NXP Semiconductors
User Guide

Document identifier: MCUOTASBLSFWUG
Rev. 1.1.0, 08 November 2021

MCU-OTA SBL and SFW User Guide

Contents

Chapter 1
Introduction

This document provides a complete description of Secure Bootloader (SBL) features, project framework, quick start, and the various software settings. It describes FOTA in detail, including image programming, switch, revert, signature, encryption, and so on. Security is very important and is described based on NXP MCU SoC secure engines. It includes detailed steps to program images via MCU ISP (UART/USB). Other necessary information can be found in the document for convenient understanding and developing.

Secure Firmware (SFW) was created based on FreeRTOS and developed to implement the complete FOTA process along with SBL. SFW supports obtaining the OTA firmware image by U-Disk, SD card in local, or AWS cloud, Aliyun cloud in the remote. Then, SBL checks, authenticates the OTA firmware image, and boots it up in normal.

1.1 Acronyms and abbreviations

The following table lists the acronyms used in this document.

Table 1. Acronyms and abbreviations

Term	Description
AES	Advanced Encryption Standard
Aliyun	Alibaba cloud
AWS	Amazon Web Services
BEE	Bus Encryption Engine
CAAM	Cryptographic Accelerator and Assurance Module
CRC	Cyclic Redundancy Check
DCP	Data Co-Processor
FOTA	Firmware Over-The-Air
HAB	High Assurance Boot
LWIP	Lightweight TCP/IP stack
MCU ISP	MCU In-System programming
MQTT	Message Queuing Telemetry Transport
OCOTP	On-Chip One Time Programmable
OTA	Over-The-Air
OTFAD	On-The-Fly AES Decryption
OTPMK	One-Time Programmable Master Key

Table continues on the next page...

Table 1. Acronyms and abbreviations (continued)

Term	Description
RTOS	Real-time operating system
SB	NXP MCU Secure Binary
SBL	Secure Bootloader
SFW	Secure Firmware
SHA	Secure Hash Algorithms
SKB	Secure Kinetis bootloader
SNVS	Secure Non-Volatile Storage
TRNG	True Random Number Generator
XIP	eXecute In Place

1.2 About MCU SBL and SFW

MCU SBL and SFW are C code projects for secure OTA, they support local OTA via UART, USB, or remote OTA via Ethernet, WIFI, and others, and can provide a complete secure trust chain. The Host Tool makes it convenient to program image via UART/USB interface, sign and encrypt image, manage the eFuse and create an SB/SB2 binary.

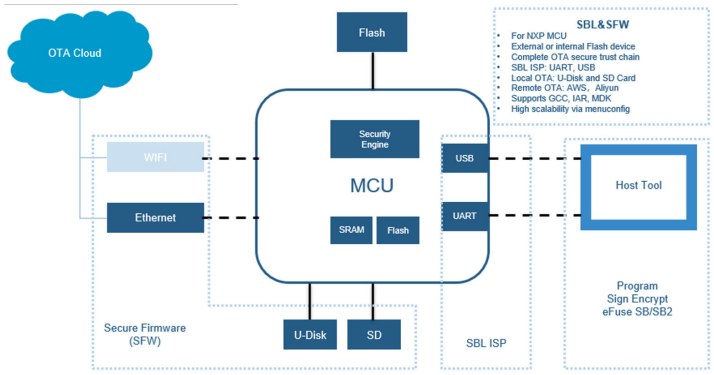


Figure 1. SBL and SFW diagram

Figure 2 shows detailed information about the SBL architecture, and the relationship with Firmware and Host Tool. It includes all the possible modules in the project. When building a specific SBL image for one MCU platform, the project can (should) be easily configured based on SoC features.

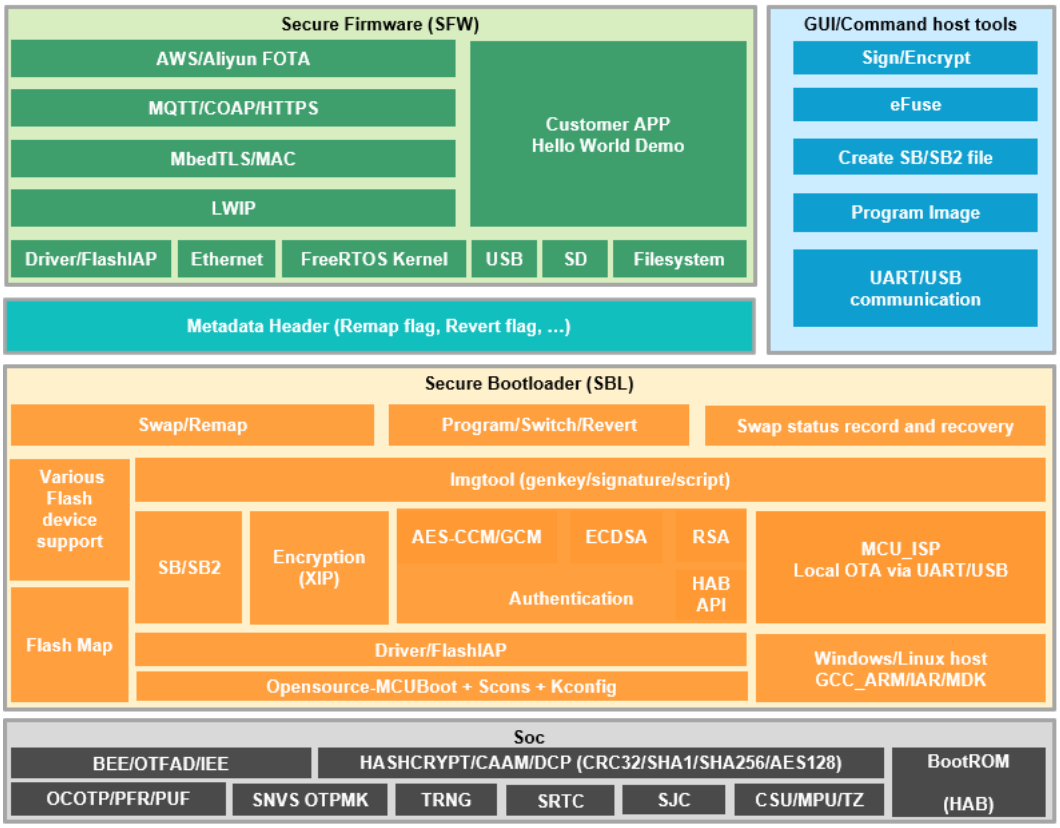


Figure 2. MCU-OTA project architecture

1.3 Features

- Support NXP MCU platforms (Table 2 lists the platforms by now), uniform code, and architecture for all platforms.
- Complete OTA secure trust chain, support SoC secure engine for signature and encryption
- Single image or swap images OTA feature
- Support SoC remap to reduce Flash erase/program
- Local OTA: UART, USB communication (SBL)
- Local OTA: SD card, U-Disk (SFW)
- Remote OTA: AWS, Aliyun (SFW)
- Minimal MCU system resource requirement
- Support external or internal flash device
- Support multiple toolchains and developing environment:
  - Linux host: GCC_ARM
  - Windows host: GCC_ARM, IAR, MDK
  - Conveniently create IAR, MDK project by SCons extended command
- High and easy scalability via Kconfig mechanism
- Following the OTA process from open-source MCUboot project

1.4 Supported MCU boards

The following table lists the NXP MCU boards supported by SBL and SFW.

Table 2. Supported NXP MCU boards

Board	Architecture	Boot Device	Security		SBL			SFW OTA			
			Signature	Encryption	ISP	Swap	Remap	U-Disk	SD card	AWS	Aliyun
evkmimxrt1010	CM7	QSPI Flash	•	•	•		•	•			
evkmimxrt1020	CM7	QSPI Flash	•	•	•	•		•	•	•	•
evkbmimxrt1050	CM7	Hyper Flash	•	•	•	•		•	•	•	•
evkmimxrt1060	CM7	QSPI Flash	•	•	•		•	•	•	•	•
evkmimxrt1064	CM7	QSPI Flash	•	•	•		•	•	•	•	•
evkmimxrt1170	CM7+CM4	QSPI Flash	•	•	•		•	•	•	•	•
evkmimxrt500	CM33+F1	Octal Flash	•	•	•		•	•	•		
evkmimxrt600	CM33+HiFi4	Octal Flash	•	•	•		•	•	•		
lpc55s69	CM33+CM33	Internal Flash	•	•	•	•		•	•		

For detailed platform information, refer to the following documents.

- MIMXRT1010 EVK Board Hardware User's Guide
- MIMXRT1020 EVK Board Hardware User's Guide
- MIMXRT1050 EVK Board Hardware User's Guide
- MIMXRT1060 EVK Board Hardware User's Guide
- MIMXRT1064 EVK Board Hardware User's Guide
- MIMXRT1170 EVK Board Hardware User's Guide
- MIMXRT500 EVK Board Hardware User's Guide
- MIMXRT600 EVK Board Hardware User's Guide
- LPC55S69 EVK Board Hardware User's Guide

1.5 SBL and SFW organization

SBL and SFW projects are constructed with source code, SCons tool, Kconfig scripts, Python scripts, Windows executable files and documents. The layer and description of SBL are showed in the Figure 3 and Table 3. The layer and description of SFW are showed in the Figure 4 and Table 4.

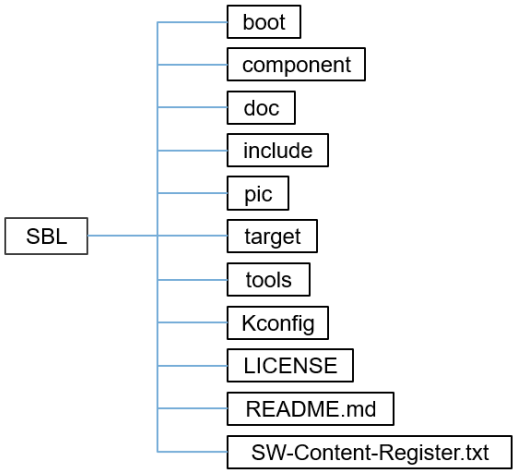


Figure 3. SBL directory organization

Table 3. SBL source code directories

Directory	Description
boot	Source code of MCUboot partly from open source
component	It includes SDK components and board peripheral drivers, for example, flash IAP and UART drivers
doc	Documents of SBL project
include	Header files of SBL project
pic	Pictures used by README.md
target	All supported platforms: RT1010, RT1020, RT1050, RT1060, RT1064, RT1170, RT500, RT600, LPC55S69
tools	Tools used to build and configure the project
Kconfig	Script file of menuconfig tool
LICENSE	Apache License
README.md	Introduction to SBL project
SW-Content-Register.txt	Used for license check of SBL project

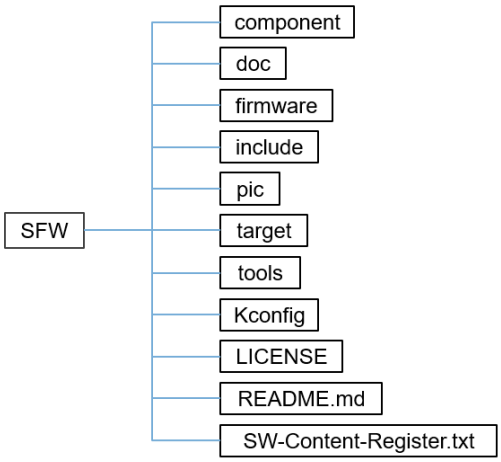


Figure 4. SFW directory organization

Table 4. SFW source code directories

Directory	Description
component	It includes SDK components and board peripheral drivers, for example, flash IAP and UART drivers
doc	Documents of SFW project
firmware	It includes OTA related source code, for example, FreeRTOS, AWS, Aliyun
include	Header files of SFW project
pic	Pictures used by README.md
target	All supported platforms: RT1010, RT1020, RT1050, RT1060, RT1064, RT1170, RT500, RT600, LPC55S69
tools	Tools used to build and configure the project
Kconfig	Script file of menuconfig tool
LICENSE	Apache License
README.md	Introduction to SFW project
SW-Content-Register.txt	Used for license check of SFW project

1.6 Host system requirements

SBL and SFW projects can be developed in both Linux host and Windows host. The system requirements are as below:

- Linux host
  - Git
  - Python 3.6
  - SCons

  - GCC_ARM toolchain
  - Library: ncurses5-dev
- Windows host
  - Git bash
  - GCC_ARM toolchain
  - or IAR IDE v8.40
  - or MDK IDE v5.30

Chapter 2
Quick start

This chapter introduces the quick start for SBL and SFW projects. Sections 2.1 and 2.2 introduce the quick start for the SBL project, while section 2.3 introduces the quick start for the SFW project. Use the EVKMIMXRT1170 platform as example.

2.1 Windows host

On the Windows host, three toolchains can be selected to build SBL: GCC_ARM, IAR, and MDK.

2.1.1 GCC_ARM

First, obtain the GCC_ARM toolchain from the Arm or MinGW website and install it to the Windows host.

1. Clone SBL project and checkout to v1.1.0, or download the release package

   `git clone https://github.com/NXPmicro/sbl.git`.

2. Enter the directory `sbl/target/evkmimxrt1170/`.
3. Double-click the batch file `env.bat`.
4. Configure the `evkmimxrt1170` project.

   In `env.bat`, run the `scons --menuconfig` command. Then the SBL configuration menu is generated.

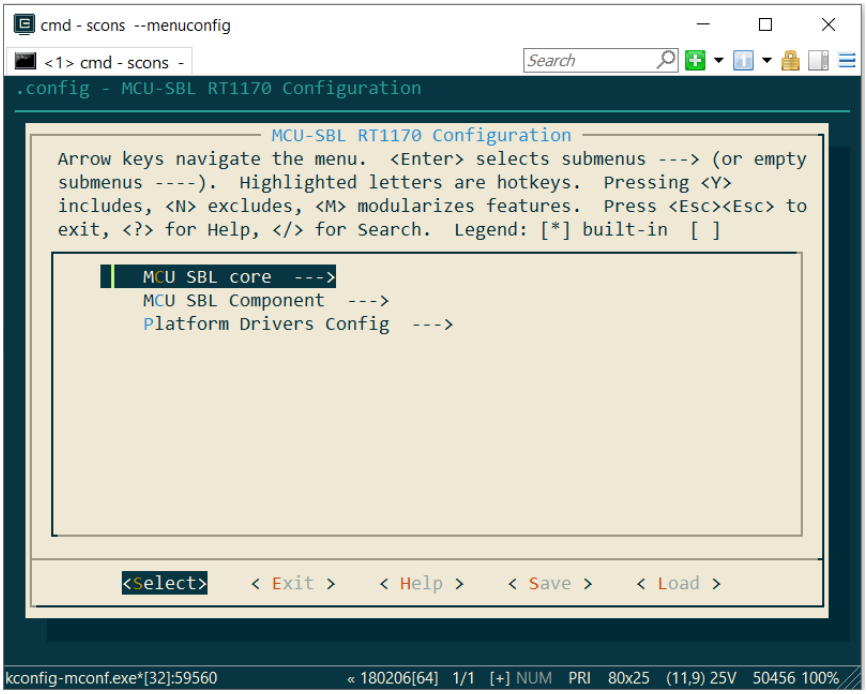


Figure 5. SBL configuration menu

Configure SBL project according to a specific platform and specific application. After the configuration is completed, save the configuration and exit the menu.

5. Build and download the SBL project.

   a. Set `EXEC_PATH` as the gcc toolchain install path in the `sblprofile.py` file for `gcc CROSS_TOOL`:

```
if  CROSS_TOOL == 'gcc':
    PLATFORM    = 'gcc'
    EXEC_PATH   = r'/opt/share/toolchain/gcc-arm-none-eabi-9-2019-q4-major/bin'
```

Figure 6. EXEC_PATH in `sblprofile.py`

For example, in my windows, the gcc toolchain install path is `C:\Program Files (x86)\GNU Tools Arm Embedded\9 2019-q4-major\bin`. Set the `EXEC_PATH` as below:

```
EXEC_PATH = r'C:\Program Files (x86)\GNU Arm Embedded Toolchain\9 2020-q2-update\bin'
```

Alternatively, `SBL_EXEC_PATH` can be added to the Windows environment variable to cover the `EXEC_PATH`. For SFW, the environment variable is `SFW_EXEC_PATH`.

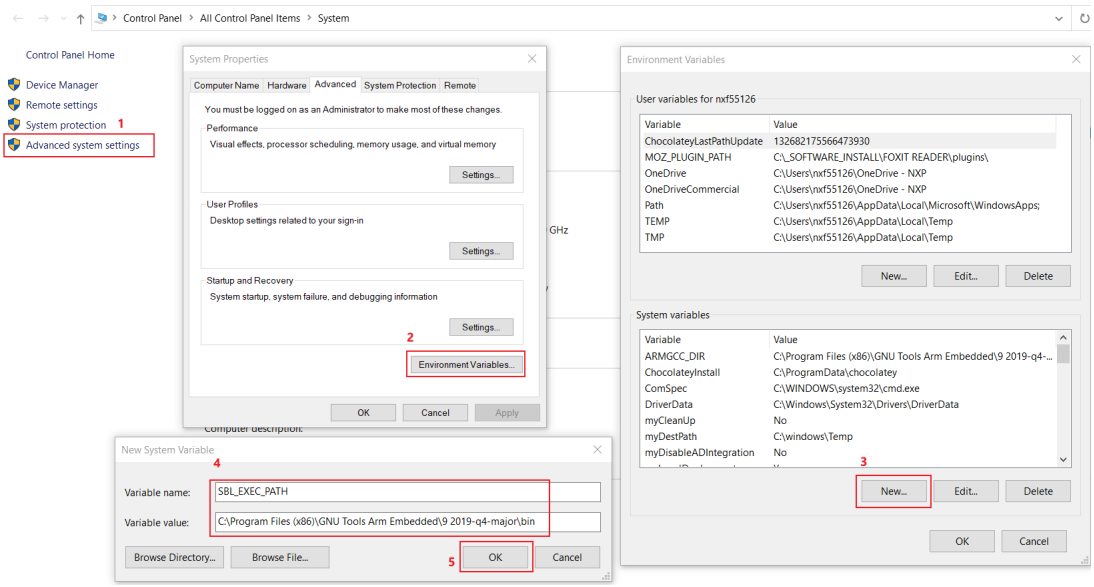


Figure 7. Windows environment variable `SBL_EXEC_PATH`

b. Build the project

   In the `env.bat` file, use the `scons` command to build the project.

   If successfully built, the `sbl.bin` image is built in the `sbl/target/evkmimxrt1170/build` directory.

c. Download the project

   i. Use a micro USB cable to connect the EVKMIMXRT1170 board to the computer.

   ii. Set the board to serial download mode.

   iii. Use the DapLink drag-n-drop function or other tools to download the `sbl.bin` image to the board.

   iv. Set the board to XIP mode

   v. Reset the board.

2.1.2 IAR IDE

For the IAR toolchain, the steps of quick start are as below:

Step 1~ Step 4 are the same as in section 2.1.1

Step 5: Build and download the SBL project.

1. Create the IAR project for the EVKMIMXRT1170 platform.

In the `env.bat` file, use the `scons --ide=iar` command to generate the IAR project.

2. Enter the directory: `sbl/target/evkmimxrt1170/iar`
3. Double-click the IAR project file: `sbl.eww`
4. Click the **Make** button to build the project.
5. Use a micro USB cable to connect the EVKMIMXRT1170 board to the computer, set the board to serial download mode. To download the project to the board, click the **Download** button. After the image is successfully downloaded into the board, set the board to XIP mode, then reset the board.

2.1.3 MDK IDE

For the MDK toolchain, the steps of quick start are as below:

Step 1~ Step 4 are the same as in section 2.1.1

Step 5: Build and download the SBL project.

1. Create the MDK project for the EVKMIMXRT1170 platform.

   In the `env.bat` file, use the `scons --ide=mdk5` command to generate the MDK project.

2. Enter the directory: `sbl/target/evkmimxrt1170/mdk`.
3. Double-click the MDK project file: `sbl.uvprojx`.
4. Click the **Build** button to build the project.
5. Use a micro USB cable to connect the EVKMIMXRT1170 board to the computer, set the board to serial download mode. To download the project to the board, click the **Download** button. After the image is successfully downloaded to the board, set the board to XIP mode, then reset the board.

NOTE

The following errors may be encountered when building SBL project with MDK.

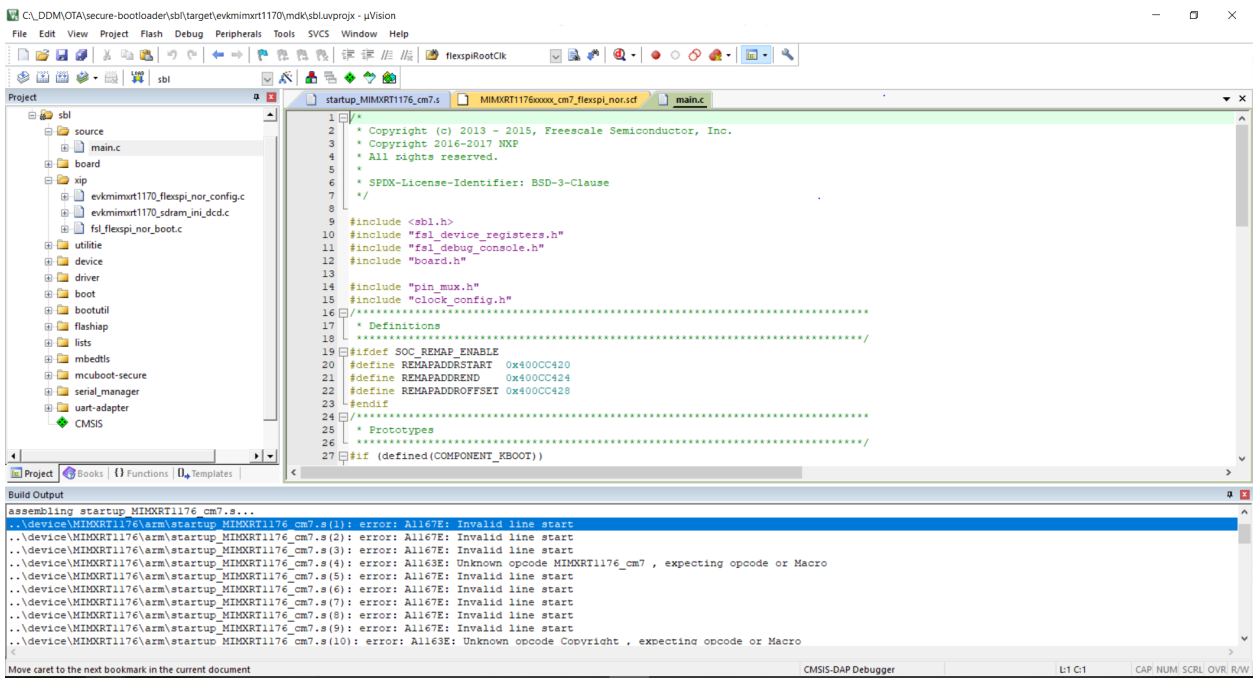


Figure 8. Compile errors

It can be solved in the following ways:

- With MDK version 5.30 (or later)

  Configure the project for the Assembler Option: armclang (GUN Syntax).

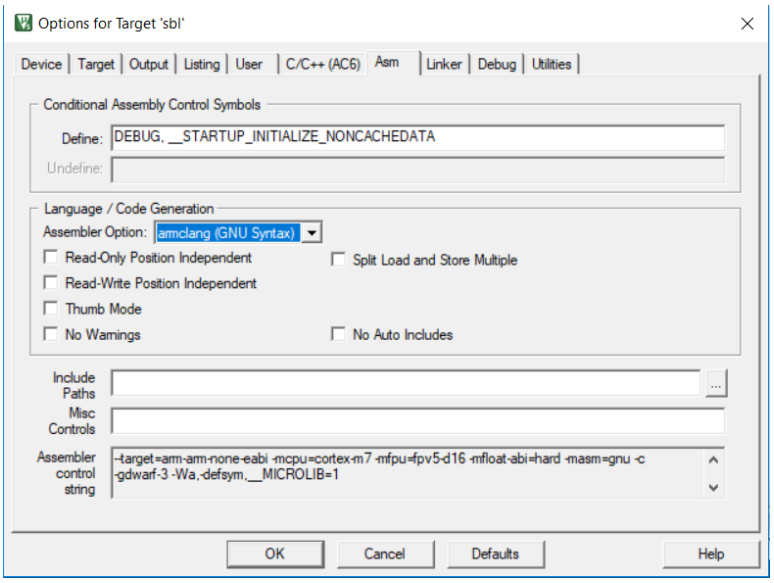


Figure 9. Assembler Option

- With an earlier version than MDK 5.30
  1. Select the option **Assemble** by using ArmClang V6
  2. Configure Misc Controls to `-masm=auto`

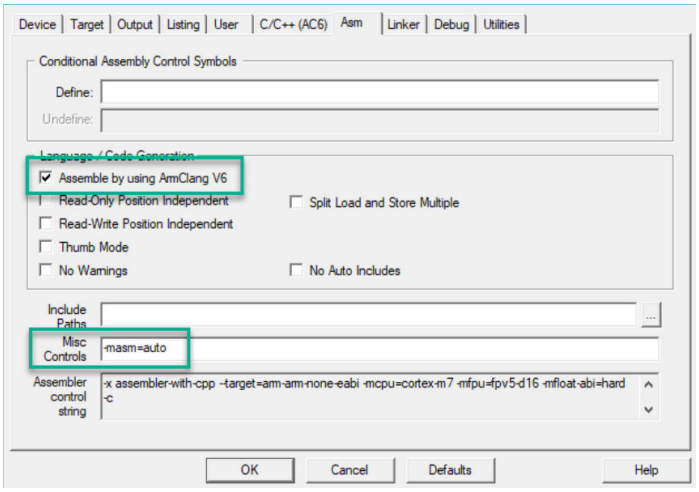


Figure 10. Asm Option

2.2 Linux host

On Linux host, the steps of quick start are as below:

1. Install SCons

   For Ubuntu or Debian, use the command:

   ```
   $ sudo apt-get install scons
   ```

For RPM-based (Red Hat, SUSE, Fedora …), use the command:

```
$ sudo yum install scons
```

2. Install the GCC_ARM toolchain like `gcc-arm-none-eabi-9-2019-q4-major`.
3. Clone the SBL project and checkout to v1.1.0, or download the release package

   ```
   $ git clone https://github.com/NXPmicro/sbl.git
   ```

4. Switch to the `evkmimxrt1170` directory

   ```
   $ cd target/evkmimxrt1170
   ```

5. Set `EXEC_PATH` as gcc toolchain install path in `sblprofile.py` file for `gcc CROSS_TOOL`:

```
if   CROSS_TOOL == 'gcc':
     PLATFORM    = 'gcc'
     EXEC_PATH   = r'/opt/share/toolchain/gcc-arm-none-eabi-9-2019-q4-major/bin'
```

Figure 11. `EXEC_PATH` in `sblprofile.py`

Alternatively, `SBL_EXEC_PATH` can be added to the Linux environment variable to cover the `EXEC_PATH`. For SFW, the environment variable is `SFW_EXEC_PATH`.

6. Configure the `evkmimxrt1170` project

   ```
   $ scons --menuconfig
   ```

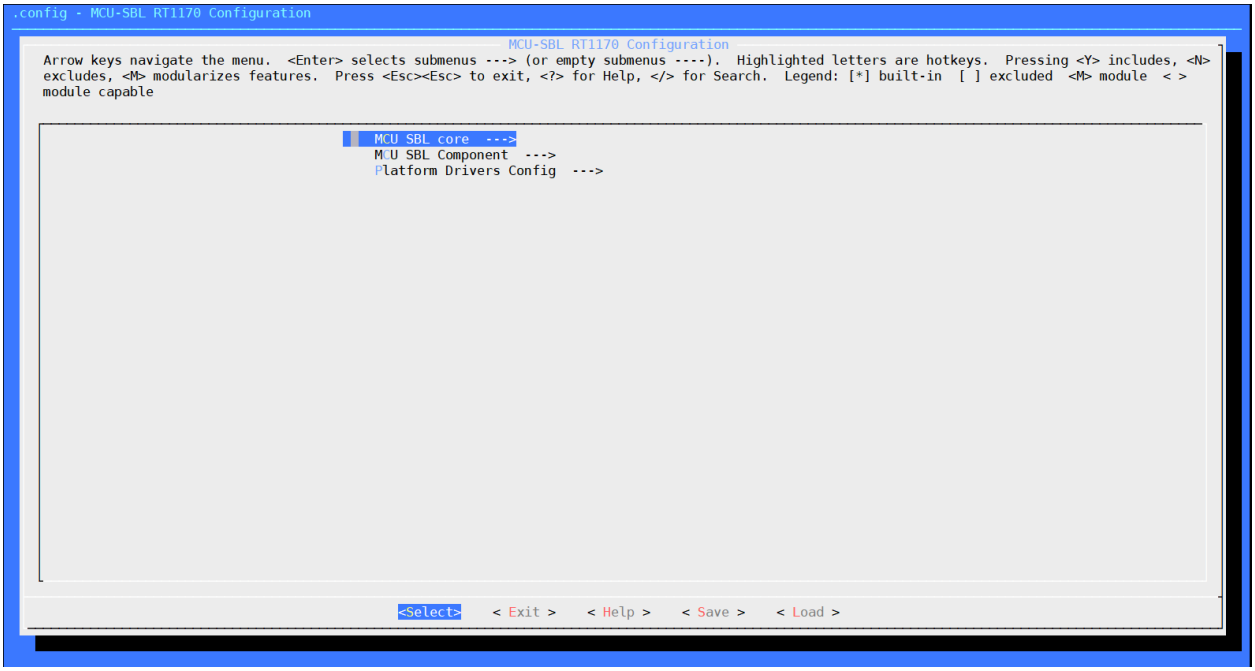


Figure 12. SBL configuration menu

In this menu, configure the SBL project according to the platform and specific bootloader case, such as enabling the single image function or not. After the configuration is completed, save the configuration and exit the menu.

7. Build the image with GCC_ARM toolchain

   ```
   $ scons
   ```

   The `sbl.bin` image is built in `sbl/target/evkmimxrt1170/build` directory.

8. Program the image.

Use a micro USB cable to connect the EVKMIMXRT1170 board to the computer, and program `sbl.bin` by DapLink drag-n-drop or other tools. Set the board to XIP boot mode, and reset to start up the SBL.

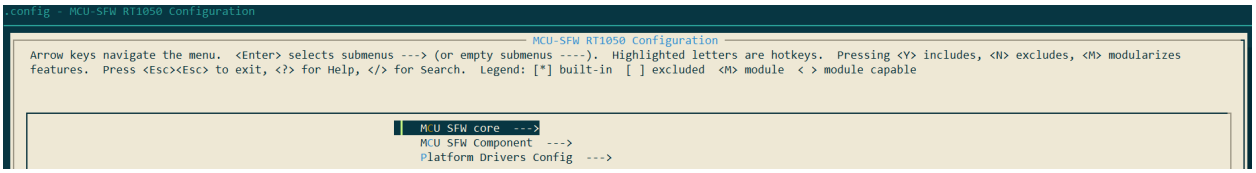
2.3 SFW

The architecture of the SFW project is similar to the SBL project, so quick start steps are the same as the SBL introduced in section 2.1 and section 2.2 except for the following steps:

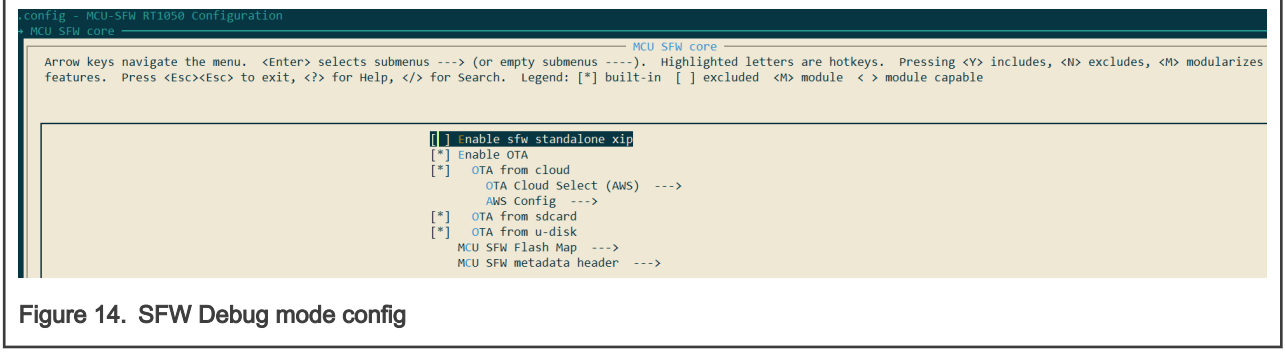
1. Clone the SFW project and checkout to v1.1.0, or download the release package

   ```
   git clone https://github.com/NXPmicro/sfw.git
   ```

2. SFW supports two debug modes: SFW project XIP separately or SFW generates the bin file used with SBL. SFW configures which mode to use via `scons --menuconfig`.



Then select `MCU SFW core`, in the MCU SFW core menu, if the `Enable sfw standalone xip` option is selected, the SFW project will XIP separately. If the `Enable sfw standalone xip` option is not selected, SFW generates the bin file which is used with SBL.



Chapter 3
Framework

This chapter introduces the build framework of SBL and SFW. SBL and SFW projects are built by SCons software construction tool and are configured by the Kconfig file.

3.1 SCons

This section gives the specifics of the SCons software construction tool.

3.1.1 Overview

SCons is an open-source build system written in Python, similar to GNU Make. However, it uses SConstruct and SConscript files instead of usual Makefile files. These files are also Python scripts and can be written using standard Python syntax. Thus, in SConstruct and SConscript files, the Python standard library can be called to perform various complex processing, not limited to the rules set by the Makefile.

SCons and Python tools should be installed before using them. On Windows host, there is no need to install these SCons and Python because the Env configuration tool in SBL comes with them. On Linux host, Python should be installed by default, and SCons can be installed following the command in section 2.1.

3.1.2 SConscript and SConstruct

SCons uses SConscript and SConstruct files to organize the source code structure.

The following three files exist in each SBL and SFW platform directory: `sblconfig.py` (for SBL) or `sfwconfig.py` (for SFW), SConstruct, and SConscript, which controls the compilation of the platform. In general, there is only one SConstruct file in one platform, but there are multiple SConscript files.

The SConscript file can control the addition of source code files and can specify the Group of source code files (similar to the concept of Group in IDEs such as MDK/IAR).

SConscript files also exist in most of the source code folders of SBL and SFW projects. These files are "found" by the SConscript files in the specific platform directory to add the source code corresponding to the macros defined in `sblconfig.h` or `sfwconfig.h` into the compiler.

3.1.3 Basic commands

This section introduces some basic SCons commands. On Windows host, these commands are used in the `env.bat` file of a specific platform in the target directory. On Linux host, these commands are used directly in a specific platform directory.

1. `scons`

   Build the project for a specific platform.

   If some source files are modified after executing the command, when the `scons` command is executed again, SCons performs incremental compilation, and only the modified source files are compiled and linked.

2. `scons --menuconfig`

   Call Kconfig file to configure the project and generate the `sblconfig.h` file.

3. `scons --ide=xxx`

   Generate IAR or MDK projects for a specific platform.

   Use the `scons --ide=iar` command to generate one IAR project.

   Use the `scons --ide=mdk5` command to generate one MDK project.

3.2 Kconfig

The SBL project uses the configuration file `sblconfig.h` generated by the Kconfig file to configure the system, and the SFW project uses `sfwconfig.h`. The Kconfig file is the source file for various configuration interfaces.

All configuration tools generate the configuration interface by reading the Kconfig file in the current platform directory. This file is the total entry for all configurations. It contains Kconfig files in other directories. The configuration tool reads each Kconfig file, generates a configuration interface for developers to configure the system, and finally generates the configuration file `sblconfig.h` of the SBL system and `sfwconfig.h` of the SFW.

When the `scons --menuconfig` command is executed with the env tool or with Linux host in the specific platform (target/xxx/) directory, the configuration interface of the SBL and SFW systems appears, as shown on Figure 15 and Figure 16 .

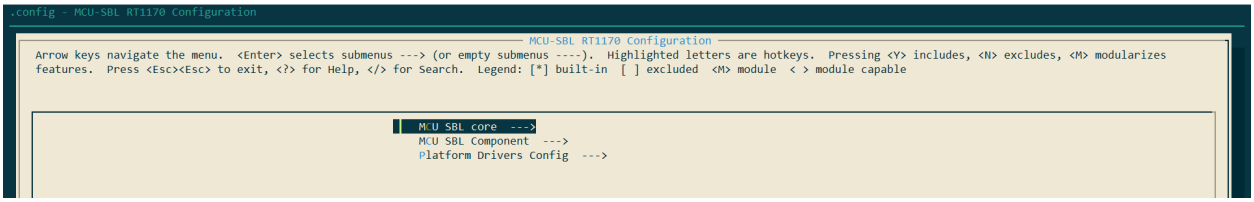


Figure 15. SBL menuconfig menu

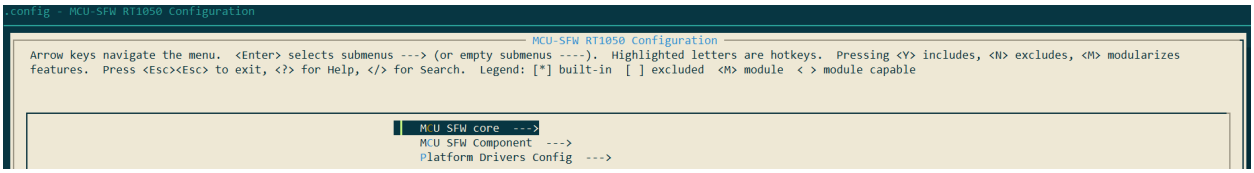


Figure 16. SFW menuconfig menu

In this menu, there are three submenus to select. For example, to select the MCU SBL core, use the submenu below:

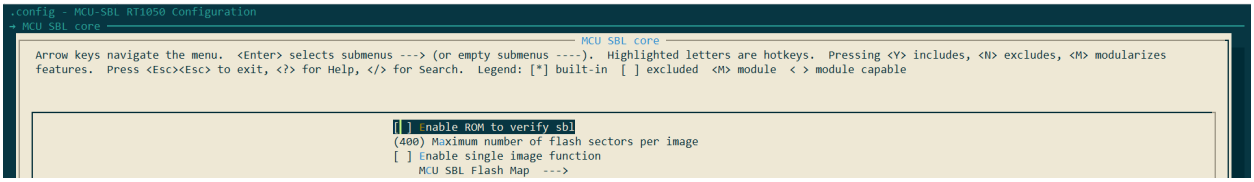


Figure 17. MCU SBL submenu

In this menu, there are some configurable items, press ‘y’ to include the item, and press ‘n’ to exclude the item.

After configuring all items, save the configuration and exit the menu. Then the project can be compiled.

3.3 Host tool

NXP provides various host tools to help with the SBL and SFW developing and testing. Here are three basic tools, for more others, visit NXP official website or contact FAE.

1. MCUXpresso Config Tools

   MCUXpresso Config Tools is an integrated suite of configuration tools that help guide users from first evaluation to production software development. These configuration tools allow developers to quickly build a custom SDK and leverage pins, clocks, and peripheral tools to generate initialization C code for custom board support. In the SBL target platform, there is an `MCUX_Config.mex` file which can be opened by MCUXpresso Config Tools and help to generate specific C code for clocks, pins, and so on. For example: `target/evkbmimxrt1050/board/MCUX_Config/ MCUX_Config.mex`. For more information, refer to the website.

2. Bootloader Host Application (blhost)

   The blhost application is a command-line utility used on the host computer to initiate communication and issue commands to the MCU ISP module over the UART or USB connections. The application only sends one command per invocation. The blhost application supports multi-platforms, including Windows, Linux (X86-based), MACOSX, and Linux (Arm-based). For more information, refer to the website.

3. MCU Boot Utility

   NXP-MCU Boot Utility is a GUI tool specially designed for NXP MCU secure boot. Its features correspond to the BootROM function in NXP MCU. Currently, it mainly supports i.MXRT series MCU chips, Compared to NXP official security enablement toolset (OpenSSL, CST, sdphost, blhost, elftosb, BD, MfgTool2), NXP-MCU Boot Utility is a real one-stop tool, a tool that includes all the features of NXP's official security enablement toolset, and what is more, it supports full graphical user interface operation. With NXP-MCU Boot Utility, it is easy to get started with NXP MCU secure boot. The main features of NXP-MCU Boot Utility include：

   - Support both UART and USB-HID serial downloader modes
   - Support various user application image file formats (elf/axf/srec/hex/bin)
   - Can validate the range and applicability of user application image
   - Support for converting bare image into bootable image
   - Support for loading bootable image into external boot devices
   - Support common boot device memory operation (Flash Programmer)

   For more information about the MCU boot utility, refer to the website.

Chapter 4
MCU ISP

This section descibes the specifics of the MCU ISP.

4.1 About ISP

The MCU ISP provides flash programming utility that operates over a serial connection on the MCUs. It enables quick and easy programming of MCUs. Host-side command line and GUI tools are available to communicate with the SBL device. Users can utilize host tools to upload/download application code and do manufacturing via the MCU ISP.

4.2 Features

- Supports UART and USB peripheral interfaces.
- Supports NXP blhost tool and NXP-MCUBootUtility GUI tool.
- Automatic detection of the active peripheral.
- User-defined timeout for active peripheral detection.
- Autobaud on UART peripheral.
- Protection of RAM used by the SBL while it is running.
- Programming Serial NOR Flash.

4.3 Set ISP timeout

In SBL menuconfig, when MCU ISP support is enabled, ISP timeout can be set, the default timeout value is 5 seconds. If SBL target does not receive the ISP command from host within the timeout period, then ISP process is bypassed.

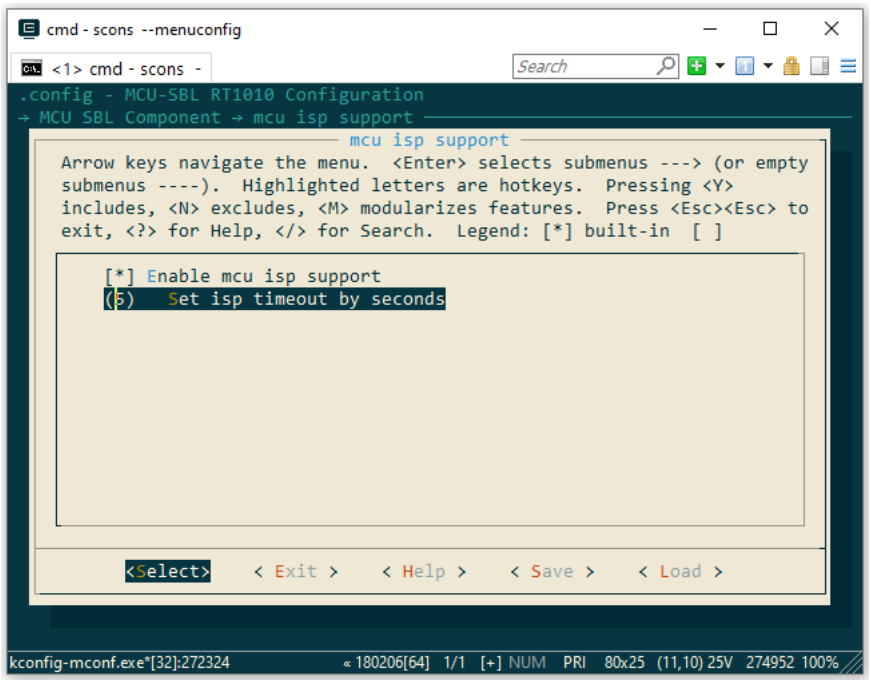


Figure 18. SBL menuconfig set timeout

If the default 5 seconds timeout value is not enough, select this option and edit the timeout value.

