AN728: Over-the-Air Bootload Server and Client Setup

This application note describes the process users should follow to perform a Zigbee® OTA (over-the-air) bootloading session between a ZCL OTA Upgrade cluster client device and server device. The hardware used is an Ember® EM35x development kit, but a similar process may be followed for the Mighty Gecko (EFR32MG)-based development kits. (See the WSTK6001 User Manual for details about how to configure the WSTK development board for use with the EFR32MG.) Users can also refer to this procedure when setting up or testing OTA bootload downloads in their own development environments with their own hardware.

KEY POINTS

- Configure and build two zigbee OTA client images.
- Configure and build the OTA server.
- Load the client software.
- Load the NCP server software.
- Load the second OTA client image on the server.
- Run the client / server setup.

1. Introduction

This application note describes the steps necessary to demonstrate a Zigbee OTA bootloading session between a ZCL OTA Upgrade cluster client device and server device. The procedures use EM35x development kits, but are applicable to other configurations, such as EFR32MG-based Mighty Gecko development kits. (See the WSTK6001 User Manual for details about how to configure the WSTK development board for use with the EFR32MG.)

Please refer to the document *UG103.6: Application Development Fundamentals: Bootloading* for more details about this bootloader. For details about the ZCL OTA Upgrade cluster, consult the latest published revision of the zigbee cluster library specification.

Note: In the Silicon Labs AppBuilder ZCL tab, the ZCL OTA Upgrade cluster is referred to as OTA Bootloading. For consistency that terminology is used in this document as well. If you are searching zigbee documents, be sure to search for 'OTA Upgrade.'

Refer to this application note when setting up and or testing the zigbee OTA bootload cluster on your own. The Advanced section describes how this example can be expanded to use your own hardware configuration, and how to change the software configuration to support your own manufacturer-specific information.

For further reference please see:

- ZigBee Document #07-5123, *ZigBee Cluster Library Specification* - "Over-the-air Upgrading" chapter; available from http://www.zigbee.org.
- Silicon Labs document *UG102: Ember Application Framework Developer Guide* – Chapter 15
- Silicon Labs document *AN714: Smart Energy ECC-Enabled Device Setup Process*
- Silicon Labs document *AN772: Using the Application Bootloader*
- Silicon Labs document *UG266: Silicon Labs Gecko Bootloader User Guide*

2. Hardware and Software Requirements

Many of the steps in this document are performed using Application Builder (AppBuilder), a component of the Simplicity Studio IDE. For more information on configuring an application, refer to *QSG106: Getting Started with EmberZNet PRO*, provided with your release.

2.1 Client Hardware

The client hardware used in these procedures is an Ember EM35x development kit running in system-on-chip mode (SOC). The procedures are also applicable to the Mighty Gecko family (EFR32MGx). You have two choices for the download space. The first is to use an external storage device, such as a serial dataflash or serial EEPROM, connected to the device. The second option, only available with devices with more than 512 kB flash space, is to use a portion of the main flash as the download space. Bootloaders using the second option are often referred to as either interal or local storage bootloaders. For a full list of supported external memory parts for Silicon Labs EM35x and early EFR32MG devices, as well as a list of available drivers for these supported parts, see *AN772: Using the Ember Application Bootloader*. For EFR32MGx parts, see *UG266: Silicon Labs Gecko Bootloader User Guide*.

The Ember EM35x development kit breakout board rev **B3** ships with an Atmel AT45db021d part with 264 kB of memory attached to the breakout board, which can be used in these procedures. The newer Ember EM35x development kit rev **C3** has a WinBond part W25Q80BVSNIG with 256 kB of memory and can also be used with these procedures.

For Mighty Gecko-based devices using the EFR32MG1, only the serial dataflash option is available; no local storage application bootloader is presently offered. However, Mighty Gecko ICs whose part numbers begin with EFR32MG1x6 or EFR32MG1x7 contain an integrated serial flash that can be utilized just like off-chip serial dataflash but without any additional components.

EM35x-based or EFR32MG12-based devices with at least 512 kB of internal flash also support internal storage for use with a local storage application bootloader design, as well as external storage for use with an SPI flash-based application bootloader design..

The zigbee OTA client cluster can be used with an EmberZNet PRO-based device running in either SOC or EZSP network coprocessor (NCP) mode. This document does not show how to configure an EZSP host application for an NCP-based client, but the server hardware and setup described in section 2.2 Server Hardware is the same when the client is using EZSP. This application note shows one example of how the OTA client cluster can be used with a specific hardware configuration.

The client software is based upon the ZCL Application Framework V2 contained within the latest version of the EmberZNet PRO stack.

2.2 Server Hardware

The server hardware used in these procedures is an EM35x- or EFR32 Mighty Gecko-based development kit run in network coprocessor (NCP) mode with a UART-connected host application communicating via EmberZNet Serial Protocol (EZSP). In this application note, either a Linux system or Windows PC with Cygwin is required to run the host software that connects to the NCP. The storage for the OTA software images is the host's local file system.

It is possible for an OTA server to run on an EM35x- or EFR32 Mighty Gecko-based SoC, or to use an NCP with a different (non-POSIX-based) host system. A different configuration than the one presented here requires an alternative mechanism for pushing images into the OTA server so they can be then served up to clients through the zigbee OTA bootload cluster protocol. For example, software images could be pushed through a utility backhaul, an Ethernet connection to a local network, or some other proprietary mechanism.

A Linux system connected to a development board acting as a UART NCP is only one possible option to serve up OTA files. It was the friendliest option for an OTA server because there are many mechanisms to push OTA files into the server's file system.

The following figure shows a diagram of the hardware configuration used for the zigbee OTA application bootload procedures. Note that only one client is required.
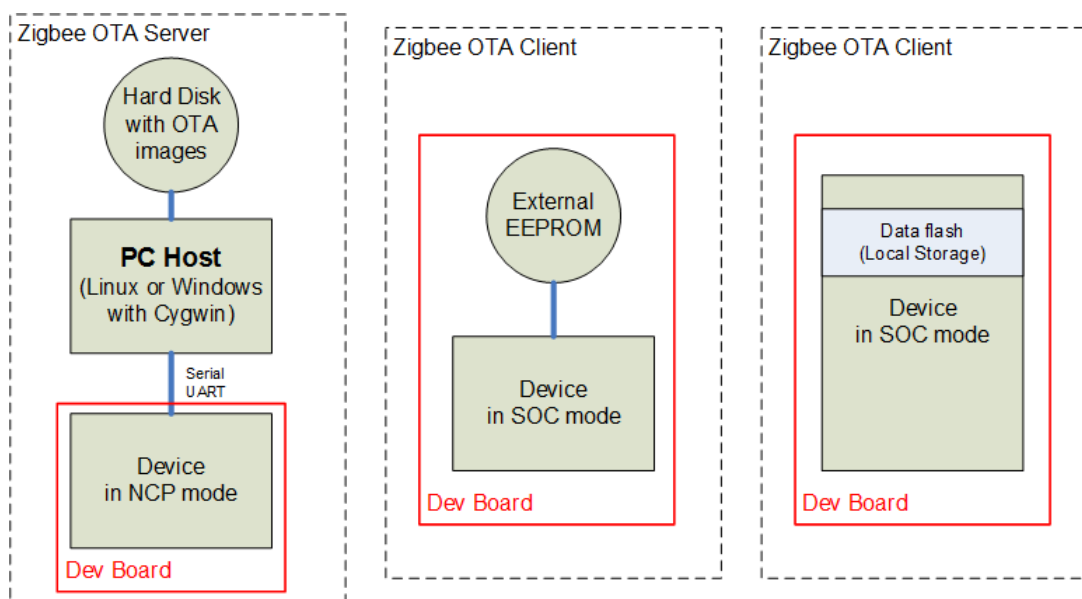

