This application note describes how to lock and unlock the debug access of Series 2 devices. Many aspects of the debug access, including the secure debug unlock, are discussed. The Debug Challenge Interface (DCI) and Mailbox Interface for locking and unlocking debug access are also included.

The debug locks and unlocks for the Cortex-M33 debug interface are implemented through the Secure Element Subsystem on Series 2 devices.

KEY POINTS

- Basic overview of the Secure Element Subsystem.
- The debug port can be accessed by Debug Challenge Interface (DCI) or Mailbox Interface.
- Series 2 devices have new locking and unlocking features.
- Examples for Public Command Key provisioning and Secure Debug Unlock.
1. Device Compatibility

This application note supports Series 2 device families. Some functionality is different depending on the device.

Wireless SoC Series 2 consists of:
- EFR32BG21A/EFR32BG22
- EFR32FG22
- EFR32MG21A/EFR32MG22
2. Introduction

2.1 Debug Lock

All devices require the capability to lock out debug access to the device. This prevents attackers from using the debug interface to perform the following illegal operations:
- Reprogramming the device
- Interrogating the device
- Interfering with the operation of the device

Three different locks can be put on the Series 2 debug interface:
- Standard debug lock
- Permanent debug lock
- Secure debug lock

For production, parts are programmed, tested, and then locked as part of the board level test.

2.2 Debug Unlock

Users need to unlock parts under a number of circumstances:
- To develop code
- Diagnosis of field failures
- Field service of products
- To reprogram already built inventory

Two different unlocks can run on the Series 2 debug interface:
- Standard debug unlock
- Secure debug unlock
3. Secure Element Subsystem

3.1 Overview

The Secure Element Subsystem of Series 2 devices can be implemented by hardware or software. Refer to the device reference manual for details.

The Secure Element (SE) refers to a separate security co-processor that provides security features to the main core.

The Virtual Secure Element (VSE) refers to a collection of capabilities available to the main user core if a separate security co-processor is not provided.

The Secure Element Subsystem is used to perform a series of cryptography operations and other secure system operations. The major operations are described in Table 3.1 Secure Element Subsystem Operations on page 4.

### Table 3.1. Secure Element Subsystem Operations

<table>
<thead>
<tr>
<th>Operation</th>
<th>SE</th>
<th>VSE</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Device Setup</td>
<td>Y</td>
<td>Y</td>
<td>This is a necessary step before secure boot and secure debug unlock can be used, but not necessary for other functionality.</td>
</tr>
<tr>
<td>Secure Boot</td>
<td>Y</td>
<td>Y</td>
<td>This makes the device only capable of running signed software.</td>
</tr>
<tr>
<td>Secure Debug</td>
<td>Y</td>
<td>Y</td>
<td>Secure debug lock and unlock the debug interface of the device.</td>
</tr>
<tr>
<td>Cryptographic Accelerator</td>
<td>Y</td>
<td>—</td>
<td>Acceleration of cryptographic functions.</td>
</tr>
<tr>
<td>TRNG</td>
<td>Y</td>
<td>—</td>
<td>True random number generation.</td>
</tr>
</tbody>
</table>

To start using the secure debug unlock functionality, the device needs to be provisioned. These steps include writing one-time-programmable (OTP) settings to the Secure Element Subsystem for deciding which functionality is enabled, and uploading the Public Command Key to validate a secure debug attempt.

This application note describes how the different debug locks and unlocks for the host are implemented through the Secure Element Subsystem on Series 2 devices.

The Secure Debug is implemented by Root code executed by the Secure Element Core or by the Cortex-M33 operating in Root mode. Table 3.2 Minimum Secure Element Subsystem Firmware Version for Secure Debug on page 4 indicates the minimum required Secure Element (SE) or Virtual Secure Element (VSE) Root code versions that support Secure Debug.

### Table 3.2. Minimum Secure Element Subsystem Firmware Version for Secure Debug

<table>
<thead>
<tr>
<th>Device</th>
<th>Secure Element Subsystem</th>
<th>Minimum Firmware Version for Secure Debug</th>
</tr>
</thead>
<tbody>
<tr>
<td>EFR32xG21A</td>
<td>Secure Element (SE)</td>
<td>Version 1.1.2</td>
</tr>
<tr>
<td>EFR32xG22</td>
<td>Virtual Secure Element (VSE)</td>
<td>Version 1.1.7</td>
</tr>
</tbody>
</table>

**Note:** Silicon Labs strongly recommends installing the latest Secure Element Subsystem firmware on Series 2 devices.

3.2 Command Interface

Interaction with the Secure Element Subsystem is performed over a command interface. The command interface is available through a dedicated Debug Challenge Interface (DCI) as well as through a mailbox interface from the Cortex-M33.

Some commands may not be available at all times and may not be accessible over both interfaces. The DCI interface typically only contains operations for setting up a new device and for locking it down (meant for production processes), while the mailbox interface also contains commands to support crypto operations in Secure Element (SE).
3.2.1 Mailbox

Mailbox operations should not be performed directly, but rather should be executed through either mbed TLS for crypto operations in Secure Element (SE) or the appropriate functions in em_se.c of emlib.

3.2.2 Debug Challenge Interface (DCI)

The Debug Challenge Interface (DCI) is made available through commands in Simplicity Studio and Simplicity Commander. This is the easiest way to access and set up the different security options.
4. Debug Lock

4.1 Overview

The debug access port connected to the Series 2 device's Cortex-M33 processor can be closed by issuing commands to the Secure Element Subsystem, either from a debugger over DCI or through the mailbox interface using `emlib`. Three properties govern the behavior of the debug lock.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description If Set</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Debug Lock</td>
<td>The debug port is kept locked on boot.</td>
<td>False (Disabled)</td>
</tr>
<tr>
<td>Device Erase</td>
<td>The <code>Device erase</code> command is available.</td>
<td>True (Enabled)</td>
</tr>
<tr>
<td>Secure Debug</td>
<td><code>Secure debug unlock</code> is available.</td>
<td>False (Disabled)</td>
</tr>
</tbody>
</table>

The following sections describe how to interact with these properties and how to enable debug locks using the Secure Element Subsystem command interface either over DCI or the mailbox interface using `emlib`. The status of the debug lock can be inspected using the `Get lock status` command.

