

AN1247: Anti-Tamper Protection Configuration and Use



This application note describes how to program, provision, and configure the Secure Element anti-tamper module. Many aspects of the anti-tamper module, including disabling the anti-tamper response when needed, are discussed.

The anti-tamper module is only available on Secure Vault devices.

KEY POINTS

- Tamper responses
- Tamper sources
- Tamper configuration
- Tamper disable
- Examples of provisioning and disabling the anti-tamper module

1. EFR32 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 EFR32 Series 2 products that included a Secure Element. The Secure Element is a tamper-resistant component used to securely store sensitive data, keys and to execute cryptographic functions and secure services.

The Secure Element is the foundation 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 to be executed.
- **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.

Some EFR32 Series 2 products offer additional security options through Secure Vault. Secure Vault is a dedicated security CPU that isolates cryptographic functions and data from the host processor core. Devices with Secure Vault offer the following security features:

- **Secure Key Storage:** Protects cryptographic keys by “wrapping” or encrypting the keys using a root key known only to the Secure Vault.
- **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 Element 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 EFR32 Series 2 debug access, including background information about the Secure Element	EFR32 Series 2
AN1218: Series 2 Secure Boot with RTSL	Describes the secure boot process on EFR32 Series 2 devices using Secure Element	EFR32 Series 2
AN1247: Anti-Tamper Protection Configuration and Use (this document)	How to program, provision, and configure the anti-tamper module	EFR32 Series 2 with Secure Vault
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	EFR32 Series 2 with Secure Vault
AN1271: Secure Key Storage	How to securely “wrap” keys so they can be stored in non-volatile storage.	EFR32 Series 2 with Secure Vault
AN1222: Production Programming of Series 2 Devices	How to program, provision, and configure security information using Secure Element during device production	EFR32 Series 2

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.

Table 1.1.

Key Name	SE Manager ID	Customer Programmed	Purpose	Used in
Public Sign key (Sign Key Public)	SL_SE_KEY_SLOT_APPLICATION_SE-CURE_BOOT_KEY	Yes	Secure Boot binary authentication and/or OTA upgrade payload authentication	AN1218 (primary), AN1222
Public Command key (Command Key Public)	SL_SE_KEY_SLOT_APPLICATION_SE-CURE_DEBUG_KEY	Yes	Secure Debug Unlock or Disable Tamper command authentication	AN1190 (primary), AN1222, AN1247
OTA Decryption key (GBL Decryption key) aka AES-128 Key	SL_SE_KEY_SLOT_APPLICATION_AES_128_KEY	Yes	Decrypting GBL payloads used for firmware upgrades	AN1222 (primary), UG266
Attestation key aka Private Device Key	SL_SE_KEY_SLOT_APPLICATION_ATTESTATION_KEY	No	Device authentication for secure identity	AN1268

2. Device Compatibility

This application note supports Series 2 device families with Secure Vault, and some functionality is different depending on the device.

Wireless SoC Series 2 with Secure Vault families consist of:

- EFR32BG21B
- EFR32MG21B

3. Introduction

The Secure Vault Anti-Tamper module is used to hamper or prevent both reverse engineering and re-engineering of proprietary software systems or applications.

Tamper attacks come from one or more vectors. Common attacks include voltage glitching, magnetic interference, and forced temperature adjustment. The Secure Vault Anti-Tamper module provides fast hardware detection of external tamper signals such as case opening, glitching, and logical attacks allowing analysis and escalation up to and including bricking the device.

The anti-tamper module connects a number of hardware and software-driven tamper signals to a set of configurable hardware and software responses. This can be used to program the device to automatically respond to external events that could signal that someone is trying to tamper with the device, and very rapidly remove secrets stored in the Secure Element.

The available tamper signals range from signals based on failed authentication and secure boot to specialized glitch detectors. When any of these signals fire, the tamper block can be configured to trigger several different responses, ranging from triggering an interrupt to erasing the One-Time-Programmable (OTP) memory, removing all Secure Element secrets and resulting in a permanently destroyed device.

For more information about Secure Element, see section "*Secure Element Subsystem*" in [AN1190: Series 2 Secure Debug](#).

4. Secure Element Manager

The Secure Element Manager provides thread-safe APIs for the Secure Element's mailbox interface. The Secure Element Manager APIs related to tamper operations are listed in [Table 4.1 Secure Element API for Tamper Operations on page 6](#).

For the Secure Element's mailbox interface, see section "*Secure Element Subsystem*" in [AN1190: Series 2 Secure Debug](#).

Table 4.1. Secure Element API for Tamper Operations

Secure Element Manager API	Usage
<code>sl_se_init_otp</code>	Initialize SE OTP configuration (including tamper settings)
<code>sl_se_read_otp</code>	Read SE OTP configuration (including tamper settings)
<code>sl_se_get_status</code>	Read the current SE status (including recorded tamper status)
<code>sl_se_get_reset_cause</code>	Read the <code>EMU->RSTCAUSE</code> register after a tamper reset

5. Tamper Responses

A [tamper source](#) can lead to a series of different autonomous responses from the SE. These responses are listed in [Table 5.1 Tamper Responses on page 7](#).

Table 5.1. Tamper Responses

Level ¹	Response ²	Description
0	Ignore	No action is taken
1	Interrupt	Triggers the SETAMPERHOST interrupt on the host
2	Filter	Increases a counter in the tamper filter
4	Reset	Resets the device
7	Erase OTP	Erases the device's OTP configuration
Note: 1. Level 3, 5 and 6 are reserved. 2. These responses are cumulative, meaning that if a filter response is triggered, an interrupt will also be triggered.		

5.1 Interrupt

If a tamper source is configured to respond with the interrupt response or higher (\geq level 1), the SETAMPERHOST interrupt line to the host Cortex-M33 will be pulsed and make the NVIC trigger the corresponding interrupt handler (SETAMPERHOST_IRQHandler).

After the interrupt has been handled, the tamper status can be found by reading the SE status (using `sl_se_get_status` in the [Secure Element Manager](#)), which contains a list of all the tamper sources that have been triggered since the last time the status was read. Reading SE status clears the registered tamper sources.

5.2 Filter

The SE has a filter that can be used to filter out spurious tamper events. The filter has a counter that is periodically reset. If a tamper source is configured to the filter level (level 2), when it is triggered, the counter is increased. If the counter value reaches a configurable threshold, the Filter counter tamper source ([number 1](#)) is triggered, which can be configured to lead to any of the other responses.

Only a single shared filter counter is available, so the cumulative triggering of all tamper sources configured to the filter level will increase the same counter. The filter can be configured to use one of the trigger thresholds and reset periods given in [Table 5.2 Filter Trigger Threshold on page 7](#) and [Table 5.3 Filter Reset Period on page 7](#). The filter counter is reset upon a tamper reset.

Table 5.2. Filter Trigger Threshold

Value (n)	Filter Trigger Threshold
0 to 7	$256/2^n$ (256 to 2)

Table 5.3. Filter Reset Period

Value (n)	Filter Reset Period
0 to 31	$32 \text{ ms} * 2^n$ (32 ms to ~795.4 days)

5.3 Reset

The reset response resets the SE and Cortex-M33. After a tamper reset, the last reset cause (`EMU->RSTCAUSE`) can be read using `sl_se_get_rstcause` in the [Secure Element Manager](#).

If a tamper reset is triggered during boot, this can lead to a boot loop. To debug such a scenario, the SE has a tamper reset counter and enters diagnostic mode if the counter reaches a [programmable threshold](#). Users can issue a non-tamper reset to clear the tamper reset counter before the programmable threshold is reached.

In diagnostic mode, the Cortex-M33 is held in reset and only DCI commands are available. The device will remain in diagnostic mode until a power-on or pin reset occurs.

For Secure Element's DCI interface, see section "*Secure Element Subsystem*" in [AN1190: Series 2 Secure Debug](#).

5.4 Erase OTP

The Erase OTP response is the strongest reaction the SE can take, and it will make the device and all wrapped secrets unrecoverable. After this response, the device will no longer be able to boot.

This response should typically only be used in situations where the device believes that it is under an actual attack, for instance through the detection of several voltage or digital glitches in a short time window.

6. Tamper Sources

The SE determines the minimum for each tamper source and the default tamper responses (> 0) are always enabled. Users may escalate the tamper response of any source when [initially configuring](#) the part. [Table 6.1 Tamper Sources on the EFR32xG21B Devices on page 9](#) lists the available tamper sources and the default level (response) on the EFR32xG21B devices.

