Software Manual

# Security Key Management Tool

# Software Manual

All information contained in these materials, including products and product specifications, represents information on the product at the time of publication and is subject to change by Renesas Electronics Corp. without notice. Please review the latest information published by Renesas Electronics Corp. through various means, including the Renesas Electronics Corp. website (http://www.renesas.com).

Renesas Electronics
www.renesas.com

Rev.1.12 Sep 2026

# 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/.

© 2026 Renesas Electronics Corporation. All rights reserved.

# 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.

# How to Use This Manual

## 1. Purpose and Target Readers

This manual is intended to help users understand the features of the Security Key Management Tool for CLI. It is intended for users who design and develop systems that use the Security Cryptographic Engine, Trusted Secure IP and RSIP-Exxx, which are security IPs built into Renesas Electronics microcomputers. To use this manual, you need a basic knowledge of microcontrollers and Windows , Linux and macOS.

| |
|---|
| Use this software after sufficiently confirming the manual of the microcontroller in use. |

## 2. Conventions

Note: Footnote for item marked with "Note" in the text.
Numeral representations: Binary ... xxxx or xxxxB
Decimal ... xxxx
Hexadecimal ... 0xXXXX or xxxxH
“ “: Any character or item on the screen that can be selected or input
[ ]: Name of a command, dialog box, tabbed page, option, or area on the screen

## 3. Terminology

| Term | Meaning |
|---|---|
| CLI | Command Line Interface |
| DLM Key | Key for Device Lifecycle Management. Not supported for all MCUs/MPUs. |
| DOTF | A function that directly decrypts an encrypted image written to an external flash ROM or other device when reading and executing the image.<br>Some MCUs/MPUs call it Decryption On-The-Fly (DOTF) or On-The-Fly Decryption (OTFD). In this document, it is referred to as DOTF. |
| Encrypted KUK | Key Update Key wrapped by a UFPK |
| Encrypted User Key | User Key wrapped by either a UFPK or a KUK |
| FSBL | First Stage Bootloader<br>Programs to verify the validity of Second Stage Bootloader or OEM_BL. |
| GUI | Graphic User Interface |
| HRK | Hardware Root Key<br>Key data set for each MCU family or MCU group. |
| HUK | Hardware Unique Key<br>Unique key data set for each MCU device. |
| KUK | Key Update Key<br>Key to update User Key |
| OEM Bootloader<br>OEM_BL | Programs that are executed after device startup to verify the validity of the application program |
| PEM | Base64-encoded file of the x509 ASN.1 key.<br>The tool supports reading keys generated by the Openssl command genrsa or the ecparam -genkey command. |
| Provisioning SW | Software used with RA for Wireless products. This software is designed to inject keys, perform DLM transitions, and provision application firmware. |
| RA for Wireless | Products in the RA family designed for Bluetooth and Wi-Fi.<br>such as RA6B1 |
| Renesas key file | Renesas key file<br>Used for key provisioning with the Renesas Flash Programmer (RFP). |
| Renesas Key Wrap service | A service that wraps a UFPK with a device-specific key to generate a W-UFPK<br>https://dlm.renesas.com/keywrap/ |
| RFP | Renesas Flash Programmer<br>https://www.renesas.com/rfp |
| RSIP | Renesas Secure IP<br>Security IP implemented in RA Family, RX Family and RZ Family, etc. |
| RSU | Renesas Secure Update<br>An image file format for program updates defined in Firmware Update Module Using Firmware Integration Technology (R01AN5824). |
| SCE | Secure Cryptographic Engine<br>Security IP implemented in RA Family and Synergy Platform. |
| SFP | Secure Factory Programming<br>Ability to write encrypted images to devices while decrypting them, which is a function of Renesas MCU boot firmware |
| TSIP | Trusted Secure IP<br>Security IP implemented in RX Family |

| Term | Meaning |
| --- | --- |
| UFPK | User Factory Programming Key<br>Key used to wrap User Keys and DLM Keys |
| User Key | Encryption key used in the user application<br>AES key, RSA public key, RSA private key, etc. |
| W-UFPK | UFPK wrapped by the Renesas Key Wrapping service |
| Wrapped User Key | User Key rewrapped with SCE or TSIP to use User Key in SCE Protected Mode or TSIP |

# Table of Contents


1. Renesas Key Management System ..... 10
1.1 Introduction to Root of Trust ..... 10
1.2 Introduction to Renesas Secure IP and Associated Keys ..... 10
1.3 Renesas Secure Key Injection Advantages ..... 13
1.3.1 Advantages of Key Wrapping over Key Encryption ..... 13
1.3.2 Advantages of Key Wrapping using an HUK ..... 14
1.4 Wrapped Key Injection Procedure Overview ..... 14
1.4.1 General Steps for Secure Key Injection ..... 14
1.4.2 General Steps for Secure Key Update ..... 17
1.5 Renesas Security Functions ..... 20
1.5.1 First Stage Bootloader ..... 20
1.5.2 Decryption On-The-Fly/On-The-Fly Decryption ..... 22
1.5.3 Secure Factory Programming ..... 23

2. Overview ..... 24
2.1 Features ..... 24
2.1.1 Secure Key Injection and Update ..... 24
2.1.2 Security features supported by Security Key Management Tool ..... 27
2.2 Operating Environment ..... 29
2.2.1 Hardware Environment ..... 29
2.2.2 Software Environment ..... 29

3. Descriptions of GUI Functions ..... 30
3.1 Main Window ..... 30
3.2 Menu bar ..... 32
3.2.1 [File] menu ..... 32
3.2.2 [View] menu ..... 32
3.2.3 [Help] menu ..... 32
3.2.4 Status area ..... 33
3.2.4.1 Right-clicking menu ..... 33
3.3 [Overview] Tab ..... 34
3.4 [Generate UFPK] Tab ..... 35
3.5 [Generate KUK] Tab ..... 36
3.6 [Wrap Key] Tab ..... 37
3.6.1 [Key Type] Tab ..... 38
3.6.2 [Key Data] Tab ..... 41
3.6.2.1 When DLM / KUK / AES / TDES / ARC4 / ChaCha20-Poly1305 / ECC Private Key Selected in [Key Type] Tab ..... 41

3.6.2.2 When RSA Public Key Selected in [Key Type] Tab ....43
3.6.2.3 When RSA Private Key Selected in [Key Type] Tab ....44
3.6.2.4 When ECC Public Key Selected in [Key Type] Tab ....45
3.6.2.5 When OEM Root public Selected in [Key Type] Tab ....46
3.6.3 Wrapping Key ....47
3.6.4 IV ....48
3.6.5 Output ....49
3.7 [TSIP Update] Tab ....51
3.7.1 Output image ....52
3.7.2 Firmware image and Secure boot image ....52
3.7.3 [RSU Header] Tab ....53
3.7.4 [Encryption Address Range] Tab ....54
3.7.5 [Image Encryption Key] Tab ....55
3.7.6 [IV] Tab ....56
3.7.7 Output ....56
3.8 [FSBL] tab ....57
3.8.1 Programming Verification Method ....59
3.8.2 [Certificate] tab ....60
3.8.3 [OEM Root Keys] tab ....61
3.8.4 [OEM Bootloader Keys] tab ....62
3.9 [DOTF/OTFD] tab ....64
3.9.1 Image Address Range to Encrypt ....65
3.9.2 Destination Address ....66
3.9.3 Image Encryption Key ....67
3.9.4 IV ....67
3.9.5 Output Image Address and Contents ....68
3.10 [SFP] tab ....69
3.10.1 [Firmware Images] tab ....72
3.10.2 [Image Encryption Key] tab ....73
3.10.3 [Nonces] tab ....74
3.10.4 [AL2/SECDBG_KEY] tab ....75
3.10.5 [AL1/NONSECDBG_KEY] tab ....76
3.10.6 [Boundary] tab ....77
3.10.7 [External Flash Memory Area] tab ....78
3.11 [RA6B1/RA6U1] tab ....79
3.11.1 [Certificates] tab ....81
3.11.2 [OEM Verification Root Keys] tab ....82
3.11.3 [OEM Provisioning Keys] tab ....83
3.11.4 [OEM Bootloader Keys] tab ....85

4. Descriptions of CLI Functions ..... 87
4.1 Command-line Syntax ..... 87
4.2 Commands ..... 88
4.3 **genufpk** Command Options ..... 89
4.4 **genkuk** Command Options ..... 90
4.5 **genkey** Command Options ..... 91
4.5.1 **mcu** Option ..... 93
4.5.2 **keytype** Option ..... 94
4.5.2.1 AES-256-DOTF Option ..... 96
4.5.3 **key** Option ..... 97
4.5.3.1 Hex Data Direct Input ..... 97
4.5.3.2 File Input ..... 102
4.5.3.3 **key** Option Omitted ..... 104
4.5.4 **filetype** Option ..... 105
4.5.4.1 **rfp** Option for **filetype** ..... 105
4.5.4.2 **csource** Option for **filetype** ..... 105
4.5.4.3 **bin** Option for **filetype** ..... 107
4.5.4.4 **mot** Option for **filetype** ..... 108
4.5.5 **fileadd** Option ..... 111
4.5.6 **bswap** Option ..... 111
4.5.7 **keyfileoutput** Option ..... 112
4.6 **enctsip** Command Options ..... 113
4.6.1 **ver** Option ..... 115
4.6.1.1 **ver** option 1 ..... 115
4.6.1.2 **ver** option 2 ..... 117
4.6.2 **mode** Option ..... 119
4.6.3 **imgflg** Option ..... 121
4.6.4 **filetype** Option ..... 121
4.6.5 **df_ena** Option ..... 121
4.7 **gencert** command options ..... 122
4.7.1 **mode** Option ..... 124
4.7.2 **oembl_endian** option ..... 124
4.7.3 Area subject to OEM Bootloader signature or CRC calculation ..... 125
4.7.3.1 **oembl_size** option is specified ..... 125
4.7.3.2 only **cfsize** option is specified ..... 126
4.8 **encdotf** command options ..... 127
4.8.1 **keytype** option ..... 129
4.8.2 **enckey** option ..... 129
4.8.3 **dotftype** option ..... 129
4.8.4 **iv_tlv_output** option ..... 130
4.9 **encsfp** command options ..... 131
4.9.1 **mcu** option ..... 135

4.9.2 **trn** option ..... 135
4.9.3 **nonce_trn, nonce_prg, nonce_key** options ..... 136
4.9.4 **prg** option ..... 136
4.9.5 **boundary** option ..... 136
4.9.5.1 Argument input ..... 137
4.9.5.2 File input ..... 137
4.9.6 **extarea0/extarea1** options ..... 138
4.9.7 **num_al2key**/**num_al1key** options ..... 139
4.10 **genhdr** options ..... 140
4.10.1 **mode** option ..... 144
4.10.2 **header** option ..... 144
4.10.3 **key_sign / code_sign** options ..... 144
4.10.4 **prg** Option ..... 144
4.10.5 Regarding the Subject of Code Certificate ..... 145
4.11 **calcresponse** Options ..... 147

5. Operating Procedure ..... 148
5.1 Standalone ..... 148
5.1.1 Windows ..... 148
5.1.1.1 GUI version ..... 148
5.1.1.2 CLI version ..... 149
5.1.2 Linux version ..... 150
5.1.2.1 GUI version ..... 150
5.1.2.2 CLI version ..... 151
5.1.3 macOS version ..... 152
5.1.3.1 GUI version ..... 152
5.1.3.2 CLI version ..... 153
5.2 e$^2$studio plugin ..... 154
5.2.1 Installing e$^2$studio plugin version ..... 154
5.2.2 Uninstalling e$^2$studio plugin version ..... 159

6. Usage Examples ..... 161
6.1 Example 1 – Inject an AES128 Key on an RX Family MCU with TSIP ..... 161
6.1.1 Using the GUI version ..... 161
6.1.2 Using the CLI version ..... 165
6.2 Example 2 – Inject a Key-Update Key on an RA Family MCU with SCE9 Security Functions and Protected Mode ..... 166
6.2.1 Using the GUI version ..... 166
6.2.2 Using the CLI version ..... 171
6.3 Example 3 – Update an RSA 2048 Public Key on an RA Family MCU with SCE9 Protected Mode ..... 173
6.3.1 Using the GUI version ..... 173

6.3.2 Using the CLI version ..... 176
6.4 Example 4 – Use RX Family TSIP Secure Update ..... 177
6.4.1 Using the GUI Version ..... 177
6.4.2 Using the CLI Version ..... 180
6.5 Example 5 – RA Family Generate FSBL Key Certificate / Code Certificate ..... 181
6.5.1 Using the GUI Version ..... 181
6.5.2 Using the CLI version ..... 185
6.6 Example 6 – Using DOTF with RA Family ..... 186
6.6.1 Using the GUI version ..... 186
6.6.2 Using the CLI version ..... 189
6.7 Example 7 – RA Family using Secure Factory Programming ..... 190
6.7.1 Using the GUI version ..... 191
6.7.2 Using the CLI version ..... 196
6.8 Example 8 – Generate RA for Wireless Header ..... 197
6.8.1 Using the GUI version ..... 197
6.8.2 Using the CLI version ..... 201

7. Notes ..... 202
7.1 Display settings when using Windows environment ..... 202
7.2 Note on using e²studio plugin version in a Linux environment ..... 203
7.3 Note on using e²studio plugin version in a macOS environment ..... 203
7.4 Note The macOS version limitation ..... 203
7.5 Notes of Secure Factory Programming Function ..... 205
7.5.1 Image Size Limitation for Secure Factory Programming Function ..... 205
7.5.1.1 How to create User data and write address/size in Security Key Management Tool ..... 205
7.6 Standalone and Plugin versions Notes on changing MCU/MPU and Crypto Engine in the Overview tab ..... 206

8. Appendix ..... 208
8.1 License ..... 208
8.2 Renesas Key File Format ..... 210
8.2.1 Text format ..... 210
8.2.2 File Structure ..... 210
8.2.3 File Extension ..... 210
8.2.4 Key Data ..... 211
8.2.4.1 Structure ..... 211
8.2.5 Encrypted Key Wrapping Formula ..... 212
8.3 Secure Factory Programming File Format ..... 213
8.3.1 File Format ..... 213
8.3.1.1 Pre-Data Field ..... 215
8.3.1.2 TLV Length Field ..... 219

8.3.1.3 TLV Field ....219
8.3.2 File extensions ....220
8.4 Key Certificate / Code Certificate ....221
8.4.1 Key Certificate File Format ....221
8.4.2 Code Certificate file format generated by **mode "signature"** ....222
8.4.3 Code Certificate file format generated by **mode "crc"** ....223
8.4.4 CRC32 calculation ....223
8.5 TSIP Update ....224
8.5.1 User program encryption formula ....224
8.6 RA6B1/RA6U1 Header format ....225
8.6.1 Product Header ....225
8.6.2 Image Header ....226
8.6.3 Key Certificate ....226
8.6.4 Code Certificate ....227

# List of Figures


Figure 1-1 Outline of a Typical Renesas Security Engine ..... 10
Figure 1-2 Outline of RX Trusted Secure IP ..... 11
Figure 1-3 Key Wrapping vs. Key Encryption ..... 13
Figure 1-4 Key Wrapping using an HUK ..... 14
Figure 1-5 Wrapping the UFPK using DLM server ..... 15
Figure 1-6 Encrypt the User Key with UFPK ..... 15
Figure 1-7 Inject a User Key over the Serial Programming Interface ..... 16
Figure 1-8 Encrypt a KUK with the UFPK ..... 17
Figure 1-9 Inject a KUK Over the Serial Programming Interface ..... 18
Figure 1-10 Encrypt the New User Key with a KUK ..... 18
Figure 1-11 Update the User Key ..... 19
Figure 1-12 Chain of Trust in a secure system. ..... 20
Figure 1-13 Authenticity Check flow performed when programming OEM_BL at the factory. ..... 21
Figure 1-14 FSBL operation at device startup. ..... 22
Figure 1-15 Example of internal system bus for DOTF implemented MCU/MPU. ..... 22
Figure 1-16 System example of MCU/MPU with Secure Factory Programming function implemented. ..... 23
Figure 3-1 Main Window of standalone version ..... 30
Figure 3-2 Main Window of e$^2$studio plugin version ..... 31
Figure 3-3 [Overview] Tab ..... 34
Figure 3-4 [Generate UFPK] Tab ..... 35
Figure 3-5 [Generate KUK] Tab ..... 36
Figure 3-6 [Wrap Key] Tab ..... 37
Figure 3-7 [Key Type] Tab ..... 38
Figure 3-8 [Key Data] Tab when DLM / KUK / AES / TDES / ARC4 / ChaCha20-Poly1305 / ECC private key is selected ..... 41
Figure 3-9 [Key Data] Tab when RSA public key is selected ..... 43
Figure 3-10 [Key Data] Tab when RSA private key is selected ..... 44
Figure 3-11 [Key Data] Tab when ECC public key is selected ..... 45
Figure 3-12 [Key Data] Tab when OEM Root public key is selected ..... 46
Figure 3-13 Wrapping Key Options ..... 47
Figure 3-14 IV Options ..... 48
Figure 3-15 Output Options ..... 49
Figure 3-16 [TSIP Update] Tab ..... 51
Figure 3-17 Output image ..... 52
Figure 3-18 Firmware image and Secure boot image ..... 52
Figure 3-19 [RSU Header] Tab ..... 53
Figure 3-20 [Encrypted Address Range] Tab ..... 54
Figure 3-21 [Image Encryption Key] Tab ..... 55
Figure 3-22 [IV] Tab ..... 56
Figure 3-23 Output ..... 56
Figure 3-24 [FSBL] Tab ..... 57
Figure 3-25 Programming Verification Method Options ..... 59
Figure 3-26 [Certificate] Tab ..... 60
Figure 3-27 [OEM Root Keys] Tab ..... 61
Figure 3-28 [OEM Bootloader Keys] Tab ..... 62
Figure 3-29 [DOTF/OTFD] Tab ..... 64
Figure 3-30 Image Address Range to Encrypt Options ..... 65
Figure 3-31 Destination Address Options ..... 66

Figure 3-32 Image Encryption Key Options ..... 67
Figure 3-33 IV Options ..... 67
Figure 3-34 Output Image Address and Contents Options ..... 68
Figure 3-35 [SFP] Tab ..... 69
Figure 3-36 [Firmware Images] Tab ..... 72
Figure 3-37 [Image Encryption Key] Tab ..... 73
Figure 3-38 [Nonces] Tab ..... 74
Figure 3-39 [AL2/SECDBG_KEY] Tab ..... 75
Figure 3-40 [AL1/NONSECDBG_KEY] Tab ..... 76
Figure 3-41 [Boundary] Tab ..... 77
Figure 3-42 [External Flash Memory Area] Tab ..... 78
Figure 3-43 [RA6B1/RA6U1] Tab ..... 79
Figure 3-44 [Certificates] Tab ..... 81
Figure 3-45 [OEM Verification Root Key] Tab ..... 82
Figure 3-46 [OEM Provisioning Keys] Tab ..... 83
Figure 3-47 [OEM Bootloader Keys] Tab ..... 85
Figure 4-1 Example of manual creation of AES 128-bit key file ..... 103
Figure 4-2 Example of manual creation of RSA 2048 public key file ..... 104
Figure 4-3 Encryption flow when ver option 1 is specified ..... 115
Figure 4-4 Encryption flow when ver option 2 is specified ..... 117
Figure 4-5 File Image Generated when factory Is Specified for the mode Option ..... 119
Figure 4-6 File Image Generated when update Is Specified for the mode Option and rsu for the filetype Option ..... 120
Figure 4-7 File Image Generated when update Is Specified for the mode Option and mot for the filetype Option ..... 120
Figure 4-8 Signature and CRC calculation target when oembl_size option is specified ..... 125
Figure 4-9 Signature and CRC calculation target when only cfsize option is specified ..... 126
Figure 4-10 Signature Scope and Output Images When the size Option Is Omitted When Specifying a Motorola Hex File for the Input Firmware Image ..... 145
Figure 4-11 Signature Scope and Output Images When the size Option Is Specified for a Motorola Hex File in the Input Firmware Image ..... 146
Figure 5-1 Security Key Management Tool – GUI Dialog at startup from Windows ..... 148
Figure 5-2 Security Key Management Tool - CLI execution example from command prompt ..... 149
Figure 5-3 Security Key Management Tool – GUI Dialog at startup from Linux ..... 150
Figure 5-4 Security Key Management Tool - CLI execution example from Linux Terminal software ..... 151
Figure 5-5 Security Key Management Tool – GUI Dialog at startup from macOS ..... 152
Figure 5-6 Security Key Management Tool - CLI execution example from macOS Terminal software ..... 153
Figure 5-7 e$^2$studio “Help(H)” – “Install New Software...” ..... 154
Figure 5-8 “Install” Dialog ..... 155
Figure 5-9 “Add Repository” Dialog ..... 155
Figure 5-10 “Install” Dialog – Select “Security Key Management Tool” ..... 156
Figure 5-11 “Install” Dialog – Install Details ..... 156
Figure 5-12 “Install” Dialog – Review License ..... 157
Figure 5-13 “Trust” Dialog ..... 157
Figure 5-14 Project “Properties” Dialog ..... 158
Figure 5-15 “About e$^2$studio” Dialog ..... 159
Figure 5-16 "e$^2$ studio Installation Details" Dialog ..... 159
Figure 5-17 "Uninstall" Dialog ..... 160
Figure 6-1 [Overview] Tab, Injecting an AES128 Key with RX Family TSIP ..... 161
Figure 6-2 [Generate UFPK] Tab, Example of UFPK generation using specified value ..... 162
Figure 6-3 Example of execution result, UFPK generation using specified value ..... 162
Figure 6-4 [Wrap Key] - [Key Type] Tab, Example of creating a AES128 key file as Motorola Hex ..... 163