Figure 2.1. Zigbee OTA Application Bootload Hardware Diagram

3. About the Zigbee OTA Image File Format

The file format is defined in zigbee document 07-5123, *Zigbee Cluster Specification*, in the "OTA File Format" section of the "Over-the-air Upgrade" chapter. Images are composed of a zigbee OTA header, followed by one or more blobs of proprietary bootloader data, and then an *optional* set of cryptographic signature data appended to the end. The OTA server only needs to read the OTA header to serve up the file, and thus can serve up files for different manufacturers and or different products. The following figure shows an example of the layout of the file format.
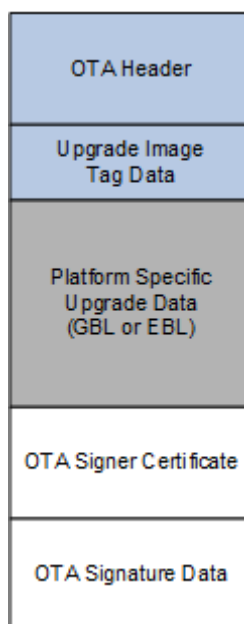

Figure 3.1. File Format Sent Over-the-Air

Silicon Labs has created a tool for generating OTA images called **Image-builder**. A detailed guide on how to use this tool can be found in document *AN716: Instructions for Using Image Builder*.

The non-ECC version of **Image-builder** is included with the EmberZNet PRO release in the **tool\image-builder** sub-directory. This example assumes that the image is not signed. For information on how to use signed images, see 5.6 Using Signed OTA Image Files in the Advanced section at the end of this document.

When generating an OTA image for the OTA server we must understand the image identity as defined by zigbee. Three elements uniquely define a zigbee OTA image file for a particular product:

- Zigbee Manufacturer ID
- Image Type ID
- Version Number

Manufacturer ID

The manufacturer ID is defined by the Zigbee Alliance. All Zigbee Alliance member companies can request a manufacturer ID. These procedures use the Silicon Labs Manufacturer ID (0x1002). However it is possible to use a different manufacturer ID. See 5.1 Specifying Your Own Manufacturer ID in the Advanced section at the end of this document.

Image Type ID

The image type ID is a manufacturer-specific number. It is used to uniquely identify different products by the same manufacturer so that the correct firmware image is downloaded for each device. Each manufacturer can define their own values and they are independent of other manufacturers. These procedures use the default image type ID of 0x0000. However, it is possible to use a different value. See 5.2 Specifying Your Own Image Type ID in the Advanced section at the end of this document.

Version Number

The version number is a 32-bit value and can be changed based on whatever the manufacturer's own versioning scheme is. A manufacturer can also use different version numbers for different products as long as they are differentiated by their Image Type IDs. Two version numbers are being used in these procedures, the old image's version number (0x00000001) and the new image's version number (0x00000022).

The only consideration for choosing a version number is how OTA servers treat version numbers. For two software images with the same manufacturer ID and image type ID, an OTA server will consider an OTA file as an upgrade from a previous version if version B is greater than version A. Therefore it is highly recommended that newer versions are higher in integer value than older versions, as is the case in these procedures.

Signing

Signing in this instance refers to signing the OTA image file, and is independent of any signature or encryption that may be present on the update image file inside the OTA wrapper file. This application note assumes that OTA images do not need to be signed. For details on how to set up the client and server for OTA image signing, see section 5.6 Using Signed OTA Image Files.

Tags

Tags are elements within the zigbee OTA file format for arbitrary blobs of data. They are generally defined by the manufacturer and can contain whatever data is needed for an upgrade. However, the specification defines a few standard tags. The tag of 0x0000 indicates an Upgrade image. See *AN716: Instructions for Using Image Builder* for more details.

By default, the Silicon Labs implementation of the **OTA Cluster Platform Bootloader** plugin looks for a tag value of 0x0000 within the zigbee OTA image that contains an EBL or GBL file. It will attempt to bootload this file.

Also, the **OTA Bootload Cluster Client Policy** plugin validates the EBL or GBL file within tag 0x0000 if the checkbox **Perform EBL verification** has been checked in AppBuilder.

4. General Procedure

The following outlines the steps to configure the OTA bootload cluster client and server. Details are presented in subsequent sections.

1. Configure and build the first OTA client image.
2. Build the second OTA client image.
3. Configure and build the OTA server.
4. Generate a zigbee OTA image file for the OTA server.
5. Load the client software.
6. Load the NCP server software.
7. Load the second OTA client image on the server.
8. Run the client / server setup.

4.1 Configure the First Zigbee OTA Client

This procedure configures a zigbee Smart Energy 1.x In-Premise Display (IPD) as an example, but it can be adapted for any sample application. Other zigbee device types can be configured similarly. The example setup includes support for the zigbee OTA bootload cluster client. The client software does not require that OTA images be signed and verified in order to successfully bootload them. See the section 5.6 Using Signed OTA Image Files for details on how to use signed images.

The procedures in this application note build two client images, the old version and the new version. This client will be the old version. The image generated will run before the zigbee OTA process upgrades the client device.

Simplicity Commander is used extentensively during these procedures. Simplicity Commander is installed during stack installation in Simplicity Studio.

Here are the basic steps to configure the OTA client:

1. In Application Builder, create and name a new project. This can be blank or based on one of the provided examples.
2. In the **ZCL Cluster Configuration** tab:
   a. From the **ZCL Device Type** drop-down list, select **SE Devices**, then select **SE In-Premises Display**.
   b. In the **Cluster List** pane on the bottom left, expand the **General** group.
   c. Check the **Over-the-Air Bootloading Cluster** client checkbox.
3. In the **Printing and CLI** tab:
   a. In the **Debug Printing** section, check the "Enable Debug Printing" checkbox.
   b. Expand the **Cluster Debugging** group.
   c. Check the **Compiled-in** and **Enabled at startup** checkboxes for the **Over-the-Air Bootloading** cluster.
4. In the **HAL Configuration** Tab, click the **Bootloader** drop-down menu and specify **Application or Local Storage**. Local storage is not an option for devices with less than 512 kB of flash.
5. In the **Plugins** Tab, make sure the following plugins are checked (other plugins may be checked by default as well):
   - OTA Bootload Cluster Client
   - OTA Bootload Cluster Client Policy
      a. Check the **Perform EBL Verification (SOC Only)** checkbox.
      b. In the **Firmware Version** text entry box, set the value to 1.
   - OTA Bootload Cluster Common Code
   - OTA Bootload Cluster Storage Common Code
   - OTA Simple Storage Module
   - OTA Simple Storage EEPROM Driver
      a. Check the **SOC Bootloading Support** checkbox.
   - EEPROM
   - OTA Cluster Platform Bootloader
6. Save your configuration (ISC) file (has the extension .isc) with a name that identifies this as the old version.
7. Click [**Generate**].
8. Build the application according to your preferred method. Make sure that the EBL or GBL file is generated.

   Note: An EEPROM/dataflash driver is not specifically included in the application's project file. The application will call into the bootloader's storage driver code to perform the low-level calls to the serial dataflash. This is why it is important to enable a bootloader in the client configuration and later to load a bootloader into the device.

4.2 Build the First Client Image

If you are working in Simplicity Studio, you can build the application in that environment.

1. Launch IAR Embedded Workbench (EWARM). The version of IAR EWARM used should correspond to the stack's required compiler version as documented in the release notes. These instructions apply to IAR-EWARM version 7.80.2.

2. Open the project file that was generated by Simplicity Studio.

3. Select the **Tools** menu and the **Rebuild All** option.

The project should successful compile all targets.

