Renesas RA Family

RA6 Secure Bootloader Using MCUboot with Protected mode and Internal Code Flash

Introduction

MCUboot is a secure bootloader for 32-bit MCUs. It defines a common infrastructure for the bootloader, defines system flash layout on microcontroller systems, and provides a secure bootloader that enables easy software update. MCUboot is independent of operating system and hardware and relies on hardware porting layers from the operating system it works with. The Renesas Flexible Software Package (FSP) integrates an MCUboot port starting from FSP v3.0.0. Users can benefit from using the FSP MCUboot Module to create a Root of Trust (RoT) for the system and perform secure booting and fail-safe application updates.

MCUboot is maintained by TrustedFirmware in the GitHub mcu-tools page https://github.com/mcu-tools/mcuboot. There is a `\docs` folder that holds the documentation for MCUboot in .md file format. This application note refers to the above-mentioned documents wherever possible and is intended to provide additional information that is related to using the MCUboot Module with Renesas RA FSP v3.0.0 or later.

RA MCUs support the use of MCUboot in Compatibility Mode as a secure bootloader solution, enabling firmware authentication and update capabilities with minimal integration effort. In addition to Compatibility Mode, RA MCUs also support Protected Mode with MCUboot, which provides enhanced security by ensuring that any cryptographic keys used are securely provisioned for the specific device. In Protected Mode, the public key used for firmware authentication must be injected through a secure key injection process. This adds a strong layer of protection against unauthorized firmware updates and key tampering.

When using the MCUboot module with internal flash in Code Flash Linear Mode on RA6 Family MCUs, users can refer to application project R11AN0497 for implementation guidance.

This application note provides step-by-step instructions for creating an application project using the MCUboot Module in Protected Mode on the Renesas EK-RA6M4 board with FSP v6.1.0. Users can follow the provided steps to recreate the bootloader and link the application projects to work with the bootloader through using FSP Solution. It also includes procedures for mastering an application to work with the bootloader, and for downloading and updating a new application.

Required Resources

Development tools and software

- The e^2 studio IDE v2025-07
- Renesas Flexible Software Package (FSP) v6.1.0
- SEGGER J-link® USB driver

Note: The above three software components are bundled in a downloadable platform installer available on the FSP webpage at renesas.com/ra/fsp.

- Python v3.12 or later (https://www.python.org/downloads/)
- Openssl: v3.0.12 or later (OpeenSSLwebsite: https://www.openssl.org/, OpenSSL App for Windows OS: https://slproweb.com/products/Win32OpenSSL.html)
- Renesas Flash Programmer (RFP) v3.20
- Renesas Security Key Management Tool v1.10
- Kleopatra v3.3.0 (based on Gpg4win-4.4.0, can be downloaded from http://www.gpg4win.org/)

Hardware

- EK-RA6M4 Evaluation Kit for RA6M4 MCU Group (http://www.renesas.com/ra/ek-ra6m4)
- Workstation running Windows® 10 or Windows® 11
- One USB device cable (type-A male to micro-B male)

Prerequisites and Intended Audience

This application note assumes that you have some experience with the Renesas e^2 studio. In addition, you should read the entire MCUboot Port section of the FSP User's Manual prior to moving forward with this application project. This application project also assumes that you have some knowledge of cryptography.

The intended audience includes product developers, product manufacturers, product support, and end users who are involved in designing application systems involving the use of a secure bootloader.

Contents

1. Overview of MCUboot

1.1 History of MCUboot

MCUboot evolved out of the Apache Mynewt bootloader, which was created by runtime.io. MCUboot was then acquired by JuulLabs in November 2018. The MCUboot github repo was later migrated from JuulLabs to the mcu-tools github project. In 2020, MCUboot was moved under the Linaro Community Project umbrella as an open-source project, and it now resides under TrustedFirmware (www.trustedfirmware.org/projects/mcuboot).

1.2 MCUboot Functionalities Overview

MCUBoot handles the firmware authenticity check after startup and the firmware switch stage of the firmware update process. Downloading the new version of the firmware is out-of-scope for MCUboot. Typically, downloading the new version of the firmware is functionality that is provided by the application project itself.

1.2.1 Validate Application before Booting and Updating

For applications using MCUboot, the MCU memory is separated into MCUboot, Primary App, Secondary App, and the Scratch Area. Figure 1 is an example of the single-image MCUboot memory map. For more information on the MCUboot memory layout, refer to the Flash Map section of the MCUboot website.

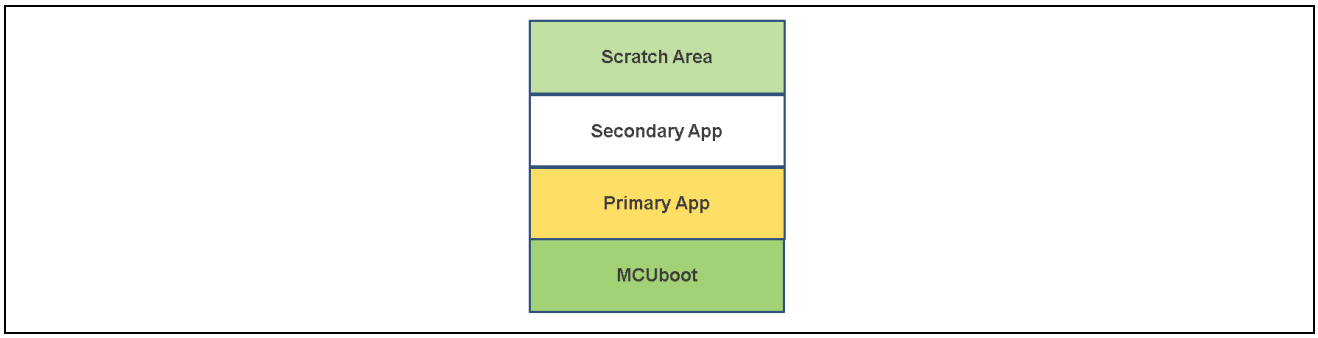


Figure 1. Single Image MCUboot Memory Flash Map

The functionality of the MCUboot during booting and updating follows the process below:

The bootloader starts when the CPU is released from reset. For TrustZone-based MCUs, MCUboot is designed to run in secure mode with all access privileges available to it. If there are images in the Secondary App memory marked as to be updated, the bootloader performs the following actions:

1. The bootloader authenticates the Secondary App image.
2. Upon successful authentication, the bootloader switches to the new image based on the update method selected. Available update methods are introduced in section 1.2.2.
3. The bootloader boots the new image.

If there is no new image in the Secondary App memory region, the bootloader authenticates the Primary applications and boots the Primary image.

The authentication of the application is configurable in terms of the authentication methods and whether the authentication is to be performed with MCUboot. If authentication is to be performed, the available methods are RSA or ECDSA. The firmware image is authenticated by hash (SHA-256) and digital signature validation. The public key used for digital signature validation can be built into the bootloader image or provisioned into the MCU during manufacturing. In the examples included in this application project, the public key is built into the bootloader images.

There is a signing tool included with MCUboot: `imgtool.py`. This tool provides services for creating Root keys, key management, and signing and packaging an image with version controls. Read the MCUboot documentation to use and understand these operations.

1.2.2 Applications Update Strategies

The following are the update strategies supported by MCUboot. The analysis of pros and cons is based on the MCUboot functionality, but not the FSP MCUboot Module functionality. In addition, this application note is not intended to provide all details on the MCUboot application update strategies. We recommend acquiring more details on these update strategies by referring to the MCUboot design page: https://github.com/mcu-tools/mcuboot/blob/master/docs/design.md

- **Overwrite**
  In the Overwrite update mode, the active firmware image is always executed from the Primary slot, and the Secondary slot is a staging area for new images. Before the new firmware image is executed, the entire contents of the Primary slot are overwritten with the contents of the Secondary slot (the new firmware image).
  - Pros
    - Fail-safe and resistant to power-cut failures.
    - Less memory overhead, with a smaller MCUboot trailer and no Scratch Area.
    - Encrypted image support is available when using external flash.
  - Cons
    - Does not support pre-testing of the new image prior to overwriting.
    - Does not support automatic application fallback mechanism.

  Overwrite upgrade mode is supported by Renesas RA FSP v3.0.0 or later. External flash memory support is supported by FSP v3.5.0 or later.

- **Swap**
  In the Swap image upgrade mode, the active image is also stored in the Primary slot and is always started by the bootloader. If the bootloader finds a valid image in the Secondary slot that is marked for upgrade, then contents of the Primary slot and the Secondary slot are swapped. The new image then starts from the Primary slot. Upgrading an old image with a new one by swapping can be a two-step process. In this process, MCUboot performs a “test” swap of image data in flash and boots the new image. The new image can then update the contents of flash at runtime to mark itself “OK”, and MCUboot will then still choose to run it during the next boot.
  - Pros
    - The bootloader can revert the swapping as a fallback mechanism to recover the previous working firmware version after a faulty update.
    - The application can perform a self-test to mark itself permanent.
    - This image upgrade mode is fail-safe and resistant to power-cut failures.
    - Encrypted image support is available when using external flash.
  - Cons
    - Need to allocate a Scratch Area.
    - Larger memory overhead, due to a larger image trailer and additional Scratch Area.
    - Larger number of write cycles in the Scratch Area, thus faster wearing out of Scratch sectors.

  Swap upgrade mode is supported by Renesas RA FSP v3.0.0 or later. Runtime image testing is supported by FSP v3.4.0 or later, excluding v3.5.0. External flash memory support is supported by FSP v3.5.0 or later. The swap update mode with test mode is demonstrated in this example application.

- **Direct execute-in-place (DXIP)**
  In the direct execute-in-place mode, the active image slot alternates with each firmware update. If this update method is used, then two firmware update images must be generated: one of them is linked to be executed from the Primary slot memory region, and the other is linked to be executed from the Secondary slot.
  - Pros
    - Faster boot time, as there is no overwrite or swap of application images needed.
    - Fail-safe and resistant to power-cut failures.
  - Cons
    - Added application-level complexity to determine which firmware image needs to be downloaded.
    - Encrypted image support is not available.

  Direct execute-in-place mode is enabled in FSP for the code flash linear mode as well as code flash dual bank mode.

- **RAM loading firmware update**
  Like the direct-XIP mode, RAM loading firmware update mode selects the newest image by reading the image version numbers in the image headers. However, instead of executing it in place, the newest image is copied to RAM for execution. The load address (the location in RAM where the image is copied to) is stored in the image header. This upgrade method is not typically used in an MCU environment. Refer to the RAM Loading section in the MCUboot page for more information on this update strategy. This image update mode does not support encrypted images (see MCUboot documentation on encrypted image operation).
  RAM loading update mode is not supported by the Renesas RA FSP.

2. Overview FSP MCUboot Module with Protected Mode

Renesas RA FSP supports MCUboot with two operating modes: Compatibility Mode and Protected Mode, each offering different features and security levels. This section provides an overview of the FSP MCUboot module, which integrates MCUboot into the FSP— primarily for use in Protected Mode. It covers the available upgrade modes, required FSP configurations and the image signing process.

2.1 FSP MCUboot Module with Protected Mode

FSP MCUboot Module supports Compatibility Mode as the standard operation mode of MCUboot. It functions as a typical bootloader to ensure basic firmware update functionality and is suitable for applications where enhanced security measures are not required.

FSP MCUBoot in Protected Mode offers enhanced security features that improve the overall protection of cryptographic keys and firmware integrity. It provides additional security by ensuring that cryptographic keys that are used were securely provisioned for the specific device. This targeted provisioning reduces the risk of key exposure or misuse across multiple devices, strengthening the overall security of the system by avoiding generic keys.

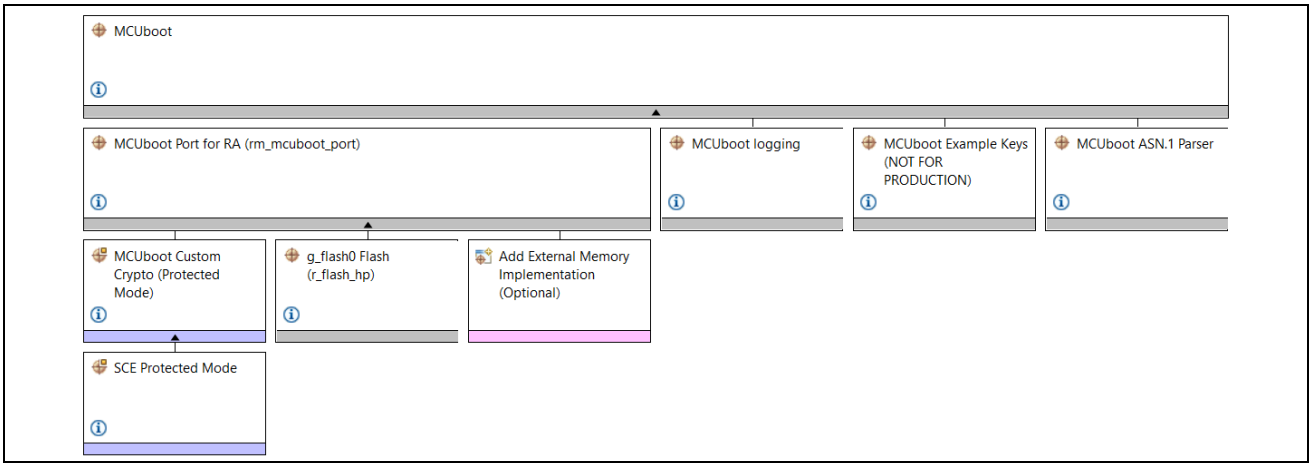


Figure 2. FSP MCUboot Stacks for Protected Mode

Protected Mode offers optimal key protection by using a wrapped key format during key injection process, which appends an MCU-unique MAC to the key. This MAC is checked by the security engine prior to using the key, ensuring that the public key has not been tampered.

A key feature of Protected Mode is the mandatory secure key injection process, which requires the public key used for firmware authentication to be securely injected into the device prior to deployment. This process prevents unauthorized firmware installations, as only firmware signed with the private key that matches the securely injected public key will be accepted and executed. Refer to application project r11an0496 for detailed information about secure keys injection.

Protected Mode also provides advantages in code size efficiency compared to Compatibility Mode, delivering enhanced security features without significantly increasing the footprint on devices with limited memory or storage. This makes it ideal for embedded systems where resource constraints are common. For example, the bootloader for the Swap update mode uses a flash area of around 64 KB when using ECDSA P-256 signature type in Compatibility Mode, but with Protected Mode, the bootloader reduces to about 40 KB.

Protected Mode allows the bootloader to operate securely even when the main application runs in Compatibility Mode. This flexibility enables developers to improve bootloader security without modifying existing application code, offering a practical path for enhancing security in legacy.

Note that there is a limitation in the current system. The ECC plaintext public key needs to be included in the bootloader for the system to function. The MCUboot code checks the hash of this plaintext public key against the hash that is in the image header to confirm the right key to use. The actual signature verification of the image uses the injected ECC public key.

2.2 MCU Memory Configuration using MCUboot Module with FSP

This section provides a high-level overview of the MCUboot Module in the FSP. Currently, the FSP supports four firmware update methods:

- **Overwrite Only**: The entire Primary slot is overwritten with the Secondary slot.
- **Overwrite Only Fast**: Only sizeof(secondary_image) is copied into Primary slot. Unused sectors are not copied.
- **Swap**: The entire Primary and Secondary slots are swapped. A Scratch region is required.
- **Direct XIP**: The new image is run directly from its flash partition.

We recommend reviewing MCUboot port section of the FSP User’s Manual to understand the Build Time Configurations for MCUboot. This section is not meant to cover all the configurable properties. Only some of the most frequently used configuration options are introduced.

2.2.1 General Configuration

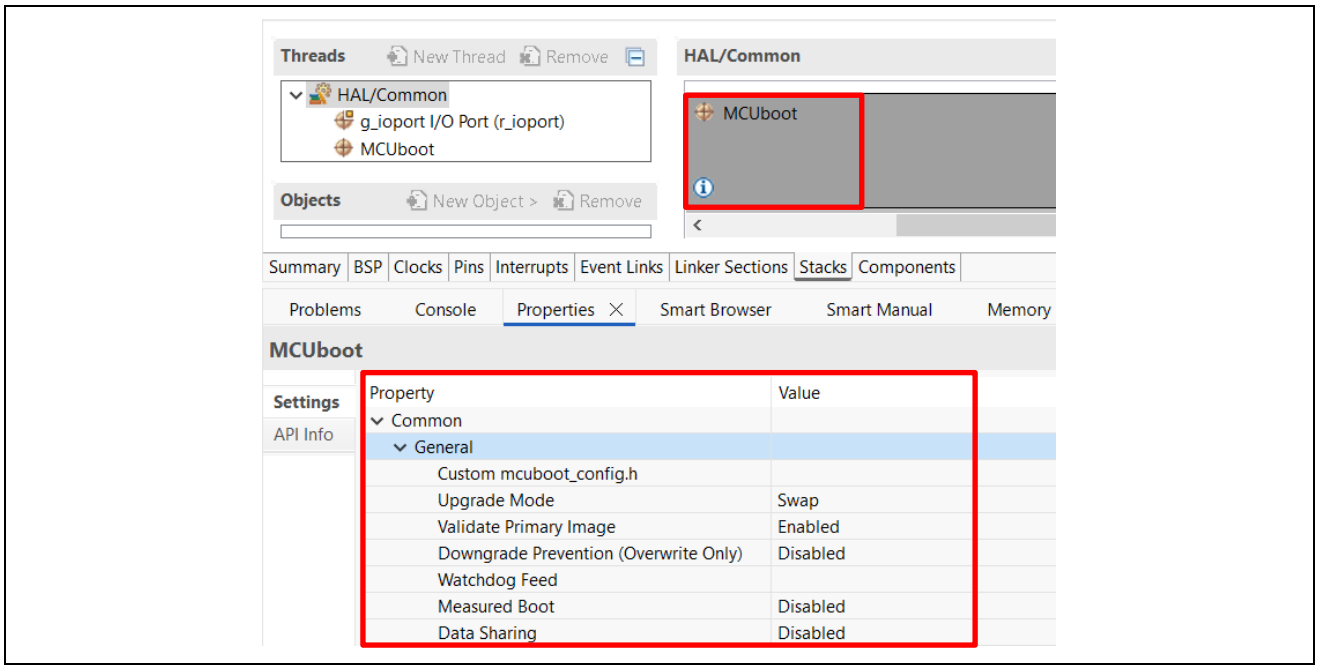


Figure 3. FSP MCUboot Module General Configuration Properties

General configuration properties include:

- **Custom mcuboot_config.h**: The default `mcuboot_config.h` file contains the MCUboot Module configuration that you selected from the RA configurator. You can create a custom version of this file to achieve additional bootloader functionalities available in MCUboot.
- **Upgrade Mode**: This property configures the application image update method selection explained at the beginning of section 2.2. The options are Overwrite Only, Overwrite Only Fast, Swap, and Direct XIP, as shown in Figure 4. Overwrite Only is the default setting.

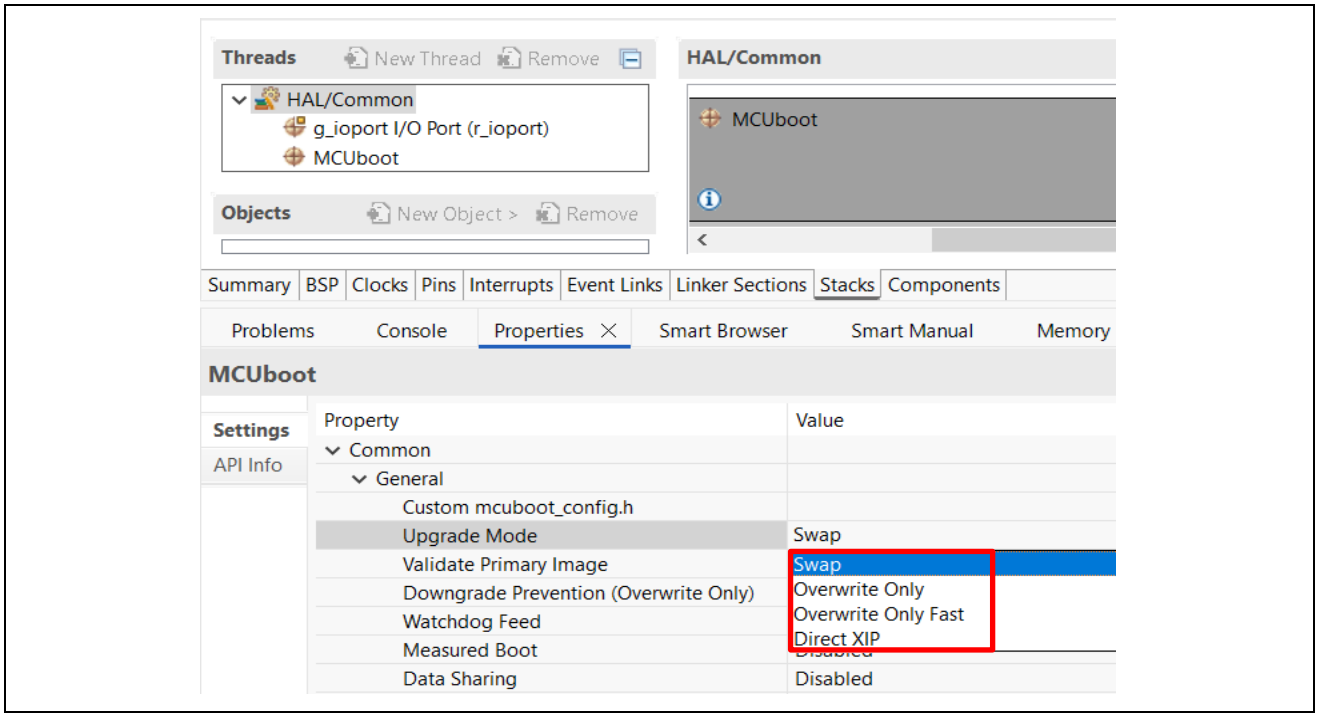


Figure 4. Application Image Update Mode

Figure 5 is a more detailed application image format that can be referenced to understand the various MCUboot property definitions.

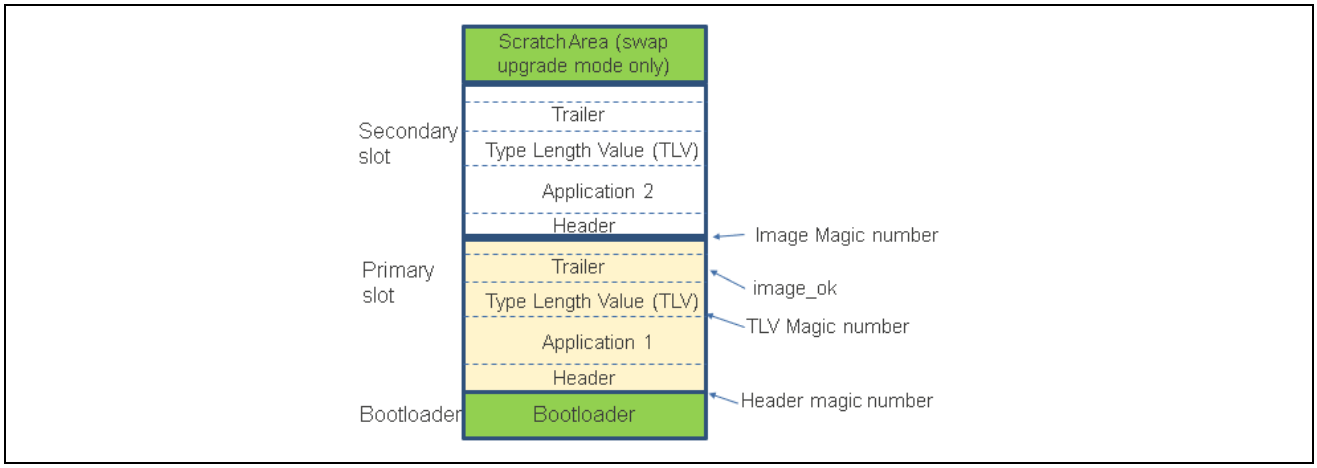


Figure 5. General Configuration for MCUboot Module

- **Validate Primary Image**:
  - When Validate Primary Image is enabled, the bootloader performs a hash or signature verification, depending on the verification method chosen, in addition to the MCUboot sanity check based on the image header and TLV area magic numbers. The Header and TLV area magic numbers are always checked as part of the sanity checking prior to the integrity checking and the signature verification.
  - When Validate Primary Image is disabled, the integrity check based on hash is performed as well as the sanity check is performed. It is highly recommended to always enable this property if boot time is not a concern. Note that the image magic number is not part of the image validation; it is a reference value that can be used for sanity check during application upgrade debugging process. This image magic number is written to the flash after a successful image upgrade.
- **Downgrade Prevention (Overwrite Only):** This property applies to Overwrite upgrade mode only. When this property is enabled, new firmware with a lower version number will not overwrite the existing application.
- **Watchdog Feed:** This function might be implemented if the OS / HW watchdog is enabled while doing a swap upgrade and the time it takes for a swapping is long enough to cause an unwanted reset. If implementing this, the OS main.c must also enable the watchdog.
- **Measured Boot:** This property applies to copying the boot data into the secure RAM, intended to be used by the secure application.
- **Data Sharing:** This property applies to copying the user data into the secure RAM, intended to be used by the secure application.

2.2.2 Application Image Signature Type Options

Application images using MCUboot must also be signed to work with MCUboot. At a minimum, this involves adding a hash and an MCUboot-specific constant value in the image trailer.

Figure 6 shows the signature types available for the application image signing methods supported by the MCUboot module. For memory restricted devices, you can choose **None** for **Signature Type**, which will reduce the bootloader size. For example, the bootloader for the Overwrite update mode uses a flash area of 64 KB when using ECDSA P-256 signature type, but when signature support is not used, the bootloader reduces to about 19 KB.

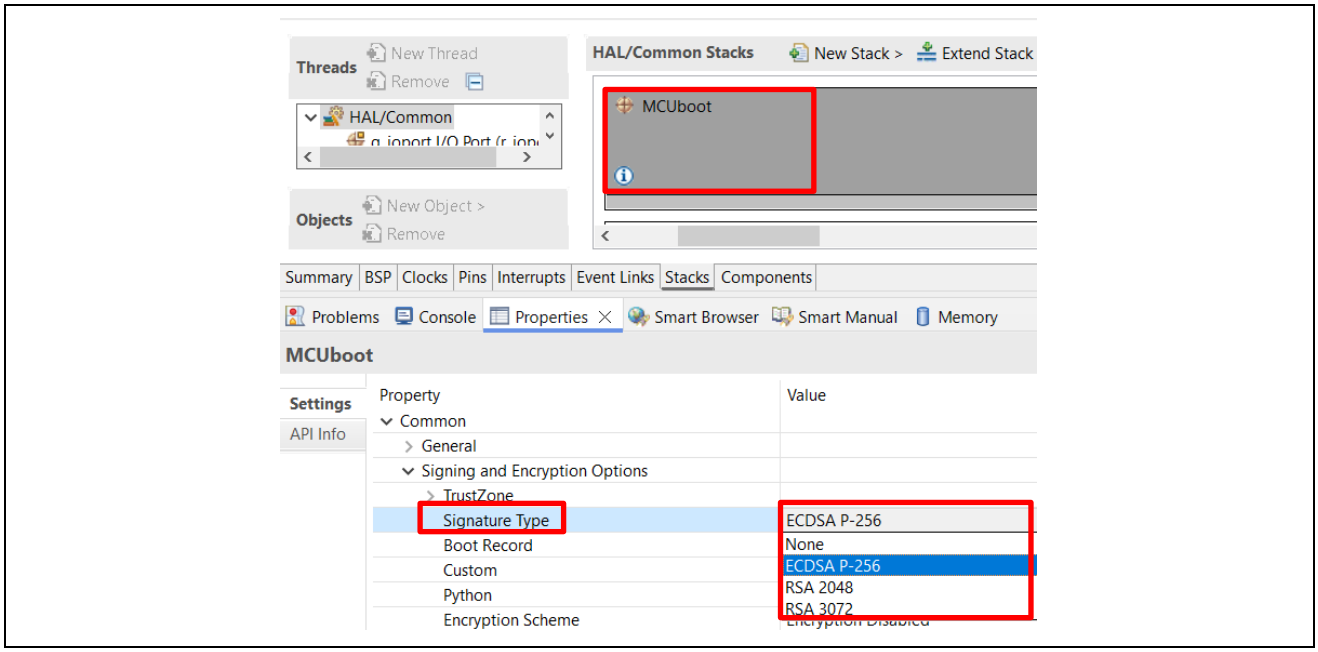


Figure 6. Application Image Signature Type for FSP MCUboot Module

2.2.3 Signing Options

Figure 7 shows the default **Custom** signing option configuration provided by FSP.