4.2 Standard Debug Lock

With the default properties (Table 4.2 Standard Debug Lock on page 6) of the debug lock set, the device can be locked using the `Lock` command. Typical flow for this configuration is simply to issue the `Lock` command after the device has been programmed, either using a DCI command from the programming debugger (see 6.2 Standard Debug Lock and Unlock) or through the mailbox interface.

<table>
<thead>
<tr>
<th>Secure Debug</th>
<th>Device Erase</th>
<th>Debug Lock</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Disabled</td>
<td>Enabled</td>
<td>Standard</td>
<td>The <code>Device erase</code> command will wipe the main Flash and RAM, and then a reset will yield an unlocked device.</td>
</tr>
</tbody>
</table>

The standard debug lock behaves similarly to Series 1 devices. The access port can be closed, but issuing a device erase wipes the device and opens the debug port again.

4.3 Permanent Debug Lock

It is possible to disable the `Device erase` command, which permanently enables the debug lock. This can be performed at any time by issuing the `Disable device erase` command, even after the debug lock has been enabled.

<table>
<thead>
<tr>
<th>Secure Debug</th>
<th>Device Erase</th>
<th>Debug Lock</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Disabled</td>
<td>Disabled</td>
<td>Permanent</td>
<td>The part cannot be unlocked. Devices with Permanent Debug Lock engaged cannot be returned for failure analysis.</td>
</tr>
</tbody>
</table>
### 4.4 Secure Debug Lock

If the Device erase command is disabled before locking, the part cannot be unlocked. For secure debug lock, the debug interface can be temporarily enabled by answering a challenge if the Secure debug option is enabled before locking.

#### Table 4.4. Secure Debug Lock

<table>
<thead>
<tr>
<th>Secure Debug</th>
<th>Device Erase</th>
<th>Debug Lock</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enabled1</td>
<td>Disabled2</td>
<td>Secure</td>
<td>Secure debug unlock is enabled, which makes it possible to securely open the debug lock temporarily to reprogram or debug a locked device.</td>
</tr>
</tbody>
</table>

**Note:**
1. Secure debug is enabled in two steps before the debug lock is enabled:
   a. Install the Public Command Key using Simplicity Studio or Simplicity Commander or directly through the Write public key command.
   b. Enable secure debug by issuing the Enable secure debug command.
2. This is an **IRREVERSIBLE** action, and should be the last step in production.
4.5 Debug Lock Command Reference

The commands for debug lock are described in the following table.

<table>
<thead>
<tr>
<th>DCI Command1</th>
<th>Mailbox API (emlib)2</th>
<th>Description</th>
<th>Availability</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lock</td>
<td>SE_debugLockApply();</td>
<td>Enables the debug lock for the part.</td>
<td>While debug is unlocked.</td>
</tr>
<tr>
<td>Get lock status</td>
<td>SE_debugLockStatus(SE_DebugStatus_t *status);</td>
<td>Returns the current debug lock configuration.</td>
<td>Always.</td>
</tr>
<tr>
<td>Disable device erase</td>
<td>SE_deviceEraseDisable();</td>
<td>Disables the Device erase command. This command does not lock the debug interface to the part, but it is an <strong>IRREVERSIBLE</strong> action for the part.</td>
<td>While debug is unlocked.</td>
</tr>
<tr>
<td>Disable secure debug</td>
<td>SE_debugSecureDisable();</td>
<td>Disables the secure debug functionality that can be used to open a locked debug port.</td>
<td>While secure debug is enabled.</td>
</tr>
<tr>
<td>Enable secure debug</td>
<td>SE_debugSecureEnable();</td>
<td>Enables the secure debug functionality that can be used to open a locked debug port through the Get challenge and Open debug commands.</td>
<td>While debug is unlocked and Public Command Key is uploaded.</td>
</tr>
<tr>
<td>Write public key</td>
<td>SE_initPubkey(uint32_t key_type, void* pubkey, uint32_t numBytes, bool signature);</td>
<td>Used during device initialization to upload a single public key. After a key has been written it cannot be changed, and the command will be unavailable for that key.</td>
<td>Available once for each key.</td>
</tr>
<tr>
<td>Read public key</td>
<td>SE_readPubkey(uint32_t key_type, void* pubkey, uint32_t numBytes, bool signature);</td>
<td>Reads the stored public key.</td>
<td>Always.</td>
</tr>
<tr>
<td>Get/Roll challenge3</td>
<td>Unsupported.</td>
<td>Used to roll the current challenge value (16 bytes) to revoke secure debug access.</td>
<td>While Public Command Key is uploaded.</td>
</tr>
</tbody>
</table>

**Note:**

1. Performing these commands over DCI is implemented in Simplicity Studio and Simplicity Commander.
2. Theses APIs are only available on Series 2 devices with Secure Element (SE). These functions are fully described in the Secure Element Subsystem **emlib** online documentation located at [https://docs.silabs.com/mcu/latest/efr32mg21/group-SE](https://docs.silabs.com/mcu/latest/efr32mg21/group-SE).
3. A new challenge will only be generated if the current one has been successfully used at least once.
5. Debug Unlock

5.1 Overview

The debug access port connected to the Series 2 device's Cortex-M33 processor can be opened by issuing commands to the Secure Element Subsystem, usually from a debugger over DCI.

New on the Series 2 devices is the addition of Secure debug unlock functionality. When enabled, it is possible to request a challenge from the device and, by answering the challenge, disable the debug lock until the next power-on or pin reset.

The status of the debug lock can be inspected using the Get lock status command.

5.2 Standard Debug Unlock

With the default properties (Table 4.1 Debug Lock Properties on page 6) of the debug lock set, the device can be unlocked using the Device erase command. This command will wipe and verify that main Flash and RAM is empty before opening the debug lock. It will not wipe user data and provisioned Secure Element Subsystem settings.