For a zigbee OTA Bootload Cluster Client, beginning with EmberZNet 5.0, AppBuilder will automatically create a Windows .BAT file that that executes image-builder and wraps the EBL or GBL file in the zigbee OTA file format. The parameters passed to image-builder are based upon those specified in the application configuration, specifically the values declared in the "OTA Bootload Cluster Client Policy" plugin.

By default IAR Workbench will not show the post build processing output. To see the output of the post processing you must change the Build Output setting in IAR Workbench. Go to the **Tools** Menu and select **Options**, then select **Messages** from the list at the left. The first option on the right side should be **Show Build Messages**. Change the drop-down menu to **All**. You should see output that looks like this:

```
Performing Post-Build Action
C:\Users\rbalexan\Ember\EmberZNet5.7.0-GA\EM35x\app\builder\OtaClient357/OtaClient357.bat C:\Users\rbalexan\Emb
er\EmberZNet5.7.0-GA\EM35x\app\builder\OtaClient357 C:\Users\rbalexan\
Ember\EmberZNet5.7.0-GA\EM35x\build\OtaClient357\OtaClient357
" "
"This converts S37 to Ember Bootload File format if a bootloader has been selected in AppBuilder"
" "
C:\Windows\system32>cmd /c ""C:\Program Files (x86)\Ember\ISA3 Utilities\bin\em3xx_convert.exe"  "C:\Users\rbal
exan\Ember\EmberZNet5.7.0-GA\EM35x\build\OtaClient357\OtaClient357.s37" "C:\
Users\rbalexan\Ember\EmberZNet5.7.0-GA\EM35x\build\OtaClient357\OtaClient357.ebl" > "C:\Users\rbalexan\Ember\Em
berZNet5.7.0-GA\EM35x\build\OtaClient357\
OtaClient357-em3xx-convert-output.txt"
em3xx_convert (Version 3.1b15.1361555916)
Parse .s37 format for flash
Parse .s37 format for flash
Targeting UNKNOWN
Approximate Usage Information:
RAM Usage:
  APPLICATION_CONFIGURATION_HEADER usage: 0x20000000-0x20000d4b (3404 bytes)
  Available for future use:         0x20000d4c-0x20001957 (3084 bytes)
  Call Stack:                       0x20001958-0x200020af (1880 bytes)
  Globals and Statics:              0x200020b0-0x20002fee (3903 bytes)
  Alignment Overhead:               0x20002fef-0x20002fef (1 bytes)
  NO_INIT and Debug Channel:        0x20002ff0-0x20002fff (16 bytes)
Flash Usage:
  Reserved for Bootloader:          0x08000000-0x08001fff (8192 bytes)
  CODE and Tables:                  0x08002000-0x080204cf (124112 bytes)
  CONST and INITC:                  0x080204d0-0x08022403 (7988 bytes)
  Available for future use:         0x08022404-0x0802dfff (48124 bytes)
  Reserved for SIMEE:               0x0802e000-0x0802ffff (8192 bytes)
Usage Summary:
  12288 total bytes RAM, 9204 used, 3084 available
  196608 total bytes Flash, 148484 used, 48124 available
Create ebl image file
DONE
" "
"This creates a ZigBee OTA file if the "OTA Client Policy Plugin" has been enabled.
"It uses the parameters defined there. "
" "
C:\Windows\system32>cmd /c "C:\Users\rbalexan\Ember\EmberZNet5.7.0-GA\EM35x\app\builder\OtaClient357\..\..\..\t
ool\image-builder\image-builder-windows.exe --create C:\Users\rbalexan\Ember\
EmberZNet5.7.0-GA\EM35x\app\builder\OtaClient357\OtaClient357.ota  --version 1 --manuf-id 0x0 --image-type 0 --
tag-id 0x0000 --tag-file C:\Users\rbalexan\Ember\EmberZNet5.7.0-GA\EM35x\build\
OtaClient357\OtaClient357.ebl --string "EBL OtaClient357" > C:\Users\rbalexan\Ember\EmberZNet5.7.0-GA\EM35x\bui
ld\OtaClient357\OtaClient357-image-builder-output.txt"
image-builder (C) 2013 by Silicon Labs
Version: 1.4.1
ECC signature support NOT present.
EBL file integrity 'OtaClient357.ebl': OK
File: C:\Users\rbalexan\Ember\EmberZNet5.7.0-GA\EM35x\app\builder\OtaClient357\OtaClient357.ota
Magic Number:         0x0BEEF11E
```

```
Header Version:       0x0100
Header Length:        56 bytes
Field Control:        0x0000
Manufacturer ID:      0x0000
Image Type:           0x0000
Firmware Version:     0x00000001
Stack Version:        0x0002
Header String:        EBL OtaClient357
Total Image Size:     132734 bytes
Total Tags:m          1
  ID:                 0x0000   (Upgrade Image)
  Length:             132672 bytes
```

4.3 Build a Second ZigBee OTA Client Image

This step builds the second, new version image. The images are identical except for the version number embedded in them.

4.3.1 Copy the First OTA Client Image

Copy this file to another location outside of the build directory so that it is not destroyed in the next step. The build directory by default is located under the protocol installation under Simplicity Studio, for example:

`\SiliconLabs\SimplicityStudio\v4\developer\sdks\gecko_sdk_suite\v1.0\protocol\zigbee_5.9\build`

Name it appropriately based on the version number, for example `OtaClient-v1.ebl`.

4.3.2 Generate a New OTA Client Configuration

To generate a new OTA Client configuration, perform the following steps.

1. Open the previously saved OTA Client ISC file.
    a. Open the **File** menu and select **Open**.
    b. In the dialog box, navigate to the location of the saved ISC file and select **Open**.
2. Select the **Plugins** tab.
    a. Select the **OTA Bootload Cluster Client Policy** plugin.
    b. Change the value in the **Firmware version number** entry box to `0x22`. This value is an arbitrary choice but it must be greater in value than the previous version (0x01).
3. Click [**Generate**].
4. Build the project using your desired method. Make sure the EBL or GBL file is generated.
5. Save the AppBuilder configuration so you can recall it for later reference.

4.3.3 Build the Second OTA Client Image

Follow the steps in the section 4.2 Build the First Client Image to build the second image. It may be helpful to take the existing configuration you saved in the previous section and perform a Save As operation to make a copy of that configuration that you can then change to reflect the new version in **OTA Bootload Cluster Client Policy** plugin settings.

4.4 Configure the Zigbee OTA Server

This procedure creates a zigbee Smart Energy 1.x Energy Services Interface (ESI) device. The device includes support for the zigbee OTA bootload cluster server. The host server software can support both signed and unsigned OTA images since it will not do any verification on these files.

Here are the basic steps to configure the OTA server:

1. In Application Builder, create and name a new project. This can be blank or based on one of the provided examples.
2. In the ZCL Cluster Configuration tab:
    a. From the **ZCL Device Type** drop-down list, select **SE Devices**, then select **SE Energy Service Interface**.
    b. In the **Cluster List** pane on the bottom left, expand the **General** group.
    c. Check the **Over-the-Air Bootloading Cluster** server checkbox.
3. In the **Printing and CLI** tab:
    a. In the **Debug Printing** section, ensure "Enable debug printing" is checked.
    b. Expand the **Cluster Debugging** group.
    c. Check the **Compiled-in** and then the **Enabled at startup** checkboxes for the over-the-air bootloading cluster.
4. In the **HAL Configuration** tab,
    a. Make sure the appropriate chip is selected.
    b. Under the **Platform Configuration** section, open the **Host** drop-down menu and select **UNIX Host**.
5. Select the **Plugins** tab. Make sure the following plugins are checked (other plugins may be checked as well):
    • Concentrator Support
    • EZSP Common
    • EZSP UART
    • File Descriptor Dispatch
    • Gateway Support
    • OTA Bootload Cluster Common Code
    • OTA Bootload Cluster Server
    • OTA Bootload Cluster Server Policy
    • OTA POSIX Filesystem Storage Module
6. Save your configuration (ISC) file (has the extension .isc).
7. Click **Generate**.