Figure 6-5 [Wrap Key] - [Key Data] Tab, Example of creating an AES128 key file as Motorola Hex.....163
Figure 6-6 Example of execution result, creating an AES128 key file as Motorola Hex .....164
Figure 6-7 Execution result of CLI genufpk command .....165
Figure 6-8 CLI example of creating an AES128 key file as Motorola Hex .....165
Figure 6-9 [Overview] Tab, Injecting a KUK with RA Family SCE9 Security Functions and Protected Mode .....166
Figure 6-10 [Generate UFPK] tab Example of UFPK generation using specified value .....167
Figure 6-11 Example execution result, UFPK generation using specified value .....167
Figure 6-12 [Generate KUK] Tab, Example of creating a KUK key file .....168
Figure 6-13 Example execution result, creating a KUK file .....168
Figure 6-14 [Wrap Key] – [Key Type] Tab, Example of creating a KUK file as an RFP file .....169
Figure 6-15 [Wrap Key] – [Key Data] Tab, Example of creating a KUK file as an RFP file .....169
Figure 6-16 Example execution result, creating a KUK file as an RFP file .....170
Figure 6-17 Execution result of CLI genufpk command .....171
Figure 6-18 Execution result of CLI genkuk command .....171
Figure 6-19 CLI example of creating an AES128 key file as an RFP file .....172
Figure 6-20 [Overview] Tab, Updating an RSA 2048 Public Key, RA Family SCE9 Security Functions and Protected Mode .....173
Figure 6-21 [Wrap Key] - [Key Type] Tab, Example of creating an RSA 2048 Public key file as C source file .....174
Figure 6-22 [Wrap Key] – [Key Data]Tab, Example of creating an RSA 2048 Public key file as C source file .....174
Figure 6-23 Example execution result, creating an RSA 2048 Public key file as C source file .....175
Figure 6-24 CLI example of creating an RSA 2048 Public key file as C source file .....176
Figure 6-25 [Overview] Tab .....177
Figure 6-26 [TSIP Update] – [RSU Header] Tab: Example Settings .....178
Figure 6-27 [TSIP Update] – [Encrypted Address Range] Tab: Other Example Settings .....178
Figure 6-28 [TSIP Update] – [Image Encryption Key] Tab: Example Settings .....179
Figure 6-29 [TSIP Update] – [IV] Tab: Example Settings .....179
Figure 6-30 [TSIP Update] Tab: Execution Example .....179
Figure 6-31 enctsip Command Execution Example .....180
Figure 6-32 [Overview] Tab, Generate Key Certificate and Code Certificate with RA Family RSIP-E51A Security Functions and Protected Mode .....181
Figure 6-33 [FSBL] – [Certificates] Tab, Setting Example .....182
Figure 6-34 [FSBL] – [OEM Root Keys] Tab, Setting Example .....183
Figure 6-35 [FSBL] – [OEM Bootloader Keys] Tab, Setting Example .....183
Figure 6-36 Example of execution result, generate Key Certificate and Code Certificate .....184
Figure 6-37 Execution result of CLI gencert command .....185
Figure 6-38 [Overview] Tab, Decryption On-The-Fly Encryption with RA Family RSIP-E51A Security Functions and Protected Mode .....186
Figure 6-39 [DOTF/OTFD] tab - Example settings for Plaintext Image, Encryption Range, and Destination Address .....187
Figure 6-40 [DOTF/OTFD] tab - Image Encryption Key, IV Setting example .....187
Figure 6-41 [DOTF/OTFD] tab – Encrypted Image and Output Image Address and Contents .....188
Figure 6-42 [DOTF/OTFD] tab – Example of execution result .....188
Figure 6-43 Execution result of CLI encdotf command .....189
Figure 6-44 [Overview] Tab, Secure Factory Programming with RA Family RSIP-E51A Security Functions and Protected Mode .....191
Figure 6-45 [SFP] – [Firmware Images] Tab, Setting Example .....192
Figure 6-46 [SFP] – [Image Encryption Key] Tab, Setting Example .....193
Figure 6-47 [SFP] – [Nonces] Tab, Setting Example .....194
Figure 6-48 [SFP] – [AL2/SECDBG_KEY] Tab, Setting Example .....194

**Figure 6-49 [SFP] tab – Example of execution result....195**
**Figure 6-50 Execution result of CLI encsfp command ....196**
**Figure 6-51 [Overview] Tab, RA Family RSIP-E31B Security Functions and Protected Mode....197**
**Figure 6-52 [RA6B1/RA6U1] Tab, Setting Example of Firmware Image Configuration ....198**
**Figure 6-53 [RA6B1/RA6U1] Tab – [Certificates] Tab, Setting Example ....198**
**Figure 6-54 [RA6B1/RA6U1] Tab – [OEM Verification Root Keys] Tab, Setting Example ....199**
**Figure 6-55 [RA6B1/RA6U1] Tab – [OEM Bootloader Keys] Tab, Setting Example....199**
**Figure 6-56 [RA6B1/RA6U1] Tab – Specified the output filename ....200**
**Figure 6-57 [RA6B1/RA6U1] Tab – Example of execution result....200**
**Figure 6-58 Execution result of CLI genhdr command....201**
**Figure 7-1 SecurityKeyManagementTool.exe Properties “High DPI scaling override” settings....202**
**Figure 7-2 [Overview] Tab, Select RX Family, TSIP for MCU/MPU and Security engine....206**
**Figure 7-3 [Wrap Key] Tab, Select RSA 2048bits, public in [Key Type] tab ....206**
**Figure 7-4 [Overview] Tab, Select RX Family, TSIP Lite for MCU/MPU and Security engine ....206**
**Figure 7-5 [Wrap Key] Tab, Display AES 128bits in [Key Type] tab and the input display RSA public key in [Key Data] tab ....207**
**Figure 8-1 Secure Factory Programming File V1 Format....213**
**Figure 8-2 Secure Factory Programming File V2 Format....214**
**Figure 8-3 Secure Factory Programming File V2 Format : Header Field ....216**
**Figure 8-4 Secure Factory Programming File V2 Format : Parameter Field....217**
**Figure 8-5 Secure Factory Programming File V2 Format : DLM_AL Key Field....217**
**Figure 8-6 Secure Factory Programming File V2 Format : Nonce and MAC for encrypted user data Field ....218**
**Figure 8-7 TLV Format....219**

# 1. Renesas Key Management System

## 1.1 Introduction to Root of Trust

Roots of trust are highly reliable hardware, firmware, and software components that perform specific, critical security functions (https://csrc.nist.gov/projects/hardware-roots-of-trust). In an IoT system, a root of trust typically consists of identity and cryptographic keys rooted in the hardware of a device. It establishes a unique, immutable, and unclonable identity to authorize a device to exist in the IoT network.

Secure boot is part of the services provided in the Root of Trust in many security systems. Authentication of the application uses Public Key Encryption. The associated keys are part of the Root of Trust of the system. Device Identity, which consists of Device Private Key and Device Certificate, is part of the Root of Trust for many IoT devices.

From the above Root of Trust discussion, we can see that leakage of cryptographic keys can bring the secure system into a risky state. Protection of the Root of Trust involves limiting key accessibility to within the cryptographic boundary only, with keys that are securely stored and preferably unclonable. The Root of Trust should be locked from read and write access by unauthorized parties.

The Renesas user key management system can provide all of the above desired protection.

## 1.2 Introduction to Renesas Secure IP and Associated Keys

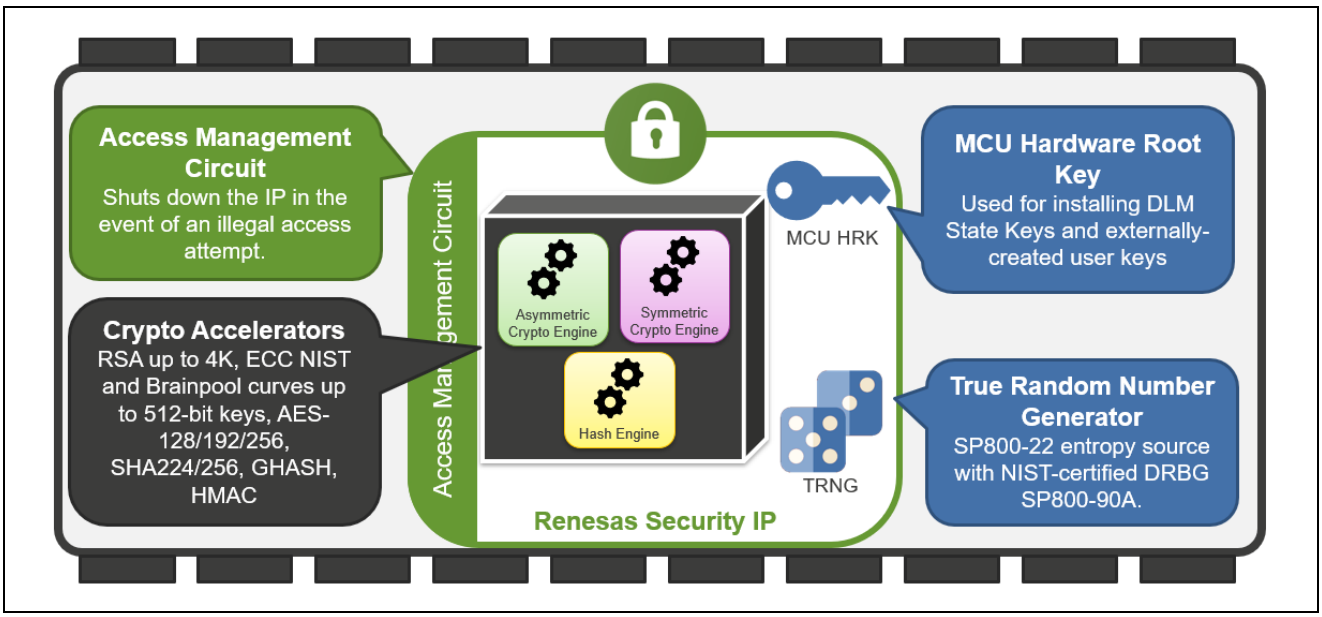


**Figure 1-1 Outline of a Typical Renesas Security Engine**

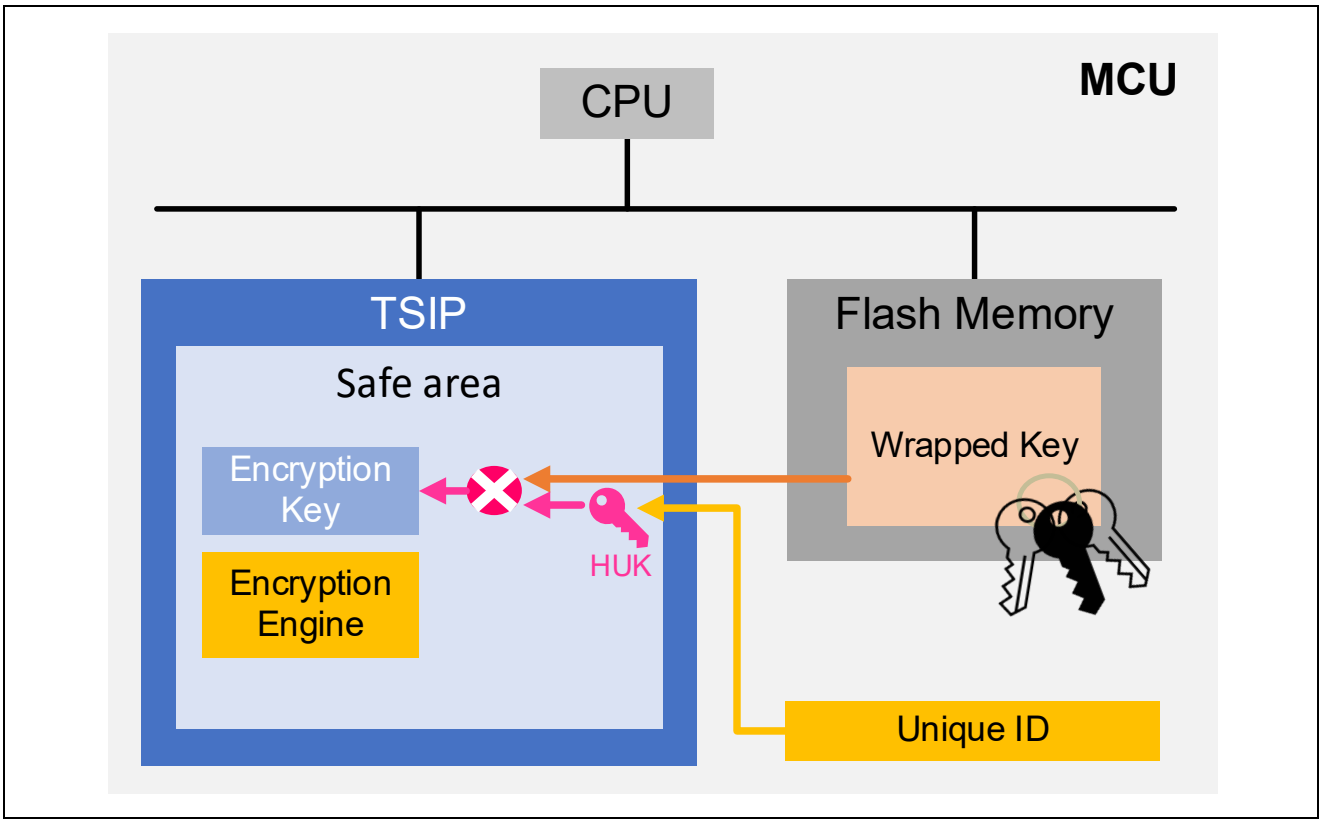


**Figure 1-2 Outline of RX Trusted Secure IP**

A Renesas Secure IP security engine, such as a Secure Crypto Engine (SCE), Trusted Secure IP (TSIP), or a RSIP-E Type security engine (RSIP), is an isolated subsystem within the MCU. The security engine contains hardware accelerators for both symmetric and asymmetric cryptographic algorithms, as well as various hashes and message authentication codes. It also contains a True Random Number Generator (TRNG), providing an entropy source for the cryptographic operations. The security engine is protected by an Access Management Circuit, which can shut down the security engine in the event of an illegal external access attempt.

Support for specific algorithms, secure key injection, and secure key update depends on the specific MCU/MPU. For more information, refer to the device's Hardware Manual plus the following web pages:

Table 1-1 MCU/MPU Related Information

| MCU/MPU | Category | URL |
|---|---|---|
| RA Family | MCU drivers | https://www.renesas.com/software-tool/flexible-software-package-fsp |
| | Application Projects | https://www.renesas.com/ra, please refer to the Documentation for the specific MCU Group |
| RX Family | MCU drivers and example project | https://www.renesas.com/software-tool/trusted-secure-ip-driver |
| RZ/T2M, RZ/T2ME, RZ/T2L, RZ/T2N, RZ/N2L | Security Package | https://www.renesas.com/software-tool/rz-mpu-security-package-rtos-bare-metal |
| Synergy Platform | MCU drivers | https://www.renesas.com/products/microcontrollers-microprocessors/renesas-synergy-platform-mcus/renesas-synergy-software-package |

## 1.3 Renesas Secure Key Injection Advantages

Secure key injection and update, combined with the security engine's support of wrapped keys, address many vulnerabilities associated with using plaintext keys:

- Plaintext keys are never stored in code flash. In the event of a program memory breach, the sensitive key material is protected.
- Plaintext keys are never stored in RAM. In the event of malicious code executing on the system, the sensitive key material is still protected.
- Keys can be securely stored in code flash, data flash, or even copied into external memory, enabling unlimited secure key storage.

In addition, Renesas key wrapping techniques protect against device cloning, as discussed below.

### 1.3.1 Advantages of Key Wrapping over Key Encryption

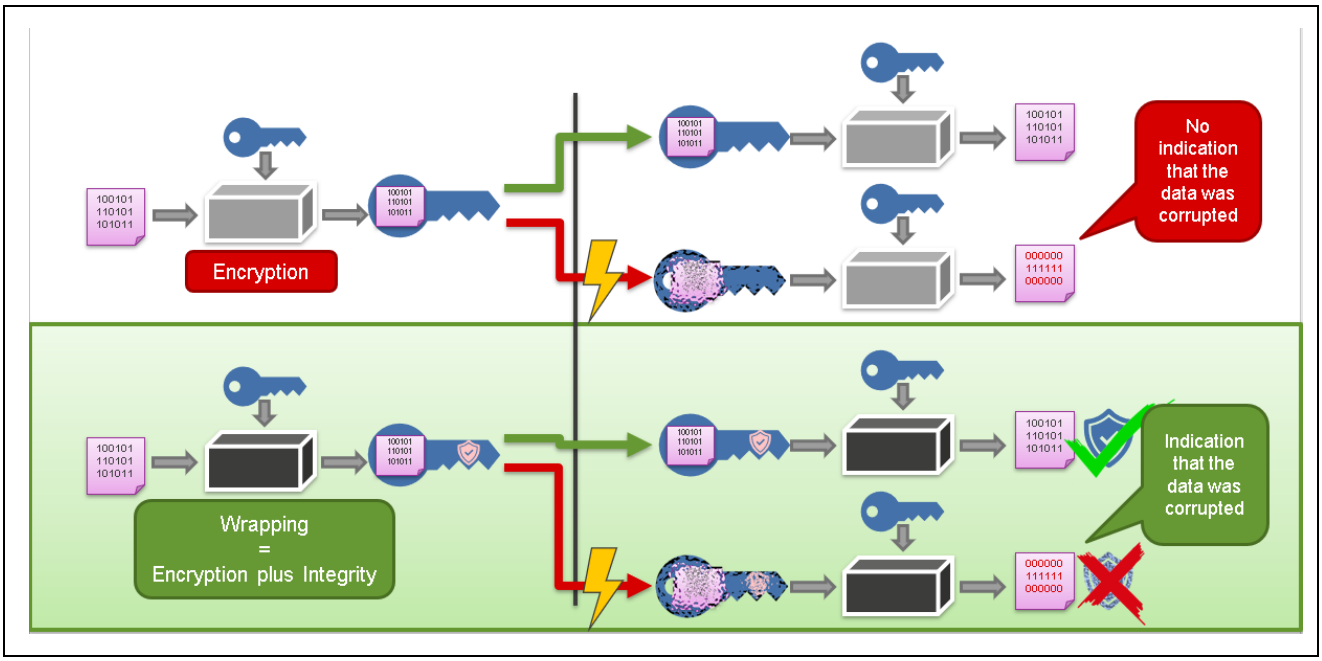


**Figure 1-3 Key Wrapping vs. Key Encryption**

It is important to understand the difference between wrapping and encrypting for secure asset storage. When data is encrypted and sent to another recipient, if that recipient has the same key, they can decrypt the data. This results in a confidential exchange of information. However, what if there was a problem with the transmission of the encrypted data? If the recipient unknowingly receives corrupted information, the decryption algorithm will generate garbage data, with no indication that the original data has been corrupted.

Wrapping solves this problem by appending a Message Authentication Code to the encrypted output for integrity checking.

### 1.3.2 Advantages of Key Wrapping using an HUK

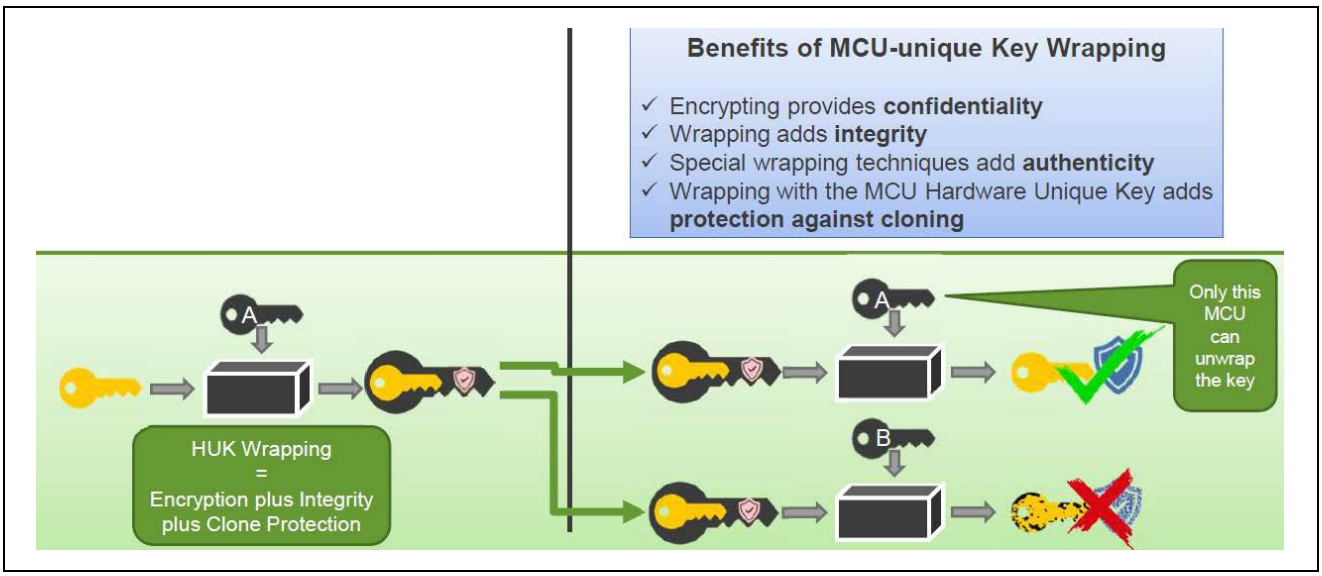


**Figure 1-4 Key Wrapping using an HUK**

Using a Hardware Unique Key (HUK) to wrap the stored keys adds another protection feature – clone protection. If the wrapped key is transmitted or copied to another MCU/MPU, that MCU/MPU will not be able to unwrap nor use the copied key. Even if the entire MCU contents are copied onto another device, the keys cannot be used nor exposed.

## 1.4 Wrapped Key Injection Procedure Overview

This section describes the injection procedure for wrapped keys, which requires the capabilities provided by the Security Key Management Tool. The types of keys supported vary by MCU/MPU. Refer to *Table 1-1 MCU/MPU Related Information* for the application notes and drivers for each device.

### 1.4.1 General Steps for Secure Key Injection

Secure Key Injection is the process by which application keys are stored MCU-uniquely wrapped via a mechanism that enables no plaintext key exposure during the provisioning process. The exact mechanism of this process is dependent on the specific MCU/MPU. Some devices support secure key injection over the programming interface; other devices support secure key injection using firmware running on the device. Note that key preparation steps using the Security Key Management Tool where key material is exposed in plaintext must be performed in a secure environment.

There are three high-level steps for key injection.

