MCU-OTA SBL and SFW User Guide

Rev 1.1.0

Table of Contents

1. Introduction

This document provides the complete description of Secure Bootloader (SBL) features, project framework, quick start, and the various software settings. It describes the FOTA in detail, including image programming, switch, revert, signature, encryption and so on. Security is very important and described based on NXP MCU Soc secure engines. It also includes detailed steps to program image via MCU ISP (UART/USB). Other necessary information also 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 together with SBL. SFW supports to obtain the OTA firmware image by U-Disk, SD card in local, or AWS cloud, Aliyun cloud in remote, then SBL will check, authenticate the OTA firmware image and bootup it in normal.

1.1. Acronyms and abbreviations

The following table lists the acronyms used in this document.

Table 1-1 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
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 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 SB/SB2 binary.

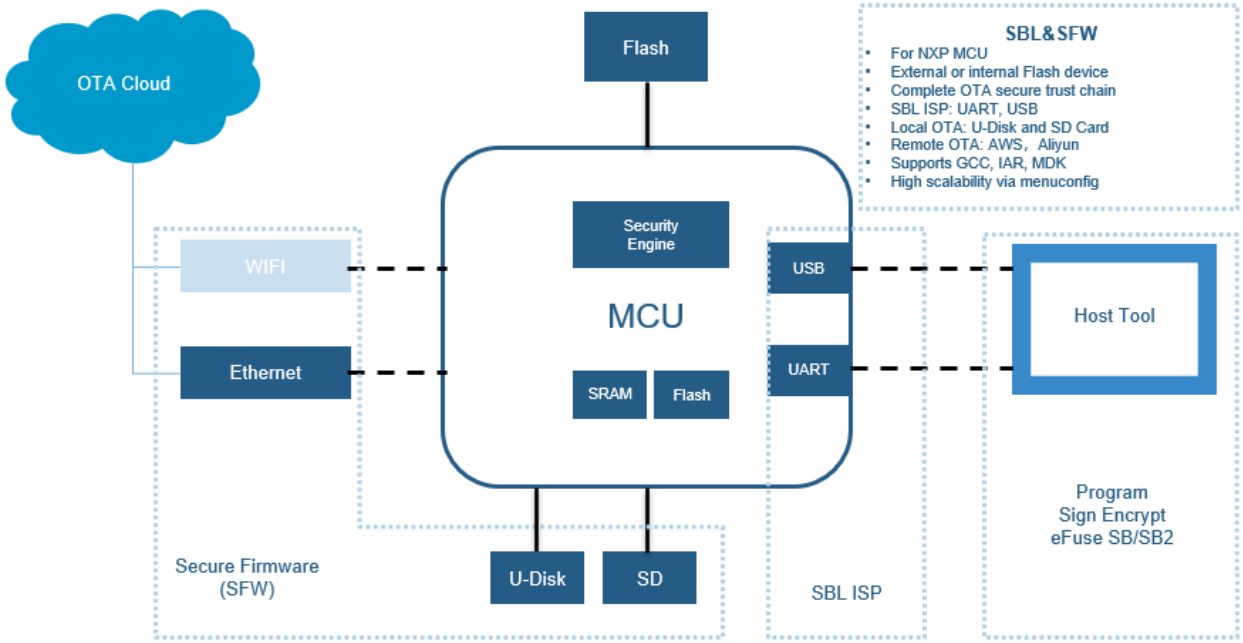


Figure 1-1 SBL and SFW diagram

Figure 1-2 shows detailed information about 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 configured easily based on the Soc features.

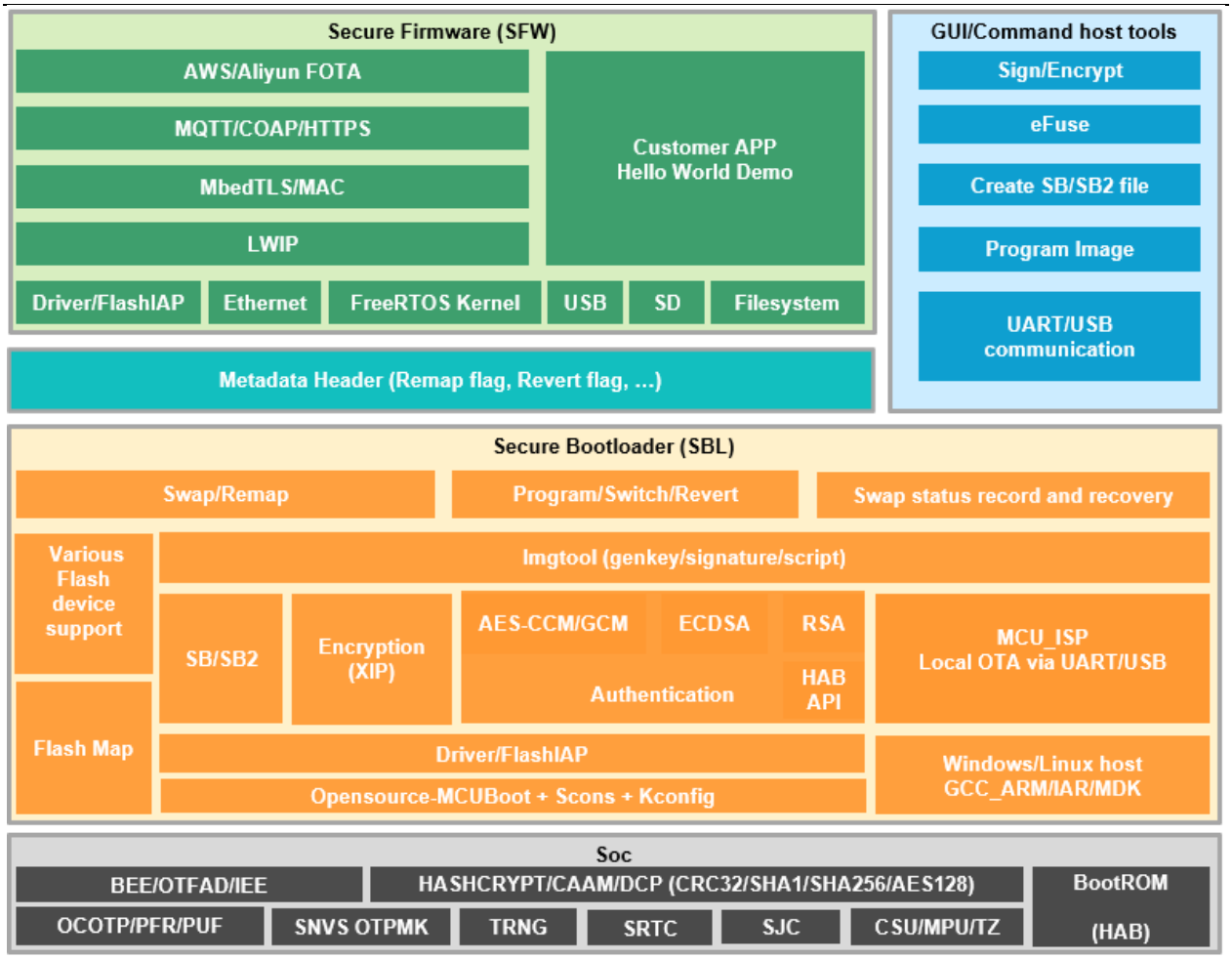


Figure 1-2 MCU-OTA project architecture

1.3. Features

- Support NXP MCU platforms (Table 1-2 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 1-2 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	●	●	●	●		●	●		

Refer to the following documents for detailed platform information.

- 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 1-3 and Table 1-3. The layer and description of SFW are showed in the Figure 1-4 and Table 1-4.

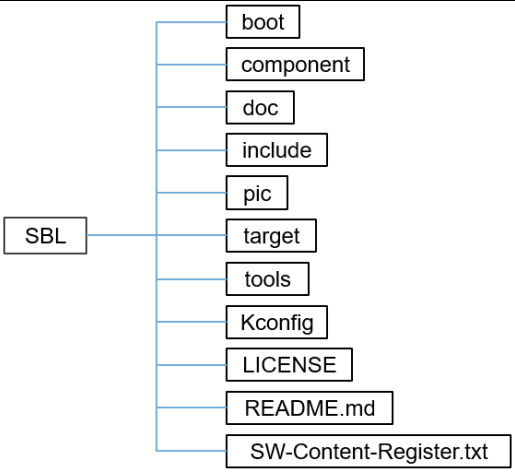


Figure 1-3 SBL directory organization

Table 1-3 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, flashiap 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

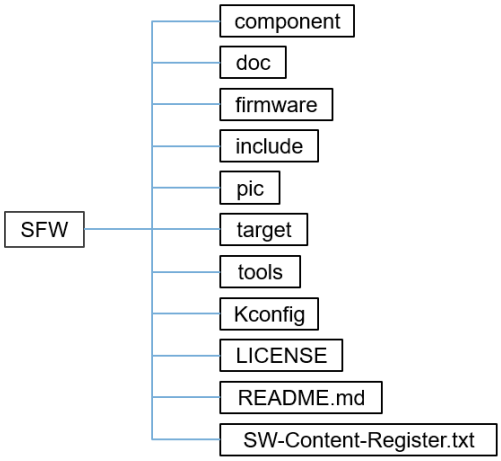


Figure 1-4 SFW directory organization

Table 1-4 SFW source code directories

Directory	Description
component	It includes SDK components and board peripheral drivers, for example, flashiap 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 project can be developed in both Linux host and Windows host. The system requirements are as below:

1. Linux host
   - Git
   - Python 3.6
   - SCons
   - GCC_ARM toolchain
   - Library: ncurses5-dev
2. Windows host

- Git bash
- GCC_ARM toolchain
- or IAR IDE v8.40
- or MDK IDE v5.30

2. Quick start

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

2.1. Windows host

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

2.1.1. GCC_ARM

At first, please obtain GCC_ARM toolchain from ARM or MinGW website and install to the Windows host.

Step 1: Clone SBL project and checkout to latest version, or download the release package

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

Step 2: Enter the directory *sbl/target/evkmimxrt1170/*

Step 3: Double click the batch file *env.bat*

Step 4: Configure the evkmimxrt1170 project

In *env.bat*, running `scons --menuconfig` command then the SBL configuration menu will be generated.

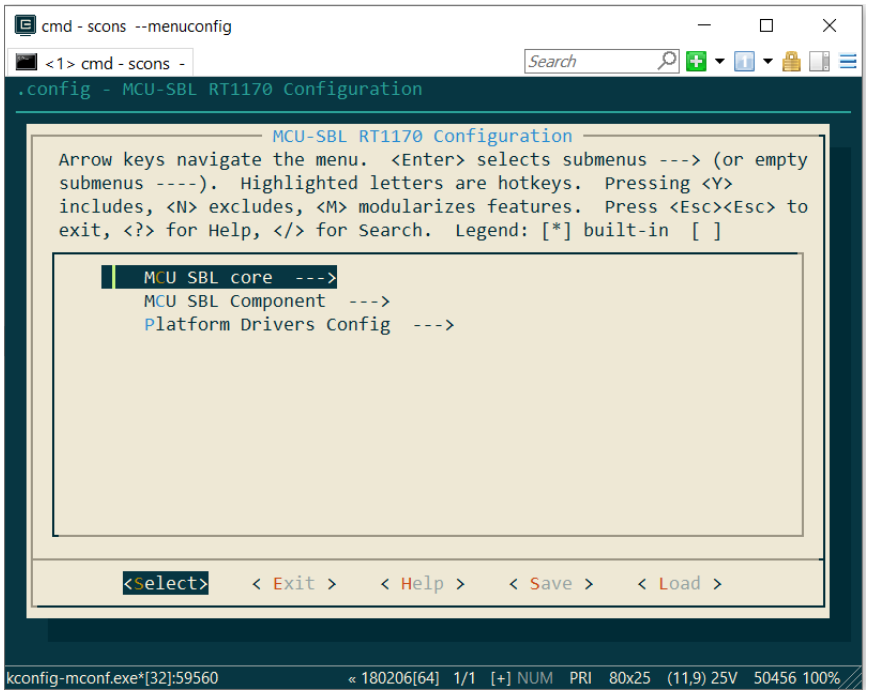


Figure 2-1 SBL configuration menu

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

Step 5: Build and download SBL project

(1) 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 2-2 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*, So , 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 into Windows environment variable to cover the 'EXEC_PATH'. For SFW, the environment variable is 'SFW_EXEC_PATH'.

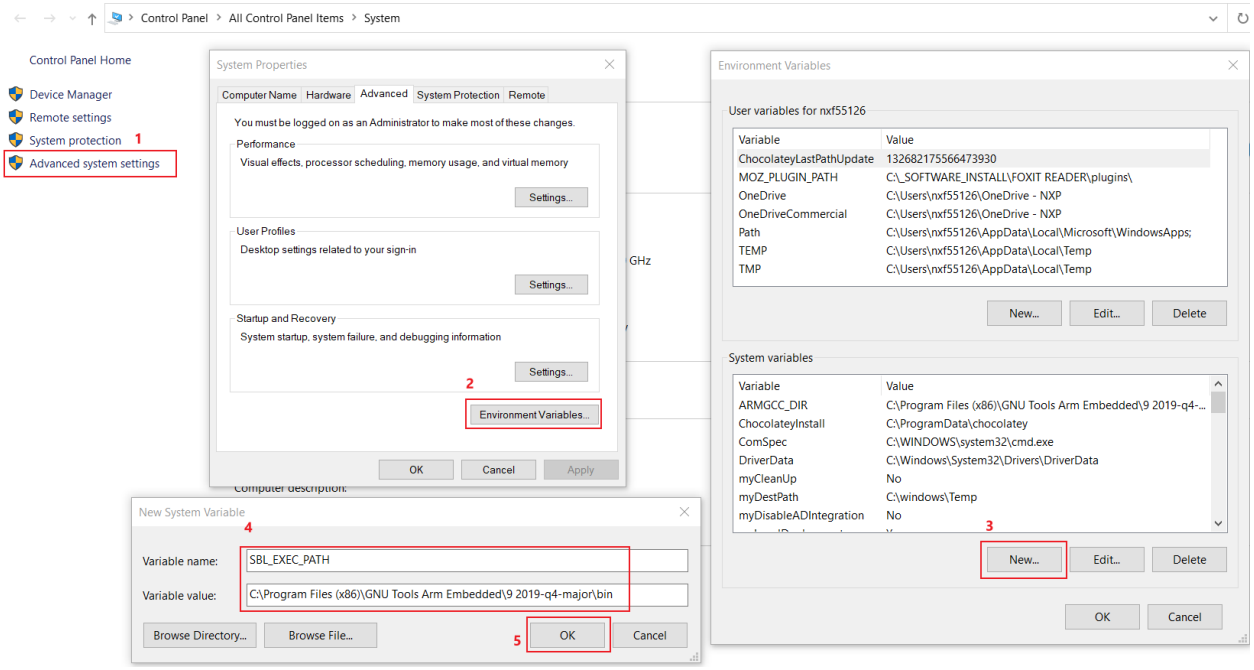


Figure 2-3 Windows environment variable SBL_EXEC_PATH

(2) Build the project

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

If successfully built, *sbl.bin* image will be built in *sbl/target/evkmimxrt1170/build* directory.

(3) Download the project

Use micro USB cable to connect EVKMIMXRT1170 board to the computer, set the board to the serial download mode, then use DapLink drag-n-drop function or other tools to download *sbl.bin* image to the board.

After image successfully downloaded into the board, set the board to XIP mode, then reset the board.

2.1.2. IAR IDE

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

Step 1~ Step 4 are same as the section 2.1.1

Step 5: Build and download SBL project

(1) Create IAR project for EVKMIMXRT1170 platform

In env.bat file, use `scons --ide=iar` command to generate 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 micro USB cable to connect EVKMIMXRT1170 board to the computer, set the board to the serial download mode, then click the "Download" button  to download the project to the board.

After image successfully downloaded into the board, set the board to XIP mode, then reset the board.

2.1.3. MDK IDE

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

Step 1~ Step 4 are same as the section 2.1.1

Step 5: Build and download SBL project

(1) Create MDK project for EVKMIMXRT1170 platform

In env.bat file, use `scons --ide=mdk5` command to generate 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 micro USB cable to connect EVKMIMXRT1170 board to the computer, set the board to the serial download mode, then click the "Download" button  to download the project to the board.

After image successfully downloaded into 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.

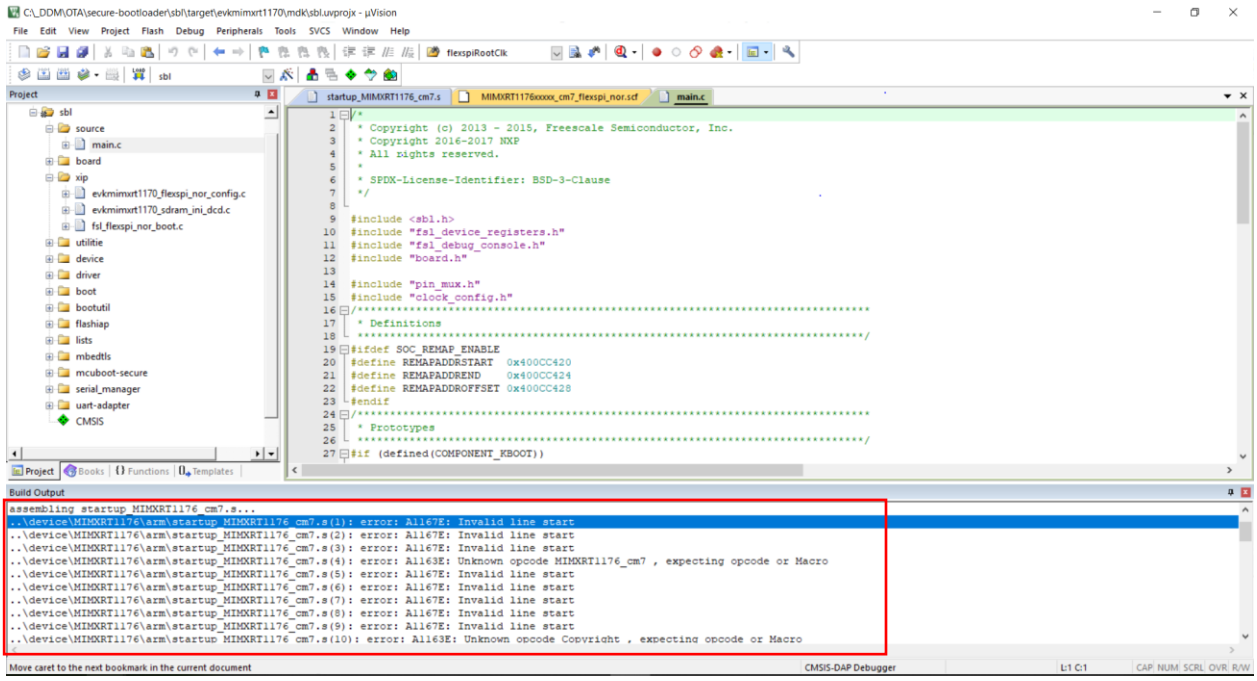


Figure 2-4 Compile errors

This can be solved by following ways:

- With MDK version 5.30(or later)

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

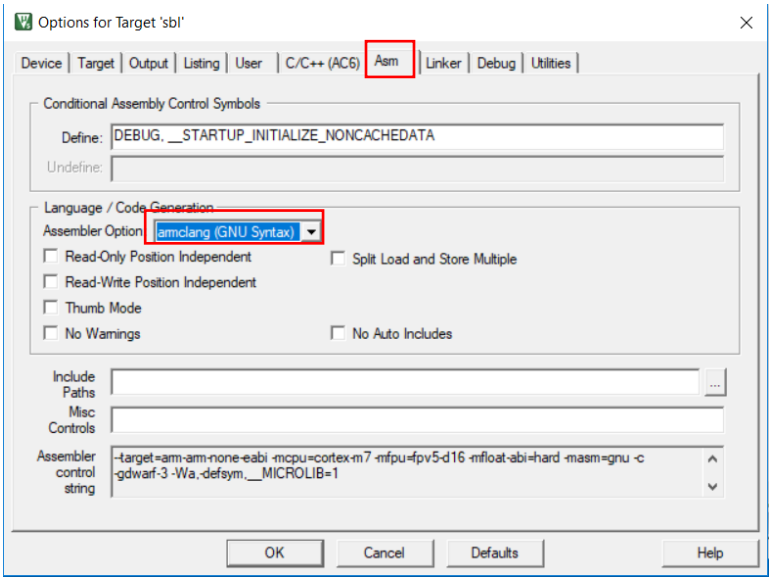


Figure 2-5 Assembler Option

- With an earlier version than MDK 5.30
  - select the option Assemble by using ArmClang V6
  - configure Misc Controls to -masm=auto

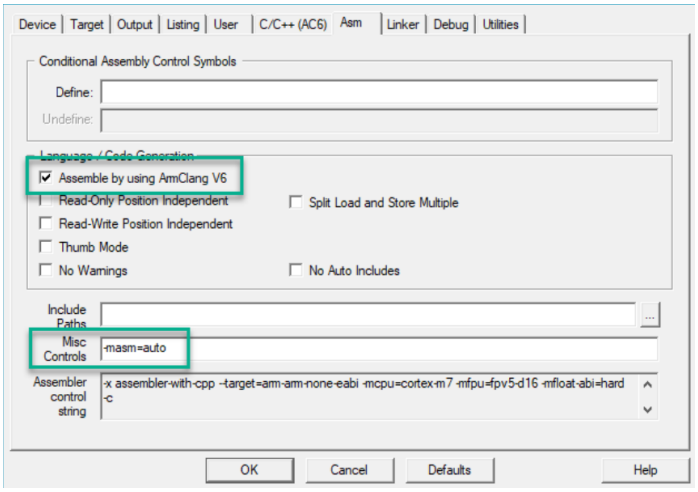


Figure 2-6 Asm Option

2.2. Linux host

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

Step 1: Install SCons

For Ubuntu or Debian, use command:

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

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

```
$ sudo yum install scons
```

Step 2: Install the GCC_ARM toolchain like: *gcc-arm-none-eabi-9-2019-q4-major*

Step 3: Clone SBL project and checkout to latest version, or download the release package

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

Step 4: Switch to the evkmimxrt1170 directory

```
$ cd target/evkmimxrt1170
```

Step 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 2-7 EXEC_PATH in sblprofile.py

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

Step 6: Configure the evkmimxrt1170 project

```
$ scons --menuconfig
```

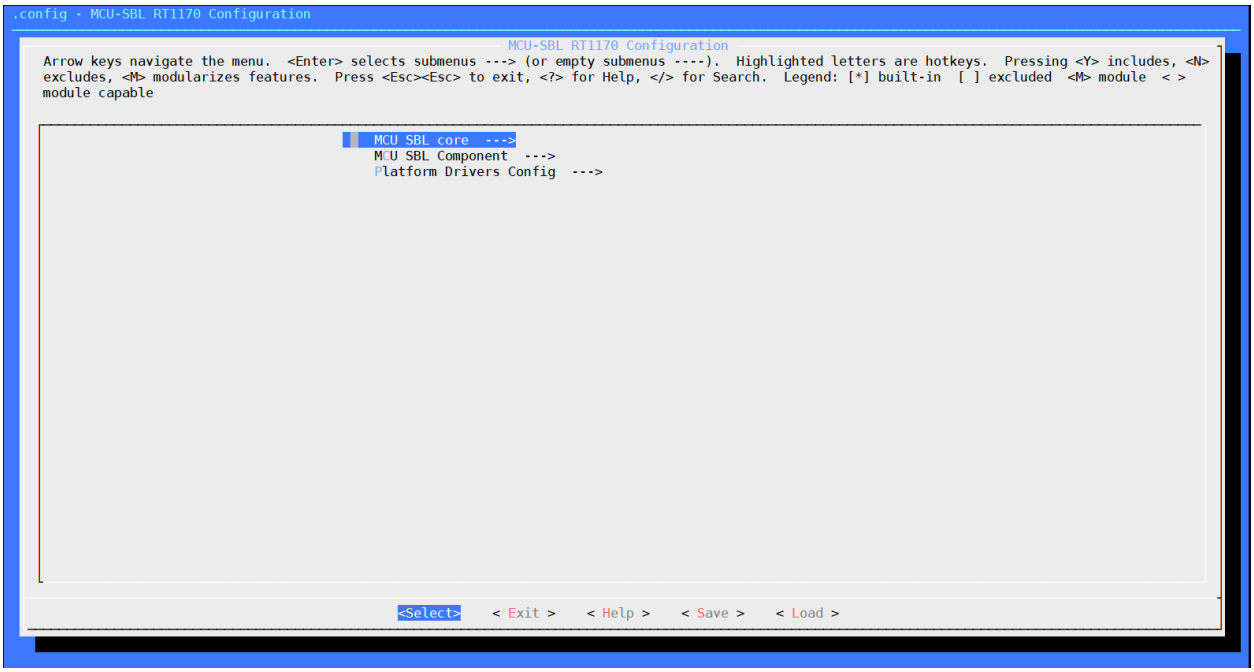


Figure 2-8 SBL configuration menu

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

Step 7: Build the image with GCC_ARM toolchain

```
$ scons
```

The *sbl.bin* image will be built in *sbl/target/evkmimxrt1170/build* directory.

Step 8: Program the image

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

2.3. SFW

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

1. Clone SFW project and checkout to latest version, or download the release package

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

2. SFW support two debug modes: SFW project XIP separately or SFW generate the bin file then used in conjunction with SBL. SFW configures which mode to use via 'scons --menuconfig'.

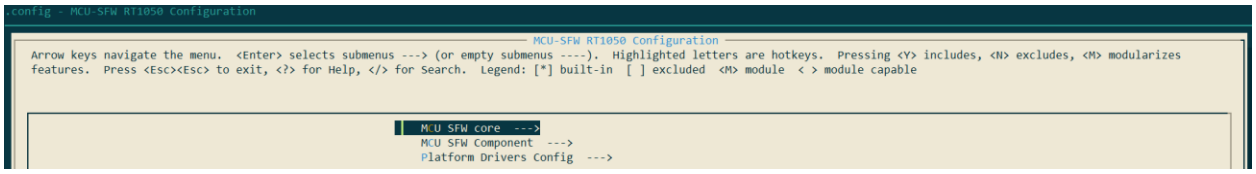


Figure 2-9 SFW configuration menu

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

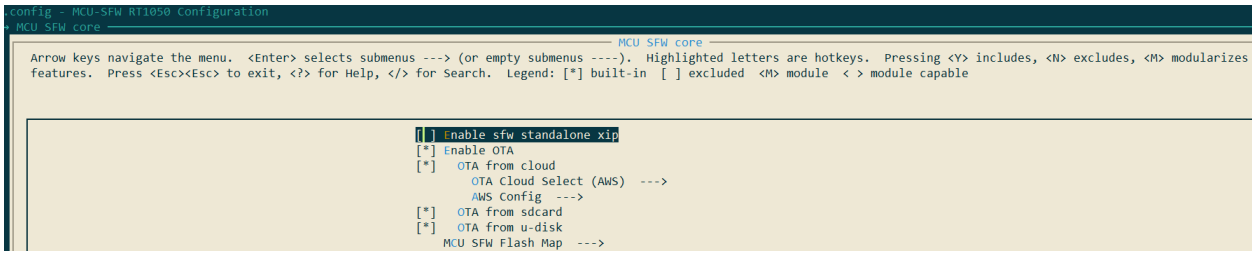


Figure 2-10 SFW debug mode config

3. Framework

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

3.1. SCons

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 using usual Makefile files. These files are also Python scripts and can be written using standard Python syntax. So, 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 in 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 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 control the compilation of the platform. In general, there is only one SConstruct file in one platform, but there will be 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 will be "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 command

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

1. scons

   Build the project for specific platform.

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

2. scons --menuconfig

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

3. scons --ide=xxx

   Generate IAR or MDK projects for specific platform.

   `scons --ide=iar` command generate one IAR project

   `scons --ide=mdk5` command generate one MDK project

3.2. Kconfig

SBL project uses the configuration file sblconfig.h generated by the Kconfig file to configure the system, and 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 will contain 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 will appear, as Figure 3-1 and Figure 3-2 shown.

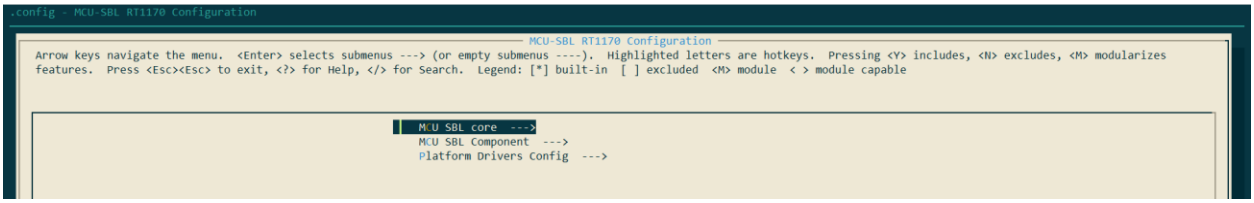


Figure 3-1 SBL menuconfig menu

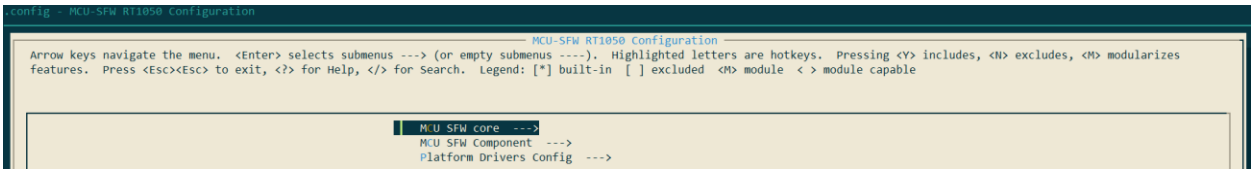


Figure 3-2 SFW menuconfig menu

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

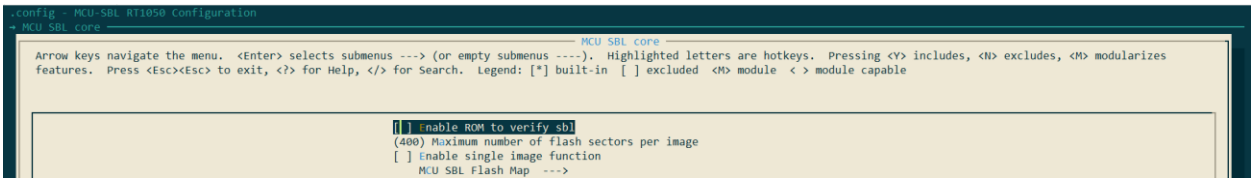


Figure 3-3 MCU SBL submenu

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 on the SBL and SFW developing and testing, here are three basic tools, for more others, please visit NXP official website or contact the FAE.

1. MCUXpresso Config Tools

   The 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 SBL target platform, there is a *MCUX_Config.mex* file which can be opened by MCUXpresso Config Tools and help to generate specific C code about clocks, pins and so on. For example: *target/evkbmimxrt1050/board/MCUX_Config/ MCUX_Config.mex*.

   For more information, please refer to the website:

   https://www.nxp.com/design/software/development-software/mcuxpresso-software-and-tools-/mcuxpresso-config-tools-pins-clocks-peripherals:MCUXpresso-Config-Tools

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). Please refer to the website for more information:

   https://www.nxp.com/docs/en/user-guide/MCUBLHOSTUG.pdf

3. MCUBootUtility

   NXP-MCUBootUtility 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-MCUBootUtility is a real one-stop tool, a tool that includes all the features of NXP's official security enablement toolset, and what's more, it supports full graphical user interface operation. With NXP-MCUBootUtility, it is easy to get started with NXP MCU secure boot. The main features of NXP-MCUBootUtility 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 MCUBootUtility, please refer to the website:

   https://github.com/JayHeng/NXP-MCUBootUtility

4. 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 doesn't receive isp command from host within the timeout period, then ISP process will be bypassed.

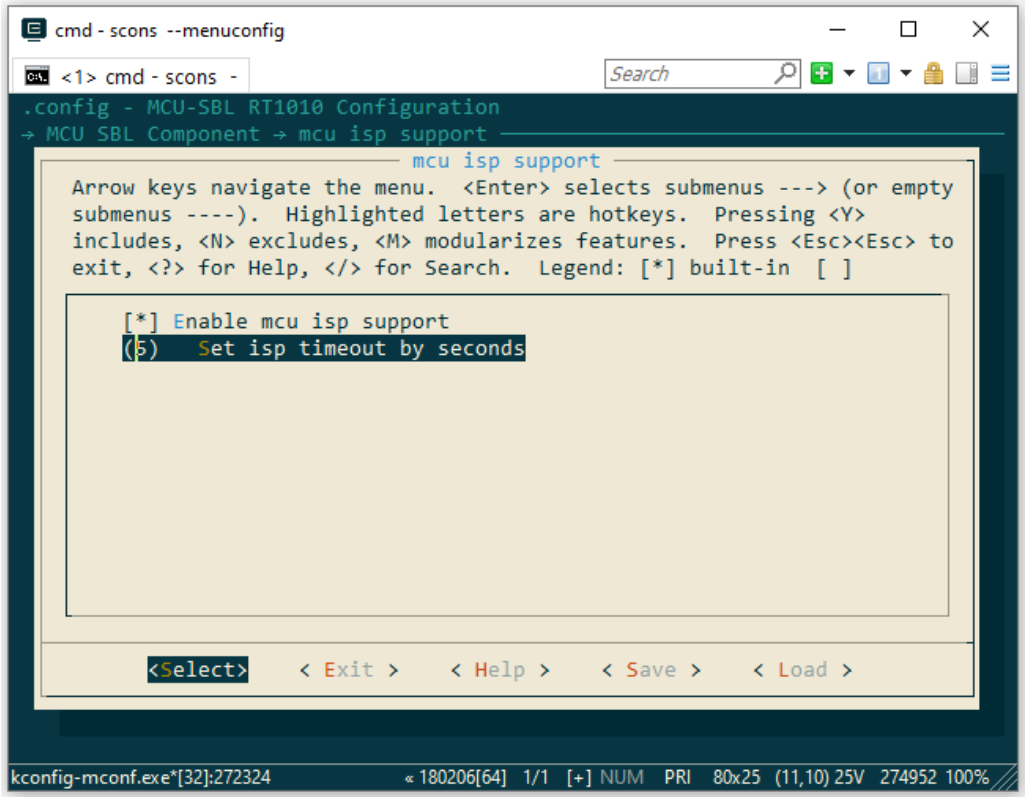


Figure 4-1 SBL menuconfig set timeout

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

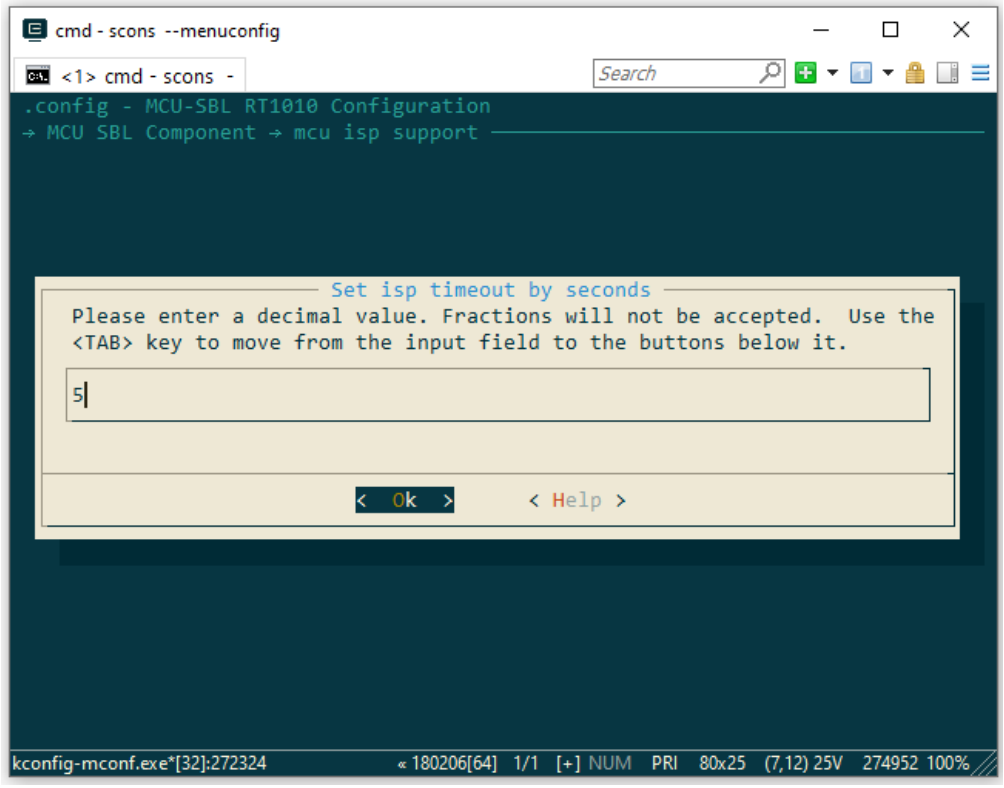


Figure 4-2 SBL menuconfig set timeout

4.4. MCUBootUtility usage

NXP-MCUBootUtility GUI tool (v3.3 or upper version) is recommended as the preferred host tool for ISP downloading (For those need usage of blhost command line tool, please contract NXP). See below steps:

Step 1: Open MCUBootUtility, set mode to “SBL OTA” in menu Tools/Run Mode.

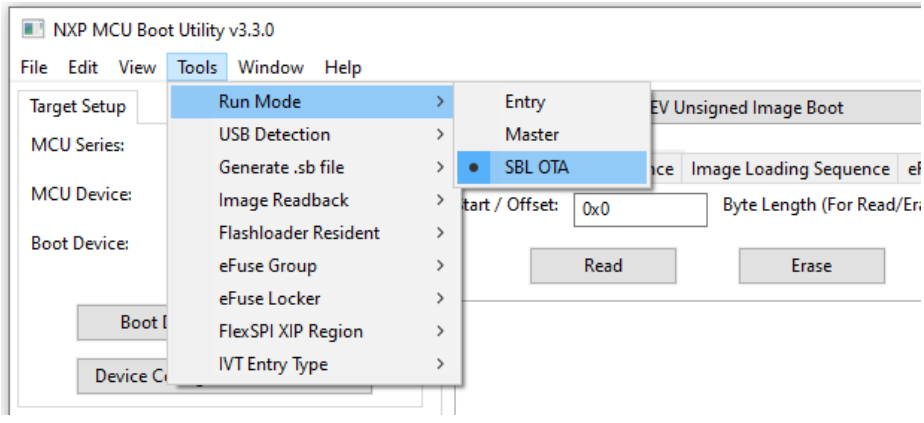


Figure 4-3 Set tool run mode

Step 2: Power on SBL target board (take EVKMIMXRT1010 as example), then connect USB cable to J9. If everything is ok, usb vid/pid will be detected. Click ‘Connect to SBL ISP’ button.

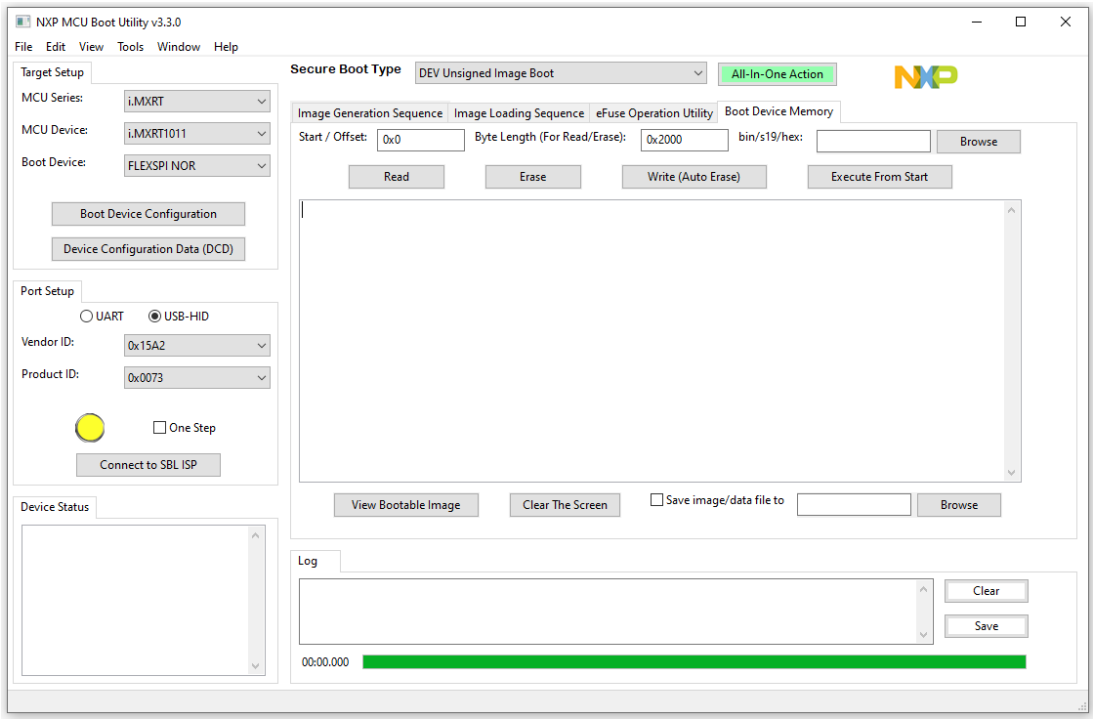


Figure 4-4 USB IDs are detected

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

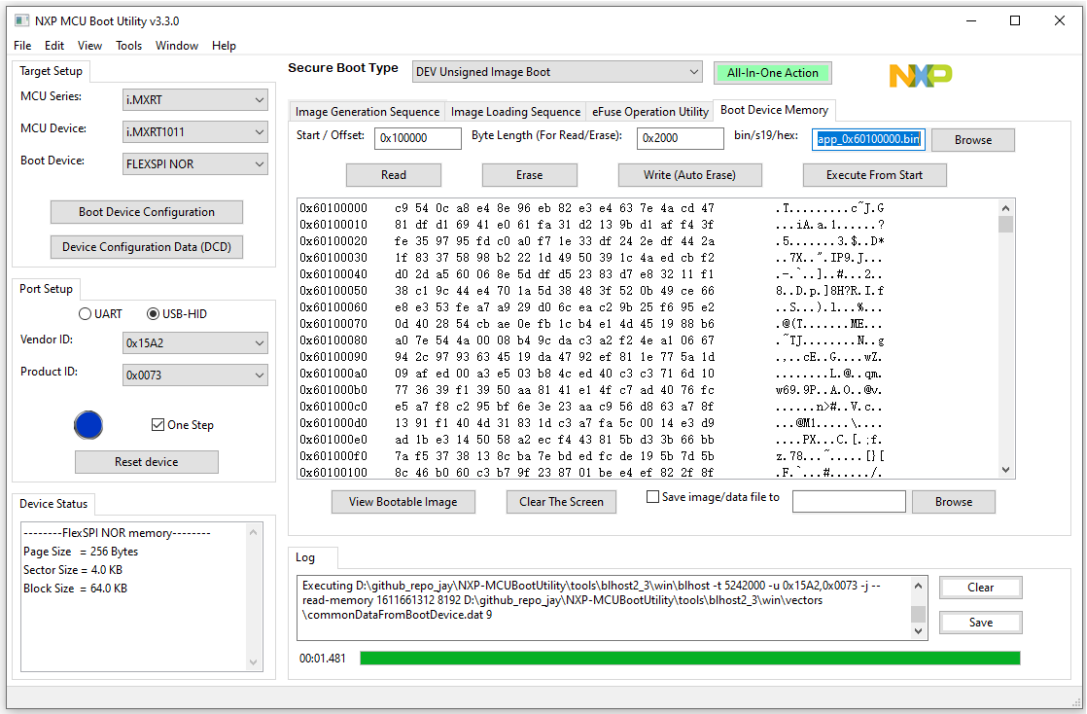


Figure 4-5 Download app

5. Security

This section describes the implemented security feature. Secure Bootloader (SBL) is based on mcuboot project. SBL keeps the mcuboot legacy signing method RSA and ECDSA. 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.