Table 6.1. Tamper Sources on the EFR32xG21B Devices

Type	Number ¹	Name	Description	Default Level ² (Response)
SE Hardware	0	Reserved	—	—
	1	Filter counter	Filter counter reaches configured threshold value	0 (Ignore)
	2	SE watchdog	Internal SE watchdog expires	4 (Reset)
	3	Reserved	—	—
	4	SE RAM CRC	SE RAM parity error occurs	4 (Reset)
	5	SE hard fault	SE core hard fault occurs	4 (Reset)
	6	Reserved	—	—
SE Software	7	SE software assertion	SE software triggers an assert	4 (Reset)
	8	Reserved	—	—
	9	User secure boot ³	Secure boot of host firmware fails	0 (Ignore)
	10	Mailbox authorization	Unauthorised command received over the Mailbox interface	0 (Ignore)
	11	DCI authorization	Unauthorised command received over the DCI interface	0 (Ignore)
	12	Flash integrity	OTP, MTP or flash content could not be properly authenticated	4 (Reset)
	13	Reserved	—	—
	14	Self test	Integrity error of internal storage is detected	4 (Reset)
	15	TRNG monitor	TRNG monitor detected lack of entropy	0 (Ignore)
Hardware	16 - 23	PRS0 - 7 ⁴	PRS channel 0 - 7 is asserted	0 (Ignore)
	24	Decouple BOD	Decouple brown-out-detector threshold alert	4 (Reset)
	25	Temperature sensor ⁵	On-chip temperature sensor detects a temperature outside the operational conditions of the device	0 (Ignore)
	26	Voltage glitch falling	Voltage glitch detector detects a falling glitch	0 (Ignore)
	27	Voltage glitch rising	Voltage glitch detector detects a rising glitch	0 (Ignore)
	28	Secure lock	Debug lock internal logic check fails	4 (Reset)
	29	SE debug	SE debug granted	0 (Ignore)
	30	Digital glitch	Digital glitch detector detects an event	0 (Ignore)
	31	SE ICACHE	SE instruction cache checksum error	4 (Reset)

Type	Number ¹	Name	Description	Default Level ² (Response)
Note: <ol style="list-style-type: none"> 1. Tamper sources 24 to 27 can operate down to Energy Mode 3 (EM3) whereas other tamper sources can operate down to Energy Mode 1 (EM1). 2. User configuration or tamper disable cannot reduce the tamper response below the default Level (Figure 9.2 Disable Tamper Command on the EFR32xG21B Devices on page 15). 3. The user secure boot source gets triggered if secure boot is enabled and host image verification fails. It is likely to put the device in boot loop if users escalate the tamper response of this source to 4 (Reset). 4. PRS inputs can allow user applications to implement additional tamper sources and feed them into the tamper response mechanism. The PRS tamper sources are under the control of the user application and could be reconfigured or disabled if the user application is compromised. 5. The Temperature Sensor tamper source is not completely accurate and is generally only suitable for systems that expect to stay well within the specified temperature range. Users requiring a tighter temperature limit can implement their own temperature monitor and provide the results as a tamper source via PRS. 				

7. Anti-Tamper Configuration

The anti-tamper configuration is one-time programmable (OTP). The SE OTP is provisioned by `sl_se_init_otp` in the [Secure Element Manager](#). This means that [tamper settings](#) must be written together with secure boot settings, and are immutable after they are written. The settings for tamper configuration are listed in [Table 7.1 Anti-Tamper Configuration on page 11](#).

For secure boot settings, see section "Secure Boot Enabling" in [AN1222: Production Programming of Series 2 Devices](#).

Table 7.1. Anti-Tamper Configuration

Setting	Description
Tamper response levels	A response level for each tamper source ¹
Filter settings	The tamper filter counter has two settings: <ul style="list-style-type: none">• Trigger threshold (Table 5.2 Filter Trigger Threshold on page 7)• Reset period (Table 5.3 Filter Reset Period on page 7)
Flags	Digital Glitch Detector Always On (bit 1) <ul style="list-style-type: none">• 0 — Digital glitch detector (number 30) runs when the SE is executing a command• 1 — Digital glitch detector runs continually even when the SE is not performing any operations²
Reset threshold	The number of consecutive tamper resets (up to 255) before the part enters diagnostic mode ³
Note: <ol style="list-style-type: none">1. It is not possible to degrade the default level of a tamper source, so if a response is set to a lower level than the default level, this will not have any effect.2. This leads to increased energy consumption.3. If the threshold is set to 0, the part will never enter the diagnostic mode due to tamper reset.	

8. Usage Example

Several of the available [tamper sources](#) report internal SE errors. By default, the anti-tamper module is configured to reset the device (level 4) if any of a number of these SE errors are detected. Custom handling of internal and external tamper sources (default level 0) can be configured to trigger an interrupt (level 1) on the Cortex-M33 or increase a counter in the tamper filter (level 2) as in [Figure 8.1 Custom Handling of Tamper Sources](#) on page 12.

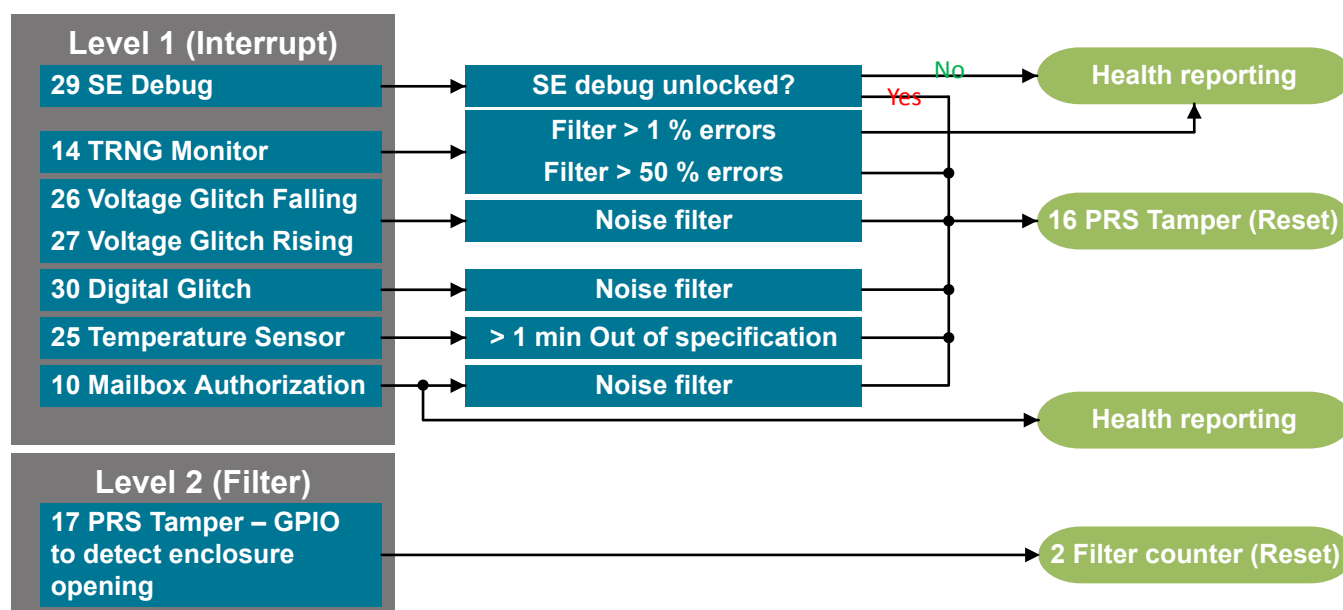


Figure 8.1. Custom Handling of Tamper Sources

Usage example highlights:

- The SE debug is used to trigger an interrupt when an attempt to open the SE debug port occurs; further action depends on debug port status.
- The further response of the TRNG monitor depends on the failure rate due to lack of entropy.
- The voltage and digital glitch detectors can see spurious activations, and should typically not be used to directly drive a serious tamper response. Instead they should feed their signals into a tamper interrupt, which can be used to activate a high-level response (Reset in this example) through PRS tamper if a certain number of detections (noise filter) occur in a short time window.
- The time out of the specification filter for the temperature sensor should be based on the system operating conditions. For some systems, any time out of specification should trigger a reset.
- Mailbox authorization is handled similarly for voltage and digital glitch detectors.
- A high-level response for a tamper interrupt is implemented by a PRS tamper, which issues a tamper reset (level 4) that will prevent or slow further attack.
- In extreme cases, if an attack is identified with high confidence, a PRS tamper can be configured as [Erase OTP](#) (level 7) to brick the part and prevent further attack. This is recommended only when destruction of parts is acceptable and high confidence of an attack can be achieved.
- Another PRS tamper is used to detect enclosure opening from GPIO. This source is fed into the tamper filter counter (level 2), which will activate a [Filter counter \(number 1\)](#) response (Reset in this example) if the filter counter reaches the [trigger threshold](#) within the [filter reset period](#). It is less flexible than the interrupt response approach, since the trigger threshold and filter reset period are only one-time programmable.

9. Tamper Disable

For diagnostic purposes, it may be necessary to disable the tamper response ('tamper disable'). For example, if a user has configured the part to [Erase OTP](#) on external tamper detection, it will be necessary to disable that response to open the unit and perform failure analysis or field service activities.