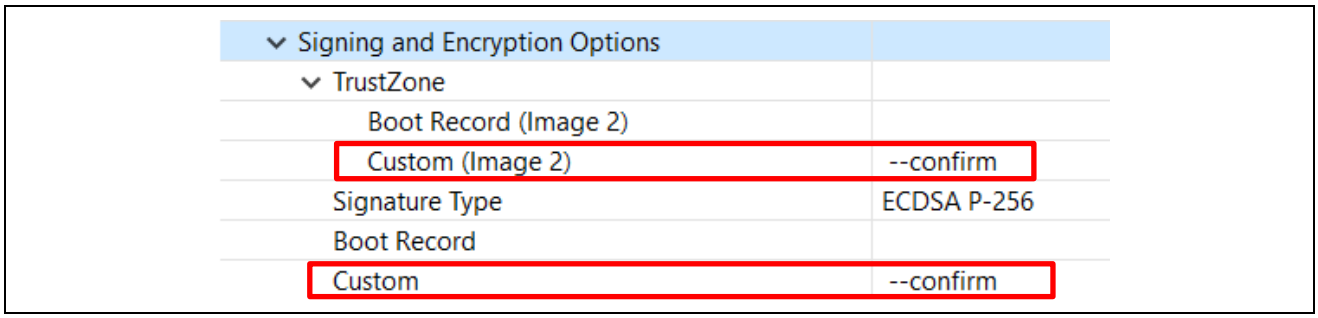


Figure 7. FSP Default Signing Option

By default, FSP sets **--confirm** for the **Custom** property for both Image 1 and Image 2 when TrustZone is used. For TrustZone-based applications, the Secure Image (Image 1) and Non-Secure Image (Image 2) can have different configurations such that there is different update policy for the Secure and Non-Secure Images. Some commonly used signing options are:

- Option **--pad**:
  This option places a trailer on the image that indicates that the image should be considered for an upgrade. Writing this image in the Secondary slot causes the bootloader to upgrade to it. When Swap mode is selected, this option generates a signing command such that the Secondary image will first be swapped with the Primary application image. On the next reset, the Primary application previously used will be swapped back and rebooted.
- Option **--confirm**:
  When Swap mode is selected, this option generates a signing command such that the Secondary image will first be swapped with the Primary application. At the next reset, there will be no swap between the Primary and Secondary application and the Secondary application will be booted. Confirm is the default Force Upgrade configuration.
- No input:
  If no option is put in this property, application images signed with the signing command generated from this setting will not be updated.

When Overwrite mode is selected, the **--pad** or **--confirm** option generates signing commands such that the overwrite will occur and the Secondary application will overwrite the Primary application.

The image signing tool `Imgtool.py` is included with MCUboot. It is integrated as a post-build tool in e² studio to sign the application image. For detailed information about using this tool with e² studio, refer to the application image signing information in section 6.2. For more information on the possible options available for this property setting, refer to the description in the imagetool.py md file and visit the MCUboot documentation page https://docs.mcuboot.com/imgtool.html.

3. Keys Preparation

3.1 Using the Renesas Key Wrap Service to Create PGP Key Pair and Exchange Keys with Renesas DLM Server

The Renesas Key Wrap Service must be used to wrap an arbitrary User Factory Programming Key (UFPK) specific to the target MCU Group and security engine operation mode. All key material exchange is performed with PGP encryption. The steps to establish this PGP-encrypted communication channel between the user and the Renesas Key Wrap Server can be referred to in section 3 "*Using the Renesas Key Wrap Service"* on application project r11an0496. This is a one-time process and does not need to be repeated for different MCUs. The output of this process is the Renesas public key, which will be used in the following step.

3.2 Renesas Security Key Management Tool

The Renesas Security Key Management Tool (SKMT) performs several functions during the secure key injection process. Open the following link to access the latest SKMT:

https://www.renesas.com/software-tool/security-key-management-tool

From the above link, find the **Downloads** area and download the latest Security Key Management Tool installer. This tool supports Windows, Linux, and macOS. The screenshots in this document came from the Windows environment.

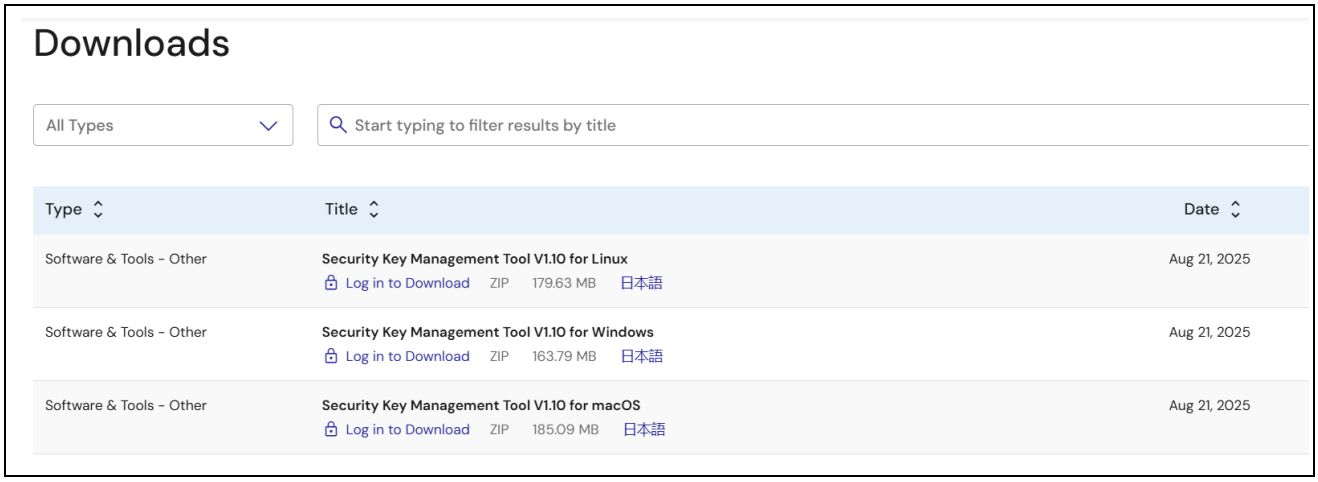


Figure 8. Download the Security Key Management Tool for Windows, Linux or macOS

Once the installer executable is downloaded, right-click on the installer and select **Run as administrator** to install this tool. Follow the prompt to select the **Setup Language**. Currently, both English and Japanese are supported. Next, select the installation folder. By default, it will be installed into `C:\Renesas\SecurityKeyManagementTool\`. If a previous version is installed, the old version will be overwritten.

The User's Manual of this tool is located in the `\DOC` folder. We recommend that you read through the user's manual before proceeding to the following section.

The SKMT provides two interfaces for users: a Command Line Interface (CLI) and a Graphic User Interface (GUI). The CLI interface is typically used for production support, and the GUI interface is primarily intended for development usage. This application note will explain how to use both interfaces to perform key injection.

3.3 Wrap Key with UFPK and W-UFPK using SKMT GUI Interface

3.3.1 Creating and Wrapping UFPK

Define a UFPK and convert it to a binary format that is compatible with the Renesas Key Wrap Service. This can be done using the Renesas Security Key Management Tool (SKMT).

The same UFPK can be used for all RA Family MCUs. However, the corresponding W-UFPK may be different as it depends on the specific MCU Group. To avoid confusion and mistakes, it is recommended to choose the correct RA MCU Family when generating UFPK using the SKMT GUI interface and name them different based on the MCU family.

Double-click **SecurityKeyManagementTool.exe** to launch the GUI interface.

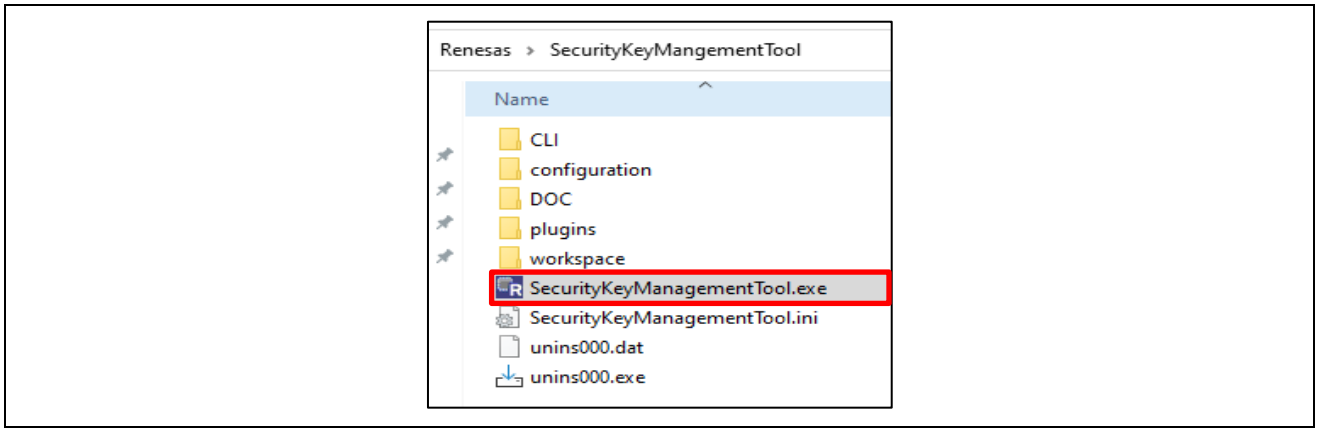


Figure 9. Launch SKMT GUI Interface

To use the example projects included in this application project, set the UFPK to
`000102030405060708090A0B0C0D0E0F000102030405060708090a0b0c0d0e0f`

Note that the 32-byte UFPK must be provided in big-endian format.
It is important to select the correct MCU family and security engine mode when using the SKMT tool. In the **Overview** window, select **RA Family, SCE9 Security Functions, and Protected Mode**.

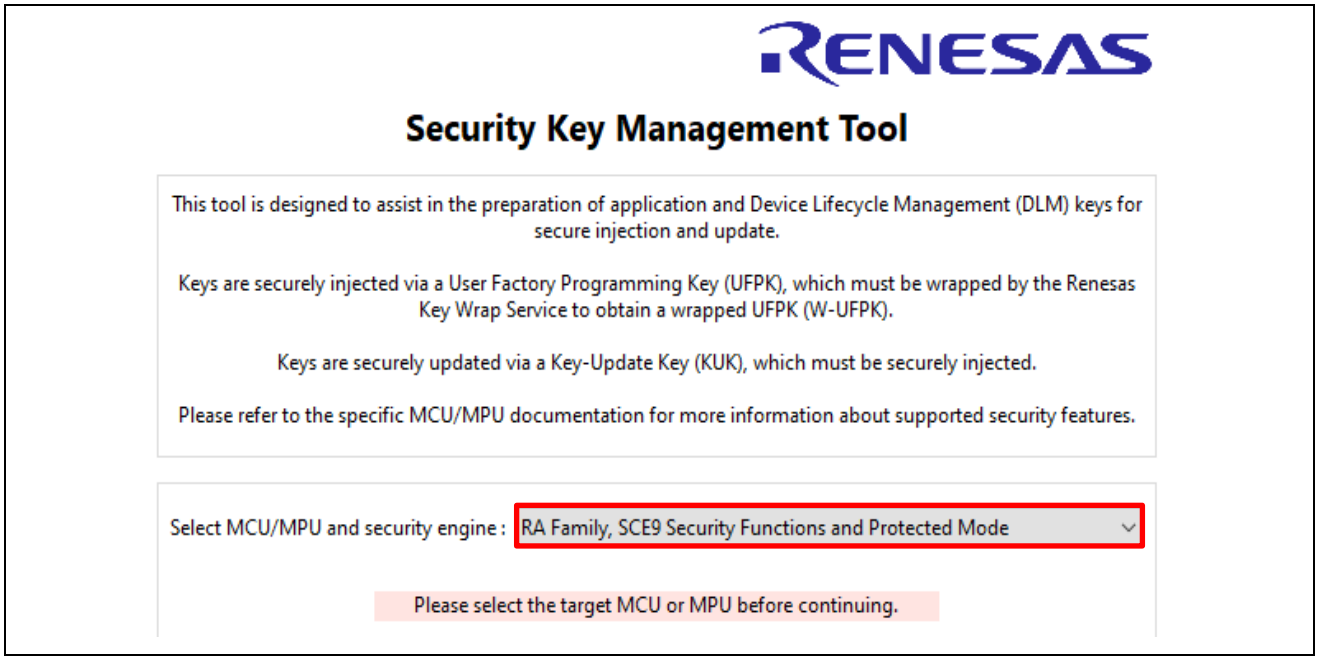


Figure 10. Select RA Family, SCE9 Protected Mode

Once the correct MCU Family, Security Engine, and Mode are selected, navigate to the **Generate UFPK** page.

- For the **User Factory Programming Key**, select **Use specified value**.
- Click the **Browse** button to select a folder to store the key and name the resulting file.
- Click **Generate UFPK key file**. The `ra6m4_ufpk.key` file will be generated.

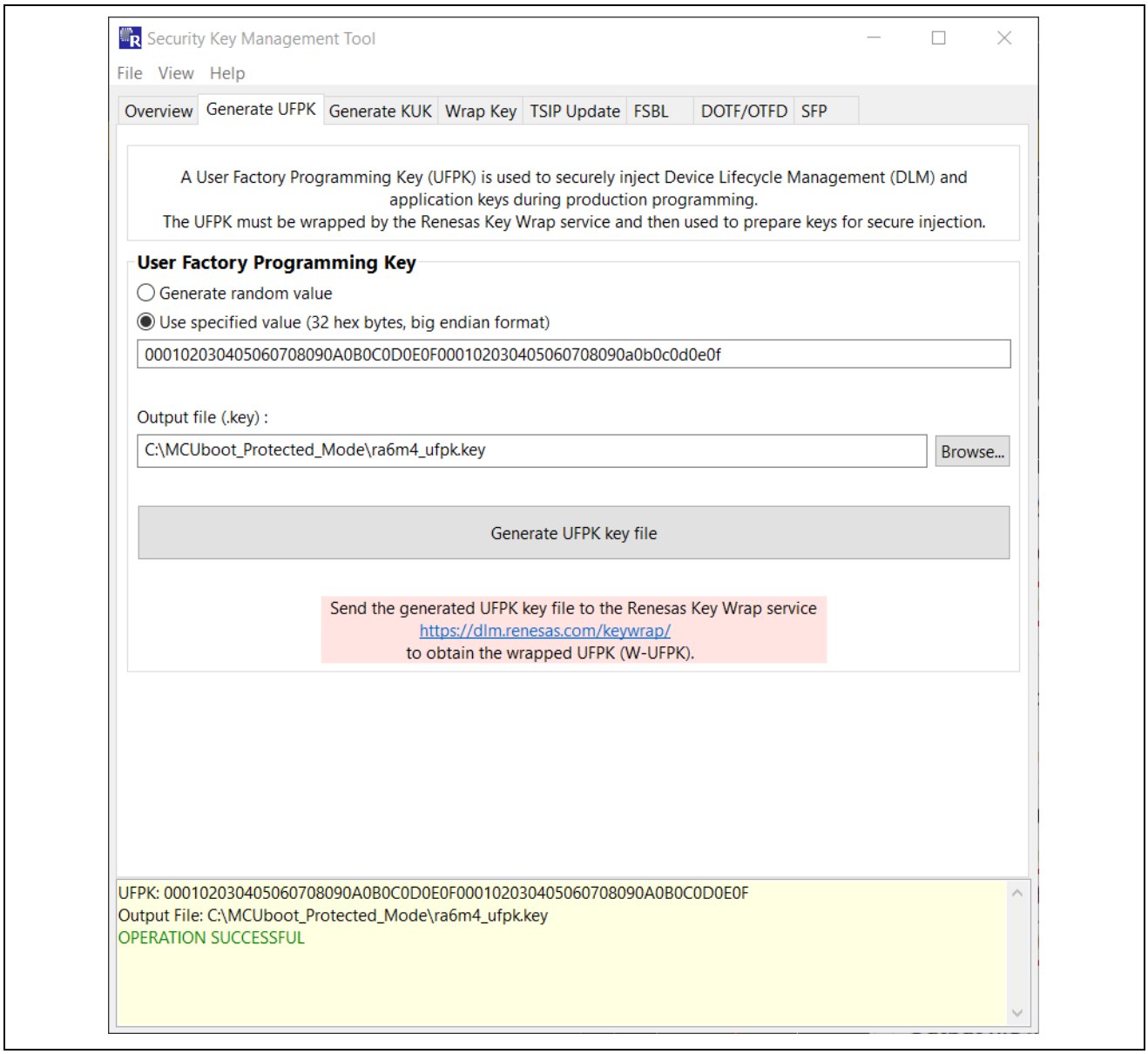


Figure 11. Generate Fixed UFPK using GUI for SCE9

Optionally, the user can also choose the **Generate random value** option to generate the UFPK

The next step is to obtain a W-UFPK from the Renesas Key Wrap Service based on the selected UFPK. Note that if the UFPK is changed, a new W-UFPK must be obtained.

1. Launch the Kleopatra program.
2. Encrypt the UFPK with the Renesas public key. This key was imported earlier to Kleopatra. Using Kleopatra, select **Sign/Encrypt…** and select the UFPK file. In this screenshot, an example file named `ra6m4_ufpk.key` file is used for demonstration purposes. Then click **Open**.

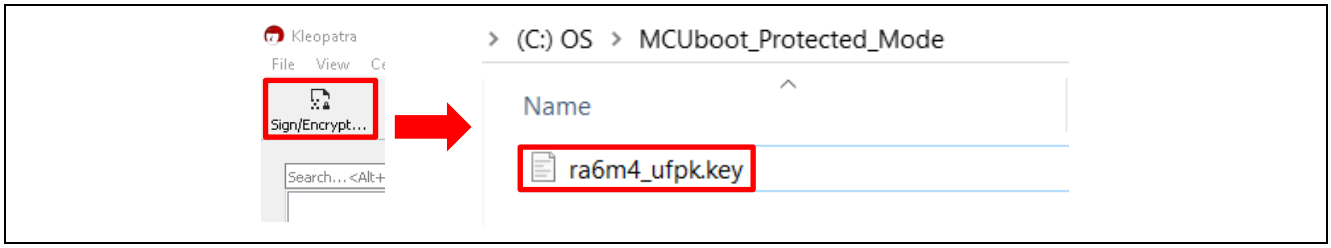


Figure 12. Encrypt the UFPK File for PGP Transfer

3. When asked which entity this file is to be encrypted for, (optionally) uncheck **Encrypt for me** and check **Sign as, Encrypt for others**, and **Encrypt / Sign each file separately**.

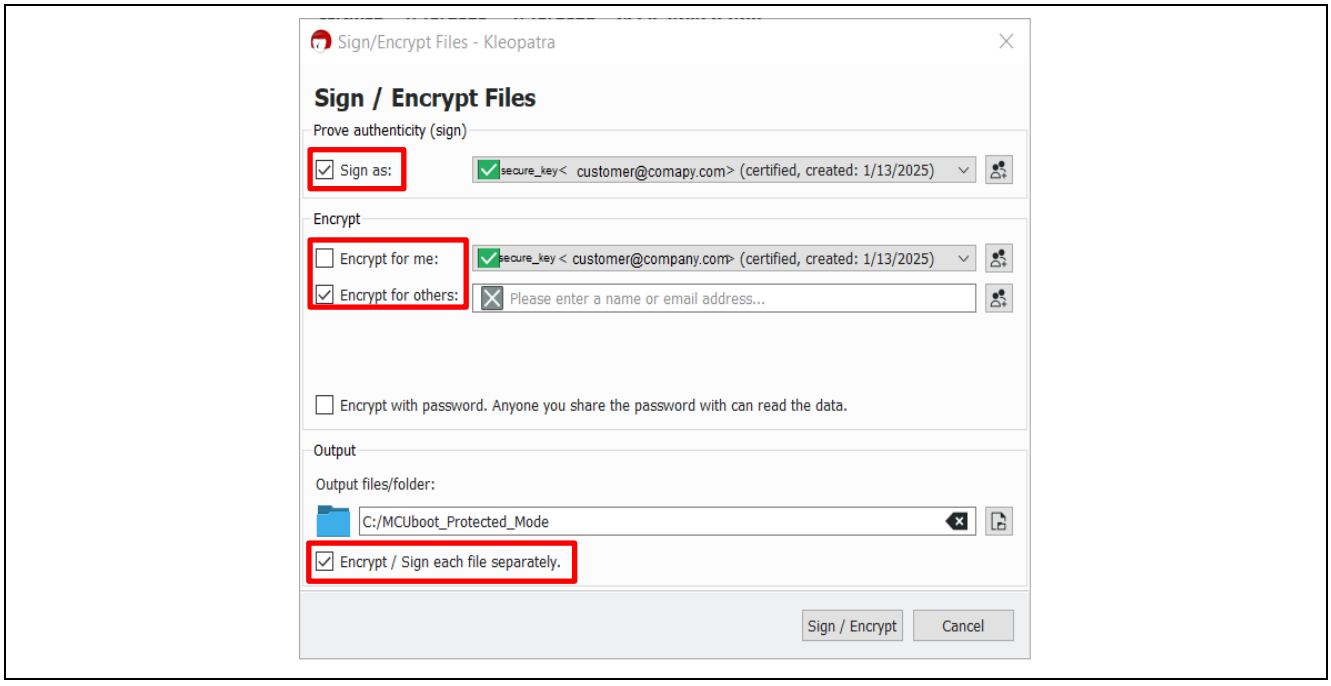


Figure 13. Select PGP Encryption Options

4. Click the **Open Selection Dialog** (the icon). This will open a **Certificate Selection** dialog box.

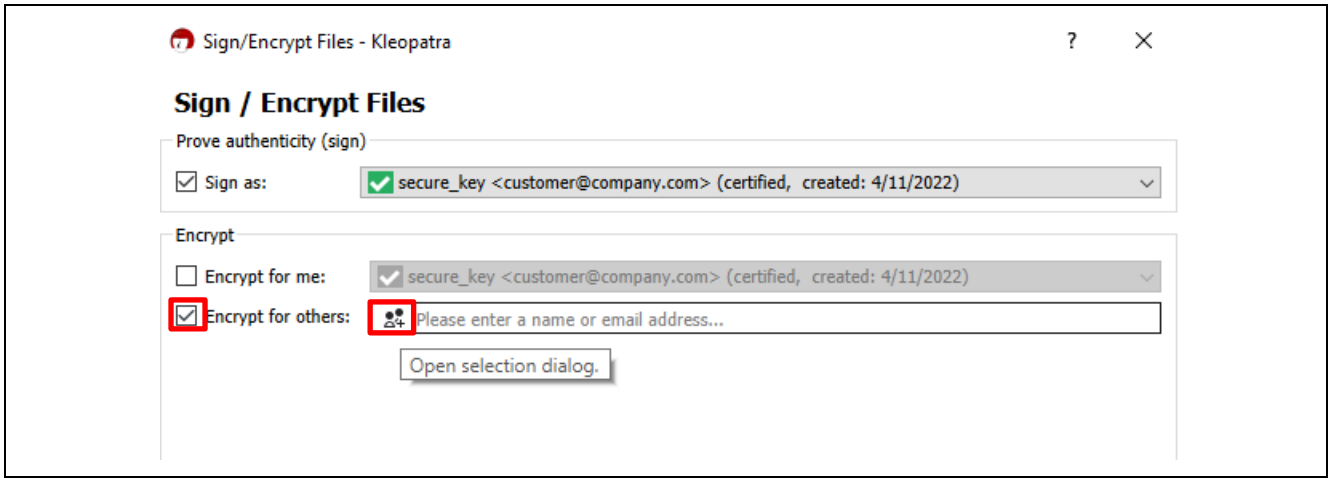


Figure 14. Open the Selection Dialog

5. In this window, select **keywrap** to select the Renesas public key, then click **OK**

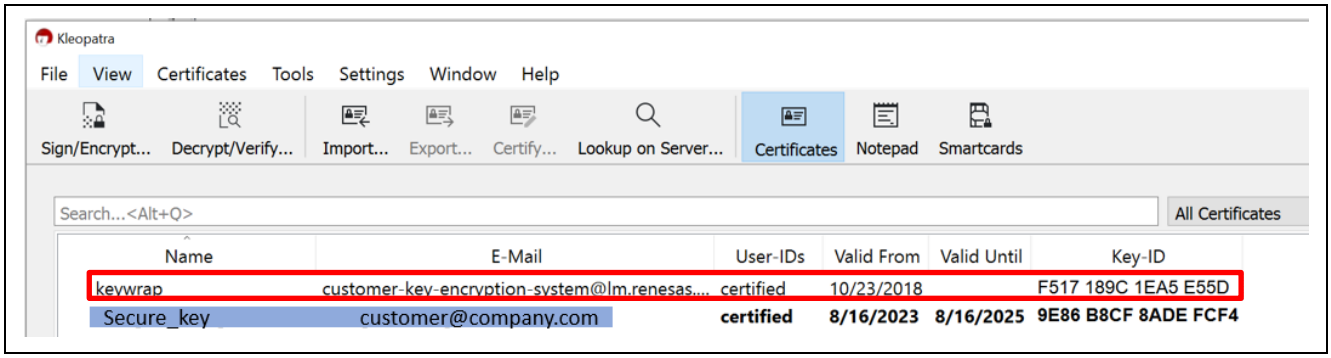


Figure 15. Select the Renesas PGP Public key

6. Ensure that the correct destination folder for the encrypted key is selected under **Output.** Finally, click **Sign/Encrypt**. It is a good practice to keep UFPK and W-WUPK for different MCU families in different folders and under different names.

