MCUXSPTUG_25.09

Secure Provisioning Tool User Guide 25.09

Rev. 18 — 10 October 2025 User guide

Document information

Information	Content
Keywords	MCUXpresso Secure Provisioning Tool
Abstract	MCUXpresso Secure Provisioning Tool (SEC) is a GUI tool made to simplify the generation and provisioning of bootable executables on NXP processors. It is built upon the proven security enablement toolset provided by NXP and takes advantage of the breadth of programming interfaces provided by the BootROM library.

1 Introduction

A Chinese translation of the user guide in the PDF format can be downloaded from the Secure Provisioning Tool home page.

The **MCUXpresso Secure Provisioning Tool (SEC)** is a graphical user interface (GUI) application designed to streamline the creation and provisioning of bootable executables for NXP processors. Built on NXP's robust security enablement toolset, SEC leverages the extensive programming capabilities of the BootROM to simplify secure image preparation and deployment.

Whether you are new to secure provisioning or an experienced developer, SEC offers a user-friendly interface that enhances productivity. New users benefit from guided workflows for preparing, flashing, and securing images, while experienced users recognize familiar utilities—such as sdphost, blhost, and nxpimage—now integrated into a cohesive and intuitive environment.

For advanced use cases, SEC also supports customization through editable scripts, enabling experienced users to tailor provisioning flows to meet specific security and deployment requirements.

Figure 1. MCUXpresso Secure Provisioning Tool

2 Features

Here are the key features of the **Secure Provisioning Tool** (SEC):

2.1 Security Enablement

- Image signing and encryptions: supports generation of signed and encrypted images using customer-provided keys and certificates.
- Secure boot configuration: automates creation of secure boot headers (for example, HAB, TrustZone-M, BEE).
- Key management: integrates with NXP's key provisioning workflows, including SRK, DEK, and OTP key programming.
- Optional signature provider: allows customizing integration of the HSM module for signing the image.
- Trust provisioning: device HSM and EdgeLock 2GO

2.2 Device communication and flashing

- BootROM interface support: communicates with BootROM via UART, USB, SPI, or I2C.

- Flash programming: supports writing to internal flash and external memory (for example, QSPI, FlexSPI NOR/NAND, eMMC).
- Device detection and connection management: auto-detects connected devices and manages communication

2.3 User experience

- Guided workflows: workspace wizard with predefined profiles to create a new configuration easily.
- Visual feedback: real-time status updates, logs, and error reporting.
- Cross-Platform support: available for Windows, Linux, and macOS.

2.4 Useful extensions

- Debug authentication support
- SB editor allowing creation of Secure Binary files
- The Merge Tool allowing merging several images into one
- MCUboot signer allowing signing the custom application via the MCUboot third-party Tool
- Manufacturing Tool for FAB operations, allowing provisioning several devices in parallel
- Additional command-line utilities for low-level interaction with the device

2.5 Device and platform support

- Broad processor family coverage: Compatible with a wide range of NXP processors (for example, i.MX, LPC, MCX).
- Support for latest silicon features: Regularly updated to support new security features and silicon revisions.

3 Terms and definitions

Table Terms and definitions

Term	Definition
AAD	Additional Authenticated Data; typically used with AES-GCM
AES	Advanced Encryption Standard
AES-128	Rijndael cipher with block and key sizes of 128 bits
AHAB	Advanced High Assurance Boot
ATF	ARM Trusted Firmware
BCA	Bootloader Configuration Area
BEE	Bus Encryption Engine
Block cipher	Encryption algorithm that works on blocks of N={64, 128, …} bits
CA	Certificate Authority, the holder of a private key used to certify public keys
CAAM	Cryptographic Acceleration and Assurance Module, an accelerator for encryption, stream cipher, and hashing algorithms, with a random number generator and runtime integrity checker
CBC	Cipher Block Chaining, a cipher mode that uses the feedback between the ciphertext blocks

Term	Definition
CBC-MAC	A message authentication code computed with a block cipher
CFPA	Customer In-field Programmable Area
Cipher block	The minimum amount of data on which a block cipher operates
Ciphertext	Encrypted data
CMAC	Cipher-based Message Authentication Code
CMPA	Customer Manufacturing/Factory Programmable Area
CMS	Cryptographic Message Syntax, a general format for data that may have cryptography applied to it, such as digital signatures and digital envelopes. HAB uses the CMS as a container holding PKCS#1 signatures.
CSEc	Cryptographic Services Engine / EdgeLock Accelerator, feature set for various encryption, decryption, and authentication algorithms built to meet the SHE specification.
CSF	Command Sequence File, a binary data structure interpreted by the HAB to guide authentication operations
DA	Debug Authentication
DAP	Debug Authentication Protocol
DCD	Device Configuration Data, a binary table used by the ROM code to configure the device at an early boot stage
DCP	Data coprocessor, an accelerator for AES encryption and SHA hashing algorithms
DEK	Data encryption key, a one-time session key used to encrypt the bulk of the boot image
DUK	Device Unique Key
DUKB	DUK certificate block
ECB	Electronic Code Book, a cipher mode with no feedback between the ciphertext blocks
EKIB	Encrypted Key Info Block
ECU	Electronic Control Unit
ELE	EdgeLock Secure Enclave
EPRDB	Encrypted Protection Region Descriptor Block
FAC	Flash Access Controlled
FCB	Flash Configuration Block or Flash Control Block
FCF	Flash Configuration Field; can also be referenced as FCB
GCM	Galois/Counter Mode; it is a mode of operation for symmetric key cryptographic block ciphers, most commonly used with AES
GUI	Graphical User Interface
HAB	High Assurance Boot, a software library executed in internal ROM on the Freescale processor at boot time that, among other things, authenticates software in external memory by

Term	Definition
	verifying digital signatures in accordance with a CSF. This document is strictly limited to processors running HABv4
Hash	Digest computation algorithm
HSM	Hardware System Module
IEE	Inline Encryption Engine
IFR	Information Flash Region
IMG	Image Signing Key, interchangeable term with ISK
ISK	Image Signing Key, interchangeable term with IMG
ISP	In-system programming, a mode in which the processor can be programmed directly into the product
IVT	Image Vector Table
KEK	Key Encryption Key, used to encrypt a session key or DEK
KeyBlob	KeyBlob is a data structure that wraps the key and the counter and the range of image decryption using AESCTR (AES in Counter mode) algorithm
KIB	Key Info Block with KEY and IV for AES128-CBC, recall key and IV used in PRDB wrap and unwrap is defined as key info block
MAC	Message Authentication Code. Provides integrity and authentication checks
Message digest	A unique value computed from the data using a hash algorithm; provides only an integrity check (unless encrypted)
NBU	Narrowband Unit
NDA	Non-disclosure Agreement
OEI	Optional Executable Image
OEM	Original Equipment Manufacturer
OS	Operating System
OTFAD	On-The-Fly AES Decryption
OTP	One-Time Programmable. OTP hardware includes masked ROM, and electrically programmable fuses (eFuses).
OTPMK	One-Time Programmable Master Key
PFR	Protected Flash Region
PKCS#1	Standard specifying the use of the RSA algorithm. For more information, see PKCS#1 Wikipedia article and RSA PKCS#1 security archive Security archive.
PKI	Public Key Infrastructure, a hierarchy of public key certificates in which each certificate (except the root certificate) can be verified using the public key above it
Plaintext	Unencrypted data
PRDB	Protection Region Descriptor Block recalls the counter and the range of image decryption using the AES-CTR algorithm

Term	Definition
PRINCE	lightweight block cipher designed for low-latency hardware implementations, especially in environments like embedded systems and IoT devices
PUC	Product Unit Certificates
PUF	Physical Unclonable Function
pyOCD	Python-based tool and API for debugging, programming, and exploring Arm Cortex microcontrollers; for details, see pyOCD official website
Rijndael	Block cipher chosen by the US Government to replace DES; pronounced *rain-dahl*
ROMCFG	ROM Bootloader configurations
RoT	Root of Trust
RSA	A public key cryptography algorithm developed by Rivest, Shamir, and Adleman; Hardware accelerator (including hash acceleration) is found on some processors
RSA-PSS	RSA probabilistic signature scheme
SDP	Serial Download Protocol, also called UART/USB Serial Download mode. IT allows code provisioning through UART or USB during production and development phases
SEC Tool	Secure Provisioning Tool
Session key	The encryption key is generated at the time of encryption. Only ever used once
SHA-1	Hash algorithm that produces a 160-bit message digest
SHE	Secure Hardware Extension
SI	Secure installer
SNVS	Secure Non-Volatile Storage
SPL	Secondary Program Loader
SPSDK	Secure Provisioning SDK, an open source Python library, and command-line tools for secure provisioning of NXP processors
SRK	Super Root Key, an RSA key pair that forms the start of the boot-time authentication chain. The hash of the SRK public key is embedded in the processor using OTP hardware. The SRK private key is held by the CA. Unless explicitly noted, SRK in this document refers to the public key only
TEE	Trusted Execution Environment
UID	Unique Identifier, a unique value (such as a serial number) assigned to each processor during fabrication
UUU	Universal Update Utility used to download images to different MPU devices
V2X	Vehicle-to-everything is a standalone cryptographic accelerator (EdgeLock Accelerator) on i.MX 95
WPC	Wireless Power Consortium

Term	Definition
XIP	Execute-In-Place refers to a software image that is executed directly from its non-volatile storage location rather than first being copied to volatile memory
XMCD	External Memory Configuration Data

4 Installation

This chapter describes the procedure required to install SEC on Windows, macOS, and Linux operating systems. For the supported operating systems, refer to Minimum system requirements.

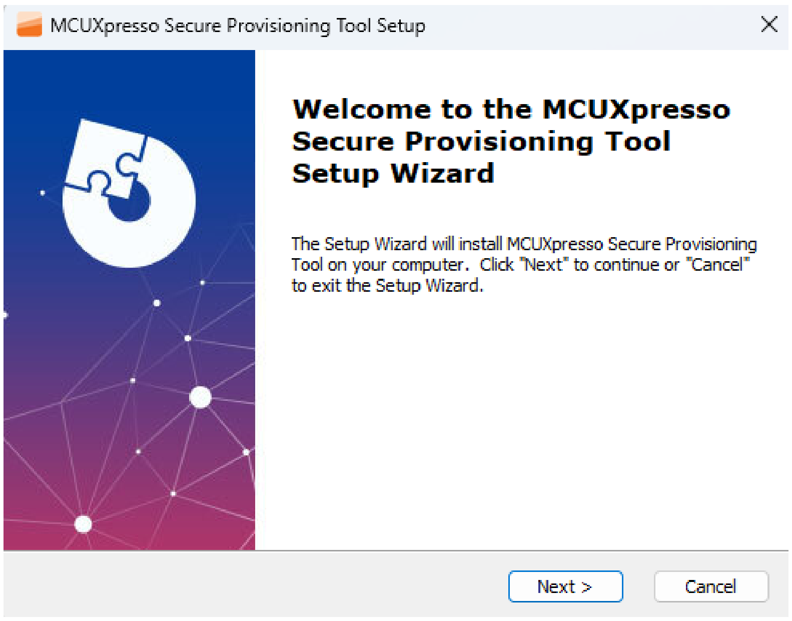
4.1 Minimum system requirements

The tool runs on Microsoft(R) Windows(R), Ubuntu, and macOS operating systems. The detailed system requirements are specified in Secure Provisioning Tool Release Notes. The document also describes what debug probe drivers should be installed. These drivers might also be needed for communication with the processor in ISP mode where the probe provides USB to UART/SPI/I2C converter.

4.2 Windows

To install SEC as a desktop application on a local host, perform the following steps:

1. Visit the Secure Provisioning Tool home page to download the SEC installer for Windows.
2. Double-click the `MCUXpresso_Secure_Provisioning_<version>.exe` installer to begin installation.
3. On the first page of the wizard, click **Next**.

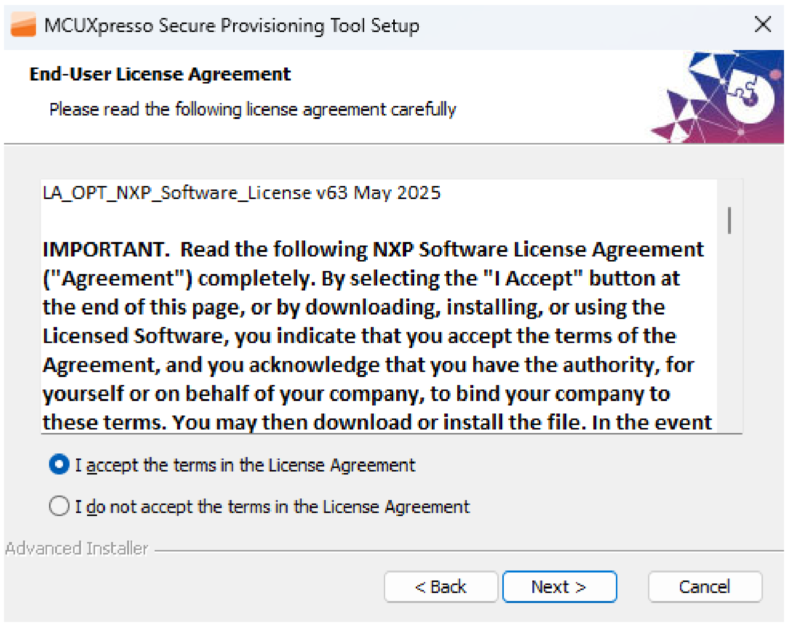


Welcome to the MCUXpresso Secure Provisioning Tool Setup Wizard

The Setup Wizard will install MCUXpresso Secure Provisioning Tool on your computer. Click "Next" to continue or "Cancel" to exit the Setup Wizard.

Figure 2. Secure Provisioning Tool Setup

4. On the **End-User License Agreement** page of the wizard, select **I accept the terms of the License Agreement** and click **Next**.



End-User License Agreement

Please read the following license agreement carefully

LA_OPT_NXP_Software_License v63 May 2025

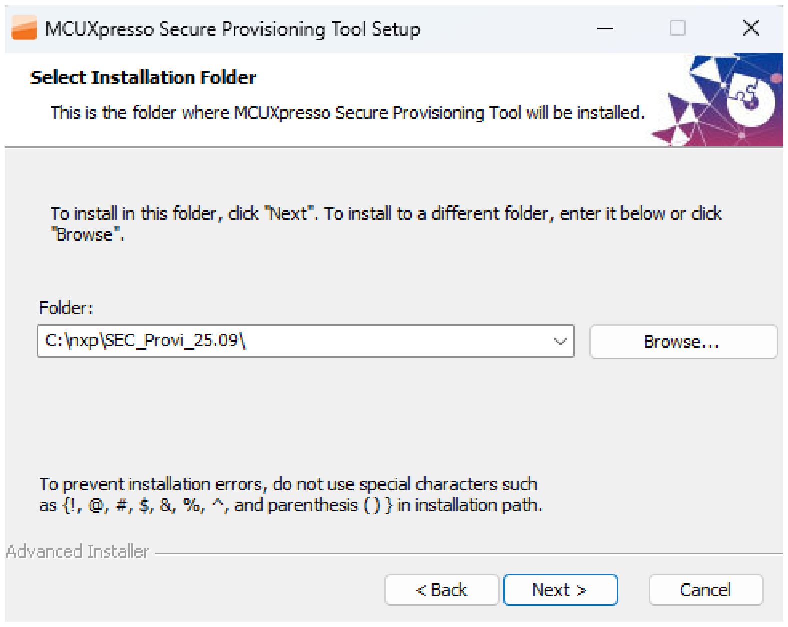
IMPORTANT. Read the following NXP Software License Agreement ("Agreement") completely. By selecting the "I Accept" button at the end of this page, or by downloading, installing, or using the Licensed Software, you indicate that you accept the terms of the Agreement, and you acknowledge that you have the authority, for yourself or on behalf of your company, to bind your company to these terms. You may then download or install the file. In the event

[✓ I accept the terms in the License Agreement]

I do not accept the terms in the License Agreement

Figure 3. Accepting the end-user license agreement

5. On the **Select Installation Folder** page of the wizard, select **Browse** and navigate to a destination folder you want to install the SEC to and click **Next**.



Select Installation Folder

This is the folder where MCUXpresso Secure Provisioning Tool will be installed.

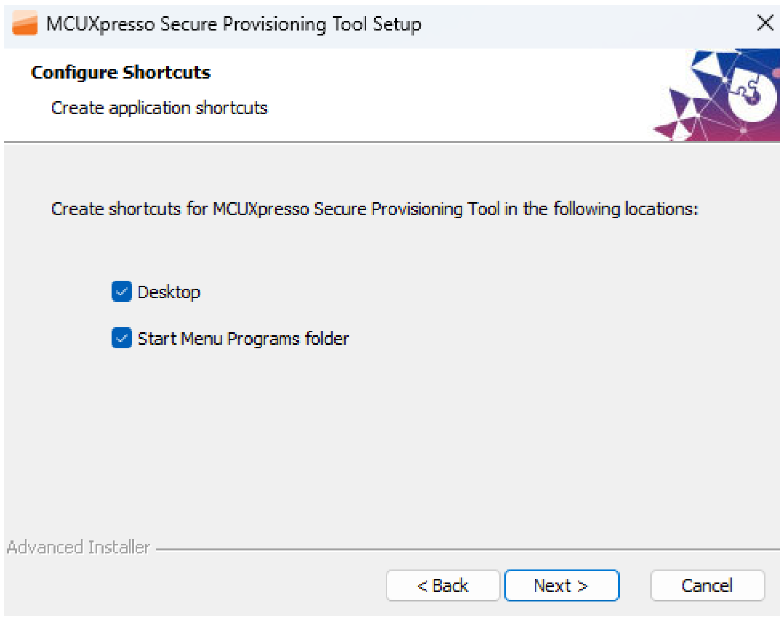
To install in this folder, click "Next". To install to a different folder, enter it below or click "Browse".

Folder: C:\nxp\SEC_Provi_25.09\

To prevent installation errors, do not use special characters such as {!, @, #, $, &, %, ^, and parenthesis ( ) } in installation path.

Figure 4. Selecting installation folder

6. On the **Configure Shortcuts** page of the wizard, select the shortcuts you want to be created for SEC and click **Next**.



Configure Shortcuts

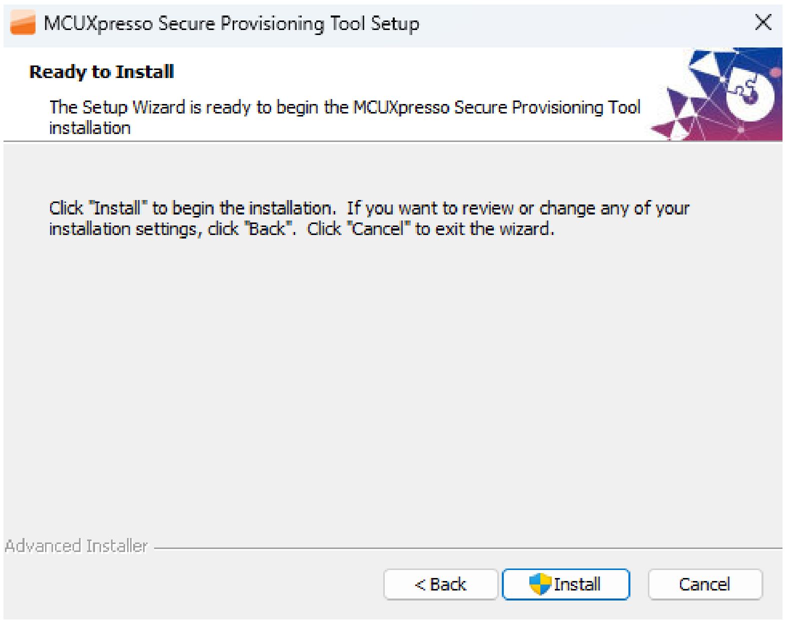
Create application shortcuts

Create shortcuts for MCUXpresso Secure Provisioning Tool in the following locations:

- [✓] Desktop
- [✓] Start Menu Programs folder

Figure 5. Configuring shortcuts

7. On the **Ready to Install** page of the wizard, select **Install**.
   The setup begins the installation.
   Note: If you want to review or change any of your installation settings, click **Back**. Click **Cancel** to exit the wizard.
   The installer prompts you when the installation completes.

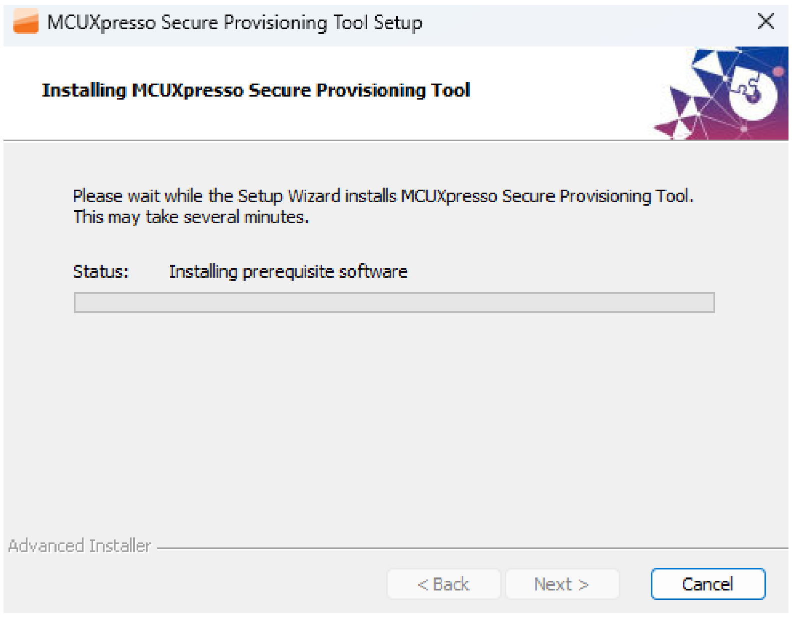


Ready to Install

The Setup Wizard is ready to begin the MCUXpresso Secure Provisioning Tool installation

Click "Install" to begin the installation. If you want to review or change any of your installation settings, click "Back". Click "Cancel" to exit the wizard.

Figure 6. Ready to install


MCUXpresso Secure Provisioning Tool Setup

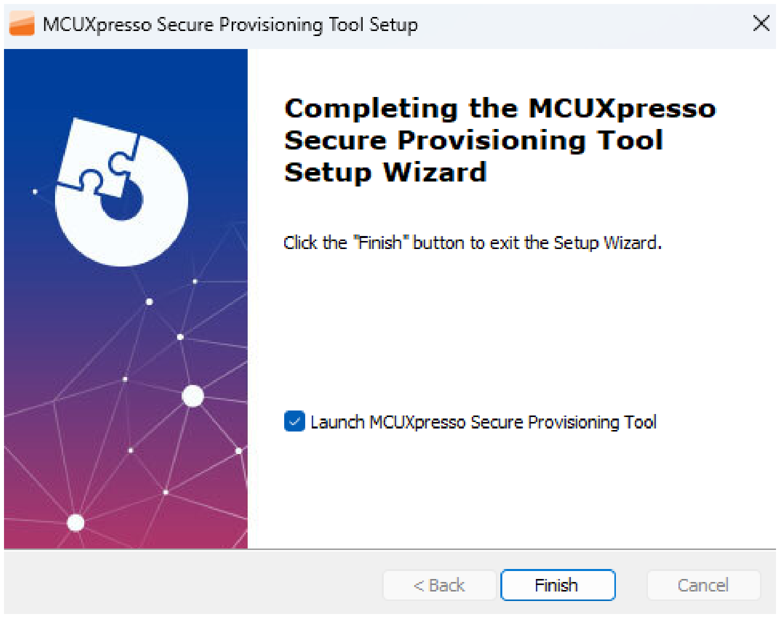
Installing MCUXpresso Secure Provisioning Tool

Please wait while the Setup Wizard installs MCUXpresso Secure Provisioning Tool.
This may take several minutes.

Status: Installing prerequisite software

Figure 7. Installing Secure Provisioning Tool

8. Click **Finish** to close and exit the setup wizard.


MCUXpresso Secure Provisioning Tool Setup

Completing the MCUXpresso Secure Provisioning Tool Setup Wizard

Click the "Finish" button to exit the Setup Wizard.

☑ Launch MCUXpresso Secure Provisioning Tool

Figure 8. Completing installation

9. To start using SEC, run the tool from the desktop shortcut on the desktop or from the **Start** menu. You can also navigate to the *<product installation folder>\bin\* folder and launch the `securep.exe` or launch the shortcut in the *<product installation folder>*.

4.2.1 Windows CLI

It is possible to install the SEC Tool using the command line. In this case, use **Run the installer** with the following parameters: start /wait

```
MCUXpresso_Secure_Provisioning_YY.MM.exe /exenoui /qn
```

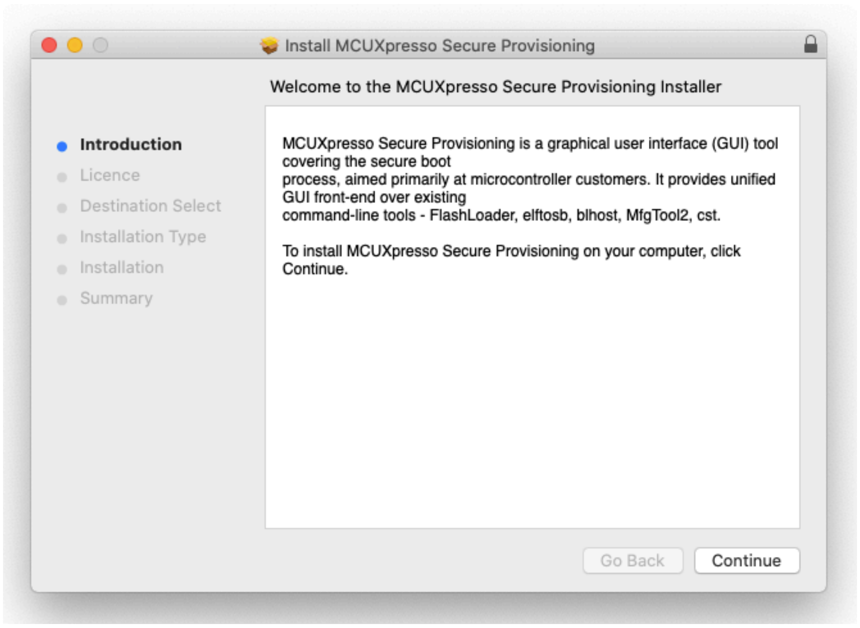
4.3 MacOS

To install SEC as a desktop application on a local host, perform the following steps:

1. Visit the NXP website to download the SEC installer for macOS. Based on your computer, select either an installer for an Intel or Apple M processor.
2. Double-click the `MCUXpresso_Secure_Provisioning_YY.MM.pkg` to start the **Install MCUXpresso Secure Provisioning Tool** wizard.

Note: When you try to open the macOS installer, you may receive an error. To avoid it, manually select the option **Mac App Store and identified developers** in the **Security & Privacy** menu.

1. On the **Introduction** page, click **Continue**.



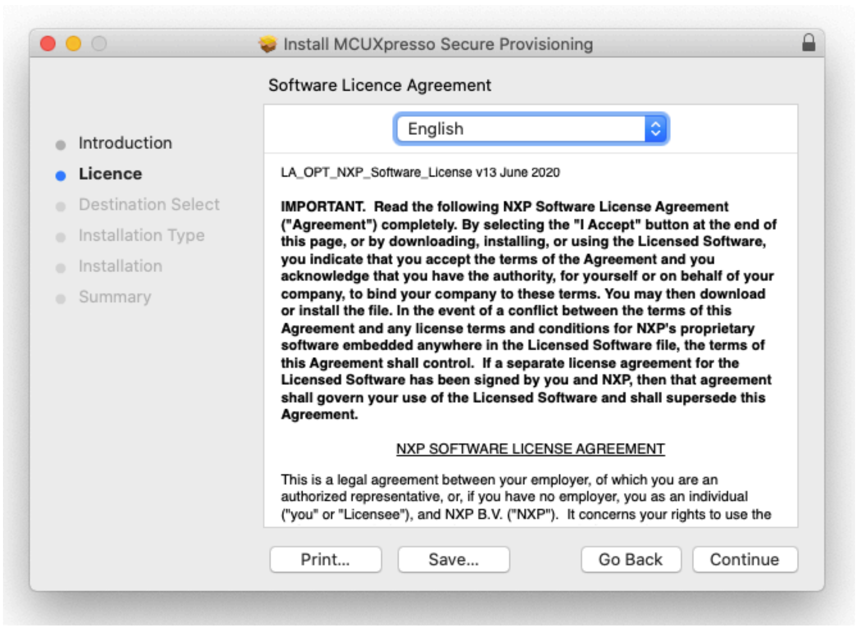
Welcome to the MCUXpresso Secure Provisioning Installer

MCUXpresso Secure Provisioning is a graphical user interface (GUI) tool covering the secure boot
process, aimed primarily at microcontroller customers. It provides unified GUI front-end over existing
command-line tools - FlashLoader, elftosb, blhost, MfgTool2, cst.

To install MCUXpresso Secure Provisioning on your computer, click Continue.

Figure 9. Introduction

2. On the **Software License Agreement** page, click **Continue**.



LA_OPT_NXP_Software_License v13 June 2020

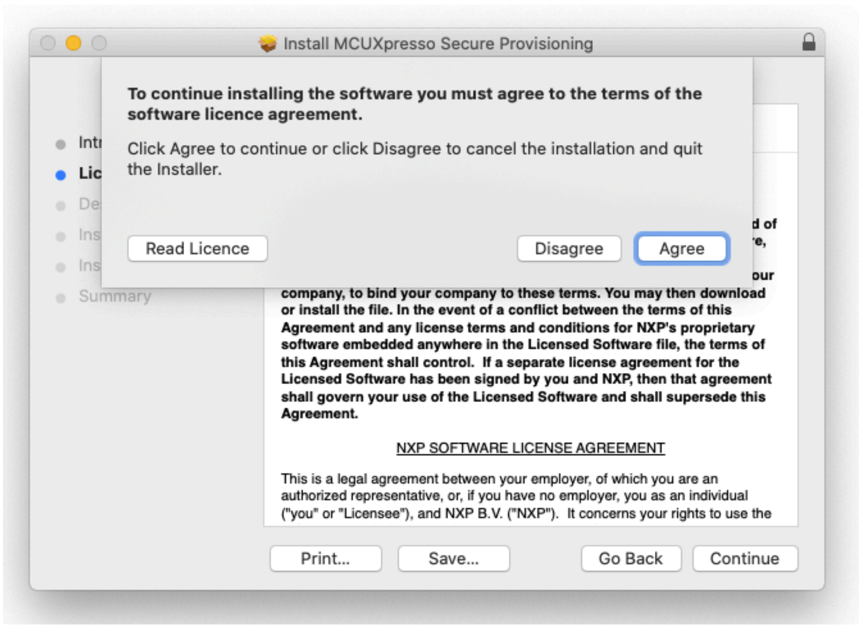
IMPORTANT. Read the following NXP Software License Agreement ("Agreement") completely. By selecting the "I Accept" button at the end of this page, or by downloading, installing, or using the Licensed Software, you indicate that you accept the terms of the Agreement and you acknowledge that you have the authority, for yourself or on behalf of your company, to bind your company to these terms. You may then download or install the file. In the event of a conflict between the terms of this Agreement and any license terms and conditions for NXP's proprietary software embedded anywhere in the Licensed Software file, the terms of this Agreement shall control. If a separate license agreement for the Licensed Software has been signed by you and NXP, then that agreement shall govern your use of the Licensed Software and shall supersede this Agreement.

NXP SOFTWARE LICENSE AGREEMENT

This is a legal agreement between your employer, of which you are an authorized representative, or, if you have no employer, you as an individual ("you" or "Licensee"), and NXP B.V. ("NXP"). It concerns your rights to use the

Figure 10. Software license agreement

3. Confirm that you have read and agreed to the terms of the Software License Agreement by clicking **Agree**.

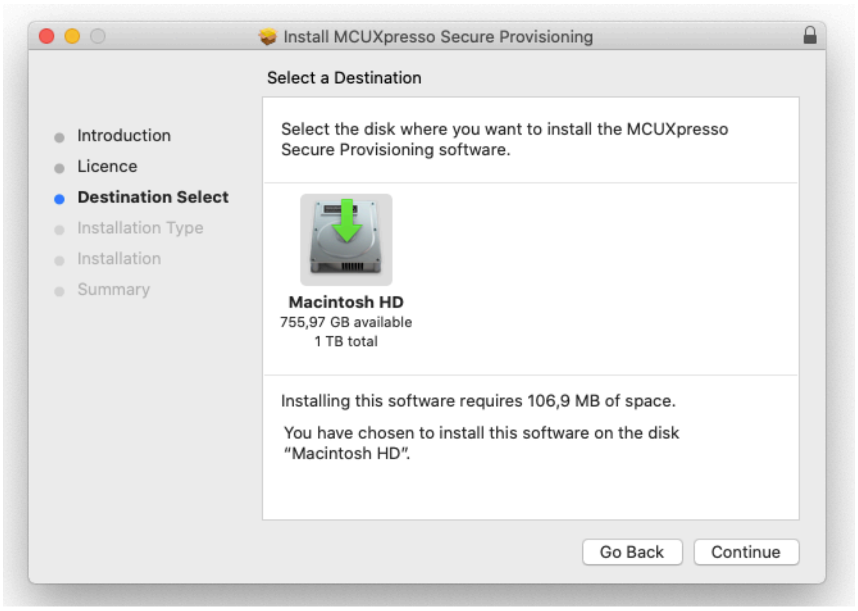


To continue installing the software you must agree to the terms of the software licence agreement.

Click Agree to continue or click Disagree to cancel the installation and quit the Installer.

Figure 11. Accepting software license agreement

4. On the **Destination Select** page, click the green arrow to select the installation folder, and once done, click **Continue**.


Install MCUXpresso Secure Provisioning

Select a Destination

Select the disk where you want to install the MCUXpresso Secure Provisioning software.

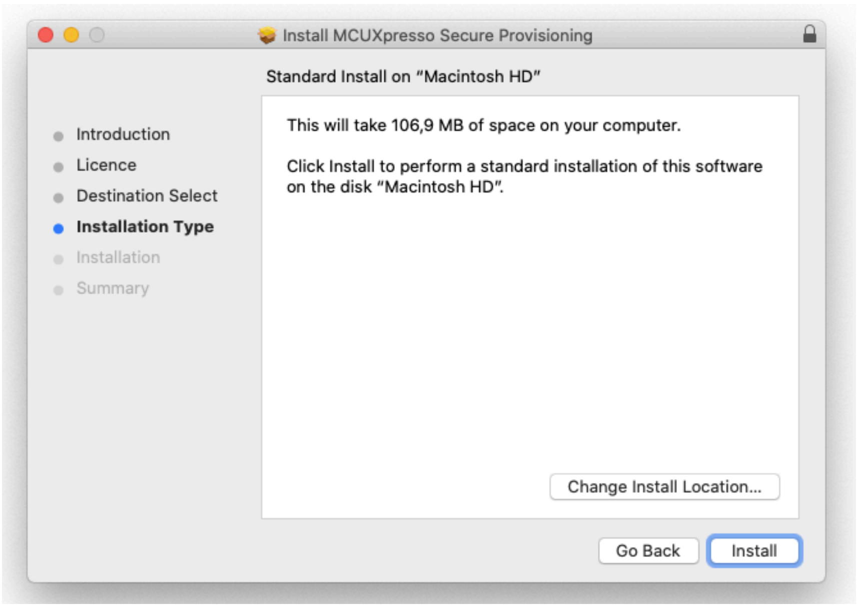
Macintosh HD
755,97 GB available
1 TB total

Installing this software requires 106,9 MB of space.

You have chosen to install this software on the disk "Macintosh HD".

Figure 12. Select destination

5. On the **Installation Type** page, click **Install**.


Install MCUXpresso Secure Provisioning

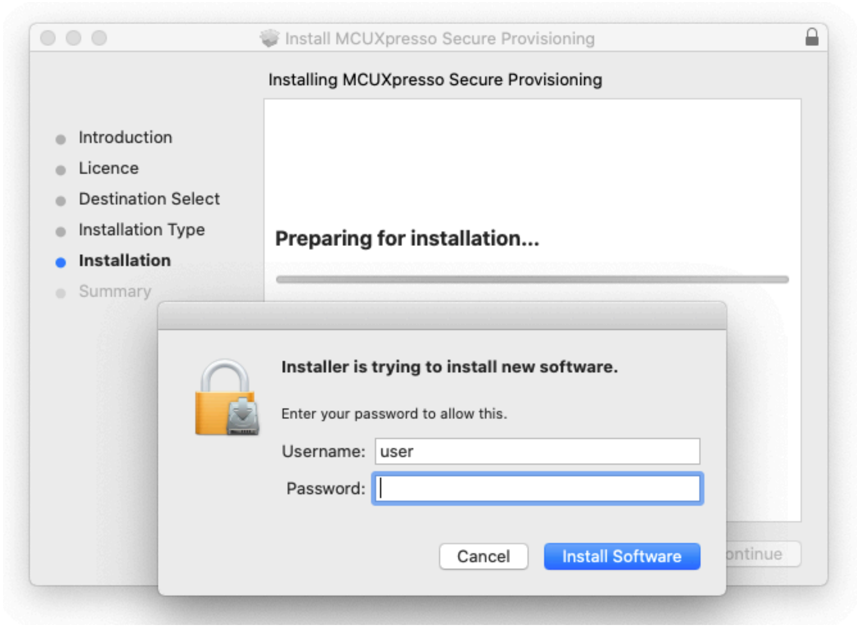
Standard Install on "Macintosh HD"

This will take 106,9 MB of space on your computer.

Click Install to perform a standard installation of this software on the disk "Macintosh HD".

Figure 13. Installation type

6. Type in your login credentials to continue with the installation and click **Install Software**.



Installing MCUXpresso Secure Provisioning

Preparing for installation...

Installer is trying to install new software.

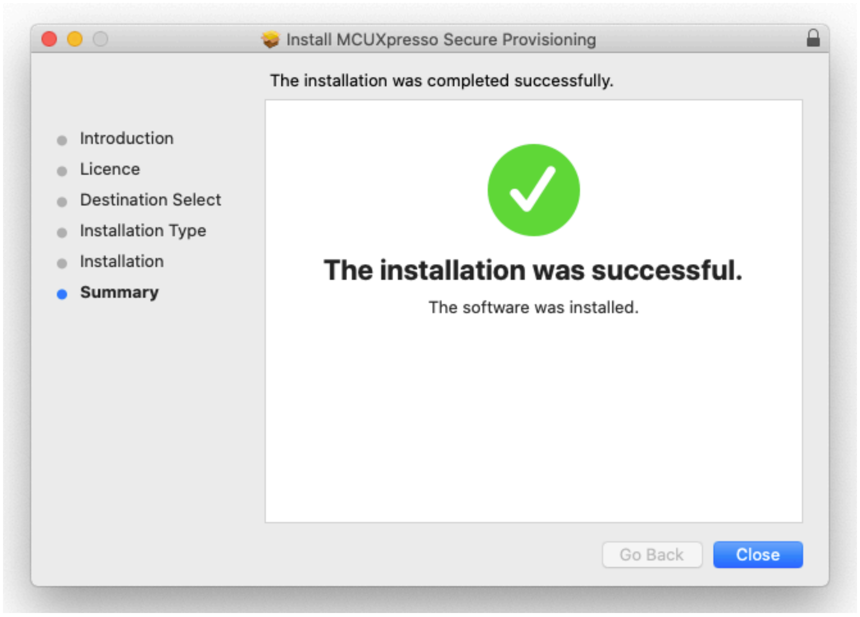
Enter your password to allow this.

Username: user

Password:

Figure 14. Install software

7. Click **Continue**.
   Unless errors are reported, the **Summary** page confirms that the installation was completed successfully.
8. On the **Summary** page, click **Close**.



The installation was completed successfully.

The installation was successful.

The software was installed.

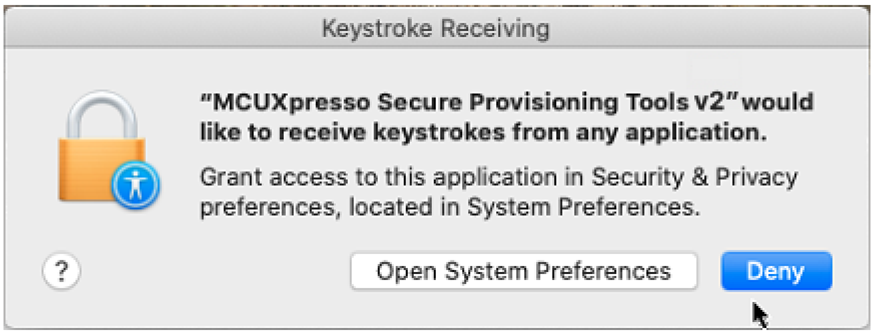
Figure 15. Summary

4.3.1 Enabling USB connection on macOS

During the first connection to the target by USB, macOS X Catalina blocks the access to USB HID devices as a security measure and the operation will fail with error. In macOS 13 (Ventura), this works differently and these steps are not needed.

Perform the following steps to enable USB connection:

1. In the OS security alert message box, select **Open System Preferences**.



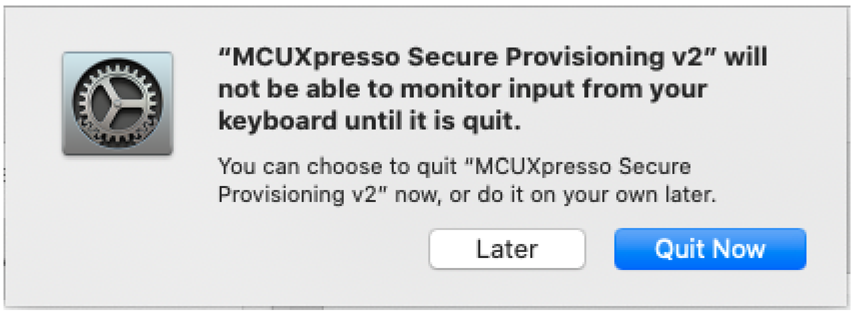
Keystroke Receiving

"MCUXpresso Secure Provisioning Tools v2" would like to receive keystrokes from any application.

Grant access to this application in Security & Privacy preferences, located in System Preferences.

Figure 16. Open System Preferences

2. Unlock **Privacy** preferences to enable changes.
3. Select **MCUXpresso Secure Provisioning Tool YY.MM**, confirm, and quit the application.



"MCUXpresso Secure Provisioning v2" will not be able to monitor input from your keyboard until it is quit.

You can choose to quit "MCUXpresso Secure Provisioning v2" now, or do it on your own later.

Figure 17. Confirm change

4. Lock **Privacy** preferences.
5. If the application was not closed, close it manually.
6. Start the application and proceed with the operation.

4.4 Linux

Installation of SEC on Ubuntu can be done in the Terminal.

1. Visit the NXP website to download the SEC installer for Linux.
2. Open the terminal and change the directory where the installer is downloaded, install using dpkg with sudo.

```
$ cd ~/Downloads
$ sudo dpkg -i mcuxpresso-secure-provisioning-<version>_<architecture>_<os-
version>.deb
```

   If the command executed with sudo is successful, the setup installs the SEC Tool in the dedicated folder /opt/nxp/.

Note: SEC depends on other packages that have to be installed in advance:

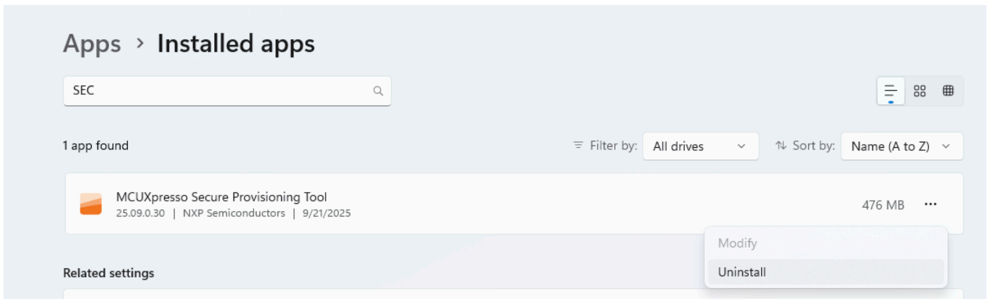
```
$ sudo apt install libhidapi-dev libsdl2-2.0-0
```

4.5 Uninstalling

4.5.1 Windows

The Secure Provisioning Tool can be uninstalled in the following ways:

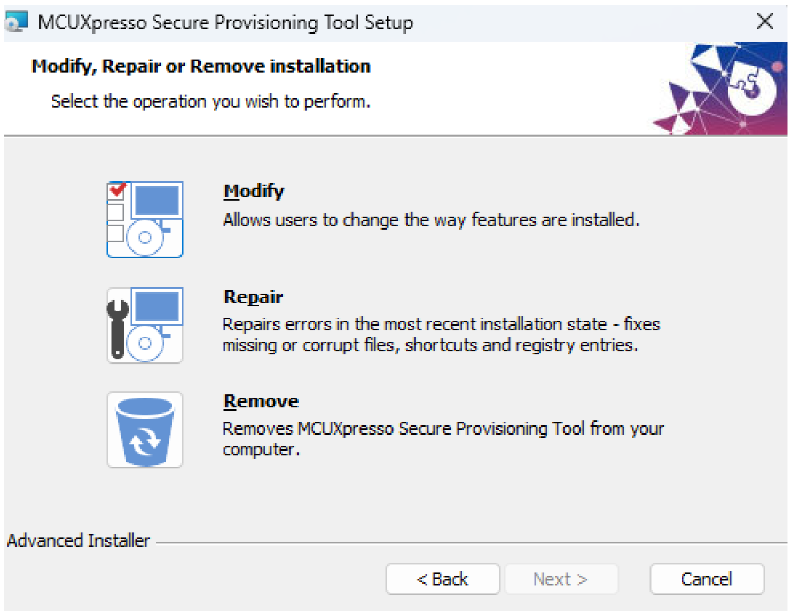
- by using Settings | Apps & features



MCUXpresso Secure Provisioning Tool
25.09.0.30 | NXP Semiconductors | 9/21/2025
476 MB

Figure 18. Settings

- by navigating into the `%APPDATA%\NXP Semiconductors` and then finding the appropriate MSI installer in the `Secure Provisioning Tool YY.MM\install\` folder and choosing the **Remove** option in the wizard.



Modify, Repair or Remove installation

Select the operation you wish to perform.

Modify
Allows users to change the way features are installed.

Repair
Repairs errors in the most recent installation state - fixes missing or corrupt files, shortcuts and registry entries.

Remove
Removes MCUXpresso Secure Provisioning Tool from your computer.

Figure 19. Secure Provisioning Tool setup wizard

4.5.2 MacOS

Secure Provisioning Tool can be uninstalled by using Finder, navigating to Applications, and moving `SEC_Provi_##.##` into the Trash.

4.5.3 Linux

The Secure Provisioning Tool can be uninstalled by using the Debian package manager.

In the Terminal you can get the list of secure provisioning tools with the package names:

```
$ dpkg --list "*-secure-provisioning*"
Desired=Unknown/Install/Remove/Purge/Hold
| Status=Not/Inst/Conf-files/Unpacked/halF-conf/Half-inst/trig-aWait/Trig-pend
|/ Err?=(none)/Reinst-required (Status,Err: uppercase=bad)
||/ Name                                    Version       Architecture
 Description
```

```
+++-======================================-============-============-
====================================
ii  mcuxpresso-secure-provisioning-25.09    25.09        amd64        MCUXpresso
 Secure Provisioning Tool
ii  mcuxpresso-secure-provisioning-v3.1     3.1          amd64        MCUXpresso
 Secure Provisioning
ii  mcuxpresso-secure-provisioning-v4       4.0          amd64        MCUXpresso
 Secure Provisioning
```

Now, the desired version can be uninstalled:

```
$ sudo dpkg --remove mcuxpresso-secure-provisioning-25.09
```

4.5.4 Remove configuration files

The user preferences are stored in the folder `<user home>\.nxp\secure_provisioning_<version>\` and are not removed by uninstalling the product. These folders can be removed manually. For more details about preferences, see Preferences.

4.5.5 Remove restricted data

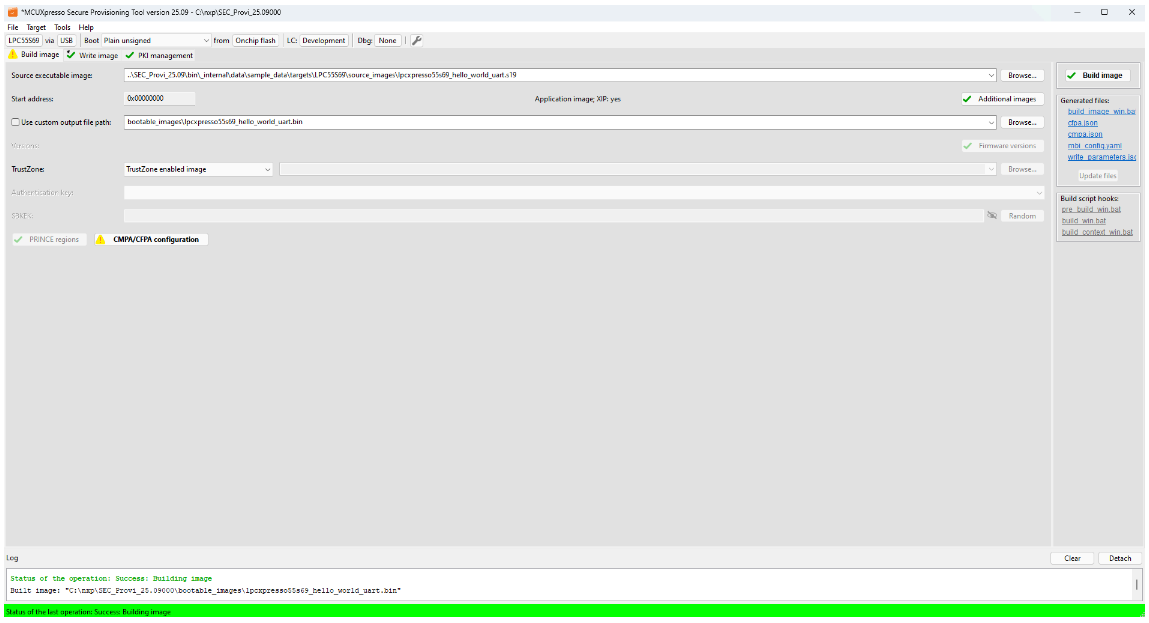
The restricted data are installed in the folder `<user home>\.nxp\secure_provisioning_restricted_data\` and are not removed by uninstalling the product. These folders can be removed manually.

5 User interface

SEC offers a simple and user-friendly interface. It consists of the **menu bar**, **toolbar**, and main views accessible through tabs:

- **Build image** allows building a bootable image.
- **Write image** allows writing a bootable image into the processor and optionally securing it.
- **PKI management** allows generating authentication keys or configuring the Signature Provider.
- The bottom part of the interface is occupied by the **Log** window and status line.

Configuration controls that are not supported for the selected processor might not be displayed. If a control is supported for the processor but is not meaningful in the current setting, it is disabled (gray). The tool highlights controls with configuration problems in red (errors) or yellow/orange (warnings). The problem is described in the tooltip. The tool displays information messages in blue.


*MCUXpresso Secure Provisioning Tool version 25.09 - C:\nxp\SEC_Provi_25.09000

LPC55S69 via USB | Boot Plain unsigned from Onchip flash | LC: Development | Dbg: None

Source executable image: ..\SEC_Provi_25.09\bin_internal\data\sample_data\targets\LPC55S69\source_images\lpcxpresso55s69_hello_world_uart.s19

Start address: 0x00000000

Application image; XIP: yes

Use custom output file path: bootable_images\lpcxpresso55s69_hello_world_uart.bin

TrustZone: TrustZone enabled image

Generated files: build_image_win.bat, cfpa.json, cmpa.json, mbi_config.yaml, write_parameters.json

Build script hooks: pre_build_win.bat, build_win.bat, build_context_win.bat

```
Status of the operation: Success: Building image
Built image: "C:\nxp\SEC_Provi_25.09000\bootable_images\lpcxpresso55s69_hello_world_uart.bin"
```

Status of the last operation: Success: Building image

Figure 20. SEC user interface

5.1 Menu and settings

This chapter gives detailed information about menu options and settings of the tool.

5.1.1 Title of the main window

The main window title shows:

- Asterisk (*), if the configuration is not saved on the disk (it is “dirty”)
- Name of the tool
- Path to the current workspace

5.1.2 Menu bar

The **menu bar** (main menu) contains several drop-down menus offering various application, configuration, and file-related functions.

File : General workspace and configuration-related operations

- **New Workspace …** : Creates a workspace. You are prompted to specify its location and choose from the supported processors. In the case the location already contains a workspace, the workspace is opened and not created. For more information, see Workspaces.
- **Import Manufacturing Package …** : Imports a manufacturing package (*.zip) with all data necessary for manufacturing and creates a “manufacturing workspace” (see Workspaces. for details about manufacturing workspace).
- **Select Workspace …** : Switches to another workspace. You are prompted to specify which workspace to open. For more information, see Workspaces.
- **Recent Workspaces** : Displays a list of recently used workspaces. For more information, see Workspaces. The number of displayed workspaces can be customized in Preferences.
- **Save Settings** : Saves the current workspace settings.
- **Export Workspace …** : Exports the current workspace as package (*.zip) with options to filter out unnecessary files such as generated scripts. For more information, see Sharing and copying workspaces.

- **Import Workspace …** : Imports workspace package (*.zip) into a specified folder.
- **Explore Workspace** : Opens the file explorer in the current workspace.
- **Open Terminal** : Opens the terminal in the current workspace directory with a path configured for SPSDK applications.
- **Preferences** : Opens the **Preferences** dialog. For more information, see Preferences.
- **Exit** : Exits SEC.

Target : This menu duplicates the operations available from the toolbar, see Toolbar for a detailed description.

Tools : List of additional tools

- **Manufacturing Tool** : Opens the **Manufacturing Tool**. For more information, see Manufacturing Tool.
- **Flash Programmer** : Opens the tool for flash programming and modifications. For more information, see Flash Programmer.
- **SB editor** : Allows creating a custom Secure Binary file for secure updates. For details, see SB editor.
- **Merge Tool** : Allows merging up to 8 images into one single binary image. For more information, see Merge Tool.
- **MCUboot - Sign Image…** : Allows signing the application image with the key for the MCUboot secondary bootloader.
- **MCUboot - Export Key…** : Exports the public key from the given private key as a C source file for the MCUboot secondary bootloader.

Help : User help and additional general information

Commands opening the local documentation:

- **User Guide** : Opens the User Guide PDF.
- **Release Notes** : Opens the Release Notes markdown file.
- **Quick Start Guide** : Opens the Quick Start Guide PDF.

Commands opening the online documentation:

- **Online Documentation** : Opens the online SEC Tool documentation (quick start guide, user guide, and release notes).
- **Community** : Opens an NXP webpage with the blog, where you can find discussions about issues related to this tool.
- **SPSDK Documentation** : Displays a web page with documentation for the NXP Secure Provisioning SDK command-line tools.

Additional commands:

- **Check for Updates** : Checks whether there is a new version of the tool available.
- **About** : Displays information about the current version.

5.1.3 Preferences

User preferences are stored in the folder `<user home>\.nxp\secure_provisioning_<version>\` and are shared for all workspaces. The preferences are backward-compatible, so for example SEC Tool v9 can load preferences from SEC Tool v8 if preferences for v9 do not exist yet. If no preferences are available, the SEC Tool starts with default values.

User preferences contain information about recently used files and workspaces, window sizes, locations, positions of the splitters, and options configurable in the Preferences dialog:

Timeout for communication re-established after reset (flashloader to be initialized) [sec] : Represents a delay (in seconds) after which the ROM bootloader or flashloader will be ready after reset of the processor. The real value may be affected by the configuration of your host. The selected value may affect the generated write script.

Maximal number of recent workspaces displayed in the File menu : Customize the maximal number of recent workspaces displayed in **File > Recent Workspaces**. Supported range 1 - 25. Default value: 9

Read current values after the OTP/PFR configuration is opened : Choose how the reading of device values on opening the **OTP Configuration** is handled.

The following options are available:

- **Never** : Do not read the values automatically
- **Ask** : Confirm the reading manually
- **Always** : Automatically read device values

Preferred language for the tool : Select the language in which the tool will be displayed. Supported languages are English and Chinese.

The following options are available:

- **Default** : The language is selected based on the language selected in the operating system. If the system language is different from the supported languages, English is used.
- **EN** : Set the tool to English
- **ZH** : Set the tool to Chinese

Save tool settings : Specifies when the tool settings must be saved to the disk.

It is possible to select one of the following options:

- **Automatically** : It is the default value. The settings are saved if needed
- **On request only** : The tool always asks whether to save the settings or not

Sound on error during configuration : If a new error is displayed in the configuration dialog, the tool notifies the user with a sound signal. The sound signal is OS-specific.

Password/key visibility at startup : Specifies whether the passwords and keys are shown or hidden at the SEC Tool startup. In the UI, the visibility can be controlled by the button, so this option configures just the initial value.

Check for the new version of the tool : Specifies how frequently the tool checks the availability of the new version during the start-up.

It is possible to select one of the following options:

- **Never** : Never check; the feature is disabled
- **Daily** : The check is executed once per day
- **Weekly** : The check is executed once per week
- **Monthly** : The check is executed once per month

Help NXP improve the tool by sending diagnostic data : If enabled, this feature allows the collection of diagnostic data to generate statistical insights into how the tool is used. This helps us enhance functionality and prioritize improvements. If disabled, no data is collected or transmitted.

Statistical Data Collection Notice

To support continuous improvement of our software, we collect diagnostic usage data that include:

- Frequency of tool and user interface feature usage

- Operating system details (type, version)
- Device configuration
  - Target device, boot type, boot device type, TrustZone status (enabled/disabled), connection type (USB, UART, and so on)
  - Trust provisioning method (Device HSM, EL2GO), Debug Authentication (enabled/disabled), Signature Provider (enabled/disabled), Debug Probe type

Although the types of data collected are clearly listed above, we explicitly confirm that the following are not collected:

- Personally identifiable information (PII), such as names, email addresses, or contact details
- User-generated content, including project files, source code, or any data stored on the user's device
- Location data or behavioral tracking is outside the scope of the tool usage
- Sensitive project information (for example, fuses, cryptographic keys, file paths or file names)

All collected data is non-personal, not shared with third parties, and is used solely to improve user experience and guide feature development.

Restricted data … : The data can be downloaded from the Secure Provisioning Tool home page, however, access to this package is contingent upon the execution of a Non-Disclosure Agreement (NDA). Among other, the data contains trust provisioning firmware. The data are installed from a ZIP archive. SEC first verifies whether the selected data are compatible with the current tool and if yes, the data are copied into the subfolder in the user home directory. To start using the data, restart SEC.

Use restricted data from directory : Allows controlling whether the restricted data are used. The checkbox is enabled only if restricted data are installed.

5.1.4 Workspaces

All files generated by the tool are stored in a dedicated folder structure called a workspace.

