AN1396: Certificate-Based *Bluetooth*® Authentication and Pairing

Certificate-based Bluetooth authentication and pairing extends the normal Bluetooth authentication and pairing process, which is defined in the Bluetooth Core Specification, with additional application layer security. This additional layer ensures that the connection is not authenticated by the user (who could, for example, authenticate the connection by reading and entering a passkey), but by a central authority via certificates. This is especially important in use cases where the user cannot always be trusted, or where the authentication must be made without any user interaction.

This application note describes the theoretical background of certificate-based authentication and pairing, and demonstrates the usage of the related sample applications that can be found in Silicon Labs' Bluetooth SDK.

KEY POINTS

- Authentication without user interaction
- Device Certificates and Central Authority
- Certificate-based authentication and pairing

Copyright © 2022 by Silicon Laboratories

1 Introduction

Bluetooth authentication and pairing, as defined in the Bluetooth Core Specification, ensures that two connecting devices can establish a secure connection between each other without any third-party device eavesdropping on the communication or acting like a Man-In-The-Middle. The authentication is based on a secret that is exchanged outside of the Bluetooth connection. This can either be a passkey, that is displayed on one device and is entered into the other device by the user, or it can also be so called Out-Of-Band (OOB) data transmitted via NFC, QR code, or any other media. It is the responsibility of the user to transmit the secret from one device to the other, and therefore the connection is authenticated by the user. Ultimately, this procedure ensures that a device can only pair with another device if a user has physical access to both devices (at least at the first pairing), which is a good solution for many use cases.

It is, however, often insufficient to rely on physical accessibility and user interaction. For example, in a factory, many people can have physical access to the machines, but only some of them have the privilege to configure them. In other use cases, devices should create an authenticated connection between each other without any user interaction. None of these can be realized with the existing Bluetooth authentication methods, thus the need for certificate-based authentication and pairing.

Certificate-based authentication and pairing relies on Bluetooth authentication and pairing, but instead of exchanging a secret outside of the Bluetooth connection, signed certificates and signed random data are exchanged via the Bluetooth connection, and the authentication relies on the validity of these signatures. This way, the authentication can be done without any user interaction, and it does not rely on the user but on the signatures added to the certificates issued by the central authority. There is only one requirement to get this procedure to work: the devices must be preprogrammed with signed certificates.

1.1 Requirements

Currently, certificate-based authentication and pairing is supported on Secure Vault High (SVH) devices noted below:

- EFR32xG21B
- EFR32xG24B

and Secure Vault Mid (SVM) devices through TrustZone noted below:

- EFR32xG21A
- EFR32xG22
- EFR32xG24A

To learn more about SVH devices, visit https://www.silabs.com/security. To learn more about TrustZone support, refer to *AN1374: Series 2 TrustZone*.

2 Theoretical Background

To understand certificate-based authentication and pairing, it is essential to understand the basics of public-key cryptography and certificates first.

2.1 Public-key Cryptography

Public-key cryptography, or asymmetric cryptography, uses private-public key pairs to sign and verify data. Any device can generate such a key pair based on open cryptographic algorithms.

- The private key must be a sufficiently random number.
- The public key is a number derived from the private key using a one-way function that ensures that the private key cannot be calculated from the public key.
- The private key must be held in secret.
- The public key can be openly distributed.
- The private key can be used to generate a signature. A signature is generated from a digest of the input data and from the private key with a cryptographic algorithm.
- The public key can be used to verify a signature. The verification needs the original input data, the signature, and the public key as an input.
- Since the data to be signed can be of any length, usually a hash is generated first from this data, and the hash serves as the input data for the signature.

