AN1218: Series 2 Secure Boot with RTSL

This application note describes the design of Secure Boot with RTSL (Root of Trust and Secure Loader), which was introduced with EFR32 Series 2 MCUs and Wireless MCUs. It also provides instructions on how to implement the Secure Boot process.

KEY POINTS

- Compares the Secure Boot process in Series 1 and Series 2 devices
- Describes the Series 2 Secure Boot with RTSL components and process
- Provides instructions on configuring a Series 2 device for the Secure Boot process
- Describes two methods to recover devices

1 Introduction to the Secure Boot Process

The purpose of Secure Boot is to protect the integrity of the behavior of the system. Because the behavior of the system is defined by the firmware running on it, Secure Boot acts to ensure the authenticity and integrity of the firmware. Secure Boot is a foundational component of platform security, and without it other security aspects such as secure storage, secure transport, secure identity, and data confidentiality can often be subverted through the injection of malicious code.

Secure Boot works as a process by which each piece of firmware is validated for authenticity and integrity before it is allowed to run. Each authenticated module can also validate additional modules before executing them, forming a chain of trust. If any module fails its security check, it is not allowed to run, and program control will typically stall in the validating module. In most lightweight IoT systems, the behavior of a Secure Boot failure is to cause the device to stop working until an authentically signed image can be loaded onto it. Whereas this may seem extreme, it is a better outcome than a smart light bulb being repurposed to mine crypto-currency, or a smart speaker being repurposed as a surveillance device on the end user's private conversations.

The first link in the chain of trust is the root of trust. This is often the weakest link in the secure boot chain because the root of trust itself is not checked for authenticity or integrity. The security strength of the root of trust lies in its immutability. The strongest roots of trust have their firmware origin in ROM and use a sign key that is also located in ROM.

EFR32 Series 1 and Series 2 devices both use a two-stage boot design consisting of a non-upgradable first stage root of trust followed by an upgradable second stage. In Series 1 devices, the root of trust (also called the first-stage bootloader) is in flash rather than ROM, and the upgradable portion (the main bootloader) is checked for integrity using a CRC32 checksum, but is not checked for authenticity using a sign key. In Series 2 devices, the root of trust is in ROM, and the upgradable portion is checked both for integrity and authenticity. The following figure illustrates the secure boot process on Series 2 devices.

