MCUXSPTUG

MCUXpresso Secure Provisioning Tool User Guide v.8

Rev. 13 — 19 January 2024

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 MCU platforms. 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

MCUXpresso Secure Provisioning Tool (SEC) is a GUI tool made to simplify the generation and provisioning of bootable executables on NXP MCU platforms. 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. New users should find it easier to prepare, flash, and secure images, while experienced users rediscover features from the existing toolset (sdphost, blhost, nxpimage, and so on) under a friendlier GUI. Experienced users can further customize secure provisioning flows by modifying scripts generated by the tool.

Figure 1. MCUXpresso Secure Provisioning Tool

2 Features

Features of the **MCUXpresso Secure Provisioning Tool** include:

- Support target connectivity via UART, USB-HID, SPI, and I2C serial download modes
- Support multiple user application image formats (bin, hex, srec, elf).
- Automated conversion of bare images to bootable images
- Downloading a bootable image in the target boot device
- Customization of booting from external flash either using GUI or predefined flash configuration blocks
- Generation of certificate trees for image signing and encryption, or use of user-supplied certificates
- Optional signature provider that allows customizing integration of HSM module for signing the image
- Support for development (unsigned and CRC) boot types
- Support for authenticated (signed) and encrypted boot types
- Key provisioning and fusing as dictated by boot type
- Advanced OTP/PFR/IFR configuration
- Trust provisioning and device HSM for production in factory
- Dual boot ping pong page
- Command-line interface for customized boot flows
- Simple Flash Programming tool
- Support for debug authentication
- SB editor tool for creation of custom SB files

- Manufacturing tool to support parallel provisioning operations in the factory
- Additional command-line utilities for low-level interaction with the device
- Integrated Development Environments supported: MCUXpresso IDE, Keil MDK 5, IAR Embedded Workbench
- Windows 64-bit, Linux 64-bit, and MacOS hosts

Note: *The complete list of supported devices and features can be found in the document "SEC-Tool-Features.xls" attached to this user guide.*

3 Terms and definitions

Table 1. Terms and definitions

Term	Definition
AES	Advanced Encryption Standard
AES-128	Rijndael cipher with block and key sizes of 128 bits
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
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
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.
CSF	Command Sequence File, a binary data structure interpreted by the HAB to guide authentication operations
CST	Code Signing Tool, an application running on a build host to generate a CSF and associated digital signatures
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
ECB	Electronic Code Book, a cipher mode with no feedback between the ciphertext blocks

Table 1. Terms and definitions*...continued*

Term	Definition
EKIB	Encrypted Key Info Block
EPRDB	Encrypted Protection Region Descriptor Block
FAC	Flash Access Controlled
FCB	Flash Configuration Block or Flash Control Block
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 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
ISK	Image Signing Key
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).
NDA	Non-disclosure Agreement
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 https://en.wikipedia.org/wiki/PKCS_1 and https://web.archive.org/web/20051029040347/http://rsasecurity.com/rsalabs/node.asp?id=2125.
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.

Table 1. Terms and definitions*...continued*

Term	Definition
PUF	Physical Unclonable Function
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. Accelerator (including hash acceleration) is found on some processors.
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	Encryption key is generated at the time of encryption. Only ever used once.
SHA-1	Hash algorithm that produces a 160-bit message digest
SNVS	Secure Non-Volatile Storage
SPSDK	Secure Provisioning SDK, an open source Python library and command-line tools for secure provisioning of NXP MCUs.
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.
UID	Unique Identifier, a unique value (such as a serial number) assigned to each processor during fabrication
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 list of supported operating systems, refer to Release Notes (https://www.nxp.com/design/software/development-software/mcuxpresso-software-and-tools-/mcuxpresso-secure-provisioning-tool:MCUXPRESSO-SECURE-PROVISIONING?tid=vanMCUXPRESSO-SECURE-PROVISIONING#documentation). Debug probe drivers are not part of the installers, see Release Notes.

4.1 Minimum system requirements

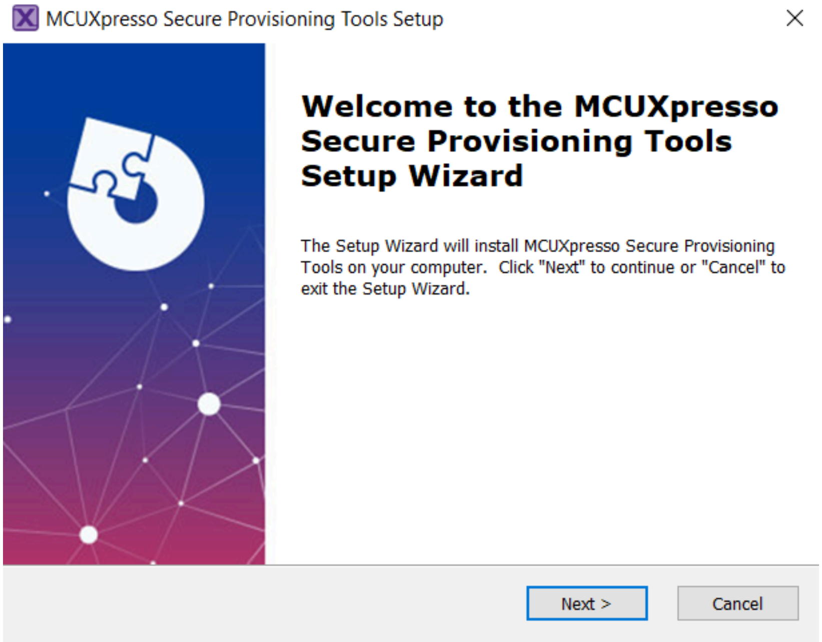
The tool runs on Microsoft(R) Windows(R), Ubuntu, and Mac OS X operating systems. The detailed system requirements are specified in ReleaseNotes.txt.

4.2 Windows

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

1. Visit the NXP website (https://www.nxp.com/mcuxpresso/secure) 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**.



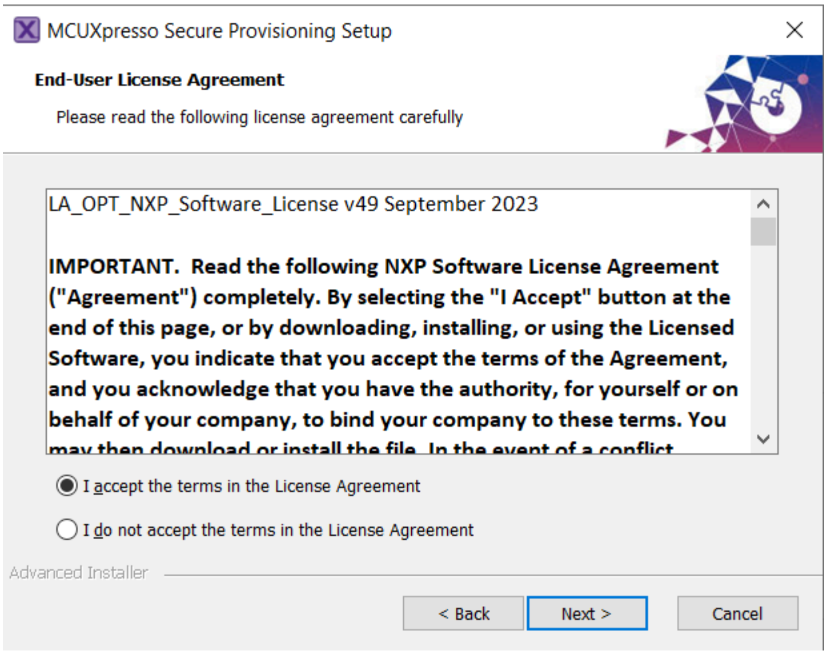
MCUXpresso Secure Provisioning Tools Setup

Welcome to the MCUXpresso Secure Provisioning Tools Setup Wizard

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

Figure 2. MCUXpresso 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**.



MCUXpresso Secure Provisioning Setup

End-User License Agreement

Please read the following license agreement carefully

LA_OPT_NXP_Software_License v49 September 2023

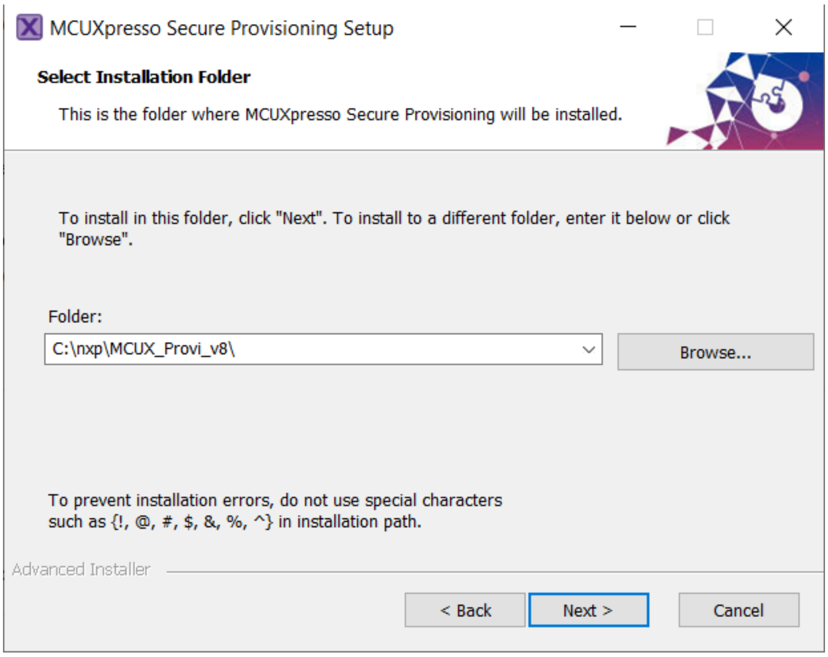
IMPORTANT. Read the following NXP Software License Agreement ("Agreement") completely. By selecting the "I Accept" button at the end of this page, or by downloading, installing, or using the Licensed Software, you indicate that you accept the terms of the Agreement, and you acknowledge that you have 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

● 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**.



MCUXpresso Secure Provisioning Setup

Select Installation Folder

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

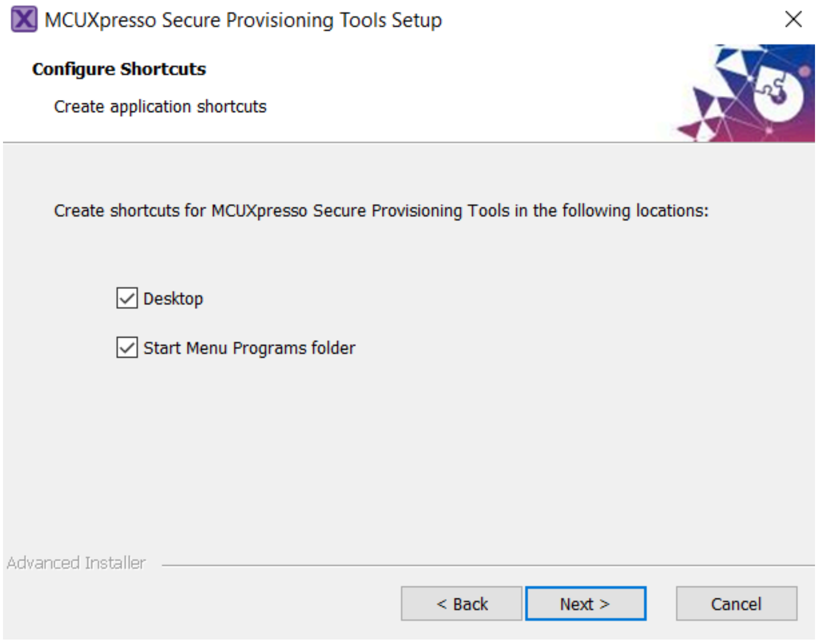
To install in this folder, click "Next". To install to a different folder, enter it below or click "Browse".

Folder:
C:\nxp\MCUX_Provi_v8\

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

Figure 4. Selecting installation folder

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



MCUXpresso Secure Provisioning Tools Setup

Configure Shortcuts

Create application shortcuts

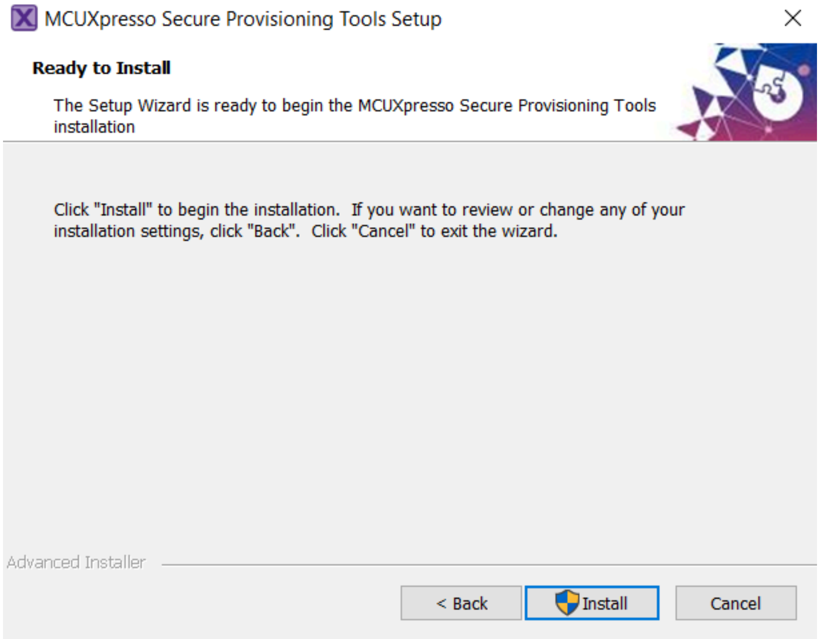
Create shortcuts for MCUXpresso Secure Provisioning Tools in the following locations:

☑ Desktop

☑ Start Menu Programs folder

Figure 5. Configuring shortcuts

7. On the **Ready to Install** page of the wizard, select **Install**.



MCUXpresso Secure Provisioning Tools Setup

Ready to Install

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

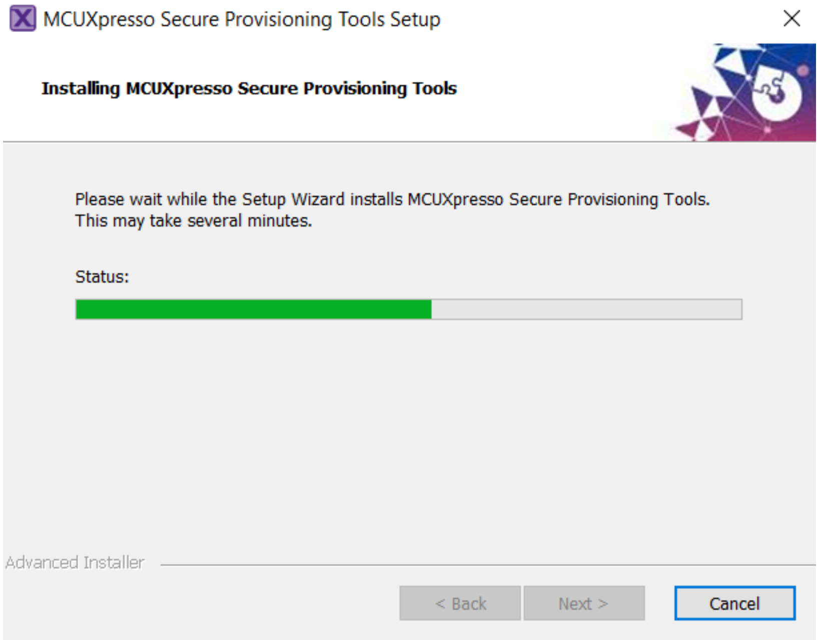
Click "Install" to begin the installation. If you want to review or change any of your installation settings, click "Back". Click "Cancel" to exit the wizard.

Figure 6. Ready to install

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.



MCUXpresso Secure Provisioning Tools Setup

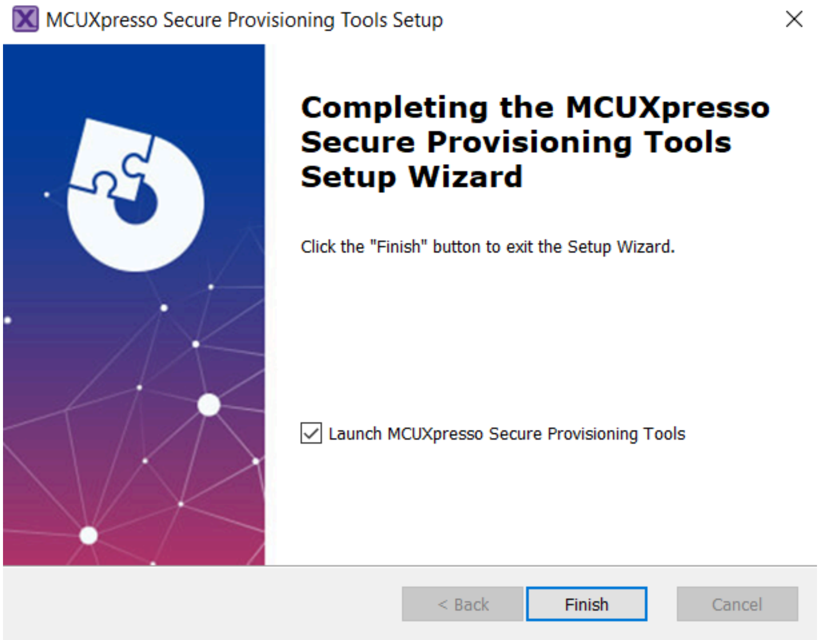
Installing MCUXpresso Secure Provisioning Tools

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

Status:

Figure 7. Installing MCUXpresso Secure Provisioning Tool

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



MCUXpresso Secure Provisioning Tools Setup

Completing the MCUXpresso Secure Provisioning Tools Setup Wizard

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

☑ Launch MCUXpresso Secure Provisioning Tools

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

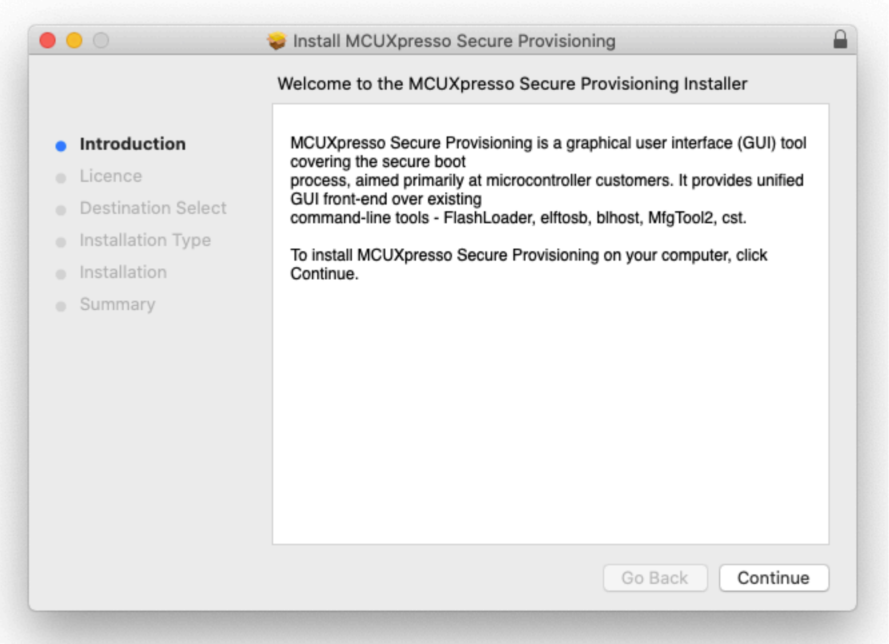
It is possible to install the SEC tool using the command line. In this case, use **Run the installer** with the following parameters:

```
MCUXpresso_Secure_Provisioning_v<version>.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 (https://www.nxp.com/mcuxpresso/secure) 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_<version>.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.*
3. On the **Introduction** page, click **Continue**.



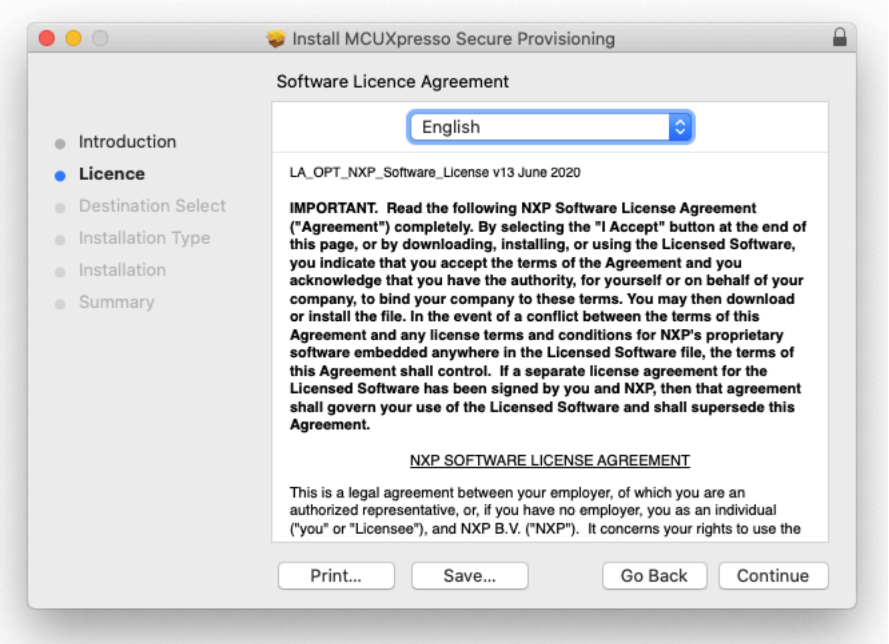
Welcome to the MCUXpresso Secure Provisioning Installer

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

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



Software Licence Agreement

LA_OPT_NXP_Software_License v13 June 2020

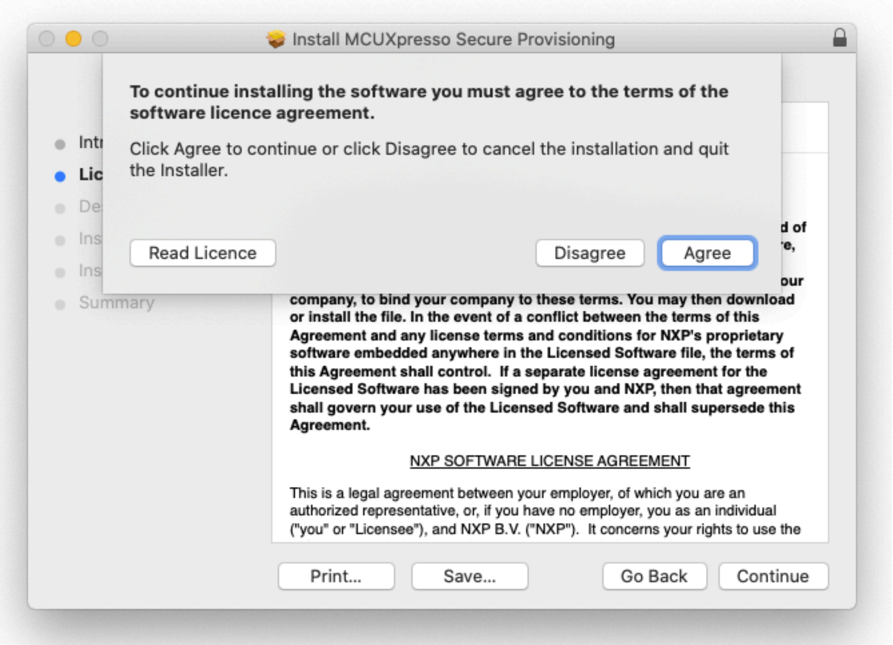
IMPORTANT. Read the following NXP Software License Agreement ("Agreement") completely. By selecting the "I Accept" button at the end of this page, or by downloading, installing, or using the Licensed Software, you indicate that you accept the terms of the Agreement and you acknowledge that you have the 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

5. 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.

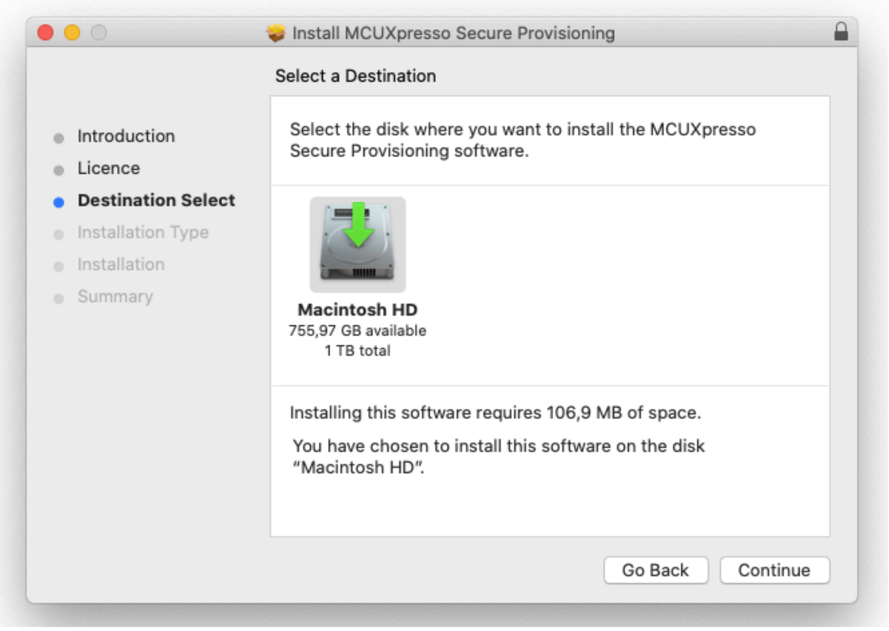
d of
e,
our
company, to bind your company to these terms. You may then download or install the file. In the event of a conflict between the terms of this Agreement and any license terms and conditions for NXP's proprietary software embedded anywhere in the Licensed Software file, the terms of this Agreement 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 11. Accepting software license agreement

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



Select a Destination

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

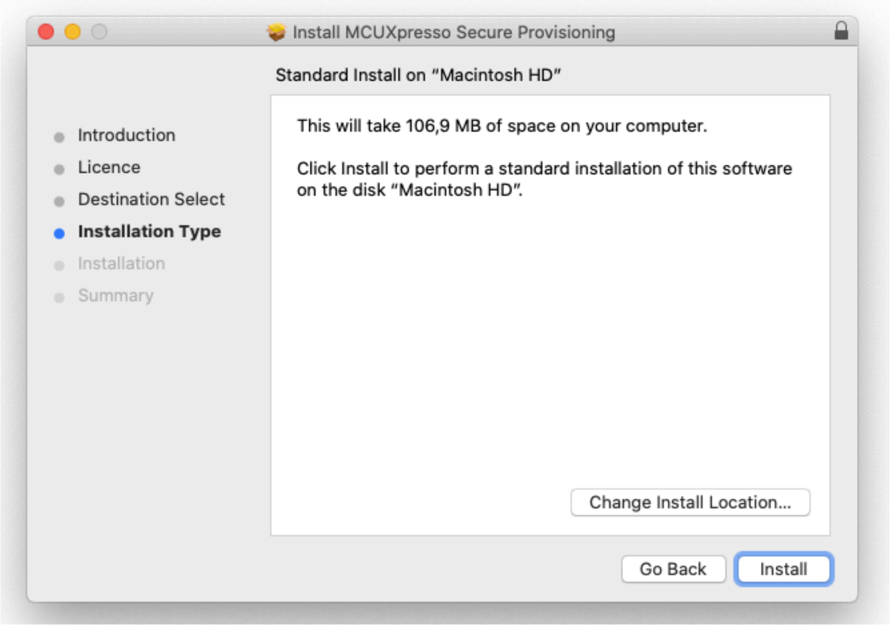
Macintosh HD
755,97 GB available
1 TB total

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

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



Install MCUXpresso Secure Provisioning

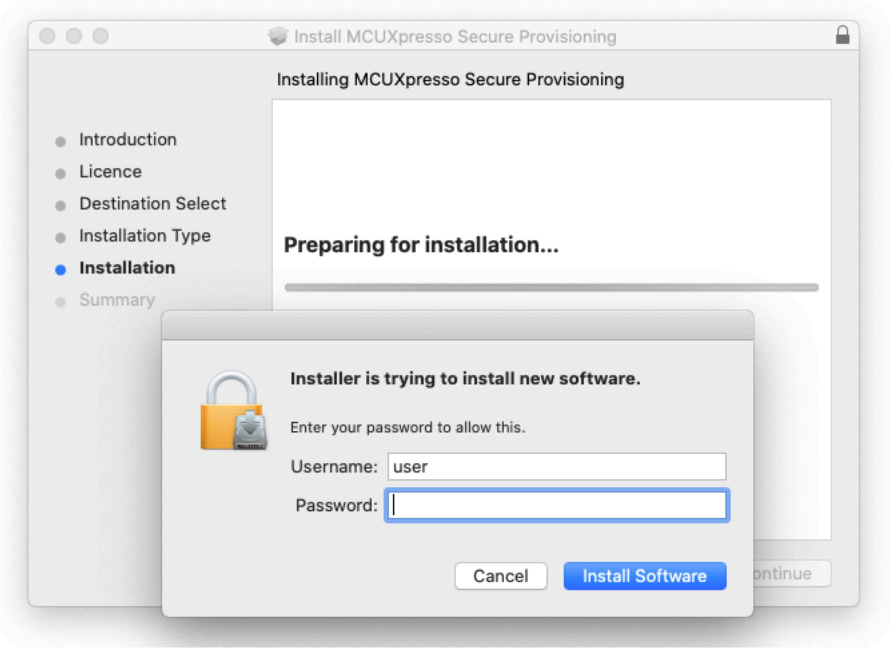
Standard Install on "Macintosh HD"

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

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



Install MCUXpresso Secure Provisioning

Installing MCUXpresso Secure Provisioning

Preparing for installation...

Installer is trying to install new software.

Enter your password to allow this.

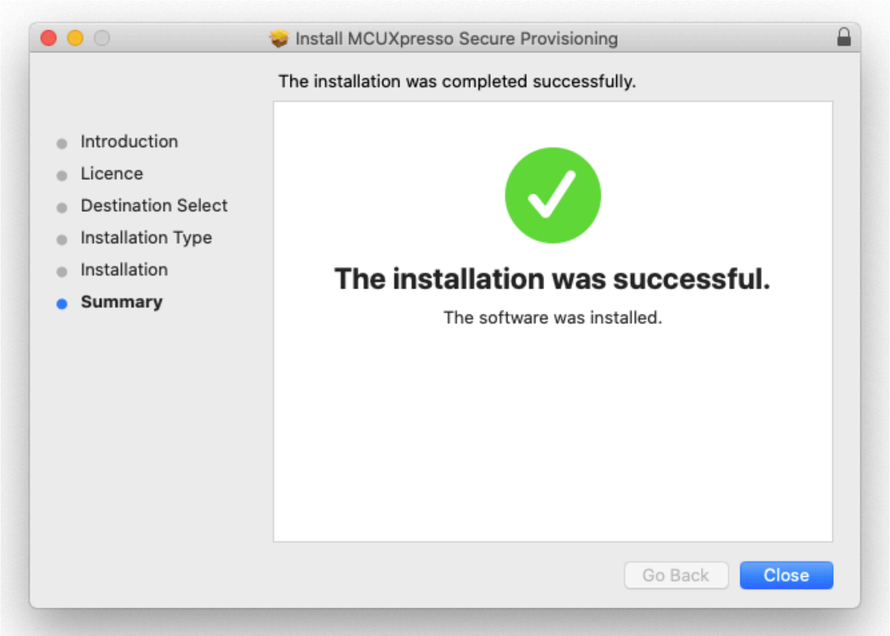
Username: user

Password:

Figure 14. Install software

9. Click **Continue**.

   Unless errors are reported, the **Summary** page confirms that the installation was completed successfully.
10. On the **Summary** page, click **Close**.



Install MCUXpresso Secure Provisioning

The installation was completed successfully.

The installation was successful.

The software was installed.

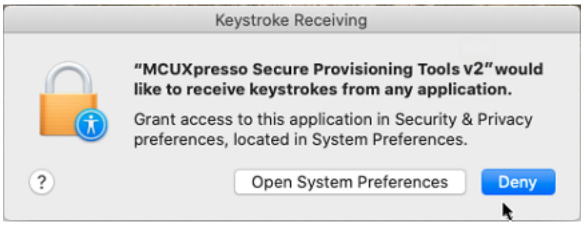
Figure 15. Summary

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**.



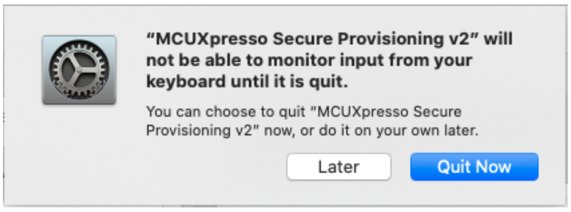
Keystroke Receiving

"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 <version>**, 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 (https://www.nxp.com/mcuxpresso/secure) 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>.deb
```

