# AN1222: Production Programming of Series 2 Devices

This application note demonstrates how to properly program, provision, and configure Series 2 devices in a production environment.

Series 2 devices contain a Secure Engine, which runs Secure Engine firmware. When a newer version of Secure Engine firmware is released, the firmware may be upgraded either in the production programming process for devices still in manufacturing or via a field update for deployed devices. Keys must be provisioned to the Secure Engine's one-time-programmable (OTP) memory to use the Secure Boot and Secure Debug features.

For more information about Secure Engine, see section "Secure Engine Subsystem" in application note AN1190: Series 2 Secure Debug.

**KEY POINTS**

- It is the customer's responsibility to ensure the Secure Engine firmware is up-to-date
- The Secure Engine firmware can be upgraded via the Serial Wire Debug (SWD) interface
- Secure Engine firmware is protected from downgrade
- Secure Engine's OTP memory prevents re-writing of:
  - GBL Decryption Key
  - Public Sign Key
  - Public Command Key
  - Secure Boot Enable flag and Tamper Configuration

# 1. Series 2 Device Security Features

Protecting IoT devices against security threats is central to a quality product. Silicon Labs offers several security options to help developers build secure devices, secure application software, and secure paths of communication to manage those devices. Silicon Labs' security offerings were significantly enhanced by the introduction of the Series 2 products that included a Secure Engine. The Secure Engine is a tamper-resistant component used to securely store sensitive data and keys and to execute cryptographic functions and secure services.

On Series 1 devices, the security features are implemented by the TRNG (if available) and CRYPTO peripherals.

On Series 2 devices, the security features are implemented by the Secure Engine and CRYPTOACC (if available). The Secure Engine may be hardware-based, or virtual (software-based). Throughout this document, the following abbreviations are used:

- HSE - Hardware Secure Engine
- VSE - Virtual Secure Engine
- SE - Secure Engine (either HSE or VSE)

Additional security features are provided by Secure Vault. Three levels of Secure Vault feature support are available, depending on the part and SE implementation, as reflected in the following table:

| Level (1) | SE Support | Part (2) |
|---|---|---|
| Secure Vault High (SVH) | HSE only (HSE-SVH) | Refer to UG103.05 for details on supporting devices. |
| Secure Vault Mid (SVM) | HSE (HSE-SVM) | " |
| " | VSE (VSE-SVM) | " |
| Secure Vault Base (SVB) | N/A | " |

**Note:**

1. The features of different Secure Vault levels can be found in https://www.silabs.com/security.
2. UG103.05.

Secure Vault Mid consists of two core security functions:

- Secure Boot: Process where the initial boot phase is executed from an immutable memory (such as ROM) and where code is authenticated before being authorized for execution.
- Secure Debug access control: The ability to lock access to the debug ports for operational security, and to securely unlock them when access is required by an authorized entity.

Secure Vault High offers additional security options:

- Secure Key Storage: Protects cryptographic keys by "wrapping" or encrypting the keys using a root key known only to the HSE-SVH.
- Anti-Tamper protection: A configurable module to protect the device against tamper attacks.
- Device authentication: Functionality that uses a secure device identity certificate along with digital signatures to verify the source or target of device communications.

A Secure Engine Manager and other tools allow users to configure and control their devices both in-house during testing and manufacturing, and after the device is in the field.

### 1.1 User Assistance

In support of these products, Silicon Labs offers whitepapers, webinars, and documentation. The following table summarizes the key security documents:

| Document | Summary | Applicability |
|---|---|---|
| AN1190: Series 2 Secure Debug | How to lock and unlock Series 2 debug access, including background information about the SE | Secure Vault Mid and High |
| AN1218: Series 2 Secure Boot with RTSL | Describes the secure boot process on Series 2 devices using SE | Secure Vault Mid and High |
| AN1247: Anti-Tamper Protection Configuration and Use | How to program, provision, and configure the anti-tamper module | Secure Vault High |
| AN1268: Authenticating Silicon Labs Devices using Device Certificates | How to authenticate a device using secure device certificates and signatures, at any time during the life of the product | Secure Vault High |
| AN1271: Secure Key Storage | How to securely “wrap” keys so they can be stored in non-volatile storage. | Secure Vault High |
| AN1222: Production Programming of Series 2 Devices (this document) | How to program, provision, and configure security information using SE during device production | Secure Vault Mid and High |

### 1.2 Key Reference

Public/Private keypairs along with other keys are used throughout Silicon Labs security implementations. Because terminology can sometimes be confusing, the following table lists the key names, their applicability, and the documentation where they are used.

| Key Name | Customer Programmed | Purpose | Used in |
|---|---|---|---|
| Public Sign key (Sign Key Public) | Yes | Secure Boot binary authentication and/or OTA upgrade payload authentication | AN1218 (primary), AN1222 |
| Public Command key (Command Key Public) | Yes | Secure Debug Unlock or Disable Tamper command authentication | AN1190 (primary), AN1222, AN1247 |
| OTA Decryption key (GBL Decryption key) aka AES-128 Key | Yes | Decrypting GBL payloads used for firmware upgrades | AN1222 (primary), UG266/UG489 |
| Attestation key aka Private Device Key | No | Device authentication for secure identity | AN1268 |

### 1.3 SE Firmware

Silicon Labs strongly recommends installing the latest SE firmware on Series 2 devices to support the required security features. Refer to AN1222 for the procedure to upgrade the SE firmware and UG103.05 for the latest SE Firmware shipped with Series 2 devices and modules.

## 2. Overview

More steps are involved in the production programming process of Series 2 devices compared to Series 1 devices. The steps vary if the device is to have Secure Boot enabled or disabled. For more information about Secure Boot, see AN1218: Series 2 Secure Boot with RTSL. Enabling Secure Debug is a recommended step in the process. For more information about Secure Debug, see AN1190: Series 2 Secure Debug.

A general overview of the production programming steps is described in the following sections. Although some steps can be performed using Simplicity Studio 5, this application note will focus more on using Simplicity Commander because it is more suitable for production environments.

Silicon Labs provides Custom Part Manufacturing Service (CPMS) to customize the users' security features and settings.

### 2.1 Production Programming for Secure Boot-Disabled Device

The following figure illustrates the production programming flow for Secure Boot-disabled devices. It is possible to upgrade Series 2 devices deployed in the field without Secure Boot to Secure Boot with RTSL.

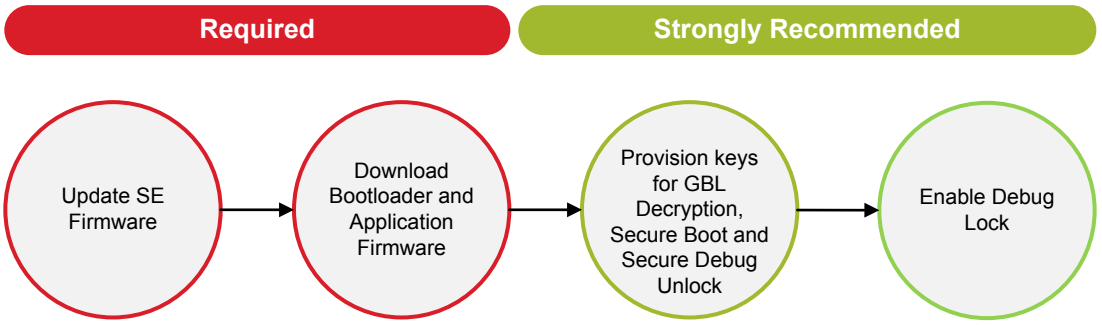



**Figure 2.1. Series 2 High-Level Production Programming Flowchart for Secure Boot-Disabled Devices**

Upgrading the SE Firmware and flashing the bootloader and application firmware are required in the production programming process. Provisioning the GBL Decryption Key for GBL payload decryption, Public Sign Key for Secure Boot, Public Command Key for Secure Debug Unlock, and enabling the Debug Lock are strongly recommended.

A more detailed version of the Series 2 production programming flowchart for a Secure Boot-disabled device is illustrated in the following figure.

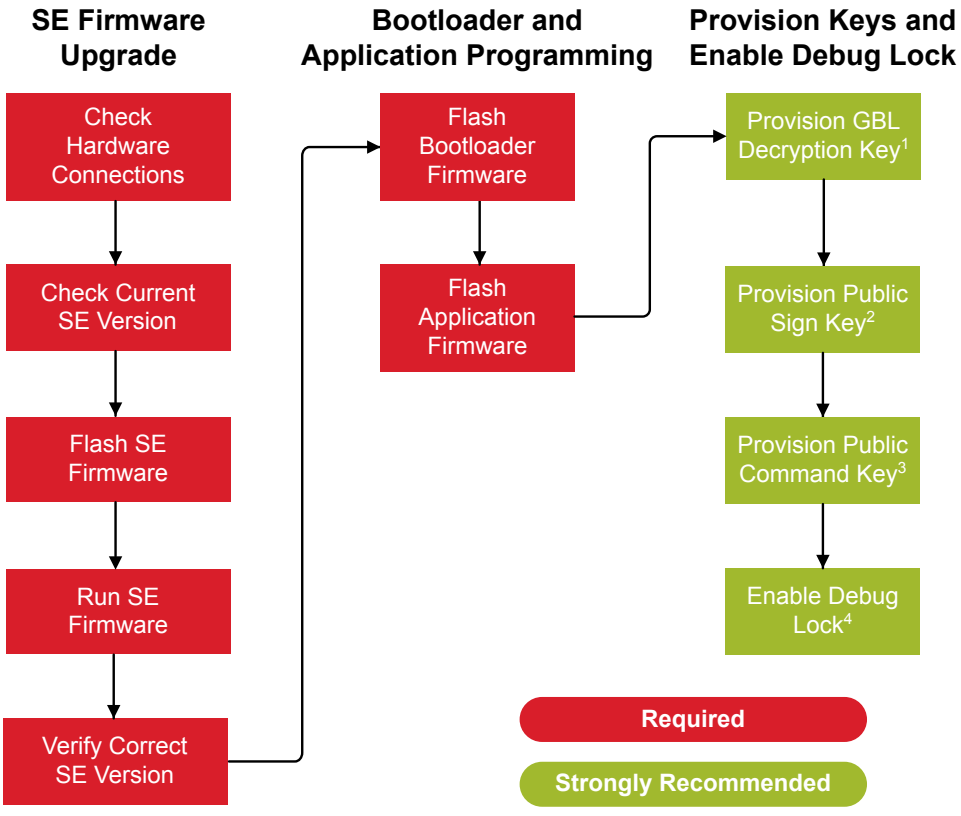



**Figure 2.2. Series 2 Step-by-Step Production Programming Flowchart for a Secure Boot-Disabled Device**

**Note:**

1. Refer to 7.2 Provisioning the GBL Decryption Key in Simplicity Commander on how to program the GBL Decryption Key to the Series 2 device.
2. The VSE devices store a Public Sign Key copy on the top page of the main flash for Secure Boot (see section "Signing for ECDSA-P256-SHA256 Secure Boot" in AN1218: Series 2 Secure Boot with RTSL).
3. The Public Command Key can also be used to temporarily disable anti-tamper protection on HSE-SVH devices (see AN1247: Anti-Tamper Protection Configuration and Use).
4. Enabling the debug lock should be the final step in production, and the following debug lock options are available on the Series 2 device.
   - Standard Debug Lock
   - Permanent Debug Lock
   - Secure Debug Lock (Public Command Key was provisioned)