5.3 Secure Debug Unlock

In a secure debug unlock scenario, the customer, who has control over the private key for a Secure Element Subsystem, has programmed a Public Command Key into it. The public key is used to verify the signature on a certificate, telling the Secure Element Subsystem what authorization has been given by the holder of the key (customer) to the one issuing the command (customer or delegate). Authorization can be granted, for example, to only unlock debug port on the Cortex-M33, or to only turn off specific tamper signals (if available).

This mode is particularly useful in failure analysis scenarios because it allows devices to be unlocked without losing Flash and RAM contents.
5.3.1 Debug Access Level

The Open debug command takes a parameter (4 bytes) to decide which level of debug access to allow, and a payload containing an access certificate and an unlock command signature. The access level of different bits are given in the following table.

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Enable debug port (should always be set to allow any debug access).</td>
</tr>
<tr>
<td>2</td>
<td>Reserved.</td>
</tr>
<tr>
<td>3</td>
<td>Reserved.</td>
</tr>
<tr>
<td>4</td>
<td>Reserved.</td>
</tr>
<tr>
<td>5</td>
<td>Reserved.</td>
</tr>
<tr>
<td>Other bits</td>
<td>Ignored - must be 0.</td>
</tr>
</tbody>
</table>

**Note:** Bits 1 - 5 must be set to 1 for full debug access.

5.3.2 Access Certificate

Users must generate an access certificate that is included in the unlock token. This access certificate can reduce the frequency of access to the Private Command Key, allowing more restrictive access control on that key. The access certificate also indicates what serial number the unlock token is restricted to and what commands the token is authorized to perform.

An access certificate can be created by Simplicity Commander. The access certificate specifies the following information:
- Magic word — a constant value used to identify the access certificate.
- Authorizations (debug access level) — which debug features can be unlocked.
- Tamper authorizations — which tamper signals (if available) can be turned off.
- Serial number — this number is compared against the serial number of the device to be unlocked.
- Public Ephemeral Key — the public key corresponding to the Private Ephemeral Key that is used to generate the unlock command signature (ECDSA, NIST256p, SHA-256).
- Certificate signature (ECDSA, NIST256p, SHA-256) — all the content of the certificate is signed by Private Command Key.

The Private/Public Ephemeral Key is a randomly generated key pair that is used to create the unlock token and can be reused for signing when device challenge is refreshed.
5.3.3 Challenge Response

The challenge response can be created by Simplicity Commander. The answer to the challenge is a signature (signed by Private Ephemeral Key) over the command word for the Open debug command, the command parameter (debug access level), and the challenge value generated by the device. This signature (ECDSA, NIST256p, SHA-256) is then attached as the final argument to the unlock command.

![Figure 5.3. Challenge Response](image)

The challenge received can be used multiple times before it is refreshed. This makes it possible to give a valid access certificate and an unlock command signature to a third-party to give them temporary debug access to a device without giving them the ability to permanently unlock the device.

The access can then be revoked by rolling the challenge. The Private Ephemeral Key should not be sent to a third party; otherwise anyone can generate a new unlock token for the relocked device.
5.3.4 Debug Access Protocol

Independent debug access for the device is granted using the access certificate and a challenge-response mechanism (see Figure 5.4 Debug Access Protocol of Secure Debug Unlock on page 12).

1. Debugger gets the serial number and challenge from the device.
2. Back-end generates the unlock token with serial number and challenge in one step:
   - Create an access certificate for a specific serial number.
   - Generate the unlock command signature over unlock command and challenge.
3. Debugger sends the unlock token to the device.
4. Device validates the unlock command signature using the Public Ephemeral Key in the certificate.
5. Device validates the serial number and the access certificate signature using the on-device serial number and Public Command Key.
6. Device receives an unlock request with a valid unlock token. Debug access will be unlocked until the next power-on or pin reset.
7. Debugger rolls the challenge to invalidate the current unlock token.

![Figure 5.4. Debug Access Protocol of Secure Debug Unlock](Image)

- **Secure Element Subsystem**: Request serial number → Serial number → Request challenge → Challenge → Unlock command + Debug access level + Certificate + Unlock command signature
- **Debugger**: Check unlock command signature with Public Ephemeral Key in certificate → Verify serial number, check certificate signature with Public Command Key → Unlock authorized debug access level → Response → Roll challenge
- **Back-end (Private Command Key)**: Magic word (4 bytes) → Authorizations (4 bytes) → Tamper authorizations (4 bytes) → Serial number (16 bytes) → Public Ephemeral Key (64 bytes) → Certificate signature (64 bytes) → Signed by Private Command Key → Unlock command word (4 bytes) → Command parameter (4 bytes) → Challenge (16 bytes) → Unlock command signature (64 bytes) → Signed by Private Ephemeral Key

- **Unlock token**: Signed by Private Command Key
5.4 Debug Unlock Command Reference

The commands for debug unlock are described in the following table.

### Table 5.2. Debug Unlock Command Reference

<table>
<thead>
<tr>
<th>DCI Command</th>
<th>Mailbox API (emlib)</th>
<th>Description</th>
<th>Availability</th>
</tr>
</thead>
<tbody>
<tr>
<td>Device erase</td>
<td>SE_deviceErase();</td>
<td>Performs a device mass erase and resets the debug configuration to its initial unlocked state.</td>
<td>While Device erase is enabled.</td>
</tr>
<tr>
<td>Read serial number</td>
<td>SE_serialNumber(void *serial);</td>
<td>This command is used to read out the serial number (16 bytes) of the Series 2 device.</td>
<td>Always.</td>
</tr>
<tr>
<td>Get/Roll challenge</td>
<td>Unsupported.</td>
<td>This command is used to read out and/or roll the current challenge value (16 bytes) for Secure debug unlock.</td>
<td>While Public Command Key is uploaded.</td>
</tr>
<tr>
<td>Open debug</td>
<td>Unsupported.</td>
<td>This command is used to open the secure debug access of the Cortex-M33.</td>
<td>Only when Secure debug is enabled.</td>
</tr>
</tbody>
</table>

**Note:**
1. Performing these commands over DCI is implemented in Simplicity Studio and Simplicity Commander.
2. These APIs are only available on Series 2 devices with Secure Element (SE). These functions are fully described in the Secure Element Subsystem `emlib` online documentation located at [https://docs.silabs.com/mcu/latest/efr32mg21/group-SE](https://docs.silabs.com/mcu/latest/efr32mg21/group-SE).
3. A new challenge will only be generated if the current one has been successfully used at least once.
6. Examples

6.1 Overview

The examples for Series 2 Secure Debug are described in Table 6.1 Secure Debug Examples on page 14.