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

Note:

For trust provisioning, additional packages are necessary for the Smart Card access:

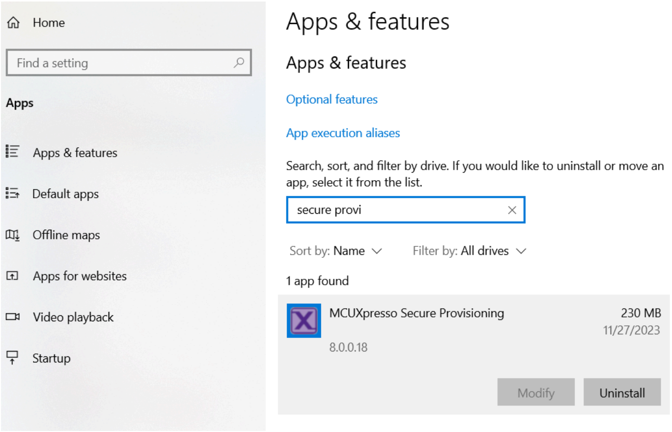
```
$ sudo apt install libpcsclite1 pcscd pcsc-tools
```

4.5 Uninstalling

4.5.1 Windows

Secure Provisioning Tool can be uninstalled in the following ways:

- by using Settings | Apps & features



Search, sort, and filter by drive. If you would like to uninstall or move an app, select it from the list.

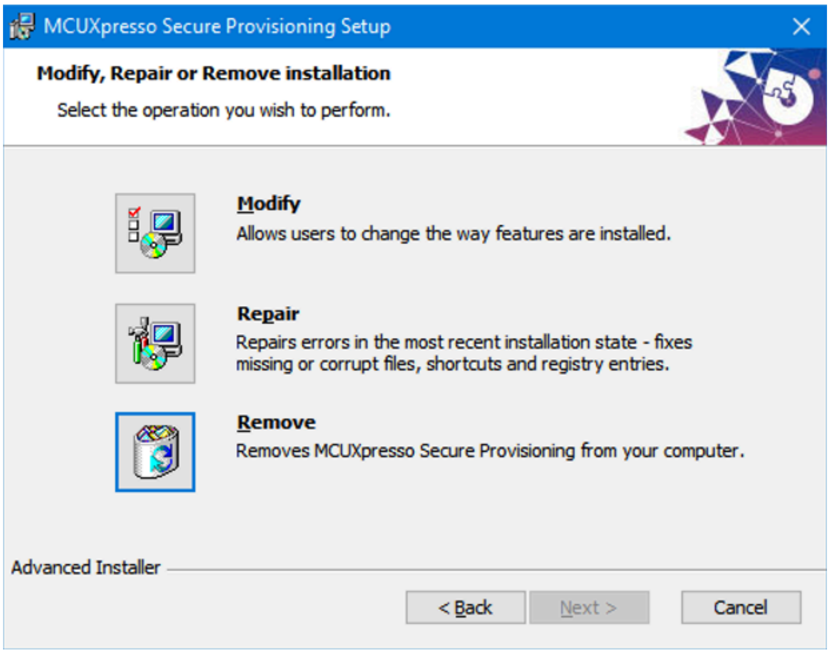
secure provi

1 app found

MCUXpresso Secure Provisioning 230 MB 11/27/2023 8.0.0.18

Figure 18. Settings

- by navigating into the `%APPDATA%\NXP Semiconductors` and then finding the appropriate MSI installer in the `MCUXpresso Secure Provisioning x.y.z.zz\install\` folder and choosing the **Remove** option in the wizard.



MCUXpresso Secure Provisioning Setup

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 from your computer.

Figure 19. MCUXpresso Secure Provisioning setup wizard

4.5.2 MacOS

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

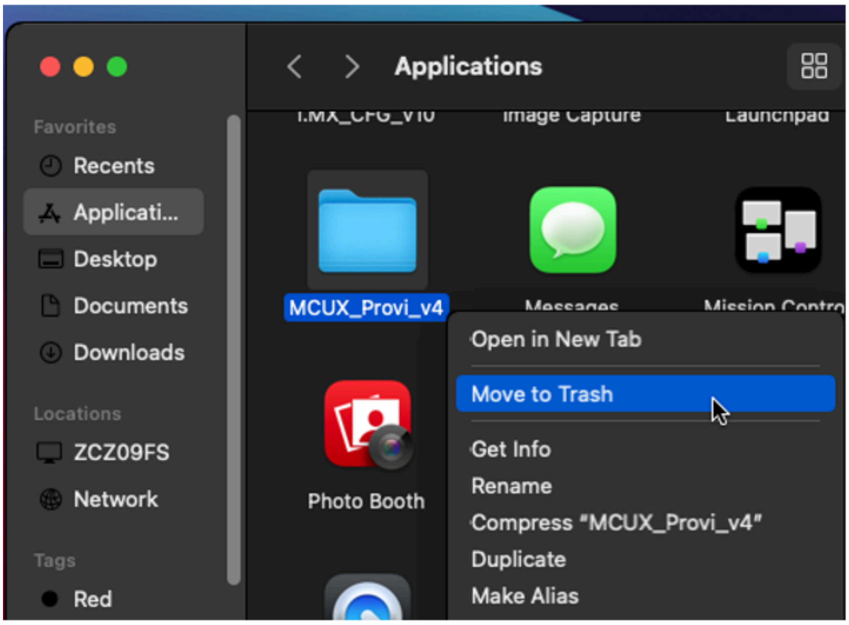


Figure 20. Applications

4.5.3 Linux

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 "mcuxpresso-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
+++-========================================-============-============-
====================================
rc  mcuxpresso-secure-provisioning-tools-v1 1.0.1        amd64        MCUXpresso
 Secure Provisioning Tools
rc  mcuxpresso-secure-provisioning-v2       2.0          amd64        MCUXpresso
 Secure Provisioning
rc  mcuxpresso-secure-provisioning-v2.1     2.1          amd64        MCUXpresso
 Secure Provisioning
ii  mcuxpresso-secure-provisioning-v3       3.0          amd64        MCUXpresso
 Secure Provisioning
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 -r mcuxpresso-secure-provisioning-v4
```

Several versions can be uninstalled at once:

```
sudo dpkg --remove mcuxpresso-secure-provisioning-v2 mcuxpresso-secure-
provisioning-v2.1
```

When additional packages have been installed for TP, you might consider to remove them as well:

```
$ sudo apt remove libpcsclite1 pcscd pcsc-tools
```

4.5.4 Remove configuration files

The user preferences are stored in the folder `<user home>\.nxp\secure_provisioning_v<version>\` and are not removed by uninstalling the product. These folders can be removed manually. For more details about preferences, see chapter "6.1.2 Preferences".

4.5.5 Remove restricted data

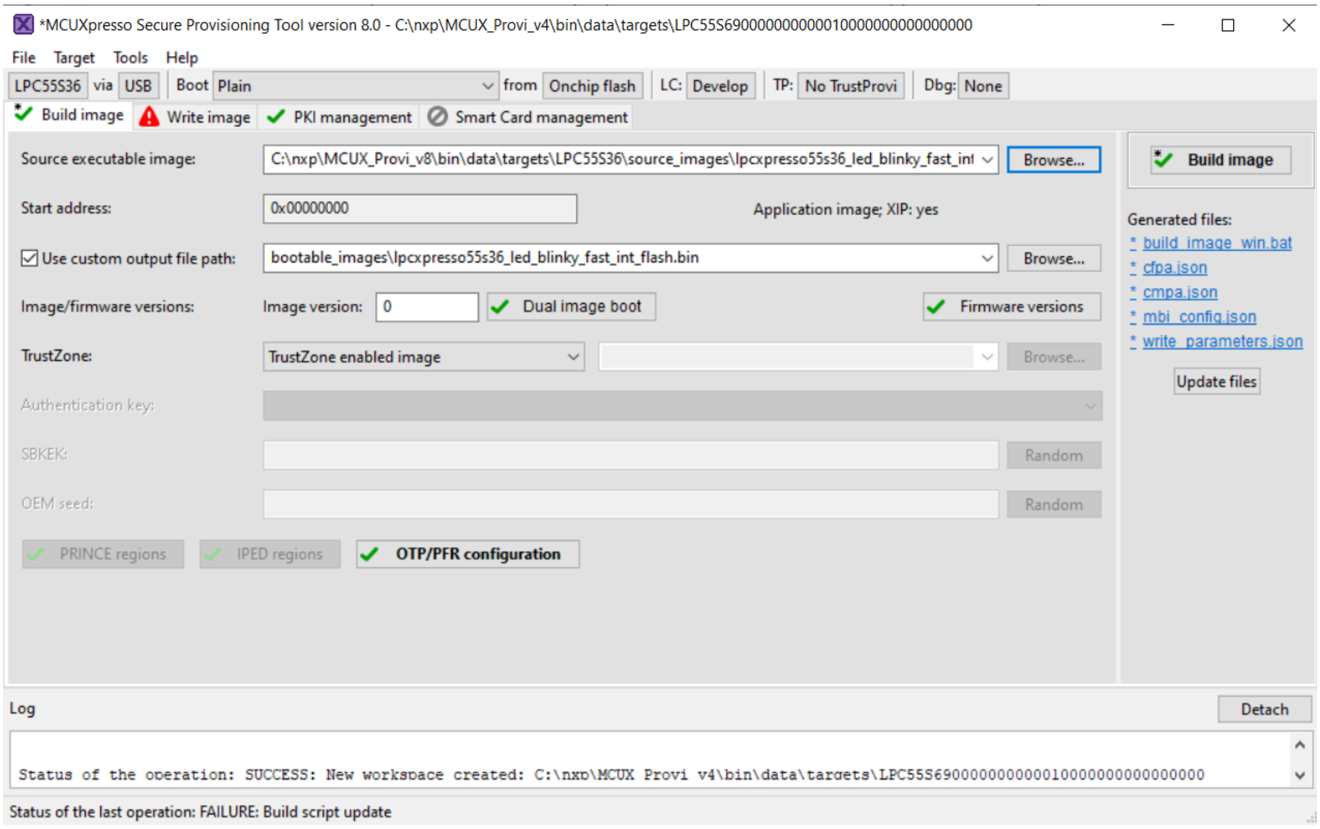
The restricted data are installed in the folder `<user home>\.nxp\secure_provisioning_restricted_data\` and are not removed by uninstalling the product. These folders can be removed manually.

5 User interface

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

- **Build image** allows building a bootable image.
- **Write image** allows writing a bootable image into the processor and securing it.
- **PKI management** allows generating authentication keys or configuring the Signature Provider.
- **Smart Card management** (if supported) allows configuring trust provisioning using a Smart Card.

The bottom part of the interface is occupied by the **Log** window and status line. Items that are not supported for the selected processor are not displayed in the tool. If an item is supported for the processor but is not meaningful in the current setting, it is disabled (gray). Items that contain any configuration problems are highlighted in red (errors) or yellow/orange (warnings). The problem is described in the tooltip.



*MCUXpresso Secure Provisioning Tool version 8.0 - C:\nxp\MCUX_Provi_v4\bin\data\targets\LPC55S690000000000010000000000000000

LPC55S36 via USB | Boot Plain from Onchip flash | LC: Develop | TP: No TrustProvi | Dbg: None

Source executable image: C:\nxp\MCUX_Provi_v8\bin\data\targets\LPC55S36\source_images\lpcxpresso55s36_led_blinky_fast_int

Start address: 0x00000000

Application image; XIP: yes

☑ Use custom output file path: bootable_images\lpcxpresso55s36_led_blinky_fast_int_flash.bin

Image/firmware versions: Image version: 0

TrustZone: TrustZone enabled image

Authentication key:

SBKEK:

OEM seed:

```
Status of the operation: SUCCESS: New workspace created: C:\nxp\MCUX Provi v4\bin\data\targets\LPC55S690000000000010000000000000000
```

Status of the last operation: FAILURE: Build script update

Figure 21. 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 title of the main window contains:

- 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** 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.
	Explore Workspace...	Opens the file explorer in the current workspace.
	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.
	Preferences	Opens the **Preferences** dialog. For more information, see Preferences.
	Exit	Exits SEC.
Target	This menu duplicates the operations available from the toolbar, see Section 5.1.5 for 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 Section 5.8.
	SB editor	Allows creating a custom Secure Binary file for secure updates. For details, see Section 5.1.8
Help	User help and additional general information	
	User Guide	Opens the User Guide.
	Community	Opens an NXP webpage with the blog, where you can find discussions about issues related to this tool.
	SPSDK Online Documentation	Displays a web page with documentation for the NXP Secure Provisioning SDK command-line tools.
	About	Displays information about the current version.

5.1.3 Preferences

User preferences are stored in the folder `<user home>\.nxp\secure_provisioning_v<version>\` and are shared for all workspaces. The preferences are backward-compatible, so for example SEC Tool v4 can load preferences from SEC Tool v3 if preferences for v4 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 read 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 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 SEC tool

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

The following options are available:

Default	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. By default, the sound signal is enabled.

Restricted data ...

Restricted data are distributed under a different license. The data can be downloaded from the NXP website and installed into the SEC tool. Among other, the data contains OTP Configuration details and 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 SEC installation folder. To start using the data, restart SEC.

Use restricted data from directory

Allows to control, whether the restricted data are used. The checkbox is enabled only if restricted data are installed for the selected processor.

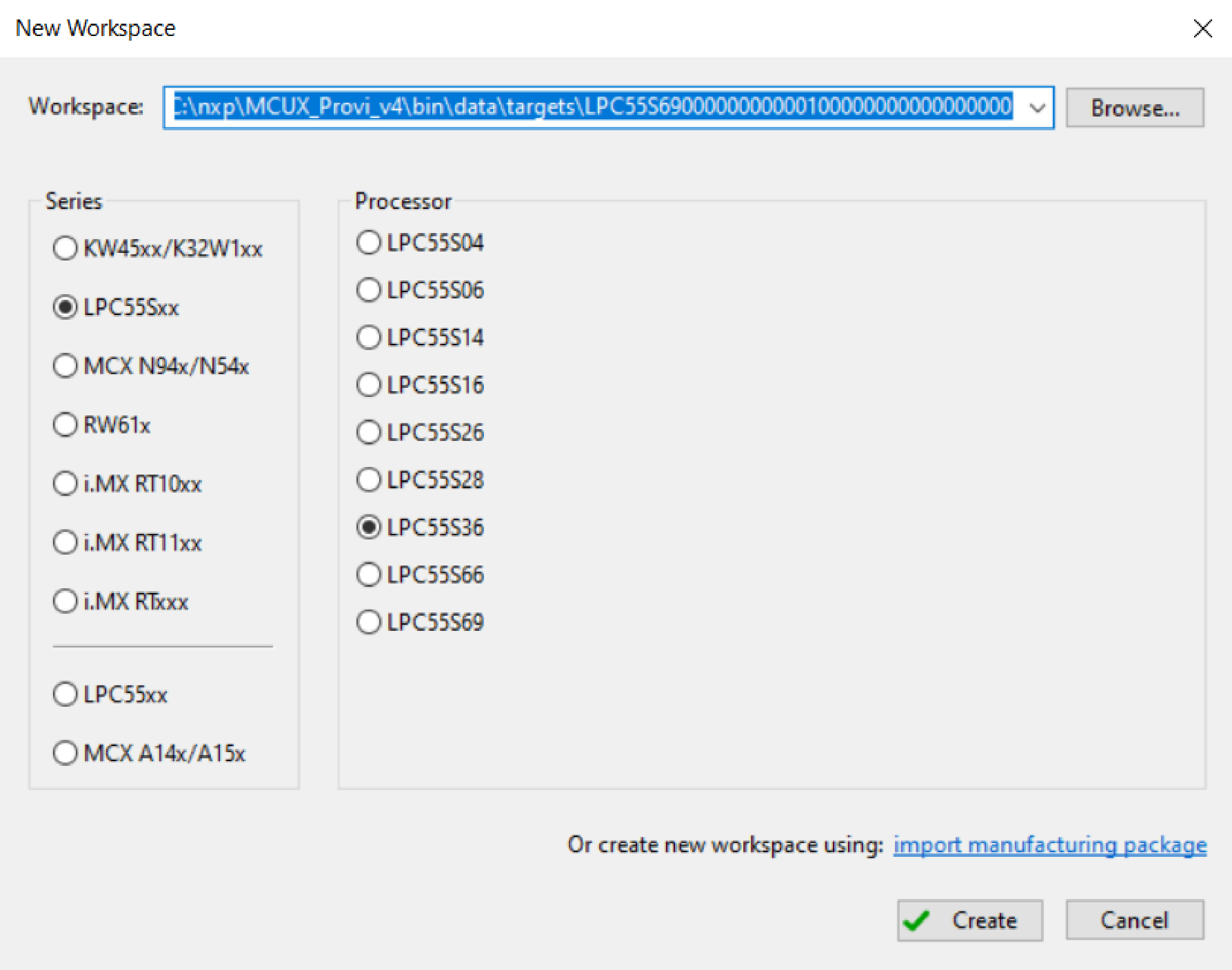
5.1.4 Workspaces

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.

To create a workspace, select **File>New Workspace ...** or **File>Import Manufacturing Package ...** from the **Menu bar**.



New Workspace

Workspace: C:\nxp\MCUX_Provi_v4\bin\data\targets\LPC55S69000000000000100000000000000000

Series

○ KW45xx/K32W1xx
● LPC55Sxx
○ MCX N94x/N54x
○ RW61x
○ i.MX RT10xx
○ i.MX RT11xx
○ i.MX RTxxx
○ LPC55xx
○ MCX A14x/A15x

Processor

○ LPC55S04
○ LPC55S06
○ LPC55S14
○ LPC55S16
○ LPC55S26
○ LPC55S28
● LPC55S36
○ LPC55S66
○ LPC55S69

Or create new workspace using: import manufacturing package

Figure 22. Create a workspace

To switch to a different workspace, select **File>Select Workspace ...** from the **Menu bar** and choose 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 Secure provisioning 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.

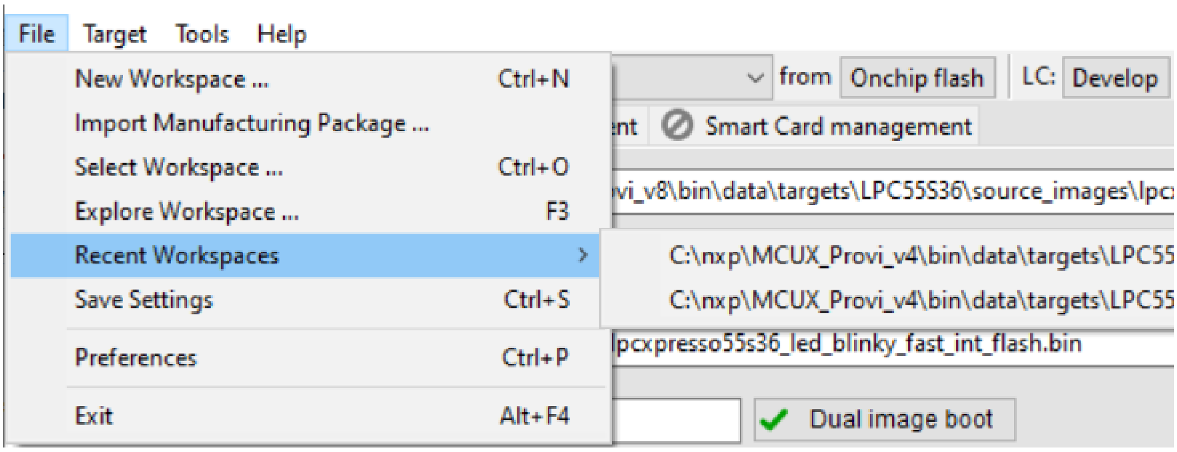

File
Target
Tools
Help
New Workspace ... Ctrl+N
Import Manufacturing Package ...
Select Workspace ... Ctrl+O
Explore Workspace ... F3
Recent Workspaces >
Save Settings Ctrl+S
Preferences Ctrl+P
Exit Alt+F4
C:\nxp\MCUX_Provi_v4\bin\data\targets\LPC55
C:\nxp\MCUX_Provi_v4\bin\data\targets\LPC55
from Onchip flash
LC: Develop
nt
Smart Card management
vi_v8\bin\data\targets\LPC55S36\source_images\lpc
lpcxpresso55s36_led_blinky_fast_int_flash.bin
✓ Dual image boot

Figure 23. 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

bd_files Generated command files used by nxpimage during bootable image generation step (nxpimage input).

bootable_images Intermediate and final bootable images (nxpimage output). The nopadding binary starts at the address IVT, while the regular binary includes everything from the beginning of the boot device.

configs Configuration files - OTFAD/IEE config file (YAML), BEE user keys config file (YAML)

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.

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 nxpimage tool. The DEK key file is used during write image to generate the key blob for the encrypted HAB boot type.

gen_sb CMPA and CFPA pages (BIN) used to configure secure boot pages and SB KEK keys (BIN and TXT) for the key store.

gen_scripts Temporary scripts for tool operation.

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.
- Log for all executed commands, recorded in file log.txt.

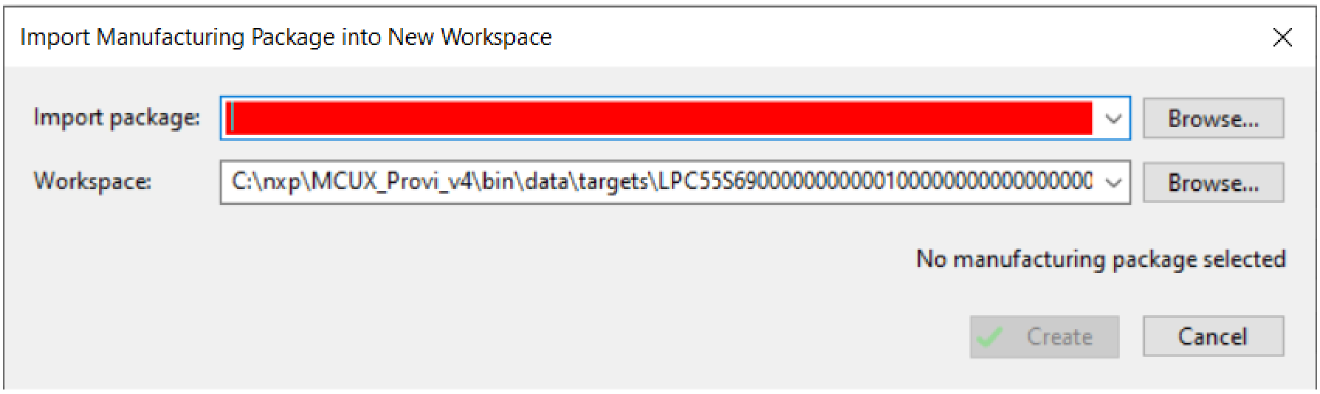
source_images Primarily intended as a folder to store input images provided by users. Also used by the tool to store input images, if input format conversion is needed.

trust_provisioning Trust provisioning files (supported only if the processor supports trust provisioning)

trustzone_files TrustZone-M configuration files (JSON or BIN) used by nxpimage during bootable image generation step (nxpimage input).

5.1.4.1 Manufacturing workspace

For manufacturing operation, the tool supports a simplified user interface optimized for factory manufacturing. Manufacturing workspace can be created for a given manufacturing package using command **main menu > File > Import Manufacturing Package** …



Import Manufacturing Package into New Workspace

Import package:

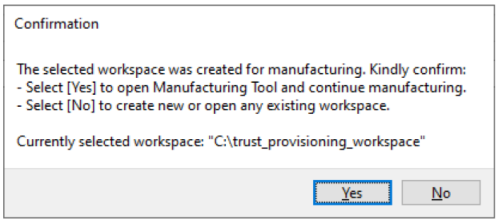
Workspace: C:\nxp\MCUX_Provi_v4\bin\data\targets\LPC55S690000000000000100000000000000000

No manufacturing package selected

Figure 24. Import Manufacturing Package dialog

In the dialog, select the manufacturing package to be imported and the workspace folder (new or empty directory). During the import, the workspace directory is created, and the package is imported into the directory. The tool allows importing manufacturing package created in the same version of the tool.

Once the workspace is created (or reopened), the Manufacturing Tool is displayed, and the rest of the tool functionality is not available. If 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 25. Confirmation to reopen manufacturing

If the manufacturing package contains 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.

5.1.4.2 Sharing and copying workspaces

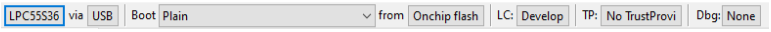
It is recommended to store all used files in the workspace. The `settings.sptjson` file contains all paths relative to the workspace root folder, so if you open settings on another computer, you can still regenerate all scripts.

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 environment variable must be specified only if the script is copied outside the workspace.

Remember that these variables are used in the build and write scripts, but can't be used in configuration files (BD, JSON, and so on). The configuration files must be updated manually.

5.1.5 Toolbar

The **Toolbar** offers a quick selection of basic settings.



LPC55S36 via USB | Boot Plain | from Onchip flash | LC: Develop | TP: No TrustProvi | Dbg: None

Figure 26. Toolbar

Processor

Shows the chosen processor. Click the button to switch the processor. You can switch to a processor from the same device family only. To select a processor from a different family, create a 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. For more information, see the attached SEC-Tool-Features.xls.

Boot device (from)

Click the button to open boot memory configuration. For more information, see Section 5.1.7.

Life cycle

Allows 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 selection of trust provisioning type and enabling it for the trust provisioning operation. This button is visible only if trust provisioning is supported for the processor. The supported types depend on processors (for more information, Section 5.1.8):

- Smart Card - the secrets are stored on the Smart Card
- Device HSM - the secrets are encrypted using keys stored in the processor

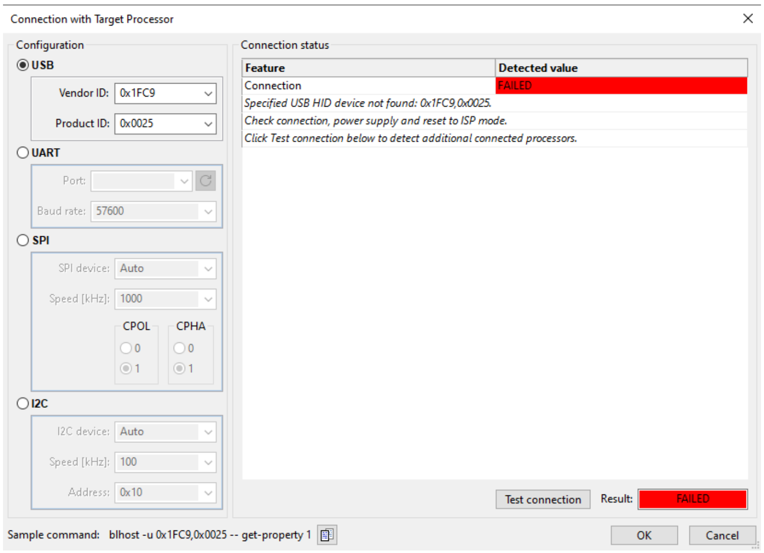
Debug probe

Allows selecting the debug probe connected to the computer; see Section 5.1.9 for details.

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**.



Connection with Target Processor

Configuration

● USB

Vendor ID: 0x1FC9

Product ID: 0x0025

○ UART

Port:

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

Connection status

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 27. Connection dialog with errors and proposed next steps

It contains the following options (supported connection types are processor-specific, see details in the attachment to this document):

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 via SPI or I2C connection using the LIBUSB interface available on:

- MCU-Link Pro (http://www.nxp.com/pages/:MCU-LINK-PRO)
- LPC-Link2 (https://www.nxp.com/design/microcontrollers-developer-resources/lpc-link2:OM13054). 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 MCUX 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.

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.
Life cycle	Life cycle that was detected in the connected processor.

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**. If the connection detection fails, the SEC tool tries to detect additional processors connected to the computer. It may help to find a wrong SEC configuration, such as a wrongly selected processor or wrongly selected VID/PID. In case you need more information about the failure, you can use SEC in verbose mode and see the console view with details of the operation.

At the bottom of the connection dialog, there is the "Sample blhost/sdphost command" that allows running the corresponding SPSDK command-line tools with specified arguments. The button on the right side copies the command with arguments into the clipboard.

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. The selection contains all memory types but unsupported types are disabled.
Predefine template	This part allows selection of the boot memory configuration template. The list is specific for each memory type and contains memories available on NXP evaluation boards. After opening the dialog, the option contains value that matches the current configuration (if any), or is empty, the first item in the drop-down menu is a memory used on the evaluation board (if applicable);
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
Convert to FCB	This option is available for FlexSPI NOR only. The button allows converting simplified FlexSPI NOR configuration into full “Flash Control Block”

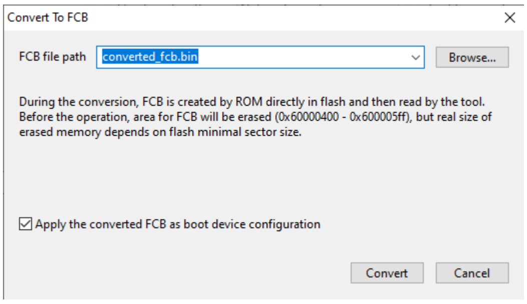
5.1.7.1 FlexSPI NOR

FlexSPI NOR flash can be configured in two ways:

- by using the flashloader/ROM based simple configuration in the **Boot Memory Configuration** dialog
- by using the complete FCB (FCB binary)

For a simple configuration, the user can modify the values suggested in the dialog box. To create your own FCB from SEC Tool, do a simple setup and use the **Convert to FCB** button in the bottom-right corner. It opens the **Convert to FCB** dialog box. The **Convert** button in the **Convert to FCB** dialog is used to test a simple configuration from the boot device configuration and to start the conversion process. 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 FCB** dialog offers a checkbox to use the created FCB binary file as a FlexSPI NOR/**user FCB file** (required for Dual boot). If unchecked, only the conversion is done.



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, area for FCB will be erased (0x60000400 - 0x600005ff), but real size of erased memory depends on flash minimal sector size.

☑ Apply the converted FCB as boot device configuration

Figure 28. Conversion dialog

FCB can be also 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.

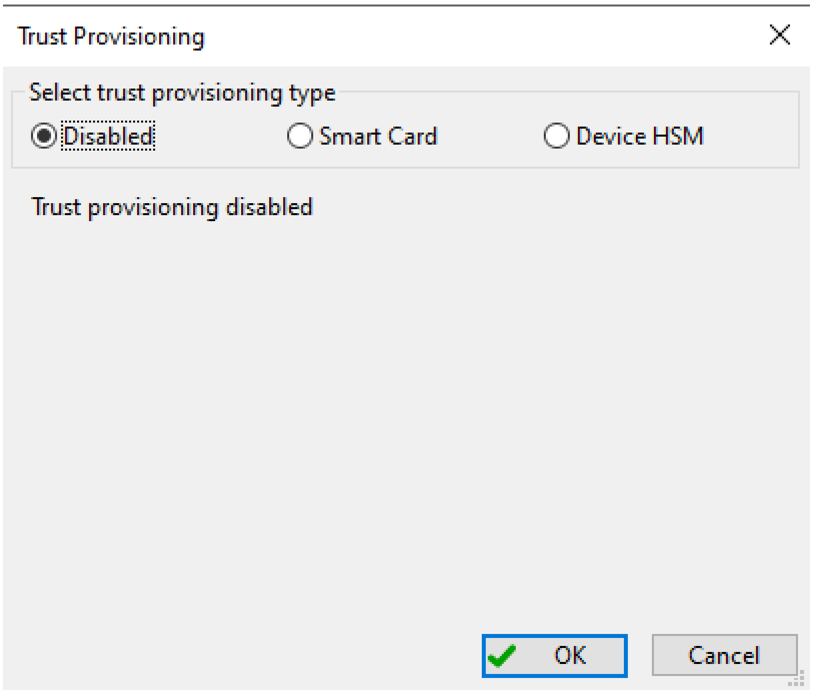
5.1.7.2 OnChip RAM

The SEC tool allows creating images executed in internal RAM, which might be useful for chip (re-)configuration. 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.*

5.1.8 Trust provisioning

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



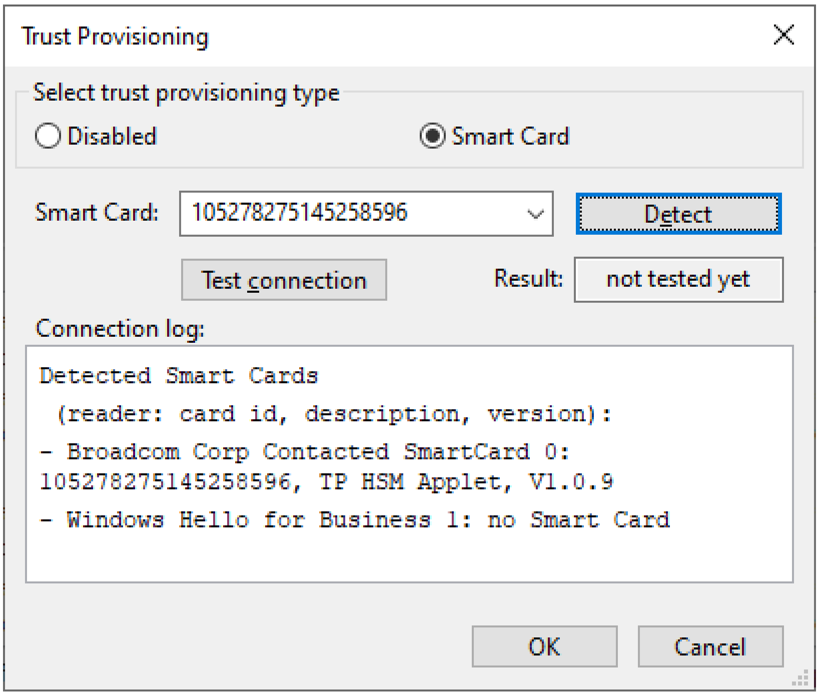
Trust Provisioning

Select trust provisioning type

● Disabled ○ Smart Card ○ Device HSM

Trust provisioning disabled

Figure 29. Trust provisioning disabled



Trust Provisioning

Select trust provisioning type

○ Disabled ● Smart Card

Smart Card: 105278275145258596

Result: not tested yet

Connection log:

```
Detected Smart Cards
 (reader: card id, description, version):
- Broadcom Corp Contacted SmartCard 0:
105278275145258596, TP HSM Applet, V1.0.9
- Windows Hello for Business 1: no Smart Card
```

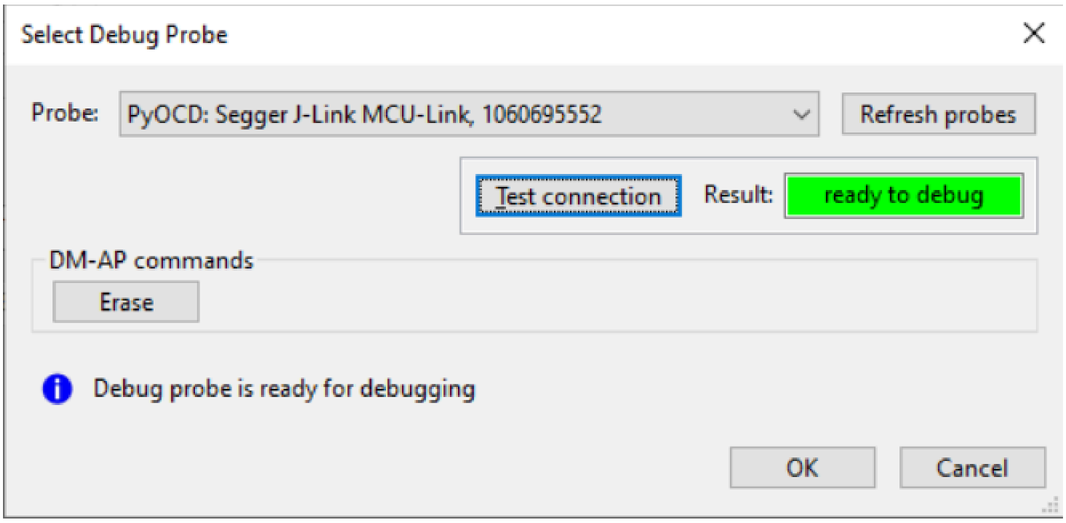
Figure 30. Smart Card provisioning type

Use the **Detect** function to scan for all connected Smart Cards. Detected Smart Cards are then listed in the Smart Card selection. For details, see the Connection log.

Use the **Test connection** function to verify that the selected Smart Card is connected and can be used for trust provisioning.

Note: *Windows 10 has a new security feature called "Windows Hello for Business". Due to this feature, the 'Windows Hello for Business 1' Smart Card reader is detected.*

5.1.9 Debug Probe Selection dialog



Select Debug Probe

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

Debug probe is ready for debugging

Figure 31. 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.

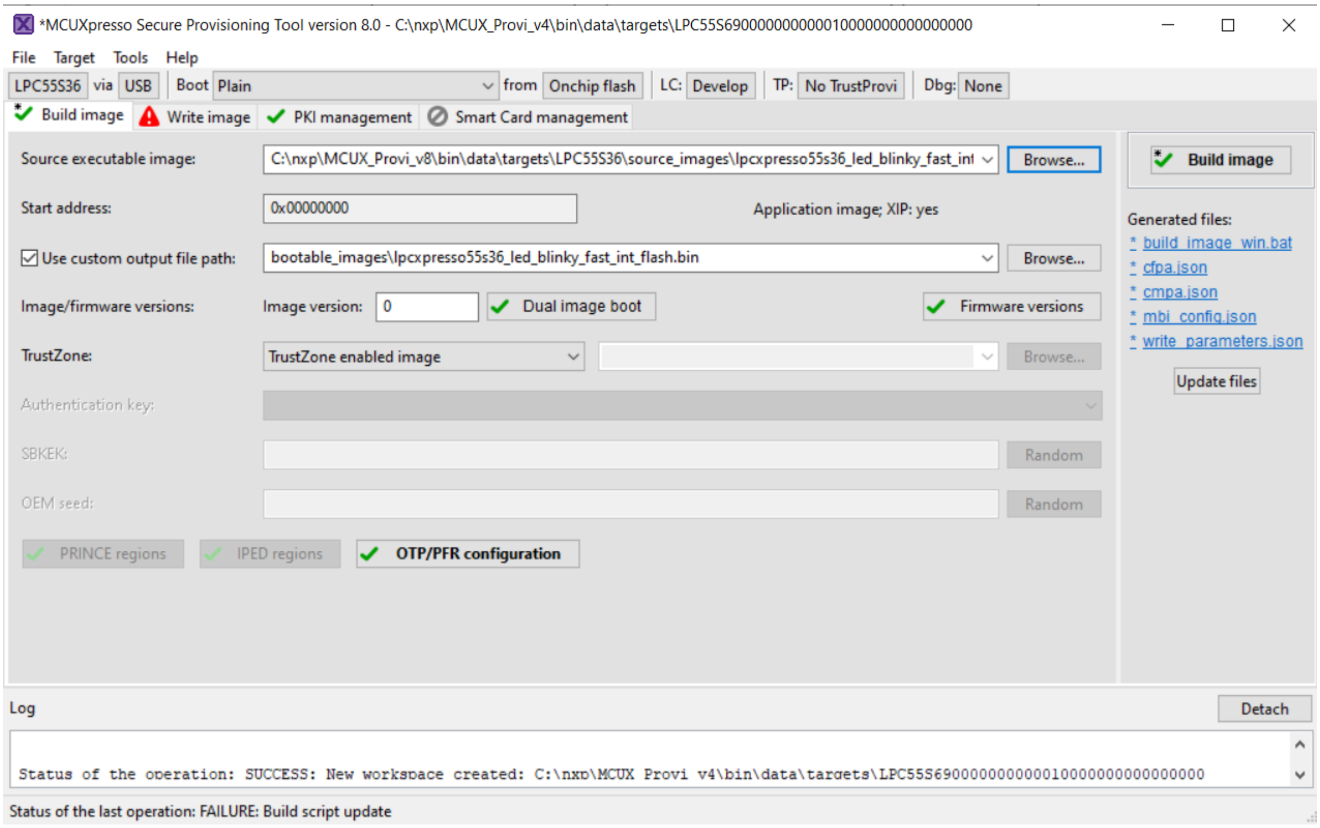
The **Test connection** button provides information on whether the debug can be started for the selected debug probe. The possible results are:

- **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

Erase button allows erasing the internal flash (mass erase). See reference manuals for 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. The **build 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 8.0 - C:\nxp\MCUX_Provi_v4\bin\data\targets\LPC55S690000000000001000000000000000

LPC55S36 via USB | Boot: Plain from Onchip flash | LC: Develop | TP: No TrustProvi | Dbg: None

Source executable image: C:\nxp\MCUX_Provi_v8\bin\data\targets\LPC55S36\source_images\lpcxpresso55s36_led_blinky_fast_int

Start address: 0x00000000

Application image; XIP: yes

☑ Use custom output file path: bootable_images\lpcxpresso55s36_led_blinky_fast_int_flash.bin

Image/firmware versions: Image version: 0

TrustZone: TrustZone enabled image

Authentication key:

SBKEK:

OEM seed:

Generated files:
- build_image_win.bat
- cfpa.json
- cmpa.json
- mbi_config.json
- write_parameters.json

Log