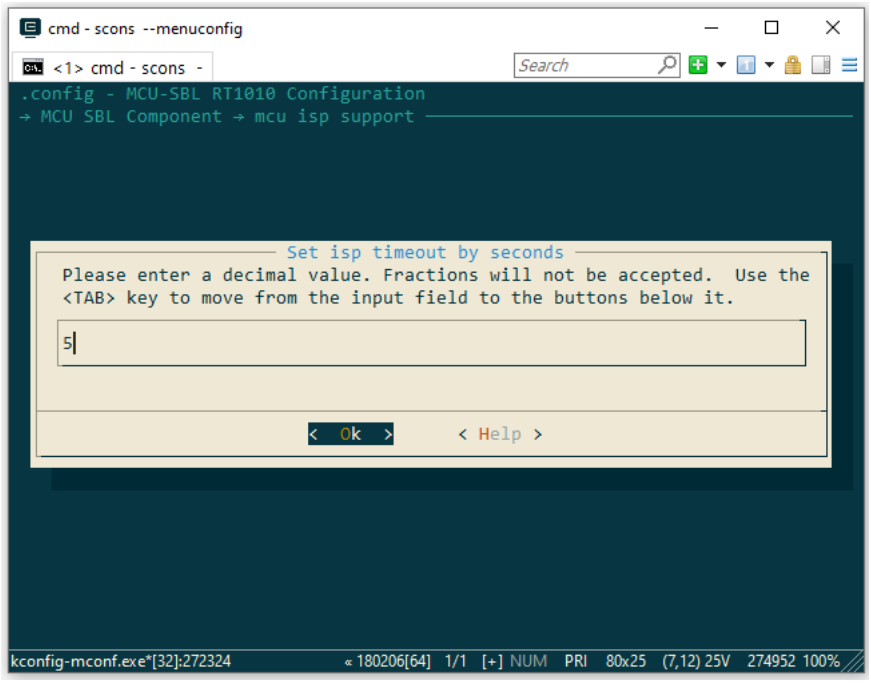


Figure 19. SBL menuconfig set timeout

4.4 MCU Boot Utility usage

The NXP-MCU Boot Utility GUI tool (v3.3 or later) is recommended as the preferred host tool for ISP downloading. The ones who want to use the blhost command-line tool, should contract NXP. For detailed information, see the steps below:

1. Open MCUBootUtility, set mode to `SBL OTA` in menu Tools/Run Mode.

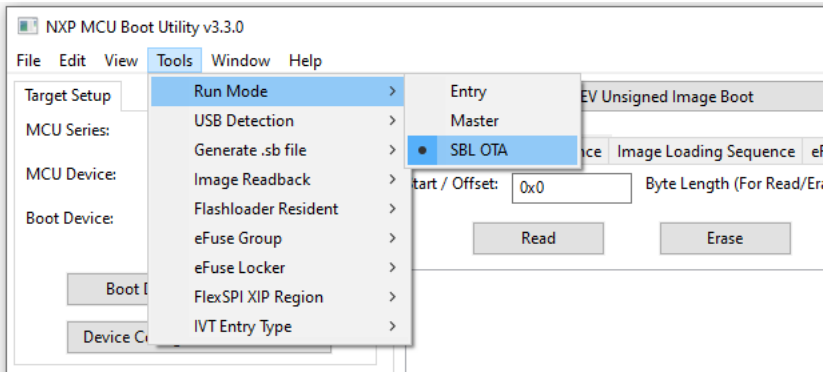


Figure 20. Set tool run mode

2. Power on the SBL target board (take EVKMIMXRT1010 as example), then connect USB cable to J9. If everything is fine, USB vid/pid is detected. Click the **Connect to SBL ISP** button.

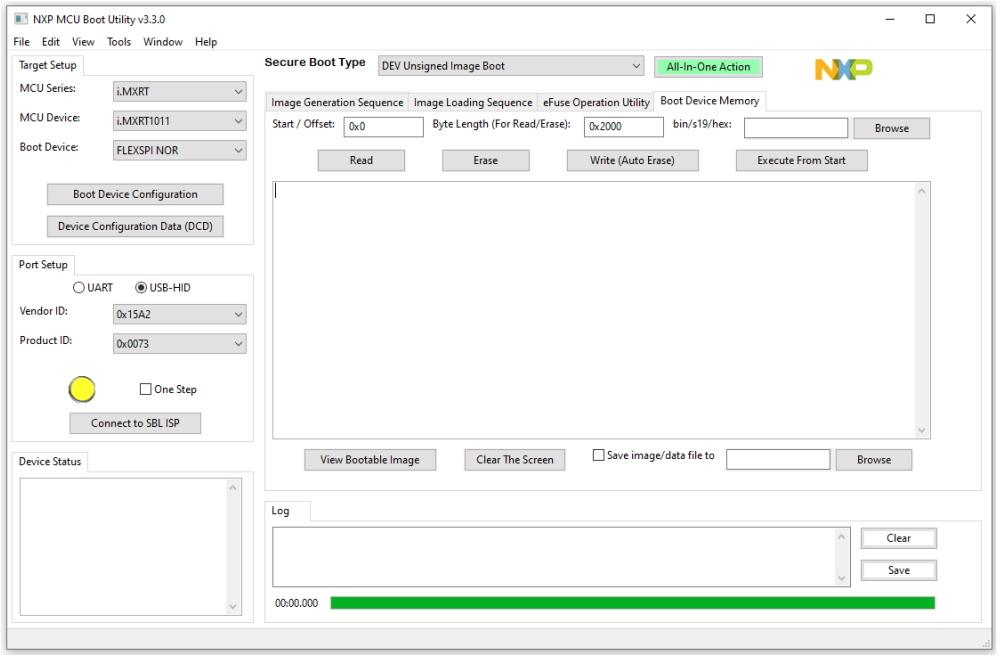


Figure 21. USB IDs are detected

3. Can do the read/erase/write ISP operation now. The image format can be bin/hex/s19

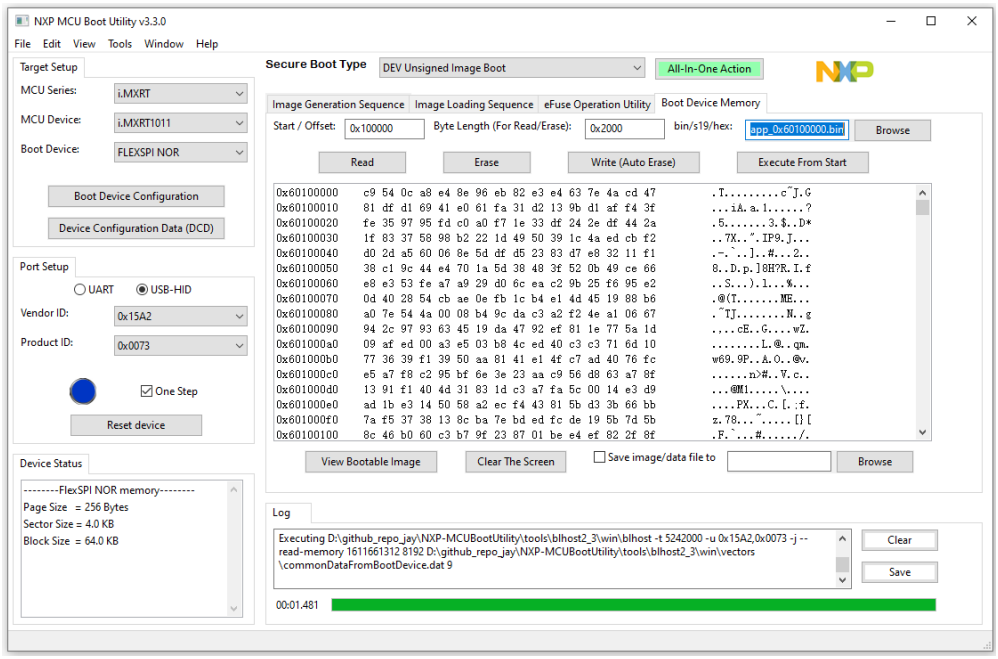


Figure 22. Download app

Chapter 5
Security

This section describes the implemented security feature. Secure Bootloader (SBL) is based on the MCUboot project. SBL keeps the MCUboot legacy RSA and ECDSA signatures. It also provides secure boot based on ROM bootloader and encrypted boot (XIP) based on hardware engine. So images can be signed, encrypted, or signed + encrypted.

MCUboot legacy signing method RSA and ECDSA sign the image by computing hash over the image and then signing that hash. Refer to the MCUboot design document for the details. SBL uses RSA-2048 and ECDSA-P256 by default.

SBL can support ROM secure boot and encrypted boot (XIP) on MIMXRT 4-digit platforms (MIMXRTxxxx), MIMXRT 3-digit platforms (MIMXRTxxx), and LPC55S69.

5.1 BootROM secure boot

The Secure Boot provides the guarantee that unauthorized code cannot be executed on a given product. It involves the device's ROM always executing when coming out of reset. The ROM examines the first user executable image resident in the flash memory to determine the authenticity of that code. If the code is authentic, the control is transferred to it. It establishes a chain of trusted code from the ROM to the user boot code. In this case, BootROM verifies SBL and SBL verifies the application image.

5.1.1 High Assurance Boot (HAB)

NXP MIMXRT 4-digit platforms provide the High Assurance Boot (HAB). It is the high-assurance boot feature in the system boot ROM, that detects and prevents the execution of unauthorized software (malware) during the boot sequence.

HAB uses asymmetric cryptography to sign the image. The bootable image can be signed by the CST tool. The tool generates the CSF data in the binary file format that consists of command sequences and signatures based on a given input command sequence file (CSF file).

The OEM uses a utility provided by NXP to generate a private key and corresponding public key pairs. Then the private key is used to encrypt the digest of the image which OEM wants to release. This encryption generates a unique identifier for the image which is called a signature. The certification with the public key is also attached to the image. Before applying the application, the public key is used to decrypt the signature. The OEMs burn the digest (hash) of the public key to the eFuses of MIMXRT chips. Once burned, it cannot be modified. BootRom can verify the public key by this value.

Below is the bootable image format for HAB.

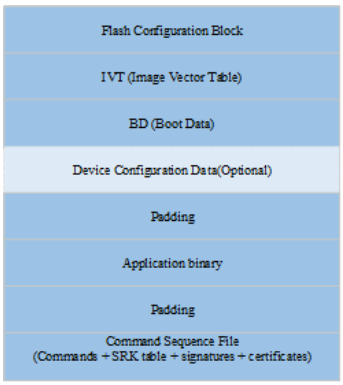


Figure 23. HAB signed image format without MCUboot header

It does not include Flash Configuration Block for a bootable application. As BootROM has configured flash by reading this field of SBL. All MIMXRT platforms support the RSA public key (1024, 2048, 3072 or 4096). MIMXRT1170 also supports ECDSA signature verification using the ECC public key (P256, P384, P521).

5.1.2 LPC55S69 secure boot

LPC55S69 devices support booting of RSA signed images using RSASSA-PKCS1-v1_5 signature verification. The boot code is signed with RSA private keys. The corresponding RSA public keys used for signature verification are contained in the signed image.

LPC55S69 devices support 2048-bit or 4096-bit RSA keys and X.509 V3 certificates.

Image validation is a two-step process.

1. Validate and extracts the Image public Key from the x509 certificate embedded in the image.

2. Uses Image_key (Public) to validate image signature.

The BootROM API skboot_authenticate is used to verify the authenticity of an image. Before running the application with this IAP API, the PFR region (CFPA and CMPA) should be configured.

PFR resides at the end of the flash region and can be programmed through ROM in ISP mode.

LPC55S69 stores configuration for the boot ROM in Protected Flash Region (PFR).

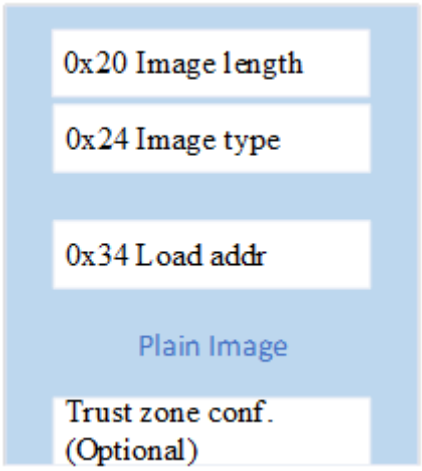


Figure 24. Signed image format without MCUboot header

5.1.3 Encrypted XIP Boot

MIMXRT 4-digit series BootROM supports XIP on the Serial NOR flash device directly with On-the-fly decryption feature (using AES) powered by BEE/OTFAD controller.

The PRINCE is used for real-time encrypt/decrypt operation on LPC55S69 on-chip flash contents.

5.1.3.1 Encrypted XIP boot based on BEE

EVKMIMXRT1060/1064/1050/1020 supports XIP with on-the-fly FlexSPI (QSPI) Flash decryption via Bus Encryption Engine (BEE). The BootROM supports two separate encrypted regions using two separate AES Keys. One encrypted region can be used for SBL, another can be used for application. The image can be encrypted by AES-CTR-128 or AES-ECB-128.

Before doing Encrypted XIP, the BootROM must set the BEE controller correctly, the configurable parameters are organized as Protection Region Descriptor Block (PRDB), the entire PRDB is encrypted using AES-CBC-128 mode with the AES KEY and IV in a Key Info Block (KIB). The KIB is encrypted as Encrypted KIB (EKIB) using the AES key provisioned in eFUSE (SW_GP2) or derived from OTPMK (One-Time Programmable Master Key). The BootROM decrypts KIB using AES ECB-128 mode, up to 2 EKIBs are supported, EKIB0 is located at offset 0x400, and KIB1 is located at offset 0x800.

The image key is AES KEY in the key info. In this solution, SW_GP2 is used as KEK to encrypt the key info.

The tool `image_enc.exe` can be used to encrypt the image on the host. It is a command-line host program that a customer can use to verify the encrypted procedure.

5.1.3.2 Encrypted XIP boot based on OTFAD

EVKMIMXRT1170/1010 and EVKMIMXRTxxx support XIP with on-the-fly FlexSPI(QSPI)Flash decryption via On-the-Fly AES Decryption Module (OTFAD). The OTFAD supports up to 4 separate encrypted regions using separate AES keys.

Before booting Encrypted XIP, the BootROM must set the OTFAD module correctly, the configurable parameters are organized as KeyBlob. A KeyBlob contains encryption keys for OTFAD, and is always encrypted with a KEK. The KEK can be scrambled for each encryption region. The entire KeyBlob is encrypted using AES-CTR-128 mode. KeyBlob is at offset 0x0 in flash.

The KEK is stored in the OTP/EFUSE block. For EVKMIMXRT1170, the KEK can be restored by the PUF, using the PUF key store as part of the Encrypted XIP image.

In this solution, two KeyBlobs are used. One KeyBlob is used for SBL and another is used for application.

5.1.3.3 Encrypted XIP boot based on PRINCE

LPC55S69 supports on-the-fly encryption/decryption to/from internal flash through PRINCE. Data stored in on-chip internal Flash could be encrypted in real time.

LPC55S69 supports 3 regions that allow multiple code images from independent encryption base to co-exist. Each PRINCE region has a secret-key supplied from on-chip SRAM PUF via secret-bus interface (not SW accessible). PRINCE encryption algorithm does not add latency.

PRINCE keys are 128-bit symmetric key and are sourced from on-chip SRAM PUF via an internal hardware interface, without exposing the key on the system bus.

The PUF controller provides secure key storage without storing the key. It is done by using the digital fingerprint of a device derived from SRAM. Instead of storing the key, a key code is generated, which in combination with the digital fingerprint is used to reconstruct PRINCE keys that are routed to the AES engine or for use by software. These key codes are stored in PFR region of flash.

During the startup, the ROM checks if valid key store data structure is present in PFR. If so, the whole key store data structure is loaded into RAM and ROM issues PUF start procedure. It initializes PUF and reconstructs original keys so that each key can be used if needed.

5.1.4 Image format

Below is the final file format. Application image can be signed, encrypted, or signed + encrypted. If the image is encrypted, the key context should be inserted into the image header part for the MIMXRT 4-digit platform. It is at offset 0x100 in the MCUboot header.

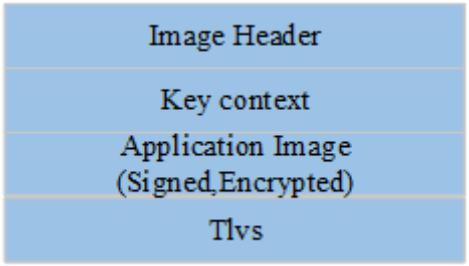


Figure 25. Final image format

5.1.5 Tools

To use the security feature, prepare the following tools:

- CST Tool (Optional) - Code Signing Tool, an application running on a build host to allow manufacturers to sign or encrypt the software for their products incorporating NXP processors.
- `elftosb.exe v4.0.0` – Combined with CST is used to generate an unsigned/signed bootable image.
- `image_enc.exe` - It is a command-line host program used to encrypt image.

- MCUXpresso Secure Provisioning Tool (SPT) – It is a GUI tool made to simplify the generation and provisioning of bootable executables on NXP MCU platforms.

MCUX Secure Provisioning Tool includes `cst.exe`, `elftosb.exe, and image_enc.exe`. Download them from the website.

In the folder `sbl\target\evkmimxrtxxxx\secure`, there are one-stop scripts to generate signed and encrypted image with these tools. For more details, please see section 7.4.

Chapter 6
Firmware

This section gives the details of the operation of Secure Firmware (SFW).

6.1 SFW

Secure Firmware (SFW) is an instance of application, it was created based on FreeRTOS, and developed to implement the complete FOTA process together with SBL. SFW supports obtaining the OTA firmware image by U-Disk, SD card in local or AWS cloud, Aliyun cloud in remote. Then SBL checks, authenticates the OTA firmware image and boots it up in normal.

SFW follows the same framework of SBL, they have the same building environment, configuring process, and compiling commands. Once familiar with SBL, it is not difficult to use SFW.

For both swap and remap mode, SFW provides a function `enable_image()` to let the users call after writing new image to the flash. Because of the different flag mechanism of these two modes, SFW uses macros to distinguish them.

6.2 Operation to set the OTA flag

This section gives the details of setting the OTA flag.

6.2.1 Operation for swap mode OTA

For the swap mode OTA, the **image_trailer** of the two slots (**the trailer is in the last 32 bytes of two slots**) is used to judge the swap type and control the rollback. Figure 26 shows the state of the flag. Unset is 0xFF, Set is 0x01.

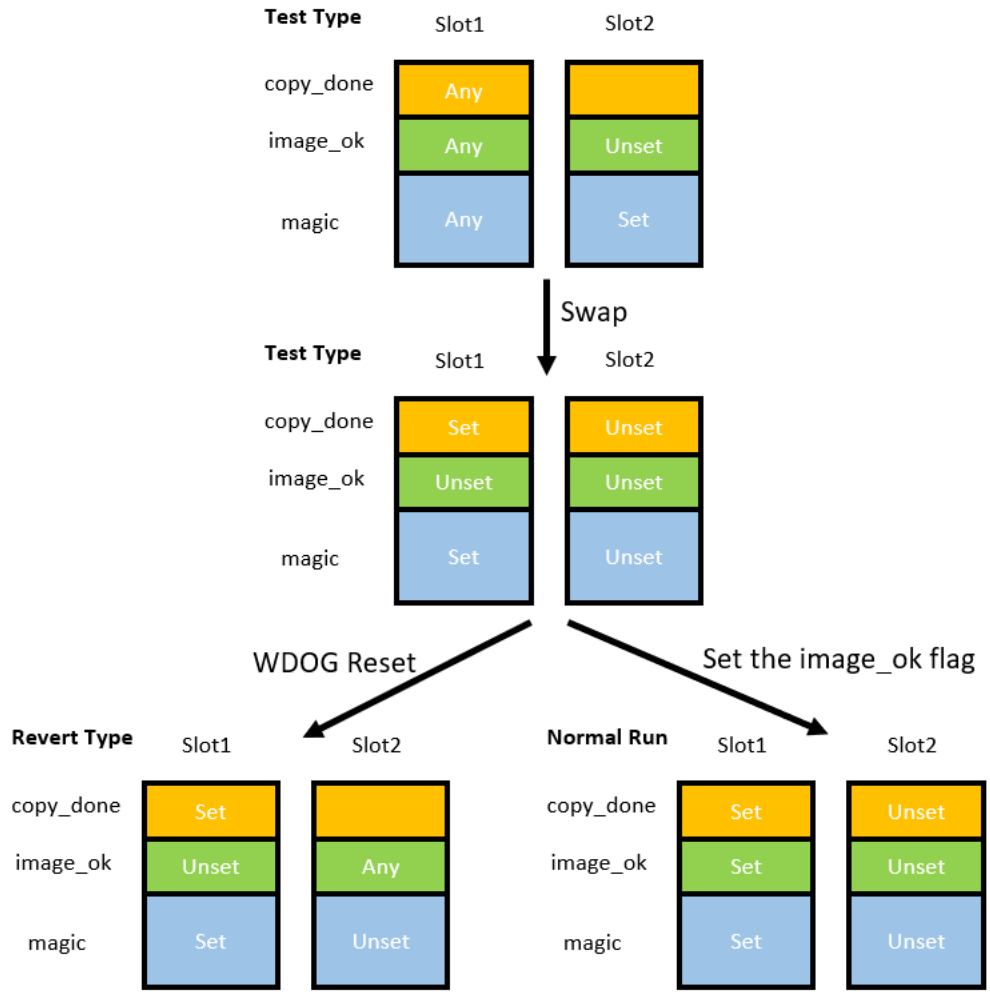


Figure 26. Swap flag state

To initialize the OTA process, after writing the new firmware to slot2, the old firmware that receives the new firmware must write the magic (fixed value, 16 bytes) to the end of slot2 to inform the bootloader that the new firmware has been written to the slot2. After writing down the magic value, reset the board.

The bootloader now detects the OTA type, which is test type. Then the bootloader performs the exchange, during the exchange process, the trailer in slot1 becomes the trailer in slot2, and the position of the trailer in slot2 is cleared. The bootloader goes to slot1 to execute the new firmware. If the new firmware operates normally, it writes the `image_ok` flag to the slot1 trailer to disable the revert. Otherwise, an error occurs in the new firmware, the `image_ok` flag is not set, then the watchdog resets the board, the bootloader judges the OTA type, now the type is Revert, exchange the two slots, and clear the trailer position of the slot2, now the trailers of the two slots are all unset.

Note: For a board using swap mode OTA, the firmware must contain two writing flag operations. First, the magic part of the flag is written, this operation must be performed after the new firmware is written. The magic address is 0x2FFFF0. The second operation is writing `image_ok` flag. After the firmware itself runs the whole task period and during the period everything is OK, the firmware must set the flag. The address of `image_ok` is 0x2FFFE8. The magic value is as on Figure 27.

```
const uint32_t boot_remap_magic[] = {
    0xf395c277,
    0x7fefd260,
    0x0f505235,
    0x8079b62c,
};
```

Figure 27. Magic value

6.2.2 Operation for the remap mode OTA

For the remap mode OTA, the remap update flag is used to judge the remap type and control the rollback. The flag is in the fixed offset address of the flash, the offset is 0xFFFE0, flag structure occupies 32 bytes of space. Figure 28 shows the state of the flag. Unset is 0xFF, set is 0x01, revert is 0x04.

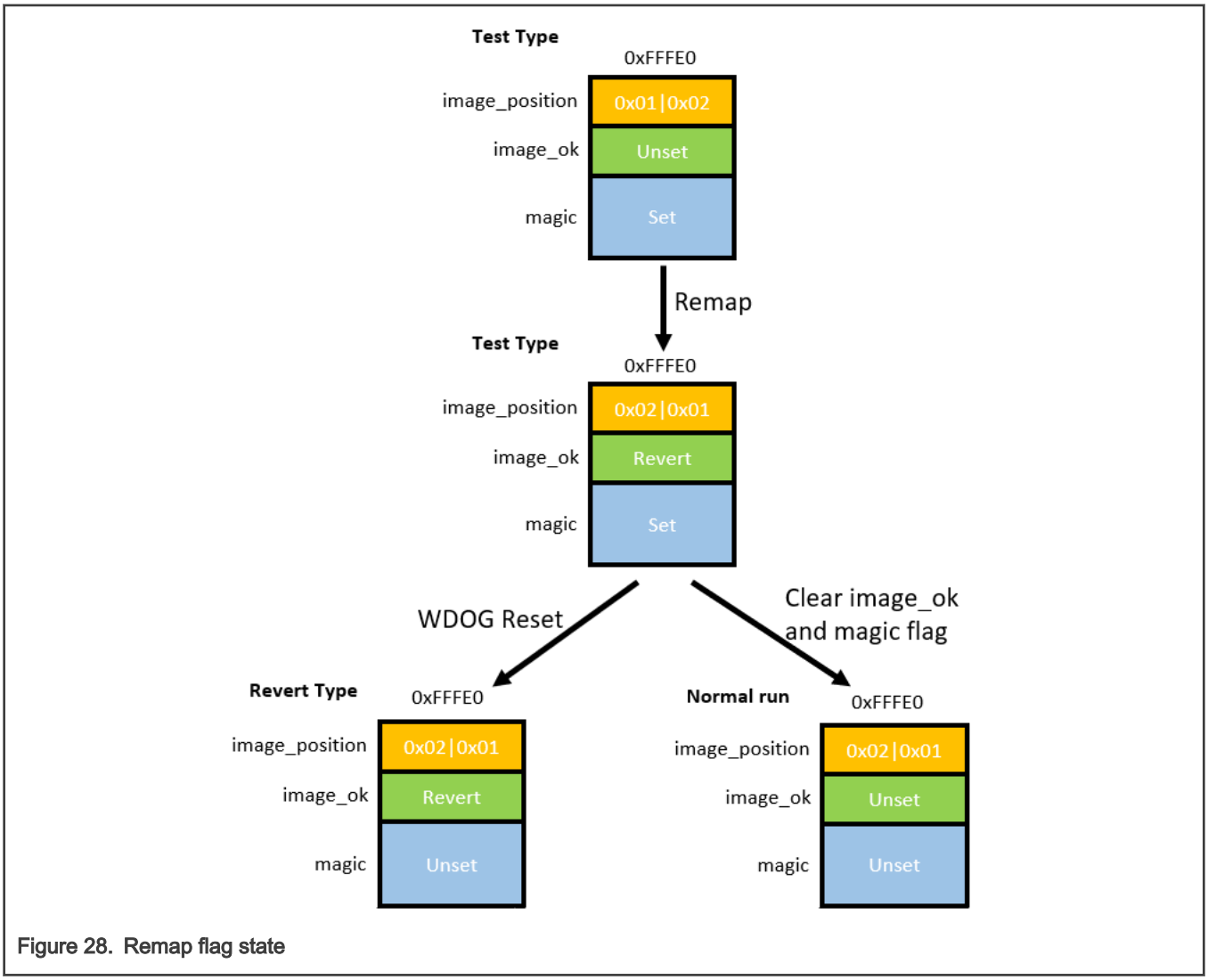


Figure 28. Remap flag state

To initialize the OTA process, after writing the new firmware to slot2 or slot1, the old firmware that receives the new firmware must write the magic (fixed value, 16 bytes, same value as the swap mode) to the position of the magic flag to inform the bootloader that the new firmware has been written to slot1 or slot2. After writing the magic value is done, reset the board.

The bootloader reads the remap update flag to get the current firmware position and judge the OTA type, now the type is test type. If the current position is 0x01 (slot1), set the `image_ok` part of the flag to 0x04 (means revert) and enable the remap function

and run on slot2 physically. If the new firmware operates normally, it clears the `image_ok` and the magic part of the flag to disable rollback. Otherwise, an error occurs in the new firmware, the new firmware does not clear the flag, then the watchdog resets the board, the bootloader judges the OTA type, now the type is Revert, flip the state of the remap function, and clear the `image_ok` and the magic part of the flag.

Note: For a board using swap mode OTA, the firmware must contain two writing flag operations. First, the magic part of the flag is written, this operation must be performed after the new firmware is written. The magic address is 0xFFFF0. The second operation is writing `image_ok` flag. After the firmware itself runs the whole task period and during the period everything is OK, the firmware must clear these two parts of the flag.

Chapter 7
FOTA

SBL is a secondary bootloader designed for the Firmware Over-The-Air (OTA) application. It stores and manages the OTA image upgrade by reading, authenticating, and writing the OTA image to internal/external memory devices.

It provides the following OTA features:

- Image swap and revert
- Image remap and revert
- FlashIAP
- Security
- ISP

7.1 Design

This section is dedicated to the design of the Firmware Over-The-Air (OTA) application.

7.1.1 Single image mode of OTA

The flash layout for the single image mode of OTA is as below:

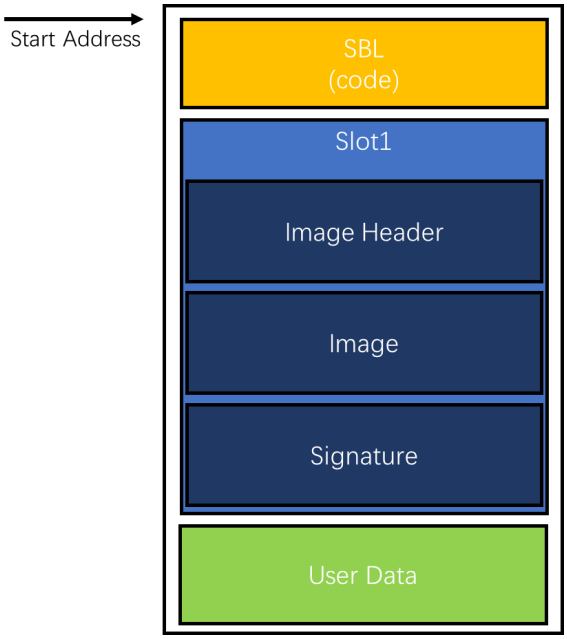


Figure 29. Flash layout for single image mode

The workflow for the single image mode of OTA is as below:

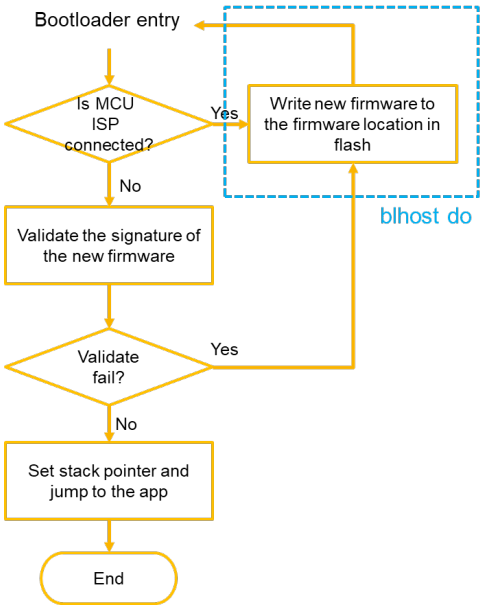


Figure 30. Bootloader workflow for single image mode

The period for the single image mode of OTA is as below:

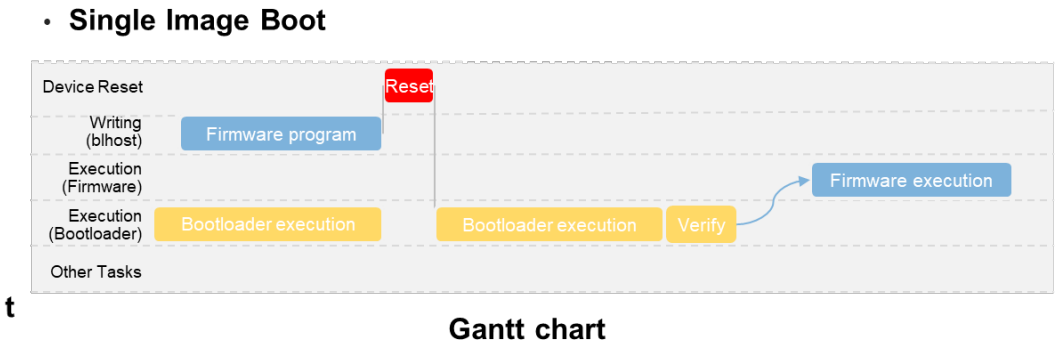


Figure 31. Update period of bootloader for single image mode

7.1.2 Swap mode of OTA

The OTA image itself consists of the image header, image data and image trailer. The image header information is shown in the below table.

Table 5. Image header format

Offset	Width (bytes)	Field	Description
0x00	4	magic	Image header tag Fixed value
0x04	4	load_addr	Point to the load address of the application

Table continues on the next page...

Table 5. Image header format (continued)

Offset	Width (bytes)	Field	Description
0x08	2	header_size	Size of the image header
0x0a	2	reserved	Reserved for future use
0x0c	4	image_size	The size of the image (not including the Image Header Size)
0x10	4	flags	Not used now
0x14	8	image_version	Image version
0x1c	4	reserved	Reserved for future use

The image data are the actual image content, it supports raw binary image format.

The image trailer information is shown in the below table.

Table 6. Image trailer format

Offset	Width (bytes)	Field	Description
0x00	1	copy_done	Flag that the swap done
0x01	7	Pad	Reserved
0x08	1	image_ok	Flag that control the OTA state
0x09	7	pad	Reserved
0x10	16	magic	Image trailer tag Fixed value

The flash layout for swap mode of OTA is as below:

Figure 32. Flash layout for swap mode

- The SBL resides at the start of the Flash memory.
- The Swap area now is equal to slot1 and 2, this area can be reduced to the size of a sector.

The workflow for swap mode of OTA is as below:

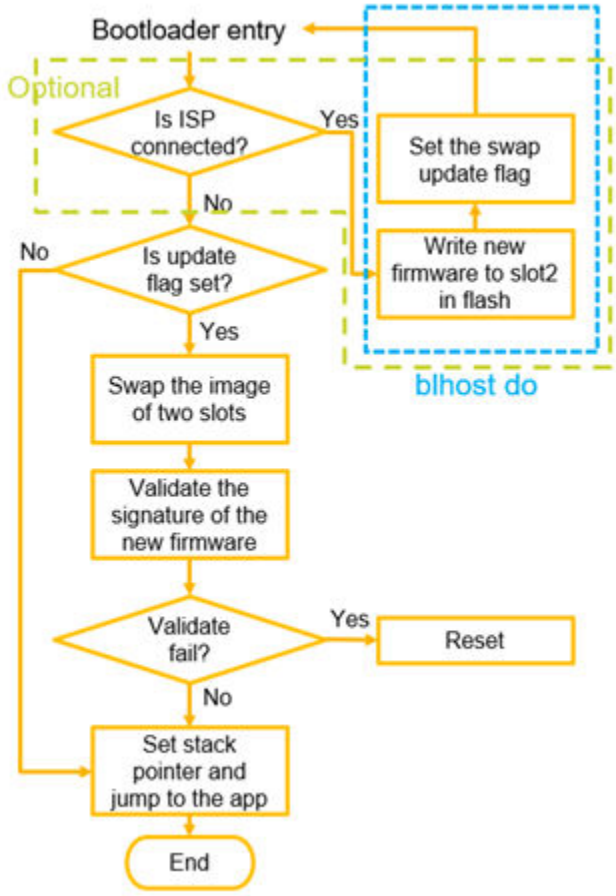


Figure 33. Bootloader workflow for swap mode

The period for swap mode of OTA is as below:

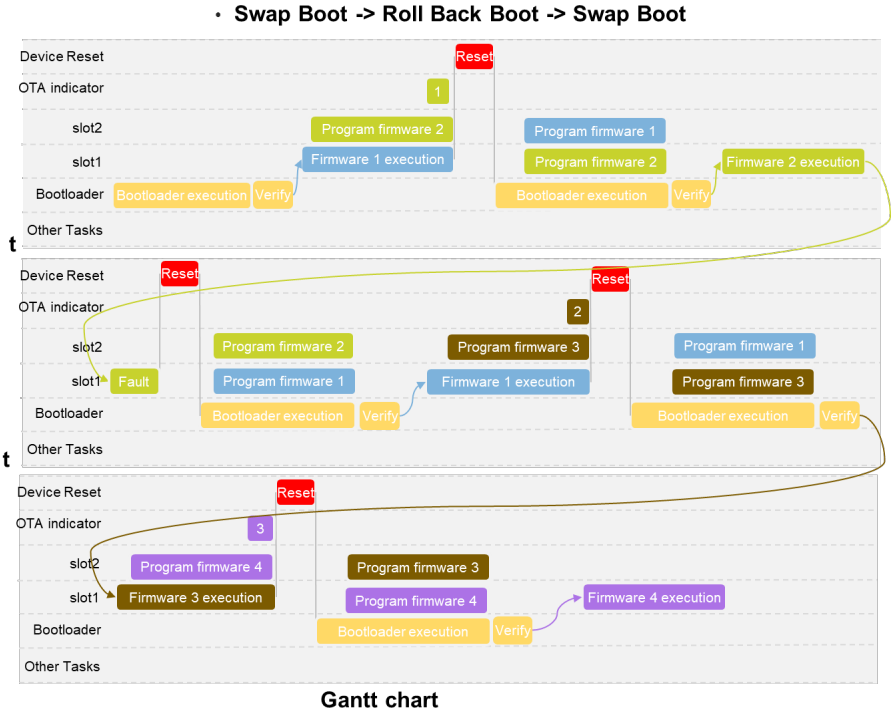


Figure 34. Update period of bootloader for swap mode

7.1.3 Remap mode of OTA

The OTA image of remap mode contains the image header, image data. The image header information is the same as swap mode, refer to Table 5.

To control the remap state, before the first slot, the remap update flag structure is set to control the update process. The format is as shown in the table below.

Table 7. Remap flag format

Offset	Width (bytes)	Field	Description
0x00	1	Image_position	The current firmware position
0x01	7	Pad	Reserved
0x08	1	image_ok	Flag that control the OTA state
0x09	7	pad	Reserved
0x10	16	magic	Image trailer magic Fixed value

The flash layout for remap mode of OTA is as below:

Figure 35. Flash layout for remap mode

The workflow for remap mode of OTA is as below:

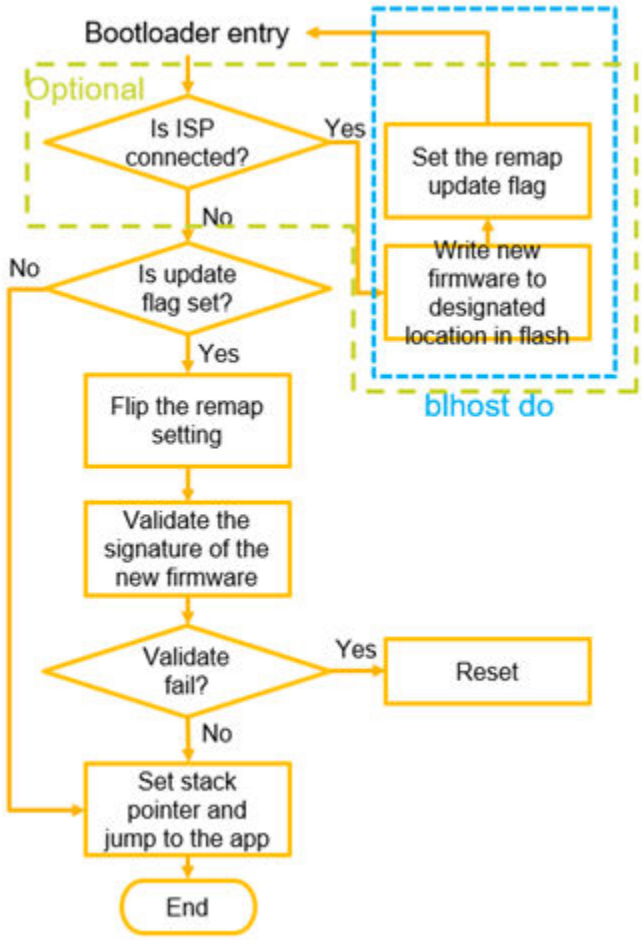


Figure 36. Bootloader workflow for remap mode

The period for remap mode of OTA is as below:

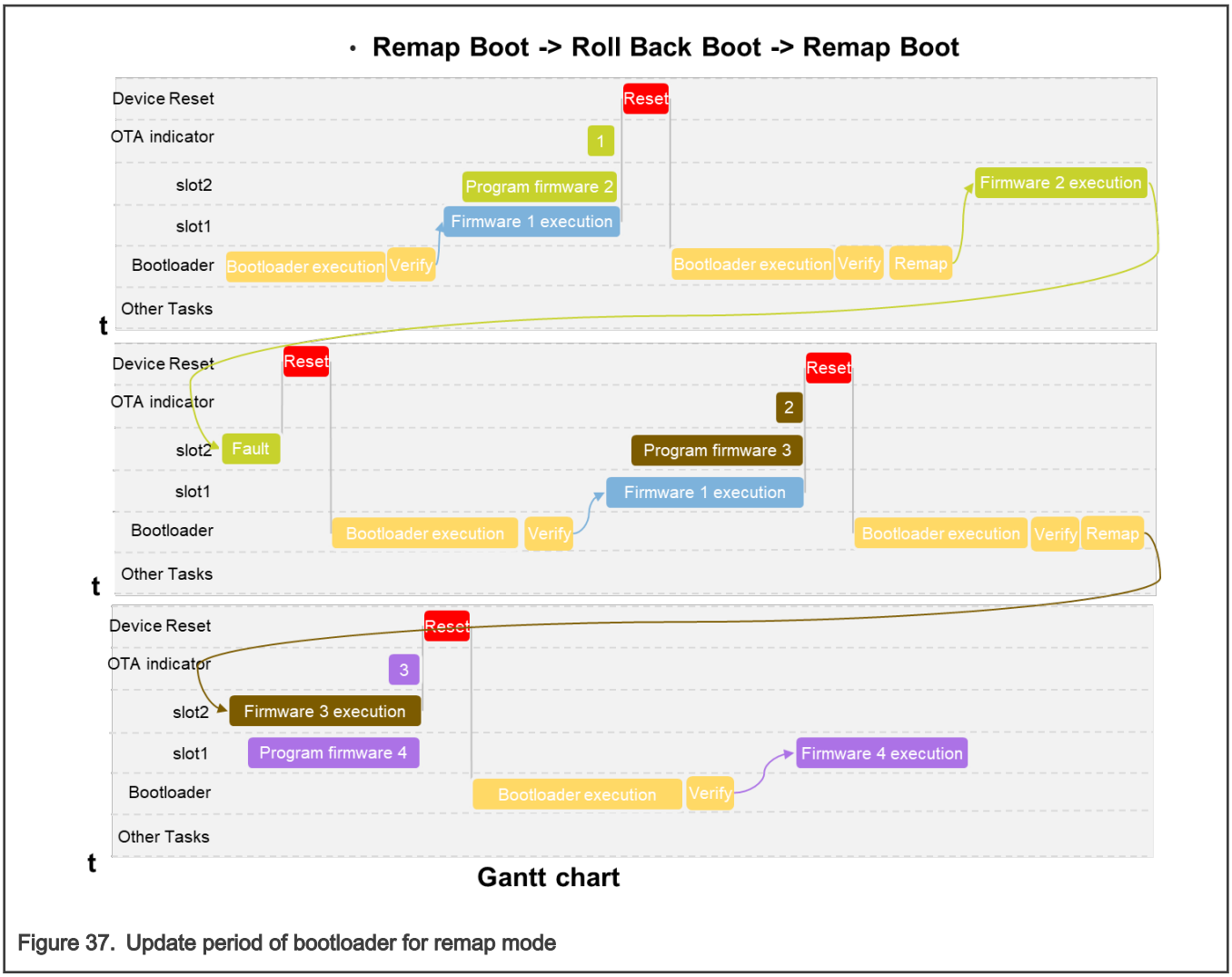


Figure 37. Update period of bootloader for remap mode

7.2 Local FOTA

For all three OTA modes (single, swap, remap), the default configuration of SBL must verify the signature of the image, so after generating the image file by IAR/MDK/GCC, add a header and signature to the image file. The steps below introduce how to make an available application image.

All SBL target can support U-Disk to update the image, and all the target can support SD Card update except EVKMIMXRT1010.

1. Prepare the image

   For single image mode:

   Select the ‘hello world’ demo from SDK as an example. First, to make space for adding a header later, change the linker file. The default application offset address is 0x100000, modify the linker to adapt to this address, the IAR linker of EVKMIMXRT1060 in the picture below can be used for reference.

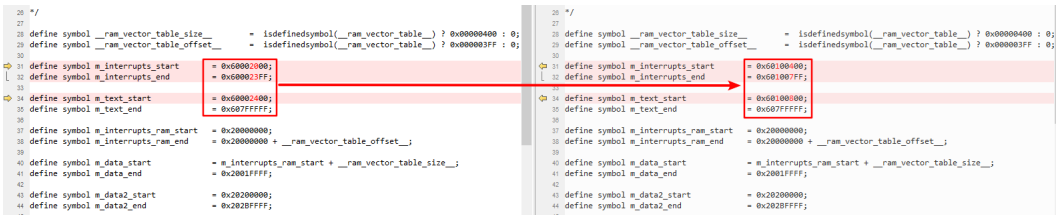


Figure 38. Linker file modification example

Remove the XIP header information by set `XIP_BOOT_HEADER_ENABLE = 0` in iar project option, and then compile the project and generate a binary file named `hello_world.bin`.

The SFW project already included above changes, so build and use the SFW image directly for single image mode.

For swap mode and remap mode:

Double-click the `env.bat` in the directory of the corresponding target of SFW. Using command then in the configuration menu, uncheck the `Enable sfw standalone xip` option, and check the `OTA from sdcard` and `OTA from u-disk` options.