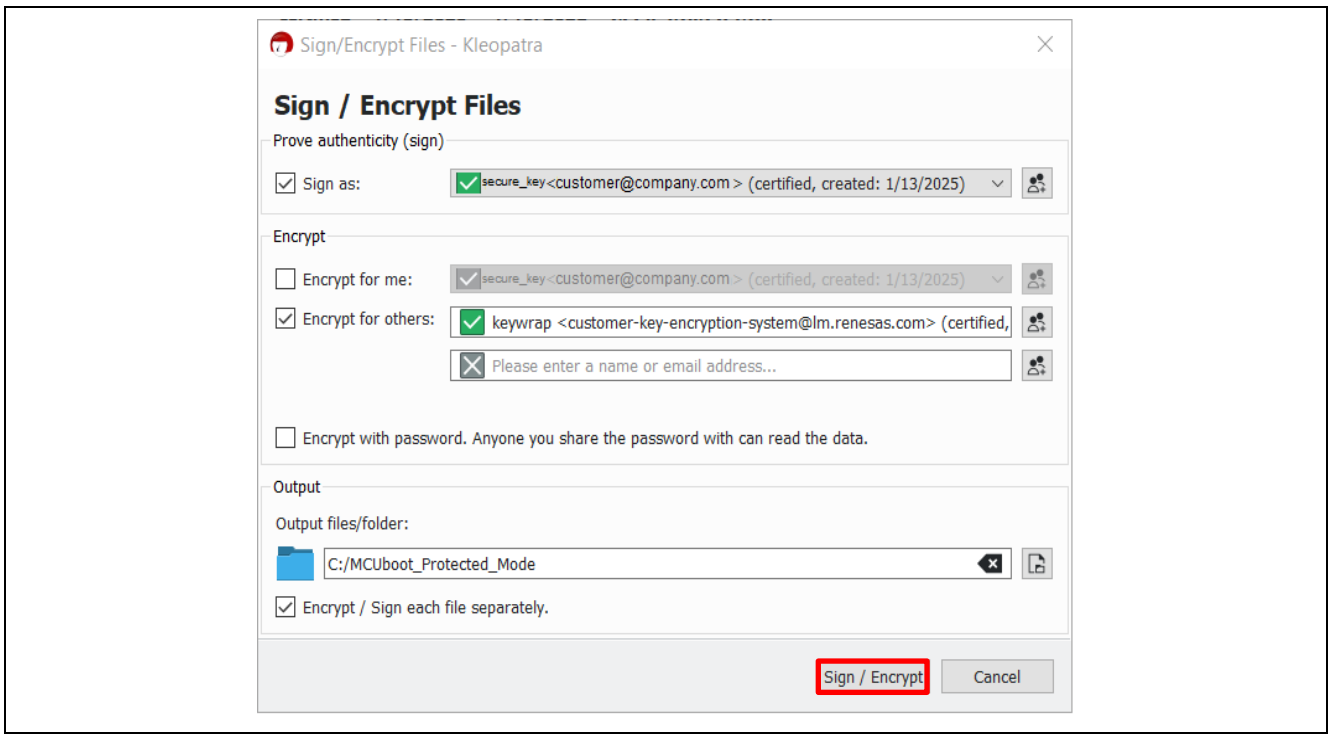


Figure 16. Encrypt UFPK using Renesas PGP Public Key

7. If you do not check **Encrypt for me**, you will get an **Encrypt-To-Self Warning** that you cannot decrypt the data. Click **Continue**.

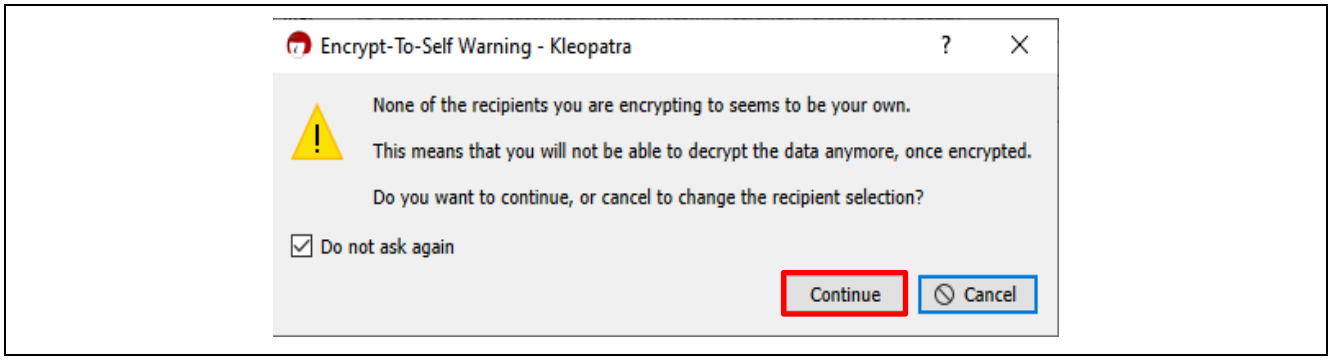


Figure 17. Start the UFPK Encryption process

8. Provide your private key passphrase, then click **OK**.

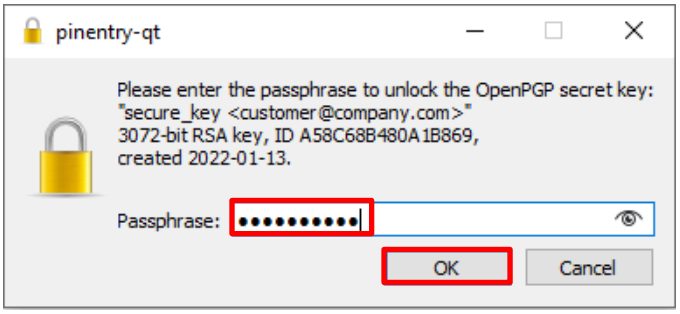


Figure 18. Provide Passphrase

9. The UFPK encrypted with the Renesas public key will be generated, with the `.gpg` added to the extension of the key. In this case, the file `ufpk_ra6m3.key.gpg` is generated. Click **Finish**.

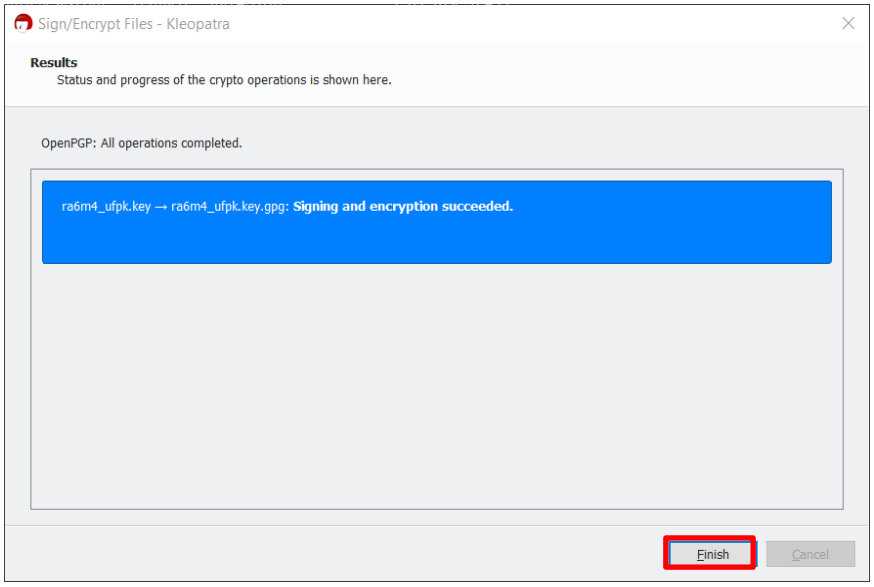


Figure 19. Encrypted Key is Generated

10. Now, we can send the UFPK that has been encrypted with Renesas Public Key to the Renesas DLM Server for wrapping. Return to the DLM Server web page:

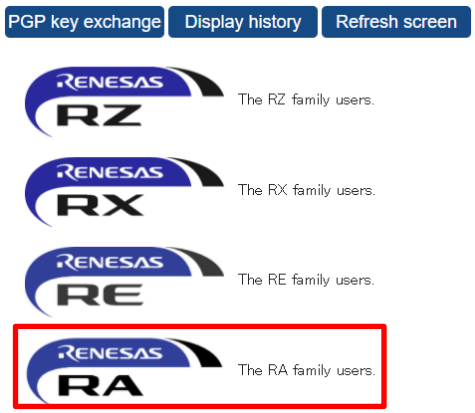


Figure 20. Select the MCU Family

When generating the Wrapped UFPK, it is important to select the correct MCU family and security engine mode.

Note: A W-UFPK generated for Compatibility Mode cannot be used to inject keys in DLM or Protected Mode, and vice versa.

To create a W-UFPK for the RA6M4 example project, select the **Renesas RA** Family and click Protected Mode **RA6M4/RA6M5 Encryption of customer's data**.

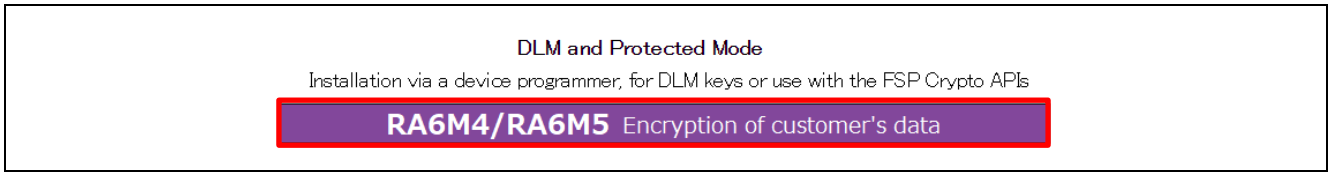


Figure 21. Select the RA6M4/RA6M5 MCU Group DLM and Protected Mode

11. Click **Encryption service for products** on the next screen. Here, the screenshot uses RA6M3 as an example; for other MCU families, a similar screen will be presented.

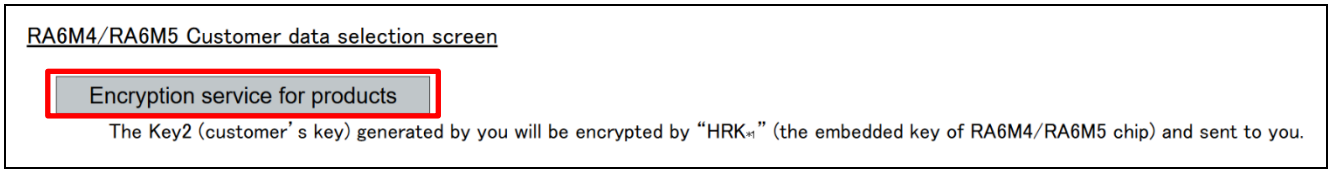


Figure 22. Choose Encryption service for products

12. Click **Reference** and select the corresponding encrypted UFPK; example shown is `ra6m4_ufpk.key.gpg` created previously and click **Open**. Note that in the DLM server description, **Key2** refers to the UFPK.

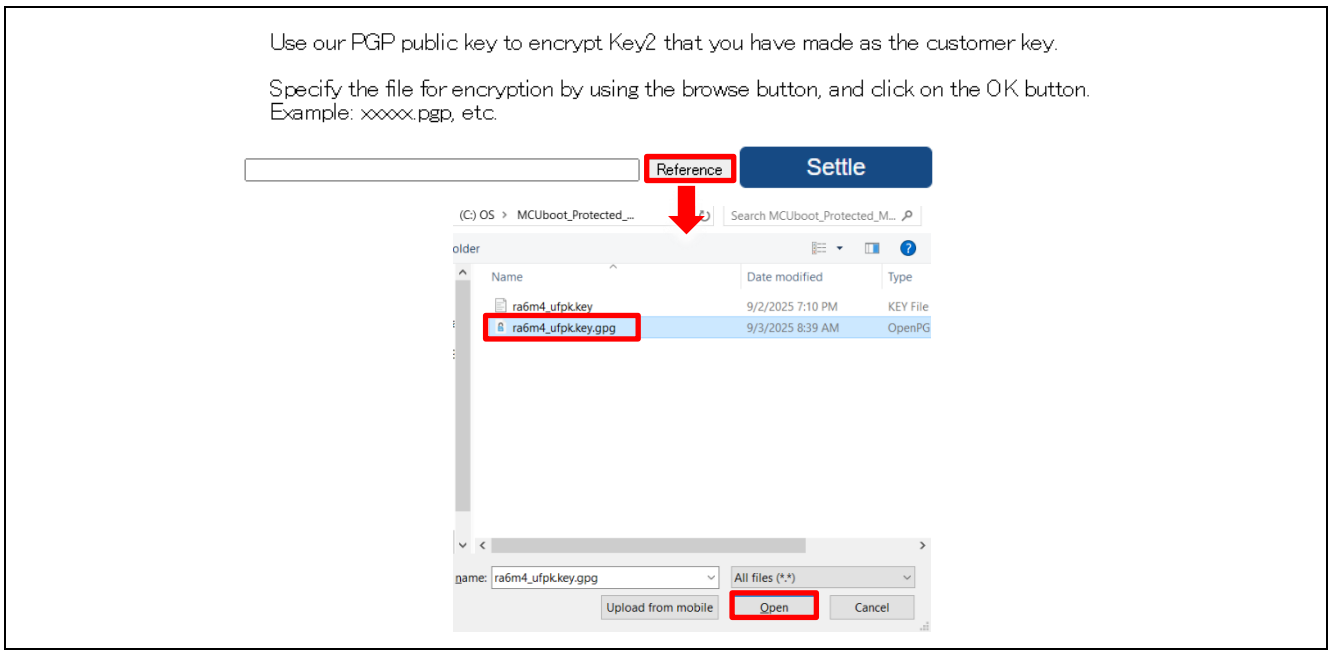


Figure 23. Select the PGP-Encrypted UFPK filek

13. Click **Settle**. The following message will be printed. Then click **Return to the menu**. You can now log out of the Renesas Key Wrap Service.

Figure 24. Return to the DLM Server Main Menu

14. The wrapped UFPK Key (W-UFPK) encrypted with your PGP public key should arrive in your email typically in about 1-2 minutes. Save the attached file.

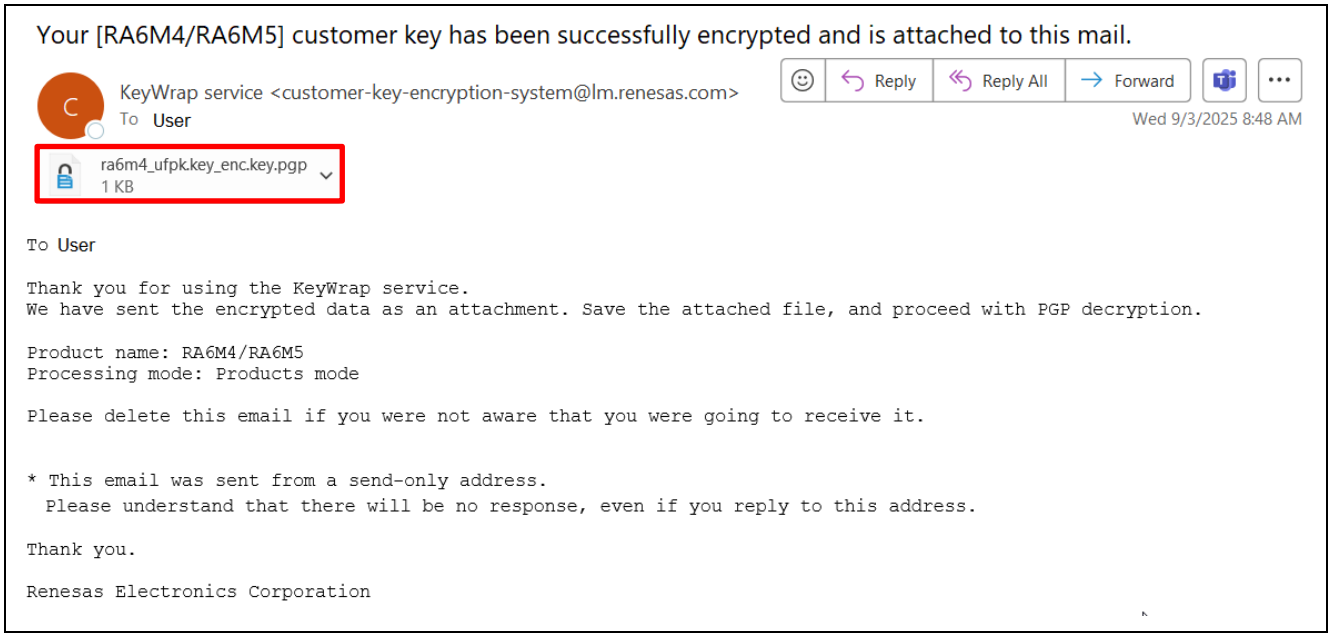


Figure 25. Receiving the W-UFPK via Email

15. With the Kleopatra program, click **Decrypt/Verify,** select the encrypted W-UFPK file, and click **Open**.

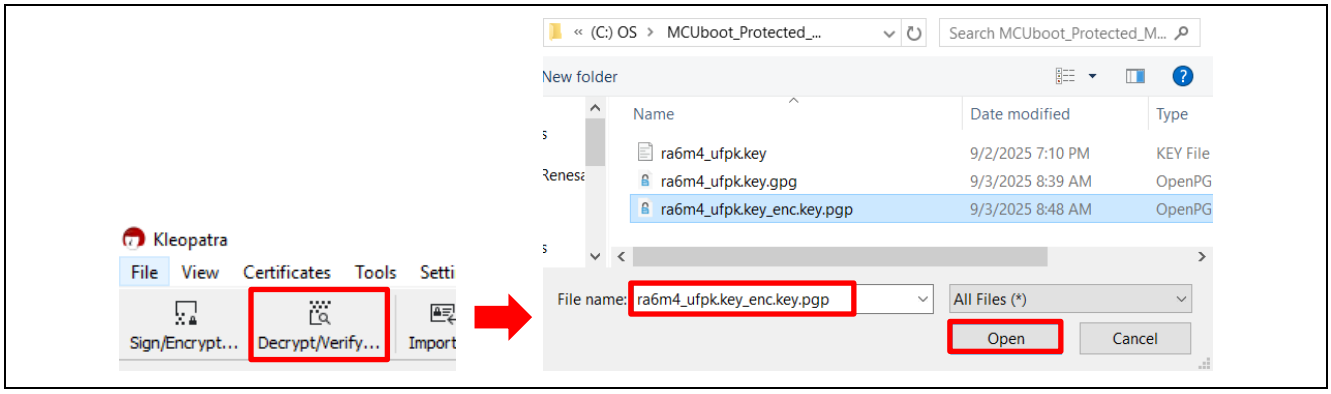


Figure 26. Decrypt the W-UFPK

16. Follow the prompt to provide your PGP private key passphrase and click **OK.** The decrypted W-UFPK is generated in the folder specified.

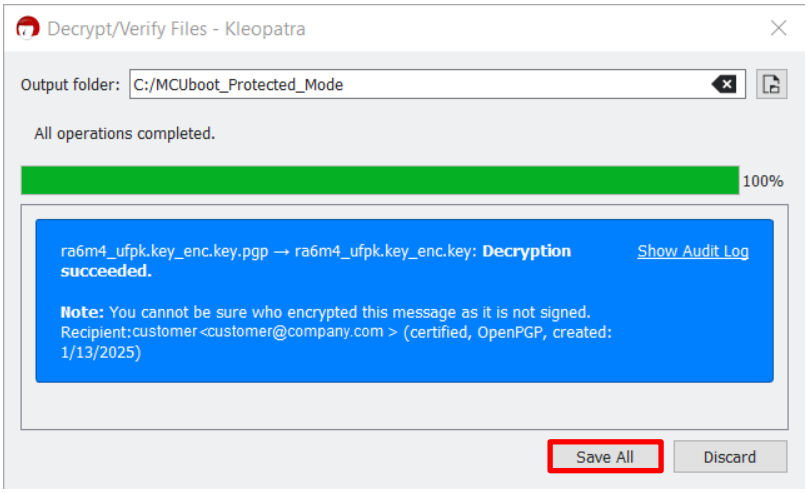


Figure 27. Decrypting the Encrypted W-UFPK

Click **Save All** to save the decrypted W-UFPK key file `ra6m4_ufpk.key_enc.key` to the same folder as the UFPK key file. Both key files are required to generate key injection bundles.

3.3.2 Wrapping User Key with UFPK and W-UFPK

From the Windows OS Start menu, open the Win64 OpenSSL Command Prompt

Figure 28. Open the Win64 OpenSSL Command Prompt

Next, user needs to extract the public key data from the .pem file used for signing the image. The test key file used depends on the **Signature Type** configured for the MCUboot project. In this example, the signature type is **ECDSA P-256**, and the project uses the default public/private key pair included in MCUboot for testing purposes. For more information, please refer to Section 5.3.

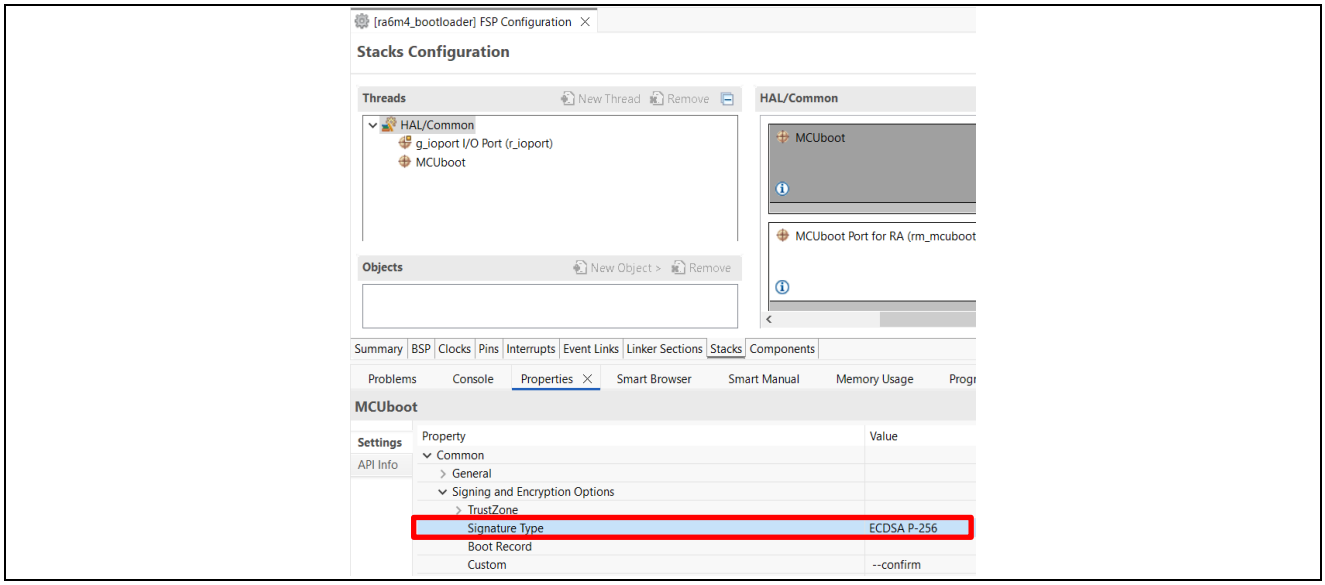


Figure 29. Signature Type Configuration

To extract the public key from the .pem file used for signing the image, execute the following command:

```
$  openssl ec -noout -text -in <path_to_pem_file_in_bootloader>
```

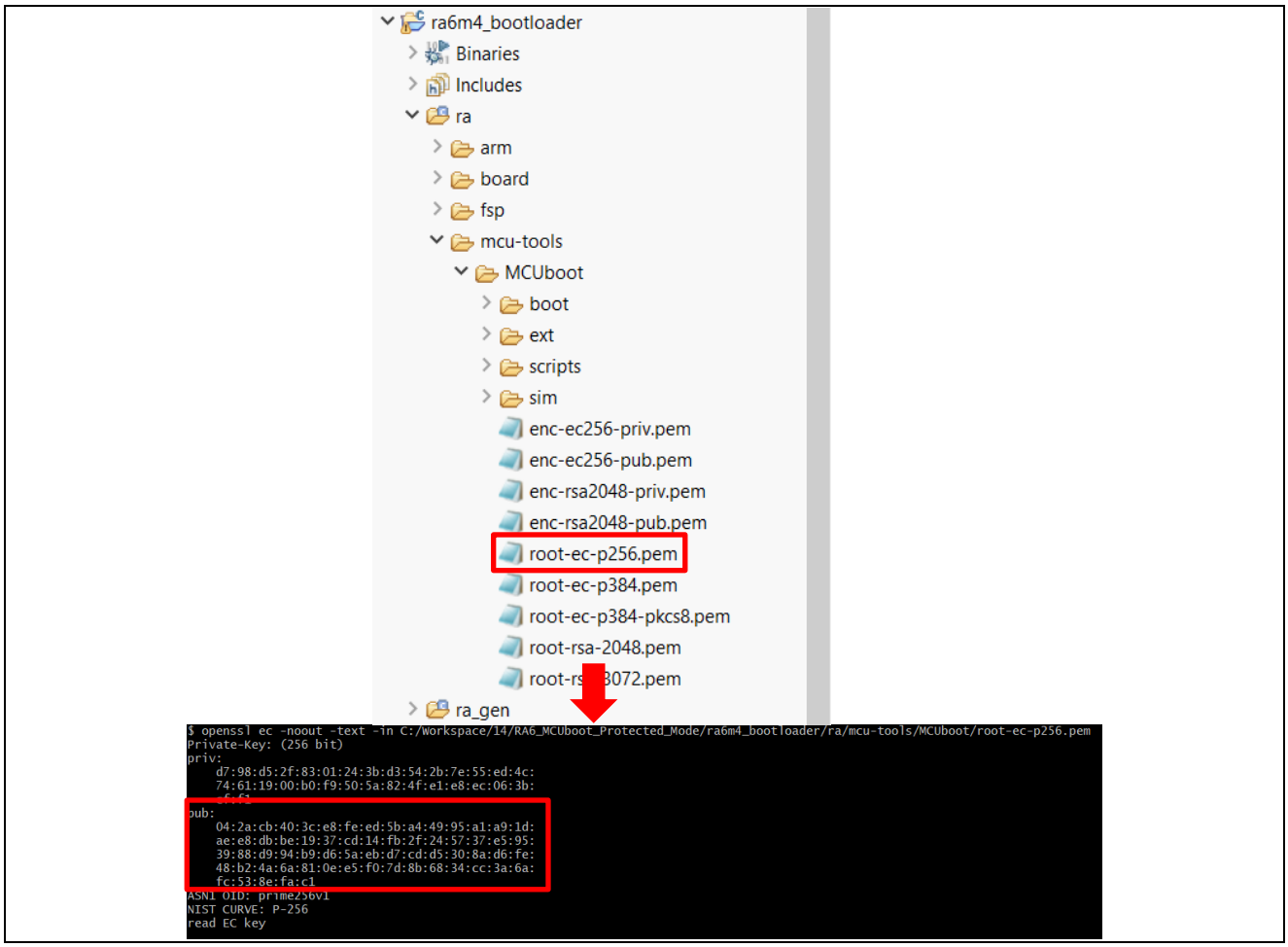


Figure 30. Print out public key data

This 64-byte public key data, which is a combination of **Qx** and **Qy**, will be used in the next steps of the user key wrapping process.

Note that when the ECC public key is printed out this way, it will contain a 0x04 ASN.1 prefix at the start, which should be discarded.

Launch the SKMT GUI and select **RA Family, SCE9 Security Functions, and Protected Mode** on the **Overview** tab. On the **Wrap Key** tab, select the **Key Type** as **ECC** and **secp256r1, public,** as shown in Figure 31.