After [tamper settings](#) have been initialized, users can temporarily restore the tamper response for a set of [tamper sources](#) via a [disable tamper command](#) authenticated against the Public Command Key in SE OTP (similar to secure debug unlock). This is only possible if the [Public Command Key](#) has been provisioned in the device.

9.1 Disable Tamper Command

The elements of the disable tamper command are described in [Figure 9.1 Disable Tamper Command on page 14](#) and [Table 9.1 Elements of Disable Tamper Command on page 14](#).

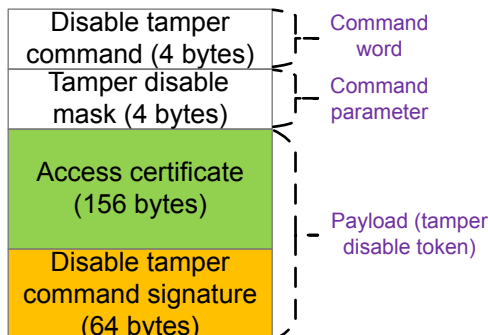


Figure 9.1. Disable Tamper Command

Table 9.1. Elements of Disable Tamper Command

Element	Value	Description
Disable tamper command	0xfd020001	The command word of disable tamper command.
Tamper disable mask	Device-dependent	The command parameter of disable tamper command (Table 9.2 Tamper Disable Mask on page 14).
Access certificate ¹	Device-dependent	See 9.2 Access Certificate .
Disable tamper command signature ¹	Device-dependent	See 9.3 Challenge Response .

Note:

1. The disable tamper command payload ([tamper disable token](#)) consists of an access certificate and a disable tamper command signature.

Table 9.2. Tamper Disable Mask

Tamper Disable Mask																																																																	
Name	Bit	Tamper source 31	31	Tamper source 30	30	Tamper source 29	29	Tamper source 28	28	Tamper source 27	27	Tamper source 26	26	Tamper source 25	25	Tamper source 24	24	Tamper source 23	23	Tamper source 22	22	Tamper source 21	21	Tamper source 20	20	Tamper source 19	19	Tamper source 18	18	Tamper source 17	17	Tamper source 16	16	Tamper source 15	15	Tamper source 14	14	Tamper source 13	13	Tamper source 12	12	Tamper source 11	11	Tamper source 10	10	Tamper source 9	9	Tamper source 8	8	Tamper source 7	7	Tamper source 6	6	Tamper source 5	5	Tamper source 4	4	Tamper source 3	3	Tamper source 2	2	Tamper source 1	1	Tamper source 0	0

Note:

- Set bit to restore the default response of corresponding [tamper source](#).

The disable tamper command simply reverts all masked tamper sources in [Table 9.2 Tamper Disable Mask on page 14](#) to the hard-coded configuration (default levels in [Table 6.1 Tamper Sources on the EFR32xG21B Devices on page 9](#)).

The disable tamper command can only undo the user level configuration ([Figure 9.2 Disable Tamper Command on the EFR32xG21B Devices on page 15](#)). The default level of a tamper source cannot be degraded.

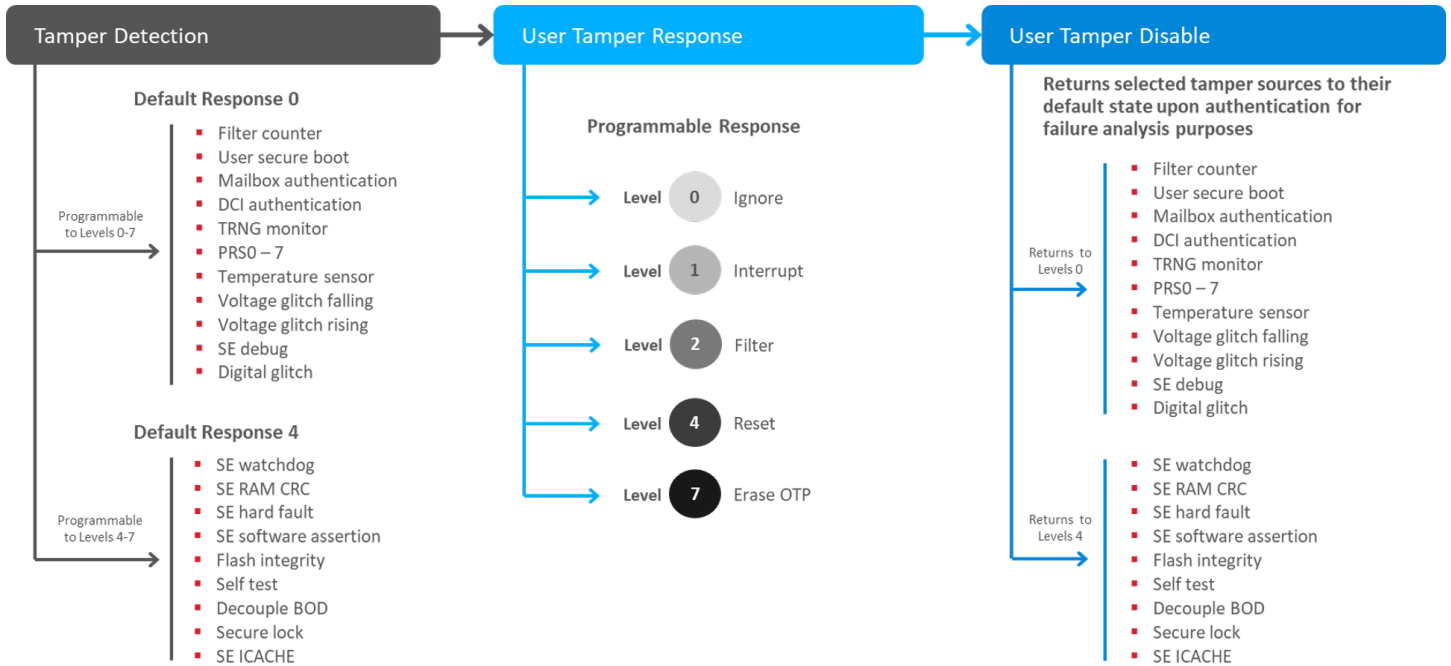


Figure 9.2. Disable Tamper Command on the EFR32xG21B Devices

9.2 Access Certificate

The elements of the access certificate are described in [9.2 Access Certificate](#) and [Table 9.3 Elements of the Access Certificate on page 16](#).

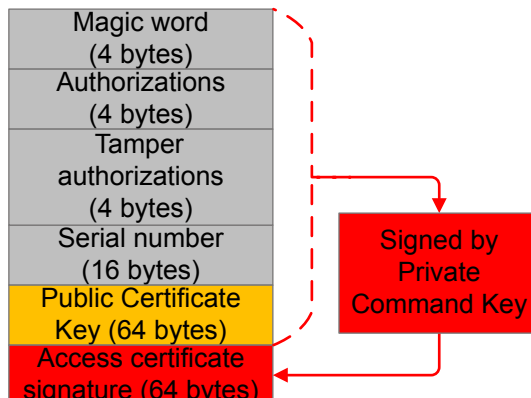


Figure 9.3. Access Certificate

Table 9.3. Elements of the Access Certificate

Element	Value	Description
Magic word	0xe5ecce01	A constant value used to identify the access certificate.
Authorizations	0x0000003e ¹	A value used to authorize which bit in debug mode request can be enabled for secure debug.
Tamper Authorizations	0xffffffffb6 ²	A value used to authorize which bit in tamper disable mask can be enabled (Table 9.4 Tamper Authorizations on page 17) for tamper disable.
Serial number	Device-dependent	A number used to compare against the on-chip serial number for secure debug or tamper disable.
Public Certificate Key ³	Device-dependent	The public key corresponding to the Private Certificate Key ³ used to generate the signature (ECDSA-P256-SHA256) in challenge response .
Access certificate signature	Device-dependent	All the content above is signed (ECDSA-P256-SHA256) by the Private Command Key corresponding to the Public Command Key in SE OTP.
Note: <ol style="list-style-type: none"> 1. Value that allows full debug access for secure debug. 2. Value that enables available bits in tamper disable mask for tamper disable. 3. The Private/Public Certificate Key is a randomly generated key pair. It can be ephemeral or retainable. 		

The Private Certificate Key can be used repeatedly to generate the signature in a [challenge response](#) on one device until the Private/Public Certificate Key pair is discarded. This can reduce the frequency of access to the Private Command Key, allowing more restrictive access control on that key.

For secure debug, see section "Secure Debug Unlock" in [AN1190: Series 2 Secure Debug](#).