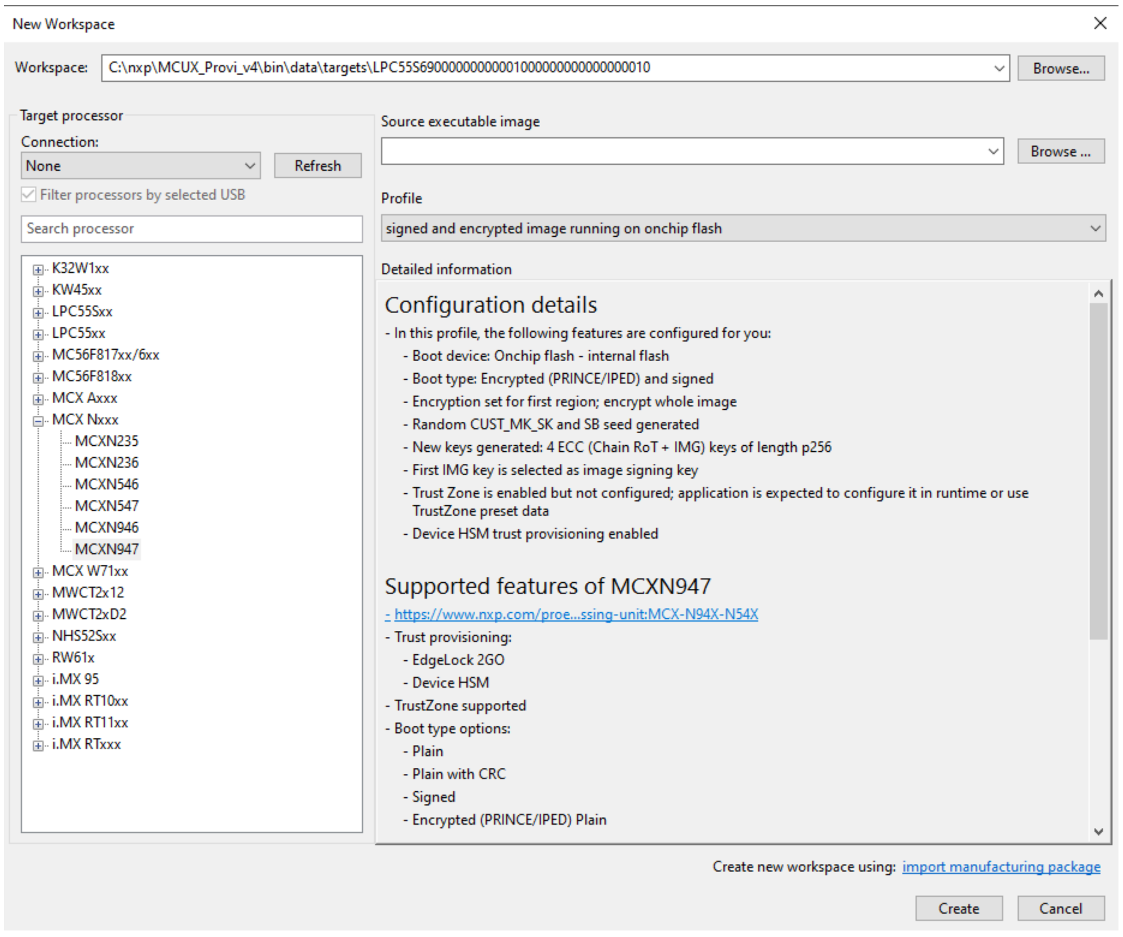
A workspace is a practical concept for operating with multiple boards, devices, or executables signed with different sets of keys. It is recommended to create a workspace for every project.

A workspace is always created for a specific device family (series of processors). Once created, it can only be used to modify the configuration of devices belonging to that family.

There are two types of workspaces:

- Development workspace is used during regular tool operation.
- Manufacturing workspace is used during manufacturing. See section Manufacturing workflow for details.

To create a workspace, select **File > New Workspace…**, **File > Import Manufacturing Package…** or **File > Import Workspace…** from the **menu bar**. See New workspace creation for details.



Workspace: C:\nxp\MCUX_Provi_v4\bin\data\targets\LPC55S69000000000000100000000000000010

Connection: None

Profile: signed and encrypted image running on onchip flash

Configuration details

- In this profile, the following features are configured for you:
  - Boot device: Onchip flash - internal flash
  - Boot type: Encrypted (PRINCE/IPED) and signed
  - Encryption set for first region; encrypt whole image
  - Random CUST_MK_SK and SB seed generated
  - New keys generated: 4 ECC (Chain RoT + IMG) keys of length p256
  - First IMG key is selected as image signing key
  - Trust Zone is enabled but not configured; application is expected to configure it in runtime or use TrustZone preset data
  - Device HSM trust provisioning enabled

Supported features of MCXN947

- https://www.nxp.com/proe...ssing-unit:MCX-N94X-N54X
- Trust provisioning:
  - EdgeLock 2GO
  - Device HSM
- TrustZone supported
- Boot type options:
  - Plain
  - Plain with CRC
  - Signed
  - Encrypted (PRINCE/IPED) Plain

Create new workspace using: import manufacturing package

Figure 21. Create a workspace

To switch to a different workspace, select **File > Select Workspace …** from the **menu bar** and choose the path from the **Open Workspace** dialog. Another way to open the existing workspace is to double-click the appropriate `settings.sptjson` file in file explorer. It opens the tool with the given workspace.

To switch to a recently used workspace, select **File > Recent Workspaces** from the **menu bar** and choose from the list.

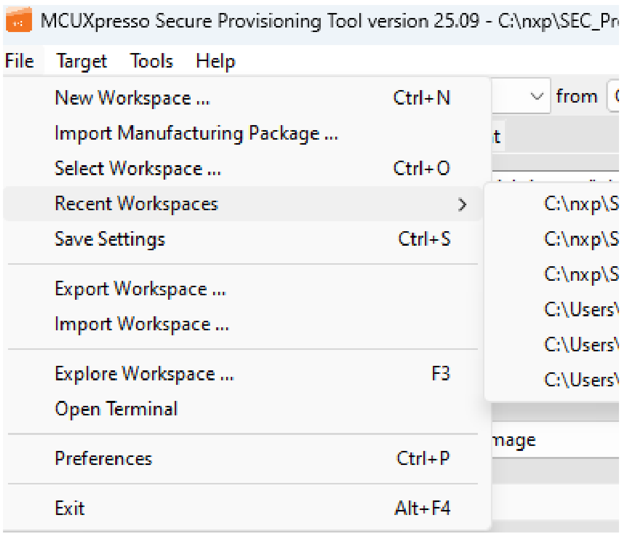


Figure 22. Selecting a recent workspace

Every created workspace contains multiple subfolders. Some of them are specific to device families. For the new workspace, most of the subfolders are empty; the files are generated or added by the user later.

- `backups/` : Backup of old keys/crts after importing or generating new ones
- `bootable_images/` : Intermediate and final bootable images (nxpimage output). The no padding binary starts at the address IVT, while the regular binary includes everything from the beginning of the boot device.
- `configs/` : Generated configuration files, for example, OTFAD/IEE config file (YAML), BEE user keys config file (YAML), and the corresponding generated (BIN) files.
- `crts/`, `keys/` : Generated certificates and their corresponding keys.
- `dcd_files/` : DCD files included in the build image step.
- `debug_auth/` : Debug authentication files generated by the tool, configuration file for certificate generation (YAML), certificate request (ZIP), certificate (DC and ZIP), and authentication script.
- `el2go/` : Files for EdgeLock 2GO trust provisioning
- `gen_bee_encrypt/` : BEE user key files created during the build image step for XIP encrypted boot types. The keys are used to burn SW_GP2/GP4 fuses during the image write step.
- `gen_hab_certs/` : Output super root key table and hash (nxpcrypto output). The table is programmed along with the bootable image. The hash is programmed in platform fuses.
- `gen_hab_encrypt/` : DEK key files generated by the nxpimage utility. The DEK key file is used during write the image to generate the key blob for the encrypted HAB boot type.
- `gen_scripts/` : Temporary scripts for tool operation.
- `hooks/` : Hook scripts allow customizing build, write, or the manufacturing process. Hooks with prefix “pre” are executed before the script is executed. Hooks with “context” in the name are executed at the beginning of the script after the environment variables are defined. They can be used for environment variables redefinition. General hook files are called multiple times during the script execution. For details, see Build image, Write image, Manufacturing Tool.
- `logs/` : Log files; log.txt contains content/history of the Log view. The folder also contains logs from SPSDK command-line applications and from the manufacturing.
- ***root folder*** : Contains the following:
  – Last configuration of the tool in file `settings.sptjson`.
  – Build and write scripts.
  – Build and write JSON files containing all parameters used to generate the build and write scripts.
- `source_images/` : Primarily intended as a folder to store input images provided by users. Also used by the tool to store input images, if input format conversion is needed.

- `trust_provisioning/` : Trust provisioning files (supported only if the processor supports trust provisioning)
- `trustzone_files/` : TrustZone-M configuration files (YAML, JSON, or BIN) used by nxpimage during bootable image generation step (nxpimage input).

5.1.4.1 New workspace creation

To create a workspace, select **main menu > File > New Workspace**. The following options can be selected in the **New Workspace** dialog:

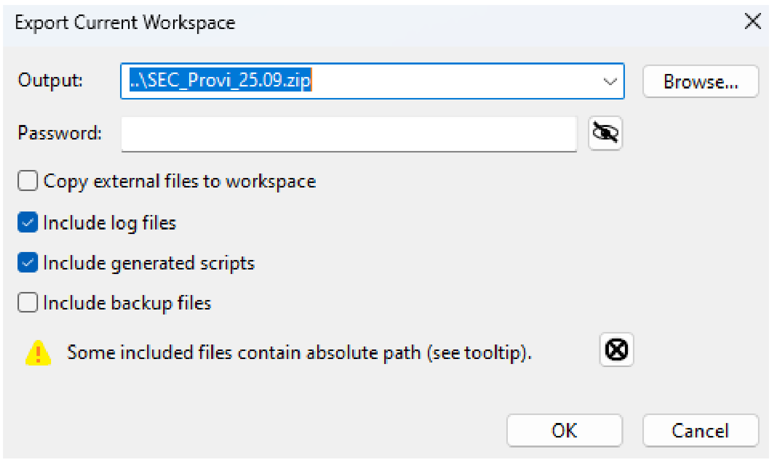
- **Workspace path** : The folder where the workspace is created; use an empty or non-existing folder.
- **Connection** : Connection that should be used in the new workspace for communication with the processor. If no connection is selected, a new workspace is created with the default connection for the processor.
- **Refresh button** : Update the list of detected connections.
- **Filter processors by selected USB** : When checked and the USB connection is selected, the device tree is filtered by the device family identified by USB VID/PID
- **Search processor** : Filter processors from the processors tree. A substring of the processor, series, or board name can be used.
- **Processor tree** : Displays processors matching the filter above. The processors are organized in folders by high-level processor category and series.
- **Source executable image** : Chooses the input executable application image file. For more information about the input image format, see Source image formats.
- **MCUboot bootloader image** : Chooses the input MCUboot bootloader image. For more information about MCUboot, see MCUboot workflow.
- **Profile** : Pre-defined settings that will be applied to the new workspace. It is recommended to start development with the default profile and verify the plain image works. For secure profiles, the Quick Fix will be applied to the configuration so there are no errors by default. To create a custom profile, see Profile creation.
- **Detailed information** : The configuration details section displays the information about pre-configuration of the workspace based on the profile selection. The “Supported features” section lists the processor features that are supported by the tool.

5.1.4.2 Sharing and copying workspaces

It is recommended to store all used files in the workspace. The `settings.sptjson` file contains all paths relative to the workspace root folder, so if you open settings on another computer, you can still regenerate all scripts. The workspace can be a part of another project. Paths to the parent folder are stored as relative, but paths to other folders are absolute. It is recommended to use the **Export Workspace** dialog for sharing workspace.

In case the script must be executed on another computer without regeneration, it uses environment variables to specify the SEC installation directory and workspace. These environment variables can be specified externally, or if not specified, the default value is used. Workspace is detected automatically and an environment variable must be specified only if the script is copied outside the workspace.

The workspace can be exported using the **Export Workspace** dialog, select **File > Export Workspace …** from the **Menu bar**. The dialog box displays options for exporting the current workspace into a ZIP file with optional AES encryption.



Output: ..\SEC_Provi_25.09.zip

Password:

☐ Copy external files to workspace

☑ Include log files

☑ Include generated scripts

☐ Include backup files

Some included files contain absolute path (see tooltip).

Figure 23. Export Workspace

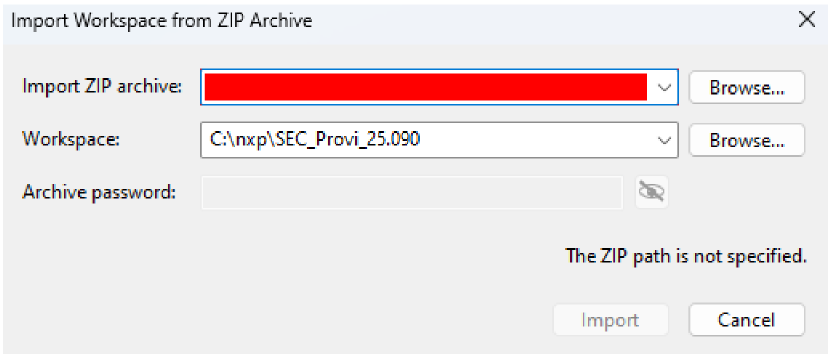
The Export Workspace dialog allows selecting the following options:

- **Output** : Path to the output ZIP file to be created. Any existing file will be overwritten.
- **Password** : Optional password for a ZIP file encryption. Leave empty for no encryption.
- **Copy external files to workspace** : Select this option to copy all external files (files located outside the workspace directory) into the exported workspace. A list of all external files is shown in the tooltip. All files will be saved in the **source_images** folder. Filename conflicts are automatically resolved to ensure that no file is overwritten. The current workspace and settings are not affected. The operation affects the ZIP archive only. The option is not available if any file does not exist.
- **Include log files** : Select this option to include all log files in the exported workspace.
- **Include generated scripts** : Select this option to include all generated scripts. The scripts contain an environment variable to specify the workspace the SEC installation directory, which is an absolute path. Regeneration of the scripts may still be needed.
- **Include backup files** : Includes files from the “backup” sub-folder.

A warning message is shown when there are some absolute paths detected in the included files. A list of these files is presented as a tooltip.

5.1.4.3 Import workspace from the ZIP

For importing the manufacturing package, select **File > Import Manufacturing Package …** from the **menu bar**. For importing the workspace archive, select **File > Import Workspace …** from the **menu bar**.



Import ZIP archive:

Workspace: C:\nxp\SEC_Provi_25.090

Archive password:

The ZIP path is not specified.

Figure 24. Import Manufacturing Package or Workspace Archive dialog

- **Import ZIP archive** : The path to the ZIP workspace archive.

- **Workspace** : The folder where the imported workspace is created; it is recommended to use an empty or non-existing folder.
- **Archive password** : A ZIP password for encrypted archives. This field is enabled only when encryption is detected.

The dialog allows importing both workspace from the ZIP archive and manufacturing package. In both cases, the workspace is imported into a new directory (or empty directory). For the workspace import, the tool supports imports from older versions, but for the manufacturing package, the tool and the manufacturing package version must be the same. If the ZIP file is encrypted, write the correct password. For more information about the manufacturing process, see Manufacturing operations.

5.1.4.4 Profile creation

A profile for New Workspace wizard can be created from any workspace using `settings.sptjson`. The profile is a subset of the workspace setting that is relevant for the profile. Profiles must be placed in the `<install_folder>/bin/_internal/sample_data/targets/<processor name>/configuration_profiles` folder to be offered as an option for the profile selection in the **New Workspace** dialog. To create a valid profile, manually specify these JSON fields:

- **is_profile** : a flag identifying a profile setting, it must be set to **true**
- **profile_name** : the name of the profile; this name is displayed in the tool and must be processor-unique
- **profile_description_introduction** : the profile description in the natural language. It is displayed before the formatted profile description. This field is optional.
- **profile_description** : should contain information what is set by the profile. It must be specified as a list of information.

Fields that should be removed from the profile settings:

- **write_image_settings**, **keys_management_settings**, and **connection** have no meaning for the new workspace.
- From **build_image_settings**, remove all fields that contain secrets that should not be distributed (such as **dek_key**, **keyblob_key_id_int**, **sbkek**, **sb_seed**) or fields that are irrelevant for the new workspace (**source_image_path**, **dcd_path**, **ele_firmware_path**, **xmcd_path**, **tz_path**, **script_updated**, **updated**)

The description of all fields can be found in the JSON schema **settings_schema_##_##.json** distributed within the tool installation files.

5.1.5 Toolbar

The **toolbar** offers a quick selection of basic settings. The same commands are also available in **main menu > Target**.

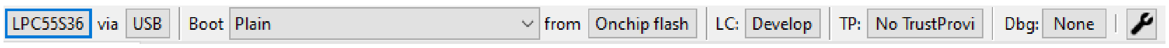


Figure 25. Toolbar

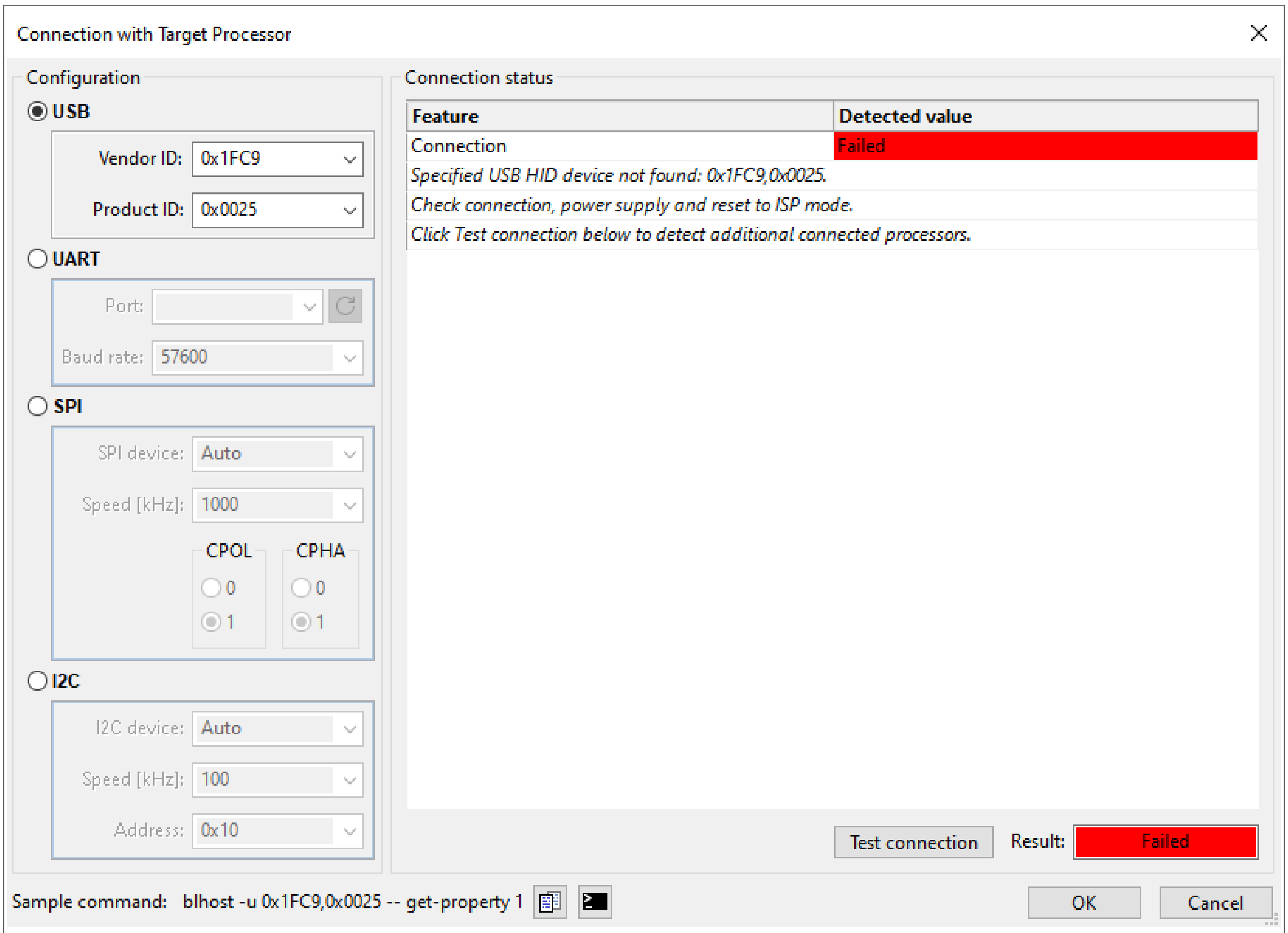
- **Processor** : Shows the chosen processor. Click the button to switch the processor. You can switch to a processor from the same device family for which the current settings are compatible. To select a processor from a different family, create a new workspace.
- **Connection (via)** : Choose the connection to the target. This release supports UART, USB-HID, SPI, and I2C connectivity. Click the button to customize connection details. For more information, see Connection.
- **Boot mode** : Choose the type of boot. The list depends on the device capabilities of the currently selected processor.
- **Boot memory (from)** : Click the button to open the boot memory configuration. For more information, see Boot memory configuration.

- **Life cycle** : Allows the selection of the processor life cycle. Click the button to select from processor-specific life cycles; the selection dialog displays a short description for each option.
- **Trust provisioning type** : Allows the selection of the trust provisioning type and enabling it for the trust provisioning operation. For details, refer to Trust provisioning.
- **Debug probe** : Allows the selection of the debug probe connected to the computer; see Debug Probe Selection dialog.
- **Quick fix** : Resolve problems that are displayed on the build tab. Not all problems can be fixed by the "quick fix" solver as not all problems have a single deterministic solution. Before the solver does any changes, the user is prompt to save settings. Once the quick fix is complete, a list of actions that were applied to fix the problems is displayed; the actions are also displayed in the log. It is recommended to review all changes carefully to ensure that the solution meets the expectations.

5.1.6 Connection

The **Connection** dialog allows you to select the connection with the target processor and test it.

The dialog is accessible from **Target > Connection** from the **menu bar** or the **toolbar**.



Configuration: [✓ USB] Vendor ID: 0x1FC9, Product ID: 0x0025

UART: Port: (empty), Baud rate: 57600

SPI: SPI device: Auto, Speed [kHz]: 1000, CPOL: 0 [✓ 1], CPHA: 0 [✓ 1]

I2C: I2C device: Auto, Speed [kHz]: 100, Address: 0x10

Feature	Detected value
Connection	Failed
Specified USB HID device not found: 0x1FC9,0x0025.	
Check connection, power supply and reset to ISP mode.	
Click Test connection below to detect additional connected processors.	

Result: Failed

Sample command: blhost -u 0x1FC9,0x0025 -- get-property 1

Figure 26. Connection dialog with errors and proposed next steps

It contains the following options (if supported for the processor):

- **USB** : Specify USB connectivity to the specified Vendor ID/Product ID pair.

- **UART** : Specify UART connectivity through the specified port and baud rate. The baud rate is automatically detected by the bootloader when processing the initial ping. This means that the target processor must be reset after a new baud rate has been selected.
- **SPI/I2C** : It is possible to connect with the processor using an SPI or I2C connection using the LIBUSB interface available on:
  - MCU-Link Pro
  - LPC-Link2. LPC-Link2 is present on several EVK boards, however, to connect via SPI you must use jumper wires to connect with the processor.

Before starting the Secure Provisioning Tool, it is necessary to download and install USB drivers from the product pages listed above. It is possible to configure the following connection parameters:

- **SPI/I2C device** : The device is specified using a USB Path. The default value in the connection dialog is “Auto”, which means if there is just one device connected to the computer, it is selected automatically. The details about the USB Path format can be found in the SPSDK documentation.
- **Speed [kHz]** : Communication clock frequency in kHz.
- **CPOL, CPHA** : Signal polarity and phase; see SPI specification for details.
- **Address** : Address of the I2C device.

Crystal frequency [kHz] : Frequency of the external crystal. It is used to configure communication speed with the processor. Currently it is used only for `lpcprog`.

Use the **Test connection** function to verify that the device can be properly accessed with the given configuration. To ensure successful detection of the processor with **Test connection**, make sure of the following:

- The board is correctly powered up
- The board is properly configured to ISP (In-System Programming) mode
- The board is connected to the computer

The connection dialog detects the following parameters:

- **Connection** : Status of the selection of a communication device (USB or serial port) or a USB path for the SPI/I2C connection
- **Mode** : Communication mode bootloader or flashloader application.
- **Processor** : *Match* if the connected processor matches the selected one, *No match* otherwise. This feature allows finding a mistake when the SEC Tool is communicating with a wrong board. This function is not 100% reliable as there is not enough information to identify each processor.
- **Life cycle** : Life cycle that was detected in the connected processor.
- **Silicon version** : Displayed only if the connected processor is of an older version. When the old revision is detected, the connection test results in a Failed state. All known issues related to this version are mentioned in the tooltip.

The following connection results are possible:

- **Not tested yet** : Use the **Test connection** button to run tests.
- **OK** : Connection successfully established.
- **FAILED** : Connection tests failed. For details, see **Connection status**. For more information about the failure, it is possible to use SEC in verbose mode and find details in the console view.

At the bottom of the connection dialog, there is the “Sample blhost/sdphost/lpcprog/nxpuuu command” that allows running the corresponding SPSDK command-line tools with specified arguments. The first button copies the command with arguments into the clipboard. The second button opens a terminal where the command can be executed.

5.1.7 Boot memory configuration

The Boot Memory Configuration dialog allows selecting and configuring a boot device. The dialog contains the following configuration parts:

- **Boot memory type** : This part allows selection of the boot memory type, and optionally, instance.
- **Predefined template** : This part allows selection of the boot memory configuration template. The list is specific for each memory type and contains memories where some are available on NXP evaluation boards. After opening the boot memory configuration dialog, the option contains the previously selected memory (if values were not changed), or is empty, the first item in the drop-down menu is a memory used on the evaluation board (if applicable). Memory configuration templates that are marked as verified were tested on hardware.
- **User configuration** : This part allows loading or saving the configuration to the selected file. It might be useful for reuse of the configuration for another project or sharing the configuration with colleagues.
- **Protected area** : This part allows specifying the memory area that must not be changed by the SEC Tool. If the tool tries to erase or modify the selected memory area, a confirmation dialog is displayed. It might be useful for protection of the custom data in a boot memory. Specify comments/reasons, because it will be displayed as part of the confirmation message.
- **Boot memory configuration parameters** : Configuration of the memory, these parameters are specific for each memory type.
- **Comment** : The description of the boot memory that contains information if the predefined template was applied
- **Test the configuration** : This button is used to test the current memory configuration with the connected processor/board. The test consists of two steps: read test (whether the memory can be read) and bootloader test that verifies the memory configuration using the blhost list-memory command.
- **Convert to Complete FCB** : This option is available for SPI NOR only. The button allows converting a simplified SPI NOR configuration into a full “Flash Control Block” (see SPI NOR)

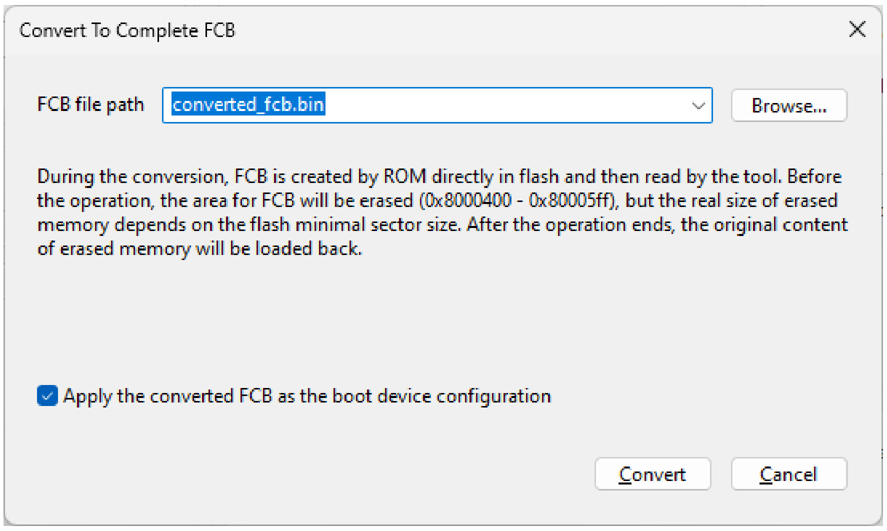
5.1.7.1 SPI NOR

SPI NOR flash can be configured in two ways:

- by using the flashloader/ROM based simplified configuration (two 32-bit words)
- by using the complete Flash Control Block (FCB binary, 512 bytes)

For a simplified configuration, the user can modify the values suggested in the dialog box. To create your own FCB from SEC Tool, use the **Convert to complete FCB** button in the bottom-right corner. It opens the **Convert To Complete FCB** dialog box that allows to convert existing simplified configuration into complete configuration using the connected processor. A simplified device configuration is written to an SPI NOR flash device and read back while the original data are preserved. When the conversion process is complete, it creates a `.bin` file in the desired location, given by the path to the FCB file.

The **Convert To Complete FCB** dialog offers a checkbox to use the created FCB binary file as a SPI NOR/**user FCB file**. If unchecked, only the conversion is done.


Convert To Complete FCB

FCB file path: converted_fcb.bin

During the conversion, FCB is created by ROM directly in flash and then read by the tool. Before the operation, the area for FCB will be erased (0x8000400 - 0x80005ff), but the real size of erased memory depends on the flash minimal sector size. After the operation ends, the original content of erased memory will be loaded back.

[✓] Apply the converted FCB as the boot device configuration

Figure 27. Convert to Complete FCB dialog

FCB can also be created in MCUXpresso IDE by adding the FCB component into Peripheral Drivers in Peripherals tools, where the full configuration can be specified.

When the complete FCB is specified in the boot device configuration, the user can specify two separate FCB files. The FCB for runtime is used for flash configuration during the processor boot. The FCB for write is used for flash configuration for programming the application.

5.1.7.2 On-chip RAM

The SEC Tool allows creating images executed in internal RAM, which might be useful for chip (re-)configuration or executing a diagnostic test. For this boot memory, the write script writes the application into the processor and intermediately launches it. If the chip is secured, the application is written and launched via the SB file.

Note: The SB file could also be used for recovery flash.

5.1.7.3 Serial downloader

The serial boot provides a means to download a boot image to the chip and execute the image. In this boot mode, a host PC/device can communicate to the ROM bootloader, using the serial downloader protocol. The SEC Tool allows execution of the image in internal or external RAM. It might be useful to distribute the application to flashless devices.

The write script contains both provisioning of secure assets to the processor and loading and execution of the application in RAM. During manufacturing, the application is not loaded.

5.1.8 Trust provisioning

The **Trust provisioning** dialog allows selecting a processor-specific trust provisioning type and enabling it for the trust provisioning operation.

The SEC Tool supports the following trust provisioning types:

- **Device HSM** - the secrets are encrypted using a key stored in a processor. It is the same for all processors in the series.
- **EdgeLock 2GO** - the secrets are stored in the NXP cloud, on the EdgeLock 2GO server.

5.1.8.1 Device HSM provisioning

There are two types of device HSM provisioning:

- The device HSM is directly supported in ROM. In this case, there is no need for extra firmware.
- There is extra firmware needed for the following use cases:
  - i.MX RT5xx/6xx where the device HSM is optional. For details on how to enable it, see Device HSM provisioning. This firmware is distributed in a restricted data package (see **main menu > File > Preferences** to install it).
  - RW61x processors where the device HSM is mandatory for secure boot types are partially supported in ROM. This firmware is distributed in tool data.

To successfully build an image with Device HSM enabled, the processor must be connected and selected in the Connection dialog. It is not required only when there is already an image built with Device HSM and the settings are the same.

5.1.8.2 EdgeLock 2GO trust provisioning

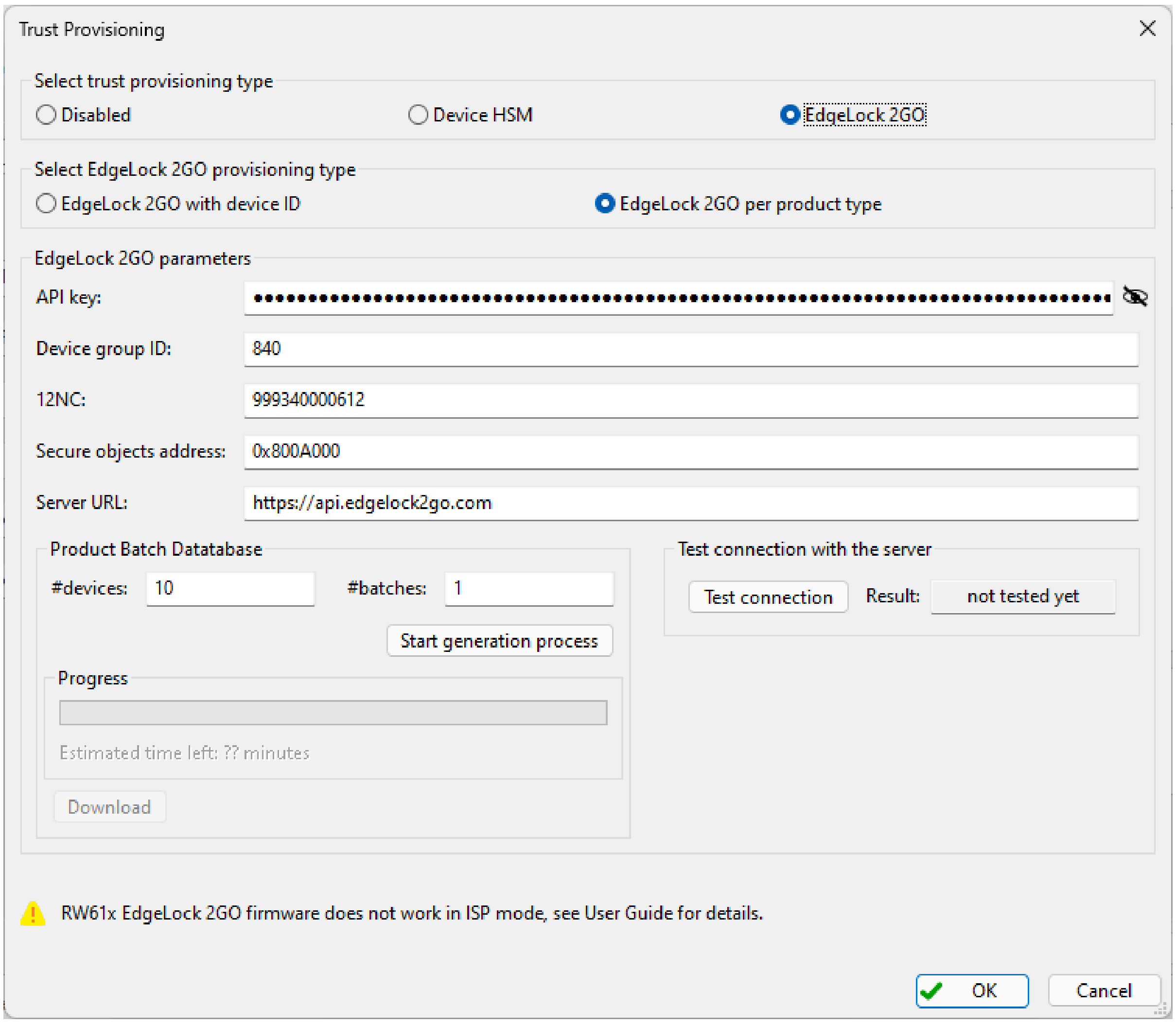
The EdgeLock 2GO parameters allow configuration of the access to the assets on the EdgeLock 2GO server.

There are two EdgeLock 2GO flows supported:

- **EdgeLock 2GO with device ID**: the secure objects are retrieved from the EdgeLock 2GO cloud server during the provisioning process
- **EdgeLock 2GO per product type**: the secure objects are retrieved from a server in a batch database and provisioning can be executed without connection to the cloud server (offline)

Other configuration controls:

- **API key** - user-specific HEX key granting access to the server. For details, see API key to access EdgeLock 2GO server.
- **Device group ID** - the identification number of the device group on the EdgeLock 2GO server that contains all the secure assets. On the EdgeLock 2GO server, it can be found under **Devices > MCU & MPU**.
- **12NC** - hardware product unique identification. On the EdgeLock 2GO server, this can be found within the information about the device group.
- **Secure objects address** - address in the flash memory where the secure objects are stored during provisioning.
- **Server URL** - URL of the EdgeLock 2GO REST API server. Use an empty string to apply the default value.
- The **Test** button allows verifying a connection to the server.
- **Product Batch Database** panel allows to generate and download a database of secure objects from the EdgeLock 2GO server and use it for offline **per product** provisioning flow. For details, see Create database with secure objects for EdgeLock 2GO per product type.
  - **#devices** - number of processors to be included in the batch database
  - **#batches** - number of batch databases to be generated



Trust Provisioning

Select trust provisioning type
( ) Disabled ( ) Device HSM [✓ EdgeLock 2GO]

Select EdgeLock 2GO provisioning type
( ) EdgeLock 2GO with device ID [✓ EdgeLock 2GO per product type]

EdgeLock 2GO parameters

API key:	••
Device group ID:	840
12NC:	999340000612
Secure objects address:	0x800A000
Server URL:	https://api.edgelock2go.com

Product Batch Datatabase
#devices: 10 #batches: 1

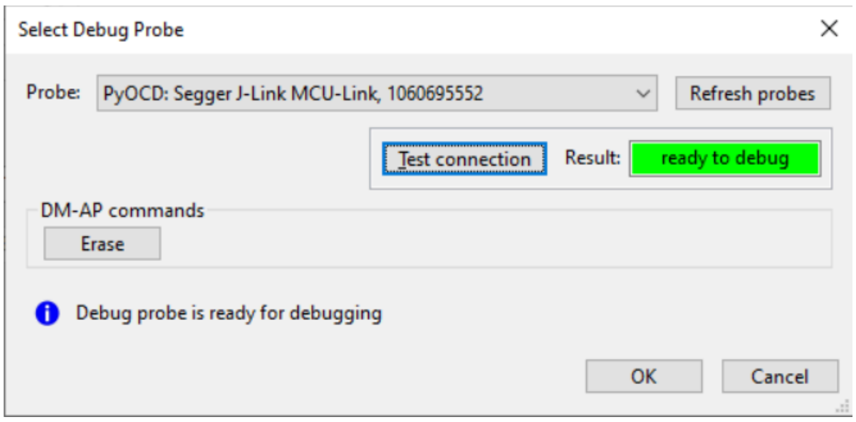
Progress
Estimated time left: ?? minutes

Test connection with the server
Result: not tested yet

RW61x EdgeLock 2GO firmware does not work in ISP mode, see User Guide for details.

Figure 28. EdgeLock 2GO trust provisioning

5.1.9 Debug Probe Selection dialog



Select Debug Probe

Probe: PyOCD: Segger J-Link MCU-Link, 1060695552

Result: ready to debug

DM-AP commands

Debug probe is ready for debugging

Figure 29. Debug Probe Selection dialog

The Debug Probe Selection dialog allows to:

- Select a probe for shadow registers (only for processors where the shadow registers for fuses are initialized via a debug probe)
- Use a test connection with the given probe
- Erase the processor flash (if supported in the processor via the debug probe API)

When the dialog is opened, the probes connected to the computer are detected. It is possible to rescan the probes anytime later again using the **Refresh probes** button.

Once the probe is selected, the selection is stored in the workspace settings including the hardware ID (serial number). If a different board is connected, update the selection. During opening the dialog, if the selected probe is not found, the tool updates the hardware ID automatically.

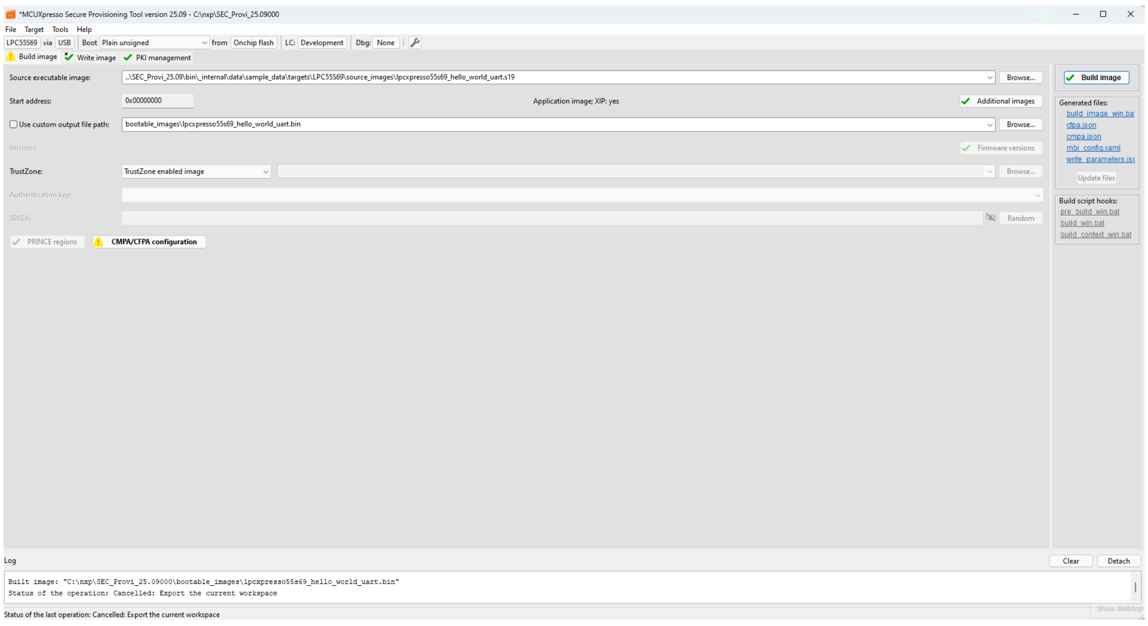
The **Test connection** button provides information on whether the debug can be started for the selected debug probe. The possible results are:

- **ready to debug** - if the debug probe and the processor are ready for debugging
- **no debug** - if the connection with the debug probe was established, but the processor cannot be debugged; in this case ensure that the debugger is properly connected and the processor is running (not ISP mode)
- **FAILED** - if the connection with the debug probe failed

The **Erase** button allows erasing the internal flash (mass erase). See the reference manual for processor-specific details.

5.2 Build image

In the **Build image** view, you can transform an application image into a bootable format compatible with the selected processor.



LPC55S69 via USB | Boot Plain unsigned from Onchip flash | LC: Development | Dbg: None

Source executable image: ..\SEC_Provi_25.09\bin_internal\data\sample_data\targets\LPC55S69\source_images\lpcxpresso55s69_hello_world_uart.s19

Start address: 0x00000000

Application image; XIP: yes

Use custom output file path: bootable_images\lpcxpresso55s69_hello_world_uart.bin

Versions:

TrustZone: TrustZone enabled image

Authentication key:

SBKEK:

Generated files:
build_image_win.ba
cfpa.json
cmpa.json
mbi_config.yaml
write_parameters.jsc

Build script hooks:
pre_build_win.bat
build_win.bat
build_context_win.bat

```
Built image: "C:\nxp\SEC_Provi_25.09000\bootable_images\lpcxpresso55s69_hello_world_uart.bin"
Status of the operation: Cancelled: Export the current workspace
```

Status of the last operation: Cancelled: Export the current workspace

Figure 30. Build image (LPC55Sxx)

5.2.1 Controls on the build image view:

Source executable image : Chooses the input executable file. For more information about the input image format, see Source image formats.

Start address : The base address of the image. It is editable only to a binary image. For ELF and S-Record and HEX files, it is detected automatically.

Application/Bootable image and XIP : The label provides the following information about the selected source image:

1. whether it is an application or a bootable image.
2. whether the selected source image is built as "eXecuted In Place", and will be executed from the boot memory, where it is stored. If not, the image must be copied to RAM before the execution. The information is derived from the starting address of the image and compared with the memory address of the selected processor, so the result might not be correct if the selected image does not match the selected processor.

Additional images : Opens the configuration dialog for the Additional User/OEM Image, see Additional images.

Use custom output file path : Name of the generated bootable image file and its location. If not specified, the tool names the image based on the input. The file extension is specific for a processor and a boot type, it is either BIN for bootable images or SB for secure-binary capsules.

Image version : Version of the bootable image. It is used for dual image boot. The image with the higher image version is booted first.

Dual image boot : Opens the configuration window for dual image ping-pong boot. Image can be written to the base (image0) and/or to the remapped (image1) space of the flash, each of the them has its own image version. The ROM then uses the image version to select the latest image to boot. If the latest image boot fails, the old image is used to boot again.

Firmware versions : Opens the configuration dialog of Firmware versions (Secure and Non-secure). The firmware version allows setting the anti-rollback protection. For details, see Firmware versions.

XMCD : Allows enabling the External Memory Configuration Data feature. XMCD is needed for comprehensive or feature-rich applications requiring large capacity of RAM (on-chip RAM is not enough). Either a YAML or BIN configuration file can be provided or XMCD simplified configuration can be prepared in XMCD configuration dialog. (for details, see the description of the **-xmcd-cfg** CLI parameter).

DCD (Binary) : Selection of what Device Configuration Data must be included in the bootable image. The option **From source image** can be used only if the source image contains DCD. The DCD enables early configuration of the platform including SDRAM. MCUXpresso Config Tools can generate a DCD in a compatible format. If the target processor does not support DCD files, the checkbox is disabled. For more information, see Creating/Customizing DCD files.

TrustZone : Allows you to enable TrustZone features. The following selection is possible:

- **TrustZone disabled image** - Disables TrustZone. This option might not be supported for some processors.
- **TrustZone enabled image** - Enables TrustZone with the default configuration preset in the processor.
- **TrustZone enabled image with preset data** - Enables TrustZone with custom TrustZone-M data. JSON and BIN file formats are supported. JSON data can be generated in and exported from the TEE Tool of MCUXpresso Config Tools. BIN file is created by the nxpimage utility. For more information, see TrustZone configuration file.

Authentication key : Signs the image with the specified key. The key can also be used for the authentication of the SB file. This option is only applicable to authenticated and encrypted boot modes and offers a selection of keys generated in the **PKI management** view.

Key id : The keyblob encryption key identifier is used in the encrypted (AHAB) boot type

AHAB/HAB encryption algorithm : Selection of AHAB/HAB encryption algorithm used in the encrypted (HAB), authenticated, or encrypted (AHAB) boot types.

Key source : A key source for signing the image.

User key : For the OTP key, the source master key is used to derive other keys. For the PUF KeyStore, the user key is used to sign the image. Only available for Plain signed boot types.

SBKEK, SB3KDK, or CUST_MK_SK : A key is used as a key-encryption key to handle an SB file. Only available for secured boot types. For RT5xx/6xx, it is only enabled when the key source is KeyStore. For LPC55Sxx devices, the key store is initialized only once in the device life cycle and after that, any change in SBKEK will cause failure to load the SB file into the processor. For more information, see PFR and PUF KeyStore. OEM seeks a hex key used to randomize the creation of the SB file with the CUST_MK_SK key.

Configuration dialogs : The following configuration dialogs are available on the Build image view:

- **XIP encryption (BEE user keys)** : Opens the configuration dialog of XIP encryption user keys. Option enabled only for *XIP encrypted (BEE user keys) authenticated* and *XIP encrypted (BEE user keys) unsigned* boot types.
- **XIP encryption (BEE OTPMK)** : Opens the configuration window of BEE with the OTP Master Key. The option is enabled only for XIP encrypted (BEE OTPMK) authenticated boot type.
- **IEE encryption** : Opens the configuration dialog of IEE encryption. The option is enabled only for IEE encrypted boot types.
- **OTFAD encryption** : Opens the configuration dialog of OTFAD encryption. Option enabled only for OTFAD encrypted boot types. For RT10xx devices, the button name is XIP encryption (OTFAD user keys).
- **XIP encryption (OTFAD OTPMK)** : Opens a configuration window of OTFAD with OTP Master Key. The option is enabled only for XIP encrypted (OTFAD OTPMK) authenticated boot type.
- **PRINCE/IPED regions** : Opens a configuration window for encrypted PRINCE/IPED regions allowing specifying which flash regions will be encrypted.
- **OTP/PFR/IFR/BCA/FCF configuration** : Opens OTP/PFR/IFR/BCA/FCF configuration.

5.2.1.1 Generated files

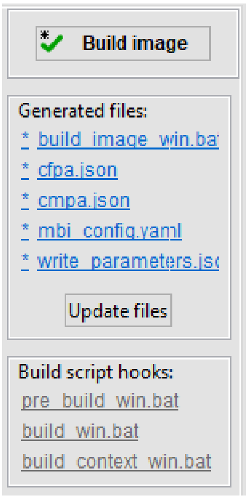


Figure 31. Build image panel

There is a build panel with the **Build image** button and a list of generated files on the right side of the build page. The **Build image** button updates all files and executes the build script. The icon on the button displays an asterisk (*) called "dirty flag" if the build script is not updated or is not successfully executed. The same indication is also displayed in the **Build image** tab.

The generated files below are displayed as clickable links, and the file content is displayed if you click them. The file name starts with an asterisk (*) if the file on the disk is not updated. It might be caused by changes in the configuration. Move the mouse cursor over the * to see the details about changes/differences.

Note: The differences are not supported for binary files or large changes.

This information is updated in the background with some delay, so it might take a couple of seconds before the real status is displayed. Until the information about the file is updated, the file icon is **?** (question mark).

The **Update files** button allows updating all files without the execution of the script.

For the EdgeLock 2GO use case, there is an icon for each file that must be uploaded to the EdgeLock 2GO server.

5.2.1.2 Build script hooks

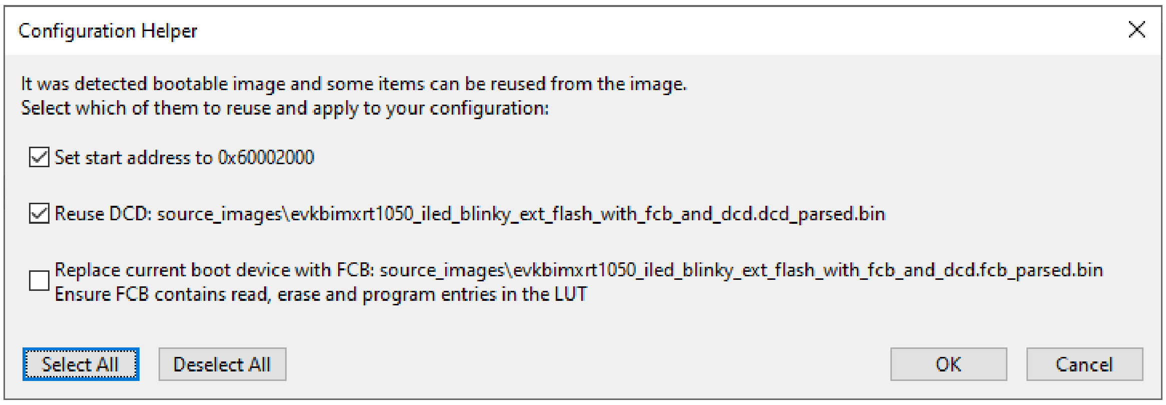
Build script hooks are displayed in the Build image panel below the generated files. Hook scripts are called before or during the build script execution. The gray label means that the file does not exist in the workspace yet. After clicking, a new file is created with the content from the example and opened for modification. If the file exists, the label is blue and after clicking the file is opened. For details, see Script hooks workflow.

5.2.2 Source image formats

SEC supports several formats for source image: ELF, HEX, BIN, or SREC/S19 (S-record). The image format is then unified into the format required by the build script, and this conversion is done inside SEC (the prior build script is called). It is recommended to avoid conversion and use the format needed for the build.

The SEC Tool partially parses the image to retrieve the entry point, to detect whether the image is XIP and validate the target address. In the S19 file format, the entry point can be listed explicitly, while for the other formats the entry point is retrieved from the interrupt vector table (so the interrupt vector table must be at the beginning of the application image).

For some processors, MCUXpresso SDK examples contain a bootable image including FCB configuration. The SEC Tool supports splitting of the bootable images into pieces (FCB, DCD, XMCD sections), and reusing the parts to build a new bootable image. Once a bootable image is selected and the parser accepts the image, the tool offers to reuse specific parts and if confirmed, the configuration is updated.


Configuration Helper

It was detected bootable image and some items can be reused from the image.
Select which of them to reuse and apply to your configuration:

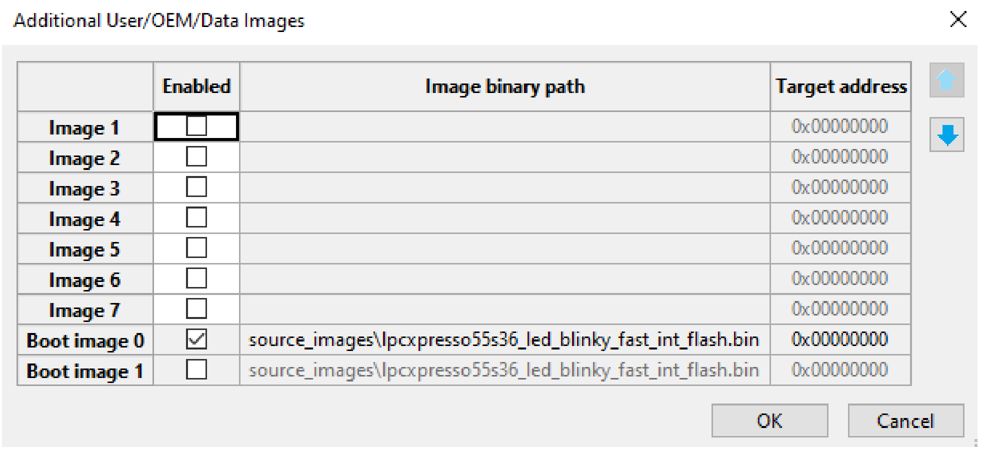
- [✓] Set start address to 0x60002000
- [✓] Reuse DCD: source_images\evkbimxrt1050_iled_blinky_ext_flash_with_fcb_and_dcd.dcd_parsed.bin
- [ ] Replace current boot device with FCB: source_images\evkbimxrt1050_iled_blinky_ext_flash_with_fcb_and_dcd.fcb_parsed.bin
  Ensure FCB contains read, erase and program entries in the LUT

Figure 32. Configuration Helper

5.2.3 Additional images

This dialog allows the user to specify up to 8 user images that will be written into boot memory together with the application. For i.MX 9x processors, the dialog allows specification of up to 16 user images as the processors support primary and secondary container sets. The configurable options (columns) depend on the image format. They are different for AHAB images and the other processors.

The configuration is represented as a table, where each row represents one user image. Columns represent configurable features for each image. A description for each feature can be found in the tooltip. All processors except the i.MX 9x family have the last image reserved for the application executable image, which is automatically updated according to the build tab. The order of any image, except for the application executable image, can be changed by the up/down arrow in the right upper corner.



	Enabled	Image binary path	Target address
Image 1	☐		0x00000000
Image 2	☐		0x00000000
Image 3	☐		0x00000000
Image 4	☐		0x00000000
Image 5	☐		0x00000000
Image 6	☐		0x00000000
Image 7	☐		0x00000000
Boot image 0	☑	source_images\lpcxpresso55s36_led_blinky_fast_int_flash.bin	0x00000000
Boot image 1	☐	source_images\lpcxpresso55s36_led_blinky_fast_int_flash.bin	0x00000000

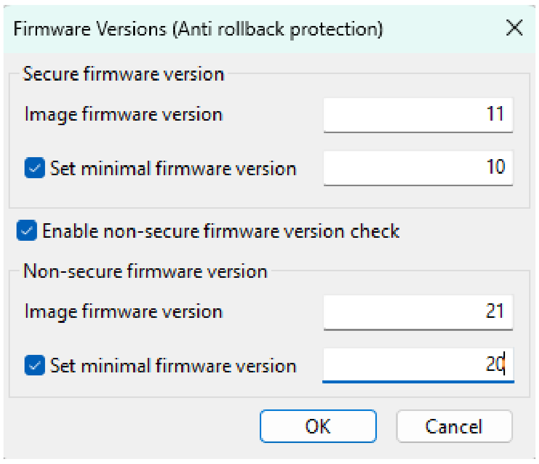
Figure 33. Additional User/OEM Images configuration

The examples of typical usage of the additional images:

- add an image for the second core
- add an application image for the secondary bootloader
- add a binary data from an external file

5.2.4 Firmware versions

The **Firmware Versions** dialog allows configuring anti-rollback protection. In the **Firmware Versions** dialog, specify Secure and Non-secure firmware versions for the image and value for CFPA or fuses that are used for anti-rollback protection.



Secure firmware version
Image firmware version: 11
☑ Set minimal firmware version: 10
☑ Enable non-secure firmware version check
Non-secure firmware version
Image firmware version: 21
☑ Set minimal firmware version: 20

Figure 34. Firmware versions dialog (LPC55S36)

Each panel (Secure firmware version and Non-secure firmware version) is used for a specific firmware version. These versions are checked during the SB file or AHAB image uploading against values in CFPA or against fuse values (the uploaded version must be equal to or greater than the appropriate value in the CFPA or fuses). The Secure firmware version specifies the secure image version checked during booting by ROM.

Specifying values for the Non-secure firmware version is optional and can be enabled or disabled. The non-secure firmware version panel is visible only for processors that support the Non-secure firmware version.

Image firmware version. This version is included in the bootable image and/or it is checked in the SB file or AHAB image. The value must be greater or equal than the minimal firmware version.

Set minimal firmware version. Specify the minimal firmware version stored in CFPA or in fuses for enabling anti-rollback protection. The value must be equal to or smaller than the specific image firmware version. If the value is not specified, it can be specified in the **OTP/PFR configuration** dialog or the default value can be used.

5.2.5 XMCD configuration dialog

The configuration dialog allows customizing parameters for simplified XMCD. These parameters are specific for the selected XMCD memory interface, it can be either:

- FlexSPI/XSPI interface for the HyperRAM/APMemory or
- SEMC interface for SDRAM

The XMCD configuration dialog can be reached by the XMCD **Edit** button on the Build image view when FlexSPI/XSPI RAM or SEMC SDRAM is selected.

The **Import** button allows importing simplified configuration from a YAML or binary file. The **Reset to defaults** button allows returning to default settings.

5.2.6 TrustZone configuration file

TrustZone and related features of the MCU can be pre-configured by data from the application image header at boot time instead of setting the registers from the application code. TEE (Trusted Execution Environment) tool from MCUXpresso Config Tools allows you to export the TZ-M preset data for use in SEC. Follow these steps to modify the existing example application, export the TZ-M file and add it into the application image.

To create, export, and import a TrustZone file, do the following:

1. Open an SDK example:
   a. From MCUXpresso IDE:
      i. In the **Quickstart** panel, select **Import SDK example(s)…**.
      ii. Select the example to import.
      iii. In **Project Explorer**, open the context menu of the imported secure project.
   b. From MCUXpresso Config Tools:
      i. On start, select **Create a new configuration and project based on an SDK example or hello world project**.
      ii. Clone one of the TrustZone enabled (secure) projects.
2. Open the TEE Tool:
   a. In MCUXpresso IDE:
      i. In the **menu bar**, select **MCUXpresso Config Tools > Open TEE**.
   b. In MCUXpresso Config Tools
      i. Select the **TEE** tool from the **Config Tools Overview**.
3. In **Security Access Configuration > Miscellaneous**, use the **Output type** drop-down list to select *ROM preset*.
4. Configure security policies of memory regions as you see fit (for details, see User Guide for MCUXpresso Config Tools (Desktop) document GSMCUXCTUG
5. In the **menu bar,** select **File > Export > TEE Tool > Export Source Files**.
6. In the **Export** window, specify the JSON file download folder and select **Finish**.

7. Remove the *BOARD_InitTrustZone()* call from the *SystemInitHook(void)* function and *tzm_config.h* include located in the main application file (for example, *hello_world_s.c*)

Alternatively, basic TZ-M-preset JSON data included within the SEC layout can also be used as a starting point template for further modifications of TrustZone pre-configuration. Device-specific template files are provided in the `sample_data\targets\<processor>\trust_zone_template.json`

Note: The TrustZone template contains all registers/options with default preset values. Because SAU and AHB are disabled in the template, it is expected that the template will be customized before use.

After the JSON file has been downloaded, you can import it in SEC:

1. In the **menu bar** of SEC, select **File > Select Workspace …** and choose a workspace. Alternatively, create a one by selecting **File > New Workspace ….**
2. In the **Build image** view, switch the **Boot type** to *Signed* or *Unsigned with CRC*.
3. Use the **TrustZone** pre-configuration drop-down list to select **TrustZone enabled image with preset data**.
4. Click **Browse** to navigate to the location of the stored JSON file and select **Open** to import it.

It is possible to use YAML format instead of JSON to import TZ-M preset data into SEC.