   **Note:** For more information about these debug lock options, see the section "Debug Lock State Transition" in AN1190: Series 2 Secure Debug.

### 2.2 Production Programming for Secure Boot-Enabled Device

The following figure illustrates the production programming flow for Secure Boot-enabled devices.

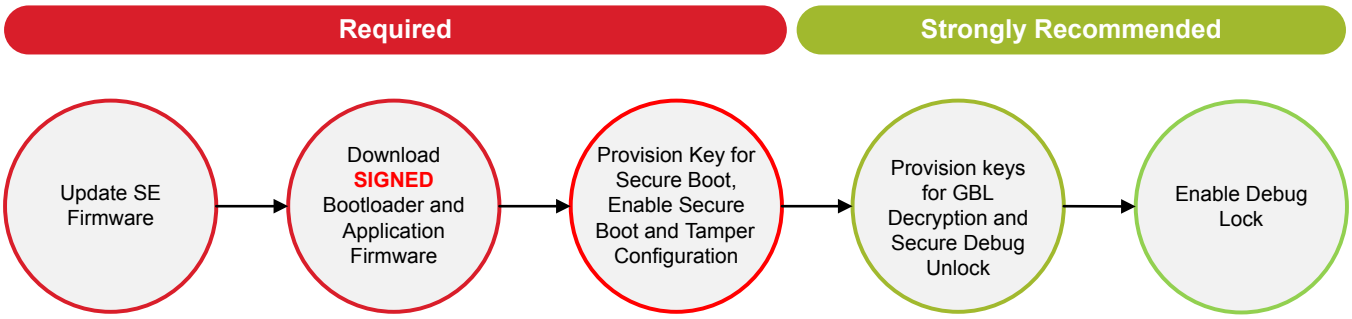



**Figure 2.3. Series 2 High-Level Production Programming Flowchart for Secure Boot-Enabled Devices**

Upgrading the SE Firmware and flashing the **SIGNED** bootloader and application firmware are required in the production programming process. Provisioning the Public Sign Key and enabling Secure Boot and Tamper Configuration (HSE-SVH only) are also needed in the production programming process to enable the Secure Boot option. Provisioning the GBL Decryption Key for GBL payload decryption, Public Command Key for Secure Debug Unlock, and enabling the Debug Lock are strongly recommended.

A more detailed version of the Series 2 production programming flowchart for a Secure Boot-enabled device is illustrated in the following figure.

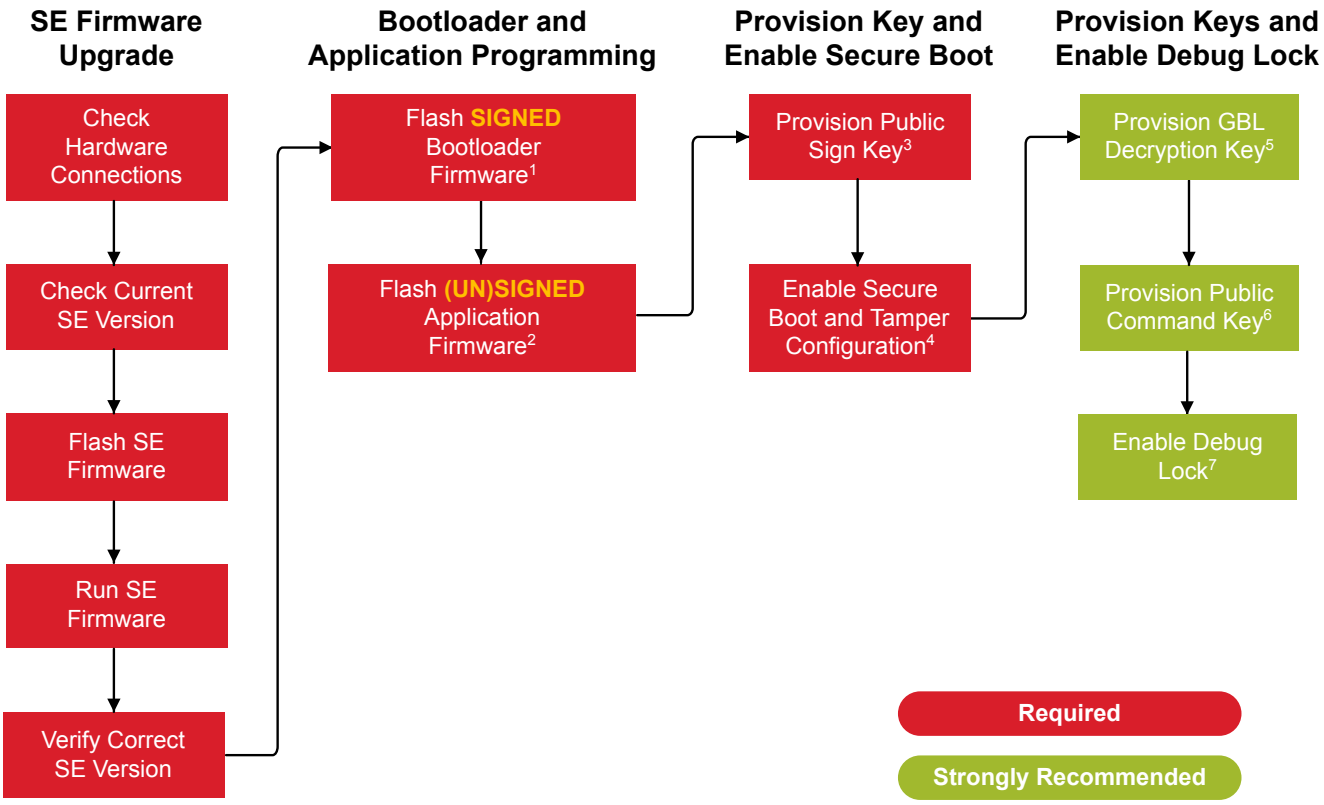



**Figure 2.4. Series 2 Step-by-Step Production Programming Flowchart for a Secure Boot-Enabled Device**

**Note:**

1. The device will enter the Secure Boot failed state if the bootloader firmware is either unsigned or incorrectly signed (see 5. Bootloader Firmware Programming).
2. If the Secure Boot option is enabled in the bootloader, the application firmware must be signed (see 6. Application Firmware Programming).
3. The VSE devices store a Public Sign Key copy on the top page of the main flash for Secure Boot (see section "Signing for ECDSA-P256-SHA256 Secure Boot" in AN1218: Series 2 Secure Boot with RTSL).
4. On HSE-SVH devices, the anti-tamper protection configuration is provisioned with Secure Boot settings (see 8. Enabling Secure Boot and Tamper Configuration).
5. Refer to 7.2 Provisioning the GBL Decryption Key in Simplicity Commander on how to program the GBL Decryption Key to the Series 2 device.
6. The Public Command Key can also be used to temporarily disable anti-tamper protection on HSE-SVH devices (see AN1247: Anti-Tamper Protection Configuration and Use).
7. Enabling the debug lock should be the final step in production, and the following debug lock options are available on the Series 2 device.
   - Standard Debug Lock
   - Permanent Debug Lock
   - Secure Debug Lock (Public Command Key was provisioned)

   **Note:** For more information about these debug lock options, see the section "Debug Lock State Transition" in AN1190: Series 2 Secure Debug.

## 3. Using Simplicity Commander

1. This application note uses Simplicity Commander v1.14.2. The procedures and console output may be different for the other versions of Simplicity Commander. The latest version of Simplicity Commander can be downloaded from https://www.silabs.com/developers/mcu-programming-options.

```
commander --version
```

```
Simplicity Commander 1v14p2b1232

JLink DLL version: 7.70d
Qt 5.12.10 Copyright (C) 2017 The Qt Company Ltd.
EMDLL Version: 0v18p7b669
mbed TLS version: 2.16.6

Emulator found with SN=440048205 USBAddr=0

DONE
```

2. The Simplicity Commander's Command Line Interface (CLI) is invoked by `commander.exe` in the Simplicity Commander folder. The location for Simplicity Studio 5 in Windows is `C:\SiliconLabs\SimplicityStudio\v5\developer\adapter_packs\commander`. For ease of use, it is highly recommended to add the path of `commander.exe` to the system `PATH` in Windows.
3. If more than one Wireless Starter Kit (WSTK) is connected via USB, the target 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.

