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.

KEY FEATURES

- 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 (for EM3xx and EFR32MG platforms)
- Installation code generation
- Installation code programming (for EM3xx and EFR32MG platforms)
- 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. 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: A number of these processes use utilities available through Simplicity Studio's development environments. New customers should access Network Analyzer and AppBuilder through Simplicity Studio. Simplicity Studio should also work with stack versions 5.4.x and 5.6.0. Otherwise, for stack versions earlier than 5.7.0, use Ember Desktop. Make sure that the EmberZNet PRO stack you are using has been added to AppBuilder's list of included stacks, as described in QSG106, *Getting Started with EmberZNet PRO*.

Similarly, there are two command-line utilities to be used for different architectures. For EM3xx and EM3xxx parts, use em35x_load; for EFR32MG parts, use Simplicity Commander. The commands for these utilities differ, so both will be provided in this document.

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. (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. In order to support both curves, two sets of certificates, private keys, and CA public keys must be installed.

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:
    1. Smart Energy 1.0 Certificates: http://www.certicom.com/index.php/smartenergydevicecertificateservice
    2. Smart Energy 1.2 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. Smart Energy 1.0 Certificates:
    1. 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.)
7. Leave the **Public Key** field empty!
8. Smart Energy 1.2 Certificates:
    1. Leave the Lifetime field empty
    2. Check the **Key Agreement** checkbox
    3. Leave the Override Certificate Attributes field empty.
    4. Leave the Public Key Contribution field empty.
9. Click **Generate**.
10. Highlight and copy the text in the HTML table on the resulting page.
11. 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 Smart Energy 1.0

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
000d6f000092e0475445535453454341010910 83d1bbd1bbd1bb
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:
02040e92886475999701a626a75be9cfb9ccb0ee7481000d6f000092e0475445535453454341010910 83d1bbd1bbd1bb
Device Private Key: 00ea5606dec9ec8f85a5f53405b67b2988b74bcd73
```

3.3 Smart Energy 1.2

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:
00e4d6197cf4b7b87e0d0811121314151617180053 20b5f6ffffffff000000000000000900
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):
00e4d6197cf4b7b87e0d08111213141516171800532 0b5f6ffffffff0000000000000000900020084994 6d913e09111970ff
24bac05605fecaf258dfc1c586de5690de8b8d11b393a1b69
Device Private Key (283k1):
01817e7b640c0833c3daa12cf57284e959551455246a0cbcf876955c4492a0416350255d
```

3.4 EM3xx Platform

1. Ensure that the Debug Adapter (ISA3) Utilities (version 1.0.2 or higher) are installed. If you want to verify, you can check your computer's Programs list for "ISA3 Utilities".
2. Change to the directory where the certificate text file is stored.
3. Determine the IP address of the target node's Debug Adapter (ISA3). You can use the Network Analyzer adapter properties window to view the IP address and other Debug Adapter (ISA3)/node details.
4. Run the following command to apply the certificate (shown for IP address 192.168.0.1 and certificate filename cert_001.txt):

```
em3xx_load --ip 192.168.0.1 --cibtokenspatch cert_001.txt
```

Note: If you are overriding previously written certificate data, you may need to add an "--Override" flag before the "--cibtokenspatch" flag. However, use this option with caution, as it can permanently corrupt your device if the flash-writing process is terminated unexpectedly.

5. Optionally, verify written certificate data with a command like the following:

```
em3xx_load --ip 192.168.0.1 –-cibtokensprint
```

3.5 EFR32MG Platform

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 whitespace 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.

5.1 EM3xx Platform

1. Change to the directory where the installation code file is stored.
2. Determine the IP address of the target node's Debug Adapter (ISA3). You can use Network Analyzer's adapter properties window to view the IP address and other Debug Adapter (ISA3)/node details.
3. Run the following command to apply the installation code (shown for IP address 192.168.0.1 and installation code filename inst_001.txt):

```
em3xx_load --ip 192.168.0.1 --cibtokenspatch inst_001.txt
```

Note: If you are overriding previously written installation code data, you may need to add an "--Override" flag before the "--cibtokenspatch" flag. However, use this option with caution, as it can permanently corrupt your device if the flash-writing process is terminated unexpectedly.