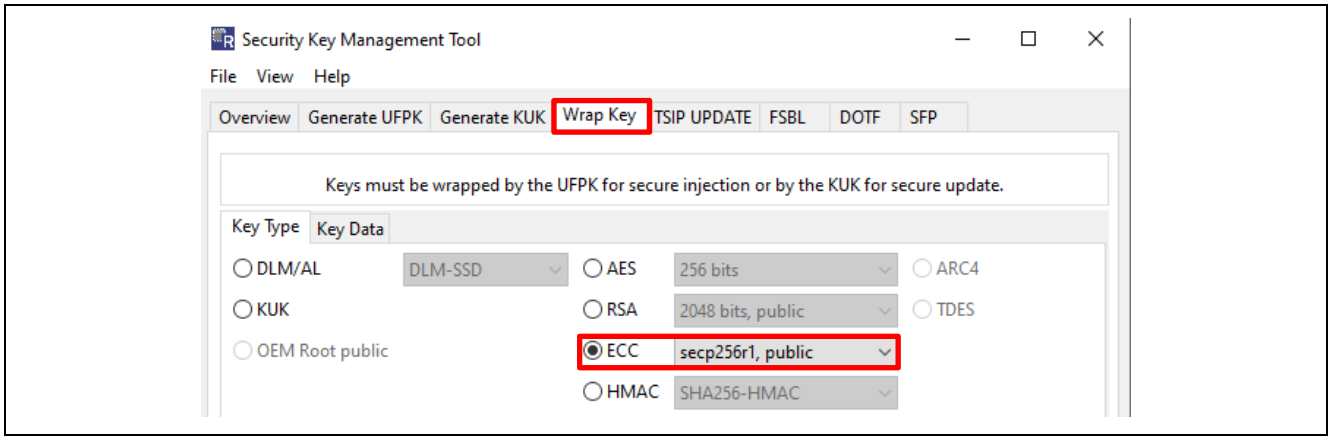


Figure 31. Choose secp256r1 Public Key

Next, configure the **Key Data**. Under the **Key Data** area, select **Raw** and provide the **Qx** and **Qy** following data in Figure as shown below. The key data is duplicated here to easily copy and paste to the GUI interface.

Qx = 2acb403ce8feed5ba44995a1a91daee8dbbe1937cd14fb2f245737e5953988d9

Qy = 94b9d65aebd7cdd5308ad6fe48b24a6a810ee5f07d8b6834cc3a6afc538efac1

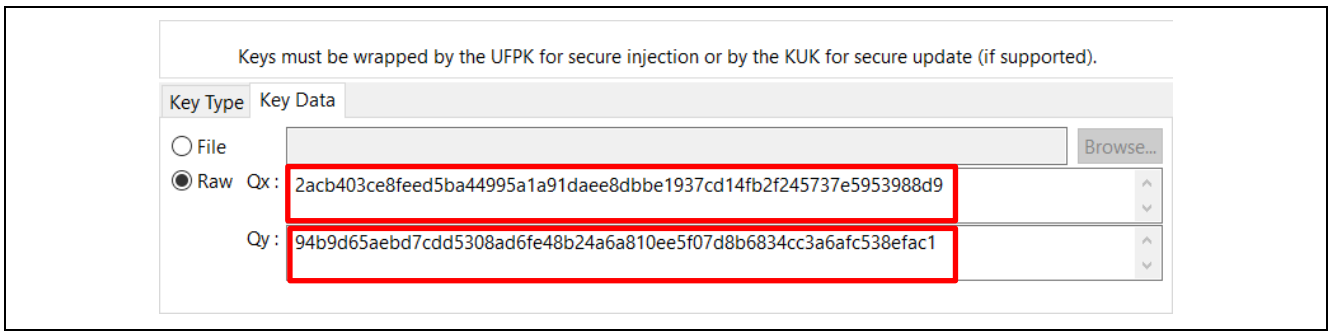


Figure 32. Provide the ECC Public Key data

Next, under the **Wrapping Key** section, click the corresponding **Browse** buttons to select the **UFPK** and **W-UFPK** key pair created in section 3.3.1. For the **IV**, select **Generate random value**. For the **Output** option, select **RFP**; then click the **Browse** button, choose the output folder, and name the output file.

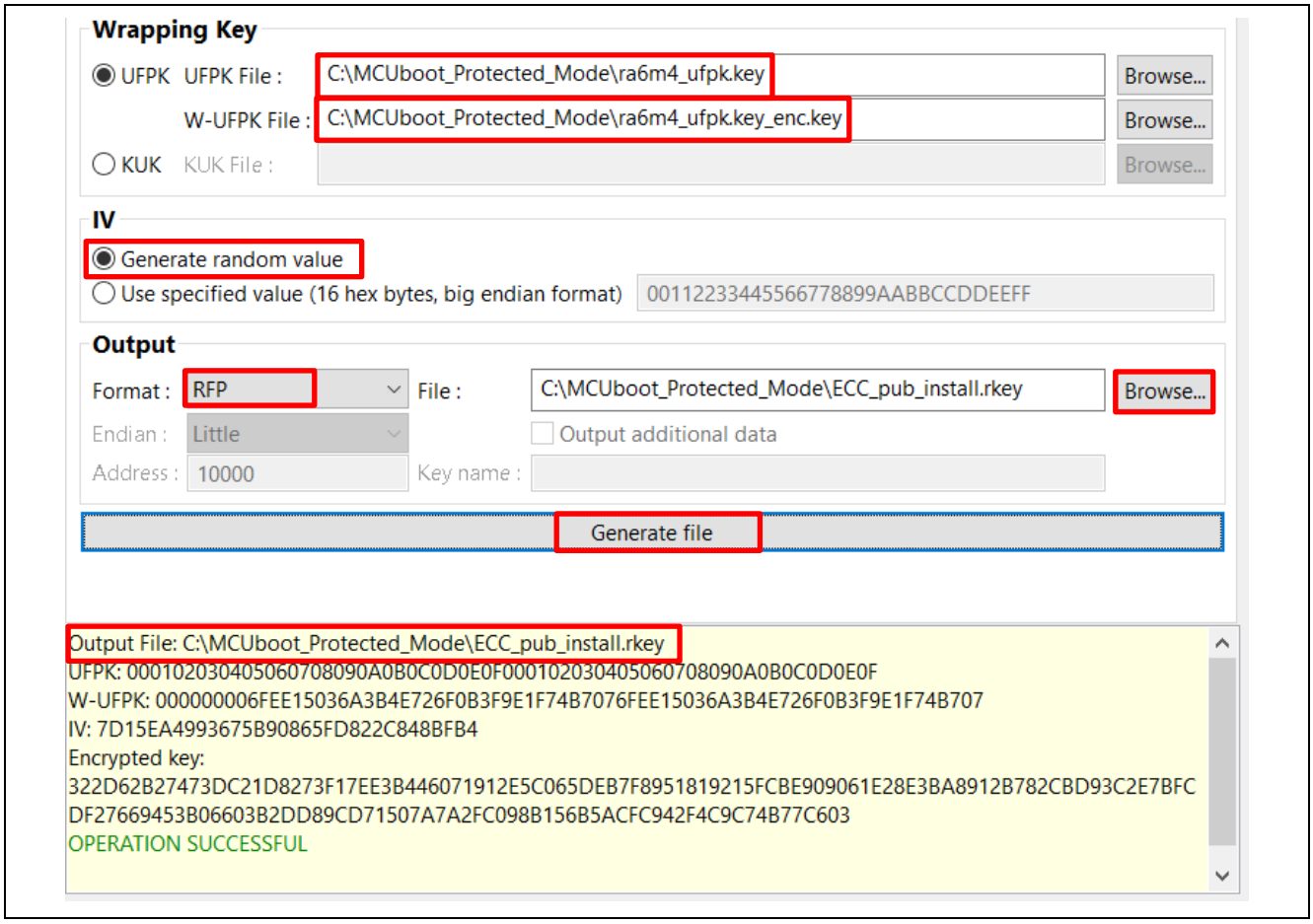


Figure 33. Generate the ECC Public Key RFP Injection Key File using GUI

The plaintext ECC key and UFPK are NOT contained in the `*.rkey` file, enabling confidential transfer of the key injection file contents.

3.4 Wrap Key with UFPK and W-UFPK using SKMT CLI Interface

This section describes how to perform the actions described above using the SKMT CLI interface. These examples use SCE9 Protected mode for EK-RA6M4. Refer to the Security Key Management Tool user's manual for more information about commands used for creating and wrapping key in this section, including the valid values for each parameter.

3.4.1 Creating and Wrapping the UFPK

Open a Command Prompt window and navigate to the folder where `skmt.exe` resides, typically under `\Renesas\Security Key Management Tool\CLI\`.

Use the following command to generate a random UFPK and place it in a key file (`ufpk.key`). If desired, a complete file name with a path may be specified. Refer to the Security Key Management Tool user's manual to understand the usage of `/genufpk` option.

```
skmt.exe /genufpk /output "C:\MCUboot_Protected_Mode\ufpk.key"
```

This command will generate a random 256-bit UFPK as shown below.

```
UFPK: E8AB23E99C9AD42823DA4215549A41496720F7243680A4715F4B944ACC94B691
Output File: C:\MCUboot_Protected_Mode\ufpk.key
```

Figure 34. Create a Random UFPK Using SKMT CLI

It is also possible to specify a specific UFPK, as shown by the following command:

```
skmt.exe /genufpk /ufpk
"000102030405060708090A0B0C0D0E0F000102030405060708090a0b0c0d0e0f" /output
"C:\MCUboot_Protected_Mode\ufpk.key"
```

```
UFPK: 000102030405060708090A0A0C0D0E0F000102030405060708090a0b0c0d0e0f
Output File: C:\MCUboot_Protected_Mode\ufpk.key
```

Figure 35. Create a Fixed UFPK Using SKMT CLI

3.4.2 Wrapping User Key with UFPK and W-UFPK

In this section, we will use the ECC public key data in Figure 30 as an example of preparing an ECC public key for secure key injection.

In the Command Prompt window opened earlier (section 3.4.1), use the following command to create the ECC public key injection file (`ECC_Public_Key_CLI.rkey`). Refer to the Security Key Management Tool user manual for more information on how to construct the command.

```
Skmt.exe /genkey /ufpk file="C:\MCUboot_Protected_Mode\ufpk.key" /wufpk
file="C:\MCUboot_Protected_Mode\ufpk.key_enc.key" /mcu "RA-SCE9" /keytype
"secp256r1-public" /key
"2acb403ce8feed5ba44995a1a91daee8dbbe1937cd14fb2f245737e5953988d994b9d65aebd7c
dd5308ad6fe48b24a6a810ee5f07d8b6834cc3a6afc538efac1" /filetype "rfp" /output
"C:\MCUboot_Protected_Mode\ECC_Public_Key_CLI.rkey"
```

Note that in this example:

- `2acb403ce8feed5ba44995a1a91daee8dbbe1937cd14fb2f245737e5953988d994b9d65aebd7c dd5308ad6fe48b24a6a810ee5f07d8b6834cc3a6afc538efac1` is the public key data from Figure 30.
- We have specified the key type "secp256r1-public".
- "`RA-SCE9`" is used for the `/mcu` option.
- We are using a randomly generated IV. The IV is updated in each encryption instance.
- The command option `/output` defines the locations and name of the output file.

```
Output File: C:\MCUboot_Protected_Mode\ECC_Public_Key_CLI.rkey
UFPK: 000102030405060708090A0B0C0D0E0F000102030405060708090A0B0C0D0E0F
W-UFPK: 1234567890ABCDEF1234567890ABCDEF1234567890ABCDEF1234567890ABCDEFE12345678
IV: 0273B7277508F33491F2BA569B092535
Encrypted key:
1234567890ABCDEF1234567890ABCDEF1234567890ABCDEF1234567890ABCDEF1234567890ABCDEF1234
567890ABCDEF1234567890ABCDEF1234567890ABCDEF1234567890ABCDEF
```

Figure 36. Create the ECC Public Key Injection File Using CLI

4. Running the Example Projects

The example projects demonstrate the MCUboot Protected Module using the Swap Test Update mode on the EK-RA6M4 board. The figure below illustrates the major events involved in the embedded system design using MCUboot as the secure bootloader in these example projects.

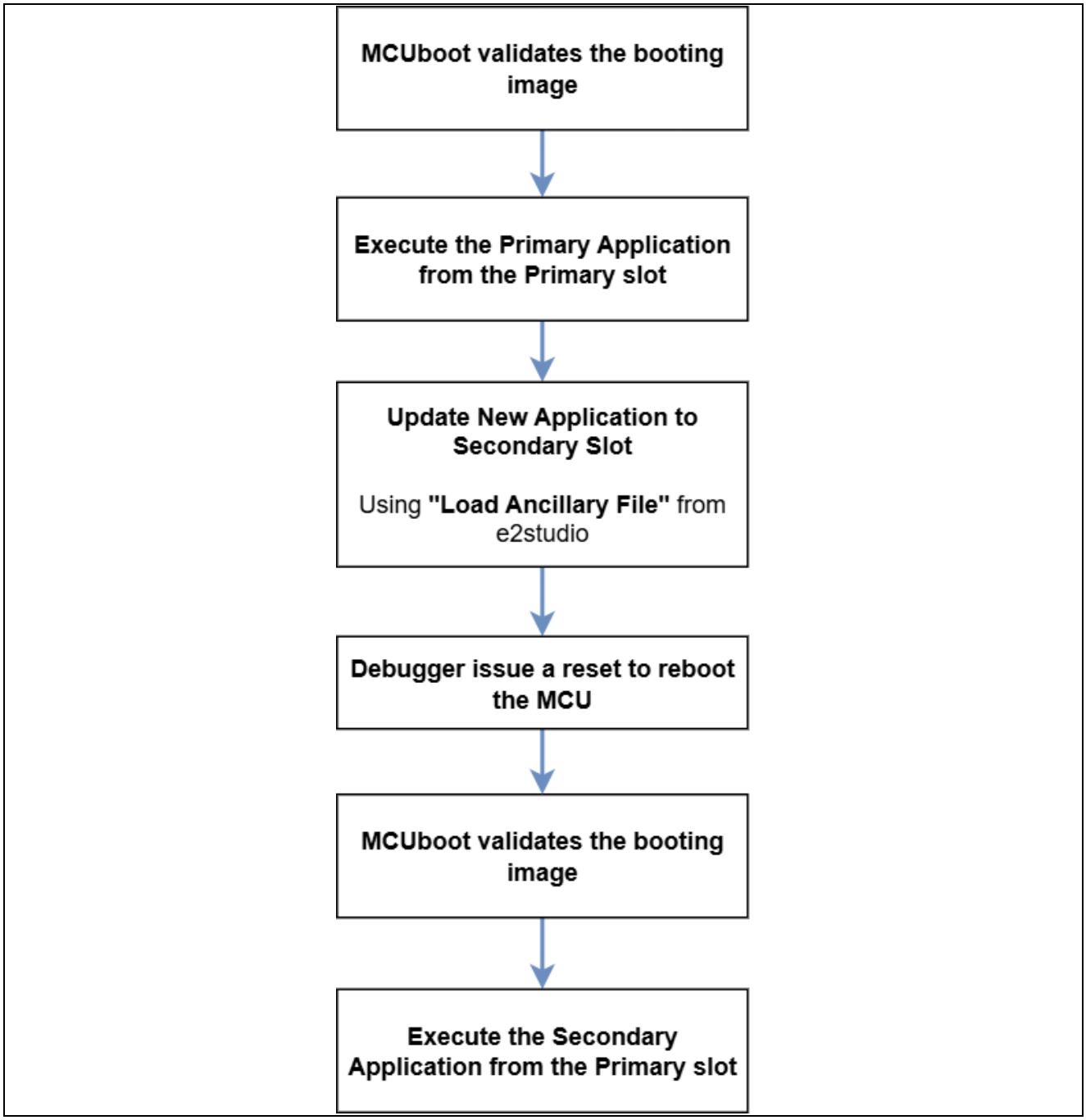


Figure 37. System Overview

Sections 4.1 to 4.2 provide guidance how to set up the environment, prepare the required hardware, and run the example project.

4.1 Configure the Python Signing Environment

If this is **NOT** the first time you have used the Python script signing tool on your computer, you can skip this section.

Download and Install Python v3.12 or later from https://www.python.org/downloads/.

If this is the first time you are using the Python script signing tool on your system, you will need to install the dependencies required for the script to work:

- From the included example project sets (refer to Figure 9), choose the set of projects you would like to do first.

- Import that set of projects into a workspace. In this example, we assume you have chosen to import the projects under folder: `\ra6m4_bootloader.`
- Navigate to folder `\MCUboot` in the bootloader project included, for example, `ra6m4_bootloader>ra>mcu-tools>MCUboot,` right click, and select **Command Prompt**. Depending on your PC policy, administrator privileges may be required when running the Command Prompt. This opens a command window with the path set to the `\mcu-tools\MCUboot` folder.

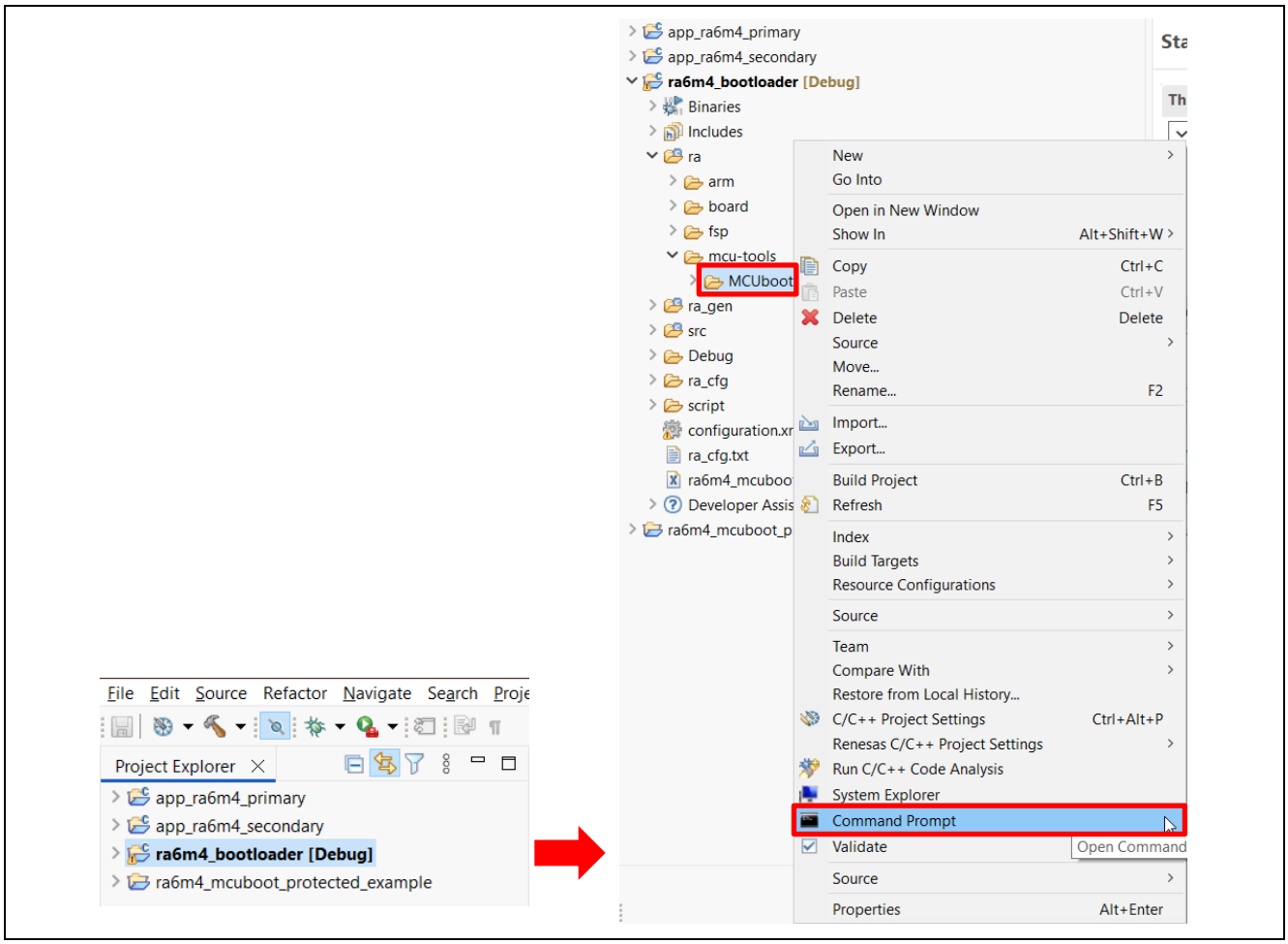


Figure 38. Open the Command Prompt

- We recommend upgrading pip prior to installing the dependencies. Enter the following command to update pip:

  ```
  python -m pip install --upgrade pip
  ```

- Note that if you have multiple Python versions installed, make sure to check that the Python version is version 3.9.0 or later.
- Next, in the command window, enter the following command line to install all the MCUboot dependencies:

  ```
  pip3 install --user -r scripts/requirements.txt
  ```

This will verify and install any dependencies that are required. Make sure this step runs successfully prior to moving to the following sections. If your project path contains special characters or spaces, an error may occur when executing the python script.

4.2 Running the application

Follow the steps below to run the example projects for EK-RA6M4 using the MCUboot Protected Module with Swap Test Update mode.

4.2.1 Set up the Hardware

- Jumper setting: J12 is set to pins 2-3 and J15 is closed.
- Connect J10 using a USB micro to B cable from EK-RA6M4 to the development PC to provide power and debug connection using the on-board debugger.

Once the EK-RA6M4 is powered up, initialize the MCU prior to exercising the bootloader project.

Erase the entire MCU flash and ensure the MCU is in Secure Software Development Device Lifecycle State. This can be achieved using the Renesas Device Partition Manager.

1. Power cycle the board, launch e^2 studio, and open the Renesas Device Partition Manager.

Figure 39. Open Renesas Device Partition Manager

2. Select **Read current device information**.
   If the DLM state is SSD, NSECSD, or DPL, proceed to step 3. Otherwise, you must switch to a different kit to continue the rest of the operation. Below is an example of the readout from an RA6M4 MCU that is in the SSD state.

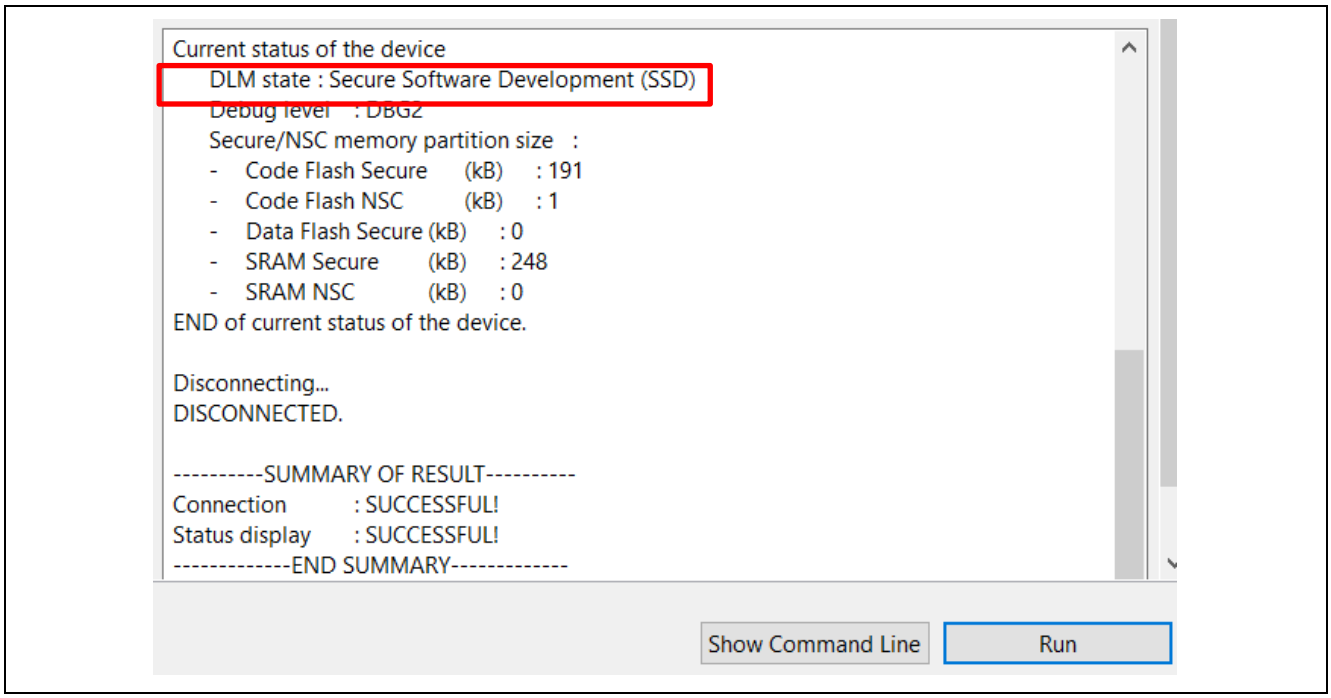


Figure 40. Read the Device Lifecycle States

3. Select **Initialize device**, choose **J-Link** as the connection method, and click **Run**.

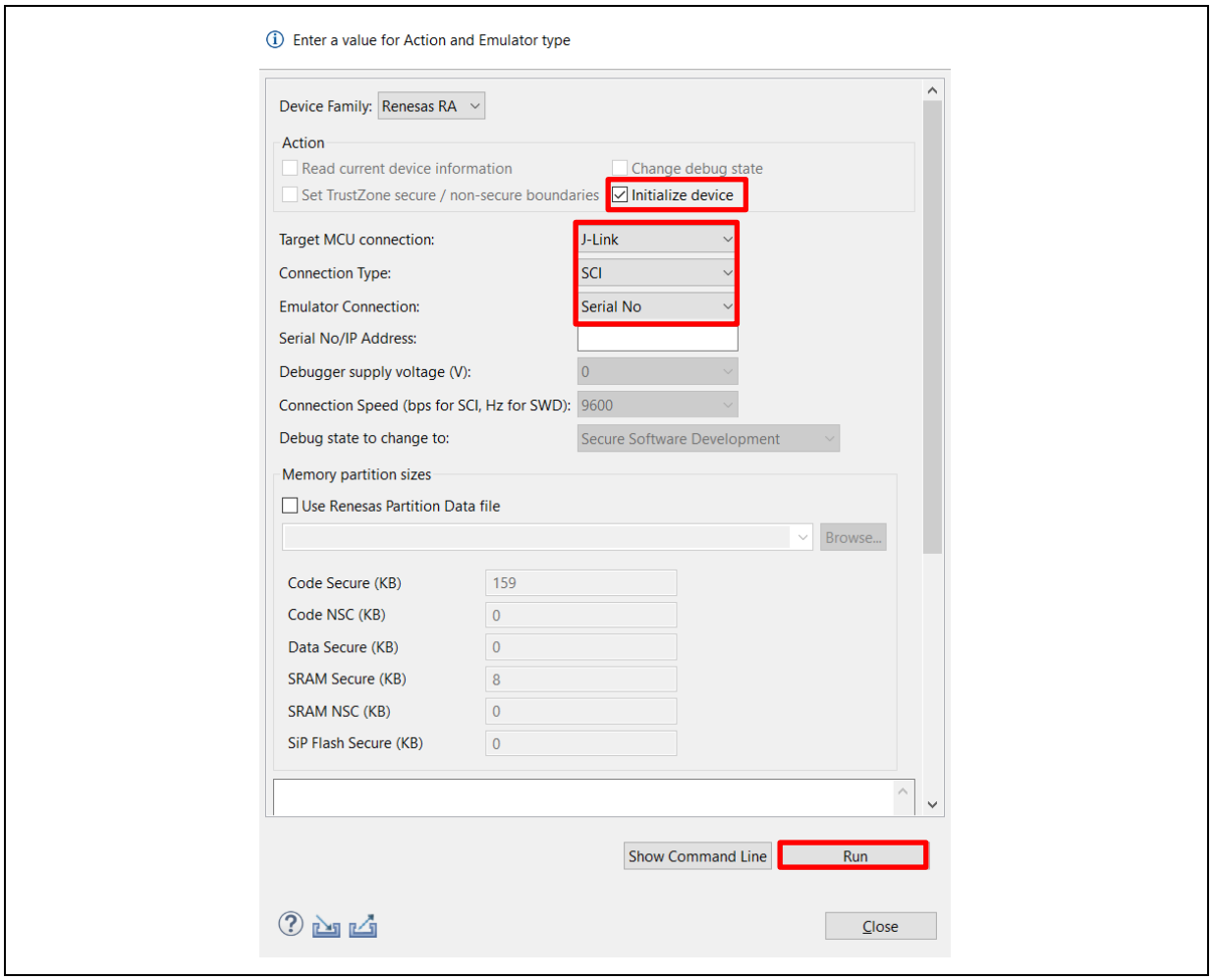


Figure 41. Initialize RA6M4 using Renesas Device Partition Manager

The entire flash will be erased if there are not permanently locked down sections. In addition, if the device is in the NSECSD or DPL state, the RA6M4 will be initialized to the SSD state.