<table>
<thead>
<tr>
<th>Example</th>
<th>Device</th>
<th>Radio Board</th>
<th>SE or VSE Firmware</th>
<th>Tool</th>
</tr>
</thead>
<tbody>
<tr>
<td>Standard debug lock and unlock¹</td>
<td>EFR32MG21A010F1024IM32</td>
<td>BRD4181A</td>
<td>Version 1.2.1</td>
<td>Simplicity Studio</td>
</tr>
<tr>
<td></td>
<td>EFR32MG22C224F512IM40</td>
<td>BRD4182A</td>
<td>Version 1.2.1</td>
<td>Simplicity Commander</td>
</tr>
<tr>
<td>Provision Public Command Key</td>
<td>EFR32MG21A010F1024IM32</td>
<td>BRD4181A</td>
<td>Version 1.2.1</td>
<td>Simplicity Studio</td>
</tr>
<tr>
<td></td>
<td>EFR32MG22C224F512IM40</td>
<td>BRD4182A</td>
<td>Version 1.2.1</td>
<td>Simplicity Commander</td>
</tr>
<tr>
<td>Secure debug unlock²</td>
<td>EFR32MG21A010F1024IM32</td>
<td>BRD4181A</td>
<td>Version 1.2.1</td>
<td>Simplicity Commander</td>
</tr>
</tbody>
</table>

Note:
1. The standard debug lock and unlock apply on devices with default Debug Lock Properties (Table 4.1 Debug Lock Properties on page 6).
2. The secure debug unlock applies on devices with Secure Debug Lock (Table 4.4 Secure Debug Lock on page 7).
6.1.1 Simplicity Studio

The security operations are performed in Security Settings of Simplicity Studio.

1. Right-click the selected debug adapter **J-Link Silicon Labs (serial number)** to display the context menu.

![Context Menu of Debug Adapters](image1)

Figure 6.1. Context Menu of Debug Adapters

2. Click **Device configuration...** to open the **Configuration of device: J-Link Silicon Labs (serial number)** dialog box. Click the **Security Settings** tab to get the selected device configuration.

![Configuration of Selected Device](image2)

Figure 6.2. Configuration of Selected Device
6.1.2 Simplicity Commander

1. Simplicity Commander’s Command Line Interface (CLI) is invoked by `commander.exe` in the Simplicity Commander folder. The location on Windows is `C:\SiliconLabs\SimplicityStudio\v4\developer\adapter_packs\commander`.

2. Simplicity Commander Version 1.8.2 is used in this application note.

```
commander --version
```

**Simplicity Commander 1v8p2b708**

JLink DLL version: 6.56a
Qt 5.12.1 Copyright (C) 2017 The Qt Company Ltd.
EMDLL Version: 0v17p10b530
mbed TLS version: 2.6.1

Emulator found with SN=440068705 USBAddr=0

DONE

3. If more than one WSTK is connected via USB, the target Wireless Starter Kit (WSTK) must be specified using the `--serialno <J-Link serial number>` option.

4. If the WSTK is in debug mode OUT, the target device must be specified using the `--device <device name>` option.

5. Run the `security genkey` command to generate the Private/Public Command Key pair (`command_key.pem` and `command_pubkey.pem`) for secure debug examples.

```
commander security genkey --type ecc-p256 --privkey command_key.pem --pubkey command_pubkey.pem
```

Generating ECC P256 key pair...
Writing private key file in PEM format to command_key.pem
Writing public key file in PEM format to command_pubkey.pem
DONE

6. Run the `gbl keyconvert` command to generate the Public Command Key text file (`command_pubkey.txt`) for the key provisioning example.

```
commander gbl keyconvert command_pubkey.pem -o command_pubkey.txt
```

Writing EC tokens to command_pubkey.txt...
DONE

For more information about Simplicity Commander, see UG162: Simplicity Commander Reference Guide.

6.1.3 External Tools

1. OpenSSL is used in the secure debug unlock example to sign the access certificate and unlock command. The Windows version of OpenSSL can be downloaded from here — https://slproweb.com/products/Win32OpenSSL.html.

2. The free Hex Editor Neo is used in the secure debug unlock example to edit the binary files generated by Simplicity Commander. The Windows version of Hex Editor Neo can be downloaded from here — https://www.hhdsoftware.com/free-hex-editor.
6.2 Standard Debug Lock and Unlock

6.2.1 Simplicity Studio

1. Open **Security Settings** of the selected device as described in 6.1.1 Simplicity Studio.
2. Click [Enable] on **Enable Debug Lock** to lock the device. The following **Enable Debug Lock Warning** is displayed. Click [Yes] to confirm.

![Figure 6.3. Enable Debug Lock](image)

*AN1190: Series 2 Secure Debug Examples*

**silabs.com | Building a more connected world.**
3. The [Enable] of Enable Secure Debug Unlock: and Enable Debug Lock: will be grayed out after standard debug lock is enabled.

Figure 6.4. Standard Debug Lock
4. Click [Device Erase] to unlock the device.

