AN708: Setting Smart Energy Certificates for Zigbee Devices

This application note gives an overview of the use of manufacturing certificates in a Smart Energy network. It also describes the components of a Smart Energy certificate and the differences between test certificates and production certificates. It also explains (with the help of examples) how to use Simplicity Commander or the EM3xx utilities to program, verify, and erase Smart Energy certificates from Silicon Labs Mighty Gecko (EFR32MG) and EM3x chips.

Refer to *AN1089: Using Installation Codes for Zigbee Devices* for how to use Simplicity Commander to program, verify, and erase installation codes with the Silicon Labs Wireless Gecko (EFR32™) portfolio.

KEY FEATURES

- Provides an overview of zigbee Smart Energy certificates, certificate components, certification authorities, and certification validation.
- Describes the importance of matching all security data components.
- Provides examples for programming the certificate into a device based on your target platform.

1 Smart Energy Certificate Overview

Zigbee uses public/private key technology to authenticate a device joining a Smart Energy network and provides a means to securely establish encryption keys for future transactions. The Smart Energy specification uses Elliptical Curve Cryptography (ECC) for cryptographic authentication and key generation.

Zigbee uses ECC with the key establishment cluster to derive a link key. It also uses ECC for creating digital signatures for software upgrade images sent through the zigbee Over-the-Air (OTA) Upgrade cluster. Certicom (www.certicom.com) provides both the certificates and the ECC technology for use in zigbee Smart Energy networks.

Smart Energy 1.0 used an ECC curve 163k1 with a 48-byte certificate. Smart Energy 1.2 introduced use of the ECC curve 283k1 with a 74-byte certificate. The certificates are separate and unique and are not interoperable. However when both are present on the same device they must contain the same 64-bit Extended Unique Identifier (EUI64). Devices may have one or both sets of security data installed. The rules for what devices must have both certificates or what devices only need one certificate is governed by the Smart Energy specification.

Note: While the Silicon Labs Gecko Bootloader also makes use of ECC-based digital authentication with public and private key pairs for authenticating signed firmware images, Smart Energy certificates are not required to use signed images, nor is the Gecko Bootloader required for zigbee Smart Energy devices. For more information about the Gecko Bootloader and its security features, see *UG266: Gecko Bootloader User's Guide*.

2 Security Components

When referring to Smart Energy certificates, there are actually three components that make up the ECC security data contained within a node:

- Certificate
- Static Private Key
- CA Public Key

2.1 Certificate

The first component of the ECC security data is the certificate. This is the device's EUI64 and static public key, which are signed using the CA's private key. The certificate is associated with a particular EUI64. **As a result, the EUI64 of the device must match the value contained in the certificate.**

Although this is part of the security data, it is not secret, and does not need to be handled in the same fashion as the private key. It is transmitted over the air during key establishment.

2.1.1 Smart Energy 1.0 Certificate

Smart Energy 1.0 uses a 48-byte certificate. Table 1 summarizes the four different fields contained within the certificate body.

Table 1. Smart Energy 1.0 Certificate Body Fields

Field name	Length (bytes)	Description
Public Key Reconstruction Data	22	Device's public key signed by the CA's private key.
Subject	8	Contains the EUI64 associated with the certificate, in big endian format.
Issuer	8	Identity of the CA that issued the certificate.
Attributes	10	An extra set of data associated with the device whose authenticity is guaranteed by the CA. It is not currently used by zigbee Smart Energy.

2.1.2 Smart Energy 1.2 Certificate

Smart Energy 1.2 uses a 74-byte certificate. Table 2 summarizes the four different fields contained within the certificate body.

Table 2. Smart Energy 1.2 Certificate Body Fields

Name	Bytes	Description
Type	1	Type of certificate = 0, implicit no extensions
SerialNo	8	Serial Number of the certificate
Curve	1	Curve identifier (sect283k1 is 13 or byte value 0x0D)
Hash	1	Hash identifier (AES-MMO is byte value 0x08)
Issuer	8	8 byte identifier, EUI64
ValidFrom	5	40-bit Unix time from which the certificate is valid
ValidTo	4	32-bit # of seconds from the ValidFrom time for which the certificate is considered valid (0xFFFFFFFF = infinite)
SubjectID	8	8 byte identifier, EUI64
KeyUsage	1	Bit flag indicating key usage (0x88 = digital signature or key agreement allowed)
PublicKey	37	37-byte compressed public key value from which the public key of the Subject is reconstructed.

2.2 Static Private Key

The second component of the ECC security data is the static private key. This is a secret piece of data that should be carefully protected. Silicon Labs recommends that during the manufacturing process only those computers that need to know this data to program the chip have access to it.

A Smart Energy 1.0 private key is 21-bytes in length. A Smart Energy 1.2 private key is 36-bytes in length.

2.3 CA Public Key

This is the CA's public key that corresponds to the secret private key held by Certicom. It is used to authenticate certificates received by remote devices and validate the keys derived using the Smart Energy Key Establishment Cluster. While it is not transmitted over the air, it is also public information and does not need to be kept secret.

A Smart Energy 1.0 CA Public Key is 22-bytes in length. A Smart Energy 1.2 CA Public Key is 37-bytes in length.

3 Certificate Authorities

Two Certificate Authorities (CA) are provided by Certicom for use in Smart Energy networks:

- A Test CA that is intended only for use in internal development environments.
- The Production CA that is intended for use in real Smart Energy deployments.

Note: Certificates from the Test CA and the Production CA are **not compatible**.

There are two CAs because developers and installers have different security needs. Developers need to be able to test and debug devices where they will have access to the private keys. Their interests lie in having access to internal data in order to accomplish their goal of writing software. On the other hand, installers want reasonable assurances that a device will not have compromised security data. Their goal is to have a secure installation. Certicom can meet both needs by providing different security for each CA.

