AN1190: Series 2 Secure Debug

This version of AN1190 has been deprecated with the release of Simplicity SDK Suite 2025.6.1.

For the latest version, see docs.silabs.com.

****

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 Engine on Series 2 devices.

KEY POINTS

- Basic overview of the Secure Engine.
- 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. Series 2 Device Security Features

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

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

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

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

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

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

Note:

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

Secure Vault Mid consists of two core security functions:

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

Secure Vault High offers additional security options:

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

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

1.1 User Assistance

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

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

1.2 Key Reference

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

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

1.3 SE Firmware

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

2. Introduction to Secure Debug

2.1 Debug Lock

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

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

A fairly standard practice during the board-level test in production is to program, test, and lock the parts.

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

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

Silicon Labs provides Custom Part Manufacturing Service (CPMS) to securely configure the debug port of the chip to one of the three possible locks before the devices leave the factory.

2.2 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

3. Secure Engine Subsystem

3.1 Overview

The HSE 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 SE is used to perform a series of cryptographic operations and other secure system operations as described in the following table.

Operation	VSE-SVM	HSE-SVM	HSE-SVH	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 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-SVM devices, the crypto engine and TRNG (True Random Number Generator) 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 SE 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 SE on Series 2 devices.

The secure debug feature is implemented by Root code executed by the HSE Core or by the Cortex-M33 operating in VSE (Root mode).

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

For GSDK v3.2 and lower:

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

For GSDK v4.0 and higher:

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

3.2 Command Interface

Interaction with the SE 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 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 HSE.

3.2.1 Mailbox

Mailbox operations should not be performed directly, but rather should be executed through 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 setup.

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

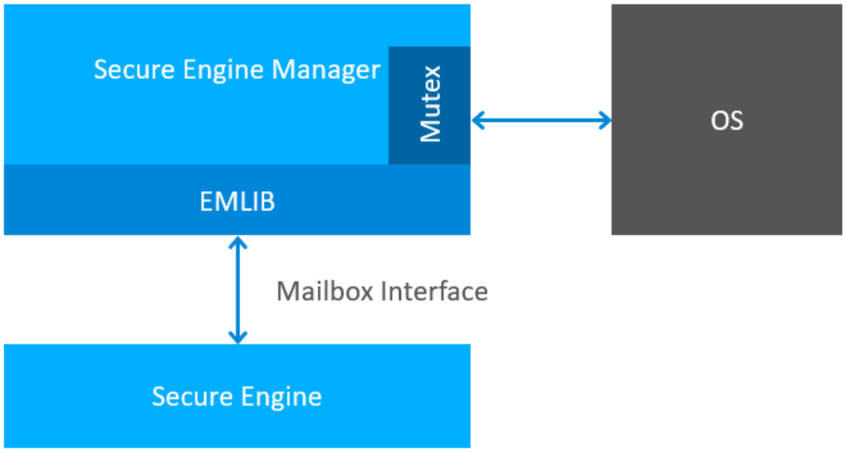


Figure 3.1. EMLIB and SE Manager

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 SE Manager.

3.2.2 Debug Challenge Interface (DCI)

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

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

4. Debug Lock

4.1 Overview

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

Table 4.1. 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 SE command interface either over DCI or the mailbox interface. The status of the debug lock can be inspected using the Read Lock Status command.

4.2 Standard Debug Unlock

The device is in standard debug unlock state if the debug lock properties are in default values.

Table 4.2. Standard Debug Unlock

Secure Debug	Device Erase	Debug Lock	Description
Disabled	Enabled	Disabled (Unlock)	All debug operations are allowed.

4.3 Standard Debug Lock

With the default properties in the table above, the device can be locked using the Apply Lock command. The 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 or through the mailbox interface.

Table 4.3. Standard Debug Lock

Secure Debug	Device Erase	Debug Lock	Description
Disabled	Enabled	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.

4.4 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 4.4. Permanent Debug Lock

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

4.5 Secure Debug Lock

For secure debug lock, the debug interface can be temporarily enabled by answering a challenge if the Secure debug property is enabled before locking.

Table 4.5. Secure Debug Lock

Secure Debug	Device Erase	Debug Lock	Description
Enabled (1)	Disabled (2)	Enabled (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 SE Manager API.
    b. Enable secure debug using Simplicity Studio or Simplicity Commander or directly through the SE Manager API.
2. Disable the device erase using Simplicity Studio or Simplicity Commander or directly through the SE Manager API. This is an **IRREVERSIBLE** action and should be disabled **AFTER** the secure debug is enabled.

4.6 Debug Lock State Transition

The following figure describes the transitions between different debug lock states.

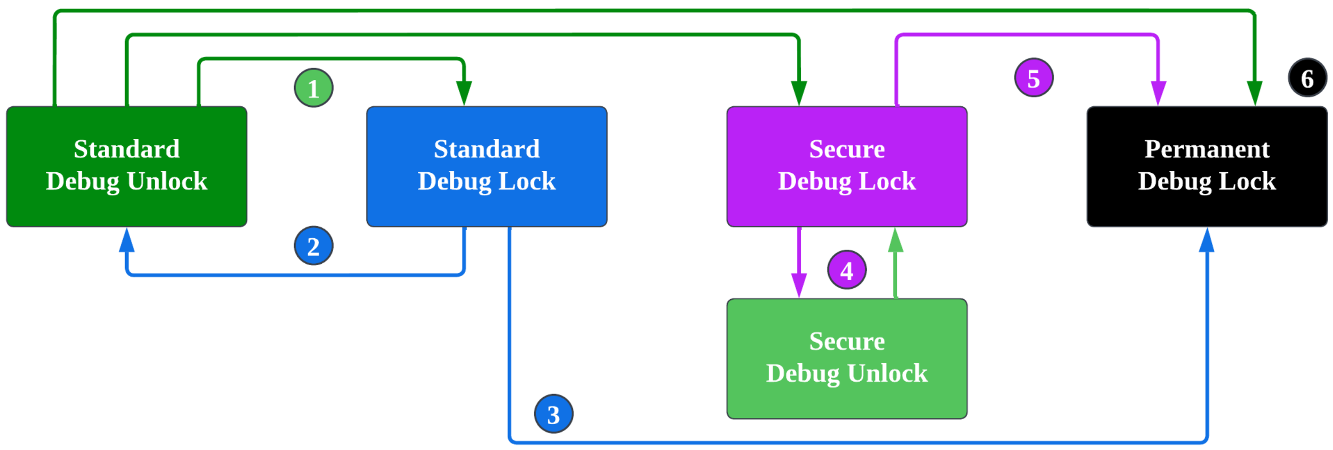


Figure 4.1. Debug Lock State Transition

1. Standard debug unlock can transit to any debug lock state.
2. Standard debug lock can revert to standard debug unlock via an Erase Device command (erase the main flash and RAM). After the device is reset, debug port remains unlocked until it is explicitly locked again.
3. Standard debug lock can transit to permanent debug lock by disabling the Device Erase property but cannot transit to secure debug lock.
4. Secure debug lock can use Debug Unlock Token to temporary transit to secure debug unlock, which does not erase the main flash and RAM but enables debug operations. The device reverts to the secure debug lock through a power-on or pin reset.
5. Secure debug lock can transit to permanent debug lock by disabling the Secure Debug property but cannot transit to standard debug lock.
6. Permanent debug lock is a terminal state and cannot transit to any debug lock or unlock state.

4.7 Debug Lock Command Reference

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

Table 4.6. Debug Lock Command Reference

DCI Command (1)	Mailbox (SE Manager) API (2)	Description	Availability
Apply Lock	sl_se_apply_debug_lock	Enables the debug lock for the part.	While debug is unlocked.
Read Lock Status	sl_se_get_debug_lock_status	Returns the current debug lock status and configuration.	Always.
Disable Device Erase	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.	Always.
Disable Secure Debug	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	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.
Set debug options	sl_se_set_debug_options	Configures the TrustZone access permissions of the debug interface. (3)	While debug is unlocked.
Init Pub Key	sl_se_init_otp_key	Used to provision a single public key during device initialization. The public key cannot be changed once written, and the command will be unavailable for that key.	Available once for each key.
Read Pub Key	sl_se_read_pubkey	Reads the stored public key.	Always.
Get Challenge	sl_se_roll_challenge	Used to roll the current challenge value (16 bytes) to revoke secure debug access. (4)	While Public Command Key is uploaded.

Note:

1. Performing these commands over DCI is implemented in Simplicity Studio and Simplicity Commander.
2. The `sl_se_apply_debug_lock`, `sl_se_get_debug_lock_status`, `sl_se_init_otp_key`, and `sl_se_read_pubkey` are available on all Series 2 devices. Other APIs are only available on HSE devices. The SE Manager API document can be found at https://docs.silabs.com/gecko-platform/latest/service/api/group-sl-se-manager.
3. For more information about debug options, see 5.3.5 TrustZone Debug Authentication.
4. A new challenge will only be generated if the current one has been successfully used at least once.

5. Debug Unlock

5.1 Overview

The debug access port connected to the Series 2 device's Cortex-M33 processor can be opened by issuing commands to the SE, either from a debugger over DCI or through the mailbox interface.

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.

5.2 Standard Debug Unlock

With the properties of the standard debug lock or secure debug lock with Device Erase enabled, the device can be returned to the standard debug unlock state using the Erase Device command. This command will wipe the main flash and RAM and verify they are empty before opening the debug lock. It will not wipe user data and provisioned SE settings.

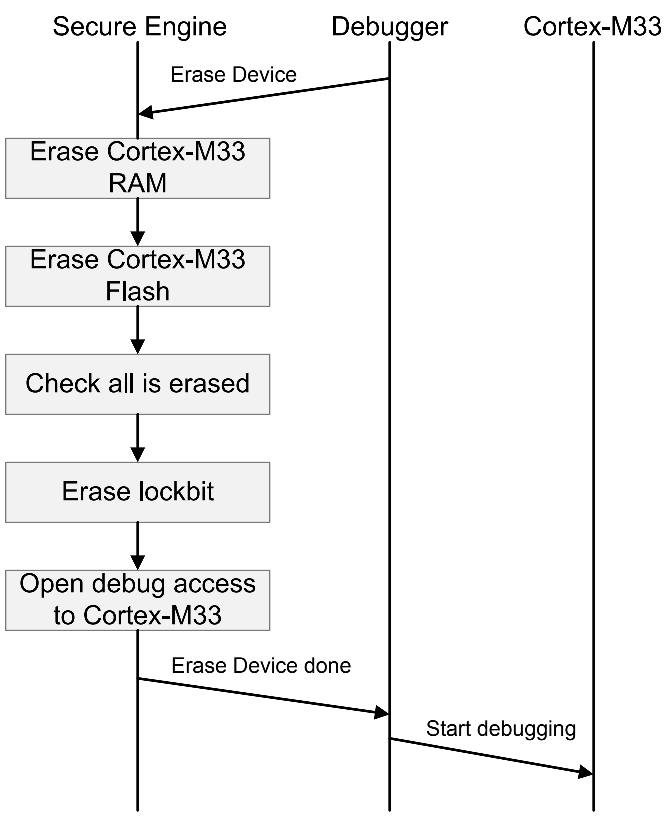


Figure 5.1. Standard Debug Unlock

5.3 Secure Debug Unlock

In a secure debug unlock scenario, the customer, who has control over the Private Command Key for a SE, has programmed a Public Command Key into the device. The Public Command Key is used to verify the signature on a certificate, telling the SE 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 HSE-SVH devices.

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

5.3.1 Debug Unlock Token

The elements of the Debug Unlock Token are described in the following figures and table.

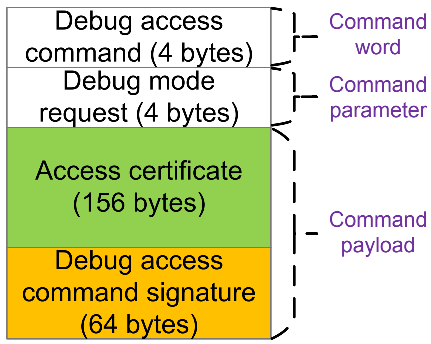


Figure 5.2. Debug Unlock Token

Table 5.1. Elements of Debug Unlock Token

Element	Value	Description
Debug access command	0xfd010001	The command word of the Debug Unlock Token.
Debug mode request	Device-dependent	The command parameter of the debug access command.
Access certificate (1)	Device-dependent	See section Access Certificate.
Debug access command signature (1)	Device-dependent	See section Challenge Response.

Note:

1. The debug access command payload consists of an access certificate and a debug access command signature.

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

Figure 5.3. Debug Mode Request

Note:

- Enable debug port - Debug port enabled if set.
- DBGLOCK (Non-secure, Invasive debug lock) - The Invasive debug features for the Non-secure state are unlocked if set.
- NIDLOCK (Non-secure, Non-invasive debug lock) - The Non-invasive debug features for the Non-secure state are unlocked if set.
- SPIDLOCK (Secure, Invasive debug lock) - The Invasive debug features for the Secure state are unlocked if set.
- SPNIDLOCK (Secure, Non-Invasive debug lock) - The Non-invasive debug features for the Secure state are unlocked if set.
- All reserved bits should be 0, and bit 1 must be 1 to access the debug port.
- For the TrustZone-unaware debugging, bits 2 to 5 are irrelevant, so bits 1 to 5 are usually set (`0x0000003e`) to match with the Authorizations in the access certificate.
- For the TrustZone-aware debugging, bits 2 to 5 are relevant. Refer to 5.3.5 TrustZone Debug Authentication for details about these debug options.

5.3.2 Access Certificate

The elements of the access certificate are described in the following figures and table.

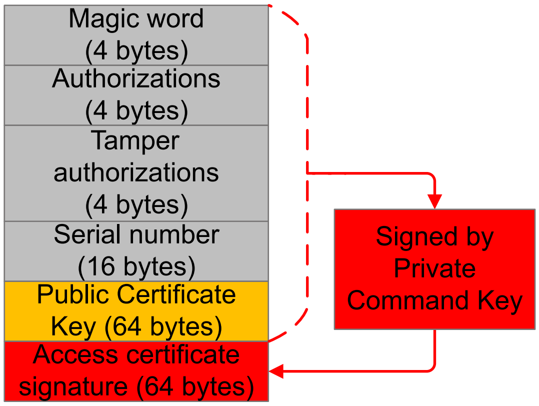


Figure 5.4. Access Certificate

Table 5.2. 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 for secure debug.
Tamper Authorizations	0x00000000 or 0xffffffb6 (2)	A value used to authorize which bit in the tamper disable mask can be set to disable the tamper response.
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 SE OTP.

Note:

1. This value allows all debug options to be reset for secure debug.
2. Value that sets available bits in the tamper disable mask for tamper disable (HSE-SVH device only).
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 more information about tamper disable, see AN1247: Anti-Tamper Protection Configuration and Use.

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

Figure 5.5. Authorizations

Note:
- Set the bit to enable the corresponding bit in the debug mode request.
- The Debug Unlock Token will reset the corresponding debug option if the same bit is set in Debug mode request and Authorizations.

5.3.3 Challenge Response

The elements of the challenge response are described in the following figure and table.

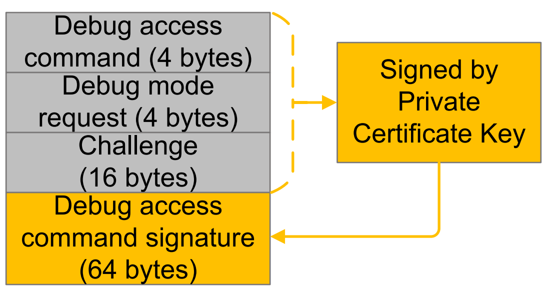


Figure 5.6. Challenge Response

Table 5.3. Elements of the Challenge Response

Element	Value	Description
Debug access command	0xfd010001	The command word of the Debug Unlock Token.
Debug mode request	Device-dependent	The command parameter of the debug access command.
Challenge	Device-dependent (1)	A random value generated by the SE.
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 Unlock Token.

5.3.4 Debug Access Flow

The debug access flow is described in the following figure.