5.2.7 OTP/PFR/IFR/BCA/FCF configuration

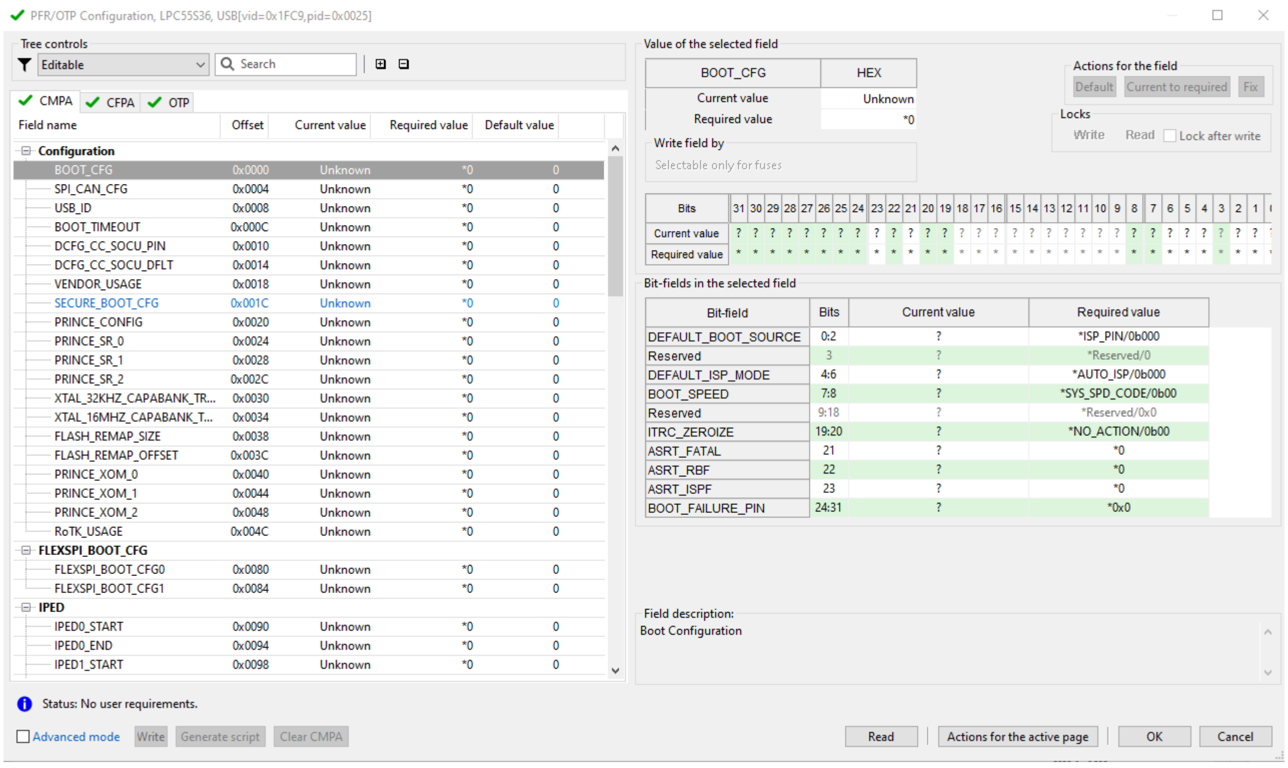
The configuration dialog allows configuring:

- One-time programmable fuses
- CFPA and CMPA pages from Protected Flash Region (PFR)
- ROMCFG/IFR page from Information Flash Region (IFR)
- Bootloader Configuration Area (BCA)
- Flash Configuration Field (FCF)

Use the configuration dialog to:

- Review the configuration prepared by SEC
- Read the current configuration from a connected processor
- Customize the configuration
- Lock the configuration (this feature is available for some fuses only)

In the OTP (One Time Programmable) Configuration, the configurable item is called a **fuse**. In the PFR (Protected Flash Region), BCA, FCF, and Information Flash Region (IFR) Configurations, the configurable item is called a **field**. The content of the dialog depends on the selected processor.



PFR/OTP Configuration, LPC55S36, USB[vid=0x1FC9,pid=0x0025]

Tree controls: Editable | Search

CMPA | CFPA | OTP

Field name	Offset	Current value	Required value	Default value
Configuration				
BOOT_CFG	0x0000	Unknown	*0	0
SPI_CAN_CFG	0x0004	Unknown	*0	0
USB_ID	0x0008	Unknown	*0	0
BOOT_TIMEOUT	0x000C	Unknown	*0	0
DCFG_CC_SOCU_PIN	0x0010	Unknown	*0	0
DCFG_CC_SOCU_DFLT	0x0014	Unknown	*0	0
VENDOR_USAGE	0x0018	Unknown	*0	0
SECURE_BOOT_CFG	0x001C	Unknown	*0	0
PRINCE_CONFIG	0x0020	Unknown	*0	0
PRINCE_SR_0	0x0024	Unknown	*0	0
PRINCE_SR_1	0x0028	Unknown	*0	0
PRINCE_SR_2	0x002C	Unknown	*0	0
XTAL_32KHZ_CAPABANK_TR...	0x0030	Unknown	*0	0
XTAL_16MHZ_CAPABANK_T...	0x0034	Unknown	*0	0
FLASH_REMAP_SIZE	0x0038	Unknown	*0	0
FLASH_REMAP_OFFSET	0x003C	Unknown	*0	0
PRINCE_XOM_0	0x0040	Unknown	*0	0
PRINCE_XOM_1	0x0044	Unknown	*0	0
PRINCE_XOM_2	0x0048	Unknown	*0	0
RoTK_USAGE	0x004C	Unknown	*0	0
FLEXSPI_BOOT_CFG				
FLEXSPI_BOOT_CFG0	0x0080	Unknown	*0	0
FLEXSPI_BOOT_CFG1	0x0084	Unknown	*0	0
IPED				
IPED0_START	0x0090	Unknown	*0	0
IPED0_END	0x0094	Unknown	*0	0
IPED1_START	0x0098	Unknown	*0	0

Status: No user requirements.

Value of the selected field

BOOT_CFG	HEX
Current value	Unknown
Required value	*0

Write field by: Selectable only for fuses

Actions for the field: Default | Current to required | Fix

Locks: Write | Read | Lock after write

Bits	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
Current value	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?
Required value	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*

Bit-fields in the selected field

Bit-field	Bits	Current value	Required value
DEFAULT_BOOT_SOURCE	0:2	?	*ISP_PIN/0b000
Reserved	3	?	*Reserved/0
DEFAULT_ISP_MODE	4:6	?	*AUTO_ISP/0b000
BOOT_SPEED	7:8	?	*SYS_SPD_CODE/0b00
Reserved	9:18	?	*Reserved/0x0
ITRC_ZEROIZE	19:20	?	*NO_ACTION/0b00
ASRT_FATAL	21	?	*0
ASRT_RBF	22	?	*0
ASRT_ISPF	23	?	*0
BOOT_FAILURE_PIN	24:31	?	*0x0

Field description:
Boot Configuration

Figure 35. OTP Configuration

Primarily all changes in the configuration dialog will be applied as part of the write script - except for Advanced mode.

The configuration window contains three main areas:

- Table of all fuses/fields (generally called items) on the left-hand side
- Detailed information about the selected fuse/field on the right-hand side
- Buttons bar in the bottom of the view

The width of the two main areas can be adjusted using the splitter.

5.2.7.1 Table of all items

PFR Configuration supports two pages: CFPA (Customer In-field Programmable Area) and CMPA (Customer Manufacturing/Factory Programmable Area). The IFR Configuration supports one-page called either ROMCFG (ROM Bootloader configurations) or IFR. Each page represents a separate list of fields organized in a tree. In OTP Configuration, all fuses are displayed in a single tree. The items in the tree are organized into logical groups. The tree of all items is displayed in form of a table, with the following columns (a column might not be displayed if the feature is not available for the processor):

Name : Human-readable name of the item. Some names may not be public and are available as a restricted data package. See the section about restricted data in Preferences.

Offset or shadow : Offset of the item address or offset of the shadow register address

: Index of the item (parameter for blhost to access the fuse)

R/W/O : Status of read/write/operational locks retrieved from the processor. For more information, see Locks.

Current value : The current value of the item read from the processor (hexadecimal)

Required value : The hexadecimal value required by SEC or by the user. Values preset by SEC are highlighted in blue and can be modified only in Advanced mode.

Default value : The default value of the item (hexadecimal) - after reset value.

Note: On some devices (KW45xx and K32W1xx), there are fuses with width bigger than 32-bit, for example, 256-bit keys, and 512-bit version counters. These long fuses are displayed in several rows in the table and have a common # index (fuse index).

5.2.7.2 Tree-filtering toolbar

It is possible to filter items displayed in the tree. There are predefined filter types in the dropdown list with a description in the tooltip. It is also possible to search for an element by name using the text box.

The toolbar also contains two buttons allowing to expand or collapse all groups.

5.2.7.3 Item editor

In the right part of the dialog, the following details are displayed for the selected item:

- Table with current and required item value as a hexadecimal number
- Current state of the read and write lock
- Selection, where the fuse will be written (see Burn fuse/write field).
- Table with current and required item value as a binary number
- Table with current and required item value as bit-fields value (only if the item is split to bit-fields)
- Description of the selected object (group of items, fuse, field, bit-field)

5.2.7.4 Buttons

The following buttons can be used for operations with the selected item:

Lock after write checkbox : Lock the item after write. (see Buttons)

Default : Remove user requirements for the item and apply a default value.

Current to required : Copy the current item value as a new requirement

Fix : Fix the displayed problems automatically. It is recommended to use the button for fixing the problems where only one value can be selected. If there is more than one option available as a fix, the function sets one of them.

The following buttons are available in the button bar:

Advanced mode : See Advanced mode.

Burn/Write : Write all the required values into the connected device. Only enabled in Advanced mode. **Note:** Use with caution. Changes are irreversible.

Generate script : Generate a script to write the required values. Expected to be used by advanced users only. By default, all items are written by the write script. Only enabled in Advanced mode.

Read : Read lock status and current values for all items from the connected target device. Values of the BCA and FCF pages can be read from the source image or connected device.

Actions for the active page : Show a context menu with actions that will be applied to a currently active page.

OK : Accept changes and return to **Write image**.

Cancel : Close the dialog without accepting changes.

The following items are available in the actions for the active page:

All current to required : Copy all current item value of the current page as new requirements. Applicable only if current values are known.

Default all : Remove all user requirements and apply a default value for all items.

Import … : Import a previously exported configuration in JSON file format.

Export … : Export the current configuration as a JSON file.

5.2.7.5 Read from connected device

Before you start the OTP/PFR/IFR/BCA/FCF Configuration tool, it is recommended to check in the **Connection** dialog if the board is connected to the host. If the target device requires a flashloader (RT10xx, RT116x/7x, and RT118x), it is recommended to click **Start flashloader** in the **Write image** view to ensure that the communication with the device can be established.

After the Configuration tool is open, it will offer to load current fuse values from the processor. This feature is optional and can be done also anytime later using the **Read** button. The Preferences dialog contains an option to configure the initial read operation.

The read operation consists of the following steps:

1. Read locks to find which fuses are readable
2. Read current fuse values
3. Detect individual write locks (if applicable for the processor, see Locks for details)

5.2.7.6 Required value

Items that must be written based on a selected configuration (for example contains preset value), are highlighted in blue. The remaining items, by default, either do not have any required value - displayed as * (asterisk) OR if default value is applied, they are displayed as *<value> (for example *0). The required value for these items can be specified:

- As a 32-bit hexadecimal number
- By bits. Click the value in the second row of the *Bits table* to toggle the bit, double-click to remove the requirement from the bit.
- Per bit-field (only if the register contains bit-fields)
  For some bit-fields, the value is selectable from a drop-down list; otherwise, it is specified as a decimal or hexadecimal number (with **0x** prefix).
- For items wider than 32 bits the value can be specified only as a HEX string. HEX string represents a sequence of bytes in the order, as they will be stored in the memory.

5.2.7.7 Burn fuse/write field

In the OTP/IFR configuration, it is possible to select the source where the fuse is burnt or the field is written:

- Write a script
- SB file that writes the application image
- Provisioning FW can be either the device HSM SB file or the SB/BIN file used during provisioning (it depends on the processor).

The available options depend on the configuration, so for example if shadow registers are used, no other option is enabled.

5.2.7.8 Locks

Locks are available in OTP Configuration to lock fuses. A lock block specifies an access restriction to the fuse - usually a read or write restriction. The locks must be programmed at the end of the development cycle when the rest of the configuration is already stable and tested and will not be changed. Locks are also used in IFR ROMCFG configuration. Here, the lock block specifies a write-access restriction to the ROMCFG block as each 16-byte block can be written only once. Similarly, the locks are also used in IFR configuration. Here, the lock specifies the write-access restriction to the IFR field as each 4-byte field can be written only once.

Two types of LOCKS are supported in SEC:

Global : Configured in a separate fuse usually called *LOCK*. The configuration is applied to several other fuses or shadow registers. READ, WRITE, or OPERATION locks exist, each type blocks the corresponding access to the fuse.

Individual write locks : For some processors, it is possible to apply a write-lock for a single fuse. Also, some fuses can be written only once and a write-lock must be applied. Both these features are presented as **Lock after write** checkbox, see description below. The **Lock after write** checkbox is also used to indicate a write restriction to the ROMCFG block or the IFR field.

The status of all locks is updated during the Read operation. The status is displayed in the fuses table, specifically, in these columns:

R : Display status of read lock for the fuse

W : Display status of write lock for the fuse (combined status of global and individual write lock)

O : Display status of operation lock for the fuse

The following icons are to represent the lock status:

Table Lock icons

Icon	Description
No icon	The fuse does not support a corresponding lock.
🔒	Lock status unknown
🔓	Fuse access unlocked
🔒	Fuse access is locked

Lock after write checkbox is dynamically enabled or disabled based on the selected fuse.

- Individual write-lock is not supported for selected fuse - checkbox is disabled and unselected
- The fuse must be locked after the first write - checkbox is disabled and selected
- Individual write-lock is optional - checkbox is enabled

For fuses configured to turn on individual write-lock, the following icon is displayed in the **Required value** column in the fuses table:

Table Individual write-lock

Icon	Description
🔏	Fuses configured to turn on individual write-lock

On RT116x, RT117x, and RT118x devices, the lock after write status cannot be detected, so the fuse might be displayed as unlocked even if it was already locked.

Lock validation

OTP Configuration reports a warning, in the case the write-lock for the fuse is on and the fuse value is not fully specified.

5.2.7.9 Calculated fields

Some registers or bit-fields contain a value that is calculated using the value of another bit field. For example:

- One item may contain the inverse value of another item
- One bit field contains the CRC of other bit-fields of the item

This feature is supported in **Configuration as validation**, the error is displayed in the case the calculated value does not match or the source value for the calculated item is not specified. See the following chapter for details about validations and problem resolutions.

5.2.7.10 Validation and problem resolution

The configuration provides validation of the required values and the problems are indicated by the icon in the window title, in the tree, and, if the item is selected, in the details section, in all editors.

In the BITS editor, the problem is displayed only for bits affected by the problem. It allows fixing the problem easily by clicking the affecting bit value (for example, inverting the required value of the bit).

For all errors, you can use the **Quick Fix** button. This button fixes all errors within the selected item. Make sure to review the changes made by the quick fix to ensure that the newly applied value matches your expectations.

Warnings may also be displayed for the current value read from the processor to inform the user when the value is invalid

Note: Specific OTP/PFR fields are used for the transition of the device through its life cycle, granting conditional or locking down access to various debug resources in the device once programmed. As the best practice, it is recommended to program these registers once the secure boot has been verified as functional. Incorrect values in these fields may render the platform nonfunctional or no longer accessible for recovery. On some platforms, these registers include special “valid” semantics - for example, on the LPC55Sxx processors, the `DCFG_CC_SOCU_PIN` and `DCFG_CC_SOCU_NS` configuration fields include bits that must be programmed as the inverse of all the other fields in the register for the configuration to be valid. The Fix Problems facility enforces this constraint only if the register contains a non-zero value - that is, an explicit user configuration of the register has been detected.

5.2.7.11 Advanced mode

By default, it is recommended to apply the modified configuration into a workspace settings file and the Write script, so it is applied together with the bootable image. However, sometimes it might be necessary to burn a single fuse value, in which case you can use the Advanced mode. The Advanced mode is designed for standalone usage of the Configuration tool and allows you to:

- Write a required value directly to the connected processor (see Write/Burn)
- Generate the script to write the required values
- Modify all required values, even the ones preset by SEC-PFR Configuration
- Clear CMPA page; apply default values to the whole CMPA page

Additionally, the reserved items values are read from the connected processor in this mode (most likely useful to NXP engineers only).

The Advanced configuration is not expected to be applied to the write script, so the **OK** button is disabled. The **Export** button can be used to store the created configuration into an external file.

Note: Advanced mode is not needed for normal workflow supported by the SEC Tool, it must be used only for the use cases not supported by the tool.

5.2.7.12 Write/Burn

The Write/Burn operation burns all required values into the connected device including all locks. The burn operation consists of the following steps:

1. Read current values from the processor
2. Update validation problems
3. Generate write/burn script
4. Execute the write/burn script

Bear in mind that the burn operation is irreversible. It is recommended to:

- Double-check all values being burned
- Double-check all items being locked
- Double-check all problems reported by the configuration
- Generate write/burn script and review the content

There is a difference between Burn and Generate Script:

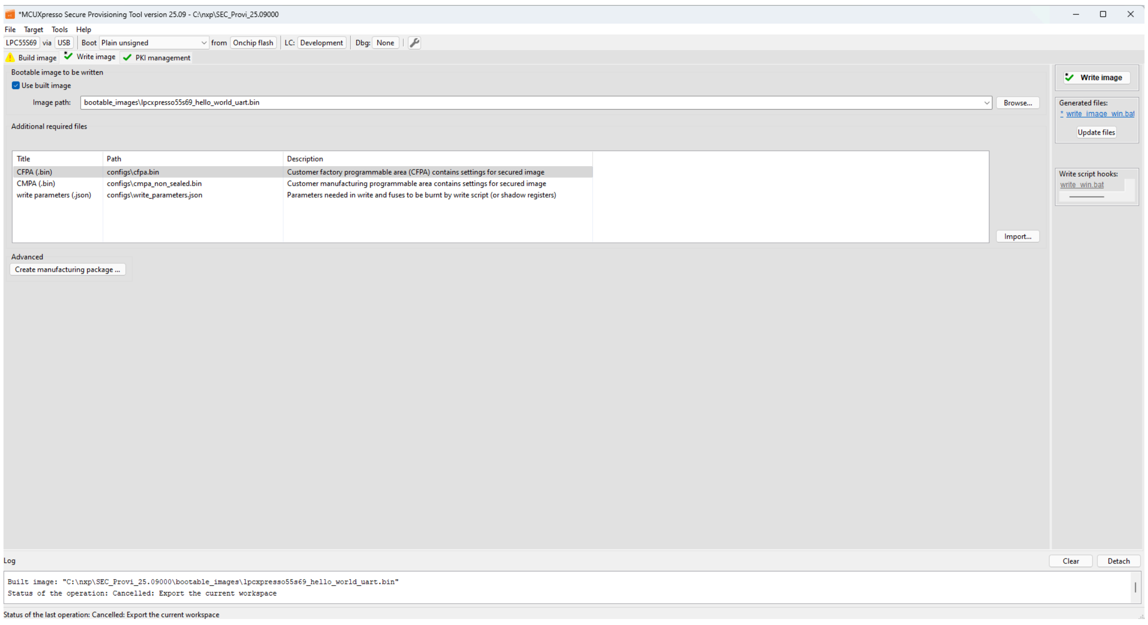
- The Burn operation is optimized for the selected processor. The fuse will not be burned if the value matches or the fuse is locked. For CFPA and CMPA the whole page is always written.
  Warning: The ROMCFG block can be written only once.
- Generate Script is expected to be used on an empty processor. It contains the configuration of all fuses and it might fail if any fuse is already burnt.

5.2.7.13 PFR/IFR/BCA/FCF and OTP differences

- Items in OTP Configuration are called “fuses” while items in other configurations are called “fields”.
- Fuses in OTP Configuration are burned item by item, so you can specify a single requirement only. The fields in IFR ROMCFG Configuration are written by 16-byte blocks, so completed 16-byte requirements must be specified. PFR always updates the whole page, so if no requirement is specified, the default value is used.
- Selection `burn fuse by` is supported only for fuses.
- Locks selections are supported for fuses, IFR fields, and the IFR ROMCFG blocks.
- CFPA, CMPA, BCA, and FCF pages can be written multiple times whereas the ROMCFG block or IFR field can be written only once.

5.3 Write image

Use the **Write image** view to write an image into the boot memory, burn platform fuses and configure the selected life cycle to achieve a secure boot. The **write image** tab may have a dirty flag on the icon, which means there are changes that were not included in the last build operation.


*MCUXpresso Secure Provisioning Tool version 25.09 - C:\nxp\SEC_Provi_25.09000
File
Target
Tools
Help
LPC55S69 via USB
Boot Plain unsigned from Onchip flash
LC: Development
Dbg: None
Build image
Write image
PKI management
Browse...
Import...
Create manufacturing package ...
Write image
Update files
Clear
Detach

Bootable image to be written

Use built image: ✓

Image path: bootable_images\lpcxpresso55s69_hello_world_uart.bin

Additional required files

Title	Path	Description
CFPA (.bin)	configs\cfpa.bin	Customer factory programmable area (CFPA) contains settings for secured image
CMPA (.bin)	configs\cmpa_non_sealed.bin	Customer manufacturing programmable area contains settings for secured image
write parameters (.json)	configs\write_parameters.json	Parameters needed in write and fuses to be burnt by write script (or shadow registers)

Advanced

Generated files: * write_image_win.bat

Write script hooks: write_win.bat

Log

```
Built image: "C:\nxp\SEC_Provi_25.09000\bootable_images\lpcxpresso55s69_hello_world_uart.bin"
Status of the operation: Cancelled: Export the current workspace
```

Status of the last operation: Cancelled: Export the current workspace

Figure 36. Write image (LPC55Sxx)

5.3.1 Controls on the write image view

Use built image : If checked, the output of the Build image operation will be used for the write.

Bootable image : Path to the image that will be written into the target device. The file extension is specific for processor and boot type, it is either BIN for bootable images or SB for secure-binary capsules. The binary image must be in “nopadding” form without the FCB block, as the FCB block is written in a separate step.

Additional input files : Display required input files for the **Write image** operation. The contents depends on the processor, boot type, and other build options. By default, the contents are output files of the **Build image** operation. You can manually replace each file with a custom file using the **Import** button.

Start flashloader : Allows you to initialize and start the flashloader on the connected processor. If security is enabled in the chip, the signed flashloader is created automatically. Useful if you want to use blhost from the command line.

Test life cycle : Opens dialog that can set a temporal advanced life-cycle state. This allows it to test processor behavior in a selected life-cycle state without burning the fuses. The board must be connected by a debugger probe.

Disable flash security : Opens a dialog where the user can specify a backdoor key to disable flash security. The backdoor key can be obtained from the FCF configuration, a bootable image, or entered manually. As a final step in the flash disabling process, it is possible to execute a mass erase to clear the entire flash memory.

Write image panel : There is the Write image panel with the **Write image** button and the file name of the write script, on the right side of the window.

Generated files : The **Generated files** panel has the same functionality as Generated files.

Write script hooks : The panel with the hook script that is called during the write script execution. The gray label means that the file does not exist in the workspace yet. After clicking, a new file is created with the contents from the example and opened for modification. If the file exists, the label is blue and after clicking the file is opened. For details, see Script hooks workflow.

Before clicking the **Write image** button, ensure that the board is connected and configured to the ISP mode. If any irreversible operation is done by the write script, a confirmation dialog with details appears.

5.3.2 Manufacturing package

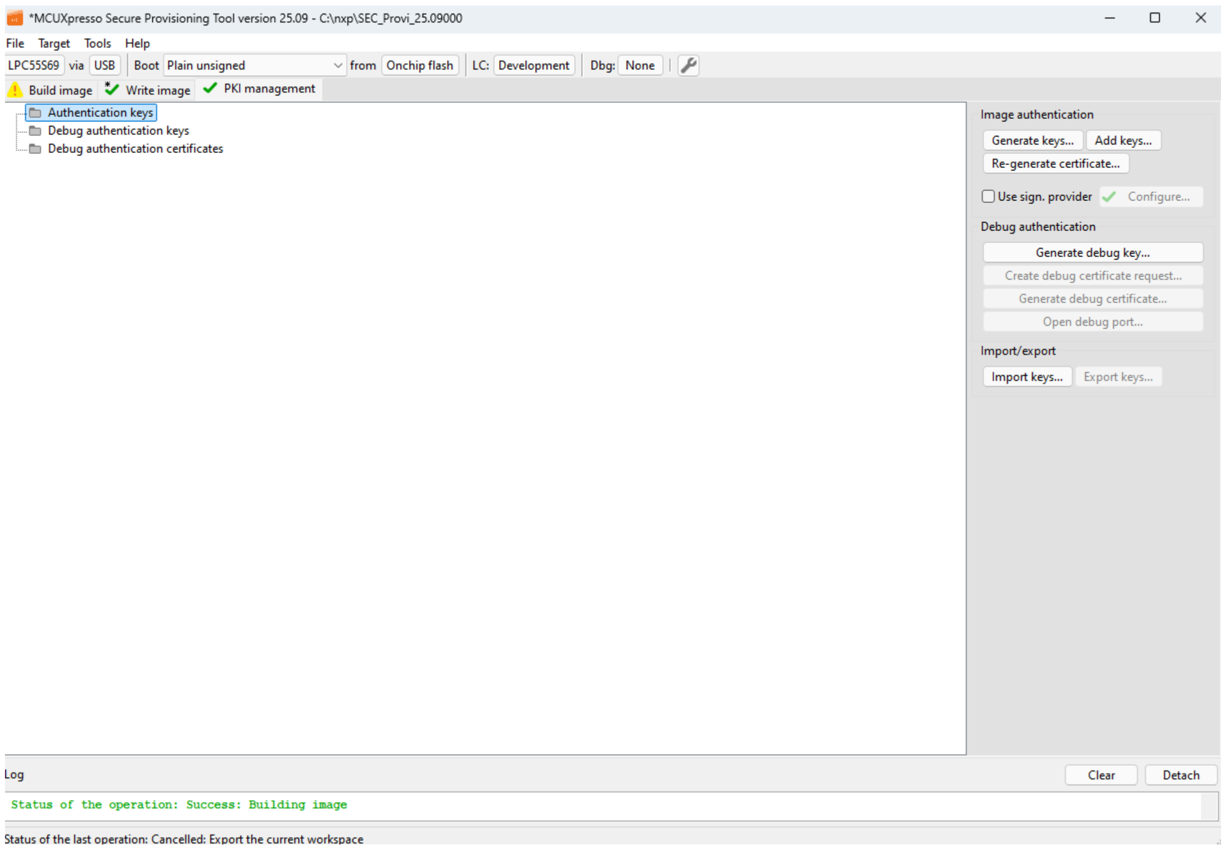
The manufacturing package is a ZIP file that contains the write script and all other files needed for the write operation and it is designed to send the files into the factory for production. The manufacturing package can be created using the button **Create manufacturing package …** on the **Write image** page. The dialog is described in Manufacturing workflow.

In the factory, the manufacturing package can be imported using **main menu > File > Import Manufacturing Package…**. For more information, see Manufacturing operations.

5.4 PKI management

The **PKI management** view displays the list of keys and certificates used to validate the authenticity of the image. Generated keys can be exported for later use. PKI management allows generating the following:

- keys for image authentication
- keys for trust provisioning
- key pair for debug authentication



LPC55S69 via USB | Boot Plain unsigned from Onchip flash | LC: Development | Dbg: None

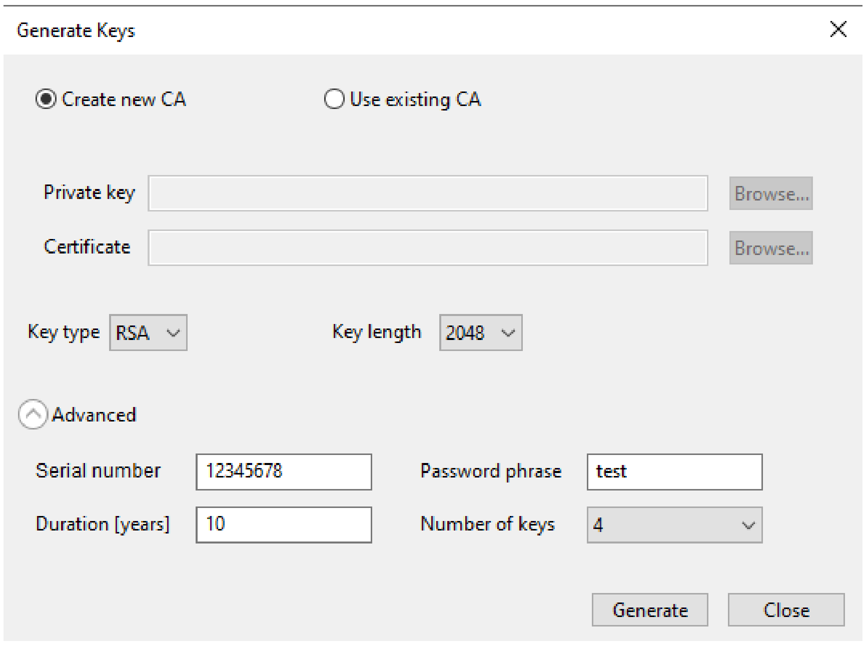
```
Status of the operation: Success: Building image
```

Status of the last operation: Cancelled: Export the current workspace

Figure 37. PKI management (LPC55Sxx)

5.4.1 Generate keys

Authenticated images rely on a Public Key Infrastructure (PKI) set of certificates. SEC includes a graphical interface that simplifies the generation of a PKI-compatible with selected processor.



[● Create new CA] ○ Use existing CA

Private key

Certificate

Key type: RSA

Key length: 2048

Advanced

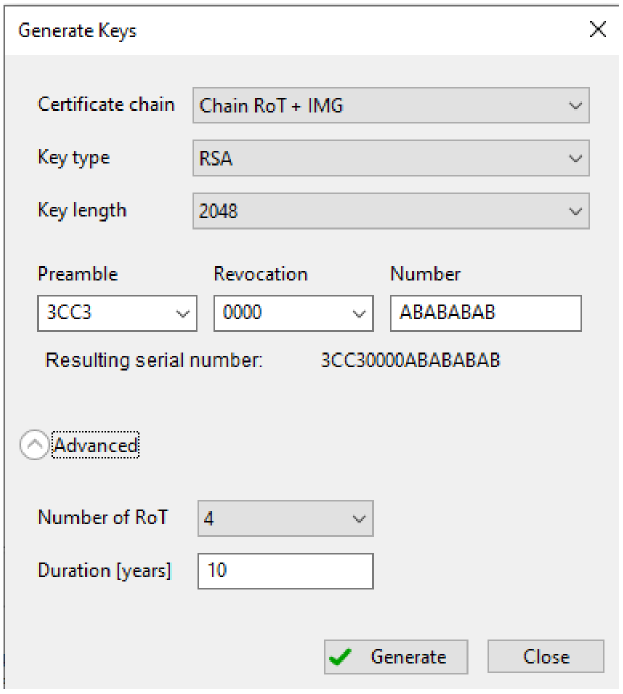
Serial number: 12345678

Password phrase: test

Duration [years]: 10

Number of keys: 4

Figure 38. Generate keys (RT1xxx)



Certificate chain: Chain RoT + IMG

Key type: RSA

Key length: 2048

Preamble: 3CC3

Revocation: 0000

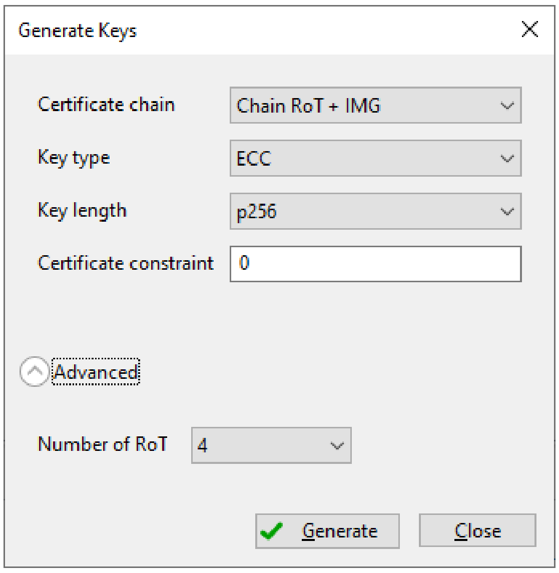
Number: ABABABAB

Resulting serial number: 3CC30000ABABABAB

Number of RoT: 4

Duration [years]: 10

Figure 39. Generate RSA keys



Certificate chain: Chain RoT + IMG
Key type: ECC
Key length: p256
Certificate constraint: 0
Number of RoT: 4

Figure 40. Generate ECC keys

Create new CA : This option allows generating keys including Certificate

Use existing CA : Enable the use of user-specified CAs. **Certificate** and **Private Key** must be in PEM format.

Private Key : Path to the private key file

Certificate : Path to the certificate file

Key type : Generated key type.

Key length : Generated key length in bits.

Serial number : The value is used for key revocation. The serial number for RSA keys on LPC55Sxx and RT5xx/6xx consists of three parts: preamble, revocation, and number. The revocation part is validated against the revocation field in OTP/CFPA.

Password phrase : Secure generated CA with the specified parameter.

Duration [years] : Set certificate validity to the specified duration in years. For supported devices, it is necessary for signing purposes only, as the duration is not directly verified in hardware.

Number of keys : Set to the number of keys to be generated. Most processors support up to 4 keys, with a recommended default of 4.

Certificate chain : The depth of the keychain.

Preamble : Prefix of the serial number, mandatory fixed value

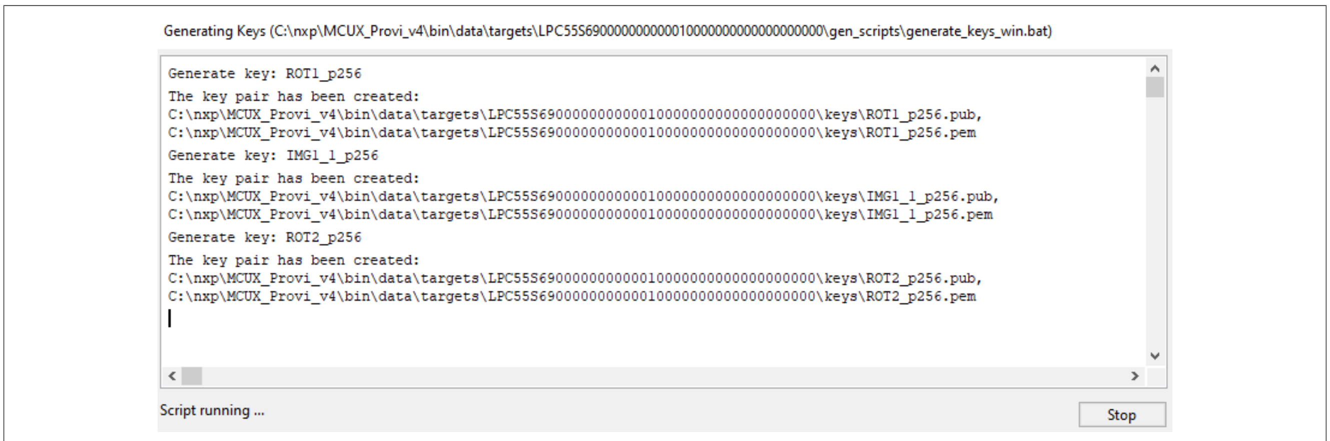
Revocation : Middle part of the serial number, 16-bit revocation ID - this value should match the IMG_REVOKE field in OTP/PFR (CFPA) on the device.

Number : Suffix of the serial number bytes used to uniquely identify the certificate/key.

Certificate constraint : Used for revocation of the image signing key, validated against the revocation field in OTP/PFR.

Refer to the OpenSSL documentation for additional details about the Password, Serial, and Duration options.

Once all parameters have been specified, click the **Generate** button. The key generation script output will be displayed in the progress window.


Generating Keys (C:\nxp\MCUX_Provi_v4\bin\data\targets\LPC55S690000000000010000000000000000000\gen_scripts\generate_keys_win.bat)

```
Generate key: ROT1_p256
The key pair has been created:
C:\nxp\MCUX_Provi_v4\bin\data\targets\LPC55S690000000000010000000000000000000\keys\ROT1_p256.pub,
C:\nxp\MCUX_Provi_v4\bin\data\targets\LPC55S690000000000010000000000000000000\keys\ROT1_p256.pem
Generate key: IMG1_1_p256
The key pair has been created:
C:\nxp\MCUX_Provi_v4\bin\data\targets\LPC55S690000000000010000000000000000000\keys\IMG1_1_p256.pub,
C:\nxp\MCUX_Provi_v4\bin\data\targets\LPC55S690000000000010000000000000000000\keys\IMG1_1_p256.pem
Generate key: ROT2_p256
The key pair has been created:
C:\nxp\MCUX_Provi_v4\bin\data\targets\LPC55S690000000000010000000000000000000\keys\ROT2_p256.pub,
C:\nxp\MCUX_Provi_v4\bin\data\targets\LPC55S690000000000010000000000000000000\keys\ROT2_p256.pem
```

Script running ...

Figure 41. Generate Keys - Progress

5.4.2 Add keys

Once keys have been generated in the **Generate keys** dialog, you can add additional image keys using the **Add keys** dialog.

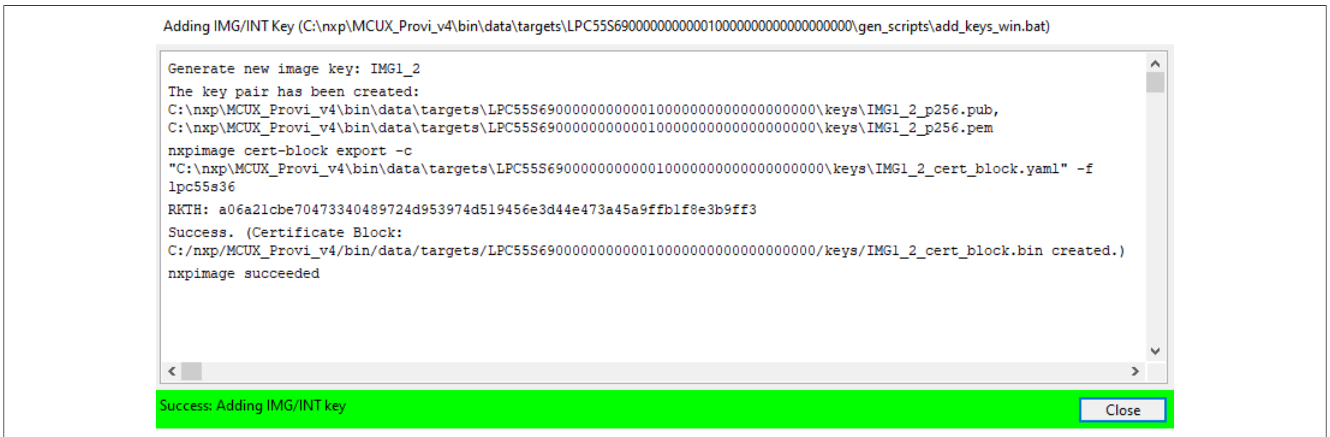
IMG key path : Path of the IMG (ISK) key to be generated and added.

CSF key path : Path of the CSF key to be generated and added.

Key : Key type to be generated, either **Image** for image signing or **Intermediate** for creating a chain of keys

The other items are described in Generate keys.

Once you have specified your preferences, click **Ok** to add the keys. The output will be displayed in the progress window.


Adding IMG/INT Key (C:\nxp\MCUX_Provi_v4\bin\data\targets\LPC55S690000000000010000000000000000000\gen_scripts\add_keys_win.bat)

```
Generate new image key: IMG1_2
The key pair has been created:
C:\nxp\MCUX_Provi_v4\bin\data\targets\LPC55S690000000000010000000000000000000\keys\IMG1_2_p256.pub,
C:\nxp\MCUX_Provi_v4\bin\data\targets\LPC55S690000000000010000000000000000000\keys\IMG1_2_p256.pem
nxpimage cert-block export -c
"C:\nxp\MCUX_Provi_v4\bin\data\targets\LPC55S690000000000010000000000000000000\keys\IMG1_2_cert_block.yaml" -f
lpc55s36
RKTH: a06a21cbe70473340489724d953974d519456e3d44e473a45a9ffb1f8e3b9ff3
Success. (Certificate Block:
C:/nxp/MCUX_Provi_v4/bin/data/targets/LPC55S690000000000010000000000000000000/keys/IMG1_2_cert_block.bin created.)
nxpimage succeeded
```

Success: Adding IMG/INT key

Figure 42. Add Keys progress

5.4.3 Re-generate certificate

The SEC Tool allows re-generating a ROT certificate, for certificate revocation (available only for RSA ROT certificates). Select the certificate and click the **Re-generate certificate…** button. In the dialog, review the parameters, update the serial number and confirm re-generation. The current certificate will be backed up into the backup folder and then re-written.

The parameters for re-generation are the same as the parameters for generation; the description can be found in the previous chapters.

5.4.4 Import/Export keys

You can export generated keys for later use with the help of the Export function. To export keys:

1. Select **Export keys** in the **PKI management** view.
2. In the dialog, navigate to the location you want to export the keys to, and select **Select folder**.

Note: Export keys also export workspace configuration, including symmetric keys, for example, SBKEK for LPC or BEE user keys for RT10xx devices.

You can later import the keys into a new workspace using the Import function. The operation makes a backup of current keys and settings inside the current workspace. The symmetric keys, for example, SBKEK for LPC55Sxx/RT5xx/RT6xx, BEE user keys for RT10xx, OTFAD keys data for RT116x/RT117x/RT118x/RT5xx/RT6xx devices, and IEE keys data for RT116x/RT117x/RT118x are restored from the imported folder.

To import keys:

1. Select **Import keys** in the **PKI management** view.
2. In the dialog, navigate to the folder containing the **keys** and **crts** subdirectories with the keys info you want to import and select **Select folder**.

To import external keys that were not exported from the SEC Tool:

1. Generate new dummy keys and export them into a new folder; ensure that the number of keys matches the keys being imported.
2. Overwrite the keys in the exported folder with the files being imported (rename keys being imported so they match the naming convention used in the SEC Tool).
3. Import the keys back from the folder.

5.4.5 Debug authentication

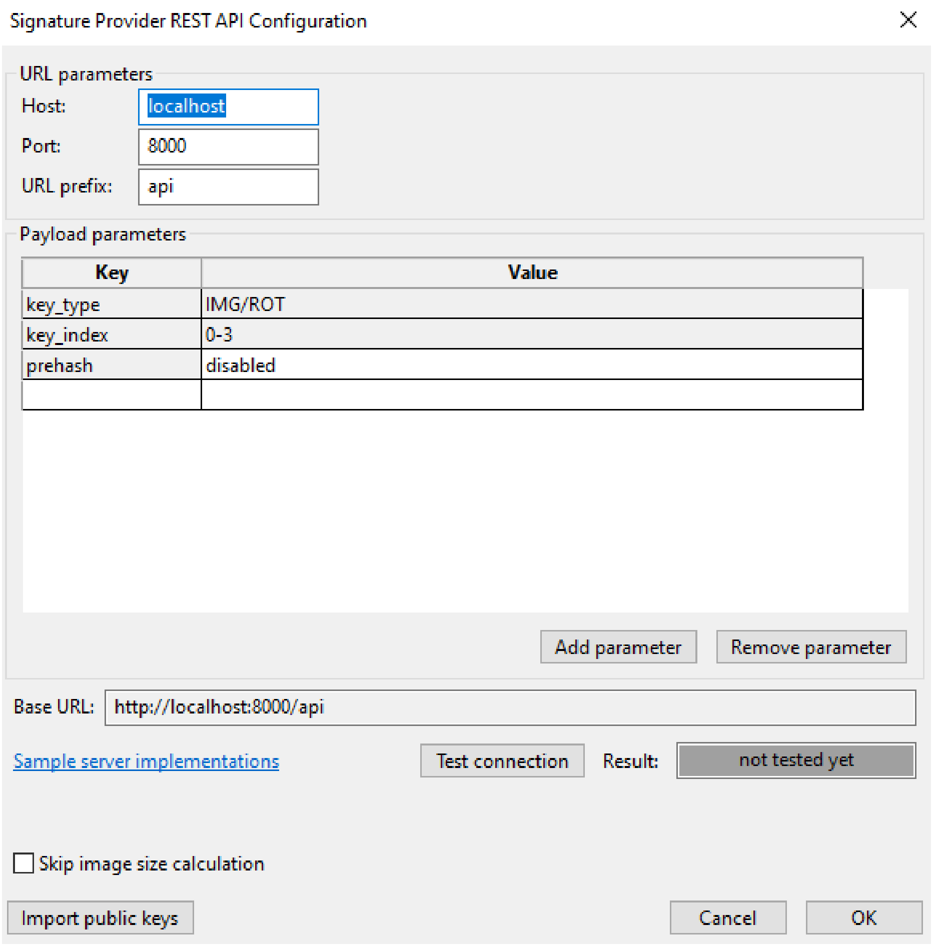
The Debug Authentication (DA) buttons are displayed in PKI management only for processors that support DA. The **Generate debug key** button is always enabled, **Create debug certificate request** and **Open debug port** are enabled only if the DA key exists. **Generate debug certificate** is enabled in a workspace with authentication keys. The **Generate debug key** button opens a dialog box for creating a debug authentication key pair. **Generate debug certificate request** allows to create a request that will be sent to the OEM, from which a debug certificate can be generated. The **Generate debug certificate** buttons display a certificate where the OEM selects the keys and sets the rights that will be granted to the certificate holder. The **Open Debug Port** dialog prepares the connected device for debugging by using the DAC.

Note: For more information, see Debug authentication workflow.

5.4.6 Signature provider

Signature provider allows using a custom provider for the authentication instead of keys stored on a local machine. Signature provider requires a custom implementation of an HTTP server with a simple API providing the authentication.

Once the **Use sign. provider** checkbox is selected, it is possible to open the configuration dialog. When enabled, the present keys from the workspace are moved to the back-up folder. The signature provider server is used as the source of public keys and provides signing. Configuration set in **Signature Provider Configuration** dialog is used in configuration files for build, key certificate regeneration, and DAC generation(`cert_block, mbi_config, sb_config, debug_credentials_config`).



Host: localhost

Port: 8000

URL prefix: api

Key	Value
key_type	IMG/ROT
key_index	0-3
prehash	disabled

Base URL: http://localhost:8000/api

Result: not tested yet

Figure 43. Signature provider configuration

5.4.6.1 URL parameters

Required parameters from URL of the server:

- **Host** host name or IP address of the server, accepted symbols as specified in RFC 3986. It is recommended to use localhost, as the HTTP communication is not secure. For communication to another computer, it is recommended to implement a proxy server forwarding the communication via a secure channel (HTTPS).
- **Port** port number of the service, integer number.
- **URL prefix** REST API prefix, use an empty string if there is not any.

5.4.6.2 Payload parameters

Payload parameters are passed as a JSON payload with each request sent to the server. Each parameter is identified by a unique key and contains a text value. The following keys are reserved for the tool:

- **type** - used by SPSDK to identify signature provider type.
- **host, port, url_prefix** - used to specify server connection.
- **data** - used as key for data payload of request.
- **key_type** - used by SEC Tool to specify whether image or root key should be used; supported values are **IMG** or **ROT**.

- **key_index** - used by the SEC Tool to identify which private key should be used, the value is a decimal number in the range 0-3 set by selecting a key from build tab ROT1 -> 0, ROT2 -> 1, IMG1_1 -> 0, IMG2_1 -> 1 and so on.
- **prehash** - allows selecting whether to send complete data for signing or hash only. The hash algorithm can be selected from the drop-down menu; however, it is recommended to use `default`.

5.4.6.3 Buttons and Base URL

- The **Remove parameter** button removes a parameter from the table, remove the selected parameter, if no parameter is selected, remove the last one.
- The **Add parameter** button adds an empty line below the selected param or at the end of the table.
- **Base URL** displays a URL created from required parameters.
- The **Test connection** button sends a request for signature length to test whether the server is up and running.
- The **Import public keys** button imports public keys from the server, request is sent to endpoint 'public_keys_certs', the server response format described in the schema `signature\_provider\_key\_tree\_schema_##\_##.json`.
- The **Sample server implementations** button opens the folder with a sample of the Signature provider server implementation.

5.4.6.4 Signature provider server API

Signature provider API must implement the following three endpoints, and optionally one additional endpoint:

`sign`

sign handles signing of the given data block; the request sends the data to be signed with a private key specified by optional parameters. Example of the signature request for MCX947 with ROT1 or ROT1-IMG1_1 selected as signing key: 'http://localhost:8000/server/sign' json payload:

Certificate signing with ROT1 key:

```
{
  "data": "hex string of certificate block to be signed",
  "key_type": "ROT"
  "key_index": 0
  "prehashed": "none if not prehashed, hash algorithm name if data is prehashed"
}
```

Image signing with IMG1_1 key:

```
{
  "data": " hex string of image block to be signed",
  "key_type": "IMG"
  "key_index": 0
  "prehashed": "none if not prehashed, hash algorithm name if data is prehashed"
}
```

`signature_length`

signature_length returns the signature length in bytes.

`verify_public_key`

verify_public_key verifies that the used public key forms a valid key pair with the private key that is on the server. The public key is sent as a data payload, the RSA key is sent in NXP proprietary format and the ECC key is sent in DER format. To identify the private key that should be matched with this key, use optional parameters. In the case of matching keys, return "true".

This API is designed to detect a problem if the server is using a key different from the client one. For environments where such validation is not needed, the implementation can return simply “true”.

`public_keys_certs`

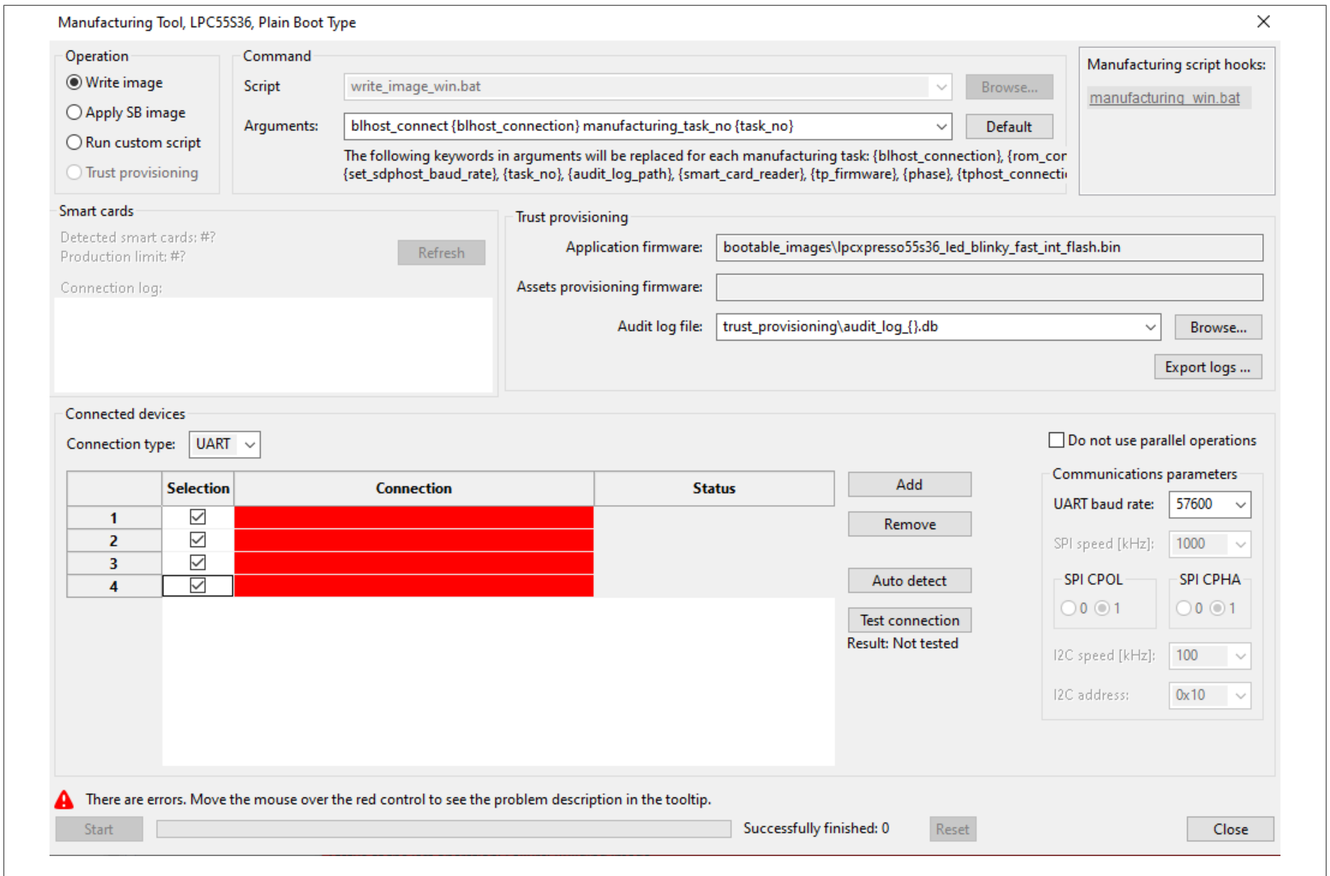
public_keys_certs is an optional endpoint. If implemented, it allows importing public keys from the Signature provider dialog into the SEC Tool. The SEC Tool expects the response from this endpoint to be a tree of Authentication keys, ROT keys at the top level, and IMG keys as the leaves. The structure of the response is described in <install_folder>/bin/_internal/schema/signature_provider_key_tree_schema_##_##.json. In general, any tree that can be generated from the Generate keys dialog can be imported, with the limitation of one child key for the root key. RSA trees do not need to have the same key length for each subkey.

5.4.6.5 Server examples

Server examples are part of the SEC Tool distribution. There are Python script examples with the server implementation. These examples are described in Signature provider workflow.

5.5 Manufacturing Tool

Use **Manufacturing Tool** to provision several devices connected to the host at the same time. **Manufacturing Tool** can be accessed from **Tools** in the **menu bar**. The devices can be connected to the host using USB or UART or SPI or I2C. The user interface is made of these areas: **Operation**, **Command**, **Connected devices**, **Communication parameters**, and the button bar. If trust provisioning is supported for the processor, there is also a **Trust provisioning** area.



Manufacturing Tool, LPC55S36, Plain Boot Type

Operation: [✓ Write image] Apply SB image / Run custom script / Trust provisioning

Command
Script: write_image_win.bat
Arguments: blhost_connect {blhost_connection} manufacturing_task_no {task_no}
The following keywords in arguments will be replaced for each manufacturing task: {blhost_connection}, {rom_cor
{set_sdphost_baud_rate}, {task_no}, {audit_log_path}, {smart_card_reader}, {tp_firmware}, {phase}, {tphost_connecti

Manufacturing script hooks: manufacturing_win.bat

Smart cards
Detected smart cards: #?
Production limit: #?
Connection log:

Trust provisioning
Application firmware: bootable_images\lpcxpresso55s36_led_blinky_fast_int_flash.bin
Assets provisioning firmware:
Audit log file: trust_provisioning\audit_log_{}.db

Connected devices
Connection type: UART

	Selection	Connection	Status
1	✓	🟥	
2	✓	🟥	
3	✓	🟥	
4	✓	🟥	

Result: Not tested

☐ Do not use parallel operations

Communications parameters
UART baud rate: 57600
SPI speed [kHz]: 1000
SPI CPOL: 0 [✓ 1]
SPI CPHA: 0 [✓ 1]
I2C speed [kHz]: 100
I2C address: 0x10

There are errors. Move the mouse over the red control to see the problem description in the tooltip.

Successfully finished: 0

Figure 44. Manufacturing Tool

Operation : The Operation area contains radio buttons representing different operations to choose from.

- **Write image** : Performs the same operation as **Write image**. Before use, the image must be built in the **Build image** view, and the **Write image** view must not show any errors.
- **Apply SB image** : Uses an SB (Secure Binary) capsule to update the existing image of the processor. For most of the processors, the SB file is created during the build secure image operation and is located in the *bootable_images* subfolder of the workspace by default. For RT10xx/RT116x/RT117x/RT118x processors, the SB file must be created manually, as it is currently not supported by SEC.
- **Run custom script** : Uses a custom script to configure provisioning. It is assumed that the script is a modified SEC write script accepting SEC parameters (especially connection parameters). Exit status 0 is considered a success.

Command : The Command area contains parameters needed for the operation selected in the Operation area. The parameters vary based on the selected operation.

- **Script (Write image, Run custom script, Trust provisioning)** : Path to the write or custom script. To locate the custom script, use the **Browse** button.
- **SB file (Apply SB image)** : Path to the SB capsule. To locate the file, use the **Browse** button.
- **Arguments** : List of arguments used during the operation. Use the **Default** button to revert any changes. The keywords enclosed in curly braces in arguments will be replaced for each manufacturing task. You can find the complete list of keywords with descriptions in the tooltip.

Manufacturing script hooks : The panel with the hook script that is called at the start and end of the manufacturing script execution. The gray label means that the file does not exist in the workspace yet. After clicking, a new file is created with the contents from the example and opened for modification. If the file exists, the label is blue and after clicking the file is opened. For details, see Script hooks workflow.

Trust provisioning : Configuration of the trust provisioning operation.

- **Application and Assets provisioning firmwares** : These file paths are listed for information only.
- **EdgeLock 2GO API key** : Access key for EdgeLock 2GO server. It can be specified here or in environment variable `SEC_EL2GO_API_KEY`.
- **EdgeLock 2GO server: Test connection** : Allows testing the connection with the EdgeLock 2GO server.
- **EdgeLock 2GO product batch DB** : Path to EdgeLock 2GO secure objects database used for the per product (offline) flow

Connection type : Allows selecting a communication interface with the processors.

#phases : A number of phases being executed for one device. The USB device identification passed to write the script might change after reset (see USB path). If the write image script supports the USB connection, the script is executed in **phases**, each phase is finished by reset and the USB identification is updated before executing the next phase. This parameter allows overriding the number of phases being executed and a forced custom number (in case the write script was modified manually). Value `disabled` means that the phases are not used. It is recommended to use the `default` value. If the `default` value is selected, the current number of phases is displayed in the tooltip.

Do not use parallel operations : the check box allows disabling parallel operations in the manufacturing. This can be used if any problem with parallel operation is detected.

Connected devices : The connected devices area contains an interactive table displaying all connected devices. The tool can detect and program devices connected using USB only if they are in ISP mode.

Columns:

- **Selection** : allows users to enable/device; each device is configured in the table. If the device is de-selected, the next autodetection keeps the device de-selected.
- **Connection** : allows users to enter manually the device name or path.
- **Status** : Displays the status of the connected device. After the operation was executed, the log file can be opened by clicking the entry in the **Status** column

Buttons to modify Connected devices:

- **Select all** : A button to select all devices in the table below.
- **Deselect all** : A button to deselect all devices in the table below.
- **Autodetect** : A button to detect automatically all devices connected to the host for a selected connection type. It is the recommended usage. For more information about the USB path, see USB path.
- **Add** : A button to manually add a device (add an empty row to the table).
- **Remove** : A button to remove the selected device from the table (remove the selected row).
- **Test connection** : A button to test (ping) the selected devices. This feature might be useful for UART, SPI, and I2C devices where the detection of the communication device (COM port or USB path) does not imply that the connection with the processor is established. Therefore, a test connection is recommended before starting the manufacturing operation.
- **Export logs** : Allows exporting manufacturing logs as a ZIP file. Besides exporting all logs, it is possible to export logs for a specific day only.

Load KeyStore (Apply SB image for RT5xx/6xx) : Load KeyStore from external flash before uploading the SB file.

Communication parameters : Configuration of the connection with the device. The *Baud rate* option allows selecting a value from the drop-down list or specifying it manually. The detailed description of the parameters can be found in the Connection chapter.

Crystal freq. [kHz] : Frequency of the external crystal used as a clock source for the processor. This is used only for `lpcprog` tool.

Button bar : The Button bar contains action buttons and displays any warnings and alerts.

- **Start** : Starts the selected and configured operation. You can observe the progress of the operation in the adjacent progress bar.
- **Successfully finished** : Label with the number of successfully finished operations. This number is incremented automatically and stored in the settings file in the workspace.
- **Reset** : The button allows resetting the “Successfully finished” counter.
- **Close** : Closes the Manufacturing Tool without running the operation.

5.5.1 USB path

The Manufacturing Tool supports the use of Serial, USB, SPI, or I2C connection. In most cases, the Auto-detect function will be sufficient for detecting the connected devices. The USB path is used to identify the USB device for USB, SPI, or I2C communication, so the devices with the same USB VID+PID can be identified uniquely. It differs depending on the operating system. The description of the USB path format can be found in the SPSDK documentation.