3.1 Test CA

Certicom provides a Test CA for quickly and easily generating certificates for use in development units wishing to use ECC. The following link can be used to obtain certificates from the Test CA: http://www.certicom.com/index.php/gencertregister.

Silicon Labs provides two test certificates in their Smart Energy sample application, bound to EUI64 addresses 0022080000000001 and 0022080000000002. These may be used in combination with the Test CA to perform key establishment. They can be found in the supplemental ECC package (available upon request via the Silicon Labs Support Portal's Software Releases tab) as certicom-test-cert-1.txt and certicom-test-cert-2.txt. The following is the CA Public Key for the Smart Energy 1.0 Test CA:

```
0200FDE8A7F3D1084224962A4E7C54E69AC3F04DA6B8
```

3.2 Production CA

The Production CA is the official CA that is used by all deployed devices. All manufactured devices should contain the Production CA's public key, and a certificate and private key associated with the Production CA.

The following is the Smart Energy 1.0 Production CA's Public Key, which should be installed on production devices:

```
0202264C5E4CBFA186A6B925B966B5B3A4D7A390344E
```

Production CAs are obtained through a different mechanism from Test CAs. Since manufacturing will involves hundreds or potentially thousands of devices, a different process is used to obtain bulk certificates for a block of EUI64 addresses. In addition, security of the transmission of the private keys is much more important and thus is handled differently from the Test CA.

It is recommended that a manufacturer obtain their own block of EUI64 addresses for production units instead of using each chip's pre-programmed 64-bit MAC address (EUI64) from Silicon Labs' address range. Certificates identify a Smart Energy device and its vendor, and a Silicon Labs chip may be used in many Smart Energy deployments. In addition, it is easier to issue certificates for blocks of EUI64 addresses.

Silicon Labs cannot guarantee a set of known EUI64 addresses with a particular set of chips, so it is recommended that the manufacturer install their own to make it easier to manage them. Contact Certicom for more information about obtaining production certificates.

4 Certificate Validation

It is important to note that ECC uses a technology called implicit certificates. Traditional SSL certificates used on the Internet contain the unencrypted public key of the device along with a separate signature field that ensures the authenticity of the data. However, with implicit certificates, the two pieces of data are combined together to shrink the size of the certificate for use in embedded devices. This is known as *Public Key Reconstruction data*.

As a result of the combination, it is not possible to verify an implicit certificate without using the certificate as part of the Key Establishment Cluster. Only by performing Key Establishment to derive a link key can a device determine if its certificate is valid.

5 Matching the Components

It is important to note that the four pieces of security data installed for Smart Energy **must all match up correctly**. The installed certificate must be the correct one for the private key that is also installed. The EUI64of the device must match the certificate that is installed. Lastly, the certificate must have been issued by the same CA whose public key is installed along with it.

If any of the components is mismatched, the device will never be able to successfully perform key establishment to authenticate itself on a Smart Energy network.

5.1 Manufacturing Test

Silicon Labs recommends that manufacturers validate their process by testing the installed certificate in their first batch of devices. A full test involves successfully joining a Smart Energy network and using the key establishment cluster to derive a link key. The devices in the test should be using Production Certificates issued by the Production CA.

Silicon Labs' SoC and network coprocessor (NCP) platforms can all store application-specific manufacturing data into an area of flash that can be used to store unique device information. This information is write-once at run-time and is typically programmed via a SerialWire or JTAG debug adapter, although some NCP customers prefer to set this data during manufacturing via the EmberZNet Serial Protocol (EZSP). Refer to *UG100: EZSP Protocol Reference Guide* for information about the available commands for configuring the NCP. Note that some ICs may disallow modification of certain manufacturing data via runtime calls after the Read Protection of flash has already been enabled.

Note: Hard-coding a device's unique information in source code does not work when using the same application image on multiple devices. The preferred alternative is to have the software read in the device's unique information from the manufacturing area of flash.

6 Working with Certificates

6.1 Smart Energy 1.0 and 1.2 Certificates

Smart Energy 1.0 used an ECC 163k1 curve with a 48-byte certificate. Smart Energy 1.2 added support for ECC 283k1 curve with a 74-byte certificate. The certificates and the libraries are not interoperable. The zigbee specification requirements describe what devices must support both certificates and both sets of libraries.

6.2 Overriding the Default EUI64

By default, a EUI64 address from Silicon Labs' range is pre-programmed into all EM3x and EFR32 chips. Customers may override this by writing their own EUI64 into the Custom EUI64 area of the manufacturing flash.

When programming certificates, the **em3xx_load.exe** tool (for EM3x-based devices) or Simplicity Commander application (for EFR32-based devices) allows you to override the default EUI64. When a custom EUI64 is already present, the certificate or installation code **must use a EUI64 that matches the custom one**. See *UG107: EM35x Utilities Guide*, for more information about using em3xx_load and related EM3x utilities with EM3x chips. See *UG162: Simplicity Commander User's Guide* for using Simplicity Commander with EFR32 chips.

If you do not want to override the EUI64, then the certificate must use the Silicon Labs EUI64, and that **EUI64 must match what is already programmed on the chip** in the TOKEN_MFG_EMBER_EUI_64 field. The tool prevents overriding EUI64 of the device with another EUI64 from Silicon Labs' range in order to prevent duplicate EUI64s in the field.

The reason these requirements are in place is to prevent certificates from being programmed on the wrong device. The certificate is bound to the EUI64 of the device and therefore a mismatch will cause problems when deploying an end product.