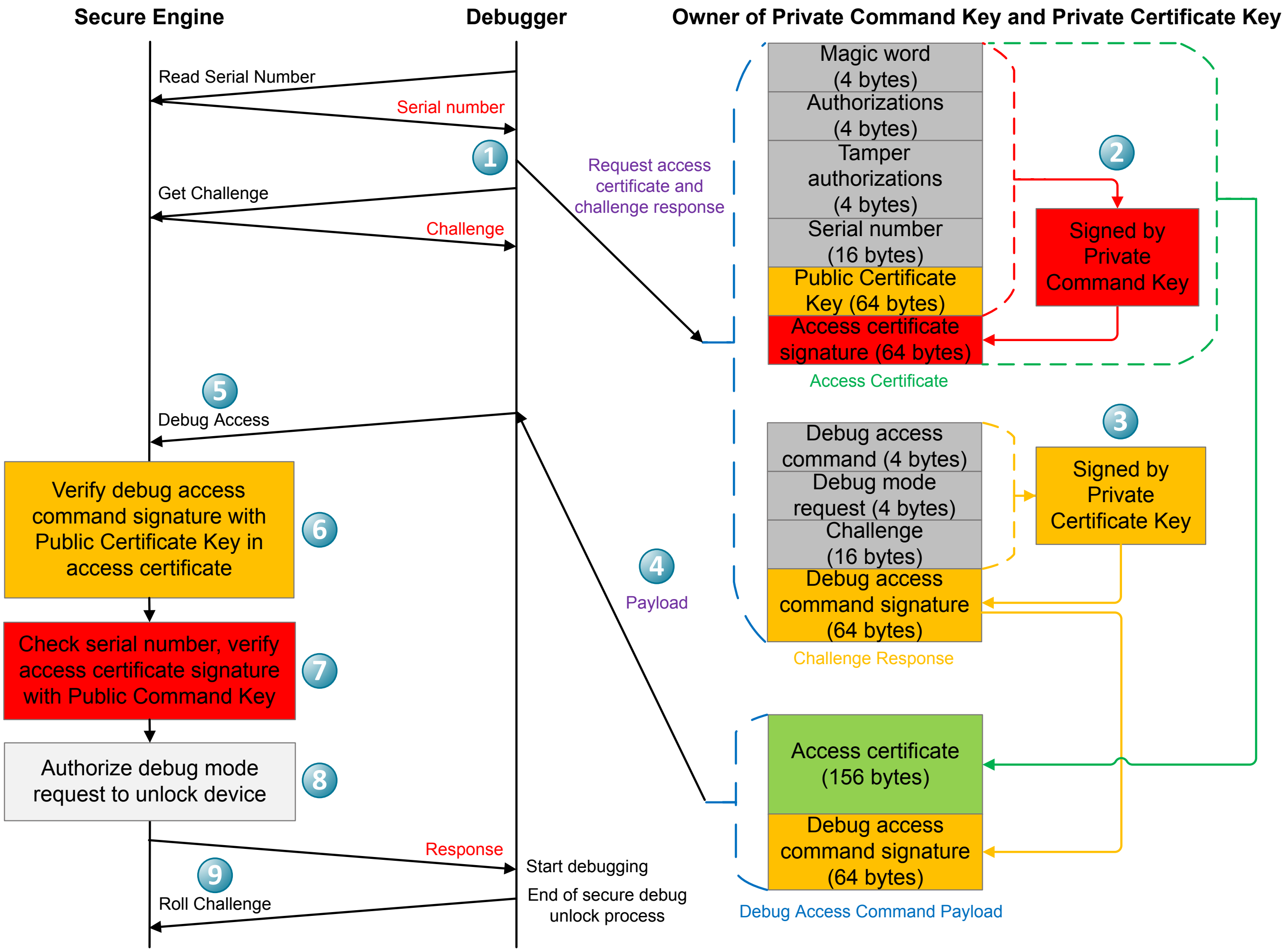


Figure 5.7. Debug Access Flow

1. Get the serial number and challenge from the SE.
2. Generate the access certificate with the device serial number.
3. Generate the challenge response with device challenge.
4. Generate the debug access command payload with access certificate and debug access command signature.
5. Send the Debug Unlock Token to the SE.
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 the SE OTP.
8. Authorize the debug mode request to reset the debug options until the next power-on or pin reset.
9. Roll the challenge to invalidate the current Debug Unlock Token.

5.3.5 TrustZone Debug Authentication

The debug and trace support in the Cortex-M33 devices are based on the CoreSight architecture, which can be classified into Invasive and Non-invasive debugging features as described in the following table.

Classification	Debug and Trace Features	Description
Invasive	Core debug (e.g., single stepping), Breakpoints, Data watchpoints, Halt mode debugging	These features halt the Cortex-M33 core and change the program execution flow.
Non-invasive	Embedded Trace Macrocell (ETM), Micro Trace Buffer (MTB), Data trace, Instrumentation Trace Macrocell (ITM), Profiling	These features have a minor or no impact on the program execution flow.

The separation of Invasive and Non-invasive debug and trace operations in CoreSight architecture can apply to TrustZone debug authentication, which defines the permission levels of the debug and trace features on Secure and Non-secure worlds.

The table below describes four debug options in SE to support TrustZone debug authentication. It is possible to restrict the TrustZone access permissions of the debug interface by setting one or more of the following options.

Debug Option	Description
DBGLOCK	Non-secure, Invasive debug lock. If this bit is set, the Invasive debug features for the Non-secure state are locked.
NIDLOCK	Non-secure, Non-invasive debug lock. If this bit is set, the Non-invasive debug features for the Non-secure state are locked.
SPIDLOCK	Secure, Invasive debug lock. If this bit is set, the Invasive debug features for the Secure state are locked.
SPNIDLOCK	Secure, Non-invasive debug lock. If this bit is set, the Non-invasive debug features for the Secure state are locked.

Note:

- Use Simplicity Commander or the SE Manager API to set the debug options.
- The state of the debug options is stored permanently in SE and can only be reset to the default value (`0000`) through the Erase Device command (if enabled).
- A secure debug lock device (Device Erase was disabled) can only use the Debug Unlock Token to temporarily unlock (reset) the debug options to debug the Secure and Non-secure applications.

The following conditions are recommended (1, 2, and 3) or mandatory (4) when setting up the debug options for secure debug unlock.

1. If SPIDLOCK is unlocked, then DBGLOCK should also be unlocked.
2. If SPNIDLOCK is unlocked, then NIDLOCK should also be unlocked.
3. If DBGLOCK is unlocked, the NIDLOCK should also be unlocked.
4. If SPIDLOCK is unlocked, then SPNIDLOCK is automatically unlocked.

The following table lists the recommended combinations of debug options.

SPNIDLOCK	SPIDLOCK	NIDLOCK	DBGLOCK	Description
0	0	0	0	Allows all debug and trace features for both the Secure and the Non-secure world (default setting).
0	1	0	0	Only allows a Non-invasive debug in the Secure world. Allows both Invasive and Non-invasive debugs in the Non-secure world.
0	1	0	1	Only allows a Non-invasive debug in the Secure and the Non-secure world.
1	1	0	0	Only allows debug and trace features in the Non-secure world.
1	1	0	1	Only allows a Non-invasive debug in the Non-secure world.
1	1	1	1	All debug and trace features are disabled.

Note:

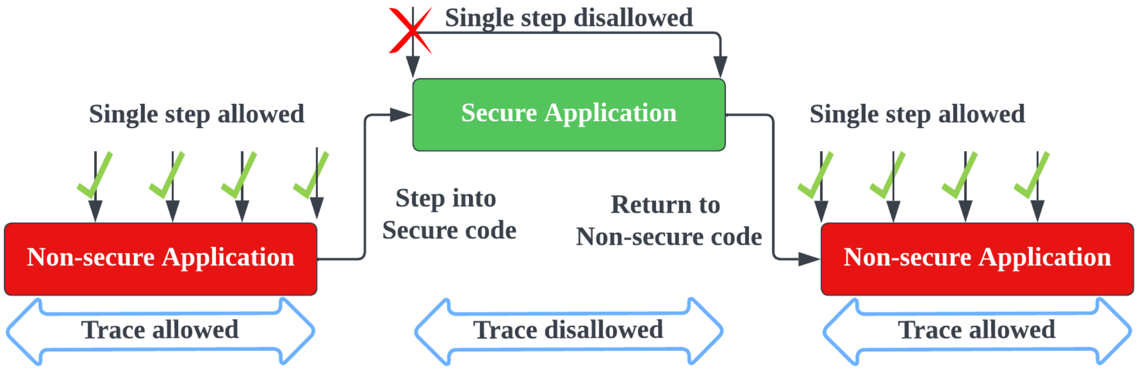
- Trace Point Interface Unit (TPIU) registers' access fault will occur and lock the processor in a security assertion if both NIDLOCK and DBGLOCK in debug option are set (`xx11`). The device will be unrecoverable if it is in the permanent debug lock state.

- The workaround is to avoid using the `xx11` debug option or avoid accessing the TPIU registers and upgrade to SE firmware ≥ **v1.2.14** (xG21 and xG22) or ≥ **v2.2.1** (other Series 2 devices) so that the debug options cannot be modified after the device is locked.

The highly recommended setting of debug options is to allow debugging in the Non-secure world while, at the same time, disabling debugging for the Secure world (`1100`).

- Secure memories (flash and RAM) are not accessible by the debugger.
- All debug access is blocked from accessing Secure addresses.
- The debugger will ignore the vector-catch events generated by the Secure exceptions.
- Trace sources (e.g., ETM) will stop generating instruction/data trace packets when the Cortex-M33 is in a Secure state.
- The debugger can neither halt a Secure application (e.g., breakpoint) nor single step into it.
- The Cortex-M33 will not stop when stepping into the Secure application until it returns to the Non-secure state.

The figure below describes the debug scenario of debug options with `1100` configuration.



The following examples describe the relationship between **debug options** and **debug mode request** when performing a secure debug unlock on Series 2 devices.

Example 1: All debug and trace features for both the Secure and the Non-secure world are allowed (`0000`)

Debug Options	Authorizations	Debug Mode Request	Debug options after Secure Debug Unlock	Description
0000	00|1111|10	00|xxxx|10	0000	No action

Example 2: Only debug and trace features in the Non-secure world are allowed (`1100`)

Debug Options	Authorizations	Debug Mode Request	Debug options after Secure Debug Unlock	Description
1100	00|1111|10	00|00xx|10	1100	No action
1100	00|1111|10	00|10xx|10	0100	Unlock SPNIDLOCK
1100	00|1111|10	00|01xx|10 or 00|11xx|10	0000 (reset SPIDLOCK will automatically unlock SPNIDLOCK)	Unlock SPNIDLOCK and SPIDLOCK

Note:

- The bit order of debug options are SPNIDLOCK (MSB), SPIDLOCK, NIDLOCK, and DBGLOCK (LSB).
- Debug options: 0 = Unlocked, 1 = Locked
- Authorizations in the access certificate: 0 = Disable, 1 = Enable
- The authorizations in the access certificate are usually set to `00|1111|10` (`0x3e`), so the corresponding debug options (bits 2 to 5) can be reset (unlocked) by debug mode request during secure debug unlock.
- Debug mode request (bits 2 to 5) in the Debug Unlock Token:
  - 0 = No action on the corresponding debug option if it was locked (i.e., 1)
  - 1 = Reset (unlock) the corresponding debug option from 1 to 0 if it was locked (i.e., 1)
  - x = No action (either 0 or 1) on the corresponding debug option if it was unlocked (i.e., 0)
- Debug options return to the original state after power-on or pin reset.

5.4 Debug Unlock Command Reference

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

Table 5.4. Debug Unlock Command Reference

DCI Command (1)	Mailbox (SE Manager) API (2)	Description	Availability
Erase Device	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	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. These APIs are only available on HSE devices. The SE Manager API document can be found at https://docs.silabs.com/gecko-platform/latest/service/api/group-sl-se-manager.

6. Examples

6.1 Overview

The examples for Series 2 debug lock and debug unlock are described in the following table.

Table 6.1. Series 2 Debug Lock and Debug Unlock Examples

Example	Device (Radio Board)	SE Firmware	Tool
Standard debug lock	EFR32MG21A010F1024IM32 (BRD4181A)	Version 1.2.14	SE Manager
Standard debug lock and unlock	EFR32MG21A010F1024IM32 (BRD4181A)	Version 1.2.14	Simplicity Commander
"	EFR32MG21A010F1024IM32 (BRD4181A)	Version 1.2.9	Simplicity Studio 5
Provision Public Command Key	EFR32MG21A010F1024IM32 (BRD4181A)	Version 1.2.14	SE Manager
Secure debug lock	EFR32MG21A010F1024IM32 (BRD4181A)	Version 1.2.14	SE Manager
Provision Public Command Key and secure debug lock	EFR32MG21A010F1024IM32 (BRD4181A)	Version 1.2.14	Simplicity Commander
"	EFR32MG21A010F1024IM32 (BRD4181A)	Version 1.2.9	Simplicity Studio 5
Secure debug unlock and roll challenge	EFR32MG21A010F1024IM32 (BRD4181A)	Version 1.2.14	SE Manager
"	EFR32MG21A010F1024IM32 (BRD4181A)	Version 1.2.14	Simplicity Commander
"	EFR32MG21A010F1024IM32 (BRD4181A)	Version 1.2.9	Simplicity Studio 5
Secure debug unlock	EFR32MG21A010F1024IM32 (BRD4181A)	Version 1.2.9	IAR v8.50.9
Permanent debug lock	EFR32MG21A010F1024IM32 (BRD4181A)	Version 1.2.14	Simplicity Commander
"	EFR32MG21A010F1024IM32 (BRD4181A)	Version 1.2.9	Simplicity Studio 5

Note: Unless specified in the example, these examples can be applied to other Series 2 devices.

6.1.1 Using Simplicity Studio

The security operations are performed in the Security Settings of Simplicity Studio. This application note uses Simplicity Studio v5.2.1.1. The procedures and pictures may be different for the other versions of Simplicity Studio 5.

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

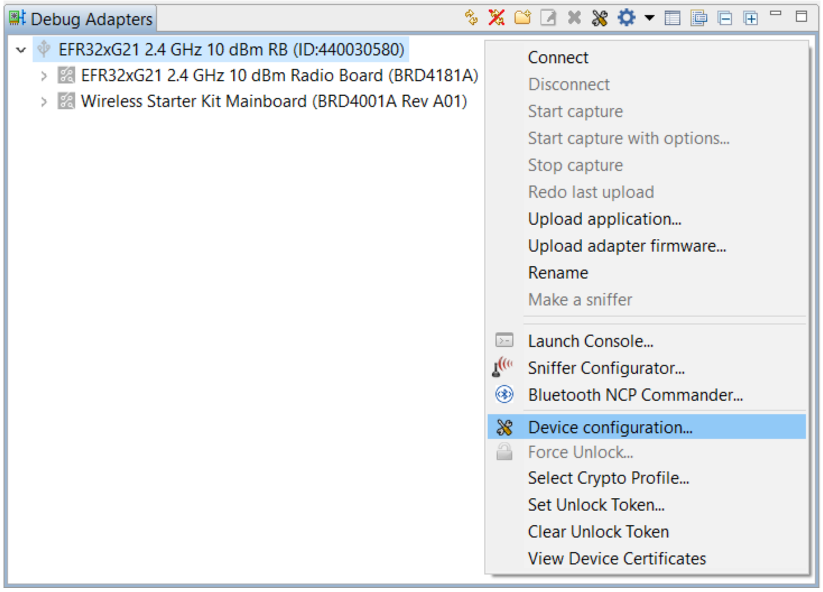


Figure 6.1. Debug Adapters Context Menu

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.

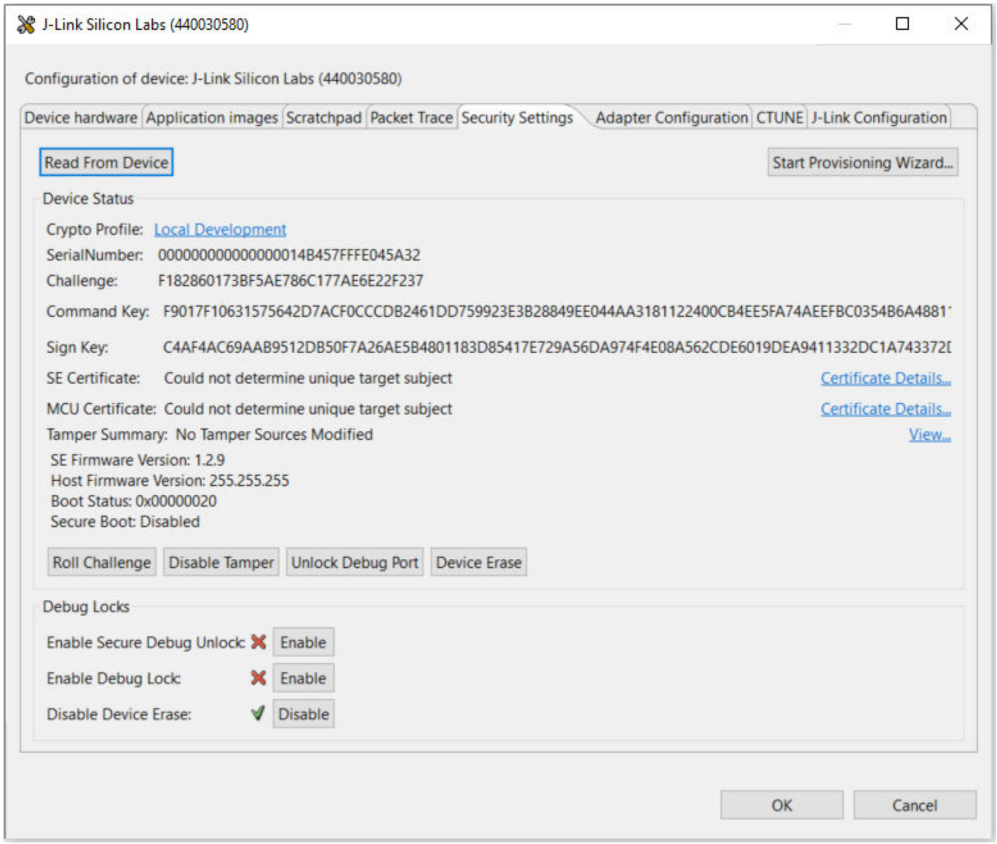


Figure 6.2. Configuration on Selected Device

6.1.2 Using Simplicity Commander

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

```
commander --version
```

```
Simplicity Commander 1v14p2b1232

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

Emulator found with SN=440048205 USBAddr=0

DONE
```

2. The Simplicity Commander's Command Line Interface (CLI) is invoked by `commander.exe` in the Simplicity Commander folder. The location for Simplicity Studio 5 in Windows is `C:\SiliconLabs\SimplicityStudio\v5\developer\adapter_packs\commander`. For ease of use, it is highly recommended to add the path of `commander.exe` to the system `PATH` in Windows.
3. If more than one Wireless Starter Kit (WSTK) is connected via USB, the target WSTK must be specified using the `--serialno <J-Link serial number>` option.
4. If the WSTK is in debug mode OUT, the target device must be specified using the `--device <device name>` option.