The Manufacturing Tool supports automatic USB path update that may happen during the manufacturing operation because of the following reasons:

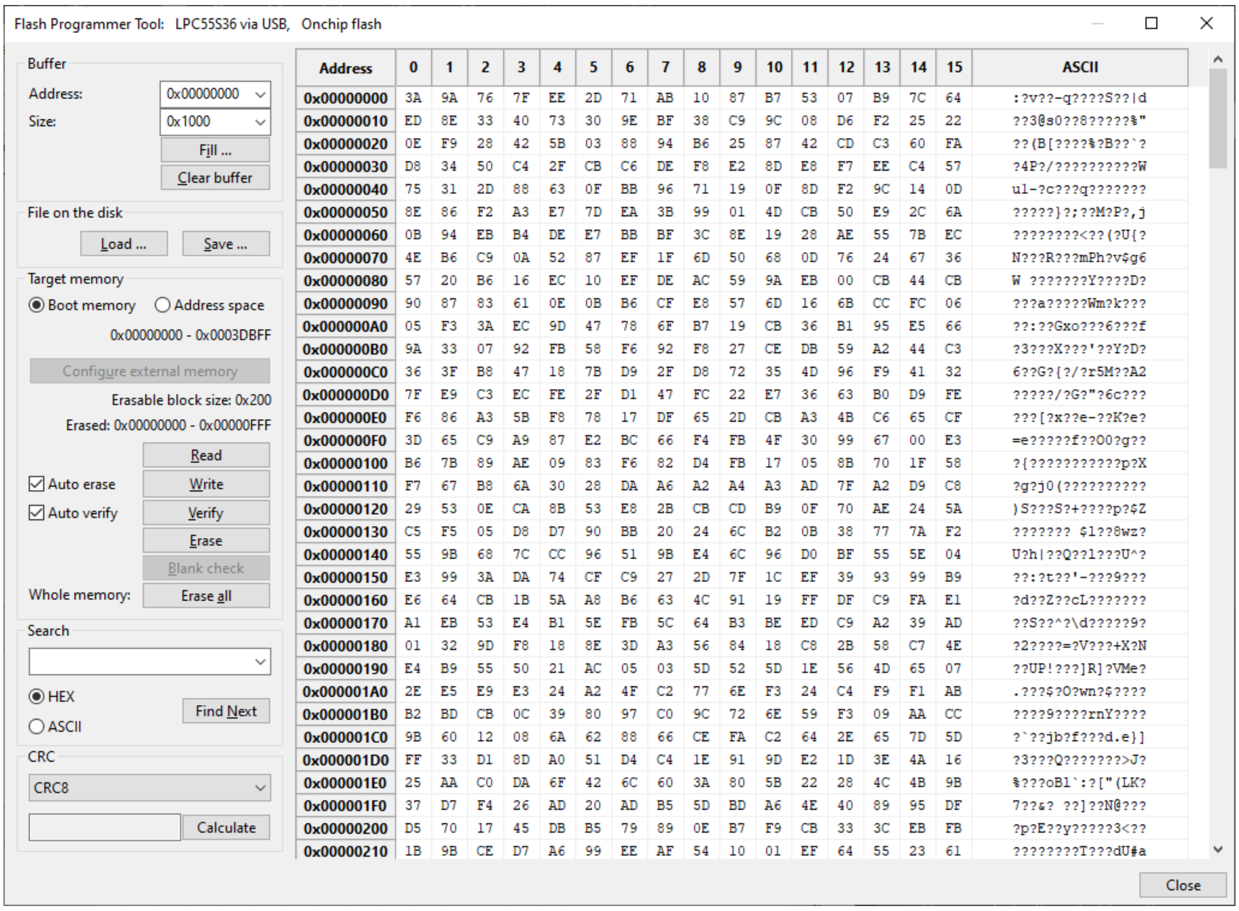
1. On some operating systems, the USB path changes after each device reset.
2. RT10xx processors in ISP mode have different VID and PID compared to the VID and PID of the flash loader application.

After the update, the Manufacturing Tool automatically executes the second part of the script that finishes the required operation.

Note: RT10xx devices in ISP mode have different VID and PID compared to the VID and PID of the flash loader application, which is uploaded to the target to program the flash. Auto-detect searches for devices with PID&VID when the target is reset in the ISP mode. If the flashloader is active on the device, reset it into ROM bootloader mode. The manufacturing process is slightly different: as a first step, the flashloader is uploaded to all boards and its USB path is retrieved. After this step, the rest of the process is executed in parallel. After manufacturing is finished, the device flashloader is still active, so the device will not be detected until you have reset the processor.

5.6 Flash Programmer

Flash Programmer is designed to read/write from the currently selected flash memory and supports all flash types including internal flash, external NOR and NAND flash, SD card, and eMMC. Flash Programmer can be accessed from **Tools** in the **menu bar**. The processor can be connected to the host using USB, UART, SPI, or I2C and must be in ISP mode (as the tool internally uses the blhost or lpcprog protocol). Flash Programmer can be used to prepare data for writing, or just to display or modify saved memory blocks even if no device is connected. The left side contains the action toolbar and the right side contains a buffer of the memory content in hexadecimal and ASCII formats. The tool has additional functionalities: “Auto erase” and “Auto verify”. To display memory value from some address, fill in the start address into the Address combo and required size (in Bytes) into the Size combo. Click Read and the value read from memory will be displayed. The result of the operation is displayed in the bottom-left corner. If the operation ends with failure, more detailed info about the encountered error is displayed in the tooltip. The settings of the flash programmer window are not saved into the workspace. They are discarded if the tool is closed.



Buffer
Address: 0x00000000
Size: 0x1000

File on the disk

Target memory
(● Boot memory) ○ Address space
0x00000000 - 0x0003DBFF
Erasable block size: 0x200
Erased: 0x00000000 - 0x00000FFF
☑ Auto erase
☑ Auto verify
Whole memory:

Search
(● HEX) ○ ASCII

CRC
CRC8

```
Address     0  1  2  3  4  5  6  7  8  9  10 11 12 13 14 15  ASCII
0x00000000  3A 9A 76 7F EE 2D 71 AB 10 87 B7 53 07 B9 7C 64  :?v??-q????S??|d
0x00000010  ED 8E 33 40 73 30 9E BF 38 C9 9C 08 D6 F2 25 22  ??3@s0??8?????%"
0x00000020  0E F9 28 42 5B 03 88 94 B6 25 87 42 CD C3 60 FA  ??(B[????%?B??`?
0x00000030  D8 34 50 C4 2F CB C6 DE F8 E2 8D E8 F7 EE C4 57  ?4P?/??????????W
0x00000040  75 31 2D 88 63 0F BB 96 71 19 0F 8D F2 9C 14 0D  u1-?c???q???????
0x00000050  8E 86 F2 A3 E7 7D EA 3B 99 01 4D CB 50 E9 2C 6A  ?????}?;??M?P?,j
0x00000060  0B 94 EB B4 DE E7 BB BF 3C 8E 19 28 AE 55 7B EC  ????????<??(?U{?
0x00000070  4E B6 C9 0A 52 87 EF 1F 6D 50 68 0D 76 24 67 36  N???R???mPh?v$g6
0x00000080  57 20 B6 16 EC 10 EF DE AC 59 9A EB 00 CB 44 CB  W ???????Y????D?
0x00000090  90 87 83 61 0E 0B B6 CF E8 57 6D 16 6B CC FC 06  ???a?????Wm?k???
0x000000A0  05 F3 3A EC 9D 47 78 6F B7 19 CB 36 B1 95 E5 66  ??:??Gxo???6???f
0x000000B0  9A 33 07 92 FB 58 F6 92 F8 27 CE DB 59 A2 44 C3  ?3???X???'??Y?D?
0x000000C0  36 3F B8 47 18 7B D9 2F D8 72 35 4D 96 F9 41 32  6??G?{?/?r5M??A2
0x000000D0  7F E9 C3 EC FE 2F D1 47 FC 22 E7 36 63 B0 D9 FE  ?????/?G?"?6c???
0x000000E0  F6 86 A3 5B F8 78 17 DF 65 2D CB A3 4B C6 65 CF  ???[?x??e-??K?e?
0x000000F0  3D 65 C9 A9 87 E2 BC 66 F4 FB 4F 30 99 67 00 E3  =e?????f??O0?g??
0x00000100  B6 7B 89 AE 09 83 F6 82 D4 FB 17 05 8B 70 1F 58  ?{??????????p?X
0x00000110  F7 67 B8 6A 30 28 DA A6 A2 A4 A3 AD 7F A2 D9 C8  ?g?j0(??????????
0x00000120  29 53 0E CA 8B 53 E8 2B CB CD B9 0F 70 AE 24 5A  )S???S?+????p?$Z
0x00000130  C5 F5 05 D8 D7 90 BB 20 24 6C B2 0B 38 77 7A F2  ??????? $l??8wz?
0x00000140  55 9B 68 7C CC 96 51 9B E4 6C 96 D0 BF 55 5E 04  U?h|??Q??l???U^?
0x00000150  E3 99 3A DA 74 CF C9 27 2D 7F 1C EF 39 93 99 B9  ??:?t??'-???9???
0x00000160  E6 64 CB 1B 5A A8 B6 63 4C 91 19 FF DF C9 FA E1  ?d??Z??cL???????
0x00000170  A1 EB 53 E4 B1 5E FB 5C 64 B3 BE ED C9 A2 39 AD  ??S??^?\d?????9?
0x00000180  01 32 9D F8 18 8E 3D A3 56 84 18 C8 2B 58 C7 4E  ?2????=?V???+X?N
0x00000190  E4 B9 55 50 21 AC 05 03 5D 52 5D 1E 56 4D 65 07  ??UP!???]R]?VMe?
0x000001A0  2E E5 E9 E3 24 A2 4F C2 77 6E F3 24 C4 F9 F1 AB  .???$?O?wn?$????
0x000001B0  B2 BD CB 0C 39 80 97 C0 9C 72 6E 59 F3 09 AA CC  ????9????rnY????
0x000001C0  9B 60 12 08 6A 62 88 66 CE FA C2 64 2E 65 7D 5D  ?`??jb?f???d.e}]
0x000001D0  FF 33 D1 8D A0 51 D4 C4 1E 91 9D E2 1D 3E 4A 16  ?3???Q???????>J?
0x000001E0  25 AA C0 DA 6F 42 6C 60 3A 80 5B 22 28 4C 4B 9B  %???oBl`:?["(LK?
0x000001F0  37 D7 F4 26 AD 20 AD B5 5D BD A6 4E 40 89 95 DF  7??&? ??]??N@???
0x00000200  D5 70 17 45 DB B5 79 89 0E B7 F9 CB 33 3C EB FB  ?p?E??y?????3<??
0x00000210  1B 9B CE D7 A6 99 EE AF 54 10 01 EF 64 55 23 61  ????????T???dU#a
```

Figure 45. Flash Programmer Tool

Buffer : The buffer represents a part (copy) of the flash memory, that will be written or read from the processor. The buffer is defined by the address, size, and content.

- **Address** : Specifies the start address of the buffered space. The address cannot go lower or above the start or end address of the target memory.

- **Size** : The size of the buffer specifies how many bytes should be read or written or erased. The size is limited to 16 MB.
- **Fill …** : Opens a dialog that can fill the buffer with value (byte, word, double word, or random). Word and double word can be applied as big or little endian.
- **Clear buffer** : Clears displayed values and set the size to 0.

File on the disk : Load/save operations with a file on the disk.

- **Load …** : Loads the file into the buffer; `.srec` and `.hex` files are loaded with formatting (start address), other file types are processed as binary files without start address information. The address is the start address in the buffer where the value should be loaded, size tells what portion of data from the start should be loaded. The loaded data are merged with the existing buffer context (for example, the buffer can be extended) and the overlapping areas are overwritten. Load is also supported via drag-and-drop functionality, allowing files to be added directly to the grid
- **Save …** : Saves buffer or portion of buffer into a file. The supported file formats are `.srec, .hex`, and `.bin`.

Target memory : This block provides basic operations on target memory. The hex range on the top is the working memory range of the selected memory.

- **Selection of the memory type** : Select which memory should be targeted: `Boot memory` refers to the same memory as selected in the tool panel. `Address space` refers to the entire address space of the processor. The `Boot memory` can be re-configured directly from the tool using button below, while for the `Address space` the tool expects the memory is already available.
- **Configure** : Tries to configure the connected memory. It executes the same memory configuration as in the Boot memory configuration. If it is memory with FCB, then the last step of configuration tries to read the size of the minimal erasable block. The command is supported for external flash only, it is disabled for internal flash.
- **Erasable block size** : Defines the minimal erasable block size. This value is read from FCB if available or from the boot memory configuration for memories that do not use FCB. If no erasable block size is found, it is not possible to compute the erased size.
- **Erased** : The address range, which will be erased by the erase operation. The address range is determined by the buffer address and size, and it is aligned to an erasable block boundary. If the buffer is not aligned with the erasable block size, a warning is displayed.
- **Read** : Reads memory of a given range to fill the buffer. For memory types with ECC, reading is not possible if the memory is erased.
- **Write** : Writes values in the buffer to the memory, see also the options “Auto erase” and “Auto verify”
- **Verify** : Checks whether the values in the buffer match values in memory, values that do not match are highlighted with red color and the tooltip displays the value that is in the memory
- **Erase** : Erases the memory, always erases the closest upper multiplication of minimal erasable block size.
- **Blank check** : Checks if the memory is blank or not supported for internal flash.
- **Erase all** : Erases entire memory.
- **Auto erase** : On the write operation, the memory is erased before writing.
- **Auto verify** : After the write operation, verifies that that buffer was written properly.

Search : Search for a value in the buffer; the value can be provided as HEX numbers grouped as bytes (“FF AA”, “12 A3 00”) or ASCII (“abc”, “hello world”). The found value is highlighted in blue and the start address is displayed in the left corner. **CRC** : Compute CRC of the selected type for the buffered data.

5.7 SB editor

The SB Editor Tool is designed to create custom secure binary files for updates of the SW, data, and/or processor configuration. The SB editor supports SB formats: `SB2.x`, `SB3.x`, and `SBx` and also supports both regular `SB` file and `device HSM SB` file. The SB editor can be started using the command **main menu > Tools**

> SB Editor. When the SB editor is opened for the first time, and the workspace already contains a YAML file with the SB configuration (which is recommended), the SB Editor Tool offers to import the file.

SB files are generated using the nxpimage or nxpdevhsm command-line utilities, so the SB editor produces a configuration YAML file that is used as input for those tools. The SB editor is a smart GUI editor for the YAML configuration file. In the GUI, there are also supported “$” extensions (see high-level commands and $ variables), that are translated into the corresponding values in YAML.

5.7.1 SB editor contains the following main UI parts:

Top buttons bar : The bar that allows selecting the SB file type, import, export, or clear the current configuration.

Properties : The view that allows specifying properties for SB file generation.

Commands : The view that allows specifying the sequence of commands that shall be executed in the SB file.

Output : The panel, where the output files (the configuration YAML file and the resulting SB file) can be specified. The button **View** allows to update YAML configuration file and open in external editor. The button **Generate** allows to (re-)generate the resulting SB file.

Status and bottom buttons : The buttons that allow creating manufacturing package and starting manufacturing tool. The **OK/Cancel** buttons allow closing the SB editor with or without saving the configuration into the workspace.

5.7.2 File type

The SB editor supports the creation of:

- **regular SB file** that is secured by OEM keys specified on the Build view;
- **device-specific SB file** used for device HSM secured by device-specific keys. Creation of this SB file requires the processor to be connected to the computer. For details, see Connection.

The properties and commands for each file type might be different, so switch the file type may affect the current configuration.

5.7.3 Properties view

The properties view allows editing general properties and property lists. In case the processor contains any list-type property, it is selectable on the left side and the right side displays a property editor table. If no list-property is supported for the processor, the GUI displays just a property editor table.

The property editor table contains the following columns:

Group : The properties are logically grouped (for information only).

Property : Name of the property (for information only)

Value : The value of the property. If the value is not specified, there is an empty string. The value can be of the following type: integer number; string; predefined options (drop-down list), which include logical type (#true/ #false); file path

Resolved value : In case the value is specified in the form of ${variable}, this column contains the value of the variable; otherwise, it is the same as the value.

The property description is displayed in a tooltip.

5.7.4 Commands view

The view allows specifying the sequence of the commands that will be stored in the SB file and will be executed in the target processor. There are two types of commands: high-level and low-level. **High-level command sequences** are sequences of low-level commands that are typically used in SB files and are designed as SEC-tool specific extension of the low-level commands. These commands simplify creation/modifications of the SB file and will be replaced by low-level commands during the generation of the configuration YAML file. The names of all high-level commands start with “$”. High-level commands do not have any arguments, and the parameters of the commands inside cannot be modified. However, it is always possible to replace a high-level command by a set of the corresponding low-level commands and then customize the low-level commands. High-level commands may contain an optional low-level command that is stored into YAML only if all variables for the low-level command are defined. This is used if a high-level command might be extended in some configurations. Use a high-level command instead of a low-level command, as high-level commands are updated with the SEC Tool configuration.

The commands page contains the following panels:

The command sequence : The list of commands to be generated into the SB file. It is possible to select whether there will be displayed $ variables or real values.

Buttons bar : The bar is used to control the command sequence with the following buttons:

Button	Description
⬅	**Add** - to add the selected command into the current sequence
🗑	**Delete** - to remove the selected command from the current sequence
⬆ ⬇	**Move** - to move the selected command up or down in the current sequence
➜	**Expand** - to expand a high-level command sequence into the list of low-level commands

All available commands : The tree with all available commands is divided into “High-level command sequences” and “Low-level commands”.

Variables : Table of $ variables applicable as command arguments. It is possible to select whether the table should contain all variables or command-specific variables (the property-specific variables will be not be displayed).

Selected command : This panel shows the parameters of the selected command in a table.

Command parameters : The parameters for the selected command can be specified in the table at the bottom of the commands page. The first column represents the command itself and the other columns represent the command parameters. If the parameter is required, not specifying it results in an error. The description of the parameter is displayed as a tooltip. Use `${variable}` for parameter values, because these variables are updated within the SEC Tool configuration.

For a high-level command, an informative description is displayed instead of a parameter.

5.7.5 $ Variables

To allow reuse of a SEC Tool configuration for the SB file generation, `$` variables are provided. Names of these variables consist of the `$` character and the name is enclosed in curly braces (for example, `${family}`). The variables are used for properties and command parameters. For properties, the `$` variable name matches the name of the property. The list of all variables can be found on the Commands page at the right. It is possible to filter the variables to hide variables for “properties” or display all variables. The variables are not editable; however, it is possible to copy the name or value into the clipboard.

5.7.6 Validation

All the input values are validated and any problems are displayed in the window. Before generating the SB file, the YAML file is validated using the JSON schema. If any validation fails, the SB file cannot be generated.

It is recommended to create a SEC Tool workspace with the SB file, import the working configuration into the SB editor and start updating the working configuration.

5.7.7 Creating a manufacturing package button

If the SB file is applied on another computer (or in a factory), it is possible to create a manufacturing package that contains all the needed files. The package can be imported on another computer, see Manufacturing workflow.

5.7.8 To Manufacturing button

The button allows opening the Manufacturing Tool and applying the created SB file into a processor without creating a manufacturing package.

5.8 Merge Tool

The Merge Tool allows users to merge up to 8 images, given by the user, into one single binary image.

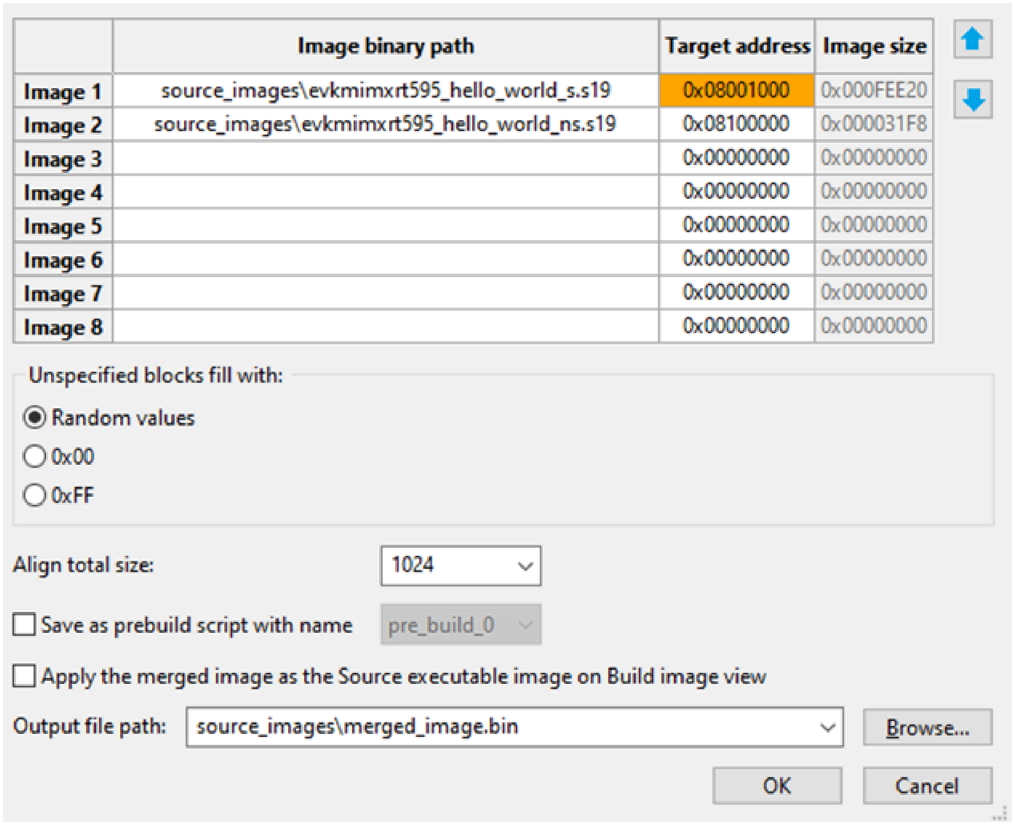


	Image binary path	Target address	Image size
Image 1	source_images\evkmimxrt595_hello_world_s.s19	0x08001000	0x000FEE20
Image 2	source_images\evkmimxrt595_hello_world_ns.s19	0x08100000	0x000031F8
Image 3		0x00000000	0x00000000
Image 4		0x00000000	0x00000000
Image 5		0x00000000	0x00000000
Image 6		0x00000000	0x00000000
Image 7		0x00000000	0x00000000
Image 8		0x00000000	0x00000000

Unspecified blocks fill with:
[✓ Random values] 0x00 0xFF

Align total size: 1024

☐ Save as prebuild script with name pre_build_0

☐ Apply the merged image as the Source executable image on Build image view

Output file path: source_images\merged_image.bin

Figure 46. Merge images tool

The configuration allows users to specify:

Images : Table to specify the path and target address for each image that used to be merged

Fill pattern : Pattern used to fill empty gaps between images, such as random value, 0x00, or 0xFF

Align total size : Align the size of the entire merged image

Pre-build script option : Merge script is executed when the **OK** button is pressed, but it can be stored as given pre-build script to be executed before build script.

Apply the merged image as the source executable image : Allows applying the merged image as a source executable image on the Build image view when the merge script is done

Output file path : Path where the merged binary image will be stored on a disk

6 Processor-specific workflows

This chapter describes the steps to successfully boot up the device to the required security level. It describes the creation of the bootable image, connecting your device, setting up your boot preferences, and writing the image into the selected boot memory. Common steps are described first, followed by device family-specific content. It is assumed that the image is executed on an NXP evaluation board.

This chapter addresses image preparation for the following toolchains:

- MCUXpresso IDE 11
- Keil Microcontroller Development Kit (MDK) 5 µVision
- IAR Embedded Workbench 8
- CodeWarrior Development Studio
- MCUXpresso for Visual Studio Code (VSCode)

On the following pages, you will learn how to:

- Get MCUXpresso SDK with an example project for a processor
- Open an example project for the processor in the toolchain
- Start with the SEC Tool
- Prepare asymmetric keys for the authenticated image
- Build a plain image in the selected toolchain
- Build a bootable image by SEC Tool
- Connect the NXP evaluation board
- Write a bootable image into the processor and (optionally) secure the processor and advance the life cycle

6.1 Common steps

This section provides common steps of the process.

6.1.1 Downloading MCUXpresso SDK

The MCUXpresso SDK offers open source drivers, middleware, and reference example applications to speed your software development. In this section, you can find information about downloading MCUXpresso SDK as a ZIP package or as a CMSIS pack and how to open an example project from the package. It is recommended to start with **iled_blinky** example. This example offers a simple check whether the resulting application is working - LED flashes with a 1 sec period.

- **Downloading MCUXpresso SDK package for MCUXpresso IDE, VSCode or CodeWarrior Development Studio**
  1. Visit MCUXpresso SDK Builder on NXP.com.

2. Select your board.
3. Build an SDK package for the selected toolchain and download it. **Note:** Starting with MCUXpresso IDE v11.1.0, you can download and install the MCUXpresso SDK package directly in the tool.

- **Downloading MCUXpresso SDK CMSIS pack**
  Alternatively, for the MDK µVision and IAR Embedded Workbench you can download CMSIS packs for the selected processor and board:
  – Device Family Pack (DFP): *NXP.{processor}_DFP.#.#.#.pack*
  – Board Support Pack (BSP): *NXP.EVK-{processor}_BSP.#.#.#.pack*
- **Downloading an example project for Keil MDK or IAR Embedded Workbench**
  For Keil MDK or IAR Embedded Workbench, it is also possible to download a single example project only. Once you have the SDK build available on MCUXpresso SDK Builder on NXP.com, click the download link and select **Download Standalone Example Project**. This project contains all sources and project files needed for the build.

6.1.2 Opening example project

- **MCUXpresso IDE**
  1. Drag-and-drop the downloaded MCUXpresso SDK package into the Installed SDKs view to install the package.
  2. Select **File > New > Import SDK examples…**.
  3. Select your processor and board and on the next page select the iled_blinky example.
- **Keil MDK 5 + Example package**
  1. Unpack the SDK package into the selected folder and open *boards\evkmimxrt10##\demo_apps\led_blinky \mdk\iled_blinky.uvmpw*.
  2. If you have downloaded a single example project only, unzip it into the selected folder and open the workspace file.
  3. Go to **Project > Options > Output** to ensure the option **Create HEX File** is selected.
- **Keil MDK 5 + CMSIS packs**
  1. Select **Project > Manage > Pack Installer**.
  2. In the **Devices** view, select **All Devices > NXP > MIMXRT10##**.
  3. In the **Packs** view, ensure that the following device-specific packs are installed: *NXP::{processor}#_DFP* and *NXP::EVK-{processor}_BSP*.
  4. Select the BSP pack
  5. In the **Examples** view, copy the **iled_blinky** example project into the selected folder.
  6. Go to **Project > Options > Output** to ensure the option **Create HEX File** is selected.
- **IAR Embedded Workbench + MCUXpresso SDK package**
  1. Unpack the SDK package into the selected folder and open *boards\evkmimxrt10##\demo_apps\led_blinky \iar\iled_blinky.eww*.
  2. If you have downloaded a single example project only, unzip it into the selected folder and open the workspace file.
- **CodeWarrior Development Studio**
  1. Unpack the SDK package into the selected folder and **import project** in the **Commander** pane.
  2. If the MCUXpresso Config Tools was used to create a project template, import this project template as described in point 1.
- **MCUXpresso for Visual Studio Code**
  1. Unpack the SDK package into the selected folder.
  2. Use command `> MCUXpresso for VSCode: Import Local/Remote repository`; select local and select the path of the SDK folder.

3. Use command `> MCUXpresso for VSCode: Import Example Application from Installed Repository`; fill all the items and confirm by clicking the **Create** button.

6.1.3 Building example project

Detailed information about project configuration and build is in the processor-specific sections below. For a quick evaluation, there are prebuilt application images from SDK examples for NXP evaluation boards provided in the installation layout. For details, refer to the installation subfolder **<SEC>sample_data/targets/{processor}/source_images**.

6.1.4 Setting up Secure Provisioning Tool

1. Start Secure Provisioning Tool:
   - Windows: Double-click the desktop shortcut, or use the Windows Start menu to locate the tool.
   - MacOS: Click the shortcut in the Dock, or use the Launchpad to locate the tool.
   - Linux: Click the shortcut in the Launcher, or use the Dash to locate the tool.
2. Create a new workspace by selecting **File > New Workspace …** from the **menu bar**. Select the path to the new workspace. Then select the target processor and click **Create**.
3. Connect the device to the host through USB, UART, SPI, or I2C.
4. Confirm that the connection is working by selecting **Target > Connection …** from the **menu bar** and clicking the **Test** button. Tweak if necessary.

6.1.5 Preparing secure keys

This section describes the generation of asymmetric keys necessary for authenticated or encrypted image creation. This operation is done only once and the keys can be used for all use-cases.

1. Select the **PKI management** view.
2. Ensure it does not already contain keys.
3. Click **Generate keys**.
4. In the **Generate keys** dialog, confirm the default settings and click **Generate**.

Note: The generated keys are located in the **keys/** subfolder and certificates (if any) in the **crts/** subfolder. It is recommended to back up generated keys before they are burned into fuses in the processor.

6.1.6 Build and write

The steps to build and write a bootable image are processor-specific, and will be listed below in the chapter for each processor. In this workflow, it is recommended to write the image prepared on the build; the SEC Tool also supports writing image built externally, but the configuration might require additional configuring and deeper experience.

6.1.6.1 BCA and FCF pages

This section applies only to processors that contain pages Bootloader Configuration Area (BCA) and Flash Configuration Field (FCF), for example MCXC series.

The BCA and FCF configuration blocks are stored in flash memory and are part of the bootable image. These blocks can be configured using the BCA/FCF configuration dialog through the BCA and FCF pages.

The BCA and FCF pages are not set into the bootable image, and the original content remains unchanged, unless there is any user requirement set to the page. If any requirement is set, the whole page is written, fields that have no user value are set to the default value. In that case, make sure that any value that was set in the source image is not unintentionally overwritten with the default values.

Note: When the life cycle is set to secured flash state, FCF is set into the bootable image even if there are no user requirements.

In the BCA/FCF configuration dialog, the current field values can be read from two sources:

1. From the flash of the connected processor.
2. From the source image selected on the **Build** tab.

6.2 i.MX 9x device workflow

This section describes the i.MX 93 and i.MX 95 device workflow in detail.

6.2.1 Preparing images for build for i.MX 9x devices

In this step, select the target memory where the image(s) is to be executed.

i.MX 9x devices have the following cores where image(s) can be executed:

- i.MX 93:
  - Cortex-M33, boot core
  - Cortex-A55, boot core
- i.MX 95:
  - Cortex-M33, boot core
  - Cortex-M7
  - Cortex-A55

The following target memories are supported for i.MX 9x devices:

- Image running in RAM

This image can be on an SD card/eMMC/FlexSPI, will be copied into RAM and executed from there during the boot. The following RAM types are supported:

- internal RAM
- SDRAM (DDR SDRAM)

There are several images required to build the resulting bootable image containing the AHAB container set.

- i.MX 93 images:
  - Primary image container set:
    - ELE firmware
    - LPDDR4 firmware files with U-Boot SPL
    - [Optional] Cortex-M33 application
  - Secondary image container set:
    - ARM Trusted Firmware (bl31 binary)
    - U-Boot
    - [Optional] TEE binary
- i.MX 95 images:
  - Primary image container set:
    - ELE firmware
    - DDR (LPDDR4 or LPDDR5) firmware files with the OEI DDR firmware
    - CM33 OEI TCM
    - CM33 System manager
    - U-Boot SPL

  - [Optional] Cortex-M7 application
- Secondary image container set:
  - ARM Trusted Firmware (bl31 binary)
  - U-Boot
  - TEE binary

For the image with Cortex-M7 and Cortex-M33 application, use the MCUXpresso SDK example. There is no need to modify the default configuration. This image is built to run in internal RAM.

DDR firmware files and ELE firmware can be downloaded from the Yocto Project:

- Example of ELE firmware: ELE firmware
- Example of DDR firmware files: DDR firmware

For details regarding the latest packages, see *i.MX Linux Release Notes*document RN00210.

The rest of the images are built from source code. Use the following repos on NXP i.MX main repository

- OEI DDR firmware, CM33 OEI TCM: OEI DDR firmware
- CM33 System manager: CM33 system manager repository
- U-Boot SPL, U-Boot: U-Boot SPL, U-Boot repository, see Build U-Boot with AHAB secure boot features for details.
- ARM Trusted Firmware (bl31 binary): ARM trusted firmware
- TEE binary: TEE Binary Repository

For additional Linux release materials, see Embedded Linux for i.MX Applications Processors

For example details, see **main menu > Help > SPSDK Documentation**:

- Examples - AHAB - i.MX 95 AHAB with U-Boot
- Examples - AHAB - i.MX 93 signed and encrypted AHAB image.

6.2.2 Connecting the board for i.MX 9x devices

This section contains information about configuring the evaluation board and connecting it to the SEC Tool:

- IMX95LPD5EVK-19 (IMX95LPD5BB-19 base board with IMX95LP5CPU-19 CPU board)
- MCIMX93-EVK (MCIMX93-BB base board with MCIMX93-SOM CPU board)
- MCIMX93-QSB

6.2.2.1 Table: Boot mode selection for board for i.MX 9x Cortex-M33 core

Boot mode/ Device	Serial bootloader (ISP mode)	eMMC	SD card	FlexSPI NOR
IMX95LPD5EVK-19	SW7: 1001	SW7: 1010	SW7: 1101	SW7: 1100 (N/A)
MCIMX93-EVK	SW1301: 1101	SW1301: 0001	SW1301: 0101	SW1301: 1011 (N/A)
MCIMX93-QSB	SW601: 1001	SW601: 1010	SW601: 1011	SW601: 1100 (N/A)

6.2.2.2 Table: Boot mode selection for board for i.MX 9x Cortex-A55 core

Boot mode/ Device	Serial bootloader (ISP mode)	eMMC	SD card	FlexSPI NOR
MCIMX93-EVK	SW1301: 1100	SW1301: 0000	SW1301: 0100	SW1301: 1010 (N/A)
MCIMX93-QSB	SW601: 0001	SW601: 0010	SW601: 0011	SW601: 0100 (N/A)

6.2.2.3 Step-by-step process:

1. See **Table: Boot mode selection for board for i.MX 9x Cortex-M33 core** or **Table: Boot mode selection for board for i.MX 9x Cortex-A55 core** in Connecting the board for i.MX 9x devices for instructions on how to set boot mode using DIP switches.
2. Connect to the USB1/USB port with the USB cable to your PC for the download link.
3. Connect to the DBG port with the USB cable to your PC for console output.
4. Power the board to POWER JACK/USB PD and power on the POWER SWITCH.
5. Ensure that the SEC Tool is already running with a workspace created for the chosen device. For more information, see Setting up Secure Provisioning Tool.
6. Open the **Connection** dialog and test the board connection.

Booting from SD card

For booting from an SD card, do the following:

1. Insert a micro SDHC card into the board.
2. Select **SD card, SDHC SD-card 64 GB** in the **Boot Memory Configuration**.

Booting from eMMC

For booting from an eMMC, do the following:

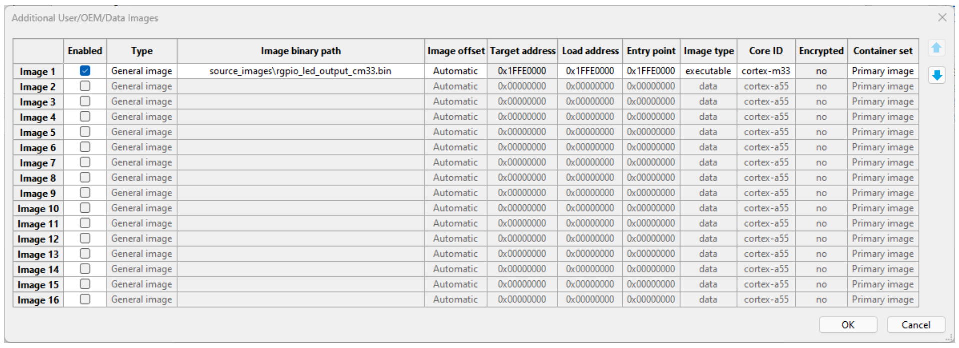
1. Check that the connected board already contains eMMC 64 GB.
2. Select eMMC: **SDHC eMMC 64 GB** in the **Boot Memory Configuration**.

6.2.3 Booting images for i.MX 9x devices

This section describes the building and writing process of bootable images.

6.2.3.1 i.MX 93 bootable image examples

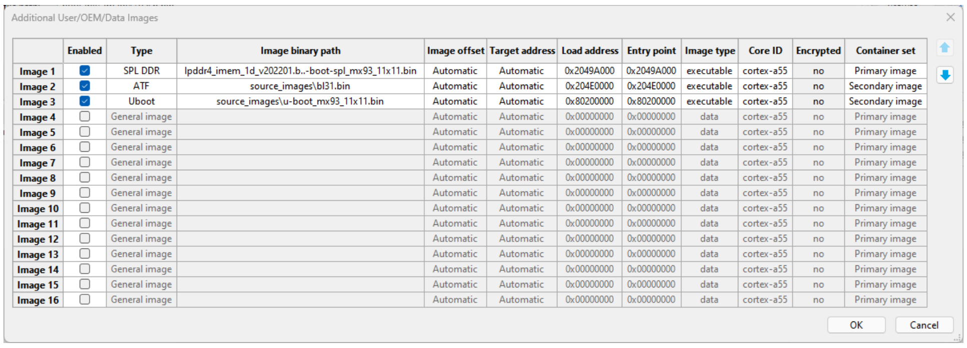
- A container set with the Cortex-M33 application. It is written to on-chip RAM by the nxpuuu utility during write and then started from there.



	Enabled	Type	Image binary path	Image offset	Target address	Load address	Entry point	Image type	Core ID	Encrypted	Container set
Image 1	✓	General image	source_images\rgpio_led_output_cm33.bin	Automatic	0x1FFE0000	0x1FFE0000	0x1FFE0000	executable	cortex-m33	no	Primary image
Image 2	☐	General image		Automatic	0x00000000	0x00000000	0x00000000	data	cortex-a55	no	Primary image
Image 3	☐	General image		Automatic	0x00000000	0x00000000	0x00000000	data	cortex-a55	no	Primary image
Image 4	☐	General image		Automatic	0x00000000	0x00000000	0x00000000	data	cortex-a55	no	Primary image
Image 5	☐	General image		Automatic	0x00000000	0x00000000	0x00000000	data	cortex-a55	no	Primary image
Image 6	☐	General image		Automatic	0x00000000	0x00000000	0x00000000	data	cortex-a55	no	Primary image
Image 7	☐	General image		Automatic	0x00000000	0x00000000	0x00000000	data	cortex-a55	no	Primary image
Image 8	☐	General image		Automatic	0x00000000	0x00000000	0x00000000	data	cortex-a55	no	Primary image
Image 9	☐	General image		Automatic	0x00000000	0x00000000	0x00000000	data	cortex-a55	no	Primary image
Image 10	☐	General image		Automatic	0x00000000	0x00000000	0x00000000	data	cortex-a55	no	Primary image
Image 11	☐	General image		Automatic	0x00000000	0x00000000	0x00000000	data	cortex-a55	no	Primary image
Image 12	☐	General image		Automatic	0x00000000	0x00000000	0x00000000	data	cortex-a55	no	Primary image
Image 13	☐	General image		Automatic	0x00000000	0x00000000	0x00000000	data	cortex-a55	no	Primary image
Image 14	☐	General image		Automatic	0x00000000	0x00000000	0x00000000	data	cortex-a55	no	Primary image
Image 15	☐	General image		Automatic	0x00000000	0x00000000	0x00000000	data	cortex-a55	no	Primary image
Image 16	☐	General image		Automatic	0x00000000	0x00000000	0x00000000	data	cortex-a55	no	Primary image

Figure 47. Additional images for Container set with Cortex-M7 application

- A container set with Cortex-A55 U-Boot (bootloader). It is written to the eMMC/SD card by the nxpuuu utility during write. During the boot, this bootloader is used to boot the Linux kernel image.



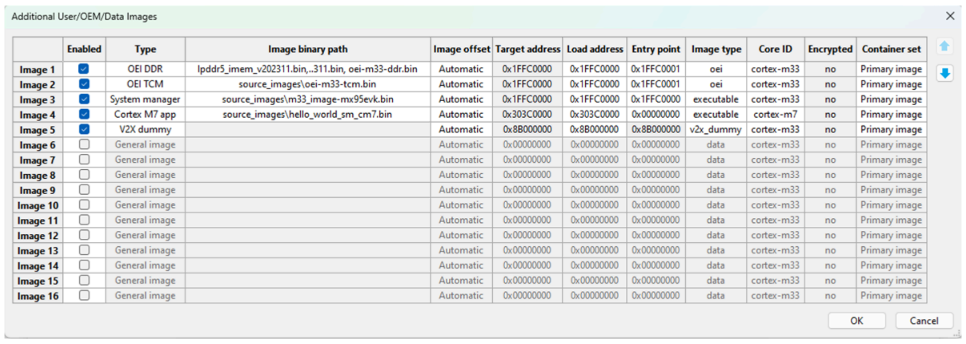
Additional User/OEM/Data Images

	Enabled	Type	Image binary path	Image offset	Target address	Load address	Entry point	Image type	Core ID	Encrypted	Container set
Image 1	✓	SPL DDR	lpddr4_imem_1d_v202201.b..-boot-spl_mx93_11x11.bin	Automatic	Automatic	0x2049A000	0x2049A000	executable	cortex-a55	no	Primary image
Image 2	✓	ATF	source_images\bl31.bin	Automatic	Automatic	0x204E0000	0x204E0000	executable	cortex-a55	no	Secondary image
Image 3	✓	Uboot	source_images\u-boot_mx93_11x11.bin	Automatic	Automatic	0x80200000	0x80200000	executable	cortex-a55	no	Secondary image
Image 4	☐	General image		Automatic	Automatic	0x00000000	0x00000000	data	cortex-a55	no	Primary image
Image 5	☐	General image		Automatic	Automatic	0x00000000	0x00000000	data	cortex-a55	no	Primary image
Image 6	☐	General image		Automatic	Automatic	0x00000000	0x00000000	data	cortex-a55	no	Primary image
Image 7	☐	General image		Automatic	Automatic	0x00000000	0x00000000	data	cortex-a55	no	Primary image
Image 8	☐	General image		Automatic	Automatic	0x00000000	0x00000000	data	cortex-a55	no	Primary image
Image 9	☐	General image		Automatic	Automatic	0x00000000	0x00000000	data	cortex-a55	no	Primary image
Image 10	☐	General image		Automatic	Automatic	0x00000000	0x00000000	data	cortex-a55	no	Primary image
Image 11	☐	General image		Automatic	Automatic	0x00000000	0x00000000	data	cortex-a55	no	Primary image
Image 12	☐	General image		Automatic	Automatic	0x00000000	0x00000000	data	cortex-a55	no	Primary image
Image 13	☐	General image		Automatic	Automatic	0x00000000	0x00000000	data	cortex-a55	no	Primary image
Image 14	☐	General image		Automatic	Automatic	0x00000000	0x00000000	data	cortex-a55	no	Primary image
Image 15	☐	General image		Automatic	Automatic	0x00000000	0x00000000	data	cortex-a55	no	Primary image
Image 16	☐	General image		Automatic	Automatic	0x00000000	0x00000000	data	cortex-a55	no	Primary image

Figure 48. Additional images for Container set with Cortex-A55 U-Boot

6.2.3.2 i.MX 95 bootable image examples

- A container set with the Cortex-M7 application. It is written to on-chip RAM by the nxpuuu utility during write and then started from there.

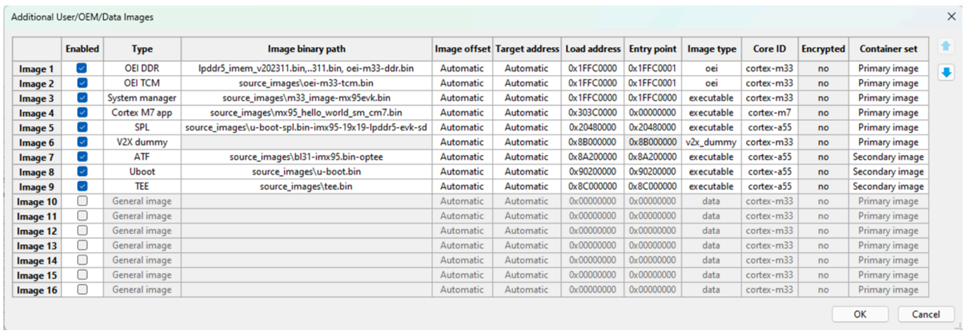


Additional User/OEM/Data Images

	Enabled	Type	Image binary path	Image offset	Target address	Load address	Entry point	Image type	Core ID	Encrypted	Container set
Image 1	✓	OEI DDR	lpddr5_imem_v202311.bin,..311.bin, oei-m33-ddr.bin	Automatic	0x1FFC0000	0x1FFC0000	0x1FFC0001	oei	cortex-m33	no	Primary image
Image 2	✓	OEI TCM	source_images\oei-m33-tcm.bin	Automatic	0x1FFC0000	0x1FFC0000	0x1FFC0001	oei	cortex-m33	no	Primary image
Image 3	✓	System manager	source_images\m33_image-mx95evk.bin	Automatic	0x1FFC0000	0x1FFC0000	0x1FFC0000	executable	cortex-m33	no	Primary image
Image 4	✓	Cortex M7 app	source_images\hello_world_sm_cm7.bin	Automatic	0x303C0000	0x303C0000	0x00000000	executable	cortex-m7	no	Primary image
Image 5	✓	V2X dummy		Automatic	0x8B000000	0x8B000000	0x8B000000	v2x_dummy	cortex-m33	no	Primary image
Image 6	☐	General image		Automatic	0x00000000	0x00000000	0x00000000	data	cortex-m33	no	Primary image
Image 7	☐	General image		Automatic	0x00000000	0x00000000	0x00000000	data	cortex-m33	no	Primary image
Image 8	☐	General image		Automatic	0x00000000	0x00000000	0x00000000	data	cortex-m33	no	Primary image
Image 9	☐	General image		Automatic	0x00000000	0x00000000	0x00000000	data	cortex-m33	no	Primary image
Image 10	☐	General image		Automatic	0x00000000	0x00000000	0x00000000	data	cortex-m33	no	Primary image
Image 11	☐	General image		Automatic	0x00000000	0x00000000	0x00000000	data	cortex-m33	no	Primary image
Image 12	☐	General image		Automatic	0x00000000	0x00000000	0x00000000	data	cortex-m33	no	Primary image
Image 13	☐	General image		Automatic	0x00000000	0x00000000	0x00000000	data	cortex-m33	no	Primary image
Image 14	☐	General image		Automatic	0x00000000	0x00000000	0x00000000	data	cortex-m33	no	Primary image
Image 15	☐	General image		Automatic	0x00000000	0x00000000	0x00000000	data	cortex-m33	no	Primary image
Image 16	☐	General image		Automatic	0x00000000	0x00000000	0x00000000	data	cortex-m33	no	Primary image

Figure 49. Additional images for Container set with Cortex-M7 application

- A container set with Cortex-A55 U-Boot (bootloader). It is written to the eMMC/SD card by the nxpuuu utility during write. During the boot, this bootloader is used to boot the Linux kernel image.



Additional User/OEM/Data Images

	Enabled	Type	Image binary path	Image offset	Target address	Load address	Entry point	Image type	Core ID	Encrypted	Container set
Image 1	✓	OEI DDR	lpddr5_imem_v202311.bin,..311.bin, oei-m33-ddr.bin	Automatic	Automatic	0x1FFC0000	0x1FFC0001	oei	cortex-m33	no	Primary image
Image 2	✓	OEI TCM	source_images\oei-m33-tcm.bin	Automatic	Automatic	0x1FFC0000	0x1FFC0001	oei	cortex-m33	no	Primary image
Image 3	✓	System manager	source_images\m33_image-mx95evk.bin	Automatic	Automatic	0x1FFC0000	0x1FFC0000	executable	cortex-m33	no	Primary image
Image 4	✓	Cortex M7 app	source_images\mx95_hello_world_sm_cm7.bin	Automatic	Automatic	0x303C0000	0x00000000	executable	cortex-m7	no	Primary image
Image 5	✓	SPL	source_images\u-boot-spl.bin-imx95-19x19-lpddr5-evk-sd	Automatic	Automatic	0x20480000	0x20480000	executable	cortex-a55	no	Primary image
Image 6	✓	V2X dummy		Automatic	Automatic	0x8B000000	0x8B000000	v2x_dummy	cortex-m33	no	Primary image
Image 7	✓	ATF	source_images\bl31-imx95.bin-optee	Automatic	Automatic	0x8A200000	0x8A200000	executable	cortex-a55	no	Secondary image
Image 8	✓	Uboot	source_images\u-boot.bin	Automatic	Automatic	0x90200000	0x90200000	executable	cortex-a55	no	Secondary image
Image 9	✓	TEE	source_images\tee.bin	Automatic	Automatic	0x8C000000	0x8C000000	executable	cortex-a55	no	Secondary image
Image 10	☐	General image		Automatic	Automatic	0x00000000	0x00000000	data	cortex-m33	no	Primary image
Image 11	☐	General image		Automatic	Automatic	0x00000000	0x00000000	data	cortex-m33	no	Primary image
Image 12	☐	General image		Automatic	Automatic	0x00000000	0x00000000	data	cortex-m33	no	Primary image
Image 13	☐	General image		Automatic	Automatic	0x00000000	0x00000000	data	cortex-m33	no	Primary image
Image 14	☐	General image		Automatic	Automatic	0x00000000	0x00000000	data	cortex-m33	no	Primary image
Image 15	☐	General image		Automatic	Automatic	0x00000000	0x00000000	data	cortex-m33	no	Primary image
Image 16	☐	General image		Automatic	Automatic	0x00000000	0x00000000	data	cortex-m33	no	Primary image

Figure 50. Additional images for Container set with Cortex-A55 U-Boot

6.2.3.3 Booting/loading unsigned image

First, build a bootable image:

1. Select the **Unsigned boot type** in the **Toolbar**.
2. Select **boot device** in the **Toolbar**.
3. Switch to the **Build image**

4. Open the **Additional User/OEM Image** dialog by clicking the **Additional images** button.
5. Configure the images for the image container, see i.MX 93 bootable image examples or i.MX 95 bootable image examples. Select the prepared images from Preparing images for build for i.MX 9x devices.
   - Enable the image in the **Enabled** column.
   - Select the image entry using **Type**
   - Select the binary(s) in the **Image binary path**.
   - Select **Container set**, it should be either Primary image or Secondary image.
   - All other parameters are preset with default values for the selected image entry type. These parameters can be customized.
6. Close the dialog by clicking the **OK** button.
7. Click the **Build image** button to build a bootable image, `flash.bin.`

When the bootable image has been successfully built:

1. Make sure that the board is in Serial bootloader (ISP) mode.
2. Switch to the **Write image** view.
3. Click the **Write image** button.
   - The nxpuuu utility is used to load the bootable image to the device.
4. If the write operation was successful:
   - For **eMMC** or **SD card**, switch boot mode (see **Table: Boot mode selection for board for i.MX 9x Cortex-M33 core** or **Table: Boot mode selection for board for i.MX 9x Cortex-A55 core** in Connecting the board for i.MX 9x devices) and reset the board. The applications shall run, `hello_world` in Cortex-M7 and/or U-Boot in Cortex-A55.
   - For **On-chip RAM**, the Cortex-M7/Cortex-M33 application runs after it is written.

6.2.3.4 Booting signed bootloader image from Cortex-A55 core on i.MX 93

This section describes the building and writing of a signed image, bootloader for Cortex-A55 for i.MX 93. Keys generated in the **PKI management** view are needed in this step. For more information about generating keys, see Generate keys.

First, build a bootable image (bootloader for Cortex-A55):

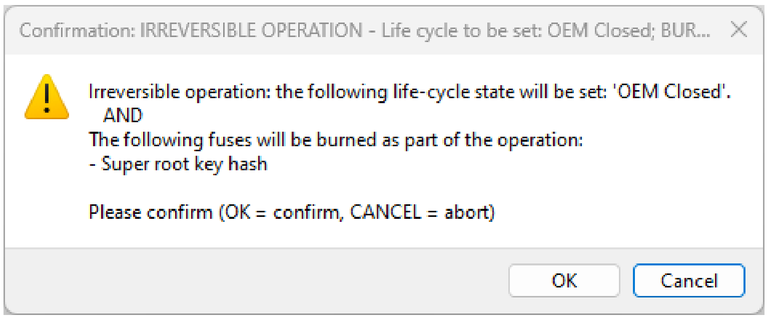
1. In the **Toolbar** set **Boot type** to **Signed**.
2. Select **boot device** in the **Toolbar**.
3. Switch to the **Build image**
4. Open the **Additional User/OEM Image** dialog and configure the images for the bootloader. See the example configuration **Additional images for Container set with Cortex-A55 U-Boot** in i.MX 93 bootable image examples and Build U-Boot with AHAB secure boot features.
5. For **Authentication key** select any key, for example, *SRK1*.
6. Select the **OEM Open** or **OEM Closed** life cycle. See chapter Get ELE events with nxpele utility how to verify the application in the OEM Open life cycle.
7. Click the **Build image** button.
8. Check that the bootable image was built successfully.

When the bootable image has been successfully built:

1. Make sure that the board is in Serial bootloader (ISP) mode.
2. Switch to the **Write image** view.
3. Make sure that the **Use built image** checkbox is selected. This will write the built bootloader to the selected boot device.

Note: To write another prepared bootable image, another bootloader or a complete image with Linux kernel, unselect the checkbox and select your image. The bootloader built on **Build image** is still used during write to write fuses and to update the life cycle with the nxpele utility.

4. Click the **Write image** button.
   - The nxpuuu utility is used to load the bootloader, with U-Boot, to RAM.
   - The nxpele utility is used to write fuses and to update life cycle.
   - The nxpuuu utility is used to load the bootloader/your prepared bootable image to the boot device.
5. In the following window, confirm to write fuses and update the life cycle:
   - **OK** - Continue writing the image and burning fuses.
   - **Cancel** - Abort writing the image and burning fuses.


Confirmation: IRREVERSIBLE OPERATION - Life cycle to be set: OEM Closed; BUR...
OK
Cancel

Irreversible operation: the following life-cycle state will be set: 'OEM Closed'.
AND
The following fuses will be burned as part of the operation:
- Super root key hash

Please confirm (OK = confirm, CANCEL = abort)

Figure 51. Burn fuses on i.MX 93

1. If the write operation was successful, switch boot mode (see **Table: Boot mode selection for board for i.MX 9x Cortex-A55 core** in Connecting the board for i.MX 9x devices) and reset the board. The applications shall run U-Boot/your prepared bootable image in Cortex-A55.

6.2.3.5 Write fuses and update life cycle with nxpele over U-Boot

The nxpele utility is used during provisioning to write fuses or to update life cycle. It is also used to read fuses in the **OTP Configuration** dialog. The nxpele utility communicates with EdgeLock Enclave over U-Boot, which must be running on the target.

In these cases, the bootable image configured in the **Additional User/OEM Image** dialog is used as a bootloader with the U-Boot. The U-Boot must be built with AHAB features, see Build U-Boot with AHAB secure boot features for details. During the device provisioning, the bootloader is loaded to the target and started.

6.2.3.6 Build U-Boot with AHAB secure boot features

The U-Boot/SPL https://github.com/nxp-imx/uboot-imx provides extra secure boot features. The features enable the nxpele utility to communicate with EdgeLock Enclave. The support is enabled by setting `CONFIG_AHAB_BOOT=y` in the build.

If the nxpele utility over fastboot is used, multiplexing of the console output to fastboot must be enabled by setting `CONFIG_CONSOLE_MUX=y`.

6.2.3.7 Get ELE events with nxpele utility

The `nxpele get-event` command can retrieve stored events in EdgeLock Enclave. The nxpele utility communicates with ELE over U-Boot, which must be running on the target.

Thanks to this, we can, for example, check the container authentication status in the OEM Open life cycle:

1. See Booting signed bootloader image from Cortex-A55 core on i.MX 93. In this procedure, keep the life cycle in **OEM Open**. In the OEM Open life cycle, the authentication result is ignored.

2. Connect the DBG port to your PC.
3. Switch to boot mode and reset the board.
4. Check what serial COM port is the U-Boot console output.
5. Close the serial connection session if opened in a terminal.
6. Execute `nxpele get-events`

Example when authentication succeeds:

```
C:\nxp\SEC_Provi_25.09\bin\_internal\tools\spsdk>nxpele -f mimx9352 -p COM21
 get-events
ELE get events ends successfully.
Event count:     0
```

Example when authentication fails:

```
C:\nxp\SEC_Provi_25.09\bin\_internal\tools\spsdk>nxpele -f mimx9352 -p COM21
 get-events
ELE get events ends successfully.
Event count:     2
Event[0]:      0x0287FAD6
  IPC ID:        Application Processor message unit
  Command:       OEM Container authenticate
  Indication:    The key hash verification does not match OTP
  Status:        The request was successful
Event[1]:      0x0287FAD6
  IPC ID:        Application Processor message unit
  Command:       OEM Container authenticate
  Indication:    The key hash verification does not match OTP
  Status:        The request was successful
```

6.3 KW45xx/K32W1xx/MCXW71xx/KW47xx/MCXW72xx device workflow

This chapter describes workflow for KW45xx/K32W1xx/MCXW71xx/KW47xx/MCXW72xx processors.

6.3.1 Preparing source image for KW45xx/K32W1xx/MCXW71xx/KW47xx/MCXW72xx devices

In this step, select the target memory where the image is executed. The following options are available for KW45xx/K32W1xx/MCXW71xx/KW47xx/MCXW72xx devices:

- Image running from an internal flash (XIP image)

There is no need to modify the default configuration, build the MCUXpresso SDK example as it is.

6.3.2 Connecting the board for KW45xx/K32W1xx/MCXW71xx/KW47xx/MCXW72xx devices

This section contains information about configuring the following evaluation boards and connecting them to SEC Tool:

- KW45B41Z-EVK
- K32W148-EVK
- MCXW71-FRDM
- MCXW71-EVK
- KW47-EVK

It is assumed that the SEC Tool is already running with a workspace created for an KW45xx/K32W1xx/MCXW71xx/KW47xx/MCXW72xx device. For more information, see Setting up Secure Provisioning Tool.

For the KW45B41Z-EVK and K32W148-EVK development boards:

1. Connect the J14 port to your PC with a USB cable.
2. Set the JP25 jumper to enable the SW4 button.
3. Enable the ISP boot mode by holding the SW4 button and reset.
4. In the **Connection** dialog, test the connection to the processor.

For the MCXW71-FRDM development board:

1. Connect the J10 port to your PC with a USB cable.
2. Enable the ISP boot mode by holding the SW3 button and reset.
3. In the **Connection** dialog, test the connection to the processor.

For the MCXW71-EVK development board:

1. Connect the J14 port to your PC with a USB cable.
2. Enable the ISP boot mode by holding the SW4 button and reset.
3. In the **Connection** dialog, test the connection to the processor.

For the KW47-EVK development board:

1. Connect the J14 port to your PC with a USB cable.
2. Enable the ISP boot mode by holding the SW4 button and reset.
3. In the **Connection** dialog, test the connection to the processor.

6.3.3 Booting images for KW45xx/K32W1xx/MCXW71xx/KW47xx/MCXW72xx devices

This section describes the building and writing of bootable images. For KW45xx/K32W1xx/MCXW71xx/KW47xx/MCXW72xx, the SEC Tool supports XIP images only.

6.3.3.1 Booting plain or CRC image

Plain images are typically used for development. Start with this boot type before working with secured images to verify that the executable image works properly. Dual image boot is supported only for secure boot types.

First, build a bootable image:

1. Make sure that you have selected the **Plain unsigned** or **Plain with CRC** boot type is selected in the toolbar.
2. Switch to the **Build image** view.
3. Select an application image built in Building example project as a **Source executable image**.
4. If there is a binary image, set the start address to 0x0.
5. Click the **Build image** button to build a bootable image. The result is a binary bootable image.

When the bootable image is built, upload it to the processor:

1. Make sure that the processor is in ISP mode.
2. Switch to the **Write image** view.
3. Click the **Write image** button.

If the write operation was successful, reset the board.

6.3.3.2 Booting signed image

This section describes building and writing a signed image.

Build a bootable image:

1. Select the **Plain signed** boot type in the toolbar.
2. Switch to the **Build image** view.
3. Select an application image built in Building example project as a **Source executable image**.
4. Ensure that you have the keys on the PKI management tab. Evaluation boards KW45B41Z-EVK, K32W148-EVK, MCXW71-FRDM, and KW47-EVK are produced with preprogrammed ROKTH and SB3KDK keys in the fuses `CUST_PROD_OEMFW_AUTH_PUK` and `CUST_PROD_OEMFW_ENC_SK`. These keys are also distributed in the SEC Tool and can be imported from the tool folder `sample_data\targets\<processor>\board_example_keys`. See Import/Export keys. These keys are intended for evaluation purposes only and must not be used for production.
5. For **Authentication key** select any key, for example ROT1: IMG1_1
6. Use an imported value or create your own (random) one for **SB3KDK** symmetric key.
7. If needed, open **Dual image boot** and configure. Image must be linked to the **Flash Logical Window.**
8. Keep **OEM Open** in the life cycle.
9. Click the **Build image** button to build a bootable image. The result is an SB3 capsule for installation into the processor.