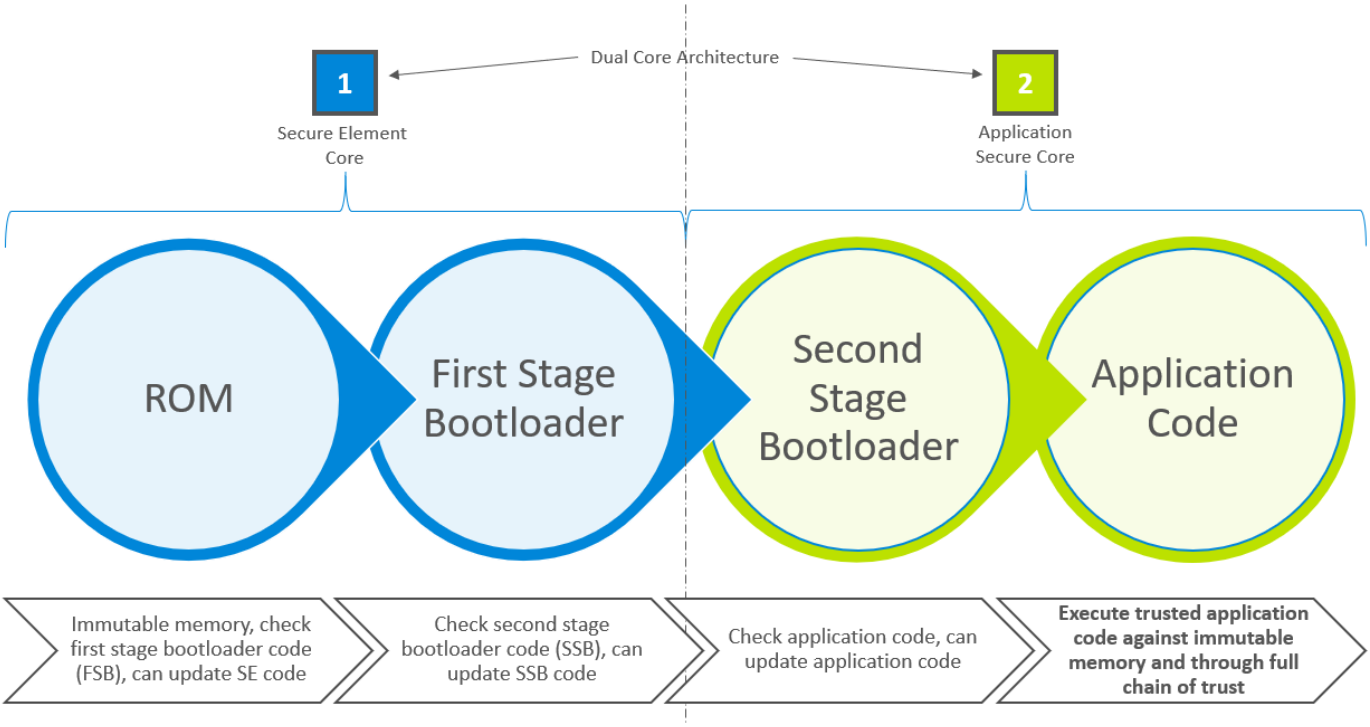


Figure 1.1. Series 2 Secure Boot Process

1.1 Secure Boot in Series 1 Devices

The Secure Boot process for Series 1 devices originates in flash, typically with the execution of the first stage of Gecko Bootloader. The first stage of Gecko Bootloader checks to see if an upgrade is pending for the second stage of Gecko Bootloader. If so, it processes the upgrade of the second stage and then executes it. Otherwise, it just executes the second stage. If Secure Boot is enabled, the second stage of Gecko Bootloader checks the integrity and authenticity of the application image before executing it. If the integrity check fails, program control remains in the second stage bootloader.

UG266: Gecko Bootloader User's Guide details the procedure for generating and downloading signed firmware images using Simplicity Commander.

1.2 Secure Boot in Series 2 Devices

The Secure Boot process for Series 2 devices originates in ROM contained in the Secure Element security co-processor (SE). The SE boots first out of reset while the host MCU is held in reset. Once the SE has completed its internal secure boot process and completed any pending firmware upgrades for itself, it checks to see if Secure Boot is enabled for the host application. If Secure Boot for the host is not enabled, the SE releases the host MCU from reset and reconfigures itself to a low power state, monitoring its mailbox interface for future commands from the host. If Secure Boot for the host is enabled, the SE calculates and validates the signature of the host application image against the public sign key stored in the SE one-time programmable memory (OTP). If the validation succeeds, the SE releases the host MCU from reset and the application will be allowed to execute. If the validation fails, the host MCU remains held in reset.

1.3 About the Sign Key and Secure Boot Enable Flag

In Series 2 devices, the sign key and the Secure Boot Enable flag are both located in immutable OTP. This means that once either is programmed, its respective value cannot be changed. Once the sign key is provisioned, it remains provisioned to that key value for the life of the device. Once Secure Boot is enabled, it remains enabled for the life of the device. Both of these assignment operations are irrevocable.

The sign key used for Series 2 devices is the public portion of an ECDSA keypair over the NIST prime curve P-256. The Sign key is a customer key and is typically provisioned during the initial product manufacturing and device programming phase. It is common for all products that share a common firmware image to be loaded with the same public Sign key. The key loaded into the device is a public key and has no confidentiality requirements. The private key associated with that public key, which will be used to sign firmware images, should be tightly held, ideally secured in a hardware security module (HSM).

1.4 About the Secure Loader

In Series 2 devices, the Secure Loader is firmware pre-loaded into the devices. It is maintained by Silicon Labs, and is deployed through secure upgrade packages. It is the functional equivalent of the first-stage Gecko Bootloader on Series 1 devices (see *UG266: Gecko Bootloader User's Guide* for more information). The Secure Loader validates the authenticity and integrity of a staged image before performing an upgrade operation. The Secure Loader requires the staged image to reside on-chip and the staged image must not overlap with the target destination address range. Firmware images that originate from off-chip, either off-chip storage, external NCP host interface, or through an OTA update procedure are expected to be staged either by the application or by Gecko Bootloader before calling the Upgrade command of the Secure Loader.

2 Implementing the Secure Boot Process

This section describes how to implement the secure boot process and how to recover a device when Secure Boot fails.

2.1 Provisioning the Sign Key

In order to use Secure Boot, a Sign Key keypair must be generated. The public portion of the Sign Key keypair is used to verify the image during Secure Boot and must then be written to the SE OTP. The private portion of the Sign Key keypair is used to sign the application image for Secure Boot, and this private key must be protected, ideally stored in a Hardware Security Module (HSM) or equivalent key storage instrument.