4.5 Build the Server

Building the OTA Server can be done either on a Linux box using GCC, or a Windows box running Cygwin and GCC. Both processes are described below.

4.5.1 Building on a Linux Box

Building the OTA server can be done on a Linux system. This application note assumes that you are NOT using a cross-compiler and that the target system is the same system as the one where the development tools exist. It is possible to cross-compile for a different Linux system (for example, on an x86 PC targeting an embedded ARM Linux system). See section 5.4 Using a Cross-compiler for the OTA Server for details on how to cross-compile for other systems.

Building the OTA Server for Linux requires a number of development tools. We assume you have access to these tools. They are as follows:

- Make
- GCC
- sed
- awk
- xargs
- The standard C Library and its development headers
- The Readline Library and its development headers (optional)
- The Ncurses Library and its development headers (optional - only needed to support Readline)

The steps to build the OTA server under Linux follow. It is assumed that these steps are run after configuring the server in Simplicity Studio.

1. Copy the EmberZNet PRO stack files from their installed location on Windows to a directory on the Linux PC. This should include the files that were generated in a previous step.
2. Launch a Bash Shell.
3. Change to the root of the EmberZNet directory on your Linux PC.
4. Run **Make** on the generated Makefile from that directory, for example `make –f OtaServer.mak` or simply `make` if the generated Makefile is in the current working directory and is specifically named "Makefile".

   If Readline support is not desired, then add the NO_READLINE argument, for example `make –f OtaServer.mak NO_READLINE`

The compilation should complete successfully.

4.5.2 Building on a Windows Box Under Cygwin

Building the OTA Server under Windows requires Cygwin. Cygwin is a set of native Windows tools that emulate a Linux environment. Cygwin can be downloaded from the Internet from http://www.cygwin.com.

Cygwin is actually a collection of hundreds of small programs that make up a Linux distribution. Cygwin can be configured in a myriad of different ways for a particular Windows PC and so by default may not include the required tools for building an EmberZNet PRO host application. Therefore proper installation of the required tools is necessary before actually building the EmberZNet PRO code.

The following is a general list of the packages necessary to build an EmberZNet PRO host application under Cygwin.

- Bash shell
- Make
- GCC4
- sed
- awk
- xargs
- The standard C Library and its development headers
- libreadline-devel - The Readline Library and its development headers (optional)
- libncurses-devel - The Ncurses Library and its development headers (optional - only needed to support Readline)

   Note: Although it is possible to use the Cygwin tools to build an application that does NOT require the Cygwin DLL, it is not possible to do this with an EmberZNet PRO host application. The host application requires access to the Cygwin DLL to correctly run certain parts of the Ember application framework code.

The steps to build the OTA server under Windows using Cygwin are as follows.

1. Launch a Cygwin Shell.
2. Change to the root of the EmberZNet directory.
3. Run **Make** on the generated Makefile from that directory, for example `make –f OtaServer.mak` or simply `make` if the generated Makefile is in the current working directory and is specifically named "Makefile".

   If Readline support is not desired, then add the NO_READLINE argument, for example `make –f OtaServer.mak NO_READLINE`

The compilation should complete successfully.

4.6 Load the Client Software

The following instructions apply to the EM35x in Simplicity Studio. For Mighty Gecko-based platforms, see *QSG106: Getting Started with EmberZNet PRO* for instructions on loading application and bootloader firmware to the target device using Simplicity Studio, or see *UG162: Simplicity Commander Reference Guide* for instructions on how to perform the firmware loading operation outside of Simplicity Studio via the Simplicity Commander command line tool. Simplicity Commander can only be used with the Mighty Gecko platform.

Use the Network Analyzer tool, available through Simplicity Studio, to load the previously built application and a bootloader.

1. If the Devices view is not already shown, open the **Window** menu, select the **Show View** sub-menu, and select the **Devices** menu item.
2. Select the appropriate device from the window, right-click and select **Connect**.
3. Right-click on the device again, and select **Upload Application**.
4. In the resulting dialog box:
    a. Click on the folder icon to the right of the **Browse** text entry box, which has the tooltip **Browse for a Custom Image**.
    b. Navigate to the folder containing the previously-built EBL file, the image that was built with a version number of 1.
    c. Select the EBL file.
    d. Click [**OK**].
5. Check the box labeled **Bootloader**.
6. Click on the folder icon to the right of the **Bootloader** drop-down menu.
7. Navigate to the EmberZNet PRO release folder containing the appropriate bootloader and select the S37 file for the EM35x Development Kit hardware.

    The zigbee OTA client uses an Application Bootloader. A sample path to the bootloader is: `C:\SiliconLabs\SimplicityStudio\v4\developer\sdks\gecko_sdk_suite\v1.0\protocol\zigbee_5.9\tool\bootloader-em357\app-bootloader\app-bootloader-spi-at45db021d.s37`.

    If using a Local Storage Bootloader, a sample path to the bootloader is. `C:\SiliconLabs\SimplicityStudio\v4\developer\sdks\gecko_sdk_suite\v1.0\protocol\zigbee_5.9\tool\bootloader-em357\app-bootloader\app-bootloader-local-storage.s37`

8. Click [**OK**].
9. Wait for the device to successfully finish a flash load of the EM35x device.

4.7 Load the NCP Server Software

In order to use an EM35x or Mighty Gecko device connected to a PC host through USB, it is necessary to load the EZSP NCP software onto the target chip. The following describes the process using an EM357 as an example for the NCP target chip, and precompiled images. You can also build your own images as described in **AN1010: Building a Customized NCP Application**.

Note: The procedure for an EFR32MG12 is similar but does not need steps 1 and 2.

1. Select the appropriate device from the Devices view, right-click and select **Connect**.
2. Identify the EM35x as a coprocessor by doing the following.
    a. Right-click on the .device entry for the EM35x device in the Devices view and select **Device Configuration**.
    b. Open the **Radio Processor** Drop-down menu.
        i. Select **EM357-EZSP** from the menu.
        ii. Check the **Protect from Auto-discovery Changes checkbox**
        iii. Click [**OK**].
3. Right-click on the device again, and select **Upload Network Coprocessor Firmware**.
4. In the resulting dialog box:
    a. Click on the folder icon to the right of the Browse text entry box, which has the tooltip **Browse for a Custom Image**.
    b. Navigate to the location of your EmberZNet PRO release, browse to the "build\em35x-ezsp-images" subdirectory, and select the pre-built NCP image, for example: `C:/SiliconLabs/SimplicityStudio/v4/developer/sdks/gecko_sdk_suite/v1.0/protocol/zigbee_5.9/ncp-images/{chipName}/{ncp-filename}`

    Note: Upload the NCP application and the bootloader separately. In the Upload dialog's "Application" field browse to the EBL or GBL file for your desired NCP firmware. Check the box next to the "Bootloader Image" field and then browse to the bootloader binary (an .s37 file typically found in a tool\bootloader-{chipName}\{bootloaderType} subdirectory).

    c. Click [**OK**].
5. Wait for the device to successfully finish a flash load of the NCP.

4.8 Load the Second OTA Client Image on the Server

The OTA Server application looks for OTA files in a directory called **ota-files** that is a sub-directory in the current working directory from which the server application is launched.

1. Launch a Cygwin or Linux shell on the OTA Server PC.
2. Navigate to the directory where the OTA Server application will be run, , make a subdirectory, and copy the ota files to it.

