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)	EFR32xG2yB (3)
Secure Vault Mid (SVM)	HSE (HSE-SVM)	EFR32xG2yA (3)
"	VSE (VSE-SVM)	EFR32xG2y, EFM32PG2y (4)
Secure Vault Base (SVB)	N/A	MCU Series 1 and Wireless SoC Series 1

Note:

1. The features of different Secure Vault levels can be found in https://www.silabs.com/security.
2. The x is a letter (B, F, M, or Z).
3. At the time of this writing, the y is a digit (1 or 3).
4. At the time of this writing, the y is a digit (2).

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
Attestation key aka Private Device Key	No	Device authentication for secure identity	AN1268

2. Device Compatibility

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

MCU Series 2 consists of:

- EFM32PG22 (VSE-SVM)

Wireless SoC Series 2 families consist of:

- EFR32BG21A/EFR32MG21A (HSE-SVM)
- EFR32BG21B/EFR32MG21B (HSE-SVH)
- EFR32BG22/EFR32FG22/EFR32MG22 (VSE-SVM)
- EFR32FG23A/EFR32ZG23A (HSE-SVM)
- EFR32FG23B/EFR32ZG23B (HSE-SVH)

3. 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 an optional 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.

3.1 Production Programming for Secure Boot-Disabled Device

The production programming flow for Secure Boot-disabled devices is illustrated in the following figure.

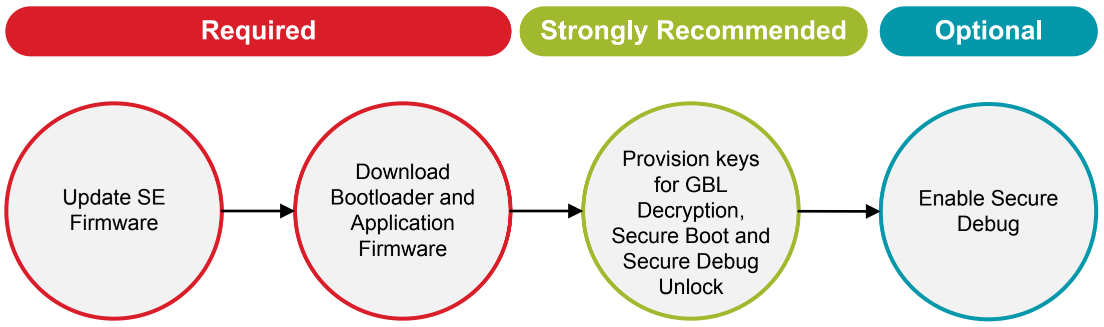

Required
Strongly Recommended
Optional
Update SE Firmware
Download Bootloader and Application Firmware
Provision keys for GBL Decryption, Secure Boot and Secure Debug Unlock
Enable Secure Debug

Figure 3.1. 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 steps in the production programming process. Provisioning the GBL Decryption Key for GBL payload decryption, Public Sign Key for Secure Boot, and Public Command Key for Secure Debug Unlock are strongly recommended. Enabling the Secure Debug feature is optional if the Public Command Key was provisioned.

