IoT device with credentials for the Thread network.
Access Point, Router, or Switch	OpenThread Border Router (OTBR) hosted on a Raspberry Pi	Router that has interfaces both on the Thread network and on the IP network to act as a bridge between the Thread network and the public internet. This allows the IoT device that communicates using the Thread wireless protocol to communicate with cloud services.

Component	Product	Function
Supply Chain Integration Service	Silicon Labs Custom Parts Manufacturer Service (CPMS)	To support network-layer onboarding, the device manufacturer provides device bootstrapping information to the to the device owner.
Authorization Service	Not implemented	Enables the network to verify that the device that is trying to onboard to it is authorized to do so.
IoT Device	Silicon Labs Thunderboard (BRD2601A)	The IoT device that is used to demonstrate trusted network- and application-layer onboarding.
Secure Storage	Secure Vault ™ High on Silicon Labs IoT device	Storage designed to be protected from unauthorized access and capable of detecting attempts to hack or modify its contents. Used to store and process private keys and other information that must be kept confidential.
Certificate Authority	Each tenant in the Kudelski keySTREAM service cloud has its own certificate signing authority	Issues and signs certificates as needed. For application-layer onboarding, the device owner has its own certificate signing authority in its portion of the Kudelski keySTREAM service cloud.
Application-Layer Onboarding Service	Kudelski keySTREAM Service	After connecting to the Thread network, the device performs application-layer onboarding by accessing the Kudelski keySTREAM service. The device and the keySTREAM service mutually authenticate; the keySTREAM service verifies the device's owner, generates an application-layer credential (i.e., an AWS certificate that is based on the device's chipset identity and owner) for the device, and provisions the device with this X.509 credential that will enable the device to access the owner's tenancy in the AWS IoT Core cloud.
Ongoing Device Authorization	N/A – Not intended for inclusion in this build	Performs activities designed to provide an ongoing assessment of the device's trustworthiness and authorization to access network resources. For example, it may perform behavioral analysis or device attestation and use the results to determine whether the device should be granted access to certain high-value resources, assign the device to a particular network segment, or take other action.

Component	Product	Function
Manufacturer Factory Provisioning Process	Silicon Labs Custom Parts Manufacturing Service (CPMS)	Manufactures the IoT device. Creates, signs, and installs the device's unique identity and other birth credentials into secure storage. Installs software and information the device requires for application-layer onboarding. May populate a manufacturer database with information regarding devices that are created and, when the devices are sold, may record what entity owns them. The MG24 "B" version comes pre-loaded with a Silicon Labs Birth certificate. The "A" or "B" version birth certificate can be modified via their Custom Part Manufacturing Service (CPMS) to be unique per end device manufacturer and signed into their Root CA if desired.

F.2 Build 4 Architecture

F.2.1 Build 4 Logical Architecture

Build 4 demonstrates a device connecting to an OpenThread network. IoT devices generate and use a pre-shared key to connect to the OpenThread network of Build 4 using the Thread MeshCoP service. Once a device is connected to the OpenThread network of Build 4, it gets access to an IP network via a border router, and then performs application-layer onboarding using the Kudelski keySTREAM Service. Kudelski keySTREAM is a device security management service that runs as a SaaS platform on the Amazon cloud. Build 4 relies on an integration that has been performed between Silicon Labs and Kudelski keySTREAM. KeySTREAM has integrated software libraries with the Silicon Lab EFR32MG24 (MG24) IoT device's secure vault to enable the private signing key that is associated with an application-layer certificate to be stored into the secure vault using security controls that are available on the MG24. This integration ensures that application-layer credentials can be provisioned into the vault securely such that no key material is misused or exposed.

At a high level, the steps required to enable demonstration of Build 4's network connection and application-layer onboarding capabilities can be broken into the following three main parts:

- Device Preparation: The IoT device is prepared for network connection and application-layer onboarding by the device manufacturer.
    - The device comes from the manufacturer ready to be provisioned onto a Thread network. No additional preparation is required.
    - The device is prepared for application-layer onboarding on behalf of a specific, pre-defined customer who will become its owner. The device is assigned ownership to this customer (e.g., customer A) and this ownership information is sealed into device firmware, permanently identifying the device as being owned by customer A. The device owner, customer A, has a tenancy on the Kudelski keySTREAM Service and is also an Amazon Web Services (AWS) customer. After the device has been prepared, the device is provided to its owner (customer A).

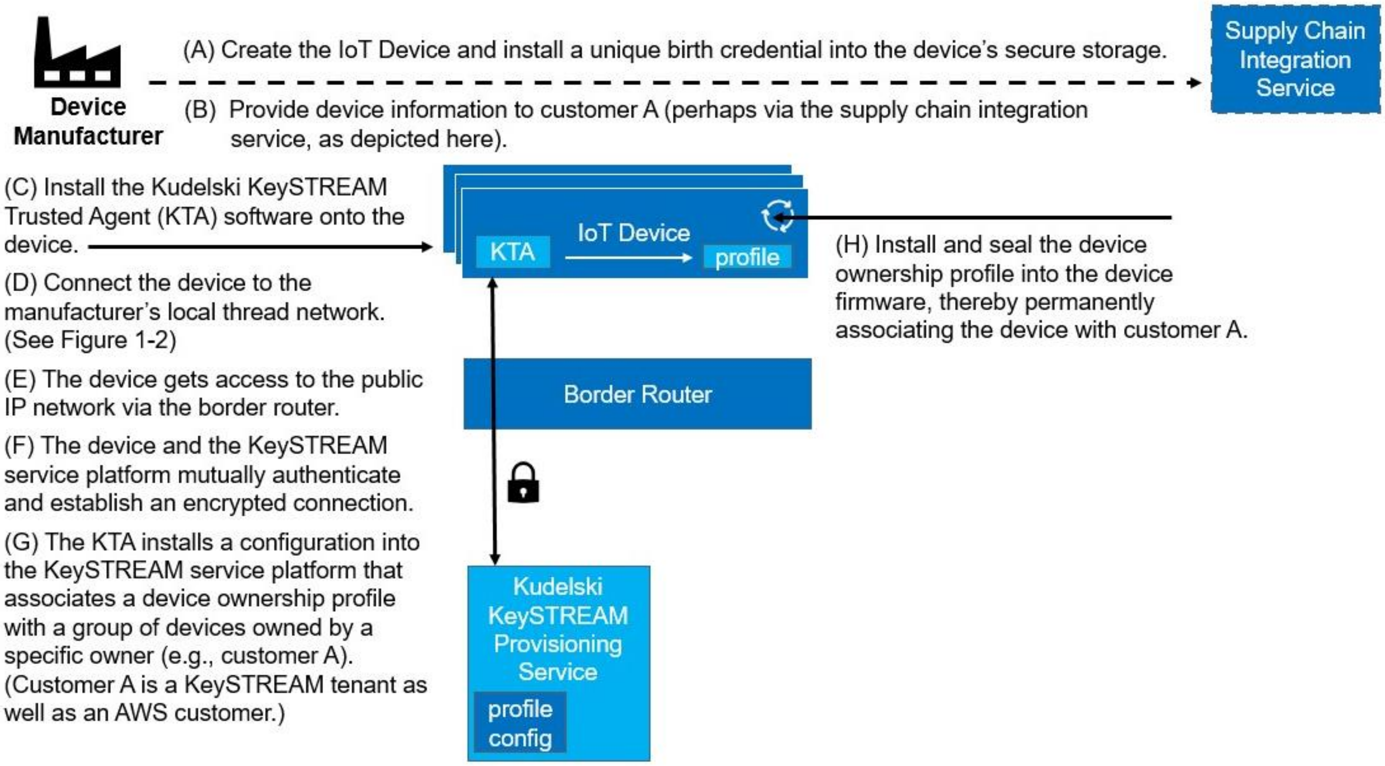
- Network Connection: Customer A connects the device to Customer A's OpenThread network by entering the pre-shared key displayed on the device's serial terminal in the OpenThread Border Router's (OTBR) web interface. This allows the network's radio channel, PAN ID, extended PAN ID and network name to be discovered, avoiding the need to preconfigure any of these parameters. Once on customer A's OpenThread network, the device has access to the public IP network via the border router.
- Application-Layer Onboarding: The device and the keySTREAM service mutually authenticate, keySTREAM confirms that customer A owns the device, and keySTREAM provisions the device with an AWS certificate that is specific to the device and to customer A, enabling the device to authenticate to customer A's tenancy in the AWS IoT Core.

Each of these three aspects of the demonstration are illustrated in its own figure and described in more detail in the three subsections below.

F.2.1.1 Device Preparation

Figure F-1 depicts the steps that are performed by the device manufacturer, which in this case is Silicon Labs, to prepare the device for network- and application-layer onboarding by a particular customer, Customer A. Each step is described in more detail below. Because these steps are performed to prepare the device for onboarding rather than as part of onboarding itself, they are labeled with letters instead of numbers in keeping with the conventions used in other build descriptions.

Figure F-1 Logical Architecture of Build 4: Device Preparation


Device Manufacturer
(A) Create the IoT Device and install a unique birth credential into the device's secure storage.
(B) Provide device information to customer A (perhaps via the supply chain integration service, as depicted here).
Supply Chain Integration Service
(C) Install the Kudelski KeySTREAM Trusted Agent (KTA) software onto the device.
(D) Connect the device to the manufacturer's local thread network. (See Figure 1-2)
(E) The device gets access to the public IP network via the border router.
(F) The device and the KeySTREAM service platform mutually authenticate and establish an encrypted connection.
(G) The KTA installs a configuration into the KeySTREAM service platform that associates a device ownership profile with a group of devices owned by a specific owner (e.g., customer A). (Customer A is a KeySTREAM tenant as well as an AWS customer.)
IoT Device
KTA
profile
(H) Install and seal the device ownership profile into the device firmware, thereby permanently associating the device with customer A.
Border Router
Kudelski KeySTREAM Provisioning Service
profile config

The following steps are performed to prepare the device for network connection and application-layer onboarding:

1. The manufacturer creates the device, which in this case is a Silicon Labs MG24, and prepares it for network connection by installing the device's unique birth credential into the device's chipset. This chipset identity is a hardware root of trust. The MG24 "B" version comes pre-loaded with a Silicon Labs Birth certificate. The "A" or "B" version birth certificate can be modified via their Custom Part Manufacturing Service (CPMS) to be unique per end device manufacturer and signed into their Root CA if desired.

2. The manufacturer provides information about the device to customer A (perhaps via the supply chain service, as depicted in Figure 1-1) so customer A can be aware that the device is expected on its network.

3. The manufacturer prepares the device for application-layer onboarding by installing the Kudelski keySTREAM Trusted Agent (KTA) software onto the device.

4. The manufacturer connects the device to the manufacturer's local OpenThread network. (See Figure 1-2 for details of the network connection steps.) Note that in this case, which is the first time that the device is being connected to a network, the device is being connected to the manufacturer's network rather than to the network of the device's eventual owner.

5. After the device connects to the manufacturer's OpenThread network, the device has access to the public IP network via the border router.

6. The device and the Kudelski keySTREAM service mutually authenticate and establish an encrypted connection.

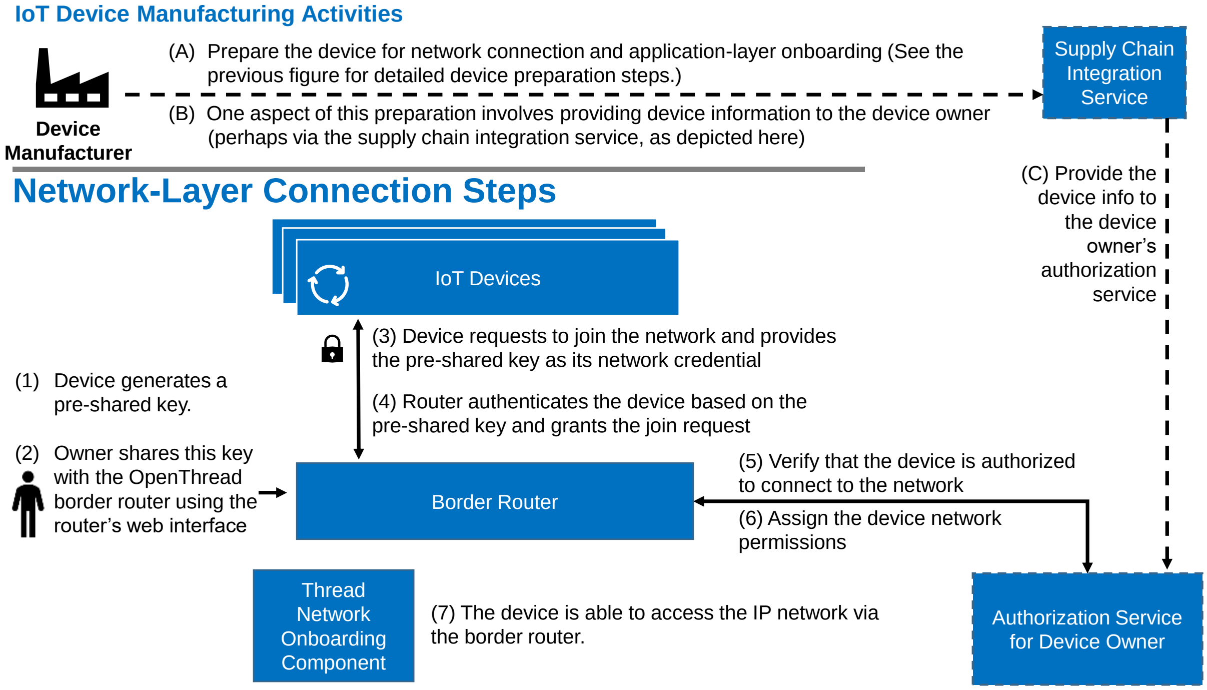
7. The KTA installs a configuration into the keySTREAM service platform that builds up a group of devices that belong to a certain end user and associates the group with a device ownership profile. This device ownership profile is associated with a particular customer (e.g., customer A). The same device profile is used by all devices in a group of devices that are owned by this owner. The profile is not specific to individual devices. The owner of these devices (customer A) has a keySTREAM tenancy, which includes a dedicated certificate signing CA. Customer A is also an AWS customer.

8. The device manufacturer installs and seals this device ownership profile into the device firmware. This profile permanently identifies the device as being owned by customer A.

F.2.1.2 Network-Layer Connection

Figure F-2 depicts the steps of an IoT device connecting to that thread network using a pre-shared key that the device generates and shares with the OpenThread boarder router. Each step is described in more detail below.

Figure F-2 Logical Architecture of Build 4: Connection to the OpenThread Network


IoT Device Manufacturing Activities
Device Manufacturer
(A) Prepare the device for network connection and application-layer onboarding (See the previous figure for detailed device preparation steps.)
(B) One aspect of this preparation involves providing device information to the device owner (perhaps via the supply chain integration service, as depicted here)
Supply Chain Integration Service
(C) Provide the device info to the device owner's authorization service
Network-Layer Connection Steps
IoT Devices
(1) Device generates a pre-shared key.
(2) Owner shares this key with the OpenThread border router using the router's web interface
(3) Device requests to join the network and provides the pre-shared key as its network credential
(4) Router authenticates the device based on the pre-shared key and grants the join request
Border Router
(5) Verify that the device is authorized to connect to the network
(6) Assign the device network permissions
Thread Network Onboarding Component
(7) The device is able to access the IP network via the border router.
Authorization Service for Device Owner

The device connects to the OpenThread network using the following steps:

1. The device generates a pre-shared key.

2. The owner starts the commissioning process by entering this pre-shared key on the OpenThread border router.

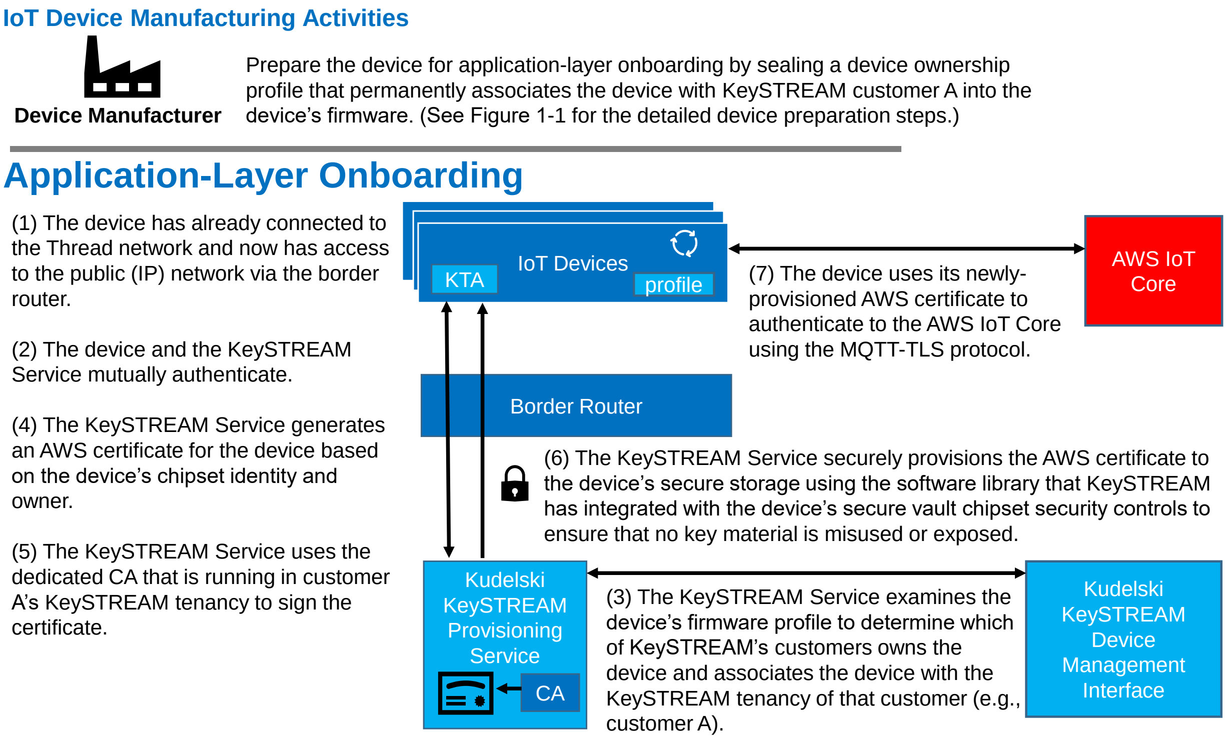
3. The device requests to join the network and provides the pre-shared key as its network credential.
4. The network authenticates the device based on the pre-shared key and grants the join request.
5. The network verifies that the device is authorized to connect to the network.
6. The network assigns the device network permissions and configures these as policies on the border router.
7. The device is able to access the IP network (and the internet) via the border router.

This completes the network-layer connection process.

F.2.1.3 Application-Layer Onboarding

Figure F-3 depicts the steps of the application-layer onboarding process using the Kudelski keySTREAM service. Each step is described in more detail below.

Figure F-3 Logical Architecture of Build 4: Application-Layer Onboarding using the Kudelski keySTREAM Service



IoT Device Manufacturing Activities
Device Manufacturer
Application-Layer Onboarding
KTA
IoT Devices
profile
AWS IoT Core
Border Router
Kudelski KeySTREAM Provisioning Service
CA
Kudelski KeySTREAM Device Management Interface

Prepare the device for application-layer onboarding by sealing a device ownership profile that permanently associates the device with KeySTREAM customer A into the device's firmware. (See Figure 1-1 for the detailed device preparation steps.)

(1) The device has already connected to the Thread network and now has access to the public (IP) network via the border router.

(2) The device and the KeySTREAM Service mutually authenticate.

(3) The KeySTREAM Service examines the device's firmware profile to determine which of KeySTREAM's customers owns the device and associates the device with the KeySTREAM tenancy of that customer (e.g., customer A).

(4) The KeySTREAM Service generates an AWS certificate for the device based on the device's chipset identity and owner.

(5) The KeySTREAM Service uses the dedicated CA that is running in customer A's KeySTREAM tenancy to sign the certificate.

(6) The KeySTREAM Service securely provisions the AWS certificate to the device's secure storage using the software library that KeySTREAM has integrated with the device's secure vault chipset security controls to ensure that no key material is misused or exposed.

(7) The device uses its newly-provisioned AWS certificate to authenticate to the AWS IoT Core using the MQTT-TLS protocol.

The application-layer onboarding steps performed to provision the device with its application-layer credentials (e.g., its AWS certificate) are as follows:

1. The device, which is already connected to the OpenThread network, accesses the IP network via the border router.
2. The device and the keySTREAM service mutually authenticate.

3. The keySTREAM Service examines the device’s firmware profile to determine which of keySTREAM’s customers owns the device. In this case, customer A is identified as the device owner. The keySTREAM service associates the device with customer A’s keySTREAM tenancy.

4. The keySTREAM Service generates an AWS IoT Core certificate for the device based on both the device’s ownership information and the secure hardware root of trust that is in the device’s chipset.

5. The keySTREAM Service uses the dedicated CA that is running in customer A’s keySTREAM tenancy to sign the AWS certificate.

6. The keySTREAM Service securely provisions the AWS certificate to the device’s secure storage using the software library that keySTREAM has integrated with the device’s secure vault chipset security controls to ensure that no key material is misused or exposed.

7. The device uses its newly provisioned application-layer credentials (i.e., the AWS certificate) to authenticate to customer A’s tenancy in the AWS IoT Core using the MQTT-TLS protocol.

F.2.2 Build 4 Physical Architecture

Section 5.5 describes the physical architecture of Build 4.

Appendix G Build 5 (BRSKI over Wi-Fi, NquiringMinds)

G.1 Technologies

Build 5 is an implementation of network-layer onboarding that uses a version of the BRSKI Protocol that has been modified to work over Wi-Fi. After the IoT device has joined the network, Build 5 also demonstrates a number of mechanisms that are performed on an ongoing basis to provide continuous, policy-based authorization and assurance. Both the network-layer onboarding infrastructure and the continuous assurance service for Build 5 were provided by NquiringMinds. This entire build can be replicated using the open sourced TrustNetZ code base.

For more information on NquiringMinds and the products and technologies that they contributed to this project overall, see Section 3.4.

Build 5 network onboarding infrastructure components within the NCCoE lab consist of a Linux based Raspberry Pi 4B router (which also runs the registrar service and MASA service), and a USB hub. The Build 5 components used to support the continuous assurance service include TrustNetZ Authorization interfaces, TrustNetZ information provider, and TrustNetZ policy engine. The IoT devices that are onboarded using Build 5 are a Raspberry Pi device. These IoT devices do not have secure storage, but use the Infineon Optiga SLB 9670 TPM 2.0 as an external secure element. Build 5 depends on an IDevID (X.509 Certificate) having been provisioned to the secure element of the IoT device (pledge) prior to onboarding, as part of the factory provisioning process (see Section H.1). For Build 5, this factory provisioning process was accomplished by the BRSKI Factory Provisioning Build, which is described in Appendix H.3.

Table G-1 lists the technologies used in Build 5. It lists the products used to instantiate each logical build component and the security function that the component provides. The components are logical. They may be combined in physical form, e.g., a single piece of hardware may house a network onboarding component, a router, and a wireless access point.

Table G-1 Build 5 Products and Technologies

Component	Product	Function
Network-Layer Onboarding Component (BRSKI Domain Registrar)	Stateful, non-persistent Linux app that has two functional interfaces for both BRSKI and for the Authentication Service. (TrustNetZ onboarding)	Runs the BRSKI protocol modified to work over Wi-Fi and acts as a BRSKI Domain Registrar. It authenticates the IoT device, receives a voucher request from the IoT device, and passes the request to the MASA. It also receives a voucher from the MASA, verifies it, and passes it to the IoT device. Assuming the IoT device finds the voucher to be valid and determines that the network is authorized to onboard it, the Domain Registrar provisions network credentials to the IoT device using EST.

Component	Product	Function
Access Point, Router, or Switch	Raspberry Pi 4B equipped with USB Wi-Fi dongle, running TrustNetZ AP code.	Router, providing an open Wi-Fi network and closed Wi-Fi network. Physical access control is mediated through the RADUIS interface (which is part of the TrustNetZ AP configuration) The AP also receives network commands from the continuous assurance service.
Supply Chain Integration Service (Manufacturer Authorized Signing Authority—MASA)	TrustNetZ MASA	The MASA creates and signs a voucher and provides this voucher to the IoT device via the Registrar so that the device can verify that the network that is trying to onboard it is authorized to do so.
Authorization Service	Linux application which contains an encapsulated policy engine (TrustNetZ policy engine)	Determines whether the device is authorized to be onboarded to the network. The application features a REST API which accepts verifiable credential claims to feed data on entities and their relationships into its SQL database. The policy engine itself is based on verifiable credentials presentation, (persisted to SQL database), making it easily configurable and extensible.
IoT Device	Raspberry Pi devices (running TrustNetZ pledge agent)	The IoT device that is used to demonstrate trusted network- and application-layer onboarding. Handles the client side BRSKI protocols, the integration with the secure storage, with factory provisioning and TLS connections.
Secure Storage	Infineon Optiga SLB 9670 TPM 2.0	Storage on the IoT device that is designed to be protected from unauthorized access and capable of detecting attempts to hack or modify its contents. Used to store and process private keys and other information that must be kept confidential.
Certificate Authority	TrustNetZ demo manufacturer CA (MPR – manufacture provisioning root) TrustNetZ Domain CA	Two CA are used in Build 5 Domain CA issues certificates and provides signing and attestation functions that model network owner relationships (e.g. sign the LDevID certificate) Manufacturer CA issues the IDevID certificates; proving the device has been created by the manufacturer.
Application-Layer Onboarding Service	TrustNetZ Demo application sever	After connecting to the network, the device mutually authenticates with a trusted application service and interacts with it at the application layer. The IDevID and TPM private key are used to establish a TLS session with the demonstration application server and send data to it from the device. This demonstrates the concept of secure connection to a third-party application server using the cryptographic artifacts from the onboarding process.

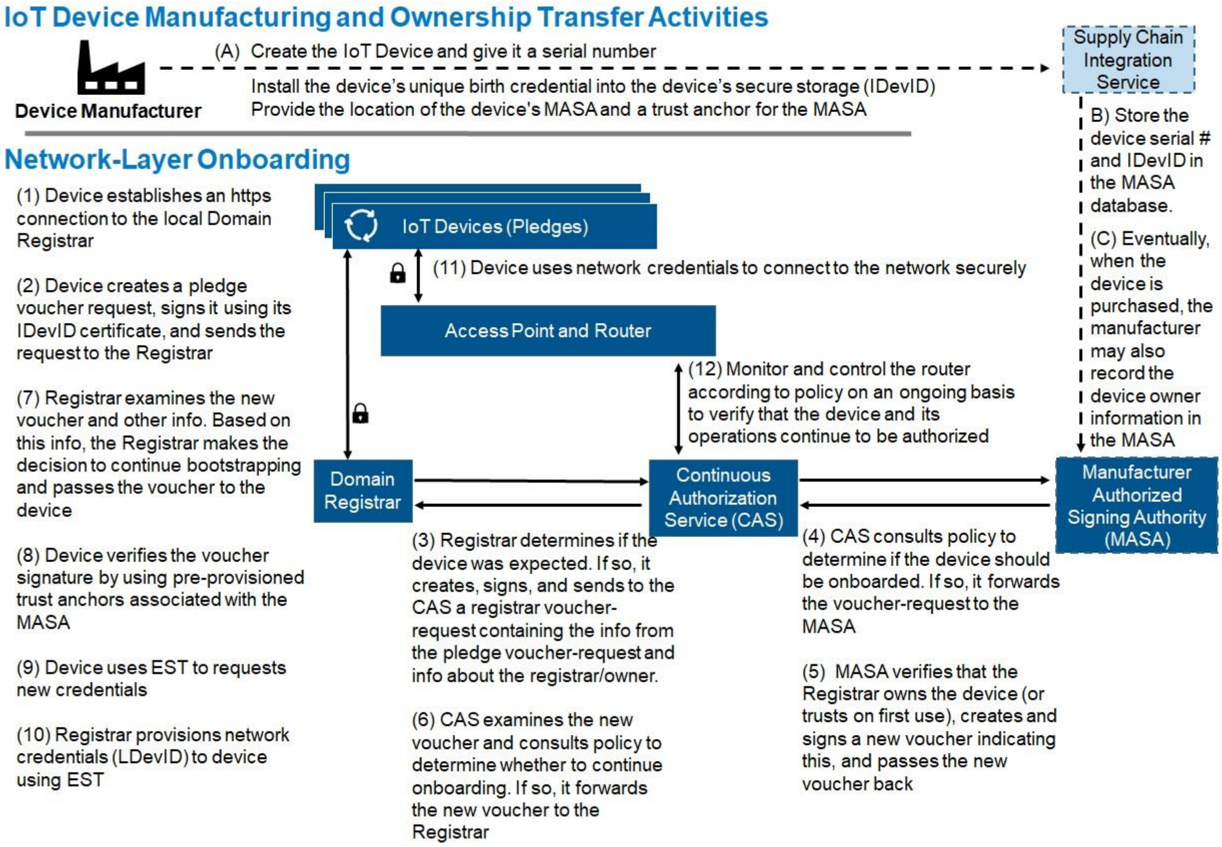
Component	Product	Function
Ongoing Device Authorization	Continuous Authorization Service, which calls into the in the TrustNetZ policy engine	Designed to perform a set of ongoing, policy-based continuous assurance and authorization checks on the device after it has connected to the network. As of this publication, the following ongoing checks have been implemented: ▪ The manufacturer of the device must be trusted by the network owner ▪ The device must be trusted by a user with appropriate privileges ▪ The device must have an associated device type ▪ The vulnerability score of the software bill of materials (SBOM) for the device type must be lower than a set threshold ▪ The device must not have contacted an IP address that is on a deny list If it fails any of these periodic checks, its voucher is revoked, which removes the device from the network.
Manufacturer Factory Provisioning Process	BRSKI Factory Provisioning Process used to provision the Infineon TPM with its private key and IDevID (See Appendix H.3)	Manufactures the IoT device. Creates, signs, and installs the device's unique identity (i.e., its IDevID, which is an X.509 certificate) into secure storage. Installs information the device requires for application-layer onboarding. Populates the MASA with information regarding devices that are created and, when the devices are sold, may record what entity owns them.

G.2 Build 5 Architecture

G.2.1 Build 5 Logical Architecture

The network-layer onboarding steps that are performed in Build 5 are depicted in Figure G-1. These steps are broken into two main parts: those required to transfer device bootstrapping information from the device manufacturer to the MASA (labeled with letters) and those required to perform network-layer onboarding of the device and establish the operation of the continuous authorization service (labeled with numbers).

Figure G-1 Logical Architecture of Build 5


IoT Device Manufacturing and Ownership Transfer Activities
Device Manufacturer
(A) Create the IoT Device and give it a serial number
Install the device's unique birth credential into the device's secure storage (IDevID)
Provide the location of the device's MASA and a trust anchor for the MASA
Supply Chain Integration Service
B) Store the device serial # and IDevID in the MASA database.
(C) Eventually, when the device is purchased, the manufacturer may also record the device owner information in the MASA
Network-Layer Onboarding
(1) Device establishes an https connection to the local Domain Registrar
(2) Device creates a pledge voucher request, signs it using its IDevID certificate, and sends the request to the Registrar
(7) Registrar examines the new voucher and other info. Based on this info, the Registrar makes the decision to continue bootstrapping and passes the voucher to the device
(8) Device verifies the voucher signature by using pre-provisioned trust anchors associated with the MASA
(9) Device uses EST to requests new credentials
(10) Registrar provisions network credentials (LDevID) to device using EST
IoT Devices (Pledges)
(11) Device uses network credentials to connect to the network securely
Access Point and Router
(12) Monitor and control the router according to policy on an ongoing basis to verify that the device and its operations continue to be authorized
Domain Registrar
Continuous Authorization Service (CAS)
Manufacturer Authorized Signing Authority (MASA)
(3) Registrar determines if the device was expected. If so, it creates, signs, and sends to the CAS a registrar voucher-request containing the info from the pledge voucher-request and info about the registrar/owner.
(6) CAS examines the new voucher and consults policy to determine whether to continue onboarding. If so, it forwards the new voucher to the Registrar
(4) CAS consults policy to determine if the device should be onboarded. If so, it forwards the voucher-request to the MASA
(5) MASA verifies that the Registrar owns the device (or trusts on first use), creates and signs a new voucher indicating this, and passes the new voucher back

The device manufacturer:

1. Creates the device and installs a unique serial number and birth credential into secure storage on the device. This unique birth credential takes the form of a private key and its associated 802.1AR certificate, e.g., the device's IDevID. As part of this factory-installed certificate process, the location of the device's manufacturer authorized signing authority (MASA) is provided in an extension to the IDevID. The device is also provided with trust anchors for the MASA entity that will sign the returned vouchers.
2. Stores information about the device, such as its serial number and its IDevID, in the MASA's database.
3. Eventually, when the device is sold, the MASA may also record the device ownership information in its database.

After obtaining the device, the device owner provisions the device with its network credentials by performing the following network-layer onboarding steps:

1. The owner puts the device (i.e., the pledge) into onboarding mode. The device establishes an https connection to the local Domain Registrar. (In a standard BRSKI implementation, the device would have wired network connectivity. The device would use its link-local network connectivity to locate a join proxy, and the join proxy would provide the device with https connectivity to the

local Domain Registrar.) The Build 5 implementation, however, relies on wireless connectivity and initially uses the unauthenticated EAP-TLS protocol. The pledge discovers potential onboarding networks by searching for public Wi-Fi networks that either match a particular SSID wildcard name or that advertise a particular realm. When the device finds a potential onboarding network, it connects to it and attempts to discover the registrar. The pledge will connect to the open Wi-Fi network and will receive either an IPv4 or IPv6 address. Subsequently, the pledge will listen to mDNS packets and will obtain the list of join proxies (IP addresses). Finally, the pledge will subsequently connect to each join proxy using the BRSKI-EST protocol.

2. The device creates a pledge voucher-request that includes the device serial number, signs this request with its IDevID certificate (i.e., its birth credential), and sends this signed request to the Registrar.

3. The Registrar receives the pledge voucher-request and considers whether the manufacturer is known to it and whether devices of that type are welcome. If so, the Registrar forms a registrar voucher-request that includes all the information from the pledge voucher request along with information about the registrar/owner. The Registrar sends this registrar voucher-request to the Continuous Authorization Service.

4. The Continuous Authorization Service consults policy to determine if this device should be permitted to be onboarded and what other conditions should be enforced. An example of policy that might be used is that the network owner wants to disable MASA validation. Assuming the device is permitted to be onboarded, the Continuous Authorization Service locates the MASA that the IoT device is known to trust (i.e., the MASA that is identified in the device's IDevID extension) and sends the registrar voucher-request to the MASA.

5. The MASA consults the information that it has stored and applies policy to determine whether to approve the Registrar's claim that it owns the device. (For example, the MASA may consult sales records for the device to verify device ownership, or it may be configured to trust that the first registrar that contacts it on behalf of a given device is in fact the device owner). Assuming the MASA decides to approve the Registrar's claim to own the device, the MASA creates a new voucher that directs the device to accept its new owner, signs this voucher, and sends it back to the Continuous Authorization Service.

6. The Continuous Authorization Service receives this new voucher and examines it in consultation with policy to determine whether to continue onboarding. Some examples of policies that might be used include: permit onboarding only if no current critical vulnerabilities have been disclosed against the declared device type, the device instance has successfully passed a site-specific test process, or a test compliance certificate has been found for the declared device type. Assuming the device is permitted to be onboarded, the Continuous Authorization Service sends the new voucher to the Domain Registrar.

7. The Domain Registrar receives and examines the new voucher along with other related information and determines whether it trusts the voucher. Assuming it trusts the voucher, the Registrar passes the voucher to the device.

8. The device uses its factory-provisioned MASA trust anchors to verify the voucher signature, thereby ensuring that the voucher can be trusted.

9. The device uses Enrollment over Secure Transport (EST) to request new credentials.

10. The Registrar provisions network credentials to the device using EST. These network credentials get stored into secure storage on the device, e.g., as an LDevID.

11. The device uses its newly provisioned network credentials to connect to the network securely.

12. After the device is connected and begins operating on the network, the Continuous Authorization Service and the router make periodic asynchronous calls to each other that enable the Continuous Authorization Service to monitor device behavior and constrain communications to and from the device as needed in accordance with policy. In this manner, the Continuous Authorization Service interacts with the router on an ongoing basis to verify that the device and its operations continue to be authorized throughout the device's tenure on the network.

This completes the network-layer onboarding process for Build 5 as well as the initialization of the Build 5 continuous authorization service. More details regarding the Build 5 implementation can be found at https://trustnetz.nqm.ai/docs/.

G.2.2 Build 5 Physical Architecture

Section 5.6 describes the physical architecture of Build 5.

Appendix H Factory Provisioning Process

H.1 Factory Provisioning Process

The Factory Provisioning Process creates and provisions a private key into the device’s secure storage; generates and signs the device’s certificate (when BRSKI is supported), generates the device’s DPP URI (when Wi-Fi Easy Connect is supported), or generates other bootstrapping information (when other trusted network-layer onboarding protocols are supported); provisions the device’s certificate, DPP URI, or other bootstrapping information onto the device; and sends the device’s certificate, DPP URI, or other bootstrapping information to the manufacturer’s database, which will eventually make this information available to the device owner to use during network-layer onboarding.

H.1.1 Device Birth Credential Provisioning Methods

There are various methods by which a device can be provisioned with its private key and bootstrapping information (e.g., its certificate, DPP URI, etc.) depending on how, where, and by what entity the public/private key pairs are generated [14]. Additional methods are also possible depending on how the device’s certificate is provided to the manufacturer’s database. The following are high-level descriptions of five potential methods for provisioning device birth credentials during various points in the device lifecycle. These methods are not intended to be exhaustive:

1. **Method 1: Key Pair Generated on IoT Device**
   Summary: Generate the private key on the device; device sends the device’s bootstrapping information (e.g., the device’s certificate or DPP URI) to the manufacturer’s database. The steps for Method 1 are:
   a. The public/private key pair is generated on the device and stored in secure storage.
   b. The device generates and signs a CSR structure and sends the CSR to the manufacturer’s IDevID CA, which sends a signed certificate (IDevID) back to the device.
   c. If BRSKI is being supported, the device loads the certificate (IDevID) into its secure storage; if Wi-Fi Easy Connect is being supported, the device creates a DPP URI and loads that into secure storage.
   d. The device sends the certificate or DPP URI to the manufacturer’s database.

   One disadvantage of this method is that the device’s random number generator is being relied upon to generate the key pair, and it is possible that a device’s random number generator will not be as robust as the random number generator that would be included in an SE, for example. An advantage of this method is that the device’s private key is not vulnerable to disclosure, assuming the device is equipped with a strong random number generator that is used for key generation and the private key is put into secure storage immediately upon generation.

2. **Method 2: Key Pair Generated in Secure Element**
   Summary: Generate the private key in a secure element on the device; IDevID CA provides the device certificate to the manufacturer’s database. The steps for Method 2 are:
   a. The public/private key pair is generated within the device’s SE.

   b. The device generates a CSR structure, the SE signs it, and the device sends the CSR to the manufacturer's IDevID CA, which sends a signed certificate (IDevID) back to the device.
   c. If BRSKI is being supported, the device loads the certificate (IDevID) into its secure storage; if Wi-Fi Easy Connect is being supported, the device creates a DPP URI and loads that into secure storage.
   d. The IDevID CA provides the certificate to the manufacturer's database. The manufacturer stores either the certificate (i.e., if BRSKI is being supported), or creates and stores a DPP URI (i.e., if Wi-Fi Easy Connect is being supported).

Method 2 is similar to Method 1 except that in method 2, the key pair is generated and stored in a secure element and the manufacturer's database receives the signed certificate directly from the CA (either via a push or a pull) rather than via the device. An advantage of method 2 is that the device's private key is not vulnerable to disclosure because secure elements are normally equipped with a strong random number generator and tamper-proof storage.

3. **Method 3: Key Pair Loaded into IoT Device**

Summary: Generate the private key in the device factory and load it onto the device. The steps for Method 3 are:

   a. The public/private key pairs and certificates are generated in advance at the device factory and recorded in the manufacturer's database.
   b. The public/private key pair and certificate are loaded onto the device at the device factory.

One advantage of this method is that there is no need to trust the random number generator on the device to generate strong public/private key pairs. However, the private keys may be vulnerable to disclosure during the period of time before they are provisioned into secure storage on the devices (and afterwards if they are not deleted once they have been copied into secure storage).

4. **Method 4: Key Pair Pre-Provisioned onto Secure Element**

Summary: Generate the private key in the SE and load the certificate on the device at the SE factory (SEF). The steps for Method 4 are:

   a. The public/private key pair and certificate are generated in advance in the SE at the SEF and the public key is recorded.
   b. The certificate is loaded onto the devices at the SEF.
   c. The certificates and the serial numbers of their corresponding devices are provided to the device manufacturer, and the device manufacturer can put them into the manufacturer database.
   d. The CA that signs the certificates that are generated and loaded onto the SEs may come from either the SEF or the device manufacturer. (Note: the CA is likely not located at the factory, which may be offshore.)

Additional trust anchors can also be loaded into the SE at the SEF (e.g., code signing keys, server public keys for TLS connections, etc.) As with methods 2 and 3, one advantage of this method (method 4) is that there is no need to trust the random number generator on the device to generate strong public/private key pairs because the random number generator on the SE is used instead. With this method, the security level of the manufacturer's factory does not need to be as high as that of the SEF because all key generation and certificate signing is performed at the SEF; the manufacturer can rely on the security of the SEF, which can be advantageous to the device manufacturer, assuming that the SEF is in fact secure.

5. **Method 5: Private Key Derived from Shared Seed**

Summary: The device's private key is derived from a shared seed. The steps for Method 5 are:

a. The chip vendor embeds a random number into each IoT device (e.g., this may be burned into fuses on the IoT device, inside the Trusted Execution Environment (TEE)).
b. The IoT device manufacturer gets a copy of this seed securely (e.g., on a USB device that is transported via trusted courier).
c. On first boot, the IoT device generates a private key from this seed.
d. The manufacturer uses the same seed to generate a public key and signs a certificate.

As with method 4, with this option (method 5), there is no need for the IoT device manufacturer to have a secure factory because the IoT device manufacturer may rely on the security of the chip manufacturer. However, the IoT device manufacturer must also rely on the security of the courier or other mechanism that is delivering the seed, and the IoT device manufacturer must ensure that the value of this seed is not disclosed.

H.2 Factory Provisioning Builds – General Provisioning Process

The Factory Provisioning Builds implemented as part of this project simulate activities performed during the IoT device manufacturing process to securely provision the device's birth credentials (i.e., its private key) into secure storage on the device and make the device's network-layer bootstrapping information available by enrolling the device's public key into a database that will make this public key accessible to the device owner in a form such as a certificate or DPP URI. The method used in the factory provisioning builds most closely resembles *Method 2: Key Pair Generated on IoT Devic*e, as described in Section H.1.1.

There are several different potential versions of the factory provisioning build architecture depending on whether the credentials being generated are designed to support BRSKI, Wi-Fi Easy Connect, Thread, or some other trusted network-layer onboarding protocol. For example, when BRSKI is being supported, the device bootstrapping information that is created takes the form of an 802.1AR certificate (IDevID); if DPP is supported, it takes the form of a DPP URI.

Because this project does not have access to a real factory or the tools necessary to provision birth credentials directly into device firmware, the factory builds simulate the firmware loading process by loading factory provisioning code into the IoT device (e.g., a Raspberry Pi device). This code plays the role of the factory in the builds by instructing the SE that is attached to the IoT device to generate the device's private key and bootstrapping information. Once the IoT device has been provisioned with its birth credentials in this manner, it can, in theory, be network-layer onboarded to one of the project build networks.

H.3 BRSKI Factory Provisioning Builds (NquiringMinds and SEALSQ)

Two variants of the BRSKI Factory Provisioning Build were implemented:

- **NquiringMinds and SEALSQ implementation** (first version): SEALSQ, a subsidiary of WISeKey, and NquiringMinds collaborated to implement one version of the BRSKI Factory Provisioning Build. This build is designed to provision birth credentials to a Raspberry Pi device that has an attached secure element provided by SEALSQ.
- **NquiringMinds and Infineon implementation** (second version): NquiringMinds implemented a second version of the BRSKI Factory Provisioning Build using an Infineon SE. This build is designed to provision birth credentials to a Raspberry Pi device that has an attached Infineon Optiga SLB 9670 TPM 2.0.

H.3.1 BRSKI Factory Provisioning Build Technologies

The general infrastructure for the first version of the BRSKI Factory Provisioning Build (i.e., the NquiringMinds and SEALSQ implementation) is provided by SEALSQ. The first version of the BRSKI Factory Provisioning Build infrastructure consists of:

- A SEALSQ VaultIC SE that is attached to the Raspberry Pi
- SEALSQ Factory Provisioning Code that is located on an SD card and that communicates with the chip in the SE to
  - create a P-256 Elliptic Curve public/private key pair within the SE,
  - construct a certificate signing request, and
  - store the certificate in the SE as well as send it to the manufacturer's database
- SEALSQ INeS CMS CA, a certificate authority for signing the device's birth certificate

As mentioned earlier, separate factory provisioning builds are required for each network-layer onboarding protocol being supported. A small amount of factory provisioning code is required to be customized for each build, depending on the onboarding protocol that is supported and how the bootstrapping information will be provided to the manufacturer. In this build, NquiringMinds provided this code and made it available to the Raspberry Pi IoT device by placing it on an SD card. (This could be either in a partition of the SD card that holds the device's BRSKI onboarding software or on a separate SD card altogether).

Table H-1 lists the technologies used in the first version of the BRSKI Factory Provisioning Build. It lists the products used to instantiate each logical build component and the security function that the component provides. The components listed are logical. They may be combined in physical form, e.g., a single piece of hardware may both generate key pairs and provide secure storage.

Table H-1 First Version of the BRSKI Factory Provisioning Build Products and Technologies

Component	Product	Function
Key Pair Generation Component	SEALSQ VaultIC and associated provisioning code	Generates and installs the public/private key pair into secure storage. The VaultIC has a SP800-90B certified random number generator for key pair generation.

Component	Product	Function
		[15][16][17] Signs the certificate signing request that is sent to the CA.
Secure Storage	SEALSQ VaultIC	Storage on the IoT device that is designed to be protected from unauthorized access and capable of detecting attempts to hack or modify its contents. Used to generate, store, and process private keys, credentials, and other information that must be kept confidential.
General Factory Provisioning Instructions	SEALSQ Factory Provisioning Code	Creates a CSR associated with the key pair, installs the signed certificate into secure storage. Creates a record of devices that it has created and their certificates.
Build-specific Factory Provisioning Instructions	NquiringMinds Factory Provisioning Code	Sends device ownership information and the certificate received by the General Factory Provisioning code to the MASA.
Manufacturer Database	MASA	When devices are manufactured, device identity and bootstrapping information is stored here by the manufacturer. Eventually, this database makes the device's bootstrapping information available to the device owner. Device bootstrapping information is information that the device owner requires to perform trusted network-layer onboarding; for BRSKI, the bootstrapping information is a signed certificate that is sent to the MASA, along with information regarding the device's owner.
Certificate Authority (CA)	SEALSQ INeS CMS CA	Issues and signs certificates as needed.

The second version of the BRSKI Factory Provisioning Build (i.e., the NquiringMinds implementation with an Infineon SE) infrastructure consists of:

- An Infineon Optiga SLB 9670 TPM 2.0. that is attached to the Raspberry Pi
- Factory Provisioning Code written by NquiringMinds that is located on an SD card and that communicates with the chip in SE to
    - create a P-256 Elliptic Curve public/private key pair within the SE,
    - construct a certificate signing request, and
    - store the certificate in the SE as well as send it to the manufacturer's database
- NquiringMinds Manufacturer Provisioning Root (MPR) server, which signs the device's IDevID birth certificate. It sits in the cloud and is securely contacted using the keys in the Infineon Optiga secure element.

In this build, NquiringMinds provided all of the factory provisioning code and made it available to the Raspberry Pi IoT device by placing it on an SD card. (This could be either in a partition of the SD card that holds the device's BRSKI onboarding software or on a separate SD card altogether).

Table H-2 lists the technologies used in the second version of the BRSKI Factory Provisioning Build. It lists the products used to instantiate each logical build component and the security function that the component provides. The components listed are logical. They may be combined in physical form, e.g., a single piece of hardware may both generate key pairs and provide secure storage.

Table H-2 Second Version of the BRSKI Factory Provisioning Build Products and Technologies

Component	Product	Function
Key Pair Generation Component	Infineon TPM and associated provisioning code	Generates and installs the public/private key pair into secure storage. Signs the certificate signing request that is sent to the CA.
Secure Storage	Infineon TPM	Storage on the IoT device that is designed to be protected from unauthorized access and capable of detecting attempts to hack or modify its contents. Used to generate, store, and process private keys, credentials, and other information that must be kept confidential.
General Factory Provisioning Instructions	Infineon TPM-specific Factory Provisioning Code	Creates a CSR associated with the key pair, installs the signed certificate into secure storage. Creates a record of devices that it has created and their certificates.
Build-specific Factory Provisioning Instructions	Build-specific Factory Provisioning Code	Sends device ownership information and the signed certificate to the MASA.
Manufacturer Database	MASA	When devices are manufactured, device identity and bootstrapping information is stored here by the manufacturer. Eventually, this database makes the device's bootstrapping information available to the device owner. Device bootstrapping information is information that the device owner requires to perform trusted network-layer onboarding; for BRSKI, the bootstrapping information is a signed certificate that is sent to the MASA, along with information regarding the device's owner.
Certificate Authority (CA)	SEALSQ INeS CMS CA NquiringMinds On-premises CA	Issues and signs certificates as needed.

H.3.2 BRSKI Factory Provisioning Build Logical Architectures

Figure H-1 depicts the logical architecture of the first version of the BRSKI factory provisioning build (i.e., the NquiringMinds and SEALSQ implementation) and is annotated with the steps that are performed in this build to prepare IoT devices for network-layer onboarding using the BRSKI protocol. Figure H-1 shows a Raspberry Pi device with a SEALSQ VaultIC SE attached. An SD card that contains factory provisioning code provided by SEALSQ and NquiringMinds is also required. To perform factory provisioning using this build, insert the SD card into the Raspberry Pi, as depicted (or activate the code in the factory provisioning partition of the SD card that is already in the Raspberry Pi). The SEALSQ software will boot up and perform the following steps to simulate the activities of a factory:

1. Instruct the SE to generate and store a private/public key pair
2. Create a certificate signing request for this key pair and have the SE sign it
3. Send the signed CSR to the IDevID CA (i.e., to the INeS CA that is operated by SEALSQ)
4. Receive back the signed certificate from the CA
5. Load the certificate into the SE
6. Send the certificate (along with device ownership information) to the manufacturer's database, which in this case is the MASA that is trusted by the owner

This completes the steps performed as part of the first version of the BRSKI Factory Provisioning Build. Once complete, shipment of the device to its owner can be simulated by walking the device across the room in the NCCoE laboratory to the Build 5 (NquiringMinds) implementation and replacing the SD card that has the factory provisioning code on it with and SD card that has the BRSKI onboarding code on it. (Alternatively, if the factory provisioning code and the BRSKI onboarding code are stored in separate partitions of the same SD card, shipment of the device to its owner can be simulated by booting up the code in the onboarding partition.) Build 5 is designed to execute this BRSKI onboarding software, which onboards the device to the device owner's network by provisioning the device with an LDevID that will serve as its network-layer credential. Such successful network-layer onboarding of the newly provisioned device using the BRSKI protocol by Build 5 would serve to confirm that the first version of the BRSKI factory provisioning process successfully provisioned the device with its birth credentials. At the time of this writing, however, this confirmation process was not able to be performed. In order to securely network-layer onboard the newly provisioned Raspberry Pi using the BRSKI protocol, the Raspberry Pi's onboarding software would need to be written to use the private key stored in the SEALSQ secure element when running the BRSKI protocol. Such software was not yet available at the time of this publication. The BRSKI onboarding code on the Raspberry Pi does not currently use the private key stored in the SEALSQ SE. As a result, Build 5 was not able to onboard this factory Pi as a way of confirming that the first version of the BRSKI factory build process completed successfully. The repository that hosts the code for this implementation can be found here at the trustnetz-se Github repository.

Figure H-1 Logical Architecture of the First Version of the BRSKI Factory Provisioning Build

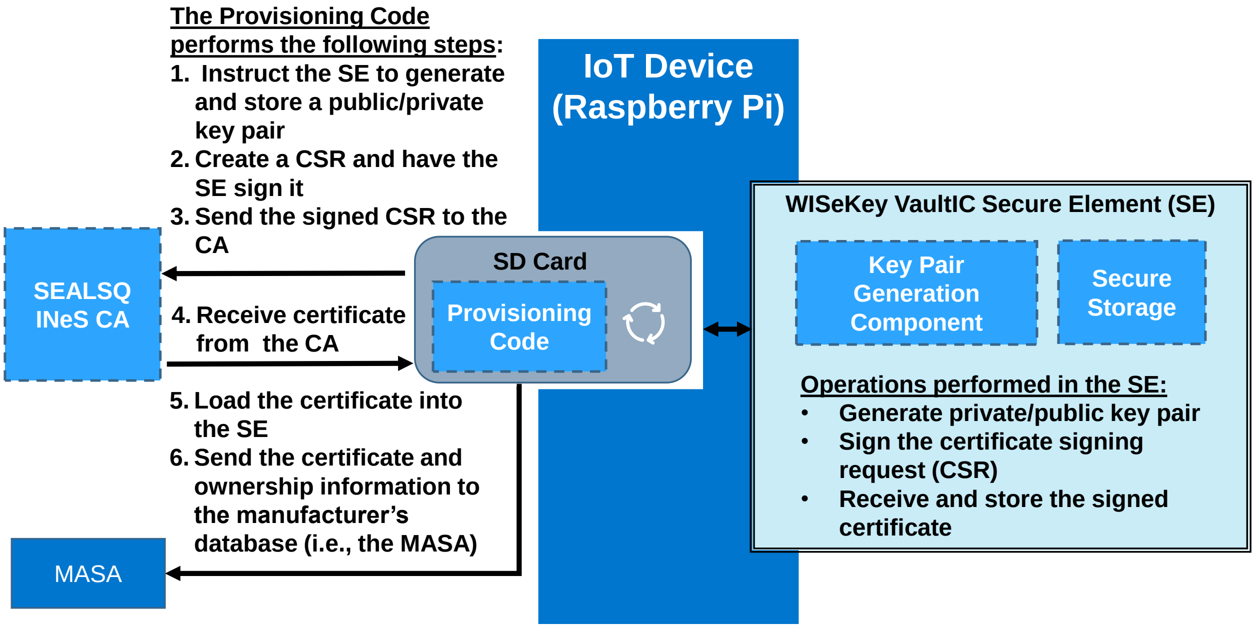
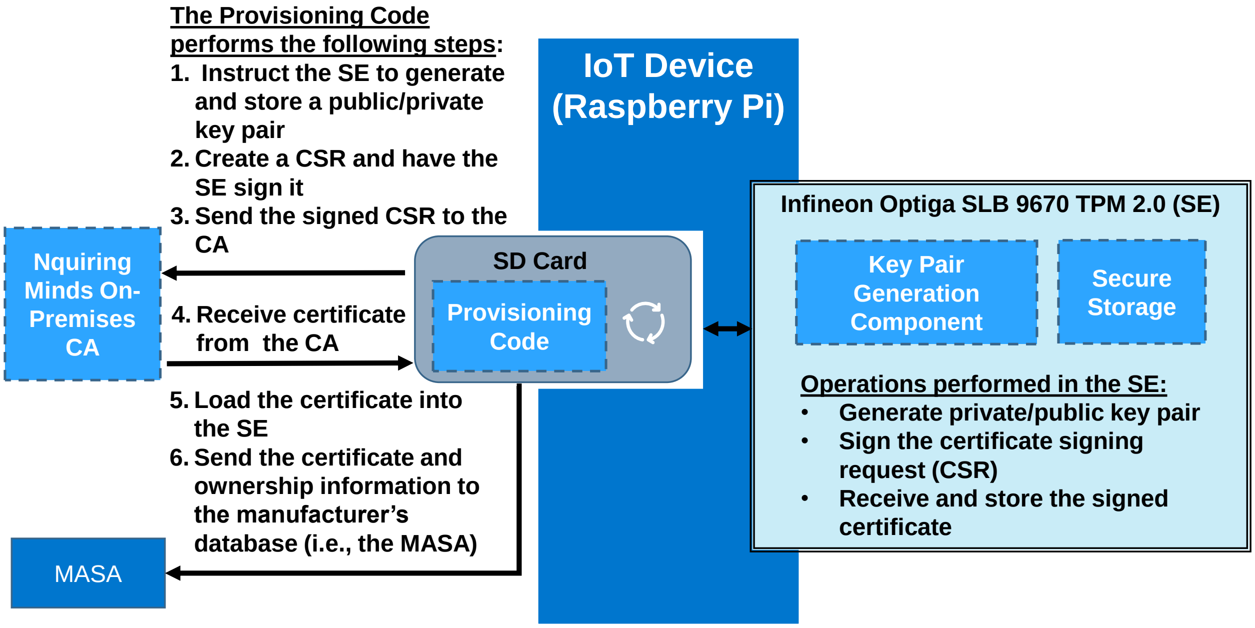

The Provisioning Code performs the following steps:
1. Instruct the SE to generate and store a public/private key pair
2. Create a CSR and have the SE sign it
3. Send the signed CSR to the CA
4. Receive certificate from the CA
5. Load the certificate into the SE
6. Send the certificate and ownership information to the manufacturer's database (i.e., the MASA)
IoT Device (Raspberry Pi)
SEALSQ INeS CA
SD Card
Provisioning Code
MASA
WISeKey VaultIC Secure Element (SE)
Key Pair Generation Component
Secure Storage
Operations performed in the SE:
• Generate private/public key pair
• Sign the certificate signing request (CSR)
• Receive and store the signed certificate

Figure H-2 depicts the logical architecture of the second version of the BRSKI factory provisioning build and is annotated with the steps that are performed in this build to prepare IoT devices for network-layer onboarding using the BRSKI protocol. Figure H-2 shows a Raspberry Pi device with an Infineon Optiga SLB 9670 TPM 2.0 SE attached. An SD card that contains factory provisioning code provided by NquiringMinds is also required. To perform factory provisioning using this build, insert the SD card into the Raspberry Pi, as depicted (or activate the code in the factory provisioning partition of the SD card that is already in the Raspberry Pi). The factory provisioning code software will boot up and perform the following steps to simulate the activities of a factory:

1. Instruct the Infineon SE to generate and store a private/public key pair
2. Create a certificate signing request for this key pair and have the SE sign it
3. Send the signed CSR to the IDevID CA (i.e., to the NquiringMinds on-premises CA/Manufacturer Provisioning Root)
4. Receive back the signed certificate from the CA
5. Load the certificate into the SE
6. Send the certificate (along with device ownership information) to the manufacturer's database, which in this case is the MASA that is trusted by the owner

This completes the steps performed as part of the second version of the BRSKI Factory Provisioning Build. Once complete, shipment of the device to its owner can be simulated by walking the device across the room in the NCCoE laboratory to the Build 5 (NquiringMinds) implementation and replacing the SD card that has the factory provisioning code on it with and SD card that has the BRSKI onboarding code on it. (Alternatively, if the factory provisioning code and the BRSKI onboarding code are stored in separate partitions of the same SD card, shipment of the device to its owner can be simulated by booting up the code in the onboarding partition.) Build 5 executes a modification of the BRSKI onboarding software that has been modified to use the IDevID resident on the Infineon TPM throughout the protocol flow, ensuring the device's IDevID's private key is never made public and never leaves the secure element. Specifically, the critical signing operations and the TLS negotiation steps are fully secured by the SE. The full BRSKI onboarding flow provisions a new LDevID onto the device. This LDevID provides the secure method for the device to connect to the domain owner's network. This successful network-layer onboarding of the IoT device by Build 5 serves as confirmation that the second version of the BRSKI factory provisioning process successfully provisioned the device with its birth credentials.

Figure H-2 Logical Architecture of the Second Version of the BRSKI Factory Provisioning Build


The Provisioning Code performs the following steps:
1. Instruct the SE to generate and store a public/private key pair
2. Create a CSR and have the SE sign it
3. Send the signed CSR to the CA
4. Receive certificate from the CA
5. Load the certificate into the SE
6. Send the certificate and ownership information to the manufacturer's database (i.e., the MASA)
Nquiring Minds On-Premises CA
MASA
IoT Device (Raspberry Pi)
SD Card
Provisioning Code
Infineon Optiga SLB 9670 TPM 2.0 (SE)
Key Pair Generation Component
Secure Storage
Operations performed in the SE:
• Generate private/public key pair
• Sign the certificate signing request (CSR)
• Receive and store the signed certificate

H.3.3 BRSKI Factory Provisioning Build Physical Architectures

Section 5.6.1 describes the physical architecture of the BRSKI Factory Provisioning Builds.

H.4 Wi-Fi Easy Connect Factory Provisioning Build (SEALSQ and Aruba/HPE)

SEALSQ, a subsidiary of WISeKey, and Aruba/HPE implemented a Wi-Fi Easy Connect Factory Provisioning Build. This build is designed to provision birth credentials to a Raspberry Pi device that has an attached secure element provided by SEALSQ.

H.4.1 Wi-Fi Easy Connect Factory Provisioning Build Technologies

The general infrastructure for the Wi-Fi Easy Connect Factory Provisioning Build is provided by SEALSQ. The Wi-Fi Easy Connect Factory Provisioning Build infrastructure consists of:

- A SEALSQ VaultIC SE that is attached to the Raspberry Pi

- SEALSQ Factory Provisioning Code that is located on an SD card and that communicates with the chip in the SE to:
    - create a P-256 Elliptic Curve public/private key pair within the SE,
    - use the public key to construct a DPP URI
    - export the DPP URI and convert it into a QR code

Table H-3 lists the technologies used in the Wi-Fi Easy Connect Factory Provisioning Build. It lists the products used to instantiate each logical build component and the security function that the component provides. The components listed are logical. They may be combined in physical form, e.g., a single piece of hardware may both generate key pairs and provide secure storage.

Table H-3 Wi-Fi Easy Connect Factory Provisioning Build Products and Technologies

Component	Product	Function
Key Pair Generation Component	SEALSQ VaultIC and associated provisioning code	Generates and installs the public/private key pair into secure storage. The VaultIC has a SP800-90B certified random number generator for key pair generation. [17]
Secure Storage	SEALSQ VaultIC	Storage on the IoT device that is designed to be protected from unauthorized access and capable of detecting attempts to hack or modify its contents. Used to generate, store, and process private keys, credentials, and other information that must be kept confidential.
General Factory Provisioning Instructions	SEALSQ Factory Provisioning Code	Creates a public/private key pair.
Build-specific Factory Provisioning Instructions	Aruba/HPE Factory Provisioning Code	Uses the public key to create a DPP URI. Exports the DPP URI and converts it into a QR code.
Manufacturer Database	Manufacturer cloud or imprint on device	The DPP URI information is stored in the QR code and is the mechanism for conveying the device’s bootstrapping information to the device owner.

H.4.2 Wi-Fi Easy Connect Factory Provisioning Build Logical Architecture

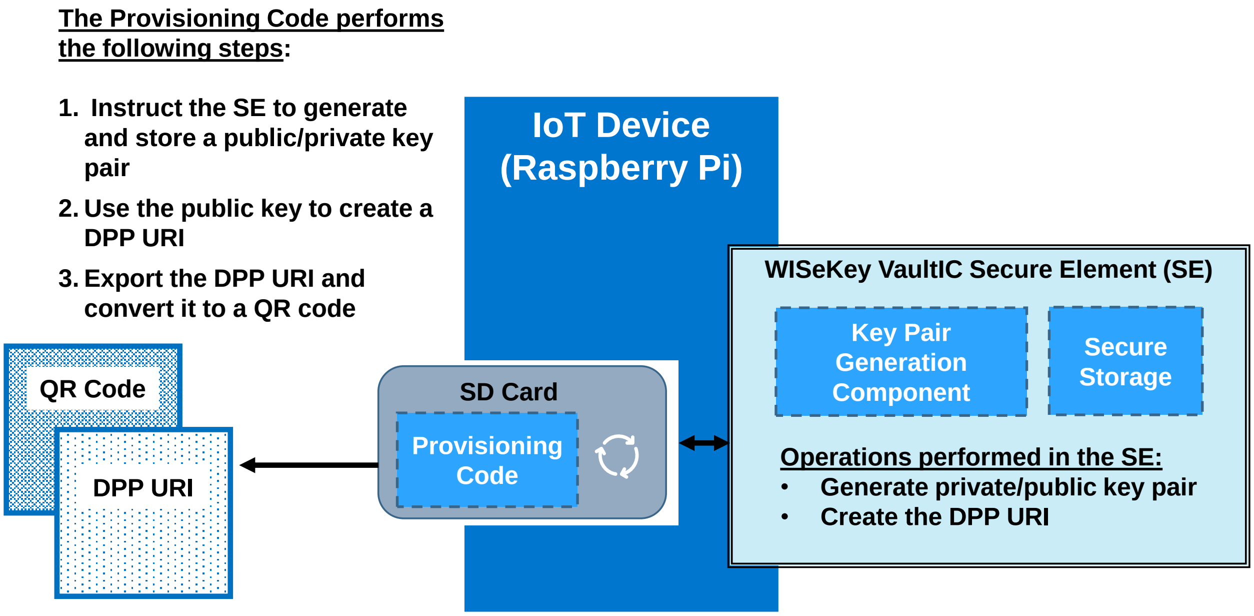
Figure H-3 depicts the logical architecture of the Wi-Fi Easy Connect factory provisioning build and is annotated with the steps that are performed in this build to prepare Raspberry Pi IoT devices for network-layer onboarding using the Wi-Fi Easy Connect protocol. Figure H-3 shows a Raspberry Pi device with a SEALSQ VaultIC SE attached. Factory provisioning code provided by SEALSQ and Aruba/HPE must also be loaded. In Figure H-3, this code is shown as being on an SD card. The factory provisioning software will boot up and perform the following steps to simulate the activities of a factory:

1. Instruct the SE to generate and store a private/public key pair
2. Use the public key to create a DPP URI

3. Export the DPP URI and convert it into a QR code

This completes the steps performed as part of the Wi-Fi Easy Connect Factory Provisioning Build. Once complete, shipment of the device to its owner can be simulated by walking the device across the room in the NCCoE laboratory to the Build 1 (Aruba/HPE) implementation. Build 1 uses the Wi-Fi Easy Connect protocol to network-layer onboard the device to the device owner's network by provisioning the device with connector that will serve as its network-layer credential. Successful network-layer onboarding of the newly provisioned device using the Wi-Fi Easy Connect protocol by Build 1 would serve to confirm that the Wi-Fi Easy Connect factory provisioning process correctly provisioned the device with its birth credentials. At the time of this writing, however, this confirmation process was not able to be performed. In order to securely network-layer onboard the newly provisioned Raspberry Pi using the Wi-Fi Easy Connect protocol, the Raspberry Pi would need to be equipped with a firmware image that uses the private key stored in the secure element when running the Wi-Fi Easy Connect protocol. Such firmware was not yet available at the time of this publication. The Wi-Fi Easy Connect code on the Raspberry Pi does not use the private key stored in the SE at this time. Confirmation that the factory build process completed successfully is limited to inspection of the .PNG file and .URI file that were created to display the QR Code and the device's DPP URI, respectively.

Figure H-3 Logical Architecture of the Wi-Fi Easy Connect Factory Provisioning Build


The Provisioning Code performs the following steps:
1. Instruct the SE to generate and store a public/private key pair
2. Use the public key to create a DPP URI
3. Export the DPP URI and convert it to a QR code
IoT Device (Raspberry Pi)
QR Code
DPP URI
SD Card
Provisioning Code
WISeKey VaultIC Secure Element (SE)
Key Pair Generation Component
Secure Storage
Operations performed in the SE:
• Generate private/public key pair
• Create the DPP URI

H.4.3 Wi-Fi Easy Connect Factory Provisioning Build Physical Architecture

Section 5.2.1 describes the physical architecture of the Factory Provisioning Build.

Appendix I References

[1] L. S. Vailshery, “Number of Internet of Things (IoT) connected devices worldwide from 2019 to 2023, with forecasts from 2022 to 2030,” Statista, July 2023. Available: https://www.statista.com/statistics/1183457/iot-connected-devices-worldwide/.

[2] S. Symington, W. Polk, and M. Souppaya, *Trusted Internet of Things (IoT) Device Network-Layer Onboarding and Lifecycle Management (Draft)*, National Institute of Standards and Technology (NIST) Draft Cybersecurity White Paper, Gaithersburg, MD, Sept. 2020, 88 pp. https://doi.org/10.6028/NIST.CSWP.09082020-draft.

[3] E. Lear, R. Droms, and D. Romascanu, *Manufacturer Usage Description Specification,* IETF Request for Comments (RFC) 8520, March 2019. Available: https://tools.ietf.org/html/rfc8520.

[4] M. Souppaya et al, *Securing Small-Business and Home Internet of Things (IoT) Devices: Mitigating Network-Based Attacks Using Manufacturer Usage Description (MUD)*, National Institute of Standards and Technology (NIST) Special Publication (SP) 1800-15, Gaithersburg, Md., May 2021, 438 pp. Available: https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.1800-15.pdf.

[5] “National Cybersecurity Center of Excellence (NCCoE) Trusted Internet of Things (IoT) Device Network-Layer Onboarding and Lifecycle Management,” Federal Register Vol. 86, No. 204, October 26, 2021, pp. 59149-59152. Available: https://www.federalregister.gov/documents/2021/10/26/2021-23293/national-cybersecurity-center-of-excellence-nccoe-trusted-internet-of-things-iot-device.

[6] Wi-Fi Alliance, *Wi-Fi Easy Connect™ Specification Version 3.0*, 2022. Available: https://www.wi-fi.org/system/files/Wi-Fi_Easy_Connect_Specification_v3.0.pdf.

[7] M. Pritikin, M. Richardson, T.T.E. Eckert, M.H. Behringer, and K.W. Watsen, *Bootstrapping Remote Secure Key Infrastructure (BRSKI)*, IETF Request for Comments (RFC) 8995, October 2021. Available: https://datatracker.ietf.org/doc/rfc8995/.

[8] Thread 1.1.1 Specification, February 13, 2017.

[9] OpenThread Released by Google. Available: https://openthread.io/.

[10] O. Friel, E. Lear, M. Pritikin, and M. Richardson, *BRSKI over IEEE 802.11*, IETF Internet-Draft (Individual), July 2018. Available: https://datatracker.ietf.org/doc/draft-friel-brski-over-802dot11/01/.

[11] NIST. *The NIST Cybersecurity Framework (CSF) 2.0*. Available: https://doi.org/10.6028/NIST.CSWP.29.

[12] *IEEE Standard for Local and Metropolitan Area Networks - Secure Device Identity*, IEEE Std 802.1AR-2018 (Revision of IEEE Std 802.1AR-2009), 2 Aug. 2018, 73 pp. Available: https://ieeexplore.ieee.org/document/8423794.

[13] F. Stajano and R. Anderson, *The Resurrecting Duckling: Security Issues for Ad-hoc Wireless Networks*, B. Christianson, B. Crispo and M. Roe (Eds.). Security Protocols, 7th International Workshop Proceedings, Lecture Notes in Computer Science, 1999. Springer-Verlag Berlin Heidelberg 1999. Available: https://www.cl.cam.ac.uk/~fms27/papers/1999-StajanoAnd-duckling.pdf.

[14] M. Richardson, *A Taxonomy of operational security considerations for manufacturer installed keys and Trust Anchors*, IETF Internet-Draft (Individual), November 2022. Available: https://datatracker.ietf.org/doc/draft-richardson-t2trg-idevid-considerations/.

[15] Certificate #4302, Cryptographic Module Validation Program, NIST Computer Security Resource Center. Available: https://csrc.nist.gov/projects/cryptographic-module-validation-program/certificate/4302.

[16] Certificate #4303, Cryptographic Module Validation Program, NIST Computer Security Resource Center. Available: https://csrc.nist.gov/projects/cryptographic-module-validation-program/certificate/4303.

[17] Entropy Certificate #E2, Cryptographic Module Validation Program, NIST Computer Security Resource Center. Available: https://csrc.nist.gov/projects/cryptographic-module-validation-program/entropy-validations/certificate/2.

NIST SPECIAL PUBLICATION 1800-36C

Trusted Internet of Things (IoT) Device Network-Layer Onboarding and Lifecycle Management:

Enhancing Internet Protocol-Based IoT Device and Network Security

Volume C:
How-To Guides

Murugiah Souppaya
Paul Watrobski
National Institute of Standards and Technology
Gaithersburg, Maryland

Chelsea Deane
Joshua Klosterman
Blaine Mulugeta
Charlie Rearick
Susan Symington
The MITRE Corporation
McLean, Virginia

Dan Harkins
Danny Jump
Aruba, a Hewlett Packard
Enterprise Company
San Jose, California

Andy Dolan
Kyle Haefner
Craig Pratt
Darshak Thakore
CableLabs
Louisville, Colorado

Nick Allot
Ashley Setter
NquiringMinds
Southampton, United Kingdom

May 2024

DRAFT

This publication is available free of charge from
https://www.nccoe.nist.gov/projects/trusted-iot-device-network-layer-onboarding-and-lifecycle-management

DISCLAIMER

Certain commercial entities, equipment, products, or materials may be identified by name or company logo or other insignia in order to acknowledge their participation in this collaboration or to describe an experimental procedure or concept adequately. Such identification is not intended to imply special status or relationship with NIST or recommendation or endorsement by NIST or NCCoE; neither is it intended to imply that the entities, equipment, products, or materials are necessarily the best available for the purpose.

While NIST and the NCCoE address goals of improving management of cybersecurity and privacy risk through outreach and application of standards and best practices, it is the stakeholder's responsibility to fully perform a risk assessment to include the current threat, vulnerabilities, likelihood of a compromise, and the impact should the threat be realized before adopting cybersecurity measures such as this recommendation.

National Institute of Standards and Technology Special Publication 1800-36C, Natl. Inst. Stand. Technol. Spec. Publ. 1800-36C, 55 pages, May 2024, CODEN: NSPUE2

FEEDBACK

You can improve this guide by contributing feedback. As you review and adopt this solution for your own organization, we ask you and your colleagues to share your experience and advice with us.

Comments on this publication may be submitted to: iot-onboarding@nist.gov.

Public comment period: May 31, 2024 through July 30, 2024

All comments are subject to release under the Freedom of Information Act.

National Cybersecurity Center of Excellence
National Institute of Standards and Technology
100 Bureau Drive
Mailstop 2002
Gaithersburg, MD 20899
Email: nccoe@nist.gov

NATIONAL CYBERSECURITY CENTER OF EXCELLENCE

The National Cybersecurity Center of Excellence (NCCoE), a part of the National Institute of Standards and Technology (NIST), is a collaborative hub where industry organizations, government agencies, and academic institutions work together to address businesses' most pressing cybersecurity issues. This public-private partnership enables the creation of practical cybersecurity solutions for specific industries, as well as for broad, cross-sector technology challenges. Through consortia under Cooperative Research and Development Agreements (CRADAs), including technology partners—from Fortune 50 market leaders to smaller companies specializing in information technology security—the NCCoE applies standards and best practices to develop modular, adaptable example cybersecurity solutions using commercially available technology. The NCCoE documents these example solutions in the NIST Special Publication 1800 series, which maps capabilities to the NIST Cybersecurity Framework and details the steps needed for another entity to re-create the example solution. The NCCoE was established in 2012 by NIST in partnership with the State of Maryland and Montgomery County, Maryland.

To learn more about the NCCoE, visit https://www.nccoe.nist.gov/. To learn more about NIST, visit https://www.nist.gov.

NIST CYBERSECURITY PRACTICE GUIDES

NIST Cybersecurity Practice Guides (Special Publication 1800 series) target specific cybersecurity challenges in the public and private sectors. They are practical, user-friendly guides that facilitate the adoption of standards-based approaches to cybersecurity. They show members of the information security community how to implement example solutions that help them align with relevant standards and best practices, and provide users with the materials lists, configuration files, and other information they need to implement a similar approach.

The documents in this series describe example implementations of cybersecurity practices that businesses and other organizations may voluntarily adopt. These documents do not describe regulations or mandatory practices, nor do they carry statutory authority.

KEYWORDS

application-layer onboarding; bootstrapping; Internet of Things (IoT); Manufacturer Usage Description (MUD); network-layer onboarding; onboarding; Wi-Fi Easy Connect.

ACKNOWLEDGMENTS

We are grateful to the following individuals for their generous contributions of expertise and time.

Name	Organization
Amogh Guruprasad Deshmukh	Aruba, a Hewlett Packard Enterprise company
Bart Brinkman	Cisco
Eliot Lear	Cisco
Peter Romness	Cisco
Tyler Baker	Foundries.io
George Grey	Foundries.io
David Griego	Foundries.io
Fabien Gremaud	Kudelski IoT
Brecht Wyseur	Kudelski IoT
Faith Ryan	The MITRE Corporation
Toby Ealden	NquiringMinds
John Manslow	NquiringMinds
Antony McCaigue	NquiringMinds
Alexandru Mereacre	NquiringMinds
Loic Cavaille	NXP Semiconductors
Mihai Chelalau	NXP Semiconductors
Julien Delplancke	NXP Semiconductors
Anda-Alexandra Dorneanu	NXP Semiconductors
Todd Nuzum	NXP Semiconductors

Name	Organization
Nicusor Penisoara	NXP Semiconductors
Laurentiu Tudor	NXP Semiconductors
Michael Richardson	Sandelman Software Works
Karen Scarfone	Scarfone Cybersecurity
Steve Clark	SEALSQ, a subsidiary of WISeKey
Pedro Fuentes	SEALSQ, a subsidiary of WISeKey
Gweltas Radenac	SEALSQ, a subsidiary of WISeKey
Kalvin Yang	SEALSQ, a subsidiary of WISeKey
Mike Dow	Silicon Labs
Steve Egerter	Silicon Labs

The Technology Partners/Collaborators who participated in this build submitted their capabilities in response to a notice in the Federal Register. Respondents with relevant capabilities or product components were invited to sign a Cooperative Research and Development Agreement (CRADA) with NIST, allowing them to participate in a consortium to build this example solution. We worked with:

Technology Collaborators

Aruba, a Hewlett Packard Enterprise company
CableLabs
Cisco
Foundries.io
Kudelski IoT
NquiringMinds
NXP Semiconductors
Open Connectivity Foundation (OCF)
Sandelman Software Works
SEALSQ, a subsidiary of WISeKey
Silicon Labs

DOCUMENT CONVENTIONS

The terms “shall” and “shall not” indicate requirements to be followed strictly to conform to the publication and from which no deviation is permitted. The terms “should” and “should not” indicate that among several possibilities, one is recommended as particularly suitable without mentioning or excluding others, or that a certain course of action is preferred but not necessarily required, or that (in the negative form) a certain possibility or course of action is discouraged but not prohibited. The terms "may" and "need not" indicate a course of action permissible within the limits of the publication. The terms "can" and "cannot" indicate a possibility and capability, whether material, physical, or causal.

CALL FOR PATENT CLAIMS

This public review includes a call for information on essential patent claims (claims whose use would be required for compliance with the guidance or requirements in this Information Technology Laboratory (ITL) draft publication). Such guidance and/or requirements may be directly stated in this ITL Publication or by reference to another publication. This call also includes disclosure, where known, of the existence of pending U.S. or foreign patent applications relating to this ITL draft publication and of any relevant unexpired U.S. or foreign patents.

ITL may require from the patent holder, or a party authorized to make assurances on its behalf, in written or electronic form, either:

a) assurance in the form of a general disclaimer to the effect that such party does not hold and does not currently intend holding any essential patent claim(s); or

b) assurance that a license to such essential patent claim(s) will be made available to applicants desiring to utilize the license for the purpose of complying with the guidance or requirements in this ITL draft publication either:

1. under reasonable terms and conditions that are demonstrably free of any unfair discrimination; or
2. without compensation and under reasonable terms and conditions that are demonstrably free of any unfair discrimination.

Such assurance shall indicate that the patent holder (or third party authorized to make assurances on its behalf) will include in any documents transferring ownership of patents subject to the assurance, provisions sufficient to ensure that the commitments in the assurance are binding on the transferee, and that the transferee will similarly include appropriate provisions in the event of future transfers with the goal of binding each successor-in-interest.

The assurance shall also indicate that it is intended to be binding on successors-in-interest regardless of whether such provisions are included in the relevant transfer documents.

Such statements should be addressed to: iot-onboarding@nist.gov.

Contents

1 Introduction 1

1.1 How to Use This Guide 1

1.2 Build Overview..3

1.2.1 Reference Architecture Summary 3

1.2.2 Physical Architecture Summary..3

1.3 Typographic Conventions 7

2 Build 1 (Wi-Fi Easy Connect, Aruba/HPE)..7

2.1 Aruba Central/Hewlett Packard Enterprise (HPE) Cloud..7

2.2 Aruba Wireless Access Point 7

2.2.1 Wi-Fi Network Setup and Configuration 8

2.2.2 Wi-Fi Easy Connect Configuration 9

2.3 Cisco Catalyst 3850-S Switch 9

2.3.1 Configuration 10

2.4 Aruba User Experience Insight (UXI) Sensor 10

2.4.1 Configuration 10

2.5 Raspberry Pi..10

2.5.1 Configuration 11

2.5.2 DPP Onboarding 11

2.6 Certificate Authority..13

2.6.1 Private Certificate Authority..13

2.6.2 SEALSQ INeS..17

2.7 UXI Cloud 18

2.8 Wi-Fi Easy Connect Factory Provisioning Build 18

2.8.1 SEALSQ VaultIC Secure Element 18

3 Build 2 (Wi-Fi Easy Connect, CableLabs, OCF) 19

3.1 CableLabs Platform Controller 19

3.1.1 Operation and Demonstration 20

3.2 CableLabs Custom Connectivity Gateway 20

3.2.1 Installation and Configuration 20

3.2.2 Integration with CableLabs Platform Controller 20

3.2.3 Operation and Demonstration 20

3.3 Reference Clients/IoT Devices 20

3.3.1 Installation and Configuration 20

3.3.2 Operation and Demonstration 20

4 Build 3 (BRSKI, Sandelman Software Works) 21

4.1 Onboarding Router/Join Proxy 21

4.1.1 Setup and Configuration 21

4.2 Minerva Join Registrar Coordinator 21

4.2.1 Setup and Configuration 21

4.3 Reach Pledge Simulator 22

4.3.1 Setup and Configuration 22

4.4 Serial Console Server 23

4.5 Minerva Highway MASA Server 23

4.5.1 Setup and Configuration 23

5 Build 4 (Thread, Silicon Labs, Kudelski IoT) 24

5.1 Open Thread Border Router 24

5.1.1 Installation and Configuration 24

5.1.2 Operation and Demonstration 24

5.2 Silicon Labs Dev Kit (BRD2601A) 25

5.2.1 Setup and Configuration 25

5.3 Kudelski keySTREAM Service 28

5.3.1 Setup and Configuration 28

5.4 AWS IoT Core 30

5.4.1 Setup and Configuration 30

5.4.2 Testing 35

6 Build 5 (BRSKI over Wi-Fi, NquiringMinds) 36

6.1 Pledge 36

6.1.1 Installation and Configuration 37

6.1.2 Operation and Demonstration 37

6.2 Router and Logical Services 37

6.2.1 Installation and Configuration 37

6.2.2 Logical services 38

6.3 Onboarding Demonstration 41

6.3.1 Prerequisites 41

6.3.2 Onboarding Demonstration 42
6.3.3 Continuous Assurance Demonstration 42
6.4 BRSKI Factory Provisioning Build 42
6.4.1 Pledge 43
6.4.2 Installation and Configuration 43
6.4.3 Operation and Demonstration 43

List of Figures

Figure 1-1 NCCoE IoT Onboarding Laboratory Physical Architecture 5
Figure 6-1 Logical Services for Build 5 38
Figure 6-2 Diagram of Physical/Logical Components Used to Demonstrate BRSKI Flow 42

1 Introduction

The following volumes of this guide show information technology (IT) professionals and security engineers how we implemented these example solutions. We cover all of the products employed in this reference design. We do not re-create the product manufacturers' documentation, which is presumed to be widely available. Rather, these volumes show how we incorporated the products together in our environment.

Note: These are not comprehensive tutorials. There are many possible service and security configurations for these products that are out of scope for this reference design.

1.1 How to Use This Guide

This NIST Cybersecurity Practice Guide demonstrates a standards-based reference design for implementing trusted IoT device network-layer onboarding and lifecycle management and describes various example implementations of this reference design. Each of these implementations, which are known as *builds,* is standards-based and is designed to help provide assurance that networks are not put at risk as new IoT devices are added to them and to help safeguard IoT devices from connecting to unauthorized networks. The reference design described in this practice guide is modular and can be deployed in whole or in part, enabling organizations to incorporate trusted IoT device network-layer onboarding and lifecycle management into their legacy environments according to goals that they have prioritized based on risk, cost, and resources.

NIST is adopting an agile process to publish this content. Each volume is being made available as soon as possible rather than delaying release until all volumes are completed.

This guide contains five volumes:

- NIST Special Publication (SP) 1800-36A: *Executive Summary* – why we wrote this guide, the challenge we address, why it could be important to your organization, and our approach to solving this challenge
- NIST SP 1800-36B: *Approach, Architecture, and Security Characteristics* – what we built and why
- NIST SP 1800-36C: *How-To Guides* – instructions for building the example implementations, including all the security-relevant details that would allow you to replicate all or parts of this project **(you are here)**
- NIST SP 1800-36D: *Functional Demonstrations* – use cases that have been defined to showcase trusted IoT device network-layer onboarding and lifecycle management security capabilities and the results of demonstrating these use cases with each of the example implementations
- NIST SP 1800-36E: *Risk and Compliance Management* – risk analysis and mapping of trusted IoT device network-layer onboarding and lifecycle management security characteristics to cybersecurity standards and recommended practices

Depending on your role in your organization, you might use this guide in different ways:

Business decision makers, including chief security and technology officers, will be interested in the *Executive Summary, NIST SP 1800-36A*, which describes the following topics:

- challenges that enterprises face in migrating to the use of trusted IoT device network-layer onboarding
- example solutions built at the NCCoE
- benefits of adopting the example solution

Technology or security program managers who are concerned with how to identify, understand, assess, and mitigate risk will be interested in *NIST SP 1800-36B*, which describes what we did and why.

Also, Section 4 of *NIST SP 1800-36E* will be of particular interest. Section 4, *Mappings*, maps logical components of the general trusted IoT device network-layer onboarding and lifecycle management reference design to security characteristics listed in various cybersecurity standards and recommended practices documents, including *Framework for Improving Critical Infrastructure Cybersecurity* (NIST Cybersecurity Framework) and *Security and Privacy Controls for Information Systems and Organizations* (NIST SP 800-53).

You might share the *Executive Summary, NIST SP 1800-36A*, with your leadership team members to help them understand the importance of using standards-based trusted IoT device network-layer onboarding and lifecycle management implementations.

IT professionals who want to implement similar solutions will find the whole practice guide useful. You can use the how-to portion of the guide, *NIST SP 1800-36C*, to replicate all or parts of the builds created in our lab. The how-to portion of the guide provides specific product installation, configuration, and integration instructions for implementing the example solution. We do not re-create the product manufacturers’ documentation, which is generally widely available. Rather, we show how we incorporated the products together in our environment to create an example solution. Also, you can use *Functional Demonstrations, NIST SP 1800-36D*, which provides the use cases that have been defined to showcase trusted IoT device network-layer onboarding and lifecycle management security capabilities and the results of demonstrating these use cases with each of the example implementations. Finally, *NIST SP 1800-36E* will be helpful in explaining the security functionality that the components of each build provide.

This guide assumes that IT professionals have experience implementing security products within the enterprise. While we have used a suite of commercial products to address this challenge, this guide does not endorse these particular products. Your organization can adopt this solution or one that adheres to these guidelines in whole, or you can use this guide as a starting point for tailoring and implementing parts of a trusted IoT device network-layer onboarding and lifecycle management solution. Your organization’s security experts should identify the products that will best integrate with your existing tools and IT system infrastructure. We hope that you will seek products that are congruent with applicable standards and recommended practices.

A NIST Cybersecurity Practice Guide does not describe “the” solution, but example solutions. We seek feedback on the publication’s contents and welcome your input. Comments, suggestions, and success stories will improve subsequent versions of this guide. Please contribute your thoughts to iot-onboarding@nist.gov.

1.2 Build Overview

This NIST Cybersecurity Practice Guide addresses the challenge of network-layer onboarding using standards-based protocols to perform trusted network-layer onboarding of an IoT device. Each build demonstrates one or more of these capabilities:

- Trusted Network-Layer Onboarding: providing the device with its unique network credentials over an encrypted channel
- Network Re-Onboarding: performing trusted network-layer onboarding of the device again, after device reset
- Network Segmentation: assigning a device to a particular local network segment to prevent it from communicating with other network components, as determined by enterprise policy
- Trusted Application-Layer Onboarding: providing the device with application-layer credentials over an encrypted channel after completing network-layer onboarding
- Ongoing Device Authorization: continuously monitoring the device on an ongoing basis, providing policy-based assurance and authorization checks on the device throughout its lifecycle
- Device Communications Intent Enforcement: Secure conveyance of device communications intent information, combined with enforcement of it, to ensure that IoT devices are constrained to sending and receiving only those communications that are explicitly required for each device to fulfill its purpose

Five builds that will serve as examples of how to onboard IoT devices using the protocols described in NIST SP 1800-36B, as well as the factory provisioning builds, are being implemented and will be demonstrated as part of this project. The remainder of this practice guide provides step-by-step instructions on how to reproduce all builds.

1.2.1 Reference Architecture Summary

The builds described in this document are instantiations of the trusted network-layer onboarding and lifecycle management logical reference architecture that is described in NIST SP 1800-36B. This architecture is organized according to five high-level processes: Device Manufacture and Factory Provisioning, Device Ownership and Bootstrapping Information Transfer, Trusted Network-Layer Onboarding, Trusted Application-Layer Onboarding, and Continuous Verification. For a full explanation of the architecture, please see NIST SP 1800-36B: *Approach, Architecture, and Security Characteristics*.

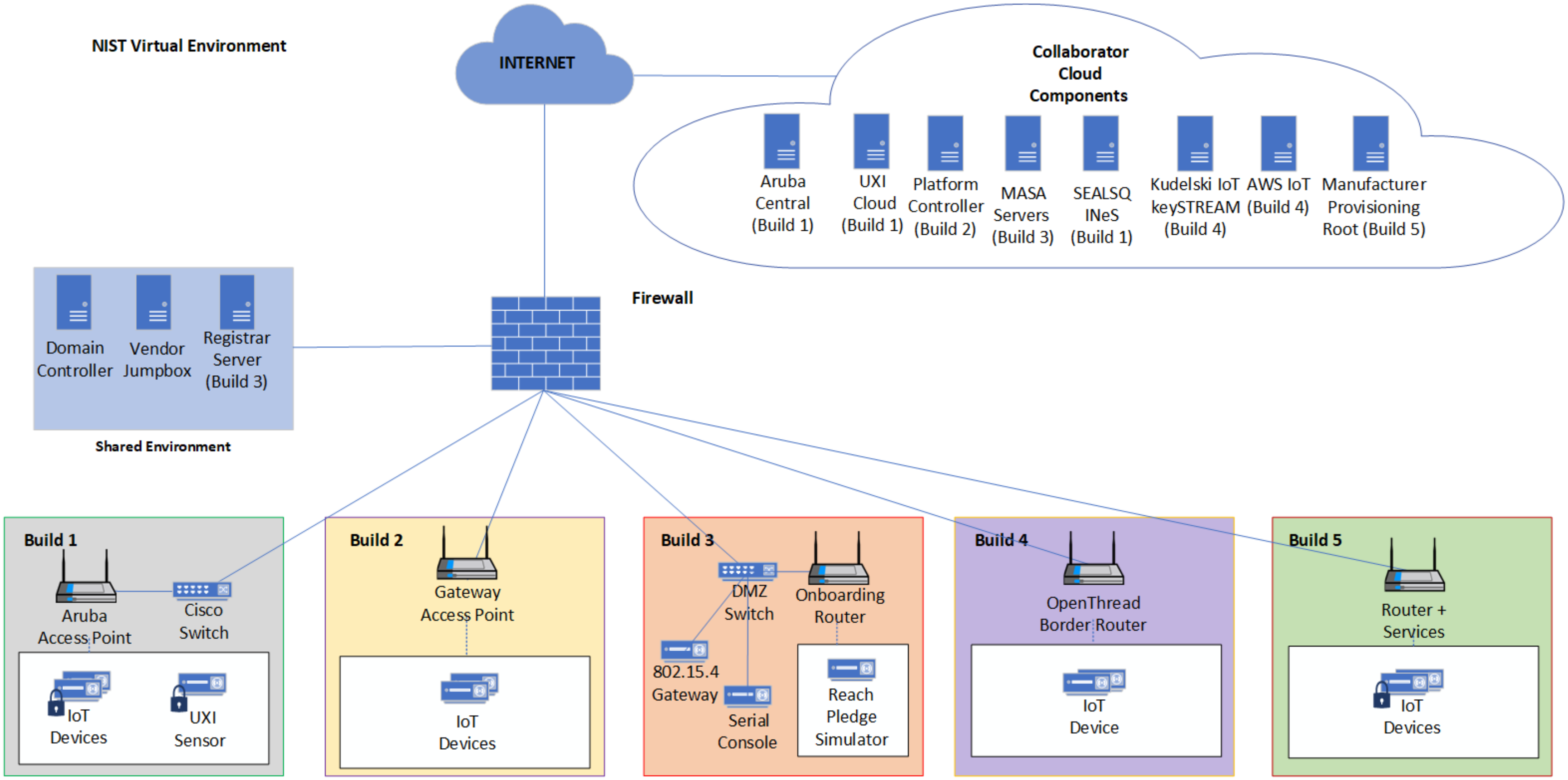
1.2.2 Physical Architecture Summary

Figure 1-1 depicts the high-level physical architecture of the NCCoE IoT Onboarding laboratory environment in which the five trusted IoT device network-layer onboarding project builds and the two factory provisioning builds are being implemented. The NCCoE provides virtual machine (VM) resources and physical infrastructure for the IoT Onboarding lab. As depicted, the NCCoE IoT Onboarding laboratory hosts collaborator hardware and software for the builds. The NCCoE also provides connectivity from the IoT Onboarding lab to the NIST Data Center, which provides connectivity to the internet and public IP spaces (both IPv4 and IPv6). Access to and from the NCCoE network is protected by a firewall.

Access to and from the IoT Onboarding lab is protected by a pfSense firewall, represented by the brick box icon in Figure 1-1. This firewall has both IPv4 and IPv6 (dual stack) configured. The IoT Onboarding lab network infrastructure includes a shared virtual environment that houses a domain controller and a vendor jumpbox. These components are used across builds where applicable. It also contains five independent virtual local area networks (VLANs), each of which houses a different trusted network-layer onboarding build.

The IoT Onboarding laboratory network has access to cloud components and services provided by the collaborators, all of which are available via the internet. These components and services include Aruba Central and the UXI Cloud (Build 1), SEALSQ INeS (Build 1), Platform Controller (Build 2), a MASA server (Build 3), Kudelski IoT keySTREAM application-layer onboarding service and AWS IoT (Build 4), and a Manufacturer Provisioning Root (Build 5).

300 **Figure 1-1 NCCoE IoT Onboarding Laboratory Physical Architecture**



All five network-layer onboarding laboratory environments, as depicted in the diagram, have been installed:

- Build 1 (i.e., the Wi-Fi Easy Connect, Aruba/HPE build) network infrastructure within the NCCoE lab consists of two components: the Aruba Access Point and the Cisco Switch. Build 1 also requires support from Aruba Central for network-layer onboarding and the UXI Cloud for application-layer onboarding. These components are in the cloud and accessed via the internet. The IoT devices that are onboarded using Build 1 include the UXI Sensor and the Raspberry Pi.
- Build 2 (i.e., the Wi-Fi Easy Connect, CableLabs, OCF build) network infrastructure within the NCCoE lab consists of a single component: the Gateway Access Point. Build 2 requires support from the Platform Controller, which also hosts the IoTivity Cloud Service. The IoT devices that are onboarded using Build 2 include three Raspberry Pis.
- Build 3 (i.e., the BRSKI, Sandelman Software Works build) network infrastructure components within the NCCoE lab include a Wi-Fi capable home router (including Join Proxy), a DMZ switch (for management), and an ESP32A Xtensa board acting as a Wi-Fi IoT device, as well as an nRF52840 board acting as an IEEE 802.15.4 device. A management system on a BeagleBone Green serves as a serial console. A registrar server has been deployed as a virtual appliance on the NCCoE private cloud system. Build 3 also requires support from a MASA server which is accessed via the internet. In addition, a Raspberry Pi 3 provides an ethernet/802.15.4 gateway, as well as a test platform.
- Build 4 (i.e., the Thread, Silicon Labs, Kudelski IoT build) network infrastructure components within the NCCoE lab include an Open Thread Border Router, which is implemented using a Raspberry Pi, and a Silicon Labs Gecko Wireless Starter Kit, which acts as an 802.15.4 antenna. Build 4 also requires support from the Kudelski IoT keySTREAM service, which is in the cloud and accessed via the internet. The IoT device that is onboarded in Build 4 is the Silicon Labs Dev Kit (BRD2601A) with an EFR32MG24 System-on-Chip. The application service to which it onboards is AWS IoT.
- Build 5 (i.e., the BRSKI over Wi-Fi, NquiringMinds build) includes 2 Raspberry Pi 4Bs running a Linux operating system. One Raspberry Pi acts as the pledge (or IoT Device) with an Infineon TPM connected. The other acts as the router, registrar and MASA all in one device. This build uses the open source TrustNetZ distribution, from which the entire build can be replicated easily. The TrustNetZ distribution includes source code for the IoT device, the router, the access point, the network onboarding component, the policy engine, the manufacturer services, the registrar and a demo application server. TrustNetZ makes use of NquiringMinds tdx Volt to issue and validate verifiable credentials.
- The BRSKI factory provisioning build is deployed in the Build 5 environment. The IoT device in this build is a Raspberry Pi equipped with an Infineon Optiga SLB 9670 TPM 2.0, which gets provisioned with birth credentials (i.e., a public/private key pair and an IDevID). The BRSKI factory provisioning build also uses an external certificate authority hosted on the premises of NquiringMinds to provide the device certificate signing service.
- The Wi-Fi Easy Connect factory provisioning build is deployed in the Build 1 environment. Its IoT devices are Raspberry Pis equipped with a SEALSQ VaultIC Secure Element, which gets provisioned with a DPP URI. The Secure Element can also be provisioned with an IDevID certificate signed by the SEALSQ INeS certification authority, which is independent of the DPP URI. Code for performing the factory provisioning is stored on an SD card.

1.3 Typographic Conventions

The following table presents typographic conventions used in this volume.

Typeface/Symbol	Meaning	Example
Italics	file names and path names; references to documents that are not hyperlinks; new terms; and placeholders	For language use and style guidance, see the *NCCoE Style Guide*.
Bold	names of menus, options, command buttons, and fields	Choose **File** > **Edit.**
`Monospace`	command-line input, onscreen computer output, sample code examples, and status codes	`mkdir`
`Monospace Bold`	command-line user input contrasted with computer output	**`service sshd start`**
blue text	link to other parts of the document, a web URL, or an email address	All publications from NIST's NCCoE are available at https://www.nccoe.nist.gov.

2 Build 1 (Wi-Fi Easy Connect, Aruba/HPE)

This section of the practice guide contains detailed instructions for installing and configuring all the products used to build an instance of the example solution. For additional details on Build 1's logical and physical architectures, see NIST SP 1800-36B: *Approach, Architecture, and Security Characteristics*.

The network-layer onboarding component of Build 1 utilizes Wi-Fi Easy Connect, also known as the Device Provisioning Protocol (DPP). The Wi-Fi Easy Connect standard is maintained by the Wi-Fi Alliance [1]. The term "DPP" is used when referring to the network-layer onboarding protocol, and "Wi-Fi Easy Connect" is used when referring to the overall implementation of the network onboarding process.

2.1 Aruba Central/Hewlett Packard Enterprise (HPE) Cloud

This build utilized Aruba Central as a cloud management service that provided management and support for the Aruba Wireless Access Point (AP) and provided authorization and DPP onboarding capabilities for the wireless network. A cloud-based application programming interface (API) endpoint provided the ability to import the DPP Uniform Resource Identifiers (URIs) in the manner of a Supply Chain Integration Service. Due to this capability and Build 1's support for Wi-Fi Easy Connect, Build 1's infrastructure fully supported interoperable network-layer onboarding with Build 2's Reference Clients ("IoT devices") provided by CableLabs.

2.2 Aruba Wireless Access Point

Use of DPP is implicitly dependent on the Aruba Central cloud service. Aruba Central provides a cloud Infrastructure as a Service (IaaS) enabled architecture that includes initial support for DPP in Central 2.5.6/ArubaOS (AOS) 10.4.0. Central and AOS support multiple deployment formats:

1. As AP only, referred to as an *underlay deployment,* where traffic is bridged locally from the APs.

2. An *overlay deployment,* where all data is securely tunneled to an on-prem gateway where advanced services can route, inspect, and analyze the data before it's either bridged locally or routed to its next hop.

3. A *mixed-mode deployment,* which is a combination of the two where a returned 'role/label' is used to determine how the data is processed and forwarded.

At the time of this publication, a user can leverage any 3xx, 5xx, or 6xx APs to support a DPP deployment, with a view that all future series APs will implicitly include support. For an existing or new user there is a prerequisite of the creation of a Service Set Identifier (SSID). Note that DPP today is not supported under Wi-Fi Protected Access 3 (WPA3); this is a roadmap item with no published timeline.

Assuming there is an existing SSID or a new one is created based upon the above security restrictions, the next step is to enable DPP (as detailed below in Section 2.2.1) such that the SSID can support multiple authentication and key managements (AKMs) on a Basic Service Set (BSS). If the chosen security type is DPP, only a single AKM will exist for that BSS.

A standards-compliant 802.3at port is the easiest method for providing the AP with power. An external power supply can also be used.

Within this document, we do not cover the specifics of radio frequency (RF) design and placement of APs. Guidance and assistance is available within the Aruba community site, https://community.arubanetworks.com or the Aruba Support Portal, https://asp.arubanetworks.com. Additionally, we do not cover onboarding and licensing of Aruba Central hardware. Documentation can be found here: https://www.arubanetworks.com/techdocs/ArubaDocPortal/content/docportal.htm.

2.2.1 Wi-Fi Network Setup and Configuration

The following instructions detail the initial setup and configuration of the Wi-Fi network upon powering on and connecting the AP to an existing network.

1. Navigate to the Aruba Central cloud management interface.

2. On the sidebar, navigate under **Global** and choose the AP-Group you want to configure/modify. (This assumes you have already grouped your APs by location/functions.)

3. Under **Devices,** click **Config** in the top right side.

4. You will now be in the Access Points tab and WLANs tab. Do one of the following:

    a. If creating a new SSID, click on **+ Add SSID.** After entering the Name (SSID) in Step 1 and configuring options as necessary in Step 2, when you get to Step 3 (Security), it will default on the slide-bar to the Personal Security Level; the alternative is the Enterprise Security Level.

        i. If you choose the **Personal Security Level,** under **Key-Management** ensure you select either **DPP** or **WPA2-Personal.** If you choose **WPA2-Personal,** expand the **Advanced Settings** section and enable the toggle button for DPP so that the SSID can broadcast the AKM. Note that this option is not available if choosing DPP for Key-Management.

   ii. If you choose the **Enterprise Security Level,** only WPA2-Enterprise Key-Management currently supports DPP. Expand the **Advanced Settings** section and enable the toggle button for **DPP** so that the SSID can broadcast the AKM.

   b. If you plan to enable DPP on a previously created SSID:

      i. Ensure you are running version 10.4+ on your devices. You also need an SSID that is configured for WPA2-Personal or WPA2-Enterprise.

      ii. When ready, float your cursor over the previously created SSID name you wish to configure and click on the edit icon.

      iii. Edit the SSID, click on **Security**, and expand the **Advanced Settings** section and enable the toggle button for **DPP.**

      iv. Click **Save Settings.**

For SSIDs that have been modified to add DPP AKM, it's also necessary to enable DPP within the radio profile.

1. Under the **Access Point** Tab, click **Radios.**
2. It's expected you'll see a **default** radio-profile. If a custom one has been created, you'll need to review your configuration before proceeding.
3. Assuming a **default** radio-profile, click on the **Edit** icon, expand **Show advanced settings,** and scroll down to **DPP Provisioning.** You can selectively enable this for 2.4 GHz or 5.0 GHz. Support for DPP on 6.0 GHz is a roadmap item at this time and is not yet available.

2.2.2 Wi-Fi Easy Connect Configuration

Configuration of the Access Point occurred through the Aruba Central cloud management interface. Standard configurations were used to stand up the Build 1 wireless network. The instructions for enabling DPP capabilities for the overall wireless network are listed below:

1. Navigate to the Aruba Central cloud management interface.
2. On the sidebar, navigate to **Security > Authentication and Policy > Config.**
3. In the **Client Access Policy** section, click **Edit.**
4. Under the **Wi-Fi Easy Connect™ Service** heading, ensure that the name of your wireless network is selected.
5. Click **Save.**

2.3 Cisco Catalyst 3850-S Switch

This build utilized a Cisco Catalyst 3850-S switch. This switch utilized a minimal configuration with two separate VLANs to allow for IoT device network segmentation and access control. The switch also provided Power-over-Ethernet support for the Aruba Wireless AP.

2.3.1 Configuration

The switch was configured with two VLANs, and a trunk port dedicated to the Aruba Wireless AP. You can find the relevant portions of the Cisco iOS configuration below:

```
interface Vlan1
 no ip address
interface Vlan2
 no ip address
interface GigabitEthernet1/0/1
 switchport mode trunk
interface GigabitEthernet1/0/2
 switchport mode access
 switchport access vlan 1
interface GigabitEthernet1/0/3
 switchport mode access
 switchport access vlan 2
```

2.4 Aruba User Experience Insight (UXI) Sensor

This build utilized an Aruba UXI Sensor as a Wi-Fi Easy Connect-capable IoT device. Models G6 and G6C support Wi-Fi Easy Connect, and all available G6 and G6C models support Wi-Fi Easy Connect within their software image. This sensor successfully utilized the network-layer onboarding mechanism provided by the wireless network and completed onboarding to the application-layer UXI cloud service. The network-layer onboarding process is automatically initiated by the device on boot.

2.4.1 Configuration

All of Aruba’s available G6 and G6C UXI sensors support the ability to complete network-layer and application-layer onboarding. No specific configuration of the physical sensor is required. As part of the supply-chain process, the cryptographic public key for your sensor(s) will be available within the cloud tenant. This public/private keypair for each device is created as part of the manufacturing process. The public key effectively identifiers the sensor to the network and as part of the Wi-Fi Easy Connect/DPP onboarding process. This allows unprovisioned devices straight from the factory to be onboarded and subsequently connect to the UXI sensor cloud to obtain their network-layer configuration. An administrator will have to define the ‘tasks’ the UXI sensor is going to perform such as monitoring SSIDs, performing reachability tests to on-prem or cloud services, and making the results of these tests available within the UXI user/administrator portal.

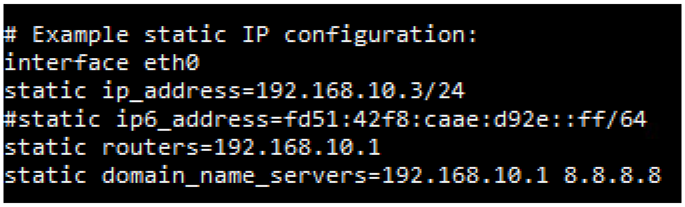
2.5 Raspberry Pi

In this build, the Raspberry Pi 3B+ acts as a DPP enrollee. In setting up the device for this build, a DPP-capable wireless adapter, the Alfa AWUS036NHA network dongle, was connected to enable the Pi to send and receive DPP frames. Once fully configured, the Pi can onboard with the Aruba AP.

2.5.1 Configuration

The following steps were completed for the Raspberry Pi to complete DPP onboarding:

1. Set the management IP for the Raspberry Pi to an IP address in the Build 1 network. To do this, add the following lines to the file *dhcpcd.conf* located at */etc/dhcpcd.conf*. For this build, the IP address was set to 192.168.10.3.



```
# Example static IP configuration:
interface eth0
static ip_address=192.168.10.3/24
#static ip6_address=fd51:42f8:caae:d92e::ff/64
static routers=192.168.10.1
static domain_name_servers=192.168.10.1 8.8.8.8
```

2. Install Linux Libraries using the apt package manager. The following packages were installed:
    a. autotools-dev
    b. automake
    c. libcurl4-openssl-dev
    d. libnl-genl-3-dev
    e. libavahi-client-dev
    f. libavahi-core-dev
    g. aircrack-ng
    h. openssl-1.1.1q
3. Install the DPP utilities. These utilities were installed from the GitHub repository https://github.com/HewlettPackard/dpp using the following command:

```
git clone https://github.com/HewlettPackard/dpp
```

2.5.2 DPP Onboarding

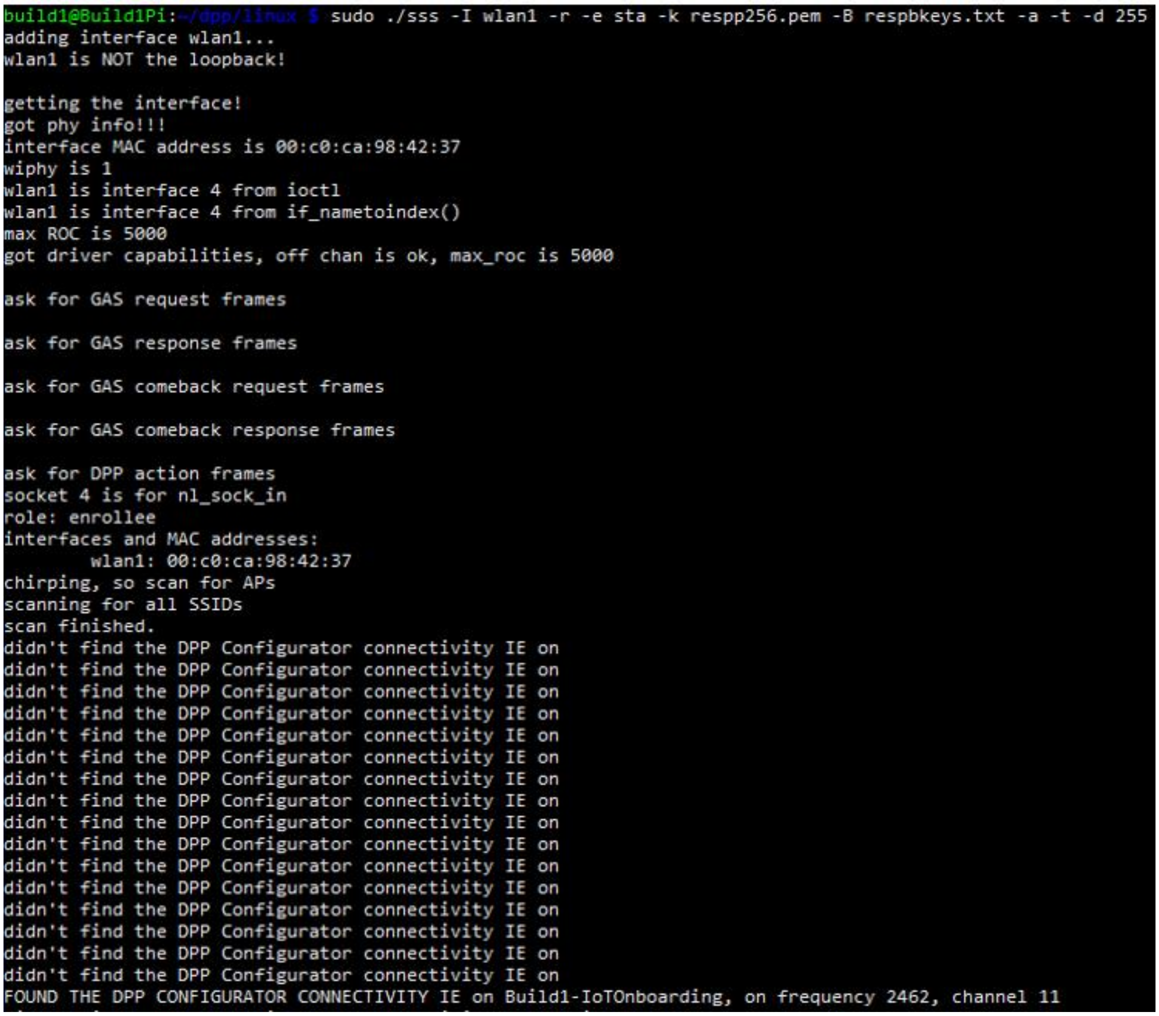
This section describes the steps for using the Raspberry Pi as a DPP enrollee. The Pi uses a DPP utility to send out chirps to make its presence known to available DPP configurators. Once the Pi is discovered, the DPP configurator (Aruba Wireless AP) initiates the DPP authentication protocol. During this phase, DPP *connectors* are created to onboard the device to the network. As soon as the Pi is fully authenticated, it is fully enrolled and can begin normal network communication.

1. Navigate to the DPP utilities directory which was installed during setup:

```
cd dpp/linux
```

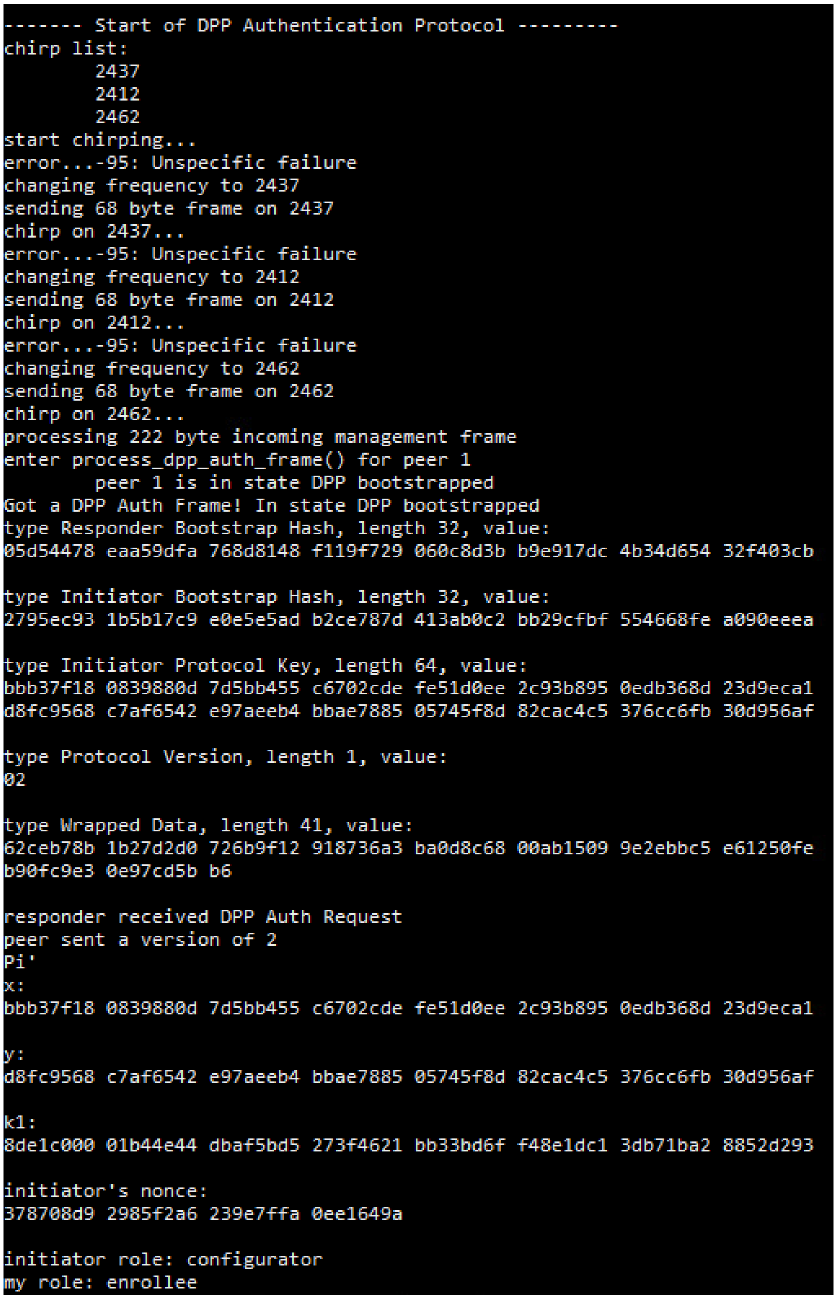
2. From the DPP utilities directory, run the following command to initiate a DPP connection:

```
sudo ./sss -I wlan1 -r -e sta -k respp256.pem -B respbkeys.txt -a -t -d 255
```



```
build1@Build1Pi:~/dpp/linux $ sudo ./sss -I wlan1 -r -e sta -k respp256.pem -B respbkeys.txt -a -t -d 255
adding interface wlan1...
wlan1 is NOT the loopback!

getting the interface!
got phy info!!!
interface MAC address is 00:c0:ca:98:42:37
wiphy is 1
wlan1 is interface 4 from ioctl
wlan1 is interface 4 from if_nametoindex()
max ROC is 5000
got driver capabilities, off chan is ok, max_roc is 5000

ask for GAS request frames

ask for GAS response frames

ask for GAS comeback request frames

ask for GAS comeback response frames

ask for DPP action frames
socket 4 is for nl_sock_in
role: enrollee
interfaces and MAC addresses:
        wlan1: 00:c0:ca:98:42:37
chirping, so scan for APs
scanning for all SSIDs
scan finished.
didn't find the DPP Configurator connectivity IE on
didn't find the DPP Configurator connectivity IE on
didn't find the DPP Configurator connectivity IE on
didn't find the DPP Configurator connectivity IE on
didn't find the DPP Configurator connectivity IE on
didn't find the DPP Configurator connectivity IE on
didn't find the DPP Configurator connectivity IE on
didn't find the DPP Configurator connectivity IE on
didn't find the DPP Configurator connectivity IE on
didn't find the DPP Configurator connectivity IE on
didn't find the DPP Configurator connectivity IE on
didn't find the DPP Configurator connectivity IE on
didn't find the DPP Configurator connectivity IE on
didn't find the DPP Configurator connectivity IE on
didn't find the DPP Configurator connectivity IE on
didn't find the DPP Configurator connectivity IE on
FOUND THE DPP CONFIGURATOR CONNECTIVITY IE on Build1-IoTOnboarding, on frequency 2462, channel 11
```

3. Once the enrollee has found a DPP configurator, the DPP authentication protocol is initiated.



```
------- Start of DPP Authentication Protocol ---------
chirp list:
        2437
        2412
        2462
start chirping...
error...-95: Unspecific failure
changing frequency to 2437
sending 68 byte frame on 2437
chirp on 2437...
error...-95: Unspecific failure
changing frequency to 2412
sending 68 byte frame on 2412
chirp on 2412...
error...-95: Unspecific failure
changing frequency to 2462
sending 68 byte frame on 2462
chirp on 2462...
processing 222 byte incoming management frame
enter process_dpp_auth_frame() for peer 1
        peer 1 is in state DPP bootstrapped
Got a DPP Auth Frame! In state DPP bootstrapped
type Responder Bootstrap Hash, length 32, value:
05d54478 eaa59dfa 768d8148 f119f729 060c8d3b b9e917dc 4b34d654 32f403cb

type Initiator Bootstrap Hash, length 32, value:
2795ec93 1b5b17c9 e0e5e5ad b2ce787d 413ab0c2 bb29cfbf 554668fe a090eeea

type Initiator Protocol Key, length 64, value:
bbb37f18 0839880d 7d5bb455 c6702cde fe51d0ee 2c93b895 0edb368d 23d9eca1
d8fc9568 c7af6542 e97aeeb4 bbae7885 05745f8d 82cac4c5 376cc6fb 30d956af

type Protocol Version, length 1, value:
02

type Wrapped Data, length 41, value:
62ceb78b 1b27d2d0 726b9f12 918736a3 ba0d8c68 00ab1509 9e2ebbc5 e61250fe
b90fc9e3 0e97cd5b b6

responder received DPP Auth Request
peer sent a version of 2
Pi'
x:
bbb37f18 0839880d 7d5bb455 c6702cde fe51d0ee 2c93b895 0edb368d 23d9eca1

y:
d8fc9568 c7af6542 e97aeeb4 bbae7885 05745f8d 82cac4c5 376cc6fb 30d956af

k1:
8de1c000 01b44e44 dbaf5bd5 273f4621 bb33bd6f f48e1dc1 3db71ba2 8852d293

initiator's nonce:
378708d9 2985f2a6 239e7ffa 0ee1649a

initiator role: configurator
my role: enrollee
```

2.6 Certificate Authority

The function of the certificate authority (CA) in this build is to issue network credentials for use in the network-layer onboarding process.