Table 9.4. Tamper Authorizations

Tamper Authorizations	
Bit	
31	Tamper disable mask 31
30	Tamper disable mask 30
29	Tamper disable mask 29
28	Tamper disable mask 28
27	Tamper disable mask 27
26	Tamper disable mask 26
25	Tamper disable mask 25
24	Tamper disable mask 24
23	Tamper disable mask 23
22	Tamper disable mask 22
21	Tamper disable mask 21
20	Tamper disable mask 20
19	Tamper disable mask 19
18	Tamper disable mask 18
17	Tamper disable mask 17
16	Tamper disable mask 16
15	Tamper disable mask 15
14	Tamper disable mask 14
13	Tamper disable mask 13
12	Tamper disable mask 12
11	Tamper disable mask 11
10	Tamper disable mask 10
9	Tamper disable mask 9
8	Tamper disable mask 8
7	Tamper disable mask 7
6	Tamper disable mask 6
5	Tamper disable mask 5
4	Tamper disable mask 4
3	Tamper disable mask 3
2	Tamper disable mask 2
1	Tamper disable mask 1
0	Tamper disable mask 0

Note:

- Set bit to enable the corresponding bit in [tamper disable mask](#).
- Default response of corresponding [tamper source](#) will be restored by disable tamper command if same bit in [Table 9.2 Tamper Disable Mask on page 14](#) and [Table 9.4 Tamper Authorizations on page 17](#) are set.

9.3 Challenge Response

The elements of challenge response are described in [Figure 9.4 Challenge Response on page 17](#) and [Table 9.5 Elements of Challenge Response on page 17](#).

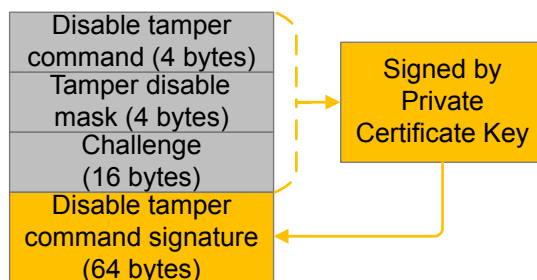


Figure 9.4. Challenge Response

Table 9.5. Elements of Challenge Response

Element	Value	Description
Disable tamper command	0xfd020001	The command word of disable tamper command .
Tamper disable mask	Device-dependent	The command parameter of disable tamper command .
Challenge	Device-dependent ¹	A random value generated by the SE.
Disable tamper command signature	Device-dependent ²	All the content above is signed (ECDSA-P256-SHA256) by the Private Certificate Key corresponding to the Public Certificate Key in the access certificate .

Note:

- The challenge remains unchanged until it is updated to a new random value by [rolling the challenge](#). The Private Certificate Key can be reused for signing when device challenge is refreshed.
- This signature is the final argument of the [disable tamper command](#).

9.4 Tamper Disable Flow

The tamper disable flow is described in [Figure 9.5 Tamper Disable Flow on page 18](#).

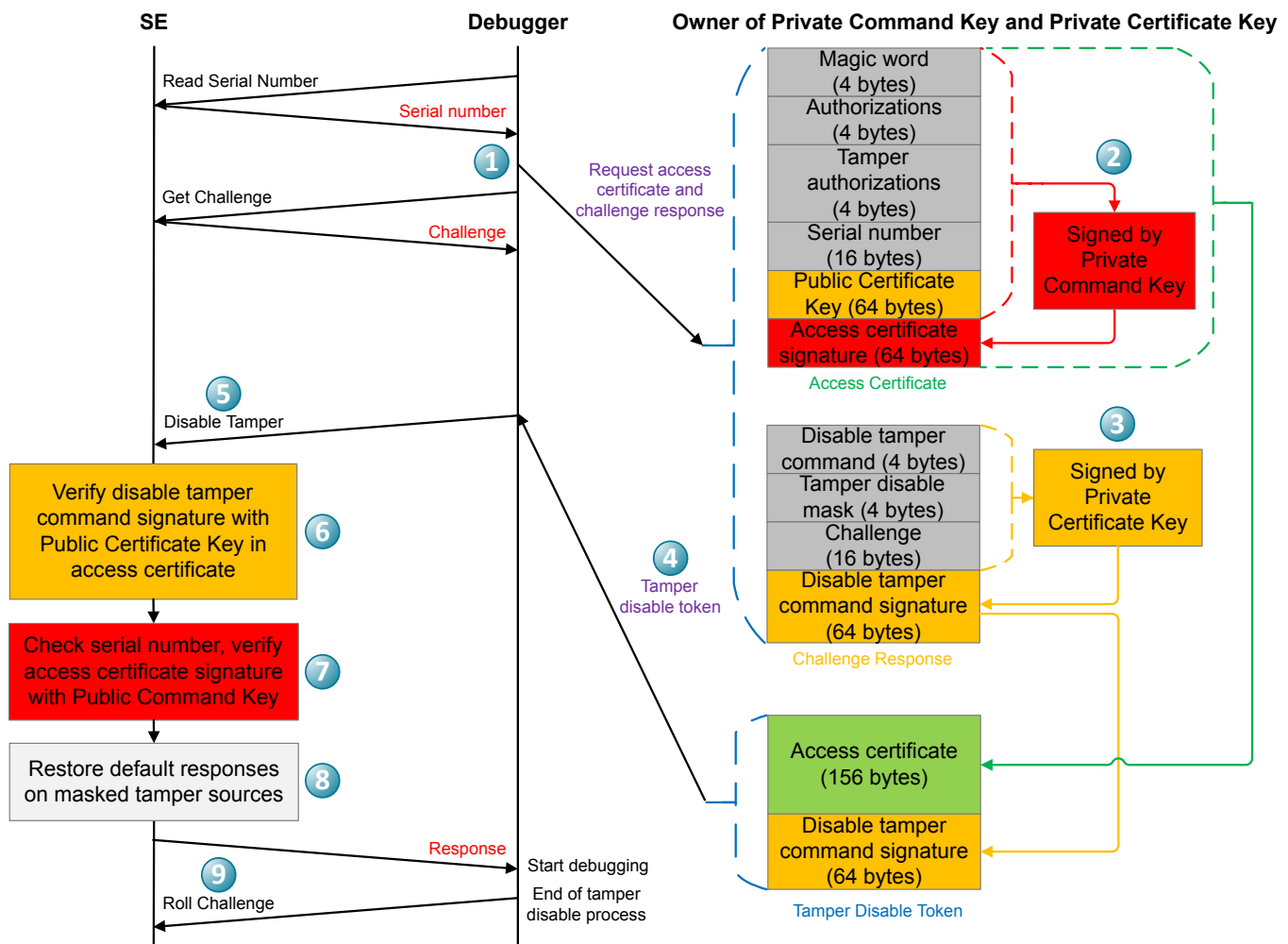


Figure 9.5. Tamper Disable Flow

1. Get the serial number and challenge from the SE.
2. Generate the [access certificate](#) with device serial number.
3. Generate the [challenge response](#) with device challenge.
4. Generate the tamper disable token ([payload](#) of disable tamper command) with access certificate and disable tamper command signature.
5. Send the disable tamper command to the SE.
6. Verify the disable tamper command signature using the Public Certificate Key in the access certificate.
7. Verify the serial number and the access certificate signature using the on-chip serial number and [Public Command Key](#) in SE OTP.
8. Restore default responses on masked [tamper sources](#) until the next power-on or pin reset.
9. [Roll the challenge](#) to invalidate the current disable tamper command.

10. Examples

10.1 Overview

The examples for Secure Vault Anti-Tamper module are described in [Table 10.1 Tamper Examples on page 19](#).

Table 10.1. Tamper Examples

Example	Device	Radio Board	SE Firmware	Tool
Provision Public Command Key	EFR32MG21B010F1024IM32	BRD4181C	Version 1.2.1	Simplicity Studio
	EFR32MG21B010F1024IM32	BRD4181C	Version 1.2.1	Simplicity Commander
Provision tamper settings	EFR32MG21B010F1024IM32	BRD4181C	Version 1.2.1	Simplicity Commander
	EFR32MG21B010F1024IM32	BRD4181C	Version 1.2.1	Secure Element Manager
Tamper disable	EFR32MG21B010F1024IM32	BRD4181C	Version 1.2.1	Simplicity Commander

10.1.1 Using Simplicity Commander

1. Simplicity Commander's Command Line Interface (CLI) is invoked by `commander.exe` in the Simplicity Commander folder. The location in Windows is `C:\SiliconLabs\SimplicityStudio\<version>\developer\adapter_packs\commander`.
2. Simplicity Commander Version 1.9.2 is used in this application note.

```
commander --version
```

```
Simplicity Commander 1v9p2b791
```

```
JLink DLL version: 6.70a
Qt 5.12.1 Copyright (C) 2017 The Qt Company Ltd.
EMDLL Version: 0v17p12b535
mbed TLS version: 2.6.1
```

```
DONE
```

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. The target device must be specified using the `--device <device name>` option if the WSTK is in debug mode OUT.
5. Run the `security genkey` command to generate the Private/Public Command Key pair (`command_key.pem` and `command_pubkey.pem`) for tamper examples.

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

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

6. Run the `gbl keyconvert` command to generate the Public Command Key text file (`command_pubkey.txt`) for the [Provision Public Command Key](#) example.

