NXP Semiconductors
User Guide

Document identifier: MCUXSPTUG
Rev. 4, 28 July 2021

MCUXpresso Secure Provisioning Tool v3.1

Contents

Chapter 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 fuse images, while experienced users will rediscover features from the existing toolset (sdphost, blhost, elftosb, cst, image_enc) 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

Chapter 2
Features

Features of **MCUXpresso Secure Provisioning Tool v3.1** include:

- Support target connectivity via UART and USB-HID serial download modes
- Support multiple user application image formats
- Automated conversion of bare images to bootable images
- Downloading a bootable image in the target boot device
- Customization of the boot device either using GUI, or predefined Flash Configuration Blocks
- Generation of certificate trees for image signing and encryption, or use of user supplied certificates
- Generation of signed and optionally encrypted executables
- Support for development (unsigned) boot type
- Support for authenticated (signed) and encrypted boot type
- Key provisioning and fusing as dictated by boot type
- Advanced OTP/PFR configuration
- Command line interface for customized boot flows
- 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

2.1 Supported devices and features

Table 1. Supported boot types

Family	Processor	Unsigned	CRC	Signed/Authenticated	Encrypted XIP	Encrypted non-XIP
LPC55Sxx	LPC55S69	✓	✓	✓	PRINCE	
	LPC55S66	✓	✓	✓	PRINCE	
	LPC55S28	✓	✓	✓	PRINCE	
	LPC55S26	✓	✓	✓	PRINCE	
	LPC55S16	✓	✓	✓	PRINCE	
	LPC55S14	✓	✓	✓	PRINCE	
	LPC55S06	✓	✓	✓	PRINCE	
	LPC55S04	✓	✓	✓	PRINCE	
MIMXRTxxx	MIMXRT533S	✓	✓	✓	OTFAD	
	MIMXRT555S	✓	✓	✓	OTFAD	
	MIMXRT595S	✓	✓	✓	OTFAD	
	MIMXRT633S	✓	✓	✓	OTFAD	

Table continues on the next page...

Table 1. Supported boot types (continued)

Family	Processor	Unsigned	CRC	Signed/Authenticated	Encrypted XIP	Encrypted non-XIP
	MIMXRT685S	✓	✓	✓	OTFAD	
MIMXRT10xx	MIMXRT1064	✓		✓	BEE	HAB
	MIMXRT1060	✓		✓	BEE	HAB
	MIMXRT1050	✓		✓	BEE	HAB
	MIMXRT1024	✓		✓	BEE	
	MIMXRT1020	✓		✓	BEE	HAB
	MIMXRT1015	✓		✓	BEE	
	MIMXRT1010	✓		✓		
MIMXRT11xx	MIMXRT1176	✓		✓	OTFAD	
	MIMXRT1175	✓		✓	OTFAD	
	MIMXRT1173	✓		✓	OTFAD	
	MIMXRT1172	✓		✓	OTFAD	
	MIMXRT1171	✓		✓	OTFAD	
	MIMXRT1166	✓		✓	OTFAD	
	MIMXRT1165	✓		✓	OTFAD	

Table 2. Supported boot devices

Family	Processor	Boot Device			
		Internal FLASH	Flash NOR	Flash NAND	SD Card
LPC55Sxx	LPC55S69	✓			
	LPC55S66	✓			
	LPC55S28	✓			
	LPC55S26	✓			
	LPC55S16	✓			
	LPC55S14	✓			
	LPC55S06	✓			
	LPC55S04	✓			
RTxxx	MIMXRT533S		✓		
	MIMXRT555S		✓		
	MIMXRT595S		✓		
	MIMXRT633S		✓		
	MIMXRT685S		✓		

Table continues on the next page...

Table 2. Supported boot devices (continued)

Family	Processor	Boot Device			
		Internal FLASH	Flash NOR	Flash NAND	SD Card
RT10xx	MIMXRT1064		✓		✓
	MIMXRT1060		✓		✓
	MIMXRT1050		✓		✓
	MIMXRT1024		✓		
	MIMXRT1020		✓		✓
	MIMXRT1015		✓		
	MIMXRT1010		✓		
RT11xx	MIMXRT1176		✓	✓	✓
	MIMXRT1175		✓	✓	✓
	MIMXRT1173		✓	✓	✓
	MIMXRT1172		✓	✓	✓
	MIMXRT1171		✓	✓	✓
	MIMXRT1166		✓	✓	✓
	MIMXRT1165		✓	✓	✓

Table 3. Supported other features

Family	Processor	Other Features			
		TrustZone	DCD	OTP Config/PFR Config	Shadow Registers
LPC55Sxx	LPC55S69	✓		PFR Config	
	LPC55S66	✓		PFR Config	
	LPC55S28	✓		PFR Config	
	LPC55S26	✓		PFR Config	
	LPC55S16	✓		PFR Config	
	LPC55S14	✓		PFR Config	
	LPC55S06	✓		PFR Config	
	LPC55S04	✓		PFR Config	
RTxxx	MIMXRT533S	✓		OTP Config	✓
	MIMXRT555S	✓		OTP Config	✓
	MIMXRT595S	✓		OTP Config	✓
	MIMXRT633S	✓		OTP Config	✓
	MIMXRT685S	✓		OTP Config	✓

Table continues on the next page...

Table 3. Supported other features (continued)

Family	Processor	Other Features			
		TrustZone	DCD	OTP Config/PFR Config	Shadow Registers
RT10xx	MIMXRT1064		✓	OTP Config	
	MIMXRT1060		✓	OTP Config	
	MIMXRT1050		✓	OTP Config	
	MIMXRT1024		✓	OTP Config	
	MIMXRT1020		✓	OTP Config	
	MIMXRT1015			OTP Config	
	MIMXRT1010			OTP Config	
RT11xx	MIMXRT1176		✓	OTP Config	
	MIMXRT1175		✓	OTP Config	
	MIMXRT1173		✓	OTP Config	
	MIMXRT1172		✓	OTP Config	
	MIMXRT1171		✓	OTP Config	
	MIMXRT1166		✓	OTP Config	
	MIMXRT1165		✓	OTP Config	

Chapter 3
Minimum System Requirements

Following are the minimum system requirements for running SEC:

- Microsoft(R) Windows(R) 10 (64-bit)
- Mac OS X operating system (10.14.6, 10.15 or later)
- Ubuntu 20.04 LTS 64 bit, with GNOME and OpenSSL 1.1.1 11 [Sep 2018]
- 4GB RAM
- Display with resolution 1024 x 768

Chapter 4
Terms and Definitions

Table 4. 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
Cipher block	The minimum amount of data on which a block cipher operates
Ciphertext	Encrypted data
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
DCD	Device Configuration Data, a binary table used by the ROM code to configure the device at 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
ECB	Electronic Code Book, a cipher mode with no feedback between the ciphertext blocks
EKIB	Encrypted Key Info Block
EPRDB	Encrypted Protection Region Descriptor Block
HAB	High Assurance Boot, a software library executed in internal ROM on the Freescale processor at boot time which, 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.

Table continues on the next page...

Table 4. Terms and Definitions (continued)

Term	Definition
Hash	Digest computation algorithm
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 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	Unique value computed from the data using a hash algorithm. Provides only an integrity check (unless encrypted).
OS	Operating System
OTFAD	On-The-Fly AES Decryption
OTP	One-Time Programmable. OTP hardware includes masked ROM, and electrically programmable fuses (eFuses).
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 AES-CTR algorithm
Rijndael	Block cipher chosen by the US Government to replace DES. Pronounced *rain-dahl*.
RSA	Public key cryptography algorithm developed by Rivest, Shamir, and Adleman. Accelerator (including hash acceleration) 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.
Session key	Encryption key generated at the time of encryption. Only ever used once.
SHA-1	Hash algorithm that produces a 160-bit message digest
SRK	Super Root Key, an RSA key pair which 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

Table continues on the next page...

Table 4. Terms and Definitions (continued)

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

Chapter 5
Installation

This chapter describes the procedure required to install SEC on Windows, MacOS, and Linux operating systems.

5.1 Windows

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

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

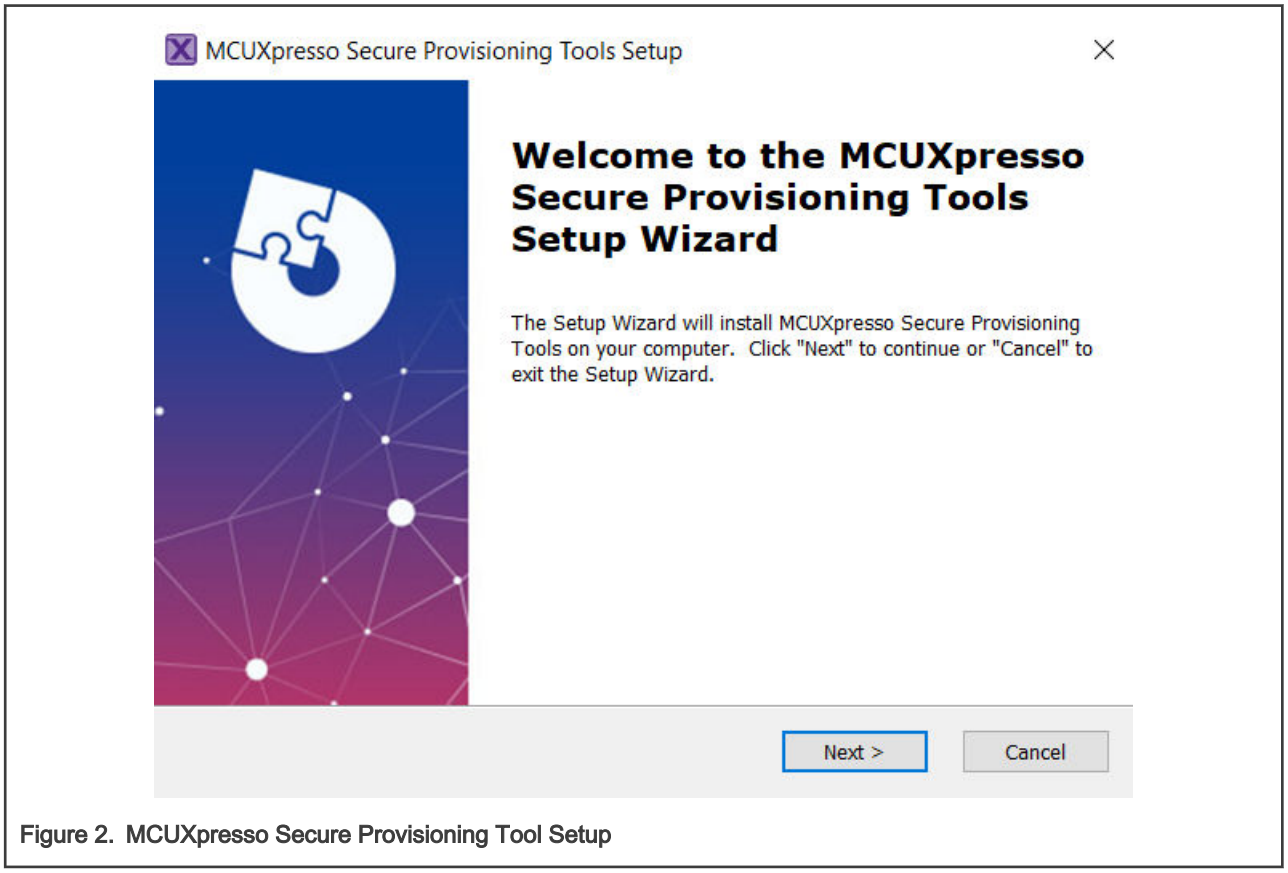

Figure 2. MCUXpresso Secure Provisioning Tool Setup

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

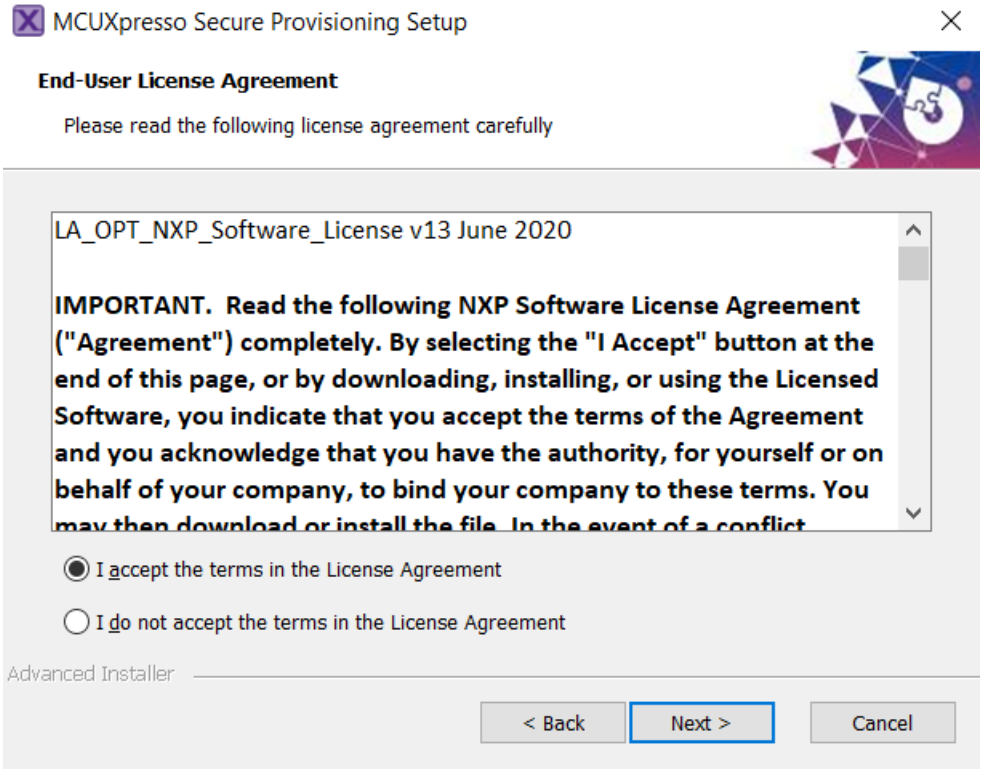

Figure 3. Accepting the end-user license agreement

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**.

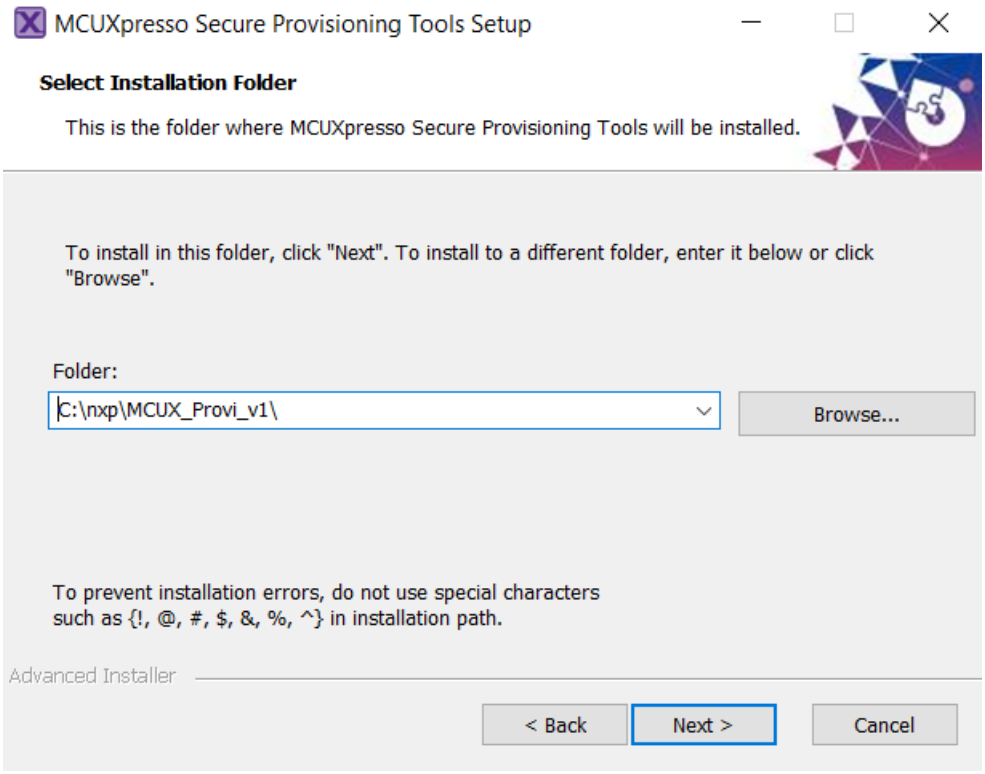

Figure 4. Selecting installation folder

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

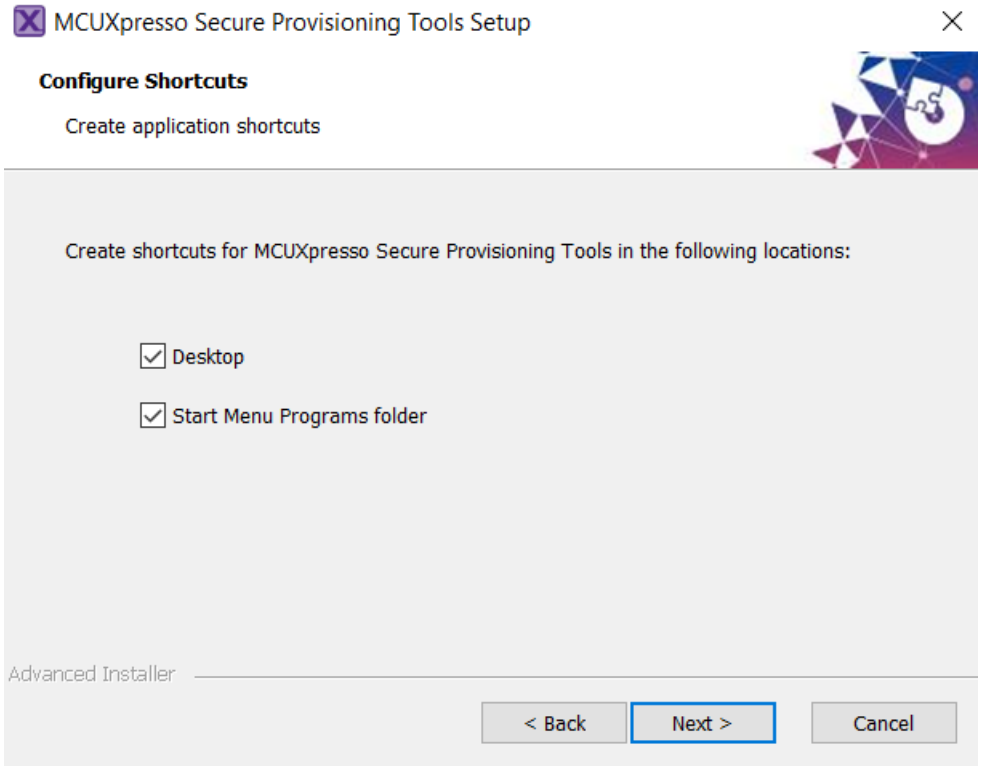

Figure 5. Configuring shortcuts

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

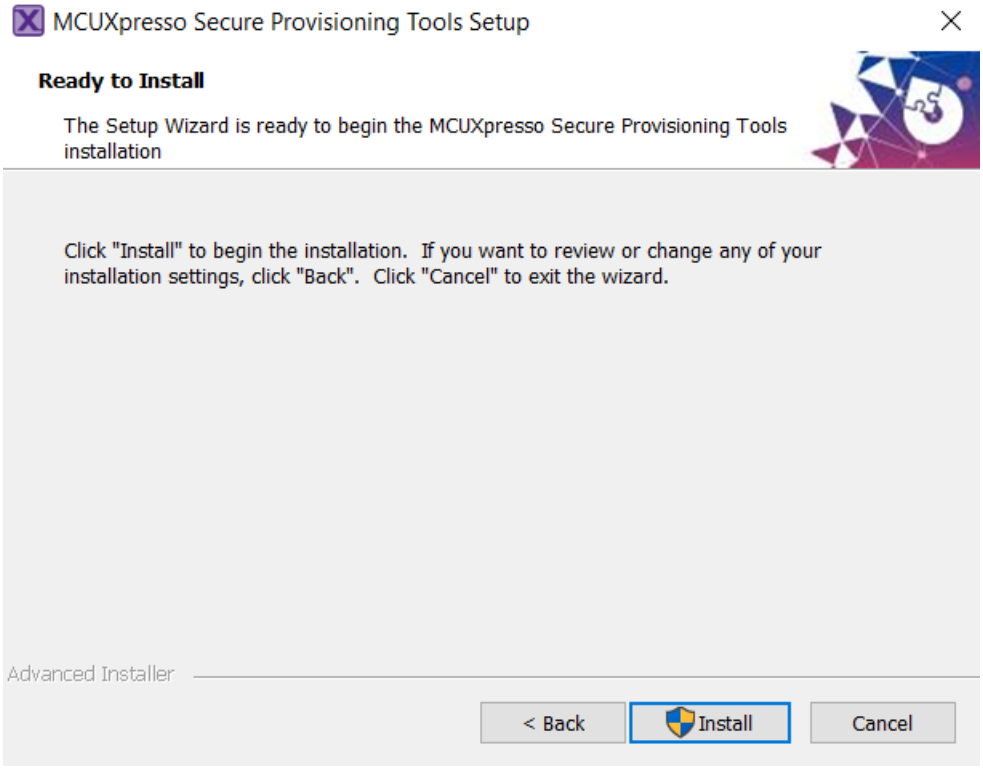

Figure 6. Ready to install

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.

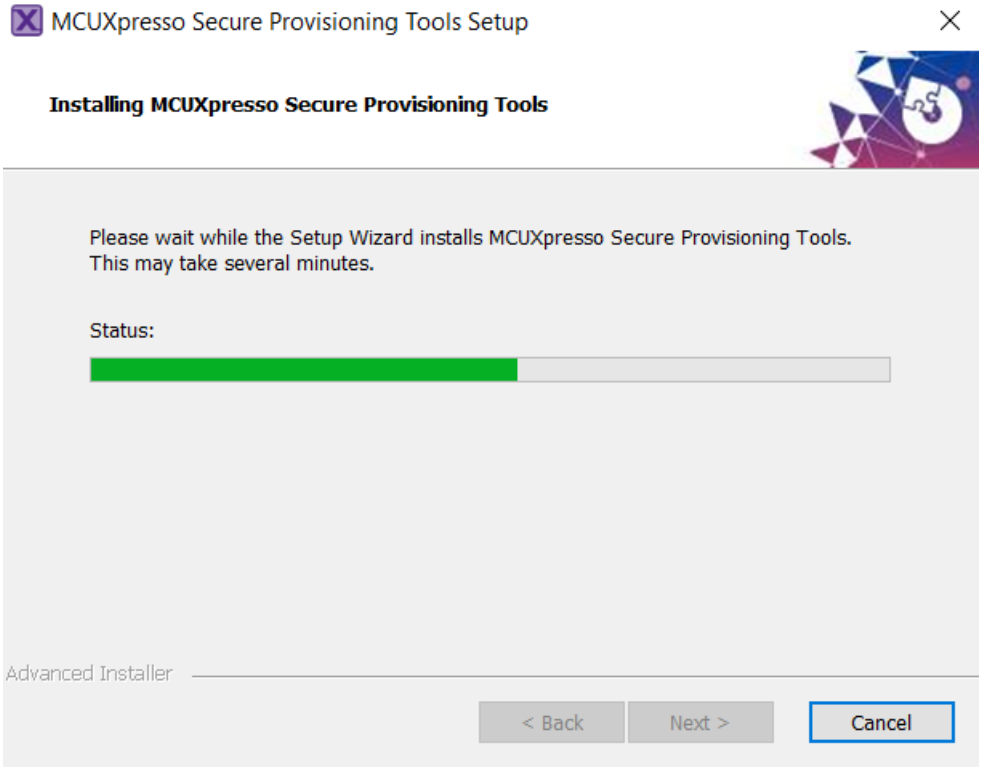

Figure 7. Installing MCUXpresso Secure Provisioning Tool

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

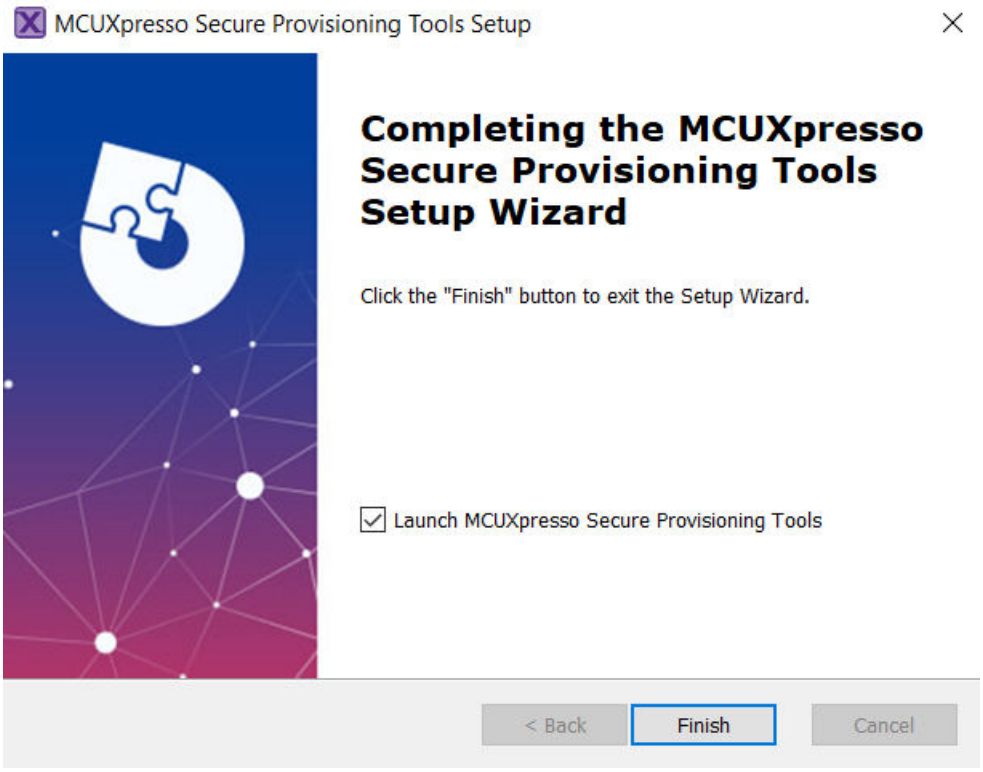

Figure 8. Completing installation

9. To start using SEC, run the tool from the desktop shortcut on 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>*.

5.2 MacOS

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

1. Visit the MCUXpresso website to download the SEC installer for MacOS.
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 and 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**.