2.1.1 Generating a Sign Key Using Simplicity Commander (Command Line)

```
commander gbl keygen --type ecc-p256 --outfile signing-key
Generating ECC P256 key pair...
Q_X: 79BF593CA56CBCEEBD7E7FB600B6EB7EE33572099220856EE62180BA6A90AB77
Q_Y: ABEBB15823554ECEF5A70ACB0FDC8DEC6C2E7BF091B333EFFC7AFD691462CDE4
D:   DE073A7B41031C1B07EF720C9583BB865E407733F17F7B43973A794A0A167DBA
Writing EC tokens to ecckey-tokens.txt...
Writing private key file in PEM format to ecckey...
Writing public key file in PEM format to ecckey.pub...
DONE
```

The above command creates an ECDSA-P256 key pair for signing: `signing-key` contains the private key in PEM format, and must be kept secret from third parties. This key will later be used to sign images. `Signing-key.pub` contains the public key in PEM format and is used to verify the image signature. Only `signing-key.pub` is loaded into the device.

2.1.2 Provisioning the Sign Key

To provision a Sign Key in Simplicity Studio:

1. Right-click the selected debug adapter [**J-Link Silicon Labs (serial number)**] to display the context menu.

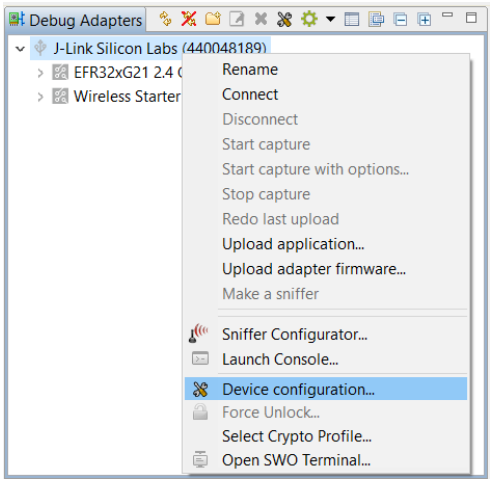


Figure 2.1. Device Configuration on the Debug Adapter Menu

2. Click **Device configuration...** to open the Device Configuration dialog box. Click the **Security Settings** tab to get the current selected device configuration (for example, an unlocked device).
3. The Debug Locks are set in Step 9. **Crypto Profile:** will show **None** if no Crypto Profile has been selected. Sign Key provisioning requires SE firmware version1.1.1 or greater.

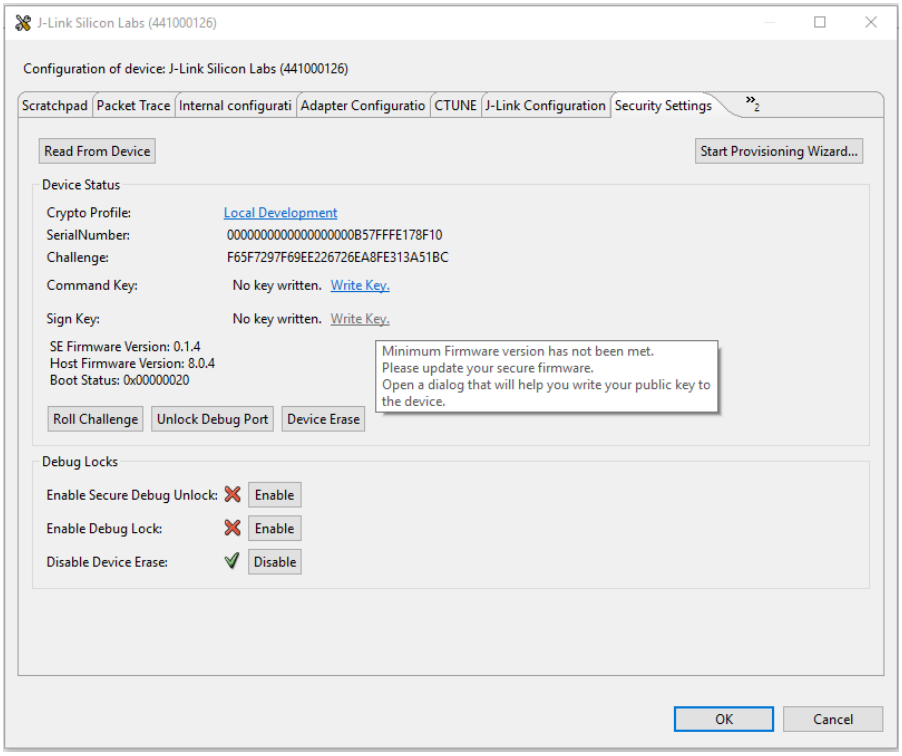


Figure 2.2. Security Settings Dialog Box Showing an Unsupported SE Firmware Version