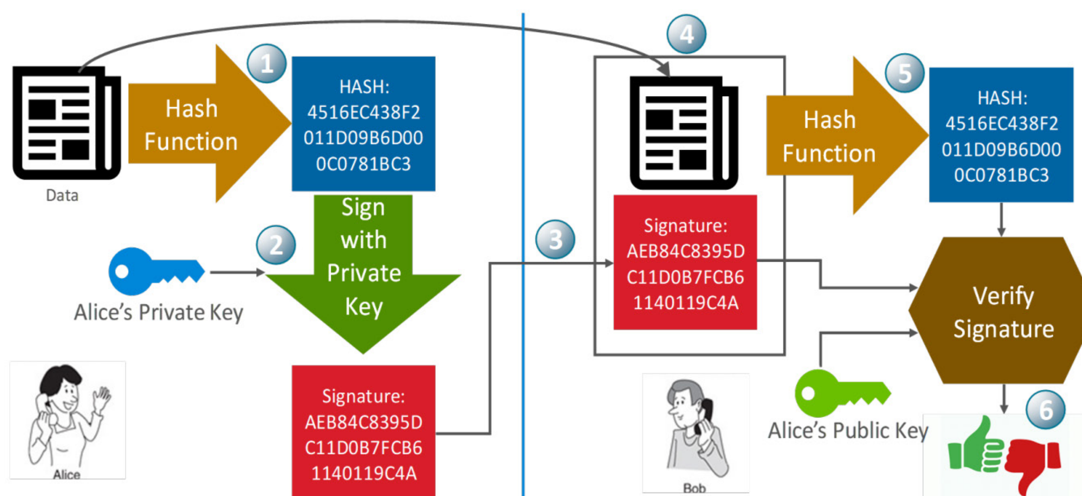


Figure 2.1. Digital Signature

The Digital Signal Standard (DSS) is specified in FIPS 186-4.

2.2 Certificates

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 signing key, or a certificate authority's private signing 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.

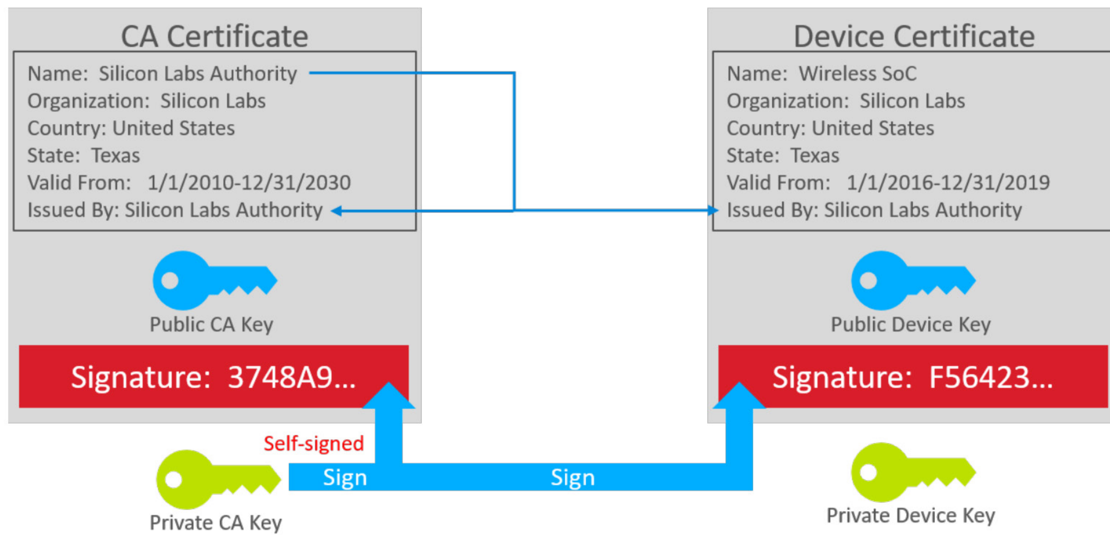


Figure 2.2. Digital Certificates and Chain of Trust

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. When a device must prove its identity, then it must prove that it has the private key by signing a random number challenge sent by the requester. The signature of this random number challenge can be verified with the corresponding public key, which is found in the certificate.