For mcuboot legacy signing method RSA and ECDSA. it signs the image by computing hash over the image, and then signing that hash. Please refer to mcuboot design document for the details. SBL use 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

Secure boot provides 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 will then examine the first user executable image resident in flash memory to determine the authenticity of that code. If the code is authentic, then control is transferred to it. This establishes a chain of trusted code from the ROM to the user boot code. In this case, BootROM verify SBL and SBL verify application image.

5.1.1. High Assurance Boot (HAB)

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

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

The OEM use a utility provided by NXP to generate private key and corresponding public key pairs. Then the private key is used to encrypt the digest of the image which OEM want to release. This encryption generates a unique identifier for the image which is called a signature. The certification with 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 public key by this value.

Below is the bootable image format for HAB.

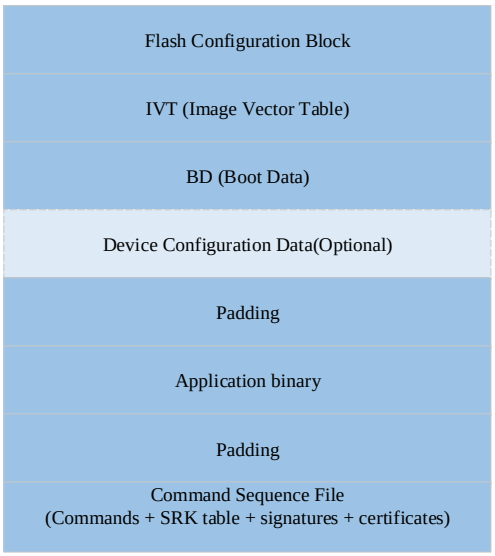


Figure 5-1 HAB signed image format without mcuboot header

It doesn't include Flash Configuration Block for bootable application. Because BootROM has configured flash by reading this field of SBL. All MIMXRT platforms support RSA public key (1024, 2048, 3072 or 4096), MIMXRT1170 also supports ECDSA signature verification using 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 x509 certificate embedded in the image.

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

The BootROM API skboot_authenticate is used to verify 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 flash region and can be programmed through ROM in ISP mode.

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

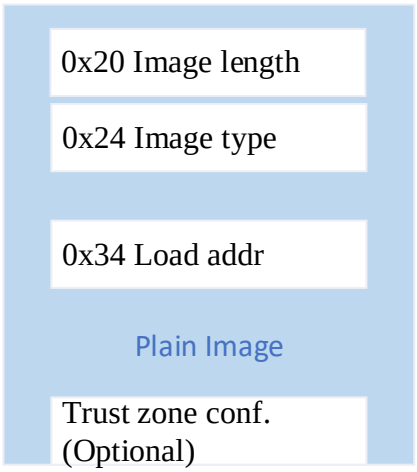


Figure 5-2 Signed image format without mcuboot header

5.2. 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.

5.2.1. 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 needs to 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.

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

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

5.2.2. 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 located 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.

5.2.3. 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, which initializes PUF and reconstruct original keys so that each key can be used if needed.

5.3. Image Format

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

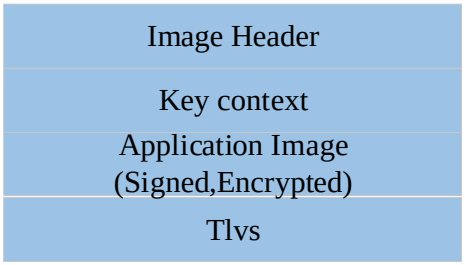


Figure 5-3 Final image format

5.4. Tools

In order to use security feature, user needs to 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 tool to generate a 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. Please download from the website:

https://www.nxp.com/design/software/development-software/mcuxpresso-software-and-tools-/mcuxpresso-secure-provisioning-tool:MCUXPRESSO-SECURE-PROVISIONING?tab=Design_Tools_Tab

In 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.

6. Firmware

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 to obtain the OTA firmware image by U-Disk, SD card in local, or AWS cloud, Aliyun cloud in remote, then SBL will check, authenticate the OTA firmware image and bootup it in normal.

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

For both swap and remap mode, SFW provide 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 use macros to distinguish them.

6.2. Operation to set 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 located in the last 32 bytes of two slots**) is used to judge the swap type and control the roll back. Figure 6-1 shows the state of the flag. Unset is 0xFF, Set is 0x01.

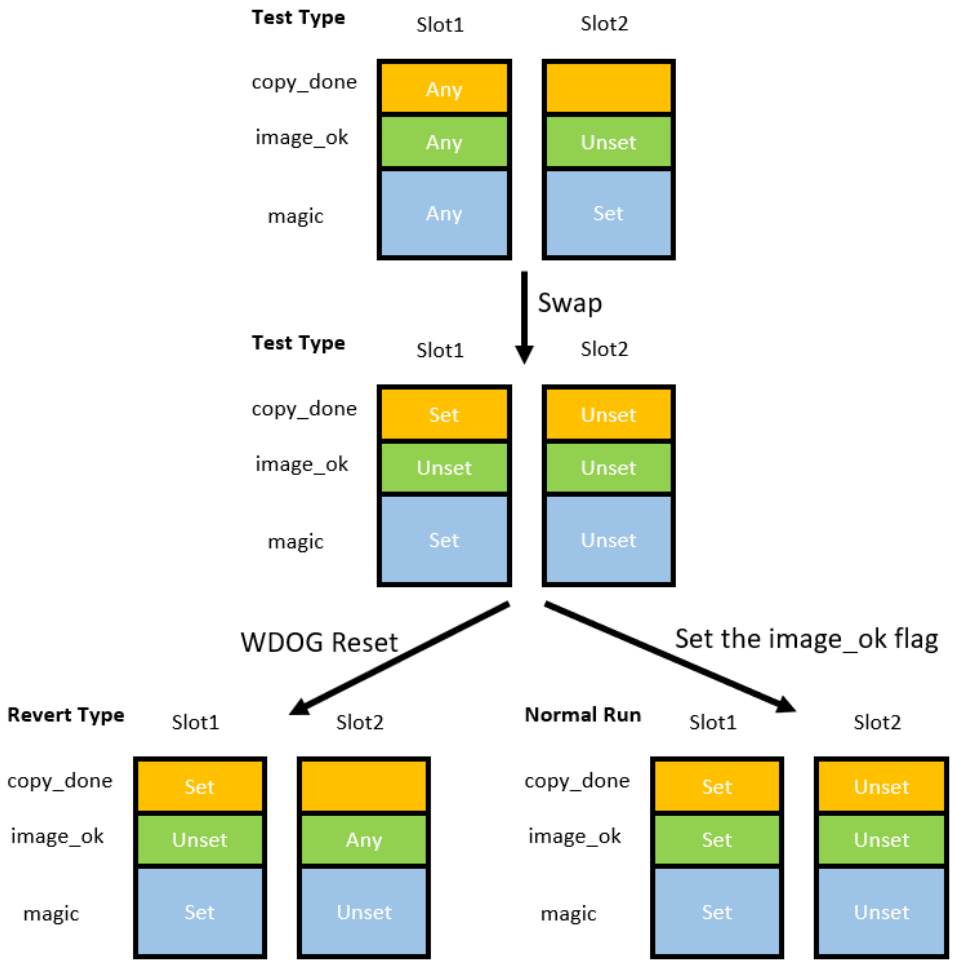


Figure 6-1 Swap flag state

To initialize an OTA process, after writing the new firmware to slot2, the old firmware which getting the new firmware should write the magic(fix value, 16bytes) to the end of slot2 to inform the bootloader that an new firmware has been written to the slot2. After writing the magic value done, reset the board.

Bootloader now judge 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 will be cleared. Then the bootloader jumps to the slot1 to execute the new firmware, if the new firmware operates normally, it will write the image_ok flag to slot1 trailer to disable the revert. Otherwise, error occurs in the new firmware, the image_ok flag not set, then the watchdog reset the board, the bootloader judge the OTA type, now the type is Revert, exchange the two slots, and clear the trailer position of the slot2, now the trailers of two slots are all unset.

Note: For the board using swap mode OTA, the firmware should contain two writing flag operations. First one is writing magic part of the flag, this operation should be executed after the new firmware is written. The magic address is 0x2FFFF0. Second operation is writing image_ok flag, after the firmware itself run the whole task period and during the period everything is ok, the firmware should set the flag. The address of image_ok is 0x2FFFE8. The magic value is as Figure 6-2.

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

Figure 6-2 Magic value

6.2.2. Operation for remap mode OTA

For the remap mode OTA, the remap update flag (**the flag is located in the fixed offset address of the flash, the offset is 0xFFFE0, flag structure occupies 32 bytes of space**) is used to judge the remap type and control the roll back. Figure 6-3 shows the state of the flag. Unset is 0xFF, Set is 0x01, revert is 0x04.

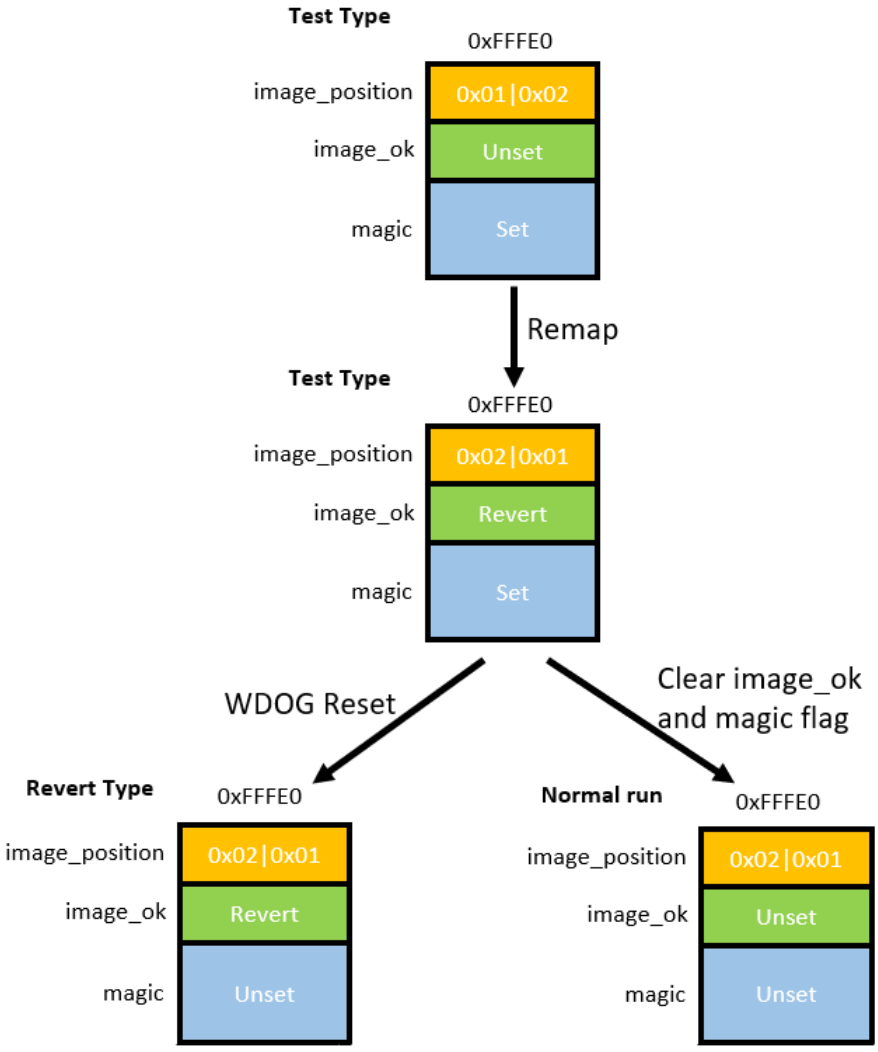


Figure 6-3 Remap flag state

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

The bootloader read the remap update flag to get the position of current firmware 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 to the slot2 physically. If the new firmware operates normally, it will clear the image_ok and the magic part of the flag to disable the roll back. Otherwise, error occurs in the new firmware, the new firmware no do clear the flag, then the watchdog reset the board, the bootloader judge the OTA type, now the type is Revert, flip the state of remap function, and clear the image_ok and the magic part of the flag.

Note: For the board using remap mode OTA, the firmware should contain flag operations. First one is writing magic part of the flag, this operation should be executed after the new firmware is written. The magic address is 0xFFFF0. Second operation is that clear the image_ok and magic part of the flag, after the firmware itself run the whole task period and during the period everything is ok, the firmware should clear these two parts of the flag.

7. FOTA

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

It provides the following OTA features:

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

7.1. Design

7.1.1. Single image mode of OTA

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

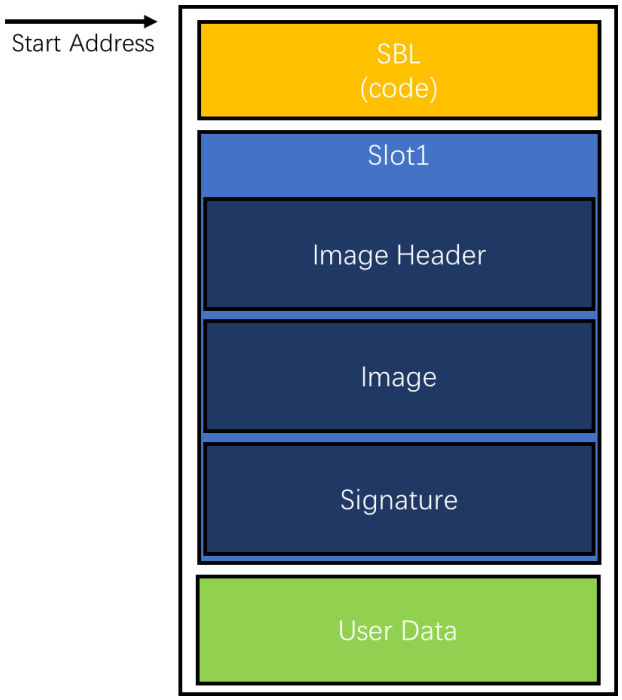


Figure 7-1 Flash layout for single image mode

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

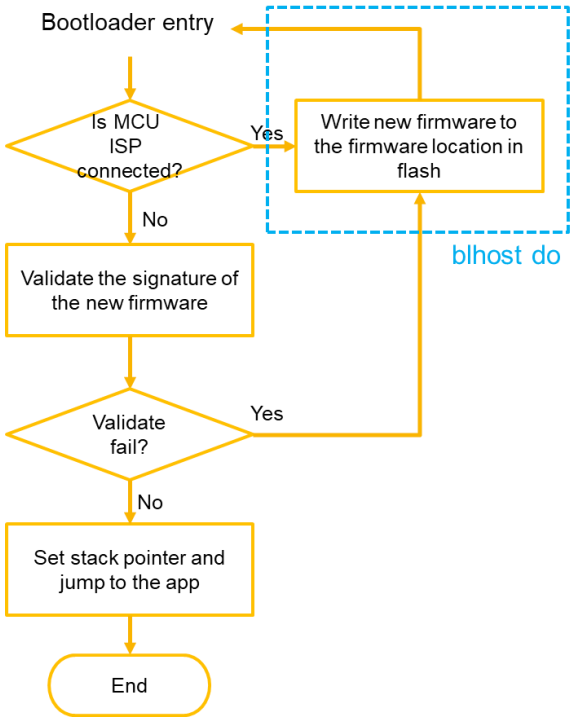
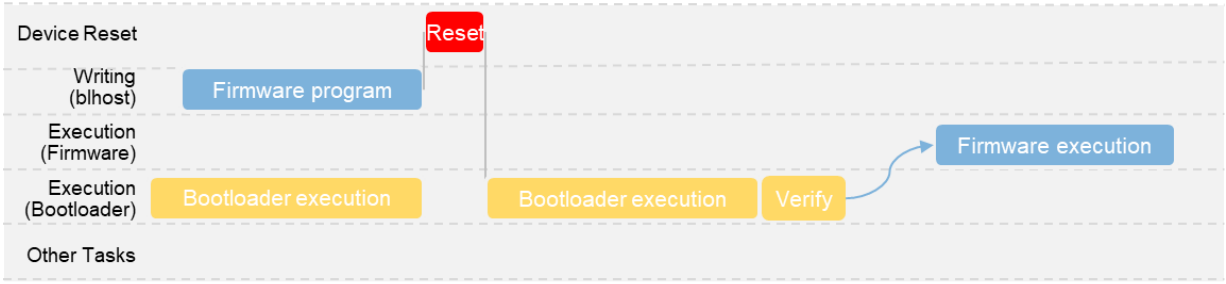


Figure 7-2 Bootloader workflow for single image mode

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

- **Single Image Boot**



t

Figure 7-3 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 7-1 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
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 7-2 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 7-4 Flash layout for swap mode

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

The workflow for swap mode of OTA is as below:

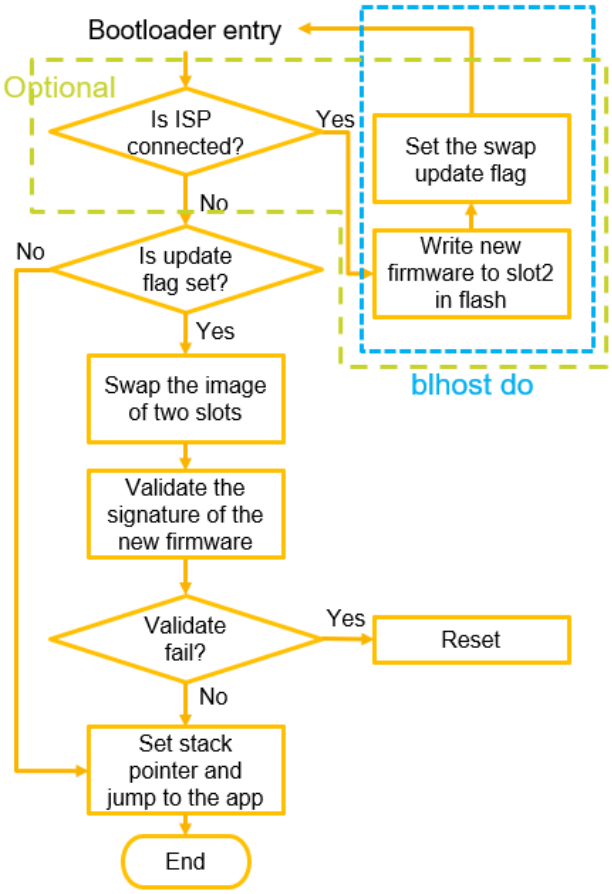


Figure 7-5 Bootloader workflow for swap mode

The period for swap mode of OTA is as below:

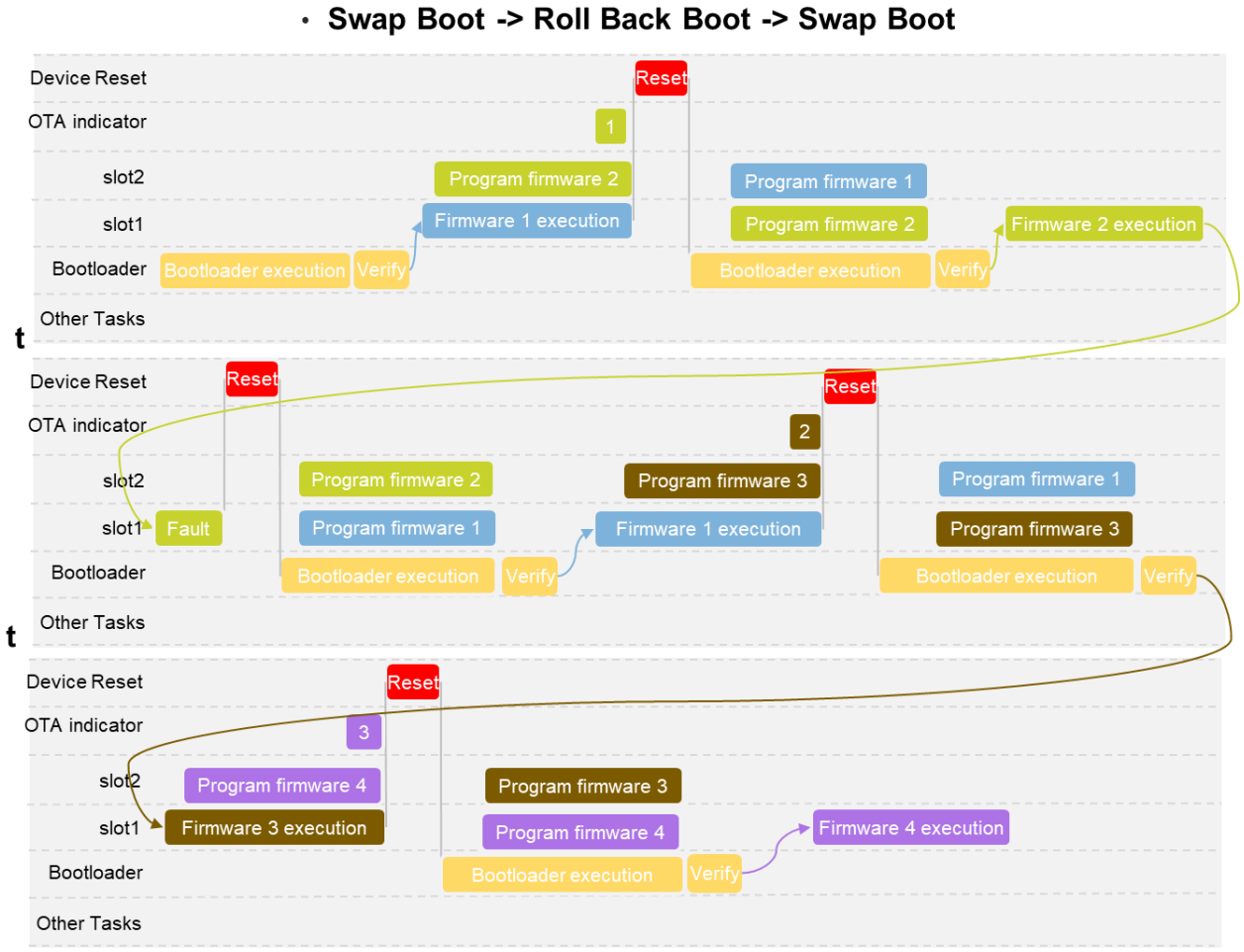


Figure 7-6 Update period of bootloader for swap mode

7.1.3. Remap mode of OTA

The OTA image of remap mode contain the image header, image data. The image header information is the same as swap mode, can refer to Table 7-1.

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

Table 7-3 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 7-7 Flash layout for remap mode

The workflow for remap mode of OTA is as below:

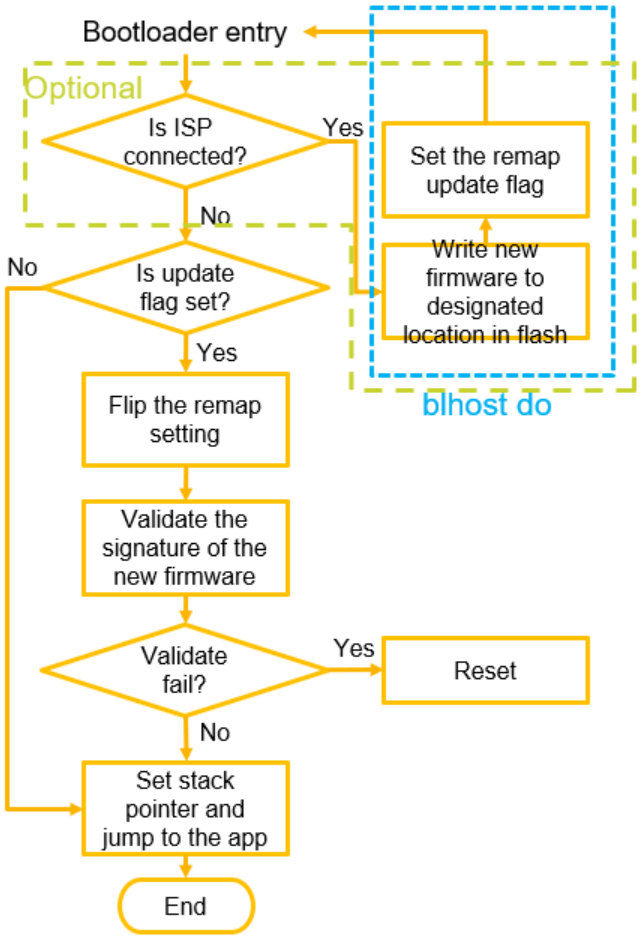


Figure 7-8 Bootloader workflow for remap mode

The period for remap mode of OTA is as below:

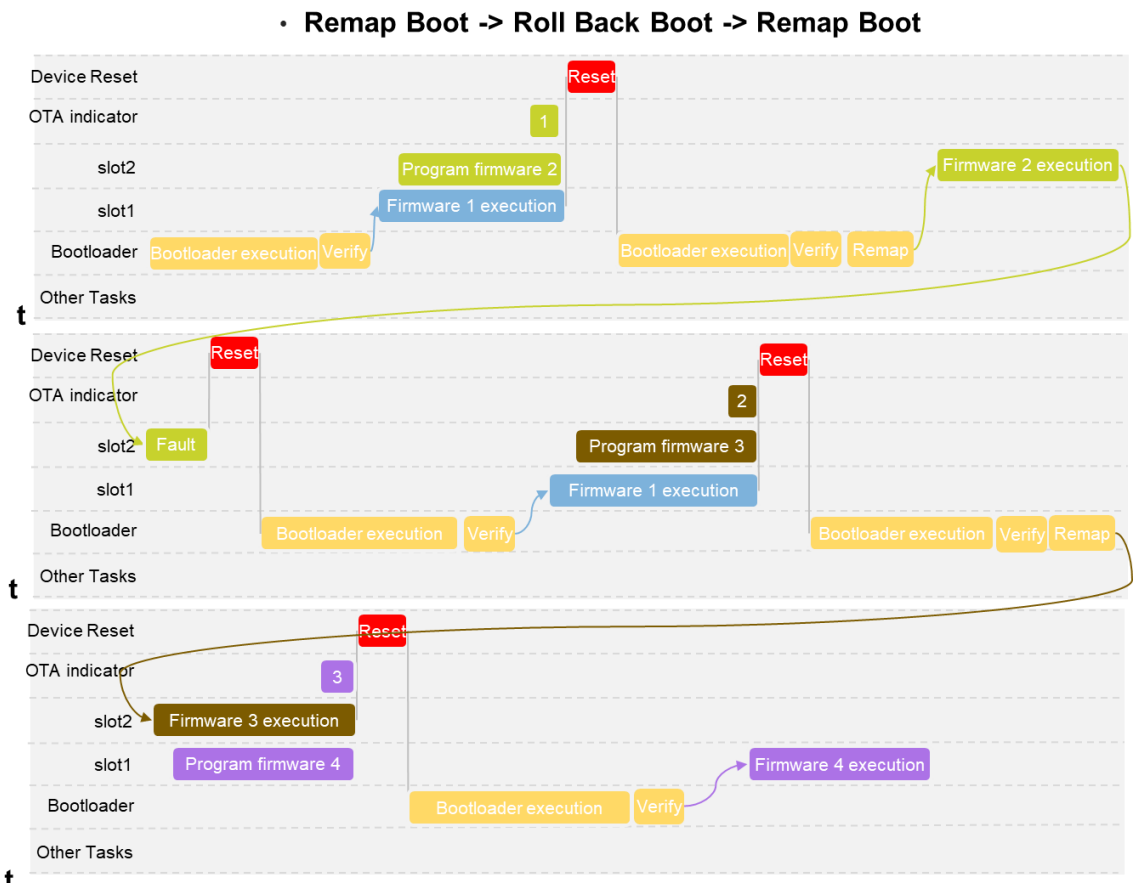


Figure 7-9 Update period of bootloader for remap mode

7.2. Local FOTA

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

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

1) Prepare the image

For single image mode:

Select the 'hello world' demo from SDK as an example. Firstly, in order to free up some space for adding header later, it is required to change the linker file. The default APP offset address is 0x100000, modify linker to adapt to this address, the IAR linker of EVKMIMXRT1060 on picture below can be used for reference.

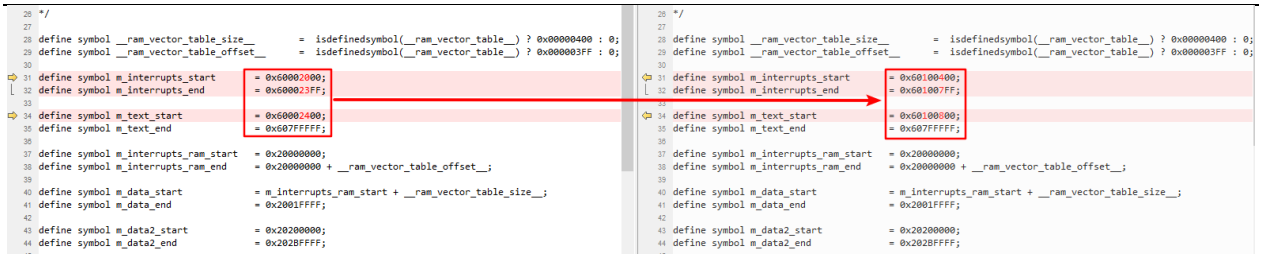


Figure 7-10 Linker file modification example

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 which may named *hello_world.bin*.

The SFW project already included above changes, so please 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 `scons --menuconfig` the configuration menu, **uncheck** the "Enable sfw standalone xip" option, and **check** the "OTA from sdcard" and "OTA from u-disk" options.

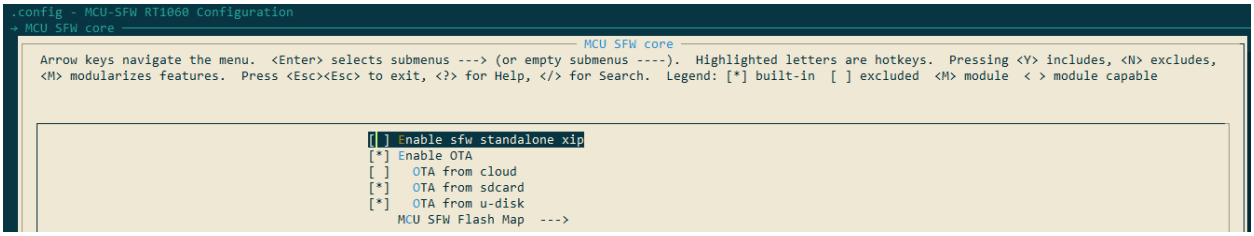


Figure 7-11 Configure the SFW

Following commands described in chapter 2 to generate project, then build the project and the images will be 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 using the script command to generate signature key: `python imgtool.py keygen -k xxxx_priv.pem -t rsa-2048-sign`

The *xxxx_priv.pem* will be used to sign the app 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, the *hello_world.bin* which generated in step1 can be used, for swap and remap mode, the image that SFW project generated can be used to run the test.

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 differ 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. And more information about the above sign commands are 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.

Note1: For EVKMIMXRT685, to enable the remap function and the single image function (which exist reset operation), must 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 can’t be powered down during the entire operation, otherwise the previous operation will be invalid. This limitation only exists on EVKMIMXRT685 FlexSPI port b. In order to ensure that the entire operation is not powered down, after the image is downloaded to the flash, please reset the MCU by SW3 instead of power-down reset. In addition, if the DAPLink is used to download images, please make JP2(PIN1-2) short, then remove the J-Link probe.

Note2: For EVKMIMXRT595 / EVKMIMXRT685 U-disk update function, please power on the EXT PWR port, otherwise, the power supply current is not enough, which will cause the U-disk update to fail.
Note3: 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, the board needs rework first.

7.2.1. Single image OTA

1) Configure the SBL

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

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

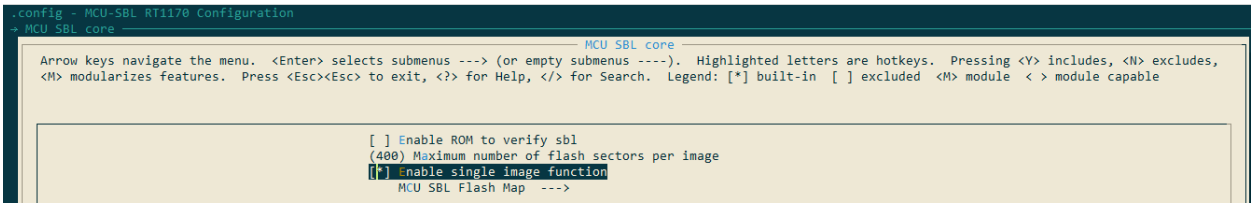


Figure 7-12 Enable single image

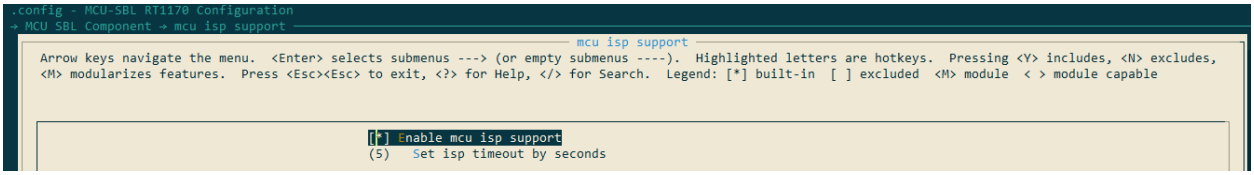


Figure 7-13 Enable mcu isp

Generate Project, compile and download the SBL to the target board, connect the UART or USB port to the PC, 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 should directly connect the UART port to PC by using TTL2USB module, please refer to the schematic of EVK board.

After reset the board, the board will wait for 5 seconds, during the 5 seconds, if the PC send the connect command(blhost send the **get-property** command and get success in terminal), the board will run into MCU ISP mode, if not, the board will run into boot mode.

On PC side, put the image which will be 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 5s after the board is powered on. `blhost -u -- get-property 1 0`

After connecting successfully, can 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: When using erase operation to erase the flash, if the size is very large, please 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.

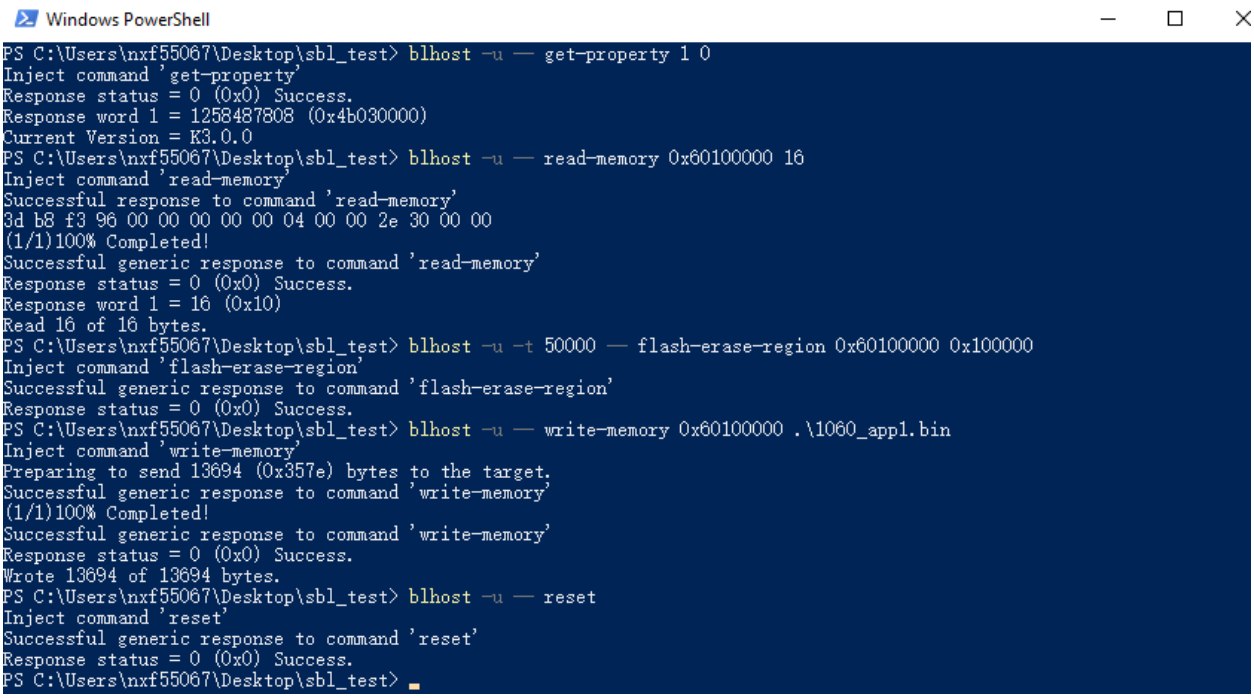


Figure 7-14 Blhost operation on PC

Write the new image to the flash slot and then type the reset command to reset the board, Enter the boot process after 5s waiting. `blhost -u -- reset`

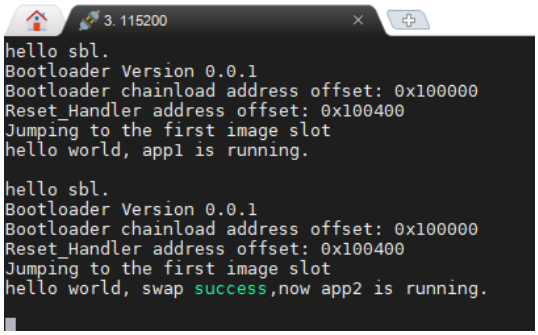


Figure 7-15 Single image updating log

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, should connect R136 on the REV C EVK board.

1) Prepare the SBL

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

2) Download the first image

For the first time to test the SBL, it is required to download the first image to the board to run the test. Using the *MCUBootUtility* tool to download the ***app1.bin*** to the first slot of the board, the default location of the slot1 is the **flash_offset+0x100000** to **flash_offset+0x200000**, the whole slot size is 1MB.

Note: If MCU ISP function is not enabled, please set the MCUBootUtility to master mode.

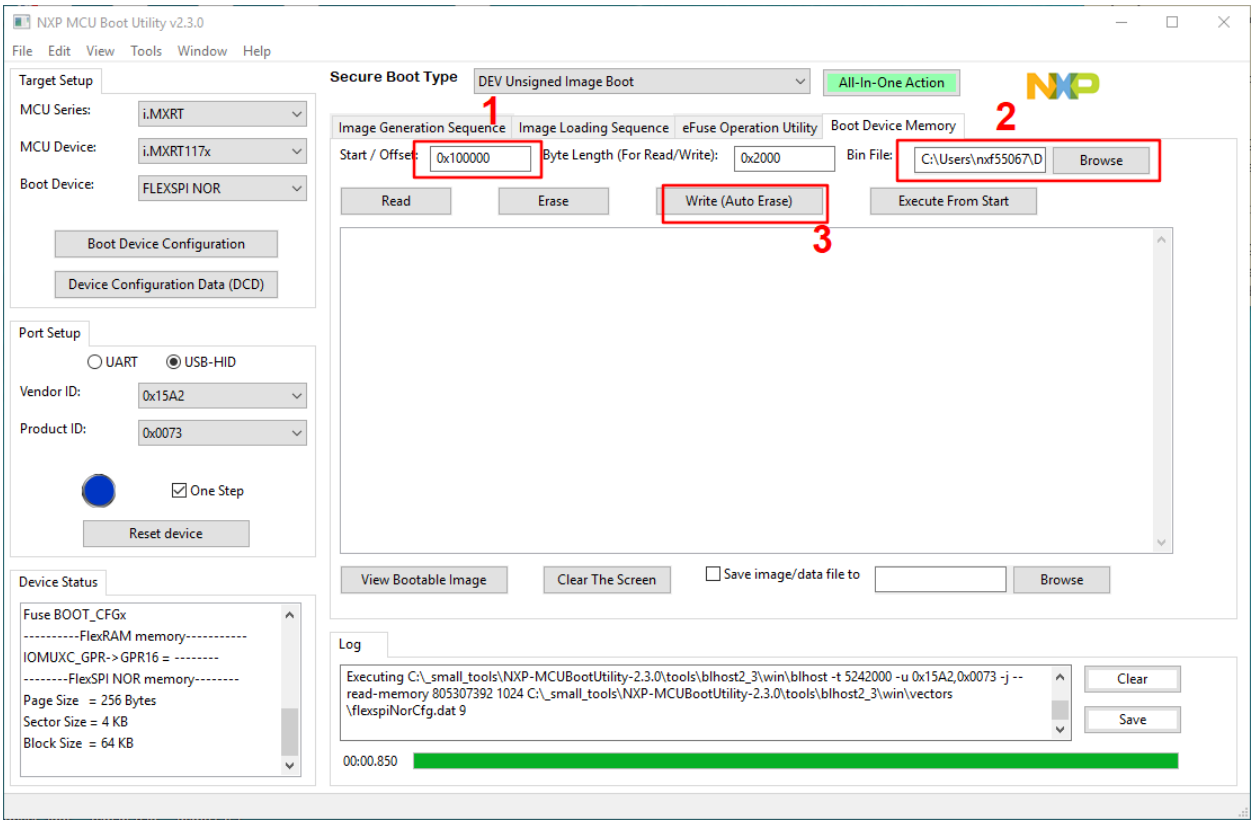


Figure 7-16 Download the image1 to the slot1

3) Run the test

Insert a SD card to the PC, and copy the *app2.bin* to the SD card, then rename the file name to ***newapp.bin.*** Remove the SD card and insert it to the target board, then the debug console will print the updating log, after the SD card downloading finished, when the log '**sys rst…**' printed, remove the SD card from the board, otherwise, it will start a new updating.

