AN1268: Authenticating Silicon Labs Devices Using Device Certificates

This version of AN1268 has been deprecated with the release of Simplicity SDK Suite 2025.6.1.

For the latest version, see docs.silabs.com.

****

This application note describes how to authenticate a device as a genuine Silicon Labs product at any time during its life. The digital certificates for secure identity are stored in the device and the Silicon Labs Server.

This secure identity feature is only available on Secure Vault High devices.

KEY POINTS

- Secure identity on Secure Vault High devices
- Device certificate options
- Entity Attestation Token (EAT)
- Remote authentication process
- Examples for certificate chain verification and remote authentication

1. Series 2 Device Security Features

Protecting IoT devices against security threats is central to a quality product. Silicon Labs offers several security options to help developers build secure devices, secure application software, and secure paths of communication to manage those devices. Silicon Labs' security offerings were significantly enhanced by the introduction of the Series 2 products that included a Secure Engine. The Secure Engine is a tamper-resistant component used to securely store sensitive data and keys and to execute cryptographic functions and secure services.

On Series 1 devices, the security features are implemented by the TRNG (if available) and CRYPTO peripherals.

On Series 2 devices, the security features are implemented by the Secure Engine and CRYPTOACC (if available). The Secure Engine may be hardware-based, or virtual (software-based). Throughout this document, the following abbreviations are used:

- HSE - Hardware Secure Engine
- VSE - Virtual Secure Engine
- SE - Secure Engine (either HSE or VSE)

Additional security features are provided by Secure Vault. Three levels of Secure Vault feature support are available, depending on the part and SE implementation, as reflected in the following table:

Level (1)	SE Support	Part (2)
Secure Vault High (SVH)	HSE only (HSE-SVH)	Refer to UG103.05 for details on supporting devices.
Secure Vault Mid (SVM)	HSE (HSE-SVM)	"
"	VSE (VSE-SVM)	"
Secure Vault Base (SVB)	N/A	"

Note:

1. The features of different Secure Vault levels can be found in https://www.silabs.com/security.
2. UG103.05.

Secure Vault Mid consists of two core security functions:

- Secure Boot: Process where the initial boot phase is executed from an immutable memory (such as ROM) and where code is authenticated before being authorized for execution.
- Secure Debug access control: The ability to lock access to the debug ports for operational security, and to securely unlock them when access is required by an authorized entity.

Secure Vault High offers additional security options:

- Secure Key Storage: Protects cryptographic keys by "wrapping" or encrypting the keys using a root key known only to the HSE-SVH.
- Anti-Tamper protection: A configurable module to protect the device against tamper attacks.
- Device authentication: Functionality that uses a secure device identity certificate along with digital signatures to verify the source or target of device communications.

A Secure Engine Manager and other tools allow users to configure and control their devices both in-house during testing and manufacturing, and after the device is in the field.

1.1 User Assistance

In support of these products Silicon Labs offers whitepapers, webinars, and documentation. The following table summarizes the key security documents:

Document	Summary	Applicability
AN1190: Series 2 Secure Debug	How to lock and unlock Series 2 debug access, including background information about the SE	Secure Vault Mid and High
AN1218: Series 2 Secure Boot with RTSL	Describes the secure boot process on Series 2 devices using SE	Secure Vault Mid and High
AN1222: Production Programming of Series 2 Devices	How to program, provision, and configure security information using SE during device production	Secure Vault Mid and High
AN1247: Anti-Tamper Protection Configuration and Use	How to program, provision, and configure the anti-tamper module	Secure Vault High
AN1268: Authenticating Silicon Labs Devices using Device Certificates (this document)	How to authenticate a device using secure device certificates and signatures, at any time during the life of the product	Secure Vault High
AN1271: Secure Key Storage	How to securely “wrap” keys so they can be stored in non-volatile storage.	Secure Vault High

1.2 Key Reference

Public/Private keypairs along with other keys are used throughout Silicon Labs security implementations. Because terminology can sometimes be confusing, the following table lists the key names, their applicability, and the documentation where they are used.

Key Name	Customer Programmed	Purpose	Used in
Public Sign key (Sign Key Public)	Yes	Secure Boot binary authentication and/or OTA upgrade payload authentication	AN1218 (primary), AN1222
Public Command key (Command Key Public)	Yes	Secure Debug Unlock or Disable Tamper command authentication	AN1190 (primary), AN1222, AN1247
OTA Decryption key (GBL Decryption key) aka AES-128 Key	Yes	Decrypting GBL payloads used for firmware upgrades	AN1222 (primary), UG266/UG489
Attestation key aka Private Device Key	No	Device authentication for secure identity	AN1268

1.3 SE Firmware

Silicon Labs strongly recommends installing the latest SE firmware on Series 2 devices to support the required security features. Refer to AN1222 for the procedure to upgrade the SE firmware and UG103.05 for the latest SE Firmware shipped with Series 2 devices and modules.

2. Introduction

One of the biggest challenges for connected devices is post-deployment authentication. Silicon Labs' factory trust provisioning service with optional secure programming provides a secure device identity certificate, analogous to a birth certificate, for each individual silicon die during integrated circuit (IC) manufacturing. This enables post-deployment security, authenticity, and attestation-based health checks. The device certificate guarantees the authenticity of the device for its lifetime. When the certificate is checked, a digital signature confirms that the certificate received has not been tampered with.

Certificates can now be used to authenticate Internet of Things (IoT) devices as well as Internet servers, now that Silicon Labs' HSE-SVH devices have both cryptographic acceleration in hardware and tamper-resistant storage to handle digital certificate operations.

The digital signature and certificates are major cryptographic tools to verify the device is authentic. These tools are described in the following sections.

2.1 Digital Signature

The digital signature is used to protect integrity and authenticity of an electronic message.

Digital Signature Example:

Alice wants to give data to Bob, and Bob wants to make sure that the data came from Alice and has not been tampered with. Alice has a private/public key pair, and has previously given Bob her public key.

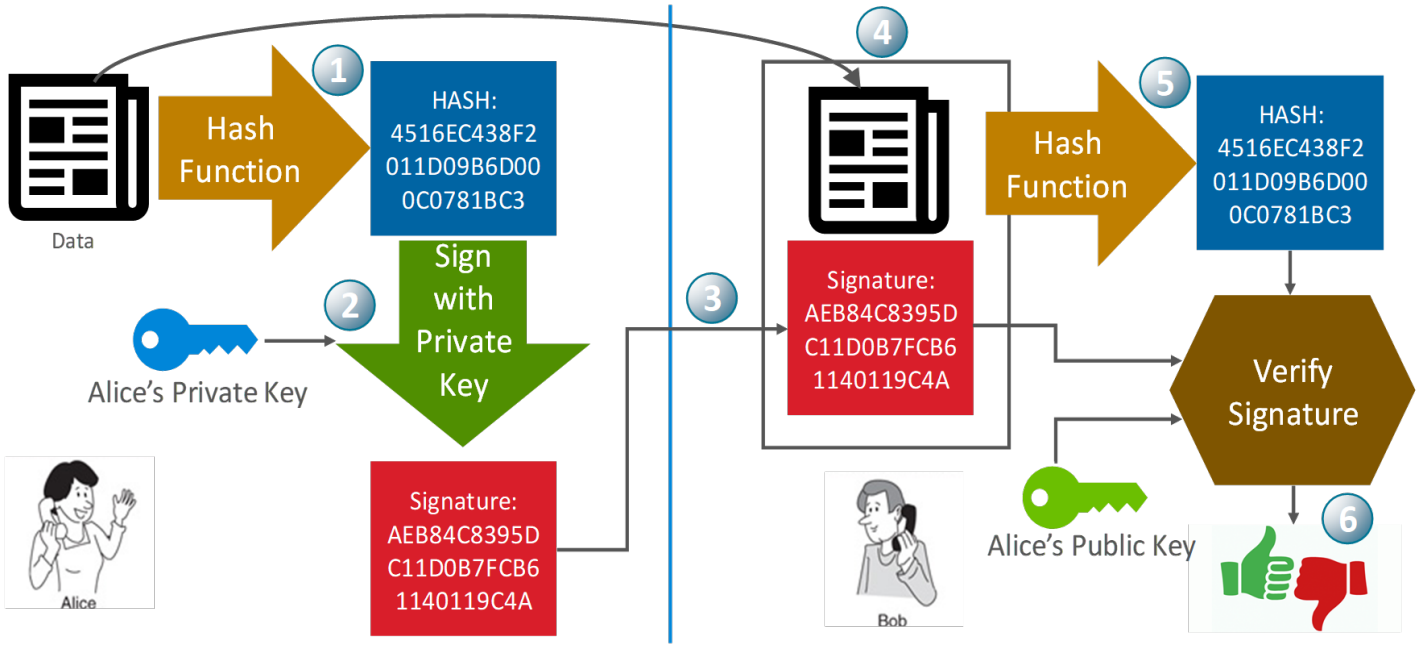


Figure 2.1. Digital Signature

1. Alice generates the hash (for example SHA256) of the data.
2. Alice's private key is used to sign the hash to create a signature. The hash is signed instead of the data itself because the signing operation is slow. Therefore it is more efficient to sign the hash instead of the arbitrarily large data.
3. The signature is attached to the end of the data.
4. The data and signature are given to Bob.
5. Bob independently generates the hash of the data.
6. The signature is verified with the hash and Alice's public key, which results in a true or false outcome indicating if the data is valid.

Note: This scheme requires distribution of Alice's public key.

2.2 Digital Certificates and Chain of Trust

In Figure 2.1 Digital Signature on page 4, Bob already had access to Alice's public key, which he trusted. However, it is not always feasible to pre-share a public key with everyone for secure identity verification, and no mechanism exists to revoke or inactivate the public key in case it gets stolen.

A digital certificate is simply a small, verifiable data file that contains identity credentials and a public key. That data is then signed either with the corresponding private key, or a different private key. The digital certificate can be used to prove the ownership of a public key.

- If it is signed using the corresponding private key, it is called a self-signed certificate.
- If it is signed by another private key, the owner of that private key is acting as a Certificate Authority (CA).
- A Certificate Authority (CA) is a trusted third party by both the owner and party relying on the certificate.

Concatenation of digital certificates builds a chain of trust.

- At the root of the chain is a self-signed certificate called a root certificate or a CA certificate.
- The root or CA certificate can be used to sign another certificate.