To learn more about certificates, see *AN1268: Authenticating Silicon Labs Devices Using Device Certificates*, which addresses certificates in detail.

3 Creating Central Authority and Device Certificates

Certificate-based authentication and pairing, as its name suggests, relies on certificates. Each device that participates in such an authentication process must be preprogrammed with a:

- **device certificate**, which holds the identity of the device (including the public key of the device), and with a
- **corresponding private key**, which can be used to generate signatures and, thereby, prove the identity of the device.

Additionally, all the devices must know which are the trusted peer devices. Therefore, all devices must be preprogrammed either with a list of device certificates, which holds the identities of the trusted devices, or with a

- **Central Authority (CA) Certificate**, which can be used to validate any certificate that belongs to a trusted device.

Because it is easier to store a single CA certificate rather than a list of device certificates, this method is applied in the sample applications.

Putting this into practice, the following steps must be done before certificate-based authentication and pairing can be applied:

1. A CA Certificate must be created (with self-signing) along with a CA private key that will be used to sign all the device certificates. This is done on a central machine. Note that the private key must be securely stored, preferably in a hardware security module (HSM). At a minimum, the private key must not leave this machine.
2. Each device must generate a private key. These private keys must be generated on the devices, and they must not leave the devices.
3. Each device must generate its device certificate signing request, which holds its public key (generated from its private key) and the credentials.
4. Each device must get its device certificate signing request signed by the CA. To do this, the certificate signing request must be transmitted to the central machine (this can be done via UART), and the signed certificate must be transmitted back to the device.
5. The CA certificate must be flashed to each device so that they can validate the device certificates of the peer devices.

Because this process is not easy to implement, Silicon Labs provides sample applications that do all the required steps.

- The **Bluetooth - SoC Certificate Signing Request Generator** sample app generates the private key and the device certificate signing request on the device. It can also be used to connect to the certificate authority and send over the device certificate signing request to be signed.
- The **create_authority_certificate.py** (CA) Python script can be used to generate the CA certificate along with the private key. It also creates a header file with the CA certificate that can be stored on the devices.
- The **production_line_tool.py** (PLT) Python script can be used for reading out the device certificate signing requests (CSRs) from the requesting devices, signing the CSRs with the CA private key, and flashing them back to the device.

  Note: the private key used by this script is visible in plain text, therefore this tool is not to be used in a secure production environment, a hardware security module (HSM) must be used instead.

The Python scripts can be found in the following folder: {SDK_folder}/app/bluetooth/script/certificate_authorities.