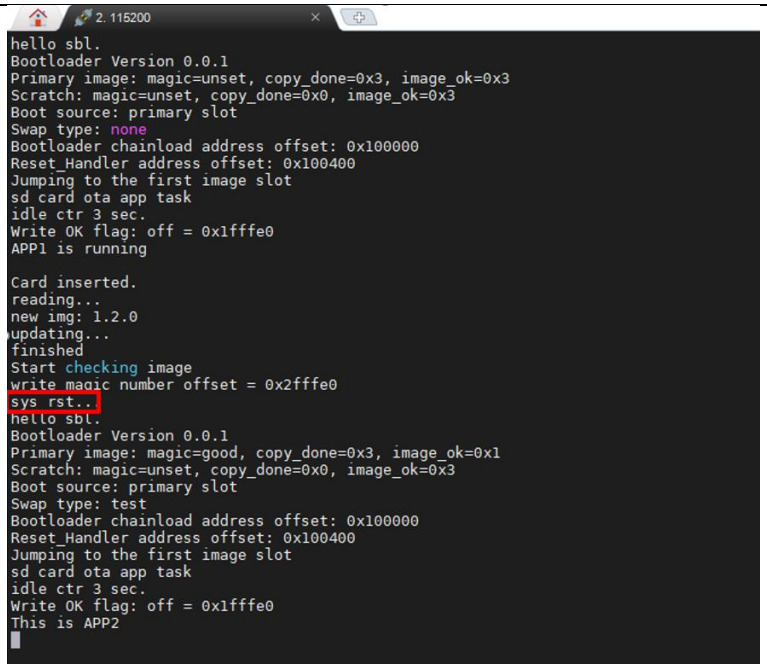


Figure 7-17 Swap updating log

After the app2 log 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

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.

2) Download the first image and the remap image flag

For the first time to test the SBL, it is required to download the first image to the board to run the test. Using the *MCUBootUtility* tool to download the ***app1.bin*** to the first slot of the board, the default location of the slot1 is the **flash_offset+0x100000** to **flash_offset+0x200000**, the whole slot size is 1MB.

3) Run the test

Connect a U-Disk to the PC, and copy the ***app2.bin*** to the U-Disk, and 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 have two usb port, connect it to USB1 port), then the debug console will print the updating log. after the image downloading finished, when the log '**sys rst…**' printed, remove the U-Disk from the board, otherwise, it will start a new updating.

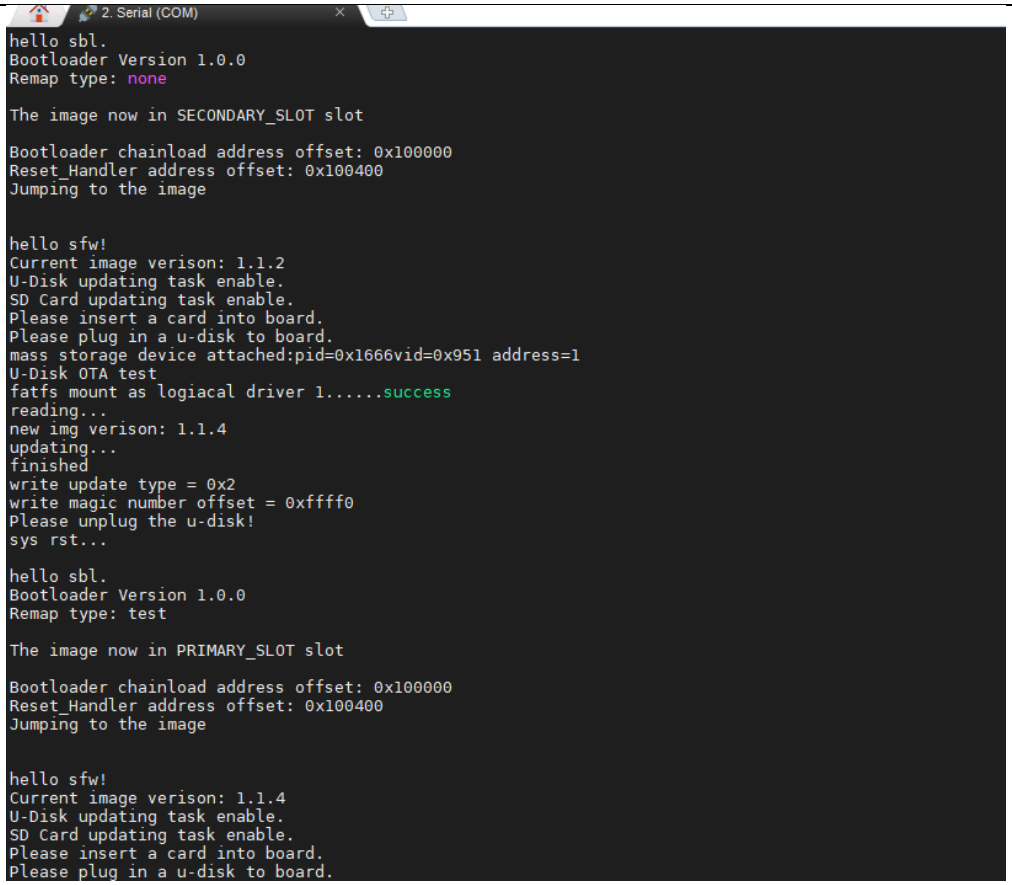


Figure 7-18 Remap updating log

After the app2 log printed, push the reset button on the board to confirm that the updating successful, the app2 log should be printed.

7.3. Remote FOTA

7.3.1. AWS OTA

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

7.3.1.1. AWS OTA Pre-Requisites

- Create AWS Account

Create 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**.

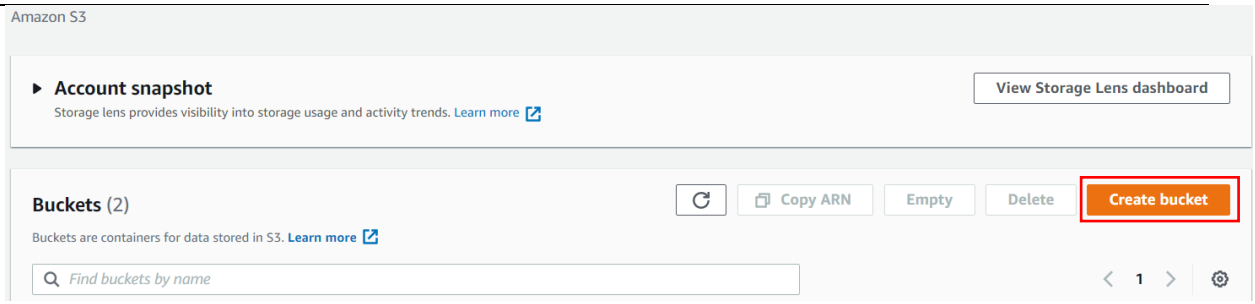


Figure 7-19 Create bucket

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

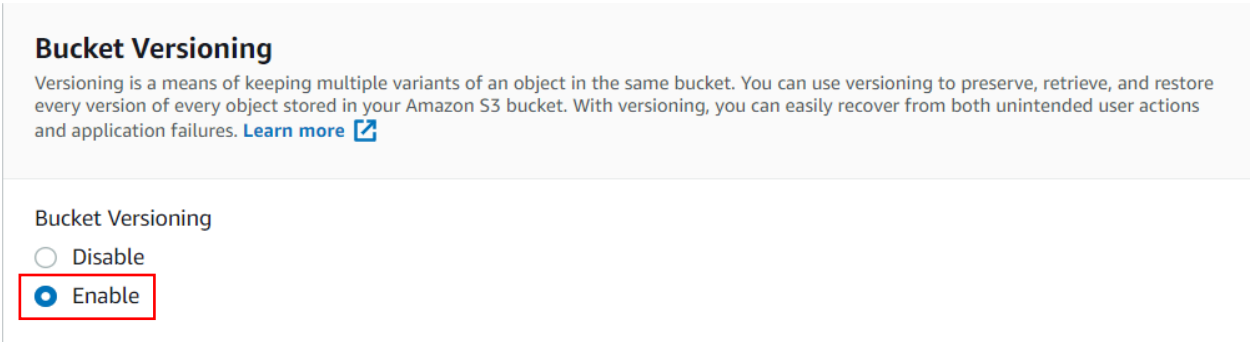


Figure 7-20 Bucket versioning

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**.

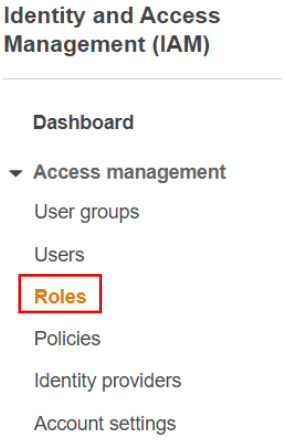


Figure 7-21 Roles

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

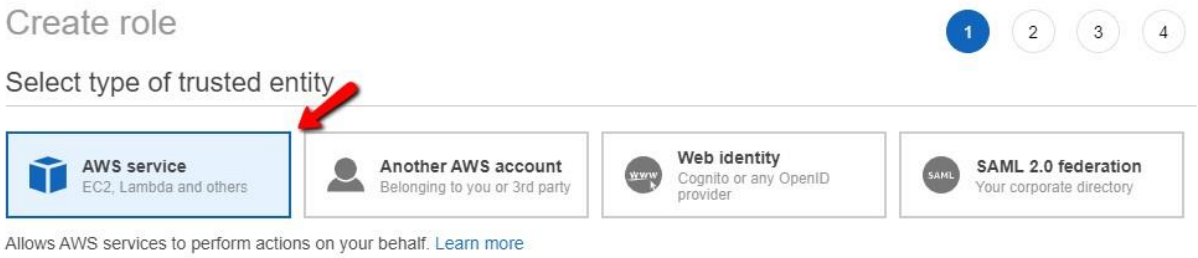


Figure 7-22 AWS service

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

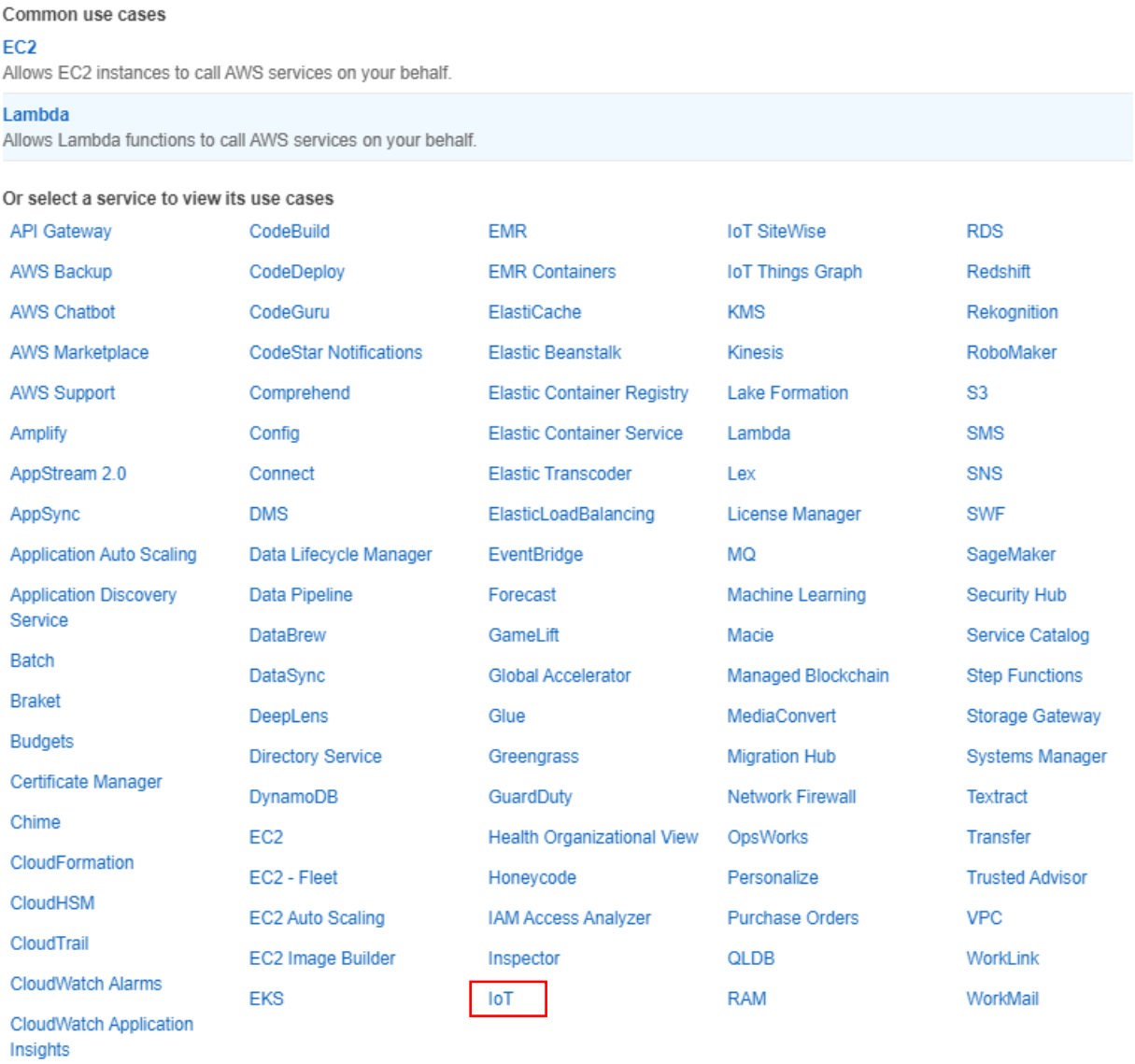


Figure 7-23 IoT service

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

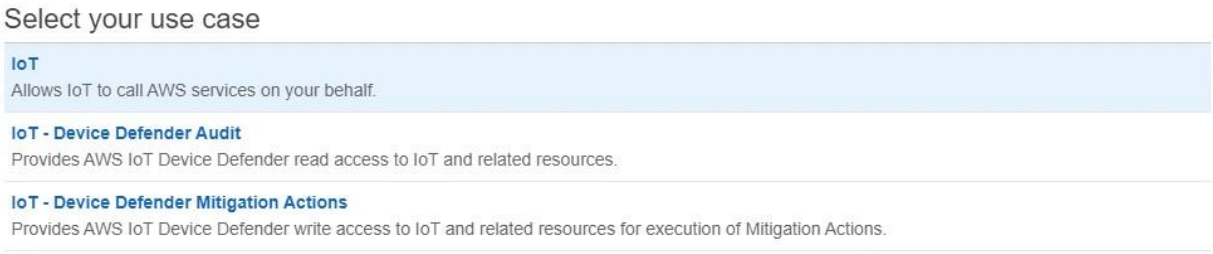


Figure 7-24 IoT use case

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**.

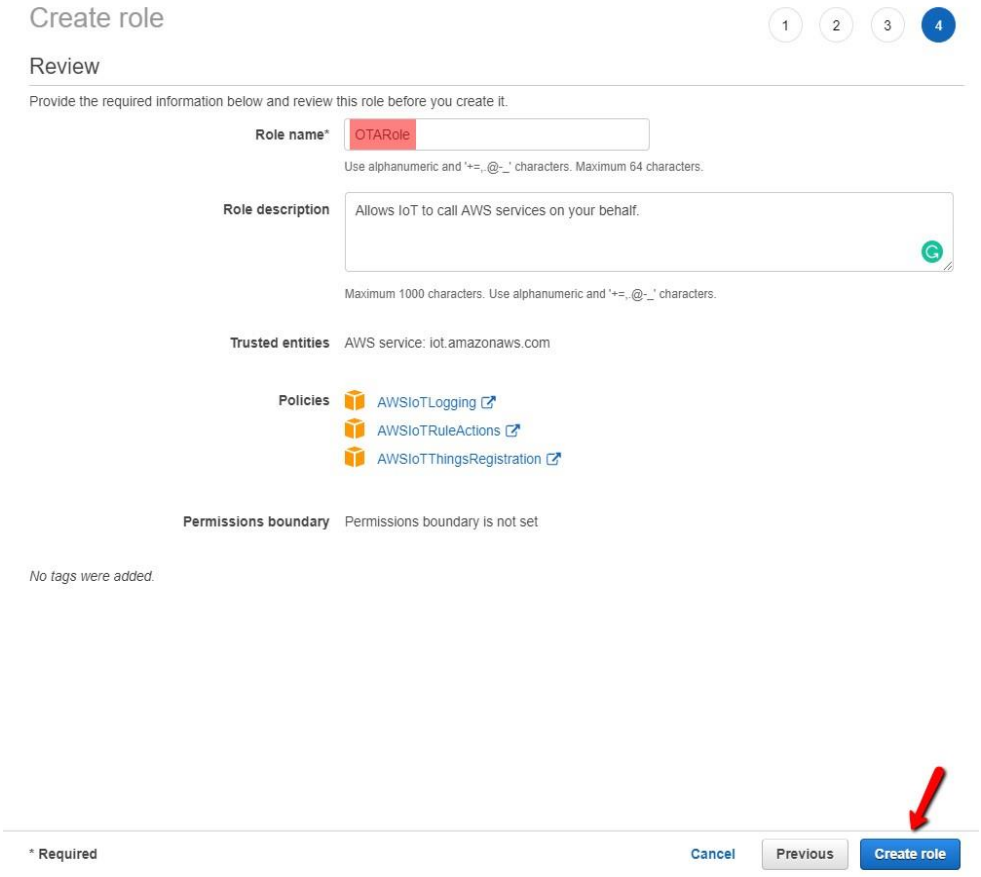


Figure 7-25 Create role

- 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.

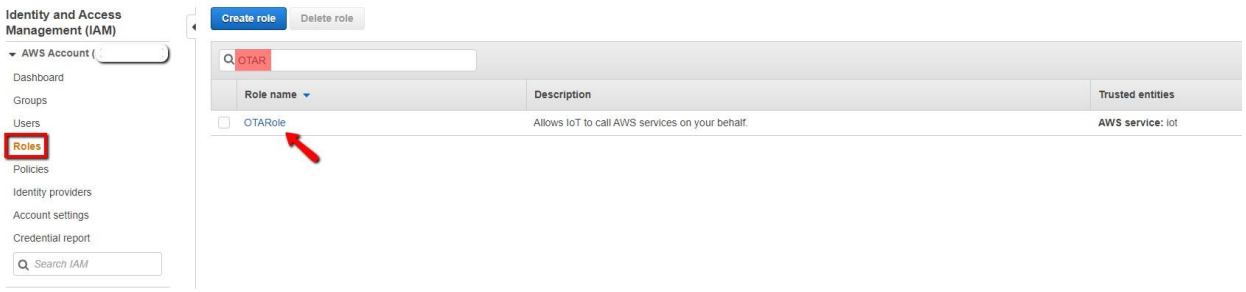


Figure 7-26 Role search

2. Choose **Attach policies**.

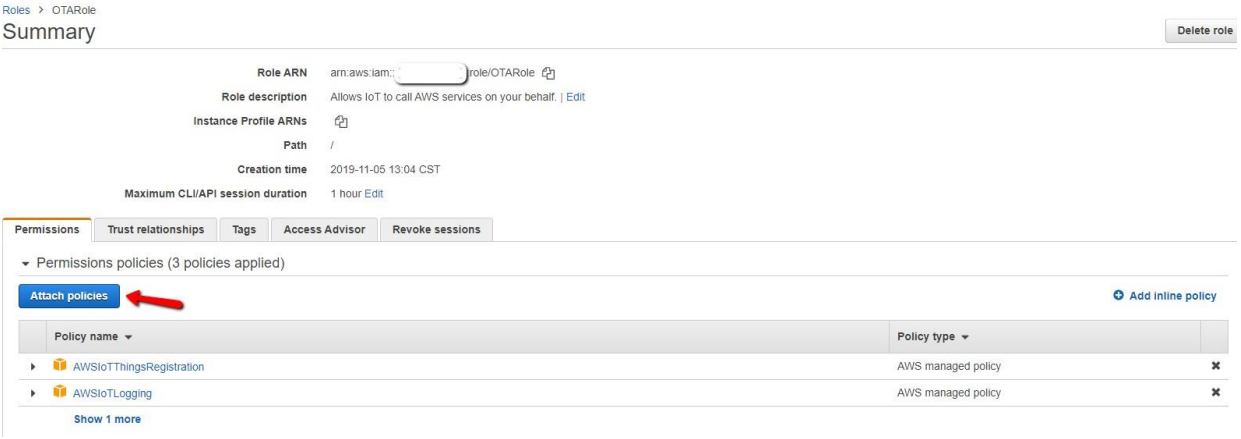


Figure 7-27 Attach policies

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

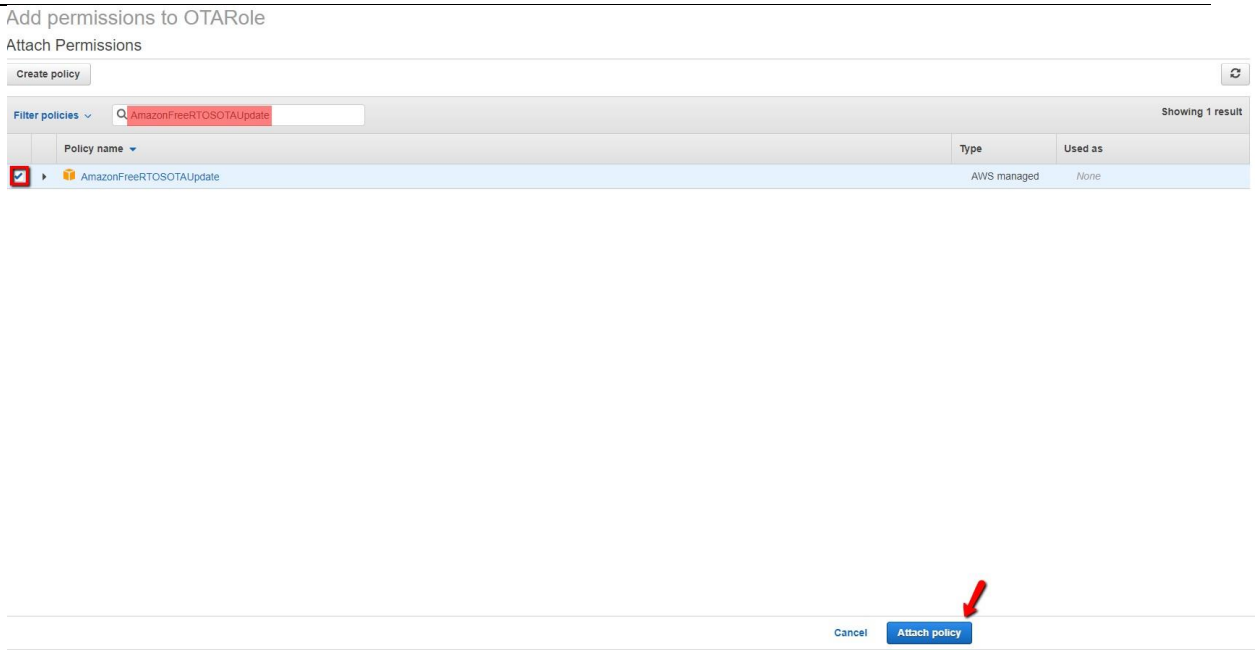


Figure 7-28 Attach AmazonFreeRTOSOTAUpdate policy

- Add the required IAM permissions to the OTA service role

1. Choose **Add inline policy**.

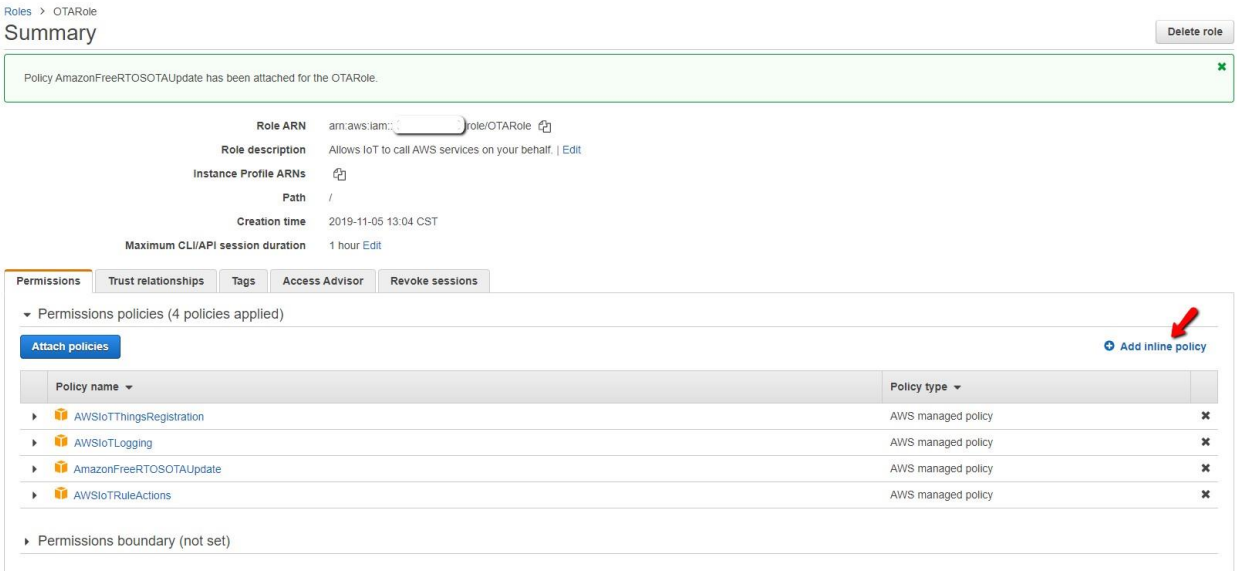


Figure 7-29 Add inline policy

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**.

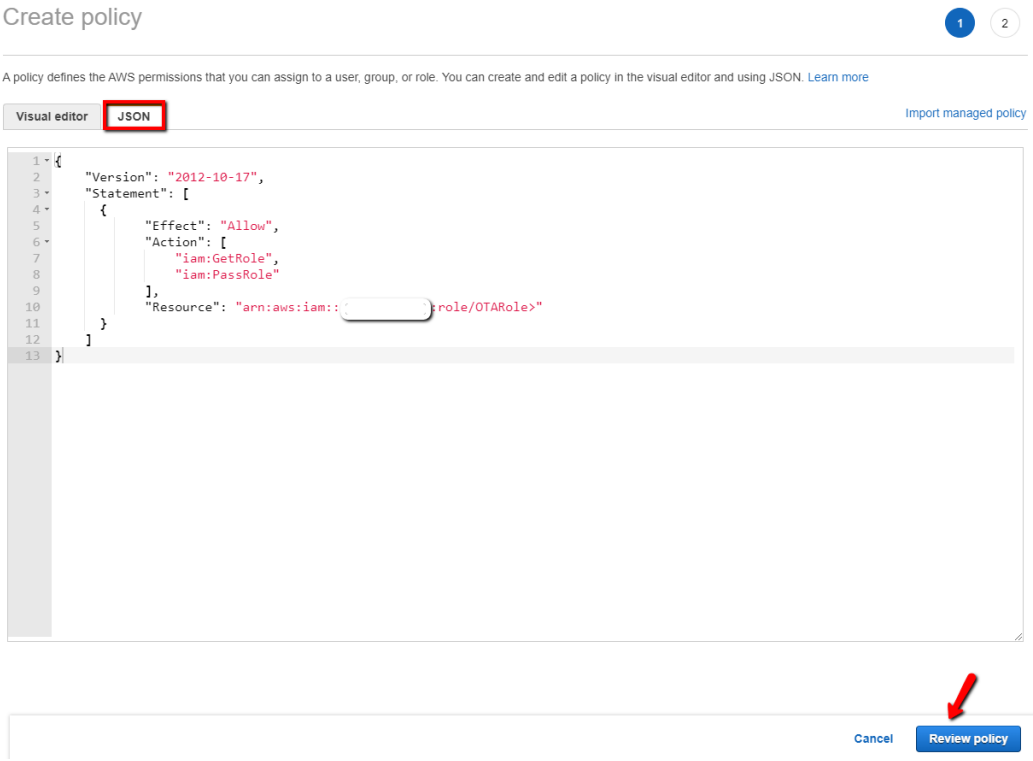


Figure 7-30 Review policy

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

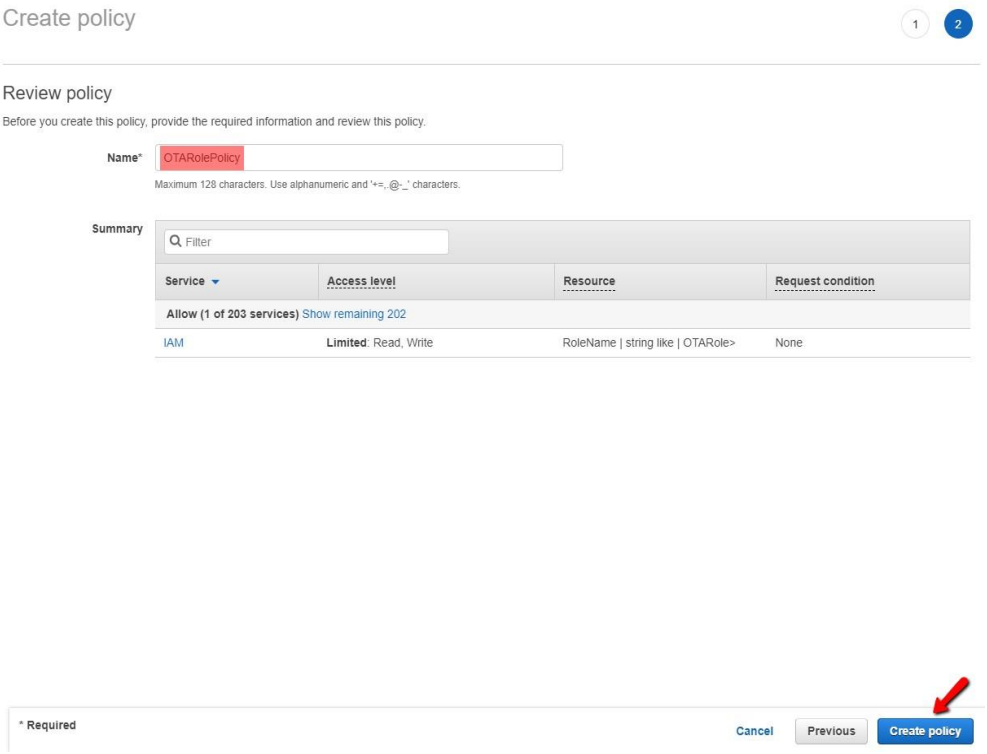


Figure 7-31 Create policy

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

1. Choose **Add inline policy**.

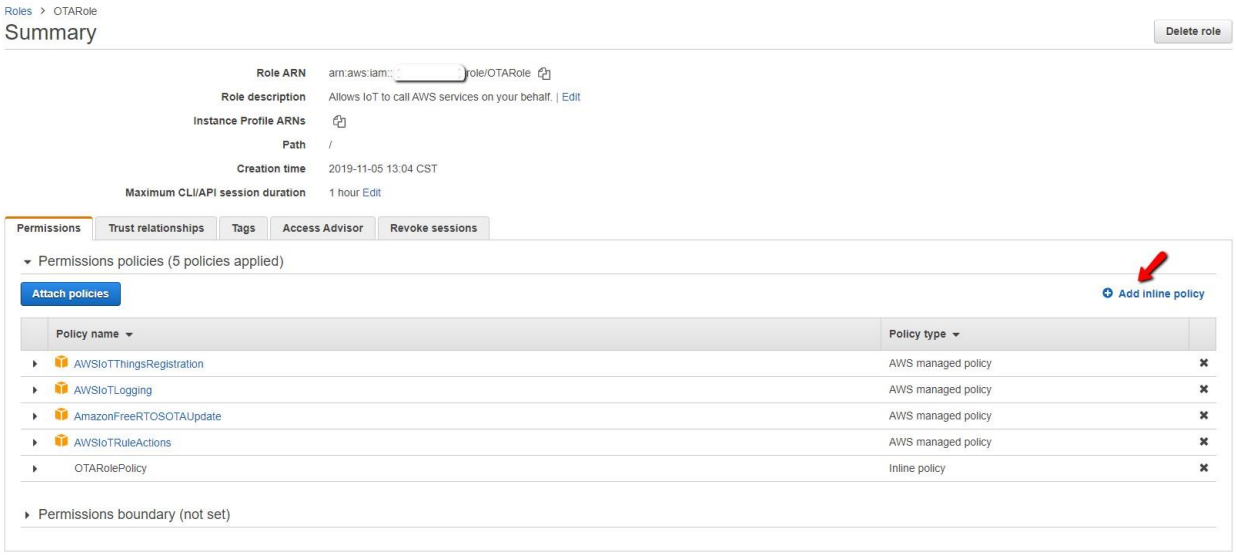


Figure 7-32 Add inline policy

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 <*examplebucket*> with the name of the bucket.

4. Choose **Review policy**.

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

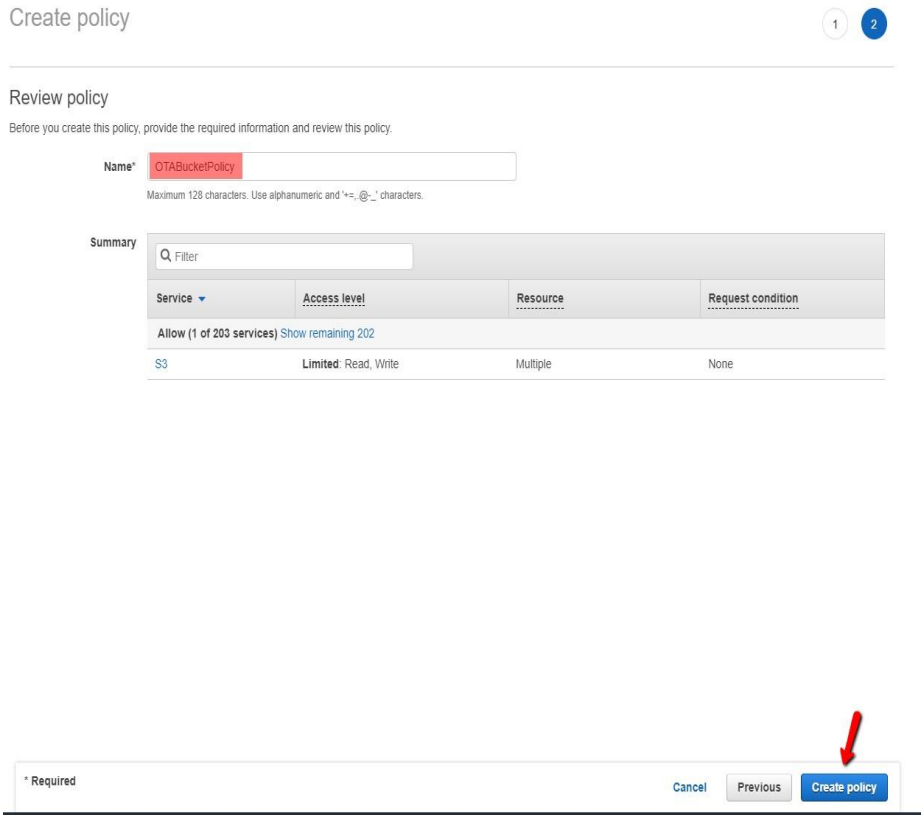


Figure 7-33 OTA bucket spolicy

- Create an OTA User Policy

1. Open the 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**.

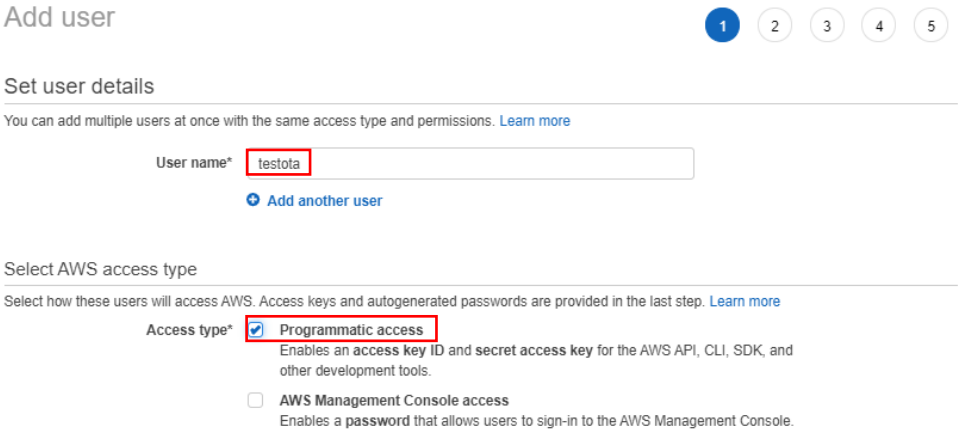


Figure 7-34 Add user

5. Choose **Next: Tags**.

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

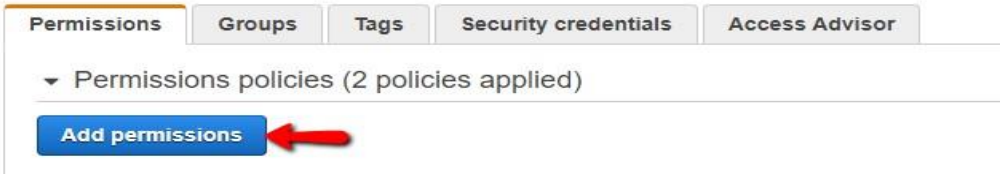


Figure 7-35 Add permissions

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

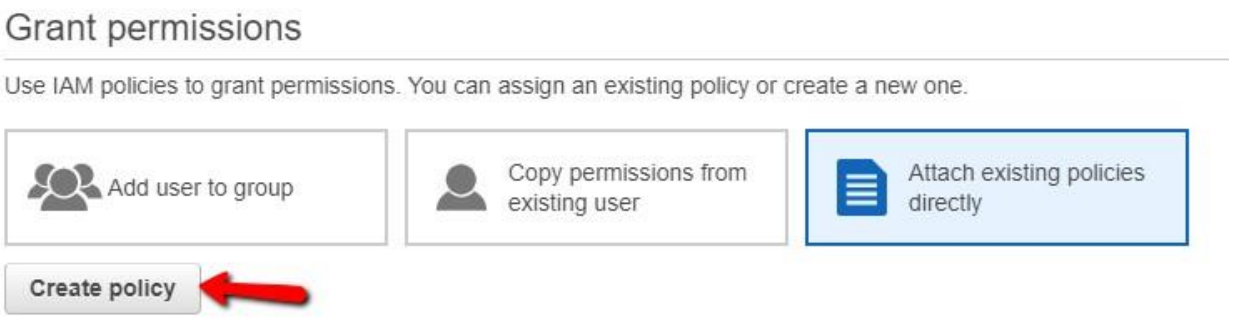


Figure 7-36 Create policy

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 IAM service role that just created.

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

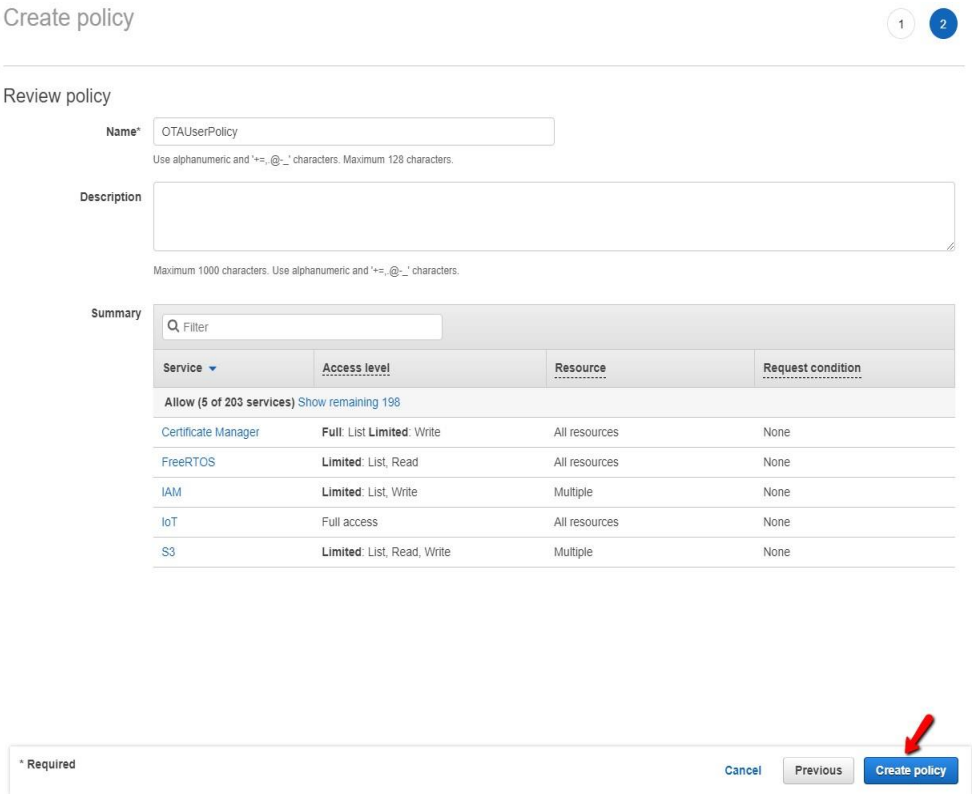


Figure 7-37 OTA user policy

- 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

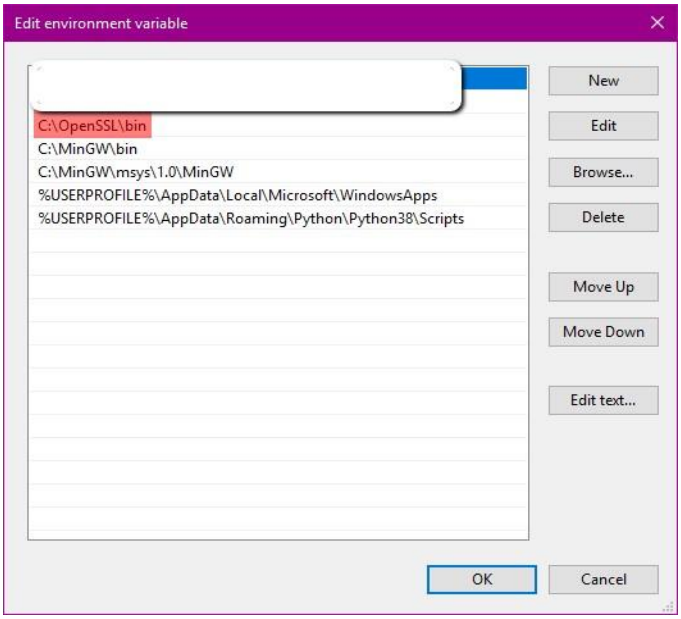


Figure 7-38 OpenSSL config

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**.