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

# 4. SE Firmware Programming

## 4.1 Overview

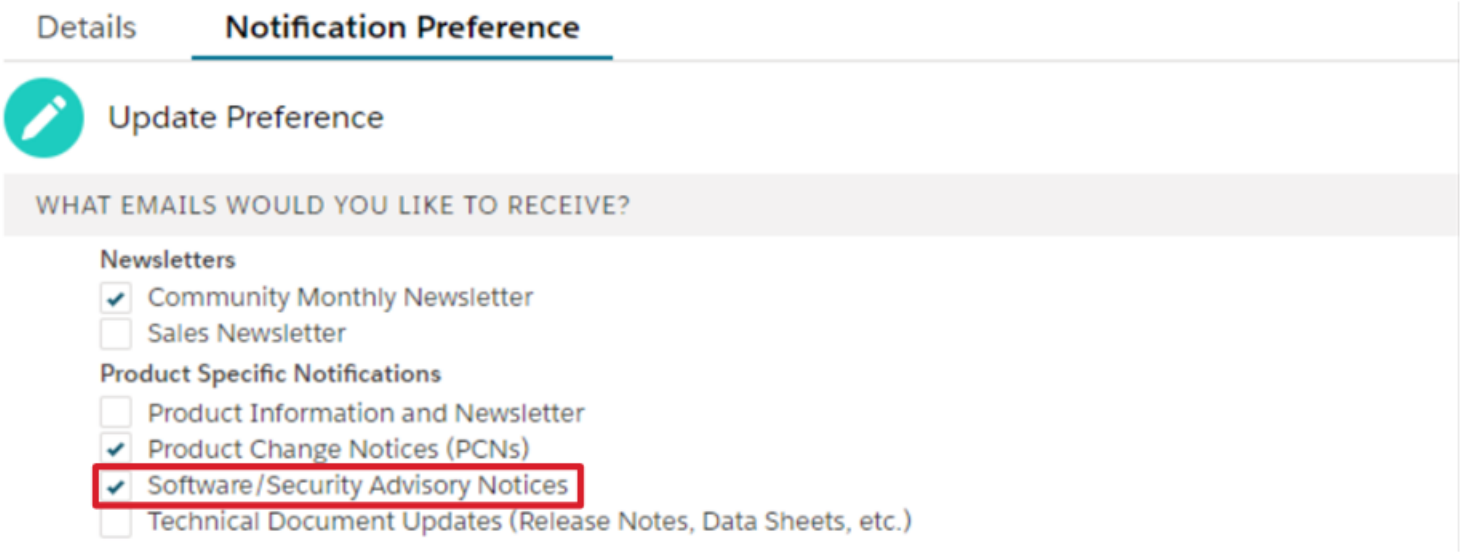
Production programming of Series 2 devices is identical to production programming of Series 1 devices, with the addition of the SE Firmware in the production programming process. Consistent with best practices for Internet of Things (IoT) security, the SE Firmware provided with Series 2 devices supports secure firmware updates. Silicon Labs will periodically release new versions of the SE Firmware to fix bugs and patch vulnerabilities, which may require updates to devices on the manufacturing line or to devices already in the field.

Silicon Labs operates under a "**Security as a Shared Responsibility Model**". This model provides flexibility to system integrators to manage SE Firmware security updates on their own timetable based on their product's use case, risk assessment, agility of their manufacturing flow, and the agility of their field firmware deployment flow.

Series 2 devices are rarely shipped with the latest SE Firmware installed, meaning system integrators must add SE Firmware programming to their production programming flow.

In all cases, Silicon Labs recommends that system integrators:

- Subscribe to security notifications by managing their notification settings in the Silicon Labs Support Portal. This is the easiest method to be notified of SE Firmware updates and discovered vulnerabilities.




- Ensure they are installing the latest SE Firmware release in their manufacturing line.
- Be prepared to deploy security-related field updates to devices in the field.

## 4.2 How to Check the SE Firmware Version on a Device

The SE Firmware version of the device can be found in two ways.

- Simplicity Studio
- Simplicity Commander

### 4.2.1 Check the SE Firmware Version Using Simplicity Studio 5

This application note uses Simplicity Studio v5.2.1.1. The procedures and pictures may be different for the other versions of Simplicity Studio 5.

1. Change the Wireless Starter Kit (WSTK) **Debug Mode:** to **External Device (OUT)**.
2. Right-click the selected debug adapter **Custom Board (ID:J-Link serial number)** to display the context menu.

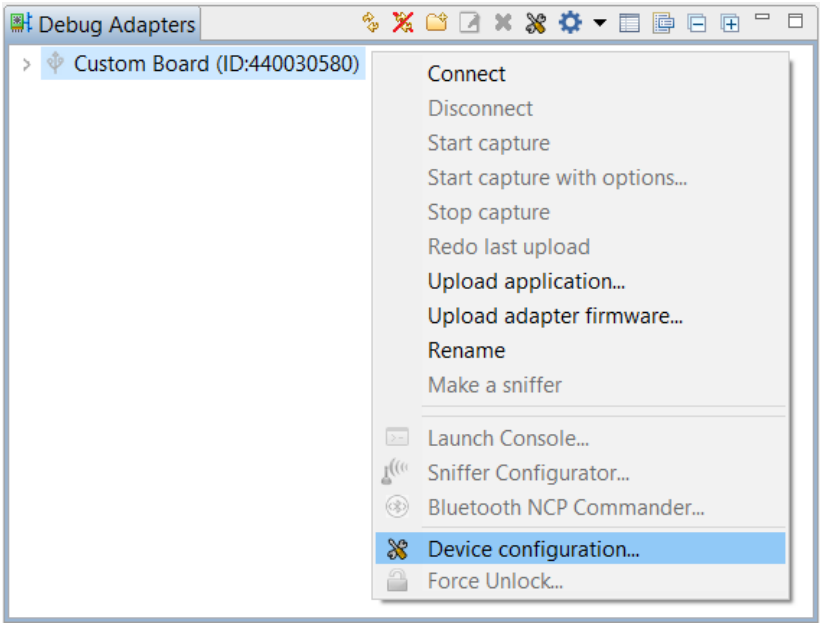



**Figure 4.1. Context Menu of Debug Adapters**

3. Click **Device configuration...** to open the **Configuration of device: J-Link Silicon Labs (serial number)** dialog box. Click the **Device hardware** tab to enter the part number in the **Target part:** box.

