AN1386: Bluetooth Mesh Certificate-Based Provisioning

Certificate-Based Provisioning (CBP) is defined in Bluetooth Mesh protocol specification version 1.1. CBP ensures that devices joining a Bluetooth Mesh network are authentic. This authentication method mitigates the risk of rogue devices joining the network by spoofing the identity of authentic devices.

This application note describes how certificates are used to establish the authenticity of devices wishing to join a mesh network.

KEY POINTS

- Certificate-based provisioning
- Bluetooth Mesh protocol specification 1.1

1 Introduction

Certificate-Based Provisioning makes mesh networks more secure by establishing an identity using a signed certificate chain. This can prevent the possibility of rogue devices joining the network by spoofing the advertisement of a legitimate device. Certificate-Based Provisioning mitigates this by requiring the device to present a signed certificate or chain of certificates which can be verified using standard public key cryptography.

1.1 Requirements

Certificate-Based Provisioning is supported by the following devices:

Table 1.1. Supported Devices and Level of Support

Device	Support
EFR32xG21B EFR32xG24B	Secure Vault

2 Theoretical Background

2.1 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.

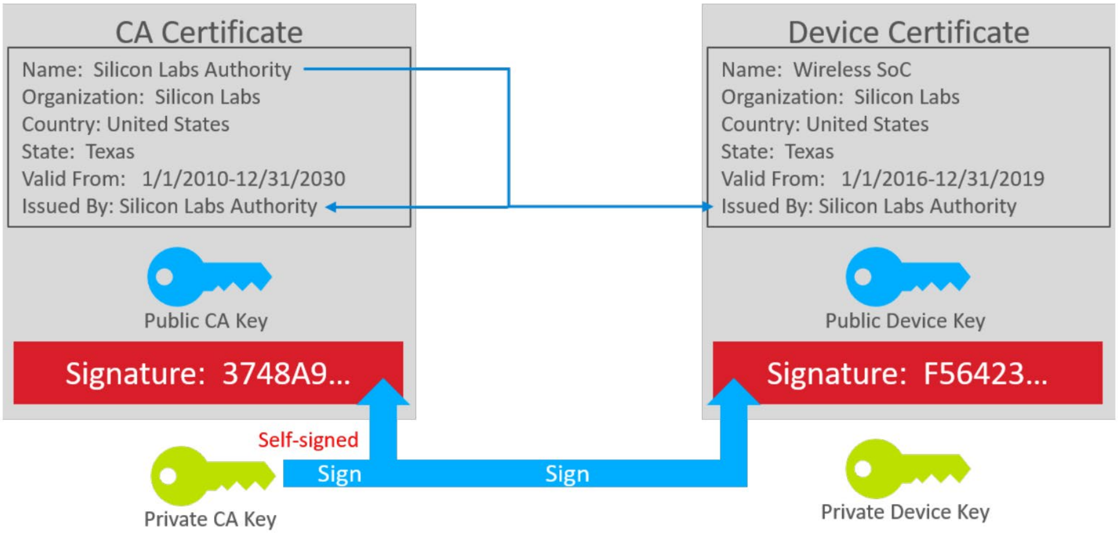


Figure 2.1. Digital Certificates and Chain of Trust

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.

2.1.1 Key Usage

Version 3 of the X.509 standard provides the ability to restrict the purposes that a certificate's public key can be used for. Bluetooth Mesh CBP certificates must allow the 'Key Agreement' usage for the public key. This allows a shared secret to be established between the provisioner and provisionee.

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

2.2 Bluetooth Mesh Certificate-Based Provisioning

Bluetooth Mesh Provisioning is described in the Bluetooth Mesh 1.1 Protocol specification. Refer to this document for specifics of provisioning messages and certificate requirements. A certificate chain is used as the out-of-band (OOB) data for provisioning. Provisioning related information is stored on a device as provisioning records. The device certificate can either be requested from the device during provisioning or obtained by the provisioner from a URI indicated in the provisioning record. The provisioner requests the device certificate by sending a Provisioning Record Request message with the ID set to 'Device Certificate' to the device to be provisioned. The provisionee responds with a Provisioning Record Response message which includes a device certificate. If intermediate certificate(s) are present, they will be requested by the provisioner and sent one at time. Up to fifteen (15) intermediate certificates can be used in the certificate chain. The provisioner verifies the certificate chain before proceeding with the provisioning process. In case the certificate chain cannot be verified, the provisioning process is terminated.

2.2.1 Obtaining Device Certificates from URI

Device certificates and intermediate certificates may be retrieved by the provisioner from the internet using HTTPS protocol. The base uniform resource indicator (URI) of the device certificate can be included provisioning record.

2.2.2 Requirements for Provisioners and Devices

The provisioner of the network must store the root-of-trust certificate for each chain of trust. The root certificate must not be sent from device to provisioner during the provisioning process nor be able available for retrieval from the URI. It is the responsibility of the provisioner to ensure the integrity of the root-of-trust certificate.