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

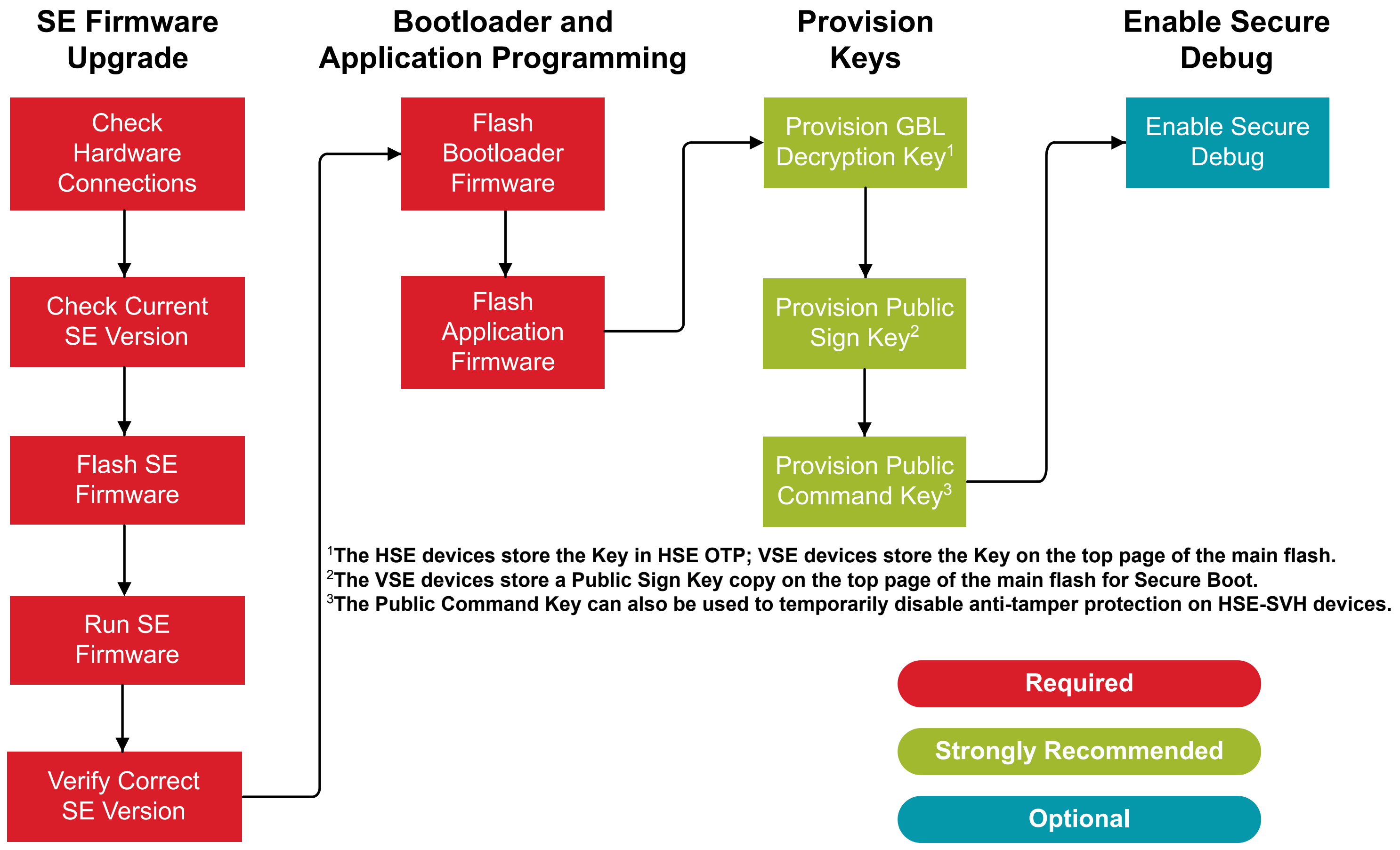

SE Firmware Upgrade
Bootloader and Application Programming
Provision Keys
Enable Secure Debug
Check Hardware Connections
Check Current SE Version
Flash SE Firmware
Run SE Firmware
Verify Correct SE Version
Flash Bootloader Firmware
Flash Application Firmware
Provision GBL Decryption Key[1]
Provision Public Sign Key[2]
Provision Public Command Key[3]
Enable Secure Debug
[1]The HSE devices store the Key in HSE OTP; VSE devices store the Key on the top page of the main flash.
[2]The VSE devices store a Public Sign Key copy on the top page of the main flash for Secure Boot.
[3]The Public Command Key can also be used to temporarily disable anti-tamper protection on HSE-SVH devices.
Required
Strongly Recommended
Optional

Figure 3.2. Series 2 Step-by-Step Production Programming Flowchart for a Secure Boot-Disabled Device

3.2 Production Programming for Secure Boot-Enabled Device

The production programming flow for Secure Boot-enabled devices is illustrated in the following figure.

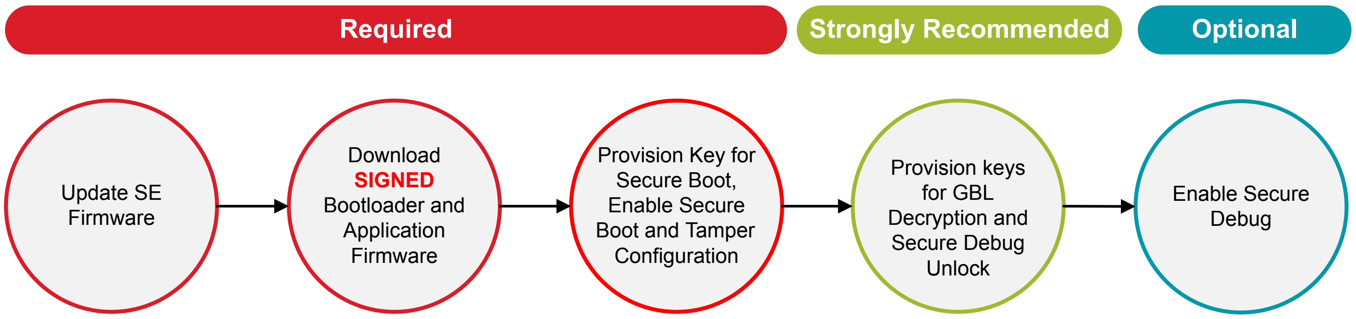

Required
Strongly Recommended
Optional
Update SE Firmware
Download SIGNED Bootloader and Application Firmware
Provision Key for Secure Boot, Enable Secure Boot and Tamper Configuration
Provision keys for GBL Decryption and Secure Debug Unlock
Enable Secure Debug

Figure 3.3. 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 steps in the production programming process. Provisioning the Public Sign Key and enabling Secure Boot and Tamper Configuration (HSE-SVH only) are also required in the production programming process to enable the Secure Boot option. Provisioning the GBL Decryption Key for GBL payload decryption and the Public Command Key for Secure Debug Unlock are strongly recommended. Enabling the Secure Debug feature is optional if the Public Command Key was provisioned.