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

6.1.3 Using External Tools

1. The secure debug unlock example uses the **OpenSSL** to sign the access certificate and challenge response. The Windows version of OpenSSL can be downloaded from https://slproweb.com/products/Win32OpenSSL.html. This application note uses OpenSSL Version 1.1.1h (Win64).

```
openssl version
```

```
OpenSSL 1.1.1h  22 Sep 2020
```

   The OpenSSL's Command Line Interface (CLI) is invoked by `openssl.exe` in the OpenSSL folder. The location in Windows (Win64) is `C:\Program Files\OpenSSL-Win64\bin`. For ease of use, it is highly recommended to add the path of `openssl.exe` to the system `PATH` in Windows.
2. The secure debug unlock example uses the free **Hex Editor Neo** to edit the binary files generated by Simplicity Commander. The Windows version of Hex Editor Neo can be downloaded from https://www.hhdsoftware.com/free-hex-editor.

6.1.4 Using Platform Examples

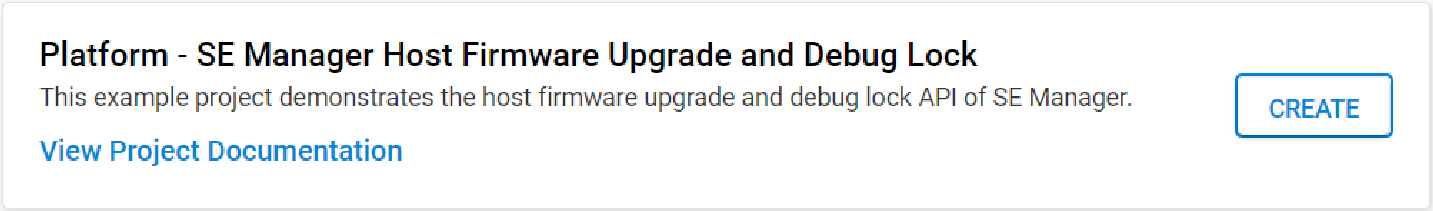
Simplicity Studio 5 includes the SE Manager platform examples for debug lock, key provisioning, and secure debug unlock. This application note uses platform examples of GSDK v4.2.1. The console output may be different on other versions of GSDK.

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

6.2 Standard Debug Lock and Unlock

6.2.1 SE Manager - Debug Lock Platform Example

Click the `View Project Documentation` link to open the `readme` file.



1. Press `SPACE` then `ENTER` to select the debug lock operation.

```
SE Manager Host Firmware Upgrade and Debug Lock Example - Core running at 38000 kHz.
  . SE manager initialization... SL_STATUS_OK (cycles: 10 time: 0 us)

  . Current selection is HOST FIRMWARE UPGRADE.
  + Press SPACE to select HOST FIRMWARE UPGRADE or DEBUG LOCK, press ENTER to run.
  + Current selection is DEBUG LOCK.

  . Get debug lock status... SL_STATUS_OK (cycles: 8788 time: 231 us)
  + Debug lock: Disabled
  + Press ENTER to apply debug lock or press SPACE to exit.
```

2. Press `ENTER` again to lock the device.

```
  . Apply the debug lock...  SL_STATUS_OK (cycles: 52585 time: 1383 us)
  + Get debug lock status... SL_STATUS_OK (cycles: 8769 time: 230 us)
  + Debug lock: Enabled

  . Current selection is DEBUG LOCK.
  + Press SPACE to select HOST FIRMWARE UPGRADE or DEBUG LOCK, press ENTER to run.
```

6.2.2 Simplicity Commander

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

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

```
SE Firmware version : 1.2.14
Serial number       : 0000000000000000014b457fffe045a93
Debug lock          : Disabled
Device erase        : Enabled
Secure debug unlock : Disabled
Tamper status       : OK
Secure boot         : Disabled
Boot status         : 0x20 - OK
DONE
```

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

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

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

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

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

```
SE Firmware version : 1.2.14
Serial number       : 0000000000000000014b457fffe045a93
Debug lock          : Enabled
Device erase        : Enabled
Secure debug unlock : Disabled
Tamper status       : OK
Secure boot         : Disabled
Boot status         : 0x20 - OK
DONE
```

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

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

```
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 EFR32MG21A010F1024 --serialno 440048205
```

```
SE Firmware version : 1.2.14
Serial number       : 0000000000000000014b457fffe045a93
Debug lock          : Disabled
Device erase        : Enabled
Secure debug unlock : Disabled
Tamper status       : OK
Secure boot         : Disabled
Boot status         : 0x20 - OK
DONE
```

6.2.3 Simplicity Studio

1. Open the **Security Settings** of the selected device as described in 6.1.1 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.

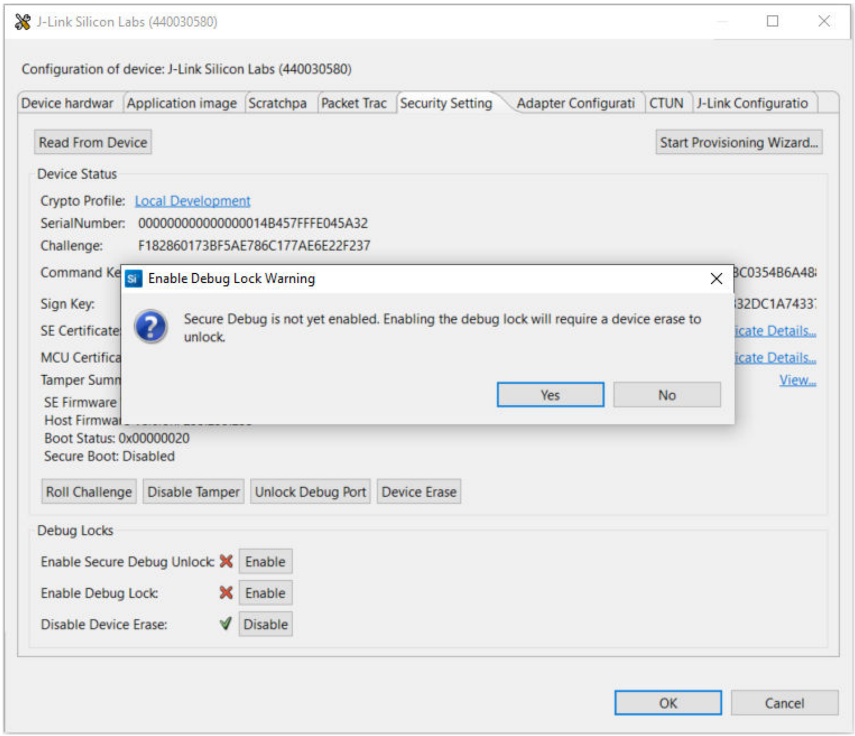


Figure 6.3. Enable Debug Lock

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

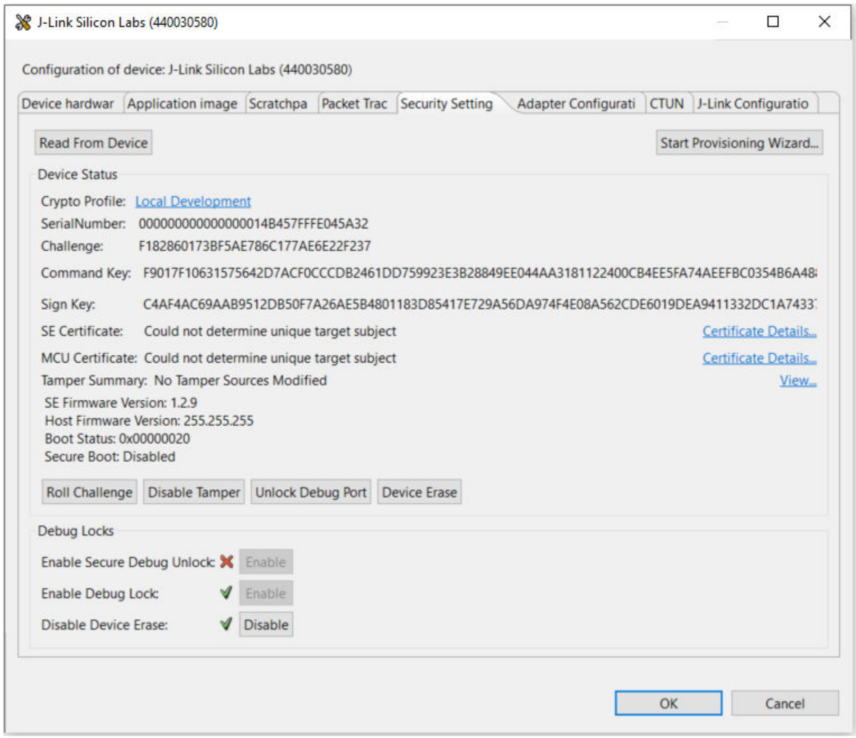


Figure 6.4. Standard Debug Lock

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

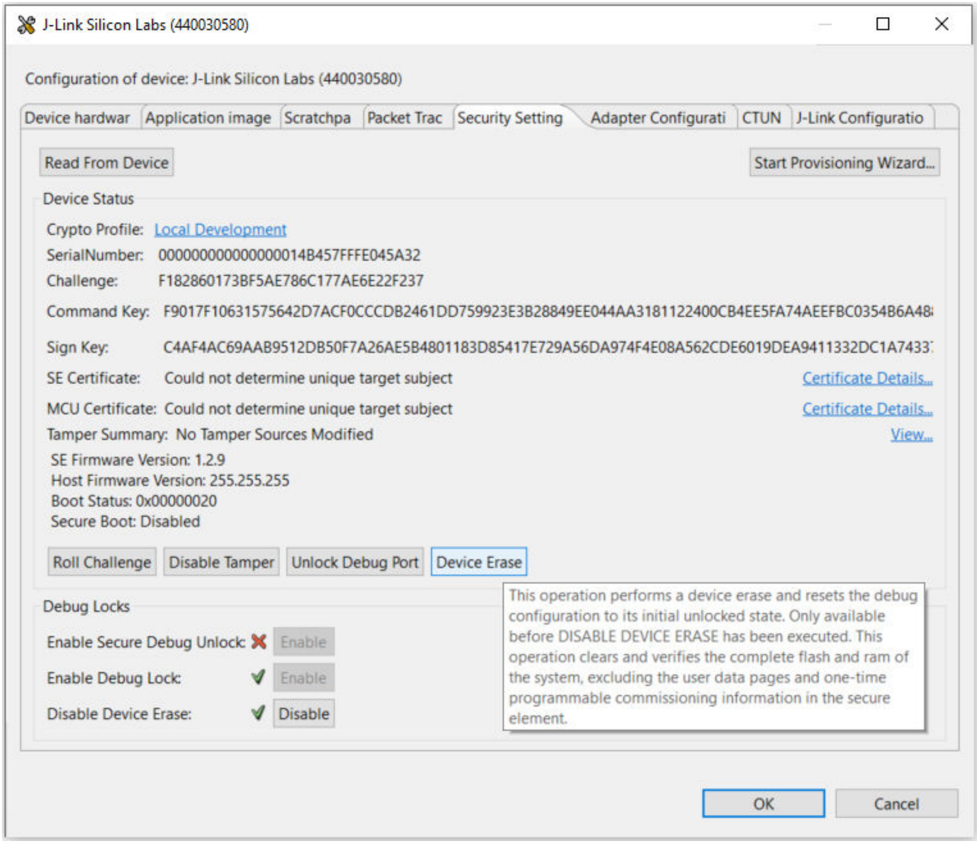


Figure 6.5. Device Erase

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

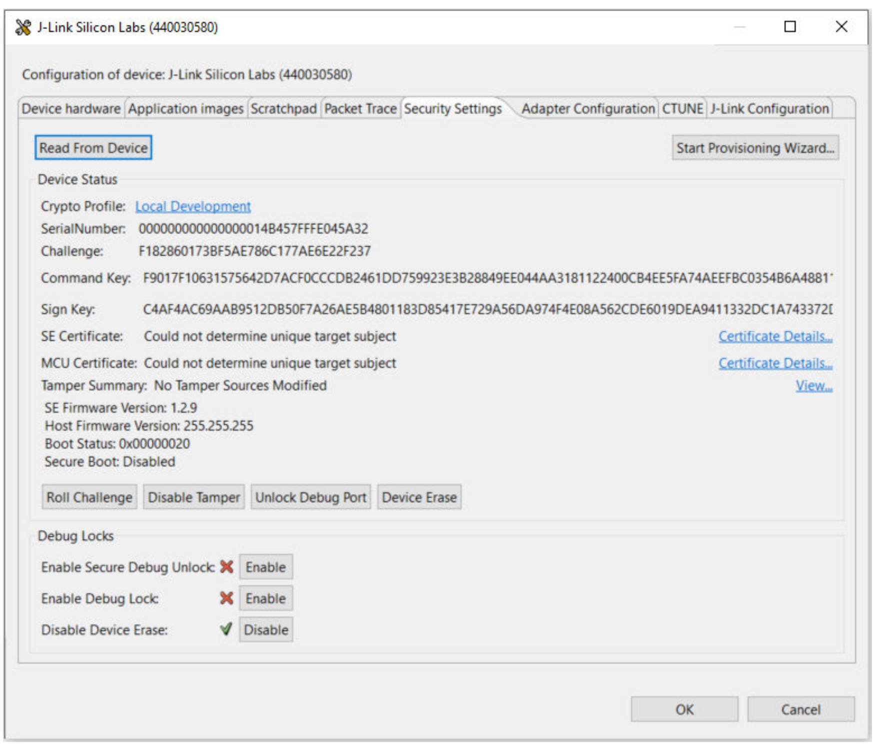
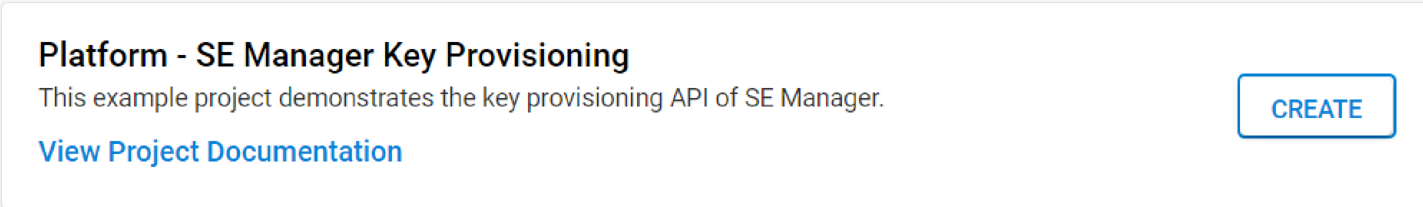


Figure 6.6. Standard Debug Unlock

6.3 Provision Public Command Key and Secure Debug Lock

6.3.1 SE Manager - Key Provisioning Platform Example

Click the `View Project Documentation` link to open the `readme` file.