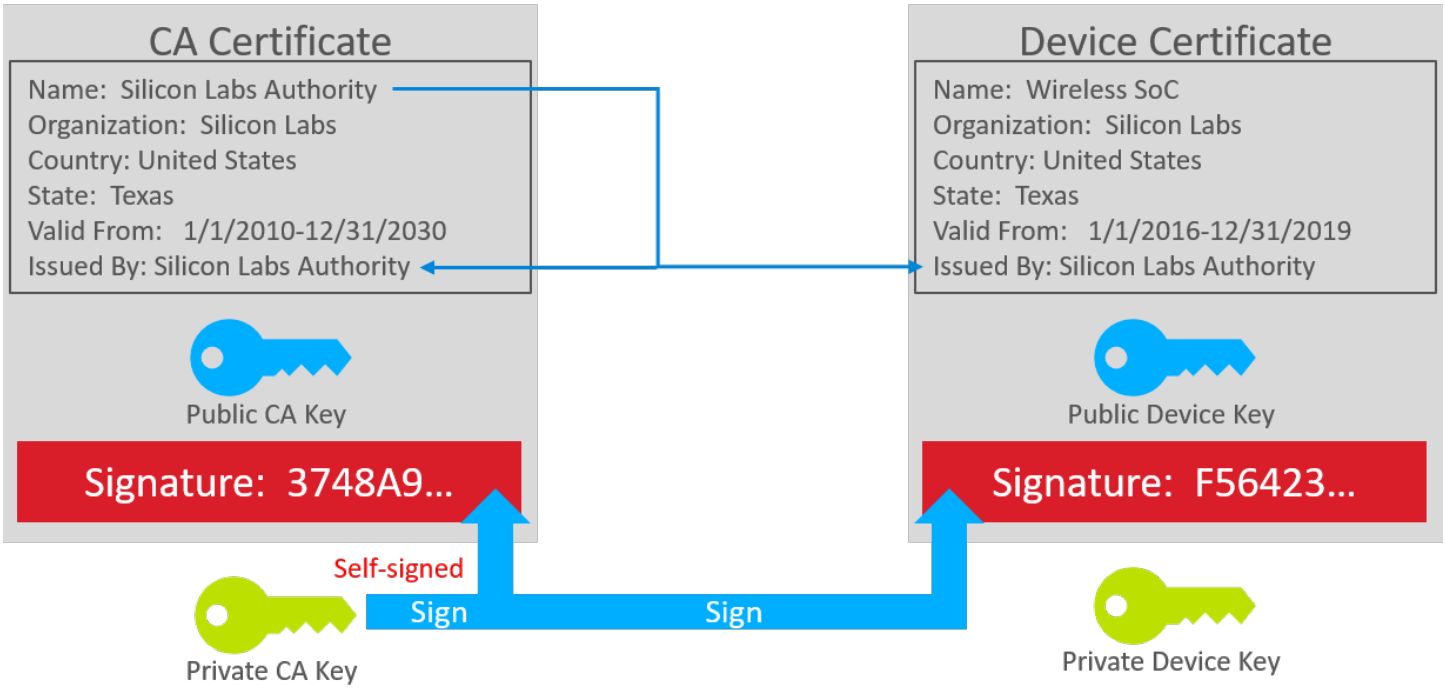


Figure 2.2. Digital Certificates and Chain of Trust

Note: The private key is never included as part of the certificate – it must be stored separately and kept private. The security of the scheme relies on protecting the private keys.

2.3 Digital Certificates Verification

This section illustrates the process shown in Figure 2.1 Digital Signature on page 4, but using digital certificates.

Digital Certificates Verification Example:

Alice wants to give data to Bob, signed with her private key. Alice has a digital certificate signed by a trusted third party (CA) in addition to her private key. Bob has a certificate from the trusted CA but nothing else is previously shared.

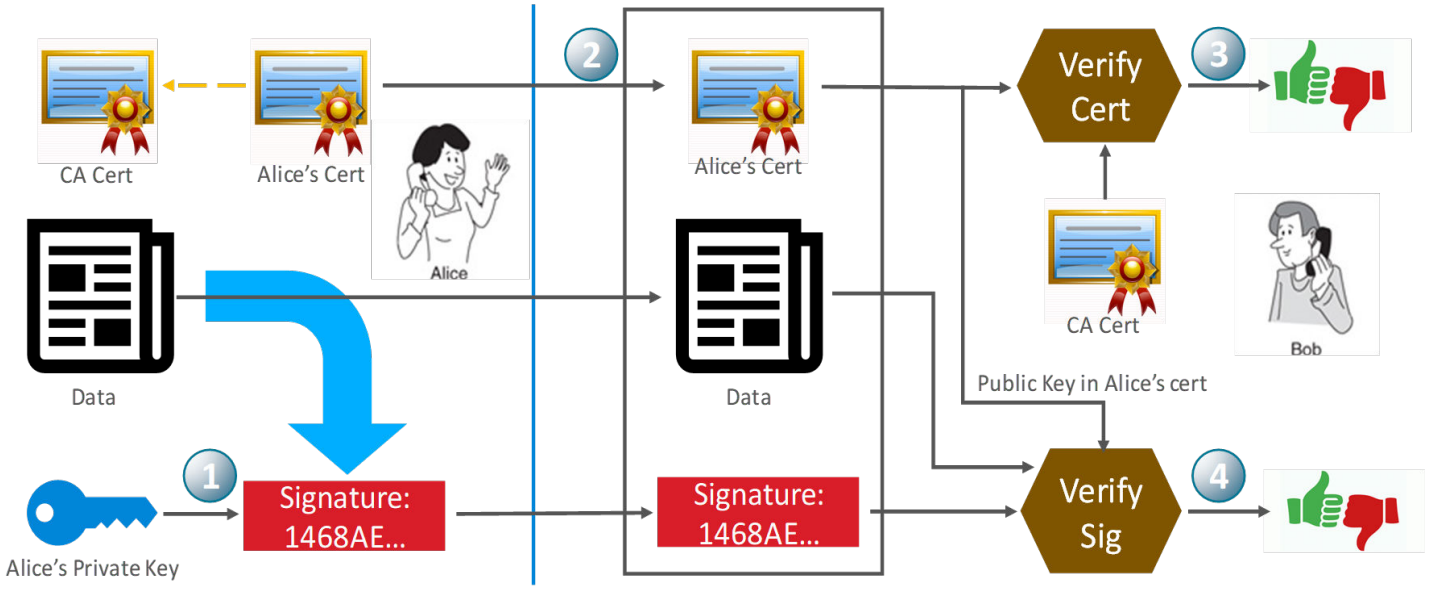


Figure 2.3. Digital Certificate Verification

1. Alice uses her private key to sign the data.
2. Alice gives the data, signature, and her certificate to Bob.
3. Bob first verifies that Alice's certificate is valid, to prove Alice is the owner of the certificate's public key. This is done by verifying that Alice's certificate contains a valid signature created by the CA.
4. Bob then verifies the signature of the data using the public key in Alice's certificate.

Note: The hash process in Figure 2.1 Digital Signature on page 4 is skipped in this example.

3. Secure Identification on HSE-SVH Devices

The goal of secure identification is to prove the ownership of a device's unique public key to an external service. It enables the external service to identify the device as legitimate and to authenticate device-generated data or messages.

3.1 Chain of Trust

The chain of trust on HSE-SVH devices is illustrated in the following figure.

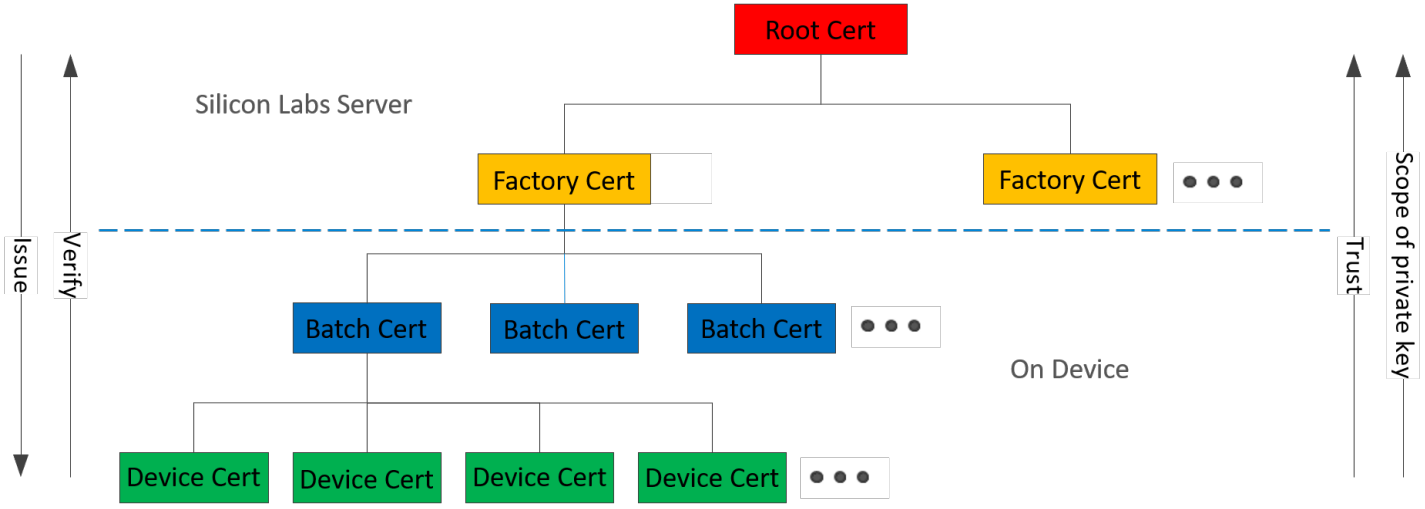


Figure 3.1. Chain of Trust

- Silicon Labs is a Certificate Authority (CA).
- The root certificate and factory certificate are stored in the Silicon Labs Server.
- The factory certificate is static per factory.
- The batch certificate and device certificate are stored on the device.
- The batch certificate is rolled per production batch.
- The device certificate is a unique cryptographic identity.
- All certificates are X.509 standard format.
  - TLS-compliant: Standard endpoint authentication methods are used in internet communications
  - Signature algorithm: ECDSA-prime256v1 with SHA256
- Each certificate in the chain is signed by the certificate above it (Figure 3.3 Signing for Certificates on page 9).

Note: A certificate can be revoked if needed, for instance if security issues arise. The certificate revocation lists are stored in the Silicon Labs Server.

3.2 Device Certificate

The device certificate example is described in the following figure.

