AN1125: Creating and Using a Secure EZSP Host-to-NCP Interface

This document describes the Secure EmberZNet PRO® Serial Protocol (Secure EZSP) and how to configure the hardware and required software to construct a secure EZSP Host-to-NCP interface.

KEY POINTS

- Introduces the Secure EZSP protocol.
- Describes the host configuration and software contents required for using Secure EZSP.
- Provides detailed instructions for configuring the hardware and using Secure EZSP.
- Includes technical details on the Secure EZSP packet format, cryptographic algorithms, attacks, API commands, and data.

1. Introduction

The EmberZNet PRO® Serial Protocol (EZSP) is the protocol used by a Host processor to interact with the EmberZNet stack running on a network coprocessor (NCP). The Secure EZSP Protocol (Secure EZSP) is an optional protocol that encrypts communications between the Host and NCP.

EZSP messages are sent between the Host and the NCP using Universal Asynchronous Receiver/Transmitter (UART). For more information, refer to:

- *UG100: EZSP Reference Guide* (explains the Host-to-NCP communications and provides details on the EZSP frames)
- *AN706: EZSP-UART Host Interfacing Guide* (describes the EZSP-to-UART communications and the EZSP-UART Protocol)

1.1 ASH and EZSP

The underlying transport for UART is a protocol known as Asynchronous Serial Host (ASH). Silicon Labs developed this protocol to provide transport reliability for the serial connection. ASH handles retries and ensures non-cryptographic packet integrity.

EZSP is the command protocol that runs on top of ASH. EZSP can query the EmberZNet stack for information, send it commands, and pass over-the-air (OTA) Zigbee messages to and from the Host. Secure EZSP is EZSP that has been modified to include authentication and encryption.

1.2 Secure EZSP Protocol Key Generation

Each Host device generates a cryptographically random 128-bit key that both the Host and NCP share for encryption of their UART communications. All identically manufactured devices use a different key. The transport of this key is initially done in-the-clear. This application note assumes that the first-time pairing of the Host and NCP is done in a safe environment (such as in a manufacturing facility) that is protected from eavesdropping. The owner of the Host and NCP can perform subsequent re-pairings as needed. Eavesdropping of these re-pairings would require physical tampering of the UART connection or compromise of one of the devices in the system.

1.3 Secure EZSP Protocol Security Key Storage on the NCP

The Secure EZSP protocol security key is stored in non-volatile memory (NVM) tokens that are read/write. The non-volatile memory model used depends on your part, stack version and, in the case of EFR32 Series 1 parts, preference. See *AN1154: Using Tokens for Non-Volatile Data Storage* for more information.

Table 1.1.

NVM Model	Part Family	EmberZNet Version
SimEEv1/v2	EFR32 Series 1	Current
NVM3	EFR32 Series 1	6.4.0 - Current
NVM3	EFR32 Series 2	6.5.2 - Current

Tokens are defined in every application to support stack behavior. Tokens are also protected by read protection. If Read Protection is enabled, it is impossible to read token contents using a debugger.

1.4 Secure EZSP Protocol Security Key Storage on the Host

The Host reference platform documented in this application note uses a Host token file (.nvm) to store the Secure EZSP protocol security key locally.

2. System Requirements

2.1 Random Number Generator

This application note assumes that both the host and the NCP have access to a cryptographic random number generator. For the NCP, Silicon Labs EmberZNet PRO has access to the radio and can use it to sample noise to generate a random number. The host must have access to its own random number generator and cannot rely on the NCP for this purpose. The Security Support plugin provides sample usage of random number generation for Unix-based host systems.

2.2 Encrypted Bootloader

Secure EZSP assumes that the NCP makes use of an encrypted bootloader to restrict what firmware images can be loaded onto the device. Only firmware images encrypted with a symmetric key are allowed to be flashed onto the device. The symmetric key is normally installed at the time of manufacturing. All images generated by the manufacturer are then encrypted with that key before being distributed to the field.

If an encrypted bootloader is not used, it is possible for a malicious attacker to craft a firmware image that can read any internal address in flash and exfiltrate the contents. This could be used to extract Zigbee security keys or the security key of the Secure EZSP protocol.

2.3 Read Protection of the NCP

This application note assumes that the manufacturer will enable read/write protection of the microcontroller to prevent unauthorized access via the JTAG debug pins.

2.4 Host Security

This application note assumes that the Host will enable the necessary security protections to prevent unauthorized access of the Secure EZSP protocol key.

2.5 Software Requirements

- EmberZNet PRO Release 5.9.1 and above
- Simplicity Studio (version compatible with your EmberZNet SDK)
- Linux or Windows

2.6 Reference Platform

- Linux on an x86 system
- EFR32 NCP

  Note: This application note used the following Host systems for testing purposes: Raspberry Pi and Macintosh.

3. Host Configuration

Customers using host platforms with an NCP must provide their own implementation of Advanced Encryption Standard (AES). Silicon Labs cannot provide a full software implementation of AES for these platforms due to export restrictions. Instead, the EmberZNet PRO software includes a plugin that serves as a wrapper for AES.