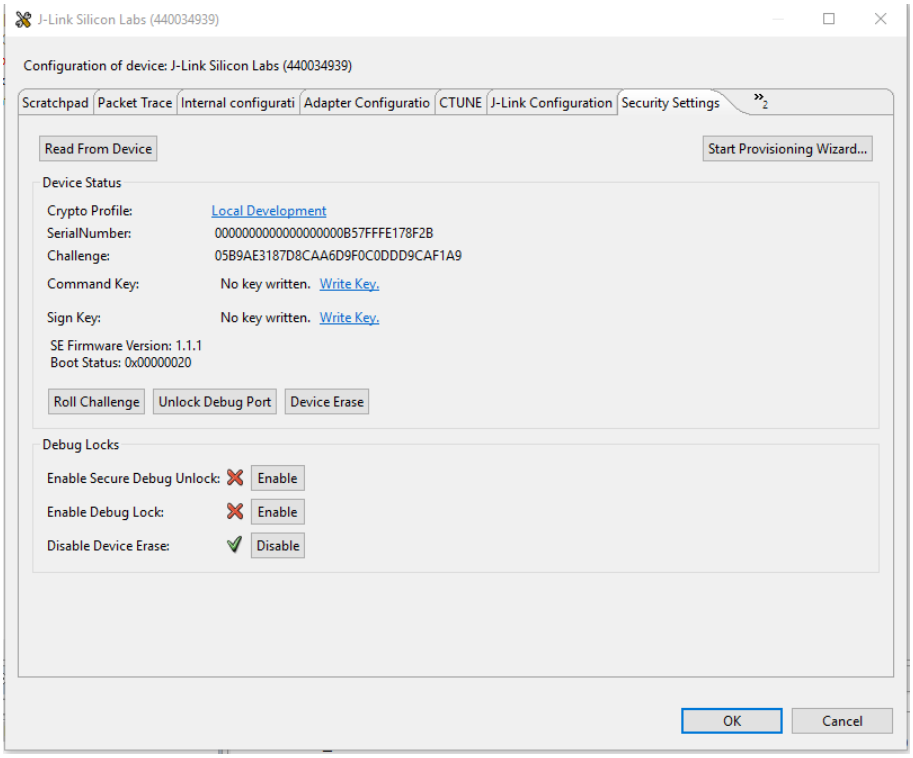


Figure 2.3. Security Settings Dialog Box for a Secure Boot-Enabled SE firmware version

4. Click [**Start Provisioning Wizard…**] in the upper right corner to display the **Secure Initialization** dialog box. **Enable Version Rollback Prevention of Host Image** is optional.