4. Power cycle the EK-RA6M4 after successfully initializing the device to the SSD state by disconnecting the USB cable and reconnecting it to the development PC.

4.2.2 Compile project

Launch e² studio and import `RA6_MCUboot_Protected_Mode.zip` file to a workspace.

New users should refer to the FSP User's Manual section on Importing Projects into the IDE for guidelines. Ensure the Python signing environment is set up referencing section 4.1.

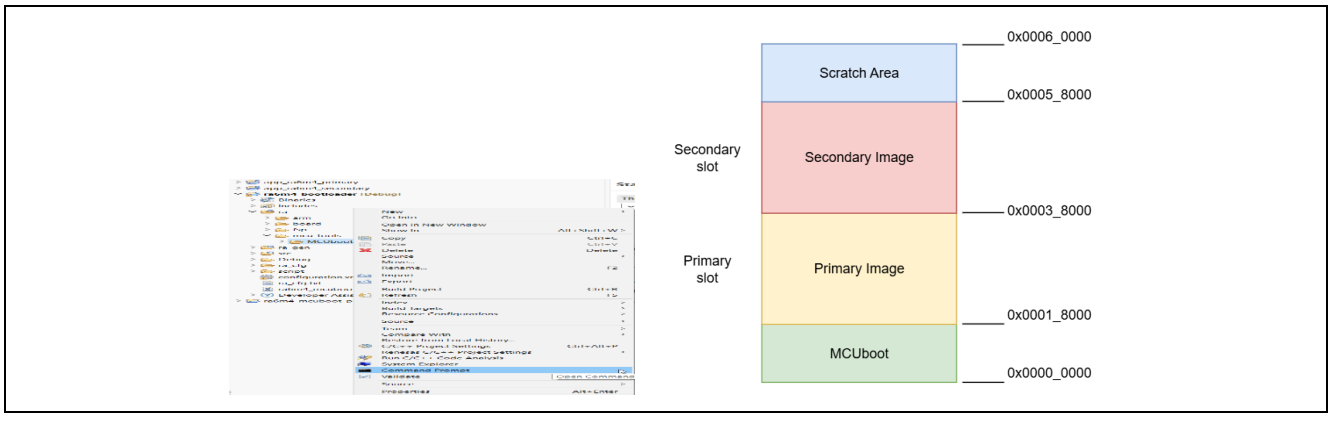


Figure 42. Example Projects for RA6M4 Swap Update Mode

- **ra6m4_bootloader**: The bootloader project configured with Swap update mode.
- **app_ra6m4_primary**: The Primary application project which calls the flash driver to erase and write to a code flash region defined in the code flash area. Upon successful flash operation, all three LEDs blink.
- **app_ra6m4_secondary**: The Secondary application project has the same functionality as the Primary application. Upon successful flash operation, only the green LEDs blink.
- **ra6m4_mcuboot_protected_example**: The Solution project used to link bootloader with Primary application to define memory map for bootloader, primary slot and secondary slot.

Right click to `ra6m4_mcuboot_protected_example`, choose **Build Project**.

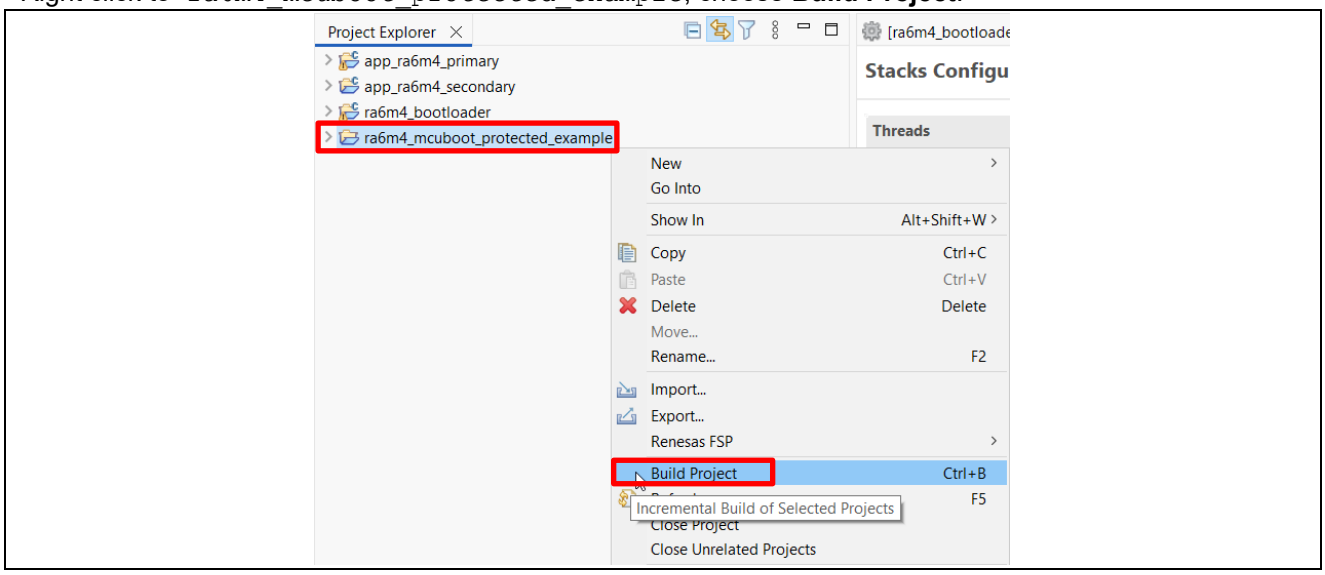


Figure 43. Compile solution project

After selecting **Build Project** for the Solution project, the bootloader project will be compiled first, followed by the primary application project. For secondary project, users must compile it separately after the solution has been completely built. To build the secondary project, choose `app_ra6m4_secondary` project, open the `configuration.xml` file, click **Generate Project Contents** and then click to build the project.

For the application projects, the post-build command will also sign the corresponding images. The signed image for the application project is located under the `/Debug` folder and is named `<application_project_name>_bin.signed` (For example, `/app_ra6m4_primary/Debug/app_ra6m4_primary.bin.signed`).

4.2.3 Secure Key Injection and Bootloader Programming via MCU Boot Interface

After initializing the evaluation board, power-cycle the board and follow the steps below to inject included example ECC public key and program bootloader image for EK-RA6M4 using the Renesas Flash Programmer.

- Note: Please do not use the example keys for production support.

1. Unzip `RA6_MCUboot_Protected_Mode.zip`
2. Launch the Renesas Flash Programmer GUI executable located in `\RA6_MCUboot_Protected_Mode\rfp_resources_ra6m4\rfp_resources_ra6m4.rpj`.

Under the **Operation** tab, click **Add/Remove Files**. Next, click **Add Files**, and then add the `.rkey` file containing the ECC public key, which for this example is `\RA6_MCUboot_Protected_Mode \rfp_resources_ra6m4\user_keys\ECC_pub_install.rkey` (Figure 33). Set the **Address** property to code flash address defined by the `mcuboot_sce9_key` section in `ra6m4_bootloader.map` file and click **OK**. In this example, the ECC public key will be injected at 0x0000A680.

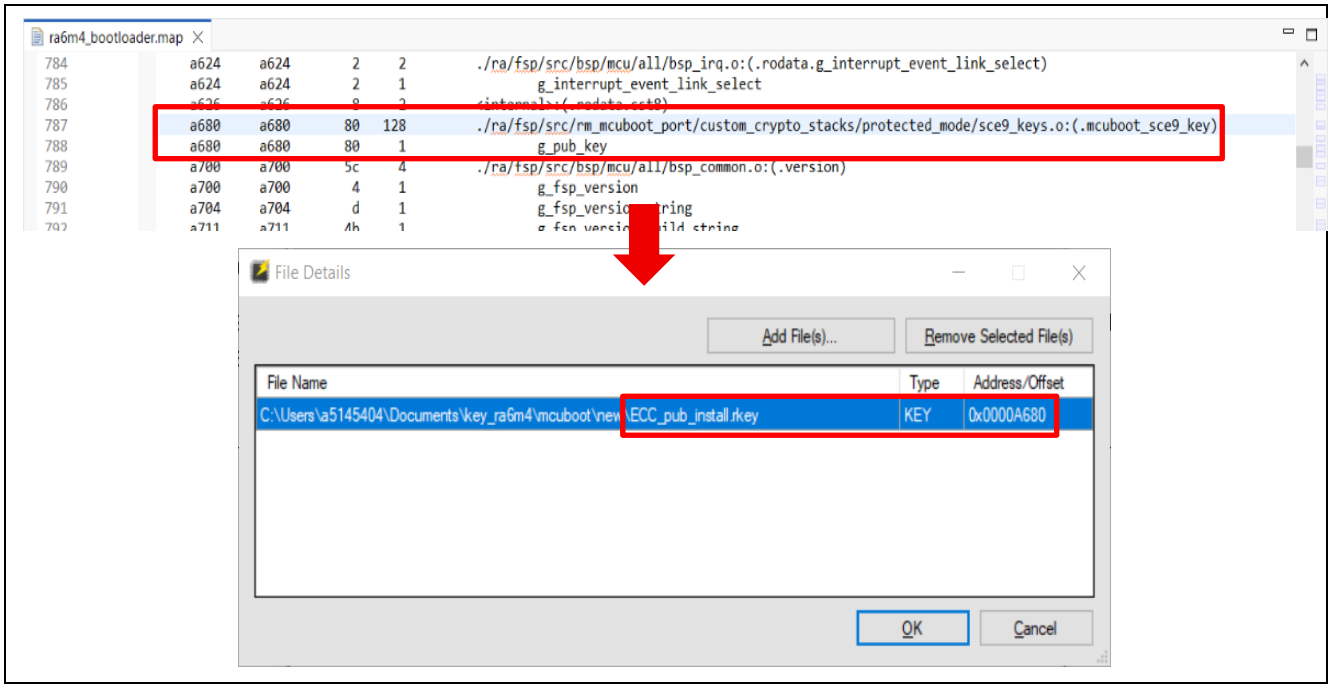


Figure 44. Configure the ECC Public Key Selection and Injection Address

Click **Add Files** again and add bootloader image is located under the `/Debug` folder then click **OK**. (For example, `/ra6m4_bootloader/Debug/ra6m4_bootloader.srec`)

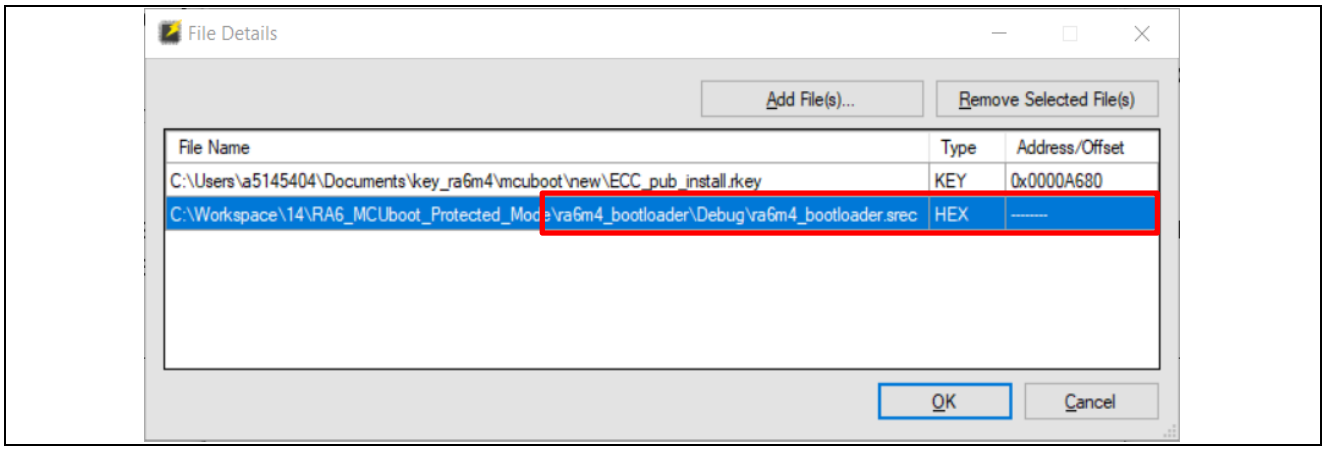


Figure 45. Configure the Bootloader Image

Click **OK** and navigate to the **Operation Settings**. Note that **Erase**, **Program**, **Verify**, and **Erase Before Program** are selected.

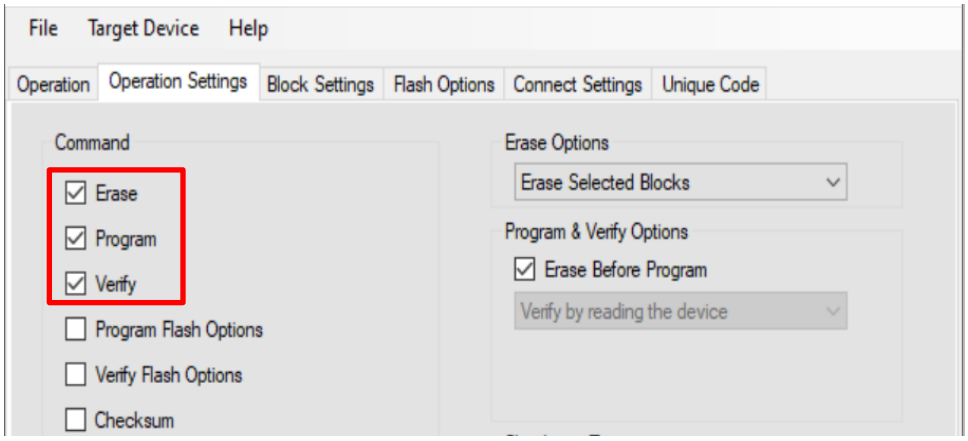


Figure 46. Select to Perform Flash Erase, Program, and Verify

- Browse to the **Block Settings** tab and note that the entire flash region is selected for Erase.

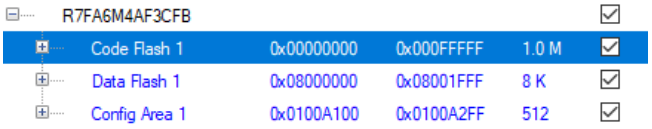


Figure 47. Entire Flash Region is Selected for Erase

Browse the **Operation** tab. Click **Start** to inject the ECC public key. The bootloader programming and user key injection process should succeed with an output message as shown below.

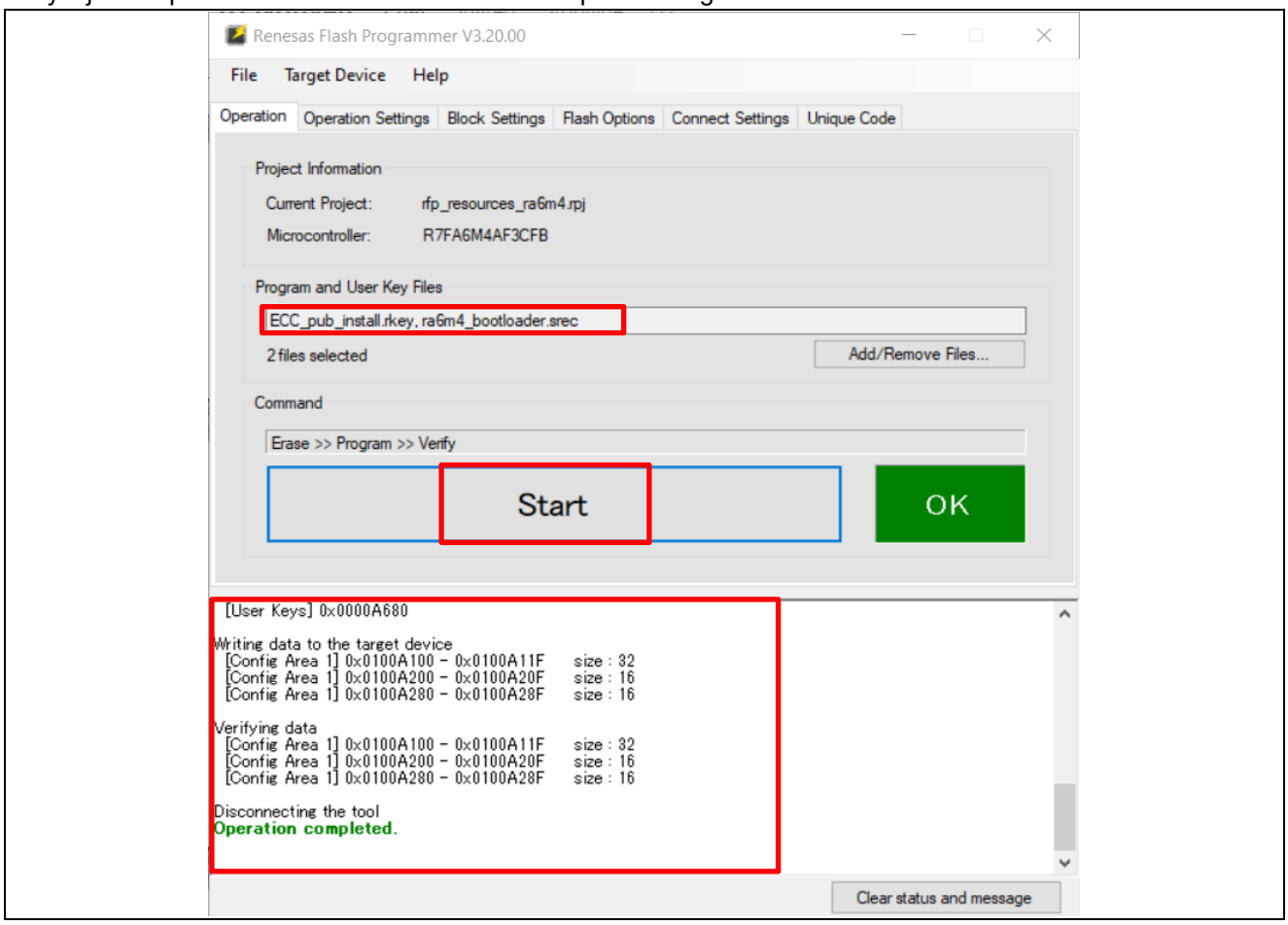


Figure 48. Programming bootloader and Secure Key Injected

4.2.4 Debug and Boot the primary application

Right-click on project `app_ra6m4_primary` and select **Debug As > Debug Configurations -> app_ra6m4_primary_Debug_Flat -> Startup** and confirm the following configuration:

- For the bootloader, select the Symbols only Load type
- For the primary application, configure two entries for the Startup configuration
  - Use the .elf file (app_ra6m4_primary.elf) and select Symbols only Load type
  - Use the signed binary (`app_ra6m4_primary.bin.signed`) and select 'Raw binary" Load type

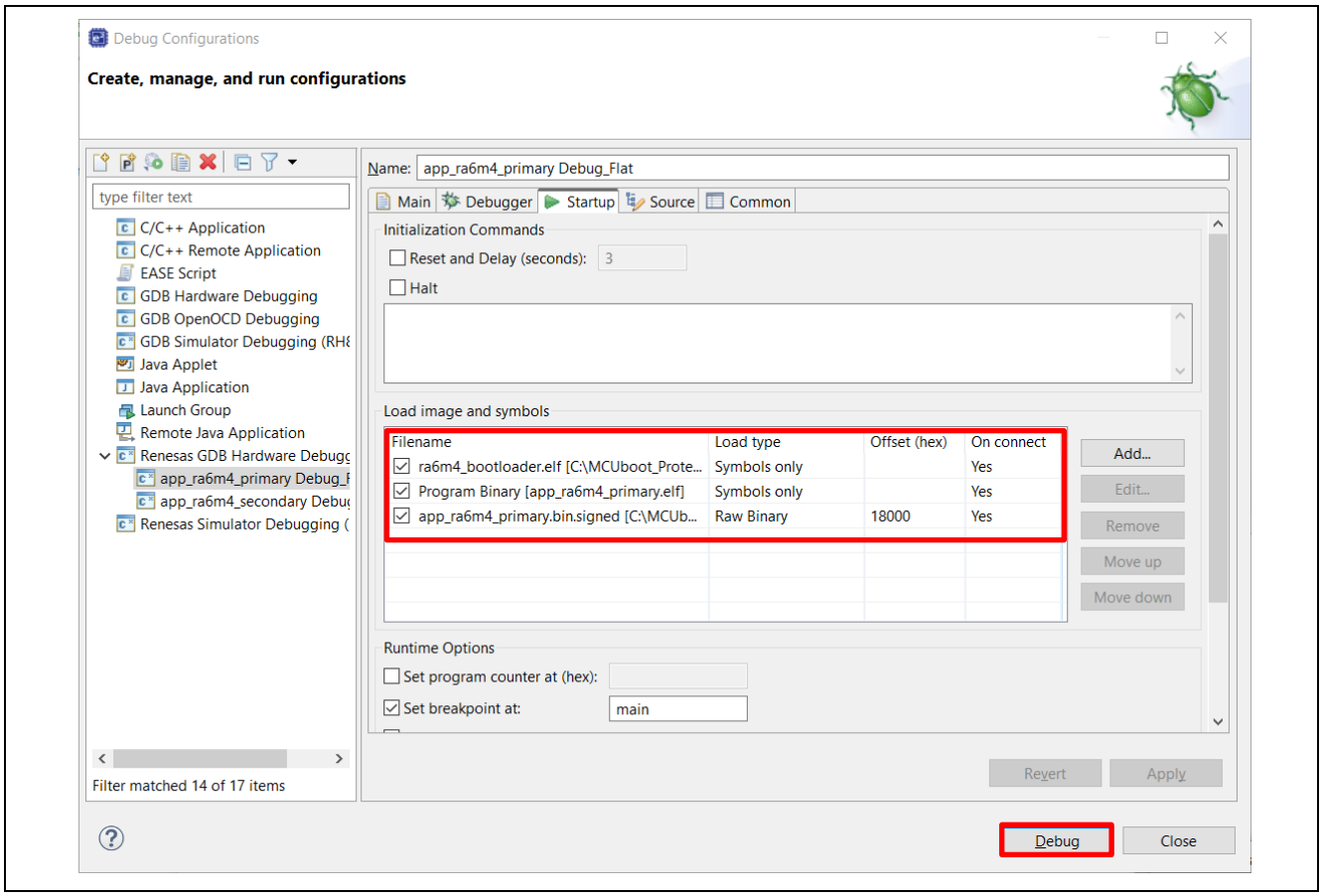


Figure 49. Debug Configuration

Click **Debug**.

The debugger should hit the reset handler in the bootloader.

Figure 50. Start the Application Execution

Click Resume twice  and boot the Primary image. All three LEDs should be blinking.

4.2.5 Open the J-Link RTT Viewer

Configure the RTT Viewer as shown below. Set the search address to 0x20004420. If the user has modified this example project, the address may be different. In that case, refer to the application's .map file and look for the _SEGGER_RTT section to determine the correct address.

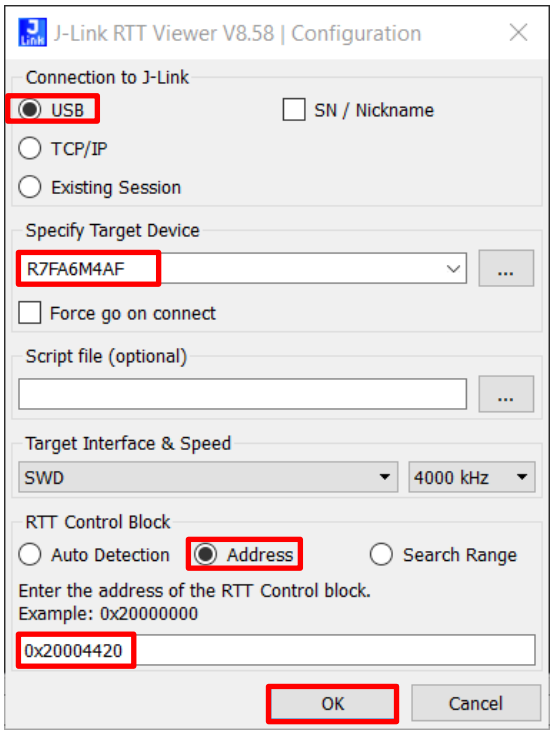


Figure 51. Configure the RTT Viewer

Click **OK** and observe the output on the RTT Viewer. This repeated output shows the Primary application is being executed and all three LEDs are blinking.

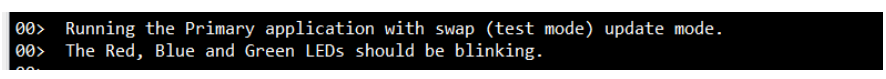


Figure 52. Execution of Primary Application

4.2.6 Download and Run secondary application

During development, you can use the ancillary loading capability to load the new image to the intended location. You can use the example new secondary application provided and follow the steps below to perform an application upgrade:

1. Press the button to pause the program.
2. On the top of the e² studio toolbar, click the Load Ancillary File button to load the new application images to the Secondary slot region. Refer to section 4.3 for troubleshooting when using the Load Ancillary File function.

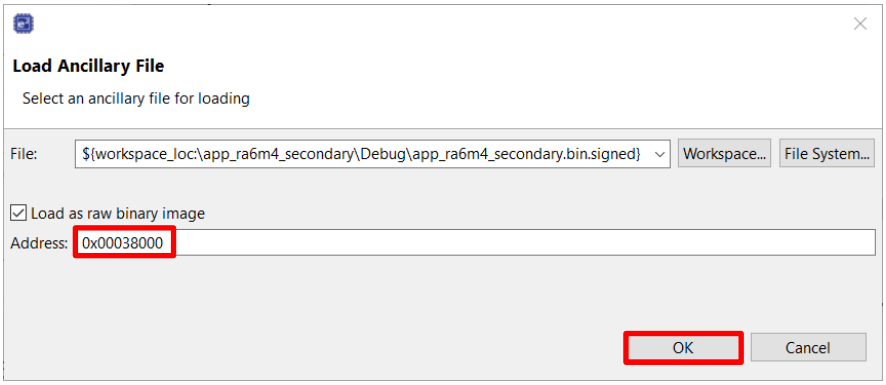


Figure 53. Load the Secondary Application Image

3. Click **Resume** . The swap occurs and the new image is executed. The green led will be blinking instead of all three LEDs.
4. Re-configure the RTT Viewer as shown below. Choose Existing Session and set the search address to 0x20005720. If the user has modified this example project, the address may be different. In that case, refer to the application's .map file and look for the _SEGGER_RTT section to determine the correct address.