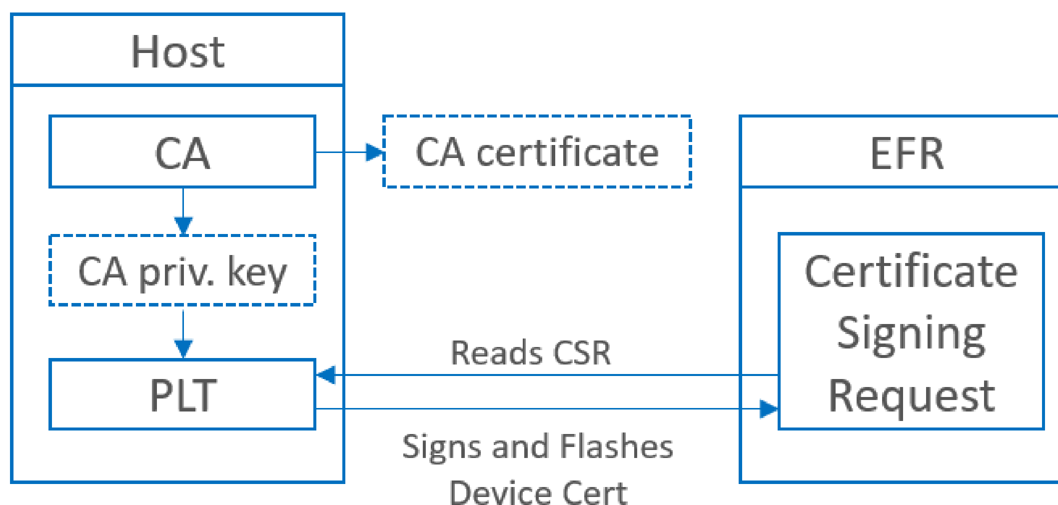
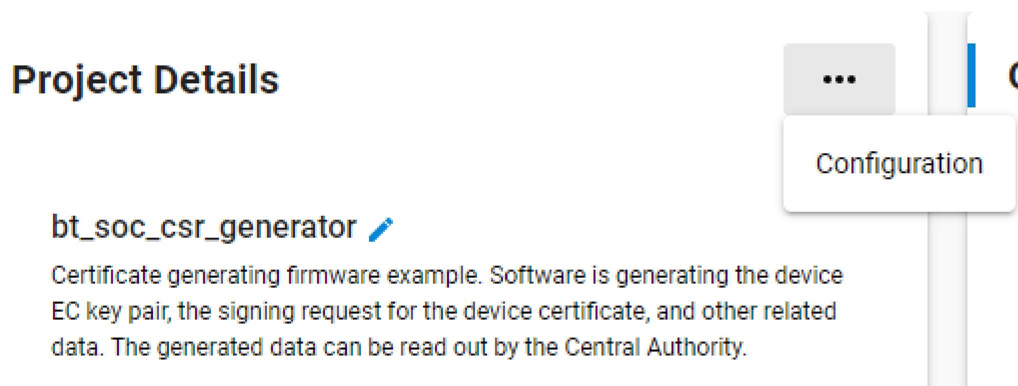


Figure 3.1. Signing the Device Certificates

To generate the device certificate and get it signed, follow this process:

1. Factory-reset your device to make sure that no keys and certificates are stored on it. You can do this with Simplicity Commander using the **Recover Bricked Device** option in the GUI or with the following CLI command: `commander device recover`.
2. Flash an **Internal Storage Bootloader** to your device. (Must be generated and built as a separate project.) See UG489 Gecko Bootloader User Guide for specific instructions.
3. Create a new **Bluetooth - SoC CSR Generator** project in Simplicity Studio.
4. Open the slcp file of the project.
5. On the Overview tab, under Project Details, open the three-dots-menu, and click the **Configuration** button as shown below.



6. Modify the **Certification Subject Data** fields so that your certificate subject contains your company's information.

Certification Subject Data

Country Identifier	State Identifier	Locality Identifier	Organization Identifier	Organization Unit Identifier
FI	Uusimaa	Espoo	Silicon Labs	Wireless

7. Build and flash the project to your device. This will automatically generate the private-public key pair and the certificate signing request on startup.
8. Create a CA certificate. Skip this step if you already have a root certificate you wish to use. Install the Python modules cryptography and jinja2 as follows:

```
pip3 install cryptography
pip3 install jinja2
```

Once these modules have been installed, create the CA certificate with the following command

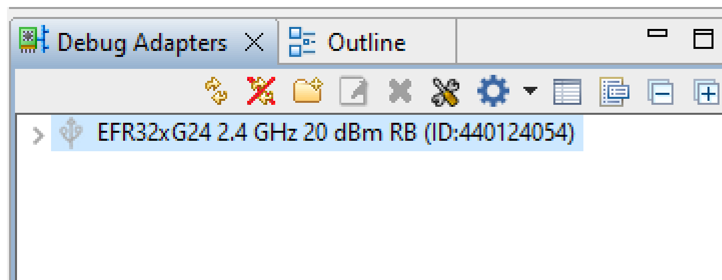
```
python3 {SDK_folder}\app\bluetooth\script\certificate authorities\create_authority_certifi-
cate.py
```

Note: this certificate will be created with factory default parameters, to customize the certificate with your own unique identifiers, see the help menu for this script which is available by running the following command:

```
python3 <Gecko sdk root>\app\bluetooth\script\certificate authorities\create_authority_certifi-
cate.py -h.
```

9. The CA certificate can now be found in *{SDK_folder}*\app\bluetooth\script\certificate authorities\central_authority\certificate.pem.