2.6.1 Private Certificate Authority

A private CA was provided as a part of the DPP demonstration utilities in the HPE GitHub repository. For demonstration purposes, the Raspberry Pi is used as the configurator and the enrollee.

2.6.1.1 Installation and Configuration

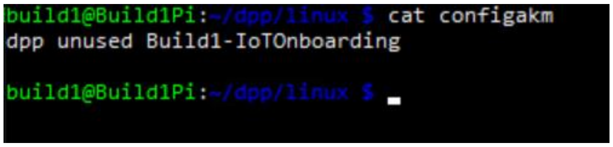
The following instructions detail the initial setup and configuration of the private CA using the DPP demonstration utilities and certificates located at https://github.com/HewlettPackard/dpp.

1. Navigate to the DPP utilities directory on the Raspberry Pi: *~dpp/linux*

   ```
   cd dpp/linux/
   ```

2. The README in the GitHub repository (https://github.com/HewlettPackard/dpp/blob/master/README) references a text file called *configakm* which contains information about the network policies for a configurator to provision on an enrollee. The format is: `<akm> <EAP server> <ssid>`. Current AKMs that are supported are DPP, dot1x, sae, and psk. For this build, DPP is used. For DPP, an Extensible Authentication Protocol (EAP) server is not used.

3. Configure the file *configakm* located in *~/dpp/linux/*. This file instructs the configurator on how to deploy a DPP connector (network credential) from the configurator to the enrollee. As shown below, the *configakm* file is filled with the following fields:
   `dpp unused Build1-IoTOnboarding.`

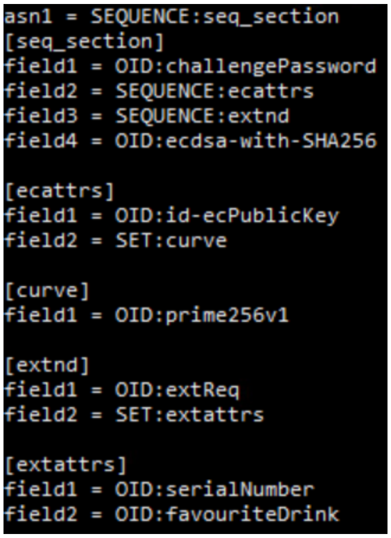
   

   ```
   build1@Build1Pi:~/dpp/linux $ cat configakm
   dpp unused Build1-IoTOnboarding

   build1@Build1Pi:~/dpp/linux $ 
   ```

4. The file *csrattrs.conf* contains attributes to construct an Abstract Syntax Notation One (ASN.1) string. This string allows the configurator to tell the enrollee how to generate a certificate signing request (CSR). The following fields were used for this demonstration:

   ```
   asn1 = SEQUENCE: seq_section
   [seq_section]
   field1 = OID:challengePassword
   field2 = SEQUENCE:ecattrs
   field3 = SEQUENCE:extnd
   field4 = OID:ecdsa-with-SHA256

   [ecattrs]
   field1 = OID:id-ecPublicKey
   field2 = SET:curve

   [curve]
   field1 = OID:prime256v1
   ```

```
[extnd]
field1 = OID:extReq
field2 = SET:extattrs

[extattrs]
field1 = OID:serialNumber
field2 = OID:favouriteDrink
```



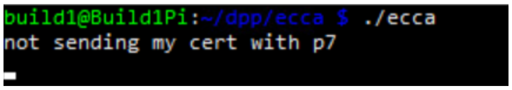
```
asn1 = SEQUENCE:seq_section
[seq_section]
field1 = OID:challengePassword
field2 = SEQUENCE:ecattrs
field3 = SEQUENCE:extnd
field4 = OID:ecdsa-with-SHA256

[ecattrs]
field1 = OID:id-ecPublicKey
field2 = SET:curve

[curve]
field1 = OID:prime256v1

[extnd]
field1 = OID:extReq
field2 = SET:extattrs

[extattrs]
field1 = OID:serialNumber
field2 = OID:favouriteDrink
```

2.6.1.2 Operation and Demonstration

Once setup and configuration have been completed, the following steps can be used to demonstrate utilizing the private CA to issue credentials to a requesting device.

1. Open three terminals on the Raspberry Pi: one to start the certificate program, one to show the configurator's point of view, and one to show the enrollee's point of view.
2. The demonstration uses an OpenSSL certificate. To run the program from the first terminal, navigate to the following directory: *~/dpp/ecca/,* and run the command:

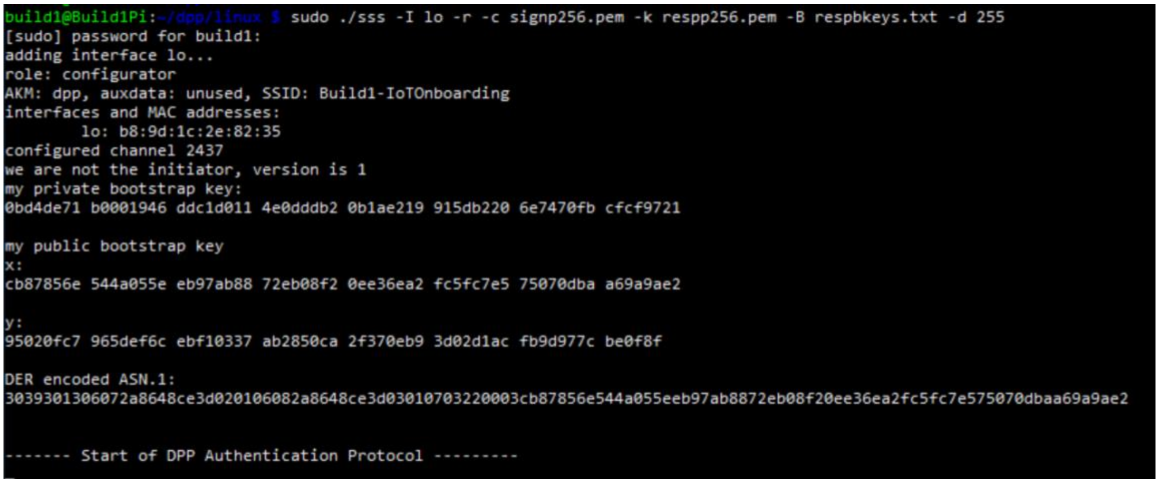
```
./ecca.
```



```
build1@Build1Pi:~/dpp/ecca $ ./ecca
not sending my cert with p7
```

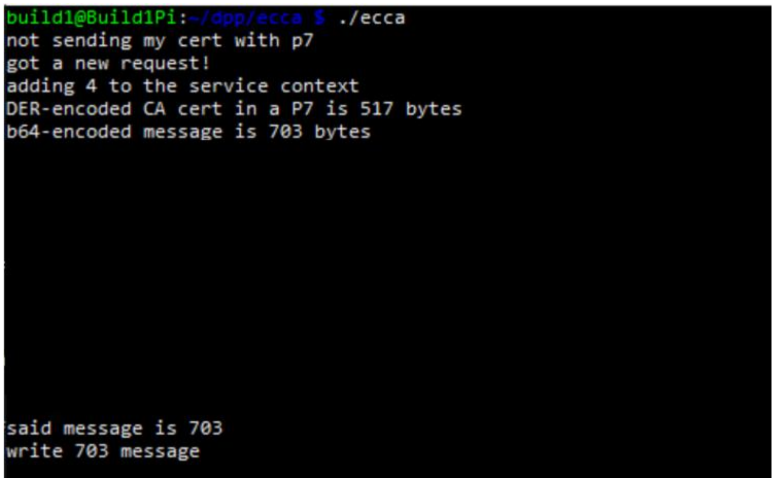
3. On the second terminal, start the configurator using the following command:

```
sudo ./sss -I lo -r -c signp256.pem -k respp256.pem -B resppbkeys.txt -d 255
```



```
build1@Build1Pi:~/dpp/linux $ sudo ./sss -I lo -r -c signp256.pem -k respp256.pem -B respbkeys.txt -d 255
[sudo] password for build1:
adding interface lo...
role: configurator
AKM: dpp, auxdata: unused, SSID: Build1-IoTOnboarding
interfaces and MAC addresses:
        lo: b8:9d:1c:2e:82:35
configured channel 2437
we are not the initiator, version is 1
my private bootstrap key:
0bd4de71 b0001946 ddc1d011 4e0dddb2 0b1ae219 915db220 6e7470fb cfcf9721

my public bootstrap key
x:
cb87856e 544a055e eb97ab88 72eb08f2 0ee36ea2 fc5fc7e5 75070dba a69a9ae2

y:
95020fc7 965def6c ebf10337 ab2850ca 2f370eb9 3d02d1ac fb9d977c be0f8f

DER encoded ASN.1:
3039301306072a8648ce3d020106082a8648ce3d03010703220003cb87856e544a055eeb97ab8872eb08f20ee36ea2fc5fc7e575070dbaa69a9ae2


------- Start of DPP Authentication Protocol ---------
```

As shown in the terminal where the ecca program is running, the configurator contacts the CA and asks for the certificate.

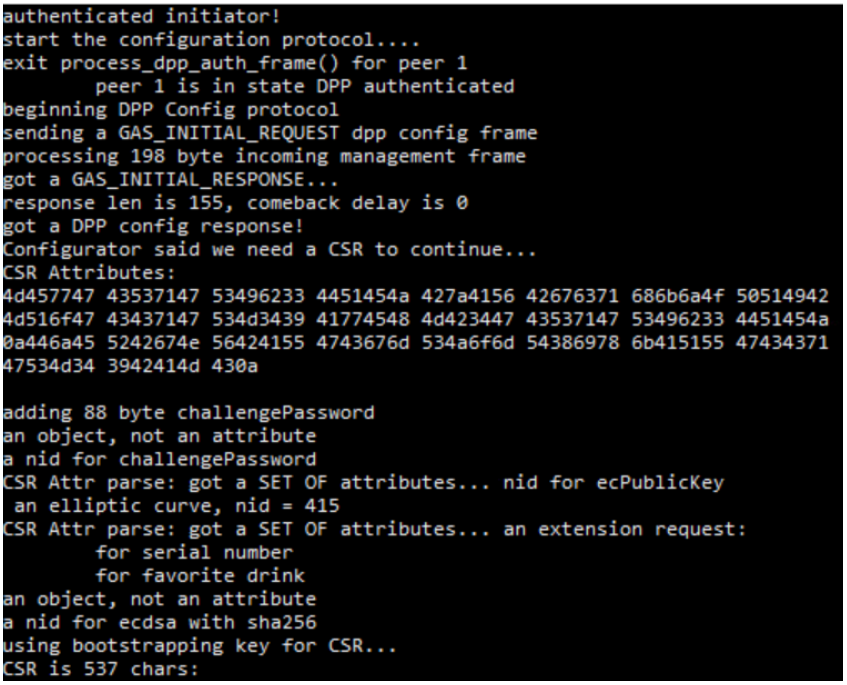


```
build1@Build1Pi:~/dpp/ecca $ ./ecca
not sending my cert with p7
got a new request!
adding 4 to the service context
DER-encoded CA cert in a P7 is 517 bytes
b64-encoded message is 703 bytes











said message is 703
write 703 message
```

4. On the third terminal, start the enrollee using the following command:

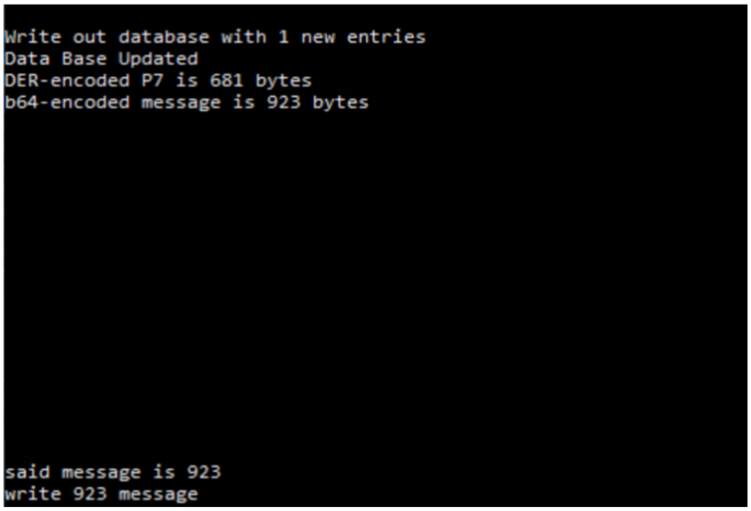
   `sudo ./sss -I lo -r -e sta -k initp256.pem -B initbkeys.txt -t -a -q -d 255`

   From the enrollee’s perspective, it will send chirps on different channels until it finds the configurator. Once found, it sends its certificate to the CA for signing. The snippet below is of the enrollee generating the CSR.



```
authenticated initiator!
start the configuration protocol....
exit process_dpp_auth_frame() for peer 1
        peer 1 is in state DPP authenticated
beginning DPP Config protocol
sending a GAS_INITIAL_REQUEST dpp config frame
processing 198 byte incoming management frame
got a GAS_INITIAL_RESPONSE...
response len is 155, comeback delay is 0
got a DPP config response!
Configurator said we need a CSR to continue...
CSR Attributes:
4d457747 43537147 53496233 4451454a 427a4156 42676371 686b6a4f 50514942
4d516f47 43437147 534d3439 41774548 4d423447 43537147 53496233 4451454a
0a446a45 5242674e 56424155 4743676d 534a6f6d 54386978 6b415155 47434371
47534d34 3942414d 430a

adding 88 byte challengePassword
an object, not an attribute
a nid for challengePassword
CSR Attr parse: got a SET OF attributes... nid for ecPublicKey
 an elliptic curve, nid = 415
CSR Attr parse: got a SET OF attributes... an extension request:
        for serial number
        for favorite drink
an object, not an attribute
a nid for ecdsa with sha256
using bootstrapping key for CSR...
CSR is 537 chars:
```

5. In the ecca terminal, the certificate from the enrollee is shown



```
Write out database with 1 new entries
Data Base Updated
DER-encoded P7 is 681 bytes
b64-encoded message is 923 bytes
```

```
said message is 923
write 923 message
```

2.6.2 SEALSQ INeS

The SEALSQ INeS Certificate Management System provides CA and certificate management capabilities for Build 1. Implementation of this system provides Build 1 with a trusted, public CA to support issuing network credentials.

2.6.2.1 Setup and Configuration

To support this build, a custom software agent was deployed on a Raspberry Pi in the Build 1 network. This agent interacted with the cloud-based CA in SEALSQ INeS via API to sign network credentials. Network-level onboarding of IoT devices was completed via DPP, with network credentials being successfully requested from and issued by SEALSQ INeS.

Additional information on interacting with the SEALSQ INeS API can be found at https://inesdev.certifyiddemo.com/. Access can be requested directly from SEALSQ via their contact form: https://www.sealsq.com/contact.

2.7 UXI Cloud

The UXI Cloud is a web-based application that serves as a monitoring hub for the UXI sensor. It provides visibility into the data captured by the performance monitoring that the UXI sensor conducts. For the purposes of this build, the dashboard was used to demonstrate application-layer onboarding, which occurs once the UXI sensor has completed network-layer onboarding. Once application-layer onboarding was completed and the application configuration had been applied to the device, our demonstration concluded.

2.8 Wi-Fi Easy Connect Factory Provisioning Build

This Factory Provisioning Build included many of the components listed above, including Aruba Central, SEALSQ INeS, the Aruba Access Point, and Raspberry Pi IoT devices. A SEALSQ VaultIC Secure Element was also included in the build and provided secure generation and storage of the key material and certificates provisioned to the device.

2.8.1 SEALSQ VaultIC Secure Element

The SEALSQ VaultIC Secure Element was connected to a Raspberry Pi via the built-in GPIO pins present on the Pi. SEALSQ provided demonstration code that generates a public/private keypair within the secure element, creates a Certificate Signing Request, and uses that CSR to obtain an IDevID certificate from SEALSQ INeS. This code supports the Raspberry Pi OS Bullseye. The demonstration code can be found at the official GitHub repository.

HPE also provided a custom DPP-based implementation of the SEALSQ code, which generates supporting material within the secure element, and then generates a DPP URI. This DPP URI is available in a string format, PNG (QR Code), and ASCII (QR Code). The DPP URI can then be used for network onboarding, as described in the rest of the Build 1 section. This code is included in the demonstration code located at the repository linked above.

2.8.1.1 Installation and Configuration

Full instructions for installation and configuration can be found in the INSTALL.txt file from the SEALSQ demonstration code mentioned above. A general set of steps for preparing to run the demonstration code is included below.

1. Install prerequisites on Raspberry Pi
   a. cmake
   b. git
   c. gcc
2. On the Raspberry Pi, run the `sudo raspi-update` command to update drivers

3. Before plugging VaultIC Secure Element into the Raspberry Pi connector, configure the jumpers:
    a. Set _VCC_ jumper
        i. CTRL = VaultIC power controlled by GPIO25 (default)
        ii. 3V3 = VaultIC power always on
    b. Set J1&J2 to select I2C or SPI
        i. If using SPI, set J1 to SS and J2 to SEL (default)
        ii. If using I2C, set J1 to SCL and J2 to SDA
4. Using the `raspi-config` command, enable the SPI or I2C interface on the Raspberry Pi
5. Run `git clone https://github.com/sclark-wisekey/NCCoE.factory.pub` to pull down the demonstration code.

2.8.1.2 Running the demonstration code

1. Navigate to the folder containing the demonstration code. Inside that folder, navigate to the VaultIC/demos folder.
2. Edit the file config.cfg and change the value of VAULTIC_COMM to match with the jumpers configured during setup.
3. The demonstrations are available with wolfSSL stacks and organized in dedicated folders. The README.TXT file in each demonstration subfolder explains how to run the demonstrations.

3 Build 2 (Wi-Fi Easy Connect, CableLabs, OCF)

This section of the practice guide contains detailed instructions for installing and configuring all of the products used to build an instance of the example solution. For additional details on Build 2's logical and physical architectures, see NIST SP 1800-36B: *Approach, Architecture, and Security Characteristics*.

The network-layer onboarding component of Build 2 utilizes Wi-Fi Easy Connect, also known as the Device Provisioning Protocol (DPP). The Wi-Fi Easy Connect standard is maintained by the Wi-Fi Alliance [1]. The term "DPP" is used when referring to the network-layer onboarding protocol, and "Wi-Fi Easy Connect" is used when referring to the overall implementation of the network onboarding process.

3.1 CableLabs Platform Controller

The CableLabs Platform Controller provides an architecture and reference implementation of a cloud-based service that provides management capability for service deployment groups, access points with the deployment groups, registration and lifecycle of user services, and the secure onboarding and lifecycle management of users' Wi-Fi devices. The controller also exposes APIs for integration with third-party systems for the purpose of integrating various business flows (e.g., integration with manufacturing process for device management).

The Platform Controller would typically be hosted by the network operator or a third-party service provider. It can be accessed via web interface. Additional information for this deployment can be accessed at the official CableLabs repository.

3.1.1 Operation and Demonstration

Once configuration of the Platform Controller, Gateway, and Reference Client has been completed, full operation can commence. Instructions for this are located at the official CableLabs repository.

3.2 CableLabs Custom Connectivity Gateway

In this deployment, the gateway software is running on a Raspberry Pi 3B+, which acts as a router, firewall, wireless access point, Open Connectivity Foundation (OCF) Diplomat, and OCF Onboarding Tool. The gateway is also connected to the CableLabs Platform Controller, which manages much of the configuration and functions of the gateway. Due to Build 2's infrastructure and support of Wi-Fi Easy Connect, Build 2 fully supported interoperable network-layer onboarding with Build 1's IoT devices.

3.2.1 Installation and Configuration

Hardware requirements, pre-installation steps, installation steps, and configuration instructions for the gateway can be found at the official CableLabs repository.

3.2.2 Integration with CableLabs Platform Controller

Once initial configuration has occurred, the gateway can be integrated with the CableLabs Platform Controller. Instructions can be found at the official CableLabs repository.

3.2.3 Operation and Demonstration

Once configuration of the Platform Controller, Gateway, and Reference Client has been completed, full operation can commence. Instructions for this are located at the official CableLabs repository.

3.3 Reference Clients/IoT Devices

Three reference clients were deployed in this build, each on a Raspberry Pi 3B+. They were each configured to emulate either a smart light switch or a smart lamp. The software deployed also included the capability to perform network-layer onboarding via Wi-Fi Easy Connect (or DPP) and application-layer onboarding using the OCF onboarding method. These reference clients were fully interoperable with network-layer onboarding to Build 1.

3.3.1 Installation and Configuration

Hardware requirements, pre-installation, installation, and configuration steps for the reference clients are detailed in the official CableLabs repository.

3.3.2 Operation and Demonstration

Once configuration of the Platform Controller, Gateway, and Reference Client has been completed, full operation can commence. Instructions for this are located at the official CableLabs repository.

For interoperability with Build 1, the IoT device's DPP URI was provided to Aruba Central, which allowed Build 1 to successfully complete network-layer onboarding with the Build 2 IoT devices.

4 Build 3 (BRSKI, Sandelman Software Works)

This section of the practice guide contains detailed instructions for installing and configuring all of the products used to build an instance of the example solution. For additional details on Build 3's logical and physical architectures, see NIST SP 1800-36B: *Approach, Architecture, and Security Characteristics*.

The network-layer onboarding component of Build 3 utilizes the Bootstrapping Remote Secure Infrastructure (BRSKI) protocol. Build 3 is representative of a typical home or small office network.

4.1 Onboarding Router/Join Proxy

The onboarding router quarantines the IoT device attempting to join the network until the BRSKI onboarding process is complete. The router in this build is a Turris MOX device, which is based on the Linux OpenWrt version 4 operating system (OS). The Raspberry Pi 3 contains software to function as the Join Proxy for pledges to the network. If another brand of device is used, a different source of compiled Join Proxy might be required.

4.1.1 Setup and Configuration

The router needs to be IPv6 enabled. In the current implementation, the join package operates on an unencrypted network.

4.2 Minerva Join Registrar Coordinator

The purpose of the Join Registrar is to determine whether a new device is allowed to join the network. The Join Registrar is located on a virtual machine running Devuan Linux 4 within the network.

4.2.1 Setup and Configuration

The Minerva Fountain Join Registrar/Coordinator is available as a Docker container and as a VM in OVA format at the Minerva fountain page. Further setup and configuration instructions are available on the Sandelman website on the configuration page.

For the Build 3 demonstration, the VM deployment was installed onto a VMware vSphere system.

A freshly booted VM image will do the following on its own:

- Configure a database
- Configure a local certificate authority (fountain:s0_setup_jrc)
- Configure certificates for the database connection
- Configure certificates for the Registrar https interface
- Configure certificates for use with the Bucardo database replication system
- Configure certificates for LDevID certification authority (fountain:s2_create_registrar)

- Start the JRC

The root user is permitted to log in on the console ("tty0") using the password “root” but is immediately forced to set a new password.

The new registrar will announce itself with the name minerva-fountain.local in mDNS.

The logs for this are put into */var/log/configure-fountain-12345.log* (where 12345 is a new number based upon the PID of the script).

4.3 Reach Pledge Simulator

The Reach Pledge Simulator acts as an IoT device in Build 3. The pledge is acting as an IoT device joining the network and is hosted on a Raspberry Pi 3. More information is available on the Sandelman website on the Reach page.

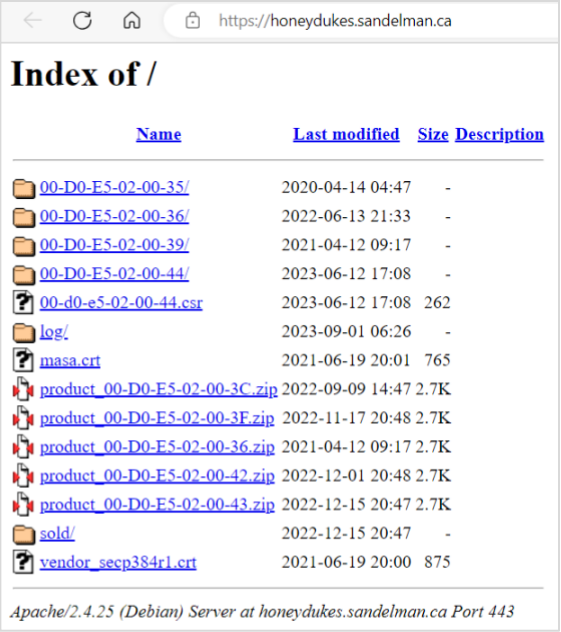
4.3.1 Setup and Configuration

While the functionality of this device is to act as an IoT device, it runs on the same software as the Join Registrar Coordinator. This software is available in both VM and Docker container format. Please see Section 4.2.1 for installation instructions.

When setting up the Reach Pledge Simulator, the address of the Join Registrar Coordinator is automatically determined by the pledge.

Currently, the Reach Pledge Simulator obtains its IDevID using the following steps:

1. View the available packages by visiting the Sandelman website.



https://honeydukes.sandelman.ca

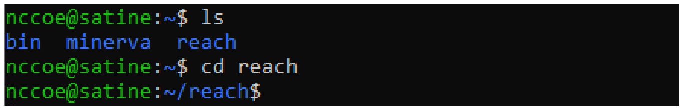
Index of /

Name	Last modified	Size	Description
00-D0-E5-02-00-35/	2020-04-14 04:47	-	
00-D0-E5-02-00-36/	2022-06-13 21:33	-	
00-D0-E5-02-00-39/	2021-04-12 09:17	-	
00-D0-E5-02-00-44/	2023-06-12 17:08	-	
00-d0-e5-02-00-44.csr	2023-06-12 17:08	262	
log/	2023-09-01 06:26	-	
masa.crt	2021-06-19 20:01	765	
product_00-D0-E5-02-00-3C.zip	2022-09-09 14:47	2.7K	
product_00-D0-E5-02-00-3F.zip	2022-11-17 20:48	2.7K	
product_00-D0-E5-02-00-36.zip	2021-04-12 09:17	2.7K	
product_00-D0-E5-02-00-42.zip	2022-12-01 20:48	2.7K	
product_00-D0-E5-02-00-43.zip	2022-12-15 20:47	2.7K	
sold/	2022-12-15 20:47	-	
vendor_secp384r1.crt	2021-06-19 20:00	875	

Apache/2.4.25 (Debian) Server at honeydukes.sandelman.ca Port 443

2. Open a terminal on the Raspberry Pi device and navigate to the Reach directory by entering:

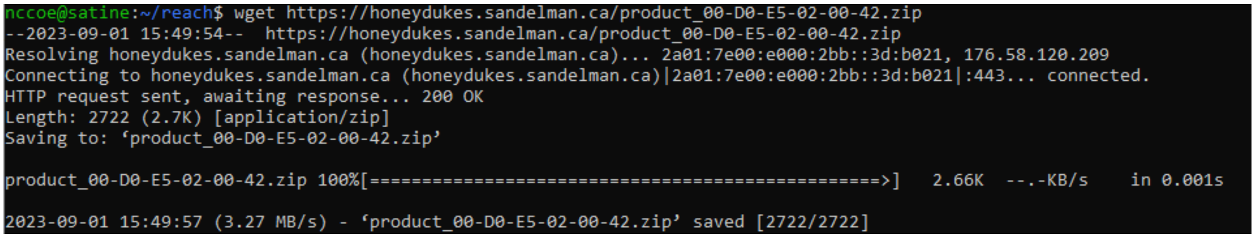
```
cd reach
```



```
nccoe@satine:~$ ls
bin  minerva  reach
nccoe@satine:~$ cd reach
nccoe@satine:~/reach$
```

3. Enter the following command while substituting the URL for one of the available zip files containing the IDevID of choice on the Sandelman website.

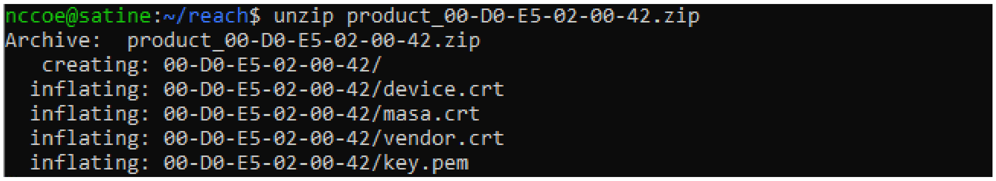
```
wget https://honeydukes.sandelman.ca/product_00-D0-E5-02-00-42.zip
```



```
nccoe@satine:~/reach$ wget https://honeydukes.sandelman.ca/product_00-D0-E5-02-00-42.zip
--2023-09-01 15:49:54--  https://honeydukes.sandelman.ca/product_00-D0-E5-02-00-42.zip
Resolving honeydukes.sandelman.ca (honeydukes.sandelman.ca)... 2a01:7e00:e000:2bb::3d:b021, 176.58.120.209
Connecting to honeydukes.sandelman.ca (honeydukes.sandelman.ca)|2a01:7e00:e000:2bb::3d:b021|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 2722 (2.7K) [application/zip]
Saving to: ‘product_00-D0-E5-02-00-42.zip’

product_00-D0-E5-02-00-42.zip 100%[===============================================>]   2.66K  --.-KB/s    in 0.001s

2023-09-01 15:49:57 (3.27 MB/s) - ‘product_00-D0-E5-02-00-42.zip’ saved [2722/2722]
```

4. Unzip the file by entering the following command, substituting the name of your zip file (the IDevID is the *device.crt* file):

```
unzip product_00-D0-E5-02-00-42.zip
```



```
nccoe@satine:~/reach$ unzip product_00-D0-E5-02-00-42.zip
Archive:  product_00-D0-E5-02-00-42.zip
   creating: 00-D0-E5-02-00-42/
  inflating: 00-D0-E5-02-00-42/device.crt
  inflating: 00-D0-E5-02-00-42/masa.crt
  inflating: 00-D0-E5-02-00-42/vendor.crt
  inflating: 00-D0-E5-02-00-42/key.pem
```

Typically, this would be accomplished through a provisioning process involving a Certificate Authority, as demonstrated in the Factory Provisioning builds.

4.4 Serial Console Server

The serial console server does not participate in the onboarding process but provides direct console access to the IoT devices. The serial console server has been attached to a multi-port USB hub and USB connectors and/or USB2TTL adapters connected to each device. The ESP32 and the nRF52840 are both connected to the serial console and receive power from the USB hub. Power to the console and IoT devices is also provided via the USB hub. A BeagleBone Green device was used as the serial console, using the "screen" program as the telecom device.

4.5 Minerva Highway MASA Server

In the current implementation of the build, the MASA server provides the Reach Pledge Simulator with an IDevID Certificate and a public/private keypair for demonstration purposes. Typically, this would be accomplished through a factory provisioning process involving a Certificate Authority, as demonstrated in the Factory Provisioning builds.

4.5.1 Setup and Configuration

Installation of the Minerva Highway MASA is described at the Highway configuration page. Additional configuration details are available at the Highway development page.

Availability of VMs and containers is described at the following Minerva page.

5 Build 4 (Thread, Silicon Labs, Kudelski IoT)

This section of the practice guide contains detailed instructions for installing and configuring all of the products used to build an instance of the example solution. For additional details on Build 4's logical and physical architectures, see NIST SP 1800-36B: *Approach, Architecture, and Security Characteristics*.

This build utilizes the Thread protocol and performs application-layer onboarding using the Kudelski keySTREAM service to provision a device to the AWS IoT Core.

5.1 Open Thread Border Router

The Open Thread Border Router forms the Thread network and acts as the router on this build. The Open Thread Border Router is run as software on a Raspberry Pi 3B. The Silicon Labs Gecko Wireless Devkit is attached to the Raspberry Pi via USB and acts as the 802.15.4 antenna for this build.

5.1.1 Installation and Configuration

On the Raspberry Pi, run the following commands from a terminal to install and configure the Open Thread Border Router software:

```
git clone https://github.com/openthread/ot-br-posix
sudo NAT64=1 DNS64=1 WEB_GUI=1 ./script/bootstrap
sudo NAT64=1 DNS64=1 WEB_GUI=1 ./script/setup
```

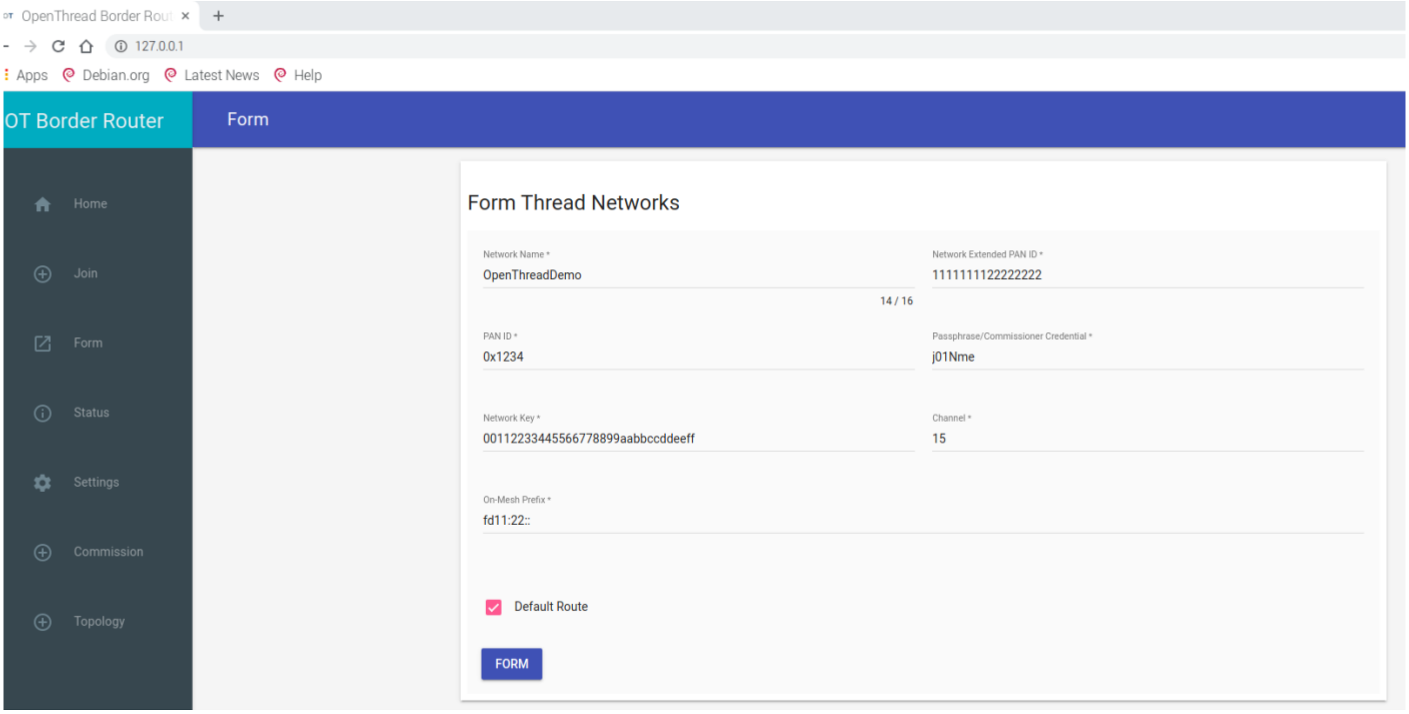
5.1.2 Operation and Demonstration

Once initial configuration has occurred, the OpenThread Border Router should be functional and operated through the web GUI.

1. To open the OpenThread Border Router GUI enter the following IP in a web browser:

   ```
   127.0.0.1
   ```

2. In the **Form** tab, enter the details for the Thread network being formed. For demonstration purposes we only updated the credentials field.



Form Thread Networks

Network Name *
OpenThreadDemo
14 / 16

Network Extended PAN ID *
1111111122222222

PAN ID *
0x1234

Passphrase/Commissioner Credential *
j01Nme

Network Key *
00112233445566778899aabbccddeeff

Channel *
15

On-Mesh Prefix *
fd11:22::

☑ Default Route

FORM

5.2 Silicon Labs Dev Kit (BRD2601A)

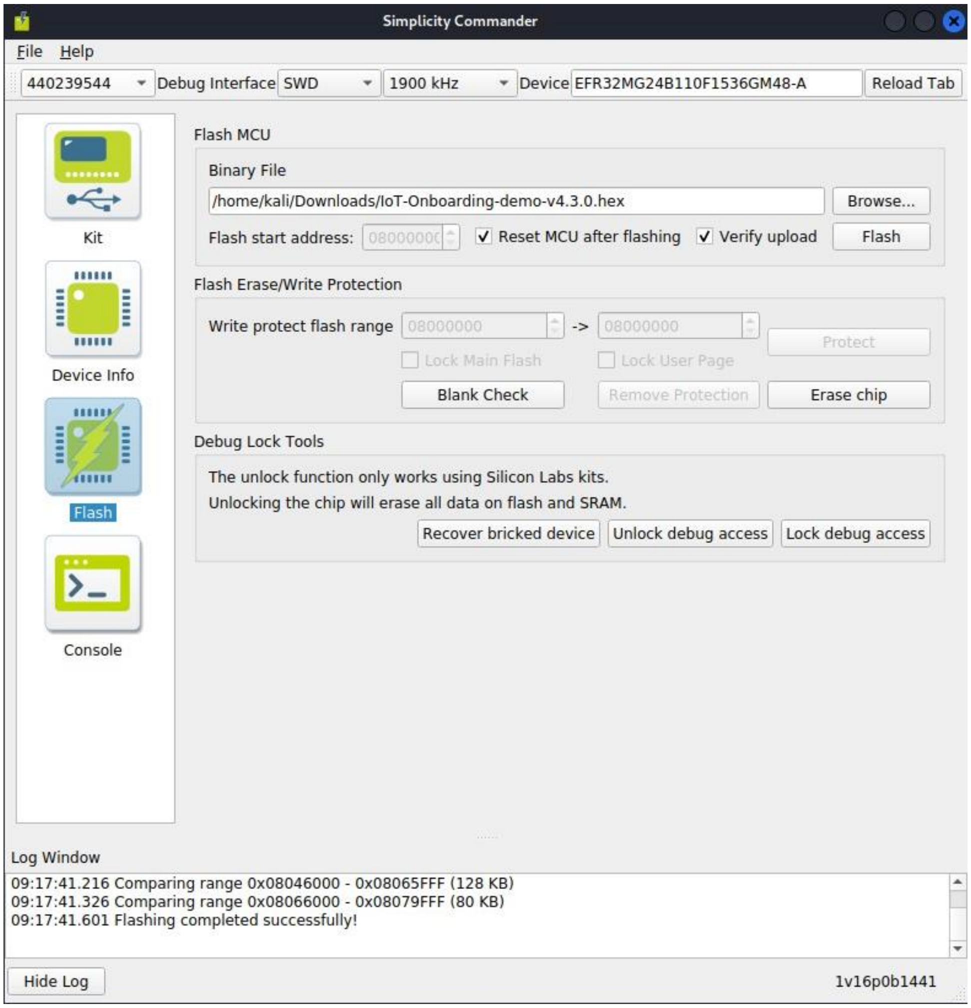
The Silicon Labs Dev Kit acts as the IoT device for this build. It is controlled using the Simplicity Studio v5 Software available at the official Simplicity Studio page and connected to a computer running Windows or Linux via USB. Our implementation leveraged a Linux machine running Simplicity Studio. Custom firmware for the Dev Kit leveraged in this use case was made by Silicon Labs.

5.2.1 Setup and Configuration

The Dev Kit custom firmware image works in conjunction with the Kudelski keySTREAM service. More information is available by contacting Silicon Labs through their contact form. Once the custom firmware has been acquired the Dev Kit can be configured using the following steps.

1. Connect the Dev Kit via USB to the machine running Simplicity Studio.

2. The firmware is installed onto the Dev Kit using the Simplicity Commander tool within Simplicity Studio.



440239544 Debug Interface SWD 1900 kHz Device EFR32MG24B110F1536GM48-A

Binary File: /home/kali/Downloads/IoT-Onboarding-demo-v4.3.0.hex

Flash start address: 08000000 ☑ Reset MCU after flashing ☑ Verify upload

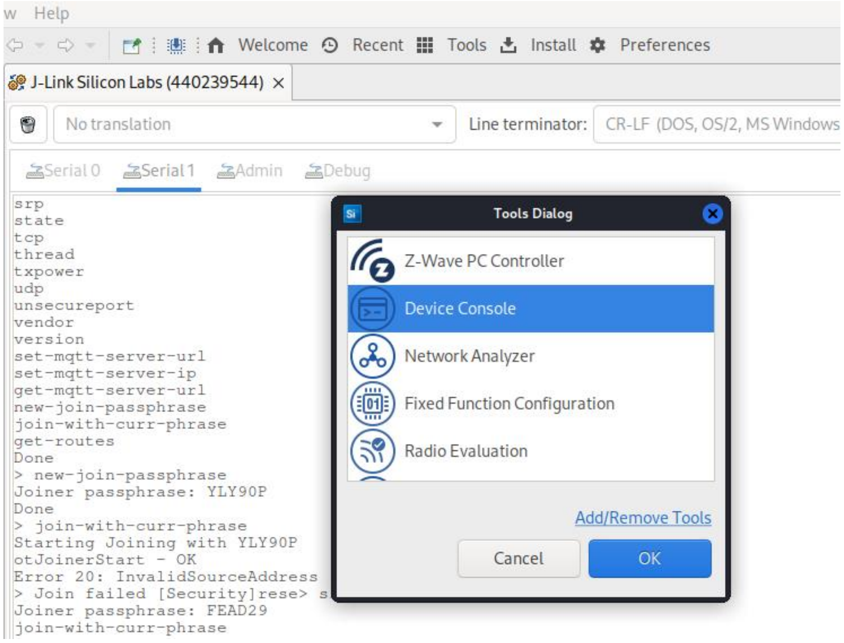
Write protect flash range 08000000 -> 08000000

The unlock function only works using Silicon Labs kits.
Unlocking the chip will erase all data on flash and SRAM.

```
09:17:41.216 Comparing range 0x08046000 - 0x08065FFF (128 KB)
09:17:41.326 Comparing range 0x08066000 - 0x08079FFF (80 KB)
09:17:41.601 Flashing completed successfully!
```

After selecting the firmware file, click **Flash** to flash the firmware the Dev Kit.

3. Open the device console in the **Tools** tab and then select the **Serial 1** tab.

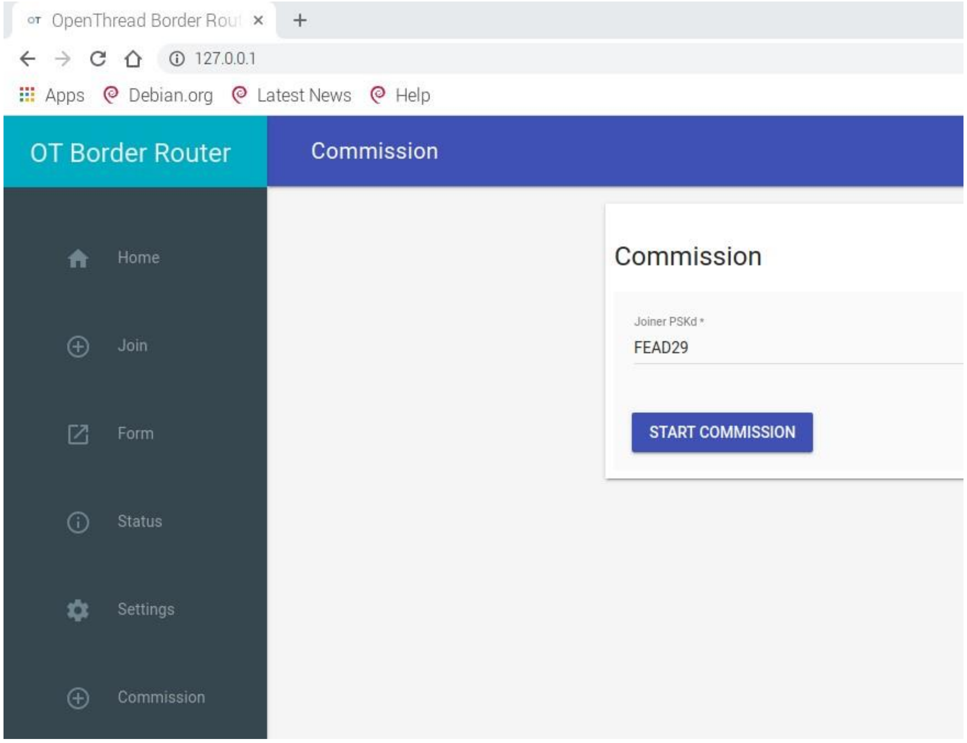


```
srp
state
tcp
thread
txpower
udp
unsecureport
vendor
version
set-mqtt-server-url
set-mqtt-server-ip
get-mqtt-server-url
new-join-passphrase
join-with-curr-phrase
get-routes
Done
> new-join-passphrase
Joiner passphrase: YLY90P
Done
> join-with-curr-phrase
Starting Joining with YLY90P
otJoinerStart - OK
Error 20: InvalidSourceAddress
> Join failed [Security]rese> s
Joiner passphrase: FEAD29
join-with-curr-phrase
```

4. Enter the following command to create a new join passphrase in the Serial 1 command line:

```
new-join-passphrase
```

5. Enter the output of the previous command in the **Commission** tab in the OpenThread Border Router GUI and click **Start Commission.**

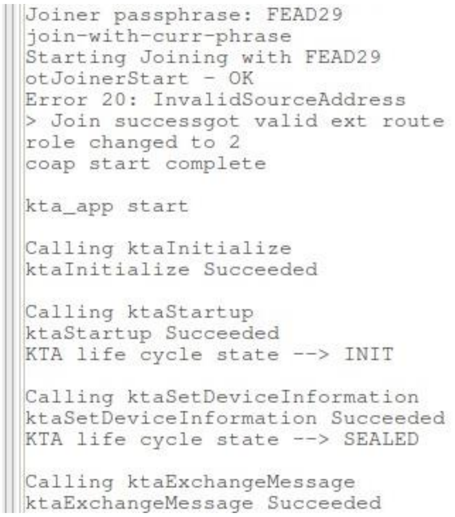


Joiner PSKd *
FEAD29

6. In the Simplicity Commander Device Console, enter the following command to begin the joining process from the Thunderboard:

```
join-with-curr-phrase
```

7. Press the **Reset** button on the Dev Kit and the device will join the thread network and reach out to the Kudelski keySTREAM service. You should see the following output in the Simplicity Commander Device Console:



```
Joiner passphrase: FEAD29
join-with-curr-phrase
Starting Joining with FEAD29
otJoinerStart - OK
Error 20: InvalidSourceAddress
> Join successgot valid ext route
role changed to 2
coap start complete

kta_app start

Calling ktaInitialize
ktaInitialize Succeeded

Calling ktaStartup
ktaStartup Succeeded
KTA life cycle state --> INIT

Calling ktaSetDeviceInformation
ktaSetDeviceInformation Succeeded
KTA life cycle state --> SEALED

Calling ktaExchangeMessage
ktaExchangeMessage Succeeded
```

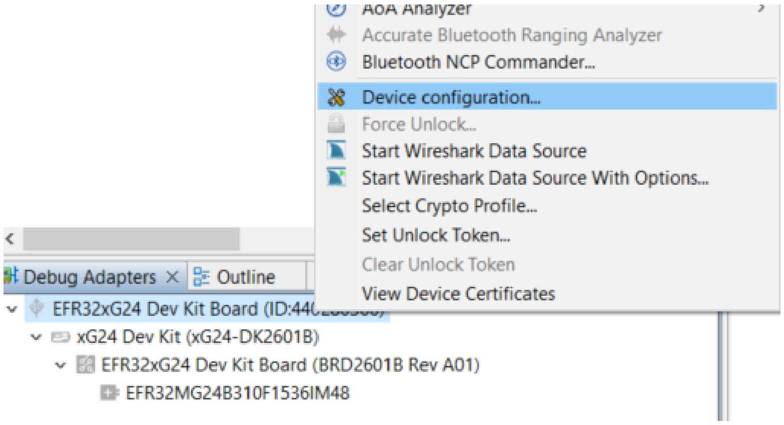
5.3 Kudelski keySTREAM Service

In this section we describe the Kudelski keySTREAM service which this build utilizes to provision certificates for connecting to the AWS IoT core. More information on keySTREAM is available at the keySTREAM page.

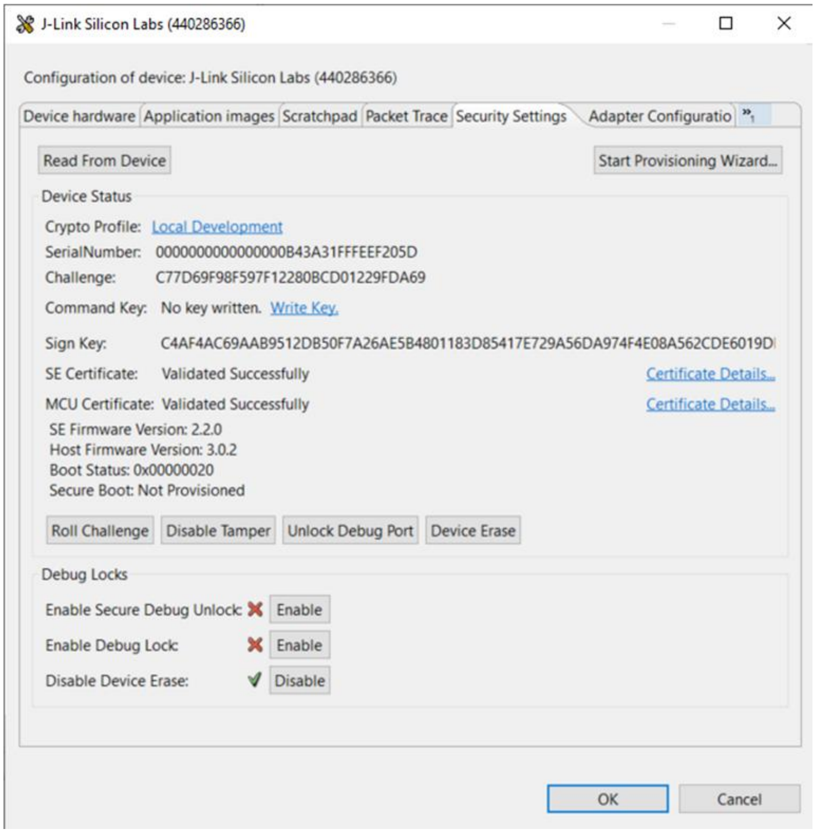
5.3.1 Setup and Configuration

The Kudelski keySTREAM service provides two certificates for the device: a CA certificate and a Proof of Possession (POP) certificate that is generated using a code from the AWS server. This section describes the steps to download these certificates.

1. Locate the Chip UID for the Silicon Labs Dev Kit in Simplicity Studio by right clicking on the **Device Adapters** tab at the bottom and selecting **Device Configuration**.


AoA Analyzer
Accurate Bluetooth Ranging Analyzer
Bluetooth NCP Commander...
Device configuration...
Force Unlock...
Start Wireshark Data Source
Start Wireshark Data Source With Options...
Select Crypto Profile...
Set Unlock Token...
Clear Unlock Token
View Device Certificates
Debug Adapters
Outline
EFR32xG24 Dev Kit Board (ID:440
xG24 Dev Kit (xG24-DK2601B)
EFR32xG24 Dev Kit Board (BRD2601B Rev A01)
EFR32MG24B310F1536IM48

2. On the **Security Settings** tab, take the last 16 characters of the serial number and remove the 'FFFE' characters from the 7th – 11th positions.



J-Link Silicon Labs (440286366)

Configuration of device: J-Link Silicon Labs (440286366)

Device Status

Crypto Profile: Local Development

SerialNumber: 0000000000000000B43A31FFFEEF205D

Challenge: C77D69F98F597F12280BCD01229FDA69

Command Key: No key written. Write Key.

Sign Key: C4AF4AC69AAB9512DB50F7A26AE5B4801183D85417E729A56DA974F4E08A562CDE6019D

SE Certificate: Validated Successfully Certificate Details...

MCU Certificate: Validated Successfully Certificate Details...

SE Firmware Version: 2.2.0
Host Firmware Version: 3.0.2
Boot Status: 0x00000020
Secure Boot: Not Provisioned

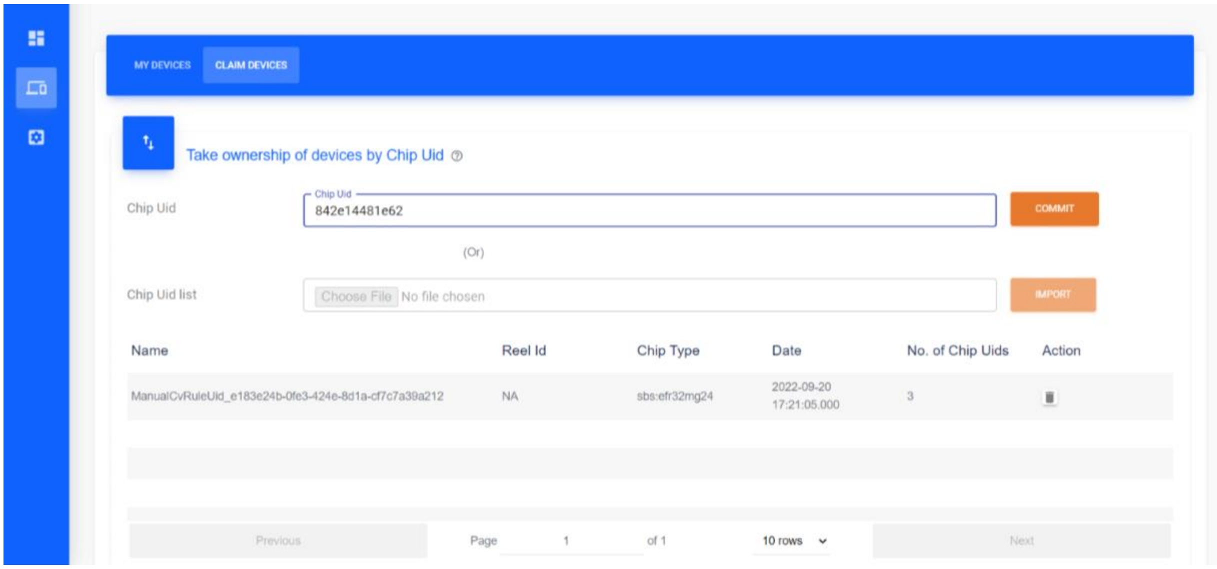
Debug Locks

Enable Secure Debug Unlock: ✗ Enable

Enable Debug Lock: ✗ Enable

Disable Device Erase: ✓ Disable

3. In the Kudelski keySTREAM service, claim your device by entering the chip UID from Simplicity Studio and clicking **Commit**.



Take ownership of devices by Chip Uid

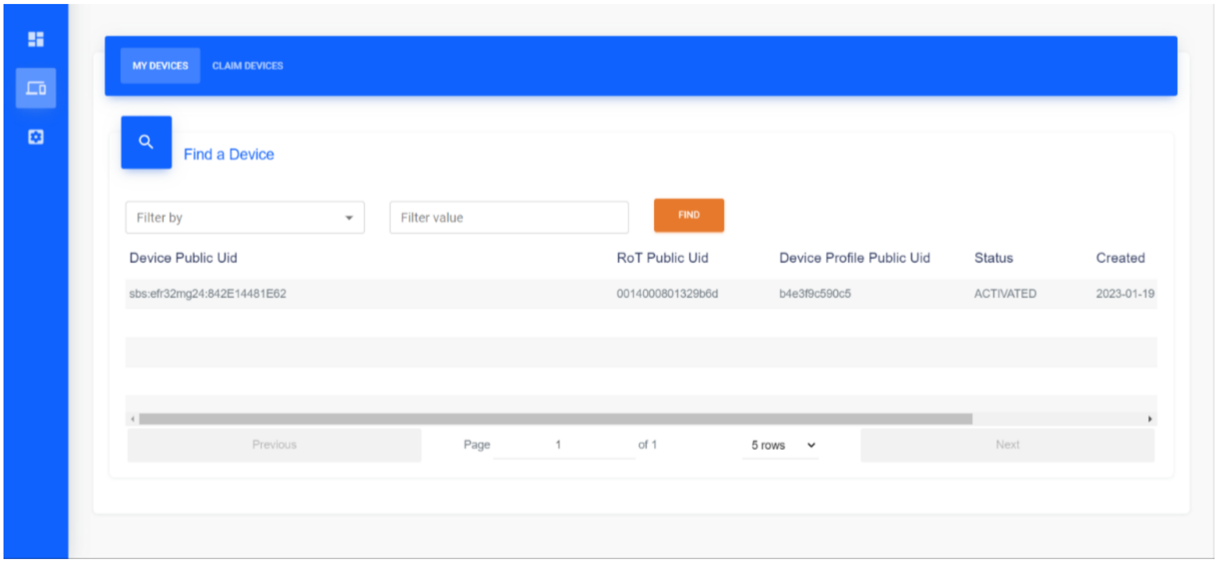
Chip Uid: 842e14481e62

(Or)

Chip Uid list: Choose File No file chosen

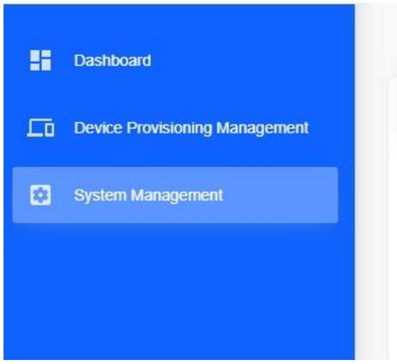
Name	Reel Id	Chip Type	Date	No. of Chip Uids	Action
ManualCvRuleUid_e183e24b-0fe3-424e-8d1a-cf7c7a39a212	NA	sbs:efr32mg24	2022-09-20 17:21:05.000	3	🗑

4. The device will now be visible in the **My Devices** tab. A device can be removed from the keySTREAM service by scrolling to the right and clicking the **Refurbish** button.

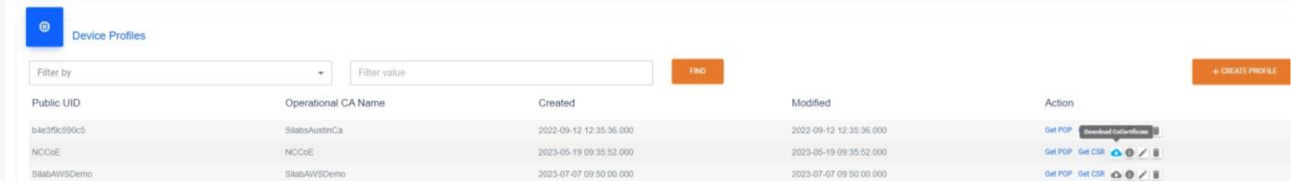


Device Public Uid	RoT Public Uid	Device Profile Public Uid	Status	Created
sbs:efr32mg24:842E14481E62	0014000801329b6d	b4e3f9c590c5	ACTIVATED	2023-01-19

5. Open the **System Management** tab on the left side:



6. Click the cloud icon to download the CA Certificate and the POP certificate, the POP certificate will require a code that is obtained from the AWS IoT Core which will be generated in Section 5.4.1.



Public UID	Operational CA Name	Created	Modified	Action
b4e3f9c590c5	SilabsAustinCa	2022-09-12 12:35:36.000	2022-09-12 12:35:36.000	Get POP
NCCoE	NCCoE	2023-05-19 09:35:52.000	2023-05-19 09:35:52.000	Get POP Get CSR
SilabAWSDemo	SilabAWSDemo	2023-07-07 09:50:00.000	2023-07-07 09:50:00.000	Get POP Get CSR

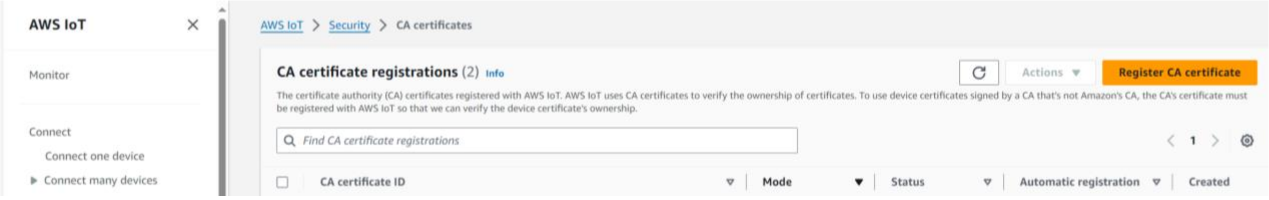
5.4 AWS IoT Core

The Silicon Labs Dev Kit will connect to the AWS MQTT test client using the certificates provisioned from the Kudelski keySTREAM service.

5.4.1 Setup and Configuration

Application-layer onboarding for this build is performed using the AWS MQTT test client. Certificates provisioned from the Kudelski keySTREAM service are uploaded to an AWS instance and the device will demonstrate its ability to successfully send a message to AWS.

1. Within the AWS IoT Core, open the **Security** drop-down menu, click on **Certificate authorities**, and click the **Register CA certificate** button on the right.



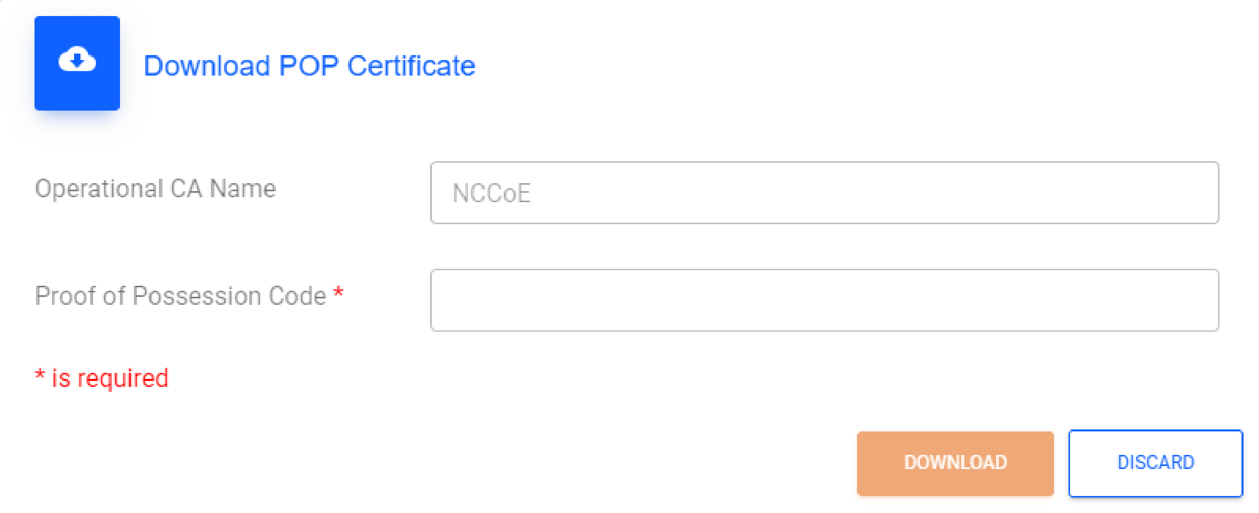
CA certificate registrations (2)

The certificate authority (CA) certificates registered with AWS IoT. AWS IoT uses CA certificates to verify the ownership of certificates. To use device certificates signed by a CA that's not Amazon's CA, the CA's certificate must be registered with AWS IoT so that we can verify the device certificate's ownership.

Find CA certificate registrations

□	CA certificate ID	Mode	Status	Automatic registration	Created

2. Select the radio button for **Register CA in Single-account mode** and copy the registration code to use as the **Proof of Possession Code** in the Kudelski keySTREAM service and download the POP certificate.

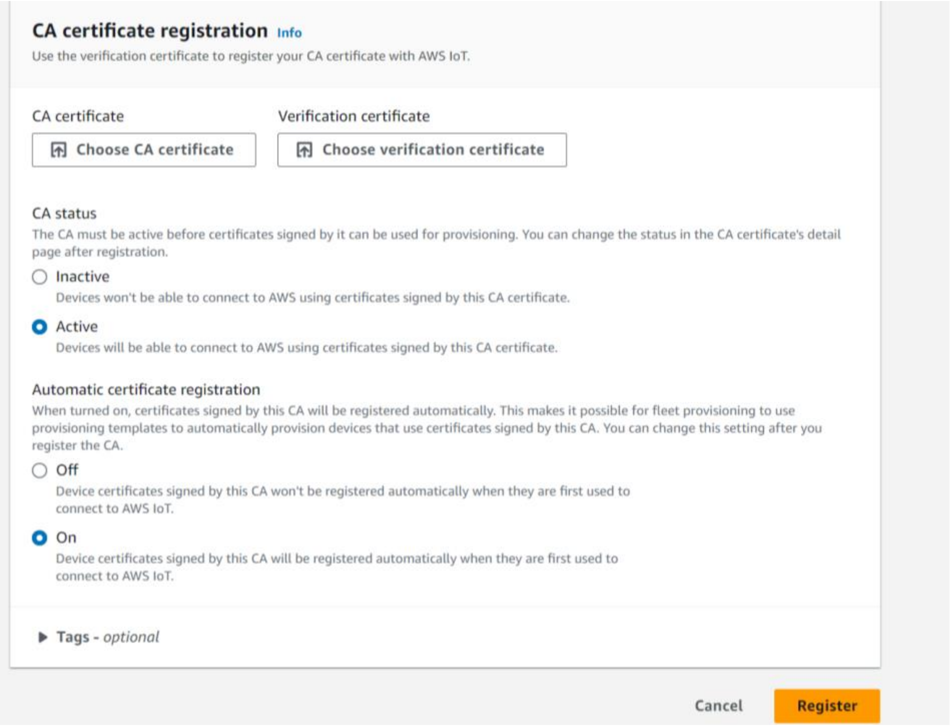


Operational CA Name: NCCoE

Proof of Possession Code *

* is required

3. After downloading the POP certificate, upload the CA certificate and the POP (verification) certificate, and select the radio buttons for **Active** under **CA Status** and **On** under **Automatic Certificate Registration**. Then click **Register**.



CA certificate registration Info

Use the verification certificate to register your CA certificate with AWS IoT.

CA certificate: Choose CA certificate

Verification certificate: Choose verification certificate

CA status

The CA must be active before certificates signed by it can be used for provisioning. You can change the status in the CA certificate's detail page after registration.

○ Inactive
Devices won't be able to connect to AWS using certificates signed by this CA certificate.

● Active
Devices will be able to connect to AWS using certificates signed by this CA certificate.

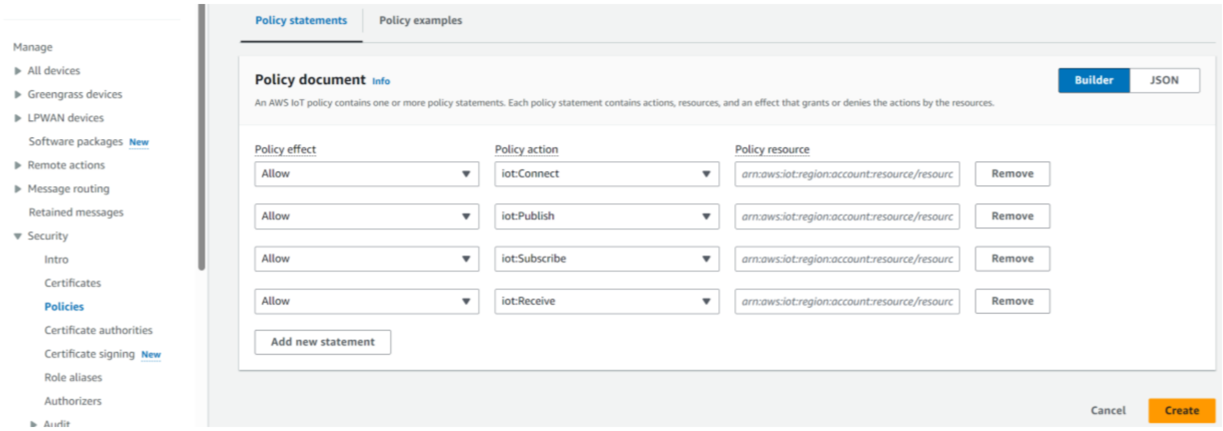
Automatic certificate registration

When turned on, certificates signed by this CA will be registered automatically. This makes it possible for fleet provisioning to use provisioning templates to automatically provision devices that use certificates signed by this CA. You can change this setting after you register the CA.

○ Off
Device certificates signed by this CA won't be registered automatically when they are first used to connect to AWS IoT.

● On
Device certificates signed by this CA will be registered automatically when they are first used to connect to AWS IoT.

4. In the **Security** drop down menu, click on **Policies** and add the policies shown below. Then, click **Create**.

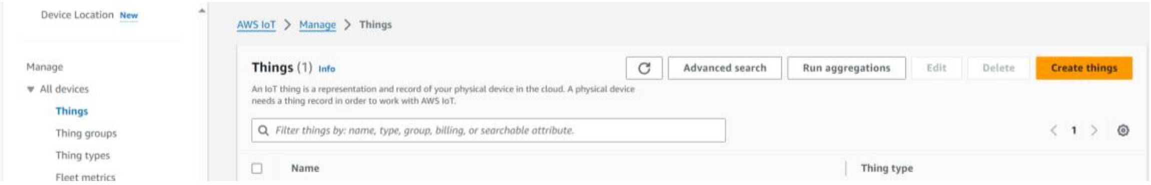


Policy document Info

An AWS IoT policy contains one or more policy statements. Each policy statement contains actions, resources, and an effect that grants or denies the actions by the resources.

Policy effect	Policy action	Policy resource	
Allow	iot:Connect	arn:aws:iot:region:account:resource/resourc	Remove
Allow	iot:Publish	arn:aws:iot:region:account:resource/resourc	Remove
Allow	iot:Subscribe	arn:aws:iot:region:account:resource/resourc	Remove
Allow	iot:Receive	arn:aws:iot:region:account:resource/resourc	Remove

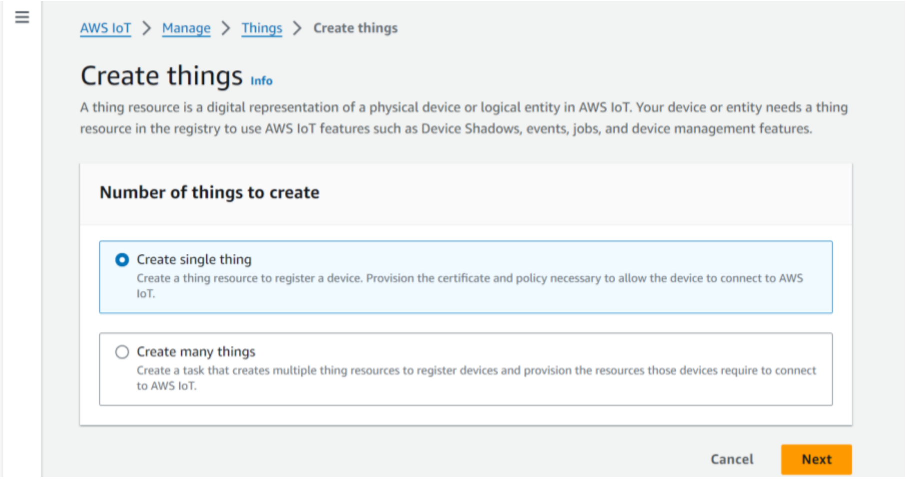
5. In the **All devices** drop-down menu, click on **Things** and click **Create things**.



Things (1) Info

An IoT thing is a representation and record of your physical device in the cloud. A physical device needs a thing record in order to work with AWS IoT.

6. Click the **Create single thing** radio button and click **Next**.


AWS IoT > Manage > Things > Create things

Create things Info

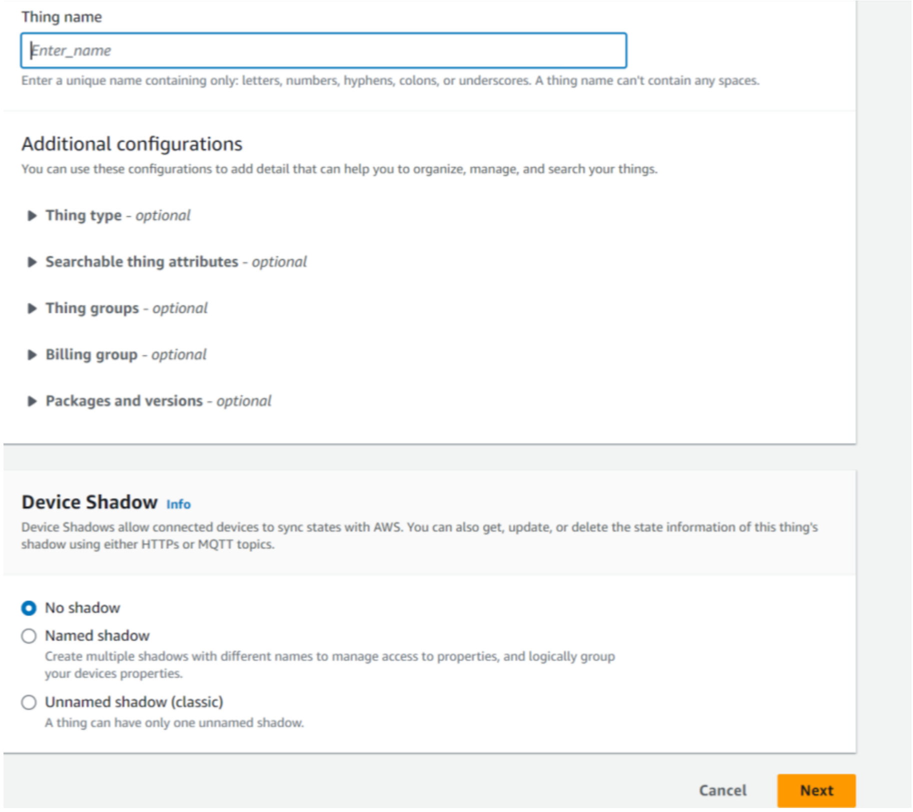
A thing resource is a digital representation of a physical device or logical entity in AWS IoT. Your device or entity needs a thing resource in the registry to use AWS IoT features such as Device Shadows, events, jobs, and device management features.

Number of things to create

● Create single thing
Create a thing resource to register a device. Provision the certificate and policy necessary to allow the device to connect to AWS IoT.

○ Create many things
Create a task that creates multiple thing resources to register devices and provision the resources those devices require to connect to AWS IoT.

832 7. Enter a **Thing name** and click **Next.**


Thing name

Enter_name

Enter a unique name containing only: letters, numbers, hyphens, colons, or underscores. A thing name can't contain any spaces.

Additional configurations

You can use these configurations to add detail that can help you to organize, manage, and search your things.

▶ Thing type - *optional*

▶ Searchable thing attributes - *optional*

▶ Thing groups - *optional*

▶ Billing group - *optional*

▶ Packages and versions - *optional*

Device Shadow Info

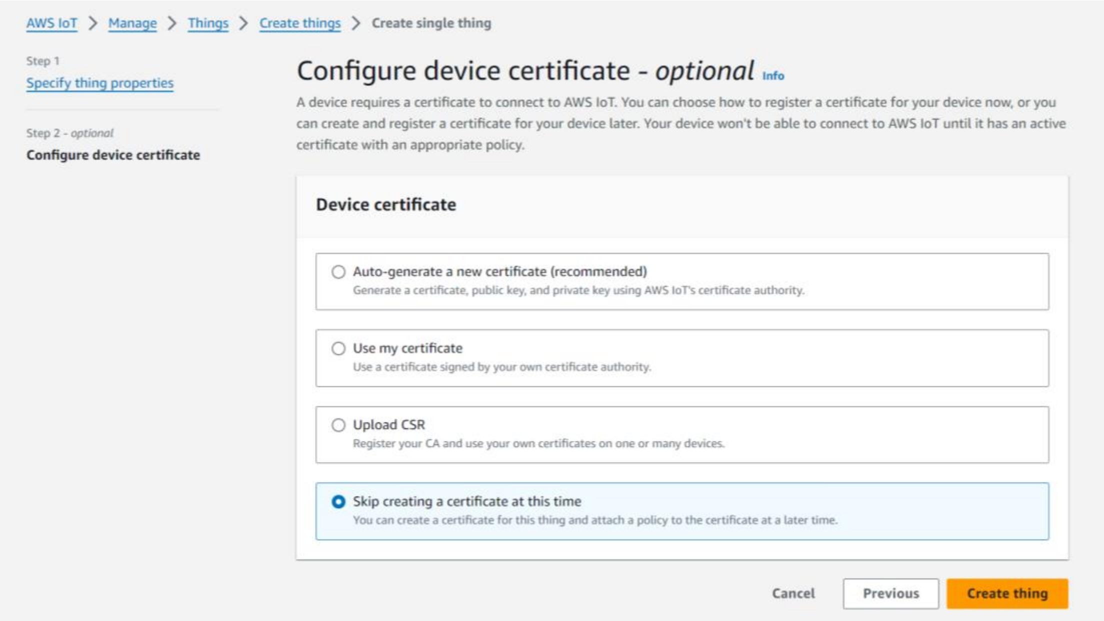
Device Shadows allow connected devices to sync states with AWS. You can also get, update, or delete the state information of this thing's shadow using either HTTPs or MQTT topics.

● No shadow

○ Named shadow
Create multiple shadows with different names to manage access to properties, and logically group your devices properties.

○ Unnamed shadow (classic)
A thing can have only one unnamed shadow.

833 8. Select the **Skip creating a certificate at this time** radio button and click **Create thing**.



AWS IoT > Manage > Things > Create things > Create single thing

Step 1
Specify thing properties

Step 2 - *optional*
Configure device certificate

Configure device certificate - *optional* Info

A device requires a certificate to connect to AWS IoT. You can choose how to register a certificate for your device now, or you can create and register a certificate for your device later. Your device won't be able to connect to AWS IoT until it has an active certificate with an appropriate policy.

Device certificate

○ Auto-generate a new certificate (recommended)
Generate a certificate, public key, and private key using AWS IoT's certificate authority.

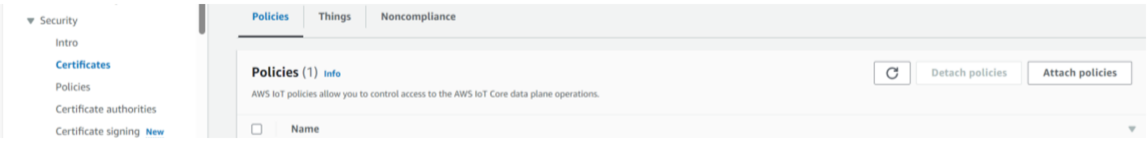
○ Use my certificate
Use a certificate signed by your own certificate authority.

○ Upload CSR
Register your CA and use your own certificates on one or many devices.

● Skip creating a certificate at this time
You can create a certificate for this thing and attach a policy to the certificate at a later time.

9. In the **Security** drop-down menu, click on **Certificates** and click the Certificate ID of the certificate that you created.

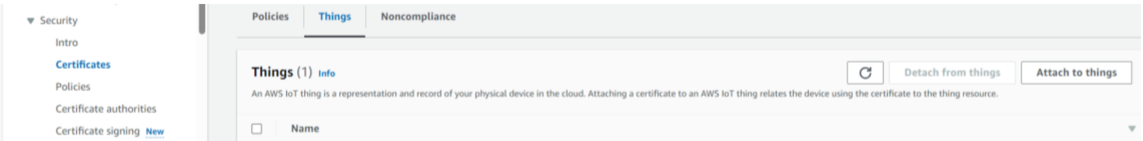
10. In the **Policies** tab at the bottom, click **Attach policies** and add the policy that you created.



Policies (1) Info

AWS IoT policies allow you to control access to the AWS IoT Core data plane operations.

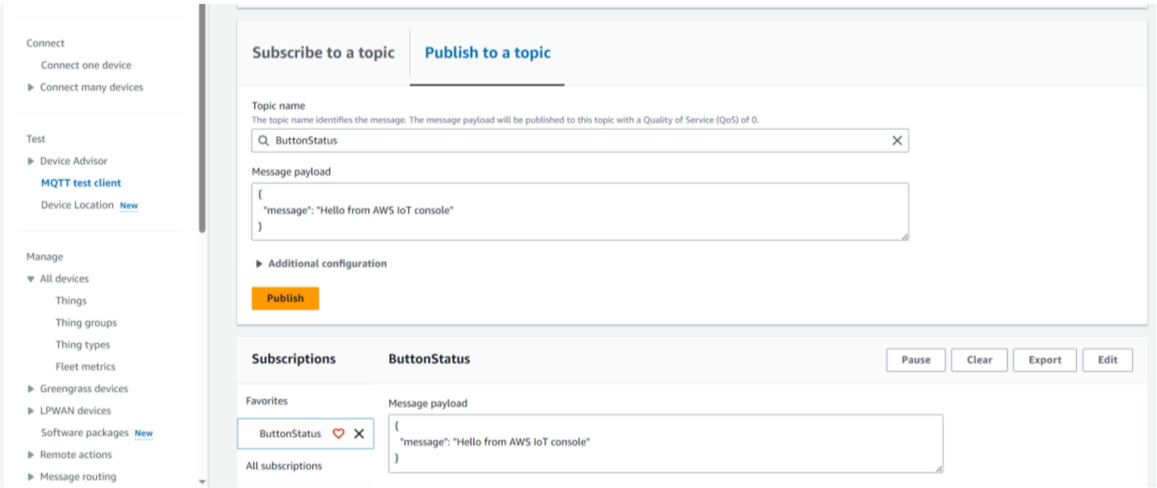
11. In the **Things** tab, click **Attach to things** and add the thing that you created.



Things (1) Info

An AWS IoT thing is a representation and record of your physical device in the cloud. Attaching a certificate to an AWS IoT thing relates the device using the certificate to the thing resource.

12. Click the **MQTT test client** on the left side of the page and click the **Publish to a topic** tab.

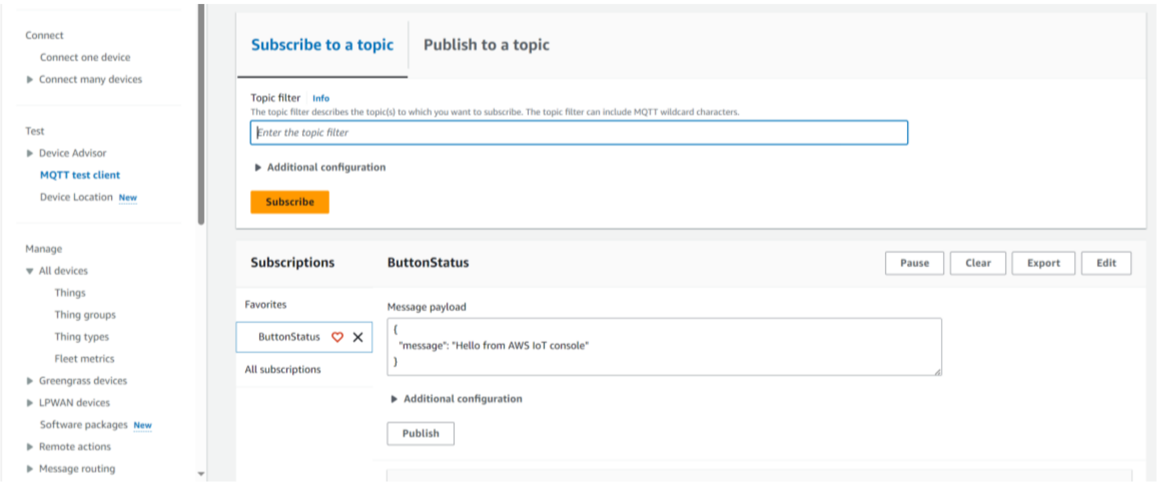


Topic name
The topic name identifies the message. The message payload will be published to this topic with a Quality of Service (QoS) of 0.
ButtonStatus

Message payload
```
{
 "message": "Hello from AWS IoT console"
}
```

Message payload
```
{
 "message": "Hello from AWS IoT console"
}
```

13. Create a message of your choosing and click **Publish**. On the **Subscribe to a topic** tab, make sure that you are subscribed to the topic that you just created.



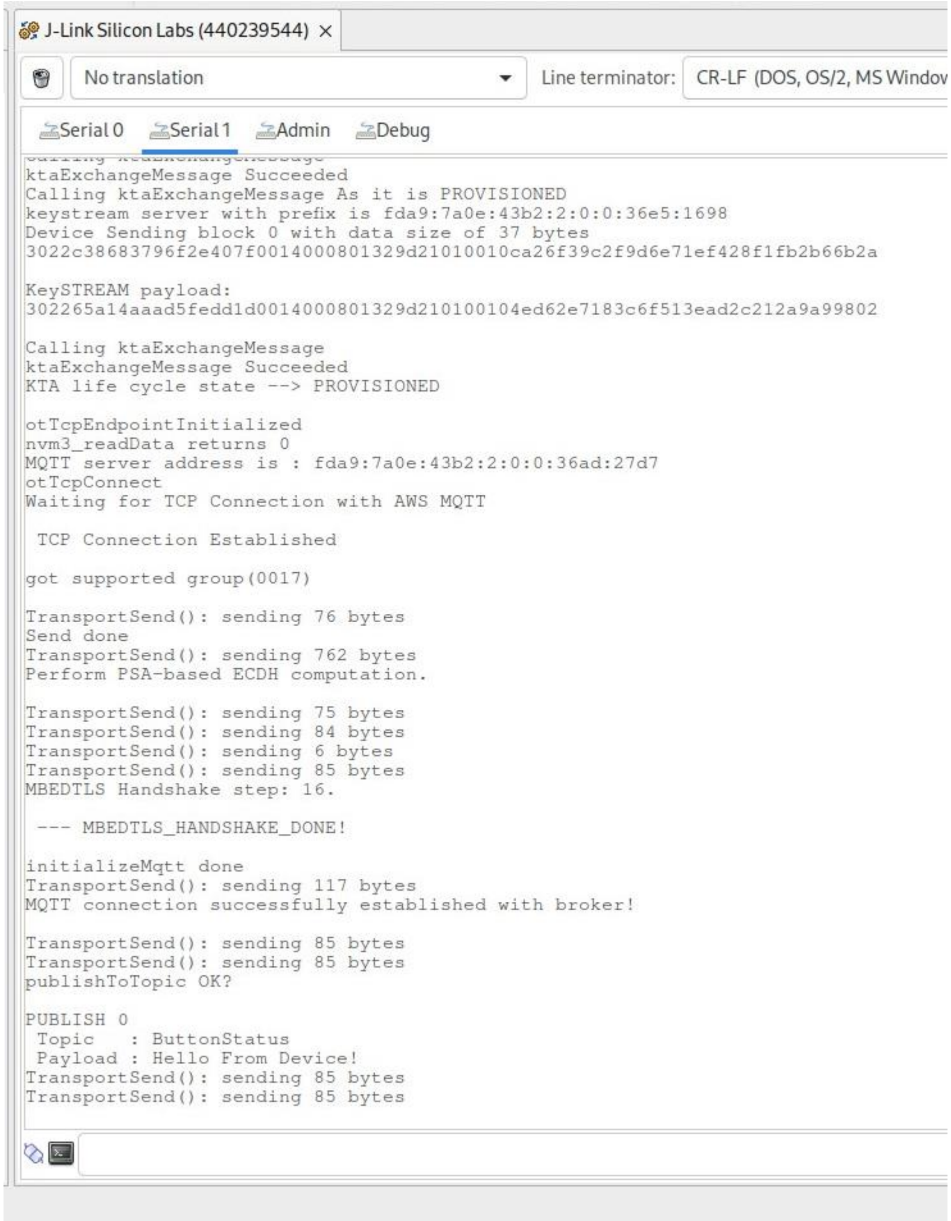
The topic filter describes the topic(s) to which you want to subscribe. The topic filter can include MQTT wildcard characters.

```
{
  "message": "Hello from AWS IoT console"
}
```

5.4.2 Testing

Information sent and received by the Silicon Labs Dev Kit to the MQTT test client will be displayed in the device console in Simplicity Commander. This section describes testing the communication between the MQTT test client and the device.

1. On the Thunderboard, press **Button 0**. This will begin the connection to the MQTT test client.


J-Link Silicon Labs (440239544) ×
No translation
Line terminator:
CR-LF (DOS, OS/2, MS Window
Serial 0
Serial 1
Admin
Debug

```
Calling ktaExchangeMessage
ktaExchangeMessage Succeeded
Calling ktaExchangeMessage As it is PROVISIONED
keystream server with prefix is fda9:7a0e:43b2:2:0:0:36e5:1698
Device Sending block 0 with data size of 37 bytes
3022c38683796f2e407f0014000801329d21010010ca26f39c2f9d6e71ef428f1fb2b66b2a

KeySTREAM payload:
302265a14aaad5fedd1d0014000801329d210100104ed62e7183c6f513ead2c212a9a99802

Calling ktaExchangeMessage
ktaExchangeMessage Succeeded
KTA life cycle state --> PROVISIONED

otTcpEndpointInitialized
nvm3_readData returns 0
MQTT server address is : fda9:7a0e:43b2:2:0:0:36ad:27d7
otTcpConnect
Waiting for TCP Connection with AWS MQTT

 TCP Connection Established

got supported group(0017)

TransportSend(): sending 76 bytes
Send done
TransportSend(): sending 762 bytes
Perform PSA-based ECDH computation.

TransportSend(): sending 75 bytes
TransportSend(): sending 84 bytes
TransportSend(): sending 6 bytes
TransportSend(): sending 85 bytes
MBEDTLS Handshake step: 16.

 --- MBEDTLS_HANDSHAKE_DONE!

initializeMqtt done
TransportSend(): sending 117 bytes
MQTT connection successfully established with broker!

TransportSend(): sending 85 bytes
TransportSend(): sending 85 bytes
publishToTopic OK?

PUBLISH 0
 Topic   : ButtonStatus
 Payload : Hello From Device!
TransportSend(): sending 85 bytes
TransportSend(): sending 85 bytes
```

6 Build 5 (BRSKI over Wi-Fi, NquiringMinds)

This section of the practice guide contains detailed instructions for installing and configuring all of the products used to build an instance of the example solution. For additional details on Build 5's logical and physical architectures, see NIST SP 1800-36B: *Approach, Architecture, and Security Characteristics*.

The network-layer onboarding component of Build 5 utilizes the BRSKI protocol.

6.1 Pledge

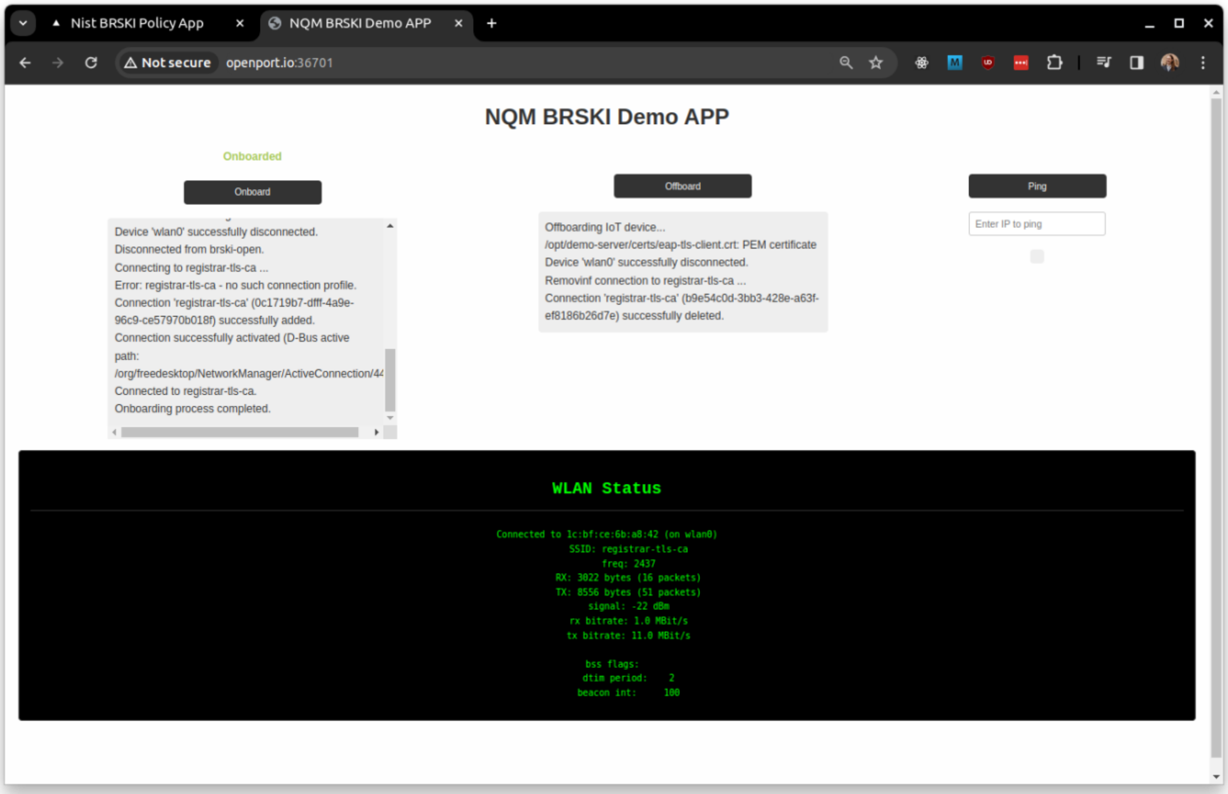
The Pledge acts as the IoT device which is attempted to onboard onto the secure network. It implements the pledge functionality as per the IETF BRSKI specification. It consists of software provided by NquiringMinds running on a Raspberry Pi Model 4B.

6.1.1 Installation and Configuration

Hardware requirements, pre-installation steps, installation steps, and configuration instructions for the pledge device can be found at the official NquiringMinds repository.

6.1.2 Operation and Demonstration

To demonstrate the onboarding and offboarding functionality, NquiringMinds has provided a web application which runs on the pledge device. It features a button one can use to manually run the onboarding script and display the output of the onboarding process, as well as a button for offboarding. It also features a button to ping an IP address, which is configured to ping the designated address via the wireless network interface.



```
Device 'wlan0' successfully disconnected.
Disconnected from brski-open.
Connecting to registrar-tls-ca ...
Error: registrar-tls-ca - no such connection profile.
Connection 'registrar-tls-ca' (0c1719b7-dfff-4a9e-
96c9-ce57970b018f) successfully added.
Connection successfully activated (D-Bus active
path:
/org/freedesktop/NetworkManager/ActiveConnection/44
Connected to registrar-tls-ca.
Onboarding process completed.
```

```
Offboarding IoT device...
/opt/demo-server/certs/eap-tls-client.crt: PEM certificate
Device 'wlan0' successfully disconnected.
Removinf connection to registrar-tls-ca ...
Connection 'registrar-tls-ca' (b9e54c0d-3bb3-428e-a63f-
ef8186b26d7e) successfully deleted.
```

WLAN Status

```
Connected to 1c:bf:ce:6b:a8:42 (on wlan0)
        SSID: registrar-tls-ca
        freq: 2437
        RX: 3022 bytes (16 packets)
        TX: 8556 bytes (51 packets)
        signal: -22 dBm
        rx bitrate: 1.0 MBit/s
        tx bitrate: 11.0 MBit/s

        bss flags:
        dtim period:    2
        beacon int:     100
```

6.2 Router and Logical Services

The router and logical services were hosted on a Raspberry Pi Model 4B equipped with 2 external Wi-Fi adapters. These additional Wi-Fi adapters are needed to support VLAN tagging which is a hardware dependent feature. The details of the physical setup and all connections are provided in the official NquiringMinds documentation.

6.2.1 Installation and Configuration

All of the services described in the next section can be installed on a Raspberry Pi using the installer provided by NquiringMinds.

The demonstration services can also be built from source code, if needed. The following links provide the instructions for building each of those services:

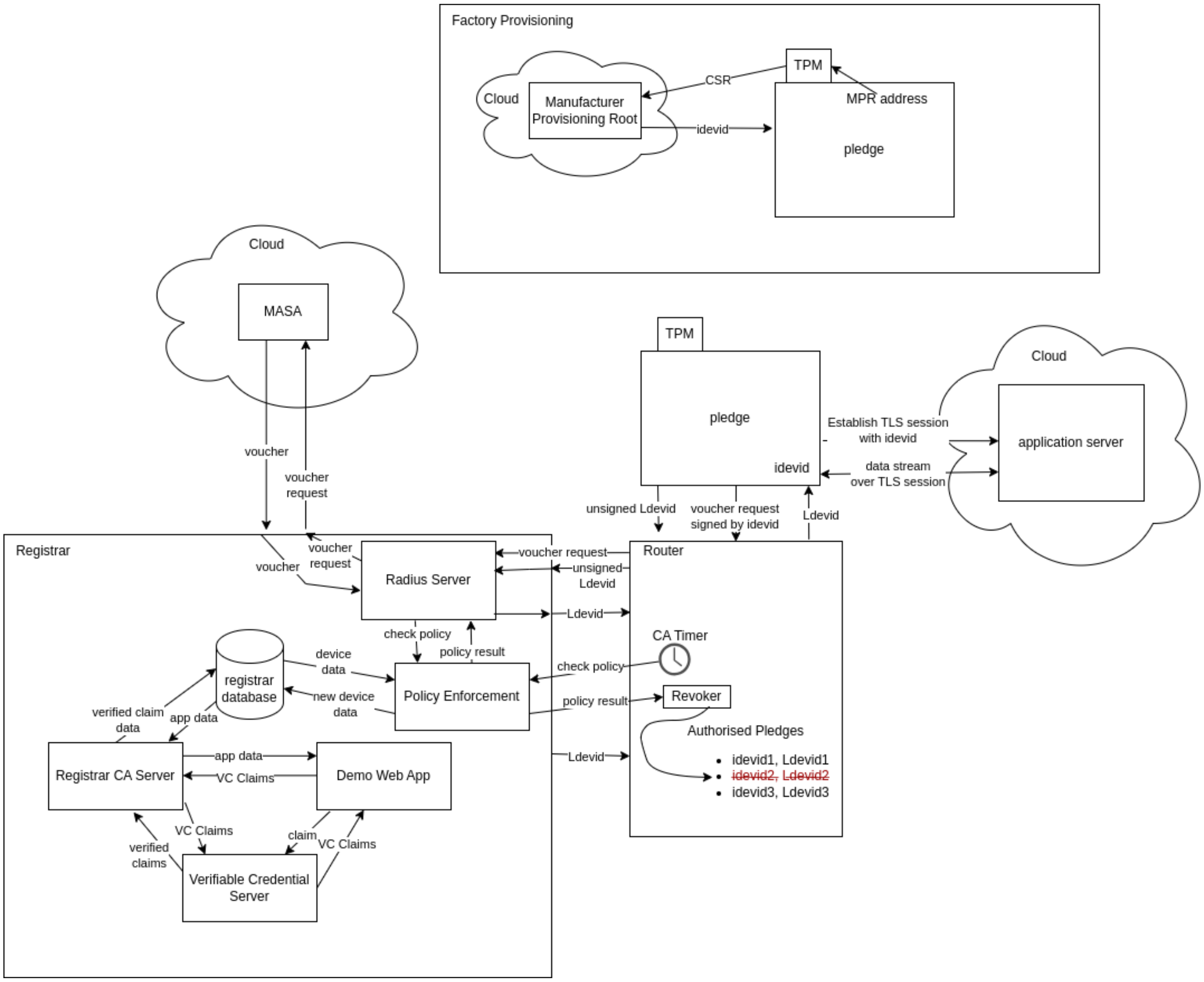
- BRSKI Demo Setup
- EAP Config
- MDNS publishing services

6.2.2 Logical services

The following logical services are installed on the Registrar and services device. The implementation of these services are to be found at the following repository links: NIST BRSKI implementation and BRSKI.

Figure 6-1 below describes how these entities and logical services fit together to perform the BRSKI flow, and a top-level view of how information is transmitted throughout the services to onboard the pledge.

Figure 6-1 Logical Services for Build 5



6.2.2.1 MASA

The MASA currently resides as a local service on the registrar. In practice, this service would be located on an external server managed by the manufacturer. The MASA verifies that the IDevID is authentic, and that the IDevID was produced by the manufacturer's MPR.

6.2.2.2 Manufacturer Provisioning Root (MPR)

The MPR sits on an external server and provides the IDevID (X.509 Certificate) for the device to initialize it after production and notarize it with a unique identity. The address of the MPR is built into the firmware of the device at build time.

6.2.2.3 Registrar

Build 5's BRSKI Domain Registrar runs the BRSKI protocol modified to work over Wi-Fi and functions as the Domain Registrar to authenticate the IoT devices, receive and transfer voucher requests and responses to and from the MASA and ultimately determines whether network-layer onboarding of the device is authorized to take place on the respective network. NquiringMinds has developed a stateful non-persistent Linux app for android that serves this purpose.

The registrar is responsible for verifying if the IDevID certificate provided by the pledge is authentic, by verifying it with the MASA and verifying that the policy for a pledge to be allowed onto the closed secure network has been met. It also runs continuous assurance periodically to ensure that the device still meets the policy requirements, revoking the pledge's access if at a later time it doesn't meet the policy requirements. Signed verifiable credential claims may be submitted to the registrar to communicate information about entities, which it uses to update its database used to determine if the policy is met, the tdx Volt is used to facilitate signing and verification of verifiable credentials. In the demonstrator system the MASA and router are integrated into the same physical device.

6.2.2.3.1 Radius server (Continuous Assurance Client)

To provide continuous assurance capabilities for connected IoT devices, the registrar includes a Radius server that integrates with the Continuous Assurance Server.

The continuous assurance policy is enforced by a script which periodically runs to check that the policy conditions are met. It accomplishes this by querying the Registrar's SQLite database. For the demonstration, the defined policy is:

- The manufacturer and device must be trusted by a user with appropriate privileges
- The device must have a device type associated
- The vulnerability score of the SBOM for the device type must be lower than 6
- The device must not have contacted a denylisted IP address within the last 2 minutes

If the device fails any of these checks, the device will be offboarded.

6.2.2.4 Continuous Assurance Server

The registrar runs several services used to power the continuous assurance flow.

6.2.2.4.1 Verifiable Credential Server

The verifiable credential server is used to sign verifiable credentials submitted through the Demo web app and verify verifiable credentials submitted to the registrar, it is powered by the functionality of the tdx Volt, a local instance of which is run on the registrar.

The code for the Verifiable Credential Server is hosted at the GitHub repository.

6.2.2.4.2 Registrar Continuous Assurance Server

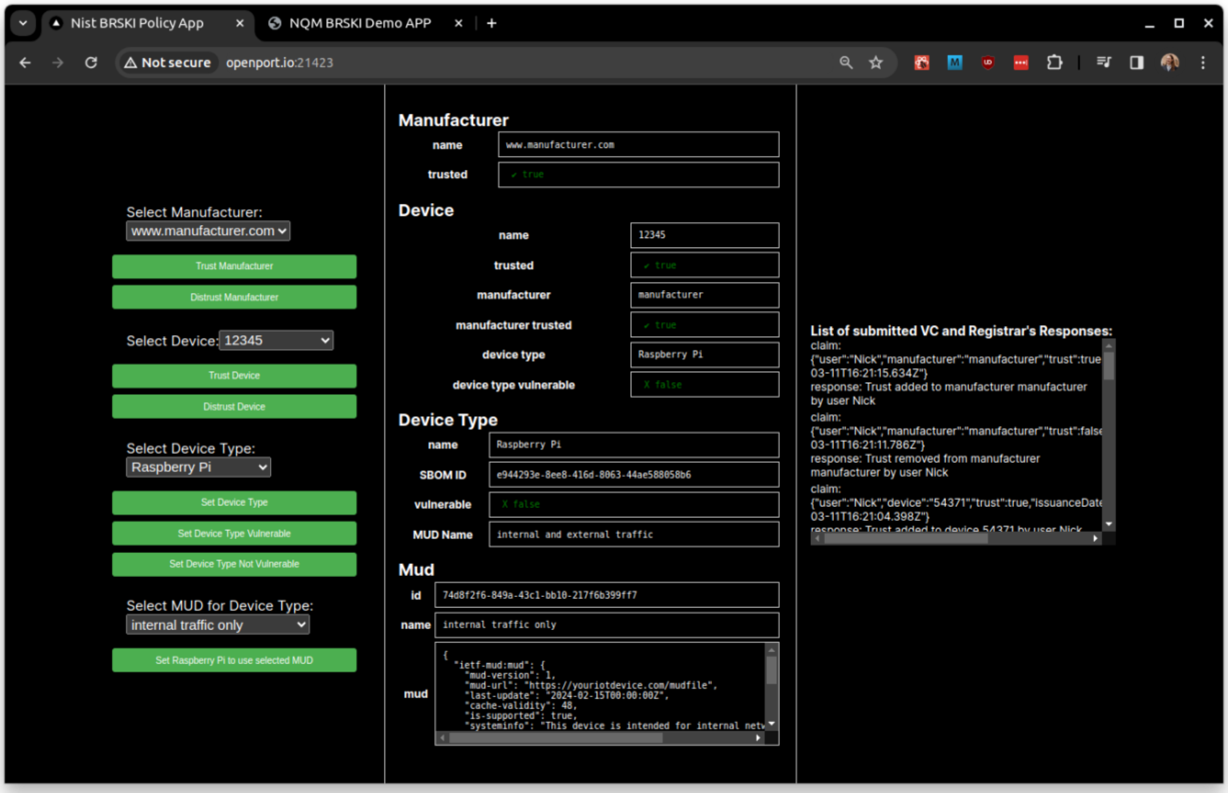
The registrar hosts a REST API which is used to interface with the registrar's SQLite database which stores information about the entities the registrar knows of. This server utilizes the verifiable credential server to verify submitted verifiable credential claims submitted to it.

The code for the Registrar Continuous Assurance Server is hosted at the GitHub repository.

6.2.2.4.3 Demo Web Application

The demo web application is used as an interactive user-friendly way to administer the registrar. Users can view the list of verifiable credentials submitted to the registrar. The application also displays the state of the manufacturers, devices, device types and Manufacturer Usage Description (MUD). There are buttons provided which allow you to trust or distrust a manufacturer, trust or distrust a device, set the device type for a device, set if a device type is vulnerable or not and set the MUD file associated with the device type. All of these operations are performed by generating a verifiable credential containing the claim being made, which is then submitted to the verifiable credential server to sign the credential. The signed verifiable credential is then sent to the registrar continuous assurance server to be verified and used to update the SQLite database on the registrar.

The code for the Demo Web Application is hosted at the GitHub repository.



Select Manufacturer: www.manufacturer.com

Select Device: 12345

Select Device Type: Raspberry Pi

Select MUD for Device Type: internal traffic only

Manufacturer

name: www.manufacturer.com
trusted: ✓ true

Device

name: 12345
trusted: ✓ true
manufacturer: manufacturer
manufacturer trusted: ✓ true
device type: Raspberry Pi
device type vulnerable: X false

Device Type

name: Raspberry Pi
SBOM ID: e944293e-8ee8-416d-8063-44ae588058b6
vulnerable: X false
MUD Name: internal and external traffic

Mud

id: 74d8f2f6-849a-43c1-bb10-217f6b399ff7
name: internal traffic only
mud:

```
{
  "ietf-mud:mud": {
    "mud-version": 1,
    "mud-url": "https://youriotdevice.com/mudfile",
    "last-update": "2024-02-15T00:00:00Z",
    "cache-validity": 48,
    "is-supported": true,
    "systeminfo": "This device is intended for internal netw
```

List of submitted VC and Registrar's Responses:

```
claim:
{"user":"Nick","manufacturer":"manufacturer","trust":true
03-11T16:21:15.634Z"}
response: Trust added to manufacturer manufacturer
by user Nick
claim:
{"user":"Nick","manufacturer":"manufacturer","trust":false
03-11T16:21:11.786Z"}
response: Trust removed from manufacturer
manufacturer by user Nick
claim:
{"user":"Nick","device":"54371","trust":true,"issuanceDate
03-11T16:21:04.398Z"}
response: Trust added to device 54371 by user Nick
```

6.2.2.5 Application server

The application server sits on a remote server and represents the server for an application which should consume data from the pledge device. The pledge device uses the IDevID certificate to establish a secure TLS connection to onboard onto the application server and begin sending data autonomously, currently OpenSSL s_client is used from the pledge to establish a TLS session with the application server, running on a server off-site, and the date and CPU temperature are sent to be logged on the application server, as a proof of principle.

6.2.2.5.1 Installation/Configuration

Hardware requirements, pre-installation steps, installation steps, and configuration instructions for the router can be found at the official NquiringMinds repository.

6.2.2.5.2 Operation/Demonstration

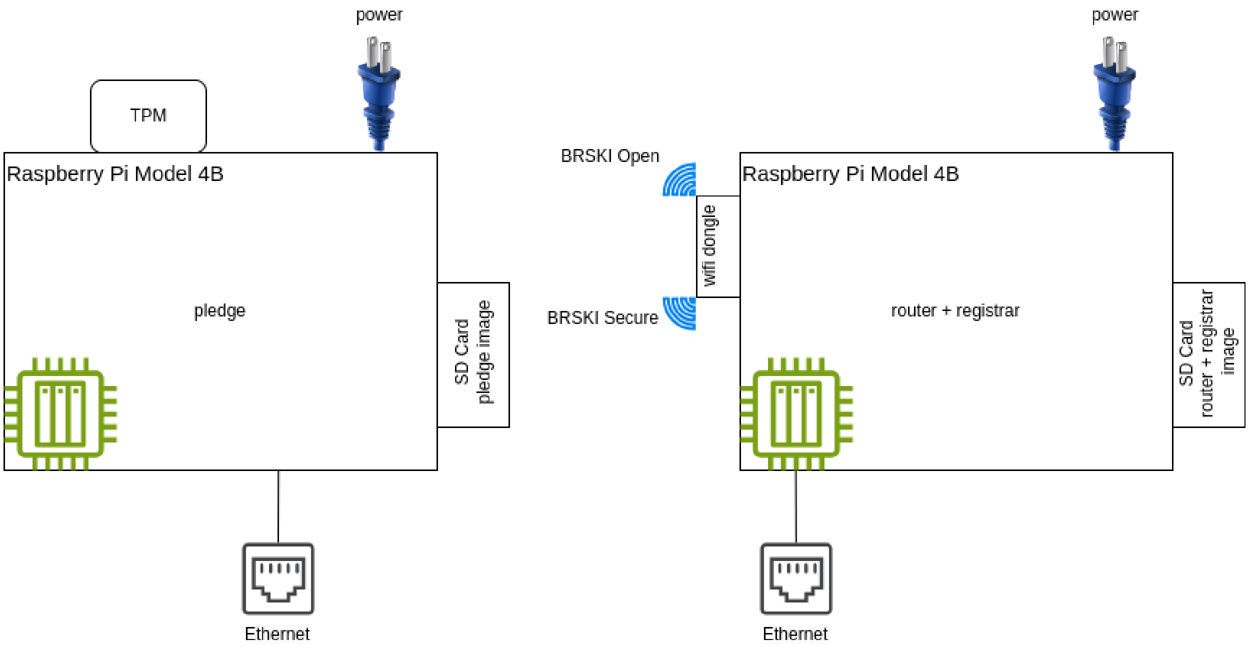
The instructions to use this factory use case code to provision an IDevID onto your pledge are also located at the official NquiringMinds repository in the above section.

6.3 Onboarding Demonstration

6.3.1 Prerequisites

Prior to beginning the demonstration, the router and pledge devices must be connected to power, and to the network via their ethernet port. On boot, both devices should start the services required to demonstrate the BRSKI flow.

Figure 6-2 Diagram of Physical/Logical Components Used to Demonstrate BRSKI Flow



To support the demo and debug features the pledge and the registrar need to be connected to physical ethernet, ideally with internet access. They should still function without an internet connection, but the vulnerability scores of the SBOMs will not be updated and the demo web apps will only be accessible on the local network.

The detailed networking setup details are available in the NquiringMinds NIST Trusted Onboarding Build-5.

6.3.2 Onboarding Demonstration

Once configuration of the devices and the prerequisite conditions have been achieved, the onboarding demonstration can be executed following NquiringMinds Demo Continuous Assurance Workflow.

6.3.3 Continuous Assurance Demonstration

The instructions to demonstrate the continuous assurance workflow are contained in the official NquiringMinds documentation.

6.4 BRSKI Factory Provisioning Build

This Factory Provisioning Build includes many of the components listed in Section 6.2, including the Pledge, Registrar, and other services. An Infineon Secure Element was also included in the build and provides secure generation and storage of the key material and certificates provisioned to the device.

6.4.1 Pledge

The Pledge acts as the IoT device which is attempting to onboard onto the secure network. It implements the pledge functionality as per the IETF BRSKI specification. It consists of a Raspberry Pi Model 4B equipped with an Infineon Optiga SLB 9670 TPM 2.0 Secure Element. The Infineon Secure Element was connected to a Raspberry Pi via the built-in GPIO pins present on the Pi.

6.4.1.1 Factory Use Case - IDevID provisioning

NquiringMinds provided demonstration code that generates a public/private keypair within the secure element, creates a CSR, and uses that CSR to obtain an IDevID certificate from tdx Volt. The demonstration process can be found at the official NquiringMinds documentation.

Initially, it generates a CSR using the TPM secure element to sign it, it then sends the CSR to the MPR server which is the manufacturer's IDevID Certificate Authority and is bootstrapped in the vanilla firmware on the pledge's creation in the factory. The MPR sends back a unique IDevID for the pledge which it stores in its secure element.

The code for this is hosted at the official NquiringMinds repository.

6.4.2 Installation and Configuration

Hardware requirements, pre-installation steps, installation steps, and configuration instructions for the pledge can be found at the official NquiringMinds repository referenced above.

6.4.3 Operation and Demonstration

The instructions to use this factory provisioning use case code to provision an IDevID onto the pledge is also located in the official NquiringMinds repository referenced above.

994 Appendix A List of Acronyms

AKM	Authentication and Key Management
AOS	ArubaOS
AP	Access Point
API	Application Programming Interface
ASN.1	Abstract Syntax Notation One
AWS	Amazon Web Services
BRSKI	Bootstrapping Remote Secure Key Infrastructure
BSS	Basic Service Set
CA	Certificate Authority
CRADA	Cooperative Research and Development Agreement
CSR	Certificate Signing Request
DMZ	Demilitarized Zone
DPP	Device Provisioning Protocol (Wi-Fi Easy Connect)
EAP	Extensible Authentication Protocol
GPIO	General Purpose Input/Output
GUI	Graphical User Interface
HPE	Hewlett Packard Enterprise
IaaS	Infrastructure as a Service
IDevID	Initial Device Identifier
IEEE	Institute of Electrical and Electronics Engineers
IoT	Internet of Things
IPv4	Internet Protocol Version 4
IPv6	Internet Protocol Version 6
LDevID	Locally Significant Device Identifier
MASA	Manufacturer Authorized Signing Authority
MPR	Manufacturer Provisioning Root
MUD	Manufacturer Usage Description
MQTT	MQ Telemetry Transport

NCCoE	National Cybersecurity Center of Excellence
NIST	National Institute of Standards and Technology
OCF	Open Connectivity Foundation
OS	Operating System
OTBR	Open Thread Border Router
PNG	Portable Network Graphics
POP	Proof of Possession
QR	Quick-Response
RF	Radio Frequency
SBOM	Software Bill of Materials
SP	Special Publication
SoC	System-on-Chip
SSID	Service Set Identifier
TPM	Trusted Platform Module
UID	Unique Identifier
URI	Uniform Resource Identifier
USB	Universal Serial Bus
UXI	User Experience Insight
VLAN	Virtual Local Area Network
VM	Virtual Machine
WLAN	Wireless Local Area Network
WPA2	Wi-Fi Protected Access 2
WPA3	Wi-Fi Protected Access 3

Appendix B References

[1] Wi-Fi Alliance. *Wi-Fi Easy Connect*. Available: https://www.wi-fi.org/discover-wi-fi/wi-fi-easy-connect.