```
Status of the operation: SUCCESS: New workspace created: C:\nxp\MCUX Provi v4\bin\data\targets\LPC55S690000000000001000000000000000
```

Status of the last operation: FAILURE: Build script update

Figure 32. Build image (LPC55Sxx)

Source executable image Chooses the input executable file. For more information about the input image format, see Section 5.2.1.

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

Application/Bootable image 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 Section 5.2.5

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. 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. Available for RTxxx, RT116x/7x, RT118x, LPC55S36, LPC553x, KW45xx, K32 W1xx.

Firmware version

Opens the configuration dialog of Firmware versions (Secure and Non-secure). Both the Secure and Non-secure firmware versions specify the image version of the image to be written. These versions are checked during the SB file or AHAB image uploading against values in CFPA (the uploaded version must be equal to or greater than the appropriate value in the CFPA) processor or against fuse values processors. The Secure firmware version specifies the secure image version checked during booting by ROM.

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 Editor (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 Section 6.2.4.

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 preset data.
- **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 Section 5.2.3.

Authentication key

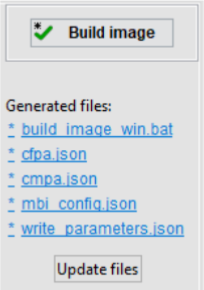
Signs the image with the specified key. The key can also be used for the authentication of the SB file. This option is only applicable to authenticated and encrypted boot modes and offers a selection of keys generated in the **PKI management** view.

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) or encrypted (AHAB) boot types.
Key source	A key source for signing the image.
User key	For OTP key source master key used to derive other keys. For PUF KeyStore user key is used to sign the image. Only available for Signed boot types.
SBKEK, SB3KDK, or CUST_MK_SK	A key is used as a key-encryption key to handle SB file. Only available for secured boot types. For RTxxx, 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 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 dialogues 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 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 to specify, which flash regions will be encrypted.
OTP/PFR/IFR configuration	Opens the OTP/PFR/IFR configuration dialog



*✓ Build image

Generated files:

- * build_image_win.bat
- * cfpa.json
- * cmpa.json
- * mbi_config.json
- * write_parameters.json

Update files

Figure 33. 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 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.*

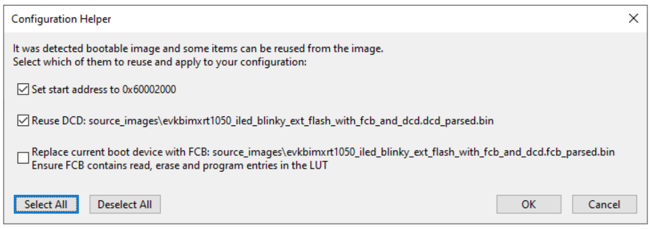
This information is updated in the background with some delay, so it might take a couple of seconds before the real status is displayed. Until the information about the file is updated, the file icon is **?** (question mark).

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

5.2.1 Source image formats

SEC supports several formats for source image: ELF, HEX, BIN, or SREC/S19. 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.

By default, the source image may not contain any boot header. For RT10xx, RT116x, RT117x bootable image can be used too; such image is parsed and if contains DCD, FCB or XMCD sections, these parts can be reused 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 34. Configuration Helper

5.2.2 XMCD Editor

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

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

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

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

5.2.3 TrustZone configuration file

TrustZone and related features of the MCU can be pre-configured by data from 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.

Note: *TrustZone feature is available only for LPC55Sxx, RTxxx, KW45xx, and K32W1xx devices.*

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 **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 MCUXpresso Config Tools User Guide https://www.nxp.com/webapp/Download?colCode=GSMCUXCTUG).
5. In **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 data\targets\LPC55S##\ and data\targets\MIMXRT###\ subfolders.

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 **Enabled (custom data)**.
4. Click **Browse** to navigate to the location of the stored JSON file and select **Open** to import it.

5.2.4 OTP/PFR/IFR configuration

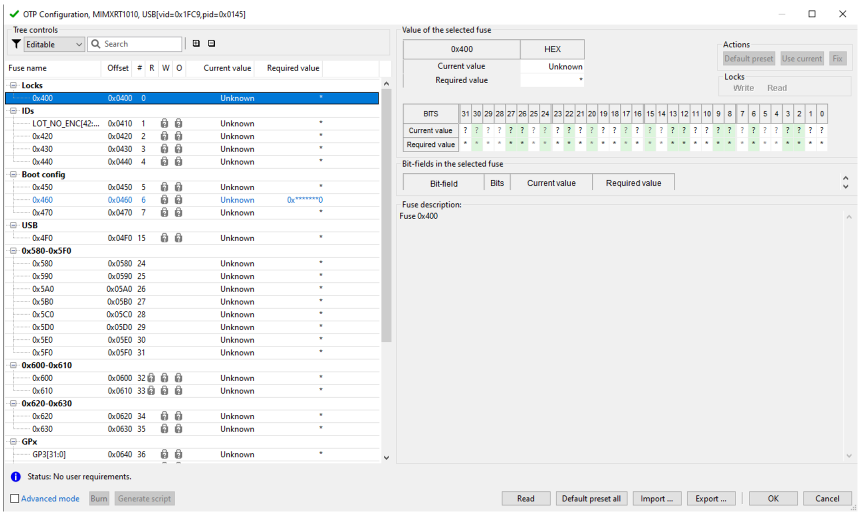
The configuration dialog allows configuring:

- one time programmable fuses
- CFPA and CMPA pages from Protected Flash Region (PFR)
- ROMCFG page from Information Flash Region (IFR)

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) and Information Flash Region (IFR) Configurations, the configurable item is called a **field**. The content of the dialog depends on the selected processor.



✓ OTP Configuration, MIMXRT1010, USB[vid=0x1FC9,pid=0x0145]

Tree controls: Editable

Fuse name	Offset	#	R	W	O	Current value	Required value
⊟ **Locks**							
0x400	0x0400	0				Unknown	*
⊟ **IDs**							
LOT_NO_ENC[42:...	0x0410	1		🔒	🔒	Unknown	*
0x420	0x0420	2		🔒	🔒	Unknown	*
0x430	0x0430	3		🔒	🔒	Unknown	*
0x440	0x0440	4		🔒	🔒	Unknown	*
⊟ **Boot config**							
0x450	0x0450	5		🔒	🔒	Unknown	*
0x460	0x0460	6		🔒	🔒	Unknown	0x*******0
0x470	0x0470	7		🔒	🔒	Unknown	*
⊟ **USB**							
0x4F0	0x04F0	15		🔒	🔒	Unknown	*
⊟ **0x580-0x5F0**							
0x580	0x0580	24				Unknown	*
0x590	0x0590	25				Unknown	*
0x5A0	0x05A0	26				Unknown	*
0x5B0	0x05B0	27				Unknown	*
0x5C0	0x05C0	28				Unknown	*
0x5D0	0x05D0	29				Unknown	*
0x5E0	0x05E0	30				Unknown	*
0x5F0	0x05F0	31				Unknown	*
⊟ **0x600-0x610**							
0x600	0x0600	32	🔒	🔒	🔒	Unknown	*
0x610	0x0610	33	🔒	🔒	🔒	Unknown	*
⊟ **0x620-0x630**							
0x620	0x0620	34		🔒	🔒	Unknown	*
0x630	0x0630	35		🔒	🔒	Unknown	*
⊟ **GPx**							
GP3[31:0]	0x0640	36		🔒	🔒	Unknown	*

Status: No user requirements.

Value of the selected fuse

0x400	HEX
Current value	Unknown
Required value	*

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	0
Current value	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?
Required value	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*

Bit-fields in the selected fuse

Bit-field	Bits	Current value	Required value

Fuse description:
Fuse 0x400

Figure 35. OTP Configuration

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

The configuration window contains three main areas:

- Table of all fuses/fields 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.

5.2.4.1 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 ROMCFG (ROM Bootloader configurations). 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 Section 5.2.4.8.
Current value	The current value of the item read from the processor (hexadecimal)
Required value	The 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: *If the information is not applicable for the configuration, some columns might not be displayed.*

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 on several rows in the table and have a common # index (fuse index).*

5.2.4.2 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.

5.2.4.3 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 Section 5.2.4.7)
- 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)

5.2.4.4 Buttons

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

Lock after write checkbox	Lock the item after write. (see Section 5.2.4.4)
Default	Remove user requirements for the item and apply a default value.
Use current	Copy the current item value as a new requirement
Fix problems	Fix the displayed problems automatically

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 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.
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.
OK	Accept changes and return to **Write image**.
Cancel	Close the dialog without accepting changes.

5.2.4.5 Read from connected device

Before you start the OTP/PFR/IFR 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 chapter 6.3.1.7 Locks for details)

5.2.4.6 Required value

Items that must be written based on 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).

5.2.4.7 Burn fuse

In the OTP configuration, it is possible to select the source, where the fuse is burnt:

- Write script
- SB file
- Device HSM firmware

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

5.2.4.8 Locks

Locks are available in OTP Configuration to lock fuses. A lock block specifies access restriction to the fuse - usually, 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 write-access restriction to the ROMCFG block as each 16-bytes block 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 write-lock for a single fuse. Additionally, some fuses can be written only once and 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 write restriction to the ROMCFG block.

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 2. Lock icons

Icon	Description
No icon	Fuse does not support a corresponding lock
🔒	Lock status unknown
🔓	Fuse access unlocked
🔒	Fuse access 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: 🔒

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.

5.2.4.9 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 inversed value of another item
- One bit field contains 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.

5.2.4.10 Validation and problem resolution

The configuration provides validation of the required values and highlights the following problems:

Errors (highlighted in red)

- The required value cannot be written. It is in conflict with the current value.
- The required value cannot be written, because there is a write-lock applied for the fuse and the required value does not match with the current value.
- The required value for the calculated item does not match the value calculated by SEC.
- PFRC: PFR Configuration internally uses PFR Checker from SPSDK to detect some kind of problems in the user configuration. If any problem is detected, it is reported on the status line.
- Other processor-specific validations.

Warnings (yellow background or highlighted in orange)

- Write-lock fuse will be burned, but the required value(s) for locked fuse(s) are not fully specified.
- The calculated value does not match with the current value in the processor (it is reported only if there is no required value for the calculated value; otherwise an error is reported for the required value).
- The reserved bit field value is selected for the bit field.

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.

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.*

5.2.4.11 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 section Section 5.2.4.12)
- Generate the script to write required values
- Modify all required values, even the ones preset by SEC-PFR Configuration: Clear 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.*

5.2.4.12 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.

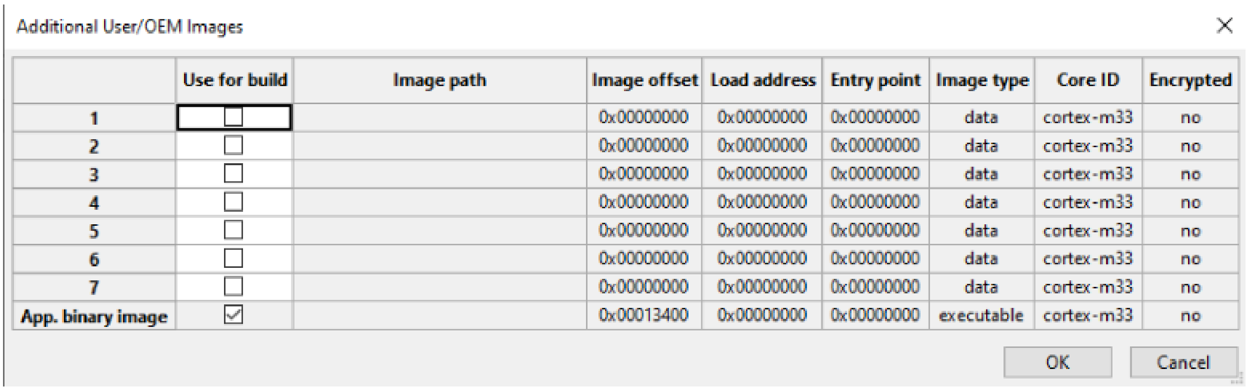
5.2.4.13 PFR/IFR and OTP differences

- PFR Configuration contains two pages: CFPA and CMPA. IFR Configuration contains one-page ROMCFG.
- Items in OTP Configuration are called “fuses” while items in PFR/IFR Configuration 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-bytes blocks, so completed 16-bytes 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 only for fuses and the IFR ROMCFG block.
- CFPA and CMPA pages can be written multiple times whereas the ROMCFG block can be written only once.

5.2.5 Additional User/OEM AHAB images

This dialog allows to specify up to 8 user images into the AHAB container. The user can add either data images (data, dcd, …) or executable images (executable normal boot image, executable fast boot image, …) in the purpose of multicore applications.

The configuration is represented as a table, where each row represents one user image. Columns represent configurable features for each image, description for each feature can be found in the tooltip. The last image is reserved for the application executable image, which is automatically updated according to the build tab.



Additional User/OEM Images

	Use for build	Image path	Image offset	Load address	Entry point	Image type	Core ID	Encrypted
1	□		0x00000000	0x00000000	0x00000000	data	cortex-m33	no
2	□		0x00000000	0x00000000	0x00000000	data	cortex-m33	no
3	□		0x00000000	0x00000000	0x00000000	data	cortex-m33	no
4	□		0x00000000	0x00000000	0x00000000	data	cortex-m33	no
5	□		0x00000000	0x00000000	0x00000000	data	cortex-m33	no
6	□		0x00000000	0x00000000	0x00000000	data	cortex-m33	no
7	□		0x00000000	0x00000000	0x00000000	data	cortex-m33	no
App. binary image	☑		0x00013400	0x00000000	0x00000000	executable	cortex-m33	no

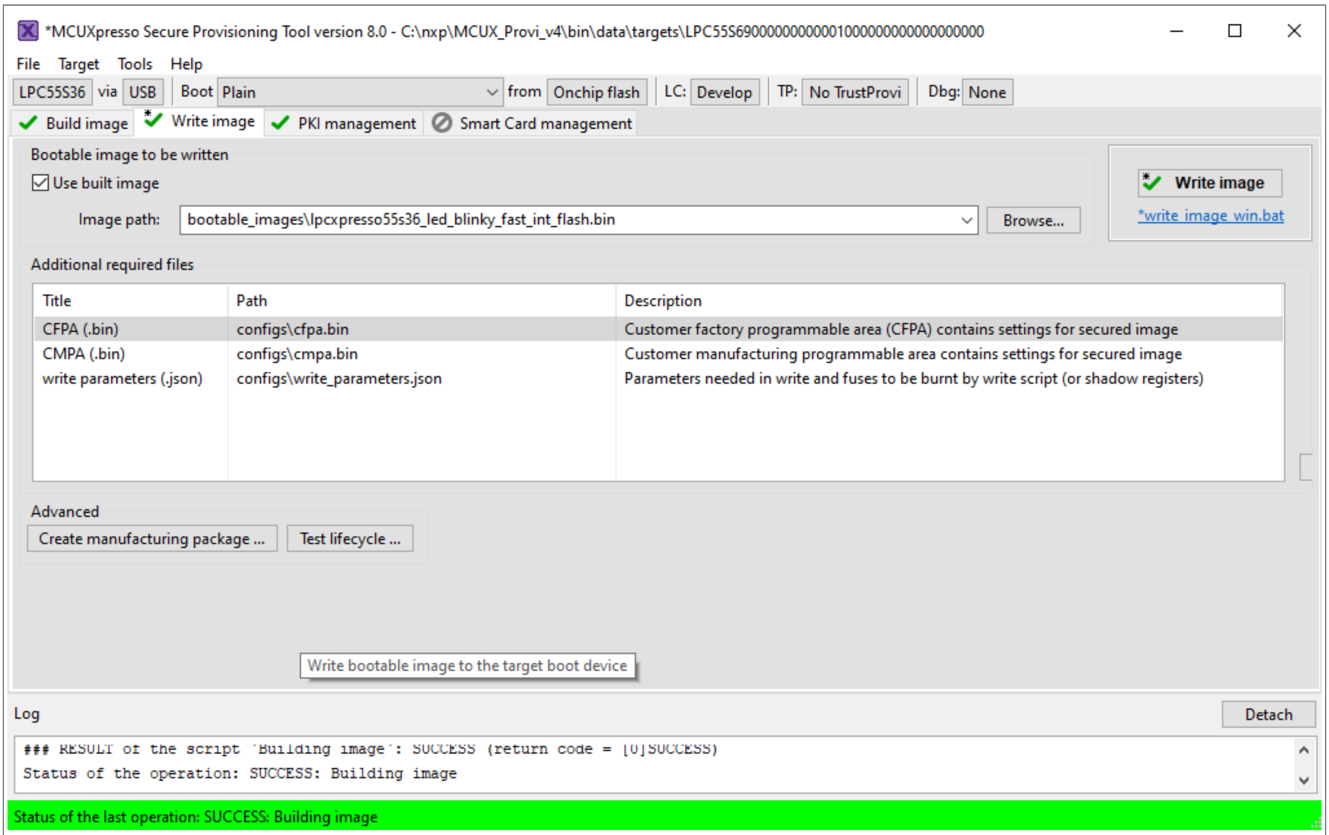
Figure 36. Additional User/OEM Images configuration

The dialog provides only simple validation of the input data, such as:

- Input value format
- Image offset overlapping
- Check if each core has only one single executable image
- Alignment of the image offset

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 8.0 - C:\nxp\MCUX_Provi_v4\bin\data\targets\LPC55S69000000000000010000000000000000
File
Target
Tools
Help
Build image
Write image
PKI management
Smart Card management
Browse...
Write image
*write_image_win.bat
Create manufacturing package ...
Test lifecycle ...
Write bootable image to the target boot device
Log
Detach

LPC55S36 via USB | Boot Plain | from Onchip flash | LC: Develop | TP: No TrustProvi | Dbg: None

Bootable image to be written

☑ Use built image

Image path: bootable_images\lpcxpresso55s36_led_blinky_fast_int_flash.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.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

```
### RESULT of the script 'Building image': SUCCESS (return code = [0]SUCCESS)
Status of the operation: SUCCESS: Building image
```

Status of the last operation: SUCCESS: Building image

Figure 37. Write image (LPC55Sxx)

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 SB capsules. For RT processors, the binary image must be in "nopadding" form without the FCB header.

Additional input files

Display 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 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. Supported debugger probes are Jlink, pyOCD and PEmicro.

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. The button and label contain an asterisk (*), if the write script is not updated on the disk. Both the button and the label update the script on the click, but the button also starts the script.

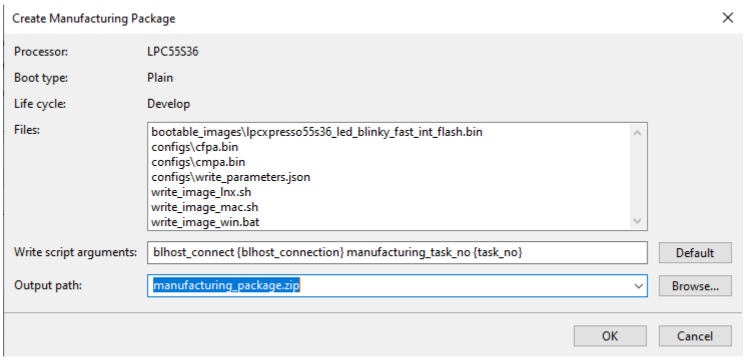
Before clicking the **Write image** button, ensure that the board is connected and configured to the ISP mode. If any irreversible operation is done by the write script, a confirmation dialog with details appears.

5.3.1 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 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



Create Manufacturing Package

Processor: LPC55S36

Boot type: Plain

Life cycle: Develop

Files:

```
bootable_images\lpcxpresso55s36_led_blinky_fast_int_flash.bin
configs\cfpa.bin
configs\cmpa.bin
configs\write_parameters.json
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

Figure 38. Create manufacturing package dialog

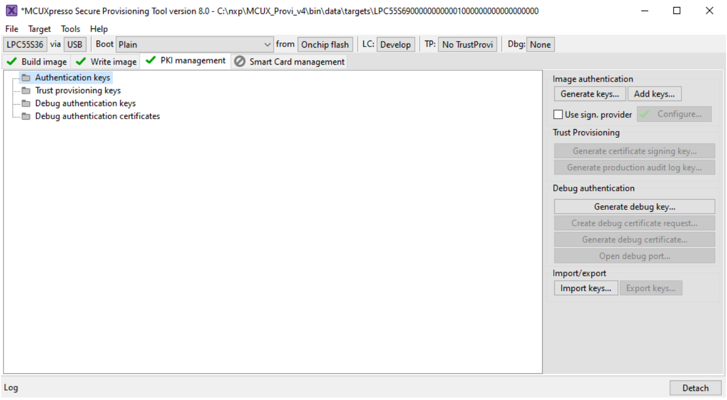
Note: *Manufacturing package is not supported if shadow registers are used.*

In the factory, the manufacturing package can be imported using **main menu > File > Import Manufacturing Package….** For more information, see Section 5.1.4.1.

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



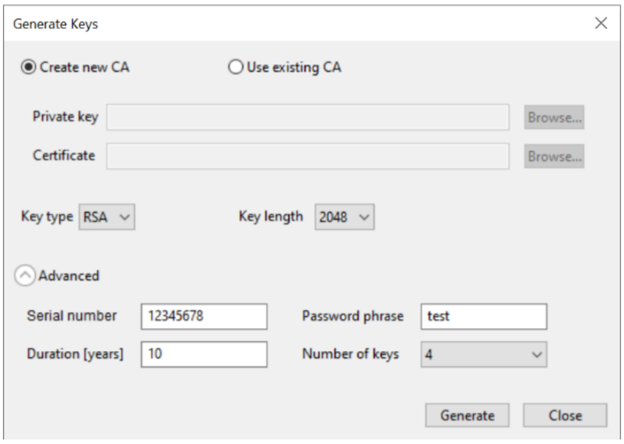
*MCUXpresso Secure Provisioning Tool version 8.0 - C:\nxp\MCUX_Provi_v4\bin\data\targets\LPC55S6900000000000010000000000000000

LPC55S36 via USB | Boot Plain from Onchip flash | LC: Develop | TP: No TrustProvi | Dbg: None

Figure 39. 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.



Generate Keys

● Create new CA ○ Use existing CA

Private key

Certificate

Key type: RSA

Key length: 2048

Advanced

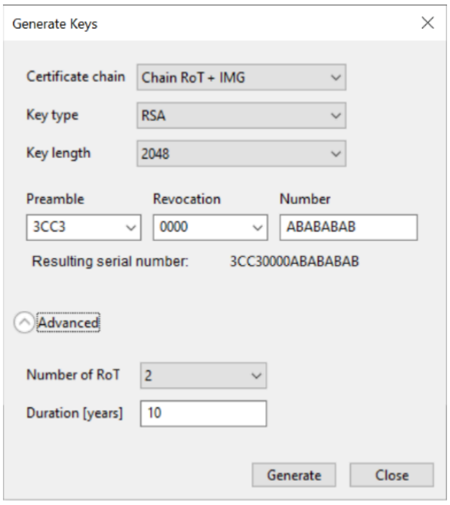
Serial number: 12345678

Password phrase: test

Duration [years]: 10

Number of keys: 4

Figure 40. Generate keys (RT10xx)



Generate Keys

Certificate chain: Chain RoT + IMG

Key type: RSA

Key length: 2048

Preamble: 3CC3

Revocation: 0000

Number: ABABABAB

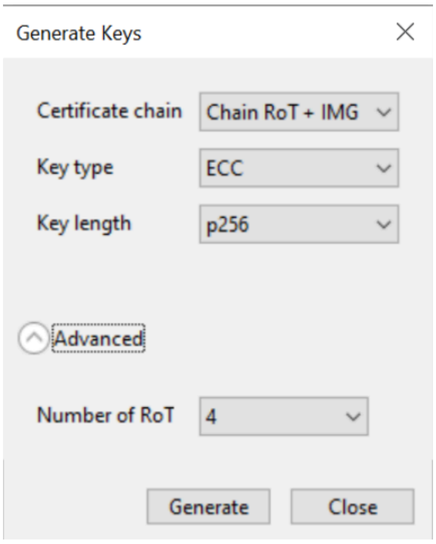
Resulting serial number: 3CC30000ABABABAB

Advanced

Number of RoT: 2

Duration [years]: 10

Figure 41. Generate RSA keys



Generate Keys

Certificate chain: Chain RoT + IMG

Key type: ECC

Key length: p256

Advanced

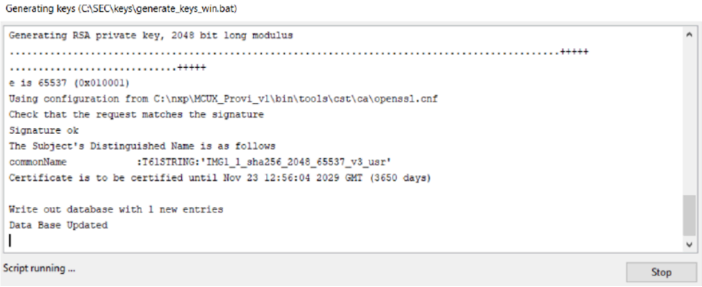
Number of RoT: 4

Figure 42. 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 RTxxx consists of three parts: preamble, revocation, and number.
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 key chain.
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.

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:\SEC\keys\generate_keys_win.bat)

```
Generating RSA private key, 2048 bit long modulus
..............................................................................................+++++
............................+++++
e is 65537 (0x010001)
Using configuration from C:\nxp\MCUX_Provi_v1\bin\tools\cst\ca\openssl.cnf
Check that the request matches the signature
Signature ok
The Subject's Distinguished Name is as follows
commonName            :T61STRING:'IMG1_1_sha256_2048_65537_v3_usr'
Certificate is to be certified until Nov 23 12:56:04 2029 GMT (3650 days)

Write out database with 1 new entries
Data Base Updated
```

Figure 43. Generate Keys - Progress

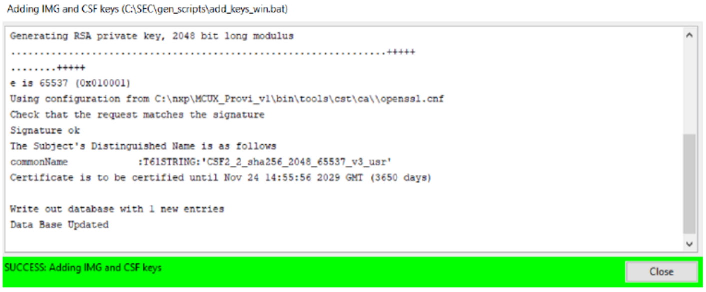
5.4.2 Add keys

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 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 Section 5.4.1

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



Adding IMG and CSF keys (C:\SEC\gen_scripts\add_keys_win.bat)

```
Generating RSA private key, 2048 bit long modulus
................................................................+++++
........+++++
e is 65537 (0x010001)
Using configuration from C:\nxp\MCUX_Provi_v1\bin\tools\cst\ca\\openssl.cnf
Check that the request matches the signature
Signature ok
The Subject's Distinguished Name is as follows
commonName            :T61STRING:'CSF2_2_sha256_2048_65537_v3_usr'
Certificate is to be certified until Nov 24 14:55:56 2029 GMT (3650 days)

Write out database with 1 new entries
Data Base Updated
```

SUCCESS: Adding IMG and CSF keys

Figure 44. 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/RTxxx, BEE user keys for RT10xx, OTFAD keys data for RT116x/RT117x/RT118x/RTxxx 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 location of the **crts** and **keys** folders containing 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 Keys for trust provisioning

If Smart Card trust provisioning is selected on the toolbar, it is possible to generate a certificate signing key and a production audit log key. Only one key of each type is supported, so if the key already exists, it is replaced. The keys then can be used at the **Smart Card management** tab.

5.4.6 Debug Authentication

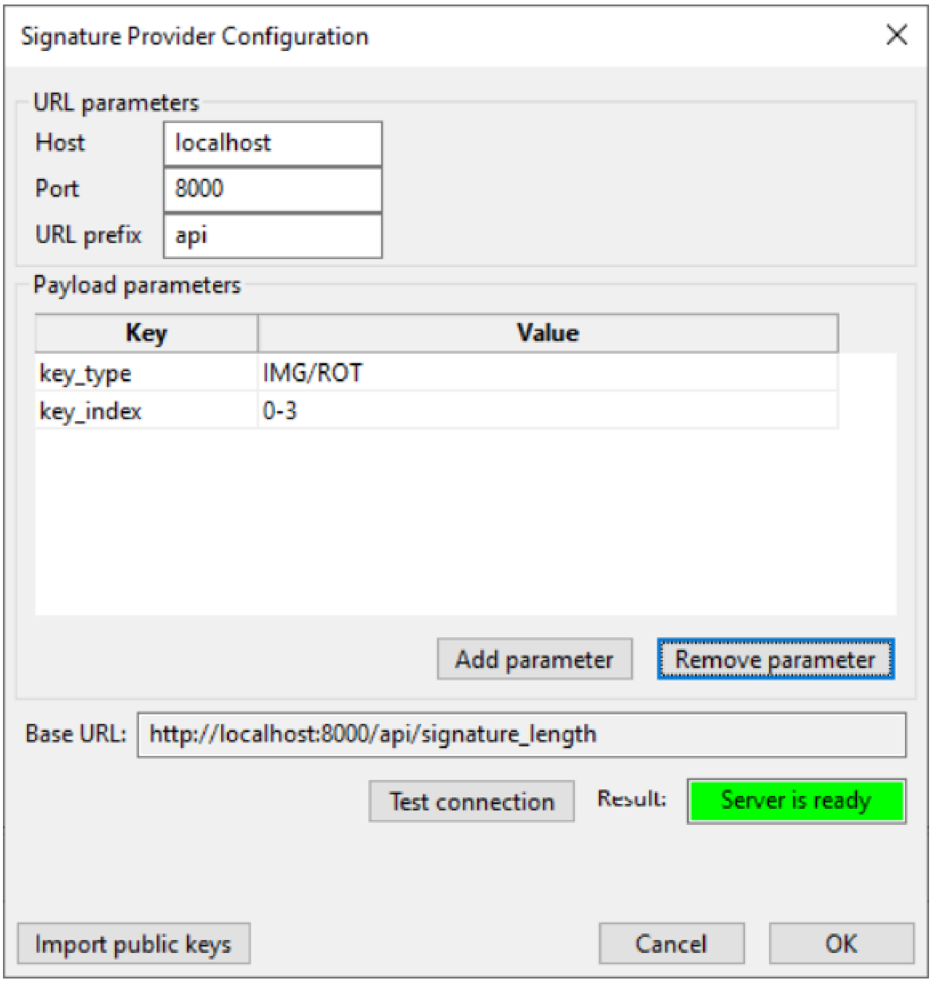
Debug Authentication (DA) buttons are displayed in PKI management only for processors that support DA. (For more information, see the attached SEC-Tool-Features.xls) The **Generate debug key** button is always enabled, **Create debug certificate request** and **Open debug port** are enabled only if the DA key exists. **Generate debug certificate** is enabled in workspace with authentication keys. The **Generate Debug Key** button opens a dialog box for creating a debug authentication key pair. **Generate Debug Certificate Request** helps you 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 DAC.

Note: *For more information, see Section 6.11*

5.4.7 Signature provider

Signature provider allows using 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 **Use sign. provider** checkbox is selected it is possible to open the configuration dialog. When enabled, the present keys will be moved to the back-up folder and the signature provider server will be used as the source of public keys and will provide signing. Configuration set in **Signature Provider Configuration** dialog will be used in configuration files used for build (cert_block, mbi_config, sb_config).



Signature Provider Configuration

URL parameters

Host: localhost

Port: 8000

URL prefix: api

Payload parameters

Key	Value
key_type	IMG/ROT
key_index	0-3

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

Result: Server is ready

Figure 45. Signature provider configuration

5.4.7.1 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 empty string if there is not any.

5.4.7.2 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, value is decimal number in the range 0-3 set by selecting key from build tab ROT1 -> 0, ROT2 -> 1, IMG1_1 -> 0, IMG2_1 -> 1 and so on.

5.4.7.3 Buttons and Base URL

Remove parameter removes a parameter from the table, remove selected parameter, if no parameter is selected, remove the last one.

Add parameter adds an empty line below the selected param or at the end of the table.

Base URL displays a URL created from required parameters.

Test connection sends a request for signature length to test whether the server is up and running.

Import public keys 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_v?.json.

