



AN714: Smart Energy ECC-Enabled Device Setup Process

This document describes how to set up a device with the security resources required to support Smart Energy (SE) security, which is based on certificate-based key establishment (CBKE) using Elliptic-Curve Cryptography (ECC). This includes the ECC 163k1 curve used by Smart Energy 1.0 and 1.1, as well as the ECC 283k1 curve used by Smart Energy 1.2 and 1.4.

KEY POINTS

- Test certificate generation
- Certificate file modification
- Installation code generation and programming
- Application setup (build time)
- Application setup (run time)
- Joining process for new device
- Network analyzer capture setup

1. Introduction

The processes described in this application note are all those required to set up Smart Energy (SE) elliptic curve cryptography (ECC)-enabled devices. The processes include the following:

- Test certificate generation
- Certificate file manipulation
- Certificate file programming
- Installation code generation
- Installation code programming
- Application setup (build time)
- Application setup (run time) if using unique link keys
- Joining a new device
- Simplicity Studio Network Analyzer capture setup

While these security resources, which also include Certicom's ECC library, available upon request from Silicon Labs' support team, are not necessary for testing SE networks, any devices wishing to participate in or host a Zigbee-compliant, production-grade (non-test) SE network must implement these features.

Readers of this document should be familiar with the Zigbee Smart Energy Application profile (available from the <http://www.zigbee.org> website in the Standards section) and have basic familiarity with the Silicon Labs' graphical and command-line-based tools. A number of these processes use utilities available through Simplicity Studio's development environment, such as Network Analyzer, AppBuilder, and Simplicity Commander. This document also assumes that the user already has access to the Certicom ECC library for their target platform, or an ECC-enabled network coprocessor image, for EZSP-based platforms. For access to this content, please contact the Silicon Labs support team through the Silicon Labs Support portal at <https://www.silabs.com/support>.

Note: Because the ECC library is compiled with IAR, SE projects should also be compiled with IAR. Projects compiled with GCC may cause unexpected resets.

Smart Energy 1.0 utilized an ECC 163k1 curve with a 48-byte certificate format. All certified devices are required to support this. Smart Energy 1.2 introduces a new curve ECC 283k1, and a 74-byte certificate format. Smart Energy 1.2 devices **must** support the existing 163k1 ECC curve and may also support the new 283k1 curve, while Smart Energy 1.4 devices **must** also support the 283k1 curve. (The requirements for what devices must support the 238k1 ECC curve is spelled out in the Zigbee Smart Energy specification.)

The certificates and libraries are **not** interoperable. To support both curves, two sets of certificates, private keys, and CA public keys must be installed.

Note: Zigbee EmberZNet SDK 7.0 introduced a new component-based architecture, along with a Project Configurator and other tools to replace AppBuilder and plugin configuration. In general, the new software components are comparable to the plugins. Unless otherwise specified, the term "component" should be understood to refer to the corresponding plugin, if you are working in SDK 6.10.x or lower. When applicable, instructions for both version 7.0 and higher and 6.10.x and lower are provided. For more information, see *AN1301: Transitioning from Zigbee EmberZNet SDK 6.x to SDK 7.x*.

2. Test Certification Generation

1. Register for an account (for test certificate generation) at <http://www.certicom.com/index.php/gencertregister>. This registration is valid for a limited time.
2. Log in to the test certificate generation site with your account login:
3. <http://www.certicom.com/index.php/devicelogin>
4. Go to the certificate generation page:
 - ECC 163k1 curve-based Certificates: <http://www.certicom.com/index.php/smartenergydevicecertificateservice>
 - ECC 283k1 curve-based Certificates: <http://www.certicom.com/index.php/smartenergydevicecertificateservice1dot2>
5. In the Subject ID field, enter the MAC address (EUI64) of the target device where the certificate will reside (MSB order, such as 000D6F... for Silicon Labs EUI64s). This can be different than your target node's current EUI64, but if the address differs, the current one will generally be overwritten during programming.
6. ECC 163k1 curve-based Certificates:
 - a. In the **Profile Attribute Data** field, enter any custom hex data you want associated with this device. (The recommended format is profile ID [0109] + cert type [1 for test] + 16-bit Zigbee Manufacturer Code + optional customer data of any length.)
 - b. Leave the **Public Key** field empty!
7. ECC 283k1 curve-based Certificates:
 - a. Leave the Lifetime field empty.
 - b. Check the **Key Agreement** checkbox.
 - c. Leave the Override Certificate Attributes field empty.
 - d. Leave the Public Key Contribution field empty.
8. Click **Generate**.
9. Highlight and copy the text in the HTML table on the resulting page.
10. Open a text editor and paste the contents of the table into the text file; save the file.