Customers are only required to provide the lower-level Application Programming Interfaces (APIs) to perform AES. To minimize the integration burden for customers on Host platforms, the wrapper is written to use the Rijndael cipher upon which AES is based. An implementation of Rijndael is in the public domain and is widely available from third parties on the Internet.

To configure enhanced security on a Host platform, customers must acquire the Rijndael source and save it in a location on the same drive as the EmberZNet PRO software installation. Implementations of Rijndael generally consist of the following files:

- rijndael-alg-fst.c
- rijndael-alg-fst.h
- rijndael-api-fst.c
- rijndael-api-fst.h

The header files rijndael-alg-fst.h and rijndael-api-fst.h are included in the EmberZNet PRO installer to assist customers in locating the corresponding source files. The following file provides implementations of Rijndael that are known to be compatible with EmberZNet PRO as of this writing (click to download):

http://www.efgh.com/software/rijndael.zip

Note: This file source is not affiliated or controlled by Silicon Labs. Silicon Labs makes no guarantee about the availability of the file or the quality or correctness of the implementation.

Once you have acquired Rijndael, perform the following steps to complete the configuration:

1. Set the path for the Rijndael algorithm source in the AES (Software) plugin to the location of rijndael-alg-fst.c.
2. Set the path for the Rijndael API source in the AES (Software) plugin to the location of rijndael-api-fst.c.

4. Software Contents

In addition to EmberZNet PRO Release 5.9.1 or later, Secure EZSP requires the following host plugins, NCP plugins, and sample applications.

Host Plugins

- EZSP Secure Protocol
- AES software plugin (with rijndael source file locations set according to section 3. Host Configuration)
- Security support plugin
- CM* encryption plugin
- If using a Unix library for Unix-based hosts, enable the Token support option in the Unix library plugin.
- EZSP Secure Protocol stub (included to make sure that the EZSP project compiles without errors if the Secure EZSP plugin is not selected in Simplicity Studio)

NCP Plugins

- Secure EZSP
- Secure EZSP stub (included to make sure that the EZSP project compiles without errors if the Secure EZSP plugin is not selected in Simplicity Studio)

Sample Applications (with Secure EZSP plugin enabled)

- XncpLedHost
- xncp-led

5. Getting Started with Secure EZSP

5.1 Configuring the Hardware

1. Set up the hardware as described in *AN706: EZSP-UART Host Interfacing Guide.*
2. Start Simplicity Studio, create the XncpLedHost sample application, and compile it for your Host:
   a. Create a project based on the XncpLedHost application.
   b. Configure AES on Host platform as described in section 3. Host Configuration. Otherwise, the application will not compile.
   c. Generate the application.
   d. Add the necessary plugins listed in Section 4. Software Contents.

      Note: Adding SECURE_EZSP_SERIAL_PRINTF_DEBUG macro to the application, using the Additional Macros section on the Includes tab, will enable helpful printing.

   e. Compile for the respective system (for example, if a Macintosh is the Host, compile the sample application for the Macintosh; if Raspberry Pi is the Host, compile the sample application for the Raspberry Pi).

      For example, go to the generated folder in terminal: <user>:~/Downloads/zigbee_5.9_release/app/builder/XncpLedHost_2 $ make
   f. Once compile is complete, “build success” indicates that the application compiled successfully.
3. Create the NCP sample application and compile it in Simplicity Studio:
   a. Create a project based on the xncp-led sample application.
   b. Configure whether the NCP-UART plugin will use hardware or software flow control.
   c. Be sure to enable the Secure EZSP plugin and disable the Secure EZSP stub plugin.
   d. Generate and compile the application.
   e. Upon successful compile, the screen displays Errors: none. In addition, the binaries are generated and located in the compiler directory in the project (for example, IAR ARM – Default or GNU ARM - Default).
4. Flash the bootloader and then application image onto the NCP.
   a. Right-click on the node, select **Upload application** and browse to the file (for example, XncpLedHost.s37).
   b. Select the **Bootloader image** checkbox and browse to the Bootloader file.

      Note: Silicon Labs provides precompiled standalone Bootloader images in protocol/zigbee_<release version>/tool/bootloader-<architecture>/serial-uart-bootloader/serial-uart-bootloader.s37.

   c. Select the **erase chip before uploading image** checkbox.
   d. Click [**OK**] to flash.
5. Connect the Host with the NCP using UART.