NIST SPECIAL PUBLICATION 1800-36D

Trusted Internet of Things (IoT) Device Network-Layer Onboarding and Lifecycle Management

Enhancing Internet Protocol-Based IoT Device and Network Security

Volume D:
Functional Demonstrations

Paul Watrobski
Murugiah Souppaya
National Cybersecurity Center of Excellence
Information Technology Laboratory

Andy Dolan
Kyle Haefner
Craig Pratt
Darshak Thakore
CableLabs,
Louisville, Colorado

Brecht Wyseur
Kudelski IoT, Cheseaux-sur-Lausanne,
Switzerland

Nick Allott
Ashley Setter
Nquiring Minds
Southampton, United Kingdom

Michael Richardson
Sandleman Software Works
Ontario, Canada

Mike Dow
Steve Egerter
Silicon Labs,
Austin, Texas

Chelsea Deane
Joshua Klosterman
Blaine Mulugeta
Charlie Rearick
Susan Symington
The MITRE Corporation
McLean, Virginia

May 2024

DRAFT

This publication is available free of charge from
https://www.nccoe.nist.gov/projects/trusted-iot-device-network-layer-onboarding-and-lifecycle-management

DISCLAIMER

Certain commercial entities, equipment, products, or materials may be identified by name or company logo or other insignia in order to acknowledge their participation in this collaboration or to describe an experimental procedure or concept adequately. Such identification is not intended to imply special status or relationship with NIST or recommendation or endorsement by NIST or NCCoE; neither is it intended to imply that the entities, equipment, products, or materials are necessarily the best available for the purpose.

National Institute of Standards and Technology Special Publication 1800-36D, Natl. Inst. Stand. Technol. Spec. Publ. 1800-36D, 51 pages, May 2024, CODEN: NSPUE2

FEEDBACK

You can improve this guide by contributing feedback. As you review and adopt this solution for your own organization, we ask you and your colleagues to share your experience and advice with us.

Comments on this publication may be submitted to: iot-onboarding@nist.gov.

Public comment period: May 31, 2024 through July 30, 2024

National Cybersecurity Center of Excellence
National Institute of Standards and Technology
100 Bureau Drive
Mailstop 2002
Gaithersburg, MD 20899
Email: nccoe@nist.gov

NATIONAL CYBERSECURITY CENTER OF EXCELLENCE

The National Cybersecurity Center of Excellence (NCCoE), a part of the National Institute of Standards and Technology (NIST), is a collaborative hub where industry organizations, government agencies, and academic institutions work together to address businesses' most pressing cybersecurity issues. This public-private partnership enables the creation of practical cybersecurity solutions for specific industries, as well as for broad, cross-sector technology challenges. Through consortia under Cooperative Research and Development Agreements (CRADAs), including technology partners—from Fortune 50 market leaders to smaller companies specializing in information technology security—the NCCoE applies standards and best practices to develop modular, adaptable example cybersecurity solutions using commercially available technology. The NCCoE documents these example solutions in the NIST Special Publication 1800 series, which maps capabilities to the NIST Cybersecurity Framework and details the steps needed for another entity to re-create the example solution. The NCCoE was established in 2012 by NIST in partnership with the State of Maryland and Montgomery County, Maryland.

To learn more about the NCCoE, visit https://www.nccoe.nist.gov/. To learn more about NIST, visit https://www.nist.gov.

NIST CYBERSECURITY PRACTICE GUIDES

NIST Cybersecurity Practice Guides (Special Publication 1800 series) target specific cybersecurity challenges in the public and private sectors. They are practical, user-friendly guides that facilitate the adoption of standards-based approaches to cybersecurity. They show members of the information security community how to implement example solutions that help them align with relevant standards and best practices, and provide users with the materials lists, configuration files, and other information they need to implement a similar approach.

The documents in this series describe example implementations of cybersecurity practices that businesses and other organizations may voluntarily adopt. These documents do not describe regulations or mandatory practices, nor do they carry statutory authority.

KEYWORDS

application-layer onboarding; bootstrapping; Internet of Things (IoT); Manufacturer Usage Description (MUD); network-layer onboarding; onboarding; Wi-Fi Easy Connect.

ACKNOWLEDGMENTS

We are grateful to the following individuals for their generous contributions of expertise and time.

Name	Organization
Amogh Guruprasad Deshmukh	Aruba, a Hewlett Packard Enterprise company
Dan Harkins	Aruba, a Hewlett Packard Enterprise company
Danny Jump	Aruba, a Hewlett Packard Enterprise company
Bart Brinkman	Cisco
Eliot Lear	Cisco
Peter Romness	Cisco
Tyler Baker	Foundries.io
George Grey	Foundries.io
David Griego	Foundries.io
Fabien Gremaud	Kudelski IoT
Faith Ryan	The MITRE Corporation
Toby Ealden	NquiringMinds
John Manslow	NquiringMinds
Antony McCaigue	NquiringMinds
Alexandru Mereacre	NquiringMinds
Loic Cavaille	NXP Semiconductors
Mihai Chelalau	NXP Semiconductors
Julien Delplancke	NXP Semiconductors
Anda-Alexandra Dorneanu	NXP Semiconductors

Name	Organization
Todd Nuzum	NXP Semiconductors
Nicusor Penisoara	NXP Semiconductors
Laurentiu Tudor	NXP Semiconductors
Michael Richardson	Sandelman Software Works
Karen Scarfone	Scarfone Cybersecurity
Steve Clark	SEALSQ, a subsidiary of WISeKey
Pedro Fuentes	SEALSQ, a subsidiary of WISeKey
Gweltas Radenac	SEALSQ, a subsidiary of WISeKey
Kalvin Yang	SEALSQ, a subsidiary of WISeKey

The Technology Partners/Collaborators who participated in this build submitted their capabilities in response to a notice in the Federal Register. Respondents with relevant capabilities or product components were invited to sign a Cooperative Research and Development Agreement (CRADA) with NIST, allowing them to participate in a consortium to build this example solution. We worked with:

Technology Collaborators		
Aruba, a Hewlett Packard Enterprise company	Kudelski IoT	Sandelman Software Works
CableLabs	NquiringMinds	Silicon Labs
Cisco	NXP Semiconductors	SEALSQ, a subsidiary of WISeKey
Foundries.io	Open Connectivity Foundation (OCF)	

DOCUMENT CONVENTIONS

The terms “shall” and “shall not” indicate requirements to be followed strictly to conform to the publication and from which no deviation is permitted. The terms “should” and “should not” indicate that among several possibilities, one is recommended as particularly suitable without mentioning or excluding others, or that a certain course of action is preferred but not necessarily required, or that (in the negative form) a certain possibility or course of action is discouraged but not prohibited. The terms “may” and “need not” indicate a course of action permissible within the limits of the publication. The terms “can” and “cannot” indicate a possibility and capability, whether material, physical, or causal.

CALL FOR PATENT CLAIMS

This public review includes a call for information on essential patent claims (claims whose use would be required for compliance with the guidance or requirements in this Information Technology Laboratory (ITL) draft publication). Such guidance and/or requirements may be directly stated in this ITL Publication or by reference to another publication. This call also includes disclosure, where known, of the existence of pending U.S. or foreign patent applications relating to this ITL draft publication and of any relevant unexpired U.S. or foreign patents.

ITL may require from the patent holder, or a party authorized to make assurances on its behalf, in written or electronic form, either:

a) assurance in the form of a general disclaimer to the effect that such party does not hold and does not currently intend holding any essential patent claim(s); or

b) assurance that a license to such essential patent claim(s) will be made available to applicants desiring to utilize the license for the purpose of complying with the guidance or requirements in this ITL draft publication either:

1. under reasonable terms and conditions that are demonstrably free of any unfair discrimination; or
2. without compensation and under reasonable terms and conditions that are demonstrably free of any unfair discrimination.

Such assurance shall indicate that the patent holder (or third party authorized to make assurances on its behalf) will include in any documents transferring ownership of patents subject to the assurance, provisions sufficient to ensure that the commitments in the assurance are binding on the transferee, and that the transferee will similarly include appropriate provisions in the event of future transfers with the goal of binding each successor-in-interest.

The assurance shall also indicate that it is intended to be binding on successors-in-interest regardless of whether such provisions are included in the relevant transfer documents.

Such statements should be addressed to: iot-onboarding@nist.gov.

Contents

1 Introduction 1
1.1 How to Use This Guide 1
2 Functional Demonstration Playbook 3
2.1 Scenario 0: Factory Provisioning 3
2.2 Scenario 1: Trusted Network-Layer Onboarding 4
2.3 Scenario 2: Trusted Application-Layer Onboarding 5
2.4 Scenario 3: Re-Onboarding a Device 6
2.5 Scenario 4: Ongoing Device Validation 7
2.6 Scenario 5: Establishment and Maintenance of Credential and Device Security Posture Throughout the Lifecycle 8
3 Functional Demonstration Results 9
3.1 Build 1 Demonstration Results 9
3.2 Build 2 Demonstration Results 16
3.3 Build 3 Demonstration Results 22
3.4 Build 4 Demonstration Results 28
3.5 Build 5 Demonstration Results 34
Appendix A References 42

List of Tables

Table 2-1 Scenario 0 Factory Provisioning Capabilities That May Be Demonstrated 4
Table 2-2 Scenario 1 Trusted Network-Layer Onboarding Capabilities That May Be Demonstrated 5
Table 2-3 Scenario 2 Trusted Application-Layer Onboarding Capabilities That May Be Demonstrated .. 6
Table 2-4 Scenario 3 Re-Onboarding Capabilities That May Be Demonstrated 6
Table 2-5 Scenario 4 Ongoing Device Validation Capabilities That May Be Demonstrated 7
Table 2-6 Scenario 5 Credential and Device Posture Establishment and Maintenance Capabilities That May Be Demonstrated 8
Table 3-1 Build 1 Capabilities Demonstrated 9
Table 3-2 Build 2 Capabilities Demonstrated 16
Table 3-3 Build 3 Capabilities Demonstrated 22
Table 3-4 Build 4 Capabilities Demonstrated 28

Table 3-5 Build 5 Capabilities Demonstrated 35

1 Introduction

In this project, the National Cybersecurity Center of Excellence (NCCoE) is applying standards, recommended practices, and commercially available technology to demonstrate various mechanisms for trusted network-layer onboarding of IoT devices and lifecycle management of those devices. We show how to provision network credentials to IoT devices in a trusted manner and maintain a secure posture throughout the device lifecycle.

This volume of the NIST Cybersecurity Practice Guide describes functional demonstration scenarios that are designed to showcase the security capabilities and characteristics supported by trusted IoT device network-layer onboarding and lifecycle management solutions. Section 2, Functional Demonstration Playbook, defines the scenarios and lists the capabilities that can be showcased in each one. Section 3, Functional Demonstration Results, reports which capabilities have been demonstrated by each of the project's implemented solutions.

1.1 How to Use This Guide

This NIST Cybersecurity Practice Guide demonstrates a standards-based reference design for implementing trusted IoT device network-layer onboarding and lifecycle management and describes various example implementations of this reference design. Each of these implementations, which are known as *builds,* is standards-based and is designed to help provide assurance that networks are not put at risk as new IoT devices are added to them, and also to help safeguard IoT devices from being taken over by unauthorized networks. The reference design described in this practice guide is modular and can be deployed in whole or in part, enabling organizations to incorporate trusted IoT device network-layer onboarding and lifecycle management into their legacy environments according to goals that they have prioritized based on risk, cost, and resources.

NIST is adopting an agile process to publish this content. Each volume is being made available as soon as possible rather than delaying release until all volumes are completed.

This guide contains five volumes:

- NIST SP 1800-36A: *Executive Summary* – why we wrote this guide, the challenge we address, why it could be important to your organization, and our approach to solving this challenge
- NIST SP 1800-36B*: Approach, Architecture, and Security Characteristics* – what we built and why
- NIST SP 1800-36C: *How-To Guides* – instructions for building the example implementations, including all the security-relevant details that would allow you to replicate all or parts of this project
- NIST SP 1800-36D: *Functional Demonstrations* – use cases that have been defined to showcase trusted IoT device network-layer onboarding and lifecycle management security capabilities, and the results of demonstrating these use cases with each of the example implementations **(you are here)**
- NIST SP 1800-36E*: Risk and Compliance Management* – risk analysis and mapping of trusted IoT device network-layer onboarding and lifecycle management security characteristics to cybersecurity standards and recommended practices

Depending on your role in your organization, you might use this guide in different ways:

Business decision makers, including chief security and technology officers, will be interested in the *Executive Summary, NIST SP 1800-36A*, which describes the following topics:

- challenges that enterprises face in migrating to the use of trusted IoT device network-layer onboarding
- example solutions built at the NCCoE
- benefits of adopting the example solution

Technology or security program managers who are concerned with how to identify, understand, assess, and mitigate risk will be interested in *NIST SP 1800-36B*, which describes what we did and why.

Also, Section 4 of *NIST SP 1800-36E* will be of particular interest. Section 4, *Mappings*, maps logical components of the general trusted IoT device network-layer onboarding and lifecycle management reference design to security characteristics listed in various cybersecurity standards and recommended practices documents, including *Framework for Improving Critical Infrastructure Cybersecurity* (NIST Cybersecurity Framework) and *Security and Privacy Controls for Information Systems and Organizations* (NIST SP 800-53).

You might share the *Executive Summary, NIST SP 1800-36A*, with your leadership team members to help them understand the importance of using standards-based trusted IoT device network-layer onboarding and lifecycle management implementations.

IT professionals who want to implement similar solutions will find the whole practice guide useful. You can use the how-to portion of the guide, *NIST SP 1800-36C*, to replicate all or parts of the builds created in our lab. The how-to portion of the guide provides specific product installation, configuration, and integration instructions for implementing the example solution. We do not re-create the product manufacturers' documentation, which is generally widely available. Rather, we show how we incorporated the products together in our environment to create an example solution. Also, you can use *Functional Demonstrations, NIST SP 1800-36D*, which provides the use cases that have been defined to showcase trusted IoT device network-layer onboarding and lifecycle management security capabilities and the results of demonstrating these use cases with each of the example implementations. Finally, *NIST SP 1800-36E* will be helpful in explaining the security functionality that the components of each build provide.

This guide assumes that IT professionals have experience implementing security products within the enterprise. While we have used a suite of commercial products to address this challenge, this guide does not endorse these particular products. Your organization can adopt this solution or one that adheres to these guidelines in whole, or you can use this guide as a starting point for tailoring and implementing parts of a trusted IoT device network-layer onboarding and lifecycle management solution. Your organization's security experts should identify the products that will best integrate with your existing tools and IT system infrastructure. We hope that you will seek products that are congruent with applicable standards and recommended practices.

A NIST Cybersecurity Practice Guide does not describe "the" solution, but example solutions. We seek feedback on the publication's contents and welcome your input. Comments, suggestions, and success stories will improve subsequent versions of this guide. Please contribute your thoughts to iot-onboarding@nist.gov.

2 Functional Demonstration Playbook

Six scenarios have been defined that demonstrate capabilities related to various aspects of trusted IoT device network-layer onboarding, application-layer onboarding, and device lifecycle management. These scenarios are as follows:

- Scenario 0: Factory Provisioning
- Scenario 1: Trusted Network-Layer Onboarding
- Scenario 2: Trusted Application-Layer Onboarding
- Scenario 3: Re-Onboarding a Device
- Scenario 4: Ongoing Device Validation
- Scenario 5: Establishment and Maintenance of Credential and Device Security Posture Throughout the Lifecycle

We executed the factory provisioning scenario (Scenario 0) using both a Bootstrapping Remote Secure Key Infrastructure (BRSKI) Factory Provisioning Build and a Wi-Fi Easy Connect Factory Provisioning Build that have been implemented as part of this project. We executed the trusted network-layer onboarding and lifecycle management scenarios using each of the onboarding builds that have been implemented as part of this project. The capabilities that were demonstrated depend both on the features of the network-layer onboarding protocol (i.e., Wi-Fi Easy Connect) that the build supports and on any additional mechanisms the build may have integrated (e.g., application-layer onboarding).

Section 2.1 defines the factory provisioning scenario (Scenario 0). Sections 2.2 through Section 2.6 define each of the five onboarding scenarios.

2.1 Scenario 0: Factory Provisioning

This scenario, which simulates the IoT device factory provisioning process, is designed to represent some steps that must be performed in the factory before the device is put into the supply chain. These steps are performed by the device manufacturer or integrator to provision a device with the information it requires to be able to participate in trusted network-layer onboarding and lifecycle management. The device is assumed to have been equipped with secure storage and with the software or firmware needed to support a specific network-layer onboarding protocol (e.g., Wi-Fi Easy Connect or BRSKI). Scenario 0 includes initial provisioning of the IoT device with its birth credential (e.g., its private key and initial device identifier (IDevID) [1]), where it is stored in secure storage to prevent tampering or disclosure. This process includes generation of the credential (e.g., a private key and other information), signing of this credential (if applicable, depending on what onboarding protocol the device is designed to support), and transfer of the device bootstrapping information (e.g., a DPP URI or the device's IDevID) to the appropriate destination to ensure that it will be available for use during the network layer onboarding process. Following provisioning, the birth credential may be used for network-layer or application-layer onboarding. Table 2-1 lists the capabilities that may be demonstrated in this factory provisioning scenario.

Table 2-1 Scenario 0 Factory Provisioning Capabilities That May Be Demonstrated

Demo ID	Capability	Description
S0.C1	Birth Credential Generation and Storage	The device's birth credentials are generated within or generated and provisioned into secure storage on the IoT device. The content and format of the credential are appropriate to the onboarding protocol (e.g., Wi-Fi Easy Connect [2] or BRSKI [3]) that the device is designed to support: • For BRSKI, the credential is a private key, a signed certificate (IDevID), a trust anchor for the manufacturer's certificate authority (CA), and the location of a trusted manufacturer authorized signing authority (MASA). • For Wi-Fi Easy Connect, the credential is a private key and a public bootstrapping key.
S0.C2	Birth Credential Signing	The credential is signed by a trusted CA.
S0.C3	Bootstrapping Information Availability	The bootstrapping information required for onboarding the device is made available as needed. The format and content of the bootstrapping information depends on the onboarding protocol that the device is designed to support: • For BRSKI, the bootstrapping information is the certificate and ownership information that is sent to the MASA. • For Wi-Fi Easy Connect, the bootstrapping information is the Device Provisioning Protocol (DPP) uniform resource identifier (URI) (which contains the public key, and optionally other information such as device serial number).

2.2 Scenario 1: Trusted Network-Layer Onboarding

This scenario involves trusted network-layer onboarding of an authorized IoT device to a local network that is operated by the owner of the IoT device. The device is assumed to have been manufactured to support the type of network-layer onboarding protocol (e.g., Wi-Fi Easy Connect or BRSKI) that is being used by the local network. The device is also assumed to have been provisioned with its birth credential in a manner similar to that described in Scenario 0: Factory Provisioning, including transfer of the device's bootstrapping information (e.g., its public key) to the operator of the local network to ensure that this information will be available to support authentication of the device during the initial phase of the trusted network-layer onboarding process. Onboarding is performed after the device has booted up and is placed in onboarding mode. Because the organization that is operating the local network is the owner of the IoT device, the device is authorized to onboard to the network and the network is authorized to onboard the device. In this scenario, after the identities of the device and the network are authenticated, a *network onboarding component*—a logical component authorized to onboard devices on behalf of the network—authenticates the device and provisions unique network credentials to the device over a secure channel. These network credentials are not just specific to the device; they are also specific to the local network. The device then uses these credentials to connect to the network. Table 2-2 lists the capabilities that may be demonstrated in this scenario.

Table 2-2 Scenario 1 Trusted Network-Layer Onboarding Capabilities That May Be Demonstrated

Demo ID	Capability	Description
S1.C1	Device Authentication	The onboarding mechanism authenticates the device's identity.
S1.C2	Device Authorization	The onboarding mechanism verifies that the device is authorized to onboard to the network.
S1.C3	Network Authentication	The device can verify the network's identity.
S1.C4	Network Authorization	The device can verify that the network is authorized to take control of it.
S1.C5	Secure Local Credentialing	The onboarding mechanism securely provisions local network credentials to the device.
S1.C6	Secure Storage	The local network credentials are provisioned to secure hardware-backed storage on the device.
S1.C7	Network Selection	The onboarding mechanism provides the IoT device with the identifier of the network to which the device should onboard.
S1.C8	Interoperability	The network-layer onboarding mechanism can onboard a minimum of two types of IoT devices (e.g., different device vendors and models).

2.3 Scenario 2: Trusted Application-Layer Onboarding

This scenario involves trusted application-layer onboarding that is performed automatically on an IoT device after the device connects to a network. As a result, this scenario can be thought of as a series of steps that would be performed as an extension of Scenario 1, assuming the device has been designed and provisioned to support application-layer onboarding. As part of these steps, the device automatically mutually authenticates with a trusted application-layer onboarding service and establishes an encrypted connection to that service so the service can provision the device with application-layer credentials. The application-layer credentials could, for example, enable the device to securely connect to a trusted lifecycle management service to check for available updates or patches. For the application-layer onboarding mechanism to be trusted, it must establish an encrypted connection to the device without exposing any information that must be protected to ensure the confidentiality of that connection. Two types of application-layer onboarding are defined in NIST SP 1800-36B: *streamlined* and *independent*. Table 2-3 lists the capabilities that may be demonstrated in this scenario, including both types of application-layer onboarding.

Table 2-3 Scenario 2 Trusted Application-Layer Onboarding Capabilities That May Be Demonstrated

Demo ID	Capability	Description
S2.C1	Automatic Initiation of Streamlined Application-Layer Onboarding	The device can automatically (i.e., with no manual intervention required) initiate trusted application-layer onboarding after performing network-layer onboarding and connecting to the network. In this case, the application-layer onboarding bootstrapping information has been securely conveyed to the device during the network-layer onboarding process.
S2.C2	Automatic Initiation of Independent Application-Layer Onboarding	The device can automatically (i.e., with no manual intervention required) initiate trusted application-layer onboarding after performing network-layer onboarding and connecting to the network. In this case, the application-layer onboarding bootstrapping information has been pre-provisioned to the device by the device manufacturer or integrator (e.g., as part of an application that was installed on the device during the manufacturing process).
S2.C3	Trusted Application-Layer Onboarding	The device and a trusted application service can establish an encrypted connection without exposing any information that must be protected to ensure the confidentiality of the connection. They can then use that secure association to exchange application-layer information.

2.4 Scenario 3: Re-Onboarding a Device

This scenario involves re-onboarding an IoT device to a network after deleting its network credentials so that the device can be re-credentialed and reconnected. If the device also supports application-layer onboarding, application-layer onboarding should also be performed again after the device reconnects to the network. This scenario assumes that the device has been able to successfully demonstrate trusted network-layer onboarding as defined in Scenario 1: Trusted Network-Layer Onboarding. If application-layer re-onboarding is to be demonstrated as well, the scenario assumes that the device has also been able to successfully demonstrate at least one method of application-layer onboarding as defined in Scenario 2: Trusted Application-Layer Onboarding. Table 2-4 lists the capabilities that may be demonstrated in this scenario.

Table 2-4 Scenario 3 Re-Onboarding Capabilities That May Be Demonstrated

Demo ID	Capability	Description
S3.C1	Credential Deletion	The device's network credential can be deleted.
S3.C2	De-Credentialed Device Cannot Connect	After the device's network credential has been deleted, the device is not able to connect to or communicate on the network securely.
S3.C3	Re-Onboarding (network layer)	After the device's network credential has been deleted, the network-layer onboarding mechanism can securely re-provision a network

Demo ID	Capability	Description
		credential to the device, which the device can then use to connect to the network securely.
S3.C4	Re-Onboarding (application layer)	After the device's network and application-layer credentials have been deleted and the device has been re-onboarded at the network layer and reconnected to the network, the device can again perform trusted application-layer onboarding.

2.5 Scenario 4: Ongoing Device Validation

This scenario involves ongoing validation of a device, not only as part of a trusted boot or attestation process prior to permitting the device to undergo network-layer onboarding, but also after the device has connected to the network. It may involve one or more security mechanisms that are designed to evaluate, validate, or respond to device trustworthiness using methods such as examining device behavior, ensuring device authenticity and integrity, and assigning the device to a specific network segment based on its conformance to policy criteria. Table 2-5 lists the capabilities that may be demonstrated in this scenario. None of these capabilities are integral to trusted network-layer onboarding; however, they may be used in conjunction with, or subsequent to, trusted network-layer onboarding to enhance device and network security.

Table 2-5 Scenario 4 Ongoing Device Validation Capabilities That May Be Demonstrated

Demo ID	Capability	Description
S4.C1	Device Attestation (initial)	The network-layer onboarding mechanism requires successful device attestation prior to permitting the device to be onboarded.
S4.C2	Device Attestation (application layer)	The application-layer onboarding mechanism requires successful device attestation prior to permitting the device to be onboarded.
S4.C3	Device Attestation (ongoing)	Successful device attestation is required prior to permitting the device to perform some operation (e.g., accessing a high-value resource).
S4.C4	Local Network Segmentation (initial)	Upon connection, the IoT device is assigned to some local network segment in accordance with policy, which may include an assessment of its security posture.
S4.C5	Behavioral Analysis	Device behavior is observed to determine whether the device meets the policy criteria required to be permitted to perform a given operation (e.g., to access a high-value resource or be placed on a given network segment).
S4.C6	Local Network Segmentation (ongoing)	The IoT device can be reassigned to a different network segment based on ongoing assessments of its conformance to policy criteria.
S4.C7	Periodic Device Reauthentication	After connection, the IoT device's identity is periodically reauthenticated in order to maintain network access.

Demo ID	Capability	Description
S4.C8	Periodic Device Reauthorization	After connection, the IoT device's authorization to access the network is periodically reconfirmed in order to maintain network access.

2.6 Scenario 5: Establishment and Maintenance of Credential and Device Security Posture Throughout the Lifecycle

This scenario involves steps used to help establish and maintain the security posture of both the device's network credentials and the device itself. It includes the capability to download and validate the device's most recent firmware updates, securely integrate with a device communications intent enforcement mechanism (e.g., Manufacturer Usage Description (MUD) [4]), keep the device updated and patched, and establish and maintain the device's network credentials by provisioning X.509 certificates or DPP Connectors to the device and updating expired network credentials. Table 2-6 lists the capabilities that may be demonstrated in this scenario. None of these capabilities are integral to trusted network-layer onboarding; however, they may be used in conjunction with or subsequent to trusted network-layer onboarding to enhance device and network security.

Table 2-6 Scenario 5 Credential and Device Posture Establishment and Maintenance Capabilities That May Be Demonstrated

Demo ID	Capability	Description
S5.C1	Trusted Firmware Updates	The device can download the most recent firmware update and verify its signature before it is installed.
S5.C2	Credential Certificate Provisioning	The onboarding mechanism can interact with a certificate authority to sign a device's X.509 certificate and provision it onto the device.
S5.C3	Credential Update	The device's network credential can be updated after it expires.
S5.C4	Server Attestation	Successful server attestation is required prior to permitting the server to perform some operation on the device (e.g., prior to downloading and installing updates onto the device).
S5.C5	Secure Integration with MUD	The network-layer onboarding mechanism can convey necessary device communications intent information (e.g., the IoT device's MUD URL) to the network in encrypted form, thereby securely binding this information to the device and ensuring its confidentiality and integrity.
S5.C6	Lifecycle Management Establishment	The device has a lifecycle management service and can automatically establish a secure association with it after performing network-layer onboarding and connecting to the network.

3 Functional Demonstration Results

This section records the capabilities that were demonstrated for each of the builds.

3.1 Build 1 Demonstration Results

Table 3-1 lists the capabilities that were demonstrated by Build 1.

Table 3-1 Build 1 Capabilities Demonstrated

Demo ID	Capability	Description	Demonstrated?	Explanation/Notes
Scenario 0: Factory Provisioning				
S0.C1	Birth Credential Generation and Storage	The device's birth credentials are generated within or generated and provisioned into secure storage on the IoT device. For Wi-Fi Easy Connect, the credential is a private key and a public bootstrapping key.	Yes	Public/private key-pair is generated within the SEALSQ VaultIC secure element.
S0.C2	Birth Credential Signing	The credential is signed by a trusted CA.	No	There is no requirement to support this capability in this build. Birth credentials for devices supporting Wi-Fi Easy Connect onboarding do not need to be signed.
S0.C3	Bootstrapping Information Availability	The bootstrapping information required for onboarding the device is made available as needed. For Wi-Fi Easy Connect, the bootstrapping information is the Device Provisioning Protocol (DPP) uniform resource identifier (URI) (which contains the public key, and optionally other information such as device serial number).	Yes	The device's DPP URI is generated using the public/private keypair that was generated in the device's secure element. This DPP URI is encoded in a QR code that is written to a Portable Network Graphics (PNG) file and may be transferred from a vendor cloud upon acquisition of the device.

Demo ID	Capability	Description	Demonstrated?	Explanation/Notes
Scenario 1: Trusted Network-Layer Onboarding				
S1.C1	Device Authentication	The onboarding mechanism authenticates the device's identity.	Yes	DPP performs device authentication.
S1.C2	Device Authorization	The onboarding mechanism verifies that the device is authorized to onboard to the network.	Yes	When the device's URI is found on the HPE cloud service, this verifies that the device is authorized to onboard to the network.
S1.C3	Network Authentication	The device can verify the network's identity.	No	This could be supported by providing the IoT device with the DPP URI of the network, but the Aruba User Experience Insight (UXI) sensor used in this build lacks the user interface needed to do so.
S1.C4	Network Authorization	The device can verify that the network is authorized to take control of it.	Yes	The network that possesses the device's public key is implicitly authorized to onboard the device by virtue of its knowledge of the device's public key. While this is not cryptographic, it does provide a certain level of assurance that the "wrong" network doesn't take control of the device.
S1.C5	Secure Local Credentialing	The onboarding mechanism securely provisions local network credentials to the device.	Yes	DPP provisions the device's network credentials over an encrypted channel.
S1.C6	Secure Storage	The local network credentials are provisioned to secure hardware-backed storage on the device.	No	The bootstrapping credentials are stored in a Trusted Platform Module (TPM) 2.0 hardware enclave, but the local network credentials are not
S1.C7	Network Selection	The onboarding mechanism provides	Yes	The network responds to device chirps.

Demo ID	Capability	Description	Demonstrated?	Explanation/Notes
		the IoT device with the identifier of the network to which the device should onboard.		
S1.C8	Interoperability	The network-layer onboarding mechanism can onboard a minimum of two types of IoT devices (e.g., different device vendors and models).	Yes	IoT devices from Build 2 were successfully onboarded in Build 1.
Scenario 2: Trusted Application-Layer Onboarding				
S2.C1	Automatic Initiation of Streamlined Application-Layer Onboarding	The device can automatically (i.e., with no manual intervention required) initiate trusted application-layer onboarding after performing network-layer onboarding and connecting to the network. In this case, the application-layer onboarding bootstrapping information has been securely conveyed to the device during the network-layer onboarding process.	No	Not supported in this build.
S2.C2	Automatic Initiation of Independent Application-Layer Onboarding	The device can automatically (i.e., with no manual intervention required) initiate trusted application-layer onboarding after performing network-layer onboarding and connecting to the network. In this case, the application-layer onboarding bootstrapping information has been	Yes	Once onboarded, the UXI sensor automatically initiates application-layer onboarding to the UXI application.

Demo ID	Capability	Description	Demonstrated?	Explanation/Notes
		pre-provisioned to the device by the device manufacturer or integrator (e.g., as part of an application that was installed on the device during the manufacturing process).		
S2.C3	Trusted Application- Layer Onboarding	The device and a trusted application service can establish an encrypted connection without exposing any information that must be protected to ensure the confidentiality of the connection. They can then use that secure association to exchange application-layer information.	Yes	Once onboarded, the UXI sensor establishes a secure connection with the UXI cloud, which provisions the sensor with its credentials for the UXI application. Later, the sensor uploads data to the UXI application securely.
Scenario 3: Re-Onboarding a Device				
S3.C1	Credential Deletion	The device's network credential can be deleted.	Yes	Factory reset and manual credential removal were leveraged.
S3.C2	De-Credentialed Device Cannot Connect	After the device's network credential has been deleted, the device is not able to connect to or communicate on the network securely.	Yes	Observed.
S3.C3	Re-Onboarding (network layer)	After the device's network credential has been deleted, the network-layer onboarding mechanism can security re-provision a network credential to the device, which the device can then use to	Yes	Observed.

Demo ID	Capability	Description	Demonstrated?	Explanation/Notes
		connect to the network securely.		
S3.C4	Re-Onboarding (application layer)	After the device's network and application-layer credentials have been deleted and the device has been re-onboarded at the network layer and re-connected to the network, the device can again perform trusted application-layer onboarding.	Yes	Observed.
Scenario 4: Ongoing Device Validation				
S4.C1	Device Attestation (initial)	The network-layer onboarding mechanism requires successful device attestation prior to permitting the device to be onboarded.	No	Not supported in this build.
S4.C2	Device Attestation (application layer)	The application-layer onboarding mechanism requires successful device attestation prior to permitting the device to be onboarded.	No	Not supported in this build.
S4.C3	Device Attestation (ongoing)	Successful device attestation is required prior to permitting the device to perform some operation (e.g., accessing a high-value resource).	No	Not supported in this build.
S4.C4	Local Network Segmentation (initial)	Upon connection, the IoT device is assigned to some local network segment in accordance with policy, which may	No	Not demonstrated in this phase.

Demo ID	Capability	Description	Demonstrated?	Explanation/Notes
		include an assessment of its security posture.		
S4.C5	Behavioral Analysis	Device behavior is observed to determine whether the device meets the policy criteria required to be permitted to perform a given operation (e.g., to access a high-value resource or be placed on a given network segment).	No	Not supported in this build.
S4.C6	Local Network Segmentation (ongoing)	The IoT device can be reassigned to a different network segment based on ongoing assessments of its conformance to policy criteria.	No	Not supported in this build.
S4.C7	Periodic Device Reauthentication	After connection, the IoT device's identity is periodically reauthenticated in order to maintain network access.	No	Not supported in this build.
S4.C8	Periodic Device Reauthorization	After connection, the IoT device's authorization to access the network is periodically reconfirmed in order to maintain network access.	No	Not supported in this build.
Scenario 5: Establishment and Maintenance of Credential and Device Security Posture Throughout the Lifecycle				
S5.C1	Trusted Firmware Updates	The device can download the most recent firmware update and verify its signature before it is installed.	No	Not supported in this build.

Demo ID	Capability	Description	Demonstrated?	Explanation/Notes
S5.C2	Credential Certificate Provisioning	The onboarding mechanism can interact with a certificate authority to sign a device's X.509 certificate and provision it onto the device.	Yes	This capability has been successfully demonstrated with the SEALSQ INeS CA.
S5.C3	Credential Update	The device's network credential can be updated after it expires.	No	Not demonstrated in this phase.
S5.C4	Server Attestation	Successful server attestation is required prior to permitting the server to perform some operation on the device (e.g., prior to downloading and installing updates onto the device).	No	Not supported in this build.
S5.C5	Secure Integration with MUD	The network-layer onboarding mechanism can convey necessary device communications intent information (e.g., the IoT device's MUD URL) to the network in encrypted form, thereby securely binding this information to the device and ensuring its confidentiality and integrity.	No	Supported by DPP, but not demonstrated because Build 1 is not integrated with MUD or any other device communications intent enforcement mechanism.
S5.C6	Lifecycle Management Establishment	The device has a lifecycle management service and can automatically establish a secure association with it after performing network-layer onboarding and	No	Not supported in this build.

Demo ID	Capability	Description	Demonstrated?	Explanation/Notes
		connecting to the network.		

3.2 Build 2 Demonstration Results

Table 3-2 lists the capabilities that were demonstrated by Build 2.

Table 3-2 Build 2 Capabilities Demonstrated

Demo ID	Capability	Description	Demonstrated?	Explanation/Notes
Scenario 1: Trusted Network-Layer Onboarding				
S1.C1	Device Authentication	The onboarding mechanism authenticates the device's identity.	Yes	DPP performs device authentication.
S1.C2	Device Authorization	The onboarding mechanism verifies that the device is authorized to onboard to the network.	Yes	Only devices that have been added/approved by the administrator are onboarded. When the device's URI is found, the controller authorizes the device to join the network.
S1.C3	Network Authentication	The device can verify the network's identity.	No	This could be supported by providing the IoT device with the DPP URI of the network, but this is not currently implemented.
S1.C4	Network Authorization	The device can verify that the network is authorized to take control of it.	Yes	The network that possesses the device's public key is implicitly authorized to onboard the device by virtue of its knowledge of the device's public key. While this is not cryptographic, it does provide a certain level of assurance that the "wrong" network doesn't take control of the device.
S1.C5	Secure Local Credentialing	The onboarding mechanism securely provisions local	Yes	DPP provisions the device's network credentials over an encrypted channel.

Demo ID	Capability	Description	Demonstrated?	Explanation/Notes
		network credentials to the device.		
S1.C6	Secure Storage	The local network credentials are provisioned to secure hardware-backed storage on the device.	No	The IoT device does not have secure hardware-backed storage.
S1.C7	Network Selection	The onboarding mechanism provides the IoT device with the identifier of the network to which the device should onboard.	Yes	Network responds to device chirps.
S1.C8	Interoperability	The network-layer onboarding mechanism can onboard a minimum of two types of IoT devices (e.g., different device vendors and models).	Yes	Build 2 was able to onboard the IoT devices from Build 1.
Scenario 2: Trusted Application-Layer Onboarding				
S2.C1	Automatic Initiation of Streamlined Application-Layer Onboarding	The device can automatically (i.e., with no manual intervention required) initiate trusted application-layer onboarding after performing network-layer onboarding and connecting to the network. In this case, the application-layer onboarding bootstrapping information has been securely conveyed to the device during the network-layer onboarding process.	Yes	This has been demonstrated with the OCF Iotivity [5] custom extension. Iotivity is an open-source software framework that implements OCF standards and enables seamless device-to-device connectivity.
S2.C2	Automatic Initiation of Independent	The device can automatically (i.e., with no manual intervention required) initiate	No	Not supported in this build.

Demo ID	Capability	Description	Demonstrated?	Explanation/Notes
	Application-Layer Onboarding	trusted application-layer onboarding after performing network-layer onboarding and connecting to the network. In this case, the application-layer onboarding bootstrapping information has been pre-provisioned to the device by the device manufacturer or integrator (e.g., as part of an application that was installed on the device during the manufacturing process).		
S2.C3	Trusted Application-Layer Onboarding	The device and a trusted application service can establish an encrypted connection without exposing any information that must be protected to ensure the confidentiality of the connection. They can then use that secure association to exchange application-layer information.	Yes	Once the device is onboarded to the network using DPP, the credentials for the application layer onboarding are sent over the secure channel and provisioned by the onboarding tool (OBT).
Scenario 3: Re-Onboarding a Device				
S3.C1	Credential Deletion	The device's network credential can be deleted.	Yes	Supports factory reset.
S3.C2	De-Credentialed Device Cannot Connect	After the device's network credential has been deleted, the device is not able to connect to or communicate on the network securely.	Yes	Observed.

Demo ID	Capability	Description	Demonstrated?	Explanation/Notes
S3.C3	Re-Onboarding (network layer)	After the device's network credential has been deleted, the network-layer onboarding mechanism can security re-provision a network credential to the device, which the device can then use to connect to the network securely.	Yes	Observed.
S3.C4	Re-Onboarding (application layer)	After the device's network and application-layer credentials have been deleted and the device has been re-onboarded at the network layer and re-connected to the network, the device can again perform trusted application-layer onboarding.	Yes	Observed.
Scenario 4: Ongoing Device Validation				
S4.C1	Device Attestation (initial)	The network-layer onboarding mechanism requires successful device attestation prior to permitting the device to be onboarded.	No	Not supported in this build.
S4.C2	Device Attestation (application layer)	The application-layer onboarding mechanism requires successful device attestation prior to permitting the device to be onboarded.	No	Not supported in this build.
S4.C3	Device Attestation (ongoing)	Successful device attestation is required prior to permitting the device to perform some operation (e.g.,	No	Not supported in this build.

Demo ID	Capability	Description	Demonstrated?	Explanation/Notes
		accessing a high-value resource).		
S4.C4	Local Network Segmentation (initial)	Upon connection, the IoT device is assigned to some local network segment in accordance with policy, which may include an assessment of its security posture.	Yes	When the device is connected to the network, the gateway places it in a restricted network segment based on policy.
S4.C5	Behavioral Analysis	Device behavior is observed to determine whether the device meets the policy criteria required to be permitted to perform a given operation (e.g., to access a high-value resource or be placed on a given network segment).	No	Not supported in this build.
S4.C6	Local Network Segmentation (ongoing)	The IoT device can be reassigned to a different network segment based on ongoing assessments of its conformance to policy criteria.	Yes	Device can be moved to new network segments programmatically. The policy to do this is not defined in this build.
S4.C7	Periodic Device Reauthentication	After connection, the IoT device's identity is periodically reauthenticated in order to maintain network access.	No	Not supported in this build.
S4.C8	Periodic Device Reauthorization	After connection, the IoT device's authorization to access the network is periodically reconfirmed in order to maintain network access.	No	Not supported in this build.
Scenario 5: Establishment and Maintenance of Credential and Device Security Posture Throughout the Lifecycle				

Demo ID	Capability	Description	Demonstrated?	Explanation/Notes
S5.C1	Trusted Firmware Updates	The device can download the most recent firmware update and verify its signature before it is installed.	No	Not supported in this build.
S5.C2	Credential Certificate Provisioning	The onboarding mechanism can interact with a certificate authority to sign a device's X.509 certificate and provision it onto the device.	No	Not supported in this build.
S5.C3	Credential Update	The device's network credential can be updated after it expires.	No	Not demonstrated in this phase.
S5.C4	Server Attestation	Successful server attestation is required prior to permitting the server to perform some operation on the device (e.g., prior to downloading and installing updates onto the device).	No	Not supported in this build.
S5.C5	Secure Integration with MUD	The network-layer onboarding mechanism can convey necessary device communications intent information (e.g., the IoT device's MUD URL) to the network in encrypted form, thereby securely binding this information to the device and ensuring its confidentiality and integrity.	No	Supported by DPP, but not demonstrated because Build 2 is not integrated with MUD or any other device communications intent enforcement mechanism.

Demo ID	Capability	Description	Demonstrated?	Explanation/Notes
S5.C6	Lifecycle Management Establishment	The device has a lifecycle management service and can automatically establish a secure association with it after performing network-layer onboarding and connecting to the network.	No	Not supported in this build.

312 ## 3.3 Build 3 Demonstration Results

313 Table 3-3 lists the capabilities that were demonstrated by Build 3.

314 **Table 3-3 Build 3 Capabilities Demonstrated**

Demo ID	Capability	Description	Demonstrated?	Explanation/Notes
Scenario 1: Trusted Network-Layer Onboarding				
S1.C1	Device Authentication	The onboarding mechanism authenticates the device's identity.	Yes	The local domain registrar receives the voucher request.
S1.C2	Device Authorization	The onboarding mechanism verifies that the device is authorized to onboard to the network.	Yes	The registrar verifies that the device is from an authorized manufacturer.
S1.C3	Network Authentication	The device can verify the network's identity.	Yes	Demonstrated by the voucher.
S1.C4	Network Authorization	The device can verify that the network is authorized to take control of it.	Yes	The registrar examines the new voucher and passes it to the device for onboarding.
S1.C5	Secure Local Credentialing	The onboarding mechanism securely provisions local network credentials to the device.	Yes	A local device identifier (LDevID) (i.e., the device's network credential) [1] is provisioned to the device after the device authentication and authorization process.

Demo ID	Capability	Description	Demonstrated?	Explanation/Notes
S1.C6	Secure Storage	The local network credentials are provisioned to secure hardware-backed storage on the device.	No	Not demonstrated in this phase.
S1.C7	Network Selection	The onboarding mechanism provides the IoT device with the identifier of the network to which the device should onboard.	No	Not demonstrated in this build.
S1.C8	Interoperability	The network-layer onboarding mechanism can onboard a minimum of two types of IoT devices (e.g., different device vendors and models).	No	Supported by BRSKI, but not demonstrated in this build.
Scenario 2: Trusted Application-Layer Onboarding				
S2.C1	Automatic Initiation of Streamlined Application-Layer Onboarding	The device can automatically (i.e., with no manual intervention required) initiate trusted application-layer onboarding after performing network-layer onboarding and connecting to the network. In this case, the application-layer onboarding bootstrapping information has been securely conveyed to the device during the network-layer onboarding process.	No	Not supported in this build.
S2.C2	Automatic Initiation of Independent Application-Layer Onboarding	The device can automatically (i.e., with no manual intervention required) initiate trusted application-layer onboarding after	No	Not supported in this build.

Demo ID	Capability	Description	Demonstrated?	Explanation/Notes
		performing network-layer onboarding and connecting to the network. In this case, the application-layer onboarding bootstrapping information has been pre-provisioned to the device by the device manufacturer or integrator (e.g., as part of an application that was installed on the device during the manufacturing process).		
S2.C3	Trusted Application-Layer Onboarding	The device and a trusted application service can establish an encrypted connection without exposing any information that must be protected to ensure the confidentiality of the connection. They can then use that secure association to exchange application-layer information.	No	Not supported in this build.
Scenario 3: Re-Onboarding a Device				
S3.C1	Credential Deletion	The device's network credential can be deleted.	Yes	Observed.
S3.C2	De-Credentialed Device Cannot Connect	After the device's network credential has been deleted, the device is not able to connect to or communicate on the network securely.	Yes	Observed.
S3.C3	Re-Onboarding (network-layer)	After the device's network credential has been deleted, the	Yes	Observed.

Demo ID	Capability	Description	Demonstrated?	Explanation/Notes
		network-layer onboarding mechanism can security re-provision a network credential to the device, which the device can then use to connect to the network securely.		
S3.C4	Re-Onboarding (application layer)	After the device's network credentials have been deleted and the device has been re-onboarded at the network layer and re-connected to the network, the device can perform application-layer onboarding automatically.	No	Not supported in this build.
Scenario 4: Ongoing Device Validation				
S4.C1	Device Attestation (initial)	The network-layer onboarding mechanism requires successful device attestation prior to permitting the device to be onboarded.	No	Not supported in this build.
S4.C2	Device Attestation (application layer)	The application-layer onboarding mechanism requires successful device attestation prior to permitting the device to be onboarded.	No	Not supported in this build.
S4.C3	Device Attestation (ongoing)	Successful device attestation is required prior to permitting the device to perform some operation (e.g., accessing a high-value resource).	No	Not supported in this build.

Demo ID	Capability	Description	Demonstrated?	Explanation/Notes
S4.C4	Local Network Segmentation (initial)	Upon connection, the IoT device is assigned to some local network segment in accordance with policy, which may include an assessment of its security posture.	No	Not supported in this build.
S4.C5	Behavioral Analysis	Device behavior is observed to determine whether the device meets the policy criteria required to be permitted to perform a given operation (e.g., to access a high-value resource or be placed on a given network segment).	No	Not supported in this build.
S4.C6	Local Network Segmentation (ongoing)	The IoT device can be reassigned to a different network segment based on ongoing assessments of its conformance to policy criteria.	No	Not supported in this build.
S4.C7	Periodic Device Reauthentication	After connection, the IoT device's identity is periodically reauthenticated in order to maintain network access.	No	Not supported in this build.
S4.C8	Periodic Device Reauthorization	After connection, the IoT device's authorization to access the network is periodically reconfirmed in order to maintain network access.	No	Not supported in this build.
Scenario 5: Establish and Maintain Credential and Device Security Posture Throughout the Lifecycle				
S5.C1	Trusted Firmware Updates	The device can download the most recent firmware update	No	Not supported in this build.

Demo ID	Capability	Description	Demonstrated?	Explanation/Notes
		and verify its signature before it is installed.		
S5.C2	Credential Certificate Provisioning	The onboarding mechanism can interact with a certificate authority to sign a device's X.509 certificate and provision it onto the device.	Yes	A vendor-installed X.509 certificate and a vendor's authorizing service use link-local connectivity to provision device credentials.
S5.C3	Credential Update	The device's network credential (e.g., its LDevID or X.509 certificate) can be updated after it expires.	No	Will be demonstrated in a future implementation of this build.
S5.C4	Server Attestation	Successful server attestation is required prior to permitting the server to perform some operation on the device (e.g., prior to downloading and installing updates onto the device).	No	Not supported in this build.
S5.C5	Secure Integration with MUD	The network-layer onboarding mechanism can convey necessary device communications intent information (e.g., the IoT device's MUD URL) to the network in encrypted form, thereby securely binding this information to the device and ensuring its confidentiality and integrity.	No	Supported by BRSKI, but not demonstrated because Build 3 is not integrated with MUD or any other device communications intent enforcement mechanism.
S5.C6	Lifecycle Management Establishment	The device has a lifecycle management service and can automatically establish a secure association	No	Not supported in this build.

Demo ID	Capability	Description	Demonstrated?	Explanation/Notes
		with it after performing network-layer onboarding and connecting to the network.		

315 ## 3.4 Build 4 Demonstration Results

316 Table 3-4 lists the capabilities that were demonstrated by Build 4.

317 **Table 3-4 Build 4 Capabilities Demonstrated**

Demo ID	Capability	Description	Demonstrated?	Explanation/Notes
Scenario 1: Trusted Network-Layer Onboarding				
S1.C1	Device Authentication	The onboarding mechanism authenticates the device's identity.	No	The build performs trusted application-layer onboarding only.
S1.C2	Device Authorization	The onboarding mechanism verifies that the device is authorized to onboard to the network.	No	The build performs trusted application-layer onboarding only.
S1.C3	Network Authentication	The device can verify the network's identity.	No	The build performs trusted application-layer onboarding only.
S1.C4	Network Authorization	The device can verify that the network is authorized to take control of it.	No	The build performs trusted application-layer onboarding only.
S1.C5	Secure Local Credentialing	The onboarding mechanism securely provisions local network credentials to the device.	No	The build performs trusted application-layer onboarding only.
S1.C6	Secure Storage	The local network credentials are provisioned to secure hardware-backed storage on the device.	Yes	The local network credentials are stored in the Silicon Labs Secure Vault on the Thunderboard.

Demo ID	Capability	Description	Demonstrated?	Explanation/Notes
S1.C7	Network Selection	The onboarding mechanism provides the IoT device with the identifier of the network to which the device should onboard.	No	The device generates a pre-shared key that is manually entered in the OpenThread Border Router [6].
S1.C8	Interoperability	The network-layer onboarding mechanism can onboard a minimum of two types of IoT devices (e.g., different device vendors and models).	No	Not supported in this build.
Scenario 2: Trusted Application-Layer Onboarding				
S2.C1	Automatic Initiation of Streamlined Application-Layer Onboarding	The device can automatically (i.e., with no manual intervention required) initiate trusted application-layer onboarding after performing network-layer onboarding and connecting to the network. In this case, the application-layer onboarding bootstrapping information has been securely conveyed to the device during the network-layer onboarding process.	No	Not supported in this build.
S2.C2	Automatic Initiation of Independent Application-	The device can automatically (i.e., with no manual intervention required) initiate	Yes	Trusted application-layer onboarding using Kudelski keySTREAM is configured to proceed automatically pending

Demo ID	Capability	Description	Demonstrated?	Explanation/Notes
	Layer Onboarding	trusted application-layer onboarding after performing network-layer onboarding and connecting to the network. In this case, the application-layer onboarding bootstrapping information has been pre-provisioned to the device by the device manufacturer or integrator (e.g., as part of an application that was installed on the device during the manufacturing process).		confirmation from a user (through the press of a button).
S2.C3	Trusted Application-Layer Onboarding	The device and a trusted application service can establish an encrypted connection without exposing any information that must be protected to ensure the confidentiality of the connection. They can then use that secure association to exchange application-layer information.	Yes	Application Layer Onboarding via Kudelski keySTREAM GUI / AWS IoT Core and through the Silicon Labs Simplicity Studio Device Console
Scenario 3: Re-Onboarding a Device				
S3.C1	Credential Deletion	The device's network credential can be deleted.	Yes	The device can be removed from the network via the Open Thread Border Router

Demo ID	Capability	Description	Demonstrated?	Explanation/Notes
				GUI and cannot rejoin without entering a new pre-shared key.
S3.C2	De-Credentialed Device Cannot Connect	After the device's network credential has been deleted, the device is not able to connect to or communicate on the network securely.	Yes	Observed.
S3.C3	Re-Onboarding (network layer)	After the device's network credential has been deleted, the network-layer onboarding mechanism can security re-provision a network credential to the device, which the device can then use to connect to the network securely.	Yes	Observed.
S3.C4	Re-Onboarding (application layer)	After the device's network and application-layer credentials have been deleted and the device has been re-onboarded at the network layer and re-connected to the network, the device can again perform trusted application-layer onboarding.	Yes	Observed.
Scenario 4: Ongoing Device Validation				
S4.C1	Device Attestation (initial)	The network-layer onboarding mechanism requires successful device attestation prior to permitting the	No	Not supported in this build.

Demo ID	Capability	Description	Demonstrated?	Explanation/Notes
		device to be onboarded.		
S4.C2	Device Attestation (application layer)	The application-layer onboarding mechanism requires successful device attestation prior to permitting the device to be onboarded.	No	Not supported in this build.
S4.C3	Device Attestation (ongoing)	Successful device attestation is required prior to permitting the device to perform some operation (e.g., accessing a high-value resource).	No	Not supported in this build.
S4.C4	Local Network Segmentation (initial)	Upon connection, the IoT device is assigned to some local network segment in accordance with policy, which may include an assessment of its security posture.	No	Not supported in this build.
S4.C5	Behavioral Analysis	Device behavior is observed to determine whether the device meets the policy criteria required to be permitted to perform a given operation (e.g., to access a high-value resource or be placed on a given network segment).	No	Not supported in this build.
S4.C6	Local Network Segmentation (ongoing)	The IoT device can be reassigned to a different network	No	Not supported in this build.

Demo ID	Capability	Description	Demonstrated?	Explanation/Notes
		segment based on ongoing assessments of its conformance to policy criteria.		
S4.C7	Periodic Device Reauthentication	After connection, the IoT device's identity is periodically reauthenticated in order to maintain network access.	No	Not supported in this build.
S4.C8	Periodic Device Reauthorization	After connection, the IoT device's authorization to access the network is periodically reconfirmed in order to maintain network access.	No	Not supported in this build.
Scenario 5: Establishment and Maintenance of Credential and Device Security Posture Throughout the Lifecycle				
S5.C1	Trusted Firmware Updates	The device can download the most recent firmware update and verify its signature before it is installed.	No	Not supported in this build.
S5.C2	Credential Certificate Provisioning	The onboarding mechanism can interact with a certificate authority to sign a device's X.509 certificate and provision it onto the device.	No	Not supported in this build.
S5.C3	Credential Update	The device's network credential can be updated after it expires.	No	Not supported in this build.
S5.C4	Server Attestation	Successful server attestation is required prior to permitting the	No	Not supported in this build.

Demo ID	Capability	Description	Demonstrated?	Explanation/Notes
		server to perform some operation on the device (e.g., prior to downloading and installing updates onto the device).		
S5.C5	Secure Integration with MUD	The network-layer onboarding mechanism can convey necessary device communications intent information (e.g., the IoT device's MUD URL) to the network in encrypted form, thereby securely binding this information to the device and ensuring its confidentiality and integrity.	No	Not supported in this build.
S5.C6	Lifecycle Management Establishment	The device has a lifecycle management service and can automatically establish a secure association with it after performing network-layer onboarding and connecting to the network.	No	Not supported in this build.

3.5 Build 5 Demonstration Results

Table 3-5 lists the capabilities that were demonstrated by Build 5.

320 Table 3-5 Build 5 Capabilities Demonstrated

Demo ID	Capability	Description	Demonstrated?	Explanation/Notes
Scenario 0: Factory Provisioning				
S0.C1	Birth Credential Generation and Storage	The device's birth credentials are generated within or generated and provisioned into secure storage on the IoT device. For BRSKI, the credential is an IDevID certificate.	Yes	Supporting public/private keypair is generated within the secure element, and signed IDevID certificate is placed into the secure element.
S0.C2	Birth Credential Signing	The credential is signed by a trusted CA.	Yes	The IDevID certificate is signed by the Build 5 Manufacturer Provisioning Root (MPR).
S0.C3	Bootstrapping Information Availability	The bootstrapping information required for onboarding the device is made available as needed. For BRSKI, the bootstrapping information is the IDevID certificate provisioned into the device's secure element.	Yes	The device's IDevID certificate is generated using the public/private keypair that was generated in the device's secure element. This IDevID certificate is presented to verify the device's identity during network-layer onboarding.
Scenario 1: Trusted Network-Layer Onboarding				
S1.C1	Device Authentication	The onboarding mechanism authenticates the device's identity.	Yes	The device is authenticated using its provisioned IDevID.
S1.C2	Device Authorization	The onboarding mechanism verifies that the device is authorized to onboard to the network.	Yes	The device is implicitly granted authorization during the onboarding process within the registrar implementation. However, this authorization is contingent upon the device satisfying the policy

Demo ID	Capability	Description	Demonstrated?	Explanation/Notes
				requirements for onboarding.
S1.C3	Network Authentication	The device can verify the network's identity.	Yes	Demonstrated by the voucher.
S1.C4	Network Authorization	The device can verify that the network is authorized to take control of it.	Yes	The device authenticates to the network using EAP-TLS. The registrar gets a voucher from the MASA verifying that the network is authorized to onboard the device and it passes this voucher to the device so the device can verify that the network is authorized to onboard it.
S1.C5	Secure Local Credentialing	The onboarding mechanism securely provisions local network credentials to the device.	Yes	A local device identifier (LDevID) (i.e., the device's network credential) [1] is provisioned to the device as the culmination of the network-layer onboarding process.
S1.C6	Secure Storage	The local network credentials are provisioned to secure hardware-backed storage on the device.	No	The IDevID (birth credential) keys are generated with a TPM secure element. The EAP-TLS negotiation is configured to use keys from the secure element. The local network credentials (LDevID) are not scored in secure storage.
S1.C7	Network Selection	The onboarding mechanism provides the IoT device with the identifier of the network to which the device should onboard.	Yes	The identifier of the network is passed back in the common name field of the LDevID X.509 certificate.
S1.C8	Interoperability	The network-layer onboarding mechanism can onboard a minimum of two types of IoT devices (e.g., different device vendors and models).	Yes	Supported by BRSKI over IEEE 802.11 [7], but not demonstrated in this build.

Demo ID	Capability	Description	Demonstrated?	Explanation/Notes
Scenario 2: Trusted Application-Layer Onboarding				
S2.C1	Automatic Initiation of Streamlined Application-Layer Onboarding	The device can automatically (i.e., with no manual intervention required) initiate trusted application-layer onboarding after performing network-layer onboarding and connecting to the network. In this case, the application-layer onboarding bootstrapping information has been securely conveyed to the device during the network-layer onboarding process.	No	Not supported in this build
S2.C2	Automatic Initiation of Independent Application-Layer Onboarding	The device can automatically (i.e., with no manual intervention required) initiate trusted application-layer onboarding after performing network-layer onboarding and connecting to the network. In this case, the application-layer onboarding bootstrapping information has been pre-provisioned to the device by the device manufacturer or integrator (e.g., as part of an application that was installed on the device during the manufacturing process).	Yes	The pledge can use its IDevID and the private key in the secure element to automatically establish a TLS connection to an application server using OpenSSL s_client. The address of the application server has been pre-provisioned to the device by the manufacturer.

Demo ID	Capability	Description	Demonstrated?	Explanation/Notes
S2.C3	Trusted Application-Layer Onboarding	The device and a trusted application service can establish an encrypted connection without exposing any information that must be protected to ensure the confidentiality of the connection. They can then use that secure association to exchange application-layer information.	Yes	The pledge can use its IDevID and the private key in the secure element to automatically establish a TLS connection to an application server using OpenSSL s_client. The address of the application server has been pre-provisioned to the device by the manufacturer.
Scenario 3: Re-Onboarding a Device				
S3.C1	Credential Deletion	The device's network credential can be deleted.	Yes	The device is removed from Radius server by revoking its voucher.
S3.C2	De-Credentialed Device Cannot Connect	After the device's network credential has been deleted, the device is not able to connect to or communicate on the network securely.	Yes	If credential is removed from the registrar/radius server, the device will not connect. Certificate revocation through CRL is also implemented.
S3.C3	Re-Onboarding (network-layer)	After the device's network credential has been deleted, the network-layer onboarding mechanism can securely re-provision a network credential to the device, which the device can then use to connect to the network securely.	Yes	Upon a voucher being revoked, the LDevID is invalidated. The pledge can then perform the onboarding process again with a newly generated LDevID.
S3.C4	Re-Onboarding (application layer)	After the device's network credentials have been deleted and the device has been re-onboarded at the	Yes	After re-establishing a network connection, application onboarding happens automatically.

Demo ID	Capability	Description	Demonstrated?	Explanation/Notes
		network layer and re-connected to the network, the device can perform application-layer onboarding automatically.		
Scenario 4: Ongoing Device Validation				
S4.C1	Device Attestation (initial)	The network-layer onboarding mechanism requires successful device attestation prior to permitting the device to be onboarded.	No	Not supported in this build.
S4.C2	Device Attestation (application layer)	The application-layer onboarding mechanism requires successful device attestation prior to permitting the device to be onboarded.	No	Not supported in this build.
S4.C3	Device Attestation (ongoing)	Successful device attestation is required prior to permitting the device to perform some operation (e.g., accessing a high-value resource).	No	Not supported in this build.
S4.C4	Local Network Segmentation (initial)	Upon connection, the IoT device is assigned to some local network segment in accordance with policy, which may include an assessment of its security posture.	No	Not supported in this build.
S4.C5	Behavioral Analysis	Device behavior is observed to determine whether the device meets the policy criteria required to be permitted to perform a given operation (e.g., to access a high-value	Yes	Real time network events are propagated from the gateway(s) to the policy engine. When suspicious behavior is identified (e.g., contact denylisted IP address) device network access is revoked.

Demo ID	Capability	Description	Demonstrated?	Explanation/Notes
		resource or be placed on a given network segment).		
S4.C6	Local Network Segmentation (ongoing)	The IoT device can be reassigned to a different network segment based on ongoing assessments of its conformance to policy criteria.	No	Not supported in this build.
S4.C7	Periodic Device Reauthentication	After connection, the IoT device's identity is periodically reauthenticated in order to maintain network access.	No	Not supported in this build.
S4.C8	Periodic Device Reauthorization	After connection, the IoT device's authorization to access the network is periodically reconfirmed in order to maintain network access.	Yes	The continuous assurance policy is checked periodically, every 30 seconds in the demo. The policy sets the requirements for a device to be authorized to have access to the network. If a device fails this check, its voucher is revoked, invalidating the device's LDevID.
Scenario 5: Establish and Maintain Credential and Device Security Posture Throughout the Lifecycle				
S5.C1	Trusted Firmware Updates	The device can download the most recent firmware update and verify its signature before it is installed.	No	Not supported in this build.
S5.C2	Credential Certificate Provisioning	The onboarding mechanism can interact with a certificate authority to sign a device's X.509 certificate and provision it onto the device.	Yes	In the BRSKI flows, the onboarding process results in an LDevID (X.509) certificate being provisioned on the device, after the trustworthiness checks have been completed. This LDevID certificate is signed by the Domain CA.

Demo ID	Capability	Description	Demonstrated?	Explanation/Notes
S5.C3	Credential Update	The device's network credential (e.g., its LDevID or X.509 certificate) can be updated after it expires.	Yes	Device will automatically generate a new LDevID and re-onboard if LDevID expires.
S5.C4	Server Attestation	Successful server attestation is required prior to permitting the server to perform some operation on the device (e.g., prior to downloading and installing updates onto the device).	No	Not supported in this build.
S5.C5	Secure Integration with MUD	The network-layer onboarding mechanism can convey necessary device communications intent information (e.g., the IoT device's MUD URL) to the network in encrypted form, thereby securely binding this information to the device and ensuring its confidentiality and integrity.	Yes	The continuous assurance policy engine sporadically resolves the MUD document of each unique connected device using all information available. In this build we use the D3DB method of resolution, which resolves using chained verifiable credentials; specifically, the MUD document is bound to the device ID using a simulated managed firmware service. This provides a verifiable credential binding a device identifier (IDevID) to a full MUD document.
S5.C6	Lifecycle Management Establishment	The device has a lifecycle management service and can automatically establish a secure association with it after performing network-layer onboarding and connecting to the network.	No	Not supported in this build.

Appendix A References

[1] *IEEE Standard for Local and Metropolitan Area Networks - Secure Device Identity*, IEEE Std 802.1AR-2018 (Revision of IEEE Std 802.1AR-2009), 2 Aug. 2018, 73 pp. Available: https://ieeexplore.ieee.org/document/8423794

[2] Wi-Fi Alliance, *Wi-Fi Easy Connect™ Specification Version 3.0*, 2022. Available: https://www.wi-fi.org/system/files/Wi-Fi_Easy_Connect_Specification_v3.0.pdf

[3] M. Pritikin, M. Richardson, T.T.E. Eckert, M.H. Behringer, and K.W. Watsen, *Bootstrapping Remote Secure Key Infrastructure (BRSKI)*, IETF Request for Comments (RFC) 8995, October 2021. Available: https://datatracker.ietf.org/doc/rfc8995/

[4] E. Lear, R. Droms, and D. Romascanu, *Manufacturer Usage Description Specification,* IETF Request for Comments (RFC) 8520, March 2019. Available: https://tools.ietf.org/html/rfc8520

[5] Open Connectivity Foundation (OCF) Iotivity: https://iotivity.org/

[6] Thread 1.1.1 Specification, February 13, 2017.

[7] O. Friel, E. Lear, M. Pritikin, and M. Richardson, *BRSKI over IEEE 802.11*, IETF Internet-Draft (Individual), July 2018. Available: https://datatracker.ietf.org/doc/draft-friel-brski-over-802dot11/01/

NIST SPECIAL PUBLICATION 1800-36

Trusted Internet of Things (IoT) Device Network-Layer Onboarding and Lifecycle Management:

Enhancing Internet Protocol-Based IoT Device and Network Security

Includes Executive Summary (A); Approach, Architecture, and Security Characteristics (B); How-To Guides (C); Functional Demonstrations (D) and Compliance and Risk Management (E)

Michael Fagan
Jeffrey Marron
Paul Watrobski
Murugiah Souppaya
William Barker
Chelsea Deane
Joshua Klosterman
Charlie Rearick
Blaine Mulugeta
Susan Symington

Dan Harkins
Danny Jump
Andy Dolan
Kyle Haefner
Craig Pratt
Darshak Thakore
Peter Romness
Tyler Baker
David Griego
Brecht Wyseur

Alexandru Mereacre
Nick Allott
Ashley Setter
Julien Delplancke
Michael Richardson
Steve Clark
Mike Dow
Steve Egerter

May 2024

DRAFT

This publication is available free of charge from
https://www.nccoe.nist.gov/projects/trusted-iot-device-network-layer-onboarding-and-lifecycle-management

NIST SPECIAL PUBLICATION 1800-36

Trusted Internet of Things (IoT) Device Network-Layer Onboarding and Lifecycle Management: Enhancing Internet Protocol-Based IoT Device and Network Security

Includes Executive Summary (A); Approach, Architecture, and Security Characteristics (B); How-To Guides (C); Functional Demonstrations (D) and Compliance and Risk Management (E)

Michael Fagan
Jeffrey Marron
Paul Watrobski
Murugiah Souppaya
National Cybersecurity Center of Excellence
Information Technology Laboratory

William Barker
Dakota Consulting
Silver Spring, Maryland

Chelsea Deane
Joshua Klosterman
Charlie Rearick
Blaine Mulugeta
Susan Symington
The MITRE Corporation
McLean, Virginia

Dan Harkins
Danny Jump
Aruba, a Hewlett Packard Enterprise Company
San Jose, California

Andy Dolan
Kyle Haefner
Craig Pratt
Darshak Thakore
CableLabs
Louisville, Colorado

Peter Romness
Cisco
San Jose, California

Tyler Baker
David Griego
Foundries.io
London, United Kingdom

Brecht Wyseur
Kudelski IoT
Cheseaux-sur-Lausanne, Switzerland

Alexandru Mereacre
Nick Allott
Ashley Setter
NquiringMinds
Southampton, United Kingdom

Julien Delplancke
NXP Semiconductors
Mougins, France

Michael Richardson
Sandelman Software Works
Ontario, Canada

Steve Clark
SEALSQ, a subsidiary of WISeKey
Geneva, Switzerland

Mike Dow
Steve Egerter
Silicon Labs
Austin, Texas

DRAFT

May 2024

U.S. Department of Commerce
Gina M. Raimondo, Secretary

National Institute of Standards and Technology
Laurie Locasio, Under Secretary of Commerce for Standards and Technology & Director, National Institute of Standards and Technology

NIST SPECIAL PUBLICATION 1800-36A

Trusted Internet of Things (IoT) Device Network-Layer Onboarding and Lifecycle Management:

Enhancing Internet Protocol-Based IoT Device and Network Security

Volume A:
Executive Summary

Michael Fagan
Jeffrey Marron
Paul Watrobski
Murugiah Souppaya
National Cybersecurity Center of Excellence
Information Technology Laboratory

Blaine Mulugeta
Susan Symington
The MITRE Corporation
McLean, Virginia

Dan Harkins
Aruba, a Hewlett Packard Enterprise company
San Jose, California

William Barker
Dakota Consulting
Silver Spring, Maryland

Michael Richardson
Sandelman Software Works
Ottawa, Ontario

May 2024

DRAFT

This publication is available free of charge from
https://www.nccoe.nist.gov/projects/trusted-iot-device-network-layer-onboarding-and-lifecycle-management

NIST NATIONAL CYBERSECURITY CENTER OF EXCELLENCE

Executive Summary

Establishing trust between a network and an Internet of Things (IoT) device (as defined in NIST Internal Report 8425) prior to providing the device with the credentials it needs to join the network is crucial for mitigating the risk of potential attacks. There are two possibilities for attack. One happens when a device is convinced to join an unauthorized network, which would take control of the device. The other occurs when a network is infiltrated by a malicious device. Trust is achieved by attesting and verifying the identity and posture of the device and the network before providing the device with its network credentials—a process known as *network-layer onboarding*. In addition, scalable, automated mechanisms are needed to safely manage IoT devices throughout their lifecycles, such as safeguards that verify the security posture of a device before the device is permitted to execute certain operations. In this practice guide, the National Cybersecurity Center of Excellence (NCCoE) applies standards, best practices, and commercially available technology to demonstrate various mechanisms for trusted network-layer onboarding of IoT devices in Internet Protocol based environments. This guide shows how to provide network credentials to IoT devices in a trusted manner and maintain a secure device posture throughout the device lifecycle, thereby enhancing IoT security in alignment with the IoT Cybersecurity Improvement Act of 2020.

CHALLENGE

With 40 billion IoT devices expected to be connected worldwide by 2025, it is unrealistic to onboard or manage these devices by manually interacting with each device. In addition, providing local network credentials at the time of manufacture requires the manufacturer to customize network-layer onboarding on a build-to-order basis, which prevents the manufacturer from taking full advantage of the economies of scale that could result from building identical devices for its customers.

There is a need to have a scalable, automated mechanism to securely manage IoT devices throughout their lifecycles and, in particular, a trusted mechanism for providing IoT devices with their network credentials and access policy at the time of deployment on the network. It is easy for a network to falsely identify itself, yet many IoT devices onboard to networks without verifying the network’s identity and ensuring that it is their intended target network. Also, many IoT devices lack user interfaces, making it cumbersome to manually input network credentials. Wi-Fi is sometimes used to provide credentials over an open (i.e., unencrypted) network, but this onboarding method risks credential disclosure. Most home networks use a single password shared among all devices, so access is controlled only by the device’s possession of the password and does not consider a unique device identity or whether the device belongs on the network. This method also increases the risk of exposing credentials to unauthorized parties. Providing unique credentials to each device is more secure, but providing unique credentials manually would be resource-intensive and error-prone, would risk credential disclosure, and cannot be performed at scale.

Once a device is connected to the network, if it becomes compromised, it can pose a security risk to both the network and other connected devices. Not keeping such a device current with the most recent software and firmware updates may make it more susceptible to compromise. The device could also be attacked through receipt of malicious payloads. Once compromised, it may be used to attack other devices on the network.

OUTCOME

The outcome of this project is development of example trusted onboarding solutions, demonstration that they support various scenarios, and publication of the findings in this practice guide, a NIST Special Publication (SP) 1800 that is composed of multiple volumes targeting different audiences.

This practice guide can help IoT device users:

Understand how to onboard their IoT devices in a trusted manner to:

- **Ensure that their network is not put at risk** as new IoT devices are added to it
- **Safeguard their IoT devices** from being taken over by unauthorized networks
- **Provide IoT devices with unique credentials** for network access
- **Provide, renew, and replace device network credentials** in a secure manner
- **Support ongoing protection of IoT devices** throughout their lifecycles

This practice guide can help manufacturers and vendors of semiconductors, secure storage components, IoT devices, and network onboarding equipment:

Understand the desired security properties for supporting trusted network-layer onboarding and explore their options with respect to recommended practices for:

- **Providing unique credentials into secure storage on IoT devices at the time of manufacture to mitigate supply chain risks** (i.e., *device credentials*)
- **Installing onboarding software onto IoT devices**
- **Providing IoT device purchasers with information needed to onboard the IoT devices to their networks** (i.e., *device bootstrapping information*)
- **Integrating support for network-layer onboarding with additional security capabilities** to provide ongoing protection throughout the device lifecycle

SOLUTION

The NCCoE recommends the use of trusted network-layer onboarding to provide scalable, automated, trusted ways to provide IoT devices with unique network credentials and manage devices throughout their lifecycles to ensure that they remain secure. The NCCoE is collaborating with technology providers and other stakeholders to implement example trusted network-layer onboarding solutions for IoT devices that:

- provide each device with unique network credentials,
- enable the device and the network to mutually authenticate,
- send devices their credentials over an encrypted channel,
- do not provide any person with access to the credentials, and

- can be performed repeatedly throughout the device lifecycle.

The capabilities demonstrated include:

- trusted network-layer onboarding of IoT devices,
- repeated trusted network-layer onboarding of devices to the same or a different network,
- trusted application-layer onboarding (i.e., automatic establishment of an encrypted connection between an IoT device and a trusted application service after the IoT device has performed trusted network-layer onboarding and used its credentials to connect to the network), and
- software-based methods to provide device credentials in the factory and transfer device bootstrapping information from device manufacturer to device purchaser.

Future capabilities may include demonstrating the integration of trusted network-layer onboarding with zero trust-inspired [Note: See NIST SP 800-207] mechanisms such as ongoing device authorization, renewal of device network credentials, device attestation to ensure that only trusted IoT devices are permitted to be onboarded, device lifecycle management, and enforcement of device communications intent.

This demonstration follows an agile methodology of building implementations (i.e., *builds*) iteratively and incrementally, starting with network-layer onboarding and gradually integrating additional capabilities that improve device and network security throughout a managed device lifecycle. This includes factory builds that simulate activities performed to securely provide device credentials during the manufacturing process, and five network-layer onboarding builds that demonstrate the Wi-Fi Easy Connect, Bootstrapping Remote Secure Key Infrastructure (BRSKI), and Thread Commissioning protocols. These builds also demonstrate both streamlined and independent trusted application-layer onboarding approaches, along with policy-based continuous assurance and authorization. The example implementations use technologies and capabilities from our project collaborators (listed below).

Collaborators

Aruba, a Hewlett Packard Enterprise company
CableLabs
Cisco
Foundries.io
Kudelski IoT
NquiringMinds
NXP Semiconductors
Open Connectivity Foundation (OCF)
Sandelman Software Works
SEALSQ, a subsidiary of WISeKey
Silicon Labs

While the NCCoE uses a suite of commercial products, services, and proof-of-concept technologies to address this challenge, this guide does not endorse these particular products, services, and technologies, nor does it guarantee compliance with any regulatory initiatives. Your organization's information security experts should identify the products and services that will best integrate with your existing tools, IT and IoT system infrastructure, and operations. Your organization can adopt these solutions or one that adheres to these guidelines in whole, or you can use this guide as a starting point for tailoring and implementing parts of a solution.

HOW TO USE THIS GUIDE

Depending on your role in your organization, you might use this guide in different ways:

Business decision makers, such as chief information security, product security, and technology officers, can use this part of the guide, *NIST SP 1800-36A: Executive Summary*, to understand the project's challenges and outcomes, as well as our solution approach.

Technology, security, and privacy program managers who are concerned with how to identify, understand, assess, and mitigate risk can use *NIST SP 1800-36B: Approach, Architecture, and Security Characteristics*. This part of the guide describes the architecture and different implementations. Also, *NIST SP 1800-36E: Risk and Compliance Management,* maps components of the trusted onboarding reference architecture to security characteristics in broadly applicable, well-known cybersecurity guidelines and practices.

IT professionals who want to implement an approach like this can make use of *NIST SP 1800-36C: How-To Guides*. It provides product installation, configuration, and integration instructions for building example implementations, allowing them to be replicated in whole or in part. They can also use *NIST SP 1800-36D*: *Functional Demonstrations,* which provides the use cases that have been defined to showcase trusted network-layer onboarding and lifecycle management security capabilities and the results of demonstrating these capabilities with each of the example implementations. These use cases may be helpful when developing requirements for systems being developed.

SHARE YOUR FEEDBACK

You can view or download the preliminary draft guide at https://www.nccoe.nist.gov/projects/building-blocks/iot-network-layer-onboarding. NIST is adopting an agile process to publish this content. Each volume is being made available as soon as possible rather than delaying release until all volumes are completed.

Help the NCCoE make this guide better by sharing your thoughts with us as you read the guide. As example implementations continue to be developed, you can adopt this solution for your own organization. If you do, please share your experience and advice with us. We recognize that technical solutions alone will not fully enable the benefits of our solution, so we encourage organizations to share lessons learned and recommended practices for transforming the processes associated with implementing this guide.

To provide comments, join the community of interest, or learn more by arranging a demonstration of these example implementations, contact the NCCoE at iot-onboarding@nist.gov.

COLLABORATORS

Collaborators participating in this project submitted their capabilities in response to an open call in the Federal Register for all sources of relevant security capabilities from academia and industry (vendors and integrators). Those respondents with relevant capabilities or product components signed a Cooperative Research and Development Agreement (CRADA) to collaborate with NIST in a consortium to build this example solution.

Certain commercial entities, equipment, products, or materials may be identified by name or company logo or other insignia in order to acknowledge their participation in this collaboration or to describe an experimental procedure or concept adequately. Such identification is not intended to imply special status or relationship with NIST or recommendation or endorsement by NIST or the NCCoE; neither is it intended to imply that the entities, equipment, products, or materials are necessarily the best available for the purpose.

NIST SPECIAL PUBLICATION 1800-36B

Trusted Internet of Things (IoT) Device Network-Layer Onboarding and Lifecycle Management:

Enhancing Internet Protocol-Based IoT Device and Network Security

Volume B:
Approach, Architecture, and Security Characteristics

Michael Fagan
Jeffrey Marron
Paul Watrobski
Murugiah Souppaya
National Cybersecurity Center of Excellence
Information Technology Laboratory

William Barker
Dakota Consulting
Silver Spring, Maryland

Chelsea Deane
Joshua Klosterman
Charlie Rearick
Blaine Mulugeta
Susan Symington
The MITRE Corporation
McLean, Virginia

Dan Harkins
Danny Jump
Aruba, a Hewlett Packard Enterprise Company
San Jose, California

Andy Dolan
Kyle Haefner
Craig Pratt
Darshak Thakore
CableLabs
Louisville, Colorado

Peter Romness
Cisco
San Jose, California

Tyler Baker
David Griego
Foundries.io
London, United Kingdom

Brecht Wyseur
Kudelski IoT
Cheseaux-sur-Lausanne,
Switzerland

Alexandru Mereacre
Nick Allott
Ashley Setter
NquiringMinds
Southampton, United Kingdom

Julien Delplancke
NXP Semiconductors
Mougins, France

Michael Richardson
Sandelman Software Works
Ontario, Canada

Steve Clark
SEALSQ, a subsidiary of WISeKey
Geneva, Switzerland

Mike Dow
Steve Egerter
Silicon Labs
Austin, Texas

May 2024

DRAFT

This publication is available free of charge from
https://www.nccoe.nist.gov/projects/trusted-iot-device-network-layer-onboarding-and-lifecycle-management

DISCLAIMER

Certain commercial entities, equipment, products, or materials may be identified by name or company logo or other insignia in order to acknowledge their participation in this collaboration or to describe an experimental procedure or concept adequately. Such identification is not intended to imply special status or relationship with NIST or recommendation or endorsement by NIST or NCCoE; neither is it intended to imply that the entities, equipment, products, or materials are necessarily the best available for the purpose.

National Institute of Standards and Technology Special Publication 1800-36B, Natl. Inst. Stand. Technol. Spec. Publ. 1800-36B, 114 pages, May 2024, CODEN: NSPUE2

FEEDBACK

You can improve this guide by contributing feedback regarding which aspects of it you find helpful as well as suggestions on how it might be improved. Should we provide guidance summaries that target specific audiences? What trusted IoT device onboarding protocols and related features are most important to you? Is there some content that is not included in this document that we should cover? Are we missing anything in terms of technologies or use cases? In what areas would it be most helpful for us to focus our future related efforts? For example, should we consider implementing builds that onboard devices supporting Matter and/or the Fast Identity Online (FIDO) Alliance application onboarding protocol? Should we implement builds that integrate security mechanisms such as lifecycle management, supply chain management, attestation, or behavioral analysis? As you review and adopt this solution for your own organization, we ask you and your colleagues to share your experience and advice with us.

Comments on this publication may be submitted to: iot-onboarding@nist.gov.

Public comment period: May 31, 2024 through July 30, 2024

All comments are subject to release under the Freedom of Information Act.

National Cybersecurity Center of Excellence
National Institute of Standards and Technology
100 Bureau Drive
Mailstop 2002
Gaithersburg, MD 20899
Email: nccoe@nist.gov

NATIONAL CYBERSECURITY CENTER OF EXCELLENCE

The National Cybersecurity Center of Excellence (NCCoE), a part of the National Institute of Standards and Technology (NIST), is a collaborative hub where industry organizations, government agencies, and academic institutions work together to address businesses' most pressing cybersecurity issues. This public-private partnership enables the creation of practical cybersecurity solutions for specific industries, as well as for broad, cross-sector technology challenges. Through consortia under Cooperative Research and Development Agreements (CRADAs), including technology partners—from Fortune 50 market leaders to smaller companies specializing in information technology security—the NCCoE applies standards and best practices to develop modular, adaptable example cybersecurity solutions using commercially available technology. The NCCoE documents these example solutions in the NIST Special Publication 1800 series, which maps capabilities to the NIST Cybersecurity Framework and details the steps needed for another entity to re-create the example solution. The NCCoE was established in 2012 by NIST in partnership with the State of Maryland and Montgomery County, Maryland.

To learn more about the NCCoE, visit https://www.nccoe.nist.gov/. To learn more about NIST, visit https://www.nist.gov.

NIST CYBERSECURITY PRACTICE GUIDES

NIST Cybersecurity Practice Guides (Special Publication 1800 series) target specific cybersecurity challenges in the public and private sectors. They are practical, user-friendly guides that facilitate the adoption of standards-based approaches to cybersecurity. They show members of the information security community how to implement example solutions that help them align with relevant standards and best practices, and provide users with the materials lists, configuration files, and other information they need to implement a similar approach.

The documents in this series describe example implementations of cybersecurity practices that businesses and other organizations may voluntarily adopt. These documents do not describe regulations or mandatory practices, nor do they carry statutory authority.

KEYWORDS

application-layer onboarding; bootstrapping; Internet of Things (IoT); Manufacturer Usage Description (MUD); network-layer onboarding; onboarding; Wi-Fi Easy Connect.

ACKNOWLEDGMENTS

We are grateful to the following individuals for their generous contributions of expertise and time.

Name	Organization
Amogh Guruprasad Deshmukh	Aruba, a Hewlett Packard Enterprise company
Bart Brinkman	Cisco

Name	Organization
Eliot Lear	Cisco
George Grey	Foundries.io
David Griego	Foundries.io
Fabien Gremaud	Kudelski IoT
Faith Ryan	The MITRE Corporation
Toby Ealden	NquiringMinds
John Manslow	NquiringMinds
Antony McCaigue	NquiringMinds
Alexandru Mereacre	NquiringMinds
Loic Cavaille	NXP Semiconductors
Mihai Chelalau	NXP Semiconductors
Julien Delplancke	NXP Semiconductors
Anda-Alexandra Dorneanu	NXP Semiconductors
Todd Nuzum	NXP Semiconductors
Nicusor Penisoara	NXP Semiconductors
Laurentiu Tudor	NXP Semiconductors
Karen Scarfone	Scarfone Cybersecurity
Pedro Fuentes	SEALSQ, a subsidiary of WISeKey
Gweltas Radenac	SEALSQ, a subsidiary of WISeKey
Kalvin Yang	SEALSQ, a subsidiary of WISeKey

The Technology Partners/Collaborators who participated in this build submitted their capabilities in response to a notice in the Federal Register. Respondents with relevant capabilities or product components were invited to sign a Cooperative Research and Development Agreement (CRADA) with NIST, allowing them to participate in a consortium to build this example solution. We worked with:

Technology Collaborators

Aruba, a Hewlett Packard Enterprise company
CableLabs
Cisco
Foundries.io
Kudelski IoT
NquiringMinds
NXP Semiconductors
Open Connectivity Foundation (OCF)
Sandelman Software Works
SEALSQ, a subsidiary of WISeKey
Silicon Labs

DOCUMENT CONVENTIONS

The terms “shall” and “shall not” indicate requirements to be followed strictly to conform to the publication and from which no deviation is permitted. The terms “should” and “should not” indicate that among several possibilities, one is recommended as particularly suitable without mentioning or excluding others, or that a certain course of action is preferred but not necessarily required, or that (in the negative form) a certain possibility or course of action is discouraged but not prohibited. The terms “may” and “need not” indicate a course of action permissible within the limits of the publication. The terms “can” and “cannot” indicate a possibility and capability, whether material, physical, or causal.

CALL FOR PATENT CLAIMS

This public review includes a call for information on essential patent claims (claims whose use would be required for compliance with the guidance or requirements in this Information Technology Laboratory (ITL) draft publication). Such guidance and/or requirements may be directly stated in this ITL Publication or by reference to another publication. This call also includes disclosure, where known, of the existence of pending U.S. or foreign patent applications relating to this ITL draft publication and of any relevant unexpired U.S. or foreign patents.

ITL may require from the patent holder, or a party authorized to make assurances on its behalf, in written or electronic form, either:

a) assurance in the form of a general disclaimer to the effect that such party does not hold and does not currently intend holding any essential patent claim(s); or

b) assurance that a license to such essential patent claim(s) will be made available to applicants desiring to utilize the license for the purpose of complying with the guidance or requirements in this ITL draft publication either:

1. under reasonable terms and conditions that are demonstrably free of any unfair discrimination; or
2. without compensation and under reasonable terms and conditions that are demonstrably free of any unfair discrimination.

Such assurance shall indicate that the patent holder (or third party authorized to make assurances on its behalf) will include in any documents transferring ownership of patents subject to the assurance, provisions sufficient to ensure that the commitments in the assurance are binding on the transferee, and that the transferee will similarly include appropriate provisions in the event of future transfers with the goal of binding each successor-in-interest.

The assurance shall also indicate that it is intended to be binding on successors-in-interest regardless of whether such provisions are included in the relevant transfer documents.

Such statements should be addressed to: iot-onboarding@nist.gov.

Contents

1 Summary 1
1.1 Challenge 1
1.2 Solution 2
1.3 Benefits 3
2 How to Use This Guide 3
2.1 Typographic Conventions 5
3 Approach 5
3.1 Audience 7
3.2 Scope 8
3.3 Assumptions and Definitions 8
3.3.1 Credential Types 8
3.3.2 Integrating Security Enhancements 10
3.3.3 Device Limitations 12
3.3.4 Specifications Are Still Improving 12
3.4 Collaborators and Their Contributions 12
3.4.1 Aruba, a Hewlett Packard Enterprise Company 14
3.4.2 CableLabs 16
3.4.3 Cisco 17
3.4.4 Foundries.io 17
3.4.5 Kudelski IoT 18
3.4.6 NquiringMinds 18
3.4.7 NXP Semiconductors 20
3.4.8 Open Connectivity Foundation (OCF) 21
3.4.9 Sandelman Software Works 21
3.4.10 SEALSQ, a subsidiary of WISeKey 22
3.4.11 VaultIC408 23
3.4.12 Silicon Labs 23
4 Reference Architecture 25
4.1 Device Manufacture and Factory Provisioning Process 26
4.2 Device Ownership and Bootstrapping Information Transfer Process 28
4.3 Trusted Network-Layer Onboarding Process 31

4.4 Trusted Application-Layer Onboarding Process 32
4.5 Continuous Verification 35
5 Laboratory Physical Architecture 37
5.1 Shared Environment 40
5.1.1 Domain Controller 40
5.1.2 Jumpbox 40
5.2 Build 1 (Wi-Fi Easy Connect, Aruba/HPE) Physical Architecture 41
5.2.1 Wi-Fi Easy Connect Factory Provisioning Build Physical Architecture 42
5.3 Build 2 (Wi-Fi Easy Connect, CableLabs, OCF) Physical Architecture 43
5.4 Build 3 (BRSKI, Sandelman Software Works) Physical Architecture 44
5.5 Build 4 (Thread, Silicon Labs, Kudelski IoT) Physical Architecture 46
5.6 Build 5 (BRSKI, NquiringMinds) Physical Architecture 47
5.6.1 BRSKI Factory Provisioning Build Physical Architecture 48
6 General Findings 49
6.1 Wi-Fi Easy Connect 49
6.1.1 Mutual Authentication 50
6.1.2 Mutual Authorization 50
6.1.3 Secure Storage 50
6.2 BRSKI 50
6.2.1 Reliance on the Device Manufacturer 51
6.2.2 Mutual Authentication 51
6.2.3 Mutual Authorization 51
6.2.4 Secure Storage 51
6.3 Thread 51
6.4 Application-Layer Onboarding 52
6.4.1 Independent Application-Layer Onboarding 52
6.4.2 Streamline Application-Layer Onboarding 52
7 Additional Build Considerations 53
7.1 Network Authentication 53
7.2 Device Communications Intent 53
7.3 Network Segmentation 54
7.4 Integration with a Lifecycle Management Service 54
7.5 Network Credential Renewal 54

7.6 Integration with Supply Chain Management Tools 54
7.7 Attestation 54
7.8 Mutual Attestation 54
7.9 Behavioral Analysis 55
7.10 Device Trustworthiness Scale 55
7.11 Resource Constrained Systems 55

Appendix A List of Acronyms 56

Appendix B Glossary 59

Appendix C Build 1 (Wi-Fi Easy Connect, Aruba/HPE) 60

C.1 Technologies 60
C.2 Build 1 Architecture 62
C.2.1 Build 1 Logical Architecture 62
C.2.2 Build 1 Physical Architecture 64

Appendix D Build 2 (Wi-Fi Easy Connect, CableLabs, OCF) 65

D.1 Technologies 65
D.2 Build 2 Architecture 67
D.2.1 Build 2 Logical Architecture 67
D.2.2 Build 2 Physical Architecture 70

Appendix E Build 3 (BRSKI, Sandelman Software Works) 71

E.1 Technologies 71
E.2 Build 3 Architecture 73
E.2.1 Build 3 Logical Architecture 73
E.2.2 Build 3 Physical Architecture 75

Appendix F Build 4 (Thread, Silicon Labs-Thread, Kudelski KeySTREAM) 76

F.1 Technologies 76
F.2 Build 4 Architecture 78
F.2.1 Build 4 Logical Architecture 78
F.2.2 Build 4 Physical Architecture 83

Appendix G Build 5 (BRSKI over Wi-Fi, NquiringMinds) 84

G.1 Technologies 84
G.2 Build 5 Architecture 86

G.2.1 Build 5 Logical Architecture 86
G.2.2 Build 5 Physical Architecture 89
Appendix H Factory Provisioning Process 90
H.1 Factory Provisioning Process 90
H.1.1 Device Birth Credential Provisioning Methods 90
H.2 Factory Provisioning Builds – General Provisioning Process 92
H.3 BRSKI Factory Provisioning Builds (NquiringMinds and SEALSQ) 93
H.3.1 BRSKI Factory Provisioning Build Technologies 93
H.3.2 BRSKI Factory Provisioning Build Logical Architectures 95
H.3.3 BRSKI Factory Provisioning Build Physical Architectures 98
H.4 Wi-Fi Easy Connect Factory Provisioning Build (SEALSQ and Aruba/HPE) 98
H.4.1 Wi-Fi Easy Connect Factory Provisioning Build Technologies 98
H.4.2 Wi-Fi Easy Connect Factory Provisioning Build Logical Architecture 99
H.4.3 Wi-Fi Easy Connect Factory Provisioning Build Physical Architecture 100
Appendix I References 101

List of Figures

Figure 3-1 Aruba/HPE DPP Onboarding Components 16
Figure 3-2 Components for Onboarding an IoT Device that Communicates Using Thread to AWS IoT .. 24
Figure 4-1 Trusted IoT Device Network-Layer Onboarding and Lifecycle Management Logical Reference Architecture 25
Figure 4-2 IoT Device Manufacture and Factory Provisioning Process 27
Figure 4-3 Device Ownership and Bootstrapping Information Transfer Process 29
Figure 4-4 Trusted Network-Layer Onboarding Process 31
Figure 4-5 Trusted Streamlined Application-Layer Onboarding Process 33
Figure 4-6 Continuous Verification 35
Figure 5-1 NCCoE IoT Onboarding Laboratory Physical Architecture 38
Figure 5-2 Physical Architecture of Build 1 42
Figure 5-3 Physical Architecture of Wi-Fi Easy Connect Factory Provisioning Build 43
Figure 5-4 Physical Architecture of Build 2 44
Figure 5-5 Physical Architecture of Build 3 45
Figure 5-6 Physical Architecture of Build 4 47

Figure 5-7 Physical Architecture of Build 5 48
Figure 5-8 Physical Architecture of BRSKI Factory Provisioning Build 49
Figure C-1 Logical Architecture of Build 1 63
Figure D-1 Logical Architecture of Build 2 68
Figure E-1 Logical Architecture of Build 3 73
Figure F-1 Logical Architecture of Build 4: Device Preparation 80
Figure F-2 Logical Architecture of Build 4: Connection to the OpenThread Network 81
Figure F-3 Logical Architecture of Build 4: Application-Layer Onboarding using the Kudelski keySTREAM Service 82
Figure G-1 Logical Architecture of Build 5 87
Figure H-1 Logical Architecture of the First Version of the BRSKI Factory Provisioning Build 97
Figure H-2 Logical Architecture of the Second Version of the BRSKI Factory Provisioning Build 98
Figure H-3 Logical Architecture of the Wi-Fi Easy Connect Factory Provisioning Build 100

List of Tables

Table 3-1 Capabilities and Components Provided by Each Technology Partner/Collaborator 13
Table 5-1 Build 1 Products and Technologies 40
Table C-1 Build 1 Products and Technologies 60
Table E-1 Build 3 Products and Technologies 71
Table F-1 Build 4 Products and Technologies 76
Table G-1 Build 5 Products and Technologies 84

1 Summary

IoT devices are typically connected to a network. As with any other device needing to communicate on a network securely, an IoT device needs credentials that are specific to that network to help ensure that only authorized devices can connect to and use the network. A typical commercially available, mass-produced IoT device cannot be pre-provisioned with local network credentials by the manufacturer during the manufacturing process. Instead, the local network credentials will be provisioned to the device at the time of its deployment. This practice guide is focused on trusted methods of providing IoT devices with the network-layer credentials and policy they need to join a network upon deployment, a process known as *network-layer onboarding.*

Establishing trust between a network and an IoT device (as defined in NIST Internal Report 8425) prior to providing the device with the credentials it needs to join the network is crucial for mitigating the risk of potential attacks. There are two possibilities for attack. One is where a device is convinced to join an unauthorized network, which would take control of the device. The other is where a network is infiltrated by a malicious device. Trust is achieved by attesting and verifying the identity and posture of the device and the network before providing the device with its network credentials—a process known as *network-layer onboarding*. In addition, scalable, automated mechanisms are needed to safely manage IoT devices throughout their lifecycles, such as safeguards that verify the security posture of a device before the device is permitted to execute certain operations.

In this practice guide, the National Cybersecurity Center of Excellence (NCCoE) applies standards, best practices, and commercially available technology to demonstrate various mechanisms for trusted network-layer onboarding of IoT devices. This guide shows how to provide network credentials to IoT devices in a trusted manner and maintain a secure device posture throughout the device lifecycle.

1.1 Challenge

With 40 billion IoT devices expected to be connected worldwide by 2025 [1], it is unrealistic to onboard or manage these devices by visiting each device and performing a manual action. While it is possible for devices to be securely provided with their local network credentials at the time of manufacture, this requires the manufacturer to customize network-layer onboarding on a build-to-order basis, which prevents the manufacturer from taking full advantage of the economies of scale that could result from building identical devices for all its customers.

The industry lacks scalable, automatic mechanisms to safely manage IoT devices throughout their lifecycles and lacks a trusted mechanism for providing IoT devices with their network credentials and policy at the time of deployment on the network. It is easy for a network to falsely identify itself, yet many IoT devices onboard to networks without verifying the network's identity and ensuring that it is their intended target network. Also, many IoT devices lack user interfaces, making it cumbersome to manually input network credentials. Wi-Fi is sometimes used to provide credentials over an open (i.e., unencrypted) network, but this onboarding method risks credential disclosure. Most home networks use a single password shared among all devices, so access is controlled only by the device's possession of the password and does not consider a unique device identity or whether the device belongs on the network. This method also increases the risk of exposing credentials to unauthorized parties. Providing unique credentials to each device is more secure, but doing so manually would be resource-intensive and error-prone, would risk credential disclosure, and cannot be performed at scale.

Once a device is connected to the network, if it becomes compromised, it can pose a security risk to both the network and other connected devices. Not keeping such a device current with the most recent software and firmware updates may make it more susceptible to compromise. The device could also be attacked through the receipt of malicious payloads. Once compromised, it may be used to attack other devices on the network.

1.2 Solution

We need scalable, automated, trusted mechanisms to safely manage IoT devices throughout their lifecycles to ensure that they remain secure, starting with secure ways to provision devices with their network credentials, i.e., beginning with network-layer onboarding. Onboarding is a particularly vulnerable point in the device lifecycle because if it is not performed in a secure manner, then both the device and the network are at risk. Networks are at risk of having unauthorized devices connect to them, and devices are at risk of being taken over by networks that are not authorized to onboard or control them.

The NCCoE has adopted the trusted network-layer onboarding approach to promote automated, trusted ways to provide IoT devices with unique network credentials and manage devices throughout their lifecycles to ensure that they remain secure. The NCCoE is collaborating with CRADA consortium technology providers in a phased approach to develop example implementations of trusted network-layer onboarding solutions. We define a *trusted network-layer onboarding solution* to be a mechanism for provisioning network credentials to a device that:

- provides each device with unique network credentials,
- enables the device and the network to mutually authenticate,
- sends devices their network credentials over an encrypted channel,
- does not provide any person with access to the network credentials, and
- can be performed repeatedly throughout the device lifecycle to enable:
  - the device's network credentials to be securely managed and replaced as needed, and
  - the device to be securely onboarded to other networks after being repurposed or resold.

The use cases designed to be demonstrated by this project's implementations include:

- trusted network-layer onboarding of IoT devices
- repeated trusted network-layer onboarding of devices to the same or a different network
- automatic establishment of an encrypted connection between an IoT device and a trusted application service (i.e., *trusted application-layer onboarding*) after the IoT device has performed trusted network-layer onboarding and used its credentials to connect to the network
- policy-based ongoing device authorization
- software-based methods to provision device birth credentials in the factory

- mechanisms for IoT device manufacturers to provide IoT device purchasers with information needed to onboard the IoT devices to their networks (i.e., *device bootstrapping information*)

1.3 Benefits

This practice guide can benefit both IoT device users and IoT device manufacturers. The guide can help IoT device users understand how to onboard IoT devices to their networks in a trusted manner to:

- Ensure that their network is not put at risk as IoT devices are added to it
- Safeguard their IoT devices from being taken over by unauthorized networks
- Provide IoT devices with unique credentials for network access
- Provide, renew, and replace device network credentials in a secure manner
- Ensure that IoT devices can automatically and securely perform application-layer onboarding after performing trusted network-layer onboarding and connecting to a network
- Support ongoing protection of IoT devices throughout their lifecycles

This guide can help IoT device manufacturers, as well as manufacturers and vendors of semiconductors, secure storage components, and network onboarding equipment, understand the desired security properties for supporting trusted network-layer onboarding and demonstrate mechanisms for:

- Placing unique credentials into secure storage on IoT devices at time of manufacture (i.e., *device birth credentials*)
- Installing onboarding software onto IoT devices
- Providing IoT device purchasers with information needed to onboard the IoT devices to their networks (i.e., *device bootstrapping information*)
- Integrating support for network-layer onboarding with additional security capabilities to provide ongoing protection throughout the device lifecycle

2 How to Use This Guide

This NIST Cybersecurity Practice Guide demonstrates a standards-based reference design for implementing trusted IoT device network-layer onboarding and lifecycle management and describes various example implementations of this reference design. Each of these implementations, which are known as *builds,* is standards-based and is designed to help provide assurance that networks are not put at risk as new IoT devices are added to them and help safeguard IoT devices from connecting to unauthorized networks. The reference design described in this practice guide is modular and can be deployed in whole or in part, enabling organizations to incorporate trusted IoT device network-layer onboarding and lifecycle management into their legacy environments according to goals that they have prioritized based on risk, cost, and resources.

NIST is adopting an agile process to publish this content. Each volume is being made available as soon as possible rather than delaying release until all volumes are completed.

This guide contains five volumes:

- NIST Special Publication (SP) 1800-36A: *Executive Summary* – why we wrote this guide, the challenge we address, why it could be important to your organization, and our approach to solving this challenge
- NIST SP 1800-36B: *Approach, Architecture, and Security Characteristics* – what we built and why **(you are here)**
- NIST SP 1800-36C: *How-To Guides* – instructions for building the example implementations, including all the security-relevant details that would allow you to replicate all or parts of this project
- NIST SP 1800-36D: *Functional Demonstrations* – use cases that have been defined to showcase trusted IoT device network-layer onboarding and lifecycle management security capabilities, and the results of demonstrating these use cases with each of the example implementations
- NIST SP 1800-36E: *Risk and Compliance Management* – risk analysis and mapping of trusted IoT device network-layer onboarding and lifecycle management security characteristics to cybersecurity standards and recommended practices

Depending on your role in your organization, you might use this guide in different ways:

Business decision makers, including chief security and technology officers, will be interested in the *Executive Summary, NIST SP 1800-36A*, which describes the following topics:

- challenges that enterprises face in migrating to the use of trusted IoT device network-layer onboarding
- example solutions built at the NCCoE
- benefits of adopting the example solution

Technology or security program managers who are concerned with how to identify, understand, assess, and mitigate risk will be interested in *NIST SP 1800-36B*, which describes what we did and why.

Also, Section 4 of *NIST SP 1800-36E* will be of particular interest. Section 4, *Mappings*, maps logical components of the general trusted IoT device network-layer onboarding and lifecycle management reference design to security characteristics listed in various cybersecurity standards and recommended practices documents, including *Framework for Improving Critical Infrastructure Cybersecurity* (NIST Cybersecurity Framework) and *Security and Privacy Controls for Information Systems and Organizations* (NIST SP 800-53).

You might share the *Executive Summary, NIST SP 1800-36A*, with your leadership team members to help them understand the importance of using standards-based implementations for trusted IoT device network-layer onboarding and lifecycle management.

IT professionals who want to implement similar solutions will find all volumes of the practice guide useful. You can use the how-to portion of the guide, *NIST SP 1800-36C*, to replicate all or parts of the builds created in our lab. The how-to portion of the guide provides specific product installation, configuration, and integration instructions for implementing the example solution. We do not re-create the product manufacturers' documentation, which is generally widely available. Rather, we show how we incorporated the products together in our environment to create an example solution. Also, you can use *Functional Demonstrations, NIST SP 1800-36D*, which provides the use cases that have been defined to showcase trusted IoT device network-layer onboarding and lifecycle management security capabilities and the results of demonstrating these use cases with each of the example implementations. Finally, *NIST SP 1800-36E* will be helpful in explaining the security functionality that the components of each build provide.

This guide assumes that IT professionals have experience implementing security products within the enterprise. While we have used a suite of commercial products to address this challenge, this guide does not endorse these particular products. Your organization can adopt this solution or one that adheres to these guidelines in whole, or you can use this guide as a starting point for tailoring and implementing parts of a trusted IoT device network-layer onboarding and lifecycle management solution. Your organization's security experts should identify the products that will best integrate with your existing tools and IT system infrastructure. We hope that you will seek products that are congruent with applicable standards and recommended practices.

A NIST Cybersecurity Practice Guide does not describe "the" solution, but example solutions. We seek feedback on the publication's contents and welcome your input. Comments, suggestions, and success stories will improve subsequent versions of this guide. Please contribute your thoughts to iot-onboarding@nist.gov.

2.1 Typographic Conventions

The following table presents typographic conventions used in this volume.

Typeface/Symbol	Meaning	Example
Italics	file names and path names; references to documents that are not hyperlinks; new terms; and placeholders	For language use and style guidance, see the *NCCoE Style Guide*.
Bold	names of menus, options, command buttons, and fields	Choose **File** > **Edit**.
`Monospace`	command-line input, onscreen computer output, sample code examples, and status codes	`mkdir`
`Monospace Bold`	command-line user input contrasted with computer output	**`service sshd start`**
blue text	link to other parts of the document, a web URL, or an email address	All publications from NIST's NCCoE are available at https://www.nccoe.nist.gov.

3 Approach

This project builds on the document-based research presented in the NIST Draft Cybersecurity White Paper, *Trusted Internet of Things (IoT) Device Network-Layer Onboarding and Lifecycle Management* [2]. That paper describes key security and other characteristics of a trusted network-layer onboarding solution as well as the integration of onboarding with related technologies such as device attestation, device communications intent [3][4], and application-layer onboarding. The security and other attributes of the onboarding process that are cataloged and defined in that paper can provide assurance that the network is not put at risk as new IoT devices are added to it and also that IoT devices are safeguarded from being taken over by unauthorized networks.

To kick off this project, the NCCoE published a Federal Register Notice [5] inviting technology providers to participate in demonstrating approaches to deploying trusted IoT device network-layer onboarding and lifecycle management in home and enterprise networks, with the objective of showing how trusted IoT device network-layer onboarding can practically and effectively enhance the overall security of IoT devices and, by extension, the security of the networks to which they connect. The Federal Register Notice invited technology providers to provide products and/or expertise to compose prototypes. Components sought included network onboarding components and IoT devices that support trusted network-layer onboarding protocols; authorization services; supply chain integration services; access points, routers, or switches; components that support device communications intent management; attestation services; controllers or application services; IoT device lifecycle management services; and asset management services. Cooperative Research and Development Agreements (CRADAs) were established with qualified respondents, and teams of collaborators were assembled to build a variety of implementations.

NIST is following an agile methodology of building implementations iteratively and incrementally, starting with network-layer onboarding and gradually integrating additional capabilities that improve device and network security throughout a managed device lifecycle. The project team began by designing a general, protocol-agnostic reference architecture for trusted network-layer onboarding (see Section 4) and establishing a laboratory infrastructure at the NCCoE to host implementations (see Section 5).

Five build teams were established to implement trusted network-layer onboarding prototypes, and a sixth build team was established to demonstrate multiple builds for factory provisioning activities performed by an IoT device manufacturer to enable devices to support trusted network-layer onboarding. Each of the build teams fleshed out the initial architectures of their example implementations. They then used technologies, capabilities, and components from project collaborators to begin creating the builds:

- Build 1 (Wi-Fi Easy Connect, Aruba/HPE) uses components from Aruba, a Hewlett Packard Enterprise company, to support trusted network-layer onboarding using the Wi-Fi Alliance's Wi-Fi Easy Connect Specification, Version 2.0 [6] and independent (see Section 3.3.2) application-layer onboarding to the Aruba User Experience Insight (UXI) cloud.
- Build 2 (Wi-Fi Easy Connect, CableLabs, OCF) uses components from CableLabs to support trusted network-layer onboarding using the Wi-Fi Easy Connect protocol that allows provisioning of per-device credentials and policy management for each device. Build 2 also uses components from the Open Connectivity Foundation (OCF) to support streamlined (see Section 3.3.2) trusted application-layer onboarding to the OCF security domain.
- Build 3 (BRSKI, Sandelman Software Works) uses components from Sandelman Software Works to support trusted network-layer onboarding using the Bootstrapping Remote Secure Key Infrastructure (BRSKI) [7] protocol and an independent, third-party Manufacturer Authorized Signing Authority (MASA).

- Build 4 (Thread [8], Silicon Labs, Kudelski IoT) uses components from Silicon Labs to support connection to an OpenThread [9] network using pre-shared credentials and components from Kudelski IoT to support trusted application-layer onboarding to the Amazon Web Services (AWS) IoT core.
- Build 5 (BRSKI over Wi-Fi, NquiringMinds) uses components from NquiringMinds to support trusted network-layer onboarding using the BRSKI protocol over 802.11 [10]. Additional components from NquiringMinds support ongoing, policy-based, continuous assurance and authorization, as well as device communications intent enforcement.
- The BRSKI Factory Provisioning Build uses components from NquiringMinds to implement the factory provisioning flows. The build is implemented on Raspberry Pi devices, where the IoT secure element is an integrated Infineon Optiga™ SLB 9670 TPM 2.0. The device certificate authority (CA) is externally hosted on NquiringMinds servers. This build demonstrates activities for provisioning IoT devices with their initial (i.e., birth—see Section 3.3) credentials for use with the BRSKI protocol and for making device bootstrapping information available to device owners.
- The Wi-Fi Easy Connect Factory Provisioning Build uses Raspberry Pi devices and code from Aruba and secure storage elements, code, and a CA from SEALSQ, a subsidiary of WISeKey. This build demonstrates activities for provisioning IoT devices with their birth credentials for use with the Wi-Fi Easy Connect protocol and for making device bootstrapping information available to device owners.

Each build team documented the architecture and design of its build (see Appendix C, Appendix D, Appendix E, Appendix F, Appendix G, and Appendix H). As each build progressed, its team also documented the steps taken to install and configure each component of the build (see NIST SP 1800-36C).

The project team then designed a set of use case scenarios designed to showcase the builds' security capabilities. Each build team conducted a functional demonstration of its build by running the build through the defined scenarios and documenting the results (see NIST SP 1800-36D).

The project team also conducted a risk assessment and a security characteristic analysis and documented the results, including mappings of the security capabilities of the reference solution to both the *Framework for Improving Critical Infrastructure Cybersecurity* (NIST Cybersecurity Framework) [11] and Security and Privacy Controls for Information Systems and Organizations (*NIST SP 800-53 Rev. 5*) (see NIST SP 1800-36E).

Finally, the NCCoE worked with industry and standards-developing organization collaborators to distill their findings and consider potential enhancements to future support for trusted IoT device network-layer onboarding (see Section 6 and Section 7).

3.1 Audience

The intended audience for this practice guide includes:

- IoT device manufacturers, integrators, and vendors
- Semiconductor manufacturers and vendors
- Secure storage manufacturers

- Network equipment manufacturers
- IoT device owners and users
- Owners and administrators of networks (both home and enterprise) to which IoT devices connect
- Service providers (internet service providers/cable operators and application platform providers)

3.2 Scope

This project focuses on the trusted network-layer onboarding of IoT devices in both home and enterprise environments. Enterprise, consumer, and industrial use cases for trusted IoT device network-layer onboarding are all considered to be in scope at this time. The project encompasses trusted network-layer onboarding of IoT devices deployed across different Internet Protocol (IP) based environments using wired, Wi-Fi, and broadband networking technologies. The project addresses the onboarding of IP-based devices in the initial phase and will consider using technologies such as Zigbee or Bluetooth in future phases of this project.

The project's scope also includes security technologies that can be integrated with and enhanced by the trusted network-layer onboarding mechanism to protect the device and its network throughout the device's lifecycle. Examples of these technologies include supply chain management, device attestation, trusted application-layer onboarding, device communications intent enforcement, device lifecycle management, asset management, the dynamic assignment of devices to various network segments, and ongoing device authorization. Aspects of these technologies that are relevant to their integration with network-layer onboarding are within scope. Demonstration of the general capabilities of these technologies independent of onboarding is not within the project's scope. For example, demonstrating a policy that requires device attestation to be performed before the device will be permitted to be onboarded would be within scope. However, the details and general operation of the device attestation mechanism would be out of scope.

3.3 Assumptions and Definitions

This project is guided by a variety of assumptions, which are categorized by subsection below.

3.3.1 Credential Types

There are several different credentials that may be related to any given IoT device, which makes it important to be clear about which credential is being referred to. Two types of IoT device credentials are involved in the network-layer onboarding process: birth credentials and network credentials. Birth credentials are installed onto the device before it is released into the supply chain; trusted network-layer onboarding solutions leverage birth credentials to authenticate devices and securely provision them with their network credentials. If supported by the device and the application service provider, application-layer credentials may be provisioned to the device after the device performs network-layer onboarding and connects to the network, during the application-layer onboarding process. These different types of IoT device credentials are defined as follows:

- **Birth Credential**: In order to participate in trusted network-layer onboarding, devices must be equipped with a birth credential, which is sometimes also referred to as a device *birth identity* or *birth certificate.* A birth credential is a unique, authoritative credential that is generated or installed into secure storage on the IoT device during the pre-market phase of the device's lifecycle, i.e., before the device is released for sale. A manufacturer, integrator, or vendor typically generates or installs the birth credential onto an IoT device in the form of an Initial Device Identifier (IDevID) [12] and/or a public/private key pair.

  Birth credentials:

  - are permanent, and their value is independent of context;
  - enable the trusted network-layer onboarding process while keeping the device manufacturing process efficient; and
  - include a unique identity and a secret and can range from simple raw public and private keys to X.509 certificates that are signed by a trusted authority.

- **Network Credential:** A network credential is the credential that is provisioned to an IoT device during network-layer onboarding. The network credential enables the device to connect to the local network securely. A device's network credential may be changed repeatedly, as needed, by subsequent invocation of the trusted network-layer onboarding process.

Additional types of credentials that may also be associated with an IoT device are:

- **Application-Layer Credential:** An application-layer credential is a credential that is provisioned to an IoT device during application-layer onboarding. After an IoT device has performed network-layer onboarding and connected to a network, it may be provisioned with one or more application-layer credentials during the application-layer onboarding process. Each application-layer credential is specific to a given application and is typically unique to the device, and it may be replaced repeatedly over the course of the device's lifetime.
- **User Credential:** An IoT device that permits authorized users to access it and restricts access only to authorized users will have one or more user credentials associated with it. These credentials are what the users present to the IoT device in order to gain access to it. The user credential is not relevant during network-layer onboarding and is generally not of interest within the scope of this project. We include it in this list only for completeness. Many IoT devices may not even have user credentials associated with them.

In order to perform network- and application-layer onboarding, the device being onboarded must already have been provisioned with birth credentials. A pre-provisioned, unique, authoritative birth credential is essential for enabling the IoT device to be identified and authenticated as part of the trusted network-layer onboarding process, no matter what network the device is being onboarded to or how many times it is onboarded. The value of the birth credential is independent of context, whereas the network credential that is provisioned during network-layer onboarding is significant only with respect to the network to which the IoT device will connect. Each application-layer credential that is provisioned during application-layer onboarding is specific to a given application, and each user credential is specific to a given user. A given IoT device only ever has one birth credential over the course of its lifetime, and the value of this birth credential remains unchanged. However, that IoT device may have any number of network, application-layer, and user credentials at any given point in time, and these credentials may be replaced repeatedly over the course of the device's lifetime.

3.3.2 Integrating Security Enhancements

Integrating trusted network-layer IoT device onboarding with additional security mechanisms and technologies can help increase trust in both the IoT device and the network to which it connects. Examples of such security mechanism integrations demonstrated in this project include:

- **Trusted Application-Layer Onboarding:** When supported, application-layer onboarding can be performed automatically after a device has connected to its local network. Trusted application-layer onboarding enables a device to be securely provisioned with the application-layer credentials it needs to establish a secure association with a trusted application service. In many cases, a network's IoT devices will be so numerous that manually onboarding devices at the application layer would not be practical; in addition, dependence on manual application-layer onboarding would leave the devices vulnerable to accidental or malicious misconfiguration. So, application-layer onboarding, like network-layer onboarding, is fundamental to ensuring the overall security posture of each IoT device.

  As part of the application-layer onboarding process, devices and the application services with which they interact perform mutual authentication and establish an encrypted channel over which the application service can download application-layer credentials and software to the device and the device can provide information to the application service, as appropriate. Application-layer onboarding is useful for ensuring that IoT devices are executing the most up-to-date versions of their intended applications. It can also be used to establish a secure association between a device and a trusted lifecycle management service, which will ensure that the IoT device continues to be patched and updated with the latest firmware and software, thereby enabling the device to remain trusted throughout its lifecycle.

  Network-layer onboarding cannot be performed until after network-layer bootstrapping information has been introduced to the device and the network. This network-layer bootstrapping information enables the device and the network to mutually authenticate and establish a secure channel. Analogously, application-layer onboarding cannot be performed until after application-layer bootstrapping information has been introduced to the device and the application servers with which they will onboard. This application-layer bootstrapping information enables the device and the application server to mutually authenticate and establish a secure channel.

  - *Streamlined Application-Layer Onboarding*—One potential mechanism for introducing this application-layer bootstrapping information to the device and the application server is to use the network-layer onboarding process. The secure channel that is established during network-layer onboarding can serve as the mechanism for exchanging application-layer bootstrapping information between the device and the application server. By safeguarding the integrity and confidentiality of the application-layer bootstrapping information as it is conveyed between the device and the application server, the trusted network-layer onboarding mechanism helps to ensure that information that the device and the application server use to authenticate each other is truly secret and known only to them, thereby establishing a firm foundation for their secure association. In this way, trusted network-layer onboarding can provide a secure foundation for trusted application-layer onboarding. We call an application-layer onboarding process that uses network-layer onboarding to exchange application-layer bootstrapping information *streamlined* application-layer onboarding.