1. The first step in the secure key injection process is to use the Renesas Device Lifecycle Management (DLM) key wrap service to wrap an arbitrary User Factory Programming Key (UFPK) using the Renesas Hardware Root Key (HRK), providing a wrapped UFPK (W-UFPK). The UFPK is a 256-bit value selected by the user. The same UFPK can be used to inject any number of keys. The Security Key Management Tool can create a random UFPK and provide it as a key file suitable for sending to the key wrap service. The Tool can also accept a specifed UFPK and create a key file suitable for sending to the key wrap service.

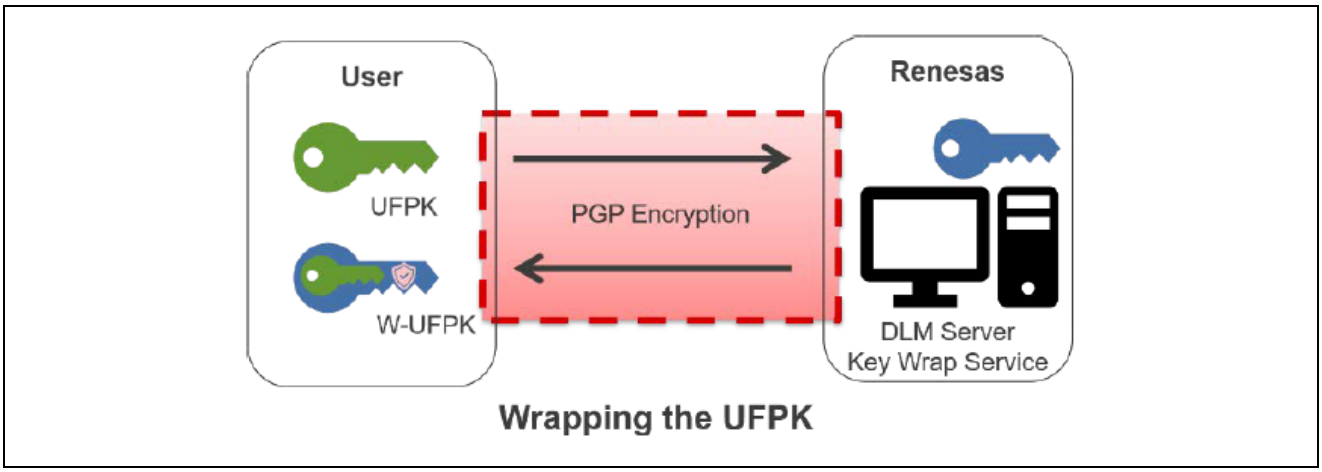


**Figure 1-5 Wrapping the UFPK using DLM server**

2. Next, the user key must be wrapped with UFPK. The Security Key Management Tool can wrap a user key, specified as either raw data or in a binary key file, and generate files that can be used for secure key injection by the selected device. Note that not all output file types are supported by all device Families. For example, the Renesas RA Family supports secure key injection only via the programming interface; therefore, a Renesas Flash Programmer secure key file must be generated.

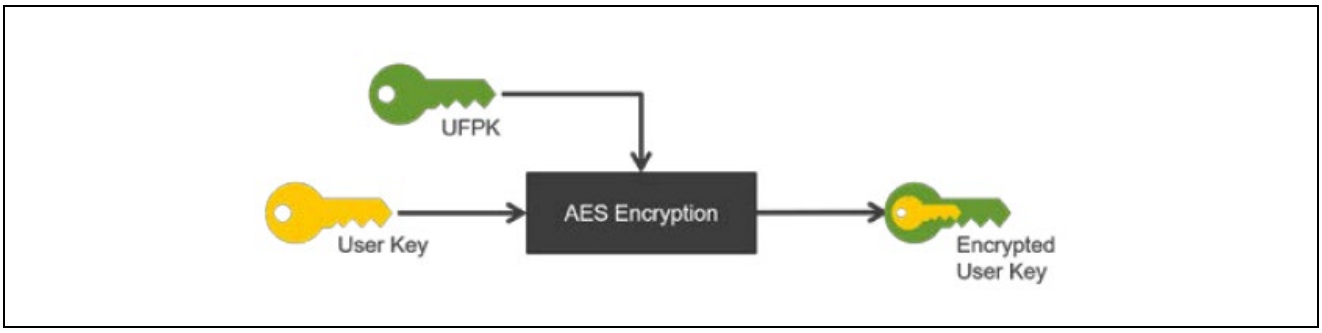


**Figure 1-6 Encrypt the User Key with UFPK**

3. Finally, the user key is injected via either the serial programming interface or by firmware running on the device, depending on the key injection process supported by the selected device. The input to this process includes both the wrapped UFPK (W-UFPK) and the wrapped user key prepared in the previous steps.

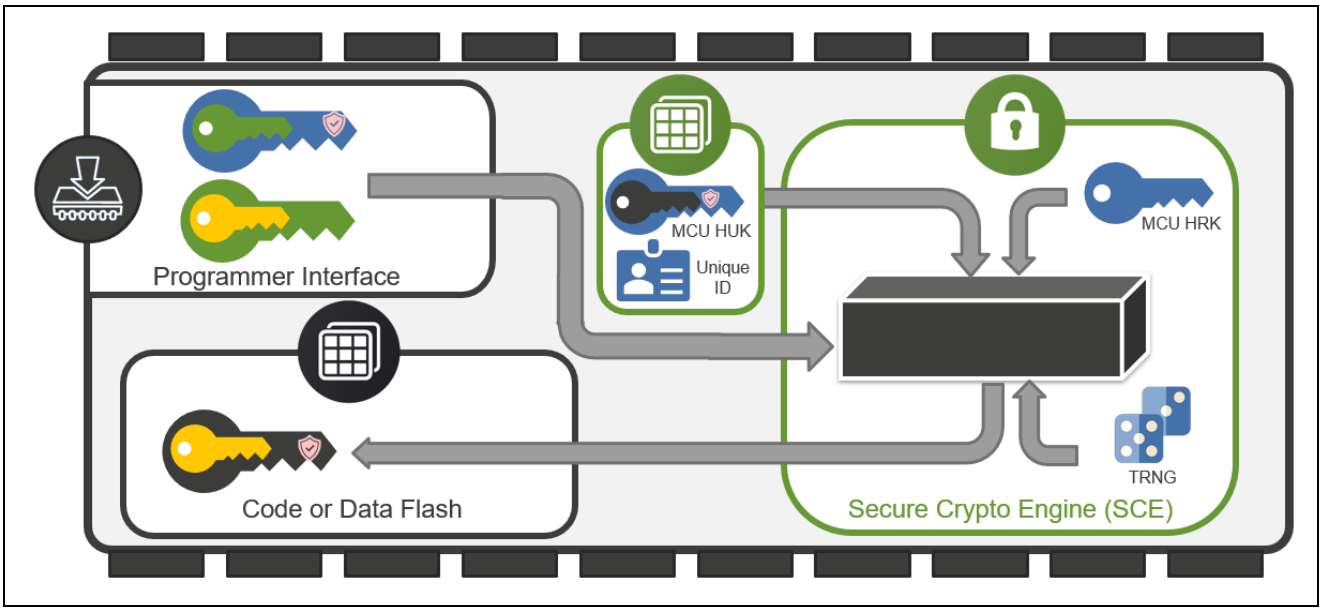


**Figure 1-7 Inject a User Key over the Serial Programming Interface**

### 1.4.2 General Steps for Secure Key Update

To enable secure key injection in the field, one or more Key-Update Keys (KUK) must be injected during production programming/provisioning. KUKs, like other cryptographic keys, can be stored in either code flash or data flash (if available on the MCU). Since injecting new keys in the field is usually done to replace older keys (key rotation or re-keying), this process is referred to as “key update”; however, this process can also be used to inject a new key in the field. No previously injected keys are deleted with this process.

Note that if the target device supports key injection via the programming interface, additional KUKs CANNOT be injected after the programming interface is disabled. In this case, once a product is in the field with its programming interface disabled, new keys can ONLY be injected via a pre-existing KUK. Therefore, it is highly recommended that multiple KUKs be injected during production provisioning. This enables the KUK to be rotated or revoked to adhere to an infrastructure security policy or to respond to a key exposure security breach.

There are three steps high-level steps for key update.

1. The first step is to generate and inject a KUK, as described in *Section 1.4.1 General Steps for Secure Key* . The KUK must be injected as a “KUK” key type.

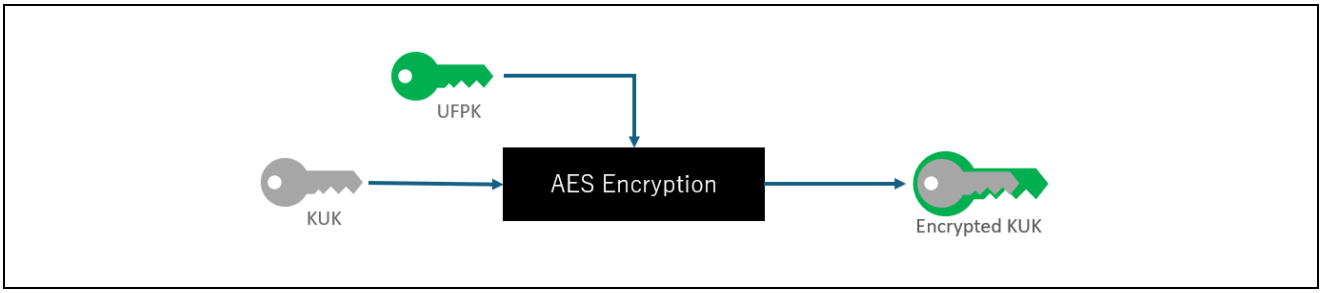


**Figure 1-8 Encrypt a KUK with the UFPK**

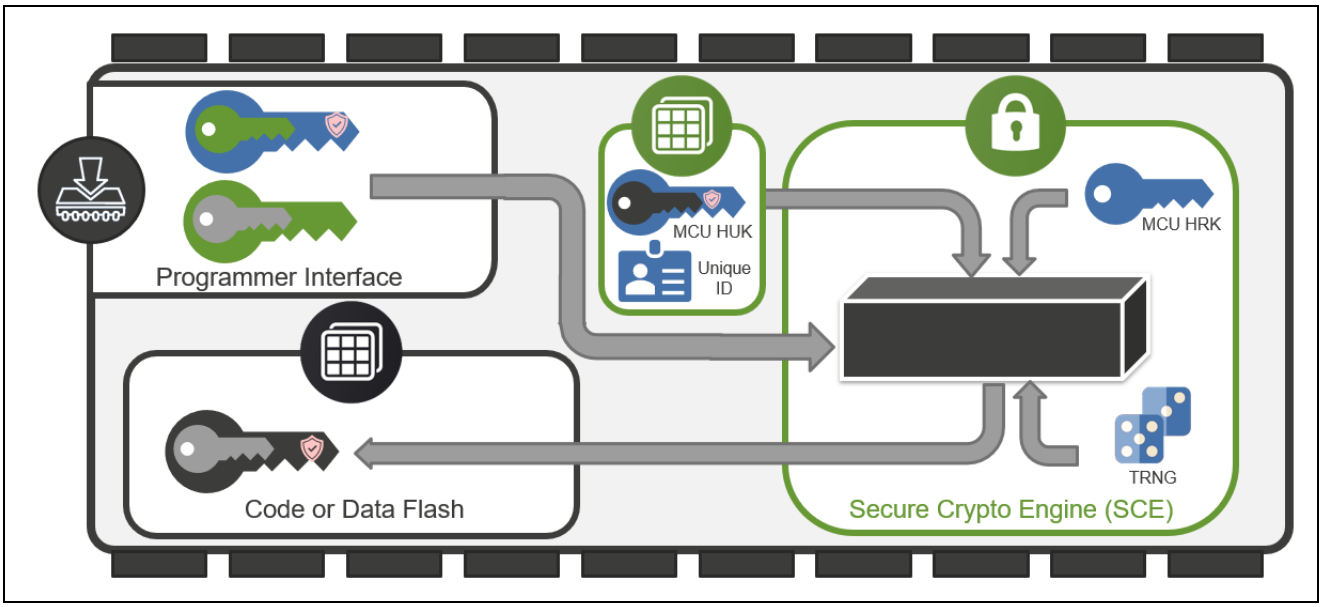


**Figure 1-9 Inject a KUK Over the Serial Programming Interface**

2. The second step is to use the KUK to wrap the new user key. This is similar to secure key injection, but using the KUK instead of the UFPK.

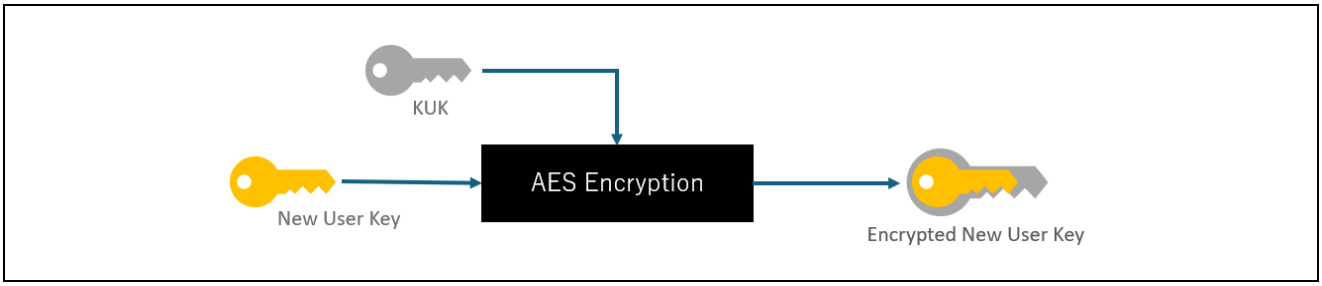


**Figure 1-10 Encrypt the New User Key with a KUK**

3. The last step is to inject a new User Key using the respective device driver and the already-injected KUK. Please refer to the manual of each MCU/MPU device driver for the key update API and how to use it.

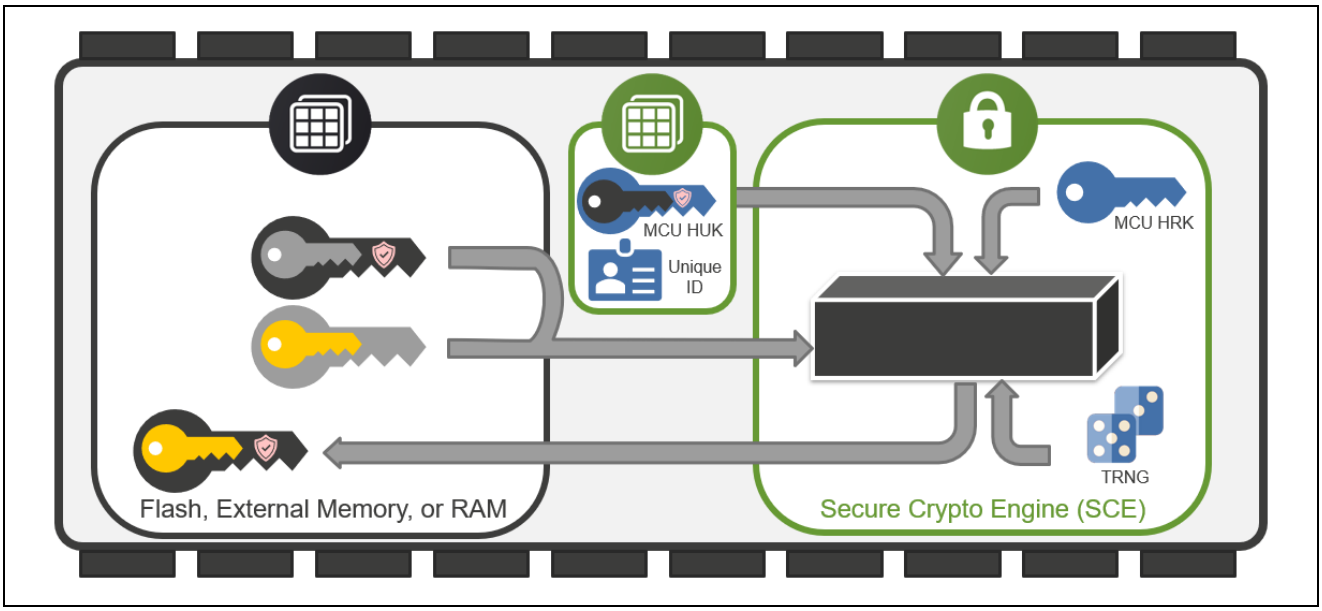


**Figure 1-11 Update the User Key**

## 1.5 Renesas Security Functions

In addition to key injection and updating, Renesas products provide various security features that utilize Renesas Secure IP. The Security Key Management Tool provides functions to support the use of those security features. Supported features vary by MCU/MPU. Please refer to the application notes and drivers for each device by referring to *Table 1-1 MCU/MPU Related Information*.

### 1.5.1 First Stage Bootloader

In a secure system, an application program must be executed only after confirming that it has not been altered, either maliciously or inadvertently. This confirmation, typically performed by a boot loader, can be a simple integrity check, or it can involve signature verification to ensure authenticity. However, it is important to note that the legitimacy of the boot loader itself must also be guaranteed. The secure boot sequence is often implemented in stages, starting with an immutable portion to ensure a strong root of trust.

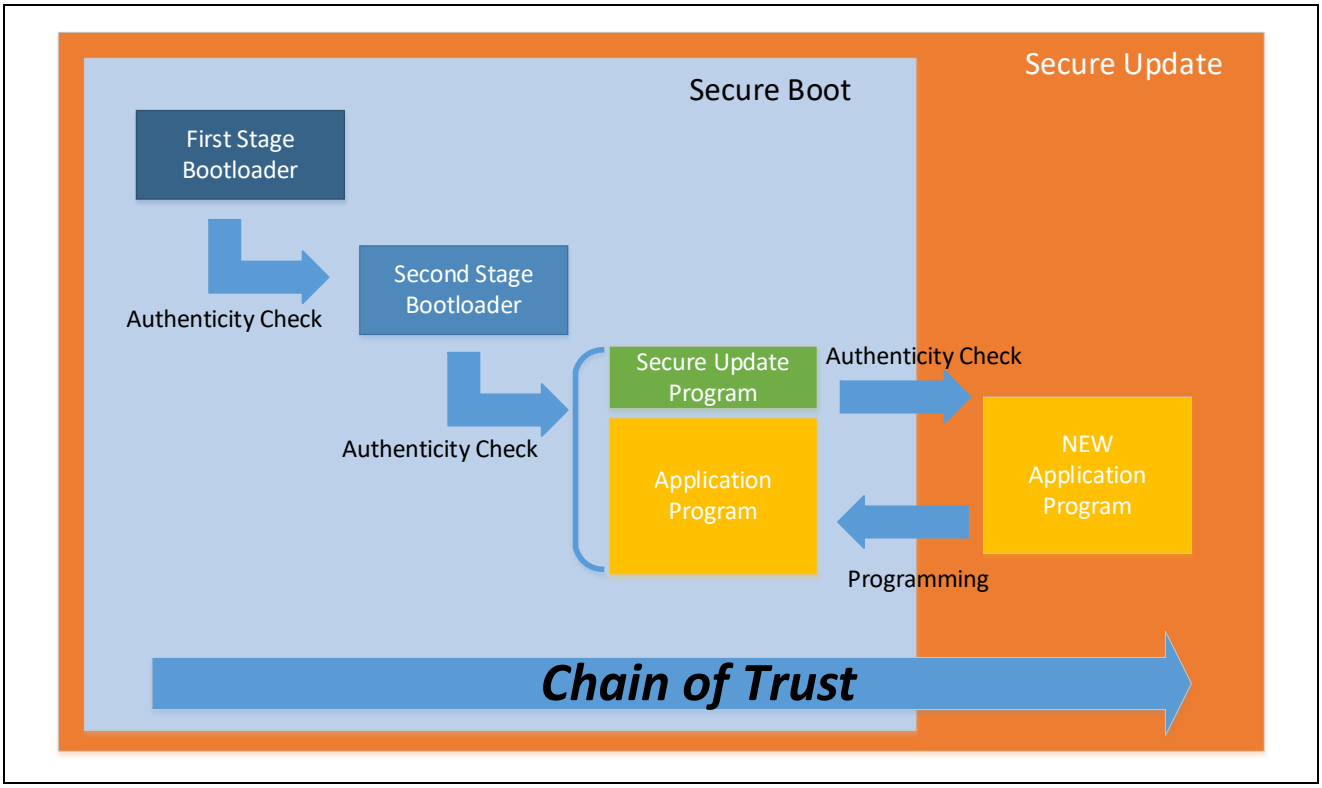


**Figure 1-12 Chain of Trust in a secure system.**

Some Renesas MCU products include an immutable First Stage Bootloader (FSBL), located in the masked ROM or in the One Time Programmable (OTP) ROM area of the device. An OEM Root Public Key is registered and locked in the device during production programming. The FSBL uses the OEM Root Public Key to authenticate the Second Stage Bootloader's public key, which is then used to authenticate the Second Stage Bootloader.

The following figures show the flow of the secure boot process for both initial production programming and normal application execution, using the OEM Root Public and Private Keys (OEM_ROOT_PK and OEM_ROOT_SK), the OEM Bootloader Public and Private Keys (OEM_BL_PK and OEM_BL_SK), and the OEM (second stage) bootloader (OEM_BL).

The Security Key Management Tool can generate the Key Certificate and Code Certificate used during this validation. For more details, please refer to the application notes and drivers for the target MCU/MPU.