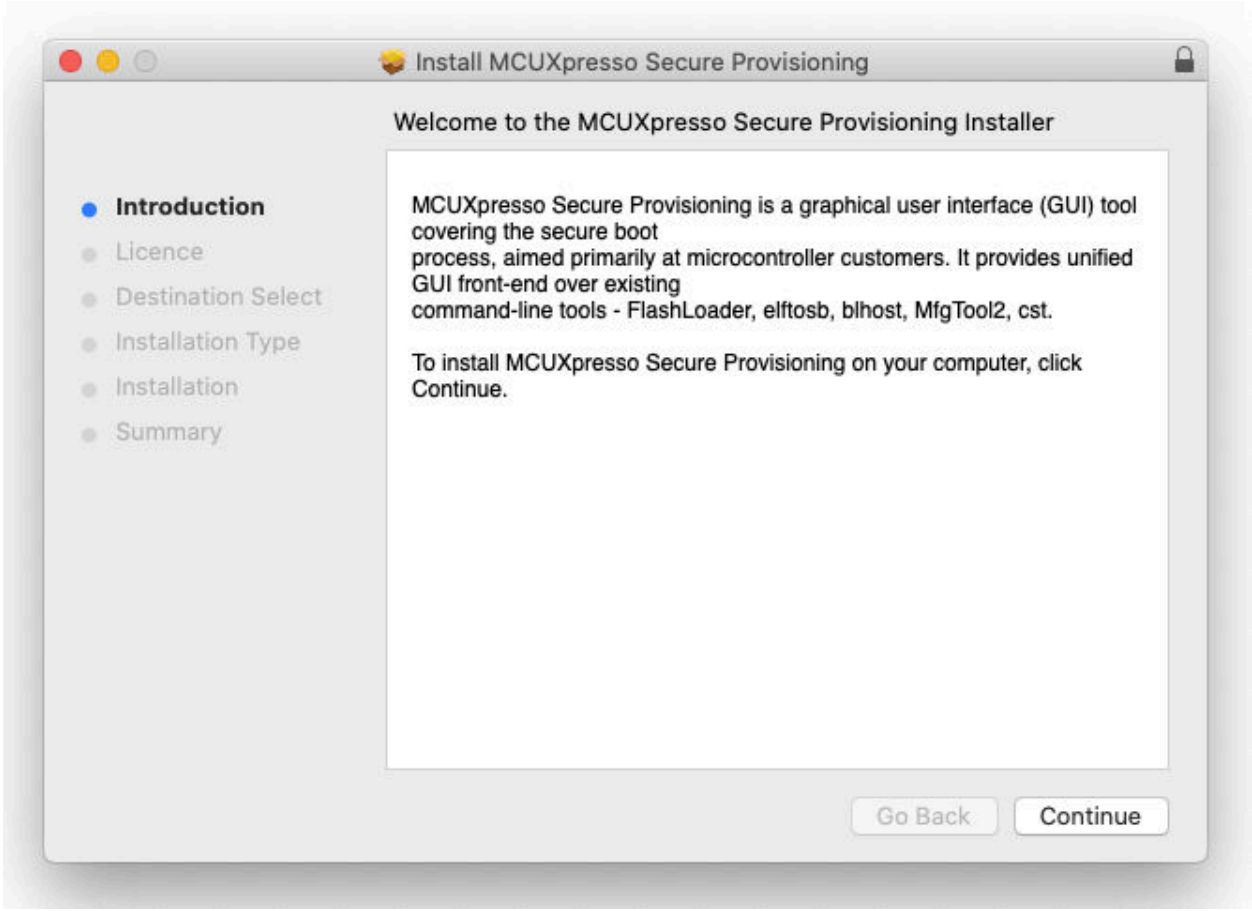

Figure 9. Introduction

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

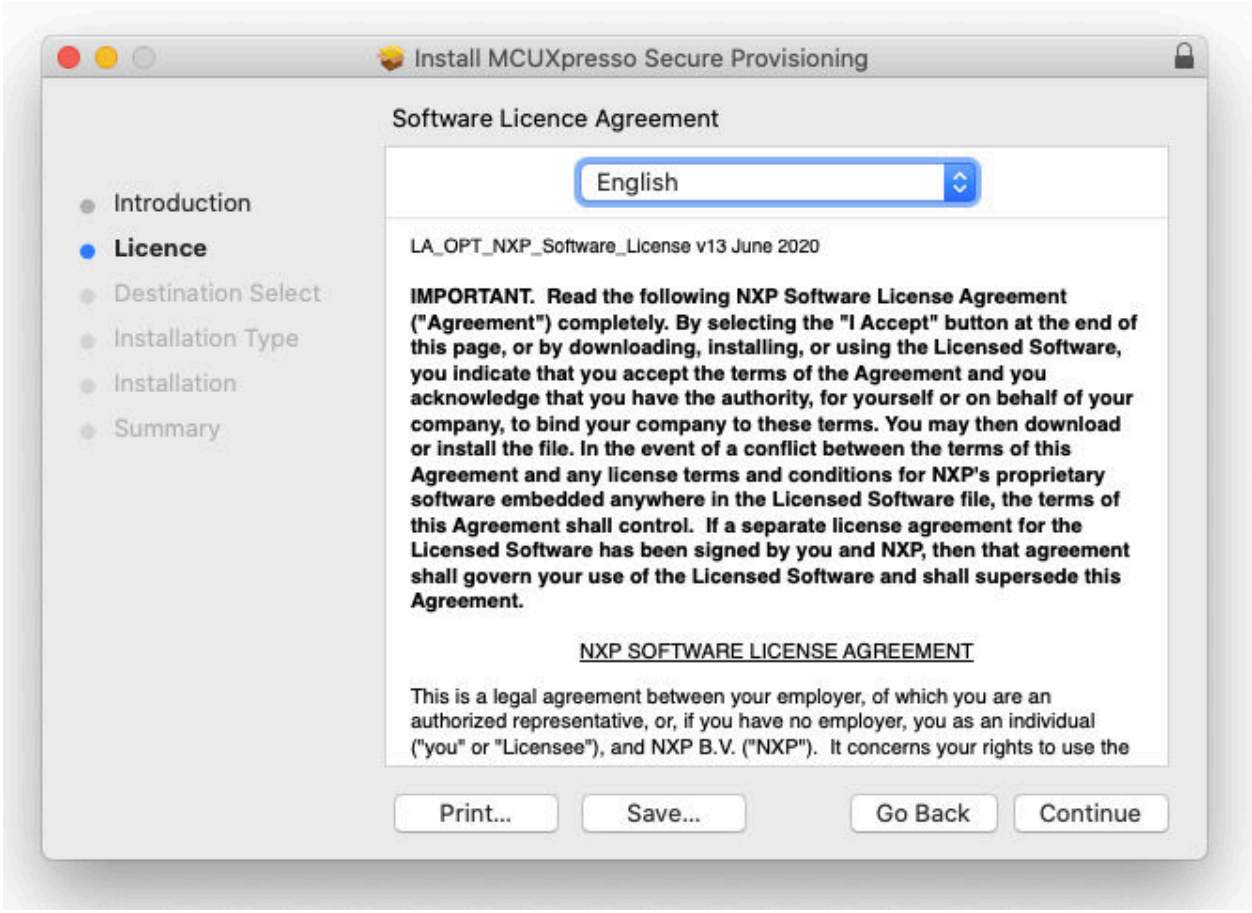

Figure 10. Software license agreement

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

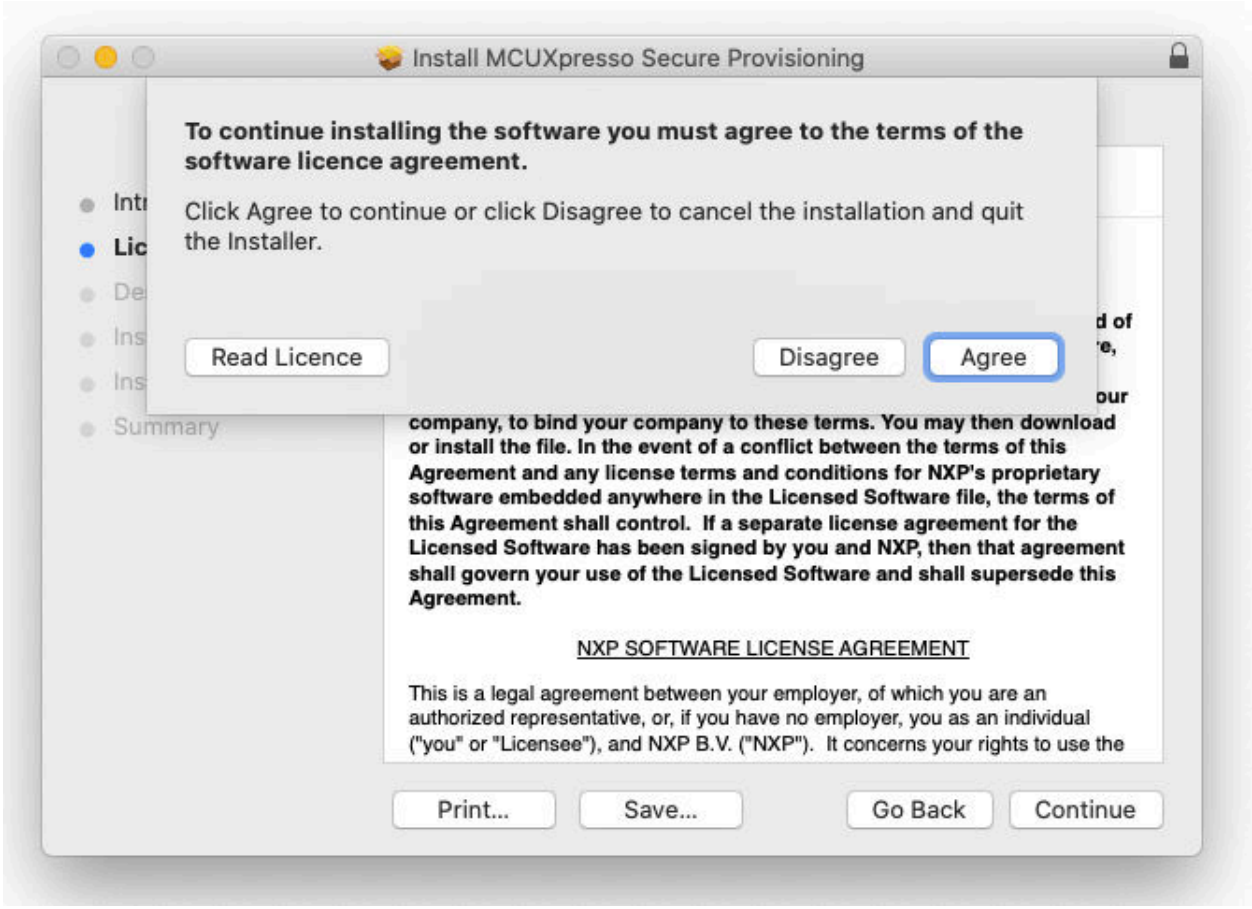

Figure 11. Accepting software license agreement

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

Figure 12. Select destination

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

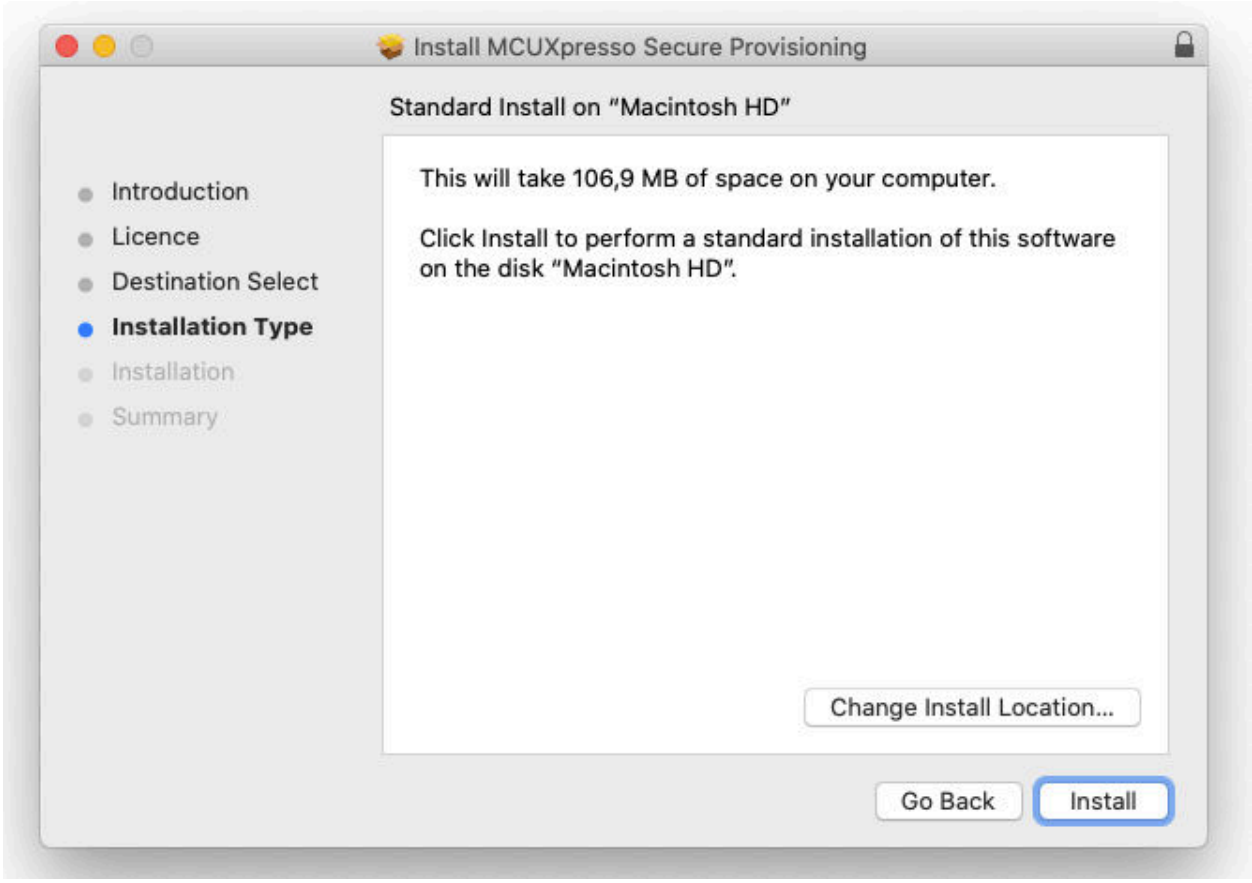

Figure 13. Installation type

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

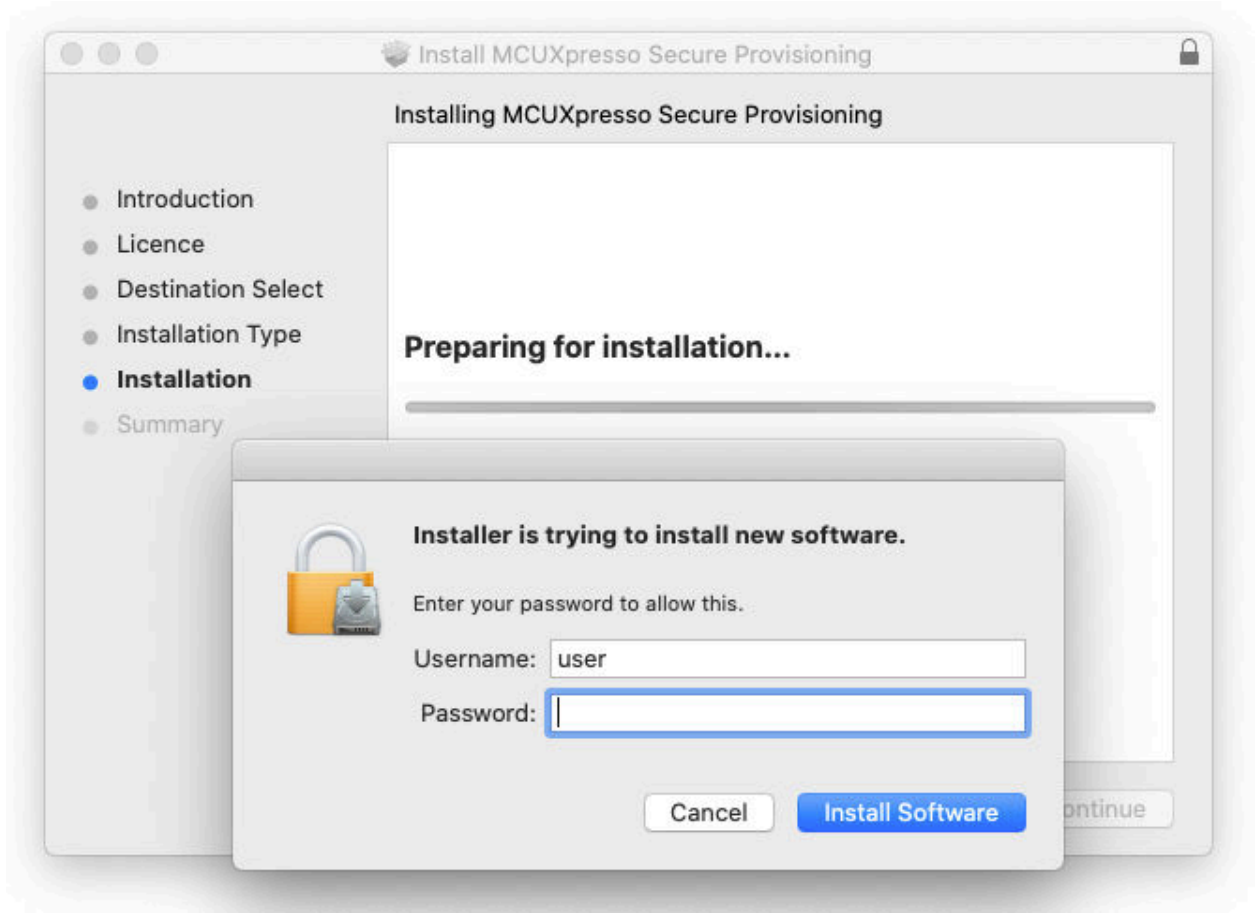

Figure 14. Install software

9. Click **Continue**.

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

10. On the **Summary** page, click **Close**.

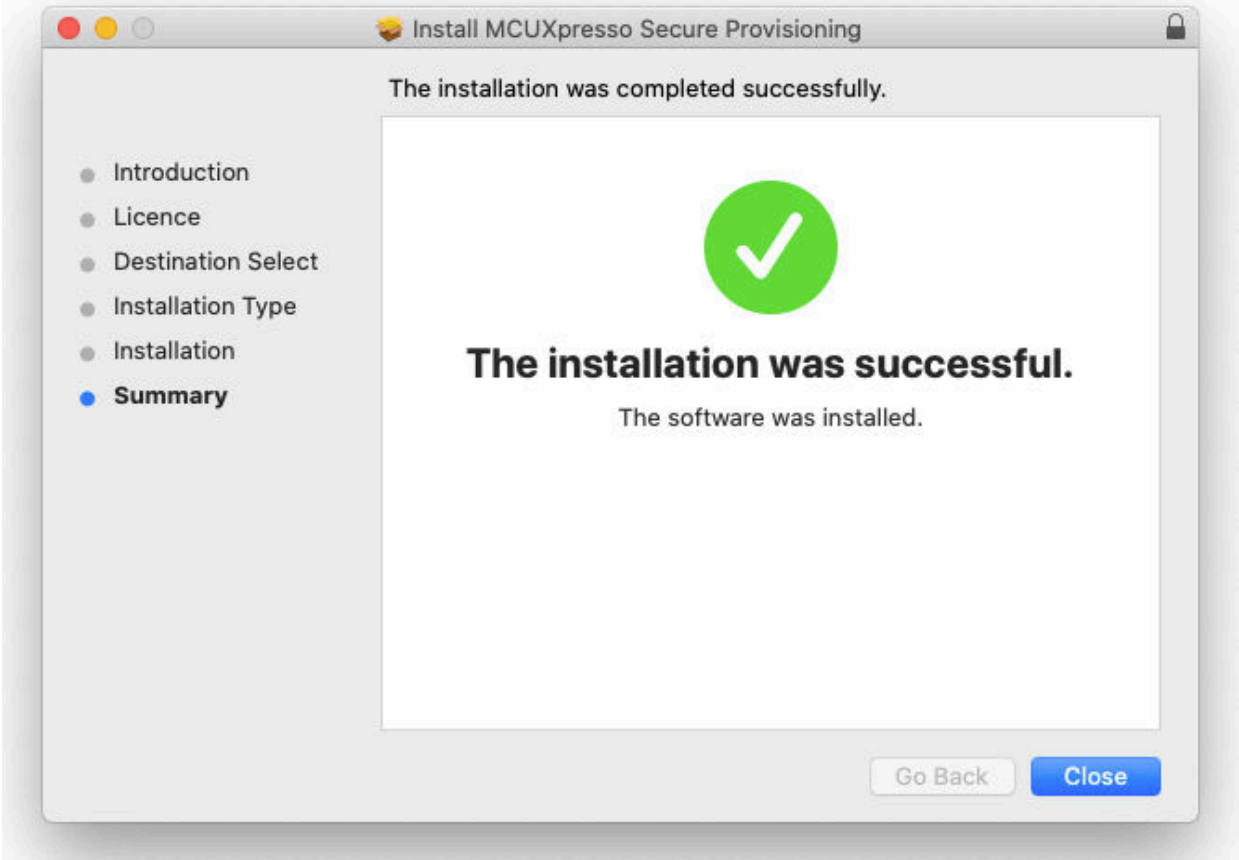

Figure 15. Summary

5.2.1 Enabling USB Connection on MacOS

During the first connection to the target by USB, MacOS X Catalina will block the access to USB HID devices as a security measure and the operation will fail with error.

Perform the following steps to enable USB connection:

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

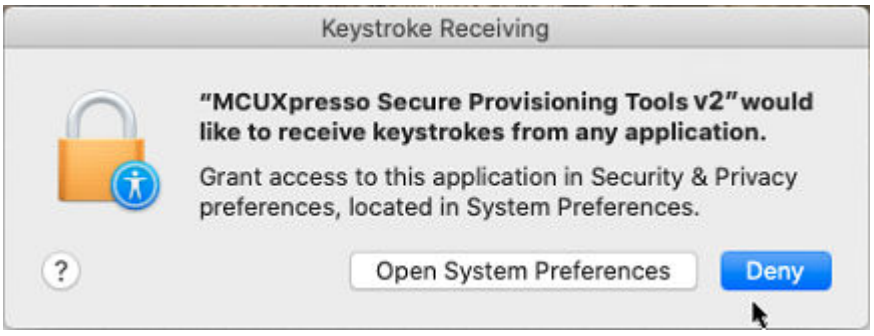

Figure 16. Open System Preferences

2. Unlock **Privacy** preferences to enable changes.
3. Select **MCUXpresso Secure Provisioning <version>**, confirm, and quit the application.

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.

5.3 Linux

You can install SEC on Ubuntu using Ubuntu Software Center. The installation uses the DEB package.

1. Visit the MCUXpresso website to download the SEC installer for Linux.
2. Open the terminal and change the directory where the installer is downloaded, make the installer executable and run with sudo.

```
$ cd ~/Downloads
$ chmod +x mcuxpresso-secure-provisioning-<version>_<architecture>.deb.bin
$ sudo ./mcuxpresso-secure-provisioning-<version>_<architecture>.deb.bin
```

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

NOTE

Installation prerequisites are necessary due to known Ubuntu bug: https://bugs.launchpad.net/ubuntu/+source/gnome-software/+bug/1573408.

Chapter 6
User Interface

SEC offers a simple and user-friendly user interface. It consists of the **Menu bar**, **Toolbar**, and three main views accessible through tabs: **Build Image**, **Write Image**, and **Keys Management**. The bottom part of the interface is occupied by the **Log** window.

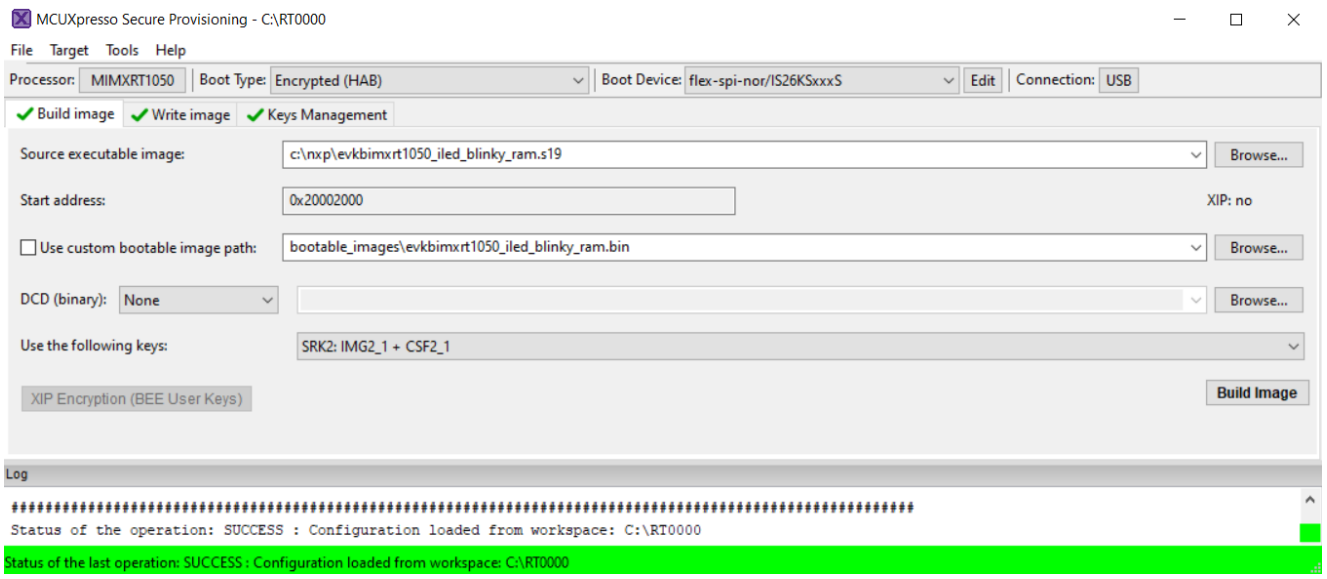

Figure 18. SEC user interface

6.1 Menu and Settings

6.1.1 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 ...	Create a new workspace. You will be prompted to specify its location and choose from the supported processors. In the case the location already contains a workspace, the workspace will be opened and not created. For more information, see Workspaces.
Select Workspace ...	Switch to another workspace. You will be prompted to specify which workspace to open. For more information, see Workspaces.
Save Settings	Save the current workspace settings.
Recent Workspaces	Display a list of recently used workspaces. For more information, see Workspaces. Number of displayed workspaces can be customized in Preferences.
Generate Scripts	Generate the build and write scripts for all operating systems according to the configuration. The scripts are generated into the workspace.
Preferences	Open the **Preferences** dialog. For more information, see Preferences.
Exit	Exit SEC.

Target Target device-related operations

	Connection ...	Open the **Connection** configuration dialog. For more information, see Connection.
Tools	List of additional tools	
	Manufacturing Tool	Open the **Manufacturing Tool**. For more information, see Manufacturing Tool.
Help	User help and additional general information	
	Contents	Open the User Manual.
	About	Display information about the current version.

6.1.2 Preferences

The **Preferences** dialog contains various settings applicable the whole SEC.

Currently, the following options are available:

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 configuration of your host. The selected value may affect the generated write script.
Maximal number of recent workspaces displayed in File menu	Customize maximal number of recent workspaces displayed in **File>Recent Workspaces**. Supported range 1 - 25. Default value: 9.
Read current values after OTP Configuration is opened	Choose how the reading of device values on opening the **OTP Configuration** is handled. Following options are available: **never** — Don't read the values automatically **ask** — Confirm the reading manually **always** — Automatically read device values
Use restricted data from directory:	Enables the use restricted data for selected processor. The checkbox is enabled only if restricted data are installed for the selected processor.
Install Restricted Data ...	Install restricted device data for use in OTP Configuration. The data are installed from a ZIP archive. SEC first verifies whether selected data are compatible with the current tool and if yes, the data are copied into SEC installation folder. Restart SEC to start using the data.

NOTE

Restricted data are only available for download from the NXP webpage after signing a Non Disclosure Agreement. Restricted data are currently available for RT10xx and RT11xx devices only.

6.1.3 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.

Workspace is always created for a specific device family. Once created, it can only be used to modify the configuration of devices belonging to that family.

To create a new workspace, select **File>New Workspace ...** from the **Menu bar**.

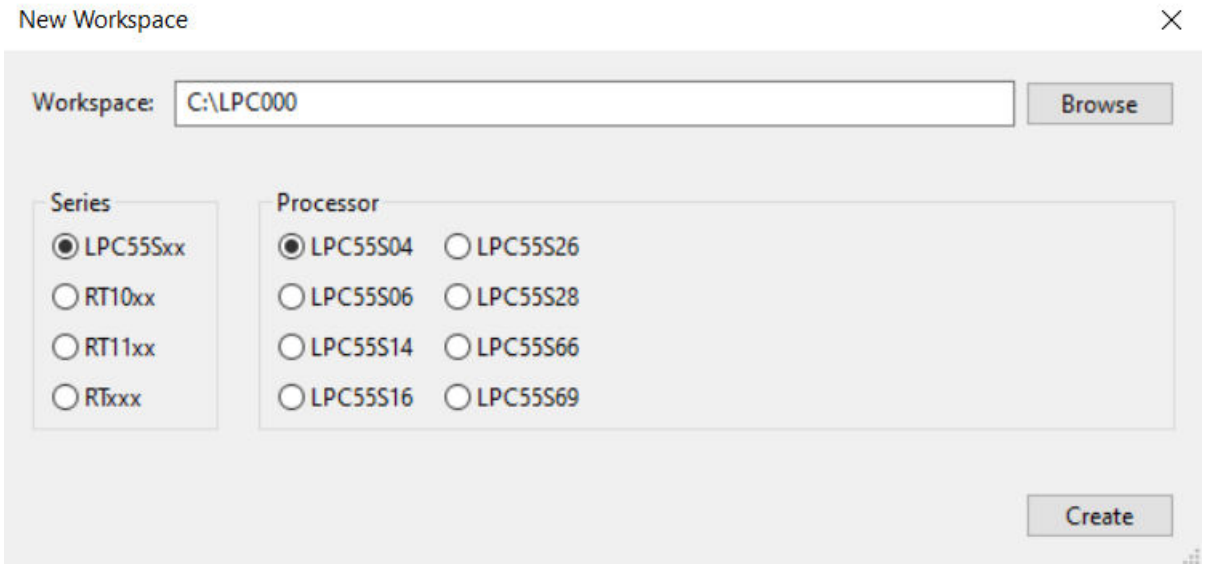

Figure 19. Creating a new workspace

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

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

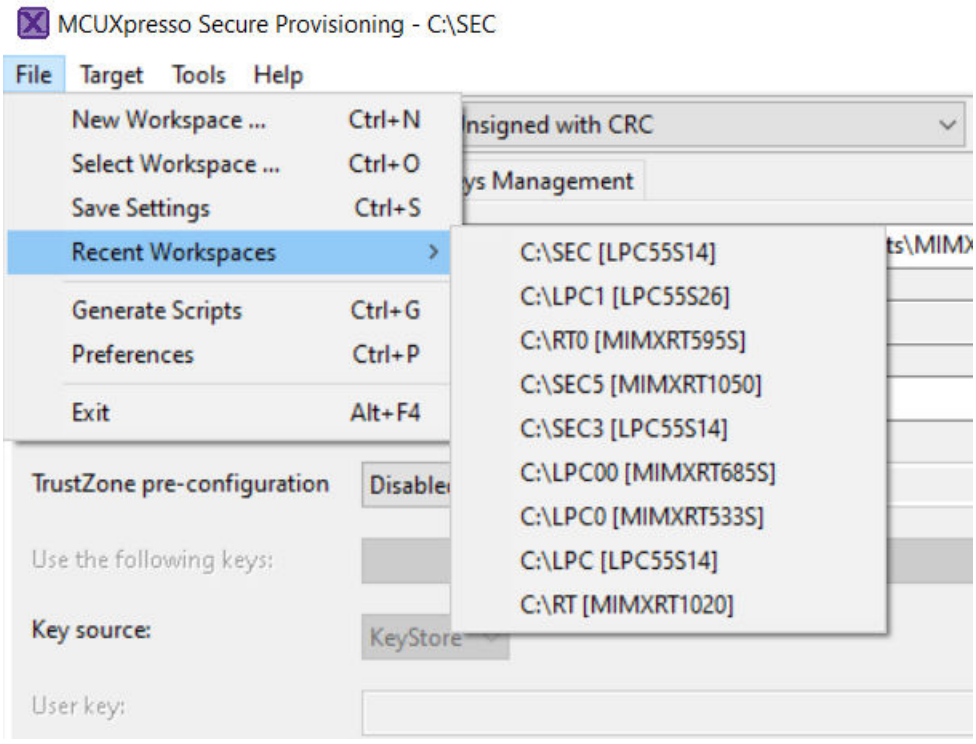

Figure 20. Selecting a recent workspace

The workspace contains multiple sub-folders, some specific to device families:

root folder	Contains the following:

	• Last configuration of the tool in file spt_settings.json. • 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.
bd_files	Generated command files used by elftosb during bootable image generation step (elftosb input).
backups	Backup of old keys/crts after importing or generating new ones
bootable_images	Intermediate and final bootable images (elftosb output). The nopadding binary starts at the address IVT, while the regular binary includes everything from the beginning of the boot device.
configs (RT10xx, RT11xx)	Configuration files - OTFAD config file (JSON), BEE user keys config file (JSON)
crts, keys	Generated certificates and their corresponding keys.
gen_bee_encrypt (RT10xx)	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_encrypt (RT10xx, RTxxx)	DEK key files generated by CST tool. The DEK key file is used during write image to generate key blob for Encrypted HAB boot type.
dcd_files (RT10xx, RT11xx)	DCD files included in the build image step.
gen_hab_certs (RT10xx, RT11xx)	Output Super Root Key table and hash (srktool output). The table is programmed along with the bootable image. The hash is programmed in platform fuses.
trustzone_files (LPC55Sxx, RTxxx)	TrustZone-M configuration files (JSON or BIN) used by elftosb during bootable image generation step (elftosb input).
gen_sb (LPC55Sxx, RTxxx)	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.