- *Independent Application-Layer Onboarding*—An alternative mechanism for introducing application-layer bootstrapping information to the device is to provide this information to the device during the manufacturing process. During manufacturing, the IoT device can be provisioned with software and associated bootstrapping information that enables the device to mutually authenticate with an application-layer service after it has connected to the network. This mechanism for performing application-layer onboarding does not rely on the network-layer onboarding process to provide application-layer bootstrapping information to the device. All that is required is that the device have connectivity to the application-layer onboarding service after it has connected to the network. We call an application-layer onboarding process that does not rely on network-layer onboarding to exchange application-layer bootstrapping information *independent* application-layer onboarding.

- **Segmentation:** Upon connection to the network, a device may be assigned to a particular local network segment to prevent it from communicating with other network components, as determined by enterprise policy. The device can be protected from other local network components that meet or do not meet certain policy criteria. Similarly, other local network components may be protected from the device if it meets or fails to meet certain policy criteria. A trusted network-layer onboarding mechanism may be used to convey information about the device that can be used to determine to which network segment it should be assigned upon connection. By conveying this information in a manner that protects its integrity and confidentiality, the trusted network-layer onboarding mechanism helps to increase assurance that the device will be assigned to the appropriate network segment. Post-onboarding, if a device becomes untrustworthy, for example because it is found to have software that has a known vulnerability or misconfiguration, or because it is behaving in a suspicious manner, the device may be dynamically assigned to a different network segment as a means of quarantining it, or its network-layer credential can be revoked or deleted.

- **Ongoing Device Authorization:** Once a device has been network-layer onboarded in a trusted manner and has possibly performed application-layer onboarding as well, it is important that as the device continues to operate on the network, it maintains a secure posture throughout its lifecycle. Ensuring the ongoing security of the device is important for keeping the device from being corrupted and for protecting the network from a potentially harmful device. Even though a device is authenticated and authorized prior to being onboarded, it is recommended that the device be subject to ongoing policy-based authentication and authorization as it continues to operate on the network. This may include monitoring device behavior and constraining communications to and from the device as needed in accordance with policy. In this manner, an ongoing device authorization service can ensure that the device and its operations continue to be authorized throughout the device's tenure on the network.

- **Device Communications Intent Enforcement:** Network-layer onboarding protocols can be used to securely transmit device communications intent information from the device to the network (i.e., to transmit this information in encrypted form with integrity protections). After the device has securely connected to the network, the network can use this device communications intent information to ensure that the device sends and receives traffic only from authorized locations. Secure conveyance of device communications intent information, combined with enforcement of it, ensures that IoT devices are constrained to sending and receiving only those communications that are explicitly required for each device to fulfill its purpose.

- **Additional Security Mechanisms:** Although not demonstrated in the implementations that have been built in this project so far, numerous additional security mechanisms can potentially be integrated with network-layer onboarding, beginning at device boot-up and extending through all phases of the device lifecycle. Examples of such mechanisms include integration with supply chain management tools, device attestation, automated lifecycle management, mutual attestation, and centralized asset management. Overall, application of these and other security protections can create a dependency chain of protections. This chain is based on a hardware root of trust as its foundation and extends up to support the security of the trusted network-layer onboarding process. The trusted network-layer onboarding process in turn may enable additional capabilities and provide a foundation that makes them more secure, thereby helping to ensure the ongoing security of the device and, by extension, the network.

3.3.3 Device Limitations

The security capabilities that any onboarding solution will be able to support will depend in part on the hardware, processing power, cryptographic modules, secure storage capacity, battery life, human interface (if any), and other capabilities of the IoT devices themselves, such as whether they support verification of firmware at boot time, attestation, application-layer onboarding, and device communications intent enforcement; what onboarding and other protocols they support; and whether they are supported by supply-chain tools. The more capable the device, the more security capabilities it should be able to support and the more robustly it should be able to support them. Depending on both device and onboarding solution capabilities, different levels of assurance may be provided.

3.3.4 Specifications Are Still Improving

Ideally, trusted network-layer onboarding solutions selected for widespread implementation and use will be openly available and standards-based. Some potential solution specifications are still being improved. In the meantime, their instability may be a limiting factor in deploying operational implementations of the proposed capabilities. For example, the details of running BRSKI over Wi-Fi are not fully specified at this time.

3.4 Collaborators and Their Contributions

Organizations participating in this project submitted their capabilities in response to an open call in the Federal Register for all sources of relevant security capabilities from academia and industry (vendors and integrators). Listed below are the respondents with relevant capabilities or product components (identified as “Technology Partners/Collaborators” herein) who signed a CRADA to collaborate with NIST in a consortium to build example trusted IoT device network-layer onboarding solution.

Technology Collaborators

Aruba, a Hewlett Packard Enterprise company
CableLabs
Cisco
Foundries.io
Kudelski IoT
NquiringMinds
NXP Semiconductors
Open Connectivity Foundation (OCF)
Sandelman Software Works
SEALSQ, a subsidiary of WISeKey
Silicon Labs

Table 3-1 summarizes the capabilities and components provided, or planned to be provided, by each partner/collaborator.

Table 3-1 Capabilities and Components Provided by Each Technology Partner/Collaborator

Collaborator	Security Capability or Component Provided
Aruba	Infrastructure for trusted network-layer onboarding using the Wi-Fi Easy Connect protocol and application-layer onboarding to the UXI cloud. IoT devices for use with both Wi-Fi Easy Connect network-layer onboarding and application-layer onboarding. The UXI Dashboard provides for an “always-on” remote technician with near real-time data insights into network and application performance.
CableLabs	Infrastructure for trusted network-layer onboarding using the Wi-Fi Easy Connect protocol. IoT devices for use with both Wi-Fi Easy Connect network-layer onboarding and application-layer onboarding to the OCF security domain.
Cisco	Networking components to support various builds.
Foundries.io	Factory software for providing birth credentials into secure storage on IoT devices and for transferring device bootstrapping information from device manufacturer to device purchaser.
Kudelski IoT	Infrastructure for trusted application-layer onboarding of a device to the AWS IoT core. The service comes with a cloud platform and a software agent that enables secure provisioning of AWS credentials into the secure storage of IoT devices.
NquiringMinds	Infrastructure for trusted network-layer onboarding using BRSKI over 802.11. Service that performs ongoing monitoring of connected devices to ensure their continued authorization (i.e., continuous authorization service), as well as device communications intent enforcement.
NXP Semiconductors	IoT devices with secure storage for use with both Wi-Fi Easy Connect and BRSKI network-layer onboarding. Service for provisioning credentials into secure storage of IoT devices.
Open Connectivity Foundation (OCF)	Infrastructure for trusted application-layer onboarding to the OCF security domain using IoTivity, an open-source software framework that implements the OCF specification.
Sandelman Software Works	Infrastructure for trusted network-layer onboarding using BRSKI. IoT devices for use with BRSKI network-layer onboarding.
SEALSQ, a subsidiary of WISeKey	Secure storage elements, code, and software that simulates factory provisioning of birth credentials to those secure elements on IoT devices in support of both Wi-Fi Easy Connect and BRSKI network-layer onboarding; certificate authority for signing device certificates.
Silicon Labs	Infrastructure for connection to a Thread network that has access to other networks for application-layer onboarding. IoT device with secure storage for use with Thread network connection and application-layer onboarding using Kudelski IoT.

Each of these technology partners and collaborators has described the relevant products and capabilities it brings to this trusted onboarding effort in the following subsections. The NCCoE does not certify or validate products or services. We demonstrate the capabilities that can be achieved by using participants' contributed technology.

3.4.1 Aruba, a Hewlett Packard Enterprise Company

Aruba, a Hewlett Packard Enterprise (HPE) company, provides secure, intelligent edge-to-cloud networking solutions that use artificial intelligence (AI) to automate the network, while harnessing data to drive powerful business outcomes. With Aruba ESP (Edge Services Platform) and as-a-service options as part of the HPE GreenLake family, Aruba takes a cloud-native approach to helping customers meet their connectivity, security, and financial requirements across campus, branch, data center, and remote worker environments, covering all aspects of wired, wireless local area networking (LAN), and wide area networking (WAN). Aruba ESP provides unified solutions for connectivity, visibility, and control throughout the IT-IoT workflow, with the objective of helping organizations accelerate IoT-driven digital transformation with greater ease, efficiency, and security. To learn more, visit Aruba at https://www.arubanetworks.com/.

3.4.1.1 Device Provisioning Protocol

Device Provisioning Protocol (DPP), certified under the Wi-Fi Alliance (WFA) as "Easy Connect," is a standard developed by Aruba that allows IoT devices to be easily provisioned onto a secure network. DPP improves security by leveraging Wi-Fi Protected Access 3 (WPA3) to provide device-specific credentials, enhance certificate handling, and support robust, secure, and scalable provisioning of IoT devices in any commercial, industrial, government, or consumer application. Aruba implements DPP through a combination of on-premises hardware and cloud-based services as shown in Table 3-1.

3.4.1.2 Aruba Access Point (AP)

From their unique vantage as ceiling furniture, Aruba Wi-Fi 6 APs have an unobstructed overhead view of all nearby devices. Built-in Bluetooth Low Energy (BLE) and Zigbee 802.15.4 IoT radios, as well as a flexible USB port, provide IoT device connectivity that allows organizations to address a broad range of IoT applications with infrastructure already in place, eliminating the cost of gateways and IoT overlay networks while enhancing IoT security.

Aruba's APs enable a DPP network through an existing Service Set Identifier (SSID) enforcing DPP access control and advertising the Configurator Connectivity Information Element (IE) to attract unprovisioned clients (i.e., clients that have not yet been onboarded). Paired with Aruba's cloud management service "Central", the APs implement the DPP protocol. The AP performs the DPP network introduction protocol (Connector exchange) with provisioned clients and assigns network roles.

3.4.1.3 Aruba Central

Aruba Central is a cloud-based networking solution with AI-powered insights, workflow automation, and edge-to-cloud security that empowers IT teams to manage and optimize campus, branch, remote, data center, and IoT networks from a single point of visibility and control. Built on a cloud-native, microservices architecture, Aruba Central is designed to simplify IT and IoT operations, improve agility, and reduce costs by unifying management of all network infrastructure.

Aruba’s “Central” Cloud DPP service exposes and controls many centralized functions to enable a seamless integrated end-to-end solution and act as a DPP service orchestrator. The cloud based DPP service selects an AP to authenticate unprovisioned enrollees (in the event that multiple APs receive the client *chirps*). The DPP cloud service holds the Configurator signing key and generates Connectors for enrollees authenticated through an AP.

3.4.1.4 IoT Operations

Available within Aruba Central, the IoT Operations service extends network administrators’ view into IoT devices and applications connected to the network. Organizations can gain critical visibility into previously invisible IoT devices, as well as reduce costs and complexity associated with deploying IoT applications. IoT Operations comprises three core elements:

- IoT Dashboard, which provides a granular view of devices connected to Aruba APs, as well as IoT connectors and applications in use.
- IoT App Store, a repository of click-and-go IoT applications that interface with IoT devices and their data.
- IoT Connector, which provisions multiple applications to be computed at the edge for agile IoT application support.

3.4.1.5 Client Insights

Part of Aruba Central, AI-powered Client Insights automatically identifies each endpoint connecting to the network with up to 99% accuracy. Client Insights discovers and classifies all connected endpoints—including IoT devices—using built-in machine learning and dynamic profiling techniques, helping organizations better understand what’s on their networks, automate access privileges, and monitor the behavior of each endpoint’s traffic flows to more rapidly spot attacks and act.

3.4.1.6 Cloud Auth

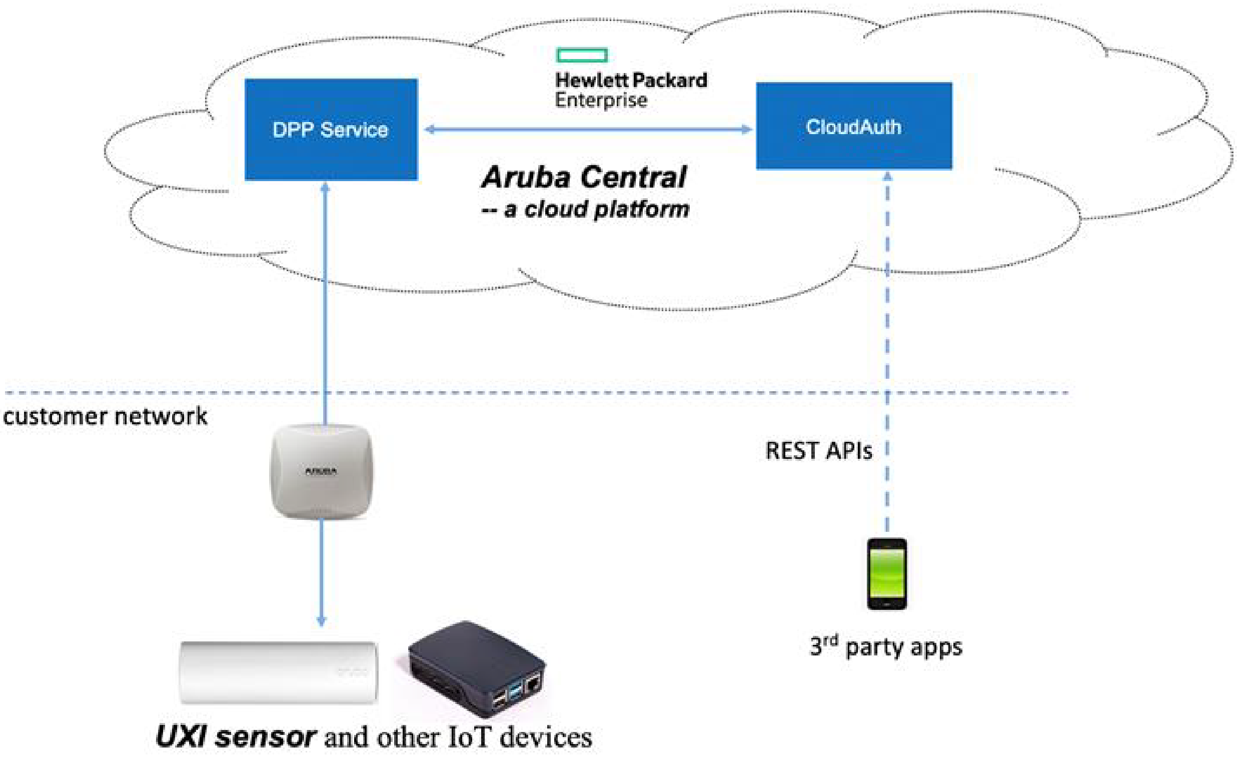
Cloud-native network access control (NAC) solution Cloud Auth delivers time-saving workflows to configure and manage onboarding, authorization, and authentication policies for wired and wireless networks. Cloud Auth integrates with an organization’s existing cloud identity store, such as Google Workspace or Azure Active Directory, to authenticate IoT device information and assign the right level of network access.

Cloud Auth operates as the DPP Authorization server and is the repository for trusted DPP Uniform Resource Identifiers (URIs) of unprovisioned enrollees. It maintains role information for each unprovisioned DPP URI and provisioned devices based on unique per-device credential (public key extracted from Connector). Representational State Transfer (RESTful) application programming interfaces (APIs) provide extensible capabilities to support third parties, making an easy path for integration and collaborative deployments.

3.4.1.7 UXI Sensor: DPP Enrollee

User Experience Insight (UXI) sensors continuously monitor end-user experience on customer networks and provide a simple-to-use cloud-based dashboard to assess networks and applications. The UXI sensor is onboarded in a zero-touch experience using DPP. Once network-layer onboarding is complete, the UXI sensor performs application-layer onboarding to the Aruba cloud to download a customer-specific profile. This profile enables the UXI sensor to perform continuous network testing and monitoring, and to troubleshoot network issues that it finds.

Figure 3-1 Aruba/HPE DPP Onboarding Components



3.4.2 CableLabs

CableLabs is an innovation lab for future-forward research and development (R&D)—a global meeting of minds dedicated to building and orchestrating emergent technologies. By convening peers and experts to share knowledge, CableLabs' objective is to energize the industry ecosystem for speed and scale. Its research facilitates solutions with the goal of making connectivity faster, easier, and more secure, and its conferences and events offer neutral meeting points to gain consensus.

As part of this project, CableLabs has provided the reference platform for its Custom Connectivity architecture for the purpose of demonstrating trusted network-layer onboarding of Wi-Fi devices using a variety of credentials. The following components are part of the reference platform.

3.4.2.1 Platform Controller

The controller provides interfaces and messaging for managing service deployment groups, access points with the deployment groups, registration and lifecycle of user services, and the secure onboarding and lifecycle management of users' Wi-Fi devices. The controller also exposes APIs for integration with third-party systems for the purpose of integrating various business flows (e.g., integration with manufacturing process for device management).

3.4.2.2 Custom Connectivity Gateway Agent

The Gateway Agent is a software component that resides on the Wi-Fi AP and gateway. It connects with the controller to coordinate the Wi-Fi and routing capabilities on the gateway. Specifically, it enforces the policies and configuration from the controller by managing the lifecycle of the Wi-Fi Extended Service Set/Basic Service Set (ESS/BSS) on the AP, authentication and credentials of the client devices that connect to the AP, and service management and routing rules for various devices. It also manages secure onboarding capabilities like Easy Connect, simple onboarding using a per-device pre-shared key (PSK), etc. The Gateway agent is provided in the form of an operational Raspberry Pi-based Gateway that also includes hostapd for Wi-Fi/DPP and open-vswitch for the creation of trust domains and routing.

3.4.2.3 Reference Clients

Three Raspberry Pi-based reference clients are provided. The reference clients have support for WFA Easy Connect-based onboarding as well as support for different Wi-Fi credentials, including per-device PSK and 802.1x certificates. One of the reference clients also has support for OCF-based streamlined application-layer onboarding.

3.4.3 Cisco

Cisco Systems, or Cisco, delivers collaboration, enterprise, and industrial networking and security solutions. The company’s cybersecurity team, Cisco Secure, is one of the largest cloud and network security providers in the world. Cisco’s Talos Intelligence Group, the largest commercial threat intelligence team in the world, is comprised of world-class threat researchers, analysts, and engineers, and supported by unrivaled telemetry and sophisticated systems. The group feeds rapid and actionable threat intelligence to Cisco customers, products, and services to help identify new threats quickly and defend against them. Cisco solutions are built to work together and integrate into your environment, using the “network as a sensor” and “network as an enforcer” approach to both make your team more efficient and keep your enterprise secure. Learn more about Cisco at https://www.cisco.com/go/secure.

3.4.3.1 Cisco Catalyst Switch

A Cisco Catalyst switch is provided to support network connectivity and network segmentation capabilities.

3.4.4 Foundries.io

Foundries.io helps organizations bring secure IoT and edge devices to market faster. The FoundriesFactory cloud platform offers DevOps teams a secure Linux-based firmware/operating system (OS) platform with device and fleet management services for connected devices, based on a fixed no-royalty subscription model. Product development teams gain enhanced security from boot to cloud while reducing the cost of developing, deploying, and updating devices across their installed lifetime. The open-source platform interfaces to any cloud and offers Foundries.io customers maximum flexibility for hardware configuration, so organizations can focus on their intellectual property, applications, and value add. For more information, please visit https://foundries.io/.

3.4.4.1 FoundriesFactory

FoundriesFactory is a cloud-based software platform provided by Foundries.io that offers a complete development and deployment environment for creating secure IoT devices. It provides a set of tools and services that enable developers to create, test, and deploy custom firmware images, as well as manage the lifecycle of their IoT devices.

Customizable components include open-source secure boot software, the open-source Linux microPlatform (LmP) distribution built with Yocto and designed for secure managed IoT and edge products, secure Over the Air (OTA) update facilities, and a Docker runtime for managing containerized applications and services. The platform is cross architecture (x86, Arm, and RISC-V) and enables secure connections to public and private cloud services.

Leveraging open standards and open software, FoundriesFactory is designed to simplify and accelerate the process of developing, deploying, and managing IoT and edge devices at scale, while also ensuring that they are secure and up to date over the product lifetime.

3.4.5 Kudelski IoT

Kudelski IoT is the Internet of Things division of Kudelski Group and provides end-to-end IoT solutions, IoT product design, and full-lifecycle services to IoT semiconductor and device manufacturers, ecosystem creators, and end-user companies. These solutions and services leverage the group's 30+ years of innovation in digital business model creation; hardware, software, and ecosystem design and testing; state-of-the-art security lifecycle management technologies and services; and managed operation of complex systems.

3.4.5.1 Kudelski IoT keySTREAM™

Kudelski IoT keySTREAM is a device-to-cloud, end-to-end solution for securing all the key assets of an IoT ecosystem during its entire lifecycle. The system provides each device with a unique, immutable, unclonable identity that forms the foundation for critical IoT security functions like in-factory or in-field provisioning, data encryption, authentication, and secure firmware updates, as well as allowing companies to revoke network access for vulnerable devices if necessary. This ensures that the entire lifecycle of the device and its data can be managed.

In this project, keySTREAM is used to enable trusted application-layer onboarding. It manages the attestation of devices, ownership, and provisioning of application credentials.

3.4.6 NquiringMinds

NquiringMinds provides intelligent trusted systems, combining AI-powered analytics with cyber security fundamentals. tdx Volt is the NquiringMinds general-purpose zero-trust services infrastructure platform, upon which it has built Cyber tdx, a cognitively enhanced cyber defense service designed for IoT. Both products are the latest iteration of the TDX product family. NquiringMinds is a UK company. Since 2010, it has been deploying its solutions into smart cities, health care, industrial, agricultural, financial technology, defense, and security sectors.

NquiringMinds collaborates within the open-standards and open-source community. It focuses on the principle of continuous assurance: the ability to continually reassess security risk by intelligently reasoning across the hard and soft information sources available. NquiringMinds' primary contributions to this project, described in the subsections below, are being made available as open source.

3.4.6.1 NquiringMinds' BRSKI Protocol Implementation

NquiringMinds has open sourced their software implementation of IETF's Bootstrapping Remote Secure Key Infrastructure (BRSKI) protocol, which provides a solution for secure zero-touch (automated) bootstrap of new (unconfigured) devices. This implementation includes the necessary adaptations for BRKSI to work with Wi-Fi networks.

The open source BRSKI implementation is available under an Apache 2.0 license at: https://github.com/nqminds/brski

3.4.6.2 TrustNetZ

NquiringMinds has open sourced the TrustNetZ (Zero Trust Networking) software stack which sits on top of their BRSKI implementation. TrustNetZ embodies the network onboarding and lifecycle management concepts into an easy to replicate demonstrator which includes the IoT device, the router, the router onboarding, the registrar, the manufacturer, the manufacturer provisioning, policy enforcement and continuous assurance servers.

This software also encapsulates NquiringMinds' continuous assurance capability, enhancing the security of the network by continually assessing whether connected IoT devices meet the policy requirements of the network. The software also includes a flexible, verifiable credential-based policy framework, which can rapidly be adapted to model different security and business model scenarios. The implementation models networking onboarding flows with EAP-TLS Wi-Fi certificates.

The open source TrustNetZ implementation is available under an Apache 2.0 license at: https://github.com/nqminds/trustnetz

3.4.6.3 edgeSEC

edgeSEC is an open-source, OpenWrt-based implementation of an intelligent secure router. It implements, on an open stack, the key components needed to implement both trusted onboarding and continuous assurance of devices. It contains an implementation of the Internet Engineering Task Force (IETF) BRSKI protocols, with the necessary adaptations for wireless onboarding, fully integrated into an open operational router. It additionally implements device communications intent constraints (IETF Manufacturer Usage Description [MUD]) and behavior monitoring (IoTSF ManySecured) that support some of the more enhanced trusted onboarding use cases. EdgeSEC additionally provides the platform for an asynchronous control plane for the continuous management of multiple routers and a general-purpose policy evaluation point, which can be used to demonstrate the breadth of onboarding and monitoring use cases that can be supported.

EdgeSEC is not directly used in the build that was demonstrated for this project, but it contains critical pieces of code that have been adapted in a simplified manner for the TrustNetZ implementation.

The open source edgeSEC implementation is available under an Apache 2.0 license at: https://github.com/nqminds/edgesec

3.4.6.4 tdx Volt

tdx Volt is NquiringMinds' zero-trust infrastructure platform. It encapsulates identity management, credential management, service discovery, and smart policy evaluation. This platform is designed to simplify the end-to-end demonstration of the trusted onboarding process and provides tools for use on the IoT device, the router, applications, and clouds. Tdx Volt is used by the TrustNetZ demonstrator as a verifiable credential issuer and verifier.

Tdx Volt is an NquiringMinds' product. Documented working implementation are available at: https://docs.tdxvolt.com/en/introduction

3.4.6.5 Reference Hardware

For demonstration purposes the NquiringMinds components can be deployed using the following hardware:

Compute hosts: Raspberry Pi 4

https://www.raspberrypi.com/products/raspberry-pi-4-model-b/. The Raspberry Pis are used to host the IoT client device, the router, and all additional compute services. Other Raspberry Pi models are also likely to work but have not been tested.

TPM/Secure Element

The secure storage for the IoT device (used in network-layer onboarding and factory provisioning) is provided by an Infineon Optiga™ SLB 9670 TPM 2.0, integrated through a Geeek Pi TPM hat. https://www.infineon.com/dgdl/Infineon-OPTIGA_SLx_9670_TPM_2.0_Pi_4-ApplicationNotes-v07_19-EN.pdf?fileId=5546d4626c1f3dc3016c3d19f43972eb.

A working version of the code is also available utilizing the SEALSQ Secure element https://www.sealsq.com/semiconductors/vaultic-secure-elements/vaultic-40x.

3.4.7 NXP Semiconductors

NXP Semiconductors focuses on secure connectivity solutions for embedded applications, NXP is impacting the automotive, industrial, and IoT, mobile, and communication infrastructure markets. Built on more than 60 years of combined experience and expertise, the company has approximately 31,000 employees in more than 30 countries. Find out more at https://www.nxp.com/.

3.4.7.1 EdgeLock SE050 secure element

The EdgeLock SE050 secure element (SE) product family offers strong protection against the latest attack scenarios and an extended feature set for a broad range of IoT use cases. This ready-to-use secure element for IoT devices provides a root of trust at the silicon level and delivers real end-to-end security – from edge to cloud – with a comprehensive software package for integration into any type of device.

3.4.7.2 EdgeLock 2GO

EdgeLock 2GO is the NXP service platform designed for easy and secure deployment and management of IoT devices. This flexible IoT service platform lets the device manufacturers and service providers choose the appropriate options to optimize costs while benefiting from an advanced level of device security. The EdgeLock 2GO service provisions the cryptographic keys and certificates into the hardware root of trust of the IoT devices and simplifies the onboarding of the devices to the cloud.

3.4.7.3 i.MX 8M family

The i.MX 8M family of applications processors based on Arm® Cortex®-A53 and Cortex-M4 cores provide advanced audio, voice, and video processing for applications that scale from consumer home audio to industrial building automation and mobile computers. It includes support for secure boot, secure debug, and lifecycle management, as well as integrated cryptographic accelerators. The development boards and Linux Board Support Package enablement provide out-of-the-box integration with an external SE050 secure element.

3.4.8 Open Connectivity Foundation (OCF)

OCF is a standards-developing organization that has had contributions and participation from over 450+ member organizations representing the full spectrum of the IoT ecosystem, from chip makers to consumer electronics manufacturers, silicon enablement software platform and service providers, and network operators. The OCF specification is an International Organization for Standardization/International Electrotechnical Commission (ISO/IEC) internationally recognized standard that was built in tandem with an open-source reference implementation called IoTivity. Additionally, OCF provides an in-depth testing and certification program.

3.4.8.1 IoTivity

OCF has contributed open-source code from IoTivity that demonstrates the advantage of secure network-layer onboarding and implements the WFA's Easy Connect to power a seamless bootstrapping of secure and trusted application-layer onboarding of IoT devices with minimal user interaction.

This code includes the interaction layer, called the OCF Diplomat, which handles secure communication between the DPP-enabled access point and the OCF application layer. The OCF onboarding tool (OBT) is used to configure and provision devices with operational credentials. The OCF reference implementation of a basic lamp is used to demonstrate both network- and application-layer onboarding and to show that once onboarded and provisioned, the OBT can securely interact with the lamp.

3.4.9 Sandelman Software Works

Sandelman Software Works (SSW) provides consulting and software design services in the areas of systems and network security. A complete stack company, SSW provides consulting and design services from the hardware driver level up to Internet Protocol Security (IPsec), Transport Layer Security (TLS), and cloud database optimization. SSW has been involved with the IETF since the 1990s, now dealing with the difficult problem of providing security for IoT systems. SSW leads standardization efforts through a combination of running code and rough consensus.

3.4.9.1 Minerva Highway IoT Network-Layer Onboarding and Lifecycle Management System

The Highway component is a cloud-native component operated by the device manufacturer (or its authorized designate). It provides the Request for Comments (RFC) 8995 [7] specified Manufacturer Authorized Signing Authority (MASA) for the BRSKI onboarding mechanism.

Highway is an asset manager for IoT devices. In its asset database it maintains an inventory of devices that have been manufactured, what type they are, and who the current owner of the device is (if it has been sold). Highway does this by taking control of the complete identity lifecycle of the device. It can aid in provisioning new device identity certificates (IDevIDs) by collecting Certificate Signing Requests and returning certificates, or by generating the new identities itself. This is consistent with Section 4.1.2.1 (On-device private key generation) and Section 4.1.2.2 (Off-device private key generation) of https://www.ietf.org/archive/id/draft-irtf-t2trg-taxonomy-manufacturer-anchors-00.html.

Highway can act as a standalone three-level private-public key infrastructure (PKI). Integrations with Automatic Certificate Management Environment (RFC 8555) allow it to provision certificates from an external PKI using the DNS-01 challenge in Section 8.4 of https://www.rfc-editor.org/rfc/rfc8555.html#section-8.4. Hardware integrations allow for the private key operations to be moved out of the main CPU. However, the needs of a busy production line in a factory would require continuous access to the hardware offload.

In practice, customers put the subordinate CA into Highway, which it needs to sign new IDevIDs, and put the trust anchor private CA into a hardware security module (HSM).

Highway provides a BRSKI-MASA interface running on a public TCP/HTTPS port (usually 443 or 9443). This service requires access to the private key associated to the anchor that has been “baked into” the Pledge device during manufacturing. The Highway instance that speaks to the world in this way does not have to be the same instance that signs IDevID certificates, and there are significant security advantages to separating them. Both instances do need access to the same database servers, and there are a variety of database replication techniques that can be used to improve resilience and security.

As IDevIDs do not expire, Highway does not presently include any mechanism to revoke IDevIDs, nor does it provide Certificate Revocation Lists (CRLs) or Online Certificate Status Protocol (OCSP). It is unclear how those mechanisms can work in practice.

Highway supports two models. In the Sales Integration model, the intended owner is known in advance. This model requires customer-specific integrations, which often occur at the database level through views or other SQL tools. In the trust on first use (TOFU) model, the first customer to claim a product becomes its owner.

3.4.10 SEALSQ, a subsidiary of WISeKey

WISeKey International Holding Ltd. (WISeKey) is a cybersecurity company that deploys digital identity ecosystems and secures IoT solution platforms. It operates as a Swiss-based holding company through several operational subsidiaries, each dedicated to specific aspects of its technology portfolio.

SEALSQ is the subsidiary of the group that focuses on designing and selling secure microcontrollers, PKI, and identity provisioning services while developing post-quantum technology hardware and software products. SEALSQ products and solutions are used across a variety of applications today, from multi-factor authentication devices, home automation systems, and network infrastructure, to automotive, industrial automation, and control systems.

3.4.11 VaultIC408

The VaultIC408 secure element combines hardware-based key storage with cryptographic accelerators to provide a wide array of cryptographic features including identity, authentication, encryption, key agreement, and data integrity. It protects against hardware attacks such as micro-probing and side channels.

The fundamental cryptography of the VaultIC family includes NIST-recommended algorithms and key lengths. Each of these algorithms, Elliptic Curve Cryptography (ECC), Rivest-Shamir-Adleman (RSA), and Advanced Encryption Standard (AES), is implemented on-chip and uses on-chip storage of the secret key material so the secrets are always protected in the secure hardware.

The secure storage and cryptographic acceleration support use cases like network and IoT end node security, platform security, secure boot, secure firmware download, secure communication or TLS, data confidentiality, encryption key storage, and data integrity.

3.4.11.1 INeS Certificate Management System (CMS)

SEALSQ’s portfolio includes INeS, a managed PKI-as-a-service solution. INeS leverages the WISeKey Webtrust-accredited trust services platform, a Matter approved Product Attestation Authority (PAA), and custom CAs. These PKI technologies support large-scale IoT deployments, where IoT endpoints will require certificates to establish their identities. The INeS CMS platform provides a secure, scalable, and manageable trust model.

INeS CMS provides certificate management, CA management, public cloud integration and automation, role-based access control (RBAC), and APIs for custom implementations.

3.4.12 Silicon Labs

Silicon Labs provides products in the area of secure, intelligent wireless technology for a more connected world. Securing IoT is challenging. It’s also mission critical. The challenge of protecting connected devices against frequently surfacing IoT security vulnerabilities follows device makers throughout the entire product lifecycle. Protecting products in a connected world is a necessity as customer data and modern online business models are increasingly targets for costly hacks and corporate brand damage. To stay secure, device makers need an underlying security platform in the hardware, software, network, and cloud. Silicon Labs offers security products with features that address escalating IoT threats, with the goal of reducing the risk of IoT ecosystem security breaches and the compromise of intellectual property and revenue loss from counterfeiting.

For this project, Silicon Labs has provided a host platform for the OpenThread border router (OTBR), a Thread radio transceiver, and an IoT device to be onboarded to the AWS cloud service and that communicates using the Thread wireless protocol.

3.4.12.1 OpenThread Border Router Platform

A Raspberry Pi serves as host platform for the OTBR. The OTBR forms a Thread network and acts as a bridge between the Thread network and the public internet, allowing the IoT device that communicates using the Thread wireless protocol and that is to be onboarded communicate with cloud services. The OTBR's connection to the internet can be made through either Wi-Fi or ethernet. Connection to the SLWSTK6023A (see Section 3.4.12.2) is made through a USB serial port.

3.4.12.2 SLWSTK6023A Thread Radio Transceiver

The SLWSTK6023A (Wireless starter kit) acts as a Thread radio transceiver or radio coprocessor (RCP). This allows the OTBR host platform to form and communicate with a Thread network.

3.4.12.3 xG24-DK2601B Thread "End" Device

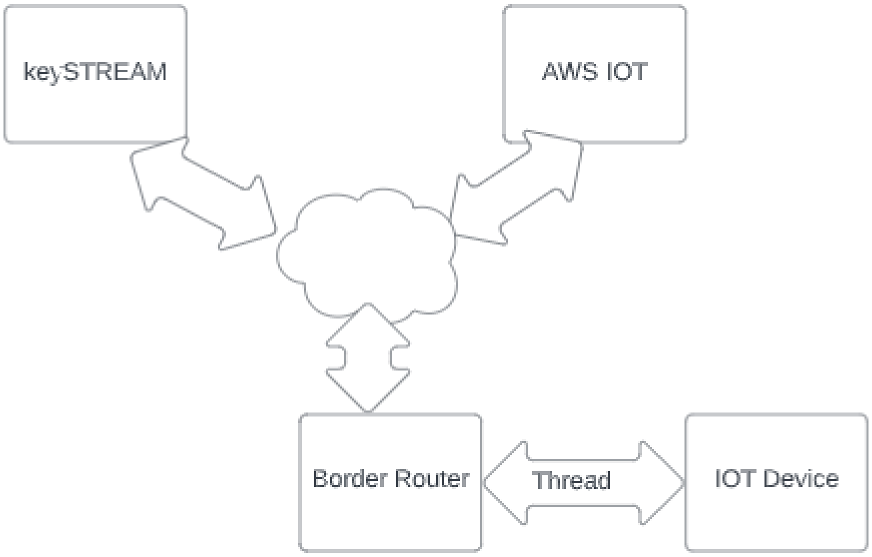
The xG24-DK2601B is the IoT device that is to be onboarded to the cloud service (AWS). It communicates using the Thread wireless protocol. Communication is bridged between the Thread network and the internet by the OTBR.

3.4.12.4 Kudelski IoT keySTREAM™

The Kudelski IoT keySTREAM solution is described more fully in Section 3.4.5.1. It is a cloud service capable of verifying the hardware-based secure identity certificate chain associated with the xG24-DK2601B component described in Section 3.4.12.3 and delivering a new certificate chain that can be refreshed or revoked as needed to assist with lifecycle management. The certificate chain is used to authenticate the xG24-DK2601B device to the cloud service (AWS).

Figure 3-2 shows the relationships among the components provided by Silicon Labs and Kudelski that support the trusted application-layer onboarding of an IoT device that communicates via the Thread protocol to AWS IoT.

Figure 3-2 Components for Onboarding an IoT Device that Communicates Using Thread to AWS IoT

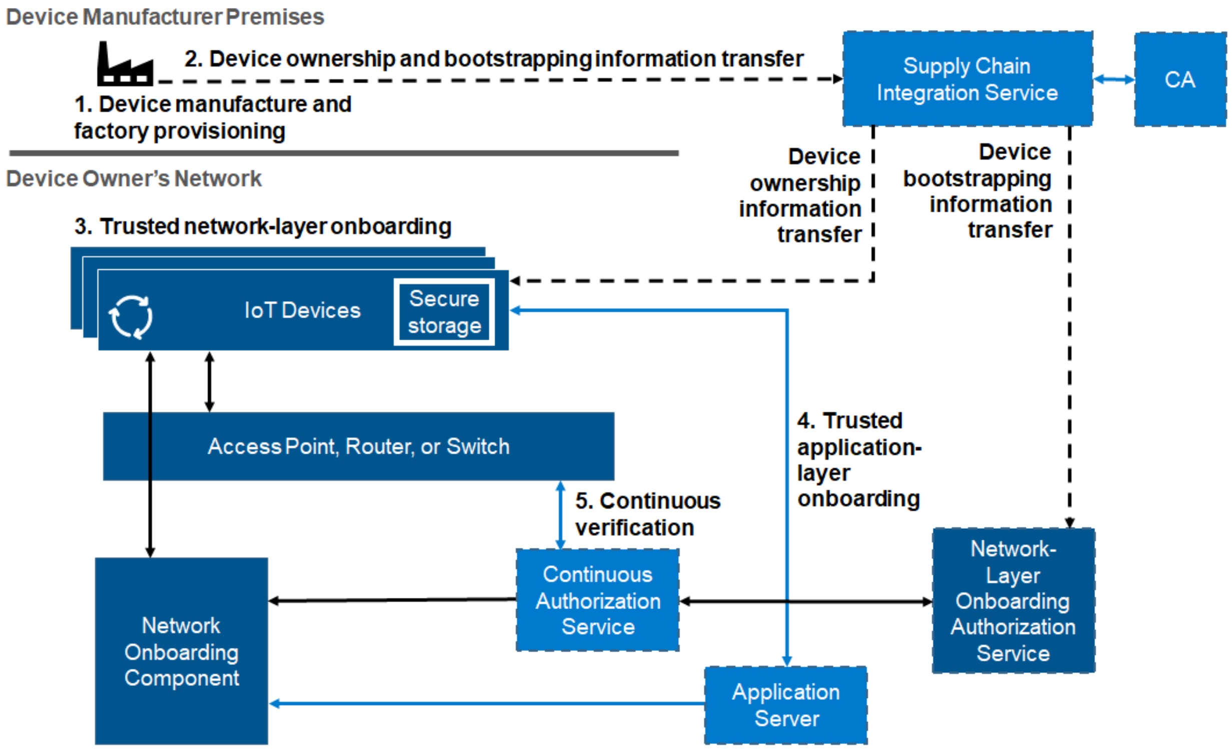


4 Reference Architecture

Figure 4-1 depicts the reference architecture to demonstrate trusted IoT device network-layer onboarding and lifecycle management used throughout this Practice Guide. This architecture shows a high-level, protocol-agnostic, and generic approach to trusted network-layer onboarding. It represents the basic components and processes, regardless of the network-layer onboarding protocol used and the particular device lifecycle management activities supported.

When implementing this architecture, an organization can follow different steps and use different components. The exact steps that are performed may not be in the same order as the steps in the logical reference architecture, and they may use components that do not have a one-to-one correspondence with the logical components in the logical reference architecture. In Appendices C, D, E, F and G we present the architectures for builds 1, 2, 3, 4 and 5, each of which is an instantiation of this logical reference architecture. Those build-specific architectures are more detailed and are described in terms of specific collaborator components and trusted network-layer onboarding protocols.

Figure 4-1 Trusted IoT Device Network-Layer Onboarding and Lifecycle Management Logical Reference Architecture


Device Manufacturer Premises
2. Device ownership and bootstrapping information transfer
1. Device manufacture and factory provisioning
Supply Chain Integration Service
CA
Device Owner's Network
Device ownership information transfer
Device bootstrapping information transfer
3. Trusted network-layer onboarding
IoT Devices
Secure storage
Access Point, Router, or Switch
4. Trusted application-layer onboarding
5. Continuous verification
Network Onboarding Component
Continuous Authorization Service
Network-Layer Onboarding Authorization Service
Application Server

There are five high-level processes to carry out this architecture, as labeled in Figure 4-1. These five processes are as follows:

1. **Device manufacture and factory provisioning** – the activities that the IoT device manufacturer performs to prepare the IoT device so that it is capable of network- and application-layer onboarding (Figure 4-2, Section 4.1).

2. **Device ownership and bootstrapping information transfer** – the transfer of IoT device ownership and bootstrapping information from the manufacturer to the device and/or the device's owner that enables the owner or an entity authorized by the owner to onboard the device securely. The component in Figure 4-1 labeled "Supply Chain Integration Service" represents the mechanism used to accomplish this information transfer (Figure 4-3, Section 4.2).

3. **Trusted network-layer onboarding** – the interactions that occur between the network-layer onboarding component and the IoT device to mutually authenticate, confirm authorization, establish a secure channel, and provision the device with its network credentials (Figure 4-4, Section 4.3).

4. **Trusted application-layer onboarding** – the interactions that occur between a trusted application server and the IoT device to mutually authenticate, establish a secure channel, and provision the device with application-layer credentials (Figure 4-5, Section 4.4).

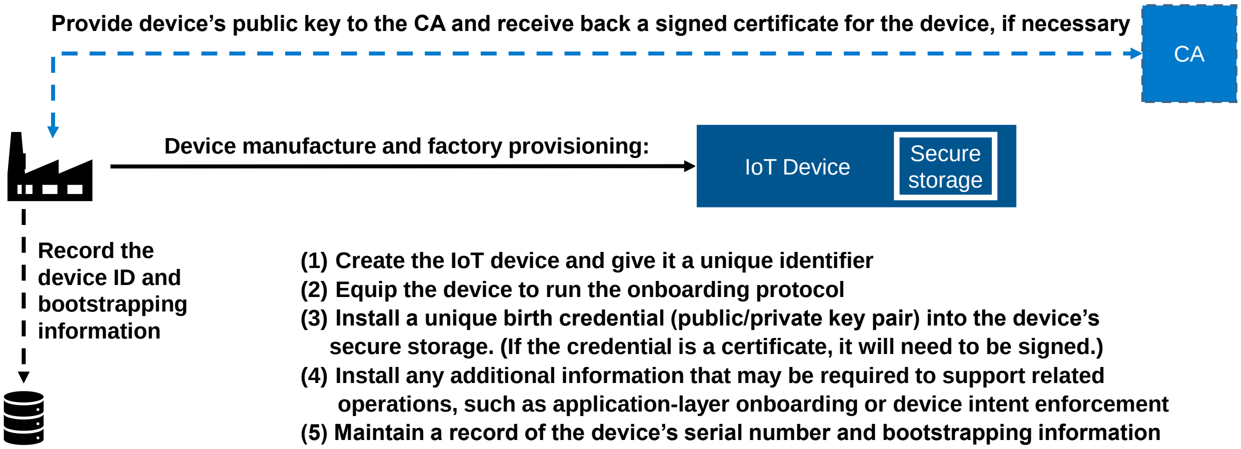
5. **Continuous verification** – ongoing, policy-based verification and authorization checks on the IoT device to support device lifecycle monitoring and control (Figure 4-6, Section 4.5).

Figure 4-1 uses two colors. The dark-blue components are central to supporting trusted network-layer onboarding itself. The light-blue components support the other aspects of the architecture. Each of the five processes is explained in more detail in the subsections below.

4.1 Device Manufacture and Factory Provisioning Process

Figure 4-2 depicts the device manufacture and factory provisioning process in more detail. As shown in Figure 4-2, the manufacturer is responsible for creating the IoT device and provisioning it with the necessary hardware, software, and birth credentials so that it is capable of network-layer onboarding. The IoT device should be manufactured with a secure root of trust as a best practice, possibly as part of a secure manufacturing process, particularly when outsourced. Visibility and control over the provisioning process and manufacturing supply chain, particularly for outsourced manufacturing, is critical in order to mitigate the risk of compromise in the supply chain, which could lead to the introduction of compromised devices. The CA component is shown in light blue in Figure 4-2 because its use is optional and depends on the type of credential that is being provisioned to the device (i.e., whether it is an 802.1AR certificate).

Figure 4-2 IoT Device Manufacture and Factory Provisioning Process


Provide device's public key to the CA and receive back a signed certificate for the device, if necessary
CA
Device manufacture and factory provisioning:
IoT Device
Secure storage
Record the device ID and bootstrapping information
(1) Create the IoT device and give it a unique identifier
(2) Equip the device to run the onboarding protocol
(3) Install a unique birth credential (public/private key pair) into the device's secure storage. (If the credential is a certificate, it will need to be signed.)
(4) Install any additional information that may be required to support related operations, such as application-layer onboarding or device intent enforcement
(5) Maintain a record of the device's serial number and bootstrapping information

At a high level, the steps that the manufacturer or an integrator performs as part of this preparation process, as shown in Figure 4-2, are as follows:

1. Create the IoT device and assign it a unique identifier (e.g., a serial number). Equip the device with secure storage.

2. Equip the device to run a specific network-layer onboarding protocol (e.g., Wi-Fi Easy Connect, BRSKI, Thread Mesh Commissioning Protocol (MeshCoP) [8]). This step includes ensuring that the device has the software/firmware needed to run the onboarding protocol as well as any additional information that may be required.

3. Generate or install the device's unique birth credential into the device's secure storage. [Note: using a secure element that has the ability to autonomously generate private/public root key pairs is inherently more secure than performing credential injection, which has the potential to expose the private key.] The birth credential includes information that must be kept secret (i.e., the device's private key) because it is what enables the device's identity to be authenticated. The contents of the birth credential will depend on what network-layer onboarding protocol the device supports. For example:

    a. If the device runs the Wi-Fi Easy Connect protocol, its birth credential will take the form of a unique private key, which has an associated DPP URI that includes the corresponding public key and possibly additional information such as Wi-Fi channel and serial number.

    b. If the device runs the BRSKI protocol, its birth credential takes the form of an 802.1AR certificate that gets installed as the device's IDevID and corresponding private key. The IDevID includes the device's public key, the location of the MASA, and trust anchors that can be used to verify vouchers signed by the MASA. The 802.1AR certificate needs to be signed by a trusted signing authority prior to installation, as shown in Figure 4-2.

4. Install any additional information that may be required to support related capabilities that are enabled by network-layer onboarding. The specific contents of the information that gets

installed on the device will vary according to what capabilities it is intended to support. For example, if the device supports:

   a. **streamlined application-layer onboarding** (see Section 3.3.2), then the bootstrapping information that is required to enable the device and a trusted application server to find and mutually authenticate each other and establish a secure association will be stored on the device. This is so it can be sent to the network during network-layer onboarding and used to automatically perform application-layer onboarding after the device has securely connected to the network. The Wi-Fi Easy Connect protocol, for example, can include such application-layer bootstrapping information as third-party information in its protocol exchange with the network, and Build 2 (i.e., the Wi-Fi Easy Connect, CableLabs, OCF build) demonstrates use of this mechanism to support streamlined application-layer onboarding.

      Note, however, that a device may still be capable of performing independent [see Section 3.3.2] application-layer onboarding even if the application-layer onboarding information is not exchanged as part of the network-layer onboarding protocol. The application that is installed on the device, i.e., the application that the device executes to fulfill its purpose, may include application-layer bootstrapping information that enables it to perform application-layer onboarding when it begins executing. Build 1 (i.e., the Wi-Fi Easy Connect, Aruba/HPE build) demonstrates independent application-layer onboarding.

   b. **device communications intent**, then the URI required to enable the network to locate the device's intent information may be stored on the device so that it can be sent to the network during network-layer onboarding. After the device has securely connected to the network, the network can use this device communications intent information to ensure that the device sends and receives traffic only from authorized locations.

5. Maintain a record of the device's serial number (or other uniquely identifying information) and the device's bootstrapping information. The manufacturer will take note of the device's ID and its bootstrapping information and store these. Eventually, when the device is sold, the manufacturer will need to provide the device's owner with its bootstrapping information. The contents of the device's bootstrapping information will depend on what network-layer onboarding protocol the device supports. For example:

   a. If the device runs the Wi-Fi Easy Connect protocol, its bootstrapping information is the DPP URI that is associated with its private key.

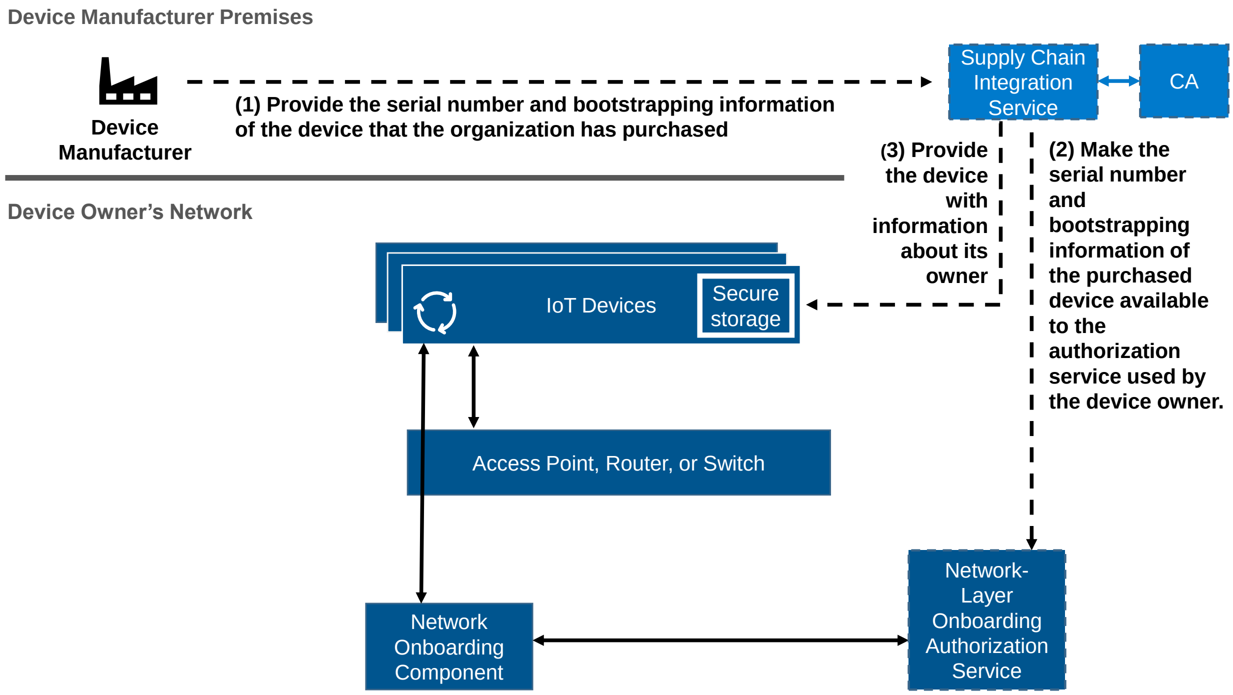
   b. If the device runs the BRSKI protocol, its bootstrapping information is its 802.1AR certificate.

4.2 Device Ownership and Bootstrapping Information Transfer Process

Figure 4-3 depicts the activities that are performed to transfer device bootstrapping information from the device manufacturer to the device owner, as well as to transfer device ownership information to the device itself, if appropriate. A high-level summary of these activities is described in the steps labeled A, B, and C.

The figure uses two colors. The dark-blue components are those used in the network-layer onboarding process. They are the same components as those depicted in the trusted network-layer onboarding process diagram provided in Figure 4-4. The light-blue components and their accompanying steps depict the portion of the diagram that is specific to device ownership and bootstrapping information transfer activities.

Figure 4-3 Device Ownership and Bootstrapping Information Transfer Process


Device Manufacturer Premises
Device Manufacturer
(1) Provide the serial number and bootstrapping information of the device that the organization has purchased
Supply Chain Integration Service
CA
(3) Provide the device with information about its owner
(2) Make the serial number and bootstrapping information of the purchased device available to the authorization service used by the device owner.
Device Owner's Network
IoT Devices
Secure storage
Access Point, Router, or Switch
Network Onboarding Component
Network-Layer Onboarding Authorization Service

These steps are as follows:

1. The device manufacturer makes the device serial number, bootstrapping information, and ownership information available so that the organization or individual who has purchased the device will have the device's serial number and bootstrapping information, and the device itself can be informed of who its owner is. In Figure 4-3, the manufacturer is shown sending this information to the supply chain integration service, which ensures that the necessary information ultimately reaches the device owner's authorization service as well as the device itself, if appropriate. (This description of the process is deliberately simple in order to enable it to be general enough that it applies to a variety of network-layer onboarding protocols.) In reality, the supply chain integration service mechanism for forwarding this bootstrapping information from the manufacturer to the owner may take many forms. For example, when BRSKI is used, the manufacturer sends the device serial number and bootstrapping information to a MASA that both the device and its owner trust. When other network-layer onboarding protocols are used, the device manufacturer may provide the device owner with this bootstrapping information directly by uploading this information to the owner's portion of a

trusted cloud. Such a mechanism is useful for the case in which the owner is a large enterprise that has made a bulk purchase of many IoT devices. In this case, the manufacturer can upload the information for hundreds or thousands of IoT devices to the supply chain integration service at once. We call this the enterprise use case. Alternatively, the device manufacturer may provide this information to the device owner indirectly by including it on or in the packaging of an IoT device that is sold at retail. We call this the consumer use case.

The contents of the device bootstrapping information will also vary according to the network-layer onboarding protocol that the device supports. For example, if the device supports the Wi-Fi Easy Connect network-layer onboarding protocol, the bootstrapping information will consist of the device's DPP URI. If the device supports the BRSKI network-layer onboarding protocol, bootstrapping information will consist of the device's IDevID (i.e., its 802.1AR certificate).

2. The supply chain integration service forwards the device serial number and bootstrapping information to an authorization service that has connectivity to the network-layer onboarding component that will onboard the device (i.e., to a network-layer onboarding component that belongs either to the device owner or to an entity that the device owner has authorized to onboard the device). The network-layer onboarding component will use the device's bootstrapping information to authenticate the device and verify that it is expected and authorized to be onboarded to the network. Again, this forwarding may take many forms, e.g., enterprise use case or consumer use case, and use a variety of different mechanisms within each use case type, e.g., information moved from one location to another in the device owner's portion of a trusted cloud, information transferred via a standardized protocol operating between the MASA and the onboarding network's domain registrar, or information scanned from a QR code on device packaging using a mobile app. In the case in which BRSKI is used, a certificate authority is consulted to help validate the signature of the 802.1AR certificate that comprises the device bootstrapping information.

3. The supply chain integration service may also provide the device with information about who its owner is. Knowing who its owner is enables the device to ensure that the network that is trying to onboard it is authorized to do so, because it is assumed that if a network owns a device, it is authorized to onboard it. The mechanisms for providing the device with assurance that the network that is trying to onboard it is authorized to do so can take a variety of forms, depending on the network-layer onboarding protocol being used. For example, if the Wi-Fi Easy Connect protocol is being used, then if an entity is in possession of the device's public key, that entity is assumed to be authorized to onboard the device. If BRSKI is being used, the device will be provided with a signed voucher verifying that the network that is trying to onboard the device is authorized to do so. The voucher is signed by the MASA. Because the device manufacturer has installed trust anchors for the MASA onto the device, the device trusts the MASA. It is also able to verify the MASA's signature.

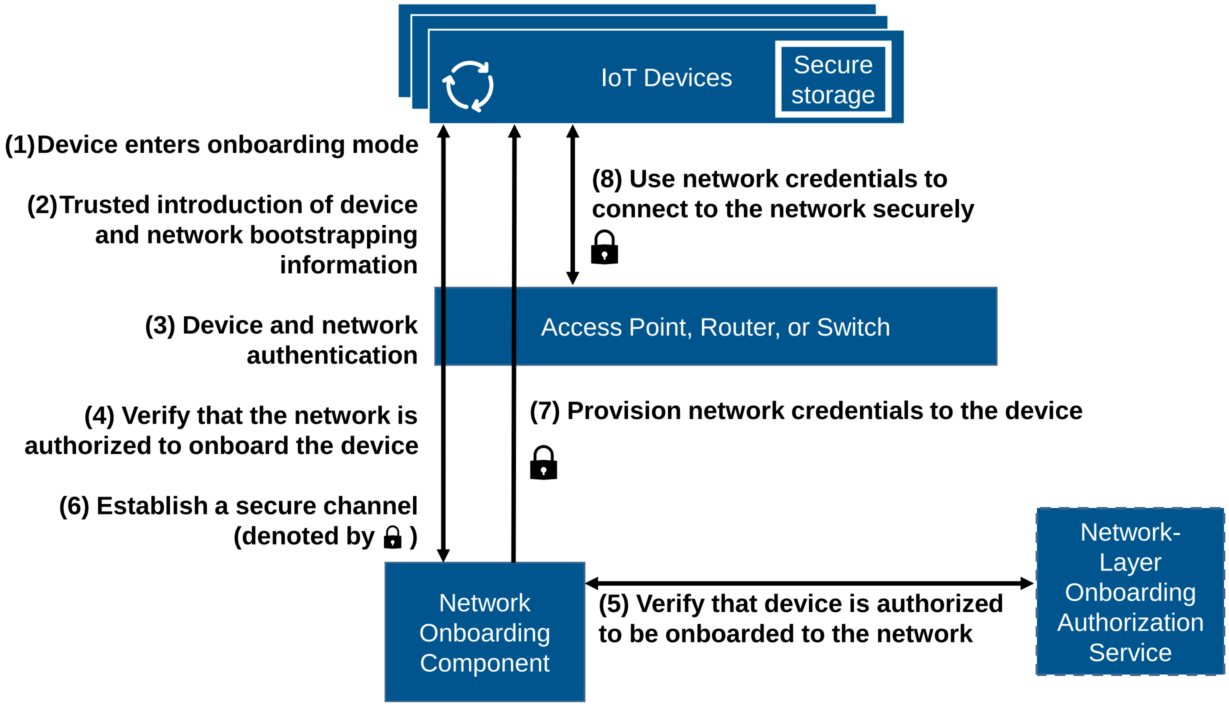
(Note: In this document, for the sake of simplicity, we often refer to the network that is authorized to onboard a device as the device owner's network. In reality, it may not always be the case that the device's owner also owns the network to which the device is being onboarded. While it is assumed that a network that owns a device is authorized to onboard it, and the device and the onboarding network are often owned by the same entity, common ownership is not a requirement. The network that is onboarding a device does not have to be the owner of that device. The network owner may permit devices that it does not own to be onboarded to the network. In order for such a device to be onboarded, the network owner must be in possession of the device's bootstrapping information. By accepting the bootstrapping information, the network owner is implicitly authorizing the device to be onboarded to its network. Conversely, a device may permit itself to be onboarded to a network that is not owned by the device's owner. A device owner that wants to authorize a network to onboard the device needs to ensure that the device trusts the onboarding network. The specific mechanism for accomplishing this will vary according to the network-layer onboarding protocol being used. When the Wi-Fi Easy Connect protocol is being used, simply providing the network with the device's public key is sufficient to authorize the network to onboard the device. When BRSKI is being used, the voucher that the MASA provides to the device must authorize the network to onboard it.)

Authentication of the network by the device may also take a variety of forms. These may range from simply trusting the person who is onboarding the device to onboard it to the correct network, to providing the IoT device with the network's public key.

4.3 Trusted Network-Layer Onboarding Process

Figure 4-4 depicts the trusted network-layer onboarding process in more detail. It shows the interactions that occur between the network-layer onboarding component and the IoT device to mutually authenticate, confirm that the device is authorized to be onboarded to the network, confirm that the network is authorized to onboard the device, establish a secure channel, and provision the device with its network credentials.

Figure 4-4 Trusted Network-Layer Onboarding Process


IoT Devices
Secure storage
(1)Device enters onboarding mode
(2)Trusted introduction of device and network bootstrapping information
(3) Device and network authentication
(4) Verify that the network is authorized to onboard the device
(6) Establish a secure channel (denoted by 🔒)
(8) Use network credentials to connect to the network securely
Access Point, Router, or Switch
(7) Provision network credentials to the device
Network Onboarding Component
(5) Verify that device is authorized to be onboarded to the network
Network-Layer Onboarding Authorization Service

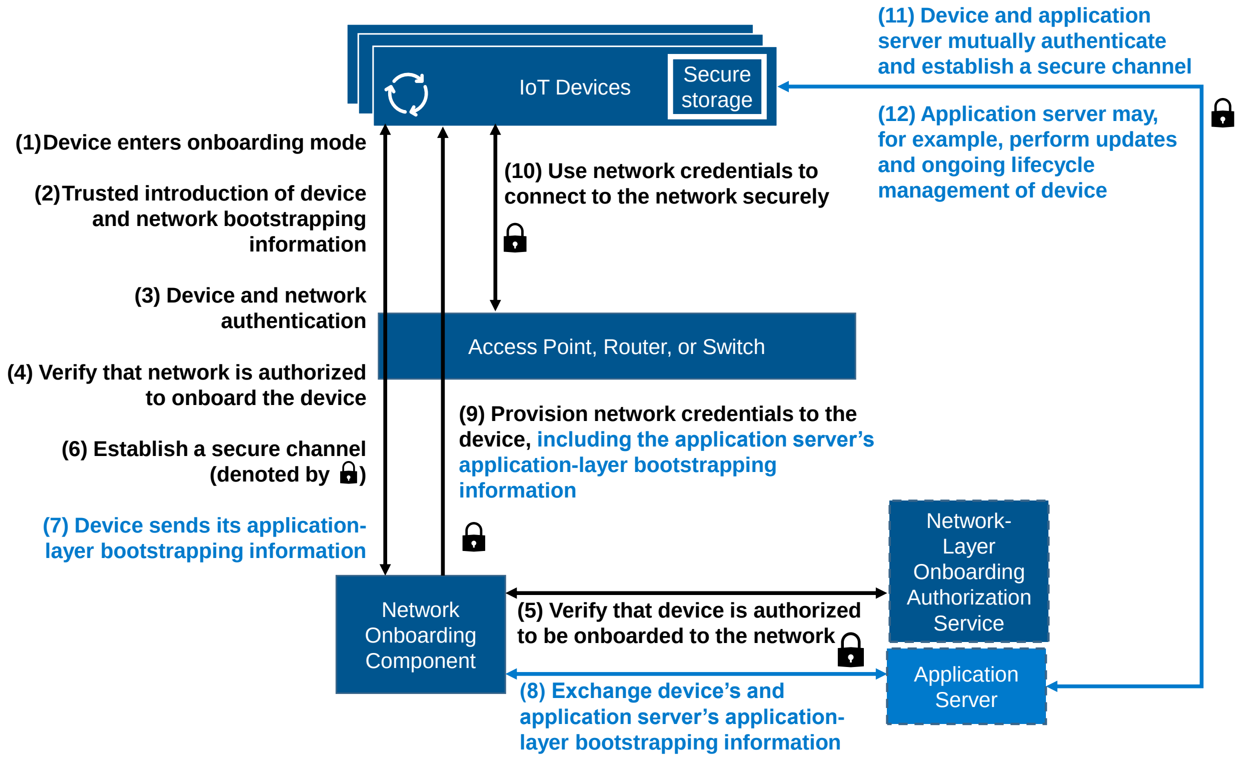
The numbered arrows in the diagram are intended to provide a high-level summary of the network-layer onboarding steps. These steps are assumed to occur after any device bootstrapping information and ownership transfer activities (as described in the previous section) that may need to be performed. The steps of the trusted network-layer onboarding process are as follows:

1. The IoT device to be onboarded is placed in onboarding mode, i.e., it is put into a state such that it is actively listening for and/or sending initial onboarding protocol messages.
2. Any required device bootstrapping information that has not already been provided to the network and any required network bootstrapping information that has not already been provided to the device are introduced in a trusted manner.
3. Using the device and network bootstrapping information that has been provided, the network authenticates the identity of the IoT device (e.g., by ensuring that the IoT device is in possession of the private key that corresponds with the public key for the device that was provided as part of the device's bootstrapping information), and the IoT device authenticates the identity of the network (e.g., by ensuring that the network is in possession of the private key that corresponds with the public key for the network that was provided as part of the network's bootstrapping information).
4. The device verifies that the network is authorized to onboard it. For example, the device may verify that it and the network are owned by the same entity, and therefore, assume that the network is authorized to onboard it.
5. The network onboarding component consults the network-layer onboarding authorization service to verify that the device is authorized to be onboarded to the network. For example, the network-layer authorization service can confirm that the device is owned by the network and is on the list of devices authorized to be onboarded.
6. A secure (i.e., encrypted) channel is established between the network onboarding component and the device.
7. The network onboarding component uses the secure channel that it has established with the device to confidentially send the device its unique network credentials.
8. The device uses its newly provisioned network credentials to establish secure connectivity to the network. The access point, router, or switch validates the device's credentials in this step. The mechanism it uses to do so varies depending on the implementation and is not depicted in Figure 4-4.

4.4 Trusted Application-Layer Onboarding Process

Figure 4-5 depicts the trusted application-layer onboarding process as enabled by the streamlined application-layer onboarding mechanism. As defined in Section 3.3.2, streamlined application-layer onboarding occurs after network-layer onboarding and depends upon and is enabled by it. The figure uses two colors. The dark-blue components are those used in the network-layer onboarding process. They and their accompanying steps (written in black font) are identical to those found in the trusted network-layer onboarding process diagram provided in Figure 4-4. The light-blue component and its accompanying steps (written in light-blue font) depict the portion of the diagram that is specific to streamlined application-layer onboarding.

Figure 4-5 Trusted Streamlined Application-Layer Onboarding Process


IoT Devices
Secure storage
(11) Device and application server mutually authenticate and establish a secure channel
(12) Application server may, for example, perform updates and ongoing lifecycle management of device
(1)Device enters onboarding mode
(2)Trusted introduction of device and network bootstrapping information
(3) Device and network authentication
(4) Verify that network is authorized to onboard the device
(6) Establish a secure channel (denoted by 🔒)
(7) Device sends its application-layer bootstrapping information
(10) Use network credentials to connect to the network securely
Access Point, Router, or Switch
(9) Provision network credentials to the device, including the application server's application-layer bootstrapping information
Network-Layer Onboarding Authorization Service
Network Onboarding Component
(5) Verify that device is authorized to be onboarded to the network
Application Server
(8) Exchange device's and application server's application-layer bootstrapping information

As is the case with Figure 4-4, the steps in this diagram are assumed to occur after any device ownership and bootstrapping information transfer activities that may need to be performed. Steps 1-6 in this figure are identical to Steps 1-6 in the trusted network-layer onboarding diagram of Figure 4-4, but steps 7 and 8 are different. With the completion of steps 1-6 in Figure 4-5, a secure channel has been established between the IoT device and the network-layer onboarding component. However, the device does not get provisioned with its network-layer credentials until step 9. To support streamlined application-layer onboarding, additional steps are required. Steps 1-12 are as follows:

1. The IoT device to be onboarded is placed in onboarding mode, i.e., it is put into a state such that it is actively listening for and/or sending initial onboarding protocol messages.

2. Any required device bootstrapping information that has not already been provided to the network and any required network bootstrapping information that has not already been provided to the device are introduced in a trusted manner.

3. Using the device and network bootstrapping information that has been provided, the network authenticates the identity of the IoT device (e.g., by ensuring that the IoT device is in possession of the private key that corresponds with the public key for the device that was provided as part of the device's bootstrapping information), and the IoT device authenticates the identity of the network (e.g., by ensuring that the network is in possession of the private key that corresponds

with the public key for the network that was provided as part of the network's bootstrapping information).

4. The device verifies that the network is authorized to onboard it. For example, the device may verify that it and the network are owned by the same entity, and therefore, assume that the network is authorized to onboard it.

5. The network onboarding component consults the network-layer onboarding authorization service to verify that the device is authorized to be onboarded to the network. For example, the network-layer authorization service can confirm that the device is owned by the network and is on the list of devices authorized to be onboarded.

6. A secure (i.e., encrypted) channel is established between the network onboarding component and the device.

7. The device sends its application-layer bootstrapping information to the network onboarding component. Just as the network required the trusted introduction of device network-layer bootstrapping information in order to enable the network to authenticate the device and ensure that the device was authorized to be network-layer onboarded, the application server requires the trusted introduction of device application-layer bootstrapping information to enable the application server to authenticate the device at the application layer and ensure that the device is authorized to be application-layer onboarded. Because this application-layer bootstrapping information is being sent over a secure channel, its integrity and confidentiality are ensured.

8. The network onboarding component forwards the device's application-layer bootstrapping information to the application server. In response, the application server provides its application-layer bootstrapping information to the network-layer onboarding component for eventual forwarding to the IoT device. The IoT device needs the application server's bootstrapping information to enable the device to authenticate the application server and ensure that it is authorized to application-layer onboard the device.

9. The network onboarding component uses the secure channel that it has established with the IoT device to confidentially send the device its unique network credentials. Along with these network credentials, the network onboarding component also sends the IoT device the application server's bootstrapping information. Because the application server's bootstrapping information is being sent over a secure channel, its integrity and confidentiality are ensured.z

10. The device uses its newly provisioned network credentials to establish secure connectivity to the network.

11. Using the device and application server application-layer bootstrapping information that has already been exchanged in a trusted manner, the application server authenticates the identity of the IoT device and the IoT device authenticates the identity of the application server. Then they establish a secure (i.e., encrypted) channel.

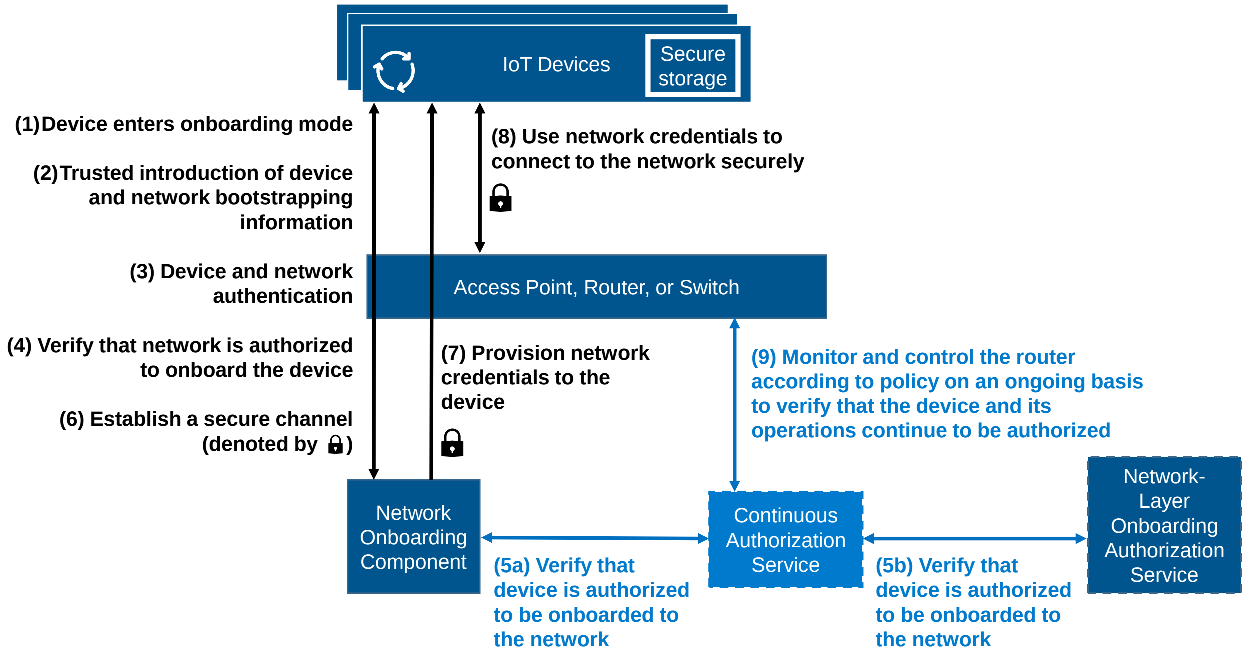
12. The application server application layer onboards the IoT device. This application-layer onboarding process may take a variety of forms. For example, the application server may download an application to the device for the device to execute. It may associate the device

with a trusted lifecycle management service that performs ongoing updates of the IoT device to patch it as needed to ensure that the device remains compliant with policy.

4.5 Continuous Verification

Figure 4-6 depicts the steps that are performed to support continuous verification. The figure uses two colors. The light-blue component and its accompanying steps (written in light-blue font) depict the portion of the diagram that is specific to continuous authorization. The dark-blue components are those used in the network-layer onboarding process. They and their accompanying steps (written in black font) are identical to those found in the trusted network-layer onboarding process diagram provided in Figure 4-4, except for step 5, *Verify that device is authorized to be onboarded to the network*.

Figure 4-6 Continuous Verification


IoT Devices
Secure storage
(1)Device enters onboarding mode
(2)Trusted introduction of device and network bootstrapping information
(3) Device and network authentication
(4) Verify that network is authorized to onboard the device
(6) Establish a secure channel (denoted by 🔒)
(8) Use network credentials to connect to the network securely
Access Point, Router, or Switch
(7) Provision network credentials to the device
(9) Monitor and control the router according to policy on an ongoing basis to verify that the device and its operations continue to be authorized
Network Onboarding Component
Continuous Authorization Service
Network-Layer Onboarding Authorization Service
(5a) Verify that device is authorized to be onboarded to the network
(5b) Verify that device is authorized to be onboarded to the network

When continuous verification is being supported, step 5 is broken into two separate steps, as shown in Figure 4-6. Instead of the network onboarding component directly contacting the network-layer onboarding authorization service to see if the device is owned by the network and on the list of devices authorized to be onboarded (as shown in the trusted network-layer onboarding architecture depicted in Figure 4-4), a set of other enterprise policies may also be applied to determine if the device is authorized to be onboarded. The application of these policies is represented by the insertion of the Continuous Authorization Service (CAS) component in the middle of the exchange between the network onboarding component and the network-layer onboarding authorization service.

For example, the CAS may have received external threat information indicating that certain device types have a vulnerability. If so, when the CAS receives a request from the network-layer onboarding component to verify that a device of this type is authorized to be onboarded to the network (Step 5a), it would immediately respond to the network-layer onboarding component that the device is not authorized to be onboarded to the network. If the CAS has not received any such threat information about the device and it checks all its policies and determines that the device should be permitted to be onboarded, it will forward the request to the network-layer onboarding authorization service (Step 5b) and receive a response (Step 5b) that it will forward to the network onboarding component (Step 5a).

As depicted by Step 9, the CAS also continues to operate after the device connects to the network and executes its application. The CAS performs asynchronous calls to the network router to monitor the device on an ongoing basis, providing policy-based verification and authorization checks on the device throughout its lifecycle.

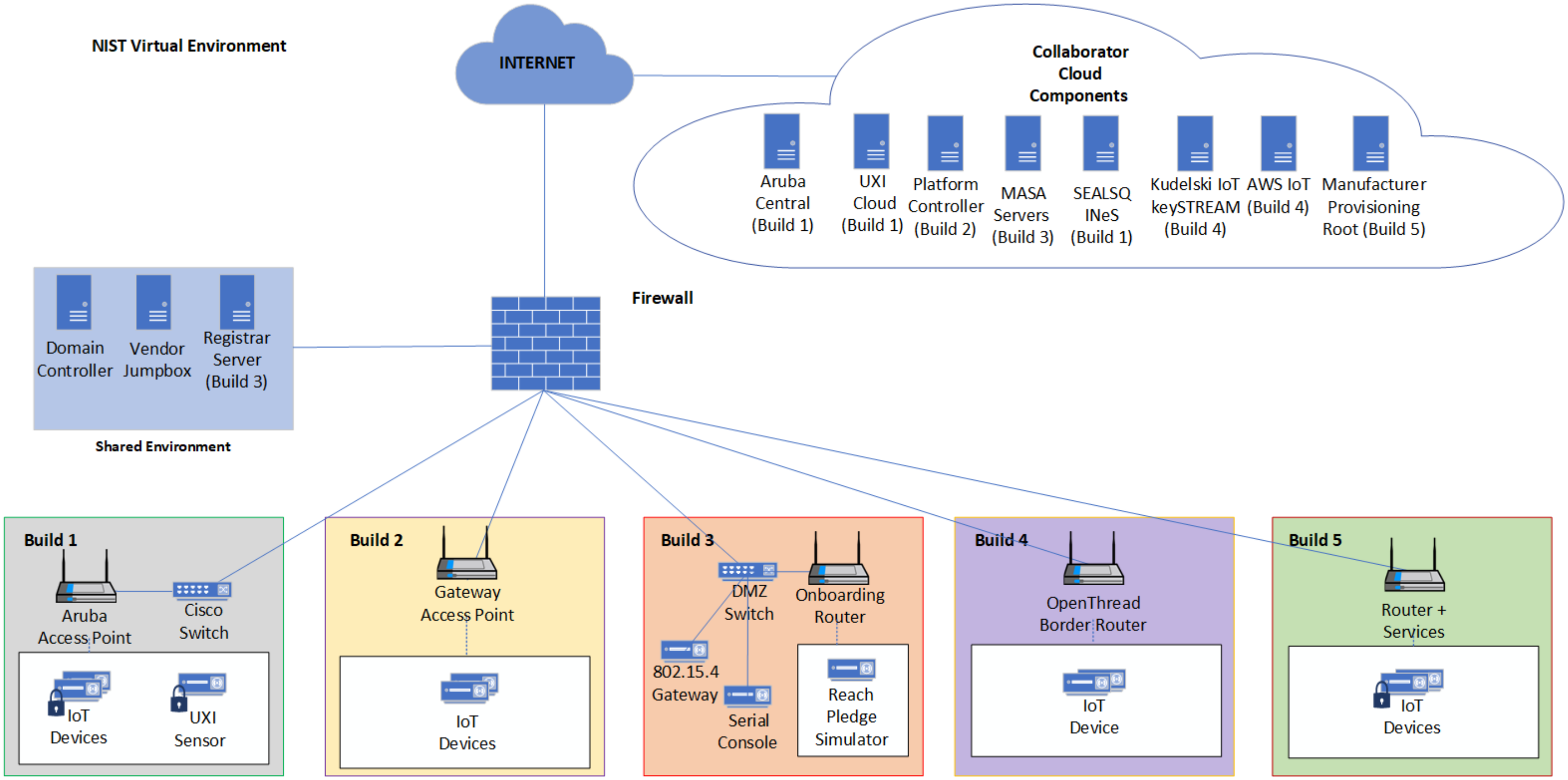
5 Laboratory Physical Architecture

Figure 5-1 depicts the high-level physical architecture of the NCCoE IoT Onboarding laboratory environment in which the five trusted IoT device network-layer onboarding project builds, and the factory provisioning builds are being implemented. The NCCoE provides virtual machine (VM) resources and physical infrastructure for the IoT Onboarding lab. As depicted, the NCCoE IoT Onboarding laboratory hosts collaborator hardware and software for the builds. The NCCoE also provides connectivity from the IoT Onboarding lab to the NIST Data Center, which provides connectivity to the internet and public IP spaces (both IPv4 and IPv6). Access to and from the NCCoE network is protected by a firewall.

Access to and from the IoT Onboarding lab is protected by a pfSense firewall, represented by the brick box icon in Figure 5-1. This firewall has both IPv4 and IPv6 (dual stack) configured. The IoT Onboarding lab network infrastructure includes a shared virtual environment that houses a domain controller and a vendor jumpbox. These components are used across builds where applicable. It also contains five independent virtual LANs, each of which houses a different trusted network-layer onboarding build.

The IoT Onboarding laboratory network has access to cloud components and services provided by the collaborators, all of which are available via the internet. These components and services include Aruba Central and the UXI Cloud (Build 1), SEALSQ INeS (Build 1), Platform Controller (Build 2), a MASA server (Build 3), Kudelski IoT keySTREAM application-layer onboarding service and AWS IoT (Build 4), and a Manufacturer Provisioning Root (Build 5).

1435 **Figure 5-1 NCCoE IoT Onboarding Laboratory Physical Architecture**



All five network-layer onboarding laboratory environments, as depicted in the diagram, have been installed:

- The Build 1 (i.e., the Wi-Fi Easy Connect, Aruba/HPE build) network infrastructure within the NCCoE lab consists of two components: the Aruba Access Point and the Cisco Switch. Build 1 also requires support from Aruba Central for network-layer onboarding and the UXI Cloud for application-layer onboarding. These components are in the cloud and accessed via the internet. The IoT devices that are onboarded using Build 1 include the UXI Sensor and the Raspberry Pi.
- The Build 2 (i.e., the Wi-Fi Easy Connect, CableLabs, OCF build) network infrastructure within the NCCoE lab consists of a single component: the Gateway Access Point. Build 2 requires support from the Platform Controller, which also hosts the IoTivity Cloud Service. The IoT devices that are onboarded using Build 2 include three Raspberry Pis.
- The Build 3 (i.e., the BRSKI, Sandelman Software Works build) network infrastructure components within the NCCoE lab include a Wi-Fi capable home router (including Join Proxy), a DMZ switch (for management), and an ESP32A Xtensa board acting as a Wi-Fi IoT device, as well as an nRF52840 board acting as an IEEE 802.15.4 device. A management system on a BeagleBone Green serves as a serial console. A registrar server has been deployed as a virtual appliance on the NCCoE private cloud system. Build 3 also requires support from a MASA server which is accessed via the internet. In addition, a Raspberry Pi 3 provides an ethernet/802.15.4 gateway, as well as a test platform.
- The Build 4 (i.e., the Thread, Silicon Labs, Kudelski IoT build) network infrastructure components within the NCCoE lab include an Open Thread Border Router, which is implemented using a Raspberry Pi, and a Silicon Labs Gecko Wireless Starter Kit, which acts as an 802.15.4 antenna. Build 4 also requires support from the Kudelski IoT keySTREAM service, which is in the cloud and accessed via the internet. The IoT device that is onboarded in Build 4 is the Silicon Labs Dev Kit (BRD2601A) with an EFR32MG24 System-on-Chip (SoC). The application service to which it onboards is AWS IoT.
- The Build 5 (i.e., the BRSKI over Wi-Fi, NquiringMinds build) includes 2 Raspberry Pi 4Bs running a Linux operating system. One Raspberry Pi acts as the pledge (or IoT Device) with an Infineon TPM connected. The other acts as the router, registrar and MASA all in one device. This build uses the open source TrustNetZ distribution, from which the entire build can be replicated easily. The TrustNetZ distribution includes source code for the IoT device, the router, the access point, the network onboarding component, the policy engine, the manufacturer services, the registrar and a demo application server. TrustNetZ makes use of NquiringMinds tdx Volt to issue and validate verifiable credentials.
- The BRSKI factory provisioning build is deployed in the Build 5 environment. The IoT device in this build is a Raspberry Pi equipped with an Infineon Optiga SLB 9670 TPM 2.0, which gets provisioned with birth credentials (i.e., a public/private key pair and an IDevID). The BRSKI factory provisioning build also uses an external certificate authority hosted on the premises of NquiringMinds to provide the device certificate signing service.
- The Wi-Fi Easy Connect factory provisioning build is deployed in the Build 1 environment. Its IoT devices are Raspberry Pis equipped with a SEALSQ VaultIC Secure Element, which gets provisioned with a DPP URI. The Secure Element can also be provisioned with an IDevID certificate signed by the SEALSQ INeS certification authority, which is independent of the DPP URI. Code for performing the factory provisioning is stored on an SD card.

Information regarding the physical architecture of all builds, their related collaborators' cloud components, and the shared environment, as well as the baseline software running on these physical architectures, are described in the subsections below. Table 5-1 summarizes the builds that were implemented and provides links to the appendices where each is described in detail.

Table 5-1 Build 1 Products and Technologies

Build	Network-Layer Protocols	Build Champions	Link to Details
Onboarding Builds			
Build 1	Wi-Fi Easy Connect	Aruba/HPE	Appendix C
Build 2	Wi-Fi Easy Connect	CableLabs and OCF	Appendix D
Build 3	BRSKI	Sandelman Software Works	Appendix E
Build 4	Thread	Silicon Labs and Kudelski IoT	Appendix F
Build 5	BRSKI over Wi-Fi	NquiringMinds	Appendix G
Factory Provisioning Builds			
BRSKI with Build 5	BRSKI over WIFI	SEALSQ and NquiringMinds	Appendix H.3
Wi-Fi Easy Connect with Build 1	Wi-Fi Easy Connect	SEALSQ and Aruba/HPE	Appendix H.4

5.1 Shared Environment

The NCCoE IoT Onboarding laboratory contains a shared environment to host several baseline services in support of the builds. These baseline services supported configuration and integration work in each of the builds and allowed collaborators to work together throughout the build process. This shared environment is contained in its own network segment, with access to/from the rest of the lab environment closely controlled. In addition, each of the systems in the shared environment is hardened with baseline configurations.

5.1.1 Domain Controller

The Domain Controller provides Active Directory and Domain Name System (DNS) services supporting network access and access control in the lab. It runs on Windows Server 2019.

5.1.2 Jumpbox

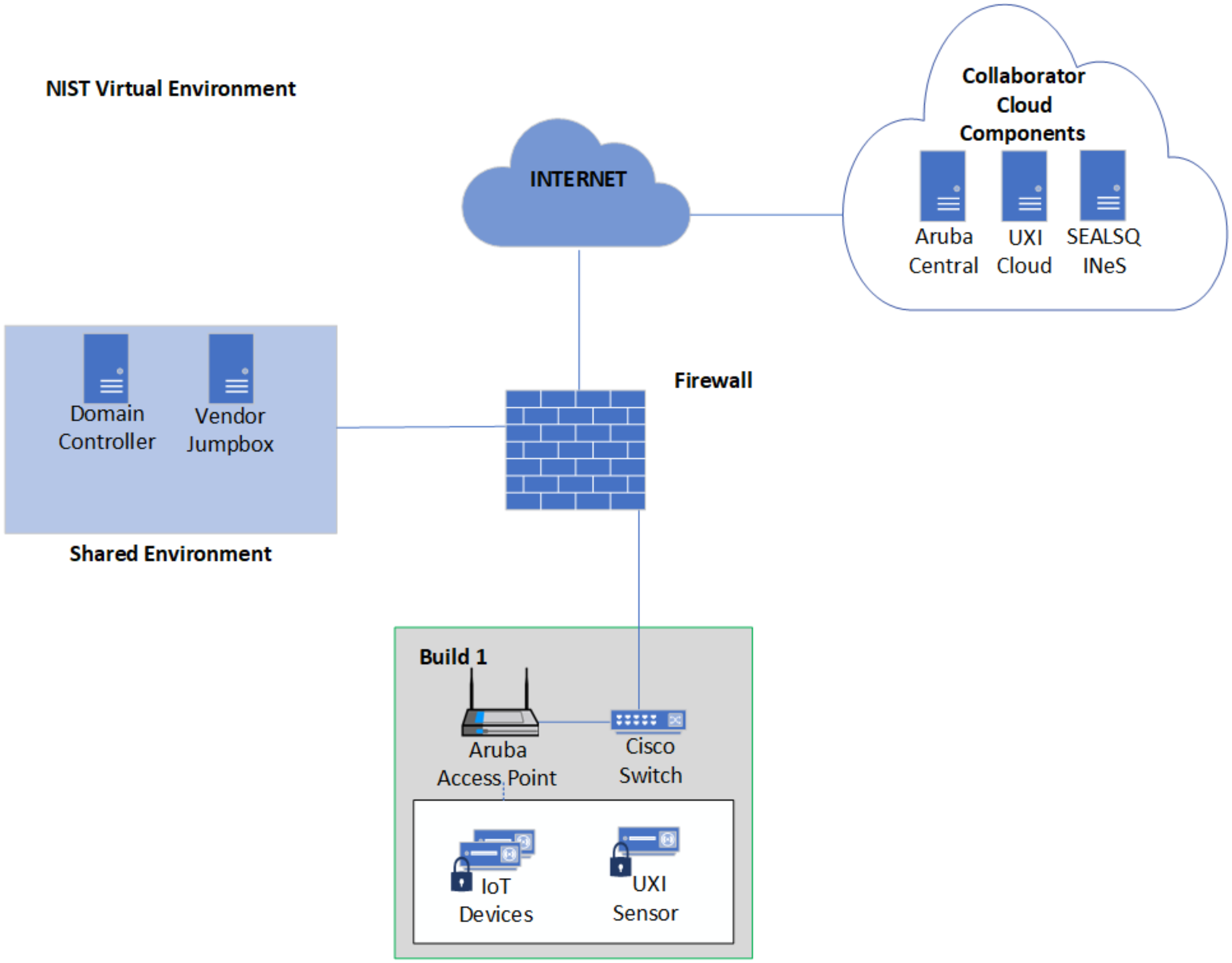
The jumpbox provides secure remote access and management to authorized collaborators on each of the builds. It runs on Windows Server 2019.

5.2 Build 1 (Wi-Fi Easy Connect, Aruba/HPE) Physical Architecture

Figure 5-2 is a view of the high-level physical architecture of Build 1 in the NCCoE IoT Onboarding laboratory. The build components include an Aruba Wireless Access Point, Aruba Central, UXI Cloud, a Cisco Catalyst switch, a SEALSQ INeS CMS CA, and the IoT devices to be onboarded, which include both a Raspberry Pi and a UXI sensor. Most of these components are described in Section 3.4.1 and Section 3.4.3.

- The Aruba Access Point acts as the DPP Configurator and relies on the Aruba Central cloud service for authentication and management purposes.
- Aruba Central ties together the IoT Operations, Client Insights, and Cloud Auth services to support the network-layer onboarding operations of the build. It also provides an API to support the device ownership and bootstrapping information transfer process.
- The Cisco Catalyst Switch provides Power-over-Ethernet and network connectivity to the Aruba Access Point.
- The UXI Sensor acts as an IoT device and onboards to the network via Wi-Fi Easy Connect. After network-layer onboarding, it performs independent (see Section 3.3.2) application-layer onboarding. Once it has application-layer onboarded and is operational on the network, it does passive and active monitoring of applications and services and will report outages, disruptions, and quality of service issues.
- UXI Cloud is an HPE cloud service that the UXI sensor contacts as part of the application-layer onboarding process. The UXI sensor downloads a customer-specific configuration from the UXI Cloud so that the UXI sensor can learn about the customer networks and services it needs to monitor.
- The Raspberry Pi acts as an IoT device and onboards to the network via Wi-Fi Easy Connect.
- SEALSQ Certificate Authority has been integrated with Build 1 to sign network credentials that are issued to IoT devices.

Figure 5-2 Physical Architecture of Build 1

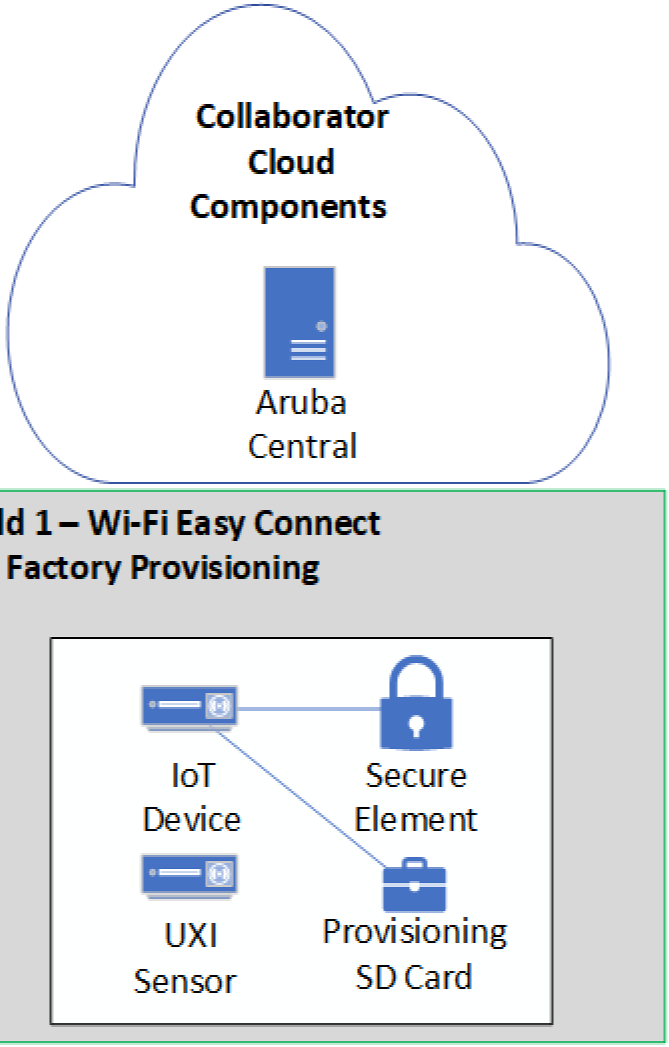


5.2.1 Wi-Fi Easy Connect Factory Provisioning Build Physical Architecture

Figure 5-3 is a view of the high-level physical architecture of the Wi-Fi Easy Connect Factory Provisioning Build in the NCCoE IoT Onboarding laboratory. The build components include the IoT device, an SD card with factory provisioning code on it, and a Secure Element. See Appendix H.4 for additional details on the Wi-Fi Easy Connect Factory Provisioning Build.

- A UXI sensor.
- The IoT Device is a Raspberry Pi.
- The Secure Element is a SEALSQ VaultIC Secure Element and is interfaced with the Raspberry Pi. The Secure Element both generates and stores the key material necessary to support the DPP URI during the Factory Provisioning Process.
- An SD card with factory provisioning code.
- Aruba Central provides an API to ingest the DPP URI in support of the device ownership and bootstrapping information transfer process.

Figure 5-3 Physical Architecture of Wi-Fi Easy Connect Factory Provisioning Build

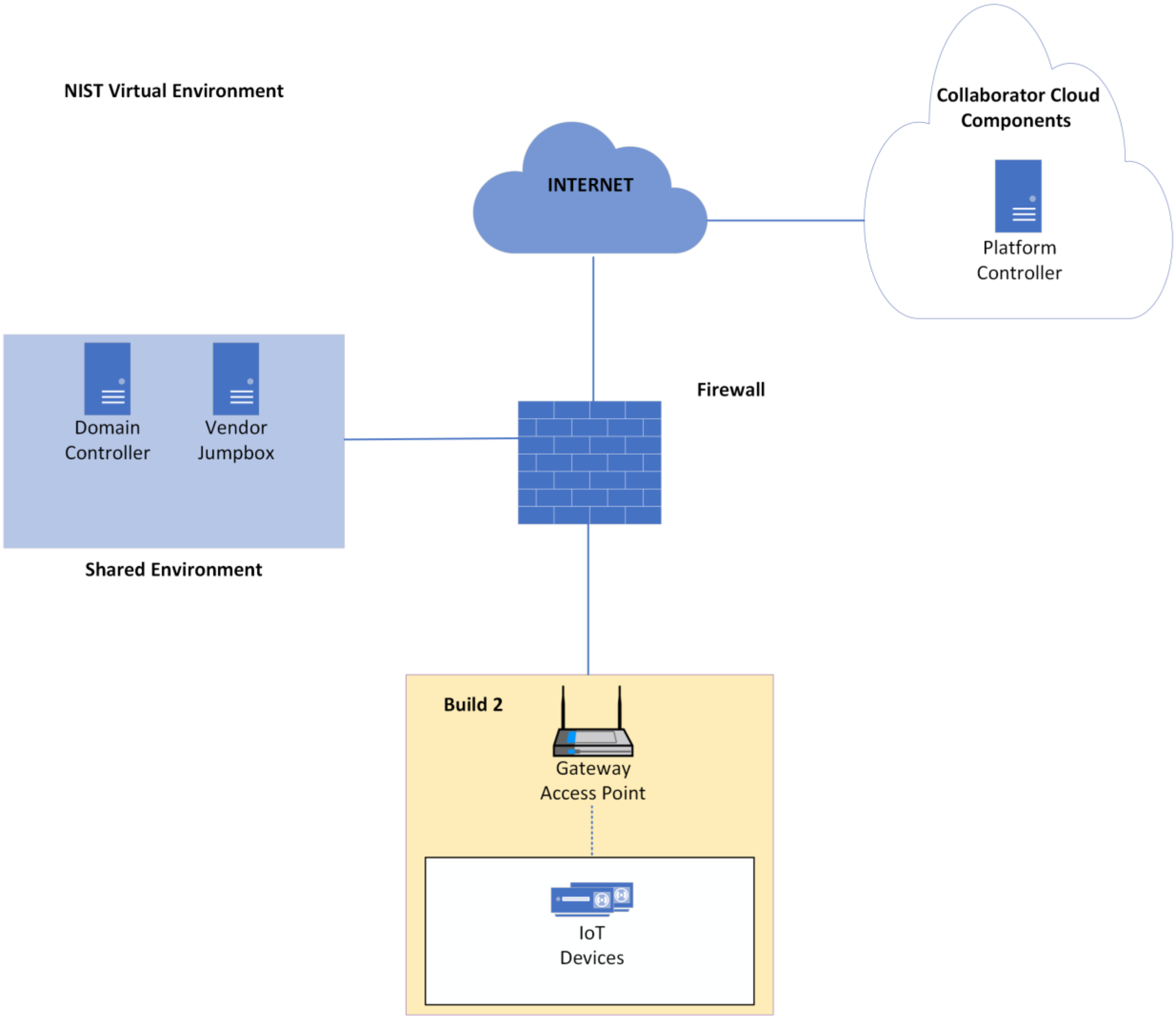

Collaborator Cloud Components
Aruba Central
Build 1 – Wi-Fi Easy Connect Factory Provisioning
IoT Device
Secure Element
UXI Sensor
Provisioning SD Card

5.3 Build 2 (Wi-Fi Easy Connect, CableLabs, OCF) Physical Architecture

Figure 5-3 is a view of the high-level physical architecture of Build 2 in the NCCoE IoT Onboarding laboratory. The Build 2 components include the Gateway Access Point, three IoT devices, and the Platform Controller, which hosts the application-layer IoTivity service.

- The Gateway Access Point acts as the Custom Connectivity Gateway Agent described in Section 3.4.2.2 and controls all network-layer onboarding activity within the network. It also hosts OCF IoTivity functions, such as the OCF OBT and the OCF Diplomat.
- The Platform Controller described in Section 3.4.2.1 provides management capabilities for the Custom Connectivity Gateway Agent. It also hosts the application-layer IoTivity service for the IoT devices as described in Section 3.4.8.1.
- The IoT devices serve as reference clients, as described in Section 3.4.2.3. They run OCF reference implementations. The IoT devices are onboarded to the network and complete both application-layer and network-layer onboarding.

Figure 5-4 Physical Architecture of Build 2



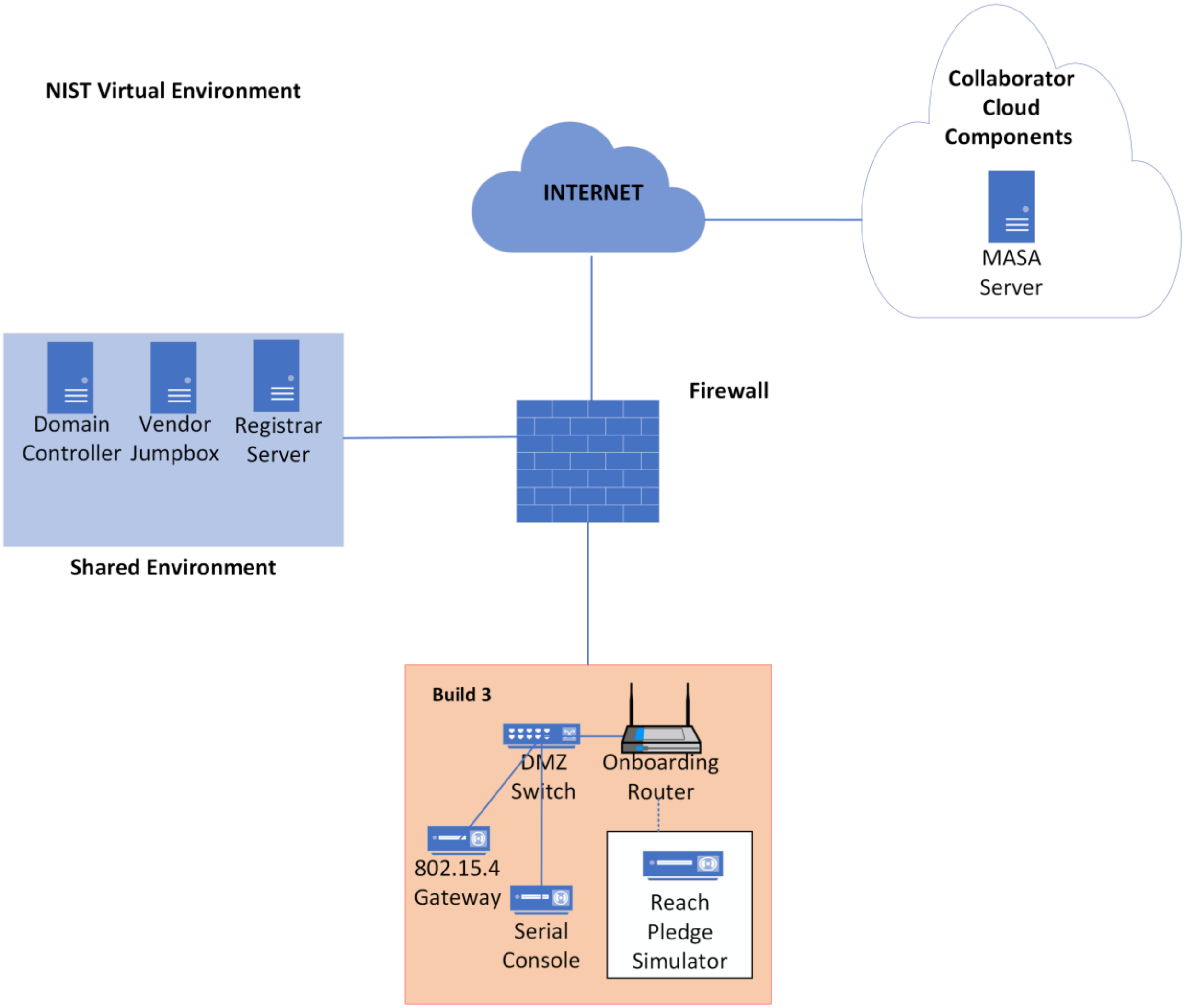
5.4 Build 3 (BRSKI, Sandelman Software Works) Physical Architecture

Figure 5-4 is a view of the high-level physical architecture of Build 3 in the NCCoE IoT Onboarding laboratory. The Build 3 components include the onboarding router, a Registrar Server, a MASA server, a DMZ switch, IoT devices, a serial console, and an 802.15.4 gateway.

- The onboarding router is a Turris MOX router running OpenWRT. The onboarding router quarantines the IoT devices until they complete the BRSKI onboarding process.
- The owner's Registrar Server hosts the Minerva Fountain Join Registrar Coordinator application running in a virtual machine. The Registrar Server determines whether or not a device meets the criteria to join the network.
- The MASA server for this build is a Minerva Highway MASA server as outlined in Section 3.4.9.1. The role of the MASA server is to receive the voucher-request from the Registrar Server and confirm that the Registrar Server has the right to own the device.

- The DMZ switch is a basic Netgear switch that segments the build from the rest of the lab.
- The IoT devices include an ESP32 Xtensa device with Wi-Fi that will be tested with FreeRTOS and RIOT-OS, a Raspberry Pi 3 running Raspbian 11, and an nRF52840 with an 802.15.4 radio that is running RIOT-OS. The IoT devices are currently not used in the build but will serve as clients to be onboarded onto the network in a future implementation of the build.
- The Sandelman Software Works Reach Pledge Simulator is the device that is onboarded to the network in the current build.
- The serial console is a BeagleBone Green with an attached USB hub. The serial console is used to access the IoT devices for diagnostic purposes. It also provides power and power control for USB-powered devices.
- The 802.15.4 gateway is integrated into the Raspberry Pi 3 via an OpenMote daughter card. This gateway will serve to onboard one of the IoT devices in a future implementation of this build.

Figure 5-5 Physical Architecture of Build 3

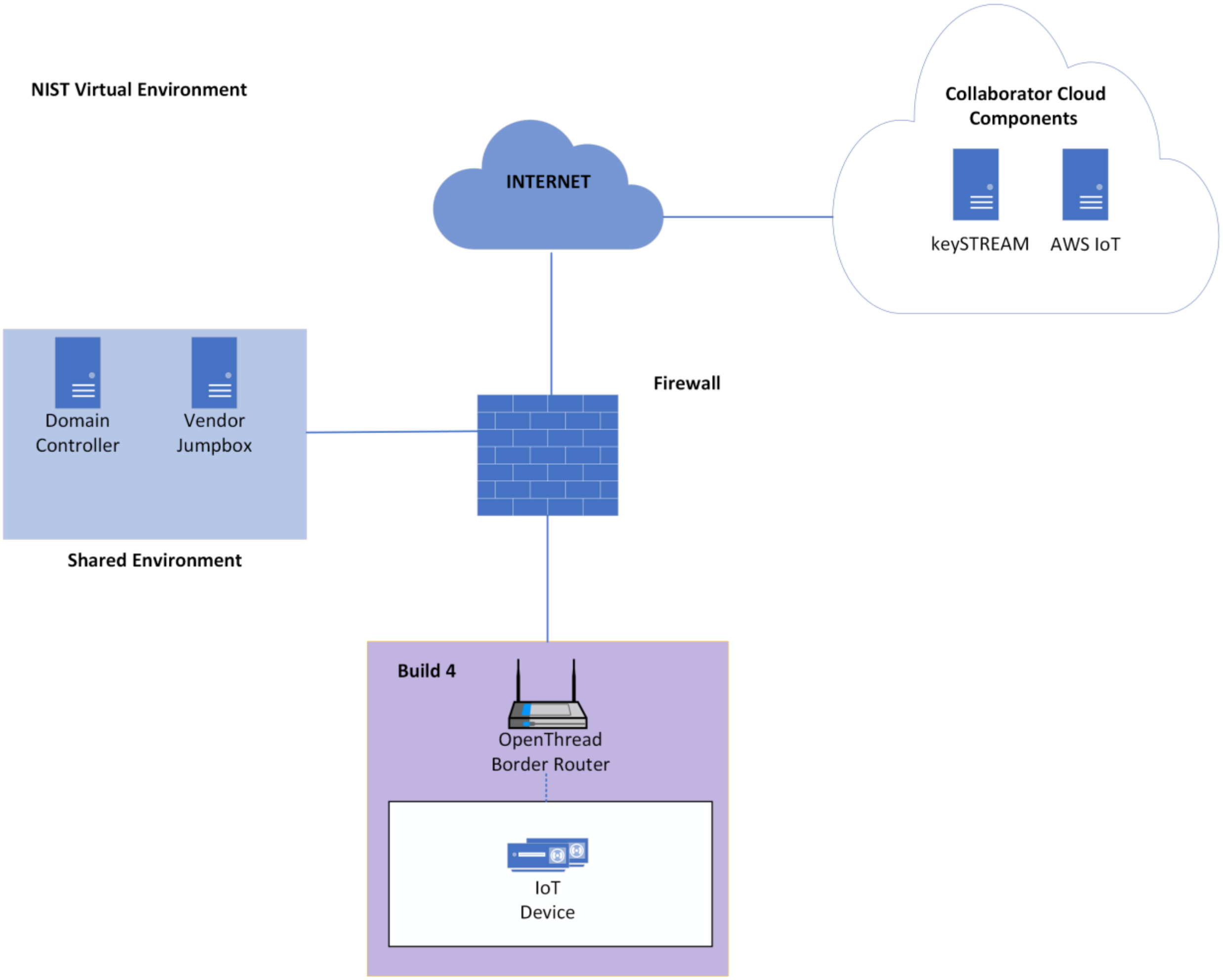


5.5 Build 4 (Thread, Silicon Labs, Kudelski IoT) Physical Architecture

Figure 5-6 is a view of the high-level physical architecture of Build 4 in the NCCoE IoT Onboarding laboratory. The Build 4 components include a keySTREAM server, an AWS IoT server, an OpenThread Border Router, and a Thread IoT device.

- The keySTREAM server described in Section 3.4.5.1 is the application layer onboarding service provided by Kudelski IoT. The IoT device will authenticate to keySTREAM using a Silicon Labs chip birth certificate and private key and leveraging Silicon Labs' Secure Engine in the EFR32MG24 chipset ("Secure Vault(TM) High" which is security certified Platform Security Architecture (PSA)/Security Evaluation Standard for IoT Platforms (SESIP) Level 3 to protect that birth identity with Secure Boot, Secure Debug, and physically unclonable function (PUF) wrapped key storage and hardware tamper protection).
- The AWS IoT server provides the MQTT test client for the trusted application-layer onboarding. The Proof of Possession Certificate is provisioned for the device using a registration code from the AWS server.
- The OpenThread Border Router is run on a Raspberry Pi 3B and serves as the Thread Commissioner and Leader. It communicates with the IoT device by means of a Silicon Labs Gecko Wireless Devkit which serves as the 802.15.4 antenna for the build.
- The IoT Device in this build is a Silicon Labs Thunderboard (BRD2601A) containing the EFR32MG24Bx 15.4 SoC with Secure Vault (TM) High running the Thread protocol. It serves as the child node on the Thread network and is onboarded onto AWS IoT Core using credentials provisioned from the Kudelski keySTREAM service.

Figure 5-6 Physical Architecture of Build 4



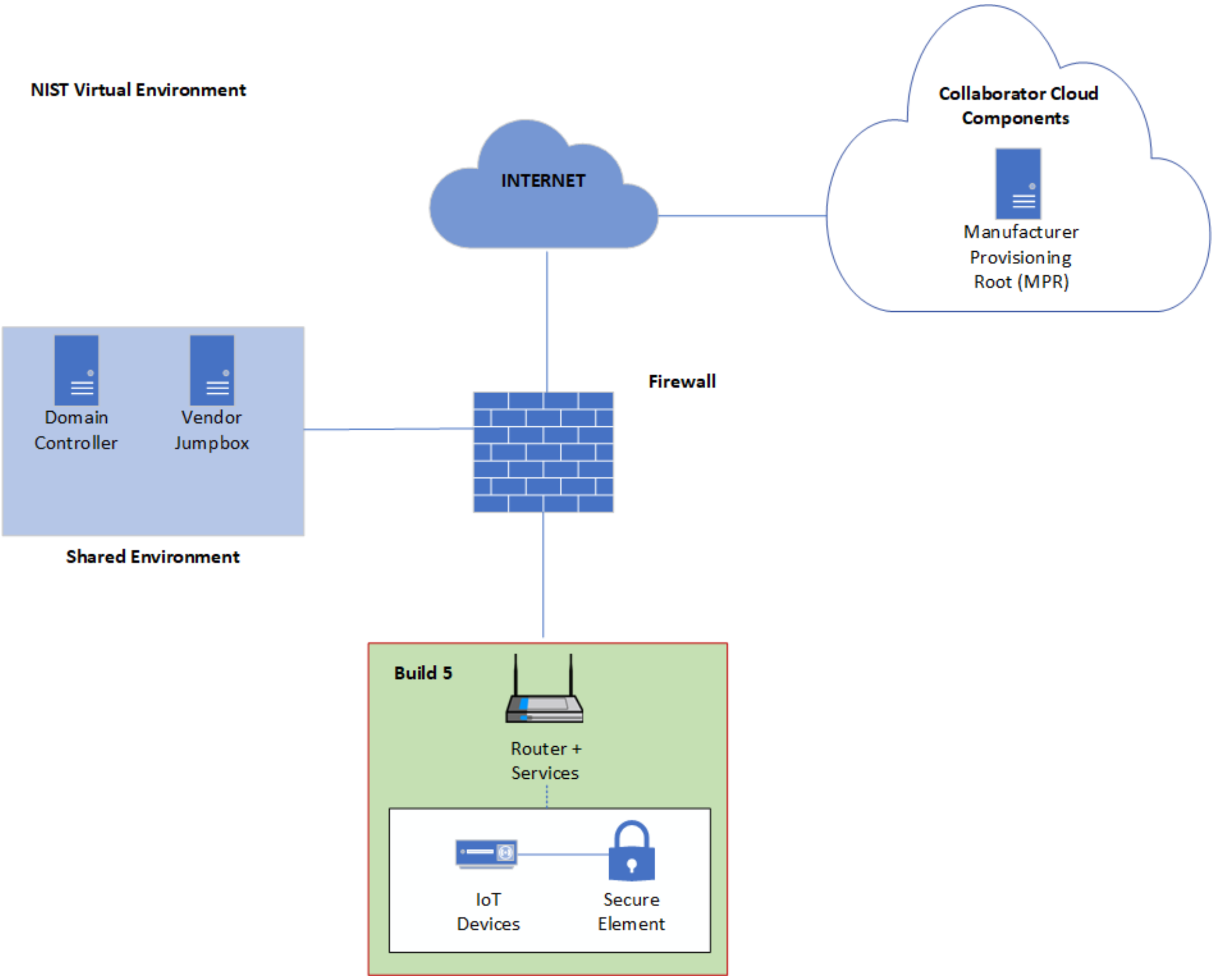
5.6 Build 5 (BRSKI, NquiringMinds) Physical Architecture

Figure 5-6 is a view of the high-level physical architecture of Build 5 in the NCCoE IoT Onboarding laboratory. The Build 5 components include a MASA, Registrar, Router Access Point, an IoT Device, and a Secure Element:

- A Raspberry Pi 4B serves as the MASA, Registrar and Router Access Point for the local network. The role of the MASA is to receive the voucher-request from the Registrar and confirm that the Registrar has the right to own the device. The registrar self-signs credentials, namely the Local Device Identifier (LDevID), issued to the IoT devices. The pledge (IoT device) gets its IDevID certificate for device identity from the Manufacturer Provisioning Root (MPR) server during the factory provisioning process, it can be assumed to be present on the device at the point of onboarding. The Registrar determines whether or not a device meets the criteria to join the network. The router access point runs an open and closed BRSKI network, the closed BRSKI network may only be accessed through secure onboarding, which is performed via the open network. The registrar leverages a local tdx Volt instance to sign and verify verifiable credentials.
- Raspberry Pi 4Bs act as IoT Devices (pledges) for this build.

- The Secure Element is an Infineon Optiga SLB 9670 TPM 2.0 Secure Element, and both generates and stores the key material necessary to support the IDevID certificate during the Factory Provisioning Process, as well as the onboarding process to request the voucher from the MASA via the registrar and the request to the registrar to sign the LDevID. The system can also be configured to use a SEALSQ VaultIC408 secure element. See Appendix H.3 for additional details on the BRSKI factory provisioning builds.

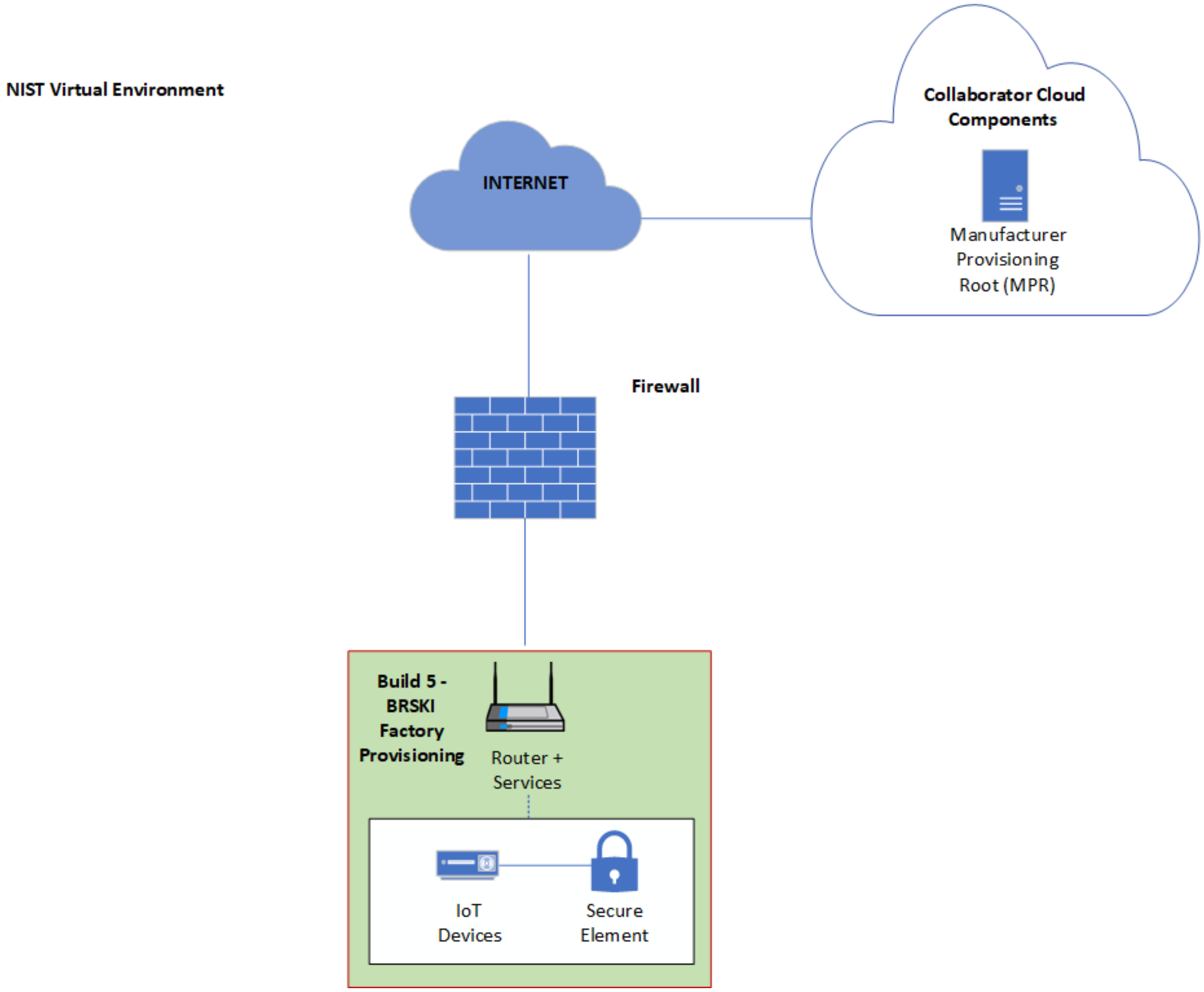
Figure 5-7 Physical Architecture of Build 5



5.6.1 BRSKI Factory Provisioning Build Physical Architecture

Figure 5-8 is a view of the high-level physical architecture of the BRSKI Factory Provisioning Build in the NCCoE IoT Onboarding laboratory. This build uses the same IoT device as Build 5: a Raspberry Pi integrated with an Infineon Optiga SLB 9670 TPM 2.0 Secure Element. The factory provisioning code is hosted on an SD card. When a provisioning event is triggered the IoT device will attempt a connection to a Manufacturer Provisioning Root (MPR) server that sits in the cloud and acts as the certification authority. It signs the IDevID (X.509) certificate, which is then passed back to the IoT device for installation. As in Build 5, the Router + Services hosts a MASA, which is given device identity information in order to verify voucher requests during the BRKSI process. See Appendix H.3 for additional details on the BRSKI factory provisioning builds.