2.2.3 Verifying the Device Certificate

The provisioner must verify the authenticity of the device certificate received. In addition to verifying the authenticity of the certificate, as defined in RFC5280, the Bluetooth Mesh protocol specification requires that the following criteria must be met:

- The device certificate must be verifiable, directly, or indirectly, by the root CA
- The device certificate is not expired
- The device certificate has not been revoked by any certificate revocation list (CRL)

The following table summarizes the fields of the device certificate which shall be present:

Table 2.1 Required Certificate Fields

Certificate Field	Requirements
tbsCertificate	As defined by RFC5280 with additional requirements as defined in this table
signatureAlgorithm	Set to “ecdsa-with-SHA256”
SignatureValue	signature computed as in RFC5280
Version	“2”
serialNumber	Must meet the requirements detailed in RFC5280
issuer	Identifies the signer of the certificate. See RFC5280.
validity	• notBefore time should not be earlier than the manufacture date of the device • notAfter time shall be set by the manufacturer
subject	Shall contain a valid distinguished name (DN) with the following restrictions: • organization name of the DN shall be set to the vendor name • common name of the DN shall contain the device UUID in
subjectPublicKeyInfo	As defined by RFC5280 with the following constraints: • algorithm field shall contain “id-ecPublicKey” as the algorithm and secp256r1 curve as the parameters. See RFC5480 for details. • subjectPublicKey field shall contain the public OOB key of the device
basicConstraints Extension	• cA field shall be present and set to FALSE • pathLenConstraint shall not be present
keyUsage Extension	keyAgreement bit shall be set as defined in RFC5280

The following table summarizes the optional fields in the device certificate which may be included. These fields shall be used as defined in RFC5280 without additional constraints.

Table 2.2 Optional Certificate Fields

Optional Certificate Fields
Authority key identifier
Subject key identifier
Certificate policies extension
Issuer Alternative Name
Subject Directory Attribute extension
CRL distribution points extension
Freshest CRL extension
Authority Information Access extension
Subject Information Access extension

All other fields defined by RFC5280 shall not be present in the device certificate.

3 Creating Example Certificate Authority and Device Certificate

This section describes how to create an example certificate authority and device certificate. **The method described here is only intended for evaluation and should not be used in a production environment.** For production, users are encouraged to obtain root certificates from a qualified certificate authority (CA). Users are strongly discouraged from acting as their own CA.

Certificate-Based Provisioning, as its name suggests, relies on certificates. Each device that participates in Certificate-Based provisioning must be preprogrammed with a:

- **private key**, which can be used to prove the identity of the device.

  Note: It is critical that the device private signing key be stored securely.
- **One of the following:**
  - **Corresponding device certificate**, which holds the identity of the device (including the public key of the device), optionally with one or more intermediate certificates
  - URI pointing to a server where the device certificate can be downloaded

Additionally, the provisioner must know which are the trusted devices. Therefore, the provisioner must be preprogrammed with the root certificate(s) used to create the certificate chain of any device to be provisioned with a **Certificate Authority (CA) Certificate**, which can be used to validate any certificate that belongs to a trusted device.

Putting this into practice, the following steps must be done before Certificate-Based Provisioning 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 computer. 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 stored on the provisioner so that it can validate the device certificates of the provisionee devices.

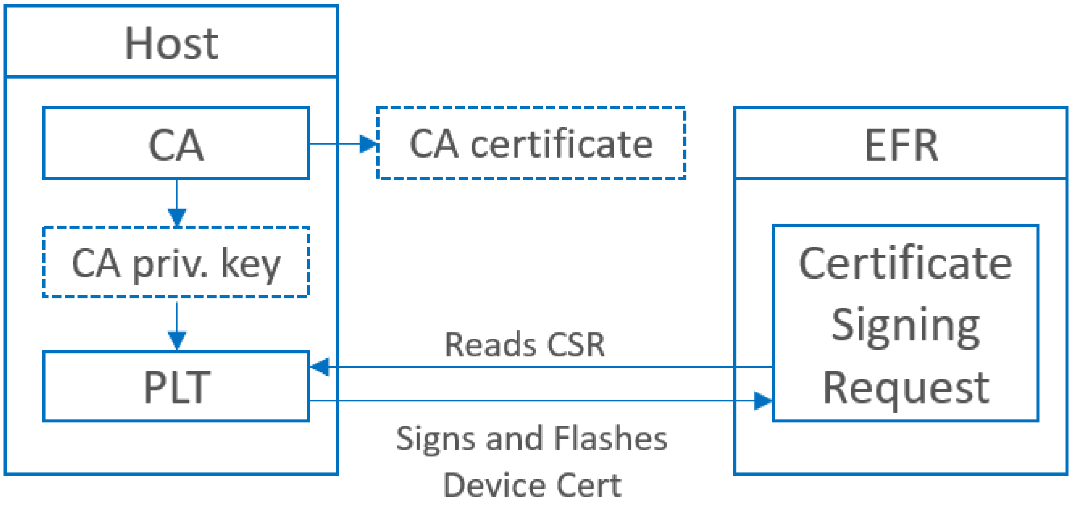
Because this process is not easy to implement, Silicon Labs provides sample applications that do all the required steps.

- The **Bluetooth Mesh - SoC CSR 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.