6. Execute the command `./<executable> -p <usb>` on the Host:

   For example: `./build/exe/XncpLedHost -p /dev/tty.usbserial-A5003vbJ`

   where:

   `-p` means the port

   `/dev/tty.usbserial-A5003vbJ` is the path to the connected device

   Note: This command line assumes that the NCP was configured for hardware flow control. Refer to *AN706: EZSP-UART Host Interfacing Guide* for details on executing with other options.

   a. If the connection is successful, you should see initialization printing and a Command Line Interface (CLI) prompt:

```
<user>:~/Downloads/zigbee_5.9_release/app/builder/XncpLedHost_2 $ ./build/exe/XncpLedHost_2 -p /dev/
tty.usbmodem1411
Reset info: 11 (SOFTWARE)
ezsp ver 0x05 stack type 0x02 stack ver. [5.9.0 GA build 103]
Ezsp Config: set source route table size to 0x0007:Success: set
…
Ezsp Policy: set Trust Center Policy to "Allow Joins, ignore preconfigured key rejoins":Success: set
XncpLedHost_2>
```

   b. If the connection is not successful, the terminal screen will display a message similar to this (error appears in **red bold**):

```
<user>:~/Downloads/zigbee_5.9_release/app/builder/XncpLedHost_2 $ ./build/exe/XncpLedHost_2 -p /dev/
tty.usbmodem1411
Reset info: 11 (SOFTWARE)
ERROR: ezspForceReset 0x21
Assertion failed: (false), function emAfResetAndInitNCP, file ../../../protocol/zigbee_5.9/app/
framework/util/af-main-host.c, line 341.
XncpLedHost_2>Child process for serial input got EOF. Exiting
Abort trap: 6
```

   Refer to *UG100: EZSP Reference* and search for the `EzspConfigId` value to learn what the error means and how to address it.

5.2 Using Secure EZSP

1. At the CLI prompt, execute the `info` command to verify that standard, unsecured EZSP Host-to-NCP communications are working correctly:

```
XncpLedHost_2>info
MFG String:
AppBuilder MFG Code: 0x1002
node [(>)000B57FFFE1939B4] chan [0] pwr [-1]
panID [0xFFFF] nodeID [0xFFFE] xpan [0x(>)0000000000000000]
ezsp ver 0x05 stack type 0x02 stack ver. [5.9.0 GA build 103]
nodeType [0x05]
Security level [05]
network state [00] Buffs: 244 / 255
Ep cnt: 1
ep 1 [endpoint enabled, device enabled] nwk [0] profile [0x0104] devId [0x0000] ver [0x00]
    in (server) cluster: 0x0000 (Basic)
    out(client) cluster: 0x0003 (Identify)
    in (server) cluster: 0x0003 (Identify)
    out(client) cluster: 0x0006 (On/off)
Nwk cnt: 1
nwk 0 [Primary (pro)]
  nodeType [0x01]
  securityProfile [0x01]
XncpLedHost_2>
```

2. To use Secure EZSP, generate a random security key by executing this command:

```
XncpLedHost_2>plugin secure-ezsp set_security_key 0
Using /dev/random for random number generation
Gathering sufficient entropy... (may take up to a minute)...
Security Key set { AA D1 67 7A B1 4D 7A EA DE C4 E9 C4 D8 DA B7 49 }
```

Note: `0` means temporary security key (that is, the security key can be changed). Only temporary keys are currently supported.

3. To use Secure EZSP, generate a random Session ID on both the Host and NCP by executing this command:

```
XncpLedHost_2>plugin secure-ezsp set_security_parameters 5
Using /dev/random for random number generation
Gathering sufficient entropy... (may take up to a minute)...
Security Parameters set { B4 4B 8C DD 8C 19 E2 D5 AB 6D 98 46 30 6C 76 87 }
```

Note: `5` means encrypted, 32-bit MIC. Only encrypted, 32-bit MIC is currently supported.

4. Execute the `info` command to verify that Secure EZSP Host-to-NCP encrypted communications are working correctly:

```
XncpLedHost_2>info
MFG String:
AppBuilder MFG Code: 0x1002
node [(>)000B57FFFE1939B4] chan [0] pwr [-1]
panID [0xFFFF] nodeID [0xFFFE] xpan [0x(>)0000000000000000]
ezsp ver 0x05 stack type 0x02 stack ver. [5.9.0 GA build 103]
nodeType [0xE8]
Security level [05]
network state [00] Buffs: 244 / 255
Ep cnt: 1
ep 1 [endpoint enabled, device enabled] nwk [0] profile [0x0104] devId [0x0000] ver [0x00]
    in (server) cluster: 0x0000 (Basic)
    out(client) cluster: 0x0003 (Identify)
    in (server) cluster: 0x0003 (Identify)
    out(client) cluster: 0x0006 (On/off)
Nwk cnt: 1
nwk 0 [Primary (pro)]
  nodeType [0x01]
  securityProfile [0x01]
```

When the Host is compiled with the debug macro, SECURE_EZSP_SERIAL_PRINTF_DEBUG, each Secure EZSP transaction is printed to the terminal such as:

```
-------- HOST ENCODE
EZSP Frame [ 3A 00 81 52 00 01 ]
Secure EZSP Frame [ 3A 00 81 00 E0 BC AB CC C6 A4 8D D4 05 00 00 00 05 1C DD D7 04 8D C4 79 ]
--------------------
-------- HOST DECODE
Secure EZSP Frame [ 3A 80 81 00 70 A7 FC 24 2D 3C 62 6E 05 00 00 00 05 DB DC 87 C7 C9 65 30 88 8F ]
EZSP Frame [ 3A 80 81 52 00 00 FF 00 ]
--------------------
```

Refer to section 6.1 Packet Format to understand how the bytes in the EZSP Frame are encoded in the Secure EZSP Frame and vice versa.