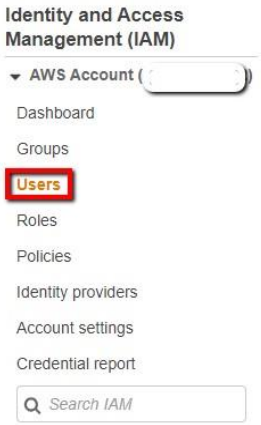


Figure 7-39 Users

   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 will look something like this:

Access key ID: AKIAIOSFODNN7EXAMPLE

Secret access key:wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY

h) To download the key pair, choose **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 in order 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 will ever ask someone for the secret key.

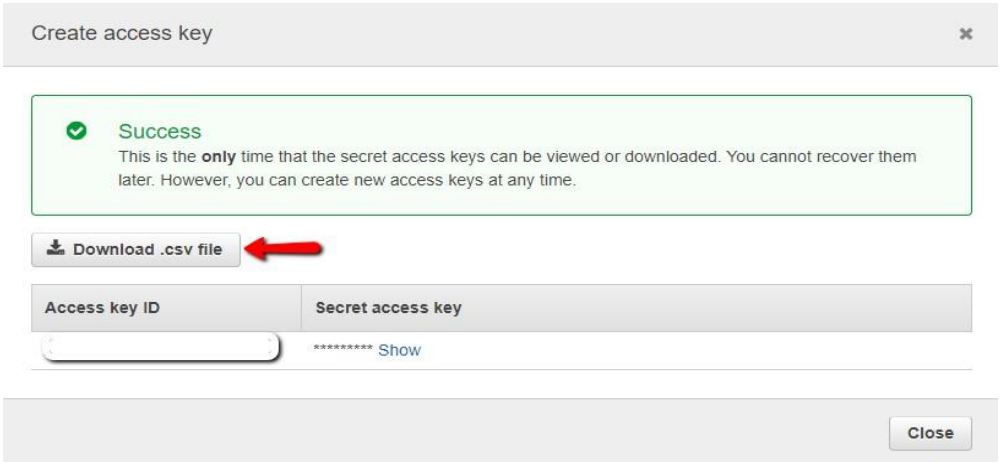


Figure 7-40 Download .csv file

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

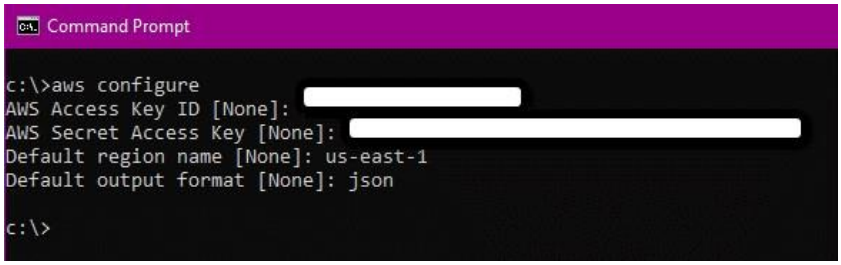


Figure 7-41 AWS CLI installation

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 an ARN for the certificate. Save it locally to use it while creating the OTA update job.

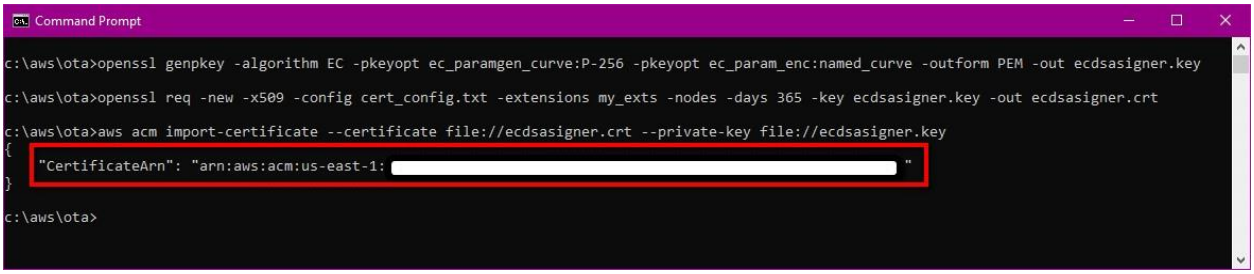


Figure 7-42 Creating a Code-Signing Certificate

- 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 7-43 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**.

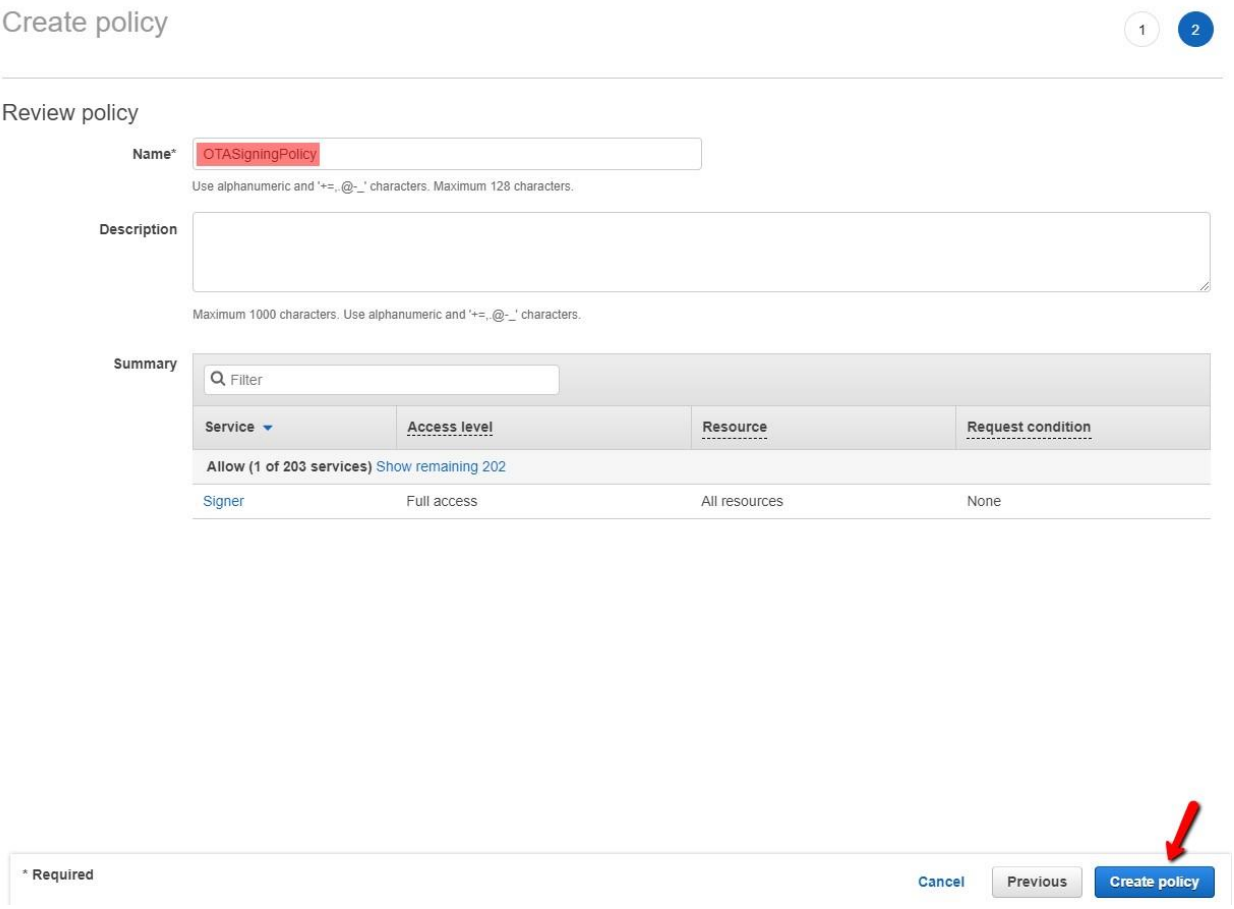


Figure 7-44 OTA signing policy

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 then select the check box next to the code-signing policy created before.

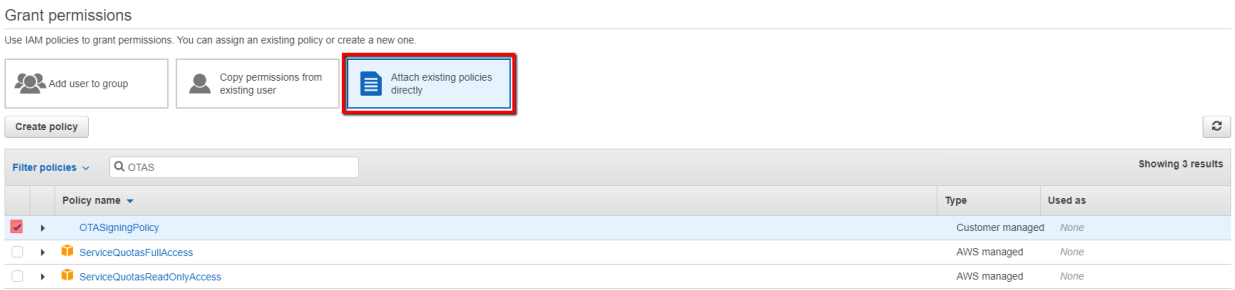


Figure 7-45 Attach policy

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, please first check if the region is the one which the bucket is locate.

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

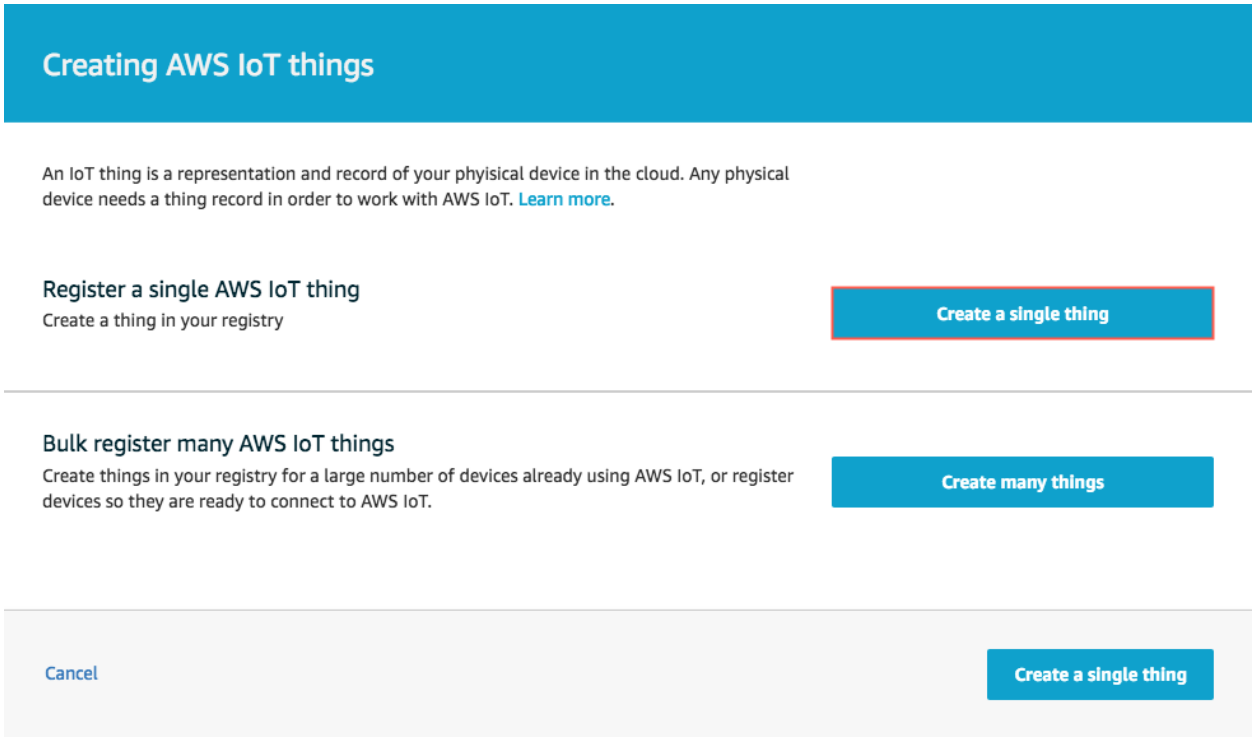


Figure 7-46 Create a single thing

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

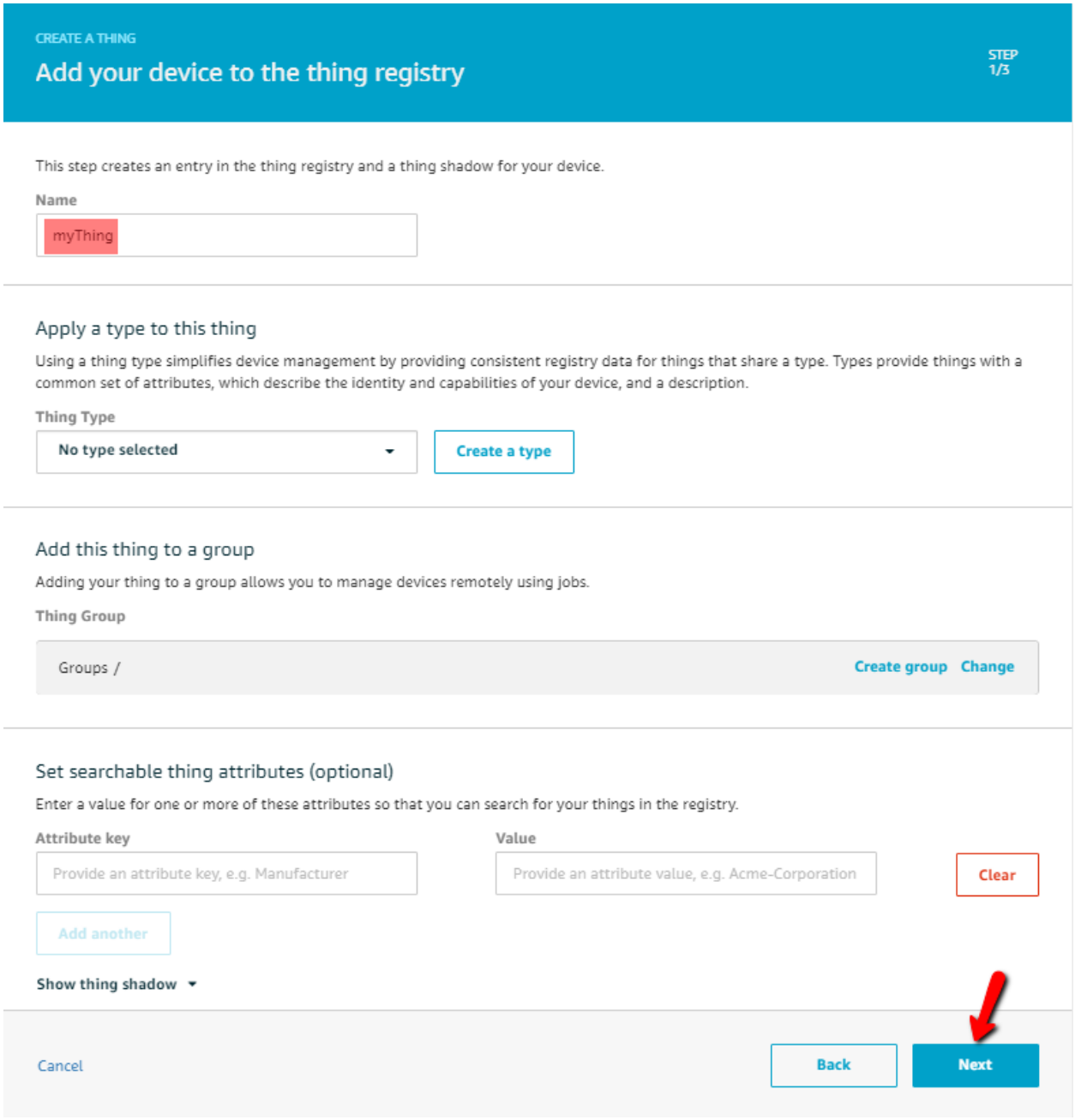


Figure 7-47 Thing name

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

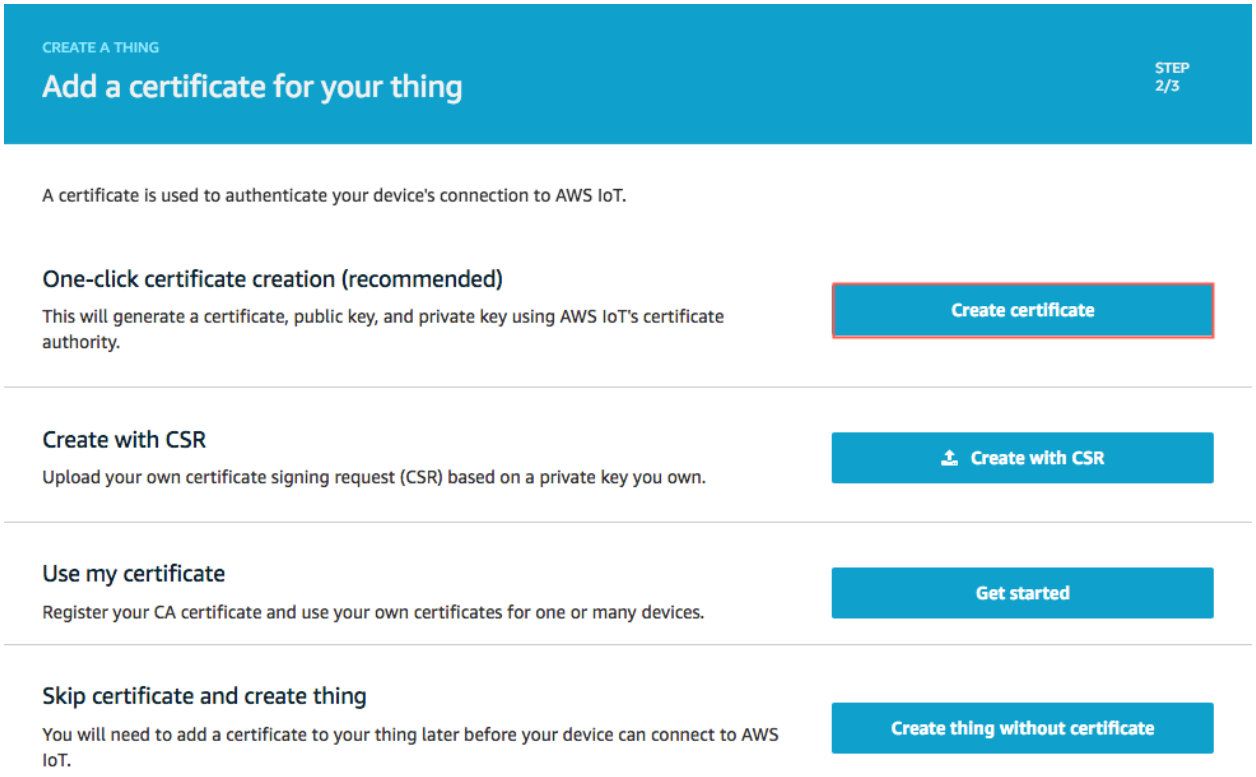


Figure 7-48 Create certificate

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**. This 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**.

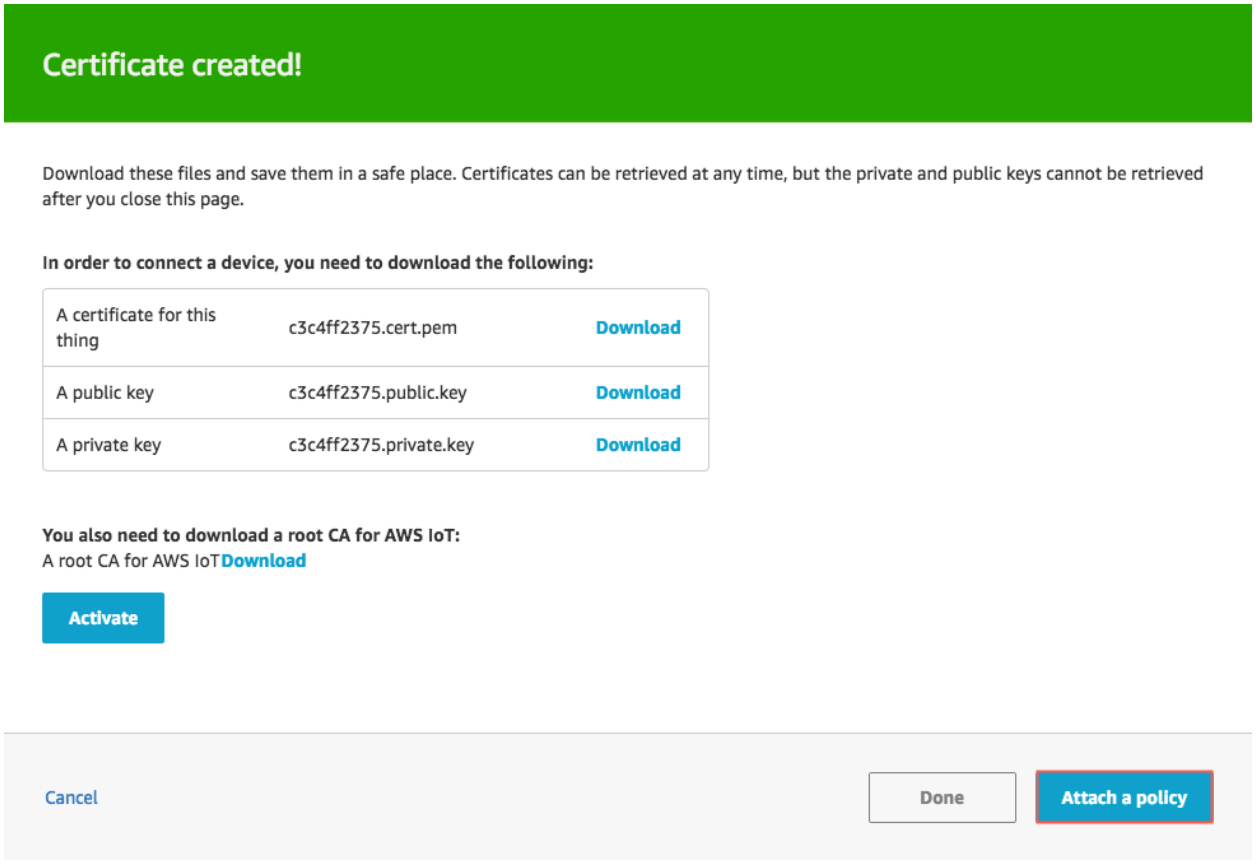


Figure 7-49 Files download

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**, and then 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** check box. This allows all clients to connect to AWS IoT. After entering the information for the policy, choose **Create**.

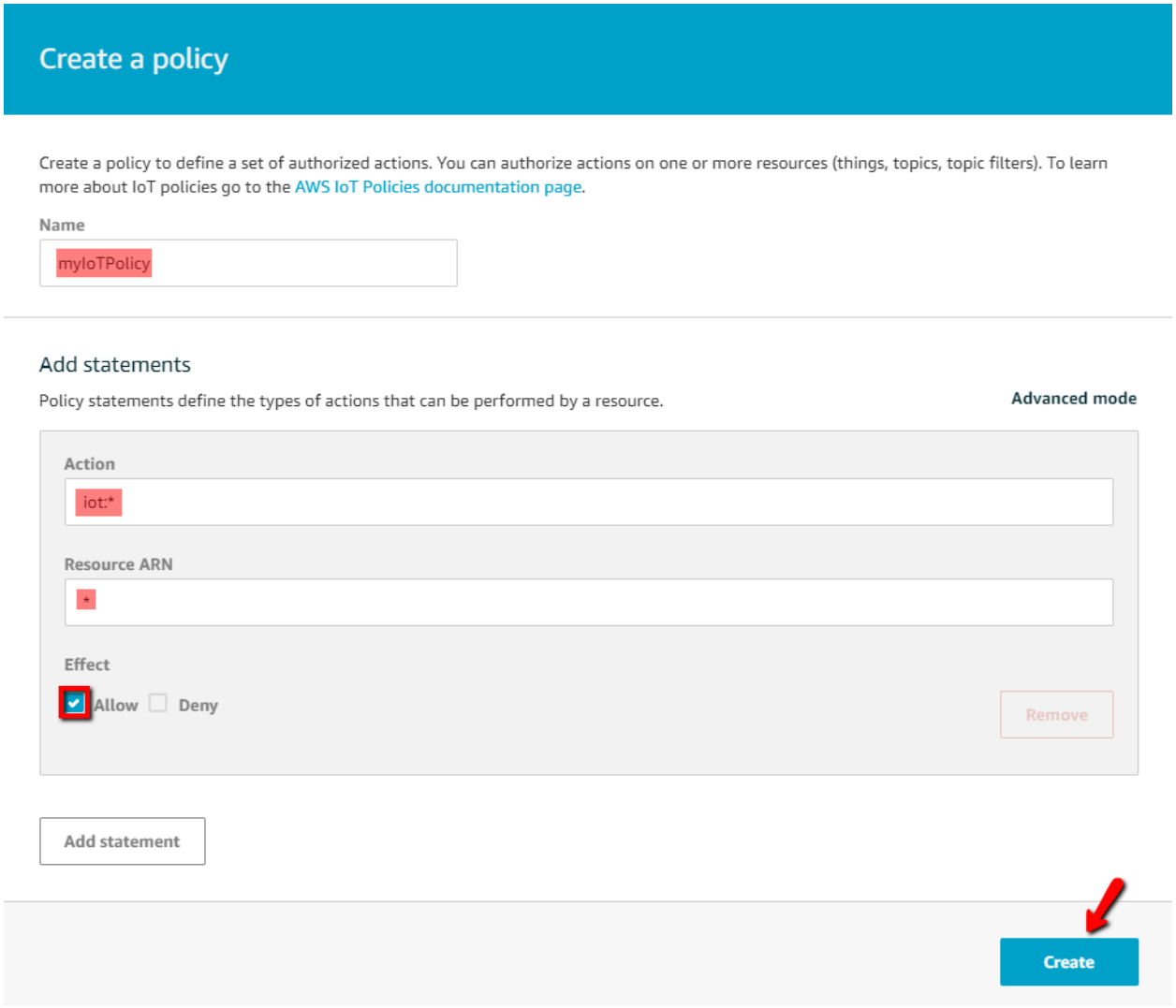


Figure 7-50 Create IoT policy

- 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 check box next to the policy created in the previous step, and then choose **Attach**.

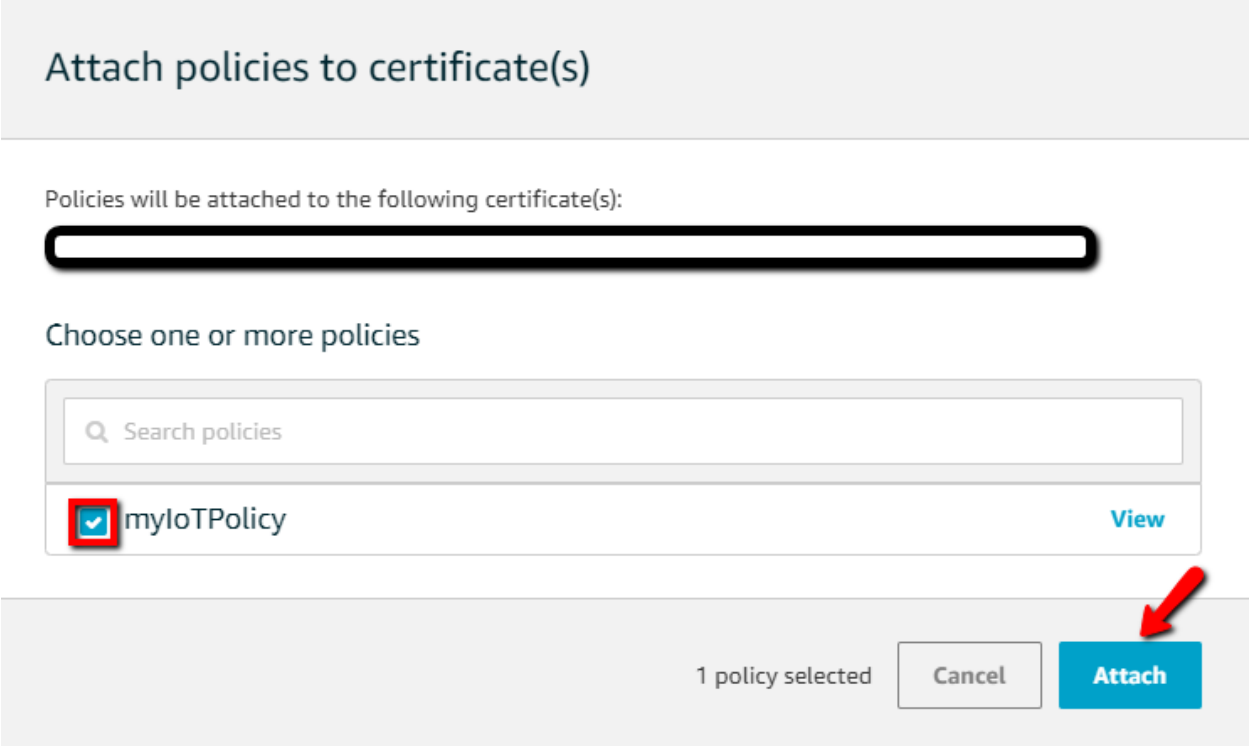


Figure 7-51 Attach IoT policy

- 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 check box next to the thing registered, and then choose **Attach**.

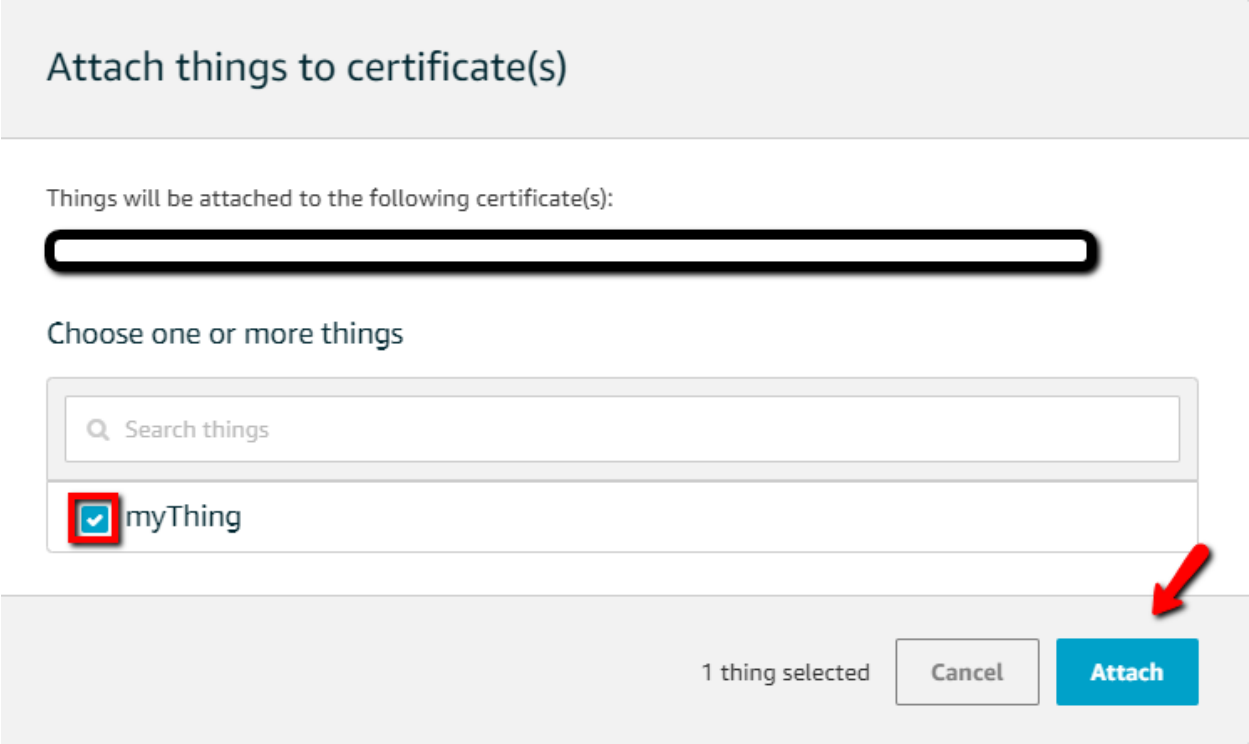


Figure 7-52 Attach Thing

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

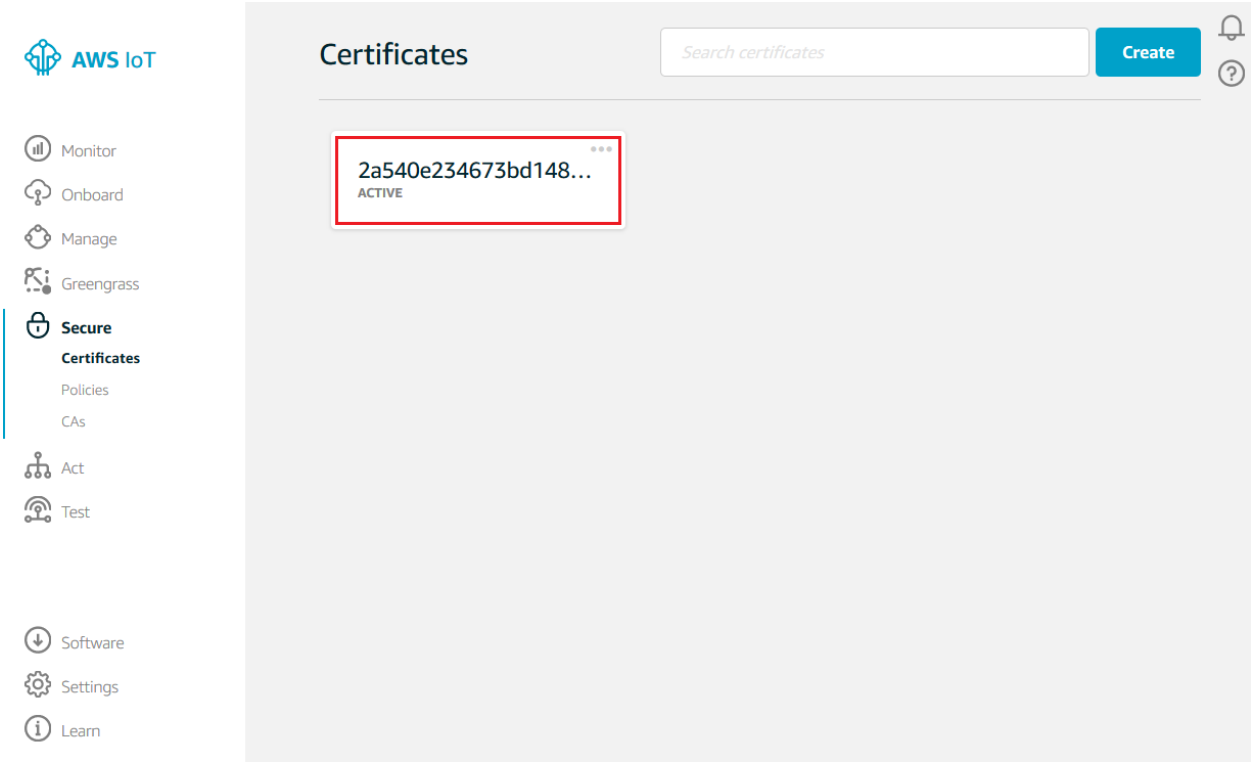


Figure 7-53 Select certificate to verify

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

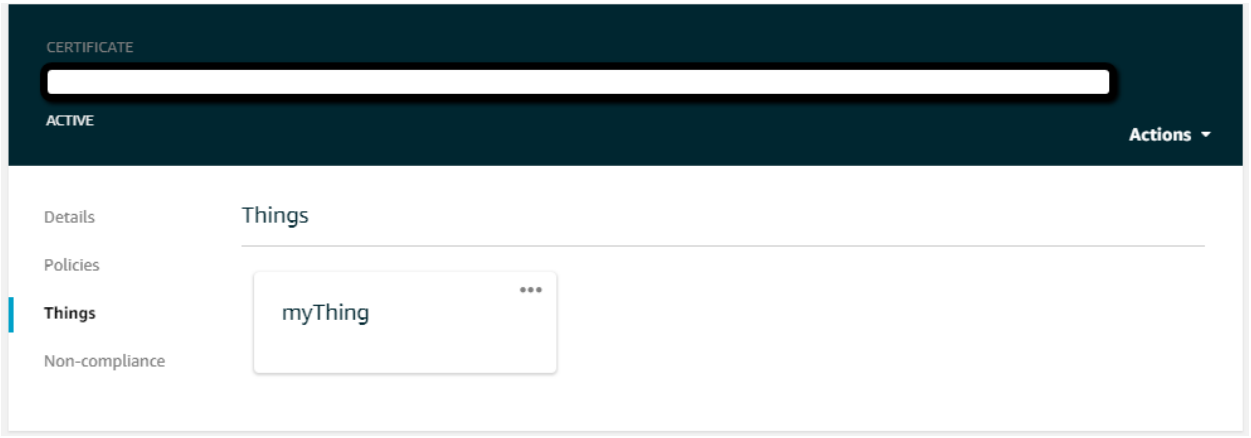


Figure 7-54 Verify attached thing

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

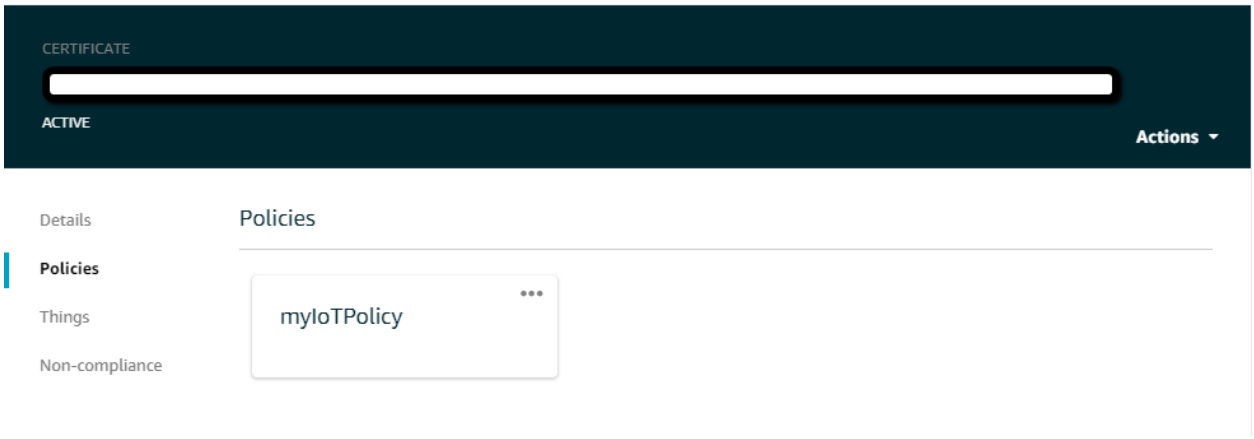


Figure 7-55 Verify attached policy

7.3.1.2. Prepare the SBL

In SBL project, enter *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, please also modify the *sign-rsa2048-pub.c*
2. Programming SBL image by drag-drop of DAPLink may erase whole flash.

7.3.1.3. Prepare the SFW

In SFW project, follow below steps to prepare SFW config:

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 previously downloaded from the Thing in '**Create an AWS IoT Thing'** part of section 7.3.1.1 and click on Generate and save aws_clientcredential_keys.h

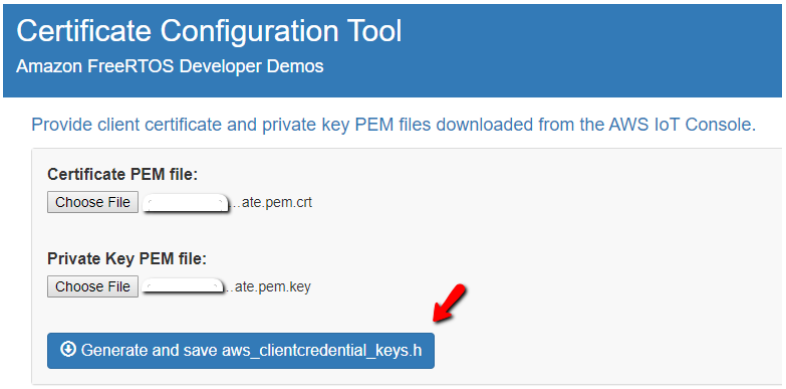


Figure 7-56 Generate aws_clientcredential_keys.h

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

2. Modify 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"
"MBMGByqGSM49AgEGCCqGSM49AwEHA0IABGUghBD51mF1J3wf4LYsQ2VgOaDpg98G\n"
"dNC38FWGS7owT4NC5848JumrD8SonnnXpu77Pt7ShuW39hC3Vdi7zlGjJDAiMAsG\n"
"AlUdDwQEAwIHgDATBgNVHSUEDDAKBggrBgEFBQcDAzAKBggqhkjOPQQDAgNIADBF\n"
"AiEAr0pNzlaMax4arCPNiW9HYFdQTvUGyZdRLcDrUol/LQoCIH2U2REoZ59V7r6z\n"
"CMLfHA+kWq84IjxDUE2OgV60RVvC\n"
"-----END CERTIFICATE-----\n";
```

Figure 7-57 Certificate format

3. Enter *sfw/target/evkmimxrt1170* path.
4. Double click the batch file *env.bat*
5. Input `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.

```
[ ] Enable sfw standalone xip
[*] Enable OTA
[*]   OTA from cloud
[]      OTA Cloud Select (AWS)  --->
        AWS Config  --->
[*]   OTA from sdcard
[*]   OTA from u-disk
    MCU SFW Flash Map  --->
```

Figure 7-58 SFW config

8. Select **AWS Config** to enter below config menu.