   For example, in Linux:

   ```
   cd /home/rob/
   mkdir ota-files
   cp –i OtaClient-v22.ota ota-files/
   ```

   For example, on Cygwin:

   ```
   cd /cygdrive/c/SiliconLabs/SimplicityStudio/v4/developer/sdks/gecko_sdk_suite/v1.0/protocol/zigbee_5.9/
   mkdir ota-files
   cp –i OtaClient-v22.ota ota-files/
   ```

4.9 Running the Zigbee OTA Client / Server

1. Run the OTA Server.
2. Make sure the development board for the OTA server is connected to the PC with a USB or serial cable.
3. Launch the OTA Server by executing the OTA server application from the command-line, and pass it the location of the communications port. The communications port will either be a USB or serial port device. The method for determining which port the EM35x is connected to depends on the operating system of the host PC.

Under Linux

The USB port is registered as /dev/ttyUSBX, where X is a number as assigned by the operating system. For example the following would use `ttyUSB0`.

  a. Launch a Shell.
  b. Change to the directory where the OTA Server application is located. This should be the same directory where a sub-directory called "ota-files" exists, for example: `cd /home/rob`.
  c. Enter `./OtaServer –p /dev/ttyUSB0`

Under Windows

The USB port is either COM1, COM2, COM3, and so on. The following example uses COM1.

  a. Launch a Cygwin Shell.
  b. Navigate to the directory where the EmberZNet PRO installation is located, for example:

```
cd /cygdrive/c/SiliconLabs/SimplicityStudio/v4/developer/sdks/gecko_sdk_suite/v1.0/protocol/zigbee_5.9/
```

  c. Type the command: `./build/OtaServer/OtaServer –p COM1`

4. Connect to the OTA Client by launching a Windows command prompt or Linux shell and connecting by Telnet to the serial port of the Debug Adapter (ISA3). For example: `telnet rob-350 4901`.
5. Make sure both the client and server have Over-the-Air Bootloading Cluster printing enabled.
  a. Execute the following command on the OTA Client: `plugin ota-storage printImages`
  b. Execute the following command on the OTA Server: `plugin ota-storage printImages`

In both cases output should print to the CLI. If no output is visible, go back and verify the configuration of the application and its Debug Printing configuration.

6. Make sure that the client is running the older version of the client software. From the OTA client's CLI type the following command. You should see the corresponding output:

```
OtaClient>plugin ota-client info
Client image query info
Manuf ID:        0x1002
Image Type ID:    0x0000
Current Version:  0x00000001
Hardware Version: NA
Query Delay ms:            300000
Server Discovery Delay ms: 600000
Download Delay ms:        0
Run Upgrade Delay ms:     600000
Verify Delay ms:          10
Download Error Threshold: 10
Upgrade Wait Threshold:   10
```

Note that the current version field says 0x00000001.

7. Verify that the OTA Server has a newer image loaded in it. From the OTA server's CLI type the following command. You should see the corresponding output:

```
OtaServer>plugin ota-storage printImages
Image 0
  Header Version: 0x0100
  Header Length:  56 bytes
  Field Control:  0x0000
  Manuf ID:       0x1002
  Image Type:     0x0000
  Version:        0x00000022
  Zigbee Version: 0x0002
  Header String:  Our Test Image
```

```
    Image Size:      146954 bytes
    Total Tags: 1
      Tag: 0x0000
        Length: 146880
1 images in OTA storage.
```

Note that the version number is 0x00000022. The exact size of the image may vary.

8. Erase any previous version file that has been saved in the client. This is not normally necessary in a production system, but in this example a previous version may cause problems. Enter the following command on the client:

```
plugin ota-storage delete 0.
```

9. Turn off receive message printing to limit console output during the download. Skipping this step will not negatively impact the functionality.

   a. On the server: `option print-rx-msgs disable`

   b. On the client: `option print-rx-msgs disable`

10. Join the devices into the same network.

    a. On the server: `network form 24 -1 0x1aaa`

    b. On the server: `network pjoin 254`

    c. On the client: `network join 24 -1 0x1aaa`

    If the client joins successful, then you should see EMBER_NETWORK_UP in the output.

    d. Wait until Smart Energy Registration has completed.

11. Start the OTA Client's state machine on the client:

```
OtaClient> plugin ota-client start
starting OTA client state machine
Bootload state: Discovering OTA Server
Svc Disc: Starting discovery for cluster 0x0019
Svc Disc: Match found from 0x0000.
Setting OTA Server to 0x0000
Svc Disc: complete.
Bootload state: Get OTA Server EUI
OTA Cluster: setting IEEE address of OTA cluster
Last offset downloaded: 0x00000000
No image found in storage.
Query next image response: New image is available for download.
Starting download, Version 0x00000022
Bootload state: Downloading Image
Download: 0% complete
Download: 5% complete
...
```

Note: The download may take up to 10 minutes.

12. If all goes well you should see the following output:

```
Download: 100% complete
Bootload state: Verifying Image
Last offset downloaded: 0x00023E6C
Starting EBL verification
EBL passed verification.
Custom verification passed: 0x00
Bootload state: Waiting for Upgrade message
Sending Upgrade End request.
OTA Cluster: wait for 0 s
RXed timeOut 0x00000000 s, MAX timeOut 0x00000DBB s
Adding 2000 ms. delay for immediate upgrade.
Countdown to upgrade: 2000 ms
Bootload state: Countdown to Upgrade
Applying upgrade
Executing bootload callback.
Reset info: 0x02 (BTL)
```

4.10 Repeating the Procedure

Once a zigbee OTA client is upgraded through the zigbee over-the-air bootload cluster, it can only be upgraded again if the server has a different version of software.

In order to perform a zigbee OTA firmware download again it is necessary to do one of the following:

1. Downgrade the running image of the client to the old version

   or
2. Create another client image with a newer software version.

Downgrade the Running Image to the Old Version

Re-run the steps in the section 4.6 Load the Client Software. Load version 0x00000001 of the software, the old image.

Create another Image with a Newer Software Version

Run the following steps:

1. Re-run the steps in section 4.1 Configure the First Zigbee OTA Client but specify a different Firmware Version that is greater than 0x00000022.

   Make sure that you click **Generate** to generate the build files.
2. Re-run the steps in sections 4.2 Build the First Client Image and 4.5 Build the Server.

   Make sure that the **--version** argument passed to image-builder reflects the newest version number.

5. Advanced Topics

This section presents a number of advanced topics that allow the developer to customize the bootload to their specific hardware and software.

5.1 Specifying Your Own Manufacturer ID

Follow these steps in order to specify a manufacturer ID other than 0x1002, which is the Silicon Labs-specific manufacturing ID.

Each member of the Zigbee Alliance has its own manufacturer ID. The list of manufacturer IDs is maintained in zigbee Document 05-3874 Manufacturer Code Database. If your company does not have a manufacturer ID then it must request one from the zigbee help desk (help@zigbee.org).

1. When configuring the OTA Client in AppBuilder, do the following:
   a. Select the **ZCL Cluster Configuration** Tab.
   b. In the **Manufacturer Code** text entry box on the right, enter your own manufacturer code.
   c. Click **Generate**.
   d. Re-compile the application using IAR workbench.
2. When generating the ZigBee OTA image file, do the following.
   a. Specify your own manufacturer ID by passing that value to the **--manuf-id** argument when executing the **image-builder** command.

   Note: You do NOT need to modify the manufacturer ID of the OTA server in this example. The server may have a different manufacturer ID than the OTA Client. You may choose to modify the manufacturer ID to match the OTA server device that your company is building.

5.2 Specifying Your Own Image Type ID

Follow these steps in order to specify an image type ID other than 0x0000.