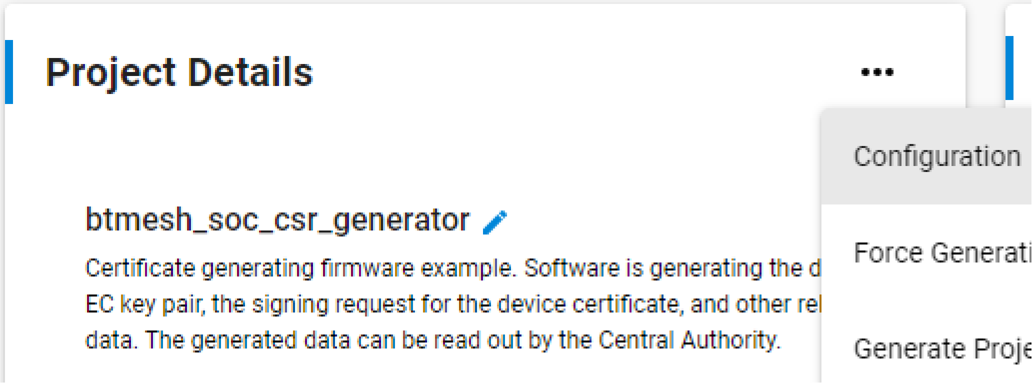
The Python scripts can be found in the following folder: {SDK_folder}/app/bluetooth/script/certificate_authorities.

Figure 3.1. Signing the Device Certificates



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

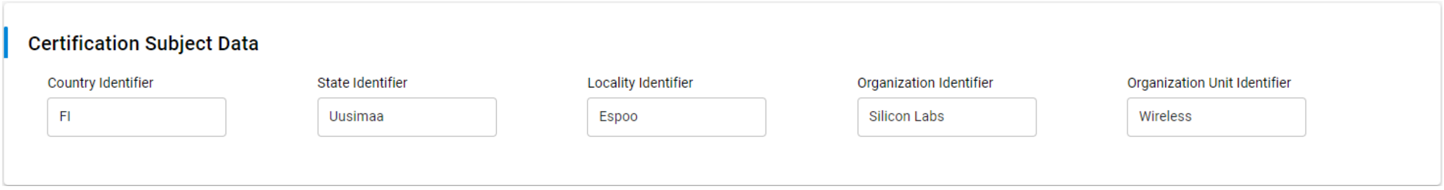
1. Factory-reset your device to make sure that no keys and certificates are stored on it. You can do this with Simplicity Commander using the **Recover Bricked Device** option in the GUI or with the following CLI command: `commander device recover`.
2. Flash an **Internal Storage Bootloader** to your device. (Must be generated and built as a separate project.) See [UG489, Gecko Bootloader User Guide]() for specific instructions.
3. Create a new **Bluetooth Mesh- 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.



btmesh_soc_csr_generator

Certificate generating firmware example. Software is generating the d
EC key pair, the signing request for the device certificate, and other rel
data. The generated data can be read out by the Central Authority.

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
```

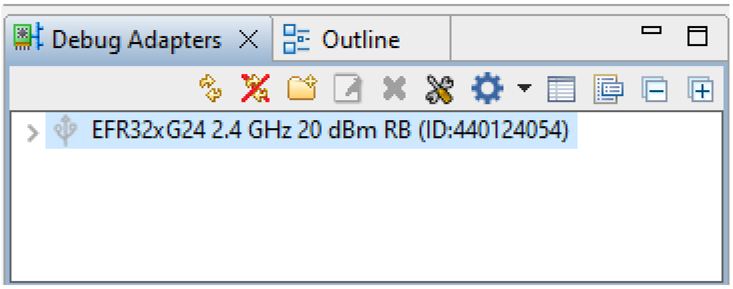
Once these modules have been installed, create the CA certificate with the following command:

```
python3 {SDK_folder}\app\bluetooth\script\certificate authorities\create_authority_certificate.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_certificate.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`.



EFR32xG24 2.4 GHz 20 dBm RB (ID:440124054)

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 --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. At this stage, it is important to ensure that the flash is not completely erased. Simply flashing a new application, and bootloader if required, are sufficient.

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. See AN1218: Secure Boot with RTSL for more information on secure boot.

4 Preparing a Device for Certificate-Based Provisioning

4.1 Device

Once a device certificate, and optionally any intermediate certificates, have been installed on the device, there are a few steps as follows to configure the device for Certificate-Based Provisioning:

1. Call `sl_btmesh_node_init_oob()` to indicate that the device has an out-of-band public key.
2. Enable support for provisioning records by calling `sl_btmesh_node_init_provisioning_records().`
3. Start unprovisioned beaconing by calling `sl_btmesh_node_start_unprov_beaconing().`