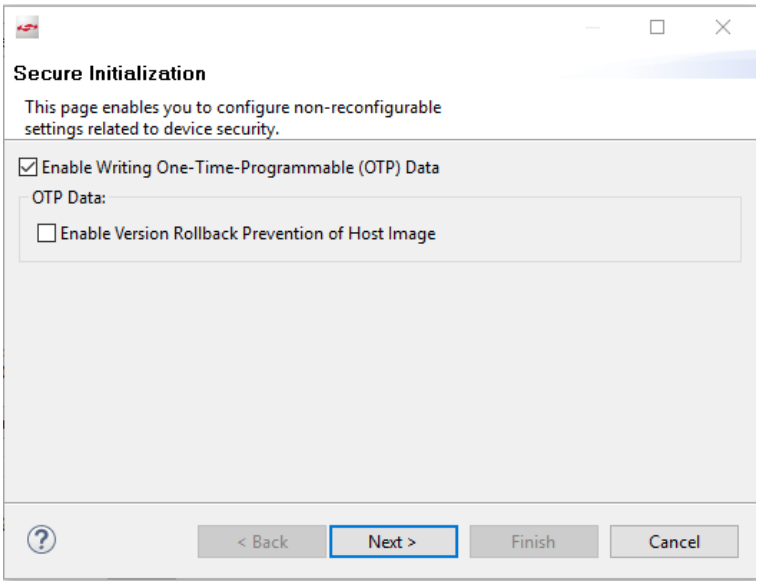


Figure 2.4. Secure Initialization Dialog Box

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

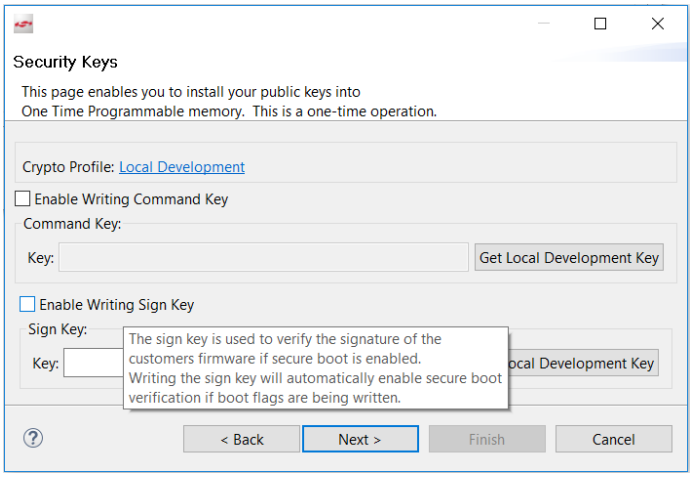


Figure 2.5. Security Keys Dialog Box

6. Checking **Enable Writing Sign Key** enables Secure Boot. The following Secure Boot Warning is displayed. Click **[Yes]** to confirm. Note that writing the Sign Key makes a **permanent irreversible change to the device**.

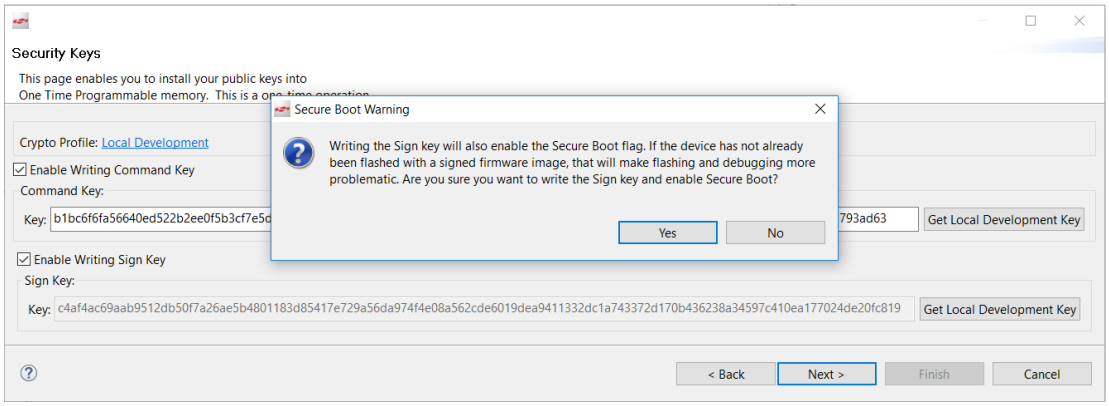


Figure 2.6. Enable Secure Boot

7. Insert a public key value for **Sign Key** from an external source (for example a Hardware Security Module or from Simplicity Commander as described above). You may also provision the **Command Key** for Secure Debug Unlock at this time.
8. Click **[Next >].** When Secure Boot is enabled, the Debug Locks are not set by default.

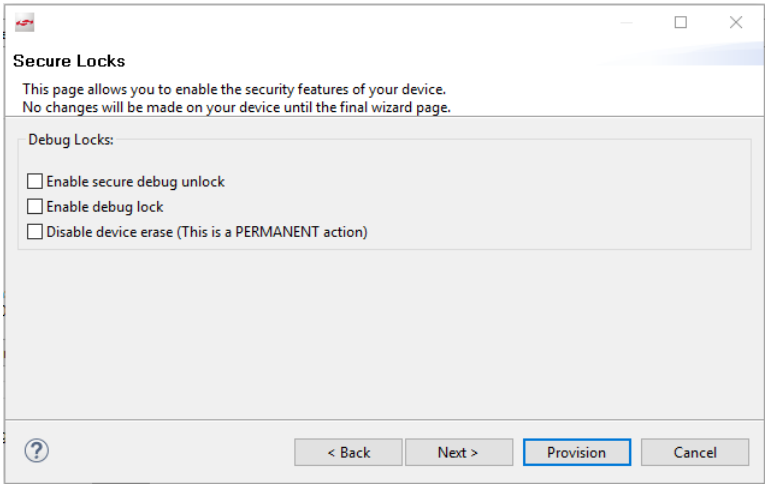


Figure 2.7. Secure Locks Dialog Box

9. Check the corresponding boxes to enable the desired Debug Locks. See *AN1190: EFR32xG21 Secure Debug* for more information about these locks.
10. If E**nable Secure Debug Unlock** is checked, the following Secure Debug Unlock Warning is displayed. Click **[No]** to abort.