To program a device with a certificate, the device must be connected via the debug cable to a supported debug adapter, such as the Ember Debug Adapter (also called the ISA3) for EM3x-based devices or the Silicon Labs Debug Adapter Kit (P/N: SLSDA001A) for EFR32-based devices. The tools provided with the EM3x Utilities or Simplicity Commander installation provide the basic elements needed to do this. The tool used for EM3x platforms is **em3xx_load.exe**, while the tool used for EFR32 platforms is **commander.exe**. Both are installed as "adapter packs" by Simplicity Studio when you install the part support for the respective chip family.

6.3 Verifying the Software Version on an EM3x Chip

To verify that you have the correct version of the tools, open a command prompt, and change to the directory where the **em3xx_load.exe** tool is installed. The default for the Simplicity Studio installation of the EM3x Utilities adapter pack is:

```
C:/SiliconLabs/SimplicityStudio/v4/developer/adapter_packs/em3xx/utils
```

Note: For installations under Mac OS, the "v4" folder hierarchy will appear within the "Simplicity Studio.app" package.

To verify the software version for the **em3xx_load.exe** tool, execute the following command**:**

```
$ em3xx_load.exe
```

To support Smart Energy 1.0 certificates, you need to be running at least version **1.0b13**:

```
$ em3xx_load.exe
em3xx_load (Version 1.0b13.1256666220)
```

To support Smart Energy 1.2 certificates, you need to be running at least version **4.1b04**:

```
$ em3xx_load.exe

em3xx_load version 4.1b04
NAME
      em3xx_load - Programming utilities for EM3XX series chips


SYNOPSIS
       em3xx_load [TARGET OPTION] [MODIFIERS] COMMAND [COMMAND ARGUMENTS]
```

6.4 Verifying the Software Version for an EFR32

To verify that you have the correct version of the tools, open a command prompt, and change to the directory where Simplicity Commander is installed. The default for the Simplicity Studio installation of the Simplicity Commander adapter pack is:

```
C:/SiliconLabs/SimplicityStudio/v4/developer/adapter_packs/commander.
```

Note: For installations under Mac OS, the “v4” folder hierarchy will appear within the “Simplicity Studio.app” package.

To verify the software version for Simplicity Commander, execute the following command**:**

```
$ commander --version
```

You should see output similar to this:

```
Simplicity Commander 0v25p0b267

JLink DLL version: 6.14
EMDLL Version: 0v14p1b246
mbed TLS version: 2.2.0
```

7 Programming the Certificate into an EM3x Chip

7.1 Checking the Node Information

Prior to modifying the certificate it is best to verify there is connectivity with the device to be programmed, and what information is currently stored on the node. To do this, execute the following command:

```
$ em3xx_load.exe    --cibtokensprint
```

You should see the following output:

```
$ em3xx_load.exe --cibtokensprint

em3xx_load version 4.1b04
Connecting to ISA via IP address 10.4.176.51
DLL version 1.1.28, compiled Sep 25 2013 13:55:00
SerialWire interface selected
SWJCLK speed is 500kHz
Targeting EM3588

'General' token group
TOKEN_MFG_CIB_OBS                [16 byte array ] : A55AFFFFFFFFFFFF FFFFFFFFFFFFFFFF
TOKEN_MFG_CUSTOM_VERSION   [     16-bit integer] : 0xFFFF
TOKEN_MFG_CUSTOM_EUI_64          [ 8 byte array ] : FFFFFFFFFFFFFFFF
TOKEN_MFG_STRING                 [16  byte  string]  :  ""  (0  of  16  chars)  FFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFF
TOKEN_MFG_BOARD_NAME             [16  byte  string]  :  ""  (0  of  16  chars)  FFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFF
TOKEN_MFG_MANUF_ID               [16-bit integer] : 0xFFFF
TOKEN_MFG_PHY_CONFIG             [16-bit integer] : 0xFF26  TOKEN_MFG_BOOTLOAD_AES_KEY[16  byte  ar-
ray ] : FFFFFFFFFFFFFFFF FFFFFFFFFFFFFFFF
TOKEN_MFG_EZSP_STORAGE           [ 8 byte array ] : FFFFFFFFFFFFFFFF
TOKEN_MFG_OSC24M_BIAS_TRIM [16-bit integer] : 0xFFFF
TOKEN_MFG_SYNTH_FREQ_OFFSET      [16-bit integer] : 0xFFFF
TOKEN_MFG_OSC24M_SETTLE_DELAY    [16-bit integer] : 0xFFFF
TOKEN_MFG_SECURITY_CONFIG        [16-bit integer] : 0xFFFF
TOKEN_MFG_CCA_THRESHOLD          [16-bit integer] : 0xFFFF
TOKEN_MFG_SECURE_BOOTLOADER_KEY  [16 byte array ] : FFFFFFFFFFFFFFFF FFFFFFFFFFFFFFFF

'Smart Energy CBKE (TOKEN_MFG_CBKE_DATA)' token group
Device Implicit Cert             [48 byte array ] : FFFFFFFFFFFFFFFF FFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFF FFFFFFFFFFFFFFFF FFFFFFFFFFFFFFFF FFFFFFFFFFFFFFFF
CA Public Key             [22 byte array ] : FFFFFFFFFFFFFFFF FFFFFFFFFFFFFFFF
FFFFFFFFFFFF
Device Private Key               [21 byte array ] : FFFFFFFFFFFFFFFF FFFFFFFFFFFFFFFF
FFFFFFFFFF
CBKE Flags                       [1 byte array ] : FF

'Smart Energy Install Code (TOKEN_MFG_INSTALLATION_CODE)' token group Install Code Flags [ 2 byte
array ] : FFFF
Install Code              [16 byte array ] : FFFFFFFFFFFFFFFF FFFFFFFFFFFFFFFF
CRC                              [16-bit integer] : 0xFFFF

'Smart Energy 1.2 CBKE (TOKEN_MFG_CBKE_283K1_DATA)' token group
Device Implicit Cert (283k1)     [74 byte array ] : FFFFFFFFFFFFFFFF FFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFF FFFFFFFFFFFFFFFF FFFFFFFFFFFFFFFF FFFFFFFFFFFFFFFF FFFFFFFFFFFFFFFF FFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFF FFFF
CA Public Key (283k1)            [37 byte array ] : FFFFFFFFFFFFFFFF FFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFF FFFFFFFFFFFFFFFF FFFFFFFFF
Device Private Key (283k1) [36 byte array ] : FFFFFFFFFFFFFFFF FFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFF FFFFFFFFFFFFFFFF FFFFFFFF
CBKE FLAGS (283k1)               [ 1 byte array ] : FF
DONE
```