6.1.3.1 Sharing and Copying Workspaces

It's recommended to store all used files in the workspace. The "spt_settings.json" file contains all paths relative to the workspace root folder, so if you open settings on another computer, you can still re-generate all scripts.

In case the script needs to be executed on another computer without re-generation, it uses environment variables to specify SEC installation directory and workspace. These environment variables can be specified externally, or if not specified, the default value will be used.

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

6.1.4 Toolbar

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

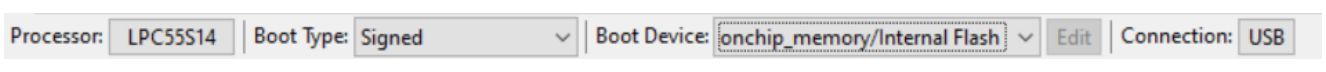

Figure 21. Toolbar

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 new workspace.
Boot Type	Choose the type of secure boot. The list depends on device capabilities of currently selected processor.
Boot Device	Choose the target memory where the device will boot from. Use the **Edit** button to open the **Boot Device Configuration** dialog. For more infomation, see Boot Device Configuration.
Connection	Choose the connection to the target. This release supports UART and USB-HID connectivity. Click the button to customize connection details. For more information, see Connection.

6.1.5 Connection

The **Connection** dialog allows you to select the connection type and test it.

The dialog is accessible from **Target > Connection** from the **Menu bar** or from the **Toolbar**.

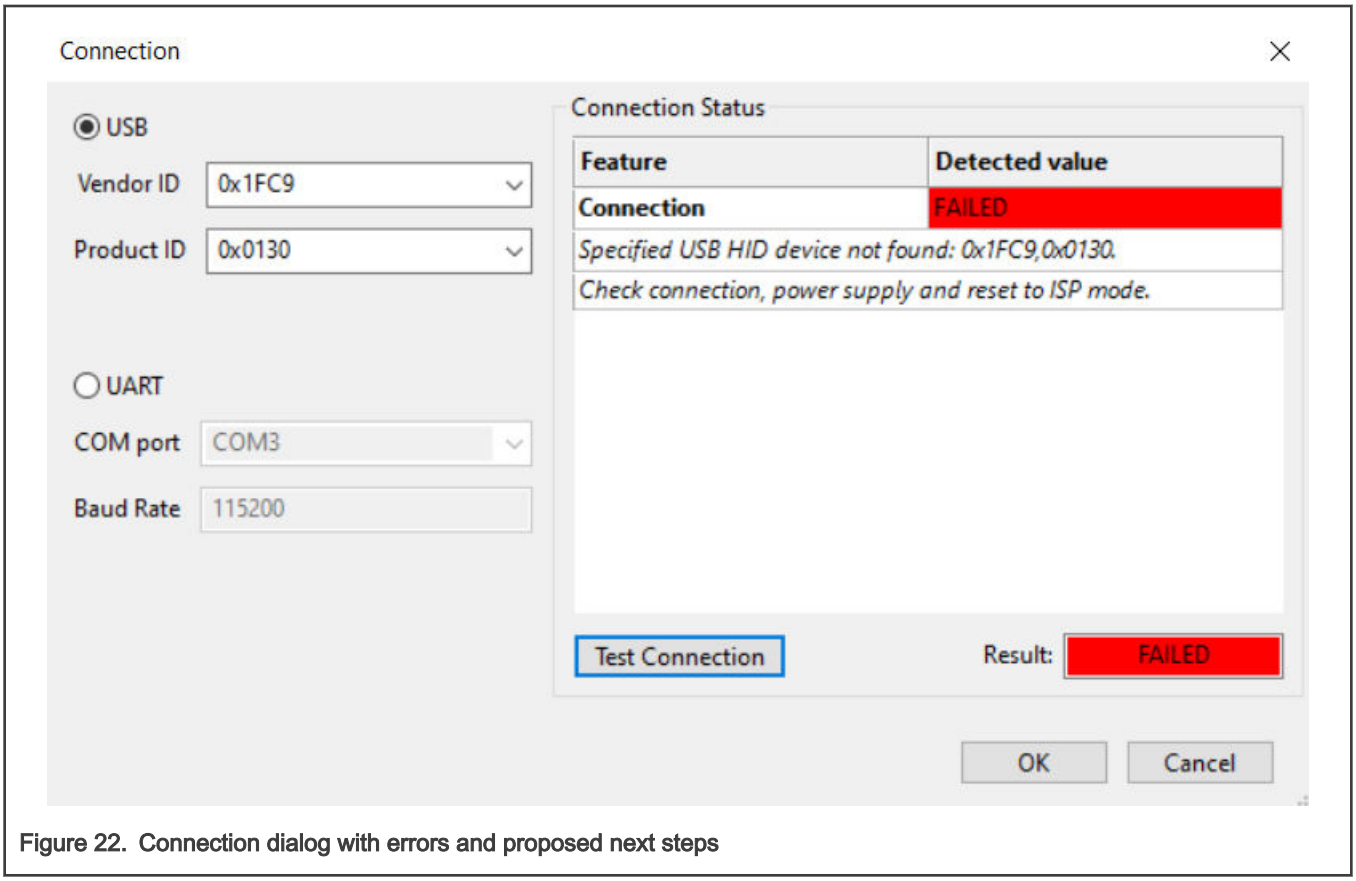

Figure 22. Connection dialog with errors and proposed next steps

It contains the following options:

USB	Specify USB connectivity to the specified Vendor ID/Product ID pair.
UART	Specify UART connectivity through the specified port and baud rate.

Use the **Test Connection** function to quickly verify that the device can be properly accessed with given configuration. To ensure a 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 through the USB OTG port or UART

Connection dialog detects following parameters:

Connection	Status of the selection communication device (USB or serial port).
Mode	Communication mode bootloader or flashloader aplication.
Processor	*Match* if connected processor matches the selected one, *No match* otherwise.
Security	Security level which was detected in the connected processor. For RT10xx/RT11xx it reports if HAB is enabled. For RTxxx it reports if SECURE_BOOT is enabled (in fuses) or DCFG_CC_SOCU enabled. For LPC55Sxx it reports if PFR is sealed.

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, SEC tries to detect additional processors connected to the computer. This may help to find a wrong SEC configuration, such as 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.

6.1.6 Boot Device Configuration

The **Boot Device Configuration** dialog contains initialization parameters. Content of the dialog depends on the selected boot device.

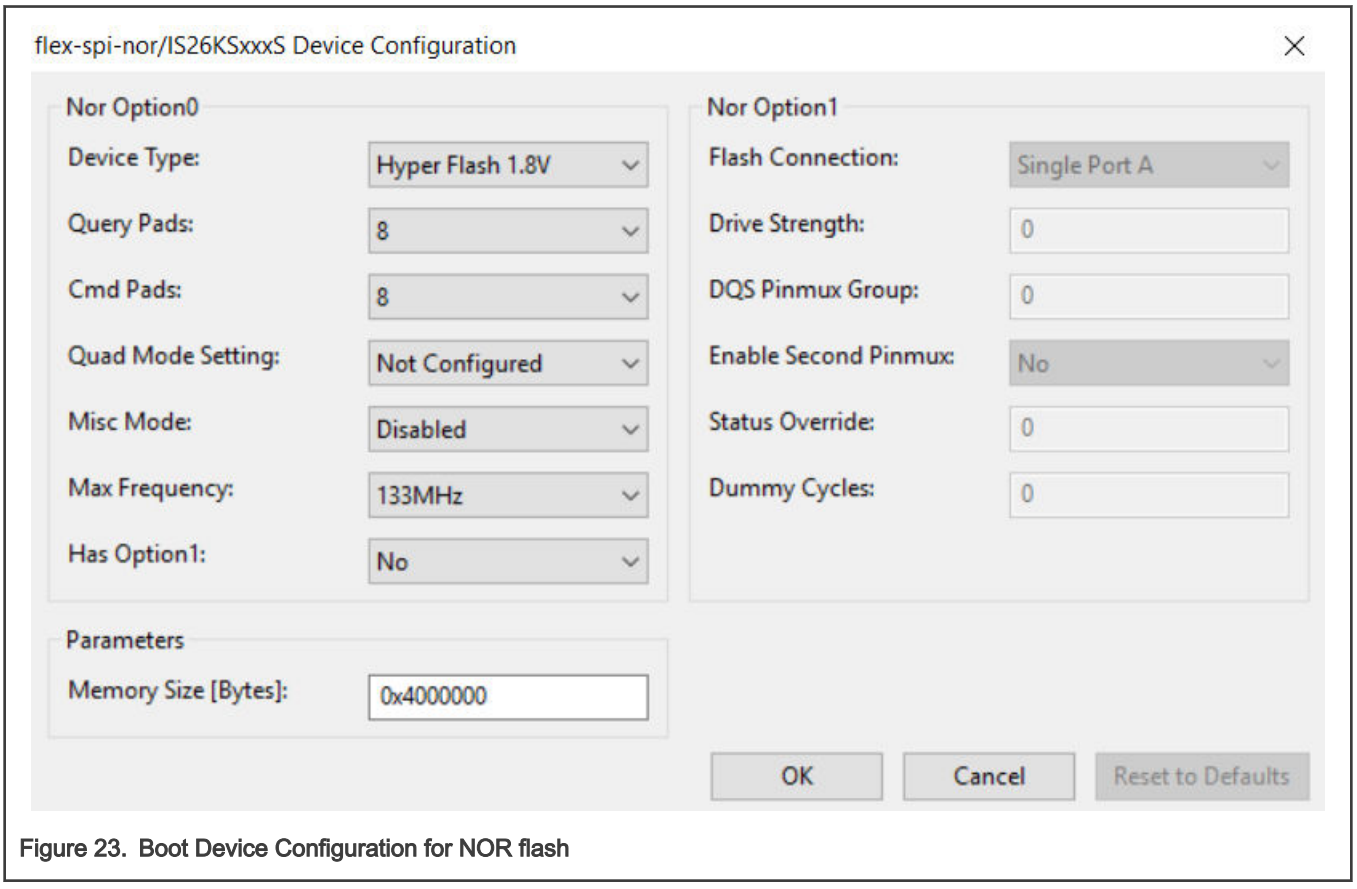

Figure 23. Boot Device Configuration for NOR flash

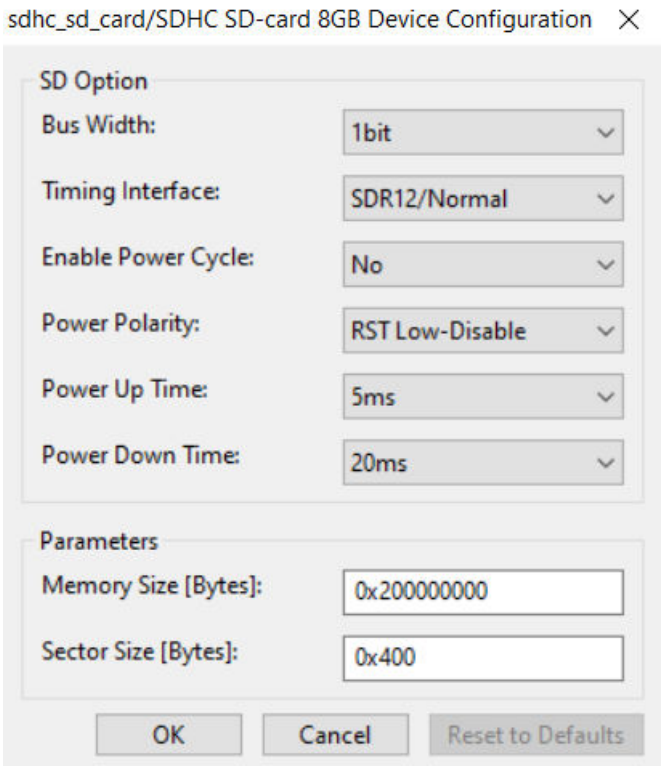

Figure 24. Boot Device Configuration for SD card

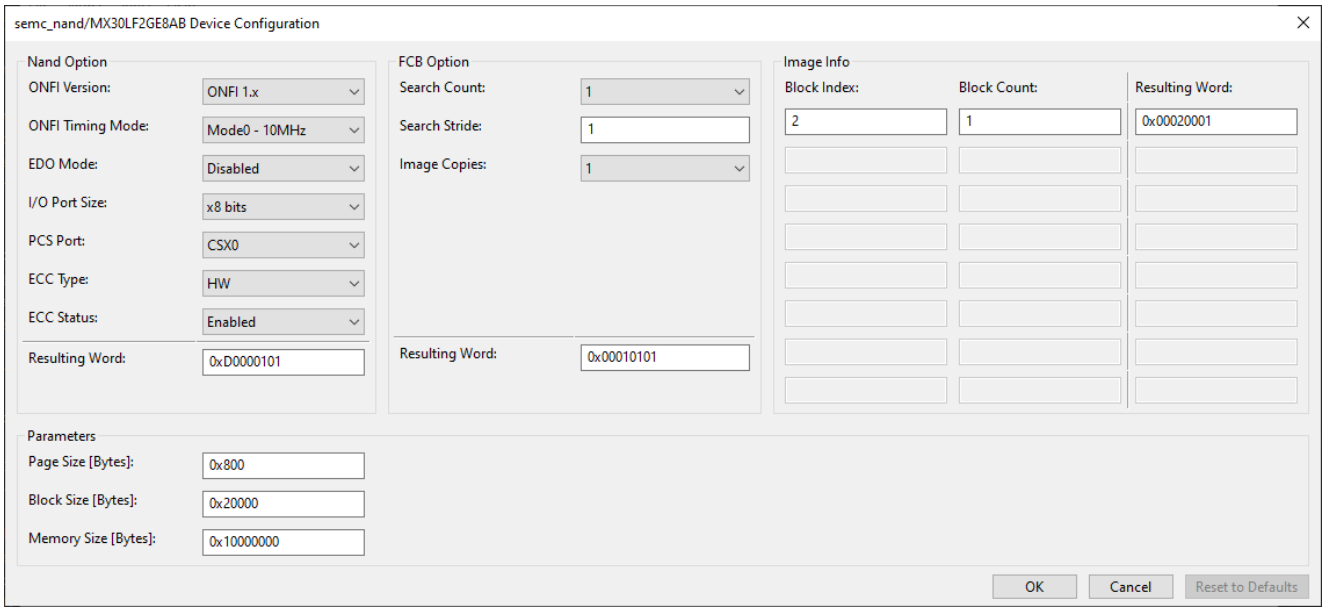

Figure 25. Boot Device Configuration for SEMC NAND (RT11xx)

NOTE

Configuration of SEMC/NAND boot device can result in fuses burn. In such case the burn fuses confirmation dialog is shown during the write image operation.

The tool includes preconfigured templates for NXP evaluation boards. They can be selected from the **Toolbar** for quick access.

6.2 Build Image

In the **Build image** view, you can transform a development image into a bootable format compatible with the device of choice. Certain options are device-family specific.

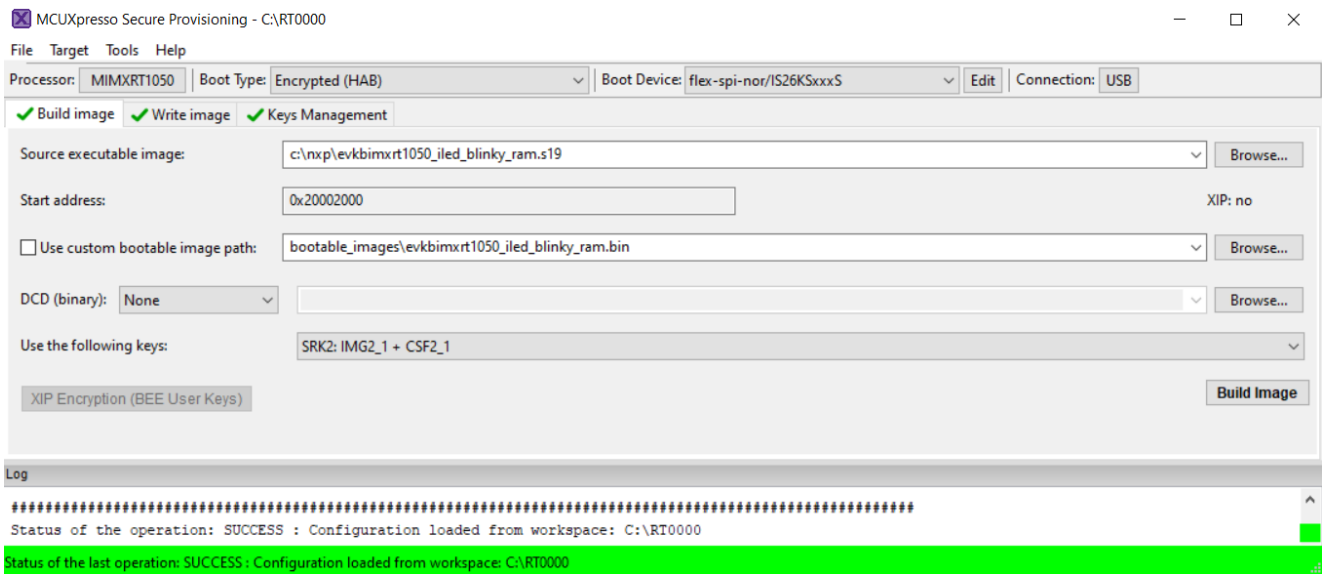

Figure 26. Build image (RT10xx)

Source executable image *(all devices)*
Choose the input executable file. The format can be ELF (compatible with MCUXpresso IDE and IAR), S-Record or plain binary or HEX file. For Keil µVision, it's recommended to use HEX format, because MDK ELF format without debug symbols can't be used. For RT10xx only, you can also use a bootable image with header.

Start address *(all devices)*
Base address of the image. Applicable only to binary image. For ELF and S-Record and HEX file, this is detected automatically.

XIP: yes/no
Whether the selected source image is built as "eXecuted In Place", and will be executed from boot device, 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 memory address of selected processor, so the result might not be correct if selected image does not match selected processor.

Use custom bootable image path *(all devices)*
Name of the generated bootable image file and its location. If not specified, the tool will name the image based on the input. File extension is specific for processor and boot type, it's either BIN for bootable images or SB for SB capsules.

DCD (Binary) *(RT10xx, RT11xx)*
Selection, what Device Configuration Data shall be included in the bootable image. Option **From Source Image** can be used only if source image contains DCD. The DCD enables early configuration of the platform including SDRAM. MCUXpresso Config Tools can generate a DCD in a compatible format. If the target processor does not support DCD files, the checkbox is disabled. For more information, see Creating/Customizing DCD files.

TrustZone pre-configuration *(LPC55Sxx, RTxxx)*
Allows you to enable TrustZone features. Following selection is possible:

- **Disabled** - Default. Disables TrustZone.
- **Enabled (preset)** - Enables TrustZone with preset data.
- **Enabled (custom data)** - Enables TrustZone with custom TrustZone-M data loaded from specified location. Overwrites preset data. JSON and BIN files supported. JSON data can be

	generated in and exported from the TEE tool of MCUXpresso Config Tools. BIN file is created by elftosb utility. For more information, see TrustZone Pre-configuration File.
Use the following keys ***(all devices)***	Sign and optionally encrypt the image with the specified root of trust key chain (SRK1..4) (RT10xx/ RT11xx) or ROT1..4: IMG (LPC55Sxx, RTxxx). This option is only applicable to authenticated and encrypted boot modes and offers selection of keys generated in the **Keys Management** view.
XIP Encryption (BEE User Keys) ***(RT10xx)***	Open 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.
OTFAD Encryption (RT11xx, RTxxx)	Open the configuration dialog of OTFAD Encryption. Option enabled only for OTFAD Encrypted boot types.
Key source ***(LPC55Sxx, RTxxx)***	Key source for signing the image. LPC55Sxx devices limited to KeyStore.
User key ***(LPC55Sxx, RTxxx)***	For OTP key source master key used to derive other keys. For KeyStore user key used to sign the image. Only available for Signed boot types.
SBKEK ***(LPC55Sxx, RTxxx)***	Key used as key encryption key to handle SB2 file. Only enabled when Key source is KeyStore. Only available for secured boot types. For LPC55Sxx devices, the key store is initialized only once in device life cycle and after that any change in SBKEK will cause failure to load SB file into the processor. For more information, see Key Store.
PRINCE Regions ***(LPC55Sxx)***	Open configuration window for encrypted PRINCE regions.

6.2.1 Source Image Formats

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

By default, the source image may not contain any boot header. For RT10xx, bootable image can be used too; such image is parsed and if contains DCD or FCB sections, these parts can be reused to build new bootable image. Once bootable image is selected and parser accepts the image, the tool offers to reuse specific parts and if confirmed, the configuration is updated.

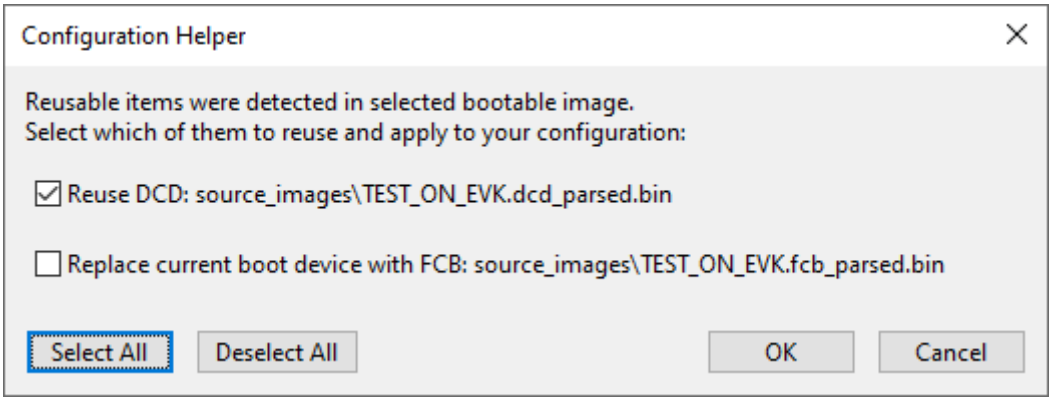

Figure 27. Configuration Helper

6.2.2 TrustZone Pre-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 and RTxxx 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).
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 futher modifications of TrustZone pre-configuration. Device-specific template files are provided in the data\targets\LPC55S##\ sub-folder.

NOTE

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

Once 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 LPC-specific workspace. Alternatively, create a new one by selecting **File>New Workspace ...**.
2. In the **Build image** tab, 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.

6.3 Write Image

Use the **Write image** view to write an image into boot device and burn platform fuses in order to achieve a secure boot. Certain options are device family-specific.

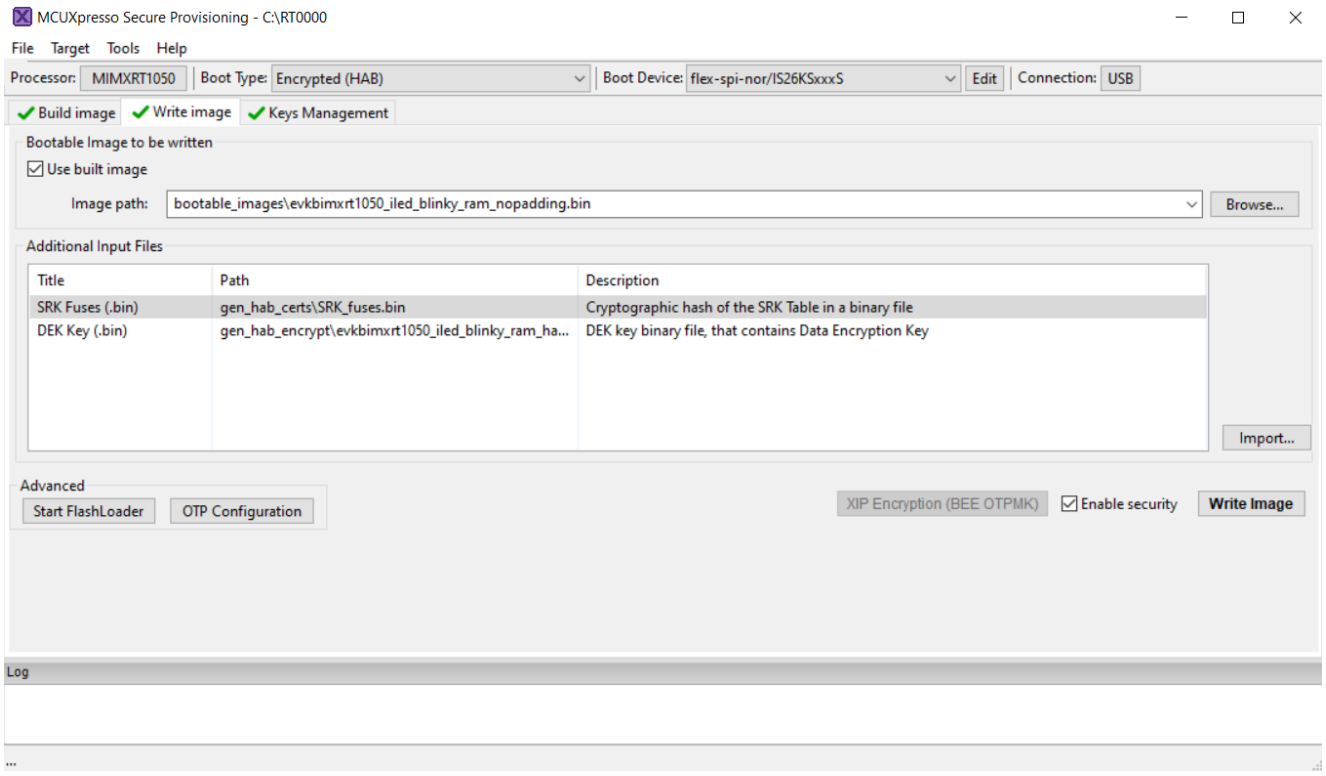

Figure 28. Write image (LPC55Sxx)

Use built image *(all devices)* If checked, the output of the Build Image operation will be used for write.

Bootable image *(all devices)* Path to the image that will be written into the target device. File extension is specific for processor and boot type, it's either BIN for bootable images or SB for SB capsules. For RT processors, the binary image must be in "nopadding" form without FCB header.

Additional Files *(all devices)* Display input files for Write Image operation. Contents depend on processor, boot type, and other build options. By default, contents are output files of the Build Image operation. You can manually replace each file with a custom file using the **Import** button.

XIP Encryption (BEE OTPMK) *(BEE OTPMK boot type, RT10xx)* Open configuration window of BEE OTP Master Key.

Enable security *(all devices)* Check to enable that the security configuration fuse will be burned during programming of the boot image. Once this is done the board will only accept authenticated images when booting from internal memory. Double-check that all settings are correct before checking this box. Mutually exclusive with **Use Shadow Registers** (RTxxx only).

Use Shadow Registers *(RTxxx)* Use shadow registers of the target device instead of fuses. This checkbox is not interactive, it's selected automatically for RTxxx processors if **Enable security** is de-selected.

Start FlashLoader *(RT10xx, RT11xx)* 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 command-line.

OTP Configuration *(RT10xx, RT11xx, RTxxx)* Open the One Time Programmable Configuration dialog. For more information, see the **OTP Configuration** section.

Before clicking the **Write Image** button, ensure the board is connected and configured to the serial downloader mode. If fuses are needed for the selected boot type, the write operation first detects if fuses are already burned. Then it runs the write script with optional arguments to write the relevant fuses.

6.3.1 OTP/PFR Configuration

OTP Configuration is accessible from the **Write Image** view for RT10xx, RT11xx, and RTxxx devices. Use the **OTP (One Time Programmable) Configuration** to:

- Review OTP Configuration prepared by SEC
- Read OTP Configuration from a connected processor
- Customize OTP Configuration
- Lock the OTP Configuration

In OTP (One Time Programmable) Configuration the configurable item is called a **fuse**. In PFR (Protected Flash Region) Configuration the configurable item is called a **field**.