![Image of Device Erase](image)

**Figure 6.5. Device Erase**
5. The device will restore to the unlock state. Click [OK] to exit.

![J-Link Silicon Labs Configuration](image)

**Figure 6.6. Standard Debug Unlock**
6.2.2 Simplicity Commander

1. Run the `security status` command to get the selected device configuration.

```
commander security status --device EFR32MG22C224F512 --serialno 440068705
```

```
SE Firmware version : 1.2.1
Serial number       : 000000000000000014b457ffed50d1e
Debug lock          : Disabled
Device erase        : Enabled
Secure debug unlock : Disabled
Secure boot         : Disabled
Boot status         : 0x20 - OK
DONE
```

2. Run the `security lock` command to lock the selected device.

```
commander security lock --device EFR32MG22C224F512 --serialno 440068705
```

```
WARNING: Secure debug unlock is disabled. Only way to regain debug access is to run a device erase.
Device is now locked.
DONE
```

3. Run the `security status` command again to check the device configuration.

```
commander security status --device EFR32MG22C224F512 --serialno 440068705
```

```
SE Firmware version : 1.2.1
Serial number       : 000000000000000014b457ffed50d1e
Debug lock          : Enabled
Device erase        : Enabled
Secure debug unlock : Disabled
Secure boot         : Disabled
Boot status         : 0x20 - OK
DONE
```

4. Run the `security erasedevice` command to unlock the selected device.

```
commander security erasedevice --device EFR32MG22C224F512 --serialno 440068705
```

```
Successfully erased device
DONE
```

Note: Issue a power-on or pin reset to complete the unlock process.

5. Run the `security status` command again to check the device configuration.

```
commander security status --device EFR32MG22C224F512 --serialno 440068705
```

```
SE Firmware version : 1.2.1
Serial number       : 000000000000000014b457ffed50d1e
Debug lock          : Disabled
Device erase        : Enabled
Secure debug unlock : Disabled
Secure boot         : Disabled
Boot status         : 0x20 - OK
DONE
```
6.3 Provision Public Command Key

6.3.1 Simplicity Studio

1. Open Security Settings of the selected device as described in 6.1.1 Simplicity Studio.
2. Click [Start Provisioning Wizard…] in the upper right corner to display the Secure Initialization dialog box.

![Secure Initialization Dialog Box]

Figure 6.7. Secure Initialization Dialog Box

3. Click [Next >]. The Security Keys dialog box is displayed.

![Security Keys Dialog Box]

Figure 6.8. Security Keys Dialog Box

4. Open the command_pubkey.txt file generated in 6.1.2 Simplicity Commander step 6.

MFG_SIGNED_BOOTLOADER_KEY_X : F9017F10631575642D7ACF0CCCDB2461DD759923E3B28849EE044AA318112240
MFG_SIGNED_BOOTLOADER_KEY_Y : 0CB4EE5FA74AEEFBC0354B6A4881158741120B0005B6F309E3BFCAC63B898120
5. Check **Enable Writing Command Key**. Copy Public Command Key (F901... first, then 0CB4...) to **Key**: box under **Command Key**:

![Figure 6.9. Public Command Key](image)

6. Click [Next >]. The [Secure Locks] dialog box is displayed. **Enable secure debug unlock** and **Enable debug lock** are set by default.

![Figure 6.10. Security Locks Dialog Box](image)

7. Check or uncheck the corresponding boxes to enable or disable the desired **Debug Locks**.
8. Click [Next >] to display the [Summary] dialog box.

![Figure 6.11. Summary Dialog Box](image)
9. If the information displayed is correct, click [Provision]. Click [Yes] to confirm.

![Device Provisioning Window](image)

**Figure 6.12. Device Provisioning Window**

**Note:** The Public Command Key cannot be changed once written.

10. The **Provisioning Status** is displayed in the **Summary** dialog box.

![Provisioning Status](image)

**Figure 6.13. Provisioning Status**
11. Click [Done] to exit the provisioning process. The device configuration is updated.

**Figure 6.14. Device Configuration after Provisioning**
12. Click [Disable] on **Disable Device Erase**: to disable the device erase. The following **Disable Device Erase Warning** is displayed. Click [Yes] to confirm.

**Figure 6.15. Disable Device Erase**

**Note:** This is an **IRREVERSIBLE** action, and should be the last step in production.
6.3.2 Simplicity Commander

1. Run the `security status` command to get the selected device configuration.

   commander security status --device EFR32MG22C224F512 --serialno 440068705

   SE Firmware version : 1.2.1
   Serial number       : 000000000000000014b457fffed50d1e
   Debug lock          : Disabled
   Device erase        : Enabled
   Secure debug unlock : Disabled
   Secure boot         : Disabled
   Boot status         : 0x20 - OK
   DONE

2. Run the `security writekey` command to provision the Public Command Key with the `command_pubkey.pem` file generated in 6.1.2 Simplicity Commander step 5.

   commander security writekey --command command_pubkey.pem --device EFR32MG22C224F512 --serialno 440068705

   Device has serial number 000000000000000014b457fffed50d1e

   Please look through any warnings before proceeding.
   THIS IS A ONE-TIME command which permanently ties debug and tamper access to certificates signed by this key.
   Type 'continue' and hit enter to proceed or Ctrl-C to abort:

   continue

   DONE

   **Note:** The Public Command Key cannot be changed once written.

3. Run the `security readkey` command to verify the Public Command Key with the `command_pubkey.txt` file generated in 6.1.2 Simplicity Commander step 6.

   commander security readkey --command --device EFR32MG22C224F512 --serialno 440068705

   F9017F10631575642D7ACF0CCCDB2461DD759923E3B28849EE044AA318112240
   0CB4EE5FA74AEEFBC0354B6A488115874112B0005B6F309E3BFCAC63B898120

   DONE

4. Run the `security lockconfig` command to enable the secure debug.

   commander security lockconfig --secure-debug-unlock enable --device EFR32MG22C224F512 --serialno 440068705

   Secure debug unlock was enabled

   DONE

5. Run the `security lock` command to lock the selected device.

   commander security lock --device EFR32MG22C224F512 --serialno 440068705

   Device is now locked.