5 Example

5.1 BTMesh SOC Empty CBP

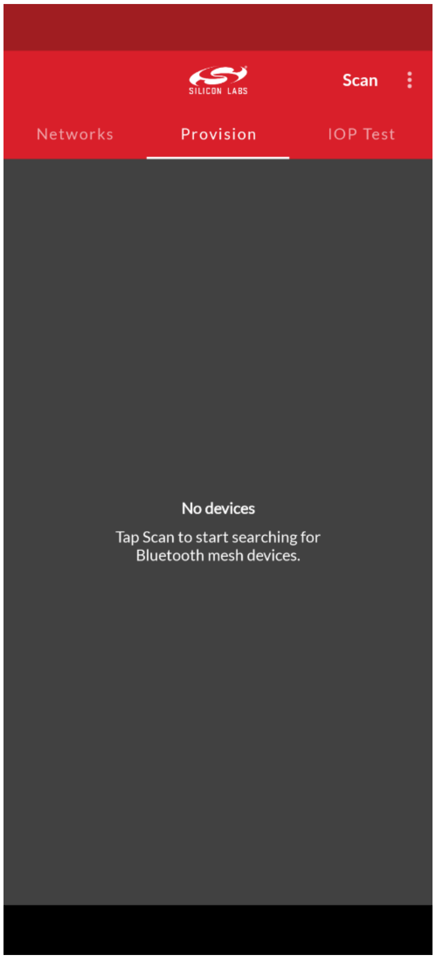
This section covers the use of the “Bluetooth Mesh – SoC Empty with Certificate-Based Provisioning Support” sample application found in the Gecko SDK.

5.1.1 Creating Device Certificate on Device

1. Follow the steps for creating a device certificate, which are detailed in section 3 Creating Example Certificate Authority and Device Certificate.
2. Create an instance of the “Bluetooth Mesh – SoC Empty with Certificate-Based Provisioning Support” sample application for your chosen device.
3. Build the application and flash the output file to your device.

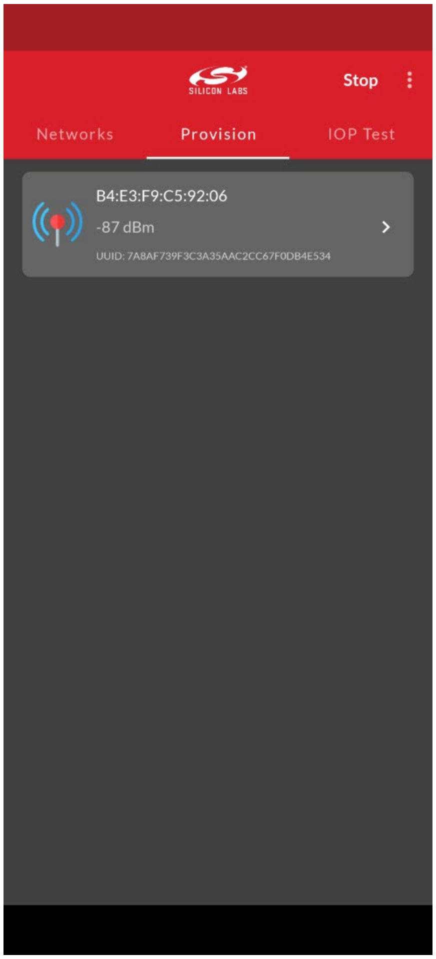
5.2 Provisioning with the Bluetooth Mesh mobile app

Silicon Labs Bluetooth Mesh mobile app supports Certificate-Based Provisioning starting in version 4.1.0. To being provisioning a node with CBP, open the Bluetooth Mesh mobile app, select the Provision tab, and tap the scan icon as shown below.



No devices

Tap Scan to start searching for Bluetooth mesh devices.

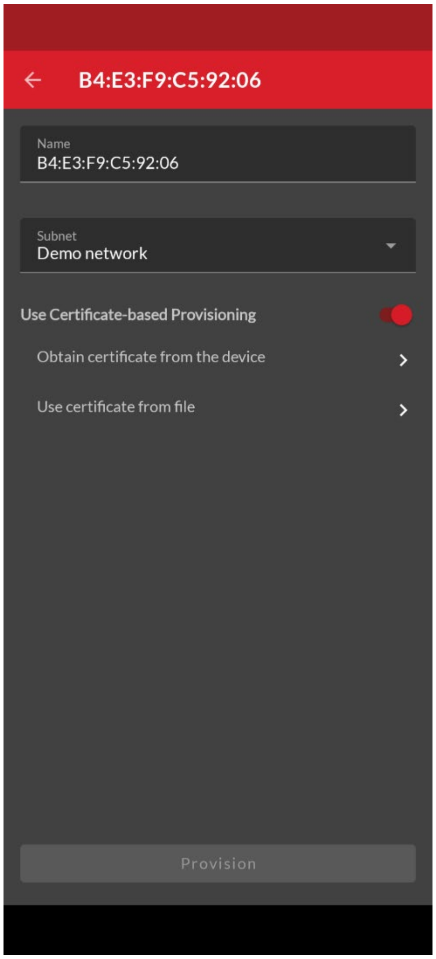


B4:E3:F9:C5:92:06

-87 dBm

UUID: 7A8AF739F3C3A35AAC2CC67F0DB4E534

In the dialog for the unprovisioned node, ensure that the **Use Certificate-based Provisioning** option is enabled then tap the **Obtain certificate from the device** icon as shown below.