3. Certificate File Modification

3.1 Combined Certificate File

If you intend to install both certificates onto a device, you may combine the contents of the certificate files into a single file.

Note: This portion of the process is subject to change if Certicom's certificate generation web page alters its output format. Silicon Labs is working with Certicom in the hope of achieving a more consistent output format in the future, one which would require less manual editing before programming.

3.2 ECC 163k1 Curve

At the beginning of this process, the certificate text file you created in the previous step should look something like this (with different hex numbers in the fields):

```
CA Pub Key: 0200fde8a7f3d1084224962a4e7c54e69ac3f04da6b8
Device Implicit Cert: 02040e92886475999701a626a75be9cfb9ccb0ee7481
000d6f000092e047544553545345434101091083d1bbd1bbd1bb
Device Private Key: 00ea5606dec9ec8f85a5f53405b67b2988b74bcd73
Device Public Key: 0202f25c9e4aaf910d6bbc16f444715d39962e3a5500
```

1. Remove "Device Public Key" line from the certificate text file, as our tools do not use it.
2. Remove extra line breaks in the hex string following "Device Implicit Cert:" so that all characters are on one line.
3. Change "CA Pub Key" to "CA Public Key"
4. Make sure that only a single blank space is between the ":" and the hex string of each line (no tabs or extra white space).
5. Remove the extra blank space at the end of each line before the line return.

At the end of the process, the certificate text file should look like the following, with the Device Implicit Cert line on a single line:

```
CA Public Key: 0200fde8a7f3d1084224962a4e7c54e69ac3f04da6b8
Device Implicit Cert:
02040e92886475999701a626a75be9cfb9ccb0ee7481000d6f000092e047544553545345434101091083d1bbd1bbd1bb
Device Private Key: 00ea5606dec9ec8f85a5f53405b67b2988b74bcd73
```

3.3 ECC 283k1 Curve

At the beginning of this process the certificate text file you created in the previous step should look something like this (with different hex numbers in the fields):

```
CA Pub Key: 0207a445022d9f39f49bdc38380026a27a9e0a1799313ab28c5c1a1c6b605154db1dff6752
Device Implicit Cert:
00e4d6197cf4b7b87e0d081112131415161718005320b5f6ffffffffff00000000000000000900
0200849946d913e09111970ff24bac05605fecaf258dfc1c586de5690de8b8d11b393a1b69
Device Private Key: 01817e7b640c0833c3daa12cf57284e959551455246a0cbcf876955c4492a0416350255d
Device Public Key: 02064af594acdee3c5cc41086c3f72f5b3ef6197cee9726e98d78227b4e0401cf08df0781d
```

1. Remove "Device Public Key" line from the certificate text file, as our tools do not use it.
2. Remove extra line breaks in the hex string following "Device Implicit Cert:" so that all characters are on one line.
3. Change "CA Pub Key" to "CA Public Key (283k1)"
4. Change "Device Implicit Cert" to "Device Implicit Cert (283k1)"
5. Change "Device Private Key" to "Device Private Key (283k1)"
6. Make sure that only a single blank space is between the ":" and the hex string of each line (no tabs or extra white space).
7. Remove the extra blank space at the end of each line before the line return.

```
CA Public Key (283k1): 0207a445022d9f39f49bdc38380026a27a9e0a1799313ab28c5c1a1c6b605154db1dff6752
Device Implicit Cert (283k1):
00e4d6197cf4b7b87e0d081112131415161718005320b5f6ffffffffff00000000000009000200849946d913e09111970ff24bac05605fecaf2
Device Private Key (283k1):
01817e7b640c0833c3daa12cf57284e959551455246a0cbcf876955c4492a0416350255d
```