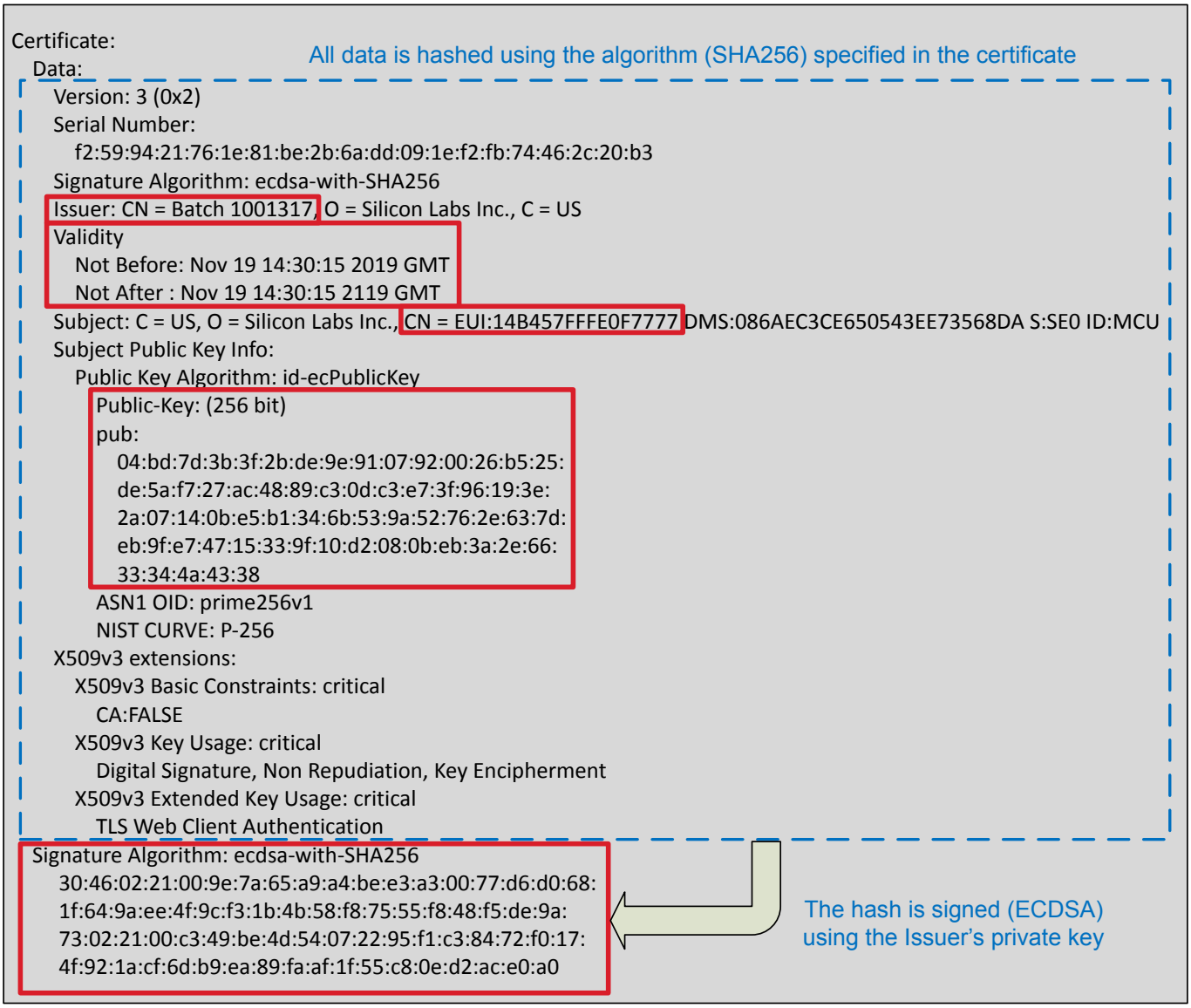


Figure 3.2. Device Certificate Example

- The device certificate is in X.509 DER format (~0.5 kB).
- The device certificate is stored in HSE one-time programmable memory (OTP). It cannot be modified once programmed.
- The batch number (`Issuer: CN = Batch` field) identifies the factory and batch in which the device was produced.
- The validity period is 100 years from device manufacture date.
- The device 64-bit hard-coded unique ID (EUI) is encoded in the `Subject: CN` field, which blinds this certificate to the device.
- The device-specific public key is embedded in the device certificate and the corresponding private key is securely stored in the Secure Key Storage on the chip.
- The Issuer's private key is used to sign the hash of the certificate data to create a device certificate signature.

3.3 Signing and Verification

Signing and verification for certificates on HSE-SVH devices are described in the following figures.

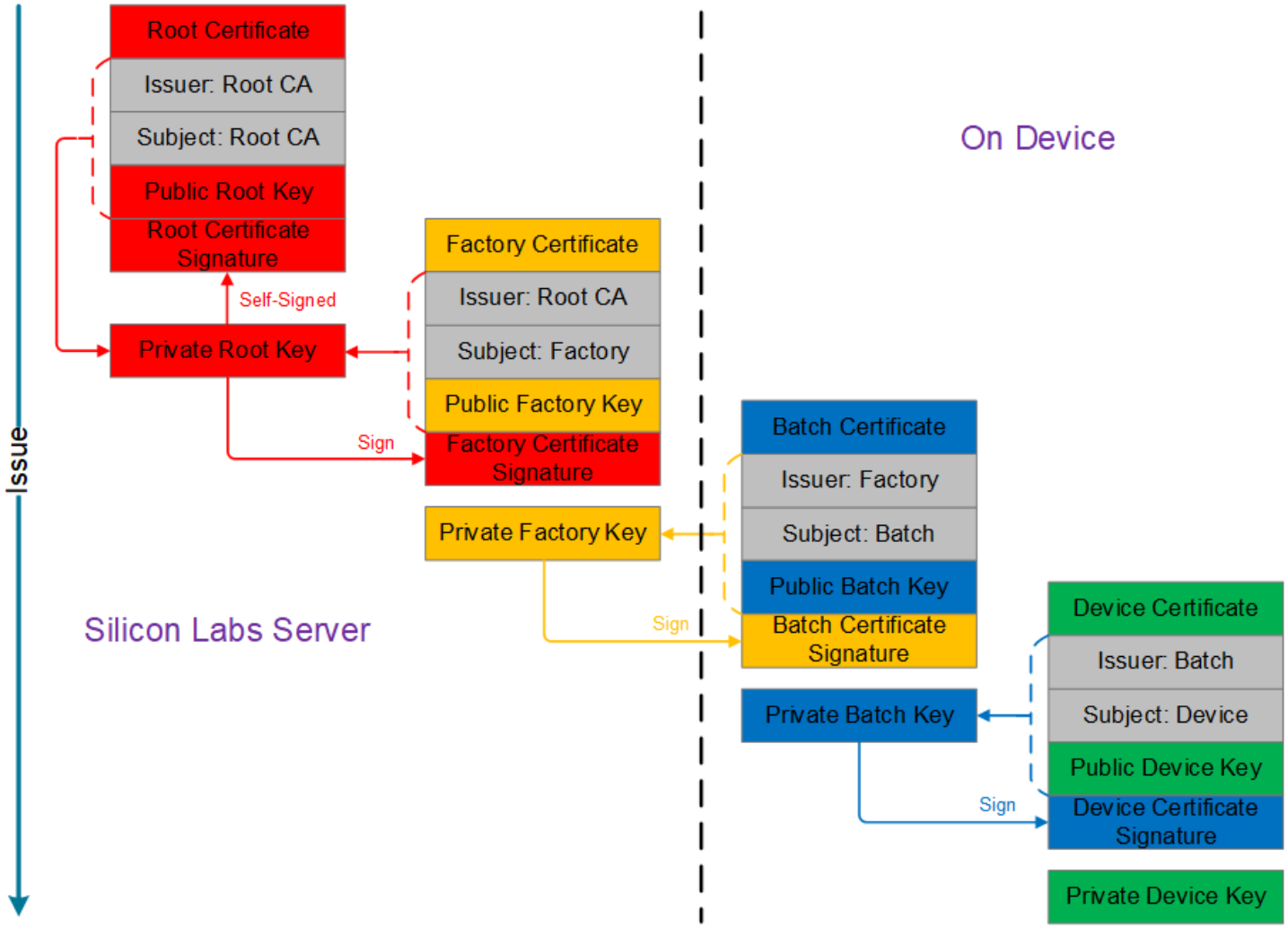


Figure 3.3. Signing for Certificates

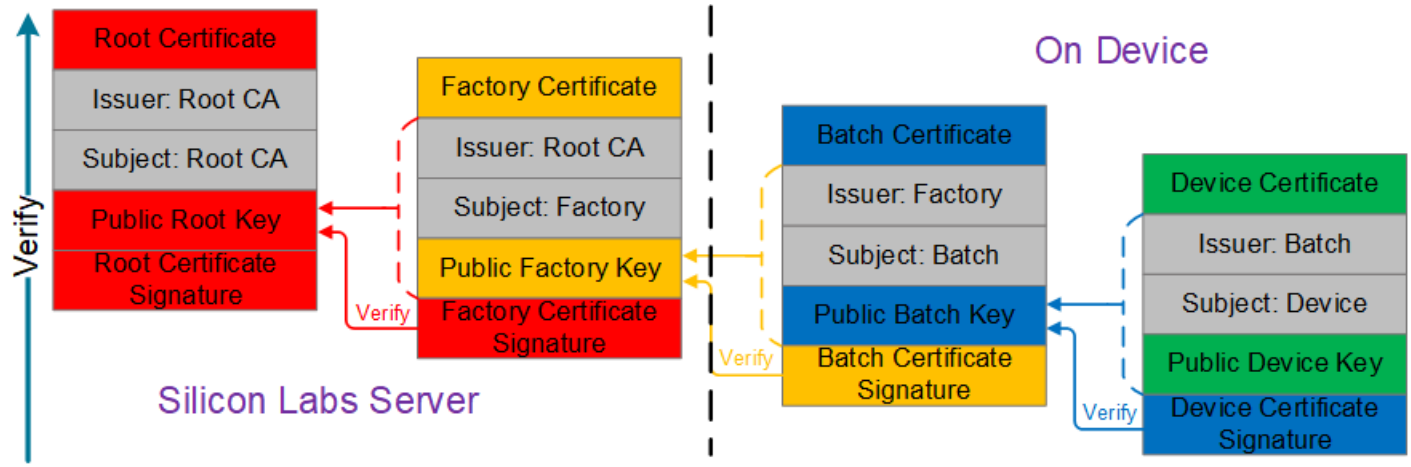


Figure 3.4. Verification for Certificates

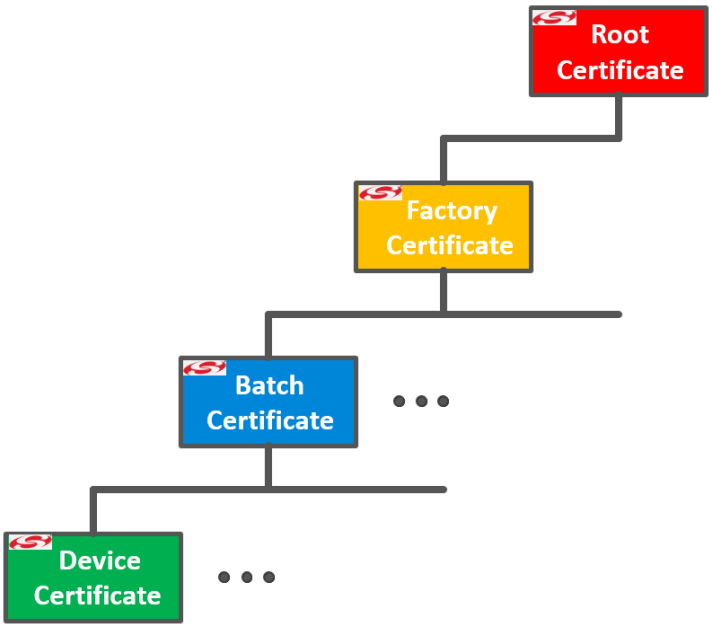
4. Device Certificate Options

The HSE-SVH devices are each programmed with a device certificate during IC production. The device certificate is signed with a Public Device Key, using a Private Batch Key that can be validated against a Silicon Labs certificate chain (Figure 3.4 Verification for Certificates on page 9 and 8.2 Certificate Chain Verification). The device private key never leaves the Secure Key Storage on the chip. Customers can create their own device certificates during their production.

Three device certificate options (standard, modified, and external) are provided to meet different requirements. Silicon Labs provides Custom Part Manufacturing Service (CPMS) to program custom certificates on your chips at the Silicon Labs factories. For more information about CPMS, see UG519: Custom Part Manufacturing Service User's Guide.