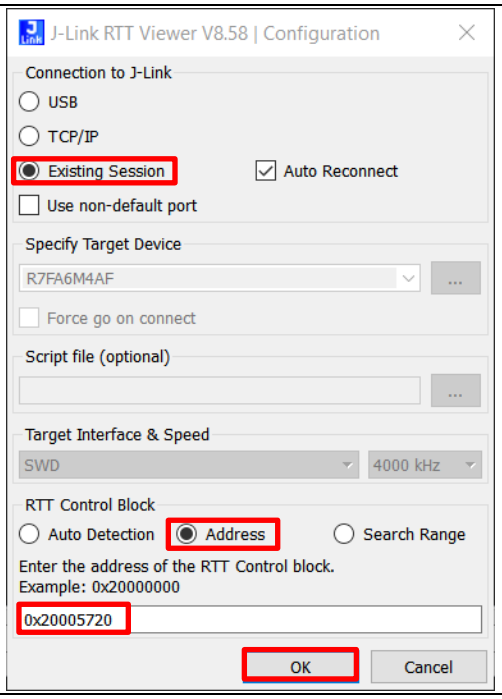


Figure 54. Configure the RTT Viewer

5. Click **OK** then confirm that the following messages are printed on the RTT Viewer output, and only green LEDs are blinking.

```
00>  Running the Secondary application with swap (test mode) update mode.
00>  The Green LED should be blinking.
```

Figure 55. Executing the Secondary image

4.3 Troubleshooting

When running the example projects, you may experience USB Debug connection or the RTT Viewer connection issue when using the “Load Ancillary File” button  to download the Secondary image. To recover from these failures:

- If the USB Debug connection disconnects, the recommendation is to try out another available USB port on the development PC for the USB Debug connection. If failure persists, contact Renesas support.
- If the RTT Viewer disconnects, the recommendation is to power cycle the board and restart the debug session.

5. Creating the Bootloader

The screen captures used in these sections are based on the RA6M4 based bootloader projects used in section 4.2. Follow this section to establish the bootloader projects used with Swap as the application update mode.

5.1 Start Bootloader Project Creation with e² studio

Follow the steps below to create the initial bootloader project based on EK-RA6M4:

1. From the e² studio Workspace, navigate to the **File > New > Renesas C/C++ Project > Renesas RA** and then select **Renesas RA C/C++ Project** and press **Next.**
   Provide the project name `ra6m4_bootloader` and click **Next**. The exact name needs to be provided to follow the default instructions in this section. If a different name is provided, all instructions related with the name of the bootloader project need to be updated accordingly.
   In the next screen, select **FSP version 6.1.0** and the **EK-RA6M4** board. Use the default **Debugger** setting **J-Link Arm**, **Toolchain** is **LLVM** version 18.1.3 and click **Next.**
   Note that if the creation process is using other newer FSP versions, some details on the error messages shown when the MCUboot module is initially added may be different. Adapt the actions accordingly to satisfy the dependencies.

2. When the following screen appears, select **Flat (Non-TrustZone) Project**.

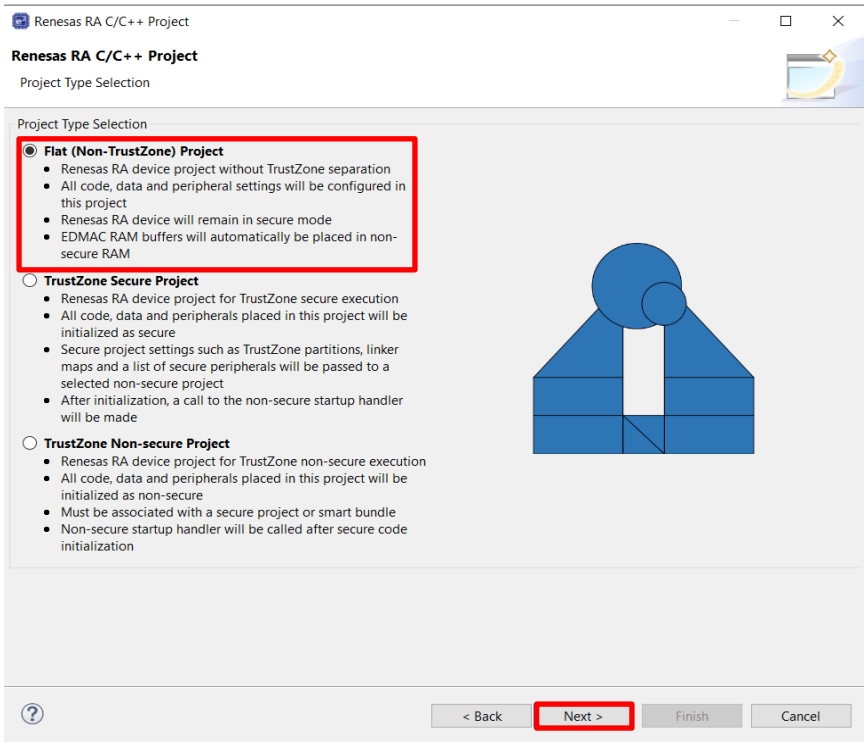


Figure 56. Choose Flat Project as Project Type

3. Choose **None** for Preceding Project or Smart Bundle Selection and click **Next**

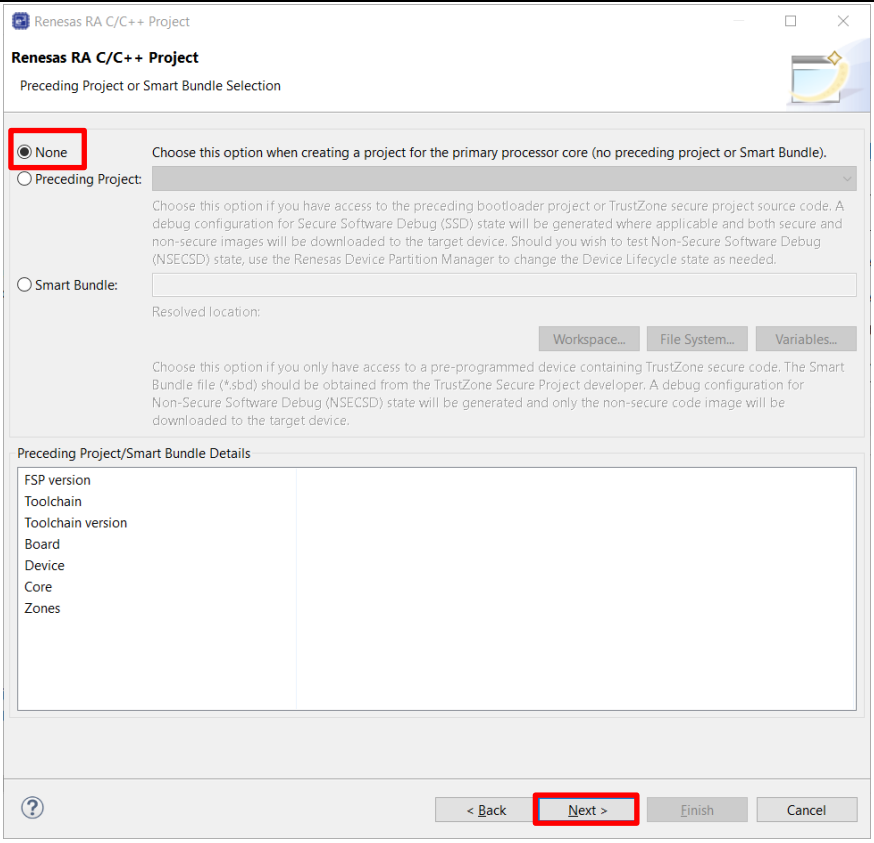


Figure 57. Configure Preceding Project

4. Choose **Executable** as the **Build Artifact Selection** and **No RTOS**. Click **Next**.

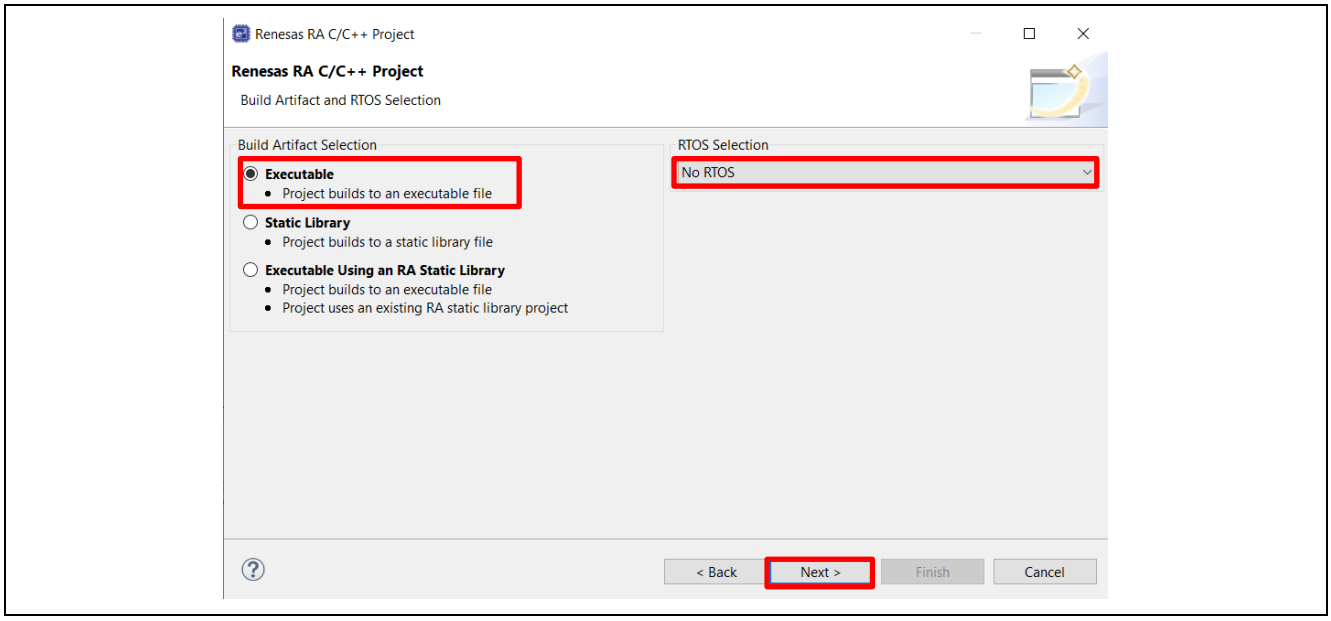


Figure 58. Choose Executable and No RTOS

5. In the next screen, select the project template.
   Choose **Bare-Metal – Minimal** as the **Project Template Selection** and click **Finish**.

6. Add the MCUboot module.
   The project will now be created, and the bootloader project configuration will be displayed. Change to the **Stacks** tab and select **New Stack > Bootloader > MCUboot**.

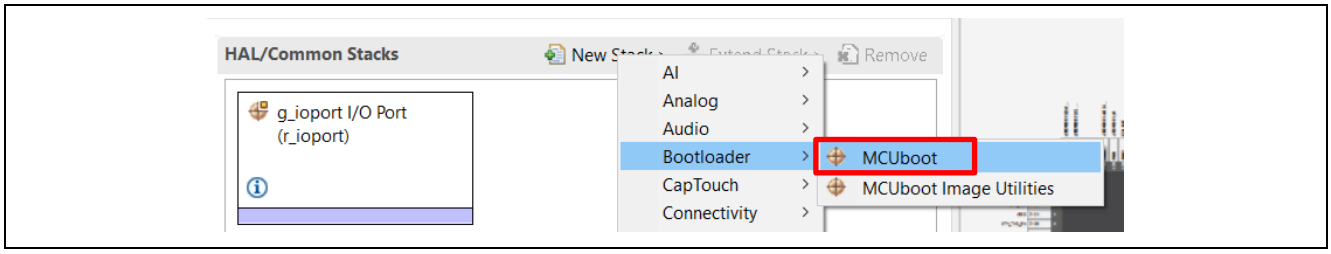


Figure 59. Add the MCUboot Module

5.2 Resolve the Configurator Dependencies

After the MCUboot module is brought into the configurator, follow the steps in this section to resolve the dependencies:

1. Resolve the following dependency of the MCUboot by adding the **MCUboot Custom Crypto (Protected Mode)** stack.

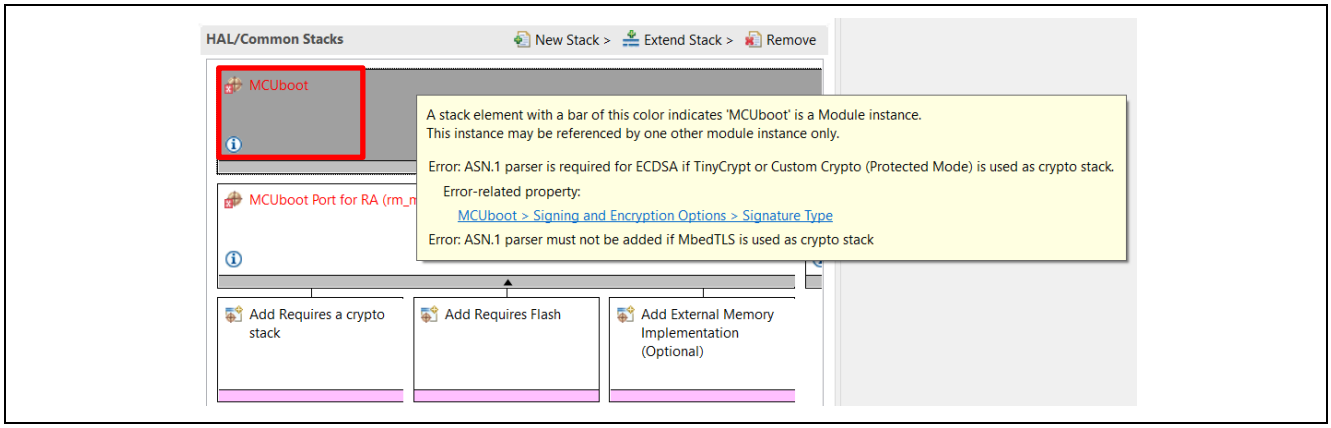


Figure 60. MCUboot Module Dependency

Left click on **Add Requires a crypto stack**, choose **New** and add the **MCUboot Custom Crypto (Protected Mode)** stack.

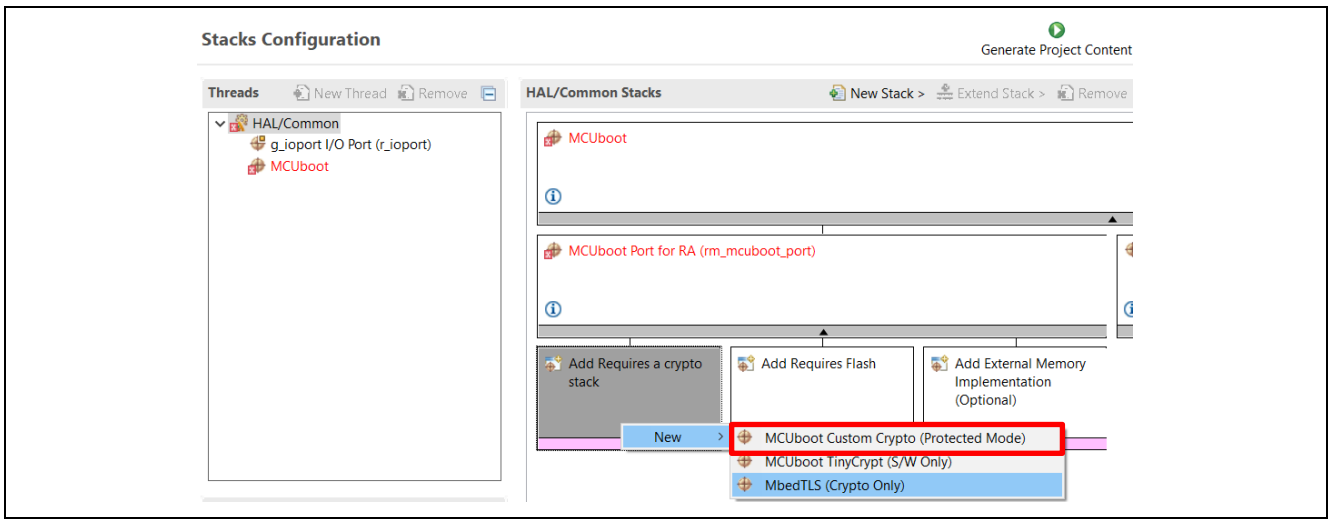


Figure 61. Add MCUboot Custom Crypto (Protected Mode) Module

2. Configure the MCUboot dependencies following the error shown in figure below:

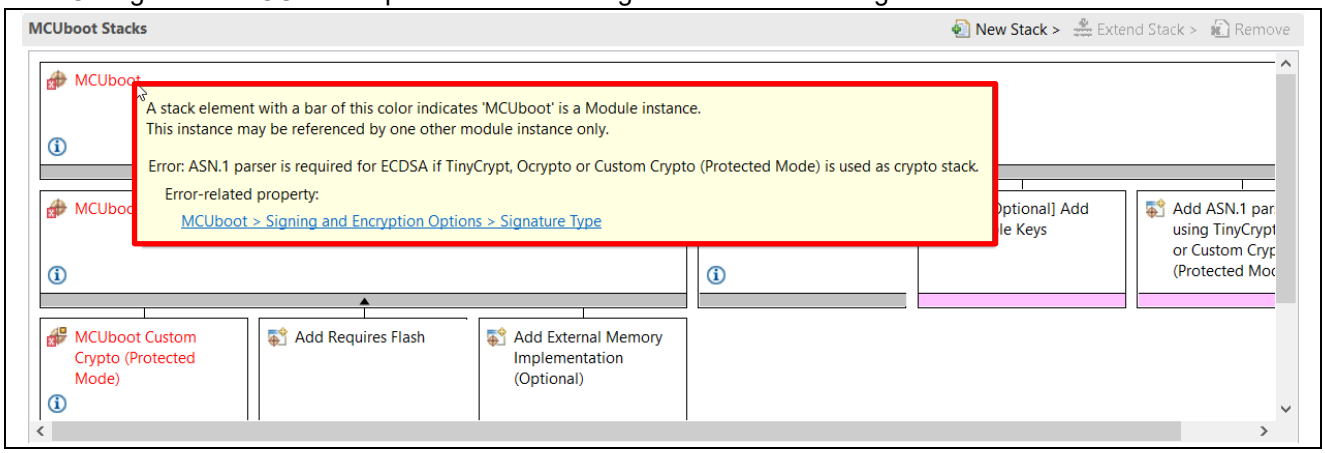


Figure 62. Dependencies of MCUboot Module for RA Stack

Add ASN.1 Parser stack:

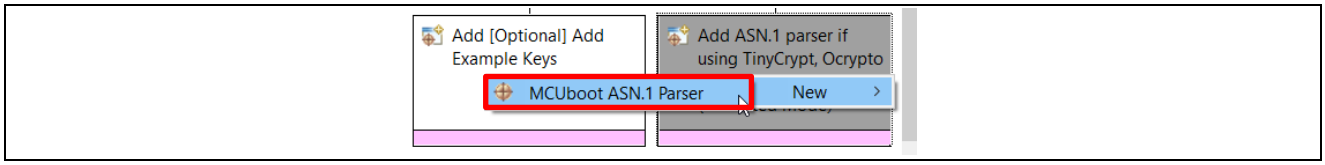


Figure 63. Add ASN.1 Parser Stack

3. Configure the “MCUboot Port for RA” dependencies.
   Follow the prompt in Figure 64 to update the corresponding properties for the MCUboot Port for RA Module.

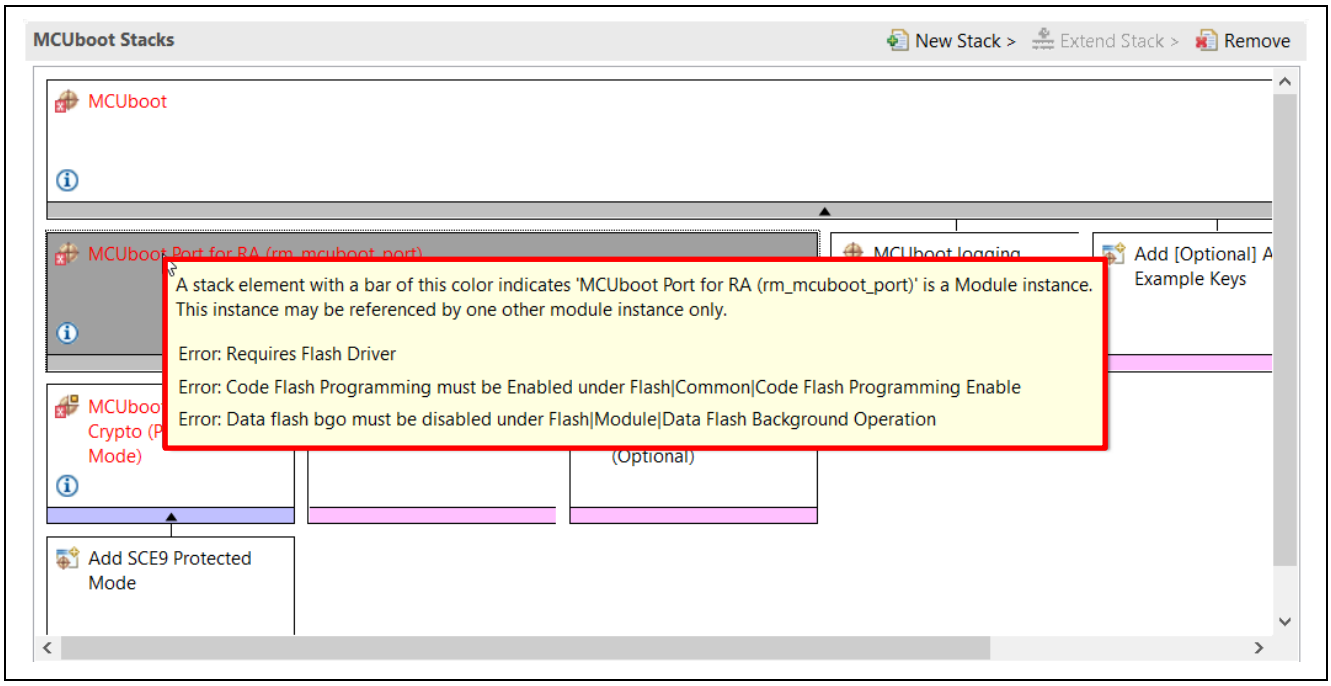


Figure 64. Dependencies of MCUboot Module for RA Stack

Add the r_flash_hp module:

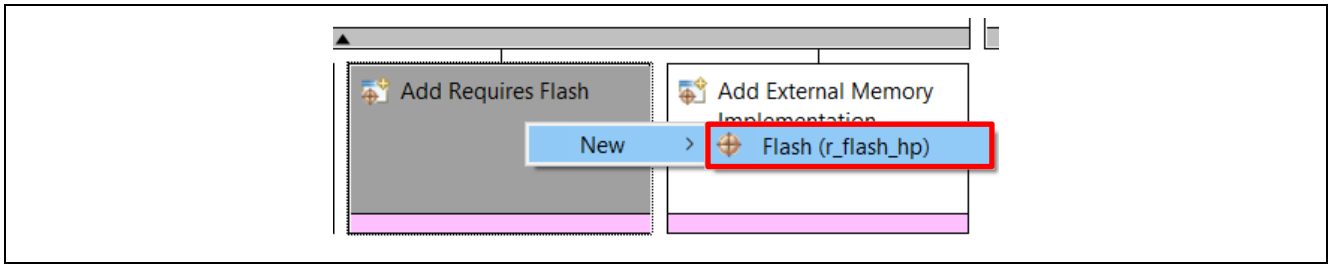


Figure 65. Add the r_flash_hp Stack

Configure the r_flash_hp stack:

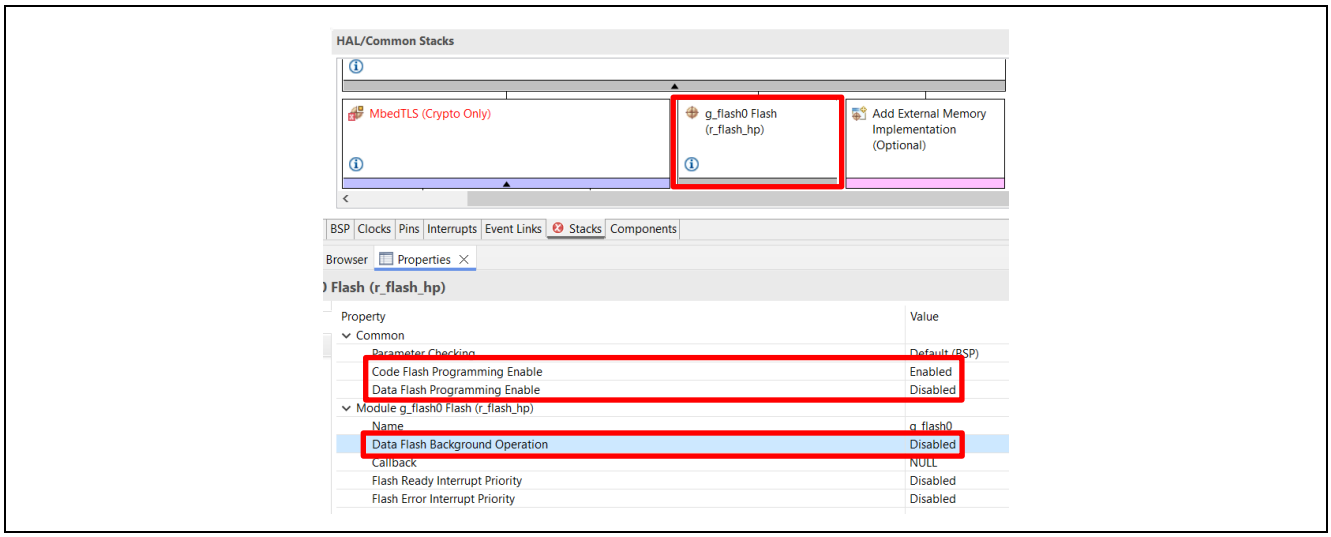
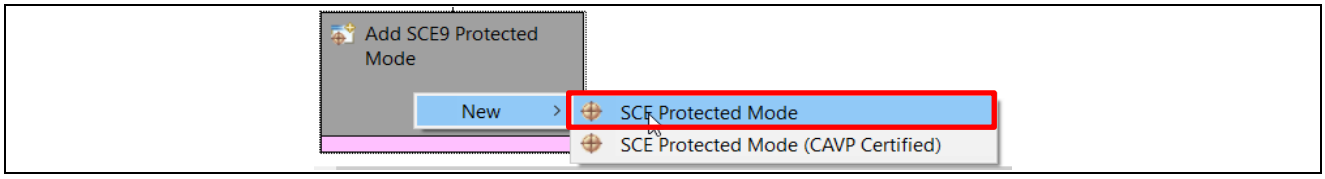


Figure 66. Configure the r_flash_hp Stack

Add SCE9 Protected Module:



4. Under the BSP tab, set up the stack and heap size to resolve the error when hover the cursor over **MCUboot Custom Crypto (Protected Mode)** stack as shown in **Figure 67**:
   - **RA Common** > (set **Main stack size** to 0x1000 and **Heap size** to 0x400)
   - **RA6M4 Device Options > OFS Registers > DUALSEL Settings: Enable**
   - **RA6M4 Device Options > OFS Registers > DUALSEL Settings > Bank Mode: Linear**

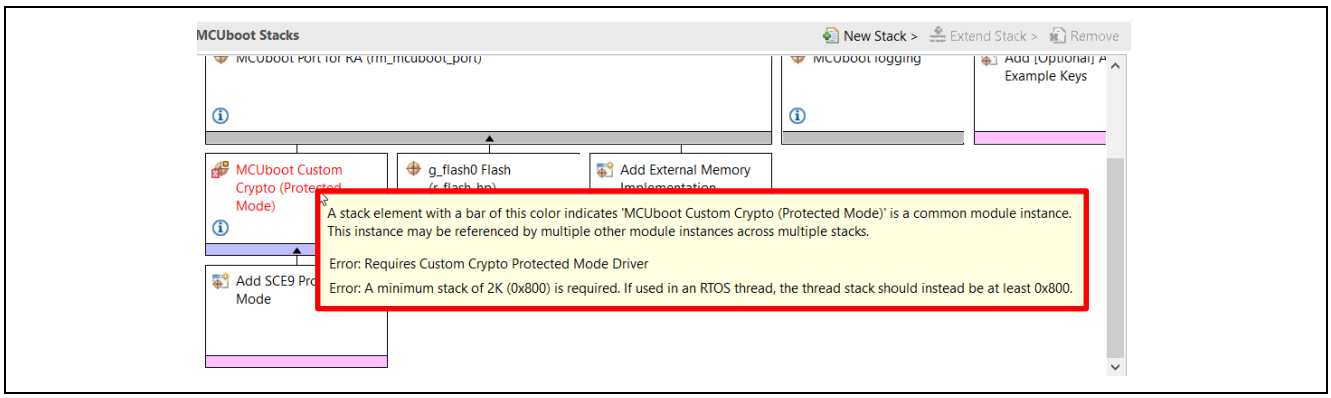


Figure 67. Dependencies of Mbed TLS (Crypto Only) Stack

5.3 Setting up the Booting Authentication Support

You can choose to use the default pair of public/private keys included in MCUboot for testing purposes:

- The default public keys are defined in `/ra6m4_bootloader/ra/mcu-tools/MCUboot/sim/mcuboot-sys/csupport/keys.c`.
- The default private keys are included in folder `/ra_mcuboot_ra6m4/ra/mcu-tools/MCUboot/sim`.