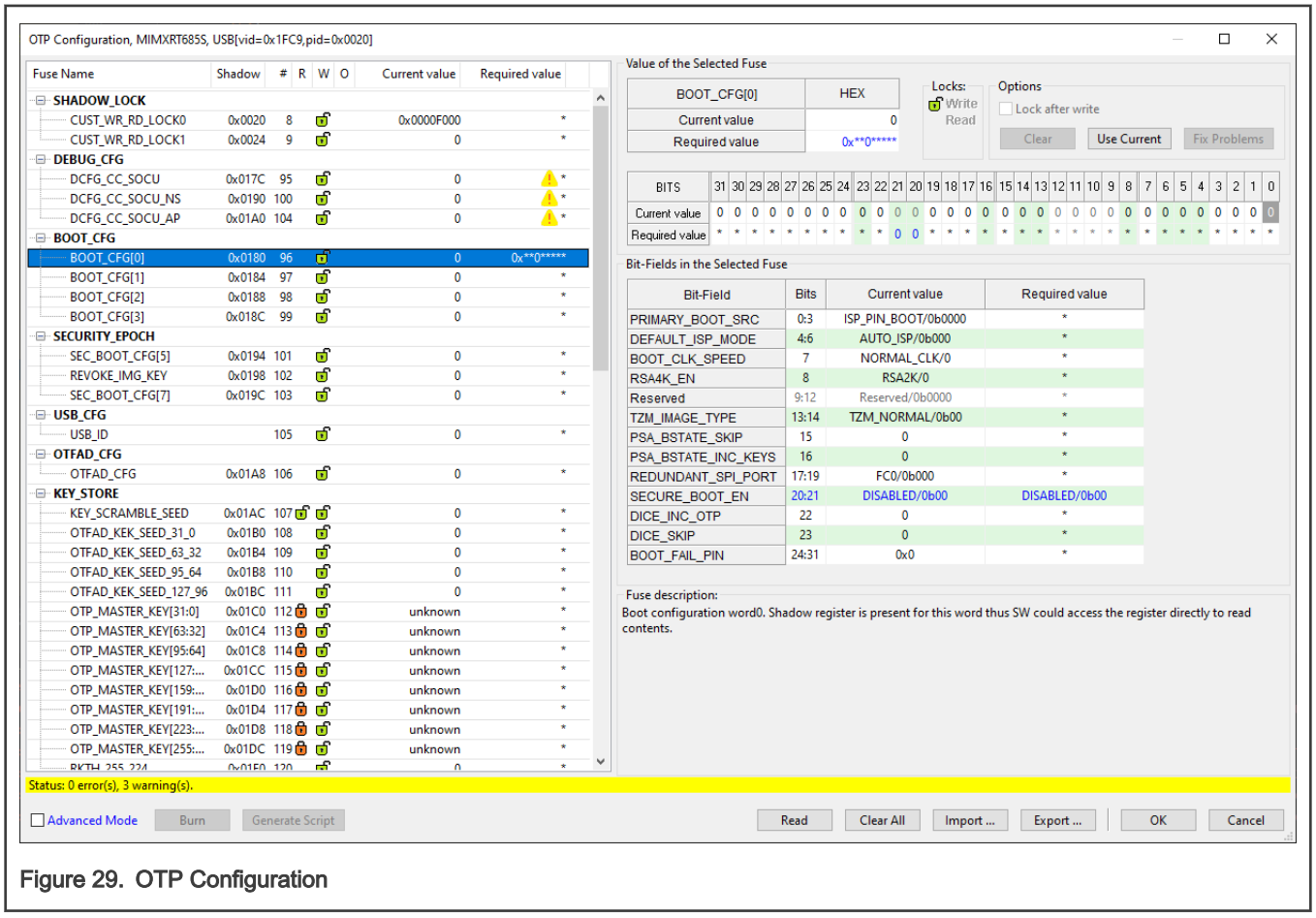

Figure 29. OTP Configuration

Primarily all changes in the OTP Configuration dialog will be applied as part of write script - with the exception of Advanced Mode.

The OTP Configuration window contains three main areas:

- Table of all fuses on the left-hand side
- Detailed information about the selected fuse on the right-hand side
- Buttons bar in the bottom of the view

The fuses table contains the list of all fuses in logical groups, with the following information organized in columns:

Fuse name	Human readable name of the fuse. Some names may not be public. See section about restricted data in Preferences.
Offset or shadow	Offset of the fuse address or offset of the shadow register address
#	Index of the fuse (parameter for blhost to access the fuse)
R/W/O	Status of read/write/operational locks read from processor. For more information, see Locks.
Current value	Current value of the fuse read from processor (hexadecimal)
Required value	Fuse value required by SEC or by the user. Values preset by SEC are highlighted in blue and can be modified only in Advanced Mode.

Following details are displayed for selected fuse:

- Current and required fuse value as hexadecimal number
- Current state of the read and write lock
- Current and required fuse value as binary number
- Current and required fuse value as bit-fields value (only if the fuse is split to bit-fields)
- Description of the selected object (group of fuses, fuse, bit-field)

Following buttons can be used for operations with the selected fuse:

Lock after write	Lock the fuse after write
Clear	Remove user requirements for the fuse
Use Current	Copy current fuse value as a new requirement
Fix Problems	Fix the displayed problems automatically

Following buttons are available in the buttons bar:

Advanced Mode	See Advanced Mode.
Burn	Burn the fuses on the connected device. Only enabled in Advanced Mode. **NOTE** Use with caution. Changes are irreversible.
Generate Script	Generates script to burn specific values. Expected to be used by advanced users only. By default, all fuses are written by the write script. Only enabled in Advanced Mode.
Read	Reads lock status and current values for all fuses from the connected target device.
Clear All	Removes all user requirements
Import ...	Imports a previously exported configuration in JSON file format.
Export ...	Exports current configuration as a JSON file.
OK	Accept changes and return to **Write Image**.
Cancel	Close the dialog without accepting changes.

6.3.1.1 Tree of All Items

PFR Configuration supports two pages: CFPA (Customer In-field Programmable Area) and CMPA (Customer Manufacturing/ Factory Programmable Area) and each page represents separate list of fields. In OTP Configuration all fuses are displayed in single tree. The items in the tree are organized in logical groups. The tree of all items is displayed in form of table, with the following columns:

Name	Human readable name of the item. Some names may not be public. See 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 processor. For more information, see Locks.
Current value	Current value of the item read from processor (hexadecimal)
Required value	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	Default value of the item (hexadecimal)

Some columns might not be displayed if the information is not applicable for the configuration.

6.3.1.2 Item Editor

Following details are displayed for selected item:

- Table with current and required item value as hexadecimal number
- Current state of the read and write lock
- Table with current and required item value as 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, item, bit-field)

6.3.1.3 Buttons

Following buttons can be used for operations with the selected item:

Lock after write	Lock the item after write.
Clear	Remove user requirements for the item.
Use Current	Copy current item value as a new requirement
Fix Problems	Fix the displayed problems automatically

Following buttons are available in the buttons 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 irrevertible.

Generate Script	Generate script to write required values. Expected to be used by advanced users only. By default, all fuses 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.
Clear All	Remove all user requirements.
Import ...	Import a previously exported configuration in JSON file format.
Export ...	Export current configuration as a JSON file.
OK	Accept changes and return to **Write Image**.
Cancel	Close the dialog without accepting changes.

6.3.1.4 Tree-Filtering Toolbar

You can filter items displayed in the tree. The following filters push buttons are supported in toolbar above the tree:

- **Filter ON/OFF** – First button can be used to turn off all filters (for example display all items) or turn on default filter.
- **Editable** – If the filter is active, editable items are displayed, for example items that are not reserved nor read-only nor preset.
- **Required** - If the filter is active, items with any user required value are displayed.
- **Preset** - If the filter is active, items with any preset value are displayed.
- **UnLocked** - If the filter is active, items that are not locked are displayed.
- **WLocked** - If the filter is active, items that are locked for write are displayed.
- **Problems** - If the filter is active, items with any warning or error are displayed.

The filters can be combined, so for example, you can select **Required** + **Preset** together.

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

6.3.1.5 Read from Connected Device

Before you start the OTP/PFR Configuration tool, it's recommended to check in the **Connection** dialog if the board is connected to the host. If the target device requires flashloader (RT10xx and RT11xx), it's 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 contain 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 next chapter for details)

6.3.1.6 Required Value

Items that need to be written based on selected configuration (for example contains preset value), are highlighted in blue. Remaining items, by default, either do not have any required value – displayed as * (star) OR if default value is applied, they are displayed as *<value> (for example *0). Required value for these items can be specified:

- As a 32-bit hexadecimal number
- By bits. Click on the value in the second row of **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's specified as decimal or hexadecimal number (with **0x** prefix).

6.3.1.7 Locks

Locks are available only in OTP Configuration to lock fuses. A lock block specify access restriction to the fuse - usually read or write restriction. The locks should 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.

Two type 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's possible to apply write-lock for 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.

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

Following icons are to represent the lock status:

Table 5. Lock icons

Icon	Description
No icon	Fuse doesn't 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
- Fuse must be locked after 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 **Required value** column in the fuses table:

On RT11xx 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.

6.3.1.8 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 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 source value for calculated item is not specified. See the following chapter for details about validations and problem resolutions.

6.3.1.9 Validation and Problem Resolution

Configuration provides validation of the required values and highlights the following problems:

Errors (highlighted in red)

- Required value cannot be written, in conflict with the current value
- Required value cannot be written, because there is a write-lock applied for the fuse and required value does not match with the current value

- Required value for calculated item does not match with the value calculated by SEC.
- Processor-specific validation
- Required value cannot be written, it is in conflict with the current value.
- Required value cannot be written, because there is a write-lock applied for the fuse and required value does not match with the current value.
- Required value for calculated item does not match with 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 detected, it is reported on 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
- Calculated value does not match with current value in the processor (this is reported only if there is no required value for the calculated value; otherwise error is reported for required value)

The problems are displayed in the tree and, if the item is selected, also in the details section, in all editors.

In the BITS editor, the problem is displayed only for bits, that are affected by the problem. This allows to fix the problem easily by clicking the affecting bit value (for example, inverting the required value of the bit).

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

6.3.1.10 Advanced Mode

By default, it's recommended to apply the modified configuration into a workspace settings file and the Write script, so it's 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 stand-alone usage of the Configuration tool and allows you to:

- Write a required value directly to the connected processor
- Generate the script to write required values
- Modify all required values, even those preset by SEC

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

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

6.3.1.11 Write/Burn

The Write/Burn operation will burn 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 fuses operation is irreversible. It's strongly recommended to:

- Double-check all values being burned
- Double-check all items being locked
- Double-check all problems reported by OTP 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 fuse is locked.
- Generate Script is expected to be used on empty processor. It contains configuration of all fuses.

6.3.1.12 PFR and OTP Differences

- PFR Configuration contains two pages: CFPA and CMPA.
- Items in OTP Configuration are called “fuses” while items in PFR Configuration are called “fields”.
- Fuses in OTP Configuration are burned item by item, so you can specify a single requirement only. PFR always updates the whole page, so if no requirement is specified, default value will be used.
- Locks are supported only for fuses.

6.4 Manufacturing Tool

Use the **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. The user interface is made of these areas: **Operation**, **Command**, **Connected Devices**, **Options**, and the button bar.

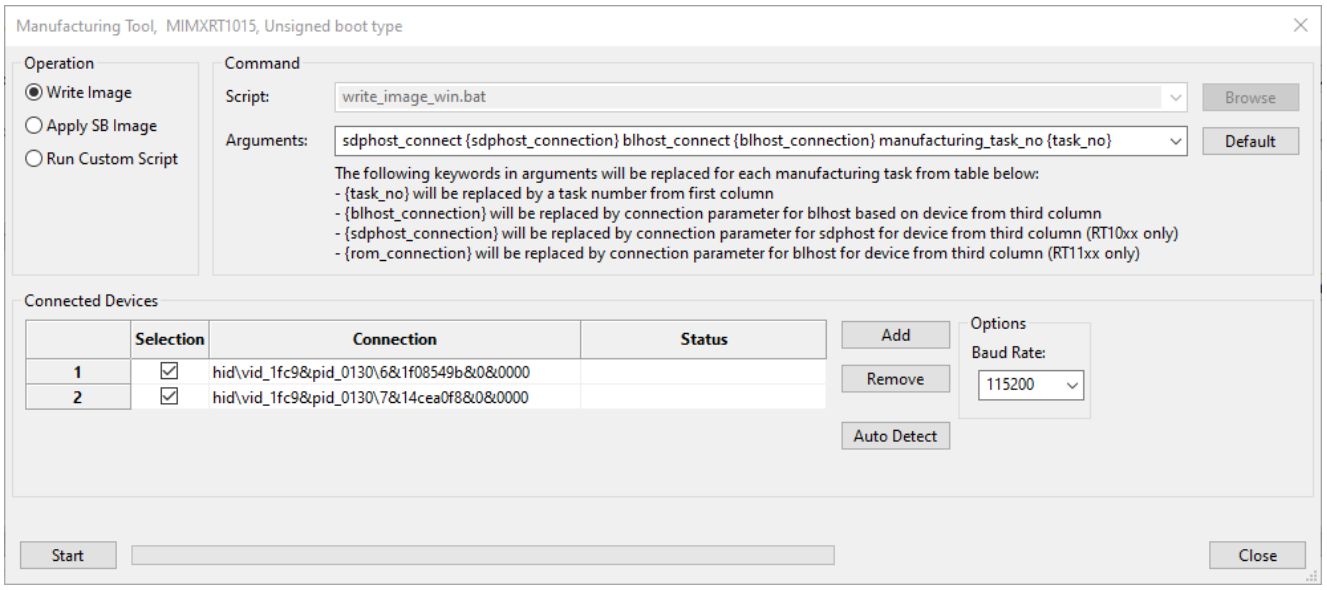

Figure 30. Manufacturing Tool

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 **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* sub-folder of the workspace by default. For RT10xx/RT11xx processors, the SB file must be created manually, as it's currently not supported by SEC.
Run Custom Script	Uses a custom script to configure provisioning. It's assumed the script is a modified SEC write script accepting SEC parameters (especially connection parameters). Exit status 0 is considered a success.

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

Script (Write Image, Run Custom Script)	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. Note that the following keywords in arguments will be replaced for each manufacturing task: • {task_no} will be replaced by index of the task • {blhost_connection} will be replaced by connection parameters for blhost • {sdphost_connection} will be replaced by connection parameters for sdphost (RT10xx only) • {rom_connection} will be replaced by connection parameter for blhost for device from third column (RT11xx only)

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're in ISP mode.

- To automatically detect all devices connected to the host using USB or UART, use the **Auto-detect** button. This is the recommended usage. For more information about the USB path, see the here.
- 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 auto-detection 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.

Options The Options contains additional operation options.

Baud rate	Use the drop-down list to select the baud rate of the operation.
Load Key-Store (Apply SB Image for RTxxx)	Load KeyStore from external flash before uploading the SB file.

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.
Close	Closes the Manufacturing Tool without running the operation.

6.4.1 USB Path

The Manufacturing Tool supports the use of either Serial or USB connection. In most cases the Auto-detect function will be sufficient for detecting the connection. The USB path, however, differs depending on operating system. The path is passed between several scripts, so it should not contain spaces.

Following is the description how the USB path is constructed on each operating system:

Windows
Using Windows Device Manager:

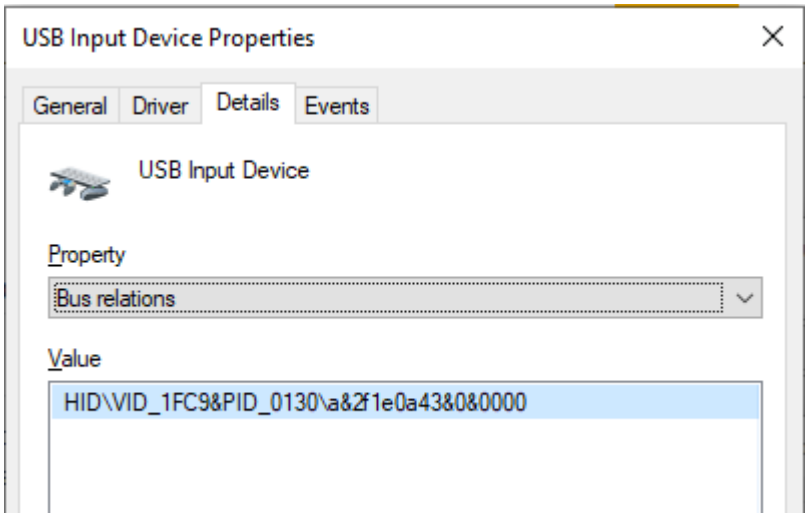

Figure 31. Windows Device Manager

Linux

Using lsusb utility the Bus and Device numbers can be retrieved and used in decimal format without leading zeros separated with *#* (hash tag)

```
$ lsusb | grep NXP
Bus 003 Device 006: ID 1fc9:0130 NXP Semiconductors
```

In the above example the USB path is *3#6*.

Mac OS

Using ioreg utility, see the *SP Blank RT Family @14600000*:

```
    | |   |   +-o HS07@14600000  <class AppleUSB20XHCIPort, id 0x1000002e1, registered,
matched, active, busy 0 (3569 ms), retain 15>
    | |   |   | +-o SP Blank RT Family @14600000  <class IOUSBHostDevice, id 0x1000008ad,
registered, matched, active, busy 0 (1335 ms), retain 30>
    | |   |   |   +-o AppleUSBHostLegacyClient  <class AppleUSBHostLegacyClient, id
0x1000008b0, !registered, !matched, active, busy 0, retain 8>
    | |   |   |   +-o AppleUSBHostCompositeDevice  <class AppleUSBHostCompositeDevice, id
0x1000008b9, !registered, !matched, active, busy 0, retain 4>
    | |   |   |   +-o IOUSBHostInterface@0  <class IOUSBHostInterface, id 0x1000008bc,
registered, matched, active, busy 0 (64 ms), retain 12>
    | |   |   |     +-o AppleUserUSBHostHIDDevice  <class AppleUserHIDDevice, id
0x1000008c3, registered, matched, active, busy 0 (2 ms), retain 13>
    | |   |   |       +-o IOHIDInterface  <class IOHIDInterface, id 0x1000008c8, registered,
matched, active, busy 0 (1 ms), retain 6>
```

In the above example the USB path is the *@14600000*.

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 in order 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 board and its USB path is retrieved. To find 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 manufacuring is finished, the device flashloader is usually still active, so the device will not be detected until you have reset the processor.

6.5 Keys Management

The **Keys Management** view displays the list of certificates used to sign and optionally encrypt the image. Generated keys can be exported for later use. Keys Management is enabled for secured boot types only.

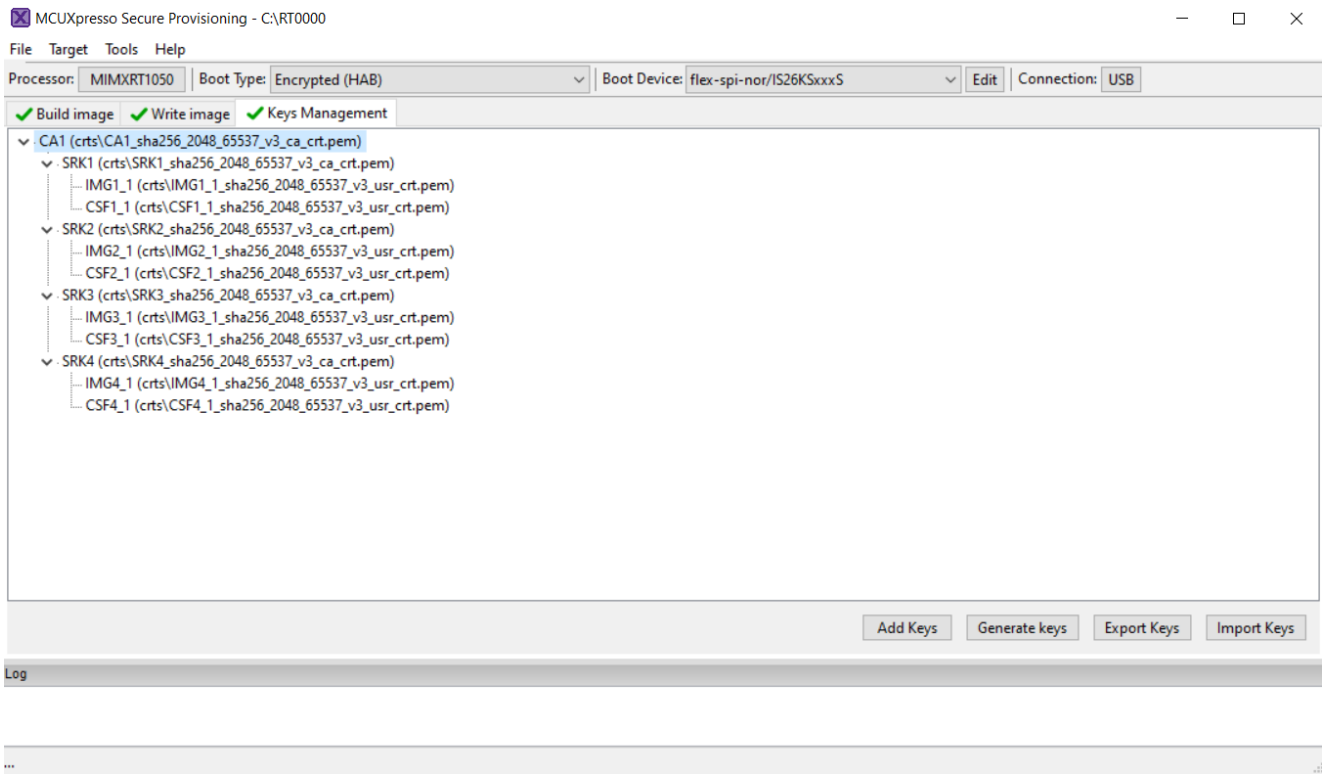

Figure 32. Keys Management (RT10xx)

6.5.1 Generate Keys

Authenticated images rely on a Public Key Infrastructure (PKI) set of certificates. PKIs are generated via the OpenSSL tool. SEC includes a graphical interface that simplifies the generation of a PKI compatible with HAB boot modes. Certain options are device-family-specific.

To access it, ensure that the **Boot Type** is set to one of the types mentioned in the previous section, and select **Generate keys**.

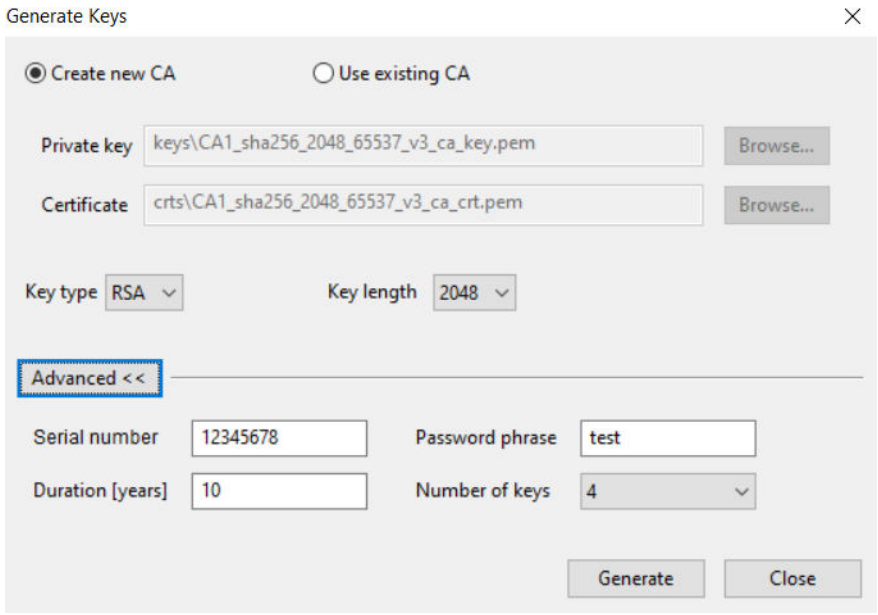

Figure 33. Generate keys (RT10xx, RT11xx)

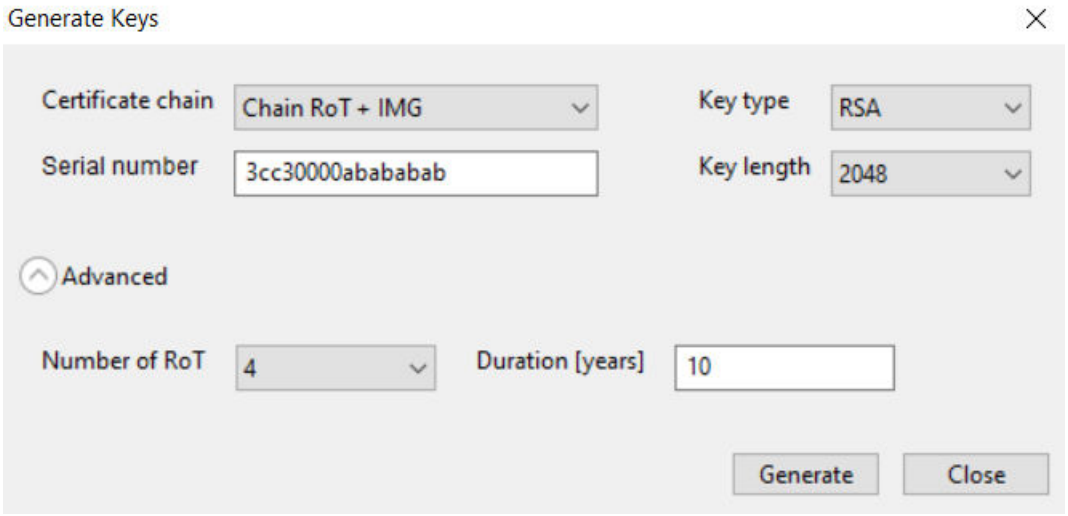

Figure 34. Generate keys (LPC55Sxx, RTxxx)

Basic options

Key Type *(all devices)*	Choose generated key type. For supported devices, RSA is the only selection.
Key Length *(all devices)*	Choose generated key length in bits.
Create new CA *(RT10xx, RT11xx)*	Use OpenSSL to generate a self-signed root Certificate Authority for development purposes.
Use existing CA *(RT10xx, RT11xx)*	Enable use of user specified CAs. **Certificate** and **Private Key** need to be in PEM format.
Serial number *(LPC55Sxx, RTxxx)*	Value used for key revocation.
Certificate chain *(LPC55Sxx, RTxxx)*	The depth of the keys.

Advanced options

Duration [years] *(all devices)*	Set certificate validity to the specified duration in years. For supported devices this is necessary for signing purposes only, as duration is not directly verified in hardware.
Number of RoT *(LPC55Sxx, RTxxx)*	Number of RoT (Root of Trust) keys to be generated. Maximum supported number of keys, as well as the recommended default, is 4.
Password phrase *(RT10xx, RT11xx)*	Secure generated CA with the specified parameter.
Serial number *(RT10xx, RT11xx)*	Set certificate serial number to the specified parameter. Must start with 3cc3 octets.
Number of keys *(RT10xx, RT11xx)*	Set to the number of SRK keys to be generated. HAB supports up to 4 keys, with a recommeded default of 4.

NOTE

Once platform fuses are programmed with the Super Root Key hash the number of SRKs can no longer be changed.

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

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

Figure 35. Generate Keys - Progress

6.5.2 Add Keys

Once keys have been generated in the **Generate Keys** dialog, you can add them to the image in the **Add Keys** dialog.

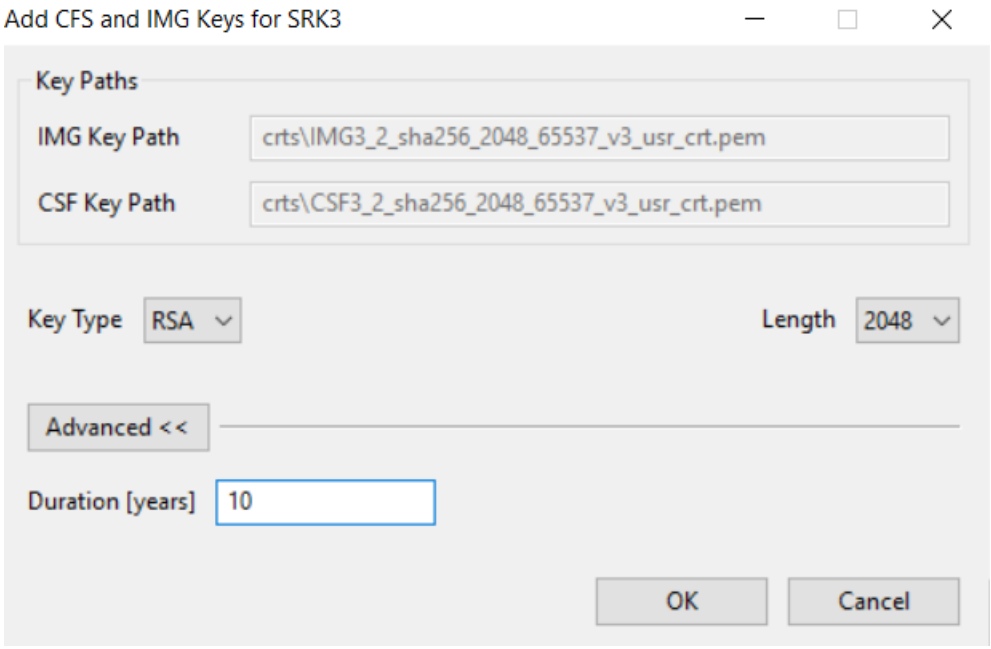

Figure 36. Add Keys (RT10xx, RT11xx)

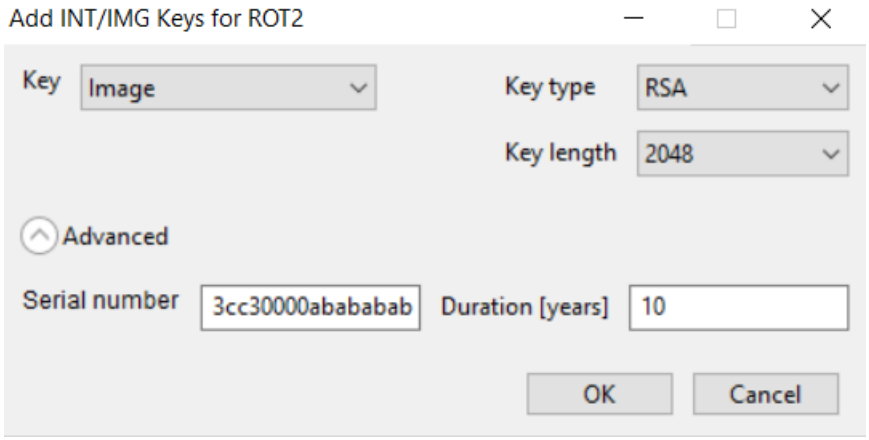

Figure 37. Add Keys (LPC55Sxx, RTxxx)

Basic options

Key Type *(all devices)*	Choose generated key type. For supported devices, RSA is the only selection.
Length *(all devices)*	Choose generated key length in bits.
IMG Key Path *(RT10xx, RT11xx)*	Path of the IMG key to be generated and added.
CSF Key Path *(RT10xx, RT11xx)*	Path of the CSF key to be generated and added.
Key *(LPC55Sxx, RTxxx)*	Type of key to be added. Choose between image key for signing, or intermediate key.