```
()  Set MQTT broker DNS name (NEW)
()  Set IoT Thing name (NEW)
```

Figure 7-59 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** select the previously created Thing.

c) In the navigation pane, choose **Interact**

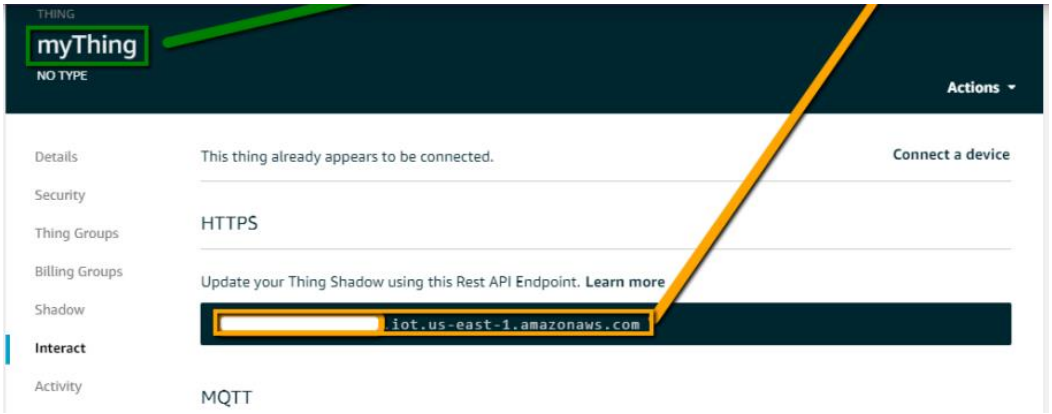


Figure 7-60 Interact of Thing

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

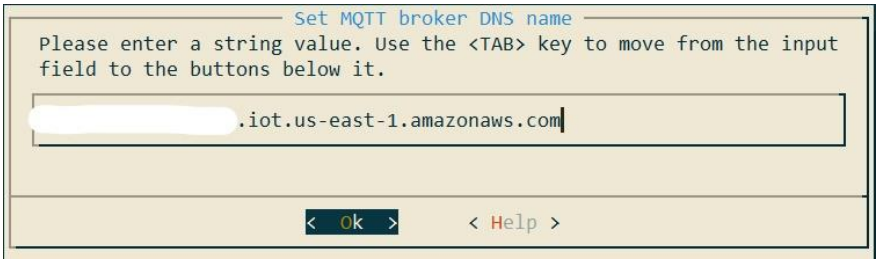


Figure 7-61 MQTT broker DNS name set

e) Select < ok >

10. Input IoT Thing name.

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

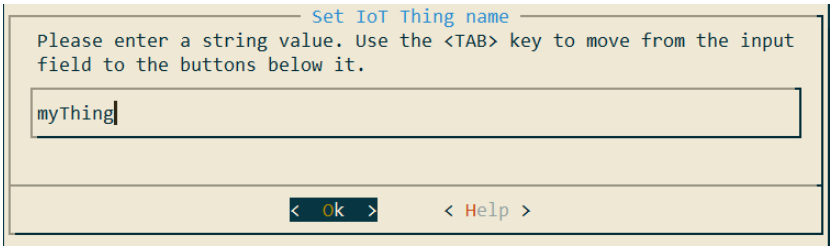


Figure 7-62 Thing name set

b) Select < ok >

11. Select **MCU SFW Component -> secure**
12. Enable mbedtls and modify mbedtls config file to aws_mbedtls_config.h, and then select < ok >

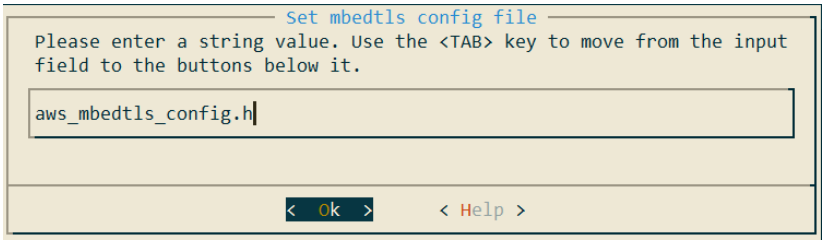


Figure 7-63 Modify mbedtls config file

13. Exit and save the configuration.

Figure 7-64 Save config

7.3.1.4. Prepare image

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

   Note: Please refer to section 2 to generate keil or gcc project.
3. Enter *sfw/target/evkmimxrt1170/iar* path, open sfw.eww project.
4. Go to options, select generate additional output and choose raw binary format.

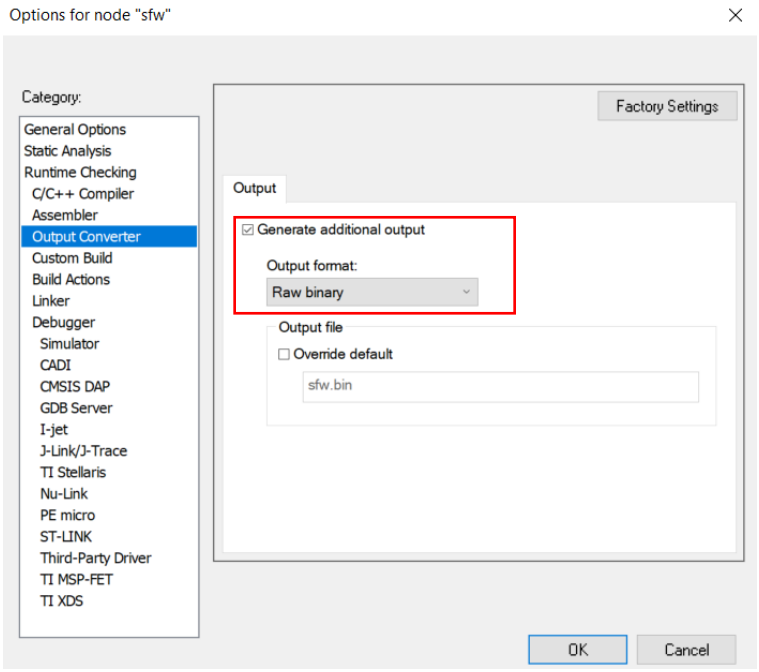


Figure 7-65 Modify to generate .bin file

5. Check 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 7-66 Application version

6. Click the make button to start building the application.
7. If build is successfully, sfw.bin will be generated in *sfw/target/evkmimxrt1170/iar/build/iar/Exe* folder. Change its name according to the application version. Move sfw_092.bin to *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 7-67 New version

9. Sign sfw_092.bin and sfw_093.bin images with RSA using below commands. Then sfw092.bin and sfw093.bin will be 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**.

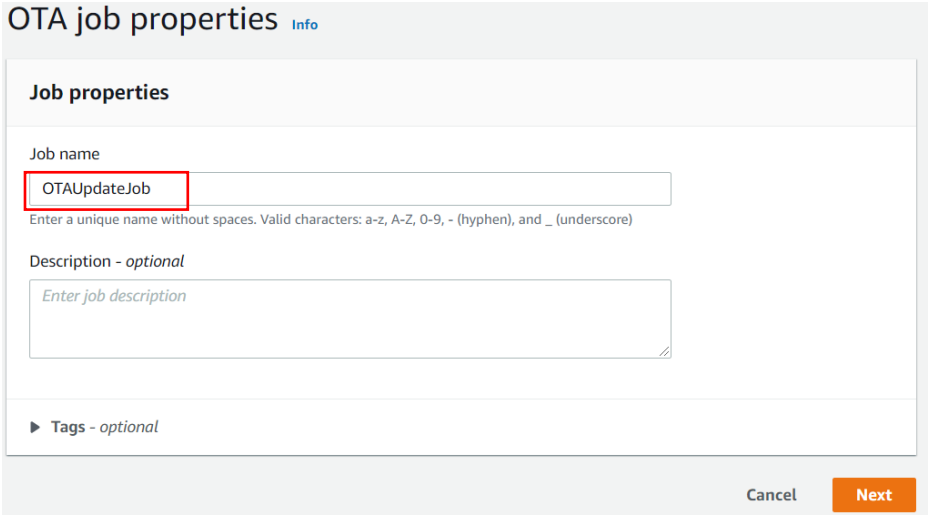


Figure 7-68 Input job name

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

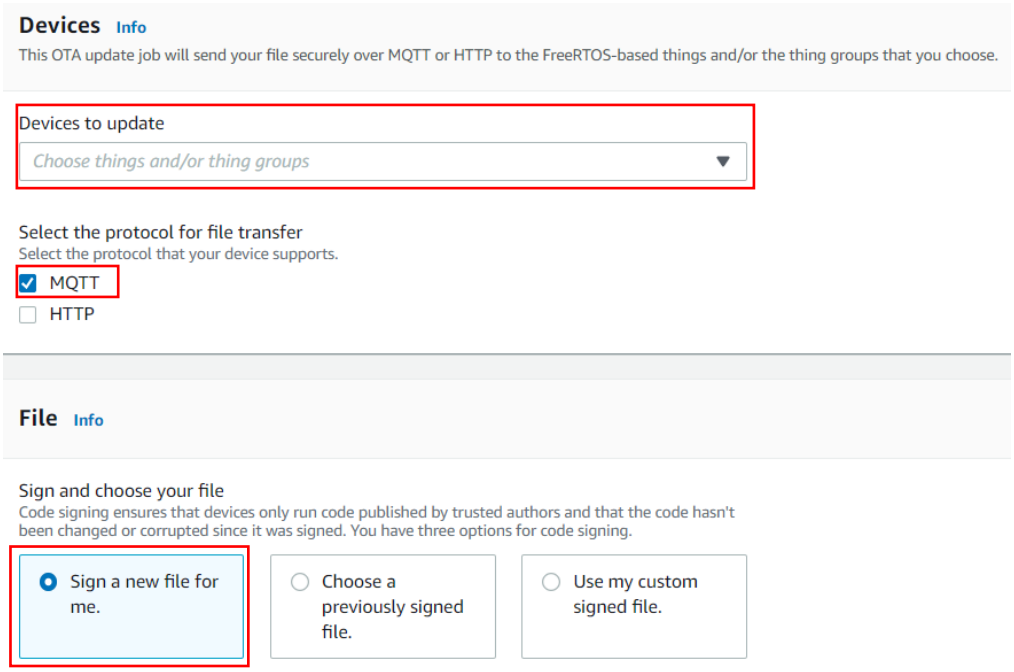


Figure 7-69 Thing, protocol and file select

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**.

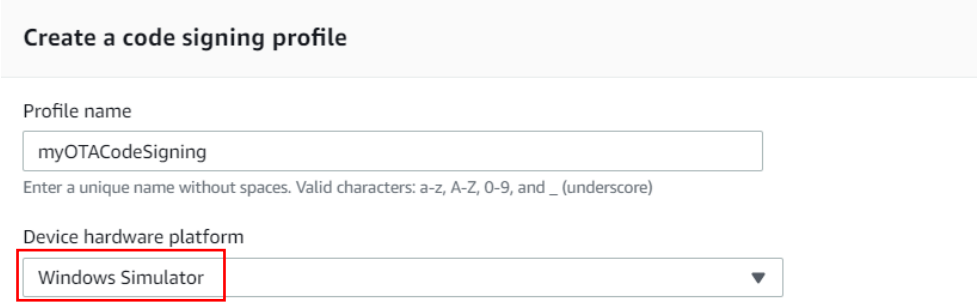


Figure 7-70 Device hardware platform

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

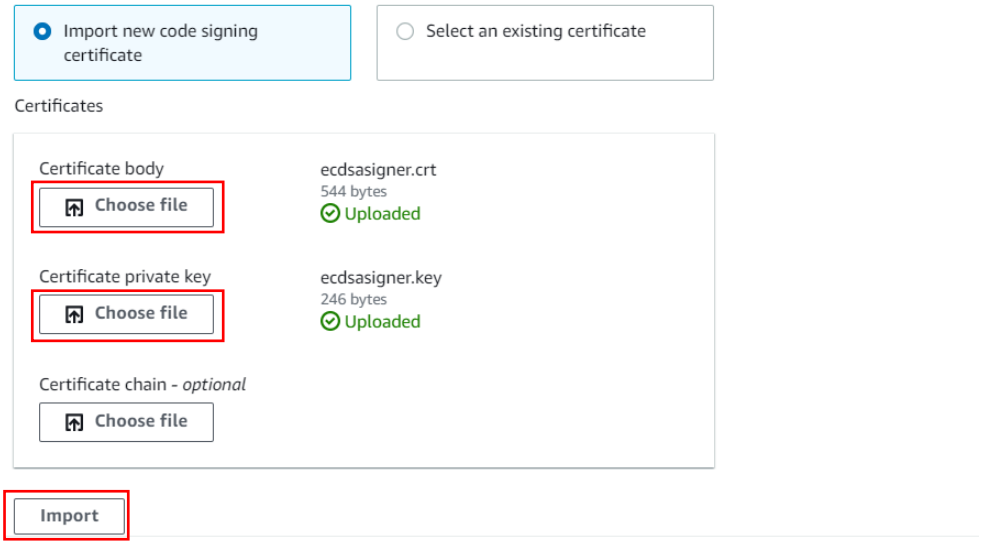


Figure 7-71 Select certificate and key

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

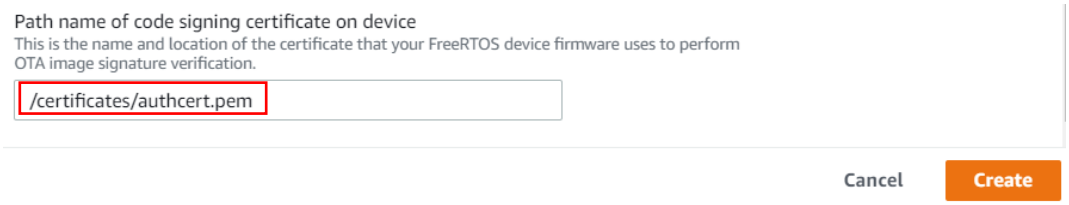


Figure 7-72 Pathname of certificate

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

   Note: Please make sure the region should be the correct one which the bucket is locate. Otherwise the uploaded binary can't be found.

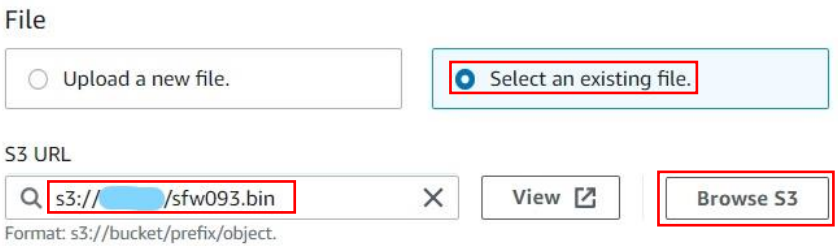


Figure 7-73 Image select

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

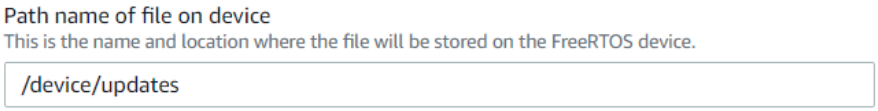


Figure 7-74 Pathname of file

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

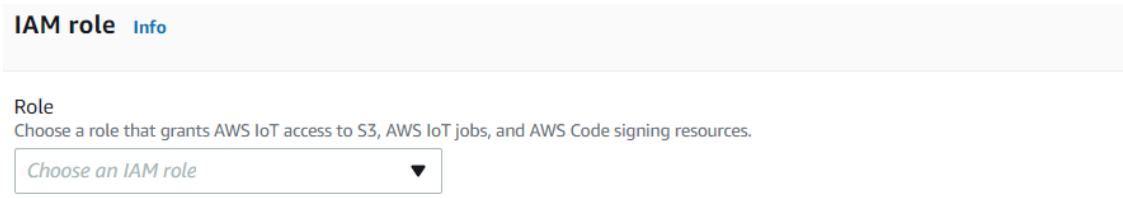


Figure 7-75 Select role

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

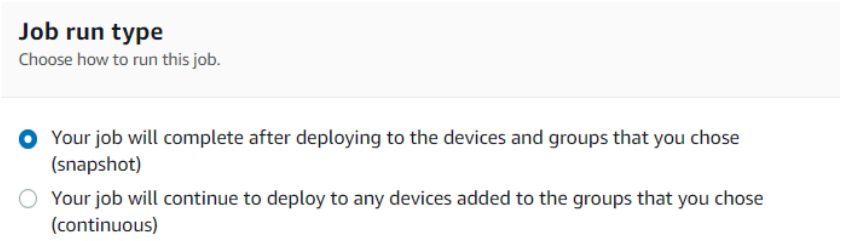


Figure 7-76 Job run type

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

7.3.1.7. Run the application

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

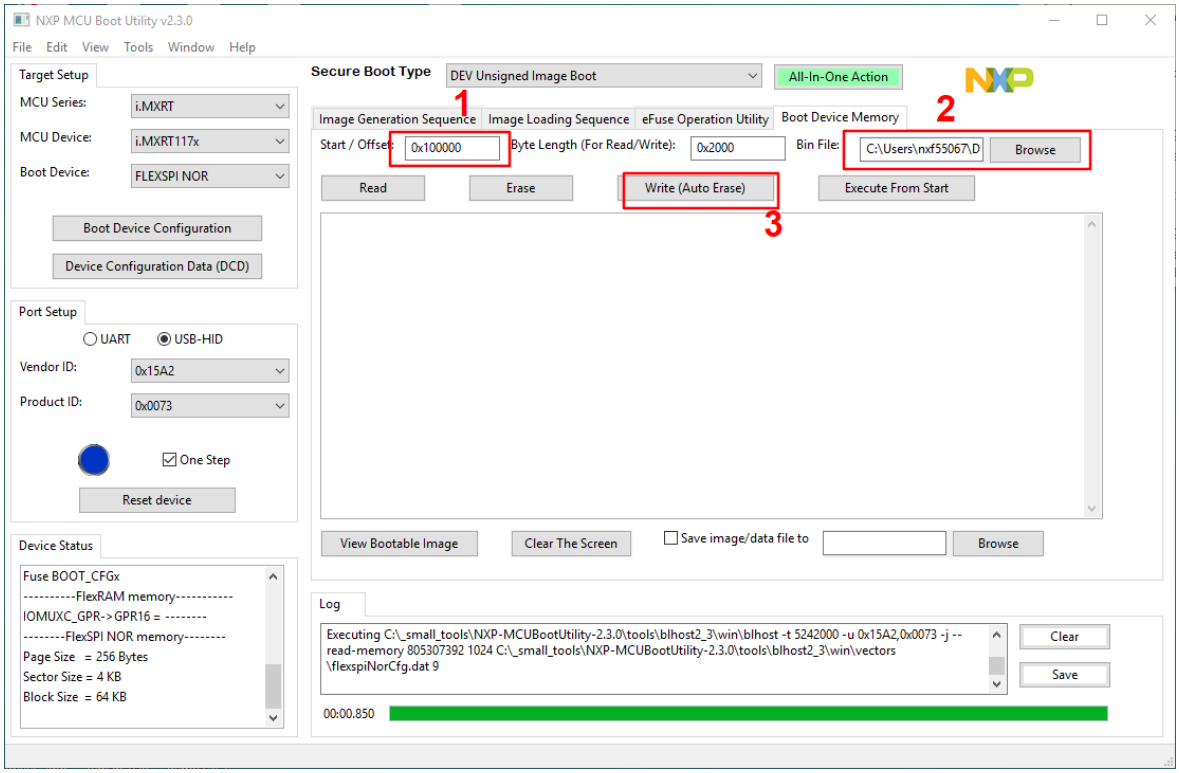
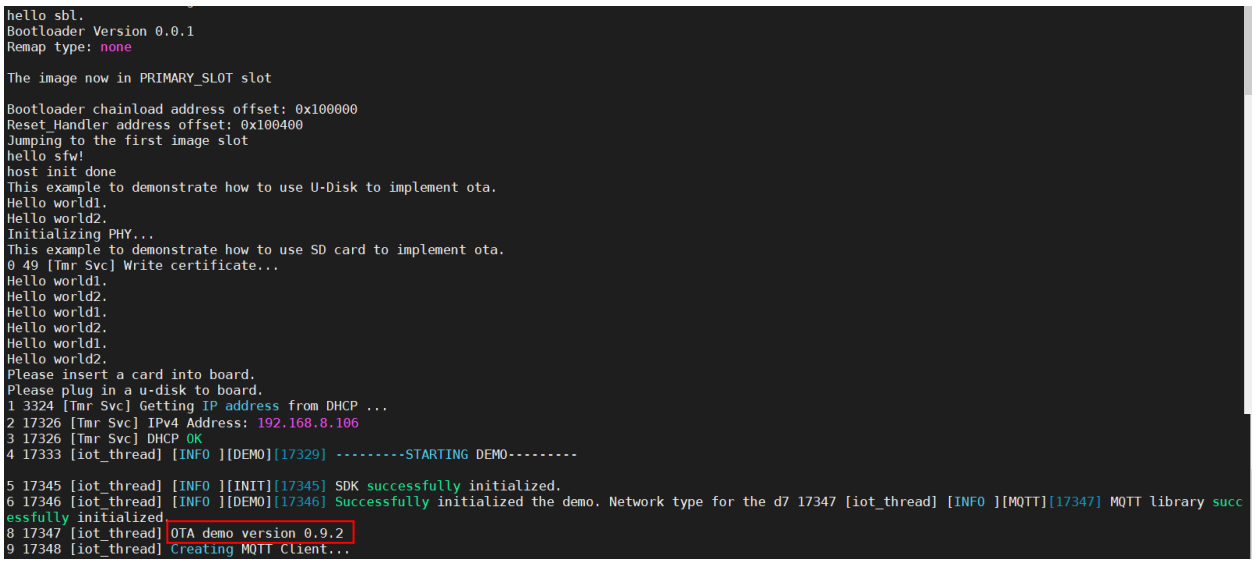


Figure 7-77 Download sfw092.bin to the slot1

2. After successfully download the image, reset the board, and debug console will print the application log as Figure 7-78 shown:



```
10 27520 [iot_thread] Connecting to broker...
11 27520 [iot_thread] [INFO ][MQTT][27520] Establishing new MQTT connection.
12 27541 [iot_thread] [INFO ][MQTT][27540] Anonymous metrics (SDK language, SDK version) will be pr13 27553 [iot_thread] [INFO ][MQTT][27553] (MQTT connection
 202d61a8, CONNECT operation 202d62c0) W14 27812 [iot_thread] [INFO ][MQTT][27811] (MQTT connection 202d61a8, CONNECT operation 202d62c0) W15 27824 [iot_threa
d] [INFO ][MQTT][27823] New MQTT connection 202c169c established.
16 27824 [iot_thread] Connected to broker.
17 27840 [iot_thread] [OTA_AgentInit_internal] OTA Task is Ready.
18 27847 [OTA Agent Task] [prvOTAAgentTask] Called handler. Current State [Ready] Event [Start] New19 27866 [OTA Agent Task] [INFO ][MQTT][27865] (MQTT connec
tion 202d61a8) SUBSCRIBE operation sched20 27875 [OTA Agent Task] [INFO ][MQTT][27875] (MQTT connection 202d61a8, SUBSCRIBE operation 202d6Hello world1.
Hello world2.
21 28150 [OTA Agent Task] [INFO ][MQTT][28149] (MQTT connection 202d61a8, SUBSCRIBE operation 202d622 28160 [OTA Agent Task] [prvSubscribeToJobNotificationTop
ics] OK: $aws/things/SFWOTA1060/jobs/$ne23 28177 [OTA Agent Task] [INFO ][MQTT][28177] (MQTT connection 202d61a8) SUBSCRIBE operation sched24 28187 [OTA Agent
 Task] [INFO ][MQTT][28187] (MQTT connection 202d61a8, SUBSCRIBE operation 202d625 28422 [OTA Agent Task] [INFO ][MQTT][28422] (MQTT connection 202d61a8, SUBS
CRIBE operation 202d626 28434 [OTA Agent Task] [prvSubscribeToJobNotificationTopics] OK: $aws/things/SFWOTA1060/jobs/not27 28443 [OTA Agent Task] [prvRequestJ
ob_Mqtt] Request #0
28 28460 [OTA Agent Task] [INFO ][MQTT][28459] (MQTT connection 202d61a8) MQTT PUBLISH operation qu29 28469 [OTA Agent Task] [INFO ][MQTT][28469] (MQTT connec
tion 202d61a8, PUBLISH operation 202d63b30 28685 [OTA Agent Task] [INFO ][MQTT][28685] (MQTT connection 202d61a8, PUBLISH operation 202d63b31 28711 [OTA Agent
 Task] [prvOTAAgentTask] Called handler. Current State [RequestingJob] Event [Re32 28720 [OTA Agent Task] [prvParseJobDoc] Size of OTA_FileContext_t [64]
37 28763 [OTA Agent Task] [prvParseJSONbyModel] parameter not present: afr_ota
38 28771 [OTA Agent Task] [prvParseJSONbyModel] parameter not present: protocols
39 28779 [OTA Agent Task] [prvParseJSONbyModel] parameter not present: files
40 28787 [OTA Agent Task] [prvParseJSONbyModel] parameter not present: filepath
41 28794 [OTA Agent Task] [prvParseJSONbyModel] parameter not present: filesize
42 28802 [OTA Agent Task] [prvParseJSONbyModel] parameter not present: fileid
43 28810 [OTA Agent Task] [prvParseJSONbyModel] parameter not present: certfile
44 28818 [OTA Agent Task] [prvParseJSONbyModel] parameter not present: sig-sha256-ecdsa
45 28827 [OTA Agent Task] [prvDefaultCustomJobCallback] Received Custom Job inside OTA Agent which 46 28836 [OTA Agent Task] [prvParseJobDoc] Ignoring job wit
hout ID.
47 28840 [iot_thread] State: Ready  Received: 1   Queued: 0   Processed: 0   Dropped: 0
48 28851 [OTA Agent Task] [prvOTA_Close] Context->0x202e65d4
49 28857 [OTA Agent Task] [OTA-NXP] Abort
50 28862 [OTA Agent Task] [OTA-NXP] SetPlatformImageState 4
51 28868 [OTA Agent Task] [OTA-NXP] GetPlatformImageState
52 28874 [OTA Agent Task] [OTA-NXP] ota_status = 0x0
53 28880 [OTA Agent Task] [prvOTAAgentTask] Handler failed. Current State [WaitingForJob] Event  [RHello world1.
Hello world2.
54 29840 [iot_thread] State: WaitingForJob  Received: 1   Queued: 0   Processed: 0   Dropped: 0
```

Figure 7-78 Application log

3. When running the application wait until the message of the OTA State Ready is shown in the serial terminal as Figure 7-79 shown:

```
54 28610 [iot_thread] State: WaitingForJob  Received: 1   Queued: 0   Processed: 0   Dropped: 0
Hello world1.
Hello world2.
55 29610 [iot_thread] State: WaitingForJob  Received: 1   Queued: 0   Processed: 0   Dropped: 0
Hello world1.
Hello world2.
56 30610 [iot_thread] State: WaitingForJob  Received: 1   Queued: 0   Processed: 0   Dropped: 0
Hello world1.
Hello world2.
```

Figure 7-79 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**.

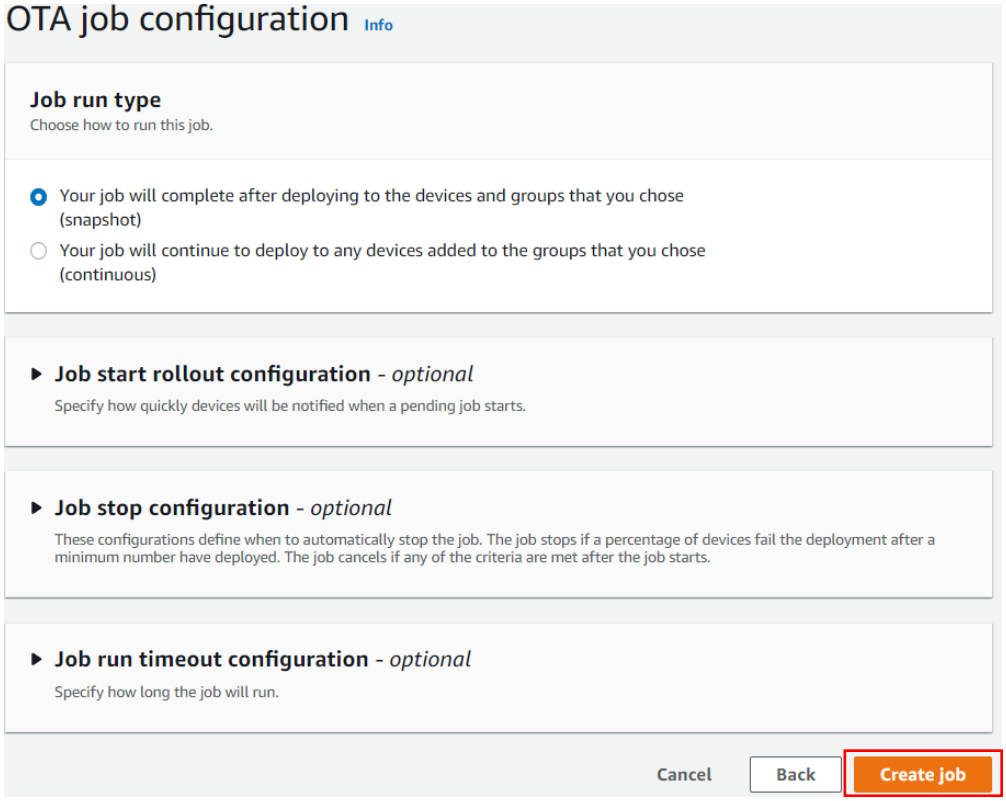


Figure 7-80 Create job

5. The process will start, and the outputs are below.

   a) Start file transfer

```
70 40447 [OTA Agent Task] [prvParseJSONbyModel] Extracted parameter [ filesize: 475842 ]
71 40455 [OTA Agent Task] [prvParseJSONbyModel] Extracted parameter [ fileid: 0 ]
72 40464 [OTA Agent Task] [prvParseJSONbyModel] Extracted parameter [ certfile: /certificates/authc73 40473 [OTA Agent Task] [prvParseJSONbyModel] Extracted p
arameter [ sig-sha256-ecdsa: MEUCICnhyW+74 40483 [OTA Agent Task] [prvParseJobDoc] Job was accepted. Attempting to start transfer.
75 40492 [OTA Agent Task] [OTA-NXP] GetPlatformImageState
76 40499 [OTA Agent Task] [OTA-NXP] ota_status = 0x0
77 40504 [OTA Agent Task] [OTA-NXP] CreateFileForRx
78 40510 [OTA Agent Task] [OTA-NXP] image_position= 1
79 40515 [OTA Agent Task] [OTA-NXP] File_Addr = 0x00200000, File_Size = 0x00100000
80 40524 [OTA Agent Task] [prvSetDataInterface] Data interface is set to MQTT.
81 40531 [OTA Agent Task] [prvProcessJobHandler] Setting OTA data inerface.
```

```
92 41394 [OTA Agent Task] [prvIngestDataBlock] Received file block 4, size 1024
93 41403 [OTA Agent Task] [OTA-NXP] WriteBlock 1000 : 400
94 41413 [OTA Agent Task] [prvIngestDataBlock] Remaining: 464
```

Figure 7-81 Start file transfer log

   b) Received whole file

```
2254 150969 [OTA Agent Task] [OTA-NXP] WriteBlock 74000 : 2c2
2255 150976 [OTA Agent Task] [prvIngestDataBlock] Received final expected block of file.
2256 150985 [OTA Agent Task] [prvStopRequestTimer] Stopping request timer.
```

Figure 7-82 Received whole file log

   c) Check file signature

