AN1190: Series 2 Secure Debug

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

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

KEY POINTS

- Basic overview of the Secure Element.
- Debug port access by Debug Challenge Interface (DCI) or Mailbox Interface.
- New locking and unlocking features for Series 2 devices.
- Examples for Public Command Key provisioning and Secure Debug Unlock.

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 (this document)	**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	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.

Key Name	SE Manager ID	Customer Programmed	Purpose	Used in
Public Sign key (Sign Key Public)	SL_SE_KEY_SLOT_APPLICATION_SECURE_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_SECURE_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, and some functionality is different depending on the device.

Wireless SoC Series 2 families consist of:

- EFR32BG21A/EFR32BG21B/EFR32BG22
- EFR32FG22
- EFR32MG21A/EFR32MG21B/EFR32MG22

3. Introduction to Secure Debug

3.1 Debug Lock

All devices require the capability to lock out debug access to the device. This prevents attackers from using the debug interface to perform the following illegal operations:

- Reprogramming the device
- Interrogating the device
- Interfering with the operation of the device

Three different locks can be put on the Series 2 debug interface:

- Standard debug lock
- Permanent debug lock
- Secure debug lock

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

3.2 Debug Unlock

Users need to unlock parts under a number of circumstances:

- Code development
- Field failure diagnosis
- Product field service
- Existing inventory reprogramming

Two different unlocks can run on the Series 2 debug interface:

- Standard debug unlock
- Secure debug unlock

4. Secure Element Subsystem

4.1 Overview

On Series 2 devices, the Secure Debug feature is implemented by the Secure Element. The Secure Element may be hardware-based, or virtual (software). If hardware-based, the implementation may be either with or without Secure Vault. Throughout this document, the following conventions will be used.

- SE - Hardware Secure Element, either with or without Secure Vault if not specified
- VSE - Virtual Secure Element
- Secure Element - Either SE or VSE

The SE refers to a separate security co-processor that provides hardware isolation between security functions and the host processor.

The VSE refers to a collection of security functions available to the host processor in Root mode if a separate security co-processor is not provided.

The Secure Element is used to perform a series of cryptographic operations and other secure system operations (Table 4.1 Secure Element Operations on page 6).

Table 4.1. Secure Element Operations

Operation	VSE	SE without Secure Vault	SE with Secure Vault	Description
Unique ID	Y	Y	Y	Software can identify every device.
Secure Boot with RTSL	Y	Y	Y	Only boot authenticated firmware.
Secure Debug	Y	Y	Y	Allow enhanced failure analysis.
Crypto Engine[1]	—	Y	Y	Up to 256-bit ciphers and elliptic curves.
TRNG[1]	—	Y	Y	Generate keys for proper cryptography.
DPA Countermeasures	—	Y	Y	Resist side channel attacks.
Secure Key Storage	—	—	Y	Protected by PUF technology.
Secure Key Management	—	—	Y	Isolate encrypted keys from application code.
Secure Attestation	—	—	Y	Ensure integrity and authenticity.
Anti-Tamper	—	—	Y	Detect tamper and protect keys/data.
Advanced Crypto	—	—	Y	Up to 512-bit ciphers and 521-bit elliptic curves.

Note:

1. On VSE devices, the crypto engine and TRNG (True Random Number Generation) are implemented by the CRYPTOACC (Cryptographic Accelerator) peripheral.

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

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

The Secure Debug feature is implemented by Root code executed by the SE Core or by the Cortex-M33 operating in VSE (Root mode). Table 4.2 Minimum Secure Element Firmware Version for Secure Debug on page 7 indicates the minimum required Secure Element Root code versions that support Secure Debug.

Table 4.2. Minimum Secure Element Firmware Version for Secure Debug

Device	Secure Element	Minimum Firmware Version for Secure Debug
EFR32xG21A	SE without Secure Vault	Version 1.1.2
EFR32xG21B	SE with Secure Vault	Version 1.2.1
EFR32xG22	VSE	Version 1.1.7
Note: Silicon Labs strongly recommends installing the latest Secure Element firmware on Series 2 devices.		

4.2 Command Interface

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

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

4.2.1 Mailbox

Mailbox operations should not be performed directly, but rather should be executed through either `mbed TLS` for cryptographic operations in SE or the appropriate functions in `em_se.c` of `emlib`.

The `em_se.c` provides an abstraction of the mailbox interface, allowing message construction and DMA transfer set up.

On top of `emlib`, the Secure Element Manager provides an abstraction of the Secure Element's command set. The Secure Element Manager also provides APIs for cryptographic operations and thread synchronization. The Secure Element Manager is available in **GSDK v3.0** or later.

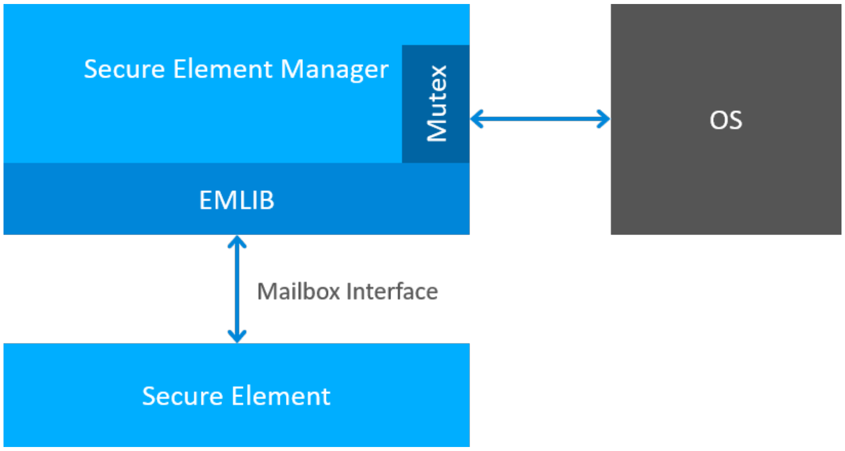


Figure 4.1. Emlib and Secure Element Manager

Note: Some functions in `em_se.c` of `emlib` are deprecated in **GSDK v3.0** and will be removed in a future version of `emlib`. All high-level functionality has been moved to the Secure Element Manager.

4.2.2 Debug Challenge Interface (DCI)

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

5. Debug Lock

5.1 Overview

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

Table 5.1. Debug Lock Properties

Property	Description If Set	Default Value
Debug Lock	The debug port is kept locked on boot.	False (Disabled)
Device Erase	The Erase Device command is available.	True (Enabled)
Secure Debug	Secure debug unlock is available.	False (Disabled)

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

5.2 Standard Debug Lock

With the default properties (Table 5.2 Standard Debug Lock on page 8) of the debug lock set, the device can be locked using the Apply Lock command. Typical flow for this configuration is simply to issue the `Apply Lock` command after the device has been programmed, either using a DCI command from the programming debugger (7.2 Standard Debug Lock and Unlock) or through the mailbox interface.

Table 5.2. Standard Debug Lock

Secure Debug	Device Erase	Debug Lock	Description
Disabled	Enabled	Standard	The Erase Device command will wipe the main Flash and RAM, and then a reset will yield an unlocked device.

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

5.3 Permanent Debug Lock

The Erase Device command can be disabled, which permanently enables the debug lock. This can be done at any time by issuing the Disable Device Erase command, even after the debug lock has been enabled.

Table 5.3. Permanent Debug Lock

Secure Debug	Device Erase	Debug Lock	Description
Disabled	Disabled	Permanent	The part cannot be unlocked. Devices with Permanent Debug Lock engaged cannot be returned for failure analysis.

5.4 Secure Debug Lock

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

Table 5.4. Secure Debug Lock

Secure Debug	Device Erase	Debug Lock	Description
Enabled[1]	Disabled[2]	Secure	Secure debug unlock is enabled, which makes it possible to securely open the debug lock temporarily to reprogram or debug a locked device.

Note:

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

5.5 Debug Lock Command Reference

The commands for debug lock are described in Table 5.5 Debug Lock Command Reference on page 10.

Table 5.5. Debug Lock Command Reference

DCI Command[1]	Mailbox API[2]	Description	Availability
`Apply Lock`	• `SE_debugLockApply` • `sl_se_apply_debug_lock`	Enables the debug lock for the part.	While debug is unlocked.
`Read Lock Status`	• `SE_debugLockStatus` • `sl_se_get_debug_lock_status`	Returns the current debug lock status and configuration.	Always.
`Disable Device Erase`	• `SE_deviceEraseDisable` • `sl_se_disable_device_erase`	Disables the Erase Device command. This command does not lock the debug interface to the part, but it is an **IRREVERSIBLE** action for the part.	While debug is unlocked.
`Disable Secure Debug`	• `SE_debugSecureDisable` • `sl_se_disable_secure_debug`	Disables the secure debug functionality that can be used to open a locked debug port.	While secure debug is enabled.
`Enable Secure Debug`	• `SE_debugSecureEnable` • `sl_se_enable_secure_debug`	Enables the secure debug functionality that can be used to open a locked debug port.	While debug is unlocked and Public Command Key is uploaded.
`Init Pub Key`	• `SE_initPubkey` • `sl_se_init_otp_key`[3]	Used during device initialization to upload a single public key. After a key has been written it cannot be changed, and the command will be unavailable for that key.	Available once for each key.
`Read Pub Key`	• `SE_readPubkey` • `sl_se_read_pubkey`[3]	Reads the stored public key.	Always.
`Get Challenge`[4]	`sl_se_roll_challenge`	Used to roll the current challenge value (16 bytes) to revoke secure debug access.	While Public Command Key is uploaded.

Note:

1. Performing these commands over DCI is implemented in Simplicity Studio and Simplicity Commander.
2. The APIs with an `SE_` prefix are for `emlib` whereas APIs with an `sl_se_` prefix are for Secure Element Manager.
3. The `sl_se_init_otp_key` and `sl_se_read_pubkey` are available on all Series 2 devices. Other APIs are only available on Series 2 devices with SE.
4. A new challenge will only be generated if the current one has been successfully used at least once.

6. Debug Unlock

6.1 Overview

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

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

The status of the debug lock can be inspected using the Read Lock Status command.

6.2 Standard Debug Unlock

With the default properties (Table 5.2 Standard Debug Lock on page 8) of the debug lock set, the device can be unlocked using the Erase Device command. This command will wipe main Flash and RAM and verify they are empty before opening the debug lock. It will not wipe user data and provisioned Secure Element settings.

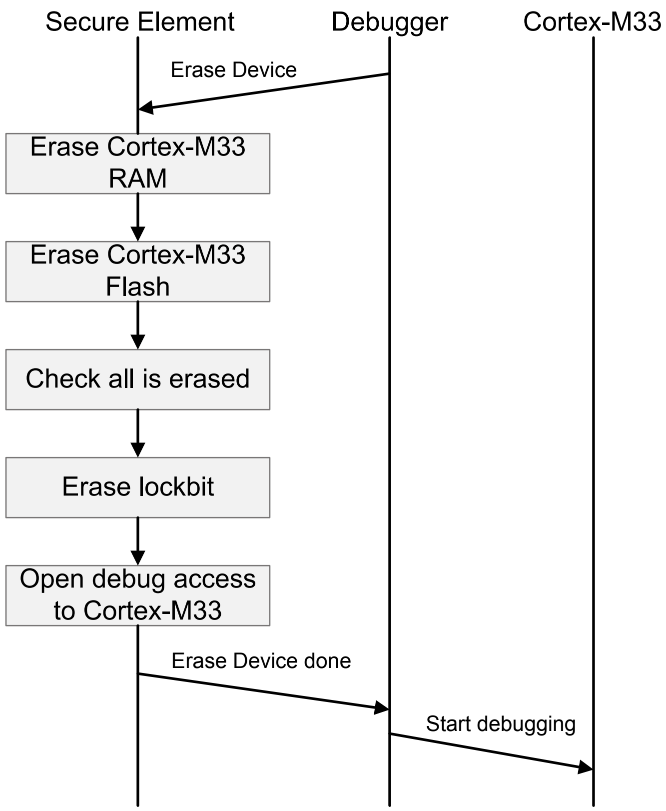


Figure 6.1. Standard Debug Unlock

6.3 Secure Debug Unlock

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

This mode is particularly useful in failure analysis scenarios because it allows devices to be unlocked without losing flash and RAM contents.