   DONE
6. Run the `security disabledeviceerase` command to disable device erase.

```
cmdr security disabledeviceerase --device EFR32MG22C224F512 --serialno 440068705

================================================================================
THIS IS A ONE-TIME command which Permanently disables device erase.
If secure debug lock has not been set, there is no way to regain debug access to this device.
Type 'continue' and hit enter to proceed or Ctrl-C to abort:
================================================================================
continue
Disabled device erase successfully
DONE
```

**Note:** This is an **IRREVERSIBLE** action, and should be the last step in production.

7. Run the `security status` command again to check the device configuration.

```
cmdr security status --device EFR32MG22C224F512 --serialno 440068705

SE Firmware version : 1.2.1
Serial number       : 000000000000000014b457fffed50d1e
Debug lock          : Enabled
Device erase        : Disabled
Secure debug unlock : Enabled
Secure boot         : Disabled
Boot status         : 0x20 - OK
DONE
```

6.4 Secure Debug Unlock

6.4.1 Local Secure Debug Unlock

The unlock token can be locally generated if the holder of the Private Command Key can access the device.
Generate the unlock token:

1. Run the `security status` command to get the selected device configuration.

```bash
commander security status --device EFR32MG21A010F1024 --serialno 440068705
```

SE Firmware version : 1.2.1
Serial number : 0000000000000000000d6ffffe0a3a5f
Debug lock : Enabled
Device erase : Disabled
Secure debug unlock : Enabled
Tamper status : OK
Secure boot : Disabled
Boot status : 0x20 - OK
DONE

2. Copy the Private Command Key file (command_key.pem) to the Simplicity Commander folder. Run the `security unlock` command with Private Command Key to unlock the selected device. The debug interface is temporarily unlocked until the next power-on or pin reset.

```bash
commander security unlock --command-key command_key.pem --device EFR32MG21A010F1024 --serialno 440068705
```

Command public key stored in:
C:/Users/amleung/AppData/Local/SiliconLabs/commander/SecurityStore/device_0000000000000000000d6ffffe0a3a5f/command_pubkey.pem
Command private key stored in:
C:/Users/amleung/AppData/Local/SiliconLabs/commander/SecurityStore/device_0000000000000000000d6ffffe0a3a5f/command_key.pem
Authorization file written to Security Store:
C:/Users/amleung/AppData/Local/SiliconLabs/commander/SecurityStore/device_0000000000000000000d6ffffe0a3a5f/certificate_authorizations.json
Generating ECC P256 key pair...
Cert public key stored at:
C:/Users/amleung/AppData/Local/SiliconLabs/commander/SecurityStore/device_0000000000000000000d6ffffe0a3a5f/cert_pubkey.pem
Cert private key stored at:
C:/Users/amleung/AppData/Local/SiliconLabs/commander/SecurityStore/device_0000000000000000000d6ffffe0a3a5f/cert_key.pem
Command key matches public command key found on device. Signing certificate...
Certificate was signed with key:
command_key.pem
Created unsigned unlock command
Signed unlock command using
C:/Users/amleung/AppData/Local/SiliconLabs/commander/SecurityStore/device_0000000000000000000d6ffffe0a3a5f/cert_key.pem
Secure debug successfully unlocked
Command unlock payload was stored in Security Store
DONE

**Note:** The unlock token is generated with the default authorization file (certificate_authorization.json).

3. Run the `device info` command to check that the device is unlocked.

```bash
commander device info --device EFR32MG21A010F1024 --serialno 440068705
```

Part Number : EFR32MG21A010F1024
Die Revision : A1
Production Ver : 0
Flash Size : 1024 kB
SRAM Size : 96 kB
Unique ID : 000d6ffffe0a3a5f
DONE

4. All the generated files, as well as the Private Command Key (command_key.pem), are stored in the Security Store. The location on Windows is C:\Users\<PC user name>\AppData\Local\SiliconLabs\commander\SecurityStore\device_<Serial number>.
5. The unlock token file (unlock_payload_0000000000011110.bin) for secure debug unlock is stored in the challenge_<Challenge value> folder. The location on Windows is C:\Users\<PC user name>\AppData\Local\SiliconLabs\commander\SecurityStore\device_<Serial number>\challenge_<Challenge value>.

![Figure 6.16. Files Generated by Security Unlock Command](image)

6. Send the device and unlock token file (unlock_payload_0000000000011110.bin) to the requesting party for software debugging or failure analysis.

Note: Other files in Security Store should not be sent to the requesting party.

Unlock the device:

1. The unlock token file (unlock_payload_0000000000011110.bin) must be placed in the Security Store to match the device’s <Serial number> and <Challenge value>. The location on Windows is C:\Users\<PC user name>\AppData\Local\SiliconLabs\commander\SecurityStore\device_<Serial number>\challenge_<Challenge value>.

2. Run the **security unlock** command to unlock the device. This unlock token can be reused after power-on or pin reset.

```
commander security unlock --device EFR32MG21A010F1024 --serialno 440068705
```

Unlocking with unlock payload:

```
C:/Users/amleung/AppData/Local/SiliconLabs/commander/SecurityStore/device_0000000000000000000d6ffe0a3a5f/challenge_40884d3d531cd64774dd5f56143c3e3a/unlock_payload_0000000000011110.bin
DONE
```

6.4.2 Remote Secure Debug Unlock

The unlock token can be remotely generated if the holder of the Private Command Key cannot access the device.
Remote secure debug unlock request:

1. Run the `security status` command to get the selected device serial number.

```
communder security status --device EFR32MG21A010F1024 --serialno 440068705
```

- SE Firmware version : 1.2.1
- Serial number : 0000000000000000000d6ffe0a3a5f
- Debug lock : Enabled
- Device erase : Disabled
- Secure debug unlock : Enabled
- Tamper status : OK
- Secure boot : Disabled
- Boot status : 0x20 - OK

DONE

2. Run the `security gencommand` command to retrieve the security challenge from the device and store it in a file (`command_unsign.bin`) with other data as described in Figure 5.3 Challenge Response on page 11.

```
communder security gencommand --action debug-unlock -o command_unsign.bin --nostore
   --device EFR32MG21A010F1024 --serialno 440068705
```

Unsigned command file written to:
command_unsign.bin
DONE

3. Send the device part number (EFR32MG21A010F1024), device serial number (0000000000000000000d6ffe0a3a5f), and unsigned command file (command_unsign.bin) to the holder of the Private Command Key.
Authorize the remote secure debug unlock request (WSTK is not required):

1. Run the `security genkey` command to generate the Private/Public Ephemeral Key pair (`cert_key.pem` and `cert_pubkey.pem`) for the following steps.