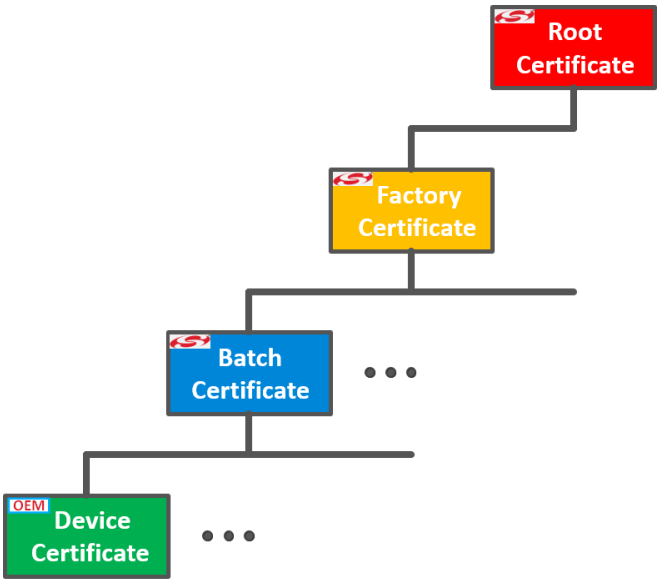
Standard Device Certificate:

- Comes standard with HSE-SVH devices.
- Cryptographically proves the device is an authentic Silicon Labs device.
- Does not protect against overproduction or counterfeit products that are built with authentic Silicon Labs devices.
- Signed to a Silicon Labs Certificate Authority (CA).
- The device can prove that it possesses the private key associated with the public key in its certificate by signing the response to a given challenge (Figure 6.1 Remote Authentication Process on page 14 and 8.3 Certificate Chain Verification and Remote Authentication).



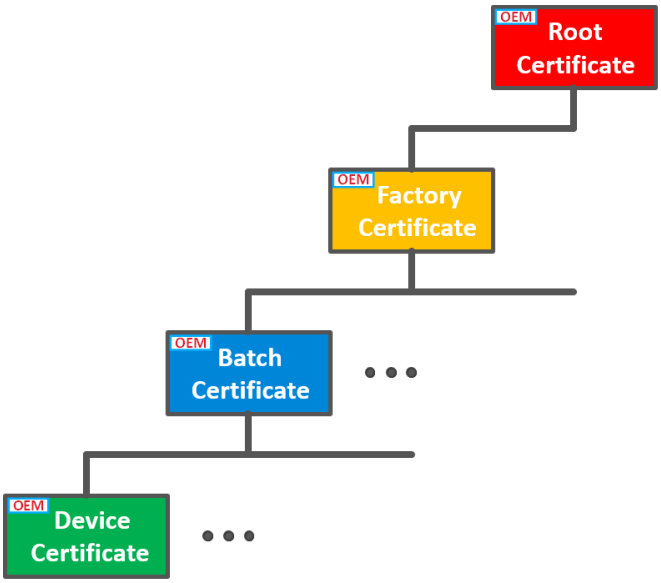
Modified Device Certificate:

- Available as a customization service on HSE-SVH devices (OEM custom part number).
- Cryptographically proves the device is an authentic Silicon Labs device that was produced for a specific OEM.
- Protects against overproduction by Contract Manufacturer (CM).
- Device Certificate X.509 fields can be specified, with restrictions.
- Signed to a Silicon Labs Certificate Authority (CA).



External Device Certificate:

- Available as a customization service on HSE-SVH devices (OEM custom part number).
- Cryptographically proves the device is an authentic Silicon Labs device that was produced for a specific OEM.
- Protects against overproduction by Contract Manufacturer (CM).
- Factory Certificate is custom for each OEM.
- Device Certificate and Factory Certificate X.509 fields can be specified, with restrictions.
- Signed to a OEM Certificate Authority (CA).
- Root Certificate Authority is OEM-specified and is optional.
- Electronic delivery of all batch and device certificates signed under this OEM factory certificate is supported.



Note: The modified and external options are subject to an additional cost. For these options, contact your local sales representative.

5. Entity Attestation Token (EAT)

The device attestation service creates a token that contains a fixed set of device-specific data when requested from the caller. The device must contain an attestation key pair, which is unique per device, to sign the token. The HSE-SVH device uses the Private Device Key (aka attestation key) to sign the token, and the caller uses the Public Device Key to verify the token's authenticity.

An Entity Attestation Token (EAT) is a mini-report that is cryptographically signed. An EAT is encoded in either one of two standardized data formats: a Concise Binary Object Representation (CBOR) or in the text-based format JSON. A digital signature is then used to protect its content. The technical specification defining the content of the EAT, which are claims about the hardware and the software running on a device, is specified by the Internet Engineering Task Force (IETF).

An EAT is a collection of Key ID-Value pairs relating to device pedigree or any other information one wants the device to attest. Collected data can originate from the Root of Trust (RoT), any protected area, or non-protected areas.

The EAT must be signed following the structure of the CBOR Object Signing and Encryption (COSE) specification. For asymmetric key algorithms, the signature structure must be COSE-Sign1. A COSE-Sign1 is a CBOR encoded, self-secured data blob that contains headers, a payload, and a signature.

The primary need for EAT verification is to check correct formatting and verify signatures as for any token. In addition, though, the verifier can operate a policy where values of some of the claims in this profile can be compared to reference values, registered with the verifier for a given deployment, to confirm that the device is endorsed by the manufacturer supply chain.

The HSE can generate the PSA attestation token or security configuration token when requested from the caller with a challenge (Auth challenge claim below). The following tables describe EAT claims that are used in the PSA attestation token and security configuration token.

Note: The actual claims returned from the tokens are HSE firmware version dependent.

Table 5.1. Claims of PSA Attestation Token

Key ID	Claim	Description	Value
-75000	Profile definition	Name of a document that describes the profile of the report.	PSA_IOT_PROFILE_1
-75001	Client ID	Represents the Partition ID of the caller.	See note below
-75002	Security lifecycle	Represent the current life cycle stage of the PSA RoT.	Device dependent
-75003	Implementation ID	Uniquely identifies the underlying immutable PSA RoT.	Device dependent (32 bytes)
-75004	Boot seed	Represents a random value created at system boot time.	Random bytes (32 bytes)
-75006	Software components	A list of Software components represents all the software loaded by PSA RoT.	See the software components table below.
-75008	Auth challenge	Input object from the caller. For example, this can be a cryptographic nonce or a hash of locally attested data. The length must be 32, 48, or 64 bytes.	Random bytes or hash (32/48/64 bytes)
-75009	Instance ID	Unique identifier of the instance.	Device EUI-64 unique ID with type byte 0x06 (9 bytes)

Note:

- Key ID 75001: Client ID if present. Otherwise the value 1 for a token requested by a secure bus master and -1 for a non-secure master.
- Key ID 75002 (For the definitions of these lifecycle states, please refer to the ARM Platform Security Model):
  - UNKNOWN (`0x0000`)
  - ASSEMBLY_AND_TEST (`0x1000`)
  - PSA_ROT_PROVISIONING (`0x2000`)
  - SECURED (`0x3000`)
  - NON_PSA_ROT_DEBUG (`0x4000`)
  - RECOVERABLE_PSA_ROT_DEBUG (`0x5000`)
  - DECOMMISSIONED (`0x6000`)

- Key ID 75003:
  - Word[0]: Die revision
  - Word[1]: HSE OTP version
  - Word[2]: Bit indicating it is an HSE-SVH device
  - Word[3]: Production version
  - Word[4:7]: Reserved (zeros)

Table 5.2. Software Components

Key ID	Type	Description	Value
1	Measurement type	A short string represents the role of this software component.	See note below
2	Measurement value	Represents a hash of the invariant software component in memory at startup time.	See note below
4	Version	The issued software version is in the form of a text string.	See note below

Note:
- Key ID 1:
  - HSE always exists — "**PRoT**"
  - If secure booted Gecko Bootloader exists at flash starting address — "**BL**"
  - If secure booted application exists at flash starting address — "**ARoT**"
- Key ID 2: SHA-256 hash (32 bytes) of the firmware (HSE, Gecko Bootloader, or application)
- Key ID 4: Version of the firmware (HSE, Gecko Bootloader, or application)

Table 5.3. Claims of Security Configuration Token

Key ID	Claim	Description	Value
-75000	Profile definition	Name of a document that describes the profile of the report.	SILABS_1
-75008	Auth challenge	Input object from the caller. For example, this can be a cryptographic nonce or a hash of locally attested data. The length must be 32 bytes.	Random bytes or hash (32 bytes)
-75009	Instance ID	Unique identifier of the instance.	Device EUI-64 unique ID with type byte 0x06 (9 bytes)
-76000	SE status	Device HSE status.	Device dependent (36 bytes)
-76001	OTP configuration	Device HSE OTP configuration if provisioned.	Device dependent (24 bytes)
-76002	Sign Key	Public Sign Key in HSE OTP if provisioned.	Device dependent (64 bytes)
-76003	Command Key	Public Command Key in HSE OTP if provisioned.	Device dependent (64 bytes)
-76004	Tamper settings	Current applied tamper settings.	Device dependent (16 bytes)

Note:
- All custom Silicon Labs claims will have a base of 76000.
- Key ID 76000: Refer to section "*Get Status*" in AN1303: Programming Series 2 Devices using the Debug Challenge Interface (DCI) and Serial Wire Debug (SWD) for the description (HSE-SVH) of the value.
- Key ID 76001: Refer to section "*Read User Configuration*" in AN1303: Programming Series 2 Devices using the Debug Challenge Interface (DCI) and Serial Wire Debug (SWD) for the description (HSE-SVH) of the value.
- Key ID 76002 and 76003: Refer to 1.2 Key Reference for Public Sign Key and Public Command Key.
- Key ID 76004: One nibble per tamper source. Refer to section "*Anti-Tamper Configuration*" in AN1303: Programming Series 2 Devices using the Debug Challenge Interface (DCI) and Serial Wire Debug (SWD) for the description of the value.

6. Remote Authentication Process

Remote authentication is used to manage attestation by requesting that the device sign a challenge or EAT based on its secure identity.

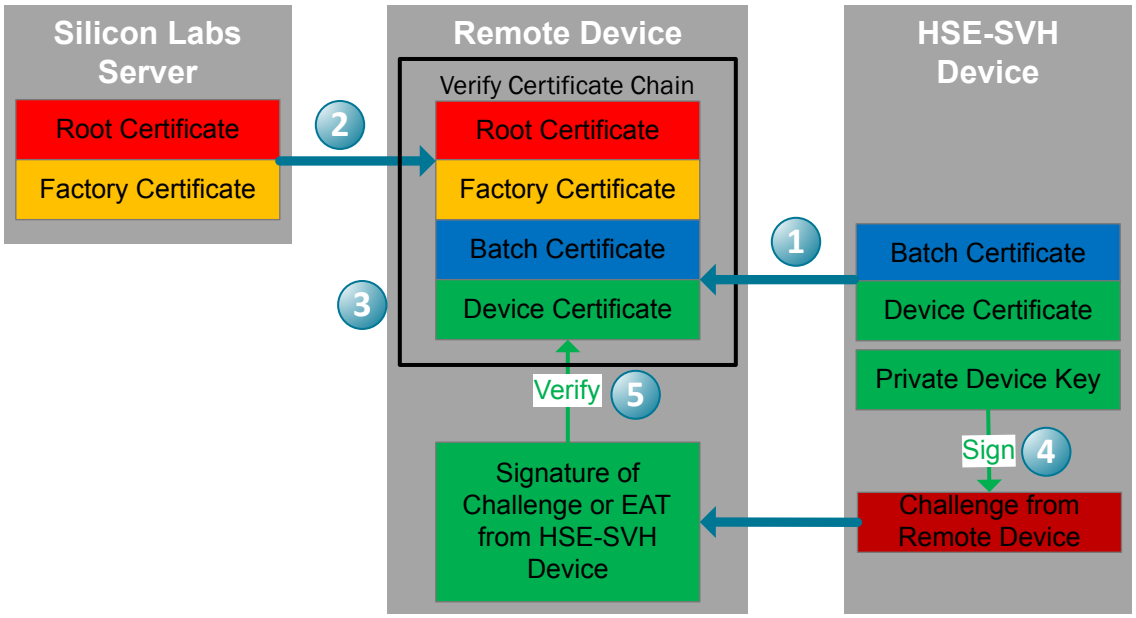


Figure 6.1. Remote Authentication Process

1. The remote device requests the device certificate and batch certificate from the HSE-SVH device.
2. The remote device looks up the factory certificate and root certificate from the Silicon Labs Server.
3. The remote device validates each certificate in the chain using the public key of each Issuer (Figure 3.4 Verification for Certificates on page 9).
4. The remote device then sends an attestation challenge (random number) to the HSE-SVH device. The HSE-SVH device uses the Private Device Key in the Secure Key Storage on the chip to sign the challenge or EAT and sends the signature of challenge or EAT to the remote device.
5. The remote device requires a small library to validate the signature of challenge or EAT using the Public Device Key in the device certificate.