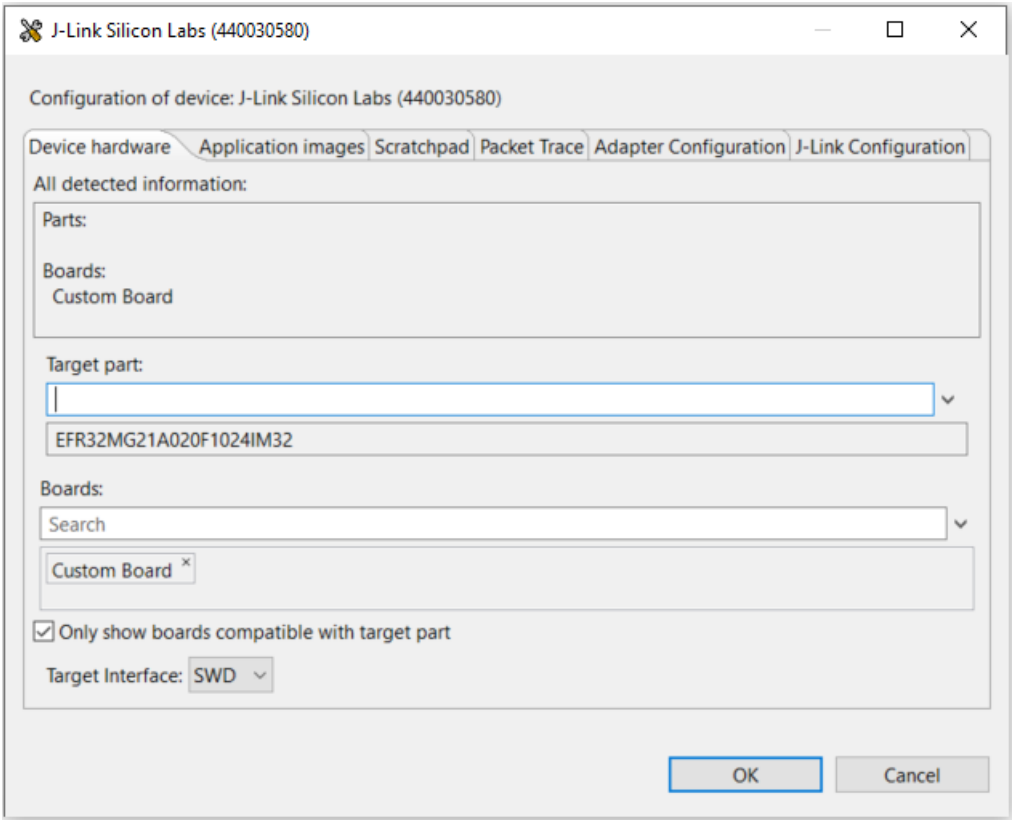



**Figure 4.2. Configuration of Device**

4. Click [**OK**] to exit.

5. Connect the device to the WSTK. Select the device in the **Debug Adapters** view.

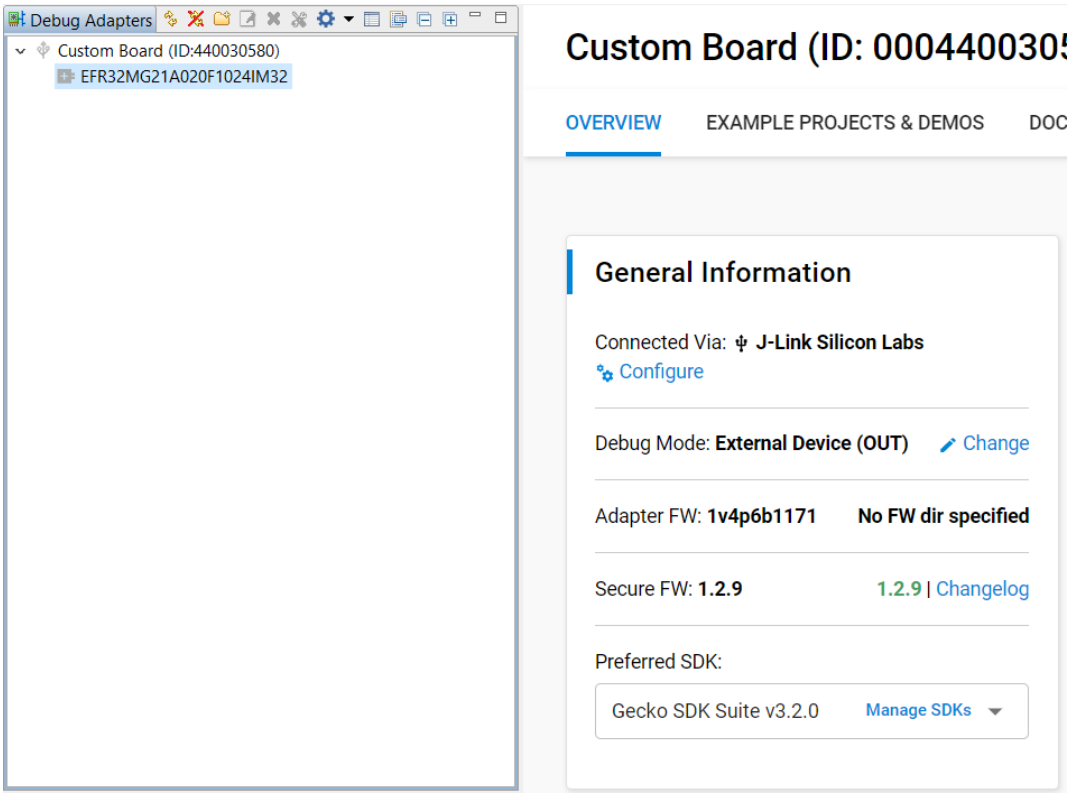



**Figure 4.3. SE Firmware Version in Simplicity Studio**

6. The SE Firmware version will appear in the **Secure FW:** row. In this example, the SE Firmware version on the EFR32MG21A is 1.2.9.

### 4.2.2 Check the SE Firmware Version Using Simplicity Commander

To check the SE Firmware version on the device, issue the Simplicity Commander security command `security status`.

```
commander security status --device EFR32MG21A010F1024 --serialno 440048205
```

```
SE Firmware version : 1.2.14
Serial number       : 000000000000000014b457fffe045a8e
Debug lock          : Disabled
Device erase        : Enabled
Secure debug unlock : Disabled
Tamper status       : OK
Secure boot         : Disabled
Boot status         : 0x20 - OK
DONE
```

In this example, the SE Firmware version on the EFR32MG21A is 1.2.14.

## 4.3 How to Find the Latest SE Firmware

Silicon Labs strongly recommends installing the latest SE firmware on Series 2 devices to support the required security features. The latest SE firmware image (`.seu` and `.hex`) and release notes can be found in the Windows folder below.

For GSDK v3.2 and lower:

```
C:\SiliconLabs\SimplicityStudio\v5\developer\sdks\gecko_sdk_suite\<GSDK VERSION>\util\se_release\public
```

For GSDK v4.0 and higher:

```
C:\Users\<PC USER NAME>\SimplicityStudio\SDKs\gecko_sdk\util\se_release\public
```

### 4.4 Serial Wire Debug (SWD)

The SE Firmware cannot be directly programmed to the SE using the SWD interface. Instead, an image containing the loader application and SE Firmware is flashed onto the host MCU. The SE Firmware is encrypted, versioned, and signed.

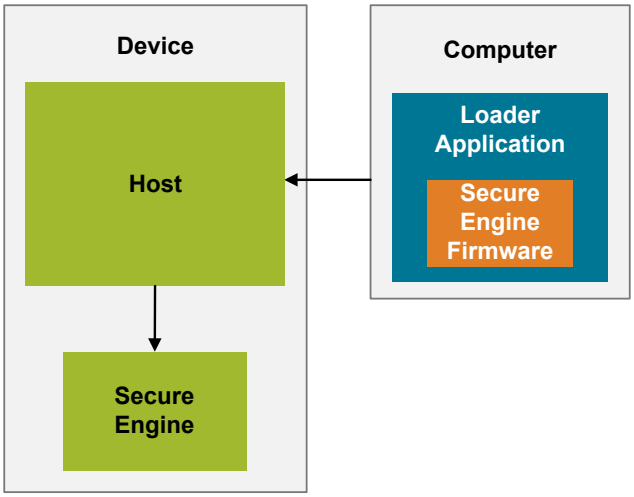



**Figure 4.4. SWD SE Firmware Upgrade Block Diagram**

Using the SWD interface, the user flashes the loader application onto the host. The host then runs the loader application, which checks the signature and version of the SE Firmware. If the signature check passes and the upgrade's version number is higher than the device's SE Firmware version, the firmware is applied to the SE.

The upgrade will not be applied if the signature check fails or if the upgrade's version number is less than or equal to the device's SE Firmware version. Trying to apply a lower SE Firmware version to the device does no harm, but the upgrade will be ignored. This also means the device's SE Firmware cannot be downgraded.

After the SE Firmware has been upgraded, the loader application can be deleted and the application firmware can be flashed via the SWD interface.

As detailed in Figure 2.2 Series 2 Step-by-Step Production Programming Flowchart for a Secure Boot-Disabled Device on page 5 or Figure 2.4 Series 2 Step-by-Step Production Programming Flowchart for a Secure Boot-Enabled Device on page 7, the steps to upgrade the SE Firmware are:

1. Connect Hardware: Connect the device's SWD interface with the WSTK and ensure proper connections.
2. Check Version: Check the SE Firmware version already on the device.
3. Flash SE Firmware: Flash the loader application onto the host processor.
4. Run: Allow the loader application to run and install the SE Firmware.
5. Re-Check Version: Ensure the update succeeded.

Each of these steps is described in more detail in the next sections.

#### 4.4.1 Connect Hardware

After connecting the device's SWD interface to the WSTK, try to read the device information using Simplicity Commander, to verify that proper connections were established to the device.

```
commander device info --device EFR32MG21A010F1024 --serialno 440048205
```

```
Part Number    : EFR32MG21A010F1024IM32
Die Revision   : A1
Production Ver : 2
Flash Size     : 1024 kB
SRAM Size      : 96 kB
Unique ID      : 14b457fffe045a8e
DONE
```

### 4.4.2 Check Version

To check the SE Firmware version on the device, issue the Simplicity Commander security command `security status`.

```
commander security status --device EFR32MG21A010F1024 --serialno 440048205
```

```
SE Firmware version : 1.2.13
Serial number       : 000000000000000014b457fffe045a8e
Debug lock          : Disabled
Device erase        : Enabled
Secure debug unlock : Disabled
Tamper status       : OK
Secure boot         : Disabled
Boot status         : 0x20 - OK
DONE
```

**Note:** The `Tamper status` item is device-dependent.

### 4.4.3 Flash SE Firmware

To flash the SE Firmware upgrade application, run

```
commander flash --masserase s2c1_se_fw_upgrade_app_1v2p14.hex --device EFR32MG21A010F1024 --serialno 440048205
```

where `s2c1_se_fw_upgrade_app_1v2p14.hex` is replaced with the name of the SE Firmware upgrade application file.

```
Parsing file s2c1_se_fw_upgrade_app_1v2p14.hex...
Erasing chip...
Flash was erased successfully
Writing 57344 bytes starting at address 0x00000000
Comparing range 0x00000000 - 0x0000DFFF (56 KB)
Programming range 0x00000000 - 0x00001FFF (8 KB)
Programming range 0x00002000 - 0x00003FFF (8 KB)
Programming range 0x00004000 - 0x00005FFF (8 KB)
Programming range 0x00006000 - 0x00007FFF (8 KB)
Programming range 0x00008000 - 0x00009FFF (8 KB)
Programming range 0x0000A000 - 0x0000BFFF (8 KB)
Programming range 0x0000C000 - 0x0000DFFF (8 KB)
DONE
```

### 4.4.4 Run

Allow the SE Firmware upgrade application to run for at least two seconds. After two seconds, the SE Firmware should have been upgraded.

### 4.4.5 Re-Check Version

Run the `security status` command again to check the upgraded SE Firmware version.

```
commander security status --device EFR32MG21A010F1024 --serialno 440048205
```

```
SE Firmware version : 1.2.14
Serial number       : 000000000000000014b457fffe045a8e
Debug lock          : Disabled
Device erase        : Enabled
Secure debug unlock : Disabled
Tamper status       : OK
Secure boot         : Disabled
Boot status         : 0x20 - OK
DONE
```

The version is now upgraded to 1.2.14.

## 5. Bootloader Firmware Programming

If Secure Boot is enabled, a **SIGNED** version of the bootloader firmware must be programmed to the flash.

Instructions on how to sign the bootloader firmware can be found in sections "Signing for ECDSA-P256-SHA256 Secure Boot" and "Signing for Certificate-Based Secure Boot" in AN1218: Series 2 Secure Boot with RTSL.

For Series 2 devices, the bootloader starting address is device-dependent. For more information about the bootloader starting address, see section "Memory Space For Bootloading" in UG103.6: Bootloader Fundamentals.

Flashing the bootloader firmware using Simplicity Commander is similar to flashing the SE Firmware upgrade application.

```
commander flash --masserase <bootloader file> --device <device name> --serialno <J-Link serial number>
```

where `<bootloader file>` is the name of the bootloader firmware file.

For the TrustZone-aware bootloader, the `<bootloader file>` is the combined image of Secure and Non-secure bootloaders.

To check the Boot status of the device, run the `security status` command.

```
commander security status --device EFR32MG21A010F1024 --serialno 440048205
```

```
SE Firmware version : 1.2.14
Serial number       : 00000000000000014b457fffe045a8e
Debug lock          : Disabled
Device erase        : Enabled
Secure debug unlock : Disabled
Tamper status       : OK
Secure boot         : Disabled
Boot status         : 0x20 - OK
DONE
```

The Secure Boot process fails if the Boot status is not `0x20 - OK`. It means the bootloader firmware is either unsigned or incorrectly signed. The only way to recover is to flash a correctly-signed image (see section "Recover Devices when Secure Boot Fails" in AN1218: Series 2 Secure Boot with RTSL).