B4:E3:F9:C5:92:06

Name
B4:E3:F9:C5:92:06

Subnet
Demo network

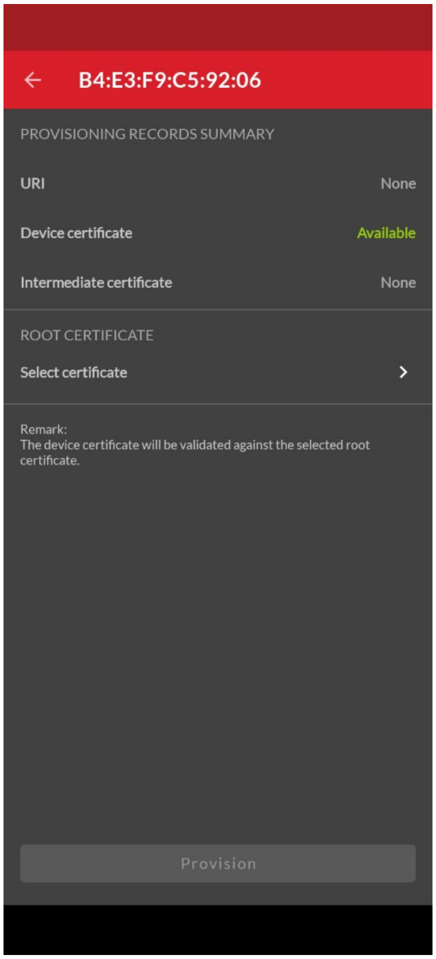
Use Certificate-based Provisioning ●

Obtain certificate from the device ›

Use certificate from file ›

Provision

Once the device certificate has been sent, the **Device certificate** status changes to **Available** as shown below.



B4:E3:F9:C5:92:06

PROVISIONING RECORDS SUMMARY

URI None

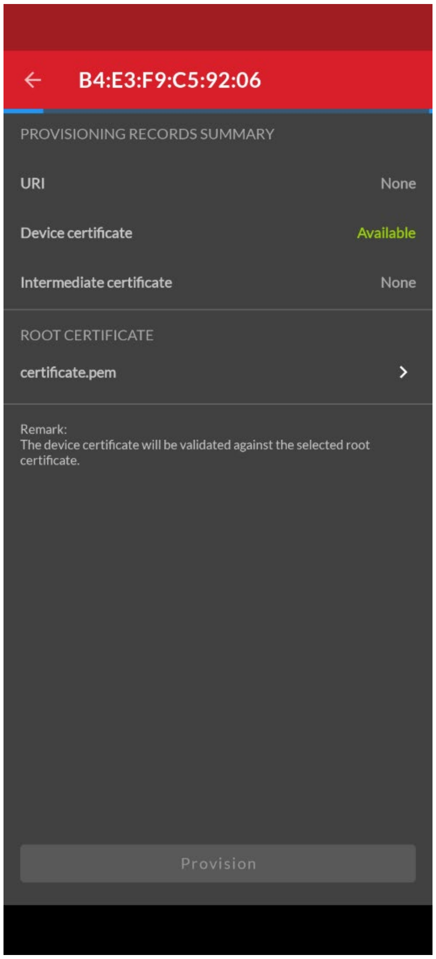
Device certificate Available

Intermediate certificate None

ROOT CERTIFICATE

Select certificate

Remark:
The device certificate will be validated against the selected root certificate.



B4:E3:F9:C5:92:06

PROVISIONING RECORDS SUMMARY

URI None

Device certificate Available

Intermediate certificate None

ROOT CERTIFICATE

certificate.pem

Remark:
The device certificate will be validated against the selected root certificate.

Tap the **Select certificate** icon to browse for the root certificate. **Note**: The root certificate must be accessible to the mobile device either through its filesystem or in a cloud storage service such as Dropbox. Once the root certificate file has been selected, it is shown on the Provisioning Records Summary dialog.

Tap the **Provisioning** button at the bottom of the screen to complete the provisioning process.

5.3 Provisioning with the BT Mesh Host Provisioner Sample Application

The btmesh-host_provisioner sample application runs on a posix-type platform such as RaspberryPi. For instructions on getting started building the Host Provisioner sample app, see section 2.3 of AN1371 - Bluetooth Mesh NCP Host Provisioner Example Walkthrough. Note that when exporting the Bluetooth Mesh Host Provisioner code it is necessary to include the CBP flag as follows:

```
make export CBP=1
```

To enable Certificate-Based Provisioning, rebuild the sample with the following command:

```
make CBP=1
```

5.3.1 Installing Root Certificates

For the BT Mesh provisioner to validate device certificate chains, it must have the root certificate for those chains. The root certificates must be stored in PEM format in the file **CA/ca-certificate.crt** under the BT Mesh host provisioner source folder.

Example:

The root certificate generated by the script mentioned in section will be C:\Users\<user>\SimplicityStudio\SDKs\gecko_sdk\app\bluetooth\script\certificate_authorities\central_authority\certificate.pem.

Copy this file to the provisioner host:

<path to exported files>/app/btmesh/example_host/btmesh_host_provisioner/CA/ca-certificate.crt

5.3.2 Specifying a Base URI for Certificate Retrieval

This section describes how to customize the sample application to include a base URI for certificate retrieval in the provisioning records.