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

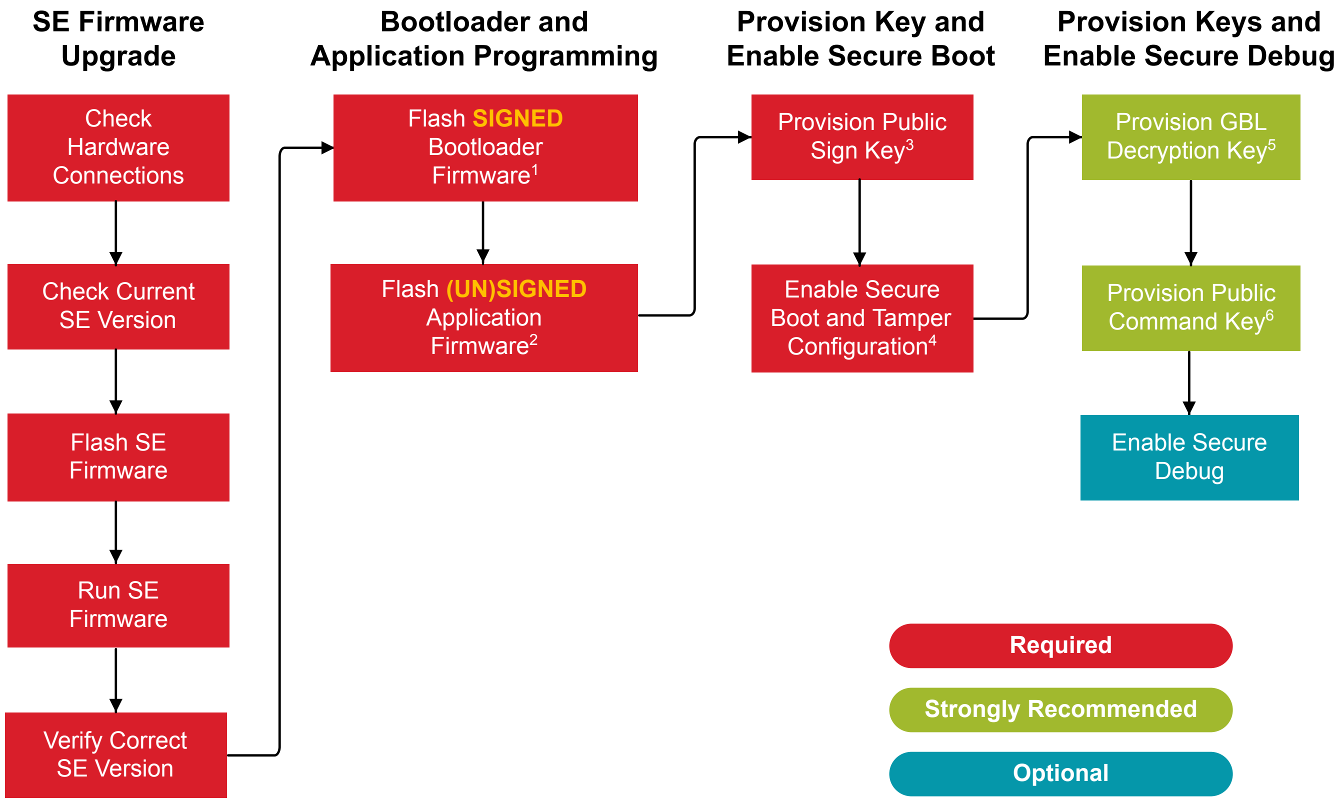

SE Firmware Upgrade
Bootloader and Application Programming
Provision Key and Enable Secure Boot
Provision Keys and Enable Secure Debug
Check Hardware Connections
Check Current SE Version
Flash SE Firmware
Run SE Firmware
Verify Correct SE Version
Flash SIGNED Bootloader Firmware[1]
Flash (UN)SIGNED Application Firmware[2]
Provision Public Sign Key[3]
Enable Secure Boot and Tamper Configuration[4]
Provision GBL Decryption Key[5]
Provision Public Command Key[6]
Enable Secure Debug
Required
Strongly Recommended
Optional

Figure 3.4. 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 6. Bootloader Firmware Programming).
2. If the Secure Boot option is enabled in the bootloader, the application firmware must be signed (see 7. 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 sections "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 9. Enabling Secure Boot and Tamper Configuration).
5. The HSE devices store the GBL Decryption Key in HSE OTP; VSE devices store the GBL Decryption Key on the top page of the main flash.
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).

4. Using Simplicity Commander

1. This application note uses Simplicity Commander v1.11.2. The console output may be different on the other version 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 1v11p2b998

JLink DLL version: 6.94d
Qt 5.12.1 Copyright (C) 2017 The Qt Company Ltd.
EMDLL Version: 0v17p18b581
mbed TLS version: 2.6.1


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.

5. SE Firmware Programming

5.1 Overview

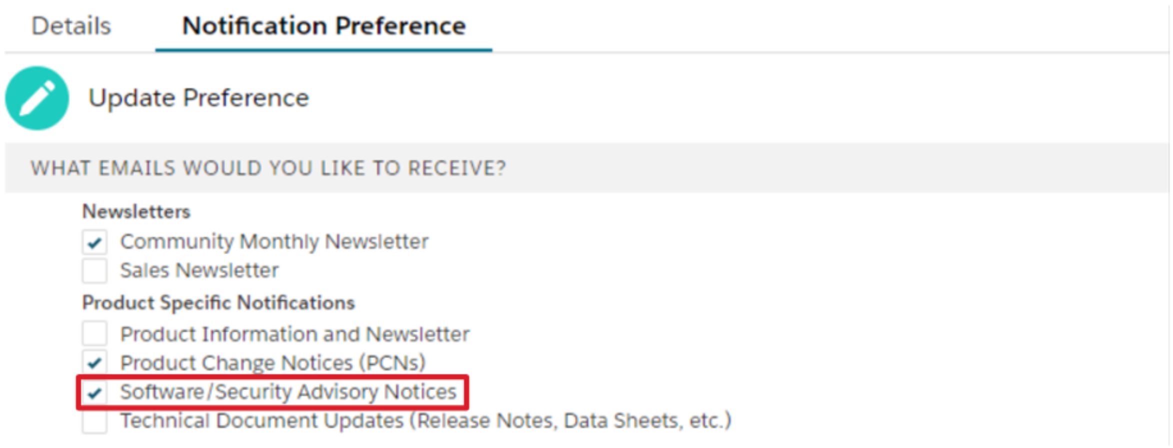
Production programming of Series 2 devices is identical to production programming of Series 1 devices, with the addition of the SE Firmware in the production programming process. Consistent with best practices for Internet of Things (IoT) security, the SE Firmware provided with Series 2 devices supports 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.



Newsletters
- [✓] Community Monthly Newsletter
- [ ] Sales Newsletter

Product Specific Notifications
- [ ] Product Information and Newsletter
- [✓] Product Change Notices (PCNs)
- [✓] Software/Security Advisory Notices
- [ ] Technical Document Updates (Release Notes, Data Sheets, etc.)

- 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.

5.2 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

At the time of this writing, the latest SE Firmware shipped with Series 2 devices and modules (if available) are listed in the following table.

Table 5.1. Latest SE Firmware Version on Shipped Series 2 Devices and Modules

MCU Series 2 and Wireless SoC Series 2	SE	Shipped SE Firmware Version (Device and Module)
EFR32xG21A	HSE-SVM	Version 1.2.6
EFR32xG23A	HSE-SVM	Version 2.1.3
EFR32xG21B	HSE-SVH	Version 1.2.6
EFR32xG23B	HSE-SVH	Version 2.1.3
EFM32PG22 and EFR32xG22	VSE-SVM	Version 1.2.6

5.2.1 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 on the other version 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.