1. When configuring the OTA Client in AppBuilder, do the following:
   a. Select the **Plugin** tab.
   b. Select the **OTA Bootload Cluster Client Policy** plugin.
   c. In the **Image Type ID** text entry box, enter your own image type ID code.
   d. Click **Generate**.
   e. Re-compile the application using IAR workbench.
2. AppBuilder will automatically update the Windows Batch file command being executed by IAR Workbench to generate a zigbee OTA file. The new Image Type argument will be passed to image-builder with **--image-type**.

5.3 Modifying the OTA Client to Use a Different EEPROM

Many different external storage parts are available, and the choice of which one is used is based on many different factors. *AN772: Using the Ember Application Bootloader* and *UG266: Silicon Labs Gecko Bootloader User Guide* detail several different parts supported by Silicon Labs and their various device parameters. The EM35x and EFR32MG1 parts in that list have pre-built application bootloader binaries included with the EmberZNet PRO stack release. A selective list of binaries is also provided for EFR32MG12 and newer devices.

If the external storage part does not support read-modify-write, then the AppBuilder application configuration must be updated. The following describes the process:

1. In AppBuilder, open the OTA client configuration (ISC) file.

2. Select the **Plugin**s tab.

3. Select the **OTA Simple Storage EEPROM Driver** plugin and uncheck the option labeled **EEPROM Read-modify-write device Support.**

4. Click **Generate** to re-generate the appropriate AppBuilder files.

5. Re-compile the application.

   Note: Note: A new Bootloader will be required to support the change in EEPROM. You must load a new bootloader by following the steps in section 4.6 Load the Client Software, and selecting one appropriate to the bootloader you are using. For the example presented in this section the appropriate bootloader would be:

   ```
   C:\SiliconLabs\SimplicityStudio\v4\developer\sdks\gecko_sdk_suite\v1.0\protocol\zigbee_5.9\tool\bootloader
   -em357\app-bootloader\app-bootloader-spi-spiflash.s37.
   ```

5.4 Using a Cross-compiler for the OTA Server

It is possible to use a cross-compiler for building the OTA server for another target system. For example, building on an x86 Windows PC for an ARM Linux system. This process is relatively straight forward once the cross-compiler is properly installed and configured. To do this you must run make and pass it the name/location of the compiler and linker on the command-line.

For example:

```
make CC=/opt/crosstool/gcc-4.0.1-glibc-2.3.5/arm-unknown-linux-gnu/bin/arm-unknown-linux-gnu-gcc LD=/opt/crosst
ool/gcc-4.0.1-glibc-2.3.5/arm-unknown-linux-gnu/bin/arm-unknown-linux-gnu-gcc
```

The Makefile expects a GCC style command-line compiler and it uses the compiler command (rather than the linker command directly) for linking.

5.5 Using Encrypted GBL or EBL Images

Encrypted images protect the new application image from inspection or tampering by using a symmetric encryption key. For more information about the the legacy Ember secure bootloader and encrypted EBL see *AN772: Using the Ember Application Bootloaders*. For more information about the Gecko Bootloader and encrypted GBL images, see *UG266: Silicon Labs Gecko Bootloader User Guide*.

5.5.1 Changing the Bootloader Type

The following instructions apply to EM35x devices. For EFR32MG devices, follow the instructions in *UG266: Silicon Labs Gecko Bootloader User's Guide.*

1. Load the ISC file configured with the OTA Bootload Cluster Client.

2. Switch to the **HAL** tab.

3. Open the **Bootloader** drop-down menu, and select **Secure Application** or **Secure Local Storage**.

4. A Secure Application bootloader requires an encryption key. There are two ways to acquire a key:

   • AppBuilder can randomly generate a key file

   • The User can specify a key file.

   a. To have AppBuilder randomly generate a key, press the **Generate Key** Button:

   b. To have AppBuilder use an existing key file, select the Browse Button and navigate to a file containing the key. The format of a key file is as follows:

```
# Comment lines begin with '#'
TOKEN_MFG_SECURE_BOOTLOADER_KEY: 0123456789ABCDEF0123456789ABCDEF
```

5. Click [**Generate**] in AppBuilder

6. Build the application using IAR Workbench following the steps in section 4.2 Build the First Client Image.

   Note that the encrypted EBL output will be: build/ApplicationName/ApplicationName.ebl.encrypted

7. AppBuilder will automatically create a ZigBee OTA file containing the encrypted EBL: build/ApplicationName/ApplicationName.ota

8. Load a Secure Bootloader onto the device. Follow the steps in section 4.6 Load the Client Software, but select a Secure Application bootloader. A sample path to the secure bootloader is:

```
C:\Program Files\Ember\EmberZNet5.7.0-GA\em35x\tool\bootloader-em357\secure-app-bootloader\secure-app-bootloader-spi-at
45db021d.s37.
```

5.5.2 Load an Encryption Key onto the Device

The following instructions apply to EM35x devices and reference the em3xx_load utility. For EFR32MG devices, follow the instructions in *UG266: Silicon Labs Gecko Bootloader User's Guide.* See *UG162: Simplicity Commander Reference Guide* for more information on applicable commands.

In order for the secure bootloader to function correctly, a Secure Bootloader key must be loaded in the manufacturer tokens of the device. To load a key into the device use **em3xx_load** and pass it the key file that was specified previously in AppBuilder.

```
em3xx_load.exe --ip <ip-address-of-ISA3> --cibtokenspatch <key-file>
```

Note: The standalone bootloader encryption key is a different manufacturing token then the Secure Bootloader Encryption key.

5.6 Using Signed OTA Image Files

The zigbee OTA Bootload Cluster has defined a means to cryptographically sign and verify images received over the air using asymmetric cryptography. The Smart Energy 1.x Profile requires that images are signed when using the Bootload cluster. Silicon Labs software supports generating and validating the signature.

Currently only the Smart Energy Profile has support for signatures on images, and this is coupled with the requirement of using Certicom's ECC software. For the OTA client to verify signatures this requires adding the Certicom ECC library into the application. This in turn will also require that the client device perform key establishment after joining to a Smart Energy network. In order to satisfy the client requirement, the OTA server must also be updated to support ECC and certificates so that it can perform key establishment with the client. However the OTA server does not need support for image signing or verification.

The process for adding signing requires the following additional steps:

1. Obtaining the necessary ECC components.
2. Configuring the client with ECC support and signature verification.
3. Generating a new application for the client.
4. Signing the client image.
5. Loading client and server certificates.
6. Running the software.

5.6.1 Obtaining the Necessary ECC Components

Due to US Export regulations on cryptography the ECC library is NOT available as part of the normal EmberZNet PRO release; it must be obtained separately. This also applies to **image-builder-ecc**, which is the ECC enabled version of the image-builder tool, and to ECC enabled NCP binaries used by the OTA server connected to the NCP.

All of the necessary files can be obtained by contacting the Silicon Labs Support portal and requesting access to ECC-related content. Once access is granted, this content will appear as a separate .zip file in the same location as each EmberZNet release installer.

5.6.2 Configuring the client with ECC Support and Signature Verification

1. In AppBuilder, open the OTA client configuration (ISC) file.
2. Switch to the **Stack Configuration** tab.
    a. Underneath the **Security** group, check the **Real ECC** checkbox.
    b. In the **Library Path** box, specify a path to the ECC Library for your target NCP.
3. Switch to the **Plugins** tab.
    a. Select the OTA Bootload Cluster Client plugin.
    b. Check the **Signature Verification Support** checkbox. This will required that signed images be used in the over-the-air bootload.
4. Save the OTA client configuration (ISC) file by opening the **File** menu and selecting **Save**.
5. Click [**Generate**].