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

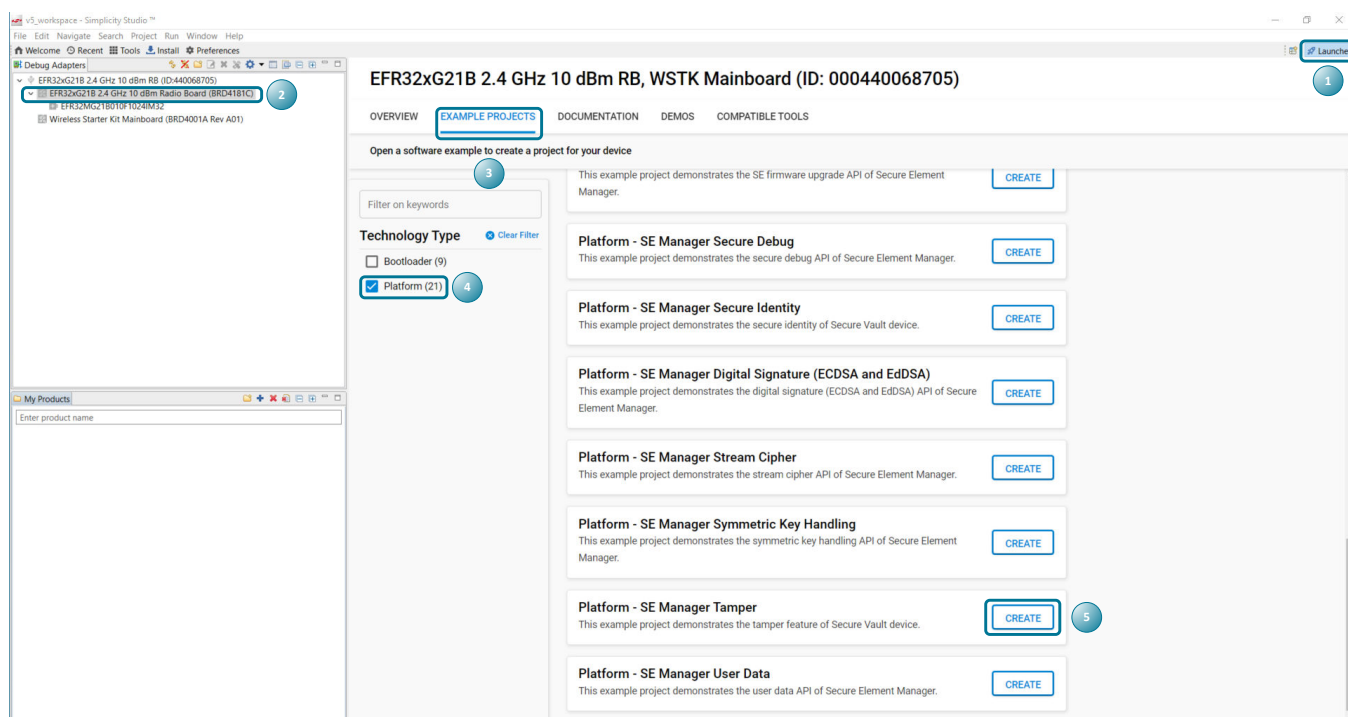
```
Writing EC tokens to command_pubkey.txt...
DONE
```

For more information about Simplicity Commander, see [UG162: Simplicity Commander Reference Guide](#).

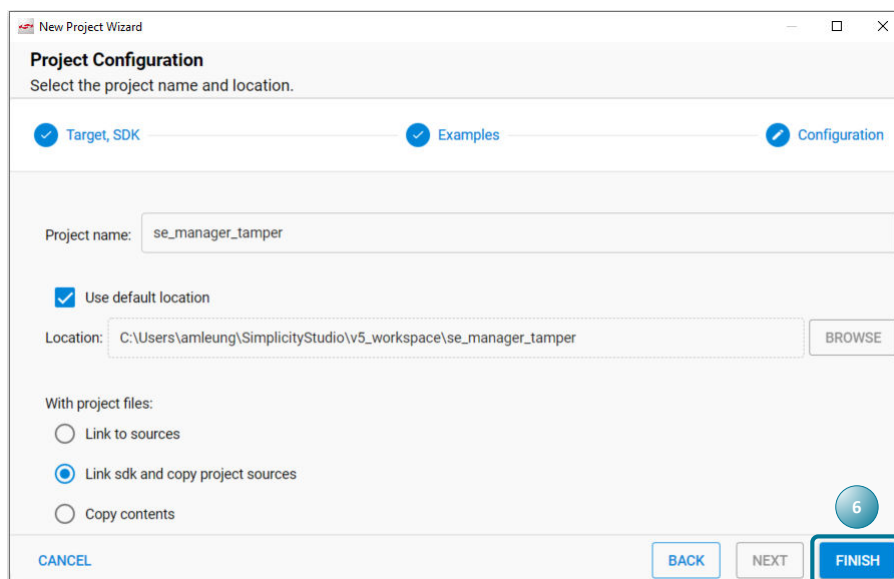
10.1.2 Using a Platform Example


This section describes how to use Simplicity Studio 5 to build the tamper platform example and program it to the Wireless Starter Kit (WSTK).

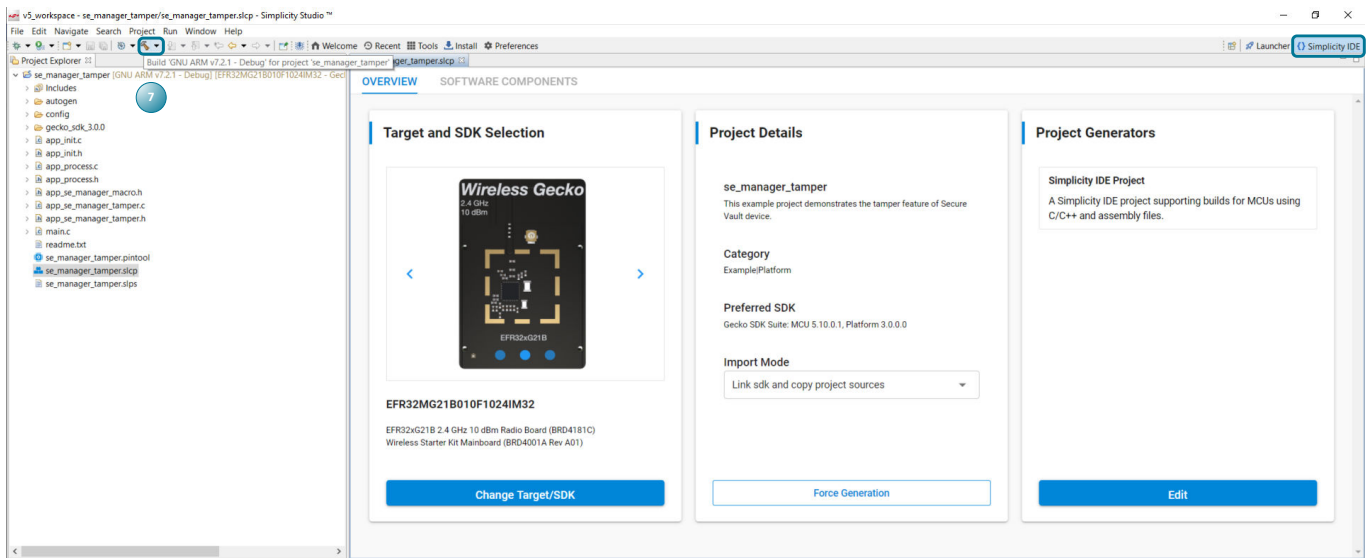
1. The connected WSTK will display on the **[Launcher]** perspective.
2. In the **[Debug Adapters]** view, click the target radio board (**BRD4181C** in this example). This automatically configures the task bars for use with your device.
3. From the **[Launcher]** perspective, click **[EXAMPLE PROJECTS]**.
4. In the **[EXAMPLE PROJECTS]** dialog, check the **Platform (n)** under **Technology Type**.
5. Search for the **Platform - SE Manager Tamper** example and click **[CREATE]**.



6. In the **[Project Configuration]** dialog, optionally name your project and select a different project location. Click **[FINISH]**. The **[Simplicity IDE]** perspective opens.