When the bootable image and SB3 capsule have been successfully built, you can upload to the processor:

1. Make sure that the processor is in ISP mode.
2. Switch to the **Write image** view.
3. Keep **OEM Open** in the life cycle.
4. Make sure that the IFR fields written in otp_config.sb are not burned already. All IFR fields are one time programmable.
5. Click the **Write image** button.

Note: If IFR fields are written in the otp_config.sb, it is not possible to receive otp_config.sb multiple times.

6.3.3.3 Booting PRINCE encrypted image

Encrypted unsigned images, images with CRC or signed images are supported. The process of creating an encrypted image is similar to a signed image. In addition, configure encrypted regions in the **Build image** view. Use the **PRINCE regions** button to configure encrypted regions. In combination with the dual boot, set one region for image0 and one for image1. Setting a region only for image0 does not encrypt image1.

Image encryption is performed when the image is written to the target memory.

The regions configuration is included into the ROMCFG page.

Note: Open OTP/IFR configuration to review the PRINCE settings in the ROMCFG block(s) as the block must be completely specified and can be written only once. It is an irreversible operation.

6.3.3.4 Life cycle for KW45xx/K32W1xx/MCXW71xx/KW47xx/MCXW72xx devices

The default life cycle, which should be used for development, is **OEM Open**. Before you deploy the application, set the **OEM Closed** or **OEM Locked** life cycle (see documentation for the target processor for detailed description).

Note: Change of the life cycle is irreversible.

Once the processor is in "OEM Closed" or "OEM Locked" mode, the tool does not allow initializing the ROMCFG page. The application can still be updated via the SB file.

Table: Boot type and life cycle for KW45xx/K32W1xx/MCXW71xx/KW47xx/MCXW72xx

	OEM OPEN	OEM CLOSED/LOCKED
Plain unsigned or CRC boot type	- Only user fuses burnt- SB file not used	- life cycle fuse burnt - SB file not used

	OEM OPEN	OEM CLOSED/LOCKED
Plain signed boot type	- RKTH and SB3KDK burnt in write script - user fuses in write script or OTP SB file- SB file used	- RKTH and SB3KDK burnt in write script - user fuses in OTP SB file- SB file used
Encrypted boot type	- RKTH and SB3KDK burnt in write script - user fuses and IFR in encryption write script or SB file- SB file used	- RKTH and SB3KDK burnt in write script - user fuses and IFR encryption in OTP SB file- SB file used

The table below shows the security assets installed in different life cycles and trust provisioning types:

	Open life cycle, no trust provisioning	Closed life cycle, no trust provisioning	Open life cycle, Edge Lock 2GO	Closed life cycle, EdgeLock 2GO
SB3KDK	Write script	Write script	sb3kdk.asc	sb3kdk.asc
RKTH fuses	Write script	Write script	rkth.bin	rkth.bin
Other custom fuses and IFR	Write script or otp_config.sb	otp_config.sb	otp_config.sb	otp_config.sb

6.3.3.5 EdgeLock 2GO

EdgeLock 2GO is described in EdgeLock 2GO trust provisioning workflow. Below are KW45xx/K32W1xx/MCXW71xx/KW47xx/MCXW72xx specific comments:

- The life cycle is set to the life cycle set to the secure objects. The RKTH and SB3KDK fuses are always burned during provisioning, even in the development life cycle.
- Address for secure objects must be in internal flash, provisioning firmware erases the area before writing the secure objects.
- For KW47 and MCXW72, the OEM_SEC_BOOT_EN fuse is burned by the provisioning firmware if the secure object is set to either “OEM Closed” or “OEM Locked” life cycle.

6.3.3.6 Update NBU firmware

This feature is available only for processors with radio. The NBU firmware is distributed within the MCUXpresso SDK as an SB3 file or as a binary file (*.xip), in folder `middleware\wireless\ble_controller\bin`. There are two options to update the NBU firmware. First, use a prepared SB file with NBU firmware. It can be used for the EVK and FRDM boards with provisioned NXP keys. The other way is to prepare a custom SB file that will load the NBU firmware.

SB file with NXP keys

1. Create or open a workspace for KW45xx/K32W1xx/MCXW71xx/KW47xx/MCXW72xx.
2. In the **menu bar**, select **Tools > Manufacturing Tool**.
3. Select the **Apply SB file** operation.
4. Provide the SB file.
5. Detect a connected device by clicking the **Auto detect** button.
6. Load the SB file by clicking the **Start** button.

Custom SB file

1. Open a workspace for KW45xx/K32W1xx/MCXW71xx/KW47xx/MCXW72xx with the custom keys that were provisioned to the device.
2. Copy NBU firmware into workspace as `source_images\nbu_firmware.xip`
3. In the **menu bar**, select **Tools > SB Editor**.
4. Fill the **Properties** tab or import a setting from the SB file created on the **Build** tab.

5. Switch to the **Commands** tab.
6. Select High-level command *update-nbu-firmware*
7. Prepare the final SB file by clicking the **Generate** button.
8. Click the **To Manufacturing tool** button to switch to the manufacturing window with preselected **Apply SB file** operation and preselected SB file.
9. Detect a connected device by clicking the **Auto detect** button.
10. Load the SB file by clicking the **Start** button.

6.4 LPC55(S)0x/1x/2x/6x device workflow

This chapter describes workflow for LPC55(S)0x/1x/2x/6x, NHS52S04, and MCXW236 processors.

6.4.1 Preparing source image for LPC55(S)0x/1x/2x/6x devices

In this step, you must select the target memory where the image will be executed. The following option is available for LPC55Sxx devices:

- Image running from an internal flash (XIP image)

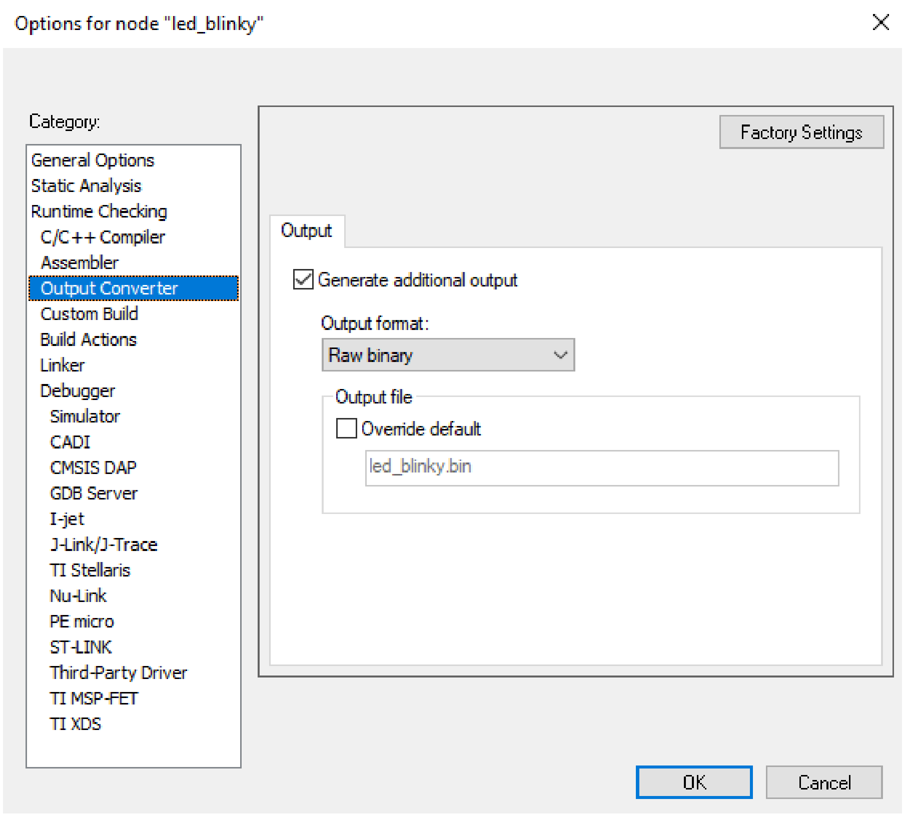
6.4.1.1 Image running from internal flash

- **MCUXpresso IDE**

1. Build the project.
2. Open the debug folder.
3. Right-click the named <your.project>.axf file.
4. Select **Binary Utilities** > **Create binary**.

- **IAR**

1. In **Project** > **Options** > **Output Converter**, check **Generate additional output** and select **Raw binary** output format.



[✓] Generate additional output

Output format: Raw binary

Output file

[ ] Override default

led_blinky.bin

Figure 52. IAR options

1. Build the project. You will find the output image built as `boards\\lpc55s\#\#\\demo\_apps\\led\_ blinky\\iar\\led\_blinky\\led\_blinky.bin`.

- **Keil MDK 5**

1. In **Project** > **Options** > **User** > **After Build/Rebuild**, check the **Run #1** option.
2. Enter the following in the **User Command** path (where *myprog* is the project's Name of Executable): `C: \Keil\ARM\ARMCC\bin\fromelf.exe --bin --output=myprog.bin myprog.axf`.
3. Build the image. You will find the output image built as `boards\lpc55s\#\#\\demo\_apps\led\_ blinky\mdk\led\_blinky\led\_blinky.bin`.

6.4.2 Connecting the board for LPC55(S)0x/1x/2x/6x devices

This section contains information about configuring the following LPC5Sxx evaluation boards and connecting them to SEC:

- LPCexpresso55S69
- LPCexpresso55S66
- LPCexpresso55S28
- LPCexpresso55S26
- LPCexpresso55S16
- LPCexpresso55S14
- LPCexpresso55S06
- LPCexpresso55S04
- NHS52Sxx-EVK
- FRDM-MCXW236B

It is assumed that the SEC Tool is already running with a workspace created for an LPC device. For more information, see Setting up Secure Provisioning Tool.

1. To communicate via UART, connect USB cable to P6 connector, for USB communication use P9 connector.
2. Enable the ISP boot mode by holding the ISP button and reset.
3. Ensure that you have selected the **Unsigned** boot type in the **Toolbar**.
4. In the **Connection** dialog, set the connection to **USB** or **UART** according to the selected port and test the connection to the processor.

6.4.3 Booting images for LPC55(S)0x/1x/2x/6x devices

This section describes the building and writing of images.

6.4.3.1 Security levels

The following security levels are supported in SEC:

Unsigned boot types : Default processor configuration that does not provide any security. It is recommended to start with the unsigned boot type to ensure the bootable image works on the processor. Unsigned boot types are intended for development only.

Signed or encrypted boot types - unsealed : Unsealed boot types are also designed to be used during development to ensure the selected boot type works well. In the KeyStore, CFPA, and CMPA pages are written into the processor. CMPA page is not sealed and can be updated or erased.

Signed or encrypted boot types - sealed : A sealed CMPA page is recommended for production. Select the **Deployment** life cycle in the **Write image** view to seal the CMPA page. Once sealed, it cannot be changed or erased.

6.4.3.2 Booting Plain/Plain with CRC image

This section describes the building and writing of plain/plain with CRC image.

1. In the **Toolbar**, set **Boot Type** to **Plain unsigned** or **Plain with CRC**.
2. As a **Source executable image**, use the image from Preparing source image for LPC55(S)0x/1x/2x/6x devices as a **Source executable image**.
3. In the case of a binary image, set the start address to *0x0*.
4. If needed, open **Dual image boot** and configure.
5. Click the **Build image** button.
6. Check that the bootable image was built successfully.

Once the image has been successfully built, do the following:

1. Make sure that the board is in ISP mode.
2. Click the **Write image** view.
3. Click the **Write image** button.

If the write operation was successful, reset the board.

6.4.3.3 Booting plain signed or PRINCE encrypted image

This section describes the building and writing of an authenticated or PRINCE encrypted image. Keys generated in the **PKI management** view are needed in this step. For more information about generating keys, see Generate keys.

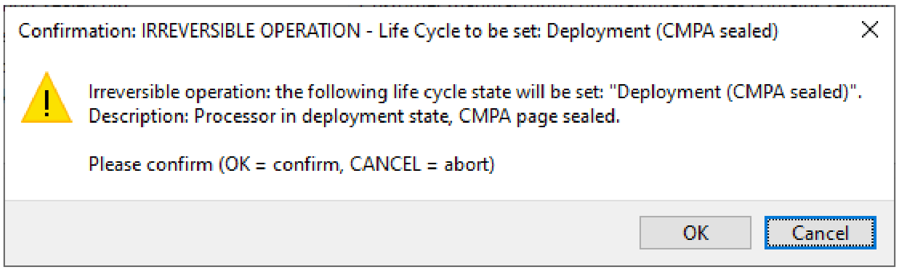
Note: The keys are also used for **Encrypted (PRINCE) Plain and with CRC** boot type because the bootable image is updated using SB capsule, which must be signed.

1. In the **Toolbar**, set **Boot type** to **Plain signed**, **Encrypted (PRINCE) unsigned, Encrypted (PRINCE) with CRC**, or **Encrypted (PRINCE) signed**.
2. In the **Build image** view, use the image from Preparing source image for LPC55(S)0x/1x/2x/6x devices as a source executable image.
3. For **Authentication key**, select any keychain, for example, *ROT1: IMG1_1_1*.
4. Open the PRINCE configuration and check the configuration. Set the size of the PRINCE region based on the size of the bootable image.
5. If needed, open **Dual image boot** and configure. For a PRINCE encrypted image, set one region for image0 and one for image1. Setting a region only for image0 does not encrypt image1.
6. Click the **Build image** button.
7. Check that the bootable image was built successfully.

To write the image, do the following:

1. Select the **Write image** view.
2. Make sure that the board is connected and the ISP mode is enabled (See Connecting the board for LPC55(S)0x/1x/2x/6x devices)
3. Click the **Write image** button.
   If the write operation was successful, reset the board.

Once the image can be successfully executed in the processor, select the **Deployment** life cycle to permanently seal the device security with sha256 signature of the CMPA page. If the option remains unselected the security can be reconfigured. After you select the **Deployment** life cycle, click the **Build image** button in the **Build image** view. Then click the **Write image** button again and confirm the following message box:


Confirmation: IRREVERSIBLE OPERATION - Life Cycle to be set: Deployment (CMPA sealed)

Irreversible operation: the following life cycle state will be set: "Deployment (CMPA sealed)".
Description: Processor in deployment state, CMPA page sealed.

Please confirm (OK = confirm, CANCEL = abort)

Figure 53. Confirm write

If the write operation was successful, reset the board.

Note: It is necessary to completely erase the entire processor before returning from PRINCE encryption to a non-encrypted image.

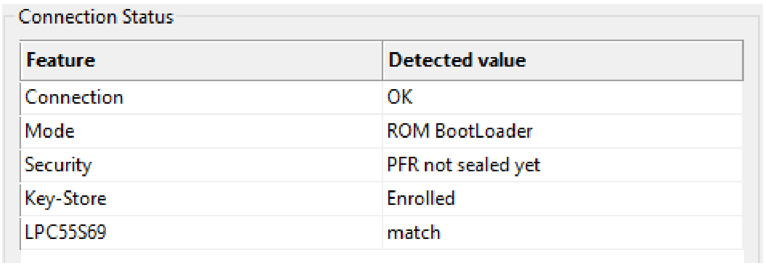
6.4.3.4 PFR and PUF KeyStore

This section provides information about PFR and PUF KeyStore.

PUF KeyStore initialization

SEC initializes KeyStore on LPC55Sxx devices only once in the device life cycle.

KeyStore enrollment status for the device is reported in the **Connection** dialog, using the **Test** button.


Connection Status

Feature	Detected value
Connection	OK
Mode	ROM BootLoader
Security	PFR not sealed yet
Key-Store	Enrolled
LPC55S69	match

Figure 54. KeyStore connection

It is possible to update the keys in the KeyStore, so it should not be needed to re-initialize KeyStore. In case of unexpected troubles, you can try to erase KeyStore, however, it is not recommended. With the KeyStore, it is recommended to also clear the CFPA page, as PRINCE IV fields in CFPA depend on the KeyStore. The device does not boot if you enroll the KeyStore and try to use it with previous IV fields.

How to erase KeyStore (example for LPC55S69)

```
bin/tools/blhost/win/blhost -u 0x1FC9,0x0021 -j -- set-property 29 1

bin/tools/blhost/win/blhost -u 0x1FC9,0x0021 -j -- write-memory 0x9E600
 zero_1536.bin
```

zero_1536.bin is a file that contains zeros, and the file size is 3*512 bytes.

How to update CFPA page (example for LPC55S69)

1. Increment the version in `cfpa.json`: it is recommended to add at least 0x10 from the last known version as the version is also incremented during PRINCE IV updates.
2. Run the following commands to update the CFPA page into the processor:

```
bin/tools/spsdk/pfr generate -c cfpa.json -o cfpa.bin
bin/tools/spsdk/blhost -u 0x1FC9,0x0021 -j -- write-memory 0x0009DE00
 cfpa.bin
```

DCFG_CC_SOCU problem on LPC55Sxx

LPC55S0x/1x/2x/6x processors do not support the `DCFG_CC_SOCU` field configured in CFPA, while `DCFG_CC_SOCU` in CMPA is zero (not configured yet) and the processor may stop work (lock) if the configuration happens. It is recommended to always configure CMPA first and erase CFPA (+reset) in case erasing of CMPA is needed.

6.5 LPC55(S)3x device workflow

6.5.1 Preparing source image for LPC55(S)3x devices

In this step, select the target memory where the image is executed. The following options are available for LPC55(S)3x devices:

- **Image running from internal FLASH** - XIP (eXecution In Place) image, which means that the image is executed directly from the memory where it is located.
  It is the default option for almost all SDK examples. There is no need to modify the default configuration, build the example as it is.
- **Image running from external FLASH** - XIP (eXecution In Place) image, which means that the image is executed directly from the memory where it is located.
  The image must start at address 0x8001000. The other locations are not supported now. There is no need to modify the default configuration.

For custom external FLASH, the configuration of external FLASH for booting can be adjusted in CMPA.

6.5.2 Connecting the board for LPC55(S)3x devices

This section contains information about configuring the evaluation board LPC55S36-EVK and connecting it to SEC.

1. Select ISP boot mode, see **Table: Boot mode selection for LPC55S36 EVK board** below.
2. Connect the J3 port to your PC with a USB cable.
3. Ensure SEC runs with a workspace created for the chosen device. For more information, see Setting up Secure Provisioning Tool.
4. Make sure that the boot memory in the toolbar matches NOR FLASH used on EVK board (for example, flex-spi-nor/ISxxxx) or internal flash.
5. Set the connection to USB and test the board connection.

6.5.2.1 Table: Boot mode selection for LPC55S36 EVK board

Board	In-System Programming (ISP) Boot	Boot from internal FLASH	Boot from external FLASH
LPC55S36-EVK	**J43:** 1-2 open, 3-4 closed	**J43:** 1-2 closed, 3-4 closed	**J43:** 1-2 closed, 3-4 open

6.5.3 Booting images for LPC55(S)3x devices

This section describes the building and writing of bootable images. For LPC55S3x, the SEC Tool supports XIP images only.

6.5.3.1 Booting plain unsigned or CRC image

A plain image is typically used for development. Start with this boot type before working with secured images to verify that the executable image works properly.

First, build a bootable image:

1. Make sure that you have selected the **Plain unsigned** or **Plain with CRC** boot type in the toolbar.
2. Switch to the **Build image** view.
3. Select image built in Preparing source image for LPC55(S)3x devices as a **Source executable image**.
4. If there is a binary image, set the start address to 0x0 for internal flash, or 0x8001000 for external flash.
5. If needed, open **Dual image boot** and configure.
6. Click the **Build image** button to build a bootable image. The result is a binary bootable image.

When the bootable image is built, upload it to the processor:

1. Make sure that the processor is in ISP mode.
2. Switch to the **Write image** view.
3. Click the **Write image** button.

If the write operation is successful, switch boot mode (see **Table: Boot mode selection for LPC55S36 EVK board** in Connecting the board for LPC55(S)3x devices) and reset the board.

6.5.3.2 Booting plain signed image

This section describes building and writing a plain signed image.

Build a bootable image:

1. Select the **Plain signed** boot type in the toolbar.
2. Switch to the **Build image** view.
3. Select image built in Preparing source image for LPC55(S)3x devices as a **Source executable image**.
4. For **Authentication key** select any key, for example ROT1: IMG1_1
5. Use random value for “CUST_MK_SK” and “OEM seed” symmetric keys.
6. If needed, open **Dual image boot** and configure.
7. Open the PFR configuration and on the CMPA page check, that the bit-field `SEC_BOOT_EN` in the `SECURE_BOOT_CFG` field is configured. It is necessary to select any type of image check.
8. Make sure that the board is connected and the processor is in ISP mode. During building processes, provisioning SB file for installation of `CUST_MK_SK` into processor is prepared.
   Note: The processor is reset after the SB file is built.
9. Keep **Develop** in life cycle
10. Click the **Build image** button to build a bootable image. The result is a binary bootable image and SB3 capsule for installation of the image into the processor.

When the bootable image has been successfully built, you can upload to the processor:

1. Make sure that the processor is in ISP mode.
2. Switch to the **Write image** view.
3. Click the **Write image** button.

6.5.3.3 Booting encrypted image

Encrypted images with CRC or signed images are supported. The process of creation an encrypted image is similar to a signed image. In addition, configure encrypted regions in the **Build image** view:

- Use the **PRINCE Regions** button to configure encrypted regions for internal FLASH
- Use the **IPED Regions** button to configure encrypted regions for external FLASH

In both cases, the whole image is encrypted by default. For clock limitations when using encrypted images, see documentation for the target processor. With the dual boot, set one region for image0 and one for image1. Setting a region only for image0 does not encrypt image1.

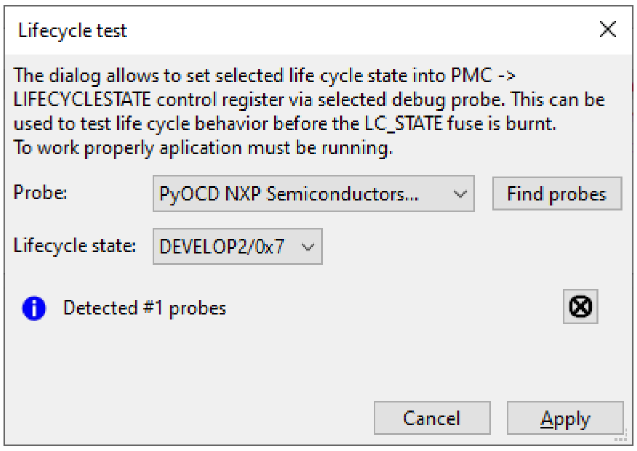
Image encryption is performed when the image is written to the target memory. The encrypted region is configured in the SB file. The decrypted regions are configured in the CMPA page, so make sure these two are aligned.

6.5.3.4 Test life cycle

To test processor behavior in the advanced life cycle, it is possible to temporarily change the life cycle to some higher level by setting control register PMC->LIFECYCLESTATE to the required level. This life-cycle state is valid until HW is reset.

Required steps:

1. Prepare the image and generate keys.
2. Set access control in SOCU registers.
3. Build the image
4. Execute write operation.
5. Run the application.
6. Connect a debug probe.
7. On the write tab, click the **Test life cycle** button and in the displayed dialog set the required life cycle state.
8. Click **Apply** to move the processor into the selected life cycle. Now, it is possible to test the processor behavior.



The dialog allows to set selected life cycle state into PMC -> LIFECYCLESTATE control register via selected debug probe. This can be used to test life cycle behavior before the LC_STATE fuse is burnt.
To work properly aplication must be running.

Probe: PyOCD NXP Semiconductors...

Lifecycle state: DEVELOP2/0x7

Detected #1 probes

Figure 55. Lifecycle test

6.5.3.5 Life cycle for LPC55(S)3x devices

The default life cycle, which should be used for development, is **Develop**. Before you deploy the application, set the “In Field” or “In Field Locked” life cycle (see documentation for the target processor for detailed description).

Note: Change of the life cycle is irreversible.

When changing to **In Field** life cycle, CMPA and CFPA pages are installed in the `dev_hsm_provi.sb` file. It is supposed that in this mode, the pages are installed into an empty processor, so there are not any failures (the page update may fail, so in development mode, these pages are updated in write script, where the progress and error report are much better). Once the processor is in In Field state, the SEC Tool supports only update of the application image; updates of CMPA and CFPA are not supported.

6.6 LPC865 workflow

This section describes the LPC865 device workflow in detail.

6.6.1 Preparing images for build for LPC865

The processor supports only images executed from internal flash (XIP). Use the default toolchain settings to build the application image.

6.6.2 Connecting the LPCXpresso860-MAX board for LPC865

This section contains information about configuring the evaluation board LPCXpresso860-MAX and connecting it to the SEC Tool.

1. To power the board, connect the J4 USB port with the USB cable to your power adapter or PC.
2. Connect pins 2(RX) and 4(TX) at the J2 connector to the UART converter and connect to your PC.
3. To switch the processor into ISP mode, hold the ISP(SW1) button and press RESET(SW3).
4. Ensure that the SEC Tool is already running with a workspace created for the chosen device. For more information, see Setting up Secure Provisioning Tool.
5. Open the **Connection** dialog and test the board connection. Mind the first board synchronization take a longer time.

6.6.3 Booting images for LPC865

This chapter describes the building and writing of plain bootable image.

6.6.3.1 Booting unsigned plain image

Plain unsigned is the only boot mode supported by this processor.

First, build a bootable image:

1. Switch to the **Build image** view.
2. Select an image built in Preparing source image for MC56F818xx/7xx/6xx and MWCT2xD2/12 devices as a **Source executable image**.
3. If there is a binary image, set the start address to 0x0.
4. Click the **Build image** button to build a bootable image. During this operation, the following changes are applied to the image: life cycle (=code read protection) and CRC.

When the bootable image is built, upload it to the processor:

1. Make sure that the processor is in ISP mode.
2. Switch to the **Write image** view.
3. Click the **Write image** button.
4. If the write operation is successful, reset the board. The processor will boot after several seconds.

6.6.4 Life cycles

The processor does not support real-life cycles, so the code read protection can be configured instead of life cycle.

6.7 MC56F818xx/7xx/6xx and MWCT2xD2/12 devices workflow

This chapter describes workflow for MC56F818xx/7xx/6xx and MWCT2xD2/12 processors.

6.7.1 Preparing source image for MC56F818xx/7xx/6xx and MWCT2xD2/12 devices

In this step, select the target memory where the image is executed. The following options are available for MC56F818xx/7xx/6xx and MWCT2xD2/12 devices:

- Image running from an internal flash - XIP (eXecution In Place) image, which means that the image is executed directly from the internal flash memory. There is no need to modify the default configuration, build the MCUXpresso SDK example as it is.

6.7.2 Connecting the board for MC56F818xx/7xx/6xx and MWCT2xD2/12 devices

This section contains information about configuring the evaluation boards and connecting it to SEC:

- MC56F81868-EVK
- MC56F81000-EVK
- WCT-QI2

1. Power the board from USB_TYPE_C port (WCT-QI2).
2. Connect the J4 1-TX, 3-RX, 5-GND port (WCT-QI2) via the MCU-Link to your PC.
3. In the **Connection** dialog, set the connection to **UART**
4. To get to ISP boot mode, send a blhost command to the processor within 5 seconds after the board startup, for example click the **Test connection** button in the **Connection** dialog.

Note: The processor boots from internal flash when there is no blhost command sent within 5 seconds after the startup.

6.7.3 Booting images for MC56F818xx/7xx/6xx and MWCT2xD2/12 devices

This chapter describes the building and writing of plain and signed bootable images.

6.7.3.1 Booting plain unsigned or CRC image

Plain images are typically used for development. Start with this boot type before working with secured images to verify that the executable image works properly.

First, build a bootable image:

1. Make sure that you have selected the **Plain unsigned** or **Plain with CRC** boot type in the toolbar.
2. Switch to the **Build image** view.
3. Select an image built in Preparing source image for MC56F818xx/7xx/6xx and MWCT2xD2/12 devices as a **Source executable image**.
4. If there is a binary image, set the start address to 0x0.
5. If needed, open **BCA/FCF** configure.
6. Click the **Build image** button to build a bootable image. The result is a binary bootable image. For secure processors there are also separated boot headers binary (BCA, FCF) and binary with application. This is due to the DUKB area in the internal flash that must stay untouched during the write.

When the bootable image is built, upload it to the processor:

1. Make sure that the processor is in ISP mode.
2. Switch to the **Write image** view.
3. Click the **Write image** button.
4. If the write operation is successful, reset the board. The processor will boot after 5 seconds.

6.7.3.2 Booting plain signed image

This section describes building and writing a plain signed image.

Build a bootable image:

1. Select the **Plain signed** boot type in the toolbar.
2. Switch to the **Build image** view.
3. Select an image built in Preparing source image for MC56F818xx/7xx/6xx and MWCT2xD2/12 devices as a **Source executable image**.
4. For **Authentication key**, select ROT1.
5. Make sure that the board is connected and the processor is in ISP mode. During building processes, an SBx file is prepared. The processor is used as HSM to create an encrypted SBx file.
6. Keep **OEM Open** in the life cycle. See Life cycle and device HSM trust provisioning for MC56F818xx/7xx/6xx and MWCT2xD2/12 devices for **OEM Closed** life cycle.
7. Click the **Build image** button to build a bootable image. The result is a binary bootable image, boot headers binary (BCA, FCF), and SBx capsule for installation of the image into the processor.

When the bootable image has been successfully built, you can upload to the processor:

1. Make sure that the processor is in ISP mode.
2. Switch to the **Write image** view.
3. Click the **Write image** button.

6.7.3.3 Life cycle and device HSM trust provisioning for MC56F818xx/7xx/6xx and MWCT2xD2/12 devices

The default life cycle used for development is **OEM Open**. Before you deploy the application, set the **OEM Closed** life cycle (see documentation for the target processor for a detailed description).

When the **OEM Closed** life cycle and **Device HSM** are set in the toolbar, the trust provisioning SBx file is created during the build. The `IFR ISK_CERT_HASH` fields are written to IFR by the `dev_hsm_provi.sbx` file. It is supposed that in this mode, the IFR fields have default values in the processor.

Warning:

- Writing of IFR fields is irreversible.
- WPC provisioning must be done before advancing the life cycle. Once the board was moved into the advanced life cycle, the provisioning command is not available.
- It is possible to reset the life cycle into an OEM Open state if needed. There are two possibilities:
  - Keep the Plain signed boot type, set the OEM Open life cycle, disable trust provisioning and do build and write. With this, the application is updated and the life cycle is set to OEM Open. Fields in IFR remain unchanged.
  - Set Plain unsigned boot type, OEM Open life cycle, disable trust provisioning and do build and write. For some devices, it may be necessary to disable flash security before executing the write operation (to enable the FlashEraseAllUnsecure command). With this, the entire flash memory is erased and the life cycle is set to OEM Open. Fields in IFR remain unchanged.
- The blhost flash-erase-all command erases the entire memory, which also affects the life cycle state. After flash-erase-all and device reset, the life cycle is in the Closed state.

Table: Life cycle and device HSM trust provisioning

	ISK_CERT_HASH	SBx file
OEM Open lifecycle, Signed boot type	Included in FCB	SBx used to write/update the application image
OEM Closed life cycle, Signed boot type	Irreversibly written to IFR in trust provisioning SBx file	SBx used to write/update the application image

6.7.4 EdgeLock 2GO WPC workflow

The EdgeLock 2GO platform provides provisioning of the Wireless Power Consortium (WPC) Certificate chain needed for WPC Qi authentication. The SEC Tool supports a WPC provisioning scenario where NXP serves as WPC Manufacturing CA Service Provider. For the workflow, it is supposed the NXP example project is used. Contact NXP sales to get the example.

6.7.4.1 EdgeLock 2GO WPC flow, step by step

To configure WPC Certificate provisioning, do the following steps:

1. Turn on secure boot mode and verify that the application works properly.
2. On the EdgeLock 2GO server, create Qi *Manufacturer CAs* and *Product Unit Certificates*
3. Create an API key. For details, see the API key to access the EdgeLock 2GO server.
4. In the **main menu > Target > Trust Provisioning Mode**, check the **enable WPC** and fill the EdgeLock2GO parameters; use Qi ID for Qi PUC Public Key.
5. Upload the unsigned ISK block binary `<workspace>/keys/ROT1_cert_block_wpc_signing.bin` to EdgeLock 2GO server for signing. After signing, download the signed block and save it as `<workspace>/keys/isk_nxp_signed.bin`.

6. On the **write** tab, click the **Extract WPC certificate** button to obtain the manufacturer CA certificate in form of C array `<workspace>/el2go/mfg_ca_cert_array.c`. The board must be connected in ISP mode.
7. Use this array to replace WpcMfgCaCert[] in `systemAuthentication.c`; certificate is placed in memory at address 0x600-0x7FF. It is expected that the application contains the WPC root CA cert hash at address 0x5E0-0x5FF.
8. Rebuild the project in Codewarrior.
9. With an updated source file, do the build operation in the SEC Tool.
10. Write the image into the board. As part of the write operation, the board is provisioned with the WPC PU certificate.

6.8 MCX A1/A2/A3/L2 device workflow

This chapter describes workflow for A13x/A14x/A15x/A16x/A17x/A25x/A26x/A35x/36x/L25x processors.

6.8.1 Preparing source image for MCX A1/A2/A3/L2 devices

- Image running from internal FLASH is the default option for almost all SDK examples. There is no need to modify the default configuration, build the example as it is.
- Image running from internal RAM; when creating this example, select the link application to RAM. (not supported for MCX L2)

6.8.2 Connecting the board

This section contains information about configuring the boards in the table below connecting to the SEC Tool.

Table: Boot mode of the MCXA/MCXL boards

Board	In-System Programming (ISP) Boot	Boot from internal FLASH
FRDM-MCXA153	SW2/JP8	By default
FRDM-MCXA156	SW3	By default
FRDM-MCXA266	SW3	By default
FRDM-MCXA346	SW3	By default
FRDM-MCXA366	SW3	By default
FRDM-MCXL255	SW3	By default

To connect the board, follow the steps below:

1. Select ISP boot mode, see the table above.
2. Connect the UART/USB port to your PC with a USB cable.
3. Ensure SEC runs with a workspace created for the chosen device. For more information, see Setting up Secure Provisioning Tool.
4. Go to **main menu > Target > Connection**, select UART/USB and test the connection.

6.8.3 Booting images MCX A1/A2/A3/L2 devices

Booting plain unsigned and plain with CRC image is the same as for the MCXN devices. For details, see Booting plain or CRC image for MCX Nx4x/N23x devices.

6.8.3.1 Setting CMPA flash access control

The CMPA configuration includes flash access control fields labeled `FLASH_ACL_#_#`. These fields are used to initialize the Memory Block Checker (MBC) configuration registers. The values defined in CMPA are loaded during boot and can be modified later, if they are not locked.

Processors are shipped with an empty flash memory, meaning the CMPA (Customer Manufacturing Programming Area) page is initially invalid. In this default state, the flash is fully accessible—allowing read, write, and execute operations.

Once CMPA is configured on the chip, access permissions must be explicitly defined based on the intended use case. Improperly set access rights may cause the application to malfunction.

By default, the SEC Tool sets access permissions to “read + execute.” However, when configuring the application image, the SEC Tool does not accept this default setting. Instead, it requires the user to explicitly define the access rights, ensuring that the configuration is intentional and verified.

It is important to note that any other memory region that may be accessed during runtime must be configured with the correct access rights. Failure to do so can result in access violations or runtime errors.

6.8.3.2 Provisioning with device HSM for MCXA25x/MCXA26x/MCXA36x

This section describes building and writing a plain and plain with CRC image provisioned by the device HSM sb file.

Build a bootable image:

1. Select the desired boot type in the toolbar.
2. Switch to the **Build image** view.
3. Select an image as a **Source executable image**.
4. Use a random value for “OEM seed” symmetric keys.
5. If needed, open **Dual image boot** and configure.
6. Ensure that the board is properly connected and the processor is in ISP mode before starting the build process. During the build, a provisioning SB3 file is prepared. The provisioning file contains the CMPA, an executable image, and additional images. If no board is connected, the build will fail when preparing the provisioning SB3 file.
7. Click the **Build Image** button to generate a bootable image. This process creates a device HSM SB3 capsule for installation of the image into the processor.

Note: In any advanced ROP state, CMPA is configured to remove the Secure Installer (SI), since SI is not supported in the advanced life cycle.

When the bootable image is successfully built, upload it to the processor:

1. Make sure that the processor is in ISP mode.
2. Switch to the **Write image** view.
3. Click the **Write image** button.

6.8.4 Life cycle for MCX A1/A2/A3/L2 devices

The default life cycle that should be used for development is **OEM Open**. Before you deploy the application, set the “In Field ROP 1”, “In Field ROP 2”, or “In Field ROP 3” life cycle (see the documentation for the target processor for a detailed description). For devices that have CMPA in the internal flash, it is possible to revert the life cycle by executing a mass erase command.

Note: In Field ROP 3, the life cycle cannot be reverted, ISP commands are not available.

6.8.4.1 Reverting life cycle

To revert the life cycle, execute the mass erase command.

For devices with Secure Installer (SI), the SI firmware must be removed first, otherwise the mass erase command fails. This is because the memory region containing the SI is protected and cannot be accessed. Once the SI is erased and the ERASE_TOKEN[x] registers are set in the CMPA, flash protection is disabled. **Note:** The mass erase operation also erases the CMPA and the erase token registers. Therefore, save the erase token value before executing the mass erase, so it can be restored afterward to regain access to the flash region where the SI was located.

6.8.5 MCUboot for MCX A1/A2/A3/L2 devices

MCUboot is described in MCUboot workflow. Below are MCX-specific comments:

- The flash access field settings in CMPA, where MCUboot is managing the application, must be set to any “unlocked” rule. An unlocked rule must be set because MCUboot modifies the region rule by setting it to a “write” rule during the application updates and to an “execute” rule when running the application.

6.9 MCX C041/C242/C444 device workflow

This chapter describes the workflow for Cx4x processors.

6.9.1 Preparing source image for MCX C041/C242/C444 devices

- An image running from the internal FLASH is the default option for almost all SDK examples. There is no need to modify the default configuration, build the example as it is.
- An image runs from internal RAM when this example is created. Select the link application to RAM. The option is not supported on MCXC041.

6.9.2 Connecting the board for MCX C041/C242/C444 devices

This section contains information about configuring the evaluation boards FRDM-MCXC041, FRDM-MCXC242, FRDM-MCXC444, and connecting them to SEC.

Table: Boot mode of the MCXC boards

Board	In-System Programming (ISP) Boot	Boot from internal FLASH
FRDM-MCXC041	SW3	By default
FRDM-MCXC242	SW3	By default
FRDM-MCXC444	SW3	By default

1. Select ISP boot mode. For details, see the table above.
2. Connect the UART/USB port to your PC with a USB cable.
3. Ensure SEC runs with a workspace created for the chosen device. For more information, see Setting up Secure Provisioning Tool.
4. Go to **main menu > Target > Connection**, select UART and test the connection.

6.9.3 Booting images for MCX C041/C242/C444 devices

This section describes building and writing of bootable images into the internal flash and booting.

6.9.3.1 Booting plain unsigned image

First, build a bootable image:

1. Switch to the **Build image** view.
2. Select an image built in Preparing source image for LPC55(S)0x/1x/2x/6x devices as a **Source executable image**.
3. If there is a binary image, set the start address to 0x00000000.
4. If needed, open **BCA/FCF** configure.
5. To build a bootable image, click the **Build image** button. The result is a binary bootable image.

When the bootable image is built, upload it to the processor:

1. Make sure that the processor is in the ISP mode.
2. Switch to the **Write image** view.
3. Click the **Write image** button.
4. If the write operation is successful, reset the board to boot the image.

6.9.3.2 Life cycle for MCX C041/C242/C444 devices

The default life cycle that should be used for development is **Flash unsecured**. Before deploying the application, set the **Flash secured** (see documentation for the target processor for a detailed description). When switching the life cycle, the best practice is to set BCA and FCF pages. Chip with the flash security enabled can be erased, if ISP is available and the FCF field ‘mass erase’ is enabled.

6.9.3.3 Backdoor key

The advanced life cycle can be temporarily disabled if the backdoor key was set and the backdoor key comparison is enabled in FCF. After the backdoor key verification, the processor behaves as if in the OEM Open life cycle until the next reset. If the verification fails, further verification is not possible until flash reset. Flash security can be disabled using the ‘Disable Flash Security’ dialog available under the Write tab.

6.10 MCX E24 device workflow

This chapter describes the workflow for MCX E24x processors.

6.10.1 Preparing source image for MCX E24x devices

An image running from the internal flash is the default option for almost all SDK examples. There is no need to modify the default configuration, build the example as it is.

Internal flash is the only supported boot memory for MCX E24x devices.

6.10.2 Connecting the board for MCX E24x devices

This section contains information about configuring the evaluation board FRDM-MCXE247 and connecting it to SEC.

6.10.2.1 Flashloader boot behavior on MCX E24x devices

MCX E24x devices do not support ISP boot mode. Instead, they are factory-preprogrammed with a flashloader image stored in the internal flash that includes both the flashloader loader and the flashloader firmware.

By default, these devices boot from internal flash memory. During the boot process, the preprogrammed flashloader is automatically loaded into RAM and executed from there. This enables flash programming functionality without requiring a separate ISP mode.

6.10.2.2 Flashloader initialization via debug probe

If the flashloader is not running on the device, it can be loaded and executed from RAM using a debug probe. The onboard MCU-Link on the FRDM-MCXE247 supports debug probe functionality and can therefore be used for this purpose.

Note: Flashloader can be initialized via MCU-Link or J-Link debug probe.

Note: SEC Tool scripts that use the debug probe host software (LinkServer or JLink) expect the host software's installation directory to be included in the `PATH` environment variable.

Select the debug probe either in **main menu > Target > Debug Probe** or in the **Toolbar**, **Dbg**.

6.10.2.3 FRDM-MCXE247 UART pin connections

To establish communication between the preprogrammed flashloader (which uses LPUART1) and the onboard MCU-Link (which uses LPUART2), connect the MikroBUS UART to the Arduino UART using two Dupont wires as follows:

- Connect Arduino J1 - Pin 2 (PTD17/LPUART2_RX-Arduino_D0) to MikroBUS J5 - Pin 3 (PTC8/LPUART1_RX-MIKROE)
- Connect Arduino J1 - Pin 4 (PTE12/LPUART2_TX-Arduino_D1) to MikroBUS J5 - Pin 4 (PTC9/LPUART1_TX-MIKROE)

With this connection, the onboard MCU-Link's USB-to-UART bridge functionality can be used to enable communication between the host PC and the flashloader via UART.

6.10.2.4 FRDM-MCXE247 connection via SPI

To establish the SPI communication between the preprogrammed flashloader and an external USB-to-SPI bridge (for example, MCU-Link Pro Debug Probe), connect the bridge to the target board as follows:

- SPI MOSI to MicroBUS J6 - Pin 5 (PTB3/LPSPI0_SIN-MIKROE)
- SPI SCK to Arduino J3 - Pin 1 (PTB2/FTM1_QD_PHB-MC_ENC_B)
- SPI MISO to IO Expander J8 - Pin 12 (PTB1)
- SPI PCS0 to IO Expander J8 - Pin 9 (PTB0)

With this connection, the external USB-to-SPI bridge can be used to enable communication between the host PC and the flashloader via SPI.

6.10.2.5 Connecting the board to host PC

1. Connect the J13 (MCU-Link USB) port to your PC with a USB cable.
2. Ensure SEC runs with a workspace created for the chosen device. For more information, see Setting up Secure Provisioning Tool.
3. Go to **main menu > Target > Connection**, select UART and test the connection.

Note: When the **Test Connection** button is clicked, the flashloader is initialized via debug probe if it is not already running. In this case, a debug probe must be selected, see Flashloader initialization via debug probe.

6.10.3 Booting images for MCX E24x devices

This section describes building and writing of bootable images into the internal flash and booting.

Note: At this moment only the **Unsigned** boot is supported in the SEC Tool for MCX E24x.

6.10.3.1 Booting plain unsigned image - unsecure boot

This section describes image building and writing for unsecure boot.

First, build a bootable image:

1. Make sure that you have selected the **Unsigned** boot type in the toolbar.
2. Switch to the **Build image** view.
3. Select an image built in Preparing source image for MCX E24x devices as a **Source executable image**.
4. If there is a binary image, set the start address to 0x00000000.
5. If needed, open **FCF configuration** and configure.
6. If needed, open **SHE keys configuration** and configure, see SHE keys configuration.
7. To build a bootable image, click the **Build image** button. The result is a binary bootable image.

When the bootable image is built, upload it to the processor:

1. Make sure that the processor is connected.
2. Switch to the **Write image** view.
3. Click the **Write image** button.
4. If the write operation is successful, reset the board to boot the image.

6.10.3.2 Booting plain signed image - secure boot

This section describes how to build and write a bootable image for secure boot. During secure boot, the bootable image code section is authenticated, and the generated MAC is compared with a value previously stored in the secure key storage (BOOT_MAC).

First, build a bootable image:

1. Make sure that you have selected the **Authenticated (SHE) serial** or **Authenticated (SHE) parallel** or **Authenticated (SHE) strict** boot type in the toolbar. For more information about secure boot types, see the boot type tooltip.
2. Switch to the **Build image** view.
3. Select an image built in Preparing source image for MCX E24x devices as a **Source executable image**.
4. If there is a binary image, set the start address to 0x00000000.
5. If needed, open **FCF configuration** and configure.
6. Open **SHE keys configuration** and configure, see SHE keys configuration.
   - Make sure BOOT_MAC_KEY is configured to enable secure boot.
   - For details on BOOT_MAC, see Manual and automatic BOOT_MAC calculation.
7. To build a bootable image, click the **Build image** button.
   - This generates a binary bootable image.
   - If enabled, BOOT_MAC is calculated.
   - Binaries for loading SHE keys are also created.

When the bootable image is built, upload it to the processor:

1. Make sure that the processor is connected.
2. Switch to the **Write image** view.
3. Click the **Write image** button.

4. If the write operation is successful, reset the board to boot the image.

Note: The **Authenticated (SHE) strict** boot mode is permanent once set and cannot be changed. Additionally, automatic BOOT_MAC calculation is not supported in this mode. Before setting this boot mode, BOOT_MAC must be calculated and stored. Failure to do so will cause the device to remain in the RESET state and prevent it from booting.

6.10.3.3 SHE keys configuration

In the **SHE keys configuration** dialog, you can configure the processor for CSEC/SHE secure operations by setting the key store size. This determines how flash memory is partitioned.

Configuring the number of keys to **No setup** results in skipping the CSEC/SHE key store setup, so flash partitioning is not performed. This selection is intended for development only.

Up to 20 keys can be configured. The first three keys have a dedicated use.

- MASTER_ECU_KEY is used to reset the CSEC/SHE to factory state or to modify any other keys.
- BOOT_MAC_KEY is used during the secure boot process to verify the authenticity of the software.
- BOOT_MAC stores the MAC value of the application image used in the secure boot process.
- USER_KEY_1 to USER_KEY_17 user keys are available for application-specific usage.

Note: If the **Write Protection** flag is set on any key, the device cannot be reset to its factory state.

6.10.3.3.1 Manual and automatic BOOT_MAC calculation

The BOOT_MAC can be calculated and programmed in two ways:

- Manually: BOOT_MAC is calculated offline using the `nxpshe calc-boot-mac` command and programmed using the `blhost key-provisioning set_user_key` command. To use this option, enable BOOT_MAC in the **SHE keys configuration** dialog in the **Build image** view. For information on updating BOOT_MAC, see SHE key update.
- Automatically using CSEC: This option should only be used if BOOT_MAC has not yet been programmed in the CSEC/SHE key store. To enable automatic calculation, disable BOOT_MAC in the **SHE keys configuration** dialog in the **Build image** view.

6.10.3.3.2 SHE key update

A key that has already been set in the CSEC/SHE key storage can be updated, provided the **Write Protection** flag is not enabled for that key. To perform a successful update, you must increase the **Counter** value for the key in the **SHE Keys Configuration** dialog.

In secure mode, when BOOT_MAC is calculated manually offline, a BOOT_MAC update is always required whenever the **Source executable image** is changed.

6.10.3.4 Life cycle for MCX E24x devices

The default life cycle that should be used for development is **Flash unsecured**. Before deploying the application, set the **Flash secured** (see documentation for the target processor for a detailed description). When switching the life cycle, the best practice is to set the FCF page.

6.10.3.5 Revert SHE keys configuration and/or flash security

The following table provides information on whether it is possible, for a specific processor configuration, to reset the SHE key store to its factory state (that is, remove partitioning) and to disable flash security by mass erase of the flash.

Life cycle	SHE keys configuration	Write protection flag	Can reset SHE key store?	Can disable flash security?
Flash unsecured	Key store size set to 0	N/A	Yes	Yes
Flash unsecured	Key store present	None	Yes	Yes
Flash unsecured	Key store present	Any key protected	**No**	**No**
Flash secured	Key store size set to 0	N/A	**No**	Yes
Flash secured	Key store present	None	**No**	**No**
Flash secured	Key store present	Any key protected	**No**	**No**

6.10.3.5.1 Reset SHE key store

You can reset the CSEC/SHE key store to factory state, provided no keys are write-protected and the life cycle is Flash Unsecure. If the life cycle is set to Flash Secure, only the **Disable flash security…** is available, which triggers a mass erase of the flash memory.

To reset the key store:

1. Open the **Reset SHE key store** dialog via the **Reset SHE key store…** button in the **Write image** view.
2. Select the reset method based on the current key storage setup on the processor.
   - Key store present - Key storage size configured to 5, 10 or 20. In this case `nxpshe reset` command is used. Only the SHE key storage is reset and the application image remains in flash.
   - No key store - Key storage size configured to 0. In this case `blhost flash-erase-all-unsecure` command is used so the complete flash memory is erased and the flash security section is recovered.
3. Click the **Reset** button.

6.10.3.5.2 Disable flash security via debug probe

A secured device can be unsecured using a debug probe, if the following conditions are met:

- The ‘mass erase’ field in the FCF configuration is not disabled.
- The SHE key store is not configured (number of keys is set to **No setup** or **0**). When the device is unsecured, a mass erase of the internal flash is automatically triggered.

Note: The mass erase operation does not affect the key storage setup.

To unsecure the device, click the **Disable flash security…** button in the **Write image** view. After the mass erase, the flashloader image is written to the internal flash and started.

6.11 MCX Nx4x/N23x device workflow

This chapter describes the workflow for Nx4x/N23x processors.

6.11.1 Preparing source image for MCX Nx4x/N23x devices

- Image running from internal FLASH is the default option for almost all SDK examples. There is no need to modify the default configuration, build the example as is.
- Image running from external FLASH must start at address 0x80001000, edit this address when creating an example in MCUXpressoIDE in advance setting.
- Image running from internal RAM when creating this example select link application to RAM.

6.11.2 Connecting the board for MCX Nx4x/N23x devices

This section contains information about configuring the evaluation boards FRDM-MCXN947, MCX-N9XX-EVK, MCX-N5XX-EVK, and connecting it to SEC.

Table: Boot mode of the MCXN boards

Board	In-System Programming (ISP) Boot	Boot from external FLASH	Boot from internal FLASH
MCX-N9XX-EVK	SW3/JP49	Boot source defined in CMPA	
FRDM-MCXN947	SW3	Boot source defined in CMPA	
MCX-N5XX-EVK	SW3/JP49	Boot source defined in CMPA	
FRDM-MCXN236	SW3	N/A	By default

1. Select ISP boot mode, see the table above.
2. Connect the UART/USB port to your PC with a USB cable.
3. Ensure SEC runs with a workspace created for the chosen device. For more information, see Setting up Secure Provisioning Tool.
4. Go to **main menu > Target > Connection** and select UART and test the connection.

6.11.3 Booting images for MCX Nx4x/N23x devices

This section describes building and writing of bootable images into the internal flash and booting. Booting from the external flash is similar, but the image linked to the external flash is used.

6.11.3.1 Booting plain or CRC image for MCX Nx4x/N23x devices

Plain images are typically used for development. Start with this boot type before working with secured images to verify that the executable image works properly.

First, build a bootable image:

1. Make sure that you have selected the **Plain unsigned** or **Plain with CRC** boot type in the toolbar.
2. Switch to the **Build image** view.
3. Select an image built in Preparing source image for LPC55(S)0x/1x/2x/6x devices as a **Source executable image**.
4. If there is a binary image, set the start address to 0x00000000 (for external flash 0x80001000).
5. If needed, open **Dual image boot** and configure.
6. Click the **Build image** button to build a bootable image. The result is a binary bootable image.

When the bootable image is built, upload it to the processor:

1. Make sure that the processor is in ISP mode.
2. Switch to the **Write image** view.
3. Click the **Write image** button.
4. If the write operation is successful, remove the ISP jumper and reset the board to boot the image.

6.11.3.2 Booting plain signed image

This section describes building and writing a plain signed image.

Build a bootable image:

1. Select the **Plain signed** boot type in the toolbar.

2. Switch to the **Build image** view.
3. Select an image built in Preparing source image for LPC55(S)0x/1x/2x/6x devices as a **Source executable image**.
4. Generate keys on PKI tab.
5. Back on the **Build image** view and select any Authentication key from the drop-down menu, for example ROT1: IMG1_1.
6. Use random value for "CUST_MK_SK" and "OEM seed" symmetric keys.
7. If needed, open **Dual image boot** and configure.
8. Make sure that the board is connected and the processor is in ISP mode. During building processes, a provisioning SB3 file for installation of CUST_MK_SK into the processor is prepared. If no board is connected, the build will fail when preparing the provisioning SB3 file. But other build processes were completed.
   Note: The processor is reset after the SB file is built.
9. Click the **Build image** button to build a bootable image. The result is a binary bootable image and SB3 capsule for installation of the image into the processor.

When the bootable image has been successfully built, you can upload to the processor:

1. Make sure that the processor is in ISP mode.
2. Switch to the **Write image** view.
3. Click the **Write image** button.

6.11.3.3 Booting encrypted image

Encrypted images with CRC or signed images are supported. The process of creation an encrypted image is similar to a signed image. In addition, configure encrypted regions in the **Build image** view:

- Use the **PRINCE Regions** button to configure encrypted regions for internal FLASH
- Use the **IPED Regions** button to configure encrypted regions for external FLASH

In both cases, the image is encrypted by default. For clock limitations when using encrypted images, see documentation for the target processor. In combination with the dual boot, set one region for image0 and one for image1. Setting a region only for image0 does not encrypt image1.

Image encryption is performed when the image is written to the target memory. The encrypted region is configured in the SB file. The decrypted regions are configured in the CMPA page, so make sure these two are aligned.

6.11.3.4 Life cycle for MCX Nx4x/N23x devices

The default life cycle that should be used for development is **Develop**. Before you deploy the application, set the In Field or In Field Locked life cycle (see documentation for the target processor for a detailed description). When switching the lifecycle, the best practice is to burn the ROTKH and PRINCE or IPED region settings into fuses. ROTKH fuses are set by the tool automatically, but the PRINCE or IPED setting is up to the user to be set.

Note: Change of the life cycle is irreversible.

The life-cycle configuration also affects the way how the security assets are installed. In the development life cycle, some assets are installed in a write script so it is easier to debug the changes and problems.

The table below contains detailed information on how the security assets are installed based on the selected life cycle:

Table: Life cycle and trust provisioning

Trust provisioning Life cycle	Device HSM Develop	Device HSM In Field	EdgeLock 2GO Develop	EdgeLock 2GO In Field
CMPA	Device HSM SB file	Device HSM SB file	el2go_provi.sb3	el2go_provi.sb3
CUST MK SK	Device HSM SB file	Device HSM SB file	CUST-MK_SK	CUST-MK_SK
ROKTH in CMPA	Device HSM SB file	Device HSM SB file	RKTH	RKTH
CFPA	Write script	Device HSM SB file	Write script	el2go_provi.sb3
RKTH fuses	Not burnt unless requested by the user	Device HSM SB file	Not burnt unless requested by the user	el2go_provi.sb3
NPX and IPED fuses	Not burnt unless requested by the user	Device HSM SB file	Not burnt unless requested by the user	el2go_provi.sb3
Other custom fuses	Write script	Device HSM SB file	Write script by default; optionally el2go_provi. sb3	el2go_provi.sb3

In the **Develop** life cycle, the CFPA is written in the write script, so any failures can be easily detected. In the **In Field** life cycle, the provisioning is supposed to be done for an empty processor.

Once the processor is in In Field state, the write script still allows updating the application image; other updates are not supported by the write script but can be done using a custom SB file.

6.11.3.5 Test life cycle

MCX Nx4x/N23x variants can set the test life cycle on the CFPA page by setting the bit-field CFPA_LC_STATE and INV_CFPA_LC_STATE in the HEADER register. By writing this CFPA setting, MCU behaves as if the LF was moved in OTP.

Note: Use only the expected LC values, other values can brick the chip.

To move back from the advanced life cycle, set the bit-field CFPA_LC_STATE back to 0x0 and write it to the chip. If the LC is moved only in the CFPA, it is possible to rewrite the CFPA page after power-on reset by executing blhost -u 0x1fc9, 0x014f - write-memory 0x01000000 cfpa.bin.

Steps to advance LC and return:

1. Open or prepare a workspace that has a secure boot type (signed or encrypted)
2. In the **OTP/PFR configuration**, set CFPA_LC_STATE to 0xF and INV_CFPA_LC_STATE to 0xF0
3. Build and write image
4. Power-cycle the device into ISP
5. In OTP/PFR configuration, read the CFPA page
6. Reset the device by clicking the **Reset** button or using the blhost reset command
7. Test the secure life-cycle behavior (limited commands, debug rights based on the SOCU register)
8. After testing is done, power-cycle the device into the ISP
9. Open the **OTP/PFR configuration** and set CFPA_LC_STATE to 0x0 and INV_CFPA_LC_STATE to 0xFF
10. Enable advanced mode and write the CFPA page into the device
11. After reset, the device will be back in normal LC
    Note: For EVK variants for power cycle, follow these steps:
    a. Power off the board
    b. Add jumper to JP22
    c. Change the power supply to USB (J28)
    d. Hold the ISP button (or short the jumper for ISP) and connect the board via J28
    e. Remove jumper JP22

6.11.3.6 EdgeLock 2GO

EdgeLock 2GO is described in EdgeLock 2GO trust provisioning workflow. Below are MCXN-specific comments:

- The assets installed by EdgeLock 2GO are different for the Develop and In Field life cycle, see Life cycle for MCX Nx4x/N23x devices for details.
- There is the `ed2go_provi.sb3` file generated in the SEC Tool that contains burning of the additional fuses. For the EdgeLock 2GO firmware, this file is optional; however, the SEC Tool uses the file to install the fuses in the In Field life cycle.
  To add the file to the EdgeLock 2GO server via web portal, use the following steps:
  – Create a Secure Binary Object
  – Select Binary File
  – Assign any name
  – Set Object Identified (OID) to 0x7FFF817C
  – Select a nonconfidential file
  – In the policies, select NONE
- For development, use the Develop life cycle in the SEC Tool and the **Open** policy in configuration with secure objects on the EdgeLock 2GO server. With these settings, the life cycle is not changed and it still erases the processor. For production, use the In Field life cycle and the **Closed** policy. This does not apply to the SB3 file above where the policy should be always NONE.
- The CMPA page must be erased before provisioning firmware is started, so it can be properly provisioned. The el2go-host application clears CMPA with parameter `-clear`. In case the production is executed with new/empty processors, the parameter can be removed.
- Address for secure objects must be in internal flash, provisioning firmware erases the area before writing the secure objects

6.12 RT10xx/RT116x/RT117x device workflow

This section describes the RT10xx/RT116x/RT117x device workflow in detail.

6.12.1 Preparing source image for RT10xx/RT116x/RT117x devices

In this step, you must select the target memory where the image will be executed. The following options are available for RT10xx/RT116x/RT117x devices:

- **Image running from an external NOR flash**
  It is the so-called **XIP**(e**X**ecution **I**n **P**lace) image, which means the image is executed directly from the memory where it is located.
- **Image running in internal RAM**
  This image can be on an SD card/eMMC or in external flash (SPI NOR, SPI NAND, or SEMC NAND) and will be copied into internal RAM and executed from there during the boot.
- **Image running in SDRAM**
  This image can be located in an SD card or external flash (SPI NOR, SPI NAND, or SEMC NAND) and during boot will be copied into SDRAM and executed from there.