```
2257 150992 [OTA Agent Task] [OTA-NXP] CloseFile
2258 150997 [OTA Agent Task] [OTA-NXP] CheckFileSignature
```

Figure 7-83 File signature check log

   d) Check image version

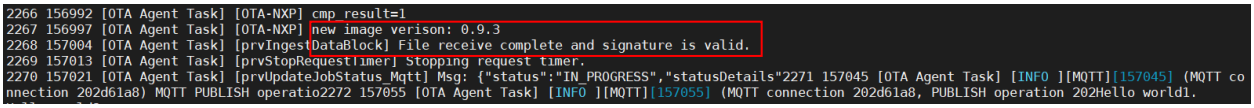


Figure 7-84 Image version check log

e) Write update type

Figure 7-85 Write update type log

f) Write image trailer

Figure 7-86 Write image trailer log

g) Active new image, device reset

Figure 7-87 Active new image log

h) Running new image

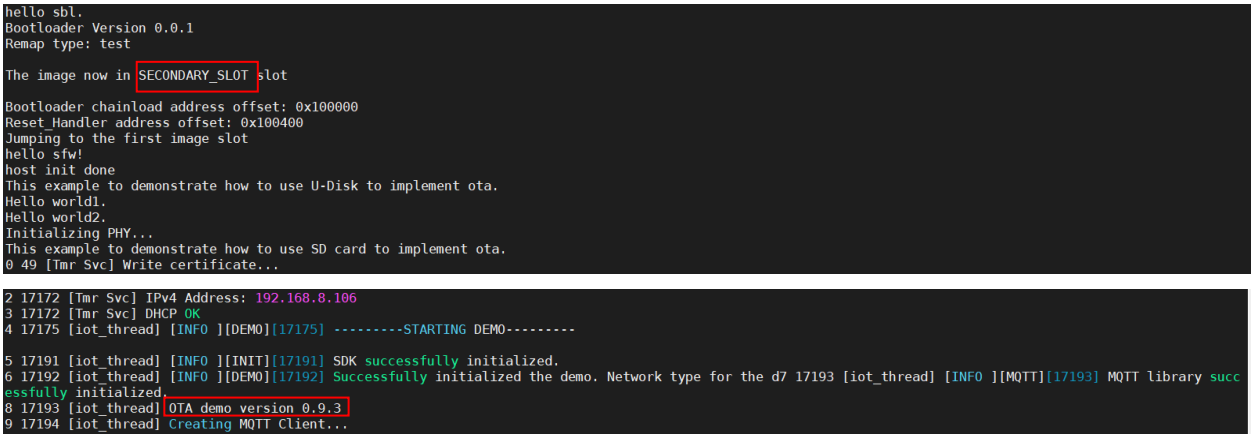


Figure 7-88 Run new image log

i) Self-test

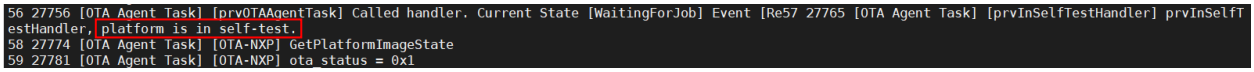


Figure 7-89 Self-test log

j) Write OK flag

Figure 7-90 Write OK flag log

k) OTA success

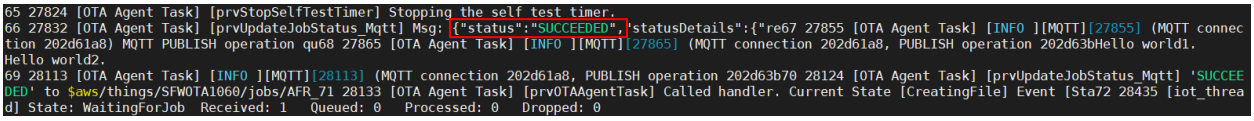


Figure 7-91 OTA success log

6. After OTA success, push the reset button on the board to confirm that the updating successful, the sfw093.bin log should be printed.

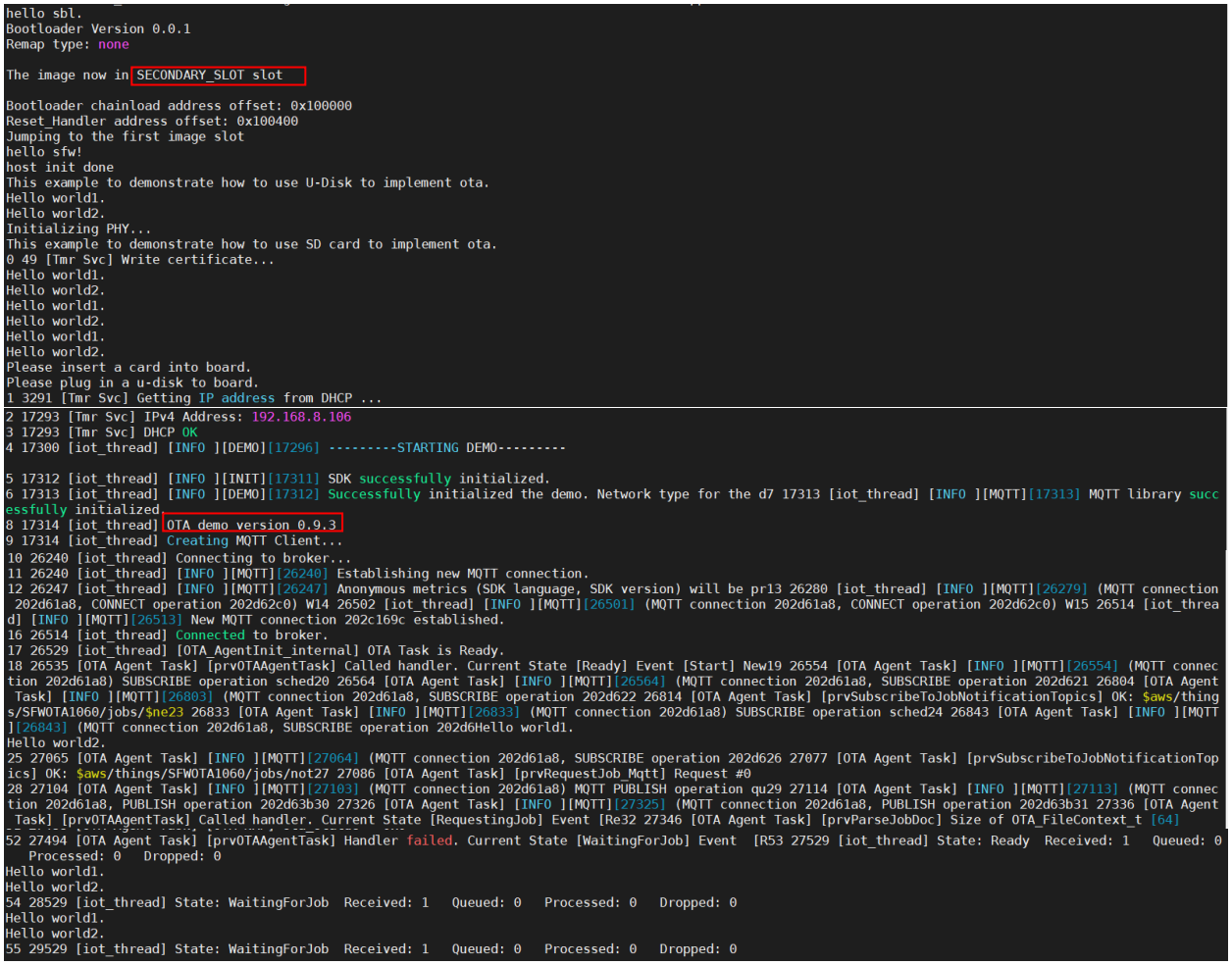


Figure 7-92 New image log

7.3.2. Aliyun OTA

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

7.3.2.1. Create testing device

Step1. Open the link: https://iot.console.aliyun.com/. Register an account to log in to the platform and create the Alibaba Cloud account.

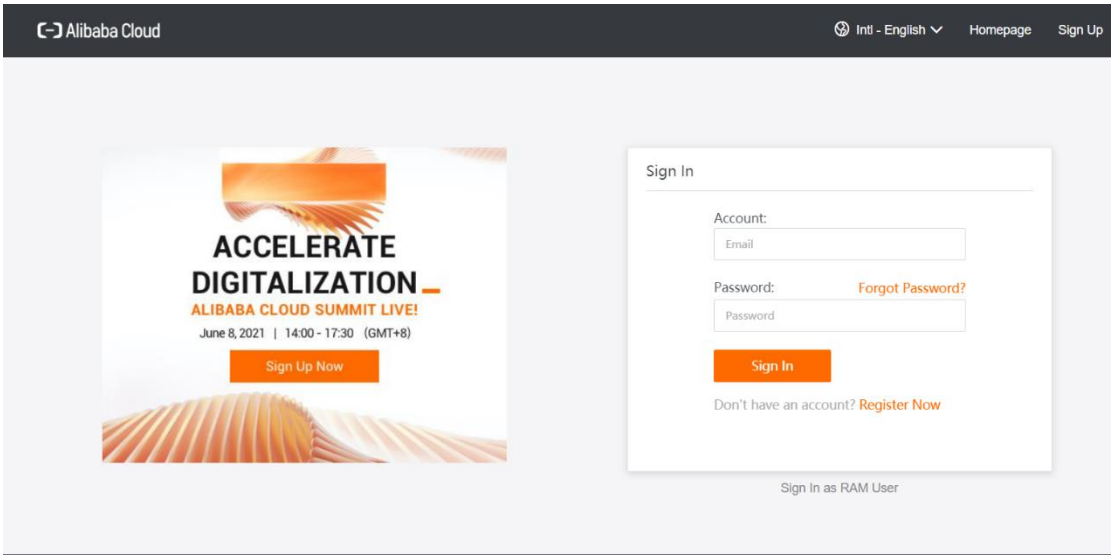


Figure 7-93 Account creation interface

Step2. After the login is successful, the page of the Alibaba Cloud IOT platform will be displayed, click to enter the "**Public Instance**" interface.

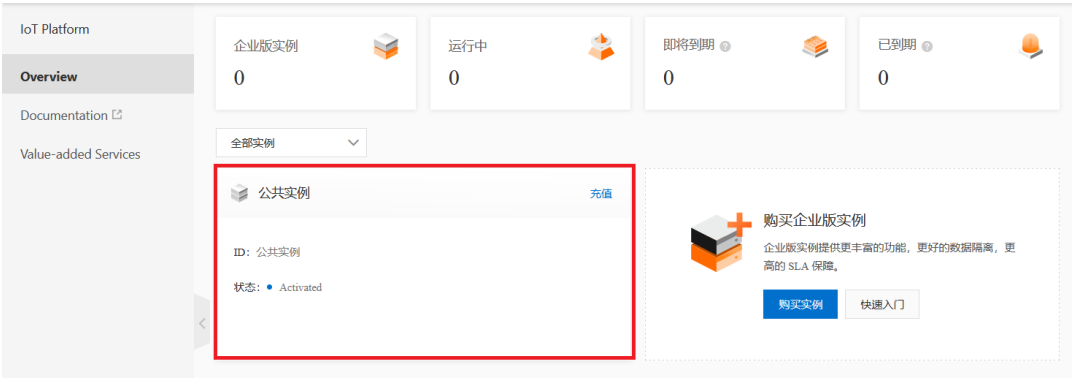


Figure 7-94 Alibaba Cloud IOT platform

Step3. 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 (just to test the OTA function).

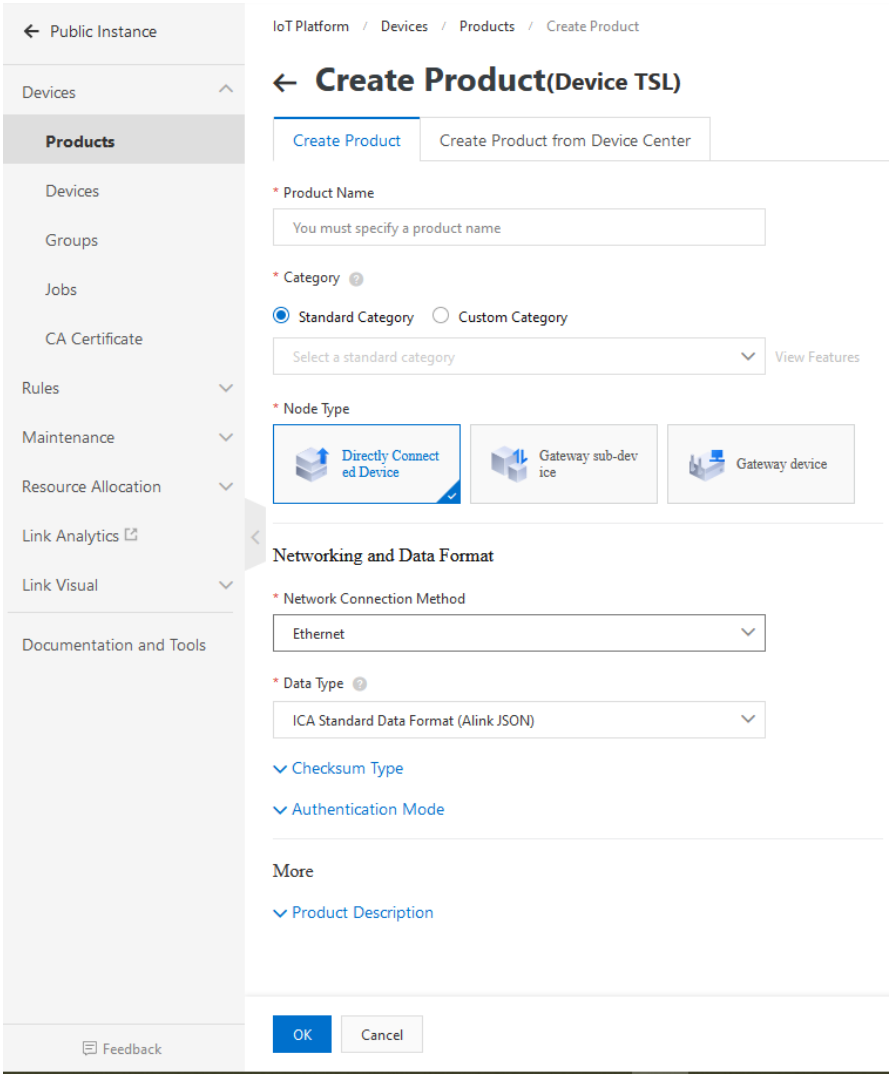


Figure 7-95 Create Product

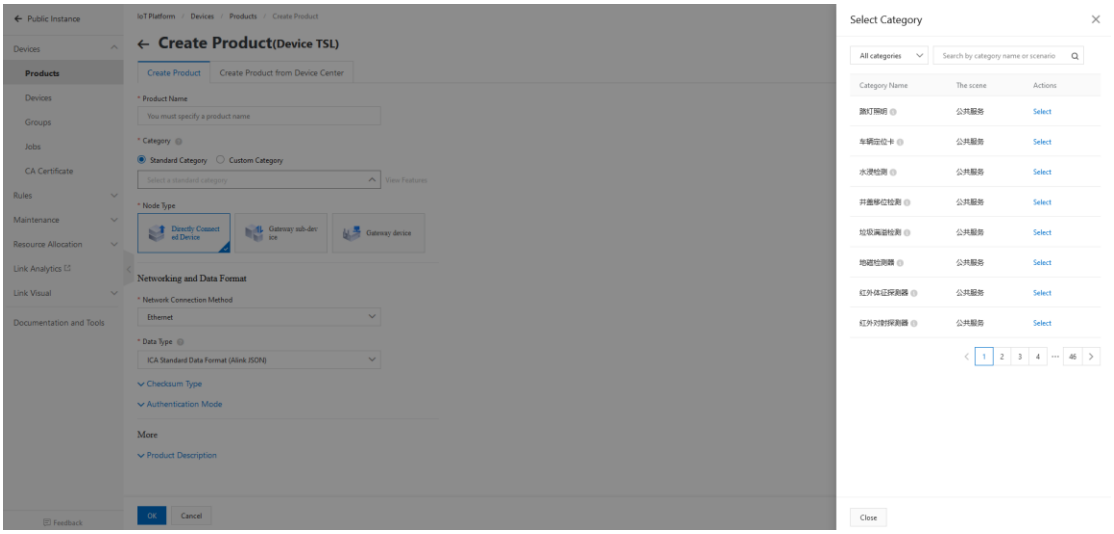


Figure 7-96 Product Type

Add equipment to the product after it is successfully created.

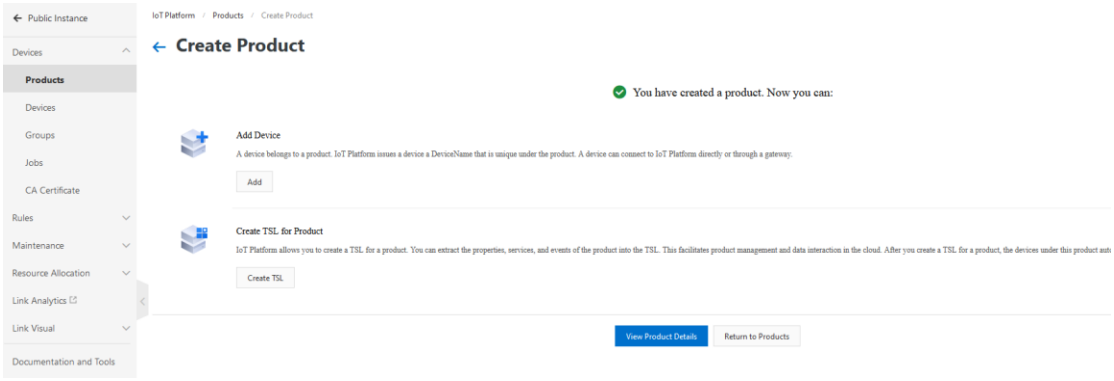


Figure 7-97 Create product successfully

Step4. Select the product that needs to add a device for testing first, and then click the blue button of "**Add Device**" to set the device name.

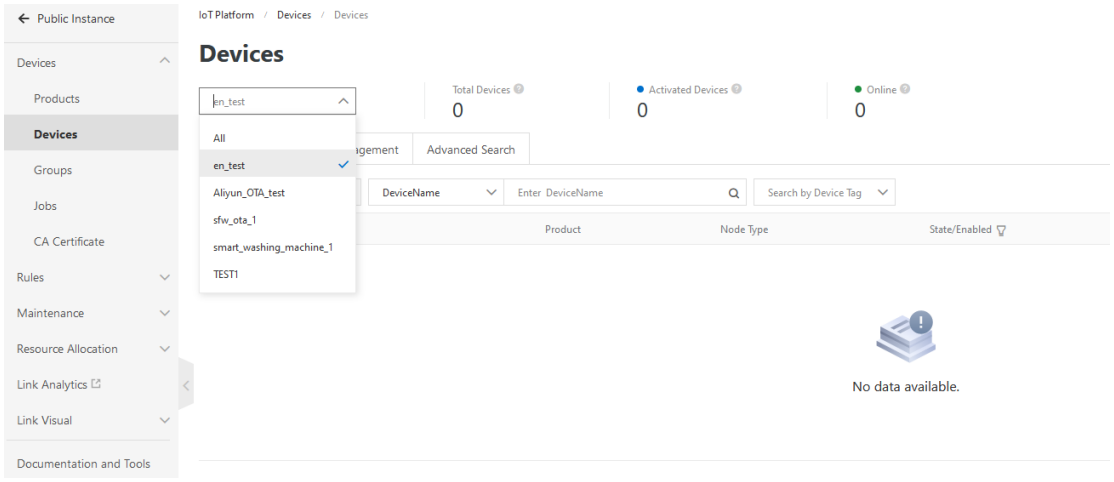


Figure 7-98 Add device

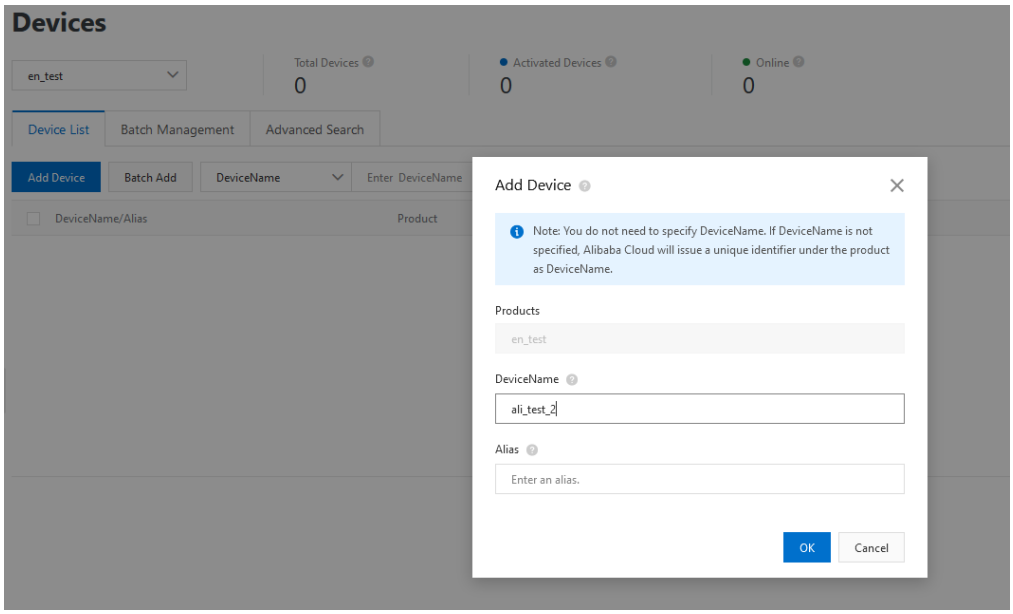


Figure 7-99 Device information

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

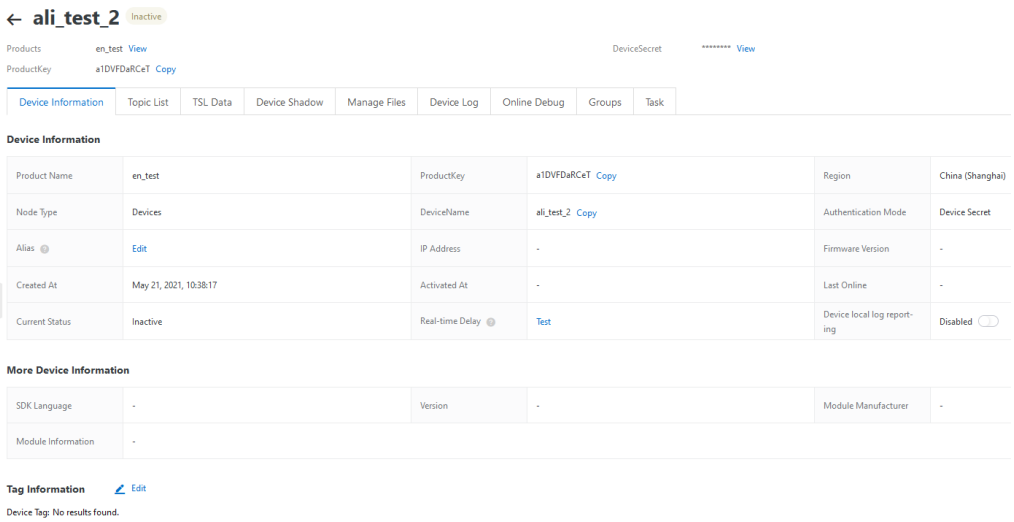


Figure 7-100 Device details

7.3.2.2. Customize device-side SDK

Step1. 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.

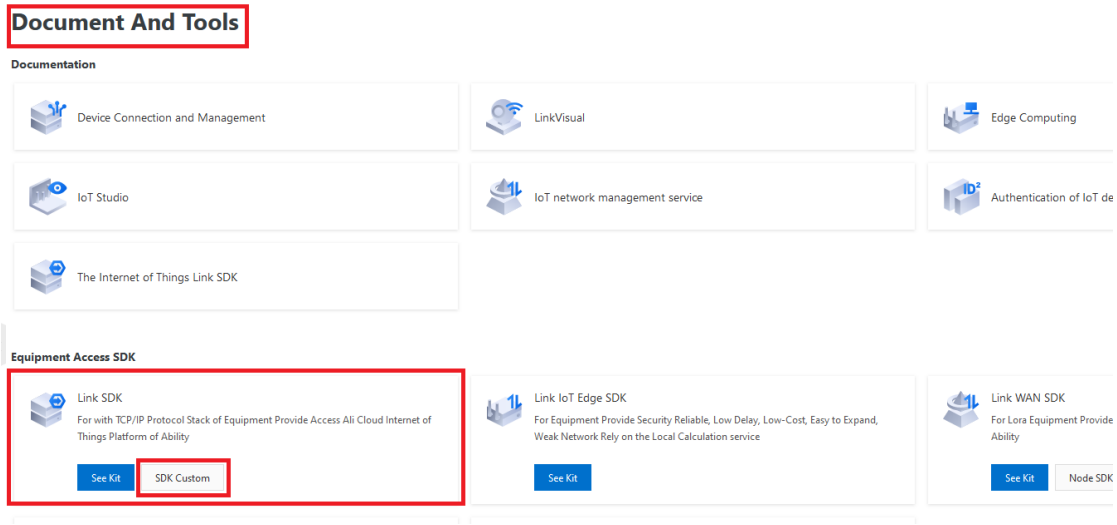


Figure 7-101 Download SDK

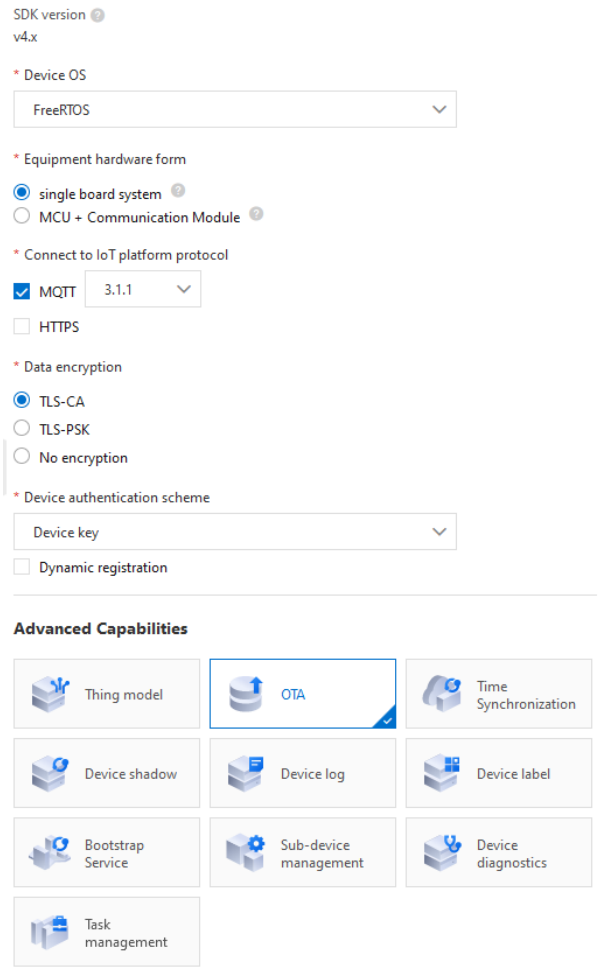


Figure 7-102 Customize SDK

Step2. 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

Step1. Under the "**Device**" option of the Alibaba Cloud IOT platform, save the **DeviceName**, **ProductKey**, and **DeviceSecret** information.

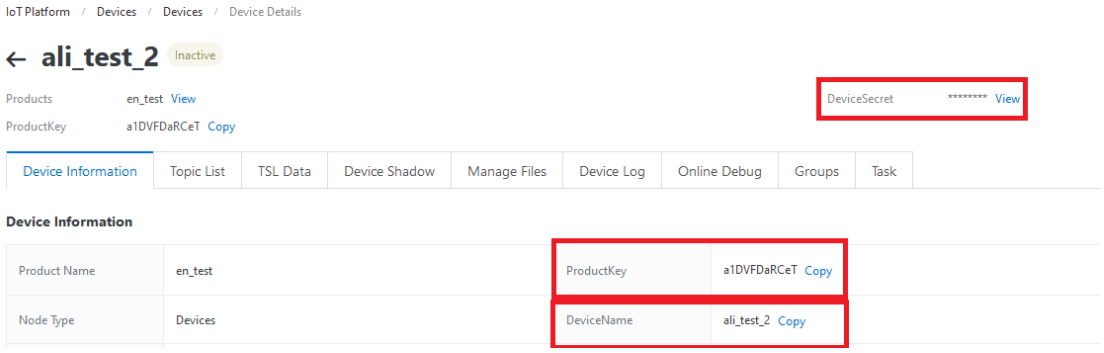


Figure 7-103 Triple information

Step2. Open the test project - \ sfw \target \evkbmimxrt10XX, double click the script **env.bat**, using command `scons --menuconfig` to select the Alibaba Cloud project, and set the triple information as shown below, copy the product key/device name/product secret into it.

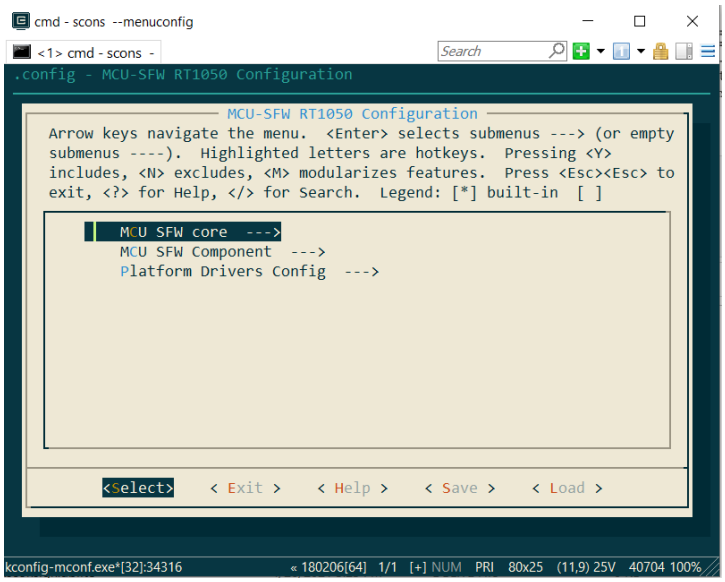


Figure 7-104 Menuconfig

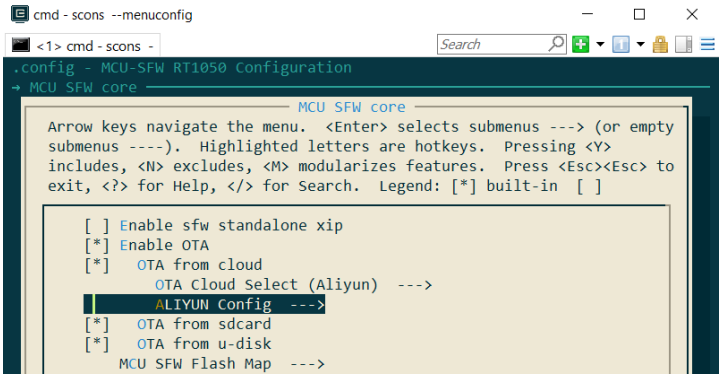


Figure 7-105 Set Aliyun OTA information

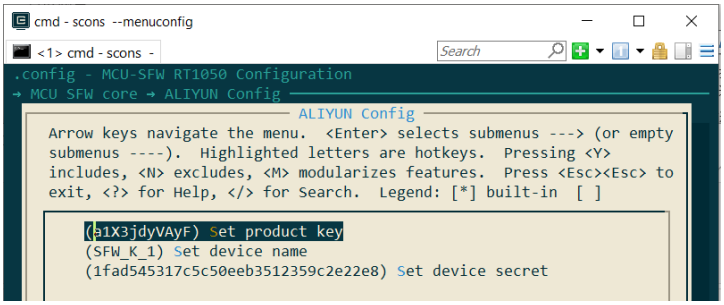


Figure 7-106 Set triple information

Step3. In the script, choose ‘MCU SFW Component -> secure -> enable mbedtls’, and change the mbedtls config file to “ksdk_mbedtls_config.h”.

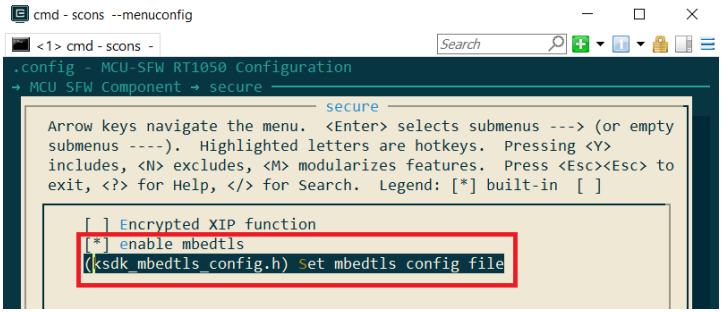


Figure 7-107 Set mbedtls config file

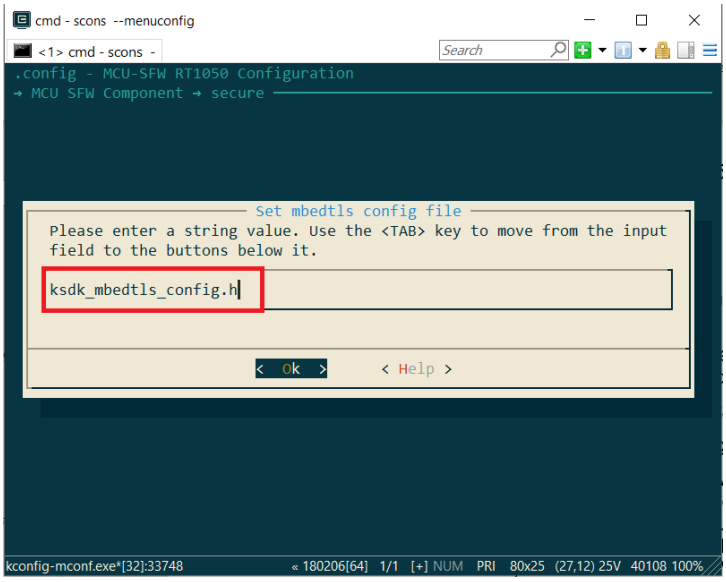


Figure 7-108 ksdk_mbedtls_config.h

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

7.3.2.4. Modify the "cur_version" for testing

Step1. Enter *sfw/target/evkmimxrt1064* path, double click the batch file *env.bat*

Step2. Input `scons –ide=iar` command to generate iar project.

Note: Please refer to section 2 to generate keil or gcc project.

Step3. Enter *sfw/target/evkmimxrt1064/iar* path, open sfw.eww project.

Step4. Go to options, select generate additional output and choose raw binary format.

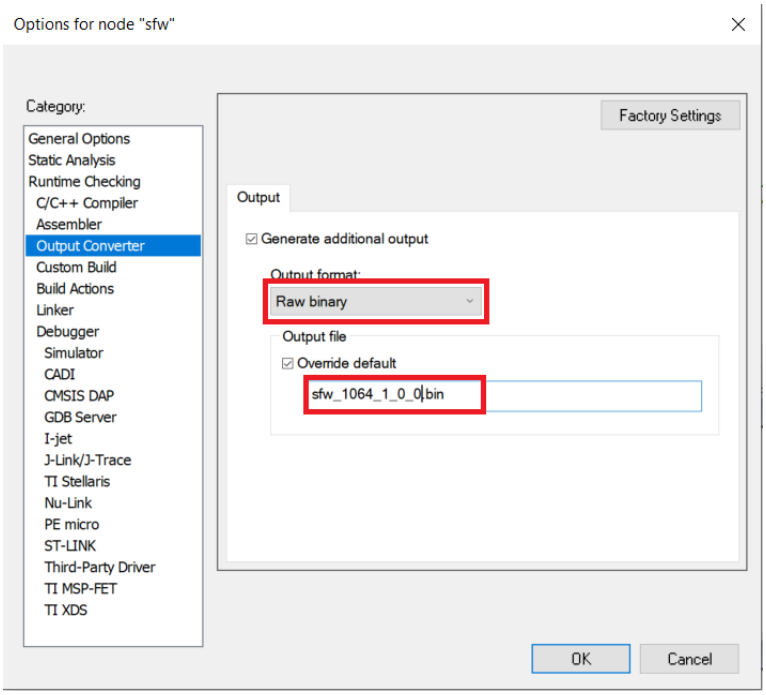


Figure 7-109 Output file

Step5. Check 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 7-110 Set the version

Step6. 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*.

Step7. Change the cur_version to “1.4.0”, and change bin file name to “sfw_1064_140.bin”. Generate it.

Step8. Copy the generated bin files to *sbl/component/secure/mcuboot/scripts* folder.

Step9. Sign sfw_1064_100.bin and sfw_1064_140.bin images with RSA using below command. Then 1064_ali_100.bin and 1064_ali_140.bin will be 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

Step1. Under the "**Maintenance**" directory on the left, select "**OTA Update**".

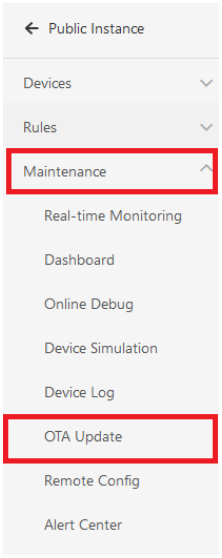


Figure 7-111 OTA upgrade

Step2. 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.

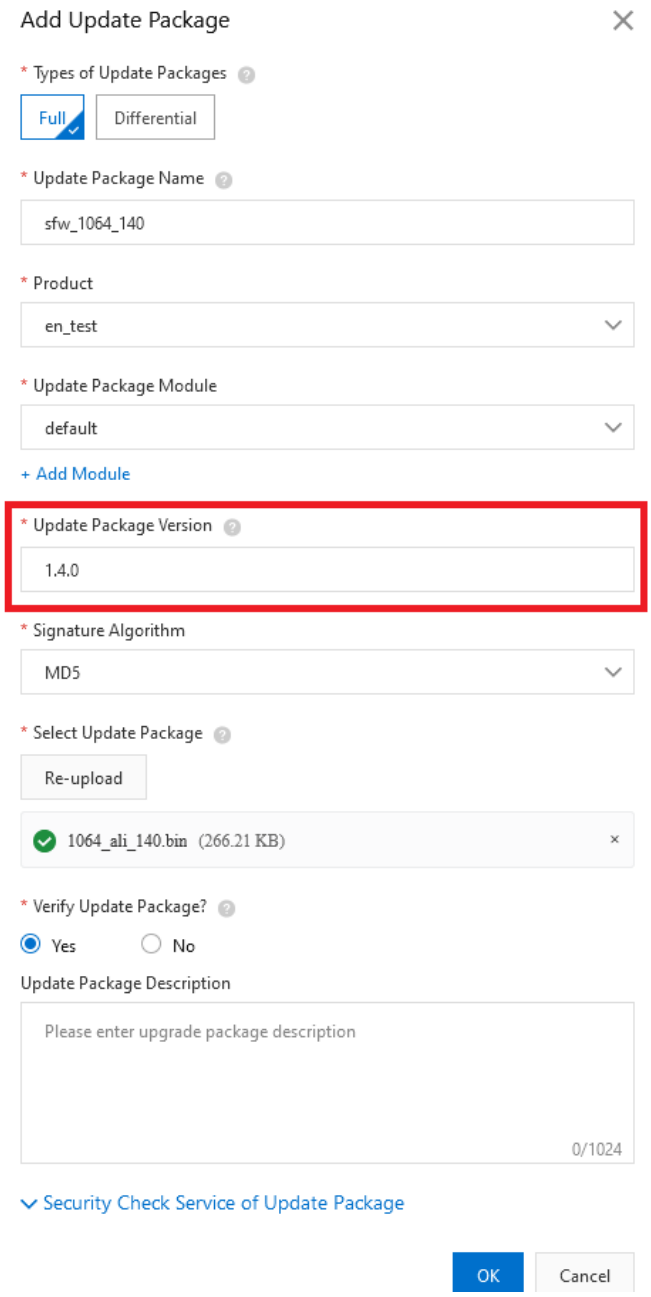


Figure 7-112 Upgrade package information

7.3.2.6. Run the application

Step1. Using the *MCUBootUtility* tool to download the *1064_ali_100.bin* generated previously to the first slot of the board, the default location of the slot1 is the **flash_offset+0x100000** to **flash_offset+0x200000**, the whole slot size is 1MB.

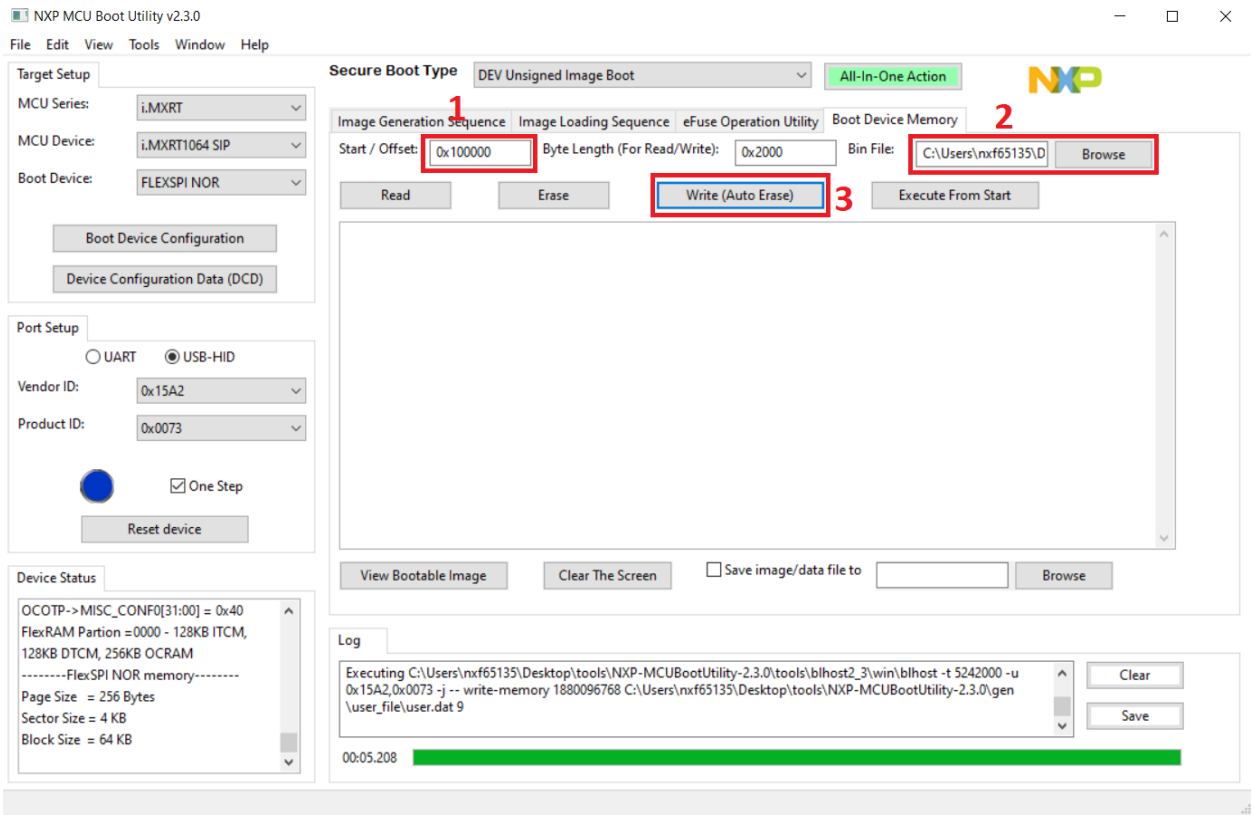


Figure 7-113 Use image tool to download bin file

Step2. 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:

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

Step3. Plug in the ethernet cable. Run the project and the debug console will print 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 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.

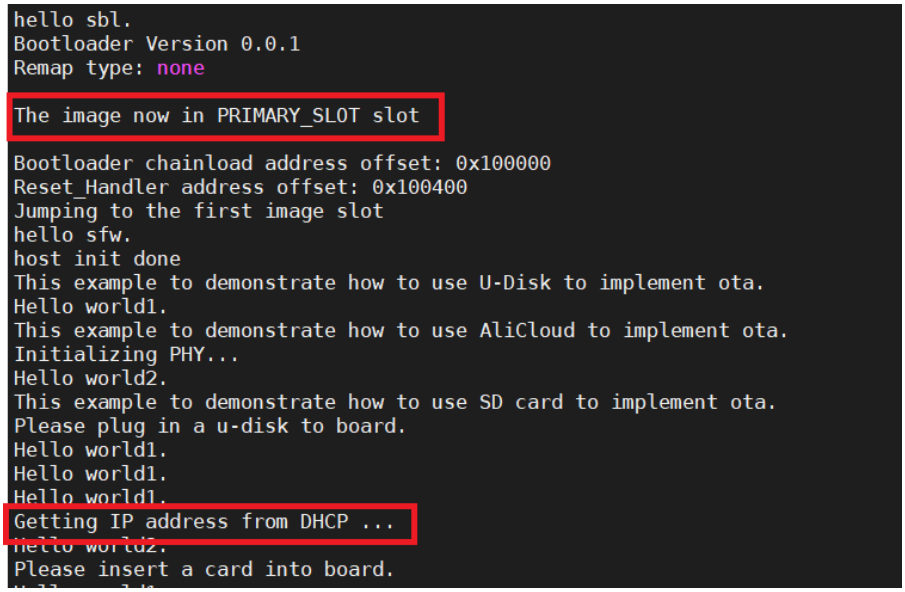


Figure 7-114 Run sbl project

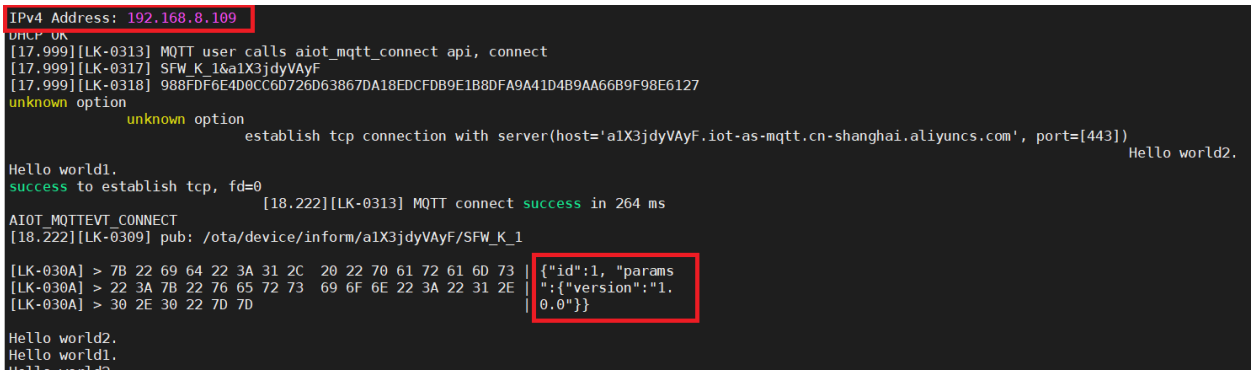


Figure 7-115 Jump to first image

Step4. 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. And click "**OK**".

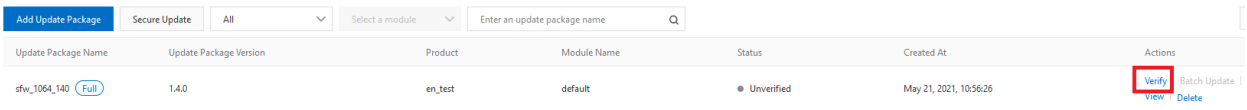


Figure 7-116 Verify the OTA package

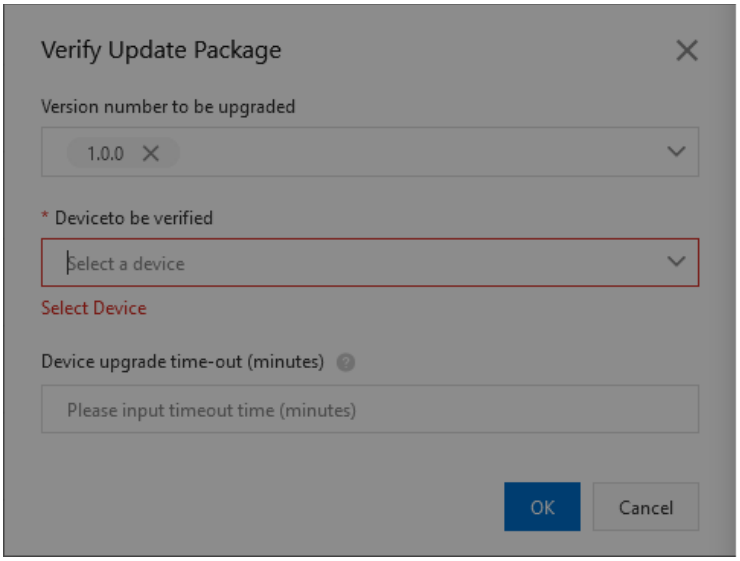


Figure 7-117 Verify the upgrade package information

Step5. The debug console will show the OTA progress.

Below shows the upgrade package information and package 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 7-118 Get the package information

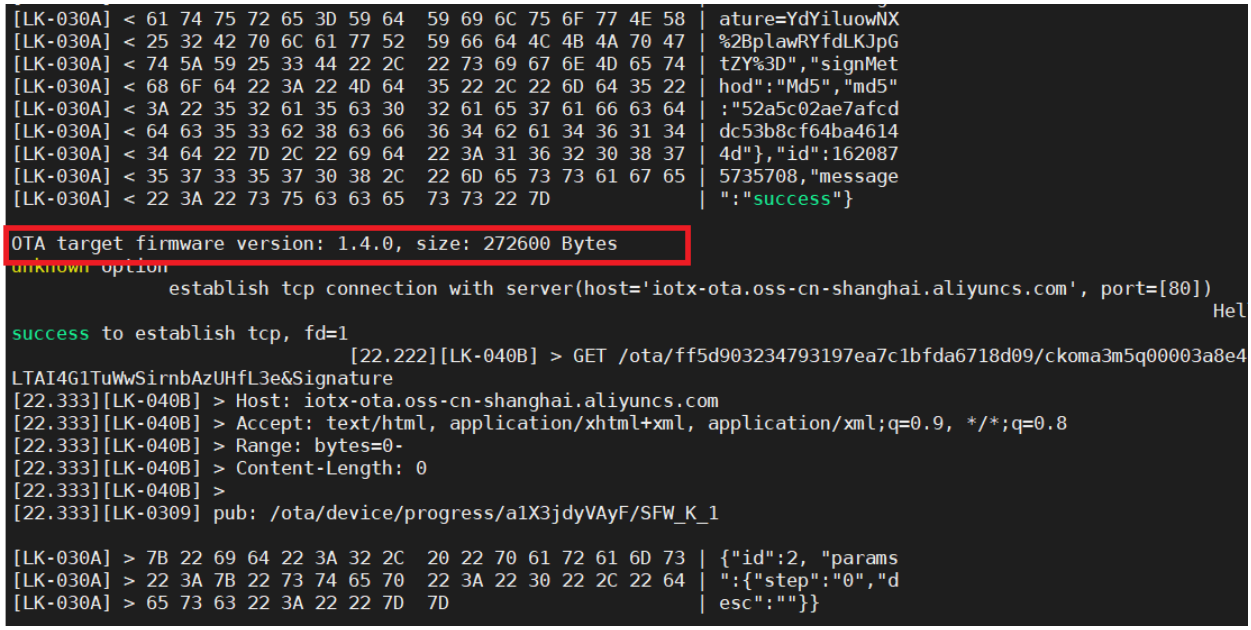


Figure 7-119 Target firmware version

Below shows the download request has been sent successfully and start the download progress.