3.4 Certificate Programming

1. Ensure that Simplicity Commander (version 0.15 or higher) is installed.
2. Change to the directory where the certificate text file is stored.
3. Determine the serial number or IP address of the target node's WSTK main board. This is visible from on the device's LCD display or can be determined by using the following command and selecting the node of interest from the pop-up window:

```
commander adapter probe
```

4. Run either of the following command to apply the certificate (shown for Serial Number 440056147 and IP address 192.168.0.1 and certificate filename cert_001.txt):

```
commander flash --tokengroup znet --tokenfile cert_001.txt --serialno 440056147  
commander flash --tokengroup znet --tokenfile cert_001.txt --ip 192.168.0.1
```

5. Run either of the following commands to verify the certificate data:

```
commander tokendump --tokengroup znet --serialno 440056147  
commander tokendump --tokengroup znet --ip 192.168.0.1
```

4. Installation Code Generation

1. Pick a random hex string of 6, 8, 12, or 16 bytes. You can generate this programmatically with a random number generator if you wish.
2. Open a text editor and enter this string (as ASCII) in a file by itself, preceded by the string "Install Code: ". Note the single white-space character following the colon.
3. Save this file for later use.

5. Installation Code Programming

Note: This is not necessary for devices that only form SE networks (like an Energy Services Interface (ESI), which is typically the Trust Center for the network) rather than join them.

1. Change to the directory where the certificate text file is stored.
2. Determine the serial number or IP address of the target node's WSTK main board. This is visible from on the device's LCD display or can be determined by using the following command and selecting the node of interest from the pop-up window:

```
commander adapter probe
```

3. Run either of the following commands to apply the installation cod (shown for Serial Number 440056147 and IP address 192.168.0.1 and certificate filename cert_001.txt):

```
commander flash --tokengroup znet --tokenfile inst_001.txt --serialno 440056147  
commander flash --tokengroup znet --tokenfile inst_001.txt --ip 192.168.0.1
```

4. Run either of the following commands to verify the installation code:

```
commander tokendump --tokengroup znet --serialno 440056147  
commander tokendump --tokengroup znet --ip 192.168.0.1
```

6. Application Setup (Build Time)

6.1 In Zigbee EmberZNet SDK 6.x

6.1.1 Common

This section deals with instructions that are common to both System-on-SoC (SoC) and Host application configurations. Further details specific to each type of configuration are covered in the sections that follow. Note that a sample application for the Dual Band Coom-sHub is available on request from Silabs support. It will be delivered as a side package for the stack. However, the steps in this section assume that you have wish to modify one of the example applications.

As a prerequisite to setting up an ECC-enabled device, first ensure that you have the ECC-enabled software content package for your EmberZNet PRO release. Access is granted by Silicon Labs Support upon request via the support portal, and the content packages can be found as ZIP files accompanying each EmberZNet PRO release in the Software Releases area of the Silicon Labs support portal. Once you've downloaded and extracted the content package, you are ready to follow the steps below.

Note: Because the ECC library is compiled with IAR, SE projects should also be compiled with IAR. Projects compiled with GCC may cause unexpected resets.

1. Create a new project, as described in *QSG106: Zigbee EmberZNet PRO Quick-Start Guide*.
2. On the Zigbee Stack tab, set one of the following:
 - If you are using unique, per-device link keys based on installation codes for joining devices, as should be the case for production deployments, set the **Security** option to "Smart Energy Security full (compliant)".
 - If you are using a single, global link key for all devices to join, as is often used in development/testing scenarios to reduce complexity, set the **Security** option to "Smart Energy Security test".
3. Follow the directions in one of the sub-sections below for building either a System-on-Chip application, a Custom Network Coprocessor (NCP), or an EZSP Host application with ECC support.
4. After completing the steps as per the subsections below, set up the remaining device configuration as appropriate, and generate the project.
5. Populate callbacks in the generated project as necessary, and then build and load to the target device (after the certificate and installation codes have been programmed as described above).