Note: The pre-programmed default EUI64 is not shown in this output as it is part of the Silicon Labs-programmed manufacturing data rather than the customer-programmed manufacturing data. To obtain the default EUI64 of a Silicon Labs device, use the following command (adding the appropriate `--ip` argument for Ethernet-connected adapters).

Executing the following command reads out the default Silicon Labs-programmed EUI64 in least-significant byte (LSB) notation:

```
em3xx_load --read @080407A2-080407A9
```

For example, the command above produces the following output (indicating a default EUI64 value of 0x000D6F000092E047):

```
em3xx_load (Version 1.0b13.1256666220)
Connecting to ISA via USB Device 0
DLL version 1.1.2, compiled Oct 09 2009
14:09:00 SerialWire interface selected
SWJCLK speed is 500kHz
Targeting EM357
Reset Chip
Getting memory from 0x080407A2 to 0x080407A9

{address:    0 1 2  3  4  5  6  7  8  9  A B C D E F}
080407A0:    . . 47 E0 92 00 00 6F 0D 00 . . . . . .

Run (by toggling nRESET) DONE
```

If the custom EUI64 is set to all F's, then the default EUI64 is used. If this value is not all F's, then it is used instead of the default EUI64.

If the data within the Smart Energy token group (certificate, private key, root CA public key, metadata) is set to all F's, then the data is not valid and will not be used by the software.

7.2 Format of the Certificate File

Note: If programming both a Smart Energy 1.0 and a Smart Energy 1.2 certificate the two files may be combined into one.

7.2.1 Smart Energy 1.0

To program a Smart Energy 1.0 certificate into the tokens, it is necessary to create a simple text file with the security information in it.

The following is the format of that file expected by the **em3xx_load.exe** tool for Smart Energy 1.0 certificates.

```
CA Public Key: <hex-string>
Device Implicit Cert: <hex-string> Device Private Key: <hex-string>
```

The following is a sample certificate file issued by the **Test CA**, for a device with a EUI64 of 0000000000000001 (big endian).

```
CA Public Key: 0200fde8a7f3d1084224962a4e7c54e69ac3f04da6b8
Device Implicit Cert:
03045fdfc8d85ffb8b3993cb72ddcaa55f00b3e87d6d0000000000000001544553545345434101090006
00000000000
Device Private Key: 00b8a900fcadebabbfa383b540fce9ed438395eaa7
```

Note: The code line starting with 'Device Implicit Cert:' and its value appears in the CA Public Key on a single line, not three lines as represented above.

7.2.2 Smart Energy 1.2

To program a Smart Energy 1.2 certificate into the tokens, it is necessary to create a simple text file with the security information in it.

The following is the format of that file expected by the **em3xx_load.exe** tool for Smart Energy 1.2 certificates.

```
CA Public Key (283k1): <hex-string>
Device Implicit Cert (283k1): <hex-string>
Device Private Key (283k1): <hex-string>
```

The following is a sample certificate file issued by the **Test CA**, for a device with a EUI64 of 0000000000000001 (big endian).

```
CA Public Key (283k1): 0207a445022d9f39f49bdc38380026a27a9e0a1799313ab28c5c1a1c6b605154db1dff6752
Device Implicit Cert (283k1):
00602a3c32e55a8acf0d08111213141516171800532 0c3a6ffffffff000000000000000001080201d
4d1887cd727e195300c9a11c4b6cd82d37f36381bc3707a05b8f01b73d236d3a59ee9
Device Private Key (283k1): 000071ab965eeedfa9ad08370fed7dc7b1e7d0940c9f3917a23b2cdb16afd2402ade31a5
```

Note: The lines with hex data on them are on the same line as the text strings and not on separate lines as shown above.

7.3 Writing the Certificate into the Manufacturing Area

To write the certificate into the manufacturing area, execute the following command:

```
$ em3xx_load.exe --cibtokenspatch sample_cert.txt
```

You should see the following output (Note: This assumes you are programming two certificates):

```
$ em3xx_load.exe --cibtokenspatch sample_cert.txt em3xx_load version 4.1b04
Connecting to ISA via IP address 10.4.176.51
DLL version 1.1.28, compiled Sep 25 2013 13:55:00 SerialWire interface selected
SWJCLK speed is 500kHz Targeting EM3588
Reset Chip

Writing to address 0x0808087E for token 'Device Implicit Cert' Writing to address 0x080808AE for token
'CA Public Key' Writing to address 0x080808C4 for token 'Device Private Key' Writing to address
0x080808D9 for token 'CBKE Flags'
Writing to address 0x0808090E for token 'Device Implicit Cert (283k1)' Writing to address 0x08080958
for token 'CA Public Key (283k1)' Writing to address 0x0808097D for token 'Device Private Key (283k1)'
Writing to address 0x080809A1 for token 'CBKE FLAGS (283k1)'

NOTE: Writing Custom EUI64 '0100000000000000' to match certificate. Address 0x08080812.

Create image file Install RAM image Verify RAM image Install Flash image Verify Flash image
Run (by toggling nRESET)
DONE
```

In addition to writing the certificate, this will update the custom EUI64 so that it matches the data in the certificate. This ensures that there is parity between the data in the certificate and the EUI64 used by the local device for its identity on the network.