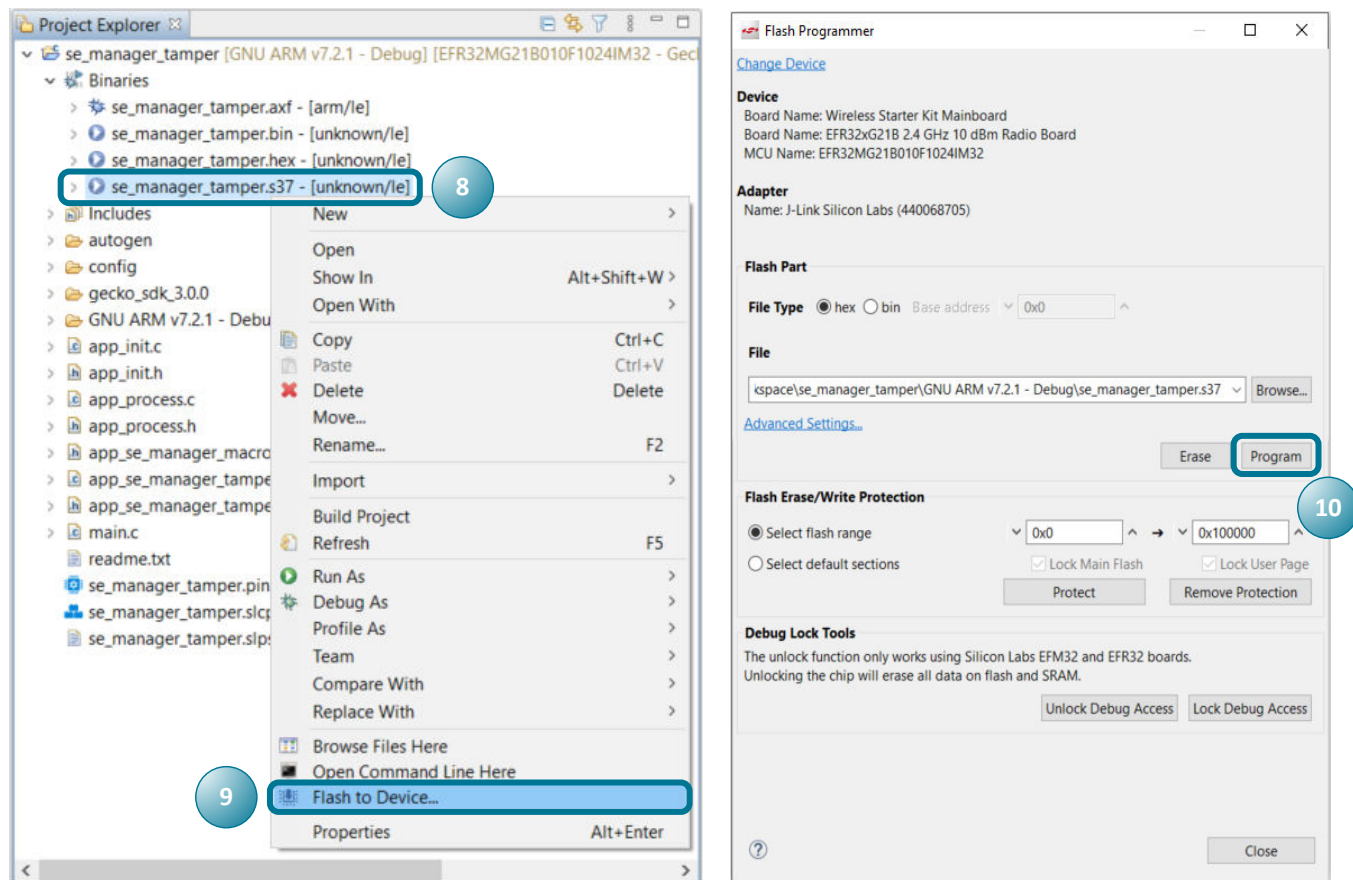
7. In the [Simplicity IDE] perspective, click the [Build] icon ().



8. In the Project Explorer view, under **Binaries**, right-click the `se_manager_tamper.s37` file.

9. In the resulting context menu, click [Flash to Device...]. This opens the **Flash Programmer**.

10. Click [Program] to flash the `se_manager_tamper.s37` file to the radio board.



10.1.3 External Tools

1. OpenSSL is used in the tamper disable example to sign the access certificate and disable tamper command. The Windows version of OpenSSL can be downloaded from here — <https://slproweb.com/products/Win32OpenSSL.html>.
2. The free Hex Editor Neo is used in the tamper disable example to edit the binary files generated by Simplicity Commander. The Windows version of Hex Editor Neo can be downloaded from here — <https://www.hhdsoftware.com/free-hex-editor>.

10.2 Provision Public Command Key

The Public Command Key (`command_pubkey.pem` and `command_pubkey.txt`) for provisioning was generated with step 5 and 6 in [10.1.1 Using Simplicity Commander](#).

10.2.1 Simplicity Studio and Simplicity Commander

For Simplicity Studio and Simplicity Commander examples, see section "*Provision Public Command Key*" in [AN1190: Series 2 Secure Debug](#). The same procedures apply to Secure Vault devices.

10.3 Provision Tamper Settings

10.3.1 Simplicity Commander

For a Simplicity Commander example, see section "*Secure Boot Enabling*" in [AN1222: Production Programming of Series 2 Devices](#).

10.3.2 Secure Element Manager

See section [10.1.2 Using a Platform Example](#) for more information on programming the tamper platform example to the WSTK. The `se_manager_tamper` example uses `sl_se_init_otp` in [Secure Element Manager](#) to provision the tamper settings on Secure Vault devices.

The example redirects standard I/O to the virtual serial port (VCOM) of the WSTK. Open a terminal program (e.g. Tera Term) and access the WSTK VCOM port (default setting is 115200 bps 8-N-1).

`se_manager_tamper` Highlights:

1. The [tamper configuration](#) can be read back by `sl_se_read_otp` in [Secure Element Manager](#) if tamper settings have been provisioned in the SE OTP.

```
SE Manager Tamper Example - Core running at 38000 kHz.
. SE manager initialization... SL_STATUS_OK (cycles: 10 time: 0 us)

. Read EMU RSTCAUSE register... SL_STATUS_OK (cycles: 3056 time: 80 us)
+ The EMU RSTCAUSE register (MSB..LSB): 00000043

. Read SE OTP configuration... SL_STATUS_OK (cycles: 7204 time: 189 us)
+ Secure boot: Disabled
+ Tamper source level
  Filter counter      : 1
  SE watchdog        : 4
  SE RAM CRC         : 4
  SE hard fault      : 4
  SE software assertion : 4
  User secure boot   : 0
  Mailbox authorization : 1
  DCI authorization  : 0
  Flash integrity    : 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
  Temperature sensor : 2
  Voltage glitch falling : 2
  Voltage glitch rising : 2
  Secure lock       : 4
  SE debug          : 0
  Digital glitch    : 2
  SE ICACHE        : 4
+ Reset period for the tamper filter counter: ~32 ms x 1024
+ Activation threshold for the tamper filter: 4
+ Digital glitch detector always on: Disabled
+ Tamper reset threshold: 5

. Test instructions:
+ Press PB0 to increase filter counter and tamper status is displayed.
+ PRS will issue a tamper reset if filter counter reaches 4 within ~32 ms x 1024.
+ Press PB1 to issue a tamper result.
+ Device will enter diagnostic mode if tamper reset reaches 5.
```

Note: Secure boot is not enabled in this example.

2. Press PB0 on the WSTK **two** times to provision the [tamper configuration](#) in this example if tamper settings have not been provisioned in the SE OTP.

```
. Read SE OTP configuration... SL_STATUS_COMMAND_IS_INVALID (cycles: 4139 time: 108 us)
+ Cannot read SE OTP configuration.
+ Press PB0 to initialize SE OTP for tamper configuration or press PB1 to abort
+ Warning: The OTP configuration cannot be changed once written!
+ Press PB0 to confirm or press PB1 to abort if you are not sure.
+ Initialize SE OTP for tamper configuration... SL_STATUS_OK (cycles: 77375 time: 2014 us)
```

3. The PRS tamper source usage in this example is listed in [Table 10.2 PRS Tamper Sources Usage on page 24](#).

Table 10.2. PRS Tamper Sources Usage

Number	Name	Default Level (Response)	User Level (Response)	PRS Producer
16	PRS0	0 (Ignore)	1 (Interrupt)	Push button PB0
17	PRS1	0 (Ignore)	1 (Interrupt)	—
18	PRS2	0 (Ignore)	2 (Filter)	Push button PB0
19	PRS3	0 (Ignore)	2 (Filter)	—
20	PRS4	0 (Ignore)	4 (Reset)	Push button PB1
21	PRS5	0 (Ignore)	4 (Reset)	Software ¹
22	PRS6	0 (Ignore)	7 (Erase OTP)	—
23	PRS7	0 (Ignore)	7 (Erase OTP)	—
Note: <ul style="list-style-type: none"> PRS5 response (Reset) is triggered by software PRS if the filter counter reaches the trigger threshold (4) within filter reset period (~32 ms x 1024). 				