7. Secure Engine Manager

The Secure Engine Manager provides thread-safe APIs for the SE's mailbox interface. The following table lists the SE Manager APIs related to secure identity. The SE Manager API document can be found at https://docs.silabs.com/gecko-platform/latest/service/api/group-sl-se-manager.

For the SE's mailbox interface, see section "*Secure Engine Subsystem*" in AN1190: Series 2 Secure Debug.

Table 7.1. SE Manager API for Security Identity

SE Manager API	Usage
sl_se_read_pubkey	Read stored Public Device Key in the HSE-SVH device.
sl_se_read_cert	Read stored certificates (DER format) in the HSE-SVH device.
sl_se_read_cert_size	Read the size of stored certificates in the HSE-SVH device.
sl_se_attestation_get_psa_iat_token	Get the PSA attestation token from the HSE with the given nonce.
sl_se_attestation_get_psa_iat_token_size	Get the size of a PSA attestation token with the given nonce.
sl_se_attestation_get_config_token	Get the security configuration token from the HSE with the given nonce.
sl_se_attestation_get_config_token_size	Get the size of a security configuration token with the given nonce.

8. Examples

8.1 Overview

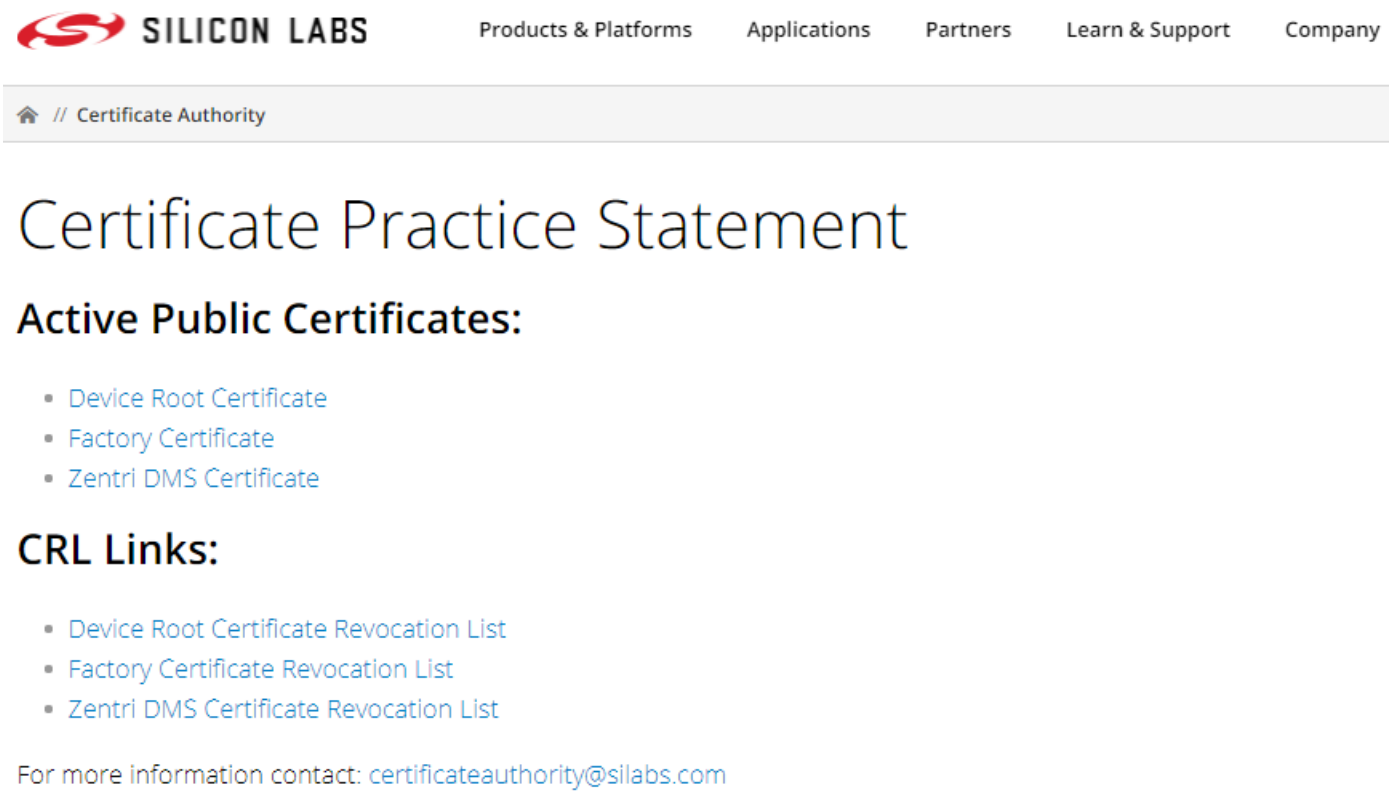
The secure device authentication examples are described in the following table.

Table 8.1. Secure Device Authentication Examples

Example	Device (Radio Board)	HSE Firmware	Tool
Certificate chain verification	EFR32MG21B010F1024IM32 (BRD4181C)	Version 1.2.9	Simplicity Commander and OpenSSL
"	EFR32MG21B010F1024IM32 (BRD4181C)	Version 1.2.9	Simplicity Commander
"	EFR32MG21B010F1024IM32 (BRD4181C)	Version 1.2.9	Simplicity Studio 5
Certificate chain verification & Remote authentication	EFR32MG21B010F1024IM32 (BRD4181C)	Version 1.2.9	SE Manager and Mbed TLS
Entity Attestation Token (EAT)	EFR32MG21B010F1024IM32 (BRD4181C)	Version 1.2.9	SE Manager
"	EFR32MG21B010F1024IM32 (BRD4181C)	Version 1.2.9	Simplicity Commander

Note: Unless specified in the example, these examples can apply to other HSE-SVH devices.

Users can download the device root certificate (`Device-Root-CA-chain.pem`) and factory certificate (`Factory-chain.pem`) from https://www.silabs.com/certificate-authority.



For Simplicity Studio v5.3.0.0 and higher, the device root certificate (`device-root-prod.pem`) and factory certificate (`factory-prod.pem`) can be found in the Window folder below.

```
C:\SiliconLabs\SimplicityStudio\v5\offline\common\certificates
```

8.1.1 Using Simplicity Commander

1. This application note uses Simplicity Commander v1.11.2. The procedures and console output may be different on the other versions of Simplicity Commander. The latest version of Simplicity Commander can be downloaded from https://www.silabs.com/developers/mcu-programming-options.

```
commander --version
```

```
Simplicity Commander 1v11p2b998

JLink DLL version: 6.94d
Qt 5.12.1 Copyright (C) 2017 The Qt Company Ltd.
EMDLL Version: 0v17p18b581
mbed TLS version: 2.6.1


DONE
```

2. The Simplicity Commander's Command Line Interface (CLI) is invoked by `commander.exe` in the Simplicity Commander folder. The location for Simplicity Studio 5 in Windows is `C:\SiliconLabs\SimplicityStudio\v5\developer\adapter_packs\commander`. For ease of use, it is highly recommended to add the path of `commander.exe` to the system `PATH` in Windows.
3. If more than one Wireless Starter Kit (WSTK) is connected via USB, the target WSTK must be specified using the `--serialno <J-Link serial number>` option.
4. If the WSTK is in debug mode OUT, the target device must be specified using the `--device <device name>` option.

For more information about Simplicity Commander, see UG162: Simplicity Commander Reference Guide.

8.1.2 Using an External Tool

The certificate chain verification example uses the **OpenSSL** to validate the certificate chain. The Windows version of OpenSSL can be downloaded from https://slproweb.com/products/Win32OpenSSL.html. This application note uses OpenSSL Version 1.1.1h (Win64).

```
openssl version
```

```
OpenSSL 1.1.1h  22 Sep 2020
```

The OpenSSL's Command Line Interface (CLI) is invoked by `openssl.exe` in the OpenSSL folder. The location in Windows (Win64) is `C:\Program Files\OpenSSL-Win64\bin`. For ease of use, it is highly recommended to add the path of `openssl.exe` to the system `PATH` in Windows.

8.1.3 Using Platform Examples

Simplicity Studio 5 includes the SE Manager platform examples for secure identity and attestation. This application note uses platform example of GSDK v3.2.3. The console output may be different on the other versions of GSDK.

Refer to the corresponding `readme` file for details about each SE Manager platform example. This file also includes the procedures to create the project and run the example.

8.2 Certificate Chain Verification

Certificate chain verification is the process of making sure a given certificate chain is well-formed, valid, properly signed, and trustworthy. The certificate signature is verified using the public key in the issuer certificate (Figure 3.4 Verification for Certificates on page 9).

8.2.1 Simplicity Commander and OpenSSL

1. Run the `security readcert` command to save the batch certificate in PEM format.

```
commander security readcert batch -o batch.pem --serialno 440030580
```

```
Writing certificate to batch.pem...
DONE
```

2. Run the `security readcert` command to save the device certificate in PEM format.

```
commander security readcert mcu -o device.pem --serialno 440030580
```

```
Writing certificate to device.pem...
DONE
```

3. Get the root certificate (`device-root-prod.pem`) and factory certificate (`factory-prod.pem`) from the certificate folder in Simplicity Studio.