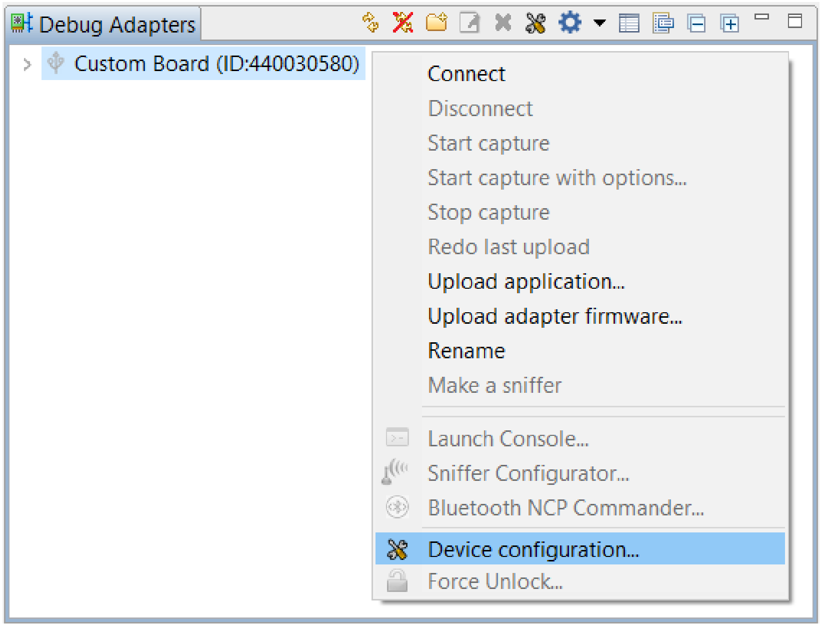


Figure 5.1. Context Menu of Debug Adapters

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.