Advanced options

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

Serial number *(LPC55Sxx, RTxxx)*	Value used for key revocation. Must start with 3cc3.

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

Figure 38. Add Keys progress

6.5.3 Import/Export Keys

You can export generated keys for later use using the Export function. To export keys:

1. Select **Export Keys** in the **Keys 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 exports workspace configuration, including SBKEK for LPC and BEE User Keys for RT10xx devices for future use.

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 SBKEK for LPC, BEE User Keys for RT10xx, and OTFAD keys data for RT11xx and RTxxx devices are restored from the imported folder.

To import keys:

1. Select **Import Keys** in the **Keys Management** view.
2. In the dialog, navigate to the location of the **crts** and **keys** folders containing keys info you want to import, and select **Select Folder**.

6.6 Log

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

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

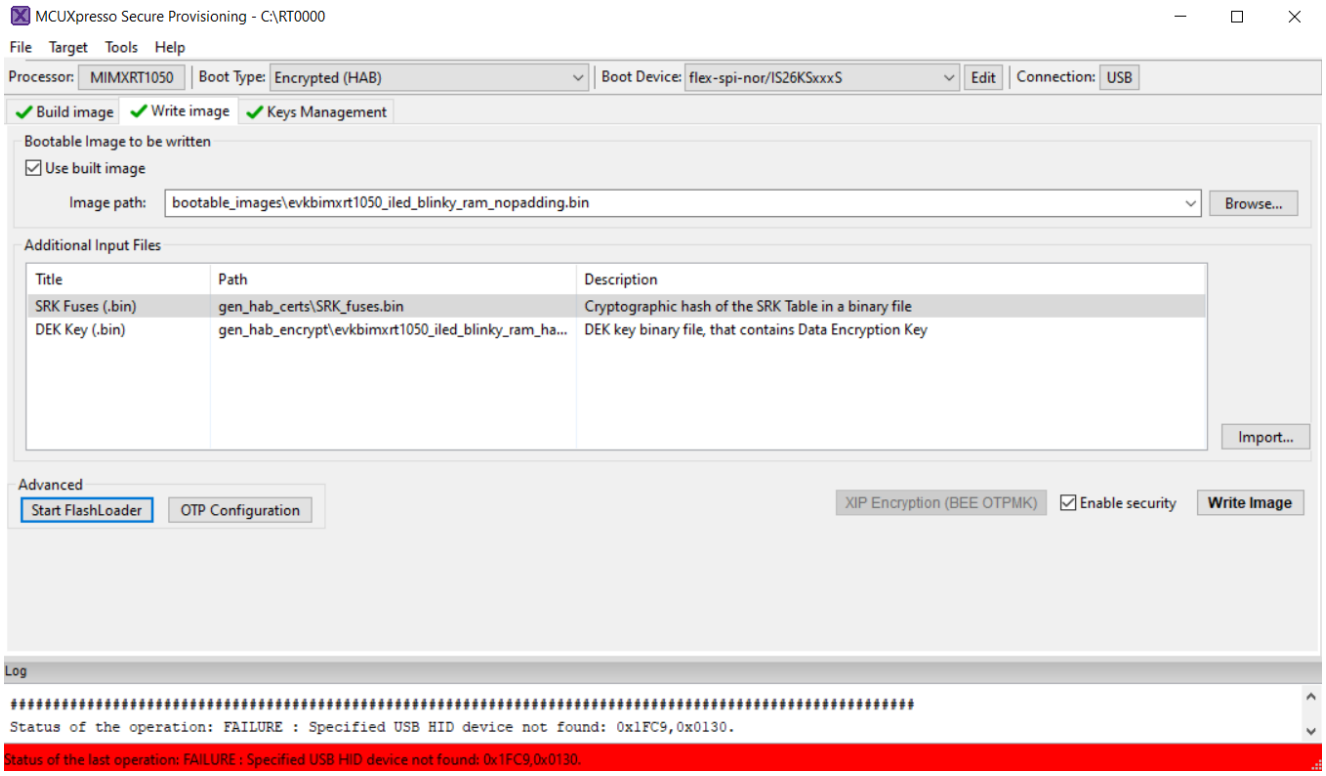

Figure 39. Log view with errors

Chapter 7
Workflow

This chapter takes you through the steps you need to take in order to successfully boot up your device to the required security level. It describes the creation of the bootable image, connecting your device, and setting up your boot preferences and writing the image into the selected boot device. Common steps will be described first, followed by device family-specific content. It's assumed the image will be executed on the NXP evaluation boards.

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 Secure Keys
- Build a plain image in the selected toolchain
- Build a bootable image by SEC
- Connect an evaluation board
- Write bootable image into the processor

7.1 Common Steps

7.1.1 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 an CMSIS pack and how to open an example project from the package. It's recommended to start with **iled_blinky** example, because it offers a simple check whether resulting application is working – LED flashes with 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 selected toolchain and download it.

  NOTE

  Starting with MCUXpresso IDE v11.1.0, you can download and install 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's 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.

7.1.2 Opening Example Project

- **MCUXpresso IDE**
  1. Drag-and-drop the downloaded MCUXpresso SDK package into Installed SDKs view to install the package.
  2. Select **File > New > Import SDK examples...**.
  3. Select your processor and board and on next page select the iled_blinky example.
- **Keil MDK 5 + Example package**

  Unpack the SDK package into the selected folder and open *boards\evkmimxrt10##\demo_apps\led_blinky\mdk\iled_blinky.uvmpw*.

  If you have downloaded a single example project only, unzip into selected folder and open the workspace file.

  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** tab, select **All Devices > NXP > MIMXRT10##**.
  3. In the **Packs** tab, ensure that the following device-specific packs are installed: *NXP::{processor}#_DFP* and *NXP::EVK-{processor}_BSP*.
  4. Select the BSP pack
  5. In **Examples** tab, copy the **iled_blinky** example project into selected folder.

     Go to **Project > Options > Output** to ensure the option **Create HEX File** is selected.
- **IAR Embedded Workbench + MCUXpresso SDK package**

  Unpack the SDK package into the selected folder and open *boards\evkmimxrt10##\demo_apps\led_blinky\iar\iled_blinky.eww*.

  If you have downloaded a single example project only, unzip into selected folder and open the workspace file.

7.1.3 Building Example Project

Detailed information about project configuration and build is described in processor specific sections below.

7.1.4 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 or UART.
4. Confirm that the connection is working by selecting **Target > Connection ...** from the **Menu bar** and clicking the **Test** button. Tweak if neccesary.

7.1.5 Preparing Secure Keys

This section describes the generation of keys necessary for the 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 **Keys Management** tab.
3. Ensure it doesn't 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's 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's recommended to back up generated keys before they are burned into fuses in the processor.

7.2 RT10xx/RT11xx Device Workflow

7.2.1 Preparing Source Image

In this step, you must select target memory where the image will be executed. Following options are available for RT10xx/ RT11xx devices:

- **Image running from external NOR FLASH**

  This is the so called **XIP**(e**X**ecution **I**n **P**lace) image, which means the image is executed directly from memory where it is located.

- **Image running in internal RAM**

  This image can be located on an SD-Card or in external FLASH (FlexSPI NOR 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 in external FLASH (FlexSPI NOR or SEMC NAND) and during boot will be copied into SDRAM and executed from there.

7.2.1.1 Image Running from External NOR Flash

NOTE

For current version of SEC, build image for images with XIP boot header is supported only for RT10xx processors. Disable XIP boot header of the image before building the image, as it is currently added to the image by SEC.

- **MCUXpresso IDE**

  The **led_blinky** example is linked into external FLASH by default.

  1. Go to **Project > Properties > C/C++ Build > Settings > MCU C Compiler > Preprocessor > Defined symbols** and set **XIP_BOOT_HEADER_ENABLE** to **0**.
  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).

3. In **Project > Options > Linker**, remove all *--keep* options and the predefined symbol **XIP_BOOT_HEADER_ENABLE**. As a result, **Misc. controls** will contain 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**.
  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*.

7.2.1.2 Image Running in Internal RAM

- **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.

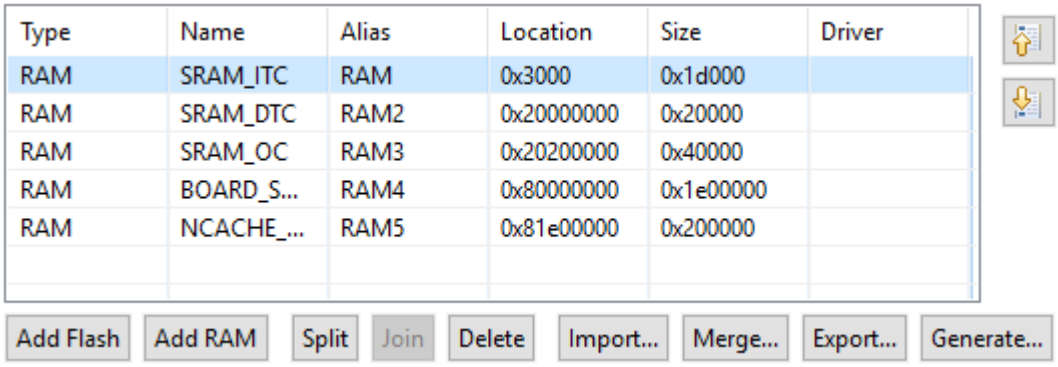

Figure 40. SRAM_ITC

  3. Move **SRAM_ITC** to first position to make it default.
  4. Build the image.

     You can find the resulting source image named as *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 0x0000 3000
#define m_interrupts_size 0x00000400
```