6.3.1 Debug Access Command

The elements of the debug access command are described in Figure 6.2 Debug Access Command on page 12 and Table 6.1 Elements of Debug Access Command on page 12.

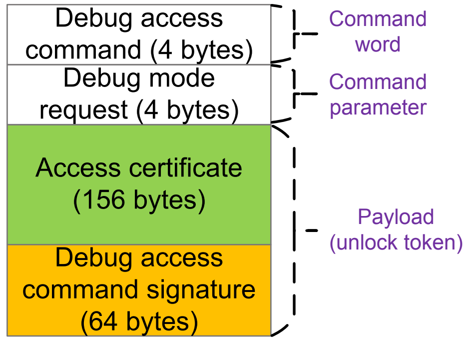


Figure 6.2. Debug Access Command

Table 6.1. Elements of Debug Access Command

Element	Value	Description
Debug access command	`0xfd010001`	The command word of the debug access command.
Debug mode request	Device-dependent	The command parameter of the debug access command (Table 6.2 Debug Mode Request on page 12).
Access certificate[1]	Device-dependent	See 6.3.2 Access Certificate.
Debug access command signature[1]	Device-dependent	See 6.3.3 Challenge Response.

Note:

1. The debug access command payload (unlock token) consists of an access certificate and a debug access command signature.

Table 6.2. Debug Mode Request

Debug Mode Request																																
Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Name	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	SPNIDLOCK	SPIDLOCK	NIDLOCK	DBGLOCK	Enable debug port	Reserved

Note:

- Enable debug port - Allow debug access if set.
- DBGLOCK - Non-Secure, Invasive debug access enabled if set.
- NIDLOCK - Non-Secure, Non-Invasive debug access enabled if set.
- SPIDLOCK - Secure, Invasive debug access enabled if set.
- SPNIDLOCK - Secure, Non-Invasive debug access enabled if set.
- All reserved bits should be 0. In general, bits 1 to 5 are set (`0x0000003e`) for full debug access.

6.3.2 Access Certificate

The elements of the access certificate are described in 6.3.2 Access Certificate and Table 6.3 Elements of the Access Certificate on page 13.

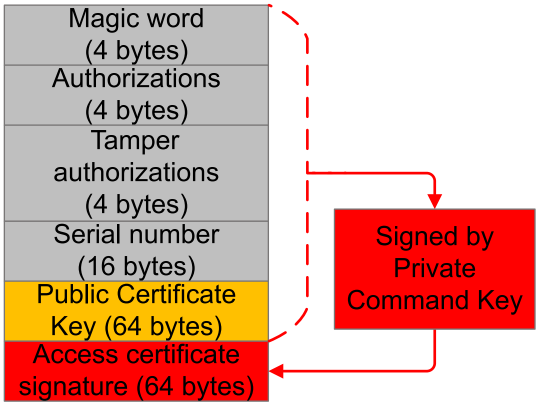


Figure 6.3. Access Certificate

Table 6.3. 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 the debug mode request can be enabled (Table 6.4 Authorizations on page 14) for secure debug.
Tamper Authorizations	• `0x00000000` or • `0xffffffb6`[2]	A value used to authorize which bit in the tamper disable mask can be enabled to disable the tamper response ("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 a 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 the Secure Element OTP.

Note:

1. Value that allows full debug access for secure debug.
2. Value that enables available bits in the tamper disable mask for tamper disable (used with SE with Secure Vault).
3. The Private/Public Certificate Key is a randomly generated keypair. 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 tamper disable, see section "*Tamper Disable*" in AN1247: Anti-Tamper Protection Configuration and Use.

Table 6.4. Authorizations

Authorizations																																
Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Name	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	SPNIDLOCK request mask	SPIDLOCK request mask	NIDLOCK request mask	DBGLOCK request mask	Enable debug port request mask	Reserved

Note:
- Set the bit to enable the corresponding bit in the debug mode request.
- The debug access command will authorize a corresponding debug mode if the same bit is set in Table 6.2 Debug Mode Request on page 12 and Table 6.4 Authorizations on page 14.

6.3.3 Challenge Response

The elements of the challenge response are described in Figure 6.4 Challenge Response on page 15 and Table 6.5 Elements of the Challenge Response on page 15.

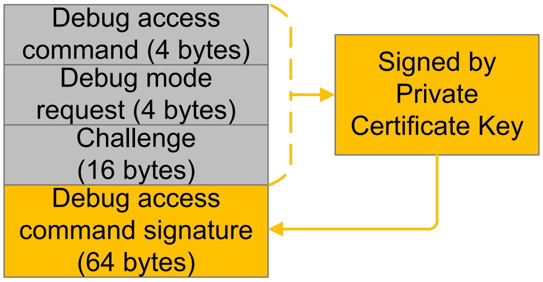


Figure 6.4. Challenge Response

Table 6.5. Elements of the Challenge Response

Element	Value	Description
Debug access command	`0xfd010001`	The command word of the debug access command.
Debug mode request	Device-dependent	The command parameter of the debug access command.
Challenge	Device-dependent[1]	A random value generated by the Secure Element.
Debug access 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:

1. 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 the device challenge is refreshed.
2. This signature is the final argument of the debug access command.

6.3.4 Debug Access Flow

The debug access flow is described in Figure 6.5 Debug Access Flow on page 16.

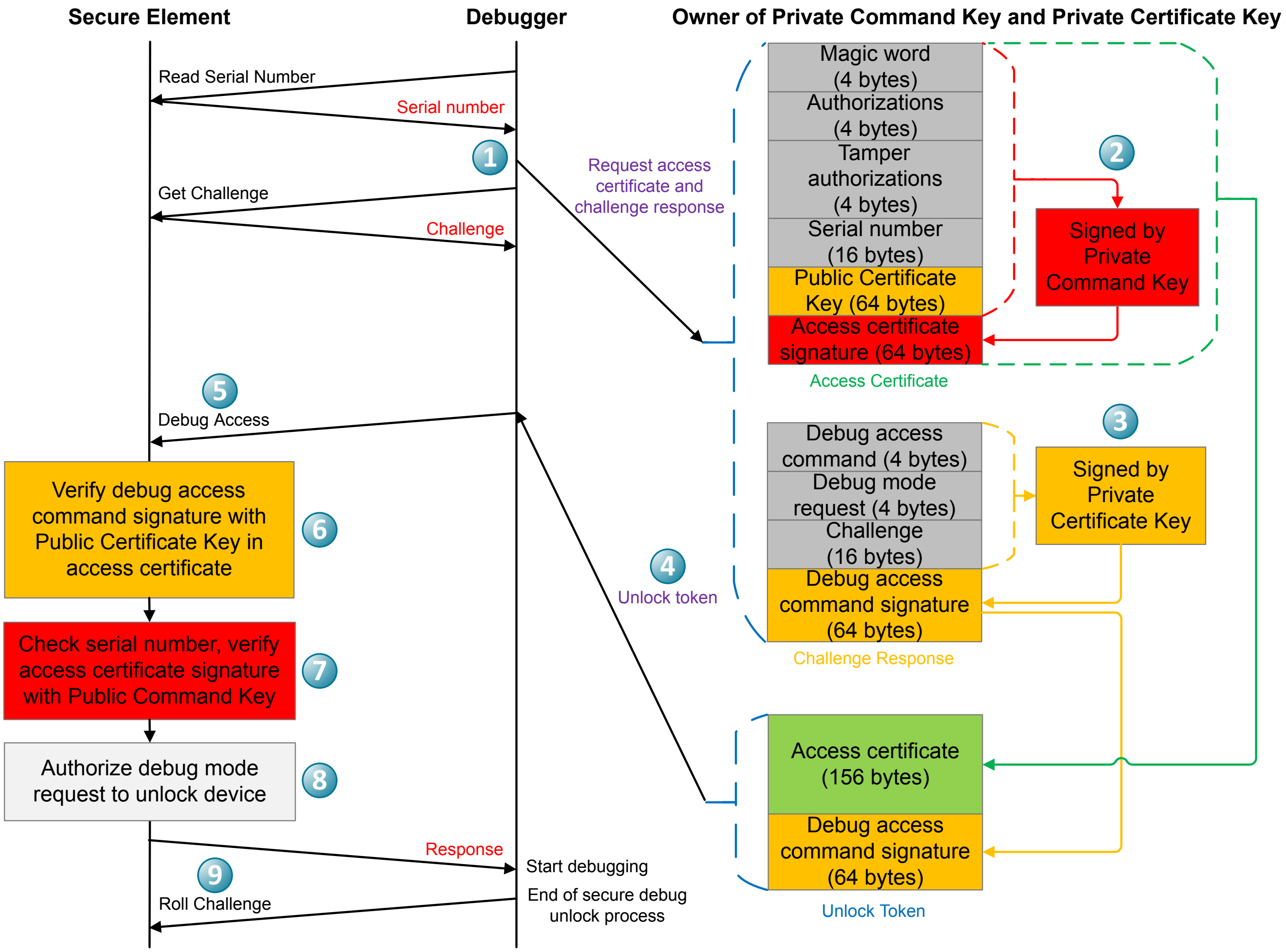


Figure 6.5. Debug Access Flow

1. Get the serial number and challenge from the Secure Element.
2. Generate the access certificate with the device serial number.
3. Generate the challenge response with device challenge.
4. Generate the unlock token (payload of debug access command) with access certificate and debug access command signature.
5. Send the debug access command to the Secure Element.
6. Verify the debug access 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 Secure Element OTP.
8. Authorize the debug mode request to unlock the device until the next power-on or pin reset.
9. Roll the challenge to invalidate the current debug access command.

6.4 Debug Unlock Command Reference

The commands for debug unlock are described in Table 6.6 Debug Unlock Command Reference on page 17.

Table 6.6. Debug Unlock Command Reference

DCI Command[1]	Mailbox API[2]	Description	Availability
`Erase Device`	• `SE_deviceErase` • `sl_se_erase_device`	Performs a device mass erase and resets the debug configuration to its initial unlocked state.	While Device Erase is enabled.
`Read Serial Number`	• `SE_serialNumber` • `sl_se_get_serialnumber`	Reads out the serial number (16 bytes) of the Series 2 device.	Always.
`Get Challenge`	`sl_se_get_challenge`	Reads out the current challenge value (16 bytes) for Secure debug unlock.	While Public Command Key is uploaded.
`Debug Access`	`sl_se_open_debug`	Opens the secure debug access of the Cortex-M33.	Only when Secure Debug is enabled.

Note:

1. Performing these commands over DCI is implemented in Simplicity Studio and Simplicity Commander.
2. The APIs with an `SE_` prefix are for `emlib`, whereas APIs with an `sl_se_` prefix are for Secure Element Manager. These APIs are only available on Series 2 devices with SE.

7. Examples

7.1 Overview

The examples for Series 2 Secure Debug are described in Table 7.1 Secure Debug Examples on page 18.