Figure 5-8 Physical Architecture of BRSKI Factory Provisioning Build


NIST Virtual Environment
INTERNET
Collaborator Cloud Components
Manufacturer Provisioning Root (MPR)
Firewall
Build 5 - BRSKI Factory Provisioning
Router + Services
IoT Devices
Secure Element

6 General Findings

6.1 Wi-Fi Easy Connect

The Wi-Fi Easy Connect solution that was demonstrated in Build 1 and Build 2 supports trusted network-layer onboarding in a manner that is secure, efficient, and flexible enough to meet the needs of various use cases. It is simple enough to be used by consumers, who typically do not have specialized technical knowledge. In addition, to meet the needs of enterprises, it may be used to onboard a large number of devices quickly. Builds 1 and 2 are implementations of this protocol, and they are interoperable: IoT devices that were provisioned for use with Build 1 were able to be onboarded onto the network using Build 2, and IoT devices that were provisioned for use with Build 2 were able to be onboarded onto the network using Build 1.

6.1.1 Mutual Authentication

Although DPP is designed to support authentication of the network by the IoT device as well as authentication of the device by the network, the Wi-Fi Easy Connect solutions that were demonstrated in builds 1 and 2 do not demonstrate mutual authentication at the network layer. They only support authentication of the device. In order to authenticate the network, the device needs to be provided with the DPP URI for the network configurator, which means that the device has to have a functional user interface so that the DPP URI can be input into it. The devices being used in builds 1 and 2 do not have user interfaces.

6.1.2 Mutual Authorization

When using DPP, device authorization is based on possession of the device's DPP URI. When the device is acquired, its DPP URI is provided to the device owner. A trusted administrator of the owner's network is assumed to approve addition of the device's DPP URI to the database or cloud service where the DPP URIs of authorized devices are stored. During the onboarding process, the fact that the owning network is in possession of the device's DPP URI indicates to the network that the device is authorized to join it.

DPP supports network authorization using the Resurrecting Duckling security model [13]. Although the device cannot cryptographically verify that the network is authorized to onboard it, the fact that the network possesses the device's public key is understood by the device to implicitly authorize the network to onboard the device. The assumption is that an unauthorized network would not have possession of the device and so would not be able to obtain the device's public key. While this assurance of authorization is not cryptographic, it does provide some level of assurance that the "wrong" network won't onboard it.

6.1.3 Secure Storage

The UXI sensor used in Build 1 has a TPM where the device's birth credential and private key are stored, providing a secure root of trust. However, the lack of secure storage on some of the other IoT devices (e.g., the Raspberry Pis) used to demonstrate onboarding in Build 2 is a current weakness. Ensuring that the confidentiality of a device's birth, network, and other credentials is protected while stored on the device is an essential aspect of ensuring the security of the network-layer onboarding process, the device, and the network itself. To fully demonstrate trusted network-layer onboarding, devices with secure storage should be used in the future whenever possible.

6.2 BRSKI

The BRSKI solution that is demonstrated in Build 3 supports trusted network-layer onboarding in a manner that is secure, efficient, and able to meet the needs of enterprises. It may be used to onboard a large number of devices quickly onto a wired network. This BRSKI build is based on IETF RFC 8995 [7]. The build has a reliance on the manufacturer to provision keys for the onboarding device and has a reliance on a cloud-based service for the MASA server. The BRSKI solution that is demonstrated in Build 5 provides similar trusted functionality for onboarding devices onto a Wi-Fi network. This BRSKI build is based on an IETF individual draft describing how to run BRSKI over IEEE 802.11 [10].

6.2.1 Reliance on the Device Manufacturer

Organizations implementing BRSKI (whether wired or over Wi-Fi) should be aware of the reliance that they will have on the IoT device manufacturer in properly and securely provisioning their devices. If keys become compromised, attackers may be able to onboard their own devices to the network, revoke certificates to prevent legitimate devices from being onboarded, or onboard devices belonging to others onto the attacker's network using the attacker's MASA. These concerns are addressed in depth in RFC 8995 section 11.6. If a device manufacturer goes out of business or otherwise shuts down their MASA servers, the onboarding services for their devices will no longer function.

During operation, onboarding services may become temporarily unavailable for a number of reasons. In the case of a DoS attack on the MASA, server maintenance, or other outage on the part of the manufacturer, an organization will not be able to access the MASA. These concerns are addressed in depth in RFC 8995 section 11.1.

6.2.2 Mutual Authentication

BRSKI supports authentication of the IoT device by the network as well as authentication of the network by the IoT device. The Registrar authenticates the device when it receives the IDevID from the device. The MASA confirms that the Registrar is the legitimate owner or authorized onboarder of the device and issues a voucher. The device is able to authenticate the network using the voucher that it receives back from the MASA. This process is explained in depth in RFC 8995 section 11.5.

6.2.3 Mutual Authorization

BRSKI authorization for the IoT device is done via the voucher that is returned to the Registrar from the MASA. The voucher states which network the IoT device is authorized to join. The Registrar determines the level of access the IoT device has to the network.

6.2.4 Secure Storage

Build 5 uses a Secure Element attached to the IoT devices (e.g., Raspberry Pi devices) to store the IDevID after it is generated during the factory provisioning process (see Appendix H.3 for more details), however the LDevID is not stored on the Secure Element after network-layer onboarding is completed. The lack of secure storage on the IoT devices (e.g., the Raspberry Pi devices) used to demonstrate onboarding in Build 3 is a current weakness. Ensuring that the confidentiality of a device's birth, network, and other credentials is protected while stored on the device is an essential aspect of ensuring the security of the network-layer onboarding process, the device, and the network itself. To fully demonstrate trusted network-layer onboarding, devices with secure storage should be used in the future whenever possible.

6.3 Thread

We do not have any findings with respect to trusted network-layer onboarding using the Thread commissioning protocol. Build 4 demonstrated the connection of an IoT device to a Thread network, but not trusted onboarding of the Thread network credentials to the device. In Build 4, a passphrase is generated on the IoT device and then a person is required to enter this passphrase into the OpenThread

Border Router's (OTBR) web interface. This passphrase serves as a pre-shared key that the device uses to join the Thread network. Due to the fact that a person must be privy to this passphrase in order to provide it to the OTBR, this network-layer onboarding process is not considered to be trusted, according to the definition of trusted network-layer onboarding that we provided in Section 1.2.

After connecting to the Thread network using the passphrase, the Build 4 device was successfully able to gain access to the public IP network via a border router. This enabled the IoT device that was communicating using the Thread wireless protocol to communicate with cloud services and use them to successfully perform trusted application-layer onboarding to the AWS IoT Core.

6.4 Application-Layer Onboarding

We successfully demonstrated both:

- streamlined application-layer onboarding (to the OCF security domain in Build 2) and
- independent application-layer onboarding (to the UXI cloud in Build 1 and to the AWS IoT Core using the Kudelski keySTREAM service in Build 4).

6.4.1 Independent Application-Layer Onboarding

Support for independent application-layer onboarding requires the device manufacturer to pre-provision the device with software to support application-layer onboarding to the specific application service (e.g., the UXI cloud or the AWS IoT Core) desired. The Kudelski keySTREAM service supports the application-layer onboarding provided in Build 4. KeySTREAM is a device security management service that runs as a SaaS platform on the Amazon cloud. Build 4 relies on an integration that has been performed between Silicon Labs and Kudelski keySTREAM. KeySTREAM has integrated software libraries with the Silicon Lab EFR32MG24 (MG24) IoT device's secure vault to enable the private signing key that is associated with an application-layer certificate to be stored into the secure vault using security controls that are available on the MG24. This integration ensures that application-layer credentials can be provisioned into the vault securely such that no key material is misused or exposed.

Because the device is prepared for application-layer onboarding on behalf of a specific, pre-defined customer in Build 4 and this ownership information is sealed into device firmware, the device is permanently identified as being owned by that customer.

6.4.2 Streamline Application-Layer Onboarding

Support for streamlined application-layer onboarding does not necessarily present such a burden on the device manufacturer to provision application-layer onboarding software and/or credentials to the device at manufacturing time. If desired, the manufacturer could pre-install application-layer bootstrapping information onto the device at manufacturing time, as must be done in the independent application-layer onboarding case. Alternatively, the device manufacturer may simply ensure that the device has the capability to generate one-time application-layer bootstrapping information at runtime and use the secure exchanges inherent in trusted network-layer onboarding to support application-layer onboarding.

7 Additional Build Considerations

The Trusted Internet of Things (IoT) Device Network-Layer Onboarding and Lifecycle Management project is now complete, so no additions or changes to the existing builds are planned as part of this project effort. As trusted network-layer onboarding is increasingly adopted, however, others may wish to continue implementation efforts to develop new build capabilities or enhance existing ones, so it is worth noting potential areas of further work. Various ways in which individual builds could be enhanced are noted in the appendices that detail each build's technologies and architectures. For example, some builds could be enhanced by the addition of architectural components that they have not yet implemented, such as secure device storage; the use of an independent, third-party certificate signing authority; support for network-layer onboarding using Thread MeshCoP; support for application-layer onboarding; and support (or enhanced support) for ongoing device authorization. In addition to adding components to support these capabilities, future work could potentially involve demonstration of application-layer onboarding using the FIDO Alliance's FIDO Device Onboard (FDO) specification and/or the Connectivity Standards Alliance (CSA) MATTER specification. Other future work could involve integrating additional security mechanisms with network-layer onboarding, beginning at device boot-up and extending through all phases of the device lifecycle, to further protect the device and, by extension, the network. For example, future builds could include the capability to demonstrate the integration of trusted network-layer onboarding with zero trust-inspired capabilities such as those described in the following subsections. In addition, the scope of implementation efforts could potentially be expanded beyond the current focus on IP-based networks. While this project's goal has been to tackle what is currently implementable, the subsections that follow briefly discuss areas that could potentially be addressed by others in the future.

7.1 Network Authentication

Future builds could be designed to demonstrate network authentication in addition to device authentication as part of the network-layer onboarding process. Network authentication enables the device to verify the identity of the network that will be taking control of it prior to permitting itself to be onboarded.

7.2 Device Communications Intent

Future builds could be designed to demonstrate the use of network-layer onboarding protocols to securely transmit device communications intent information from the device to the network (i.e., to transmit this information in encrypted form with integrity protections). Secure conveyance of device communications intent information, combined with enforcement of it, would enable the build to ensure that IoT devices are constrained to sending and receiving only those communications that are explicitly required for each device to fulfill its purpose. Build 5 currently enforces device communications intent as part of its continuous assurance process. Build 5 determines device communications intent information (e.g., the device's MUD file URL) based on device type rather than conveying this information from the device to the network during onboarding.

7.3 Network Segmentation

Future builds could demonstrate the ability of the onboarding network to dynamically assign each new device that is permitted to join the network to a specific subnetwork. The router may have multiple network segments configured to which an onboarded device may be dynamically assigned. The decision regarding which segment (subnetwork) to which to assign the device could potentially be based on the device's DHCP fingerprint, other markers of the device's type, or some indication of the device's trustworthiness, subject to organizational policy.

7.4 Integration with a Lifecycle Management Service

Future builds could demonstrate trusted network-layer onboarding of a device, followed by streamlined trusted application-layer onboarding of that device to a lifecycle management application service. Such a capability would ensure that, once connected to the local network, the IoT device would automatically and securely establish an association with a trusted lifecycle management service that is designed to keep the device updated and patched on an ongoing basis.

7.5 Network Credential Renewal

Some devices may be provisioned with network credentials that are X.509 certificates and that will, therefore, eventually expire. Future build efforts could explore and demonstrate potential ways of renewing such credentials without having to reprovision the credentials to the devices.

7.6 Integration with Supply Chain Management Tools

Future work could include definition of an open, scalable supply chain integration service that can provide additional assurance of device provenance and trustworthiness automatically as part of the onboarding process. The supply chain integration service could be integrated with the authorization service to ensure that only devices whose provenance meets specific criteria and that reach a threshold level of trustworthiness will be onboarded or authorized.

7.7 Attestation

Future builds could integrate device attestation capabilities with network-layer onboarding to ensure that only IoT devices that meet specific attestation criteria are permitted to be onboarded. In addition to considering the attestation of each device as a whole, future attestation work could also focus on attestation of individual device components, so that detailed attestation could be performed for each board, integrated circuit, and software program that comprises a device.

7.8 Mutual Attestation

Future builds could implement mutual attestation of the device and its application services. In one direction, device attestation could be used to enable a high-value application service to determine whether a device should be given permission to access it. In the other direction, attestation of the application service could be used to enable the device to determine whether it should give the application service permission to access and update the device.

7.9 Behavioral Analysis

Future builds could integrate artificial intelligence (AI) and machine learning (ML)-based tools that are designed to analyze device behavior to spot anomalies or other potential signs of compromise. Any device that is flagged as a potential threat by these tools could have its network credentials invalidated to effectively evict it from the network, be quarantined, or have its interaction with other devices restricted in some way.

7.10 Device Trustworthiness Scale

Future efforts could incorporate the concept of a device trustworthiness scale in which information regarding device capabilities, secure firmware updates, the existence (or not) of a secure element for private key protection, type and version of each of the software components that comprise the device, etc., would be used as input parameters to calculate each device's trustworthiness value. Calculating such a value would essentially provide the equivalent of a background check. A history for the device could be maintained, including information about whether it has ever been compromised, if it has a known vulnerability, etc. Such a trustworthiness value could be provided as an onboarding token or integrated into the authorization service so permission to onboard to the network, or to access certain resources once joined, could be granted or denied based on historical data and trustworthiness measures.

7.11 Resource Constrained Systems

At present, onboarding solutions for technologies such as Zigbee, Z-Wave, and BLE use their own proprietary mechanisms or depend on gateways. In the future, efforts could be expanded to include onboarding in highly resource-constrained systems and non-IP systems without using gateways. Future work could include trying to perform trusted onboarding in these smaller microcontroller-constrained spaces in a standardized way with the goal of bringing more commonality across various solutions without having to rely on IP gateways.

Appendix A List of Acronyms

AAA	Authentication, Authorization, and Accounting
ACL	Access Control List
AES	Advanced Encryption Standard
AI	Artificial Intelligence
AP	Access Point
API	Application Programming Interface
AWS	Amazon Web Services
BLE	Bluetooth Low Energy
BRSKI	Bootstrapping Remote Secure Key Infrastructure
BSS	Basic Service Set
CA	Certificate Authority
CAS	Continuous Authorization Service
CMS	Certificate Management System
CPU	Central Processing Unit
CRADA	Cooperative Research and Development Agreement
CRL	Certificate Revocation List
DHCP	Dynamic Host Configuration Protocol
DMZ	Demilitarized Zone
DNS	Domain Name System
DPP	Device Provisioning Protocol
DTLS	Datagram Transport Layer Security
ECC	Elliptic Curve Cryptography
ESP	(Aruba) Edge Services Platform
ESS	Extended Service Set
EST	Enrollment over Secure Transport
HPE	Hewlett Packard Enterprise
HSM	Hardware Security Module
HTTPS	Hypertext Transfer Protocol Secure

IDevID	Initial Device Identifier
IE	Information Element
IEC	International Electrotechnical Commission
IETF	Internet Engineering Task Force
IoT	Internet of Things
IP	Internet Protocol
IPsec	Internet Protocol Security
ISO	International Organization for Standardization
LAN	Local Area Network, Local Area Networking
LDevID	Local Device Identifier
LmP	Linux microPlatform
MASA	Manufacturer Authorized Signing Authority
MeshCoP	Thread Mesh Commissioning Protocol
ML	Machine Learning
mPKI	Managed Public Key Infrastructure
MUD	Manufacturer Usage Description
NAC	Network Access Control
NCCoE	National Cybersecurity Center of Excellence
NIST	National Institute of Standards and Technology
OBT	Onboarding Tool
OCF	Open Connectivity Foundation
OCSP	Online Certificate Status Protocol
OS	Operating System
OTA	Over the Air
OTBR	OpenThread Border Router
PKI	Public Key Infrastructure
PSK	Pre-Shared Key
R&D	Research & Development
RBAC	Role-Based Access Control

RCP	Radio Coprocessor
RESTful	Representational State Transfer
RFC	Request for Comments
RoT	Root of Trust
RSA	Rivest-Shamir-Adleman (public-key cryptosystem)
SaaS	Software as a Service
SE	Secure Element
SEF	Secure Element Factory
SoC	System-on-Chip
SP	Special Publication
SSID	Service Set Identifier
SSW	Sandelman Software Works
TCP	Transmission Control Protocol
TLS	Transport Layer Security
TOFU	Trust On First Use
TPM	Trusted Platform Module
URI	Uniform Resource Identifier
UXI	(Aruba) User Experience Insight
VM	Virtual Machine
WAN	Wide Area Network, Wide Area Networking
WFA	Wi-Fi Alliance
WPA2	Wi-Fi Protected Access 2
WPA3	Wi-Fi Protected Access 3

Appendix B Glossary

Application-Layer Bootstrapping Information Information that the device and an application-layer service must have in order for them to mutually authenticate and use a trusted application-layer onboarding protocol to onboard a device at the application layer. There is application-layer bootstrapping information about the device that the network must be in possession of, and application-layer bootstrapping information about the application service that the device must be in possession of. A typical example of application-layer bootstrapping information that the device must have is the public key that corresponds to the trusted application service's private key.

Application-Layer Onboarding The process of providing IoT devices with the application-layer credentials they need to establish a secure (i.e., encrypted) association with a trusted application service. This document defines two types of application-layer onboarding: independent and streamlined.

Independent Application-Layer Onboarding An application-layer onboarding process that does not rely on use of the network-layer onboarding process to transfer application-layer bootstrapping information between the device and the application service.

Network-Layer Bootstrapping Information Information that the device and the network must have in order for them to use a trusted network-layer onboarding protocol to onboard a device. There is network-layer bootstrapping information about the device that the network must be in possession of, and network-layer bootstrapping information about the network that the device must be in possession of. A typical example of device bootstrapping information that the network must have is the public key that corresponds with the device's private key.

Network-Layer Onboarding The process of providing IoT devices with the network-layer credentials and policy they need to join a network upon deployment.

Streamlined Application-Layer Onboarding An application-layer onboarding process that uses the network-layer onboarding protocol to securely transfer application-layer bootstrapping information between the device and the application service.

Trusted Network-Layer Onboarding A network-layer onboarding process that meets the following criteria:

- provides each device with unique network credentials,
- enables the device and the network to mutually authenticate,
- sends devices their network credentials over an encrypted channel,
- does not provide any person with access to the network credentials, and
- can be performed repeatedly throughout the device lifecycle to enable:
  - the device's network credentials to be securely managed and replaced as needed, and
  - the device to be securely onboarded to other networks after being repurposed or resold.

Appendix C Build 1 (Wi-Fi Easy Connect, Aruba/HPE)

C.1 Technologies

Build 1 is an implementation of network-layer onboarding that uses the Wi-Fi Easy Connect protocol. The onboarding infrastructure and related technology components for Build 1 have been provided by Aruba/HPE. IoT devices that were onboarded using Build 1 were provided by Aruba/HPE and CableLabs. The CA used for signing credentials issued to IoT devices was provided by SEALSQ, a subsidiary of WISeKey. For more information on these collaborators and the products and technologies that they contributed to this project overall, see Section 3.4.

Build 1 network onboarding infrastructure components within the NCCoE lab consist of the Aruba Access Point. Build 1 also requires support from Aruba Central and the UXI Cloud, which are accessed via the internet. IoT devices that can be network-layer onboarded using Build 1 include the Aruba/HPE UXI sensor and CableLabs Raspberry Pi. The UXI sensor also includes the Aruba UXI Application, which enables it to use independent (see Section 3.3.2) application-layer onboarding to be onboarded at the application layer as well, providing that the network to which the UXI sensor is onboarded has connectivity to the UXI Cloud via the internet. The Build 1 implementation supports the provisioning of all three types of network credentials defined in DPP:

- Connector for DPP-based network access
- Password/passphrase/PSK for WPA3/WPA2 network access
- X.509 certificates for 802.1X network access

Build 1 has been integrated with the SEALSQ CA on SEALSQ INeS CMS to enable Build 1 to obtain signed certificates from this CA when Build 1 is onboarding devices and issuing credentials for 802.1X network access. When issuing credentials for DPP and WPA3/WPA2-based network access, the configurator does not need to use a CA.

Table C-1 lists the technologies used in Build 1. It lists the products used to instantiate each component of the reference architecture and describes the security function that the component provides. The components listed are logical. They may be combined in physical form, e.g., a single piece of hardware may house a network onboarding component, a router, and a wireless access point.

Table C-1 Build 1 Products and Technologies

Component	Product	Function
Network-Layer Onboarding Component (Wi-Fi Easy Connect Configurator)	Aruba Access Point with support from Aruba Central	Runs the Wi-Fi Easy Connect network-layer onboarding protocol to interact with the IoT device to perform one-way or mutual authentication, establish a secure channel, and securely provide local network credentials to the device. If the network credential that is being provided to the device is a certificate, the onboarding component will interact with a certificate authority to sign the certificate. The configurator deployed in Build 1 supports DPP 2.0, but it is also backward compatible with DPP 1.0.

Component	Product	Function
Access Point, Router, or Switch	Aruba Access Point	Wireless access point that also serves as a router. It may get configured with per-device access control lists (ACLs) and policy when devices are onboarded.
Supply Chain Integration Service	Aruba Central	The device manufacturer provides device bootstrapping information to the HPE Cloud via the REST API that is documented in the DPP specification. Once the device is transferred to an owner, the HPE Cloud provides the device bootstrapping information (i.e., the device's DPP URI) to the device owner's private tenancy within the HPE Cloud.
Authorization Service	Cloud Auth (on Aruba Central)	The authorization service provides the configurator and router with the information needed to determine if the device is authorized to be onboarded to the network and, if so, whether it should be assigned any special roles or be subject to any specific access controls. It provides device authorization, role-based access control, and policy enforcement.
Build-Specific IoT Device	Aruba UXI Sensor	The IoT device that is used to demonstrate both trusted network-layer onboarding and trusted application-layer onboarding. It runs the Wi-Fi Easy Connect network-layer onboarding protocol supported by the build to securely receive its network credentials. It also has an application that enables it to perform independent (see Section 3.3.2) application-layer onboarding.
Generic IoT Device	Raspberry Pi	The IoT device that is used to demonstrate only trusted network-layer onboarding.
Secure Storage	Aruba UXI Sensor Trusted Platform Module (TPM)	Storage on the IoT device that is designed to be protected from unauthorized access and capable of detecting attempts to hack or modify its contents. Used to store and process private keys, credentials, and other information that must be kept confidential.
Certificate Authority (CA)	SEALSQ INeS CMS CA	Issues and signs certificates as needed. These certificates can be used by the device to connect to any 802.1a-based network.
Application-Layer Onboarding Service	UXI Application and UXI Cloud	After connecting to the network, the device downloads its application-layer credentials from the UXI cloud and uses them to authenticate to the UXI application, with which it interacts.

Component	Product	Function
Ongoing Device Authorization	N/A – Not intended for inclusion in this build	Performs activities designed to provide an ongoing assessment of the device’s trustworthiness and authorization to access network resources. For example, it may perform behavioral analysis or device attestation and use the results to determine whether the device should be granted access to certain high-value resources, assigned to a particular network segment, or other action taken.
Manufacturer Factory Provisioning Process	N/A (Not implemented at the time of publication)	Manufactures the IoT device. Creates, signs, and installs the device’s unique identity and other birth credentials into secure storage. Installs information the device requires for application-layer onboarding (if applicable). May populate a manufacturer database with information regarding devices that are created and, when the devices are sold, may record what entity owns them.

C.2 Build 1 Architecture

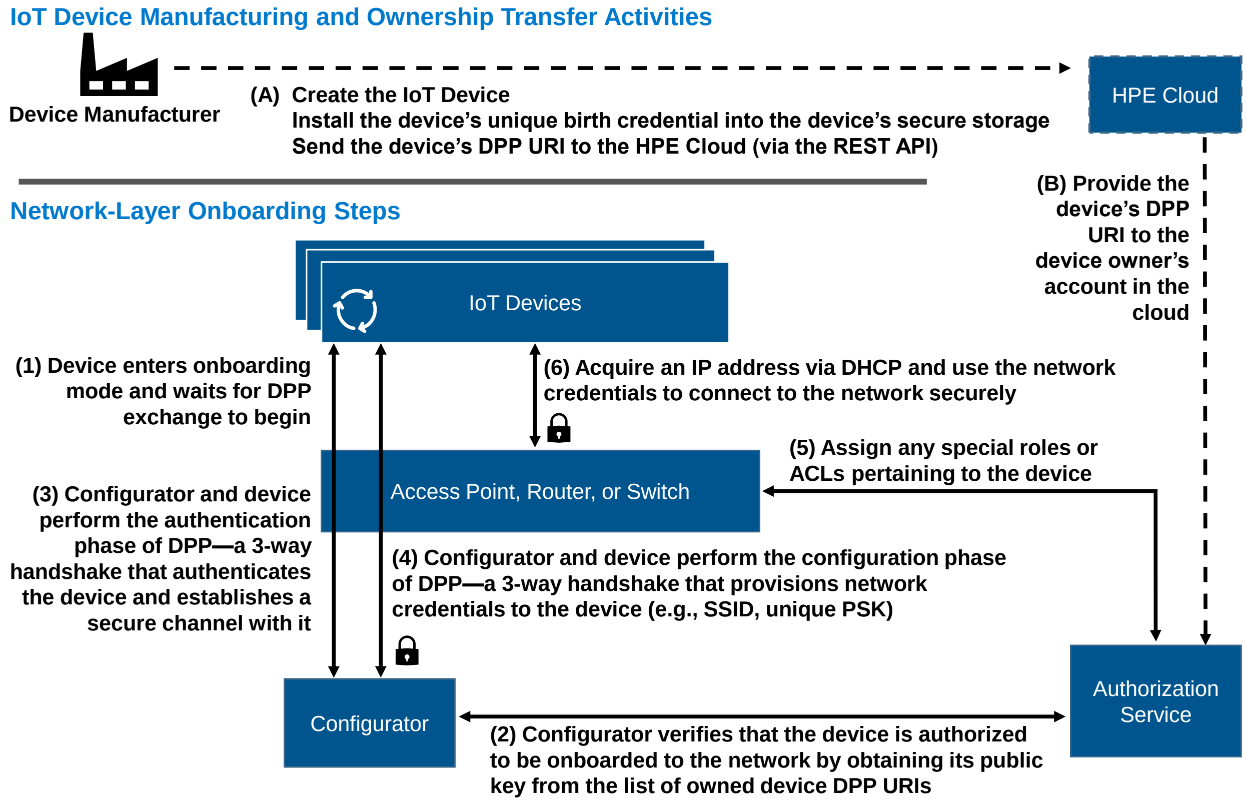
C.2.1 Build 1 Logical Architecture

The network-layer onboarding steps that are performed in Build 1 are depicted in Figure C-1. These steps are broken into two main parts: those required to transfer device bootstrapping information from the device manufacturer to the device owner’s authorization service (labeled with letters) and those required to perform network-layer onboarding of the device (labeled with numbers).

The device manufacturer:

1. Creates the device and installs a unique birth credential into secure storage on the device. Then the manufacturer sends the device’s bootstrapping information, which takes the form of a DPP URI, to Aruba Central in the HPE cloud. The device manufacturer interfaces with the HPE cloud via a REST API.
2. When the device is purchased, the device’s DPP URI is sent to the HPE cloud account of the device’s owner. The device owner’s cloud account contains the DPP URIs for all devices that it owns.

Figure C-1 Logical Architecture of Build 1


IoT Device Manufacturing and Ownership Transfer Activities
Device Manufacturer
(A) Create the IoT Device
Install the device's unique birth credential into the device's secure storage
Send the device's DPP URI to the HPE Cloud (via the REST API)
HPE Cloud
(B) Provide the device's DPP URI to the device owner's account in the cloud
Network-Layer Onboarding Steps
IoT Devices
(1) Device enters onboarding mode and waits for DPP exchange to begin
(6) Acquire an IP address via DHCP and use the network credentials to connect to the network securely
(5) Assign any special roles or ACLs pertaining to the device
Access Point, Router, or Switch
(3) Configurator and device perform the authentication phase of DPP—a 3-way handshake that authenticates the device and establishes a secure channel with it
(4) Configurator and device perform the configuration phase of DPP—a 3-way handshake that provisions network credentials to the device (e.g., SSID, unique PSK)
Configurator
Authorization Service
(2) Configurator verifies that the device is authorized to be onboarded to the network by obtaining its public key from the list of owned device DPP URIs

After obtaining the device, the device owner provisions the device with its network credentials by performing the following network-layer onboarding steps:

1. The owner puts the device into onboarding mode. The device waits for the DPP exchange to begin. This exchange includes the device issuing a discovery message, which the owner's configurator hears. The discovery message is secured such that it can only be decoded by an entity that possesses the device's DPP URI.

2. The configurator consults the list of DPP URIs of all owned devices to decode the discovery message and verify that the device is owned by the network owner and is therefore assumed to be authorized to be onboarded to the network.

3. Assuming the configurator finds the device's DPP URI, the configurator and the device perform the authentication phase of DPP, which is a three-way handshake that authenticates the device and establishes a secure (encrypted) channel with it.

4. The configurator and the device use this secure channel to perform the configuration phase of DPP, which is a three-way handshake that provisions network credentials to the device, along with any other information that may be needed, such as the network SSID.

5. The router or switch consults the owner's authentication, authorization, and accounting (AAA) service to determine if the device should be assigned any special roles or if any special ACL entries should be made for the device. If so, these are configured on the router or switch.

6. The device uses Dynamic Host Configuration Protocol (DHCP) to acquire an IP address and then uses its newly provisioned network credentials to connect to the network securely.

This completes the network-layer onboarding process.

After the device is network-layer onboarded and connects to the network, it automatically performs independent (see Section 3.3.2) application-layer onboarding. The application-layer onboarding steps are not depicted in Figure C-1. During the application-layer onboarding process, the IoT device, which is a UXI sensor, authenticates itself to the UXI cloud using its manufacturing certificate and pulls its application-layer credentials from the UXI cloud. In addition, if a firmware update is relevant, this also happens. The UXI sensor contacts the UXI cloud service to download a customer-specific configuration that tells it what to monitor on the customer's network. The UXI sensor then conducts the network performance monitoring functions it is designed to perform and uploads the data it collects to the UXI application dashboard.

C.2.2 Build 1 Physical Architecture

Section 5.2 describes the physical architecture of Build 1.

Appendix D Build 2 (Wi-Fi Easy Connect, CableLabs, OCF)

D.1 Technologies

Build 2 is an implementation of network-layer onboarding that uses the Wi-Fi Easy Connect protocol. Build 2 also supports streamlined (see Section 3.3.2) application-layer onboarding to the OCF security domain. The network-layer onboarding infrastructure for Build 2 is provided by CableLabs and the application-layer onboarding infrastructure is provided by OCF. IoT devices that were network-layer onboarded using Build 2 were provided by Aruba/HPE and OCF. Only the IoT devices provided by OCF were capable of being both network-layer onboarded and streamlined application-layer onboarded. For more information on these collaborators and the products and technologies that they contributed to this project overall, see Section 3.4.

Build 2 onboarding infrastructure components consist of the CableLabs Custom Connectivity Gateway Agent, which runs on the Gateway Access Point, and the Platform Controller. IoT devices onboarded by Build 2 include the Aruba UXI Sensor and CableLabs Raspberry Pi.

Table D-1 lists the technologies used in Build 2. It lists the products used to instantiate each logical build component and the security function that the component provides. The components listed are logical. They may be combined in physical form, e.g., a single piece of hardware may house a network onboarding component, a router, and a wireless access point.

Table D-1 Build 2 Products and Technologies

Component	Product	Function
Network-Layer Onboarding Component (Configurator)	CableLabs Custom Connectivity Gateway Agent with support from CableLabs Platform Controller	Runs the Wi-Fi Easy Connect network-layer onboarding protocol to interact with the IoT device to perform one-way or mutual authentication, establish a secure channel, and securely provide local network credentials to the device. It also securely conveys application-layer bootstrapping information to the device as part of the Wi-Fi Easy Connect protocol to support application-layer onboarding. The network-layer onboarding component deployed in Build 2 supports DPP 2.0, but it is also backward compatible with DPP 1.0.
Access Point, Router, or Switch	Raspberry Pi (running Custom Connectivity Gateway Agent)	The access point includes a configurator that runs the Wi-Fi Easy Connect Protocol. It also serves as a router that: 1) routes all traffic exchanged between IoT devices and the rest of the network, and 2) assigns each IoT device to a local network segment appropriate to the device's trust level (optional).

Component	Product	Function
Supply Chain Integration Service	CableLabs Platform Controller/IoTivity Cloud Service	The device manufacturer provides device bootstrapping information (i.e., the DPP URI) to the CableLabs Web Server. There are several potential mechanisms for sending the DPP URI to the CableLabs Web Server. The manufacturer can send the device's DPP URI to the Web Server directly, via an API. The API used is not the REST API that is documented in the DPP specification. However, the API is published and was made available to manufacturers wanting to onboard their IoT devices using Build 2. Once the device is transferred to an owner, the CableLabs Web Server provides the device's DPP URI to the device owner's authorization service, which is part of the owner's configurator.
Authorization Service	CableLabs Platform Controller	The authorization service provides the configurator and router with the information needed to determine if the device is authorized to be onboarded to the network and, if so, whether it should be assigned any special roles, assigned to any specific network segments, or be subject to any specific access controls.
Build-Specific IoT Device	Raspberry Pi (Bulb) Raspberry Pi (switch)	The IoT devices that are used to demonstrate both trusted network-layer onboarding and trusted application-layer onboarding. They run the Wi-Fi Easy Connect network-layer onboarding protocol to securely receive their network credentials. They also support application-layer onboarding of the device to the OCF environment by conveying the device's application-layer bootstrapping information as part of the network-layer onboarding protocol.
Generic IoT Device	Aruba UXI Sensor	The IoT device that is used to demonstrate only trusted network-layer onboarding.
Secure Storage	N/A (IoT device is not equipped with secure storage)	Storage designed to be protected from unauthorized access and capable of detecting attempts to hack or modify its contents. Used to store and process private keys and other information that must be kept confidential.
Certificate Authority	N/A (Not implemented at the time of publication)	Issues and signs certificates as needed.
Application-Layer Onboarding Service	OCF Diplomat and OCF OBT within IoTivity	After connecting to the network, the OCF Diplomat authenticates the devices, establishes secure channels with them, and sends them access control lists that control which bulbs each switch is authorized to turn on and off.

Component	Product	Function
Ongoing Device Authorization	N/A – Not intended for inclusion in this build	Performs activities designed to provide ongoing assessment of the device's trustworthiness and authorization to access network resources. For example, it may perform behavioral analysis or device attestation and use the results to determine whether the device should be granted access to certain high-value resources, assigned to a particular network segment, or other action taken.
Manufacturer Factory Provisioning Process	N/A (Not yet implemented)	Manufactures the IoT device. Creates, signs, and installs the device's unique identity and other birth credentials into secure storage. Installs information the device requires for application-layer onboarding (if applicable). May populate a manufacturer database with information regarding devices that are created and, when the devices are sold, may record what entity owns them.

D.2 Build 2 Architecture

D.2.1 Build 2 Logical Architecture

The network-layer onboarding steps that are performed in Build 2 are depicted in Figure D-1. These steps are broken into two main parts: those required to transfer device bootstrapping information from the device manufacturer to the device owner's authorization service (labeled with letters) and those required to perform network-layer onboarding of the device (labeled with numbers).

The device manufacturer:

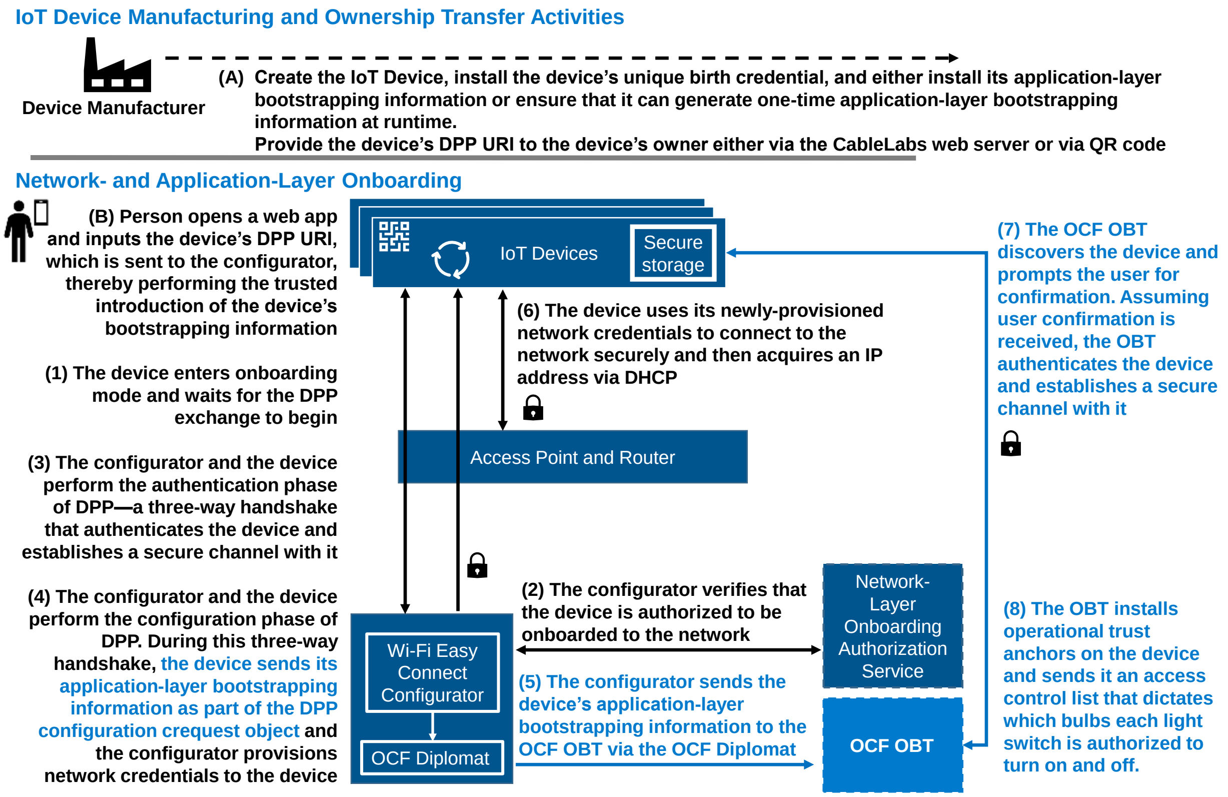
1. Creates the device and installs a unique birth credential into secure storage on the device. Because the device created for use in Build 2 will also perform application-layer onboarding into the OCF security domain, as part of the manufacturing process the manufacturer also either installs application-layer bootstrapping information onto the device or ensures that the device has the capability to generate one-time application-layer bootstrapping information at runtime. Then the manufacturer makes the device's network-layer bootstrapping information, which takes the form of a DPP URI, available to the device's owner.

   Build 2 supports several mechanisms whereby the manufacturer can make the device's network-layer bootstrapping information (i.e., its DPP URI) available to the device owner. The device's DPP URI can be uploaded directly to a device owner's cloud account or web server via API (as might come in handy when onboarding many enterprise devices at one time). Alternatively, the DPP URI can be manually entered into a local web portal that runs a configuration webpage that a device on the same Wi-Fi network can connect to for purposes of scanning a QR code or typing in the DPP URI. A DPP URI that is to be entered manually could, for example, be emailed to the owner or encoded into a QR code and printed on the device chassis, in device documentation, or on device packaging. Table D-1 depicts the case in which the manufacturer provides the device's DPP URI to the owner for manual entry. When the owner receives the device's DPP URI, the owner may optionally add the device's DPP URI to a list of

DPP URIs for devices that it owns that is maintained as part of the owner's authorization service. Such a list would enable the owner's network to determine if a device is authorized to be onboarded to it.

2. The person onboarding the device opens a web application and enters the device's DPP URI. The web application then sends the DPP URI to the Wi-Fi Easy Connect configurator, e.g., through a web request. (Note: Although the laboratory implementation of Build 2 requires the user to enter the DPP URI via a web page, an implementation designed for operational use would typically require the user to provide the DPP URI by scanning a QR code into a network operator-provided app that is logged into the user's account.)

Figure D-1 Logical Architecture of Build 2


IoT Device Manufacturing and Ownership Transfer Activities
Device Manufacturer
(A) Create the IoT Device, install the device's unique birth credential, and either install its application-layer bootstrapping information or ensure that it can generate one-time application-layer bootstrapping information at runtime.
Provide the device's DPP URI to the device's owner either via the CableLabs web server or via QR code
Network- and Application-Layer Onboarding
(B) Person opens a web app and inputs the device's DPP URI, which is sent to the configurator, thereby performing the trusted introduction of the device's bootstrapping information
(1) The device enters onboarding mode and waits for the DPP exchange to begin
(3) The configurator and the device perform the authentication phase of DPP—a three-way handshake that authenticates the device and establishes a secure channel with it
(4) The configurator and the device perform the configuration phase of DPP. During this three-way handshake, the device sends its application-layer bootstrapping information as part of the DPP configuration crequest object and the configurator provisions network credentials to the device
IoT Devices
Secure storage
(6) The device uses its newly-provisioned network credentials to connect to the network securely and then acquires an IP address via DHCP
Access Point and Router
(2) The configurator verifies that the device is authorized to be onboarded to the network
Wi-Fi Easy Connect Configurator
OCF Diplomat
(5) The configurator sends the device's application-layer bootstrapping information to the OCF OBT via the OCF Diplomat
Network-Layer Onboarding Authorization Service
OCF OBT
(7) The OCF OBT discovers the device and prompts the user for confirmation. Assuming user confirmation is received, the OBT authenticates the device and establishes a secure channel with it
(8) The OBT installs operational trust anchors on the device and sends it an access control list that dictates which bulbs each light switch is authorized to turn on and off.

After ensuring that the device's network-layer bootstrapping information (i.e., its DPP URI) has been uploaded to the configurator, the device owner performs both trusted network-layer onboarding and streamlined application-layer onboarding to the OCF security domain by performing the steps depicted in Figure D-1. In this diagram, the components that relate to network-layer onboarding are depicted in dark blue and their associated steps are written in black font. The components and steps that are related to application-layer onboarding are depicted in light blue. The steps are as follows:

1. The owner puts the device into onboarding mode. The device waits for the DPP exchange to begin. This exchange includes the device issuing a discovery message, which the owner's configurator hears. The discovery message is secured such that it can only be decoded by an entity that possesses the device's DPP URI.

2. Optionally, if such a list is being maintained, the configurator consults the list of DPP URIs of all owned devices to verify that the device is owned by the network owner and is, therefore, assumed to be authorized to be onboarded to the network. (If the device is being onboarded by an enterprise, the enterprise would likely maintain such a list; however, if the device is being onboarded to a home network, this step might be omitted.)

3. Assuming the configurator finds the device's DPP URI, the configurator and the device perform the authentication phase of DPP, which is a three-way handshake that authenticates the device and establishes a secure (encrypted) channel with it.

4. The configurator and the device use this secure channel to perform the configuration phase of DPP, which is a three-way handshake that provisions network credentials to the device, along with any other information that may be needed, such as the network SSID. In particular, as part of the three-way handshake in the Build 2 demonstration, the device sends its application-layer bootstrapping information to the configurator as part of the DPP configuration request object.

5. The configurator receives the device's application-layer bootstrapping information and forwards it to the OCF Diplomat. The purpose of the OCF Diplomat is to provide a bridge between the network and application layers. It accomplishes this by parsing the org.openconnectivity fields of the DPP request object, which contains the UUID of the device and the application-layer bootstrapping credentials, and sending these to the OCF OBT as part of a notification that the OBT has a new device to onboard. The Diplomat and the OBT use a subscribe and notify mechanism to ensure that the OBT will receive the onboarding request even if the OBT is unreachable for a period of time (e.g., the OBT is out of the home).

6. The device uses its newly provisioned network credentials to connect to the network securely and then uses DHCP to acquire an IP address. This completes the network-layer onboarding process.

7. The OBT implements a filtered discovery mechanism using the UUID provided from the OCF Diplomat to discover the new device on the network. Once it discovers the device, before proceeding, the OBT may optionally prompt the user for confirmation that they want to perform application-layer onboarding to the OCF security domain. This prompting may be accomplished, for example, by sending a confirmation request to an OCF app on the user's mobile device. Assuming the user responds affirmatively, the OBT uses the application-layer bootstrapping information to authenticate the device and take ownership of it by setting up a Datagram Transport Layer Security (DTLS) connection with the device.

8. The OBT then installs operational trust anchors and access control lists onto the device. For example, in the access control list, each light bulb may have an access control entry dictating which light switches are authorized to turn it on and off. This completes the application-layer onboarding process.

Note that, at this time, the application-layer bootstrapping information is provided unilaterally in the Build 2 application-layer onboarding demonstration. The application-layer bootstrapping information of the device is provided to the OCF Diplomat, enabling the OBT to authenticate the device. In a future version of this process, the application-layer bootstrapping information could be provided bidirectionally, meaning that the OCF Diplomat could also send the OCF operational root of trust to the IoT device as part of the DPP configuration response frame. Exchanging application-layer bootstrapping information bilaterally in this way would enable the secure channel set up as part of the network-layer onboarding process to support establishment of a mutually authenticated session between the device and the OBT.

In the Build 2 demonstration, two IoT devices, a switch and a light bulb, are onboarded at both the network and application layers. Each of these devices sends the OCF Diplomat its application-layer bootstrapping information over the secure network-layer onboarding channel during the network-layer onboarding process. Immediately after they complete the network-layer onboarding process and connect to the network, the OCF Diplomat provides their application-layer bootstrapping information to the OBT. The OBT then uses the provided application-layer bootstrapping information to discover, authenticate, and onboard each device. Because the devices have no way to authenticate the identity of the OBT in the current implementation, the devices are configured to trust the OBT upon first use.

After the OBT authenticates the devices, it establishes secure channels with them and provisions them access control lists that control which bulbs each switch is authorized to turn on and off. To demonstrate that the application onboarding was successful, Build 2 demonstrates that the switch is able to control only those bulbs that the OCF OBT has authorized it to.

D.2.2 Build 2 Physical Architecture

Section 5.3 describes the physical architecture of Build 2.

Appendix E Build 3 (BRSKI, Sandelman Software Works)

E.1 Technologies

Build 3 is an implementation of network-layer onboarding that uses the BRSKI protocol. Build 3 does not support application-layer onboarding. The network-layer onboarding infrastructure and related technology components for Build 3 were provided by Sandelman Software Works. The Raspberry Pi, ESP32, and Nordic NRF IoT devices that will be onboarded in a future implementation of Build 3 were also provided by Sandelman Software Works, as was the Sandelman Software Works Reach Pledge Simulator, which is the device that is onboarded in the current build. The IoT devices do not have secure storage, but future plans are to integrate them with secure storage elements. Build 3 issues private PKI certificates as network credentials at this time, but future plans are to integrate Build 3 with a third-party private CA from which it can obtain signed certificates. For more information on Sandelman Software Works and the products and technologies that it contributed to this project overall, see Section 3.4.

Onboarding Build 3 infrastructure components consist of Raspberry Pi, Nordic NRF, ESP32, Sandelman Software Works Minerva Fountain Join Registrar/Coordinator, Sandelman Software Works Minerva. Highway, Sandelman Software Works Reach Pledge Simulator, and a Minerva Fountain internal CA.

Table E-1 lists the technologies used in Build 3. It lists the products used to instantiate each logical build component and the security function that the component provides. The components are logical. They may be combined in physical form, e.g., a single piece of hardware may house both a network onboarding component and a router and/or wireless access point.

Table E-1 Build 3 Products and Technologies

Component	Product	Function
Network-Layer Onboarding Component (BRSKI Domain Registrar)	Sandelman Software Works Minerva Fountain Registrar	Runs the BRSKI protocol. It authenticates the IoT device, receives a voucher-request from the IoT device, and passes the request to the MASA. It also receives a voucher from the MASA, verifies it, and passes it to the IoT device. Assuming the IoT device finds the voucher to be valid and determines that the network is authorized to onboard it, the Domain Registrar provisions network credentials to the IoT device using EST.
Access Point, Router, or Switch	Turris MOX router running OpenWRT	The Onboarding Router segments the onboarding device from the rest of the network until the BRSKI onboarding is complete

Component	Product	Function
Supply Chain Integration Service (Manufacturer Authorized Signing Authority—MASA)	Minerva Highway, which is a MASA provided by Sandelman Software Works	The device manufacturer provides device bootstrapping information (e.g., the device's X.509 certificate) and device ownership information to the MASA. The MASA creates and signs a voucher saying who the owner of the device is and provides this voucher to the IoT device via the Domain Registrar so that the device can verify that the network that is trying to onboard it is authorized to do so.
Authorization Service	Minerva Highway, which is a MASA provided by Sandelman Software Works	As described in the previous row.
IoT Device (Pledge)	Sandelman Software Works Reach Pledge Simulator	The device that is used to demonstrate trusted network-layer onboarding by joining the network.
Secure Storage	N/A (The IoT devices and the Sandelman Software Works Reach Pledge Simulator do not include secure storage)	Storage on the IoT device that is designed to be protected from unauthorized access and capable of detecting attempts to hack or modify its contents. Used to store and process private keys, credentials, and other information that must be kept confidential.
Certificate Authority	N/A (self-signed certificates were used)	Issues and signs certificates as needed.
Application-Layer Onboarding Service	None. Not supported in this build.	After connecting to the network, the device mutually authenticates with a trusted application service and interacts with it at the application layer.
Ongoing Device Authorization	N/A – Not intended for inclusion in this build	Performs activities designed to provide an ongoing assessment of the device's trustworthiness and authorization to access network resources. For example, it may perform behavioral analysis or device attestation and use the results to determine whether the device should be granted access to certain high-value resources, assigned to a particular network segment, or other action taken.
Manufacturer Factory Provisioning Process	N/A (Not implemented at the time of publication)	Manufactures the IoT device. Creates, signs, and installs the device's unique identity and other birth credentials into secure storage. Installs information the device requires for application-layer onboarding (if applicable). May populate a manufacturer database with information regarding devices that are created and, when the devices are sold, may record what entity owns them.

E.2 Build 3 Architecture

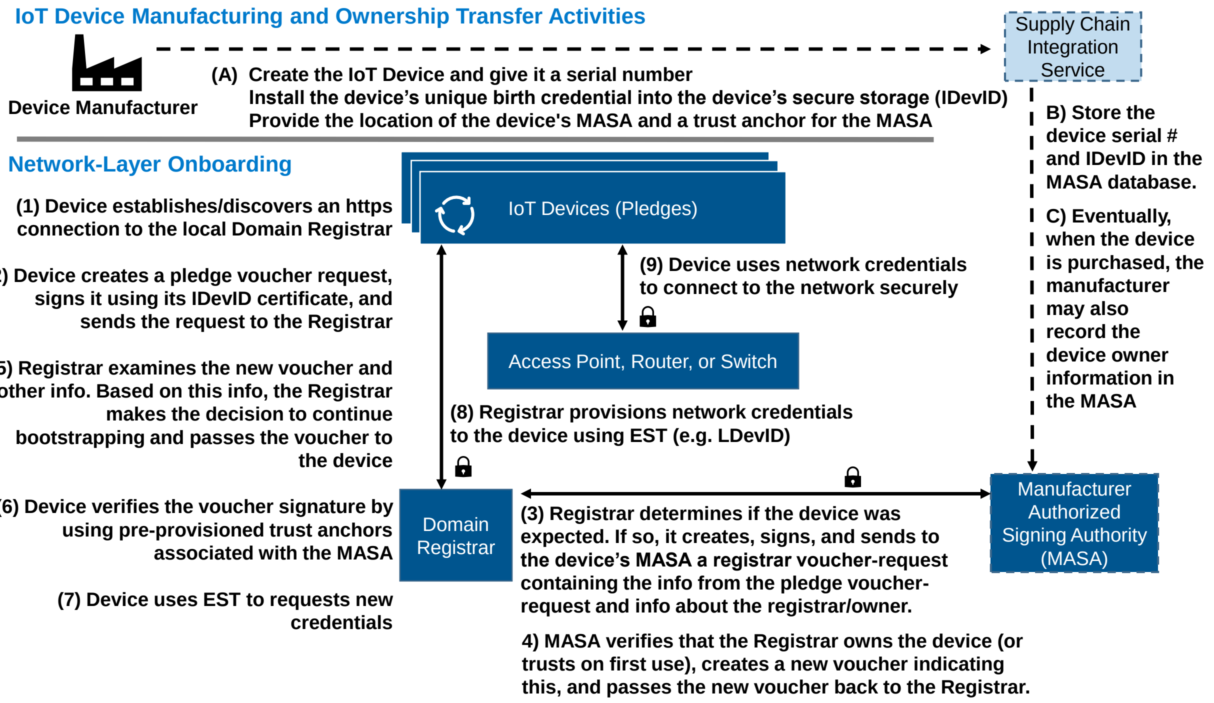
E.2.1 Build 3 Logical Architecture

The network-layer onboarding steps that are performed in Build 3 are depicted in Figure E-1. These steps are broken into two main parts: those required to transfer device bootstrapping information from the device manufacturer to the device owner’s authorization service (labeled with letters) and those required to perform network-layer onboarding of the device (labeled with numbers). These steps are described in greater detail in IETF RFC 8995.

The device manufacturer:

1. Creates the device and installs a unique serial number and birth credential into secure storage on the device. This unique birth credential takes the form of a private key and its associated 802.1AR certificate, e.g., the device’s IDevID. As part of this factory-installed certificate process, the location of the device’s MASA is provided in an extension to the IDevID. The device is also provided with trust anchors for the MASA entity that will sign the returned vouchers.
2. Stores information about the device, such as its serial number and its IDevID, in the MASA’s database.
3. Eventually, when the device is sold, the MASA may also record the device ownership information in its database.

Figure E-1 Logical Architecture of Build 3


IoT Device Manufacturing and Ownership Transfer Activities
Device Manufacturer
Supply Chain Integration Service
Network-Layer Onboarding
IoT Devices (Pledges)
Access Point, Router, or Switch
Domain Registrar
Manufacturer Authorized Signing Authority (MASA)

(A) Create the IoT Device and give it a serial number
Install the device’s unique birth credential into the device’s secure storage (IDevID)
Provide the location of the device's MASA and a trust anchor for the MASA

B) Store the device serial # and IDevID in the MASA database.

C) Eventually, when the device is purchased, the manufacturer may also record the device owner information in the MASA

(1) Device establishes/discovers an https connection to the local Domain Registrar

(2) Device creates a pledge voucher request, signs it using its IDevID certificate, and sends the request to the Registrar

(3) Registrar determines if the device was expected. If so, it creates, signs, and sends to the device’s MASA a registrar voucher-request containing the info from the pledge voucher-request and info about the registrar/owner.

4) MASA verifies that the Registrar owns the device (or trusts on first use), creates a new voucher indicating this, and passes the new voucher back to the Registrar.

(5) Registrar examines the new voucher and other info. Based on this info, the Registrar makes the decision to continue bootstrapping and passes the voucher to the device

(6) Device verifies the voucher signature by using pre-provisioned trust anchors associated with the MASA

(7) Device uses EST to requests new credentials

(8) Registrar provisions network credentials to the device using EST (e.g. LDevID)

(9) Device uses network credentials to connect to the network securely

After obtaining the device, the device owner provisions the device with its network credentials by performing the following network-layer onboarding steps:

1. The owner puts the device into onboarding mode. The device establishes an https connection to the local Domain Registrar. Trust in the Domain Registrar is provisional. (In a standard implementation, the device would use link-local network connectivity to locate a join proxy, and the join proxy would provide the device with https connectivity to the local Domain Registrar. The Build 3 implementation, however, does not support discovery at this time. To overcome this code limitation, the IoT device has been pre-provided with the address of the local Domain Registrar, to which it connects directly.)

2. The device creates a pledge voucher-request that includes the device serial number, signs this request with its IDevID certificate (i.e., its birth credential), and sends this signed request to the Registrar.

3. The Registrar receives the pledge voucher-request and considers whether the manufacturer is known to it and whether devices of that type are welcome. If so, the Registrar forms a registrar voucher-request that includes all the information from the pledge voucher-request along with information about the registrar/owner. The Registrar signs this registrar voucher-request. It locates the MASA that the IoT device is known to trust (e.g., the MASA that is identified in the device's IDevID extension) and sends the registrar voucher-request to the MASA.

4. The MASA consults the information that it has stored and applies policy to determine whether or not to approve the Registrar's claim that it owns and/or is authorized to onboard the device. (For example, the MASA may consult sales records for the device to verify device ownership, or it may be configured to trust that the first registrar that contacts it on behalf of a given device is in fact the device owner.) Assuming the MASA decides to approve the Registrar's claim to own and/or be authorized to onboard the device, the MASA creates a voucher that directs the device to accept its new owner/authorized network, signs this voucher, and sends it back to the Registrar.

5. The Registrar receives this voucher, examines it along with other related information (such as security posture, remote attestation results, and/or expected device serial numbers), and determines whether it trusts the voucher. Assuming it trusts the voucher, the Registrar passes the voucher to the device.

6. The device uses its factory-provisioned MASA trust anchors to verify the voucher signature, thereby ensuring that the voucher can be trusted. The voucher also validates the Registrar and represents the intended owner, ending the provisional aspect of the EST connection.

7. The device uses Enrollment over Secure Transport (EST) to request new credentials.

8. The Registrar provisions network credentials to the device using EST. These network credentials get stored into secure storage on the device, e.g., as an LDevID.

9. The device uses its newly provisioned network credentials to connect to the network securely.

This completes the trusted network-layer onboarding process for Build 3.

E.2.2 Build 3 Physical Architecture

Section 5.4 describes the physical architecture of Build 3.

Appendix F Build 4 (Thread, Silicon Labs-Thread, Kudelski KeySTREAM)

F.1 Technologies

Build 4 is an implementation of network-layer connection to an OpenThread network, followed by use of the Kudelski IoT keySTREAM Service to perform independent (see Section 3.3.2) application-layer onboarding of the device to a particular customer's tenancy in the AWS IoT Core. To join the network, the joining device generates and displays a pre-shared key that the owner enters on the commissioner, through a web interface, for authentication. The network-layer infrastructure for Build 4 was provided by Silicon Labs. The application-layer onboarding infrastructure for Build 4 was provided by Kudelski IoT. IoT devices that were onboarded using Build 4 were provided by Silicon Labs. For more information on these collaborators and the products and technologies that they contributed to this project overall, see Section 3.4.

Build 4 network infrastructure components within the NCCoE lab consist of a Thread border router (which is implemented using a Raspberry Pi) and a Silicon Labs Gecko Wireless Starter Kit. Build 4 also requires support from the Kudelski IoT keySTREAM service to perform application-layer onboarding. The keySTREAM service comes as a SaaS platform that is running in the cloud (accessible via the internet), and a software library (KTA – Kudelski Trusted Agent) that is integrated in the IoT device software stack. The KTA integrates with the Silicon Labs' Hardware Root of Trust (Secure Vault). The IoT device that is connected to the network and application-layer onboarded using Build 4 is the Silicon Labs Thunderboard (BRD2601A) with EFR32MG24x with Secure Vault(TM) High which is security certified to PSA/SESIP Level 3.

Table F-1 lists the technologies used in Build 4. It lists the products used to instantiate each logical build component and the security function that the component provides. The components are logical. They may be combined in physical form, e.g., a single piece of hardware may house a network onboarding component, a router, and a wireless access point.

Table F-1 Build 4 Products and Technologies

Component	Product	Function
Network-Layer Onboarding Component (Thread Protocol Component)	SLWSTK6023A Thread Radio Transceiver (Wireless starter kit);	The SLWSTK6023A acts as a Thread radio transceiver or radio coprocessor (RCP), allowing the open thread boarder router host platform to form and communicate with a Thread network. If the Thread MeshCoP were running on this device, it would provision the IoT device with credentials for the Thread network.
Access Point, Router, or Switch	OpenThread Border Router (OTBR) hosted on a Raspberry Pi	Router that has interfaces both on the Thread network and on the IP network to act as a bridge between the Thread network and the public internet. This allows the IoT device that communicates using the Thread wireless protocol to communicate with cloud services.

Component	Product	Function
Supply Chain Integration Service	Silicon Labs Custom Parts Manufacturer Service (CPMS)	To support network-layer onboarding, the device manufacturer provides device bootstrapping information to the to the device owner.
Authorization Service	Not implemented	Enables the network to verify that the device that is trying to onboard to it is authorized to do so.
IoT Device	Silicon Labs Thunderboard (BRD2601A)	The IoT device that is used to demonstrate trusted network- and application-layer onboarding.
Secure Storage	Secure Vault ™ High on Silicon Labs IoT device	Storage designed to be protected from unauthorized access and capable of detecting attempts to hack or modify its contents. Used to store and process private keys and other information that must be kept confidential.
Certificate Authority	Each tenant in the Kudelski keySTREAM service cloud has its own certificate signing authority	Issues and signs certificates as needed. For application-layer onboarding, the device owner has its own certificate signing authority in its portion of the Kudelski keySTREAM service cloud.
Application-Layer Onboarding Service	Kudelski keySTREAM Service	After connecting to the Thread network, the device performs application-layer onboarding by accessing the Kudelski keySTREAM service. The device and the keySTREAM service mutually authenticate; the keySTREAM service verifies the device's owner, generates an application-layer credential (i.e., an AWS certificate that is based on the device's chipset identity and owner) for the device, and provisions the device with this X.509 credential that will enable the device to access the owner's tenancy in the AWS IoT Core cloud.
Ongoing Device Authorization	N/A – Not intended for inclusion in this build	Performs activities designed to provide an ongoing assessment of the device's trustworthiness and authorization to access network resources. For example, it may perform behavioral analysis or device attestation and use the results to determine whether the device should be granted access to certain high-value resources, assign the device to a particular network segment, or take other action.

Component	Product	Function
Manufacturer Factory Provisioning Process	Silicon Labs Custom Parts Manufacturing Service (CPMS)	Manufactures the IoT device. Creates, signs, and installs the device's unique identity and other birth credentials into secure storage. Installs software and information the device requires for application-layer onboarding. May populate a manufacturer database with information regarding devices that are created and, when the devices are sold, may record what entity owns them. The MG24 "B" version comes pre-loaded with a Silicon Labs Birth certificate. The "A" or "B" version birth certificate can be modified via their Custom Part Manufacturing Service (CPMS) to be unique per end device manufacturer and signed into their Root CA if desired.

F.2 Build 4 Architecture

F.2.1 Build 4 Logical Architecture

Build 4 demonstrates a device connecting to an OpenThread network. IoT devices generate and use a pre-shared key to connect to the OpenThread network of Build 4 using the Thread MeshCoP service. Once a device is connected to the OpenThread network of Build 4, it gets access to an IP network via a border router, and then performs application-layer onboarding using the Kudelski keySTREAM Service. Kudelski keySTREAM is a device security management service that runs as a SaaS platform on the Amazon cloud. Build 4 relies on an integration that has been performed between Silicon Labs and Kudelski keySTREAM. KeySTREAM has integrated software libraries with the Silicon Lab EFR32MG24 (MG24) IoT device's secure vault to enable the private signing key that is associated with an application-layer certificate to be stored into the secure vault using security controls that are available on the MG24. This integration ensures that application-layer credentials can be provisioned into the vault securely such that no key material is misused or exposed.

At a high level, the steps required to enable demonstration of Build 4's network connection and application-layer onboarding capabilities can be broken into the following three main parts:

- Device Preparation: The IoT device is prepared for network connection and application-layer onboarding by the device manufacturer.
    - The device comes from the manufacturer ready to be provisioned onto a Thread network. No additional preparation is required.
    - The device is prepared for application-layer onboarding on behalf of a specific, pre-defined customer who will become its owner. The device is assigned ownership to this customer (e.g., customer A) and this ownership information is sealed into device firmware, permanently identifying the device as being owned by customer A. The device owner, customer A, has a tenancy on the Kudelski keySTREAM Service and is also an Amazon Web Services (AWS) customer. After the device has been prepared, the device is provided to its owner (customer A).

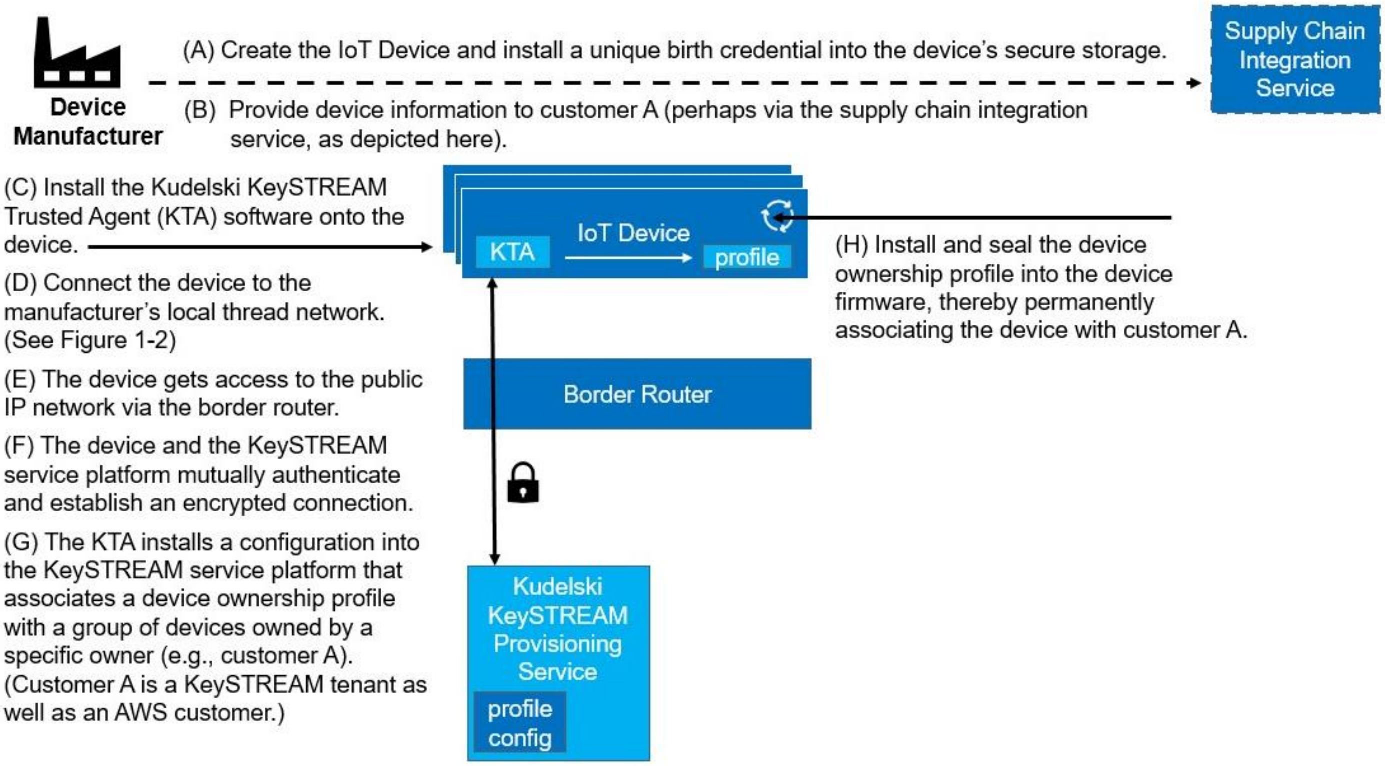
- Network Connection: Customer A connects the device to Customer A's OpenThread network by entering the pre-shared key displayed on the device's serial terminal in the OpenThread Border Router's (OTBR) web interface. This allows the network's radio channel, PAN ID, extended PAN ID and network name to be discovered, avoiding the need to preconfigure any of these parameters. Once on customer A's OpenThread network, the device has access to the public IP network via the border router.
- Application-Layer Onboarding: The device and the keySTREAM service mutually authenticate, keySTREAM confirms that customer A owns the device, and keySTREAM provisions the device with an AWS certificate that is specific to the device and to customer A, enabling the device to authenticate to customer A's tenancy in the AWS IoT Core.

Each of these three aspects of the demonstration are illustrated in its own figure and described in more detail in the three subsections below.

F.2.1.1 Device Preparation

Figure F-1 depicts the steps that are performed by the device manufacturer, which in this case is Silicon Labs, to prepare the device for network- and application-layer onboarding by a particular customer, Customer A. Each step is described in more detail below. Because these steps are performed to prepare the device for onboarding rather than as part of onboarding itself, they are labeled with letters instead of numbers in keeping with the conventions used in other build descriptions.

Figure F-1 Logical Architecture of Build 4: Device Preparation


Device Manufacturer
(A) Create the IoT Device and install a unique birth credential into the device's secure storage.
(B) Provide device information to customer A (perhaps via the supply chain integration service, as depicted here).
Supply Chain Integration Service
(C) Install the Kudelski KeySTREAM Trusted Agent (KTA) software onto the device.
(D) Connect the device to the manufacturer's local thread network. (See Figure 1-2)
(E) The device gets access to the public IP network via the border router.
(F) The device and the KeySTREAM service platform mutually authenticate and establish an encrypted connection.
(G) The KTA installs a configuration into the KeySTREAM service platform that associates a device ownership profile with a group of devices owned by a specific owner (e.g., customer A). (Customer A is a KeySTREAM tenant as well as an AWS customer.)
IoT Device
KTA
profile
(H) Install and seal the device ownership profile into the device firmware, thereby permanently associating the device with customer A.
Border Router
Kudelski KeySTREAM Provisioning Service
profile config

The following steps are performed to prepare the device for network connection and application-layer onboarding:

1. The manufacturer creates the device, which in this case is a Silicon Labs MG24, and prepares it for network connection by installing the device's unique birth credential into the device's chipset. This chipset identity is a hardware root of trust. The MG24 "B" version comes pre-loaded with a Silicon Labs Birth certificate. The "A" or "B" version birth certificate can be modified via their Custom Part Manufacturing Service (CPMS) to be unique per end device manufacturer and signed into their Root CA if desired.

2. The manufacturer provides information about the device to customer A (perhaps via the supply chain service, as depicted in Figure 1-1) so customer A can be aware that the device is expected on its network.

3. The manufacturer prepares the device for application-layer onboarding by installing the Kudelski keySTREAM Trusted Agent (KTA) software onto the device.

4. The manufacturer connects the device to the manufacturer's local OpenThread network. (See Figure 1-2 for details of the network connection steps.) Note that in this case, which is the first time that the device is being connected to a network, the device is being connected to the manufacturer's network rather than to the network of the device's eventual owner.

5. After the device connects to the manufacturer's OpenThread network, the device has access to the public IP network via the border router.

6. The device and the Kudelski keySTREAM service mutually authenticate and establish an encrypted connection.

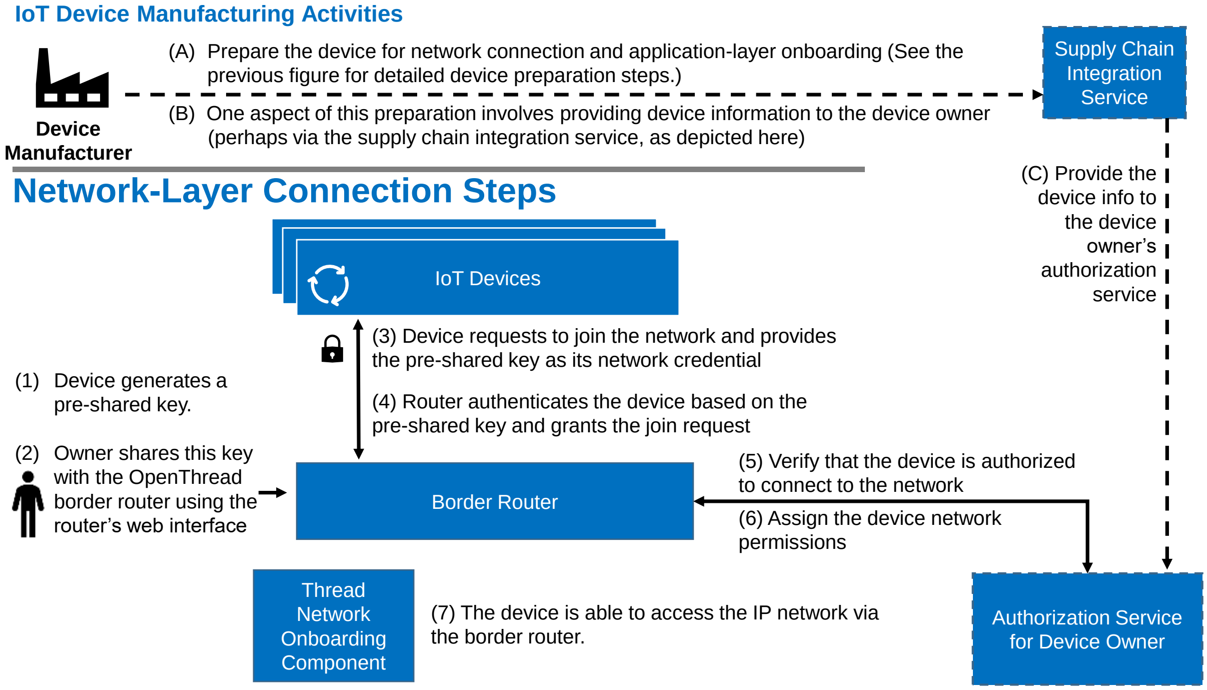
7. The KTA installs a configuration into the keySTREAM service platform that builds up a group of devices that belong to a certain end user and associates the group with a device ownership profile. This device ownership profile is associated with a particular customer (e.g., customer A). The same device profile is used by all devices in a group of devices that are owned by this owner. The profile is not specific to individual devices. The owner of these devices (customer A) has a keySTREAM tenancy, which includes a dedicated certificate signing CA. Customer A is also an AWS customer.

8. The device manufacturer installs and seals this device ownership profile into the device firmware. This profile permanently identifies the device as being owned by customer A.

F.2.1.2 Network-Layer Connection

Figure F-2 depicts the steps of an IoT device connecting to that thread network using a pre-shared key that the device generates and shares with the OpenThread boarder router. Each step is described in more detail below.

Figure F-2 Logical Architecture of Build 4: Connection to the OpenThread Network


IoT Device Manufacturing Activities
Device Manufacturer
(A) Prepare the device for network connection and application-layer onboarding (See the previous figure for detailed device preparation steps.)
(B) One aspect of this preparation involves providing device information to the device owner (perhaps via the supply chain integration service, as depicted here)
Supply Chain Integration Service
(C) Provide the device info to the device owner's authorization service
Network-Layer Connection Steps
IoT Devices
(1) Device generates a pre-shared key.
(2) Owner shares this key with the OpenThread border router using the router's web interface
(3) Device requests to join the network and provides the pre-shared key as its network credential
(4) Router authenticates the device based on the pre-shared key and grants the join request
Border Router
(5) Verify that the device is authorized to connect to the network
(6) Assign the device network permissions
Thread Network Onboarding Component
(7) The device is able to access the IP network via the border router.
Authorization Service for Device Owner

The device connects to the OpenThread network using the following steps:

1. The device generates a pre-shared key.

2. The owner starts the commissioning process by entering this pre-shared key on the OpenThread border router.

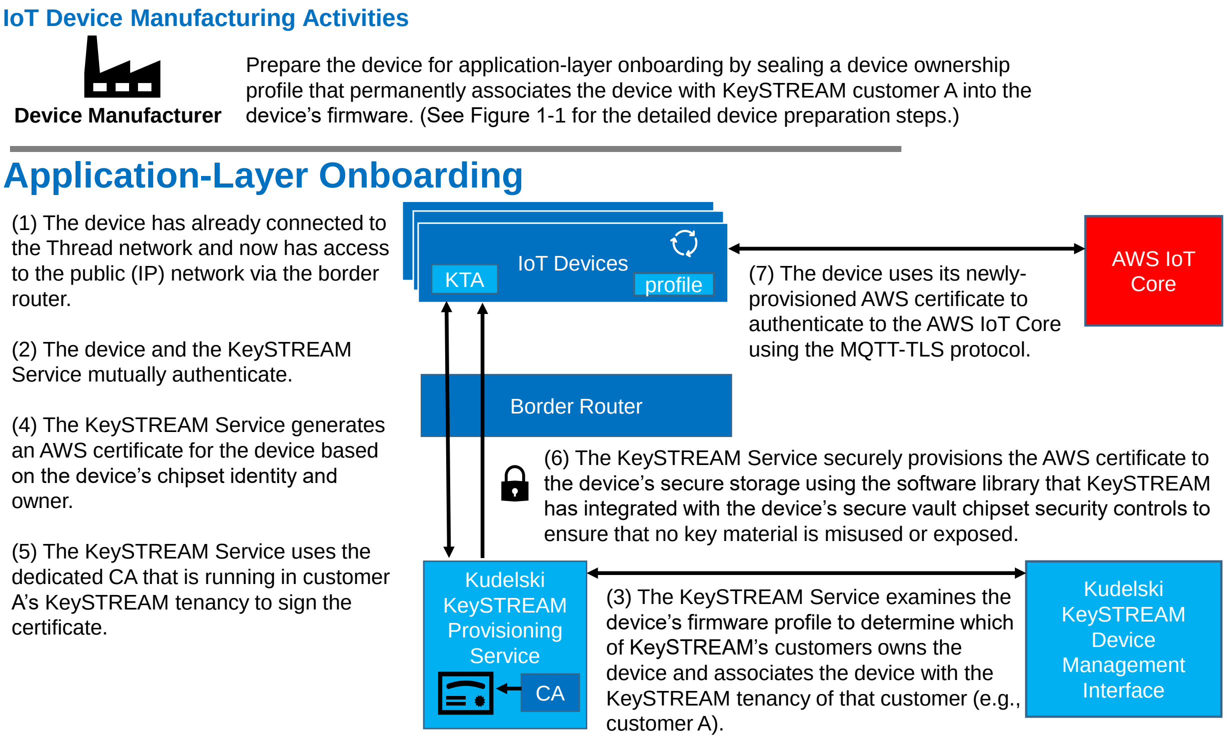
3. The device requests to join the network and provides the pre-shared key as its network credential.
4. The network authenticates the device based on the pre-shared key and grants the join request.
5. The network verifies that the device is authorized to connect to the network.
6. The network assigns the device network permissions and configures these as policies on the border router.
7. The device is able to access the IP network (and the internet) via the border router.

This completes the network-layer connection process.

F.2.1.3 Application-Layer Onboarding

Figure F-3 depicts the steps of the application-layer onboarding process using the Kudelski keySTREAM service. Each step is described in more detail below.

Figure F-3 Logical Architecture of Build 4: Application-Layer Onboarding using the Kudelski keySTREAM Service


IoT Device Manufacturing Activities
Device Manufacturer
Application-Layer Onboarding
KTA
IoT Devices
profile
AWS IoT Core
Border Router
Kudelski KeySTREAM Provisioning Service
CA
Kudelski KeySTREAM Device Management Interface

Prepare the device for application-layer onboarding by sealing a device ownership profile that permanently associates the device with KeySTREAM customer A into the device's firmware. (See Figure 1-1 for the detailed device preparation steps.)

(1) The device has already connected to the Thread network and now has access to the public (IP) network via the border router.

(2) The device and the KeySTREAM Service mutually authenticate.

(3) The KeySTREAM Service examines the device's firmware profile to determine which of KeySTREAM's customers owns the device and associates the device with the KeySTREAM tenancy of that customer (e.g., customer A).

(4) The KeySTREAM Service generates an AWS certificate for the device based on the device's chipset identity and owner.

(5) The KeySTREAM Service uses the dedicated CA that is running in customer A's KeySTREAM tenancy to sign the certificate.

(6) The KeySTREAM Service securely provisions the AWS certificate to the device's secure storage using the software library that KeySTREAM has integrated with the device's secure vault chipset security controls to ensure that no key material is misused or exposed.

(7) The device uses its newly-provisioned AWS certificate to authenticate to the AWS IoT Core using the MQTT-TLS protocol.

The application-layer onboarding steps performed to provision the device with its application-layer credentials (e.g., its AWS certificate) are as follows:

1. The device, which is already connected to the OpenThread network, accesses the IP network via the border router.
2. The device and the keySTREAM service mutually authenticate.

3. The keySTREAM Service examines the device’s firmware profile to determine which of keySTREAM’s customers owns the device. In this case, customer A is identified as the device owner. The keySTREAM service associates the device with customer A’s keySTREAM tenancy.

4. The keySTREAM Service generates an AWS IoT Core certificate for the device based on both the device’s ownership information and the secure hardware root of trust that is in the device’s chipset.

5. The keySTREAM Service uses the dedicated CA that is running in customer A’s keySTREAM tenancy to sign the AWS certificate.

6. The keySTREAM Service securely provisions the AWS certificate to the device’s secure storage using the software library that keySTREAM has integrated with the device’s secure vault chipset security controls to ensure that no key material is misused or exposed.

7. The device uses its newly provisioned application-layer credentials (i.e., the AWS certificate) to authenticate to customer A’s tenancy in the AWS IoT Core using the MQTT-TLS protocol.

F.2.2 Build 4 Physical Architecture

Section 5.5 describes the physical architecture of Build 4.

Appendix G Build 5 (BRSKI over Wi-Fi, NquiringMinds)

G.1 Technologies

Build 5 is an implementation of network-layer onboarding that uses a version of the BRSKI Protocol that has been modified to work over Wi-Fi. After the IoT device has joined the network, Build 5 also demonstrates a number of mechanisms that are performed on an ongoing basis to provide continuous, policy-based authorization and assurance. Both the network-layer onboarding infrastructure and the continuous assurance service for Build 5 were provided by NquiringMinds. This entire build can be replicated using the open sourced TrustNetZ code base.

For more information on NquiringMinds and the products and technologies that they contributed to this project overall, see Section 3.4.

Build 5 network onboarding infrastructure components within the NCCoE lab consist of a Linux based Raspberry Pi 4B router (which also runs the registrar service and MASA service), and a USB hub. The Build 5 components used to support the continuous assurance service include TrustNetZ Authorization interfaces, TrustNetZ information provider, and TrustNetZ policy engine. The IoT devices that are onboarded using Build 5 are a Raspberry Pi device. These IoT devices do not have secure storage, but use the Infineon Optiga SLB 9670 TPM 2.0 as an external secure element. Build 5 depends on an IDevID (X.509 Certificate) having been provisioned to the secure element of the IoT device (pledge) prior to onboarding, as part of the factory provisioning process (see Section H.1). For Build 5, this factory provisioning process was accomplished by the BRSKI Factory Provisioning Build, which is described in Appendix H.3.

Table G-1 lists the technologies used in Build 5. It lists the products used to instantiate each logical build component and the security function that the component provides. The components are logical. They may be combined in physical form, e.g., a single piece of hardware may house a network onboarding component, a router, and a wireless access point.

Table G-1 Build 5 Products and Technologies

Component	Product	Function
Network-Layer Onboarding Component (BRSKI Domain Registrar)	Stateful, non-persistent Linux app that has two functional interfaces for both BRSKI and for the Authentication Service. (TrustNetZ onboarding)	Runs the BRSKI protocol modified to work over Wi-Fi and acts as a BRSKI Domain Registrar. It authenticates the IoT device, receives a voucher request from the IoT device, and passes the request to the MASA. It also receives a voucher from the MASA, verifies it, and passes it to the IoT device. Assuming the IoT device finds the voucher to be valid and determines that the network is authorized to onboard it, the Domain Registrar provisions network credentials to the IoT device using EST.

Component	Product	Function
Access Point, Router, or Switch	Raspberry Pi 4B equipped with USB Wi-Fi dongle, running TrustNetZ AP code.	Router, providing an open Wi-Fi network and closed Wi-Fi network. Physical access control is mediated through the RADUIS interface (which is part of the TrustNetZ AP configuration) The AP also receives network commands from the continuous assurance service.
Supply Chain Integration Service (Manufacturer Authorized Signing Authority—MASA)	TrustNetZ MASA	The MASA creates and signs a voucher and provides this voucher to the IoT device via the Registrar so that the device can verify that the network that is trying to onboard it is authorized to do so.
Authorization Service	Linux application which contains an encapsulated policy engine (TrustNetZ policy engine)	Determines whether the device is authorized to be onboarded to the network. The application features a REST API which accepts verifiable credential claims to feed data on entities and their relationships into its SQL database. The policy engine itself is based on verifiable credentials presentation, (persisted to SQL database), making it easily configurable and extensible.
IoT Device	Raspberry Pi devices (running TrustNetZ pledge agent)	The IoT device that is used to demonstrate trusted network- and application-layer onboarding. Handles the client side BRSKI protocols, the integration with the secure storage, with factory provisioning and TLS connections.
Secure Storage	Infineon Optiga SLB 9670 TPM 2.0	Storage on the IoT device that is designed to be protected from unauthorized access and capable of detecting attempts to hack or modify its contents. Used to store and process private keys and other information that must be kept confidential.
Certificate Authority	TrustNetZ demo manufacturer CA (MPR – manufacture provisioning root) TrustNetZ Domain CA	Two CA are used in Build 5 Domain CA issues certificates and provides signing and attestation functions that model network owner relationships (e.g. sign the LDevID certificate) Manufacturer CA issues the IDevID certificates; proving the device has been created by the manufacturer.
Application-Layer Onboarding Service	TrustNetZ Demo application sever	After connecting to the network, the device mutually authenticates with a trusted application service and interacts with it at the application layer. The IDevID and TPM private key are used to establish a TLS session with the demonstration application server and send data to it from the device. This demonstrates the concept of secure connection to a third-party application server using the cryptographic artifacts from the onboarding process.

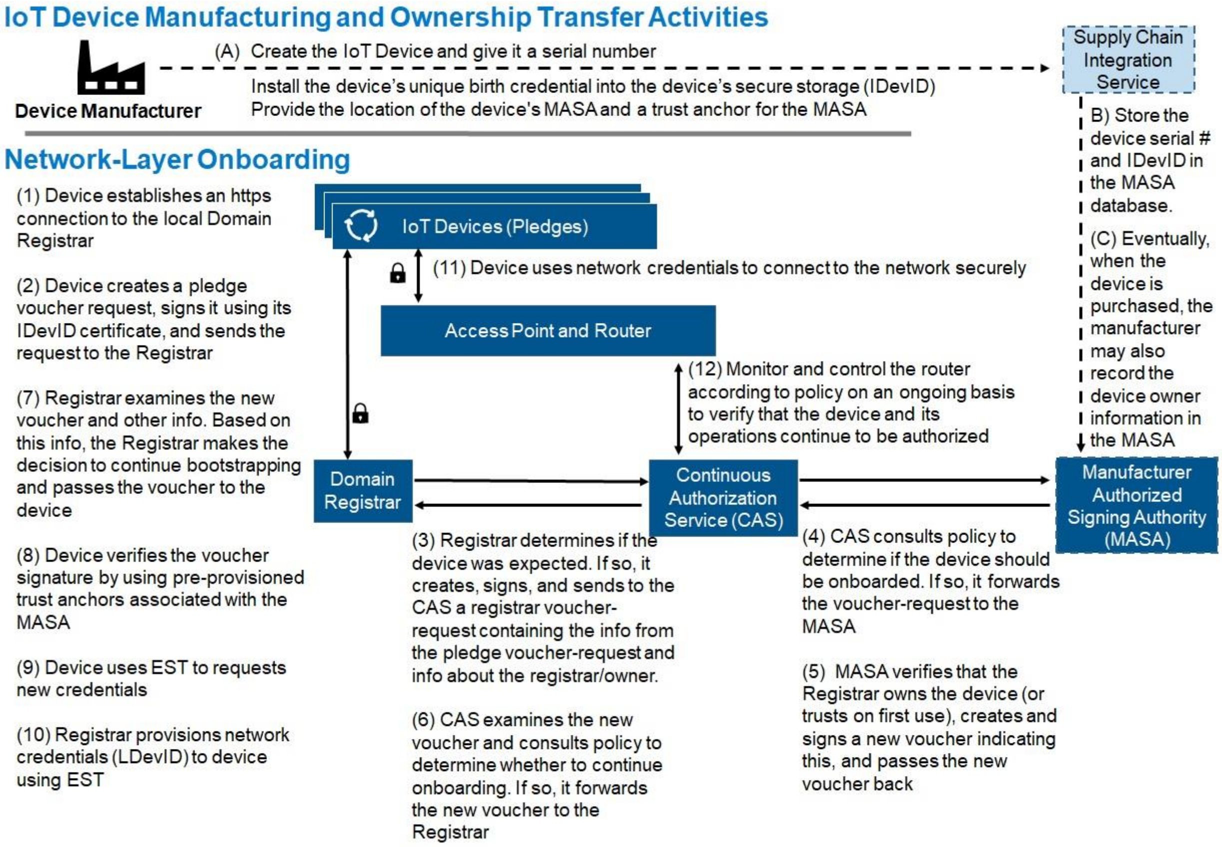
Component	Product	Function
Ongoing Device Authorization	Continuous Authorization Service, which calls into the in the TrustNetZ policy engine	Designed to perform a set of ongoing, policy-based continuous assurance and authorization checks on the device after it has connected to the network. As of this publication, the following ongoing checks have been implemented: - The manufacturer of the device must be trusted by the network owner - The device must be trusted by a user with appropriate privileges - The device must have an associated device type - The vulnerability score of the software bill of materials (SBOM) for the device type must be lower than a set threshold - The device must not have contacted an IP address that is on a deny list If it fails any of these periodic checks, its voucher is revoked, which removes the device from the network.
Manufacturer Factory Provisioning Process	BRSKI Factory Provisioning Process used to provision the Infineon TPM with its private key and IDevID (See Appendix H.3)	Manufactures the IoT device. Creates, signs, and installs the device's unique identity (i.e., its IDevID, which is an X.509 certificate) into secure storage. Installs information the device requires for application-layer onboarding. Populates the MASA with information regarding devices that are created and, when the devices are sold, may record what entity owns them.

G.2 Build 5 Architecture

G.2.1 Build 5 Logical Architecture

The network-layer onboarding steps that are performed in Build 5 are depicted in Figure G-1. These steps are broken into two main parts: those required to transfer device bootstrapping information from the device manufacturer to the MASA (labeled with letters) and those required to perform network-layer onboarding of the device and establish the operation of the continuous authorization service (labeled with numbers).

Figure G-1 Logical Architecture of Build 5


IoT Device Manufacturing and Ownership Transfer Activities
Device Manufacturer
(A) Create the IoT Device and give it a serial number
Install the device's unique birth credential into the device's secure storage (IDevID)
Provide the location of the device's MASA and a trust anchor for the MASA
Supply Chain Integration Service
B) Store the device serial # and IDevID in the MASA database.
(C) Eventually, when the device is purchased, the manufacturer may also record the device owner information in the MASA
Network-Layer Onboarding
(1) Device establishes an https connection to the local Domain Registrar
(2) Device creates a pledge voucher request, signs it using its IDevID certificate, and sends the request to the Registrar
(7) Registrar examines the new voucher and other info. Based on this info, the Registrar makes the decision to continue bootstrapping and passes the voucher to the device
(8) Device verifies the voucher signature by using pre-provisioned trust anchors associated with the MASA
(9) Device uses EST to requests new credentials
(10) Registrar provisions network credentials (LDevID) to device using EST
IoT Devices (Pledges)
(11) Device uses network credentials to connect to the network securely
Access Point and Router
(12) Monitor and control the router according to policy on an ongoing basis to verify that the device and its operations continue to be authorized
Domain Registrar
Continuous Authorization Service (CAS)
Manufacturer Authorized Signing Authority (MASA)
(3) Registrar determines if the device was expected. If so, it creates, signs, and sends to the CAS a registrar voucher-request containing the info from the pledge voucher-request and info about the registrar/owner.
(6) CAS examines the new voucher and consults policy to determine whether to continue onboarding. If so, it forwards the new voucher to the Registrar
(4) CAS consults policy to determine if the device should be onboarded. If so, it forwards the voucher-request to the MASA
(5) MASA verifies that the Registrar owns the device (or trusts on first use), creates and signs a new voucher indicating this, and passes the new voucher back

The device manufacturer:

1. Creates the device and installs a unique serial number and birth credential into secure storage on the device. This unique birth credential takes the form of a private key and its associated 802.1AR certificate, e.g., the device's IDevID. As part of this factory-installed certificate process, the location of the device's manufacturer authorized signing authority (MASA) is provided in an extension to the IDevID. The device is also provided with trust anchors for the MASA entity that will sign the returned vouchers.
2. Stores information about the device, such as its serial number and its IDevID, in the MASA's database.
3. Eventually, when the device is sold, the MASA may also record the device ownership information in its database.

After obtaining the device, the device owner provisions the device with its network credentials by performing the following network-layer onboarding steps:

1. The owner puts the device (i.e., the pledge) into onboarding mode. The device establishes an https connection to the local Domain Registrar. (In a standard BRSKI implementation, the device would have wired network connectivity. The device would use its link-local network connectivity to locate a join proxy, and the join proxy would provide the device with https connectivity to the

local Domain Registrar.) The Build 5 implementation, however, relies on wireless connectivity and initially uses the unauthenticated EAP-TLS protocol. The pledge discovers potential onboarding networks by searching for public Wi-Fi networks that either match a particular SSID wildcard name or that advertise a particular realm. When the device finds a potential onboarding network, it connects to it and attempts to discover the registrar. The pledge will connect to the open Wi-Fi network and will receive either an IPv4 or IPv6 address. Subsequently, the pledge will listen to mDNS packets and will obtain the list of join proxies (IP addresses). Finally, the pledge will subsequently connect to each join proxy using the BRSKI-EST protocol.

2. The device creates a pledge voucher-request that includes the device serial number, signs this request with its IDevID certificate (i.e., its birth credential), and sends this signed request to the Registrar.

3. The Registrar receives the pledge voucher-request and considers whether the manufacturer is known to it and whether devices of that type are welcome. If so, the Registrar forms a registrar voucher-request that includes all the information from the pledge voucher request along with information about the registrar/owner. The Registrar sends this registrar voucher-request to the Continuous Authorization Service.

4. The Continuous Authorization Service consults policy to determine if this device should be permitted to be onboarded and what other conditions should be enforced. An example of policy that might be used is that the network owner wants to disable MASA validation. Assuming the device is permitted to be onboarded, the Continuous Authorization Service locates the MASA that the IoT device is known to trust (i.e., the MASA that is identified in the device's IDevID extension) and sends the registrar voucher-request to the MASA.

5. The MASA consults the information that it has stored and applies policy to determine whether to approve the Registrar's claim that it owns the device. (For example, the MASA may consult sales records for the device to verify device ownership, or it may be configured to trust that the first registrar that contacts it on behalf of a given device is in fact the device owner). Assuming the MASA decides to approve the Registrar's claim to own the device, the MASA creates a new voucher that directs the device to accept its new owner, signs this voucher, and sends it back to the Continuous Authorization Service.

6. The Continuous Authorization Service receives this new voucher and examines it in consultation with policy to determine whether to continue onboarding. Some examples of policies that might be used include: permit onboarding only if no current critical vulnerabilities have been disclosed against the declared device type, the device instance has successfully passed a site-specific test process, or a test compliance certificate has been found for the declared device type. Assuming the device is permitted to be onboarded, the Continuous Authorization Service sends the new voucher to the Domain Registrar.

7. The Domain Registrar receives and examines the new voucher along with other related information and determines whether it trusts the voucher. Assuming it trusts the voucher, the Registrar passes the voucher to the device.

8. The device uses its factory-provisioned MASA trust anchors to verify the voucher signature, thereby ensuring that the voucher can be trusted.
9. The device uses Enrollment over Secure Transport (EST) to request new credentials.
10. The Registrar provisions network credentials to the device using EST. These network credentials get stored into secure storage on the device, e.g., as an LDevID.
11. The device uses its newly provisioned network credentials to connect to the network securely.
12. After the device is connected and begins operating on the network, the Continuous Authorization Service and the router make periodic asynchronous calls to each other that enable the Continuous Authorization Service to monitor device behavior and constrain communications to and from the device as needed in accordance with policy. In this manner, the Continuous Authorization Service interacts with the router on an ongoing basis to verify that the device and its operations continue to be authorized throughout the device’s tenure on the network.

This completes the network-layer onboarding process for Build 5 as well as the initialization of the Build 5 continuous authorization service. More details regarding the Build 5 implementation can be found at https://trustnetz.nqm.ai/docs/.

G.2.2 Build 5 Physical Architecture

Section 5.6 describes the physical architecture of Build 5.

Appendix H Factory Provisioning Process

H.1 Factory Provisioning Process

The Factory Provisioning Process creates and provisions a private key into the device's secure storage; generates and signs the device's certificate (when BRSKI is supported), generates the device's DPP URI (when Wi-Fi Easy Connect is supported), or generates other bootstrapping information (when other trusted network-layer onboarding protocols are supported); provisions the device's certificate, DPP URI, or other bootstrapping information onto the device; and sends the device's certificate, DPP URI, or other bootstrapping information to the manufacturer's database, which will eventually make this information available to the device owner to use during network-layer onboarding.

H.1.1 Device Birth Credential Provisioning Methods

There are various methods by which a device can be provisioned with its private key and bootstrapping information (e.g., its certificate, DPP URI, etc.) depending on how, where, and by what entity the public/private key pairs are generated [14]. Additional methods are also possible depending on how the device's certificate is provided to the manufacturer's database. The following are high-level descriptions of five potential methods for provisioning device birth credentials during various points in the device lifecycle. These methods are not intended to be exhaustive:

1. **Method 1: Key Pair Generated on IoT Device**
   Summary: Generate the private key on the device; device sends the device's bootstrapping information (e.g., the device's certificate or DPP URI) to the manufacturer's database. The steps for Method 1 are:
    a. The public/private key pair is generated on the device and stored in secure storage.
    b. The device generates and signs a CSR structure and sends the CSR to the manufacturer's IDevID CA, which sends a signed certificate (IDevID) back to the device.
    c. If BRSKI is being supported, the device loads the certificate (IDevID) into its secure storage; if Wi-Fi Easy Connect is being supported, the device creates a DPP URI and loads that into secure storage.
    d. The device sends the certificate or DPP URI to the manufacturer's database.

   One disadvantage of this method is that the device's random number generator is being relied upon to generate the key pair, and it is possible that a device's random number generator will not be as robust as the random number generator that would be included in an SE, for example. An advantage of this method is that the device's private key is not vulnerable to disclosure, assuming the device is equipped with a strong random number generator that is used for key generation and the private key is put into secure storage immediately upon generation.

2. **Method 2: Key Pair Generated in Secure Element**
   Summary: Generate the private key in a secure element on the device; IDevID CA provides the device certificate to the manufacturer's database. The steps for Method 2 are:
    a. The public/private key pair is generated within the device's SE.

   b. The device generates a CSR structure, the SE signs it, and the device sends the CSR to the manufacturer's IDevID CA, which sends a signed certificate (IDevID) back to the device.
   c. If BRSKI is being supported, the device loads the certificate (IDevID) into its secure storage; if Wi-Fi Easy Connect is being supported, the device creates a DPP URI and loads that into secure storage.
   d. The IDevID CA provides the certificate to the manufacturer's database. The manufacturer stores either the certificate (i.e., if BRSKI is being supported), or creates and stores a DPP URI (i.e., if Wi-Fi Easy Connect is being supported).

Method 2 is similar to Method 1 except that in method 2, the key pair is generated and stored in a secure element and the manufacturer's database receives the signed certificate directly from the CA (either via a push or a pull) rather than via the device. An advantage of method 2 is that the device's private key is not vulnerable to disclosure because secure elements are normally equipped with a strong random number generator and tamper-proof storage.

3. **Method 3: Key Pair Loaded into IoT Device**

Summary: Generate the private key in the device factory and load it onto the device. The steps for Method 3 are:

   a. The public/private key pairs and certificates are generated in advance at the device factory and recorded in the manufacturer's database.
   b. The public/private key pair and certificate are loaded onto the device at the device factory.

One advantage of this method is that there is no need to trust the random number generator on the device to generate strong public/private key pairs. However, the private keys may be vulnerable to disclosure during the period of time before they are provisioned into secure storage on the devices (and afterwards if they are not deleted once they have been copied into secure storage).

4. **Method 4: Key Pair Pre-Provisioned onto Secure Element**

Summary: Generate the private key in the SE and load the certificate on the device at the SE factory (SEF). The steps for Method 4 are:

   a. The public/private key pair and certificate are generated in advance in the SE at the SEF and the public key is recorded.
   b. The certificate is loaded onto the devices at the SEF.
   c. The certificates and the serial numbers of their corresponding devices are provided to the device manufacturer, and the device manufacturer can put them into the manufacturer database.
   d. The CA that signs the certificates that are generated and loaded onto the SEs may come from either the SEF or the device manufacturer. (Note: the CA is likely not located at the factory, which may be offshore.)

Additional trust anchors can also be loaded into the SE at the SEF (e.g., code signing keys, server public keys for TLS connections, etc.) As with methods 2 and 3, one advantage of this method (method 4) is that there is no need to trust the random number generator on the device to generate strong public/private key pairs because the random number generator on the SE is used instead. With this method, the security level of the manufacturer's factory does not need to be as high as that of the SEF because all key generation and certificate signing is performed at the SEF; the manufacturer can rely on the security of the SEF, which can be advantageous to the device manufacturer, assuming that the SEF is in fact secure.

5. **Method 5: Private Key Derived from Shared Seed**

Summary: The device's private key is derived from a shared seed. The steps for Method 5 are:

a. The chip vendor embeds a random number into each IoT device (e.g., this may be burned into fuses on the IoT device, inside the Trusted Execution Environment (TEE)).
b. The IoT device manufacturer gets a copy of this seed securely (e.g., on a USB device that is transported via trusted courier).
c. On first boot, the IoT device generates a private key from this seed.
d. The manufacturer uses the same seed to generate a public key and signs a certificate.

As with method 4, with this option (method 5), there is no need for the IoT device manufacturer to have a secure factory because the IoT device manufacturer may rely on the security of the chip manufacturer. However, the IoT device manufacturer must also rely on the security of the courier or other mechanism that is delivering the seed, and the IoT device manufacturer must ensure that the value of this seed is not disclosed.

H.2 Factory Provisioning Builds – General Provisioning Process

The Factory Provisioning Builds implemented as part of this project simulate activities performed during the IoT device manufacturing process to securely provision the device's birth credentials (i.e., its private key) into secure storage on the device and make the device's network-layer bootstrapping information available by enrolling the device's public key into a database that will make this public key accessible to the device owner in a form such as a certificate or DPP URI. The method used in the factory provisioning builds most closely resembles *Method 2: Key Pair Generated on IoT Devic*e, as described in Section H.1.1.

There are several different potential versions of the factory provisioning build architecture depending on whether the credentials being generated are designed to support BRSKI, Wi-Fi Easy Connect, Thread, or some other trusted network-layer onboarding protocol. For example, when BRSKI is being supported, the device bootstrapping information that is created takes the form of an 802.1AR certificate (IDevID); if DPP is supported, it takes the form of a DPP URI.

Because this project does not have access to a real factory or the tools necessary to provision birth credentials directly into device firmware, the factory builds simulate the firmware loading process by loading factory provisioning code into the IoT device (e.g., a Raspberry Pi device). This code plays the role of the factory in the builds by instructing the SE that is attached to the IoT device to generate the device's private key and bootstrapping information. Once the IoT device has been provisioned with its birth credentials in this manner, it can, in theory, be network-layer onboarded to one of the project build networks.

H.3 BRSKI Factory Provisioning Builds (NquiringMinds and SEALSQ)

Two variants of the BRSKI Factory Provisioning Build were implemented:

- **NquiringMinds and SEALSQ implementation** (first version): SEALSQ, a subsidiary of WISeKey, and NquiringMinds collaborated to implement one version of the BRSKI Factory Provisioning Build. This build is designed to provision birth credentials to a Raspberry Pi device that has an attached secure element provided by SEALSQ.
- **NquiringMinds and Infineon implementation** (second version): NquiringMinds implemented a second version of the BRSKI Factory Provisioning Build using an Infineon SE. This build is designed to provision birth credentials to a Raspberry Pi device that has an attached Infineon Optiga SLB 9670 TPM 2.0.

H.3.1 BRSKI Factory Provisioning Build Technologies

The general infrastructure for the first version of the BRSKI Factory Provisioning Build (i.e., the NquiringMinds and SEALSQ implementation) is provided by SEALSQ. The first version of the BRSKI Factory Provisioning Build infrastructure consists of:

- A SEALSQ VaultIC SE that is attached to the Raspberry Pi
- SEALSQ Factory Provisioning Code that is located on an SD card and that communicates with the chip in the SE to
    - create a P-256 Elliptic Curve public/private key pair within the SE,
    - construct a certificate signing request, and
    - store the certificate in the SE as well as send it to the manufacturer’s database
- SEALSQ INeS CMS CA, a certificate authority for signing the device’s birth certificate

As mentioned earlier, separate factory provisioning builds are required for each network-layer onboarding protocol being supported. A small amount of factory provisioning code is required to be customized for each build, depending on the onboarding protocol that is supported and how the bootstrapping information will be provided to the manufacturer. In this build, NquiringMinds provided this code and made it available to the Raspberry Pi IoT device by placing it on an SD card. (This could be either in a partition of the SD card that holds the device’s BRSKI onboarding software or on a separate SD card altogether).

Table H-1 lists the technologies used in the first version of the BRSKI Factory Provisioning Build. It lists the products used to instantiate each logical build component and the security function that the component provides. The components listed are logical. They may be combined in physical form, e.g., a single piece of hardware may both generate key pairs and provide secure storage.

Table H-1 First Version of the BRSKI Factory Provisioning Build Products and Technologies

Component	Product	Function
Key Pair Generation Component	SEALSQ VaultIC and associated provisioning code	Generates and installs the public/private key pair into secure storage. The VaultIC has a SP800-90B certified random number generator for key pair generation.

Component	Product	Function
		[15][16][17] Signs the certificate signing request that is sent to the CA.
Secure Storage	SEALSQ VaultIC	Storage on the IoT device that is designed to be protected from unauthorized access and capable of detecting attempts to hack or modify its contents. Used to generate, store, and process private keys, credentials, and other information that must be kept confidential.
General Factory Provisioning Instructions	SEALSQ Factory Provisioning Code	Creates a CSR associated with the key pair, installs the signed certificate into secure storage. Creates a record of devices that it has created and their certificates.
Build-specific Factory Provisioning Instructions	NquiringMinds Factory Provisioning Code	Sends device ownership information and the certificate received by the General Factory Provisioning code to the MASA.
Manufacturer Database	MASA	When devices are manufactured, device identity and bootstrapping information is stored here by the manufacturer. Eventually, this database makes the device's bootstrapping information available to the device owner. Device bootstrapping information is information that the device owner requires to perform trusted network-layer onboarding; for BRSKI, the bootstrapping information is a signed certificate that is sent to the MASA, along with information regarding the device's owner.
Certificate Authority (CA)	SEALSQ INeS CMS CA	Issues and signs certificates as needed.

The second version of the BRSKI Factory Provisioning Build (i.e., the NquiringMinds implementation with an Infineon SE) infrastructure consists of:

- An Infineon Optiga SLB 9670 TPM 2.0. that is attached to the Raspberry Pi
- Factory Provisioning Code written by NquiringMinds that is located on an SD card and that communicates with the chip in SE to
    - create a P-256 Elliptic Curve public/private key pair within the SE,
    - construct a certificate signing request, and
    - store the certificate in the SE as well as send it to the manufacturer's database
- NquiringMinds Manufacturer Provisioning Root (MPR) server, which signs the device's IDevID birth certificate. It sits in the cloud and is securely contacted using the keys in the Infineon Optiga secure element.

In this build, NquiringMinds provided all of the factory provisioning code and made it available to the Raspberry Pi IoT device by placing it on an SD card. (This could be either in a partition of the SD card that holds the device's BRSKI onboarding software or on a separate SD card altogether).

Table H-2 lists the technologies used in the second version of the BRSKI Factory Provisioning Build. It lists the products used to instantiate each logical build component and the security function that the component provides. The components listed are logical. They may be combined in physical form, e.g., a single piece of hardware may both generate key pairs and provide secure storage.

Table H-2 Second Version of the BRSKI Factory Provisioning Build Products and Technologies

Component	Product	Function
Key Pair Generation Component	Infineon TPM and associated provisioning code	Generates and installs the public/private key pair into secure storage. Signs the certificate signing request that is sent to the CA.
Secure Storage	Infineon TPM	Storage on the IoT device that is designed to be protected from unauthorized access and capable of detecting attempts to hack or modify its contents. Used to generate, store, and process private keys, credentials, and other information that must be kept confidential.
General Factory Provisioning Instructions	Infineon TPM-specific Factory Provisioning Code	Creates a CSR associated with the key pair, installs the signed certificate into secure storage. Creates a record of devices that it has created and their certificates.
Build-specific Factory Provisioning Instructions	Build-specific Factory Provisioning Code	Sends device ownership information and the signed certificate to the MASA.
Manufacturer Database	MASA	When devices are manufactured, device identity and bootstrapping information is stored here by the manufacturer. Eventually, this database makes the device's bootstrapping information available to the device owner. Device bootstrapping information is information that the device owner requires to perform trusted network-layer onboarding; for BRSKI, the bootstrapping information is a signed certificate that is sent to the MASA, along with information regarding the device's owner.
Certificate Authority (CA)	SEALSQ INeS CMS CA NquiringMinds On-premises CA	Issues and signs certificates as needed.

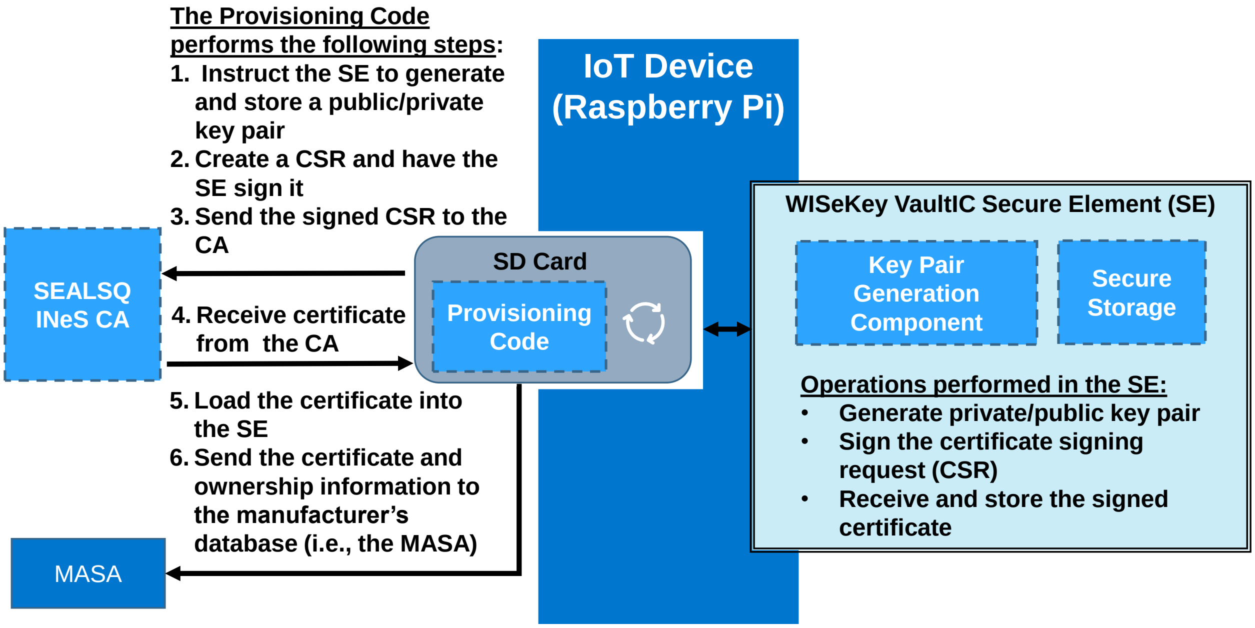
H.3.2 BRSKI Factory Provisioning Build Logical Architectures

Figure H-1 depicts the logical architecture of the first version of the BRSKI factory provisioning build (i.e., the NquiringMinds and SEALSQ implementation) and is annotated with the steps that are performed in this build to prepare IoT devices for network-layer onboarding using the BRSKI protocol. Figure H-1 shows a Raspberry Pi device with a SEALSQ VaultIC SE attached. An SD card that contains factory provisioning code provided by SEALSQ and NquiringMinds is also required. To perform factory provisioning using this build, insert the SD card into the Raspberry Pi, as depicted (or activate the code in the factory provisioning partition of the SD card that is already in the Raspberry Pi). The SEALSQ software will boot up and perform the following steps to simulate the activities of a factory:

1. Instruct the SE to generate and store a private/public key pair
2. Create a certificate signing request for this key pair and have the SE sign it
3. Send the signed CSR to the IDevID CA (i.e., to the INeS CA that is operated by SEALSQ)
4. Receive back the signed certificate from the CA
5. Load the certificate into the SE
6. Send the certificate (along with device ownership information) to the manufacturer's database, which in this case is the MASA that is trusted by the owner

This completes the steps performed as part of the first version of the BRSKI Factory Provisioning Build. Once complete, shipment of the device to its owner can be simulated by walking the device across the room in the NCCoE laboratory to the Build 5 (NquiringMinds) implementation and replacing the SD card that has the factory provisioning code on it with and SD card that has the BRSKI onboarding code on it. (Alternatively, if the factory provisioning code and the BRSKI onboarding code are stored in separate partitions of the same SD card, shipment of the device to its owner can be simulated by booting up the code in the onboarding partition.) Build 5 is designed to execute this BRSKI onboarding software, which onboards the device to the device owner's network by provisioning the device with an LDevID that will serve as its network-layer credential. Such successful network-layer onboarding of the newly provisioned device using the BRSKI protocol by Build 5 would serve to confirm that the first version of the BRSKI factory provisioning process successfully provisioned the device with its birth credentials. At the time of this writing, however, this confirmation process was not able to be performed. In order to securely network-layer onboard the newly provisioned Raspberry Pi using the BRSKI protocol, the Raspberry Pi's onboarding software would need to be written to use the private key stored in the SEALSQ secure element when running the BRSKI protocol. Such software was not yet available at the time of this publication. The BRSKI onboarding code on the Raspberry Pi does not currently use the private key stored in the SEALSQ SE. As a result, Build 5 was not able to onboard this factory Pi as a way of confirming that the first version of the BRSKI factory build process completed successfully. The repository that hosts the code for this implementation can be found here at the trustnetz-se Github repository.