5.6.3 Server Configuration

There are no changes to the AppBuilder configuration for the EZSP-host application running as the OTA server. In fact it is not possible to select the Real ECC option for an EZSP host application since ECC support is pre-compiled into the NCP binaries. The proper ECC-enabled NCP image must be loaded. The details for doing this are described in a subsequent step.

5.6.4 Building and Signing the Client image

In order to generate a signed image, the ECC version of the Image-builder tool must be used. The **image-builder-ecc** application does everything that the normal Image-builder application does, but can also sign images.

The **image-builder-ecc** application can sign images with a test private key or a manufacturer specific private key. The test private key is NOT intended for production systems and is only used as an example in this application note.

See document AN716, *Instructions for Using Image Builder*, for instructions on using a manufacturer specific private key.

These steps assume that the **image-builder-ecc** application has been obtained and copied to the EmberZNet PRO stack release directory.

Signing Automatically with a Test Certificate

Starting with EmberZNet 5.0, AppBuilder will generate a Windows BAT file that will execute image-builder as part of the post build step of IAR workbench. It will sign the image automatically if the option has been selected in the **OTA Bootload Cluster Client** plugin. To do this it requires the following additional steps:

1. Copy image-builder-ecc-windows.exe to the EmberZNet PRO directory tool\image-builder\.
2. Re-launch AppBuilder and open the OTA Client ISC file.
3. To validate that image-builder-ecc has been installed correctly:
   a. Press **Preview.**
   b. Select the BATCH file from the drop down list (example: OtaClient.bat).
   c. Examine the final steps in the BATCH file; the last "cmd /c" step will be one of the following:

   An echo command describing why image-builder is not executed. Example:

   ```
   cmd /c "@echo Image-builder executable cannot be found in 'tool/image-builder/image-builder-ecc-windows
   .exe' or 'tool/ota/image-builder/build.windows/image-builder-ecc.exe' > %TARGET_BPATH%-image-builder-ou
   tput.txt"
   ```

   An execution of image-builder with the parameters defined by App Builder. Example:

   ```
   cmd /c "%PROJECT_DIR%\..\..\..\tool\image-builder\image-builder-ecc-windows.exe --create %PROJECT_DIR%\
   OtaSocTest.ota  --version 1 --manuf-id 0x1002 --image-type 0 --tag-id 0x0000 --tag-file %TARGET_BPATH%.
   ebl.encrypted --string "Encrypted EBL OtaClient" --test-sign > %TARGET_BPATH%-image-builder-output.txt"
   ```

4. Click **Generate**.
5. Launch **IAR Workbench**.
6. Open and build the generated project file.

5.6.5 Using a Manufacturer-Specific Certificate

This advanced section uses a test certificate and requires the devices to have Smart Energy 1.x certificates loaded on the nodes. However a test certificate is not secure. For production devices we recommend using a unique manufacturer certificate that is signed by the Production Certificate Authority.

Please see the steps in document AN716, *Instructions for Using Image Builder*, for configuring the OTA client to use a certificate bound to your own manufacturer system.

Note that a ZigBee OTA image file signed with a production certificate can only be used on a Smart Energy device with a production ECC certificate. Test certificates and production certificates are not interchangeable.

The production certificate used for signing should NOT be the same certificate used by any other device. It should be a separate certificate used only for signing.

5.6.6 Loading the Client Software

Follow the directions in the section 4.6 Load the Client Software for loading the client software. Use the newly built ECC-enabled client software.

5.6.7 Loading the Server Software

Follow the directions in the previous section 4.7 Load the NCP Server Software for loading the NCP image. Use the ECC-enabled EM357-EZSP image.

5.6.8 Loading the Client and Server Certificate

In order for the Smart Energy devices to perform key establishment to derive an authorized link key for encrypting communications on the network, a device must have a Smart Energy ECC certificate installed.

Installing a certificate on EM35x devices requires using the **em3xx_load** command-line tool that is installed as part of the Debug Adapter (ISA3) utilities. Installing a certificate on Mighty Gecko (EFR32MG-based) devices requires using Simplicity Commander as described in *UG162: Simplicity Commander User's Guide*.

For details on obtaining and installing certificates for your Silicon Labs ZigBee device, please refer to document *AN714: Smart Energy ECC-Enabled Device Setup Process*.

To verify if a certificate has been correctly set up you can execute the "info" command on the application processor (SOC or Host, depending on your architecture) and look for the following output:

```
OtaClient>info
MFG String:
node [(>)000D6F000092E112] chan [24] pwr [-1]
panID [0x1AAA] nodeID [0x0215] xpan [0x(>)402EEABC45FEF393]
stack ver. [0x46C6]
nodeType [0x02]
Security level [05], Security Profile [SE Full]
SE Security Info [RealEcc RealCbke GoodCert]
network state [02] Buffs: 75 / 75
Ep cnt: 1
ep 1 [enabled] profile [0x0109] devId [0x0502] ver [0x00]
    in (server) cluster: 0x0000 (Basic)
    out(client) cluster: 0x0019 (Over the Air Bootloading Cluster)
    out(client) cluster: 0x0800 (Key establishment)
    in (server) cluster: 0x0800 (Key establishment)
```

The SE Security info line validates that the ECC Library is present (RealECC), the CBKE Library is present (RealCBKE) and there is a certificate installed (GoodCert). If this line does not match the one presented above then one or more of the necessary elements is missing from the application and needs to be corrected before proceeding.

5.6.9 Loading the Signed Image on the Server

Remove any existing OTA client image from the OTA server directory and copy the signed image to the OTA server's directory (otafiles).

5.6.10 Running the Software

The process for running an OTA download with signed images is otherwise the same as without signed images. Two additional steps should be done to verify that the client and server software are ready to be used with a signed image.

1. Verify that the server has a signed image loaded for the client by executing the following command:

```
OtaServer>plugin ota-storage printImages
Image 0
  Header Version: 0x0100
  Header Length:  56 bytes
  Field Control:  0x0000
  Manuf ID:       0x1002
  Image Type:     0x0000
  Version:        0x00000022
  Zigbee Version: 0x0002
  Header String:  Our Test Image
  Image Size:     147052 bytes
  Total Tags: 3
    Tag: 0x0000
      Length: 146880
    Tag: 0x0002
      Length: 48
    Tag: 0x0001
      Length: 50
1 images in OTA storage.
```

You can verify that the image is signed if there are two tags at the end of the image with ID 0x0002 and 0x0001.

2. Verify that the client requires signed images by examining the output of the following command:

```
OtaClient>plugin ota-client info
Client image query info
Manuf ID:         0x1002
Image Type ID:    0x0000
Current Version:  0x00000001
Hardware Version: NA
Query Delay ms:            300000
Server Discovery Delay ms: 600000
Download Delay ms:         0
Run Upgrade Delay ms:      600000
Verify Delay ms:           10
Download Error Threshold:  10
Upgrade Wait Threshold:    10
Allowed Signers of images, big endian (NULL EUI64 is invalid)
0: 00 0D 6F 00 00 19 8B 36
1: 00 00 00 00 00 00 00 00
2: 00 00 00 00 00 00 00 00
```

If you do not see a list of Allowed Signers, then the image does not have support for signature verification. The output above is using the EUI64 of a test signer that is embedded within the image-builder-ecc application. A test-signer certificate is used when the --test-sign option is specified.