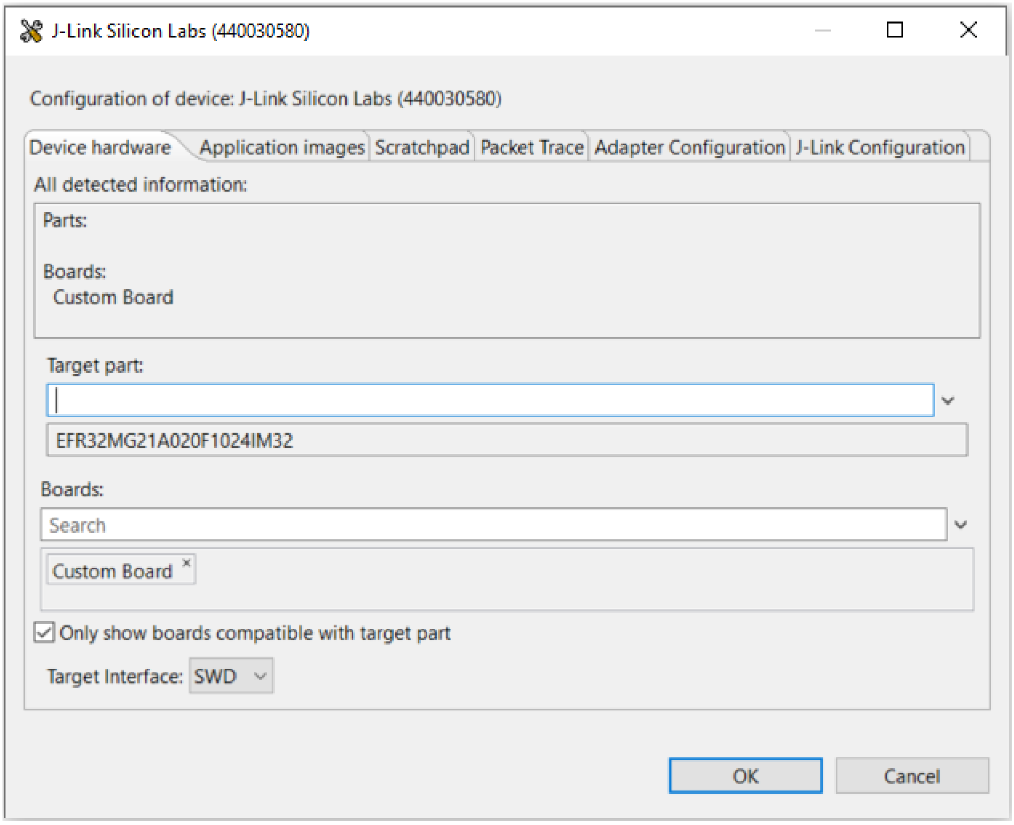


Figure 5.2. Configuration of Device

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

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

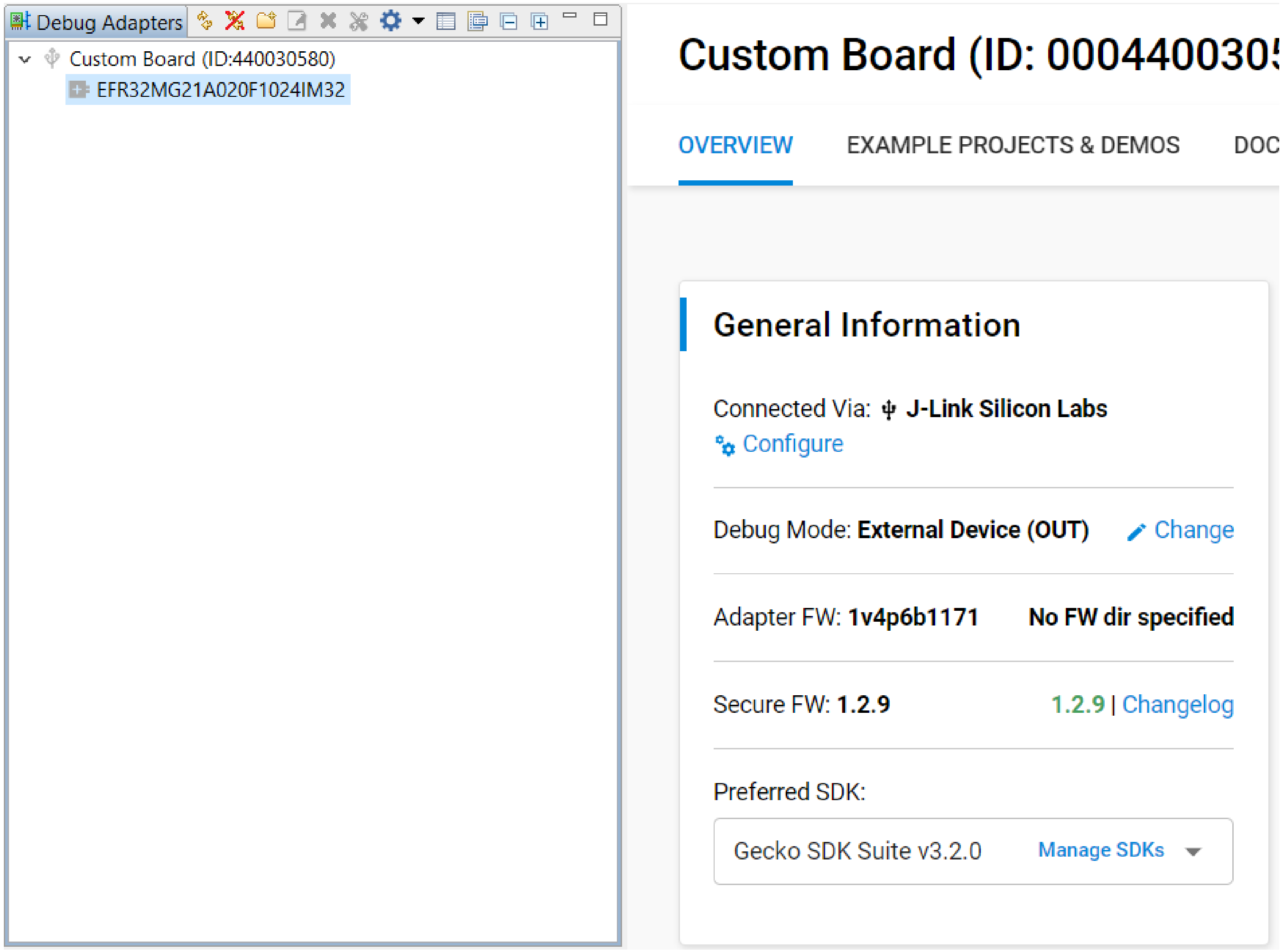


Figure 5.3. SE Firmware Version in Simplicity Studio

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.

5.2.2 Check the SE Firmware Version Using Simplicity Commander

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

```
commander security status --device EFR32MG22C224F512 --serialno 440030580
```

```
SE Firmware version : 1.2.1
Serial number       : 0000000000000000014b457fffed50d1e
Debug lock          : Disabled
Device erase        : Enabled
Secure debug unlock : Disabled
Secure boot         : Disabled
Boot status         : 0x20 - OK
DONE
```

In this example, the SE Firmware version on the EFR32MG22 is 1.2.1.

5.3 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 (`.sec` and `.hex`) and release notes can be found in the Windows folders below (`<version>` should be `v3.1` or above).
`C:\SiliconLabs\SimplicityStudio\v5\developer\sdks\gecko_sdk_suite\<version>\util\se_release\public`

5.4 Serial Wire Debug (SWD)

The SE Firmware cannot be directly programmed to the SE using the SWD interface. Instead, a loader application that contains and will install the SE Firmware is flashed onto the host MCU. The SE Firmware is encrypted, versioned, and signed.

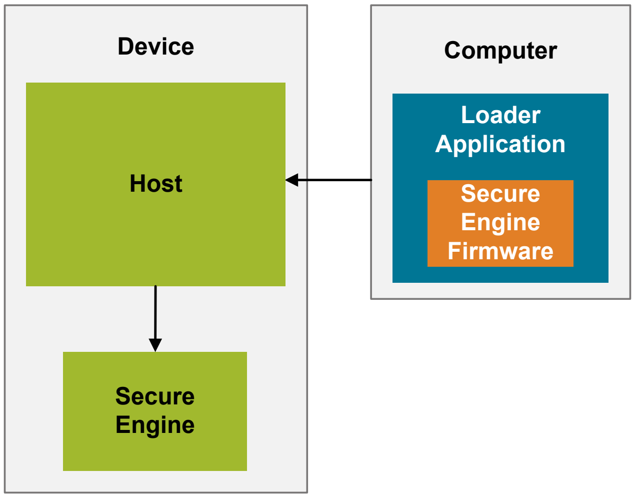


Figure 5.4. SWD SE Firmware Upgrade Block Diagram

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 3.2 Series 2 Step-by-Step Production Programming Flowchart for a Secure Boot-Disabled Device on page 6 or Figure 3.4 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.

5.4.1 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 440030580
```

```
Part Number    : EFR32MG21A010F1024
Die Revision   : A1
Production Ver : 0
Flash Size     : 1024 kB
SRAM Size      : 96 kB
Unique ID      : 000d6ffffead3d94
DONE
```

5.4.2 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 440030580
```

```
SE Firmware version : 1.2.0
Serial number       : 0000000000000000d6ffffead3d94
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.

5.4.3 Flash SE Firmware

To flash the SE Firmware upgrade application, run

```
commander flash --masserase s2c1_se_fw_upgrade_app_1v2p9.hex --device EFR32MG21A010F1024 --serialno 440030580
```

where `s2c1_se_fw_upgrade_app_1v2p9.hex` is replaced with the name of the SE Firmware upgrade application file.

```
Parsing file s2c1_se_fw_upgrade_app_1v2p9.hex...
Erasing chip...
Flash was erased successfully
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
```

5.4.4 Run

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

5.4.5 Re-Check Version

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

```
commander security status --device EFR32MG21A010F1024 --serialno 440030580
```

```
SE Firmware version : 1.2.9
Serial number       : 0000000000000000d6ffffead3d94
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 1.2.9.

6. 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 "ECDSA-P256-SHA256 Secure Boot" and "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.

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

```
commander security status --device EFR32MG21A010F1024 --serialno 440030580
```

```
SE Firmware version : 1.2.9
Serial number       : 0000000000000000d6ffffead3d94
Debug lock          : Disabled
Device erase        : Enabled
Secure debug unlock : Disabled
Tamper status       : OK
Secure boot         : Enabled
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).

7. 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 "ECDSA-P256-SHA256 Secure Boot" and "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.

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

8. Key Provisioning

8.1 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.

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

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

The VSE devices store the GBL Decryption Key on the top page of the main flash instead of OTP.

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.

8.2 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.