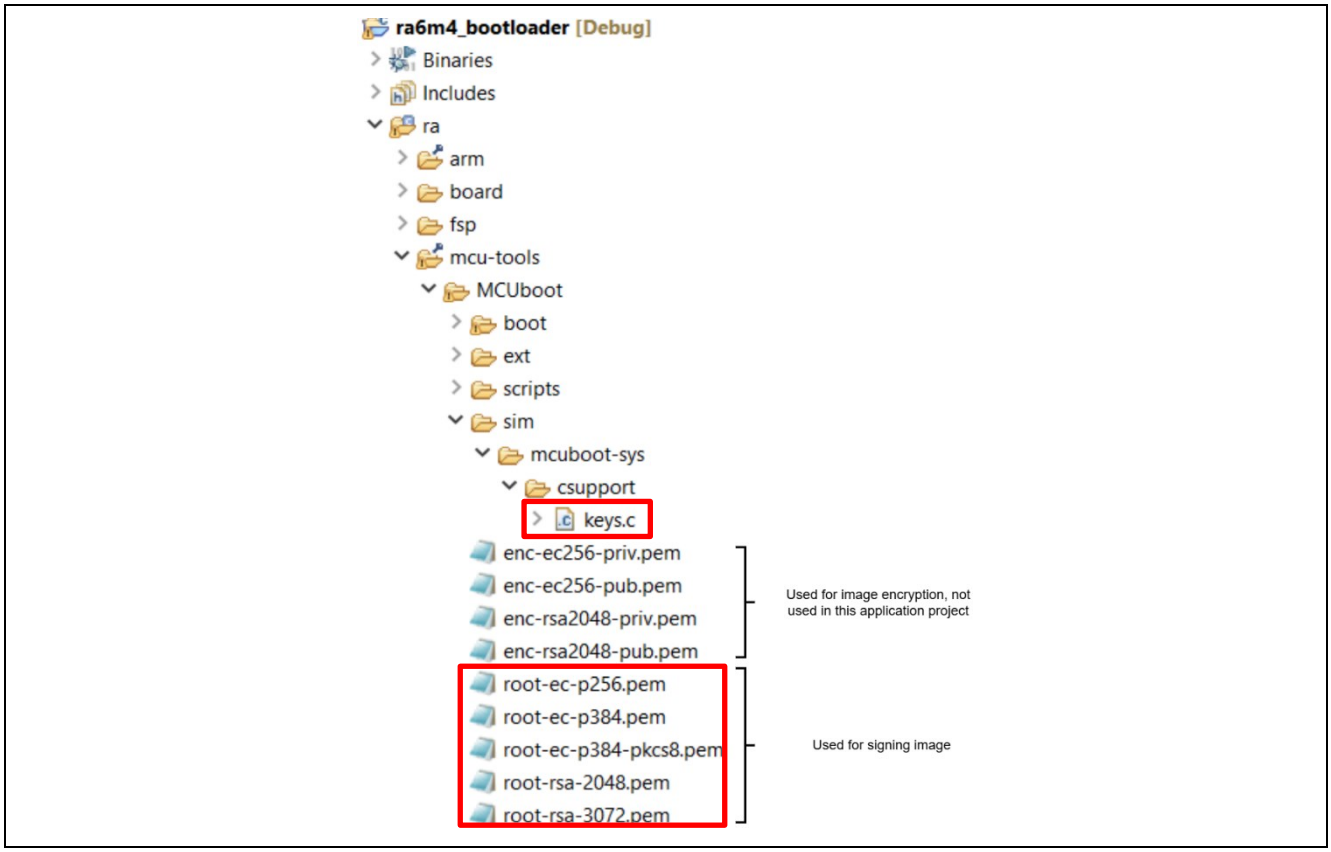


Figure 68. Example Public Keys and Private Keys Included in MCUboot Port Stack

To use the example keys, select **Add Example Keys** > **New** > **MCUboot Example Keys (NOT FOR PRODUCTION)**.

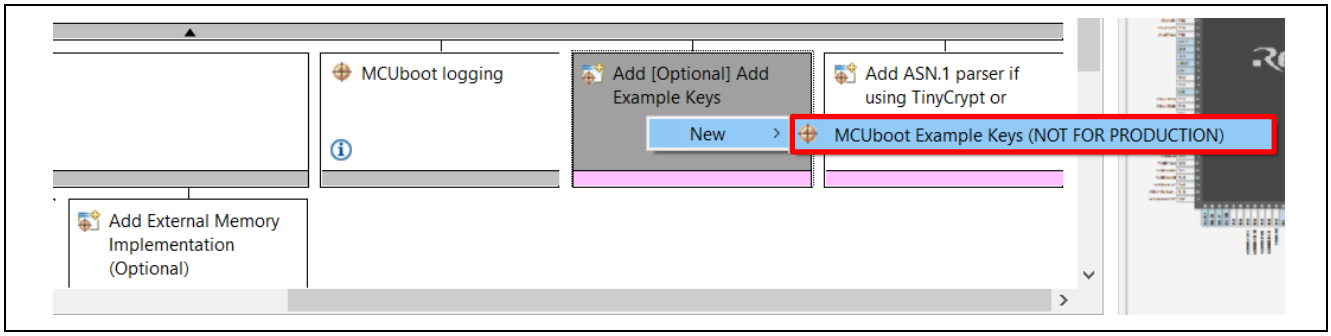
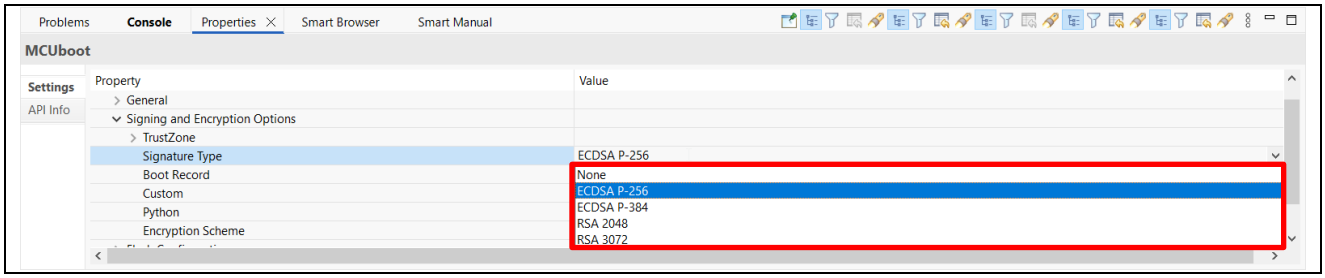


Figure 69. Add the MCUboot Example Key

Note: The example public key and private key used in the MCUboot is for testing purposes only. Application Project R11AN0567 includes procedures to create customized key pair preparation. Refer to R11AN0567 to create customized key pairs.

5.4 Setting up the Application Authentication Signature Type

There are 4 signature types supported in FSP as shown below. Open the **Property** page of stack **MCUboot** > **Common** > **Signing and Encryption Options** to look at the signing options. In this example implementation, ECDSA P-256 is used for the example bootloader.



5.5 Add MCUboot Initialization Code

Follow the steps below to add the MCUboot activation code and compile the bootloader:

1. Add the source code and compile the bootloader.

After creating and setting up the MCUboot module, save the `configuration.xml` file and click **Generate Project Content**, then follow the steps below to add the source code to the bootloader project and compile the project.

Open `hal_entry.c`.

- Open **Developer Assistance**.
- Go to **HAL/Common** > **MCUboot** > **Quick Setup**. Drag **Call Quick Setup** to the top of the `hal_entry.c` file before the `hal_entry()` function call.
- Call this function at the top of the `hal_entry()` function
  - `mcuboot_quick_setup();`

Notes on the `mcuboot_quick_setup` function

- The main functionality established in the bootloader project is established by function `mcuboot_quick_setup`, which performs the following functions:
  - The `boot_go` function does most of the functions of a bootloader except the final step of jumping to the main image. This function returns a structure pointer (`rsp` for return structure pointer) for the image to boot from.
  - The `RM_MCUBOOT_PORT_BootApp` function cleans up resources used by the bootloader and jumps to the application image.

2. Compile the bootloader project.

Save the source code and then compile the project.

6. Using the Bootloader with Application

Follow the steps below to configure the standalone application projects to use the bootloader and sign the application.

6.1 Configure the Application Projects to Use the Bootloader

When creating the application project, at the **Preceding Project or Smart Bundle Selection** step (as shown in Figure 57), make sure to select the previously created bootloader project as the **Preceding Project.** This example is using ra6m4_bootloader project.

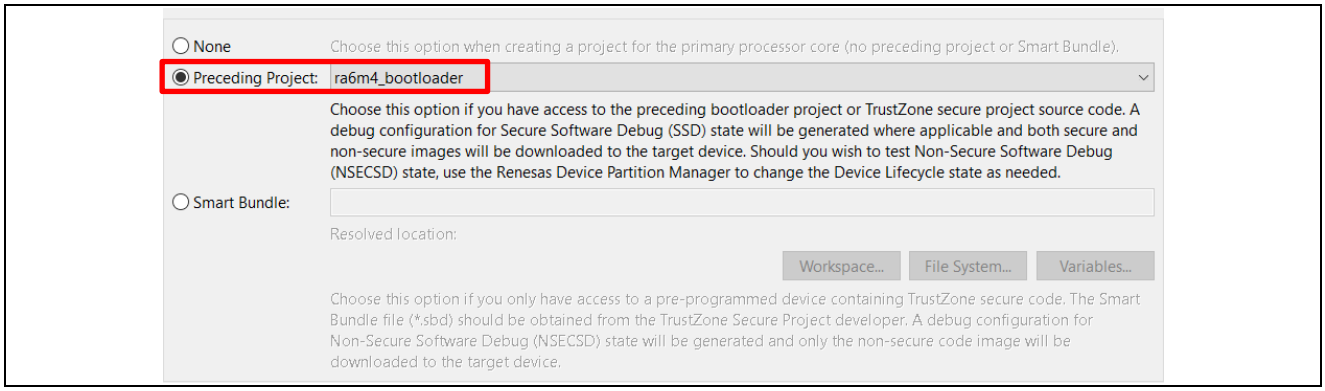


Figure 70. Choose bootloader project as Preceding Project

6.2 Signing the Existing Application Projects to Use the Bootloader

Each application can have a defined version number. This version can be used in the Overwrite Upgrade mode when **Downgrade Prevention** is **Enabled**. This is achieved by defining an Environment Variable: **MCUBOOT_IMAGE_VERSION**. If there is signature verification, then it is necessary to set the Environment Variable: **MCUBOOT_IMAGE_SIGNING_KEY** with value in this case is **${workspace_loc:ra6m4_bootloader}/ra/mcu-tools/MCUboot/root-ec-p256.pem**.

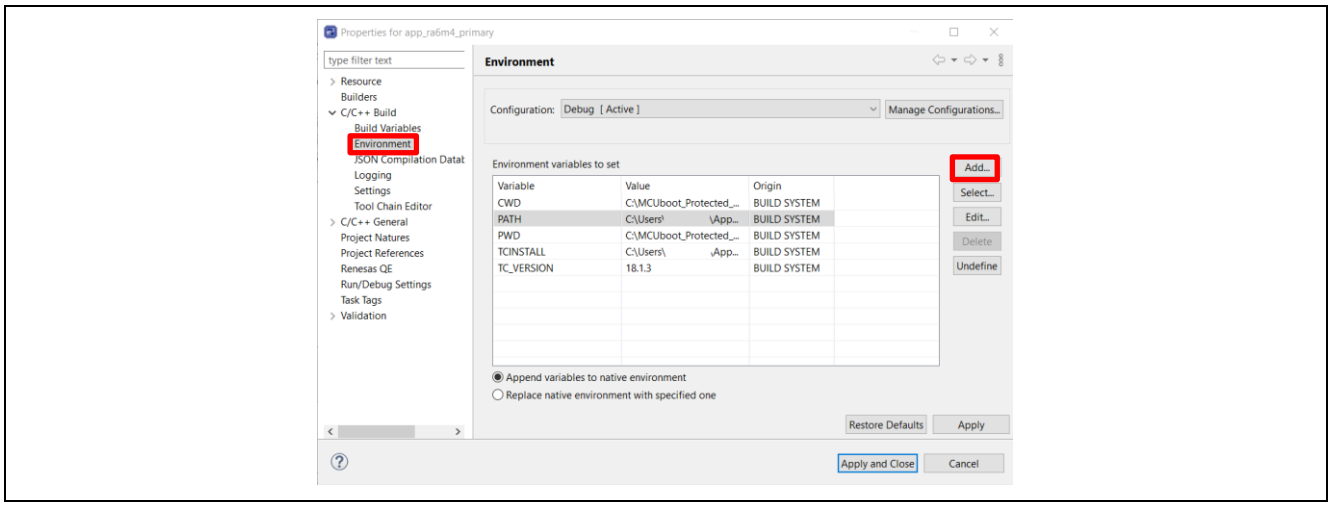


Figure 71. Add New Environment Variable

Add environment variable for the application image version.

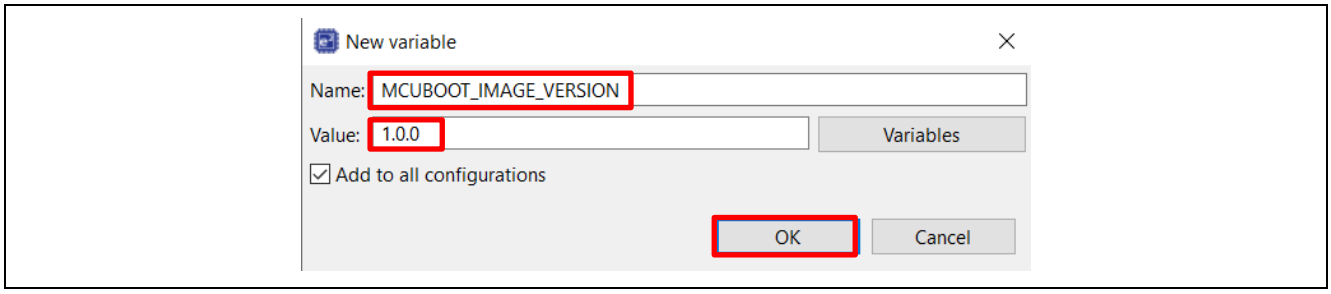


Figure 72. Add MCUBOOT_IMAGE_VERSION Variable

Add an environment variable to configure the application image signing key.

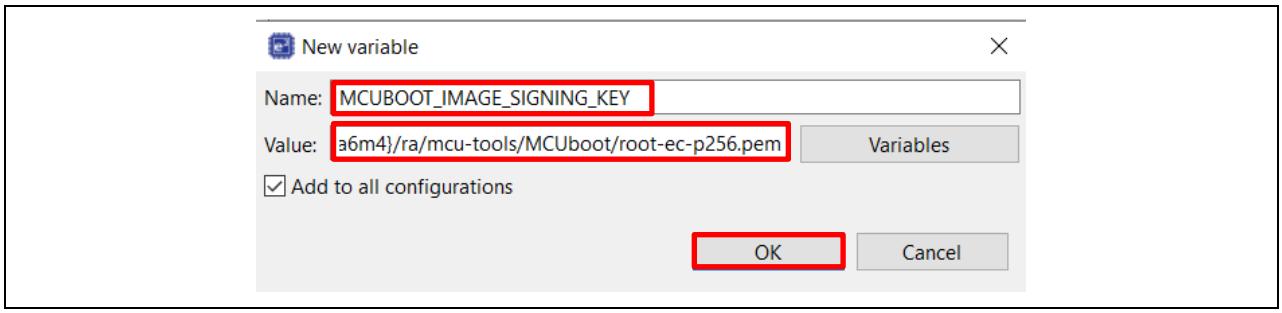


Figure 73. Add MCUBOOT_IMAGE_SIGNING_KEY Variable

Add an environment variable to convert the .elf file to a raw binary format used in the image signing process. Set variable **MCUBOOT_APP_BIN_CONVERTER** to the path of the appropriate tool—objcopy, arm-none-eabi-objcopy, fromelf, or ielftool—depending on the toolchain used for the project. This example project is using LLVM toolchain, so this variable's value is set to **${TCINSTALL}bin\llvm-objcopy.exe**.

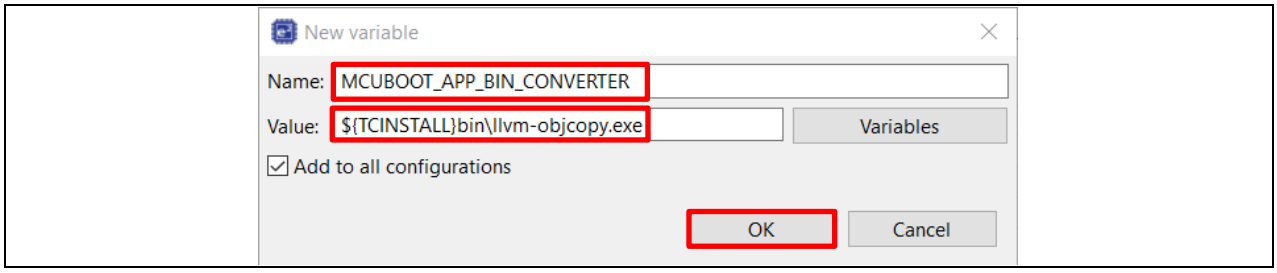


Figure 74. Add MCUBOOT_APP_BIN_CONVERTER Variable

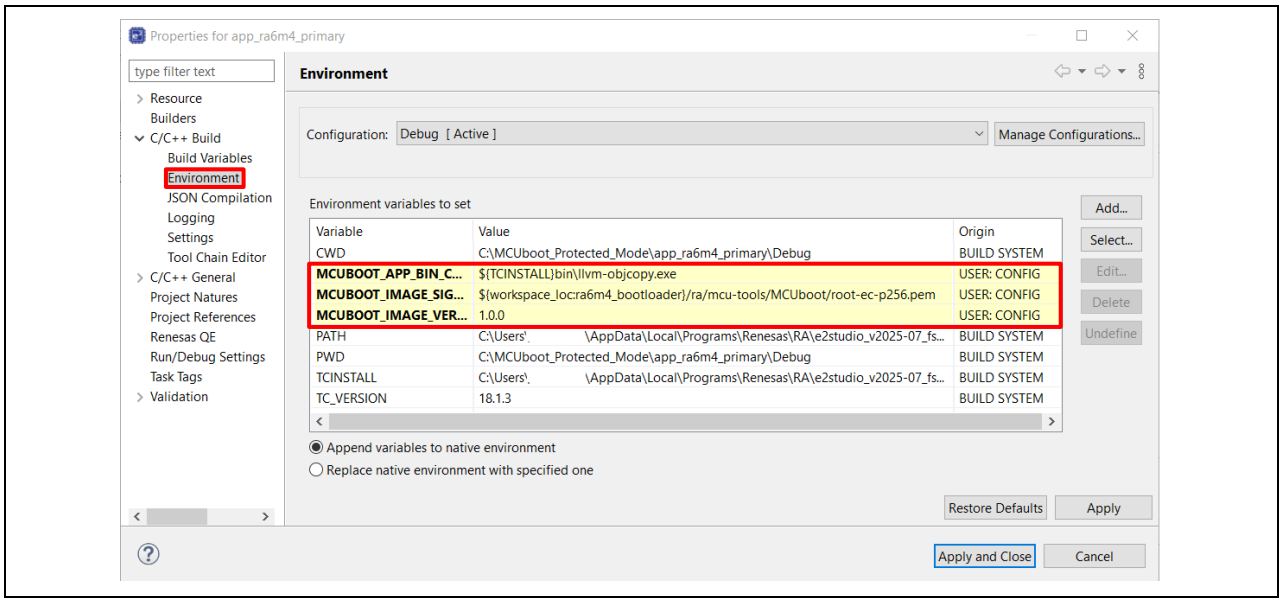


Figure 75. Configure the Signing Key and Application Version

Note: The private key used for signing the application image is indicated in the signing command.

`/ra/mcu-tools/MCUboot/root-ec-p256.pem` is used for the example bootloader. This key is used for testing purposes only. For real world use case and production support, user MUST change this to the private key of their choice.

To be able to always recompile the project when the environment variables or the linker script are updated, we recommend adding a **Pre-build step** to always delete the `.elf` file as shown in Figure 76.

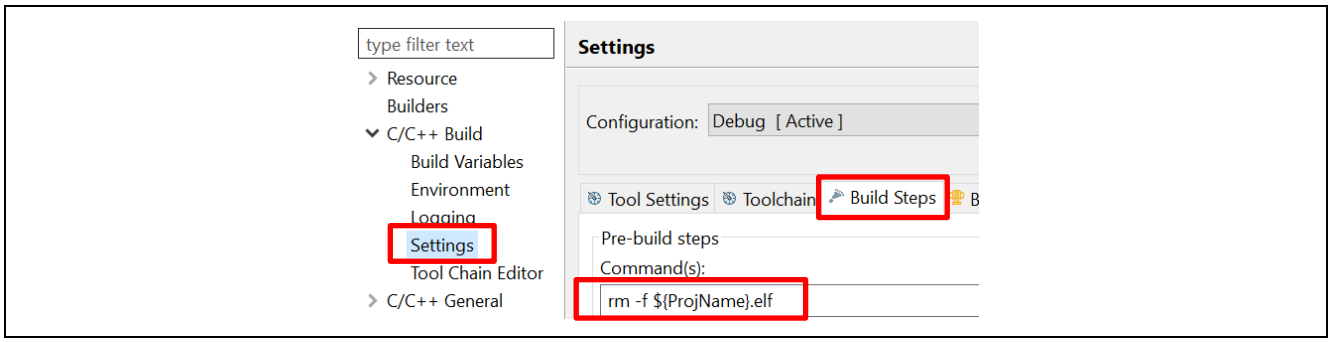


Figure 76. Configure the Pre-build Command

6.3 Linking Application with Bootloader

From FSP 6.0.0, the MCUboot project and application project must be linked by using RA FSP solution project to set up the bootloader slots, the slot size and header information was previously in the MCUboot module. Please ensure that both the bootloader project and the application project have been successfully built before proceeding with the linking process. The following section outlines the steps to create an FSP solution project.

1. From the e² studio Workspace, navigate to the **File > New > C/C++ Project > Renesas RA** and then select **Renesas FSP Solution Project (Advanced)** and press **Next.** Provide the solution project name `ra6m4_mcuboot_protected_example` and click **Next**.
2. When the following screen appears, select the created primary project as the final project of a chain of projects that constitute a complete FSP application and click **Finish**.

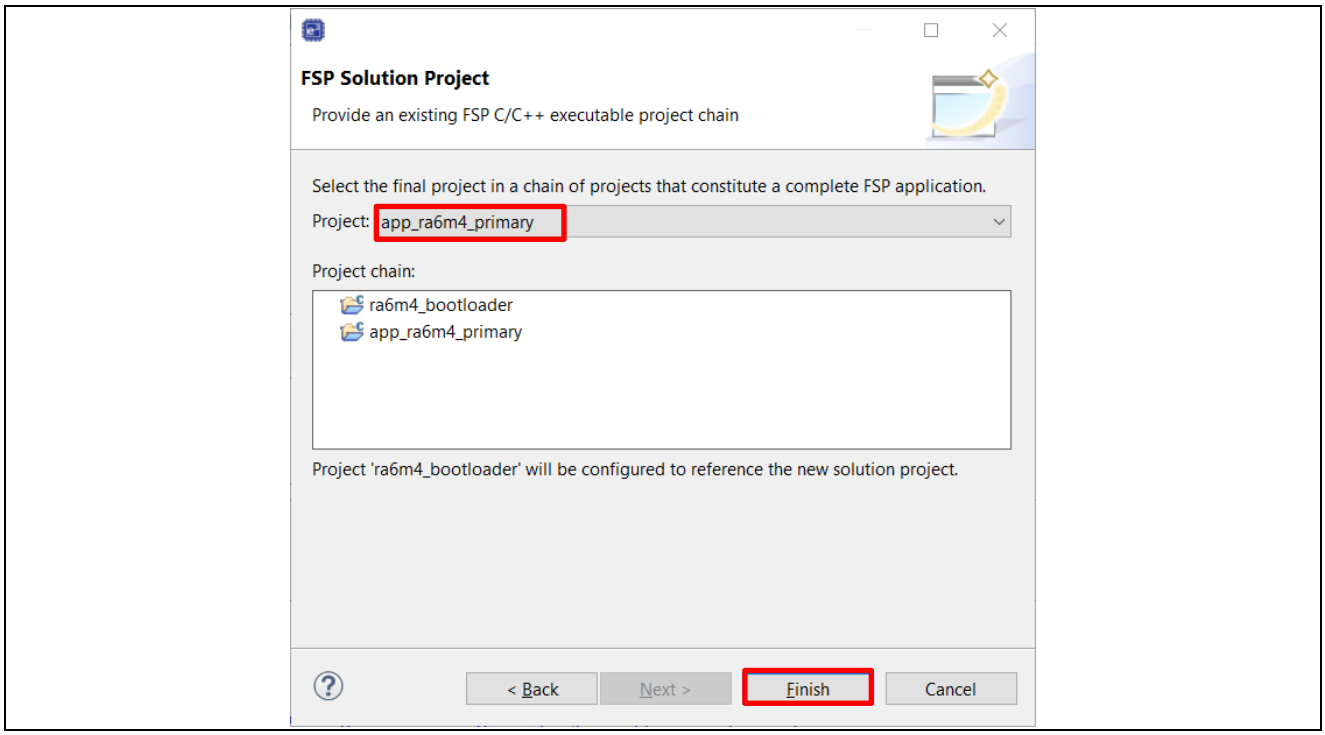


Figure 77. Choose primary application as final project

3. The solution project will now be created, and the project configuration will be displayed. Select the **Memories** tab then click on **Flash > Add Partitions**.