Note: The certificate and installation code data only reside on the processor where the stack is being run, which is either the SoC processor itself in the SoC use case or the NCP, *not* the host processor, in the EZSP-enabled host/NCP use case.

6.1.2 System on Chip (SoC)

Use these steps to configure an SoC-based Smart Energy application with ECC support.

1. On the Plugins tab, in the Stack Libraries section, confirm the following settings:
 - Enable the **CBKE 163k Library** plugin.
 - If you intend to use the 283k1 ECC Curve (Smart Energy 1.2 or higher), then enable the **Use CBKE 283k Library** plugin.
 - If supporting OTA upgrades with image signature verification, enable the **CBKE DSA Verify Library** and, if supporting the 283k1 ECC curve, also enable the **CBKE 283k1 DSA Verify Library**.
 - Enable the **CBKE Core Library** plugin.
 - Enable the **ECC 163k1 Library** plugin, and then, in the **Library path** text entry box for the plugin options, enter the file path to the location of your non-283k1 ECC library file, typically named something like `ecc-library- $\{platformName\}$.a`.
 - Enable the **ECC 283k1 Library** plugin, and then, in the **Library path** text entry box for the plugin options, enter the file path to the location of your 283k1 ECC library file, typically named something like `ecc-library-283k1- $\{platformName\}$.a`.
 - Enable the **Security Core Library** plugin.
 - Enable the **Security Link Keys Library** plugins and set the **Link Key Table Size** according to your needs in the options area for this plugin.
2. In the Plugins tab, in the Smart Energy section, enable the **Key Establishment** plugin, and then, in the Options area, select the appropriate choice in the **SE Version** list depending on whether your device should behave in an SE 1.2 fashion (in which both 163k1 and 283k1 ECC curves are required for CBKE) or a pre-1.2 SE 1.x fashion (in which only 163k1 ECC support is used for CBKE).

Note: Some older versions of AppBuilder do not support white space in the ECC library path, so you may need to relocate these files to satisfy this requirement.

Then enable the **Smart Energy registration** plugin. This will take care of the CBKE (certificate-based key establishment) process that will determine the link key.

6.1.3 Custom Network Coprocessor

Use these steps to configure a custom EmberZNet PRO network coprocessor (NCP) via the Customizable Network Coprocessor Applications framework. Refer to AN1010, *Building a Customized NCP Application*, for more information about custom NCP design using this framework. Note that this is not the same as configuring a host application (which talks to an NCP via EZSP) for a Smart Energy device. That process is covered in the next section, [6.1.4 EZSP Host](#).

There are different versions of the network coprocessor with and without ECC support compiled in. Access to the ECC version is granted by Silicon Labs Support upon request. The ECC version of the network coprocessor firmware contains support for both ECC 163k1 and ECC 283k1 curves. However, the curves can be utilized depend on the certificates that are installed on the coprocessor.

When configuring a custom NCP to support ECC-enabled Smart Energy operation, do the following in AppBuilder, Plugins tab:

1. In the Stack Libraries section, enable the same library plugins and similar options as described in the section [6.1.2 System on Chip \(SoC\)](#). However, for libraries that are not being enabled, you must enable the corresponding “Stub Library” plugin, such as the **CBKE 283k1 DSA Verify Stub Library** plugin as a substitute for the **CBKE 283k1 DSA Verify Library** plugin.
2. In the Command Handler section, enable the **CBKE Library EZSP Command Handlers** plugin.

6.1.4 EZSP Host

Use these steps to configure an ECC-enabled host application, which is designed to talk to an ECC-enabled EmberZNet PRO NCP via EZSP. In AppBuilder, on the Plugins tab:

1. In the Smart Energy section, enable the **Key Establishment** plugin, and then, in the options area, select the appropriate choice in the **SE Version** list depending on whether your device should behave in an SE 1.2 or later fashion (in which both 162k1 and 283k1 ECC curves are required for CBKE) or a pre-1.2 SE 1.x fashion (in which only 163k1 ECC support is used for CBKE).
2. In the Utility section, enable the **NCP Configuration** plugin, and then, in the options area, set the **Link Key Table Size** according to your needs.