5.4.7.4 Signature provider server API

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

5.4.7.4.1 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
}
```

Image signing with IMG1_1 key:

```
{
  "data": " hex string of image block to be signed",
  "key_type": "IMG"
  "key_index": 0
}
```

5.4.7.4.2 signature_length

signature_length returns the signature length in bytes.

5.4.7.4.3 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's one. For environments where such validation is not needed, the implementation can return simply "true”.

5.4.7.4.4 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/schema/signature_provider_key_tree_schema_v?.json. Generally, 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.

5.4.7.5 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 Section 6.12.

5.5 Smart Card management

This view is displayed only for processors that support trust provisioning. (For more information, see the attached SEC-Tool-Features.xls) It is only enabled if the Smart Card on the toolbar is enabled.

This view allows to:

- Configure a Smart Card
- Create a manufacturing package
- Verify the audit log package received from the factory and extract certificates

Note: *For more information, see Section 6.10*

5.5.1 Smart Card configuration

This section allows applying OEM assets into the Smart Card. The following items can be configured:

USER KEK	A symmetric key for the OEM custom key store. This value can be specified or a random button can be used to generate a value.
Device identity	The checkbox enables the generation of certificates with device identity. If selected, make sure that Device Identify Configuration contains all necessary data and select the existing certificate signing key. For more information, see Section 6.10.7.
Certificate signing key	This value is only used for device identity certificates, so the option is grayed out unless the **Device identity** checkbox is selected.
Production audit log key	Either select the existing PEM key or generate a new key in the **PKI management** view.
Production limit	A number of processors, that can be provisioned with the Smart Card.

Additionally, the following items are used for the Smart Card configuration (this is processor-specific):

SBKEK	This value is specified in the **Build image** view.
Life cycle	This value is specified in the toolbar.
PFR pages	The CMPA and CFPA pages binary configuration is created when the build and the file paths are displayed in the **Write image** view.

dev_hsm_provisioning.sb3 A device provisioning secure binary file generated during the build that contains CMPA and CFPA pages.

Prepare Smart Card The button for the Smart Card configuration. After you click the button, it is possible to create either a sealed or non-sealed Smart Card. The process of Smart Card configuration is described in the Section 6.10 section.

5.5.2 Manufacturing package

The manufacturing package contains all the data needed to start trust provisioning using the Smart Card in the factory.

No configuration is required, the input data for the manufacturing package are displayed for information only.

The process of creation and import of the Manufacturing package is described in the Trust provisioning workflow section.

5.5.3 Factory audit logs

This section allows verifying audit logs package received from the factory and optionally extracting device certificates.

The following items can be configured:

Extract certificates Enable checkbox if the device certificates must be extracted from the log. If the checkbox is enabled, it is also possible to specify:

- **Type of the certificate** Select the type of certificates to be extracted.
- **Device certificate encoding** Select the format of the device certificate to be extracted.
- **New or empty target directory** New or empty directory, where to store the extracted certificates.

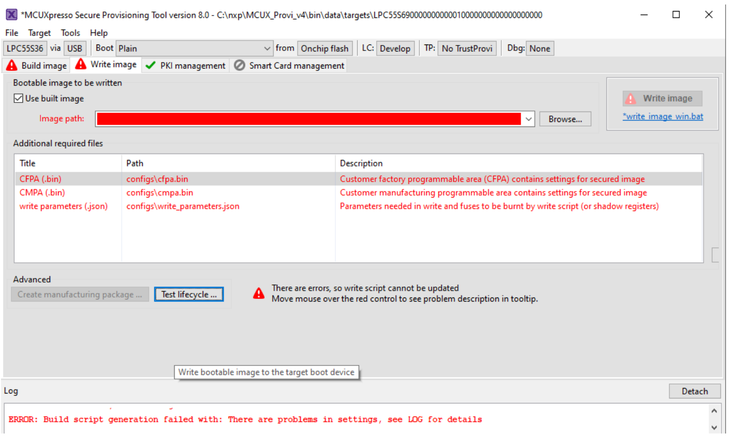
Verify audit logs … The button to start verification of the audit logs package. The package file selection must be done as the first step of the process. For the verification, the corresponding production audit log key must be selected in the Smart Card configuration section. The process of verification of audit logs is described in the Trust provisioning workflow section.

5.6 Log

The lower part of the user interface is occupied by the extendable detachable **Log** view. The view logs information about the actions performed, including errors.

The log can be detached by the **Detached** button in the upper-right corner and attached back by closing the **Detached log** view.

The information is stored in the *<workspace>\log.txt* file. The contents of this file are rotated once a threshold is reached.



*MCUXpresso Secure Provisioning Tool version 8.0 - C:\nxp\MCUX_Provi_v4\bin\data\targets\LPC55S69000000000000100000000000000000

LPC55S36 via USB | Boot: Plain | from: Onchip flash | LC: Develop | TP: No TrustProvi | Dbg: None

Bootable image to be written

☑ Use built image

Image path:

*write_image_win.bat

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.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

There are errors, so write script cannot be updated
Move mouse over the red control to see problem description in tooltip.

Write bootable image to the target boot device

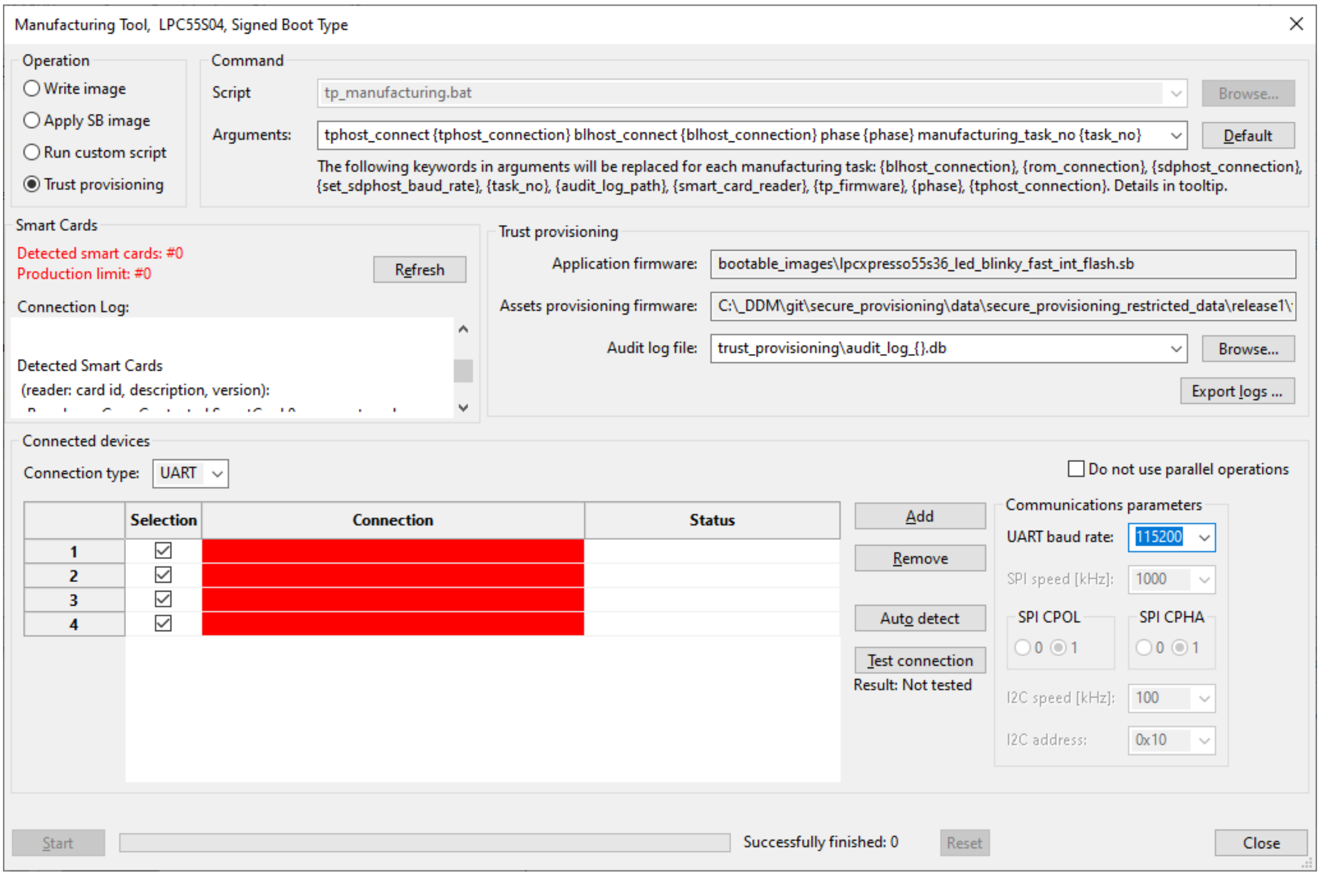
Log

```
ERROR: Build script generation failed with: There are problems in settings, see LOG for details
```

Figure 46. Log view with errors

5.7 Manufacturing Tool

Use **Manufacturing Tool** to configure provisioning on 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 are also **Smart Card** and **Trust provisioning** areas.



Manufacturing Tool, LPC55S04, Signed Boot Type

Operation

○ Write image
○ Apply SB image
○ Run custom script
● Trust provisioning

Command

Script: tp_manufacturing.bat

Arguments: tphost_connect {tphost_connection} blhost_connect {blhost_connection} phase {phase} manufacturing_task_no {task_no}

The following keywords in arguments will be replaced for each manufacturing task: {blhost_connection}, {rom_connection}, {sdphost_connection}, {set_sdphost_baud_rate}, {task_no}, {audit_log_path}, {smart_card_reader}, {tp_firmware}, {phase}, {tphost_connection}. Details in tooltip.

Smart Cards

Detected smart cards: #0
Production limit: #0

Connection Log:

Detected Smart Cards
(reader: card id, description, version):

Trust provisioning

Application firmware: bootable_images\lpcxpresso55s36_led_blinky_fast_int_flash.sb

Assets provisioning firmware: C:_DDM\git\secure_provisioning\data\secure_provisioning_restricted_data\release1\

Audit log file: trust_provisioning\audit_log_{}.db

Connected devices

Connection type: UART

□ Do not use parallel operations

	Selection	Connection	Status
1	☑		
2	☑		
3	☑		
4	☑		

Result: Not tested

Communications parameters

UART baud rate: 115200

SPI speed [kHz]: 1000

SPI CPOL: ○ 0 ● 1

SPI CPHA: ○ 0 ● 1

I2C speed [kHz]: 100

I2C address: 0x10

Successfully finished: 0

Figure 47. 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 LPC55Sxx and RTxxx devices, 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 the script is a modified SEC write script accepting SEC parameters (especially connection parameters). Exit status 0 is considered a success.
Trust provisioning	Performs the trust provisioning operation using the selected Smart Card. See chapter Trust provisioning workflow for details.

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	Interactive 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.

Smart Cards

Information about Smart Cards used for the trust provisioning operation. The following items are displayed:

Detected Smart Cards	Number of detected cards
Production limit	Total limit of all detected cards, number of processors that can be provisioned
Connection log	Detailed information about the Smart Card connection
Refresh	The button to update the information

Trust provisioning

Configuration of the trust provisioning operation.

Application and Assert provisioning firmwares

These file paths are listed for information only.

Audit log

It is possible to specify a path for the audit log. The log file must be assigned to the Smart Card, so the file name must contain the {} suffix that will be replaced by the Smart Card ID.

Export logs

Allow exporting audit into the audit log package. The package can later be verified.

Connection type

Allow selecting a communication interface with the processors.

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.

- To automatically detect all devices connected to the host for a selected connection type, use the **Autodetect** button. It is the recommended usage. For more information about the USB path, see section 6.4.1 USB Path.
- To manually add a device, use the **Add** button.
- To remove a device, use the **Remove** button or the checkbox in the **Selection** column of the table. If the device is de-selected, the next autodetection keeps the device de-selected.
- To manually modify the device path, change the entry in the **Connection** column.
- To view the log containing information about device connection, click the entry in the **Status** column.

To test (ping) selected devices, click the **Test connection** button. 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 operation.

Load KeyStore (Apply SB image for RTxxx)

Load KeyStore from external flash before uploading the SB file.

Communication parameters

Communication parameters contain additional operation options.

Baud rate

Use the drop-down list to select the baud rate of the operation. For a detailed description, see Section 5.1.6.

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.

5.7.1 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 connection. The USB path is used to identify the USB device for USB, SPI, or I2C communication. It differs depending on the operating system. The description of the USB path format can be found in SPSDK documentation.

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

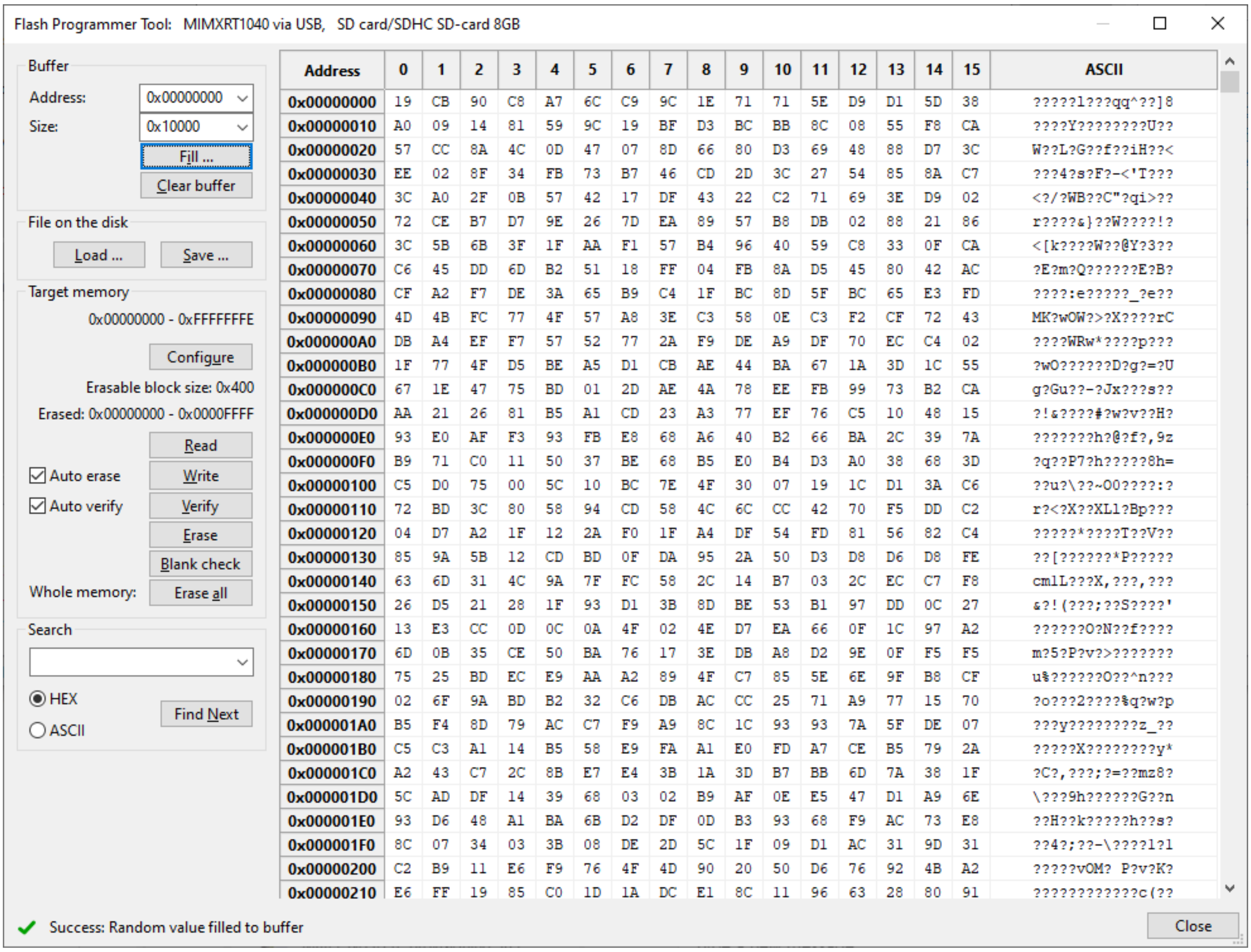
1. On some operating systems, the USB path changes after each device reset.
2. RT10xx processors in ISP mode have different VID and PID compared to the VID and PID of the flash loader application.

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 VID and PID of 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. To find the corresponding USB path for each processor, this operation cannot be executed in parallel. 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.*

5.8 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 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.



Flash Programmer Tool: MIMXRT1040 via USB, SD card/SDHC SD-card 8GB

Buffer

Address: 0x00000000

Size: 0x10000

Target memory

0x00000000 - 0xFFFFFFFE

Erasable block size: 0x400

Erased: 0x00000000 - 0x0000FFFF

☑ Auto erase

☑ Auto verify

Whole memory: Erase all

Search

● HEX

○ ASCII

Address	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	ASCII
0x00000000	19	CB	90	C8	A7	6C	C9	9C	1E	71	71	5E	D9	D1	5D	38	?????l???qq^??]8
0x00000010	A0	09	14	81	59	9C	19	BF	D3	BC	BB	8C	08	55	F8	CA	????Y????????U??
0x00000020	57	CC	8A	4C	0D	47	07	8D	66	80	D3	69	48	88	D7	3C	W??L?G??f??iH??<
0x00000030	EE	02	8F	34	FB	73	B7	46	CD	2D	3C	27	54	85	8A	C7	???4?s?F?-<'T???
0x00000040	3C	A0	2F	0B	57	42	17	DF	43	22	C2	71	69	3E	D9	02	<?/?WB??C"?qi>??
0x00000050	72	CE	B7	D7	9E	26	7D	EA	89	57	B8	DB	02	88	21	86	r????&}??W????!?
0x00000060	3C	5B	6B	3F	1F	AA	F1	57	B4	96	40	59	C8	33	0F	CA	<[k????W??@Y?3??
0x00000070	C6	45	DD	6D	B2	51	18	FF	04	FB	8A	D5	45	80	42	AC	?E?m?Q??????E?B?
0x00000080	CF	A2	F7	DE	3A	65	B9	C4	1F	BC	8D	5F	BC	65	E3	FD	????:e?????_?e??
0x00000090	4D	4B	FC	77	4F	57	A8	3E	C3	58	0E	C3	F2	CF	72	43	MK?wOW?>?X????rC
0x000000A0	DB	A4	EF	F7	57	52	77	2A	F9	DE	A9	DF	70	EC	C4	02	????WRw*????p???
0x000000B0	1F	77	4F	D5	BE	A5	D1	CB	AE	44	BA	67	1A	3D	1C	55	?wO??????D?g?=?U
0x000000C0	67	1E	47	75	BD	01	2D	AE	4A	78	EE	FB	99	73	B2	CA	g?Gu??-?Jx???s??
0x000000D0	AA	21	26	81	B5	A1	CD	23	A3	77	EF	76	C5	10	48	15	?!&????#?w?v??H?
0x000000E0	93	E0	AF	F3	93	FB	E8	68	A6	40	B2	66	BA	2C	39	7A	???????h?@?f?,9z
0x000000F0	B9	71	C0	11	50	37	BE	68	B5	E0	B4	D3	A0	38	68	3D	?q??P7?h?????8h=
0x00000100	C5	D0	75	00	5C	10	BC	7E	4F	30	07	19	1C	D1	3A	C6	??u?\??~O0????:?
0x00000110	72	BD	3C	80	58	94	CD	58	4C	6C	CC	42	70	F5	DD	C2	r?<?X??XLl?Bp???
0x00000120	04	D7	A2	1F	12	2A	F0	1F	A4	DF	54	FD	81	56	82	C4	?????*????T??V??
0x00000130	85	9A	5B	12	CD	BD	0F	DA	95	2A	50	D3	D8	D6	D8	FE	??[??????*P?????
0x00000140	63	6D	31	4C	9A	7F	FC	58	2C	14	B7	03	2C	EC	C7	F8	cm1L???X,???,???
0x00000150	26	D5	21	28	1F	93	D1	3B	8D	BE	53	B1	97	DD	0C	27	&?!(???;??S????'
0x00000160	13	E3	CC	0D	0C	0A	4F	02	4E	D7	EA	66	0F	1C	97	A2	??????O?N??f????
0x00000170	6D	0B	35	CE	50	BA	76	17	3E	DB	A8	D2	9E	0F	F5	F5	m?5?P?v?>???????
0x00000180	75	25	BD	EC	E9	AA	A2	89	4F	C7	85	5E	6E	9F	B8	CF	u%???????O??^n???
0x00000190	02	6F	9A	BD	B2	32	C6	DB	AC	CC	25	71	A9	77	15	70	?o???2????%q?w?p
0x000001A0	B5	F4	8D	79	AC	C7	F9	A9	8C	1C	93	93	7A	5F	DE	07	???y????????z_??
0x000001B0	C5	C3	A1	14	B5	58	E9	FA	A1	E0	FD	A7	CE	B5	79	2A	?????X????????y*
0x000001C0	A2	43	C7	2C	8B	E7	E4	3B	1A	3D	B7	BB	6D	7A	38	1F	?C?,???;?=??mz8?
0x000001D0	5C	AD	DF	14	39	68	03	02	B9	AF	0E	E5	47	D1	A9	6E	\????9h??????G??n
0x000001E0	93	D6	48	A1	BA	6B	D2	DF	0D	B3	93	68	F9	AC	73	E8	??H??k?????h??s?
0x000001F0	8C	07	34	03	3B	08	DE	2D	5C	1F	09	D1	AC	31	9D	31	??4?;??-\????1?1
0x00000200	C2	B9	11	E6	F9	76	4F	4D	90	20	50	D6	76	92	4B	A2	?????vOM? P?v?K?
0x00000210	E6	FF	19	85	C0	1D	1A	DC	E1	8C	11	96	63	28	80	91	????????????c(??

✓ Success: Random value filled to buffer

Figure 48. Flash programmer tool

Buffer

Controls the range of the displayed space on the right.

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/written**

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, does not change the memory state

All information provided in this document is subject to legal disclaimers.

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.

Save … Saves buffer or portion of buffer into a file. 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.

Configure Tries to configure the connected memory. It executes the same memory configuration as in the Section 5.1.7. 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 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.

5.9 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 2.x and 3.x. 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 SPSDK/nxpimage command-line tool, so the SB editor produces a configuration YAML file that is used as input for nxpimage.

SB editor contains the following main UI parts:

Top buttons bar	The bar that allows to 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
Status and bottom buttons	The buttons that allow creating manufacturing package and start manufacturing tool. The **OK/Cancel** buttons allow closing the SB editor with or without saving the configuration into the workspace

5.9.1 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.

If the property value is a file path, it is possible to open a browse dialog by double-clicking the property name.

5.9.2 Commands view

The view allows specifying the sequence of the commands that will be stored in the SB file and must 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. These commands are visible in the GUI only to simplify the edition 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 configuration.

The commands page contains the following panels:

The command sequence It is the current command sequence. 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:

- ⬅ **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.

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.

5.9.3 $ Variables

To allow reusage 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.

5.9.4 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 Section 5.1.4.1 .

5.9.5 To Manufacturing button

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

6 Workflow

This chapter takes you through the steps you must take to successfully boot up your 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 the image is executed on an NXP evaluation board.

This chapter addresses image preparation for the following toolchains:

- MCUXpresso IDE 11
- Keil MDK 5 µVision
- IAR Embedded Workbench 8

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 SEC
- Prepare asymmetric keys
- Build a plain image in the selected toolchain
- Build a bootable image by SEC
- 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

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, because it offers a simple check whether the resulting application is working – LED flashes with a 1 sec period.

- **Downloading MCUXpresso SDK package for MCUXpresso IDE**
  1. Visit http://mcuxpresso.nxp.com.
  2. Select your board.
  3. Build 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 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 SDK build available on MCUXpresso SDK Dashboard, 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.
     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.

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 binary SDK examples for NXP evaluation boards provided in the installation layout. For details, refer to the installation sub-folder **<SEC>/bin/data/targets/{processor}/source_images**.

6.1.4 Setting up Secure Provisioning Tool

1. Start MCUXpresso 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 device series and the 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 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. Set **Boot type** to any secured boot type, for example, **Signed** or **Authenticated**.

2. Select the **PKI management** view.

3. Ensure it does not already contain keys.

4. Click **Generate keys**.

5. In the **Generate keys** dialog, confirm default settings and click **Generate**.

6. Set **Boot type** back to **Unsigned** mode. It is recommended to start with the **Unsigned** mode and verify the unsigned image works on the board. Once unsigned mode is working, you can continue to secured mode and the generated keys will be used.

NOTE: The generated keys are generated in **crts/** and **keys/** subfolders in the workspace. It is recommended to back up generated keys before they are burned into fuses in the processor.

6.2 RT10xx/RT116x/RT117x device workflow

This section describes the RT10xx/RT116x/RT117x device workflow in detail.

6.2.1 Preparing source image

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 (FlexSPI NOR, FlexSPI 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 (FlexSPI NOR, FlexSPI NAND, or SEMC NAND) and during boot will be copied into SDRAM and executed from there.

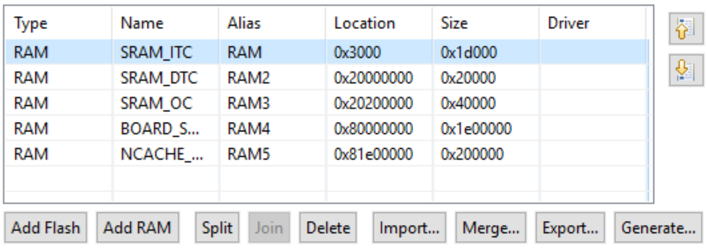
6.2.1.1 Image running from external NOR flash

- **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 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 **Source executable image** by SEC.
- **Keil MDK 5**
  1. In the toolbar, select **iled_blinky flexspi_nor_debug** target.
  2. 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 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. 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.*

6.2.1.2 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 49. SRAM_ITC

3. Move **SRAM_ITC** to the 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** 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*.

6.2.1.3 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 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*.

6.2.2 Connecting the board

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 3 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 SEC.
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 MCUXpresso Secure Provisioning Tool, select **Boot memory: sdhc_sd_card/SDHC SD card 8 GB** in the **Toolbar**.

Table 3. DIP Switches: Boot mode selection for EVKs

Boot mode/ Device	Serial bootloader (ISP mode)	Flex-SPI NOR (QSPI, Hyper Flash)	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							
RT1020-EVK				SW8: 0110			
RT1024-EVK							
RT1040-EVK	SW4: 0001	SW4: 0010	SW4: 0010 or 0110 SW2: 1000	SW4: 1010			
RT1050-EVKB	SW7: 0001	SW7: 0110	SW7: 0010 or 0110 SW5: 1000	SW7: 1010	N/A	N/A	N/A
RT1060-EVK		SW7: 0010					
RT1064-EVK							
RT1160-EVK	SW1: 0001 SW2: 00000 00000	SW1: 0010 SW2: 00000 00000	SW1: 0010 SW2: 0100000000	SW1: 0010 SW2: 00000 01000	SW1: 0010 SW2: 00000 00100	SW1: 0010 SW2: 00000 10000	N/A
RT1170-EVKB							

6.2.3 Booting images

This section describes the building and writing of bootable images.

You can use several combinations of used memories:

Table 4. Booting image

Memory where the image is executed	Memory where the image is written	DCD/XMCD needed	XIP
External NOR flash[1]	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

[1] For RT116x/7x devices, two FlexSPI instances are supported. There is the `FLEXSPI_INSTANCE` fuse `(BOOT_CFG2[3])/GPIO` boot pin which determines which FlexSPI instance to use. Set a corresponding GPIO boot pin to use Instance 2 without burning the fuse.

Note:

- ***Memory, where the image is executed*** *- Explained in Section 6.2.1.*
- ***Memory, where the image is written*** *- Configured as **Boot memory** in SEC.*

6.2.3.1 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 you have selected the **Unsigned boot type** in the **Toolbar**.
2. Switch to the **Build image** view.
3. Select the image built in Section 6.2.1 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 Section 6.2.4.*
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. Make sure that the **Use built image** check-box is selected.
4. Click the **Write image** button.

If the write operation was successful, switch boot mode (see Table 3) and reset the board.

6.2.3.2 Booting authenticated (HAB) image

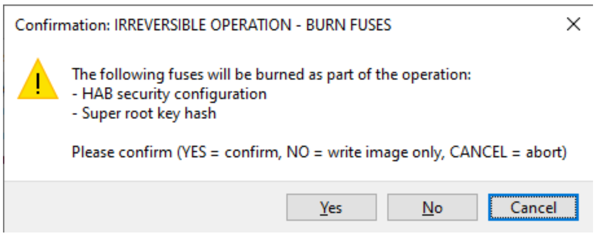
This section describes the building and writing of an authenticated image. If you want to use an encrypted image, you can skip this step.

1. In the **Toolbar** set the **Boot type** to **Authenticated (HAB)**.
2. In the **Build image** view, use the image from Section 6.2.1 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 3 for more information.
2. Make sure that the **Use built image** checkbox is selected.
3. Click the **Write image** button.
4. 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 50. Burn fuses

If the write operation was successful, switch boot mode (see Table 3) and reset the board.

6.2.3.3 Booting encrypted (HAB) image

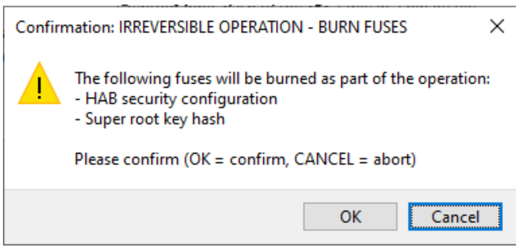
This section describes the building and writing of an encrypted image. This image will be decrypted into RAM during booting operation, so an XIP image cannot be used.

To build the image, do the following:

1. In the **Toolbar** set the **Boot type** to **Encrypted (HAB)**.
2. As a **Source executable image**, use the image from Section 6.2.1 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 3 for more information.
3. Make sure that the **Use built image** checkbox is selected.
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
- Super root key hash

Please confirm (OK = confirm, CANCEL = abort)

Figure 51. Burn fuses

If the write operation was successful, switch boot mode (see Table 3) 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)*** *boot type is not available.*

6.2.3.4 Booting XIP encrypted image (BEE OTPMK) authenticated (RT10xx)

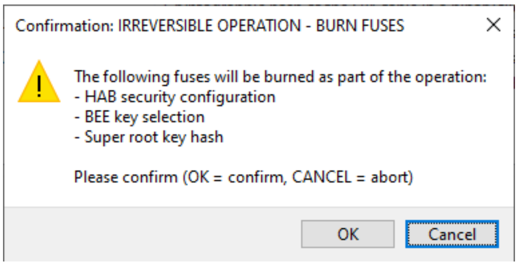
This section describes the building and writing of an XIP encrypted image using the OTP master key. An authenticated image is built and then encrypted on-the-fly during the write operation. The source image for the encrypted XIP with the BEE feature must be an XIP image.

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 Section 6.2.1 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 3 for more information.
3. Make sure that the **Use built image** checkbox is selected.
4. Enable XIP encryption by setting a corresponding GPIO pin (see Table 3 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:
- HAB security configuration
- BEE key selection
- Super root key hash

Please confirm (OK = confirm, CANCEL = abort)

Figure 52. Burn fuses

If the write operation was successful, switch boot mode (see Table 3) and reset the board.

Note: *Step 5 can be replaced by setting the EncryptedXIP fuse in the OTP configuration.*

6.2.3.5 Booting XIP encrypted image (BEE user keys) unsigned (RT10xx)

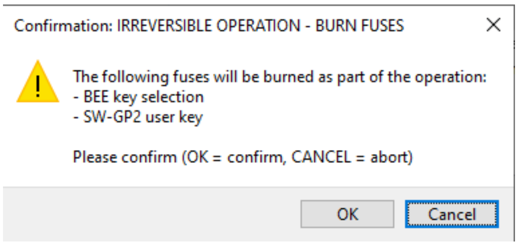
This section describes the building and writing of an XIP encrypted image using user keys. The image itself is built in two steps. First, the unsigned bootable image is built and then this unsigned image is encrypted for use with enabled encrypted XIP. The source image for the encrypted XIP with the 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 Section 6.2.1 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 3 for more information.
3. Ensure that **Use built image** checkbox is selected.
4. Enable XIP encryption by setting a corresponding GPIO pin (see Table 3 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:
- BEE key selection
- SW-GP2 user key

Please confirm (OK = confirm, CANCEL = abort)

Figure 53. Burn fuses

If the write operation was successful, switch boot mode (see Table 3) and reset the board.

Note: *Step 5 can be replaced by setting the EncryptedXIP fuse in the OTP configuration.*

6.2.3.6 Booting XIP encrypted image (BEE user keys) authenticated (RT10xx)

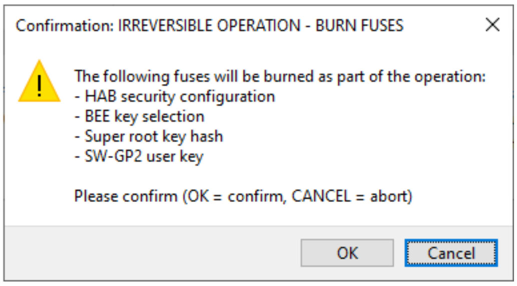
This section describes the building and writing of an XIP encrypted image using user keys. The image itself is built in two steps. First, the authenticated bootable image is built and then this authenticated image is encrypted for use with enabled encrypted XIP. The source image for the encrypted XIP 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 Section 6.2.1 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 3 for more information.
3. Ensure that **Use built image** checkbox is selected.
4. Enable XIP encryption by setting a corresponding GPIO pin (see Table 3 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.



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 54. Burn fuses

If the write operation was successful, switch boot mode (see Table 3) and reset the board.

Note: *Step 5 can be replaced by setting the EncryptedXIP fuse in the OTP configuration.*

6.2.3.7 Booting XIP encrypted image (OTFAD OTPMK) authenticated (RT10xx)

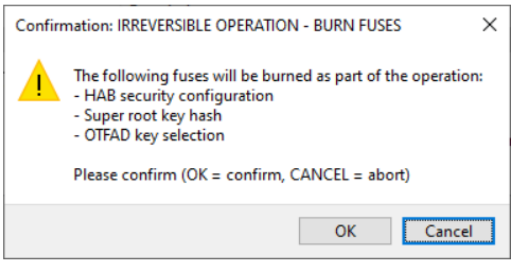
This section describes building and writing of an XIP encrypted image using the OTP master key. The authenticated image is built and then encrypted on-the-fly during the write operation. The source image for the encrypted XIP with the OTFAD feature must be an XIP image.

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 Section 7.2.1 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 6 for more information.
3. Make sure that the **Use built image** checkbox is selected.
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
- Super root key hash
- OTFAD key selection

Please confirm (OK = confirm, CANCEL = abort)

Figure 55. Burn fuses

If the write operation was successful, switch boot mode (see Table 6) and reset the board.

6.2.3.8 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:

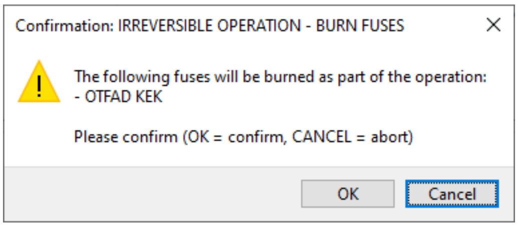
1. In the **Toolbar** set **Boot Type** to **Encrypted**, **(OTFAD) unsigned** for RT116x/7x or **XIP encrypted (OTFAD user keys) unsigned** for RT10xx.
2. As **Source executable image**, use the image external NOR flash from Preparing source image as a **Source executable image**.
3. Click **OTFAD encryption**/ **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 3 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. Ensure that **Use built image** checkbox is selected.
5. Enable XIP encryption (RT116x/7x) by setting a corresponding GPIO pin (see Table 3 for more information).
6. Click the **Write image** button.
7. 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 56. Burn fuses

If the write operation was successful, switch boot mode (see Table 3) and reset the board.

Note: *Step 6 (RT116x/7x) can be replaced by setting the `ENCRYPT_XIP_EN` fuse in the OTP configuration.*

6.2.3.9 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.

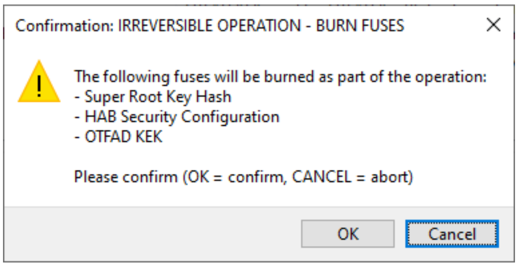
To build the image, do the following:

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 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, *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 3 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. Ensure that **Use built image** checkbox is selected.
5. Enable XIP encryption (RT116x/7x) by setting a corresponding GPIO pin (see Table 3 for more information).
6. Click the **Write image** button.
7. 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 57. Burn fuses

If the write operation was successful, switch boot mode (see Table 3) and reset the board.

Note: *Step 6 (RT116x/7x) can be replaced by setting the* `ENCRYPT_XIP_EN` *fuse in OTP configuration.*

6.2.3.10 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.

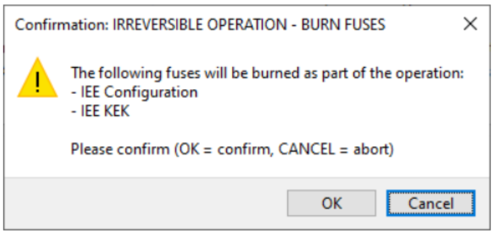
To build the image, do the following:

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 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 6 for more information.
3. Reset the board. It is required for successful key store enrollment.
4. Ensure that the **Use built image** checkbox is selected.
5. Enable XIP encryption by setting a corresponding GPIO pin (see Table 6 for more information).
6. Click the **Write image** button.
7. 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 58. Burn fuses

If the write operation was successful, switch boot mode (see Table 6) and reset the board.

Note: *Step 5 can be replaced by setting the `ENCRYPT_XIP_EN` fuse in the OTP configuration.*

6.2.3.11 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.

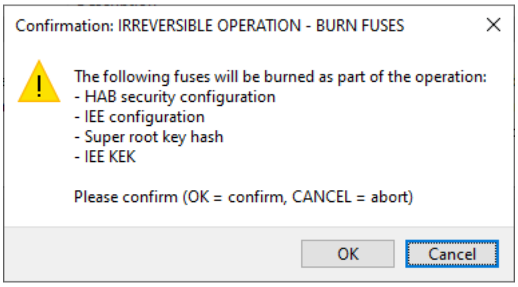
To build the image, do the following:

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 as a **Source executable image.**
3. Ensure you have keys generated in the **PKI management** view. For more information, see Section 5.4.
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 6 for more information.
3. Reset the board. It is required for successful key store enrollment.
4. Ensure that the **Use built image** checkbox is selected.
5. Enable XIP encryption by setting a corresponding GPIO pin (see Table 6 for more information).
6. Click the **Write image** button.
7. 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 59. Burn fuses

If the write operation was successful, switch boot mode (see Table 6) and reset the board.

Note: *Step 6 can be replaced by setting the `ENCRYPT_XIP_EN` fuse in the OTP configuration.*

6.2.4 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 **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 **Menu bar > 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.*

6.3 RT118x device workflow

This section describes the RT118x device workflow in detail.

6.3.1 Preparing source image

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 or in external flash (FlexSPI NOR, FlexSPI NAND) and will be copied into RAM and executed from there during the boot. The following RAM types are supported:
  – internal RAM
  – SDRAM
  – HyperRAM
  The source image preparation is similar to RT116x/7x, see Section 6.2.1. Some RT118x specifics:
  – Select an example for core cm33 and then set the target to Cortex-M33.

– Step regarding **XIP_BOOT_HEADER_ENABLE** is not optional. The XIP boot header must be disabled because the bootable image cannot be used as the input on the build tab for RT118x.

6.3.2 Connecting the board

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

- MIMXRT1180-EVK
- MIMXRT1180A-EVK
- MIMXRT1180-144

1. See Table 5 for instructions on how to set boot mode using DIP switches.
2. Make sure you have J1 set to 3-4 to power the board from USB (from UART on RT1180-144).
3. Connect to the J33 (J53 on RT1180-144) 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 Section 6.1.4.
5. Make sure that the **Boot device** in the **Boot Memory Configuration** matches the FlexSPI NOR flash used on the EVK board.
6. Set the connection to **USB** (**UART** on MIMXRT1180-144) 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 8GB USDHC1** in the **Boot Memory Configuration**.

- 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**.

Table 5. DIP Switches: Boot mode selection for EVKs

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)

6.3.3 Booting images

This section describes the building and writing of bootable images.

You can use several combinations of used memories:

Table 6. Booting image

Memory where the image is executed	Boot memory: Memory where the image is written	XMCD needed	XIP
External NOR flash1	External NOR flash	No	Yes
Internal RAM	External NOR or NAND flash	No	No

Table 6. Booting image*...continued*

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 Section 6.4.1.*
- *Memory, where the image is written is configured as Boot Memory in SEC.*

6.3.3.1 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 you have selected the **Unsigned boot type** in the **Toolbar**.
2. Switch to the **Build image** view.
3. Select the image built in Section 6.3.1 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: bin/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. Make sure that the **Use built image** checkbox is selected.
4. Click the **Write image** button.
5. If the write operation was successful, switch boot mode (see Table 5) and reset the board.

6.3.3.2 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 Section 5.4.1 If you want to use an encrypted

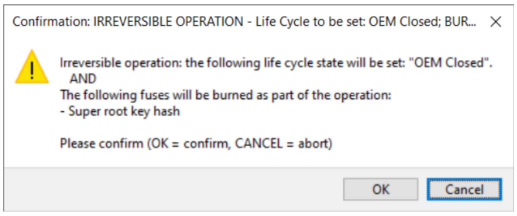
image, you can skip this step.

First, build a bootable image:

1. In the **Toolbar** set **Boot type** to **Signed**.
2. In the **Build image** view, use the image from Section 6.3.1 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 5 for more information.
2. Make sure that the **Use built image** checkbox is selected.
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:
- Super root key hash

Please confirm (OK = confirm, CANCEL = abort)

Figure 60. Burn fuses

5. If the write operation was successful, switch boot mode (see Table 5) and reset the board.

6.3.3.3 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 Section 5.4.1.

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 Section 6.3.1 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 3 for more information.
3. Make sure that the **Use built image** checkbox is selected.
4. Click the **Write image**. button.
5. In the following window, confirm to write fuses:
   - **OK** - Continue writing the image and burning fuses.
   - **Cancel** - Abort writing the image and burning fuses.
6. If the write operation was successful, switch boot mode (see Table 5) 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.*

6.3.3.4 Booting OTFAD encrypted image

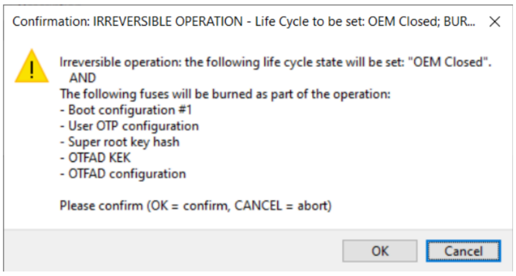
This section describes the building and writing of an OTFAD encrypted image. The image itself is built in two steps. First, the unsigned/signed AHAB image is built and then this AHAB image is encrypted for use with OTFAD. The source image for the OTFAD feature must be an XIP image. The Keys generated in the **PKI management** view are needed in this step. For more information about generating keys, see Section 5.4.1.

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 Section 6.3.1 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 5 for more information.
3. Ensure that the **Use built image** checkbox is selected.
4. Click the **Write image** button.
5. 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 61. Burn fuses

6. If the write operation was successful, switch boot mode (see Table 5) and reset the board.

6.3.3.5 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 Section 5.4.1.

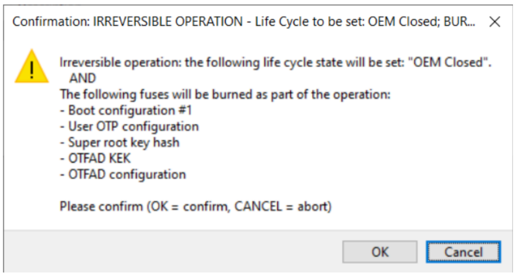
To build the image, do the following:

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 Section 6.3.1 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 5 for more information.
3. Ensure that the **Use built image** checkbox is selected.
4. Click the **Write image** button.
5. 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 62. Burn fuses

6. If the write operation was successful, switch boot mode (see Table 5) and reset the board.

6.3.3.6 Booting multicore images

This section describes how to build and write an image for multiple cores (Cortex M33 and Cortex M7).

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 Section 6.3.3.5.*

6.3.3.7 Life cycle

The default life cycle, which should be used for development, is **OEM Open**. Before you deploy the application, set **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

6.3.3.8 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.

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

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

6.4.1 Preparing source image

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**
  It is the only supported boot memory for the LPC55Sxx/LPC55xx device family. The image is executed directly from internal flash memory. (XIP)

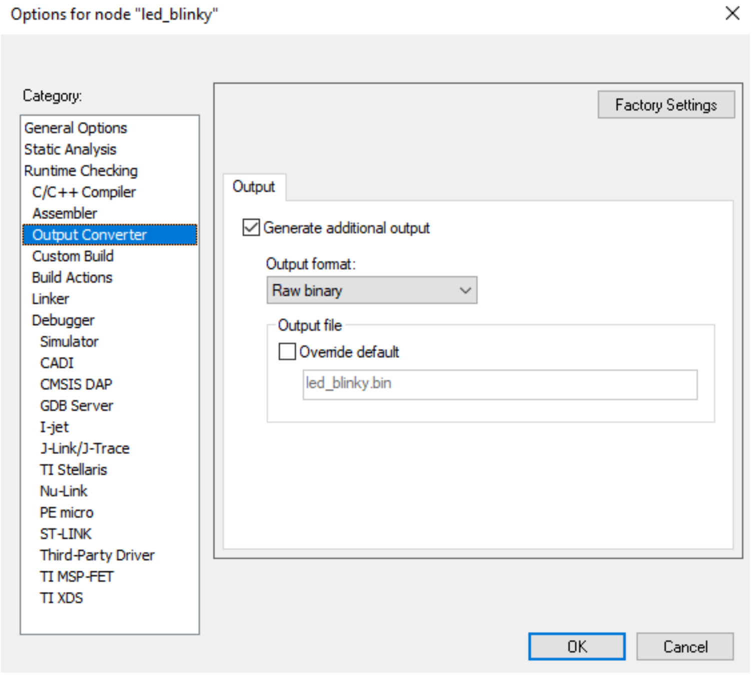
6.4.1.1 Image running from internal flash

- **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 63. IAR options

2. 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

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

- LPCexpresso55S69
- LPCexpresso55S66

- LPCexpresso55S28
- LPCexpresso55S26
- LPCexpresso55S16
- LPCexpresso55S14
- LPCexpresso55S06
- LPCexpresso55S04

It is assumed SEC tool is already running with a workspace created for an LPC device. For more information, see Section 6.1.4.

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 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

This section describes the building and writing of images.

6.4.3.1 Security levels

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** or **Plain with CRC**.
2. As a **Source executable image**, use the image from Section 6.4.1 as a **Source executable image**.
3. In the case of a binary image, set the start address to *0x0*.
4. Click the **Build image** button.
5. 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. Make sure that the **Use built image** checkbox is selected.

4. Click the **Write image** button.

If the write operation was successful, reset the board.

6.4.3.3 Booting 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 Section 5.4.1.

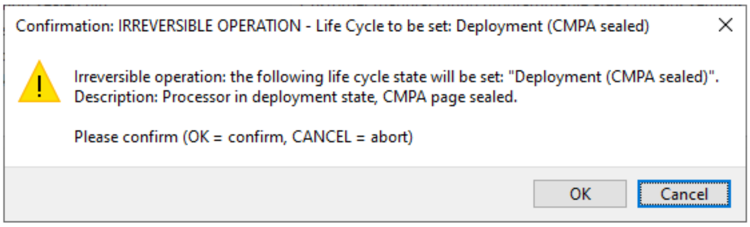
Note: *The keys are also used for* ***Encrypted (PRINCE) Plain and with CRC*** *boot type because the bootable image is updated using SB capsule, which must be signed.*

1. In the **Toolbar**, set **Boot type** to **Signed**, **Encrypted (PRINCE) Plain, Encrypted (PRINCE) with CRC**, or **Encrypted (PRINCE) and signed**.
2. In the **Build image** view, use the image from Section 6.4.1 as a source executable image.
3. For **Authentication key**, select any key chain, 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. Click the **Build image** button.
6. Check that the bootable image was built successfully.

To write the image, do the following:

1. Click the **Write image** view.
2. Make sure that the board is connected and the ISP mode is enabled (See Section 6.4.2.)
3. Make sure that the **Use built image** checkbox is selected.
4. 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)

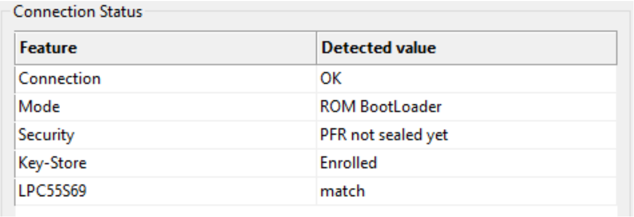
Figure 64. Confirm write

If the write operation was successful, reset the board.

6.4.3.4 PUF KeyStore

SEC initializes KeyStore on LPC55Sxx devices only once in the device life cycle. After that, the changes in the SBKEK key are not supported.

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 65. 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.

6.4.3.4.1 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.

6.4.3.4.2 How to update CFPA page (example for LPC55S69)

1. Increment 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.
   Default version value since SEC 3.0:
   ```
   "VERSION": "0x0000_0002",
   ```
   Default version value in SEC 2.x:
   ```
   "VERSION": "0x0200_0000",
   ```
2. Run the following commands to update CFPA page into 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
   ```