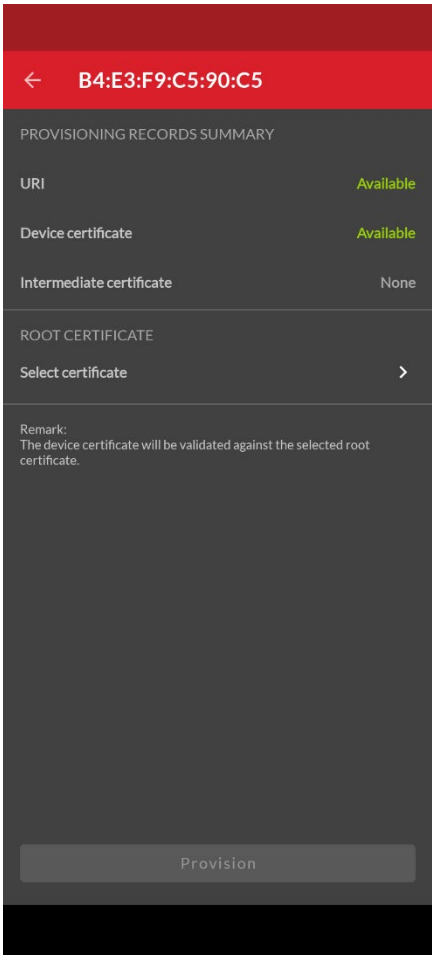
1. Add the following code to the top level of the project's app.c file.

```
struct mesh_const_provisioning_record {
  const uint8_t *ptr; //< Provisioning record data
  uint16_t len; //< Length of provisioning record data
};

/** Number of spec-defined intermediate certificates */
#define MESH_INTERMEDIATE_CERTIFICATE_COUNT 15
struct mesh_const_provisioning_records {
  /** Certificate-based provisioning base URI */
  struct mesh_const_provisioning_record base_uri;
  /** Complete local name */
  struct mesh_const_provisioning_record complete_local_name;
  /** Appearance */
  struct mesh_const_provisioning_record appearance;
  /** Intermediate certificate 1 to 15 */
  struct mesh_const_provisioning_record intermediate_certificate[MESH_INTERMEDIATE_CERTIFICATE_COUNT];
};

extern const struct mesh_const_provisioning_records *mesh_const_provisioning_records;

static const struct mesh_const_provisioning_records records = {
  .base_uri = {
    (const uint8_t *)"\x17//www.example.com/",
    19,
  },
  .complete_local_name = {
    NULL,
    0,
  },
  .appearance = {
    NULL,
    0,
  },
  .intermediate_certificate = {
    { NULL, 0 }, { NULL, 0 }, { NULL, 0 }, { NULL, 0 }, { NULL, 0 },
    { NULL, 0 }, { NULL, 0 }, { NULL, 0 }, { NULL, 0 }, { NULL, 0 },
    { NULL, 0 }, { NULL, 0 }, { NULL, 0 }, { NULL, 0 }, { NULL, 0 },
  },

};
```

2. Edit the 'base_uri' member to point to the desired URI.
3. Add the following line to `app_init()`:

```
mesh_const_provisioning_records = &records;
```

4. Build the project and download the binary to the target board.
5. Scan for the node as described in section 5.2 above. Now the BTMESH mobile app will show that the URI is available as shown below:



B4:E3:F9:C5:90:C5

PROVISIONING RECORDS SUMMARY

URI	Available
Device certificate	Available
Intermediate certificate	None

ROOT CERTIFICATE

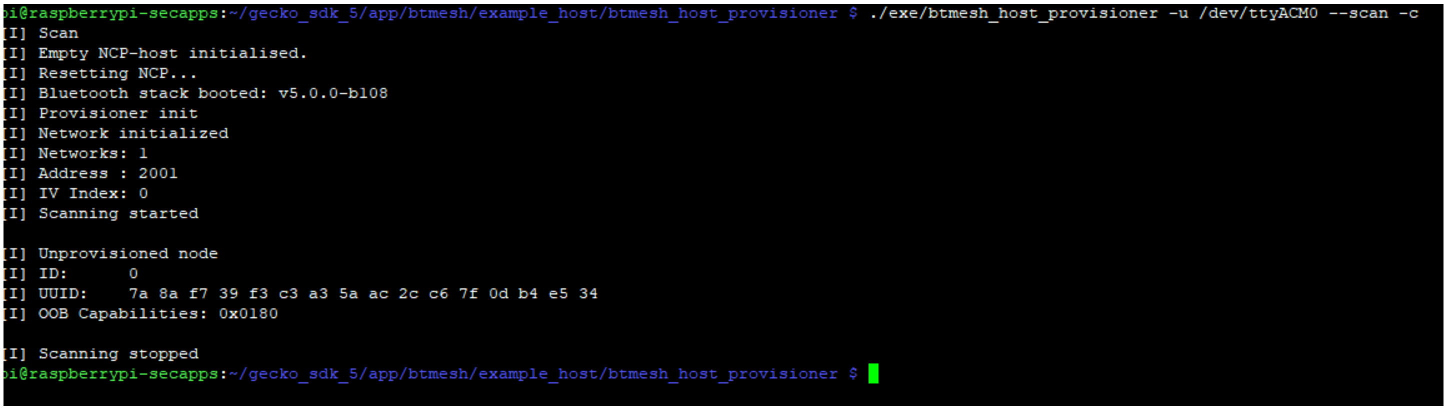
Select certificate

Remark:
The device certificate will be validated against the selected root certificate.