```
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 7-120 Download request

Below shows the download progress finished and start the system reset action.

```
[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 7-121 Download all

Below shows the current package version is 1.4.0. And 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 7-122 Run the new image

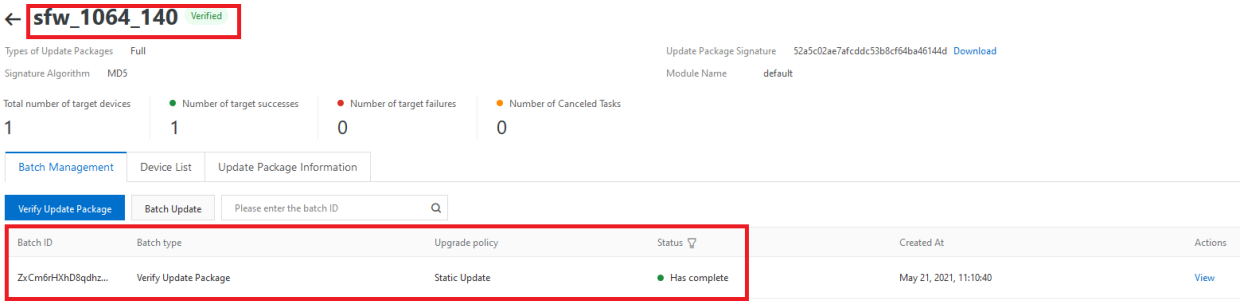


Figure 7-123 Alibaba Cloud IOT platform shows the OTA success

7.4. Secure FOTA

7.4.1. Secure boot demonstration for platform EVKMIMXRTxxxx

This section describes the steps to enable secure boot and generate signed image. The demo targeted for 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, user need to 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, user need to create a new workspace.

   Note: user must open MCUXpresso Secure Provisioning tool with administrator mode. Otherwise some important material will not be generated.

   Note: For EVKMIMXRT1010 platform, user may select MIMXRT1015 to generate keys.

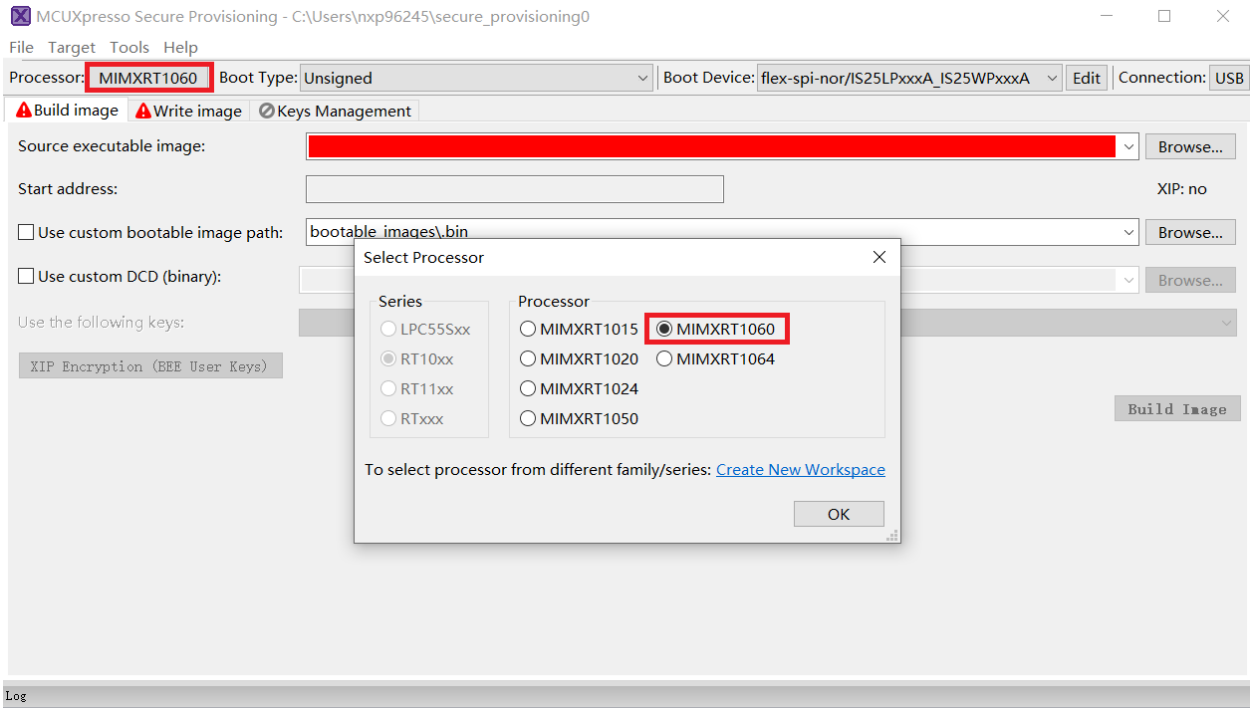


Figure 7-124: Creating a new workspace

3. Choose Boot Type as Authenticated (HAB).

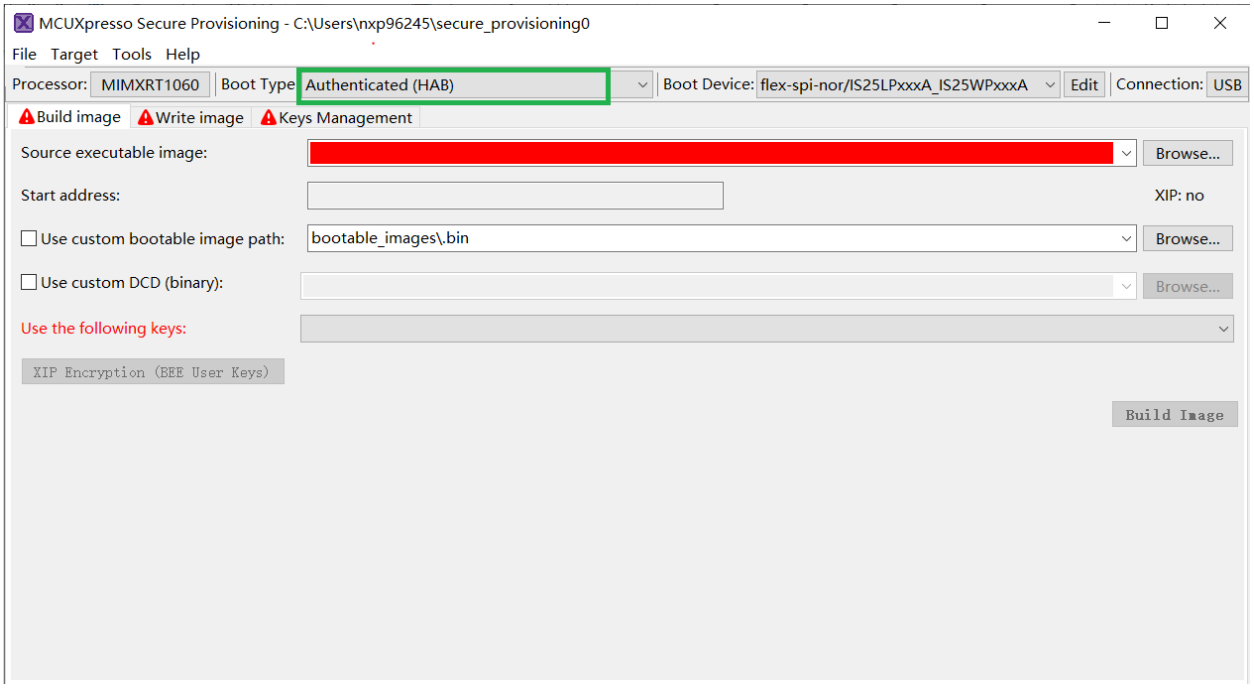


Figure 7-125: Selecting Boot Type

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

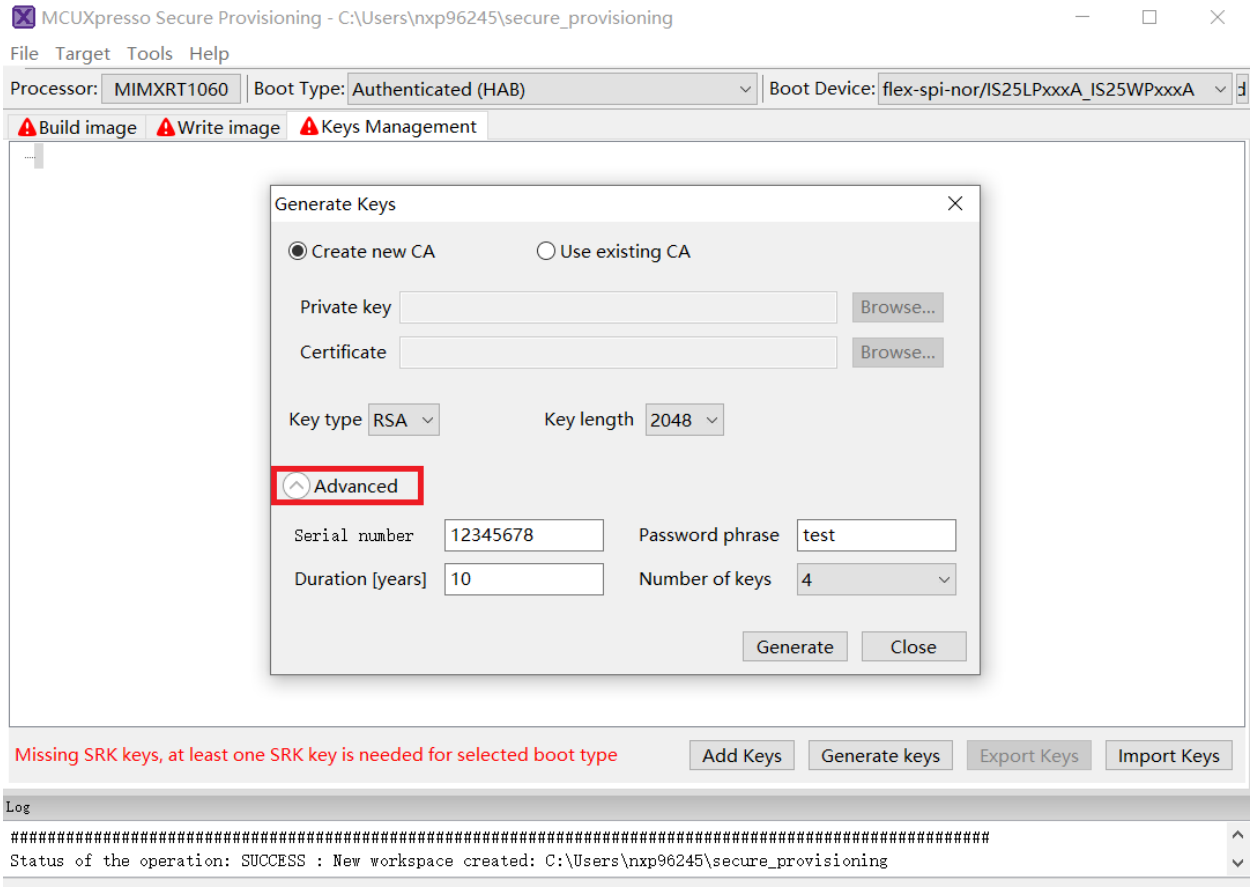


Figure 7-126: Generate keys (EVKMIMXRTRT10xx, EVKMIMXRTRT11xx)

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

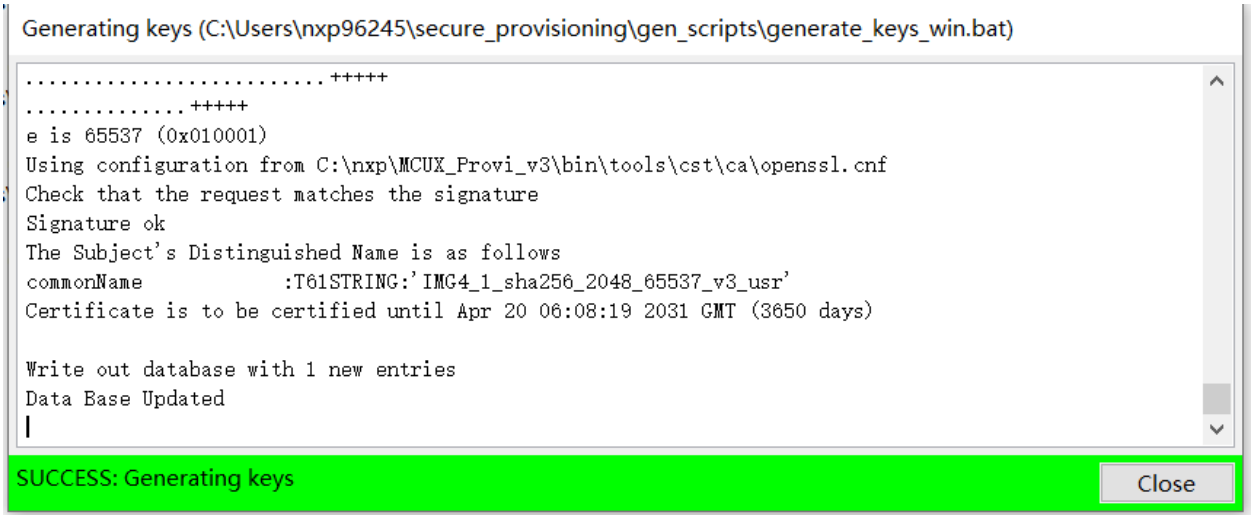


Figure 7-127: Generate Keys - Success

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

7.4.1.2. SBL image preparation

To generate 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"

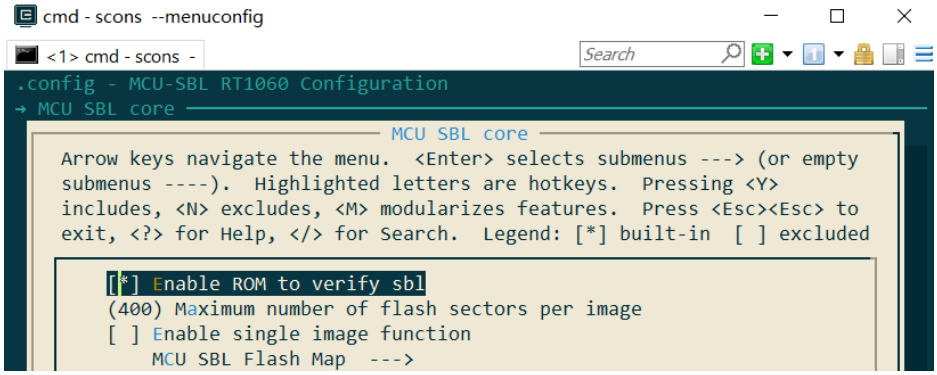


Figure 7-128:Enable ROM to verify sbl

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

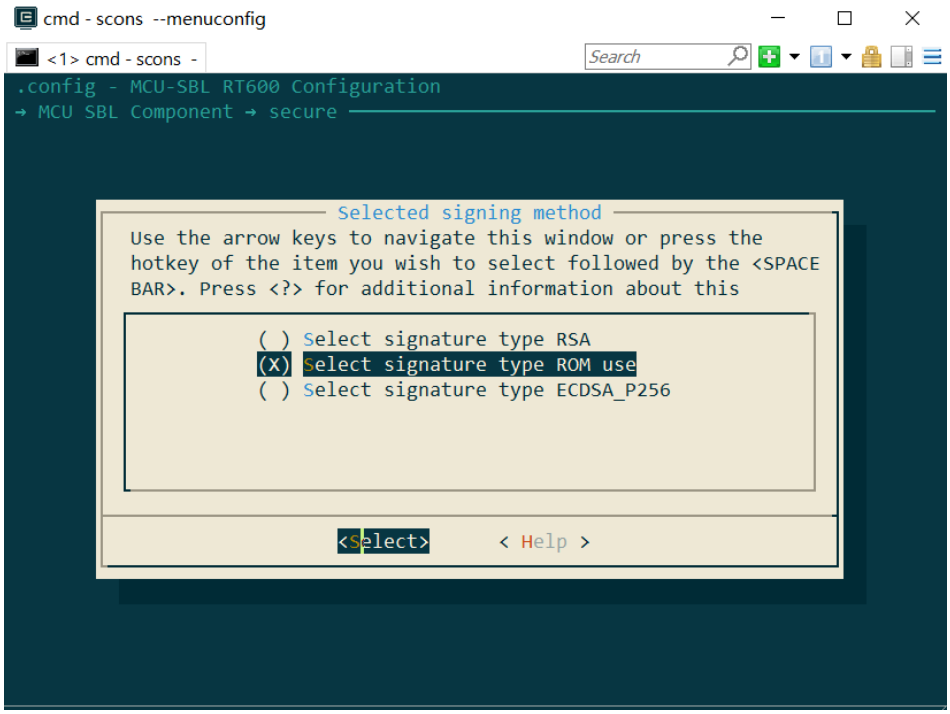


Figure 7-129: Select signing method

4. Run `scons --ide=iar` command to generate IAR project or run `scons --ide=mdk5` to generate Keil project.

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

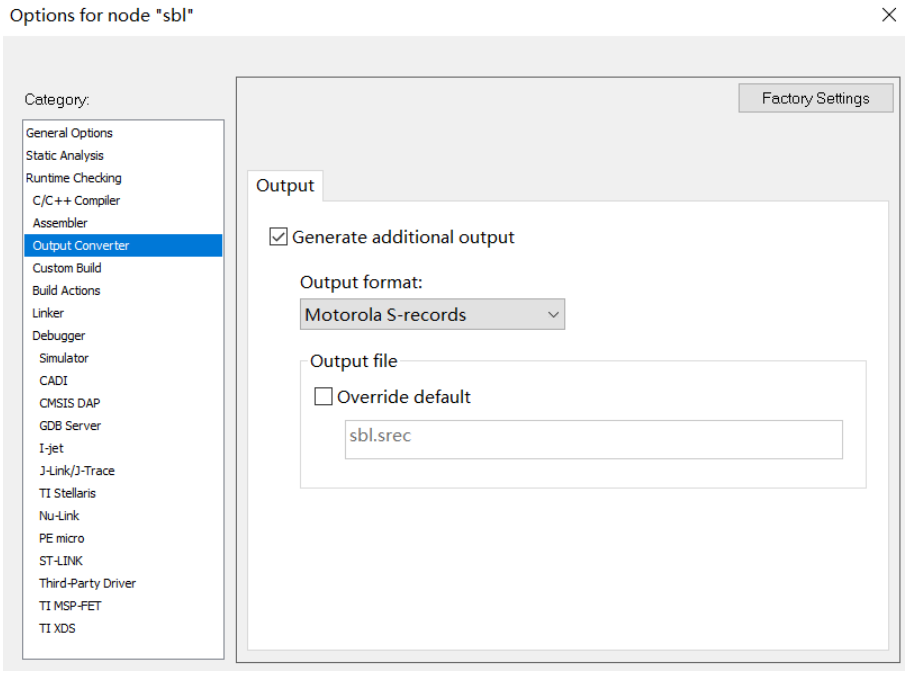


Figure 7-130: Update defined Symbols

7.4.1.3. Application image preparation

To generate 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

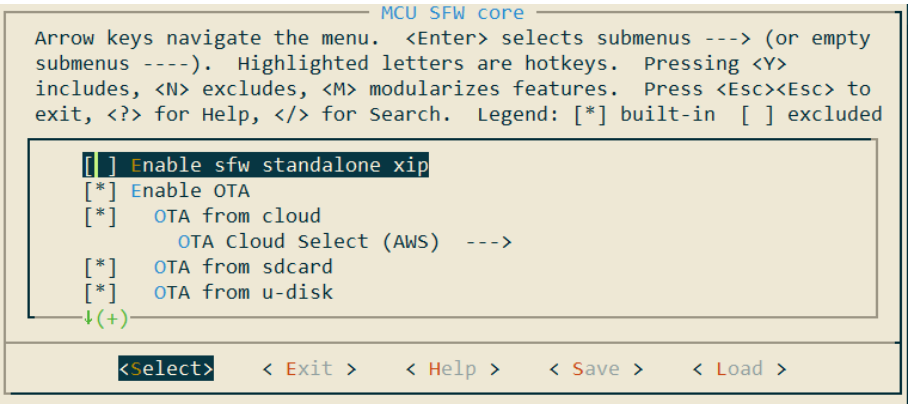


Figure 7-131: Uncheck enable sfw standalone

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

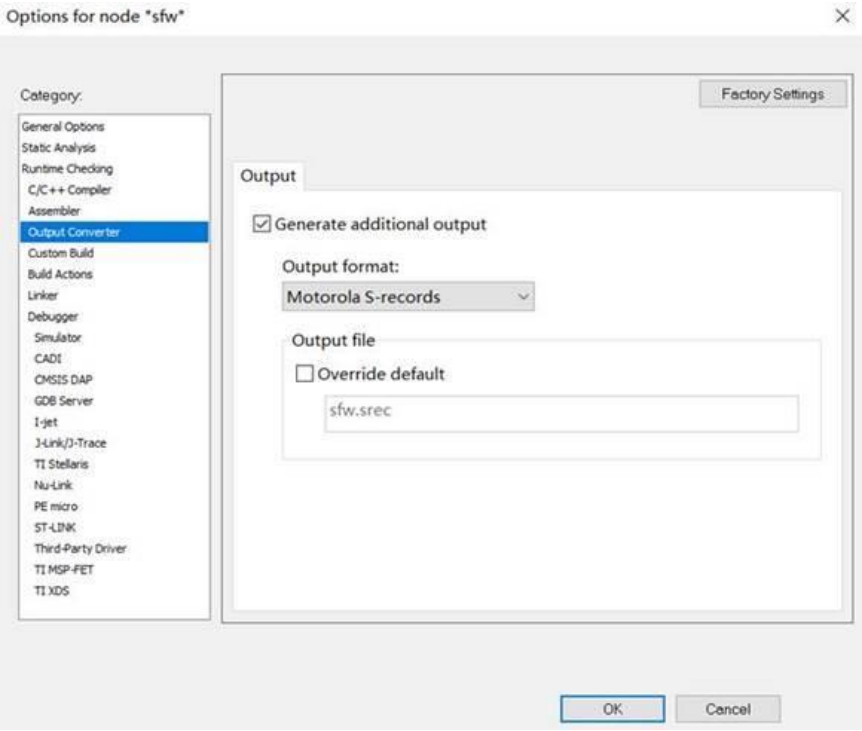


Figure 7-132: Output binary format

Note: For other signing method, user should generate binary format application.

7.4.1.4. Program OCOTP (eFuse)

Below is an example to program SRK table and enable HAB closed mode. **In Development phase, the device may be under HAB open mode for most use cases**.

1. Find and open 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 file SRK_fuses.bin in hex mode under folder gen_hab_certs

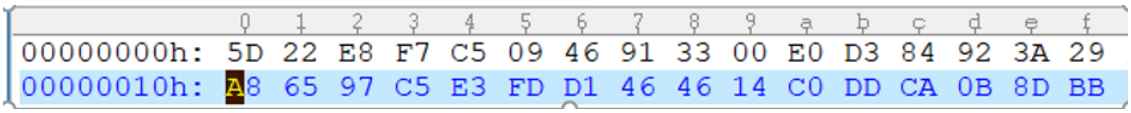


Figure 7-133: SRK hash value in hex

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. Please note the data endianness.

   If user program efuses with 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. User needs to enable it by hand. For other platform, eFuse OCOTP index of SRK hash table may be different. Please refer to the fuse map for the NXP processor which 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. User can enable HAB close mode using following command. In Production phase, user should enable HAB closed mode and must sign SBL image.

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

5. User can 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

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

1. Copy sbl.srec and sfw.srec to folder *sbl/target/evkmimxrt1060/secure*. If user select RSA or ECDSA signing type, please use sfw.bin here.
2. Make EVKMIMXRT1060 board to enter Serial Downloader mode and connect USB OTG and DEBUG USB port
3. Enter folder secure in scons environment by input `cd secure`
4. Run sign_sbl_app.bat to generate final signed SBL and application. If user wants to generate the application image for next update, remember to change the parameter version number in imgtool.py command line.
5. Download signed SBL and application by script sign_sbl_app.bat or other tools. This script will download the application image into slot1. If user 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, user will see output in terminal

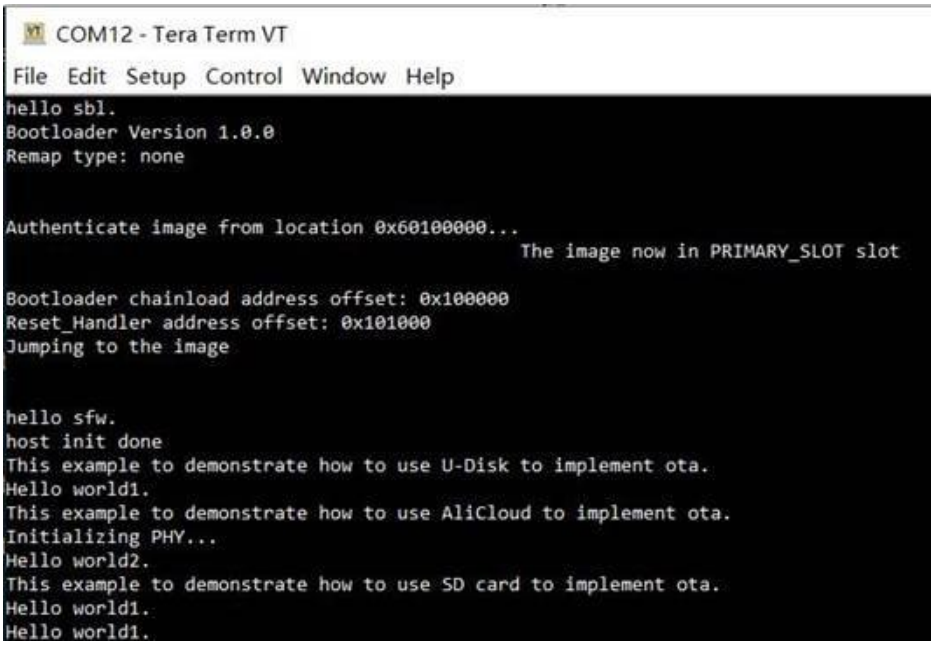


Figure 7-134: Terminal log

7.4.2. Encrypted XIP boot demonstration for 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 target board
2. Run `scons --menuconfig` to enter menu "MCU SBL Core" to select "Enable ROM to verify sbl"

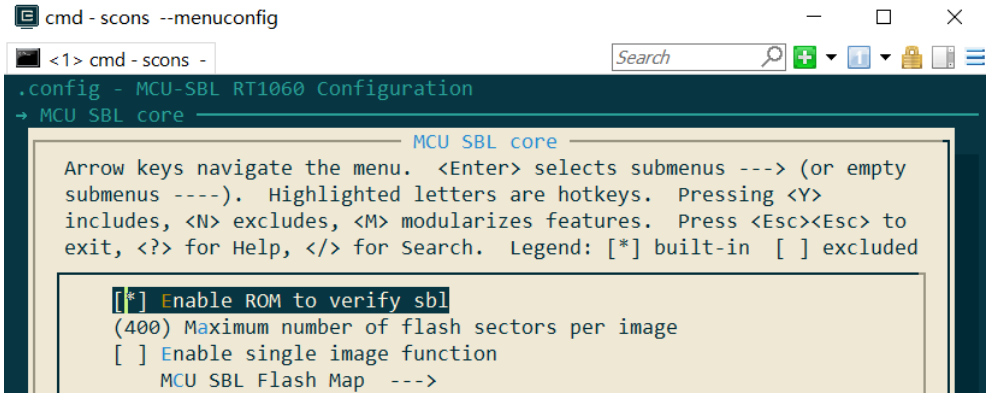


Figure 7-135: Enable ROM to verify sbl

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

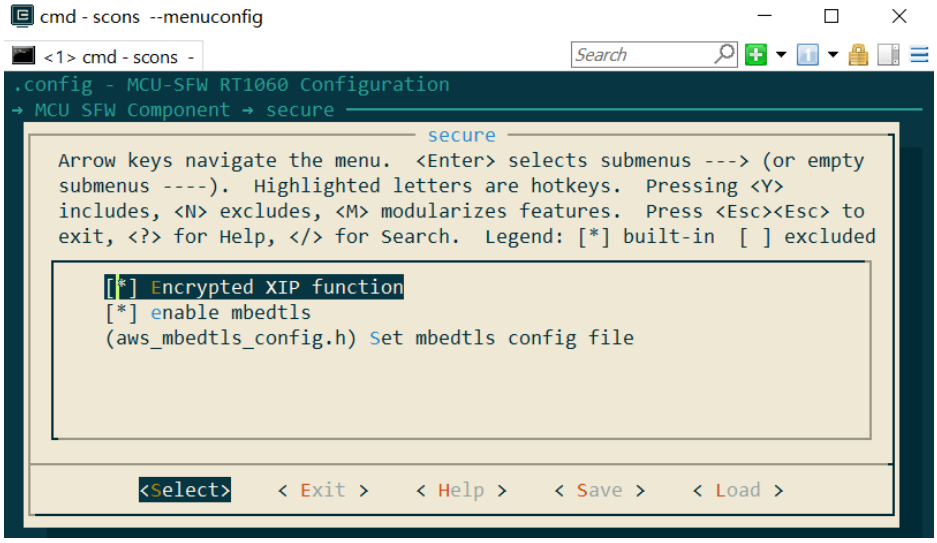


Figure 7-136: Enable key context upgrade in SBL

4. Run `scons --ide=iar` command to generate IAR project
5. Configure project to generate an image with .srec format, then build the project. For Keil project, user may generate 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 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 target platform

3. Enter menu "MCU SFW core" and uncheck menu "Enable sfw standalone xip"
4. Enter menu "MCU SFW component > secure" and check menu "Encrypted XIP function", then save and quit menuconfig

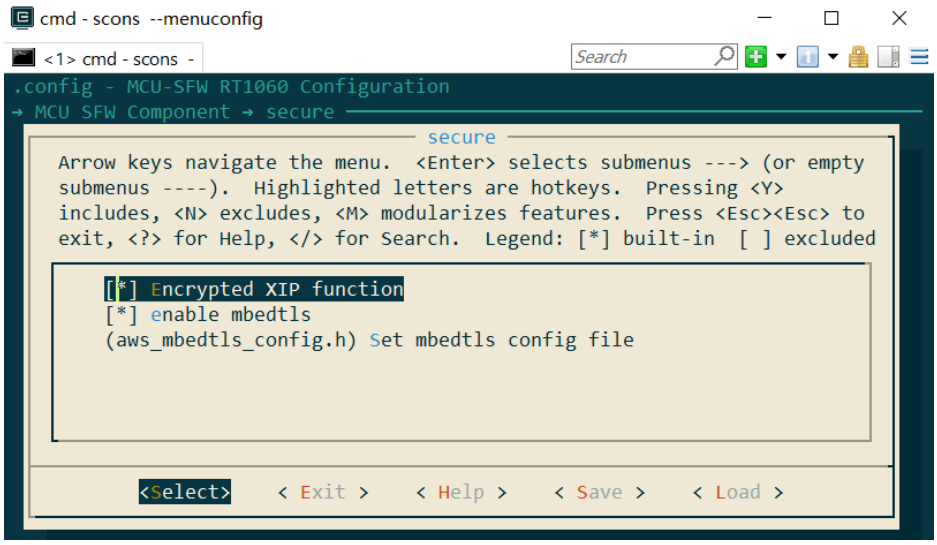


Figure 7-137: Enable key context upgrade in SFW

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

User may not write KEK and keep that efuse value as all 0s in test phase. Below is an example to burn KEK to efuse SW_GP2 for EVKMIMXRT1060, user can add below command into 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 EVKMIMXRT1010 or EVKMIMXRT1170 used, user should 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 folder *sbl/target/evkmimxrt1060/secure*
2. Edit sign_enc_sbl_app.bat and set the KEK and encrypted region

3. Make EVKMIMXRT1060 board to enter Serial Downloader mode and connect USB OTG and 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, user will see output in terminal

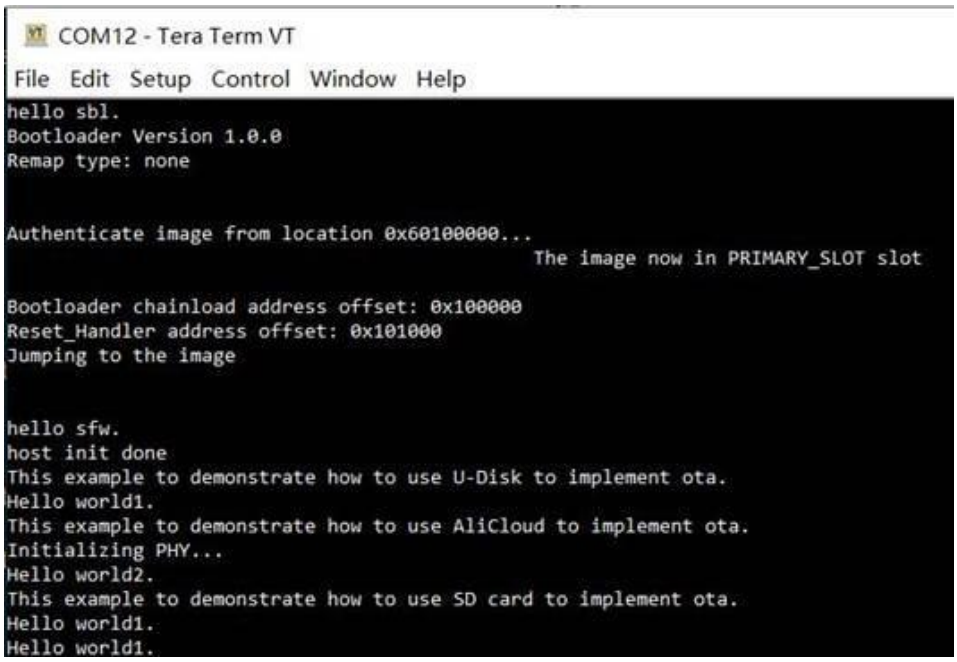


Figure 7-138: Terminal log

7.4.2.5. Application OTA image preparation

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

1. Generate project as 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 final image

7.4.3. Secure boot demonstration for platform LPC55S69

This section describes the steps to enable 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, user need to generate keys and certificates, do the following 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

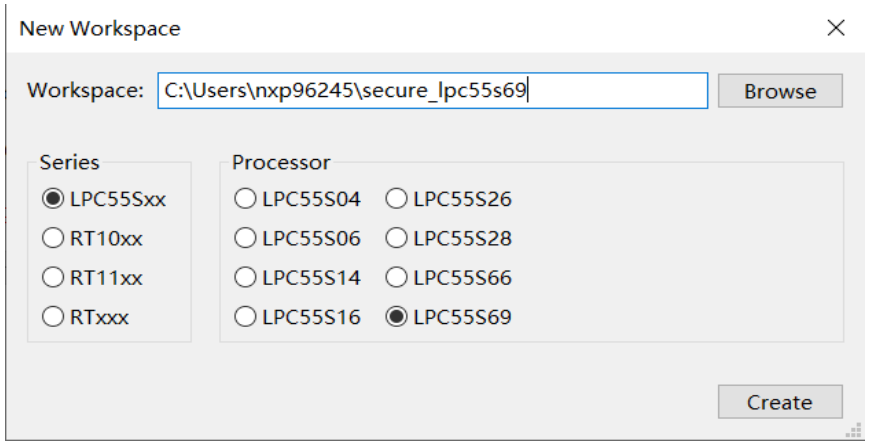


Figure 7-139: Creating a new workspace

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

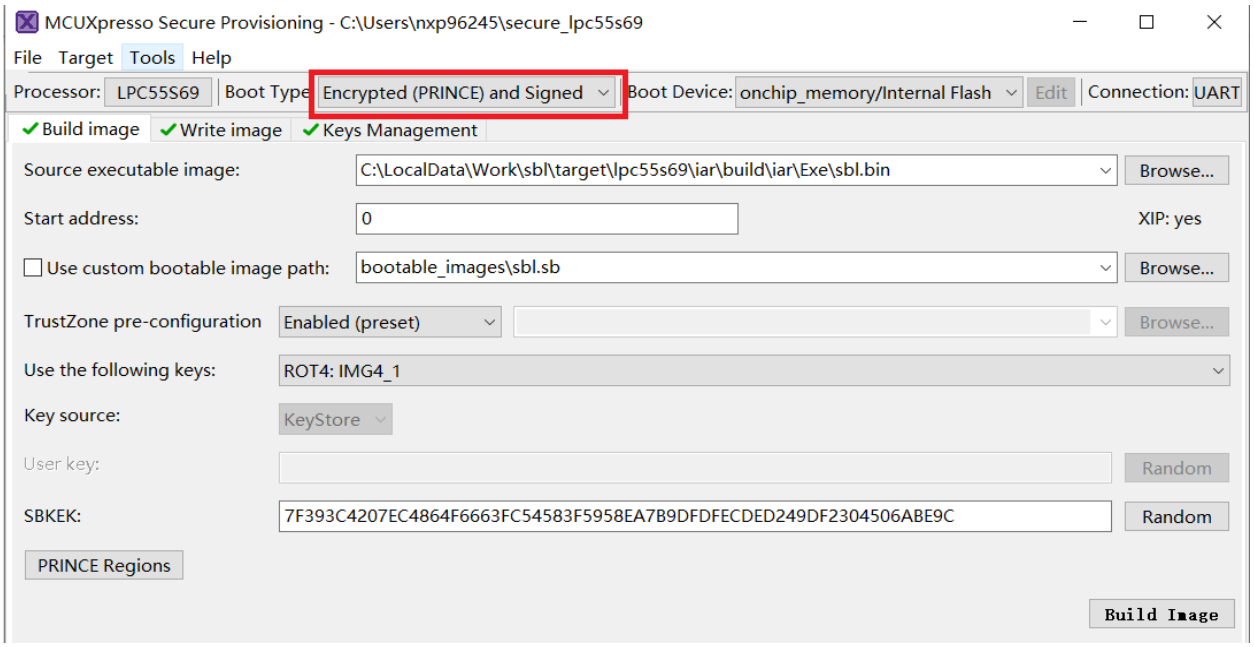


Figure 7-140: Selecting Boot Type

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

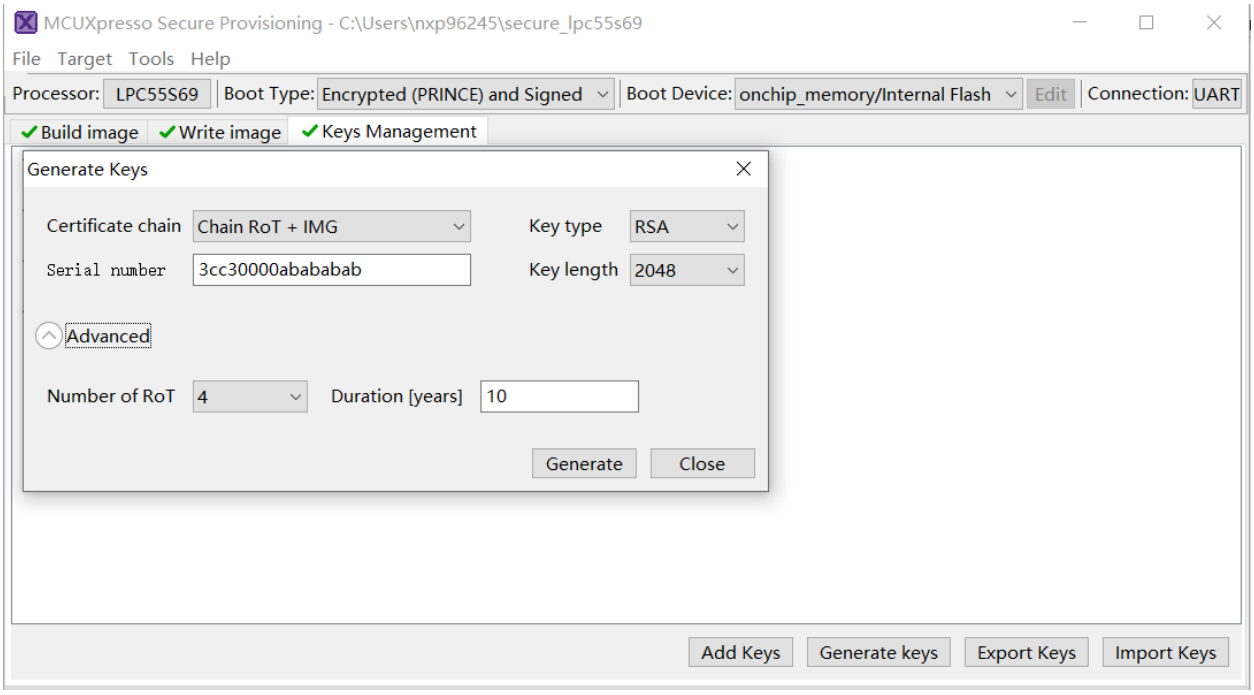


Figure 7-141: Generate keys

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

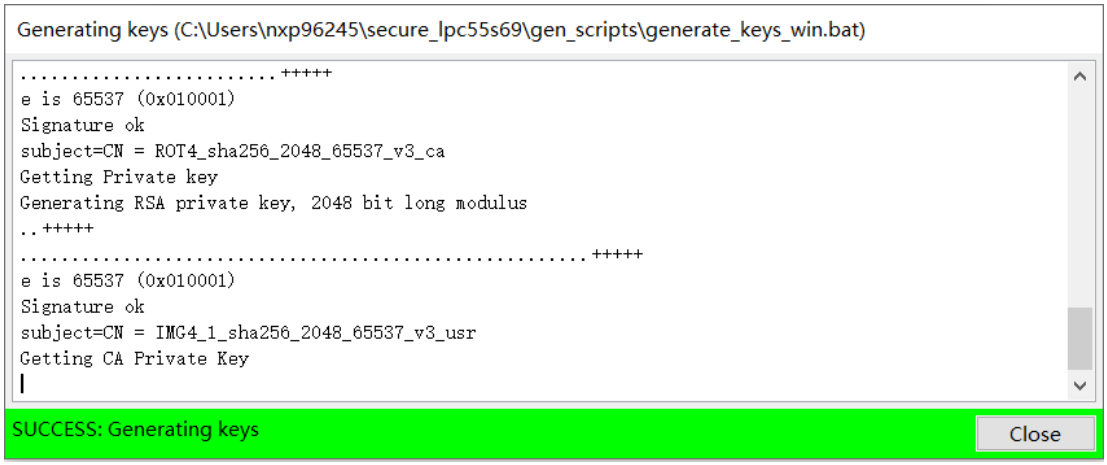


Figure 7-142: Generating keys - success

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

7.4.3.2. SBL image preparation

The following steps describe the procedure for creating SBL image.

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

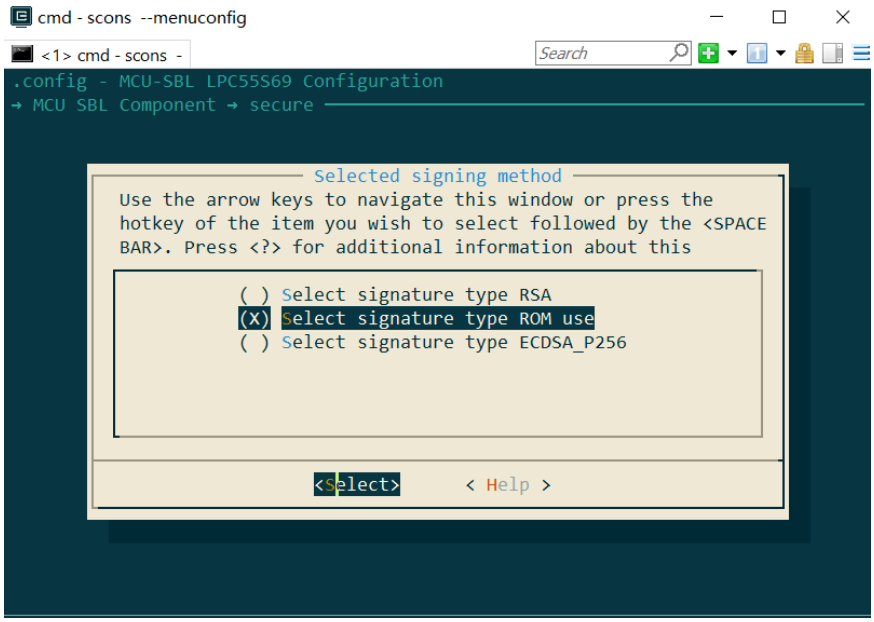


Figure 7-143: Selecting signing method

3. Run `scons --ide=iar` command to generate IAR project
4. Build binary 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 menu "MCU SFW core" and uncheck menu "Enable sfw standalone xip", save and quit menuconfig

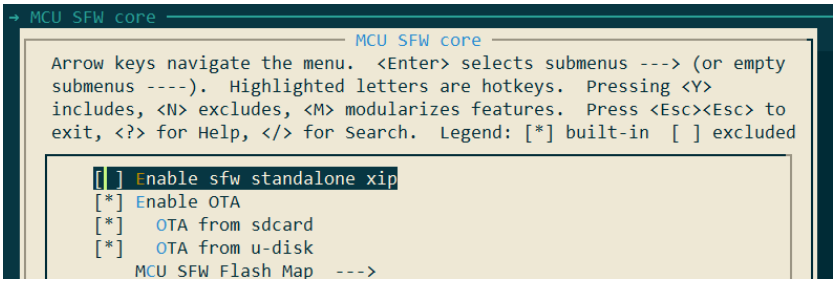


Figure 7-144: Uncheck Enable sfw standalone

3. Generate project, below is an example to generate IAR project

   ```
   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 **Random** button

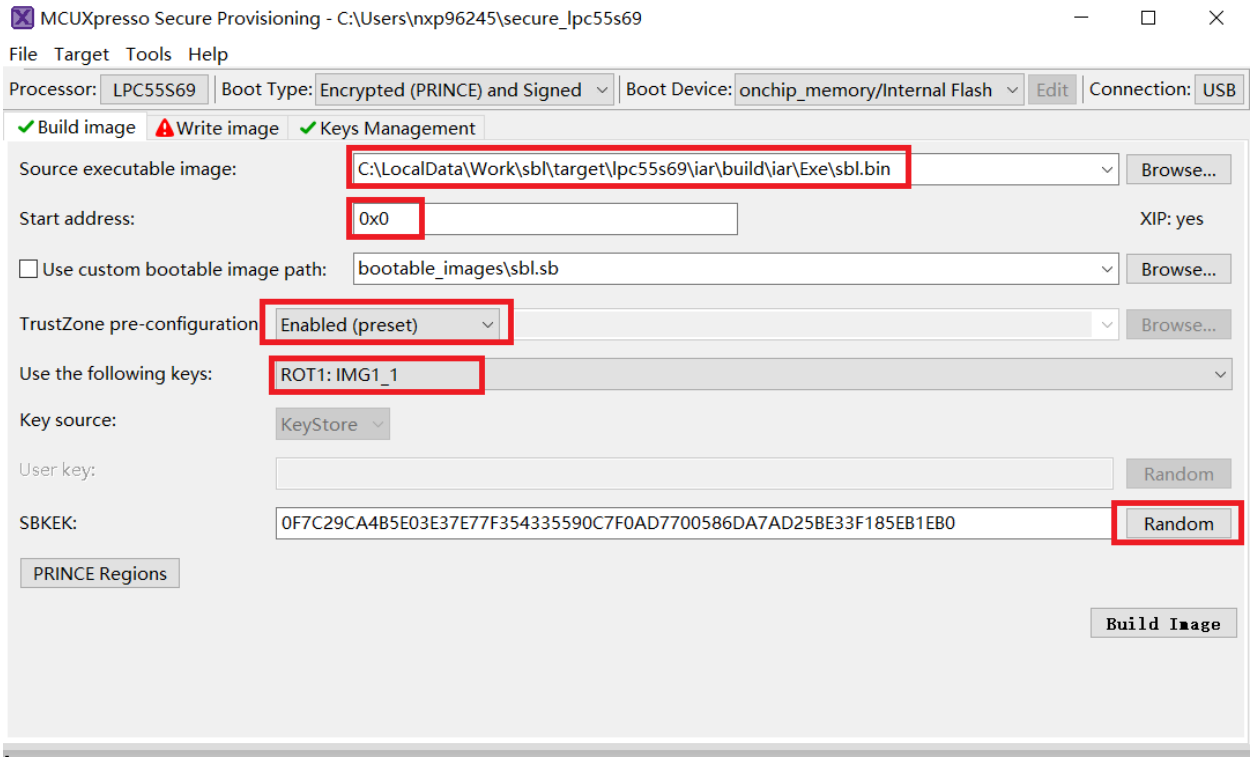


Figure 7-145: Setting parameters for building image

6. Open the PRINCE configuration and check the configuration. Set the encrypted size of the SBL image in PRINCE region. If user don't encrypt the image, please skip this step. And select Signed in Boot Type.

