AN1089: Using Installation Codes with Zigbee Devices

This application note provides an overview of using installation codes with zigbee devices. It also explains (with the help of examples) how to use the EM3xx utilities or Simplicity Commander to check, write, verify, and erase installation codes on Silicon Labs EM3x and Wireless Gecko (EFR32™) devices.

KEY FEATURES

- Concepts of the zigbee installation code
- Programming examples for installation codes on EM3x and EFR32 devices
- Checking, writing, verifying, and erasing installation codes on target devices

1 Installation Code Overview

1.1 What Is an Installation Code?

Zigbee installation codes, sometimes also referred to as “install codes,” are provided as a means for a device to join a zigbee network in a reasonably secure fashion. The installation code itself is a random value installed on the joining device at manufacturing time, and is used to encrypt the initial message exchange between it and the Trust Center zigbee network’s centralized Trust Center device (the coordinator). With the creation of the zigbee 3.0 standard in late 2016, all zigbee devices capable of joining networks (as opposed to forming them) must support the use of installation codes during joining as this is a requirement for zigbee 3.0 compliance.

The installation code can be thought of as similar to the PIN code on Bluetooth devices when two devices are paired. The PIN code is provided as an authorization code for the parent device so that the joining device knows it is receiving information securely, such as when a hands-free headset is paired to a smartphone.

The installation code is typically printed on the case or packaging of the device, either as a hexadecimal string or in an encoded fashion such as a barcode or QR code, and provided via an out-of-band mechanism to the Trust Center device or its associated web/cloud interface, along with the 64-bit IEEE MAC address (“EUI64”) of the device. If this device-specific data is stored on a remote web server or cloud-based system, that remote system then securely transports that information to the Trust Center to establish security credentials for the joining device in advance of the in-band joining process.

1.2 Caveats for Zigbee Smart Energy (ZSE) Devices

The Trust Center and the joining device use the installation code as a shared key to establish an initial bond of trust allowing the new device to join the zigbee network. Once the device has successfully joined the network for which it is authorized, zigbee requires that the node negotiate a new Trust Center link key for future secure exchanges with the Trust Center. In traditional zigbee 3.0 networks, this occurs via a key request directly to the Trust Center. However, in zigbee smart energy networks, which behave differently from normal zigbee 3.0 networks, the new Trust Center link key is derived through a special process known as Certificate-Based Key Establishment (CBKE). For more information about the CBKE process, refer to *UG103.5: Fundamentals of Zigbee Security*. Note that the CBKE process requires installing CBKE data certificates signed by Certicom during the manufacturing process. Refer to *AN708: Setting Smart Energy Certificates for Zigbee Devices* for details about how to set these certificate data. Also consult *AN714: Smart Energy ECC-Enabled Device Setup Process* for more information about the requirements for preparing zigbee smart energy (ZSE) devices to be able to join a network and for troubleshooting this process.

This document outlines common practices relating to installation codes for either a standard zigbee 3.0 device or a ZSE device.

2 Security Use

An installation code is used to create a preconfigured, link key. The installation code is transformed into a link key by use on an AES-MMO hash algorithm. For more information and sample code, consult the Install Codes section of the Security chapter of the zigbee alliance’s Base Device Behavior Specification (zigbee document #13-0402).

The installation code, while not exactly a secret, cannot be easily guessed by a malicious device that hears the initial exchange between the joining device and the Trust Center. Without knowledge of the installation code and thus the key, the malicious device cannot decrypt the messages.

The derived zigbee link will be known only by the Trust Center and the joining device. The Trust Center uses that key to securely transport the zigbee network key to the device. Once the device has the network key, it can communicate at the network layer to the zigbee network. It has the ability to perform service discovery and begin the application’s initialization process. In zigbee 3.0 (non-ZSE) networks, having the network key is generally enough for standard messaging across various clusters. However, ZSE networks have additional restrictions as discussed below.

The initial link key derived from the installation code does not have full access privileges on a ZSE network. Attempts to use it for Smart Energy messaging are not allowed and will be ignored by other ZSE devices. Shortly after joining a network, a device must use the Key Establishment cluster to establish a new link key with the Trust Center via the CBKE process. Only when key establishment completes successfully will a device have full privileges on the network and be able send and receive certain ZSE messages.