Note: This table size setting cannot exceed the maximum link key table size configured within your NCP firmware. If it does, an error will be detected and reported in the standard output stream of the host application during NCP initialization. If necessary, you can adjust the link key table size of the NCP using the Custom Network Processor method described in the previous section.

6.2 In Zigbee EmberZNet SDK 7.x

6.2.1 Common

This section deals with instructions that are common to both System-on-SoC (SoC) and Host application configurations. Further details specific to each type of configuration are covered in the sections that follow. Note that a sample of Smart Energy applications are available on request from Silicon Labs Support. It will be delivered as a side package for the stack. However, the steps in this section assume that you wish to modify one of the example applications.

As a prerequisite to setting up an ECC-enabled device, first ensure that you have the ECC-enabled software content package for your EmberZNet PRO release. Access is granted by Silicon Labs Support upon request via the support portal, and the content packages can be found as ZIP files accompanying each EmberZNet PRO release in the Software Releases area of the Silicon Labs Support portal. The side package contains Smart Energy sample applications and also ECC curves support, which needs to be extracted within the installed SDK. After you have downloaded and extracted the content package, you are ready to follow the steps below.

Note: Because the ECC library is compiled with IAR, SE projects should also be compiled with IAR. Projects compiled with GCC may cause unexpected resets.

1. Create a new project, as described in *QSG106: Zigbee EmberZNet PRO Quick-Start Guide*.
2. On the Software Components tab, set one of the following:
 - If you are using unique, per-device link keys based on installation codes for joining devices, as should be the case for production deployments, configure the **Primary Network Security Type** option to "Smart Energy Security full (compliant)" in the **Zigbee device config** component.
 - If you are using a single, global link key for all devices to join, as is often used in development/testing scenarios to reduce complexity, configure the **Primary Network Security Type** option to "Smart Energy Security test" in the **Zigbee device config** component.
3. Follow the directions in one of the sub-sections below for building either a System-on-Chip application, a Custom Network Coprocessor (NCP), or an EZSP Host application with ECC support.
4. After completing the steps as per the subsections below, set up the remaining device configuration as appropriate, and save the project.
5. Populate callbacks in the generated project as necessary, and then build and load to the target device (after the certificate and installation codes have been programmed as described above).

Note: The certificate and installation code data only reside on the processor where the stack is being run, which is either the SoC processor itself in the SoC use case or the NCP, *not* the host processor, in the EZSP-enabled host/NCP use case.

6.2.2 System on Chip (SoC)

Use these steps to configure an SoC-based Smart Energy application with ECC support.

1. On the Software Components tab, in the Zigbee Stack component category section, confirm the following settings:
 - Enable **CBKE 163k1** and/or **CBKE 283k1** component(s) as intended for the application.
 - Enable the **ECC 163k1 Support** component.
 - If you intend to use the 283k1 ECC Curve (Smart Energy 1.2 or higher), then enable the **CBKE 283k1 Support** component.
 - If supporting OTA upgrades with image signature verification, enable the **CBKE DSA Verify** component and, if supporting the 283k1 ECC curve, also enable the **CBKE 283k1 DSA Verify** component.
 - Enable the **CBKE Core** component.
 - Enable the **ECC 163k1 Support** component. This component, `ecc-163k1-library.a`, is located in `protocol/zigbee/build/ECC` from the installed SDK location.
 - Enable the **ECC 283k1 Support** component. This component, `ecc-283k1-library.a`, is located in `protocol/zigbee/build/ECC` from the installed SDK location.
 - Enable the **Security Link Keys** component and configure the **Link Key Table Size** according to your needs.
2. In the Software Components tab, in the Smart Energy component category, enable the **Key Establishment** component. This component defaults to use CBKE functions with 163k1 ECC curve support to behave as per pre-1.2 SE fashion, but you can also choose the 283k1 ECC curve support by selecting **CBKE 283k1** component. Select the appropriate CBKE component(s) depending on whether your device should behave in an SE 1.2 fashion (in which both 163k1 and 283k1 ECC curves are required for CBKE) or a pre-1.2 SE 1.x fashion (in which only 163k1 ECC support is used for CBKE).