```
# Key randomly generated by 'util genkey'
TOKEN_MFG_SECURE_BOOTLOADER_KEY: 056895E0DDDD792B4C4A3EFC3C9FFBC5
```

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 440030580
```

It can execute this command only once per device.

```
Device has serial number 000000000000000000d6ffffead3d94

================================================================================
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: It cannot read back the GBL Decryption Key from the HSE OTP.

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

```
commander flash --tokengroup znet --tokenfile aes_key.txt --device EFR32MG22C224F512 --serialno 440030580
```

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

8.3 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 440030580
```

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

```
Device has serial number 0000000000000000d6ffffead3d94

================================================================================
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 440030580
```

```
C4AF4AC69AAB9512DB50F7A26AE5B4801183D85417E729A56DA974F4E08A562C
DE6019DEA9411332DC1A743372D170B436238A34597C410EA177024DE20FC819
DONE
```

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

```
commander gbl keyconvert sign_pubkey.pem -o 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 440030580
```

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

8.4 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 440030580
```

where `command_pubkey.pem` is the Public Command Key in PEM format. It can execute this command only once per device.

```
Device has serial number 0000000000000000d6ffffead3d94

================================================================================
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 440030580
```

```
B1BC6F6FA56640ED522B2EE0F5B3CF7E5D48F60BE8148F0DC08440F0A4E1DCA4
7C04119ED6A1BE31B7707E5F9D001A659A051003E95E1B936F05C37EA793AD63
DONE
```

9. 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.
- The Simplicity Commander Version 1.11.2 can only support tamper configuration on EFR32xG21B devices.

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

```
commander security writeconfig --configfile user_configuration.json --device EFR32MG21B010F1024
--serialno 440030580
```

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

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
```

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

```
commander security genconfig --nostore -o user_configuration.json --device EFR32MG21B010F1024
--serialno 440030580
```

```
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 9.1 Secure Boot Items (mcu_flags) for Series 2 Devices on page 22 and Table 9.2 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": 4,
        "MAILBOX_AUTH": 0,
        "DCI_AUTH": 0,
        "OTP_READ": 0,
        "AUTO_CODE_AUTH": 0,
        "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": 1,
        "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 9.1. Secure Boot Items (mcu_flags) for Series 2 Devices

Name	Description
SECURE_BOOT_ENABLE	If set, verifies the 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 application.
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 signature of the application 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 signature of the application is located, including the last page if the signature is not on a page boundary.

Table 9.2. 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.

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

```
commander security readconfig --serialno 440030580
```

```
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     : 0
WATCHDOG           : 4
SE_RAM_CRC         : 4
SE_HARDFAULT       : 4
SOFTWARE_ASSERTION : 4
SE_CODE_AUTH       : 4
USER_CODE_AUTH     : 4
MAILBOX_AUTH       : 0
DCI_AUTH           : 0
OTP_READ           : 0
AUTO_CODE_AUTH     : 0
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        : 1
VGLITCH_FALLING    : 0
VGLITCH_RISING     : 0
SECURE_LOCK        : 4
SE_DEBUG           : 0
DGLITCH            : 0
SE_ICACHE          : 4

Tamper Filter
Filter Period   : 0
Filter Treshold : 0
Reset Treshold  : 0

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

10. Enabling Secure Debug

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 EFR32MG22C224F512 --serialno 440030580
```

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

After enabling the Secure Debug feature, you can lock the device by using the following command.

```
commander security lock --device EFR32MG22C224F512 --serialno 440030580
```

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

After locking the device, you can disable the device erase by using the following command.

```
commander security disabledeviceerase --device EFR32MG22C224F512 --serialno 440030580
```

```
================================================================================
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.

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

```
commander security status --device EFR32MG22C224F512 --serialno 440030580
```

```
SE Firmware version : 1.2.7
Serial number       : 0000000000000000014b457fffed50d1e
Debug lock          : Enabled
Device erase        : Disabled
Secure debug unlock : Enabled
Secure boot         : Disabled
Boot status         : 0x20 - OK
DONE
```

11. Field Upgrade the SE Firmware

11.1 Secure Boot-Disabled Device

It can use Simplicity Commander or Gecko Bootloader 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 440030580
```

```
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
```

It should unlock the device 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. It should re-program the bootloader and application firmware after the SE Firmware upgrade.

Gecko Bootloader:

To generate the SE Firmware (e.g. `s2c1_se_fw_upgrade_1v2p9.seu`) GBL upgrade file, run

```
commander gbl create seupgrade.gbl --seupgrade s2c1_se_fw_upgrade_1v2p9.seu
```

```
Initializing GBL file...
Adding Secure Element upgrade image to GBL...
Writing GBL file seupgrade.gbl...
DONE
```

The Gecko Bootloader can still parse the SE Firmware GBL upgrade file and flash its content to the device even the standard or secure debug lock applies. Details on SE Firmware upgrade can be found in section "Gecko Bootloader Operation - Secure Engine Upgrade" in UG266: Silicon Labs Gecko Bootloader User's Guide.

Note: The application firmware must be re-programmed after the SE Firmware upgrade if the GBL file storage overwrites the existing application.

11.2 Secure Boot-Enabled Device

It can use Simplicity Commander or Gecko Bootloader 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

Simplicity Commander:

Note: DO NOT use this method on the "**Secure Debug Unlock**" device. It will usually **BRICK** the device since the SE Firmware upgrade erases the signed host image for Secure Boot and Device Erase was disabled (see section "Precautions - Secure Boot and Debug Lock" in AN1190: Series 2 Secure Debug).

It should use a signed SE Firmware 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 bloader_cert.bin
--keyfile bloadercert_key.pem --outfile s2c1_se_fw_upgrade_app_1v2p9_signed.hex
```

where `bloader_cert.bin` is the bootloader certificate and `bloadercert_key.pem` is the Private Bootloader Key to sign the bootloader certificate.

```
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 UG162: Simplicity Commander Reference Guide (using a Hardware Security Module for signing) and sections "ECDSA-P256-SHA256 Secure Boot" and "Certificate-Based Secure Boot" in AN1218: Series 2 Secure Boot with RTSL provide additional firmware signing information.

If the `SECURE_BOOT_PAGE_LOCK_NARROW` or `SECURE_BOOT_PAGE_LOCK_FULL` in Table 9.1 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 440030580
```

to perform a device erase.

```
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 440030580
```

```
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. It should re-program the **SIGNED** bootloader and application firmware after the SE Firmware upgrade.
2. If the `SECURE_BOOT_ANTI_ROLLBACK` in Table 9.1 Secure Boot Items (mcu_flags) for Series 2 Devices on page 22 was enabled for Secure Boot. It 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, it should use the Gecko Bootloader to upgrade the SE firmware.

   ```
   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:

It should use a signed GBL file for the upgrade. To generate the signed SE Firmware (e.g. `s2c1_se_fw_upgrade_1v2p9.seu`) GBL upgrade file on ECDSA-P256-SHA256 Secure Boot device, run

```
commander gbl create seupgrade.gbl --seupgrade s2c1_se_fw_upgrade_1v2p9.seu --sign sign_key.pem
```

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

```
Initializing GBL file...
Adding Secure Element upgrade image to GBL...
Signing GBL...
Image SHA256: 599d7fc35996b4715441b642709ed262525d09d811d4726e423c0d605ec0f0bf
R = 9F1039C5D21CA0710260870378A06309EB9ACD599D582870C4CD45774EC5050E
S = CFD3786DC2EEABBF4A75F47AB59C5AD8AB90AF842FACC96825AB7107E6592D54
Writing GBL file seupgrade.gbl...
DONE
```

To generate the signed SE Firmware (e.g. `s2c1_se_fw_upgrade_1v2p9.seu`) GBL upgrade file on Certificate-based Secure Boot device, run

```
commander gbl create seupgrade.gbl --seupgrade s2c1_se_fw_upgrade_1v2p9.seu --sign bloadercert_key.pem
```

where `bloadercert_key.pem` is the Private Bootloader Key to sign the bootloader certificate.

```
Initializing GBL file...
Adding Secure Element upgrade image to GBL...
Signing GBL...
Image SHA256: 599d7fc35996b4715441b642709ed262525d09d811d4726e423c0d605ec0f0bf
R = EF8EC2DDEDDF44DF88FEAD4ED0A9FDC6351B4D745D5A05BFB87204791871A525
S = FCB26EF005D97E8C5341153A210AE9927E1CF646A3E473FFB90DA8C857E6421F
Writing GBL file seupgrade.gbl...
DONE
```

The UG162: Simplicity Commander Reference Guide (using a Hardware Security Module for signing) and sections "ECDSA-P256-SHA256 Secure Boot" and "Certificate-Based Secure Boot" in AN1218: Series 2 Secure Boot with RTSL provide additional firmware signing information.

The Gecko Bootloader can still parse the signed SE Firmware GBL upgrade file and flash its content to the device even the standard or secure debug lock applies. Details on SE Firmware upgrade can be found in section "Gecko Bootloader Operation - Secure Engine Upgrade" in UG266: Silicon Labs Gecko Bootloader User's Guide.

Note: The application firmware must be re-programmed after the SE Firmware upgrade if the GBL file storage overwrites the existing application.

12. Alternatives for Series 2 Programming

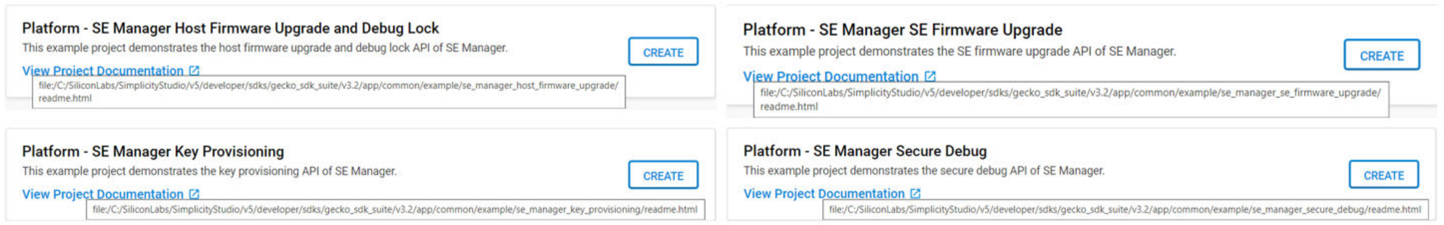
Except for the Simplicity Studio and Studio Commander, a mailbox interface from the Cortex-M33 or a dedicated Debug Challenge Interface (DCI) can be used to program the Series 2 devices.

12.1 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	Unlock the device, enable secure debug, and disable device erase.


Platform - SE Manager Host Firmware Upgrade and Debug Lock
This example project demonstrates the host firmware upgrade and debug lock API of SE Manager.
View Project Documentation
file:/C:/SiliconLabs/SimplicityStudio/v5/developer/sdks/gecko_sdk_suite/v3.2/app/common/example/se_manager_host_firmware_upgrade/readme.html

Platform - SE Manager SE Firmware Upgrade
This example project demonstrates the SE firmware upgrade API of SE Manager.
View Project Documentation
file:/C:/SiliconLabs/SimplicityStudio/v5/developer/sdks/gecko_sdk_suite/v3.2/app/common/example/se_manager_se_firmware_upgrade/readme.html

Platform - SE Manager Key Provisioning
This example project demonstrates the key provisioning API of SE Manager.
View Project Documentation
file:/C:/SiliconLabs/SimplicityStudio/v5/developer/sdks/gecko_sdk_suite/v3.2/app/common/example/se_manager_key_provisioning/readme.html

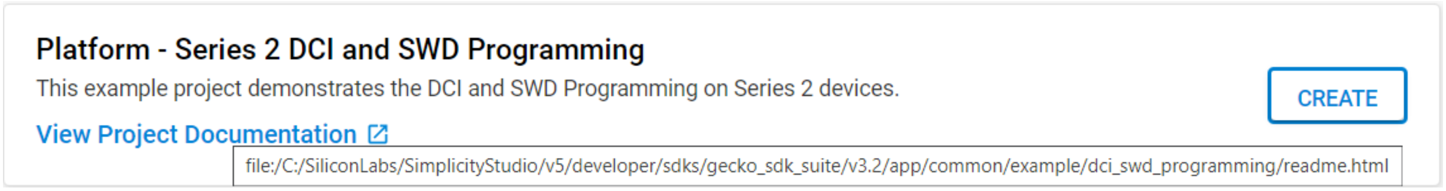
Platform - SE Manager Secure Debug
This example project demonstrates the secure debug API of SE Manager.
View Project Documentation
file:/C:/SiliconLabs/SimplicityStudio/v5/developer/sdks/gecko_sdk_suite/v3.2/app/common/example/se_manager_secure_debug/readme.html

Refer to the corresponding `readme.html` 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.html` file.