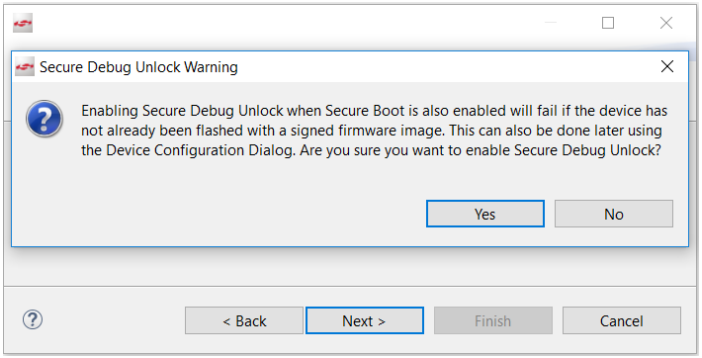


Figure 2.8. Secure Debug Unlock Warning

11. Click [**Next >**] to display the **Summary** dialog box.

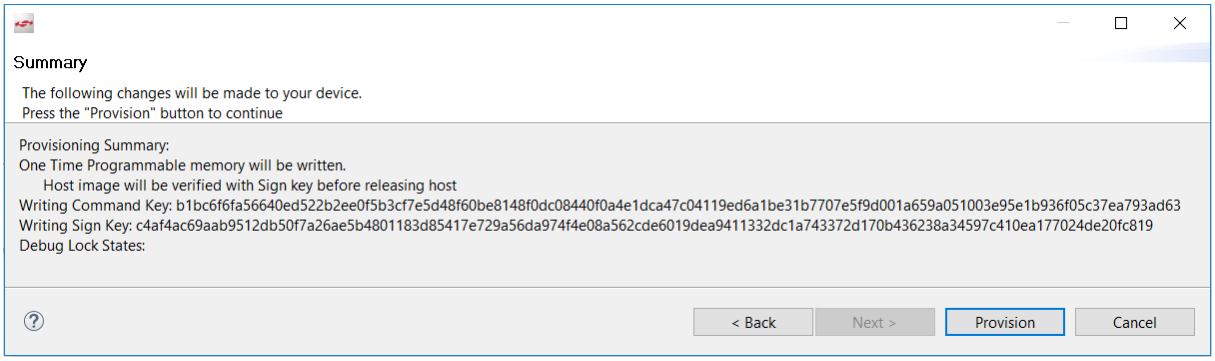


Figure 2.9. Provisioning Summary

12. If the information displayed is correct, click [**Provision**]. You are asked to confirm, as once written this information cannot be changed. Click **[Yes]** to confirm.