7.4 Verifying the Stored Certificate

After writing the certificate, it is best to verify the information by executing the following command:

```
$ ./em3xx_load.exe  --cibtokensprint
You should see the following output:
$ ./em3xx_load.exe  --cibtokensprint em3xx_load version 4.1b04
Connecting to ISA via IP address 10.4.176.51
DLL version 1.1.28, compiled Sep 25 2013 13:55:00 SerialWire interface selected
SWJCLK speed is 500kHz Targeting EM3588


'General' token group
TOKEN_MFG_CIB_OBS          [16 byte array ] : A55AFFFFFFFFFFFF FFFFFFFFFFFFFFFF
```

```
TOKEN_MFG_CUSTOM_VERSION   [16-bit integer] : 0xFFFF TOKEN_MFG_CUSTOM_EUI_64     [ 8 byte array ] :
0100000000000000
TOKEN_MFG_STRING           [16 byte string] : "" (0 of 16 chars) FFFFFFFFFFFFFFFF FFFFFFFFFFFFFFFF
TOKEN_MFG_BOARD_NAME       [16 byte string] : "" (0 of 16 chars) FFFFFFFFFFFFFFFF FFFFFFFFFFFFFFFF
TOKEN_MFG_MANUF_ID         [16-bit integer] : 0xFFFF
TOKEN_MFG_PHY_CONFIG       [16-bit integer] : 0xFF26  TOKEN_MFG_BOOTLOAD_AES_KEY [16 byte array ] :
FFFFFFFFFFFFFFFF FFFFFFFFFFFFFFFF
TOKEN_MFG_EZSP_STORAGE     [ 8 byte array ] : FFFFFFFFFFFFFFFF
TOKEN_MFG_OSC24M_BIAS_TRIM [16-bit integer] : 0xFFFF
TOKEN_MFG_SYNTH_FREQ_OFFSET      [16-bit integer] : 0xFFFF
TOKEN_MFG_OSC24M_SETTLE_DELAY    [16-bit integer] : 0xFFFF
TOKEN_MFG_SECURITY_CONFIG        [16-bit integer] : 0xFFFF
TOKEN_MFG_CCA_THRESHOLD          [16-bit integer] : 0xFFFF
TOKEN_MFG_SECURE_BOOTLOADER_KEY  [16 byte array ] : FFFFFFFFFFFFFFFF FFFFFFFFFFFFFFFF

'Smart Energy CBKE (TOKEN_MFG_CBKE_DATA)' token group
Device Implicit Cert             [48 byte array ] :  03045FDFC8D85FFB 8B3993CB72DDCAA5
                                                     5F00B3E87D6D0000 0000000000015445
                                                     5354534543410109 0006000000000000
CA Public Key                    [22 byte array ] :  0200FDE8A7F3D108 4224962A4E7C54E6
                                                     9AC3F04DA6B8
Device Private Key               [21 byte array ] :  00B8A900FCADEBAB BFA383B540FCE9ED
                                                     438395EAA7
CBKE Flags                       [ 1 byte array ] :  00

'Smart Energy Install Code (TOKEN_MFG_INSTALLATION_CODE)' token group
Install Code Flags               [2 byte array ] : FFFF
Install Code                     [16 byte array ] : FFFFFFFFFFFFFFFF FFFFFFFFFFFFFFFF
CRC                              [16-bit integer] : 0xFFFF

'Smart Energy 1.2 CBKE (TOKEN_MFG_CBKE_283K1_DATA)' token group
Device Implicit Cert (283k1)     [74 byte array ] :  00602A3C32E55A8A CF0D081112131415
                                                     161718005320C3A6 FFFFFFFF00000000
                                                     00000001080201D4 D1887CD727E19530
                                                     0C9A11C4B6CD82D3 7F36381BC3707A05
                                                     B8F01B73D236D3A5 9EE9
CA Public Key (283k1)            [37 byte array ] :  0207A445022D9F39 F49BDC38380026A2
                                                     7A9E0A1799313AB2 8C5C1A1C6B605154 DB1DFF6752
Device Private Key (283k1)       [36 byte array ] :  000071AB965EEEDF A9AD08370FED7DC7
                                                     B1E7D0940C9F3917 A23B2CDB16AFD240 2ADE31A5
CBKE FLAGS (283k1)               [ 1 byte array ] : 00
DONE
```

8 Erasing the Certificate on an EM3x Chip

If the wrong device is programmed with a certificate, it is necessary to remove this security data from the device. To erase token data create an erase file listing the token name using **!ERASE!** as the token data. For example to erase an ECC 163k1 certificate:

```
CA Public Key: !ERASE!
Device Implicit Cert: !ERASE!
Device Private Key: !ERASE!
```

To erase an ECC 283k1 certificate:

```
CA Public Key (283k1): !ERASE!
Device Implicit Cert (283k1): !ERASE!
Device Private Key (283k1): !ERASE!
```

Note: You can erase both certificates simultaneously by putting the above lines in the same file.