10. Check the Jlink serial number of your debug adapter either with Simplicity Studio or with Simplicity Commander by using the command 'commander adapter probe'.



11. Run the **production_line_tool.py** python script on your computer with the following parameters:

```
Python3 {SDK_folder}\app\bluetooth\script\certificate authorities\production_line_tool.py -p ble
--serial <serialnumber>
```

    This will read out the signing request, sign the device certificate, and flash the signed certificate on the device. *Note: the 'serial' parameter is not required if only one device is connected to your PC.*

12. Now the key pair and the signed certificate are stored on your device. You can flash a new application to the device. This will not remove the keys and the certificate.
13. To also flash the CA certificate (root certificate), you will have to copy the generated `sl_bt_cbap_root_cert.h` file (found under *{SDK_folder}/app/bluetooth/script/certificate_authorities/central_authority*) into your new application project under the */config* folder. This step is to be done later.

The new application project can be anything that uses the device certificates. In the following sections, the **Bluetooth – SoC Certificate-Based Authentication and Pairing** sample app is demonstrated.

Note: It is important to add a bootloader to your new project that has secure boot enabled. This ensures that the firmware you programmed to the device cannot be changed, and therefore it ensures the originality of the CA certificate, which is part of the firmware. See AN1218: Secure Boot with RTSL for more information on secure boot.

4 Certificate-Based Authentication and Pairing

4.1 Out-Of-Band (OOB) Pairing

Certificate-based authentication and pairing is based on Bluetooth LE Secure Connections pairing (available since Bluetooth 4.2) using the Out-Of-Band (OOB) pairing method. This method is described in the Bluetooth Core Specification. It is essential to understand how this procedure works before certificate-based authentication and pairing is explained.

OOB pairing ensures an authenticated connection between two devices, assuming that the out-of-band data exchange is done, such that no third-party has access to the exchanged data. The data can be exchanged over different media, such as NFC or QR codes, or a user can manually enter the data displayed on the peer device.

Note: the OOB data is much longer than the 6-digit passkey used in passkey exchange methods.

The OOB pairing procedure consists of 3 phases:

1. Public key exchange
    - In this phase, the two devices exchange their public keys via the Bluetooth connection and calculate a common secret from them using the Elliptic Curve Diffie Hellman (ECDH) key exchange. The common secret can then be used to encrypt the connection. However, this step does not authenticate the connection, because one cannot be sure if the received public key belongs to the device they wanted to pair with or to an adversary device that acts like a Man-In-The-Middle (MITM).
    - **Important note**: the public key used here is generated by the Bluetooth Security Manager (SM), and it is not the same as the public key stored in the device certificate!
2. Authentication stage 1: Out-of-band data exchange and confirmation
    - In this stage, each device picks a random number and generates a confirmation value from that random number using its own public key (the one that was generated by the SM). The random numbers and the confirmation values are exchanged out-of-band. Finally, both devices calculate the confirmation value again from the received random number with the public key it had received before. If the received confirmation value matches the calculated one, it ensures that the previously received public key belongs to the device with which the user wanted to pair.
3. Authentication stage 2 and long-term key calculation
    - The second stage of authentication confirms that both devices have successfully completed the exchange by transmitting newly calculated confirmation values via the Bluetooth connection. If any device fails to present a valid confirmation value, the pairing is aborted. Additionally, a long-term key (LTK) is generated in this phase from newly exchanged random numbers, from the common secret (calculated earlier with ECDH algorithm), and from the device addresses.