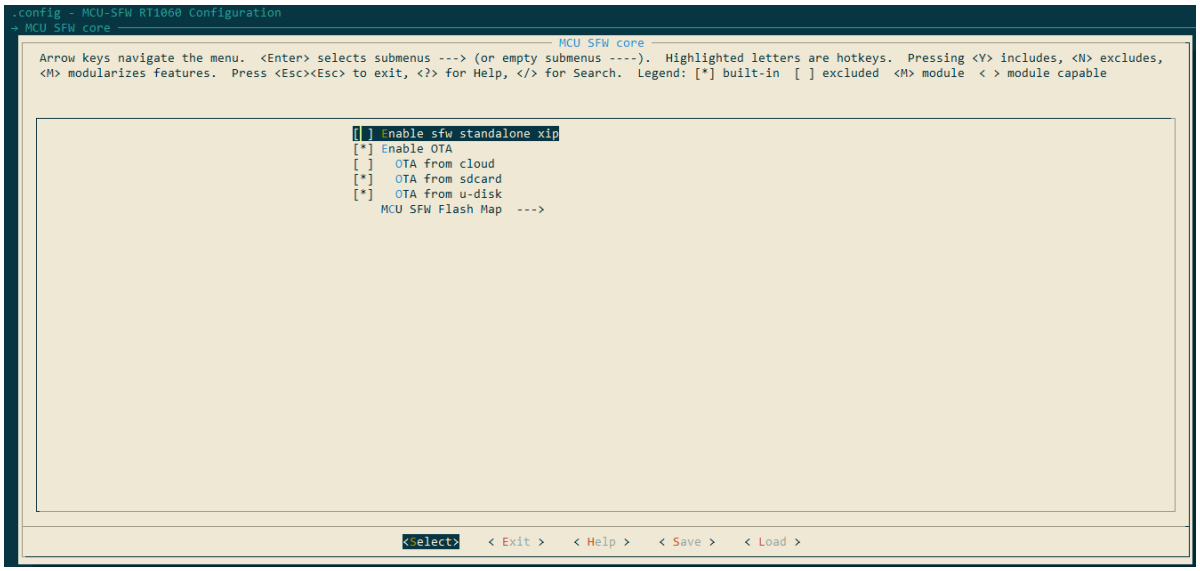


Figure 39. Configure the SFW

Following commands described in chapter 2 to generate project, build the project and the images are generated .

2. Generate signature key pair and prepare the bootloader

   Double click the `env.bat` in the directory of the corresponding target of SBL. Run to the directory in the `cd ..\..\component\secure\mcuboot\scripts` pop-up command shell, then use the script command to generate a signature key:

   ```
   python imgtool.py keygen -k xxxx_priv.pem -t rsa-2048-sign
   ```

   The `xxxx_priv.pem` is used to sign the application image.

   Generate the public key `python imgtool.py getpub -k xxxx_priv.pem -o xxxx_pub.pem -t sign`.

   The `xxxx_pub.c` file generated by the above command contains the data structure of the public key, which is an array. It should be compiled with the bootloader to verify the signature. Use it to replace the content in `sign-rsa2048-pub.c` in `sbl\component\secure\mcuboot\` directory.

3. Add signature and header to the image

Using the `imgtool` command to generate the useful application image, type the command in the shell to finish the operation. For single image mode, use `hello_world.bin` generated in step1. For swap and remap modes, to run the test, use the image that SFW project generated.

And use the command below to generate the first signed image:

```
python imgtool.py sign --key xxxx_priv.pem --align 4 --version "1.1" --header-size 0x400 --
pad-header --slot-size 0x100000 --max-sectors 32 hello_world1.bin app1.bin
```

Use another image which differs from the first image and repeat the command:

```
python imgtool.py sign --key xxxx_priv.pem --align 4 --version "1.2" --header-size 0x400 --
pad-header --slot-size 0x100000 --max-sectors 32 hello_world2.bin app2.bin
```

Now two signed images are generated. More information about the above sign commands is described below.

- `xxxx_priv.pem`: The private key certificate generated in step 2
- `--version`: The version format can be `major.minor.rev`
- `hello_world.bin`, `app.bin`: The file path can also be added for `hello_world1.bin`, `app1.bin`, `hello_world2.bin`, and `app2.bin` in the command.

NOTE

For EVKMIMXRT685, to enable the remap function and the single image function (which have reset operations), use J-link to write the shadow register in advance, the specific steps are as follows:

1. Remove the JP2 jumper cap.
2. Make sure to connect EVKMIMXRT685 with J-link, then use J-link to write the shadow register in advance, the instruction is as follows: `w4 0x40130184 0x314000`.
3. After shadow register is written, make sure that the MCU cannot be powered down during the entire operation, otherwise the previous operation is invalid. This limitation only exists on EVKMIMXRT685 FlexSPI port b. To ensure that the entire operation is not powered down, after the image is downloaded to the flash, reset the MCU by SW3 instead of power-down reset. In addition, if the DAPLink is used to download images, make JP2(PIN1-2) short, then remove the J-Link probe.

NOTE

For the EVKMIMXRT595 / EVKMIMXRT685 U-disk update function, power on the EXT PWR port. Otherwise, the power supply current is not enough, which causes the U-disk update to fail.

NOTE

The default SDIO interface of the latest EVKMIMXRT595 board is eMMC, not SD card. When using the U disk update function with the latest board, rewrite the board first.

7.2.1 Single image OTA

1. Configure the SBL

   The single image function only involves verification of the signature of the image, and there is no erase and write operations on the flash during the bootloader stage of SBL. OTA in this mode must be carried out with MCU ISP.

   Check the `Enable single image` function option in the menuconfig interface of Scons to enable single image mode. At the same time, check the`Enable mcu isp support` option in the menuconfig interface of Scons to enable the MCU ISP function.

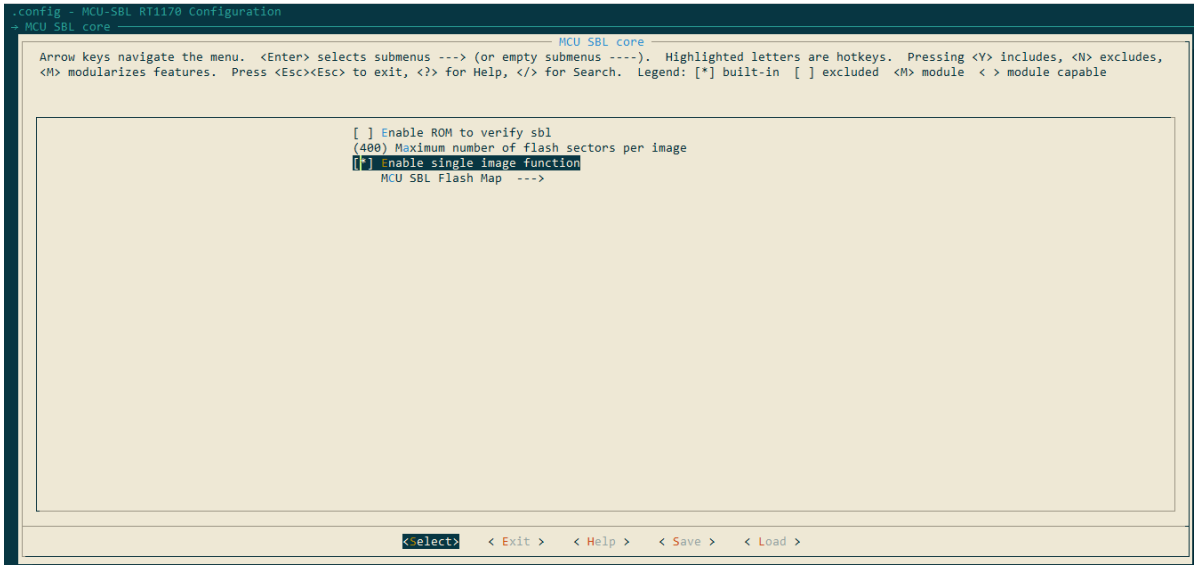


Figure 40. Enable single image

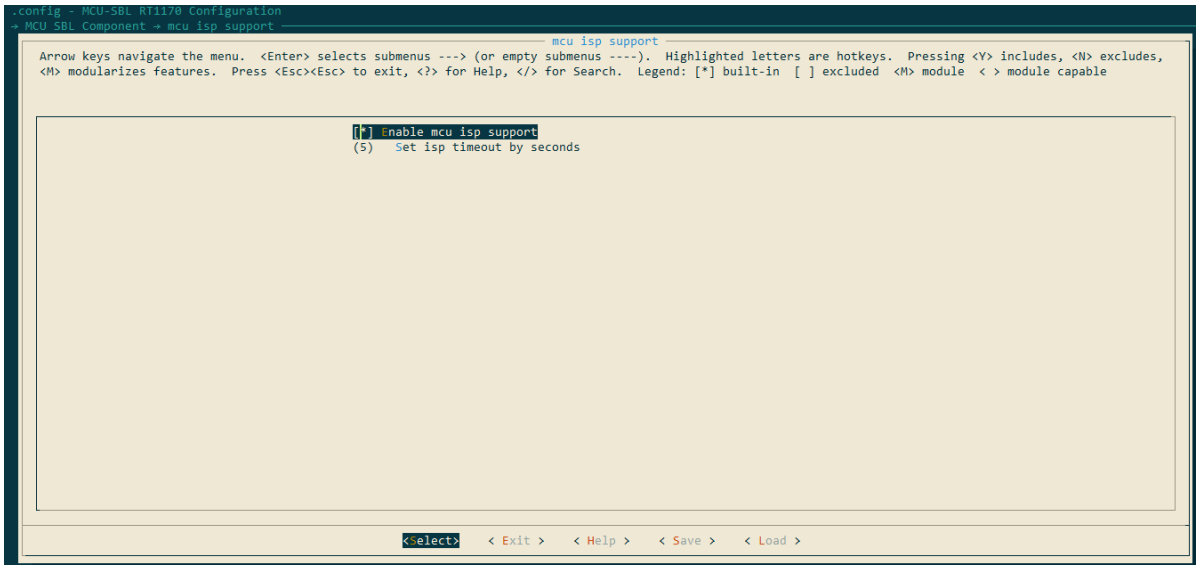


Figure 41. Enable mcu ISP

Generate the project, compile, and download the SBL to the target board. Then, connect the UART or USB port to the PC and reset the board.

2. Run the test

   UART connect command type: `blhost -p COMx,115200 -- command`

   USB connect command type: `blhost -u <vid>,<pid> -- command`

> **NOTE**
>
> When using UART type to run the test, it must directly connect the UART port to PC by using the TTL2USB module, refer to the schematic of EVK board.

After the reset, the board waits for 5 seconds. During the 5 seconds, if the PC sends the `connect` command (blhost send the `get-property` command and get success in terminal), the board runs into MCU ISP mode. If it does not, the board runs into boot mode.

On PC side, put the image which is download and the `blhost.exe` tool into a folder, and open the terminal under this folder, type the command to connect the blhost and the board within 5 s after the board is powered on. `blhost -u -- get-property 1 0`

After connecting successfully, type the below commands to do the flash related operations.

Read: `blhost -u -- read-memory [address] [size]`

Erase: `blhost -u -t [ms] -- flash-erase-region [address] [size]`

Write: `blhost -u -- write-memory 0x60100000 xxx.bin`

NOTE

If the size is very large, when using erase operation to erase the flash, add **-t [ms]** in the command to add the timeout. In case the timeout is too short, the mcu ISP may return erase failed.

The picture below shows the entire PC-side blhost update image operation process.

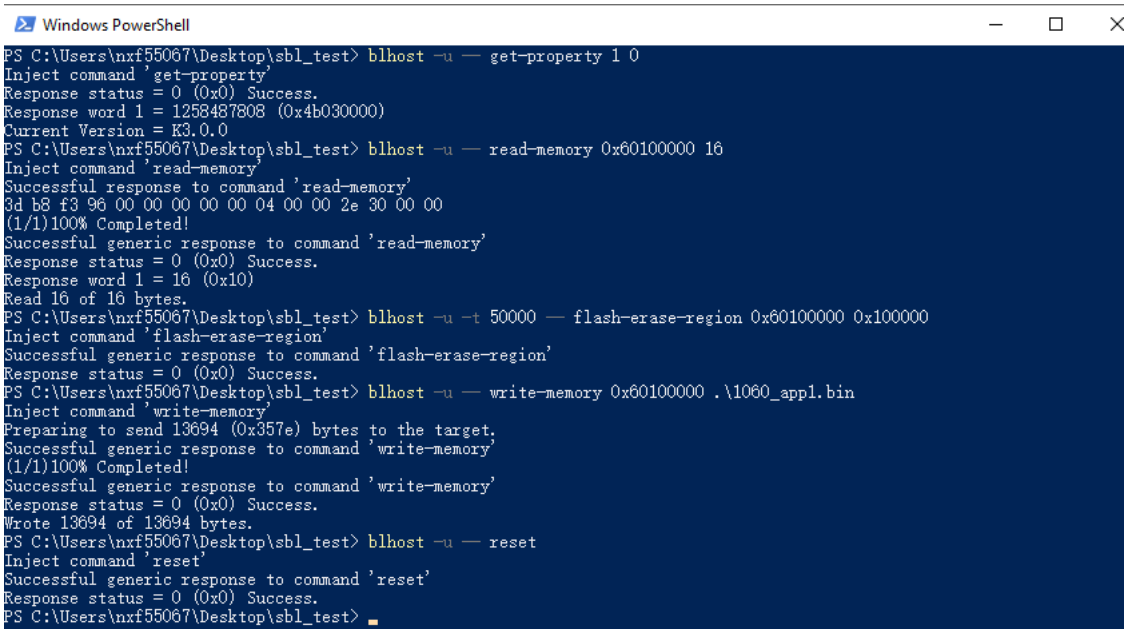


Figure 42. Blhost operation on PC

To reset the board, write the new image to the flash slot and then type the reset command. Enter the boot process after 5 s.

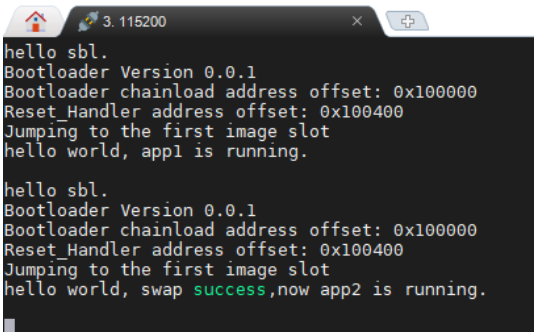


Figure 43. Single image updating log

7.2.2 SD card OTA

In this mode, the SD card is used to update the image as reference.

NOTE

For EVKMIMXRT1170, to enable the SD card, connect R136 to the REV C EVK board.

1. Prepare the SBL

   To disable single image mode and disable MCU ISP support, disable the `'Enable single image function` option and the `Enable mcu isp support` option in the menuconfig interface of Scons.

   Compile the SBL project and download it to the target board.

   NOTE

   For EVKMIMXRT595, EVKMIMXRT685, EVKMIMXRT1170, and EVKMIMXRT1010, when downloading the `sbl.bin` file, start from 0x400 offset of the flash.

2. Download the first image

   To test the SBL for the first time, download the first image to the board to run the test. Use the `MCUBootUtility` tool to download `app1.bin` to the first slot of the board, the default location of the slot1 is from `flash_offset+0x100000` to `flash_offset+0x200000`, the whole slot size is 1 MB.

   NOTE

   Set the MCUBootUtility to master mode, if MCU ISP function is not enabled.

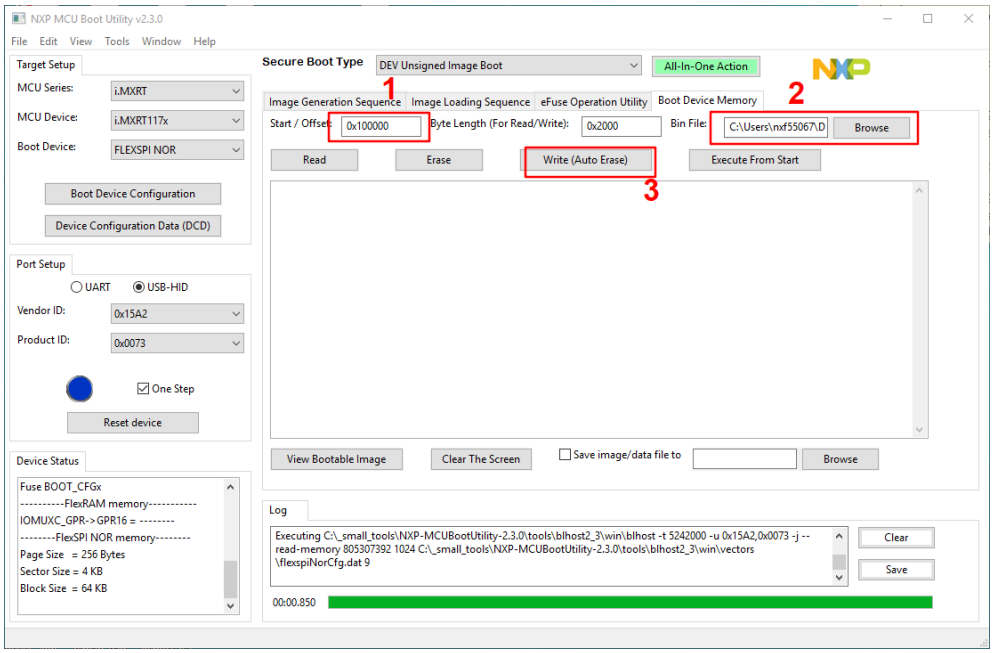


Figure 44. Download image1 to slot1

3. Run the test
   - Insert an SD card to the PC, and copy the `app2.bin` to the SD card.
   - Rename the filename to `newapp.bin`.
   - Remove the SD card and insert it to the target board.
   - The debug console then prints the updating log.
   - After the SD card downloading finished, when the log '`sys rst…`' printed, remove the SD card from the board, otherwise, it starts a new updating.

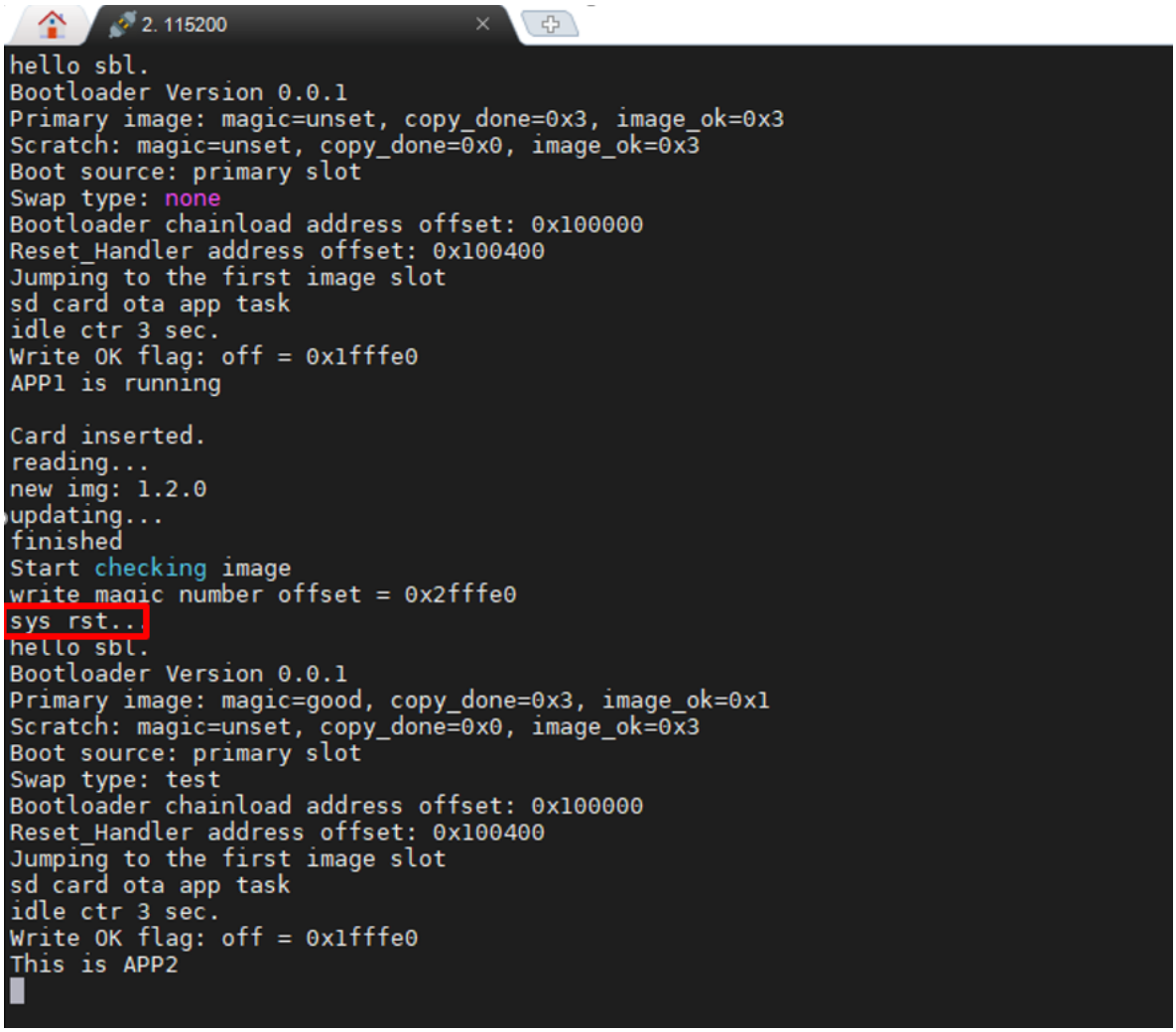


Figure 45. Swap updating log

After the app2 log is printed, push the **Reset** button on the board. To confirm that the updating is successful, the app2 log should be printed.

7.2.3 U-Disk OTA

In this mode, the U-Disk is used to update the image as reference.

1. Prepare the SBL

   To disable single image mode and disable MCU ISP support, disable the `'Enable single image function` option and the `Enable mcu isp support` option in the menuconfig interface of Scons.

   Compile the SBL project and download it to the target board.

2. Download the first image and the remap image flag

   To test the SBL for the first time, download the first image to the board to run the test. Use the `MCUBootUtility` tool to download `app1.bin` to the first slot of the board, the default location of the slot1 is from `flash_offset+0x100000` to `flash_offset+0x200000`, the whole slot size is 1 MB.

3. Run the test

   - Connect a U-Disk to the PC.

- Copy the `app2.bin` to the U-Disk,
- Rename the bin file to `newapp.bin`
- Disconnect the U-Disk from the PC and connect it to the target board (using an otg cable, if the board has two USB ports, connect it to USB1)
- The debug console prints the updating log
- after the image downloading finished, when the log `sys rst…` printed, remove the U-Disk from the board, otherwise, it starts a new updating.

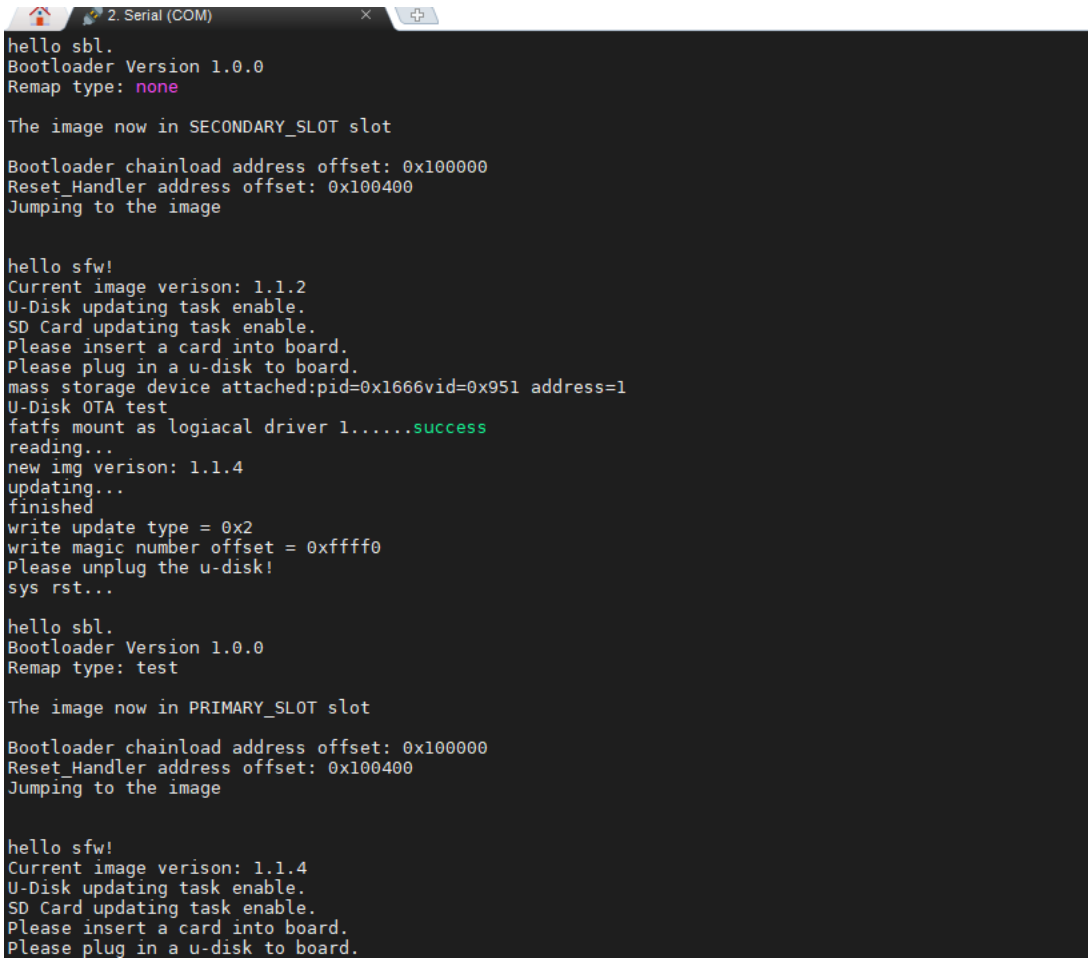


Figure 46. Remap updating log

After the app2 log is printed, push the **Reset** button on the board. To confirm that the updating is successful, the app2 log should be printed.

7.3 Remote FOTA

This section is dedicated to remote FOTA.

7.3.1 AWS OTA

This section walks through the steps how to perform the AWS OTA firmware update of the board using AWS IoT and the EVKMIMXRT1170 platform as an example. The aim is to demonstrate the testing process.

7.3.1.1 AWS OTA Prerequisites

- Create an AWS Account

  Create an AWS account: https://console.aws.amazon.com/console/home

- Create an Amazon S3 Bucket to store the update

1. Go to the https://console.aws.amazon.com/s3/
2. Choose **Create bucket**.

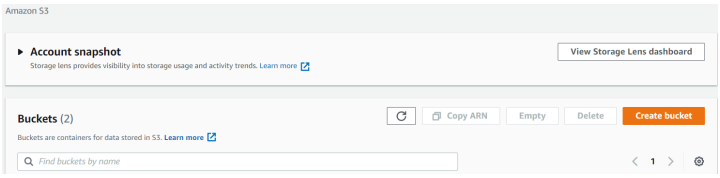


Figure 47. Create bucket

3. Type a bucket name.
4. Select **Enable** for Bucket Versioning.

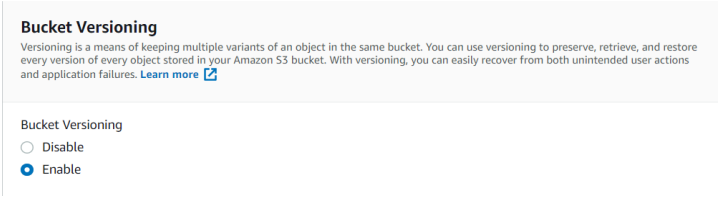


Figure 48. Bucket versioning

5. Other options keep default configurations and choose **Create bucket.**

- Create an OTA service role

1. Sign in to the https://console.aws.amazon.com/iam/
2. From the navigation pane, choose **Roles**.

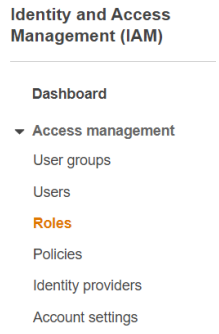


Figure 49. Roles

3. Choose **Create role**.
4. Under **Select type** of trusted entity, choose **AWS Service**.

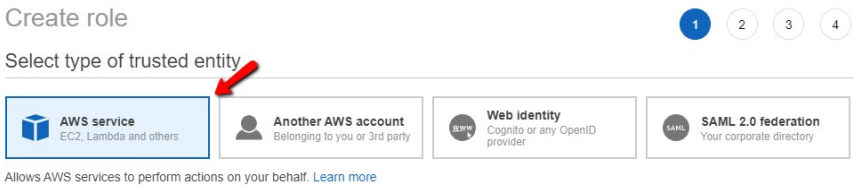


Figure 50. AWS service

5. Choose **IoT** from the list of AWS services.

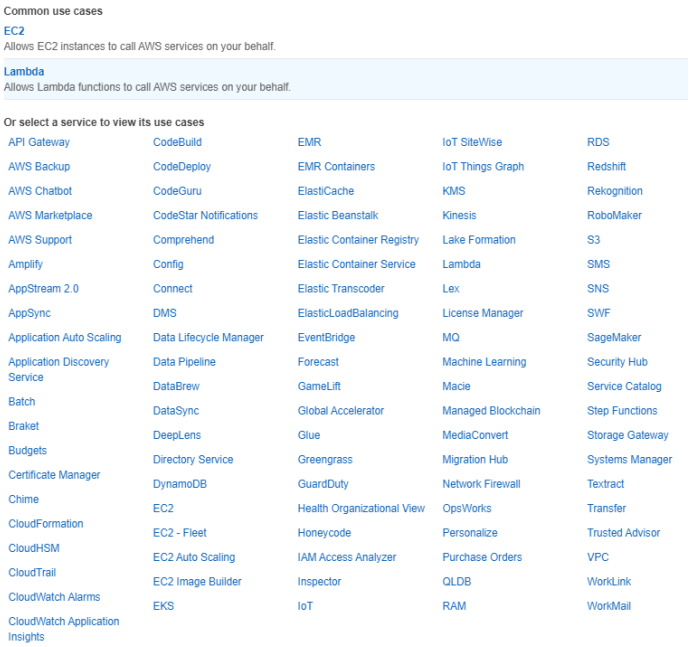


Figure 51. IoT service

6. Under Select the use case, choose **IoT**.

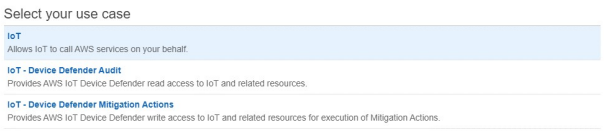


Figure 52. IoT use case

7. Choose **Next: Permissions**.
8. Choose **Next: Tags**.
9. Choose **Next: Review**.
10. Enter a role name and description and then choose to **Create role**.

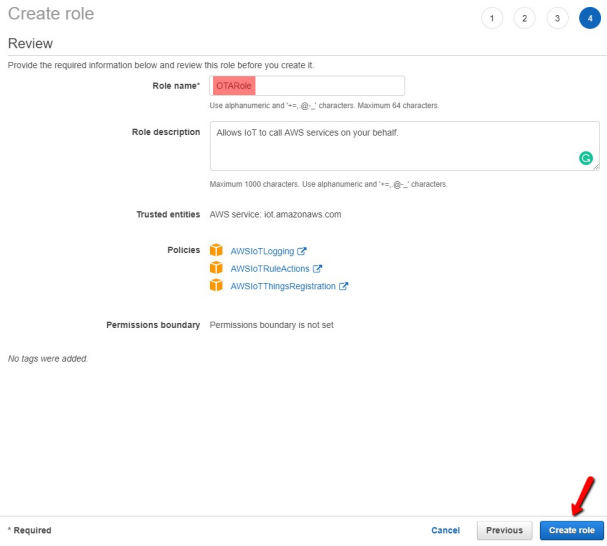


Figure 53. Create role

- Add OTA update permissions to the OTA service role

1. In the search box on the IAM console page, enter the name of the role, and then choose it from the list.

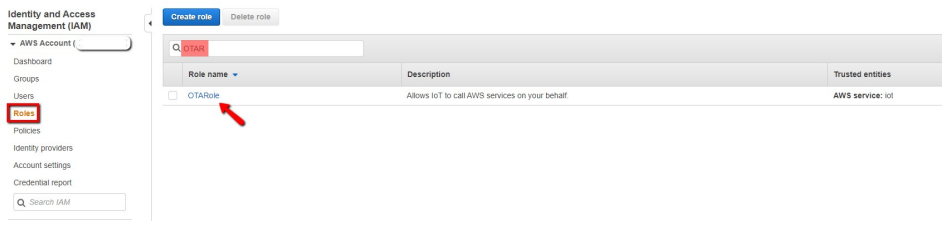


Figure 54. Role search

2. Choose **Attach policies**.

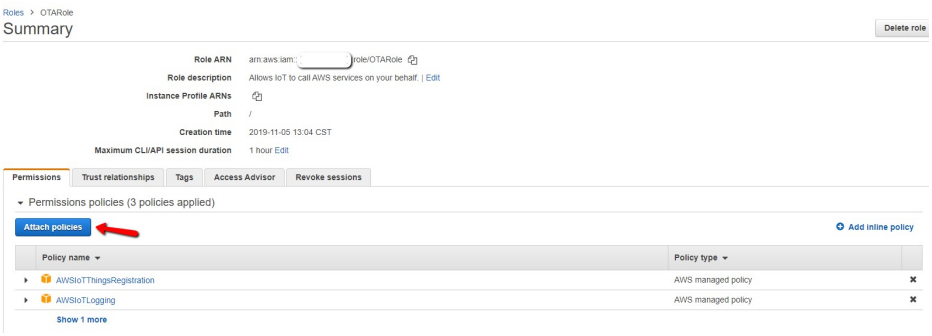


Figure 55. Attach policies

3. In the Search box, enter AmazonFreeRTOSOTAUpdate, select AmazonFreeRTOSOTAUpdate from the list of filtered policies, and choose **Attach policy** to attach the policy to the service role.

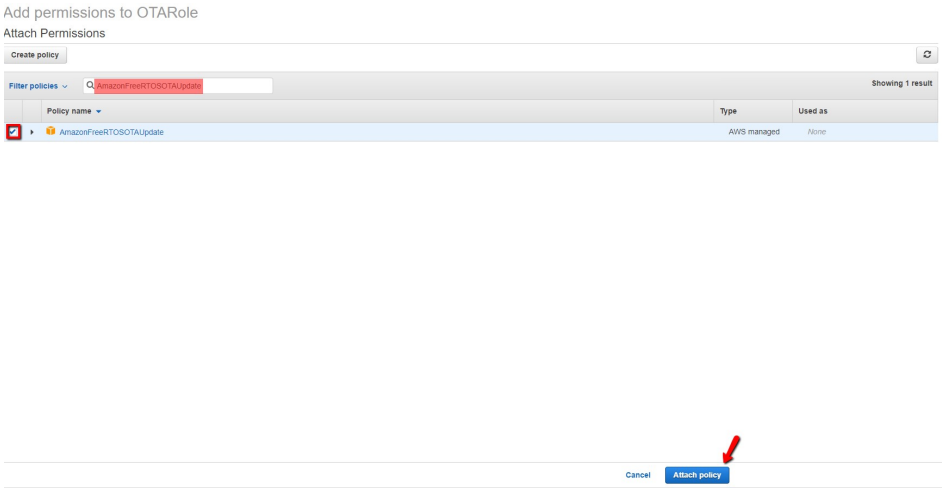


Figure 56. Attach AmazonFreeRTOSOTAUpdate policy

- Add the required IAM permissions to the OTA service role

1. Choose **Add inline policy**.

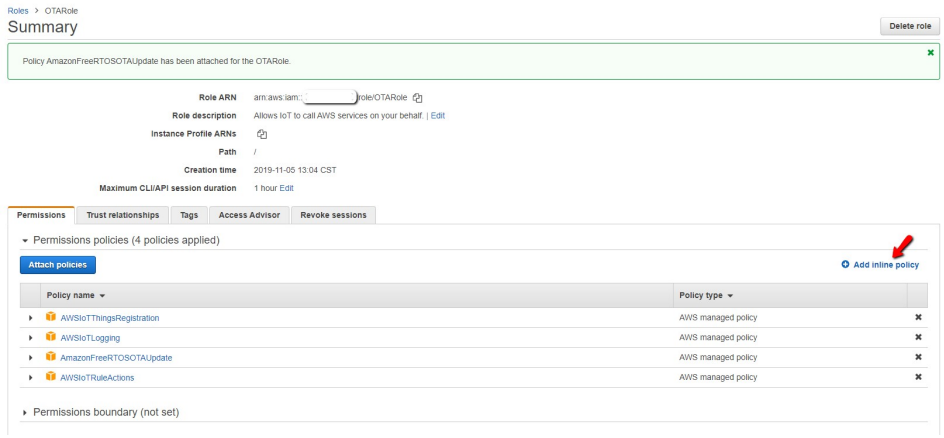


Figure 57. Add inline policy

2. Choose the **JSON** tab.
3. Copy and paste the following policy document into the text box:

```
{
       "Version": "2012-10-17",
       "Statement": [
          {
            "Effect": "Allow",
            "Action": [
              "iam:GetRole",
              "iam:PassRole"
            ],
            "Resource": "arn:aws:iam::<your_account_id>:role/<your_role_name>"
          }
        ]
      }
```

Make sure to replace <your_account_id> with the AWS account ID, and <your_role_name> with the name of the OTA service role.

4. Choose **Review policy**.

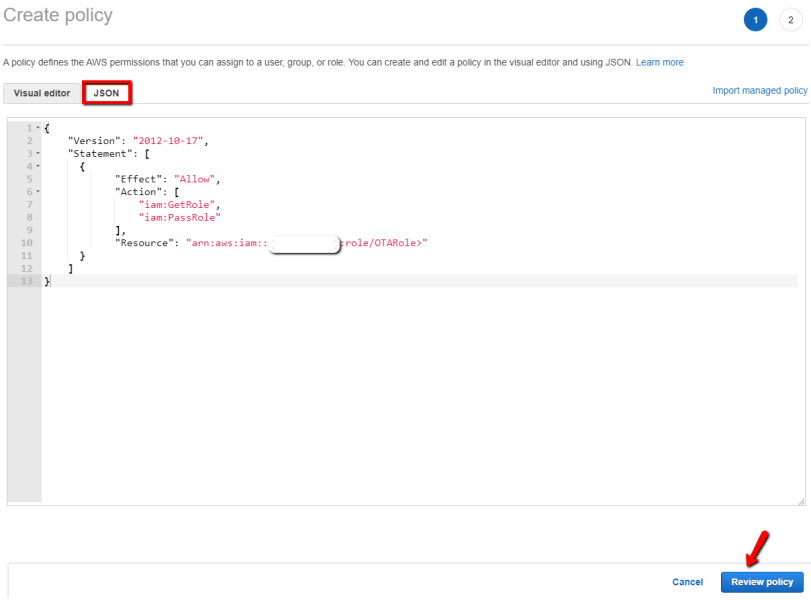


Figure 58. Review policy

5. Enter a name for the policy, and then choose **Create policy**.

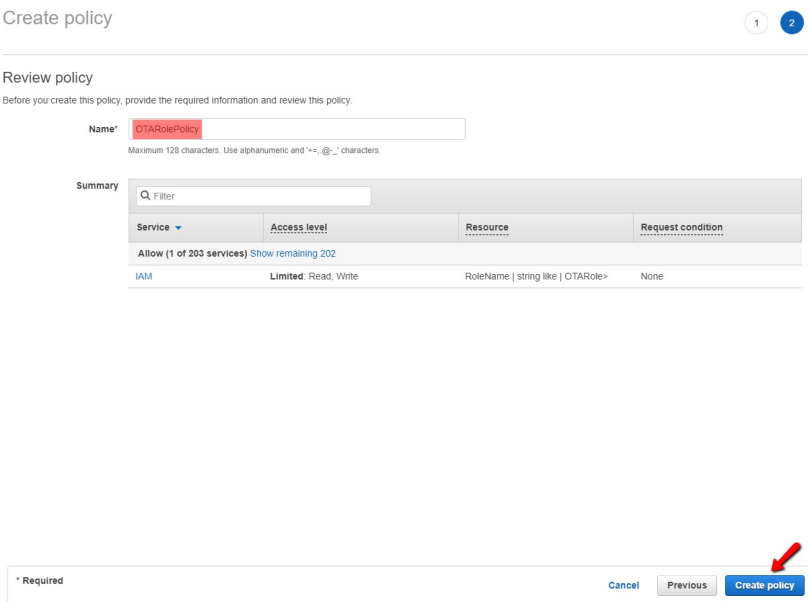


Figure 59. Create policy

- Add the required Amazon S3 permissions to the OTA service role

1. Choose **Add inline policy**.

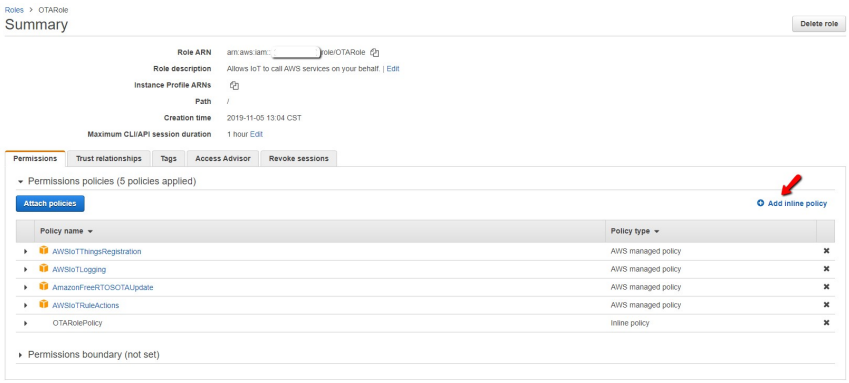


Figure 60. Add inline policy

2. Choose the **JSON** tab.
3. Copy and paste the following policy document into the box:

```
{
  "Version": "2012-10-17",
  "Statement": [
     {
        "Effect": "Allow",
        "Action": [
           "s3:ListBucketVersions",
           "s3:GetObjectVersion",
        "s3:GetObject",
        "s3:PutObject"
        ],
        "Resource": [
           "arn:aws:s3:::<example-bucket>/*",
           "arn:aws:s3:::<example-bucket>"
       ]
     }
   ]
}
```

   This policy grants the OTA service role permission to read Amazon S3 objects. Make sure to replace <example-bucket> with the name of the bucket.
4. Choose **Review policy**.
5. Enter a name for the policy, and then choose **Create policy**.

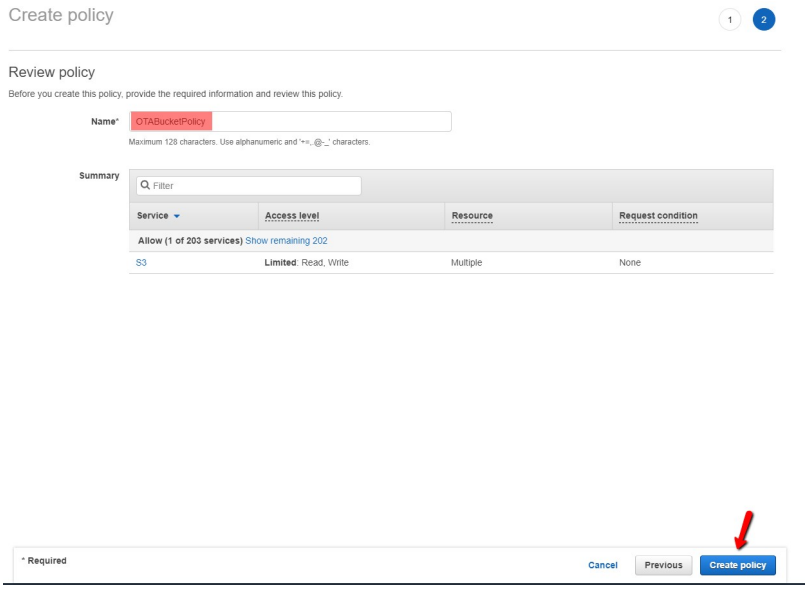


Figure 61. OTA bucket policy

- Create an OTA User Policy

1. Open the https://console.aws.amazon.com/iam/https://console.aws.amazon.com/iam/console.
2. In the navigation pane, choose **Users**.
3. Choose **Add user**.
4. Enter a user name, select **Programmatic access** type, and then choose **Next: Permissions**.

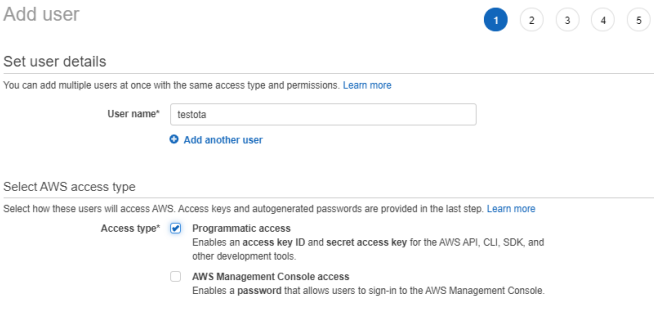


Figure 62. Add user

5. Choose **Next: Tags**.
6. Choose **Next: Review**
7. Choose **Create user**.
8. After adding the user, choose the IAM user from the list.
9. Choose **Add permissions**.