4. The tamper status will be displayed if any tamper sources with level 1 (PRS0 in this example) or 2 (PRS2 in this example) is triggered.
5. Press PB0 to trigger PRS0 and PRS2. The active tamper sources (0x00050000) are 16 (PRS0) and 18 (PRS2).

```
. Get tamper status... SL_STATUS_OK (cycles: 11268 time: 296 us)
+ Recorded tamper status (MSB..LSB): 00050001
+ Currently active tamper sources (MSB..LSB): 00050000
```

6. Press PB0 (PRS2) 4 times within ~32.7 seconds to issue a tamper reset through PRS5 (software PRS). The active tamper sources (0x00050002) are 2 (Filter), 16 (PRS0) and 18 (PRS2).

```
. Get tamper status... SL_STATUS_OK (cycles: 11462 time: 301 us)
+ Recorded tamper status (MSB..LSB): 00050002
+ Currently active tamper sources (MSB..LSB): 00050002
+ Tamper filter threshold is reached, issue a reset through PRS
```

7. Press PB1 (PRS4) to issue a tamper reset.
8. After a tamper reset, the SETAMPER bit (location 13) in EMU->RSTCAUSE register is set.

```
. Read EMU RSTCAUSE register... SL_STATUS_OK (cycles: 3064 time: 80 us)
+ The EMU RSTCAUSE register (MSB..LSB): 00002000
+ The tamper reset is observed
```

9. After five consecutive tamper resets ([reset threshold](#) in this example), the device will enter diagnostic mode. The device will remain in diagnostic mode until a power-on or pin reset.
10. PRS6 (Erase OTP) and PRS7 (Erase OTP) will brick the device, so these tamper sources are not used in this example.

10.4 Tamper Disable

The `se_manager_tamper` example ([10.3.2 Secure Element Manager](#)) and [tamper disable mask](#) `0x00fa0000` are used to demonstrate the tamper disable feature in [10.4.1 Local Tamper Disable](#) and [10.4.2 Remote Tamper Disable](#).

The Private/Public Command Key pair (`command_key.pem` and `command_pubkey.pem`) was generated with step 5 in [10.1.1 Using Simplicity Commander](#). The [Public Command Key](#) and [tamper settings](#) must be provisioned in advance for tamper disable.

10.4.1 Local Tamper Disable

The [disable tamper command](#) file can be locally generated if the owner of the Private Command Key can access the device.

Generate the disable tamper command file:

1. Run the `security disabletamper` command with `--disable-param` option and Private Command Key (`command_key.pem` in [10.1.1 Using Simplicity Commander](#) step 5) to restore the masked tamper sources. The [tamper disable mask](#) (`0x00fa0000` in this example) decides which tamper sources will be restored. The masked tamper sources are temporarily restored until the next power-on or pin reset.

```
commander security disabletamper --disable-param 0x00fa0000 --command-key command_key.pem --device EFR32MG21B010F1024 --serialno 440068705
```

```
Command public key stored in:
C:/Users/amleung/AppData/Local/SiliconLabs/commander/SecurityStore/device_00000000000000014b457fffe0f77ce/
command_pubkey.pem
Command private key stored in:
C:/Users/amleung/AppData/Local/SiliconLabs/commander/SecurityStore/device_00000000000000014b457fffe0f77ce/
command_key.pem
Authorization file written to Security Store:
C:/Users/amleung/AppData/Local/SiliconLabs/commander/SecurityStore/device_00000000000000014b457fffe0f77ce/
certificate_authorizations.json
Generating ECC P256 key pair...
Cert public key stored at:
C:/Users/amleung/AppData/Local/SiliconLabs/commander/SecurityStore/device_00000000000000014b457fffe0f77ce/
cert_pubkey.pem
Cert private key stored at:
C:/Users/amleung/AppData/Local/SiliconLabs/commander/SecurityStore/device_00000000000000014b457fffe0f77ce/
cert_key.pem
Certificate was signed with key:
C:/Users/amleung/AppData/Local/SiliconLabs/commander/SecurityStore/device_00000000000000014b457fffe0f77ce/
command_key.pem
Created unsigned disable tamper command
Signed disable tamper command using
C:/Users/amleung/AppData/Local/SiliconLabs/commander/SecurityStore/device_00000000000000014b457fffe0f77ce/
cert_key.pem
Tamper successfully disabled.
Command disable tamper payload was stored in Security Store
DONE
```

Note: The disable tamper command file is generated with the default certificate authorization file (`certificate_authorization.json`) which uses `0x0000003e` for Authorizations and `0xffffffffb6` for Tamper Authorizations ([Table 9.3 Elements of the Access Certificate on page 16](#)).

2. All the generated files, as well as the Private Command Key (`command_key.pem`), are stored in the Security Store. The location in Windows is `C:\Users\<PC user name>\AppData\Local\SiliconLabs\commander\SecurityStore\device_<Serial number>`.



[illegible]

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

See • [Restore tamper responses: on page 31.](#)

The [disable tamper command](#) file can be remotely generated if the owner of the Private Command Key cannot access the device.

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

```
SE Firmware version : 1.2.1
Serial number      : 000000000000000014b457fffe0f77ce
Debug lock          : Disabled
Device erase        : Enabled
Secure debug unlock : Disabled
Tamper status       : OK
Secure boot         : Disabled
Boot status         : 0x20 - OK
DONE
```

```
commander security gencommand --action disable-tamper --disable-param 0x00fa0000 -o command_unsign.bin --nostore --device EFR32MG21B010F1024 --serialno 440068705
```

```
Unsigned command file written to:
command_unsign.bin
DONE
```

command_unsign.bin																
00000174	00	01	02	03	04	05	06	07	08	09	0a	0b	0c	0d	0e	0f
00000000	01	00	02	fd	00	00	fa	00	fc	3d	2a	b4	1c	07	56	2b
00000010	d3	1e	3a	15	42	d6	fb	d5								

3. Send the device part number (EFR32MG21B010F1024), device serial number (000000000000000014b457fffe0f77ce), and unsigned command file (command_unsign.bin) to the owner of the Private Command Key.

Authorize the remote tamper disable request (WSTK is not required):

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

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

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

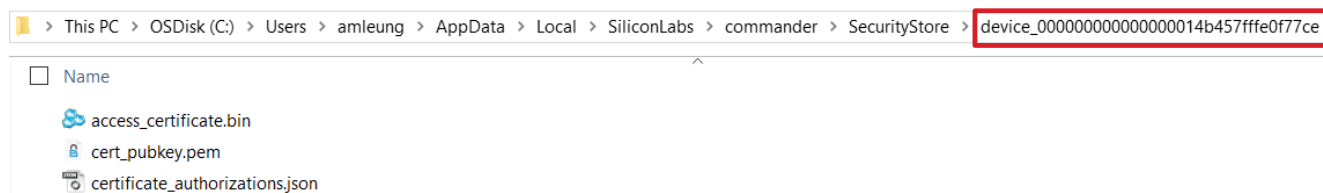
2. Run the `security gencert` command with device part number, device serial number (from the requesting party), and Public Certificate Key (`cert_pubkey.pem`) generated in step 1 to generate an unsigned [access certificate](#) (`access_certificate.bin`).

```
commander security gencert --device EFR32MG21B010F1024 --deviceserialno 00000000000000014b457fffe0f77ce --cert-pubkey cert_pubkey.pem
```

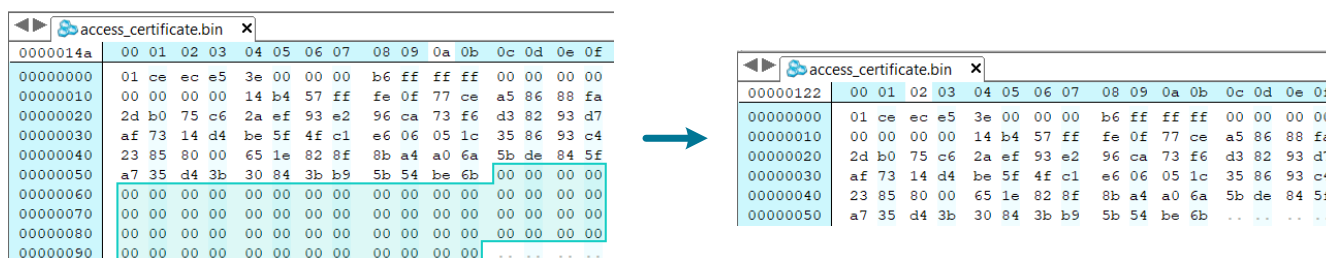
```
Authorization file written to Security Store:
C:/Users/amleung/AppData/Local/SiliconLabs/commander/SecurityStore/device_00000000000000014b457fffe0f77ce/
certificate_authorizations.json
Cert key written to Security Store:
C:/Users/amleung/AppData/Local/SiliconLabs/commander/SecurityStore/device_00000000000000014b457fffe0f77ce/
cert_pubkey.pem
Certificate was not signed
DONE
```

Note: The unsigned access certificate is generated with the default certificate authorization file (`certificate_authorization.js` on) which uses `0x0000003e` for Authorizations and `0xffffffb6` for Tamper Authorizations ([Table 9.3 Elements of the Access Certificate on page 16](#)).

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



4. Copy the unsigned access certificate file (`access_certificate.bin`), Private Command Key file (`command_key.pem`), and Public Command Key file (`command_pubkey.pem`) to the Simplicity Commander folder.
5. Open the `access_certificate.bin` file to remove the 64 bytes of `0x00` reserved for signature.