4.2 Authentication with Certificates

Authentication without user interaction is not simple. If there is no possibility to exchange any secret outside of the Bluetooth connection, then it may happen that an adversary device plays Machine-In-The-Middle; that is, it creates an authenticated connection with both device A and device B while pretending that it is device B in the communication with device A and vice versa. Since the authentication procedure is carried out without any issue on both sides, neither device A nor device B will detect that they are part of an attack. In this case, the MITM device cannot only eavesdrop on the connection, but also control it.

One solution for this is to exchange some data between the devices well before the pairing happens. For example, two devices can be preprogrammed with a common secret, or with each other’s public key. This ensures authenticity, too. However, this solution does not scale, because each device should store a secret for each device it wants to pair with. Also, it is hard to add a new trusted device to the system later.

Certificate-based authentication and pairing solves this issue with device certificates that are signed by a Central Authority. In this case the only information the devices must store, other than their own certificate and associated private signing key, is the CA Certificate.

Authentication with certificates needs several steps. Here we assume that all the devices are already preprogrammed with a private key, with a device certificate signed by a CA, and with a CA certificate.

1. On startup each device verifies its own device certificate with the stored CA certificate. The signature on the device certificate can be verified using the public key stored in the CA certificate.
2. Two devices connect and sign that they want to initiate a certificate-based authentication and pairing.
3. The two devices exchange their device certificates via the connection.

4. Both devices verify the signature of the received device certificate chain using the public key stored in the CA certificate. If the verification succeeds, it means that the other device is potentially a trusted device. However, the peer device must still prove that it also has the private key corresponding to the device certificate to verify its identity.
5. The devices initiate an OOB pairing, where the OOB data is sent in-band, that is on the Bluetooth connection. This pairing process ensures an authenticated connection, provided that the OOB data is exchanged without a MITM.
6. To make sure that the OOB data really comes from a trusted device and not from an adversary device acting like a MITM, both devices sign the OOB data with their own private key. The signature is exchanged along with the OOB data.
7. Both devices verify the signature with the public key of the other device – which is included in the already received and verified device certificate. If the signature is valid, then it is ensured that the OOB data comes from a trusted device and the OOB pairing procedure is carried on.
8. The devices now have an encrypted and authenticated connection.