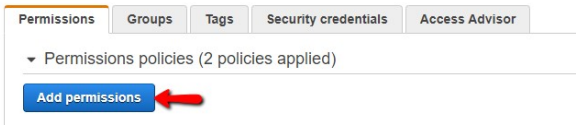


Figure 63. Add permissions

10. Choose **Attach existing policies directly,** and then choose **Create policy**.

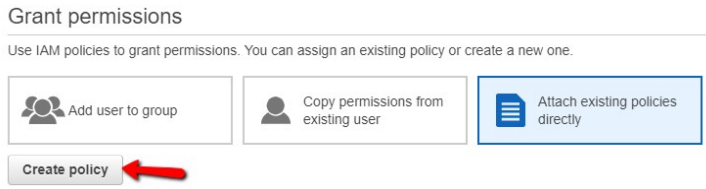


Figure 64. Create policy

11. Choose the **JSON** tab, and copy and paste the following policy document into the policy editor:

```
{
  "Version": "2012-10-17",
  "Statement": [
     {
        "Effect": "Allow",
        "Action": [
           "s3:ListBucket",
           "s3:ListAllMyBuckets",
           "s3:CreateBucket",
           "s3:PutBucketVersioning",
           "s3:GetBucketLocation",
           "s3:GetObjectVersion",
           "acm:ImportCertificate",
           "acm:ListCertificates",
           "iot:*",
           "iam:ListRoles",
           "freertos:ListHardwarePlatforms",
           "freertos:DescribeHardwarePlatform"
        ],
        "Resource": "*"
     },
     {
        "Effect": "Allow",
        "Action": [
           "s3:GetObject",
           "s3:PutObject"
        ],
        "Resource": "arn:aws:s3:::<example-bucket>/*"
     },
     {
        "Effect": "Allow",
        "Action": "iam:PassRole",
        "Resource": "arn:aws:iam::<your-account-id>:role/<role-name>"
     }
  ]
}
```

Replace <example-bucket> with the name of the Amazon S3 bucket where the OTA update firmware image is stored.

Replace <your-account-id> with the AWS account ID. The AWS account ID can be found in the upper right of the console. When entering the account ID, remove any dashes (-). Replace <role-name> with the name of the created IAM service role.

12. Choose **Review policy**.
13. Enter a name for the new OTA user policy, and then choose **Create policy**.

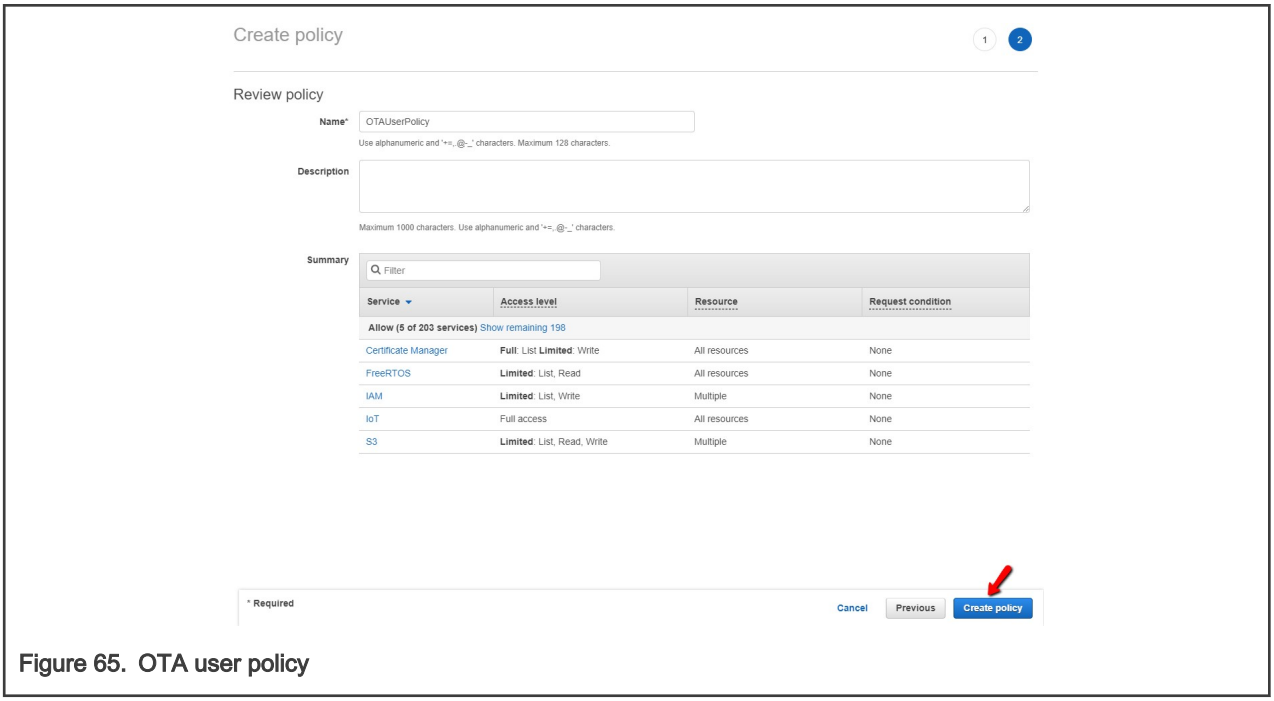


Figure 65. OTA user policy

- Windows Pre-Requisites

1. OpenSSL
   a. Install OpenSSL: https://slproweb.com/products/Win32OpenSSL.html
   b. Modify the system environment variable path to add the OpenSSL bin directory

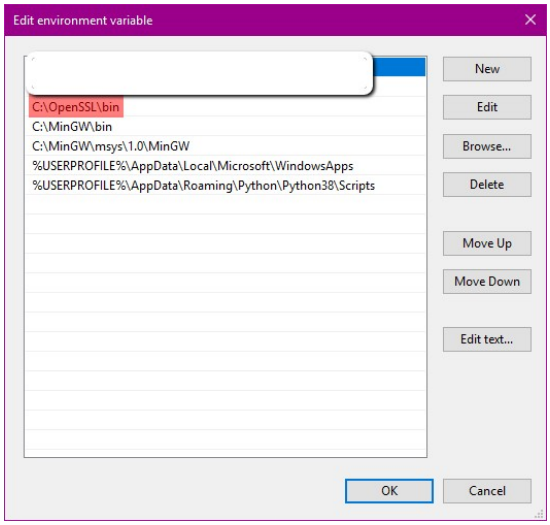


Figure 66. OpenSSL config

Make sure OpenSSL gets assigned to the OpenSSL executable in the command prompt or terminal environment.

2. Install the AWS CLI

   a. Follow the instructions for AWS CLI version 1 bundler installer https://docs.aws.amazon.com/cli/latest/userguide/install-cliv1.html

   b. Go to the IAM console https://console.aws.amazon.com/iam/

   c. In the navigation pane, choose **Users**.

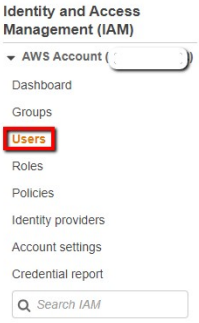


Figure 67. Users

   d. Choose the IAM user account.

   e. Select **Security credentials**.

   f. In the **Access keys** section, choose **Create access key**.

   g. To view the new access key pair, choose **Show**. The secret access key cannot be accessed again after closing this dialog box. The credentials look something like this:

   Access key ID: `AKIAIOSFODNN7EXAMPLE`

   Secret access key:`wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY`

   h. To download the key pair, choose the `Download .csv` file. Store the keys in a secure location. The secret access key cannot be accessed again after closing this dialog box. Keep the keys confidential to protect the AWS account and never email them. Do not share them outside user's organization, even if an inquiry appears to come from AWS or Amazon.com. No one who legitimately represents Amazon ever asks someone for the secret key.

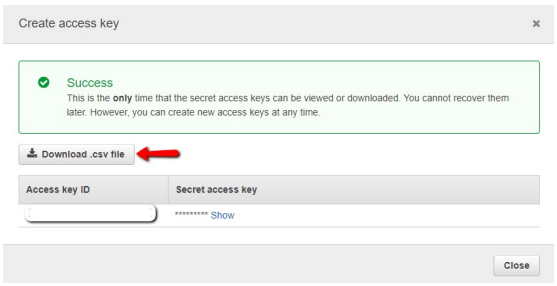


Figure 68. Download .csv file

   i. After downloading the `.csv` file, choose **Close**. Once the access key is generated, the key pair is active by default, and the pair can be used right away.

   j. For general use, the AWS configure command is the fastest way to set up the AWS CLI installation

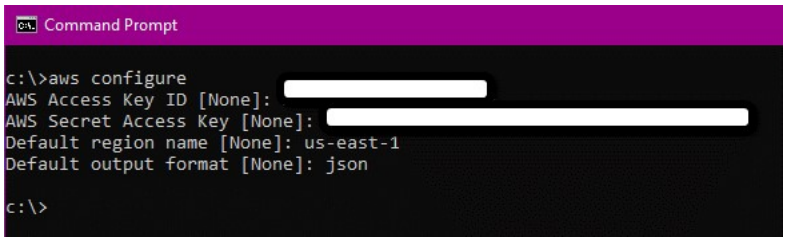


Figure 69. AWS CLI installation

3. Creating a Code-Signing Certificate

   a. In the working directory, use the following text to create a file named `cert_config.txt`. Replace *test_signer@amazon.com* with user's email address:

```
[ req ]
prompt = no
distinguished_name = my_dn
[ my_dn ]
commonName = test_signer@amazon.com
[ my_exts ]
keyUsage = digitalSignature
extendedKeyUsage = codeSigning
```

   b. Using openSSL command line, create an ECDSA code-signing private key:

```
openssl genpkey -algorithm EC -pkeyopt ec_paramgen_curve:P-256 -pkeyopt
ec_param_enc:named_curve -outform PEM -out ecdsasigner.key
```

   c. Create an ECDSA code-signing certificate:

```
openssl req -new -x509 -config cert_config.txt -extensions my_exts
 -nodes -days 365 -key ecdsasigner.key -out ecdsasigner.crt
```

   d. Import the code-signing certificate, private key, and certificate chain into AWS Certificate Manager:

```
aws acm import-certificate --certificate file://ecdsasigner.crt
--private-key file://ecdsasigner.key
```

NOTE

This command displays ARN for the certificate. Save it locally to use it while creating the OTA update job.

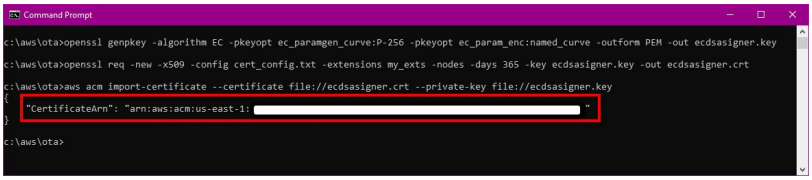


Figure 70. Creating a Code-Signing certificate

- Grant access to code signing for AWS IoT

1. Sign in to the https://console.aws.amazon.com/iam/
2. In the navigation pane, choose **Policies**.

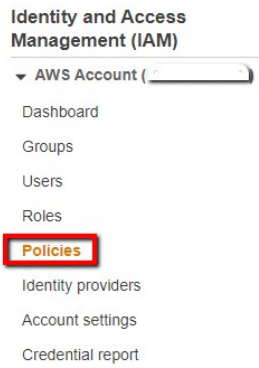


Figure 71. Policies

3. Choose **Create Policy**.
4. On the **JSON** tab, copy and paste the following JSON document into the policy editor. This policy allows the IAM user access to all code-signing operations:

```
{
  "Version": "2012-10-17",
  "Statement": [
     {
        "Effect": "Allow",
        "Action": [
           "signer:*"
        ],
        "Resource": "*"
     }
   ]
}
```

5. Choose **Review policy**.
6. Enter a policy name and description, and then choose **Create policy**.

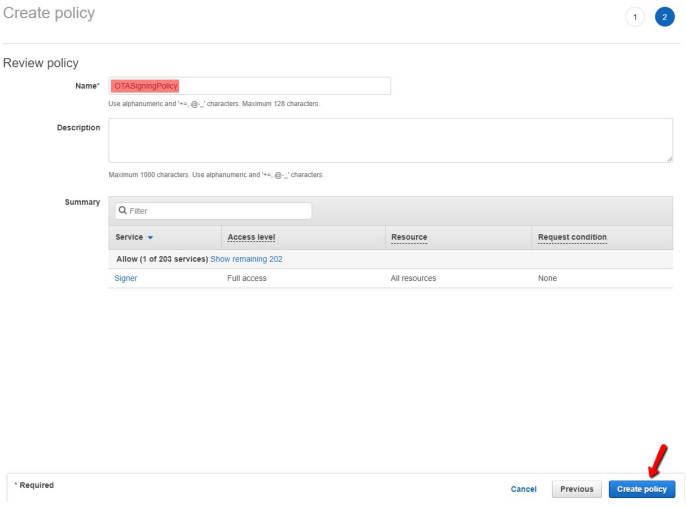


Figure 72. OTA signing policy

7. In the navigation pane, choose **Users**.

8. Choose the IAM user account.
9. On the Permissions tab, choose **Add permissions**.
10. Choose **Attach existing policies directly**, and select the checkbox next to the code-signing policy created before.

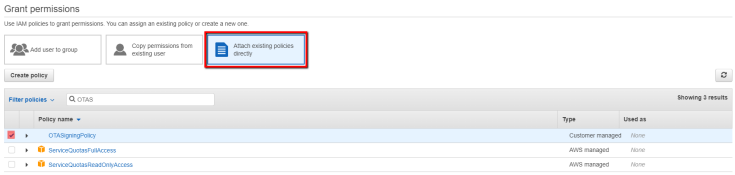


Figure 73. Attach policy

11. Choose **Next: Review**.
12. Choose **Add permissions**.

- Create an AWS IoT Thing

1. Open the AWS IoT console website https://console.aws.amazon.com/iot/

> **NOTE**
> When opening the web, first check if the region is the one which the bucket is.

2. In the navigation pane, choose **Manage** -> **Things**.
3. Choose **Create**.
4. On the Creating AWS IoT things page, choose **Create a single thing**.

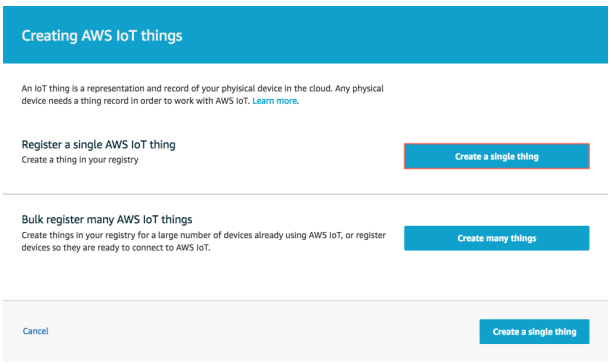


Figure 74. Create a single thing

5. On the Create a thing page, in the **Name** field, enter a name for the thing, such as MyThing. Choose **Next**.

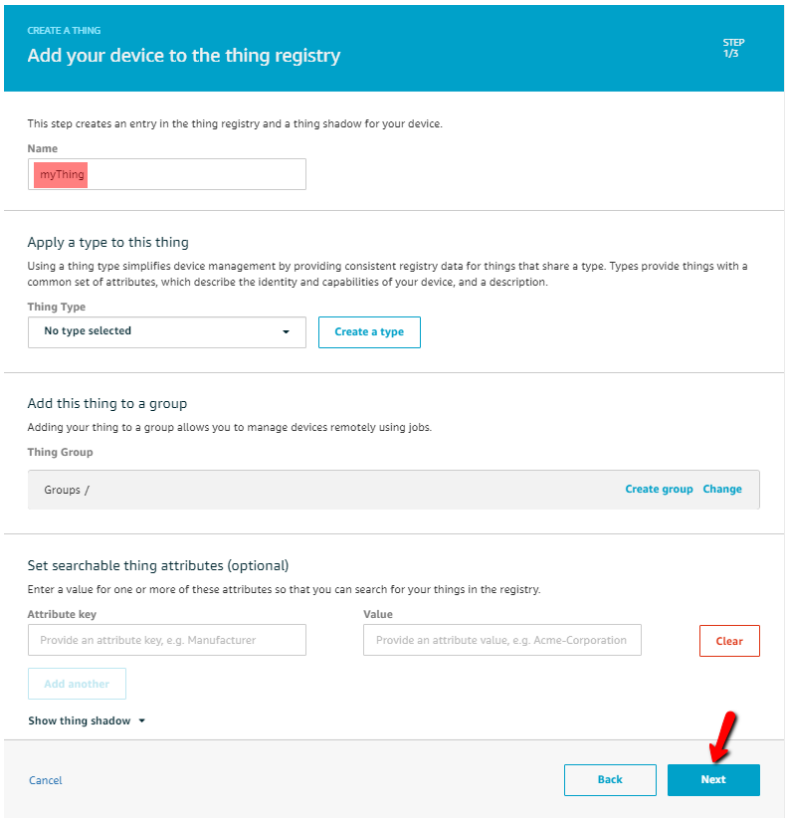


Figure 75. Thing name

6. On the Add a certificate for the thing page, choose **Create certificate**. It generates an X.509 certificate and key pair.

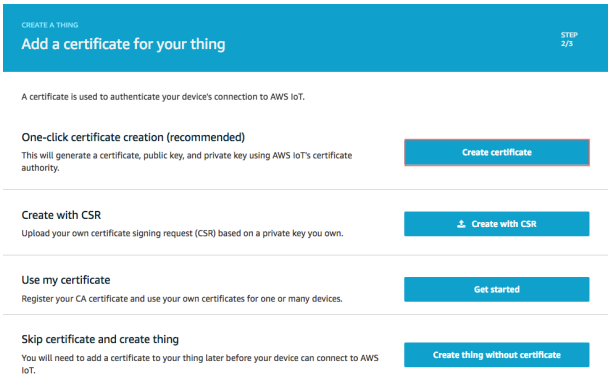


Figure 76. Create certificate

7. On the 'Certificate created!' page, download the public and private keys, certificate, and root certificate authority (CA).
8. Choose **Download** for the certificate.
9. Choose **Download** for the private key.
10. Choose **Download** for the Amazon root CA. A new webpage is displayed. Choose **RSA 2048 bit key: Amazon Root CA1**. It opens another webpage with the text of the root CA certificate. Copy this text and paste it into a file named `Amazon_Root_CA_1.pem`.

Most web browsers save downloaded files into a Downloads directory. Copy these files to a different directory when running the sample applications. Choose **Activate** to activate the X.509 certificate, and then choose **Attach a policy**.

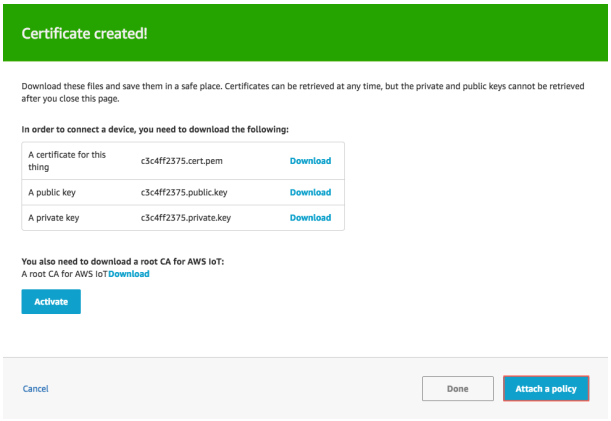


Figure 77. Files download

11. On the `Add a policy for your thing` page, choose **Register Thing**. After registering the thing, create and attach a new policy to the certificate.

- Create an AWS IoT Policy

1. Open the AWS IoT console website: https://console.aws.amazon.com/iot/.
2. In the left navigation pane, choose **Secure**, choose **Policies**, then choose **Create**.
3. On the `Create a policy` page, in the **Name** field, enter a name for the policy (for example, `MyIotPolicy`). In the **Action** field, enter `iot:*`. In the **Resource ARN** field, enter `*`. Select the **Allow** checkbox. It allows all clients to connect to AWS IoT. After entering the information for the policy, choose **Create**.

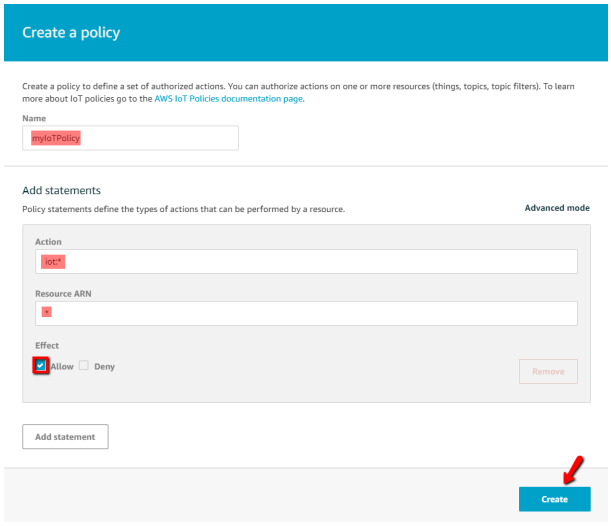


Figure 78. Create IoT policy

- Attach an AWS IoT Policy to a Device Certificate

1. Open the AWS IoT console website:https://console.aws.amazon.com/iot/.
2. In the left navigation pane, choose **Secure**, and then choose **Certificates**.
3. In the box for the certificate created, choose ... to open a drop-down menu, and then choose **Attach policy**.

4. In Attach policies to certificate(s), select the checkbox next to the policy created in the previous step, and then choose **Attach**.

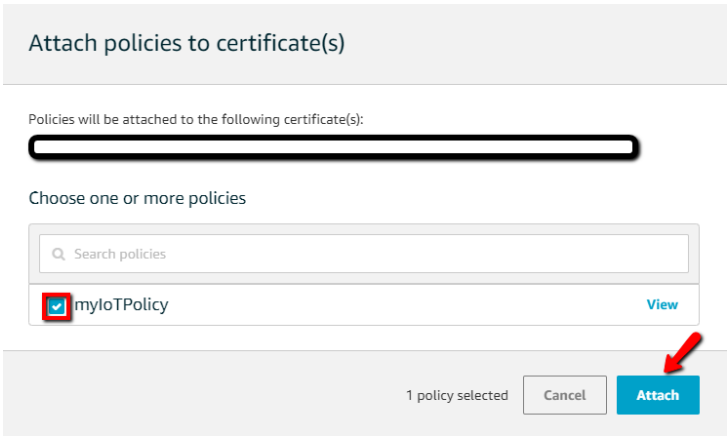


Figure 79. Attach IoT policy

- Attach a Certificate to a Thing

1. Open the AWS IoT console website: https://console.aws.amazon.com/iot/.
2. In the left navigation pane, choose **Secure**, and then choose **Certificates.**
3. In the box for the certificate created, choose ... to open a drop-down menu, and then choose **Attach thing**.
4. In Attach things to certificate(s), select the checkbox next to the thing registered, and then choose **Attach**.

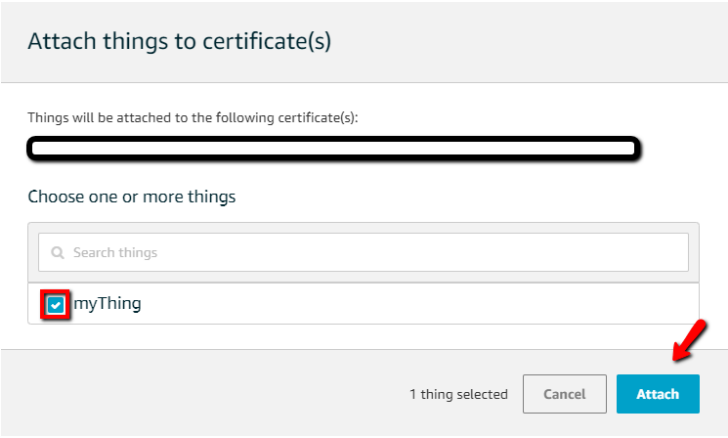


Figure 80. Attach Thing

5. To verify that the thing is attached, select the box for the certificate.

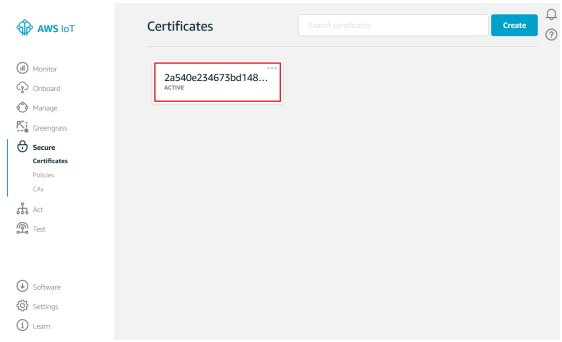


Figure 81. Select certificate to verify

6. On the **Details** page for the certificate, in the left navigation pane, choose **Things**.

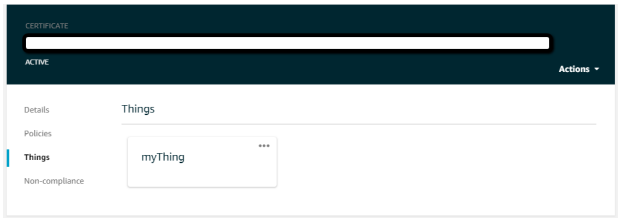


Figure 82. Verify attached thing

7. To verify that the policy is attached, on the Details page for the certificate, in the left navigation pane, choose **Policies**.

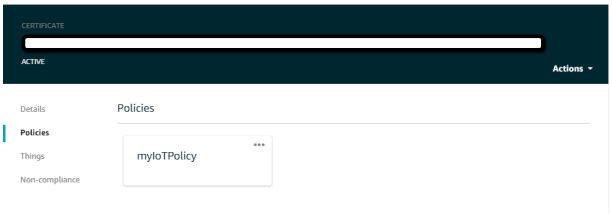


Figure 83. Verify attached policy

7.3.1.2 Prepare the SBL

In SBL project, enter the `sbl/target/evkmimxrt1170` path.

Disable the `Enable single image function` option and the `Enable mcu isp support` option in the menuconfig interface of Scons to disable single image mode and disable MCU ISP support.

Compile the SBL project and download it to the target board.

NOTE

1. If the new signature key is used, modify the `sign-rsa2048-pub.c`
2. Programming SBL image by drag-drop of DAPLink may erase the whole flash.

7.3.1.3 Prepare the SFW

To prepare SFW config in an SFW project, follow the steps below:

1. Generate `aws_clientcredential_keys.h` file.
   a. Enter `sfw/firmware/aws_ota/tool` path.

b. Using a web browser, open the `CertificateConfigurator.html`.

c. Browse to the Certificate and Key files downloaded from the Thing in '**Create an AWS IoT Thing'** part of section 7.3.1.1. Click on Generate and save `aws_clientcredential_keys.h`.

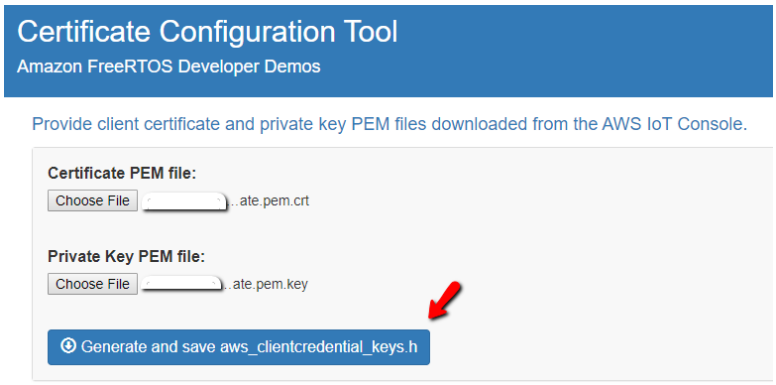


Figure 84. Generate `aws_clientcredential_keys.h`

d. Replace the `sfw/firmware/aws_ota/demos/include/aws_clientcredential_keys.h` with the file generated in the c) step.

2. Modify the `aws_ota_codesigner_certificate.h` file.

a. Open the `ecdsaigner.crt` file generated in '**Windows Pre-Requisites**' part of section 7.3.1.1 using a text editor.

b. Open`sfw/firmware/aws_ota/demos/include/aws_ota_codesigner_certificate.h` file.

c. Copy all the content in `ecdsaigner.crt` and paste to `aws_ota_codesigner_certificate.h` in the `signingcredentialSIGNING_CERTIFICATE_PEM`.

NOTE

Be sure to add ' " ' at the begging of a line and ' \n" ' on every line break as below figure.

```
/*
 * PEM-encoded code signer certificate
 *
 * Must include the PEM header and footer:
 * "-----BEGIN CERTIFICATE-----\n"
 * "...base64 data...\n"
 * "-----END CERTIFICATE-----\n";
 */
static const char signingcredentialSIGNING_CERTIFICATE_PEM[] =
"-----BEGIN CERTIFICATE-----\n"
"MIIBYTCCAQegAwIBAgIJAKCX9bIhkilFMAoGCCqGSM49BAMCMCMxITAfBgNVBAMM\n"
"GEFsZWphbmRyYS5HdXptYW5AbnhwLmNvbTAeFw0xOTEwMjMxNjMzNDJaFw0yMDEw\n"
"MjIxNjMzNDJaMCMxITAfBgNVBAMMGEFsZWphbmRyYS5HdXptYW5AbnhwLmNvbTBZ\n"
"MBMGByqGSM49AgEGCCqGSM49AwEHA0IABGUghBD5lmFlJ3wf4LYsQ2VgOaDpg98G\n"
"dNC38FWGS7owI4NC5848JumrD8SonnnXpu77Pt7ShuW39hC3Vdi7z1GjJDAiMAsG\n"
"A1UdDwQEAwIHgDATBgNVHSUEDDAKBggrBgEFBQcDAzAKBggqhkjOPQQDAgNIADBF\n"
"AiEAr0pNzlaMax4arCPNiW9HYFdQTvUGyZdRLcDrUol/LQoCIH2U2REoZ59V7r6z\n"
"CMLfHA+kWq84IjxDUE2OgV60RVvC\n"
"-----END CERTIFICATE-----\n";
```

Figure 85. Certificate format

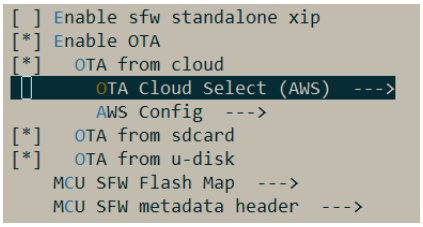
3. Enter `sfw/target/evkmimxrt1170` path.

4. Double-click the batch file `env.bat`

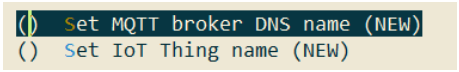
5. Input the `scons --menuconfig` command to configure the `evkmimxrt1170` project

6. Select **MCU SFW core**.

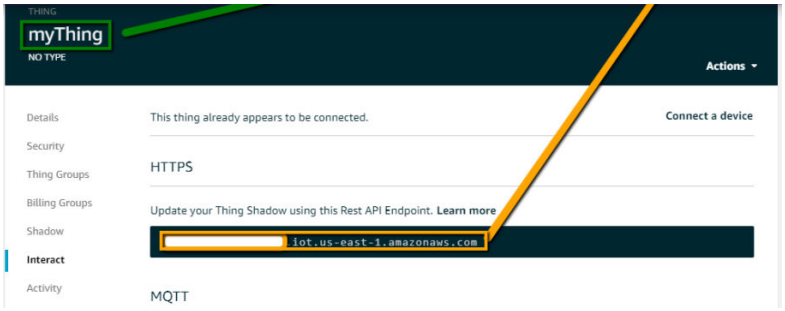
7. Disable `Enable sfw standalone xip` option, enable OTA, and select AWS OTA cloud.



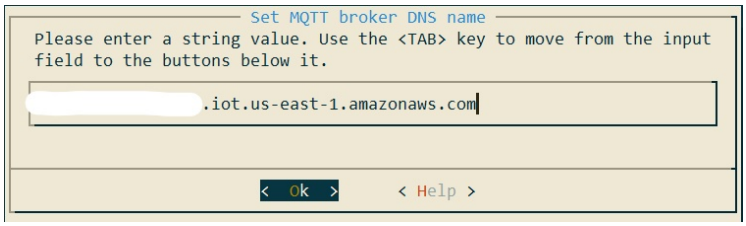
8. To enter below config menu, select **AWS Config**.



9. Input MQTT DNS name.
   a. Open the AWS IoT console website https://console.aws.amazon.com/iot/
   b. In the navigation pane, choose **Manage – Things** and select the previously created Thing.
   c. In the navigation pane, choose **Interact**



   d. Select **Set MQTT broker DNS name**, copy **Rest API Endpoint** and paste.



   e. Press **Ok**

10. Input IoT Thing name.
    a. Select **Set IoT Thing name**, copy **IoT Thing** name, and paste.

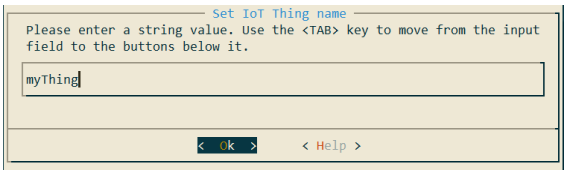


Figure 90. Thing name set

b. Press **Ok**

11. Select **MCU SFW Component -> secure**
12. Enable `mbedtls` and modify `mbedtls` config file to `aws_mbedtls_config.h`, and press **Ok**

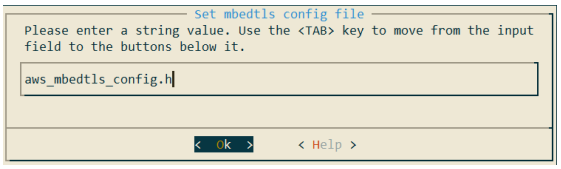


Figure 91. Modify `mbedtls` config file

13. Exit and save the configuration.

Figure 92. Save config

7.3.1.4 Prepare image

1. Enter the `sfw/target/evkmimxrt1170` path, double click the batch file `env.bat`
2. Input command to generate the iar project.

NOTE

To generate the keil or gcc project, refer to section 2.

3. Enter the `sfw/target/evkmimxrt1170/iar` path, open `sfw.eww` project.
4. Go to options, select generate additional output, and choose raw binary format.

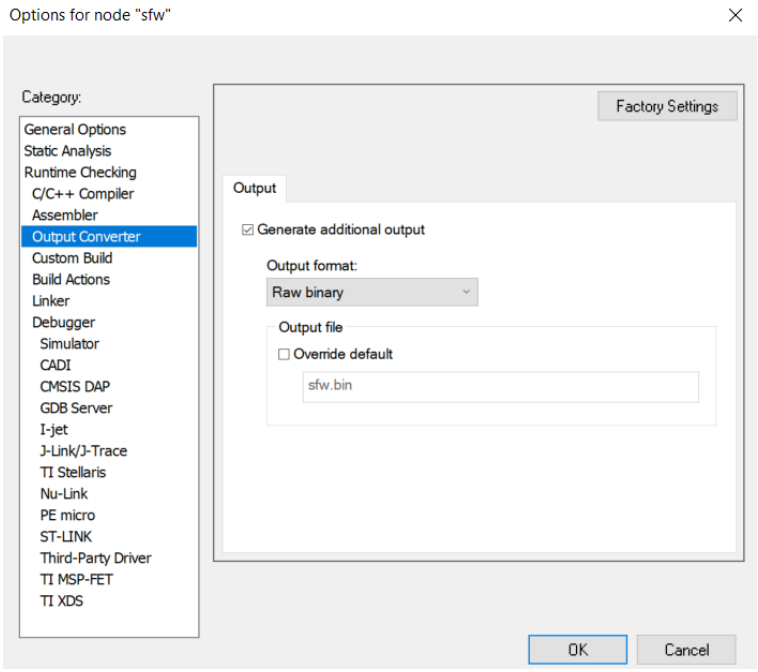


Figure 93. Modify to generate `.bin` file

5. Check the application version in `sfw/firmware/aws_ota/main_enet.c` file

```
119  #define APP_VERSION_MAJOR 0
120  #define APP_VERSION_MINOR 9
121  #define APP_VERSION_BUILD 2
```

Figure 94. Application version

6. Click the `make` button to start building the application.
7. If the build is successful, sfw.bin is generated in `sfw/target/evkmimxrt1170/iar/build/iar/Exe` folder. Change its name according to the application version. Move `sfw_092.bin` to the `sbl/component/secure/mcuboot/scripts` folder.
8. Change `APP_VERSION_BUILD` to 3 to build a newer image. Rename the new bin file to `sfw_093.bin` and also move it to `sbl/component/secure/mcuboot/scripts` folder.

```
119  #define APP_VERSION_MAJOR 0
120  #define APP_VERSION_MINOR 9
121  #define APP_VERSION_BUILD 3
```

Figure 95. New version

9. Sign `sfw_092.bin` and `sfw_093.bin` images with RSA using below commands. Then `sfw092.bin` and `sfw093.bin` are generated.

```
python imgtool.py sign --key sign-rsa2048-priv.pem --align 4 --version "0.9.2" --header-size
0x400 --pad-header --slot-size 0x100000 --max-sectors 32 sfw_092.bin sfw092.bin
```

```
python imgtool.py sign --key sign-rsa2048-priv.pem --align 4 --version "0.9.3" --header-size
0x400 --pad-header --slot-size 0x100000 --max-sectors 32 sfw_093.bin sfw093.bin
```

7.3.1.5 Upload new image to S3 bucket

1. Use AWS console to open the S3 service https://console.aws.amazon.com/s3
2. Select the previously created bucket.
3. Click **Upload**.
4. Drag and drop `sfw093.bin`.
5. Click **Upload**.

7.3.1.6 Create OTA Job

1. Open the AWS IoT console website https://console.aws.amazon.com/iot/
2. In the navigation pane, choose **Manage – Jobs**
3. Select **Create job**
4. Choose **Create FreeRTOS OTA update job,** then choose **Next**.
5. In step 1, Input **Job name**, then choose **Next**.

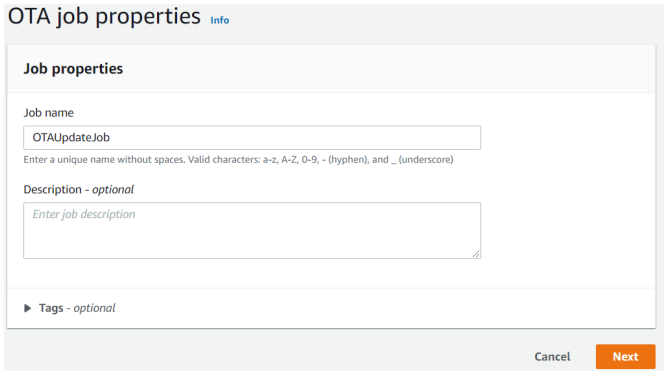


Figure 96. Input job name

6. In step 2, choose **Thing** created in previous section, choose **MQTT**, and **Sign a new file for me**

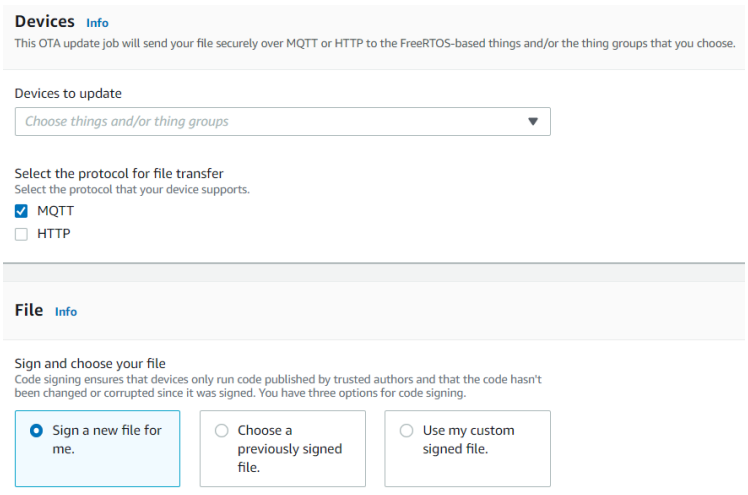


Figure 97. Thing, protocol, and file select

7. Under **Code signing profile**, choose **Create new profile**.

8. Enter a name for the code-signing profile.
    a. Under **Device hardware platform**, choose **Windows Simulator**.

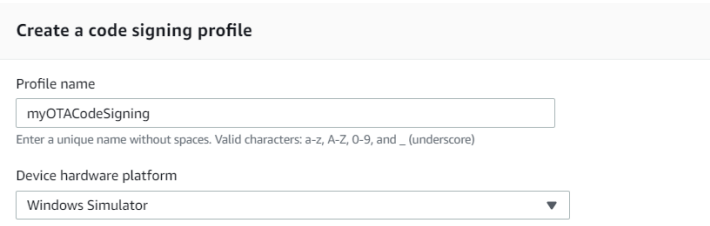


Figure 98. Device hardware platform

    b. Under **Code signing certificate**, choose **Import new code signing certificate**, and browse for the certificate files created with AWS CLI, then choose **Import**.

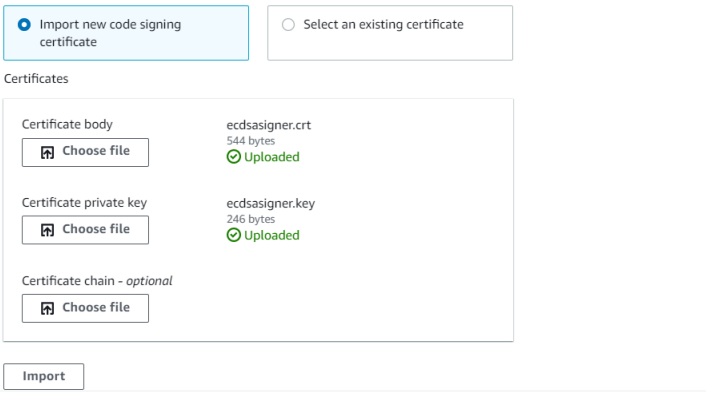


Figure 99. Select certificate and key

    c. Under **Pathname of code signing certificate on device**, type the default `path /certificates/authcert.pem`, then click **Create**.

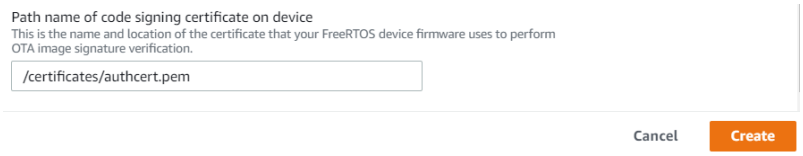


Figure 100. Pathname of certificate

9. Under **File**, choose **Select an existing file,** then choose **Browse S3** to select `sfw093.bin` file uploaded in S3.

NOTE

Make sure that the region must be the correct one where the bucket is located. Otherwise the uploaded binary cannot be found.

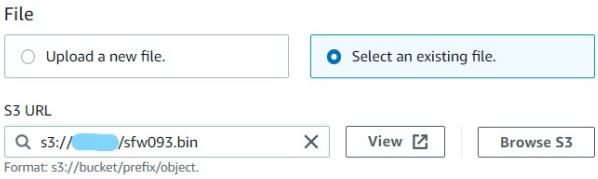


Figure 101. Image select

10. Under **Pathname of file on device**, type the default path /device/updates.

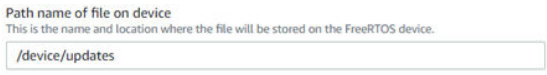


Figure 102. Pathname of file

11. Under **IAM role**, choose the role created in previous steps.

Figure 103. Select role

12. Choose **Next**.
13. Under **Job run type**, choose first item.

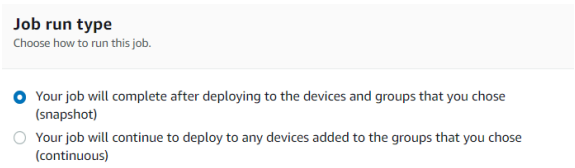


Figure 104. Job run type

14. Before clicking **Create job**, run the application.

7.3.1.7 Run the application

1. Use the `MCUBootUtility` tool to download the `sfw092.bin` generated previously to the first slot of the board. The default location of slot1 is the **flash_offset+0x100000** to **flash_offset+0x200000**, the whole slot size is 1 MB.