To erase a sample certificate, execute this command:

```
$ em3xx_load.exe --cibtokenspatch sample_erase_cert.txt
```

You should see the following output:

```
$ em3xx_load.exe --cibtokenspatch sample_erase_cert.txt em3xx_load (Version 1.0b13.1256666220)

Connecting to ISA via USB Device 0
DLL version 1.1.2, compiled Oct 09 2009 14:09:00 SerialWire interface selected
SWJCLK speed is 500kHz Targeting EM3577
Reset Chip

Writing to address 0x0804087E for token 'Device Implicit Cert' Writing to address 0x080408AE for token
'CA Public Key' Writing to address 0x080408C4 for token 'Device Private Key' Writing to address
0x080408D9 for token 'CBKE Flags'

NOTE: Writing Custom EUI64 'FFFFFFFFFFFFFFFF' to match certificate. Address 0x08040812.

Create image file Install RAM image Verify RAM image
Install Flash image Verify Flash image
Run (by toggling nRESET)
DONE
```

9 Programming the Certificates into an EFR32

9.1 Checking the Node Information

Prior to modifying the certificate it is best to verify there is connectivity with the device to be programmed, and what information is currently stored on the node. The `tokendump` command prints manufacturing token data as key-value pairs. Simplicity Commander supports more than one group of tokens. In this example, the token group named 'znet' is used.

```
$ commander tokendump --tokengroup znet
```

You should see the following output:

```
#
# The token data can be in one of three main forms: byte-array, integer, or string.
# Byte-arrays are a series of hexadecimal numbers of the required length.
# Integers are BIG endian hexadecimal numbers.
# String data is a quoted set of ASCII characters.
#
# MFG_EMBER_EUI_64     : A8D417FEFF570B00
MFG_CUSTOM_VERSION   : 0xFFFF
MFG_CUSTOM_EUI_64    : FFFFFFFFFFFFFFFF
MFG_STRING           : ""
MFG_BOARD_NAME       : ""
MFG_MANUF_ID         : 0xFFFF
MFG_PHY_CONFIG       : 0xFFFF
MFG_SYNTH_FREQ_OFFSET: 0xFFFF
MFG_CCA_THRESHOLD    : 0xFFFF
MFG_EZSP_STORAGE     : FFFFFFFFFFFFFFFF
MFG_CTUNE            : 0xFFFF
MFG_XO_TUNE          : 0xFFFF
MFG_LOCKBITS_PLW     : 0x000000000000000000000000FFFFFFFF
MFG_LOCKBITS_CLW0    : 0xFFFFFFFF
MFG_LOCKBITS_MLW     : 0xFFFFFFFF
MFG_LOCKBITS_ULW     : 0xFFFFFFFF
MFG_LOCKBITS_DLW     : 0xFFFFFFFF
MFG_BOOTLOAD_AES_KEY : FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
MFG_SECURITY_CONFIG  : 0xFFFF

MFG_ASH_CONFIG[0]  : 0xFFFF
MFG_ASH_CONFIG[1]  : 0xFFFF
MFG_ASH_CONFIG[2]  : 0xFFFF
MFG_ASH_CONFIG[3]  : 0xFFFF
MFG_ASH_CONFIG[4]  : 0xFFFF
MFG_ASH_CONFIG[5]  : 0xFFFF
MFG_ASH_CONFIG[6]  : 0xFFFF
MFG_ASH_CONFIG[7]  : 0xFFFF
MFG_ASH_CONFIG[8]  : 0xFFFF
MFG_ASH_CONFIG[9]  : 0xFFFF
MFG_ASH_CONFIG[10] : 0xFFFF
MFG_ASH_CONFIG[11] : 0xFFFF
MFG_ASH_CONFIG[12] : 0xFFFF
MFG_ASH_CONFIG[13] : 0xFFFF
MFG_ASH_CONFIG[14] : 0xFFFF
MFG_ASH_CONFIG[15] : 0xFFFF
MFG_ASH_CONFIG[16] : 0xFFFF
MFG_ASH_CONFIG[17] : 0xFFFF
MFG_ASH_CONFIG[18] : 0xFFFF
MFG_ASH_CONFIG[19] : 0xFFFF

#'MFG_CBKE_DATA (Smart Energy CBKE)' token group
Device Implicit Cert :
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
CA Public Key        : FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
Device Private Key   : FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
# CBKE Flags           : 0xFF
```

```
#'MFG_INSTALLATION_CODE (Smart Energy Install Code)' token group
# Install Code Flags : 0xFFFF
Install Code       : FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
# CRC                : 0xFFFF

#'MFG_SECURE_BOOTLOADER_KEY (Manufacture token space for storing secure bootloader key.)' token
group
MFG_SECURE_BOOTLOADER_KEY : FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF

#'MFG_CBKE_283K1_DATA (Smart Energy 1.2 CBKE)' token group
Device Implicit Cert (283k1) :
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFpFFFFFFFFF
CA Public Key (283k1)        :
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
Device Private Key (283k1)   :
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
# CBKE FLAGS (283k1)           : 0xFF

#'MFG_SIGNED_BOOTLOADER_KEY_X (Manufacture token space for storing ECDSA signed bootloader key (X-
point).)' token group
MFG_SIGNED_BOOTLOADER_KEY_X : FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF

#'MFG_SIGNED_BOOTLOADER_KEY_Y (Manufacture token space for storing ECDSA signed bootloader key (Y-
point).)' token group
MFG_SIGNED_BOOTLOADER_KEY_Y : FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF

DONE
```

The pre-programmed EUI64 is read out by executing the following command.

```
commander tokendump --tokengroup znet --token MFG_EMBER_EUI_64
#
# The token data can be in one of three main forms: byte-array, integer, or string.
# Byte-arrays are a series of hexadecimal numbers of the required length.
# Integers are BIG endian hexadecimal numbers.
# String data is a quoted set of ASCII characters.
#
# MFG_EMBER_EUI_64: A8D417FEFF570B00

DONE
```