Table 7.1. Secure Debug Examples

Example	Device	Radio Board	SE or VSE Firmware	Tool
Standard debug lock and unlock[1]	EFR32MG21A010F1024IM32	BRD4181A	Version 1.2.1	Simplicity Studio
	EFR32MG22C224F512IM40	BRD4182A	Version 1.2.1	Simplicity Commander
Provision Public Command Key	EFR32MG21A010F1024IM32	BRD4181A	Version 1.2.1	Simplicity Studio
	EFR32MG22C224F512IM40	BRD4182A	Version 1.2.1	Simplicity Commander
Secure debug unlock[2]	EFR32MG21B010F1024IM32	BRD4181C	Version 1.2.1	Secure Element Manager[3]
	EFR32MG21A010F1024IM32	BRD4181A	Version 1.2.1	Simplicity Commander

Note:
1. The standard debug lock and unlock examples may be used on devices with default Debug Lock Properties (Table 5.2 Standard Debug Lock on page 8).
2. The secure debug unlock example may be used on devices with Secure Debug Lock (Table 5.4 Secure Debug Lock on page 9).
3. The Secure Element Manager example can only run on Series 2 device with SE.

7.1.1 Using Simplicity Studio

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

1. Right-click the selected debug adapter **Radio Board (ID:J-Link serial number)** to display the context menu.

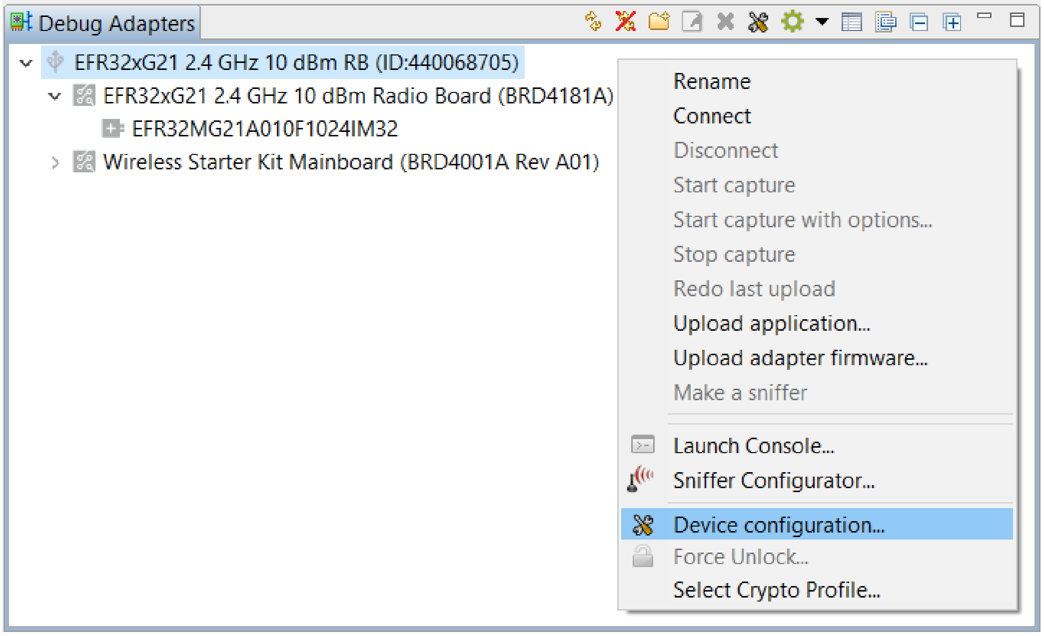


Figure 7.1. Debug Adapter Context Menu

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

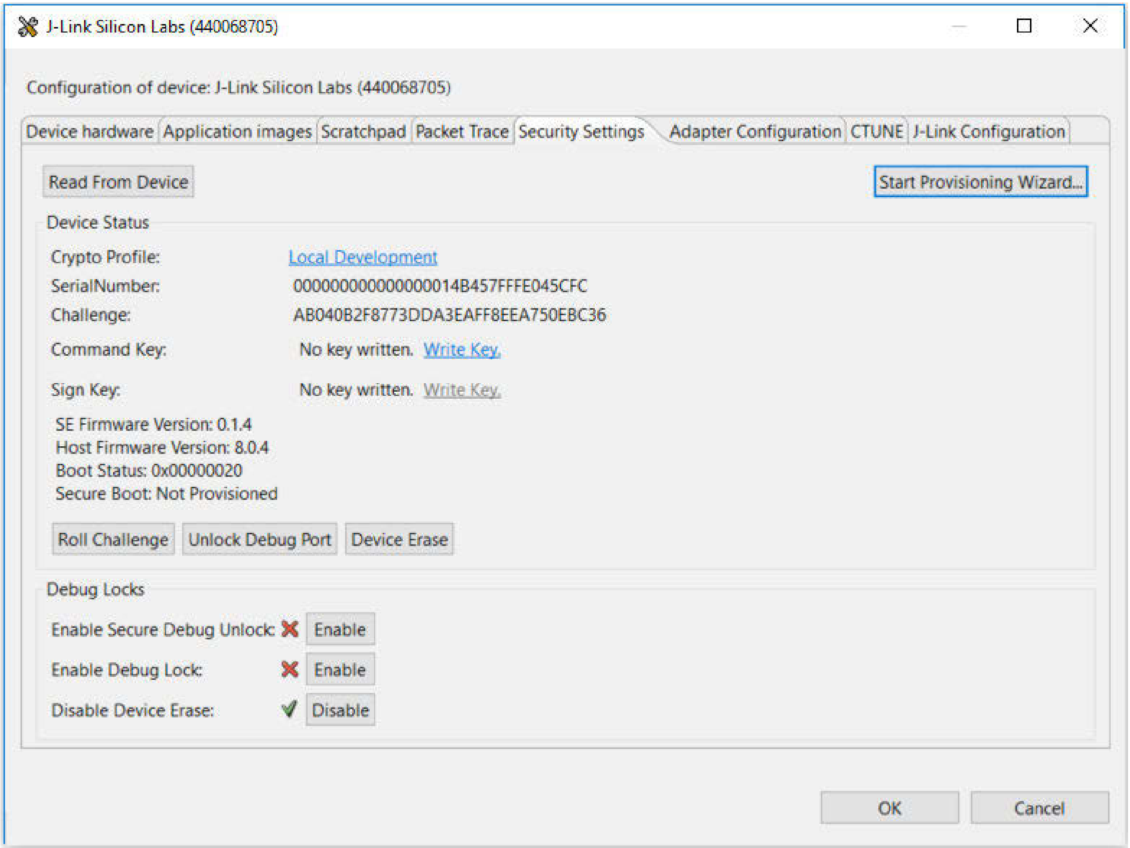


Figure 7.2. Configuration on Selected Device

7.1.2 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 WSTK is connected via USB, the target Wireless Starter Kit (WSTK) must be specified using the `--serialno <J-Link serial number>` option.
4. If the WSTK is in debug mode OUT, the target device must be specified using the `--device <device name>` option
5. Run the `security genkey` command to generate the Private/Public Command Key pair (`command_key.pem` and `command_pubkey.pem`) for secure debug examples.

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

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

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

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

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

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

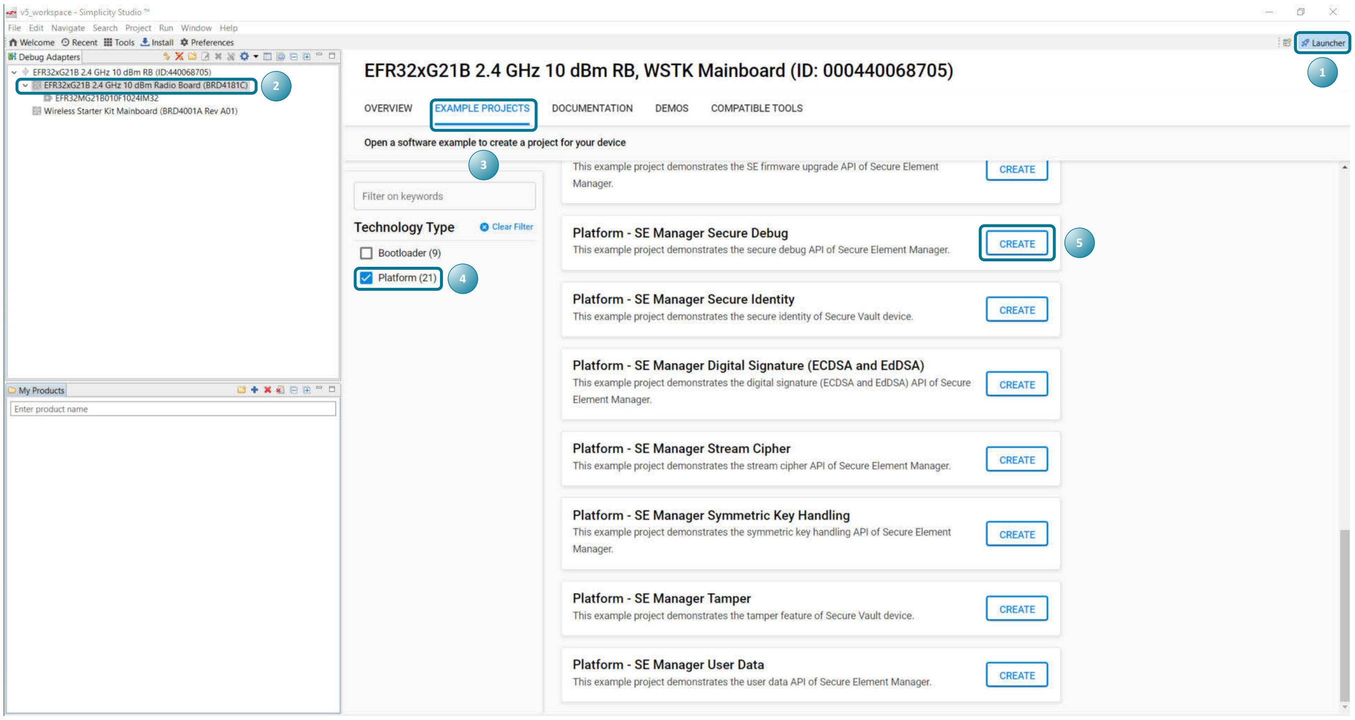
7.1.3 Using External Tools

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

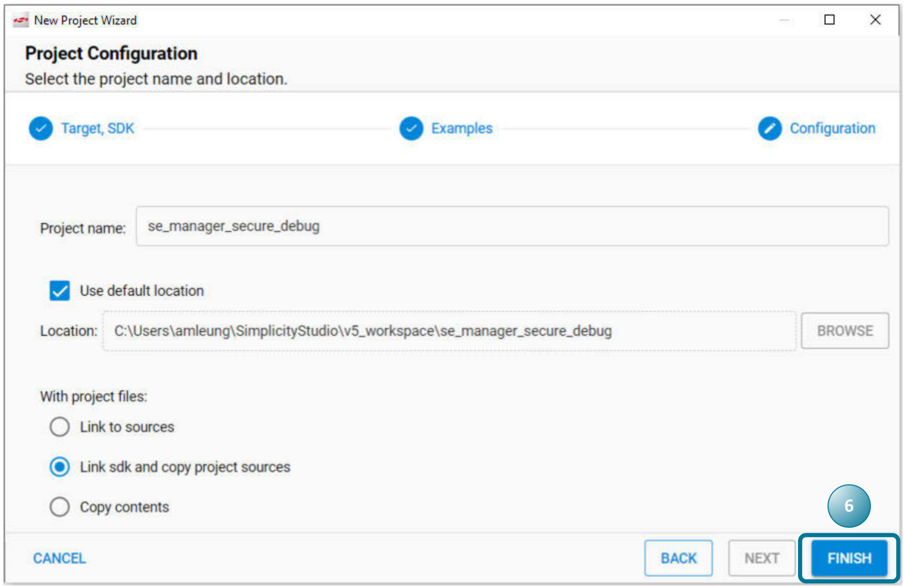
7.1.4 Using a Platform Example

This section describes how to build the secure debug platform example provided in Simplicity Studio 5, and program it to the Wireless Starter Kit (WSTK).