6.12.1.1 Image running from external NOR flash

- **MCUXpresso IDE**
  The **led_blinky** example is linked into external flash by default.
  1. Optionally go to **Project > Properties > C/C++ Build > Settings > MCU C Compiler > Preprocessor > Defined symbols** and set **XIP_BOOT_HEADER_ENABLE** to **0**. This step is now optional, because a bootable image can be used as an input on the build tab.

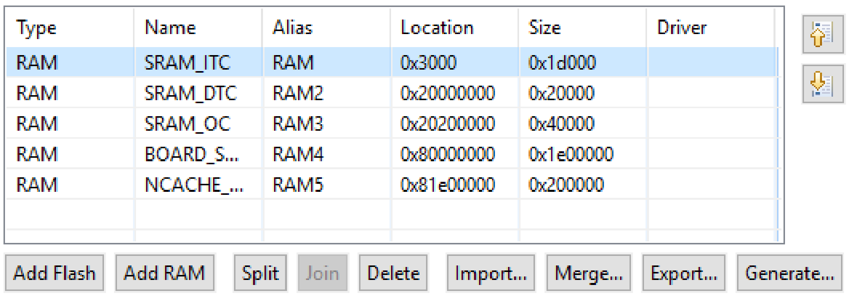
2. Build the image. You will find the resulting source image as `Debug\evkmimxrt10##\_iled_blinky.axf`. You can later use it as a **Source executable image** by SEC.

- **Keil MDK 5**
  1. In the toolbar, select **iled_blinky flexspi_nor_debug** target.
  2. Optionally, in **Project > Options > "*C/C++*"**, disable define symbol **XIP_BOOT_HEADER_ENABLE=0** (set to 0). This step is now optional, because a bootable image can be used as an input on the build tab.
  3. In **Project > Options > Linker**, remove all *-keep* options and the predefined symbol **XIP_BOOT_HEADER_ENABLE**. As a result, **Misc. controls** contains only *-remove*.
  4. Build the image. You will find the output image as `boards\evkmimxrt10\#\#\demo\_apps\led_blinky\mdk\flexspi_nor_debug\iled_blinky.hex`.
- **IAR Embedded Workbench**
  1. In **Project > Edit Configurations ...**, select **flexspi_nor_debug**
  2. Optionally, in Project **Options > C/C++ Compiler >Preprocessor>Defined Symbols**, add, or change the existing **XIP_BOOT_HEADER_ENABLE define** to **0**.This step is now optional, because a bootable image can be used as input on the build tab.
  3. On multicore processors set the Processor variant in **Project > Options... > General Options > Target**, for example *Cortex-M7* for *iled_blinky_cm7* on RT1176.
  4. Build the image. You will find the output image as `boards\evkmimxrt10##\demo_apps\led_blinky\iar\flexspi_nor_debug\iled\_blinky.out`.

6.12.1.2 Image running in internal RAM

Note: Memory addresses and sizes in this section are used as an example and depend on the selected processor.

- **MCUXpresso IDE**
  1. Select **Project > Properties - C/C++ Build > Settings > Tool Settings > MCU Linker > Managed Linker Script** and check **Link application to RAM**.
  2. In **Project > Properties > C/C++ Build > MCU settings**, delete **Flash**, and modify SRAM_ITC to start at 0x3000 with size 0x1D000.



Type	Name	Alias	Location	Size	Driver
RAM	SRAM_ITC	RAM	0x3000	0x1d000	
RAM	SRAM_DTC	RAM2	0x20000000	0x20000	
RAM	SRAM_OC	RAM3	0x20200000	0x40000	
RAM	BOARD_S...	RAM4	0x80000000	0x1e00000	
RAM	NCACHE_...	RAM5	0x81e00000	0x200000	

Figure 56. SRAM_ITC

  1. Move **SRAM_ITC** to the first position to make it default.
  2. Build the image. You can find the resulting source image named `Debug\\evkmimxrt10\#\#\_iled\_blinky.axf`.
- **Keil MDK 5**
  1. In the toolbar, select **iled_blinky** debug target.
  2. Open **Project > Options > Linker** and click **Edit** to edit the Scatter file.
  3. Close the window and make the following changes in the linker file (changes ***highlighted***):

```
#define m_interrupts_start 0x00003000
#define m_interrupts_size 0x00000400
```

```
#define m_text_start 0x00003400
#define m_text_size 0x0001DC00
```

4. Build the image.
   You can find the resulting image as `boards\\evkmimxrt10\#\#\\demo\_apps\\led\_blinky\\mdk\\debug\\iled\_blinky.hex`.

- **IAR Embedded Workbench**
  1. Select **Project < Edit Configurations … > Debug**.
  2. Open file *MIMXRT10##xxxxx_ram.icf* from project root folder and make the following changes:

```
define symbol m_interrupts_start = 0x00003000;
define symbol m_interrupts_end = 0x000033FF;

define symbol m_text_start = 0x00003400;
define symbol m_text_end = 0x0001FFFF;
```

  3. On multicore processors set the Processor variant in **Project > Options… > General Options > Target**, for example *Cortex-M7* for *iled_blinky_cm7* on RT1176.
  4. Save the changes and build the image.
     You can find the resulting image built as *boards\evkmimxrt10##\demo_apps\led_blinky\iar\debug\iled_blinky.out*.

6.12.1.3 Image running from external SDRAM

- **MCUXpresso IDE**
  1. Select **Project > Properties - C/C++ Build > Settings > Tool Settings > MCU Linker > Managed Linker Script** and check Link application to RAM.
  2. Select **Project > Properties - C/C++ Build > Settings > Tool Settings > MCU C Compiler > Preprocessor** and add defined symbol **SKIP_SYSCLK_INIT=1**.
  3. In **Project > Properties > C/C++ Build > MCU settings**, delete **Flash**, and modify BOARD_SDRAM to start at *0x80002000* with size *0x1dfe000*. Move BOARD_SDRAM to first position to make it default.
  4. Build the image. You can find the resulting source image named *Debug\evkmimxrt10##_iled_blinky.axf*.
- **Keil MDK 5**
  1. In the toolbar, select **iled_blinky sdram_debug** target.
  2. Open **Project > Options > Linker** and click **Edit** to edit the Scatter file.
  3. Close the window and make the following changes in the linker file (changes ***highlighted***):

```
#define m_interrupts_start 0x80002000
#define m_interrupts_size 0x00000400

#define m_text_start 0x80002400
#define m_text_size 0x0001DC00

#define m_data_start 0x80020000
#define m_data_size 0x01DE0000
```

  4. Build the image.
     You can find the resulting image as *boards\evkmimxrt10##\demo_apps\led_blinky\mdk\sdram_debug\iled_blinky.hex*.
- **IAR Embedded Workbench**
  1. Select **Project > Edit Configurations … > sdram_debug**.
  2. Open file *MIMXRT10##xxxxx_sdram.icf* from project root folder and make the following changes:

```
define symbol m_interrupts_start = 0x80002000;
define symbol m_interrupts_end = 0x800023FF;
define symbol m_text_start = 0x80002400;
define symbol m_text_end = 0x8001FFFF;

define symbol m_data_start = 0x80020000;
define symbol m_data_end = 0x8002FFFF;

define symbol m_data2_start = 0x80200000;
define symbol m_data2_end = 0x8023FFFF;

define symbol m_data3_start = 0x80300000;
define symbol m_data3_end = 0x81DFFFFF;

define symbol m_ncache_start = 0x81E00000;
define symbol m_ncache_end = 0x81FFFFFF;
```

3. On multicore processors set the Processor variant in **Project > Options… > General Options > Target**, for example *Cortex-M7* for *iled_blinky_cm7* on RT1176.
4. Save the changes and build the image. You can find the resulting image built as `boards\ \evkmimxrt10\#\#\\demo\_apps\\led\_blinky\\iar\\sdram\_debug\\iled\_blinky.out` .

6.12.2 Connecting the board for RT10xx/RT116x/RT117x devices

This section contains information about configuring the following evaluation boards and connecting them to SEC:

- MIMXRT1010-EVK
- MIMXRT1015-EVK
- MIMXRT1020-EVK
- MIMXRT1024-EVK
- MIMXRT1040-EVK
- MIMXRT1050-EVKB
- MIMXRT1060-EVK
- MIMXRT1064-EVK
- MIMXRT1160-EVK
- MIMXRT1170-EVKB

1. See **Table: Boot mode selection for RT1xxx EVK boards** in Connecting the board for RT10xx/RT116x/RT117x devices for instructions on how to set boot mode using DIP switches.
2. Make sure you have **J1** (J38 on RT1176, RT1166) set to **3-4** to power the board from USB OTG.
3. Connect to the J9 (J20 on RT1176, RT1166) port with the USB cable to your PC.
4. Ensure that SEC is already running with a workspace created for the chosen device. For more information, see Setting up Secure Provisioning Tool.
5. Make sure that the **Boot memory** in the toolbar matches the NOR flash used on the EVK board (for example*flex-spi-nor/ISxxxx*).
6. Set the connection to **USB** and test the board connection.

Booting from SD card

For booting from an SD card, do the following:

1. Insert a micro SDHC card into the board.
2. In the Secure Provisioning Tool, select **Boot memory: sdhc_sd_card/SDHC SD card 8 GB** in the **Toolbar**.

Table: Boot mode selection for RT1xxx EVK boards

Boot mode/ Device	Serial bootloader (ISP mode)	Flex-SPI NOR (QSPI, HyperFlash)	Flex-SPI NOR + Encrypted XIP (BEE/ OTFAD/IEE)	SD card	eMMC	SEMC NAND	FlexSPI NAND
RT1010-EVK	SW8: 0001	SW8: 0010	SW8: 0010	N/A	N/A	N/A	N/A
RT1015-EVK	SW8: 0001	SW8: 0010	SW8: 1010	N/A	N/A	N/A	N/A
RT1020-EVK	SW8: 0001	SW8: 0010	SW8: 1010	SW8: 0110	N/A	N/A	N/A
RT1024-EVK	SW8: 0001	SW8: 0010	SW8: 1010	SW8: 0110	N/A	N/A	N/A
RT1040-EVK	SW4: 0001	SW4: 0010	SW4: 0010 or 0110 SW2: 1000	SW4: 1010	N/A	N/A	N/A
RT1050-EVKB	SW7: 0001	SW7: 0110	SW7: 0010 or 0110 SW5: 1000	SW7: 1010	N/A	N/A	N/A
RT1060-EVK	SW7: 0001	SW7: 0010	SW7: 0010 or 0110 SW5: 1000	SW7: 1010	N/A	N/A	N/A
RT1064-EVK	SW7: 0001	SW7: 0010	SW7: 0010 or 0110 SW5: 1000	SW7: 1010	N/A	N/A	N/A
RT1160-EVK	SW1: 0001 SW2: 0000000000	SW1: 0010 SW2: 0000000000	SW1: 0010; SW2: 0100000000	SW1: 0010; SW2: 0000001000	SW1: 0010 SW2: 0000000100	SW1: 0010 SW2: 0000010000	N/A
RT1170-EVKB	SW1: 0001 SW2: 0000000000	SW1: 0010 SW2: 0000000000	SW1: 0010 SW2: 0100000000	SW1: 0010 SW2: 0000001000	SW1: 0010 SW2: 0000000100	SW1: 0010 SW2: 0000010000	N/A

6.12.3 Booting images for RT10xx/RT116x/RT117x devices

This section describes the building and writing of bootable images.

You can use several combinations of used memories:

Memory where the image is executed	Memory where the image is written	DCD/XMCD needed	XIP
External NOR flash	External NOR flash	No	Yes
Internal RAM	External NOR or NAND flash	No	No
Internal RAM	SD card or eMMC	No	No
SDRAM	External NOR or NAND flash	Yes	No
SDRAM	SD card or eMMC	Yes	No

Additional info:

- **Memory, where the image is executed** - Explained in Preparing source image for RT10xx/RT116x/RT117x devices.
- **Memory, where the image is written** - Configured as **Boot memory** in the SEC Tool.

Note: For RT116x/7x devices, two FlexSPI instances are supported for FlexSPI NOR/NAND boot devices. There is the `FLEXSPI_INSTANCE` fuse `\(BOOT_CFG2\[3\]\)` or GPIO boot pin that determines which FlexSPI instance to use. Set a corresponding GPIO boot pin to use Instance 2 without burning the fuse.

6.12.3.1 Booting unsigned image

An unsigned image is typically used for development. It is recommended to start with this boot type before working with secured images to verify that the executable image works properly.

First, build a bootable image:

1. Make sure that you have selected the **Unsigned boot type** in the **Toolbar**.
2. Switch to the **Build image** view.
3. Select the image built in Preparing source image for RT10xx/RT116x/RT117x devices as a **Source executable image**.
4. For images executed from SDRAM, configure SDRAM using DCD or XMCD (RT116x/7x). For EVK boards, the following DCD file can be used: `data\targets\MIMXRT1###/evkmimxrt1xxx_SDRAM_dcd.bin`. For RT116x/7x, the following XMCD configuration file can be used: `data\targets\MIMXRT11##/evkmimxrt11xx_xmcd_semc_sdram_simplified.yaml`.
   Note: For customization of DCD files, refer to Creating/Customizing DCD files.
5. If needed, open **Dual image boot** and configure (supported only for RT116x/7x and FlexSPI NOR).
6. Click the **Build image** button to build a bootable image.

When the bootable image has been successfully built:

1. Make sure that the board is in Serial Boot mode.
2. Switch to the **Write image** view.
3. Click the **Write image** button.

If the write operation was successful, switch boot mode (**Table: Boot mode selection for RT1xxx EVK boards** in Connecting the board for RT10xx/RT116x/RT117x devices) and reset the board.

6.12.3.2 Booting authenticated (HAB) image

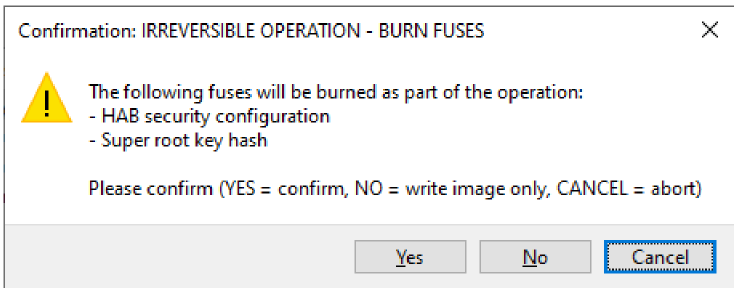
This section describes the building and writing of an authenticated image. If you want to use an encrypted image, you can skip this step.

1. In the **Toolbar** set the **Boot type** to **Authenticated (HAB)**.
2. In the **Build image** view, use the image from Preparing source image for RT10xx/RT116x/RT117x devices as a **Source executable image**.
3. For **Authentication key** select any key, for example, *SRK1: IMG1_1+CSF1_1*.
4. If needed, open **Dual image boot** and configure. (RT116x/7x - FlexSPI NOR)
5. Select the **HAB Closed** life cycle.
6. Click the **Build image** button.
7. Check that the bootable image was built successfully.

To write the image, switch to **Write image** view.

1. Make sure that the board is set to Serial bootloader (ISP) mode. See **Table: Boot mode selection for RT1xxx EVK boards** in Connecting the board for RT10xx/RT116x/RT117x devices and reset the board. for more information.
2. Click the **Write image** button.
3. In the following window, confirm to write fuses:
   - **Yes** - Continue writing the image and burning fuses.

  Note: Burning fuses can only be done once, after that the processor can only execute authenticated images.
- **No** - Do not burn fuses, continue writing the image.
- **Cancel** - Abort writing the image and burning fuses.


Confirmation: IRREVERSIBLE OPERATION - BURN FUSES

The following fuses will be burned as part of the operation:
- HAB security configuration
- Super root key hash

Please confirm (YES = confirm, NO = write image only, CANCEL = abort)

Figure 57. Burn fuses on i.MX RT1xxx for signed image

If the write operation was successful, switch boot mode (see **Table: Boot mode selection for RT1xxx EVK boards** in Connecting the board for RT10xx/RT116x/RT117x devices) and reset the board.) and reset the board.

6.12.3.3 Booting encrypted (HAB) authenticated image

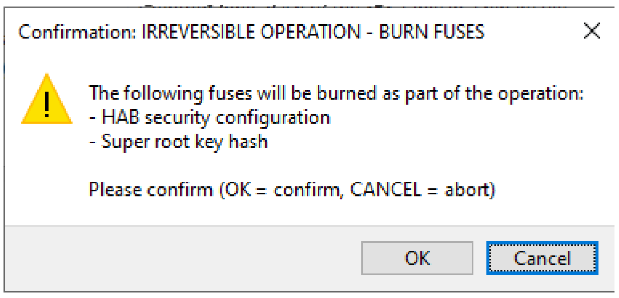
This section describes the building and writing of an encrypted image. This image will be decrypted into RAM during booting operation, so an XIP image cannot be used.

To build the image, do the following:

1. In the **Toolbar** set the **Boot type** to **Encrypted (HAB)** authenticated.
2. As a **Source executable image**, use the image from Preparing source image for RT10xx/RT116x/RT117x devices as a **Source executable image**.
3. For **Authentication key** select any key, for example, *SRK1: IMG1_1+CSF1_1*.
4. If needed, open **Dual image boot** and configure. (RT116x/7x - FlexSPI NOR)
5. Select the **HAB Closed** life cycle.
6. Click the **Build image** button.
7. Check that the bootable image was built successfully.

To write the image, do the following:

1. Switch to **Write image** view.
2. Make sure that the board is set to Serial bootloader (ISP) mode. See **Table: Boot mode selection for RT1xxx EVK boards** in Connecting the board for RT10xx/RT116x/RT117x devices for more information.
3. Click the **Write image** button.
4. In the following window, confirm to write fuses:
   - **OK** - Continue writing the image and burning fuses.
     Note: Burning fuses can only be done once, after that the processor can only execute authenticated images.
   - **Cancel** - Abort writing the image and burning fuses.



Confirmation: IRREVERSIBLE OPERATION - BURN FUSES

The following fuses will be burned as part of the operation:
- HAB security configuration
- Super root key hash

Please confirm (OK = confirm, CANCEL = abort)

Figure 58. Burn fuses on i.MX RT1xxx for HAB encrypted and signed image

If the write operation was successful, switch boot mode (see **Table: Boot mode selection for RT1xxx EVK boards** in Connecting the board for RT10xx/RT116x/RT117x devices) and reset the board.

Note: Part of the encrypted image is a DEK key blob encrypted using a master key from the processor. This master key is specific for each processor and cannot be used for another processor.

Note: RT101x and RT102x processors do not support running encrypted images in the NOR flash. In case no other booting device is supported for those processors, the **Encrypted (HAB)** authenticated boot type is not available.

6.12.3.4 Booting XIP encrypted image (BEE OTPMK) authenticated (RT10xx)

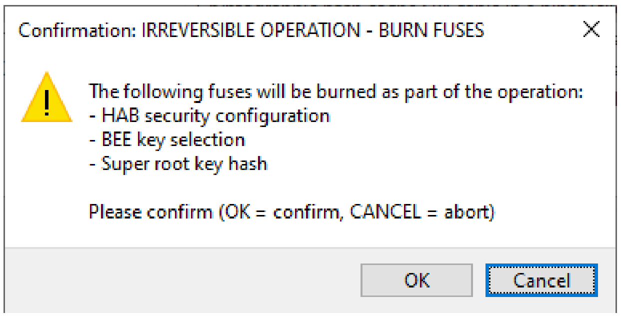
This section describes the building and writing of an XIP encrypted image using the OTP master key. An authenticated image is built and then encrypted on-the-fly during the write operation. The source image for the encrypted XIP with the BEE feature must be an XIP image.

To build the image, do the following:

1. In the **Toolbar**, set the **Boot type** to **XIP encrypted (BEE OTPMK) authenticated**.
2. As a **Source executable image**, use the image running from external NOR flash from Preparing source image for RT10xx/RT116x/RT117x devices as a **Source executable image**.
3. For **Authentication key**, select any key, for example, *SRK1: IMG1_1+CSF1_1*.
4. Click **XIP encryption (BEE OTPMK)** to open the **BEE OTPMK** window. In the window, keep the default settings to encrypt the whole image or configure your own FAC Protected Region ranges within the first BEE encrypted region.
5. Select the **HAB Closed** life cycle.
6. Click the **Build image** button.
7. Check that the bootable image was built successfully.

To write the image, do the following:

1. Switch to **Write image** view.
2. Make sure that the board is set to Serial bootloader (ISP) mode. See **Table: Boot mode selection for RT1xxx EVK boards** in Connecting the board for RT10xx/RT116x/RT117x devices for more information.
3. Enable XIP encryption by setting a corresponding GPIO pin (see **Table: Boot mode selection for RT1xxx EVK boards** in Connecting the board for RT10xx/RT116x/RT117x devices for more information).
4. Click the **Write image** button.
5. In the following window, confirm to write fuses:
   - **OK** - Continue writing the image and burning fuses.
     Note: Burning fuses can only be done once, after that the processor can only execute authenticated images.
   - **Cancel** - Abort writing the image and burning fuses.


Confirmation: IRREVERSIBLE OPERATION - BURN FUSES

The following fuses will be burned as part of the operation:
- HAB security configuration
- BEE key selection
- Super root key hash

Please confirm (OK = confirm, CANCEL = abort)

Figure 59. Burn fuses on i.MX RT10xx for BEE OTPMK encrypted image

If the write operation was successful, switch boot mode (see **Table: Boot mode selection for RT1xxx EVK boards** in Connecting the board for RT10xx/RT116x/RT117x devices) and reset the board.

Note: Step 5 can be replaced by setting the EncryptedXIP fuse in the OTP configuration.

6.12.3.5 Booting XIP encrypted image (BEE user keys) unsigned (RT10xx)

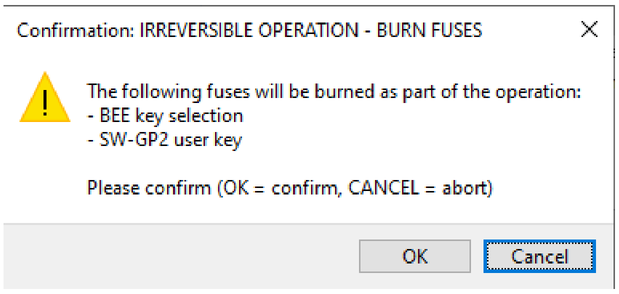
This section describes the building and writing of an XIP encrypted image using user keys. The image itself is built in two steps. First, the unsigned bootable image is built and then this unsigned image is encrypted for use with enabled encrypted XIP. The source image for the encrypted XIP with the BEE feature must be an XIP image.

To build the image, do the following:

1. In the **Toolbar**, set the **Boot type** to **XIP encrypted (BEE user keys) unsigned**.
2. As a **Source executable image**, use the image external NOR flash from Preparing source image for RT10xx/RT116x/RT117x devices as a **Source executable image**.
3. Click **XIP encryption (BEE user keys)** to open the BEE user keys window. In the window, keep the default settings to encrypt the whole image, or edit **User keys data** to provide your specific key. Furthermore, the window allows you to configure additional BEE parameters (Both regions (engines), user key(s) for regions, FAC Protected Region ranges, random key generation).
4. Click the **Build image** button.
5. Check that the bootable image was built successfully.

To write the image, do the following:

1. Switch to **Write image** view.
2. Make sure that the board is set to Serial bootloader (ISP) mode. See **Table: Boot mode selection for RT1xxx EVK boards** in Connecting the board for RT10xx/RT116x/RT117x devices for more information.
3. Enable XIP encryption by setting a corresponding GPIO pin (see **Table: Boot mode selection for RT1xxx EVK boards** in Connecting the board for RT10xx/RT116x/RT117x devices for more information).
4. Click the **Write image** button.
5. In the following window, confirm to write fuses:
   - **OK** - Continue writing the image and burning fuses.
     Note: Burning fuses can only be done once, after that it is not possible to modify them.
   - **Cancel** - Abort writing the image and burning fuses.


Confirmation: IRREVERSIBLE OPERATION - BURN FUSES

The following fuses will be burned as part of the operation:
- BEE key selection
- SW-GP2 user key

Please confirm (OK = confirm, CANCEL = abort)

Figure 60. Burn fuses on i.MX RT10xx for BEE encrypted with user keys

If the write operation was successful, switch boot mode (see **Table: Boot mode selection for RT1xxx EVK boards** in Connecting the board for RT10xx/RT116x/RT117x devices) and reset the board.

Note: Step 5 can be replaced by setting the EncryptedXIP fuse in the OTP configuration.

6.12.3.6 Booting XIP encrypted image (BEE user keys) authenticated (RT10xx)

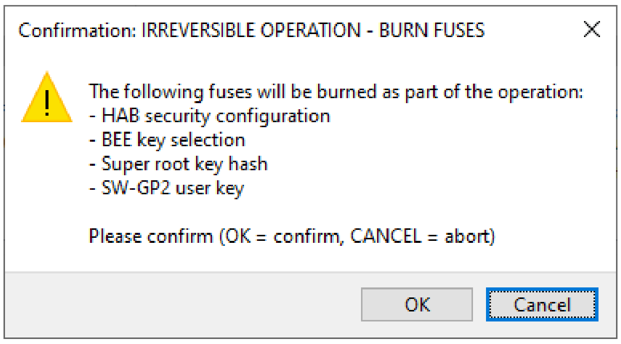
This section describes the building and writing of an XIP encrypted image using user keys. The image itself is built in two steps. First, the authenticated bootable image is built and then this authenticated image is encrypted for use with enabled encrypted XIP. The source image for the encrypted XIP with the BEE feature must be an XIP image.

To build the image, do the following:

1. In the **Toolbar** set the **Boot type** to **XIP encrypted (BEE user keys) authenticated**.
2. As a **Source executable image**, use the image external NOR flash from Preparing source image for RT10xx/RT116x/RT117x devices as a **Source executable image**.
3. For **Authentication key**, select any key, for example, *SRK1: IMG1_1+CSF1_1*.
4. Click XIP encryption (BEE user keys) to open the BEE user keys window. In the window, keep the default settings to encrypt the whole image, or edit user keys data to provide your specific key. Additionally, the window allows you to configure additional BEE parameters (Both regions (engines), user key(s) for regions, FAC Protected Region ranges, random key generation).
5. Select the **HAB Closed** life cycle.
6. Click the **Build image** button.
7. Check that the bootable image was built successfully.

To write the image, do the following:

1. Switch to **Write image** view.
2. Make sure that the board is set to Serial bootloader (ISP) mode. See **Table: Boot mode selection for RT1xxx EVK boards** in Connecting the board for RT10xx/RT116x/RT117x devices for more information.
3. Enable XIP encryption by setting a corresponding GPIO pin. See **Table: Boot mode selection for RT1xxx EVK boards** in Connecting the board for RT10xx/RT116x/RT117x devices for more information).
4. Click the **Write image** button.
5. In the following window, confirm to write fuses:
   - **OK** - Continue writing the image and burning fuses.
     Note: Burning fuses can only be done once, after that the processor can only execute authenticated images.
   - **Cancel** - Abort writing the image and burning fuses.



Confirmation: IRREVERSIBLE OPERATION - BURN FUSES

The following fuses will be burned as part of the operation:
- HAB security configuration
- BEE key selection
- Super root key hash
- SW-GP2 user key

Please confirm (OK = confirm, CANCEL = abort)

Figure 61. Burn fuses on i.MX RT10xx for BEE encrypted and signed image

If the write operation was successful, switch boot mode (see Preparing source image for RT10xx/RT116x/RT117x devices and reset the board.

Note: Step 5 can be replaced by setting the EncryptedXIP fuse in the OTP configuration.

6.12.3.7 Booting XIP encrypted image (OTFAD OTPMK) authenticated (RT10xx)

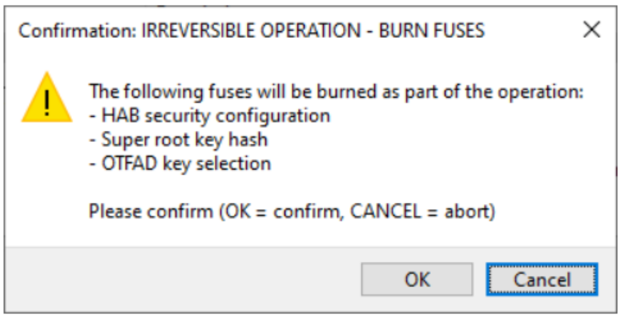
This section describes building and writing of an XIP encrypted image using the OTP master key. The authenticated image is built and then encrypted on-the-fly during the write operation. The source image for the encrypted XIP with the OTFAD feature must be an XIP image.

To build the image, do the following:

1. In the **Toolbar**, set the **Boot type** to **XIP encrypted (OTFAD OTPMK)** authenticated.
2. As a **Source executable image**, use the image running from external NOR flash from Preparing source image for RT10xx/RT116x/RT117x devices as a **Source executable image**.
3. For **Authentication key**, select any key, for example, *SRK1: IMG1_1+CSF1_1*.
4. Click **XIP encryption (OTFAD OTPMK)** to open the **OTFAD OTPMK** window. In the window, keep the default settings to encrypt the whole image or configure your own Protected Region ranges.
5. Select the **HAB Closed** life cycle.
6. Click the **Build image** button.
7. Check that the bootable image was built successfully.

To write the image, do the following:

1. Switch to **Write image** view.
2. Make sure that the board is set to Serial bootloader (ISP) mode. See **Table: Boot mode selection for RT1xxx EVK boards** in Connecting the board for RT10xx/RT116x/RT117x devices for more information.
3. Click the **Write image** button.
4. In the following window, confirm to write fuses:
   - **OK** - Continue writing the image and burning fuses.
     Note: Burning fuses can only be done once, after that the processor can only execute authenticated images.
   - **Cancel** - Abort writing the image and burning fuses.



Confirmation: IRREVERSIBLE OPERATION - BURN FUSES

The following fuses will be burned as part of the operation:
- HAB security configuration
- Super root key hash
- OTFAD key selection

Please confirm (OK = confirm, CANCEL = abort)

Figure 62. Burn fuses on i.MX RT10xx for OTFAD encrypted and signed image

If the write operation was successful, switch boot mode (see **Table: Boot mode selection for RT1xxx EVK boards** in Connecting the board for RT10xx/RT116x/RT117x devices and reset the board.

6.12.3.8 Booting OTFAD encrypted image unsigned with user keys.

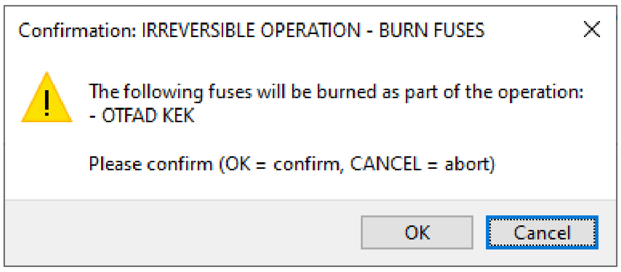
This section describes the building and writing of an OTFAD encrypted image. The image itself is built in two steps.

To build the image, do the following:

1. In the **Toolbar** set **Boot Type** to **Encrypted**, **(OTFAD) unsigned** for RT116x/7x or **XIP encrypted (OTFAD user keys) unsigned** for RT10xx.
2. As a **Source executable image**, use the image external NOR flash from Preparing source image for RT10xx/RT116x/RT117x devices as a **Source executable image**.
3. Click **OTFAD encryption/XIP encryption (OTFAD user keys)** to open the OTFAD Configuration window. In the window set random keys. The window allows you to configure the number of OTFAD regions (contexts), KEK source (OTP or PUF KeyStore), KEK, Key scramble, user keys for regions, regions ranges, random key generation.
4. Open **OTP configuration** and review the settings and fix any reported problems.
5. Click the **Build image** button.
6. Check that the bootable image was built successfully.

To write the image, do the following:

1. Switch to **Write image** view.
2. Make sure that the board is set to Serial bootloader (ISP) mode. See **Table: Boot mode selection for RT1xxx EVK boards** in Connecting the board for RT10xx/RT116x/RT117x devices for more information.
3. Reset the board if the OTFAD KEK source is set to PUF KeyStore. It is necessary so that the key store is enrolled successfully.
4. Enable XIP encryption (RT116x/7x) by setting a corresponding GPIO pin (see **Table: Boot mode selection for RT1xxx EVK boards** in Connecting the board for RT10xx/RT116x/RT117x devices for more information).
5. Click the **Write image** button.
6. In the following window, confirm to write fuses:
   - **OK** - Continue writing the image and burning fuses.
     Note: Burning fuses can only be done once, after that it is not possible to modify them.
   - **Cancel** - Abort writing the image and burning fuses.



Confirmation: IRREVERSIBLE OPERATION - BURN FUSES

The following fuses will be burned as part of the operation:
- OTFAD KEK

Please confirm (OK = confirm, CANCEL = abort)

Figure 63. Burn fuses on i.MX RT10xx/RT116x/RT117x for OTFAD encrypted image

If the write operation was successful, switch boot mode (see **Table: Boot mode selection for RT1xxx EVK boards** in Connecting the board for RT10xx/RT116x/RT117x devices) and reset the board.

Note: Step 6 (RT116x/7x) can be replaced by setting the `ENCRYPT_XIP_EN` fuse in the OTP configuration.

6.12.3.9 Booting OTFAD encrypted image authenticated with user keys

This section describes the building and writing of an OTFAD encrypted image. The image itself is built in two steps. First, the authenticated bootable image is built and then this authenticated image is encrypted for use with OTFAD. The source image for the OTFAD feature must be an XIP image.

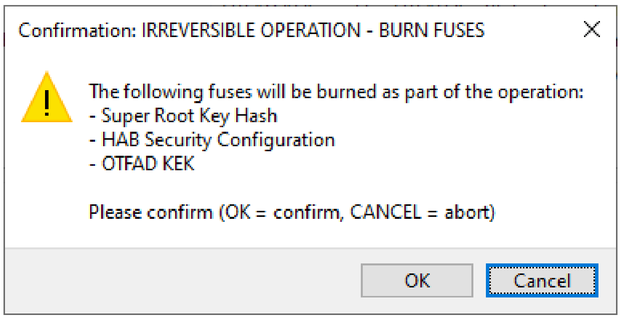
To build the image, do the following:

1. In the **Toolbar** set **Boot type** to **Encrypted (OTFAD) authenticated** for RT116x/7x or **XIP encrypted (OTFAD user keys) authenticated** for RT10xx.
2. As a **Source executable image**, use the image external NOR flash from Preparing source image for RT10xx/RT116x/RT117x devices as a **Source executable image**.
3. Ensure that you have keys generated in the **PKI management** view. For more information, see PKI management.
4. For **Authentication key** select any key, for example, *SRK1: IMG1_1+CSF1_1*.
5. Click **OTFAD encryption** / **XIP encryption (OTFAD user keys)** to open the OTFAD configuration window. In the window set random keys. The window allows you to configure the number of OTFAD regions (contexts), KEK source (OTP or KeyStore), KEK, Key scramble, user keys for regions, regions ranges, random key generation.
6. Select the **HAB Closed** life cycle.
7. Open **OTP configuration** and review the settings and fix any reported problems.
8. Click the **Build image** button.
9. Check that the bootable image was built successfully.

To write the image, do the following:

1. Switch to **Write image** view.
2. Make sure that the board is set to Serial bootloader (ISP) mode. See **Table: Boot mode selection for RT1xxx EVK boards** in Connecting the board for RT10xx/RT116x/RT117x devices for more information.
3. Reset the board if the OTFAD KEK source is set to KeyStore. It is necessary so that the KeyStore is enrolled successfully
4. Enable XIP encryption (RT116x/7x) by setting a corresponding GPIO pin (see **Table: Boot mode selection for RT1xxx EVK boards** in Connecting the board for RT10xx/RT116x/RT117x devices for more information).
5. Click the **Write image** button.
6. In the following window, confirm to write fuses:
   - **OK** - Continue writing the image and burning fuses.

  Note: Burning fuses can only be done once, after that the processor can only execute authenticated images.
- **Cancel** - Abort writing the image and burning fuses.



Confirmation: IRREVERSIBLE OPERATION - BURN FUSES

The following fuses will be burned as part of the operation:
- Super Root Key Hash
- HAB Security Configuration
- OTFAD KEK

Please confirm (OK = confirm, CANCEL = abort)

Figure 64. Burn fuses on i.MX RT10xx/RT116x/RT117x for OTFAD encrypted and sized image

If the write operation was successful, switch boot mode (see **Table: Boot mode selection for RT1xxx EVK boards** in Connecting the board for RT10xx/RT116x/RT117x devices) and reset the board.

Note: Step 6 (RT116x/7x) can be replaced by setting the `ENCRYPT_XIP_EN` fuse in OTP configuration.

6.12.3.10 Booting IEE encrypted image unsigned (RT116x/7x)

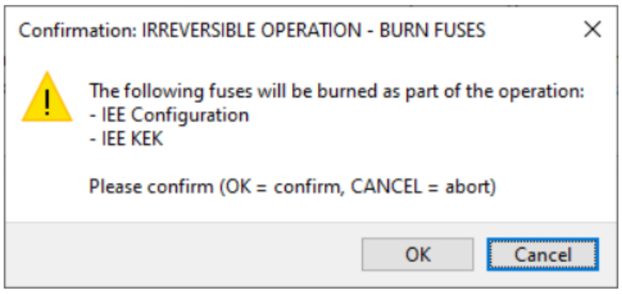
This section describes how to build and write an IEE encrypted image. The image itself is built in two steps. First, the unsigned bootable image is built, and then this unsigned image is encrypted for use with the IEE. The source image for the IEE feature must be an XIP image.

To build the image, do the following:

1. In the **Toolbar** set the **Boot type** to **Encrypted (IEE) unsigned**.
2. As a **Source executable image**, use the image external NOR flash from Preparing source image for RT10xx/RT116x/RT117x devices as a **Source executable image.**
3. Click **IEE encryption** to open the IEE Configuration window. In the window set random keys. The window allows you to configure the number of IEE regions (contexts), KEK, AES encryption mode, and user keys for regions, regions ranges, random key generation.
4. Open the **OTP configuration**, review the settings, and fix any reported problems.
5. Click the **Build image** button.
6. Check that the bootable image was built successfully.

To write the image, do the following:

1. Switch to the **Write image** view.
2. Make sure that the board is set to Serial bootloader (ISP) mode. See **Table: Boot mode selection for RT1xxx EVK boards** in Connecting the board for RT10xx/RT116x/RT117x devices for more information.
3. Reset the board. It is required for successful key store enrollment.
4. Enable XIP encryption by setting a corresponding GPIO pin (see **Table: Boot mode selection for RT1xxx EVK boards** in Connecting the board for RT10xx/RT116x/RT117x devices for more information).
5. Click the **Write image** button.
6. In the following window, confirm to write fuses:
   - **OK** - Continue writing the image and burning fuses.
     Note: Burning fuses can only be done once, after that it is not possible to modify them.
   - **Cancel** - Abort writing the image and burning fuses.



Confirmation: IRREVERSIBLE OPERATION - BURN FUSES

The following fuses will be burned as part of the operation:
- IEE Configuration
- IEE KEK

Please confirm (OK = confirm, CANCEL = abort)

Figure 65. Burn fuses on RT116x/7x for IEE encrypted image

If the write operation was successful, switch boot mode (see **Table: Boot mode selection for RT1xxx EVK boards** in Connecting the board for RT10xx/RT116x/RT117x devices) and reset the board.

Note: Step 5 can be replaced by setting the `ENCRYPT_XIP_EN` fuse in the OTP configuration.

6.12.3.11 Booting IEE encrypted image authenticated (RT116x/7x)

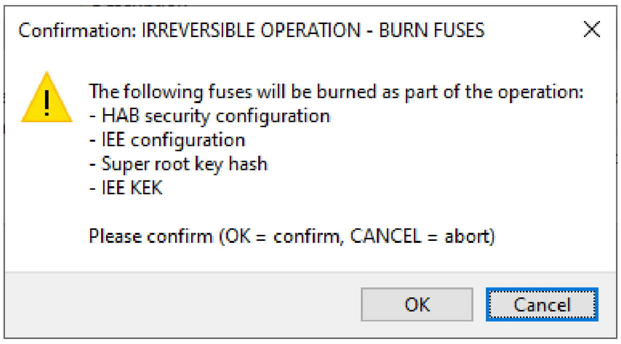
This section describes how to build and write an IEE encrypted image. The image itself is built in two steps. First, the unsigned bootable image is built, and then this unsigned image is encrypted for use with the IEE. The source image for the IEE feature must be an XIP image.

To build the image, do the following:

1. In the **Toolbar** set the **Boot type** to **Encrypted (IEE) authenticated**.
2. As a **Source executable image**, use the image external NOR flash from Preparing source image for RT10xx/RT116x/RT117x devices as a **Source executable image.**
3. Ensure you have keys generated in the **PKI management** view. For more information, see section PKI management in PKI management.
4. For **Authentication key** select any key, for example, SRK1: IMG1_1+CSF1_1.
5. Click **IEE encryption** to open the IEE Configuration window. In the window set random keys. The window allows you to configure the number of IEE regions (contexts), KEK, AES encryption mode, and user keys for regions, regions ranges, random key generation.
6. Select the **HAB Closed** life cycle.
7. Open the **OTP configuration**, review the settings, and fix any reported problems.
8. Click the **Build image** button.
9. Check that the bootable image was built successfully.

To write the image, do the following:

1. Switch to **Write image** view.
2. Make sure that the board is set to Serial bootloader (ISP) mode. See **Table: Boot mode selection for RT1xxx EVK boards** in Connecting the board for RT10xx/RT116x/RT117x devices for more information.
3. Reset the board. It is required for successful key store enrollment.
4. Enable XIP encryption by setting a corresponding GPIO pin (see **Table: Boot mode selection for RT1xxx EVK boards** in Connecting the board for RT10xx/RT116x/RT117x devices for more information).
5. Click the **Write image** button.
6. In the following window, confirm to write fuses:
   - **OK** - Continue writing the image and burning fuses.
     Note: Burning fuses can only be done once, after that it is not possible to modify them.
   - **Cancel** - Abort writing the image and burning fuses.


Confirmation: IRREVERSIBLE OPERATION - BURN FUSES

The following fuses will be burned as part of the operation:
- HAB security configuration
- IEE configuration
- Super root key hash
- IEE KEK

Please confirm (OK = confirm, CANCEL = abort)

Figure 66. Burn fuses on RT116x/7x for IEE encrypted and signed image

If the write operation was successful, switch boot mode (see **Table: Boot mode selection for RT1xxx EVK boards** in Connecting the board for RT10xx/RT116x/RT117x devices) and reset the board.

Note: Step 6 can be replaced by setting the `ENCRYPT_XIP_EN` fuse in the OTP configuration.

6.12.4 Creating/Customizing DCD files

It is recommended to use MCUXpresso Config Tools or MCUXpresso IDE to prepare a DCD binary file.

1. In any of the tools open any project/configuration for the selected processor.
2. Import existing DCD configuration from an SDK source code by selecting **main menu > File > Import > MCUXpresso Config Tools > Import Source**.
3. Select the file from SDK package in *boards\evkmimxrt10##\xip\evkmimxrt10##_sdram_ini_dcd.c*.
4. Switch to the Device Configuration tool by selecting **main menu > Config Tools > Device Configuration**.
5. In the toolbar of the **DCD view**, select **Output Format** to **binary**.
6. Navigate to **Code Preview** and in the toolbar click the **Export** button and select the location where to generate a binary file.
   Note: Refer to the documentation of the Device Configuration Tool for more information.

6.13 RT118x device workflow

This section describes the RT118x device workflow in detail.

6.13.1 Preparing source image for RT118x devices

In this step, select the target memory where the image will be executed. The following options are available for RT118x devices:

- **Image running from an external NOR flash**
  It is the so-called **XIP**(e**X**ecution **I**n **P**lace) image. It means that the image is executed directly from the memory where it is.
- **Image running in RAM**
  This image can be on an SD card/eMMC, in external flash (FlexSPI NOR, FlexSPI NAND). During boot, it will be copied into RAM and executed from there, or alternatively, it can be loaded into RAM using the serial downloader. The following RAM types are supported:
  – internal RAM
  – SDRAM
  – HyperRAM The source image preparation is similar to RT116x/7x, see Preparing source image for RT10xx/RT116x/RT117x devices. Some RT118x specifics:

- Select an example for core cm33 and then set the target to Cortex-M33. ELE firmware is distributed in MCUXpresso SDK in the ‘ELE crypto’ component, typically the `firmware\edgelock\mxrt1180-ahab-container.img` file.
- For MCUXpresso IDE and CM33 image in internal RAM, the following project modifications are needed. These changes were verified in MCUXpresso IDE version 25.06 in conjunction with MCUXpresso SDK version 25.09:
  - ensure the ITC is the first RAM listed in the available memory areas (see *project Properties* > *C/C++ Build* > *MCU settings*)
  - change global data placement and “CodeQuickAccess” to DTC_cm33 and ensure “Link application to RAM” checkbox is selected (see *project Properties* > *C/C++ Build* > *Settings* > *Managed Linker Script*)
  - change `data_init()` and `bss_init()` functions in `startup_mimxrt1189_cm33.c` code:

```
__attribute__ ((section(".after_vectors.init_data")))
void data_init(unsigned int romstart, unsigned int start, unsigned int len) {
    if (start == romstart)
        return;
    unsigned int *pulDest = (unsigned int*) start;
    unsigned int *pulSrc = (unsigned int*) romstart;
    unsigned int loop;
    for (loop = 0; loop < len; loop = loop + 4)
        *pulDest++ = *pulSrc++;
}

__attribute__ ((section(".after_vectors.init_bss")))
void bss_init(unsigned int start, unsigned int len) {
    unsigned int *pulDest = (unsigned int*) start;
    unsigned int loop;
    for (loop = 0; loop < len; loop += 4, ++pulDest)
        if (*pulDest != 0)
         *pulDest = 0;
}
```

6.13.2 Connecting the board for RT118x devices

This section contains information about configuring the RT118x boards and connecting them to the SEC Tool.

EVK/FRDM boards overview:

EVK/FRDM boards	Power selection jumper	USB	UART	FlexSPI instance 1	FlexSPI instance 2
MIMXRT1180-EVK	J1 (3-4 over USB; 7-8 over UART)	J33	J53	FlexSPI NOR	HyperRAM
MIMXRT1180A-EVK	J1 (3-4 over USB; 7-8 over UART)	J33	J53	FlexSPI NOR	HyperRAM
MIMXRT1180-144	J1 (3-4 over UART)	N/A	J53	FlexSPI NOR	HyperRAM
FRDM-IMXRT1186	J4 (3-4 over USB; 5-6 over UART)	J63	J23	HyperRAM	FlexSPI NOR

Note:

- For FRDM-IMXRT1186 the SEC Tool automatically configures booting from FlexSPI instance 2 (FlexSPI NOR), which results in the BOOT_CFG2 fuse being burnt

1. See **Table: Boot mode selection for RT1xxx EVK boards** in Connecting the board for RT118x devices in for instructions on how to set boot mode using DIP switches.
2. Make sure that you have the power selection jumper set to 3-4 to power the board from USB (from UART on RT1180-144).
3. Connect the board over the USB cable to your PC over UART or USB.
4. Ensure that SEC is already running with a workspace created for the chosen device. For more information, see Setting up Secure Provisioning Tool.
5. Make sure that the **Boot device** in the **Boot Memory Configuration** matches the FlexSPI NOR flash used on the EVK/FRDM board.
6. Set the connection to your selected connection (**USB** or **UART**) and test the board connection.

6.13.2.1 Booting from SD card

For booting from an SD card, do the following:

1. Insert a micro SDHC card into the board.
2. Select **SD card, SDHC SD-card 8GB USDHC1** in the **Boot Memory Configuration**.

6.13.2.2 Booting from eMMC:

For booting from an eMMC, do the following:

1. eMMC can be installed on board, or it is possible to connect eMMC through SD slot by SD/eMMC adapter.
2. Select **eMMC, SDHC eMMC 8 GB USDHC1** in the **Boot Memory Configuration**.

6.13.3 Booting from serial downloader

For booting from a serial downloader, do the following:

1. Set ISP boot mode on board boot mode switch
2. Select **RAM via serial downloader** in the **Boot Memory Configuration**

6.13.3.1 Table: Boot mode selection for RT118x EVK boards

Boot mode/Device	Serial bootloader (ISP mode)	Flex-SPI NOR	SD card/ eMMC	FlexSPI NAND
RT1180-EVK	SW5: x001	SW5: x100	SW5: x011	SW5: x101 (N/A)
RT1180A-EVK	SW5: x001	SW5: x100	SW5: x011	SW5: x101 (N/A)
RT1180-144	SW5: x100	SW5: x001	SW5: x110	SW5: x101 (N/A)
FRDM-IMXRT1186	J60: 100	J60: 001	J60: 110	J60: 101 (N/A)

6.13.4 Booting images for RT118x devices

This section describes the building and writing of bootable images.

You can use several combinations of used memories:

Memory where the image is executed	Boot memory: Memory where the image is written	XMCD needed	XIP
External NOR flash	External NOR flash	No	Yes
Internal RAM	External NOR or NAND flash	No	No

Memory where the image is executed	Boot memory: Memory where the image is written	XMCD needed	XIP
Internal RAM	SD card or eMMC	No	No
SDRAM/HyperRAM	External NOR or NAND flash	Yes	No
SDRAM/HyperRAM	SD card or eMMC	Yes	No

Note:

- Memory, where the image is executed is explained in Preparing source image for LPC55(S)0x/1x/2x/6x devices.
- Memory, where the image is written is configured as Boot Memory in SEC.

6.13.4.1 Booting unsigned image

An unsigned image is typically used for development. Start with this boot type before working with secured images to verify that the executable image works properly.

First, build a bootable image:

1. Make sure that you have selected the **Unsigned boot type** in the **Toolbar**.
2. Switch to the **Build image** view.
3. Select the image built in Preparing source image for RT118x devices as a **Source executable image**.
4. For images executed from SDRAM/HyperRAM, configure SEMC SDRAM / FlexSPI HyperRAM using XMCD. For EVK boards, the following XMCD configuration files can be used: <SEC>/sample_data/targets/MIMXRT118#/evkmimxrt118#_xmcd_*_simplified.yaml.
5. If needed, open **Dual image boot** and configure.
6. Click the **Build image** button to build a bootable image.

When the bootable image has been successfully built:

1. Make sure that the board is in Serial bootloader (ISP) mode.
2. Switch to the **Write image** view.
3. Click the **Write image** button.
4. If the write operation was successful, switch boot mode (see **Table: Boot mode selection for RT118x EVK boards** in Connecting the board for RT118x devices) and reset the board.

6.13.4.2 Booting signed image

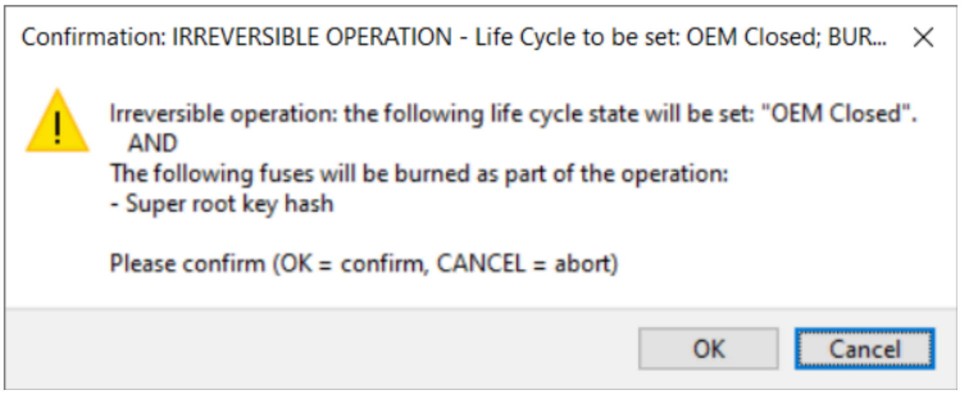
This section describes the building and writing of a signed image. Keys generated in the **PKI management** view are needed in this step. For more information about generating keys, see Generate keys. If you want to use an encrypted image, you can skip this step.

First, build a bootable image:

1. In the **Toolbar** set **Boot type** to **Signed**.
2. In the **Build image** view, use the image from Preparing source image for RT118x devices as a **Source executable image**.
3. For **Authentication key** select any key, for example, *SRK1*.
4. Select the **OEM Closed** life cycle.
5. Click the **Build image** button.
6. Check that the bootable image was built successfully.

To write the image, switch to **Write image** view.

1. Make sure that the board is set to Serial bootloader (ISP) mode. See **Table: Boot mode selection for RT118x EVK boards** in Connecting the board for RT118x devices for more information.
2. Click the **Write image** button.
3. In the following window, confirm to write fuses:
   - **OK** - Continue writing the image and burning fuses.
   - **Cancel** - Abort writing the image and burning fuses.


Confirmation: IRREVERSIBLE OPERATION - Life Cycle to be set: OEM Closed; BUR...

Irreversible operation: the following life cycle state will be set: "OEM Closed".
AND
The following fuses will be burned as part of the operation:
- Super root key hash

Please confirm (OK = confirm, CANCEL = abort)

Figure 67. Burn fuses on i.MX RT118x for signed image

1. If the write operation was successful, switch boot mode (see **Table: Boot mode selection for RT118x EVK boards** in Connecting the board for RT118x devices) and reset the board.

6.13.4.3 Booting encrypted (AHAB) image

This section describes the building and writing of an encrypted image. This image is decrypted into RAM during booting operation, so an XIP image cannot be used. The keys generated in the **PKI management** view are needed in this step. For more information about generating keys, see Generate keys.

To build the image, do the following:

1. In the **Toolbar** set the **Boot type** to **Encrypted (AHAB)**.
2. As a **Source executable image**, use the image from Preparing source image for RT118x devices as a **Source executable image**.
3. For **Authentication key** select any key, for example, *SRK1*.
4. Select the **OEM Closed** life cycle.
5. Click the **Build image** button.
6. Check that the bootable image was built successfully.

To write the image, do the following:

1. Switch to the **Write image** view.
2. Make sure that the board is set to Serial bootloader (ISP) mode. See **Table: Boot mode selection for RT118x EVK boards** for more information.
3. Click the **Write image**. button.
4. In the following window, confirm to write fuses:
   - **OK** - Continue writing the image and burning fuses.
   - **Cancel** - Abort writing the image and burning fuses.
5. If the write operation was successful, switch boot mode (see **Table: Boot mode selection for RT118x EVK boards** in Connecting the board for RT118x devices) and reset the board.

Note: Part of the encrypted image is a DEK key blob encrypted using a master key from the processor. This master key is specific for each processor and cannot be used for another processor. DEK key blob is generated during write and the AHAB image is then updated with this processor-specific key blob.

6.13.4.4 Booting OTFAD encrypted image

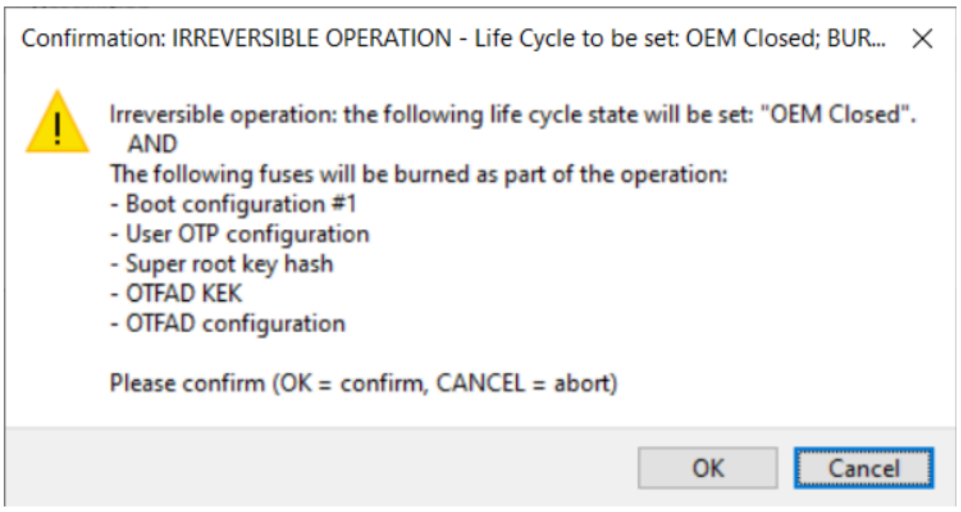
This section describes the building and writing of an OTFAD encrypted image. The image itself is built in two steps. First, the unsigned/signed AHAB image is built and then this AHAB image is encrypted for use with OTFAD. The source image for the OTFAD feature must be an XIP image. The Keys generated in the **PKI management** view are needed in this step. For more information about generating keys, see Generate keys.

To build the image, do the following:

1. In the **Toolbar** set the **Boot type** to **Encrypted (OTFAD) signed** or **Encrypted (OTFAD) unsigned**.
2. As a **Source executable image**, use the image external NOR flash from Preparing source image for RT118x devices as a **Source executable image**.
3. For **Authentication key** select any key, for example, *SRK1*.
4. Click **OTFAD encryption** to open the OTFAD configuration window. In the window set random keys. The window allows you to configure the number of OTFAD regions (contexts), KEK source (OTP or DUK), KEK, Key scramble, user keys for regions, regions ranges, random key generation.
5. Select the **OEM Closed** life cycle.
6. Open the **OTP configuration** and review the settings and fix any reported problems.
7. Click the **Build image** button.
8. Check that the bootable image was built successfully.

To write the image, do the following:

1. Switch to **Write image** view.
2. Make sure that the board is set to Serial bootloader (ISP) mode. See **Table: Boot mode selection for RT118x EVK boards** in Connecting the board for RT118x devices for more information.
3. Click the **Write image** button.
4. In the following window, confirm to write fuses:
   - **OK** - Continue writing the image and burning fuses.
   - **Cancel** - Abort writing the image and burning fuses.


Confirmation: IRREVERSIBLE OPERATION - Life Cycle to be set: OEM Closed; BUR...

Irreversible operation: the following life cycle state will be set: "OEM Closed".
AND
The following fuses will be burned as part of the operation:
- Boot configuration #1
- User OTP configuration
- Super root key hash
- OTFAD KEK
- OTFAD configuration

Please confirm (OK = confirm, CANCEL = abort)

Figure 68. Burn fuses on i.MX RT118x for OTFAD encrypted image

1. If the write operation was successful, switch boot mode (see **Table: Boot mode selection for RT118x EVK boards** in Connecting the board for RT118x devices) and reset the board.

6.13.4.5 Booting IEE encrypted

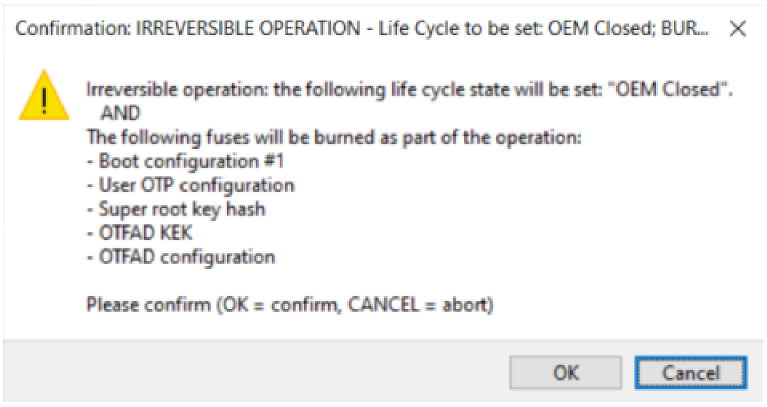
This section describes the building and writing of an IEE encrypted image. The image itself is built in two steps. First, the unsigned/signed AHAB image is built and then this AHAB image is encrypted for use with IEE. The source image for the IEE feature must be an XIP image. The Keys generated in the **PKI management** view are needed in this step. For more information about generating keys, see Generate keys.

To build the image, do the following:

1. In the **Toolbar** set the **Boot type** to **Encrypted (IEE) signed** or **Encrypted (IEE) unsigned**.
2. As a **Source executable image**, use the image external NOR flash from Preparing source image for RT118x devices as a **Source executable image**.
3. For **Authentication key** select any key, for example, *SRK1*.
4. Click **IEE encryption** to open the IEE configuration window. In the window set random keys. The window allows you to configure the number of IEE regions (contexts), AES encryption mode, and user keys for regions, regions ranges, random key generation.
5. Select the **OEM Closed** life cycle.
6. Open the **OTP configuration** and review the settings and fix any reported problems.
7. Click the **Build image** button.
8. Check that the bootable image was built successfully.