4. Use OpenSSL to display the certificate information (e.g., `device.pem`).

```
openssl x509 -in device.pem -text -noout
```

```
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number:
            66:f8:5a:e6:b4:ef:6e:49:d3:36:95:63:c9:c3:99:13:e4:71:93:f6
        Signature Algorithm: ecdsa-with-SHA256
        Issuer: CN = Batch 1001317, O = Silicon Labs Inc., C = US
        Validity
            Not Before: Nov 19 15:10:33 2019 GMT
            Not After : Nov 19 15:10:33 2119 GMT
        Subject: C = US, O = Silicon Labs Inc., CN = EUI:14B457FFFE0F77CE DMS:086AEC3C645836BFB04D312F S:SE0 ID:MCU
        Subject Public Key Info:
            Public Key Algorithm: id-ecPublicKey
                Public-Key: (256 bit)
                pub:
                    04:5c:4b:c9:b0:b3:ff:fa:99:81:c5:99:be:ff:ae:
                    77:74:1a:f4:30:f1:1e:0e:2d:df:96:4b:ff:d2:46:
                    fa:fa:e7:23:4b:79:cb:0a:c7:71:13:fa:7c:39:5f:
                    e2:18:9e:4e:06:43:88:a7:9c:65:53:f3:a3:a1:06:
                    81:e6:06:f2:11
                ASN1 OID: prime256v1
                NIST CURVE: P-256
        X509v3 extensions:
            X509v3 Basic Constraints: critical
                CA:FALSE
            X509v3 Key Usage: critical
                Digital Signature, Non Repudiation, Key Encipherment
            X509v3 Extended Key Usage: critical
                TLS Web Client Authentication
    Signature Algorithm: ecdsa-with-SHA256
         30:44:02:20:57:12:a4:84:d8:37:b8:c0:44:8f:16:ac:c1:a3:
         be:a9:f1:16:38:9f:b9:a2:57:e6:12:49:bf:96:a9:a9:d2:b8:
         02:20:5f:ae:22:f5:00:05:49:b1:da:ee:4a:84:48:70:27:97:
         1c:40:2d:85:5f:f2:12:b3:8b:4a:d7:9a:ee:60:81:7c
```

5. Use OpenSSL to verify the certificate chain from steps 1 to 3.

```
openssl verify -show_chain -CAfile device-root-prod.pem -untrusted factory-prod.pem -untrusted batch.pem device.pem
```

```
device.pem: OK
Chain:
depth=0: C = US, O = Silicon Labs Inc., CN = EUI:14B457FFFE0F7777 DMS:086AEC3CE650543EE73568DA S:SE0 ID:MCU (untrusted)
depth=1: CN = Batch 1001317, O = Silicon Labs Inc., C = US (untrusted)
depth=2: CN = Factory, O = Silicon Labs Inc., C = US (untrusted)
depth=3: CN = Device Root CA, O = Silicon Labs Inc., C = US
```

8.2.2 Simplicity Commander

Run the `security readcert` command to display the key information about the on-chip certificates (e.g., `mcu`).

```
commander security readcert mcu --serialno 440030580
```

```
Version            : 3
Subject            : C=US  O=Silicon Labs Inc.  CN=EUI:14B457FFFE0F77CE DMS:086AEC3C645836BFB04D312F S:SE0 ID:MCU
Issuer             : CN=Batch 1001317  O=Silicon Labs Inc.  C=US
Valid From         : November 19 2019
Valid To           : November 19 2119
Signature algorithm: SHA256
Public Key Type    : ECDSA
Public key         : 5c4bc9b0b3fffa9981c599beffae77741af430f11e0e2ddf964bffd246fafae7
                     234b79cb0ac77113fa7c395fe2189e4e064388a79c6553f3a3a10681e606f211
DONE
```

Run the `security attestation` command to verify the on-chip batch and device certificates with root and factory certificates.

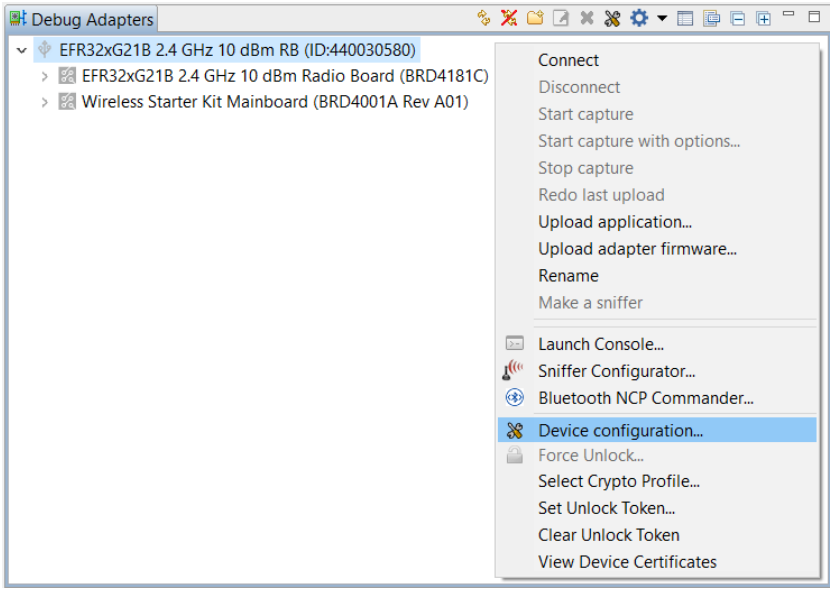
```
commander security attestation --serialno 440030580
```

```
Certificate chain successfully validated up to Silicon Labs device root certificate.

-75008 ARM PSA nonce                  : 05a88aeef627dd663058e3d758fe9a827942da0793da72af81c79a4f60fa9824
-75000 ARM PSA Profile ID             : SILABS_1
-75009 ARM PSA/IETF EAT UEID          : 0614b457fffe0f77ce
-76000 SE status                      : 00000001000000000000000000000003a90000002000010209ffffffff0000002500000000
-76001 OTP configuration              : 00000000100444400401041411224477242204420a060005
-76002 MCU sign key                   : c4af4ac69aab9512db50f7a26ae5b4801183d85417e729a56da974f4e08a562c
                                        de6019dea9411332dc1a743372d170b436238a34597c410ea177024de20fc819
-76003 MCU command key                : b1bc6f6fa56640ed522b2ee0f5b3cf7e5d48f60be8148f0dc08440f0a4e1dca4
                                        7c04119ed6a1be31b7707e5f9d001a659a051003e95e1b936f05c37ea793ad63
-76004 Current applied tamper settings : 150444400401041411224477142204 42

Successfully validated signature of attestation token.
DONE
```

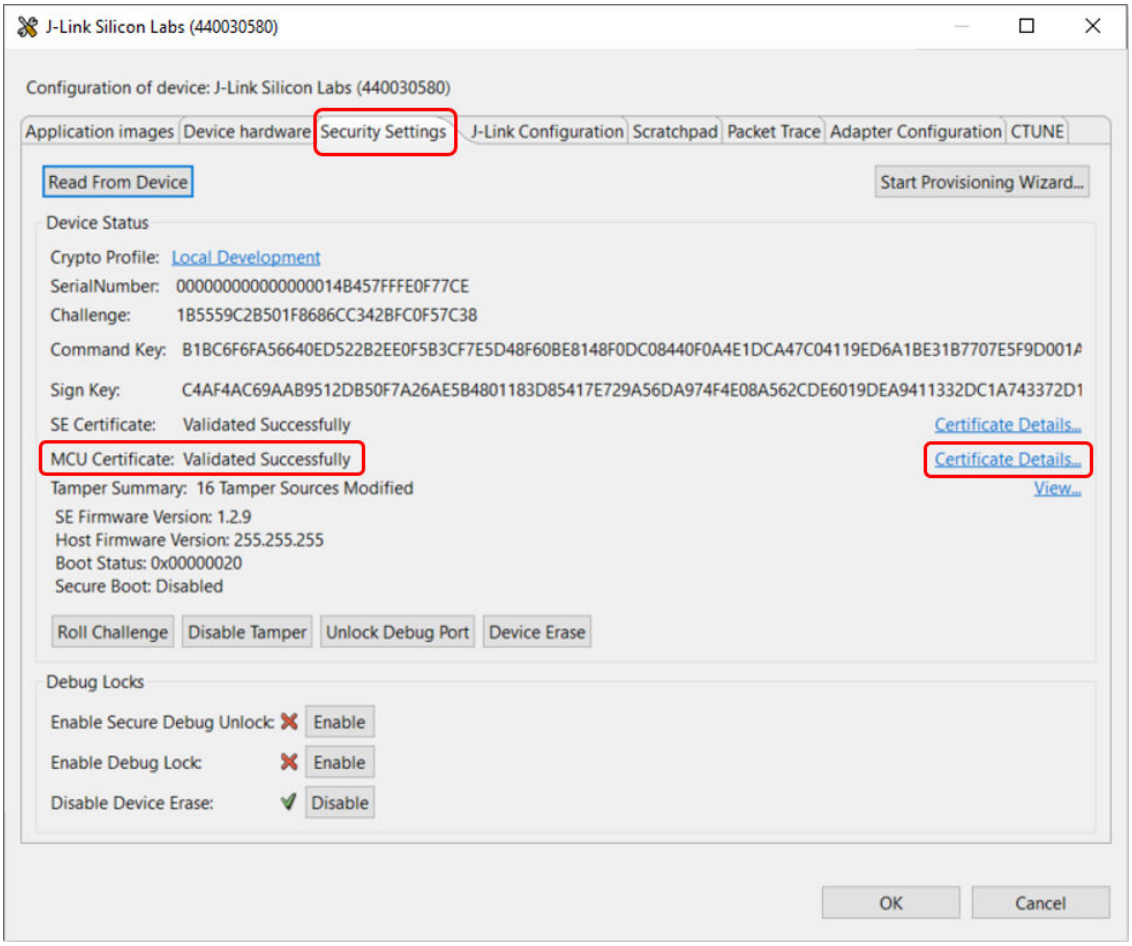
8.2.3 Simplicity Studio

This application note uses Simplicity Studio v5.2.1.1. The procedures and pictures may be different on the other versions of Simplicity Studio 5.

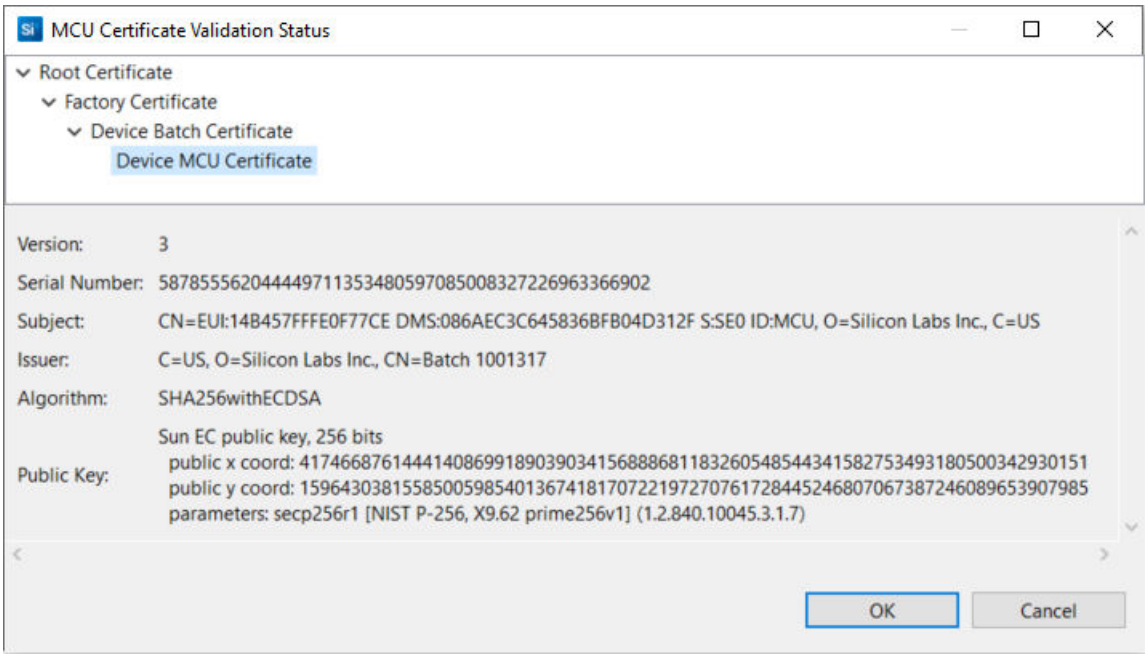
1. Right-click the selected debug adapter **RB (ID:J-Link serial number)** to display the context menu.



2. Click **Device configuration...** to open the **Configuration of device: J-Link Silicon Labs (serial number)** dialog box. Click the **Security Settings** tab to get the selected device configuration.
3. The **MCU Certificate:** will display **Validated Successfully** if it passed the certificate chain verification process.