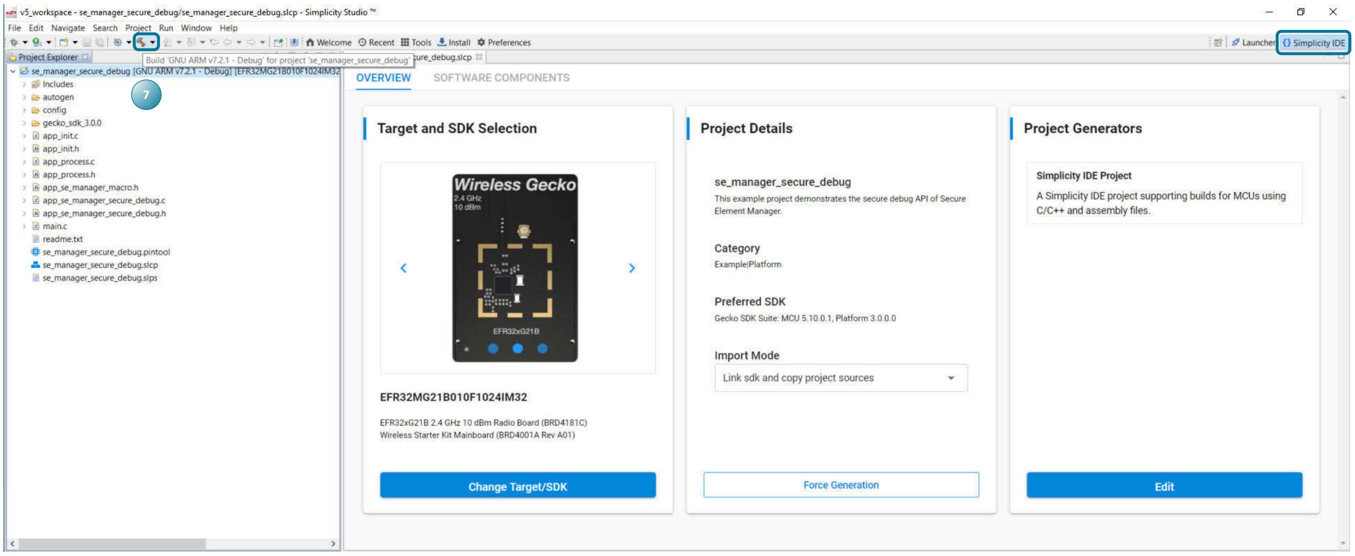
1. The connected WSTK displays in the [**Debug Adapters**] view in the [**Launcher**] perspective.Click the target radio board (**BRD4181C** in this example). This automatically configures the task bars for use with your device.
2. From the [**Launcher**] perspective, click [**EXAMPLE PROJECTS**].
3.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



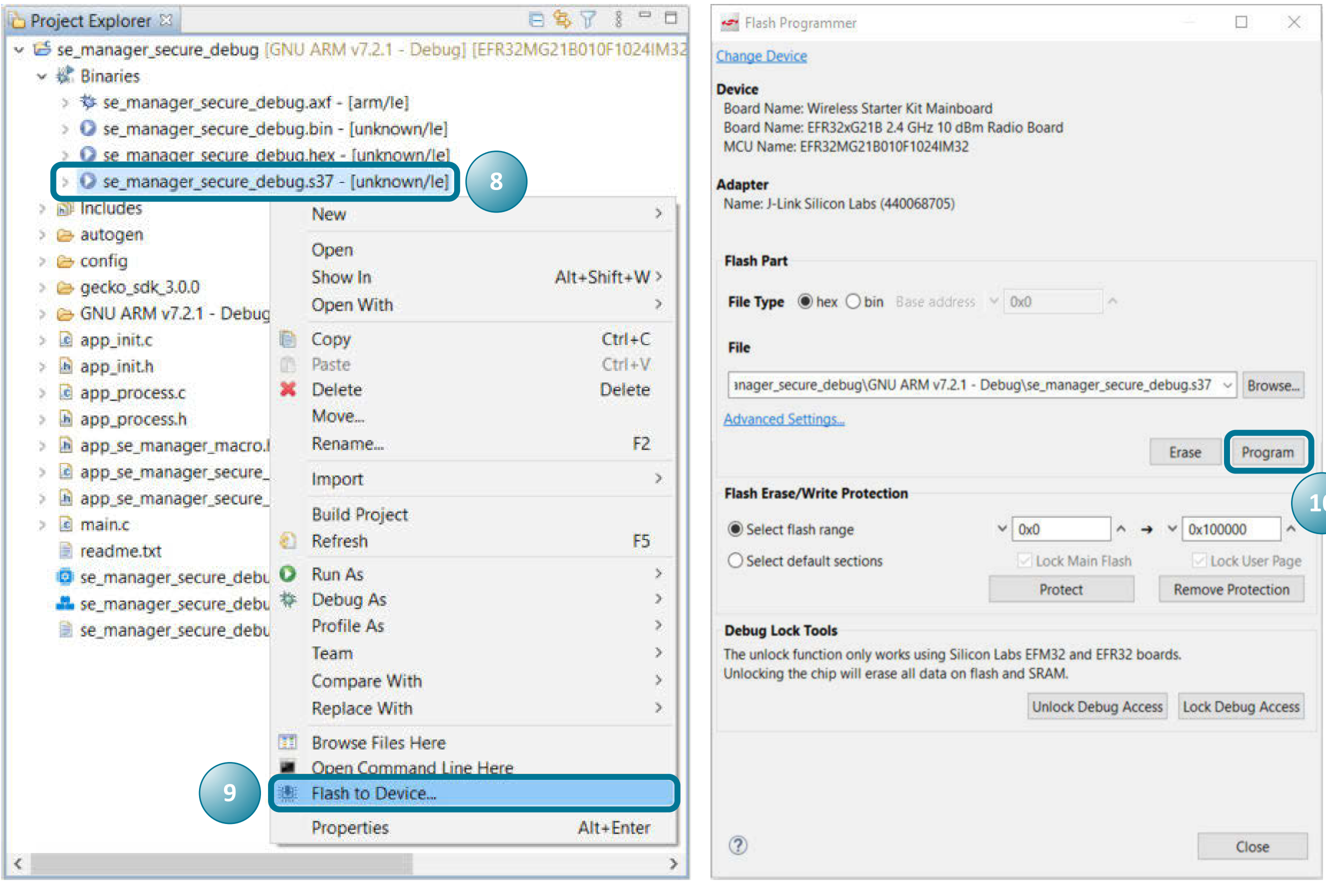
5. In the [**Project Configuration**] dialog, optionally name your project and select a different project location. Click [**FINISH**]. The [**Simplicity IDE**] perspective opens.



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



7. In the Project Explorer view, under **Binaries**, right-click the `se_manager_secure_debug.s37` file.
8. In the resulting context menu, click [**Flash to Device...**]. This opens the **Flash Programmer**.
9. Click [**Program**] to flash the `se_manager_secure_debug.s37` file to the radio board.



7.2 Standard Debug Lock and Unlock

7.2.1 Simplicity Studio

1. Open the **Security Settings** of the selected device as described in 7.1.1 Using Simplicity Studio.
2. Click [**Enable**] next to **Enable Debug Lock:** to lock the device. The following **Enable Debug Lock Warning** is displayed. Click [**Yes**] to confirm. This configures standard debug lock.

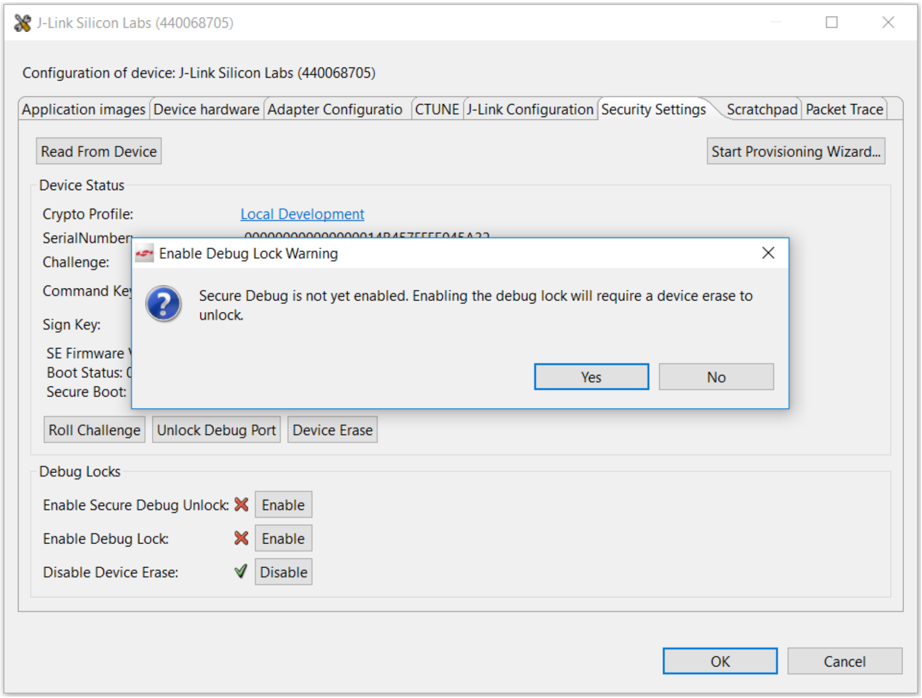


Figure 7.3. Enable Debug Lock

The [**Enable**] controls next to **Enable Secure Debug Unlock:** and **Enable Debug Lock:** are grayed out after standard debug lock is enabled.