5. To reset ESZP security and start over, execute the `plugin secure-ezsp reset_to_factory_defaults` command:

```
XncpLedHost_2>plugin secure-ezsp reset_to_factory_defaults
```

5.3 Host and NCP Initialization States

The following table summarizes what happens when the Host and NCP are initialized. The result depends on how security is set on the Host and the NCP. A key set in a previous initialization is treated as though the key is unset.

Table 5.1. Host and NCP Initialization States

Host Security	NCP Security	Result
Set	Set	Encrypted communication
Set	Unset	Unencrypted communication
Unset	Set	Fatal error (assert)
Unset	Unset	Unencrypted communication (standard EZSP)

Note: You can change this behavior with `emberSecureEzspInitCallback`. It is currently defined in secure-ezsp-cli.c.

6. Technical Details

6.1 Packet Format

Packet format and the contents of the Authentication Header (AH or Aux Header), Encrypted Payload, and MIC (Message Integrity Code) are illustrated in the following figures.

Unencrypted	Authenticated but not Encrypted	Authenticated and Encrypted

Figure 6.1. Color Legend

ASH	EZSP									ASH
	Sequence (1)	Frame Control (2)		Authentication Header (14)				Encrypted Payload (variable)		MIC (4)	
		Low Byte (1)	High Byte (1)	Control (1)	Source Address (8)	Frame Counter (4)	Nonce Security Level (1)	Frame ID (2)	Frame Payload (variable)		

Figure 6.2. Packet Format

Note: Parentheses contain the number of bytes—for example (1) = 1 byte.

6.1.1 Frame Control

Frame Control Low Byte								
Bit	**7(MSB)**	**6**	**5**	**4**	**3**	**2**	**1**	**0 (LSB)**
Command	0	networkIndex[1]	networkIndex[0]	0 (reserved)	0 (reserved)	0 (reserved)	sleepMode[1]	sleepMode[0]
Response	1	networkIndex[1]	networkIndex[0]	callbackType[1]	callbackType[0]	callbackPending	truncated	overflow

Frame Control High Byte								
Bit	**7(MSB)**	**6**	**5**	**4**	**3**	**2**	**1**	**0 (LSB)**
Command	securityEnabled	paddingEnabled	0 (reserved)	0 (reserved)	0 (reserved)	0 (reserved)	frameFormatVersion[1]	frameFormatVersion[0]
Response	securityEnabled	paddingEnabled	0 (reserved)	0 (reserved)	0 (reserved)	0 (reserved)	frameFormatVersion[1]	frameFormatVersion[0]

Starting with EZSP version 8, the Frame Control consists of two bytes. For example, a secure EZSP command would have 0x00 for the Frame Control Low Byte (a command with nosleeping) and 0x81 for the Frame Control High Byte (security enabled and using frame format version 1). The “frameFormatVersion” is included in EZSP Version 8 and its value is set to 1. With earlier versions of EZSP, the two bits of “frameFormatVersion” are set to 0.

The Frame Control defines a number of bits and all other bits are reserved. Any reserved bits must be set to `0`. If either side of the Secure EZSP protocol sees a reserved bit set to `1`, it must reject the entire frame to indicate a new feature that one side does not support.

6.1.2 Security Enabled

The Security Enabled bit in the **Frame Control High Byte** indicates an Authentication Header, Encrypted Payload, and MIC.

6.1.3 Authenticated Header

The Authentication Header is only present if the **Security Enabled** field is set to `1`. The Authentication Header indicates the parameters of the encrypted payload. It is not encrypted but it is authenticated.

6.1.4 Frame ID

Statring from EZSP version 8, the Frame ID is extended from 1 byte to 2 bytes.

6.2 Cryptographic Algorithms

Secure EZSP leverages IEEE 802.15.4 security as much as possible. This generally means using AES-based algorithms.

6.2.1 Block Cipher

Secure EZSP uses Advanced Encryption Standard-CBC Counter Mode (AES-CCM) with a 128-bit key because AES-CCM only requires AES Encrypt and not AES Decrypt. The AES-CCM cipher creates a block of "noise" and then XORs that with the plaintext or encrypted data to perform the encrypt or decrypt operations. The EM3xx hardware does not expose the AES key schedule necessary to perform the AES Decrypt operation in hardware. IEEE 802.15.4 defines the AES-CCM specifics. Silicon Labs modified it slightly for use in the Secure EZSP protocol.

6.2.2 Hashing

Silicon Labs uses Matyas–Meyer–Oseas (AES-MMO) due to its reliance on AES as the underlying encryption mechanism.

6.2.3 Nonce

Because Secure EZSP uses AES-CCM, it is critical that this protocol never uses the same set of security parameters to encrypt two different messages. If that happens, an attacker can XOR those encrypted messages and obtain the XOR of the two unencrypted message. The nonce is composed of multiple pieces, as described in IEEE 802.15.4-2015, section 9.3.2.1.