Once you have performed these steps, follow the directions in the previous section . The output on the client will appear slightly different and there will be a number of messages printed about the signature calculation. An example is shown below:

```
OtaClient> plugin ota-client start
starting OTA client state machine
Bootload state: Discovering OTA Server
Svc Disc: Starting discovery for cluster 0x0019
Svc Disc: Match found from 0x0000.
Setting OTA Server to 0x0000
Svc Disc: complete.
Bootload state: Get OTA Server EUI
OTA Cluster: setting IEEE address of OTA cluster
Last offset downloaded: 0x00000000
No image found in storage.
Query next image response: New image is available for download.
Starting download, Version 0x00000022
Bootload state: Downloading Image
Download: 0% complete
```

```
Download: 5% complete
...
```

Note: The download may take up to 10 minutes.

If all goes well you should see the following output:

```
Download: 100% complete
Bootload state: Verifying Image
Last offset downloaded: 0x00023E6C
Client Verifying Signature.
Signer Certificate:
03 07 79 29 47 B3 85 0A
95 85 BF 8E 25 C1 9D 8E
86 78 43 4F 58 36 00 0D
6F 00 00 19 8B 36 54 45
53 54 53 45 43 41 01 09
00 00 00 00 00 00 00 00
Signature
03 6B 0D 13 69 5F F9 4E
B3 BA 41 8E C6 C2 6F 20
F0 D9 9E 2A 70 02 21 A5
36 26 D2 59 F5 59 64 93
6E D6 C9 F5 60 37 9F 3D
6B 5E 00 00 05 00 00 00
Starting new digest calculation
Digest Calculate: 0% complete
Digest Calculate: 5% complete
Digest Calculate: 10% complete
Digest Calculate: 15% complete
Digest Calculate: 20% complete
Digest Calculate: 25% complete
Digest Calculate: 30% complete
Digest Calculate: 35% complete
Digest Calculate: 40% complete
Digest Calculate: 45% complete
Digest Calculate: 50% complete
Digest Calculate: 55% complete
Digest Calculate: 60% complete
Digest Calculate: 65% complete
Digest Calculate: 70% complete
Digest Calculate: 75% complete
Digest Calculate: 80% complete
Digest Calculate: 85% complete
Digest Calculate: 90% complete
Digest Calculate: 95% complete
Digest Calculate: 100% complete
Calculated Digest:
E0 99 0F A3 E8 25 C2 9B  59 EC B8 B7 35 E4 24 74
DSA Verify returned: 0x00 (Success)
Starting EBL verification
EBL passed verification.
Custom verification passed: 0x00
Bootload state: Waiting for Upgrade message
Sending Upgrade End request.
OTA Cluster: wait for 0 s
RXed timeOut 0x00000000 s, MAX timeOut 0x00000DBB s
Adding 2000 ms. delay for immediate upgrade.
Countdown to upgrade: 2000 ms
Bootload state: Countdown to Upgrade
Applying upgrade
Executing bootload callback.
Reset info: 0x02 (BTL)
```
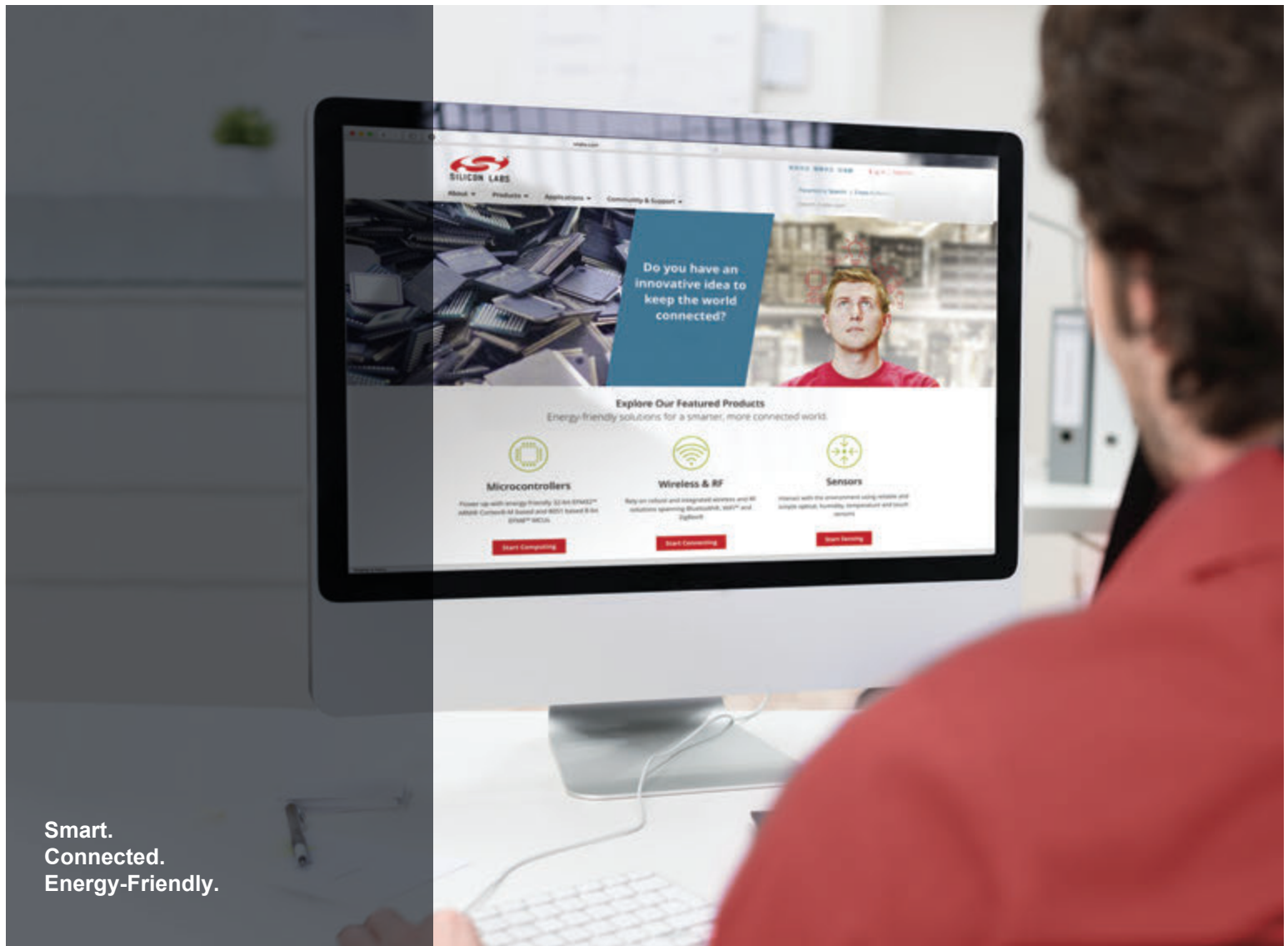
Smart.
Connected.
Energy-Friendly.

Products	**Quality**	**Support and Community**
www.silabs.com/products	www.silabs.com/quality	community.silabs.com

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 and limitation to product information, specifications, and descriptions herein, and does not give warranties as to the accuracy or completeness of the included information. Silicon Labs shall have no liability for the consequences of use of the information supplied herein. This document does not imply or express copyright licenses granted hereunder to design or fabricate any integrated circuits. The products are not designed or authorized to be used within any Life Support System 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.

Trademark Information

Silicon Laboratories Inc.® , Silicon Laboratories®, Silicon Labs®, SiLabs® and the Silicon Labs logo®, Bluegiga®, Bluegiga Logo®, Clockbuilder®, CMEMS®, DSPLL®, EFM®, EFM32®, EFR, Ember®, Energy Micro, Energy Micro logo and combinations thereof, "the world's most energy friendly microcontrollers", Ember®, EZLink®, EZRadio®, EZRadioPRO®, Gecko®, ISOmodem®, Precision32®, ProSLIC®, Simplicity Studio®, SiPHY®, Telegesis, the Telegesis Logo®, USBXpress® 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. 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

http://www.silabs.com