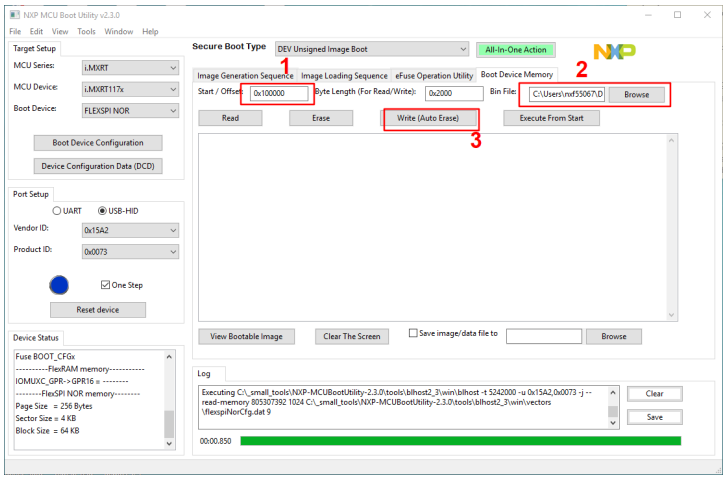


Figure 105. Download `sfw092.bin` to slot1

2. After successfully downloading the image, reset the board. The debug console prints the application log as shown:

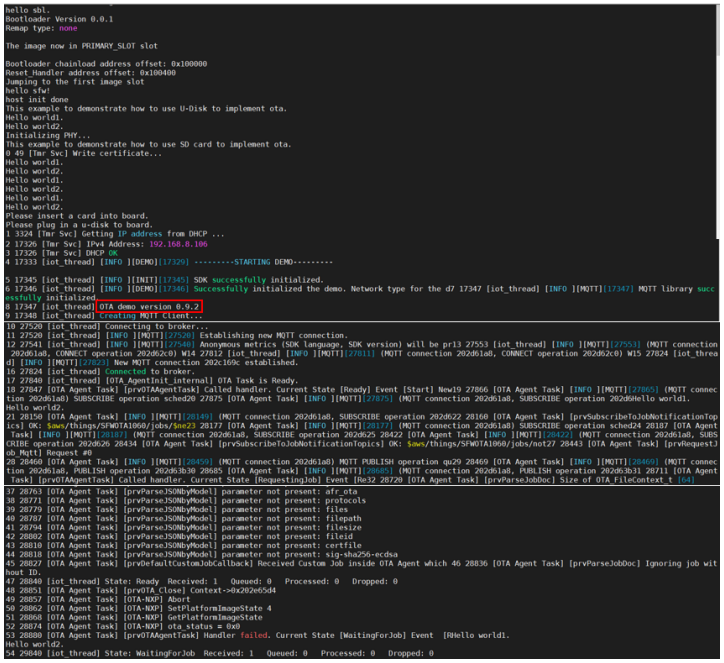


Figure 106. Application log

3. When running the application, wait until the message of the OTA State Ready appears in the serial terminal as shown:

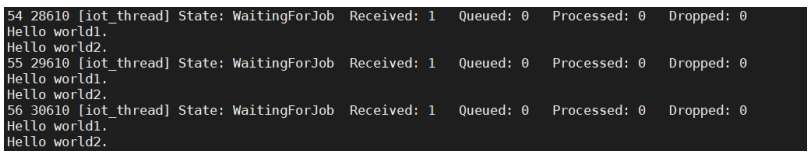


Figure 107. OTA ready log

4. At this point, the OTA agent is waiting for an OTA job. Go back to the Create OTA job window and click **Create job**.

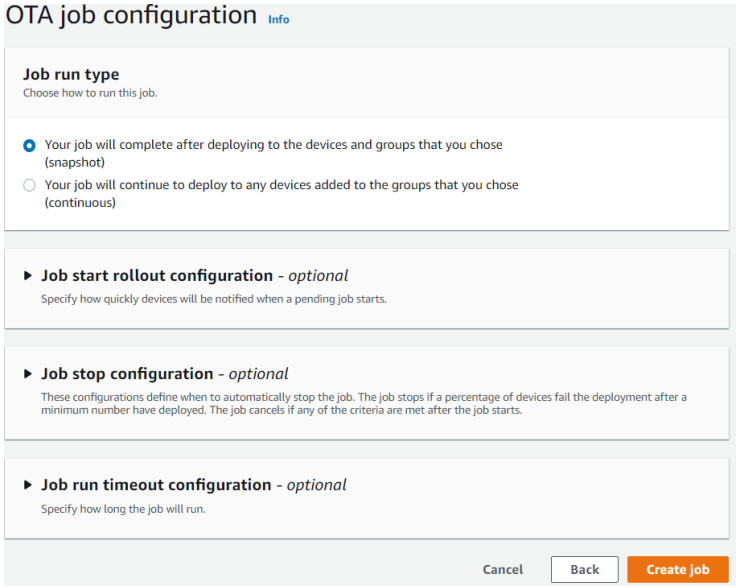


Figure 108. Create job

5. The process starts, and the outputs are as below.
   a. Start file transfer

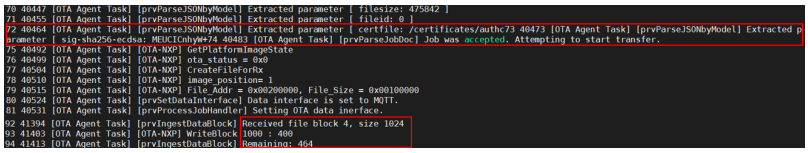


Figure 109. Start file transfer log

   b. Received the whole file

Figure 110. Received whole file log

   c. Check file signature

Figure 111. File signature check log

   d. Check image version

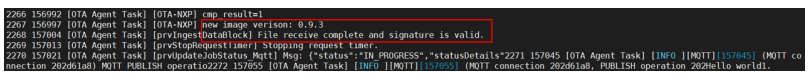


Figure 112. Image version check log

   e. Write update type

Figure 113. Write update type log

f. Write image trailer

```
2283 158208 [OTA Agent Task] [OTA-NXP] Write image trailer
write magic number offset = 0xfff0
```

Figure 114. Write image trailer log

g. Active new image, device reset

```
2284 158218 [OTA Agent Task] [OTA-NXP] ActivateNewImage
2285 158224 [OTA Agent Task] [OTA-NXP] ResetDevice
```

Figure 115. Active new image log

h. Running new image

```
hello sbl.
Bootloader Version 0.0.1
Remap type: test

The image now in SECONDARY_SLOT slot

Bootloader chainload address offset: 0x100000
Reset_Handler address offset: 0x100400
Jumping to the first image slot
hello sfw!
host init done
This example to demonstrate how to use U-Disk to implement ota.
Hello world1.
Hello world2.
Initializing PHY...
This example to demonstrate how to use SD card to implement ota.
0 49 [Tmr Svc] Write certificate...
2 17172 [Tmr Svc] IPv4 Address: 192.168.8.106
3 17172 [Tmr Svc] DHCP OK
4 17175 [iot_thread] [INFO ][DEMO][17175] ---------STARTING DEMO---------

5 17191 [iot_thread] [INFO ][INIT][17191] SDK successfully initialized.
6 17192 [iot_thread] [INFO ][DEMO][17192] Successfully initialized the demo. Network type for the d7 17193 [iot_thread] [INFO ][MQTT][17193] MQTT library succ
essfully initialized.
8 17193 [iot_thread] OTA demo version 0.9.3
9 17194 [iot_thread] Creating MQTT Client...
```

Figure 116. Run new image log

i. Self-test

```
56 27756 [OTA Agent Task] [prvOTAAgentTask] Called handler. Current State [WaitingForJob] Event [Re57 27765 [OTA Agent Task] [prvInSelfTestHandler] prvInSelfT
estHandler, platform is in self-test.
58 27774 [OTA Agent Task] [OTA-NXP] GetPlatformImageState
59 27781 [OTA Agent Task] [OTA-NXP] ota_status = 0x1
```

Figure 117. Self-test log

j. Write OK flag

```
64 27814 [OTA Agent Task] [OTA-NXP] ota_status = 0x1
Write OK flag: off = 0xfffe0
```

Figure 118. Write OK flag log

k. OTA success

```
65 27824 [OTA Agent Task] [prvStopSelfTestTimer] Stopping the self test timer.
66 27832 [OTA Agent Task] [prvUpdateJobStatus_Mqtt] Msg: {"status":"SUCCEEDED","statusDetails":{"re67 27855 [OTA Agent Task] [INFO ][MQTT][27855] (MQTT connec
tion 202d61a8) MQTT PUBLISH operation qos8 27865 [OTA Agent Task] [INFO ][MQTT][27865] (MQTT connection 202d61a8, PUBLISH operation 202d63bHello world1.
Hello world2.
69 28113 [OTA Agent Task] [INFO ][MQTT][28113] (MQTT connection 202d61a8, PUBLISH operation 202d63b70 28124 [OTA Agent Task] [prvUpdateJobStatus_Mqtt] 'SUCCEE
DED' to $aws/things/SFWOTA1060/jobs/AFR_71 28133 [OTA Agent Task] [prvOTAAgentTask] Called handler. Current State [CreatingFile] Event [Sta72 28435 [iot_threa
d] State: WaitingForJob  Received: 1   Queued: 0   Processed: 0   Dropped: 0
```

Figure 119. OTA success log

6. After OTA success, push the **Reset** button on the board to confirm that the update is successful, the `sfw093.bin` log must be printed.

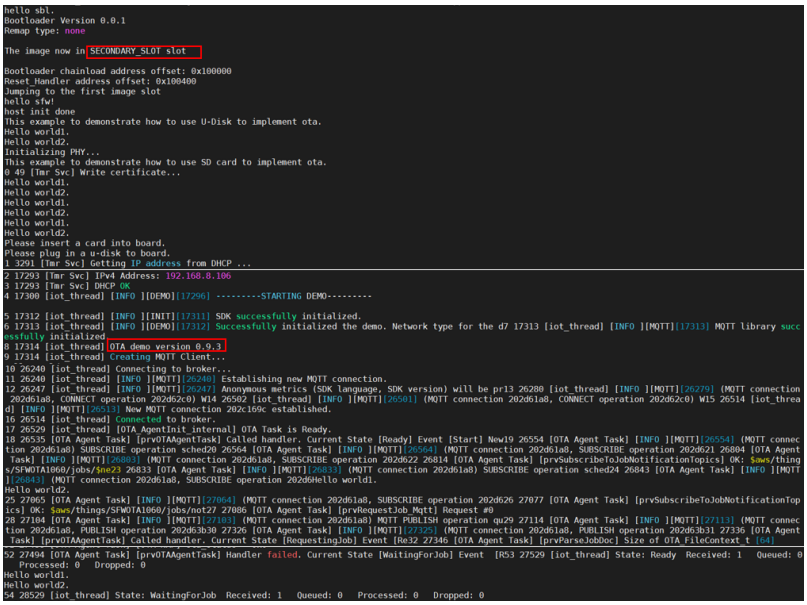


Figure 120. New image log

7.3.2 Aliyun OTA

This section walks through the steps of how to perform the Aliyun OTA firmware update of the board using Aliyun IoT and use the EVKMIMXRT1064 platform as an example to demonstrate the testing process.

7.3.2.1 Create a testing device

1. Open the link: https://iot.console.aliyun.com/. Register an account to log in to the platform and create the Alibaba Cloud account.

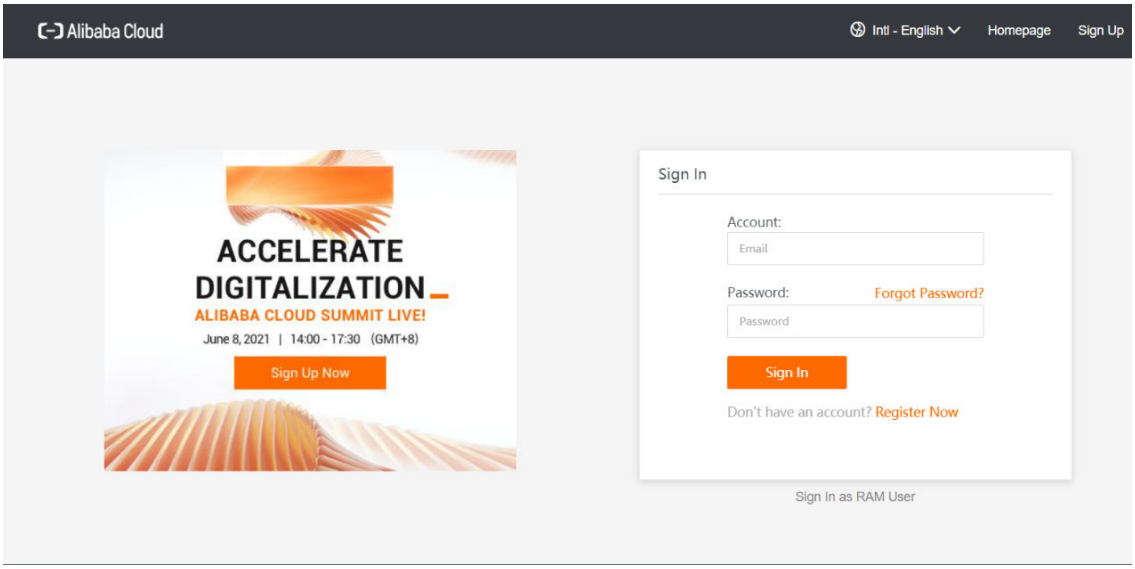


Figure 121. Account creation interface

2. After the login is successful, the page of the Alibaba Cloud IOT platform is displayed, click to enter the "**Public Instance**" interface.

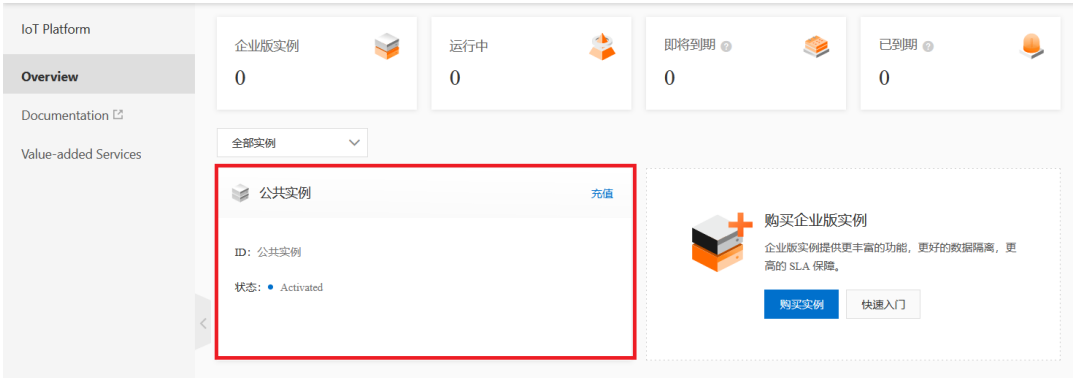


Figure 122. Alibaba Cloud IOT platform

3. After entering the "**Public Instance**", click "**Create Product**", set the parameters as shown in the figure on the "**New Product**" page, customize a product name, and select the first one by default in the category (to test the OTA function).

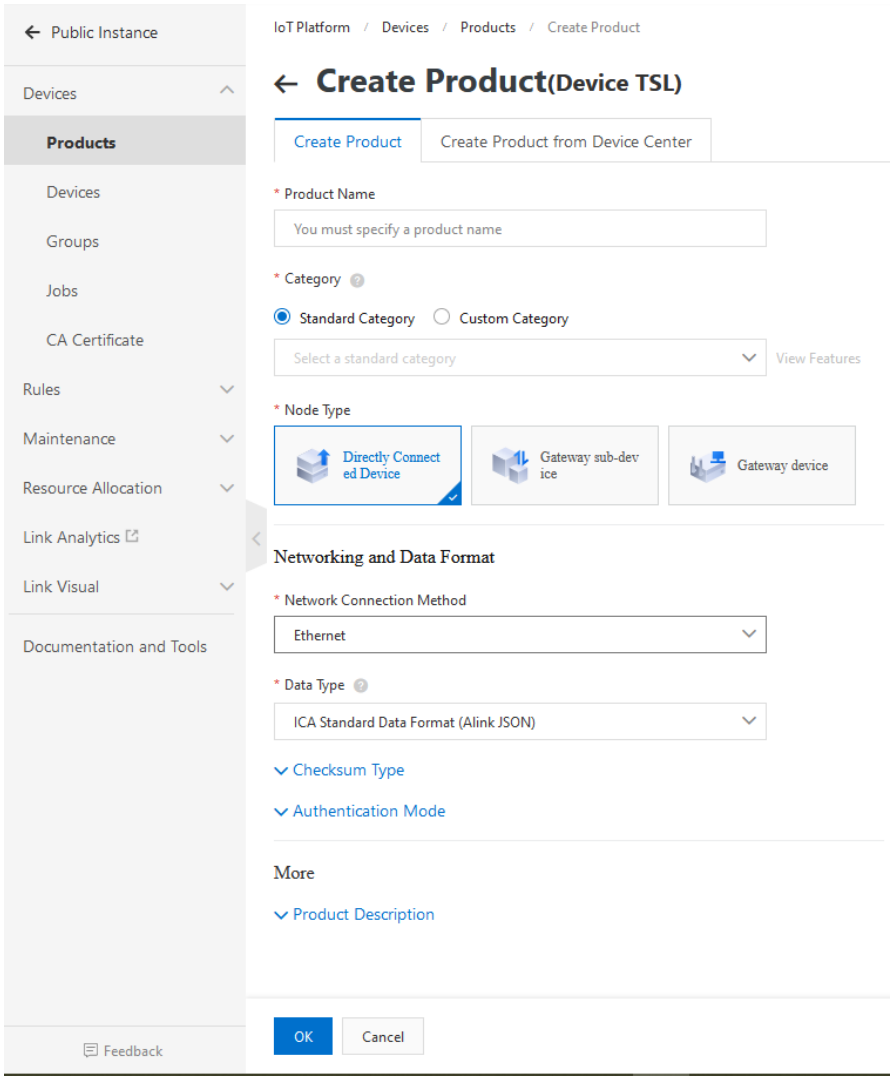


Figure 123. Create Product

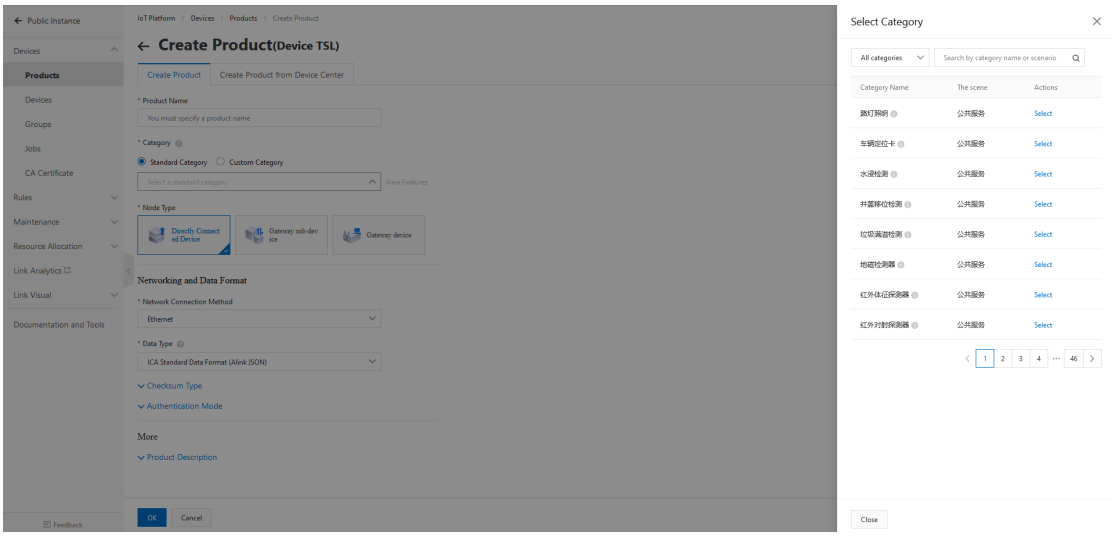


Figure 124. Product Type

Add equipment to the product after it is successfully created.

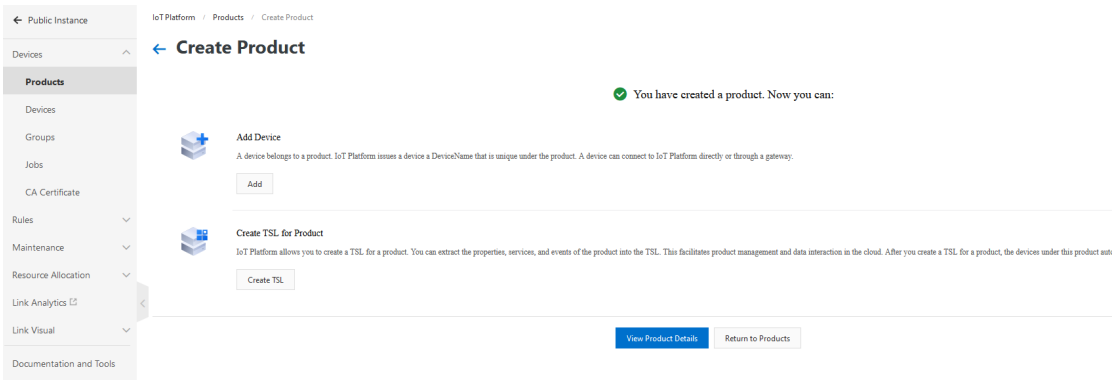


Figure 125. Create product successfully

4. Select the product that needs to add a device for testing, and click the blue button of “**Add Device**” to set the device name.

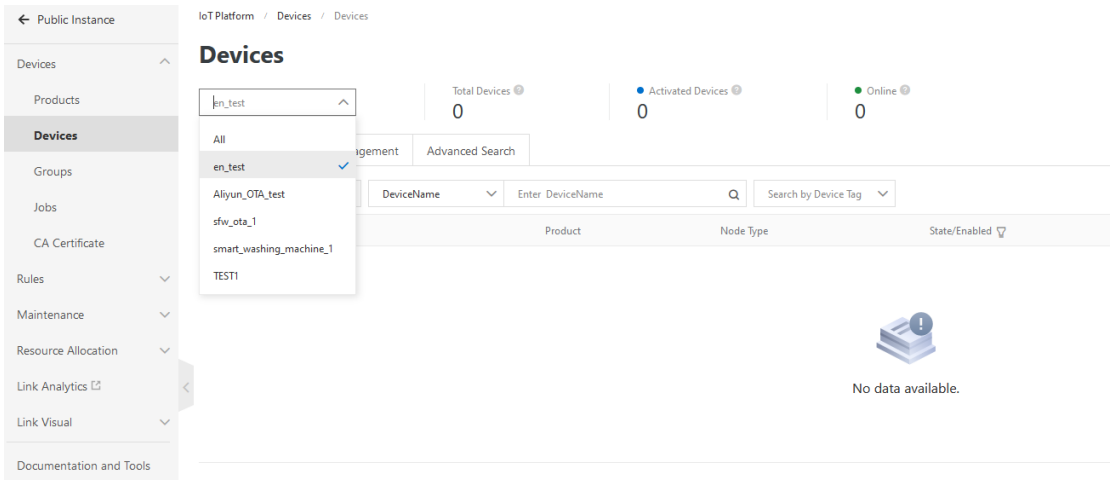


Figure 126. Add device

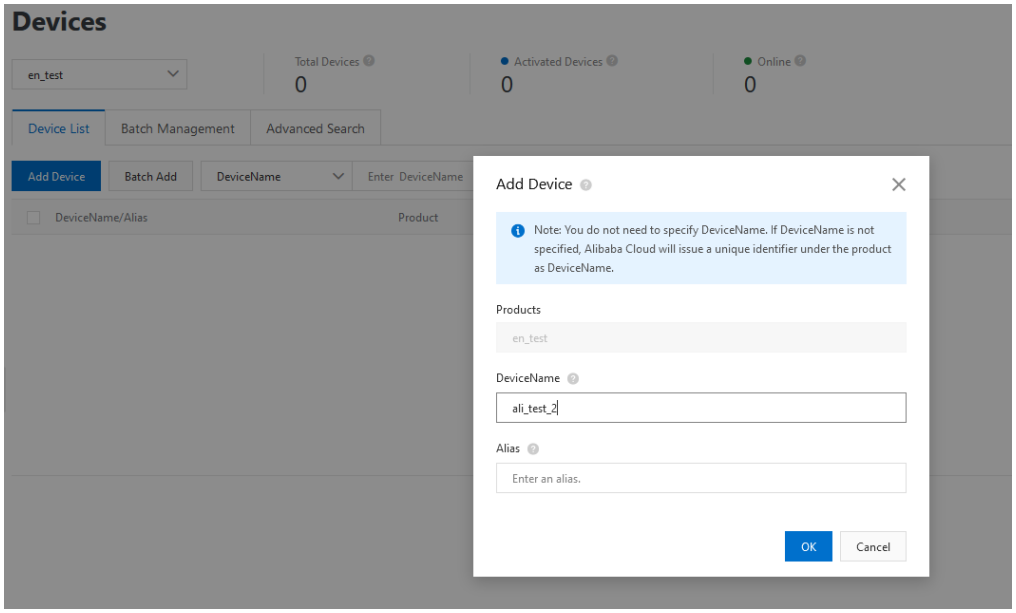


Figure 127. Device information

After the equipment is added, as shown in the figure below:

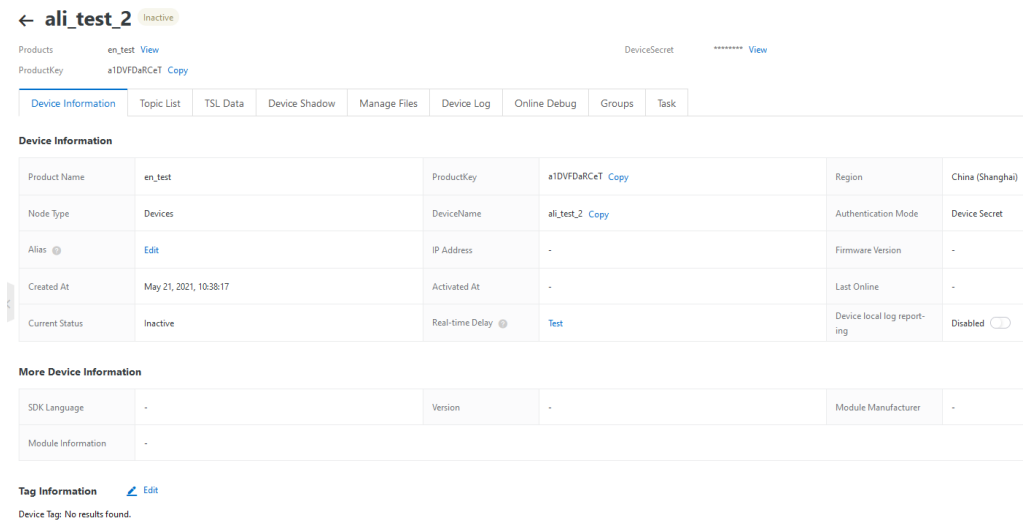


Figure 128. Device details

7.3.2.2 Customize device-side SDK

1. Select “**Documents and Tools**” in the menu bar on the left. To display the interface as shown in the figure below, select “**SDK custom**” in “**Link SDK**”, set the SDK version information as shown in the figure below, then click “**Start Generation**” to generate the device side SDK.

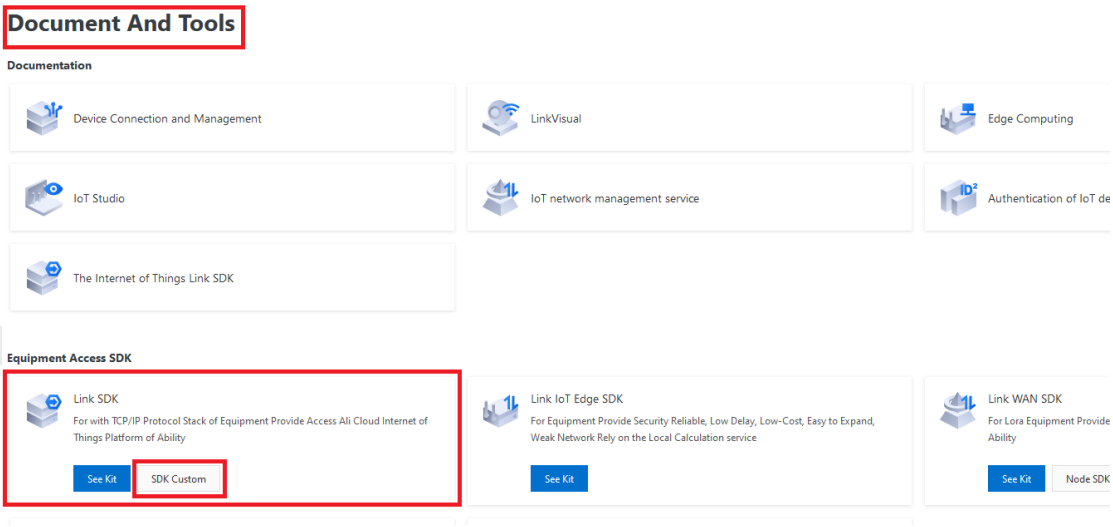


Figure 129. Download SDK

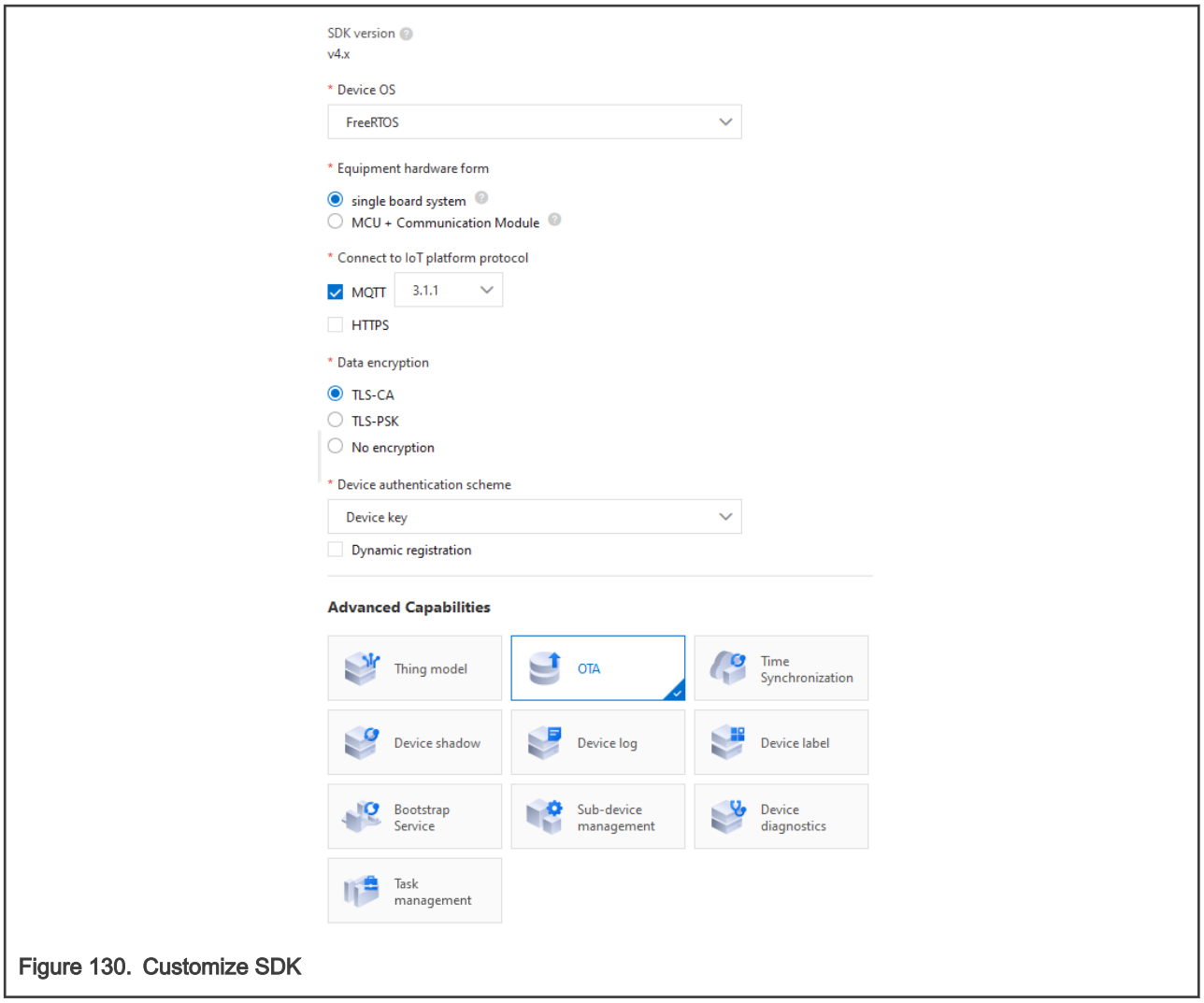


Figure 130. Customize SDK

2. Unzip the downloaded SDK package, replace the `ali_ca_cert.c` in `\LinkSDK\external` with the corresponding file of the tested project. It is consistent with the certificate in the current project, and it does not need to be replaced in this test.

7.3.2.3 Set test equipment information

1. Under the **Device** option of the Alibaba Cloud IOT platform, save the **DeviceName**, **ProductKey**, and **DeviceSecret** information.

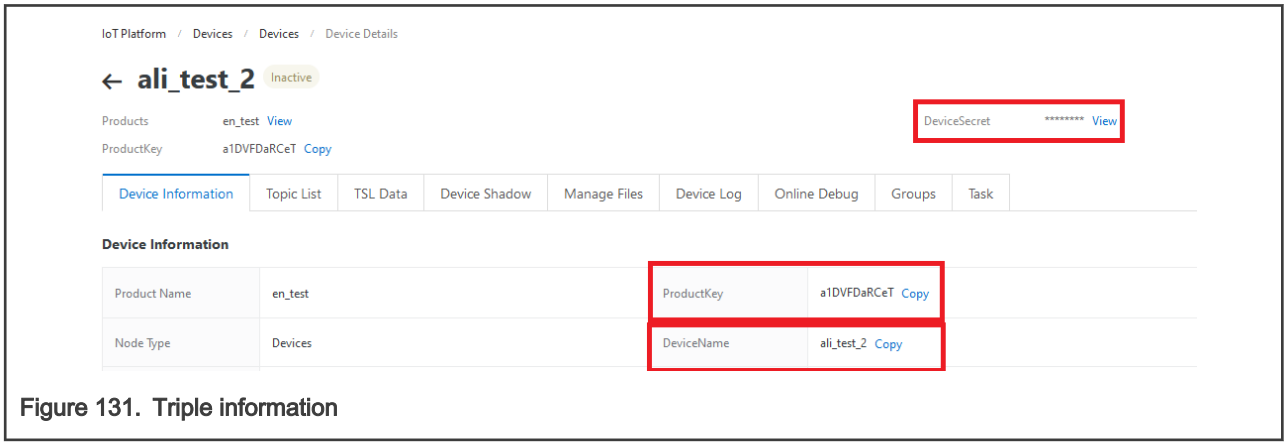


Figure 131. Triple information

2. Open the test project `\ sfw \target \evkbmimxrt10XX`, double-click the script `env.bat`, using the command `scons --menuconfig` to select the Alibaba Cloud project, set the triple information as shown below. Copy the product key/ device name/product secret into it.

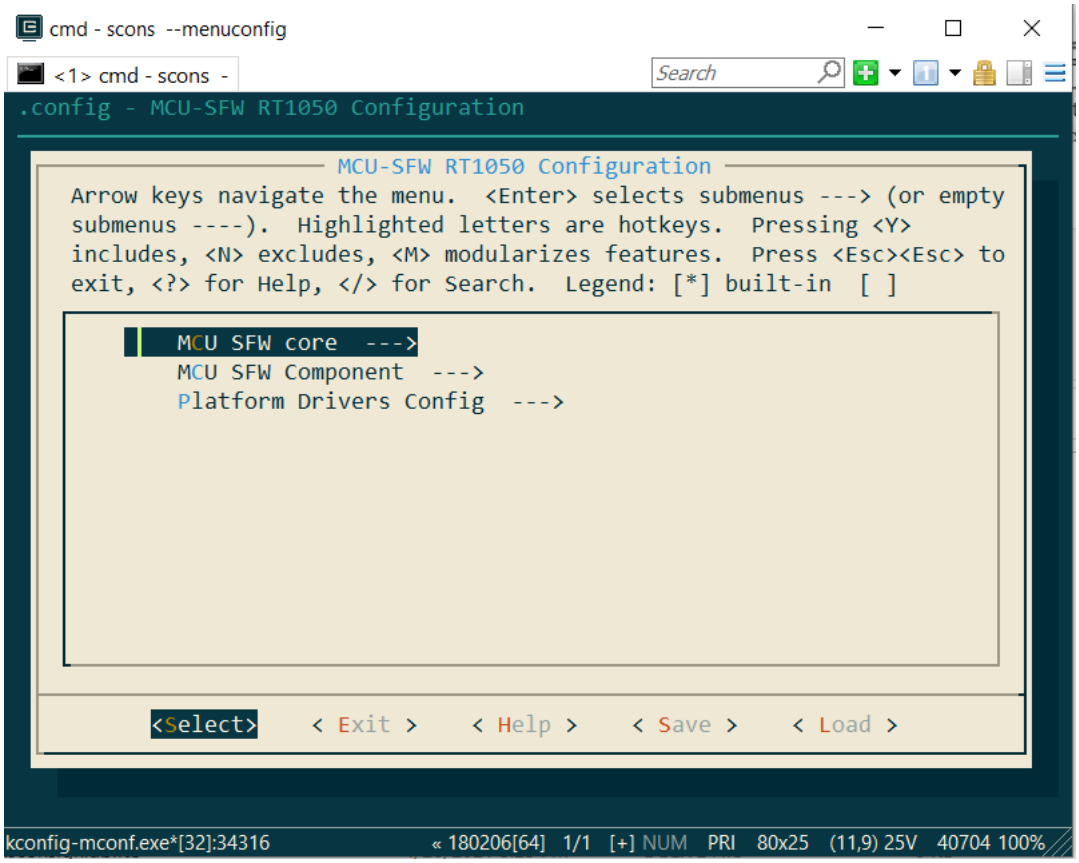


Figure 132. Menuconfig

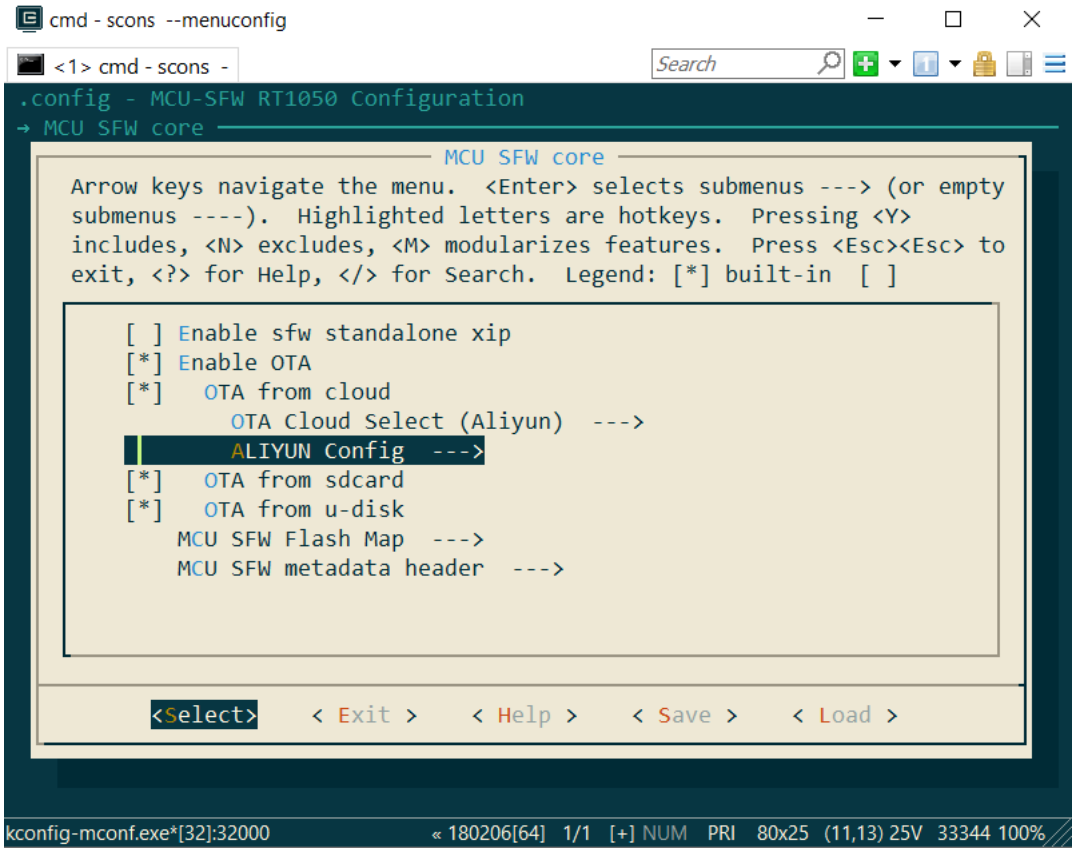


Figure 133. Set Aliyun OTA information

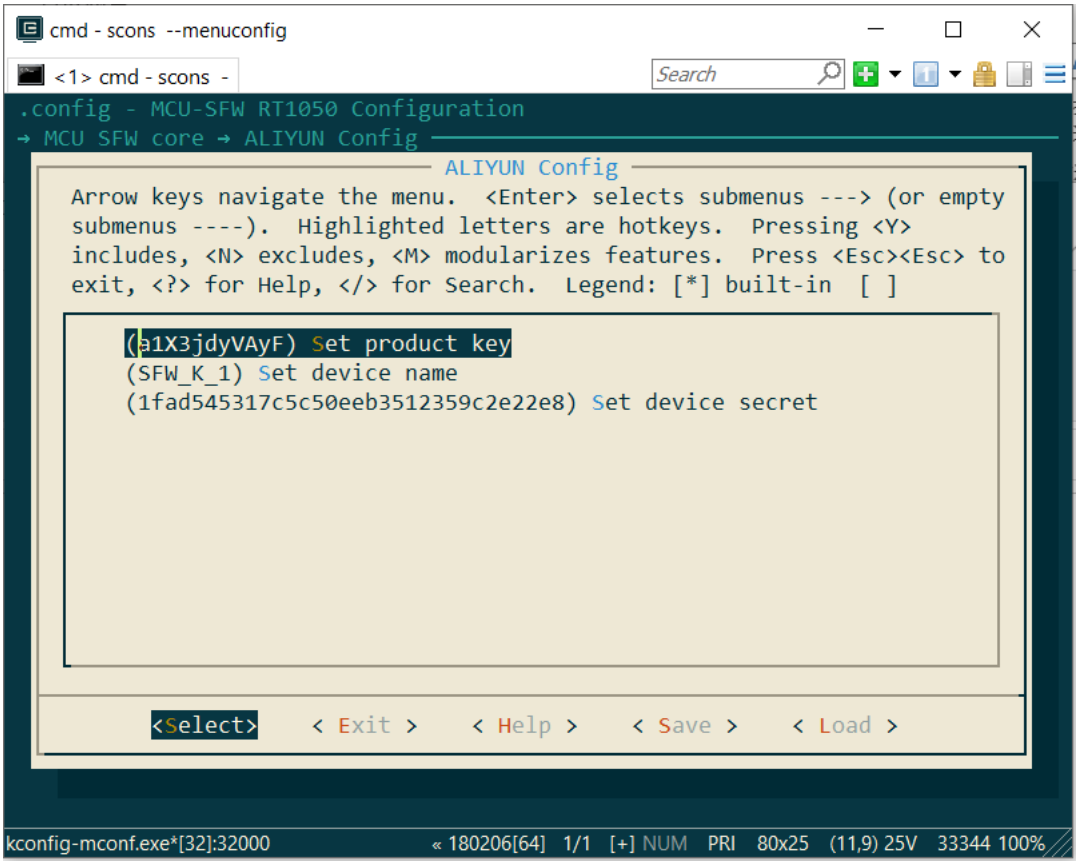


Figure 134. Set triple information

3. In the script, choose `MCU SFW Component -> secure -> enable mbedtls`, and change the mbedtls config file to `ksdk_mbedtls_config.h`

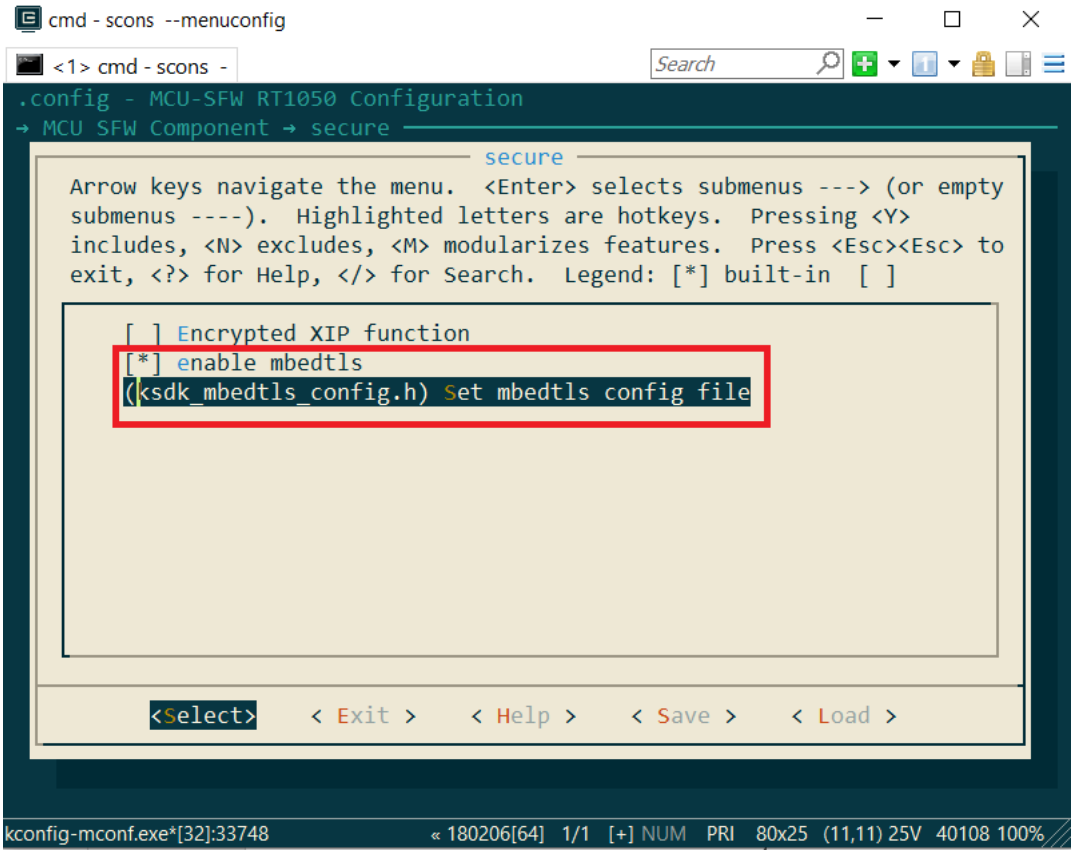


Figure 135. Set mbedtls config file

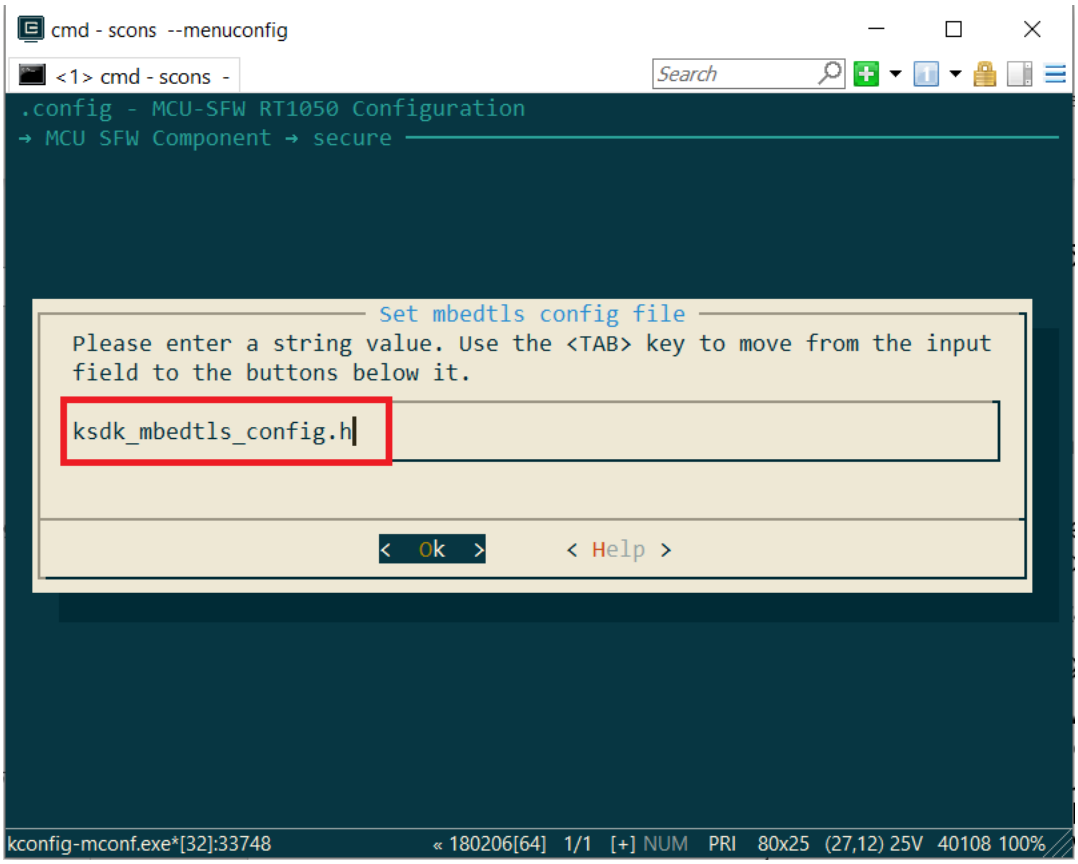


Figure 136. `ksdk_mbedtls_config.h`

Save the setting and use command `scons –ide=iar` to generate the iar project.