To write the image, do the following:

1. Switch to the **Write image** view.
2. Make sure that the board is set to Serial bootloader (ISP) mode. See **Table: Boot mode selection for RT118x EVK boards** in Connecting the board for RT118x devices for more information.
3. Click the **Write image** button.
4. In the following window, confirm to write fuses:
   - **OK** - Continue writing the image and burning fuses.
   - **Cancel** - Abort writing the image and burning fuses.


Confirmation: IRREVERSIBLE OPERATION - Life Cycle to be set: OEM Closed; BUR...

Irreversible operation: the following life cycle state will be set: "OEM Closed".
AND
The following fuses will be burned as part of the operation:
- Boot configuration #1
- User OTP configuration
- Super root key hash
- OTFAD KEK
- OTFAD configuration

Please confirm (OK = confirm, CANCEL = abort)

Figure 69. Burn fuses on i.MX RT118x for IEE encrypted image

1. If the write operation was successful, switch boot mode (see **Table: Boot mode selection for RT118x EVK boards** in Connecting the board for RT118x devices) and reset the board.

6.13.4.6 Booting multicore images

This section describes how to build and write an image for multiple cores (Cortex M33 and Cortex M7). Only Cortex M33 is a bootable core. The source image used for Cortex M33 must trigger the second core in the code (Cortex M7). See `demo_apps\multicore_trigger` example in the SDK package for RT118x.

To build multicore images, do the following:

- In this example, the Cortex M7 XIP image runs from external Flash (start address 0x2800B000) and the Cortex M33 image runs from internal RAM (start address 0xFFE0000):

1. Set the **Source executable image** (image for Cortex M33) in the **Build** tab.
2. Open the **Additional User/OEM Image** dialog via the **Additional images** button (the application binary image is automatically filled up).
3. Specify a standalone Cortex M7 executable binary image running from the flash memory and set the following values:
   - Image offset - 0xA000. It is calculated as: Load address (0x2800B000) - FlexSPI NOR base address (0x28000000) - AHAB image offset in FlexSPI NOR (0x1000)
   - Load address - 0x2800B000
   - Entry point - 0x2800B000
   - Core Id - cortex-m7
   - Image type - executable
4. Close the dialog by clicking the **OK** button.
5. Click the **Build image** button.

- In this example, the Cortex M7 image runs from internal ITCM RAM (start address 0x0) and the Cortex M33 XIP image runs from external flash (start address 0x2810B000). This use case requires additional fuses, so the internal RAM is properly accessible.

1. Set the **Source executable image** (image for Cortex M33) in the **Build** tab.
2. Open the **Additional User/OEM Image** dialog via the **Additional images** button (the application binary image is automatically filled up).
3. Specify a standalone Cortex M7 executable binary image running from ITCM RAM and set the following values:
   - Image offset - 0x10F400 - this can be any value, which does not overlap with other image (in this example the image was placed directly after the Cortex M33 image)
   - Load address - 0x303C0000 (secured alias of CM7 ITCM in the CM33 core address space)
   - Entry point - 0x0 (the start addresses of the image in the CM7 address space)
   - Core Id - cortex-m7
   - Image type - executable
4. Close the dialog by clicking the **OK** button.
5. Open the **OTP Configuration** dialog by clicking the **OTP configuration** button in the left-bottom corner.
6. Set POR_PRELOAD_MC7_TCM_ECC and RELEASE_M7_RST_STAT fuses (BOOT_CFG7) to 1 and fix any reported problems.
7. Close the dialog via the **OK** button.
8. Click the **Build image** button.

Note: The write process of the multicore images is the same as for Booting IEE encrypted.

6.13.4.7 Life cycle for RT118x device workflow

The default life cycle, which should be used for development, is **OEM Open**. Before you deploy the application, set the **OEM Closed** or **OEM Locked** life cycle (see documentation for the target processor for detailed description).

Note: Change of the life cycle is irreversible.

Once the processor is in “OEM Closed” or “OEM Locked” mode:

- The tool processor does not allow burn fuses via blhost. The application can still be updated.
- The processor can only execute signed images

6.13.4.8 Firmware version

The firmware version value is included in the AHAB container header, field Fuse version. This version value is fused indirectly through the ELE commit API when the AHAB container is authenticated. This commit API must be called from the application itself.

The AHAB image with a lower fuse version than the version fused cannot be loaded during booting.

The firmware version is supported for signed images only.

6.14 RT5xx/6xx device workflow

This section describes the RT5xx/6xx device workflow in detail.

6.14.1 Preparing source image RT5xx/6xx devices

In this step, you must select the target memory where the image will be executed.

Supported boot memories are NOR flash, SD card, and eMMC.

The following boot types are available for RT5xx/6xx processors:

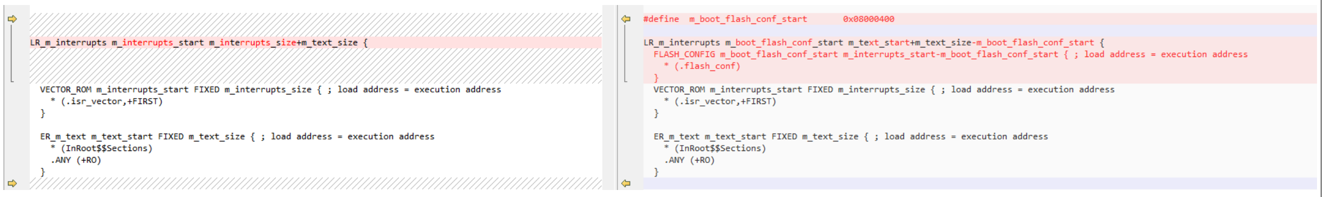
- Image running in external flash: The XIP image (eXecuted In Place)
- Image running in internal RAM: The image is copied from FLASH, SD card, or eMMC to RAM before the execution

It is recommended to use the **gpio_led_output** example for verification, the image is started properly. By default, this example triggers LED blinking only if the user button is clicked, but it is possible to modify it to blink all the time.

6.14.1.1 Image running in external flash

The gpio_led_output example is linked into external flash by default. Disable the XIP Boot header, as it will be created by SEC.

- **MCUXpresso IDE**
  1. Optional step: go to Project > Properties > C/C++ Build > Settings > MCU C Compiler > Preprocessor > Defined symbols and set `BOOT_HEADER_ENABLE` to `0`.
  2. Build the image. You will find the resulting source image named as `Debug\\evkmimxrt685_gpio_led_output.axf`. It can be used as an input for the bootable image in SEC Tool.
- **Keil MDK**
  1. In the **Toolbar**, select **gpio_output_flash_debug** target.
  2. Optional step: in **Project > Options > “*C/C++*”** disable define symbol `BOOT_HEADER_ENABLE=0` (set to 0).
  3. In **Project > Options > Output** select the **Create HEX file** checkbox.
  4. Double-check that the application is linked to 0x8001000. If not, the following fix must be applied to the linker as a workaround for the problem:



```
LR_m_interrupts m_interrupts_start m_interrupts_size+m_text_size {

  VECTOR_ROM m_interrupts_start FIXED m_interrupts_size { ; load address = execution address
    * (.isr_vector,+FIRST)
  }

  ER_m_text m_text_start FIXED m_text_size { ; load address = execution address
    * (InRoot$$Sections)
    .ANY (+RO)
  }
```

```
#define  m_boot_flash_conf_start       0x08000400

LR_m_interrupts m_boot_flash_conf_start m_text_start+m_text_size-m_boot_flash_conf_start {
  FLASH_CONFIG m_boot_flash_conf_start m_interrupts_start-m_boot_flash_conf_start { ; load address = execution address
    * (.flash_conf)
  }
  VECTOR_ROM m_interrupts_start FIXED m_interrupts_size { ; load address = execution address
    * (.isr_vector,+FIRST)
  }

  ER_m_text m_text_start FIXED m_text_size { ; load address = execution address
    * (InRoot$$Sections)
    .ANY (+RO)
  }
```

Figure 70. Keil MDK workaround

1. Build the image. You will find the output image as `boards\evkmimxrt685\driver_examples\gpio\led_output\mdk\flash_debug\gpio_led_output.out`.

- **IAR Embedded Workbench**
  1. In **Project > Edit Configurations …**, select `flash_debug`.
  2. Optional step: in **Project Options > C/C++ Compiler > Preprocessor > Defined Symbols**, add or change the existing `BOOT_HEADER_ENABLE` define to `0`.
  3. Build the image. You will find the output image as `boards\evkmimxrt###\driver_examples\gpio\led_output\iar\flash_debug\gpio_led_output.out`.

6.14.1.2 Image running in internal RAM

- **MCUXpresso IDE**
  1. Optional step: Go to **Project > Properties > C/C++ Build > Settings > MCU C Compiler > Preprocessor > Defined symbols** and set `BOOT_HEADER_ENABLE` to `0`.
  2. Select **Project > Properties - C/C++ Build > Settings > Tool Settings > MCU Linker > Managed Linker Script** and check `Link application to RAM`.
  3. Build the image. You will find the built image as `Debug\evkmimxrt685_gpio_led_output.axf`. You can later use it as a **Source executable image** by SEC.
- **Keil MDK**
  Because example projects for MDK are not built into RAM, you must manually modify the linker file. Generic description of such changes is not documented yet.
- **IAR Embedded Workbench**
  1. In **Project > Edit Configurations …**, select `debug`.
  2. Optional step: In **Project Options > C/C++ Compiler > Preprocessor > Defined Symbols**, add or change the existing `BOOT_HEADER_ENABLE` define to `0`.
  3. Build the image. You will find the output image built as `boards\evkmimxrt###\driver_examples\gpio\led_output\iar\debug\gpio_led_output.out`.

6.14.2 Connecting the board for RT5xx/6xx devices

This section contains information about configuring the following evaluation boards and connecting them to SEC:

- MIMXRT595-EVK
- MIMXRT685-AUD-EVK

Table: Boot configurations of the RT5xx/6xx EVK boards

Boot Mode/Device	ISP Mode	FlexSPI Boot	SD card	eMMC
MIMXRT595-EVK	SW7[1:3]: 100 (UART, SPI, I2C)SW7[1:3]: 101 (USB)	SW7[1:3]: 001	SW7[1:3] 011	SW7[1:3] 110
MIMXRT685-AUD-EVK	SW5[1:3]: 100	SW5[1:3]: 101	SW5[1:3] 011	SW5[1:3] 110

where **1** means the switch is ON, **0** means the switch is OFF.

1. Switch the board to ISP Mode and reset. For more information, see the above table.
2. Connect to the **J7** port with the USB cable to your PC.
3. Ensure that you have started SEC with a new workspace. For more information, see Setting up Secure Provisioning Tool.
4. Set the connection to **USB** and test the board connection.

Booting from SD card:

For booting from an SD card, do the following:

1. Insert SDHC card into the board in Secure Provisioning Tool
2. Select the Boot memory: sd_card/SDHC SD card 8 GB in the **Toolbar**.
3. SDHC power cycle must be set to ENABLED for evk boards

Booting from eMMC:

For booting from an eMMC, do the following:

1. eMMC can be installed on board, or it is possible to connect eMMC through SD slot by SD/eMMC adapter.
2. Select Boot memory: eMMC/SDHC eMMC 8 GB in the **Toolbar.**

6.14.3 Booting images for RT5xx/6xx devices

This chapter describes the building and writing of plain and signed bootable images.

6.14.3.1 Booting a plain/plain with CRC image

A plain image is typically used for development. It is recommended to start with this boot type before working with secured images to verify that the executable image works properly.

To build a bootable image, follow these steps:

1. In the **Toolbar**, select **Plain unsigned** or **Plain with CRC** in **Boot type**.
2. Switch to the **Build image** view.
3. Select image build in Preparing source image RT5xx/6xx devices as a **Source executable image**. If needed, open the **Dual image boot** and configure. If configured, open **OTP configuration** and review all reported problems. For fuse *BOOT_CFG[3]* being locked after write, it is necessary to specify the whole value as it is programmed only once.
4. Click the **Build image** button to build a bootable image.

When the bootable image has been successfully built:

1. Connect the board, see Connecting the board RT5xx/6xx devices.
2. Switch to the **Write image** view.
3. **Reset the board**. If the write script is executed twice without resetting the board, the configuration of external memory may fail unless the fuse with the QSPI reset pin is not burnt.
4. Click the **Write image** button.

If the write operation was successful, switch boot mode to **FlexSPI boot** (**Table: Boot configurations of the RT5xx/6xx EVK boards** in Connecting the board RT5xx/6xx devices) and reset the board.

6.14.3.2 Booting plain signed image using shadow registers

This section describes the building and writing of an authenticated image.

1. In the **Toolbar** set **Boot type** to **Plain signed**.

2. In the **Build image** view, use the image from Preparing source image RT5xx/6xx devices as a **Source executable image**.
3. Ensure that you have keys generated in the **PKI management** view. For more information, see PKI management.
4. For **Authentication key**, select any key, for example, *ROT1: IMG1_1*.
5. As a **Key source**, select **OTP** or **KeyStore**. KeyStore represents a higher security level, as PUF is used. See Securing the processor for the limitations.
6. Generate a random **User key** and **SBKEK**.
7. If needed, open **Dual image boot** and configure.
8. Select the **Development** life cycle.
9. Open **OTP configuration** and review all reported problems. For fuses *BOOT_CFG[0]* and *BOOT_CFG[3]* being locked after write, it is necessary to specify the whole value, as it will be programmed only once.
10. Click the **Build image** button and check that the bootable image was built successfully.

To write the image, do the following:

1. To write the image, switch to **Write image** view.
2. Make sure that the board is set to Serial bootloader (ISP) mode. For more information, see Connecting the board RT5xx/6xx devices.
3. **Reset the board**. It is necessary because:
   - Shadow registers cannot be updated or written twice, they can be set to a “clean” processor only.
   - If the fuse for the QSPI reset pin is not burnt, it is necessary to reset the flash manually before each configuration.
4. Click the **Write image** button.

During the write operation, the following steps are performed:

1. Fuses are checked to ensure that the board is in an unsecured mode.
2. A simple application is written into RAM. The application initializes shadow registers.
3. Shadow registers data are written into RAM. The application is started.
4. The application resets the processor.
5. The write_image script is started to configure external flash and write the application into flash.

6.14.3.3 Booting OTFAD encrypted image using shadow registers

This section describes the building and writing of an encrypted image (OTFAD encryption).

1. In the **Toolbar** set Boot Type to **Encrypted (OTFAD) with CRC** or **Encrypted (OTFAD) signed**.
2. In the **Build image** view, use the image from Preparing source image RT5xx/6xx devices as a **Source executable image**.
3. Ensure you have keys generated in the **PKI management** view. For more information, see PKI management.
4. For **Authentication key**, select any key, for example, ROT1: IMG1_1.
5. As a **Key source**, select **OTP** or **KeyStore**. KeyStore represents a higher security level, as PUF is used. See Securing the processor for the limitations.
6. Generate a random **User key** and **SBKEK**.
7. Open **OTFAD encryption** and set random keys.
8. If needed, open **Dual image boot** and configure.
9. Select **Development** life cycle.
10. Open **OTP configuration** and review all reported problems. For fuses *BOOT_CFG[0]* and *BOOT_CFG[3]* being locked after write, it is necessary to specify the whole value, as it will be programmed only once.
11. Click the **Build image** button and check that the bootable image was built successfully.

To write the image, do the following:

1. To write the image, switch to **Write image** view.
2. Make sure that the board is set to Serial bootloader (ISP) mode. For more information, see Connecting the board RT5xx/6xx devices.
3. **Reset the board**. It is necessary because:
   - Shadow registers cannot be updated or written twice, they can be set to a “clean” processor only.
   - If the fuse for the QSPI reset pin is not burnt, it is necessary to reset the flash manually before each configuration.
4. Click the **Write image** button.

During the write operation, the following steps are performed:

1. Fuses are checked to ensure that the board is in an unsecured mode.
2. A simple application is written into RAM. The application initializes shadow registers.
3. Shadow registers data are written into RAM. The application is started.
4. The application resets the processor.
5. The SB file is applied to the processor, and during this process, the following actions will be done:
   - Configure external flash
   - Erase flash (KeyStore is preserved if it is used)
   - Create an FCB block at the beginning of the flash
   - Write an encrypted application
   - Write OTFAD key blobs

Note: Repetitive Write to QSPI flash might fail if the board is not Reset.

6.14.3.4 Booting signed/encrypted image - burn fuses

Burning fuses is an irreversible operation, which should be performed only after the bootable image was tested with shadow registers. It is also recommended to safely back up all keys prior to burning fuses. The booting process is identical to the process described in Booting plain signed image using shadow registers with only one difference in write operation - the Deployment life cycle must be selected. During the write operation, the shadow registers will not be initialized, and the write image script will burn the fuses instead. The GUI will display confirmation with a list of fuses groups to be burnt.

Note: Detailed info about the modification of fuses can be reviewed in OTP/PFR/IFR configuration.

6.14.3.5 Securing the processor

To enable full security on RT5xx/6xx processors, `DCFG_CC_SOCU` and `DCFG_CC_SOCU_AP` fuses must be burnt.

SEC does not set up these fuses for burning by default (even if the Deployment life cycle is selected), but you can configure it in **OTP/PFR configuration**.

Once the `DCFG_CC_SOCU` fuses are set, it is no longer possible to modify security configuration parameters, and no changes in the key store are allowed using blhost. If KeyStore is used, the image can be updated in SEC only if:

- ISP mode is still enabled, and
- `QSPI_ISP_AUTO_PROBE_EN` bin in *BOOT_CFG[1]* fuse is enabled.
  This feature is not supported on RT6xx processors, and the image can be updated only using a custom bootloader.

If the keys are stored in OTP fuses, no limitation applies.

Note that OTFAD encryption on RT600 is not supported with KeyStore, because there is no support to back up and restore KeyStore during erasing the flash in the SB file.

To test the `DCFG_CC_SOCU` configuration in the shadow registers, the following steps must be followed:

- prepare and test the application image without `DCFG_CC_SOCU` (for example, all `DCFG_CC_SOCU` must be zero)
- once the image is running and the FCB configuration is valid and available in FLASH, configure `DCFG_CC_SOCU`. The processor will now be set to full security, so some blhost operations are no longer available. The processor can be updated via SB file only.

6.14.3.6 Device HSM provisioning

This section describes the provisioning of the processor using key blob encrypted using device HSM, which allows to transfer keys (fuses values) securely and application bootable image into the factory. Device HSM is supported for all secure boot types: Plain signed and OTFAD encrypted in the **deployment** life cycle. Device HSM can be selected from the toolbar in the trust provisioning type selection dialog.

In device HSM mode, several fuses are configured by trust provisioning firmware. These fuses must be configured, so the tool displays an error if the required value is not specified. The encryption of the fuses into the key blob is done using the EVK evaluation board. During the build operation, the fuses values are written into RAM, and then trust provisioning firmware creates a key blob.

To do the build, follow the steps below:

1. On the toolbar, ensure that the **Plain signed** or **Encrypted** boot type is selected.
2. On the toolbar, ensure that the **Deployment** life cycle is selected.
3. On the toolbar, select **Device HSM** provisioning type.
4. In OTP configuration, ensure that all fuses for device HSM are specified; additional fuses can be burnt in the application SB file.
5. Connect EVK board using UART or USB connector.
6. Open the **Connection** dialog and test the connection.
7. Run the build. During the build, the bootable image and SB file are created and then the fuses values are written to processor RAM and then encrypted using provisioning firmware. The fuses of the processor are not affected in this step.
8. After the build script, the encrypted key blob is read.

Write

Write operation is the same as secure boot type using a write script. The write script uses the same provisioning firmware to decrypt the key blob and burn selected fuses and then it resets the processor. It uses the SB file to install the application image. In case the booting device is selected by the fuse, ensure that the booting device is empty, so the processor falls into ISP mode after the reset and the application image can be uploaded.

Manufacturing package

For the manufacturing operation, it is recommended to create a manufacturing package. For more information, see Manufacturing workflow.

6.15 RT7xx device workflow

This section describes the i.MX RT7xx device workflow in details.

6.15.1 Preparing source image for RT7xx devices

In this step, select the target memory where the image is to be executed.

The supported boot memories are NOR flash (instance 0 or 1) or eMMC.

The following boot types are available for RT7xx processors:

- Image running in external flash: the XIP image (eXecuted In Place)
- Image running in internal RAM. The ROM during the boot time copies the image from FLASH or eMMC to RAM.
- Image running in external RAM (PSRAM). The ROM during the boot time initializes the external RAM (see XMCD) and copies the image from FLASH or eMMC to external RAM.

6.15.2 Connecting the board for RT7xx devices

This section contains information about configuring the MIMXRT700-EVK evaluation board and connecting it to SEC.

Table: Boot configurations of the RT7xx boards

Board	ISP mode	XSPI0	XSPI1	eMMC	PSRAM
MIMXRT700-EVK	SW10: 1-4 OFF 2-3 ON	SW10: 1-4 ON 2-3 OFF	SW10: 1-4 ON 2-3 ON	SW10: 1-4 OFF 2-3 OFF	JP45 to 1-2

To connect the board, follow the steps below:

1. Switch the board to ISP Mode and reset (SW2). For more information, see **Table: Boot configurations of the RT7xx boards** in Connecting the board for RT7xx devices.
2. Connect your PC to the **U7** port using the USB cable.
3. Ensure you have started SEC with a new workspace. For more information, see Setting up Secure Provisioning Tool.
4. Set the connection to **UART** and test the board connection.
   Note: USB is not supported yet.

6.15.3 Booting images for RT7xx devices

This chapter describes the building and writing of plain and signed bootable images.

By default, the workspace is configured to boot from external flash. To boot from eMMC, select **eMMC** in the **main menu > Target > Boot memory**.

The image located in XSPI0 flash or eMMC can be copied into XSPI1 external RAM (PSRAM) during boot and executed from there. XMCD configuration must be used to initialize the external RAM. For the EVK board, the XMCD configuration is distributed within the SEC Tool.

Table: Combinations of used memories for RT7xx boot image

Memory where the image is executed	Memory where the image is written	XMCD needed	XIP
External NOR flash, XSPI0	External NOR flash, XSPI0	No	Yes
External NOR flash, XSPI1	External NOR flash, XSPI1	No	Yes
Internal RAM	XSPI0, XSPI1 or eMMC	No	No
External RAM, XSPI1	XSPI0 or eMMC	Yes	No

6.15.3.1 Life cycle for RT7xx devices

There are two types of life cycles supported: **shadows** and **OTP**. In the **shadows**, the OTP fuses are not burnt and instead shadow registers are used. The shadow registers are initialized via a debug probe so during write the debug probe must be connected with an ISP communication cable. The shadow registers can be used only for development, not for production.

6.15.3.2 Booting a plain unsigned/CRC image

Plain images are typically used for development. It is recommended to start with this boot type before working with secured images to verify that the executable image works properly.

To build a bootable image, follow these steps:

1. In the toolbar, select **Plain unsigned** or **Plain with CRC** as **Boot type**.
2. In the toolbar, set the life cycle to **Development, OTP**. Shadow registers will be used later.
3. Switch to the **Build image** view.
4. Select the image build in Preparing source image for RT7xx devices as a **Source executable image**. If needed, open the **Dual image boot** and configure. If configured, open **OTP configuration** and review all reported problems.
5. Click the **Build image** button to build a bootable image.

When the bootable image is successfully built:

1. Connect the board, see Connecting the board for RT7xx devices.
2. Switch to the **Write image** view.
3. Click the **Write image** button.
4. If the write operation was successful, switch boot mode and reset the board.

If the write operation was successful, switch boot mode to **XSPI boot**(see **Table: Boot configurations of the RT7xx boards** in Connecting the board for RT7xx devices.) and reset the board.

6.15.3.3 Booting encrypted unsigned or CRC image

Encryption is supported for the image running from the extranal Flash. To configure encryption for the plain or CRC image, follow the steps below:

1. Switch **Boot type** to **Encrypted (IPED) unsigned** or **Encrypted (IPED) CRC**
2. Open **IPED regions** configuration on the Build image view and review the settings. By default, the whole image is encrypted automatically. Use CTR or GCM encryption mode.
3. Click the **Build image** button to build a bootable image. **Note:** The image is not encrypted yet, it will be encrypted during writing into flash.

When the bootable image is successfully built, write it into the board the same way as the image in the previous chapter.

In case dual boot is used, it is recommended to set manually same encryption for both regions.

6.15.3.4 Using RT7xx shadow registers

As a next step, it is recommended to try shadow registes via debug probe.

1. Use the configuration from the previous step.
2. In the toolbar, change the life cycle to **Development, shadows**.
3. Select command **main menu > Target > Debug Probe** and select PyOCD debug probe. The debug probe is connected by the same cable as the UART (both communicate in parallel).

4. Click the **Build image** button to build a bootable image.

When the bootable image is successfully built, write it into the board the same way as the image in the previous chapter.

The high level actions that are applied in the write script with shadow registers:

Step No.	Unsigned and CRC boot types	Signed boot types
Step 1:	flash configuration	flash configuration
Step 2:	writes the application to the flash	applies device HSM SB file to set CUST-MK-SK shadow regs
Step 3:	applies the shadow regs and resets the chip	applies the shadow regs and resets the chip
Step 4:		applies app. SB file to write the application image

6.15.3.5 Booting signed image using shadow registers

This section describes the building and writing of an authenticated image.

A signed image cannot be used with the **Development, OTP** life cycle because the ROM does not contain any command to burn CUST-MK-SK without advancing the life cycle.

To build a signed image, use the following steps:

1. In the toolbar, set the **Boot type** to **Plain signed** or **Encrypted (IPED) signed**.
2. In the toolbar, set life cycle to **Development, shadows**, **InField, shadows** or **InField-Locked, shadows**.
3. In the **Build image** view, use the image from Preparing source image for RT7xx devices as a **Source executable image**.
4. Ensure you have keys generated in the **PKI management** view. For more information, see PKI management.
5. For **Authentication key**, select any key, for example, *ROT1: IMG1_1*.
6. Generate a random **CUST-MK-SK** and **OEM seed**.
7. If needed, open **Dual image boot** and configure.
8. Open **OTP configuration** and review all reported problems.
9. Click the **Build image** button and check that the bootable image was built successfully. During the build, the tool communicates with the processor to create the device HSM SB file. This does not cause any changes in the processor or in the flash memory.

To write the image, do the following:

1. Switch to **Write image** view.
2. Ensure the debug probe is still selected (see toolbar).
3. Make sure that the board is set to Serial bootloader (ISP) mode. For more information, see RT7xx device workflow.
4. Click the **Write image** button.
5. If the write operation was successful, switch boot mode and do soft reset (button SW2) the board.

6.15.3.6 Device HSM

The table below shows how the shadows/fuses are configured in signed boot types:

Fuse	Any shadows life-cycle	InField, OTP life-cycle	Development, OTP life-cycle
CUST-MK_SK	The device HSM SB file is used to initialize the shadow registers.	Device HSM SB file	Not supported
Other fuses	OTP shadow registers are initialized via the debug probe.	Device HSM SB file	Not supported

6.15.3.7 Pre-erase in write script

For development purposes, the write script supports the pre-erase parameter. It allows erasing flash for the following reasons:

- remove any previous IPED configuration
- remove bootable image, so the processor goes into ISP mode after reset (for the case when reset is needed during provisioning)

The pre-erase parameter is used in development in GUI only, it is not used in the manufacturing, where it is expected to use empty flash.

6.16 RW61x device workflow

This chapter describes the workflow for RW61x processors.

6.16.1 Preparing source image for RW61x devices

The only available boot memory for the RT61x processor is external flash.

Optional step: to create the source image for the build in SEC Tool, you can disable the boot header by defining the symbol `BOOT_HEADER_ENABLE` to `0` (in MCUXpresso IDE, go to **Project > Properties > C/C++ Build > Settings > MCU C Compiler > Preprocessor > Defined symbols** and set `BOOT_HEADER_ENABLE` to `0`). From SEC Tool v9, this step is optional. The SEC Tool can parse the bootable image and retrieve the application image from the bootable image.

The image shall be at address 0x8001000 (default in MCUXpresso SDK examples).

6.16.2 Connecting the board for RW61x devices

This section contains information about configuring the evaluation boards RD-RW612-BGA or RD-RW61x-QFP or FRDM-RW612 and connecting it to SEC.

1. Select ISP boot mode, see **Table: Boot configuration of RW61x boards** below.
2. Connect the J7 port (on FRDM, it is J10) to your PC with a USB cable.
3. Ensure SEC runs with a workspace created for the chosen device. For more information, see Setting up Secure Provisioning Tool.
4. Go to **main menu > Target > Connection** and select UART and test the connection.

6.16.2.1 Table: Boot configuration of RW61x boards

Board	In-System Programming(ISP) Boot	Boot from external FLASH
RD-RW612-BGA	U38 switch: 0001	U38 switch: 0000
RD-RW612-QFP	U38 switch: 0001	U38 switch: 0000

Board	In-System Programming(ISP) Boot	Boot from external FLASH
FRDM-RW612	Hold the **SW3** button during reset.	Reset without the **ISP** button.

6.16.3 Shadow registers for fuses via debug probe

The SEC Tool uses **In-Field, shadow registers** as the default life cycle. It means that the **In-Field** life cycle will be configured using shadow registers. For development, start with shadow registers to avoid irreversible changes in fuses, but this should not be used for production. The shadow registers configuration is done via a debug probe, so the probe must be selected first:

1. Go to the **main menu >Target > Debug Probe** to open a dialog for debug probe selection.
2. Select a debug probe from the drop-down menu
3. Switch the board switch to boot mode from an external flash and reset the processor; click “Test connection” to check the connection with the debug probe

The processor does not allow using shadow registers in the “Develop” life cycle, so by default, the shadow registers for the life cycle will configure the “In-Field” state.

Also, to make image booting using shadow registers, it is necessary to configure the boot source in BOOT_CFG0 shadow register. As the fuse is locked after the first write, the tool will ask you to specify all the other bits (even for shadow registers). Go to **OTP configuration** to specify them.

In the shadow registers life cycle, the fuses shadow registers are configured immediately after the write image is successfully finished. The SEC Tool launches the `write_shadows` script that will set the shadow registers and then resets the processor. After the reset, the processor boots the image.

For the encrypted mode, the application is written via the SB file, so RKTH and CUST_MK_SK fuses must be programmed (even if the **shadow registers** life cycle is selected).

6.16.4 Booting images for RW61x devices

This section describes building and writing bootable images into the external flash and booting.

The default SEC Tool configuration is prepared for the EVK board. If you are using an FRDM board, use the FlexSPI configuration from the W25Q512 template. In case the flash is locked for write, it is possible to unlock it using the FCB provided in `sample_data`:

1. Select **main menu > Target > Boot Memory.**
2. Select FlexSPI NOR - complete FCB.
3. Select the FCB file from the disk.
4. Select the file `sample_data\targets\RW612\source_images\frdmrw612_wb_reset_fcb_quad_spi_v2.fcb`. This file must not be used for production.

6.16.4.1 Booting plain or CRC image

Plain images are typically used for development. Start with this boot type before working with secured images to verify that the executable image works properly.

First, build a bootable image:

1. Make sure you have selected the **Plain unsigned** or **Plain with CRC** boot type in the toolbar.
2. Switch to the **Build image** view.
3. Select an image built in Preparing source image for LPC55(S)0x/1x/2x/6x devices as a **Source executable image**.
4. If there is a binary image, set the start address to 0x8001000.
5. Configure `BOOT_CFG0` fuse (see the previous paragraph).

6. If needed, open **Dual image boot** and configure.
7. Click the **Build image** button to build a bootable image. The result is a binary bootable image.

When the bootable image is built, upload it to the processor:

1. Make sure that the processor is in ISP mode.
2. Switch to the **Write image** view.
3. Click the **Write image** button.
4. If the write operation was successful, reset the board.
   - the image boots automatically if shadow registers are selected
   - otherwise switch boot mode (see **Table: Boot configuration of RW61x boards** in Connecting the board for RW61x devices) and reset the board to boot the image.

6.16.4.2 Booting plain signed image

This section describes building and writing a plain signed image.

Build a bootable image:

1. Select the **Plain signed** boot type in the toolbar.
2. Switch to the **Build image** view.
3. Select an image built in Preparing source image for LPC55(S)0x/1x/2x/6x devices as a **Source executable image**.
4. For **Authentication key** select any key, for example ROT1: IMG1_1
5. Use random value for “CUST_MK_SK” and “OEM seed” symmetric keys.
6. Ensure that there is no error in OTP configuration. For ECC p384 key length you will need to configure the BOOT_CFG3 fuse.
7. Make sure that the board is connected and the processor is in ISP mode. During building processes, provisioning SB3 file for installation of CUST_MK_SK into processor is prepared.
   Note: The processor is reset after the SB file is built.
8. Keep **In-Field, shadow regs** life cycle to avoid irreversible changes in the processor
9. Click the **Build image** button to build a bootable image. The result is a binary bootable image and SB3 capsule for installation of the image into the processor.

When the bootable image and SB3 capsule have been successfully built, you can upload to the processor:

1. Make sure that the processor is in ISP mode.
2. Switch to the **Write image** view.
3. Click the **Write image** button.

In the shadow register life cycle, no fuses are burnt. The signed application is written into the flash and then shadow registers are applied and the processor is reset to start the application.

Once you advance to the life cycle without shadow registers, the fuses will be burnt irreversibly and the SB3 capsule will be used to write the application into the flash.

6.16.4.3 Booting encrypted image

Encrypted images with plain, CRC, or signed images are supported. The process of creation an encrypted image is similar to a signed image.

In addition, it is necessary to configure encrypted regions via the **IPED Regions** button on the **Build image** view (by default, the whole application is encrypted). IPED region alignment is based on the page size of the target FLASH, which is retrieved from FCB flash configuration. See Boot memory configuration for details on how to convert Flex-SPI NOR simplified configuration to full FCB configuration.

Due to ROM limitations, the image must be always located inside the IPED region, it is not allowed to erase or write range that exceeds the encrypted region.

In combination with the dual boot, configure encryption for image0 only, and the same settings are applied for image1.

Image encryption is performed when the image is written to the target memory. The application is written via SB file, so RKTH and CUST_MK_SK fuses must be burnt, so the processor can decrypt and authenticate SB file content. Mind these fuses will be burnt even if the shadow registers are selected.

The encrypted region is configured in the SB file. The decrypted regions are configured in fuses, so make sure these two are aligned.

6.16.4.4 Device HSM and provisioning

The CUST_MK_SK is a customer key for SB file encryption/decryption and this key can be installed into the processor only using device HSM SB file. The key is stored in the fuses, so the installation is irreversible and once the fuse is written it is locked for write (the lock is also used to detect if the key is already installed or not). In the write script, there is an argument to decide whether the key shall be burnt or not.

To apply device HSM into the processor, it is necessary to use a device HSM loader FW distributed in restricted data (for details, see Preferences.

The content of device HSM depends on the life cycle, see next chapter for details.

6.16.4.5 Boot type and life cycle for RW61x

The following table provides an overview of fuses burnt by the write script for different configurations of life cycle and boot type and contains information on whether the SB3 capsule can be used to update the application image.

	Shadow regs life cycle	Develop life cycle	In-Field life cycle
Plain unsigned or CRC boot type	- No fuses burnt - SB file not used	- Fuses burnt, see OTP Configuration dialog - SB file not used	- Same as develop + life cycle fuse burnt - SB file not used
Plain signed boot type	- No fuses burnt - SB file not used, however it is generated during build	- Fuses burnt, see OTP Configuration dialog - RKTH and CUST_MK_SK burnt - SB file used	- Same as develop + life cycle fuse burnt - SB file used
Encrypted boot type	- RKTH and CUST_MK_SK burnt - SB file used - No other fuses burnt	- Fuses burnt, see OTP Configuration dialog - IPED, RKTH, and CUST_MK_SK burnt - SB file used	- Same as develop + life cycle fuse burnt - SB file used

6.16.4.6 Life cycle and trust provisioning

The table below shows the security assets installed in different life cycles and trust provisioning types:

	Device HSM Develop	Device HSM In Field	EdgeLock 2GO Develop	EdgeLock 2GO In Field
CUST MK SK	Device HSM SB file	Device HSM SB file	CUST-MK_SK	CUST-MK_SK
RKTH fuses	Write script	Device HSM SB file	el2go_provi_otp.bin	el2go_provi_otp.bin
Other custom fuses	Write script	Device HSM SB file	el2go_provi_otp.bin	el2go_provi_otp.bin

6.16.4.7 EdgeLock 2GO

EdgeLock 2GO is described in EdgeLock 2GO trust provisioning workflow. Below are RW61x-specific comments:

- The provisioning firmware does not work if the processor boots into ISP mode. The processor must be in RUN mode and the external flash must be erased. After resetting the processor, as there is no application in the flash, the processor will fall back into ISP mode and EdgeLock 2GO provisioning can be started.
- The address for secure objects must be in external flash. After the provisioning is finished, all secure objects (even secure objects that are already installed) remain in the memory.
- In case of the Develop life cycle, the provisioning works only in **DRY RUN** mode, allowing you to test the process without affecting the processor. In this mode, the secure objects are applied to the processor (external flash) and the provisioning firmware is executed, but the firmware just verifies the secure objects and does not burn fuses and does not change the life cycle. The application image is **not provisioned**, as the SB file cannot be loaded yet.
- To upload the SB3 file into the EdgeLock 2GO server as a secure object, use the following parameters:
  – Create New Secure Object
  – Binary File
  – Assign any name
  – Set the Object Identified (OID) to 0x7FFF817C
  – Non confidential file
  – Add Policy/Select Device Life Cycle: Closed
  – Select permitted algorithm: NONE
  – Policies for this profile: NONE For uploading secure objects to the EdgeLock 2GO server, use the **Closed** or **Closed/Locked** policy. The policy must be same for all secure objects, otherwise the provisioning fails. The **Open** policy is not a preferable option, as **DRY RUN** mode supports the **Closed** policies too.

7 Generic workflows

This section provides information on workflow for some typical use cases. These chapters are not specific for any processor.

7.1 Debug authentication workflow

This section describes the process of opening the debug port. This tool offers Debug Authentication Protocol (DAP) as a mechanism to authenticate the debugger (an external entity) for the field technician, which has the credentials approved by the product manufacturer (OEM) before granting the debug access to the device. For Debug Authentication (DA) to work, processor-specific fuses or PFR fields must be set. For more information see the device user manual, chapter Debug subsystems.

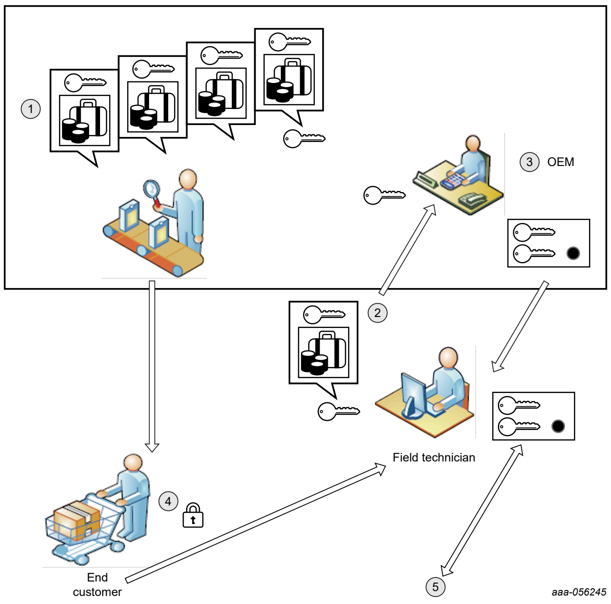
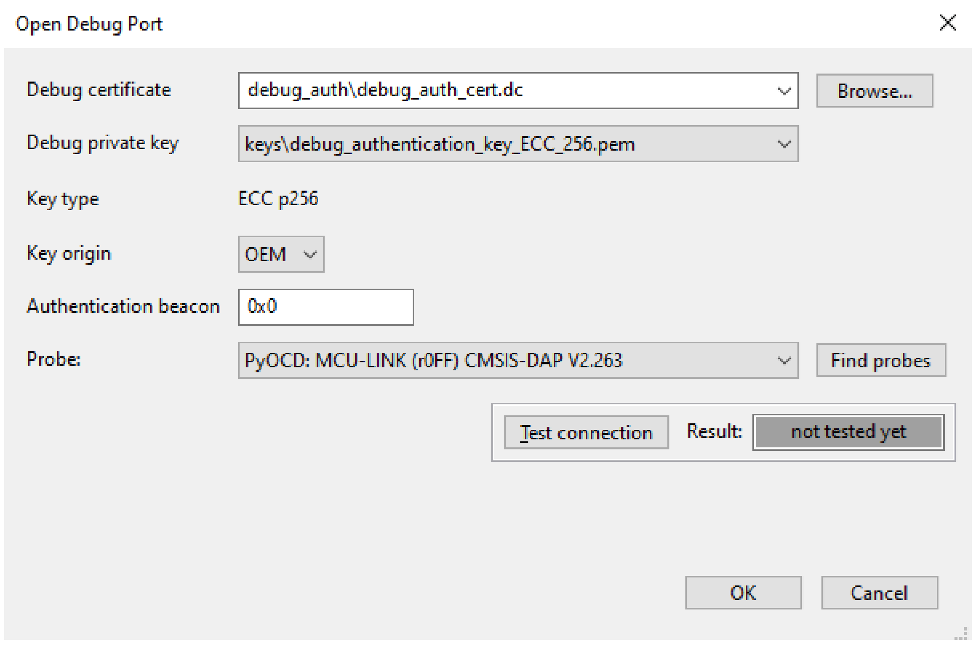


Figure 71. Debug Authentification protocol usage example

To open the debug port, do the following:

Field Technician

1. To get the key type and ROT length, contact the OEM. OEM decides whether to use the generated certificate for any device with the same ROT keys or just one by specifying the UUID.
2. In the SEC Tool, create a workspace for the used target device, switch to the “PKI Management” view
3. Click the **Generate a debug key…** button and verify that both the key type and length of the DA key match those of the corresponding ROT key.
4. Click the **Create debug certificate request…** button. Optionally, specify UUID to limit the use of the debug certificate. If the UUID is set to zero, it can be used for any device. UUID can be read from a device via UART or USB if the processor is in ISP mode and in the development life cycle. For processors in an advanced life cycle, the debug probe works. On most devices, CHECK_UUID must be set in the SOCU fuse/PFR field to enable verification of the UUID in the debug certificate.
5. Send the certificate request to OEM
6. After the certificate from OEM is received, click the **Open debug port…** button. The connected probes are detected upon the dialog display. The list of detected probes can be updated by clicking the **Find probes** button. Select one of the detected probes. The authentication beacon is an optional parameter and it is independent from the credential beacon provided by the OEM. It is not interpreted by the debug authentication protocol, it is passed to the debugged application. When the dialog is confirmed, a script is generated into workspace\debug_auth\open_debug_port.[bat|sh] and is executed. The dialog will be closed if no error is reported by the script (the operation is successful). In case of failure, refer to section Debug authentication for details on how to enable debug authentication.
   Note: nxpdebugmbox CLI tool can be found in `<installation_dir>/tools/spsdk/` folder



Open Debug Port

Debug certificate: debug_auth\debug_auth_cert.dc

Debug private key: keys\debug_authentication_key_ECC_256.pem

Key type: ECC p256

Key origin: OEM

Authentication beacon: 0x0

Probe: PyOCD: MCU-LINK (r0FF) CMSIS-DAP V2.263

Result: not tested yet

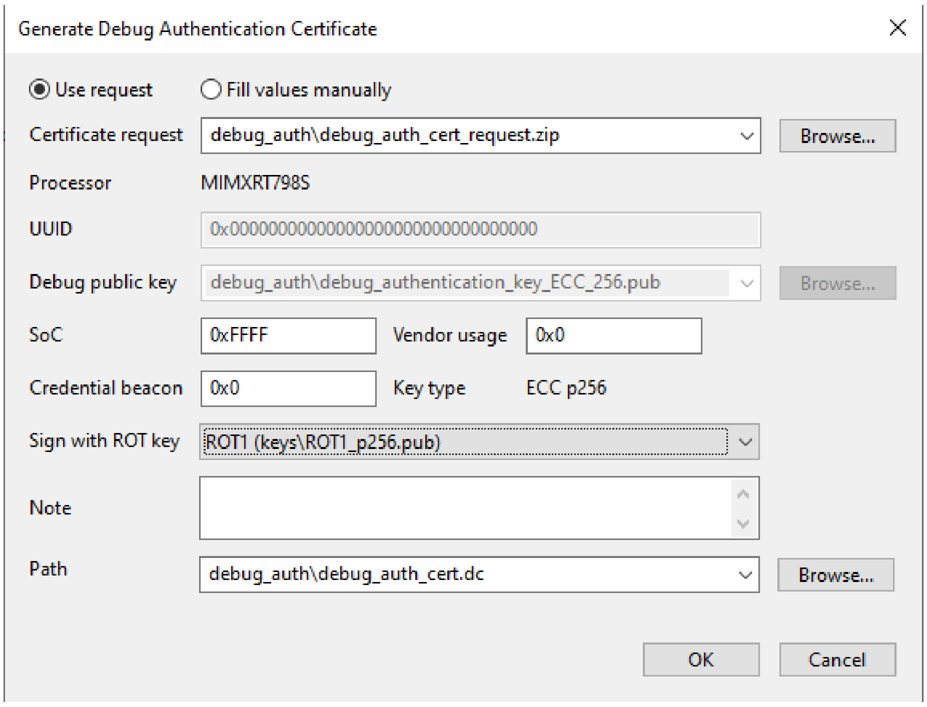
Figure 72. Dialog for opening debug port

OEM Developer

After receiving the request, click the **Generate debug certificate…** button.

The certificate is by default generated into `<workspace>/debug_auth`folder, `debug_auth_cert.dc`. A *.zip archive with the same name is created, it contains the certificate and the `.txt` file note from OEM. The note that is passed from field technician to OEM is displayed in the note field (see Figure “Generation of debug certificate from certificate request”).

- **SoC** - mask value of `DCFG_CC_SOCU` controlling which debug domains are accessed via the authentication protocol
- **Vendor usage** - field that can be used to define a vendor-specific debug policy. The use case can be Debug Credential (DC) certificate revocations, the department identifier, or the model identifier.
- **Credential beacon** - value that is not interpreted by DAP, it is passed to the application. The value is independent of the authentication beacon that will be provided by the field technician when the port is opened.
- **Note** - text field where OEM can describe comments about reasons to generate the certificate
- **Sign with ROT key** - sign the certificate with one of the ROT keys that were used to secure the device.


Generate Debug Authentication Certificate

[✓ Use request] Fill values manually

Certificate request	debug_auth\debug_auth_cert_request.zip
Processor	MIMXRT798S
UUID	0x00000000000000000000000000000000
Debug public key	debug_auth\debug_authentication_key_ECC_256.pub
SoC	0xFFFF
Vendor usage	0x0
Credential beacon	0x0
Key type	ECC p256
Sign with ROT key	ROT1 (keys\ROT1_p256.pub)
Note	
Path	debug_auth\debug_auth_cert.dc

Figure 73. Generation of debug certificate from certificate request

7.1.1 Example of access rights to debug domains

Examples are intended for testing purposes. Before the final usage, the setting should be revisited and modified to fulfill security requirements. In all examples below, ISP is enabled and UUID check is disabled. For some processors, UUID check must be set to enable the read of UUID by a debug probe.

Table KW45xx/K32W1xx and MCX W71x

Fuse	Everything disabled	Everything enabled	Controlled by DA
DCFG_CC_SOCU_L1, DCFG_CC_SOCU_L2	0x000000FF	0x0000FFFF	0x00004040
DBG_AUTH_DIS	0x0	0x0	0x0

Table KW47xx and MCX W72x

Fuse	Everything disabled	Everything enabled	Controlled by DA
DCFG_CC_SOCU_L1, DCFG_CC_SOCU_L2	0x000001FF	0x0003FFFF	0x00008040
DBG_AUTH_DIS	0x0	0x0	0x0

Table LPC55S0x/1x, MCXW236 and NHS52S04

PFR field	Everything disabled	Everything enabled	Controlled by DA
DCFG_CC_SOCU_NS_PIN, DCFG_CC_SOCU_PIN	0xFF2000DF	0xFF2000DF	0xFFBF0040

PFR field	Everything disabled	Everything enabled	Controlled by DA
DCFG_CC_SOCU_NS_DFLT, DCFG_CC_SOCU_DFLT	0xFFFF0000	0xFF2000DF	0xFFBF0040

Table LPC55S2x

PFR field	Everything disabled	Everything enabled	Controlled by DA
DCFG_CC_SOCU_NS_PIN, DCFG_CC_SOCU_PIN	0xFF2000DF	0xFF2C00D3	0xFFBF0040
DCFG_CC_SOCU_NS_DFLT, DCFG_CC_SOCU_DFLT	0xFFFF0000	0xFF2C00D3	0xFFBF0040

Table LPC55S3x

PFR field	Everything disabled	Everything enabled*	Controlled by DA
DCFG_CC_SOCU_NS_PIN, DCFG_CC_SOCU_PIN	0xFE3001CF	0xFE3001CF	0xFFFF0000
DCFG_CC_SOCU_NS_DFLT, DCFG_CC_SOCU_DFLT	0xFFBF0040	0xFE3001CF	0xFFFF0000

Note: For debugging, authentication is still required but the domain cannot be disabled by the SoC mask in the DAC.

Table LPC55S6x

PFR field	Everything disabled	Everything enabled	Controlled by DA
DCFG_CC_SOCU_NS_PIN, DCFG_CC_SOCU_PIN	0xFD0002FF	0xFD0002FF	0xFFBF0040
DCFG_CC_SOCU_NS_DFLT, DCFG_CC_SOCU_DFLT	0xFFBF0040	0xFD0002FF	0xFFBF0040

Table MCX Nx4x and MCX N23x

PFR field	Everything disabled	Everything enabled*	Controlled by DA
DCFG_CC_SOCU_NS_PIN, DCFG_CC_SOCU_PIN	0xF81007EF	0xF81007EF	0xFFBF0040
DCFG_CC_SOCU_NS_DFLT, DCFG_CC_SOCU_DFLT	0xFFBF0040	0xF81007EF	0xFFBF0040

Note: For debugging, authentication is still required but the domain cannot be disabled by the SoC mask in the DAC.

Table RT5xx/6xx

Fuse	Everything disabled	Everything enabled	Controlled by DA
DCFG_CC_SOCU, DCFG_CC_SOCU_NS	0x80FF408D	0x80FFFF20	0x00404088
DCFG_CC_SOCU_AP	0x7F00BF72	0x7F0000DF	0xFFBFBF77

Table RW61x

Fuse	Everything disabled	Everything enabled*	Controlled by DA
DCFG_CC_SOCU, DCFG_CC_SOCU_NS	0x3FFA007E	0x3FFFFF14	0x1002000F
DCFG_CC_SOCU_AP	0xC005FF81	0xC00000EB	0xEFFDFFF0

Note: For debugging, authentication is still required but the domain cannot be disabled by the SoC mask in the DAC.

RT118x does not have any fuse to control debugging rights. Debugging depends on the LC, for OEM_OPEN: all debug allowed, OEM_CLOSE: all closed but can be enabled by DAC, and OEM_LOCKED: all closed and cannot be enabled. The only way to manage debugging rights in OEM_CLOSE is by setting the SoC in DAC. For examples of SoC masks, see the device user manual.

7.2 Signature provider workflow

This section describes the process of setting up signature provider and building an image signed by the signature provider. There are examples of the signature provider server located in `<install_folder>/sample_data/signature_provider_examples`, one working with ROT ECC keys and the other with ROT RSA keys. These examples demonstrate the full implementation of the API; however in the real world, it is expected that the implementation will be changed by communication with HW HSM module or custom HTTPS communication to another server. Both examples of the server can be used as they are to test tool behavior when using the signature provider. Each server has example private keys and prepared public key response for the `public_keys_certs` endpoint. A prepared ECC/RSA public tree have 4 ROT keys/certs and each ROT key/cert has one IMG key/cert. These keys should not be used in final products.

The figure below displays variants of signature provider, SEC Tool send requests to Custom signature provider HTTP server. This server should pass the request to one of the secure solutions and then pass the response back to the SEC Tool. Prepared examples implement only the Custom signature provider HTTP server. The example server is doing all the operation that should be done by an HSM or external signature provider. It is up to the user to implement a complete solution.

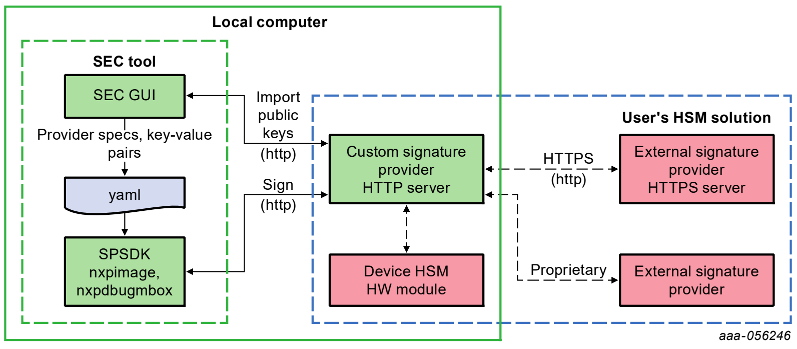

Local computer
SEC tool
SEC GUI
Provider specs, key-value pairs
yaml
SPSDK nxpimage, nxpdbugmbox
Import public keys (http)
Sign (http)
User's HSM solution
Custom signature provider HTTP server
Device HSM HW module
HTTPS (http)
Proprietary
External signature provider HTTPS server
External signature provider
aaa-056246

Figure 74. Expected structure of signature provider

7.2.1 Run the server

Ensure python 3.12 or higher is installed. Open the directory where the server implementation and specify version of SPSDK package in `requirements.txt` to be used (use same version as listed in **main menu > Help > About**) and run the following commands:

```
# ensure, venv is installed
pip install virtualenv
# create virtual environment into .venv subfolder
python -m venv ".venv"
# activate it
.venv\Scripts\activate
# install all required packages into venv
pip install -r requirements.txt
# start the server
python server.py
```

The server logs every action, so it is possible to review what actions were executed.

7.2.2 Set up in the SEC Tool

To use the signature provider example with the SEC Tool, follow these steps:

1. Create/use workspace for the processor.
2. Select the check-box **Use sign. provider** on the PKI tab. If there are keys in the workspace, they are moved to a back-up subfolder in the workspace.
3. Open the signature provider dialog by clicking **Configure…** next to the check-box from step 2.
4. Review the default parameters of your signature provider. If using the signature provider server from `resources\signature_provider_examples`, the default settings will work.
5. Select the prehash option in the parameters table. When prehash is enabled, only a hash of the data is sent to the server for signing.
6. Click the **Test connection** button to verify if the server is configured properly.
7. There are two options to Import public keys:
   - In the same dialog, click the **Import public keys** button to import public keys from the server; this is a recommended way, however it can be used only if the server implements optional API **public_keys_certs**.
   - If the **Import public keys** command is not supported, the alternative way is to use **Import keys** from the **PKI management** tab. Make sure that public keys match private keys that are used on the signature provider site (copy the keys to the folder with the signature provider example).
8. On the **Build** tab, select the key as normally, now the config files for SB, MBI, and certification block will be using the signature provider configuration.
9. Now, the signature provider is configured. It is possible to build a signed image.

7.2.2.1 Signature provider and debug authentication

The SEC Tool supports signing of the DA certificate by the signature provider. The field technician flow of the DA key generation is the same as without the signature provider; the debug key pair generation is supported only locally. The only difference is that in the DAC generation on the OEM site, the signature provider is used for signing. This is transparent to the user as the public key is selected from the workspace keys. The only difference is in the generated configuration file, where the **sign_provider** field is set according to the signature provider setting.

7.3 Script hooks workflow

Script hooks are executed before or during build, write, and manufacturing script execution. Script hooks enable script customization outside the generated scripts. Hook scripts are located in the “hooks” subfolder and the new workspace contain the examples for the selected processor. Script hooks are not generated by the tool; they are expected to be fully under user control. If the hook script does not exist, it can be created with a single click in the GUI.

7.3.1 Build script hooks

1. Pre-build hook: `pre_build_<os-name>.bat/sh` is executed before the build script. This script can be generated to sign the MCUboot application image. This script does not have any argument and it is called only from the GUI before the build script is executed.
2. Build context hook: `build_context_<os-name>.bat/sh` is called at the beginning of the build script and allows users to add or modify the environment variables. This hook is called without arguments.
3. Build hook: the build_.bat/sh script is executed after each main step in the build script. This script is called from the build script and the name of the previous step is passed as an argument. All the supported steps are handled in the generated examples. If a hook step call fails, the build script execution stops and exits with an error.

7.3.2 Write script hooks

1. Write the context hook: the `write_context_<os-name>.bat/sh` script is called at the beginning of the write script and allows users to add or modify the environment variables. This hook is called without arguments.
2. Write the hook: the `write_<os-name>.bat/sh` script is executed after every main step in the write script. The script is called from the write script and the name of the previous step is passed as an argument. All the supported steps are handled in the generated example. If a hook step call fails, the write script execution stops and exits with an error.

7.3.3 Manufacturing hooks

Manufacturing hook: `manufacturing_<os-name>.bat/sh` is called at the beginning and end of the manufacturing process before the first task is started and after the last task is finished. If manufacturing hook execution for step **started** fails, planned manufacturing steps are not executed. The call for step **finished** has an additional argument **status** that can have two values **ok** or **fail** that denote the execution status of the manufacturing process: **ok** if all tasks were finished successfully, **fail** otherwise.

7.3.4 Typical usage

Here are a few examples how the hook scripts can be used to customize the build, write, or manufacturing script without the modification of the script generated from the SEC Tool:

- Update the input source file(s) for the build.
- Fix the problem in the write script or apply an additional action that updates the previous action.
- Synchronize the manufacturing operation with the assembly line.

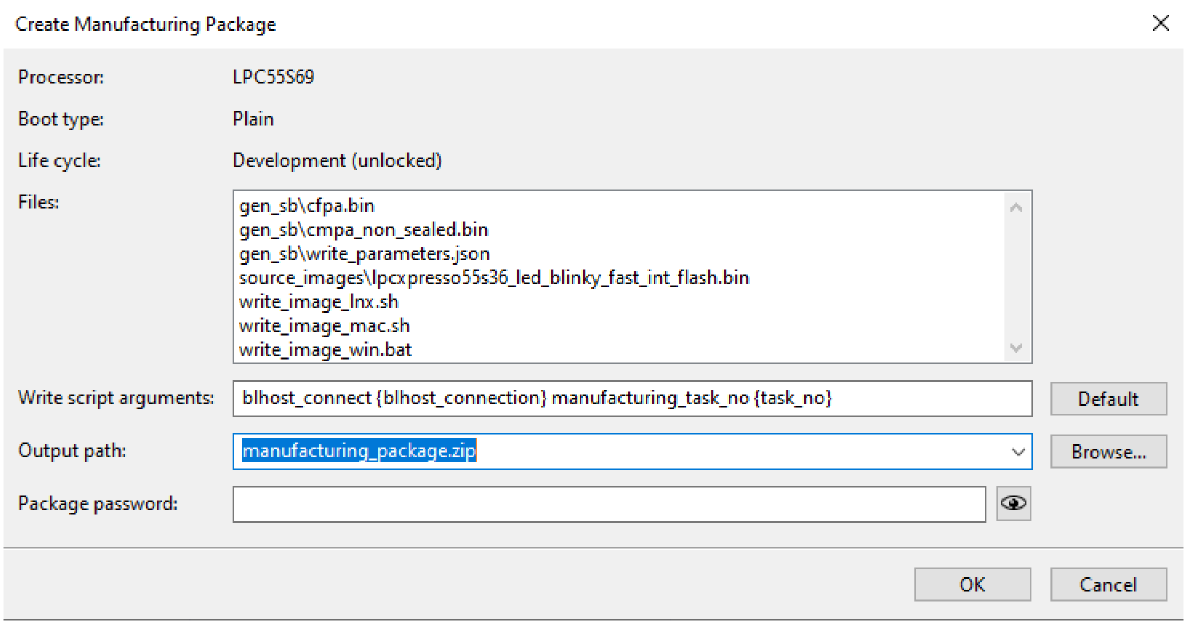
7.4 Manufacturing workflow

For the manufacturing operations, the tool offers a simplified user interface focused on the manufacturing only.