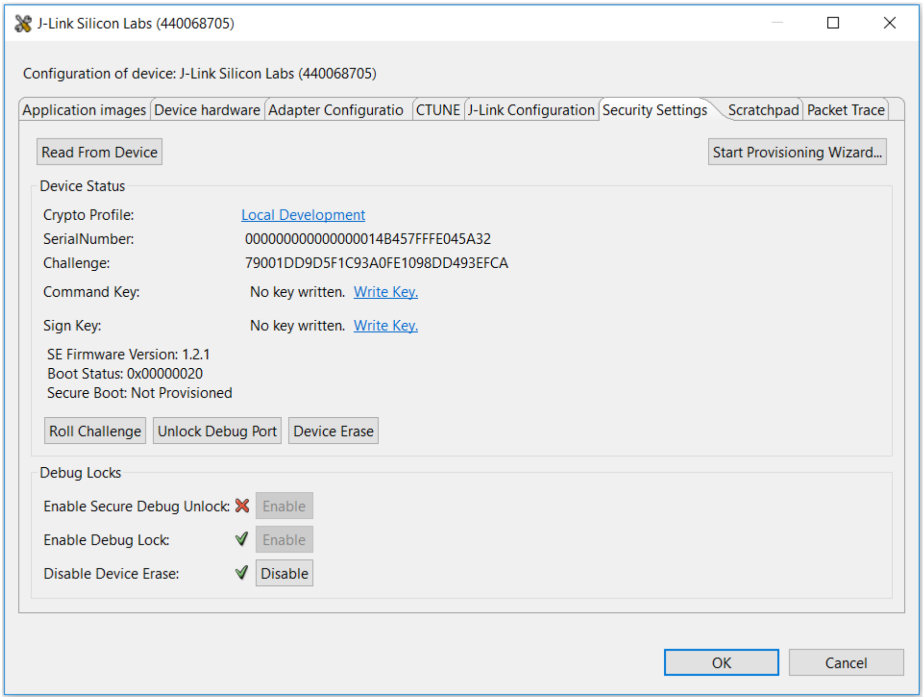


Figure 7.4. Standard Debug Lock

3. Click [**Device Erase**] to unlock the device.

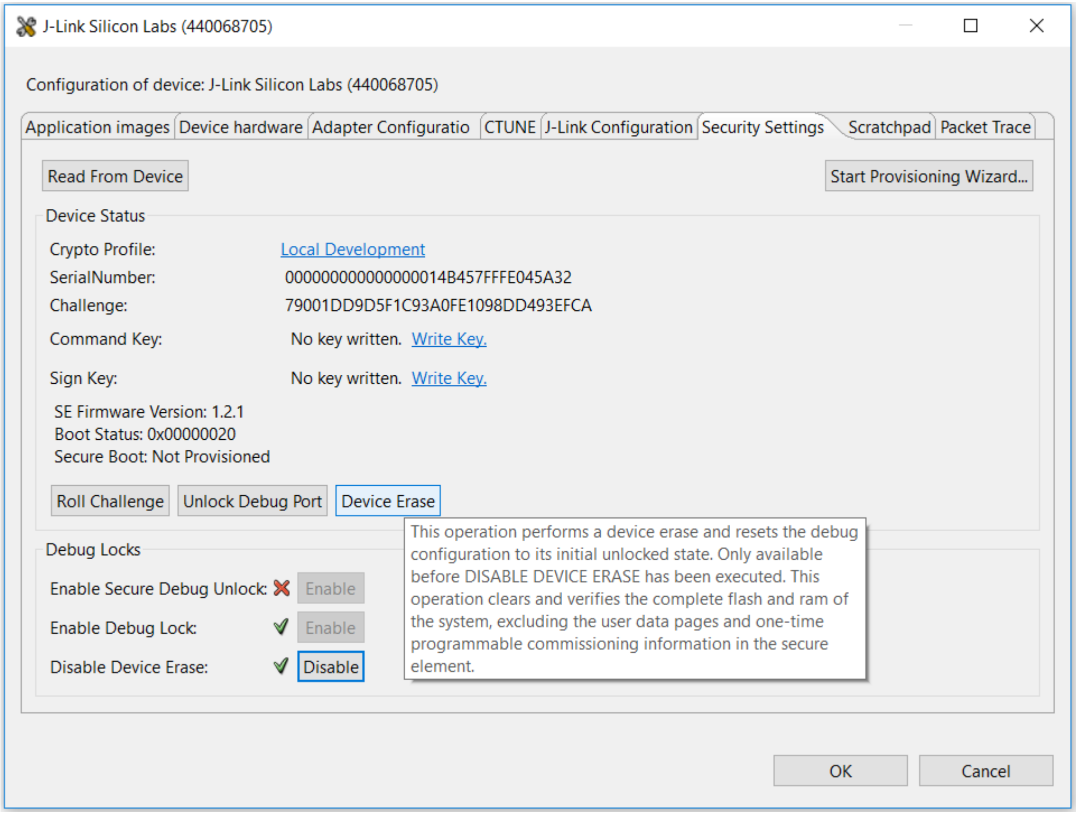


Figure 7.5. Device Erase

4. The device will return to the unlock state. Click [**OK**] to exit.

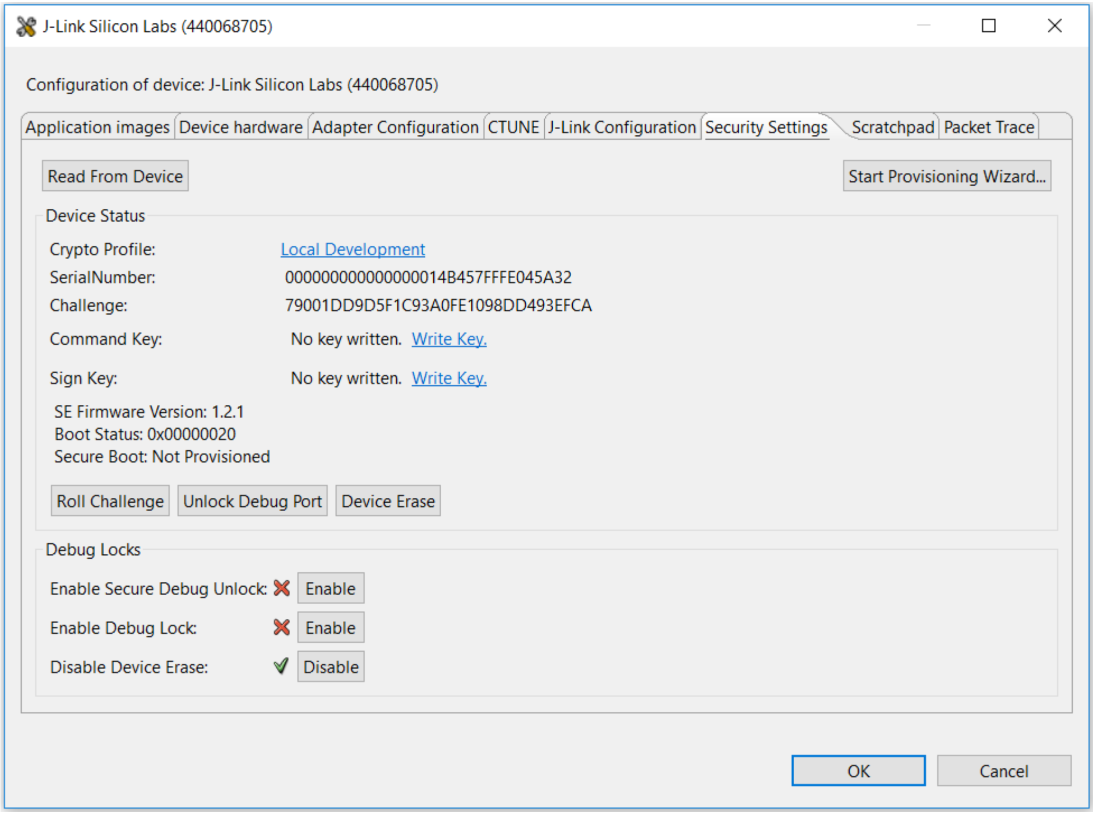


Figure 7.6. Standard Debug Unlock

7.2.2 Simplicity Commander

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

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

```
SE Firmware version : 1.2.1
Serial number       : 0000000000000014b457fffed50d1e
Debug lock          : Disabled
Device erase        : Enabled
Secure debug unlock : Disabled
Secure boot         : Disabled
Boot status         : 0x20 - OK
DONE
```

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

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

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

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

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

```
SE Firmware version : 1.2.1
Serial number       : 0000000000000014b457fffed50d1e
Debug lock          : Enabled
Device erase        : Enabled
Secure debug unlock : Disabled
Secure boot         : Disabled
Boot status         : 0x20 - OK
DONE
```

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

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

```
Successfully erased device
DONE
```

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

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

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

```
SE Firmware version : 1.2.1
Serial number       : 0000000000000014b457fffed50d1e
Debug lock          : Disabled
Device erase        : Enabled
Secure debug unlock : Disabled
Secure boot         : Disabled
Boot status         : 0x20 - OK
DONE
```

7.3 Provision Public Command Key

7.3.1 Simplicity Studio

1. Open **Security Settings** of the selected device as described in 7.1.1 Using Simplicity Studio.
2. Click [**Start Provisioning Wizard…**] in the upper right corner to display the **Secure Initialization** dialog box.

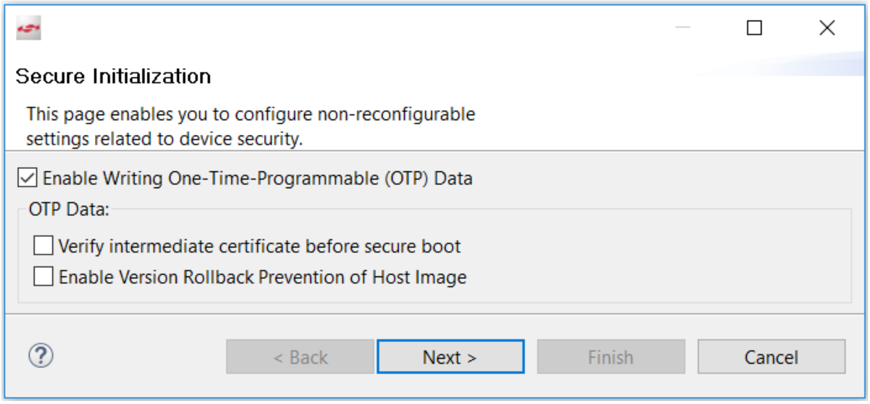


Figure 7.7. Secure Initialization Dialog Box

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

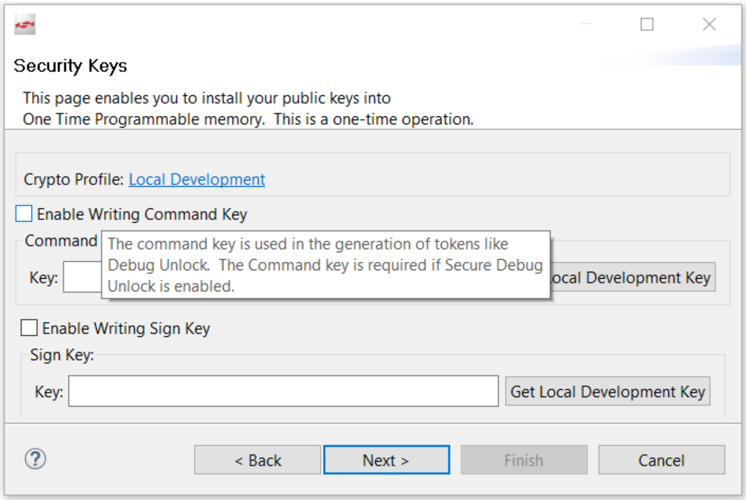


Figure 7.8. Security Keys Dialog Box

4. Open the `command_pubkey.txt` file generated in 7.1.2 Using Simplicity Commander step 6.

```
MFG_SIGNED_BOOTLOADER_KEY_X : F9017F10631575642D7ACF0CCCDB2461DD759923E3B28849EE044AA318112240
MFG_SIGNED_BOOTLOADER_KEY_Y : 0CB4EE5FA74AEEFBC0354B6A4881158741120B0005B6F309E3BFCAC63B898120
```