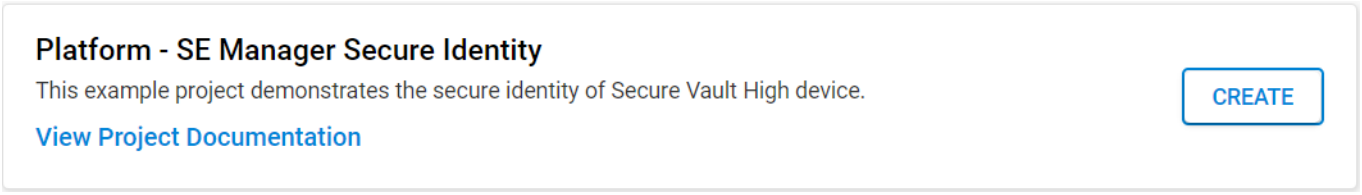
4. Click **Certificate Details...** to browse the details of different certificates (e.g., Device MCU Certificate in the figure below).



8.3 Certificate Chain Verification and Remote Authentication

The SE Manager Secure Identity platform example uses APIs in SE Manager and Mbed TLS to emulate the processes in Figure 6.1 Remote Authentication Process on page 14.

Click the `View Project Documentation` link to open the `readme` file.



The HSE-SVH device simulates the operations in the remote device to eliminate the communications between different parties in this example. The factory certificate and root certificate are hard-coded in the `app_mbedtls_x509.c` file.

The Private Device Key in the Secure Key Storage on the chip is used to sign the challenge from the remote device. Therefore this example can only run on a chip with the Standard Device Certificate.

Step 1 in the Remote Authentication Process:

```
SE Manager Secure Identity Example - Core running at 38000 kHz.
  . SE manager initialization... SL_STATUS_OK (cycles: 6 time: 0 us)

  . Secure Vault High device:
  + Read size of on-chip certificates... SL_STATUS_OK (cycles: 5296 time: 139 us)
  + Read on-chip device certificate... SL_STATUS_OK (cycles: 5138 time: 135 us)
  + Parse the device certificate (DER format)... SL_STATUS_OK (cycles: 167043 time: 4395 us)
  + Get the public device key in device certificate... OK
  + Read on-chip batch certificate... SL_STATUS_OK (cycles: 5080 time: 133 us)
  + Parse the batch certificate (DER format)... SL_STATUS_OK (cycles: 173151 time: 4556 us)
```

Steps 2 and 3 in the Remote Authentication Process (certificate chain printout is disabled):

```
  . Remote device:
  + Parse the factory certificate (PEM format)... SL_STATUS_OK (cycles: 5373122 time: 141 ms)
  + Parse the root certificate (PEM format)... SL_STATUS_OK (cycles: 5448802 time: 143 ms)
  + Verify the certificate chain with root certificate... SL_STATUS_OK (cycles: 958730 time: 25229 us)
```

Steps 2 and 3 in the Remote Authentication Process (certificate chain printout is enabled):

```
. Remote device:
+ Parse the factory certificate (PEM format)... SL_STATUS_OK (cycles: 5373935 time: 141 ms)
+ Parse the root certificate (PEM format)... SL_STATUS_OK (cycles: 5449622 time: 143 ms)
+ Verify requested for (Depth 3) ... OK
    cert. version     : 3
    serial number     : 12:E6:A2:A5:9C:AA:27:F9
    issuer name       : CN=Device Root CA, O=Silicon Labs Inc., C=US
    subject name      : CN=Device Root CA, O=Silicon Labs Inc., C=US
    issued  on        : 2018-10-10 17:32:00
    expires on        : 2118-09-16 17:32:00
    signed using      : ECDSA with SHA256
    EC key size       : 256 bits
    basic constraints : CA=true, max_pathlen=2
    key usage         : Digital Signature, Key Cert Sign, CRL Sign
+ Verify requested for (Depth 2) ... OK
    cert. version     : 3
    serial number     : 24:DC:7B:40:0C:32:9C:0A
    issuer name       : CN=Device Root CA, O=Silicon Labs Inc., C=US
    subject name      : CN=Factory, O=Silicon Labs Inc., C=US
    issued  on        : 2018-10-10 17:33:00
    expires on        : 2118-09-16 17:32:00
    signed using      : ECDSA with SHA256
    EC key size       : 256 bits
    basic constraints : CA=true, max_pathlen=1
    key usage         : Digital Signature, Key Cert Sign, CRL Sign
+ Verify requested for (Depth 1) ... OK
    cert. version     : 3
    serial number     : 23:09:DA:39:B4:78:05:AA
    issuer name       : CN=Factory, O=Silicon Labs Inc., C=US
    subject name      : CN=Batch 1001317, O=Silicon Labs Inc., C=US
    issued  on        : 2019-10-17 21:20:20
    expires on        : 2118-09-16 17:32:00
    signed using      : ECDSA with SHA256
    EC key size       : 256 bits
    basic constraints : CA=true, max_pathlen=0
    key usage         : Digital Signature, Key Cert Sign
+ Verify requested for (Depth 0) ... OK
    cert. version     : 3
    serial number     : 66:F8:5A:E6:B4:EF:6E:49:D3:36:95:63:C9:C3:99:13:E4:71:93:F6
    issuer name       : CN=Batch 1001317, O=Silicon Labs Inc., C=US
    subject name      : C=US, O=Silicon Labs Inc., CN=EUI:14B457FFFE0F77CE DMS:086AEC3C645836BFB04D312F S:SE0 ID:MCU
    issued  on        : 2019-11-19 15:10:33
    expires on        : 2119-11-19 15:10:33
    signed using      : ECDSA with SHA256
    EC key size       : 256 bits
    basic constraints : CA=false
    key usage         : Digital Signature, Non Repudiation, Key Encipherment
    ext key usage     : TLS Web Client Authentication
+ Verify the certificate chain with root certificate... SL_STATUS_OK (cycles: 9703861 time: 255 ms)
```

Note: The longer processing time (255 ms) is due to the certificate chain printout.

Steps 4 and 5 (signature of a challenge) in the Remote Authentication Process:

```
. Remote authentication:
+ Create a 16 bytes challenge (random number) in remote device for signing... SL_STATUS_OK (cycles: 3700 time: 97 us)
+ Sign challenge with private device key in Secure Vault High device... SL_STATUS_OK (cycles: 221983 time: 5841 us)
+ Get public device key in Secure Vault High device... SL_STATUS_OK (cycles: 199788 time: 5257 us)
+ Verify signature with public device key in Secure Vault High device... SL_STATUS_OK (cycles: 229054 time: 6027 us)
+ Verify signature with public device key in remote device... SL_STATUS_OK (cycles: 230442 time: 6064 us)

. SE manager deinitialization... SL_STATUS_OK (cycles: 6 time: 0 us)
```

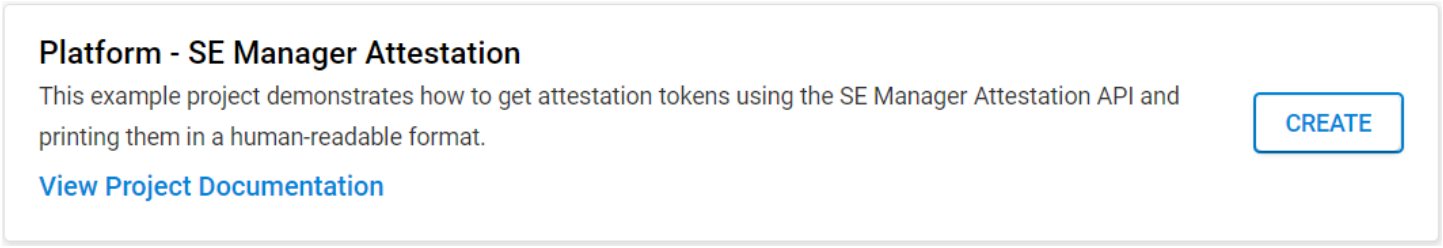
8.4 Entity Attestation Token (EAT)

These examples demonstrate how to retrieve the EAT tokens from the HSE-SVH device.

8.4.1 SE Manager - Attestation Platform Example

The SE Manager Attestation platform example uses APIs in SE Manager to retrieve the PSA attestation token and security configuration token from the HSE.

Click the `View Project Documentation` link to open the `readme` file.



Press `SPACE` to cycle the challenge size for the PSA attestation token. Press `ENTER` to make a selection and run the program.

```
SE Manager Attestation Example - Core running at 38000 kHz.
Initializing SE Manager...
SL_STATUS_OK (cycles: 10 time: 0 us)

Select nonce size for the IAT token (32, 48 or 64 bytes).
Press SPACE to cycle through the options.
Press ENTER to make a selection.
    Current nonce size:  32
    Selected nonce size: 32

Calling sl_se_attestation_get_psa_iat_token...
SL_STATUS_OK (cycles: 661072 time: 17396 us)
```

PSA Attestation Token (Table 5.1 Claims of PSA Attestation Token on page 12 and Table 5.2 Software Components on page 13)

```
PSA IAT token
=============
----------------------------------------------------------------------
Raw token:
d28443a10126a058e4a83a000124ff58204ca14d0bc8601cad2e511de1964e93
9338b6fc20f8231aa178ca79519b0ffae73a000124f7715053415f494f545f50
524f46494c455f313a00012500490614b457fffe0f77ce3a000124f8013a0001
24f91920003a000124fa5820011c000106000000001000000f2030f0000000000
0000000000000000000000003a000124fb58204922b7bbd31c0c81c9b0485ccf
b5396ec24ffa877ece441e11c947b791218cf83a000124fd81a3016450526f54
046830303031303230390258206d39caedba129297062b820ba6d85b3e432c44
3c8a8a31d3c6232be6906d38dc584030f9d61523204793965fc9eb2be788db9d
2b02692d877673c86ebffbfb6769984515d2f1a287a92d2c134c1024f20f018d
be952a2ccae7ed2980a9f242d02c9c

COSE_Sign1 structure:
d2    ; tag(18)
  84    ; array(4)
    43    ; byte_str(3)
      a10126
    a0    ; map(0)
    58    ; byte_str(228)
      a83a000124ff58204ca14d0bc8601cad2e511de1964e939338b6fc20f8231aa1
      78ca79519b0ffae73a000124f7715053415f494f545f50524f46494c455f313a
      00012500490614b457fffe0f77ce3a000124f8013a000124f91920003a000124
      fa5820011c000106000000001000000f2030f0000000000000000000000000000
      0000003a000124fb58204922b7bbd31c0c81c9b0485ccfb5396ec24ffa877ece
      441e11c947b791218cf83a000124fd81a3016450526f54046830303031303230
      390258206d39caedba129297062b820ba6d85b3e432c443c8a8a31d3c6232be6
      906d38dc
    58    ; byte_str(64)
      30f9d61523204793965fc9eb2be788db9d2b02692d877673c86ebffbfb676998
      4515d2f1a287a92d2c134c1024f20f018dbe952a2ccae7ed2980a9f242d02c9c
----------------------------------------------------------------------
Token claims:
a8    ; map(8)
  3a    ; int(-75008)
```

```
  58    ; byte_str(32)
    4ca14d0bc8601cad2e511de1964e939338b6fc20f8231aa178ca79519b0ffae7
  3a    ; int(-75000)
  71    ; text_str(17)
    "PSA_IOT_PROFILE_1"
  3a    ; int(-75009)
  49    ; byte_str(9)
    0614b457fffe0f77ce
  3a    ; int(-75001)
  01    ; int(1)
  3a    ; int(-75002)
  19    ; int(8192)
  3a    ; int(-75003)
  58    ; byte_str(32)
    011c0001060000000100000f2030f000000000000000000000000000000000000
  3a    ; int(-75004)
  58    ; byte_str(32)
    4922b7bbd31c0c81c9b0485ccfb5396ec24ffa877ece441e11c947b791218cf8
  3a    ; int(-75006)
  81    ; array(1)
    a3    ; map(3)
      01    ; int(1)
      64    ; text_str(4)
        "PRoT"
      04    ; int(4)
      68    ; text_str(8)
        "00010209"
      02    ; int(2)
      58    ; byte_str(32)
        6d39caedba129297062b820ba6d85b3e432c443c8a8a31d3c6232be6906d38dc

List of claims printed with human-friendly names:
  ARM PSA Nonce, Claim ID: -75008
  58    ; byte_str(32)
    4ca14d0bc8601cad2e511de1964e939338b6fc20f8231aa178ca79519b0ffae7

  ARM PSA Profile ID, Claim ID: -75000
  71    ; text_str(17)
    "PSA_IOT_PROFILE_1"

  ARM PSA / IETF EAT UEID, Claim ID: -75009
  49    ; byte_str(9)
    0614b457fffe0f77ce

  ARM PSA Partition ID, Claim ID: -75001
  01    ; int(1)

  ARM PSA Lifecycle, Claim ID: -75002
  19    ; int(8192)

  ARM PSA Implementation ID, Claim ID: -75003
  58    ; byte_str(32)
    011c0001060000000100000f2030f000000000000000000000000000000000000

  ARM PSA Boot seed, Claim ID: -75004
  58    ; byte_str(32)
    4922b7bbd31c0c81c9b0485ccfb5396ec24ffa877ece441e11c947b791218cf8

  ARM PSA Software components, Claim ID: -75006
  81    ; array(1)
    a3    ; map(3)
      01    ; int(1)
      64    ; text_str(4)
        "PRoT"
      04    ; int(4)
      68    ; text_str(8)
        "00010209"
      02    ; int(2)
      58    ; byte_str(32)
        6d39caedba129297062b820ba6d85b3e432c443c8a8a31d3c6232be6906d38dc
```