1. Press `SPACE` to skip the programming of AES-128 key.

```
SE Manager Key Provisioning Example - Core running at 38000 kHz.
  . SE manager initialization... SL_STATUS_OK (cycles: 9 time: 0 us)

  . Get current SE firmware version... SL_STATUS_OK (cycles: 3578 time: 94 us)
  + Current SE firmware version (MSB..LSB): 00010209

  . Read SE OTP configuration... SL_STATUS_COMMAND_IS_INVALID (cycles: 3908 time: 102 us)

  . Press ENTER to program 128-bit AES key in SE OTP or press SPACE to skip.

  . Encrypt 16 bytes plaintext with 128-bit AES OTP key... SL_STATUS_FAIL (cycles: 4627 time: 121 us)

  . Press ENTER to program public sign key in SE OTP or press SPACE to skip.
```

2. Press `SPACE` to skip the programming of Public Sign Key.

```
  . Get public sign key... SL_STATUS_FAIL (cycles: 4144 time: 109 us)

  . Press ENTER to program public command key in SE OTP or press SPACE to skip.
```

3. Press `ENTER` to program the default Public Command Key in flash to the SE OTP.

```
  + Warning: The public command key in SE OTP cannot be changed once written!
  + Press ENTER to confirm or press SPACE to skip if you are not sure.
```

4. Press `ENTER` to confirm the operation.

```
  . Initialize public command key... SL_STATUS_OK (cycles: 56052 time: 1475 us)

  . Get public command key... SL_STATUS_OK (cycles: 7135 time: 187 us)
  + The public command key (64 bytes):
    B1 BC 6F 6F A5 66 40 ED 52 2B 2E E0 F5 B3 CF 7E
    5D 48 F6 0B E8 14 8F 0D C0 84 40 F0 A4 E1 DC A4
    7C 04 11 9E D6 A1 BE 31 B7 70 7E 5F 9D 00 1A 65
    9A 05 10 03 E9 5E 1B 93 6F 05 C3 7E A7 93 AD 63

  . Press ENTER to initialize SE OTP for secure boot configuration or press SPACE to skip.
```

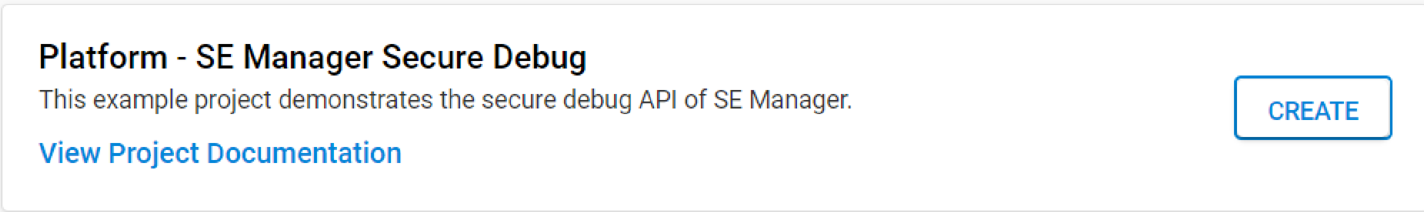
5. Press `SPACE` to skip the secure boot configuration.

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

6.3.2 SE Manager - Secure Debug Platform Example

Click the `View Project Documentation` link to open the `readme` file.

Note: The secure debug platform example can only run on the HSE device.



1. Use a standard debug unlock device with matched Public Command Key.

```
SE Manager Secure Debug Example - Core running at 38000 kHz.
  . SE manager initialization... SL_STATUS_OK (cycles: 9 time: 0 us)

  . Get SE status... SL_STATUS_OK (cycles: 8496 time: 223 us)
  + The SE firmware version (MSB..LSB): 0001020E
  + Debug lock: Disabled
  + Device Erase: Enabled
  + Secure debug: Disabled
  + Secure boot: Disabled

  + Debug lock state: Unlocked

  + Non-secure, Invasive debug lock (DBGLOCK) configuration: Unlocked
  + Non-secure, Non-invasive debug lock (NIDLOCK) configuration: Unlocked
  + Secure, Invasive debug lock (SPIDLOCK) configuration: Unlocked
  + Secure, Non-invasive debug lock (SPNIDLOCK) configuration: Unlocked

  + Non-secure, Invasive debug lock (DBGLOCK) current state: Unlocked
  + Non-secure, Non-invasive debug lock (NIDLOCK) current state: Unlocked
  + Secure, Invasive debug lock (SPIDLOCK) current state: Unlocked
  + Secure, Non-invasive debug lock (SPNIDLOCK) current state: Unlocked

  . 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: 202467 time: 5328 us)
  + Reading the public command key from SE OTP... SL_STATUS_OK (cycles: 7589 time: 199 us)
  + Comparing exported public command key with SE OTP public command key... OK
  + Press ENTER to enable secure debug or press SPACE to exit.
```

2. Press `ENTER` to enable the secure debug.

```
  + Enable the secure debug... SL_STATUS_OK (cycles: 48313 time: 1271 us)
  + Press ENTER to lock the device or press SPACE to disable the secure debug and exit.
```

3. Press `ENTER` to lock the device with debug options `0x0c`.

```
  + Setting the debug options (0xc)... SL_STATUS_OK (cycles: 51091 time: 1344 us)
  + Locking the device... SL_STATUS_OK (cycles: 89683 time: 2360 us)
  + Device erase is enabled, press ENTER to disable device erase (optional if just for testing) or press SPACE to skip.
```

4. Press `ENTER` to disable device erase.

```
  + Warning: This is a ONE-TIME command which PERMANETLY disables device erase!
  + Press ENTER to confirm or press SPACE to skip if you are not sure.
```

5. Press `ENTER` to confirm the operation.

```
. Get SE status... SL_STATUS_OK (cycles: 8496 time: 223 us)
+ The SE firmware version (MSB..LSB): 0001020E
+ Debug lock: Enabled
+ Device Erase: Disabled
+ Secure debug: Enabled
+ Secure boot: Disabled

+ Debug lock state: Locked

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

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

. The device is in secure debug lock state.
+ Press ENTER to issue a secure debug unlock or press SPACE to exit.
```

6. Press `SPACE` to exit.

```
. SE manager deinitialization... SL_STATUS_OK (cycles: 9 time: 0 us)
```

6.3.3 Simplicity Commander

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

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

```
SE Firmware version : 1.2.14
Serial number       : 000000000000000014b457fffe045a93
Debug lock          : Disabled
Device erase        : Enabled
Secure debug unlock : Disabled
Tamper status       : OK
Secure boot         : Disabled
Boot status         : 0x20 - OK
DONE
```

2. Run the `security writekey` command to provision the Public Command Key (e.g., `command_pubkey.pem`).

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

```
Device has serial number 000000000000000014b457fffe045a93

================================================================================
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 read the Public Command Key from the SE OTP.

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

```
B1BC6F6FA56640ED522B2EE0F5B3CF7E5D48F60BE8148F0DC08440F0A4E1DCA4
7C04119ED6A1BE31B7707E5F9D001A659A051003E95E1B936F05C37EA793AD63
DONE
```

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

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

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

5. a. For the **TrustZone-unaware** application, run the `security lock` command to lock the selected device.

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

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

b. For the **TrustZone-aware** application, run the `security lock --trustzone ####` command to set the debug options (e.g., `1100`) and lock the selected device. The bit order of `####` is SPNIDLOCK (MSB), SPIDLOCK, NIDLOCK, and DBGLOCK (LSB).

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

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

Note:

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

6. Run the `security disabledeviceerase` command to disable device erase. This is an **IRREVERSIBLE** action, and should be the last step in production.

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

```

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

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

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

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

```
SE Firmware version : 1.2.14
Serial number       : 000000000000000014b457fffe045a93
Debug lock          : Enabled
Device erase        : Disabled
Secure debug unlock : Enabled
Tamper status       : OK
Secure boot         : Disabled
Boot status         : 0x20 - OK
DONE
```

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

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

```
SE Firmware version : 1.2.14
Serial number       : 000000000000000014b457fffe045a93
Debug lock          : Enabled
Device erase        : Disabled
Secure debug unlock : Enabled

Debug lock state: Locked

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

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

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

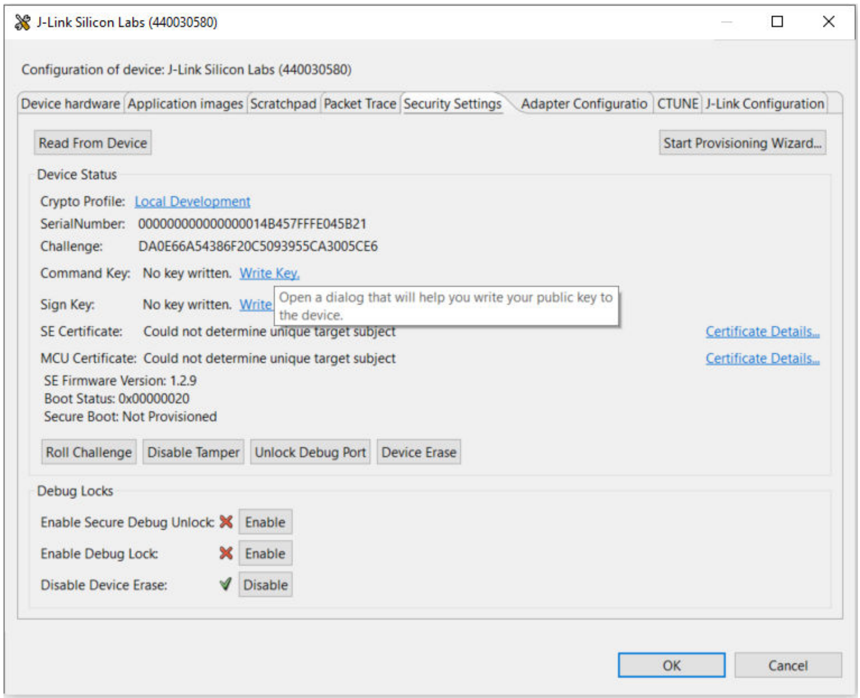
6.3.4 Simplicity Studio

1. Run the `util keytotoken` command to convert the Public Command Key file (PEM format) into a text file (`command_pubkey.txt`).

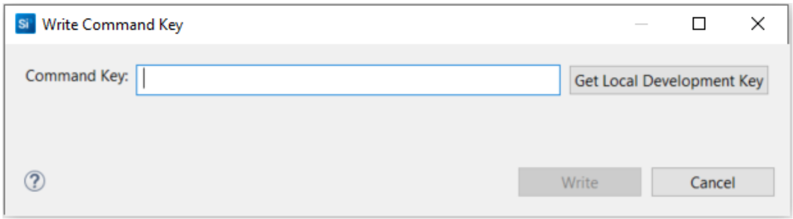
```
commander util keytotoken command_pubkey.pem --outfile command_pubkey.txt
```

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

2. Open **Security Settings** of the selected device as described in 6.1.1 Using Simplicity Studio.
3. Click the **WriteKey** link next to **Command Key:** to open a dialog box.



4. The **Write Command Key** dialog box is displayed.



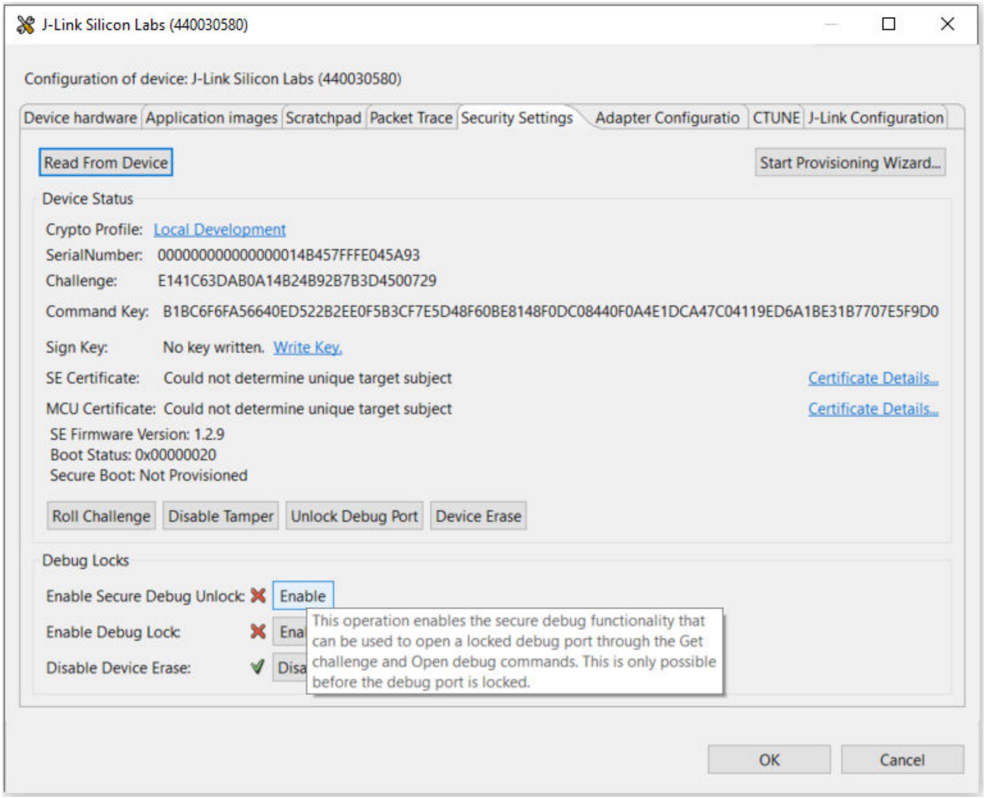
5. Open the `command_pubkey.txt` file generated in step 1.

```
MFG_SIGNED_BOOTLOADER_KEY_X : B1BC6F6FA56640ED522B2EE0F5B3CF7E5D48F60BE8148F0DC08440F0A4E1DCA4
MFG_SIGNED_BOOTLOADER_KEY_Y : 7C04119ED6A1BE31B7707E5F9D001A659A051003E95E1B936F05C37EA793AD63
```

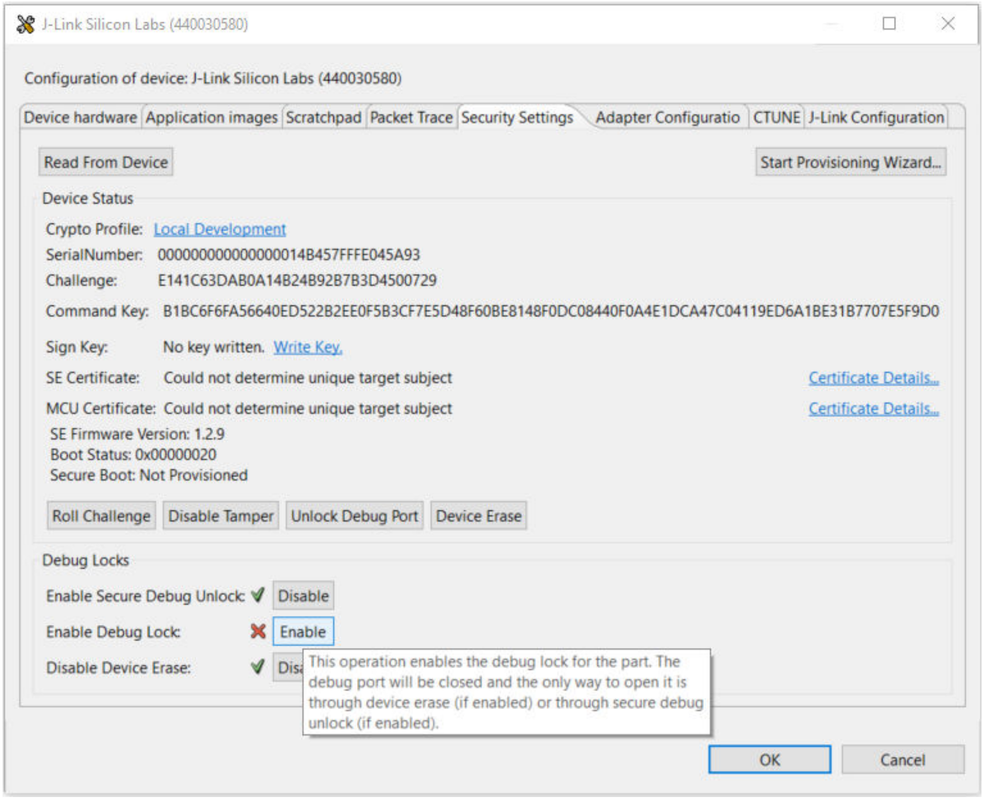
6. Copy Public Command Key (X-point `B1BC...` first, then Y-point `7C04...`) to **Command Key:** box.

7. Click [**Write**] to provision the Public Command Key.

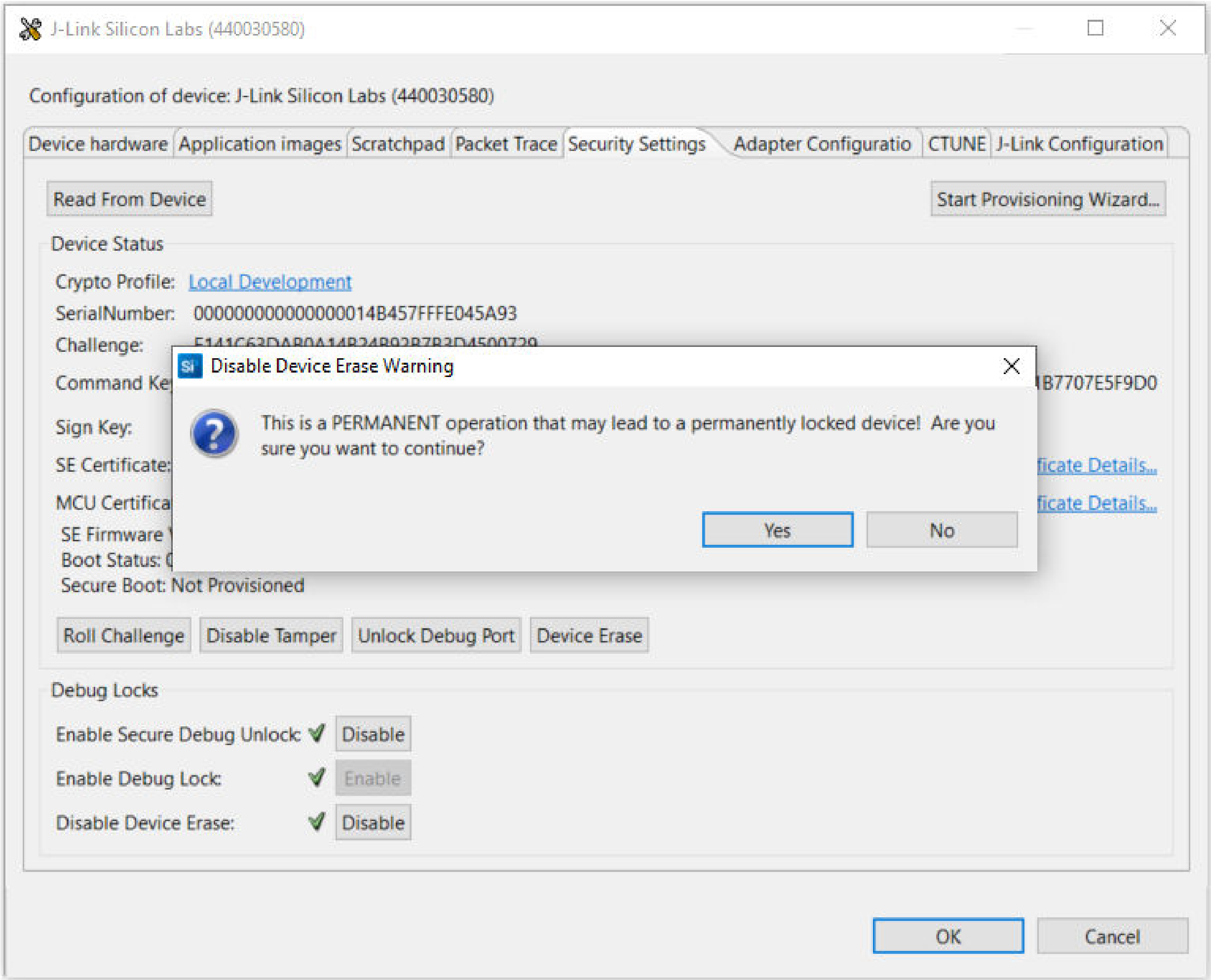
8. Click [**Enable**] next to **Enable Secure Debug Unlock:** to enable the secure debug functionality.



9. Click [**Enable**] next to **Enable Debug Lock:** to lock the device. This configures secure debug lock.



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



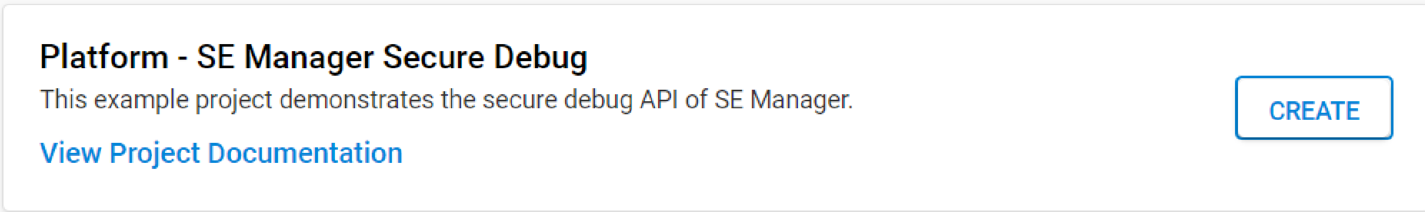
Note: This is an **IRREVERSIBLE** action, and should be the last step in production.

6.4 Secure Debug Unlock and Roll Challenge

6.4.1 SE Manager - Secure Debug Platform Example

Click the `View Project Documentation` link to open the `readme` file.

Note: The secure debug platform example can only run on the HSE device.



1. Use a secure debug lock device with matched Public Command Key.