## 6. Application Firmware Programming

If the Secure Boot option is enabled in the bootloader, a **SIGNED** version of the application firmware must be programmed to the flash.

Instructions on how to sign the application firmware can be found in sections "Signing for ECDSA-P256-SHA256 Secure Boot" and "Signing for Certificate-Based Secure Boot" in AN1218: Series 2 Secure Boot with RTSL.

For Series 2 devices, the application firmware starting address is device-dependent. For more information about the application starting address, see section "Memory Space For Bootloading" in UG103.6: Bootloader Fundamentals.

Flashing the application firmware using Simplicity Commander is similar to flashing the SE Firmware upgrade application.

```
commander flash <application file> --device <device name> --serialno <J-Link serial number>
```

where `<application file>` is the name of the application firmware file.

For the TrustZone-aware application, the `<application file>` is the combined image of Secure and Non-secure applications.

**Note:** Do not use the `--masserase` option to flash the application firmware since it will erase the bootloader at the starting address.

# 7. Key Provisioning

## 7.1 Overview

The symmetric GBL Decryption Key is used to decrypt GBL files. All encrypted images on this device must be encrypted with the same 128-bit AES key. 7.2 Provisioning the GBL Decryption Key in Simplicity Commander describes different ways to program the GBL Decryption Key to the Series 2 devices.

If the Secure Boot feature is to be used, the Public Sign Key must be provisioned to the device.

If the Secure Debug feature is to be used, the Public Command Key must be provisioned to the device.

The GBL Decryption Key (HSE device), Public Sign Key, and the Public Command Key are written to one-time-programmable (OTP) memory. Once written, they cannot be changed.

**Note:** Silicon Labs strongly recommends provisioning these keys for future-proofing even if the device does not use the GBL Encryption, Secure Boot, and Secure Debug features.

## 7.2 Provisioning the GBL Decryption Key in Simplicity Commander

To generate the text file for the GBL Decryption Key, run the command

```
commander util genkey --type aes-ccm --outfile aes_key.txt
```

```
Using Windows' Cryptographic random number generator
DONE
```

where `aes_key.txt` contains the randomly generated AES-128 key.

Use the text editor to replace the randomly generated key in `aes_key.txt` with the desired GBL Decryption Key as below.

```
# Key randomly generated by 'util genkey'
TOKEN_MFG_SECURE_BOOTLOADER_KEY: 81A5E21FA15286F1DF445C2CC120FA3F
```

To write the GBL Decryption Key to the HSE device, run the command

```
commander security writekey --decrypt aes_key.txt --device EFR32MG21A010F1024 --serialno 440048205
```

This command can be executed only once per device.

```
Device has serial number 000000000000000014b457fffe045a8e

================================================================================
Please look through any warnings before proceeding.
THIS IS A ONE-TIME command, any encrypting of GBL files must be done with this key.
Type 'continue' and hit enter to proceed or Ctrl-C to abort:
================================================================================
continue
DONE
```

**Note:** The GBL Decryption Key cannot be read back from the HSE OTP.

To write the GBL Decryption Key to the `Application Properties Strut` of the GBL, run the command

```
commander convert bootloader-uart-xmodem.s37 --aeskey aes_key.txt --outfile bootloader-uart-xmodem.s37
```

```
Parsing file bootloader-uart-xmodem.s37...
Writing to bootloader-uart-xmodem.s37...
Overwriting file: bootloader-uart-xmodem.s37...
DONE
```

where `bootloader-uart-xmodem.s37` is the GBL image file.

**Note:**

- The `--aeskey` option for the `convert` command requires Simplicity Commander **v1.12.3** or above.
- The GBL Decryption Key can only be added to the GBL with `Application Properties Struct` **v1.2** or higher (GSDK ≥ **v4.1.0**).
- This procedure must be implemented before signing the GBL image for Secure Boot.

To write the GBL Decryption Key to the top page of the main flash of the Series 2 device, run the command

```
commander flash --tokengroup znet --tokenfile aes_key.txt --device EFR32MG22C224F512 --serialno 440048205
```

```
Writing 8192 bytes starting at address 0x0007e000
Comparing range 0x0007E000 - 0x0007FFFF (8 KB)
Erasing range 0x0007E000 - 0x0007FFFF (1 sector, 8 KB)
Programming range 0x0007E000 - 0x0007FFFF (8 KB)
DONE
```

**Note:** The MCU Series 2 devices (like EFM32PG22C200F512IM40) require Simplicity Commander Version 1.12.2 or above to support the `flash --tokengroup znet` command.

### 7.3 Provisioning the Public Sign Key in Simplicity Commander

To write the Public Sign Key to the device, run the command

```
commander security writekey --sign sign_pubkey.pem --device EFR32MG21A010F1024 --serialno 440048205
```

where `sign_pubkey.pem` is the Public Sign Key in Privacy Enhanced Mail (PEM) format. This command can be executed only once per device.

```
Device has serial number 00000000000000014b457fffe045a8e

================================================================================
Please look through any warnings before proceeding.
THIS IS A ONE-TIME command, all code to be run on the device must be signed by this key.
Type 'continue' and hit enter to proceed or Ctrl-C to abort:
================================================================================
continue
DONE
```

To read the Public Sign Key on the device, run the command

```
commander security readkey --sign --device EFR32MG21A010F1024 --serialno 440048205
```

```
C4AF4AC69AAB9512DB50F7A26AE5B4801183D85417E729A56DA974F4E08A562C
DE6019DEA9411332DC1A743372D170B436238A34597C410EA177024DE20FC819
DONE
```

To generate the Public Sign Key token file for the VSE device, run the command

```
commander util keytotoken sign_pubkey.pem --outfile sign_pubkey.txt
```

```
Writing EC tokens to sign_pubkey.txt...
DONE
```

To store a Public Sign Key copy on the top page of the main flash in the VSE device for ECDSA-P256-SHA256 Secure Boot, run the command

```
commander flash --tokengroup znet --tokenfile sign_pubkey.txt --device EFR32MG22C224F512 --serialno 440048205
```

```
Writing 8192 bytes starting at address 0x0007e000
Comparing range 0x0007E000 - 0x0007FFFF (8 KB)
Erasing range 0x0007E000 - 0x0007FFFF (1 sector, 8 KB)
Programming range 0x0007E000 - 0x0007FFFF (8 KB)
DONE
```

**Note:** The MCU Series 2 VSE devices (like EFM32PG22C200F512IM40) require Simplicity Commander Version 1.12.2 or above to support the `flash --tokengroup znet` command.

### 7.4 Provisioning the Public Command Key in Simplicity Commander

To write the Public Command Key to the device, run the command

```
commander security writekey --command command_pubkey.pem --device EFR32MG21A010F1024 --serialno 440048205
```

where `command_pubkey.pem` is the Public Command Key in PEM format. This command can be executed only once per device.

```
Device has serial number 000000000000000014b457fffe045a8e

================================================================================
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
```

To read the Public Command Key on the device, run the command

```
commander security readkey --command --device EFR32MG21A010F1024 --serialno 440048205
```

```
B1BC6F6FA56640ED522B2EE0F5B3CF7E5D48F60BE8148F0DC08440F0A4E1DCA4
7C04119ED6A1BE31B7707E5F9D001A659A051003E95E1B936F05C37EA793AD63
DONE
```

# 8. Enabling Secure Boot and Tamper Configuration

The Secure Boot feature verifies the integrity and authenticity of the host application before allowing it to execute. Enabling this feature is **IRREVERSIBLE**, which means once enabled, Secure Boot can no longer be disabled throughout the life of the device. The Secure Boot settings are written to the one-time-programmable (OTP) memory. They cannot be changed once programmed.

On HSE-SVH devices, the anti-tamper configuration is provisioned with Secure Boot settings. The anti-tamper configuration determines the response from the HSE-SVH device if a tamper event occurs.

**Note:**

- All tamper-related information in the following sections is only valid on HSE-SVH devices.
- For more information about anti-tamper configuration, see AN1247: Anti-Tamper Protection Configuration and Use.
- Except for the EFR32xG21B devices, other HSE-SVH devices require Simplicity Commander Version 1.12.2 or above for tamper configuration.

The `user_configuration.json` is a `JSON` file that contains the desired Secure Boot settings and anti-tamper configuration. Use the following command on the target device (e.g., EFR32MG21B010F1024) to generate a default configuration file.

```
commander security genconfig --nostore -o user_configuration.json --device EFR32MG21B010F1024
--serialno 440048205
```

```
DONE
```

**Note:** The content of the `JSON` file is device-dependent (`--device <device name>`).

The `security genconfig` command above generates a generic configuration file for **EFR32MG21B010F1024** consisting of the properties listed in Table 8.1 Secure Boot Items (mcu_flags) for Series 2 Devices on page 22 and Table 8.2 Tamper Items for HSE-SVH Devices on page 22. A text editor can be used to modify the default settings shown below to the desired configuration.

```
{
   "mcu_flags": {
       "SECURE_BOOT_ENABLE": true,
       "SECURE_BOOT_VERIFY_CERTIFICATE": false,
       "SECURE_BOOT_ANTI_ROLLBACK": true,
       "SECURE_BOOT_PAGE_LOCK_NARROW": false,
       "SECURE_BOOT_PAGE_LOCK_FULL": true
   },
   "tamper_levels": {
        "FILTER_COUNTER": 0,
        "WATCHDOG": 4,
        "SE_RAM_CRC": 4,
        "SE_HARDFAULT": 4,
        "SOFTWARE_ASSERTION": 4,
        "SE_CODE_AUTH": 4,
        "USER_CODE_AUTH": 0,
        "MAILBOX_AUTH": 0,
        "DCI_AUTH": 0,
        "OTP_READ": 4,
        "SELF_TEST": 4,
        "TRNG_MONITOR": 0,
        "PRS0": 0,
        "PRS1": 0,
        "PRS2": 0,
        "PRS3": 0,
        "PRS4": 0,
        "PRS5": 0,
        "PRS6": 0,
        "PRS7": 0,
        "DECOUPLE_BOD": 4,
        "TEMP_SENSOR": 0,
        "VGLITCH_FALLING": 0,
        "VGLITCH_RISING": 0,
        "SECURE_LOCK": 4,
        "SE_DEBUG": 0,
        "DGLITCH": 0,
        "SE_ICACHE": 4
   },
   "tamper_filter" : {
        "FILTER_PERIOD": 0,
        "FILTER_THRESHOLD": 0,
        "RESET_THRESHOLD": 0
   },
   "tamper_flags": {
        "DGLITCH_ALWAYS_ON": false
   }
}
```