9.2 Format of the Certificate File

9.2.1 Smart Energy 1.0

Simplicity Commander uses the same text file format as the **em3xx_load.exe** tool (see **section 7.2.1, Smart Energy 1.0**). Note, however, that Simplicity Commander does not automatically update the EUI64 address of the device automatically, so you must specify this manually in the token file:

```
CA Public Key: 0200fde8a7f3d1084224962a4e7c54e69ac3f04da6b8
Device Implicit Cert:
03045fdfc8d85ffb8b3993cb72ddcaa55f00b3e87d6d000000000000000154455354534543410109000600000000000
Device Private Key: 00b8a900fcadebabbfa383b540fce9ed438395eaa7
MFG_CUSTOM_EUI_64: 0000000000000001
```

9.2.2 Smart Energy 1.2

Simplicity Commander uses the same text file format as the **em3xx_load.exe** tool (see **section 7.2.2, Smart Energy 1.2**). Note, however, that Simplicity Commander does not automatically update the EUI64 address of the device automatically, so you must specify this manually in the token file:

```
CA Public Key (283k1): 0207a445022d9f39f49bdc38380026a27a9e0a1799313ab28c5c1a1c6b605154db1dff6752
Device Implicit Cert (283k1):
00602a3c32e55a8acf0d08111213141516171800532 0c3a6ffffffff0000000000000000 1080201d4d1887cd727e195300c9
a11c4b6cd82d37f36381bc3707a05b8f01b73d236d3a59ee9
Device Private Key (283k1):
000071ab965eeedfa9ad08370fed7dc7b1e7d0940c9f3917a23b2cdb16afd2402ade31a5
MFG_CUSTOM_EUI_64: 0000000000000001
```

9.3 Writing the Certificate into the Manufacturing Area

Simplicity Commander does not automatically update the custom EUI64 address to match the certificate, so you need to make sure that these are in sync. This can be accomplished by adding it to the text file with the certificate (`sample_cert.txt` in the example).

```
$ commander flash --tokengroup znet --tokenfile sample_cert.txt
```

You should see the following output:

```
Writing 2048 bytes starting at address 0x0fe04000
Comparing range 0x0FE04000 - 0x0FE047FF (2 KB)
Programming range 0x0FE04240 - 0x0FE0435F (288 Bytes)
Verifying range 0x0FE04000 - 0x0FE047FF (2 KB)
DONE
```

9.4 Verifying the Stored Certificate

After writing the certificate, it is best to verify the information by executing the following command:

```
$ commander tokendump --tokengroup znet
#
# The token data can be in one of three main forms: byte-array, integer, or string.
# Byte-arrays are a series of hexadecimal numbers of the required length.
# Integers are BIG endian hexadecimal numbers.
# String data is a quoted set of ASCII characters.
#
# MFG_EMBER_EUI_64     : E33409FEFF570B00
MFG_CUSTOM_VERSION   : 0xFFFF
MFG_CUSTOM_EUI_64    : FFFFFFFFFFFFFFFF
MFG_STRING           : ""
MFG_BOARD_NAME       : ""
MFG_MANUF_ID         : 0xFFFF
MFG_PHY_CONFIG       : 0xFFFF
MFG_SYNTH_FREQ_OFFSET: 0xFFFF
MFG_CCA_THRESHOLD    : 0xFFFF
MFG_EZSP_STORAGE     : FFFFFFFFFFFFFFFF
MFG_CTUNE            : 0x7BF0
MFG_XO_TUNE          : 0xFFFF
MFG_LOCKBITS_PLW     : 0x00000000000000000000000FFFFFFFF
MFG_LOCKBITS_CLW0    : 0xFFFFFFFF
MFG_LOCKBITS_MLW     : 0xFFFFFFFF
MFG_LOCKBITS_ULW     : 0xFFFFFFFF
MFG_LOCKBITS_DLW     : 0xFFFFFFFF
MFG_BOOTLOAD_AES_KEY : FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
MFG_SECURITY_CONFIG  : 0xFFFF

MFG_ASH_CONFIG[0]  : 0xFFFF
MFG_ASH_CONFIG[1]  : 0xFFFF
MFG_ASH_CONFIG[2]  : 0xFFFF
MFG_ASH_CONFIG[3]  : 0xFFFF
MFG_ASH_CONFIG[4]  : 0xFFFF
```

```
MFG_ASH_CONFIG[5]  : 0xFFFF
MFG_ASH_CONFIG[6]  : 0xFFFF
MFG_ASH_CONFIG[7]  : 0xFFFF
MFG_ASH_CONFIG[8]  : 0xFFFF
MFG_ASH_CONFIG[9]  : 0xFFFF
MFG_ASH_CONFIG[10] : 0xFFFF
MFG_ASH_CONFIG[11] : 0xFFFF
MFG_ASH_CONFIG[12] : 0xFFFF
MFG_ASH_CONFIG[13] : 0xFFFF
MFG_ASH_CONFIG[14] : 0xFFFF
MFG_ASH_CONFIG[15] : 0xFFFF
MFG_ASH_CONFIG[16] : 0xFFFF
MFG_ASH_CONFIG[17] : 0xFFFF
MFG_ASH_CONFIG[18] : 0xFFFF
MFG_ASH_CONFIG[19] : 0xFFFF

#'MFG_CBKE_DATA (Smart Energy CBKE)' token group
Device Implicit Cert :
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
CA Public Key        : FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
Device Private Key   : FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
# CBKE Flags           : 0xFF

#'MFG_INSTALLATION_CODE (Smart Energy Install Code)' token group
# Install Code Flags : 0xFFFF
Install Code       : FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
# CRC                : 0xFFFF

#'MFG_SECURE_BOOTLOADER_KEY (Manufacture token space for storing secure bootloader key.)' token
group
MFG_SECURE_BOOTLOADER_KEY : FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF

#'MFG_CBKE_283K1_DATA (Smart Energy 1.2 CBKE)' token group
Device Implicit Cert (283k1) :
00602A3C32E55A8ACF0D0811121314151617180053 20C3A6FFFFFFFF00000000000000001080201D4D1887CD727E195300C9
A11C4B6CD82D37F36381BC3707A05B8F01B73D236D3A59EE9
CA Public Key (283k1)        :
0207A445022D9F39F49BDC38380026A27A9E0A1799313AB28C5C1A1C6B605154DB1DFF6752
Device Private Key (283k1)   :
000071AB965EEEDFA9AD08370FED7DC7B1E7D0940C9F3917A23B2CDB16AFD2402ADE31A5
# CBKE FLAGS (283k1)           : 0x00

#'MFG_SIGNED_BOOTLOADER_KEY_X (Manufacture token space for storing ECDSA signed bootloader key (X-
point).)' token group
MFG_SIGNED_BOOTLOADER_KEY_X : FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF

#'MFG_SIGNED_BOOTLOADER_KEY_Y (Manufacture token space for storing ECDSA signed bootloader key (Y-
point).)' token group
MFG_SIGNED_BOOTLOADER_KEY_Y : FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF

DONE
```