Octets: 8	4	1
Source Address	Frame Counter	Nonce Security Level

Normally the source address is composed of the 64-bit Global Identifier (EUI64) as a means to make it unique, while the frame counter is a monotonically increasing value used to prevent replay attacks. The frame counter starts at 0.

All frames except for asynchronous callbacks will use the regular frame counters. Asynchronous callbacks are dispatched by the NCP out-of-order. As a result, there will be a separate outgoing frame counter on the NCP and a separate incoming frame counter on the Host for these callbacks.

Rather than use an EUI64, Silicon Labs creates a Session ID. The Session ID is a randomly-generated number contributed to by both the Host and the NCP and it will be generated at each reboot (during the negotiation phase). Having both the Host and the NCP contribute to the value prevents one side from choosing a number that might have been previously used (either because of a bug or by malicious intent).

The Session ID is created with these steps:

1. Device A (Host) generates a random 128-bit number (Rand-1) and sends it to Device B (NCP).
2. Device B (NCP) generates a random 128-bit number (Rand-2) and sends it to Device A (Host).
3. Both devices construct a string of bits: *Rand-1* || *Rand-2*
4. Both devices perform HMAC using the Secure EZSP Protocol Security Key using AES-MMO as the underlying hashing algorithm.
5. The resulting 16-byte number is split into two 8-byte values as follows: Result = *Session-ID-Host* || *Session-ID-NCP*

6.3 Attacks

6.3.1 Guessing the Key

An attacker can try sending frames encrypted with a key known to the attacker in an attempt to determine the correct key.

Protections

A properly, randomly chosen, 128-bit key would require an astronomically long amount of time to guess correctly.

6.3.2 Observing the Key Transport

It is possible for an attacker to obtain the key by observing the key transport when it initially occurs.

Protections

The key transport can be done in a safe environment, such as in the manufacturing facility. All subsequent communications do not expose the key until the devices are reset to factory defaults. Resetting to factory defaults also wipes out security material associated with the Zigbee network, requiring the user to rejoin the network. Even if the key is obtained, this key is unique to a single set of Host and NCP devices and not global to all devices of that manufacturer.

6.3.3 Replaying Frames

One of the most basic attacks is to record a legitimate encrypted frame and replay it a second time to have the system accept it.

Protections

Each side maintains a RAM copy of the Session ID. Only frames associated with the current Session ID will be accepted. Furthermore, each side of the communication sends a frame with their half of the Session ID and only accepts a received frame with the corresponding other half of the Session ID.

Each side maintains a RAM copy of their outgoing frame counter and a copy of the other side's frame counter as an incoming frame counter. The outgoing frame counter starts from 0 when a session is established. Upon receiving a secured message, the Secure EZSP software running on either the Host or NCP verifies that the message has a frame counter at least 1 higher than the value it currently has for the incoming frame counter. Once a message is received and verified, Secure EZSP records the frame counter of the message as the next minimum incoming frame counter.

Frame counters will not be stored across reboot. Instead, a new Session ID will be established and thus the frame counters can revert to 0 on both sides (incoming and outgoing). Old messages recorded with a different Session ID will not be accepted.

The frame counter cannot wrap from 0xFFFFFFFF to 0. If that ever occurs, a reboot will trigger both sides to establish a new Session ID and reset the frame counters to 0.

6.3.4 Choosing a Session ID

One attack is to get both sides to agree on a Session ID that is chosen by the attacker. Choosing the Session ID allows the attacker to specify the nonce and thus get well-behaving devices to use a previous nonce value.

If one side is replaced with a malicious device, one attack vector is to capture traffic from a session with desirable commands (for example, **Door Unlock**), and then try to get the other side to randomly choose the same Session ID as a previously used one. Once a previous Session ID is accepted, the attacker can replay old messages with old frame counter values and they will be accepted.

Protections

To protect against this potential intrusion Secure EZSP ensures there is enough entropy in the Session ID that is selected and prevents one side from just picking the value (because that could be a malicious device). Using HMAC to generate the Session ID allows both sides to contribute a 128-bit number, while still constraining the size of the Session ID to match the existing nonce format as specified by 802.15.4. The brute force attack of continually rebooting a device to get a desired Session ID is then largely mitigated because the chance of getting the same 64-bit number as a previously recorded session is extremely low.

6.3.5 Guessing a MIC Value

First and foremost this attack requires defeating replay protection.

Another brute force attack that is possible is a malicious device that captures a command, modifies it, and then tries to guess valid MIC values. For example, if a Door Lock command was sent from a Host to an NCP, this could be modified to be a Door Unlock command because the location within the frame of the ZCL Command ID would be known. The command ID can be replaced with a value that equals the **Door Unlock** command ID XOR'ed with the "noise" used to encrypt the **Door Lock** Command ID.

Modifying that command ID does render the MIC invalid but the attacker can now just guess MIC values. Because there are only four bytes of MIC, this means the attacker has 2^{32} guesses and on average will be successful in determining the correct value in half that number of guesses. The mitigation tends to be the rate that guesses can be made, which is limited by the underlying speed of the communication medium. The maximum rate for Zigbee OTA is 250 kb/s. For a point-to-point serial protocol, the rate is 115,280 bytes/second. Because the rate of guesses is slower in a UART link than OTA, it seems unnecessary to protect against this because it would be easier for an attacker to target Zigbee OTA instead.