**Note:** For `USER_CODE_AUTH` (user secure boot failed), recommends setting is 0 (Ignore) to avoid boot loops.

**Table 8.1. Secure Boot Items (mcu_flags) for Series 2 Devices**

| Name | Description |
|---|---|
| SECURE_BOOT_ENABLE | If set, verifies the host image on the Cortex-M33 before releasing the Cortex-M33 from reset. |
| SECURE_BOOT_VERIFY_CERTIFICATE | If set, requires certificate-based signing of the host image. |
| SECURE_BOOT_ANTI_ROLLBACK | If set, prevents secure upgrading to a host image with a lower version than the image that is currently stored in flash. |
| SECURE_BOOT_PAGE_LOCK_NARROW | If set, locks flash pages that have been validated by the Secure Boot process to prevent re-flashing by other means than through the SE. Write/erase locks pages from 0 through the page where the Secure Boot host image signature is located, not including the last page if the signature is not on a page boundary. |
| SECURE_BOOT_PAGE_LOCK_FULL | If set, locks flash pages that have been validated by the Secure Boot process to prevent re-flashing by other means than through the SE. Write/erase locks pages from 0 through the page where the Secure Boot host image signature is located, including the last page if the signature is not on a page boundary. |

**Note:** The host image is the firmware in the device's flash starting address. It is usually the Gecko Bootloader (GBL).

**Table 8.2. Tamper Items for HSE-SVH Devices**

| Name | Description |
|---|---|
| tamper_levels | The tamper levels of different tamper sources. |
| tamper_filter | The settings for tamper filters. |
| tamper_flags | The settings for tamper flags. |

The following command writes the Secure Boot settings and anti-tamper configuration in `user_configuration.json` file to the device. This command can be executed only once per device.

```
commander security writeconfig --configfile user_configuration.json --device EFR32MG21B010F1024
--serialno 440048205
```

```
================================================================================

THIS IS A ONE-TIME configuration: Please inspect file before confirming:
user_configuration.json
Type 'continue' and hit enter to proceed or Ctrl-C to abort:
================================================================================
continue
DONE
```

To check the device's Secure Boot settings and anti-tamper configuration, run the `security readconfig` command.

```
commander security readconfig --serialno 440048205
```

```
MCU Flags
Secure Boot                    : Enabled
Secure Boot Verify Certificate : Disabled
Secure Boot Anti Rollback      : Enabled
Secure Boot Page Lock Narrow   : Disabled
Secure Boot Page Lock Full     : Enabled

Tamper Levels
FILTER_COUNTER     : 1
WATCHDOG           : 4
SE_RAM_CRC         : 4
SE_HARDFAULT       : 4
SOFTWARE_ASSERTION : 4
SE_CODE_AUTH       : 4
USER_CODE_AUTH     : 0
MAILBOX_AUTH       : 1
DCI_AUTH           : 0
OTP_READ           : 4
SELF_TEST          : 4
TRNG_MONITOR       : 1
PRS0               : 1
PRS1               : 1
PRS2               : 2
PRS3               : 2
PRS4               : 4
PRS5               : 4
PRS6               : 7
PRS7               : 7
DECOUPLE_BOD       : 4
TEMP_SENSOR        : 2
VGLITCH_FALLING    : 2
VGLITCH_RISING     : 2
SECURE_LOCK        : 4
SE_DEBUG           : 0
DGLITCH            : 2
SE_ICACHE          : 4

Tamper Filter
Filter Period   : 10
Filter Threshold : 6
Reset Threshold  : 5

Tamper Flags
Digital Glitch Detector Always On: Disabled
DONE
```

# 9. Enabling Debug Lock

The debug lock is an important feature to prevent attackers from using the debug interface to perform illegal operations on the device. The following sections describe how to apply three different locks to the Series 2 debug interface.

## 9.1 Standard Debug Lock

The following command locks the debug interface.

```
commander security lock --device EFR32MG21A010F1024 --serialno 440048205
```

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

To check the debug lock status of the device, run the `security status` command

```
commander security status --device EFR32MG21A010F1024 --serialno 440048205
```

```
SE Firmware version : 1.2.14
Serial number       : 00000000000000014b457fffe045a8e
Debug lock          : Enabled
Device erase        : Enabled
Secure debug unlock : Disabled
Tamper status       : OK
Secure boot         : Disabled
Boot status         : 0x20 - OK
DONE
```

## 9.2 Permanent Debug Lock

The following command locks the debug interface.

```
commander security lock --device EFR32MG21A010F1024 --serialno 440048205
```

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

After locking the device, disable the device erase using the following command. This is an **IRREVERSIBLE** action and should be the last step in production.

```
commander security disabledeviceerase --device EFR32MG21A010F1024 --serialno 440048205
```

```
================================================================================
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
```

To check the debug lock status of the device, run the `security status` command.

```
commander security status --device EFR32MG21A010F1024 --serialno 440048205
```

```
SE Firmware version : 1.2.14
Serial number       : 00000000000000014b457fffe045a8e
Debug lock          : Enabled
Device erase        : Disabled
Secure debug unlock : Disabled
Secure boot         : Disabled
Boot status         : 0x20 - OK
DONE
```

### 9.3 Secure Debug Lock

The Secure Debug feature is enabled through the `security lockconfig` command. After locking the device, the `security unlock` command securely unlocks the device for debugging until the next device reset without erasing flash and RAM contents. For more information about Secure Debug Unlock, see AN1190: Series 2 Secure Debug.

The following command enables the secure debug unlock.

```
commander security lockconfig --secure-debug-unlock enable --device EFR32MG21A010F1024 --serialno 440048205
```

```
Secure debug unlock was enabled
DONE
```

For the **TrustZone-unaware** application, after enabling the Secure Debug feature, lock the debug interface using the following command.

```
commander security lock --device EFR32MG21A010F1024 --serialno 440048205
```

```
Device is now locked.
DONE
```

For the **TrustZone-aware** application, after enabling the Secure Debug feature, set the debug options (e.g., `1100`) and lock the debug interface using the following command.

```
commander security lock --trustzone 1100 --device EFR32MG21A010F1024 --serialno 440048205
```

```
Writing debug restriction bits:
DBGLOCK:   0
NIDLOCK:   0
SPIDLOCK:  1
SPNIDLOCK: 1
Device is now locked.
DONE
```

**Note:**

- The `--trustzone` option for the `security lock` command requires Simplicity Commander **≥ v1.13.3**.
- It is strongly recommended to upgrade to SE firmware **≥ v1.2.14** (xG21 and xG22) or **≥ v2.2.1** (other Series 2 devices) so that the debug options cannot be modified after the device is locked.
- Use `commander security lock` without the `--trustzone ####` option if the default setting of debug options (`0000`) is good enough for a TrustZone-aware application.
- For more information about debug options, see the "TrustZone Debug Authentication" section in AN1190: Series 2 Secure Debug.

After locking the device, disable the device erase using the following command. This is an **IRREVERSIBLE** action and should be the last step in production.

```
commander security disabledeviceerase --device EFR32MG21A010F1024 --serialno 440048205
```

```
================================================================================
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:** The debug options cannot be reset to the default value `0000` (unlock) if the `device erase` option is disabled.

For Simplicity Commander **< v1.13.3**, run the `security status` command to check the debug lock status of the device.

```
commander security status --device EFR32MG21A010F1024 --serialno 440048205
```

```
SE Firmware version : 1.2.14
Serial number       : 00000000000000014b457fffe045a8e
Debug lock          : Enabled
Device erase        : Disabled
Secure debug unlock : Enabled
Tamper status       : OK
Secure boot         : Disabled
Boot status         : 0x20 - OK
DONE
```

For Simplicity Commander **≥ v1.13.3**, run the `security status --trustzone` command to check the full debug lock status of the device.