```
. Get SE status... SL_STATUS_OK (cycles: 8496 time: 223 us)
+ The SE firmware version (MSB..LSB): 0001020E
+ Debug lock: Enabled
+ Device Erase: Disabled
+ Secure debug: Enabled
+ Secure boot: Disabled

+ Debug lock state: Locked

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

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

. The device is in secure debug lock state.
+ Press ENTER to issue a secure debug unlock or press SPACE to exit.
```

2. Press `ENTER` to unlock the device with debug mode request `0x3e`.

```
+ Creating a private certificate key in a buffer... SL_STATUS_OK (cycles: 202354 time: 5325 us)
+ Exporting a public certificate key from a private certificate key... SL_STATUS_OK (cycles: 199394 time: 5247 us)
+ Read the serial number of the SE and save it to access certificate... SL_STATUS_OK (cycles: 7084 time: 186 us)
+ Signing the access certificate with private command key... SL_STATUS_OK (cycles: 221849 time: 5838 us)
+ Request challenge from the SE and save it to challenge response... SL_STATUS_OK (cycles: 4418 time: 116 us)
+ Signing the challenge response with private certificate key... SL_STATUS_OK (cycles: 220833 time: 5811 us)
+ Creating an unlock token (DEBUG_MODE_REQUEST = 0x3e) to unlock the device... SL_STATUS_OK (cycles: 935778 time: 24625 us)
+ Get debug status to verify the device is unlocked... SL_STATUS_OK (cycles: 9017 time: 237 us)
+ Success to unlock the device!

. Get SE status... SL_STATUS_OK (cycles: 8683 time: 228 us)
+ The SE firmware version (MSB..LSB): 0001020D
+ Debug lock: Enabled
+ Device Erase: Enabled
+ Secure debug: Enabled
+ Secure boot: Disabled

+ Debug lock state: Unlocked

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

+ Non-secure, Invasive debug lock (DBGLOCK) current state: Unlocked
+ Non-secure, Non-invasive debug lock (NIDLOCK) current state: Unlocked
+ Secure, Invasive debug lock (SPIDLOCK) current state: Unlocked
+ Secure, Non-invasive debug lock (SPNIDLOCK) current state: Unlocked

. The device is in secure debug unlock state.
+ Issue a power-on or pin reset to re-enable the secure debug lock.
+ Press ENTER to roll the challenge to invalidate the current unlock token or press SPACE to exit.
```

3. Press `ENTER` to roll the challenge.

```
. Check and roll the challenge.
+ Request current challenge from the SE... SL_STATUS_OK (cycles: 4450 time: 117 us)
+ The current challenge (16 bytes):
  FA A7 AA 5E EF E6 18 23 E5 21 89 84 DB 7E 52 7D
+ Rolling the challenge... SL_STATUS_OK (cycles: 19757 time: 519 us)
+ Request rolled challenge from the SE... SL_STATUS_OK (cycles: 4628 time: 121 us)
+ The rolled challenge (16 bytes):
  5A 7A 81 CC 6E 46 C1 EF B4 A4 CA 7A DD A9 85 EB
+ Issue a power-on or pin reset to activate the rolled challenge.

. SE manager deinitialization... SL_STATUS_OK (cycles: 9 time: 0 us)
```

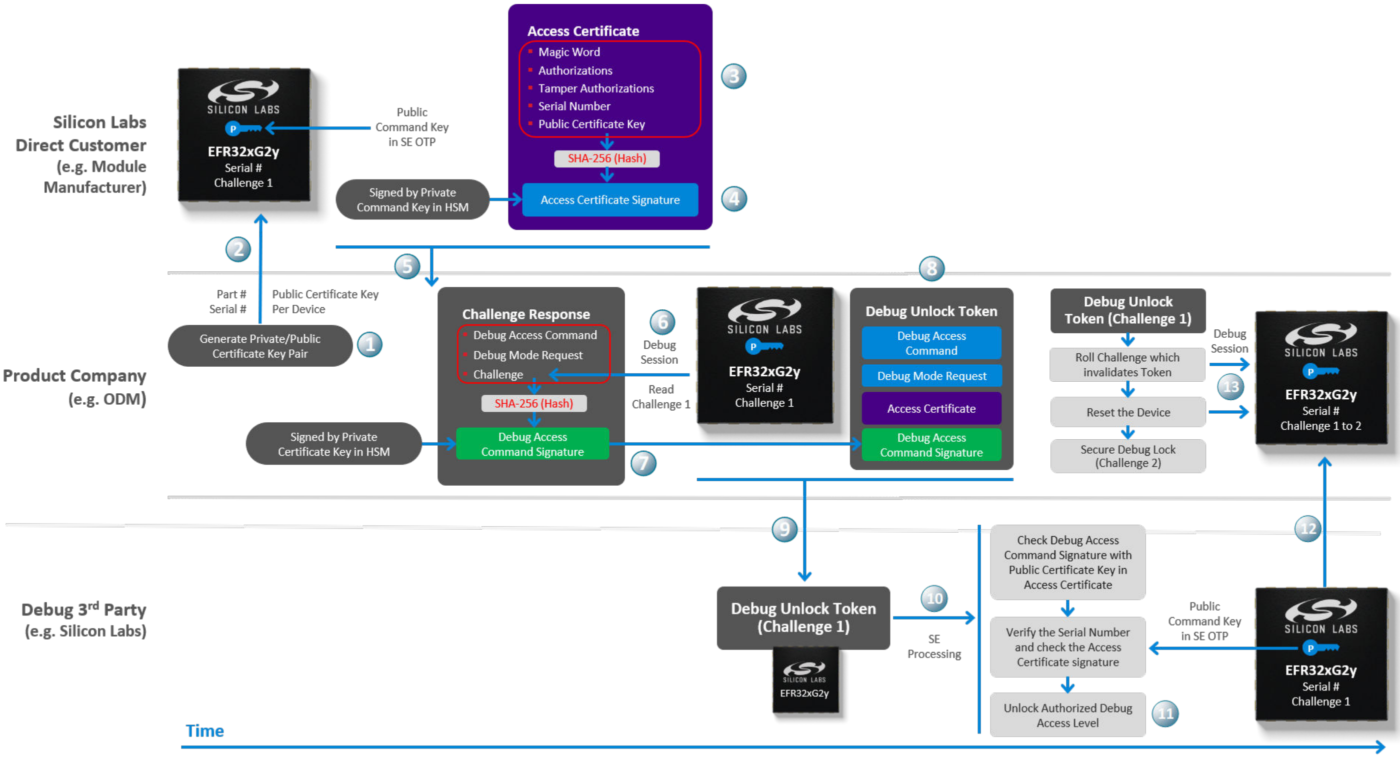
6.4.2 Simplicity Commander

The secure debug was designed with three organizations in mind:

- Direct Customer to whom Silicon Labs sells the chip. This chip has the Public Command Key installed in the SE OTP.
- That Direct Customer may be creating a white-labeled product for another company or a sub-component that goes into another company's product. The Product Company is the customer of the direct customer.
- The Debug 3rd Party could be anyone, internal or external, that the Product Company decides is qualified to debug the device.

Because the Public Command Key is installed into the SE OTP of a large number of devices and cannot be changed, the corresponding Private Command Key must be guarded by a very stringent process. If this Private Command Key is ever leaked, all the devices programmed with the corresponding Public Command Key will be compromised.

A secure debug unlock user case is described in the following figure.



The secure debug unlock flow moving across the time axis from left to right is explained below:

1. The Product Company creates a Private/Public Certificate Key pair for each device. Because the key pair is assigned only to a single device, the company may not need to protect the Private Certificate Key as securely as the Private Command Key by the Direct Customer.

   In this example, the Private/Public Certificate Key pair (`cert_key.pem` and `cert_pubkey.pem`) is generated by running the `util genkey` command.

```
commander util 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. The Public Certificate Key (`cert_pubkey.pem`) for each device is passed to the Silicon Labs Direct Customer. The part number and serial number are also required if Direct Customer cannot access the device.

   If necessary, run the `security status` command to get the device serial number.

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

```
SE Firmware version : 1.2.14
Serial number       : 0000000000000000014b457fffe045a32
Debug lock          : Enabled
Device erase        : Disabled
Secure debug unlock : Enabled
Tamper status       : OK
Secure boot         : Disabled
Boot status         : 0x20 - OK
DONE
```

3. The Direct Customer then places that Public Certificate Key in the access certificate. The access certificate is per device because it contains the unique device serial number. This certificate is generated once upon creation of the device, and thereafter, is generally only modified when the Private/Public Certificate Key pair is changed by the Product Company.

   Run the `security gencert` command with the following parameters from the Product Company to generate an unsigned access certificate (`access_certificate.extsign`) in Security Store:

   - Device part number
   - Device serial number
   - Public Certificate Key

```
commander security gencert --device EFR32MG21A010F1024 --deviceserialno 0000000000000000014b457fffe045a32
--cert-pubkey cert_pubkey.pem --extsign
```

```
Authorization file written to Security Store:
C:/Users/<username>/AppData/Local/SiliconLabs/commander/SecurityStore/
device_0000000000000000014b457fffe045a32/certificate_authorizations.json
Cert key written to Security Store:
C:/Users/<username>/AppData/Local/SiliconLabs/commander/SecurityStore/
device_0000000000000000014b457fffe045a32/cert_pubkey.pem
Created an unsigned certificate in Security Store:
C:/Users/<username>/AppData/Local/SiliconLabs/commander/SecurityStore/
device_0000000000000000014b457fffe045a32/access_certificate.extsign
DONE
```

   Note:

   - The `--extsign` option to create an unsigned access certificate is only available in Simplicity Commander Version 1.11.2 or above.
   - The unsigned access certificate is generated with the default certificate authorization file (`certificate_authorization.json`) which uses `0x0000003e` for Authorizations and `0x00000000` (HSE-SVM device) for Tamper Authorizations (Table 5.2 Elements of the Access Certificate on page 12).

4. The signing of the access certificate can be done by passing an unsigned access certificate to a Hardware Security Module (HSM) containing the Private Command Key.

   In this example, the OpenSSL is used to sign the access certificate (`access_certificate.extsign`) in Security Store with the Private Command Key (`command_key.pem`). The access certificate signature is in the `cert_signature.bin` file.

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

   Run the `util signcert` command with the following parameters to verify the signature and generate the signed access certificate (`access_certificate.bin`):
   - Unsigned access certificate
   - Access certificate signature
   - Public Command Key

```
commander util signcert access_certificate.extsign --cert-type access --signature cert_signature.bin
--verify command_pubkey.pem --outfile access_certificate.bin
```

```
R = D97E43FEA278207080D6D0808B46810C1167F123AF1CA9FAF2DE0F4322B97ACE
S = FEDFEA11A3C83AFFCD5293283B13A50580862B9F651AAE08012C2BFB6BA8E697
Successfully verified signature
Successfully signed certificate
DONE
```

   Note:
   - Put the required files in the same folder to run the command.
   - The `util signcert` command for access certificate is only available in Simplicity Commander Version 1.11.2 or above.
   - The access certificate signature can be in a Raw or Distinguished Encoding Rules (DER) format.

5. The access certificate is passed to the Product Company. The purpose of the access certificate is to grant overall debug access capabilities to the Product Company and authorize them to allow third parties to debug the device. The Product Company can now use the access certificate to generate the Debug Unlock Token. The same access certificate can be used to generate as many Debug Unlock Tokens as necessary without having to ever go back to the Direct Customer.
6. To create the Debug Unlock Token, a debug session must be started with the device and the challenge value (which is a random number `Challenge 1` in this example) should be read out to generate the challenge response.

   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 --unlock-param 1111 -o command_unsign.bin --nostore
--device EFR32MG21A010F1024 --serialno 440048205
```

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

   Note:
   - The data in the `--unlock-param` option are the bits 2 to 5 of debug mode request in the challenge response.
   - The default value `1111` (reset all debug options) is in place if the `security gencommand` command does not include the `--unlock-param` option.

7. The challenge response is then cryptographically hashed (SHA-256) to create a digest. The digest is then signed by the Private Certificate Key to generate the debug access command signature.

   The signing of the challenge response can be done by passing an unsigned challenge response to a Hardware Security Module (HSM) containing the Private Certificate Key.

   In this example, the OpenSSL is used to sign the challenge response (`command_unsign.bin`) with the Private Certificate Key (`cert_key.pem`). 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
```

8. Run the `security unlock` command with the access certificate (`access_certificate.bin`) from Direct Customer and debug access command signature (`command_signature.bin`) in step 7 to generate the Debug Unlock Token.

```
commander security unlock --cert access_certificate.bin --command-signature command_signature.bin
--unlock-param 1111 --device EFR32MG21A010F1024 --serialno 440048205
```

```
Certificate written to Security Store:
C:/Users/<username>/AppData/Local/SiliconLabs/commander/SecurityStore/
device_0000000000000000014b457fffe045a32/access_certificate.bin
R = 67B51151F1E5F1BB9A49EC8D5885B221BD3D331D53741EEF54F81F0F3CB40455
S = 066C6AB5EDE3AE784DB1F75F44C5CA931736116D5A2104DBF44BC77ED8F49282
Command signature is valid
Secure debug successfully unlocked
Command unlock payload was stored in Security Store
DONE
```

Note:

- Put the required files in the same folder to run the command.
- The debug access command signature can be in a Raw or Distinguished Encoding Rules (DER) format.
- It requires Simplicity Commander Version 1.11.2 or above to support signature in DER format.
- The data in the `--unlock-param` option are the bits 2 to 5 of debug mode request in the Debug Unlock Token. This value **MUST** be equal to the value of `--unlock-param` option in step 6.
- The default value `1111` (reset all debug options) is in place if the `security unlock` command does not include the `--unlock-param` option.