Note: If you select the appropriate CBKE component, then all the required components should be enabled as a part of dependency except the DSA Verify components. However, Silicon Labs recommends to visit components to make sure they are enabled and to change the default configurations to your needs.

Then enable the **Smart Energy Registration** component. This will take care of the CBKE (certificate-based key establishment) process that will determine the link key.

6.2.3 Custom Network Coprocessor

Use these steps to configure a custom EmberZNet PRO network coprocessor (NCP) application. Refer to *AN1320: Building a Customized NCP Application* for more information about custom NCP design using this framework. Note that this is not the same as configuring a host application (which talks to an NCP via EZSP) for a Smart Energy device. That process is covered in the next section, [6.2.4 EZSP Host](#).

When configuring a custom NCP to support ECC-enabled Smart Energy operation, do the following in Simplicity Studio, Software components tab:

1. In the Zigbee Stack Component category section, enable the same stack components and similar options as described in the section [6.2.2 System on Chip \(SoC\)](#).

Note: For reference, you can use the 'ncp-uart-hw' sample app from the SDK and make it an ECC-enabled NCP app for Smart Energy operations by adding required components described in this section.

6.2.4 EZSP Host

Use these steps to configure an ECC-enabled host application, which is designed to talk to an ECC-enabled EmberZNet PRO NCP via EZSP. In Simplicity Studio, on the Software components tab:

1. In the Smart Energy component category, enable the **Key Establishment** component. This component defaults to use CBKE functions with 163k1 ECC curve support to behave as per pre-1.2 SE fashion, but you can also choose the 283k1 ECC curve support by selecting the **CBKE 283k1** component. Select the appropriate CBKE component(s) depending on whether your device should behave in an SE 1.2 fashion (in which both 163k1 and 283k1 ECC curves are required for CBKE) or a pre-1.2 SE 1.x fashion (in which only 163k1 ECC support is used for CBKE).
2. In the Zigbee Utility component category, enable the **NCP Configuration** component, and then click the **Configure** button and set the **Link Key Table Size** according to your needs.

Note: This table size setting cannot exceed the maximum link key table size configured within your NCP firmware. If it does, an error will be detected and reported in the standard output stream of the host application during NCP initialization. If necessary, you can adjust the link key table size of the NCP using the Custom Network Processor method described in the previous section.

7. Application Setup (Run Time) Prior to Joining a New Device to the HAN

Note: This process is only required if you are using installation codes to generate unique, per-device link keys, such as when "Smart Energy Security (full)" is selected as the security model for your application configuration in AppBuilder.) If you are using a global link key for joining, such as when using AppBuilder's "Smart Energy Security (test)" security model, skip this section.

This section describes the process of setting up the Trust Center device (the network coordinator) for the SE network to accept an incoming SE device into its home area network (HAN). It also describes the process of setting up a new HAN device prior to joining the HAN created by this Trust Center.

7.1 Procedure for Development Prototypes / Debugging

This process relies on the serial command line interface (CLI) to the Zigbee application framework. If the CLI is no longer supported or accessible on your network's Trust Center or incoming HAN device, please refer to section 7.2 Procedure for Production/Field Deployments.

1. Before joining a HAN device to the SE network, determine its installation code string, including its 16-bit CRC (in LSB order), as well as its EUI64 (MAC address). In a development/debugging environment, you can do this using the Simplicity Commander tool with the appropriate "tokendump" option to output the installation code data. See the section 5. Installation Code Programming for more information.
2. Use the "option install-code ..." CLI command at the Trust Center device during the development/debugging phase. The application framework then verifies the CRC for the provided installation code, computes the device-specific link key from this code, and then adds the hashed key result to the specified key table index on the Trust Center for the specified EUI64.
3. Confirm that the proper key table entry now exists and is displayed in the output of the "keys print" command at the trust center.