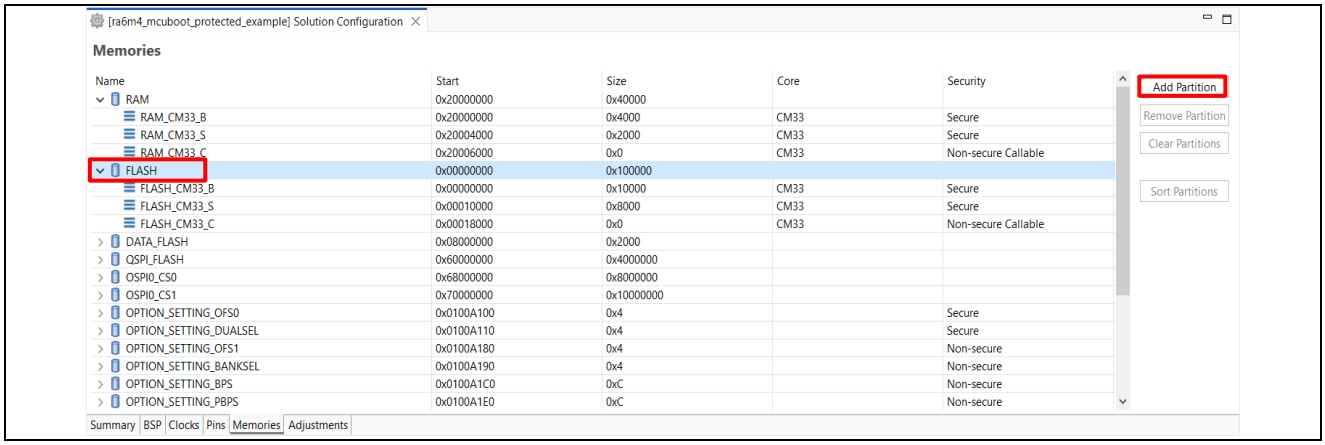


Figure 78. Add Partitions for Flash Memory

4. Add the memory layout for bootloader and application. In this example, the memory partitions are defined according to the memory map shown in Figure 79. About the symbol conventions for memory partitioning, user can refer to the guide at *FSP User's Manual* under *RA Flexible Software Package Documentation > API Reference > Modules > Bootloader > MCUboot Port (rm_mcuboot_port).*

Once all partitions have been defined, choose Flash > Sort Partitions to automatically arrange them in order based on their addresses.

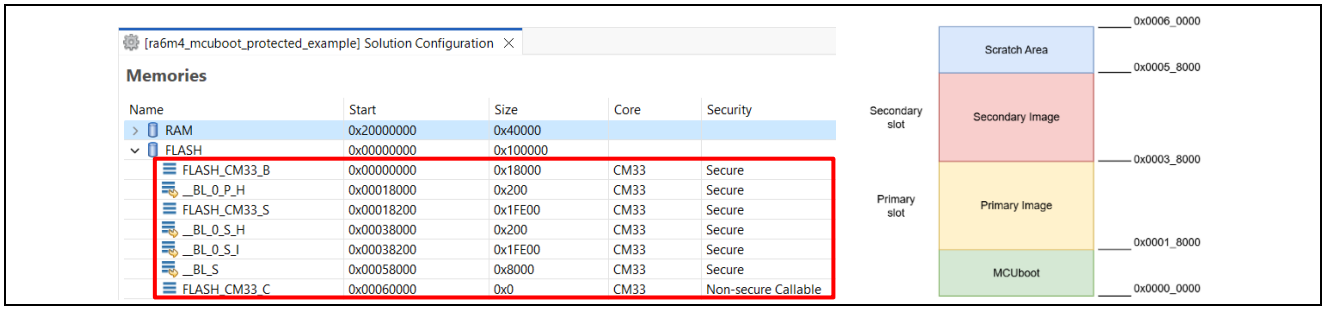


Figure 79. Define Flash Partitions for Bootloader and Application Images

5. After configuring the memory layout in the Solution Project, right click on the FSP Solution Project → **Build Project.** Both the bootloader and the primary application project will be rebuilt using the updated memory partition settings.

Note: The secondary application project must be compiled separately after the Solution Project has been fully built to apply the updated memory map settings.

6.4 Configure the Debug Configuration

This section provides instructions for configuring debugging in e² studio after injecting the wrapped public key and programming the bootloader image, as described in Section 4.2.3.

1. Open the **Debug Configurations**: **app_ra6m4_primary > Debug As > Debug Configurations** Make sure that **app_ra6m4_primary Debug_Flat** is selected and select the **Startup** tab.

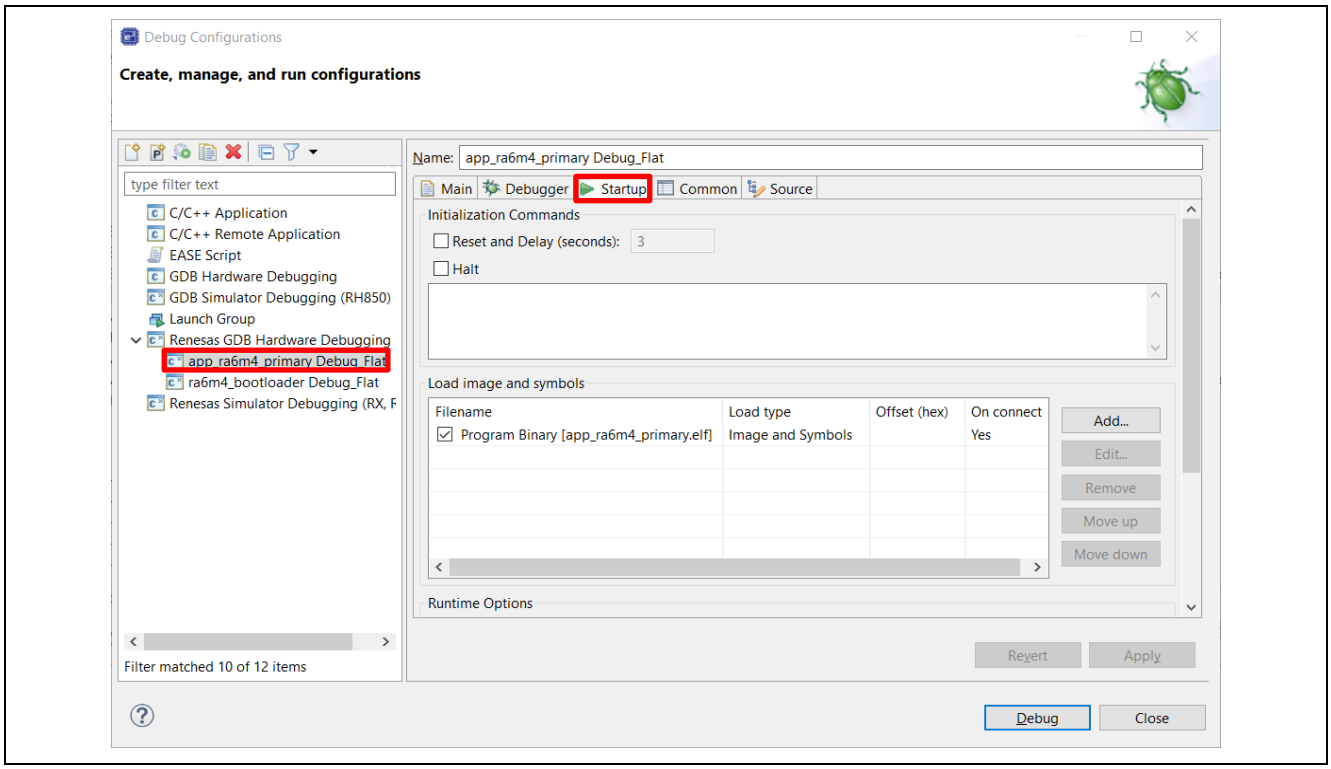


Figure 80. Configure the Primary Secure Project Debug Startup

2. Set up the **Debug Configurations**.
   Click **Add…** and then **Workspace**. Navigate to the **ra6m4_bootloader** project and select the **ra6m4_bootloader.elf** file from the debug folder. Click **OK**.

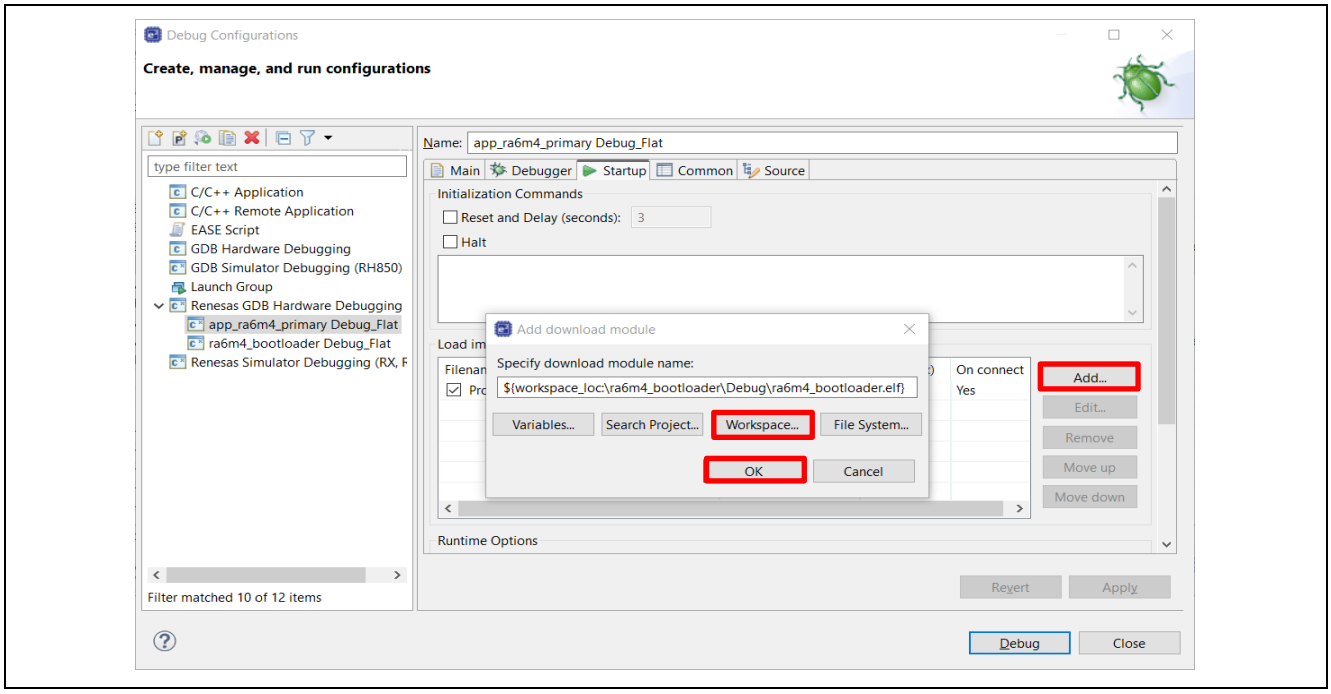


Figure 81. Add the Bootloader Project

Change the **Load type** of the **ra6m4_bootloader** and **app_ra6m4_primary** to **Symbols only** by clicking on the cell for Load type and selecting **Symbols only** from the drop-down menu.

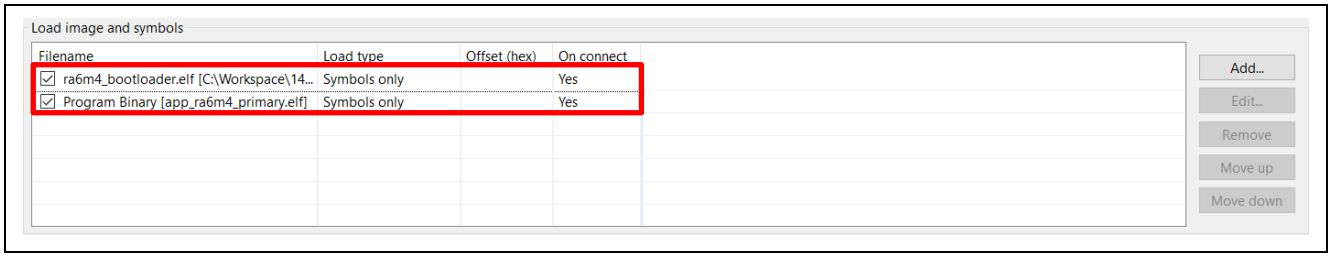


Figure 82. Select to load Symbols only for the Bootloader and Primary Application Project

3. Add the signed binary image to the download options using Raw Binary Load type.

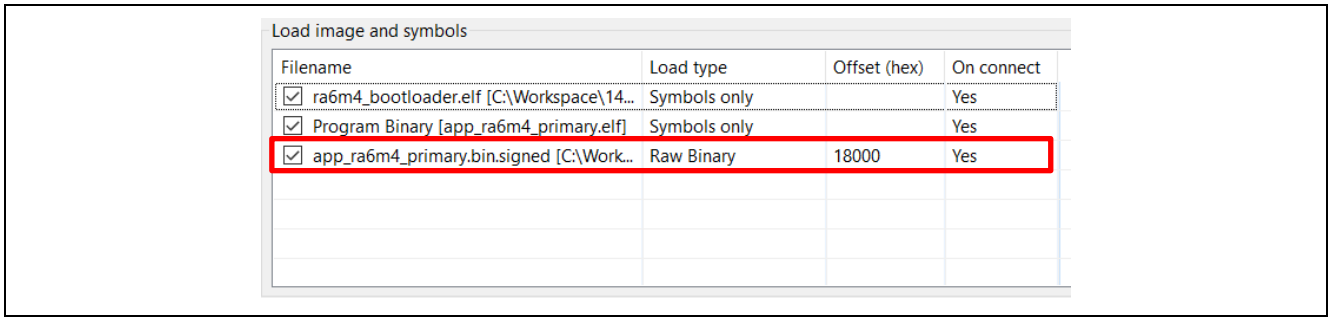


Figure 83. Load the Signed Primary Image

Note that for different update modes and different application images, the load address needs to be updated.

4. After the above is set up, follow section 4.2 to run the projects with Swap Update mode.

7. Production Support Considerations

Mastering and delivering a new application involves similar steps to those described in Sections 5 and 6. Typically, the following aspects must be considered when designing the process for delivering new applications:

1. Create and sign the new applications.
   Refer to Section 6 for instructions on how to configure the new application to work with the bootloader and how to sign it.
2. Download the new application to the Secondary slots.
   This step may vary depending on the download method chosen by the user. In this example project, the *Ancillary File Download* feature in e² studio is used for demonstration purposes. This method can serve as a useful testing tool during the development of a custom image downloader. Alternatively, you can use the **Renesas Flash Programmer** tool to program the bootloader, application and public key in a single operation. For reference, application projects **R01AN7766** and **R11AN0567** provide examples of image downloaders using XMODEM protocol and the USB PCDC interface.

8. References

- Renesas RA Family Secure Bootloader for RA2 MCU Series (R11AN0516)
- Renesas RA Family Booting Encrypted Image using MCUboot and QSPI (R11AN0567)
- Renesas RA Family RA2 MCU Advanced Secure Bootloader Design using MCUboot Internal Code Flash and Memory Mirror Function (R01AN7766)
- Renesas RA Family RA6 MCU Advanced Secure Bootloader Design using MCUboot and Code Flash Dualbank Mode (R11AN0570)
- Renesas RA Family Injecting and Updating Secure User Keys (R11AN0496)

9. Website and Support

Visit the following URLs to learn about the RA family of microcontrollers, download tools and documentation, and get support.

EK-RA6M4 Resources	renesas.com/ra/ek-ra6m4
RA Product Information	renesas.com/ra
Flexible Software Package (FSP)	renesas.com/ra/fsp
RA Product Support Forum	renesas.com/ra/forum
Renesas Support	renesas.com/support

Revision History

Rev.	Date	Description	
		Page	Summary
1.0.0	Sep.29.25	-	First release document.

General Precautions in the Handling of Microprocessing Unit and Microcontroller Unit Products

The following usage notes are applicable to all Microprocessing unit and Microcontroller unit products from Renesas. For detailed usage notes on the products covered by this document, refer to the relevant sections of the document as well as any technical updates that have been issued for the products.

1. Precaution against Electrostatic Discharge (ESD)

   A strong electrical field, when exposed to a CMOS device, can cause destruction of the gate oxide and ultimately degrade the device operation. Steps must be taken to stop the generation of static electricity as much as possible, and quickly dissipate it when it occurs. Environmental control must be adequate. When it is dry, a humidifier should be used. This is recommended to avoid using insulators that can easily build up static electricity. Semiconductor devices must be stored and transported in an anti-static container, static shielding bag or conductive material. All test and measurement tools including work benches and floors must be grounded. The operator must also be grounded using a wrist strap. Semiconductor devices must not be touched with bare hands. Similar precautions must be taken for printed circuit boards with mounted semiconductor devices.
2. Processing at power-on

   The state of the product is undefined at the time when power is supplied. The states of internal circuits in the LSI are indeterminate and the states of register settings and pins are undefined at the time when power is supplied. In a finished product where the reset signal is applied to the external reset pin, the states of pins are not guaranteed from the time when power is supplied until the reset process is completed. In a similar way, the states of pins in a product that is reset by an on-chip power-on reset function are not guaranteed from the time when power is supplied until the power reaches the level at which resetting is specified.
3. Input of signal during power-off state

   Do not input signals or an I/O pull-up power supply while the device is powered off. The current injection that results from input of such a signal or I/O pull-up power supply may cause malfunction and the abnormal current that passes in the device at this time may cause degradation of internal elements. Follow the guideline for input signal during power-off state as described in your product documentation.
4. Handling of unused pins

   Handle unused pins in accordance with the directions given under handling of unused pins in the manual. The input pins of CMOS products are generally in the high-impedance state. In operation with an unused pin in the open-circuit state, extra electromagnetic noise is induced in the vicinity of the LSI, an associated shoot-through current flows internally, and malfunctions occur due to the false recognition of the pin state as an input signal become possible.
5. Clock signals

   After applying a reset, only release the reset line after the operating clock signal becomes stable. When switching the clock signal during program execution, wait until the target clock signal is stabilized. When the clock signal is generated with an external resonator or from an external oscillator during a reset, ensure that the reset line is only released after full stabilization of the clock signal. Additionally, when switching to a clock signal produced with an external resonator or by an external oscillator while program execution is in progress, wait until the target clock signal is stable.
6. Voltage application waveform at input pin

   Waveform distortion due to input noise or a reflected wave may cause malfunction. If the input of the CMOS device stays in the area between V_{IL} (Max.) and V_{IH} (Min.) due to noise, for example, the device may malfunction. Take care to prevent chattering noise from entering the device when the input level is fixed, and also in the transition period when the input level passes through the area between V_{IL} (Max.) and V_{IH} (Min.).
7. Prohibition of access to reserved addresses

   Access to reserved addresses is prohibited. The reserved addresses are provided for possible future expansion of functions. Do not access these addresses as the correct operation of the LSI is not guaranteed.
8. Differences between products

   Before changing from one product to another, for example to a product with a different part number, confirm that the change will not lead to problems. The characteristics of a microprocessing unit or microcontroller unit products in the same group but having a different part number might differ in terms of internal memory capacity, layout pattern, and other factors, which can affect the ranges of electrical characteristics, such as characteristic values, operating margins, immunity to noise, and amount of radiated noise. When changing to a product with a different part number, implement a system-evaluation test for the given product.

Notice

1. Descriptions of circuits, software and other related information in this document are provided only to illustrate the operation of semiconductor products and application examples. You are fully responsible for the incorporation or any other use of the circuits, software, and information in the design of your product or system. Renesas Electronics disclaims any and all liability for any losses and damages incurred by you or third parties arising from the use of these circuits, software, or information.
2. Renesas Electronics hereby expressly disclaims any warranties against and liability for infringement or any other claims involving patents, copyrights, or other intellectual property rights of third parties, by or arising from the use of Renesas Electronics products or technical information described in this document, including but not limited to, the product data, drawings, charts, programs, algorithms, and application examples.
3. No license, express, implied or otherwise, is granted hereby under any patents, copyrights or other intellectual property rights of Renesas Electronics or others.
4. You shall be responsible for determining what licenses are required from any third parties, and obtaining such licenses for the lawful import, export, manufacture, sales, utilization, distribution or other disposal of any products incorporating Renesas Electronics products, if required.
5. You shall not alter, modify, copy, or reverse engineer any Renesas Electronics product, whether in whole or in part. Renesas Electronics disclaims any and all liability for any losses or damages incurred by you or third parties arising from such alteration, modification, copying or reverse engineering.
6. Renesas Electronics products are classified according to the following two quality grades: “Standard” and “High Quality”. The intended applications for each Renesas Electronics product depends on the product’s quality grade, as indicated below.
   "Standard": Computers; office equipment; communications equipment; test and measurement equipment; audio and visual equipment; home electronic appliances; machine tools; personal electronic equipment; industrial robots; etc.
   "High Quality": Transportation equipment (automobiles, trains, ships, etc.); traffic control (traffic lights); large-scale communication equipment; key financial terminal systems; safety control equipment; etc.

   Unless expressly designated as a high reliability product or a product for harsh environments in a Renesas Electronics data sheet or other Renesas Electronics document, Renesas Electronics products are not intended or authorized for use in products or systems that may pose a direct threat to human life or bodily injury (artificial life support devices or systems; surgical implantations; etc.), or may cause serious property damage (space system; undersea repeaters; nuclear power control systems; aircraft control systems; key plant systems; military equipment; etc.). Renesas Electronics disclaims any and all liability for any damages or losses incurred by you or any third parties arising from the use of any Renesas Electronics product that is inconsistent with any Renesas Electronics data sheet, user’s manual or other Renesas Electronics document.
7. No semiconductor product is absolutely secure. Notwithstanding any security measures or features that may be implemented in Renesas Electronics hardware or software products, Renesas Electronics shall have absolutely no liability arising out of any vulnerability or security breach, including but not limited to any unauthorized access to or use of a Renesas Electronics product or a system that uses a Renesas Electronics product. RENESAS ELECTRONICS DOES NOT WARRANT OR GUARANTEE THAT RENESAS ELECTRONICS PRODUCTS, OR ANY SYSTEMS CREATED USING RENESAS ELECTRONICS PRODUCTS WILL BE INVULNERABLE OR FREE FROM CORRUPTION, ATTACK, VIRUSES, INTERFERENCE, HACKING, DATA LOSS OR THEFT, OR OTHER SECURITY INTRUSION (“Vulnerability Issues”). RENESAS ELECTRONICS DISCLAIMS ANY AND ALL RESPONSIBILITY OR LIABILITY ARISING FROM OR RELATED TO ANY VULNERABILITY ISSUES. FURTHERMORE, TO THE EXTENT PERMITTED BY APPLICABLE LAW, RENESAS ELECTRONICS DISCLAIMS ANY AND ALL WARRANTIES, EXPRESS OR IMPLIED, WITH RESPECT TO THIS DOCUMENT AND ANY RELATED OR ACCOMPANYING SOFTWARE OR HARDWARE, INCLUDING BUT NOT LIMITED TO THE IMPLIED WARRANTIES OF MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE.
8. When using Renesas Electronics products, refer to the latest product information (data sheets, user’s manuals, application notes, “General Notes for Handling and Using Semiconductor Devices” in the reliability handbook, etc.), and ensure that usage conditions are within the ranges specified by Renesas Electronics with respect to maximum ratings, operating power supply voltage range, heat dissipation characteristics, installation, etc. Renesas Electronics disclaims any and all liability for any malfunctions, failure or accident arising out of the use of Renesas Electronics products outside of such specified ranges.
9. Although Renesas Electronics endeavors to improve the quality and reliability of Renesas Electronics products, semiconductor products have specific characteristics, such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Unless designated as a high reliability product or a product for harsh environments in a Renesas Electronics data sheet or other Renesas Electronics document, Renesas Electronics products are not subject to radiation resistance design. You are responsible for implementing safety measures to guard against the possibility of bodily injury, injury or damage caused by fire, and/or danger to the public in the event of a failure or malfunction of Renesas Electronics products, such as safety design for hardware and software, including but not limited to redundancy, fire control and malfunction prevention, appropriate treatment for aging degradation or any other appropriate measures. Because the evaluation of microcomputer software alone is very difficult and impractical, you are responsible for evaluating the safety of the final products or systems manufactured by you.
10. Please contact a Renesas Electronics sales office for details as to environmental matters such as the environmental compatibility of each Renesas Electronics product. You are responsible for carefully and sufficiently investigating applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive, and using Renesas Electronics products in compliance with all these applicable laws and regulations. Renesas Electronics disclaims any and all liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.
11. Renesas Electronics products and technologies shall not be used for or incorporated into any products or systems whose manufacture, use, or sale is prohibited under any applicable domestic or foreign laws or regulations. You shall comply with any applicable export control laws and regulations promulgated and administered by the governments of any countries asserting jurisdiction over the parties or transactions.
12. It is the responsibility of the buyer or distributor of Renesas Electronics products, or any other party who distributes, disposes of, or otherwise sells or transfers the product to a third party, to notify such third party in advance of the contents and conditions set forth in this document.
13. This document shall not be reprinted, reproduced or duplicated in any form, in whole or in part, without prior written consent of Renesas Electronics.
14. Please contact a Renesas Electronics sales office if you have any questions regarding the information contained in this document or Renesas Electronics products.

(Note1) “Renesas Electronics” as used in this document means Renesas Electronics Corporation and also includes its directly or indirectly controlled subsidiaries.

(Note2) “Renesas Electronics product(s)” means any product developed or manufactured by or for Renesas Electronics.

(Rev.5.0-1 October 2020)

Corporate Headquarters

TOYOSU FORESIA, 3-2-24 Toyosu,
Koto-ku, Tokyo 135-0061, Japan
www.renesas.com

Trademarks

Renesas and the Renesas logo are trademarks of Renesas Electronics Corporation. All trademarks and registered trademarks are the property of their respective owners.

Contact information

For further information on a product, technology, the most up-to-date version of a document, or your nearest sales office, please visit:
www.renesas.com/contact/.