3 Installation Code Format

While zigbee smart energy networks allow the installation code to be comprised of either 6-, 8-, 12-, or 16-byte random, hexadecimal number with a 2-byte CRC appended to the end, zigbee 3.0 networks specifically require 16-byte hexadecimal installation codes**,** also accompanied by a 2-byte CRC. Note that the CRC16 should be delivered to the user in least significant byte (LSB) order, as this is what is expected when the code is entered into the device that performs the AES-MMO hash algorithm. As far as the user is concerned, the CRC is part of the installation code and they do not need to know that it is there or why. Therefore, from the user's point of view, the length of the install code is 18 bytes (with potentially 8-, 10-, or 14-byte variants possible in ZSE devices).

Manufacturing and managing the list of installation codes will play a part in choosing the size, security, and user experience in installing the device. A larger installation code size will mean less of a chance of an attacker "guessing" the installation code and eavesdropping on the initial join. However smaller installation code is much easier for a user to read off the device during installation.

Note: It is required by the zigbee 3.0 Base Device Behavior Specification that you only use a 16-byte installation code. While this may be more difficult to enter, it provides sufficient strength against an attacker from guessing the installation code and gaining unauthorized access to network or device.

4 Installation Code CRC

The installation code CRC is mechanism used to verify the integrity of an installation code when it is transmitted via an out-of-band mechanism to the utility. This transport mechanism involves human interaction in some way. As a result, the CRC was designed as a way to verify that an installation code is valid and was not mistakenly changed during transport.

The zigbee installation model enables users to install a device themselves. Users simply read the installation code on the back of the device and enter it into a webpage or provide it over the phone to a utility service. Because the number is a hexadecimal value, it is easy to transpose digits or read the wrong value.

4.1 Validation

The zigbee specification expects that the server processing the out-of-band installation code entry from the installer will perform basic checking of the installation code for validity. The server then calculates the CRC over all bytes in the installation code except the final two. It then compares the final two bytes of the installation code with the calculated CRC to see if they match. If they do not match, the user entering the installation code can be informed immediately that it does not look valid. The user should then double-check the value.

Zigbee specifications do not require the Trust Center to validate the installation code directly. (Any validation can be done on a remote web- or cloud-based server if the Trust Center doesn’t have this capability locally.) The Trust Center expects to receive a pre-configured link key along with the EUI64 of the new joining device. It does not need to have any knowledge about how that key was derived. It is up to the particular utility how it wishes to manage and transport the link key to the Trust Center.

For details on how the CRC is calculated, including sample code, consult the Install Codes section of the Security Chapter of the zigbee 3.0 Base Device Behavior Specification (zigbee document #13-0402).

4.2 Generation

Silicon Labs recommends that the installation code be a random number. This reduces the chances of an attacker guessing the installation code and compromising the initial join procedure. The installation code should not be based on the manufacturing process, such as tied to the EUI64 or sequential numbering based on the manufacturing lot.

If that were the case, an attacker with knowledge about the type of device being joined would have a known range of installation codes it could try to compromise the network and clone the device’s identity. An installation code does not have to be unique across all zigbee devices for all manufacturers.

4.3 Labels

The device’s installation code should be printed on a label on the outside of the device along with its EUI64. Both elements should be identified with text indicating what they are. The installation code should not be printed on the outside of the box because that makes it easier for an attacker to gain knowledge of the installation code and potentially compromise the device. It is recommended that the installation code be printed in 2-byte blocks (for example, 83FE D340 7A93 9723 A5C6 39B2 6916 D505 C3B5).

Note: The CRC should be appended to the installation code in little endian format on the label.

4.4 Example

The following is an 18-byte installation code label (16-byte random code with a 2-byte CRC):

```
83FE D340 7A93 9723 A5C6 39B2 6916 D505 C3B5 C3B5
```

The random number portion of the code is the first 16 sequential bytes. The calculated CRC value is 0xB5C3, but it is appended in little-endian format.

5 Programming the Installation Code on a Zigbee Device

5.1 Format of the Installation Code File

To program the installation code, create a simple text file with the value of the installation code (without the CRC). This file is passed into Simplicity Commander.