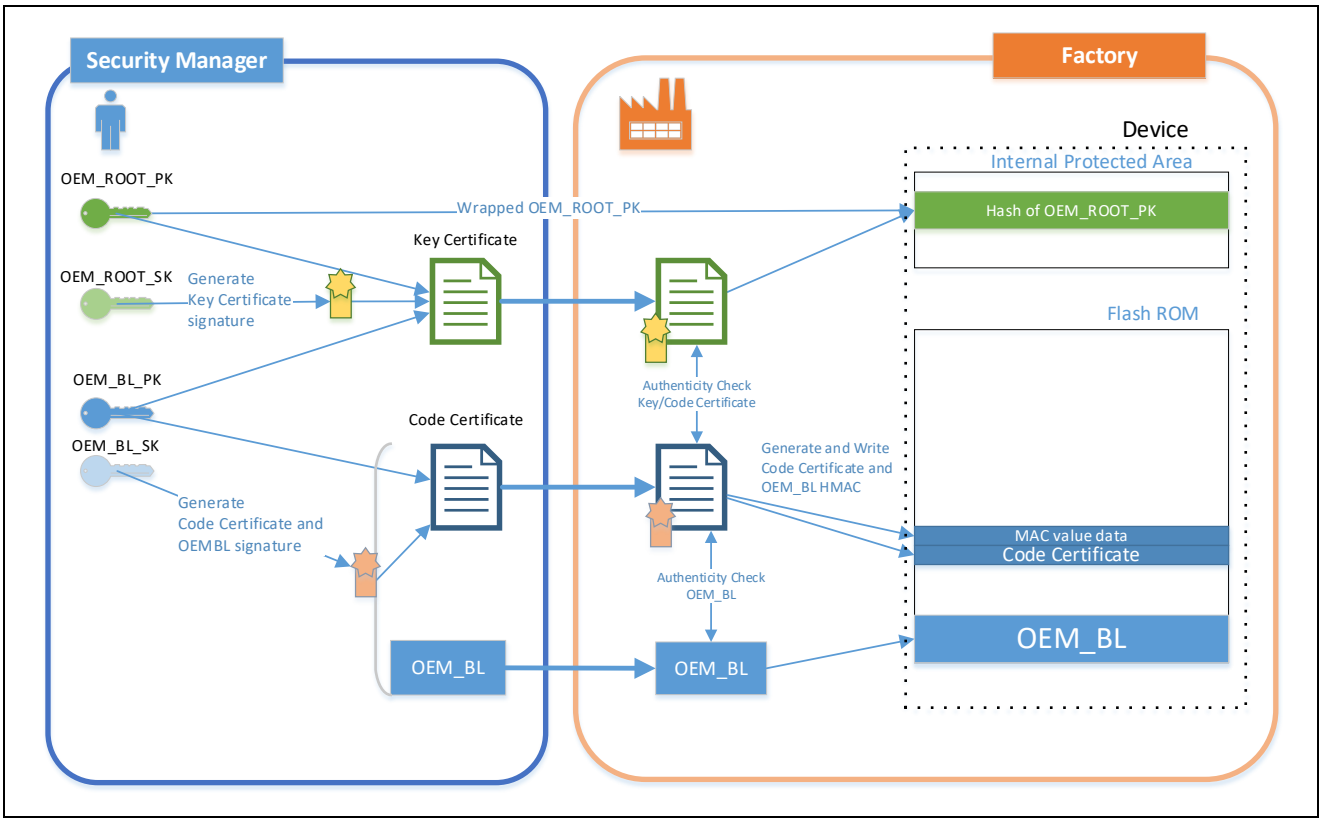


**Figure 1-13 Authenticity Check flow performed when programming OEM_BL at the factory.**

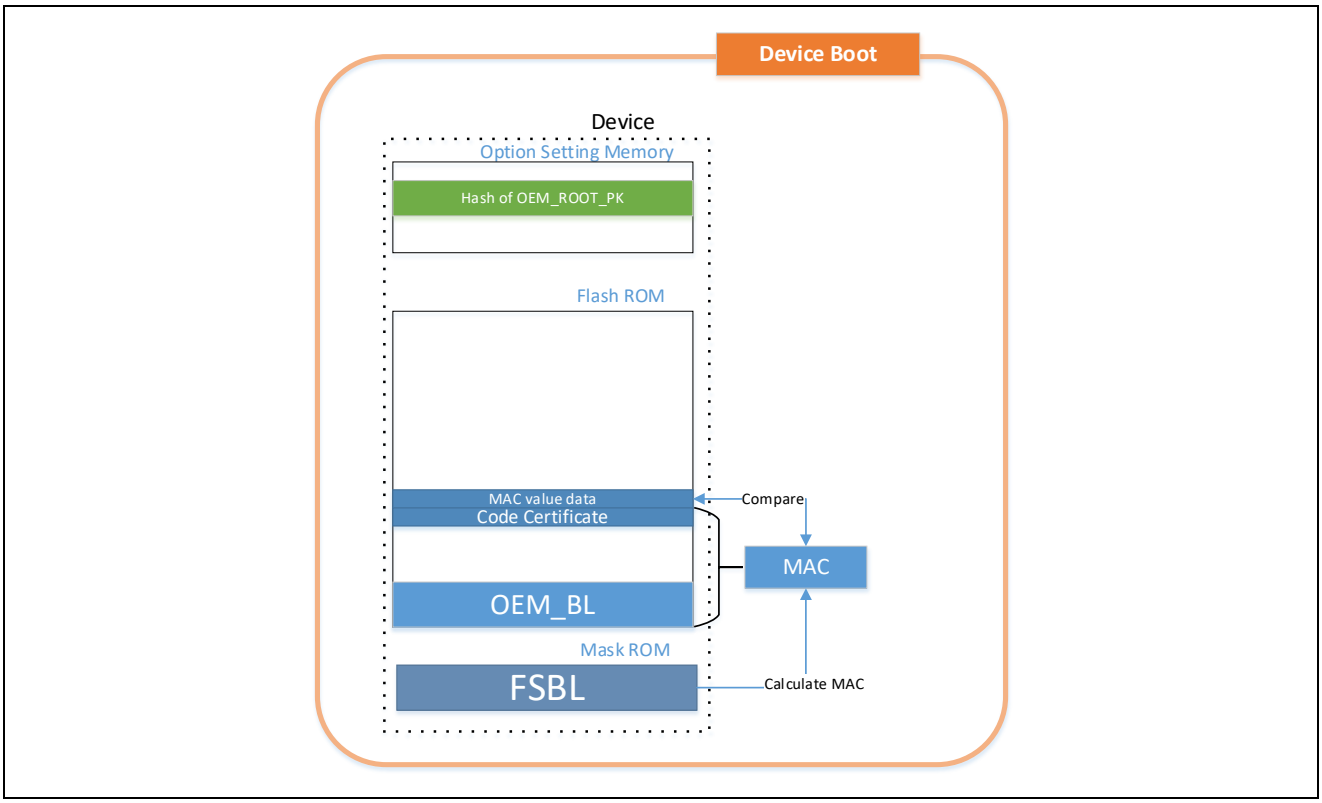


**Figure 1-14 FSBL operation at device startup.**

### 1.5.2 Decryption On-The-Fly/On-The-Fly Decryption

To protect application programs and sensitive data stored in external memory, the contents of the external memory should be encrypted. The Decryption On-The-Fly or On-The-Fly Decryption feature (DOTF) supported on select Renesas MCU/MPU products enables transparent decryption of encrypted external memory when executing code or reading data.

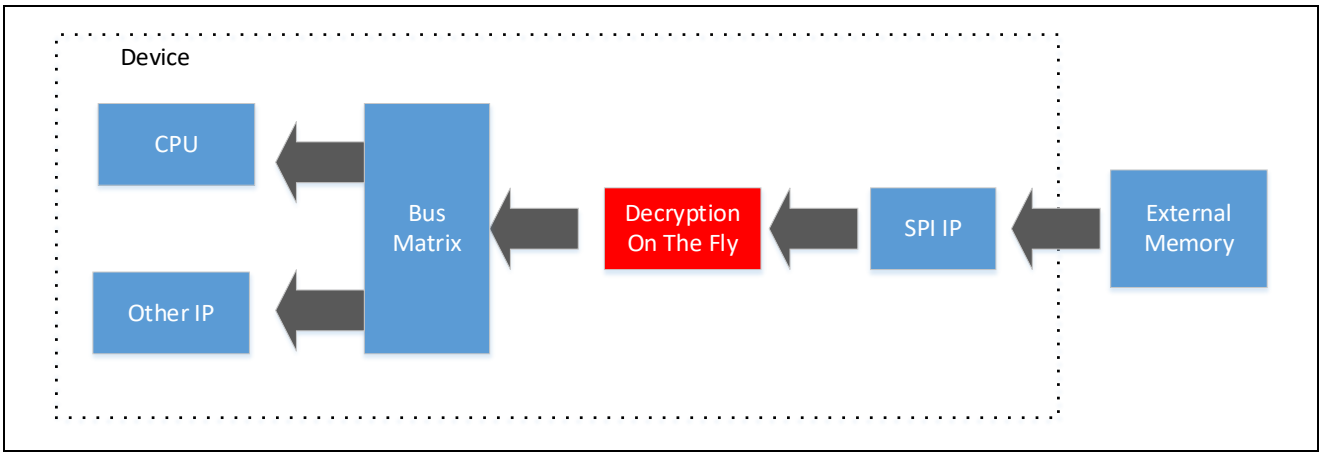


**Figure 1-15 Example of internal system bus for DOTF implemented MCU/MPU.**

The Security Key Management Tool supports preparing application information (executable code or sensitive data) for use with Renesas DOTF feature.

### 1.5.3 Secure Factory Programming

The Secure Factory Programming feature supported on select Renesas MCU/MPU products enables secure application programming in a non-secure programming environment.

The complete final content of the MCU (code, data, and keys) is delivered in encrypted format, protected by a Chain of Trust such that no plaintext of the MCU contents is ever exposed outside the MCU. This ensures that even if the application programming file is leaked from the factory at the time of production programming, the confidentiality of the contents of the programming file is maintained.

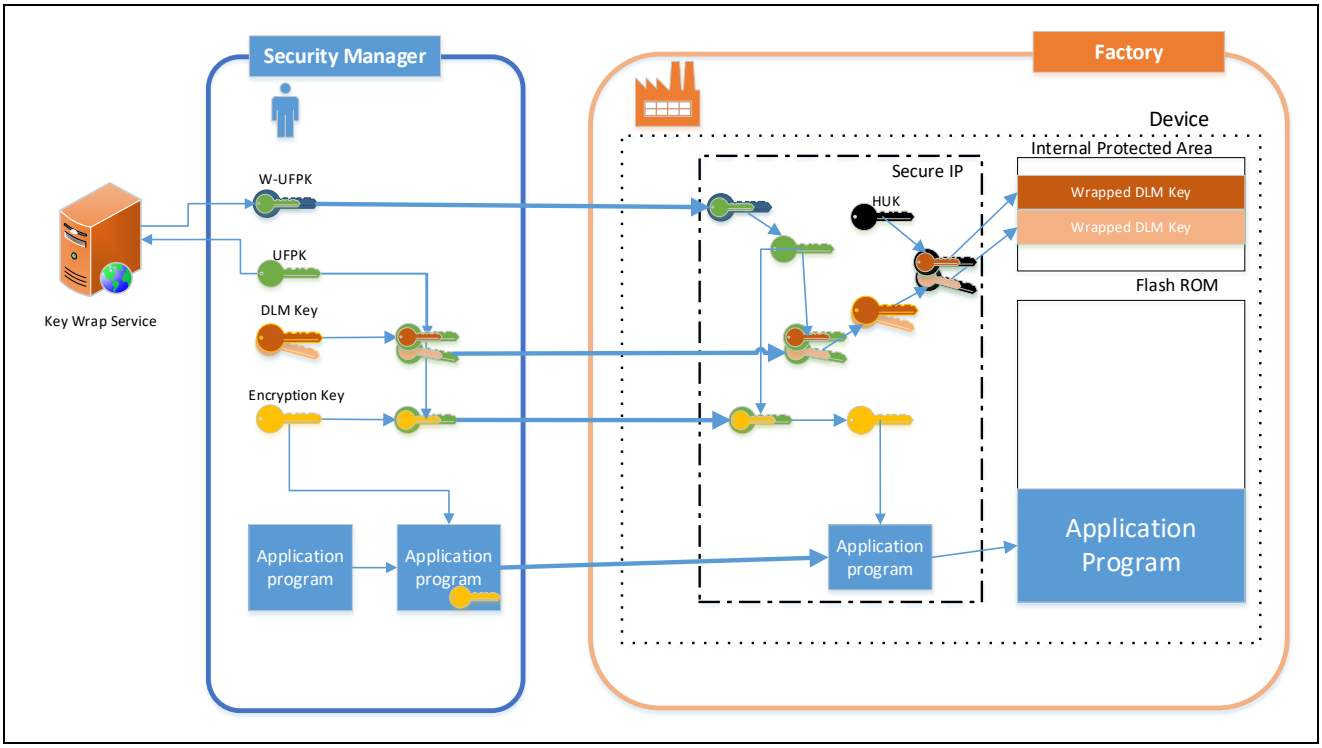


**Figure 1-16 System example of MCU/MPU with Secure Factory Programming function implemented.**

The Security Key Management Tool supports the following functions to realize the Secure Factory Programming function:

- Encryption of user application program
- Wrapping of application keys used in the application program
- Injection of Device Lifecycle Management / Authentication Level keys
- First Stage Bootloader setup
- Transition to final DLM state

Please refer to the MCU/MPU documentation for the list of features supported by the specific device and any limitations of Secure Factory Programming, such as support for programming internal memories only.

# 2. Overview

The Security Key Management Tool supports the security features of Renesas MCUs/MPUs.

Three versions of the tool are available:

- Command Line Interface – Execute single commands from the Operating Systems command line, or include several commands in a batch file or script. Designed to support key managers and group development.
- Standalone Graphical User Interface – An intuitive GUI designed to assist firmware developers. Useful when developing with third party IDEs.
- e$^2$ studio plugin – The GUI interface integrated into Renesas's e$^2$ studio, for simplified development support.

## 2.1 Features

The functions and MCU/MPU devices supported by the Security Key Management Tool are summarized below:

### 2.1.1 Secure Key Injection and Update

The following secure key injection and update related functions are supported. Refer to *Table 2-1* for a list of supported MCUs and MPUs.

[1] UFPK generation and generation of files in a format suitable for submission to the Renesas Key Wrap Service
[2] Wrapping DLM keys with a UFPK for secure key injection (if DLM keys are supported on the selected MCU or MPU)
[3] Wrapping user keys with a UFPK for secure key injection
[4] Wrapping user keys with a KUK for secure key update

Table 2-1 Supported Key Injection/Update Functions by MCU/MPU Family (1)

| Family | Supported Functions | | | |
|---|---|---|---|---|
| | [1] | [2] | [3] | [4] |
| RA Family | | | | |
| RSIP-E51A Security Function and Protected Mode | ✓ | ✓ | ✓ | ✓ |
| RSIP-E51A Compatibility Mode | ✓ | - | ✓ | - |
| RSIP-E50D Security Function and Protected Mode | ✓ | ✓ | ✓ | ✓ |
| RSIP-E50D Compatibility Mode | ✓ | - | ✓ | - |
| RSIP-E31A Security Function and Protected Mode | ✓ | ✓ | ✓ | ✓ |
| RSIP-E31A Compatibility Mode | ✓ | - | ✓ | - |
| RSIP-E31B Security Function and Protected Mode | ✓ | ✓ | ✓ | ✓ |
| RSIP-E31B Compatibility Mode | ✓ | - | ✓ | - |
| RSIP-E11A Security Function and Protected Mode | ✓ | ✓ | ✓ | ✓ |
| RSIP-E11A Compatibility Mode | ✓ | - | ✓ | - |
| SCE9 Security Function and Protected Mode | ✓ | ✓ | ✓ | ✓ |
| SCE9 Compatibility Mode | ✓ | - | ✓ | - |
| SCE7 | ✓ | - | ✓ | - |
| SCE5_B | ✓ | ✓ | ✓ | - |
| SCE5 | ✓ | - | ✓ | - |

Table 2-2 Supported Key Injection/Update Functions by MCU/MPU Family (2)

| Family | Supported Functions | | | |
|---|---|---|---|---|
| | [1] | [2] | [3] | [4] |
| RX Family | | | | |
| TSIP | ✓ | - | ✓ | ✓ |
| TSIP-Lite | ✓ | - | ✓ | ✓ |
| RSIP-E50D Security Function and Protected Mode | ✓ | ✓ | ✓ | ✓ |
| RSIP-E50D Compatibility Mode | ✓ | - | ✓ | - |
| RSIP-E11A Security Function and Protected Mode | ✓ | - | ✓ | ✓ |
| RSIP-E11A Compatibility Mode | ✓ | - | ✓ | - |
| RZ Family | | | | |
| RZ/T2M | ✓ | - | ✓ | ✓ |
| RZ/T2ME | ✓ | - | ✓ | ✓ |
| RZ/T2L | ✓ | - | ✓ | ✓ |
| RZ/N2L | ✓ | - | ✓ | ✓ |
| TSIP | ✓ | - | ✓ | ✓ |
| Renesas Synergy Platform | | | | |
| SCE7 | ✓ | - | ✓ | ✓ |
| SCE5 | ✓ | - | ✓ | ✓ |

Output in the following file formats is supported. (Note that not all MCUs and MPUs support Renesas key files.):

- Renesas key file
- C source file
- Binary data
- Motorola hex file

### 2.1.2 Security features supported by Security Key Management Tool

In addition to injecting and updating secure keys, the Security Key Management Tool can generate data for use by the following functions of the device. See Table 2-3 for supported MCUs/MPUs.

(1) Generate Key Certificate and Code Certificate for FSBL
(2) Encrypt code/data for use with DOTF
(3) Prepare a file for use with Secure Factory Programming
(4) Prepare a file for use with the TSIP Update solution
(5) Generate a header for RA for Wireless

Table 2-3 MCU/MPU Family Secure Feature Support (1)

| MCU/MPU | Security Functions | | | | |
|---|---|---|---|---|---|
| | (1) | (2) | (3) | (4) | (5) |
| RA Family | | | | | |
| RSIP-E51A Security Function and Protected Mode | ✓ | ✓ | ✓ | - | - |
| RSIP-E51A Compatibility Mode | - | ✓ | - | - | - |
| RSIP-E50D Security Function and Protected Mode | ✓ | ✓ | ✓ | - | - |
| RSIP-E50D Compatibility Mode | - | ✓ | - | - | - |
| RSIP-E31A Security Function and Protected Mode | - | - | ✓ | - | - |
| RSIP-E31A Compatibility Mode | - | - | - | - | - |
| RSIP-E31B Security Function and Protected Mode | - | ✓ | ✓ | - | ✓ |
| RSIP-E31B Compatibility Mode | - | ✓ | - | - | - |
| RSIP-E11A Security Function and Protected Mode | - | - | ✓ | - | - |
| RSIP-E11A Compatibility Mode | - | - | - | - | - |
| SCE9 Security Function and Protected Mode | - | - | - | - | - |
| SCE9 Compatibility Mode | - | - | - | - | - |
| SCE7 | - | - | - | - | - |
| SCE5_B | - | - | - | - | - |
| SCE5 | - | - | - | - | - |

Table 2-4 MCU/MPU Family Secure Feature Support (2)

| MCU/MPU | Security Functions | | | | |
|---|---|---|---|---|---|
| | (1) | (2) | (3) | (4) | (5) |
| RX Family | | | | | |
| TSIP | - | - | - | ✓ | - |
| TSIP-Lite | - | - | - | ✓ | - |
| RSIP-E50D Security Function and Protected Mode | ✓ | ✓ | ✓ | - | - |
| RSIP-E50D Compatibility Mode | - | ✓ | - | - | - |
| RSIP-E11A Security Function and Protected Mode | - | - | - | ✓ | - |
| RSIP-E11A Compatibility Mode | - | - | - | - | - |
| RZ Family | | | | | |
| RZ/T2M | - | - | - | - | - |
| RZ/T2ME | - | ✓ | - | - | - |
| RZ/T2L | - | - | - | - | - |
| RZ/N2L | - | - | - | - | - |
| TSIP | - | - | - | - | - |
| Renesas Synergy Platform | | | | | |
| SCE7 | - | - | - | - | - |
| SCE5 | - | - | - | - | - |

## 2.2 Operating Environment

### 2.2.1 Hardware Environment

#### (1) Host PC

- Processor: 1GHz or faster
- Main memory: 1Gbyte or more
- Display setting : Resolution 1366 x 768 or higher
  Display Scale 100% (recommended)

Note:

If the resolution or display scale is not listed above, some options in the GUI may not be displayed

### 2.2.2 Software Environment

#### (1) OSes supported

- Windows11
- Linux (Ubuntu 22.04 LTS, Ubuntu 24.04 TLS)
- macOS 14 Sonoma (Support for Apple Silicon only)

#### (2) e$^2$studio plugin version e$^2$studio operation check version

- e$^2$studio 2026-07

# 3. Descriptions of GUI Functions

This chapter describes the screen structure and functions of the GUI version of the Security Key Management Tool.

The Standalone and e²studio plugin versions have the same GUI configuration. The explanation is given for the Standalone version.