Protections

To protect against this attach, the NCP returns a failed decryption status to the Host, resulting in a reboot. Rebooting is the most effective because it would trigger a new session ID and frame counter, requiring the attacker to start over their attack by obtaining a new copy of an encrypted command, modifying it, and replaying it to guess the 32-bit MIC.

6.4 Secure EZSP API

Refer to secure-eszp-host.h for the Secure EZSP API commands.

6.5 Data Flow

The following figures illustrate the data flow for the NCP, Host, and security handshake. 'NVM' represents the applicable non-volatile memory model.

6.5.1 NCP

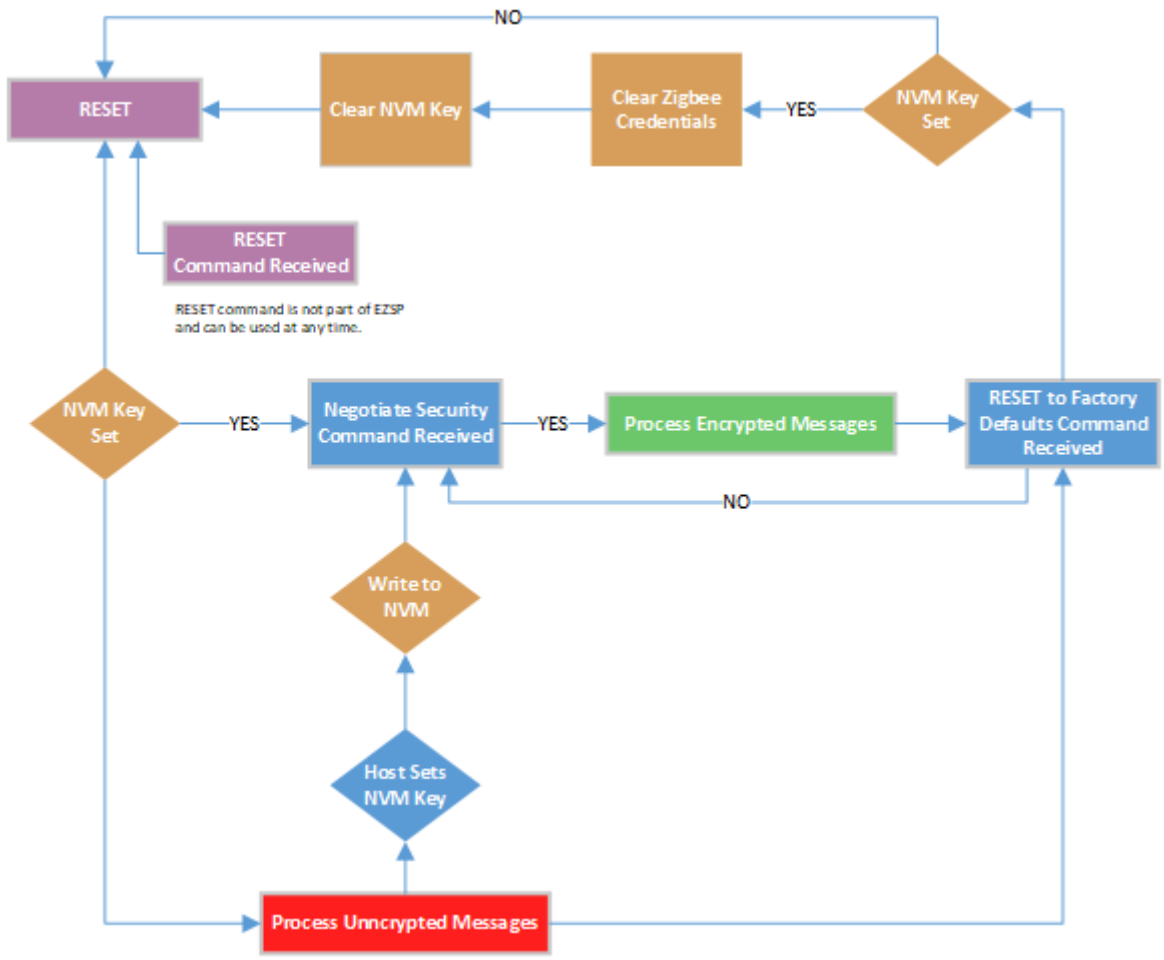


Figure 6.3. NCP Data Flow

6.5.1.1 First Time

1. The NCP Application boots up and looks at its Secure EZSP Security Key in non-volatile memory. If that location is not written, no rewritable Secure EZSP Protocol Security is yet supported.
2. The Host application initiates the ASH connection to the NCP. It can either start communicating using unencrypted EZSP immediately and forgo any security or set up security.
3. The Host Application sets the encryption key by sending the Secure EZSP Security Key in-the-clear to the NCP Application. The Host Application selects the Security Level to be a rewritable key. A permanent manufacturing key is currently not supported.
4. The Host Application sets the parameters of the Secure EZSP protocol (frame counters and nonce).
5. Once negotiated, the NCP and Host switch their messages into only using encryption and authentication with the following exceptions:
   - `RESET` - Allows the Host to trigger a reset of the NCP to re-establish the security parameters.
   - `EZSP_RESET_TO_FACTORY_DEFAULTS` - Allows the Host to reset the NCP to the factory defaults. This first resets all Zigbee credentials on the NCP and then resets the security key.