6.5 LPC55(S)3x device workflow

This chapter describes workflow for LPC55(S)3x processors.

6.5.1 Preparing source image

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

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

Table 7. Boot mode selection for EVK

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

1. Select ISP boot mode, see Table 7.
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 Section 6.1.4.
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.

6.5.3 Booting images

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

6.5.3.1 Booting plain image or plain with CRC image

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 you have selected the **Plain** or **Plain with CRC** boot type in the toolbar.
2. Switch to the **Build image** view.
3. Select image built in Section 6.5.1 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. Make sure that the **Use built image** check-box is selected.
4. Click the **Write image** button.

If the write operation is successful, switch boot mode (see Table 7) and reset the board.

6.5.3.2 Booting signed image

This section describes building and writing a signed image.

Build a bootable image:

1. Select **Signed** boot type in the toolbar.

2. Switch to the **Build image** view.
3. Select image built in Section 6.5.1 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 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 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 processor:

1. Make sure the processor is in ISP mode.
2. Switch to the **Write image** view.
3. Make sure that the **Use built image** check-box is selected.
4. 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.

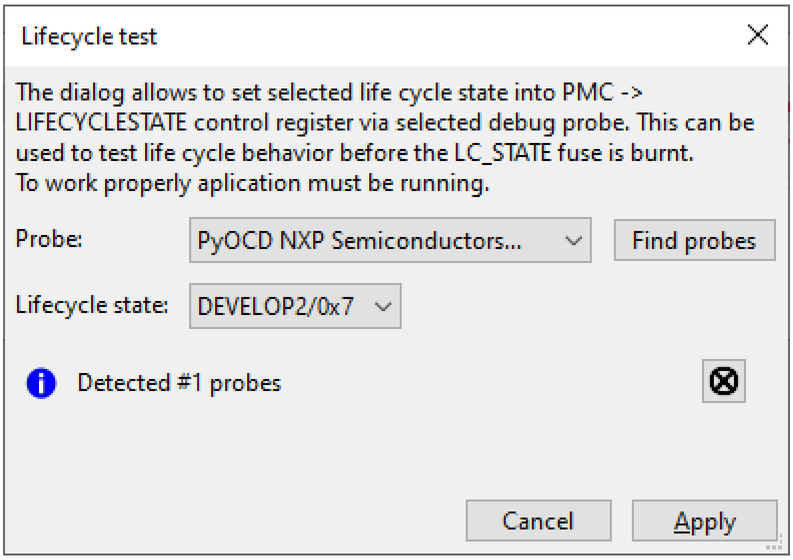
Image encryption is performed when the image is written to the target memory. The encrypted region is configured in the SB file. The decrypted regions are configured in CMPA page, so make sure these two are aligned.

6.5.3.4 Test life cycle

To test processor behavior in an 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. Supported probes are Jlink, pyOCD, and PEmicro.

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 debug probe.
7. On the write tab click the **Test lifecycle** button and in the displayed dialog set the required life cycle state.
8. Click **Apply** to move processor into the selected life cycle. Now it is possible to test the processor behavior.



Lifecycle test

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

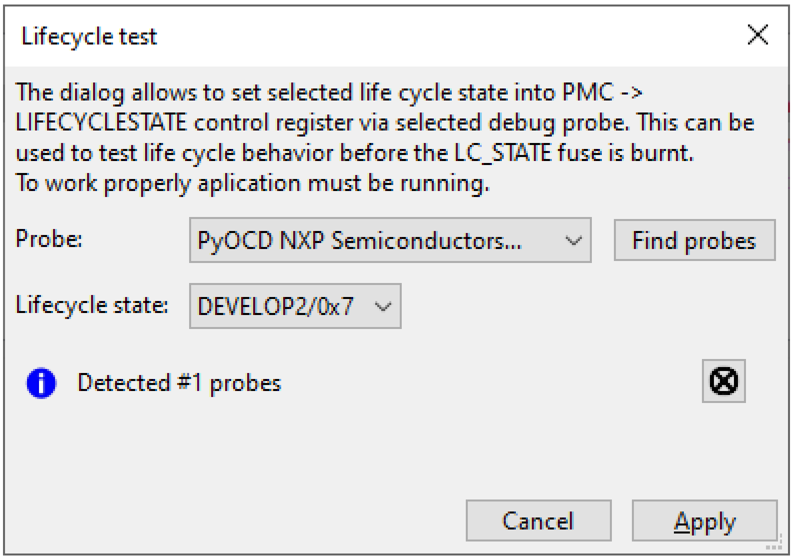
Figure 66. Lifecycle test

6.5.3.5 Life cycle

The default life cycle, which should be used for development, is **Develop**. Before you deploy the application, set “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.



Lifecycle test

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 67. Lifecycle test

6.6 RTxxx device workflow

This section describes RTxxx device workflow in detail.

6.6.1 Preparing source image

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.

Following boot types are available for RTxxx processors:

• Image running in external flash: The XIP image (eXecuted In Place)

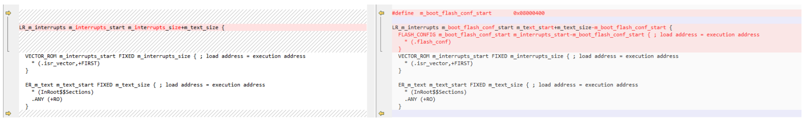
• Image running in internal RAM: The image is copied from FLASH, SD card, or eMMC to RAM before the execution

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 you can easily modify it to blink all the time.

6.6.1.1 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. 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 input for the bootable image in SEC.
- **Keil MDK**
  1. In the **Toolbar**, select **gpio_output_flash_debug** target.
  2. In **Project > Options > "*C/C++*"** disable define symbol **BOOT_HEADER_ENABLE=0 (set to 0)**.
  3. In **Project > Options > Output** select **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 68. Keil MDK workaround

  5. 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. 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*.

6.6.1.2 Image running in internal RAM

- **MCUXpresso IDE**
  1. 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 **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. 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*.

6.6.2 Connecting the board

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

- MIMXRT595-EVK
- MIMXRT685-EVK

Table 8. RTxxx EVK boot configuration

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	RT595 SW7[1:3] 011	RT595 SW7[1:3] 110
MIMXRT685-EVK	SW5[1:3]: 100	SW5[1:3]: 101	RT685 SW5[1:3] 011	RT685 SW5[1:3] 110

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 you have started SEC with a new workspace. For more information, see Section 6.1.4.
4. Set the connection to **USB** and test the board connection.
5. **Booting from SD card**:
6. For booting from an SD card, do the following:

1. Insert SDHC card into the board in MCUXpresso 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.**

6.6.3 Booting images

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

6.6.3.1 Booting a plain/plain with CRC image

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** or **Plain with CRC** in **Boot type**.

2. Switch to the **Build image** view.

3. Select image build in Section 6.6.1 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 Section 6.6.2.

2. Switch to the **Write image** view.

3. Make sure that the **Use built image** checkbox is selected.

4. **Reset the board**. Note that 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.

5. Click the **Write image** button.

If the write operation was successful, switch boot mode to **FlexSPI boot** (see Table 8) and reset the board.

6.6.3.2 Booting signed image using shadow registers

This section describes the building and writing of an authenticated image.

1. In the **Toolbar** set **Boot type** to **Signed**.
2. In the **Build image** view, use the image from Preparing source image as a **Source executable image**.
3. Ensure you have keys generated in the **PKI management** view. For more information, see Section 5.4.
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 Section 6.6.3.5 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 Section 6.6.2.

3. Make sure that the **Use built image** checkbox is selected.
4. **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.
5. 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.

6.6.3.3 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 Section 6.2.1 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 Full Security 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 Section 6.6.2.
3. Make sure that the **Use built image** checkbox is selected.
4. **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.
5. 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.*

6.6.3.4 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 previously 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 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 Section 5.2.4.*

6.6.3.5 Securing the processor

To enable full security on RTxxx 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**.

Note that once the *DCFG_CC_SOCU* fuses are set, it is no longer possible to modify security configuration parameters, and no changes in 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.

6.6.3.6 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: 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 **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.

Note: *Provisioning firmware is distributed in a restricted data package, see Preferences for details.*

Write

Write operation is the same as secure boot type using 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, 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 Section 5.3.1

6.7 KW45xx/K32W1xx device workflow

This chapter describes workflow for KW45xx/K32W1xx processors.

6.7.1 Preparing source image

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

- **Image running from an internal flash**

It is the only supported boot memory for the KW45xx/K32W1xx device family. The image is executed directly from 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

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

- KW45B41Z-EVK
- K32W148-EVK

It is assumed SEC tool is already running with a workspace created for an KW45xx/K32W1xx device. For more information, see Section 6.1.4

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.