7.3.2.4 Modify the `cur_version` for testing

1. Enter `sfw/target/evkmimxrt1064` path, double-click the batch file `env.bat`
2. Input `scons –ide=iar` command to generate the iar project.

> **NOTE**
> Refer to section 2 to generate keil or gcc project.

3. Enter `sfw/target/evkmimxrt1064/iar` path, open sfw.eww project.
4. Go to options, select `generate additional output`, and choose raw binary format.

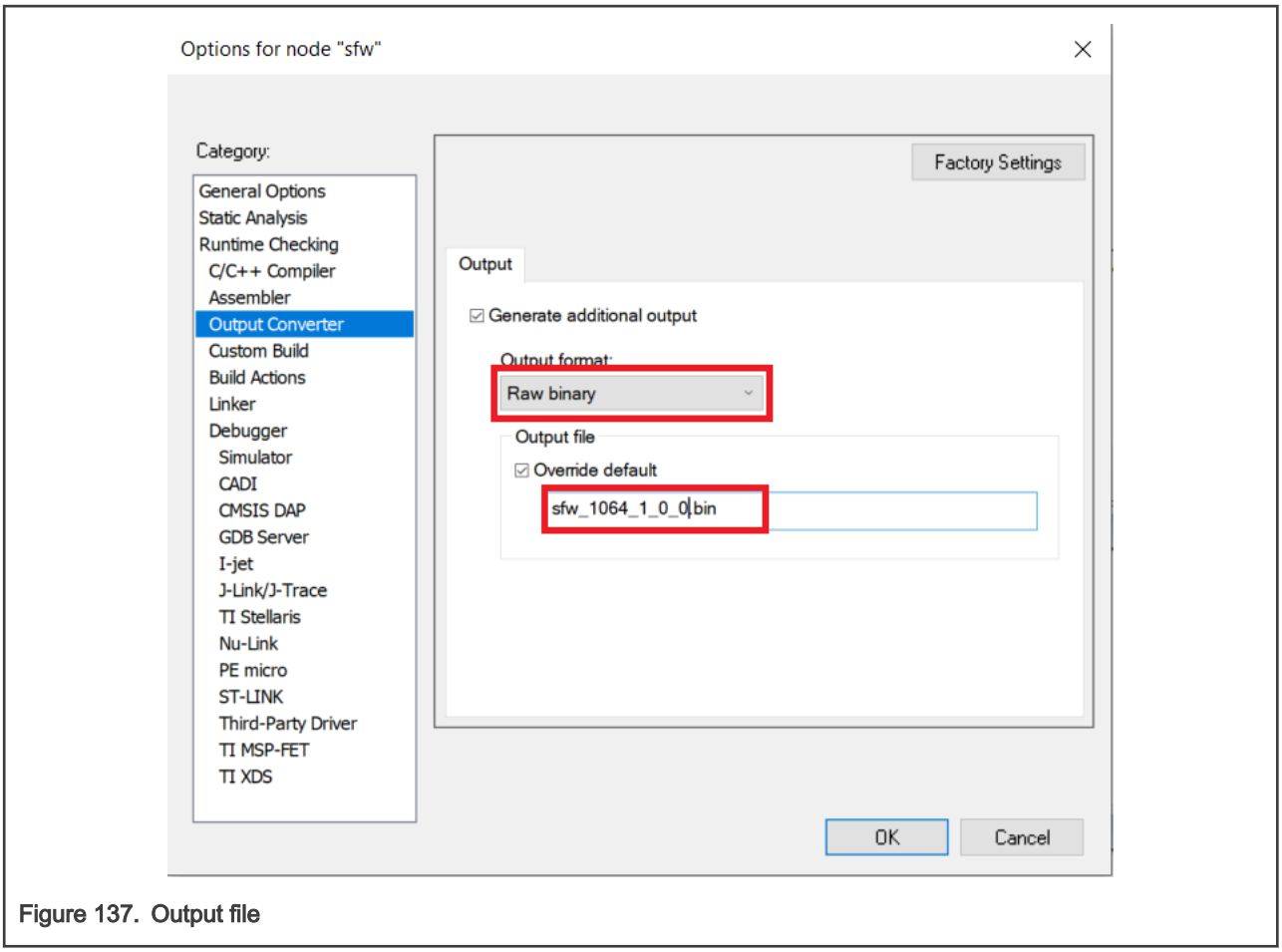


Figure 137. Output file

5. Check the current version in `sfw/firmware/Aliyun_ota/fota_basic_demo.c` file. Set `cur_version` to 1.0.0.

```
342     cur_version = "1.0.0";  //更改为所需要更新的版本，如1.1.0
```

Figure 138. Set the version

6. Change bin file to `sfw_1064_100.bin` and click make button to build the project. The bin file will be generated in `/sfw/target/evkmimxrt1064/iar/build/iar/Exe`.
7. Change the cur_version to “1.4.0”, and change bin file name to `sfw_1064_140.bin`. Generate it.
8. Copy the generated bin files to `sbl/component/secure/mcuboot/scripts` folder.
9. Sign `sfw_1064_100.bin` and `sfw_1064_140.bin` images with RSA using the command below. Then `1064_ali_100.bin` and `1064_ali_140.bin` are generated.

```
python imgtool.py sign --key sign-rsa2048-priv.pem --align 4 --version "1.0.0" --header-size
 0x400 --pad-header --slot-size 0x100000 --max-sectors 32 sfw_1064_100.bin 1064_ali_100.bin
python imgtool.py sign --key sign-rsa2048-priv.pem --align 4 --version "1.4.0" --header-size
 0x400 --pad-header --slot-size 0x100000 --max-sectors 32 sfw_1064_140.bin 1064_ali_140.bin
```

7.3.2.5 Create OTA task

1. Under the "**Maintenance**" directory on the left, select "**OTA Update**".

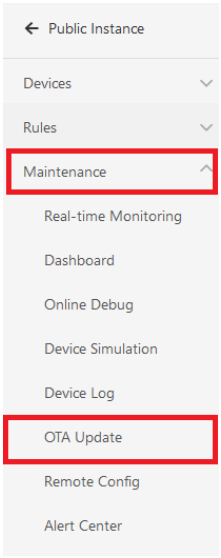


Figure 139. OTA upgrade

2. Click the “**Add Update Package**” button to add the upgrade package, enter the upgrade package name, select the corresponding product module, and “**Update package version**” should correspond to the version of the uploaded bin file, and then click upload to confirm. Now the 1.4.0 version number is used.

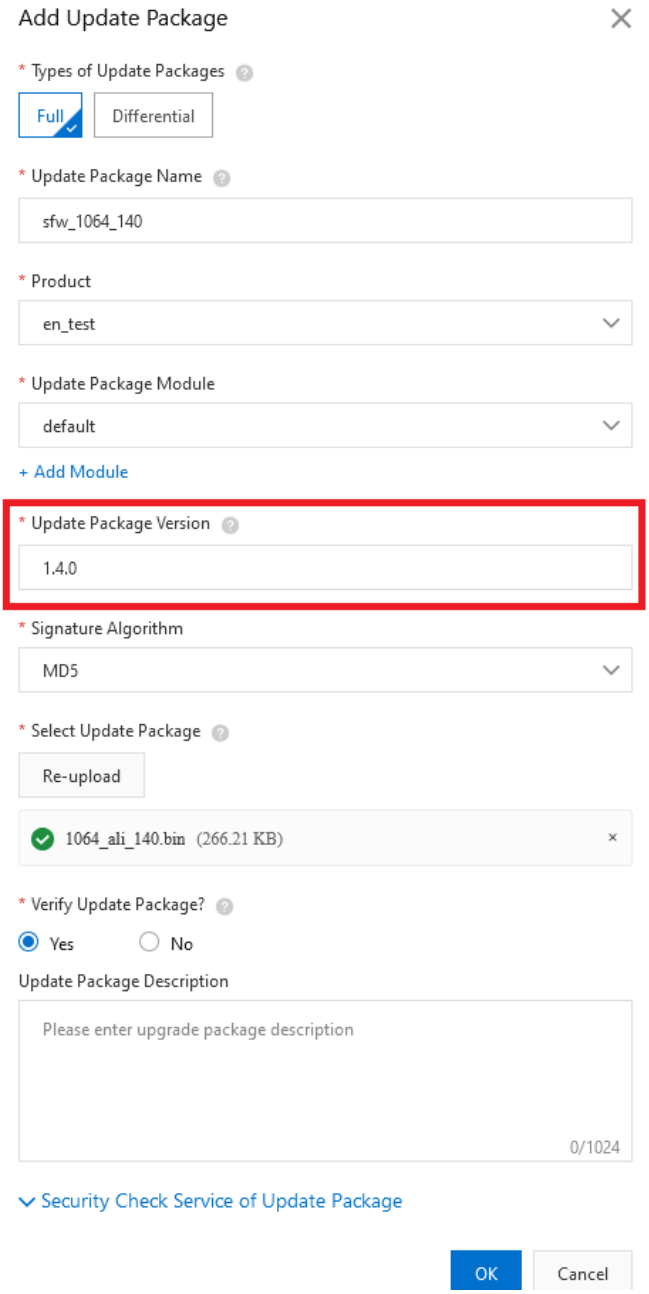


Figure 140. Upgrade package information

7.3.2.6 Run the application

1. Using the `MCUBootUtility` tool to download the `1064_ali_100.bin` generated previously to the first slot of the board. The default location of slot1 is the **flash_offset+0x100000** to **flash_offset+0x200000**, the whole slot size is 1MB.

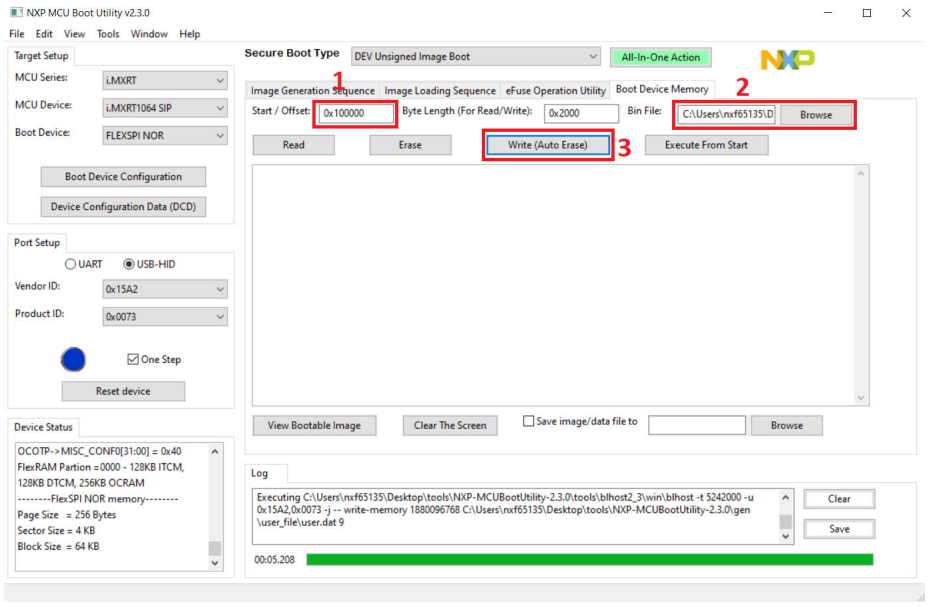


Figure 141. Use image tool to download bin file

2. Prepare the bootloader.

   In SBL project, enter `sbl/target/evkmimxrt1064` path. Disable the `Enable single image function` option and the `Enable mcu isp support` option in the menuconfig interface of Scons to disable single image mode and disable MCU ISP support.

   Compile the SBL project and download it to the target board.

   NOTE

   a. If the new signature key is used, please also modify the `sign-rsa2048-pub.c`

   b. Programming SBL image by drag-drop of DAPLink may erase whole flash.

3. Plug in the ethernet cable. Run the project and the debug console prints the log as shown below.

   The log “The image now in PRIMARY_SLOT slot” and “Getting IP address from DHCP” shows that the image in first slot is booted successfully. The “IPv4 Address:” and “version:1.0.0” shows that the network is connected successfully and Alibaba Cloud get the current device version 1.0.0.

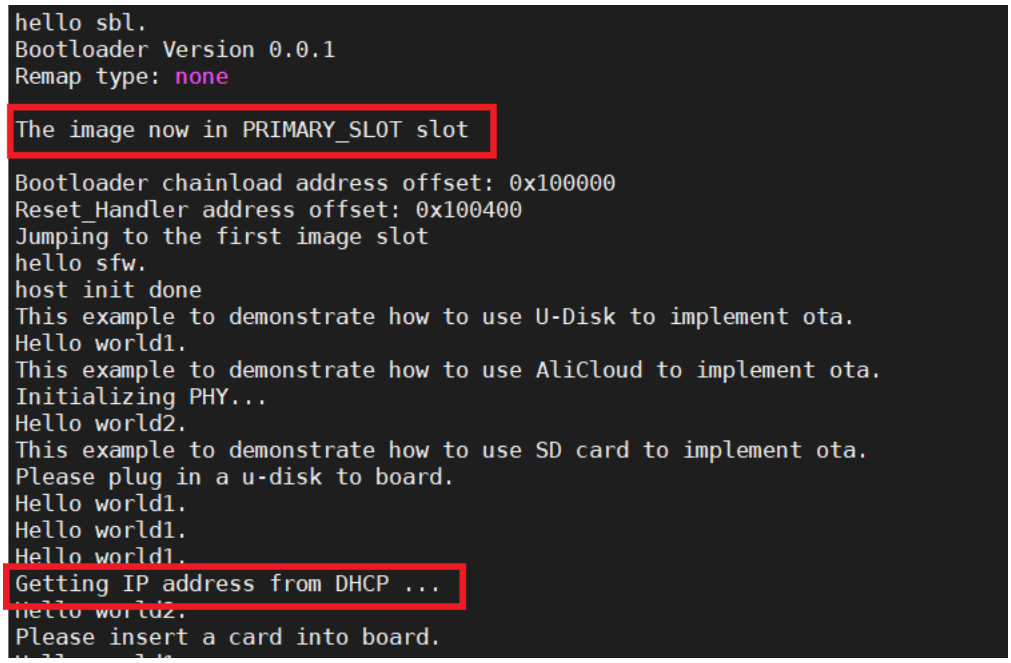


Figure 142. Run sbl project

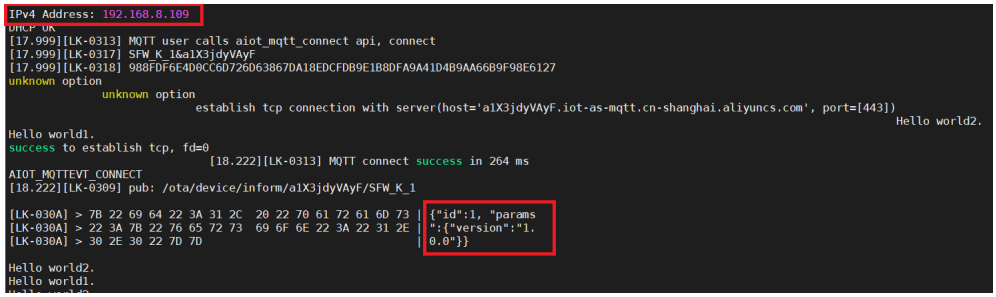


Figure 143. Jump to first image

4. Verify the upgrade package in Web. Click the **Verify** button, fill in the version number that needs to be upgraded, and select the device to be tested. The upgrade timeout period can be omitted. Then click "**OK**".

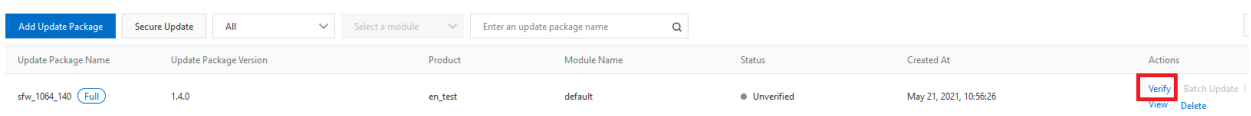


Figure 144. Verify the OTA package

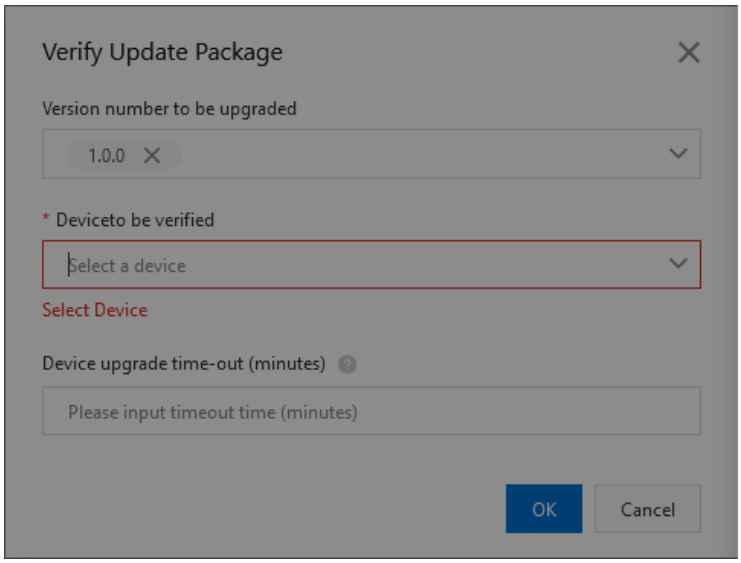


Figure 145. Verify the upgrade package information

5. The debug console shows the OTA progress.

Below is the information about the uploaded package and its version.

```
[21.888][LK-0309] pub: /ota/device/upgrade/a1X3jdyVAyF/SFW_K_1

[LK-030A] < 7B 22 63 6F 64 65 22 3A  22 31 30 30 30 22 2C 22 | {"code":"1000","
[LK-030A] < 64 61 74 61 22 3A 7B 22  73 69 7A 65 22 3A 32 37 | data":{"size":27
[LK-030A] < 32 36 30 30 2C 22 64 69  67 65 73 74 53 69 67 6E | 2600,"digestSign
[LK-030A] < 22 3A 22 45 34 45 4F 72  42 6D 4C 37 77 55 46 6B | ":"E4EOrBmL7wUFk
[LK-030A] < 41 78 39 66 6E 57 67 62  39 35 33 30 67 2F 37 51 | Ax9fnWgb9530g/7Q
[LK-030A] < 64 48 78 4F 6C 51 31 32  6C 4C 63 7A 6E 46 45 4F | dHxOlQ12lLcznFEO
[LK-030A] < 46 4C 62 58 5A 66 42 4F  58 76 76 30 55 77 63 6A | FLbXZfBOXvv0Uwcj
[LK-030A] < 73 7A 65 75 56 6B 66 72  77 38 30 38 62 6B 76 43 | szeuVkfrw808bkvC
[LK-030A] < 38 33 57 71 52 4A 58 6C  2F 6D 67 6A 73 50 36 54 | 83WqRJXl/mgjsP6T
[LK-030A] < 45 33 62 6A 41 51 7A 50  32 7A 2F 2F 45 52 67 36 | E3bjAQzP2z//ERg6
[LK-030A] < 31 56 6B 4B 7A 37 6B 70  75 44 58 48 6D 59 50 66 | 1VkKz7kpuDXHmYPf
[LK-030A] < 35 37 6B 6A 49 32 54 53  42 65 6A 51 77 50 71 6E | 57kjI2TSBejQwPqn
[LK-030A] < 58 2F 2F 44 59 53 56 2B  76 49 78 63 37 6D 6E 32 | X//DYSV+vIxc7mn2
[LK-030A] < 71 57 73 35 6A 43 49 69  6B 37 41 42 72 33 31 58 | qWs5jCIik7ABr31X
[LK-030A] < 7A 5A 34 67 48 36 55 77  53 69 2B 53 50 62 72 56 | zZ4gH6UwSi+SPbrV
[LK-030A] < 6A 32 48 4E 6C 49 31 6C  4F 43 4B 4C 37 59 57 55 | j2HNlI1lOCKL7YWU
[LK-030A] < 61 65 38 76 51 58 61 43  62 56 50 73 72 2F 4F 79 | ae8vQXaCbVPsr/Oy
[LK-030A] < 50 64 68 5A 4B 51 52 62  39 4A 62 78 54 38 72 61 | PdhZKQRb9JbxT8ra
[LK-030A] < 4F 55 43 64 52 37 37 68  57 39 65 45 6D 6B 50 47 | OUCdR77hW9eEmkPG
[LK-030A] < 68 2F 6A 50 57 37 39 45  50 30 6F 6D 53 38 33 4B | h/jPW79EP0omS83K
[LK-030A] < 49 4A 54 71 34 56 6D 45  6B 4E 44 72 5A 49 69 4D | IJTq4VmEkNDrZIiM
[LK-030A] < 71 74 68 4A 45 79 38 54  59 71 72 46 79 59 46 5A | qthJEy8TYqrFyYFZ
[LK-030A] < 64 6E 77 79 63 48 50 4E  53 64 41 4E 37 37 71 4D | dnwycHPNSdAN77qM
[LK-030A] < 6B 66 5A 52 66 4F 4E 49  4E 66 6C 55 39 78 6A 33 | kfZRfONINflU9xj3
[LK-030A] < 68 39 79 6A 36 2B 63 79  77 3D 3D Hello world1.
22 2C 22 73 69 | h9yj6+cyw==","si
[LK-030A] < 67 6E 22 3A 22 35 32 61  35 63 30 32 61 65 37 61 | gn":"52a5c02ae7a
[LK-030A] < 66 63 64 64 63 35 33 62  38 63 66 36 34 62 61 34 | fcddc53b8cf64ba4
[LK-030A] < 36 31 34 34 64 22 2C 22  76 65 72 73 69 6F 6E 22 | 6144d","version"
[LK-030A] < 3A 22 31 2E 34 2E 30 22  2C 22 75 72 6C 22 3A 22 | :"1.4.0","url":"
[LK-030A] < 68 74 74 70 73 3A 2F 2F  69 6F 74 78 2D 6F 74 61 | https://iotx-ota
[LK-030A] < 2E 6F 73 73 2D 63 6E 2D  73 68 61 6E 67 68 61 69 | .oss-cn-shanghai
[LK-030A] < 2E 61 6C 69 79 75 6E 63  73 2E 63 6F 6D 2F 6F 74 | .aliyuncs.com/ot
[LK-030A] < 61 2F 66 66 35 64 39 30  33 32 33 34 37 39 33 31 | a/ff5d9032347931
[LK-030A] < 39 37 65 61 37 63 31 62  66 64 61 36 37 31 38 64 | 97ea7c1bfda6718d
[LK-030A] < 30 39 2F 63 6B 6F 6D 61  33 6D 35 71 30 30 30 30 | 09/ckoma3m5q0000
[LK-030A] < 33 61 38 65 34 65 35 30  77 38 38 6B 2E 62 69 6E | 3a8e4e50w88k.bin
```

Figure 146. Get the package information

```
[LK-030A] < 61 74 75 72 65 3D 59 64  59 69 6C 75 6F 77 4E 58 | ature=YdYiluowNX
[LK-030A] < 25 32 42 70 6C 61 77 52  59 66 64 4C 4B 4A 70 47 | %2BplawRYfdLKJpG
[LK-030A] < 74 5A 59 25 33 44 22 2C  22 73 69 67 6E 4D 65 74 | tZY%3D","signMet
[LK-030A] < 68 6F 64 22 3A 22 4D 64  35 22 2C 22 6D 64 35 22 | hod":"Md5","md5"
[LK-030A] < 3A 22 35 32 61 35 63 30  32 61 65 37 61 66 63 64 | :"52a5c02ae7afcd
[LK-030A] < 64 63 35 33 62 38 63 66  36 34 62 61 34 36 31 34 | dc53b8cf64ba4614
[LK-030A] < 34 64 22 7D 2C 22 69 64  22 3A 31 36 32 30 38 37 | 4d"},"id":162087
[LK-030A] < 35 37 33 35 37 30 38 2C  22 6D 65 73 73 61 67 65 | 5735708,"message
[LK-030A] < 22 3A 22 73 75 63 63 65  73 73 22 7D            | ":"success"}

OTA target firmware version: 1.4.0, size: 272600 Bytes
unknown option
            establish tcp connection with server(host='iotx-ota.oss-cn-shanghai.aliyuncs.com', port=[80])
                                                                                                    Hel
success to establish tcp, fd=1
                              [22.222][LK-040B] > GET /ota/ff5d903234793197ea7c1bfda6718d09/ckoma3m5q00003a8e4
LTAI4G1TuWwSirnbAzUHfL3e&Signature
[22.333][LK-040B] > Host: iotx-ota.oss-cn-shanghai.aliyuncs.com
[22.333][LK-040B] > Accept: text/html, application/xhtml+xml, application/xml;q=0.9, */*;q=0.8
[22.333][LK-040B] > Range: bytes=0-
[22.333][LK-040B] > Content-Length: 0
[22.333][LK-040B] >
[22.333][LK-0309] pub: /ota/device/progress/a1X3jdyVAyF/SFW_K_1

[LK-030A] > 7B 22 69 64 22 3A 32 2C  20 22 70 61 72 61 6D 73 | {"id":2, "params
[LK-030A] > 22 3A 7B 22 73 74 65 70  22 3A 22 30 22 2C 22 64 | ":{"step":"0","d
[LK-030A] > 65 73 63 22 3A 22 22 7D  7D                     | esc":""}}
```

Figure 147. Target firmware version

Below is shown that the download request was successfully sent and the download process begins.

```
download renewal request has been sent successfully
[22.888][LK-040D] < HTTP/1.1 206 Partial Content
[22.888][LK-040D] < Server: AliyunOSS
[22.888][LK-040D] < Date: Thu, 13 May 2021 03:15:46 GMT
[22.999][LK-040D] < Content-Type: application/octet-stream
[22.999][LK-040D] < Content-Length: 272600
[22.999][LK-040D] < Connection: keep-alive
[22.999][LK-040D] < x-oss-request-id: 609C99E21B27393636E1E89F
[22.999][LK-040D] < Content-Range: bytes 0-272599/272600
[22.999][LK-040D] < Accept-Ranges: bytes
[22.999][LK-040D] < ETag: "52A5C02AE7AFCDDC53B8CF64BA46144D"
[22.999][LK-040D] < Last-Modified: Thu, 13 May 2021 02:34:39 GMT
[22.999][LK-040D] < x-oss-object-type: Normal
[22.999][LK-040D] < x-oss-hash-crc64ecma: 5693646425570967251
[22.999][LK-040D] < x-oss-storage-class: Standard
[22.999][LK-040D] < Content-MD5: UqXAKuevzdxTuM9kukYUTQ==
[22.999][LK-040D] < x-oss-server-time: 13
[22.999][LK-040D] <
Hello world1.
Hello world2.
download   5% done, +2048 bytes
[23.666][LK-0309] pub: /ota/device/progress/a1X3jdyVAyF/SFW_K_1

[LK-030A] > 7B 22 69 64 22 3A 34 2C  20 22 70 61 72 61 6D 73 | {"id":4, "params
[LK-030A] > 22 3A 7B 22 73 74 65 70  22 3A 22 35 22 2C 22 64 | ":{"step":"5","d
[LK-030A] > 65 73 63 22 3A 22 22 7D  7D                     | esc":""}}

Hello world1.
Hello world2.
download  10% done, +2048 bytes
[24.444][LK-0309] pub: /ota/device/progress/a1X3jdyVAyF/SFW_K_1

[LK-030A] > 7B 22 69 64 22 3A 35 2C  20 22 70 61 72 61 6D 73 | {"id":5, "params
[LK-030A] > 22 3A 7B 22 73 74 65 70  22 3A 22 31 30 22 2C 22 | ":{"step":"10","
[LK-030A] > 64 65 73 63 22 3A 22 22  7D 7D                  | desc":""}}
```

Figure 148. Download request

Below is shown the download progress finished and the system reset action started.

```
[39.666][LK-0901] digest matched
download 100% done, +216 bytes
[39.666][LK-0309] pub: /ota/device/progress/a1X3jdyVAyF/SFW_K_1

[LK-030A] > 7B 22 69 64 22 3A 32 33  2C 20 22 70 61 72 61 6D | {"id":23, "param
[LK-030A] > 73 22 3A 7B 22 73 74 65  70 22 3A 22 31 30 30 22 | s":{"step":"100"
[LK-030A] > 2C 22 64 65 73 63 22 3A  22 22 7D 7D             | ,"desc":""}}

write update type = 0x4
write magic number offset = 0xffff0
Down finished all.SystemReset Now...hello sbl.
Bootloader Version 0.0.1
Remap type: test

The image now in SECONDARY_SLOT slot
```

Figure 149. Download all

Below is shown the current package version is 1.4.0. In Alibaba Cloud web, the OTA information shows the verification is done successfully.

```
success to establish tcp, fd=0
                              [16.666][LK-0313] MQTT connect success in 354 ms
AIOT_MQTTEVT_CONNECT
Update done, the last update type: ALI platform
Write OK flag: off = 0xfffe0
[16.666][LK-0309] pub: /ota/device/inform/a1X3jdyVAyF/SFW_K_1

[LK-030A] > 7B 22 69 64 22 3A 31 2C  20 22 70 61 72 61 6D 73 | {"id":1, "params
[LK-030A] > 22 3A 7B 22 76 65 72 73  69 6F 6E 22 3A 22 31 2E | ":{"version":"1.
[LK-030A] > 34 2E 30 22 7D 7D                               | 4.0"}}
```

Figure 150. Run the new image

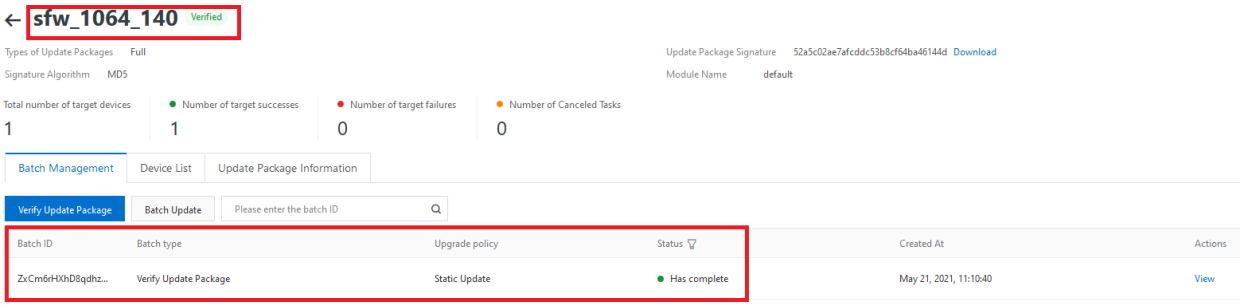


Figure 151. Alibaba Cloud IOT platform shows the OTA success

7.4 Secure FOTA

This section walks through the steps of how to perform the Secure OTA firmware.

7.4.1 Secure boot demonstration for the platform EVKMIMXRTxxxx

This section describes the steps how to enable secure boot and generate signed image. The demo targeted for the EVKMIMXRT1060 hardware platform is used as an example, although these steps can be applied to other i.MX RT platforms.

7.4.1.1 Generating Keys and Certificates

To enable ROM secure boot, generate keys and certificates. Do the following steps to generate them.

1. Retrieve and install the MCUXpresso Secure Provisioning tool
2. Run this tool, click the button to switch the processor, select MIMXRT1060. To select a processor from a different family, create a new workspace.

NOTE

: Open MCUXpresso Secure Provisioning tool with administrator mode. Otherwise some important material will not be generated.

NOTE

: For EVKMIMXRT1010 platform, select MIMXRT1015 to generate keys.

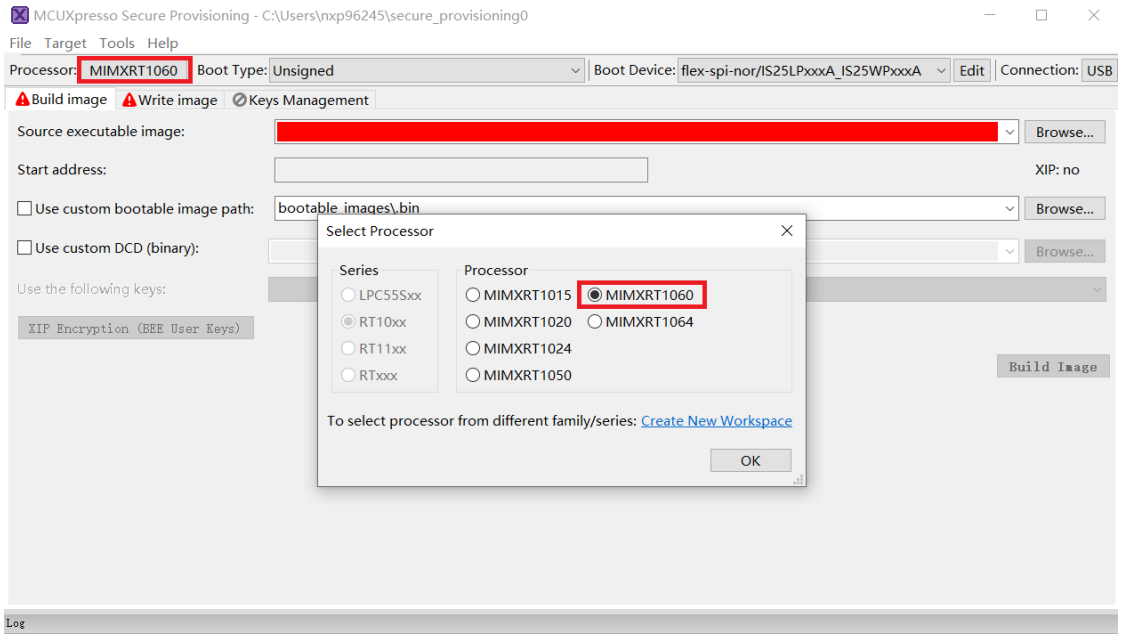


Figure 152. Creating a new workspace

3. Choose Boot Type as Authenticated (HAB).

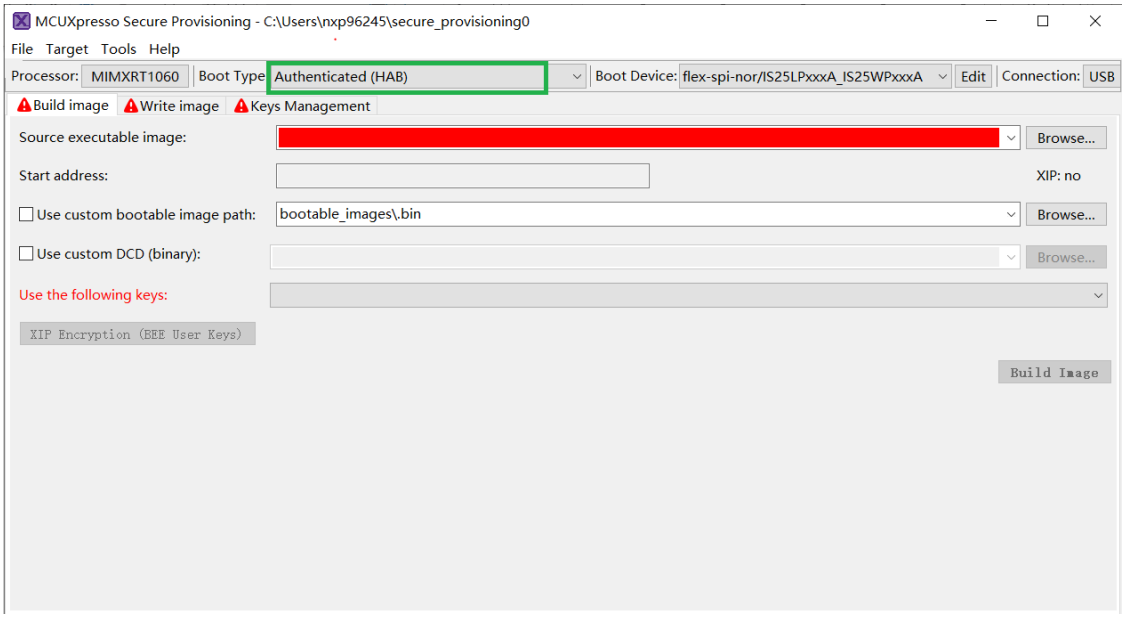


Figure 153. Selecting Boot Type

4. In the **Keys Management** view, click **Generated keys**, then specify all parameters in this menu.

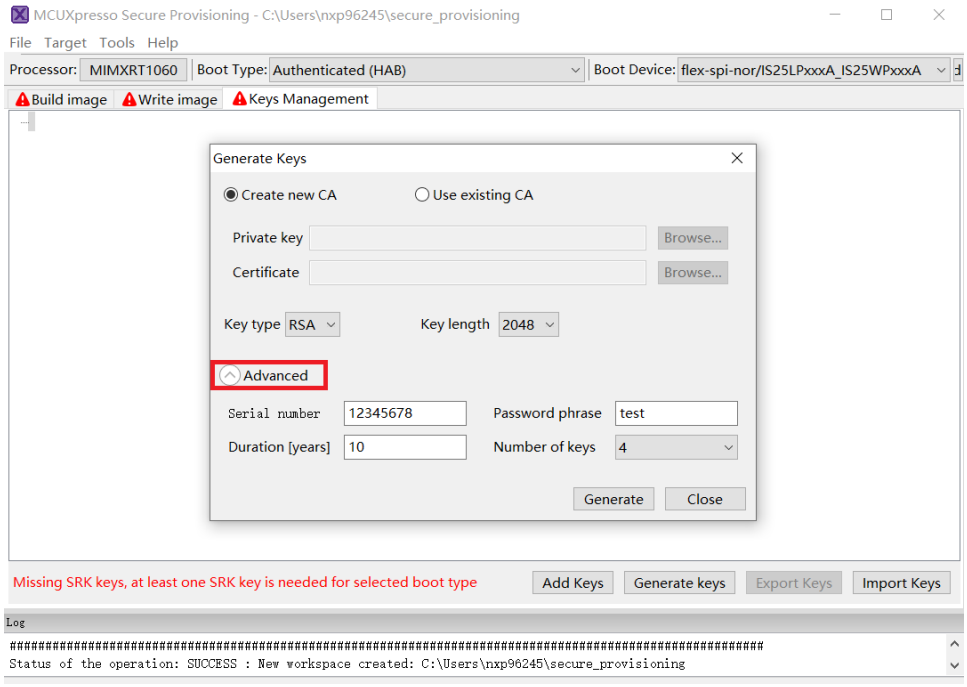


Figure 154. Generate keys (EVKMIMXRTRT10xx, EVKMIMXRTRT11xx)

5. Click **Generate**, OpenSSL output is displayed in the progress window.

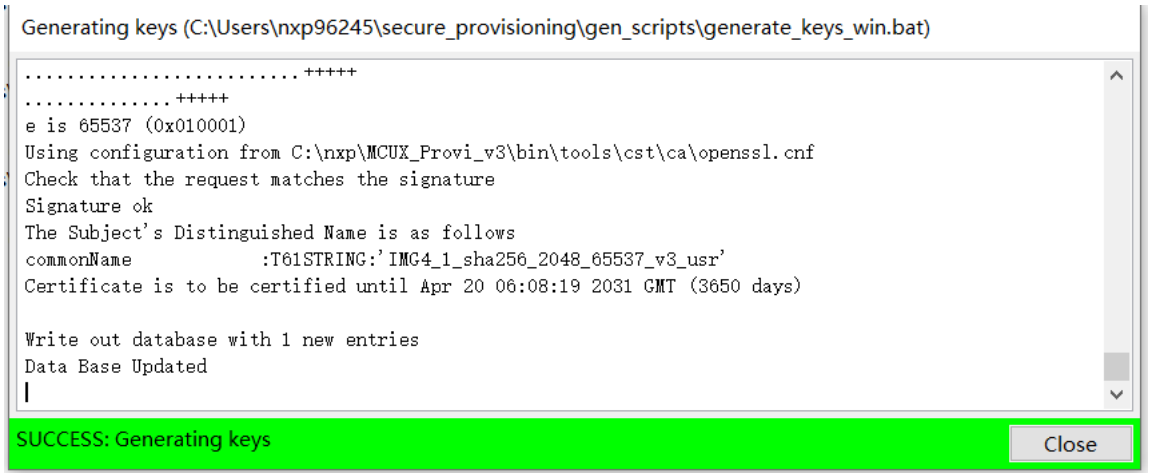


Figure 155. Generate Keys - Success

6. You can find generated keys and certificates in the folder “keys” and “crts” in the workspace directory, Copy folder “keys”, “crts” and “gen_hab_certs” to the folder `sbl/target/evkmimxrt1060/secure`.