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

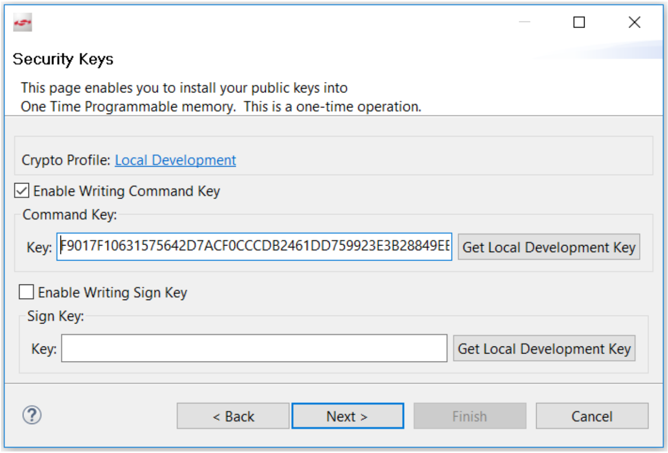


Figure 7.9. Public Command Key

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

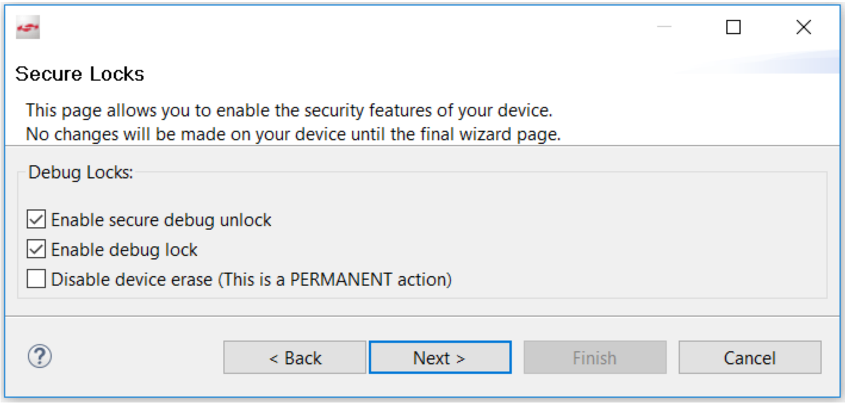


Figure 7.10. Security Locks Dialog Box

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

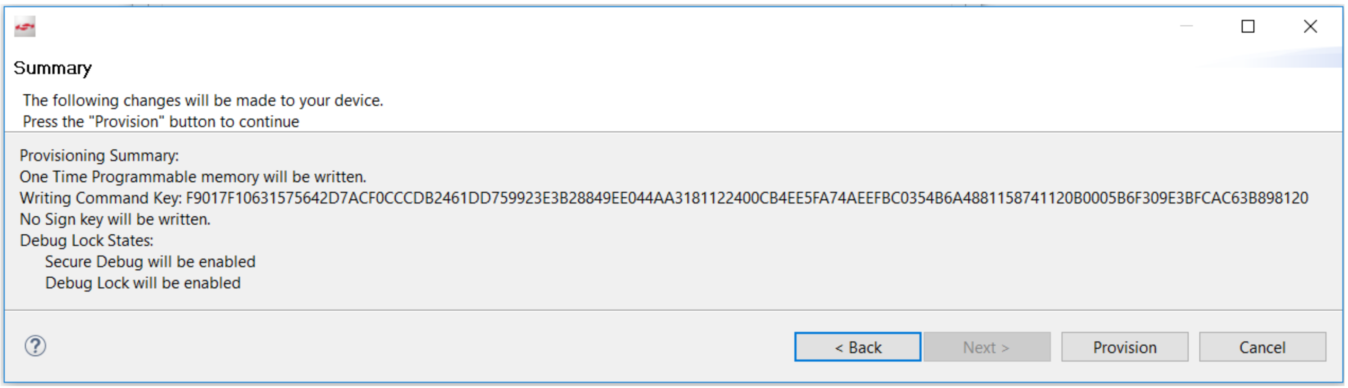


Figure 7.11. Summary Dialog Box

9. If the information displayed is correct, click [**Provision**]. Click [**Yes**] to confirm.

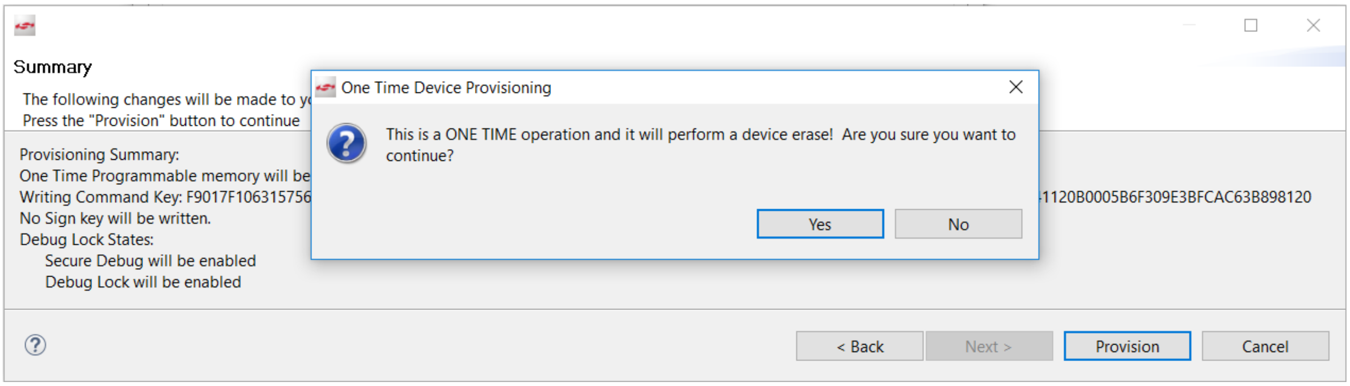
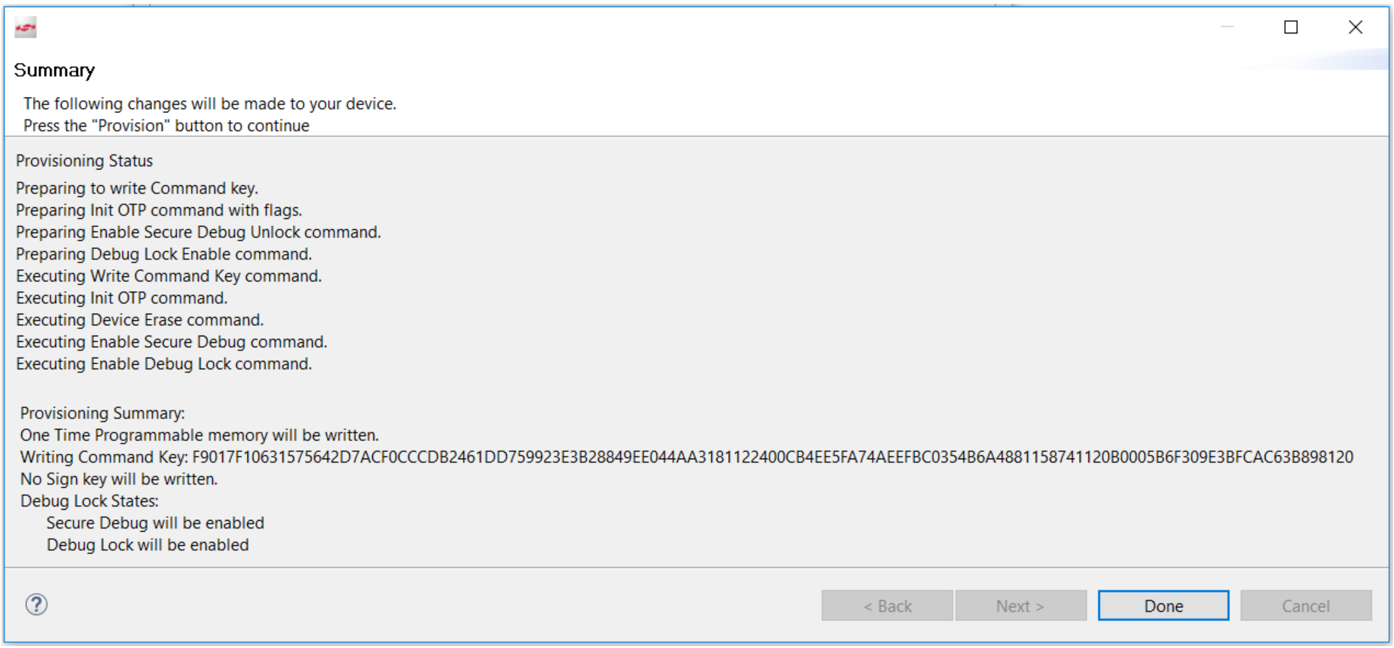


Figure 7.12. Device Provisioning Window

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

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

Figure 7.13. Provisioning Status



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

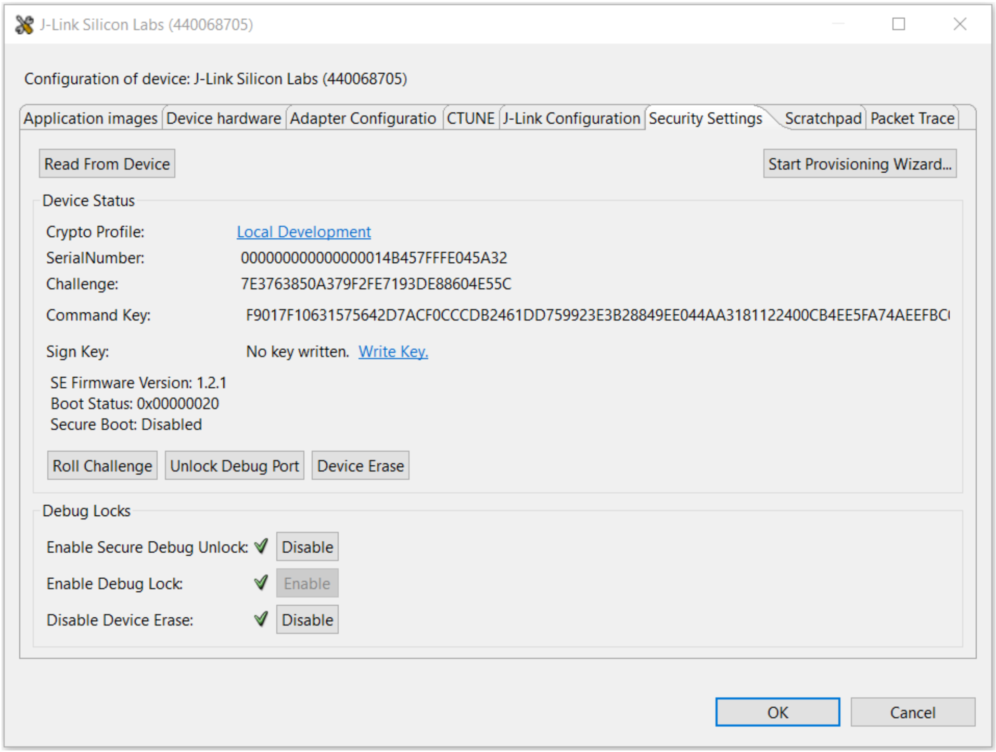


Figure 7.14. Device Configuration after Provisioning

12. Click [**Disable**] next to **Disable Device Erase:** to disable the device erase. The following **Disable Device Erase Warning** is displayed. Click [**Yes**] to confirm.

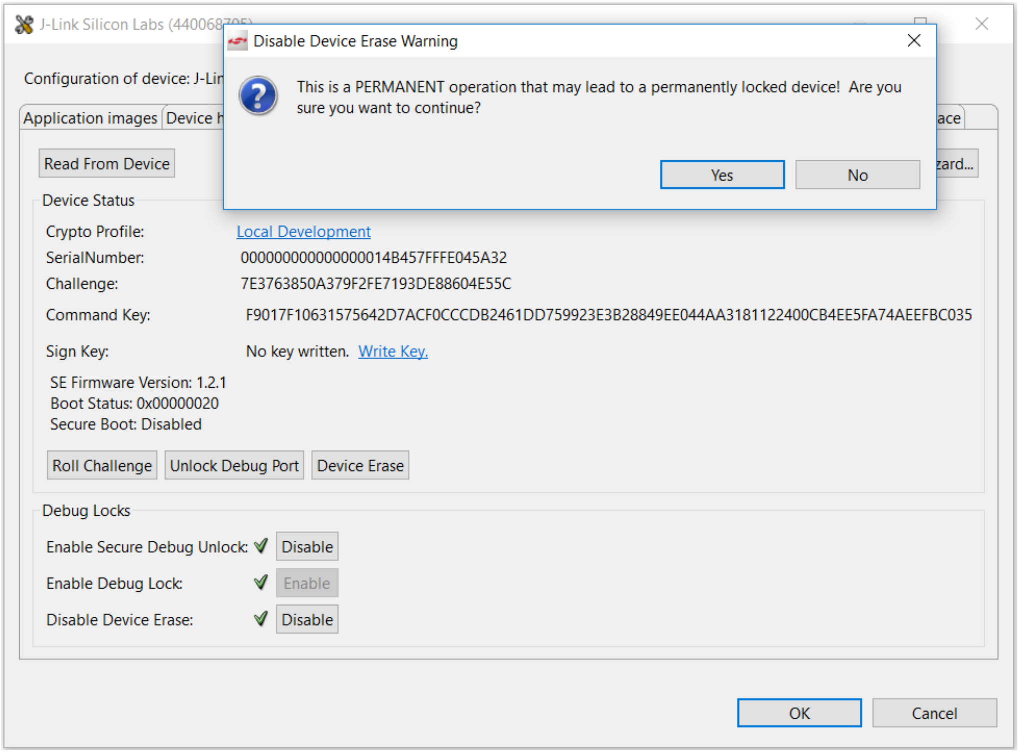


Figure 7.15. Disable Device Erase

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

7.3.2 Simplicity Commander

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

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

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

2. Run the `security writekey` command to provision the Public Command Key with the `command_pubkey.pem` file generated in 7.1.2 Using Simplicity Commander step 5.