## 3.1 Main Window

The main window after startup consists of the following:

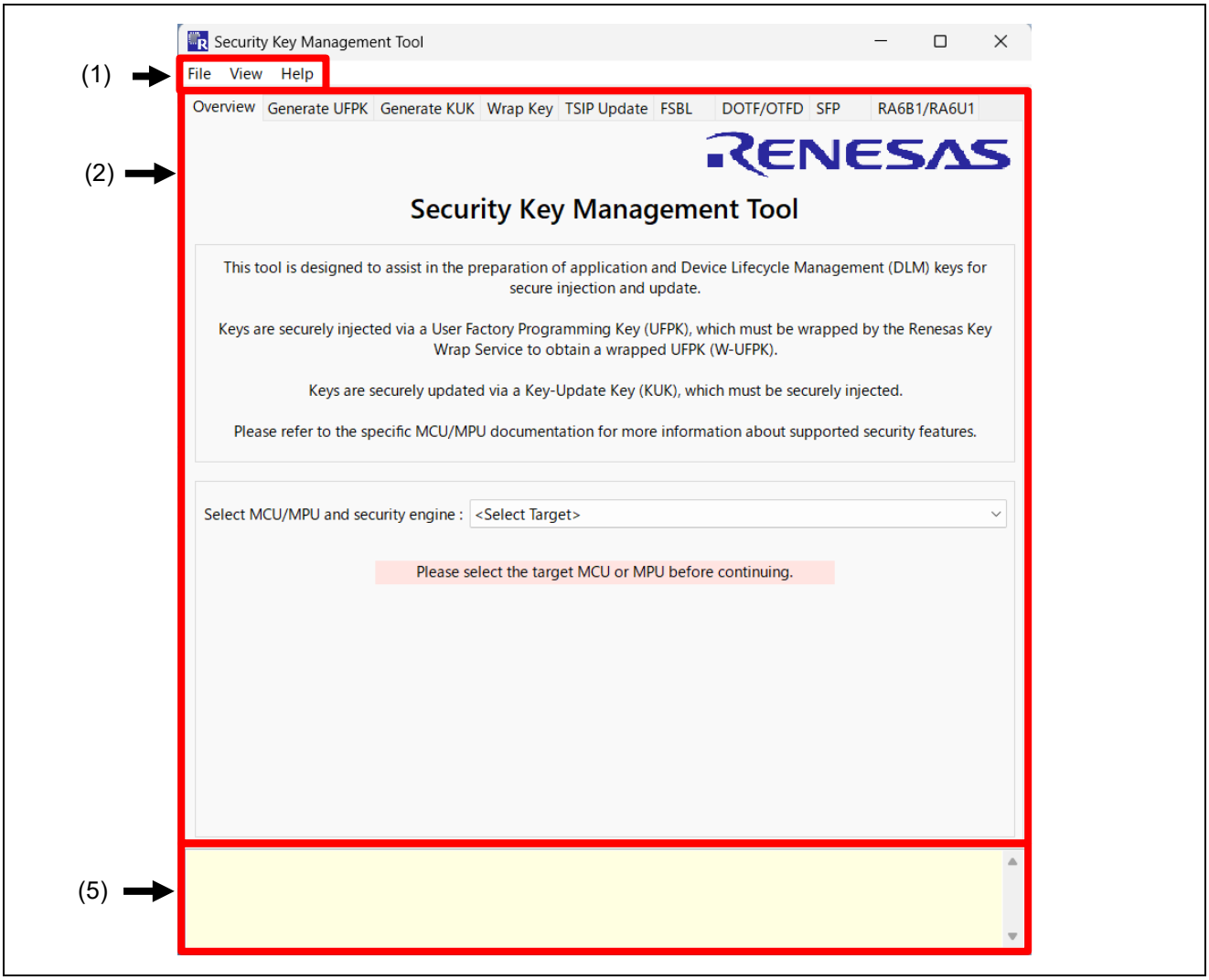


**Figure 3-1 Main Window of standalone version**

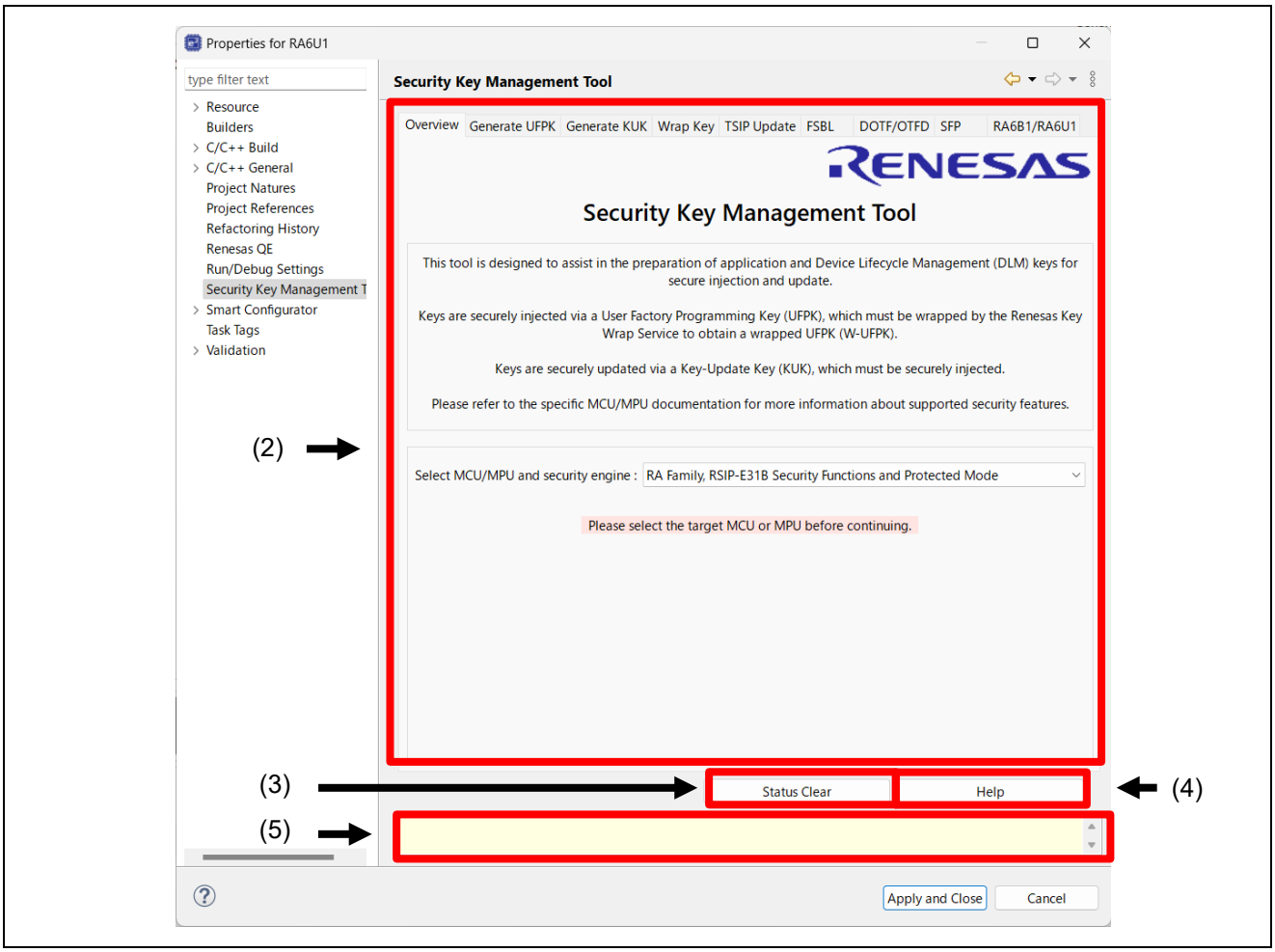


**Figure 3-2 Main Window of e²studio plugin version**

| No | Items | Description |
|---|---|---|
| (1) | Menu bar | Function only in the standalone version.<br>See 3.2 Menu bar for details. |
| (2) | Tab Window | Each tab provides an interface for generating files used as part of the secure key injection and/or update process. |
| (3) | **Status Clear** | Clears the execution results displayed in the Status area.<br>For the Standalone version, this function is provided in the Menu bar. |
| (4) | **Help** | Shows resources that are useful for understanding the Renesas key management system.<br>For the Standalone version. This function is provided in the Menu bar. |
| (5) | Status | Displays the status of the requested operation. |

## 3.2 Menu bar

This is a feature of the standalone version only.

### 3.2.1 [File] menu

Select from a menu of functions related to saving settings and a log function.

- [**Save…**]

Saves the input/output file information set in each tab to an XML file format file. Key data is not saved.

- [**Load…**]

The settings file saved in [**Save...**] is loaded and reflected in each tab.

In the case of the e$^2$studio plugin version, pressing the "**Apply and Close**" button saves the configuration information for each tab to the e$^2$studio project.

- [**Log Start…**]

Selecting [**Log Start**...] will write the text displayed in the GUI status window to the specified log file. For the e$^2$studio plugin version, this is supported via the right-click menu on the Status area.

- [**Log End**]

Stop the log output started with [**Log Start**...]. For the e$^2$studio plugin version, this is supported via the right-click menu on the Status area.

### 3.2.2 [View] menu

This menu is related to the GUI display.

- [**Status Clear**]

Clears the strings displayed in the Status area.

- [**Display Command Line**]

The Status area displays the command content when running in the CLI version.
The default setting enables command display.

### 3.2.3 [Help] menu

- [About Security Key Management Tool**…**]

Display the Help dialog. Shows resources that are useful for understanding the Renesas key management system.

### 3.2.4 Status area

The Status area displays the results of actions performed in each tab.
Right-clicking on the Status area allows you to perform several actions from the File menu and View menu.

#### 3.2.4.1 Right-clicking menu

- [**Copy**]

Selecting [**Copy**] copies the selected text in the Status area to the clipboard.
If no text is selected, it copies all text in the Status area to the clipboard.

- [**Log Start… / Log End**]

Selecting [**Log Start...**] writes the text displayed in the Status area to the specified log file. For the standalone version, this is supported via the [File] menu.

- [**Display Command Line**]

The Status area displays the command content when running in the CLI version.
The default setting enables command display.
For the standalone version, this is supported via the [File] menu.

- [**Clear**]

Clears the execution results displayed in the Status area.

## 3.3 [Overview] Tab

In this tab, select the target MCU/MPU and security engine to be used. Be sure to select the target device prior to performing operations on any other tab. The functionality of the other tabs is dictated by the device selection on this tab.

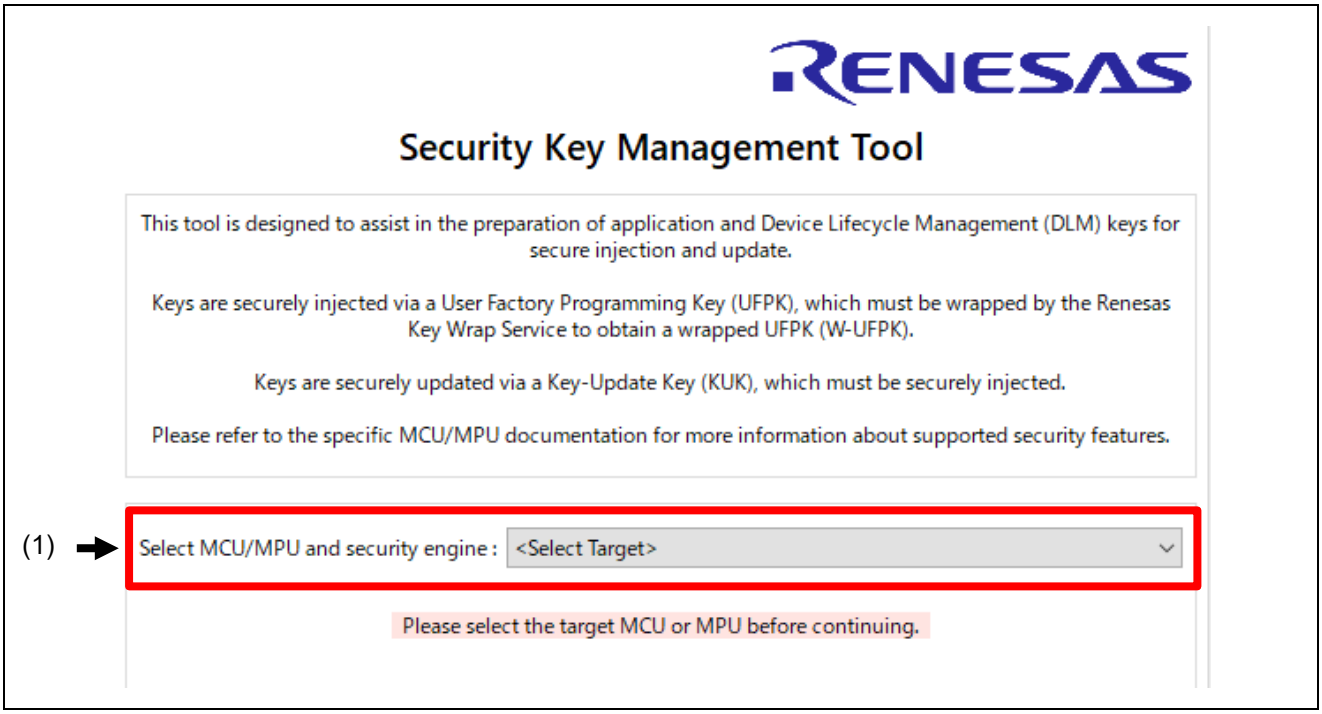


**Figure 3-3 [Overview] Tab**

| No | Item | Description |
|---|---|---|
| (1) | Select MPU/MCU and security engine | Select the target MCU/MPU and the cryptographic engine for secure key injection and/or update. |

## 3.4 [Generate UFPK] Tab

This tab generates a User Factory Programming Key (UFPK) file as a binary `*.key` file. This file must then be sent to the Renesas Key Wrap Service to obtain the wrapped UFPK (W-UFPK). The UFPK file will also be used to prepare keys for secure key injection.

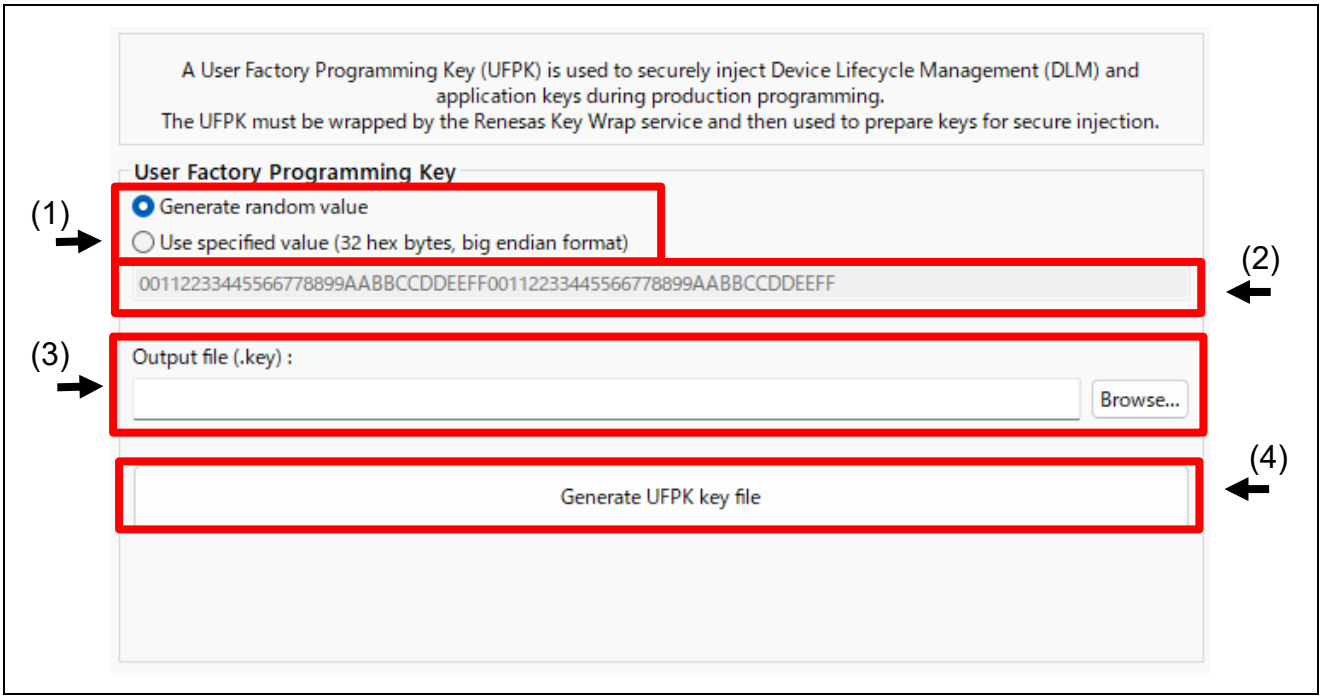


**Figure 3-4 [Generate UFPK] Tab**

| No | Items | Description |
|---|---|---|
| (1) | UFPK input format | Select whether to use the input value from (2) for the UFPK value or to use a random number generated by the tool. The specific amount of randomness of the random values generated by this tool is not guaranteed. |
| (2) | UFPK value | When **Use specified value** is selected in (1), the value entered in the text box is used as the UFPK value. This value must be specified as 32 hex bytes in big-endian format. |
| (3) | Output file(.key) | Set the path and file name of the UFPK file to be output. The extension of the output file must be set to `.key`. |
| (4) | **Generate UFPK key file** | Generate a UFPK file based on the information entered above.<br>Enabled when MCU/MPU and security engine is selected on [**Overview**] tab. |

The UFPK file generated in this tab must be sent to the Renesas Key Wrap Service (https://dlm.renesas.com/keywrap) to generate the W-UFPK. For more information on the Renesas Key Wrap Service, refer to the FAQ on the Key Wrap Service and additional information for each MCU/MPU (*Table 1-1 MCU/MPU Related Information*).

## 3.5 [Generate KUK] Tab

This tab generates the Key-Update Key (KUK) file as a binary `*.key` file. This file will be used not only for the secure injection of the KUK, but also for preparing other keys for secure key update.

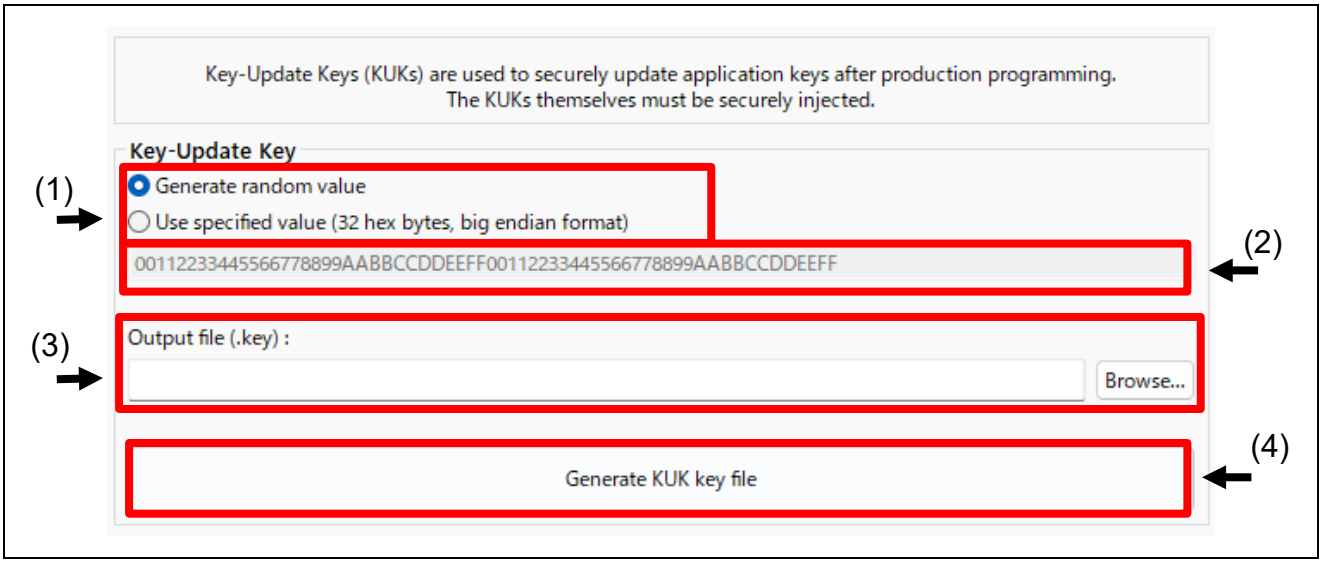


**Figure 3-5 [Generate KUK] Tab**

| No | Items | Description |
|---|---|---|
| (1) | KUK input format | Select whether to use the input value from (2) for the KUK value or to use a random number generated by the tool. The specific amount of randomness of the random values generated by this tool is not guaranteed. |
| (2) | KUK value | When **Use specified value** is selected in (1), the value entered in the text box is used as the KUK value. This value must be specified as 32 hex bytes in big-endian format. |
| (3) | Output file (.key) | Set the path and file name of the KUK file to be output. The extension of the output file must be set to `.key`. |
| (4) | **Generate KUK key file** | Generate a KUK file based on the information entered above.<br>Enabled when MCU/MPU and security engine is selected on [**Overview**] tab. |

## 3.6 [Wrap Key] Tab

Use this tab to encrypt the User Key and generate the files needed for secure injection or update.

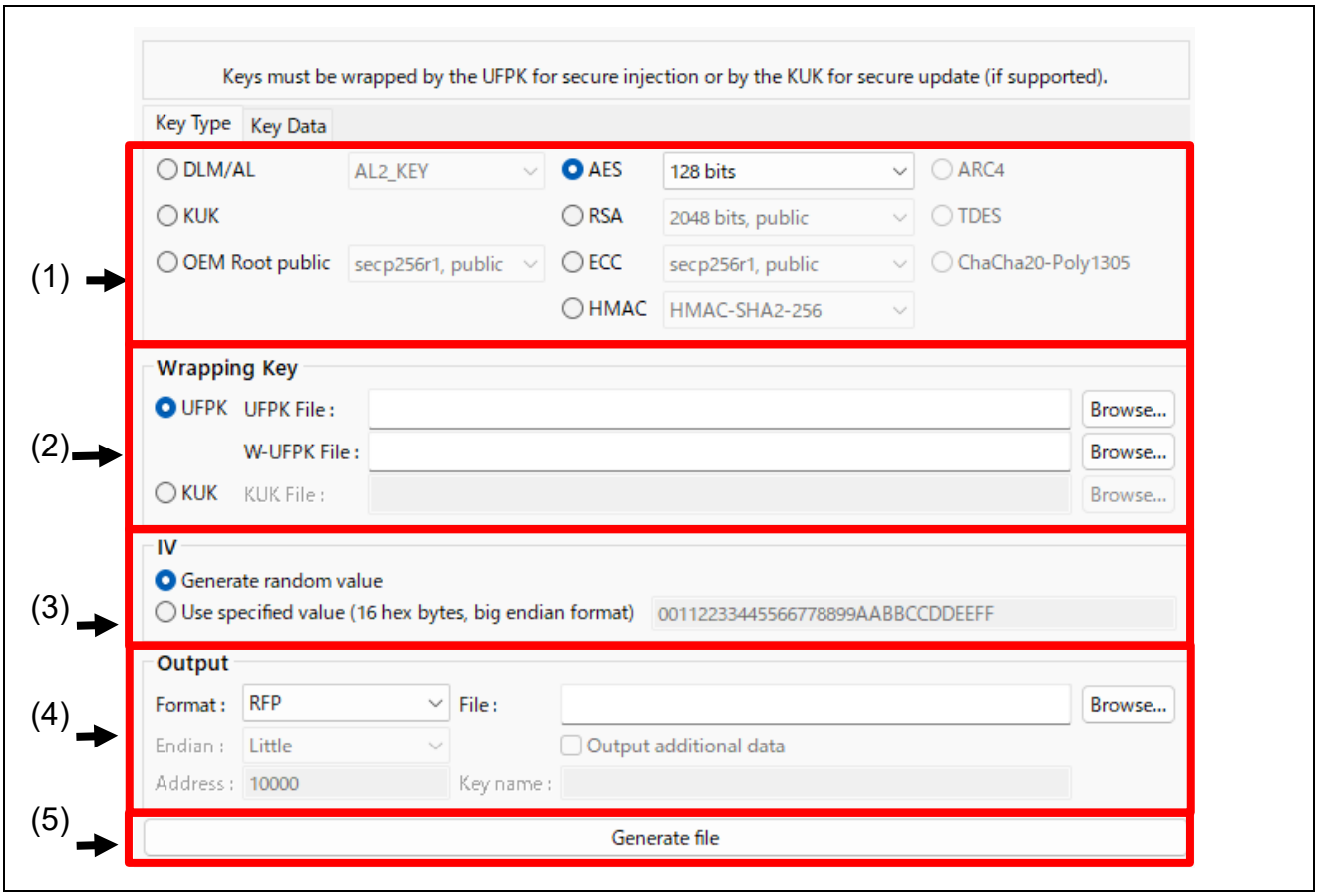


**Figure 3-6 [Wrap Key] Tab**

| No | Items | Description |
|---|---|---|
| (1) | [**Key Type**] and [**Key Data**] Tabs | After selecting the type of User key to be encrypted in the [**Key Type**] tab, enter the key data in the [**Key Data**] tab.<br>For details on the [**Key Type**] tab, refer to *Section 3.5.1 [Key Type] Tab*.<br>For details on the [**Key Data**] tab, refer to *Section 3.5.2 [Key Data] Tab*. |
| (2) | Wrapping Key | Set the key to be used for wrapping.<br>For details of the settings, refer to *Section 3.5.3 Wrapping Key*. |
| (3) | IV | Set the Initialization Vector (IV) to be used for wrapping.<br>For details of the settings, refer to *Section 3.5.4 IV*. |
| (4) | Output | Select the output file format. Note that not all devices support all possible output file formats. Refer to *Section 3.5.5 Output* for details. |
| (5) | **Generate file** | Generate a wrapped key file for injection or update in the format specified in (3) using the information in (1) and (2).<br>Enabled when MCU/MPU and security engine is selected on [**Overview**] tab. |

### 3.6.1 [Key Type] Tab

Use this tab to specify the type of key that is being prepared for secure injection or update. Not all key types and options are supported by all device families.