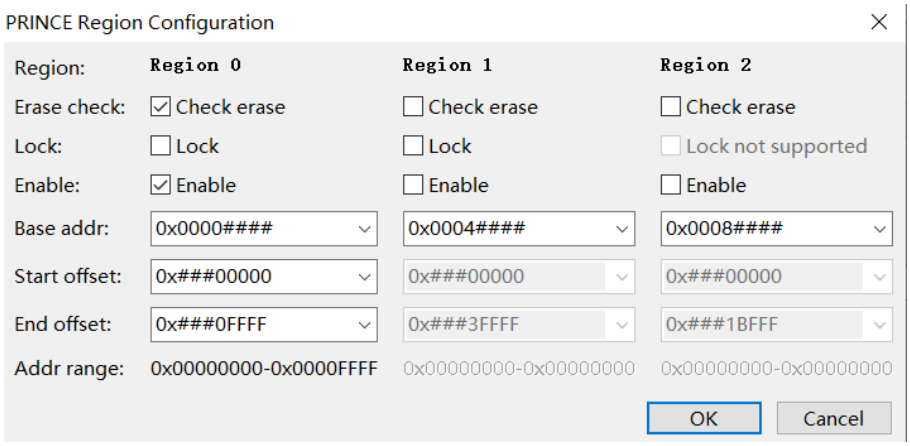


Figure 7-146: Setting encryption region

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

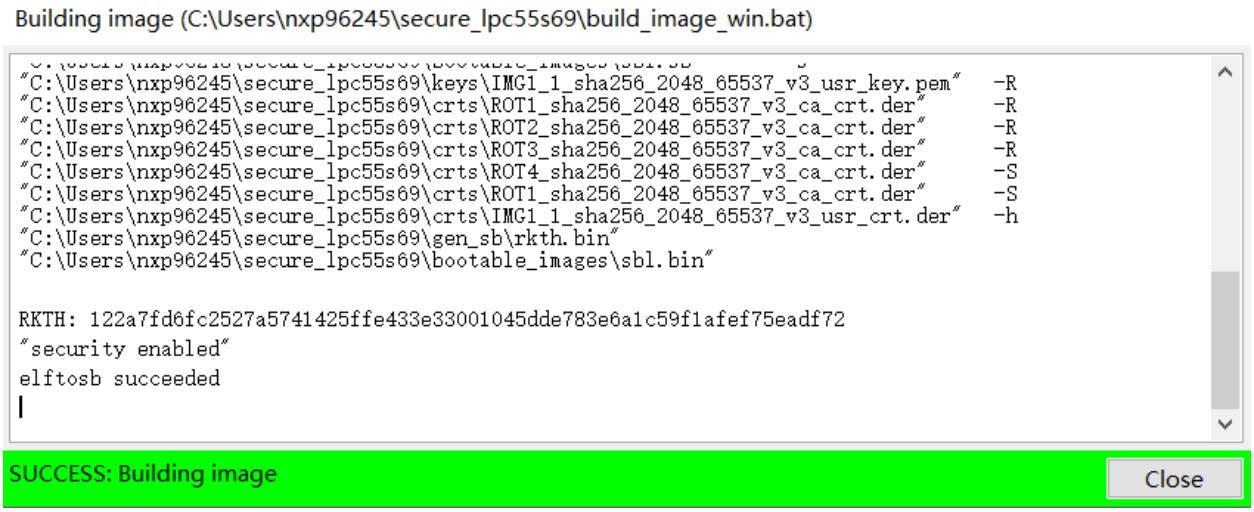


Figure 7-147: Generate booting image

8. Make sure the board is connected and make the processor into ISP mode by pressing 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

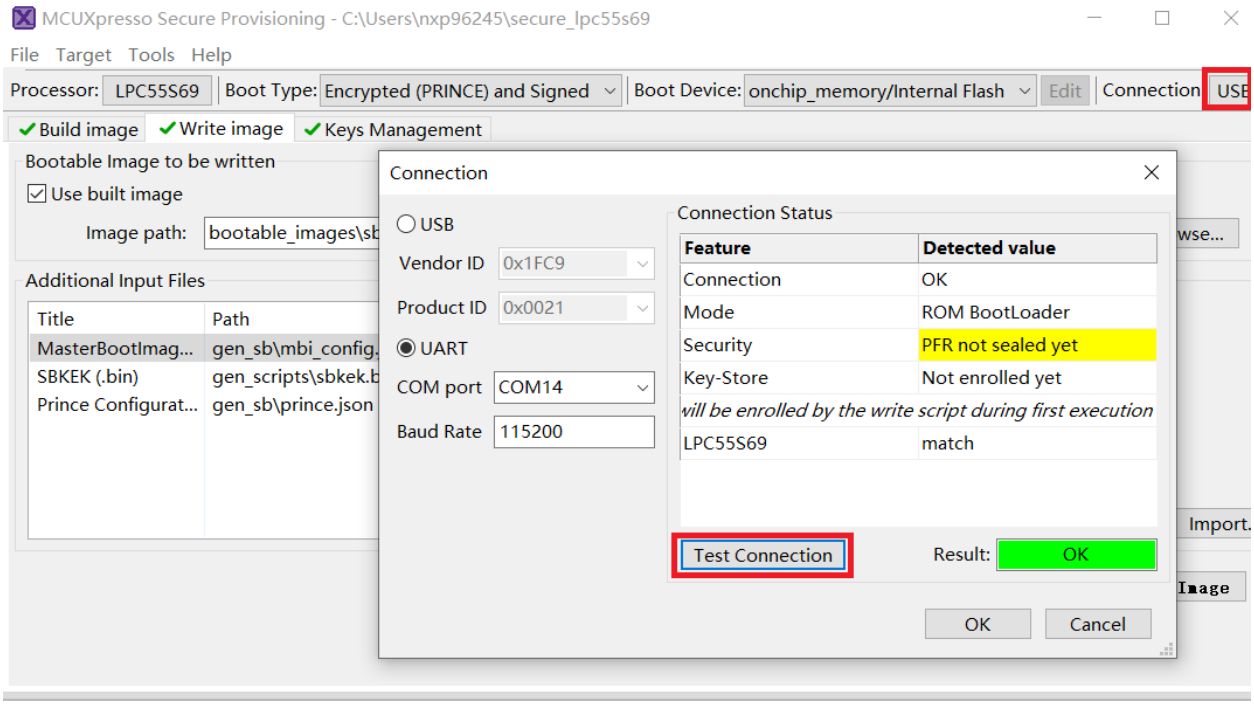


Figure 7-148: Connection dialog

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

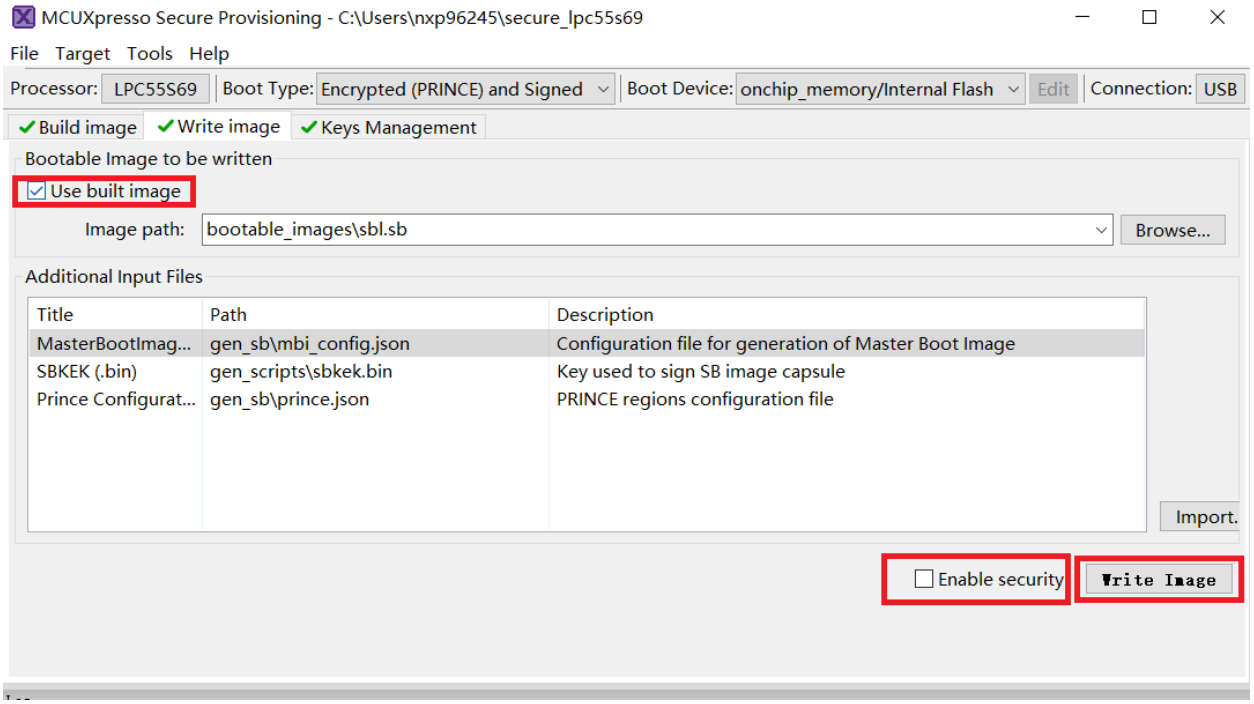


Figure 7-149: Programming image SBL

Note: check **Enable security** checkbox to permanently seal the device security

12. Write result will be displayed in the progress window.

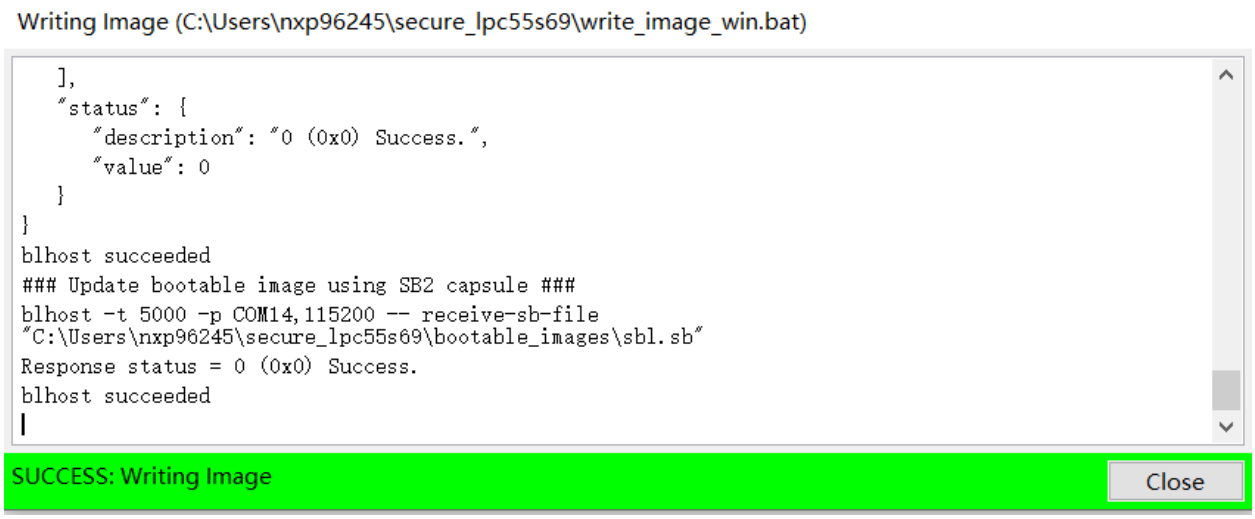


Figure 7-150: Write image message - success

7.4.3.5. Sign and Program application image

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

1. Copy sfw.bin to folder *sbl/target/lpc55s69/secure*
2. Edit mbi_config.json to set correct keys and certifications path or copy folder 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 input `cd secure`
6. Run sign_enc_sfw.bat to generate final signed application image
7. Press any key to configure PRINCE region to set encrypted region after generating signed image, or user can omit this step and download the signed image
8. Make LPC55S69 EVK board to enter ISP mode
9. Download image, if the write operation was successful, reset the board

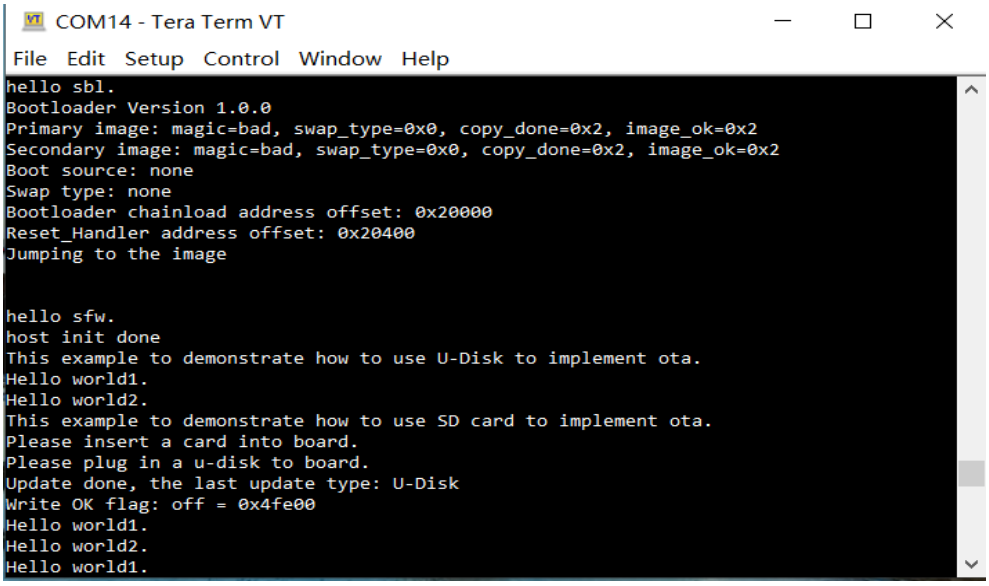


Figure 7-151: Terminal log

Note: For whole encrypted application image, user should pad image with 8k bytes aligned.

7.4.4. Secure boot demonstration for platform EVKMIMXRTxxx

This section describes the steps to enable **XIP** encrypted authenticated boot. The demo targeted for 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, user need to generate keys and certificates, do the following 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

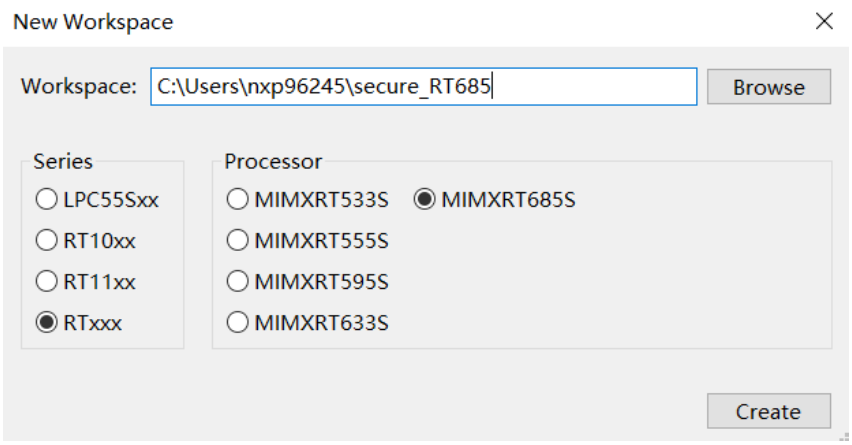


Figure 7-152: Creating a new workspace

3. Choose **Boot Type** as **Signed**

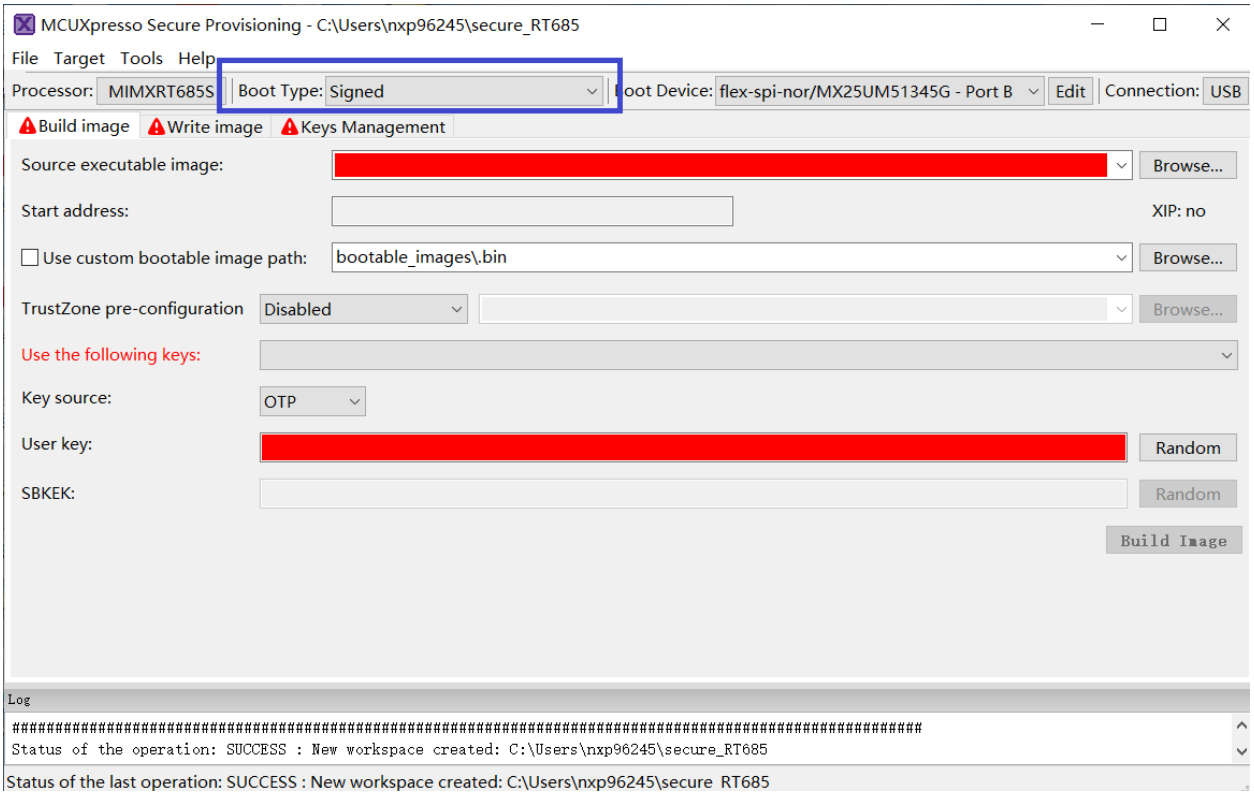


Figure 7-153: Selecting Boot Type

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

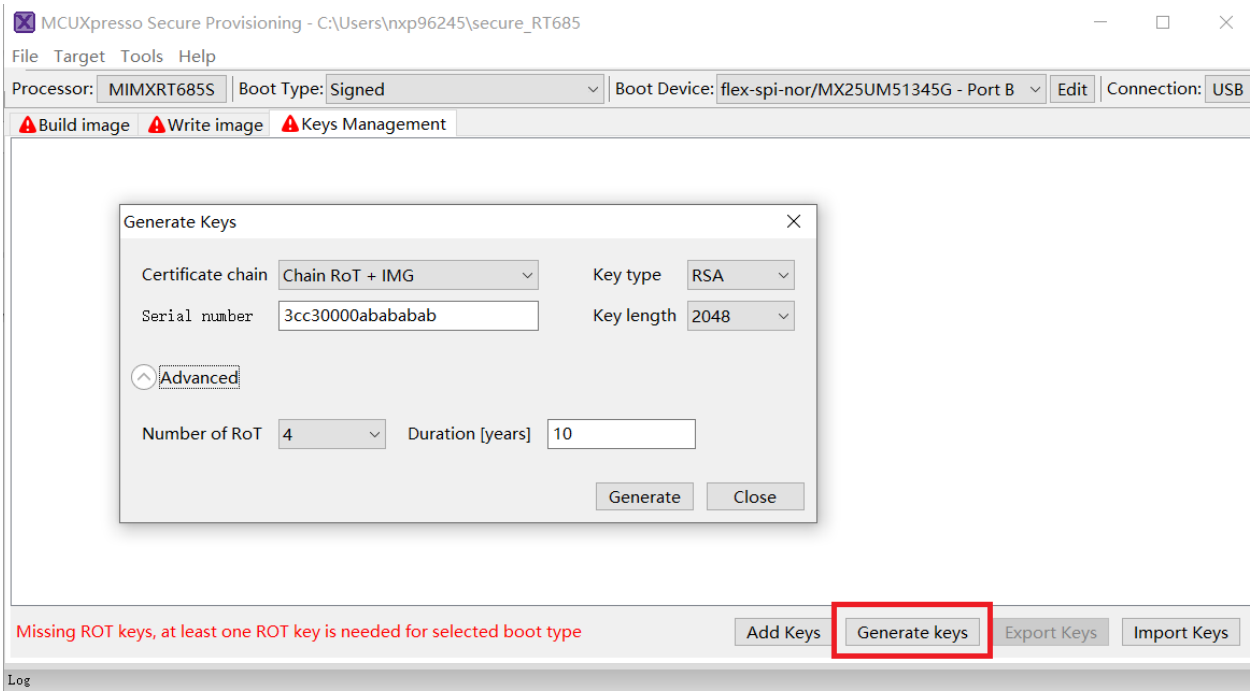


Figure 7-154: Generate keys

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

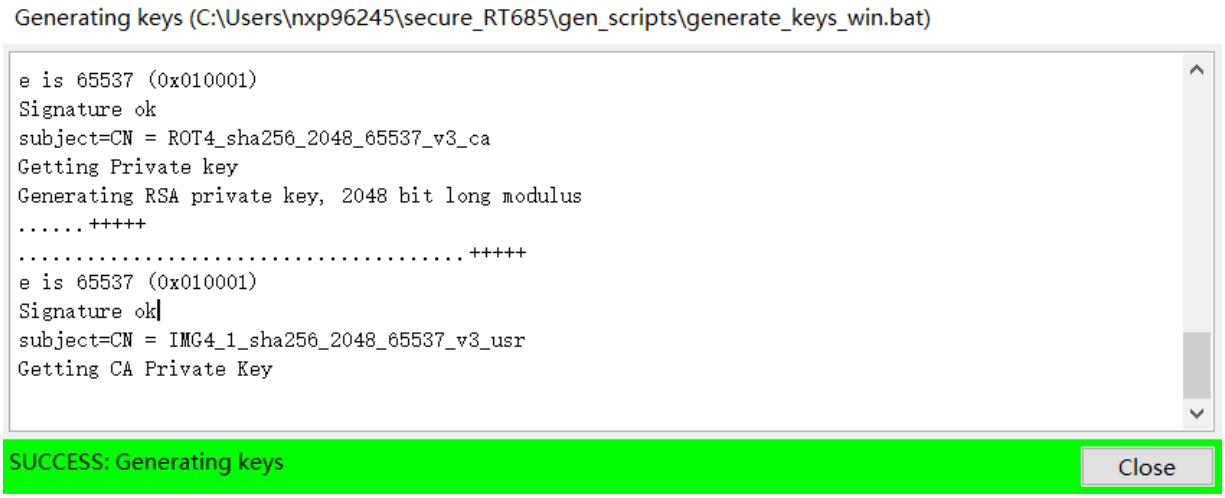


Figure 7-155: Generating keys - success

6. User can find generated keys and certificates in 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 menu "MCU SBL Core" to select "Enable ROM to verify sbl"

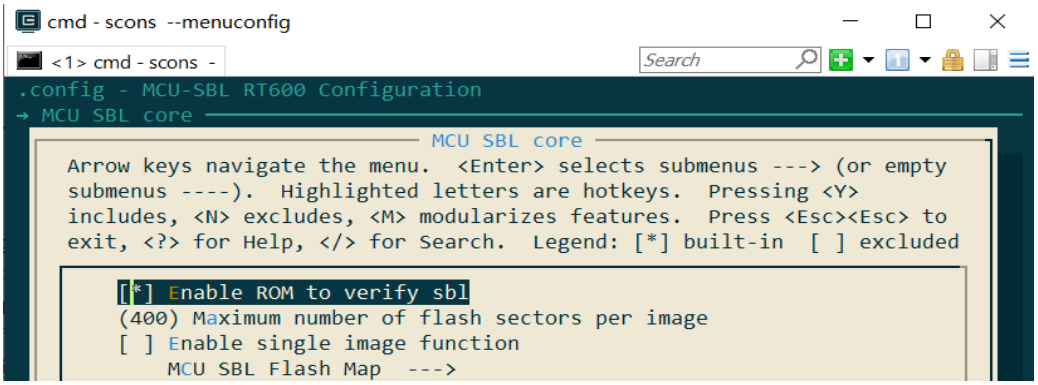


Figure 7-156: Enable ROM to verify sbl

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

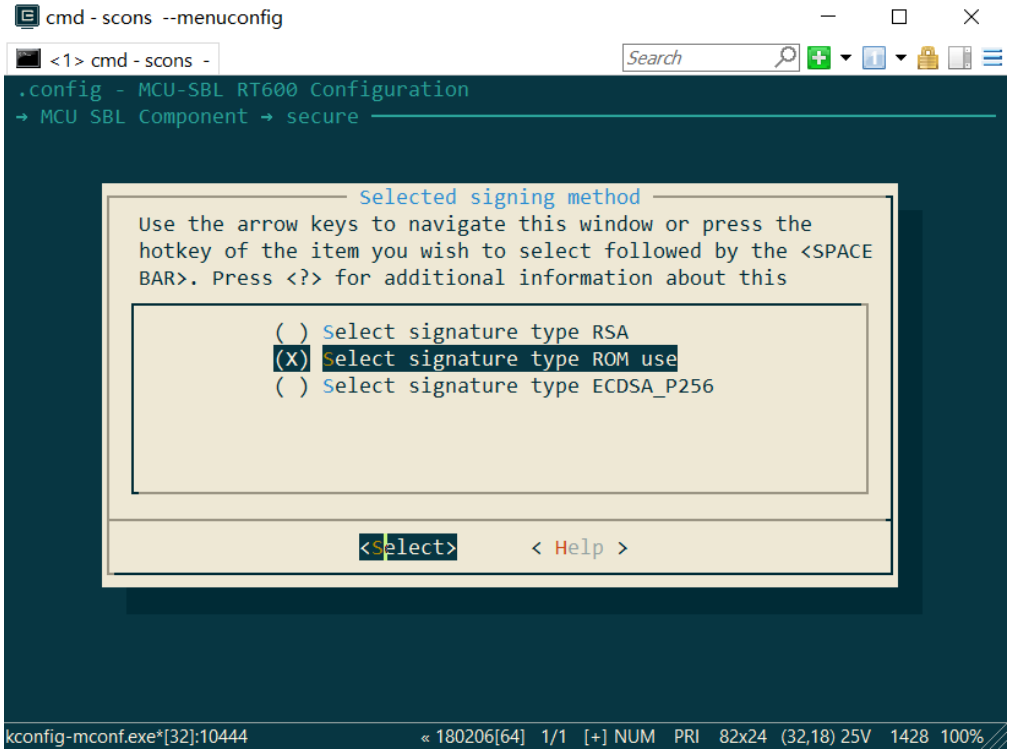


Figure 7-157: Selecting signing method

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

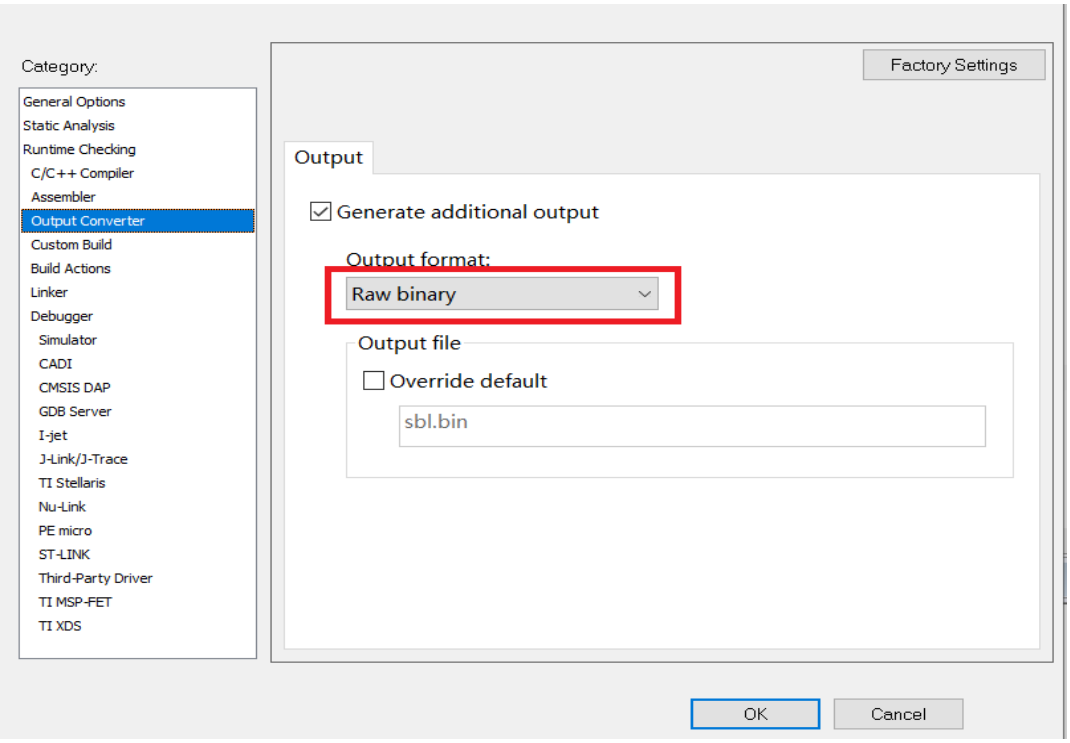


Figure 7-158: Set output format

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"

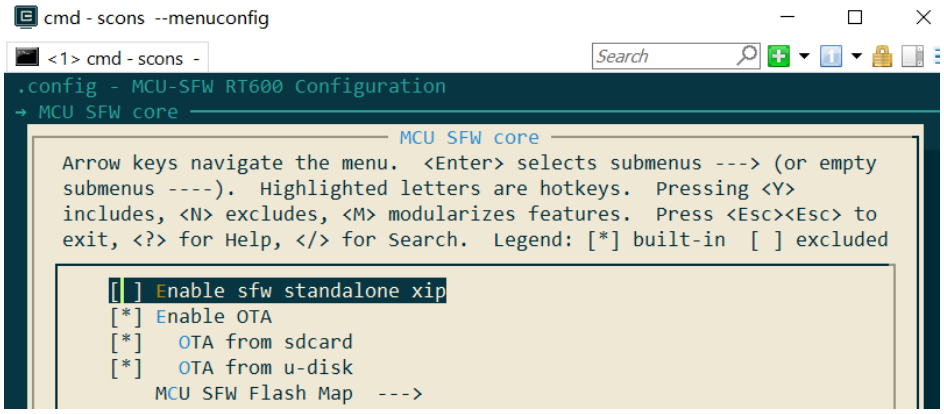


Figure 7-159: Uncheck enable sfw standalone

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

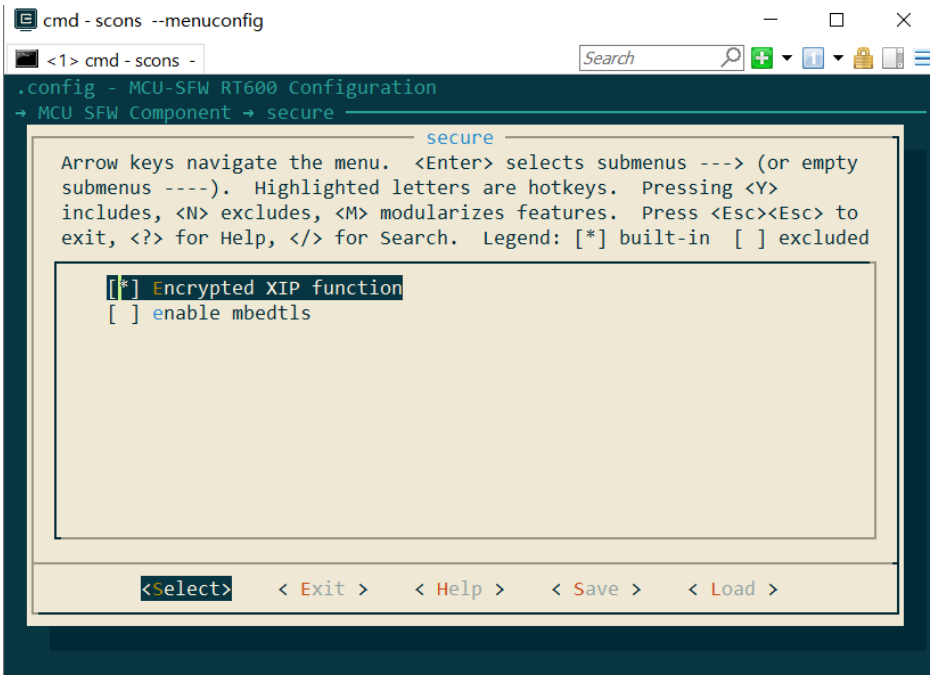


Figure 7-160: Enable key context upgrade in sfw

5. Generate the SFW project
6. Configure 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 folder *sbl/target/ evkmimxrt600/secure*
2. Make 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 input `cd secure`

5. Run sign_sbl_app.bat to generate final signed SBL and application. If user wants to generate the application image for next update, remember to change the parameter version number in imgtool.py command line
6. Download signed SBL and application by script sign_sbl_app.bat or other tools. This script will download the application image into slot1. If user 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, user will see output in terminal

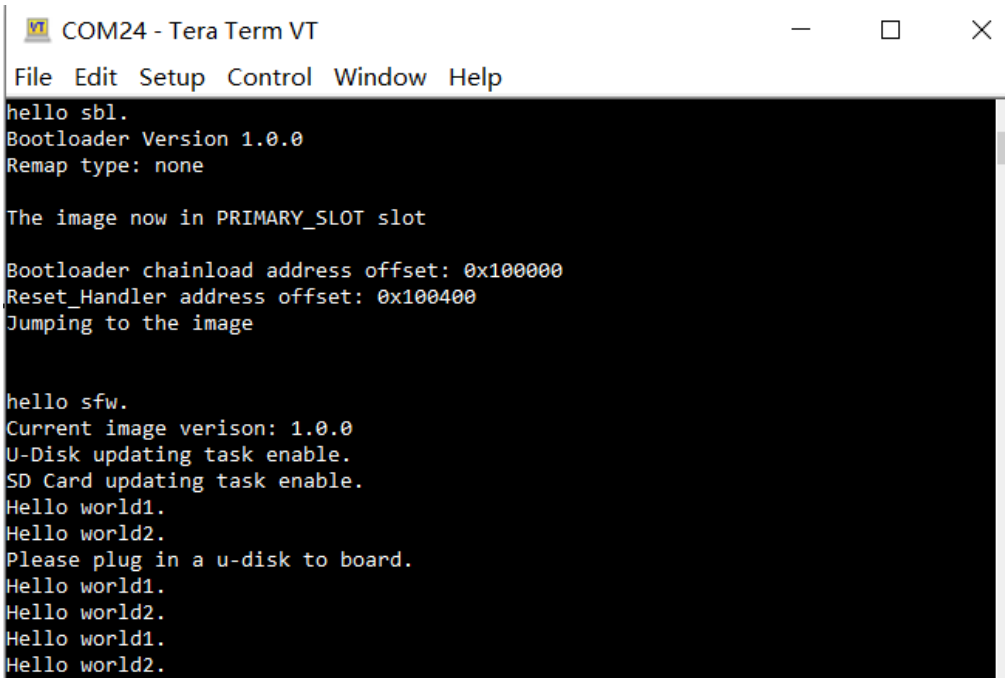


Figure 7-161: Terminal log

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 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 input `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, user will see output in terminal

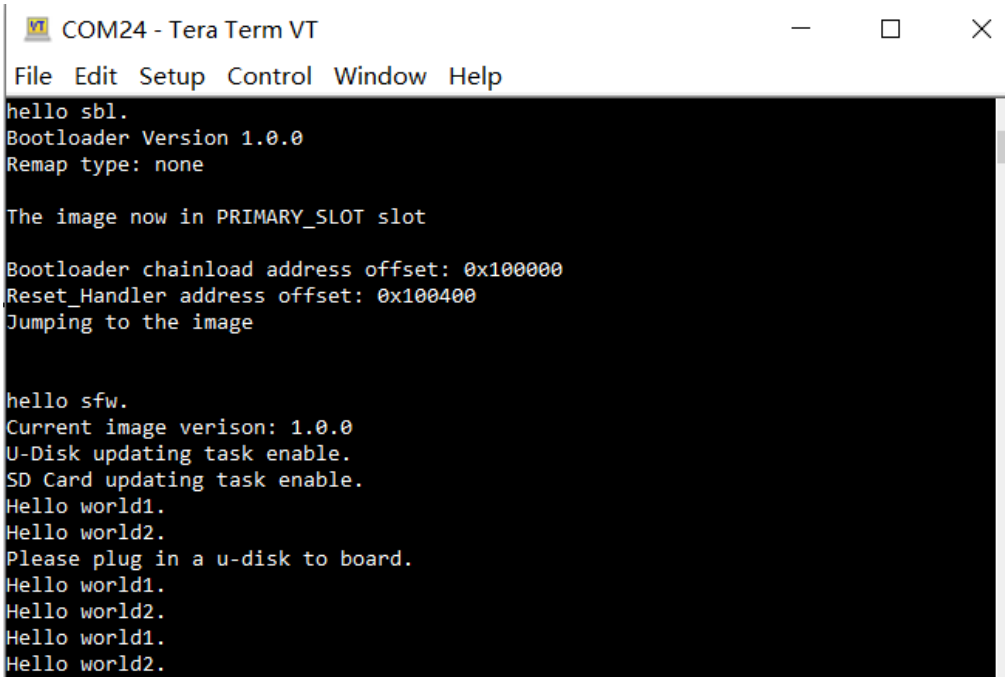


Figure 7-162: Terminal log

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 7-163: 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. e.g. 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 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. User should 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 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 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

8. Known issues

The following table lists the known issues of this release.

Table 8-1 Known issues

Item	Description
1	
2	
3	

9. Revision History

Revision	Date	Description
0.1	4/12/2020	Initial internal release
1.0	8/26/2021	First release to open source
1.1.0	11/3/2021	1. Support RT500, RT600 signature and encryption feature 2. Optimize the revert flow