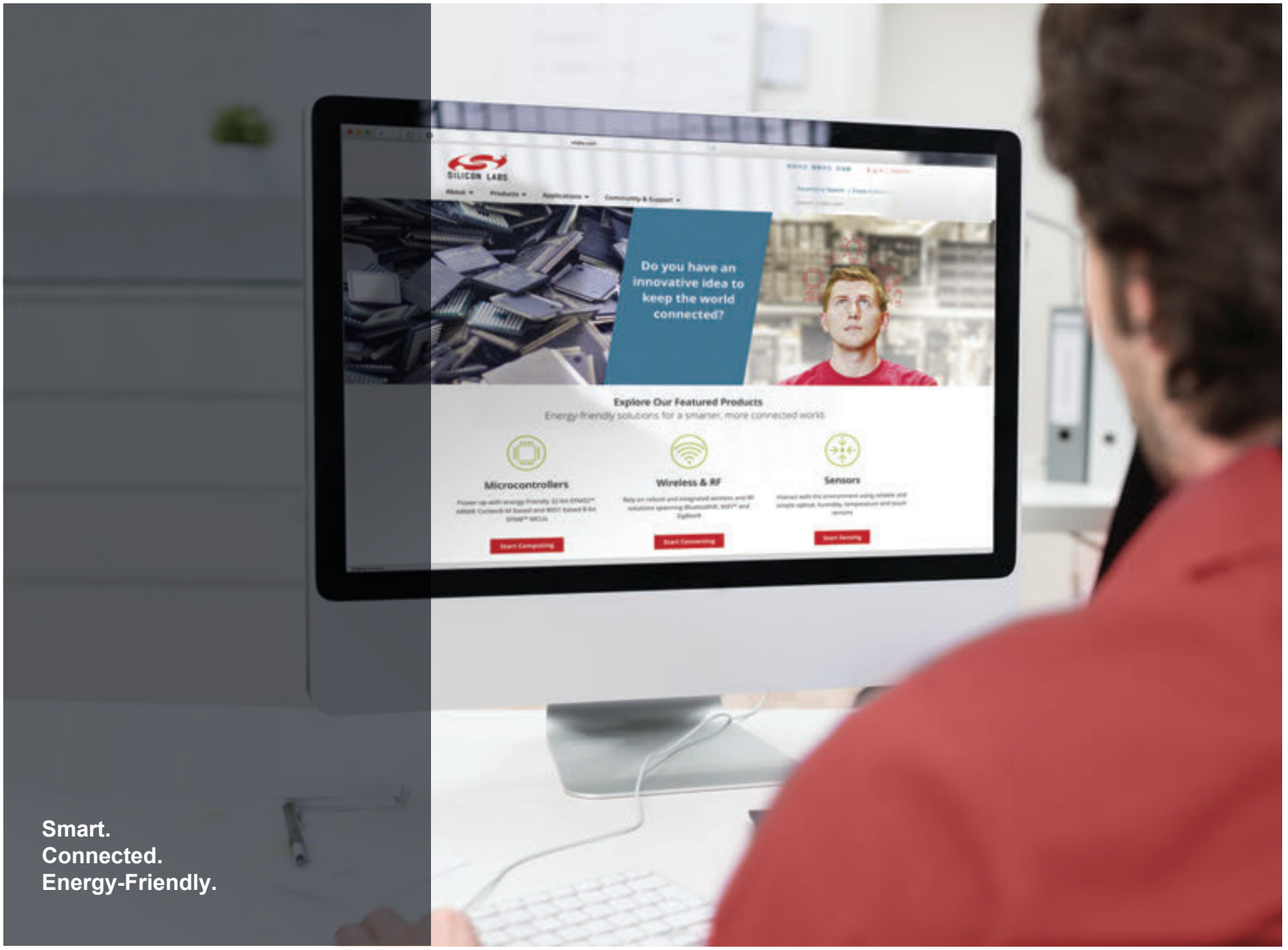
10 Erasing the Certificate on an EFR32

If the wrong device is programmed with a certificate, or it is necessary to remove this security data from the device. See **section 8, Erasing the Certificate on an EM3x Chip**, for more information on the contents of the sample_erase_cert.txt file. To erase a certificate, execute this command:

```
$ commander flash --tokenfile sample_erase_cert.txt --tokengroup znet
```

You should see the following output:

```
Writing 2048 bytes starting at address 0x0fe04000
Comparing range 0x0FE04000 - 0x0FE047FF (2 KB)
Erasing range 0x0FE04240 - 0x0FE0435F (288 Bytes)
Verifying range 0x0FE04000 - 0x0FE047FF (2 KB)
DONE
```



Products
www.silabs.com/products

Quality
www.silabs.com/quality

Support and Community
community.silabs.com

SILICON LABS®

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

http://www.silabs.com