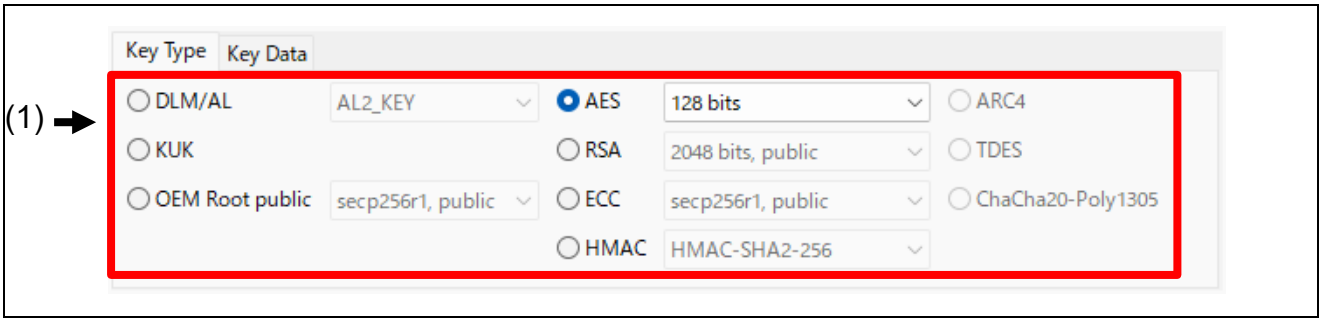


**Figure 3-7 [Key Type] Tab**

| No | Item | Description |
|---|---|---|
| (1) | Key type and key length selection | Select the algorithm and key length of the key that will be injected/updated. |

The following key types and key lengths can be selected. The supported key types and key lengths vary depending on the target MCU/MPU. For details on which keys and key lengths are supported, refer to the device's Hardware Manual and additional device information (*Table 1-1 MCU/MPU Related Information*).

Table 3-1 Options when DLM is selected

| Options | Description |
|---|---|
| DLM-SSD | Authentication key for SDD state transitions in DLM. |
| DLM-NSECSD | Authentication key for NSECSD state transitions in DLM. |
| DLM-RMA-REQ | Authentication key for RMA-REQ state transitions in DLM |
| AL2-KEY | Key for DLM authentication level AL2 transition |
| AL1-KEY | Key for DLM authentication level AL1 transition |
| RMA-KEY | Key for DLM authentication level RMA transition |

Table 3-2 Options when AES is selected

| Options | Description |
|---|---|
| 128 bits | AES 128-bit key |
| 192 bits | AES 192-bit key |
| 256 bits | AES 256-bit key |
| 128 bits, XTS | AES 128-bit XTS key |
| 256 bits, XTS | AES 256-bit XTS key |
| 256 bits, DOTF | AES 256-bit key<br>Outputs Encrypted User Key binary data for RA for Wireless DOTF. For the file format, refer to 4.5.2.1 AES-256-DOTF Option. |

Table 3-3 Options when RSA is selected

| Options | Description |
|---|---|
| 1024 bits, public | RSA 1024-bit public key |
| 1024 bits, private | RSA 1024-bit private key |
| 2048 bits, public | RSA 2048-bit public key |
| 2048 bits, private | RSA 2048-bit private key |
| 3072 bits, public | RSA 3072-bit public key |
| 3072 bits, private | RSA 3072-bit private key |
| 4096 bits, public | RSA 4096-bit public key |
| 4096 bits, private | RSA 4096-bit private key |
| RSA-2048-public-TLS | RSA 2048-bit public key for TLS API |

Table 3-4 Options when ECC is selected

| Options | Description |
|---|---|
| secp192r1, public | ECC NIST P-192 (secp192r1) public key |
| secp192r1, private | ECC NIST P-192 (secp192r1) private key |
| secp224r1, public | ECC NIST P-224 (secp224r1) public key |
| secp224r1, private | ECC NIST P-224 (secp224r1) private key |
| secp256r1, public | ECC NIST P-256 (secp256r1) public key |
| secp256r1, private | ECC NIST P-256 (secp256r1) private key |
| secp384r1, public | ECC NIST P-384 (secp384r1) public key |
| secp384r1, private | ECC NIST P-384 (secp384r1) private key |
| secp521r1, public | ECC NIST P-521 (secp521r1) public key |
| secp521r1, private | ECC NIST P-521 (secp521r1) private key |
| brainpool P256, public | ECC brainpoolP256r1 public key |
| brainpool P256, private | ECC brainpoolP256r1 private key |
| brainpool P384, public | ECC brainpoolP384r1 public key |
| brainpool P384, private | ECC brainpoolP384r1 private key |
| brainpool P512, public | ECC brainpoolP512r1 public key |
| brainpool P512, private | ECC brainpoolP512r1 private key |
| secp256k1, public | ECC Koblitz curve secp256k1 public key |
| secp256k1, public | ECC Koblitz curve secp256k1 private key |
| Ed25519, public | EdDSA Ed25519 public key |
| Ed25519, private | EdDSA Ed25519 private key |

Table 3-5 Options when HMAC is selected

| Options | Description |
|---|---|
| HMAC-SHA-1 | HMAC-SHA1 key |
| HMAC-SHA2-224 | HMAC-SHA2-224 key |
| HMAC-SHA2-256 | HMAC-SHA2-256 key |
| HMAC-SHA2-384 | HMAC-SHA2-384 key |
| HMAC-SHA2-512 | HMAC-SHA2-512 key |
| HMAC-SHA2-512/244 | HMAC-SHA2-512/224 key |
| HMAC-SHA2-512/256 | HMAC-SHA2-512/256 key |
| HMAC-SHA3-224 | HMAC-SHA3-224 key |
| HMAC-SHA3-256 | HMAC-SHA3-256 key |
| HMAC-SHA3-384 | HMAC-SHA3-384 key |
| HMAC-SHA3-512 | HMAC-SHA3-512 key |

Table 3-6 Options when OEM Root public is selected

| Options | Description |
|---|---|
| secp256r1, public | secp256r1 public key for using OEM Root key or OEM Root of Trust Key(OEM RoT Key) |
| secp384r1, public | Secp384r1 public key for using OEM Root key or OEM RoT Key |

Please refer to *Section 6 Usage Examples* of each key setting.

### 3.6.2 [Key Data] Tab

Use the [**Key Data**] tab to specify the plaintext key material to be prepared for secure injection or update. Note that the appearance of this tab depends on the key type that is specified on the [**Key Type**] tab.

#### 3.6.2.1 When DLM / KUK / AES / TDES / ARC4 / ChaCha20-Poly1305 / ECC Private Key Selected in [Key Type] Tab

**Figure 3-8 [Key Data] Tab when DLM / KUK / AES / TDES / ARC4 / ChaCha20-Poly1305 / ECC private key is selected**

| No | Items | Description |
|---|---|---|
| (1) | Input format | Select whether to provide a key file containing the key data in (2), to provide the raw plaintext data of the key in (3), or to have the tool generate a key (or key pair) with the output file name specified in (4). |
| (2) | File name | When **File** is selected in (1), select the key file that contains the plaintext key. Key files must be binary files and have the extension `*.key`. |
| (3) | Raw data | When **Raw** is selected in (1), enter the plaintext key data in hex format. The amount of data is dependent on the key type that was specified on the [**Key Type**] tab. |
| (4) | Output key file | When **Random** is selected in (1), the tool will generate the key data. Specify the output file of the plaintext key data generated in the tool. The output plaintext key file can be a text file or a binary file.<br>To generate ECC keys, select the “private” key type. Both private and public keys are generated at the same time, and the specified file name is appended with `_public` for public keys and `_private` for private keys. Injection/update files are generated only for the private key. The generated public key file can be used as input to create key injection files for the public key. Not all ECC key types are supported. See *Table 3-7 Supported ECC key generation.*<br>**Note**: The specific amount of randomness of the random values generated by this tool is not guaranteed. Keys generated by this tool should be used for prototyping and test purposes only. |

Table 3-7 Supported ECC key generation

| Key type and key length |
| --- |
| secp256r1, secp384r1, secp521r1<br>brainpool P256, brainpool P384, brainpool P512<br>Ed25519 |

**Note**: The macOS version does not support brainpool curve key generation.

### 3.6.2.2 When RSA Public Key Selected in [Key Type] Tab

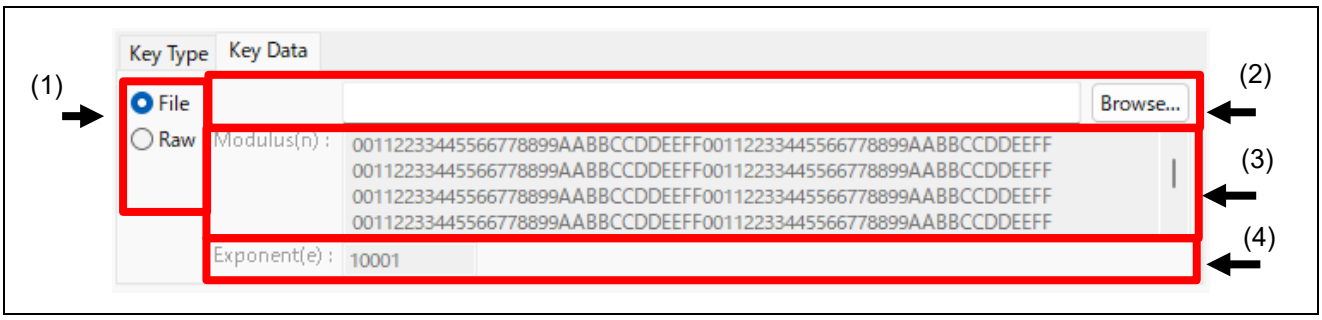


**Figure 3-9 [Key Data] Tab when RSA public key is selected**

| No | Items | Description |
|---|---|---|
| (1) | Input format | Select whether to provide a key file containing the key data in (2), or to provide the raw plaintext data of the key in (3) and (4). |
| (2) | File name | When **File** is selected in (1), select the key file that contains the plaintext key. Key files must be binary files and have the extension `*.key`. |
| (3) | Modulus(n) | When **Raw** is selected in (1), enter the plaintext data for the RSA modulus n in hex format. |
| (4) | Exponent (e) | When **Raw** is selected in (1), enter the plaintext data for the RSA exponent e in hex format. |

### 3.6.2.3 When RSA Private Key Selected in [Key Type] Tab

**Figure 3-10 [Key Data] Tab when RSA private key is selected**

| No | Items | Description |
|---|---|---|
| (1) | Input format | Select whether to provide a key file containing the key data in (2), to provide the raw plaintext data of the key in (3) and (4), or to have the tool generate a key pair with the output file name specified in (5). |
| (2) | File name | When **File** is selected in (1), select the key file that contains the plaintext key. Key files must be binary files and have the extension `*.key`. |
| (3) | Modulus(n) | When **Raw** is selected in (1), enter the plaintext data for the RSA modulus n in hex format. |
| (4) | Decryption Exponent (d) | When **Raw** is selected in (1), enter the plaintext data for the RSA decryption exponent d in hex format. |
| (5) | Output key file | When **Random** is selected in (1), the tool will generate the key data. Specify the output file of the plaintext key data generated in the tool. The output plaintext key file can be a text file or a binary file.<br>To generate keys, select the “private” key type. Both private and public keys are generated at the same time, and the specified file name is appended with `_public` for public keys and `_private` for private keys. Injection/update files are generated only for the private key. The generated public key file can be used as input to create key injection files for the public key.<br>Note: Keys generated by this tool should be used for prototyping and test purposes only. |

### 3.6.2.4 When ECC Public Key Selected in [Key Type] Tab

**Figure 3-11 [Key Data] Tab when ECC public key is selected**

| No | Items | Description |
|---|---|---|
| (1) | Input format | Select whether to provide a key file containing the key data in (2), or to provide the raw plaintext data of the key in (3) and (4). |
| (2) | File name | When **File** is selected in (1), select the key file that contains the plaintext key. Key files must be binary files and have the extension `*.key`. |
| (3) | Qx | When **Raw** is selected in (1), enter the plaintext data of the ECC public key Qx in hex format. |
| (4) | Qy | When **Raw** is selected in (1), enter the plaintext data of the ECC public key Qy in hex format. |

### 3.6.2.5 When OEM Root public Selected in [Key Type] Tab

**Figure 3-12 [Key Data] Tab when OEM Root public key is selected**

| No | Items | Description |
|---|---|---|
| (1) | Input format | Select whether to provide a key file containing the key data in (2), or to provide the raw plaintext data of the key in (3) and (4), or to have the tool generate a key (or key pair) with the output file name specified in (5). |
| (2) | File name | When **File** is selected in (1), select the key file that contains the plaintext key. Key files must be binary files and have the extension `*.key`. |
| (3) | Qx | When **Raw** is selected in (1), enter the plaintext data of the ECC public key Qx in hex format. |
| (4) | Qy | When **Raw** is selected in (1), enter the plaintext data of the ECC public key Qy in hex format. |
| (5) | Output key file | When **Random** is selected in (1), the tool will generate the key data. Specify the output file of the plaintext key data generated in the tool. The output plaintext key file can be a text file or a binary file.<br>**Note**: The specific amount of randomness of the random values generated by this tool is not guaranteed. Keys generated by this tool should be used for prototyping and test purposes only. |

### 3.6.3 Wrapping Key

If the new key is being prepared for secure injection, it must be wrapped with the UFPK, and the W-UFPK must be provided. If the new key is being prepared for secure update, it must be wrapped with the KUK.

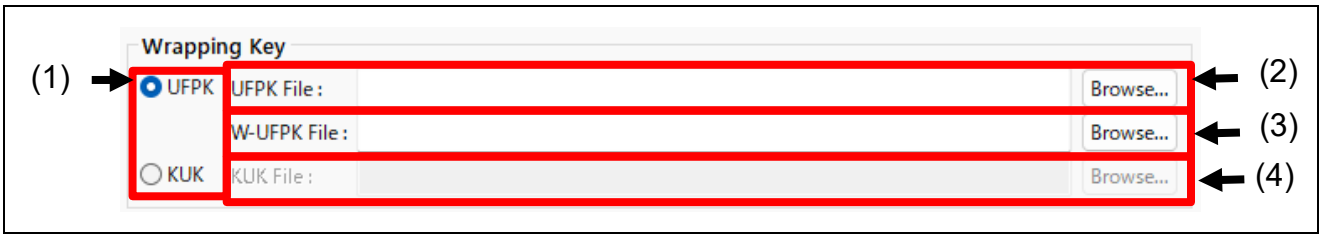


**Figure 3-13 Wrapping Key Options**

| No | Items | Description |
|---|---|---|
| (1) | Wrapping Key Type | Select the type of wrapping key. If the new key is being prepared for secure injection, select **UFPK** and provide the UFPK file in (2) and the W-UFPK file in (3). If the new key is being prepare for secure update, select **KUK** and provide the KUK file in (4). |
| (2) | UFPK File | If **UFPK** is selected in (1), enter the UFPK `*.key` file. This file is generated in the [**Generate UFPK**] tab. |
| (3) | W-UFPK File | If **UFPK** is selected in (1), enter the W-UFPK `*.key` file that corresponds to the specified UFPK. This file must be obtained from the Renesas Key Wrap service. |
| (4) | KUK File | If **KUK** is selected in (1), enter the KUK `*.key` file. This file is generated in the [**Generate KUK**] tab. |

### 3.6.4 IV

If the new key is being prepared for secure injection, it must be wrapped with the UFPK, and the W-UFPK must be provided. If the new key is being prepared for secure update, it must be wrapped with the KUK. In both cases, an Initialization Vector (IV) is required. The IV can either be specified, or the tool can generate one.

**Figure 3-14 IV Options**

| No | Items | Description |
|---|---|---|
| (1) | IV format | Select whether to use the input value from (2) for the IV value or to use a random number generated by the tool. The specific amount of randomness of the random values generated by this tool is not guaranteed. |
| (2) | Use specified value | When **Use specified value** is selected in (1), the value entered here will be used as the Initialization Vector during encryption. This value must be specified as 16 hex bytes in big-endian format. |

### 3.6.5 Output

This section specifies the type of output file to generate for the secure key injection or update. Note that not all options are supported for all MCU/MPU Families. For example, RA Family SCE9 Protected Mode key injection is supported only via a device programmer, so only the RFP output format is supported if the UFPK is selected as the wrapping key.

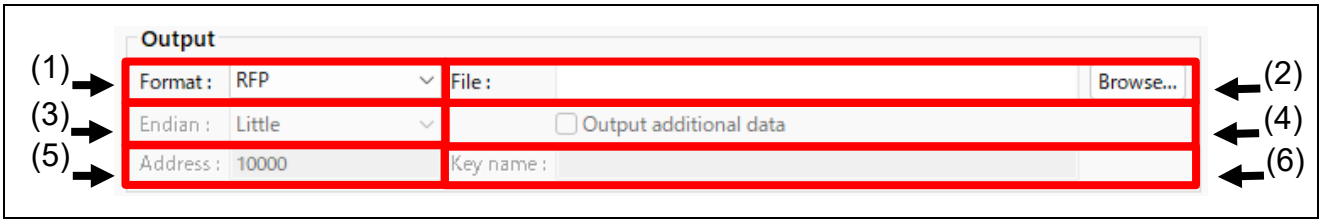


**Figure 3-15 Output Options**

| No | Items | Description |
|---|---|---|
| (1) | Format | Select the file format for output. Refer to *Table 3-8 Output file formats* for the possible formats. |
| (2) | File | Set the output file name and path. When specifying an asymmetric private key and requesting the tool to generate the key pair, the output encrypted key file name is appended with "`_private`". |
| (3) | Endian | This function is enabled when **Motorola Hex** or **Binary** is selected in (1).<br>Determines the data sequence of the output data. See *Section 4.5.6* ***bswap*** *Option* for details. |
| (4) | Output additional data | This function is enabled when **C Source**, **Motorola Hex** or **Binary** is selected in (1).<br>If the output file name is an existing file name, the key encryption information is appended to the end of the file. See *Section 4.5.5* ***fileadd*** *Option* option for details. |
| (3) | Address | This setting is enabled when **Motorola Hex** is selected in (1). Enter the address to be set in the Motorola Hex file. |
| (4) | Key name | This function is enabled when **C source** is selected in (1). This specifies the <keyname> portion of the structure, variable, and data size value defined in the C source and header files. See *Section 4.5.4.2* ***csource*** *Option for* ***filetype*** for a detailed description of how <keyname> is used.. |

Table 3-8 Output file formats

| Options | Description |
|---|---|
| C Source | Outputs the C source file and header. |
| Binary | Outputs binary data. |
| Motorola Hex | Outputs Motorola Hex file. |
| RFP | Outputs key data in Renesas Key File format. |

Table 3-9 Endian option

| Options | Description |
|---|---|
| Little | Outputs a Motorola hex file or binary data with 4 bytes swapped for the data to be output when **Big** is selected. |
| Big | Outputs Motorola hex file or binary data in byte order with data according to the formats in *Table 4-63* and *Table 4-64.* |

## 3.7 [TSIP Update] Tab

This tab is used to encrypt TSIP secure update solution user programs.