```
#define m_text_start 0x0000 3400
#define m_text_size 0x000 1DC00
```

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 = 0x0000 3000;
define symbol m_interrupts_end = 0x0000 33FF;
```

```
define symbol m_text_start = 0x0000 3400;
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*.

7.2.1.3 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 as *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 0x 80002000
#define m_interrupts_size 0x00000400
```

```
#define m_text_start 0x 80002400
#define m_text_size 0x000 1DC00
```

```
#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*.

7.2.2 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
- MIMXRT1050-EVKB
- MIMXRT1060-EVK
- MIMXRT1064-EVK

- MIMXRT1160-EVK
- MIMXRT1170-EVK

1. See Table 6 for instructions how to set boot mode using DIP switches.
2. Make sure you have **J1** (J38 on RT1176, RT1166) set to **3-4** to power board from USB OTG.
3. Connect to the J9 (J20 on RT1176, RT1166) port with the USB cable to your PC.
4. Ensure SEC is already running with a workspace created for the chosen device. For more information, see Setting up SEC.
5. Make sure that **Boot device** in the toolbar matches NOR FLASH used on EVK board (for example *flex-spi-nor/ISxxxx*).
6. Set 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 MCUXpresso Secure Provisioning Tool, select **Boot Device: sdhc_sd_card/SDHC SD card 8GB** in the **Toolbar**.

Table 6. DIP Switches: Boot mode selection for EVKs

Boot mode/Device	Serial bootloader (ISP mode)	Flex-SPI NOR (QSPI, HyperFlash)	Flex-SPI NOR + Encrypted XIP (BEE or OTFAD)	SD card	SEMC NAND
RT1010-EVK	SW8: 0001	SW8: 0010	N/A	N/A	N/A
RT1015-EVK			SW8: 1010		
RT1020-EVK				SW8: 0110	
RT1024-EVK					
RT1050-EVKB	SW7: 0001	SW7: 0110	SW7: 0010 or 0110	SW7: 1010	N/A
RT1060-EVK		SW7: 0010	SW5: 1000		
RT1064-EVK					
RT1160-EVK	SW1: 0001	SW1: 0010	SW1: 0010	SW1: 0010	SW1: 0010
RT1170-EVK	SW2: 0000000000	SW2: 0000000000	SW2: 0100000000	SW2: 0000001000	SW2: 0000010000

7.2.3 Booting Images

This section describes the building and writing of bootable images.

See the flowcharts for visual representation of the process.

You can use several combinations of used memories:

Table 7. Booting image

Memory where the image is executed	Memory where the image is written	DCD needed	XIP
External NOR FLASH	External NOR FLASH	No	Yes
Internal RAM	External NOR or NAND FLASH	No	No

Table continues on the next page...

Table 7. Booting image (continued)

Memory where the image is executed	Memory where the image is written	DCD needed	XIP
Internal RAM	SD Card	No	No
SDRAM	External NOR or NAND FLASH	Yes	No
SDRAM	SD Card	Yes	No

NOTE

- **Memory, where image is executed** - Explained in Preparing Source Image.
- **Memory, where image is written** - Configured as **Boot Device** in SEC.

7.2.3.1 Booting Unsigned Image

Unsigned image is typically used for development. It's recommended to start with this boot type before working with secured images to verify that the executable image works properly.

See the RT10xx and RT11xx flowcharts for visual representation of the proces.

First, build a bootable image:

1. Make sure you have selected **Unsigned Boot Type** in the **Toolbar**.
2. Switch to the **Build image** view.
3. Select image build in Preparing Source Image as a **Source executable image**.
4. For images executed from SDRAM, configure SDRAM using a DCD file. For EVK boards, the following DCD file can be used: *data\targets\MIMXRT1###/evkmimxrt1xxx_SDRAM_dcd.bin*.

NOTE

For customization of DCD files, refer to Creating/Customizing DCD files.

5. Click **Build image** to build a bootable image.

When the bootable image has been successfully built:

1. Make sure 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 **Write image**.

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

7.2.3.2 Booting Authenticated 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.

See the RT10xx and RT11xx flowcharts for visual representation of the proces.

1. In the **Toolbar** set **Boot Type** to **Authenticated (HAB)**.
2. In the **Build Image** view, use the image from Preparing Source Image as a **Source executable image**.
3. For **Use the following keys** select any key, for example *SRK1: IMG1_1+CSF1_1*.
4. Click **Build image**.

5. Check that the bootable image was built successfully.

1. To write the image, switch to **Write image** view.
2. Make sure the board is set to Serial bootloader (ISP) mode. See Table 6 for more information.
3. Make sure the **Use built image** checkbox is selected.
4. Select the **Enable security** checkbox.
5. Click **Write Image**.
6. 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 processor can only execute authenticated images.

   - **No** - Do not burn fuses, continue writing the image.
   - **Cancel** - Abort writing the image and burning fuses.

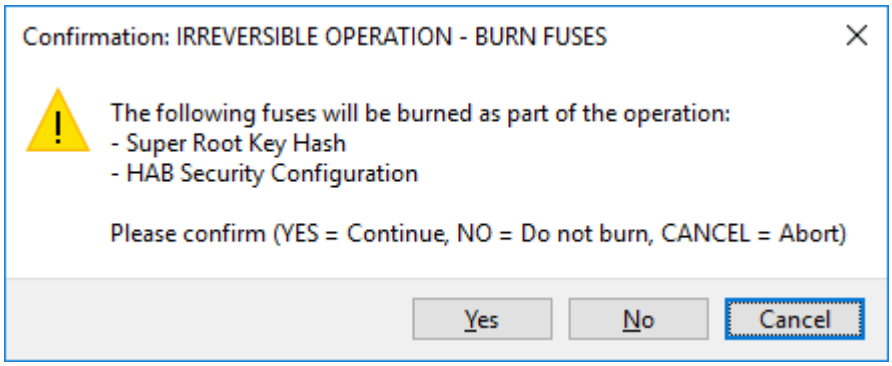

Figure 41. Burn fuses

If the write operation was successful, switch SW7 DIP (RT1050/RT1060/RT1064) or SW8 DIP (RT1015/RT1020/RT1024) or SW1/SW2 DIP (RT1166/RT1176) (see Table 6) and reset the board.

7.2.3.3 Booting Encrypted Image (RT10xx)

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

See the flowchart for visual representation of the process.

To build the image, do the following:

1. In the **Toolbar** set **Boot Type** to **Encrypted (HAB)**.
2. As **Source executable image**, use the image from Preparing Source Image as a **Source executable image**.
3. For **Use the following keys** select any key, for example *SRK1: IMG1_1+CSF1_1*.
4. Click **Build image**.
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 the board is set to Serial bootloader (ISP) mode. See Table 6 for more information.

3. Make sure the **Use built image** checkbox is selected.
4. Select the **Enable security** checkbox.
5. Click **Write Image**.
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 processor can only execute authenticated images.

   - **Cancel** - Abort writing the image and burning fuses.

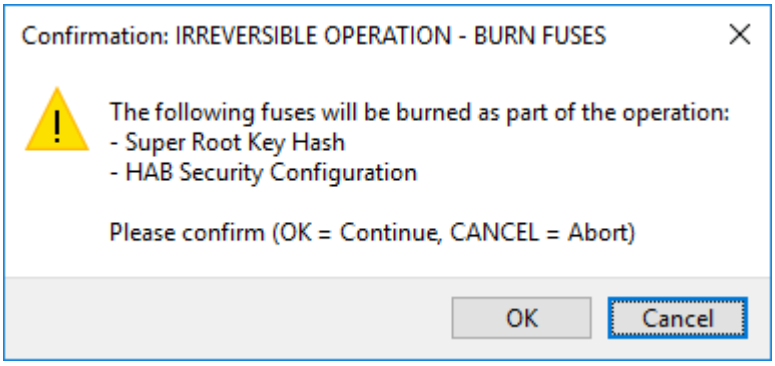

Figure 42. Burn fuses

If the write operation was successful, switch SW7 DIP (RT1050/RT1060/RT1064) or SW8 DIP (RT1020/RT1024) (see Table 6) and reset the board.

NOTE

Part of the encrypted image is a DEK key blob encrypted using 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 located in NOR FLASH. In case no other booting device is supported for those processors, the **Encrypted (HAB)** boot type is not available.

7.2.3.4 Booting XIP Encrypted Image (BEE OTPMK) (RT10xx)

This section describes the building and writing of an XIP encryted image using the OTP Master Key. Authenticated image is built and then it encrypted on-the-fly during the write operation. The source image for the Encrypted XIP with BEE feature must be a XIP image.

See the flowchart for visual representation of the process.

To build the image, do the following:

1. In the **Toolbar**, set **Boot Type** to **XIP Encrypted (BEE OTPMK Key)**.
2. As **Source executable image**, use the image running from external NOR FLASH from Preparing Source Image as a **Source executable image**.
3. For **Use the following keys**, select any key, for example *SRK1: IMG1_1+CSF1_1*.
4. Click **Build image**.
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 the board is set to Serial bootloader (ISP) mode. See Table 6 for more information.
3. Make sure the **Use built image** checkbox is selected.
4. Click **XIP Encryption (BEE OTPMK Key)** 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. Set a corresponding GPIO pin to enable XIP encryption without burning the fuse. See Table 6 for more information.
6. Select the **Enable security** checkbox.
7. Click **Write Image**.
8. 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 processor can only execute authenticated images.

   - **Cancel** - Abort writing the image and burning fuses.

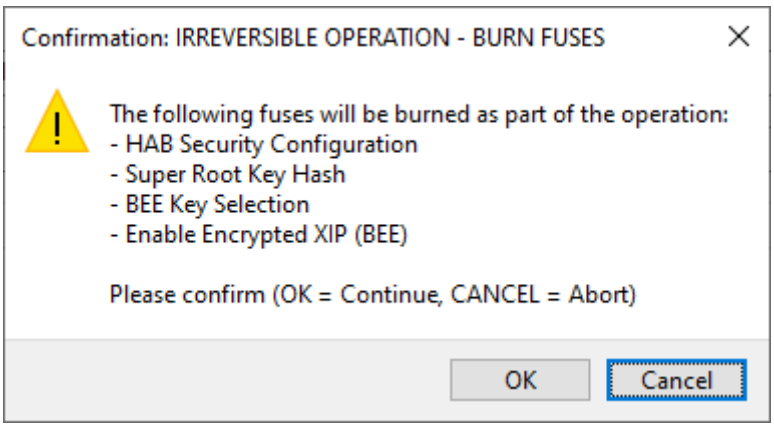

Figure 43. Burn fuses

If the write operation was successful, switch SW7 DIP (RT1050/RT1060/RT1064) or SW8 DIP (RT1015/RT1020/RT1024) (see Table 6) and reset the board.

7.2.3.5 Booting XIP Encrypted Image (BEE user keys) Unsigned (RT10xx)

This section describes the building and writing of an XIP encrypted image using User Keys. 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 BEE feature must be a XIP image.

See the flowchart for visual representation of the process.

To build the image, do the following:

1. In the **Toolbar**, set **Boot Type** to **XIP Encrypted (BEE User Keys) Unsigned**.
2. As **Source executable image**, use the image external NOR FLASH from Preparing Source Image 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 **Build image**.
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 the board is set to Serial bootloader (ISP) mode. See Table 6 for more information.
3. Ensure **Use built image** checkbox is selected.
4. Set a corresponding GPIO pin to enable XIP encryption without burning the fuse. See Table 6 for more information.
5. Click **Write Image**.
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's not possible to modify them.

   - **Cancel** - Abort writing the image and burning fuses.

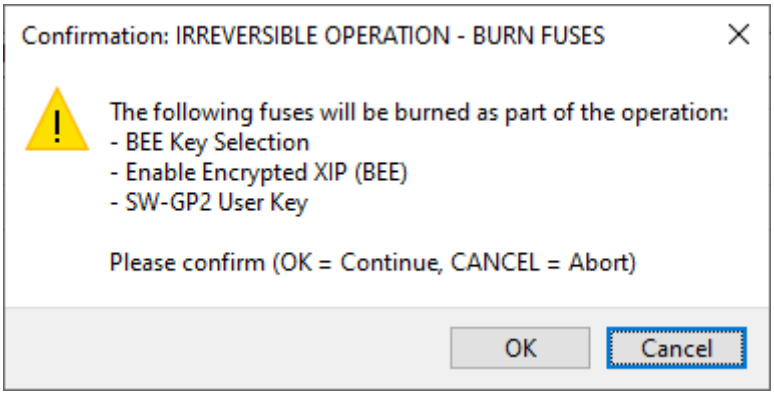

Figure 44. Burn fuses

If the write operation was successful, switch SW7 DIP (RT1050/RT1060/RT1064) or SW8 DIP (RT1015/RT1020/RT1024) (see Table 6) and reset the board.

7.2.3.6 Booting XIP Encrypted Image (BEE User Keys) Authenticated (RT10xx)

This section describes the building and writing of an XIP encryted image using User Keys. 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 BEE feature must be a XIP image.

See the flowchart for visual representation of the process.

To build the image, do the following:

1. In the **Toolbar** set **Boot Type** to **XIP Encrypted (BEE User Keys) Authenticated**.
2. As **Source executable image**, use the image external NOR FLASH from Preparing Source Image as a **Source executable image**.
3. For **Use the following keys** 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. Click **Build image**.
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 the board is set to Serial bootloader (ISP) mode. See Table 6 for more information.
3. Ensure **Use built image** checkbox is selected.
4. Set a corresponding GPIO pin to enable XIP encryption without burning the fuse. See Table 6 for more information.
5. Select the **Enable security** checkbox.
6. Click **Write Image**.
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 processor can only execute authenticated images.

   - **Cancel** - Abort writing the image and burning fuses.

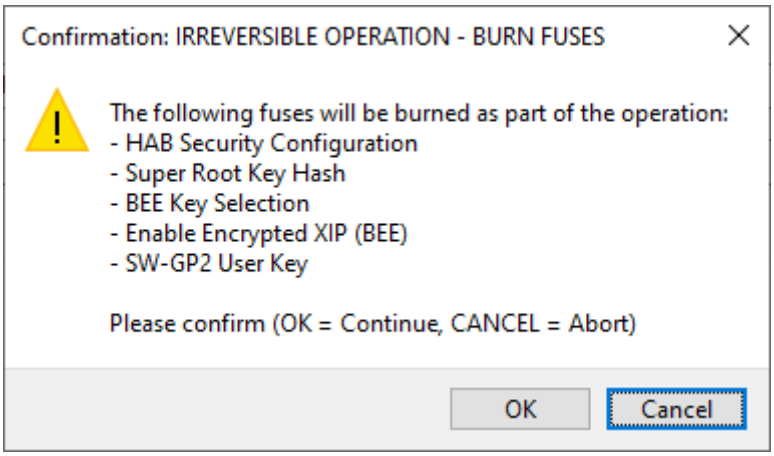

Figure 45. Burn fuses

If the write operation was successful, switch SW7 DIP (RT1050/RT1060/RT1064) or SW8 DIP (RT1015/RT1020/RT1024) (see Table 6) and reset the board.

7.2.3.7 Booting OTFAD Encrypted Image (User Keys) Unsigned (RT11xx)

This section describes the building and writing of an OTFAD 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 enabled Encrypted XIP. The source image for the Encrypted XIP with OTFAD feature must be a XIP image.

See the flowchart for visual representation of the process.

To build the image, do the following:

1. In the **Toolbar** set **Boot Type** to **OTFAD Encrypted (User Keys) Authenticated**.
2. As **Source executable image**, use the image external NOR FLASH from Preparing Source Image as a **Source executable image**.

3. 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 KeyStore), KEK, KEK Scramble, User Keys for regions, Regions ranges, random key generation.
4. Click **Build image**.
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 the board is set to Serial bootloader (ISP) mode. See Table 6 for more information.
3. Reset the board if the OTFAD KEK Source is set to KeyStore. This is necessary so that the Key Store is enrolled successfully.
4. Ensure **Use built image** checkbox is selected.
5. Open **OTP Configuration** and review the settings and fix any reported problems.
6. Set a corresponding GPIO pin to enable XIP encryption without burning the fuse. See Table 6 for more information.
7. Click **Write Image**.
8. 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.

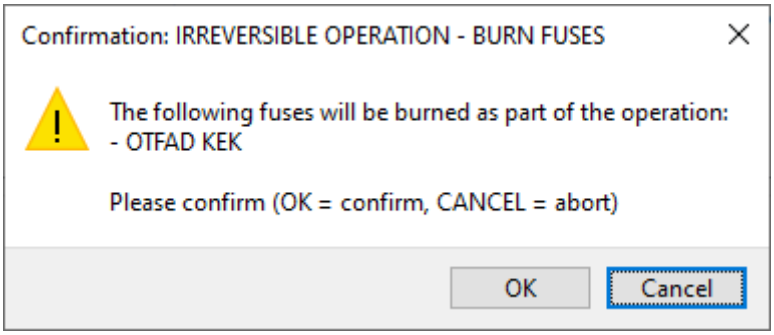

Figure 46. Burn Fuses

If the write operation was successful, switch SW1 DIP (see Table 6) and reset the board.

7.2.3.8 Booting OTFAD Encrypted Image (User Keys) Authenticated (RT11xx)

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 enabled Encrypted XIP. The source image for the Encrypted XIP with OTFAD feature must be a XIP image.

See the flowchart for visual representation of the process.

To build the image, do the following:

1. In the **Toolbar** set **Boot Type** to **OTFAD Encrypted (User Keys) Authenticated**.
2. As **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 **Keys Management** view. For more information, see Keys Management.

4. For **Use the following keys** select any key, for example *SRK1: IMG1_1+CSF1_1*.
5. 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 KeyStore), KEK, KEK Scramble, User Keys for regions, Regions ranges, random key generation.
6. Click **Build image**.
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 the board is set to Serial bootloader (ISP) mode. See Table 6 for more information.
3. Reset the board if the OTFAD KEK Source is set to KeyStore. This is necessary so that the Key Store is enrolled successfully
4. Ensure **Use built image** checkbox is selected.
5. Open **OTP Configuration** and review the settings and fix any reported problems.
6. Set a corresponding GPIO pin to enable XIP encryption without burning the fuse. See Table 6 for more information.
7. Select the **Enable security** checkbox.
8. Click **Write Image**.
9. 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 processor can only execute authenticated images.

   - **Cancel** - Abort writing the image and burning fuses.

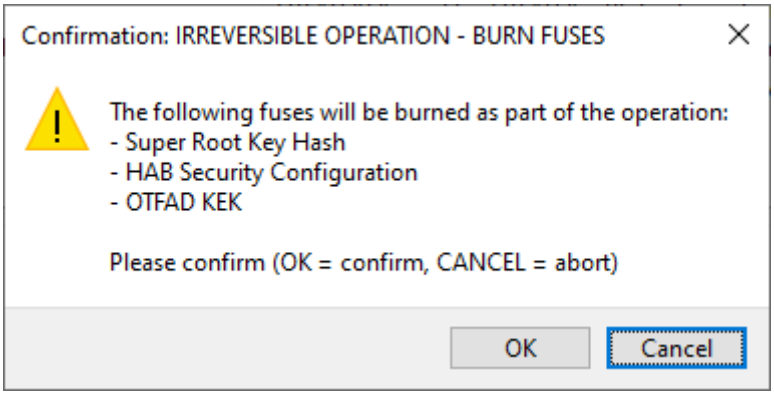

Figure 47. Burn Fuses

If the write operation was successful, switch SW1 DIP (see Table 6) and reset the board.

7.2.4 Creating/Customizing DCD files

It's 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 a SDK source code by selecting **File > Import > MCUXpresso Config Tools > Import Source**.
3. Select the file from SDK package located in *boards\evkmimxrt10##\xip\evkmimxrt10##_sdram_ini_dcd.c*.

4. Switch to 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 location where to generate binary file.

NOTE

Refer to the documentation of Device Configuration Tool for more information.

7.2.5 Flowcharts (RT10xx)

Following flowcharts display the process of generation and provisioning of RT10xx devices in sequence, with focus on input, output and temporary files and their location in the workspace.

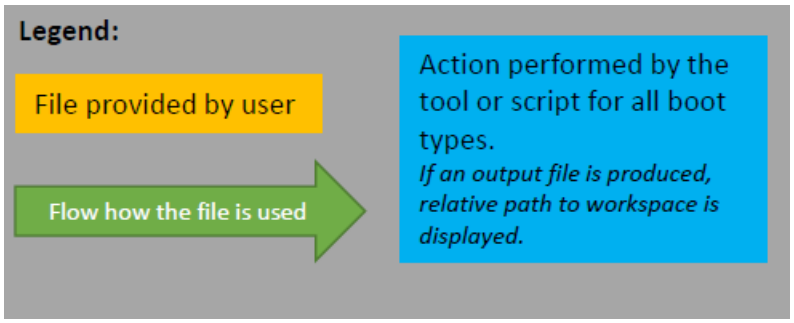

Figure 48. Legend

7.2.5.1 Unsigned

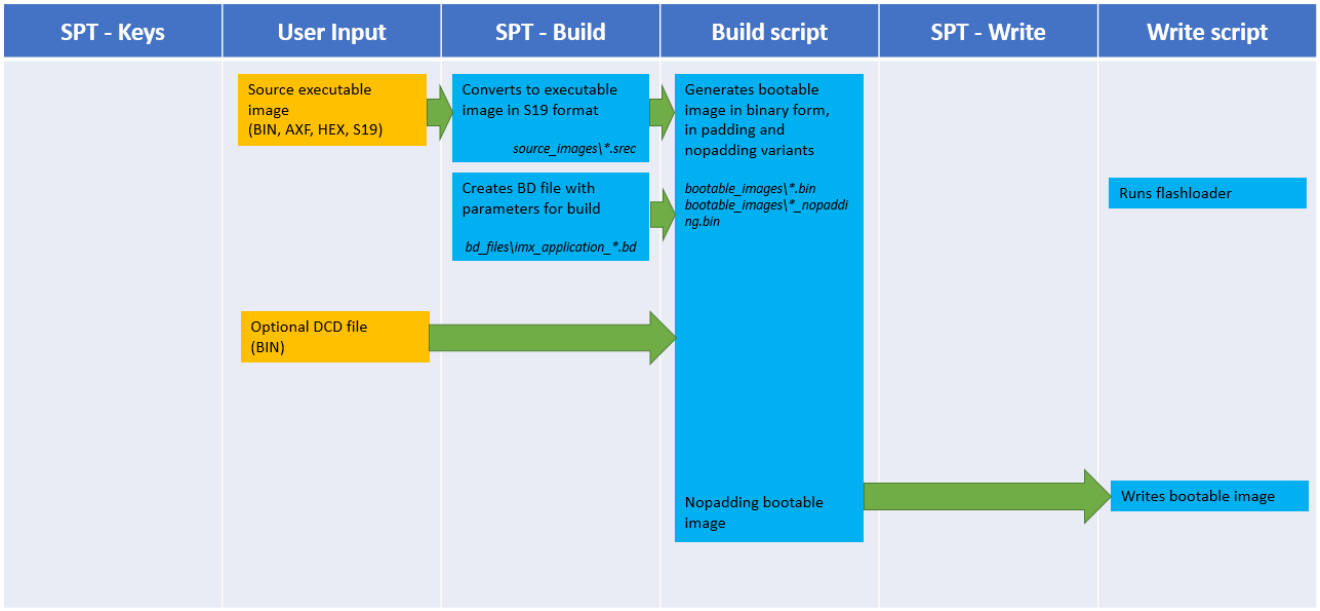

Figure 49. RT10xx: Unsigned

7.2.5.2 Authenticated (HAB)

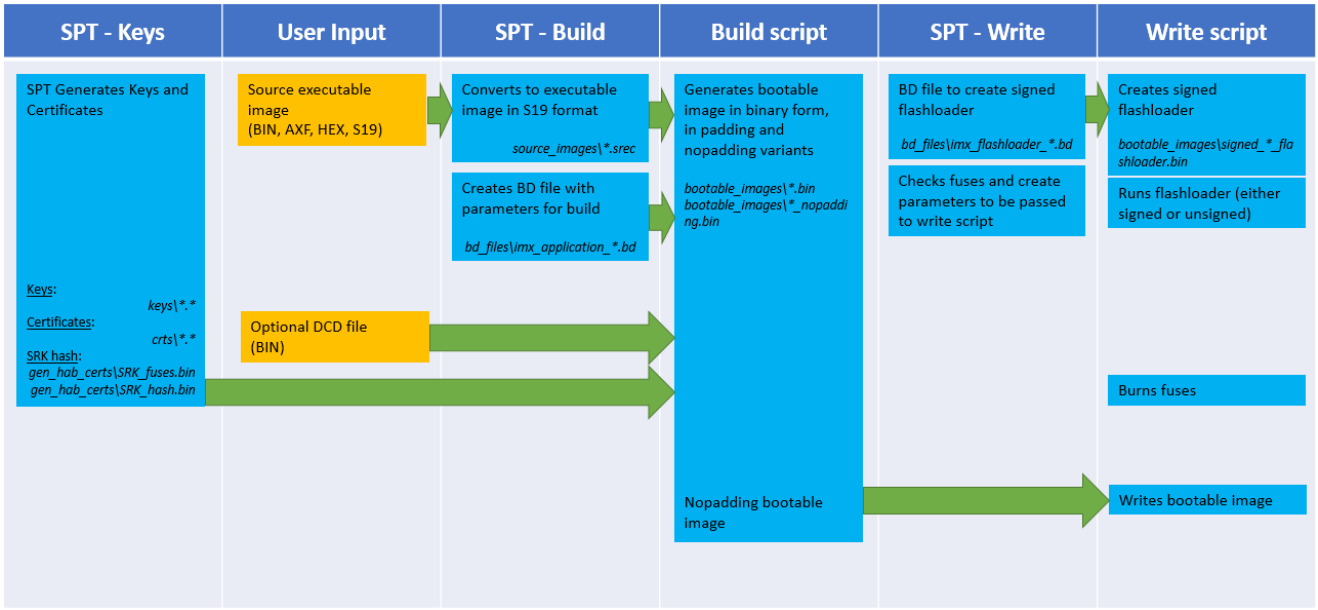

Figure 50. RT10xx: Authenticated (HAB)

7.2.5.3 Encrypted (HAB)

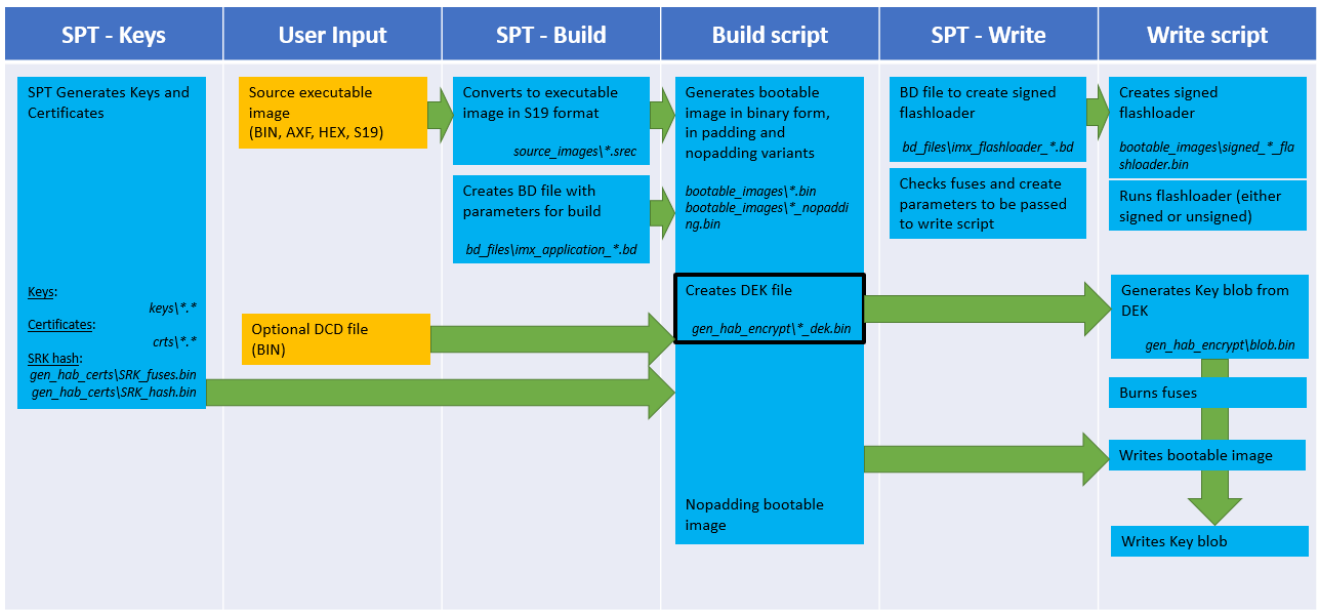

Figure 51. RT10xx: Encrypted (HAB)

7.2.5.4 XIP Encrypted (BEE OTMPK)

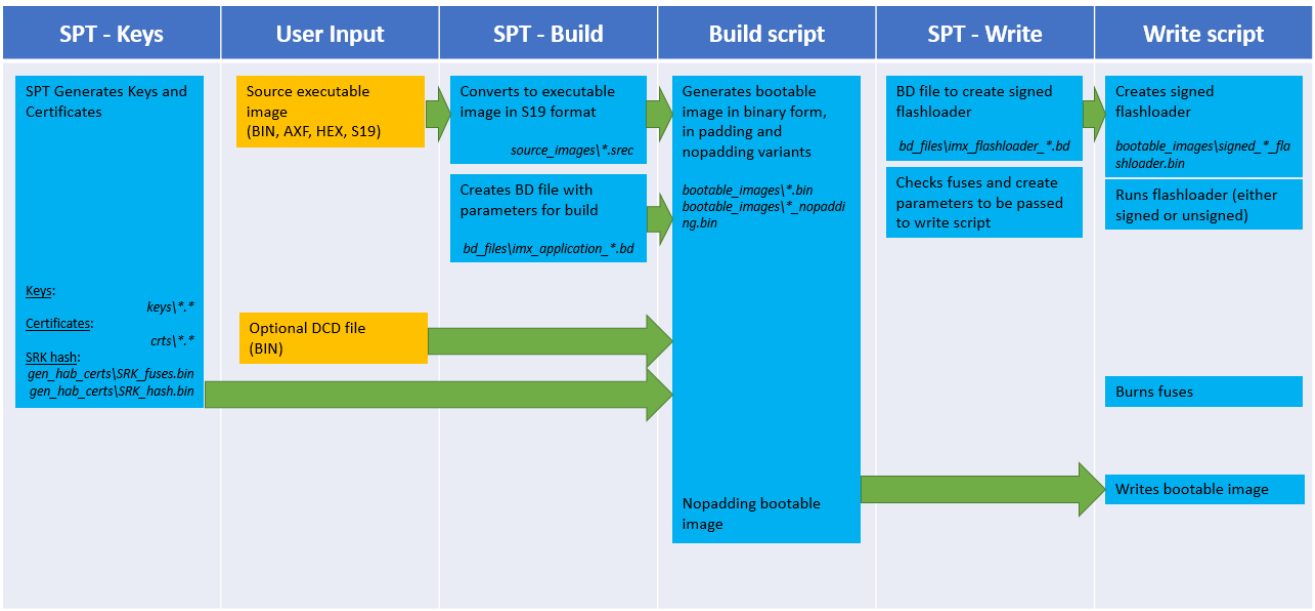

Figure 52. RT10xx: XIP encrypted (BEE OTMPK)

7.2.5.5 XIP Encrypted (BEE User Keys) Unsigned

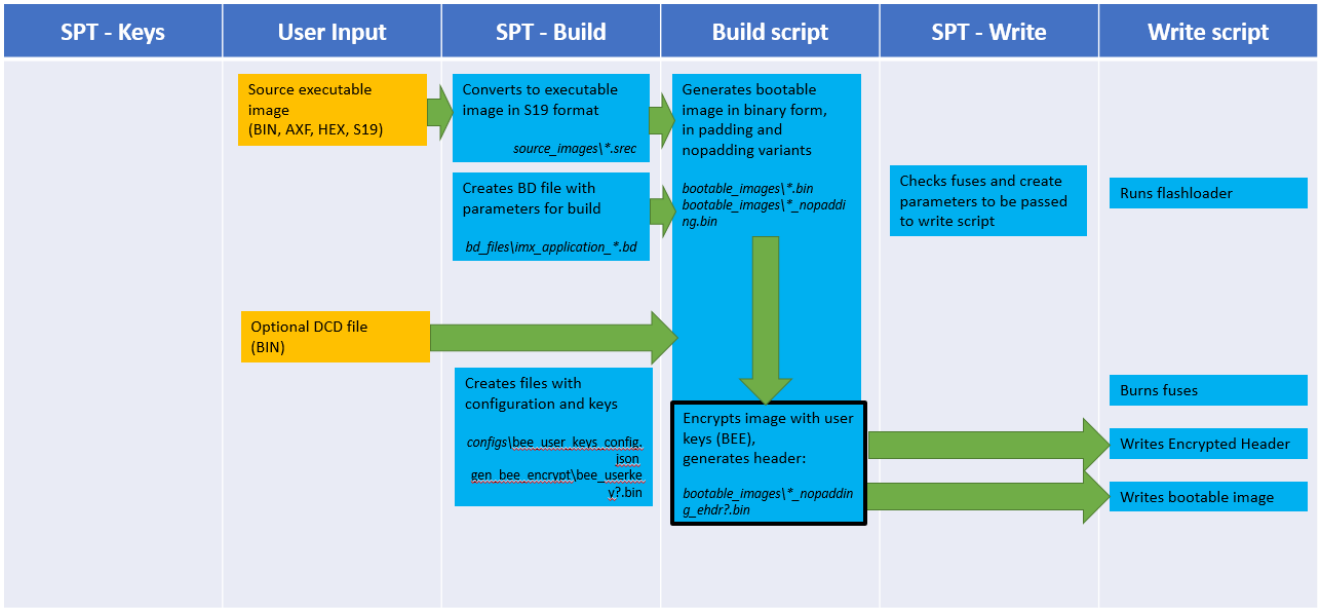

Figure 53. RT10xx: XIP encrypted (BEE user keys) unsigned

7.2.5.6 XIP Encrypted (BEE User Keys) Authenticated

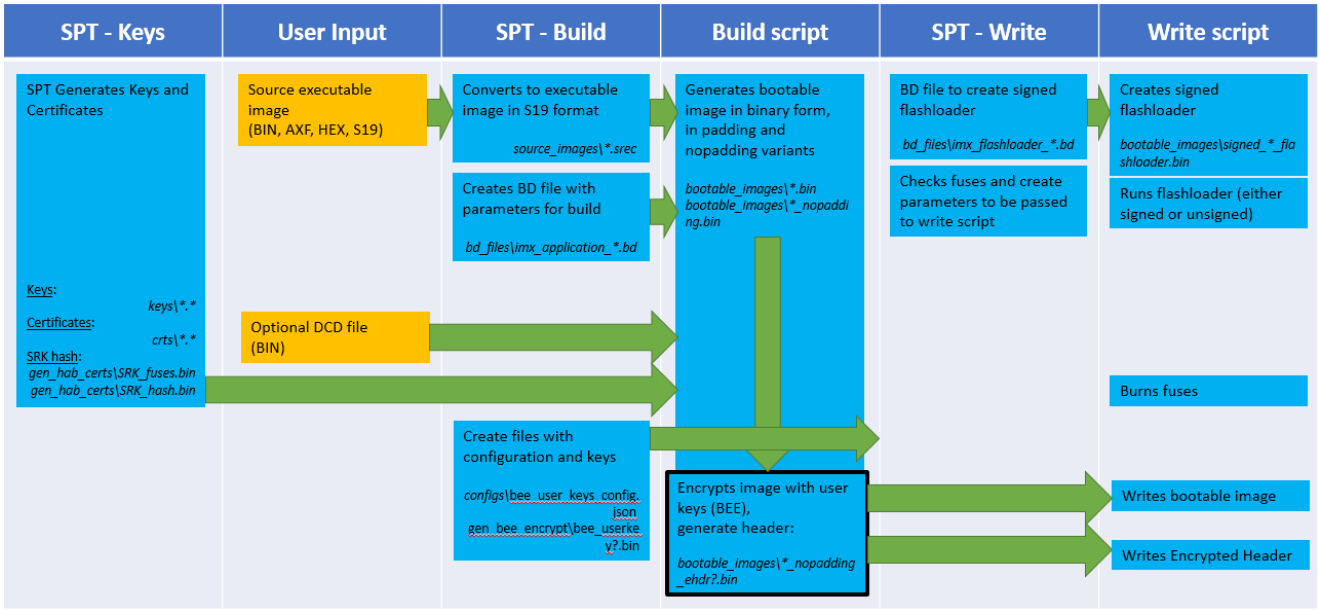

Figure 54. RT10xx: XIP encrypted (BEE user keys) authenticated

7.2.6 Flowcharts (RT11xx)

Following flowcharts display the process of generation and provisioning of RT11xx devices in sequence, with focus on input, output and temporary files and their location in the workspace.

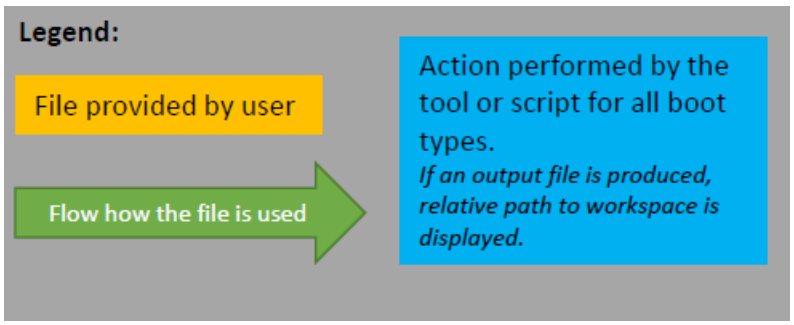

Figure 55. Legend

7.2.6.1 Unsigned

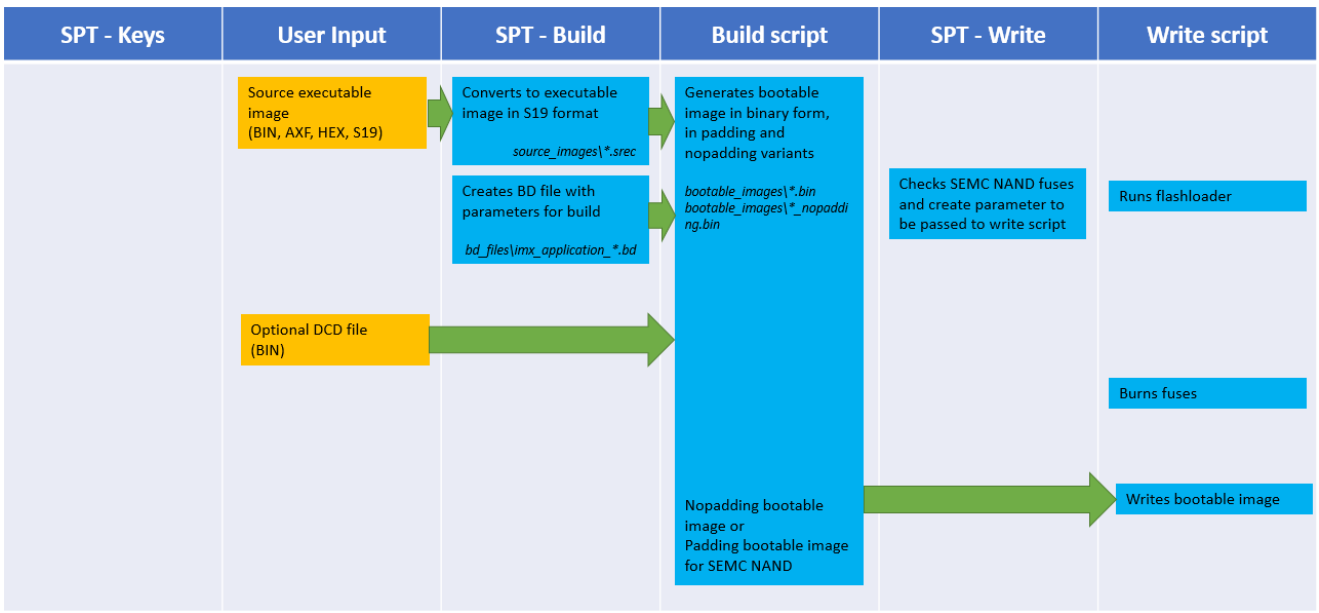

Figure 56. RT11xx: Unsigned

7.2.6.2 Authenticated

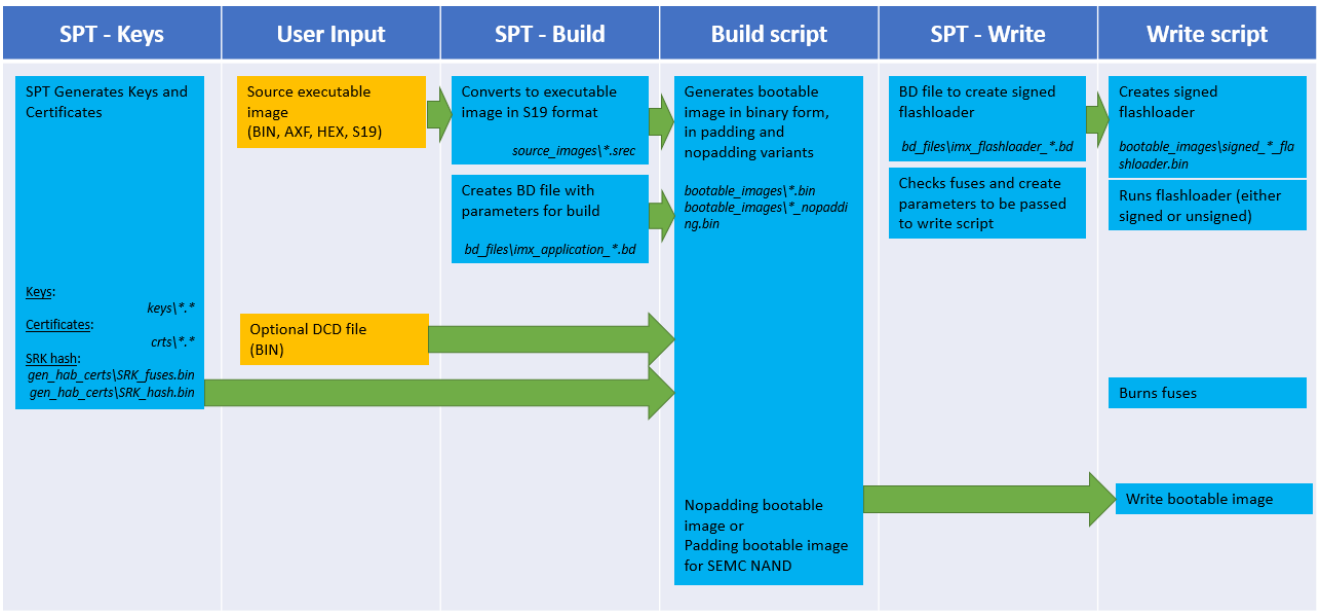

Figure 57. RT11xx: Authenticated

7.2.6.3 OTFAD Encrypted (User Key) Unsigned

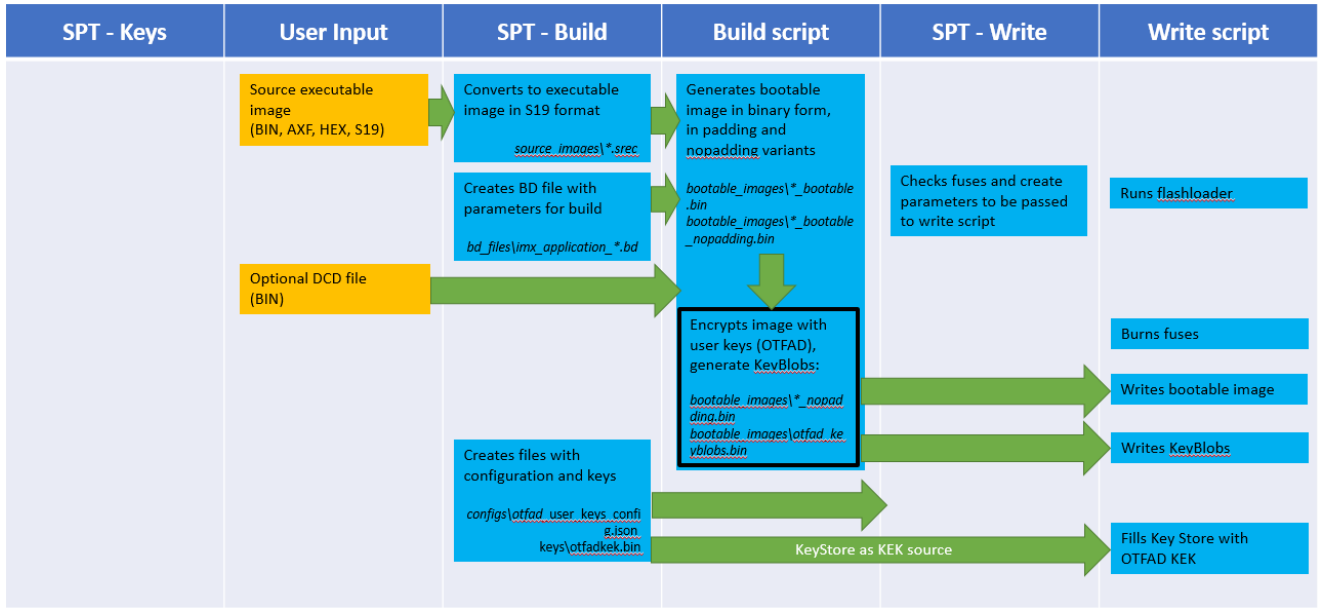

Figure 58. RT10xx: OTFAD Encrypted (User Key) Unsigned

7.2.6.4 OTFAD Encrypted (User Key) Authenticated

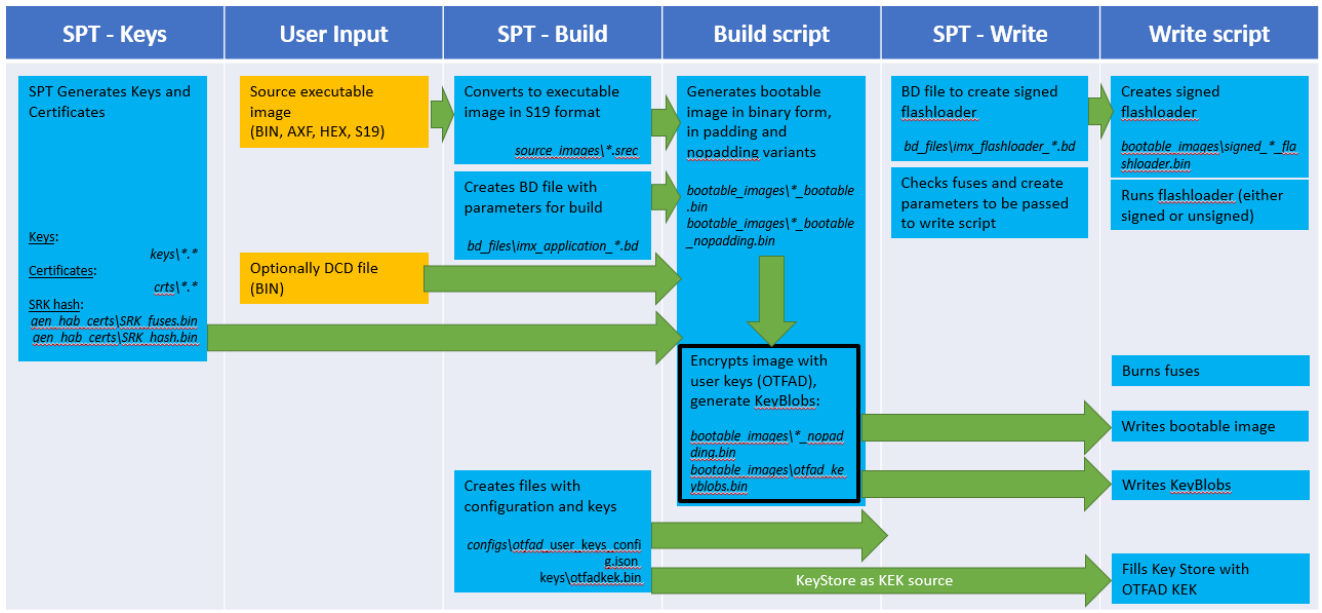

Figure 59. RT10xx: OTFAD Encrypted (User Key) Authenticated

7.3 LPC55Sxx Device Workflow

7.3.1 Preparing Source Image

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

- **Image running from internal FLASH**

  This is the only supported boot device for the LPC55Sxx device family. The image is executed directly from internal flash memory.

7.3.1.1 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.
  2. Build the project.

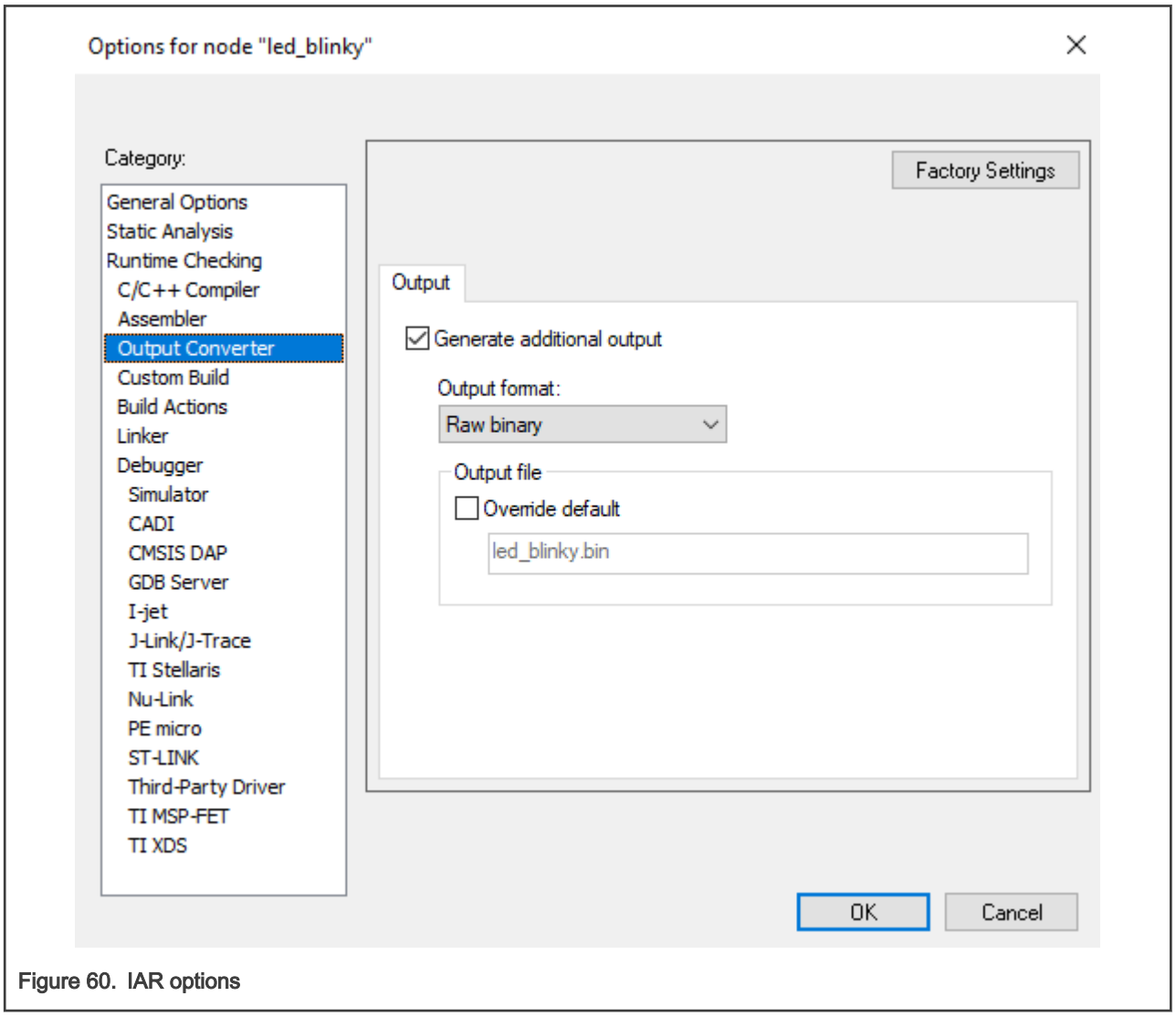

Figure 60. IAR options

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 **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*.

7.3.2 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's assumed SEC is already running with a workspace created for an LPC55Sxx device. For more information, see Setting up Secure Provisioning Tool.

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

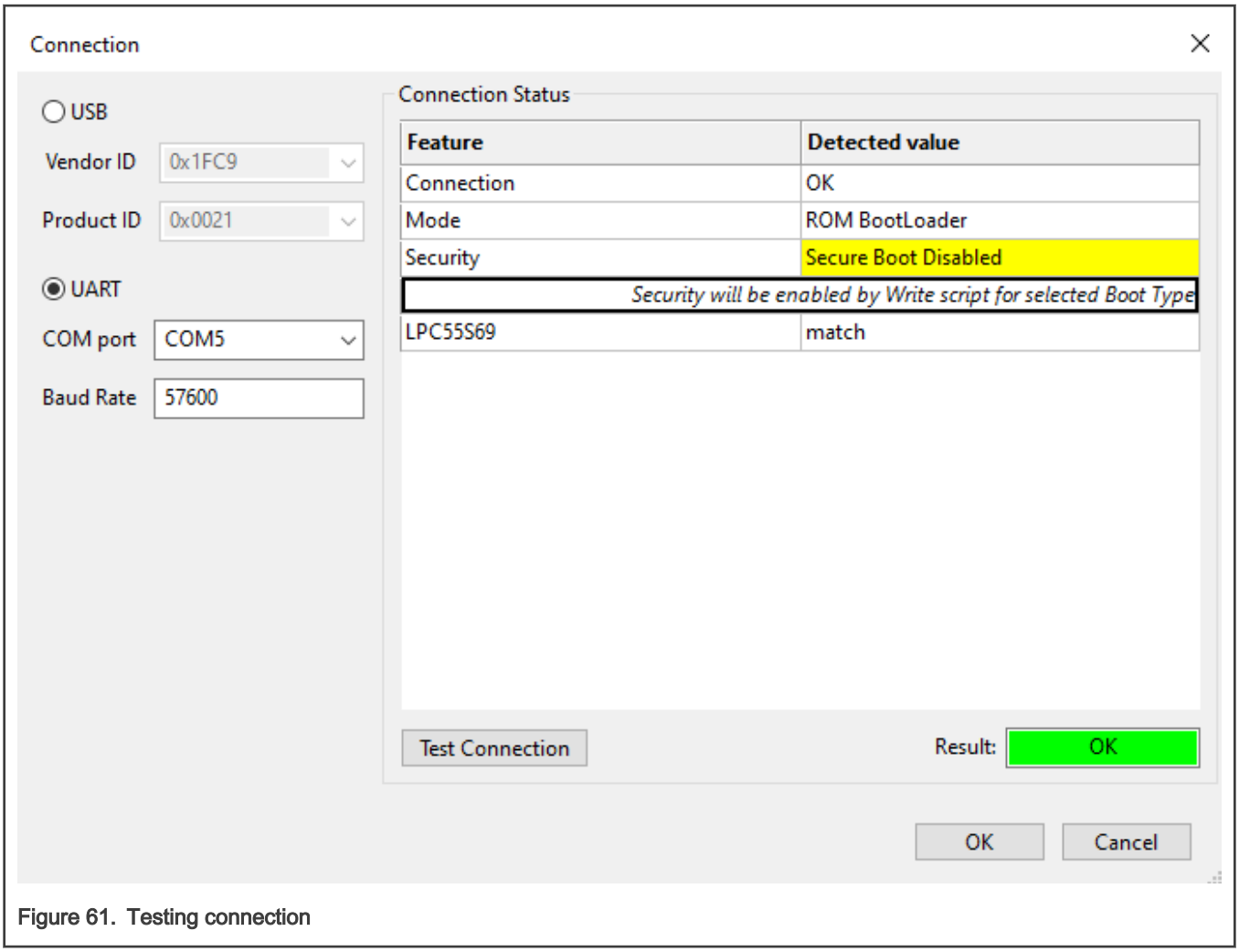

Figure 61. Testing connection

7.3.3 Booting Images

This section describes the building and writing of images.

7.3.3.1 Security Levels

Following security levels are supported in SEC:

Unsigned boot types	Default processor configuration that does not provide any security. It's 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 Key store, CFPA and CMPA pages are written into the processor. CMPA page is not sealed and can be updated or erased (for more information, see clear_security).
Signed or encrypted boot types - sealed	Sealed CMPA page is recommended for production. Select **Enable security** checkbox in the **Write Image** view to seal the CMPA page. Once sealed, it cannot be changed not erased.

7.3.3.2 Booting Unsigned/Unsigned with CRC Image

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

See the flowcharts for visual representation of the Unsigned and Unsigned with CRC processes.

1. In the **Toolbar**, set **Boot Type** to **Unsigned** or **Unsigned with CRC**.
2. As a **Source executable image**, use the image from Preparing Source Image as a **Source executable image**.
3. In case of binary image set the start address to *0x0*.
4. Click **Build image**.
5. Check that the bootable image was built successfully.

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

1. Make sure the board is in ISP mode.
2. Click the **Write image** tab.
3. Make sure that the **Use built image** checkbox is selected.
4. Click **Write image**.

If the write operation was successful, reset the board.

7.3.3.3 Booting Signed or PRINCE Encrypted Image

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

See the flowcharts for visual representation of the Signed, PRINCE Encrypted with CRC, and PRINCE Encrypted and Signed processes.

> **NOTE**
>
> The keys are also used for **PRINCE encrypted 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) with CRC**, or **Encrypted (PRINCE) and Signed**.
2. In the **Build Image** view, use the image from Preparing Source Image as a Source executable image.
3. For **Use the following keys**, 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 **Build image**.
6. Check that the bootable image was built successfully.

Once the image has been successfully built, check enable security checkbox to permanently seal the device security with sha256 signature of the CMPA page. If the option remain unselected the security can be reconfigured.

To write the image, do the following:

1. Click the **Write image** tab.
2. Make sure the board is connected and the ISP mode is enabled (See Connecting the Board.)
3. Make sure the **Use built image** checkbox is selected.
4. Click **Write Image**.

   If the write operation was successful, reset the board.

   Once the image can be successfully executed in the processor, check **Enable Security** checkbox to permanently seal the device security with sha256 signature of the CMPA page. If the option remain unselected the security can be reconfigured. After you select the checkbox, click **Write Image** again and confirm the following message box:

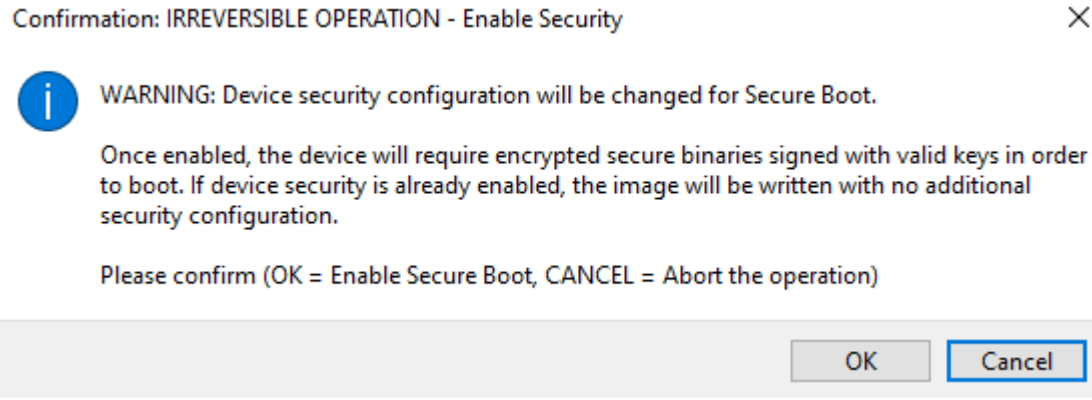

Figure 62. Confirm write

If the write operation was successful, reset the board.

7.3.3.4 Key Store

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

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

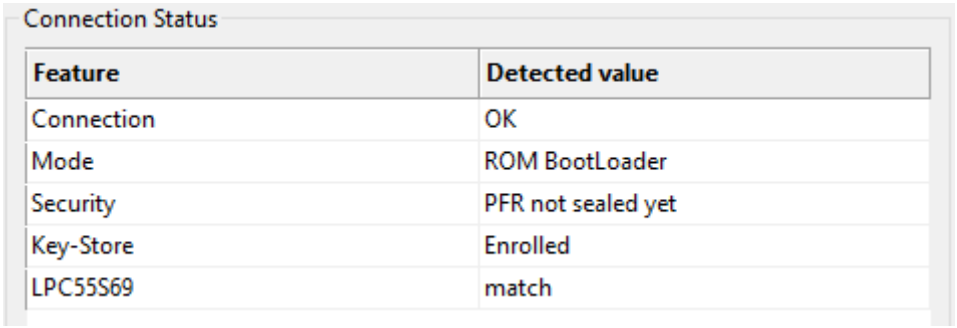

Figure 63. Key store connection

If you need to update key store, you can erase it, but it's strongly recommended to also clear the CFPA page, as PRINCE IV fields in CFPA depend on key store. The device will not boot if you enroll key store and try to use it with previous IV fields.

How to erase key store (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.

Bear in mind that tools/spsdk/blhost does not support set-property command. You must use the legacy blhost (bin/tools/blhost).

How to update CFPA page (example for LPC55S69)

1. Increment version in cfpa.json: it's 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
```