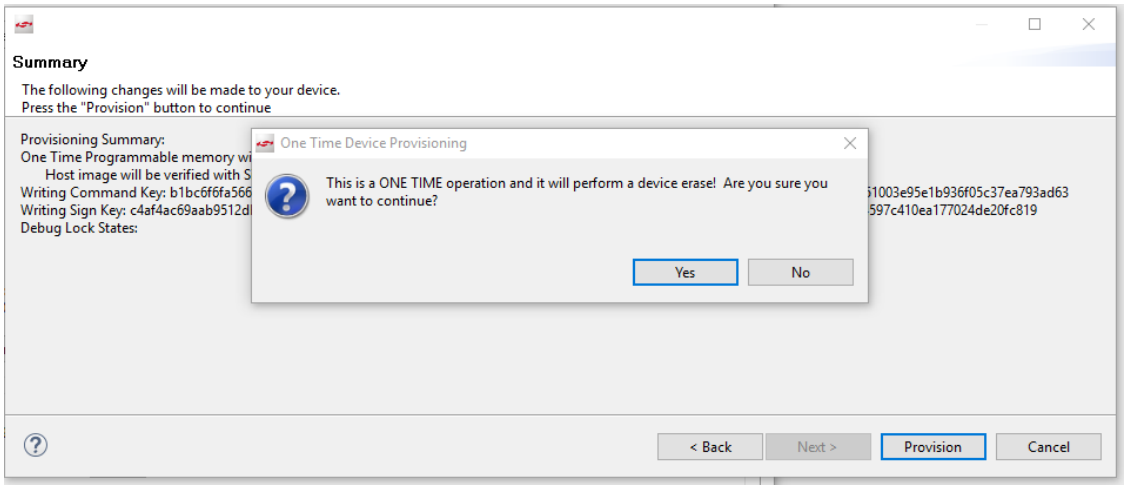


Figure 2.10. Provisioning Confirmation

13. The Summary dialog box changes to display the provisioning status.

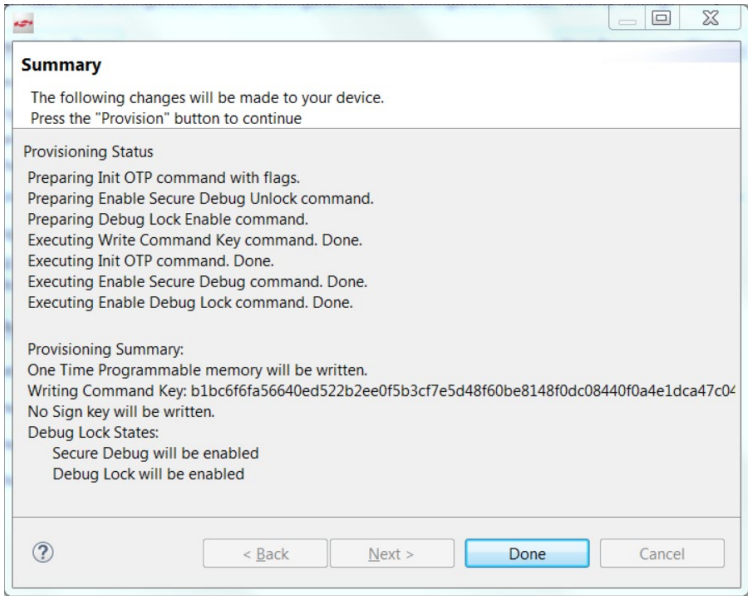


Figure 2.11. Provisioning Done

14. Click [**Done**] to finish and exit the provisioning process.

2.2 Recover Devices When Secure Boot Fails

If Secure Boot is enabled, the device is locked. If a Secure Boot process fails (meaning firmware image validation fails), the only way to recover is to flash a correctly-signed image. The device must be erased and unlocked before you can flash the new image. This section describes two methods by which to erase and unlock a device.

Note: The device cannot recover if Device Erase has been disabled.

2.2.1 Simplicity Commander (Command Line)

Use the device recover command to erase and unlock the device.

```
commander device recover
Recovering "bricked" device...
DONE
```