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

```
Device has serial number 0000000000000000014b457fffed50d1e

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

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

3. Run the `security readkey` command to verify the Public Command Key with the `command_pubkey.txt` file generated in 7.1.2 Using Simplicity Commander step 6.

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

```
F9017F10631575642D7ACF0CCCDB2461DD759923E3B28849EE044AA318112240
0CB4EE5FA74AEEFBC0354B6A4881158741120B0005B6F309E3BFCAC63B898120
DONE
```

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

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

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

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

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

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

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

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

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

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

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

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

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

7.4 Secure Debug Unlock

7.4.1 Secure Element Manager

See section 7.1.4 Using a Platform Example for more information on programming the secure debug platform example to the WSTK.

For demonstration purpose, a default Private Command Key is stored in device memory to sign the access certificate for secure debug unlock. The corresponding Public Command Key must be programmed to the SE OTP to perform secure debug unlock. The location of the default Private Command Key (`cmd-unsafe-privkey.pem`) in Windows is `C:\SiliconLabs\SimplicityStudio\<version>\developer\adapter_packs\secmgr\scripts\offline`.

If the device does not have a Public Command Key in the SE OTP, press PB0 on the WSTK twice to program the default Public Command Key to the device.

The user can change the Private Command Key (`private_command_key[]`) in `app_se_manager_secure_debug.c` to pair with the device's Public Command Key in the SE OTP for the secure debug unlock test. The hard-coded Private Command Key is an insecure method, so the user should find a way to import the signed access certificate for secure debug unlock.

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

Get the Secure Element (SE) status:

```
SE Manager Secure Debug Example - Core running at 38000 kHz.
  . SE manager initialization... SL_STATUS_OK (cycles: 8 time: 0 us)

  . Get SE status... SL_STATUS_OK (cycles: 11391 time: 299 us)
  + The SE firmware version (MSB..LSB): 00010201
  + Debug lock: Disabled
  + Debug lock state: False
  + Device Erase: Enabled
  + Secure debug: Disabled
  + Secure boot: Disabled
```

Enable secure debug and lock the device:

```
. The device is in normal state and secure debug is disabled.
+ Exporting a public command key from a hard-coded private command key... SL_STATUS_OK (cycles: 197404 time: 5194 us)
+ Reading the public command key from SE OTP... SL_STATUS_OK (cycles: 6938 time: 182 us)
+ Comparing exported public command key with SE OTP public command key... OK
+ Press PB0 to enable secure debug or press PB1 to exit.
+ Enable the secure debug... SL_STATUS_OK (cycles: 43357 time: 1140 us)
+ Press PB0 to lock the device or press PB1 to disable the secure debug and exit.
+ Locking the device... SL_STATUS_OK (cycles: 46960 time: 1235 us)
+ Device erase is enabled, press PB0 to disable device erase (optional) or press PB1 to skip.

. Get SE status... SL_STATUS_OK (cycles: 11281 time: 296 us)
+ The SE firmware version (MSB..LSB): 00010201
+ Debug lock: Enabled
+ Debug lock state: True
+ Device Erase: Enabled
+ Secure debug: Enabled
+ Secure boot: Disable
```

Perform secure debug unlock:

```
. The device is in secure debug lock state.
+ Press PB0 to issue a secure debug unlock or press PB1 to exit.
+ Creating a private certificate key in a buffer... SL_STATUS_OK (cycles: 213016 time: 5605 us)
+ Exporting a public certificate key from a private certificate key... SL_STATUS_OK (cycles: 211205 time: 5558 us)
+ Read the serial number of the SE and save it to access certificate... SL_STATUS_OK (cycles: 7973 time: 209 us)
+ Signing the access certificate with private command key... SL_STATUS_OK (cycles: 216541 time: 5698 us)
+ Request challenge from the SE and save it to challenge response... SL_STATUS_OK (cycles: 4796 time: 126 us)
+ Signing the challenge response with private certificate key... SL_STATUS_OK (cycles: 219730 time: 5782 us)
+ Setting the debug options... SL_STATUS_OK (cycles: 8795 time: 231 us)
+ Creating an unlock token to unlock the device... SL_STATUS_OK (cycles: 908489 time: 23907 us)
+ Get debug status to verify the device is unlocked... SL_STATUS_OK (cycles: 7413 time: 195 us)
+ Success to unlock the device!

. Get SE status... SL_STATUS_OK (cycles: 11471 time: 301 us)
+ The SE firmware version (MSB..LSB): 00010201
+ Debug lock: Enabled
+ Debug lock state: False
+ Device Erase: Enabled
+ Secure debug: Enabled
+ Secure boot: Disabled
```

Roll the challenge:

```
. The device is in secure debug unlock state.
+ Issue a power-on or pin reset to re-enable the secure debug lock.
+ Press PB0 to roll the challenge to invalidate the current unlock token or press PB1 to exit.
+ The challenge cannot be rolled before it has been used at least once (secure debug unlock was issued).
+ Request current challenge from the SE... SL_STATUS_OK (cycles: 4527 time: 119 us)
+ The current challenge (16 bytes): 9A 6E 29 7B 8A CD F3 70 FA C4 78 4B 1A D3 24 F3
+ Rolling the challenge... SL_STATUS_OK (cycles: 19891 time: 523 us)
+ Request rolled challenge from the SE... SL_STATUS_OK (cycles: 4782 time: 125 us)
+ The rolled challenge (16 bytes): 4C 76 C8 19 82 59 B0 EA 3C 30 D5 F4 01 8C 66 26
+ Issue a power-on or pin reset to activate the rolled challenge.

. SE manager deinitialization... SL_STATUS_OK (cycles: 7 time: 0 us)
```

7.4.2 Simplicity Commander

The Private/Public Command Key pair (`command_key.pem` and `command_pubkey.pem`) was generated with step 5 in 7.1.2 Using Simplicity Commander. The Public Command Key must be provisioned in advance for secure debug unlock.

7.4.2.1 Local Secure Debug Unlock

The debug access command file can be locally generated if the owner of the Private Command Key can access the device.

Generate the debug access command file:

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

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

```
SE Firmware version : 1.2.1
Serial number       : 0000000000000000d6ffffe0a3a5f
Debug lock          : Enabled
Device erase        : Disabled
Secure debug unlock : Enabled
Tamper status       : OK
Secure boot         : Disabled
Boot status         : 0x20 - OK
DONE
```

2. Run the `security unlock` command with the Private Command Key (`command_key.pem` in 7.1.2 Using Simplicity Commander step 5) to unlock the selected device. The debug interface is temporarily unlocked until the next power-on or pin reset.

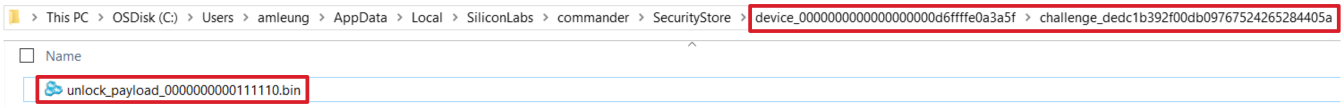
```
commander security unlock --command-key command_key.pem --device EFR32MG21A010F1024 --serialno 440068705
```

```
Command public key stored in:
C:/Users/amleung/AppData/Local/SiliconLabs/commander/SecurityStore/device_0000000000000000d6ffffe0a3a5f/
command_pubkey.pem
Command private key stored in:
C:/Users/amleung/AppData/Local/SiliconLabs/commander/SecurityStore/device_0000000000000000d6ffffe0a3a5f/
command_key.pem
Authorization file written to Security Store:
C:/Users/amleung/AppData/Local/SiliconLabs/commander/SecurityStore/device_0000000000000000d6ffffe0a3a5f/
certificate_authorizations.json
Generating ECC P256 key pair...
Cert public key stored at:
C:/Users/amleung/AppData/Local/SiliconLabs/commander/SecurityStore/device_0000000000000000d6ffffe0a3a5f/
cert_pubkey.pem
Cert private key stored at:
C:/Users/amleung/AppData/Local/SiliconLabs/commander/SecurityStore/device_0000000000000000d6ffffe0a3a5f/
cert_key.pem
Command key matches public command key found on device. Signing certificate...
Certificate was signed with key:
command_key.pem
Created unsigned unlock command
Signed unlock command using
C:/Users/amleung/AppData/Local/SiliconLabs/commander/SecurityStore/device_0000000000000000d6ffffe0a3a5f/
cert_key.pem
Secure debug successfully unlocked
Command unlock payload was stored in Security Store
DONE
```

Note: The debug access command file is generated with the default certificate authorization file (`certificate_authorization.json`), which uses `0x0000003e` for Authorizations and `0x00000000` or `0xffffffb6` (SE with Secure Vault) for Tamper Authorizations (Table 6.3 Elements of the Access Certificate on page 13).

3. 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>`.

4. The debug access command file (`unlock_payload_<Debug mode request>.bin`) for secure debug unlock is stored in the `challenge_<Challenge value>` folder. The location in Windows is `C:\Users\<PC user name>\AppData\Local\SiliconLabs\commander\SecurityStore\device_<Serial number>\challenge_<Challenge value>`.



5. Send the device and debug access command file (`unlock_payload_0000000000111110.bin`) in step 4 to the requesting party for software debugging or failure analysis.

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

Unlock the device:

See ● Unlock the device: on page 39.

7.4.2.2 Remote Secure Debug Unlock

The debug access command file can be remotely generated if the owner of the Private Command Key cannot access the device.

Remote secure debug unlock request:

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

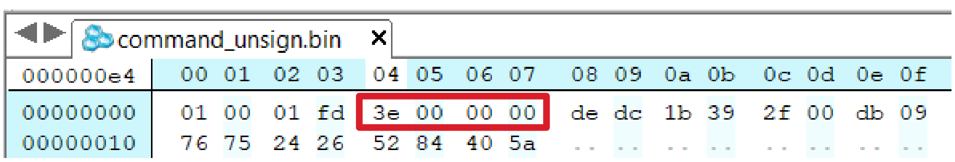
```
commander security status --device EFR32MG21A010F1024 --serialno 440068705
```

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

2. Run the `security gencommand` command to generate the challenge response without debug access command signature and store it in a file (`command_unsign.bin`).

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

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



Note: All debug mode request (`0x0000003e`) are enabled by default.

3. Send the device part number (`EFR32MG21A010F1024`), device serial number (`0000000000000000000d6ffffe0a3a5f`), and unsigned command file (`command_unsign.bin`) to the owner of the Private Command Key.

Authorize the remote secure debug unlock request (WSTK is not required):

1. Run the `security genkey` command to generate the Private/Public 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
```

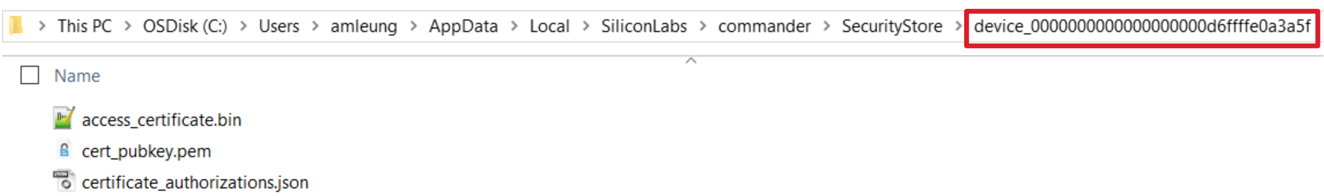
2. Run the `security gencert` command with device part number, device serial number (from the requesting party), and Public Certificate Key generated in step 1 to generate an unsigned access certificate (`access_certificate.bin`).

```
commander security gencert --device EFR32MG21A010F1024 --deviceserialno 0000000000000000d6ffffe0a3a5f --
cert-pubkey cert_pubkey.pem
```