7.3.4 Flowcharts

Following flowcharts display the process of generation and provisioning of LPC55Sxx devices in sequence, with focus on input, output and temporary files and their location in the workspace.

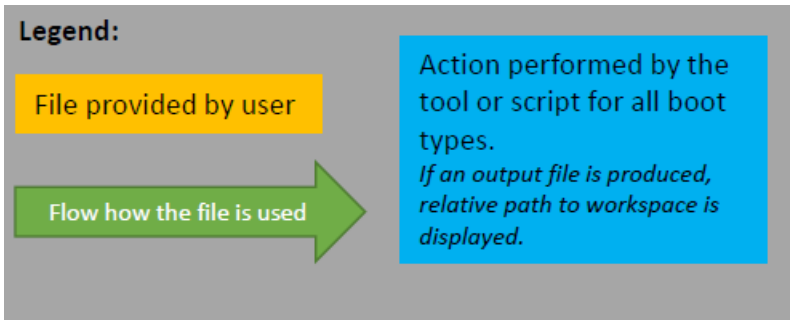

Figure 64. Legend

7.3.4.1 Unsigned

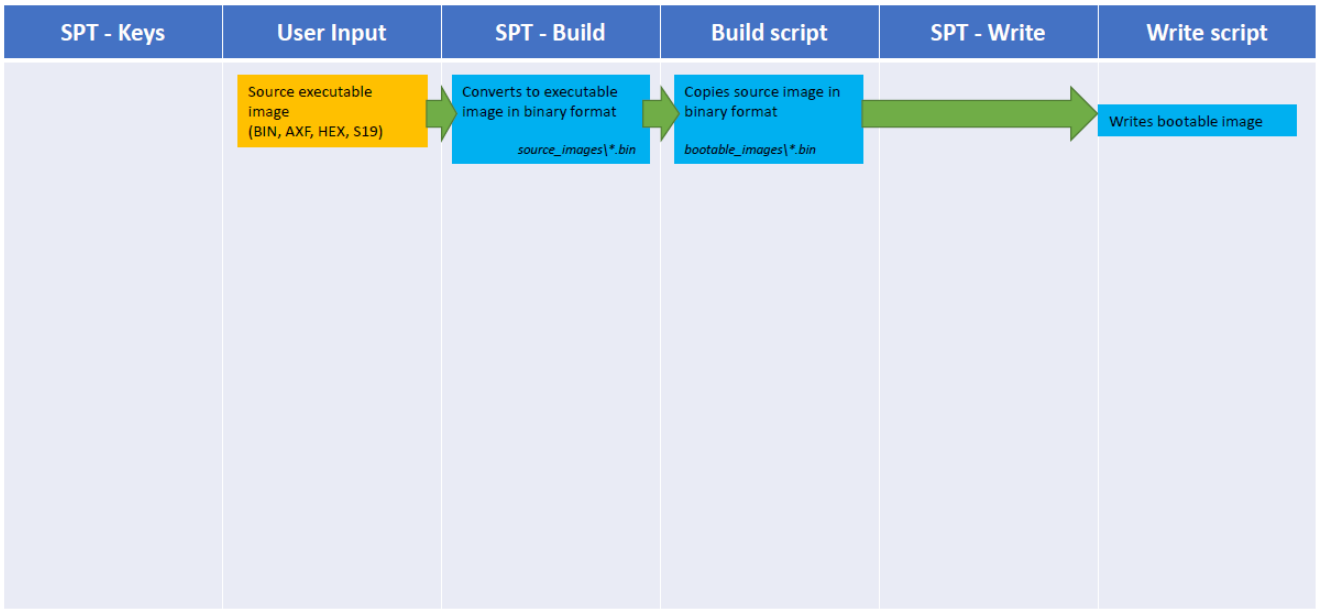

Figure 65. LPC55Sxx unsigned

7.3.4.2 Unsigned with CRC

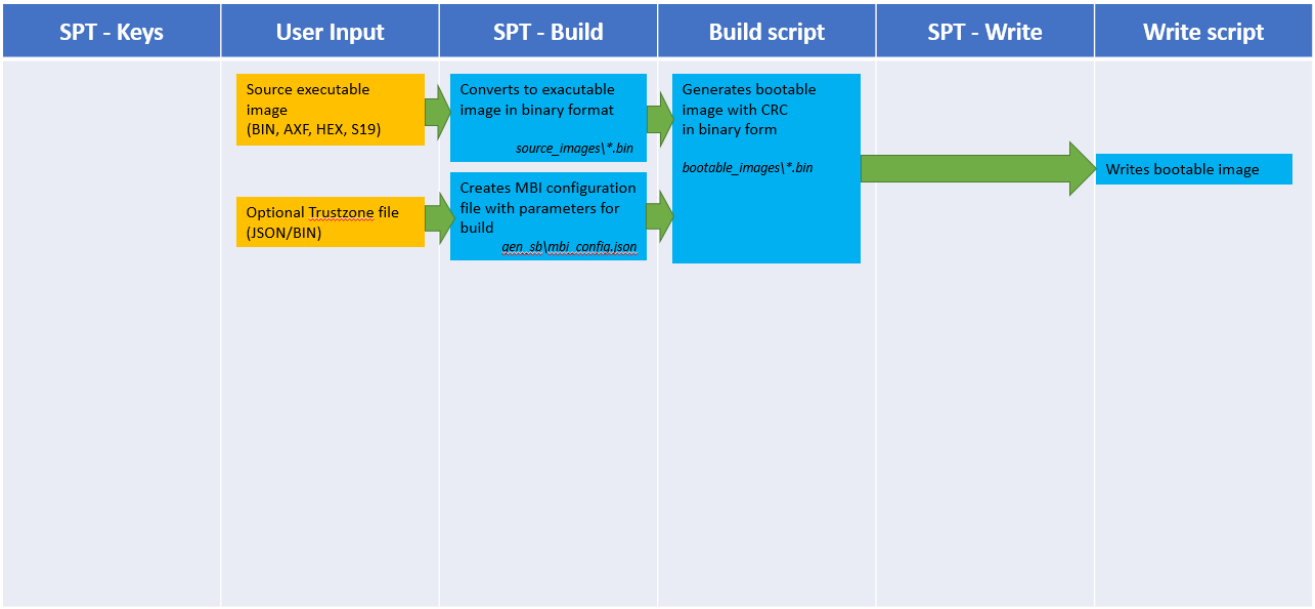

Figure 66. LPC55Sxx unsigned with CRC

7.3.4.3 Signed

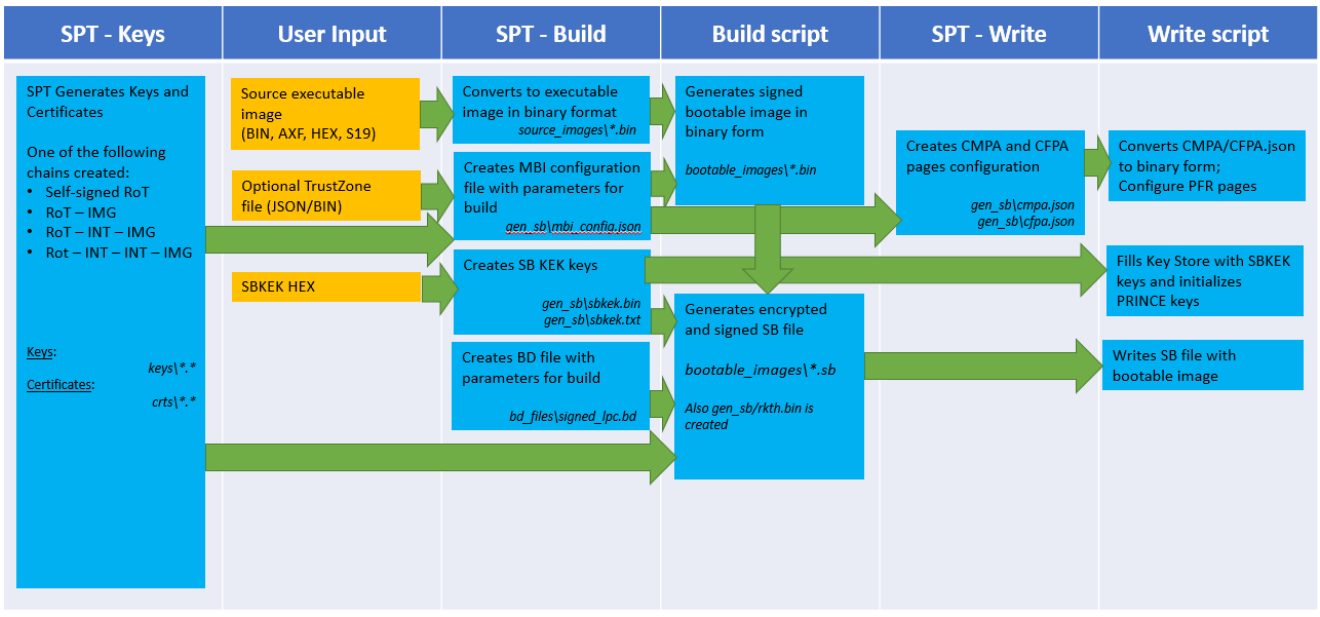

Figure 67. LPC55Sxx signed

7.3.4.4 PRINCE Encrypted and Signed

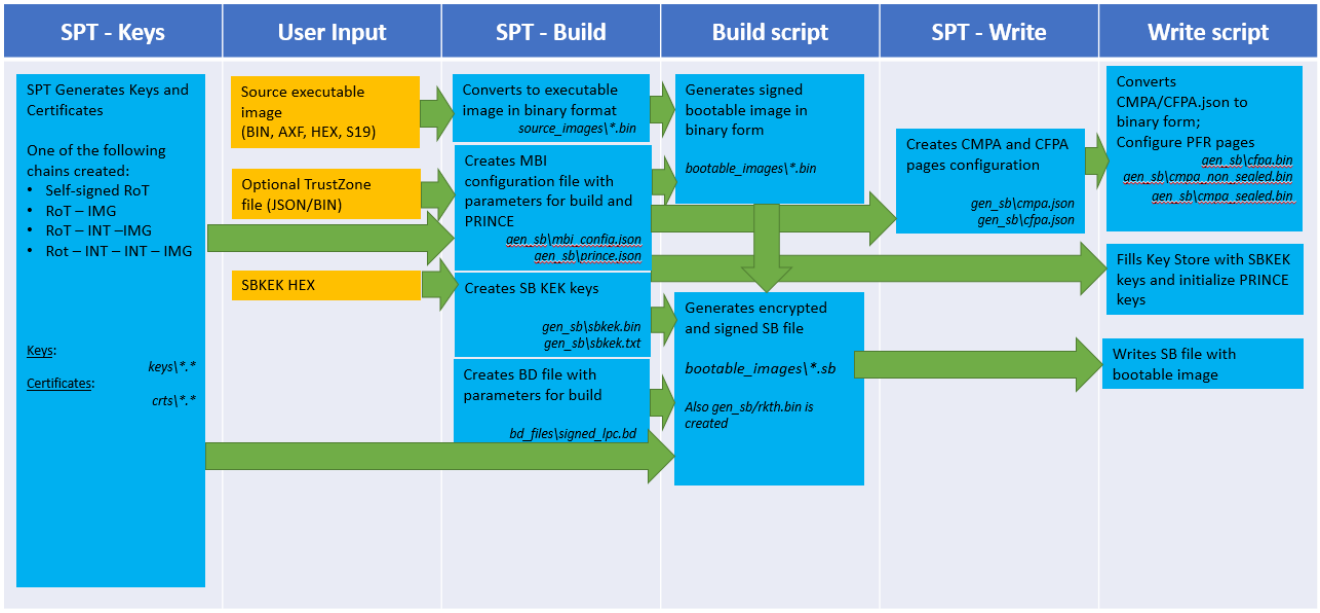

Figure 68. LPC55Sxx PRINCE encrypted and signed

7.3.4.5 PRINCE Encrypted with CRC

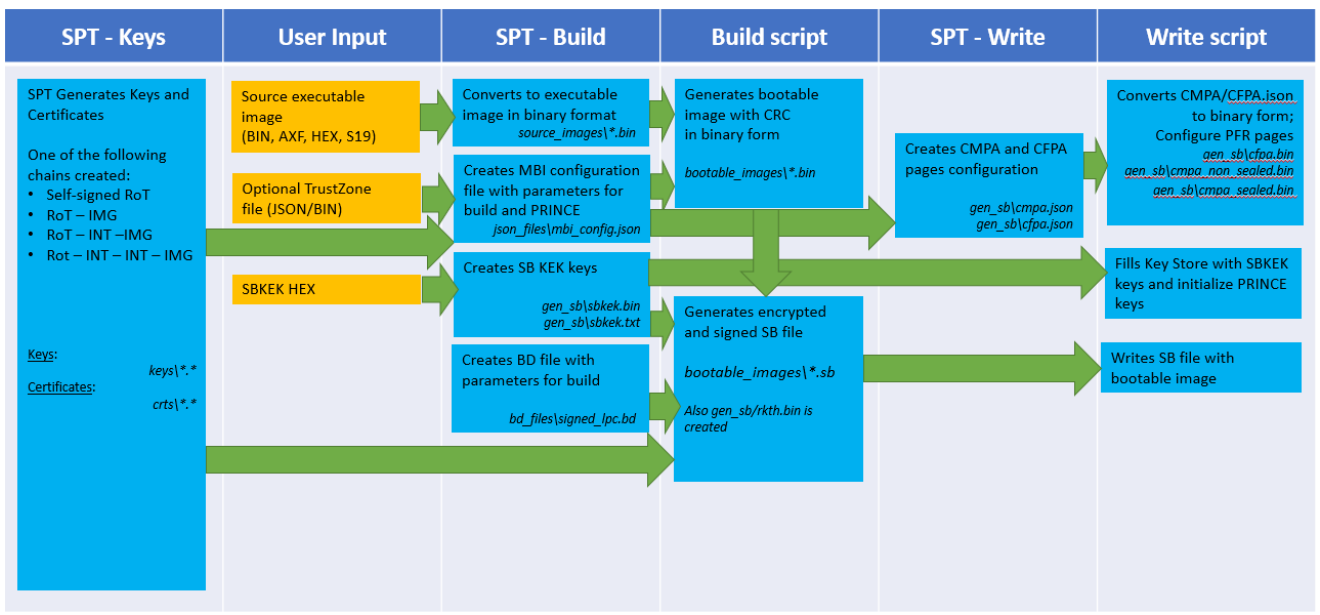

Figure 69. LPC55Sxx PRINCE encrypted with CRC

7.4 RTxxx Device Workflow

7.4.1 Preparing Source Image

In this step, you must select target memory where the image will be executed.

Currently, the only supported boot device is NOR FLASH.

Following boot devices 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 to RAM before the execution

It's 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.

7.4.1.1 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 into linker as a workaround for the problem:

     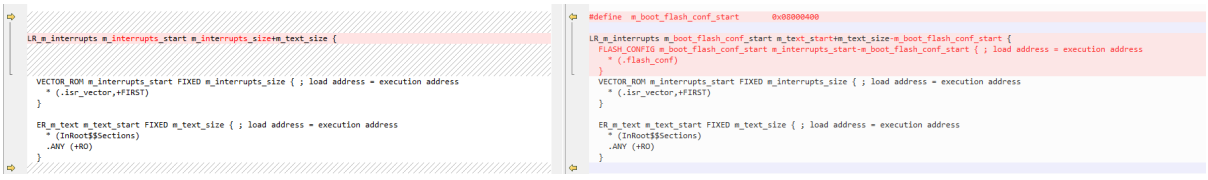

     Figure 70. Keil MDK workaround
  5. Build the image.

     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*.

7.4.1.2 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 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*.

7.4.2 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
MIMXRT595-EVK	SW7[1:3]: 100 (UART) SW7[1:3]: 101 (USB)	SW7[1:3]: 001
MIMXRT685-EVK	SW5[1:3]: 100	SW5[1:3]: 101

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 Setting up Secure Provisioning Tool.
4. Set connection to **USB** and test the board connection.

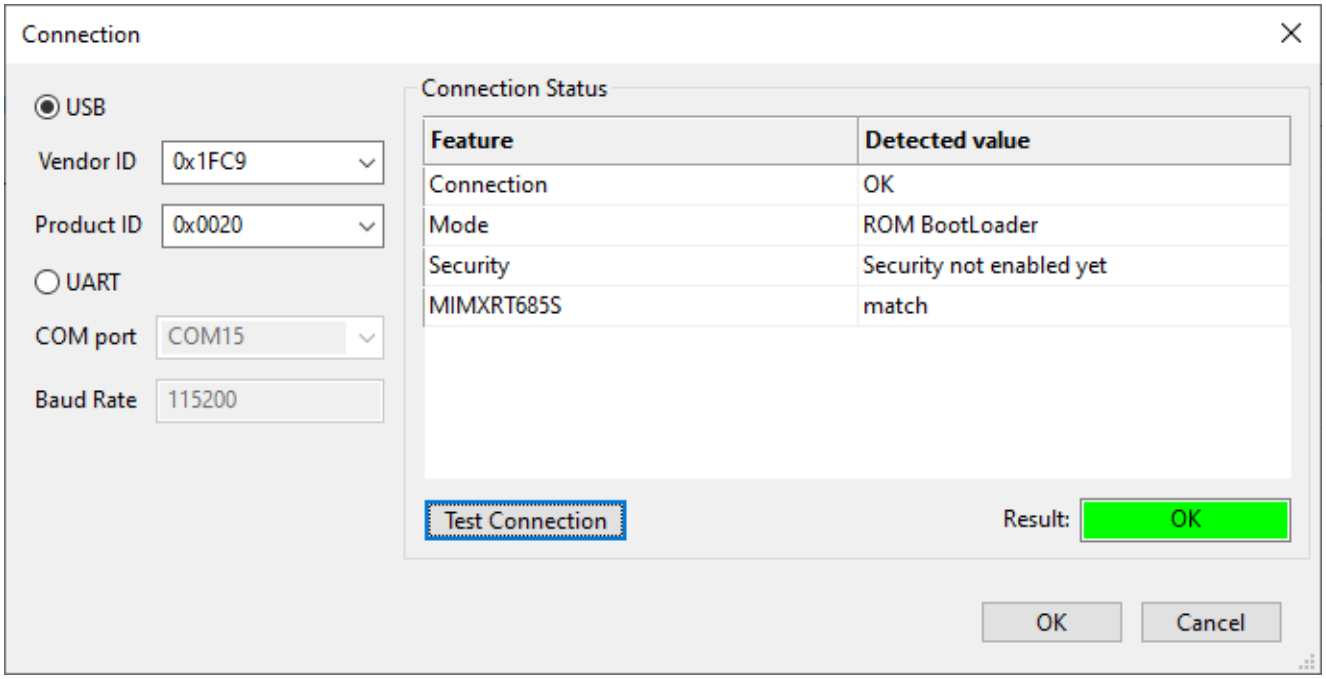

Figure 71. Testing the connection

7.4.3 Booting Images

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

7.4.3.1 Booting an Unsigned/Unsigned with CRC Image

Unsigned image is typically used for development. It's recommended to start with this boot type before working with securedimages to verify that the executable image works properly.