2.2.2 Simplicity Studio Device Configuration

1. Right-click the selected debug adapter [**J-Link Silicon Labs (serial number)**] to display the context menu.

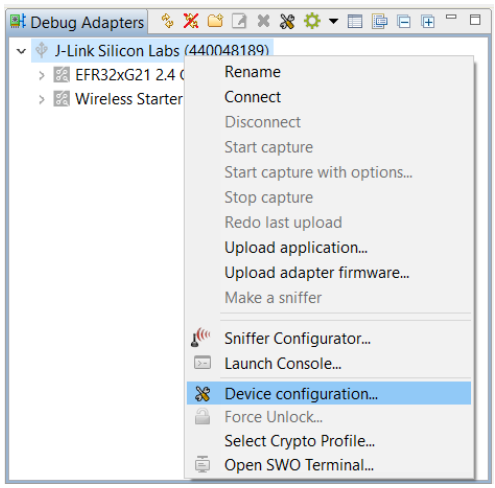


Figure 2.12. Device Configuration on the Debug Adapter Menu

2. Click **Device configuration...** to open the device configuration window, and click the **Security Settings** tab.

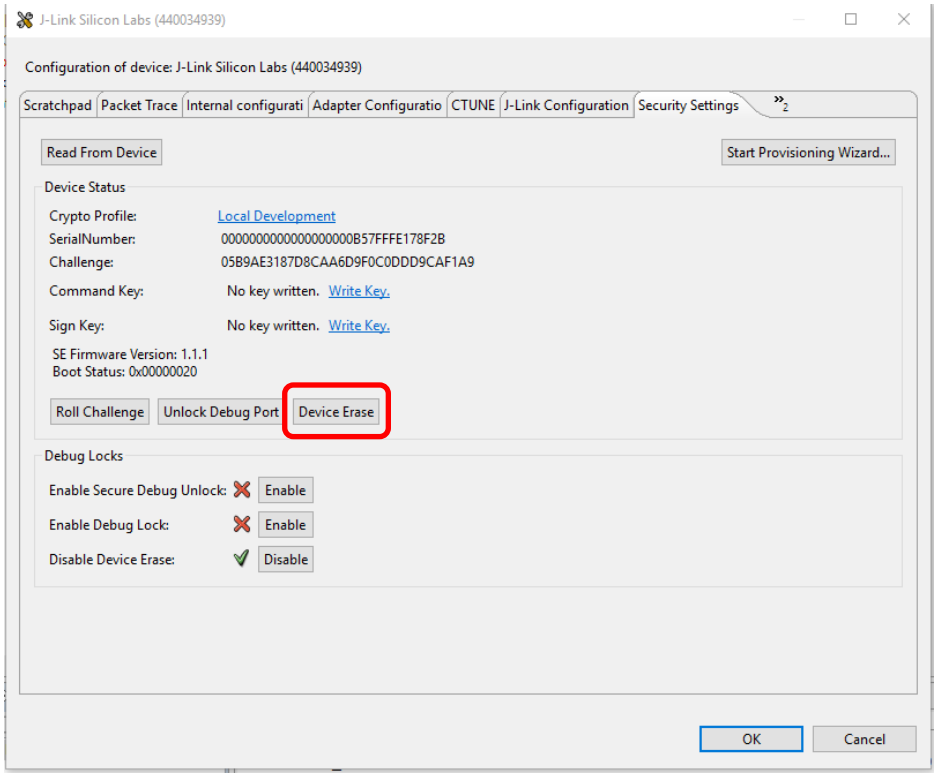


Figure 2.13. Device Erase Operation

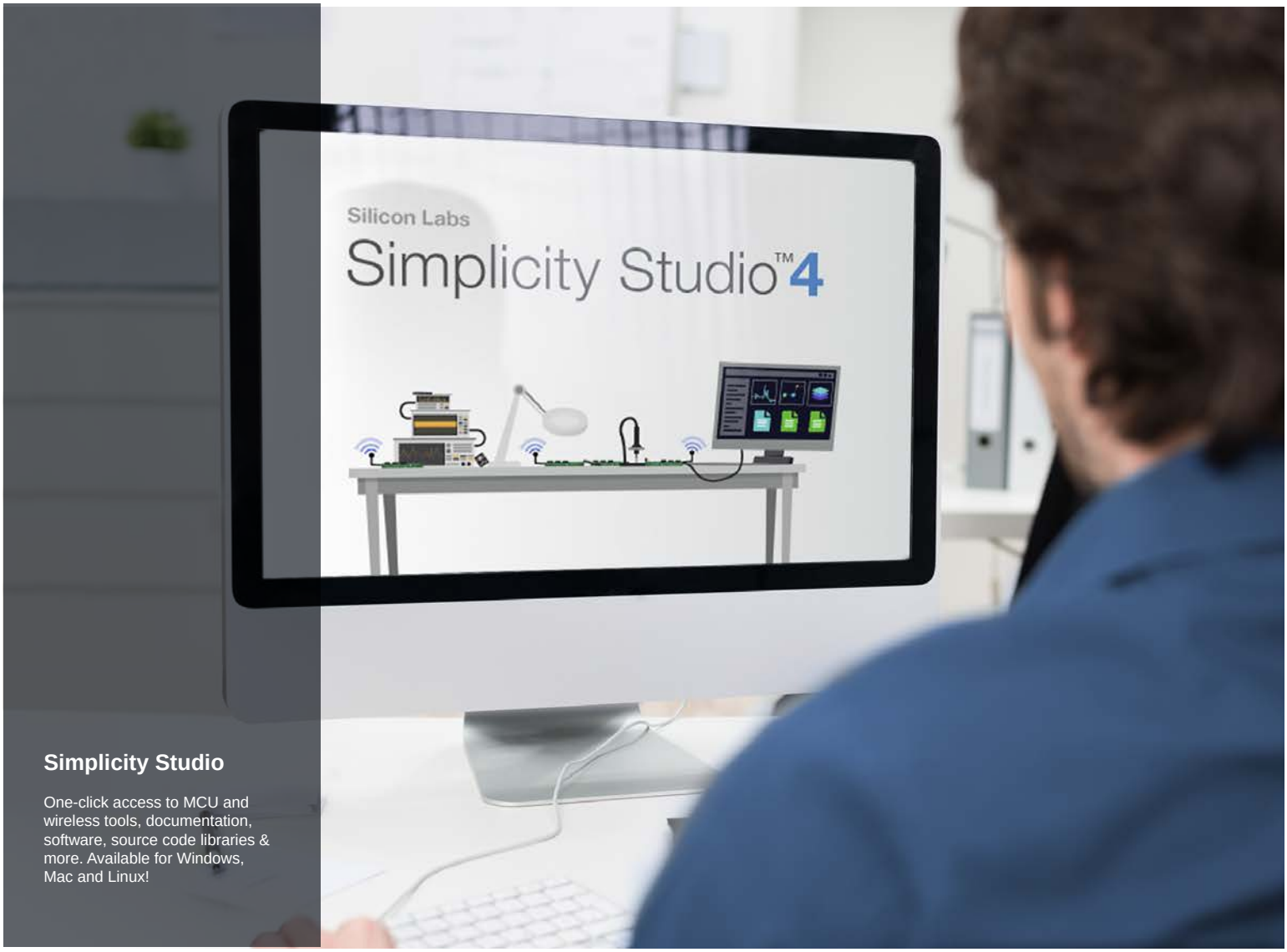
3. Click [**Device Erase**] to erase and unlock the device.

3 Revision History

Revision 0.1

August 2019

- Initial Release



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 and Community
community.silabs.com

SILICON LABS

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

http://www.silabs.com