6.5.1.2 Subsequent Boots

1. The NCP boots and recognizes that its Secure EZSP Security Key is set, so it only allows the following:
   - `EZSP_SET_SECURITY_PARAMETERS` - Allows the Host to negotiate the encryption parameters (frame counters and nonce).
   - `EZSP_RESET_TO_FACTORY_DEFAULTS` - Allows the Host to reset the NCP to the factory defaults. This resets all Zigbee credentials on the NCP first and then resets the security key.
2. Once negotiated, the NCP and Host switch their messages into only using encryption and authentication with the following exceptions:
   - `RESET` - Allows the Host to trigger a reset of the NCP to re-establish the security parameters.
   - `EZSP_RESET_TO_FACTORY_DEFAULTS` - Allows the Host to reset the NCP to the factory defaults. This first resets all Zigbee credentials on the NCP and then resets the security key.

6.5.2 HOST

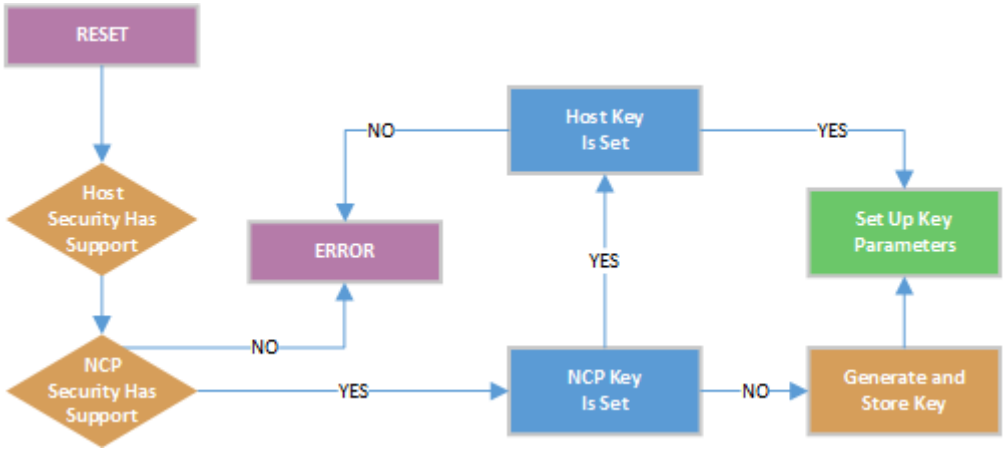


Figure 6.4. Host Data Flow

Note: You can change the initial behavior using `emberSecureEzspInitCallback`.