9. **(Alternative)** The key protection is not required if the Private Certificate Key is ephemeral. Steps 6 to 8 can be implemented by running the `security unlock` command with the access certificate (`access_certificate.bin`) from the Direct Customer and Private Certificate Key (`cert_key.pem`) to generate the Debug Unlock Token.

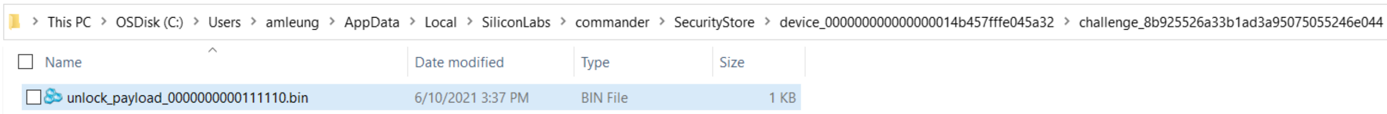
```
commander security unlock --cert access_certificate.bin --cert-privkey cert_key.pem --unlock-param 1111
--device EFR32MG21A010F1024 --serialno 440048205
```

```
Certificate written to Security Store:
C:/Users/<username>/AppData/Local/SiliconLabs/commander/SecurityStore/
device_0000000000000000014b457fffe045a32/access_certificate.bin
Cert key written to Security Store:
C:/Users/<username>/AppData/Local/SiliconLabs/commander/SecurityStore/
device_0000000000000000014b457fffe045a32/cert_pubkey.pem
Created unsigned unlock command
Signed unlock command using
C:/Users/<username>/AppData/Local/SiliconLabs/commander/SecurityStore/
device_0000000000000000014b457fffe045a32/cert_key.pem
Secure debug successfully unlocked
Command unlock payload was stored in Security Store
DONE
```

Note:

- The data in the `--unlock-param` option are the bits 2 to 5 of debug mode request in the Debug Unlock Token.
- The default value `1111` (reset all debug options) is in place if the `security unlock` command does not include the `--unlock-param` option.

10. The Debug Unlock Token (aka `Command unlock payload`) file (`unlock_payload_0000000000111110.bin`, where `0000000000111110` is the value of debug mode request) is stored in the Security Store. The location in Windows is `C:\Users\<PC user name>\AppData\Local\SiliconLabs\commander\SecurityStore\device_<Serial number>\challenge_<Challenge value>`.



Users can also use the `security getpath` command to get the path of the Security Store or a specified device.

```
commander security getpath
```

```
C:/Users/<username>/AppData/Local/SiliconLabs/commander/SecurityStore
DONE
```

```
commander security getpath --deviceserialno 000000000000000014b457fffe045a32
```

```
C:/Users/<username>/AppData/Local/SiliconLabs/commander/SecurityStore/
device_000000000000000014b457fffe045a32
DONE
```

11. The Debug Unlock Token and the device are now delivered to the Debug 3rd Party.

Run the `security gencommand` command to create the Security Store to place the Debug Unlock Token file.

```
commander security gencommand --action debug-unlock --device EFR32MG21A010F1024 --serialno 440048205
```

```
Unsigned command file written to Security Store:
C:/Users/<username>/AppData/Local/SiliconLabs/commander/SecurityStore/
device_000000000000000014b457fffe045a32/challenge_8b925526a33b1ad3a95075055246e044/
unlock_command_to_be_signed16_06_2021.bin
DONE
```

Copy the Debug Unlock Token file (`unlock_payload_0000000000111110.bin`) from Product Company to the Windows Security Store `challenge_<Challenge value>` folder located in `C:\Users\<PC user name>\AppData\Local\SiliconLabs\commander\SecurityStore\device_<Serial number>\challenge_<Challenge value>`.

12. The device compares the Debug Unlock Token contents with its internal serial number, challenge value, and Public Command Key to determine the token's authenticity. If authentic, it will execute the debug access command to unlock the device; otherwise, it will ignore the command.

Run the `security unlock` command to unlock the device.

```
commander security unlock --unlock-param 1111 --device EFR32MG21A010F1024 --serialno 440048205
```

```
Unlocking with unlock payload:
C:/Users/<username>/AppData/Local/SiliconLabs/commander/SecurityStore/
device_000000000000000014b457fffe045a32/challenge_8b925526a33b1ad3a95075055246e044/
unlock_payload_0000000000111110.bin
Secure debug successfully unlocked
DONE
```

Note:

- If the security store has multiple tokens for the selected device, use `--unlock-param` option to specify which unlock token is chosen to unlock the device.
- Simplicity Commander will only use the token with value `1111` (error if not available) from the security store to unlock the device if the `security unlock` command does not include the `--unlock-param` option.

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

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

```
SE Firmware version : 1.2.14
Serial number       : 000000000000000014b457fffe045a32
Debug lock          : Enabled
Device erase        : Disabled
Secure debug unlock : Enabled

Debug lock state: Unlocked

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

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

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

14. The Debug 3rd Party can now use this same Debug Unlock Token to unlock the device (step 12), over and over again after each power-on or pin reset, until they have finished debugging the device.
15. Once the Debug 3rd Party has finished debugging, they will send the device back to the Product Company.
16. Once the Product Company receives the device, they will immediately start a debug session, roll the challenge (from `Challenge 1` to `Challenge 2` in this example), and put the device back into the secure debug lock state. Rolling the challenge will effectively invalidate any Debug Unlock Token that has been previously given to any third party.

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

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

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

    The unlock token is invalidated after rolling the challenge because any previously issued Debug Unlock Token now contains a different challenge value (`Challenge 1`) than the challenge value currently in the device (`Challenge 2`).

    The validation process of any previously issued Debug Unlock Token will always fail until a new Debug Unlock Token is issued with a current matching challenge value (`Challenge 2`).

Note: Direct Customer can directly use the Private Command Key on the connected chip to generate the Debug Unlock Token in Security Store. But it has a high risk (cannot use HSM) to leak the Private Command Key to a 3rd party when using this approach.

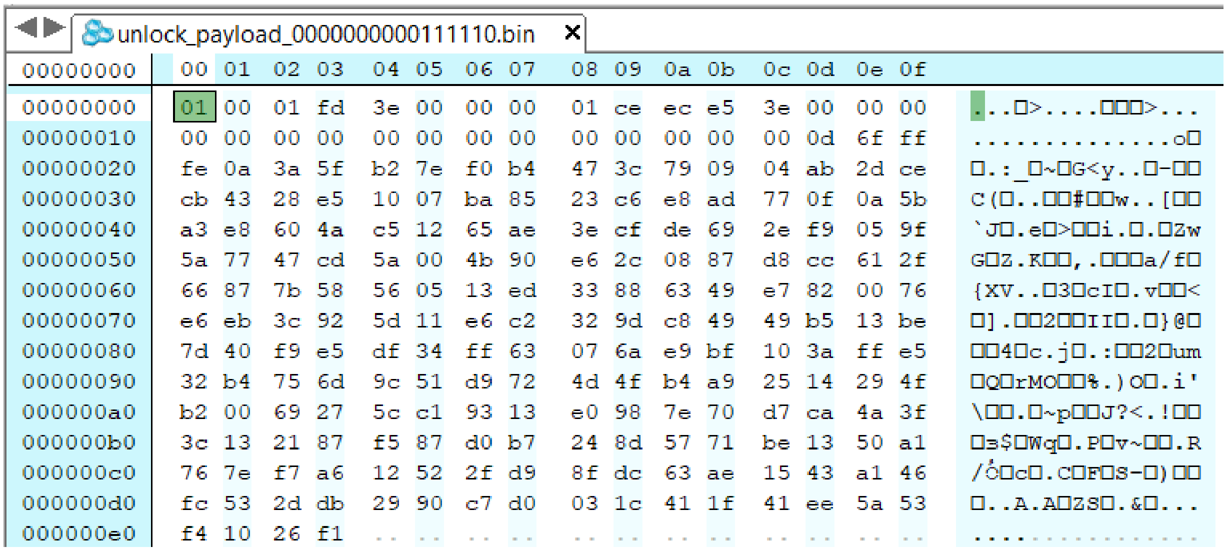
```
commander security unlock --command-key command_key.pem --unlock-param 1111 --device EFR32MG21A010F1024
--serialno 440048205
```

```
Authorization file written to Security Store:
C:/Users/<username>/AppData/Local/SiliconLabs/commander/SecurityStore/device_000000000000000014b457fffe045a32/
certificate_authorizations.json
Generating ECC P256 key pair...
Cert public key stored at:
C:/Users/<username>/AppData/Local/SiliconLabs/commander/SecurityStore/device_000000000000000014b457fffe045a32/
cert_pubkey.pem
Cert private key stored at:
C:/Users/<username>/AppData/Local/SiliconLabs/commander/SecurityStore/device_000000000000000014b457fffe045a32/
cert_key.pem
Command public key stored at:
C:/Users/<username>/AppData/Local/SiliconLabs/commander/SecurityStore/device_000000000000000014b457fffe045a32/
command_pubkey.pem
Command private key stored at:
C:/Users/<username>/AppData/Local/SiliconLabs/commander/SecurityStore/device_000000000000000014b457fffe045a32/
command_key.pem
Certificate was signed with key:
C:/Users/<username>/AppData/Local/SiliconLabs/commander/SecurityStore/device_000000000000000014b457fffe045a32/
command_key.pem
Certificate written to Security Store:
C:/Users/<username>/AppData/Local/SiliconLabs/commander/SecurityStore/device_000000000000000014b457fffe045a32/
access_certificate.bin
Created unsigned unlock command
Signed unlock command using
C:/Users/<username>/AppData/Local/SiliconLabs/commander/SecurityStore/device_000000000000000014b457fffe045a32/
cert_key.pem
Secure debug successfully unlocked
Command unlock payload was stored in Security Store
DONE
```

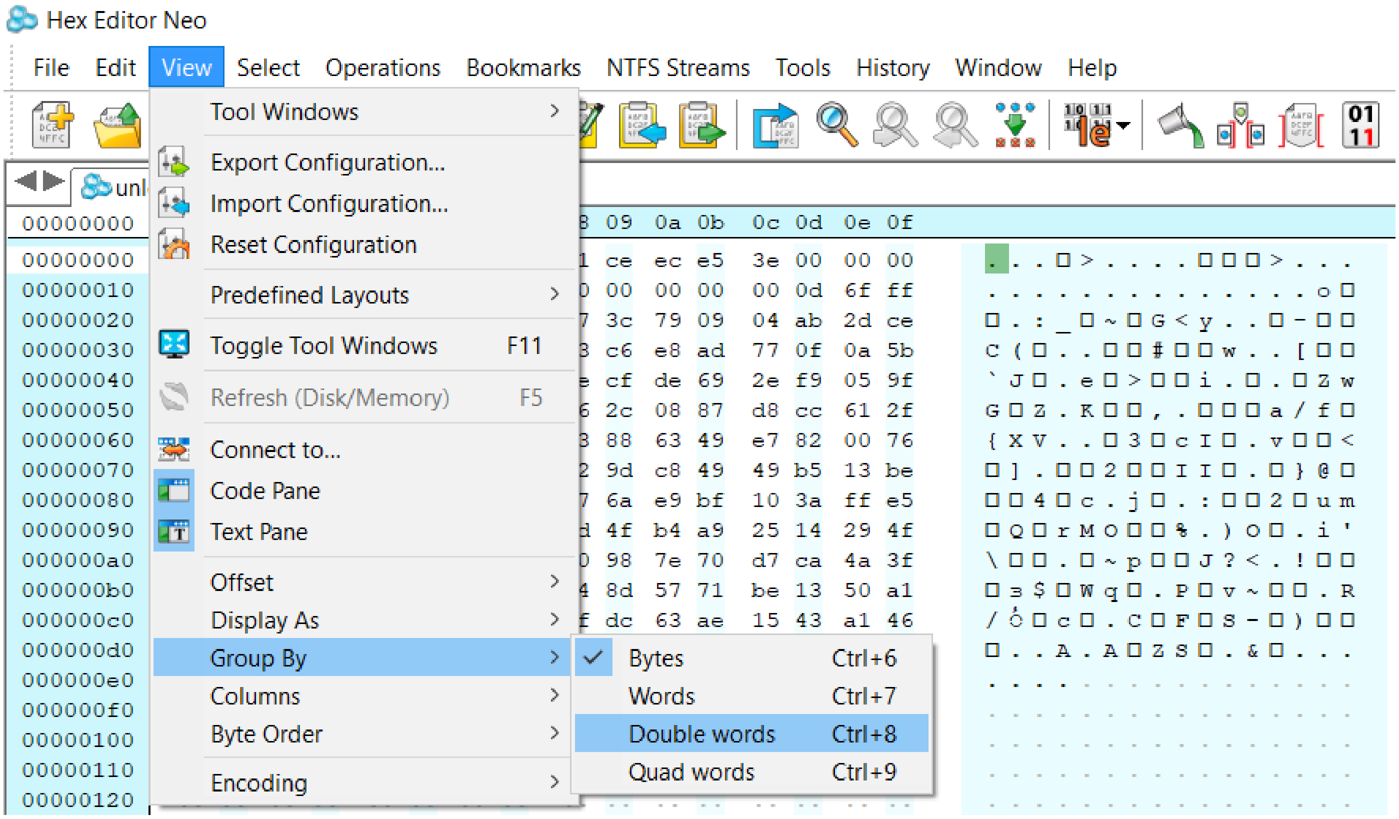
6.4.3 Simplicity Studio

Use the Debug Unlock Token file (`unlock_payload_0000000000111110.bin`) generated in 6.4.2 Simplicity Commander steps 8 or 9 to unlock the device with Simplicity Studio.

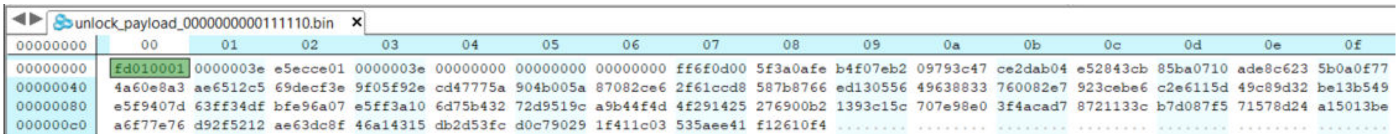
1. Open the `unlock_payload_0000000000111110.bin` file with the Hex Editor Neo.



2. Click **View** to open the context menu, and then select **Group By** → **Double words** to convert the token into a little-endian format.



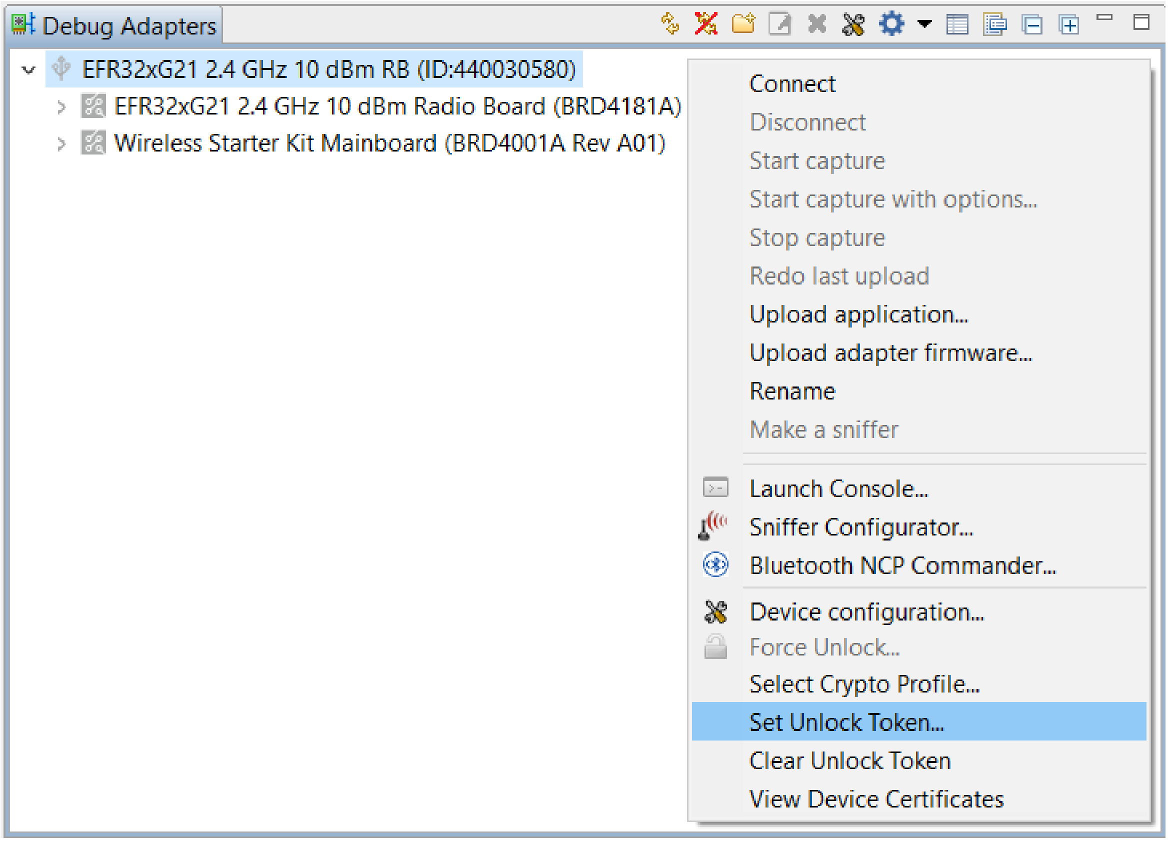
3. Select all (Ctrl+A) and copy (Ctrl+C) the Debug Unlock Token to a text editor.



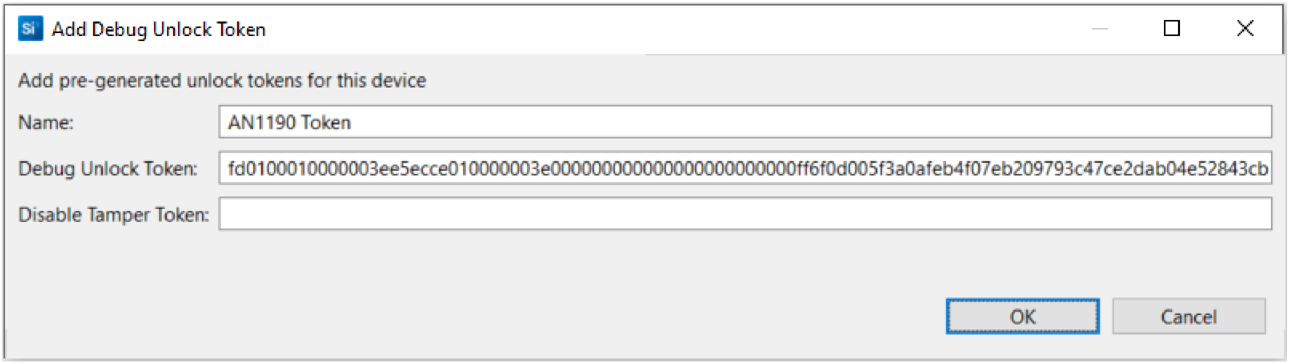
4. Use the text editor to remove all the spaces from the token.