   ```
   commander security genkey --type ecc-p256 --privkey cert_key.pem --pubkey cert_pubkey.pem
   ```

   Generating ECC P256 key pair...
   Writing private key file in PEM format to cert_key.pem
   Writing public key file in PEM format to cert_pubkey.pem
   DONE

2. Run the `security gencert` command with device part number, device serial number (from the requesting party), and Public Ephemeral Key generated in step 1 to generate an unsigned access certificate (`access_certificate.bin`) as described in Figure 5.2 Access Certificate on page 10.

   ```
   commander security gencert --device EFR32MG21A010F1024 --deviceserialno 0000000000000000000d6ffffe0a3a5f --cert-pubkey cert_pubkey.pem
   ```

   Authorization file written to Security Store:
   C:/Users/amleung/AppData/Local/SiliconLabs/commander/SecurityStore/device_0000000000000000000d6ffffe0a3a5f/certificate_authorizations.json
   Cert key written to Security Store:
   C:/Users/amleung/AppData/Local/SiliconLabs/commander/SecurityStore/device_0000000000000000000d6ffffe0a3a5f/cert_pubkey.pem
   Certificate was not signed
   DONE

   **Note:** The unsigned access certificate is generated with the default authorization file (`certificate_authorization.json`).

3. All the generated files are stored in the Security Store. The location on Windows is `C:\Users\<PC user name>\AppData\Local\SiliconLabs\commander\SecurityStore\device_<Serial number>`.

   ![Figure 6.17. Files Generated by Security Gencert Command](image)

4. Open the `access_certificate.bin` file to remove the 64 bytes 0x00 reserved for signature.

   ![Figure 6.18. Unsigned Access Certificate File](image)

5. Copy the `access_certificate.bin` file, Private Command Key file (`command_key.pem`), and Public Command Key file (`command_pubkey.pem`) to the Simplicity Commander folder.
6. Use OpenSSL to sign the `access_certificate.bin` file with Private Command Key (`command_key.pem`). The certificate signature is in the `cert_signature.bin` file.

```
openssl dgst -sha256 -binary -sign command_key.pem -out cert_signature.bin access_certificate.bin
```

7. Use OpenSSL to verify the signature in `cert_signature.bin` file with Public Command Key (`command_pubkey.pem`).

```
openssl dgst -sha256 -verify command_pubkey.pem -signature cert_signature.bin access_certificate.bin
```

Verified OK

8. Use OpenSSL to extract the raw signature in `cert_signature.bin` file.

```
openssl asn1parse -inform der -in cert_signature.bin
```

9. Open the `cert_signature.bin` file to remove the ASN.1 headers in signature.

![Access Certificate Signature File]

Figure 6.19. Access Certificate Signature File

10. Use OpenSSL to sign the `command_unsign.bin` file from the request party with Private Ephemeral Key (`cert_key.pem`). The unlock command signature is in the `command_signature.bin` file.

```
openssl dgst -sha256 -binary -sign cert_key.pem -out command_signature.bin command_unsign.bin
```

11. Use OpenSSL to verify the signature in the `command_signature.bin` file with the Public Ephemeral Key (`cert_pubkey.pem`).

```
openssl dgst -sha256 -verify cert_pubkey.pem -signature command_signature.bin command_unsign.bin
```

Verified OK

12. Use OpenSSL to extract the raw signature in the `command_signature.bin` file.

```
openssl asn1parse -inform der -in command_signature.bin
```

![Command Signature File]

![Command Signature File]

![Command Signature File]

![Command Signature File]
13. Open the `command_signature.bin` file to remove the ASN.1 headers in signature.

14. Send the Public Ephemeral Key (`cert_pubkey.pem`), access certificate signature file (`cert_signature.bin`), and unlock command signature file (`command_signature.bin`) to the request party.

Figure 6.20. Unlock Command Signature File
Generate the unlock token:

1. Run the `security gencert` command with the Public Ephemeral Key (`cert_pubkey.pem`) to generate an unsigned access certificate (`access_certificate.bin`) as described in Figure 5.2 Access Certificate on page 10.

   ```
   commander security gencert --cert-pubkey cert_pubkey.pem --device EFR32MG21A010F1024 --serialno 440068705
   ```

   Authorization file written to Security Store:
   C:/Users/amleung/AppData/Local/SiliconLabs/commander/SecurityStore/device_0000000000000000000d6ffffe0a3a5f/
   certificate_authorizations.json

   Cert key written to Security Store:
   C:/Users/amleung/AppData/Local/SiliconLabs/commander/SecurityStore/device_0000000000000000000d6ffffe0a3a5f/
   cert_pubkey.pem

   Certificate was not signed

   **Note:** The unsigned access certificate is generated with the default authorization file (`certificate_authorization.json`).

2. Run the `security unlock` command with Public Ephemeral Key (`cert_pubkey.pem`), access certificate signature file (`cert_signature.bin`), and unlock command signature file (`command_signature.bin`) to unlock the selected device. The debug interface is temporarily unlocked until the next power-on or pin reset.

   ```
   commander security unlock --cert-pubkey cert_pubkey.pem --cert-signature cert_signature.bin
   --command-signature command_signature.bin --device EFR32MG21A010F1024 --serialno 440068705
   ```

   Using certificate from Security Store:
   C:/Users/amleung/AppData/Local/SiliconLabs/commander/SecurityStore/device_0000000000000000000d6ffffe0a3a5f/
   access_certificate.bin

   Certificate in Security Store is not signed.