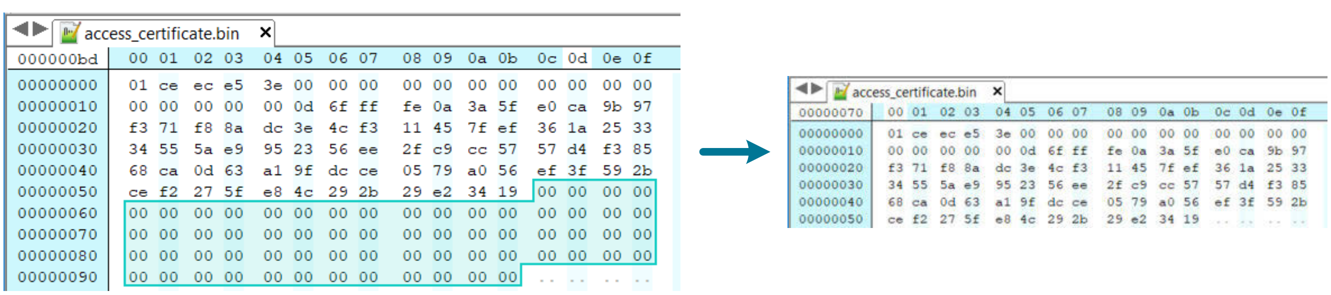
```
Authorization file written to Security Store:
C:/Users/amleung/AppData/Local/SiliconLabs/commander/SecurityStore/device_0000000000000000d6ffffe0a3a5f/
certificate_authorizations.json
Cert key written to Security Store:
C:/Users/amleung/AppData/Local/SiliconLabs/commander/SecurityStore/device_0000000000000000d6ffffe0a3a5f/
cert_pubkey.pem
Certificate was not signed
DONE
```

Note: The unsigned access certificate is generated with the default certificate authorization file (`certificate_authorization.json`) which uses `0x0000003e` for Authorizations and `0x00000000` or `0xfffffb6` (SE with Secure Vault) for Tamper Authorizations (Table 6.3 Elements of the Access Certificate on page 13).

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 the signature.



6. Use OpenSSL to sign the `access_certificate.bin` file with the 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 the Public Command Key (`command_pubkey.pem`).

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

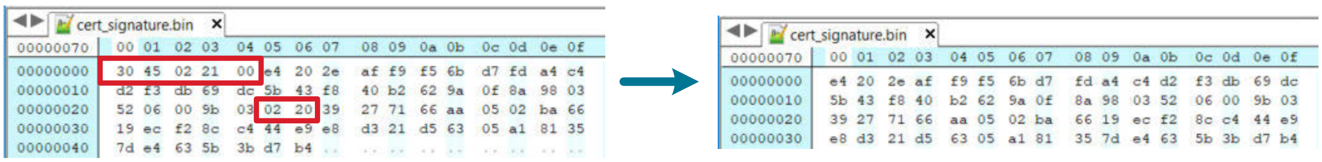
```
Verified OK
```

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

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

```
0:d=0 hl=2 l=69 cons: SEQUENCE
2:d=1 hl=2 l=33 prim: INTEGER   :E4202EAFF9F56BD7FDA4C4D2F3DB69DC5B43F840B2629A0F8A98035206009B03
37:d=1 hl=2 l=32 prim: INTEGER  :39277166AA0502BA6619ECF28CC444E9E8D321D56305A181357DE4635B3BD7B4
```

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



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 debug access 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
```

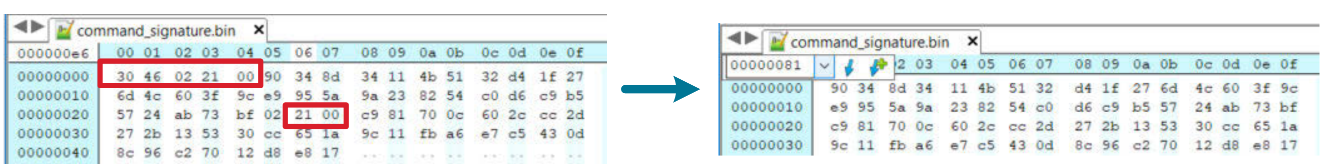
```
Verified OK
```

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=70 cons: SEQUENCE
2:d=1 hl=2 l=33 prim: INTEGER   :90348D34114B5132D41F276D4C603F9CE9955A9A238254C0D6C9B55724AB73BF
37:d=1 hl=2 l=33 prim: INTEGER  :C981700C602CCC2D272B135330CC651A9C11FBA6E7C5430D8C96C27012D8E817
```

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

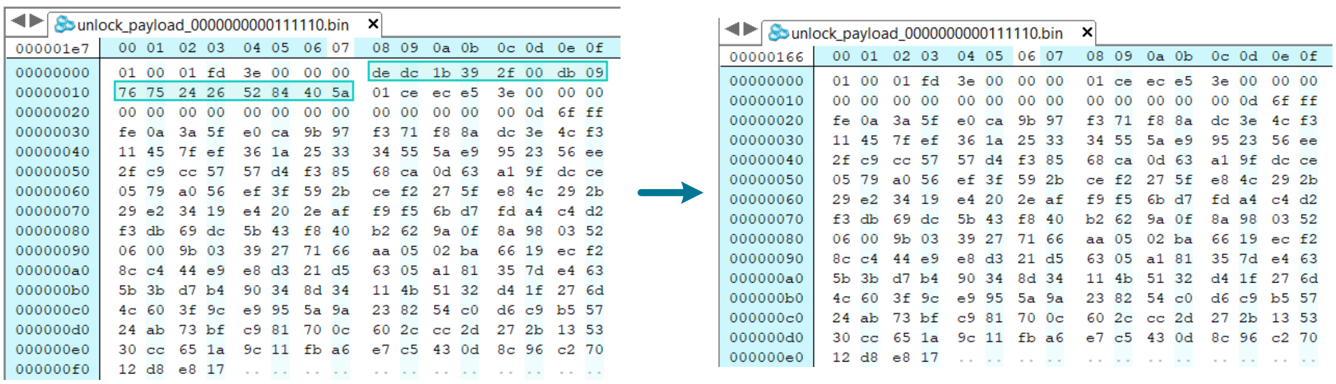


14. Use the DOS `copy` command to merge the files below to generate the debug access command file (**`unlock_payload_<Debug mode request>.bin`**).
    - The unsigned command file (`command_unsign.bin`) from the requesting party.
    - The unsigned access certificate file (`access_certificate.bin`) in step 5.
    - The access certificate signature file (`cert_signature.bin`) in step 9.
    - The debug access command signature file (`command_signature.bin`) in step 13.

```
copy /b command_unsign.bin+access_certificate.bin+cert_signature.bin+command_signature.bin
unlock_payload_0000000000111110.bin
```

```
command_unsign.bin
access_certificate.bin
cert_signature.bin
command_signature.bin
        1 file(s) copied.
```

15. Open the `unlock_payload_0000000000111110.bin` file to remove the 16 bytes challenge in the challenge response that is not required in the debug access command.



16. Send the debug access command file (`unlock_payload_0000000000111110.bin`) in step 15 to the requesting party for software debugging or failure analysis.

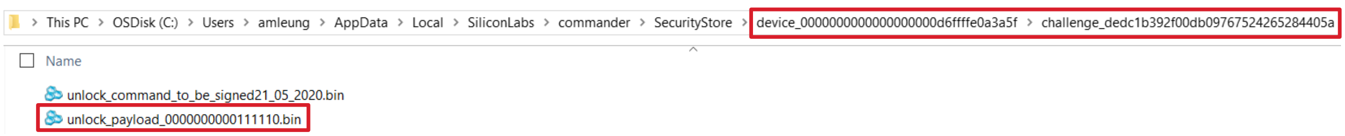
Unlock the device:

1. Run the `security gencommand` command to create the required folder in which to place the debug access command file.

```
commander security gencommand --action debug-unlock --device EFR32MG21A010F1024 --serialno 440068705
```

```
Unsigned command file written to Security Store:
C:/Users/amleung/AppData/Local/SiliconLabs/commander/SecurityStore/device_0000000000000000d6ffffe0a3a5f/
challenge_dedc1b392f00db09767524265284405a/unlock_command_to_be_signed18_05_2020.bin
DONE
```

2. Copy the debug access command file (`unlock_payload_0000000000111110.bin`) to the folder (location in Windows is `C:\Users\<PC user name>\AppData\Local\SiliconLabs\commander\SecurityStore\device_<Serial number>\challenge_<Challenge value>`) created in step 1.



3. Run the `security unlock` command to unlock the device. This debug access command can be reused after power-on or pin reset.

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

```
Unlocking with unlock payload:
C:/Users/amleung/AppData/Local/SiliconLabs/commander/SecurityStore/device_0000000000000000d6ffffe0a3a5f/
challenge_dedc1b392f00db09767524265284405a/unlock_payload_0000000000111110.bin
Secure debug successfully unlocked
DONE
```

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

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

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

5. The debug access command file can be reused with the `security unlock` command after power-on or pin reset.

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

```
Unlocking with unlock payload:
C:/Users/amleung/AppData/Local/SiliconLabs/commander/SecurityStore/device_0000000000000000d6ffffe0a3a5f/
challenge_dedc1b392f00db09767524265284405a/unlock_payload_0000000000111110.bin
Secure debug successfully unlocked
DONE
```

7.4.2.3 Roll Challenge to Revoke Secure Debug Access

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

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

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

2. Run the `security unlock` command to verify the current debug access command file is no longer valid.

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

```
Authorization file written to Security Store:
C:/Users/amleung/AppData/Local/SiliconLabs/commander/SecurityStore/device_0000000000000000000d6ffffe0a3a5f/
certificate_authorizations.json
Generating ECC P256 key pair...
Cert public key stored at:
C:/Users/amleung/AppData/Local/SiliconLabs/commander/SecurityStore/device_0000000000000000000d6ffffe0a3a5f/
cert_pubkey.pem
Cert private key stored at:
C:/Users/amleung/AppData/Local/SiliconLabs/commander/SecurityStore/device_0000000000000000000d6ffffe0a3a5f/
cert_key.pem
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
```

8. Revision History

Revision 0.3

September 2020

- Added EFR32BG21B and EFR32MG21B to 2. Device Compatibility.
- Modified 4. Secure Element Subsystem for SE with Secure Vault devices.
- Added Secure Element Manager to 4.2.1 Mailbox.
- Added Secure Element Manager APIs to Table 5.5 Debug Lock Command Reference on page 10.
- Updated content in 6.3 Secure Debug Unlock.
- Added Secure Element Manager APIs to Table 6.6 Debug Unlock Command Reference on page 17.
- Updated the figures in 7.1.1 Using Simplicity Studio to Simplicity Studio v5.
- Updated Simplicity Commander version to 1.9.2 in 7.1.2 Using Simplicity Commander.
- Added 7.1.4 Using a Platform Example in 7.1 Overview.
- Updated content in 7.4 Secure Debug Unlock.
- Added 7.4.1 Secure Element Manager example in 7.4 Secure Debug Unlock.

Revision 0.2

March 2020

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

Revision 0.1

February 2019

- Initial Revision.

Simplicity Studio

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

IoT Portfolio
www.silabs.com/IoT

SW/HW
www.silabs.com/simplicity

Quality
www.silabs.com/quality

Support & Community
www.silabs.com/community

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

http://www.silabs.com