6.5.3 Security Handshake

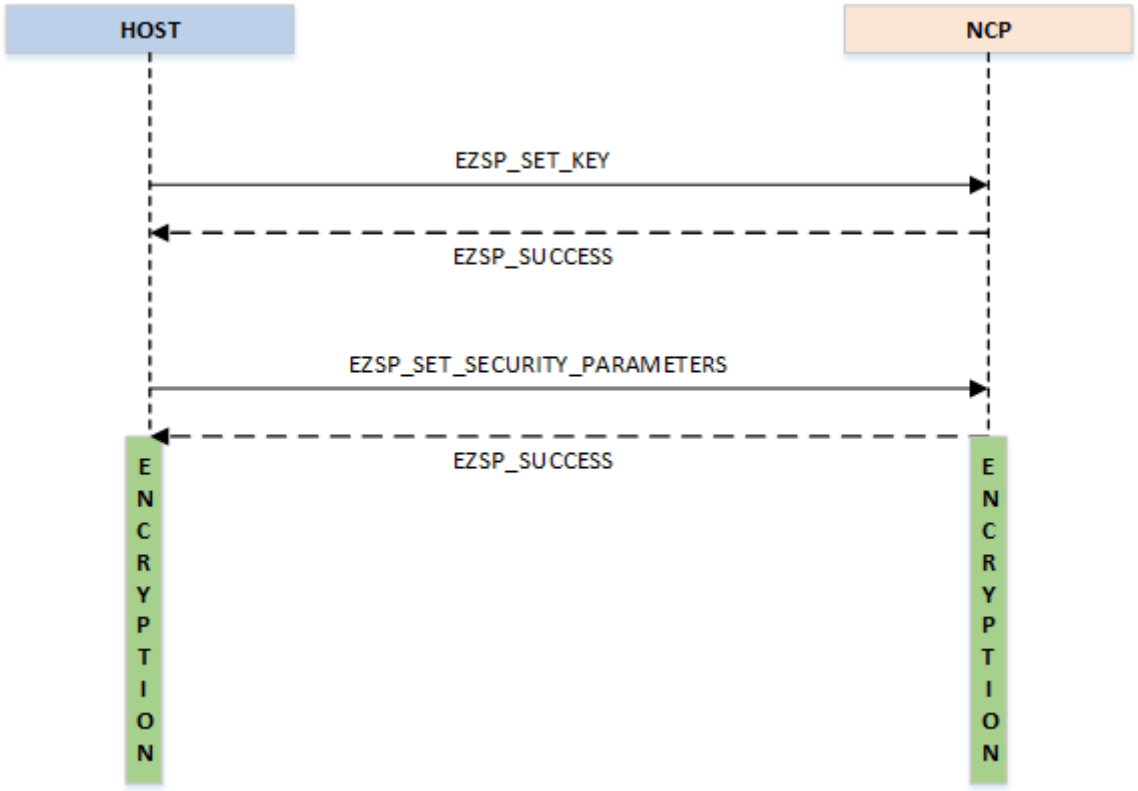
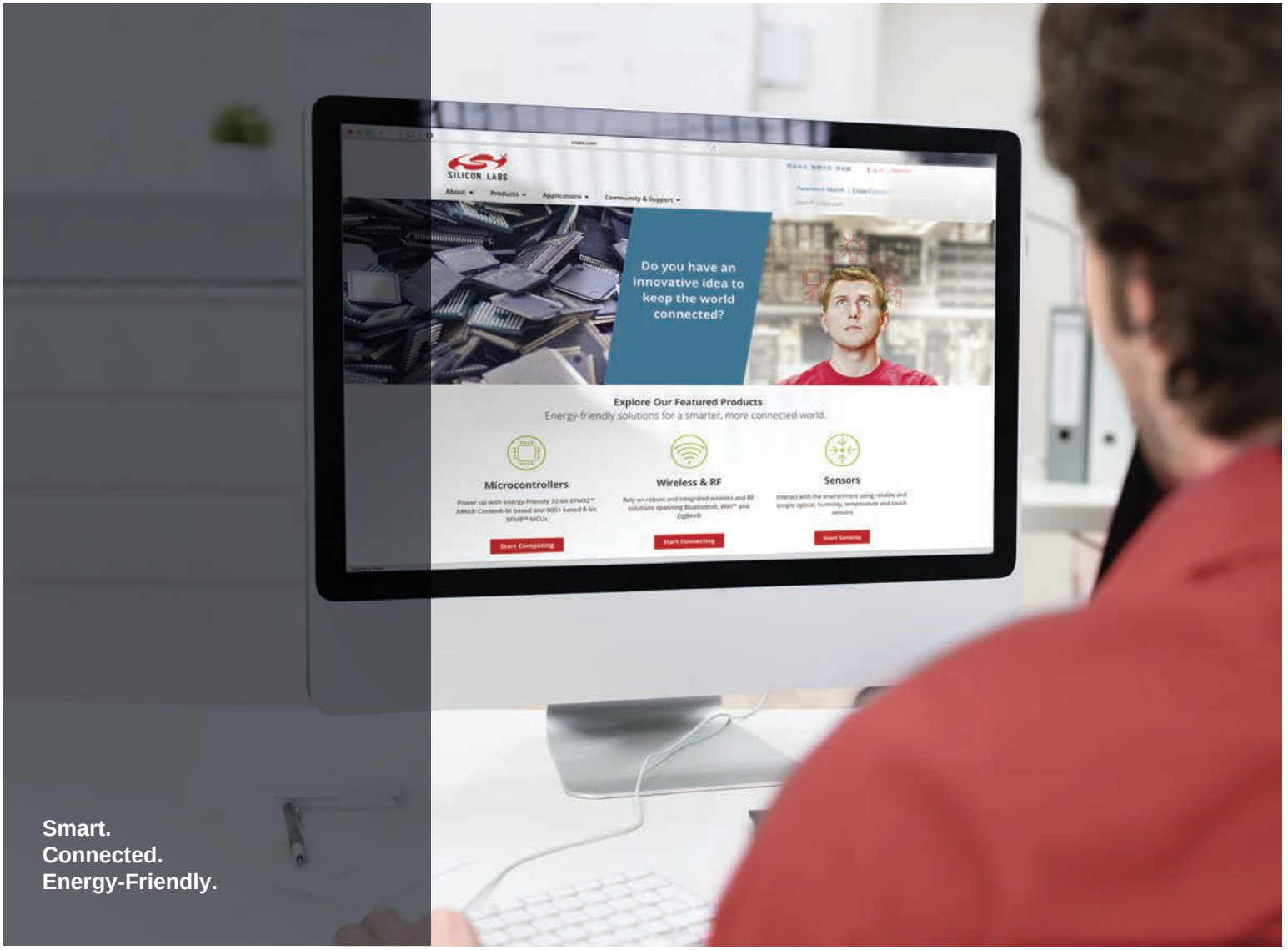


Figure 6.5. Security Handshake Data Flow



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