```
fd0100010000003ee5ecce010000003e000000000000000000000000ff6f0d00...
```

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



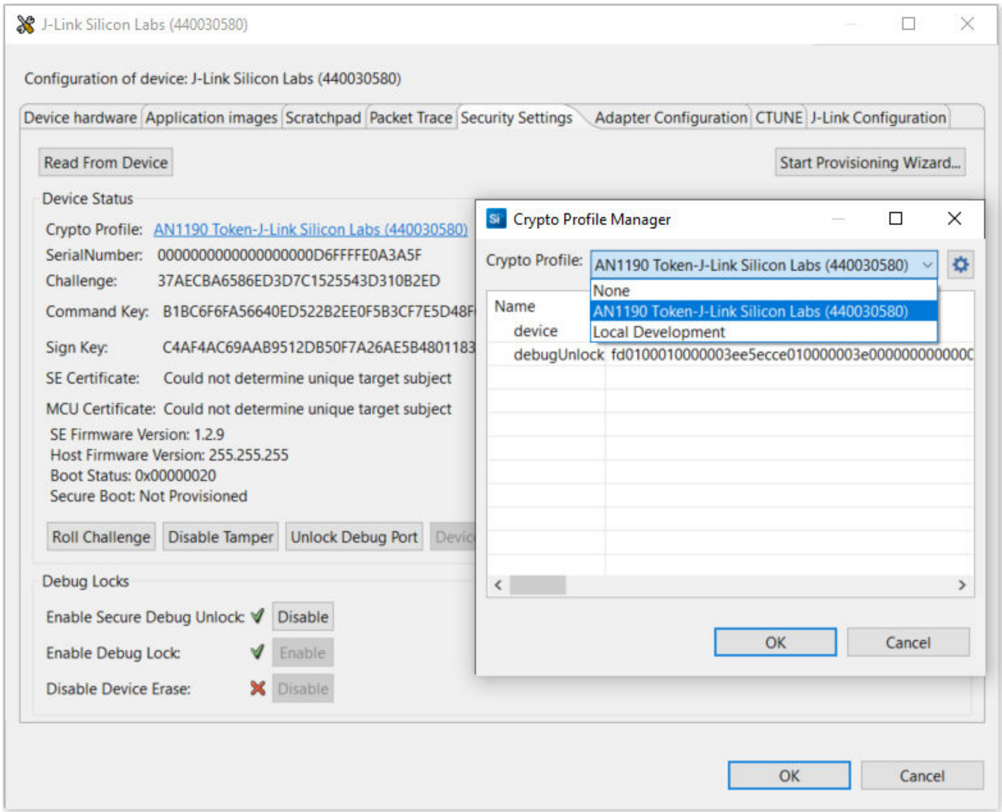
6. Click **Set Unlock Token** to open the **Add Debug Unlock Token** dialog box. Enter the name (e.g., `AN1190 Token`) for this Debug Unlock Token, and copy the content in step 4 to the **Debug Unlock Token:** box. Click [**OK**] to confirm and exit.



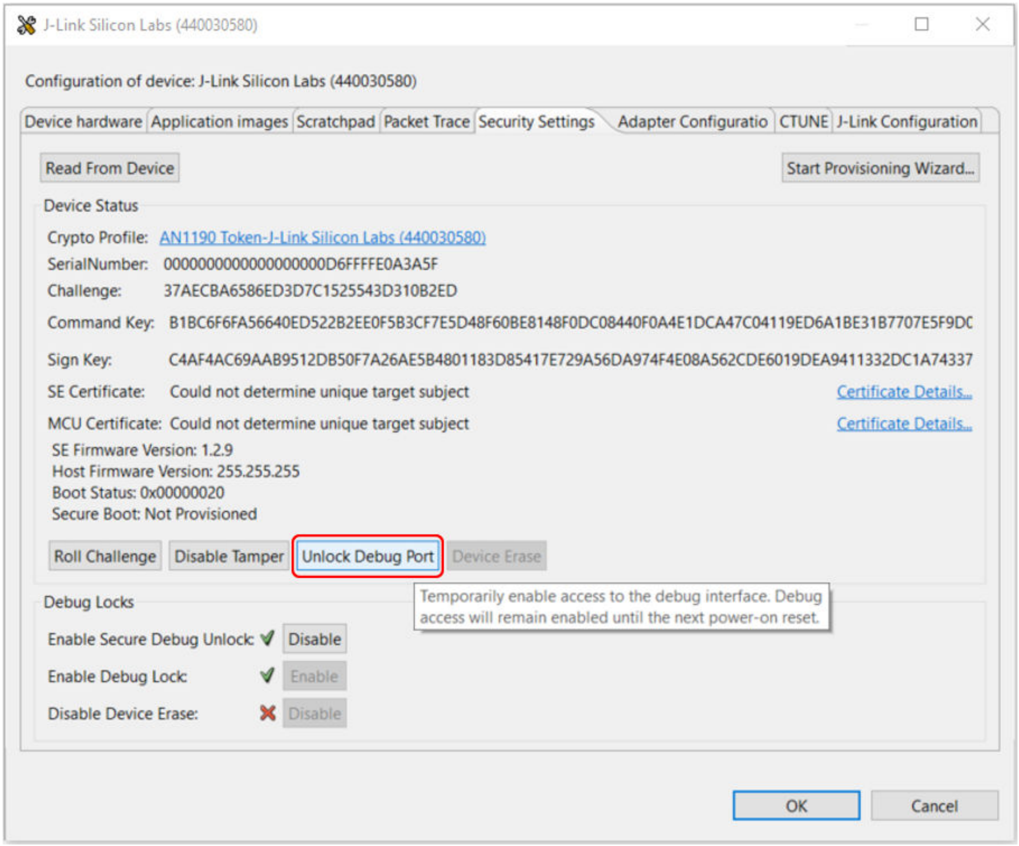
Note: The Simplicity Studio can only keep one Debug Unlock Token on each WSTK.

7. Open **Security Settings** of the selected device as described in 6.1.1 Using Simplicity Studio.

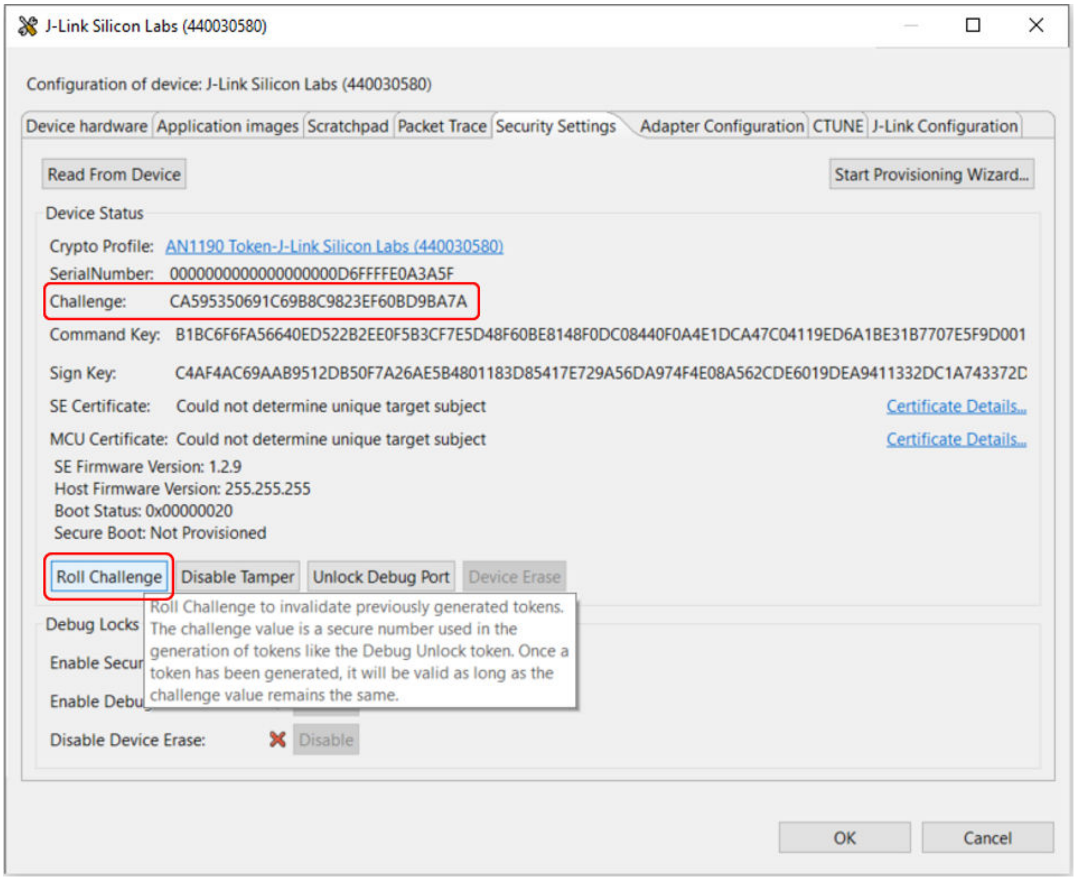
8. The token added in step 6 should display on the **Crypto Profile:** field. If not, click the link next to **Crypto Profile:** to select the token from the **Crypto Profile Manager** drop-down list. The Simplicity Studio will automatically add the WSTK J-Link serial number (`-J-Link Silicon Labs (serial number)`) to the token's name.



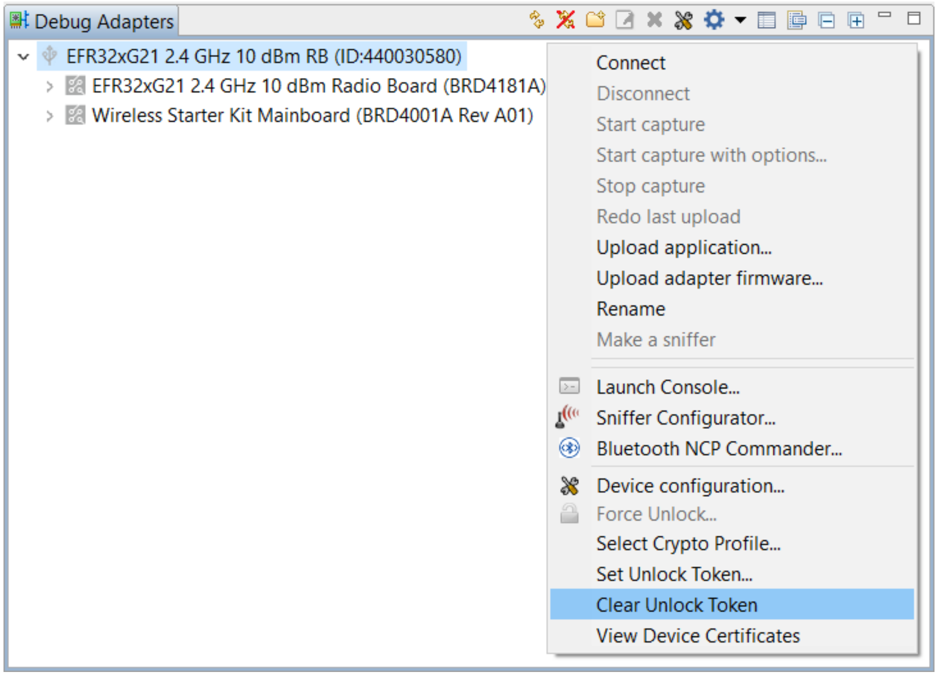
9. Click [**Unlock Debug Port**] to use the token in **Crypto Profile:** to unlock the device (invalid token will display an error message). The device stays in the unlock state until the next power-on or pin reset. Click [**OK**] to exit.



10. The Simplicity IDE will automatically use the selected Debug Unlock Token in **Crypto Profile** for debugging and flashing. For other IDE, the device should unlock again (step 9) after power-on or pin reset. After finished debugging, open the **Security Settings** of the selected device as described in 6.1.1 Using Simplicity Studio.
11. Click [**Roll Challenge**] to generate a new challenge value to invalidate the Debug Unlock Token added in step 6. Click [**OK**] to exit.



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

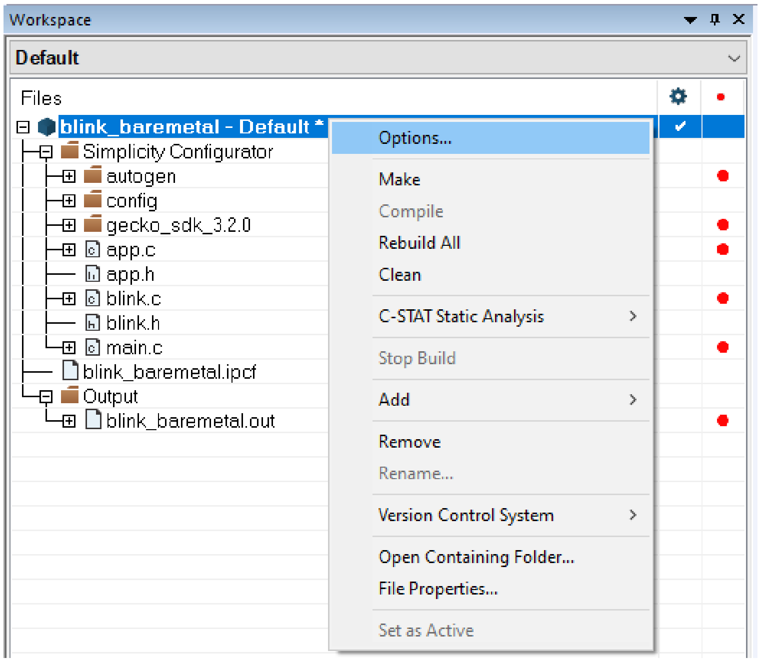


13. Click [**Clear Unlock Token**] to delete the WSTK Debug Unlock Token from Simplicity Studio.

6.4.4 IAR

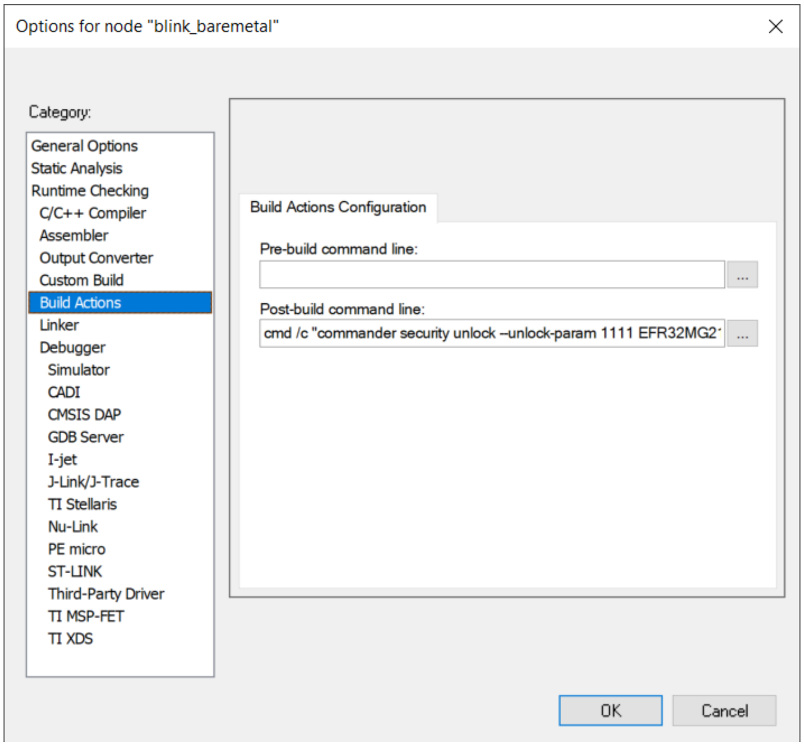
Use the Debug Unlock Token file (`unlock_payload_0000000000111110.bin`) in the Security Store (6.4.2 Simplicity Commander step 11) to unlock the device with IAR (Windows).

1. The Windows environment variable `PATH` should include the folder (`C:\SiliconLabs\SimplicityStudio\v5\developer\adapter_packs\commander`) that locates the `commander.exe` of Simplicity Commander.
2. Right-click the project in the workspace, and then click **Options...**.

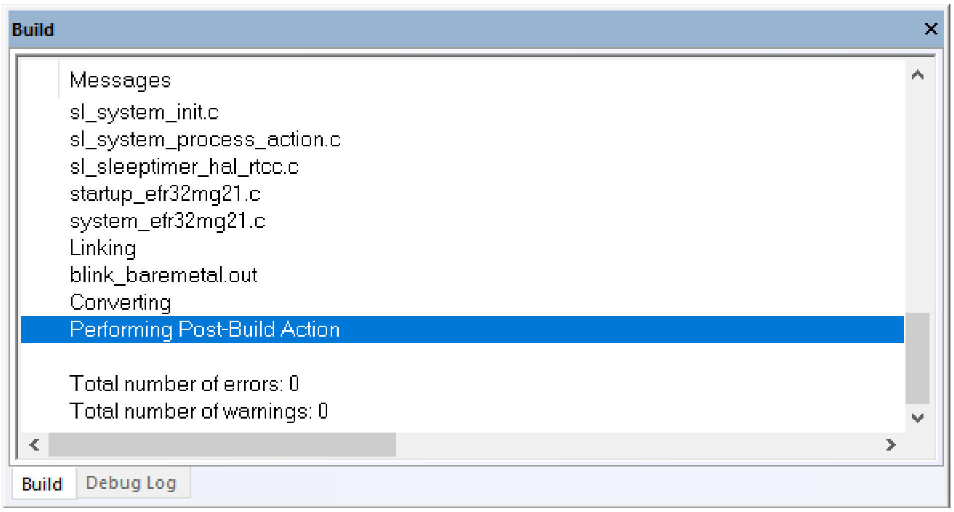


3. Click **Build Actions** to open the **Build Actions Configuration** dialog box. Enter the phrase below to the **Post-build command line:** box. Click [**OK**] to exit.

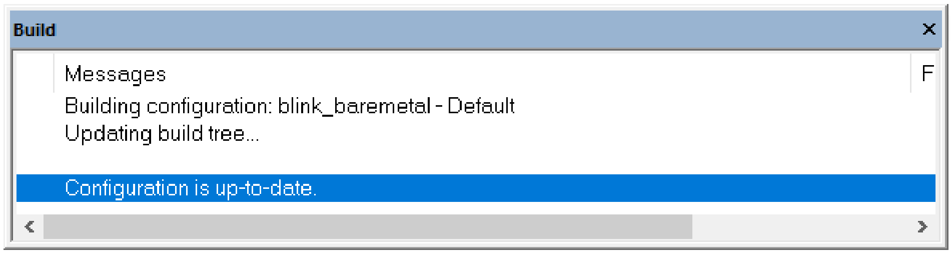
```
cmd /c "commander security unlock --unlock-param 1111 EFR32MG21A010F1024 --serialno 440048205 > $PROJ_DIR$\log.txt 2>&1"
```



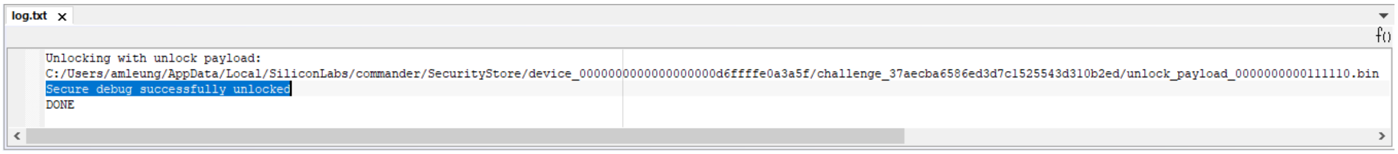
4. After building the project, the `security unlock` in the **Post-build command** unlocks the device using the Debug Unlock Token in Security Store. The device stays in the unlock state until the next power-on or pin reset.



Note: If the project is already up-to-date, it will not invoke the **Post-build command** to unlock the device. Use a dummy edit (add space or newline) on one of the source files in the project to trigger the build action.



5. The `> $PROJ_DIR$\log.txt 2>&1` redirects the `security unlock` command output to the `log.txt` file in the IAR project folder.



6.5 Permanent Debug Lock

6.5.1 Simplicity Commander

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

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