The format of the file is as follows:

```
Install Code: <ascii-hex>
```

Here is a sample installation code file. The CRC for that code is 0xB5C3 and does not need to be present in the file.

```
Install Code: 83FED3407A939723A5C639B26916D505
```

The installation code must be 16 bytes in length (not including the two-byte CRC).

5.2 Checking the Installation Code on an EM3x Device

To get started, it is best to verify there is connectivity with the device to be programmed, and what information is currently stored on the node. To do this, execute the following command to print all manufacturing token data from an EM3x-based device:

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

You should see output similar to the following, where the highlighted portion below reflects the significant fields related to the installation code:

```
$ em3xx_load.exe --cibtokensprint

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

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

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

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

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

Note: The same installation code token is used for both zigbee smart energy devices and zigbee 3.0 devices, so it is acceptable that the installation code token appears with a note about “Smart Energy Install Code” in the above output.

5.3 Writing the Installation Code into the Manufacturing Area on an EM3x Device

To write the installation code into the manufacturing area, execute the following command:

```
$ em3xx_load.exe --cibtokenspatch install-code-file.txt
```

5.4 Verifying the Stored Installation Code on an EM3x Device

After writing the installation code, it is best to verify the information by executing the following command again:

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

Output of this command should be similar to that shown in **section 5.2, Checking the Installation Code on an EM3x Device**, but with the installation code fields updated according to the `install-code-file.txt` file.

5.5 Checking the Installation Code on an EFR32 Device

To get started, it is best to verify there is connectivity with the device to be programmed, and what information is currently stored on the node. To do this, execute the following command to print all manufacturing token data from an EFR32-based device. The `tokendump` command prints manufacturing token data as key-value pairs. Simplicity Commander supports more than one group of tokens. In this example, the token group named ‘znet’ is used.

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

You should see the following output, where the highlighted portion below reflects the significant fields related to the installation code:

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

```
MFG_XO_TUNE          : 0xFFFF
MFG_LOCKBITS_PLW     : 0x000000000000000000000000FFFFFFFF
MFG_LOCKBITS_CLW0    : 0xFFFFFFFF
MFG_LOCKBITS_MLW     : 0xFFFFFFFF
MFG_LOCKBITS_ULW     : 0xFFFFFFFF
MFG_LOCKBITS_DLW     : 0xFFFFFFFF
MFG_BOOTLOAD_AES_KEY : FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
MFG_SECURITY_CONFIG  : 0xFFFF

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

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

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

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

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

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

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

DONE
```

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

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

DONE
```

5.6 Writing the Installation Code into the Manufacturing Area on an EFR32 Device

To write the installation code into the manufacturing area, execute the following command:

```
$ commander flash --tokengroup znet --tokenfile install-code-file.txt
```

You should see output similar to the following:

```
Writing 2048 bytes starting at address 0x0fe04000
Comparing range 0x0FE04000 - 0x0FE047FF (2 KB)
Programming range 0x0FE04270 - 0x0FE04283 (20 Bytes)
Verifying range 0x0FE04000 - 0x0FE047FF (2 KB)
DONE
```

5.7 Verifying the Stored Installation Code on an EFR32 Device

After writing the installation code, it is best to verify the information by executing the following command again:

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

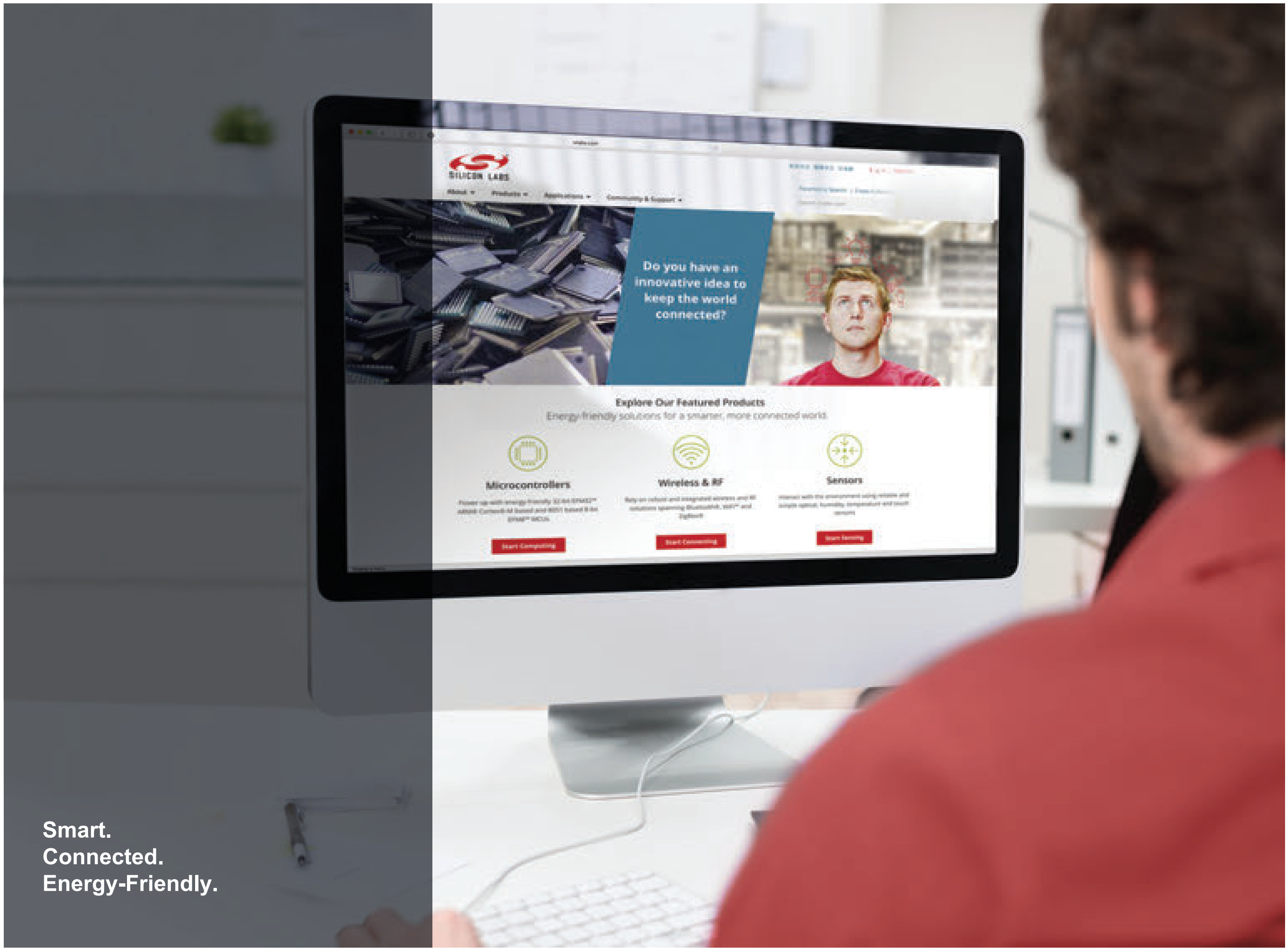
You should see output similar to the yellow-highlighted area of the example code in **section 5.5, Checking the Installation Code on an EFR32 Device**, but with the `MFG_INSTALLATION_CODE` data now representing your chosen code and LSB CRC.

6 Erasing the Installation Code

If you want to remove this security data from the device, simply create an installation code file with the contents as “!ERASE!” such as the example below, and then program this file into the target per the instructions either **section 5.3, Writing the Installation Code into the Manufacturing Area on an EM3x Device** or **section 5.6, Writing the Installation Code into the Manufacturing Area on an EFR32 Device**, depending on your target device.

```
Install Code: !ERASE!
```

Output it similar to that seen in **section 5.3, Writing the Installation Code into the Manufacturing Area on an EM3x Device** and can be confirmed with a procedure similar to **section 5.2, Checking the Installation Code on an EM3x Device** but now your `MFG_INSTALLATION_CODE` token data should reflect all 0xFF bytes.


SILICON LABS
Do you have an innovative idea to keep the world connected?
Explore Our Featured Products
Energy-friendly solutions for a smarter, more connected world.
Microcontrollers
Wireless & RF
Sensors
Smart.
Connected.
Energy-Friendly.

Products
www.silabs.com/products

Quality
www.silabs.com/quality

Support and Community
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®, Micrium, Precision32®, ProSLIC®, Simplicity Studio®, SiPHY®, Telegesis, the Telegesis Logo®, USBXpress®, Zentri 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 LABS®

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

http://www.silabs.com