Figure H-1 Logical Architecture of the First Version of the BRSKI Factory Provisioning Build



IoT Device (Raspberry Pi)
SEALSQ INeS CA
MASA
SD Card
Provisioning Code
WISeKey VaultIC Secure Element (SE)
Key Pair Generation Component
Secure Storage

The Provisioning Code performs the following steps:

1. Instruct the SE to generate and store a public/private key pair
2. Create a CSR and have the SE sign it
3. Send the signed CSR to the CA
4. Receive certificate from the CA
5. Load the certificate into the SE
6. Send the certificate and ownership information to the manufacturer's database (i.e., the MASA)

Operations performed in the SE:

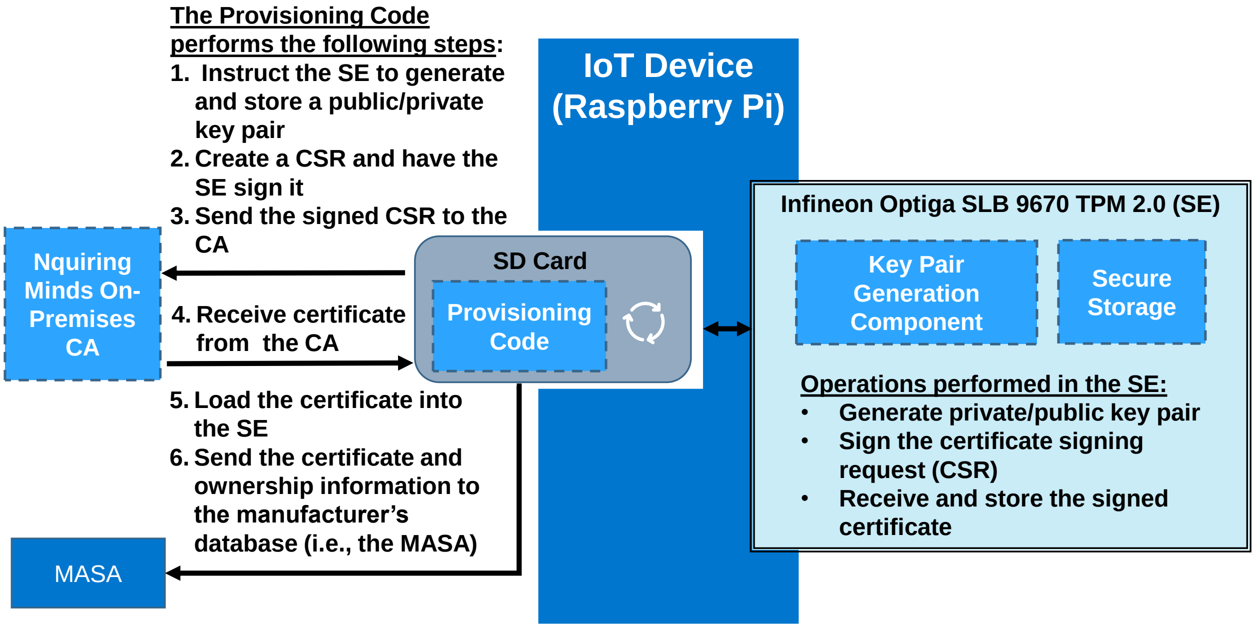
- Generate private/public key pair
- Sign the certificate signing request (CSR)
- Receive and store the signed certificate

Figure H-2 depicts the logical architecture of the second version of the BRSKI factory provisioning build and is annotated with the steps that are performed in this build to prepare IoT devices for network-layer onboarding using the BRSKI protocol. Figure H-2 shows a Raspberry Pi device with an Infineon Optiga SLB 9670 TPM 2.0 SE attached. An SD card that contains factory provisioning code provided by NquiringMinds is also required. To perform factory provisioning using this build, insert the SD card into the Raspberry Pi, as depicted (or activate the code in the factory provisioning partition of the SD card that is already in the Raspberry Pi). The factory provisioning code software will boot up and perform the following steps to simulate the activities of a factory:

1. Instruct the Infineon SE to generate and store a private/public key pair
2. Create a certificate signing request for this key pair and have the SE sign it
3. Send the signed CSR to the IDevID CA (i.e., to the NquiringMinds on-premises CA/Manufacturer Provisioning Root)
4. Receive back the signed certificate from the CA
5. Load the certificate into the SE
6. Send the certificate (along with device ownership information) to the manufacturer's database, which in this case is the MASA that is trusted by the owner

This completes the steps performed as part of the second version of the BRSKI Factory Provisioning Build. Once complete, shipment of the device to its owner can be simulated by walking the device across the room in the NCCoE laboratory to the Build 5 (NquiringMinds) implementation and replacing the SD card that has the factory provisioning code on it with and SD card that has the BRSKI onboarding code on it. (Alternatively, if the factory provisioning code and the BRSKI onboarding code are stored in separate partitions of the same SD card, shipment of the device to its owner can be simulated by booting up the code in the onboarding partition.) Build 5 executes a modification of the BRSKI onboarding software that has been modified to use the IDevID resident on the Infineon TPM throughout the protocol flow, ensuring the device's IDevID's private key is never made public and never leaves the secure element. Specifically, the critical signing operations and the TLS negotiation steps are fully secured by the SE. The full BRSKI onboarding flow provisions a new LDevID onto the device. This LDevID provides the secure method for the device to connect to the domain owner's network. This successful network-layer onboarding of the IoT device by Build 5 serves as confirmation that the second version of the BRSKI factory provisioning process successfully provisioned the device with its birth credentials.

Figure H-2 Logical Architecture of the Second Version of the BRSKI Factory Provisioning Build



The Provisioning Code performs the following steps:
1. Instruct the SE to generate and store a public/private key pair
2. Create a CSR and have the SE sign it
3. Send the signed CSR to the CA
4. Receive certificate from the CA
5. Load the certificate into the SE
6. Send the certificate and ownership information to the manufacturer's database (i.e., the MASA)

Operations performed in the SE:
- Generate private/public key pair
- Sign the certificate signing request (CSR)
- Receive and store the signed certificate

IoT Device (Raspberry Pi)
Nquiring Minds On-Premises CA
MASA
SD Card
Provisioning Code
Infineon Optiga SLB 9670 TPM 2.0 (SE)
Key Pair Generation Component
Secure Storage

H.3.3 BRSKI Factory Provisioning Build Physical Architectures

Section 5.6.1 describes the physical architecture of the BRSKI Factory Provisioning Builds.

H.4 Wi-Fi Easy Connect Factory Provisioning Build (SEALSQ and Aruba/HPE)

SEALSQ, a subsidiary of WISeKey, and Aruba/HPE implemented a Wi-Fi Easy Connect Factory Provisioning Build. This build is designed to provision birth credentials to a Raspberry Pi device that has an attached secure element provided by SEALSQ.

H.4.1 Wi-Fi Easy Connect Factory Provisioning Build Technologies

The general infrastructure for the Wi-Fi Easy Connect Factory Provisioning Build is provided by SEALSQ. The Wi-Fi Easy Connect Factory Provisioning Build infrastructure consists of:

- A SEALSQ VaultIC SE that is attached to the Raspberry Pi

- SEALSQ Factory Provisioning Code that is located on an SD card and that communicates with the chip in the SE to:
    - create a P-256 Elliptic Curve public/private key pair within the SE,
    - use the public key to construct a DPP URI
    - export the DPP URI and convert it into a QR code

Table H-3 lists the technologies used in the Wi-Fi Easy Connect Factory Provisioning Build. It lists the products used to instantiate each logical build component and the security function that the component provides. The components listed are logical. They may be combined in physical form, e.g., a single piece of hardware may both generate key pairs and provide secure storage.

Table H-3 Wi-Fi Easy Connect Factory Provisioning Build Products and Technologies

Component	Product	Function
Key Pair Generation Component	SEALSQ VaultIC and associated provisioning code	Generates and installs the public/private key pair into secure storage. The VaultIC has a SP800-90B certified random number generator for key pair generation. [17]
Secure Storage	SEALSQ VaultIC	Storage on the IoT device that is designed to be protected from unauthorized access and capable of detecting attempts to hack or modify its contents. Used to generate, store, and process private keys, credentials, and other information that must be kept confidential.
General Factory Provisioning Instructions	SEALSQ Factory Provisioning Code	Creates a public/private key pair.
Build-specific Factory Provisioning Instructions	Aruba/HPE Factory Provisioning Code	Uses the public key to create a DPP URI. Exports the DPP URI and converts it into a QR code.
Manufacturer Database	Manufacturer cloud or imprint on device	The DPP URI information is stored in the QR code and is the mechanism for conveying the device’s bootstrapping information to the device owner.

H.4.2 Wi-Fi Easy Connect Factory Provisioning Build Logical Architecture

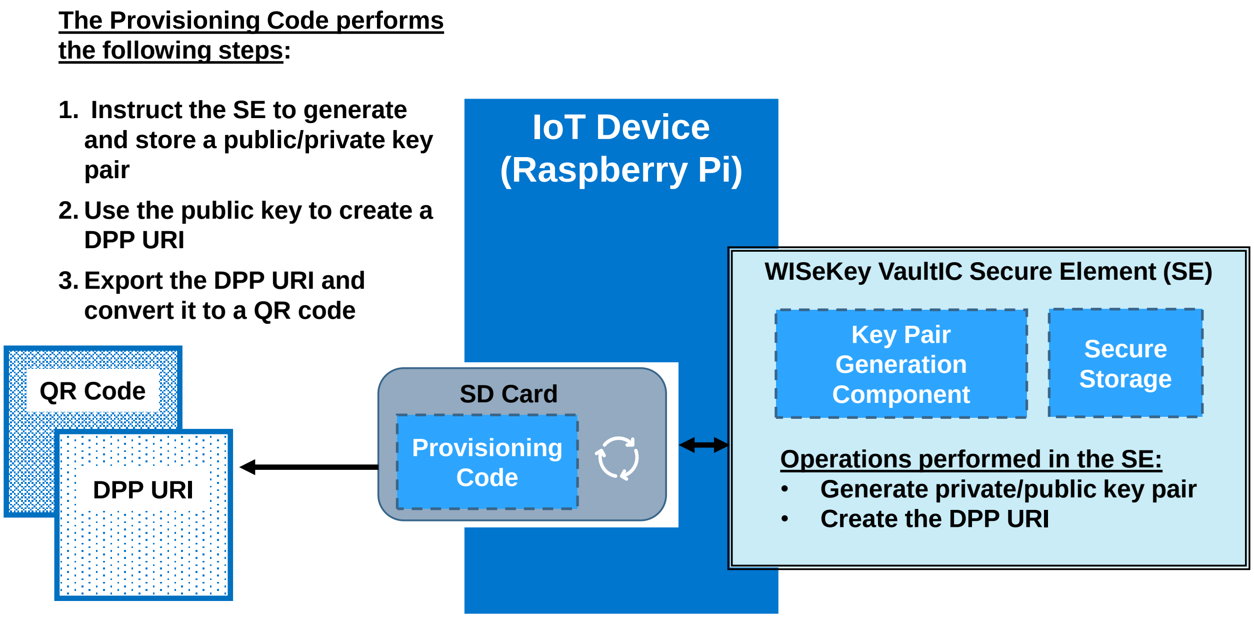
Figure H-3 depicts the logical architecture of the Wi-Fi Easy Connect factory provisioning build and is annotated with the steps that are performed in this build to prepare Raspberry Pi IoT devices for network-layer onboarding using the Wi-Fi Easy Connect protocol. Figure H-3 shows a Raspberry Pi device with a SEALSQ VaultIC SE attached. Factory provisioning code provided by SEALSQ and Aruba/HPE must also be loaded. In Figure H-3, this code is shown as being on an SD card. The factory provisioning software will boot up and perform the following steps to simulate the activities of a factory:

1. Instruct the SE to generate and store a private/public key pair
2. Use the public key to create a DPP URI

3. Export the DPP URI and convert it into a QR code

This completes the steps performed as part of the Wi-Fi Easy Connect Factory Provisioning Build. Once complete, shipment of the device to its owner can be simulated by walking the device across the room in the NCCoE laboratory to the Build 1 (Aruba/HPE) implementation. Build 1 uses the Wi-Fi Easy Connect protocol to network-layer onboard the device to the device owner's network by provisioning the device with connector that will serve as its network-layer credential. Successful network-layer onboarding of the newly provisioned device using the Wi-Fi Easy Connect protocol by Build 1 would serve to confirm that the Wi-Fi Easy Connect factory provisioning process correctly provisioned the device with its birth credentials. At the time of this writing, however, this confirmation process was not able to be performed. In order to securely network-layer onboard the newly provisioned Raspberry Pi using the Wi-Fi Easy Connect protocol, the Raspberry Pi would need to be equipped with a firmware image that uses the private key stored in the secure element when running the Wi-Fi Easy Connect protocol. Such firmware was not yet available at the time of this publication. The Wi-Fi Easy Connect code on the Raspberry Pi does not use the private key stored in the SE at this time. Confirmation that the factory build process completed successfully is limited to inspection of the .PNG file and .URI file that were created to display the QR Code and the device's DPP URI, respectively.

Figure H-3 Logical Architecture of the Wi-Fi Easy Connect Factory Provisioning Build


The Provisioning Code performs the following steps:
1. Instruct the SE to generate and store a public/private key pair
2. Use the public key to create a DPP URI
3. Export the DPP URI and convert it to a QR code
IoT Device (Raspberry Pi)
WISeKey VaultIC Secure Element (SE)
Key Pair Generation Component
Secure Storage
QR Code
DPP URI
SD Card
Provisioning Code
Operations performed in the SE:
• Generate private/public key pair
• Create the DPP URI

H.4.3 Wi-Fi Easy Connect Factory Provisioning Build Physical Architecture

Section 5.2.1 describes the physical architecture of the Factory Provisioning Build.

Appendix I References

[1] L. S. Vailshery, "Number of Internet of Things (IoT) connected devices worldwide from 2019 to 2023, with forecasts from 2022 to 2030," Statista, July 2023. Available: https://www.statista.com/statistics/1183457/iot-connected-devices-worldwide/.

[2] S. Symington, W. Polk, and M. Souppaya, *Trusted Internet of Things (IoT) Device Network-Layer Onboarding and Lifecycle Management (Draft)*, National Institute of Standards and Technology (NIST) Draft Cybersecurity White Paper, Gaithersburg, MD, Sept. 2020, 88 pp. https://doi.org/10.6028/NIST.CSWP.09082020-draft.

[3] E. Lear, R. Droms, and D. Romascanu, *Manufacturer Usage Description Specification,* IETF Request for Comments (RFC) 8520, March 2019. Available: https://tools.ietf.org/html/rfc8520.

[4] M. Souppaya et al, *Securing Small-Business and Home Internet of Things (IoT) Devices: Mitigating Network-Based Attacks Using Manufacturer Usage Description (MUD)*, National Institute of Standards and Technology (NIST) Special Publication (SP) 1800-15, Gaithersburg, Md., May 2021, 438 pp. Available: https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.1800-15.pdf.

[5] "National Cybersecurity Center of Excellence (NCCoE) Trusted Internet of Things (IoT) Device Network-Layer Onboarding and Lifecycle Management," Federal Register Vol. 86, No. 204, October 26, 2021, pp. 59149-59152. Available: https://www.federalregister.gov/documents/2021/10/26/2021-23293/national-cybersecurity-center-of-excellence-nccoe-trusted-internet-of-things-iot-device.

[6] Wi-Fi Alliance, *Wi-Fi Easy Connect™ Specification Version 3.0*, 2022. Available: https://www.wi-fi.org/system/files/Wi-Fi_Easy_Connect_Specification_v3.0.pdf.

[7] M. Pritikin, M. Richardson, T.T.E. Eckert, M.H. Behringer, and K.W. Watsen, *Bootstrapping Remote Secure Key Infrastructure (BRSKI)*, IETF Request for Comments (RFC) 8995, October 2021. Available: https://datatracker.ietf.org/doc/rfc8995/.

[8] Thread 1.1.1 Specification, February 13, 2017.

[9] OpenThread Released by Google. Available: https://openthread.io/.

[10] O. Friel, E. Lear, M. Pritikin, and M. Richardson, *BRSKI over IEEE 802.11*, IETF Internet-Draft (Individual), July 2018. Available: https://datatracker.ietf.org/doc/draft-friel-brski-over-802dot11/01/.

[11] NIST. *The NIST Cybersecurity Framework (CSF) 2.0*. Available: https://doi.org/10.6028/NIST.CSWP.29.

[12] *IEEE Standard for Local and Metropolitan Area Networks - Secure Device Identity*, IEEE Std 802.1AR-2018 (Revision of IEEE Std 802.1AR-2009), 2 Aug. 2018, 73 pp. Available: https://ieeexplore.ieee.org/document/8423794.

[13] F. Stajano and R. Anderson, *The Resurrecting Duckling: Security Issues for Ad-hoc Wireless Networks*, B. Christianson, B. Crispo and M. Roe (Eds.). Security Protocols, 7th International Workshop Proceedings, Lecture Notes in Computer Science, 1999. Springer-Verlag Berlin Heidelberg 1999. Available: https://www.cl.cam.ac.uk/~fms27/papers/1999-StajanoAnd-duckling.pdf.

[14] M. Richardson, *A Taxonomy of operational security considerations for manufacturer installed keys and Trust Anchors*, IETF Internet-Draft (Individual), November 2022. Available: https://datatracker.ietf.org/doc/draft-richardson-t2trg-idevid-considerations/.

[15] Certificate #4302, Cryptographic Module Validation Program, NIST Computer Security Resource Center. Available: https://csrc.nist.gov/projects/cryptographic-module-validation-program/certificate/4302.

[16] Certificate #4303, Cryptographic Module Validation Program, NIST Computer Security Resource Center. Available: https://csrc.nist.gov/projects/cryptographic-module-validation-program/certificate/4303.

[17] Entropy Certificate #E2, Cryptographic Module Validation Program, NIST Computer Security Resource Center. Available: https://csrc.nist.gov/projects/cryptographic-module-validation-program/entropy-validations/certificate/2.

NIST SPECIAL PUBLICATION 1800-36C

Trusted Internet of Things (IoT) Device Network-Layer Onboarding and Lifecycle Management:

Enhancing Internet Protocol-Based IoT Device and Network Security

Volume C:
How-To Guides

Murugiah Souppaya
Paul Watrobski
National Institute of Standards and Technology
Gaithersburg, Maryland

Chelsea Deane
Joshua Klosterman
Blaine Mulugeta
Charlie Rearick
Susan Symington
The MITRE Corporation
McLean, Virginia

Dan Harkins
Danny Jump
Aruba, a Hewlett Packard
Enterprise Company
San Jose, California

Andy Dolan
Kyle Haefner
Craig Pratt
Darshak Thakore
CableLabs
Louisville, Colorado

Nick Allot
Ashley Setter
NquiringMinds
Southampton, United Kingdom

May 2024

DRAFT

This publication is available free of charge from
https://www.nccoe.nist.gov/projects/trusted-iot-device-network-layer-onboarding-and-lifecycle-management

DISCLAIMER

Certain commercial entities, equipment, products, or materials may be identified by name or company logo or other insignia in order to acknowledge their participation in this collaboration or to describe an experimental procedure or concept adequately. Such identification is not intended to imply special status or relationship with NIST or recommendation or endorsement by NIST or NCCoE; neither is it intended to imply that the entities, equipment, products, or materials are necessarily the best available for the purpose.

While NIST and the NCCoE address goals of improving management of cybersecurity and privacy risk through outreach and application of standards and best practices, it is the stakeholder's responsibility to fully perform a risk assessment to include the current threat, vulnerabilities, likelihood of a compromise, and the impact should the threat be realized before adopting cybersecurity measures such as this recommendation.

National Institute of Standards and Technology Special Publication 1800-36C, Natl. Inst. Stand. Technol. Spec. Publ. 1800-36C, 55 pages, May 2024, CODEN: NSPUE2

FEEDBACK

You can improve this guide by contributing feedback. As you review and adopt this solution for your own organization, we ask you and your colleagues to share your experience and advice with us.

Comments on this publication may be submitted to: iot-onboarding@nist.gov.

Public comment period: May 31, 2024 through July 30, 2024

All comments are subject to release under the Freedom of Information Act.

National Cybersecurity Center of Excellence
National Institute of Standards and Technology
100 Bureau Drive
Mailstop 2002
Gaithersburg, MD 20899
Email: nccoe@nist.gov

NATIONAL CYBERSECURITY CENTER OF EXCELLENCE

The National Cybersecurity Center of Excellence (NCCoE), a part of the National Institute of Standards and Technology (NIST), is a collaborative hub where industry organizations, government agencies, and academic institutions work together to address businesses' most pressing cybersecurity issues. This public-private partnership enables the creation of practical cybersecurity solutions for specific industries, as well as for broad, cross-sector technology challenges. Through consortia under Cooperative Research and Development Agreements (CRADAs), including technology partners—from Fortune 50 market leaders to smaller companies specializing in information technology security—the NCCoE applies standards and best practices to develop modular, adaptable example cybersecurity solutions using commercially available technology. The NCCoE documents these example solutions in the NIST Special Publication 1800 series, which maps capabilities to the NIST Cybersecurity Framework and details the steps needed for another entity to re-create the example solution. The NCCoE was established in 2012 by NIST in partnership with the State of Maryland and Montgomery County, Maryland.

To learn more about the NCCoE, visit https://www.nccoe.nist.gov/. To learn more about NIST, visit https://www.nist.gov.

NIST CYBERSECURITY PRACTICE GUIDES

NIST Cybersecurity Practice Guides (Special Publication 1800 series) target specific cybersecurity challenges in the public and private sectors. They are practical, user-friendly guides that facilitate the adoption of standards-based approaches to cybersecurity. They show members of the information security community how to implement example solutions that help them align with relevant standards and best practices, and provide users with the materials lists, configuration files, and other information they need to implement a similar approach.

The documents in this series describe example implementations of cybersecurity practices that businesses and other organizations may voluntarily adopt. These documents do not describe regulations or mandatory practices, nor do they carry statutory authority.

KEYWORDS

application-layer onboarding; bootstrapping; Internet of Things (IoT); Manufacturer Usage Description (MUD); network-layer onboarding; onboarding; Wi-Fi Easy Connect.

ACKNOWLEDGMENTS

We are grateful to the following individuals for their generous contributions of expertise and time.

Name	Organization
Amogh Guruprasad Deshmukh	Aruba, a Hewlett Packard Enterprise company
Bart Brinkman	Cisco
Eliot Lear	Cisco
Peter Romness	Cisco
Tyler Baker	Foundries.io
George Grey	Foundries.io
David Griego	Foundries.io
Fabien Gremaud	Kudelski IoT
Brecht Wyseur	Kudelski IoT
Faith Ryan	The MITRE Corporation
Toby Ealden	NquiringMinds
John Manslow	NquiringMinds
Antony McCaigue	NquiringMinds
Alexandru Mereacre	NquiringMinds
Loic Cavaille	NXP Semiconductors
Mihai Chelalau	NXP Semiconductors
Julien Delplancke	NXP Semiconductors
Anda-Alexandra Dorneanu	NXP Semiconductors
Todd Nuzum	NXP Semiconductors

Name	Organization
Nicusor Penisoara	NXP Semiconductors
Laurentiu Tudor	NXP Semiconductors
Michael Richardson	Sandelman Software Works
Karen Scarfone	Scarfone Cybersecurity
Steve Clark	SEALSQ, a subsidiary of WISeKey
Pedro Fuentes	SEALSQ, a subsidiary of WISeKey
Gweltas Radenac	SEALSQ, a subsidiary of WISeKey
Kalvin Yang	SEALSQ, a subsidiary of WISeKey
Mike Dow	Silicon Labs
Steve Egerter	Silicon Labs

The Technology Partners/Collaborators who participated in this build submitted their capabilities in response to a notice in the Federal Register. Respondents with relevant capabilities or product components were invited to sign a Cooperative Research and Development Agreement (CRADA) with NIST, allowing them to participate in a consortium to build this example solution. We worked with:

Technology Collaborators

Aruba, a Hewlett Packard Enterprise company
CableLabs
Cisco
Foundries.io
Kudelski IoT
NquiringMinds
NXP Semiconductors
Open Connectivity Foundation (OCF)
Sandelman Software Works
SEALSQ, a subsidiary of WISeKey
Silicon Labs

DOCUMENT CONVENTIONS

The terms “shall” and “shall not” indicate requirements to be followed strictly to conform to the publication and from which no deviation is permitted. The terms “should” and “should not” indicate that among several possibilities, one is recommended as particularly suitable without mentioning or excluding others, or that a certain course of action is preferred but not necessarily required, or that (in the negative form) a certain possibility or course of action is discouraged but not prohibited. The terms "may" and "need not" indicate a course of action permissible within the limits of the publication. The terms "can" and "cannot" indicate a possibility and capability, whether material, physical, or causal.

CALL FOR PATENT CLAIMS

This public review includes a call for information on essential patent claims (claims whose use would be required for compliance with the guidance or requirements in this Information Technology Laboratory (ITL) draft publication). Such guidance and/or requirements may be directly stated in this ITL Publication or by reference to another publication. This call also includes disclosure, where known, of the existence of pending U.S. or foreign patent applications relating to this ITL draft publication and of any relevant unexpired U.S. or foreign patents.

ITL may require from the patent holder, or a party authorized to make assurances on its behalf, in written or electronic form, either:

a) assurance in the form of a general disclaimer to the effect that such party does not hold and does not currently intend holding any essential patent claim(s); or

b) assurance that a license to such essential patent claim(s) will be made available to applicants desiring to utilize the license for the purpose of complying with the guidance or requirements in this ITL draft publication either:

1. under reasonable terms and conditions that are demonstrably free of any unfair discrimination; or
2. without compensation and under reasonable terms and conditions that are demonstrably free of any unfair discrimination.

Such assurance shall indicate that the patent holder (or third party authorized to make assurances on its behalf) will include in any documents transferring ownership of patents subject to the assurance, provisions sufficient to ensure that the commitments in the assurance are binding on the transferee, and that the transferee will similarly include appropriate provisions in the event of future transfers with the goal of binding each successor-in-interest.

The assurance shall also indicate that it is intended to be binding on successors-in-interest regardless of whether such provisions are included in the relevant transfer documents.

Such statements should be addressed to: iot-onboarding@nist.gov.

Contents

1 Introduction 1

1.1 How to Use This Guide 1

1.2 Build Overview..3

1.2.1 Reference Architecture Summary 3

1.2.2 Physical Architecture Summary..3

1.3 Typographic Conventions..7

2 Build 1 (Wi-Fi Easy Connect, Aruba/HPE)..7

2.1 Aruba Central/Hewlett Packard Enterprise (HPE) Cloud..7

2.2 Aruba Wireless Access Point 7

2.2.1 Wi-Fi Network Setup and Configuration 8

2.2.2 Wi-Fi Easy Connect Configuration 9

2.3 Cisco Catalyst 3850-S Switch 9

2.3.1 Configuration 10

2.4 Aruba User Experience Insight (UXI) Sensor 10

2.4.1 Configuration 10

2.5 Raspberry Pi..10

2.5.1 Configuration 11

2.5.2 DPP Onboarding 11

2.6 Certificate Authority..13

2.6.1 Private Certificate Authority..13

2.6.2 SEALSQ INeS..17

2.7 UXI Cloud 18

2.8 Wi-Fi Easy Connect Factory Provisioning Build 18

2.8.1 SEALSQ VaultIC Secure Element 18

3 Build 2 (Wi-Fi Easy Connect, CableLabs, OCF) 19

3.1 CableLabs Platform Controller 19

3.1.1 Operation and Demonstration 20

3.2 CableLabs Custom Connectivity Gateway..20

3.2.1 Installation and Configuration 20

3.2.2 Integration with CableLabs Platform Controller 20

3.2.3 Operation and Demonstration 20

3.3 Reference Clients/IoT Devices 20
3.3.1 Installation and Configuration 20
3.3.2 Operation and Demonstration 20
4 Build 3 (BRSKI, Sandelman Software Works) 21
4.1 Onboarding Router/Join Proxy 21
4.1.1 Setup and Configuration 21
4.2 Minerva Join Registrar Coordinator 21
4.2.1 Setup and Configuration 21
4.3 Reach Pledge Simulator 22
4.3.1 Setup and Configuration 22
4.4 Serial Console Server 23
4.5 Minerva Highway MASA Server 23
4.5.1 Setup and Configuration 23
5 Build 4 (Thread, Silicon Labs, Kudelski IoT) 24
5.1 Open Thread Border Router 24
5.1.1 Installation and Configuration 24
5.1.2 Operation and Demonstration 24
5.2 Silicon Labs Dev Kit (BRD2601A) 25
5.2.1 Setup and Configuration 25
5.3 Kudelski keySTREAM Service 28
5.3.1 Setup and Configuration 28
5.4 AWS IoT Core 30
5.4.1 Setup and Configuration 30
5.4.2 Testing 35
6 Build 5 (BRSKI over Wi-Fi, NquiringMinds) 36
6.1 Pledge 36
6.1.1 Installation and Configuration 37
6.1.2 Operation and Demonstration 37
6.2 Router and Logical Services 37
6.2.1 Installation and Configuration 37
6.2.2 Logical services 38
6.3 Onboarding Demonstration 41
6.3.1 Prerequisites 41

6.3.2 Onboarding Demonstration 42
6.3.3 Continuous Assurance Demonstration 42
6.4 BRSKI Factory Provisioning Build 42
6.4.1 Pledge 43
6.4.2 Installation and Configuration 43
6.4.3 Operation and Demonstration 43

List of Figures

Figure 1-1 NCCoE IoT Onboarding Laboratory Physical Architecture 5
Figure 6-1 Logical Services for Build 5 38
Figure 6-2 Diagram of Physical/Logical Components Used to Demonstrate BRSKI Flow 42

1 Introduction

The following volumes of this guide show information technology (IT) professionals and security engineers how we implemented these example solutions. We cover all of the products employed in this reference design. We do not re-create the product manufacturers' documentation, which is presumed to be widely available. Rather, these volumes show how we incorporated the products together in our environment.

Note: These are not comprehensive tutorials. There are many possible service and security configurations for these products that are out of scope for this reference design.

1.1 How to Use This Guide

This NIST Cybersecurity Practice Guide demonstrates a standards-based reference design for implementing trusted IoT device network-layer onboarding and lifecycle management and describes various example implementations of this reference design. Each of these implementations, which are known as *builds,* is standards-based and is designed to help provide assurance that networks are not put at risk as new IoT devices are added to them and to help safeguard IoT devices from connecting to unauthorized networks. The reference design described in this practice guide is modular and can be deployed in whole or in part, enabling organizations to incorporate trusted IoT device network-layer onboarding and lifecycle management into their legacy environments according to goals that they have prioritized based on risk, cost, and resources.

NIST is adopting an agile process to publish this content. Each volume is being made available as soon as possible rather than delaying release until all volumes are completed.

This guide contains five volumes:

- NIST Special Publication (SP) 1800-36A: *Executive Summary* – why we wrote this guide, the challenge we address, why it could be important to your organization, and our approach to solving this challenge
- NIST SP 1800-36B: *Approach, Architecture, and Security Characteristics* – what we built and why
- NIST SP 1800-36C: *How-To Guides* – instructions for building the example implementations, including all the security-relevant details that would allow you to replicate all or parts of this project **(you are here)**
- NIST SP 1800-36D: *Functional Demonstrations* – use cases that have been defined to showcase trusted IoT device network-layer onboarding and lifecycle management security capabilities and the results of demonstrating these use cases with each of the example implementations
- NIST SP 1800-36E: *Risk and Compliance Management* – risk analysis and mapping of trusted IoT device network-layer onboarding and lifecycle management security characteristics to cybersecurity standards and recommended practices

Depending on your role in your organization, you might use this guide in different ways:

Business decision makers, including chief security and technology officers, will be interested in the *Executive Summary, NIST SP 1800-36A*, which describes the following topics:

- challenges that enterprises face in migrating to the use of trusted IoT device network-layer onboarding
- example solutions built at the NCCoE
- benefits of adopting the example solution

Technology or security program managers who are concerned with how to identify, understand, assess, and mitigate risk will be interested in *NIST SP 1800-36B*, which describes what we did and why.

Also, Section 4 of *NIST SP 1800-36E* will be of particular interest. Section 4, *Mappings*, maps logical components of the general trusted IoT device network-layer onboarding and lifecycle management reference design to security characteristics listed in various cybersecurity standards and recommended practices documents, including *Framework for Improving Critical Infrastructure Cybersecurity* (NIST Cybersecurity Framework) and *Security and Privacy Controls for Information Systems and Organizations* (NIST SP 800-53).

You might share the *Executive Summary, NIST SP 1800-36A*, with your leadership team members to help them understand the importance of using standards-based trusted IoT device network-layer onboarding and lifecycle management implementations.

IT professionals who want to implement similar solutions will find the whole practice guide useful. You can use the how-to portion of the guide, *NIST SP 1800-36C*, to replicate all or parts of the builds created in our lab. The how-to portion of the guide provides specific product installation, configuration, and integration instructions for implementing the example solution. We do not re-create the product manufacturers' documentation, which is generally widely available. Rather, we show how we incorporated the products together in our environment to create an example solution. Also, you can use *Functional Demonstrations, NIST SP 1800-36D*, which provides the use cases that have been defined to showcase trusted IoT device network-layer onboarding and lifecycle management security capabilities and the results of demonstrating these use cases with each of the example implementations. Finally, *NIST SP 1800-36E* will be helpful in explaining the security functionality that the components of each build provide.

This guide assumes that IT professionals have experience implementing security products within the enterprise. While we have used a suite of commercial products to address this challenge, this guide does not endorse these particular products. Your organization can adopt this solution or one that adheres to these guidelines in whole, or you can use this guide as a starting point for tailoring and implementing parts of a trusted IoT device network-layer onboarding and lifecycle management solution. Your organization's security experts should identify the products that will best integrate with your existing tools and IT system infrastructure. We hope that you will seek products that are congruent with applicable standards and recommended practices.

A NIST Cybersecurity Practice Guide does not describe "the" solution, but example solutions. We seek feedback on the publication's contents and welcome your input. Comments, suggestions, and success stories will improve subsequent versions of this guide. Please contribute your thoughts to iot-onboarding@nist.gov.

1.2 Build Overview

This NIST Cybersecurity Practice Guide addresses the challenge of network-layer onboarding using standards-based protocols to perform trusted network-layer onboarding of an IoT device. Each build demonstrates one or more of these capabilities:

- Trusted Network-Layer Onboarding: providing the device with its unique network credentials over an encrypted channel
- Network Re-Onboarding: performing trusted network-layer onboarding of the device again, after device reset
- Network Segmentation: assigning a device to a particular local network segment to prevent it from communicating with other network components, as determined by enterprise policy
- Trusted Application-Layer Onboarding: providing the device with application-layer credentials over an encrypted channel after completing network-layer onboarding
- Ongoing Device Authorization: continuously monitoring the device on an ongoing basis, providing policy-based assurance and authorization checks on the device throughout its lifecycle
- Device Communications Intent Enforcement: Secure conveyance of device communications intent information, combined with enforcement of it, to ensure that IoT devices are constrained to sending and receiving only those communications that are explicitly required for each device to fulfill its purpose

Five builds that will serve as examples of how to onboard IoT devices using the protocols described in NIST SP 1800-36B, as well as the factory provisioning builds, are being implemented and will be demonstrated as part of this project. The remainder of this practice guide provides step-by-step instructions on how to reproduce all builds.

1.2.1 Reference Architecture Summary

The builds described in this document are instantiations of the trusted network-layer onboarding and lifecycle management logical reference architecture that is described in NIST SP 1800-36B. This architecture is organized according to five high-level processes: Device Manufacture and Factory Provisioning, Device Ownership and Bootstrapping Information Transfer, Trusted Network-Layer Onboarding, Trusted Application-Layer Onboarding, and Continuous Verification. For a full explanation of the architecture, please see NIST SP 1800-36B: *Approach, Architecture, and Security Characteristics*.

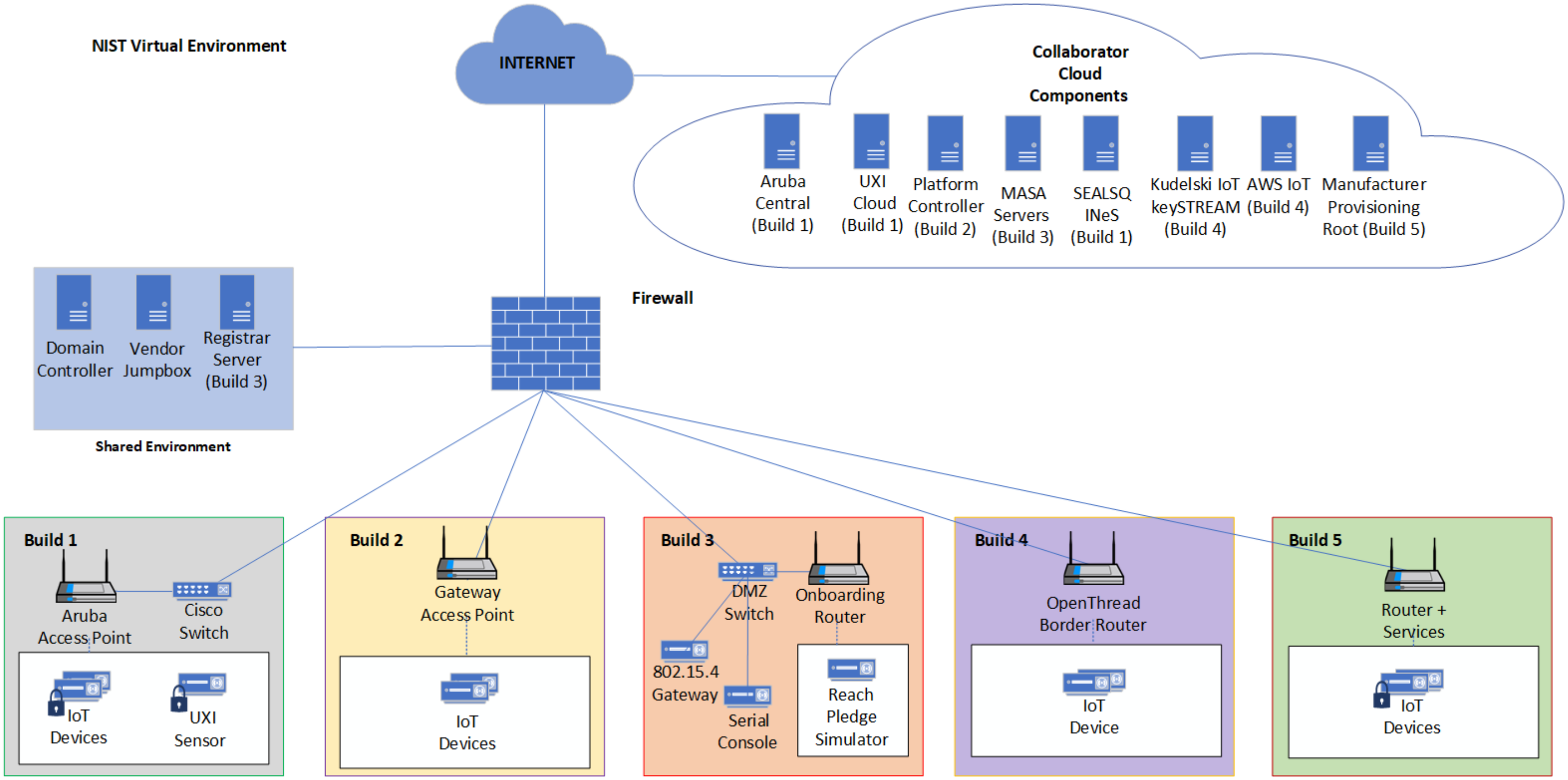
1.2.2 Physical Architecture Summary

Figure 1-1 depicts the high-level physical architecture of the NCCoE IoT Onboarding laboratory environment in which the five trusted IoT device network-layer onboarding project builds and the two factory provisioning builds are being implemented. The NCCoE provides virtual machine (VM) resources and physical infrastructure for the IoT Onboarding lab. As depicted, the NCCoE IoT Onboarding laboratory hosts collaborator hardware and software for the builds. The NCCoE also provides connectivity from the IoT Onboarding lab to the NIST Data Center, which provides connectivity to the internet and public IP spaces (both IPv4 and IPv6). Access to and from the NCCoE network is protected by a firewall.

Access to and from the IoT Onboarding lab is protected by a pfSense firewall, represented by the brick box icon in Figure 1-1. This firewall has both IPv4 and IPv6 (dual stack) configured. The IoT Onboarding lab network infrastructure includes a shared virtual environment that houses a domain controller and a vendor jumpbox. These components are used across builds where applicable. It also contains five independent virtual local area networks (VLANs), each of which houses a different trusted network-layer onboarding build.

The IoT Onboarding laboratory network has access to cloud components and services provided by the collaborators, all of which are available via the internet. These components and services include Aruba Central and the UXI Cloud (Build 1), SEALSQ INeS (Build 1), Platform Controller (Build 2), a MASA server (Build 3), Kudelski IoT keySTREAM application-layer onboarding service and AWS IoT (Build 4), and a Manufacturer Provisioning Root (Build 5).

300 **Figure 1-1 NCCoE IoT Onboarding Laboratory Physical Architecture**



All five network-layer onboarding laboratory environments, as depicted in the diagram, have been installed:

- Build 1 (i.e., the Wi-Fi Easy Connect, Aruba/HPE build) network infrastructure within the NCCoE lab consists of two components: the Aruba Access Point and the Cisco Switch. Build 1 also requires support from Aruba Central for network-layer onboarding and the UXI Cloud for application-layer onboarding. These components are in the cloud and accessed via the internet. The IoT devices that are onboarded using Build 1 include the UXI Sensor and the Raspberry Pi.
- Build 2 (i.e., the Wi-Fi Easy Connect, CableLabs, OCF build) network infrastructure within the NCCoE lab consists of a single component: the Gateway Access Point. Build 2 requires support from the Platform Controller, which also hosts the IoTivity Cloud Service. The IoT devices that are onboarded using Build 2 include three Raspberry Pis.
- Build 3 (i.e., the BRSKI, Sandelman Software Works build) network infrastructure components within the NCCoE lab include a Wi-Fi capable home router (including Join Proxy), a DMZ switch (for management), and an ESP32A Xtensa board acting as a Wi-Fi IoT device, as well as an nRF52840 board acting as an IEEE 802.15.4 device. A management system on a BeagleBone Green serves as a serial console. A registrar server has been deployed as a virtual appliance on the NCCoE private cloud system. Build 3 also requires support from a MASA server which is accessed via the internet. In addition, a Raspberry Pi 3 provides an ethernet/802.15.4 gateway, as well as a test platform.
- Build 4 (i.e., the Thread, Silicon Labs, Kudelski IoT build) network infrastructure components within the NCCoE lab include an Open Thread Border Router, which is implemented using a Raspberry Pi, and a Silicon Labs Gecko Wireless Starter Kit, which acts as an 802.15.4 antenna. Build 4 also requires support from the Kudelski IoT keySTREAM service, which is in the cloud and accessed via the internet. The IoT device that is onboarded in Build 4 is the Silicon Labs Dev Kit (BRD2601A) with an EFR32MG24 System-on-Chip. The application service to which it onboards is AWS IoT.
- Build 5 (i.e., the BRSKI over Wi-Fi, NquiringMinds build) includes 2 Raspberry Pi 4Bs running a Linux operating system. One Raspberry Pi acts as the pledge (or IoT Device) with an Infineon TPM connected. The other acts as the router, registrar and MASA all in one device. This build uses the open source TrustNetZ distribution, from which the entire build can be replicated easily. The TrustNetZ distribution includes source code for the IoT device, the router, the access point, the network onboarding component, the policy engine, the manufacturer services, the registrar and a demo application server. TrustNetZ makes use of NquiringMinds tdx Volt to issue and validate verifiable credentials.
- The BRSKI factory provisioning build is deployed in the Build 5 environment. The IoT device in this build is a Raspberry Pi equipped with an Infineon Optiga SLB 9670 TPM 2.0, which gets provisioned with birth credentials (i.e., a public/private key pair and an IDevID). The BRSKI factory provisioning build also uses an external certificate authority hosted on the premises of NquiringMinds to provide the device certificate signing service.
- The Wi-Fi Easy Connect factory provisioning build is deployed in the Build 1 environment. Its IoT devices are Raspberry Pis equipped with a SEALSQ VaultIC Secure Element, which gets provisioned with a DPP URI. The Secure Element can also be provisioned with an IDevID certificate signed by the SEALSQ INeS certification authority, which is independent of the DPP URI. Code for performing the factory provisioning is stored on an SD card.

1.3 Typographic Conventions

The following table presents typographic conventions used in this volume.

Typeface/Symbol	Meaning	Example
Italics	file names and path names; references to documents that are not hyperlinks; new terms; and placeholders	For language use and style guidance, see the *NCCoE Style Guide*.
Bold	names of menus, options, command buttons, and fields	Choose **File** > **Edit.**
`Monospace`	command-line input, onscreen computer output, sample code examples, and status codes	`mkdir`
`Monospace Bold`	command-line user input contrasted with computer output	**`service sshd start`**
blue text	link to other parts of the document, a web URL, or an email address	All publications from NIST's NCCoE are available at https://www.nccoe.nist.gov.

2 Build 1 (Wi-Fi Easy Connect, Aruba/HPE)

This section of the practice guide contains detailed instructions for installing and configuring all the products used to build an instance of the example solution. For additional details on Build 1's logical and physical architectures, see NIST SP 1800-36B: *Approach, Architecture, and Security Characteristics*.

The network-layer onboarding component of Build 1 utilizes Wi-Fi Easy Connect, also known as the Device Provisioning Protocol (DPP). The Wi-Fi Easy Connect standard is maintained by the Wi-Fi Alliance [1]. The term "DPP" is used when referring to the network-layer onboarding protocol, and "Wi-Fi Easy Connect" is used when referring to the overall implementation of the network onboarding process.

2.1 Aruba Central/Hewlett Packard Enterprise (HPE) Cloud

This build utilized Aruba Central as a cloud management service that provided management and support for the Aruba Wireless Access Point (AP) and provided authorization and DPP onboarding capabilities for the wireless network. A cloud-based application programming interface (API) endpoint provided the ability to import the DPP Uniform Resource Identifiers (URIs) in the manner of a Supply Chain Integration Service. Due to this capability and Build 1's support for Wi-Fi Easy Connect, Build 1's infrastructure fully supported interoperable network-layer onboarding with Build 2's Reference Clients ("IoT devices") provided by CableLabs.

2.2 Aruba Wireless Access Point

Use of DPP is implicitly dependent on the Aruba Central cloud service. Aruba Central provides a cloud Infrastructure as a Service (IaaS) enabled architecture that includes initial support for DPP in Central 2.5.6/ArubaOS (AOS) 10.4.0. Central and AOS support multiple deployment formats:

1. As AP only, referred to as an *underlay deployment,* where traffic is bridged locally from the APs.

2. An *overlay deployment,* where all data is securely tunneled to an on-prem gateway where advanced services can route, inspect, and analyze the data before it's either bridged locally or routed to its next hop.

3. A *mixed-mode deployment,* which is a combination of the two where a returned 'role/label' is used to determine how the data is processed and forwarded.

At the time of this publication, a user can leverage any 3xx, 5xx, or 6xx APs to support a DPP deployment, with a view that all future series APs will implicitly include support. For an existing or new user there is a prerequisite of the creation of a Service Set Identifier (SSID). Note that DPP today is not supported under Wi-Fi Protected Access 3 (WPA3); this is a roadmap item with no published timeline.

Assuming there is an existing SSID or a new one is created based upon the above security restrictions, the next step is to enable DPP (as detailed below in Section 2.2.1) such that the SSID can support multiple authentication and key managements (AKMs) on a Basic Service Set (BSS). If the chosen security type is DPP, only a single AKM will exist for that BSS.

A standards-compliant 802.3at port is the easiest method for providing the AP with power. An external power supply can also be used.

Within this document, we do not cover the specifics of radio frequency (RF) design and placement of APs. Guidance and assistance is available within the Aruba community site, https://community.arubanetworks.com or the Aruba Support Portal, https://asp.arubanetworks.com. Additionally, we do not cover onboarding and licensing of Aruba Central hardware. Documentation can be found here: https://www.arubanetworks.com/techdocs/ArubaDocPortal/content/docportal.htm.

2.2.1 Wi-Fi Network Setup and Configuration

The following instructions detail the initial setup and configuration of the Wi-Fi network upon powering on and connecting the AP to an existing network.

1. Navigate to the Aruba Central cloud management interface.

2. On the sidebar, navigate under **Global** and choose the AP-Group you want to configure/modify. (This assumes you have already grouped your APs by location/functions.)

3. Under **Devices,** click **Config** in the top right side.

4. You will now be in the Access Points tab and WLANs tab. Do one of the following:

    a. If creating a new SSID, click on **+ Add SSID.** After entering the Name (SSID) in Step 1 and configuring options as necessary in Step 2, when you get to Step 3 (Security), it will default on the slide-bar to the Personal Security Level; the alternative is the Enterprise Security Level.

        i. If you choose the **Personal Security Level,** under **Key-Management** ensure you select either **DPP** or **WPA2-Personal.** If you choose **WPA2-Personal,** expand the **Advanced Settings** section and enable the toggle button for DPP so that the SSID can broadcast the AKM. Note that this option is not available if choosing DPP for Key-Management.

    ii. If you choose the **Enterprise Security Level,** only WPA2-Enterprise Key-Management currently supports DPP. Expand the **Advanced Settings** section and enable the toggle button for **DPP** so that the SSID can broadcast the AKM.

  b. If you plan to enable DPP on a previously created SSID:

    i. Ensure you are running version 10.4+ on your devices. You also need an SSID that is configured for WPA2-Personal or WPA2-Enterprise.

    ii. When ready, float your cursor over the previously created SSID name you wish to configure and click on the edit icon.

    iii. Edit the SSID, click on **Security**, and expand the **Advanced Settings** section and enable the toggle button for **DPP.**

    iv. Click **Save Settings.**

For SSIDs that have been modified to add DPP AKM, it's also necessary to enable DPP within the radio profile.

1. Under the **Access Point** Tab, click **Radios.**
2. It's expected you'll see a **default** radio-profile. If a custom one has been created, you'll need to review your configuration before proceeding.
3. Assuming a **default** radio-profile, click on the **Edit** icon, expand **Show advanced settings,** and scroll down to **DPP Provisioning.** You can selectively enable this for 2.4 GHz or 5.0 GHz. Support for DPP on 6.0 GHz is a roadmap item at this time and is not yet available.

2.2.2 Wi-Fi Easy Connect Configuration

Configuration of the Access Point occurred through the Aruba Central cloud management interface. Standard configurations were used to stand up the Build 1 wireless network. The instructions for enabling DPP capabilities for the overall wireless network are listed below:

1. Navigate to the Aruba Central cloud management interface.
2. On the sidebar, navigate to **Security > Authentication and Policy > Config.**
3. In the **Client Access Policy** section, click **Edit.**
4. Under the **Wi-Fi Easy Connect™ Service** heading, ensure that the name of your wireless network is selected.
5. Click **Save.**

2.3 Cisco Catalyst 3850-S Switch

This build utilized a Cisco Catalyst 3850-S switch. This switch utilized a minimal configuration with two separate VLANs to allow for IoT device network segmentation and access control. The switch also provided Power-over-Ethernet support for the Aruba Wireless AP.

2.3.1 Configuration

The switch was configured with two VLANs, and a trunk port dedicated to the Aruba Wireless AP. You can find the relevant portions of the Cisco iOS configuration below:

```
interface Vlan1
 no ip address
interface Vlan2
 no ip address
interface GigabitEthernet1/0/1
 switchport mode trunk
interface GigabitEthernet1/0/2
 switchport mode access
 switchport access vlan 1
interface GigabitEthernet1/0/3
 switchport mode access
 switchport access vlan 2
```

2.4 Aruba User Experience Insight (UXI) Sensor

This build utilized an Aruba UXI Sensor as a Wi-Fi Easy Connect-capable IoT device. Models G6 and G6C support Wi-Fi Easy Connect, and all available G6 and G6C models support Wi-Fi Easy Connect within their software image. This sensor successfully utilized the network-layer onboarding mechanism provided by the wireless network and completed onboarding to the application-layer UXI cloud service. The network-layer onboarding process is automatically initiated by the device on boot.

2.4.1 Configuration

All of Aruba's available G6 and G6C UXI sensors support the ability to complete network-layer and application-layer onboarding. No specific configuration of the physical sensor is required. As part of the supply-chain process, the cryptographic public key for your sensor(s) will be available within the cloud tenant. This public/private keypair for each device is created as part of the manufacturing process. The public key effectively identifiers the sensor to the network and as part of the Wi-Fi Easy Connect/DPP onboarding process. This allows unprovisioned devices straight from the factory to be onboarded and subsequently connect to the UXI sensor cloud to obtain their network-layer configuration. An administrator will have to define the 'tasks' the UXI sensor is going to perform such as monitoring SSIDs, performing reachability tests to on-prem or cloud services, and making the results of these tests available within the UXI user/administrator portal.

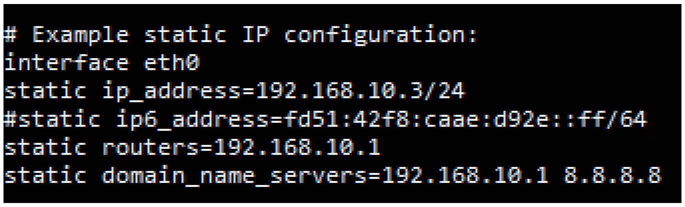
2.5 Raspberry Pi

In this build, the Raspberry Pi 3B+ acts as a DPP enrollee. In setting up the device for this build, a DPP-capable wireless adapter, the Alfa AWUS036NHA network dongle, was connected to enable the Pi to send and receive DPP frames. Once fully configured, the Pi can onboard with the Aruba AP.

2.5.1 Configuration

The following steps were completed for the Raspberry Pi to complete DPP onboarding:

1. Set the management IP for the Raspberry Pi to an IP address in the Build 1 network. To do this, add the following lines to the file *dhcpcd.conf* located at */etc/dhcpcd.conf*. For this build, the IP address was set to 192.168.10.3.



```
# Example static IP configuration:
interface eth0
static ip_address=192.168.10.3/24
#static ip6_address=fd51:42f8:caae:d92e::ff/64
static routers=192.168.10.1
static domain_name_servers=192.168.10.1 8.8.8.8
```

2. Install Linux Libraries using the apt package manager. The following packages were installed:
    a. autotools-dev
    b. automake
    c. libcurl4-openssl-dev
    d. libnl-genl-3-dev
    e. libavahi-client-dev
    f. libavahi-core-dev
    g. aircrack-ng
    h. openssl-1.1.1q
3. Install the DPP utilities. These utilities were installed from the GitHub repository https://github.com/HewlettPackard/dpp using the following command:

```
git clone https://github.com/HewlettPackard/dpp
```

2.5.2 DPP Onboarding

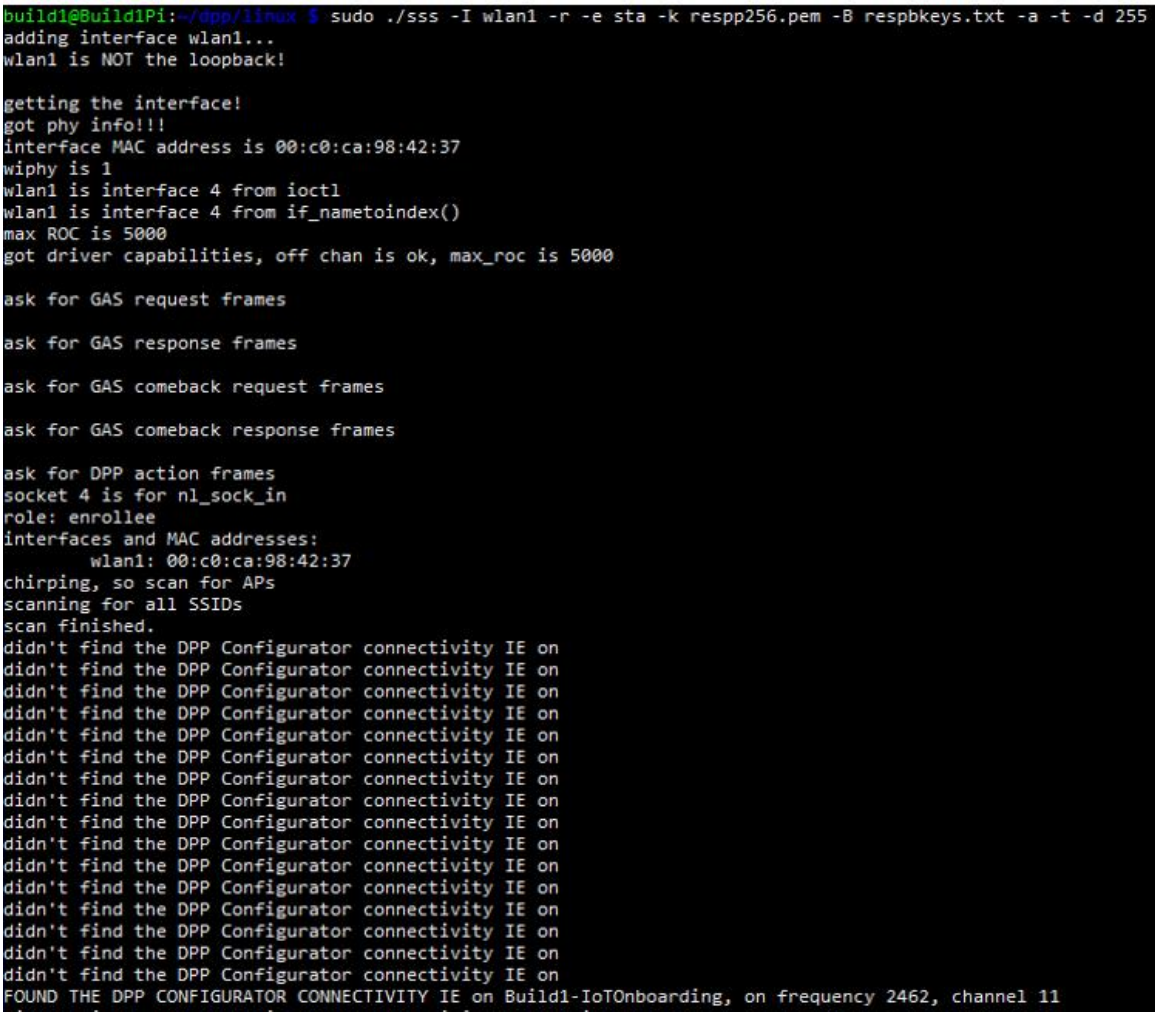
This section describes the steps for using the Raspberry Pi as a DPP enrollee. The Pi uses a DPP utility to send out chirps to make its presence known to available DPP configurators. Once the Pi is discovered, the DPP configurator (Aruba Wireless AP) initiates the DPP authentication protocol. During this phase, DPP *connectors* are created to onboard the device to the network. As soon as the Pi is fully authenticated, it is fully enrolled and can begin normal network communication.

1. Navigate to the DPP utilities directory which was installed during setup:

```
cd dpp/linux
```

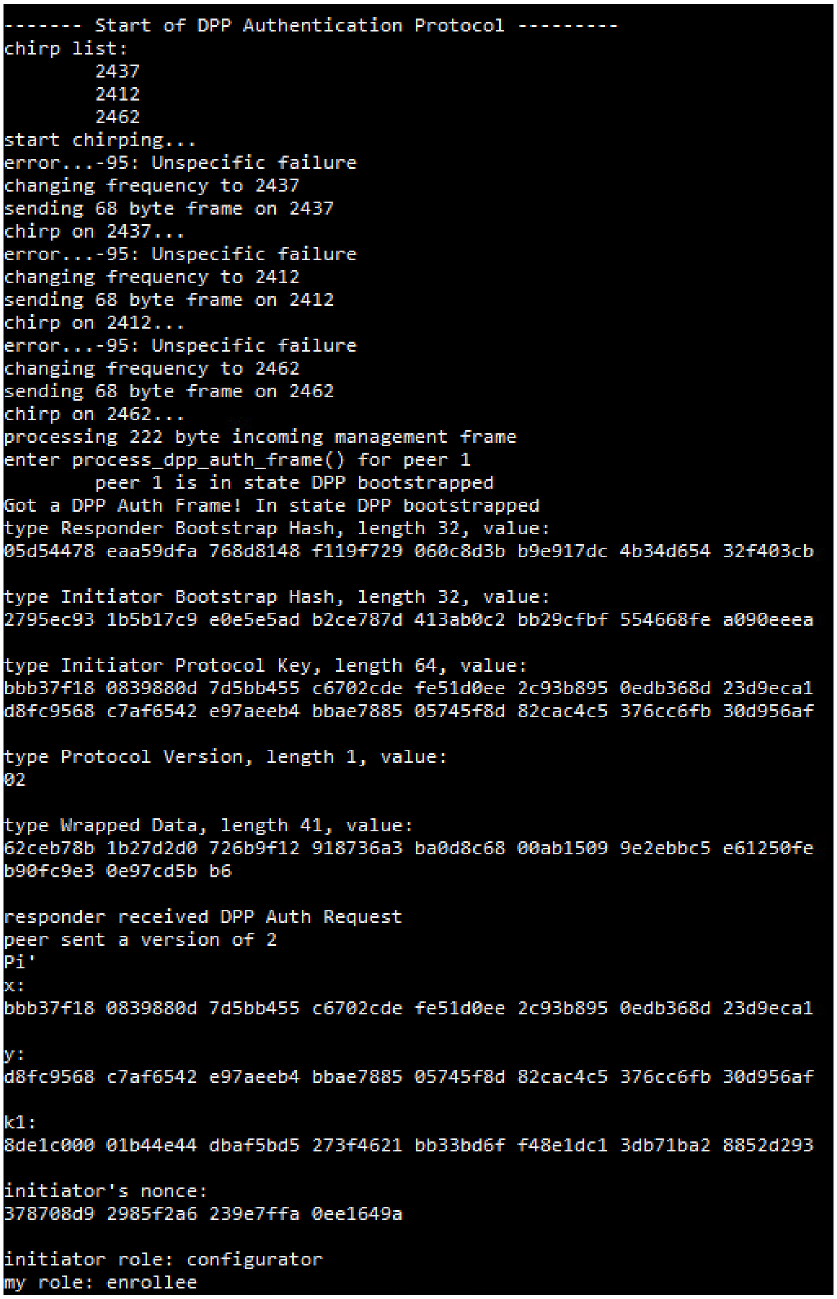
2. From the DPP utilities directory, run the following command to initiate a DPP connection:

```
sudo ./sss -I wlan1 -r -e sta -k respp256.pem -B respbkeys.txt -a -t -d 255
```



```
build1@Build1Pi:~/dpp/linux $ sudo ./sss -I wlan1 -r -e sta -k respp256.pem -B respbkeys.txt -a -t -d 255
adding interface wlan1...
wlan1 is NOT the loopback!

getting the interface!
got phy info!!!
interface MAC address is 00:c0:ca:98:42:37
wiphy is 1
wlan1 is interface 4 from ioctl
wlan1 is interface 4 from if_nametoindex()
max ROC is 5000
got driver capabilities, off chan is ok, max_roc is 5000

ask for GAS request frames

ask for GAS response frames

ask for GAS comeback request frames

ask for GAS comeback response frames

ask for DPP action frames
socket 4 is for nl_sock_in
role: enrollee
interfaces and MAC addresses:
        wlan1: 00:c0:ca:98:42:37
chirping, so scan for APs
scanning for all SSIDs
scan finished.
didn't find the DPP Configurator connectivity IE on
didn't find the DPP Configurator connectivity IE on
didn't find the DPP Configurator connectivity IE on
didn't find the DPP Configurator connectivity IE on
didn't find the DPP Configurator connectivity IE on
didn't find the DPP Configurator connectivity IE on
didn't find the DPP Configurator connectivity IE on
didn't find the DPP Configurator connectivity IE on
didn't find the DPP Configurator connectivity IE on
didn't find the DPP Configurator connectivity IE on
didn't find the DPP Configurator connectivity IE on
didn't find the DPP Configurator connectivity IE on
didn't find the DPP Configurator connectivity IE on
didn't find the DPP Configurator connectivity IE on
didn't find the DPP Configurator connectivity IE on
didn't find the DPP Configurator connectivity IE on
FOUND THE DPP CONFIGURATOR CONNECTIVITY IE on Build1-IoTOnboarding, on frequency 2462, channel 11
```

3. Once the enrollee has found a DPP configurator, the DPP authentication protocol is initiated.



```
------- Start of DPP Authentication Protocol ---------
chirp list:
        2437
        2412
        2462
start chirping...
error...-95: Unspecific failure
changing frequency to 2437
sending 68 byte frame on 2437
chirp on 2437...
error...-95: Unspecific failure
changing frequency to 2412
sending 68 byte frame on 2412
chirp on 2412...
error...-95: Unspecific failure
changing frequency to 2462
sending 68 byte frame on 2462
chirp on 2462...
processing 222 byte incoming management frame
enter process_dpp_auth_frame() for peer 1
        peer 1 is in state DPP bootstrapped
Got a DPP Auth Frame! In state DPP bootstrapped
type Responder Bootstrap Hash, length 32, value:
05d54478 eaa59dfa 768d8148 f119f729 060c8d3b b9e917dc 4b34d654 32f403cb

type Initiator Bootstrap Hash, length 32, value:
2795ec93 1b5b17c9 e0e5e5ad b2ce787d 413ab0c2 bb29cfbf 554668fe a090eeea

type Initiator Protocol Key, length 64, value:
bbb37f18 0839880d 7d5bb455 c6702cde fe51d0ee 2c93b895 0edb368d 23d9eca1
d8fc9568 c7af6542 e97aeeb4 bbae7885 05745f8d 82cac4c5 376cc6fb 30d956af

type Protocol Version, length 1, value:
02

type Wrapped Data, length 41, value:
62ceb78b 1b27d2d0 726b9f12 918736a3 ba0d8c68 00ab1509 9e2ebbc5 e61250fe
b90fc9e3 0e97cd5b b6

responder received DPP Auth Request
peer sent a version of 2
Pi'
x:
bbb37f18 0839880d 7d5bb455 c6702cde fe51d0ee 2c93b895 0edb368d 23d9eca1

y:
d8fc9568 c7af6542 e97aeeb4 bbae7885 05745f8d 82cac4c5 376cc6fb 30d956af

k1:
8de1c000 01b44e44 dbaf5bd5 273f4621 bb33bd6f f48e1dc1 3db71ba2 8852d293

initiator's nonce:
378708d9 2985f2a6 239e7ffa 0ee1649a

initiator role: configurator
my role: enrollee
```

2.6 Certificate Authority

The function of the certificate authority (CA) in this build is to issue network credentials for use in the network-layer onboarding process.

2.6.1 Private Certificate Authority

A private CA was provided as a part of the DPP demonstration utilities in the HPE GitHub repository. For demonstration purposes, the Raspberry Pi is used as the configurator and the enrollee.

2.6.1.1 Installation and Configuration

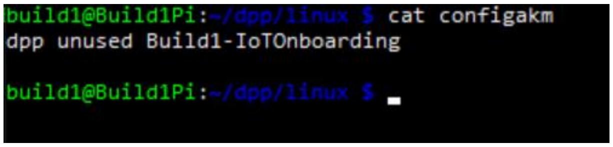
The following instructions detail the initial setup and configuration of the private CA using the DPP demonstration utilities and certificates located at https://github.com/HewlettPackard/dpp.

1. Navigate to the DPP utilities directory on the Raspberry Pi: *~dpp/linux*

    ```
    cd dpp/linux/
    ```

2. The README in the GitHub repository (https://github.com/HewlettPackard/dpp/blob/master/README) references a text file called *configakm* which contains information about the network policies for a configurator to provision on an enrollee. The format is: `<akm> <EAP server> <ssid>`. Current AKMs that are supported are DPP, dot1x, sae, and psk. For this build, DPP is used. For DPP, an Extensible Authentication Protocol (EAP) server is not used.

3. Configure the file *configakm* located in *~/dpp/linux/*. This file instructs the configurator on how to deploy a DPP connector (network credential) from the configurator to the enrollee. As shown below, the *configakm* file is filled with the following fields:
    `dpp unused Build1-IoTOnboarding.`

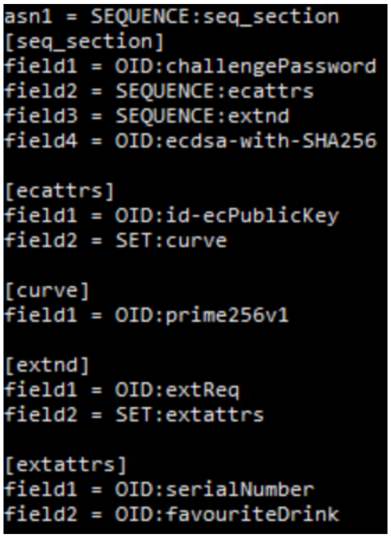


```
build1@Build1Pi:~/dpp/linux $ cat configakm
dpp unused Build1-IoTOnboarding

build1@Build1Pi:~/dpp/linux $
```

4. The file *csrattrs.conf* contains attributes to construct an Abstract Syntax Notation One (ASN.1) string. This string allows the configurator to tell the enrollee how to generate a certificate signing request (CSR). The following fields were used for this demonstration:

```
asn1 = SEQUENCE: seq_section
[seq_section]
field1 = OID:challengePassword
field2 = SEQUENCE:ecattrs
field3 = SEQUENCE:extnd
field4 = OID:ecdsa-with-SHA256

[ecattrs]
field1 = OID:id-ecPublicKey
field2 = SET:curve

[curve]
field1 = OID:prime256v1
```

```
[extnd]
field1 = OID:extReq
field2 = SET:extattrs

[extattrs]
field1 = OID:serialNumber
field2 = OID:favouriteDrink
```



```
asn1 = SEQUENCE:seq_section
[seq_section]
field1 = OID:challengePassword
field2 = SEQUENCE:ecattrs
field3 = SEQUENCE:extnd
field4 = OID:ecdsa-with-SHA256

[ecattrs]
field1 = OID:id-ecPublicKey
field2 = SET:curve

[curve]
field1 = OID:prime256v1

[extnd]
field1 = OID:extReq
field2 = SET:extattrs

[extattrs]
field1 = OID:serialNumber
field2 = OID:favouriteDrink
```

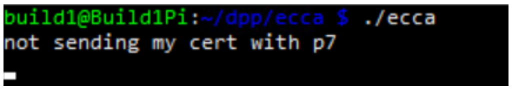
2.6.1.2 Operation and Demonstration

Once setup and configuration have been completed, the following steps can be used to demonstrate utilizing the private CA to issue credentials to a requesting device.

1. Open three terminals on the Raspberry Pi: one to start the certificate program, one to show the configurator's point of view, and one to show the enrollee's point of view.

2. The demonstration uses an OpenSSL certificate. To run the program from the first terminal, navigate to the following directory: *~/dpp/ecca/,* and run the command:

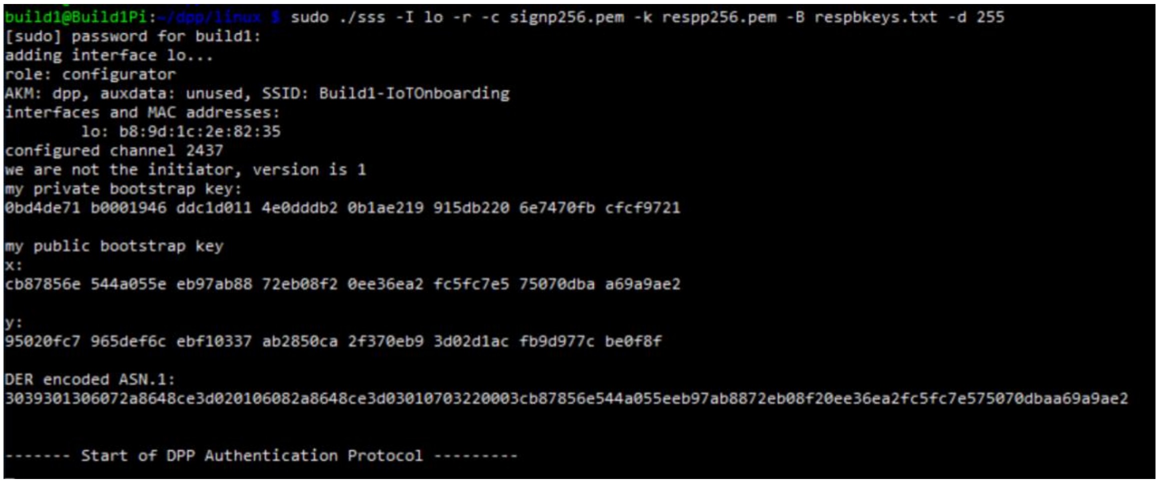
   ```
   ./ecca.
   ```



```
build1@Build1Pi:~/dpp/ecca $ ./ecca
not sending my cert with p7
```

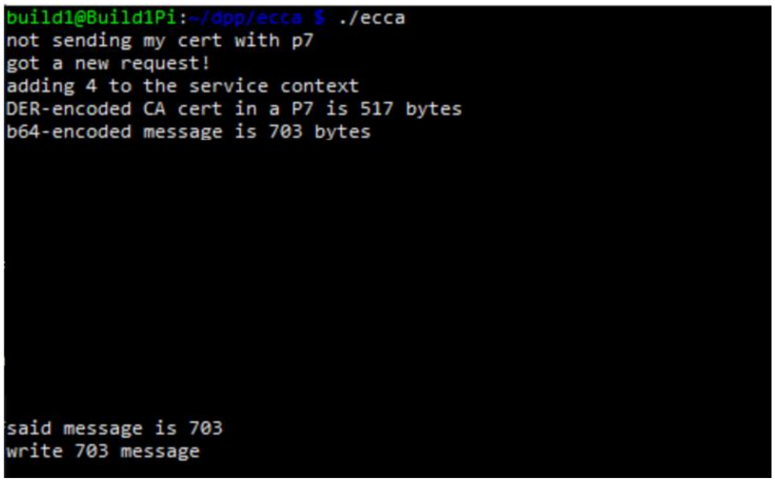
3. On the second terminal, start the configurator using the following command:

   ```
   sudo ./sss -I lo -r -c signp256.pem -k respp256.pem -B resppbkeys.txt -d 255
   ```



```
build1@Build1Pi:~/dpp/linux $ sudo ./sss -I lo -r -c signp256.pem -k respp256.pem -B respbkeys.txt -d 255
[sudo] password for build1:
adding interface lo...
role: configurator
AKM: dpp, auxdata: unused, SSID: Build1-IoTOnboarding
interfaces and MAC addresses:
        lo: b8:9d:1c:2e:82:35
configured channel 2437
we are not the initiator, version is 1
my private bootstrap key:
0bd4de71 b0001946 ddc1d011 4e0dddb2 0b1ae219 915db220 6e7470fb cfcf9721

my public bootstrap key
x:
cb87856e 544a055e eb97ab88 72eb08f2 0ee36ea2 fc5fc7e5 75070dba a69a9ae2

y:
95020fc7 965def6c ebf10337 ab2850ca 2f370eb9 3d02d1ac fb9d977c be0f8f

DER encoded ASN.1:
3039301306072a8648ce3d020106082a8648ce3d03010703220003cb87856e544a055eeb97ab8872eb08f20ee36ea2fc5fc7e575070dbaa69a9ae2


------- Start of DPP Authentication Protocol ---------
```

As shown in the terminal where the ecca program is running, the configurator contacts the CA and asks for the certificate.

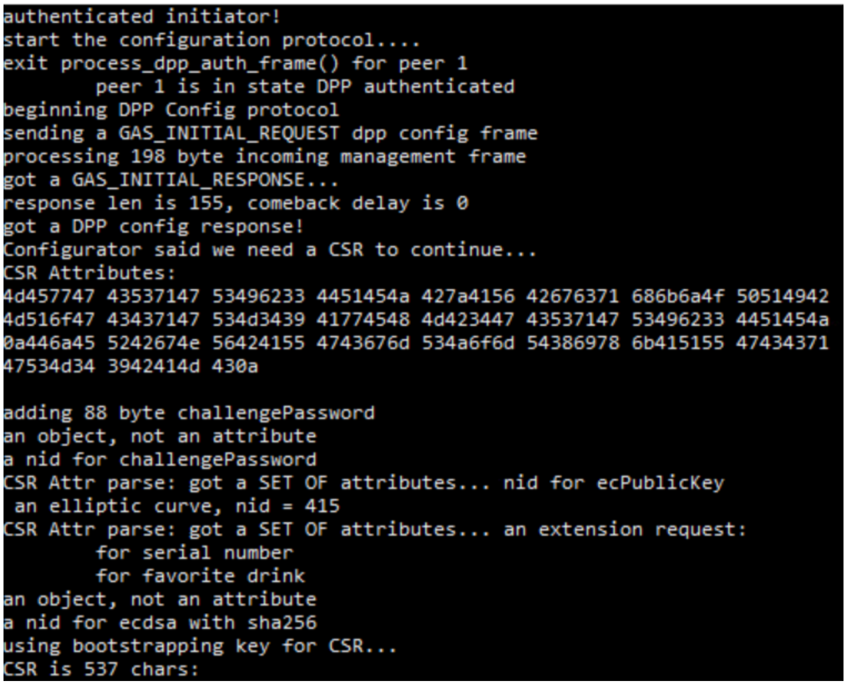


```
build1@Build1Pi:~/dpp/ecca $ ./ecca
not sending my cert with p7
got a new request!
adding 4 to the service context
DER-encoded CA cert in a P7 is 517 bytes
b64-encoded message is 703 bytes











said message is 703
write 703 message
```

4. On the third terminal, start the enrollee using the following command:

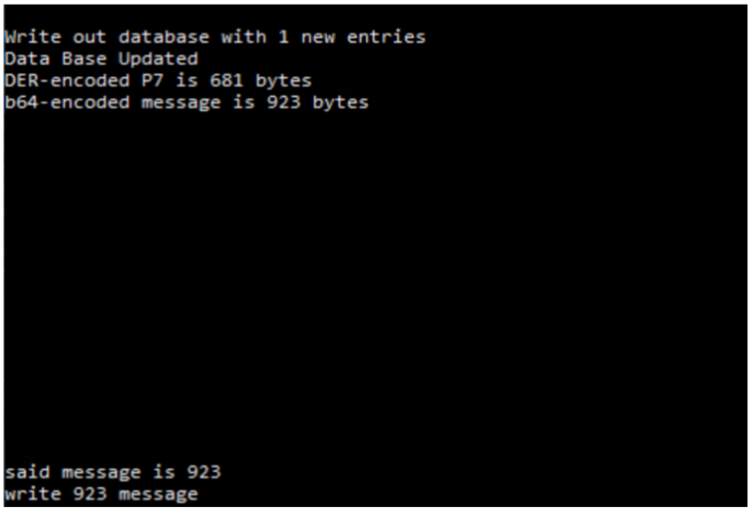
   `sudo ./sss -I lo -r -e sta -k initp256.pem -B initbkeys.txt -t -a -q -d 255`

   From the enrollee’s perspective, it will send chirps on different channels until it finds the configurator. Once found, it sends its certificate to the CA for signing. The snippet below is of the enrollee generating the CSR.



```
authenticated initiator!
start the configuration protocol....
exit process_dpp_auth_frame() for peer 1
        peer 1 is in state DPP authenticated
beginning DPP Config protocol
sending a GAS_INITIAL_REQUEST dpp config frame
processing 198 byte incoming management frame
got a GAS_INITIAL_RESPONSE...
response len is 155, comeback delay is 0
got a DPP config response!
Configurator said we need a CSR to continue...
CSR Attributes:
4d457747 43537147 53496233 4451454a 427a4156 42676371 686b6a4f 50514942
4d516f47 43437147 534d3439 41774548 4d423447 43537147 53496233 4451454a
0a446a45 5242674e 56424155 4743676d 534a6f6d 54386978 6b415155 47434371
47534d34 3942414d 430a

adding 88 byte challengePassword
an object, not an attribute
a nid for challengePassword
CSR Attr parse: got a SET OF attributes... nid for ecPublicKey
 an elliptic curve, nid = 415
CSR Attr parse: got a SET OF attributes... an extension request:
        for serial number
        for favorite drink
an object, not an attribute
a nid for ecdsa with sha256
using bootstrapping key for CSR...
CSR is 537 chars:
```

5. In the ecca terminal, the certificate from the enrollee is shown



```
Write out database with 1 new entries
Data Base Updated
DER-encoded P7 is 681 bytes
b64-encoded message is 923 bytes









said message is 923
write 923 message
```

2.6.2 SEALSQ INeS

The SEALSQ INeS Certificate Management System provides CA and certificate management capabilities for Build 1. Implementation of this system provides Build 1 with a trusted, public CA to support issuing network credentials.

2.6.2.1 Setup and Configuration

To support this build, a custom software agent was deployed on a Raspberry Pi in the Build 1 network. This agent interacted with the cloud-based CA in SEALSQ INeS via API to sign network credentials. Network-level onboarding of IoT devices was completed via DPP, with network credentials being successfully requested from and issued by SEALSQ INeS.

Additional information on interacting with the SEALSQ INeS API can be found at https://inesdev.certifyiddemo.com/. Access can be requested directly from SEALSQ via their contact form: https://www.sealsq.com/contact.

2.7 UXI Cloud

The UXI Cloud is a web-based application that serves as a monitoring hub for the UXI sensor. It provides visibility into the data captured by the performance monitoring that the UXI sensor conducts. For the purposes of this build, the dashboard was used to demonstrate application-layer onboarding, which occurs once the UXI sensor has completed network-layer onboarding. Once application-layer onboarding was completed and the application configuration had been applied to the device, our demonstration concluded.

2.8 Wi-Fi Easy Connect Factory Provisioning Build

This Factory Provisioning Build included many of the components listed above, including Aruba Central, SEALSQ INeS, the Aruba Access Point, and Raspberry Pi IoT devices. A SEALSQ VaultIC Secure Element was also included in the build and provided secure generation and storage of the key material and certificates provisioned to the device.

2.8.1 SEALSQ VaultIC Secure Element

The SEALSQ VaultIC Secure Element was connected to a Raspberry Pi via the built-in GPIO pins present on the Pi. SEALSQ provided demonstration code that generates a public/private keypair within the secure element, creates a Certificate Signing Request, and uses that CSR to obtain an IDevID certificate from SEALSQ INeS. This code supports the Raspberry Pi OS Bullseye. The demonstration code can be found at the official GitHub repository.

HPE also provided a custom DPP-based implementation of the SEALSQ code, which generates supporting material within the secure element, and then generates a DPP URI. This DPP URI is available in a string format, PNG (QR Code), and ASCII (QR Code). The DPP URI can then be used for network onboarding, as described in the rest of the Build 1 section. This code is included in the demonstration code located at the repository linked above.

2.8.1.1 Installation and Configuration

Full instructions for installation and configuration can be found in the INSTALL.txt file from the SEALSQ demonstration code mentioned above. A general set of steps for preparing to run the demonstration code is included below.

1. Install prerequisites on Raspberry Pi
   a. cmake
   b. git
   c. gcc
2. On the Raspberry Pi, run the `sudo raspi-update` command to update drivers

3. Before plugging VaultIC Secure Element into the Raspberry Pi connector, configure the jumpers:
    a. Set _VCC_ jumper
        i. CTRL = VaultIC power controlled by GPIO25 (default)
        ii. 3V3 = VaultIC power always on
    b. Set J1&J2 to select I2C or SPI
        i. If using SPI, set J1 to SS and J2 to SEL (default)
        ii. If using I2C, set J1 to SCL and J2 to SDA
4. Using the `raspi-config` command, enable the SPI or I2C interface on the Raspberry Pi
5. Run `git clone https://github.com/sclark-wisekey/NCCoE.factory.pub` to pull down the demonstration code.

2.8.1.2 Running the demonstration code

1. Navigate to the folder containing the demonstration code. Inside that folder, navigate to the VaultIC/demos folder.
2. Edit the file config.cfg and change the value of VAULTIC_COMM to match with the jumpers configured during setup.
3. The demonstrations are available with wolfSSL stacks and organized in dedicated folders. The README.TXT file in each demonstration subfolder explains how to run the demonstrations.

3 Build 2 (Wi-Fi Easy Connect, CableLabs, OCF)

This section of the practice guide contains detailed instructions for installing and configuring all of the products used to build an instance of the example solution. For additional details on Build 2's logical and physical architectures, see NIST SP 1800-36B: *Approach, Architecture, and Security Characteristics*.

The network-layer onboarding component of Build 2 utilizes Wi-Fi Easy Connect, also known as the Device Provisioning Protocol (DPP). The Wi-Fi Easy Connect standard is maintained by the Wi-Fi Alliance [1]. The term "DPP" is used when referring to the network-layer onboarding protocol, and "Wi-Fi Easy Connect" is used when referring to the overall implementation of the network onboarding process.

3.1 CableLabs Platform Controller

The CableLabs Platform Controller provides an architecture and reference implementation of a cloud-based service that provides management capability for service deployment groups, access points with the deployment groups, registration and lifecycle of user services, and the secure onboarding and lifecycle management of users' Wi-Fi devices. The controller also exposes APIs for integration with third-party systems for the purpose of integrating various business flows (e.g., integration with manufacturing process for device management).

The Platform Controller would typically be hosted by the network operator or a third-party service provider. It can be accessed via web interface. Additional information for this deployment can be accessed at the official CableLabs repository.

3.1.1 Operation and Demonstration

Once configuration of the Platform Controller, Gateway, and Reference Client has been completed, full operation can commence. Instructions for this are located at the official CableLabs repository.

3.2 CableLabs Custom Connectivity Gateway

In this deployment, the gateway software is running on a Raspberry Pi 3B+, which acts as a router, firewall, wireless access point, Open Connectivity Foundation (OCF) Diplomat, and OCF Onboarding Tool. The gateway is also connected to the CableLabs Platform Controller, which manages much of the configuration and functions of the gateway. Due to Build 2’s infrastructure and support of Wi-Fi Easy Connect, Build 2 fully supported interoperable network-layer onboarding with Build 1’s IoT devices.

3.2.1 Installation and Configuration

Hardware requirements, pre-installation steps, installation steps, and configuration instructions for the gateway can be found at the official CableLabs repository.

3.2.2 Integration with CableLabs Platform Controller

Once initial configuration has occurred, the gateway can be integrated with the CableLabs Platform Controller. Instructions can be found at the official CableLabs repository.

3.2.3 Operation and Demonstration

Once configuration of the Platform Controller, Gateway, and Reference Client has been completed, full operation can commence. Instructions for this are located at the official CableLabs repository.

3.3 Reference Clients/IoT Devices

Three reference clients were deployed in this build, each on a Raspberry Pi 3B+. They were each configured to emulate either a smart light switch or a smart lamp. The software deployed also included the capability to perform network-layer onboarding via Wi-Fi Easy Connect (or DPP) and application-layer onboarding using the OCF onboarding method. These reference clients were fully interoperable with network-layer onboarding to Build 1.

3.3.1 Installation and Configuration

Hardware requirements, pre-installation, installation, and configuration steps for the reference clients are detailed in the official CableLabs repository.

3.3.2 Operation and Demonstration

Once configuration of the Platform Controller, Gateway, and Reference Client has been completed, full operation can commence. Instructions for this are located at the official CableLabs repository.

For interoperability with Build 1, the IoT device’s DPP URI was provided to Aruba Central, which allowed Build 1 to successfully complete network-layer onboarding with the Build 2 IoT devices.

4 Build 3 (BRSKI, Sandelman Software Works)

This section of the practice guide contains detailed instructions for installing and configuring all of the products used to build an instance of the example solution. For additional details on Build 3’s logical and physical architectures, see NIST SP 1800-36B: *Approach, Architecture, and Security Characteristics*.

The network-layer onboarding component of Build 3 utilizes the Bootstrapping Remote Secure Infrastructure (BRSKI) protocol. Build 3 is representative of a typical home or small office network.

4.1 Onboarding Router/Join Proxy

The onboarding router quarantines the IoT device attempting to join the network until the BRSKI onboarding process is complete. The router in this build is a Turris MOX device, which is based on the Linux OpenWrt version 4 operating system (OS). The Raspberry Pi 3 contains software to function as the Join Proxy for pledges to the network. If another brand of device is used, a different source of compiled Join Proxy might be required.

4.1.1 Setup and Configuration

The router needs to be IPv6 enabled. In the current implementation, the join package operates on an unencrypted network.

4.2 Minerva Join Registrar Coordinator

The purpose of the Join Registrar is to determine whether a new device is allowed to join the network. The Join Registrar is located on a virtual machine running Devuan Linux 4 within the network.

4.2.1 Setup and Configuration

The Minerva Fountain Join Registrar/Coordinator is available as a Docker container and as a VM in OVA format at the Minerva fountain page. Further setup and configuration instructions are available on the Sandelman website on the configuration page.

For the Build 3 demonstration, the VM deployment was installed onto a VMware vSphere system.

A freshly booted VM image will do the following on its own:

- Configure a database
- Configure a local certificate authority (fountain:s0_setup_jrc)
- Configure certificates for the database connection
- Configure certificates for the Registrar https interface
- Configure certificates for use with the Bucardo database replication system
- Configure certificates for LDevID certification authority (fountain:s2_create_registrar)

- Start the JRC

The root user is permitted to log in on the console ("tty0") using the password “root” but is immediately forced to set a new password.

The new registrar will announce itself with the name minerva-fountain.local in mDNS.

The logs for this are put into */var/log/configure-fountain-12345.log* (where 12345 is a new number based upon the PID of the script).

4.3 Reach Pledge Simulator

The Reach Pledge Simulator acts as an IoT device in Build 3. The pledge is acting as an IoT device joining the network and is hosted on a Raspberry Pi 3. More information is available on the Sandelman website on the Reach page.

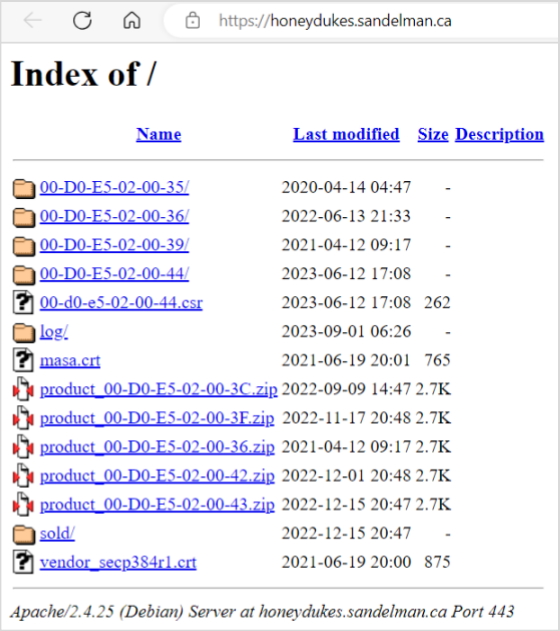
4.3.1 Setup and Configuration

While the functionality of this device is to act as an IoT device, it runs on the same software as the Join Registrar Coordinator. This software is available in both VM and Docker container format. Please see Section 4.2.1 for installation instructions.

When setting up the Reach Pledge Simulator, the address of the Join Registrar Coordinator is automatically determined by the pledge.

Currently, the Reach Pledge Simulator obtains its IDevID using the following steps:

1. View the available packages by visiting the Sandelman website.



https://honeydukes.sandelman.ca

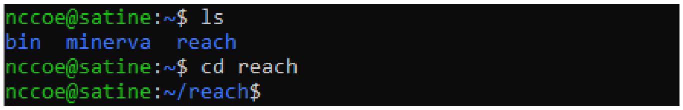
Index of /

Name	Last modified	Size	Description
00-D0-E5-02-00-35/	2020-04-14 04:47	-	
00-D0-E5-02-00-36/	2022-06-13 21:33	-	
00-D0-E5-02-00-39/	2021-04-12 09:17	-	
00-D0-E5-02-00-44/	2023-06-12 17:08	-	
00-d0-e5-02-00-44.csr	2023-06-12 17:08	262	
log/	2023-09-01 06:26	-	
masa.crt	2021-06-19 20:01	765	
product_00-D0-E5-02-00-3C.zip	2022-09-09 14:47	2.7K	
product_00-D0-E5-02-00-3F.zip	2022-11-17 20:48	2.7K	
product_00-D0-E5-02-00-36.zip	2021-04-12 09:17	2.7K	
product_00-D0-E5-02-00-42.zip	2022-12-01 20:48	2.7K	
product_00-D0-E5-02-00-43.zip	2022-12-15 20:47	2.7K	
sold/	2022-12-15 20:47	-	
vendor_secp384r1.crt	2021-06-19 20:00	875	

Apache/2.4.25 (Debian) Server at honeydukes.sandelman.ca Port 443

2. Open a terminal on the Raspberry Pi device and navigate to the Reach directory by entering:

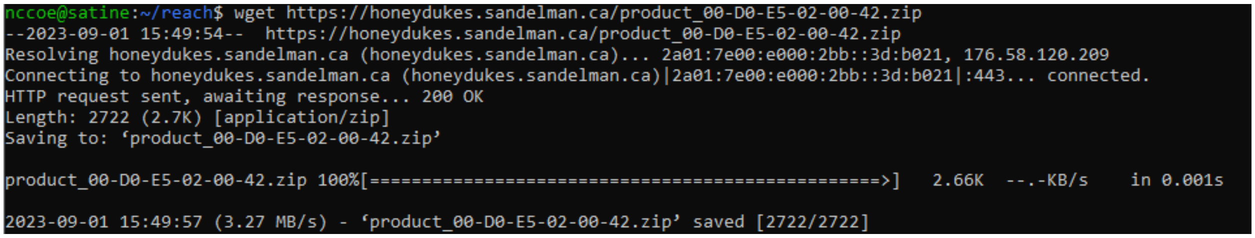
```
cd reach
```



```
nccoe@satine:~$ ls
bin  minerva  reach
nccoe@satine:~$ cd reach
nccoe@satine:~/reach$
```

3. Enter the following command while substituting the URL for one of the available zip files containing the IDevID of choice on the Sandelman website.

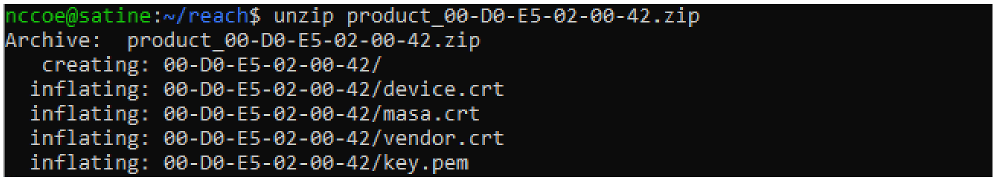
```
wget https://honeydukes.sandelman.ca/product_00-D0-E5-02-00-42.zip
```



```
nccoe@satine:~/reach$ wget https://honeydukes.sandelman.ca/product_00-D0-E5-02-00-42.zip
--2023-09-01 15:49:54--  https://honeydukes.sandelman.ca/product_00-D0-E5-02-00-42.zip
Resolving honeydukes.sandelman.ca (honeydukes.sandelman.ca)... 2a01:7e00:e000:2bb::3d:b021, 176.58.120.209
Connecting to honeydukes.sandelman.ca (honeydukes.sandelman.ca)|2a01:7e00:e000:2bb::3d:b021|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 2722 (2.7K) [application/zip]
Saving to: ‘product_00-D0-E5-02-00-42.zip’

product_00-D0-E5-02-00-42.zip 100%[===============================================>]   2.66K  --.-KB/s    in 0.001s

2023-09-01 15:49:57 (3.27 MB/s) - ‘product_00-D0-E5-02-00-42.zip’ saved [2722/2722]
```

4. Unzip the file by entering the following command, substituting the name of your zip file (the IDevID is the *device.crt* file):

```
unzip product_00-D0-E5-02-00-42.zip
```



```
nccoe@satine:~/reach$ unzip product_00-D0-E5-02-00-42.zip
Archive:  product_00-D0-E5-02-00-42.zip
   creating: 00-D0-E5-02-00-42/
  inflating: 00-D0-E5-02-00-42/device.crt
  inflating: 00-D0-E5-02-00-42/masa.crt
  inflating: 00-D0-E5-02-00-42/vendor.crt
  inflating: 00-D0-E5-02-00-42/key.pem
```

Typically, this would be accomplished through a provisioning process involving a Certificate Authority, as demonstrated in the Factory Provisioning builds.

4.4 Serial Console Server

The serial console server does not participate in the onboarding process but provides direct console access to the IoT devices. The serial console server has been attached to a multi-port USB hub and USB connectors and/or USB2TTL adapters connected to each device. The ESP32 and the nRF52840 are both connected to the serial console and receive power from the USB hub. Power to the console and IoT devices is also provided via the USB hub. A BeagleBone Green device was used as the serial console, using the "screen" program as the telecom device.

4.5 Minerva Highway MASA Server

In the current implementation of the build, the MASA server provides the Reach Pledge Simulator with an IDevID Certificate and a public/private keypair for demonstration purposes. Typically, this would be accomplished through a factory provisioning process involving a Certificate Authority, as demonstrated in the Factory Provisioning builds.

4.5.1 Setup and Configuration

Installation of the Minerva Highway MASA is described at the Highway configuration page. Additional configuration details are available at the Highway development page.

Availability of VMs and containers is described at the following Minerva page.

5 Build 4 (Thread, Silicon Labs, Kudelski IoT)

This section of the practice guide contains detailed instructions for installing and configuring all of the products used to build an instance of the example solution. For additional details on Build 4's logical and physical architectures, see NIST SP 1800-36B: *Approach, Architecture, and Security Characteristics*.

This build utilizes the Thread protocol and performs application-layer onboarding using the Kudelski keySTREAM service to provision a device to the AWS IoT Core.

5.1 Open Thread Border Router

The Open Thread Border Router forms the Thread network and acts as the router on this build. The Open Thread Border Router is run as software on a Raspberry Pi 3B. The Silicon Labs Gecko Wireless Devkit is attached to the Raspberry Pi via USB and acts as the 802.15.4 antenna for this build.

5.1.1 Installation and Configuration

On the Raspberry Pi, run the following commands from a terminal to install and configure the Open Thread Border Router software:

```
git clone https://github.com/openthread/ot-br-posix
sudo NAT64=1 DNS64=1 WEB_GUI=1 ./script/bootstrap
sudo NAT64=1 DNS64=1 WEB_GUI=1 ./script/setup
```

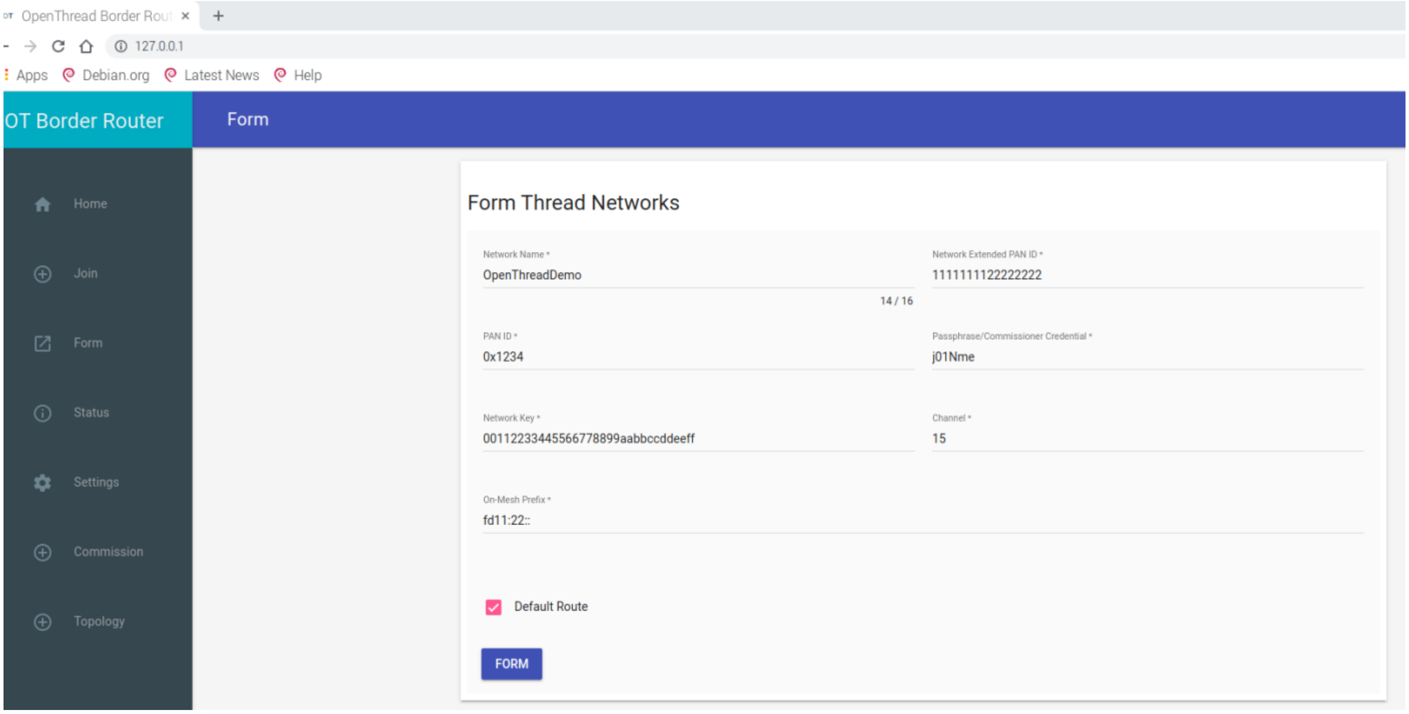
5.1.2 Operation and Demonstration

Once initial configuration has occurred, the OpenThread Border Router should be functional and operated through the web GUI.

1. To open the OpenThread Border Router GUI enter the following IP in a web browser:

   ```
   127.0.0.1
   ```

2. In the **Form** tab, enter the details for the Thread network being formed. For demonstration purposes we only updated the credentials field.



Form Thread Networks

Network Name *
OpenThreadDemo
14 / 16

Network Extended PAN ID *
1111111122222222

PAN ID *
0x1234

Passphrase/Commissioner Credential *
j01Nme

Network Key *
00112233445566778899aabbccddeeff

Channel *
15

On-Mesh Prefix *
fd11:22::

☑ Default Route

FORM

5.2 Silicon Labs Dev Kit (BRD2601A)

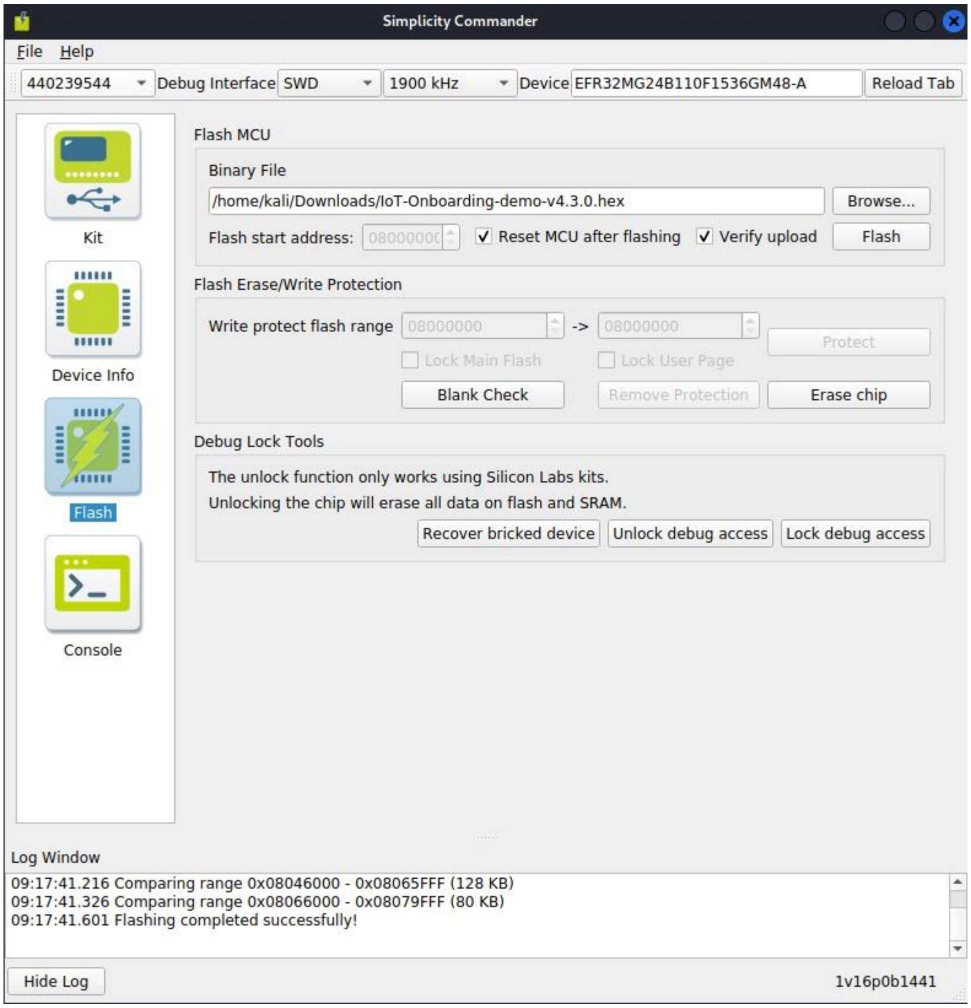
The Silicon Labs Dev Kit acts as the IoT device for this build. It is controlled using the Simplicity Studio v5 Software available at the official Simplicity Studio page and connected to a computer running Windows or Linux via USB. Our implementation leveraged a Linux machine running Simplicity Studio. Custom firmware for the Dev Kit leveraged in this use case was made by Silicon Labs.

5.2.1 Setup and Configuration

The Dev Kit custom firmware image works in conjunction with the Kudelski keySTREAM service. More information is available by contacting Silicon Labs through their contact form. Once the custom firmware has been acquired the Dev Kit can be configured using the following steps.

1. Connect the Dev Kit via USB to the machine running Simplicity Studio.

2. The firmware is installed onto the Dev Kit using the Simplicity Commander tool within Simplicity Studio.



440239544 Debug Interface SWD 1900 kHz Device EFR32MG24B110F1536GM48-A

Binary File: /home/kali/Downloads/IoT-Onboarding-demo-v4.3.0.hex

Flash start address: 0800000(☑ Reset MCU after flashing ☑ Verify upload

Write protect flash range 08000000 -> 08000000

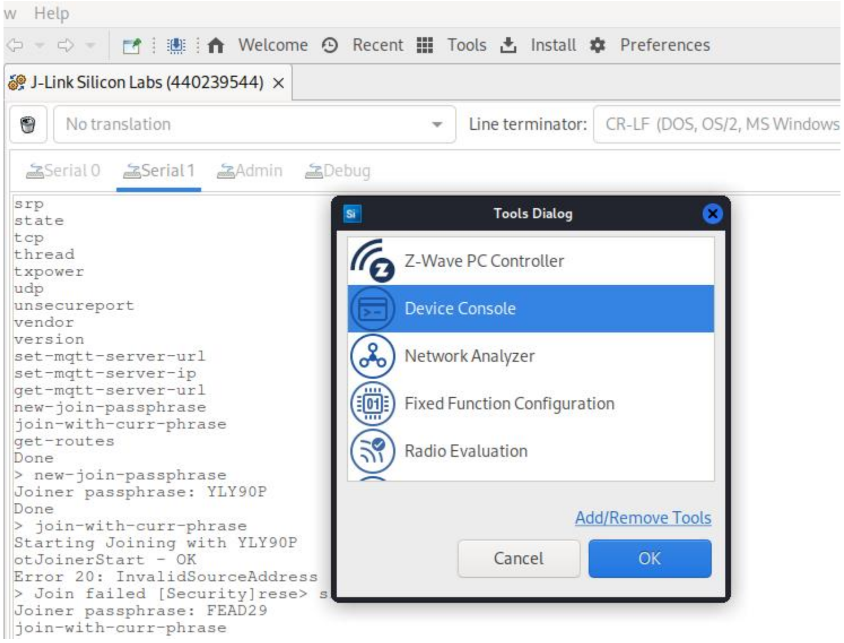
The unlock function only works using Silicon Labs kits.
Unlocking the chip will erase all data on flash and SRAM.

```
09:17:41.216 Comparing range 0x08046000 - 0x08065FFF (128 KB)
09:17:41.326 Comparing range 0x08066000 - 0x08079FFF (80 KB)
09:17:41.601 Flashing completed successfully!
```

1v16p0b1441

After selecting the firmware file, click **Flash** to flash the firmware the Dev Kit.

3. Open the device console in the **Tools** tab and then select the **Serial 1** tab.

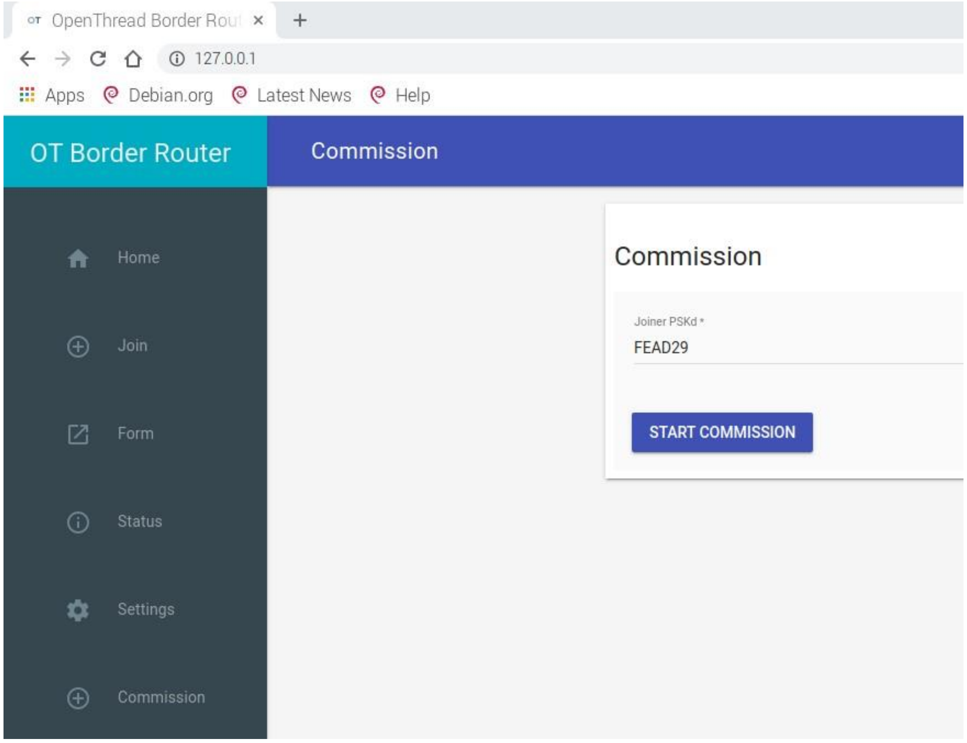


```
srp
state
tcp
thread
txpower
udp
unsecureport
vendor
version
set-mqtt-server-url
set-mqtt-server-ip
get-mqtt-server-url
new-join-passphrase
join-with-curr-phrase
get-routes
Done
> new-join-passphrase
Joiner passphrase: YLY90P
Done
> join-with-curr-phrase
Starting Joining with YLY90P
otJoinerStart - OK
Error 20: InvalidSourceAddress
> Join failed [Security]rese> s
Joiner passphrase: FEAD29
join-with-curr-phrase
```

4. Enter the following command to create a new join passphrase in the Serial 1 command line:

```
new-join-passphrase
```

5. Enter the output of the previous command in the **Commission** tab in the OpenThread Border Router GUI and click **Start Commission.**

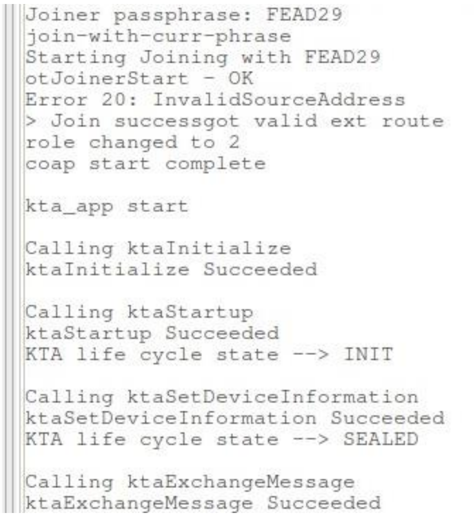


Joiner PSKd *
FEAD29

6. In the Simplicity Commander Device Console, enter the following command to begin the joining process from the Thunderboard:

```
join-with-curr-phrase
```

7. Press the **Reset** button on the Dev Kit and the device will join the thread network and reach out to the Kudelski keySTREAM service. You should see the following output in the Simplicity Commander Device Console:



```
Joiner passphrase: FEAD29
join-with-curr-phrase
Starting Joining with FEAD29
otJoinerStart - OK
Error 20: InvalidSourceAddress
> Join successgot valid ext route
role changed to 2
coap start complete

kta_app start

Calling ktaInitialize
ktaInitialize Succeeded

Calling ktaStartup
ktaStartup Succeeded
KTA life cycle state --> INIT

Calling ktaSetDeviceInformation
ktaSetDeviceInformation Succeeded
KTA life cycle state --> SEALED

Calling ktaExchangeMessage
ktaExchangeMessage Succeeded
```

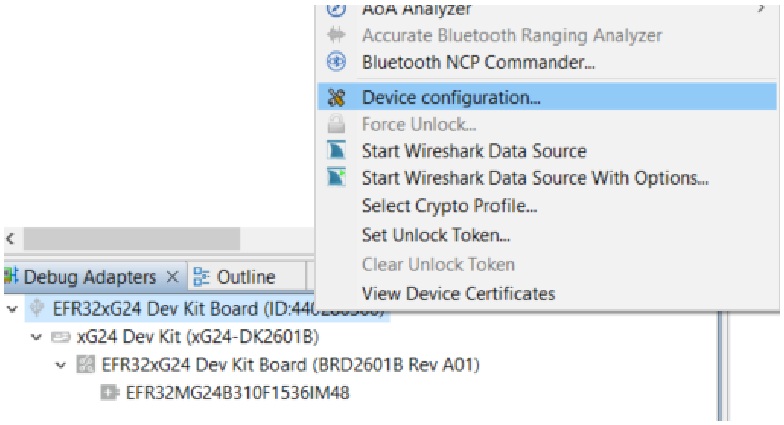
5.3 Kudelski keySTREAM Service

In this section we describe the Kudelski keySTREAM service which this build utilizes to provision certificates for connecting to the AWS IoT core. More information on keySTREAM is available at the keySTREAM page.

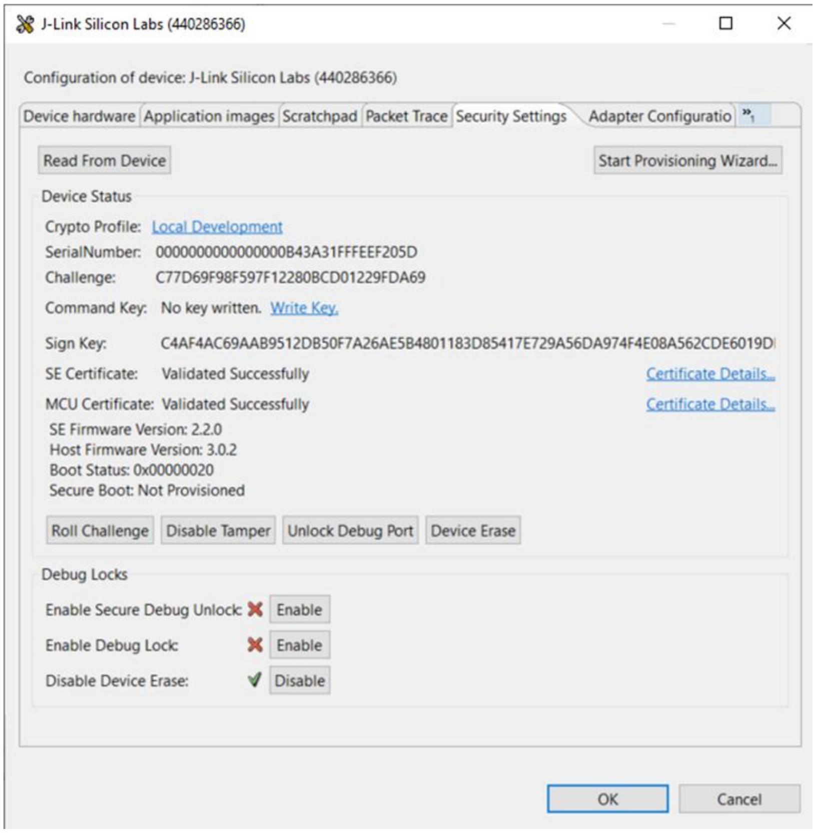
5.3.1 Setup and Configuration

The Kudelski keySTREAM service provides two certificates for the device: a CA certificate and a Proof of Possession (POP) certificate that is generated using a code from the AWS server. This section describes the steps to download these certificates.

1. Locate the Chip UID for the Silicon Labs Dev Kit in Simplicity Studio by right clicking on the **Device Adapters** tab at the bottom and selecting **Device Configuration**.


AOA Analyzer
Accurate Bluetooth Ranging Analyzer
Bluetooth NCP Commander...
Device configuration...
Force Unlock...
Start Wireshark Data Source
Start Wireshark Data Source With Options...
Select Crypto Profile...
Set Unlock Token...
Clear Unlock Token
View Device Certificates
Debug Adapters
Outline
EFR32xG24 Dev Kit Board (ID:440
xG24 Dev Kit (xG24-DK2601B)
EFR32xG24 Dev Kit Board (BRD2601B Rev A01)
EFR32MG24B310F1536IM48

2. On the **Security Settings** tab, take the last 16 characters of the serial number and remove the ‘FFFE’ characters from the 7th – 11th positions.



Configuration of device: J-Link Silicon Labs (440286366)

Device Status

Crypto Profile: Local Development

SerialNumber: 0000000000000000B43A31FFFEEF205D

Challenge: C77D69F98F597F12280BCD01229FDA69

Command Key: No key written. Write Key.

Sign Key: C4AF4AC69AAB9512DB50F7A26AE5B4801183D85417E729A56DA974F4E08A562CDE6019D

SE Certificate: Validated Successfully

MCU Certificate: Validated Successfully

SE Firmware Version: 2.2.0
Host Firmware Version: 3.0.2
Boot Status: 0x00000020
Secure Boot: Not Provisioned

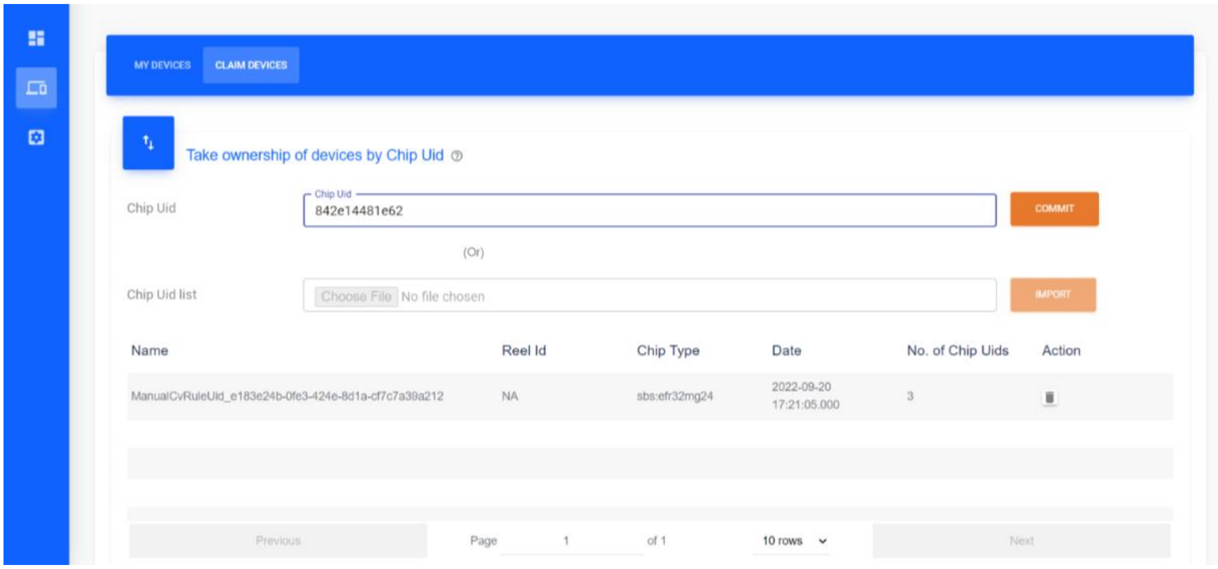
Debug Locks

Enable Secure Debug Unlock: ✗

Enable Debug Lock: ✗

Disable Device Erase: ✓

3. In the Kudelski keySTREAM service, claim your device by entering the chip UID from Simplicity Studio and clicking **Commit**.