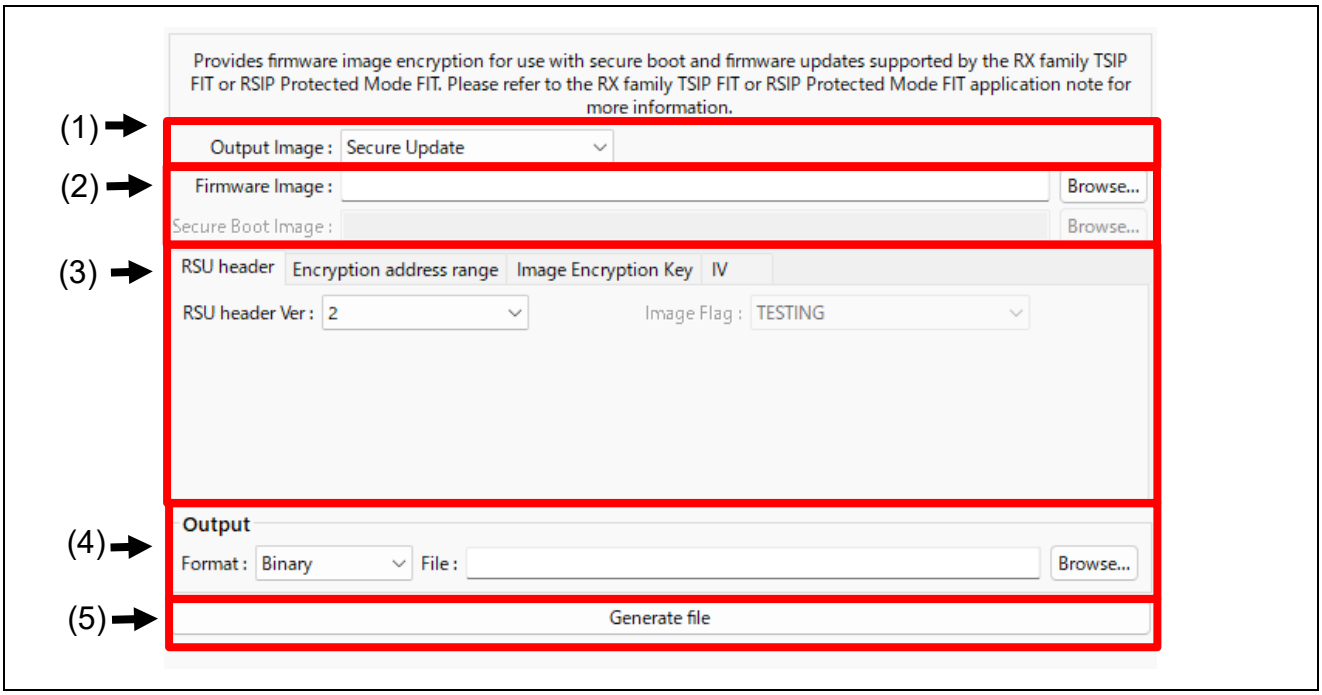


**Figure 3-16 [TSIP Update] Tab**

| No. | Items | Description |
|---|---|---|
| (1) | Output image | Select the data type to be output and specify the file to be input.<br>**Factory Programming**: Generates an image file to be used for programming at the factory. Specify files for both **Firmware image** and **Secure boot image**.<br>Secure Update: Generates an image file to be used when performing a firmware update. Specify a file for Firmware image.<br>For details, refer to *Section 3.7.1*, *Output image*. |
| (2) | Firmware image/<br>Secure boot image | Enter the program MOT file to be encrypted in the **Firmware image**. When Factory Programming is selected for Output image, for **Secure boot image** enter the secure boot program MOT file to be appended to the encrypted firmware update image file. |
| (3) | Settings tabs | Enter settings related to encryption and the output image. Refer to the following sections for the settings on each tab:<br>*3.7.3 [RSU Header] Tab*<br>*3.7.3 [Encrypted Address Range] Tab*<br>*3.7.4 [Image Encryption Key] Tab*<br>*3.7.5 [IV] Tab* |
| (4) | Output | Specify the file to be output. For details, refer to *Section 3.7.7*, *Output*. |
| (5) | **Generate file** | Click this button to generate a file in the format specified by (4). |

### 3.7.1 Output image

Specifies the data format to be output from the tab.

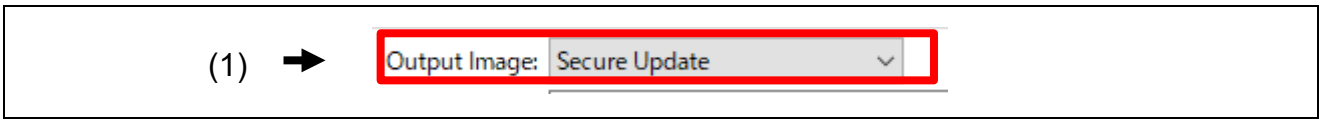


**Figure 3-17 Output image**

| No | Item | Description |
|---|---|---|
| (1) | Output image | Select the file type to be created on the tab. |

Table 3-10 Output image Option

| Options | Description |
|---|---|
| Factory Programming | Generates a MOT file containing the file specified by Secure boot image and the file specified by Firmware image as encrypted data for use for programming at the factory. |
| Secure Update | Generates an encrypted MOT file or binary file with RSU header from the specified area of the MOT file entered as Firmware image for use as a firmware update.<br>The secure boot portion is not encrypted. It is recommended that writing be done in a secure environment. |

### 3.7.2 Firmware image and Secure boot image

Specify the firmware image to be encrypted and the secure boot image to be appended to the encrypted data.

**Figure 3-18 Firmware image and Secure boot image**

| No | Items | Description |
|---|---|---|
| (1) | Firmware Image | Specify the MOT file of the program to be encrypted. The data in the area of the designated MOT file specified by Encryption Address Range is encrypted for use in a TSIP firmware update solution. For the supported cryptographic algorithms, refer to the application notes covering the TSIP. |
| (2) | Secure Boot Image | When **Factory Programming** is specified for Output image, specify the secure boot image to accompany the encrypted firmware image.<br>Secure boot images are not encrypted. |

### 3.7.3 [RSU Header] Tab

Specify the file format for outputting the RSU header in this section.

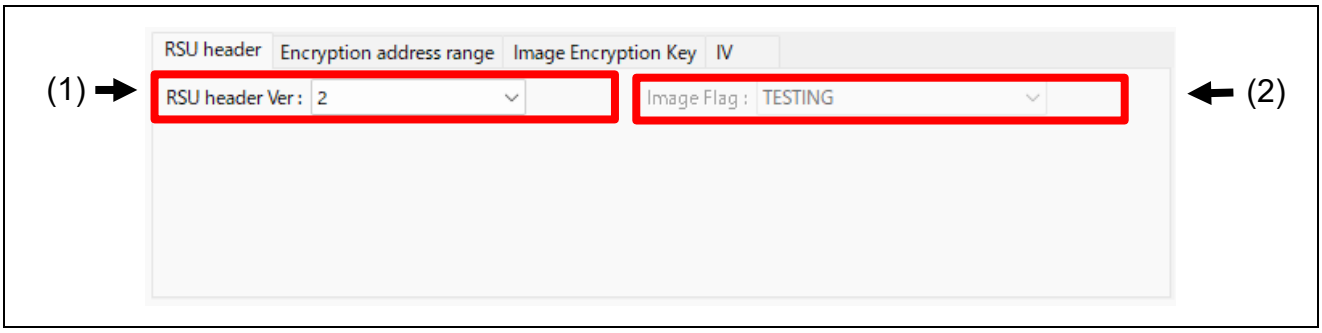


**Figure 3-19 [RSU Header] Tab**

| No | Items | Description |
|---|---|---|
| (1) | RSU header Ver | Specify the RSU header version number to be appended to the encrypted firmware image. Version can be 1 or 2.For information on RSU header versions, refer to *Section 4.6.1* ***ver*** *Option*. |
| (2) | Image Flags | Select the value to be set for the Image Flags RSU header. If **2** is selected in (2), Image Flags cannot be specified because it is not required. Refer to *Table 3-11* for the available setting values. |

Table 3-11 RSU Header Image Flag Values

| Options | Description |
|---|---|
| BLANK | No firmware update image present. |
| TESTING | Firmware update image being updated, tested. |
| INSTALLING | Firmware update image is initial image being installed, tested. |
| VALID | Firmware update image is valid. |
| INVALID | Firmware update image is invalid. |
| END_OF_LIFE | Firmware update image has reached end of life. |

### 3.7.4 [Encryption Address Range] Tab

Specifies the area of the firmware image to be encrypted.

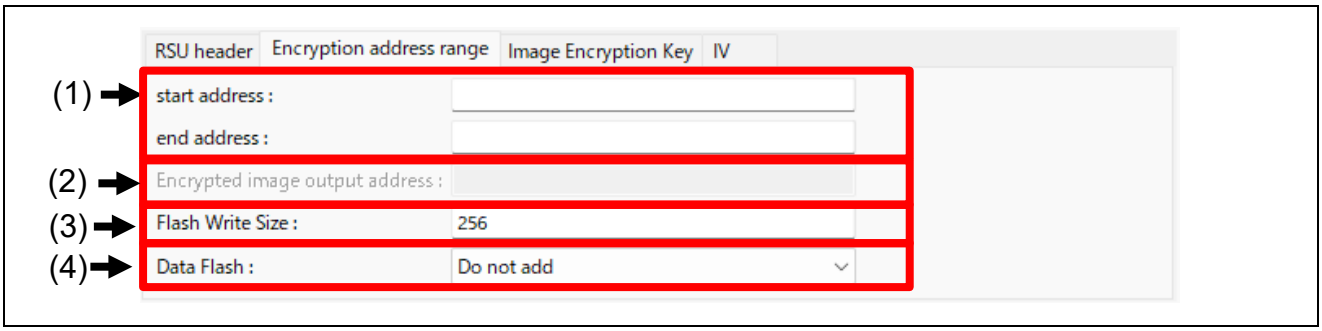


**Figure 3-20 [Encrypted Address Range] Tab**

| No | Items | Description |
|---|---|---|
| (1) | start address/<br>end address | Specify the area of the MOT file specified by Firmware image to be encrypted.<br>The maximum encryption range is 8MB. |
| (2) | Encrypted image output address | If MOT format is specified for the output file, specify the address to which to output the encrypted firmware image. |
| (3) | Flash Write Size | Specify the Flash write unit.<br>The size should be specified in units of 16 bytes. |
| (4) | Data Flash | It is possible to specify whether or not the data flash data included in the firmware image is appended to the output file.<br>See *Table 3-12 Data Flash Settings* for details. |

Table 3-12 Data Flash Settings

| Options | Description |
|---|---|
| Add | Firmware image data flash data is added to the output file. Output image:<br>Selectable when **Factory Programming** is selected. |
| Encrypt and append | Encrypts and appends the firmware image data flash data to the output file.<br>selectable only for RSU header Ver2. |
| Do not add | No firmware image data flash data is appended to the output file. |

### 3.7.5 [Image Encryption Key] Tab

Specify the key data to be used to encrypt the firmware image.

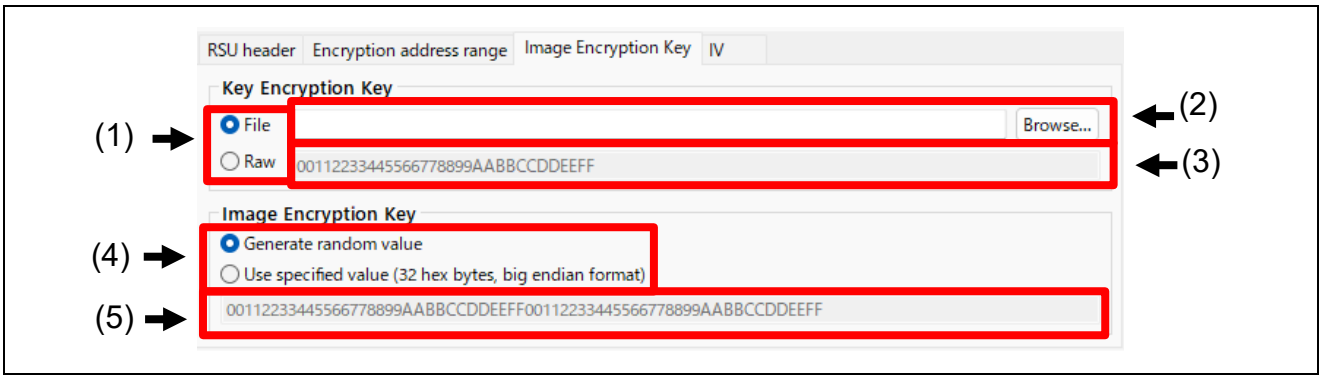


**Figure 3-21 [Image Encryption Key] Tab**

| No | Items | Description |
|---|---|---|
| (1) | Key Encryption Key | Specify the key used to wrap the firmware image encryption key (Image Encryption Key).<br>**File** : Specify the key file in (2).<br>**Raw** : The hex data entered in (3) is used as the key. |
| (2) | File | When you select **File** in (1), specify the key file for Key Encryption Key. You can specify binary files with the `*.key` extension. |
| (3) | Raw | When you select **Raw** in (1), enter the Key Encryption Key in hexadecimal, big-endian format. |
| (4) | Format of Image Encryption Key | Select the key value to be used to encrypt the firmware image.<br>**Use random number generator**: The Security Key Management Tool's random number generator function is used to generate the key.<br>**Use specified value**: The hex data entered in (5) is used to generate the key. |
| (5) | Use specified value | When **Use specified value** is selected for (4), the value entered here is used as the encryption key. Enter a value in hexadecimal big-endian format. |

### 3.7.6 [IV] Tab

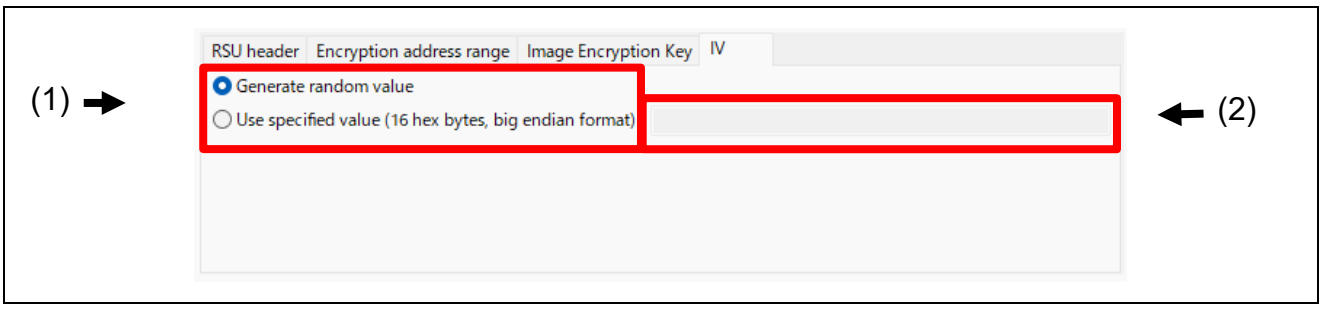


**Figure 3-22 [IV] Tab**

| No | Items | Description |
|---|---|---|
| (1) | Format of initialization vector | Select the initialization vector value to be used when encrypting the firmware image.<br>**Use random number generator**: The Security Key Management Tool's random number generator function is used to generate the initialization vector.<br>**Use specified value**: The hex data entered in (2) is used to generate the initialization vector. |
| (2) | Use specified value | When **Use specified value** is selected for (1), the value entered here is used as the initialization vector when encrypting the firmware image. Enter a value in hexadecimal big-endian format. |

### 3.7.7 Output

Specify the output file format in this section.

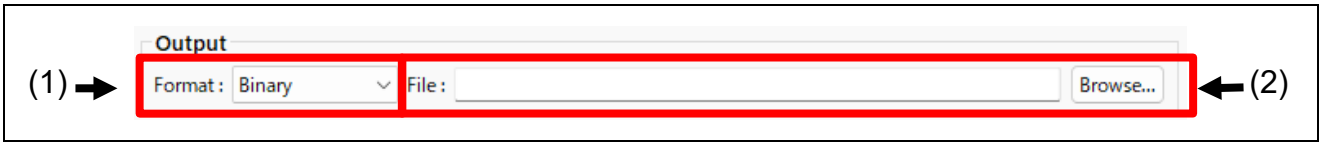


**Figure 3-23 Output**

| No | Items | Description |
|---|---|---|
| (1) | Format | Select the output file format. Refer to *Table 3-13* for the available formats. |
| (2) | File | Specify the output file name and path. |

Table 3-13 Available Formats

| Options | Description |
|---|---|
| Binary | Outputs binary data with an RSU file header appended. RSU can be specified only when **Secure Update** is selected for **Output image**. |
| Motorola hex | Outputs a Motorola hex (MOT) file. |

## 3.8 [FSBL] tab

This tab is used to generate Key Certificates and Code Certificates that can be used in Renesas devices with First Stage Bootloader (FSBL) functionality.

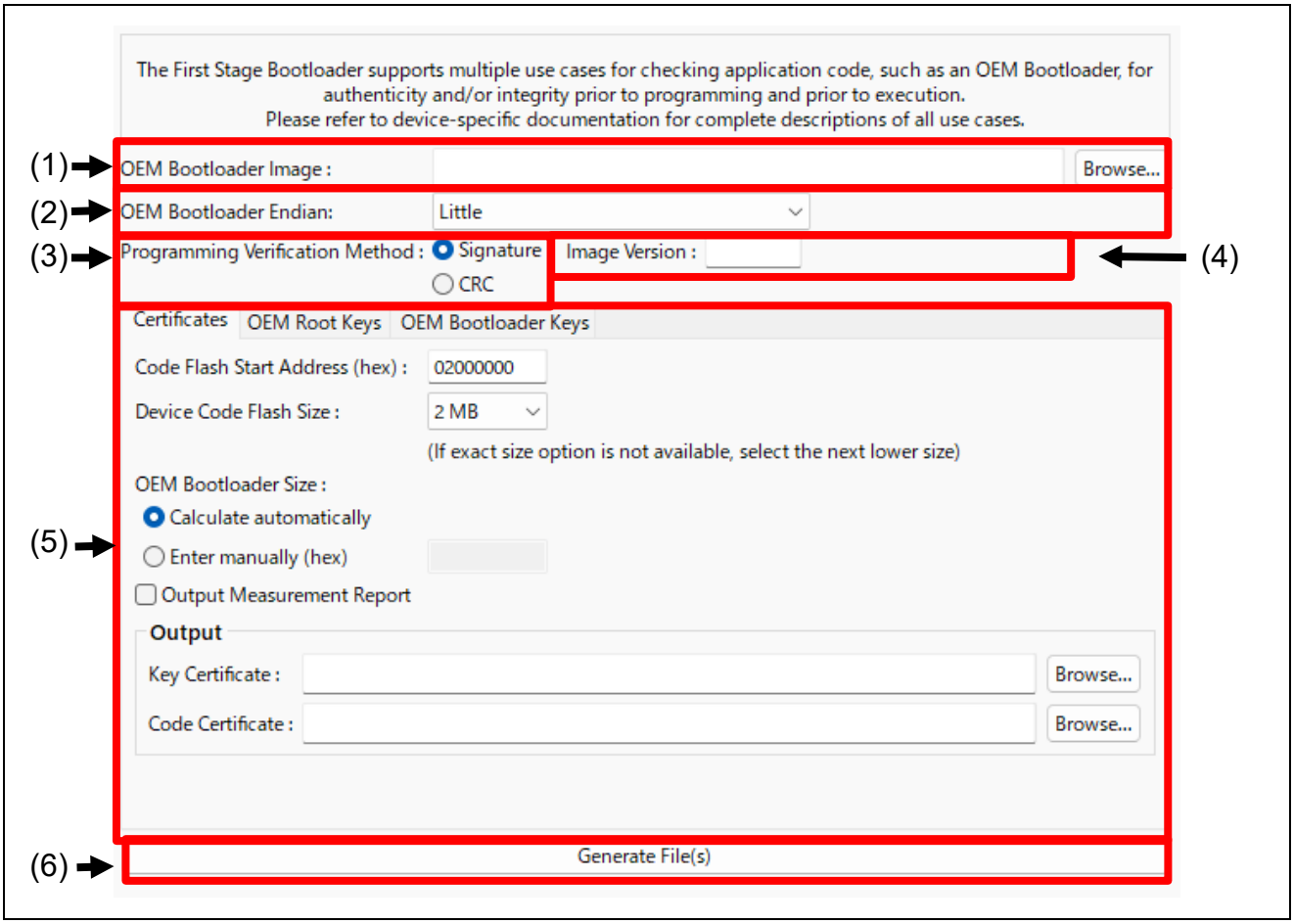


**Figure 3-24 [FSBL] Tab**

| No. | Items | Description |
|---|---|---|
| (1) | OEM Bootloader Image | Specify the Motorola hex file of the OEM Bootloader to be verified by the FSBL. |
| (2) | OEM Bootloader Endian | Select the endianness (Little or Big) of the OEM_BL to be entered. |
| (3) | Programming Verification Method | Signature: Generate Code Certificate and Key Certificate with ECDSA signature.<br>CRC: Create only Code Certificate for simple verification using CRC.<br>For details, refer to *Section 3.8.1* ***Programming Verification Method***. |
| (4) | Image version | When selecting **Signature** in the Programming Verification Method, specify the version of the OEM Bootloader. This version is included in the Code Certificate. |
| (5) | [**Certificate**]<br>[**OEM Root Keys**]<br>[**OEM Bootloader Keys**]<br>tab | [**Certificate**] tab: Specify OEM Bootloader information and output file names of the Key Certificate and the Code Certificate.<br>[**OEM Root Keys**] tab and [**OEM Bootloader Keys**] tab:<br>Key information is required if **Signature** is selected for **Programming Verification Method**.<br>For details, refer to *Section 3.8.2* ***[Certificate]*** *, 3.8.3* ***[OEM Root Keys]****,* and *3.8.4* ***[OEM Bootloader Keys]****.* |
| (6) | **Generate File(s)** | Generates the Key Certificate and Code Certificate specified in (4). |

### 3.8.1 Programming Verification Method

Specify the OEM Bootloader signing method.

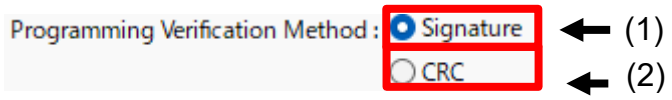


**Figure 3-25 Programming Verification Method Options**

| No | Items | Description |
|---|---|---|
| (1) | Signature | The OEM Bootloader is authenticated via a Chain of Trust. A generated Key Certificate authenticates the OEM Bootloader Key (signed by the OEM Root Private Key). A generated Code Certificate authenticates the OEM Bootloader (signed by the OEM Bootloader Private Key).<br>When **Signature** is selected, the OEM Root Key pair must be entered on the [**OEM Root Keys**] tab, and the OEM Bootloader Key pair must be entered on the [**OEM Bootloader Keys**] tab.<br>For the structure of the certificates, please refer to the Hardware Manual of the device that implements the FSBL function. |
| (2) | CRC | The integrity of the OEM Bootloader is checked using a CRC. When **CRC** is selected, the tool calculates the CRC value of the OEM Bootloader and generates only a Code Certificate. *Note |

Note: When Code Certificate is generated with **CRC** specified, the CRC value is output as a dummy value for Signer ID. If you select "CRC + report measurement" in FSBL operation mode and want to output a measurement report using the OEM Bootloader Key, select **Signature** to generate a Code Certificate.