7.4.1.2 SBL image preparation

To generate a signed bootable SBL image, the steps are as below:

1. Run `env.bat` in target board
2. Run `scons --menuconfig` to enter menu `MCU SBL Core` to select `Enable ROM to verify sbl`

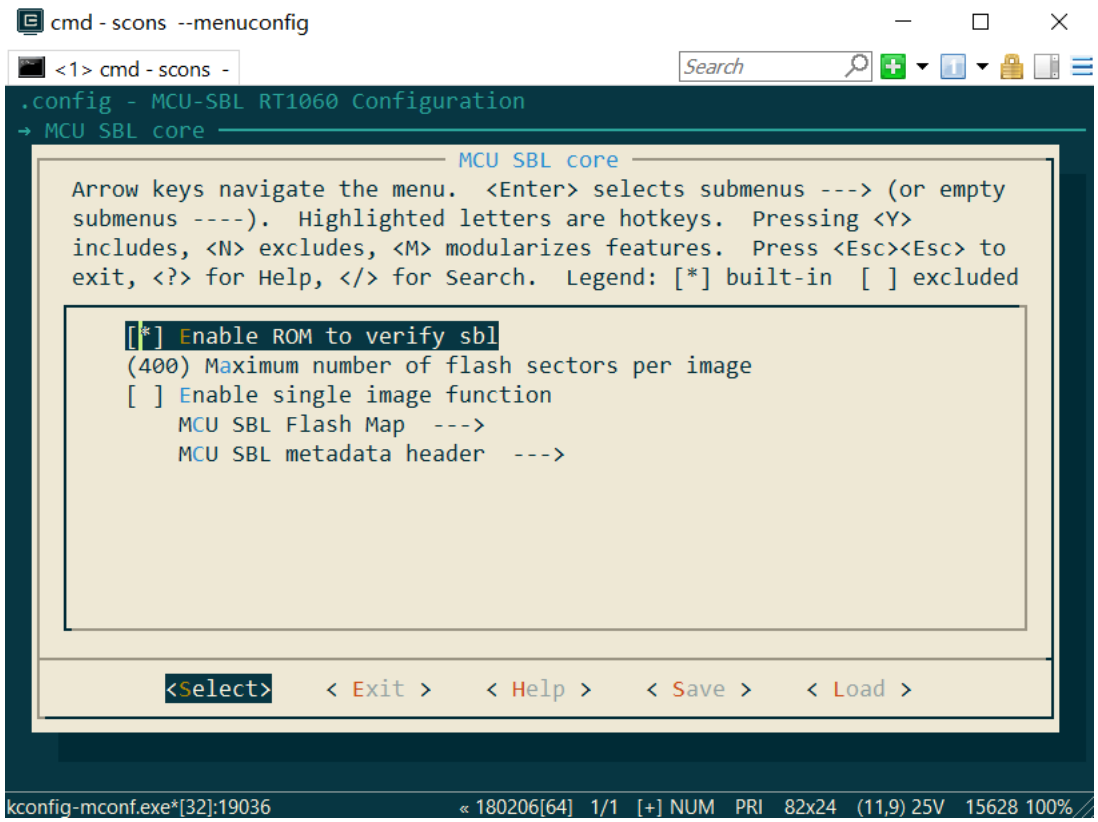


Figure 156. Enable ROM to verify sbl

3. Enter menu `MCU SBL Component > secure > selected signing method`, select one application signature type, save, and quit menuconfig.

Figure 157. Select signing method

4. Run `scons --ide=iar` command to generate IAR project or run `scons --ide=mdk5` to generate Keil project.
5. Configure the option to generate an image with `.srec format` in IAR project, then build the project. For Keil project, user may generate a srec format image by running:

```
fromelf.exe --m32combined --output "$L@L.srec" "#L"
```

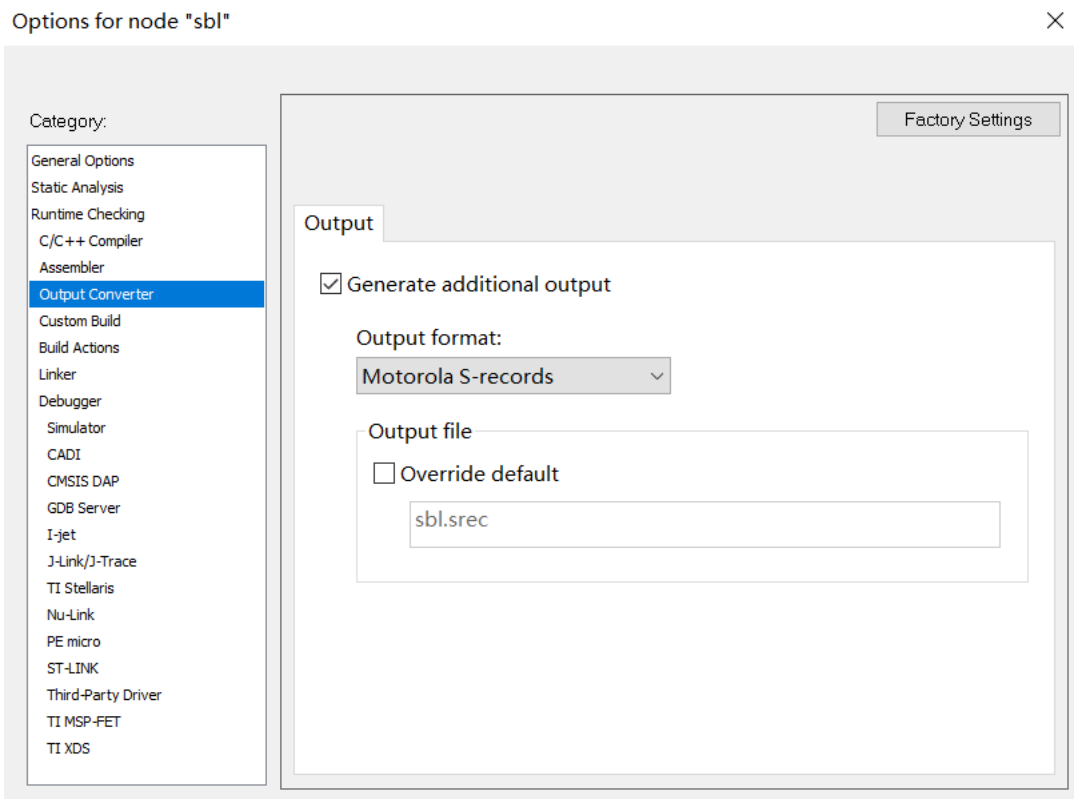


Figure 158. Update defined Symbols

7.4.1.3 Application image preparation

To generate the application image, the steps are as below:

1. Open linker file `sfw\target\xxxx\board\link\MIMXRTXXXX_flexspi_nor.icf` (take IAR linker file as example), and make the following changes in the linker file:

```
define symbol m_interrupts_start       = BOOT_FLASH_ACT_APP + 0x2000;
define symbol m_interrupts_end         = BOOT_FLASH_ACT_APP + 0x2000 + 0x3FF;

define symbol m_text_start             = BOOT_FLASH_ACT_APP + 0x2000 + 0x400;
define symbol m_text_end               = BOOT_FLASH_CAND_APP - 0x1000;
```

2. Run `env.bat` in one target board in SFW
3. Enter menu `MCU SFW core` and uncheck menu `Enable sfw standalone xip`, save and quit menuconfig

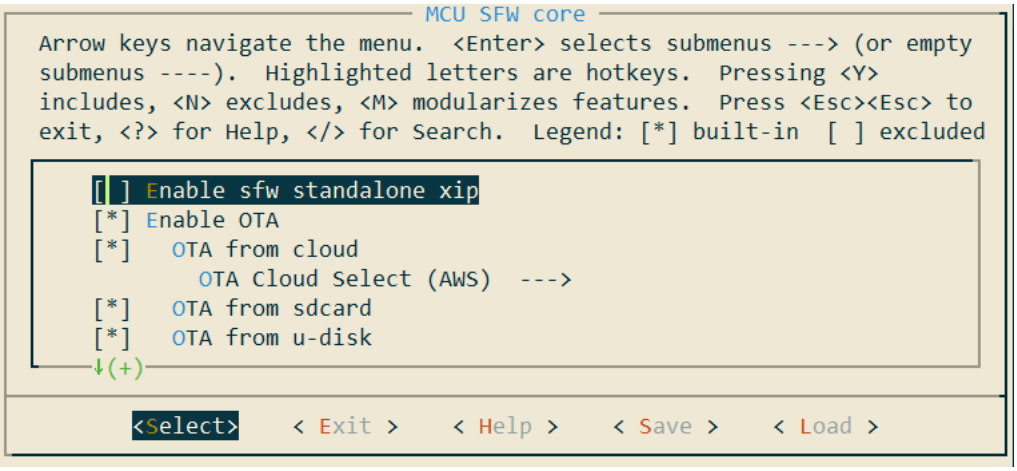


Figure 159. Uncheck enable sfw standalone

4. Run `scons --ide=iar` to generate IAR project or run `scons --ide=mdk5` to generate the Keil project for SFW.
5. Configure the option to generate an image with `.srec` format, then build the project.

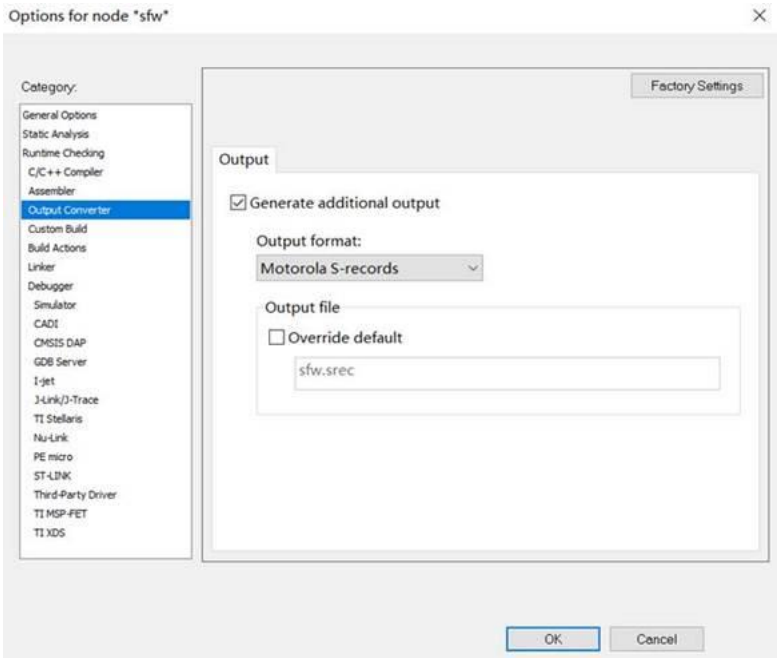


Figure 160. Output binary format

NOTE

For the other signing method, generate a binary format application.

7.4.1.4 Program OCOTP (eFuse)

Below is an example of how to program SRK table and enable HAB closed mode.

NOTE

In the Development phase, the device may be under HAB open mode for most use cases.

1. Find and open the script `program_ocotp.bat` in `sbl/target/evkmimxrt1060/secure`, then set the correct installation path for tools elftosb, cst, blhost and so on:

```
SET "PATH=C:\nxp\MCUX_Provi_v3\bin\tools\elftosb\win;%PATH%"
SET "PATH=C:\nxp\MCUX_Provi_v3\bin\tools\sdphost\win;%PATH%"
SET "PATH=C:\nxp\MCUX_Provi_v3\bin\tools\blhost\win;%PATH%"
SET "PATH=C:\nxp\MCUX_Provi_v3\bin\tools\cst\mingw32\bin;%PATH%"
```

2. Open the file `SRK_fuses.bin` in hex mode under the folder `gen_hab_certs`.

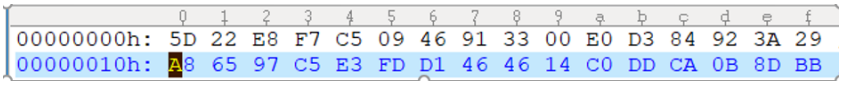


Figure 161. SRK hash value in hex

3. The value located in the efuse file is intended to be burned to the `SRK_HASH` efuse field on the SoC. This hash value must be burned to the SoC efuses in the following order (the first word to the first fuse row index):

   f7e8225d, 914609c5, d3e00033, 293a9284, c59765a8, 46d1fde3, ddc01446, bb8d0bca. Note the data endianness.

   If the user program efuses with the script `program_ocotp.bat`, remember to update the SRK hash value according to user's `SRK_fuses.bin` in script. The commands used to program efuses are commented out by default. Enable it by hand. For other platform, eFuse OCOTP index of SRK hash table may be different. Refer to the fuse map for the NXP processor used.

```
blhost.exe -u 0x15A2,0x0073 -j -- efuse-program-once 0x18 f7e8225d
blhost.exe -u 0x15A2,0x0073 -j -- efuse-program-once 0x19 914609c5
blhost.exe -u 0x15A2,0x0073 -j -- efuse-program-once 0x1a d3e00033
blhost.exe -u 0x15A2,0x0073 -j -- efuse-program-once 0x1b 293a9284
blhost.exe -u 0x15A2,0x0073 -j -- efuse-program-once 0x1c c59765a8
blhost.exe -u 0x15A2,0x0073 -j -- efuse-program-once 0x1d 46d1fde3
blhost.exe -u 0x15A2,0x0073 -j -- efuse-program-once 0x1e ddc01446
blhost.exe -u 0x15A2,0x0073 -j -- efuse-program-once 0x1f bb8d0bca
```

4. Enable HAB close mode using the following command. In Production phase, enable HAB closed mode and sign the SBL image.

   ```
   blhost.exe -u 0x15A2,0x0073 -j -- efuse-program-once 0x06 00000002
   ```

5. Verify the eFuse value via command efuse-read-once.

   ```
   blhost.exe -u 0x15A2,0x0073 -j -- efuse-read-once 0x18
   ```

6. Put switch SW7-4 on the EVKMIMXRT1060 board to enter Serial Downloader mode, then run script `program_ocotp.bat`.

7.4.1.5 Run SBL and application

To generate signed SBL and application, do the following steps. Then program them to board.

1. Copy `sbl.srec` and `sfw.srec` to the folder `sbl/target/evkmimxrt1060/secure`. If you select RSA or ECDSA signing type, use `sfw.bin`.
2. To enter Serial Downloader mode and connect USB OTG and DEBUG USB port, make the EVKMIMXRT1060 board.
3. Enter folder secure in scons environment by inputting `cd secure`.
4. Run `sign_sbl_app.bat` to generate the final signed SBL and application. If you want to generate the application image for the next update, remember to change the parameter version number in the `imgtool.py` command line.
5. Download the signed SBL and application by the script `sign_sbl_app.bat` or other tools. This script downloads the application image into slot1. If you tested the OTA procedure before, SBL may still try to run application in slot2.

6. Switch (SW7-3 on, SW7-4 off) to normal boot mode, then reset board. You see output in the terminal.

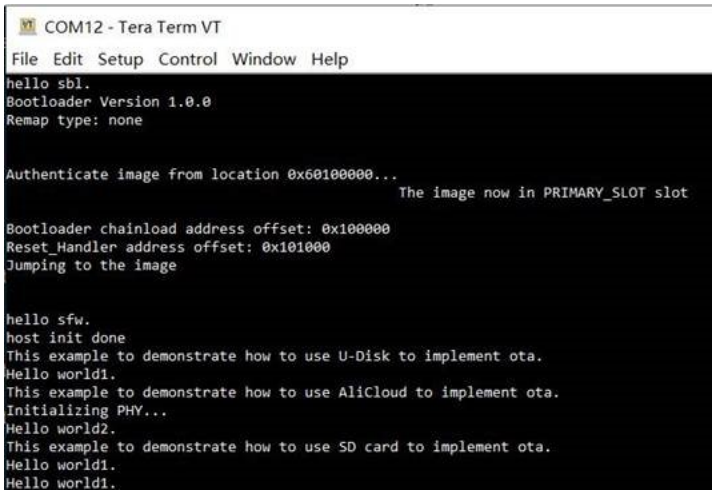


Figure 162. Terminal log

7.4.2 Encrypted XIP boot demonstration for the platform EVKMXRTxxxx

This section describes the steps to enable XIP encrypted authenticated boot. The demo targeted for EVKMIMXRT1060 hardware platform is used as an example.

7.4.2.1 SBL image preparation

To generate signed bootable SBL image, the steps are as below:

1. Run `env.bat` in the target board.
2. Run `scons --menuconfig` to enter menu "MCU SBL Core" to select "Enable ROM to verify sbl".

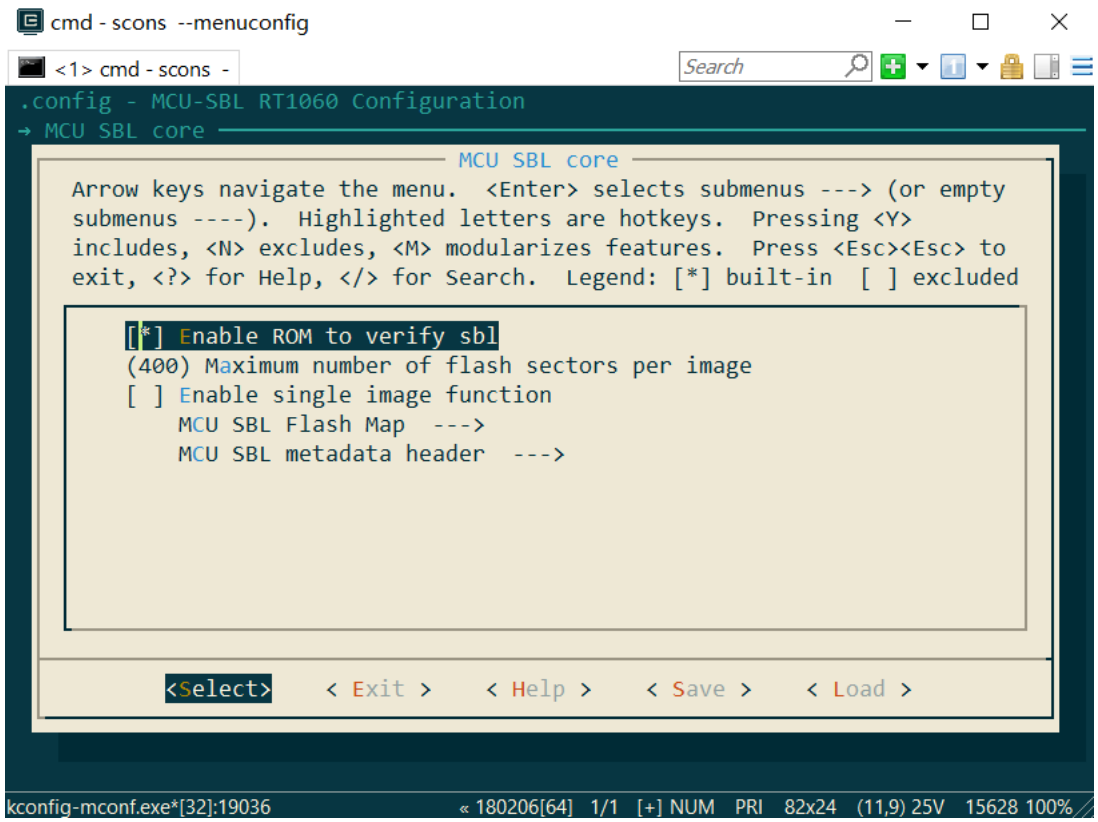


Figure 163. Enable ROM to verify sbl

3. Enter menu `MCU SBL Component > secure`, select Encrypted XIP function, save and quit menuconfig.

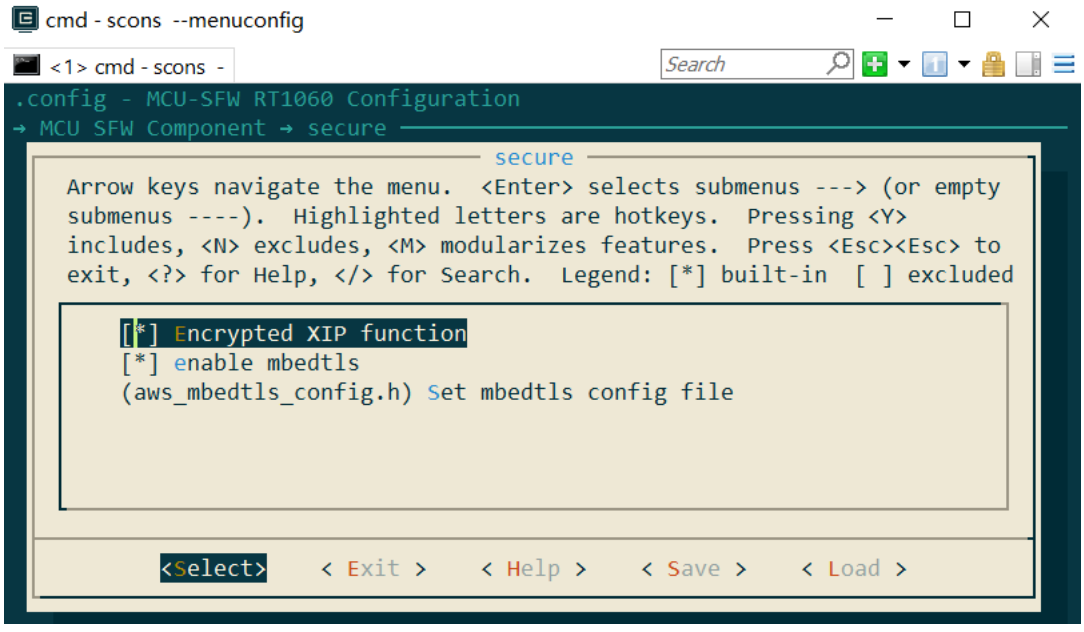


Figure 164. Enable key context upgrade in SBL

4. Run the `scons --ide=iar` command to generate the IAR project.

5. Configure the project to generate an image with the `.srec` format, then build the project. For the Keil project, generate the srec format image by running `fromelf.exe --m32combined --output "$L@L.srec"`.

7.4.2.2 Application image preparation

To generate application image for encryption, the steps are as below:

1. Open the linker file `sfw\target\xxxx\board\link\MIMXRTXXXX_flexspi_nor.icf` (take the IAR linker file as example), and make the following changes in the linker file:

```
define symbol m_interrupts_start       = BOOT_FLASH_ACT_APP + 0x2000;
define symbol m_interrupts_end         = BOOT_FLASH_ACT_APP + 0x2000 + 0x3FF;

define symbol m_text_start             = BOOT_FLASH_ACT_APP + 0x2000 + 0x400;
define symbol m_text_end               = BOOT_FLASH_CAND_APP - 0x1000;
```

2. Run `env.bat` in target platform
3. Enter the menu `MCU SFW core` and uncheck the menu `Enable sfw standalone xip`
4. Enter the menu `MCU SFW component > secure` and check the menu `Encrypted XIP function`, then save and quit menuconfig.

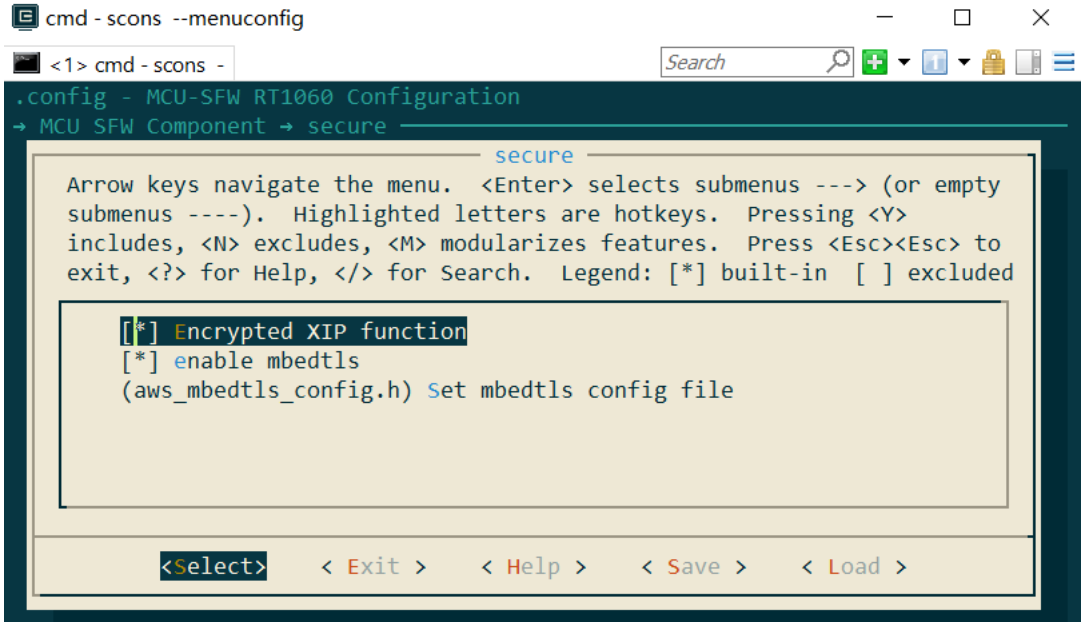


Figure 165. Enable key context upgrade in SFW

5. Generate the SFW project.
6. Change codes to call the function `update_key_context()` after calling enable_image() if application supports OTA.
7. Configure option to generate an image with thesrec format, then build the project.

> **NOTE**
>
> For RSA or ECDSA signing method, don't change linker file in step1 and need to generate binary format image.

7.4.2.3 Program KEK (eFuse)

The KEK (Key of Encryption Key) serves as the key for the BEE to unwrap the Key context.

You may not write KEK and keep that efuse value as all 0 s in test phase. Below is an example how to burn KEK to efuse SW_GP2 for EVKMIMXRT1060, add the command below into the file `program_ocotp.bat`.

```
 :: kek=00112233445566778899aabbccddeeff
blhost.exe -u 0x15A2,0x0073 -j -- efuse-program-once 0x29 ccddeeff
blhost.exe -u 0x15A2,0x0073 -j -- efuse-program-once 0x2a 8899aabb
blhost.exe -u 0x15A2,0x0073 -j -- efuse-program-once 0x2b 44556677
blhost.exe -u 0x15A2,0x0073 -j -- efuse-program-once 0x2c 00112233
```

The KEK (Key of Encryption Key) serves as the key for the OTFAD which uses EVKMIMXRT1010 or EVKMIMXRT1170. Follow the order shown below: `0xffeeddcc, 0xbbaa9988, 0x77665544, 0x33221100`.

7.4.2.4 Run SBL and application

Do the following steps to generate signed and encrypted SBL and application together.

1. Copy `sbl.srec` and `sfw.srec` to the folder `sbl/target/evkmimxrt1060/secure`
2. Edit `sign_enc_sbl_app.bat` and set the KEK and the encrypted region.
3. Make the EVKMIMXRT1060 board to enter Serial Downloader mode and connect USB OTG and the DEBUG USB port.
4. Run `sign_enc_sbl_app.bat` to generate final signed and encrypted SBL and application1.
5. Download SBL and application by this script or other tools.
6. Switch (SW7-3 ON, SW7-4 OFF) to normal boot mode, set SW5-1 ON then reset board. You see the output in the terminal.

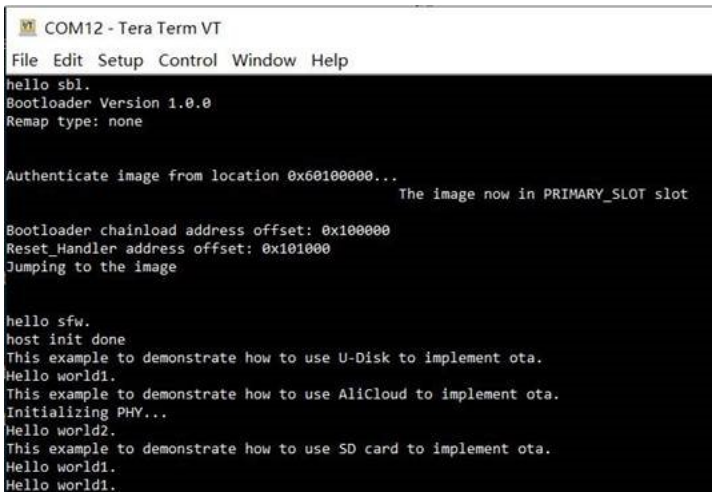


Figure 166. Terminal log

7.4.2.5 Application OTA image preparation

To generate the signed and encrypted application for upgrading, do the following steps:

1. Generate a project as in section 7.4.2.2
2. Build the image and rename it to `sfw2.srec` or `sfw2.bin`
3. Open script `sign_enc_sfw2.bat`, set the KEK and encrypted region.
4. Run `sign_enc_sfw2.bat`, file `sfw_2_enc.bin` is the final image.

7.4.3 Secure boot demonstration for the platform LPC55S69

This section describes the steps to enable an XIP encrypted authenticated boot. The demo targeted for LPC55S69(revision 1B) hardware platform is used as an example.

7.4.3.1 Generating Keys and Certificates

To enable ROM secure boot, generate keys and certificates. Follow the steps to generate them.

1. Install the MCUXpresso Secure Provisioning tool
2. Run this tool, click `File > New Workspace`, then select processor LPC55S69 to create a new workspace.

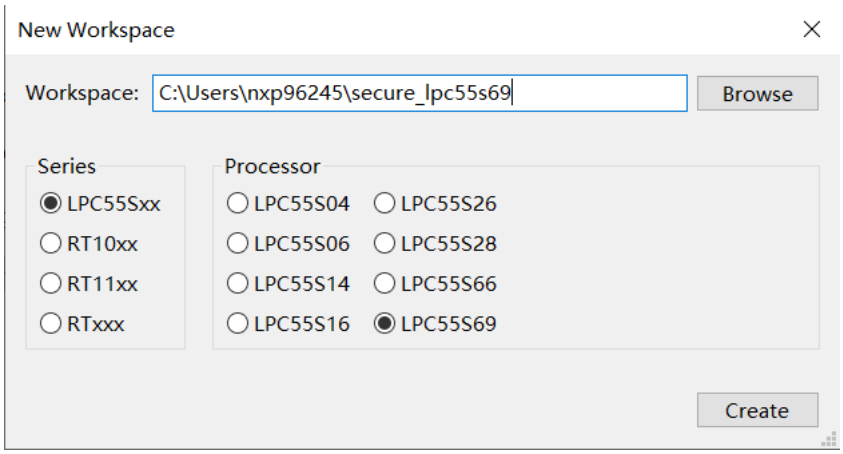


Figure 167. Creating a new workspace

3. Choose **Boot Type** as Encrypted (PRINCE) and Signed.

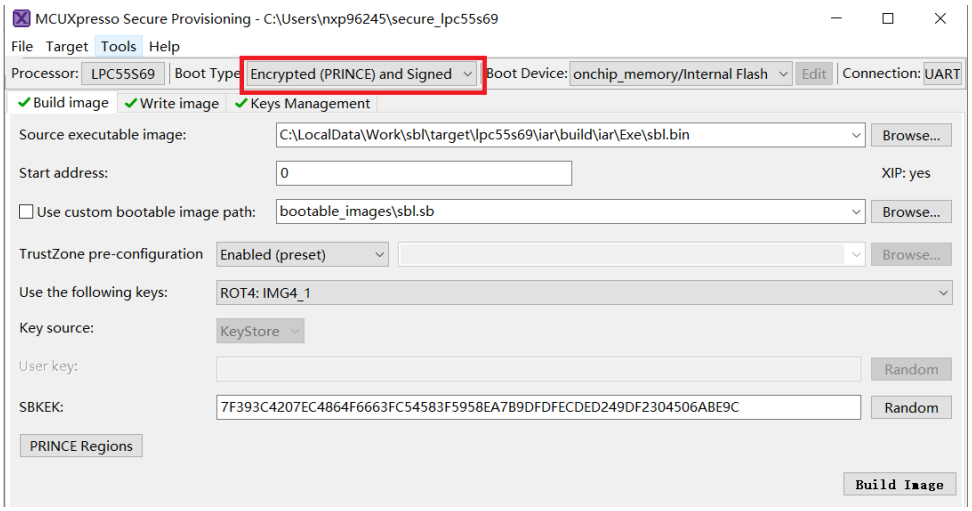


Figure 168. Selecting Boot Type

4. In the **Keys Management** view, click the button **Generated keys**, then specify all parameters in this menu.

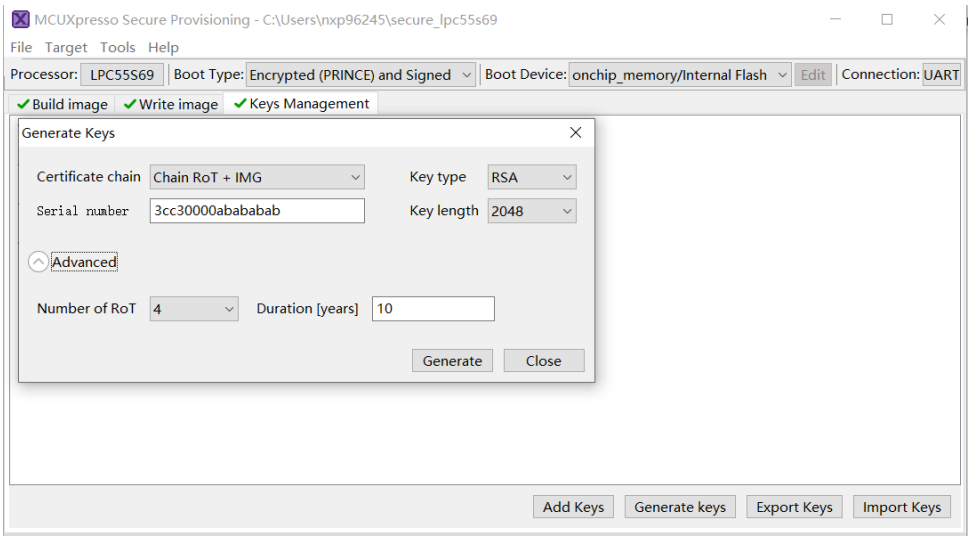


Figure 169. Figure 7-141: Generate keys

5. Click the button **Generate**, OpenSSL output is displayed in the progress window.

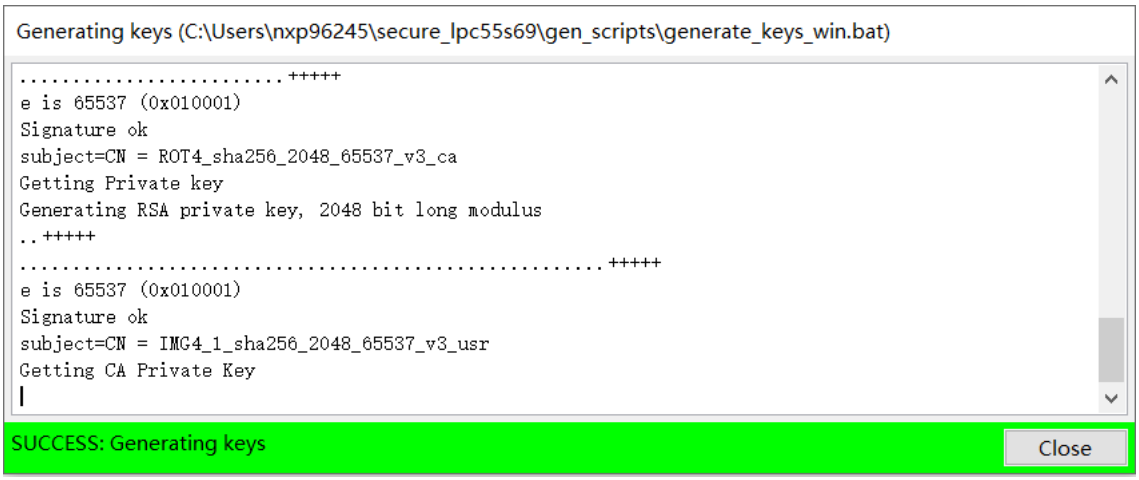


Figure 170. Generating keys - success

6. You can find generated keys and certificates in the folder `keys and crts` in the workspace directory.

7.4.3.2 SBL image preparation

The following steps describe the procedure of creating an SBL image.

1. Run `env.bat` in target board.
2. Enter the menu `MCU SFW Core >`, select one application signature type, save and quit menuconfig.

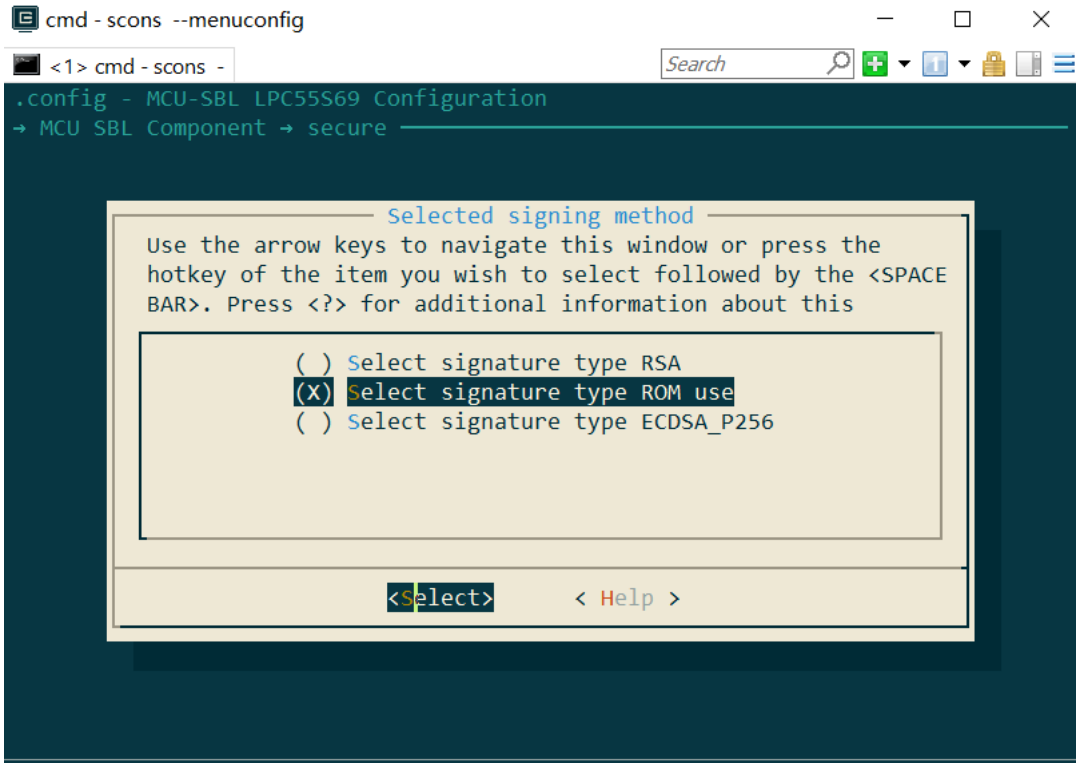


Figure 171. Selecting signing method

3. Run the `scons --ide=iar` command to generate the IAR project.
4. Build binary a image from IDE, it is plain image for unsecure boot.

7.4.3.3 Application image preparation

The following steps describe the procedure for creating signed image.

1. Run `env.bat` in target board under SFW repo
2. Enter the menu `MCU SFW core` and uncheck the menu `Enable sfw standalone xip`, save and quit menuconfig

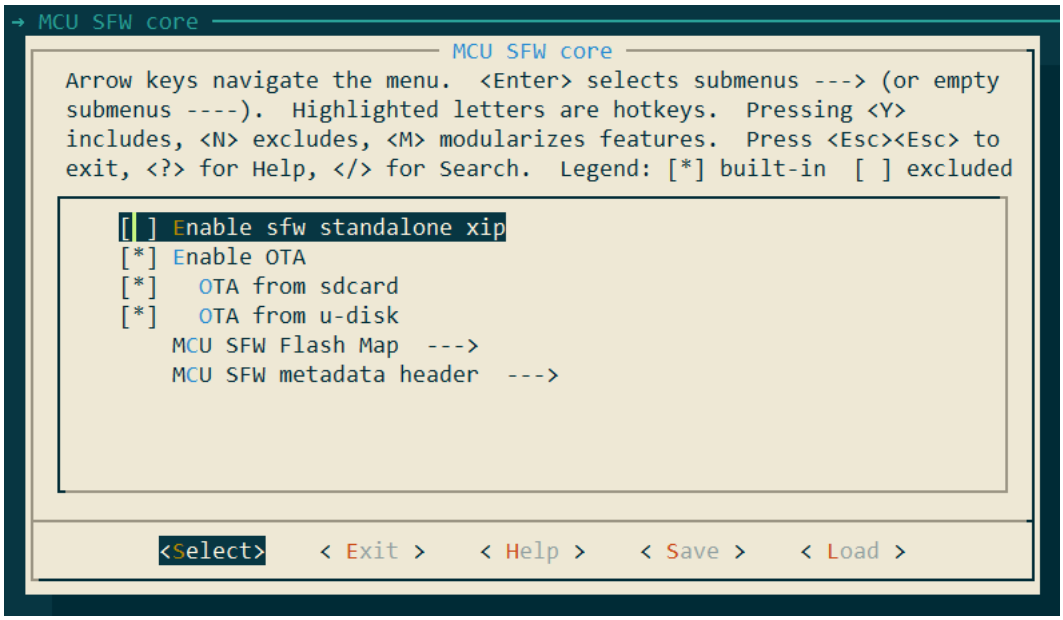


Figure 172. Uncheck Enable sfw standalone

3. Generate an IAR project, following the example below:

   ```
   scons --ide=iar
   ```

4. In `Project > Options > Output Converter`, check Generate additional output and select `Raw binary output format`
5. Build the project.

7.4.3.4 Sign and Program encrypted SBL

This section describes the building and writing of authenticated SBL image by tool MCUXpresso Secure Provisioning Tool (SPT).

1. In the **Build Image** view, select the image position as a Source executable image.
2. In Start address tab, input 0.
3. Select **Enabled(present)** in TrustZone pre-configuration.
4. For use the following keys, select any key chain, for example `ROT1: IMG1_1`.
5. Input SBKEK or generate one with the **Random** button.

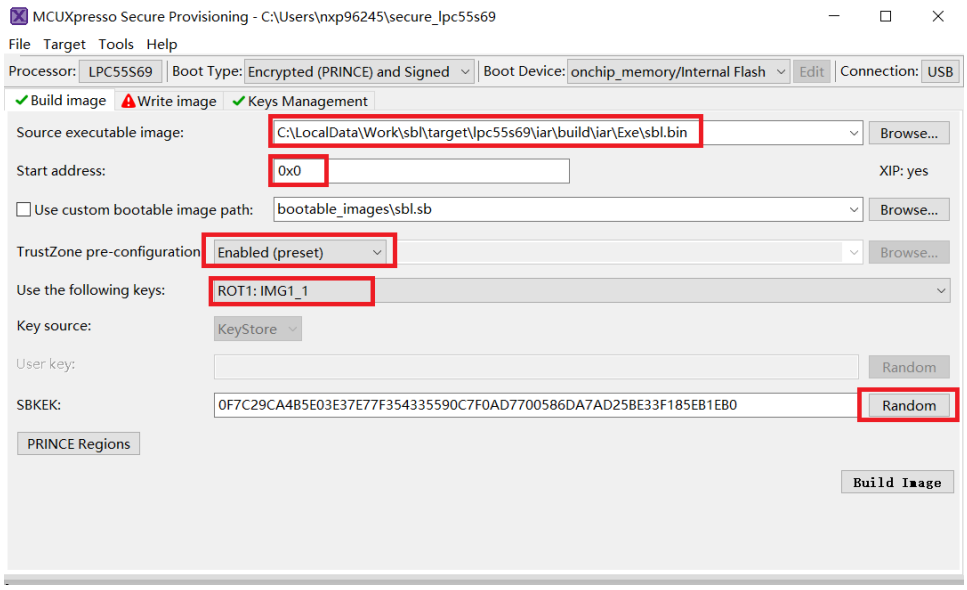


Figure 173. Setting parameters for building image

6. Open the PRINCE configuration and check the configuration. Set the encrypted size of the SBL image in the PRINCE region. If you do not encrypt the image, skip this step. Select Signed in Boot Type.