6.7.3 Booting images

This section describes building and writing of bootable images. For KW45xx/K32W1xx, SEC tool supports XIP images only.

6.7.3.1 Booting plain image or plain with 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. Dual image boot is supported only for secure boot types.

First, build a bootable image:

1. Make sure you have selected the **Plain** or **Plain with CRC** boot type in the toolbar.

2. Switch to the **Build image** view.

3. Select an image built in Section 6.10.1 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. Make sure that the **Use built image** check-box is selected.

4. Click the **Write image** button.

If the write operation was successful, reset the board.

6.7.3.2 Booting signed image

This section describes building and writing a signed image.

Build a bootable image:

1. Select **Signed** boot type in the toolbar.
2. Switch to the **Build image** view.
3. Select an image built in Section 6.10.1 as a **Source executable image**.
4. Ensure you have the keys on PKI management. Evaluation boards KW45B41Z-EVK and K32W148-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 “bin\data\targets\<processor>\evk_keys” (see Section 5.4.4). 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 **Development** 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. Make sure that the **Use built image** check-box is selected.
4. Keep **OEM Open** in the life cycle.
5. Click the **Write image** button.

6.7.3.3 Booting PRINCE encrypted image

Encrypted plain 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.*

6.7.3.4 Life cycle

The default life cycle, which should be used for development, is **OEM Open**. Before you deploy the application, set **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 ROMCFG page. The application still can be updated via SB file.

6.8 RW61x device workflow

This chapter describes workflow for RW61x processors.

6.8.1 Preparing source image

The only available boot memory for the RT61x processor is external flash.

To create the source image for the build in SEC tool, disable the boot header, set define 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**).

The image shall be at address 0x8001000 (default in MCUXpresso SDK examples).

6.8.2 Connecting the board

This section contains information about configuring the evaluation boards RD-RW612-BGA or RD-RW61x-QFP and connecting it to SEC.

Table 9. TBD

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

1. Select ISP boot mode, see Table 9

2. Connect the J7 port to your PC with a USB cable.

3. Ensure SEC runs with a workspace created for the chosen device. For more information, see Section 6.1.4.

4. Go to **main menu – Target – Connection** and select UART and test the connection.

6.8.3 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 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 to use 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).

6.8.4 Booting images

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

6.8.4.1 Booting plain image or plain with 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 you have selected the **Plain** or **Plain with CRC** boot type in the toolbar.

2. Switch to the **Build image** view.

3. Select an image built in Section 6.4.1 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.

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. Make sure that the **Use built image** check-box is selected.

4. Click the **Write image** button.

5. If the write operation was successful, reset the board.

- the image boots automatically if shadow registers are selected
- otherwise switch boot mode (see Table 9) and reset the board to boot the image.

6.8.4.2 Booting signed image

This section describes building and writing a signed image.

Build a bootable image:

1. Select **Signed** boot type in the toolbar.
2. Switch to the **Build image** view.
3. Select an image built in Section 6.4.1 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 there is no error in OTP configuration. For ECC p384 key length you will need to configure BOOT_CFG3 fuse.
7. Make sure 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 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. Make sure that the **Use built image** check-box is selected.
4. 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 life cycle without shadow registers, the fuses will be burnt irreversibly and SB3 capsule will be used to write the application into the flash.

6.8.4.3 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 Section 5.1.7 how to convert Flex-SPI NOR simplified configuration to full FCB configuration.

For clock limitations when using encrypted images, see documentation for the target processor.

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.

6.8.4.4 Life cycle

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.

Table 10. TBD

	Shadow regs life cycle	Develop life cycle	In-Field life cycle
Plain 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
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

6.8.4.5 Device HSM and CUST_MK_SK

The CUST_MK_SK is a customer key for SB file encryption/decryption and this key can be installed into 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, the key is already installed or not).

6.9 MCX Nx4x/A14x/A15x device workflow

This chapter describes workflow for Nx4x/A14x/A15x processors.

6.9.1 Preparing source image

- 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.

6.9.2 Connecting the board

This section contains information about configuring the evaluation boards FRDM-MCXN947, MCX-N9XX-EVK, MCX-N5XX-EVK, or FRDM-MCXA153 and connecting it to SEC.

Table 11. TBD

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		
MCX-N5XX-EVK	SW3/JP49		
FRDM-MCXA153	SW2/JP8	N/A	By default

1. Select ISP boot mode, see Table

2. Connect the J5 (J15 onFRDM-MCXA153) port to your PC with a USB cable.

3. Ensure SEC runs with a workspace created for the chosen device. For more information, see Section 6.1.4.

4. Go to **main menu – Target – Connection** and select UART and test the connection.

6.9.3 Booting images

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.

6.9.3.1 Booting plain image or plain with 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 you have selected the **Plain** or **Plain with CRC** boot type in the toolbar.

2. Switch to the **Build image** view.

3. Select an image built in Section 6.4.1 as a **Source executable image**.

4. If there is a binary image, set the start address to 0x00000000 (for external flash 0x80001000).

5. Configure BOOT_CFG0 fuse (see the previous paragraph).

6. If needed, open **Dual image boot** and configure.

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. Make sure that the **Use built image** check-box is selected.

4. Click the **Write image** button.

5. If the write operation is successful, remove the ISP jumper and reset the board to boot the image.

6.9.3.2 Booting signed image

This section describes building and writing a signed image.

Build a bootable image:

1. Select **Signed** boot type in the toolbar.
2. Switch to the **Build image** view.
3. Select an image built in Section 6.4.1 as a **Source executable image**.
4. Generate keys on PKI tab.
5. Back on the **Build image** view select any key, 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 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. If no board is connected 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. Make sure that the **Use built image** check-box is selected.
4. Click the **Write image** button.

6.9.3.3 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 CMPA page, so make sure these two are aligned.

6.9.3.4 Life cycle

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).

When switching the lifecycle, it is recommended to burn ROTKH and PRINCE or IPED region settings into fuses. ROTKH fuses are set by the tool automatically, but PRINCE or IPED setting is up to the user to be set.

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 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.

6.9.3.5 Test life cycle

MCX Nx4x 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 **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 **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*

6.10 Smart Card trust provisioning workflow

Trust provisioning allows OEMs (Original Equipment Manufacturers) to contract manufacturing of the product by a third-party company including firmware provisioning without access of the third-party company to the OEM confidential assets (such as application firmware and keys).

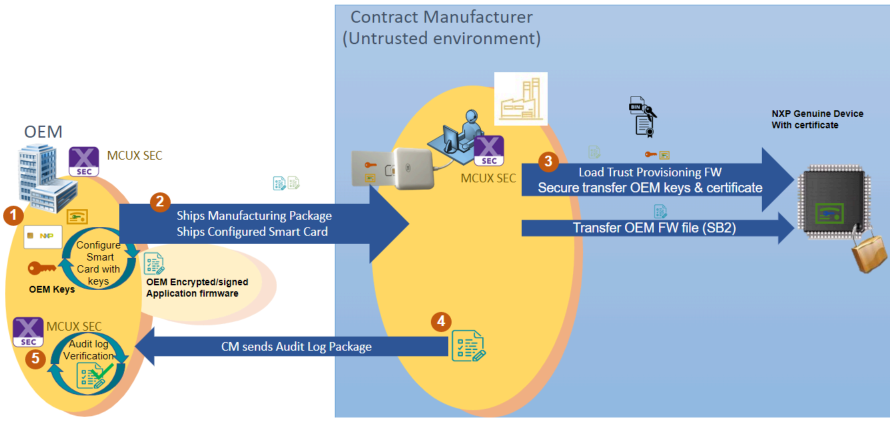

Contract Manufacturer
(Untrusted environment)
OEM
MCUX SEC
1
2
Ships Manufacturing Package
Ships Configured Smart Card
Configure Smart Card with keys
OEM Keys
OEM Encrypted/signed Application firmware
MCUX SEC
3
Load Trust Provisioning FW
Secure transfer OEM keys & certificate
Transfer OEM FW file (SB2)
NXP Genuine Device
With certificate
4
CM sends Audit Log Package
MCUX SEC
5
Audit log Verification

Figure 69. High-level trust provisioning workflow

High-level workflow description:

1. OEM develops application firmware (sb file capsule). OEM generates keys and configures Smart Card with the keys and other assets. OEM creates a manufacturing package with the application firmware.
2. OEM sends Smart Card and manufacturing package to the factory.
3. In the factory, the firmware is provisioned into flash, the record is added into the audit log for each device.
4. The factory sends the audit log to OEM.
5. OEM checks the validity of the audit log and the number of devices being manufactured, optionally extracts certificates, and uploads them to the cloud.

Features:

- Application firmware is encrypted using OEM keys.
- Smart Card is used to secure the OEM keys.
- Smart Card contains the production counter, so OEM can control the number of products.
- Simple manufacturing dialog allows producing of several devices in parallel.
- Audit log in SQL-lite DB format.
- Optionally, there can be generated up to 4 device identity certificates for each product.
- Supported processors can be found in the attached SEC-Tool-Features.xls
- For details, see a demo video (webinar) at nxp.com (https://www.nxp.com/video/enabling-smart-card-trust-provisioning-on-lpc5500:SMARTCARDTP).

6.10.1 Preparation of OEM-specific inputs

To start trust provisioning, you need:

- Smart Card – Please contact NXP local representatives for Smart Cards. Starting from SEC tool version 7, Smart Card with applet version 1.2 or above is required.
- Processor that is supported for trust provisioning.
- Application firmware – bootable application built with a secure boot type (authenticated or encrypted). The application must be built in the **Build image** view. SBKEK must be specified.
- The secured life cycle is configured in the toolbar.
  Note: *For development purposes, it is possible to use the development life cycle, but only on the LPC55S36 processor*
- Valid PFR Configuration in the **Build image** view. The CFPA version must be higher than the CFPA version in the current processor (it is recommended to add at least 10), otherwise the CFPA page cannot be updated.

- The audit log key and optionally also the certificate signing key. These keys can be provided externally or created on the **PKI management** view (See Section 5.4).
- For the LPC55S3x device, an HSM provisioning SB3 file that contains CFPA and CMPA; the format of the file is different for a Smart Card, so it must be re-built after the card is selected.
- Asset provisioning firmware.
  The firmware is distributed within the SEC tool; however, for some processors, part of the firmware is delivered as restricted data. For details, see Section 5.1.3.
- Optionally, you can also configure the device identity for IoT (Internet of Thing) devices. This feature is described in Section 6.10.7.

6.10.2 Configuration of Smart Card at OEM

After you have all inputs, enable the Smart Card trust provisioning type on the toolbar. Click the **Detect** button to find the Smart Card. Click the **Test connection** button to ensure that the Smart Card is recognized by the tool and communication can be established.

After you enable the Smart Card, the **Smart Card management** view is enabled. You can generate trust provisioning keys (certificate signing key and production audit log key) in the **PKI management** view. Review and configure all items at the Smart Card configuration (For more information, see Section 5.5). Then hit the **Prepare Smart Card** button.

Note: *You cannot generate the certificate signing key and audit log keys until you enable the Smart Card.*

For development purposes, you do not need to seal the Smart Card immediately. The unsealed Smart Card can be reconfigured. It is recommended to test the Smart Card before it is sealed.

SBKEK and PFR Configuration are also stored on the Smart Card, reconfigure the Smart Card if there are changes in any of them.

6.10.3 Testing the OEM configuration

This describes how to verify trust provisioning on the OEM side. This step is optional; however, it is a good practice to ensure trust provisioning works with your configuration.

- Use a new empty processor.
  - If you have previously used the processor, ensure that no fuses are burnt and the processor is still in the development life cycle. Use the **Clear CMPA** button in the **PFR configuration** dialog to clear the CMPA page. Ensure the CFPA version being provisioned has a higher version than the CFPA page currently used in the processor. Erase the whole flash.
- Open the manufacturing dialog from the **Main menu > Tools > Manufacturing Tool** and confirm to re-generate the trust provisioning script. Select the **trust provisioning** operation, use default arguments for the trust provisioning command.
- Ensure that the **Smart Card** is properly selected and click the **Refresh** button to check that communication with the Smart Card can be established and the production limit on the Card is non-zero. Ensure the application firmware and provisioning firmware and audit log are selected. For some processors, there might be two different versions of provisioning firmware depending on the silicon revision. In this case, provisioning firmware is not selected by default, it is auto-detected by the **Test connection** button.
- Boot the processor into the ISP mode and connect it via a serial port or USB to your computer.
- Click the **Auto detect** button to update the connected board in the Connected devices table.
- If the corresponding serial port is listed, ensure it is enabled and all other serial ports are disabled and click the **Test connection** button to ensure that the communication with the target processor can be established. Verify, that all connected devices are listed in the table. If the table contains devices, that should not be

configured, such as other serial ports with other devices, disable them. Then click the **Test connection** button to ensure that the communication with the target processor can be established.

- To start the trust provisioning operation, click the **Start** button.
  Note: *It is an irreversible operation, it can be executed only once for each processor.*
  The operation consists of three phases:
  - Load trust provisioning firmware into the processor (this operation can be executed in parallel)
  - Provisioning asserts to the processor using Smart Card (this operation can be executed only one per Smart Card)
  - Application phase: install custom application firmware into the processor (this operation can be executed in parallel)

6.10.4 Preparation of the final data for factory

On the **Smart Card management** view, go to the **Manufacturing package** section and click the **Create package** button. Select the filename of the ZIP file and save.

Manufacturing package contains:

- Asset provisioning firmware, including license and software content register (these files are processor-specific, so the attached documents may vary per device).
- Your application firmware is encrypted in the SB capsule.

Click the **Prepare Smart Card** button again and confirm sealing. Do it for all the cards before shipment. The seal status can be checked in the **Trust Provisioning** dialog with the **Test connection** function, see Section 5.1.8.

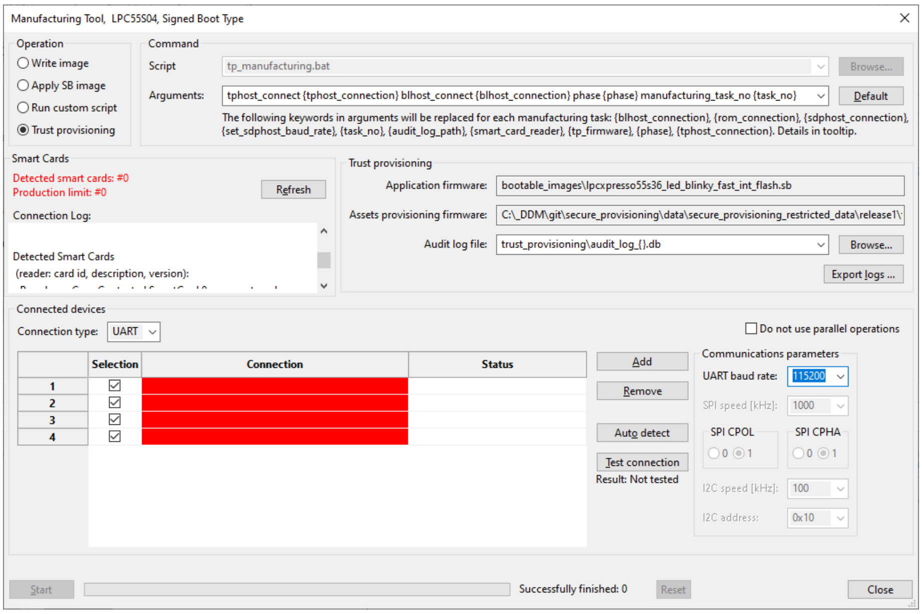
Then send the Smart Card and the manufacturing package ZIP file to the factory.

It is also recommended to back up the complete configuration including all keys into a safe location, this means, back-up workspace and all the used files outside the workspace.

6.10.5 Factory: Manufacturing process

Import manufacturing package, see Section 6.1.3.1 Manufacturing workspace for details.

When the import operation is successfully done, Manufacturing Tool is open, see the picture below. The workspace is in the **Manufacturing** mode, it shows that the Manufacturing Tool with the **trust provisioning** operation enabled only and does not allow other operations like **Building** or **Writing the image**.



Manufacturing Tool, LPC55S04, Signed Boot Type

Operation: ○ Write image ○ Apply SB image ○ Run custom script ● Trust provisioning

Command

Script: tp_manufacturing.bat

Arguments: tphost_connect {tphost_connection} blhost_connect {blhost_connection} phase {phase} manufacturing_task_no {task_no}

The following keywords in arguments will be replaced for each manufacturing task: {blhost_connection}, {rom_connection}, {sdphost_connection}, {set_sdphost_baud_rate}, {task_no}, {audit_log_path}, {smart_card_reader}, {tp_firmware}, {phase}, {tphost_connection}. Details in tooltip.

Smart Cards

Detected smart cards: #0

Production limit: #0

Connection Log:

Detected Smart Cards

(reader: card id, description, version):

Trust provisioning

Application firmware: bootable_images\lpcxpresso55s36_led_blinky_fast_int_flash.sb

Assets provisioning firmware: C:_DDM\git\secure_provisioning\data\secure_provisioning_restricted_data\release1\

Audit log file: trust_provisioning\audit_log_{}.db

Connected devices

Connection type: UART

□ Do not use parallel operations

	Selection	Connection	Status
1	☑	[redacted]	
2	☑	[redacted]	
3	☑	[redacted]	
4	☑	[redacted]	

Result: Not tested

Communications parameters

UART baud rate: 115200

SPI speed [kHz]: 1000

SPI CPOL: ○ 0 ● 1

SPI CPHA: ○ 0 ● 1

I2C speed [kHz]: 100

I2C address: 0x10

Successfully finished: 0

Figure 70. Manufacturing Tool

Use default arguments for the `trust provisioning` command.

Insert a Smart Card into your computer (if you have a built-in reader) or connect an external Smart Card reader. To find the connected Smart Cards, click the **Refresh** button. Verify, that production limit available on the Smart Cards is higher than number of connected processors.

Select the audit log file, that is created during firmware provisioning. It is not allowed to use one log file for several Smart Cards. It is recommended to use one log file per Smart Card.

For serial connection:

Adjust the baud rate, the default value is 115200; however, it was successfully tested baud rates up to 1000000.

Production steps:

1. Connect one or more processors via USB or serial line 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. The tool may contain several versions of the provisioning firmware for different silicon revisions. If the provisioning firmware is not autoselected, click the **Test connection** button provides detection of the silicon revision and assignment of the firmware.
2. To start the **trust provisioning** operation, click the **Start** button. Wait until all operations are finished.
3. Continue with step 1

The number of successfully provisioned devices is displayed on the bottom of the Manufacturing Tool window. This number is not 100% reliable, it is recommended to check remaining production limit reported for the connected Smart Cards.

After the production is done, click the **Export logs** button to export the log into the audit log package and send it to OEM.

If the trust provisioning operation fails, it depends on the status of the device, whether the next attempt can succeed. It is recommended to reset the device before the operation is repeated.

6.10.6 Audit log verification at OEM side

Audit log package contains:

- Audit log, SQL lite DB, that is used for verification and contain certificates
- LOGs from the manufacturing process, text files

To verify the audit log package, open the **Smart Card management** view and ensure that the corresponding **Production audit log key** is selected.

If the audit log contains device identities, you can optionally export certificates generated for each device. In this case, configure the corresponding parameters (see the next chapter for details).

Click the **Verify audit logs** button and select the path to the received audit log ZIP package. After the verification is completed, find the detailed results in the LOG view.

6.10.7 Device identity and cloud service provider registration

IoT applications naturally rely on the services provided by the cloud for their operation; therefore, cloud services must have the means to verify the device identity against a known trusted source. OEM can prepare cloud infrastructure by pre-registering all device unique certificates if available, or by registering a single CA against which all devices are verified upon connection. In the latter case, the infrastructure verifies that the device unique certificates have been signed by the user-specified CA and automatically registers the devices.

6.10.7.1 Configure device identity

Trust provisioning capabilities in the Secure Provisioning Tool complement the just-in-time provisioning flow. The **Device Identity Configuration** window expects an OEM CA certificate as an input in the **Issuer** view, along with the list of X.509 attributes that define what the unique device certificates must contain and where must they reside. During the manufacturing process, device-unique certificates are generated and signed in a secure way in the Smart Card by the CA. They can be installed in the desired location for use when interacting with the cloud. The certificates are generated in the X.509 format and during the device provisioning they are recorded into the audit log. When extracted from the audit log, they can be directly used within the cloud.

Some processors may support generation of the following certificate types:

- CA: see USKS usage control property in the processor Reference Manual.
- RTF: see URTF usage control property in the processor Reference Manual.

On the **Smart Card management** view, go to the **Smart Card configuration** section, enable **Device identity** and click the **Device identity configuration...** button.

- In the configuration dialog, specify the number of device identity private/public key pairs to be generated during provisioning.
- Specify the duration for generated certificates.
- In the **Certificate addresses** section, specify the addresses where the corresponding certificates will be placed in the device NVM memory of the target processor.
- In the **Certificate fields** section on the **Subject** view, you can optionally configure X.509 attributes for the device identity certificates. On the **Issuer** view, import the attributes for the device identity certificate signing key from the CA certificate or edit the attributes manually.

In the **Smart Card configuration** section, specify the certificate signing key to be used for signing the generated device certificates.

The details about the information created in the flash during trust provisioning process can be found in SPSDK documentation, see `main menu > Help > SPSDK Online Documentation.`

6.10.7.2 Extract certificates from the audit log package

During verification of the audit log, it is possible to optionally extract device certificates. The NXP identity certificates are generated in the process of verifying that provisioning happens on a genuine NXP device. Typical use cases do not require their use; for example, registration of a device with a cloud provider can be done with the device identity certificates as described in the sections above.

You can use the **Extract certificates** checkbox to enable the feature. Select whether you want to extract device certificates only or also NXP identity certificates; the format of device identity certificates is selectable, while NXP identity certificates are always in binary form. Specify target directory, it must be always selected empty or new directory.

6.10.7.3 Deploy certificates

Before provisioned devices can interact with IoT cloud services, the CA should be registered with the cloud provider. The procedure varies across vendors. Find some useful resources below:

1. Amazon Web Services IoT:
   - https://docs.aws.amazon.com/iot/latest/developerguide/iot-provision.html
   - https://docs.aws.amazon.com/iot/latest/developerguide/jit-provisioning.html
   - https://docs.aws.amazon.com/iot/latest/developerguide/register-device-cert.html
2. Microsoft IoT Hub
   - https://docs.microsoft.com/en-us/azure/iot-hub/iot-hub-x509ca-concept
   - https://docs.microsoft.com/en-us/azure/iot-hub/iot-hub-x509ca-overview

6.11 Debug authentication

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.

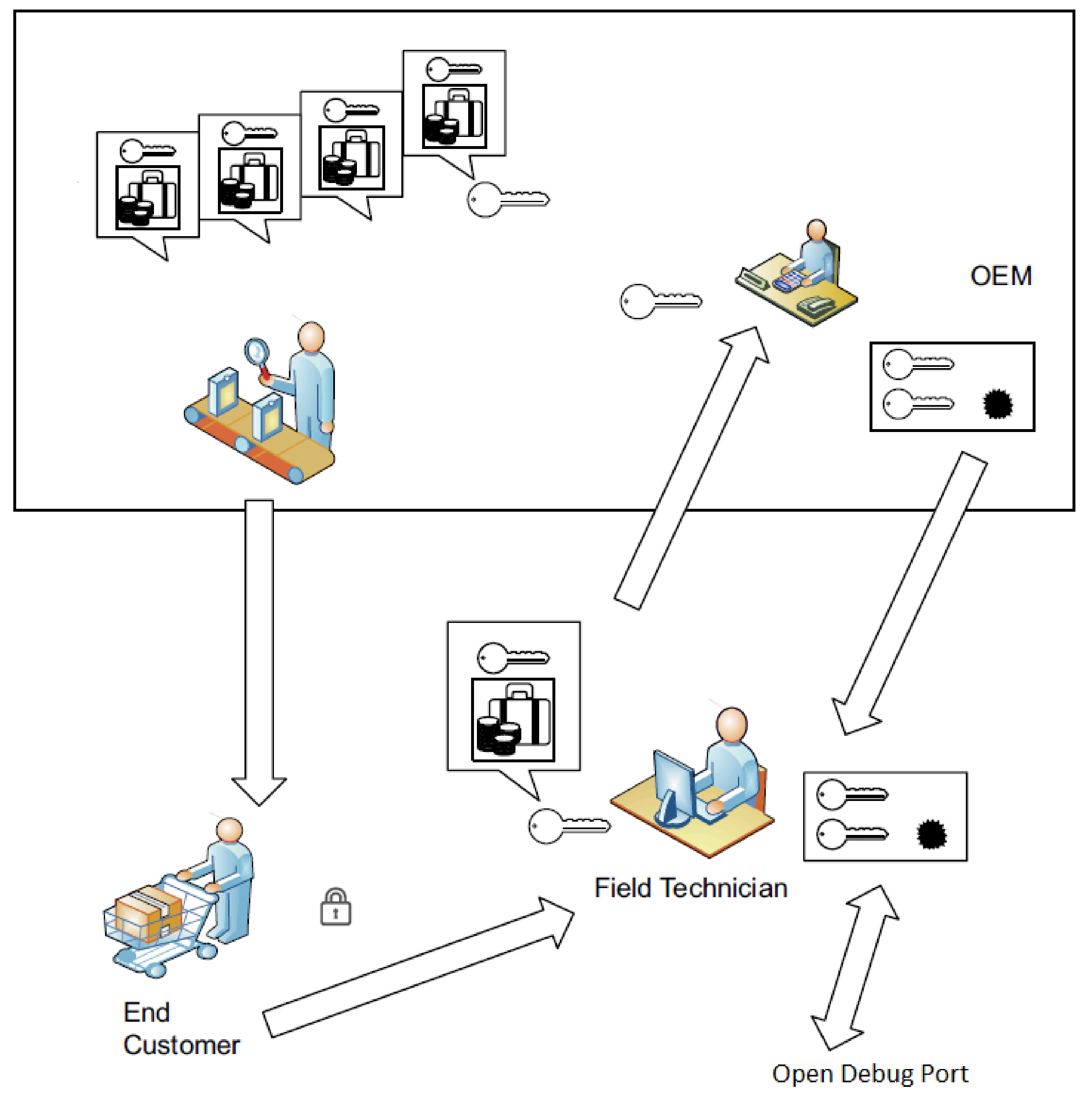


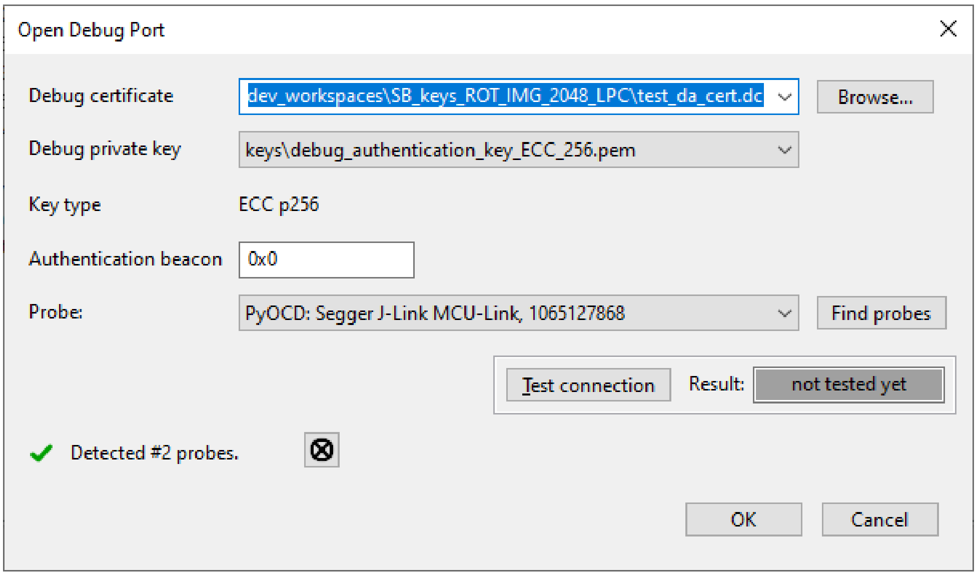
Figure 71. Debug Authentification protocol usage example

To open the debug port, do the following:

Field Technician

1. Contact OEM to acquire the key type and the length of ROT. 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. Generate a debug key, the DA key type and the length must be the same as that of the ROT key
4. **Create debug certificate request**, specify UUID to limit use of 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 device is in the development life cycle. Or via the debug probe when the device is in the advanced life cycle, on most devices CHECK_UUID must be set in the SOCU register/PFR field, for this option to work.
5. Send the certificate request to OEM
6. After OEM sends the certificate, select **Open debug port**. Connected probes are detected upon dialog display. The list of detected probes can be updated by **Find probes**. Select one of the detected probes. The authentication beacon is in no way dependent on 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, there is a script generated into workspace\debug_auth\open_debug_port.[bat|sh] and the script 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 8.11 for useful tips how to enable debug authentication.
   Note: *nxpdebugmbox CLI tool can be found in* `<installation_dir>/tools/spsdk/` *folder*



Open Debug Port

Debug certificate: dev_workspaces\SB_keys_ROT_IMG_2048_LPC\test_da_cert.dc

Debug private key: keys\debug_authentication_key_ECC_256.pem

Key type: ECC p256

Authentication beacon: 0x0

Probe: PyOCD: Segger J-Link MCU-Link, 1065127868

Result: not tested yet

✓ Detected #2 probes.