6. Use OpenSSL to sign the `access_certificate.bin` file with Private Command Key (`command_key.pem`). The certificate signature is in the `cert_signature.bin` file.

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

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

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

Verified OK

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

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

```
0:d=0 hl=2 l= 68 cons: SEQUENCE
2:d=1 hl=2 l= 32 prim: INTEGER   :9CB0F5712C66651CF43E9EC18E09201CB96D8BE03893E11227E2CA726BC7C355
36:d=1 hl=2 l= 32 prim: INTEGER   :485CBAE93B5E341CAFCCAB83A550376E8EC5B3FCEC4F4E5B1450E519779152E6
```

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

cert_signature.bin		cert_signature.bin	
00000186	00 01 02 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f	00000157	00 01 02 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f
00000000	30 45 02 21 00 9c b0 f5 71 2c 66 65 1c f4 3e 9e	00000000	9c b0 f5 71 2c 66 65 1c f4 3e 9e c1 8e 09 20 1c
00000010	c1 8e 09 20 1c b9 6d 8b e0 38 93 e1 12 27 e2 ca	00000010	b9 6d 8b e0 38 93 e1 12 27 e2 ca 72 6b c7 c3 55
00000020	72 6b c7 c3 55 02 20 48 5c ba e9 3b 5e 34 1c af	00000020	48 5c ba e9 3b 5e 34 1c af cc ab 83 a5 50 37 6e
00000030	cc ab 83 a5 50 37 6e 8e c5 b3 fc ec 4f 4e 5b 14	00000030	8e c5 b3 fc ec 4f 4e 5b 14 50 e5 19 77 91 52 e6
00000040	50 e5 19 77 91 52 e6		

10. Use OpenSSL to sign the command_unsign.bin file (from the requesting party) with the Private Certificate Key (cert_key.pem) generated in step 1. The disable tamper command signature is in the command_signature.bin file.

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

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

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

Verified OK

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

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

```
0:d=0 hl=2 l= 69 cons: SEQUENCE
2:d=1 hl=2 l= 33 prim: INTEGER   :D63002CC4FB501771CC334A8C01C148EE1A8AE50ED610140CEF5D80527CB1174
37:d=1 hl=2 l= 32 prim: INTEGER   :B23BBBA1B20F9862C21C629562D68915B86AFEC0BC70CE7E611D62BC07B8E3E3
```

13. Open the command_signature.bin file to remove the ASN.1 headers in signature.

command_signature.bin		command_signature.bin	
0000013b	00 01 02 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f	000000f6	00 01 02 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f
00000000	30 46 02 21 00 d6 30 02 cc 4f b5 01 77 1c c3 34	00000000	d6 30 02 cc 4f b5 01 77 1c c3 34 a8 c0 1c 14 8e
00000010	a8 c0 1c 14 8e e1 a8 ae 50 ed 61 01 40 ce f5 d8	00000010	e1 a8 ae 50 ed 61 01 40 ce f5 d8 05 27 cb 11 74
00000020	05 27 cb 11 74 02 21 00 b2 3b bb a1 b2 0f 98 62	00000020	b2 3b bb a1 b2 0f 98 62 c2 1c 62 95 62 d6 89 15
00000030	c2 1c 62 95 62 d6 89 15 b8 6a fe c0 bc 70 ce 7e	00000030	b8 6a fe c0 bc 70 ce 7e 61 1d 62 bc 07 b8 e3 e3
00000040	61 1d 62 bc 07 b8 e3 e3		

1. Run the `security gencommand` command with the `--disable-param` option to create the required folder in which to place the `disable tamper` command file.

```
Unsigned command file written to Security Store:
C:/Users/amleung/AppData/Local/SiliconLabs/commander/SecurityStore/device_00000000000000014b457fffe0f77ce/
challenge_fc3d2ab41c07562bd31e3a1542d6fbd5/disable_tamper_command_to_be_signed20_05_2020.bin
DONE
```

- [illegible]

- ```
Disabling tamper with tamper payload:
C:/Users/amleung/AppData/Local/SiliconLabs/commander/SecurityStore/device_00000000000000014b457fffe0f77ce/
challenge_fc3d2ab41c07562bd31e3a1542d6fdb5/tamper_payload_111110100000000000000000.bin
Tamper successfully disabled.
DONE
```

- ```
. Get tamper status... OK (cycles: 12049 time: 313 us)
+ Recorded tamper status (MSB..LSB): 00050000
+ Currently active tamper sources (MSB..LSB): 00050000
```

- ```
Disabling tamper with tamper payload:
C:/Users/amleung/AppData/Local/SiliconLabs/commander/SecurityStore/device_00000000000000014b457fffe0f77ce/
challenge_fc3d2ab41c07562bd31e3a1542d6fdb5/tamper_payload_111110100000000000000000.bin
Tamper successfully disabled.
DONE
```

### 10.4.3 Roll Challenge to Revoke Tamper Disable

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

```
commander security rollchallenge --device EFR32MG21B010F1024 --serialno 440068705
```

```
Challenge was rolled successfully.
DONE
```

2. Run the `security disabletamper` command to verify the current disable tamper command file is no longer valid.

```
commander security disabletamper --disable-param 0x00fa0000 --device EFR32MG21B010F1024 --serialno 440068705
```

```
Authorization file written to Security Store:
C:/Users/amleung/AppData/Local/SiliconLabs/commander/SecurityStore/device_00000000000000014b457fffe0f77ce/
certificate_authorizations.json
Generating ECC P256 key pair...
Cert public key stored at:
C:/Users/amleung/AppData/Local/SiliconLabs/commander/SecurityStore/device_00000000000000014b457fffe0f77ce/
cert_pubkey.pem
Cert private key stored at:
C:/Users/amleung/AppData/Local/SiliconLabs/commander/SecurityStore/device_00000000000000014b457fffe0f77ce/
cert_key.pem
Certificate was not signed
ERROR: Created an unsigned certificate. Please provide command key with option --command-key to sign the
certificate or provide the certificate signature with option --cert-signature
DONE
```



## 11. Revision History

### Revision 0.1

September 2020

- 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](http://www.silabs.com/IoT)



**SW/HW**

[www.silabs.com/simplicity](http://www.silabs.com/simplicity)



**Quality**

[www.silabs.com/quality](http://www.silabs.com/quality)



**Support & Community**

[www.silabs.com/community](http://www.silabs.com/community)

## Disclaimer

Silicon Labs intends to provide customers with the latest, accurate, and in-depth documentation of all peripherals and modules available for system and software implementers using or intending to use the Silicon Labs products. Characterization data, available modules and peripherals, memory sizes and memory addresses refer to each specific device, and "Typical" parameters provided can and do vary in different applications. Application examples described herein are for illustrative purposes only. Silicon Labs reserves the right to make changes without further notice to the product information, specifications, and descriptions herein, and does not give warranties as to the accuracy or completeness of the included information. Without prior notification, Silicon Labs may update product firmware during the manufacturing process for security or reliability reasons. Such changes will not alter the specifications or the performance of the product. Silicon Labs shall have no liability for the consequences of use of the information supplied in this document. This document does not imply or expressly grant any license to design or fabricate any integrated circuits. The products are not designed or authorized to be used within any FDA Class III devices, applications for which FDA premarket approval is required, or Life Support Systems without the specific written consent of Silicon Labs. A "Life Support System" is any product or system intended to support or sustain life and/or health, which, if it fails, can be reasonably expected to result in significant personal injury or death. Silicon Labs products are not designed or authorized for military applications. Silicon Labs products shall under no circumstances be used in weapons of mass destruction including (but not limited to) nuclear, biological or chemical weapons, or missiles capable of delivering such weapons. Silicon Labs disclaims all express and implied warranties and shall not be responsible or liable for any injuries or damages related to use of a Silicon Labs product in such unauthorized applications.

## Trademark Information

Silicon Laboratories Inc.®, Silicon Laboratories®, Silicon Labs®, SiLabs® and the Silicon Labs logo®, Bluegiga®, Bluegiga Logo®, ClockBuilder®, CMEMS®, DSPLL®, EFM®, EFM32®, EFR, Ember®, Energy Micro, Energy Micro logo and combinations thereof, "the world's most energy friendly microcontrollers", Ember®, EZLink®, EZRadio®, EZRadioPRO®, Gecko®, Gecko OS, Gecko OS Studio, ISOModem®, Precision32®, ProSLIC®, Simplicity Studio®, SiPHY®, Telegesis, the Telegesis Logo®, USBXpress®, Zentri, the Zentri logo and Zentri DMS, Z-Wave®, and others are trademarks or registered trademarks of Silicon Labs. ARM, CORTEX, Cortex-M3 and THUMB are trademarks or registered trademarks of ARM Holdings. Keil is a registered trademark of ARM Limited. Wi-Fi is a registered trademark of the Wi-Fi Alliance. All other products or brand names mentioned herein are trademarks of their respective holders.



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

<http://www.silabs.com>