4. Optionally, verify written installation code data with a command like the following:

```
em3xx_load --ip 192.168.0.1 –-cibtokensprint
```

5.2 EFR32MG Platform

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 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 sample AppBuilder configuration scenarios, which include sample callback code for demonstrating the functionality, are provided for several common Smart Energy device types already in your EmberZNet PRO release, so you can refer to these as examples to illustrate proper application setup for these kinds of devices. These application scenarios are offered as alternatives to “Start with a blank application” during new AppBuilder configuration creation and begin with the prefix, “SeSample”, such as “SeSampleCommsHub”, “SeSampleEsiSoc” or “SeSample GSME”. However, the steps in this section assume that you have chosen “Start with a blank application” or wish to modify one of the aforementioned examples.

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.

1. Create a new Silicon Labs AppBuilder Project for your desired EmberZNet framework from Simplicity Studio’s File | New | Project menu.
2. On the Znet 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".
1. 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.
2. After completing the steps as per the subsections below, set up the remaining device configuration as appropriate, and generate the project.
3. Populate callbacks in the generated project as necessary, 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 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 EmberZNet 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), 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 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).

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.

6.3 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, **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 Smart Energy 1.0 and Smart Energy 1.2 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 EmberZNet Libraries section, enable the same library plugins and similar options as described in the section **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.

6.4 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 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.

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 application framework V2. If the CLI is no longer supported or accessible on your network's Trust Center or incoming HAN device, please refer to section **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 em3xx_load tools with the appropriate "print" option to output the installation code data. See the section **Installation Code Programming** section for more information.
2. Use the MMO hash algorithm to compute the device-specific link key from the installation code. Devices running EmberZNet PRO 4.3.0 and later can perform an AES MMO hash operation using the `emberAesHashSimple()` method provided on SoC platforms or the `ezspAesMmoHash()` API provided on EZSP host platforms. Devices running EmberZNet PRO 4.7.0 and later can utilize the "option install-code …" CLI command at the Trust Center device as an alternative 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, thereby allowing you to skip step 3.
3. (Skip this step if using the "option install-code ..." CLI command method described in Step 2.) Once you have the EUI64 and device-specific link key, access the trust center for this HAN and register the key into the key table using the "option link ..." CLI command. This command takes a table index, an EUI64 specified in MSB order (or LSB if using EmberZNet PRO versions older than 4.3), and a key (in MSB order, just as it is printed by "keys print") for that device. For example, for EUI64 0x000D6F0011223344 and key 0x00112233445566778899AABBCCDDEEFF (shown with EUI in MSB as expected in newer stack versions):

```
option link 0 {44 33 22 11 00 6F 0D 00} {00 11 22 33 44 55 66 77 88 99 AA BB CC DD EE FF}
```

4. 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). Devices running EmberZNet PRO 4.3.0 and later can perform this AES MMO hash operation using the `emberAesHashSimple()` method provided on SoC platforms or

the `ezspAesMmoHash()` API provided on EZSP host platforms, as demonstrated by the `optionInstallCodeCommand()` routine found in app/framework/cli/option-cli.c in EmberZNet PRO 4.7.0 and later.

5. Once this key has been determined for the incoming HAN device, install a link key table entry in the HAN's trust center by using the `emberSetLinkKeyTableEntry()` or `emberAddOrUpdateKeyTableEntry()` function with the device-specific key and EUI-64.

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 **Application Setup (Run Time) Prior to Joining a New Device to the HAN** should have been performed.