   Moved existing file to:
   C:/Users/amleung/AppData/Local/SiliconLabs/commander/SecurityStore/device_0000000000000000000d6ffffe0a3a5f/
   archive/access_certificate.bin

   Signed certificate written to Security Store:
   C:/Users/amleung/AppData/Local/SiliconLabs/commander/SecurityStore/device_0000000000000000000d6ffffe0a3a5f/
   access_certificate.bin

   Command signature is valid

   Secure debug successfully unlocked

   Command unlock payload was stored in Security Store

   **DONE**

3. Run the `device info` command to check the device is unlocked.

   ```
   commander device info --device EFR32MG21A010F1024 --serialno 440068705
   ```

   **DONE**

4. All the generated files are stored in the Security Store. The location on Windows is `C:\Users\<PC user name>\AppData\Local\SiliconLabs\commander\SecurityStore\device_<Serial number>`. 

   **AN1190: Series 2 Secure Debug**

   **Examples**

   silabs.com | Building a more connected world.
5. The unlock token file (unlock_payload_0000000000000000111110.bin) for secure debug unlock is stored in the challenge_<Challenge value> folder. The location on Windows is C:\Users\<PC user name>\AppData\Local\SiliconLabs\commander\SecurityStore\device_<Serial number>\challenge_<Challenge value>.

6. The unlock token can be reused with the security unlock command after power-on or pin reset.

```bash
commander security unlock --device EFR32MG21A010F1024 --serialno 440068705
```

Unlocking with unlock payload:

```bash
C:/Users/amleung/AppData/Local/SiliconLabs/commander/SecurityStore/device_0000000000000000000d6fffffe0a3a5f/
challenge_dec1b392f00db097675242652684405a/unlock_payload_0000000000111110.bin
DONE
```

6.4.3 Roll Challenge to Revoke Secure Debug Access

1. Run the security rollchallenge command and reset the device to invalidate the current unlock token. The challenge cannot be rolled before it has been used at least once — that is, by running the security unlock command.

```bash
commander security rollchallenge --device EFR32MG21A010F1024 --serialno 440068705
```

Challenge was rolled successfully.

DONE

2. Run the security unlock command to verify the current unlock token is no longer valid.

```bash
commander security unlock --device EFR32MG21A010F1024 --serialno 440068705
```

Authorization file written to Security Store:

```bash
C:/Users/amleung/AppData/Local/SiliconLabs/commander/SecurityStore/device_0000000000000000000d6fffffe0a3a5f/
certificate_authorizations.json
```

Generating ECC P256 key pair...

Cert public key stored at:

```bash
C:/Users/amleung/AppData/Local/SiliconLabs/commander/SecurityStore/device_0000000000000000000d6fffffe0a3a5f/
cert_pubkey.pem
```

Cert private key stored at:

```bash
C:/Users/amleung/AppData/Local/SiliconLabs/commander/SecurityStore/device_0000000000000000000d6fffffe0a3a5f/
cert_key.pem
```

Certificate was not signed

ERROR: Created an unsigned certificate. Please provide command key with option --command-key to sign the certificate or provide the certificate signature with option --cert-signature

DONE

Or

Using certificate from Security Store:

```bash
C:/Users/amleung/AppData/Local/SiliconLabs/commander/SecurityStore/device_0000000000000000000d6fffffe0a3a5f/
access_certificate.bin
```

ERROR: Need private key corresponding to public key in certificate to sign command. Please provide with option --cert-privkey

DONE
7. Revision History

Revision 0.2
March 2020

• Changed EFR32xG21 to Series 2.
• Changed device compatibility to include EFM32xG22 devices.
• Modified Secure Element section, added Virtual Secure Element (VSE) for EFR32xG22 devices.
• Updated Table 4.4 and Table 4.5.
• Updated Table 5.1 and Table 5.2.
• Updated Figure 5.2, 5.3, and 5.4.
• Updated Secure Debug Unlock section.
• Combined all examples into one section and updated the content.

Revision 0.1
February 2019

• Initial Revision.
Simplicity Studio
One-click access to MCU and wireless tools, documentation, software, source code libraries & more. Available for Windows, Mac and Linux!

IoT Portfolio
www.silabs.com/IoT

SW/HW
www.silabs.com/simplicity

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 to the product information, specifications, and descriptions herein, and does not give warranties as to the accuracy or completeness of the included information. Without prior notification, Silicon Labs may update product firmware during the manufacturing process for security or reliability reasons. Such changes will not alter the specifications or the performance of the product. Silicon Labs shall have no liability for the consequences of use of the information supplied in this document. This document does not imply or expressly grant any license to design or fabricate any integrated circuits. The products are not designed or authorized to be used within any FDA Class III devices, applications for which FDA premarket approval is required, or Life Support Systems without the specific written consent of Silicon Labs. A "Life Support System" is any product or system intended to support or sustain life and/or health, which, if it fails, can be reasonably expected to result in significant personal injury or death. Silicon Labs products are not designed or authorized for military applications. Silicon Labs products shall under no circumstances be used in weapons of mass destruction including (but not limited to) nuclear, biological or chemical weapons, or missiles capable of delivering such weapons. Silicon Labs disclaims all express and implied warranties and shall not be responsible or liable for any injuries or damages related to use of a Silicon Labs product in such unauthorized applications.

Trademark Information
Silicon Laboratories Inc.®, Silicon Laboratories®, SiLabs®, 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®, Gecko OS, Gecko OS Studio, ISOmodem®, Precision32®, ProSLIC®, Simplicity Studio®, SiPHY®, Telegesis, the Telegesis Logo®, USBXpress®, Zentri, the Zentri logo and Zentri DMS, Z-Wave®, and others are trademarks or registered trademarks of Silicon Labs. ARM, CORTEX, Cortex-M3 and THUMB are trademarks or registered trademarks of ARM Holdings. Keil is a registered trademark of ARM Limited. Wi-Fi is a registered trademark of the Wi-Fi Alliance. All other products or brand names mentioned herein are trademarks of their respective holders.

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

http://www.silabs.com