7.2 Procedure for Production/Field Deployments

1. Before joining a HAN device to the SE network, determine its installation code string, including its 16-bit CRC (in LSB order), as well as its EUI64 (MAC address); these byte values are meant to be published externally with the device for use during installation.
2. Using some out-of-band (non-Zigbee) method, such as verbally or through some proprietary communications interface, convey the HAN device's installation code to the ESI for the HAN, its trust center device (if different from the ESI), or some head-end device in the utility's backhaul network that has access to the HAN's ESI.
3. Have the ESI (or the utility's head-end) sanity-check the provided installation code by computing the CRC of those hex bytes (less the final two, which are the provided CRC16 from the installer) using the CRC16 algorithm described in the "CRC Algorithm Information" section (section 5.4.8.1.1.1 in the current [r15] version) of the Zigbee Smart Energy Application Profile Specification (Zigbee document 075356). The computed CRC (when converted into LSB order) for the 6/8/12/16-digit code should match the last 4 digits of the provided installation code string.
4. Once the installation code string (variable-length code + 2-byte CRC16 in LSB order) has been verified, have the ESI or its head-end compute the device-specific initial link key by performing an AES MMO hash function against the entire 8/10/14/18-byte string, using the algorithm described in the "MMO Hash Code Example" section (section 5.4.8.1.2.1 in the current [r15] version) of the ZigBee Smart Energy Application Profile Specification (ZigBee document 075356).

8. Joining Process for New Device

If you are using installation codes to create unique keys, the application setup process discussed in the previous section [7. Application Setup \(Run Time\) Prior to Joining a New Device to the HAN](#) should have been performed.

1. Join the new HAN device to the ESI's HAN (using the Button Form/Join component or "network join ..." CLI command or `emberAfStartSearchForJoinableNetwork` API function, for example).
2. The ESI (as Trust Center) will trigger its `emberAfTrustCenterJoinCallback()` to indicate the outcome of the initial joining process. If successful, the joining device should begin CBKE.
3. If both the ESI and HAN devices support CBKE and ECC and have valid certificates issued by the same authority, CBKE should succeed, and the HAN device should move on to registration (asking the ESI to bind to it on specific clusters). When the SE registration process completes, the HAN device will trigger its `emberAfRegistrationCallback()` to notify the application.
4. After the SE registration completes successfully, the HAN device can now send/receive messages to/from the ESI using its CBKE-authorized link key.

Note that steps 2 through 4 are automatically handled by the **Smart Energy Registration** component.

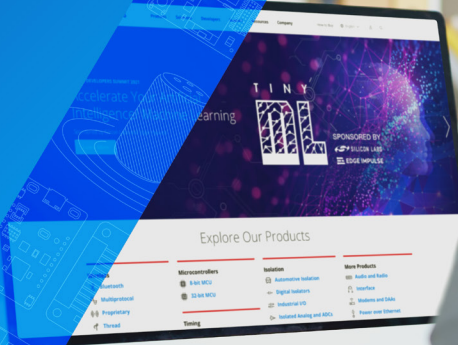
9. Network Analyzer Capture Setup

To capture the initial joining process before CBKE, first determine the device's preconfigured link key (either the unique one generated from the installation code as described in the section [7. Application Setup \(Run Time\) Prior to Joining a New Device to the HAN](#) or the global one if using SE Security Test mode). Enter this code into Network Analyzer using either of the two methods described below before attempting to join the HAN.

To capture the encrypted SE traffic after CBKE/registration has been performed, determine the CBKE-authorized link key by querying either the HAN device or the ESI. If the serial CLI is still available, you can use the "keys print" command. If you are using the stack API directly, use the `emberGetKey()` API. Enter the discovered key into Network Analyzer (see the two methods below) after CBKE completes; remember that this key is unique for each instance of CBKE, even for the same pair of devices.

You can enter the key into Network Analyzer through the main list of keys at Window > Preferences > Network Analyzer > Decoding > Security Keys. Keys must be entered here before beginning a capture session, else they will not be used for decryption.

Smart. Connected. Energy-Friendly.



IoT Portfolio
www.silabs.com/products



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