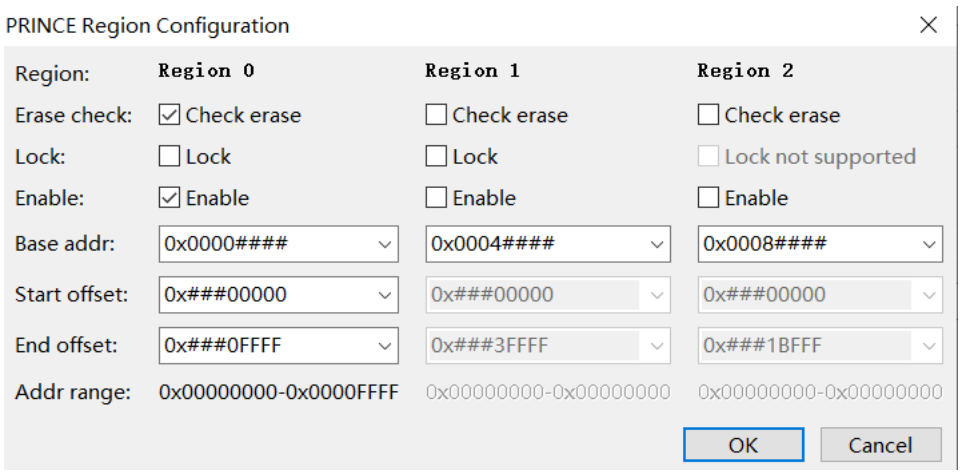


Figure 174. Setting encryption region

7. Click **Build image**, output is displayed in the progress window.

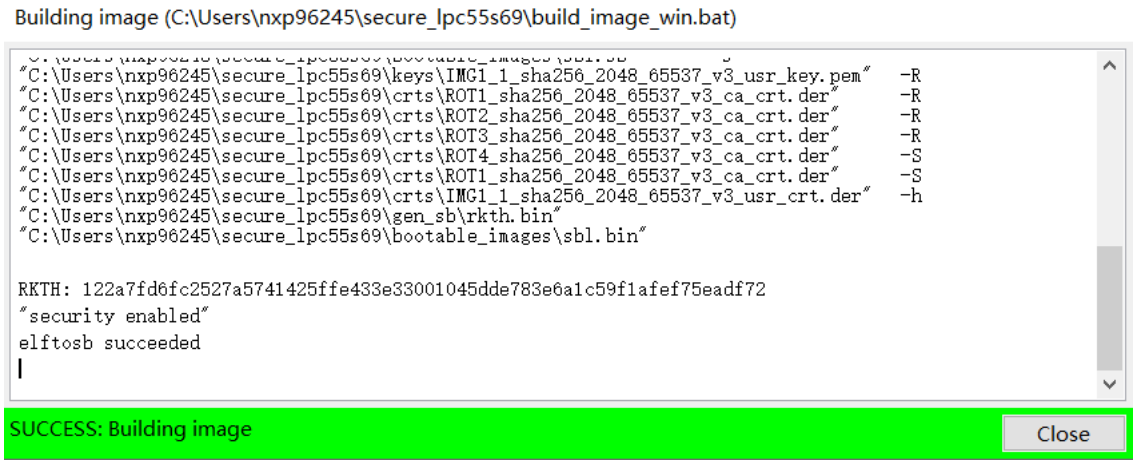


Figure 175. Generate booting image

8. Make sure that the board is connected and make the processor into ISP mode by pressing the ISP pin during reset stage.
9. Click the **USB** tab, set the connection to USB or UART according to selected port and test the connection to the processor

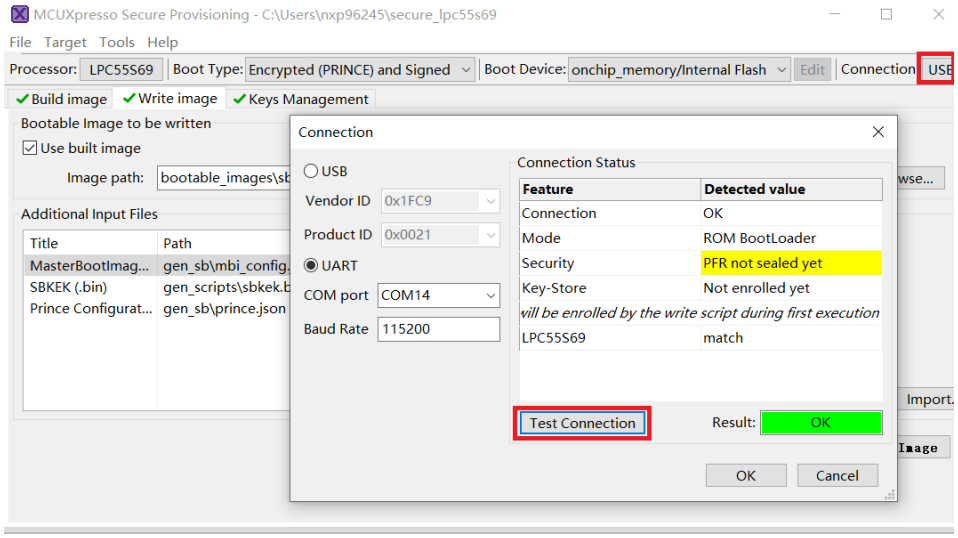


Figure 176. Connection dialog

10. Make sure that the Use built image checkbox is selected.
11. Click **Write Image**.

NOTE

check **Enable security** checkbox to seal the device security permanently.

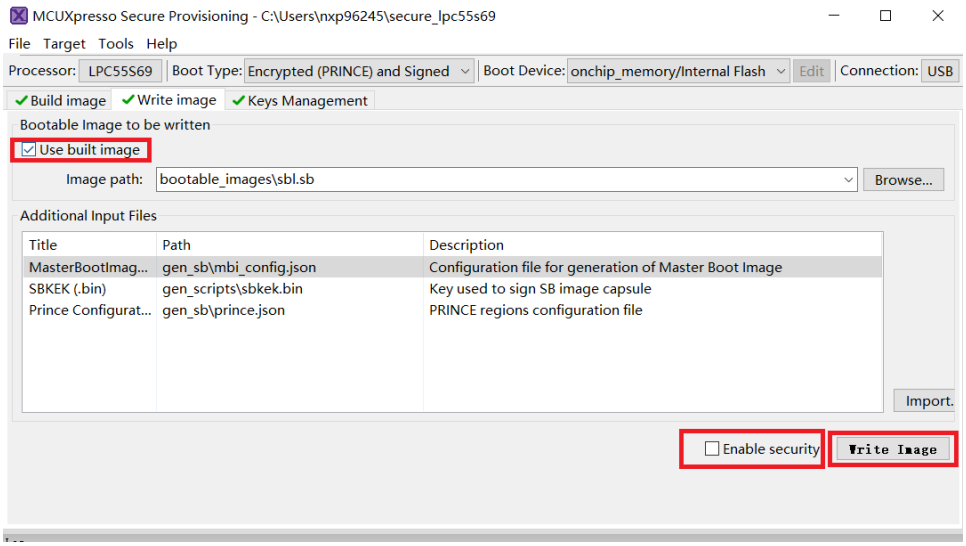


Figure 177. Programming image SBL

12. Write result is displayed in the progress window.

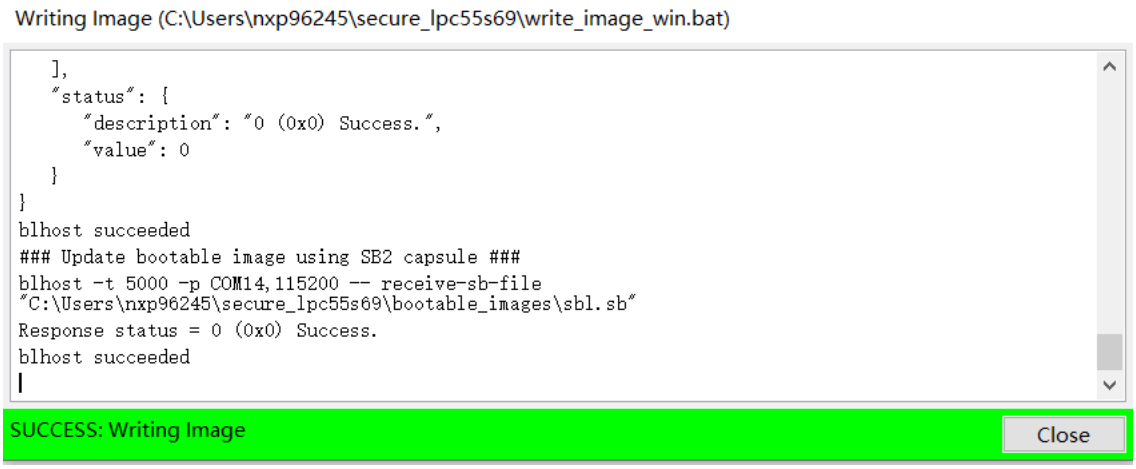


Figure 178. Write image message - success

7.4.3.5 Sign and Program application image

You can download an SFW image with BootRom in ISP mode or with SBL in ISP mode. Do the following steps to generate a signed SFW image:

1. Copy `sfw.bin` into folder *sbl/target/lpc55s69/secure*.
2. Edit `mbi_config.json` to set correct keys and certifications path or copy the older `keys and crts` from MCUXpresso Secure Provisioning lpc55s69 workspace to here.

```
"rootCertificate0File": "./crts/ROT1_sha256_2048_65537_v3_ca_crt.der",
"rootCertificate1File": "./crts/ROT2_sha256_2048_65537_v3_ca_crt.der",
"rootCertificate2File": "./crts/ROT3_sha256_2048_65537_v3_ca_crt.der",
"rootCertificate3File": "./crts/ROT4_sha256_2048_65537_v3_ca_crt.der",
"mainCertPrivateKeyFile": "./keys/IMG1_1_sha256_2048_65537_v3_usr_key.pem",
"chainCertificate0File0": "./crts/ROT1_sha256_2048_65537_v3_ca_crt.der",
"chainCertificate0File1": "./crts/IMG1_1_sha256_2048_65537_v3_usr_crt.der"
```

3. Open `signed_enc_sfw.bat` and set the correct installation path for tools `elftosb` and `blhost`.

```
SET "PATH=C:\nxp\MCUX_Provi_v3\bin\tools\elftosb\win;%PATH%"
SET "PATH=C:\nxp\MCUX_Provi_v3\bin\tools\blhost\win;%PATH%"
```

4. Set correct com port and configure PRINCE region in `sign_enc_sfw.bat`.
5. Enter folder secure in scons environment by inputting `cd secure`.
6. Run `sign_enc_sfw.bat` to generate final signed application image.
7. Press any key to configure the PRINCE region to set encrypted region after generating signed image. You can omit this step and download the signed image.
8. Make the LPC55S69 EVK board to enter ISP mode.
9. Download image, if the write operation was successful, reset the board.

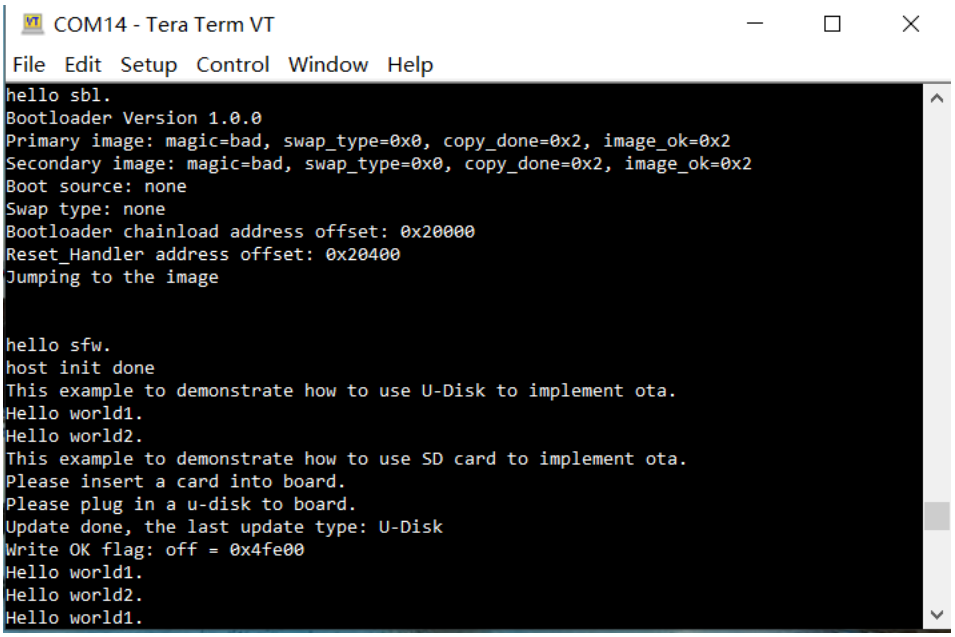


Figure 179. Terminal log

NOTE

For whole encrypted application image, pad image with 8k bytes aligned.

7.4.4 Secure boot demonstration for the platform EVKMIMXRTxxx

This section describes the steps to enable **XIP** encrypted authenticated boot. The demo targeted for the MIMXRT685S hardware platform is used as an example. This section also applies to platform MIMXRT595S.

7.4.4.1 Generating Keys and Certificates

To enable ROM secure boot, generate keys and certificates. Follow the steps to generate them.

1. Install the MCUXpresso Secure Provisioning tool.
2. Run this tool, click `File > New Workspace`, then select processor MIMXRT685 to create a new workspace.

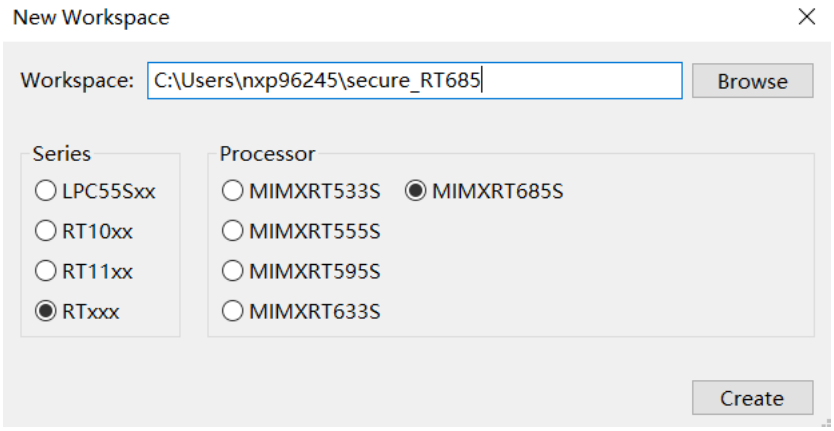


Figure 180. Creating a new workspace

3. Choose Boot Type as Signed.

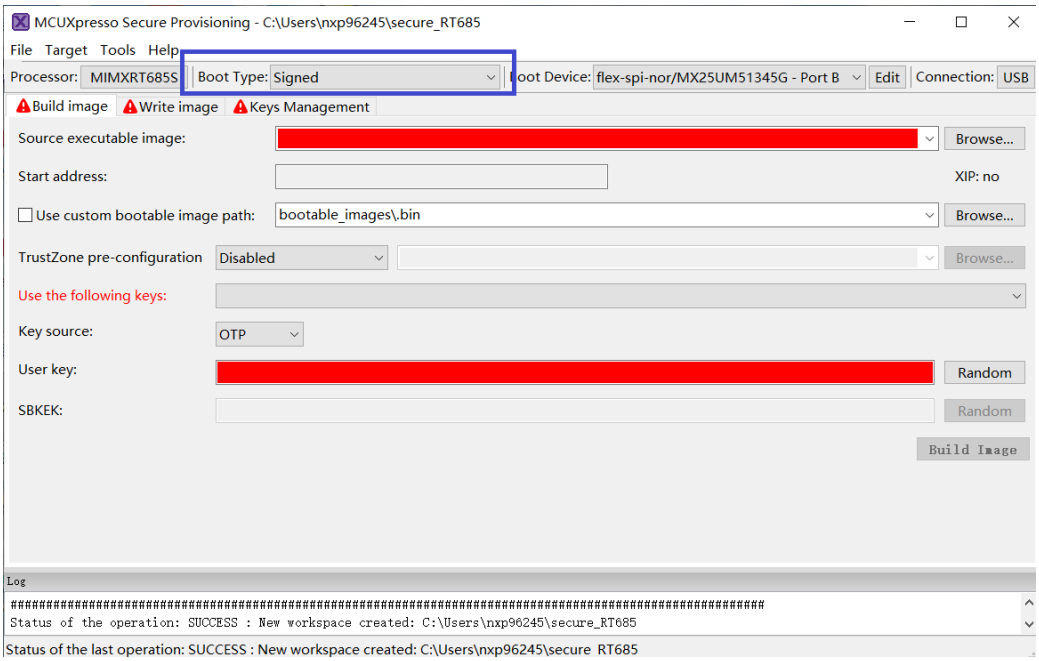


Figure 181. Selecting Boot Type

4. In the **Keys Management** view, click the button **Generated keys**, then specify all parameters in this menu.

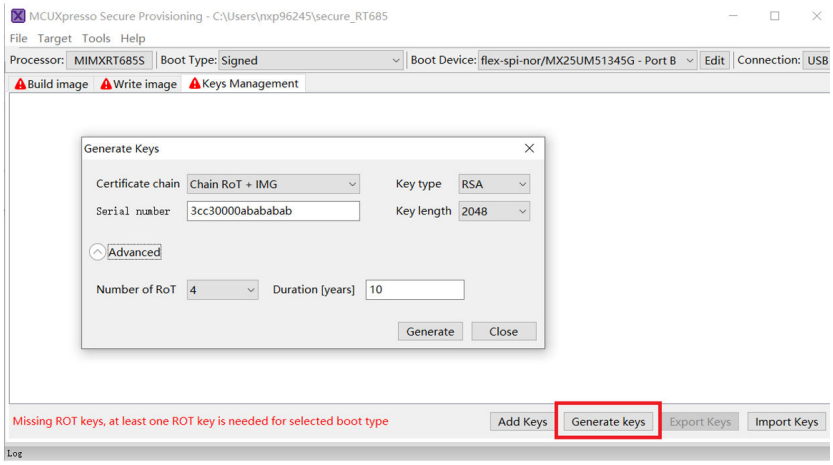


Figure 182. Generate keys

5. Click the button **Generate**. OpenSSL output is displayed in the progress window.

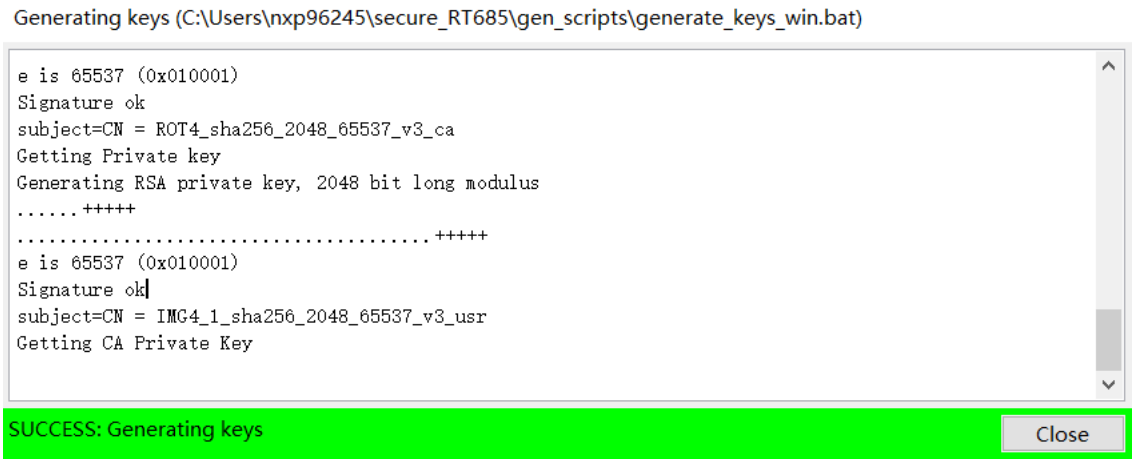


Figure 183. Generating keys - success

6. You can find generated keys and certificates in the folder `keys and crts` in the workspace directory. Copy folder “keys” and “crts” to folder `sbl/target/evkmimxrt600/secure`.

7.4.4.2 SBL image preparation

The following steps describe the procedure for creating SBL image.

1. Run `env.bat` which is under folder `sbl/target/evkmimxrt600`
2. Run `scons --menuconfig` to enter the menu `MCU SBL Core` to select `Enable ROM to verify sbl`.

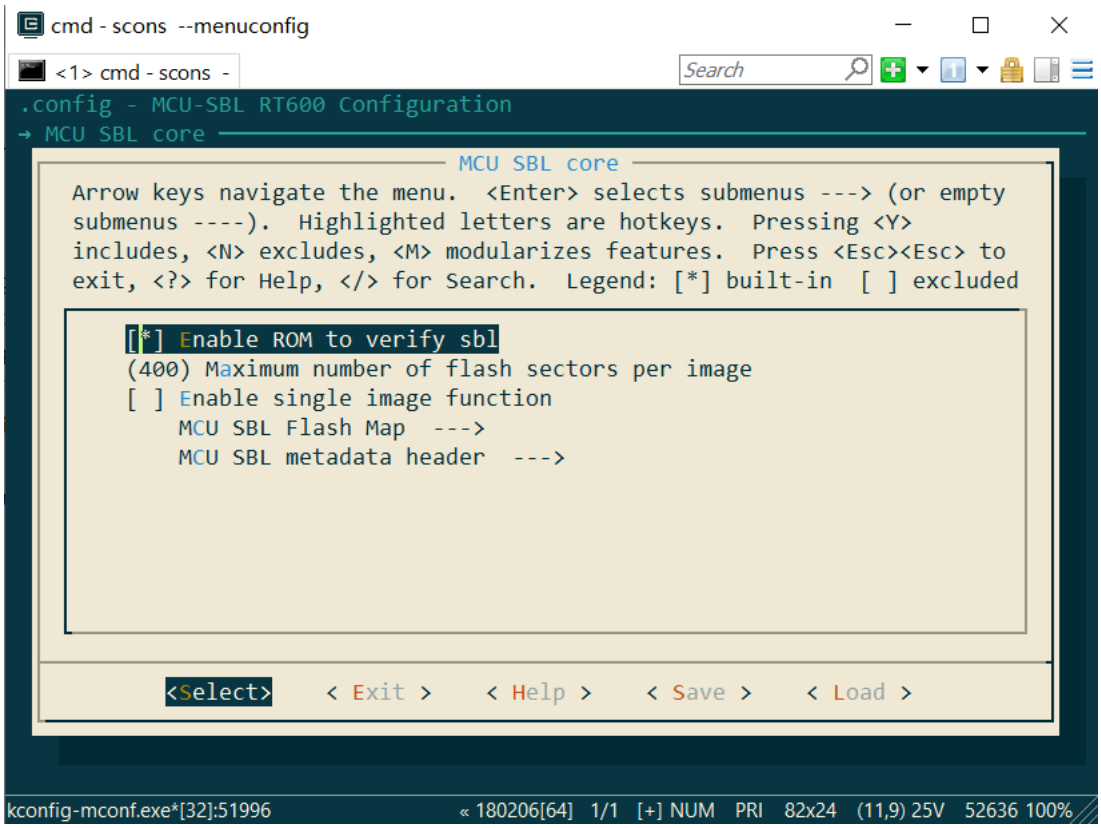


Figure 184. Enable ROM to verify sbl

3. Enter menu `MCU SBL Component > secure > selected signing method`, select one application signature type, save, and quit menuconfig.

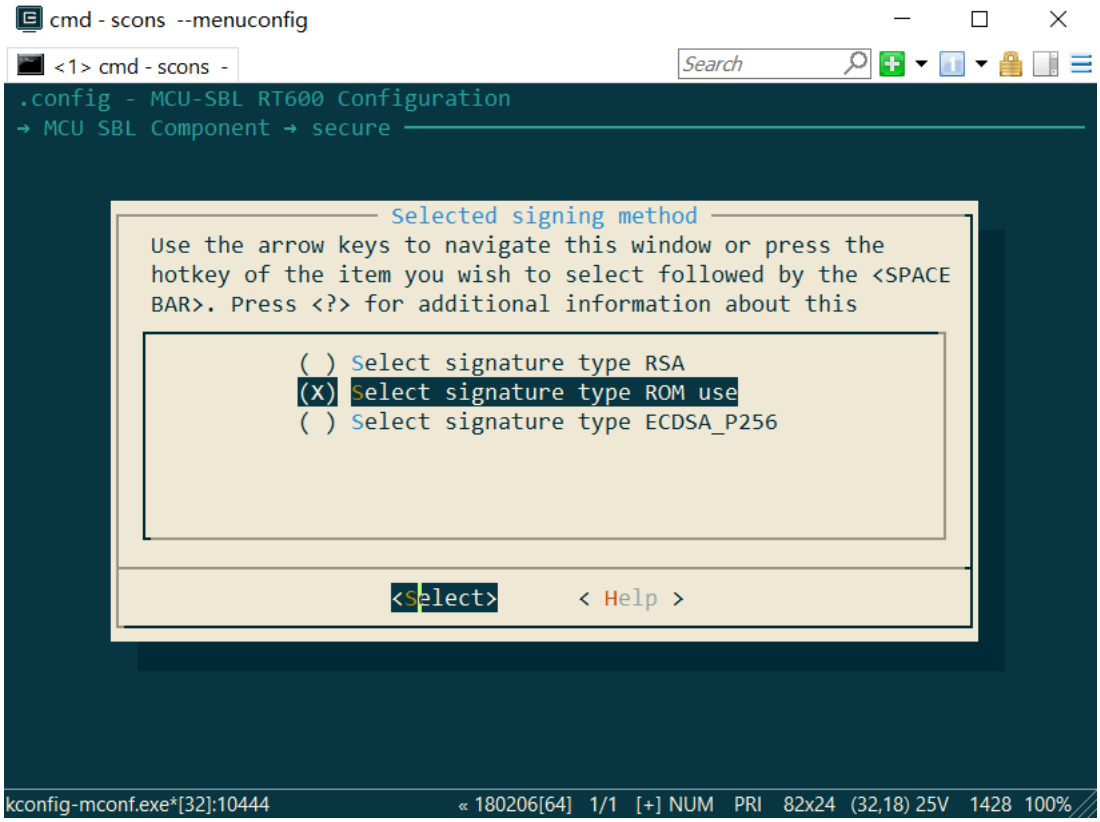


Figure 185. Selecting signing method

4. Run `scons --ide=iar` command to generate the IAR project or run `scons --ide=mdk5` to generate the Keil project.
5. Configure option to generate an image with binary format, then build the project.

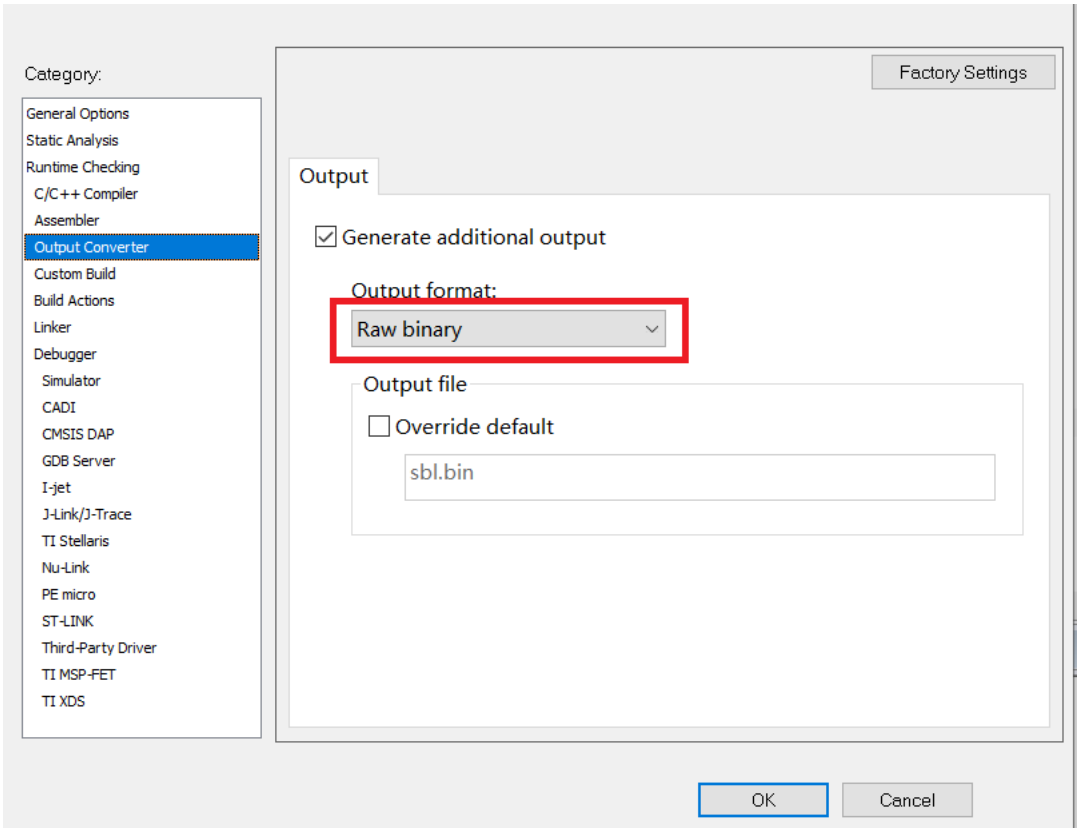


Figure 186. Set output format

7.4.4.3 Application image preparation

To generate plain application image, the steps are as below:

1. Run `env.bat` which is under folder `sfw/target/evkmimxrt600`.
2. Run command `scons --menuconfig`.
3. Enter menu `MCU SFW core` and uncheck menu `Enable sfw standalone xip`.

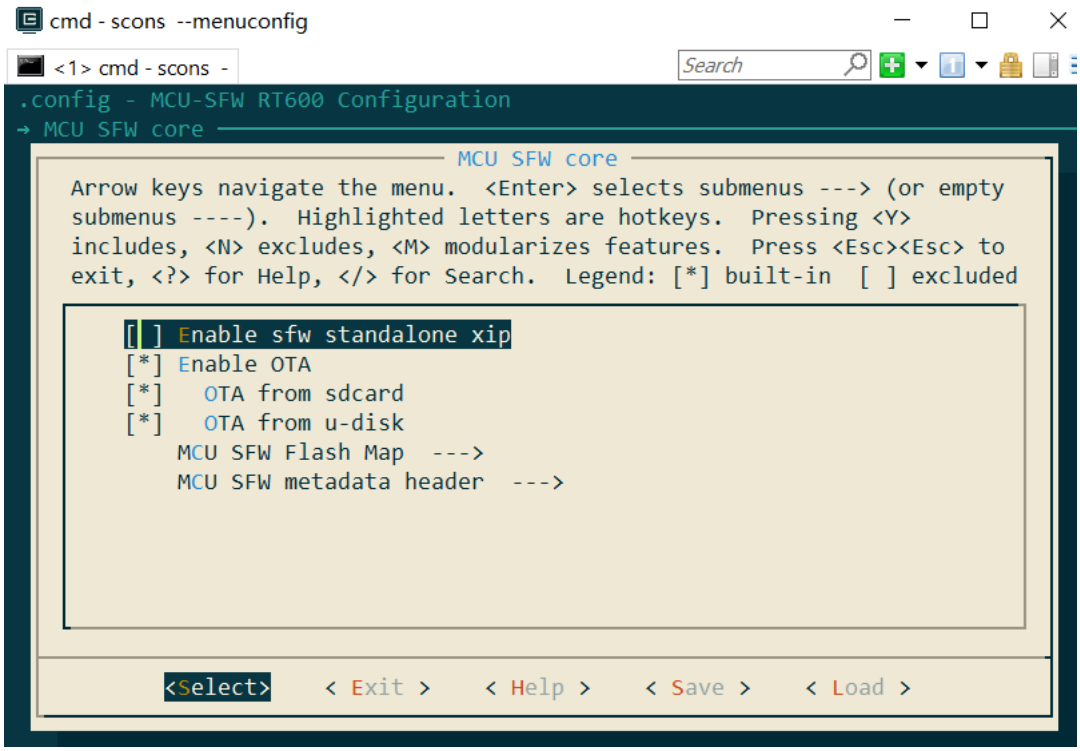


Figure 187. Uncheck enable sfw standalone

4. For image, which must be encrypted, enter the menu `MCU SFW component > secure` and check the menu `Encrypted XIP function`, then save and quit menuconfig .

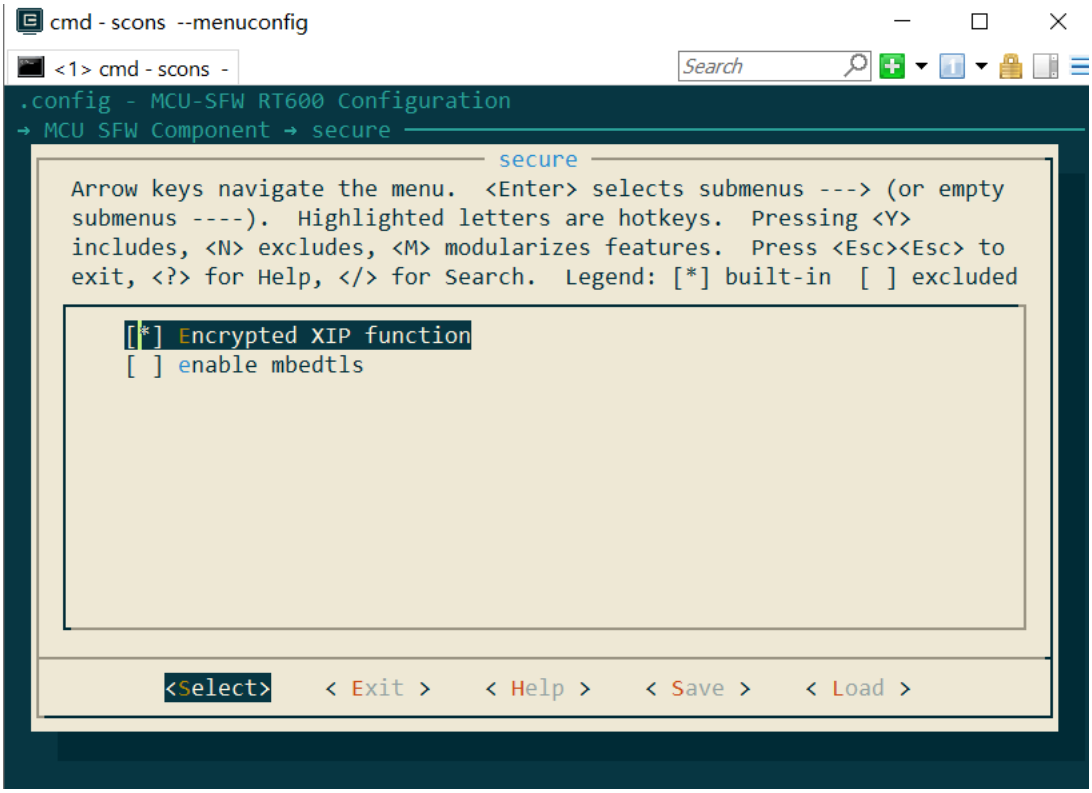


Figure 188. Enable key context upgrade in sfw

5. Generate the SFW project.
6. Configure the option to generate an image with binary format, then build the project.

7.4.4.4 Run signed SBL and SFW

Do the following steps to generate signed SBL and application, then program them to board.

1. Copy `sbl.bin` and `sfw.bin into` the folder `sbl/target/ evkmimxrt600/secure`.
2. Make the evkmimxrt600 board to enter ISP mode Downloader modes based on ISP pins (ISP0 on, ISP1 off, ISP2 off).
3. Connect a micro USB cable from connector J5 (LINK USB).
4. Enter folder secure in scons environment by inputting `cd secure`.
5. Run `sign_sbl_app.bat` to generate final signed SBL and application. If you want to generate the application image for the next update, remember to change the parameter version number in the `imgtool.py` command line.
6. Download signed SBL and application by the script `sign_sbl_app.bat` or other tools. This script downloads the application image into slot1. If you test the OTA procedure before, SBL may still try to run application in slot2.
7. Program OTP according to section 7.4.4.6
8. Switch to normal boot mode, then reset board. You see the output in the terminal.

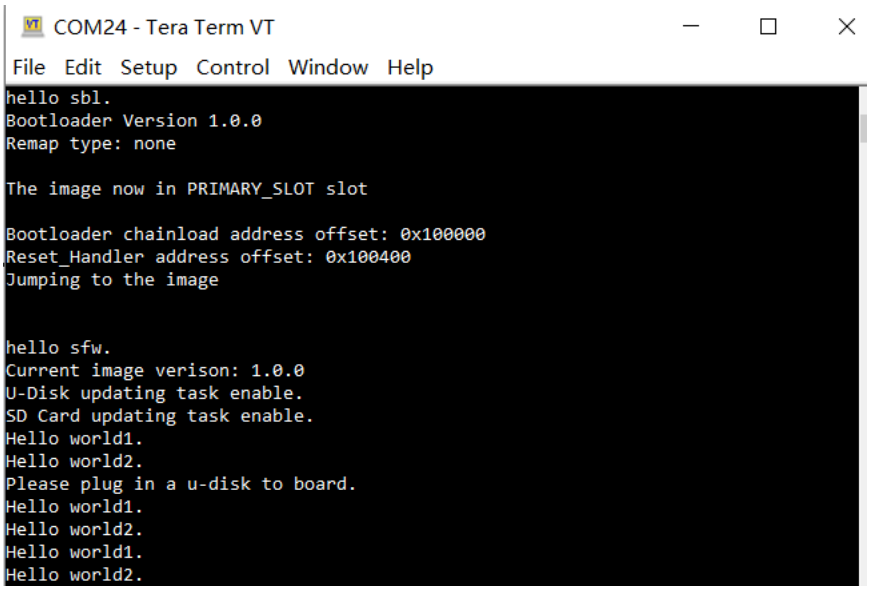


Figure 189. Terminal log

7.4.4.5 Run encrypted SBL and SFW

Do the following steps to generate signed and encrypted SBL and application together.

1. Copy `sbl.bin` and `sfw.bin` into folder `sbl/target/ evkmimxrt600/secure`.
2. Edit `sign_enc_sbl_app.bat` and set the KEK and encrypted region.
3. Make the evkmimxrt600 board to enter ISP mode Downloader modes based on ISP pins (ISP0 on, ISP1 off, ISP2 off).
4. Connect a micro USB cable from connector J5 (LINK USB).
5. Enter folder secure in scons environment by inputting `cd secure`.
6. Run `sign_enc_sbl_app.bat` to generate final signed and encrypted SBL and application.
7. Download SBL and application by this script or other tools.
8. Program OTP according to section 7.4.4.6
9. Switch to normal boot mode, then reset board. You see the output in the terminal.

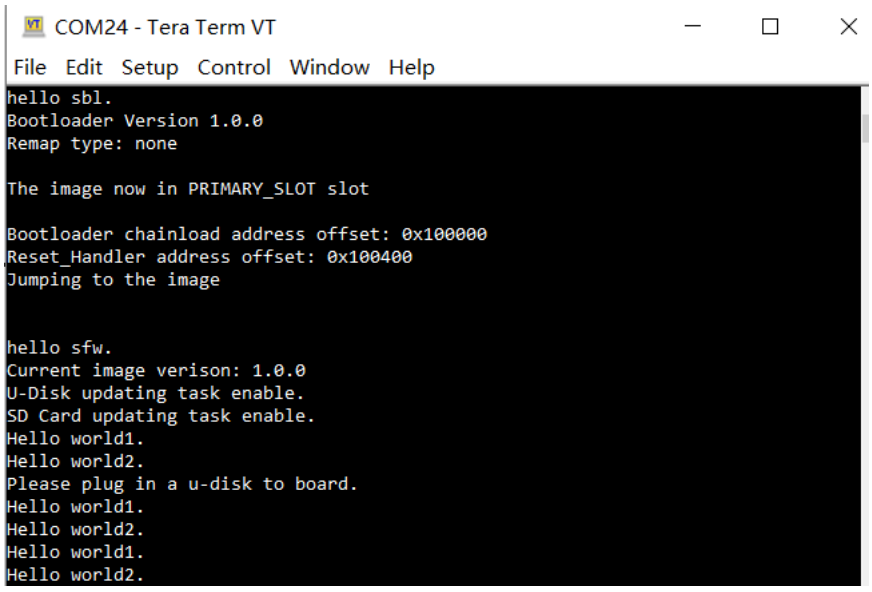


Figure 190. Terminal log

7.4.4.6 Program OTP (eFuse)

Below is an example to program SRK table and enable secure boot. In Development phase, user could use shadow registers to test the Secure boot.

1. Find and open script `program_ocotp.bat` in `sbl/target/evkmimxrt600/secure`, then set the correct installation path for tool blhost and com port.

```
SET "PATH=C:\nxp\MCUX_Provi_v3\bin\tools\blhost\win;%PATH%"
SET com_port=COMX,115200
```

2. Find the RKT hash(RKTH) value in the log of generating SBL image.

```
15. Output the root certificates SHA256 hash (RKTH).
        Success.
        RKTH: 8b8123193c27489fe835e104be046187dbc5507c310de41b469e68d5842decc0
```

Figure 191. RKTH value in hex

3. The RKTH value showed in the log is intended to be burned into the OTP fuses.The RKTH value generated by elftosb is in big-endian format. To store the value in OTP correctly, the byte order of the words supplied to efuse-program-once command needs to be byte-swapped to little-endian format. For example, 1923818b, 9f48273c, 04e135e8, 876104be, 7c50c5db, 1be40d31, d5689e46, c0ec2d84.

```
blhost -p COMx,115200 -t 15000 -- efuse-program-once 0x78 1923818b
blhost -p COMx,115200 -t 15000 -- efuse-program-once 0x79 9f48273c
blhost -p COMx,115200 -t 15000 -- efuse-program-once 0x7A 04e135e8
blhost -p COMx,115200 -t 15000 -- efuse-program-once 0x7B 876104be
blhost -p COMx,115200 -t 15000 -- efuse-program-once 0x7C 7c50c5db
blhost -p COMx,115200 -t 15000 -- efuse-program-once 0x7D 1be40d31
blhost -p COMx,115200 -t 15000 -- efuse-program-once 0x7E d5689e46
blhost -p COMx,115200 -t 15000 -- efuse-program-once 0x7F c0ec2d84
```

NOTE

If user program efuses with script program_ocotp.bat, remember to update the RKTH value according to user's log. The commands used to program efuses are commented out by default. User needs to enable it by hand.

4. User can enable secure boot using the following command:

```
blhost -p COMx,115200 -t 15000 -- efuse-program-once 0x60 0x900000
```

5. User can verify the eFuse value via command efuse-read-once

```
blhost -p COMx,115200 -t 15000 -- efuse-read-once 0x60
```

6. Enable OTFAD and configure OTP_MASTER_KEY, OTFAD_SEED efuses for image decryption

```
// program OTP_MASTER_KEY
blhost -p COMx,115200 -t 15000 -- efuse-program-once 0x70 ccddeeff
blhost -p COMx,115200 -t 15000 -- efuse-program-once 0x71 8899aabb
blhost -p COMx,115200 -t 15000 -- efuse-program-once 0x72 44556677
blhost -p COMx,115200 -t 15000 -- efuse-program-once 0x73 00112233
blhost -p COMx,115200 -t 15000 -- efuse-program-once 0x74 0c0d0e0f
blhost -p COMx,115200 -t 15000 -- efuse-program-once 0x75 08090a0b
blhost -p COMx,115200 -t 15000 -- efuse-program-once 0x76 04050607
blhost -p COMx,115200 -t 15000 -- efuse-program-once 0x77 00010203
// program OTFAD SEED
blhost -p COMx,115200 -t 15000 -- efuse-program-once 0x6c 62184d50
blhost -p COMx,115200 -t 15000 -- efuse-program-once 0x6d d5ae8d29
blhost -p COMx,115200 -t 15000 -- efuse-program-once 0x6e bf6af264
blhost -p COMx,115200 -t 15000 -- efuse-program-once 0x6f 3a72eb7f
// enable OTFAD boot
blhost -p COMx,115200 -t 15000 -- efuse-program-once 0x6a 00001000
// config flash settings for MIMXRT685-EVK
blhost -p COMx,115200 -t 15000 -- efuse-program-once 0x61 314000
```

7. Put ISP pins (ISP0 on, ISP1 off, ISP2 off) to make evkmimxrt600 board enter ISP mode Downloader mode, then run the script `program_ocotp.bat`.

7.4.4.7 Application OTA image preparation

Do the following steps to generate signed and encrypted application for upgrading.

1. Build the image as in section 7.4.4.3
2. Rename it to `sfw2.bin` and copy it to folder `sbl/target/evkmimxrt600/secure`.
3. Open `script sign_enc_sfw2.bat`, set the KEK and encrypted region.
4. Run `sign_enc_sfw2.bat`, file `sfw_2_enc.bin` is final image.

Chapter 8
Known issues

None

Chapter 9
Revision history

Table 8. Revision history

Revision	Date	Description
0	26 August 2021	First release to open source
1.1.0	08 November 2021	1. Support RT500, RT600 signature and encryption feature 2. Optimize the revert flow

How To Reach Us

Home Page:

nxp.com

Web Support:

nxp.com/support

Limited warranty and liability — 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.

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

Security — Customer understands that all NXP products may be subject to unidentified or documented vulnerabilities. Customer is responsible for the design and operation of its applications and products throughout their lifecycles to reduce the effect of these vulnerabilities on customer’s applications and products. Customer’s responsibility also extends to other open and/or proprietary technologies supported by NXP products for use in customer’s applications. NXP accepts no liability for any vulnerability. Customer should regularly check security updates from NXP and follow up appropriately. Customer shall select products with security features that best meet rules, regulations, and standards of the intended application and make the ultimate design decisions regarding its products and is solely responsible for compliance with all legal, regulatory, and security related requirements concerning its products, regardless of any information or support that may be provided by NXP. NXP has a Product Security Incident Response Team (PSIRT) (reachable at PSIRT@nxp.com) that manages the investigation, reporting, and solution release to security vulnerabilities of NXP products.

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

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

Date of release: 08 November 2021

Document identifier: MCUOTASBLSFWUG

arm