```
commander security status --trustzone --device EFR32MG21A010F1024 --serialno 440048205
```

```
SE Firmware version : 1.2.14
Serial number       : 00000000000000014b457fffe045a8e
Debug lock          : Enabled
Device erase        : Disabled
Secure debug unlock : Enabled

Debug lock state: Locked

Non-secure, invasive debug lock     (DBGLOCK)  : Unlocked
Non-secure, non-invasive debug lock (NIDLOCK)  : Unlocked
Secure, invasive debug lock         (SPIDLOCK) : Locked
Secure, non-invasive debug lock     (SPNIDLOCK): Locked

Non-secure, invasive debug lock state     (DBGLOCK)  : Unlocked
Non-secure, non-invasive debug lock state (NIDLOCK)  : Unlocked
Secure, invasive debug lock state         (SPIDLOCK) : Locked
Secure, non-invasive debug lock state     (SPNIDLOCK): Locked

Tamper status       : OK
Secure boot         : Disabled
Boot status         : 0x20 - OK
DONE
```

**Note:** For more information about Secure and Non-secure debug locks, see the " TrustZone Debug Authentication" section in AN1190: Series 2 Secure Debug.

# 10. Field Upgrade the SE Firmware

## 10.1 Secure Boot-Disabled Device

Simplicity Commander or Gecko Bootloader can be used to upgrade the SE Firmware on a Secure Boot-disabled device. The following table lists the scenarios of SE Firmware upgrade on the Secure Boot-disabled device.

| Secure Debug | Device Erase | Debug Lock | State | SE Firmware Upgrade |
|---|---|---|---|---|
| Disabled | Enabled | Disabled | Unlock | Simplicity Commander or Gecko Bootloader |
| Disabled | Enabled | Enabled | Standard debug lock | Simplicity Commander or Gecko Bootloader |
| Disabled | Disabled | Enabled | Permanent debug lock | Gecko Bootloader |
| Enabled | Disabled | Enabled | Secure debug lock | Simplicity Commander or Gecko Bootloader |

**Simplicity Commander:**

To flash the SE Firmware upgrade application (e.g., `s2c1_se_fw_upgrade_app_1v2p9.hex`), run

```
commander flash s2c1_se_fw_upgrade_app_1v2p9.hex --device EFR32MG21A010F1024 --serialno 440048205
```

```
Parsing file s2c1_se_fw_upgrade_app_1v2p9.hex...
Writing 49152 bytes starting at address 0x00000000
Comparing range 0x00000000 - 0x0000BFFF (48 KB)
Programming range 0x00000000 - 0x00001FFF (8 KB)
Programming range 0x00002000 - 0x00003FFF (8 KB)
Programming range 0x00004000 - 0x00005FFF (8 KB)
Programming range 0x00006000 - 0x00007FFF (8 KB)
Programming range 0x00008000 - 0x00009FFF (8 KB)
Programming range 0x0000A000 - 0x0000BFFF (8 KB)
DONE
```

The device should be unlocked before upgrading the SE Firmware if the standard or secure debug lock applies. The sections "Standard Debug Lock and Unlock" and "Secure Debug Unlock and Roll Challenge" in AN1190: Series 2 Secure Debug describes how to unlock the device.

**Note:** This method will **OVERWRITE** the bootloader and application firmware on the device. The user should then re-program the bootloader and application firmware after the SE Firmware upgrade.

**Gecko Bootloader:**

Refer to section "Generate a GBL Upgrade Image File" (Secure Engine Upgrade) in AN1218: Series 2 Secure Boot with RTSL for details.

The Gecko Bootloader can still parse the SE GBL upgrade image file and flash its content to the device even if a debug lock applies. The application firmware must be updated through the Gecko Bootloader after the SE Firmware upgrade if the SE GBL upgrade image file storage overwrites the existing application. Refer to section "Gecko Bootloader Operation - Secure Engine Upgrade" in UG266/UG489 for details.

### 10.2 Secure Boot-Enabled Device

Simplicity Commander or Gecko Bootloader can be used to upgrade the SE Firmware on a Secure Boot-enabled device. The following table lists the scenarios of SE Firmware upgrade on the Secure Boot-enabled device.

| Secure Debug | Device Erase | Debug Lock | State | SE Firmware Upgrade |
|---|---|---|---|---|
| Disabled | Enabled | Disabled | Unlock | Simplicity Commander or Gecko Bootloader |
| Disabled | Enabled | Enabled | Standard debug lock | Simplicity Commander or Gecko Bootloader |
| Disabled | Disabled | Enabled | Permanent debug lock | Gecko Bootloader |
| Enabled | Disabled | Enabled | Secure debug lock | Gecko Bootloader |

**Note:** Using Simplicity Commander to upgrade the SE Firmware on a Secure Debug Locked device is not recommended. It causes Secure Boot failure since the SE Firmware upgrade erases the signed host image for Secure Boot. The user may have issues when recovering a Secure Boot failure device with Device Erase disabled. See section "Recover Devices when Secure Boot Fails" in AN1218: Series 2 Secure Boot with RTSL for details.

**Simplicity Commander:**

A signed SE Firmware should be used for the upgrade. To sign the SE Firmware upgrade application (e.g., `s2c1_se_fw_upgrade_app_1v2p9.hex`) on ECDSA-P256-SHA256 Secure Boot device, run

```
commander convert s2c1_se_fw_upgrade_app_1v2p9.hex --secureboot --keyfile sign_key.pem
--outfile s2c1_se_fw_upgrade_app_1v2p9_signed.hex
```

where `sign_key.pem` is the Private Sign Key for Secure Boot.

```
Parsing file s2c1_se_fw_upgrade_app_1v2p9.hex...
Found Application Properties at 0x00000f18
Writing Application Properties signature pointer to point to 0x0000b364
Setting signature type in Application Properties: 0x00000001
Image SHA256: 251d76d8f3a479c11d55b5788e4fad9bc3b87b440f21eccccf7993b729e520e9
R = 9A391F0503DD25C60D28E7B685DEAD0739A13474567B1029C49C094F6F0BC679
S = BB29D28BFF26D8A38E34DA9CC45F6090486FC517D7D2D79C5CF257B0C94A26DA
Writing to s2c1_se_fw_upgrade_app_1v2p9_signed.hex...
DONE
```

To sign the SE Firmware upgrade application (e.g., `s2c1_se_fw_upgrade_app_1v2p9.hex`) on Certificate-based Secure Boot device, run

```
commander convert s2c1_se_fw_upgrade_app_1v2p9.hex --secureboot --certificate bl_cert.bin
--keyfile bl_cert_key.pem --outfile s2c1_se_fw_upgrade_app_1v2p9_signed.hex
```

where `bl_cert.bin` is the bootloader certificate and `bl_cert_key.pem` is the Private Bootloader Key for Certificate-based Secure Boot.

```
Parsing file s2c1_se_fw_upgrade_app_1v2p9.hex...
Writing certificate to location 0x0000b364
Private key matches public key in certificate.
Found Application Properties at 0x00000f18
Writing Application Properties signature pointer to point to 0x0000b3ec
Setting signature type in Application Properties: 0x00000001
Image SHA256: 7474265e2b99d81a69ecb195a5953cd24acd17b978794d6a24dccf2963fc7979
R = DF1D279B7B632A870EE1604637F1FD38ECBDF11F329E8C7A94D5430111279417
S = F3458E5BFC3BF524AA762F95D29F50E3B7F623589B061204CDAD1CAB435ED3E7

Verifying signed image...
Writing to s2c1_se_fw_upgrade_app_1v2p9_signed.hex...
DONE
```

The sections "Signing for ECDSA-P256-SHA256 Secure Boot" and "Signing for Certificate-Based Secure Boot" in AN1218: Series 2 Secure Boot with RTSL provide additional firmware signing information with HSM.

If the SECURE_BOOT_PAGE_LOCK_NARROW or SECURE_BOOT_PAGE_LOCK_FULL in Table 8.1 Secure Boot Items (mcu_flags) for Series 2 Devices on page 22 was enabled for Secure Boot or the standard debug lock applies, run

```
commander security erasedevice --device EFR32MG21A010F1024 --serialno 440048205
```

to perform a device erase. Issue a power-on or pin reset to complete the device erase process.

```
Successfully erased device
DONE
```

To flash the signed SE Firmware upgrade application (s2c1_se_fw_upgrade_app_1v2p9_signed.hex), run

```
commander flash s2c1_se_fw_upgrade_app_1v2p9_signed.hex --device EFR32MG21A010F1024 --serialno 440048205
```

```
Parsing file s2c1_se_fw_upgrade_app_1v2p9_signed.hex...
Writing 49152 bytes starting at address 0x00000000
Comparing range 0x00000000 - 0x0000BFFF (48 KB)
Erasing range 0x00000000 - 0x00007FFF (4 sectors, 32 KB)
Erasing range 0x00008000 - 0x0000BFFF (2 sectors, 16 KB)
Programming range 0x00000000 - 0x00001FFF (8 KB)
Programming range 0x00002000 - 0x00003FFF (8 KB)
Programming range 0x00004000 - 0x00005FFF (8 KB)
Programming range 0x00006000 - 0x00007FFF (8 KB)
Programming range 0x00008000 - 0x00009FFF (8 KB)
Programming range 0x0000A000 - 0x0000BFFF (8 KB)
DONE
```

**Note:**

1. This method will **OVERWRITE** the signed bootloader and application firmware on the device. The user should then re-program the **SIGNED** bootloader and application firmware after the SE Firmware upgrade.
2. If the SECURE_BOOT_ANTI_ROLLBACK in Table 8.1 Secure Boot Items (mcu_flags) for Series 2 Devices on page 22 was enabled for Secure Boot, the device will prevent the signed SE Firmware upgrade when the host image version (e.g., Gecko Bootloader v1.12.0) is equal to or higher than the SE Firmware version (e.g., v1.2.9). Under this situation, the Gecko Bootloader should be used to upgrade the SE firmware. This method will **OVERWRITE** the bootloader version in SE flash with the SE Firmware version after the upgrade.