Take ownership of devices by Chip Uid

Chip Uid: 842e14481e62

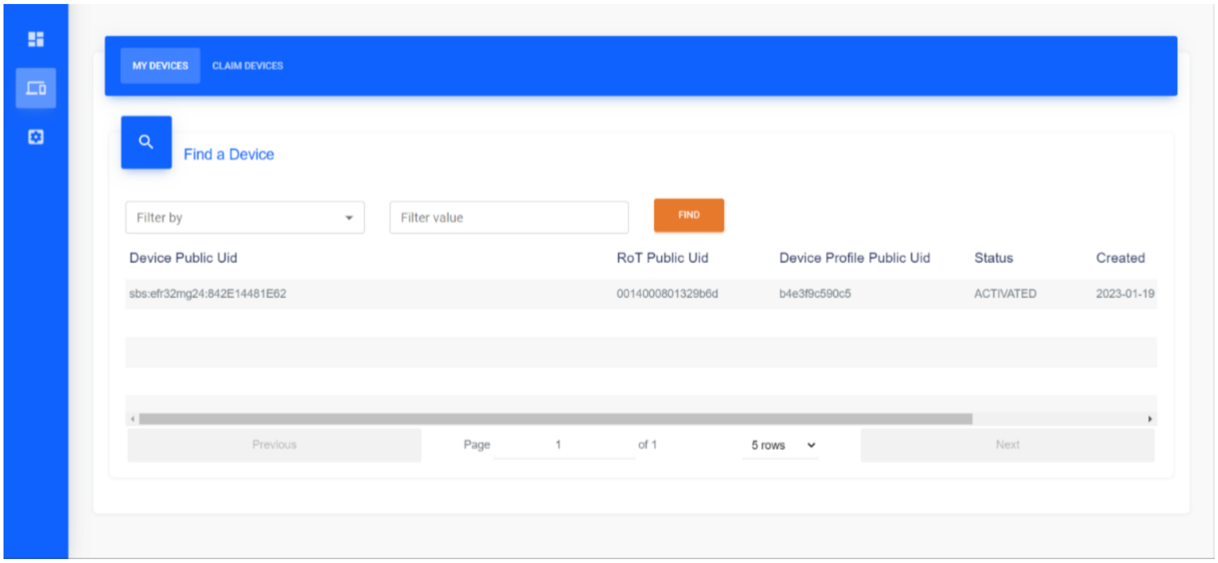
(Or)

Chip Uid list: No file chosen

Name	Reel Id	Chip Type	Date	No. of Chip Uids	Action
ManualCvRuleUid_e183e24b-0fe3-424e-8d1a-cf7c7a39a212	NA	sbs:efr32mg24	2022-09-20 17:21:05.000	3	🗑

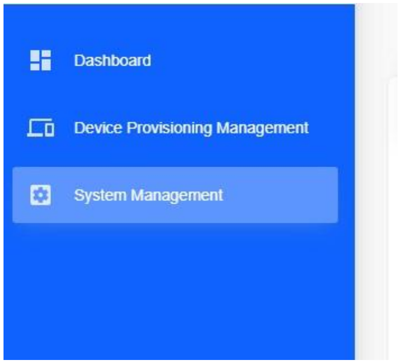
Page 1 of 1 10 rows

4. The device will now be visible in the **My Devices** tab. A device can be removed from the keySTREAM service by scrolling to the right and clicking the **Refurbish** button.

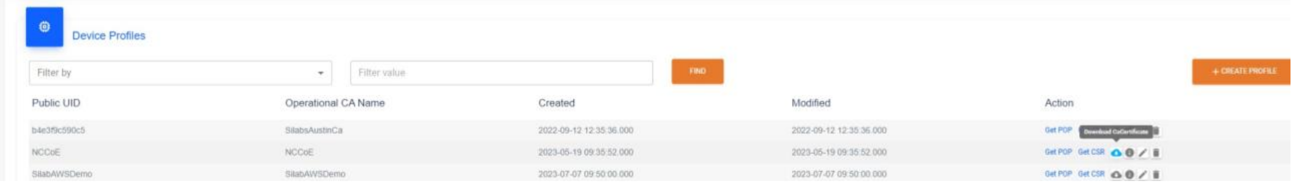


Device Public Uid	RoT Public Uid	Device Profile Public Uid	Status	Created
sbs:efr32mg24:842E14481E62	0014000801329b6d	b4e3f9c590c5	ACTIVATED	2023-01-19

5. Open the **System Management** tab on the left side:



6. Click the cloud icon to download the CA Certificate and the POP certificate, the POP certificate will require a code that is obtained from the AWS IoT Core which will be generated in Section 5.4.1.



Public UID	Operational CA Name	Created	Modified	Action
b4e3f9c590c5	SilabsAustinCa	2022-09-12 12:35:36.000	2022-09-12 12:35:36.000	Get POP
NCCoE	NCCoE	2023-05-19 09:35:52.000	2023-05-19 09:35:52.000	Get POP Get CSR
SilabAWSDemo	SilabAWSDemo	2023-07-07 09:50:00.000	2023-07-07 09:50:00.000	Get POP Get CSR

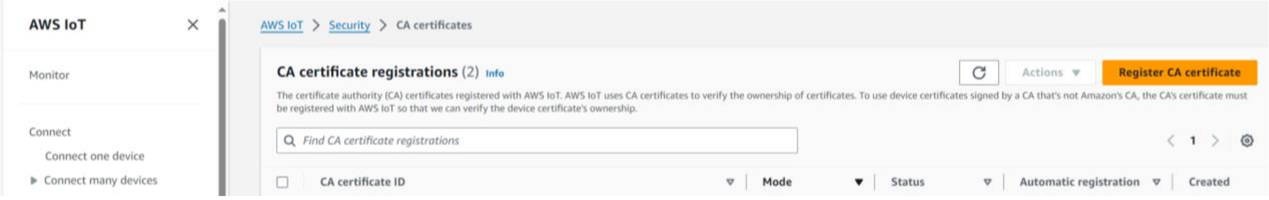
5.4 AWS IoT Core

The Silicon Labs Dev Kit will connect to the AWS MQTT test client using the certificates provisioned from the Kudelski keySTREAM service.

5.4.1 Setup and Configuration

Application-layer onboarding for this build is performed using the AWS MQTT test client. Certificates provisioned from the Kudelski keySTREAM service are uploaded to an AWS instance and the device will demonstrate its ability to successfully send a message to AWS.

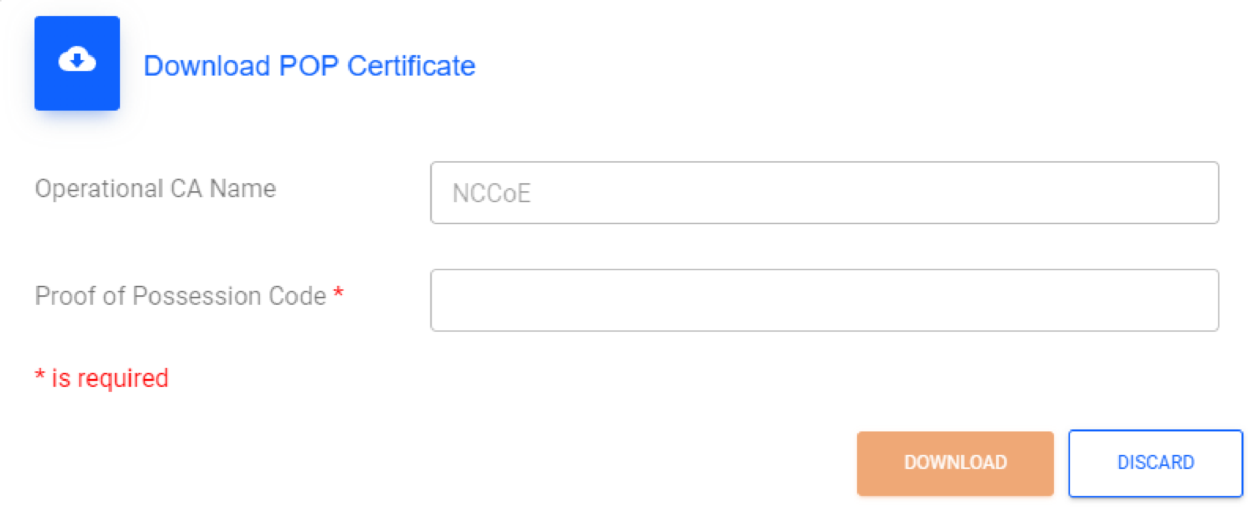
1. Within the AWS IoT Core, open the **Security** drop-down menu, click on **Certificate authorities**, and click the **Register CA certificate** button on the right.



CA certificate registrations (2) Info

The certificate authority (CA) certificates registered with AWS IoT. AWS IoT uses CA certificates to verify the ownership of certificates. To use device certificates signed by a CA that's not Amazon's CA, the CA's certificate must be registered with AWS IoT so that we can verify the device certificate's ownership.

2. Select the radio button for **Register CA in Single-account mode** and copy the registration code to use as the **Proof of Possession Code** in the Kudelski keySTREAM service and download the POP certificate.



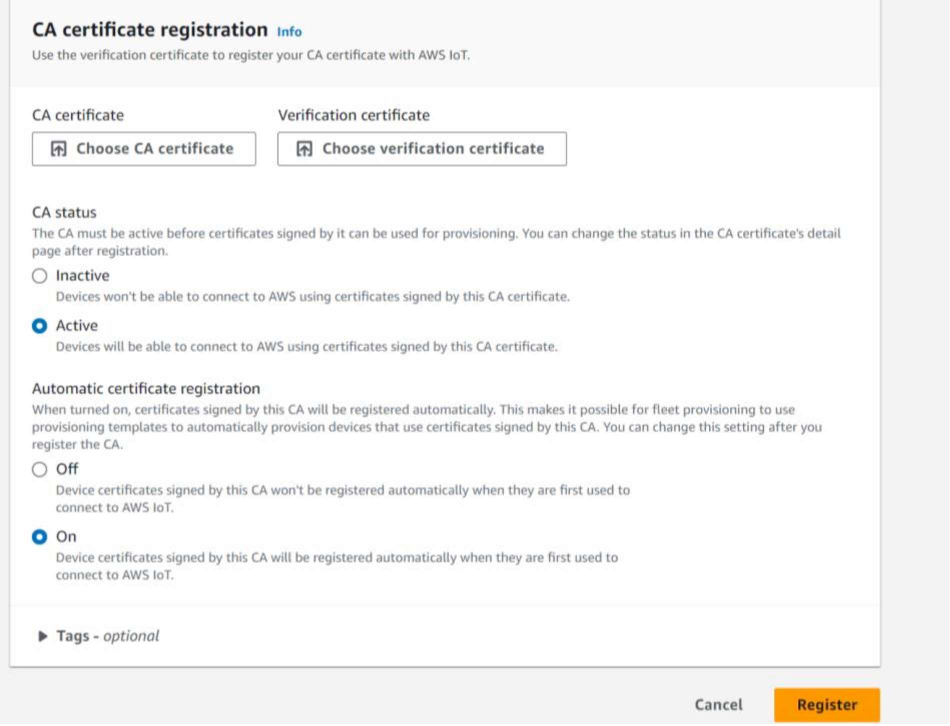
Download POP Certificate

Operational CA Name: NCCoE

Proof of Possession Code *:

* is required

3. After downloading the POP certificate, upload the CA certificate and the POP (verification) certificate, and select the radio buttons for **Active** under **CA Status** and **On** under **Automatic Certificate Registration**. Then click **Register**.



CA certificate registration Info

Use the verification certificate to register your CA certificate with AWS IoT.

CA certificate

Verification certificate

CA status

The CA must be active before certificates signed by it can be used for provisioning. You can change the status in the CA certificate's detail page after registration.

○ Inactive

Devices won't be able to connect to AWS using certificates signed by this CA certificate.

● Active

Devices will be able to connect to AWS using certificates signed by this CA certificate.

Automatic certificate registration

When turned on, certificates signed by this CA will be registered automatically. This makes it possible for fleet provisioning to use provisioning templates to automatically provision devices that use certificates signed by this CA. You can change this setting after you register the CA.

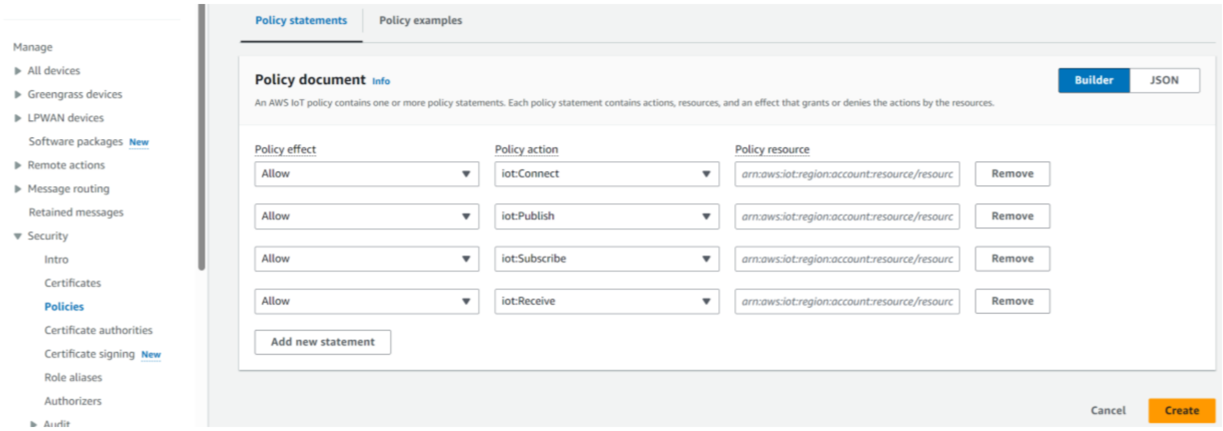
○ Off

Device certificates signed by this CA won't be registered automatically when they are first used to connect to AWS IoT.

● On

Device certificates signed by this CA will be registered automatically when they are first used to connect to AWS IoT.

4. In the **Security** drop down menu, click on **Policies** and add the policies shown below. Then, click **Create**.

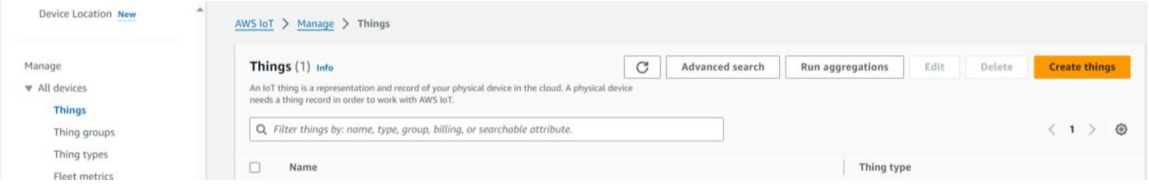


Policy document Info

An AWS IoT policy contains one or more policy statements. Each policy statement contains actions, resources, and an effect that grants or denies the actions by the resources.

Policy effect	Policy action	Policy resource
Allow	iot:Connect	arn:aws:iot:region:account:resource/resourc
Allow	iot:Publish	arn:aws:iot:region:account:resource/resourc
Allow	iot:Subscribe	arn:aws:iot:region:account:resource/resourc
Allow	iot:Receive	arn:aws:iot:region:account:resource/resourc

5. In the **All devices** drop-down menu, click on **Things** and click **Create things**.

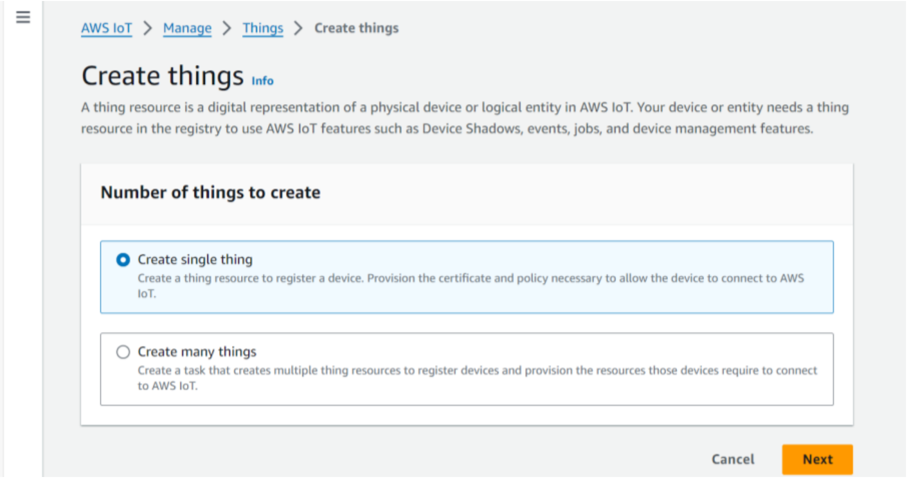


AWS IoT > Manage > Things

Things (1) Info

An IoT thing is a representation and record of your physical device in the cloud. A physical device needs a thing record in order to work with AWS IoT.

6. Click the **Create single thing** radio button and click **Next**.



AWS IoT > Manage > Things > Create things

Create things Info

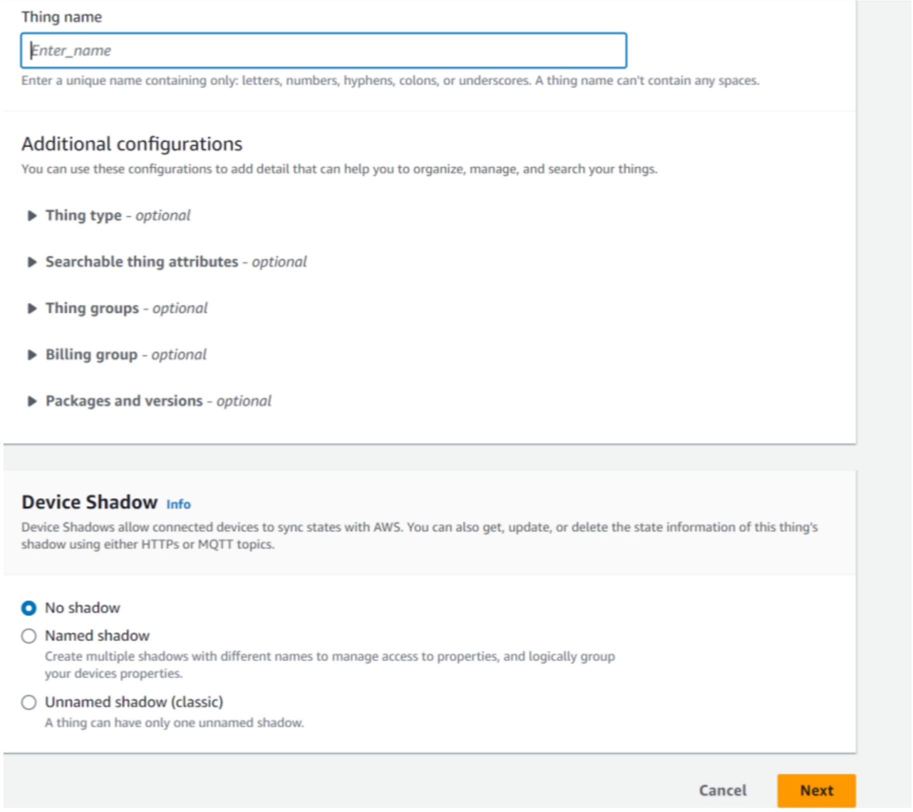
A thing resource is a digital representation of a physical device or logical entity in AWS IoT. Your device or entity needs a thing resource in the registry to use AWS IoT features such as Device Shadows, events, jobs, and device management features.

Number of things to create

● Create single thing
Create a thing resource to register a device. Provision the certificate and policy necessary to allow the device to connect to AWS IoT.

○ Create many things
Create a task that creates multiple thing resources to register devices and provision the resources those devices require to connect to AWS IoT.

7. Enter a **Thing name** and click **Next.**



Thing name

Enter_name

Enter a unique name containing only: letters, numbers, hyphens, colons, or underscores. A thing name can't contain any spaces.

Additional configurations

You can use these configurations to add detail that can help you to organize, manage, and search your things.

▶ **Thing type** - *optional*

▶ **Searchable thing attributes** - *optional*

▶ **Thing groups** - *optional*

▶ **Billing group** - *optional*

▶ **Packages and versions** - *optional*

Device Shadow Info

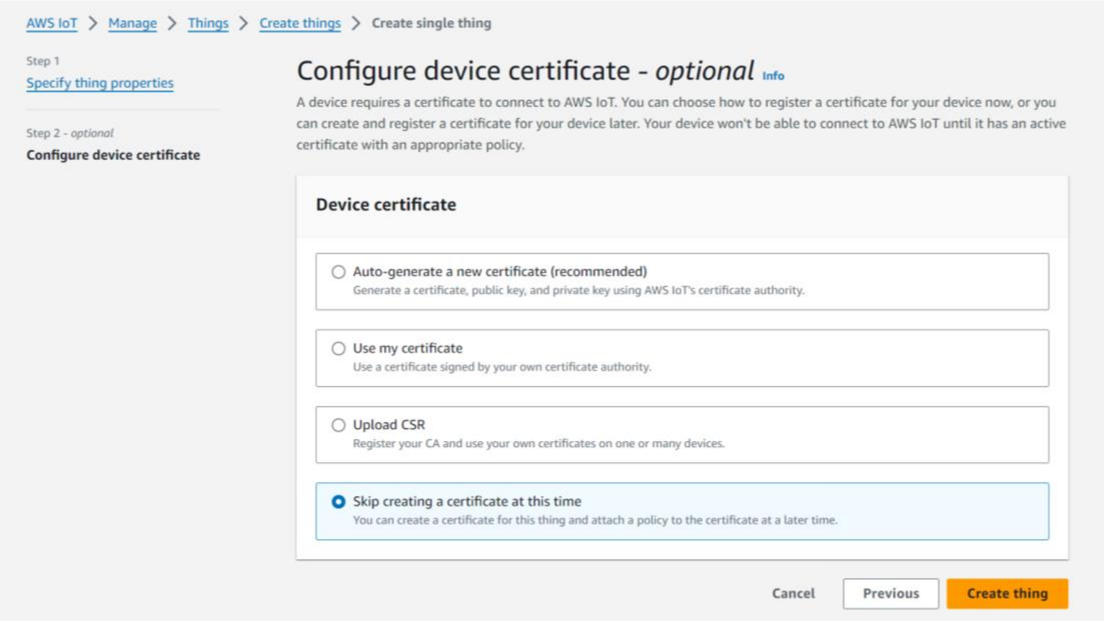
Device Shadows allow connected devices to sync states with AWS. You can also get, update, or delete the state information of this thing's shadow using either HTTPs or MQTT topics.

● No shadow

○ Named shadow
Create multiple shadows with different names to manage access to properties, and logically group your devices properties.

○ Unnamed shadow (classic)
A thing can have only one unnamed shadow.

8. Select the **Skip creating a certificate at this time** radio button and click **Create thing**.


AWS IoT > Manage > Things > Create things > Create single thing
Step 1
Specify thing properties
Step 2 - optional
Configure device certificate
Cancel
Previous
Create thing

Configure device certificate - *optional* Info

A device requires a certificate to connect to AWS IoT. You can choose how to register a certificate for your device now, or you can create and register a certificate for your device later. Your device won't be able to connect to AWS IoT until it has an active certificate with an appropriate policy.

Device certificate

○ Auto-generate a new certificate (recommended)
Generate a certificate, public key, and private key using AWS IoT's certificate authority.

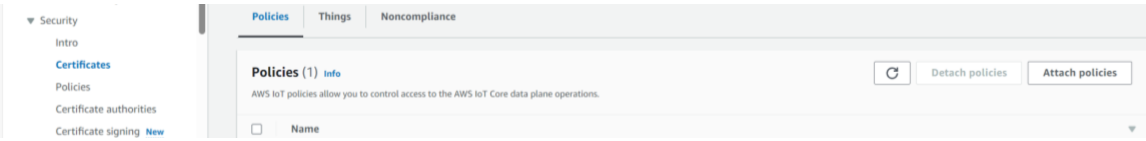
○ Use my certificate
Use a certificate signed by your own certificate authority.

○ Upload CSR
Register your CA and use your own certificates on one or many devices.

● Skip creating a certificate at this time
You can create a certificate for this thing and attach a policy to the certificate at a later time.

9. In the **Security** drop-down menu, click on **Certificates** and click the Certificate ID of the certificate that you created.

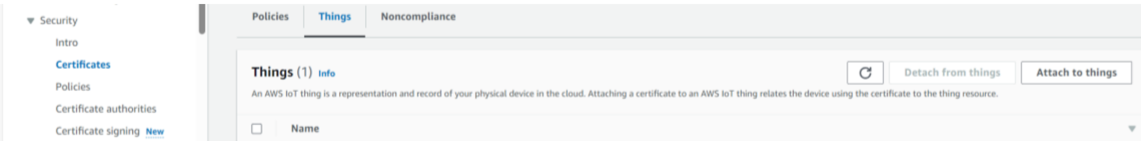
10. In the **Policies** tab at the bottom, click **Attach policies** and add the policy that you created.



Policies (1) Info

AWS IoT policies allow you to control access to the AWS IoT Core data plane operations.

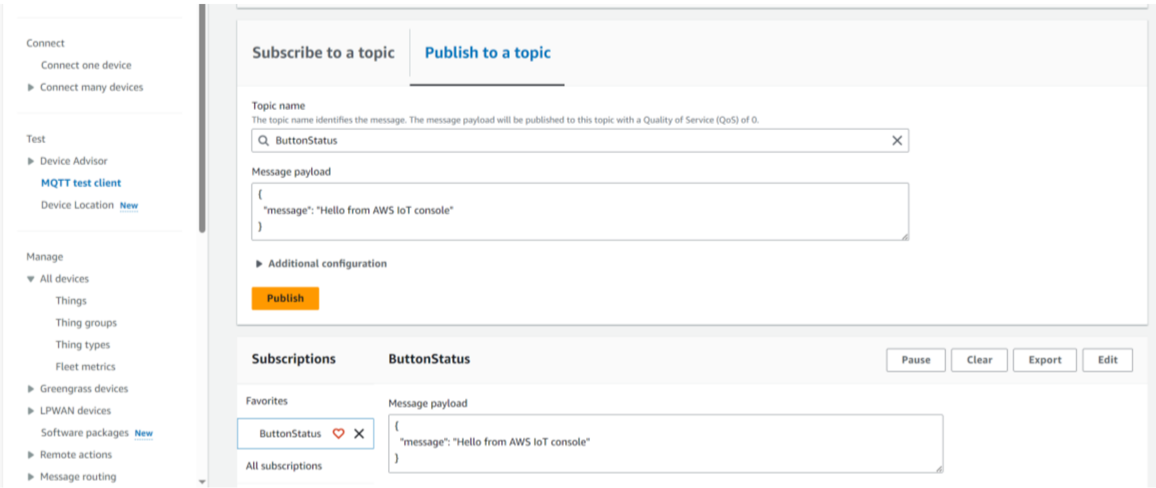
11. In the **Things** tab, click **Attach to things** and add the thing that you created.



Things (1) Info

An AWS IoT thing is a representation and record of your physical device in the cloud. Attaching a certificate to an AWS IoT thing relates the device using the certificate to the thing resource.

12. Click the **MQTT test client** on the left side of the page and click the **Publish to a topic** tab.



Topic name
The topic name identifies the message. The message payload will be published to this topic with a Quality of Service (QoS) of 0.
ButtonStatus

Message payload

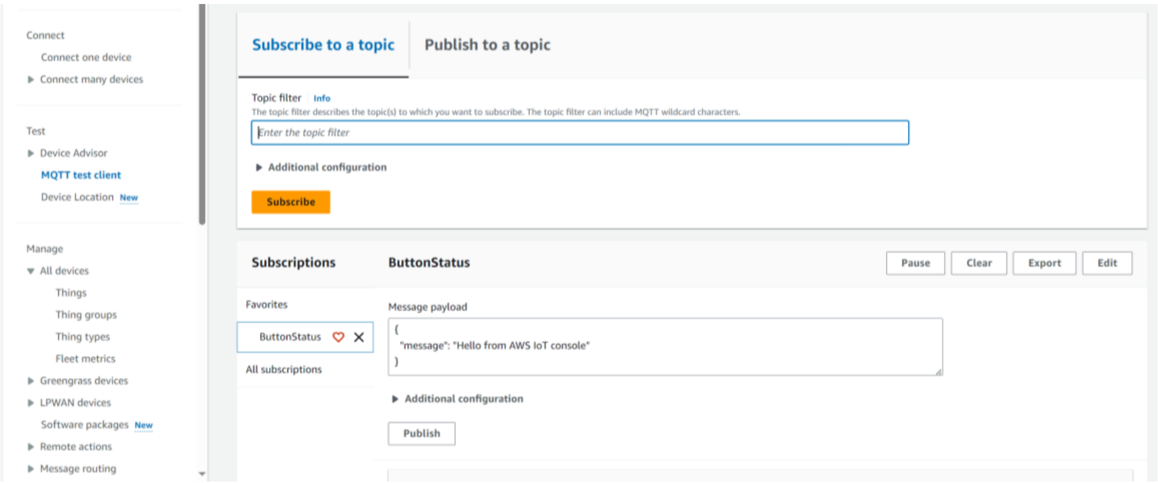
```
{
 "message": "Hello from AWS IoT console"
}
```

ButtonStatus

Message payload

```
{
 "message": "Hello from AWS IoT console"
}
```

13. Create a message of your choosing and click **Publish**. On the **Subscribe to a topic** tab, make sure that you are subscribed to the topic that you just created.



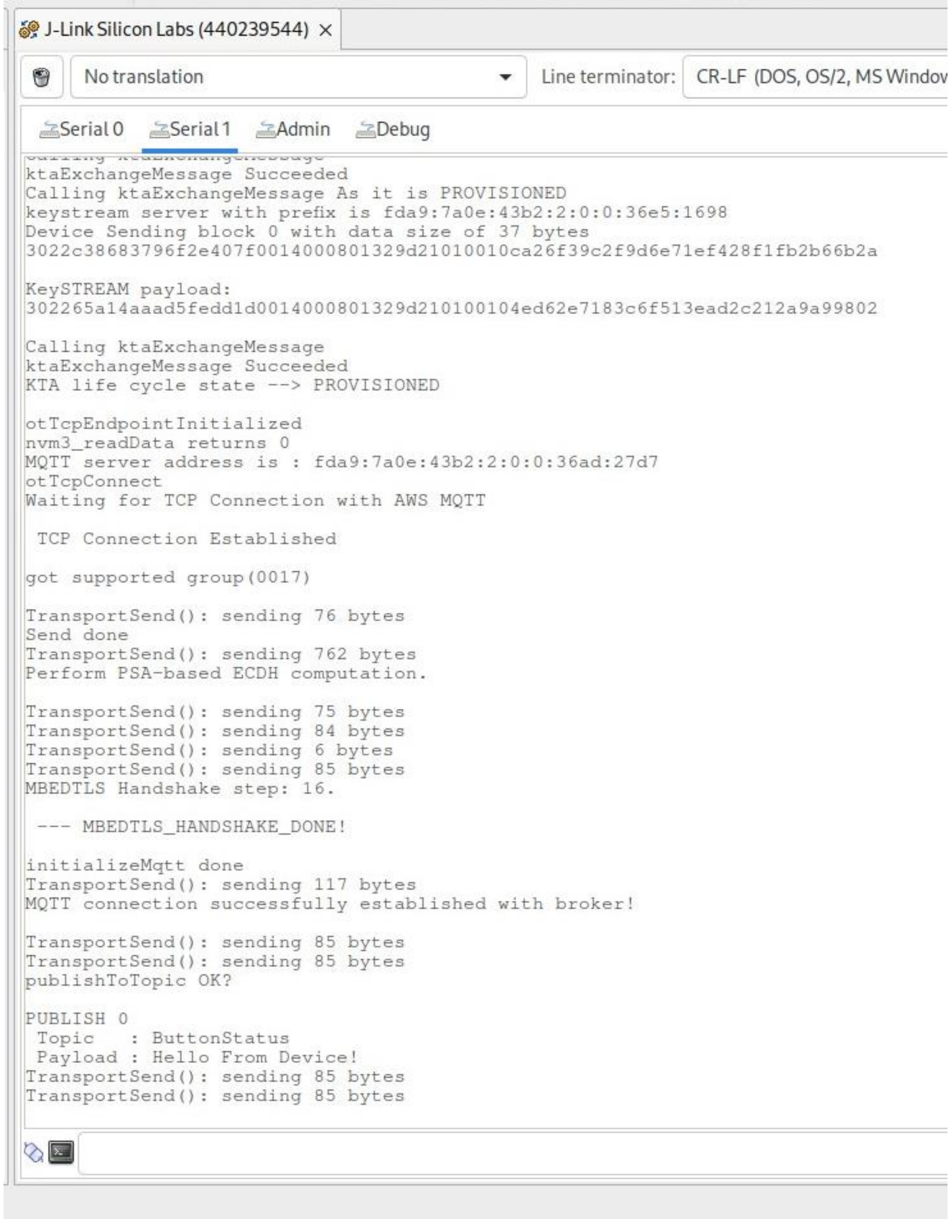
The topic filter describes the topic(s) to which you want to subscribe. The topic filter can include MQTT wildcard characters.

```
{
  "message": "Hello from AWS IoT console"
}
```

5.4.2 Testing

Information sent and received by the Silicon Labs Dev Kit to the MQTT test client will be displayed in the device console in Simplicity Commander. This section describes testing the communication between the MQTT test client and the device.

1. On the Thunderboard, press **Button 0**. This will begin the connection to the MQTT test client.



```
Calling ktaExchangeMessage
ktaExchangeMessage Succeeded
Calling ktaExchangeMessage As it is PROVISIONED
keystream server with prefix is fda9:7a0e:43b2:2:0:0:36e5:1698
Device Sending block 0 with data size of 37 bytes
3022c38683796f2e407f0014000801329d21010010ca26f39c2f9d6e71ef428f1fb2b66b2a

KeySTREAM payload:
302265a14aaad5fedd1d0014000801329d210100104ed62e7183c6f513ead2c212a9a99802

Calling ktaExchangeMessage
ktaExchangeMessage Succeeded
KTA life cycle state --> PROVISIONED

otTcpEndpointInitialized
nvm3_readData returns 0
MQTT server address is : fda9:7a0e:43b2:2:0:0:36ad:27d7
otTcpConnect
Waiting for TCP Connection with AWS MQTT

 TCP Connection Established

got supported group(0017)

TransportSend(): sending 76 bytes
Send done
TransportSend(): sending 762 bytes
Perform PSA-based ECDH computation.

TransportSend(): sending 75 bytes
TransportSend(): sending 84 bytes
TransportSend(): sending 6 bytes
TransportSend(): sending 85 bytes
MBEDTLS Handshake step: 16.

 --- MBEDTLS_HANDSHAKE_DONE!

initializeMqtt done
TransportSend(): sending 117 bytes
MQTT connection successfully established with broker!

TransportSend(): sending 85 bytes
TransportSend(): sending 85 bytes
publishToTopic OK?

PUBLISH 0
 Topic   : ButtonStatus
 Payload : Hello From Device!
TransportSend(): sending 85 bytes
TransportSend(): sending 85 bytes
```

6 Build 5 (BRSKI over Wi-Fi, NquiringMinds)

This section of the practice guide contains detailed instructions for installing and configuring all of the products used to build an instance of the example solution. For additional details on Build 5's logical and physical architectures, see NIST SP 1800-36B: *Approach, Architecture, and Security Characteristics*.

The network-layer onboarding component of Build 5 utilizes the BRSKI protocol.

6.1 Pledge

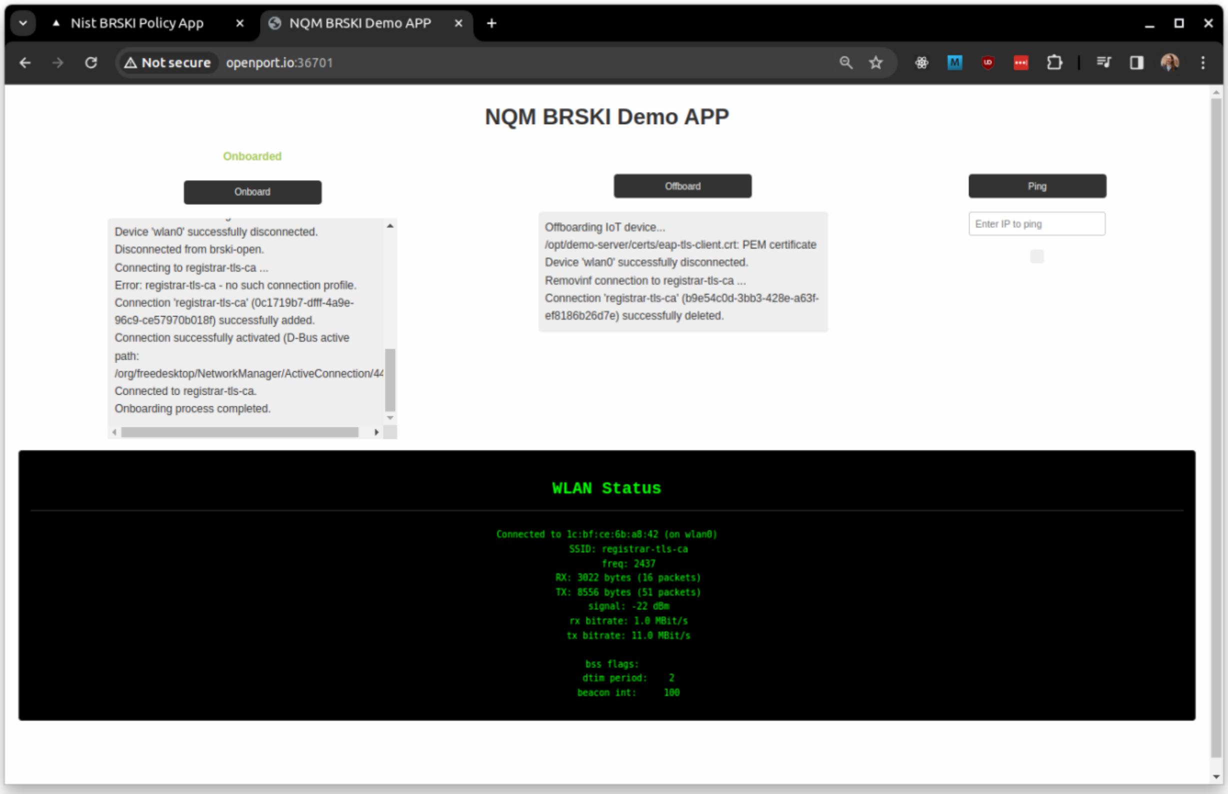
The Pledge acts as the IoT device which is attempted to onboard onto the secure network. It implements the pledge functionality as per the IETF BRSKI specification. It consists of software provided by NquiringMinds running on a Raspberry Pi Model 4B.

6.1.1 Installation and Configuration

Hardware requirements, pre-installation steps, installation steps, and configuration instructions for the pledge device can be found at the official NquiringMinds repository.

6.1.2 Operation and Demonstration

To demonstrate the onboarding and offboarding functionality, NquiringMinds has provided a web application which runs on the pledge device. It features a button one can use to manually run the onboarding script and display the output of the onboarding process, as well as a button for offboarding. It also features a button to ping an IP address, which is configured to ping the designated address via the wireless network interface.



```
Device 'wlan0' successfully disconnected.
Disconnected from brski-open.
Connecting to registrar-tls-ca ...
Error: registrar-tls-ca - no such connection profile.
Connection 'registrar-tls-ca' (0c1719b7-dfff-4a9e-
96c9-ce57970b018f) successfully added.
Connection successfully activated (D-Bus active
path:
/org/freedesktop/NetworkManager/ActiveConnection/44
Connected to registrar-tls-ca.
Onboarding process completed.
```

```
Offboarding IoT device...
/opt/demo-server/certs/eap-tls-client.crt: PEM certificate
Device 'wlan0' successfully disconnected.
Removinf connection to registrar-tls-ca ...
Connection 'registrar-tls-ca' (b9e54c0d-3bb3-428e-a63f-
ef8186b26d7e) successfully deleted.
```

```
Connected to 1c:bf:ce:6b:a8:42 (on wlan0)
        SSID: registrar-tls-ca
        freq: 2437
        RX: 3022 bytes (16 packets)
        TX: 8556 bytes (51 packets)
        signal: -22 dBm
        rx bitrate: 1.0 MBit/s
        tx bitrate: 11.0 MBit/s

        bss flags:
        dtim period:    2
        beacon int:     100
```

6.2 Router and Logical Services

The router and logical services were hosted on a Raspberry Pi Model 4B equipped with 2 external Wi-Fi adapters. These additional Wi-Fi adapters are needed to support VLAN tagging which is a hardware dependent feature. The details of the physical setup and all connections are provided in the official NquiringMinds documentation.

6.2.1 Installation and Configuration

All of the services described in the next section can be installed on a Raspberry Pi using the installer provided by NquiringMinds.

The demonstration services can also be built from source code, if needed. The following links provide the instructions for building each of those services:

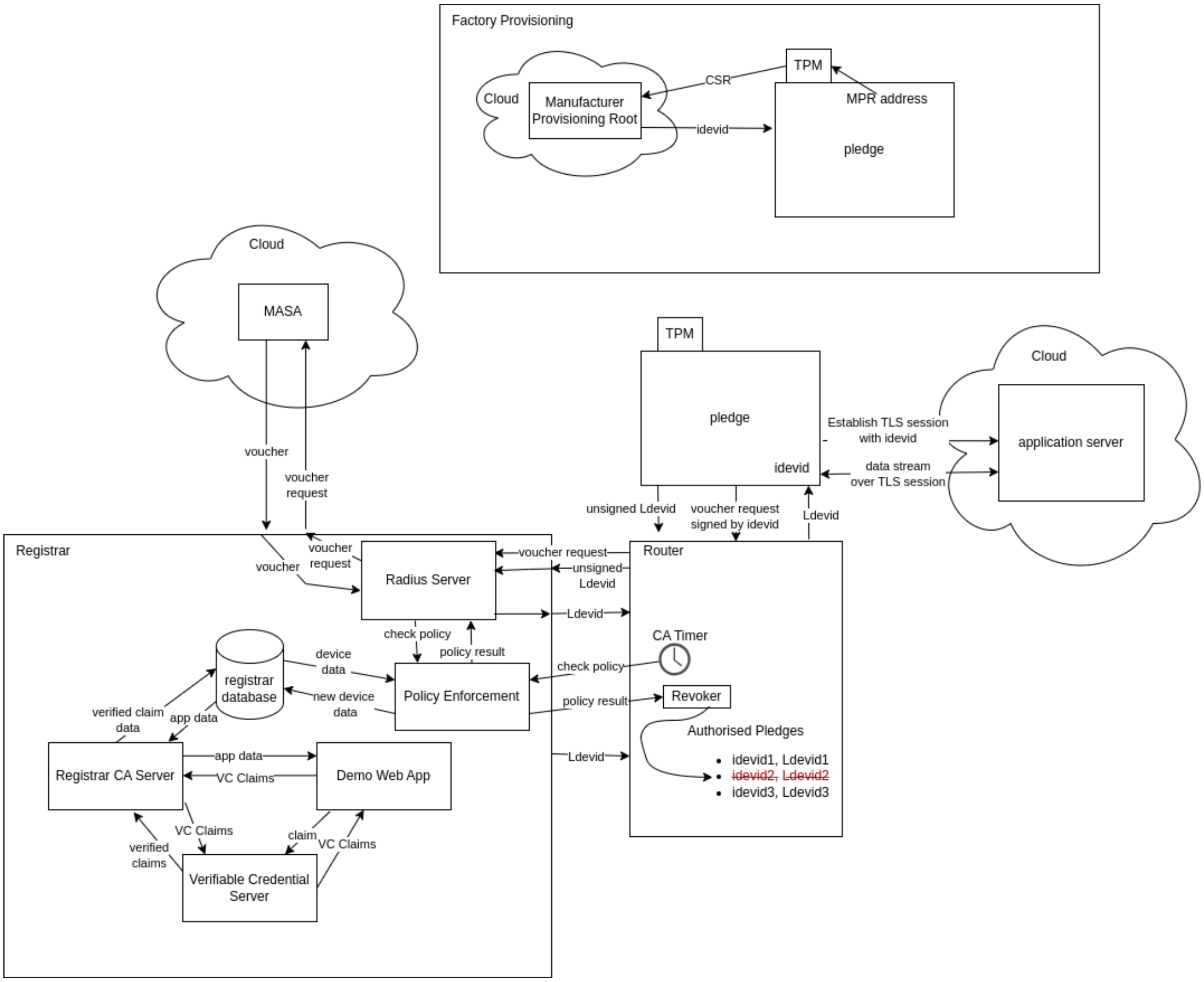
- BRSKI Demo Setup
- EAP Config
- MDNS publishing services

6.2.2 Logical services

The following logical services are installed on the Registrar and services device. The implementation of these services are to be found at the following repository links: NIST BRSKI implementation and BRSKI.

Figure 6-1 below describes how these entities and logical services fit together to perform the BRSKI flow, and a top-level view of how information is transmitted throughout the services to onboard the pledge.

Figure 6-1 Logical Services for Build 5



6.2.2.1 MASA

The MASA currently resides as a local service on the registrar. In practice, this service would be located on an external server managed by the manufacturer. The MASA verifies that the IDevID is authentic, and that the IDevID was produced by the manufacturer's MPR.

6.2.2.2 Manufacturer Provisioning Root (MPR)

The MPR sits on an external server and provides the IDevID (X.509 Certificate) for the device to initialize it after production and notarize it with a unique identity. The address of the MPR is built into the firmware of the device at build time.

6.2.2.3 Registrar

Build 5's BRSKI Domain Registrar runs the BRSKI protocol modified to work over Wi-Fi and functions as the Domain Registrar to authenticate the IoT devices, receive and transfer voucher requests and responses to and from the MASA and ultimately determines whether network-layer onboarding of the device is authorized to take place on the respective network. NquiringMinds has developed a stateful non-persistent Linux app for android that serves this purpose.

The registrar is responsible for verifying if the IDevID certificate provided by the pledge is authentic, by verifying it with the MASA and verifying that the policy for a pledge to be allowed onto the closed secure network has been met. It also runs continuous assurance periodically to ensure that the device still meets the policy requirements, revoking the pledge's access if at a later time it doesn't meet the policy requirements. Signed verifiable credential claims may be submitted to the registrar to communicate information about entities, which it uses to update its database used to determine if the policy is met, the tdx Volt is used to facilitate signing and verification of verifiable credentials. In the demonstrator system the MASA and router are integrated into the same physical device.

6.2.2.3.1 Radius server (Continuous Assurance Client)

To provide continuous assurance capabilities for connected IoT devices, the registrar includes a Radius server that integrates with the Continuous Assurance Server.

The continuous assurance policy is enforced by a script which periodically runs to check that the policy conditions are met. It accomplishes this by querying the Registrar's SQLite database. For the demonstration, the defined policy is:

- The manufacturer and device must be trusted by a user with appropriate privileges
- The device must have a device type associated
- The vulnerability score of the SBOM for the device type must be lower than 6
- The device must not have contacted a denylisted IP address within the last 2 minutes

If the device fails any of these checks, the device will be offboarded.

6.2.2.4 Continuous Assurance Server

The registrar runs several services used to power the continuous assurance flow.

6.2.2.4.1 Verifiable Credential Server

The verifiable credential server is used to sign verifiable credentials submitted through the Demo web app and verify verifiable credentials submitted to the registrar, it is powered by the functionality of the tdx Volt, a local instance of which is run on the registrar.

The code for the Verifiable Credential Server is hosted at the GitHub repository.

6.2.2.4.2 Registrar Continuous Assurance Server

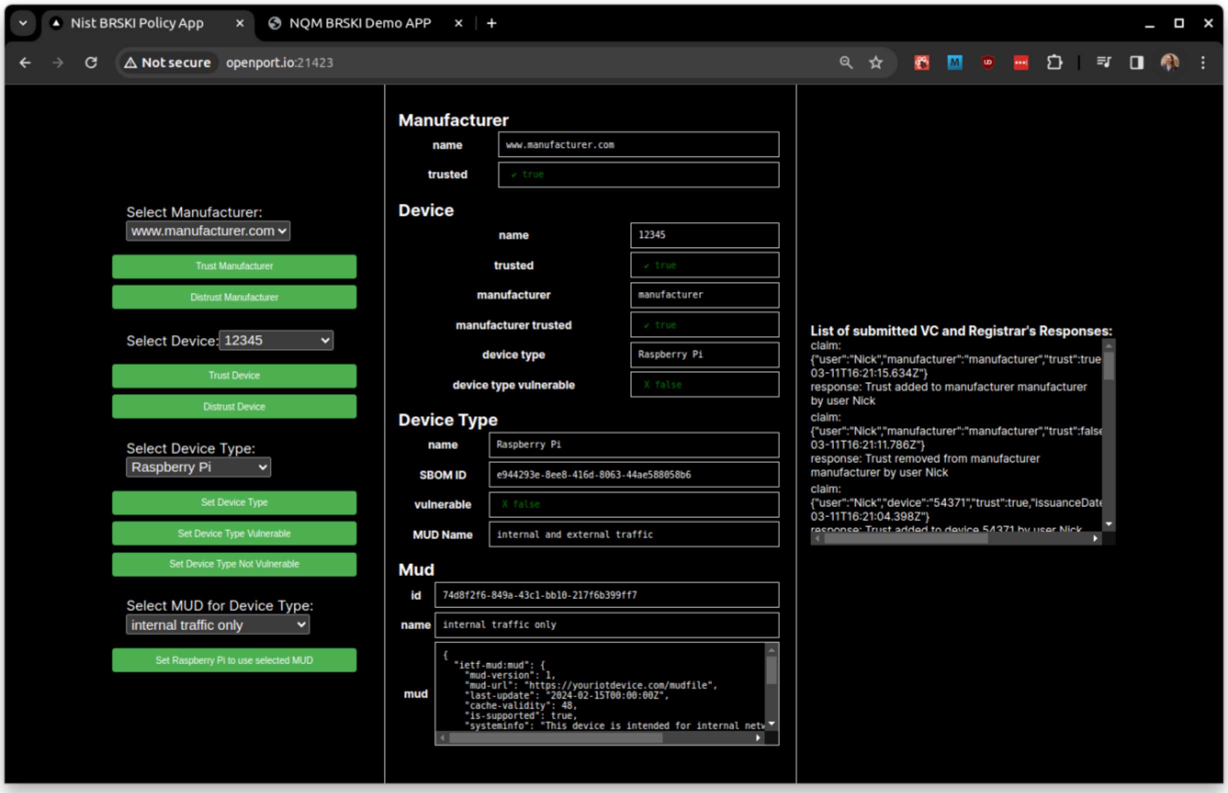
The registrar hosts a REST API which is used to interface with the registrar's SQLite database which stores information about the entities the registrar knows of. This server utilizes the verifiable credential server to verify submitted verifiable credential claims submitted to it.

The code for the Registrar Continuous Assurance Server is hosted at the GitHub repository.

6.2.2.4.3 Demo Web Application

The demo web application is used as an interactive user-friendly way to administer the registrar. Users can view the list of verifiable credentials submitted to the registrar. The application also displays the state of the manufacturers, devices, device types and Manufacturer Usage Description (MUD). There are buttons provided which allow you to trust or distrust a manufacturer, trust or distrust a device, set the device type for a device, set if a device type is vulnerable or not and set the MUD file associated with the device type. All of these operations are performed by generating a verifiable credential containing the claim being made, which is then submitted to the verifiable credential server to sign the credential. The signed verifiable credential is then sent to the registrar continuous assurance server to be verified and used to update the SQLite database on the registrar.

The code for the Demo Web Application is hosted at the GitHub repository.



Select Manufacturer: www.manufacturer.com

Select Device: 12345

Select Device Type: Raspberry Pi

Select MUD for Device Type: internal traffic only

Manufacturer

name: www.manufacturer.com

trusted: ✓ true

Device

name: 12345

trusted: ✓ true

manufacturer: manufacturer

manufacturer trusted: ✓ true

device type: Raspberry Pi

device type vulnerable: X false

Device Type

name: Raspberry Pi

SBOM ID: e944293e-8ee8-416d-8063-44ae588058b6

vulnerable: X false

MUD Name: internal and external traffic

Mud

id: 74d8f2f6-849a-43c1-bb10-217f6b399ff7

name: internal traffic only

mud:

```
{
  "ietf-mud:mud": {
    "mud-version": 1,
    "mud-url": "https://youriotdevice.com/mudfile",
    "last-update": "2024-02-15T00:00:00Z",
    "cache-validity": 48,
    "is-supported": true,
    "systeminfo": "This device is intended for internal netw
```

List of submitted VC and Registrar's Responses:

```
claim:
{"user":"Nick","manufacturer":"manufacturer","trust":true
03-11T16:21:15.634Z"}
response: Trust added to manufacturer manufacturer
by user Nick
claim:
{"user":"Nick","manufacturer":"manufacturer","trust":false
03-11T16:21:11.786Z"}
response: Trust removed from manufacturer
manufacturer by user Nick
claim:
{"user":"Nick","device":"54371","trust":true,"issuanceDate
03-11T16:21:04.398Z"}
response: Trust added to device 54371 by user Nick
```

6.2.2.5 Application server

The application server sits on a remote server and represents the server for an application which should consume data from the pledge device. The pledge device uses the IDevID certificate to establish a secure TLS connection to onboard onto the application server and begin sending data autonomously, currently OpenSSL s_client is used from the pledge to establish a TLS session with the application server, running on a server off-site, and the date and CPU temperature are sent to be logged on the application server, as a proof of principle.

6.2.2.5.1 Installation/Configuration

Hardware requirements, pre-installation steps, installation steps, and configuration instructions for the router can be found at the official NquiringMinds repository.

6.2.2.5.2 Operation/Demonstration

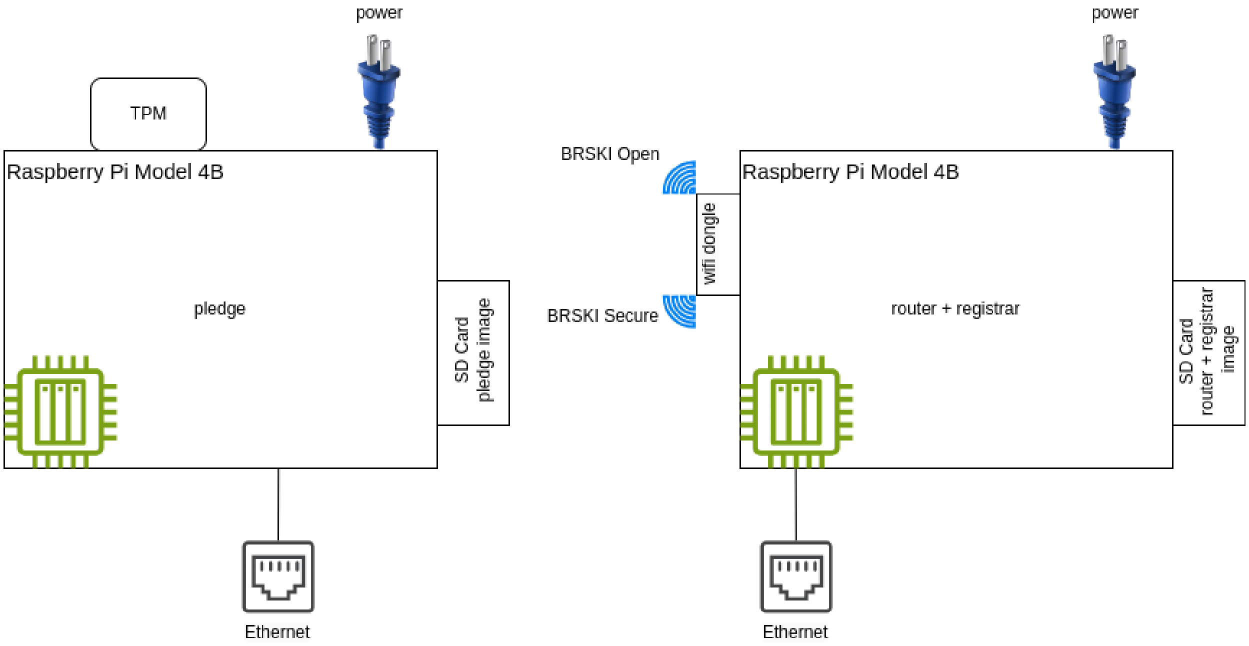
The instructions to use this factory use case code to provision an IDevID onto your pledge are also located at the official NquiringMinds repository in the above section.

6.3 Onboarding Demonstration

6.3.1 Prerequisites

Prior to beginning the demonstration, the router and pledge devices must be connected to power, and to the network via their ethernet port. On boot, both devices should start the services required to demonstrate the BRSKI flow.

Figure 6-2 Diagram of Physical/Logical Components Used to Demonstrate BRSKI Flow



To support the demo and debug features the pledge and the registrar need to be connected to physical ethernet, ideally with internet access. They should still function without an internet connection, but the vulnerability scores of the SBOMs will not be updated and the demo web apps will only be accessible on the local network.

The detailed networking setup details are available in the NquiringMinds NIST Trusted Onboarding Build-5.

6.3.2 Onboarding Demonstration

Once configuration of the devices and the prerequisite conditions have been achieved, the onboarding demonstration can be executed following NquiringMinds Demo Continuous Assurance Workflow.

6.3.3 Continuous Assurance Demonstration

The instructions to demonstrate the continuous assurance workflow are contained in the official NquiringMinds documentation.

6.4 BRSKI Factory Provisioning Build

This Factory Provisioning Build includes many of the components listed in Section 6.2, including the Pledge, Registrar, and other services. An Infineon Secure Element was also included in the build and provides secure generation and storage of the key material and certificates provisioned to the device.

6.4.1 Pledge

The Pledge acts as the IoT device which is attempting to onboard onto the secure network. It implements the pledge functionality as per the IETF BRSKI specification. It consists of a Raspberry Pi Model 4B equipped with an Infineon Optiga SLB 9670 TPM 2.0 Secure Element. The Infineon Secure Element was connected to a Raspberry Pi via the built-in GPIO pins present on the Pi.

6.4.1.1 Factory Use Case - IDevID provisioning

NquiringMinds provided demonstration code that generates a public/private keypair within the secure element, creates a CSR, and uses that CSR to obtain an IDevID certificate from tdx Volt. The demonstration process can be found at the official NquiringMinds documentation.

Initially, it generates a CSR using the TPM secure element to sign it, it then sends the CSR to the MPR server which is the manufacturer's IDevID Certificate Authority and is bootstrapped in the vanilla firmware on the pledge's creation in the factory. The MPR sends back a unique IDevID for the pledge which it stores in its secure element.

The code for this is hosted at the official NquiringMinds repository.

6.4.2 Installation and Configuration

Hardware requirements, pre-installation steps, installation steps, and configuration instructions for the pledge can be found at the official NquiringMinds repository referenced above.

6.4.3 Operation and Demonstration

The instructions to use this factory provisioning use case code to provision an IDevID onto the pledge is also located in the official NquiringMinds repository referenced above.

994 Appendix A List of Acronyms

AKM	Authentication and Key Management
AOS	ArubaOS
AP	Access Point
API	Application Programming Interface
ASN.1	Abstract Syntax Notation One
AWS	Amazon Web Services
BRSKI	Bootstrapping Remote Secure Key Infrastructure
BSS	Basic Service Set
CA	Certificate Authority
CRADA	Cooperative Research and Development Agreement
CSR	Certificate Signing Request
DMZ	Demilitarized Zone
DPP	Device Provisioning Protocol (Wi-Fi Easy Connect)
EAP	Extensible Authentication Protocol
GPIO	General Purpose Input/Output
GUI	Graphical User Interface
HPE	Hewlett Packard Enterprise
IaaS	Infrastructure as a Service
IDevID	Initial Device Identifier
IEEE	Institute of Electrical and Electronics Engineers
IoT	Internet of Things
IPv4	Internet Protocol Version 4
IPv6	Internet Protocol Version 6
LDevID	Locally Significant Device Identifier
MASA	Manufacturer Authorized Signing Authority
MPR	Manufacturer Provisioning Root
MUD	Manufacturer Usage Description
MQTT	MQ Telemetry Transport

NCCoE	National Cybersecurity Center of Excellence
NIST	National Institute of Standards and Technology
OCF	Open Connectivity Foundation
OS	Operating System
OTBR	Open Thread Border Router
PNG	Portable Network Graphics
POP	Proof of Possession
QR	Quick-Response
RF	Radio Frequency
SBOM	Software Bill of Materials
SP	Special Publication
SoC	System-on-Chip
SSID	Service Set Identifier
TPM	Trusted Platform Module
UID	Unique Identifier
URI	Uniform Resource Identifier
USB	Universal Serial Bus
UXI	User Experience Insight
VLAN	Virtual Local Area Network
VM	Virtual Machine
WLAN	Wireless Local Area Network
WPA2	Wi-Fi Protected Access 2
WPA3	Wi-Fi Protected Access 3

Appendix B References

[1] Wi-Fi Alliance. *Wi-Fi Easy Connect*. Available: https://www.wi-fi.org/discover-wi-fi/wi-fi-easy-connect.

NIST SPECIAL PUBLICATION 1800-36D

Trusted Internet of Things (IoT) Device Network-Layer Onboarding and Lifecycle Management

Enhancing Internet Protocol-Based IoT Device and Network Security

Volume D:
Functional Demonstrations

Paul Watrobski
Murugiah Souppaya
National Cybersecurity Center of Excellence
Information Technology Laboratory

Andy Dolan
Kyle Haefner
Craig Pratt
Darshak Thakore
CableLabs,
Louisville, Colorado

Brecht Wyseur
Kudelski IoT, Cheseaux-sur-Lausanne,
Switzerland

Nick Allott
Ashley Setter
Nquiring Minds
Southampton, United Kingdom

Michael Richardson
Sandleman Software Works
Ontario, Canada

Mike Dow
Steve Egerter
Silicon Labs,
Austin, Texas

Chelsea Deane
Joshua Klosterman
Blaine Mulugeta
Charlie Rearick
Susan Symington
The MITRE Corporation
McLean, Virginia

May 2024

DRAFT

This publication is available free of charge from
https://www.nccoe.nist.gov/projects/trusted-iot-device-network-layer-onboarding-and-lifecycle-management

DISCLAIMER

Certain commercial entities, equipment, products, or materials may be identified by name or company logo or other insignia in order to acknowledge their participation in this collaboration or to describe an experimental procedure or concept adequately. Such identification is not intended to imply special status or relationship with NIST or recommendation or endorsement by NIST or NCCoE; neither is it intended to imply that the entities, equipment, products, or materials are necessarily the best available for the purpose.

National Institute of Standards and Technology Special Publication 1800-36D, Natl. Inst. Stand. Technol. Spec. Publ. 1800-36D, 51 pages, May 2024, CODEN: NSPUE2

FEEDBACK

You can improve this guide by contributing feedback. As you review and adopt this solution for your own organization, we ask you and your colleagues to share your experience and advice with us.

Comments on this publication may be submitted to: iot-onboarding@nist.gov.

Public comment period: May 31, 2024 through July 30, 2024

National Cybersecurity Center of Excellence
National Institute of Standards and Technology
100 Bureau Drive
Mailstop 2002
Gaithersburg, MD 20899
Email: nccoe@nist.gov

NATIONAL CYBERSECURITY CENTER OF EXCELLENCE

The National Cybersecurity Center of Excellence (NCCoE), a part of the National Institute of Standards and Technology (NIST), is a collaborative hub where industry organizations, government agencies, and academic institutions work together to address businesses' most pressing cybersecurity issues. This public-private partnership enables the creation of practical cybersecurity solutions for specific industries, as well as for broad, cross-sector technology challenges. Through consortia under Cooperative Research and Development Agreements (CRADAs), including technology partners—from Fortune 50 market leaders to smaller companies specializing in information technology security—the NCCoE applies standards and best practices to develop modular, adaptable example cybersecurity solutions using commercially available technology. The NCCoE documents these example solutions in the NIST Special Publication 1800 series, which maps capabilities to the NIST Cybersecurity Framework and details the steps needed for another entity to re-create the example solution. The NCCoE was established in 2012 by NIST in partnership with the State of Maryland and Montgomery County, Maryland.

To learn more about the NCCoE, visit https://www.nccoe.nist.gov/. To learn more about NIST, visit https://www.nist.gov.

NIST CYBERSECURITY PRACTICE GUIDES

NIST Cybersecurity Practice Guides (Special Publication 1800 series) target specific cybersecurity challenges in the public and private sectors. They are practical, user-friendly guides that facilitate the adoption of standards-based approaches to cybersecurity. They show members of the information security community how to implement example solutions that help them align with relevant standards and best practices, and provide users with the materials lists, configuration files, and other information they need to implement a similar approach.

The documents in this series describe example implementations of cybersecurity practices that businesses and other organizations may voluntarily adopt. These documents do not describe regulations or mandatory practices, nor do they carry statutory authority.

KEYWORDS

application-layer onboarding; bootstrapping; Internet of Things (IoT); Manufacturer Usage Description (MUD); network-layer onboarding; onboarding; Wi-Fi Easy Connect.

ACKNOWLEDGMENTS

We are grateful to the following individuals for their generous contributions of expertise and time.

Name	Organization
Amogh Guruprasad Deshmukh	Aruba, a Hewlett Packard Enterprise company
Dan Harkins	Aruba, a Hewlett Packard Enterprise company
Danny Jump	Aruba, a Hewlett Packard Enterprise company
Bart Brinkman	Cisco
Eliot Lear	Cisco
Peter Romness	Cisco
Tyler Baker	Foundries.io
George Grey	Foundries.io
David Griego	Foundries.io
Fabien Gremaud	Kudelski IoT
Faith Ryan	The MITRE Corporation
Toby Ealden	NquiringMinds
John Manslow	NquiringMinds
Antony McCaigue	NquiringMinds
Alexandru Mereacre	NquiringMinds
Loic Cavaille	NXP Semiconductors
Mihai Chelalau	NXP Semiconductors
Julien Delplancke	NXP Semiconductors
Anda-Alexandra Dorneanu	NXP Semiconductors

Name	Organization
Todd Nuzum	NXP Semiconductors
Nicusor Penisoara	NXP Semiconductors
Laurentiu Tudor	NXP Semiconductors
Michael Richardson	Sandelman Software Works
Karen Scarfone	Scarfone Cybersecurity
Steve Clark	SEALSQ, a subsidiary of WISeKey
Pedro Fuentes	SEALSQ, a subsidiary of WISeKey
Gweltas Radenac	SEALSQ, a subsidiary of WISeKey
Kalvin Yang	SEALSQ, a subsidiary of WISeKey

The Technology Partners/Collaborators who participated in this build submitted their capabilities in response to a notice in the Federal Register. Respondents with relevant capabilities or product components were invited to sign a Cooperative Research and Development Agreement (CRADA) with NIST, allowing them to participate in a consortium to build this example solution. We worked with:

Technology Collaborators		
Aruba, a Hewlett Packard Enterprise company	Kudelski IoT	Sandelman Software Works
CableLabs	NquiringMinds	Silicon Labs
Cisco	NXP Semiconductors	SEALSQ, a subsidiary of WISeKey
Foundries.io	Open Connectivity Foundation (OCF)	

DOCUMENT CONVENTIONS

The terms “shall” and “shall not” indicate requirements to be followed strictly to conform to the publication and from which no deviation is permitted. The terms “should” and “should not” indicate that among several possibilities, one is recommended as particularly suitable without mentioning or excluding others, or that a certain course of action is preferred but not necessarily required, or that (in the negative form) a certain possibility or course of action is discouraged but not prohibited. The terms “may” and “need not” indicate a course of action permissible within the limits of the publication. The terms “can” and “cannot” indicate a possibility and capability, whether material, physical, or causal.

CALL FOR PATENT CLAIMS

This public review includes a call for information on essential patent claims (claims whose use would be required for compliance with the guidance or requirements in this Information Technology Laboratory (ITL) draft publication). Such guidance and/or requirements may be directly stated in this ITL Publication or by reference to another publication. This call also includes disclosure, where known, of the existence of pending U.S. or foreign patent applications relating to this ITL draft publication and of any relevant unexpired U.S. or foreign patents.

ITL may require from the patent holder, or a party authorized to make assurances on its behalf, in written or electronic form, either:

a) assurance in the form of a general disclaimer to the effect that such party does not hold and does not currently intend holding any essential patent claim(s); or

b) assurance that a license to such essential patent claim(s) will be made available to applicants desiring to utilize the license for the purpose of complying with the guidance or requirements in this ITL draft publication either:

1. under reasonable terms and conditions that are demonstrably free of any unfair discrimination; or
2. without compensation and under reasonable terms and conditions that are demonstrably free of any unfair discrimination.

Such assurance shall indicate that the patent holder (or third party authorized to make assurances on its behalf) will include in any documents transferring ownership of patents subject to the assurance, provisions sufficient to ensure that the commitments in the assurance are binding on the transferee, and that the transferee will similarly include appropriate provisions in the event of future transfers with the goal of binding each successor-in-interest.

The assurance shall also indicate that it is intended to be binding on successors-in-interest regardless of whether such provisions are included in the relevant transfer documents.

Such statements should be addressed to: iot-onboarding@nist.gov.

Contents

1 Introduction 1
1.1 How to Use This Guide 1
2 Functional Demonstration Playbook 3
2.1 Scenario 0: Factory Provisioning 3
2.2 Scenario 1: Trusted Network-Layer Onboarding 4
2.3 Scenario 2: Trusted Application-Layer Onboarding 5
2.4 Scenario 3: Re-Onboarding a Device 6
2.5 Scenario 4: Ongoing Device Validation 7
2.6 Scenario 5: Establishment and Maintenance of Credential and Device Security Posture Throughout the Lifecycle 8
3 Functional Demonstration Results 9
3.1 Build 1 Demonstration Results 9
3.2 Build 2 Demonstration Results 16
3.3 Build 3 Demonstration Results 22
3.4 Build 4 Demonstration Results 28
3.5 Build 5 Demonstration Results 34
Appendix A References 42

List of Tables

Table 2-1 Scenario 0 Factory Provisioning Capabilities That May Be Demonstrated 4
Table 2-2 Scenario 1 Trusted Network-Layer Onboarding Capabilities That May Be Demonstrated 5
Table 2-3 Scenario 2 Trusted Application-Layer Onboarding Capabilities That May Be Demonstrated 6
Table 2-4 Scenario 3 Re-Onboarding Capabilities That May Be Demonstrated 6
Table 2-5 Scenario 4 Ongoing Device Validation Capabilities That May Be Demonstrated 7
Table 2-6 Scenario 5 Credential and Device Posture Establishment and Maintenance Capabilities That May Be Demonstrated 8
Table 3-1 Build 1 Capabilities Demonstrated 9
Table 3-2 Build 2 Capabilities Demonstrated 16
Table 3-3 Build 3 Capabilities Demonstrated 22
Table 3-4 Build 4 Capabilities Demonstrated 28

Table 3-5 Build 5 Capabilities Demonstrated 35

1 Introduction

In this project, the National Cybersecurity Center of Excellence (NCCoE) is applying standards, recommended practices, and commercially available technology to demonstrate various mechanisms for trusted network-layer onboarding of IoT devices and lifecycle management of those devices. We show how to provision network credentials to IoT devices in a trusted manner and maintain a secure posture throughout the device lifecycle.

This volume of the NIST Cybersecurity Practice Guide describes functional demonstration scenarios that are designed to showcase the security capabilities and characteristics supported by trusted IoT device network-layer onboarding and lifecycle management solutions. Section 2, Functional Demonstration Playbook, defines the scenarios and lists the capabilities that can be showcased in each one. Section 3, Functional Demonstration Results, reports which capabilities have been demonstrated by each of the project's implemented solutions.

1.1 How to Use This Guide

This NIST Cybersecurity Practice Guide demonstrates a standards-based reference design for implementing trusted IoT device network-layer onboarding and lifecycle management and describes various example implementations of this reference design. Each of these implementations, which are known as *builds,* is standards-based and is designed to help provide assurance that networks are not put at risk as new IoT devices are added to them, and also to help safeguard IoT devices from being taken over by unauthorized networks. The reference design described in this practice guide is modular and can be deployed in whole or in part, enabling organizations to incorporate trusted IoT device network-layer onboarding and lifecycle management into their legacy environments according to goals that they have prioritized based on risk, cost, and resources.

NIST is adopting an agile process to publish this content. Each volume is being made available as soon as possible rather than delaying release until all volumes are completed.

This guide contains five volumes:

- NIST SP 1800-36A: *Executive Summary* – why we wrote this guide, the challenge we address, why it could be important to your organization, and our approach to solving this challenge
- NIST SP 1800-36B*: Approach, Architecture, and Security Characteristics* – what we built and why
- NIST SP 1800-36C: *How-To Guides* – instructions for building the example implementations, including all the security-relevant details that would allow you to replicate all or parts of this project
- NIST SP 1800-36D: *Functional Demonstrations* – use cases that have been defined to showcase trusted IoT device network-layer onboarding and lifecycle management security capabilities, and the results of demonstrating these use cases with each of the example implementations **(you are here)**
- NIST SP 1800-36E*: Risk and Compliance Management* – risk analysis and mapping of trusted IoT device network-layer onboarding and lifecycle management security characteristics to cybersecurity standards and recommended practices

Depending on your role in your organization, you might use this guide in different ways:

Business decision makers, including chief security and technology officers, will be interested in the *Executive Summary, NIST SP 1800-36A*, which describes the following topics:

- challenges that enterprises face in migrating to the use of trusted IoT device network-layer onboarding
- example solutions built at the NCCoE
- benefits of adopting the example solution

Technology or security program managers who are concerned with how to identify, understand, assess, and mitigate risk will be interested in *NIST SP 1800-36B*, which describes what we did and why.

Also, Section 4 of *NIST SP 1800-36E* will be of particular interest. Section 4, *Mappings*, maps logical components of the general trusted IoT device network-layer onboarding and lifecycle management reference design to security characteristics listed in various cybersecurity standards and recommended practices documents, including *Framework for Improving Critical Infrastructure Cybersecurity* (NIST Cybersecurity Framework) and *Security and Privacy Controls for Information Systems and Organizations* (NIST SP 800-53).

You might share the *Executive Summary, NIST SP 1800-36A*, with your leadership team members to help them understand the importance of using standards-based trusted IoT device network-layer onboarding and lifecycle management implementations.

IT professionals who want to implement similar solutions will find the whole practice guide useful. You can use the how-to portion of the guide, *NIST SP 1800-36C*, to replicate all or parts of the builds created in our lab. The how-to portion of the guide provides specific product installation, configuration, and integration instructions for implementing the example solution. We do not re-create the product manufacturers' documentation, which is generally widely available. Rather, we show how we incorporated the products together in our environment to create an example solution. Also, you can use *Functional Demonstrations, NIST SP 1800-36D*, which provides the use cases that have been defined to showcase trusted IoT device network-layer onboarding and lifecycle management security capabilities and the results of demonstrating these use cases with each of the example implementations. Finally, *NIST SP 1800-36E* will be helpful in explaining the security functionality that the components of each build provide.

This guide assumes that IT professionals have experience implementing security products within the enterprise. While we have used a suite of commercial products to address this challenge, this guide does not endorse these particular products. Your organization can adopt this solution or one that adheres to these guidelines in whole, or you can use this guide as a starting point for tailoring and implementing parts of a trusted IoT device network-layer onboarding and lifecycle management solution. Your organization's security experts should identify the products that will best integrate with your existing tools and IT system infrastructure. We hope that you will seek products that are congruent with applicable standards and recommended practices.

A NIST Cybersecurity Practice Guide does not describe "the" solution, but example solutions. We seek feedback on the publication's contents and welcome your input. Comments, suggestions, and success stories will improve subsequent versions of this guide. Please contribute your thoughts to iot-onboarding@nist.gov.

2 Functional Demonstration Playbook

Six scenarios have been defined that demonstrate capabilities related to various aspects of trusted IoT device network-layer onboarding, application-layer onboarding, and device lifecycle management. These scenarios are as follows:

- Scenario 0: Factory Provisioning
- Scenario 1: Trusted Network-Layer Onboarding
- Scenario 2: Trusted Application-Layer Onboarding
- Scenario 3: Re-Onboarding a Device
- Scenario 4: Ongoing Device Validation
- Scenario 5: Establishment and Maintenance of Credential and Device Security Posture Throughout the Lifecycle

We executed the factory provisioning scenario (Scenario 0) using both a Bootstrapping Remote Secure Key Infrastructure (BRSKI) Factory Provisioning Build and a Wi-Fi Easy Connect Factory Provisioning Build that have been implemented as part of this project. We executed the trusted network-layer onboarding and lifecycle management scenarios using each of the onboarding builds that have been implemented as part of this project. The capabilities that were demonstrated depend both on the features of the network-layer onboarding protocol (i.e., Wi-Fi Easy Connect) that the build supports and on any additional mechanisms the build may have integrated (e.g., application-layer onboarding).

Section 2.1 defines the factory provisioning scenario (Scenario 0). Sections 2.2 through Section 2.6 define each of the five onboarding scenarios.

2.1 Scenario 0: Factory Provisioning

This scenario, which simulates the IoT device factory provisioning process, is designed to represent some steps that must be performed in the factory before the device is put into the supply chain. These steps are performed by the device manufacturer or integrator to provision a device with the information it requires to be able to participate in trusted network-layer onboarding and lifecycle management. The device is assumed to have been equipped with secure storage and with the software or firmware needed to support a specific network-layer onboarding protocol (e.g., Wi-Fi Easy Connect or BRSKI). Scenario 0 includes initial provisioning of the IoT device with its birth credential (e.g., its private key and initial device identifier (IDevID) [1]), where it is stored in secure storage to prevent tampering or disclosure. This process includes generation of the credential (e.g., a private key and other information), signing of this credential (if applicable, depending on what onboarding protocol the device is designed to support), and transfer of the device bootstrapping information (e.g., a DPP URI or the device's IDevID) to the appropriate destination to ensure that it will be available for use during the network layer onboarding process. Following provisioning, the birth credential may be used for network-layer or application-layer onboarding. Table 2-1 lists the capabilities that may be demonstrated in this factory provisioning scenario.

Table 2-1 Scenario 0 Factory Provisioning Capabilities That May Be Demonstrated

Demo ID	Capability	Description
S0.C1	Birth Credential Generation and Storage	The device's birth credentials are generated within or generated and provisioned into secure storage on the IoT device. The content and format of the credential are appropriate to the onboarding protocol (e.g., Wi-Fi Easy Connect [2] or BRSKI [3]) that the device is designed to support: • For BRSKI, the credential is a private key, a signed certificate (IDevID), a trust anchor for the manufacturer's certificate authority (CA), and the location of a trusted manufacturer authorized signing authority (MASA). • For Wi-Fi Easy Connect, the credential is a private key and a public bootstrapping key.
S0.C2	Birth Credential Signing	The credential is signed by a trusted CA.
S0.C3	Bootstrapping Information Availability	The bootstrapping information required for onboarding the device is made available as needed. The format and content of the bootstrapping information depends on the onboarding protocol that the device is designed to support: • For BRSKI, the bootstrapping information is the certificate and ownership information that is sent to the MASA. • For Wi-Fi Easy Connect, the bootstrapping information is the Device Provisioning Protocol (DPP) uniform resource identifier (URI) (which contains the public key, and optionally other information such as device serial number).

2.2 Scenario 1: Trusted Network-Layer Onboarding

This scenario involves trusted network-layer onboarding of an authorized IoT device to a local network that is operated by the owner of the IoT device. The device is assumed to have been manufactured to support the type of network-layer onboarding protocol (e.g., Wi-Fi Easy Connect or BRSKI) that is being used by the local network. The device is also assumed to have been provisioned with its birth credential in a manner similar to that described in Scenario 0: Factory Provisioning, including transfer of the device's bootstrapping information (e.g., its public key) to the operator of the local network to ensure that this information will be available to support authentication of the device during the initial phase of the trusted network-layer onboarding process. Onboarding is performed after the device has booted up and is placed in onboarding mode. Because the organization that is operating the local network is the owner of the IoT device, the device is authorized to onboard to the network and the network is authorized to onboard the device. In this scenario, after the identities of the device and the network are authenticated, a *network onboarding component*—a logical component authorized to onboard devices on behalf of the network—authenticates the device and provisions unique network credentials to the device over a secure channel. These network credentials are not just specific to the device; they are also specific to the local network. The device then uses these credentials to connect to the network. Table 2-2 lists the capabilities that may be demonstrated in this scenario.

Table 2-2 Scenario 1 Trusted Network-Layer Onboarding Capabilities That May Be Demonstrated

Demo ID	Capability	Description
S1.C1	Device Authentication	The onboarding mechanism authenticates the device's identity.
S1.C2	Device Authorization	The onboarding mechanism verifies that the device is authorized to onboard to the network.
S1.C3	Network Authentication	The device can verify the network's identity.
S1.C4	Network Authorization	The device can verify that the network is authorized to take control of it.
S1.C5	Secure Local Credentialing	The onboarding mechanism securely provisions local network credentials to the device.
S1.C6	Secure Storage	The local network credentials are provisioned to secure hardware-backed storage on the device.
S1.C7	Network Selection	The onboarding mechanism provides the IoT device with the identifier of the network to which the device should onboard.
S1.C8	Interoperability	The network-layer onboarding mechanism can onboard a minimum of two types of IoT devices (e.g., different device vendors and models).

2.3 Scenario 2: Trusted Application-Layer Onboarding

This scenario involves trusted application-layer onboarding that is performed automatically on an IoT device after the device connects to a network. As a result, this scenario can be thought of as a series of steps that would be performed as an extension of Scenario 1, assuming the device has been designed and provisioned to support application-layer onboarding. As part of these steps, the device automatically mutually authenticates with a trusted application-layer onboarding service and establishes an encrypted connection to that service so the service can provision the device with application-layer credentials. The application-layer credentials could, for example, enable the device to securely connect to a trusted lifecycle management service to check for available updates or patches. For the application-layer onboarding mechanism to be trusted, it must establish an encrypted connection to the device without exposing any information that must be protected to ensure the confidentiality of that connection. Two types of application-layer onboarding are defined in NIST SP 1800-36B: *streamlined* and *independent*. Table 2-3 lists the capabilities that may be demonstrated in this scenario, including both types of application-layer onboarding.

Table 2-3 Scenario 2 Trusted Application-Layer Onboarding Capabilities That May Be Demonstrated

Demo ID	Capability	Description
S2.C1	Automatic Initiation of Streamlined Application-Layer Onboarding	The device can automatically (i.e., with no manual intervention required) initiate trusted application-layer onboarding after performing network-layer onboarding and connecting to the network. In this case, the application-layer onboarding bootstrapping information has been securely conveyed to the device during the network-layer onboarding process.
S2.C2	Automatic Initiation of Independent Application-Layer Onboarding	The device can automatically (i.e., with no manual intervention required) initiate trusted application-layer onboarding after performing network-layer onboarding and connecting to the network. In this case, the application-layer onboarding bootstrapping information has been pre-provisioned to the device by the device manufacturer or integrator (e.g., as part of an application that was installed on the device during the manufacturing process).
S2.C3	Trusted Application-Layer Onboarding	The device and a trusted application service can establish an encrypted connection without exposing any information that must be protected to ensure the confidentiality of the connection. They can then use that secure association to exchange application-layer information.

2.4 Scenario 3: Re-Onboarding a Device

This scenario involves re-onboarding an IoT device to a network after deleting its network credentials so that the device can be re-credentialed and reconnected. If the device also supports application-layer onboarding, application-layer onboarding should also be performed again after the device reconnects to the network. This scenario assumes that the device has been able to successfully demonstrate trusted network-layer onboarding as defined in Scenario 1: Trusted Network-Layer Onboarding. If application-layer re-onboarding is to be demonstrated as well, the scenario assumes that the device has also been able to successfully demonstrate at least one method of application-layer onboarding as defined in Scenario 2: Trusted Application-Layer Onboarding. Table 2-4 lists the capabilities that may be demonstrated in this scenario.

Table 2-4 Scenario 3 Re-Onboarding Capabilities That May Be Demonstrated

Demo ID	Capability	Description
S3.C1	Credential Deletion	The device's network credential can be deleted.
S3.C2	De-Credentialed Device Cannot Connect	After the device's network credential has been deleted, the device is not able to connect to or communicate on the network securely.
S3.C3	Re-Onboarding (network layer)	After the device's network credential has been deleted, the network-layer onboarding mechanism can securely re-provision a network

Demo ID	Capability	Description
		credential to the device, which the device can then use to connect to the network securely.
S3.C4	Re-Onboarding (application layer)	After the device's network and application-layer credentials have been deleted and the device has been re-onboarded at the network layer and reconnected to the network, the device can again perform trusted application-layer onboarding.

2.5 Scenario 4: Ongoing Device Validation

This scenario involves ongoing validation of a device, not only as part of a trusted boot or attestation process prior to permitting the device to undergo network-layer onboarding, but also after the device has connected to the network. It may involve one or more security mechanisms that are designed to evaluate, validate, or respond to device trustworthiness using methods such as examining device behavior, ensuring device authenticity and integrity, and assigning the device to a specific network segment based on its conformance to policy criteria. Table 2-5 lists the capabilities that may be demonstrated in this scenario. None of these capabilities are integral to trusted network-layer onboarding; however, they may be used in conjunction with, or subsequent to, trusted network-layer onboarding to enhance device and network security.

Table 2-5 Scenario 4 Ongoing Device Validation Capabilities That May Be Demonstrated

Demo ID	Capability	Description
S4.C1	Device Attestation (initial)	The network-layer onboarding mechanism requires successful device attestation prior to permitting the device to be onboarded.
S4.C2	Device Attestation (application layer)	The application-layer onboarding mechanism requires successful device attestation prior to permitting the device to be onboarded.
S4.C3	Device Attestation (ongoing)	Successful device attestation is required prior to permitting the device to perform some operation (e.g., accessing a high-value resource).
S4.C4	Local Network Segmentation (initial)	Upon connection, the IoT device is assigned to some local network segment in accordance with policy, which may include an assessment of its security posture.
S4.C5	Behavioral Analysis	Device behavior is observed to determine whether the device meets the policy criteria required to be permitted to perform a given operation (e.g., to access a high-value resource or be placed on a given network segment).
S4.C6	Local Network Segmentation (ongoing)	The IoT device can be reassigned to a different network segment based on ongoing assessments of its conformance to policy criteria.
S4.C7	Periodic Device Reauthentication	After connection, the IoT device's identity is periodically reauthenticated in order to maintain network access.

Demo ID	Capability	Description
S4.C8	Periodic Device Reauthorization	After connection, the IoT device's authorization to access the network is periodically reconfirmed in order to maintain network access.

2.6 Scenario 5: Establishment and Maintenance of Credential and Device Security Posture Throughout the Lifecycle

This scenario involves steps used to help establish and maintain the security posture of both the device's network credentials and the device itself. It includes the capability to download and validate the device's most recent firmware updates, securely integrate with a device communications intent enforcement mechanism (e.g., Manufacturer Usage Description (MUD) [4]), keep the device updated and patched, and establish and maintain the device's network credentials by provisioning X.509 certificates or DPP Connectors to the device and updating expired network credentials. Table 2-6 lists the capabilities that may be demonstrated in this scenario. None of these capabilities are integral to trusted network-layer onboarding; however, they may be used in conjunction with or subsequent to trusted network-layer onboarding to enhance device and network security.

Table 2-6 Scenario 5 Credential and Device Posture Establishment and Maintenance Capabilities That May Be Demonstrated

Demo ID	Capability	Description
S5.C1	Trusted Firmware Updates	The device can download the most recent firmware update and verify its signature before it is installed.
S5.C2	Credential Certificate Provisioning	The onboarding mechanism can interact with a certificate authority to sign a device's X.509 certificate and provision it onto the device.
S5.C3	Credential Update	The device's network credential can be updated after it expires.
S5.C4	Server Attestation	Successful server attestation is required prior to permitting the server to perform some operation on the device (e.g., prior to downloading and installing updates onto the device).
S5.C5	Secure Integration with MUD	The network-layer onboarding mechanism can convey necessary device communications intent information (e.g., the IoT device's MUD URL) to the network in encrypted form, thereby securely binding this information to the device and ensuring its confidentiality and integrity.
S5.C6	Lifecycle Management Establishment	The device has a lifecycle management service and can automatically establish a secure association with it after performing network-layer onboarding and connecting to the network.

3 Functional Demonstration Results

This section records the capabilities that were demonstrated for each of the builds.

3.1 Build 1 Demonstration Results

Table 3-1 lists the capabilities that were demonstrated by Build 1.

Table 3-1 Build 1 Capabilities Demonstrated

Demo ID	Capability	Description	Demonstrated?	Explanation/Notes
Scenario 0: Factory Provisioning				
S0.C1	Birth Credential Generation and Storage	The device's birth credentials are generated within or generated and provisioned into secure storage on the IoT device. For Wi-Fi Easy Connect, the credential is a private key and a public bootstrapping key.	Yes	Public/private key-pair is generated within the SEALSQ VaultIC secure element.
S0.C2	Birth Credential Signing	The credential is signed by a trusted CA.	No	There is no requirement to support this capability in this build. Birth credentials for devices supporting Wi-Fi Easy Connect onboarding do not need to be signed.
S0.C3	Bootstrapping Information Availability	The bootstrapping information required for onboarding the device is made available as needed. For Wi-Fi Easy Connect, the bootstrapping information is the Device Provisioning Protocol (DPP) uniform resource identifier (URI) (which contains the public key, and optionally other information such as device serial number).	Yes	The device's DPP URI is generated using the public/private keypair that was generated in the device's secure element. This DPP URI is encoded in a QR code that is written to a Portable Network Graphics (PNG) file and may be transferred from a vendor cloud upon acquisition of the device.

Demo ID	Capability	Description	Demonstrated?	Explanation/Notes
Scenario 1: Trusted Network-Layer Onboarding				
S1.C1	Device Authentication	The onboarding mechanism authenticates the device's identity.	Yes	DPP performs device authentication.
S1.C2	Device Authorization	The onboarding mechanism verifies that the device is authorized to onboard to the network.	Yes	When the device's URI is found on the HPE cloud service, this verifies that the device is authorized to onboard to the network.
S1.C3	Network Authentication	The device can verify the network's identity.	No	This could be supported by providing the IoT device with the DPP URI of the network, but the Aruba User Experience Insight (UXI) sensor used in this build lacks the user interface needed to do so.
S1.C4	Network Authorization	The device can verify that the network is authorized to take control of it.	Yes	The network that possesses the device's public key is implicitly authorized to onboard the device by virtue of its knowledge of the device's public key. While this is not cryptographic, it does provide a certain level of assurance that the "wrong" network doesn't take control of the device.
S1.C5	Secure Local Credentialing	The onboarding mechanism securely provisions local network credentials to the device.	Yes	DPP provisions the device's network credentials over an encrypted channel.
S1.C6	Secure Storage	The local network credentials are provisioned to secure hardware-backed storage on the device.	No	The bootstrapping credentials are stored in a Trusted Platform Module (TPM) 2.0 hardware enclave, but the local network credentials are not
S1.C7	Network Selection	The onboarding mechanism provides	Yes	The network responds to device chirps.

Demo ID	Capability	Description	Demonstrated?	Explanation/Notes
		the IoT device with the identifier of the network to which the device should onboard.		
S1.C8	Interoperability	The network-layer onboarding mechanism can onboard a minimum of two types of IoT devices (e.g., different device vendors and models).	Yes	IoT devices from Build 2 were successfully onboarded in Build 1.
Scenario 2: Trusted Application-Layer Onboarding				
S2.C1	Automatic Initiation of Streamlined Application-Layer Onboarding	The device can automatically (i.e., with no manual intervention required) initiate trusted application-layer onboarding after performing network-layer onboarding and connecting to the network. In this case, the application-layer onboarding bootstrapping information has been securely conveyed to the device during the network-layer onboarding process.	No	Not supported in this build.
S2.C2	Automatic Initiation of Independent Application-Layer Onboarding	The device can automatically (i.e., with no manual intervention required) initiate trusted application-layer onboarding after performing network-layer onboarding and connecting to the network. In this case, the application-layer onboarding bootstrapping information has been	Yes	Once onboarded, the UXI sensor automatically initiates application-layer onboarding to the UXI application.

Demo ID	Capability	Description	Demonstrated?	Explanation/Notes
		pre-provisioned to the device by the device manufacturer or integrator (e.g., as part of an application that was installed on the device during the manufacturing process).		
S2.C3	Trusted Application- Layer Onboarding	The device and a trusted application service can establish an encrypted connection without exposing any information that must be protected to ensure the confidentiality of the connection. They can then use that secure association to exchange application-layer information.	Yes	Once onboarded, the UXI sensor establishes a secure connection with the UXI cloud, which provisions the sensor with its credentials for the UXI application. Later, the sensor uploads data to the UXI application securely.
Scenario 3: Re-Onboarding a Device				
S3.C1	Credential Deletion	The device's network credential can be deleted.	Yes	Factory reset and manual credential removal were leveraged.
S3.C2	De-Credentialed Device Cannot Connect	After the device's network credential has been deleted, the device is not able to connect to or communicate on the network securely.	Yes	Observed.
S3.C3	Re-Onboarding (network layer)	After the device's network credential has been deleted, the network-layer onboarding mechanism can security re-provision a network credential to the device, which the device can then use to	Yes	Observed.

Demo ID	Capability	Description	Demonstrated?	Explanation/Notes
		connect to the network securely.		
S3.C4	Re-Onboarding (application layer)	After the device's network and application-layer credentials have been deleted and the device has been re-onboarded at the network layer and re-connected to the network, the device can again perform trusted application-layer onboarding.	Yes	Observed.
Scenario 4: Ongoing Device Validation				
S4.C1	Device Attestation (initial)	The network-layer onboarding mechanism requires successful device attestation prior to permitting the device to be onboarded.	No	Not supported in this build.
S4.C2	Device Attestation (application layer)	The application-layer onboarding mechanism requires successful device attestation prior to permitting the device to be onboarded.	No	Not supported in this build.
S4.C3	Device Attestation (ongoing)	Successful device attestation is required prior to permitting the device to perform some operation (e.g., accessing a high-value resource).	No	Not supported in this build.
S4.C4	Local Network Segmentation (initial)	Upon connection, the IoT device is assigned to some local network segment in accordance with policy, which may	No	Not demonstrated in this phase.

Demo ID	Capability	Description	Demonstrated?	Explanation/Notes
		include an assessment of its security posture.		
S4.C5	Behavioral Analysis	Device behavior is observed to determine whether the device meets the policy criteria required to be permitted to perform a given operation (e.g., to access a high-value resource or be placed on a given network segment).	No	Not supported in this build.
S4.C6	Local Network Segmentation (ongoing)	The IoT device can be reassigned to a different network segment based on ongoing assessments of its conformance to policy criteria.	No	Not supported in this build.
S4.C7	Periodic Device Reauthentication	After connection, the IoT device's identity is periodically reauthenticated in order to maintain network access.	No	Not supported in this build.
S4.C8	Periodic Device Reauthorization	After connection, the IoT device's authorization to access the network is periodically reconfirmed in order to maintain network access.	No	Not supported in this build.
Scenario 5: Establishment and Maintenance of Credential and Device Security Posture Throughout the Lifecycle				
S5.C1	Trusted Firmware Updates	The device can download the most recent firmware update and verify its signature before it is installed.	No	Not supported in this build.

Demo ID	Capability	Description	Demonstrated?	Explanation/Notes
S5.C2	Credential Certificate Provisioning	The onboarding mechanism can interact with a certificate authority to sign a device's X.509 certificate and provision it onto the device.	Yes	This capability has been successfully demonstrated with the SEALSQ INeS CA.
S5.C3	Credential Update	The device's network credential can be updated after it expires.	No	Not demonstrated in this phase.
S5.C4	Server Attestation	Successful server attestation is required prior to permitting the server to perform some operation on the device (e.g., prior to downloading and installing updates onto the device).	No	Not supported in this build.
S5.C5	Secure Integration with MUD	The network-layer onboarding mechanism can convey necessary device communications intent information (e.g., the IoT device's MUD URL) to the network in encrypted form, thereby securely binding this information to the device and ensuring its confidentiality and integrity.	No	Supported by DPP, but not demonstrated because Build 1 is not integrated with MUD or any other device communications intent enforcement mechanism.
S5.C6	Lifecycle Management Establishment	The device has a lifecycle management service and can automatically establish a secure association with it after performing network-layer onboarding and	No	Not supported in this build.

Demo ID	Capability	Description	Demonstrated?	Explanation/Notes
		connecting to the network.		

3.2 Build 2 Demonstration Results

Table 3-2 lists the capabilities that were demonstrated by Build 2.

Table 3-2 Build 2 Capabilities Demonstrated

Demo ID	Capability	Description	Demonstrated?	Explanation/Notes
Scenario 1: Trusted Network-Layer Onboarding				
S1.C1	Device Authentication	The onboarding mechanism authenticates the device's identity.	Yes	DPP performs device authentication.
S1.C2	Device Authorization	The onboarding mechanism verifies that the device is authorized to onboard to the network.	Yes	Only devices that have been added/approved by the administrator are onboarded. When the device's URI is found, the controller authorizes the device to join the network.
S1.C3	Network Authentication	The device can verify the network's identity.	No	This could be supported by providing the IoT device with the DPP URI of the network, but this is not currently implemented.
S1.C4	Network Authorization	The device can verify that the network is authorized to take control of it.	Yes	The network that possesses the device's public key is implicitly authorized to onboard the device by virtue of its knowledge of the device's public key. While this is not cryptographic, it does provide a certain level of assurance that the "wrong" network doesn't take control of the device.
S1.C5	Secure Local Credentialing	The onboarding mechanism securely provisions local	Yes	DPP provisions the device's network credentials over an encrypted channel.

Demo ID	Capability	Description	Demonstrated?	Explanation/Notes
		network credentials to the device.		
S1.C6	Secure Storage	The local network credentials are provisioned to secure hardware-backed storage on the device.	No	The IoT device does not have secure hardware-backed storage.
S1.C7	Network Selection	The onboarding mechanism provides the IoT device with the identifier of the network to which the device should onboard.	Yes	Network responds to device chirps.
S1.C8	Interoperability	The network-layer onboarding mechanism can onboard a minimum of two types of IoT devices (e.g., different device vendors and models).	Yes	Build 2 was able to onboard the IoT devices from Build 1.
Scenario 2: Trusted Application-Layer Onboarding				
S2.C1	Automatic Initiation of Streamlined Application-Layer Onboarding	The device can automatically (i.e., with no manual intervention required) initiate trusted application-layer onboarding after performing network-layer onboarding and connecting to the network. In this case, the application-layer onboarding bootstrapping information has been securely conveyed to the device during the network-layer onboarding process.	Yes	This has been demonstrated with the OCF Iotivity [5] custom extension. Iotivity is an open-source software framework that implements OCF standards and enables seamless device-to-device connectivity.
S2.C2	Automatic Initiation of Independent	The device can automatically (i.e., with no manual intervention required) initiate	No	Not supported in this build.

Demo ID	Capability	Description	Demonstrated?	Explanation/Notes
	Application-Layer Onboarding	trusted application-layer onboarding after performing network-layer onboarding and connecting to the network. In this case, the application-layer onboarding bootstrapping information has been pre-provisioned to the device by the device manufacturer or integrator (e.g., as part of an application that was installed on the device during the manufacturing process).		
S2.C3	Trusted Application-Layer Onboarding	The device and a trusted application service can establish an encrypted connection without exposing any information that must be protected to ensure the confidentiality of the connection. They can then use that secure association to exchange application-layer information.	Yes	Once the device is onboarded to the network using DPP, the credentials for the application layer onboarding are sent over the secure channel and provisioned by the onboarding tool (OBT).
Scenario 3: Re-Onboarding a Device				
S3.C1	Credential Deletion	The device's network credential can be deleted.	Yes	Supports factory reset.
S3.C2	De-Credentialed Device Cannot Connect	After the device's network credential has been deleted, the device is not able to connect to or communicate on the network securely.	Yes	Observed.

Demo ID	Capability	Description	Demonstrated?	Explanation/Notes
S3.C3	Re-Onboarding (network layer)	After the device's network credential has been deleted, the network-layer onboarding mechanism can security re-provision a network credential to the device, which the device can then use to connect to the network securely.	Yes	Observed.
S3.C4	Re-Onboarding (application layer)	After the device's network and application-layer credentials have been deleted and the device has been re-onboarded at the network layer and re-connected to the network, the device can again perform trusted application-layer onboarding.	Yes	Observed.
Scenario 4: Ongoing Device Validation				
S4.C1	Device Attestation (initial)	The network-layer onboarding mechanism requires successful device attestation prior to permitting the device to be onboarded.	No	Not supported in this build.
S4.C2	Device Attestation (application layer)	The application-layer onboarding mechanism requires successful device attestation prior to permitting the device to be onboarded.	No	Not supported in this build.
S4.C3	Device Attestation (ongoing)	Successful device attestation is required prior to permitting the device to perform some operation (e.g.,	No	Not supported in this build.

Demo ID	Capability	Description	Demonstrated?	Explanation/Notes
		accessing a high-value resource).		
S4.C4	Local Network Segmentation (initial)	Upon connection, the IoT device is assigned to some local network segment in accordance with policy, which may include an assessment of its security posture.	Yes	When the device is connected to the network, the gateway places it in a restricted network segment based on policy.
S4.C5	Behavioral Analysis	Device behavior is observed to determine whether the device meets the policy criteria required to be permitted to perform a given operation (e.g., to access a high-value resource or be placed on a given network segment).	No	Not supported in this build.
S4.C6	Local Network Segmentation (ongoing)	The IoT device can be reassigned to a different network segment based on ongoing assessments of its conformance to policy criteria.	Yes	Device can be moved to new network segments programmatically. The policy to do this is not defined in this build.
S4.C7	Periodic Device Reauthentication	After connection, the IoT device's identity is periodically reauthenticated in order to maintain network access.	No	Not supported in this build.
S4.C8	Periodic Device Reauthorization	After connection, the IoT device's authorization to access the network is periodically reconfirmed in order to maintain network access.	No	Not supported in this build.
Scenario 5: Establishment and Maintenance of Credential and Device Security Posture Throughout the Lifecycle				

Demo ID	Capability	Description	Demonstrated?	Explanation/Notes
S5.C1	Trusted Firmware Updates	The device can download the most recent firmware update and verify its signature before it is installed.	No	Not supported in this build.
S5.C2	Credential Certificate Provisioning	The onboarding mechanism can interact with a certificate authority to sign a device's X.509 certificate and provision it onto the device.	No	Not supported in this build.
S5.C3	Credential Update	The device's network credential can be updated after it expires.	No	Not demonstrated in this phase.
S5.C4	Server Attestation	Successful server attestation is required prior to permitting the server to perform some operation on the device (e.g., prior to downloading and installing updates onto the device).	No	Not supported in this build.
S5.C5	Secure Integration with MUD	The network-layer onboarding mechanism can convey necessary device communications intent information (e.g., the IoT device's MUD URL) to the network in encrypted form, thereby securely binding this information to the device and ensuring its confidentiality and integrity.	No	Supported by DPP, but not demonstrated because Build 2 is not integrated with MUD or any other device communications intent enforcement mechanism.

Demo ID	Capability	Description	Demonstrated?	Explanation/Notes
S5.C6	Lifecycle Management Establishment	The device has a lifecycle management service and can automatically establish a secure association with it after performing network-layer onboarding and connecting to the network.	No	Not supported in this build.

312

3.3 Build 3 Demonstration Results

313 Table 3-3 lists the capabilities that were demonstrated by Build 3.

314 **Table 3-3 Build 3 Capabilities Demonstrated**

Demo ID	Capability	Description	Demonstrated?	Explanation/Notes
Scenario 1: Trusted Network-Layer Onboarding				
S1.C1	Device Authentication	The onboarding mechanism authenticates the device's identity.	Yes	The local domain registrar receives the voucher request.
S1.C2	Device Authorization	The onboarding mechanism verifies that the device is authorized to onboard to the network.	Yes	The registrar verifies that the device is from an authorized manufacturer.
S1.C3	Network Authentication	The device can verify the network's identity.	Yes	Demonstrated by the voucher.
S1.C4	Network Authorization	The device can verify that the network is authorized to take control of it.	Yes	The registrar examines the new voucher and passes it to the device for onboarding.
S1.C5	Secure Local Credentialing	The onboarding mechanism securely provisions local network credentials to the device.	Yes	A local device identifier (LDevID) (i.e., the device's network credential) [1] is provisioned to the device after the device authentication and authorization process.

Demo ID	Capability	Description	Demonstrated?	Explanation/Notes
S1.C6	Secure Storage	The local network credentials are provisioned to secure hardware-backed storage on the device.	No	Not demonstrated in this phase.
S1.C7	Network Selection	The onboarding mechanism provides the IoT device with the identifier of the network to which the device should onboard.	No	Not demonstrated in this build.
S1.C8	Interoperability	The network-layer onboarding mechanism can onboard a minimum of two types of IoT devices (e.g., different device vendors and models).	No	Supported by BRSKI, but not demonstrated in this build.
Scenario 2: Trusted Application-Layer Onboarding				
S2.C1	Automatic Initiation of Streamlined Application-Layer Onboarding	The device can automatically (i.e., with no manual intervention required) initiate trusted application-layer onboarding after performing network-layer onboarding and connecting to the network. In this case, the application-layer onboarding bootstrapping information has been securely conveyed to the device during the network-layer onboarding process.	No	Not supported in this build.
S2.C2	Automatic Initiation of Independent Application-Layer Onboarding	The device can automatically (i.e., with no manual intervention required) initiate trusted application-layer onboarding after	No	Not supported in this build.

Demo ID	Capability	Description	Demonstrated?	Explanation/Notes
		performing network-layer onboarding and connecting to the network. In this case, the application-layer onboarding bootstrapping information has been pre-provisioned to the device by the device manufacturer or integrator (e.g., as part of an application that was installed on the device during the manufacturing process).		
S2.C3	Trusted Application-Layer Onboarding	The device and a trusted application service can establish an encrypted connection without exposing any information that must be protected to ensure the confidentiality of the connection. They can then use that secure association to exchange application-layer information.	No	Not supported in this build.
Scenario 3: Re-Onboarding a Device				
S3.C1	Credential Deletion	The device's network credential can be deleted.	Yes	Observed.
S3.C2	De-Credentialed Device Cannot Connect	After the device's network credential has been deleted, the device is not able to connect to or communicate on the network securely.	Yes	Observed.
S3.C3	Re-Onboarding (network-layer)	After the device's network credential has been deleted, the	Yes	Observed.

Demo ID	Capability	Description	Demonstrated?	Explanation/Notes
		network-layer onboarding mechanism can security re-provision a network credential to the device, which the device can then use to connect to the network securely.		
S3.C4	Re-Onboarding (application layer)	After the device's network credentials have been deleted and the device has been re-onboarded at the network layer and re-connected to the network, the device can perform application-layer onboarding automatically.	No	Not supported in this build.
Scenario 4: Ongoing Device Validation				
S4.C1	Device Attestation (initial)	The network-layer onboarding mechanism requires successful device attestation prior to permitting the device to be onboarded.	No	Not supported in this build.
S4.C2	Device Attestation (application layer)	The application-layer onboarding mechanism requires successful device attestation prior to permitting the device to be onboarded.	No	Not supported in this build.
S4.C3	Device Attestation (ongoing)	Successful device attestation is required prior to permitting the device to perform some operation (e.g., accessing a high-value resource).	No	Not supported in this build.

Demo ID	Capability	Description	Demonstrated?	Explanation/Notes
S4.C4	Local Network Segmentation (initial)	Upon connection, the IoT device is assigned to some local network segment in accordance with policy, which may include an assessment of its security posture.	No	Not supported in this build.
S4.C5	Behavioral Analysis	Device behavior is observed to determine whether the device meets the policy criteria required to be permitted to perform a given operation (e.g., to access a high-value resource or be placed on a given network segment).	No	Not supported in this build.
S4.C6	Local Network Segmentation (ongoing)	The IoT device can be reassigned to a different network segment based on ongoing assessments of its conformance to policy criteria.	No	Not supported in this build.
S4.C7	Periodic Device Reauthentication	After connection, the IoT device's identity is periodically reauthenticated in order to maintain network access.	No	Not supported in this build.
S4.C8	Periodic Device Reauthorization	After connection, the IoT device's authorization to access the network is periodically reconfirmed in order to maintain network access.	No	Not supported in this build.
Scenario 5: Establish and Maintain Credential and Device Security Posture Throughout the Lifecycle				
S5.C1	Trusted Firmware Updates	The device can download the most recent firmware update	No	Not supported in this build.

Demo ID	Capability	Description	Demonstrated?	Explanation/Notes
		and verify its signature before it is installed.		
S5.C2	Credential Certificate Provisioning	The onboarding mechanism can interact with a certificate authority to sign a device's X.509 certificate and provision it onto the device.	Yes	A vendor-installed X.509 certificate and a vendor's authorizing service use link-local connectivity to provision device credentials.
S5.C3	Credential Update	The device's network credential (e.g., its LDevID or X.509 certificate) can be updated after it expires.	No	Will be demonstrated in a future implementation of this build.
S5.C4	Server Attestation	Successful server attestation is required prior to permitting the server to perform some operation on the device (e.g., prior to downloading and installing updates onto the device).	No	Not supported in this build.
S5.C5	Secure Integration with MUD	The network-layer onboarding mechanism can convey necessary device communications intent information (e.g., the IoT device's MUD URL) to the network in encrypted form, thereby securely binding this information to the device and ensuring its confidentiality and integrity.	No	Supported by BRSKI, but not demonstrated because Build 3 is not integrated with MUD or any other device communications intent enforcement mechanism.
S5.C6	Lifecycle Management Establishment	The device has a lifecycle management service and can automatically establish a secure association	No	Not supported in this build.

Demo ID	Capability	Description	Demonstrated?	Explanation/Notes
		with it after performing network-layer onboarding and connecting to the network.		

315 ## 3.4 Build 4 Demonstration Results

316 Table 3-4 lists the capabilities that were demonstrated by Build 4.

317 **Table 3-4 Build 4 Capabilities Demonstrated**

Demo ID	Capability	Description	Demonstrated?	Explanation/Notes
Scenario 1: Trusted Network-Layer Onboarding				
S1.C1	Device Authentication	The onboarding mechanism authenticates the device's identity.	No	The build performs trusted application-layer onboarding only.
S1.C2	Device Authorization	The onboarding mechanism verifies that the device is authorized to onboard to the network.	No	The build performs trusted application-layer onboarding only.
S1.C3	Network Authentication	The device can verify the network's identity.	No	The build performs trusted application-layer onboarding only.
S1.C4	Network Authorization	The device can verify that the network is authorized to take control of it.	No	The build performs trusted application-layer onboarding only.
S1.C5	Secure Local Credentialing	The onboarding mechanism securely provisions local network credentials to the device.	No	The build performs trusted application-layer onboarding only.
S1.C6	Secure Storage	The local network credentials are provisioned to secure hardware-backed storage on the device.	Yes	The local network credentials are stored in the Silicon Labs Secure Vault on the Thunderboard.

Demo ID	Capability	Description	Demonstrated?	Explanation/Notes
S1.C7	Network Selection	The onboarding mechanism provides the IoT device with the identifier of the network to which the device should onboard.	No	The device generates a pre-shared key that is manually entered in the OpenThread Border Router [6].
S1.C8	Interoperability	The network-layer onboarding mechanism can onboard a minimum of two types of IoT devices (e.g., different device vendors and models).	No	Not supported in this build.
		Scenario 2: Trusted Application-Layer Onboarding		
S2.C1	Automatic Initiation of Streamlined Application-Layer Onboarding	The device can automatically (i.e., with no manual intervention required) initiate trusted application-layer onboarding after performing network-layer onboarding and connecting to the network. In this case, the application-layer onboarding bootstrapping information has been securely conveyed to the device during the network-layer onboarding process.	No	Not supported in this build.
S2.C2	Automatic Initiation of Independent Application-	The device can automatically (i.e., with no manual intervention required) initiate	Yes	Trusted application-layer onboarding using Kudelski keySTREAM is configured to proceed automatically pending

Demo ID	Capability	Description	Demonstrated?	Explanation/Notes
	Layer Onboarding	trusted application-layer onboarding after performing network-layer onboarding and connecting to the network. In this case, the application-layer onboarding bootstrapping information has been pre-provisioned to the device by the device manufacturer or integrator (e.g., as part of an application that was installed on the device during the manufacturing process).		confirmation from a user (through the press of a button).
S2.C3	Trusted Application-Layer Onboarding	The device and a trusted application service can establish an encrypted connection without exposing any information that must be protected to ensure the confidentiality of the connection. They can then use that secure association to exchange application-layer information.	Yes	Application Layer Onboarding via Kudelski keySTREAM GUI / AWS IoT Core and through the Silicon Labs Simplicity Studio Device Console
Scenario 3: Re-Onboarding a Device				
S3.C1	Credential Deletion	The device's network credential can be deleted.	Yes	The device can be removed from the network via the Open Thread Border Router

Demo ID	Capability	Description	Demonstrated?	Explanation/Notes
				GUI and cannot rejoin without entering a new pre-shared key.
S3.C2	De-Credentialed Device Cannot Connect	After the device's network credential has been deleted, the device is not able to connect to or communicate on the network securely.	Yes	Observed.
S3.C3	Re-Onboarding (network layer)	After the device's network credential has been deleted, the network-layer onboarding mechanism can security re-provision a network credential to the device, which the device can then use to connect to the network securely.	Yes	Observed.
S3.C4	Re-Onboarding (application layer)	After the device's network and application-layer credentials have been deleted and the device has been re-onboarded at the network layer and re-connected to the network, the device can again perform trusted application-layer onboarding.	Yes	Observed.
Scenario 4: Ongoing Device Validation				
S4.C1	Device Attestation (initial)	The network-layer onboarding mechanism requires successful device attestation prior to permitting the	No	Not supported in this build.

Demo ID	Capability	Description	Demonstrated?	Explanation/Notes
		device to be onboarded.		
S4.C2	Device Attestation (application layer)	The application-layer onboarding mechanism requires successful device attestation prior to permitting the device to be onboarded.	No	Not supported in this build.
S4.C3	Device Attestation (ongoing)	Successful device attestation is required prior to permitting the device to perform some operation (e.g., accessing a high-value resource).	No	Not supported in this build.
S4.C4	Local Network Segmentation (initial)	Upon connection, the IoT device is assigned to some local network segment in accordance with policy, which may include an assessment of its security posture.	No	Not supported in this build.
S4.C5	Behavioral Analysis	Device behavior is observed to determine whether the device meets the policy criteria required to be permitted to perform a given operation (e.g., to access a high-value resource or be placed on a given network segment).	No	Not supported in this build.
S4.C6	Local Network Segmentation (ongoing)	The IoT device can be reassigned to a different network	No	Not supported in this build.

Demo ID	Capability	Description	Demonstrated?	Explanation/Notes
		segment based on ongoing assessments of its conformance to policy criteria.		
S4.C7	Periodic Device Reauthentication	After connection, the IoT device's identity is periodically reauthenticated in order to maintain network access.	No	Not supported in this build.
S4.C8	Periodic Device Reauthorization	After connection, the IoT device's authorization to access the network is periodically reconfirmed in order to maintain network access.	No	Not supported in this build.
Scenario 5: Establishment and Maintenance of Credential and Device Security Posture Throughout the Lifecycle				
S5.C1	Trusted Firmware Updates	The device can download the most recent firmware update and verify its signature before it is installed.	No	Not supported in this build.
S5.C2	Credential Certificate Provisioning	The onboarding mechanism can interact with a certificate authority to sign a device's X.509 certificate and provision it onto the device.	No	Not supported in this build.
S5.C3	Credential Update	The device's network credential can be updated after it expires.	No	Not supported in this build.
S5.C4	Server Attestation	Successful server attestation is required prior to permitting the	No	Not supported in this build.

Demo ID	Capability	Description	Demonstrated?	Explanation/Notes
		server to perform some operation on the device (e.g., prior to downloading and installing updates onto the device).		
S5.C5	Secure Integration with MUD	The network-layer onboarding mechanism can convey necessary device communications intent information (e.g., the IoT device's MUD URL) to the network in encrypted form, thereby securely binding this information to the device and ensuring its confidentiality and integrity.	No	Not supported in this build.
S5.C6	Lifecycle Management Establishment	The device has a lifecycle management service and can automatically establish a secure association with it after performing network-layer onboarding and connecting to the network.	No	Not supported in this build.

318 3.5 Build 5 Demonstration Results

319 Table 3-5 lists the capabilities that were demonstrated by Build 5.

320 Table 3-5 Build 5 Capabilities Demonstrated

Demo ID	Capability	Description	Demonstrated?	Explanation/Notes
Scenario 0: Factory Provisioning				
S0.C1	Birth Credential Generation and Storage	The device's birth credentials are generated within or generated and provisioned into secure storage on the IoT device. For BRSKI, the credential is an IDevID certificate.	Yes	Supporting public/private keypair is generated within the secure element, and signed IDevID certificate is placed into the secure element.
S0.C2	Birth Credential Signing	The credential is signed by a trusted CA.	Yes	The IDevID certificate is signed by the Build 5 Manufacturer Provisioning Root (MPR).
S0.C3	Bootstrapping Information Availability	The bootstrapping information required for onboarding the device is made available as needed. For BRSKI, the bootstrapping information is the IDevID certificate provisioned into the device's secure element.	Yes	The device's IDevID certificate is generated using the public/private keypair that was generated in the device's secure element. This IDevID certificate is presented to verify the device's identity during network-layer onboarding.
Scenario 1: Trusted Network-Layer Onboarding				
S1.C1	Device Authentication	The onboarding mechanism authenticates the device's identity.	Yes	The device is authenticated using its provisioned IDevID.
S1.C2	Device Authorization	The onboarding mechanism verifies that the device is authorized to onboard to the network.	Yes	The device is implicitly granted authorization during the onboarding process within the registrar implementation. However, this authorization is contingent upon the device satisfying the policy

Demo ID	Capability	Description	Demonstrated?	Explanation/Notes
				requirements for onboarding.
S1.C3	Network Authentication	The device can verify the network's identity.	Yes	Demonstrated by the voucher.
S1.C4	Network Authorization	The device can verify that the network is authorized to take control of it.	Yes	The device authenticates to the network using EAP-TLS. The registrar gets a voucher from the MASA verifying that the network is authorized to onboard the device and it passes this voucher to the device so the device can verify that the network is authorized to onboard it.
S1.C5	Secure Local Credentialing	The onboarding mechanism securely provisions local network credentials to the device.	Yes	A local device identifier (LDevID) (i.e., the device's network credential) [1] is provisioned to the device as the culmination of the network-layer onboarding process.
S1.C6	Secure Storage	The local network credentials are provisioned to secure hardware-backed storage on the device.	No	The IDevID (birth credential) keys are generated with a TPM secure element. The EAP-TLS negotiation is configured to use keys from the secure element. The local network credentials (LDevID) are not scored in secure storage.
S1.C7	Network Selection	The onboarding mechanism provides the IoT device with the identifier of the network to which the device should onboard.	Yes	The identifier of the network is passed back in the common name field of the LDevID X.509 certificate.
S1.C8	Interoperability	The network-layer onboarding mechanism can onboard a minimum of two types of IoT devices (e.g., different device vendors and models).	Yes	Supported by BRSKI over IEEE 802.11 [7], but not demonstrated in this build.