See the flowcharts for visual representation of the unsigned and unsigned with CRC processes.

To build a bootable image, follow these steps:

1. In the **Toolbar**, select **Unsigned** or **Unsigned with CRC** in **Boot Type**.
2. Switch to the **Build image** view.
3. Select image build in Preparing Source Image as a **Source executable image**.
4. Click **Build image** to build a bootable image.

When the bootable image has been successfully built:

1. Connect the board, see Connecting the Board.
2. Switch to the **Write image** view.
3. Make sure 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 QSPI reset pin is not burnt.
5. Click **Write image**.

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

7.4.3.2 Booting Signed Image Using Shadow Registers

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

See the Flowchart for a visual representation of the process.

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 **Keys Management** view. For more information, see Keys Management.
4. For **Use the following keys**, select any key, for example *ROT1: IMG1_1*.
5. As **Key Source** select **OTP** or **KeyStore**. KeyStore represents a higher security level, as PUF is used. See Full Security for the limitations.
6. Generate random **User Key** and **SBKEK**.
7. Click **Build image** 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 the board is set to Serial bootloader (ISP) mode. For more information, see Connecting the Board.
3. Make sure the **Use built image** checkbox is selected.
4. Open **OTP Configuration** and review all reported problems. For fuse **BOOT_CFG[0]** being locked after write, it's necessary to specify the whole value, as it will be programmed only once.
5. **Reset the board**. This is necessary because:
   - Shadow registers cannot be updated or written twice, they can be set to a “clean” processor only.
   - If the fuse for QSPI reset pin is not burnt, it's necessary to reset the FLASH manually before each configuration.
6. Select the **Use Shadow Registers** checkbox.

NOTE

Do **not** select the **Enable security** checkbox.

7. Click **Write Image**.

During the write operation, the following steps are performed:

1. Fuses are checked to ensure the board is in an unsecured mode.
2. 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

7.4.3.3 Booting OTFAD Encrypted Image Using Shadow Registers

This section describes the building and writing of an encrypted image (OTFAD Encryption).

See the flowchart for a visual representation of the process.

1. In the **Toolbar** set Boot Type to **OTFAD Encrypted (User Keys) with CRC** or **OTFAD Encrypted (User Keys) 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 **Keys Management** view. For more information, see Keys Management.
4. For **Use the following keys**, select any key, for example ROT1: IMG1_1.

5. As **Key Source** select **OTP** or **KeyStore**. KeyStore represents a higher security level, as PUF is used. See Full Security for the limitations.
6. Generate random **User Key** and **SBKEK**.
7. Open **OTFAD Encryption** and set random keys.
8. Click **Build image** 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 the board is set to Serial bootloader (ISP) mode. For more information, see Connecting the Board.
3. Make sure the **Use built image** checkbox is selected.
4. Open **OTP Configuration** and review all reported problems. For fuse **BOOT_CFG[0]** being locked after write, it's necessary to specify the whole value, as it will be programmed only once.
5. **Reset the board**. This is necessary because:
   - Shadow registers cannot be updated or written twice, they can be set to a “clean” processor only.
   - If the fuse for QSPI reset pin is not burnt, it's necessary to reset the FLASH manually before each configuration.
6. Deselect **Enable security** checkbox, and ensure the **Use Shadow Registers** checkbox is selected.
7. Click **Write Image**.

During the write operation, the following steps are performed:

1. Fuses are checked to ensure the board is in an unsecured mode.
2. 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 processor, and during this process the following actions will be done:
   - Configure external FLASH
   - Erase FLASH (key store is preserved, if it is used)
   - Create FCB block at the beginning of the FLASH
   - Write encrypted application
   - Write OTFAD key blobs

NOTE

Repetitive Write to QSPI flash might fail in case the board is not Reset, see "Booting OTFAD Encrypted Image Using Shadow Registers" steps.

7.4.3.4 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's also recommended to back up all keys. The booting process is identical to the the process described previously with only one difference in write operation - the **Enable security** checkbox 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 list of fuses groups to be burnt.

NOTE

Detailed info about the modification of fuses can be reviewed in OTP/PFR Configuration.

NOTE

Repetitive Write to QSPI flash might fail in case the board is not Reset, see "Booting OTFAD Encrypted Image Using Shadow Registers" steps.

7.4.3.5 Full Security

To enable full security on RTxxx processors, *DCFG_CC_SOCU* and *DCFG_CC_SOCU_AP* fuses must be burnt.

SEC does not set these fuses up for burning by default (even when **Enable security** checkbox is selected), but you can configure it in **OTP Configuration**.

Note that once the *DCFG_CC_SOCU* fuses are set, it's 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 backup and restore keystore during erasing the FLASH in SB file.

7.4.4 Flowcharts

Following flowcharts display the process of generation and provisioning of RTxxx processors in sequence, with focus on input, output and temporary files and their location in the workspace.

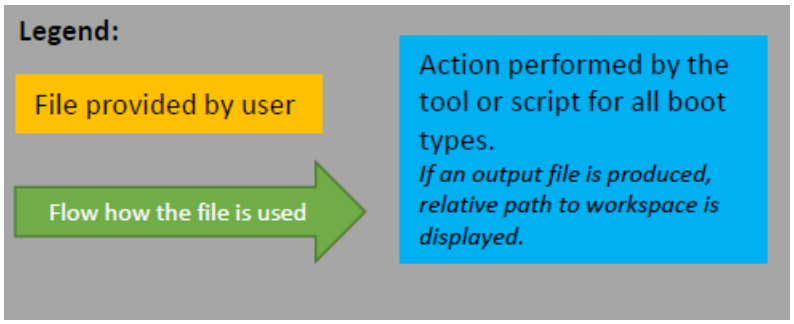

Figure 72. Legend

7.4.4.1 Unsigned

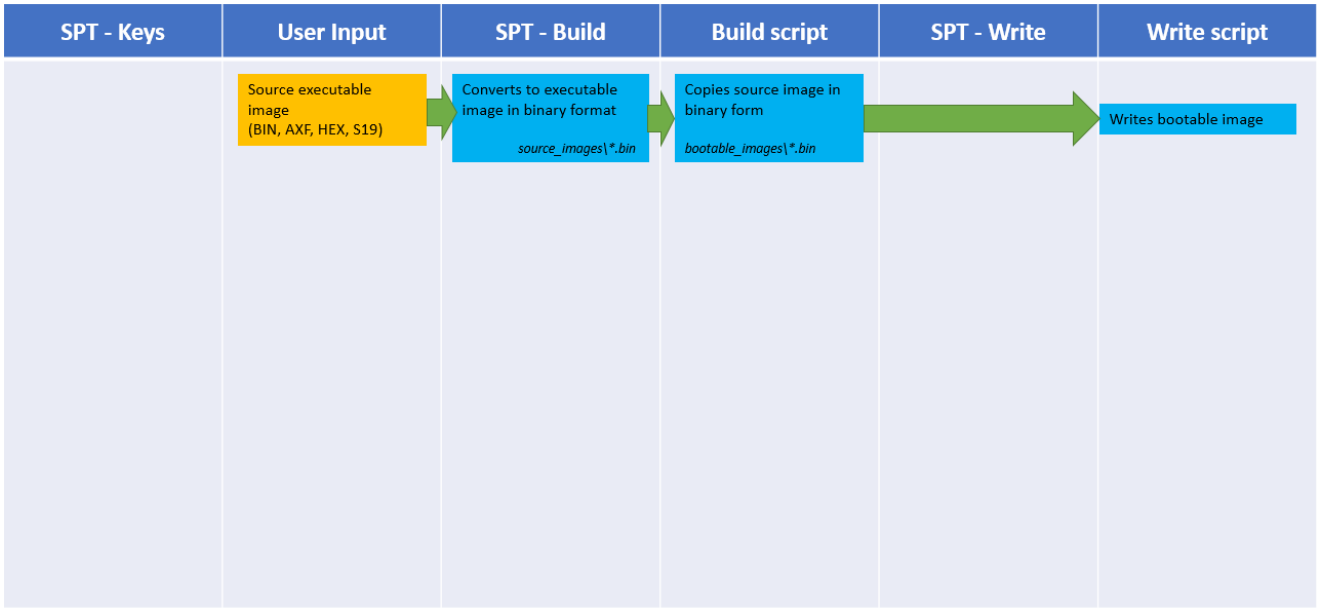

Figure 73. RTxxx unsigned

7.4.4.2 Unsigned with CRC

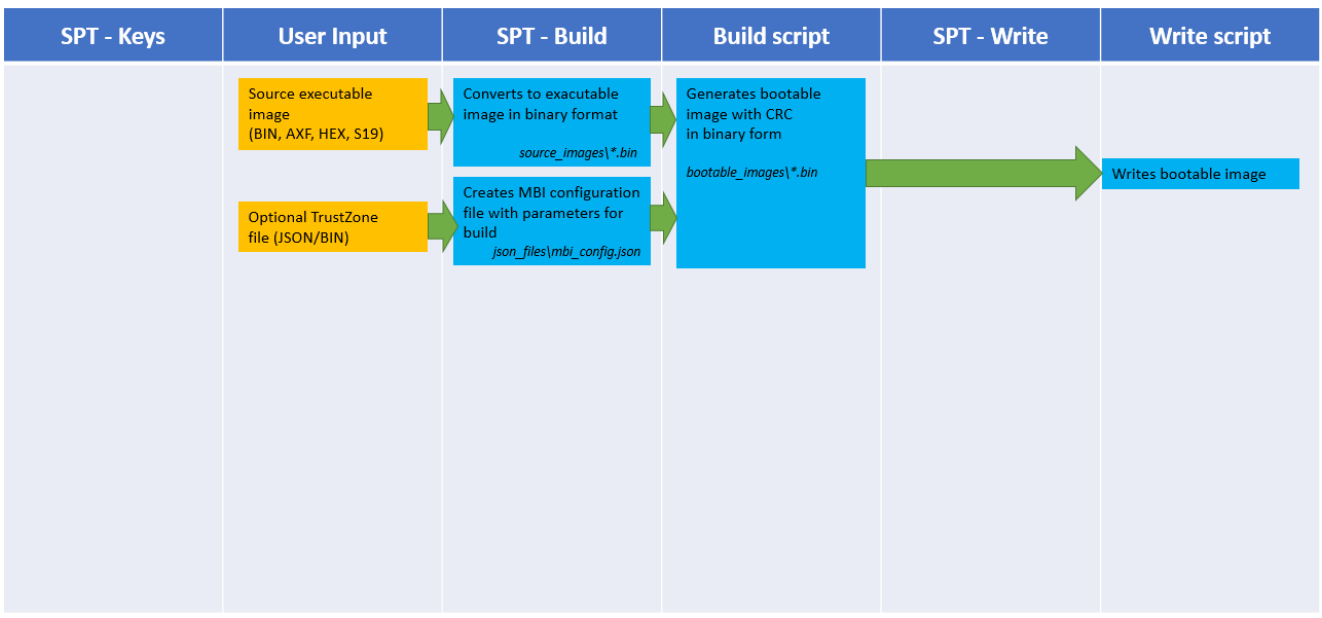

Figure 74. RTxxx unsigned with CRC

7.4.4.3 Signed

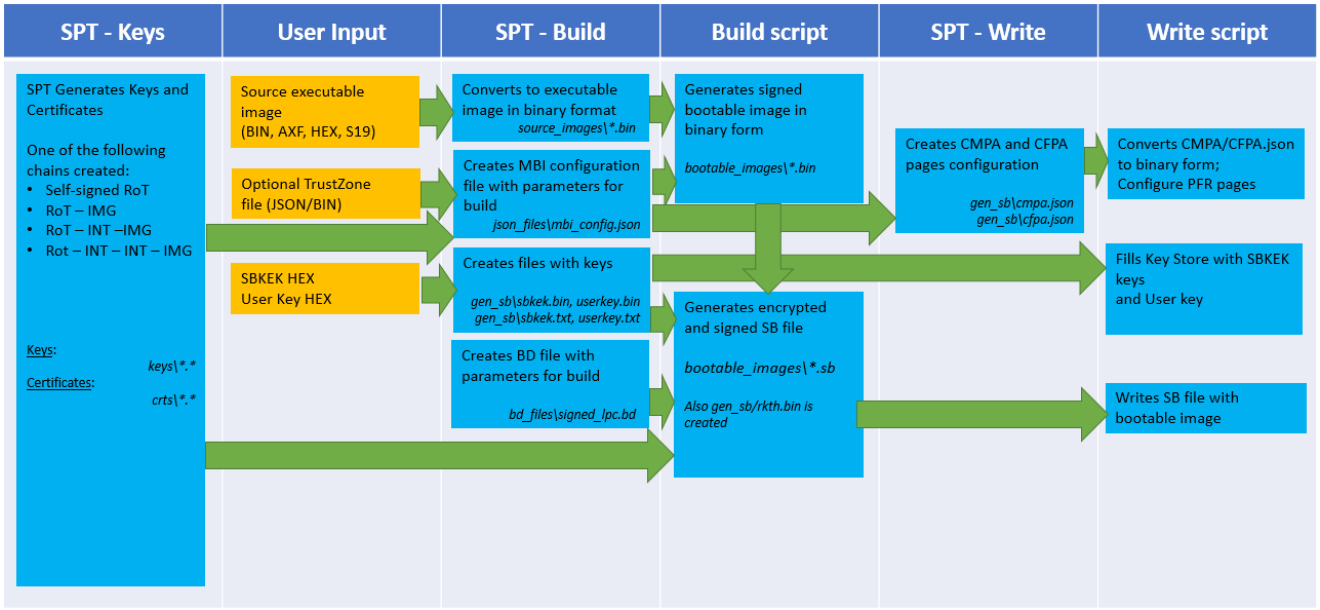

Figure 75. RTxxx signed

7.4.4.4 OTFAD

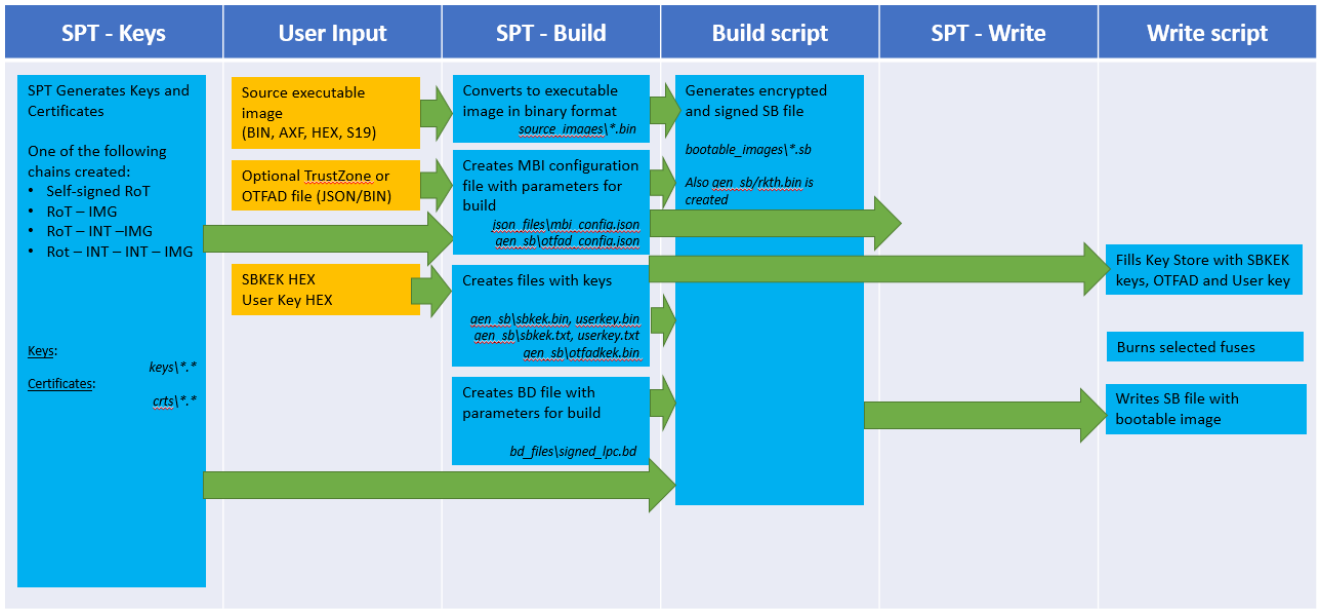

Figure 76. RTxxx: OTFAD Encrypted (User Keys) with CRC/Signed

Chapter 8
Command Line Operations

SEC also offers a command-line interface, enabling integration in automated enviroments 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_v3/bin/securep.exe -h
```

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

```
c:/nxp/MCUX_Provi_v3/bin/securep.exe <command> -h
```

NOTE

The location of the file is subject to installation folder.

8.1 Build

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

8.1.1 Build Arguments

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

Table 9. 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.
--target-image TARGET_IMAGE	Target image for building the boot image.
--secret-key-type {AES-128,AES-192,AES-256}	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's 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 will be imported.
--csf-cert CSF_CERT	Path to the public CSF key file that is used for signing the image. If not specified then it's derived from the img-cert path name according to HAB4 PKI Tree naming convention.
--dcd DCD	Path to the Device Configuration Data binary file.
--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.

Table continues on the next page...

Table 9. Build-specific arguments (continued)

Argument	Description
--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.
-v, --verbose	Increase output verbosity
--device {See Note for supported strings}	Target processor
--boot-type {unsigned,unsigned_with_crc,signed_sb,authenticated_hab,encrypted_hab,xip_encrypted_bee_otpmk_key,xip_encrypted_bee_user_keys_unsigned,xip_encrypted_bee_user_keys_authenticated,prince_encrypted_crc,prince_encrypted_signed,otfad_encrypted_user_keys_unsigned,otfad_encrypted_user_keys_crc,otfad_encrypted_user_keys_authenticated}	Secure boot 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.
--trust-zone TRUST_ZONE	Either 'disabled' (if the configuration is not part of the build image) or 'default' (if the default trust zone configuration shall be enabled) or path to the custom trust zone configuration JSON file.
--keysource {OTP, KeyStore}	Key-source for RTxxx secured images
--userkey USERKEY	Key applicable for RTxxx secured images: for OTP key-source it represents master key; for key-store it represents key used for signature
--sbkek SBKEK	64 HEX characters: Key used as key encryption key to handle SB2 file; Needed only for Secure Binary images; If not specified, it will be 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.

Table 10. Boot-device arguments (mutually exclusive)

Argument	Description
--boot-device {Adesto_AT25SF128A,ISSI_IS25LPxxxA_IS25WPxxxA,ISSI_IS26K	Predefined boot device

Table continues on the next page...

Table 10. Boot-device arguments (mutually exclusive) (continued)

Argument	Description
SxxxS,Internal_Flash,SDHC_SDcard_8GB,Winbond_W25Q32JV, Macronix_MX25UM51345G_A, Macronix_MX25UM51345G_B, Macronix_MX30LF2GE8AB, Micron_MT29F4G16ABAFAH4}	
--boot-device-file BOOT_DEVICE_FILE	File with boot device configuration
--boot-device-type {flex-spi-nor,onchip_memory,sdhc_sd_card}	Boot device type. Default predefined boot device of this type is set.

NOTE

Supported processors: LPC55S69 LPC55S66 LPC55S28 LPC55S26 LPC55S16 LPC55S14 LPC55S06 LPC55S04 IMXRT1064 IMXRT1060 IMXRT1050 IMXRT1024 IMXRT1020 IMXRT1015 IMXRT1010 IMXRT1176 IMXRT1175 IMXRT1173 IMXRT1172 IMXRT1171 IMXRT1166 IMXRT1165 MIMXRT533S MIMXRT555S MIMXRT595S MIMXRT633S MIMXRT685S

8.2 Write

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

8.2.1 Write Arguments

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

Table 11. 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
--enable-security {True,False}	Enable/Disable security. Value of argument is 'true' or 'yes' or 'false' or 'no'. The parameter: • Is ignored for unsigned images • Must be specified for authenticated images - HAB security model • Is automatically enabled for encrypted images • Is automatically enabled for signed images - SB security model. • Is ignored if security is already enabled.
--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.
--region0-fac-array REGION0_FAC_ARRAY	BEE OTPMK Key only - FAC protected areas for region [start,length], max number of pairs is 3, comma separated. Example: --region0-fac-array [0x60001000,0x1000],[0x60002000,0xe000]

Table continues on the next page...

Table 11. Write-specific arguments (continued)

Argument	Description
-v, --verbose	Increase output verbosity
--device {See Note for supported strings}	Target processor
--boot-type {unsigned,unsigned_with_crc,signed_sb,authenticated_hab,encrypted_hab,xip_encrypted_bee_otpmk_key,xip_encrypted_bee_user_keys_unsigned,xip_encrypted_bee_user_keys_authenticated,prince_encrypted_crc,prince_encrypted_signed,otfad_encrypted_user_keys_unsigned,otfad_encrypted_user_keys_crc,otfad_encrypted_user_keys_authenticated}	Secure boot 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.
--version	Display tool version

Table 12. Boot-device arguments (mutually exclusive)

Argument	Description
--boot-device {Adesto_AT25SF128A,ISSI_IS25LPxxxA_IS25WPxxxA,ISSI_IS26KSxxxS,Internal_Flash,SDHC_SDcard_8GB,Winbond_W25Q32JV, Macronix_MX25UM51345G_A, Macronix_MX25UM51345G_B, Macronix_MX30LF2GE8AB, Micron_MT29F4G16ABAFAH4}	Predefined boot device
--boot-device-file BOOT_DEVICE_FILE	File with boot device configuration
--boot-device-type {flex-spi-nor,onchip_memory,sdhc_sd_card}	Boot device type. Default predefined boot device of this type is set.
--otp-cfg OTP_CFG	Path to JSON file with user OTP configuration exported from the OPT Configuration dialog.

Table 13. 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
--baud-rate BAUD_RATE	Connect to target over UART with specified baud rate. --uart argument has to be specified too. Example: --baud-rate 9600

NOTE

Supported processors: LPC55S69 LPC55S66 LPC55S28 LPC55S26 LPC55S16 LPC55S14 LPC55S06 LPC55S04 IMXRT1064 IMXRT1060 IMXRT1050 IMXRT1024 IMXRT1020 IMXRT1015 IMXRT1010 IMXRT1176 IMXRT1175 IMXRT1173 IMXRT1172 IMXRT1171 IMXRT1166 IMXRT1165 MIMXRT533S MIMXRT555S MIMXRT595S MIMXRT633S MIMXRT685S

NOTE

For connection to board, USB or Serial port has to be specified. If nothing is specified, USB auto-detection is applied

8.3 Generate Keys

With the **Generate** command, you can perform actions that you can otherwise perform in the **Generate Keys** view. Compared to GUI, command line functionality is restricted.

8.3.1 Generate Keys Arguments

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

Table 14. Generate-specific arguments

Argument	Description
-h, --help	Show this help message and exit
--keys-cfg KEYS_CFG JSON	File with the keys configuration
--device {See Note for supported strings}	Target processor
--boot-type {unsigned,unsigned_with_crc,signed_sb,authenticated_hab,encrypted_hab,xip_encrypted_bee_otpmk_key,xip_encrypted_bee_user_keys_unsigned,xip_encrypted_bee_user_keys_authenticated,prince_encrypted_crc,prince_encrypted_signed,otfad_encrypted_user_keys_unsigned,otfad_encrypted_user_keys_crc,otfad_encrypted_user_keys_authenticated}	Secure boot 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 15. Boot-device arguments (mutually exclusive)

Argument	Description
--boot-device {Adesto_AT25SF128A,ISSI_IS25LPxxxA_IS25WPxxxA,ISSI_IS26KSxxxS,Internal_Flash,SDHC_SDcard_8GB,Winbond_W25Q32JV. Macronix_MX25UM51345G_A, Macronix_MX25UM51345G_B, Macronix_MX30LF2GE8AB, Micron_MT29F4G16ABAFAH4}	Predefined boot device

Table continues on the next page...

Table 15. Boot-device arguments (mutually exclusive) (continued)

Argument	Description
--boot-device-file BOOT_DEVICE_FILE	File with boot device configuration
--boot-device-type {flex-spi-nor,onchip_memory,sdhc_sd_card}	Boot device type. Default predefined boot device of this type is set.

NOTE

Supported processors: LPC55S69 LPC55S66 LPC55S28 LPC55S26 LPC55S16 LPC55S14 LPC55S06 LPC55S04 IMXRT1064 IMXRT1060 IMXRT1050 IMXRT1024 IMXRT1020 IMXRT1015 IMXRT1010 IMXRT1176 IMXRT1175 IMXRT1173 IMXRT1172 IMXRT1171 IMXRT1166 IMXRT1165 MIMXRT533S MIMXRT555S MIMXRT595S MIMXRT633S MIMXRT685S

8.4 Manufacture

8.4.1 Manufacture Arguments

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

Table 16. 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.
--connections CONNECTIONS [CONNECTIONS …]	List of all connections devices to be used in manufacturing, in format "-p <port>,<baud>" or "-u <usb-path>" or "-u <autodetect-all-USBs>" to find all available USB connections automatically. The option for auto-detection cannot be combined with the others options.

8.5 Clear Security (LPC55Sxx)

With the **clear-security** command, you can reset the security configuration of the device for use in unsigned boot modes. This command works only if the security of the device is not sealed.

NOTE

As of Secure Provisioning Tool 4, this command will no longer be supported. Make sure to exclude it from all scripts to avoid issues.

8.5.1 Clear Security Arguments

Following arguments are available for the **clear-security** command.

Table 17. Clear-security-specific arguments

Argument	Description
-h, --help	Show this help message and exit.

Table continues on the next page...

Table 17. Clear-security-specific arguments (continued)

Argument	Description
--device {LPC55S69, LPC55S66, LPC55S28, LPC55S26, LPC55S16, LPC55S14, LPC55S06, LPC55S04}	Target processor
--boot-type {unsigned,unsigned_with_crc,signed_sb}	Secure boot type
--boot-device {Internal_Flash}	Predefined boot device

Table 18. 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

8.6 Command Line Examples

Example: How to build and write an image for configuration stored in *./workspaces/mcuxprovi* in the workspace folder

NOTE

GUI was already used to prepare complete configuration within a workspace (keys generated, build image configured, write image configured).

```
securep.exe -w /workspaces/mcuxprovi build
```

```
securep.exe -w /workspaces/mcuxprovi write
```

For detailed examples, use the following command:

```
securep.exe print-cli-examples
```

8.7 Command Line Tools

SEC uses the following command line tools to generate keys and build/write the image:

blhost Communication with bootloader (legacy, replaced by spsdk)

cst Key generation and image encoding

elftosb Image build

ide_utils ARM utilities for parsing of ELF format

image_enc Image encryption

openssl Key generation

sdphost Communication with ROM bootloader on RT10xx (legacy, replaced by spsdk)

spsdk Secure Provisioning SDK, for more information see https://spsdk.readthedocs.io/.

Following tools are available as part of SPSDK:

blhost Replacement for the legacy blhost tool

sdphost	Replacement for the legacy sdphost tool
pfr	Generating Protected Flash Region files (cmpa/cfpa)

Chapter 9
Revision History

Table 19. Revision history

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

How To Reach Us :

Home Page:

nxp.com

Web Support:

nxp.com/support

Information in this document is provided solely to enable system and software implementers to use NXP products. There are no express or implied copyright licenses granted hereunder to design or fabricate any integrated circuits based on the information in this document. NXP reserves the right to make changes without further notice to any products herein.

NXP makes no warranty, representation, or guarantee regarding the suitability of its products for any particular purpose, nor does NXP assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages. “Typical” parameters that may be provided in NXP data sheets and/or specifications can and do vary in different applications, and actual performance may vary over time. All operating parameters, including “typicals,” must be validated for each customer application by customer's technical experts. NXP does not convey any license under its patent rights nor the rights of others. NXP sells products pursuant to standard terms and conditions of sale, which can be found at the following address: nxp.com/SalesTermsandConditions.

While NXP has implemented advanced security features, all products may be subject to unidentified vulnerabilities. Customers are responsible for the design and operation of their applications and products to reduce the effect of these vulnerabilities on customer’s applications and products, and NXP accepts no liability for any vulnerability that is discovered. Customers should implement appropriate design and operating safeguards to minimize the risks associated with their applications and products.

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

For sales office addresses, please send an email to: salesaddresses@nxp.com

Date of release: 28 July 2021

Document identifier: MCUXSPTUG

arm