Security Configuration Token (Table 5.3 Claims of Security Configuration Token on page 13)

```
----------------------------------------------------------------------

Calling sl_se_attestation_get_config_token...
SL_STATUS_OK (cycles: 541281 time: 14244 us)

Config token
============
----------------------------------------------------------------------
Raw token:
d28443a10126a0590133a83a000124ff5820c3e3664dcc47711bf81734bc95f0
87d81dd841d73fc805fc9237c7b3dfa25c503a000124f76853494c4142535f31
3a00012500490614b457fffe0f77ce3a000128df582400000001000000000000
0000000000000000002000010209ffffffff0000002500000000 3a000128e058
180000000010044440040104141122447742204420a0600053a000128e15840
c4af4ac69aab9512db50f7a26ae5b4801183d85417e729a56da974f4e08a562c
de6019dea9411332dc1a743372d170b436238a34597c410ea177024de20fc819
3a000128e25840b1bc6f6fa56640ed522b2ee0f5b3cf7e5d48f60be8148f0dc0
8440f0a4e1dca47c04119ed6a1be31b7707e5f9d001a659a051003e95e1b936f
05c37ea793ad633a000128e3501504444004010414112244771422044258 40b7
47d98be9cef8a91af0292a479a3fa499527018b97ac1188ddefb0fa6fcb9b3d1
d4159240a8663c8803a2ef7cebdf7644fa3394cf1057d612e1b3977d9de92d

COSE_Sign1 structure:
d2    ; tag(18)
  84    ; array(4)
    43    ; byte_str(3)
      a10126
    a0    ; map(0)
    59    ; byte_str(307)
      a83a000124ff5820c3e3664dcc47711bf81734bc95f087d81dd841d73fc805fc
      9237c7b3dfa25c503a000124f76853494c4142535f313a00012500490614b457
      fffe0f77ce3a000128df582400000001000000000000000000000000000000020
      00010209ffffffff0000002500000000 3a000128e05818000000001004444004
      0104141122447742204420a0600053a000128e15840c4af4ac69aab9512db50
      f7a26ae5b4801183d85417e729a56da974f4e08a562cde6019dea9411332dc1a
      743372d170b436238a34597c410ea177024de20fc8193a000128e25840b1bc6f
      6fa56640ed522b2ee0f5b3cf7e5d48f60be8148f0dc08440f0a4e1dca47c0411
      9ed6a1be31b7707e5f9d001a659a051003e95e1b936f05c37ea793ad633a0001
      28e3501504444004010414112244771422044 2
    58    ; byte_str(64)
      b747d98be9cef8a91af0292a479a3fa499527018b97ac1188ddefb0fa6fcb9b3
      d1d4159240a8663c8803a2ef7cebdf7644fa3394cf1057d612e1b3977d9de92d
----------------------------------------------------------------------
Token claims:
a8    ; map(8)
  3a    ; int(-75008)
  58    ; byte_str(32)
    c3e3664dcc47711bf81734bc95f087d81dd841d73fc805fc9237c7b3dfa25c50
  3a    ; int(-75000)
  68    ; text_str(8)
    "SILABS_1"
  3a    ; int(-75009)
  49    ; byte_str(9)
    0614b457fffe0f77ce
  3a    ; int(-76000)
  58    ; byte_str(36)
    00000001000000000000000000000000000002000010209ffffffff0000002500000000
  3a    ; int(-76001)
  58    ; byte_str(24)
    000000001004444004010414112244772422044 20a060005
  3a    ; int(-76002)
  58    ; byte_str(64)
    c4af4ac69aab9512db50f7a26ae5b4801183d85417e729a56da974f4e08a562c
    de6019dea9411332dc1a743372d170b436238a34597c410ea177024de20fc819
  3a    ; int(-76003)
  58    ; byte_str(64)
    b1bc6f6fa56640ed522b2ee0f5b3cf7e5d48f60be8148f0dc08440f0a4e1dca4
    7c04119ed6a1be31b7707e5f9d001a659a051003e95e1b936f05c37ea793ad63
  3a    ; int(-76004)
  50    ; byte_str(16)
```

```
    150444400401041411224477142204 42

List of claims printed with human-friendly names:
  ARM PSA Nonce, Claim ID: -75008
  58    ; byte_str(32)
    c3e3664dcc47711bf81734bc95f087d81dd841d73fc805fc9237c7b3dfa25c50

  ARM PSA Profile ID, Claim ID: -75000
  68    ; text_str(8)
    "SILABS_1"

  ARM PSA / IETF EAT UEID, Claim ID: -75009
  49    ; byte_str(9)
    0614b457fffe0f77ce

  SE Status, Claim ID: -76000
  58    ; byte_str(36)
    000000010000000000000000000000000000000020000102 09ffffffff0000002500000000

  OTP Configuration, Claim ID: -76001
  58    ; byte_str(24)
    000000001004444004010414112244772422044 20a060005

  OTP MCU Boot key, Claim ID: -76002
  58    ; byte_str(64)
    c4af4ac69aab9512db50f7a26ae5b4801183d85417e729a56da974f4e08a562c
    de6019dea9411332dc1a743372d170b436238a34597c410ea177024de20fc819

  OTP MCU Auth key, Claim ID: -76003
  58    ; byte_str(64)
    b1bc6f6fa56640ed522b2ee0f5b3cf7e5d48f60be8148f0dc08440f0a4e1dca4
    7c04119ed6a1be31b7707e5f9d001a659a051003e95e1b936f05c37ea793ad63

  Current applied tamper settings, Claim ID: -76004
  50    ; byte_str(16)
    150444400401041411224477142204 42
-------------------------------------------------------------------


Exiting...
SL_STATUS_OK (cycles: 8 time: 0 us)
```

Note: The reserved tamper source in ID 76004 returns a value of 0 or 5.

8.4.2 Simplicity Commander

Run the `security attestation` command to retrieve and validate the security configuration token (Table 5.3 Claims of Security Configuration Token on page 13) from the HSE.

```
commander security attestation --serialno 440030580
```

```
Certificate chain successfully validated up to Silicon Labs device root certificate.

-75008 ARM PSA nonce                  : 05a88aeef627dd663058e3d758fe9a827942da0793da72af81c79a4f60fa9824
-75000 ARM PSA Profile ID             : SILABS_1
-75009 ARM PSA/IETF EAT UEID          : 0614b457fffe0f77ce
-76000 SE status                      : 000000010000000000000000000003a90000002000010209ffffffff0000002500000000
-76001 OTP configuration              : 000000001004444004010414112244772422044 20a060005
-76002 MCU sign key                   : c4af4ac69aab9512db50f7a26ae5b4801183d85417e729a56da974f4e08a562c
                                        de6019dea9411332dc1a743372d170b436238a34597c410ea177024de20fc819
-76003 MCU command key                : b1bc6f6fa56640ed522b2ee0f5b3cf7e5d48f60be8148f0dc08440f0a4e1dca4
                                        7c04119ed6a1be31b7707e5f9d001a659a051003e95e1b936f05c37ea793ad63
-76004 Current applied tamper settings : 150444400401041411224477142204 42

Successfully validated signature of attestation token.
DONE
```

Note: The reserved tamper source in ID 76004 returns a value of 0 or 5.

9. Revision History

Revision 0.5

August 2023

- Updated table and note in 1. Series 2 Device Security Features.
- Replaced Device Compatibility with SE Firmware in 1. Series 2 Device Security Features.
- Updated 4. Device Certificate Options.
- Updated 5. Entity Attestation Token (EAT).

Revision 0.4

March 2022

- Added digit 4 to Note 3 in 1. Series 2 Device Security Features.
- Updated Device Compatibility and moved it under 1. Series 2 Device Security Features.
- Updated 8.2.1 Simplicity Commander and OpenSSL.

Revision 0.3

January 2022

- Added Entity Attestation Token (EAT) to key points on the front page.
- Added UG489 to the table in 1.2 Key Reference.
- Added CPMS information to 4. Device Certificate Options.
- Added 5. Entity Attestation Token (EAT).
- Added EAT to 6. Remote Authentication Process.
- Added API for retrieving attestation tokens from the HSE to 7. Secure Engine Manager.
- Added Entity Attestation Token (EAT) example to 8.1 Overview.
- Added the Windows folder for the root and factory certificates to 8.1 Overview.
- Added attestation example to 8.1.3 Using Platform Examples with GSDK v3.2.3.
- Added 8.4 Entity Attestation Token (EAT) to 8. Examples.

Revision 0.2

September 2021

- Formatting updates for source compatibility.
- Added revised terminology to 1. Series 2 Device Security Features and use this terminology throughout the document.
- Updated Device Compatibility.
- Added 7. Secure Engine Manager.
- Added the URL to retrieve the root and factory certificates to 8. Examples.
- Added 8.2.3 Simplicity Studio to 8. Examples.
- Revised 8. Examples to the latest Simplicity Studio and Simplicity Commander version, updated the content.

Revision 0.1

September 2020

- Initial Revision.

Simplicity Studio

One-click access to MCU and wireless tools, documentation, software, source code libraries & more. Available for Windows, Mac and Linux!

IoT Portfolio
www.silabs.com/IoT

SW/HW
www.silabs.com/simplicity

Quality
www.silabs.com/quality

Support & Community
www.silabs.com/community

SILICON LABS

Silicon Laboratories Inc.
400 West Cesar Chavez
Austin, TX 78701
USA

www.silabs.com