12.2 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.html` 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.html` file.


Platform - Series 2 DCI and SWD Programming
This example project demonstrates the DCI and SWD Programming on Series 2 devices.
View Project Documentation
file:/C:/SiliconLabs/SimplicityStudio/v5/developer/sdks/gecko_sdk_suite/v3.2/app/common/example/dci_swd_programming/readme.html

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).

13. 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
- 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)

14. Revision History

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 2. Device Compatibility.
- Removed terminology in 3. Overview.
- Updated Simplicity Commander version to 1.11.2 in 4. Using Simplicity Commander.
- Added security notification figure to 5.1 Overview.
- Updated SE Firmware version in Table 5.1 Latest SE Firmware Version on Shipped Series 2 Devices and Modules on page 10.
- Updated the figures in 5.2.1 Check the SE Firmware Version Using Simplicity Studio 5.
- Updated 5.3 How to Find the Latest SE Firmware.
- Updated 8.2 Provisioning the GBL Decryption Key in Simplicity Commander.
- Updated 8.3 Provisioning the Public Sign Key in Simplicity Commander
- Renamed Enabling Secure Boot to 9. Enabling Secure Boot and Tamper Configuration, updated the content.
- Updated 11. Field Upgrade the SE Firmware.
- Added 12. Alternatives for Series 2 Programming.
- Updated 13. 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 2. Device Compatibility.
- Added SE conventions to 3. Overview, updated the figures and content.
- Updated Simplicity Commander version to 1.9.2 in 4. Using Simplicity Commander.
- Added EFRxG21B to 5.2 How to Check the SE Firmware Version on a Device.
- Updated the figures in 5.2.1 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 6. Bootloader Firmware Programming.
- Updated 7. Application Firmware Programming.
- Updated Enabling Secure Boot for SE with Secure Vault devices.
- Added 8.2 Provisioning the GBL Decryption Key in Simplicity Commander.
- Added AN1247 and AN1271 to 13. Related Documents.

Revision 0.2

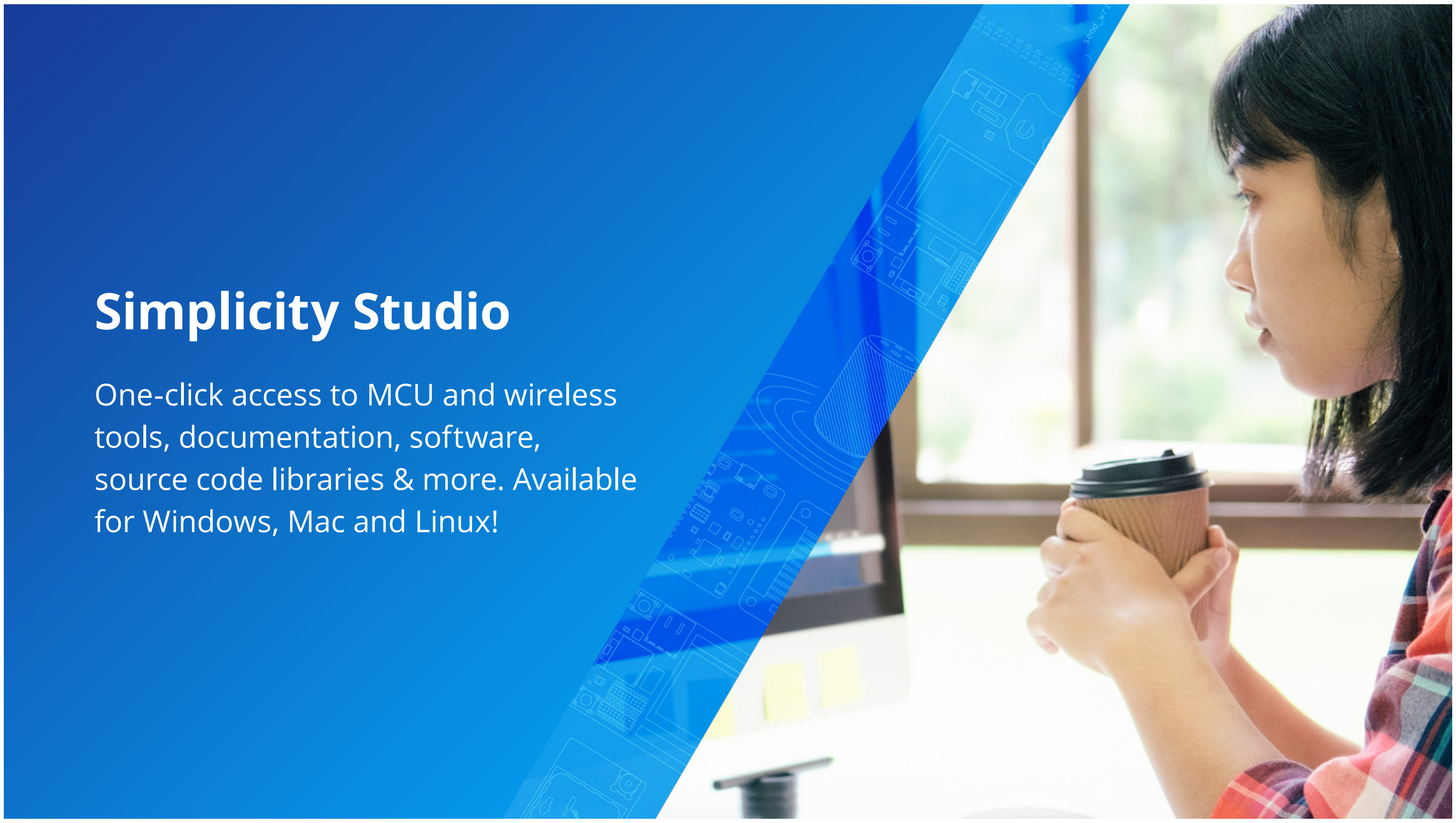
March 2020

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

www.silabs.com