5.3.3 Running the BT Mesh Host Provisioner

The BT Mesh host provisioner can provision nodes to a network with the node’s 128-bit UUID which is obtained by scanning as shown below:

```
./exe/btmesh_host_provisioner -u <serial port> --scan --c
```



```
pi@raspberrypi-secapps:~/gecko_sdk_5/app/btmesh/example_host/btmesh_host_provisioner $ ./exe/btmesh_host_provisioner -u /dev/ttyACM0 --scan -c
[I] Scan
[I] Empty NCP-host initialised.
[I] Resetting NCP...
[I] Bluetooth stack booted: v5.0.0-b108
[I] Provisioner init
[I] Network initialized
[I] Networks: 1
[I] Address : 2001
[I] IV Index: 0
[I] Scanning started

[I] Unprovisioned node
[I] ID:       0
[I] UUID:     7a 8a f7 39 f3 c3 a3 5a ac 2c c6 7f 0d b4 e5 34
[I] OOB Capabilities: 0x0180

[I] Scanning stopped
pi@raspberrypi-secapps:~/gecko_sdk_5/app/btmesh/example_host/btmesh_host_provisioner $
```

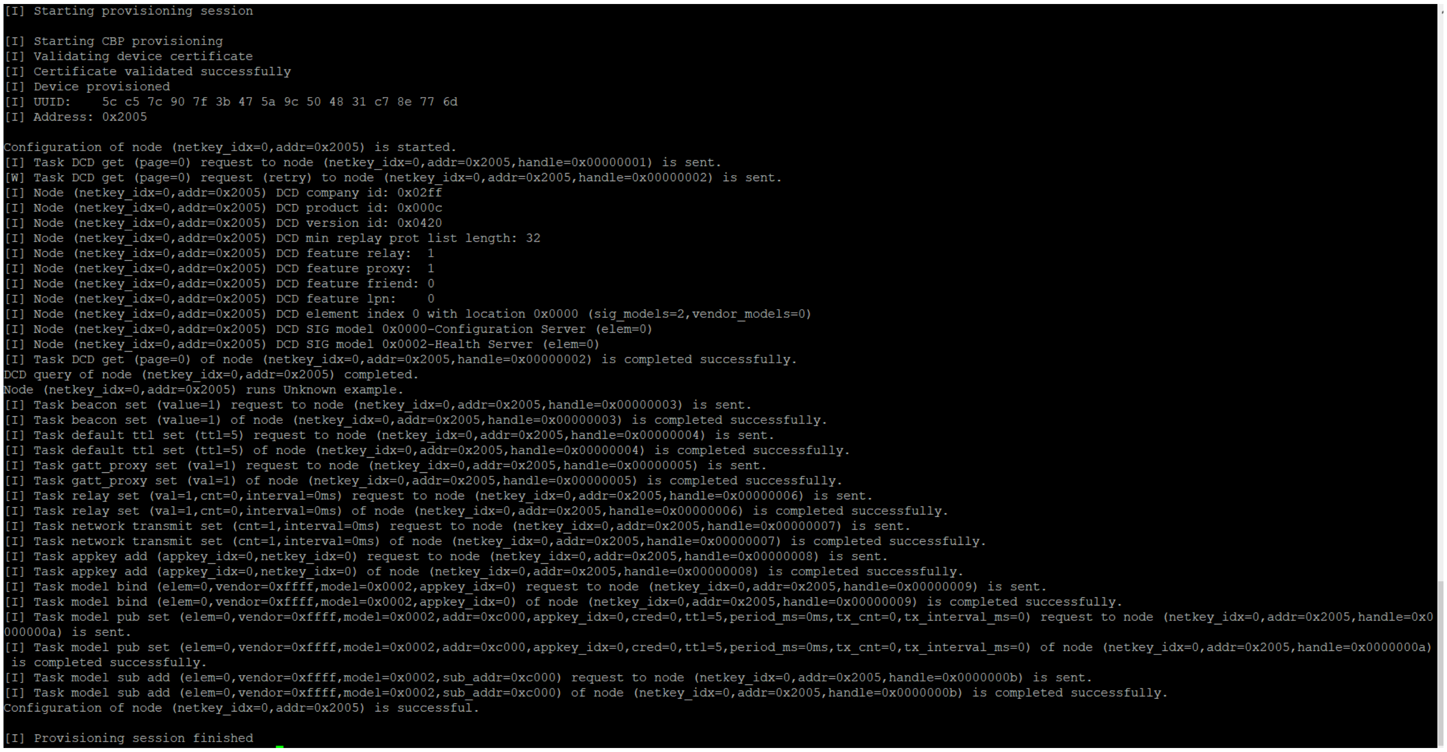
as desc

Once the node’s UUID has been found, it can be used to provision the device as shown below:

```
./exe/btmesh_host_provisioner -u <serial port. -c -provision <UUID>
```