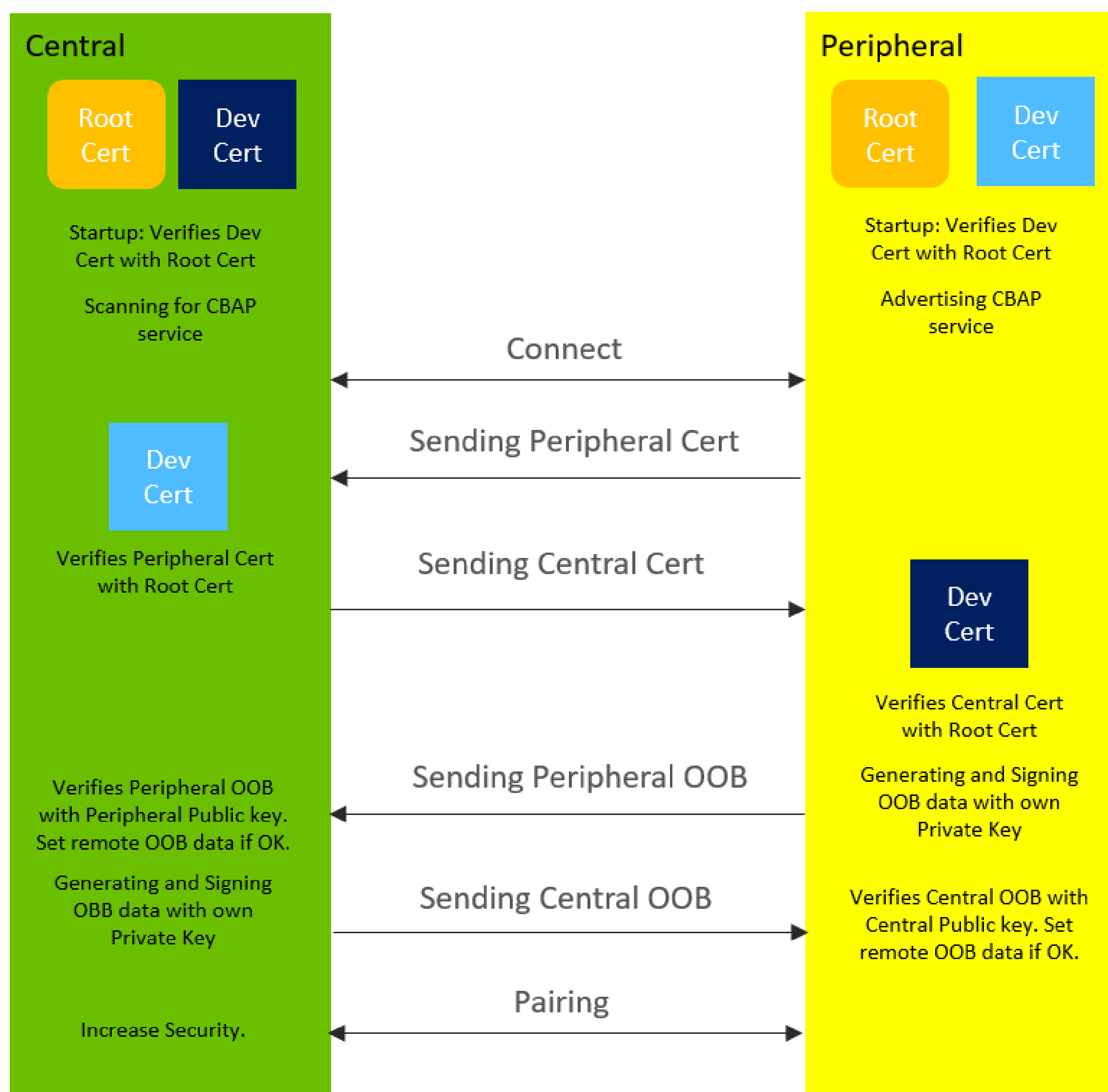


Figure 4.1. Certificate-Based Authentication and Pairing

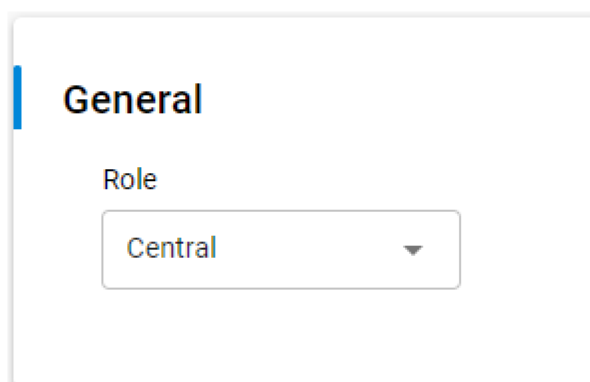
Silicon Labs' Bluetooth SDK provides a sample application to demonstrate this procedure: **Bluetooth – SoC Certificate-Based Authentication and Pairing**. This sample application can be used as a starting point for developing any Bluetooth application that needs certificate-based authentication and pairing.

The example application works only if the private key, the device certificate, and the CA certificate are present on the device. These can be generated using the CSR Generator. Before using this example application, make sure you have properly-signed certificates present on your device, see section 3 Creating Central Authority and Device Certificates.

In a pairing example, two devices are needed: a central and a peripheral. The example is written so that it can act as either the central or as the peripheral device. In the central role, the device looks for the peripheral (that advertises the custom CBAP service), connects to it, initiates certificate-based authentication and pairing, and finally writes a characteristic that can only be written via an authenticated connection. In the peripheral role, the device advertises, accepts connections, participates in certificate-based authentication and pairing, and finally turns on an LED when its dedicated characteristic is written. The role can be defined in the project configuration.