Demo ID	Capability	Description	Demonstrated?	Explanation/Notes
Scenario 2: Trusted Application-Layer Onboarding				
S2.C1	Automatic Initiation of Streamlined Application-Layer Onboarding	The device can automatically (i.e., with no manual intervention required) initiate trusted application-layer onboarding after performing network-layer onboarding and connecting to the network. In this case, the application-layer onboarding bootstrapping information has been securely conveyed to the device during the network-layer onboarding process.	No	Not supported in this build
S2.C2	Automatic Initiation of Independent Application-Layer Onboarding	The device can automatically (i.e., with no manual intervention required) initiate trusted application-layer onboarding after performing network-layer onboarding and connecting to the network. In this case, the application-layer onboarding bootstrapping information has been pre-provisioned to the device by the device manufacturer or integrator (e.g., as part of an application that was installed on the device during the manufacturing process).	Yes	The pledge can use its IDevID and the private key in the secure element to automatically establish a TLS connection to an application server using OpenSSL s_client. The address of the application server has been pre-provisioned to the device by the manufacturer.

Demo ID	Capability	Description	Demonstrated?	Explanation/Notes
S2.C3	Trusted Application-Layer Onboarding	The device and a trusted application service can establish an encrypted connection without exposing any information that must be protected to ensure the confidentiality of the connection. They can then use that secure association to exchange application-layer information.	Yes	The pledge can use its IDevID and the private key in the secure element to automatically establish a TLS connection to an application server using OpenSSL s_client. The address of the application server has been pre-provisioned to the device by the manufacturer.
Scenario 3: Re-Onboarding a Device				
S3.C1	Credential Deletion	The device's network credential can be deleted.	Yes	The device is removed from Radius server by revoking its voucher.
S3.C2	De-Credentialed Device Cannot Connect	After the device's network credential has been deleted, the device is not able to connect to or communicate on the network securely.	Yes	If credential is removed from the registrar/radius server, the device will not connect. Certificate revocation through CRL is also implemented.
S3.C3	Re-Onboarding (network-layer)	After the device's network credential has been deleted, the network-layer onboarding mechanism can securely re-provision a network credential to the device, which the device can then use to connect to the network securely.	Yes	Upon a voucher being revoked, the LDevID is invalidated. The pledge can then perform the onboarding process again with a newly generated LDevID.
S3.C4	Re-Onboarding (application layer)	After the device's network credentials have been deleted and the device has been re-onboarded at the	Yes	After re-establishing a network connection, application onboarding happens automatically.

Demo ID	Capability	Description	Demonstrated?	Explanation/Notes
		network layer and re-connected to the network, the device can perform application-layer onboarding automatically.		
		Scenario 4: Ongoing Device Validation		
S4.C1	Device Attestation (initial)	The network-layer onboarding mechanism requires successful device attestation prior to permitting the device to be onboarded.	No	Not supported in this build.
S4.C2	Device Attestation (application layer)	The application-layer onboarding mechanism requires successful device attestation prior to permitting the device to be onboarded.	No	Not supported in this build.
S4.C3	Device Attestation (ongoing)	Successful device attestation is required prior to permitting the device to perform some operation (e.g., accessing a high-value resource).	No	Not supported in this build.
S4.C4	Local Network Segmentation (initial)	Upon connection, the IoT device is assigned to some local network segment in accordance with policy, which may include an assessment of its security posture.	No	Not supported in this build.
S4.C5	Behavioral Analysis	Device behavior is observed to determine whether the device meets the policy criteria required to be permitted to perform a given operation (e.g., to access a high-value	Yes	Real time network events are propagated from the gateway(s) to the policy engine. When suspicious behavior is identified (e.g., contact denylisted IP address) device network access is revoked.

Demo ID	Capability	Description	Demonstrated?	Explanation/Notes
		resource or be placed on a given network segment).		
S4.C6	Local Network Segmentation (ongoing)	The IoT device can be reassigned to a different network segment based on ongoing assessments of its conformance to policy criteria.	No	Not supported in this build.
S4.C7	Periodic Device Reauthentication	After connection, the IoT device's identity is periodically reauthenticated in order to maintain network access.	No	Not supported in this build.
S4.C8	Periodic Device Reauthorization	After connection, the IoT device's authorization to access the network is periodically reconfirmed in order to maintain network access.	Yes	The continuous assurance policy is checked periodically, every 30 seconds in the demo. The policy sets the requirements for a device to be authorized to have access to the network. If a device fails this check, its voucher is revoked, invalidating the device's LDevID.
Scenario 5: Establish and Maintain Credential and Device Security Posture Throughout the Lifecycle				
S5.C1	Trusted Firmware Updates	The device can download the most recent firmware update and verify its signature before it is installed.	No	Not supported in this build.
S5.C2	Credential Certificate Provisioning	The onboarding mechanism can interact with a certificate authority to sign a device's X.509 certificate and provision it onto the device.	Yes	In the BRSKI flows, the onboarding process results in an LDevID (X.509) certificate being provisioned on the device, after the trustworthiness checks have been completed. This LDevID certificate is signed by the Domain CA.

Demo ID	Capability	Description	Demonstrated?	Explanation/Notes
S5.C3	Credential Update	The device's network credential (e.g., its LDevID or X.509 certificate) can be updated after it expires.	Yes	Device will automatically generate a new LDevID and re-onboard if LDevID expires.
S5.C4	Server Attestation	Successful server attestation is required prior to permitting the server to perform some operation on the device (e.g., prior to downloading and installing updates onto the device).	No	Not supported in this build.
S5.C5	Secure Integration with MUD	The network-layer onboarding mechanism can convey necessary device communications intent information (e.g., the IoT device's MUD URL) to the network in encrypted form, thereby securely binding this information to the device and ensuring its confidentiality and integrity.	Yes	The continuous assurance policy engine sporadically resolves the MUD document of each unique connected device using all information available. In this build we use the D3DB method of resolution, which resolves using chained verifiable credentials; specifically, the MUD document is bound to the device ID using a simulated managed firmware service. This provides a verifiable credential binding a device identifier (IDevID) to a full MUD document.
S5.C6	Lifecycle Management Establishment	The device has a lifecycle management service and can automatically establish a secure association with it after performing network-layer onboarding and connecting to the network.	No	Not supported in this build.

Appendix A References

[1] *IEEE Standard for Local and Metropolitan Area Networks - Secure Device Identity*, IEEE Std 802.1AR-2018 (Revision of IEEE Std 802.1AR-2009), 2 Aug. 2018, 73 pp. Available: https://ieeexplore.ieee.org/document/8423794

[2] Wi-Fi Alliance, *Wi-Fi Easy Connect™ Specification Version 3.0*, 2022. Available: https://www.wi-fi.org/system/files/Wi-Fi_Easy_Connect_Specification_v3.0.pdf

[3] M. Pritikin, M. Richardson, T.T.E. Eckert, M.H. Behringer, and K.W. Watsen, *Bootstrapping Remote Secure Key Infrastructure (BRSKI)*, IETF Request for Comments (RFC) 8995, October 2021. Available: https://datatracker.ietf.org/doc/rfc8995/

[4] E. Lear, R. Droms, and D. Romascanu, *Manufacturer Usage Description Specification,* IETF Request for Comments (RFC) 8520, March 2019. Available: https://tools.ietf.org/html/rfc8520

[5] Open Connectivity Foundation (OCF) Iotivity: https://iotivity.org/

[6] Thread 1.1.1 Specification, February 13, 2017.

[7] O. Friel, E. Lear, M. Pritikin, and M. Richardson, *BRSKI over IEEE 802.11*, IETF Internet-Draft (Individual), July 2018. Available: https://datatracker.ietf.org/doc/draft-friel-brski-over-802dot11/01/

NIST SPECIAL PUBLICATION 1800-36E

Trusted Internet of Things (IoT) Device Network-Layer Onboarding and Lifecycle Management:

Enhancing Internet Protocol-Based IoT Device and Network Security

Volume E:

Risk and Compliance Management

Michael Fagan
Jeffrey Marron
Paul Watrobski
Murugiah Souppaya
National Cybersecurity Center of Excellence
Information Technology Laboratory

Susan Symington
The MITRE Corporation
McLean, Virginia

Dan Harkins
Aruba, a Hewlett Packard Enterprise Company
San Jose, California

Steve Clark
SEALSQ, a Subsidiary of WISeKey
Geneva, Switzerland

Andy Dolan
Kyle Haefner
Craig Platt
Darshak Thakore
CableLabs, Louisville, Colorado

Karen Scarfone
Scarfone Cybersecurity
Clifton, Virginia

William Barker
Dakota Consulting
Largo, Maryland

Nick Allott
Ashley Setter
NquiringMinds,
Southampton, United Kingdom

Brecht Wyseur
Kudelsky IoT
Cheseaux-sur-Lausanne, Switzerland

Mike Dow
Steve Egerter
Silicon Labs, Austin, Texas

Michael Richardson
Sandelman Software Works,
Ontario, Canada

May 2024

DRAFT

This publication is available free of charge from
https://www.nccoe.nist.gov/projects/trusted-iot-device-network-layer-onboarding-and-lifecycle-management

DISCLAIMER

Certain commercial entities, equipment, products, or materials may be identified by name or company logo or other insignia in order to acknowledge their participation in this collaboration or to describe an experimental procedure or concept adequately. Such identification is not intended to imply special status or relationship with NIST or recommendation or endorsement by NIST or NCCoE; neither is it intended to imply that the entities, equipment, products, or materials are necessarily the best available for the purpose.

National Institute of Standards and Technology Special Publication 1800-36E, Natl. Inst. Stand. Technol. Spec. Publ. 1800-36E, 22 pages, May 2024, CODEN: NSPUE2

FEEDBACK

You can improve this guide by contributing feedback on the mappings included in this volume. Do you find the mappings that we have provided in this document helpful to you as you try to achieve your cybersecurity goals? Could the mappings that we have provided be improved, either in terms of their content or format? Are there additional standards, best practices, or other guidance documents that you would like us to map to and from trusted IoT device network-layer onboarding and lifecycle management capabilities? Are there additional use cases for these mappings that we should consider in the future? As you review and adopt this solution for your own organization, we ask you and your colleagues to share your experience and advice with us.

Comments on this publication may be submitted to: iot-onboarding@nist.gov.

Public comment period: May 31, 2024 through July 30, 2024

All comments are subject to release under the Freedom of Information Act.

National Cybersecurity Center of Excellence
National Institute of Standards and Technology
100 Bureau Drive
Mailstop 2002
Gaithersburg, MD 20899
Email: nccoe@nist.gov

NATIONAL CYBERSECURITY CENTER OF EXCELLENCE

The National Cybersecurity Center of Excellence (NCCoE), a part of the National Institute of Standards and Technology (NIST), is a collaborative hub where industry organizations, government agencies, and academic institutions work together to address businesses’ most pressing cybersecurity issues. This public-private partnership enables the creation of practical cybersecurity solutions for specific industries, as well as for broad, cross-sector technology challenges. Through consortia under Cooperative Research and Development Agreements (CRADAs), including technology partners—from Fortune 50 market leaders to smaller companies specializing in information technology security—the NCCoE applies standards and best practices to develop modular, adaptable example cybersecurity solutions using commercially available technology. The NCCoE documents these example solutions in the NIST Special Publication 1800 series, which maps capabilities to the NIST Cybersecurity Framework and details the steps needed for another entity to re-create the example solution. The NCCoE was established in 2012 by NIST in partnership with the State of Maryland and Montgomery County, Maryland.

To learn more about the NCCoE, visit https://www.nccoe.nist.gov/. To learn more about NIST, visit https://www.nist.gov.

NIST CYBERSECURITY PRACTICE GUIDES

NIST Cybersecurity Practice Guides (Special Publication 1800 series) target specific cybersecurity challenges in the public and private sectors. They are practical, user-friendly guides that facilitate the adoption of standards-based approaches to cybersecurity. They show members of the information security community how to implement example solutions that help them align with relevant standards and best practices, and provide users with the materials lists, configuration files, and other information they need to implement a similar approach.

The documents in this series describe example implementations of cybersecurity practices that businesses and other organizations may voluntarily adopt. These documents do not describe regulations or mandatory practices, nor do they carry statutory authority.

KEYWORDS

application-layer onboarding; bootstrapping; Internet of Things (IoT); Manufacturer Usage Description (MUD); network-layer onboarding; onboarding; Wi-Fi Easy Connect.

ACKNOWLEDGMENTS

We are grateful to the following individuals for their generous contributions of expertise and time.

Name	Organization
Amogh Guruprasad Deshmukh	Aruba, a Hewlett Packard Enterprise company
Danny Jump	Aruba, a Hewlett Packard Enterprise company

Name	Organization
Bart Brinkman	Cisco
Eliot Lear	Cisco
Peter Romness	Cisco
Tyler Baker	Foundries.io
George Grey	Foundries.io
David Griego	Foundries.io
Fabien Gremaud	Kudelski IoT
Faith Ryan	The MITRE Corporation
Toby Ealden	NquiringMinds
John Manslow	NquiringMinds
Antony McCaigue	NquiringMinds
Alexandru Mereacre	NquiringMinds
Loic Cavaille	NXP Semiconductors
Mihai Chelalau	NXP Semiconductors
Julien Delplancke	NXP Semiconductors
Anda-Alexandra Dorneanu	NXP Semiconductors
Todd Nuzum	NXP Semiconductors
Nicusor Penisoara	NXP Semiconductors
Laurentiu Tudor	NXP Semiconductors
Pedro Fuentes	SEALSQ, a subsidiary of WISeKey

Name	Organization
Gweltas Radenac	SEALSQ, a subsidiary of WISeKey
Kalvin Yang	SEALSQ, a subsidiary of WISeKey

The Technology Partners/Collaborators who participated in this build submitted their capabilities in response to a notice in the Federal Register. Respondents with relevant capabilities or product components were invited to sign a Cooperative Research and Development Agreement (CRADA) with NIST, allowing them to participate in a consortium to build this example solution. We worked with:

Technology Collaborators

Aruba, a Hewlett Packard Enterprise company
CableLabs
Cisco
Foundries.io
Kudelski IoT
NquiringMinds
NXP Semiconductors
Open Connectivity Foundation (OCF)
Sandelman Software Works
SEALSQ, a subsidiary of WISeKey
Silicon Labs

DOCUMENT CONVENTIONS

The terms "shall" and "shall not" indicate requirements to be followed strictly to conform to the publication and from which no deviation is permitted. The terms "should" and "should not" indicate that among several possibilities, one is recommended as particularly suitable without mentioning or excluding others, or that a certain course of action is preferred but not necessarily required, or that (in the negative form) a certain possibility or course of action is discouraged but not prohibited. The terms "may" and "need not" indicate a course of action permissible within the limits of the publication. The terms "can" and "cannot" indicate a possibility and capability, whether material, physical, or causal.

CALL FOR PATENT CLAIMS

This public review includes a call for information on essential patent claims (claims whose use would be required for compliance with the guidance or requirements in this Information Technology Laboratory (ITL) draft publication). Such guidance and/or requirements may be directly stated in this ITL Publication or by reference to another publication. This call also includes disclosure, where known, of the existence of pending U.S. or foreign patent applications relating to this ITL draft publication and of any relevant unexpired U.S. or foreign patents.

ITL may require from the patent holder, or a party authorized to make assurances on its behalf, in written or electronic form, either:

a) assurance in the form of a general disclaimer to the effect that such party does not hold and does not currently intend holding any essential patent claim(s); or

b) assurance that a license to such essential patent claim(s) will be made available to applicants desiring to utilize the license for the purpose of complying with the guidance or requirements in this ITL draft publication either:

1. under reasonable terms and conditions that are demonstrably free of any unfair discrimination; or
2. without compensation and under reasonable terms and conditions that are demonstrably free of any unfair discrimination.

Such assurance shall indicate that the patent holder (or third party authorized to make assurances on its behalf) will include in any documents transferring ownership of patents subject to the assurance, provisions sufficient to ensure that the commitments in the assurance are binding on the transferee, and that the transferee will similarly include appropriate provisions in the event of future transfers with the goal of binding each successor-in-interest.

The assurance shall also indicate that it is intended to be binding on successors-in-interest regardless of whether such provisions are included in the relevant transfer documents.

Such statements should be addressed to: iot-onboarding@nist.gov.

Contents

1 Introduction 1
1.1 How to Use This Guide 1
2 Risks Addressed by Trusted Network-Layer Onboarding and Lifecycle Management 3
2.1 Risks to the Network 3
2.1.1 Risks to the Network Due to Device Limitations 3
2.1.2 Risks to the Network Due to Use of Shared Network Credentials 3
2.1.3 Risks to the Network Due to Insecure Network Credential Provisioning 4
2.1.4 Risks to the Network Due to Supply Chain Attacks 4
2.2 Risks to the Device 4
2.3 Risks to Secure Lifecycle Management 4
2.4 Limitations and Dependencies of Trusted Onboarding 5
3 Mapping Use Cases, Approach, and Terminology 6
3.1 Use Cases 6
3.2 Mapping Producers 7
3.3 Mapping Approach 7
3.3.1 Mapping Terminology 8
3.3.2 Mapping Process 8
4 Mappings 9
4.1 NIST CSF Subcategory Mappings 10
4.1.1 Mappings Between Reference Design Functions and NIST CSF Subcategories 10
4.1.2 Mappings Between Specific Onboarding Protocols and NIST CSF Subcategories 10
4.1.3 Mappings Between Specific Builds and NIST CSF Subcategories 10
4.2 NIST SP 800-53 Control Mappings 12
4.2.1 Mappings Between Reference Design Functions and NIST SP 800-53 Controls 12
4.2.2 Mappings Between Specific Onboarding Protocols and NIST SP 800-53 Controls 12
4.2.3 Mappings Between Specific Builds and NIST SP 800-53 Controls 13
Appendix A References 15

1 Introduction

In this project, the National Cybersecurity Center of Excellence (NCCoE) applies standards, recommended practices, and commercially available technology to demonstrate various mechanisms for trusted network-layer onboarding of IoT devices and lifecycle management of those devices. We show how to provision network credentials to IoT devices in a trusted manner and maintain a secure posture throughout the device lifecycle.

This volume of the NIST Cybersecurity Practice Guide discusses risks addressed by the trusted IoT device network-layer onboarding and lifecycle management reference design. It also maps between cybersecurity functionality provided by logical components of the reference design and Subcategories in the NIST Cybersecurity Framework (CSF) and controls in NIST Special Publication (SP) 800-53, *Security and Privacy Controls for Information Systems and Organizations*. (Note: The reference design is described in detail in NIST SP 1800-36B, Section 4.)

Mappings are also provided between cybersecurity functionality provided by specific network-layer onboarding protocols (e.g., Wi-Fi Easy Connect and Bootstrapping Remote Secure Key Infrastructure [BRSKI]) and those same Subcategories and controls, as well as between cybersecurity functionality provided by builds of the reference design that have been implemented as part of this project and those same Subcategories and controls. (Note: the composition of the builds is described in detail in the appendices of NIST SP 1800-36B.)

None of the mappings we provide is intended to be exhaustive; the mappings focus on the strongest relationships involving each reference design cybersecurity function in order to help organizations prioritize their work. The mappings help users understand how trusted IoT device network-layer onboarding and lifecycle management can help them achieve their cybersecurity goals in terms of CSF Subcategories and SP 800-53 controls. The mappings also help users understand how they can implement trusted onboarding and lifecycle management by identifying how trusted onboarding functionality is supported by the user’s existing implementations of CSF Subcategories and SP 800-53 controls.

1.1 How to Use This Guide

This NIST Cybersecurity Practice Guide demonstrates a standards-based reference design for implementing trusted IoT device network-layer onboarding and lifecycle management and describes various example implementations of this reference design. Each of these implementations, which are known as *builds,* is standards-based and is designed to help provide assurance that networks are not put at risk as new IoT devices are added to them and help safeguard IoT devices from being taken over by unauthorized networks. The reference design described in this practice guide is modular and can be deployed in whole or in part, enabling organizations to incorporate trusted IoT device network-layer onboarding and lifecycle management into their legacy environments according to goals that they have prioritized based on risk, cost, and resources.

NIST is adopting an agile process to publish this content. Each volume is being made available as soon as possible rather than delaying release until all volumes are completed.

This guide contains five volumes:

- NIST SP 1800-36A: *Executive Summary* – why we wrote this guide, the challenge we address, why it could be important to your organization, and our approach to solving this challenge
- NIST SP 1800-36B*: Approach, Architecture, and Security Characteristics* – what we built and why
- NIST SP 1800-36C: *How-To Guides* – instructions for building the example implementations, including all the security-relevant details that would allow you to replicate all or parts of this project
- NIST SP 1800-36D: *Functional Demonstrations* – use cases that have been defined to showcase trusted IoT device network-layer onboarding and lifecycle management security capabilities, and the results of demonstrating these use cases with each of the example implementations
- NIST SP 1800-36E*: Risk and Compliance Management* – risk analysis and mapping of trusted IoT device network-layer onboarding and lifecycle management security characteristics to cybersecurity standards and best practices **(you are here)**

Depending on your role in your organization, you might use this guide in different ways:

Business decision makers, including chief security and technology officers, will be interested in the *Executive Summary, NIST SP 1800-36A*, which describes the following topics:

- challenges that enterprises face in migrating to the use of trusted IoT device network-layer onboarding
- example solutions built at the NCCoE
- benefits of adopting the example solution

Technology or security program managers who are concerned with how to identify, understand, assess, and mitigate risk will be interested in *NIST SP 1800-36B*, which describes what we did and why.

Also, Section 4 of *NIST SP 1800-36E* will be of particular interest. Section 4, *Mappings*, maps logical components of the general trusted IoT device network-layer onboarding and lifecycle management reference design to security characteristics listed in various cybersecurity standards and recommended practices documents, including *Framework for Improving Critical Infrastructure Cybersecurity* (NIST Cybersecurity Framework) and *Security and Privacy Controls for Information Systems and Organizations* (NIST SP 800-53).

You might share the *Executive Summary, NIST SP 1800-36A*, with your leadership team members to help them understand the importance of using standards-based trusted IoT device network-layer onboarding and lifecycle management implementations.

IT professionals who want to implement similar solutions will find the whole practice guide useful. You can use the how-to portion of the guide, *NIST SP 1800-36C*, to replicate all or parts of the builds created in our lab. The how-to portion of the guide provides specific product installation, configuration, and integration instructions for implementing the example solution. We do not re-create the product manufacturers' documentation, which is generally widely available. Rather, we show how we incorporated the products together in our environment to create an example solution. Also, you can use *Functional Demonstrations, NIST SP 1800-36D*, which provides the use cases that have been defined to showcase trusted IoT device network-layer onboarding and lifecycle management security capabilities and the results of demonstrating these use cases with each of the example implementations. Finally, *NIST SP 1800-36E* will be helpful in explaining the security functionality that the components of each build provide.

This guide assumes that IT professionals have experience implementing security products within the enterprise. While we have used a suite of commercial products to address this challenge, this guide does not endorse these particular products. Your organization can adopt this solution or one that adheres to these guidelines in whole, or you can use this guide as a starting point for tailoring and implementing parts of a trusted IoT device network-layer onboarding and lifecycle management solution. Your organization's security experts should identify the products that will best integrate with your existing tools and IT system infrastructure. We hope that you will seek products that are congruent with applicable standards and recommended practices.

A NIST Cybersecurity Practice Guide does not describe "the" solution, but example solutions. We seek feedback on the publication's contents and welcome your input. Comments, suggestions, and success stories will improve subsequent versions of this guide. Please contribute your thoughts to iot-onboarding@nist.gov.

2 Risks Addressed by Trusted Network-Layer Onboarding and Lifecycle Management

Historically, IoT devices have not tended to be onboarded to networks in a trusted manner. This has left networks open to the threat of having unauthorized devices connect to them. It has also left devices open to the threat of being onboarded to networks that are not authorized to control them.

2.1 Risks to the Network

Unauthorized devices that are able to connect to a network pose many risks to that network. They may be able to send and receive data on that network, scan the network for vulnerabilities, eavesdrop on the communications of other devices, and attack other connected devices to exfiltrate or modify their data or to compromise those devices and co-opt them into service to launch distributed denial of service (DDoS) attacks.

2.1.1 Risks to the Network Due to Device Limitations

Many IoT devices are manufactured to be as inexpensive as possible, which sometimes means that the devices are not equipped with secure storage, cryptographic modules, unique authoritative birth credentials, or other features needed to enable the devices to be identified and authenticated. This can make it impossible for a network to determine if a device attempting to connect to it is the intended device. Lack of these features can also make it impossible to protect the confidentiality of a device's network credentials, both during the provisioning process and after the credentials have been installed on the device.

2.1.2 Risks to the Network Due to Use of Shared Network Credentials

If a network uses a single network password that is shared among all devices rather than providing each device with a unique network credential, the network will be vulnerable to having unauthorized devices connect to it if the shared network password falls into the wrong hands, which can happen relatively easily. It also means that the network will permit devices to connect to it simply because a device presents the correct shared password, regardless of the device's type or identity, or whether it has any legitimate reason to connect to the network.

2.1.3 Risks to the Network Due to Insecure Network Credential Provisioning

If devices are manually provisioned with their network credentials, the provisioning process is error-prone, cumbersome, and vulnerable to having the device's network credentials disclosed. If the devices are provisioned automatically over Wi-Fi or some other interface that does not use an encrypted channel, the credentials are also vulnerable to unauthorized disclosure. If the network credentials are not provisioned in a trusted manner, the credentials are vulnerable to disclosure not only the first time the device is onboarded to the network, but every time it is onboarded, which may occur many times during the device lifecycle. For example, the device may need to be re-onboarded periodically to change its credentials in accordance with security policy, or it may need to be re-onboarded due to a security breach, hardware repair, security update, or other reasons. Any insecure features of the onboarding process, therefore, will render the device and network vulnerable every time the device is onboarded.

2.1.4 Risks to the Network Due to Supply Chain Attacks

If a device is compromised while in the supply chain or at some other point prior to being onboarded, then even though the device may be onboarded in a trusted manner, it may still pose a threat to the network, its data, and all devices connected to it. If, on the other hand, the trusted network-layer onboarding mechanism is integrated with a device attestation or supply chain management service that is capable of evaluating the integrity and provenance of the device and detecting that it has been compromised or may have been tampered with, the trusted network-layer onboarding mechanism could prevent such a compromised device from being onboarded and connected to the network.

2.2 Risks to the Device

Although it is relatively easy for one network to masquerade as another, IoT devices often do not authenticate the identity of the networks to which they allow themselves to be onboarded and connected. Devices may be unwittingly tricked into onboarding and connecting to imposter networks that are not authorized to onboard them. This makes those devices vulnerable to being taken control of by those unauthorized networks and thereby prevented from connecting to and providing their intended function on their authorized network.

2.3 Risks to Secure Lifecycle Management

Even if a device is authorized to connect to a network and the network is authorized to control the device, if the device has not been onboarded in a trusted manner, then other security-related operations that are performed after the device has connected to the network may not have as secure a foundation as they would if the device had been onboarded in a trusted manner. For example, if device communications intent enforcement is performed but the integrity and confidentiality of the communicated device intent information was not protected (as it would be by a trusted network-layer onboarding mechanism), then trust in the device communications intent enforcement mechanism may not be as robust as it could have been. Similarly, if application-layer onboarding is performed after the device connects, but the information needed to bootstrap the application-layer onboarding process did not have its integrity and confidentiality protected (as it would be by a trusted network-layer onboarding mechanism), then trust in the application-layer onboarding mechanism may not be as robust as it could have been. Lack of trust in the application-layer onboarding mechanism may, in turn, undermine trust in the device lifecycle management or other application-layer service that is invoked as part of the application-layer onboarding process.

2.4 Limitations and Dependencies of Trusted Onboarding

While implementing trusted IoT device network-layer onboarding and lifecycle management addresses many risks, it also has limitations. Use of trusted network-layer onboarding is designed to enable IoT devices to be provisioned with unique local network credentials in a manner that preserves credential confidentiality. As part of the trusted network-layer onboarding process, the device and the network may mutually authenticate one another, thereby protecting the network from having unauthorized devices connect to it and the device from being taken over by an unauthorized network. However, if the network also enables devices that do not support the trusted network-layer onboarding solution to be provisioned with network credentials and connect to it using a different (untrusted) onboarding solution, the network and all devices on it will still be at risk from IoT devices that have been onboarded using untrusted mechanisms, and the devices that are onboarded using untrusted mechanisms will still be at risk of being taken over by networks that are not authorized to control them.

The trusted network-layer onboarding solution leverages the device's unique, authoritative *birth credentials*, which are provisioned to the device by the device manufacturer and must consist, at a minimum, of a unique device identity and a secret. The trustworthiness of the network-layer onboarding process and the network credentials that it provisions to the device depends on the uniqueness, integrity, and confidentiality of the device's birth credentials which, in many cases, depend on the device's hardware root of trust. If the manufacturer does not ensure that the device's credentials are unique, the identity of the device cannot be definitively authenticated. If the manufacturer is not able to maintain the confidentiality of the secret that is part of the device credentials, the trustworthiness of the device authentication process will be undermined, and the channel over which the device's credentials are provisioned will be vulnerable to eavesdropping.

The trusted network-layer onboarding solution depends upon the trustworthiness of the device's secure storage to ensure the confidentiality of the device and network credentials. If the device's secure storage is vulnerable, the trustworthiness of the network-layer onboarding process and the confidentiality of the device's network credentials will be compromised. If the secure storage in which the device's network credentials are stored is vulnerable, the network will be at risk of having unauthorized devices attach to it.

If the trusted network-layer onboarding mechanism is integrated with additional security capabilities such as device attestation, device communications intent enforcement, application-layer onboarding, and device lifecycle management, it can further increase trust in both the IoT device and, by extension, the network to which the device connects, assuming that these additional security capabilities themselves are secure and robust. If these security capabilities are not implemented correctly, then integrating with them is of no additional value and in fact may provide a false sense of security.

3 Mapping Use Cases, Approach, and Terminology

A *mapping* indicates that one concept is related to another concept. The remainder of this volume describes the mappings between trusted IoT device network-layer onboarding and lifecycle management cybersecurity functions and the security characteristics enumerated in relevant cybersecurity documents.

For this mapping, we have used the supportive relationship mapping style as defined in Section 4.2 of draft NIST Internal Report (IR) 8477, *Mapping Relationships Between Documentary Standards, Regulations, Frameworks, and Guidelines: Developing Cybersecurity and Privacy Concept Mappings* [1].

Each set of mappings involves one of the following types of trusted IoT device network-layer onboarding and lifecycle management cybersecurity functions:

- Cybersecurity functions performed by the reference design's logical components (see NIST SP 1800-36B Section 4)
- Cybersecurity functions provided by specific network-layer onboarding protocols (e.g., Wi-Fi Easy Connect and BRSKI)
- Cybersecurity functions provided by builds of the reference design that have been implemented as part of this project

Each of the cybersecurity functions is mapped to the security characteristics concepts found in the following widely used cybersecurity guidance documents:

- Subcategories from the NIST Cybersecurity Framework (CSF) 1.1 [2] which are also mapped to *The NIST Cybersecurity Framework 2.0 (CSF 2.0)* [3]. The CSF identifies enterprise-level security outcomes. Stakeholders have identified these outcomes as helpful for managing cybersecurity risk, but organizations adopting the CSF need to determine how to achieve the outcomes. Executive Order (EO) 13800, *Strengthening the Cybersecurity of Federal Networks and Critical Infrastructure* [4], made the CSF mandatory for federal government agencies, and other government agencies and sectors have also made the CSF mandatory.
- Security controls from NIST SP 800-53r5 (*Security and Privacy Controls for Information Systems and Organizations*) [5]. NIST SP 800-53 identifies security controls that apply to systems on which those enterprises are reliant. Which SP 800-53 controls need to be employed depends on system functions and a risk assessment of the perceived impact of loss of system functionality or exposure of information from the system to unauthorized entities. In the case of systems owned by or operated on behalf of federal government enterprises, the risk assessment and applicable SP 800-53 controls are mandated under the Federal Information Security Modernization Act (FISMA) [6]. Many other governments and private sector organizations voluntarily employ the Risk Management Framework [7] and associated SP 800-53 controls.

3.1 Use Cases

All of the elements in these mappings—the trusted IoT device network-layer onboarding and lifecycle management cybersecurity functions, cybersecurity functions provided by specific network-layer onboarding protocols, cybersecurity functions provided by specific builds, CSF Subcategories, and SP 800-53 controls—are concepts involving ways to reduce cybersecurity risk.

There are two primary use cases for this mapping. They are not intended to be comprehensive, but rather to capture the strongest relationships involving the trusted IoT device network-layer onboarding and lifecycle management cybersecurity functions.

1. **Why should organizations implement trusted IoT device network-layer onboarding and lifecycle management?** This use case identifies how implementing trusted IoT device network-layer onboarding and lifecycle management can support organizations with achieving CSF Subcategories and SP 800-53 controls. This helps communicate to an organization's chief information security officer, security team, and senior management that expending resources to implement trusted IoT device network-layer onboarding and lifecycle management can also aid in fulfilling other security requirements.

2. **How can organizations implement trusted IoT device network-layer onboarding and lifecycle management?** This use case identifies how an organization's existing implementations of CSF Subcategories and SP 800-53 controls can help support a trusted IoT device network-layer onboarding and lifecycle management implementation. An organization wanting to implement trusted IoT device network-layer onboarding and lifecycle management might first assess its current security capabilities so that it can plan how to add missing capabilities and enhance existing capabilities. Organizations can leverage their existing security investments and prioritize future security technology deployment to address the gaps.

These mappings are intended to be used by any organization that is interested in implementing trusted IoT device network-layer onboarding and lifecycle management or that has begun or completed an implementation.

3.2 Mapping Producers

The NCCoE trusted IoT device network-layer onboarding and lifecycle management project team performed the mappings between the cybersecurity functions performed by the reference design's logical components (see NIST SP 1800 36B Section 4) and the security characteristics in the cybersecurity documents. They also performed the mappings between the cybersecurity functions performed by the specific network-layer onboarding protocols (i.e., Wi-Fi Easy Connect and BRSKI) and the security characteristics in the cybersecurity documents. These mappings were performed with input and feedback from the collaborators who have contributed technology to the builds of the reference design. Collaborators for each build, in conjunction with the NCCoE trusted IoT device network-layer onboarding and lifecycle management project team, performed the mappings between the cybersecurity functions provided by their contributed technologies in each build and the security characteristics in the cybersecurity documents.

3.3 Mapping Approach

In addition to performing general mappings between the reference design's cybersecurity functions and various sets of security characteristics, as well as between specific network-layer onboarding protocol cybersecurity functions and various sets of security characteristics, the NCCoE asked the collaborators for each build to indicate the mapping between the cybersecurity functions their technology components provide in that build and the sets of security characteristics.

Using the logical components in the reference design as the organizing principle for the initial mapping of cybersecurity functions to security characteristics and then providing onboarding protocol-specific mappings was intended to make it easier for collaborators to map their build-specific technology contributions. Using this approach, the build-specific technology mappings are instantiations of the project's general reference design and protocol-specific mappings for each document.

3.3.1 Mapping Terminology

In this publication, we use the following relationship types from NIST IR 8477 [1] to describe how the functions in our reference design are related to the NIST reference documents. Note that the *Supports* relationship applies only to use case 1 in Section 3.1 and the *Is Supported By* relationship applies only to use case 2.

- **Supports:** Trusted IoT device network-layer onboarding and lifecycle management function X *supports* security control/Subcategory/capability/requirement Y when X can be applied alone or in combination with one or more other functions to achieve Y in whole or in part.
- **Is Supported By:** Trusted IoT device network-layer onboarding and lifecycle management function X is *supported by* security control/Subcategory/capability/requirement Y when Y can be applied alone or in combination with one or more other security controls/Subcategories/capabilities/requirements to achieve X in whole or in part.

Each *Supports* and *Is Supported By* relationship has one of the following properties assigned to it:

- **Example of:** The supporting concept X is one way (*an example*) of achieving the supported concept Y in whole or in part. However, Y could also be achieved without applying X.
- **Integral to:** The supporting concept X is *integral to* and a component of the supported concept Y. X must be applied as part of achieving Y.
- **Precedes:** The supporting concept X *precedes* the supported concept Y when X must be achieved before applying Y. In other words, X is a prerequisite for Y.

When determining whether a reference design function's support for a given CSF Subcategory or SP 800-53 control is integral to that support versus an example of that support, we do not consider how that function may in general be used to support the Subcategory, control, capability, or requirement. Rather, we consider only how that function is intended to support that Subcategory, control, capability, or requirement within the context of our reference design.

Also, when determining whether a function is supported by a CSF Subcategory, SP 800-53 control, capability, etc. with the relationship property of *precedes*, we do not consider whether it is possible to apply the function without first achieving the Subcategory, control, capability, or requirement. Rather, we consider whether, according to our reference design, the Subcategory, control, capability, or requirement is to be achieved prior to applying that function.

3.3.2 Mapping Process

The process that the NCCoE used to create the mapping from the logical components of the reference design to the security characteristics of a given document was as follows:

1. Create a table that lists each of the logical components of the reference design in column 1.

2. Describe each logical component's cybersecurity function in column 2.

3. Map each cybersecurity function to each of the security characteristics in the document to which the function is most strongly related, and list each of these security characteristics on different sub-rows within column 3. Begin each security characteristic entry with an underlined keyword that describes the mapping's relationship type (i.e., *Supports*, *Is Supported By*). After the keyword indicating the relationship type, put in parentheses the underlined keyword describing the relationship's property (i.e., *Example of*, *Integral to*, or *Precedes*).

4. In the fourth column, provide a brief explanation of why that relationship type and property apply to the mapping.

5. After completing the mapping table entries as described above for all the logical components in the reference design, examine the mapping in the other direction, i.e., starting with the security characteristics listed in the document and considering whether they have a relationship to the logical components' cybersecurity functions in the reference design. In other words, step through each of the security characteristics in the document and determine if there is some logical component in the reference design that has a strong relationship to that security characteristic. If so, add an entry for that security characteristic mapping to that logical component's row in the table. By examining the mapping in both directions in this manner, security characteristic mappings are less likely to be overlooked or omitted.

6. Once these steps are complete, any rows in the table that don't have any mappings should be deleted.

The NCCoE applied this mapping process separately for each reference document. None of the mappings is intended to be exhaustive; they all focus on the strongest relationships involving each cybersecurity function in order to help organizations prioritize their work. Mapping every possible relationship, no matter how tenuous, would create so many mappings that they would not have any value in prioritization.

4 Mappings

The mappings are provided in the form of Excel files. Links to the mapping Excel files are organized in the remainder of this document as follows:

- Section 4.1 – NIST CSF 1.1 [2] and NIST CSF 2.0 [3] mappings. These include:
  - Section 4.1.1 – Mappings between reference design functions and NIST CSF Subcategories
  - Section 4.1.2 – Mappings between specific onboarding protocol (i.e., Wi-Fi Easy Connect and BRSKI) functions and NIST CSF Subcategories
  - Section 4.1.3 – Mappings between specific build functions and NIST CSF Subcategories
- Section 4.2 – NIST SP 800-53r5 [5] mappings. These include:
  - Section 4.2.1 – Mappings between reference design functions and NIST SP 800-53r5 controls

- Section 4.2.2 – Mappings between specific onboarding protocol (i.e., Wi-Fi Easy Connect and BRSKI) functions and NIST SP 800-53r5 controls
- Section 4.2.3 – Mappings between specific build functions and NIST SP 800-53r5 controls

4.1 NIST CSF Subcategory Mappings

This section provides links to mappings between various elements that provide trusted network-layer onboarding functionality and NIST CSF Subcategories.

4.1.1 Mappings Between Reference Design Functions and NIST CSF Subcategories

This Excel file provides mappings between the logical components of the reference design and the NIST CSF Subcategories. These mappings indicate how trusted IoT device network-layer onboarding and lifecycle management functions help support CSF Subcategories and vice versa.

Link to the Excel file called "IoT Volume E CSF 1-1 and 2-0", and to the tab called "CSF-to-Reference Arch" (first tab)

4.1.2 Mappings Between Specific Onboarding Protocols and NIST CSF Subcategories

This section provides mappings between the functionality provided by two network-layer onboarding protocols, Wi-Fi Easy Connect and BRSKI, and the NIST CSF Subcategories.

4.1.2.1 Mapping Between Wi-Fi Easy Connect and NIST CSF Subcategories

This Excel file provides a mapping between the functionality provided by the Wi-Fi Easy Connect protocol and the NIST CSF Subcategories. These mappings indicate how Wi-Fi Easy Connect functionality helps support CSF Subcategories and vice versa.

Link to the Excel file called "CSF 1.1 and 2.0 Tables", and to the tab called "CSF-to-Wi-Fi EasyCnct" (third tab)

4.1.2.2 Mapping Between BRSKI and NIST CSF Subcategories

This Excel file provides a mapping between the functionality provided by BRSKI and the NIST CSF Subcategories. These mappings indicate how BRSKI functionality helps support CSF Subcategories and vice versa.

Link to the Excel file called "CSF 1.1 and 2.0 Tables", and to the tab called "CSF-to-BRSKI" (second tab)

4.1.3 Mappings Between Specific Builds and NIST CSF Subcategories

This section provides mappings between the functionality provided by builds of the trusted IoT device network-layer onboarding and lifecycle management reference design that were implemented as part of this project and the NIST CSF Subcategories.

4.1.3.1 Mapping Between Build 1 and NIST CSF Subcategories

Build 1 is an implementation of network-layer onboarding that uses the Wi-Fi Easy Connect protocol. The onboarding infrastructure and related technology components for Build 1 have been provided by Aruba/HPE. IoT devices that were onboarded using Build 1 were provided by Aruba/HPE and CableLabs. The technologies used in Build 1 are detailed in Appendix C of SP 1800-36B.

This Excel file details the mapping between the functionality provided by Build 1 components and CSF Subcategories. These mappings indicate how these components help support CSF Subcategories and vice versa.

Link to the Excel file called "CSF 1.1 and 2.0 Tables", and to the tab called "CSF-to-B1" (fourth tab)

4.1.3.2 Mapping Between Build 2 and NIST CSF Subcategories

Build 2 is an implementation of network-layer onboarding that uses the Wi-Fi Easy Connect protocol. The onboarding infrastructure and related technology components for Build 2 have been provided by CableLabs and OCF. IoT devices that were onboarded using Build 2 were provided by CableLabs, OCF, and Aruba/HPE. The technologies used in Build 2 are detailed in Appendix D of SP 1800-36B.

This Excel file details the mapping between the functionality provided by Build 2 components and CSF Subcategories. These mappings indicate how these components help support CSF Subcategories and vice versa.

Link to the Excel file called "CSF 1.1 and 2.0 Tables", and to the tab called "CSF-to-B2" (fifth tab)

4.1.3.3 Mapping Between Build 3 and NIST CSF Subcategories

Build 3 is an implementation of network-layer onboarding that uses BRSKI. The onboarding infrastructure and related technology components for Build 3 have been provided by Sandelman Software Works. The IoT device that was used to demonstrate onboarding in Build 3 was a pledge simulator provided by Sandelman. The technologies used in Build 3 are detailed in Appendix E of SP 1800-36B.

This Excel file details the mapping between the functionality provided by Build 3 components and CSF Subcategories. These mappings indicate how these components help support CSF Subcategories and vice versa.

Link to the Excel file called "CSF 1.1 and 2.0 Tables", and to the tab called "CSF-to-B3" (sixth tab)

4.1.3.4 Mapping Between Build 4 and NIST CSF Subcategories

Build 4 is an implementation of network-layer connection to an OpenThread network using pre-provisioned network credentials as well as independent application-layer onboarding using the Kudelski KeySTREAM service. The network infrastructure and related technology components for Build 4 have been provided by Silicon Labs and Kudelski. The IoT device that was used to demonstrate onboarding in Build 4 was provided by Silicon Labs. The technologies used in Build 4 are detailed in Appendix F of SP 1800-36B.

This Excel file details the mapping between the functionality provided by Build 4 components and CSF Subcategories These mappings indicate how these components help support CSF Subcategories and vice versa.

Link to the Excel file called "CSF 1.1 and 2.0 Tables", and to the tab called "CSF-to-B4" (seventh tab)

4.1.3.5 Mapping Between Build 5 and NIST CSF Subcategories

Build 5 is an implementation of network-layer onboarding using BRSKI over Wi-Fi, as well as demonstration of a continuous authorization service. The network layer onboarding infrastructure and related technology components for Build 5 have been provided by NquiringMinds. The IoT devices that were used to demonstrate onboarding in Build 5 were provided by NquiringMinds. The technologies used in Build 5 are detailed in Appendix G of SP 1800-36B.

This Excel file details the mapping between the functionality provided by Build 5 components and CSF Subcategories. These mappings indicate how these components help support CSF Subcategories and vice versa.

Link to the Excel file called "CSF 1.1 and 2.0 Tables", and to the tab called "CSF-to-B5" (eighth tab)

4.2 NIST SP 800-53 Control Mappings

This section provides mappings between various elements that provide trusted network-layer onboarding functionality and NIST SP 800-53 controls.

4.2.1 Mappings Between Reference Design Functions and NIST SP 800-53 Controls

This Excel file provides a mapping between the logical components of the reference design and NIST SP 800-53 security controls. These mappings indicate how trusted IoT device network-layer onboarding and lifecycle management functions help support NIST SP 800-53 controls. Because hundreds of NIST SP 800-53 controls can help support these functions, we have limited use case 2 (see Section 3.1) mappings to those controls on which specified supporting controls directly depend (e.g., dependence of cryptographic protection on key management). Readers needing to determine how their trusted IoT device network-layer onboarding and lifecycle management implementations support RMF processes can refer to these mappings.

Link to the Excel file called "800-53 Tables", and to the tab called "800-53-to-Reference Arch" (first tab)

4.2.2 Mappings Between Specific Onboarding Protocols and NIST SP 800-53 Controls

This section provides mappings between the functionality provided by specific network-layer onboarding protocols and the NIST SP 800-53 controls. Mappings are provided for both the Wi-Fi Easy Connect protocol and BRSKI.

4.2.2.1 Mapping Between Wi-Fi Easy Connect and NIST SP 800-53 Controls

This Excel file provides a mapping between the functionality provided by the Wi-Fi Easy Connect protocol and the NIST SP 800-53 controls. These mappings indicate how Wi-Fi Easy Connect functions help support NIST SP 800-53 controls and vice versa.

Link to the Excel file called "800-53 Tables", and to the tab called "800-53-to-Wi-Fi EasyCnct" (second tab)

4.2.2.2 Mapping Between BRSKI and NIST SP 800-53 Controls

This Excel file provides a mapping between the functionality provided by BRSKI and the NIST SP 800-53 controls. These mappings indicate how BRSKI functions help support NIST SP 800-53 controls and vice versa.

Link to the Excel file called "800-53 Tables", and to the tab called "800-53-to-BRSKI" (third tab)

4.2.3 Mappings Between Specific Builds and NIST SP 800-53 Controls

This section provides mappings between the functionality provided by builds of the trusted IoT device network-layer onboarding and lifecycle management reference design that were implemented as part of this project and the NIST SP 800-53 controls.

4.2.3.1 Mapping Between Build 1 and NIST SP 800-53 Controls

Build 1 is an implementation of network-layer onboarding that uses the Wi-Fi Easy Connect protocol. The onboarding infrastructure and related technology components for Build 1 have been provided by Aruba/HPE. IoT devices that were onboarded using Build 1 were provided by Aruba/HPE and CableLabs. The technologies used in Build 1 are detailed in Appendix C of SP 1800-36B.

This Excel file details the mapping between the functionality provided by Build 1 components and SP 800-53 controls. These mappings indicate how these components help support SP 800-53 controls and vice versa.

Link to the Excel file called "800-53 Tables", and to the tab called "800-53-to-B1" (fourth tab)

4.2.3.2 Mapping Between Build 2 and NIST SP 800-53 Controls

Build 2 is an implementation of network-layer onboarding that uses the Wi-Fi Easy Connect protocol. The onboarding infrastructure and related technology components for Build 2 have been provided by CableLabs and OCF. IoT devices that were onboarded using Build 2 were provided by CableLabs, OCF, and Aruba/HPE. The technologies used in Build 1 are detailed in Appendix D of SP 1800-36B.

This Excel file details the mapping between the functionality provided by Build 2 components and SP 800-53 controls These mappings indicate how these components help support SP 800-53 controls and vice versa.

Link to the Excel file called "800-53 Tables", and to the tab called "800-53-to-B2" (fifth tab)

4.2.3.3 Mapping Between Build 3 and NIST SP 800-53 Controls

Build 3 is an implementation of network-layer onboarding that uses BRSKI. The onboarding infrastructure and related technology components for Build 3 have been provided by Sandelman Software Works. The IoT device that was used to demonstrate onboarding in Build 3 was a pledge simulator provided by Sandelman. The technologies used in Build 3 are detailed in Appendix E of SP 1800-36B.

This Excel file details the mapping between the functionality provided by Build 3 components and SP 800-53 controls. These mappings indicate how these components help support SP 800-53 controls and vice versa.

Link to the Excel file called "800-53 Tables", and to the tab called "800-53-to-B3" (sixth tab)

4.2.3.4 Mapping Between Build 4 and NIST SP 800-53 Controls

Build 4 is an implementation of network-layer connection to an OpenThread network using pre-provisioned network credentials as well as independent application-layer onboarding using the Kudelski KeySTREAM service. The network infrastructure and related technology components for Build 4 have been provided by Silicon Labs and Kudelski. The IoT device that was used to demonstrate onboarding in Build 4 was provided by Silicon Labs. The technologies used in Build 4 are detailed in Appendix F of SP 1800-36B.

This Excel file details the mapping between the functionality provided by Build 4 components and SP 800-53 controls. These mappings indicate how these components help support SP 800-53 controls and vice versa.

Link to the Excel file called "800-53 Tables", and to the tab called "800-53-to-B4" (seventh tab)

4.2.3.5 Mapping Between Build 5 and NIST SP 800-53 Controls

Build 5 is an implementation of network-layer onboarding using BRSKI over Wi-Fi, as well as demonstration of a continuous authorization service. The network layer onboarding infrastructure and related technology components for Build 5 have been provided by NquiringMinds. The IoT devices that were used to demonstrate onboarding in Build 5 were provided by NquiringMinds. The technologies used in Build 5 are detailed in Appendix G of SP 1800-36B.

This Excel file details the mapping between the functionality provided by Build 5 components and SP 800-53 controls. These mappings indicate how these components help support SP 800-53 controls and vice versa.

Link to the Excel file called "800-53 Tables", and to the tab called "800-53-to-B5" (eighth tab)

Appendix A References

[1] K. Scarfone, M. Souppaya, and M. Fagan, Mapping Relationships Between Documentary Standards, Regulations, Frameworks, and Guidelines: Developing Cybersecurity and Privacy Content Mappings, National Institute of Standards and Technology (NIST) Internal Report (IR) 8477, Gaithersburg, Md., August 2023, 26 pp. Available: https://doi.org/10.6028/NIST.IR.8477.ipd

[2] National Institute of Standards and Technology (2018) Framework for Improving Critical Infrastructure Cybersecurity, Version 1.1. (National Institute of Standards and Technology, Gaithersburg, MD), NIST Cybersecurity White Paper (CSWP) NIST CSWP 6. https://doi.org/10.6028/NIST.CSWP.6

[3] National Institute of Standards and Technology, Version 2.0. The NIST Cybersecurity Framework 2.0 (CSF 2.0) (National Institute of Standards and Technology, Gaithersburg, MD), https://csrc.nist.gov/pubs/cswp/29/the-nist-cybersecurity-framework-20/ipd

[4] Executive Order 13800 (2017) Strengthening the Cybersecurity of Federal Networks and Critical Infrastructure. (The White House, Washington, DC), DCPD-201700327, May 11, 2017. https://www.govinfo.gov/app/details/DCPD-201700327

[5] Joint Task Force (2020) Security and Privacy Controls for Information Systems and Organizations. (National Institute of Standards and Technology, Gaithersburg, MD), NIST Special Publication (SP) 800-53, Rev. 5. Includes updates as of December 10, 2020. https://doi.org/10.6028/NIST.SP.800-53r5

[6] S.2521 - Federal Information Security Modernization Act of 2014, 113th Congress (2013-2014), Became Public Law No: 113-283, December 18, 2014. Available: https://www.congress.gov/bill/113th-congress/senate-bill/2521

[7] Joint Task Force (2018) Risk Management Framework for Information Systems and Organizations: A System Life Cycle Approach for Security and Privacy. (National Institute of Standards and Technology, Gaithersburg, MD), NIST Special Publication (SP) 800-37, Rev. 2. https://doi.org/10.6028/NIST.SP.800-37r2

NIST SPECIAL PUBLICATION 1800-36E

Trusted Internet of Things (IoT) Device Network-Layer Onboarding and Lifecycle Management:

Enhancing Internet Protocol-Based IoT Device and Network Security

Volume E:

Risk and Compliance Management

Michael Fagan
Jeffrey Marron
Paul Watrobski
Murugiah Souppaya
National Cybersecurity Center of Excellence
Information Technology Laboratory

Susan Symington
The MITRE Corporation
McLean, Virginia

Dan Harkins
Aruba, a Hewlett Packard Enterprise Company
San Jose, California

Steve Clark
SEALSQ, a Subsidiary of WISeKey
Geneva, Switzerland

Andy Dolan
Kyle Haefner
Craig Platt
Darshak Thakore
CableLabs, Louisville, Colorado

Karen Scarfone
Scarfone Cybersecurity
Clifton, Virginia

William Barker
Dakota Consulting
Largo, Maryland

Nick Allott
Ashley Setter
NquiringMinds,
Southampton, United Kingdom

Brecht Wyseur
Kudelsky IoT
Cheseaux-sur-Lausanne, Switzerland

Mike Dow
Steve Egerter
Silicon Labs, Austin, Texas

Michael Richardson
Sandelman Software Works,
Ontario, Canada

May 2024

DRAFT

This publication is available free of charge from
https://www.nccoe.nist.gov/projects/trusted-iot-device-network-layer-onboarding-and-lifecycle-management

NIST | NATIONAL CYBERSECURITY CENTER OF EXCELLENCE

DISCLAIMER

Certain commercial entities, equipment, products, or materials may be identified by name or company logo or other insignia in order to acknowledge their participation in this collaboration or to describe an experimental procedure or concept adequately. Such identification is not intended to imply special status or relationship with NIST or recommendation or endorsement by NIST or NCCoE; neither is it intended to imply that the entities, equipment, products, or materials are necessarily the best available for the purpose.

National Institute of Standards and Technology Special Publication 1800-36E, Natl. Inst. Stand. Technol. Spec. Publ. 1800-36E, 22 pages, May 2024, CODEN: NSPUE2

FEEDBACK

You can improve this guide by contributing feedback on the mappings included in this volume. Do you find the mappings that we have provided in this document helpful to you as you try to achieve your cybersecurity goals? Could the mappings that we have provided be improved, either in terms of their content or format? Are there additional standards, best practices, or other guidance documents that you would like us to map to and from trusted IoT device network-layer onboarding and lifecycle management capabilities? Are there additional use cases for these mappings that we should consider in the future? As you review and adopt this solution for your own organization, we ask you and your colleagues to share your experience and advice with us.

Comments on this publication may be submitted to: iot-onboarding@nist.gov.

Public comment period: May 31, 2024 through July 30, 2024

All comments are subject to release under the Freedom of Information Act.

National Cybersecurity Center of Excellence
National Institute of Standards and Technology
100 Bureau Drive
Mailstop 2002
Gaithersburg, MD 20899
Email: nccoe@nist.gov

NATIONAL CYBERSECURITY CENTER OF EXCELLENCE

The National Cybersecurity Center of Excellence (NCCoE), a part of the National Institute of Standards and Technology (NIST), is a collaborative hub where industry organizations, government agencies, and academic institutions work together to address businesses' most pressing cybersecurity issues. This public-private partnership enables the creation of practical cybersecurity solutions for specific industries, as well as for broad, cross-sector technology challenges. Through consortia under Cooperative Research and Development Agreements (CRADAs), including technology partners—from Fortune 50 market leaders to smaller companies specializing in information technology security—the NCCoE applies standards and best practices to develop modular, adaptable example cybersecurity solutions using commercially available technology. The NCCoE documents these example solutions in the NIST Special Publication 1800 series, which maps capabilities to the NIST Cybersecurity Framework and details the steps needed for another entity to re-create the example solution. The NCCoE was established in 2012 by NIST in partnership with the State of Maryland and Montgomery County, Maryland.

To learn more about the NCCoE, visit https://www.nccoe.nist.gov/. To learn more about NIST, visit https://www.nist.gov.

NIST CYBERSECURITY PRACTICE GUIDES

NIST Cybersecurity Practice Guides (Special Publication 1800 series) target specific cybersecurity challenges in the public and private sectors. They are practical, user-friendly guides that facilitate the adoption of standards-based approaches to cybersecurity. They show members of the information security community how to implement example solutions that help them align with relevant standards and best practices, and provide users with the materials lists, configuration files, and other information they need to implement a similar approach.

The documents in this series describe example implementations of cybersecurity practices that businesses and other organizations may voluntarily adopt. These documents do not describe regulations or mandatory practices, nor do they carry statutory authority.

KEYWORDS

application-layer onboarding; bootstrapping; Internet of Things (IoT); Manufacturer Usage Description (MUD); network-layer onboarding; onboarding; Wi-Fi Easy Connect.

ACKNOWLEDGMENTS

We are grateful to the following individuals for their generous contributions of expertise and time.

Name	Organization
Amogh Guruprasad Deshmukh	Aruba, a Hewlett Packard Enterprise company
Danny Jump	Aruba, a Hewlett Packard Enterprise company

Name	Organization
Bart Brinkman	Cisco
Eliot Lear	Cisco
Peter Romness	Cisco
Tyler Baker	Foundries.io
George Grey	Foundries.io
David Griego	Foundries.io
Fabien Gremaud	Kudelski IoT
Faith Ryan	The MITRE Corporation
Toby Ealden	NquiringMinds
John Manslow	NquiringMinds
Antony McCaigue	NquiringMinds
Alexandru Mereacre	NquiringMinds
Loic Cavaille	NXP Semiconductors
Mihai Chelalau	NXP Semiconductors
Julien Delplancke	NXP Semiconductors
Anda-Alexandra Dorneanu	NXP Semiconductors
Todd Nuzum	NXP Semiconductors
Nicusor Penisoara	NXP Semiconductors
Laurentiu Tudor	NXP Semiconductors
Pedro Fuentes	SEALSQ, a subsidiary of WISeKey

Name	Organization
Gweltas Radenac	SEALSQ, a subsidiary of WISeKey
Kalvin Yang	SEALSQ, a subsidiary of WISeKey

The Technology Partners/Collaborators who participated in this build submitted their capabilities in response to a notice in the Federal Register. Respondents with relevant capabilities or product components were invited to sign a Cooperative Research and Development Agreement (CRADA) with NIST, allowing them to participate in a consortium to build this example solution. We worked with:

Technology Collaborators

Aruba, a Hewlett Packard Enterprise company
CableLabs
Cisco
Foundries.io
Kudelski IoT
NquiringMinds
NXP Semiconductors
Open Connectivity Foundation (OCF)
Sandelman Software Works
SEALSQ, a subsidiary of WISeKey
Silicon Labs

DOCUMENT CONVENTIONS

The terms “shall” and “shall not” indicate requirements to be followed strictly to conform to the publication and from which no deviation is permitted. The terms “should” and “should not” indicate that among several possibilities, one is recommended as particularly suitable without mentioning or excluding others, or that a certain course of action is preferred but not necessarily required, or that (in the negative form) a certain possibility or course of action is discouraged but not prohibited. The terms “may” and “need not” indicate a course of action permissible within the limits of the publication. The terms “can” and “cannot” indicate a possibility and capability, whether material, physical, or causal.

CALL FOR PATENT CLAIMS

This public review includes a call for information on essential patent claims (claims whose use would be required for compliance with the guidance or requirements in this Information Technology Laboratory (ITL) draft publication). Such guidance and/or requirements may be directly stated in this ITL Publication or by reference to another publication. This call also includes disclosure, where known, of the existence of pending U.S. or foreign patent applications relating to this ITL draft publication and of any relevant unexpired U.S. or foreign patents.

ITL may require from the patent holder, or a party authorized to make assurances on its behalf, in written or electronic form, either:

a) assurance in the form of a general disclaimer to the effect that such party does not hold and does not currently intend holding any essential patent claim(s); or

b) assurance that a license to such essential patent claim(s) will be made available to applicants desiring to utilize the license for the purpose of complying with the guidance or requirements in this ITL draft publication either:

1. under reasonable terms and conditions that are demonstrably free of any unfair discrimination; or
2. without compensation and under reasonable terms and conditions that are demonstrably free of any unfair discrimination.

Such assurance shall indicate that the patent holder (or third party authorized to make assurances on its behalf) will include in any documents transferring ownership of patents subject to the assurance, provisions sufficient to ensure that the commitments in the assurance are binding on the transferee, and that the transferee will similarly include appropriate provisions in the event of future transfers with the goal of binding each successor-in-interest.

The assurance shall also indicate that it is intended to be binding on successors-in-interest regardless of whether such provisions are included in the relevant transfer documents.

Such statements should be addressed to: iot-onboarding@nist.gov.

Contents

1 Introduction....................1
1.1 How to Use This Guide....................1
2 Risks Addressed by Trusted Network-Layer Onboarding and Lifecycle Management 3
2.1 Risks to the Network....................3
2.1.1 Risks to the Network Due to Device Limitations 3
2.1.2 Risks to the Network Due to Use of Shared Network Credentials 3
2.1.3 Risks to the Network Due to Insecure Network Credential Provisioning 4
2.1.4 Risks to the Network Due to Supply Chain Attacks 4
2.2 Risks to the Device....................4
2.3 Risks to Secure Lifecycle Management 4
2.4 Limitations and Dependencies of Trusted Onboarding....................5
3 Mapping Use Cases, Approach, and Terminology 6
3.1 Use Cases....................6
3.2 Mapping Producers....................7
3.3 Mapping Approach 7
3.3.1 Mapping Terminology....................8
3.3.2 Mapping Process....................8
4 Mappings....................9
4.1 NIST CSF Subcategory Mappings....................10
4.1.1 Mappings Between Reference Design Functions and NIST CSF Subcategories....................10
4.1.2 Mappings Between Specific Onboarding Protocols and NIST CSF Subcategories 10
4.1.3 Mappings Between Specific Builds and NIST CSF Subcategories....................10
4.2 NIST SP 800-53 Control Mappings 12
4.2.1 Mappings Between Reference Design Functions and NIST SP 800-53 Controls....................12
4.2.2 Mappings Between Specific Onboarding Protocols and NIST SP 800-53 Controls 12
4.2.3 Mappings Between Specific Builds and NIST SP 800-53 Controls 13
Appendix A References 15

1 Introduction

In this project, the National Cybersecurity Center of Excellence (NCCoE) applies standards, recommended practices, and commercially available technology to demonstrate various mechanisms for trusted network-layer onboarding of IoT devices and lifecycle management of those devices. We show how to provision network credentials to IoT devices in a trusted manner and maintain a secure posture throughout the device lifecycle.

This volume of the NIST Cybersecurity Practice Guide discusses risks addressed by the trusted IoT device network-layer onboarding and lifecycle management reference design. It also maps between cybersecurity functionality provided by logical components of the reference design and Subcategories in the NIST Cybersecurity Framework (CSF) and controls in NIST Special Publication (SP) 800-53, *Security and Privacy Controls for Information Systems and Organizations*. (Note: The reference design is described in detail in NIST SP 1800-36B, Section 4.)

Mappings are also provided between cybersecurity functionality provided by specific network-layer onboarding protocols (e.g., Wi-Fi Easy Connect and Bootstrapping Remote Secure Key Infrastructure [BRSKI]) and those same Subcategories and controls, as well as between cybersecurity functionality provided by builds of the reference design that have been implemented as part of this project and those same Subcategories and controls. (Note: the composition of the builds is described in detail in the appendices of NIST SP 1800-36B.)

None of the mappings we provide is intended to be exhaustive; the mappings focus on the strongest relationships involving each reference design cybersecurity function in order to help organizations prioritize their work. The mappings help users understand how trusted IoT device network-layer onboarding and lifecycle management can help them achieve their cybersecurity goals in terms of CSF Subcategories and SP 800-53 controls. The mappings also help users understand how they can implement trusted onboarding and lifecycle management by identifying how trusted onboarding functionality is supported by the user’s existing implementations of CSF Subcategories and SP 800-53 controls.

1.1 How to Use This Guide

This NIST Cybersecurity Practice Guide demonstrates a standards-based reference design for implementing trusted IoT device network-layer onboarding and lifecycle management and describes various example implementations of this reference design. Each of these implementations, which are known as *builds,* is standards-based and is designed to help provide assurance that networks are not put at risk as new IoT devices are added to them and help safeguard IoT devices from being taken over by unauthorized networks. The reference design described in this practice guide is modular and can be deployed in whole or in part, enabling organizations to incorporate trusted IoT device network-layer onboarding and lifecycle management into their legacy environments according to goals that they have prioritized based on risk, cost, and resources.

NIST is adopting an agile process to publish this content. Each volume is being made available as soon as possible rather than delaying release until all volumes are completed.

This guide contains five volumes:

- NIST SP 1800-36A: *Executive Summary* – why we wrote this guide, the challenge we address, why it could be important to your organization, and our approach to solving this challenge
- NIST SP 1800-36B*: Approach, Architecture, and Security Characteristics* – what we built and why
- NIST SP 1800-36C: *How-To Guides* – instructions for building the example implementations, including all the security-relevant details that would allow you to replicate all or parts of this project
- NIST SP 1800-36D: *Functional Demonstrations* – use cases that have been defined to showcase trusted IoT device network-layer onboarding and lifecycle management security capabilities, and the results of demonstrating these use cases with each of the example implementations
- NIST SP 1800-36E*: Risk and Compliance Management* – risk analysis and mapping of trusted IoT device network-layer onboarding and lifecycle management security characteristics to cybersecurity standards and best practices **(you are here)**

Depending on your role in your organization, you might use this guide in different ways:

Business decision makers, including chief security and technology officers, will be interested in the *Executive Summary, NIST SP 1800-36A*, which describes the following topics:

- challenges that enterprises face in migrating to the use of trusted IoT device network-layer onboarding
- example solutions built at the NCCoE
- benefits of adopting the example solution

Technology or security program managers who are concerned with how to identify, understand, assess, and mitigate risk will be interested in *NIST SP 1800-36B*, which describes what we did and why.

Also, Section 4 of *NIST SP 1800-36E* will be of particular interest. Section 4, *Mappings*, maps logical components of the general trusted IoT device network-layer onboarding and lifecycle management reference design to security characteristics listed in various cybersecurity standards and recommended practices documents, including *Framework for Improving Critical Infrastructure Cybersecurity* (NIST Cybersecurity Framework) and *Security and Privacy Controls for Information Systems and Organizations* (NIST SP 800-53).

You might share the *Executive Summary, NIST SP 1800-36A*, with your leadership team members to help them understand the importance of using standards-based trusted IoT device network-layer onboarding and lifecycle management implementations.

IT professionals who want to implement similar solutions will find the whole practice guide useful. You can use the how-to portion of the guide, *NIST SP 1800-36C*, to replicate all or parts of the builds created in our lab. The how-to portion of the guide provides specific product installation, configuration, and integration instructions for implementing the example solution. We do not re-create the product manufacturers’ documentation, which is generally widely available. Rather, we show how we incorporated the products together in our environment to create an example solution. Also, you can use *Functional Demonstrations, NIST SP 1800-36D*, which provides the use cases that have been defined to showcase trusted IoT device network-layer onboarding and lifecycle management security capabilities and the results of demonstrating these use cases with each of the example implementations. Finally, *NIST SP 1800-36E* will be helpful in explaining the security functionality that the components of each build provide.

This guide assumes that IT professionals have experience implementing security products within the enterprise. While we have used a suite of commercial products to address this challenge, this guide does not endorse these particular products. Your organization can adopt this solution or one that adheres to these guidelines in whole, or you can use this guide as a starting point for tailoring and implementing parts of a trusted IoT device network-layer onboarding and lifecycle management solution. Your organization's security experts should identify the products that will best integrate with your existing tools and IT system infrastructure. We hope that you will seek products that are congruent with applicable standards and recommended practices.

A NIST Cybersecurity Practice Guide does not describe "the" solution, but example solutions. We seek feedback on the publication's contents and welcome your input. Comments, suggestions, and success stories will improve subsequent versions of this guide. Please contribute your thoughts to iot-onboarding@nist.gov.

2 Risks Addressed by Trusted Network-Layer Onboarding and Lifecycle Management

Historically, IoT devices have not tended to be onboarded to networks in a trusted manner. This has left networks open to the threat of having unauthorized devices connect to them. It has also left devices open to the threat of being onboarded to networks that are not authorized to control them.

2.1 Risks to the Network

Unauthorized devices that are able to connect to a network pose many risks to that network. They may be able to send and receive data on that network, scan the network for vulnerabilities, eavesdrop on the communications of other devices, and attack other connected devices to exfiltrate or modify their data or to compromise those devices and co-opt them into service to launch distributed denial of service (DDoS) attacks.

2.1.1 Risks to the Network Due to Device Limitations

Many IoT devices are manufactured to be as inexpensive as possible, which sometimes means that the devices are not equipped with secure storage, cryptographic modules, unique authoritative birth credentials, or other features needed to enable the devices to be identified and authenticated. This can make it impossible for a network to determine if a device attempting to connect to it is the intended device. Lack of these features can also make it impossible to protect the confidentiality of a device's network credentials, both during the provisioning process and after the credentials have been installed on the device.

2.1.2 Risks to the Network Due to Use of Shared Network Credentials

If a network uses a single network password that is shared among all devices rather than providing each device with a unique network credential, the network will be vulnerable to having unauthorized devices connect to it if the shared network password falls into the wrong hands, which can happen relatively easily. It also means that the network will permit devices to connect to it simply because a device presents the correct shared password, regardless of the device's type or identity, or whether it has any legitimate reason to connect to the network.

2.1.3 Risks to the Network Due to Insecure Network Credential Provisioning

If devices are manually provisioned with their network credentials, the provisioning process is error-prone, cumbersome, and vulnerable to having the device's network credentials disclosed. If the devices are provisioned automatically over Wi-Fi or some other interface that does not use an encrypted channel, the credentials are also vulnerable to unauthorized disclosure. If the network credentials are not provisioned in a trusted manner, the credentials are vulnerable to disclosure not only the first time the device is onboarded to the network, but every time it is onboarded, which may occur many times during the device lifecycle. For example, the device may need to be re-onboarded periodically to change its credentials in accordance with security policy, or it may need to be re-onboarded due to a security breach, hardware repair, security update, or other reasons. Any insecure features of the onboarding process, therefore, will render the device and network vulnerable every time the device is onboarded.

2.1.4 Risks to the Network Due to Supply Chain Attacks

If a device is compromised while in the supply chain or at some other point prior to being onboarded, then even though the device may be onboarded in a trusted manner, it may still pose a threat to the network, its data, and all devices connected to it. If, on the other hand, the trusted network-layer onboarding mechanism is integrated with a device attestation or supply chain management service that is capable of evaluating the integrity and provenance of the device and detecting that it has been compromised or may have been tampered with, the trusted network-layer onboarding mechanism could prevent such a compromised device from being onboarded and connected to the network.

2.2 Risks to the Device

Although it is relatively easy for one network to masquerade as another, IoT devices often do not authenticate the identity of the networks to which they allow themselves to be onboarded and connected. Devices may be unwittingly tricked into onboarding and connecting to imposter networks that are not authorized to onboard them. This makes those devices vulnerable to being taken control of by those unauthorized networks and thereby prevented from connecting to and providing their intended function on their authorized network.

2.3 Risks to Secure Lifecycle Management

Even if a device is authorized to connect to a network and the network is authorized to control the device, if the device has not been onboarded in a trusted manner, then other security-related operations that are performed after the device has connected to the network may not have as secure a foundation as they would if the device had been onboarded in a trusted manner. For example, if device communications intent enforcement is performed but the integrity and confidentiality of the communicated device intent information was not protected (as it would be by a trusted network-layer onboarding mechanism), then trust in the device communications intent enforcement mechanism may not be as robust as it could have been. Similarly, if application-layer onboarding is performed after the device connects, but the information needed to bootstrap the application-layer onboarding process did not have its integrity and confidentiality protected (as it would be by a trusted network-layer onboarding mechanism), then trust in the application-layer onboarding mechanism may not be as robust as it could have been. Lack of trust in the application-layer onboarding mechanism may, in turn, undermine trust in the device lifecycle management or other application-layer service that is invoked as part of the application-layer onboarding process.

2.4 Limitations and Dependencies of Trusted Onboarding

While implementing trusted IoT device network-layer onboarding and lifecycle management addresses many risks, it also has limitations. Use of trusted network-layer onboarding is designed to enable IoT devices to be provisioned with unique local network credentials in a manner that preserves credential confidentiality. As part of the trusted network-layer onboarding process, the device and the network may mutually authenticate one another, thereby protecting the network from having unauthorized devices connect to it and the device from being taken over by an unauthorized network. However, if the network also enables devices that do not support the trusted network-layer onboarding solution to be provisioned with network credentials and connect to it using a different (untrusted) onboarding solution, the network and all devices on it will still be at risk from IoT devices that have been onboarded using untrusted mechanisms, and the devices that are onboarded using untrusted mechanisms will still be at risk of being taken over by networks that are not authorized to control them.

The trusted network-layer onboarding solution leverages the device's unique, authoritative *birth credentials*, which are provisioned to the device by the device manufacturer and must consist, at a minimum, of a unique device identity and a secret. The trustworthiness of the network-layer onboarding process and the network credentials that it provisions to the device depends on the uniqueness, integrity, and confidentiality of the device's birth credentials which, in many cases, depend on the device's hardware root of trust. If the manufacturer does not ensure that the device's credentials are unique, the identity of the device cannot be definitively authenticated. If the manufacturer is not able to maintain the confidentiality of the secret that is part of the device credentials, the trustworthiness of the device authentication process will be undermined, and the channel over which the device's credentials are provisioned will be vulnerable to eavesdropping.

The trusted network-layer onboarding solution depends upon the trustworthiness of the device's secure storage to ensure the confidentiality of the device and network credentials. If the device's secure storage is vulnerable, the trustworthiness of the network-layer onboarding process and the confidentiality of the device's network credentials will be compromised. If the secure storage in which the device's network credentials are stored is vulnerable, the network will be at risk of having unauthorized devices attach to it.

If the trusted network-layer onboarding mechanism is integrated with additional security capabilities such as device attestation, device communications intent enforcement, application-layer onboarding, and device lifecycle management, it can further increase trust in both the IoT device and, by extension, the network to which the device connects, assuming that these additional security capabilities themselves are secure and robust. If these security capabilities are not implemented correctly, then integrating with them is of no additional value and in fact may provide a false sense of security.

3 Mapping Use Cases, Approach, and Terminology

A *mapping* indicates that one concept is related to another concept. The remainder of this volume describes the mappings between trusted IoT device network-layer onboarding and lifecycle management cybersecurity functions and the security characteristics enumerated in relevant cybersecurity documents.

For this mapping, we have used the supportive relationship mapping style as defined in Section 4.2 of draft NIST Internal Report (IR) 8477, *Mapping Relationships Between Documentary Standards, Regulations, Frameworks, and Guidelines: Developing Cybersecurity and Privacy Concept Mappings* [1].

Each set of mappings involves one of the following types of trusted IoT device network-layer onboarding and lifecycle management cybersecurity functions:

- Cybersecurity functions performed by the reference design's logical components (see NIST SP 1800-36B Section 4)
- Cybersecurity functions provided by specific network-layer onboarding protocols (e.g., Wi-Fi Easy Connect and BRSKI)
- Cybersecurity functions provided by builds of the reference design that have been implemented as part of this project

Each of the cybersecurity functions is mapped to the security characteristics concepts found in the following widely used cybersecurity guidance documents:

- Subcategories from the NIST Cybersecurity Framework (CSF) 1.1 [2] which are also mapped to *The NIST Cybersecurity Framework 2.0 (CSF 2.0)* [3]. The CSF identifies enterprise-level security outcomes. Stakeholders have identified these outcomes as helpful for managing cybersecurity risk, but organizations adopting the CSF need to determine how to achieve the outcomes. Executive Order (EO) 13800, *Strengthening the Cybersecurity of Federal Networks and Critical Infrastructure* [4], made the CSF mandatory for federal government agencies, and other government agencies and sectors have also made the CSF mandatory.
- Security controls from NIST SP 800-53r5 (*Security and Privacy Controls for Information Systems and Organizations*) [5]. NIST SP 800-53 identifies security controls that apply to systems on which those enterprises are reliant. Which SP 800-53 controls need to be employed depends on system functions and a risk assessment of the perceived impact of loss of system functionality or exposure of information from the system to unauthorized entities. In the case of systems owned by or operated on behalf of federal government enterprises, the risk assessment and applicable SP 800-53 controls are mandated under the Federal Information Security Modernization Act (FISMA) [6]. Many other governments and private sector organizations voluntarily employ the Risk Management Framework [7] and associated SP 800-53 controls.

3.1 Use Cases

All of the elements in these mappings—the trusted IoT device network-layer onboarding and lifecycle management cybersecurity functions, cybersecurity functions provided by specific network-layer onboarding protocols, cybersecurity functions provided by specific builds, CSF Subcategories, and SP 800-53 controls—are concepts involving ways to reduce cybersecurity risk.

There are two primary use cases for this mapping. They are not intended to be comprehensive, but rather to capture the strongest relationships involving the trusted IoT device network-layer onboarding and lifecycle management cybersecurity functions.

1. **Why should organizations implement trusted IoT device network-layer onboarding and lifecycle management?** This use case identifies how implementing trusted IoT device network-layer onboarding and lifecycle management can support organizations with achieving CSF Subcategories and SP 800-53 controls. This helps communicate to an organization's chief information security officer, security team, and senior management that expending resources to implement trusted IoT device network-layer onboarding and lifecycle management can also aid in fulfilling other security requirements.

2. **How can organizations implement trusted IoT device network-layer onboarding and lifecycle management?** This use case identifies how an organization's existing implementations of CSF Subcategories and SP 800-53 controls can help support a trusted IoT device network-layer onboarding and lifecycle management implementation. An organization wanting to implement trusted IoT device network-layer onboarding and lifecycle management might first assess its current security capabilities so that it can plan how to add missing capabilities and enhance existing capabilities. Organizations can leverage their existing security investments and prioritize future security technology deployment to address the gaps.

These mappings are intended to be used by any organization that is interested in implementing trusted IoT device network-layer onboarding and lifecycle management or that has begun or completed an implementation.

3.2 Mapping Producers

The NCCoE trusted IoT device network-layer onboarding and lifecycle management project team performed the mappings between the cybersecurity functions performed by the reference design's logical components (see NIST SP 1800 36B Section 4) and the security characteristics in the cybersecurity documents. They also performed the mappings between the cybersecurity functions performed by the specific network-layer onboarding protocols (i.e., Wi-Fi Easy Connect and BRSKI) and the security characteristics in the cybersecurity documents. These mappings were performed with input and feedback from the collaborators who have contributed technology to the builds of the reference design. Collaborators for each build, in conjunction with the NCCoE trusted IoT device network-layer onboarding and lifecycle management project team, performed the mappings between the cybersecurity functions provided by their contributed technologies in each build and the security characteristics in the cybersecurity documents.

3.3 Mapping Approach

In addition to performing general mappings between the reference design's cybersecurity functions and various sets of security characteristics, as well as between specific network-layer onboarding protocol cybersecurity functions and various sets of security characteristics, the NCCoE asked the collaborators for each build to indicate the mapping between the cybersecurity functions their technology components provide in that build and the sets of security characteristics.

Using the logical components in the reference design as the organizing principle for the initial mapping of cybersecurity functions to security characteristics and then providing onboarding protocol-specific mappings was intended to make it easier for collaborators to map their build-specific technology contributions. Using this approach, the build-specific technology mappings are instantiations of the project's general reference design and protocol-specific mappings for each document.

3.3.1 Mapping Terminology

In this publication, we use the following relationship types from NIST IR 8477 [1] to describe how the functions in our reference design are related to the NIST reference documents. Note that the *Supports* relationship applies only to use case 1 in Section 3.1 and the *Is Supported By* relationship applies only to use case 2.

- **Supports:** Trusted IoT device network-layer onboarding and lifecycle management function X *supports* security control/Subcategory/capability/requirement Y when X can be applied alone or in combination with one or more other functions to achieve Y in whole or in part.
- **Is Supported By:** Trusted IoT device network-layer onboarding and lifecycle management function X is *supported by* security control/Subcategory/capability/requirement Y when Y can be applied alone or in combination with one or more other security controls/Subcategories/capabilities/requirements to achieve X in whole or in part.

Each *Supports* and *Is Supported By* relationship has one of the following properties assigned to it:

- **Example of:** The supporting concept X is one way (*an example*) of achieving the supported concept Y in whole or in part. However, Y could also be achieved without applying X.
- **Integral to:** The supporting concept X is *integral to* and a component of the supported concept Y. X must be applied as part of achieving Y.
- **Precedes:** The supporting concept X *precedes* the supported concept Y when X must be achieved before applying Y. In other words, X is a prerequisite for Y.

When determining whether a reference design function's support for a given CSF Subcategory or SP 800-53 control is integral to that support versus an example of that support, we do not consider how that function may in general be used to support the Subcategory, control, capability, or requirement. Rather, we consider only how that function is intended to support that Subcategory, control, capability, or requirement within the context of our reference design.

Also, when determining whether a function is supported by a CSF Subcategory, SP 800-53 control, capability, etc. with the relationship property of *precedes*, we do not consider whether it is possible to apply the function without first achieving the Subcategory, control, capability, or requirement. Rather, we consider whether, according to our reference design, the Subcategory, control, capability, or requirement is to be achieved prior to applying that function.

3.3.2 Mapping Process

The process that the NCCoE used to create the mapping from the logical components of the reference design to the security characteristics of a given document was as follows:

1. Create a table that lists each of the logical components of the reference design in column 1.

2. Describe each logical component's cybersecurity function in column 2.

3. Map each cybersecurity function to each of the security characteristics in the document to which the function is most strongly related, and list each of these security characteristics on different sub-rows within column 3. Begin each security characteristic entry with an underlined keyword that describes the mapping's relationship type (i.e., *Supports*, *Is Supported By*). After the keyword indicating the relationship type, put in parentheses the underlined keyword describing the relationship's property (i.e., *Example of*, *Integral to*, or *Precedes*).

4. In the fourth column, provide a brief explanation of why that relationship type and property apply to the mapping.

5. After completing the mapping table entries as described above for all the logical components in the reference design, examine the mapping in the other direction, i.e., starting with the security characteristics listed in the document and considering whether they have a relationship to the logical components' cybersecurity functions in the reference design. In other words, step through each of the security characteristics in the document and determine if there is some logical component in the reference design that has a strong relationship to that security characteristic. If so, add an entry for that security characteristic mapping to that logical component's row in the table. By examining the mapping in both directions in this manner, security characteristic mappings are less likely to be overlooked or omitted.

6. Once these steps are complete, any rows in the table that don't have any mappings should be deleted.

The NCCoE applied this mapping process separately for each reference document. None of the mappings is intended to be exhaustive; they all focus on the strongest relationships involving each cybersecurity function in order to help organizations prioritize their work. Mapping every possible relationship, no matter how tenuous, would create so many mappings that they would not have any value in prioritization.

4 Mappings

The mappings are provided in the form of Excel files. Links to the mapping Excel files are organized in the remainder of this document as follows:

- Section 4.1 – NIST CSF 1.1 [2] and NIST CSF 2.0 [3] mappings. These include:
    - Section 4.1.1 – Mappings between reference design functions and NIST CSF Subcategories
    - Section 4.1.2 – Mappings between specific onboarding protocol (i.e., Wi-Fi Easy Connect and BRSKI) functions and NIST CSF Subcategories
    - Section 4.1.3 – Mappings between specific build functions and NIST CSF Subcategories
- Section 4.2 – NIST SP 800-53r5 [5] mappings. These include:
    - Section 4.2.1 – Mappings between reference design functions and NIST SP 800-53r5 controls

  - Section 4.2.2 – Mappings between specific onboarding protocol (i.e., Wi-Fi Easy Connect and BRSKI) functions and NIST SP 800-53r5 controls
  - Section 4.2.3 – Mappings between specific build functions and NIST SP 800-53r5 controls

4.1 NIST CSF Subcategory Mappings

This section provides links to mappings between various elements that provide trusted network-layer onboarding functionality and NIST CSF Subcategories.

4.1.1 Mappings Between Reference Design Functions and NIST CSF Subcategories

This Excel file provides mappings between the logical components of the reference design and the NIST CSF Subcategories. These mappings indicate how trusted IoT device network-layer onboarding and lifecycle management functions help support CSF Subcategories and vice versa.

Link to the Excel file called "IoT Volume E CSF 1-1 and 2-0", and to the tab called "CSF-to-Reference Arch" (first tab)

4.1.2 Mappings Between Specific Onboarding Protocols and NIST CSF Subcategories

This section provides mappings between the functionality provided by two network-layer onboarding protocols, Wi-Fi Easy Connect and BRSKI, and the NIST CSF Subcategories.

4.1.2.1 Mapping Between Wi-Fi Easy Connect and NIST CSF Subcategories

This Excel file provides a mapping between the functionality provided by the Wi-Fi Easy Connect protocol and the NIST CSF Subcategories. These mappings indicate how Wi-Fi Easy Connect functionality helps support CSF Subcategories and vice versa.

Link to the Excel file called "CSF 1.1 and 2.0 Tables", and to the tab called "CSF-to-Wi-Fi EasyCnct" (third tab)

4.1.2.2 Mapping Between BRSKI and NIST CSF Subcategories

This Excel file provides a mapping between the functionality provided by BRSKI and the NIST CSF Subcategories. These mappings indicate how BRSKI functionality helps support CSF Subcategories and vice versa.

Link to the Excel file called "CSF 1.1 and 2.0 Tables", and to the tab called "CSF-to-BRSKI" (second tab)

4.1.3 Mappings Between Specific Builds and NIST CSF Subcategories

This section provides mappings between the functionality provided by builds of the trusted IoT device network-layer onboarding and lifecycle management reference design that were implemented as part of this project and the NIST CSF Subcategories.

4.1.3.1 Mapping Between Build 1 and NIST CSF Subcategories

Build 1 is an implementation of network-layer onboarding that uses the Wi-Fi Easy Connect protocol. The onboarding infrastructure and related technology components for Build 1 have been provided by Aruba/HPE. IoT devices that were onboarded using Build 1 were provided by Aruba/HPE and CableLabs. The technologies used in Build 1 are detailed in Appendix C of SP 1800-36B.

This Excel file details the mapping between the functionality provided by Build 1 components and CSF Subcategories. These mappings indicate how these components help support CSF Subcategories and vice versa.

Link to the Excel file called "CSF 1.1 and 2.0 Tables", and to the tab called "CSF-to-B1" (fourth tab)

4.1.3.2 Mapping Between Build 2 and NIST CSF Subcategories

Build 2 is an implementation of network-layer onboarding that uses the Wi-Fi Easy Connect protocol. The onboarding infrastructure and related technology components for Build 2 have been provided by CableLabs and OCF. IoT devices that were onboarded using Build 2 were provided by CableLabs, OCF, and Aruba/HPE. The technologies used in Build 2 are detailed in Appendix D of SP 1800-36B.

This Excel file details the mapping between the functionality provided by Build 2 components and CSF Subcategories. These mappings indicate how these components help support CSF Subcategories and vice versa.

Link to the Excel file called "CSF 1.1 and 2.0 Tables", and to the tab called "CSF-to-B2" (fifth tab)

4.1.3.3 Mapping Between Build 3 and NIST CSF Subcategories

Build 3 is an implementation of network-layer onboarding that uses BRSKI. The onboarding infrastructure and related technology components for Build 3 have been provided by Sandelman Software Works. The IoT device that was used to demonstrate onboarding in Build 3 was a pledge simulator provided by Sandelman. The technologies used in Build 3 are detailed in Appendix E of SP 1800-36B.

This Excel file details the mapping between the functionality provided by Build 3 components and CSF Subcategories. These mappings indicate how these components help support CSF Subcategories and vice versa.

Link to the Excel file called "CSF 1.1 and 2.0 Tables", and to the tab called "CSF-to-B3" (sixth tab)

4.1.3.4 Mapping Between Build 4 and NIST CSF Subcategories

Build 4 is an implementation of network-layer connection to an OpenThread network using pre-provisioned network credentials as well as independent application-layer onboarding using the Kudelski KeySTREAM service. The network infrastructure and related technology components for Build 4 have been provided by Silicon Labs and Kudelski. The IoT device that was used to demonstrate onboarding in Build 4 was provided by Silicon Labs. The technologies used in Build 4 are detailed in Appendix F of SP 1800-36B.

This Excel file details the mapping between the functionality provided by Build 4 components and CSF Subcategories These mappings indicate how these components help support CSF Subcategories and vice versa.

Link to the Excel file called "CSF 1.1 and 2.0 Tables", and to the tab called "CSF-to-B4" (seventh tab)

4.1.3.5 Mapping Between Build 5 and NIST CSF Subcategories

Build 5 is an implementation of network-layer onboarding using BRSKI over Wi-Fi, as well as demonstration of a continuous authorization service. The network layer onboarding infrastructure and related technology components for Build 5 have been provided by NquiringMinds. The IoT devices that were used to demonstrate onboarding in Build 5 were provided by NquiringMinds. The technologies used in Build 5 are detailed in Appendix G of SP 1800-36B.

This Excel file details the mapping between the functionality provided by Build 5 components and CSF Subcategories. These mappings indicate how these components help support CSF Subcategories and vice versa.

Link to the Excel file called "CSF 1.1 and 2.0 Tables", and to the tab called "CSF-to-B5" (eighth tab)

4.2 NIST SP 800-53 Control Mappings

This section provides mappings between various elements that provide trusted network-layer onboarding functionality and NIST SP 800-53 controls.

4.2.1 Mappings Between Reference Design Functions and NIST SP 800-53 Controls

This Excel file provides a mapping between the logical components of the reference design and NIST SP 800-53 security controls. These mappings indicate how trusted IoT device network-layer onboarding and lifecycle management functions help support NIST SP 800-53 controls. Because hundreds of NIST SP 800-53 controls can help support these functions, we have limited use case 2 (see Section 3.1) mappings to those controls on which specified supporting controls directly depend (e.g., dependence of cryptographic protection on key management). Readers needing to determine how their trusted IoT device network-layer onboarding and lifecycle management implementations support RMF processes can refer to these mappings.

Link to the Excel file called "800-53 Tables", and to the tab called "800-53-to-Reference Arch" (first tab)

4.2.2 Mappings Between Specific Onboarding Protocols and NIST SP 800-53 Controls

This section provides mappings between the functionality provided by specific network-layer onboarding protocols and the NIST SP 800-53 controls. Mappings are provided for both the Wi-Fi Easy Connect protocol and BRSKI.

4.2.2.1 Mapping Between Wi-Fi Easy Connect and NIST SP 800-53 Controls

This Excel file provides a mapping between the functionality provided by the Wi-Fi Easy Connect protocol and the NIST SP 800-53 controls. These mappings indicate how Wi-Fi Easy Connect functions help support NIST SP 800-53 controls and vice versa.

Link to the Excel file called "800-53 Tables", and to the tab called "800-53-to-Wi-Fi EasyCnct" (second tab)

4.2.2.2 Mapping Between BRSKI and NIST SP 800-53 Controls

This Excel file provides a mapping between the functionality provided by BRSKI and the NIST SP 800-53 controls. These mappings indicate how BRSKI functions help support NIST SP 800-53 controls and vice versa.

Link to the Excel file called "800-53 Tables", and to the tab called "800-53-to-BRSKI" (third tab)

4.2.3 Mappings Between Specific Builds and NIST SP 800-53 Controls

This section provides mappings between the functionality provided by builds of the trusted IoT device network-layer onboarding and lifecycle management reference design that were implemented as part of this project and the NIST SP 800-53 controls.

4.2.3.1 Mapping Between Build 1 and NIST SP 800-53 Controls

Build 1 is an implementation of network-layer onboarding that uses the Wi-Fi Easy Connect protocol. The onboarding infrastructure and related technology components for Build 1 have been provided by Aruba/HPE. IoT devices that were onboarded using Build 1 were provided by Aruba/HPE and CableLabs. The technologies used in Build 1 are detailed in Appendix C of SP 1800-36B.

This Excel file details the mapping between the functionality provided by Build 1 components and SP 800-53 controls. These mappings indicate how these components help support SP 800-53 controls and vice versa.

Link to the Excel file called "800-53 Tables", and to the tab called "800-53-to-B1" (fourth tab)

4.2.3.2 Mapping Between Build 2 and NIST SP 800-53 Controls

Build 2 is an implementation of network-layer onboarding that uses the Wi-Fi Easy Connect protocol. The onboarding infrastructure and related technology components for Build 2 have been provided by CableLabs and OCF. IoT devices that were onboarded using Build 2 were provided by CableLabs, OCF, and Aruba/HPE. The technologies used in Build 1 are detailed in Appendix D of SP 1800-36B.

This Excel file details the mapping between the functionality provided by Build 2 components and SP 800-53 controls These mappings indicate how these components help support SP 800-53 controls and vice versa.

Link to the Excel file called “800-53 Tables”, and to the tab called “800-53-to-B2” (fifth tab)

4.2.3.3 Mapping Between Build 3 and NIST SP 800-53 Controls

Build 3 is an implementation of network-layer onboarding that uses BRSKI. The onboarding infrastructure and related technology components for Build 3 have been provided by Sandelman Software Works. The IoT device that was used to demonstrate onboarding in Build 3 was a pledge simulator provided by Sandelman. The technologies used in Build 3 are detailed in Appendix E of SP 1800-36B.

This Excel file details the mapping between the functionality provided by Build 3 components and SP 800-53 controls. These mappings indicate how these components help support SP 800-53 controls and vice versa.

Link to the Excel file called “800-53 Tables”, and to the tab called “800-53-to-B3” (sixth tab)

4.2.3.4 Mapping Between Build 4 and NIST SP 800-53 Controls

Build 4 is an implementation of network-layer connection to an OpenThread network using pre-provisioned network credentials as well as independent application-layer onboarding using the Kudelski KeySTREAM service. The network infrastructure and related technology components for Build 4 have been provided by Silicon Labs and Kudelski. The IoT device that was used to demonstrate onboarding in Build 4 was provided by Silicon Labs. The technologies used in Build 4 are detailed in Appendix F of SP 1800-36B.

This Excel file details the mapping between the functionality provided by Build 4 components and SP 800-53 controls. These mappings indicate how these components help support SP 800-53 controls and vice versa.

Link to the Excel file called “800-53 Tables”, and to the tab called “800-53-to-B4” (seventh tab)

4.2.3.5 Mapping Between Build 5 and NIST SP 800-53 Controls

Build 5 is an implementation of network-layer onboarding using BRSKI over Wi-Fi, as well as demonstration of a continuous authorization service. The network layer onboarding infrastructure and related technology components for Build 5 have been provided by NquiringMinds. The IoT devices that were used to demonstrate onboarding in Build 5 were provided by NquiringMinds. The technologies used in Build 5 are detailed in Appendix G of SP 1800-36B.

This Excel file details the mapping between the functionality provided by Build 5 components and SP 800-53 controls. These mappings indicate how these components help support SP 800-53 controls and vice versa.

Link to the Excel file called “800-53 Tables”, and to the tab called “800-53-to-B5” (eighth tab)

Appendix A References

[1] K. Scarfone, M. Souppaya, and M. Fagan, Mapping Relationships Between Documentary Standards, Regulations, Frameworks, and Guidelines: Developing Cybersecurity and Privacy Content Mappings, National Institute of Standards and Technology (NIST) Internal Report (IR) 8477, Gaithersburg, Md., August 2023, 26 pp. Available: https://doi.org/10.6028/NIST.IR.8477.ipd

[2] National Institute of Standards and Technology (2018) Framework for Improving Critical Infrastructure Cybersecurity, Version 1.1. (National Institute of Standards and Technology, Gaithersburg, MD), NIST Cybersecurity White Paper (CSWP) NIST CSWP 6. https://doi.org/10.6028/NIST.CSWP.6

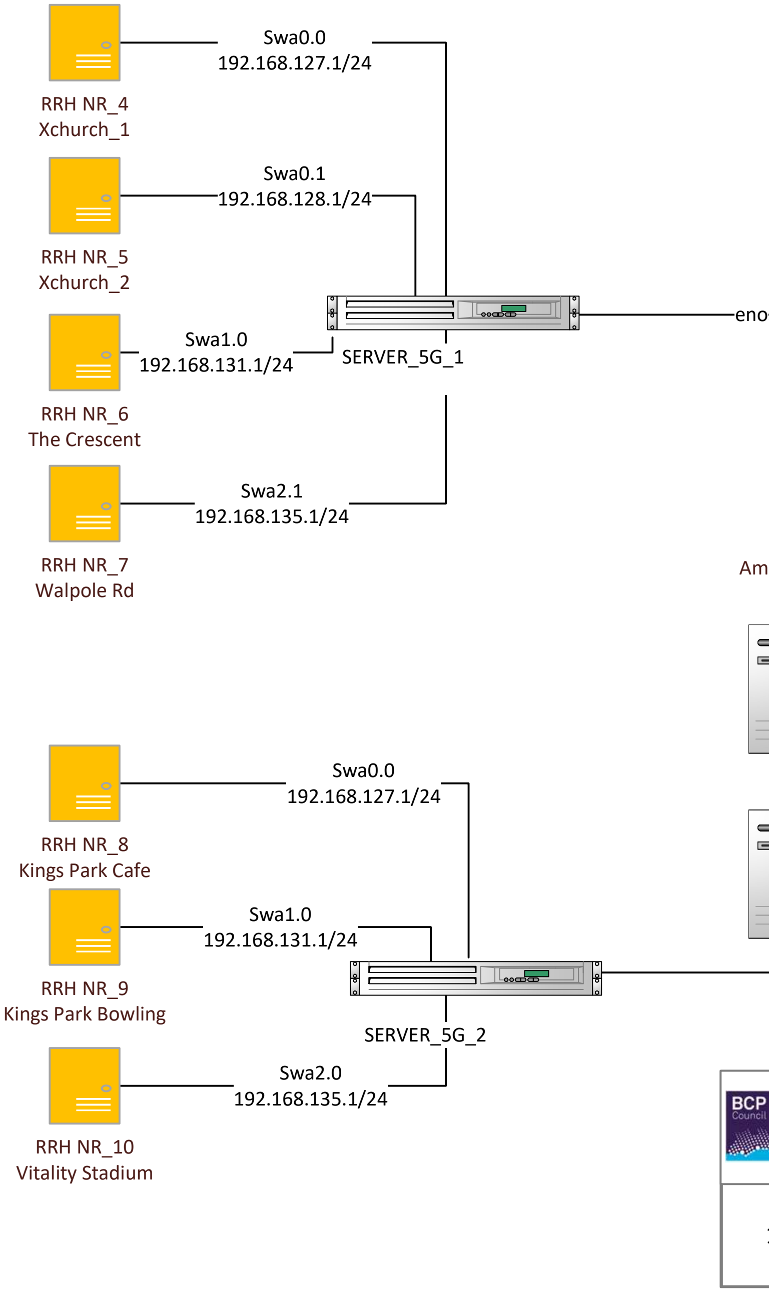
[3] National Institute of Standards and Technology, Version 2.0. The NIST Cybersecurity Framework 2.0 (CSF 2.0) (National Institute of Standards and Technology, Gaithersburg, MD), https://csrc.nist.gov/pubs/cswp/29/the-nist-cybersecurity-framework-20/ipd

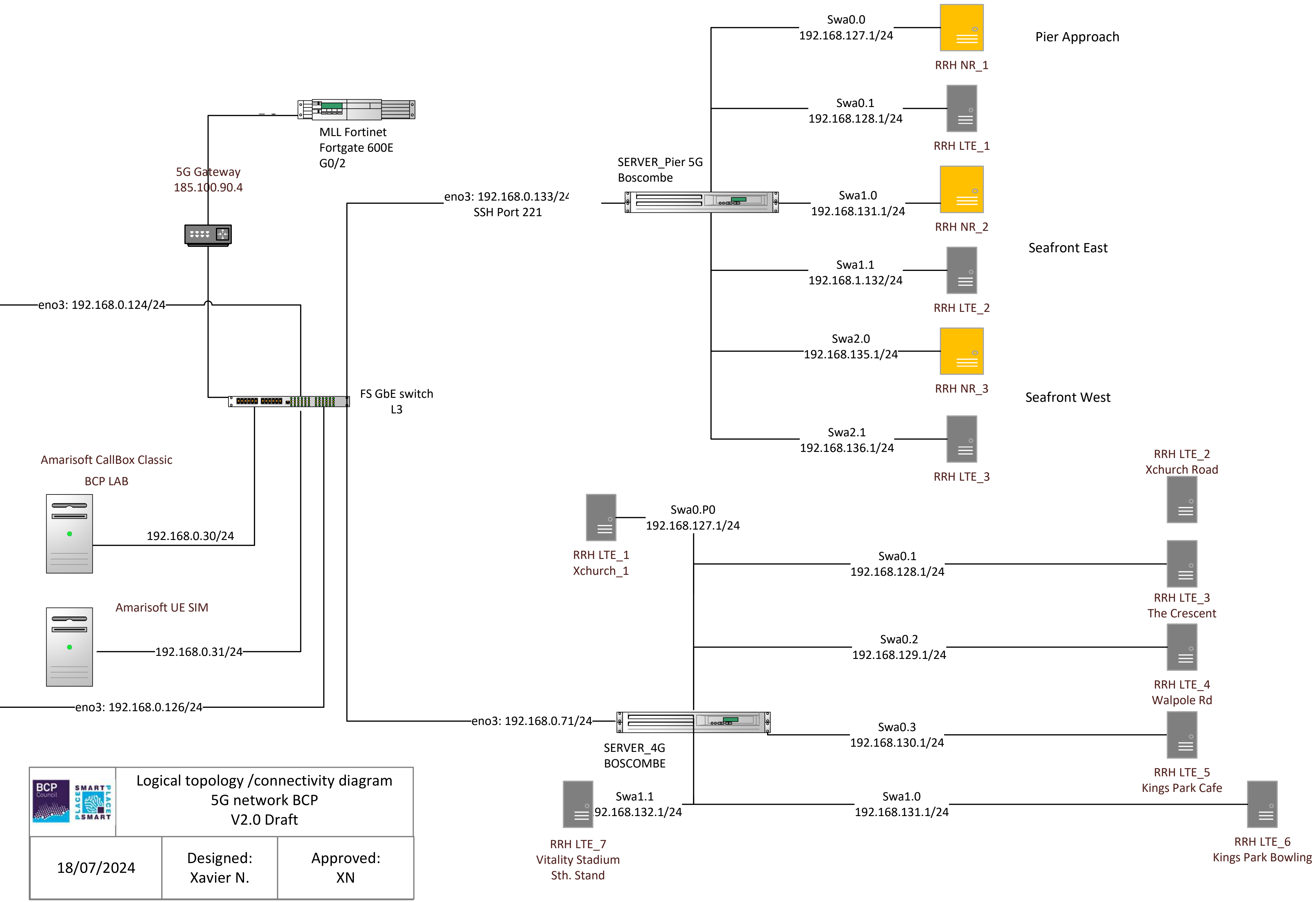
[4] Executive Order 13800 (2017) Strengthening the Cybersecurity of Federal Networks and Critical Infrastructure. (The White House, Washington, DC), DCPD-201700327, May 11, 2017. https://www.govinfo.gov/app/details/DCPD-201700327

[5] Joint Task Force (2020) Security and Privacy Controls for Information Systems and Organizations. (National Institute of Standards and Technology, Gaithersburg, MD), NIST Special Publication (SP) 800-53, Rev. 5. Includes updates as of December 10, 2020. https://doi.org/10.6028/NIST.SP.800-53r5

[6] S.2521 - Federal Information Security Modernization Act of 2014, 113th Congress (2013-2014), Became Public Law No: 113-283, December 18, 2014. Available: https://www.congress.gov/bill/113th-congress/senate-bill/2521

[7] Joint Task Force (2018) Risk Management Framework for Information Systems and Organizations: A System Life Cycle Approach for Security and Privacy. (National Institute of Standards and Technology, Gaithersburg, MD), NIST Special Publication (SP) 800-37, Rev. 2. https://doi.org/10.6028/NIST.SP.800-37r2


Swa0.0
192.168.127.1/24
RRH NR_4
Xchurch_1
Swa0.1
192.168.128.1/24
RRH NR_5
Xchurch_2
Swa1.0
192.168.131.1/24
SERVER_5G_1
eno
RRH NR_6
The Crescent
Swa2.1
192.168.135.1/24
RRH NR_7
Walpole Rd
Am
Swa0.0
192.168.127.1/24
RRH NR_8
Kings Park Cafe
Swa1.0
192.168.131.1/24
RRH NR_9
Kings Park Bowling
SERVER_5G_2
Swa2.0
192.168.135.1/24
RRH NR_10
Vitality Stadium
BCP
Council

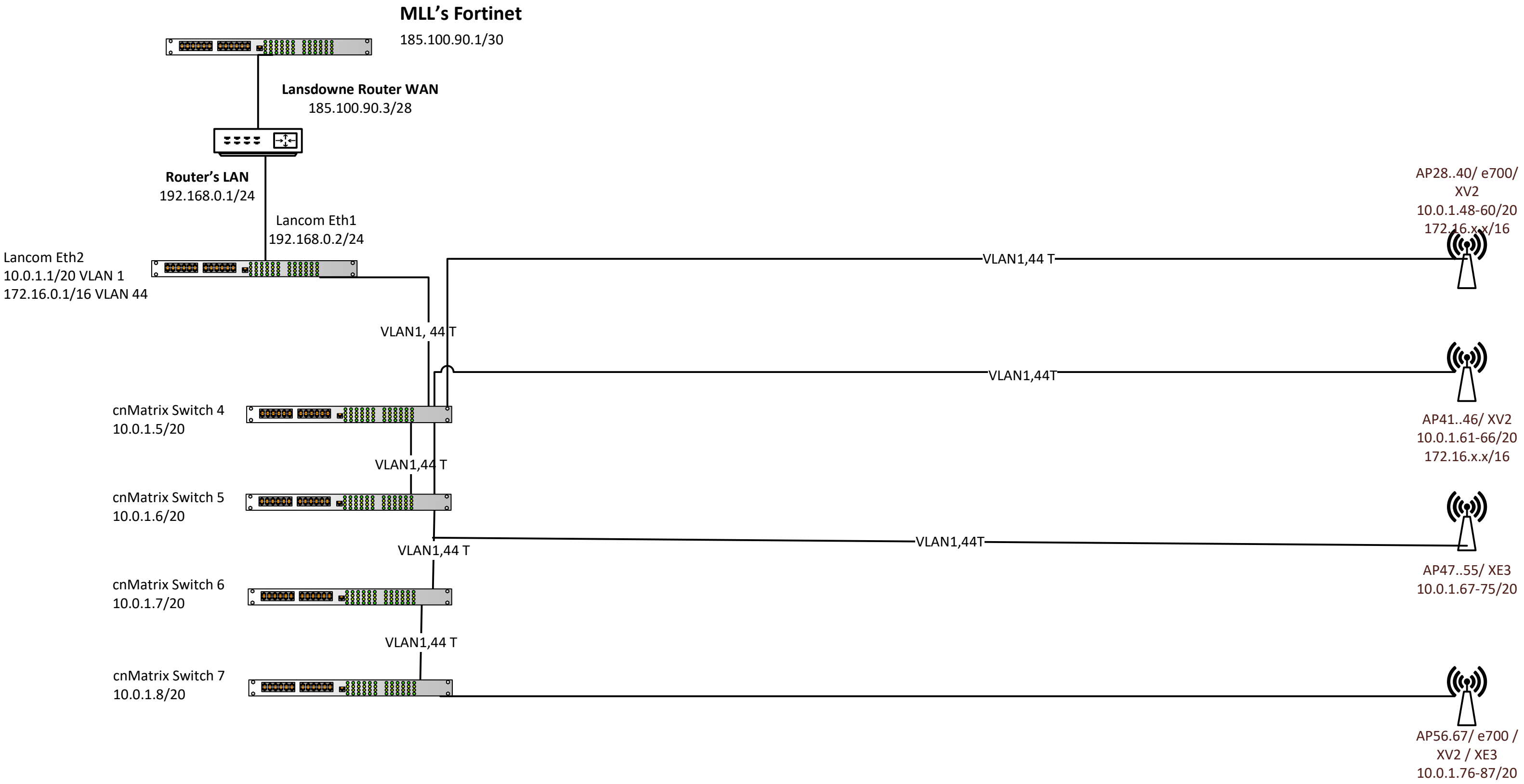


BCP Council SMART PLACE

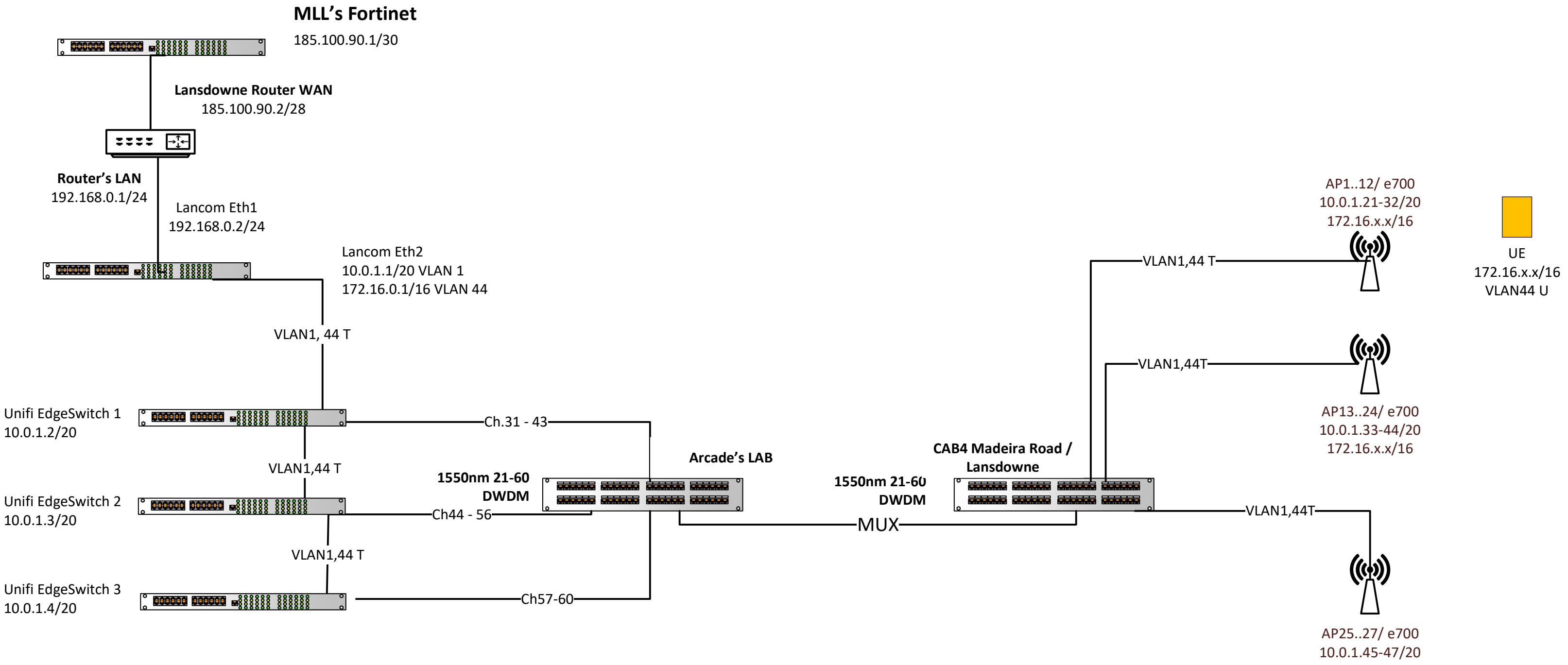
Logical topology /connectivity diagram
5G network BCP
V2.0 Draft

18/07/2024	Designed: Xavier N.	Approved: XN

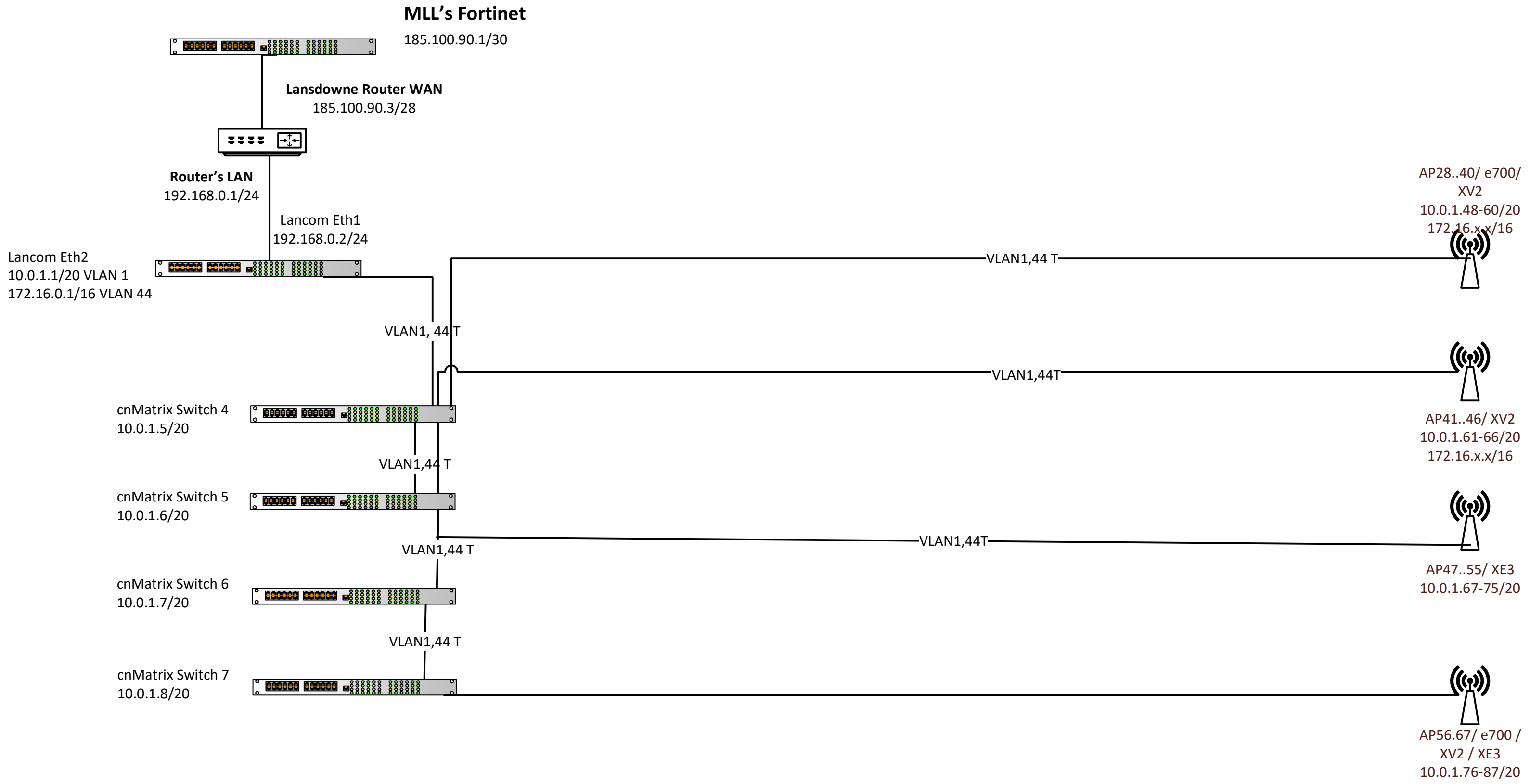
BOSCOMBE PUBLIC WI FI ARCHITECTURE



LANSDOWNE PUBLIC WI FI ARCHITECTURE



BOSCOMBE PUBLIC WI FI ARCHITECTURE


MLL's Fortinet
185.100.90.1/30
Lansdowne Router WAN
185.100.90.3/28
Router's LAN
192.168.0.1/24
Lancom Eth1
192.168.0.2/24
Lancom Eth2
10.0.1.1/20 VLAN 1
172.16.0.1/16 VLAN 44
VLAN1, 44 T
cnMatrix Switch 4
10.0.1.5/20
VLAN1,44 T
cnMatrix Switch 5
10.0.1.6/20
VLAN1,44 T
cnMatrix Switch 6
10.0.1.7/20
VLAN1,44 T
cnMatrix Switch 7
10.0.1.8/20
VLAN1,44 T
VLAN1,44T
VLAN1,44T
AP28..40/ e700/
XV2
10.0.1.48-60/20
172.16.x.x/16
AP41..46/ XV2
10.0.1.61-66/20
172.16.x.x/16
AP47..55/ XE3
10.0.1.67-75/20
AP56.67/ e700 /
XV2 / XE3
10.0.1.76-87/20