Example:

```
./exe/btmesh_host_provisioner    -u    /dev/ttyACM0    --c    -b    115200    --provision
9c93e21b4ea4a65d871031d7f67bb702
```



```
[I] Starting provisioning session

[I] Starting CBP provisioning
[I] Validating device certificate
[I] Certificate validated successfully
[I] Device provisioned
[I] UUID:    5c c5 7c 90 7f 3b 47 5a 9c 50 48 31 c7 8e 77 6d
[I] Address: 0x2005

Configuration of node (netkey_idx=0,addr=0x2005) is started.
[I] Task DCD get (page=0) request to node (netkey_idx=0,addr=0x2005,handle=0x00000001) is sent.
[W] Task DCD get (page=0) request (retry) to node (netkey_idx=0,addr=0x2005,handle=0x00000002) is sent.
[I] Node (netkey_idx=0,addr=0x2005) DCD company id: 0x02ff
[I] Node (netkey_idx=0,addr=0x2005) DCD product id: 0x000c
[I] Node (netkey_idx=0,addr=0x2005) DCD version id: 0x0420
[I] Node (netkey_idx=0,addr=0x2005) DCD min replay prot list length: 32
[I] Node (netkey_idx=0,addr=0x2005) DCD feature relay:  1
[I] Node (netkey_idx=0,addr=0x2005) DCD feature proxy:  1
[I] Node (netkey_idx=0,addr=0x2005) DCD feature friend: 0
[I] Node (netkey_idx=0,addr=0x2005) DCD feature lpn:    0
[I] Node (netkey_idx=0,addr=0x2005) DCD element index 0 with location 0x0000 (sig_models=2,vendor_models=0)
[I] Node (netkey_idx=0,addr=0x2005) DCD SIG model 0x0000-Configuration Server (elem=0)
[I] Node (netkey_idx=0,addr=0x2005) DCD SIG model 0x0002-Health Server (elem=0)
[I] Task DCD get (page=0) of node (netkey_idx=0,addr=0x2005,handle=0x00000002) is completed successfully.
DCD query of node (netkey_idx=0,addr=0x2005) completed.
Node (netkey_idx=0,addr=0x2005) runs Unknown example.
[I] Task beacon set (value=1) request to node (netkey_idx=0,addr=0x2005,handle=0x00000003) is sent.
[I] Task beacon set (value=1) of node (netkey_idx=0,addr=0x2005,handle=0x00000003) is completed successfully.
[I] Task default ttl set (ttl=5) request to node (netkey_idx=0,addr=0x2005,handle=0x00000004) is sent.
[I] Task default ttl set (ttl=5) of node (netkey_idx=0,addr=0x2005,handle=0x00000004) is completed successfully.
[I] Task gatt_proxy set (val=1) request to node (netkey_idx=0,addr=0x2005,handle=0x00000005) is sent.
[I] Task gatt_proxy set (val=1) of node (netkey_idx=0,addr=0x2005,handle=0x00000005) is completed successfully.
[I] Task relay set (val=1,cnt=0,interval=0ms) request to node (netkey_idx=0,addr=0x2005,handle=0x00000006) is sent.
[I] Task relay set (val=1,cnt=0,interval=0ms) of node (netkey_idx=0,addr=0x2005,handle=0x00000006) is completed successfully.
[I] Task network transmit set (cnt=1,interval=0ms) request to node (netkey_idx=0,addr=0x2005,handle=0x00000007) is sent.
[I] Task network transmit set (cnt=1,interval=0ms) of node (netkey_idx=0,addr=0x2005,handle=0x00000007) is completed successfully.
[I] Task appkey add (appkey_idx=0,netkey_idx=0) request to node (netkey_idx=0,addr=0x2005,handle=0x00000008) is sent.
[I] Task appkey add (appkey_idx=0,netkey_idx=0) of node (netkey_idx=0,addr=0x2005,handle=0x00000008) is completed successfully.
[I] Task model bind (elem=0,vendor=0xffff,model=0x0002,appkey_idx=0) request to node (netkey_idx=0,addr=0x2005,handle=0x00000009) is sent.
[I] Task model bind (elem=0,vendor=0xffff,model=0x0002,appkey_idx=0) of node (netkey_idx=0,addr=0x2005,handle=0x00000009) is completed successfully.
[I] Task model pub set (elem=0,vendor=0xffff,model=0x0002,addr=0xc000,appkey_idx=0,cred=0,ttl=5,period_ms=0ms,tx_cnt=0,tx_interval_ms=0) request to node (netkey_idx=0,addr=0x2005,handle=0x0
000000a) is sent.
[I] Task model pub set (elem=0,vendor=0xffff,model=0x0002,addr=0xc000,appkey_idx=0,cred=0,ttl=5,period_ms=0ms,tx_cnt=0,tx_interval_ms=0) of node (netkey_idx=0,addr=0x2005,handle=0x0000000a)
 is completed successfully.
[I] Task model sub add (elem=0,vendor=0xffff,model=0x0002,sub_addr=0xc000) request to node (netkey_idx=0,addr=0x2005,handle=0x0000000b) is sent.
[I] Task model sub add (elem=0,vendor=0xffff,model=0x0002,sub_addr=0xc000) of node (netkey_idx=0,addr=0x2005,handle=0x0000000b) is completed successfully.
Configuration of node (netkey_idx=0,addr=0x2005) is successful.

[I] Provisioning session finished
```

Now the node is provisioned on the network and can be configured for groups and functionality. For specifics, refer to AN1371, Bluetooth Mesh NCP Host Provisioner Example Walkthrough.