To test the example:

1. Connect two devices to your PC.
2. Make sure you have run the CSR generator on both devices, so that you have a private key and a signed device certificate on both devices.
3. Create an **SoC Bluetooth Apploader OTA DFU Bootloader** project and flash the bootloader to both devices.
4. Create a **Bluetooth – SoC Certificate-Based Authentication and Pairing** example project.
5. The CA certificate must be stored in the application, so you must copy the generated `sl_bt_cbap_root_cert.h` file into this project, under the */config* folder.
6. Build this example and flash it to one of the devices.
7. Open the slcp file of this project.
8. On the Overview tab, under Project Details, open the three-dots-menu, and click **Configuration**.
9. Change the Role from **Peripheral** to **Central** as shown below:



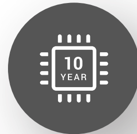
10. Build the project again and flash it to the other device.
11. Open a terminal program and connect to both devices to see their debug messages.
12. Reset both devices. The central will automatically connect to the peripheral and after a few seconds you should see the LED on the peripheral turning on.

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

Disclaimer
Silicon Labs intends to provide customers with the latest, accurate, and in-depth documentation of all peripherals and modules available for system and software implementers using or intending to use the Silicon Labs products. Characterization data, available modules and peripherals, memory sizes and memory addresses refer to each specific device, and "Typical" parameters provided can and do vary in different applications. Application examples described herein are for illustrative purposes only. Silicon Labs reserves the right to make changes without further notice to the product information, specifications, and descriptions herein, and does not give warranties as to the accuracy or completeness of the included information. Without prior notification, Silicon Labs may update product firmware during the manufacturing process for security or reliability reasons. Such changes will not alter the specifications or the performance of the product. Silicon Labs shall have no liability for the consequences of use of the information supplied in this document. This document does not imply or expressly grant any license to design or fabricate any integrated circuits. The products are not designed or authorized to be used within any FDA Class III devices, applications for which FDA premarket approval is required or Life Support Systems without the specific written consent of Silicon Labs. A "Life Support System" is any product or system intended to support or sustain life and/or health, which, if it fails, can be reasonably expected to result in significant personal injury or death. Silicon Labs products are not designed or authorized for military applications. Silicon Labs products shall under no circumstances be used in weapons of mass destruction including (but not limited to) nuclear, biological or chemical weapons, or missiles capable of delivering such weapons. Silicon Labs disclaims all express and implied warranties and shall not be responsible or liable for any injuries or damages related to use of a Silicon Labs product in such unauthorized applications. **Note: This content may contain offensive terminology that is now obsolete. Silicon Labs is replacing these terms with inclusive language wherever possible. For more information, visit www.silabs.com/about-us/inclusive-lexicon-project**

Trademark Information
Silicon Laboratories Inc.®, Silicon Laboratories®, Silicon Labs®, SiLabs® and the Silicon Labs logo®, Bluegiga®, Bluegiga Logo®, EFM®, EFM32®, EFR, Ember®, Energy Micro, Energy Micro logo and combinations thereof, "the world's most energy friendly microcontrollers", Redpine Signals®, WiSeConnect , n-Link, ThreadArch®, EZLink®, EZRadio®, EZRadioPRO®, Gecko®, Gecko OS, Gecko OS Studio, Precision32®, Simplicity Studio®, Telegesis, the Telegesis Logo®, USBXpress® , Zentri, the Zentri logo and Zentri DMS, Z-Wave®, and others are trademarks or registered trademarks of Silicon Labs. ARM, CORTEX, Cortex-M3 and THUMB are trademarks or registered trademarks of ARM Holdings. Keil is a registered trademark of ARM Limited. Wi-Fi is a registered trademark of the Wi-Fi Alliance. All other products or brand names mentioned herein are trademarks of their respective holders.

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

www.silabs.com