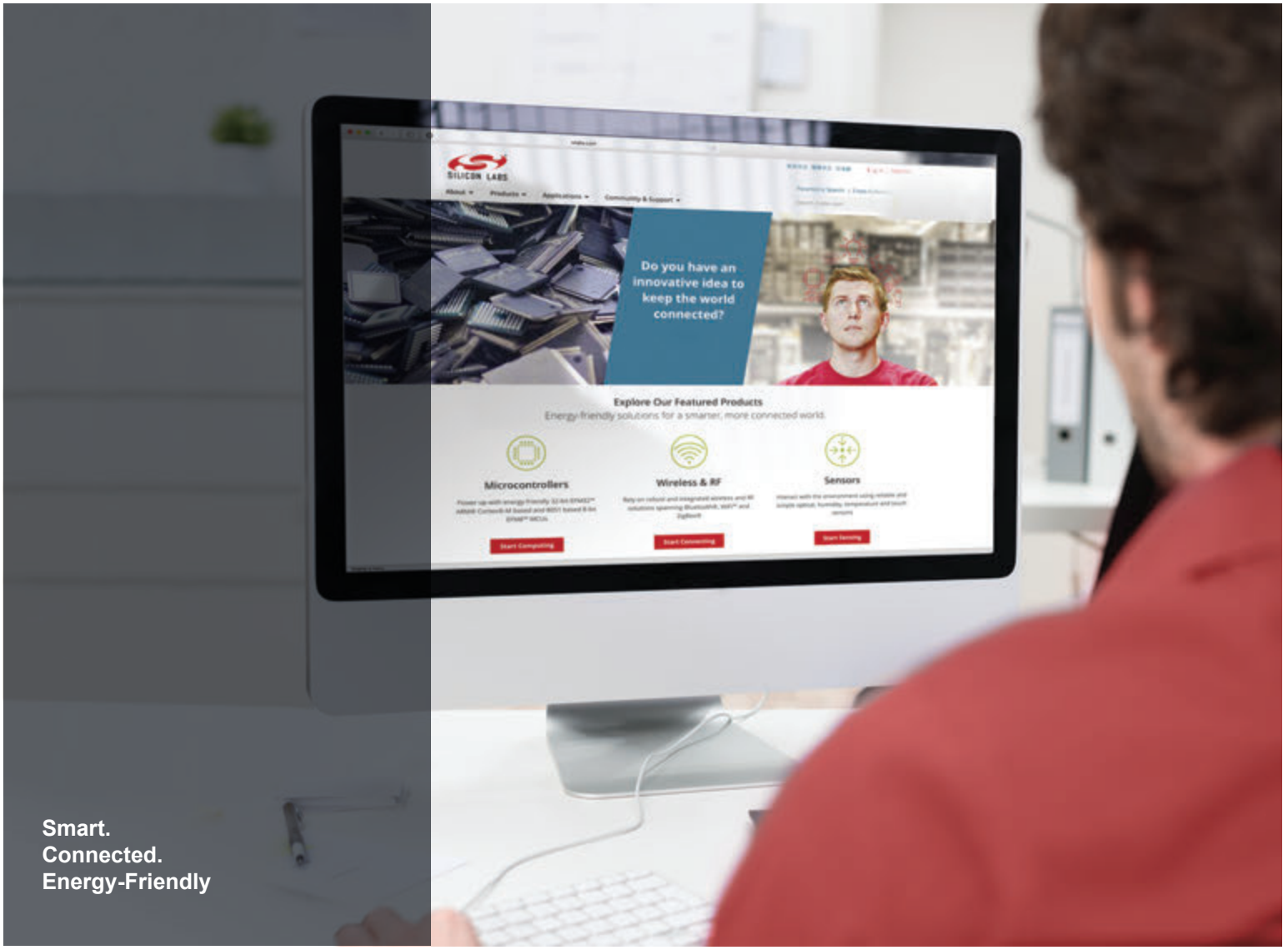
1. Join the new HAN device to the ESI's HAN (using the Button Form/Join Plugin or "network join ..." CLI command or `emberAfStartSearchForJoinableNetwork` API function, for example).
2. The ESI (as Trust Center) will trigger its `emberAfTrustCenterJoinCallback()` to indicate the outcome of the initial joining process. If successful, the joining device should begin CBKE (certificate-based key establishment).
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.

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 **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.



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