```
Parsing file s2c1_se_fw_upgrade_app_1v2p9_signed.hex...
Writing 49152 bytes starting at address 0x00000000
Comparing range 0x00000000 - 0x0000BFFF (48 KB)
Erasing range 0x00000000 - 0x00007FFF (4 sectors, 32 KB)
Programming range 0x00000000 - 0x00001FFF (8 KB)
Programming range 0x00002000 - 0x00003FFF (8 KB)
Programming range 0x00004000 - 0x00005FFF (8 KB)
Programming range 0x00006000 - 0x00007FFF (8 KB)
Programming range 0x00008000 - 0x00009FFF (8 KB)
Programming range 0x0000A000 - 0x0000BFFF (8 KB)
JLinkError: Failed to halt CPU.
DONE
```

```
SE Firmware version : 1.2.6
Serial number       : 00000000000000014b457fffe045a20
Debug lock          : Disabled
Device erase        : Enabled
Secure debug unlock : Disabled
Tamper status       : Not OK
Secure boot         : Enabled
Boot status         : 0x10 - Failed: Error occured due to rollback prevention. Device has seen application with higher
version number
DONE
```

**Gecko Bootloader:**

Refer to section "Generate a GBL Upgrade Image File" (Secure Engine Upgrade) in AN1218: Series 2 Secure Boot with RTSL for details.

The Gecko Bootloader can still parse the SE GBL upgrade image file and flash its content to the device even if a debug lock applies. The application firmware must be updated through the Gecko Bootloader after the SE Firmware upgrade if the SE GBL upgrade image file storage overwrites the existing application. Refer to section "Gecko Bootloader Operation - Secure Engine Upgrade" in UG266/UG489 for details.

# 11. Alternatives for Series 2 Programming

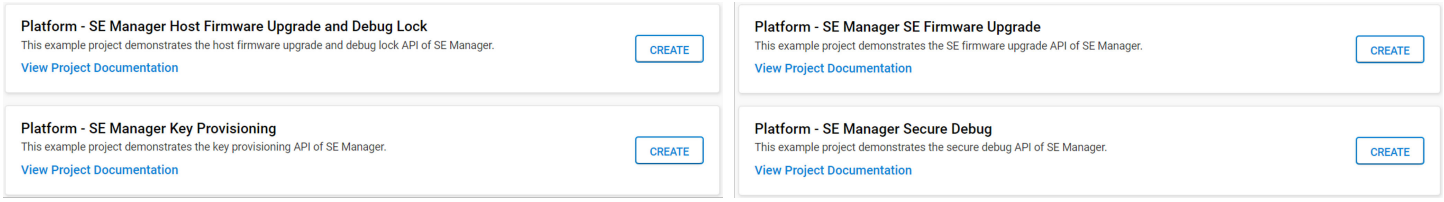
Besides the Simplicity Studio and Simplicity Commander, a mailbox interface from the Cortex-M33 or a dedicated Debug Challenge Interface (DCI) can be used to program the Series 2 devices.

## 11.1 Mailbox Interface

The SE Manager can provision and program Series 2 devices through the Mailbox interface. For more information about the Mailbox interface, see section "Command Interface - Mailbox" in AN1190: Series 2 Secure Debug.

Simplicity Studio 5 includes the SE Manager platform examples for Series 2 devices programming and provisioning as described in the following table. The Secure Debug platform example can only run on the HSE device.

| SE Manager Platform Example | Usage |
|---|---|
| SE Manager Host Firmware Upgrade and Debug Lock | Upgrade the host (Cortex-M33) firmware and enable debug lock. |
| SE Manager Key Provisioning | Key provisioning, enabling secure boot and tamper configuration. |
| SE Manager SE Firmware Upgrade | Upgrade the SE Firmware. |
| SE Manager Secure Debug (HSE only) | Unlock the device, enable secure debug, and disable device erase. |

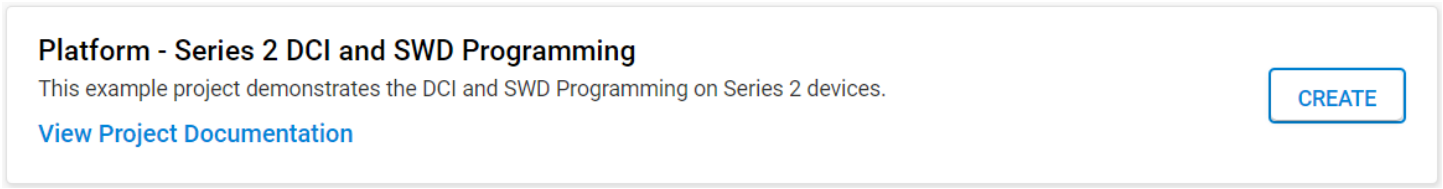



Refer to the corresponding `readme` file for details about each SE Manager platform example. This file also includes the procedures to create the project and run the example. Click the `View Project Documentation` link to open the `readme` file.

## 11.2 Debug Challenge Interface (DCI)

Simplicity Studio 5 includes an SE Manager platform example (using BRD4182A radio board) to use GPIO to emulate the Serial Wire Debug (SWD) interface to provision and program Series 2 devices through the dedicated Debug Challenge Interface (DCI).

Refer to the corresponding `readme` file for details about this platform example. This file also includes the procedures to create the project and run the example. Click the `View Project Documentation` link to open the `readme` file.




For more information about DCI and SWD programming, see AN1303: Programming Series 2 Devices using the Debug Challenge Interface (DCI) and Serial Wire Debug (SWD).

## 12. Related Documents

- AN136: Silicon Labs Production Programming Options
- AN958: Debugging and Programming Interfaces for Custom Designs
- UG103.6: Bootloader Fundamentals
- UG162: Simplicity Commander Reference Guide
- UG266: Silicon Labs Gecko Bootloader User's Guide for GSDK 3.2 and Lower
- UG489: Silicon Labs Gecko Bootloader User's Guide for GSDK 4.0 and Higher
- AN1190: Series 2 Secure Debug
- AN1218: Series 2 Secure Boot with RTSL
- AN1247: Anti-Tamper Protection Configuration and Use
- AN1303: Programming Series 2 Devices using the Debug Challenge Interface (DCI) and Serial Wire Debug (SWD)

# 13. Revision History

### Revision 0.9

February 2023

- Updated figures and content (replace optional Enable Secure Debug with strongly recommended Enable Debug Lock) in 2.1 Production Programming for Secure Boot-Disabled Device and 2.2 Production Programming for Secure Boot-Enabled Device.
- Updated 3. Using Simplicity Commander to v1.14.2.
- Fixed a typo (`.sec` to `.seu`) in 4.3 How to Find the Latest SE Firmware.
- Updated 5. Bootloader Firmware Programming for TrustZone-aware bootloader.
- Updated 6. Application Firmware Programming for TrustZone-aware application.
- Updated 7.2 Provisioning the GBL Decryption Key in Simplicity Commander for the GBL Decryption Key in the `Application Properties Strut` of the GBL.
- Updated 8. Enabling Secure Boot and Tamper Configuration for Simplicity Commander v1.14.2.
- Added 9. Enabling Debug Lock (change from Enabling Secure Debug), 9.1 Standard Debug Lock, and 9.2 Permanent Debug Lock.
- Updated 9.3 Secure Debug Lock for TrustZone-unaware and TrustZone-aware applications.

### Revision 0.8

June 2022

- Updated table and note in 1. Series 2 Device Security Features.
- Replaced Device Compatibility with SE Firmware in 1. Series 2 Device Security Features.
- Removed Table 4.1 in 4.2 How to Check the SE Firmware Version on a Device (this table is moved to UG103.05).

### Revision 0.7

March 2022

- Added digit 4 to Note 3 in 1. Series 2 Device Security Features.
- Updated Device Compatibility and moved it under 1. Series 2 Device Security Features.

### Revision 0.6

January 2022

- Added CPMS information to 2. Overview.
- Updated Shipped SE Firmware Version in Table 4.1.
- Updated 4.3 How to Find the Latest SE Firmware with Windows folder for GSDK v4.0 and higher.
- Added note to 7.2 Provisioning the GBL Decryption Key in Simplicity Commander for MCU Series 2 VSE devices.
- Added note to 7.3 Provisioning the Public Sign Key in Simplicity Commander for MCU Series 2 VSE devices.
- Updated note for Simplicity Commander support of tamper configuration on HSE-SVH devices in 8. Enabling Secure Boot and Tamper Configuration
- Updated Table 8.1 Secure Boot Items (mcu_flags) for Series 2 Devices on page 22.
- Updated 10. Field Upgrade the SE Firmware.
- Added UG489 to the table in 1.2 Key Reference and 12. Related Documents.

**Revision 0.5**

September 2021

- Formatting updates for source compatibility.
- Added revised terminology to 1. Series 2 Device Security Features and use this terminology throughout the document.
- Updated Device Compatibility.
- Removed terminology in 2. Overview.
- Updated Simplicity Commander version to 1.11.2 in 3. Using Simplicity Commander.
- Added security notification figure to 4.1 Overview.
- Updated SE Firmware version in Table 4.1.
- Updated the figures in 4.2.1 Check the SE Firmware Version Using Simplicity Studio 5.
- Updated 4.3 How to Find the Latest SE Firmware.
- Updated 7.2 Provisioning the GBL Decryption Key in Simplicity Commander.
- Updated 7.3 Provisioning the Public Sign Key in Simplicity Commander
- Renamed Enabling Secure Boot to 8. Enabling Secure Boot and Tamper Configuration, updated the content.
- Updated 10. Field Upgrade the SE Firmware.
- Added 11. Alternatives for Series 2 Programming.
- Updated 12. Related Documents.

**Revision 0.4**

October 2020

- Removed a duplicate paragraph from 1.1 User Assistance.

**Revision 0.3**

September 2020

- Added EFR32BG21B and EFR32MG21B to Device Compatibility.
- Added SE conventions to 2. Overview, updated the figures and content.
- Updated Simplicity Commander version to 1.9.2 in 3. Using Simplicity Commander.
- Added EFRxG21B to 4.2 How to Check the SE Firmware Version on a Device.
- Updated the figures in 4.2.1 Check the SE Firmware Version Using Simplicity Studio 5 to Simplicity Studio v5.
- Renamed Field Upgrade to Field Upgrade the Secure Element Firmware on a Secure Boot-Enabled Device, updated the content and moved up two levels.
- Removed Over-The-Air (OTA) section.
- Added 5. Bootloader Firmware Programming.
- Updated 6. Application Firmware Programming.
- Updated Enabling Secure Boot for SE with Secure Vault devices.
- Added 7.2 Provisioning the GBL Decryption Key in Simplicity Commander.
- Added AN1247 and AN1271 to 12. Related Documents.

**Revision 0.2**

March 2020

- Added EFR32xG22 devices to Device Compatibility section.
- Added Simplicity Commander section.
- Updated figures in Check SE Firmware Version Using Simplicity Studio
- Modified Check SE Firmware Version Using Simplicity Studio section.
- Added Note to Check Version section.
- Added Field Upgrade to Serial Wire Debug (SWD) section.
- Added disable device erase procedure to Secure Debug Enabling section.
- Added UG162 to Related Documents section.
- Changed all Simplicity Commander outputs to text, easy to update in the future.

**Revision 0.1**

December 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 & Community**
www.silabs.com/community



SILICON LABS

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

**www.silabs.com**