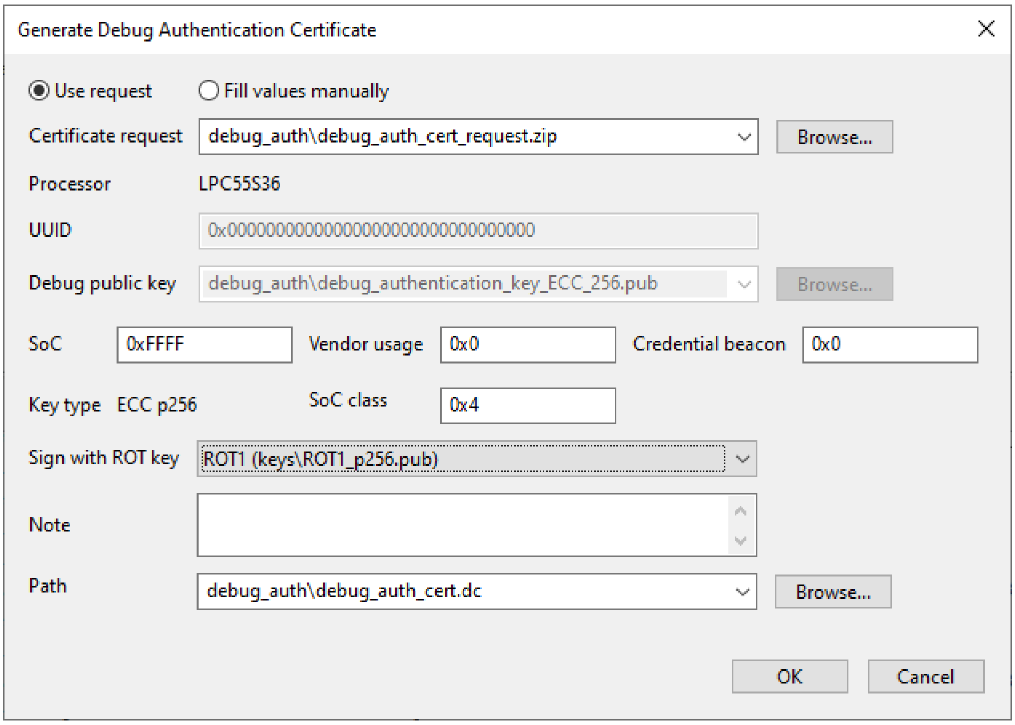
Figure 72. Dialog for opening debug port

OEM

Having the request received, click **Generate debug certificate**.

The certificate is by default generated into `<workspace>/debug_auth` folder, `debug_auth_cert.dc`. A *.zip folder 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 73).

- **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: LPC55S36

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

SoC class: 0x4

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

6.11.1 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 12. RTxxx

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 13. 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
*For debugging, authentication is still required but the domain cannot be disabled by the SoC mask in the DAC.			

Table 14. MCX Nx4x

PFR field	Everything disabled	Everything enabled*	Controlled by DA
DCFG_CC_SOCU_NS_PIN,	0xF81007EF	0xF81007EF	0xFFBF0040

Table 14. MCX Nx4x*...continued*

PFR field	Everything disabled	Everything enabled*	Controlled by DA
DCFG_CC_SOCU_PIN			
DCFG_CC_SOCU_NS_DFLT, DCFG_CC_SOCU_DFLT	0xFFBF0040	0xF81007EF	0xFFBF0040
*For debugging, authentication is still required but the domain cannot be disabled by the SoC mask in the DAC.			

Table 15. LPC55Sxx

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 16. 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
*For debugging, authentication is still required but the domain cannot be disabled by the SoC mask in the DAC.			

Table 17. KW45xx/K32W1xx

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

RT 118x 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.

6.12 Signature provider workflow

This section describes the process of setting up signature provider and building image signed by the signature provider. There are examples of signature provider server located in <install_folder>/bin/resources/signature_provider_example, one working with ROT ECC keys and other with ROT RSA keys. These examples demonstrate full implementation of the API, however in real world, it is expected the implementation will be changed by communication with HW HSM module or custom HTTPS communication to another server. Both examples of server can be used as they are to test tool behavior when using signature provider. Each server has example private keys and prepared public key response for **public_keys_certs** endpoint. 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.

Figure 1 display 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 SEC tool. Prepared examples implement only the Custom signature provider HTTP server, example server is doing all the operation that should be done by HSM or external signature provider. It is up to user to implement complete solution.

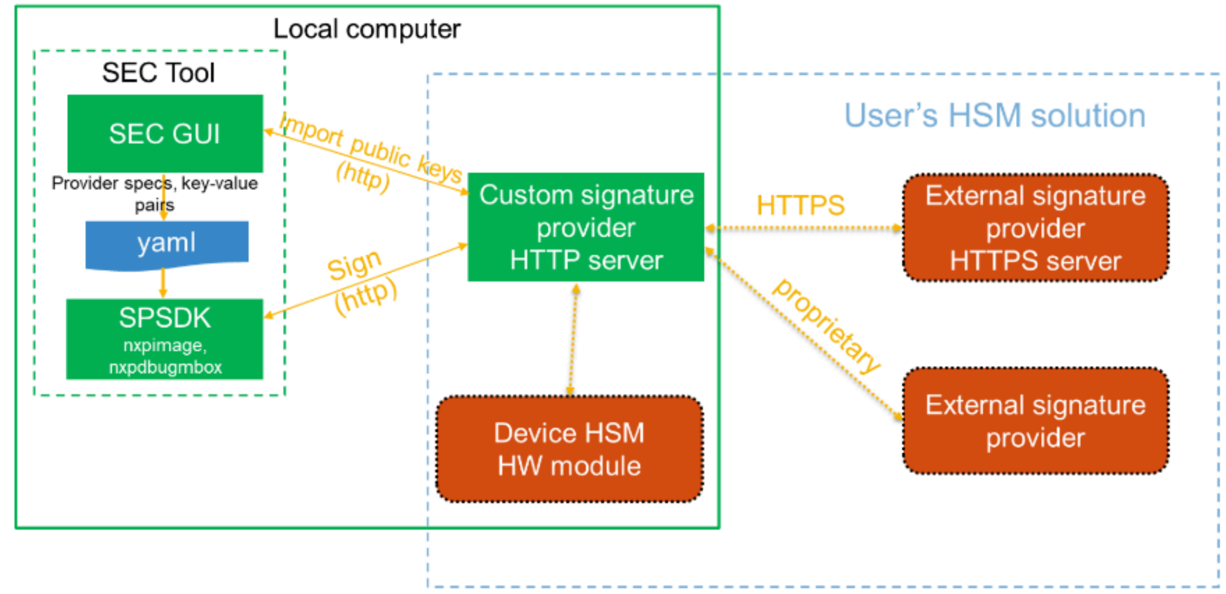

Local computer
SEC Tool
SEC GUI
Provider specs, key-value pairs
yaml
SPSDK
nxpimage, nxpdbugmbox
Import public keys (http)
Sign (http)
Custom signature provider HTTP server
User's HSM solution
HTTPS
External signature provider HTTPS server
proprietary
External signature provider
Device HSM HW module

Figure 74. Expected structure of signature provider

6.12.1 Run the server

To run, several prerequisites are required:

- Python 3.10 or later
- Python packages specified in requirements.txt. Using a virtual environment is recommended.

To start the required server, launch the following command on the command line: **python server.py**

The server logs every action, so it is possible to review what actions were executed.

6.12.1.1 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 will be 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_example` the setting can be left as is.

5. Click the **Test connection** button to verify if the server is configured properly.
6. 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**.
   - Close the dialog and go back to PKI management tab and click the button **Import keys** to import public keys from the previous workspace. 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).
7. On the **Build** tab select the key as normally, now config files for SB, MBI, and certification block will be using the signature provider configuration.
8. Now, the signature provider is configured. It is possible to build a signed image.

7 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 identifying the top-level operation (building, flashing, provisioning, generating keys or detecting the list of USB devices) and additional operation-specific options.

To display available commands, arguments, and examples, run the following command from the command prompt:

```
c:/nxp/MCUX_Provi_v4.1/bin/securep.exe -h
```

To display available arguments for a specific command, run the following command from the command prompt:

```
c:/nxp/MCUX_Provi_v4.1/bin/securep.exe <command> -h
```

Note: *The location of the file is subject to installation folder.*

7.1 Build

With the **build** command, you can perform actions that you can otherwise perform in the Build image view of SEC.

7.1.1 Build arguments

The following arguments are available to the **build** command:

Table 18. Build-specific arguments

Argument	Description
-h, --help	Show this help message and exit.
--source-image SOURCE_IMAGE	Source image 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.
--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.
--firmware-version FIRMWARE_VERSION	Version of the application image firmware. The argument is applicable for K32W1xx, KW45xx, LPC55S3x, RT118x, MCXN94x, MCXN54x, RW61x processors.
--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.
--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.

Table 18. Build-specific arguments*...continued*

Argument	Description
--keyblob-keyid KEYBLOB_KEYID	32-bits value: Keyblob encryption key identifier. The argument is applicable for processors with the AHAB security system.
--target-image TARGET_IMAGE	Target image for building the boot image.
--secret-key-type {AES-128, AES-192, AES-256}	The HAB encryption algorithm, default is processor-specific.
--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.
--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	Path to the Device Configuration Data binary file.
--xip-enc-otpmk-config XIP_ENC_OTPMK_CONFIG	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.
--bee-user-keys-config BEE_USER_KEYS_CONFIG	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.
--otfad-config OTFAD_CONFIG	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.
--iee-config IEE_CONFIG	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.
-v, --verbose	Increase output verbosity
--device name of the selected processor	Target processor
--boot-type VALUE	Secure boot type. Run securep --help to see all supported boot types.
--life-cycle VALUE	Requested life-cycle state of the processor, one of closed_deploy_infield_shadows,infield_locked_shadows,open_develop,reduced,closed_deploy_infield,infield_locked,closed_level_4.
--trust-provi {disabled, smart_card, device_hsm}	Trust provisioning type.
--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.*

Table 18. Build-specific arguments*...continued*

Argument	Description
--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 file (for the TrustZone enabled image with custom configuration)
--keysource {OTP, KeyStore}	Key source for RTxxx secured images
--userkey USERKEY	Key applicable for RTxxx secured images: for OTP key-source it represents the master key; for key-store it represents the key used for signature
--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
--prince-cfg PRINCE_CFG	JSON file with PRINCE configuration. See bin/schema/prince_config_schema_v<version>.json in the SEC installation folder.
--iped-cfg IPED_CFG	JSON file with PRINCE configuration; file format is specified by schema/prince_config_schema_v?.json
--otp-cfg OTP_CFG	Path to JSON file with USER OTP configuration. It is recommended to export the file from the OTP Configuration dialog.
--cmpa-cfg CMPA_CFG	Path to JSON file with USER CMPA configuration. It is recommended to export the file from the PFR Configuration dialog.
--cfpa-cfg CFPA_CFG	Path to JSON file with USER CFPA configuration. It is recommended to export the file from the PFR Configuration dialog.
--romcfg-cfg ROMCFG_CFG	Path to JSON file with USER ROMCFG configuration. It is recommended to export the file from the IFR Configuration dialog.
--cmpa-cfg/--cfpa-cfg PATH	Path to JSON file with PFR configuration of the given CMPA/CFPA page. It is recommended to export the file from the PFR Configuration dialog.
--dual-image-boot-cfg DUAL_IMAGE_BOOT_CFG	JSON file with dual image boot configuration; file format is specified by schema/dual_image_boot_schema_v?.json. The argument is applicable for RTxxx, RT116x, RT117x, RT118x, LPC55S36 (only FlexSPI NOR) and LPC553x processors.
--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`. The argument is only applicable for RT116x, RT117x, and RT118x processors.

Table 19. 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

Table 19. Boot-device arguments (mutually exclusive)*...continued*

Argument	Description
--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.

7.2 Write

With the **write** command, you can perform actions that you can otherwise perform in the Write image view of SEC.

7.2.1 Write arguments

The following arguments are available to the **write** command:

Table 20. Write-specific arguments

Argument	Description
-h, --help	Show this help message and exit
--source-image SOURCE_IMAGE	Source image to be uploaded to the target
--write-params-cfg WRITE_PARAMS_CFG	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 VALUE	Requested life-cycle state of the processor, one of open_ develop, closed_deploy_infield, infield_locked.
--trust-provi {disabled,smart_card,device_hsm}	Trust provisioning type.
-v, --verbose	Increase output verbosity
--device name of the selected processor	Target processor
--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 mutually exclusive boot-device arguments, see Table 19

Table 21. 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

Table 21. Connection arguments (mutually exclusive)*...continued*

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

Note: *For connection to the board, USB or Serial port has to be specified. If nothing is specified, USB autodetection is applied.*

7.3 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.

7.3.1 Generate keys arguments

Following arguments are available to the **generate** command:

Table 22. Generate-specific arguments

Argument	Description
-h, --help	Show this help message and exit
--keys-cfg KEYS_CFG JSON	File with the keys configuration
--device name of the selected processor	Target processor
--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 memory arguments, see Table 19

7.4 Manufacture

7.4.1 Manufacture arguments

Manufacture command allows running the selected script several times in parallel, each time for different connection. Following arguments are available to the **manufacture** command:

Table 23. 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.

Table 23. Manufacture-specific arguments*...continued*

Argument	Description
--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 auto-detection cannot be combined with the other options. Parameters and default values for SIO operation are described in SPSDK documentation.

7.5 Command-line examples

Example: How to build and write an image for configuration stored in */workspaces/mcuxprovi* 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
```

7.6 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
- **ifr** Generating the content of the flash region file (ROMCFG).
- **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 OEM.
- **nxpdevscan** Utility that detects NXP devices connected to the host PC over USB, UART, I2C, and SPI connections.

nxpele	Utility for communication with the EdgeLock Enclave on target
nxpimage	Builds bootable image and SB files
pfr	Generating protected flash region files (cmpa/cfpa)
sdphost	Replacement for the legacy sdphost tool
shadowregs	Shadow registers control tool.
tpconfig	Configuration of Smart Card for trust provisioning
tphost	Trust provisioning operation

8 Troubleshooting

This chapter contains known problems and recommended solutions. Refer to Release Notes for last-minute issues.

- The application must be installed into the location where the user has write access.
- By default, 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 and fix problems.

8.1 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).

8.2 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 with higher priority by renaming the file with a lower number at the beginning.
- Ubuntu 22 and USB2Serial CP210x. On Ubuntu 22, there is a conflicting package brltty that causes generic issues with CP210x USB to serial converter. Uninstalling the `brltty` package fixes the issue. For more details, see https://bugs.launchpad.net/ubuntu/+source/brltty/+bug/1970408
- The SEC tool works well with the Xorg display server. Wayland, default in Ubuntu, causes various UI glitches, that is why application shortcut contains configuration to use Xorg (x11) backend. If 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/MCUX_Provi_v8/bin/securep
  ```

8.3 Mac OS X

- Fields with invalid input are marked with a background red color. Fixing the value might not change the background color correctly and the focus must be changed to another field for correct repaint.
- P&E Micro does not support Mac OS Aarch64 yet.

8.4 i.MX RTxxx

- 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 documentation RTxxx Device workflow/Booting images chapters.
- 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 in the processor only if the processor is unsecured.

8.5 i.MX RT1024

SD card boot device is not supported for the MIMXRT1024-EVK board due to limitation in flashloader.

8.6 i.MX RT1015-EVK / Mac OS X / UART

OpenSDA does not work on Mac OS X when the device has HAB enabled and the UART port is used for communication. Either use USB HID communication, or disconnect OpenSDA from RX and TX pins (jumpers J45 and J46). The device must be programmed via an external USB to serial converter (3.3 V).

8.7 i.MX RT1060-EVK / Mac OS X / UART

For communication over UART on MacOS, open J44 jumper (SDA_RST_TGTMCU).

8.8 Bootable image as source for build

On the **build** view, the bootable image can be used as input only for RT10xx processors. Support of other processors is planned for the next version.

8.9 Workspace compatibility

When workspace is opened in the newer version, it is automatically converted into new format that prevents opening in previous version of the SEC (SEC tool might not start). To open it back in older tool, settings backup is created in the workspace and manual file rename has to be done.

8.10 Debug probes

- Detecting PEmicro probes and detection of probes after PEmicro was detected has these limitations:
  1. PEmicro must be connected before the first attempt for detection, if not connected it will be impossible to detect 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 be always listed as an available probe even if it is not connected anymore.
- On Mac OS, the PEmicro debug probe does not work with the pyOcd library, it is recommended to use the PE micro-library.
- Dependency on USB drivers for debug probe drivers is described in ReleaseNotes.txt in "System requirements".

8.11 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). Here are tips what to check to make it working:

- The processor should not be in ISP mode.
- The processor should be secured. The operation may fail if the debug port is already opened.

9 Note about the source code in the document

Example code shown in this document has the following copyright and BSD-3-Clause license:

10 Revision history

Table 24. Revision history

Document ID	Release date	Description
MCUXSPTUG v.13	19 January 2024	Updated for MCUXpresso Secure Provisioning Tools v8. Section 6.3, Section 6.8, Section 6.9 are added.
MCUXSPTUG v.12	12 July 2023	Updated for MCUXpresso Secure Provisioning Tools v7: minor updates.
MCUXSPTUG v.11	15 March 2023	Updated for MCUXpresso Secure Provisioning Tools v6: minor updates.
MCUXSPTUG v.10	13 January 2023	Added: support for LPC55S36, section "LPC55S3x device workflow"; section Section 2 is modified.
MCUXSPTUG v.9	26 September 2022	Added: support for LPC55Sxx and LPC55xx families (LPC553x, LPC552x, LPC551x, and LPC550x), Section 5.4.6, Section 5.8, Boot device configuration are added.
MCUXSPTUG v.8	24 June 2022	Updated for MCUXpresso Secure Provisioning Tools v4.1
MCUXSPTUG v.7	09 May 2022	The term "Java Smart Card" is replaced with "Smart Card".
MCUXSPTUG v.6	22 April 2022	Updated for MCUXpresso Secure Provisioning Tool v4: Section 6.10, and Section 8.8 are added, screenshots are updated.
MCUXSPTUG v.5	30 September 2021	The acronym "SPT" is replaced with "SEC" in all the flowcharts of the document.

Table 24. Revision history*...continued*

Document ID	Release date	Description
MCUXSPTUG v.4	28 July 2021	OTP/PFR Configuration rework, new workflows/flowcharts for OTFAD boot type, restructuring, new device information, minor changes
MCUXSPTUG v.3	20 April 2021	Updated for MCUXpresso Secure Provisioning Tools v3
MCUXSPTUG v.2	14 October 2020	Updated for MCUXpresso Secure Provisioning Tools v2.1
MCUXSPTUG v.1	25 August 2020	Updated for MCUXpresso Secure Provisioning Tools v2
MCUXSPTUG v.0	08 January 2020	Initial version

Legal information

Definitions

Draft — A draft status on a document indicates that the content is still under internal review and subject to formal approval, which may result in modifications or additions. NXP Semiconductors does not give any representations or warranties as to the accuracy or completeness of information included in a draft version of a document and shall have no liability for the consequences of use of such information.

Disclaimers

Limited warranty and liability — Information in this document is believed to be accurate and reliable. However, NXP Semiconductors does not give any representations or warranties, expressed or implied, as to the accuracy or completeness of such information and shall have no liability for the consequences of use of such information. NXP Semiconductors takes no responsibility for the content in this document if provided by an information source outside of NXP Semiconductors.

In no event shall NXP Semiconductors be liable for any indirect, incidental, punitive, special or consequential damages (including - without limitation - lost profits, lost savings, business interruption, costs related to the removal or replacement of any products or rework charges) whether or not such damages are based on tort (including negligence), warranty, breach of contract or any other legal theory.

Notwithstanding any damages that customer might incur for any reason whatsoever, NXP Semiconductors' aggregate and cumulative liability towards customer for the products described herein shall be limited in accordance with the Terms and conditions of commercial sale of NXP Semiconductors.

Right to make changes — NXP Semiconductors reserves the right to make changes to information published in this document, including without limitation specifications and product descriptions, at any time and without notice. This document supersedes and replaces all information supplied prior to the publication hereof.

Suitability for use — NXP Semiconductors products are not designed, authorized or warranted to be suitable for use in life support, life-critical or safety-critical systems or equipment, nor in applications where failure or malfunction of an NXP Semiconductors product can reasonably be expected to result in personal injury, death or severe property or environmental damage. NXP Semiconductors and its suppliers accept no liability for inclusion and/or use of NXP Semiconductors products in such equipment or applications and therefore such inclusion and/or use is at the customer's own risk.

Applications — Applications that are described herein for any of these products are for illustrative purposes only. NXP Semiconductors makes no representation or warranty that such applications will be suitable for the specified use without further testing or modification.

Customers are responsible for the design and operation of their applications and products using NXP Semiconductors products, and NXP Semiconductors accepts no liability for any assistance with applications or customer product design. It is customer's sole responsibility to determine whether the NXP Semiconductors product is suitable and fit for the customer's applications and products planned, as well as for the planned application and use of customer's third party customer(s). Customers should provide appropriate design and operating safeguards to minimize the risks associated with their applications and products.

NXP Semiconductors does not accept any liability related to any default, damage, costs or problem which is based on any weakness or default in the customer's applications or products, or the application or use by customer's third party customer(s). Customer is responsible for doing all necessary testing for the customer's applications and products using NXP Semiconductors products in order to avoid a default of the applications and the products or of the application or use by customer's third party customer(s). NXP does not accept any liability in this respect.

Terms and conditions of commercial sale — NXP Semiconductors products are sold subject to the general terms and conditions of commercial sale, as published at https://www.nxp.com/profile/terms, unless otherwise agreed in a valid written individual agreement. In case an individual agreement is concluded only the terms and conditions of the respective agreement shall apply. NXP Semiconductors hereby expressly objects to applying the customer's general terms and conditions with regard to the purchase of NXP Semiconductors products by customer.

Export control — This document as well as the item(s) described herein may be subject to export control regulations. Export might require a prior authorization from competent authorities.

Suitability for use in non-automotive qualified products — Unless this document expressly states that this specific NXP Semiconductors product is automotive qualified, the product is not suitable for automotive use. It is neither qualified nor tested in accordance with automotive testing or application requirements. NXP Semiconductors accepts no liability for inclusion and/or use of non-automotive qualified products in automotive equipment or applications.

In the event that customer uses the product for design-in and use in automotive applications to automotive specifications and standards, customer (a) shall use the product without NXP Semiconductors' warranty of the product for such automotive applications, use and specifications, and (b) whenever customer uses the product for automotive applications beyond NXP Semiconductors' specifications such use shall be solely at customer's own risk, and (c) customer fully indemnifies NXP Semiconductors for any liability, damages or failed product claims resulting from customer design and use of the product for automotive applications beyond NXP Semiconductors' standard warranty and NXP Semiconductors' product specifications.

Translations — A non-English (translated) version of a document, including the legal information in that document, is for reference only. The English version shall prevail in case of any discrepancy between the translated and English versions.

Security — Customer understands that all NXP products may be subject to unidentified vulnerabilities or may support established security standards or specifications with known limitations. Customer is responsible for the design and operation of its applications and products throughout their lifecycles to reduce the effect of these vulnerabilities on customer's applications and products. Customer's responsibility also extends to other open and/or proprietary technologies supported by NXP products for use in customer's applications. NXP accepts no liability for any vulnerability. Customer should regularly check security updates from NXP and follow up appropriately.

Customer shall select products with security features that best meet rules, regulations, and standards of the intended application and make the ultimate design decisions regarding its products and is solely responsible for compliance with all legal, regulatory, and security related requirements concerning its products, regardless of any information or support that may be provided by NXP.

NXP has a Product Security Incident Response Team (PSIRT) (reachable at PSIRT@nxp.com) that manages the investigation, reporting, and solution release to security vulnerabilities of NXP products.

NXP B.V. — NXP B.V. is not an operating company and it does not distribute or sell products.

Trademarks

Notice: All referenced brands, product names, service names, and trademarks are the property of their respective owners.

NXP — wordmark and logo are trademarks of NXP B.V.

Amazon Web Services, AWS, the Powered by AWS logo, and FreeRTOS — are trademarks of Amazon.com, Inc. or its affiliates.

AMBA, Arm, Arm7, Arm7TDMI, Arm9, Arm11, Artisan, big.LITTLE, Cordio, CoreLink, CoreSight, Cortex, DesignStart, DynamIQ, Jazelle, Keil, Mali, Mbed, Mbed Enabled, NEON, POP, RealView, SecurCore, Socrates, Thumb, TrustZone, ULINK, ULINK2, ULINK-ME, ULINK-PLUS, ULINKpro, μVision, Versatile — are trademarks and/or registered trademarks of Arm Limited (or its subsidiaries or affiliates) in the US and/or elsewhere. The related technology may be protected by any or all of patents, copyrights, designs and trade secrets. All rights reserved.

Apple — is a registered trademark of Apple Inc.

EdgeLock — is a trademark of NXP B.V.

Freescale — is a trademark of NXP B.V.

IAR — is a trademark of IAR Systems AB.

i.MX — is a trademark of NXP B.V.

Intel, the Intel logo, Intel Core, OpenVINO, and the OpenVINO logo — are trademarks of Intel Corporation or its subsidiaries.

J-Link — is a trademark of SEGGER Microcontroller GmbH.

MCX — is a trademark of NXP B.V.

Microsoft, Azure, and ThreadX — are trademarks of the Microsoft group of companies.

Oracle and Java — are registered trademarks of Oracle and/or its affiliates.

Contents

Please be aware that important notices concerning this document and the product(s) described herein, have been included in section 'Legal information'.

For more information, please visit: https://www.nxp.com

Date of release: 19 January 2024
Document identifier: MCUXSPTUG