7.4.1 Create manufacturing package

OEM can create a manufacturing package (*.zip) that can be used to transfer files to the factory. The manufacturing package can be created:

- On the **Write image** view - for provisioning using the current write script
- In the **SB editor** - to apply commands specified in the SB file



Processor: LPC55S69

Boot type: Plain

Life cycle: Development (unlocked)

Files:

```
gen_sb\cfpa.bin
gen_sb\cmpa_non_sealed.bin
gen_sb\write_parameters.json
source_images\lpcxpresso55s36_led_blinky_fast_int_flash.bin
write_image_lnx.sh
write_image_mac.sh
write_image_win.bat
```

Write script arguments: blhost_connect {blhost_connection} manufacturing_task_no {task_no}

Output path: manufacturing_package.zip

Package password:

Figure 75. Create Manufacturing Package dialog

The **Create Manufacturing Package** dialog allows the user to:

- Review files included in the package.
- Check the write script arguments, the arguments are in the same format as they are used in Manufacturing Tool.
- Select the output manufacturing package file path.
- Set the password for the ZIP file. The AES algorithm is used to encrypt files in the ZIP file; however, the file metadata (like file names and file sizes) are not encrypted.
- For the EdgeLock 2GO provisioning, it is possible to optionally specify the API key in the manufacturing package, but it can be specified later during manufacturing

7.4.2 Performance optimization

It is supposed that the provisioning/write script will be executed several times to address different development issues, so the script is designed primary for this use case and may contain some parts that are not necessary in manufacturing when the provisioning is applied to an empty processor. Here are some tips to improve manufacturing performance by manually modifying the generated script:

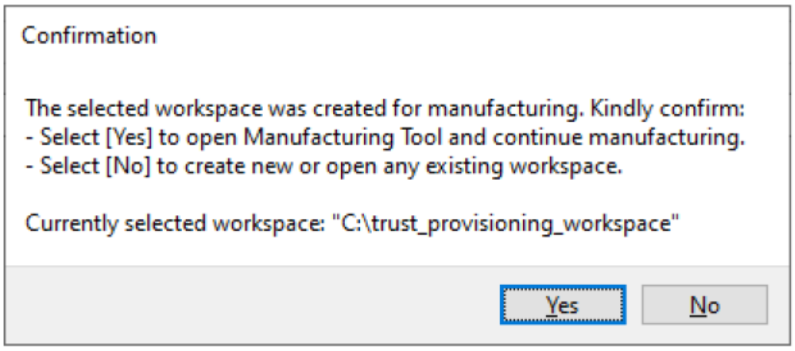
- It might not be needed to erase boot memory. If the processor is new, the flash is already erased.
- It might not be needed to detect if the chip is already secured; this part in the write script allows updating application for secured processors (for chips where it is supported)
- A longer sequence of **blhost** operations might not be optimal, because the **blhost** initialization time is not insignificant. It might be better to move the operations into the SB file (see SB editor) or use **blhost batch** command (see SPSDK documentation for details).

- If the processor is reset during manufacturing, there is a delay time until it boots and the OS driver reconnects. By default it is 3 seconds. The time can be adjusted in preferences.

7.4.3 Manufacturing operations

On the manufacturing site, the manufacturing package can be imported using **main menu > File > Import Manufacturing Package…**. During import, the files from the package are extracted into the new workspace - called **manufacturing workspace**. For more information about importing the package, see Import workspace from the ZIP.

Once the workspace is created (or reopened), the Manufacturing Tool (for details, see Manufacturing Tool) is displayed, and the rest of the tool functionality is not available. If the Manufacturing Tool is closed, the whole tool operation is finished. If you restart the tool, it offers to continue manufacturing, or to select another workspace:


Confirmation

The selected workspace was created for manufacturing. Kindly confirm:
- Select [Yes] to open Manufacturing Tool and continue manufacturing.
- Select [No] to create new or open any existing workspace.

Currently selected workspace: "C:\trust_provisioning_workspace"

Figure 76. Confirmation to reopen manufacturing

If the manufacturing package contains a write script, check if the `SPT_INSTALL_BIN` environment variable in the script points to the installation directory on the computer. If not, it is recommended to set the environment variable globally or update the write script manually.

For **serial** connection: Adjust the baud rate, the default value is 115200; however, for several processors, baud rates up to 1000000 were successfully tested.

7.4.4 Steps in the manufacturing production

1. Connect one or more processors via USB or serial line or I2C or SPI and click the Autodetect button to detect the connected devices. In case the tool detects devices that should not be affected by the manufacturing, such as serial ports used by other devices, disable them. Then click Test connection below to check the connection with all enabled processors and ensure the test pass.
2. To start the trust provisioning operation, click the Start button. Wait until all operations are finished.
3. If any problem is reported, click to the status cell to show the log and fix the problem.
4. Continue with step 1

The number of successfully provisioned devices is displayed on the bottom of the Manufacturing Tool window, but it is not 100% reliable.

7.4.5 Manufacturing logs

Manufacturing logs are stored in the manufacturing workspace, in the subdirectory `logs/YYYY-MM-DD/`. The log file name is `manufacturing_log_YYYY-MM-DD_hh-mm-ss_#.log`, where `#` represents index of the manufacturing task being executed in parallel. It is possible to export them using the **Export logs** button from the manufacturing window. The export allows to select one day (a date) or all logs and exports the selected logs into the ZIP file.

7.5 MCUboot workflow

MCUboot is an open source secure bootloader for 32-bit microcontrollers. For details, see MCUboot Documentation. It has been ported to many NXP processors, and you can find it supported in the MCUXpresso SDK. For details, see `boards\<board>\ota_examples\mcuboot*`.

For most of the processors, the SEC Tool contains a default configuration compatible with project examples provided in the MCUXpresso SDK. Therefore, there is a “sample application” built from MCUXpresso SDK for the EVK/FRDM board and the default imgtool parameters match MCUXpresso SDK.

7.5.1 Steps to start MCUboot for NXP board via New Workspace

1. Open the **New Workspace** dialog and select the processor.
2. Connect the board via UART, ensure that the processor boot is in ISP mode. It is possible to use other connections, the UART is used because the mcuboot application is built with the debug console and prints the status that can be verified using the terminal.
3. Select the MCUboot bootloader image, and the corresponding sample application for the selected processor (including both the bootloader and executable image) will be automatically chosen. If no image is selected, it indicates that a sample application is not available for the selected processor. In such cases, refer to the manual configuration instructions below
4. Select the required profile with memory settings compatible with the sample application, and create the workspace. For MCX N series, IFR must be used.
5. The MCUboot Sign Image dialog will open with all required options preselected, so it is possible to save the options and sign the image.
6. Build and write the bootable image into the processor. **Note:** To verify everything works correctly, refer to step 9. of the manual configuration below.

7.5.2 Steps to start MCUboot with such a processor manually

1. Create a new workspace for the selected processor.
2. Connect the board via UART, ensure that the processor boot is in ISP mode, and check the connection (**main menu > Target > Connection**…). It is possible to use other connections, the UART is used because the mcuboot application is built with the debug console and prints the status that can be verified using the terminal.
3. Verify the selected boot memory in **main menu > Target > Boot Memory…**. For MCX N series, IFR must be used.
4. On the **Build image** view, select the “mcuboot opensource” application as “Source executable image”. The prebuilt application is available in `sample_data\targets\<processor>\source_images\<board>_mcuboot_opensource.s19`. You can also build your own from MCUXpresso SDK. If the application contains a configuration of the external flash (FCB), the tool detect and offers to use it. It is recommended to accept.
5. Open **main menu > Tools > MCUboot > Sign Image** and configure the following:
   - The secondary application to be signed; the prebuilt application is available in `sample_data\targets\<processor>\source_images\<board>_ota_mcuboot_basic.s19`
   - The signing key; it is located in the same folder that the prebuilt application or in MCUXpresso SDK, in folder `boards\<board>\ota_examples\mcuboot_opensource\keys\`. Depending on the platform. use either `sign-rsa2048-priv.pem` or `sign-ecdsa-p256-priv.pem`. Keys protected by a password are not supported yet.
   - The imgtool arguments by default should match the SDK example. It is not needed to change them.
   - Click the **Sign** button to sign the application; fix problems, if any.
   - Ensure that the check boxes below are selected:
     – the first check-box: the tool creates the pre-build hook script that signs the application before each build

    – second check-box: the tool reconfigures additional images, so the signed image is written to the target address; keep “Image 1”
  - Verify the target address of the application.
  - Close the dialog by clicking the **Save & Close** button.
6. On the **Build** tab, double-check that the **Build script hooks** section contains the pre-build script.
7. Open **Additional Images** and check that the signed application is properly configured as “Image 1”.
8. Build and write the bootable image into the processor.
9. Open a terminal (in MCUXpresso IDE go to **main menu > Windows > Show View > Other > Terminal**), select the UART port, confirm, and reset the processor. You should receive the following text:

```
hello sbl.
Bootloader Version 2.0.0
Primary   slot: version=1.0.0+0
Image 0 Secondary slot: Image not found
writing copy_done; fa_id=0 off=0xfffd0 (0xfffd0)
Image 0 loaded from the primary slot
Bootloader chainload address offset: 0x0
Reset_Handler address offset: 0x400
Jumping to the image

Booting the primary slot - flash remapping is disabled

*************************************
* Basic MCUBoot application example *
*************************************
Built Apr 16 YYYY 12:34:56
```

To sign and write a second copy of the application, follow additional steps:

1. Disconnect the terminal if it is still open. Reset the board to ISP mode.
2. Open **main menu > Tools > MCUboot > Sign Image**…
3. Keep the same input application image.
4. Change the output image name, for example add suffix 1_1
5. In imgtool arguments, change version to 1.1
6. In the **Set additional image** section change:
   - Image 2
   - Target address - increase the address by the slot size (see the slot size parameter in the imgtool arguments)
7. Save and Close.
   Note: The pre-build script will be overwritten and it will sign Image 2. The previously signed Image 1 remains on the disk untouched and it will be written to flash at the original location.
8. Open **Additional Images** and check that there are two applications: “Image 1” and “Image 2”.
9. Build and write.
10. Connect the terminal, switch the board to boot from the selected boot memory, and reset. The terminal should receive the following text:

```
hello sbl.
Bootloader Version 2.0.0
Primary   slot: version=1.0.0+0
Secondary slot: version=1.1.0+0
writing copy_done; fa_id=1 off=0xfffd0 (0x1fffd0)
Image 0 loaded from the secondary slot
Bootloader chainload address offset: 0x100000
Reset_Handler address offset: 0x100400
Jumping to the image
```

```
Booting the secondary slot - flash remapping is enabled

*************************************
* Basic MCUBoot application example *
*************************************

Built Apr 16 YYYY 13:24:56
```

7.6 EdgeLock 2GO trust provisioning workflow

EdgeLock 2GO is a fully managed cloud platform operated by NXP that provides secure provisioning services for easy deployment and maintenance of IoT devices using supported NXP products. The service allows creating and managing secure objects, such as symmetric keys, key-pairs, and certificates that are then securely provisioned into your NXP MCU or MPU.

The tool supports the following flows:

- **EdgeLock 2GO with device ID** flow, where manufacturing facility retrievs secure objects from EdgeLock 2GO cloud server during the production process
- **EdgeLock 2GO per product type** flow, where OEM retrieves secure objects database from EdgeLock 2GO server and production in manufacturing facility can operate without online access to EdgeLock 2GO server

7.6.1 EdgeLock 2GO with device ID, high-level description

The figures below correspond to the sequence of actions indicated on Figure “EdgeLock 2GO trust provisioning workflow”.

Configuration process

1. OEM prepares a secure configuration and verifies it locally. Then the SEC Tool provides all secure objects, and the OEM uploads them via the EdgeLock 2GO web portal.
2. OEM send to manufacturing facility manufacturing package generated from SEC Tool.

Manufacturing process for EdgeLock 2GO with device ID

1. The tool fetches the processor UUID
2. The tool requests secure objects for the given UUID. The EdgeLock 2GO server validates UUID, generates secure objects, and encrypts them with EdgeLock 2GO root of trust. Secure objects are downloaded to the manufacturing machine.
3. Secure objects and EdgeLock 2GO provisioning firmware are loaded to the target processor. The firmware is invoked to install the secure objects into the target location. The application image is applied into the target boot memory.

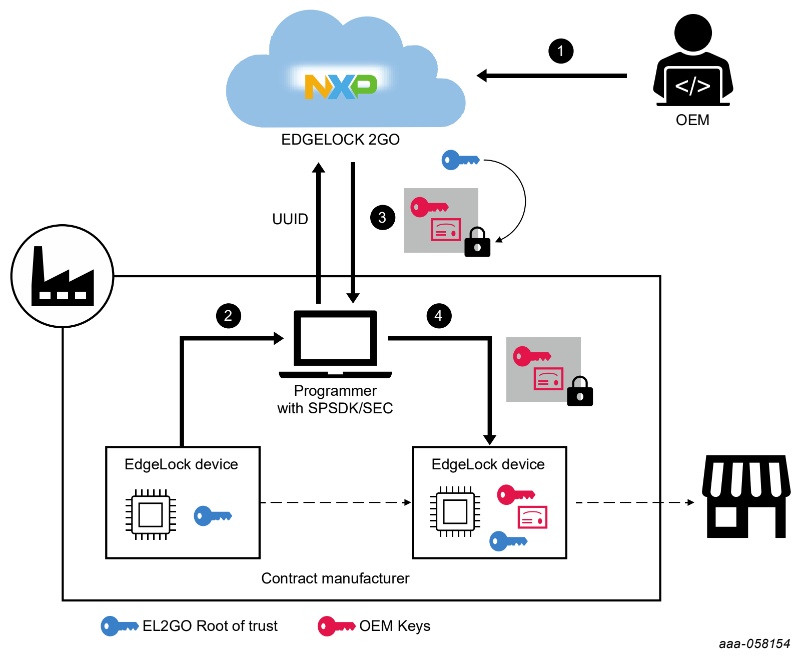


Figure 77. EdgeLock 2GO trust provisioning workflow

7.6.2 EdgeLock 2GO per product type, high-level description

Configuration process

1. OEM prepares a secure configuration and verifies it locally. Then the SEC Tool provides all secure objects, and the OEM uploads them via the EdgeLock 2GO web portal.
2. OEM downloads the database of secure objects and provides the database and manufacturing package to the manufacturing facility.

Manufacturing process for EdgeLock 2GO per product type

1. The tool fetches secure objects from the database.
2. Secure objects and EdgeLock 2GO provisioning firmware are loaded to the target processor. The firmware is invoked to install the secure objects into the target location. The application image is applied into the target boot memory.

7.6.3 EdgeLock 2GO flow, step by step

To configure EdgeLock 2GO provisioning, do the following steps:

1. Turn on secure boot mode and verify that the application works properly.
2. On the EdgeLock 2GO server, create the device group for the product and create an API key. For details, see EdgeLock 2GO Configuration via web portal.
3. In the **main menu > Target > Trust Provisioning Mode**, select **EdgeLock 2GO**. Select either **EdgeLock 2GO with device ID** or **EdgeLock 2GO per product type**. Fill the EdgeLock 2GO parameters and Test connection to the server.
4. Install the GnuPG Tool and take advantage of the prepared encryption script.
   OS-specific notes:
   - Windows: Kleopatra for Windows is recommended: Kleopatra for Windows (GPG4Win). The encryption can also be done both manually using “Kleopatra GUI”. See *EdgeLock 2GO Quick Start Guide* (document

AN13255 on EdgeLock 2GO server) or using the way described above (GnuPG utilities are installed with gpg4win).
- Ubuntu: use the gpg package in the Ubuntu repositories.
- macOS: gnupg available in the brew package system is a recommended option; GnuPG for OS X should also work (see GnuPG downloads)

5. Steps needed to enable the GPG encryption in the build script:
   - The encryption is done in the `el2go/encrypt_el2go_files.*` script generated into the workspace. Use the **Update files** button on the **Build** page to update the script.
   - Import the EdgeLock 2GO public key for encryption, see the generated script for how to do it
   - Generate RSA key pair for signing - **gpg -full-generate-key**. Use 2048 or 4096 key size.
   - Set environment variables
     - EL2GO_GPG_PASSWORD - password for the generated key pair
     - EL2GO_GPG_SIGN_KEY - identify which key should be used for signing using key ID available by **gpg-list-secret-keys**. If the EL2GO_GPG_SIGN_KEY environment variable is defined, the encryption is invoked from the build script as a part of the build action.
6. Ensure that the development life cycle is selected in the **main menu > Target > Life Cycle**. For development purposes, the EdgeLock 2GO verification is supported with the open life cycle. This is not intended for production.
7. On the build page, click **Build image** to build the application. Ensure that there are no errors.
8. Open the `<workspace>/el2go/publish` subfolder and upload all secure assets into the EdgeLock 2GO server. Follow the instructions provided in EdgeLock 2GO Configuration via web portal.
   Note: Files for the EdgeLock 2GO server are also displayed on the build page. In the generated files, they are marked by the el2go cloud icon. The description is displayed in the tooltip.
9. For EdgeLock 2GO per product flow, follow Create database with secure objects for EdgeLock 2GO per product type.
10. Go to the **Write** page and run the Write operation.
11. If everything works as expected, set the deployment life cycle and repeat steps 7-9.
12. Create the manufacturing package. For details, see Create manufacturing package. For API key distribution, see API key to access EdgeLock 2GO server.

For processor-specific information, see Processor-specific workflows.

7.6.4 API key to access EdgeLock 2GO server

- For development, the API key must be specified in the **Trust Provisioning Mode** dialog.
- For **EdgeLock 2GO per product type**, the API key is not used in the manufacturing
- For **EdgeLock 2GO with device ID**, the API key is also needed for manufacturing:
  - For the manufacturing package, the API key can be specified optionally; if it is specified, it is recommended to encrypt the package with a password to protect the access.
  - If the API key is not specified in the manufacturing package, it can be specified in the manufacturing dialog or via the environment variable SEC_EL2GO_API_KEY.
  - For EdgeLock 2GO trust provisioning, the following access needs to be enabled for the manufacturing API key on the EdgeLock 2GO server:
    - View - View device groups
    - General - Register devices
    - Remote trust provisioning - View secure objects and their intermediate CAs
  - For EdgeLock 2GO WPC provisioning, the following access needs to be enabled for the manufacturing API key on the EdgeLock 2GO server:
    - View - View Qi related data
    - Manage - Assign and unassign Qi templates to Device Groups and query jobs for batch processing

7.6.5 EdgeLock 2GO configuration via web portal

EdgeLock 2GO web portal URL: EdgeLock 2GO web portal

The following Application Note provides detailed information: EdgeLock 2GO Provisioning via Secure Provisioning Tool (SEC) for MCUs (document AN14624).

How to create device group

- Devices > New Device Group
- Assign any name
- Hardware family type: MPU and MCU
- Hardware option: product name
- Hardware type and product name: select a processor by name

How to create RKTH secure object

- Secure Objects > New Secure Object
- Select type: OEM FW Authentication Key Hash
- Assign any name
- Provide a binary file: <upload the file>
- Usage policies: add a custom policy. Select the **Open** or **Closed/Locked** policy. **Open** is used for development and testing, **Closed** and **Closed/Locked** are used for production. The policy is to be aligned with the life cycle selected in the tool. For details, see processor-specific recommendations.

How to create CUST-MK-SK secure object

- Secure Objects > New Secure Object
- Select type: OEM FW Decryption Key
- Assign any name
- Provide the key material (.asc): provide a signed and encrypted key. It is generated by a SEC Tool via GnuPG.
- Provide the public key to use it for verifying the signature of the encrypted key. (.asc): your public key
- Usage policies: same as RKTH; the policy must be the same for all secure objects used in one device group.

Other secure object

See processor-specific information in Processor-specific workflows.

7.6.6 Create database with secure objects for EdgeLock 2GO per product type

There are two types of the databases:

- **static database**: does not contain any processor specific secure objects
- **dynamic database**: contains processor-specific certificates

To create a database of secure objects for “offline” provisioning (EdgeLock 2GO per product type), follow these steps:

1. Open **main menu > Targets > Trust Provisioning Mode…** and ensure **EdgeLock 2GO per product type** is selected.
2. In **Per Product Database** panel specify:
   - number of devices per database
   - number of databases to be generated

   These parameters will be used for dynamic database only. For a static database, the parameters will be ignored.

3. Click the button to start generation. Mind the generation of dynamic databases may take significant time (typically several hours) and there is a displayed progress bar with the estimated time. Do not close the window until the database is generated.
4. Click the **Download** button to download the databases from the server.

For the write operation, the database name is hard-coded to `<workspace>/trust_provisioning/el2go_product_batch.db`. For manufacturing operation, the database name and location can be selected in the manufacturing dialog.

Databases are not included in the manufacturing package and are expected to be delivered separately.

7.7 Merge Images Tool workflow

This section describes merging two separate images into one single image. The reason for merging might be signing both images together during the build process. This section is written for MCUXpresso IDE with the Trust zone hello world SDK example, which contains two separate projects `_hello_world_s` and `_hello_worlds_ns` linked together. It is necessary to remove the link between the projects (MCUXpresso IDE) to generate two separated images instead of one image created during the build process by merging both projects together. As an example processor the MIMXRT595 is used. For other processors the workflow is the same, except of the start addresses.

7.7.1 Building the TrustZone example projects for merge

1. Remove linking between both projects by selecting the non-secured project `_hello_world_ns` go to **Project > Properties > Project References** and unchecking the `_hello_world_s` from the references.
2. Build both projects without any modifications
3. This example uses .s19 files so the final .axf files need to be converted. To do so, go to the Debug folder for both projects, right-click the generated `_hello_world_*.axf` file, and choose **Binary Utilities > Create S-Record**.

7.7.2 Merging two images into one

1. Open the Merge Tool by selecting **main menu > Tools > Merge Tool**.
2. Select `_hello_world_s.s19`. The target address will be automatically set to `0x18001000` (secured memory area)
3. For MIMXRT595 processors, only the non-secured address can be written by the user. The given address of the first image is in the secured memory map. Change the target address of the first image to `0x08001000` (non-secured memory area)
4. Select `_hello_world_ns.s19`. The target address will be automatically set to `0x08100000` (non-secured memory area)
5. Check the **Apply the merged images as the Source executable image on Build image view** checkbox
6. Leave the rest as it is and press the OK button
7. Check that the image is automatically applied as a Source executable image and its start address is updated as well

7.7.3 Signing merged image

To sign and build the merged image, see the Booting signed image using shadow registers section, but use the merged image as a Source executable image instead.

7.8 Custom USB VID and PID devices workflow

If you have a device with custom VID&PID, it is possible to select the custom VID&PID in the **main menu > Target > Connection**. Selected VID&PID is used for the *whole* write operation. Write script does not support a change of VID&PID during its execution. It is recommended to change VID&PID prior to the write operation (for example, using PFR configuration or OTP configuration in **Advanced mode**), reset the processor and then start the write operation with custom VID&PID.

For manufacturing operation, custom VID&PID must be selected in the Connection configuration during the creation of the manufacturing package. With such configuration, the manufacturing operation accepts both default and custom VID&PID.

8 Command-line operations

SEC also offers a command-line interface, enabling integration in automated environments or customization of image building/burning procedure. Operation requires a verb (command) identifying the top-level operation (building, flashing, provisioning, generating keys or detecting the list of USB devices) and additional operation-specific options.

To display the available commands, arguments, and examples, run the following command from the command prompt:

```
c:/nxp/SEC_Provi_25.09/bin/securep.exe -h
```

To display the available arguments for a specific command, run the following command from the command prompt:

```
c:/nxp/SEC_Provi_25.09/bin/securep.exe <command> -h
```

Note: The location of the SEC Tool application is subject to the installation folder.

All the supported commands and arguments used in the command line as described in chapters below can also be specified in a separate configuration JSON file. This JSON file is then passed as a command-line argument, see Example how to use args-file arguments.

Table args-file argument

Argument	Description
`--args-file ARGS_FILE`	Path to the JSON file with CLI arguments allowing to specify all arguments in one file. The path is absolute or relative to the current working directory. The file format is specified by `schema/cli_args_file_schema_v?.json.`

8.1 Build

With the **build** command, you can perform actions that you can otherwise perform in the **Build image** view of SEC.

The following arguments are available to the **build** command:

Table Build-specific arguments

Argument	Description
`-h, --help`	Show this help message and exit.

Argument	Description
`--additional-images-cfg`	Path to JSON with configuration of additional images for the build. The file format is specified by the schema `schema/additional_images_schema_v?.json`.
`--bee-user-keys-config BEE_USER_KEYS_CONFIG.json`	JSON file with the BEE configuration. See the schema/bee_image_encryption_schema_v2.json in the installation folder. The parameter is applicable for encrypted XIP (BEE user keys) boot type only.
`--boot-type VALUE`	Secure boot type. Run securep -help to see all supported boot types.
`--cfpa-cfg CFPA_CFG.json`	Path to JSON file with USER CFPA configuration. It is recommended to export the file from the PFR Configuration dialog.
`--cmpa-cfg CMPA_CFG.json`	Path to JSON file with USER CMPA configuration. It is recommended to export the file from the PFR Configuration dialog.
`--csf-cert CSF_CERT`	Path to the public CSF key file that is used for signing the image. If not specified then it is derived from the img-cert pathname according to the HAB4 PKI Tree naming convention.
`--dcd DCD.bin`	Path to the Device Configuration Data binary file.
`--dekkey DEKKEY`	32/48/64 HEX characters: data encryption key used for AHAB encryption. The argument is applicable for processors with the AHAB security system.
`--device {name of the selected processor}`	Target processor. The list of supported processors is displayed using command `securep -h`
`--dual-image-boot-cfg DUAL_IMAGE_BOOT_CFG.json`	JSON file with dual image boot configuration; file format is specified by schema/dual_image_boot_schema_v?.json. The argument is applicable for processors that supports dual image boot
`--ele-firmware ELE_FIRMWARE`	Path to the EdgeLock Enclave (ELE) firmware file. The argument is applicable for encrypted boot types for processors with the AHAB security system.
`--firmware-version FIRMWARE_VERSION`	Version of the application image firmware.
`--iee-config IEE_CONFIG.json`	JSON file with the IEE configuration. See the schema/iee_image_encryption_schema_v?.json in the installation folder. The parameter is applicable for IEE encrypted boot type only.
`--image-version IMAGE_VERSION`	The version of the bootable image can be either in 4-bytes format, for example, 0xFFFE0001 (the lower 2 bytes are the real version number, and the upper 2 bytes are the invert value of lower 2 bytes) or just the real version number (2 bytes). The argument is only applicable for processors that support the image version on the build tab.
`--img-cert IMG_CERT`	Path to the public IMG key file that is used for signing the image. It is recommended to use the command with a workspace with already initialized key management. If the keys are not specified in the workspace settings file, they are imported.

Argument	Description
`--iped-cfg IPED_CFG.json`	JSON file with PRINCE configuration; file format is specified by schema/prince_config_schema_v?.json
`--keyblob-keyid KEYBLOB_KEYID`	32-bit value: Keyblob encryption key identifier. The argument is applicable for processors with the AHAB security system.
`--keysource \{OTP, KeyStore\}`	Key source for RT5xx/6xx secured images
`--life-cycle LIFE-CYCLE-ID`	Requested life-cycle state of the processor. The list of supported IDs is displayed using command `securep -h`.
`--otfad-config OTFAD_CONFIG.json`	JSON file with the OTFAD configuration. See the schema/ otfad_image_encryption_schema_v?.json in the installation folder. The parameter is applicable for OTFAD encrypted boot type only.
`--otp-cfg OTP_CFG.json`	Path to JSON file with USER OTP configuration. It is recommended to export the file from the OTP Configuration dialog.
`--prince-cfg PRINCE_CFG.json`	JSON file with PRINCE configuration. See bin/schema/ prince_config_schema_v<version>.json in the SEC installation folder.
`--romcfg-cfg ROMCFG_CFG.json`	Path to JSON file with USER ROMCFG configuration. It is recommended to export the file from the IFR Configuration dialog.
`--save-settings`	Save workspace settings.
`--sbkek/--cust\_mk\_sk/--sb3kdk SBKEK`	64 HEX characters: Key used as key encryption key to handle SB2 file; Needed only for Secure Binary images; If not specified, it is taken from workspace
`--script-only`	Generate build script only, do not launch it
`--secret-key-type \{AES-128, AES-192, AES-256\}`	The HAB encryption algorithm, default is processor-specific.
`--source-image SOURCE_IMAGE`	Source image path for building the boot image.
`--start-address START_ADDRESS`	Start address of the executable image data within the source image. Applicable and required only for binary source images.
`--target-image TARGET_IMAGE`	Target image path for building the boot image.
`--trust-provi {disabled, device_hsm, el2go_indirect, wpc_no_provi, wpc_device_hsm}`	Trust provisioning type. `el2go_indirect` stands for Edge Lock 2GO proxy flow.
`--trust-zone TRUST_ZONE`	Either `disabled` (for the TrustZone disabled image) or `default` (for the TrustZone enabled image with default data from the processor) or path to the custom TrustZone configuration JSON or YAML file (for the TrustZone enabled image with custom configuration)
`--userkey USERKEY`	Key applicable for RT5xx/6xx secured images: for OTP key-source it represents the master key; for key-store it represents the key used for the signature
`-v, --verbose`	Increase output verbosity
`-w WORKSPACE, --workspace WORKSPACE`	Workspace location, path to the workspace directory. **Note:** Any settings from the workspace are loaded automatically.

Argument	Description
	All command-line parameters can be used to override loaded settings.
`--xip-enc-otpmk-config XIP_ENC_OTPMK_CONFIG.json`	JSON file with the XIP Encryption with OTPMK configuration. See the schema/xip_enc_otpmk_schema_v?.json in the installation folder. The argument is applicable for XIP encrypted (BEE OTPMK) and XIP encrypted (OTFAD OTPMK) boot types only.
`--xmcd-cfg XMCD_CFG`	Path to YAML or binary file with the XMCD configuration (simplified or full); for file format, see the SPSDK command `nxpimage bootable-image xmcd get-templates`.

Table Boot-device arguments (mutually exclusive)

Argument	Description
`--boot-device VALUE`	Predefined boot memory. Run `securep --help` to see all supported boot memories.
`--boot-device-file BOOT_DEVICE_FILE`	File with boot memory configuration
`--boot-device-type TYPE`	Boot memory type, one of the following types: `flex-spi-nor`, `flex_spi_nand`, `ifr_memory`, `onchip_memory`, `onchip_ram`, `sdhc_emmc`, `sdhc_emmc_mpu`, `sdhc_sd_card`, `sdhc_sd_card_mpu`, `semc_nand`, `serial_downloader`, `xspi-nor`. Default-predefined boot memory of this type is applied.

8.2 Write

With the **write** command, you can perform actions that you can otherwise perform in the **Write image** view of SEC.

The following arguments are available to the **write** command:

Table Write-specific arguments

Argument	Description
`-h, --help`	Show this help message and exit
`--source-image SOURCE_IMAGE`	Source image path to be uploaded to the target
`--write-params-cfg WRITE_PARAMS_CFG.json`	JSON file with parameters needed in write and fuses to be burnt by write script (or shadow registers). See the `schema/write_parameters_schema_v?.json` in the installation folder.
`--life-cycle LIFE-CYCLE-ID`	Requested life-cycle state of the processor. The list of supported IDs is displayed using command `securep -h`.
`--trust-provi {disabled, device_hsm, el2go_indirect, wpc_no_provi, wpc_device_hsm}`	Trust provisioning type.
`-v, --verbose`	Increase output verbosity
`--device {name of the selected processor}`	Target processor. The list of supported processors is displayed using command `securep -h`
`--boot-type VALUE`	Secure boot type. Run `securep --help` to see all supported boot types.

Argument	Description
`--script-only`	Generate script only, do not launch
`-w WORKSPACE, --workspace WORKSPACE`	Workspace location. **Note:** Any settings from the workspace are loaded automatically. All command-line parameters can be used to override loaded settings.
`--debug-probe PROBE`	Select a debug probe. Use `-debug-probe auto` to select any debug probe. Use `-debug-probe invalid` to list all connected debug probes.

For boot-device arguments, see **Table Boot-device arguments (mutually exclusive)**

Table Connection arguments (mutually exclusive)

Argument	Description
`--usb VID PID`	Connect to target over USB HID device denoted by vid/pid. USB HID connection is default. vid/pid can be specified in decimal form (for example, `123`) or hexadecimal form (for example, `0xbeef`).
`--uart UART`	Connect to target over UART. Specify COM port (see `--baud-rate argument`). Example: `--uart COM3`
`--i2c address speed_kHz`	Connect to target over I2C via USB bridge. Specify I2C device address and clock in kHz. SIO device is autoselected if the `--sio-device` argument is not specified. Example: -i2c 0x10 400
`--spi speed_kHz polarity phase`	Connect to target over SPI via USB bridge. Specify SPI clock in kHz, polarity (SPI CPOL option) and phase (SPI CPHA option). SIO device is autoselected if the -sio-device argument is not specified. Example: `--sp i 1000 1 1`
`--baud-rate BAUD_RATE`	Connect to target over UART with a specified baud rate. -uart argument has to be specified too. Example: `--baud-rate 9600`
`--sio-device SIO_DEVICE`	Connect to target over USB-SIO (I2C or SPI) via a specified SIO device. `--i2c` or `--spi` argument has to be specified too. Example: `--sio-device HID\VID_1FC9&PID_0090&MI_03\7&96E050B&0&0000`

Note: For connection to the board, a USB or Serial port has to be specified. If nothing is specified, USB autodetection is applied.

8.3 Generate keys

With the **Generate** command, you can perform actions that you can otherwise perform in the **Generate Keys** view. Compared to GUI, command-line functionality is restricted.

The following arguments are available to the **generate** command:

Table generate-specific arguments

Argument	Description
`-h, --help`	Show this help message and exit
`--keys-cfg KEYS_CFG.json`	File with the keys configuration

Argument	Description
`--device {name of the selected processor}`	Target processor. The list of supported processors is displayed using command `securep -h`
`--boot-type VALUE`	Secure boot type. Run `securep --help` to see all supported boot types.
`--script-only`	Generate script only, do not launch
`-w WORKSPACE, --workspace WORKSPACE`	Workspace location. **Note:** Any settings from the workspace are loaded automatically. All command-line parameters can be used to override loaded settings.

For boot-device arguments, see **Table Boot-device arguments (mutually exclusive)**

8.4 Manufacture

Manufacture command allows running the selected script several times in parallel, each time for a different connection. The following arguments are available to the **manufacture** command:

Table Manufacture-specific arguments

Argument	Description
`-h, --help`	Show this help message and exit
`--script_path SCRIPT_PATH`	Path to the script to be executed
`--script_params SCRIPT_PARAMS`	Parameters of the script. For more information, see Manufacturing Tool.
`--connections CONNECTIONS {CONNECTIONS ...}`	List of all connections devices to be used in manufacturing, in format `-p <port\>,<baud\>` or `-u <usb-path\>` or `-l usb,<usb-path\>,spi\[,<port\>,<pin\>,<speed\_kHz\>,<polarity\>,<phase\>\]` or `-l usb,<usb-path\>,i2c\[,<address\>,<speed\_kHz\>\]`. To find all available USB/USB-SIO connections, automatically use `-u <autodetect-all-USBs\>` or `-l usb,<autodetect-all-USBSIOs\>,spi\[,<port\>,<pin\>,<speed\_kHz\>,<polarity\>,<phase\>\]` or `-l usb,<autodetect-all-USBSIOs\>,i2c\[,<address\>,<speed\_kHz\>\]`. The options for autodetection cannot be combined with the other options. Parameters and default values for SIO operation are described in the SPSDK documentation.

8.5 Devices info

With the **devices-info** command, you can get information about supported processors and their supported boot devices.

The **devices-info** command have two modes:

Table Devices-info command modes

Mode	Description
`processors`	Information about supported processors and their supported boot devices. This mode is default.
`boot-devices`	Information about supported boot devices memory.

The following arguments are available for the **devices-info** command:

Table Devices-info specific arguments

Argument	Description
`-h, --help`	Show this help message and exit.
`--device DEVICE`	Device name substring, which is searched in the processor. For boot devices, the whole name of processor is needed to show supported memories. If not provided, unfiltered data are returned.
`--format \{json,txt\}`	Format of the output. Default is json for file output, txt for STDOUT. For processors json, see schema/devices_info_schema_v?.json. For boot devices json, see schema/boot_devices_info_schema_v?.json
`--output OUTPUT`	Path to a file where to store the output. If not provided, the output is printed to STDOUT.

8.6 Test Connection

The **test-connection** command performs a processor-specific connection test. The device must be specified either in the workspace or via the command-line option. If no connection is specified in the workspace or as an option, the default connection is used. The connection test is successful if a matching device is connected and is in ISP mode.

Table Supported arguments

Argument	Description
`-h, --help`	Show this help message and exit.
`-v, --verbose`	Increase output verbosity
`-w WORKSPACE, --workspace WORKSPACE`	Workspace location.
`--device {name of the selected processor}`	Target processor. The list of supported processors is displayed using command `securep -h`

Table Connection arguments (mutually exclusive)

Argument	Description
`--usb VID PID`	Connect to target over USB HID device denoted by vid/pid. USB HID connection is default. vid/pid can be specified in decimal form (for example, `123`) or hexadecimal form (for example, `0xbeef`).
`--uart UART`	Connect to target over UART. Specify COM port (see `--baud-rate argument`). Example: `--uart COM3`
`--i2c address speed_kHz`	Connect to target over I2C via USB bridge. Specify I2C device address and clock in kHz. SIO device is autoselected if the `--sio-device` argument is not specified. Example: -i2c 0x10 400
`--spi speed_kHz polarity phase`	Connect to target over SPI via USB bridge. Specify SPI clock in kHz, polarity (SPI CPOL option) and phase (SPI CPHA option). SIO device is autoselected if the -sio-device argument is not specified. Example: `--sp i 1000 1 1`

Argument	Description
`--baud-rate BAUD_RATE`	Connect to target over UART with a specified baud rate. -uart argument has to be specified too. Example: `--baud-rate 9600`
`--sio-device SIO_DEVICE`	Connect to target over USB-SIO (I2C or SPI) via a specified SIO device. `--i2c` or `--spi` argument has to be specified too. Example: `--sio-device HID\VID_1FC9&PID_0090&MI_03\7&96E050B&0&0000`

8.7 Detect USB devices

The **detect** command identifies any processor connected via USB. For detection to succeed, the processor must be in ISP mode.

8.8 Start flashloader

Start the flashloader on the RT1xxx processor. The following arguments are available for the **start-flashloader** command:

Table start-flashloader command specific arguments

Argument	Description
`-h, --help`	Show this help message and exit.
`-v, --verbose`	Increase output verbosity
`-w WORKSPACE, --workspace WORKSPACE`	Workspace location.
`--script-only`	Generate script only, do not launch.

Table Connection arguments (mutually exclusive)

Argument	Description
`--usb VID PID`	Connect to target over USB HID device denoted by vid/pid. USB HID connection is default. vid/pid can be specified in decimal form (for example, `123`) or hexadecimal form (for example, `0xbeef`).
`--uart UART`	Connect to target over UART. Specify COM port (see `--baud-rate argument`). Example: `--uart COM3`
`--i2c address speed_kHz`	Connect to target over I2C via USB bridge. Specify I2C device address and clock in kHz. SIO device is autoselected if the `--sio-device` argument is not specified. Example: -i2c 0x10 400
`--spi speed_kHz polarity phase`	Connect to target over SPI via USB bridge. Specify SPI clock in kHz, polarity (SPI CPOL option) and phase (SPI CPHA option). SIO device is autoselected if the -sio-device argument is not specified. Example: `--sp i 1000 1 1`
`--baud-rate BAUD_RATE`	Connect to target over UART with a specified baud rate. -uart argument has to be specified too. Example: `--baud-rate 9600`
`--sio-device SIO_DEVICE`	Connect to target over USB-SIO (I2C or SPI) via a specified SIO device. `--i2c` or `--spi` argument has to be specified

Argument	Description
	too. Example: `--sio-device HID\VID_1FC9&PID_0090&MI_03\7&96E050B&0&0000`

8.9 Apply settings

Command **apply-settings** supports two main use cases

- Modifying an existing workspace
- Creating a new workspace

For modifying an existing workspace, specify the path to the workspace and the options to change. General options can be used for this purpose.

For creating a new workspace, at minimum, the boot-device and device options must be specified.

Table apply-settings command arguments

Argument	Description
`-h, --help`	Show this help message and exit.
`-v, --verbose`	Increase output verbosity
`-w WORKSPACE, --workspace WORKSPACE`	Workspace location.
`--life-cycle LIFE-CYCLE-ID`	Requested life-cycle state of the processor. The list of supported IDs is displayed using command `securep -h`.
`--trust-provi {disabled, device_hsm, el2go_indirect, wpc_no_provi, wpc_device_hsm}`	Trust provisioning type.
`--device {name of the selected processor}`	Target processor. The list of supported processors is displayed using command `securep -h`
`--boot-type VALUE`	Secure boot type. Run `securep --help` to see all supported boot types.
`--boot-device VALUE`	Predefined boot memory. Run `securep --help` to see all supported boot memories.
`--boot-device-file BOOT_DEVICE_FILE`	File with boot memory configuration
`--boot-device-type {flex-spi-nor, flex_spi_nand, onchip_memory, sdhc_sd_card, semc_nand}`	Boot memory type. Default-predefined boot memory of this type is set.

Table Connection arguments (mutually exclusive)

Argument	Description
`--usb VID PID`	Connect to target over USB HID device denoted by vid/pid. USB HID connection is default. vid/pid can be specified in decimal form (for example, `123`) or hexadecimal form (for example, `0xbeef`).
`--uart UART`	Connect to target over UART. Specify COM port (see `--baud-rate argument`). Example: `--uart COM3`
`--i2c address speed_kHz`	Connect to target over I2C via USB bridge. Specify I2C device address and clock in kHz. SIO device is autoselected if the `--sio-device` argument is not specified. Example: -i2c 0x10 400

Argument	Description
`--spi speed_kHz polarity phase`	Connect to target over SPI via USB bridge. Specify SPI clock in kHz, polarity (SPI CPOL option) and phase (SPI CPHA option). SIO device is autoselected if the -sio-device argument is not specified. Example: `--sp i 1000 1 1`
`--baud-rate BAUD_RATE`	Connect to target over UART with a specified baud rate. -uart argument has to be specified too. Example: `--baud-rate 9600`
`--sio-device SIO_DEVICE`	Connect to target over USB-SIO (I2C or SPI) via a specified SIO device. `--i2c` or `--spi` argument has to be specified too. Example: `--sio-device HID\VID_1FC9&PID_0090&MI_03\7&96E050B&0&0000`

8.10 Environment variables in filepath-based arguments

SEC Tool accepts environment variables in all arguments specifying paths, for example:

```
securep.exe -w /workspaces/mcuxprovi --device MIMX9596 --boot-device-type
onchip_ram --boot-type unsigned build --additional-images
additional_images_cfg.json --ele-firmware
"${MCUX_SDK_16}\firmware\edgelock\mx95a0-ahab-container.img" --save-settings
```

8.11 Command-line examples

8.11.1 Example: How to build and write an image for configuration stored in the workspace folder

In this example, it is assumed that the GUI was already used to prepare complete configuration within a workspace (keys generated, build image configured, write image configured).

```
securep.exe -w /workspaces/mcuxprovi build
```

```
securep.exe -w /workspaces/mcuxprovi write
```

For detailed examples, use the following command:

```
securep.exe print-cli-examples
```

8.11.2 Example how to use args-file arguments

In the following examples, CLI arguments are converted to the JSON file.

Arguments in the command line for build:

```
securep.exe -w /workspaces/mcuxprovi --device MIMX9596 --boot-device-type
onchip_ram --boot-type unsigned build --additional-images
additional_images_cfg.json --ele-firmware mx95a0-ahab-container.img --save-
settings
```

Usage of args-file argument for above build arguments:

```
securep.exe --args-file args_file_build.json
```

args_file_build.json content (only the first additional image is listed):

```
{
  "cli_args": {
    "-w": "/workspaces/mcuxprovi",
    "--device": "MIMX9596",
    "--boot-device-type": "onchip_ram",
    "--boot-type": "unsigned",
    "build": [],
    "--additional-images": {
      "images": [
        {
          "entry_type": "oei_ddr",
          "container_set": "#1",
          "extra_settings": {
            "lpddr_imem_path": "${ENV_VAR_DDR}/lpddr5_imem_v202311.bin",
            "lpddr_imem_qb_path": "${ENV_VAR_DDR}/lpddr5_imem_qb_v202311.bin",
            "lpddr_dmem_path": "${ENV_VAR_DDR}/lpddr5_dmem_v202311.bin",
            "lpddr_dmem_qb_path": "${ENV_VAR_DDR}/lpddr5_dmem_qb_v202311.bin",
            "oei_ddr_path": "source_images/oei-m33-ddr.bin"
          }
        }
      ]
    },
    "--ele-firmware": "source_images/mx95a0-ahab-container.img",
    "--save-settings": []
  }
}
```

Arguments in command line for write:

```
securep.exe -w /workspaces/mcuxprovi --device MIMX9596 --boot-device-type
onchip_ram --boot-type unsigned write --source-image bootable_images/flash.bin
```

Usage of args-file argument for above write arguments:

```
securep.exe -w /workspaces/mcuxprovi --args-file args_file_write.json
```

args_file_write.json content:

```
{
  "cli_args": {
    "--device": "MIMX9596",
    "--boot-device-type": "onchip_ram",
    "--boot-type": "unsigned",
    "write": [],
    "--source-image": "bootable_images/flash.bin"
  }
}
```

Arguments passed directly in the command line are combined together with arguments in the args-file. Arguments cannot be specified in both places.

8.12 Command-line tools

SEC uses the following command-line tools to generate keys and build/write the image:

openssl : Key generation

spsdk : Secure Provisioning SDK, for more information, see main menu > Help > SPSDK Online Documentation. The following tools are available as part of SPSDK:

- **blhost** : Replacement for the legacy blhost tool
- **dk6prog** : Tool for reading and programming flash memory of DK6 target devices.
- **el2go-host** : Managing the EdgeLock 2GO provisioning operations
- **lpcprog** : Utility for communication with the bootloader on LPC8xx target.
- **nxpcrypto** : Operations with keys and certificates
- **nxpdebugmbox** : Debug mailbox and debug credential file generator tool
- **nxpdevhsm** : The application is designed to create an SB3 provisioning file for initial provisioning of the device by the OEM.
- **nxpdevscan** : Utility that detects NXP devices connected to the host PC over USB, UART, I2C, and SPI connections
- **nxpdice** : Application designed to cover DICE-related operations.
- **nxpele** : Utility for communication with the EdgeLock Enclave on target.
- **nxpfuses** : NXP Fuse Tool.
- **nxpimage** : Builds bootable image and SB files
- **nxpmemcfg** : Collection of utilities for memory configuration operations.
- **nxpshe** : NXP tool for working with SHE (Secure Hardware Extension).
- **nxpuuu** : The application for image deployment for i.MX MPUs. It is based on libUUU (universal update utility).
- **nxpwpc** : Utility covering WPC operations.
- **pfr** : Generating protected flash region files (cmpa/cfpa) and IFR.
- **sdphost** : Replacement for the legacy sdphost utility
- **sdpshost** : Utility for communication with ROM on i.MX targets using SDPS protocol (i.MX8/9).
- **shadowregs** : Shadow registers control tool.

imgtool : MCUboot’s image signing and key management

9 Troubleshooting

This chapter contains known problems and recommended solutions. For last-minute issues, refer to *Secure Provisioning Tool Release Notes* available as part of the SEC Tool and on the web document MCUXSPTRN.

9.1 General

- The application must be installed into the location where the user has the write access.
- By default, the Secure Provisioning Tool does not configure all possible security features that are available in the processor. Only the ones required by the selected boot type are configured. Configure the rest in OTP/PFR/IFR Configuration.
- If the tool is started with the option `-v` (verbose mode), it provides additional details (logs) that can be used to analyze problems.
  How to do in on Windows:
  – Click Windows Start button and run “Command Prompt”
  – Use command `cd c:\nxp\SEC_Provi_25.09\bin` to switch the current directory
  – Use command `securep.exe -v`

9.2 Windows

- On the Windows platform, make sure that the Windows FIND utility is found first on the PATH (GNU find utils could break the functionality).

9.3 Linux

- On the Linux platform, the USB and/or Serial device files have to be readable and writable by the current user. To solve this issue, see `resources/udev/99-secure-provisioning.rules` installed into `/etc/udev/rules.d/99-secure-provisioning.rules`. There can be a conflicting rule with higher priority on the machine. In this case, update the conflicting rule or make this rule file so they are applied in the expected order.
- Ubuntu 22 and USB2Serial CP210x: On Ubuntu 22, there is a conflicting package `brltty` that causes generic issues with the CP210x USB to serial converter. Uninstalling the `brltty` package fixes the issue. For more details, see Ubuntu bug report - BRLTTY issue
- The SEC Tool works well with the Xorg display server. Wayland, default in Ubuntu, causes various UI glitches, that is why the application shortcut contains the configuration to use the Xorg (x11) backend. If the SEC GUI is executed manually under the Wayland display server, make sure it is executed with proper environment variables, for example:
  `UBUNTU_MENUPROXY=0 GDK_BACKEND=x11 /opt/nxp/SEC_Provi_25.09/bin/securep`.
- The nxpuuu utility used for i.MX 9x might produce lots of escape characters in the Log view. To reduce the number of escape characters, use the following commands:

```
cd /opt/nxp/SEC_Provi_25.09/bin/tools/spsdk/
sudo sh -c "nxpuuu -udev >> /etc/udev/rules.d/70-nxpuuu.rules"
sudo udevadm control --reload
```

- P&E Micro Debug Probe is detected properly and offers PEMicro and pyOCD interfaces. PEMicro interface does not work - pyOCD interface has to be selected and used.

9.4 macOS

- GUI controls with invalid input are marked with a background red color. Sometimes fixing the value might not change the background color correctly and the focus must be changed to another field for the correct repaint.
- The PEmicro debug probe does not support macOS Aarch64 yet.
- The SPI and I2C communication is not reliable if more USB-to-SPI/I2C converters (MCU-Link PRO boards) are connected to the computer.
- If an env. variable (such as variable for EdgeLock 2GO) must be propagated to SEC executed via launcher, it has to be set using the `launchctl` command, for example:
  `launchctl setenv EL2GO_GPG_SIGN_KEY MY_KEY_ID`
  For a permanent env. variable, see the solution described at macOS environment sync solution.

9.4.1 macOS i.MX 9x nxpuuu

Due to a known limitation in libUUU and the underlying libusb library on macOS (limitation without a real solution see libusb macOS issue for more details), writing firmware using nxpuuu fails when not executed with sudo. Building of the firmware works without issues.

To make nxpuuu work on macOS in headless mode, configure the sudoers file using the `visudo` command. You can either grant the user rights to execute any command without providing a password or limit it to a single application, as described below. Replace `an_user` with the corresponding username.

Open the Terminal application and execute:

```
sudo visudo
```

At the end of the file, in the “User specification” section, add the following line. Ensure that the absolute path matches the installed SEC Tool. Make sure spaces in the command path are escaped using backslashes:

```
an_user ALL=(ALL) NOPASSWD: /Applications/SEC_Provi_25.09/Secure\ Provisioning\
 25.09.app/Contents/Frameworks/tools/spsdk/nxpuuu
```

Write script then uses `sudo` when it allows execution without a password prompt.

9.5 i.MX RT5xx/6xx

- Repetitive write to QSPI flash might fail in case the board is not reset and the reset pin is not configured in fuses. For more information, see RT5xx/6xx Device workflow/Booting images in section RT5xx/6xx device workflow in Processor-specific workflows.
- If shadow registers are used, it is necessary to HW reset the processor, before a new image or fuses configuration is applied because the shadow registers can be set into an unsecured processor only.

9.6 i.MX RT1024

- SD card boot device is not supported for the MIMXRT1024-EVK board due to limitation in the flashloader.

9.7 i.MX RT118x

- For i.MX RT118x the additional images may not work in combination with Serial downloader over UART.

9.8 MC56F818xx and MWCT2xD2

- During the build process for secured boot, the creation of secure binary files may sporadically fail. This issue affects both the device provisioning SBx file and the application SBx file. The build script may terminate with the following error `SpsdkNoDeviceFoundError`. This issue appears to be caused by a brief interruption in the ISP connection. Restart the **Build image** operation to proceed.

9.9 Debug probes

- Detecting PEmicro probes and detection of other probes after PEmicro was detected has these limitations:
  1. PEmicro must be connected before the first attempt for detection, if not connected it cannot be detected and the SEC Tool must be restarted for detection to succeed.
  2. When PEmicro is detected, it is impossible to detect any other pyOCD probe without restarting the SEC Tool. Also, if PEmicro is once detected it will always be listed as an available probe even if it is not connected anymore.
- Dependency on USB drivers for debug probe drivers is described in ReleaseNotes.txt in “System requirements”.
- The LinkServer binary is not found in the system PATH. On Linux systems, this can be resolved by creating a symbolic link to the LinkServer executable. For system-wide availability, place the link in /usr/bin/; for user-specific access, use ~/.local/bin/.

9.10 Debug authentication

If the Open debug port fails, there are usually no error messages (due to security reasons, the processor returns only a general error code). To make it working, check the following items:

- The processor should not be in ISP mode.
- The processor should be secured.
- The operation may fail if the debug port is already opened.
- After a failed connection test by the debug probe, it might be necessary to reset the processor to be able to open the debug port. On the KW45xx and K32E1xx series it is necessary.

9.11 Zephyr projects

Not every Zephyr example project can be used as it is. Additional workflow or changes in the project are required. For example, for MCX N devices the QSPI examples require a (MCUBoot) bootloader in the Internal Flash by default. The default start address is not usable for a bootable image creation without the bootloader. For details, see the Zephyr documentation for the given board. Either the bootloader in the internal flash has to be used or the start address must be changed. For the mentioned MCX N devices, the start address can be changed in the `build/zephyr/.config` file:

```
CONFIG_FLASH_BASE_ADDRESS=0x90000000
```

has to be changed to

```
CONFIG_FLASH_BASE_ADDRESS=0x90001000
```

10 References

10.1 User Guides

The following User Guides are referenced in this document:

10.1.1 Secure Provisioning Tool User Guide (Chinese version)

Secure Provisioning Tool User Guide (Document MCUXSPTUG_ZH)

10.1.2 Secure Provisioning Tool Quick Start Guide

https://docs.mcuxpresso.nxp.com/secure/latest/quick_start_guide.html

10.1.3 User Guide for MCUXpresso Config Tools

User Guide for MCUXpresso Config Tools (Desktop) (document GSMCUXCTUG)

10.2 Release Notes

The following Release Notes are referenced in this document:

10.2.1 Secure Provisioning Tool Release Notes

https://docs.mcuxpresso.nxp.com/secure/latest/release_notes.html

Secure Provisioning Tool Release Notes (document MCUXSPTRN)

10.2.2 i.MX Linux Release Notes

i.MX Linux Release Notes (document RN00210)

10.3 NXP SW resources

The following NXP SW resources are referenced in this document:

10.3.1 Secure Provisioning Tool home page

https://www.nxp.com/mcuxpresso/secure

10.3.2 MCU-Link Pro product page

http://www.nxp.com/pages/:MCU-LINK-PRO

10.3.3 LPC-Link2 product page

https://www.nxp.com/design/microcontrollers-developer-resources/lpc-link2:OM13054

10.3.4 MCUXpresso SDK Builder on NXP.com

http://mcuxpresso.nxp.com

10.3.5 EdgeLock 2GO web portal

https://edgelock2go.com

10.3.6 EdgeLock 2GO application note

AN14624 https://docs.nxp.com/bundle/AN14624/page/topics/introduction.html

10.3.7 ELE firmware

https://www.nxp.com/lgfiles/NMG/MAD/YOCTO/firmware-ele-imx-0.1.3-4e6938c.bin

10.3.8 Embedded Linux for i.MX Applications processors

https://www.nxp.com/design/design-center/software/embedded-software/i-mx-software/embedded-linux-for-i-mx-applications-processors:IMXLINUX

10.3.9 NXP i.MX main repository

https://github.com/nxp-imx

10.3.10 OEI DDR firmware

https://github.com/nxp-imx/imx-oei

10.3.11 CM33 System Manager repository

https://github.com/nxp-imx/imx-sm

10.3.12 U-Boot SPL, U-Boot repository

https://github.com/nxp-imx/uboot-imx

10.3.13 ARM Trusted Firmware

https://github.com/nxp-imx/imx-atf

10.3.14 TEE Binary repository

https://github.com/nxp-imx/imx-optee-os

10.4 Third party resources

The following Third party resources are referenced in this document:

10.4.1 PKCS#1 wikipedia article

https://en.wikipedia.org/wiki/PKCS_1

10.4.2 Security archive

RSA PKCS#1 security archive https://web.archive.org/web/20051029040347/http://rsasecurity.com/rsalabs/node.asp?id=2125

10.4.3 MCUboot documentation

https://docs.mcuboot.com/

10.4.4 pyOCD official website

http://pyocd.io/

10.4.5 Kleopatra for Windows (GPG4Win)

https://www.gpg4win.org/get-gpg4win.html

10.4.6 GnuPG downloads

https://gnupg.org/download/

10.4.7 Ubuntu bug report - BRLTTY issue

https://bugs.launchpad.net/ubuntu/+source/brltty/+bug/1970408

10.4.8 libusb macOS issue

https://github.com/libusb/libusb/issues/1014

10.4.9 macOS environment sync solution

https://github.com/ersiner/osx-env-sync

11 Revision history

Table Revision history

Document ID	Release date	Description
MCUXSPTUG_25.09 v.18	10 October 2025	Updated for MCUXpresso Secure Provisioning tool 25.09.
MCUXSPTUG_25.06 v.17	30 June 2025	Updated for MCUXpresso Secure Provisioning tool 25.06.
MCUXSPTUG_25.03 v.16	7 April 2025	Updated for MCUXpresso Secure Provisioning tool 25.03.
MCUXSPTUG_10.0 v.15	13 November 2024	Updated for MCUXpresso Secure Provisioning tool v10.
MCUXSPTUG_9.0 v.14	31 July 2024	Updated for MCUXpresso Secure Provisioning tool v9.
MCUXSPTUG_8.0 v.13	19 January 2024	Updated for MCUXpresso Secure Provisioning Tools v8. RT118x device workflow, RW61x device workflow, MCX Nx4x/N23x device workflow are added.
MCUXSPTUG_7.0 v.12	12 July 2023	Updated for MCUXpresso Secure Provisioning Tools v7: minor updates.
MCUXSPTUG_6.0 v.11	15 March 2023	Updated for MCUXpresso Secure Provisioning Tools v6: minor updates.
MCUXSPTUG_5.0 v.10	13 January 2023	Added: support for LPC55S36, section “LPC55S3x device workflow”; section Features is modified.
MCUXSPTUG_4.1 v.9	26 September 2022	Added: support for LPC55Sxx and LPC55xx families (LPC553x, LPC552x, LPC551x, and LPC550x), Debug Authentication, Flash Programmer, Boot device configuration are added.
MCUXSPTUG_4.1 v.8	24 June 2022	Updated for MCUXpresso Secure Provisioning Tools v4.1
MCUXSPTUG_4.0 v.7	09 May 2022	The term “Java Smart Card” is replaced with “Smart Card”.
MCUXSPTUG_4.0 v.6	22 April 2022	Updated for MCUXpresso Secure Provisioning Tool v4: Smart Card trust provisioning workflow is added. Information about bootable images as a source for build is added, screenshots are updated.
MCUXSPTUG_3.0 v.5	30 September 2021	The acronym “SPT” is replaced with “SEC” in all the flowcharts of the document.
MCUXSPTUG_3.0 v.4	28 July 2021	OTP/PFR Configuration rework, new workflows/flowcharts for OTFAD boot type, restructuring, new device information, minor changes

Document ID	Release date	Description
MCUXSPTUG_3.0 v.3	20 April 2021	Updated for MCUXpresso Secure Provisioning Tools v3
MCUXSPTUG_2.1 v.2	14 October 2020	Updated for MCUXpresso Secure Provisioning Tools v2.1
MCUXSPTUG_2.0 v.1	25 August 2020	Updated for MCUXpresso Secure Provisioning Tools v2
MCUXSPTUG_1 v.0	08 January 2020	Initial version

12 Note about the source code in the document

Example code shown in this document has the following copyright and BSD-3-Clause license:

Legal information

Intel, the Intel logo, Intel Core, OpenVINO, and the OpenVINO logo — are trademarks of Intel Corporation or its subsidiaries.

Kinetis — is a trademark of NXP B.V.

Contents