```
SE Firmware version : 1.2.14
Serial number       : 0000000000000000014b457fffe045a32
Debug lock          : Disabled
Device erase        : Enabled
Secure debug unlock : Disabled
Tamper status       : OK
Secure boot         : Disabled
Boot status         : 0x20 - OK
DONE
```

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

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

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

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

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

```

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

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

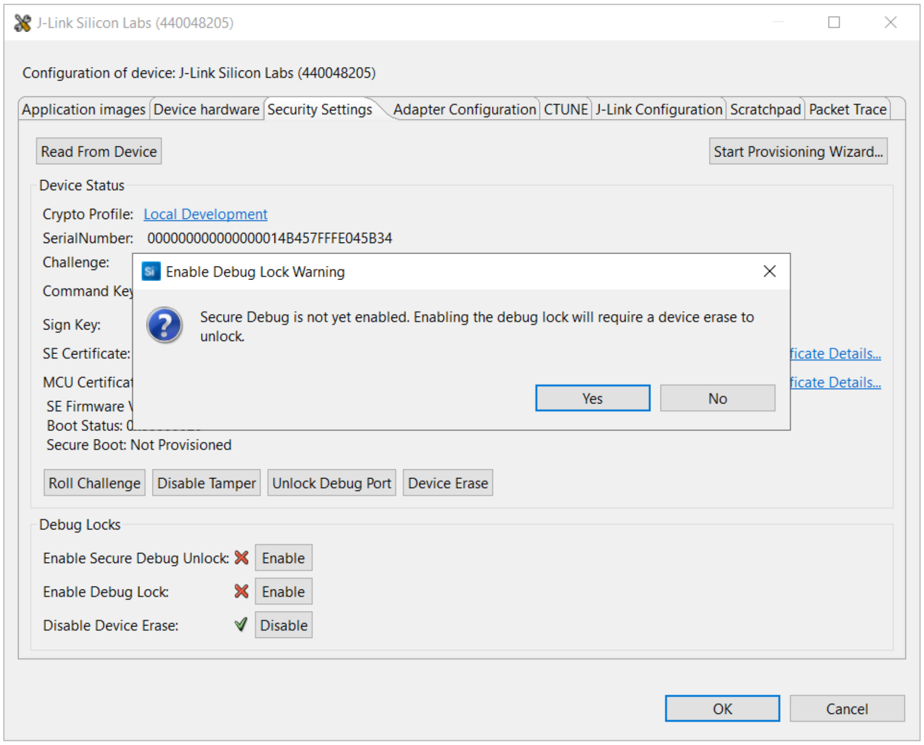
4. Run the `security status` command again to check the device configuration.

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

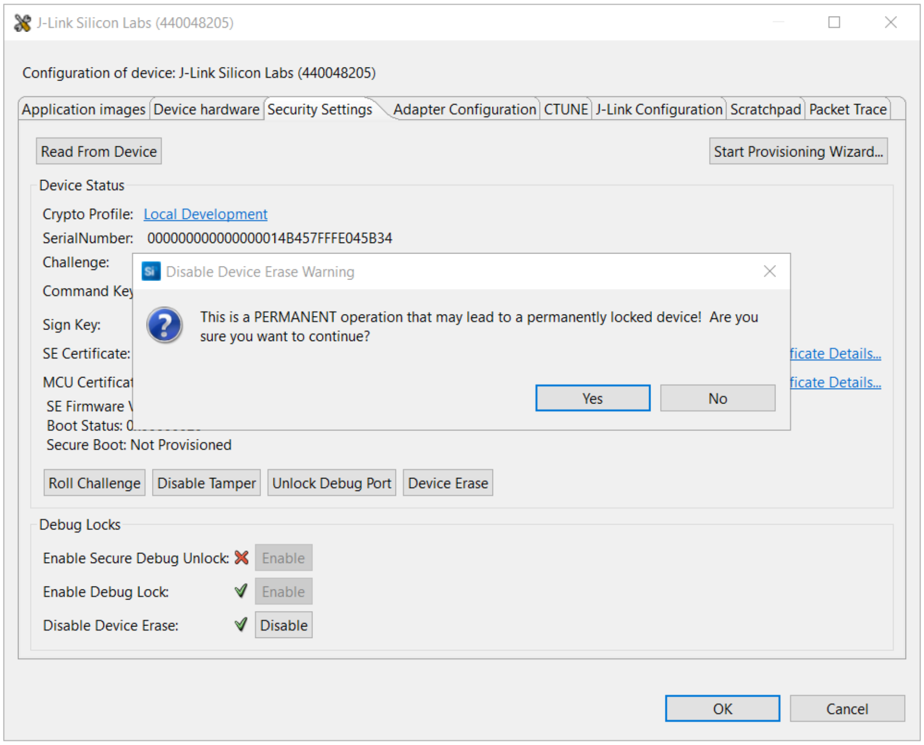
```
SE Firmware version : 1.2.14
Serial number       : 0000000000000000014b457fffe045a32
Debug lock          : Enabled
Device erase        : Disabled
Secure debug unlock : Disabled
Tamper status       : OK
Secure boot         : Disabled
Boot status         : 0x20 - OK
DONE
```

6.5.2 Simplicity Studio

1. Open **Security Settings** of the selected device as described in 6.1.1 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.



3. Click [**Disable**] next to **Disable Device Erase:** to disable the device erase. The following **Disable Device Erase Warning** is displayed. Click [**Yes**] to confirm. This configures a permanent debug lock.



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

7. Precautions

7.1 Device Erase for Secure Debug

Disable the Device Erase is mandatory for secure debug as described in the following table.

Secure Debug	Device Erase	Debug Lock	State	Description
Enabled	Enabled	Enabled	Insecure debug lock (1)	The device will return to the default debug lock properties after applying the standard debug unlock. (2)
Enabled	Disabled (3)	Enabled	Secure debug lock	The device cannot be unlocked using the Erase Device command. The device will change to the permanent debug lock state if disabling the Secure Debug property. (4)

Note:

1. This state is only for secure debug testing.
2. See 5.2 Standard Debug Unlock.
3. This is an **IRREVERSIBLE** action and should be disabled **AFTER** the secure debug is enabled.
4. See 4.4 Permanent Debug Lock.

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

```

================================================================================
WARNING: Device erase is disabled and secure debug access is locked.
If disabling secured debug access, there is no way to regain debug access to this device if continuing
with this command.
Type 'continue' and hit enter to proceed or Ctrl-C to abort:
================================================================================
continue
Secure debug unlock was disabled
DONE
```

7.2 Secure Boot and Debug Lock

The following table describes the different debug lock scenarios on the secure boot-enabled device.

Secure Debug	Device Erase	Debug Lock	State	Recover from Secure Boot Failure
Disabled	Enabled	Disabled	Standard debug unlock	Flash a correctly signed image.
Disabled	Enabled	Enabled	Standard debug lock	Flash a correctly signed image after standard debug unlocking the device.
Disabled	Disabled	Enabled	Permanent debug lock	There is no way to recover the device. Make sure the programmed image is correctly signed before locking the device.
Enabled	Disabled	Enabled	Secure debug lock	Flash a correctly signed image after secure debug unlocking the device.

Note: See section "Recover Devices when Secure Boot Fails" in AN1218: Series 2 Secure Boot with RTSL to flash a correctly signed image on different debug lock scenarios.

8. Failure Analysis

The following table describes the different scenarios when returning a Series 2 device to Silicon Labs for failure analysis.

State	Secure Boot Disabled	Secure Boot Enabled (2)
Standard debug unlock	Device erase is not necessary for failure analysis.	Device erase is not necessary, but a correctly signed image is required to perform failure analysis.
Standard debug lock	Device erase is required to perform failure analysis.	Require device erase and correctly signed image to perform failure analysis.
Permanent debug lock	Cannot perform failure analysis.	Cannot perform failure analysis.
Secure debug lock (1)	Require debug unlock token to perform failure analysis.	Require debug unlock token and correctly signed image to perform failure analysis.

Note:

1. Follow the procedures in 6.4.2 Simplicity Commander to generate a valid debug unlock token for each device returned to Silicon Labs for failure analysis.
2. Secure boot enabled devices, especially with secure boot failure, may limit Silicon Labs' ability to determine the root cause of failure.

9. Revision History

Revision 0.8

February 2023

- Fixed a typo (`.sec` to `.seu`) in 3.1 Overview.
- Moved 4.2 Standard Debug Unlock.
- Added 4.6 Debug Lock State Transition.
- Added `sl_se_set_debug_options` to 4.7 Debug Lock Command Reference.
- Updated note in Figure 5.3 Debug Mode Request on page 11 for TrustZone.
- Updated note in Table 5.2 Elements of the Access Certificate on page 12 and Figure 5.5 Authorizations on page 13 for debug option.
- Updated 5.3.4 Debug Access Flow step 8 for debug option.
- Added 5.3.5 TrustZone Debug Authentication.
- Updated some examples in 6.1 Overview to SE firmware v1.2.14.
- Added Permanent debug lock to 6.1 Overview.
- Updated 6.1.2 Using Simplicity Commander to v1.14.2.
- Updated 6.1.4 Using Platform Examples to GSDK v4.2.1.
- Updated 6.3.2 SE Manager - Secure Debug Platform Example to GSDK v4.2.1.
- Updated 6.3.3 Simplicity Commander for TrustZone-unaware and TrustZone-aware applications.
- Updated 6.4.1 SE Manager - Secure Debug Platform Example to GSDK v4.2.1.
- Updated 6.4.2 Simplicity Commander with `--unlock-param` option.
- Added 6.5 Permanent Debug Lock.

Revision 0.7

June 2022

- Updated table and note in 1. Series 2 Device Security Features.
- Replaced Device Compatibility with SE Firmware in 1. Series 2 Device Security Features.
- Added Standard Debug Unlock to 4.1 Overview.
- Updated 5.2 Standard Debug Unlock.
- Added 8. Failure Analysis.

Revision 0.6

March 2022

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

Revision 0.5

January 2022

- Added UG489 to the table in 1.2 Key Reference.
- Added CPMS information to 2.1 Debug Lock.
- Updated 3.1 Overview with Windows folder for GSDK v4.0 and higher.
- Added SE Manager Secure debug lock example to Table 6.1 Series 2 Debug Lock and Debug Unlock Examples on page 18.
- Updated step 2 in 6.3.1 SE Manager - Key Provisioning Platform Example.
- Added 6.3.2 SE Manager - Secure Debug Platform Example to 6.3 Provision Public Command Key and Secure Debug Lock.
- Replaced `gbl keyconvert` with `util keytotoken` in 6.3.4 Simplicity Studio.
- Moved step 6 to step 4 in 6.4.4 IAR.
- Updated 7.2 Secure Boot and Debug Lock.

Revision 0.4

September 2021

- Formatting updates for source compatibility.
- Added revised terminology to 1. Series 2 Device Security Features and use this terminology throughout the document.
- Updated Device Compatibility.
- Removed terminology in 3. Secure Engine Subsystem.
- Removed Table 4.2.
- Removed deprecated em lib API in Table 4.6 Debug Lock Command Reference on page 9 and Table 5.4 Debug Unlock Command Reference on page 17.
- Replaced Debug Access Command with 5.3.1 Debug Unlock Token.
- Revised Examples chapter to the latest Simplicity Studio and Simplicity Commander version, updated the content.
- Added 6.4.3 Simplicity Studio and 6.4.4 IAR examples to 6.4 Secure Debug Unlock and Roll Challenge.
- Added 7. Precautions.

Revision 0.3

September 2020

- Added EFR32BG21B and EFR32MG21B to Device Compatibility.
- Modified Secure Element Subsystem for SE with Secure Vault devices.
- Added Secure Element Manager to 3.2.1 Mailbox.
- Added Secure Element Manager APIs to Table 4.6 Debug Lock Command Reference on page 9.
- Updated content in 5.3 Secure Debug Unlock.
- Added Secure Element Manager APIs to Table 5.4 Debug Unlock Command Reference on page 17.
- Updated the figures in 6.1.1 Using Simplicity Studio to Simplicity Studio v5.
- Updated Simplicity Commander version to 1.9.2 in 6.1.2 Using Simplicity Commander.
- Added 6.1.4 Using Platform Examples section.
- Updated content in Secure Debug Unlock example.
- Added Secure Element Manager example in 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

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.

SILICON LABS

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

www.silabs.com