### 3.8.2 [Certificate] tab

Specify the OEM Bootloader information and the file names of the Code Certificate and the Key Certificate.

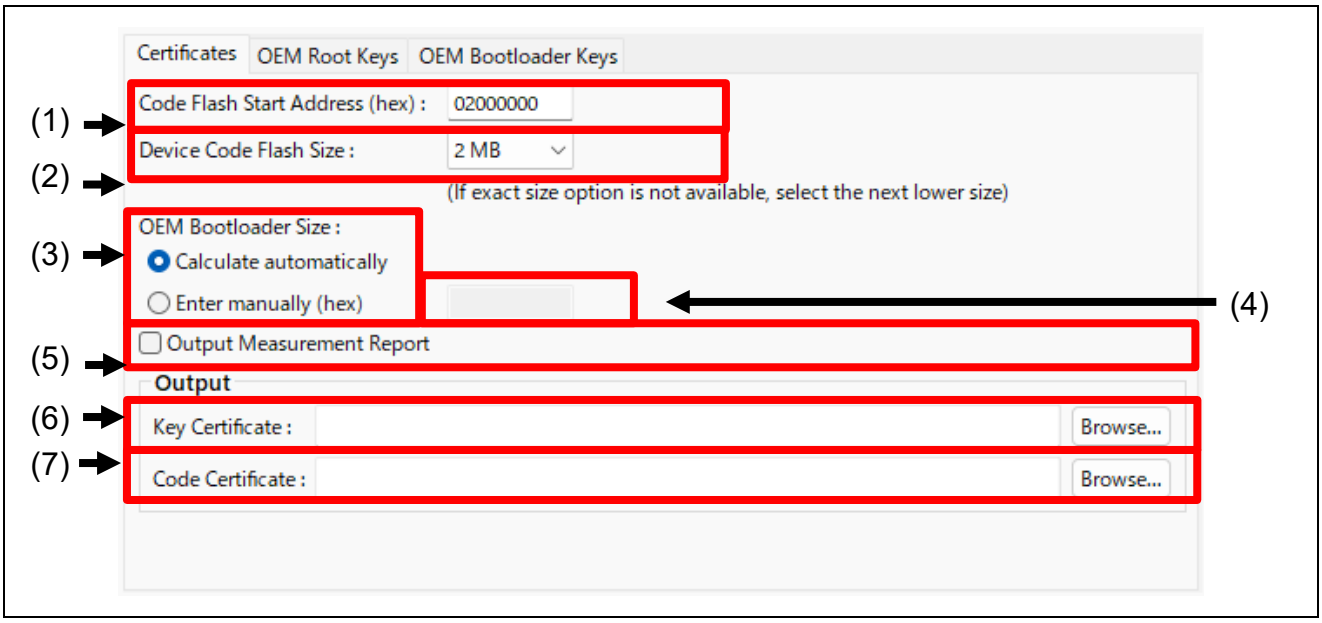


**Figure 3-26 [Certificate] Tab**

| No | Items | Description |
|---|---|---|
| (1) | Code Flash Start Address | Specify the starting address of the OEM Bootloader. |
| (2) | Device Code Flash Size | Specify the Code Flash size of the device to be used.<br>If there is no applicable size, specify a size close to it. |
| (3) | OEM Bootloader Size | Set the method used to calculate the OEM Bootloader size.<br>Calculate automatically: The size of the OEM Bootloader is calculated from the mot file and the specified **Device Code Flash Size**.<br>Enter manually: The value entered is the OEM Bootloader size in (4).<br>Please refer to *Section 4.7.3* ***Area subject to OEM Bootloader signature or CRC calculation***. |
| (4) | OEM Bootloader size value | Enter the OEM Bootloader size in bytes when **Enter Manually** is selected in (3).<br>Enter a size -1 value where size is a multiple of 16. |
| (5) | Output Measurement Report | If checked, Measurement Report is output to the status window upon successful key certificate and code certificate generation. |
| (6) | Key Certificate | Specify the output file name of the Key Certificate.<br>Enter this when **Signature** is selected under **Programming Verification Method**. |
| (7) | Code Certificate | Specify the output file name of the Code Certificate. |

### 3.8.3 [OEM Root Keys] tab

Enter the OEM Root Key Pair when **Signature** is selected in **Programming Verification Method**. Confirm the ECC curve that is used by the selected MCU/MPU.

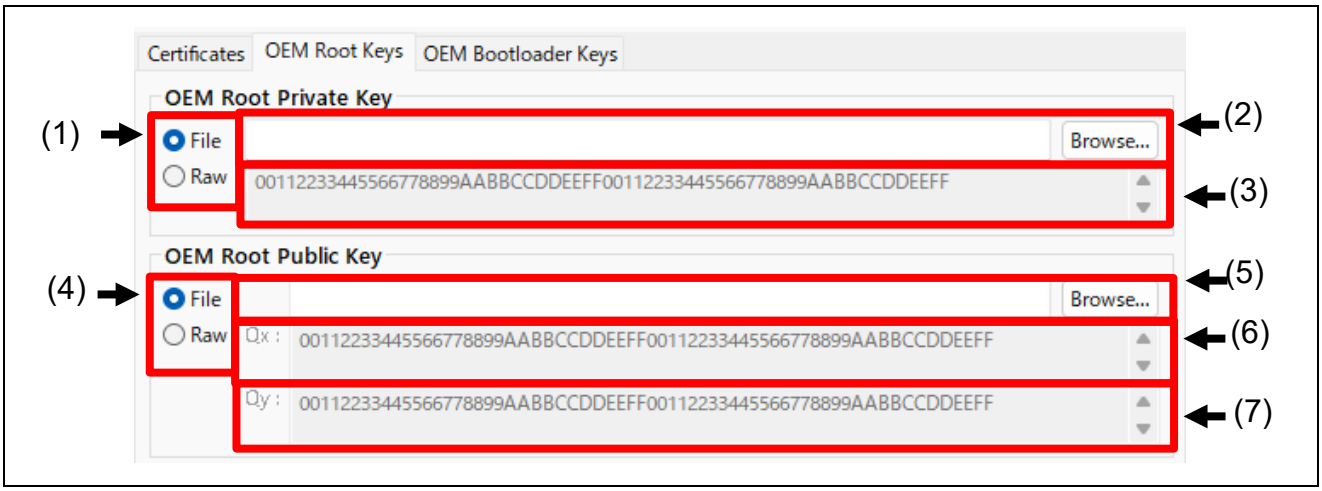


**Figure 3-27 [OEM Root Keys] Tab**

| No | Items | Description |
|---|---|---|
| (1) | Input format of OEM Root Private Key | Select whether to provide a key file containing the key data in (2) or to provide the raw plaintext data of the key in (3). |
| (2) | File name | When **File** is selected in (1), select the key file that contains the plaintext key. The file can be a binary file with a `*.key` extension, a text file with a `*.txt` extension, or a PEM file with a `*.pem` extension.<br>Note that PEM files also contain public key information. When a PEM file is specified, the **OEM Root Public Key** is not required. |
| (3) | Raw data | When **Raw** is selected in (1), enter the plain text key data in hexadecimal.<br>If the OEM Root Private Key is secp256r1, enter 32 bytes of data. |
| (4) | Input format of OEM Root Public Key | Select whether to provide a key file containing the key data in (5) or to provide the raw plaintext data of the key in (6) and (7). |
| (5) | File name | When **File** is selected in (4), select the key file that contains the plaintext key. The file can be a binary file with a `*.key` extension or a text file with a `*.txt` extension. Note that if a PEM file was specified for the private key, that file also contains the public key. |
| (6) | Qx | When **Raw** is selected in (4), enter the plaintext data of the OEM Root public key Qx in hex format. |
| (7) | Qy | When **Raw** is selected in (4), enter the plaintext data of the OEM Root public key Qy in hex format. |

### 3.8.4 [OEM Bootloader Keys] tab

Enter the OEM Bootloader Key Pair when **Signature** is selected under **Programming Verification Method**.

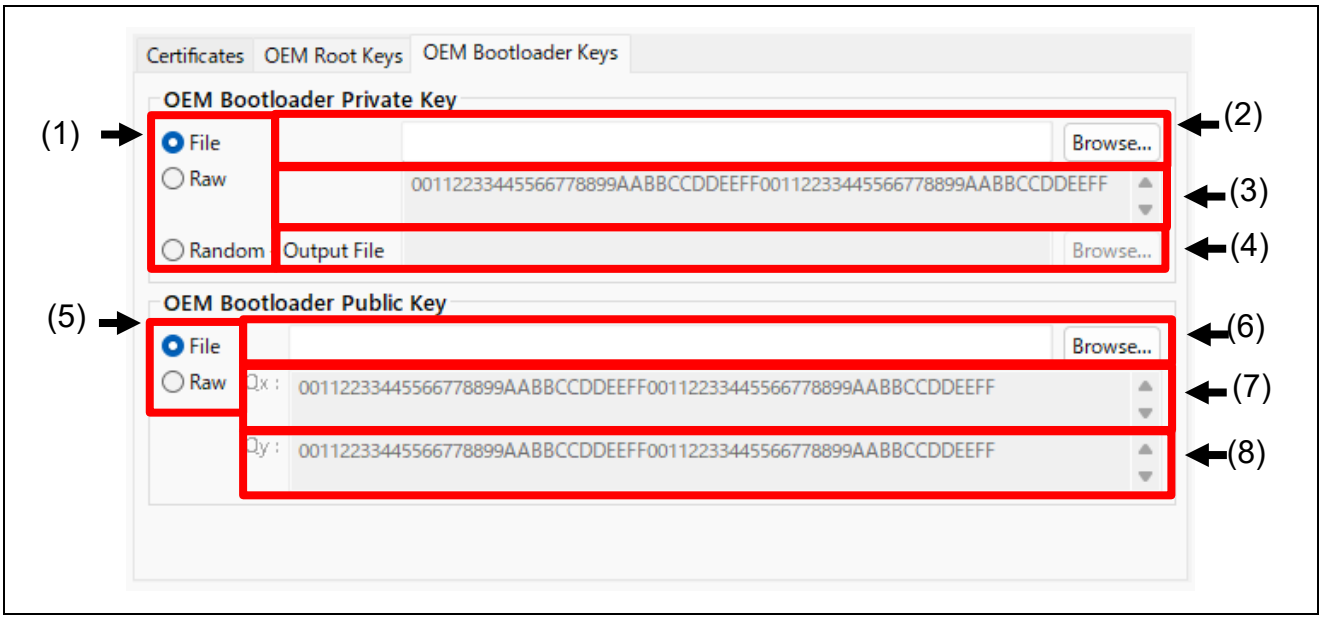


**Figure 3-28 [OEM Bootloader Keys] Tab**

| No | Items | Description |
|---|---|---|
| (1) | Input format of OEM Bootloader Private Key | Select whether to provide a key file containing the key data in (2), to provide the raw plaintext data of the key in (3), or to have the tool generate a key pair with the output file name specified in (4). |
| (2) | File name | When **File** is selected in (1), select the key file that contains the plaintext key. The file can be a binary file with a `*.key` extension, a text file with a `*.txt` extension, or a PEM file with a `*.pem` extension.<br>Note that PEM files also contain public key information. When a PEM file is specified, the **OEM Bootloader Public Key** is not required. |
| (3) | Raw data | When **Raw** is selected in (1), enter the plaintext key data in hexadecimal.<br>For example, if the OEM Bootloader uses the secp256r1 ECC curve, enter 32 bytes of data. |
| (4) | Output file name | When **Random** is selected in (1), the tool will generate the key data. Specify the output file of the plaintext key data generated by the tool. The output plaintext key file can be a text file (`*.txt`) or a binary file (`*.key`).<br>Both private and public keys are generated at the same time. The specified file name is appended with `_public` for public keys and `_private` for private keys.<br>Injection/update files are generated only for the private key. The generated public key file can be used as input to create key injection files for the public key.<br>**Note**: The specific amount of randomness of the random values generated by this tool is not guaranteed. Keys generated by this tool should be used for prototyping and test purposes only |

| No | Items | Description |
|---|---|---|
| (5) | Input format of OEM Bootloader Public Key | Select whether to provide a key file containing the key data in (6), or to provide the raw plaintext data of the key in (7) and (8). |
| (6) | File name | When **File** is selected in (5), select the key file that contains the plaintext key. The file can be a binary file with a `*.key` extension or a text file with a `*.txt` extension. Note that if a PEM file was specified for the private key, that file also contains the public key. |
| (7) | Qx | When **Raw** is selected in (5), enter the plaintext data of the OEM Bootloader public key Qx in hex format. |
| (8) | Qy | When **Raw** is selected in (5), enter the plaintext data of the OEM Bootloader public key Qy in hex format. |

## 3.9 [DOTF/OTFD] tab

This tab is used to encrypt user data (executable binary code or application data) that can be used with Renesas devices equipped with the DOTF feature.

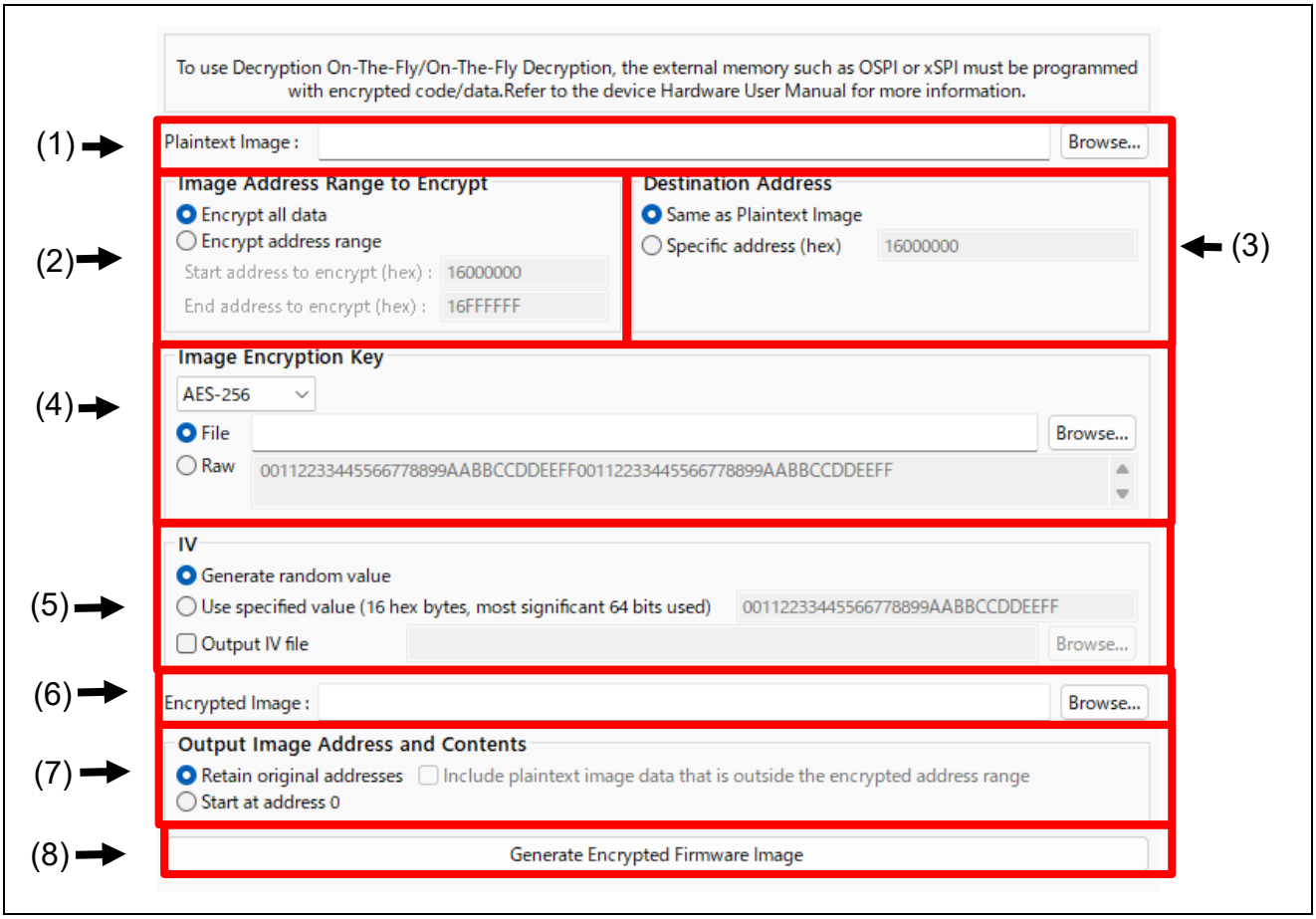


**Figure 3-29 [DOTF/OTFD] Tab**

| No | Items | Description |
|---|---|---|
| (1) | Plaintext Image | Specifies the Motorola Hex file containing the image to be encrypted. |
| (2) | Image Address Range to Encrypt | Specifies the range to encrypt within the file specified in (1).<br>Please refer to *Section 3.9.1* ***Image Address Range to Encrypt.*** |
| (3) | Destination Address | Specifies the address where data will be placed and executed.<br>Please refer to *Section 3.9.2* ***Destination Address****.* |
| (4) | Image Encryption Key | Specifies the key used for encryption.<br>Please refer to *Section 3.9.3* ***Image Encryption Key****.* |
| (5) | IV | Specifies the nonce value to be used for encryption.<br>Please refer to *Section 3.9.4* ***IV****.* |
| (6) | Encrypted Image | Specifies the output file name of the encrypted data. |
| (7) | Output Image Address and Contents | Specifies options for the output file.<br>Please refer to *Section 3.9.5* ***Output Image Address and Contents****.* |
| (8) | **Generate Encrypted Firmware Image** | Generates the encrypted image file specified in (6). |

### 3.9.1 Image Address Range to Encrypt

Specifies the range of encryption within the input plaintext image file.

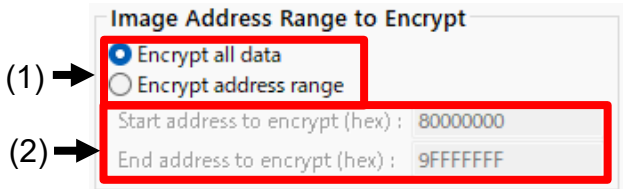


**Figure 3-30 Image Address Range to Encrypt Options**

| No | Items | Description |
|---|---|---|
| (1) | Encryption range selection | **Encrypt all data**: Encrypts all data in files entered in **Plaintext Image**. *Note1, *Note2<br>**Encrypt address range**: Encrypts the range specified in (2) within the file entered in **Plaintext Image**. *Note2 |
| (2) | Encryption range | Enter the address to start encryption in (1) and the address to end encryption in (2). The address range must be 16-byte aligned. |

Note1: If the start and end addresses of the input file are not 16-byte aligned, the input data will be padded with 0x00 for 16-byte alignment.

Note2: Any data gaps within the encryption address range will be filled with 0x00.

### 3.9.2 Destination Address

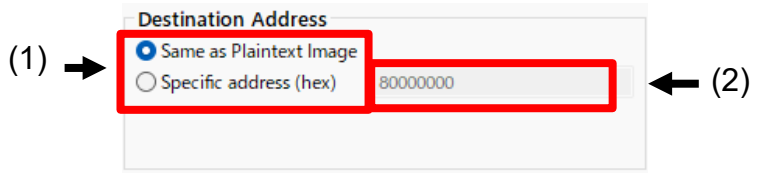


**Figure 3-31 Destination Address Options**

| No | Items | Description |
|---|---|---|
| (1) | Select Destination address | **Same as Plaintext Image**: The address entered in the plaintext image is used as the destination address.<br>**Specific address**: The address specified in (2) is used as the destination address. |
| (2) | Specific address | When **Specific address** is selected in (1), specify the address where the encrypted data will be executed/read. |

### 3.9.3 Image Encryption Key

Specify the key data to be used when encrypting the firmware image.

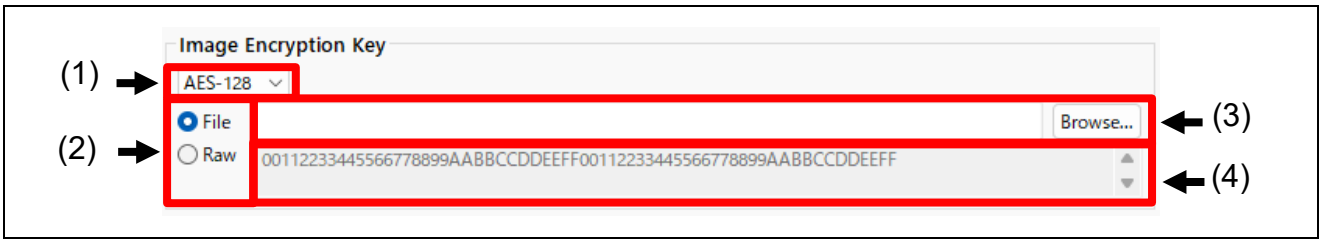


**Figure 3-32 Image Encryption Key Options**

| No | Items | Description |
|---|---|---|
| (1) | Key length specification | Specify the key length to be used for encryption. Specify from **AES-128**, **AES-192**, or **AES-256**. Please refer to the MCU / MPU Hardware Manual to determine which key lengths are supported by MCU / MPU. |
| (2) | Input format | Select whether to provide a key file containing the key data in (3), or to provide the raw plaintext data of the key in (4). |
| (3) | File name | When **File** is selected in (2), select the key file that contains the plaintext key. Key files must be binary files and have the extension `*.key` or be text files and have the extension `*.txt`. |
| (4) | Raw data | When **Raw** is selected in (2), enter the plaintext key data in hex format. |

### 3.9.4 IV

Specify the IV (Counter value) to be used when encrypting the firmware image.

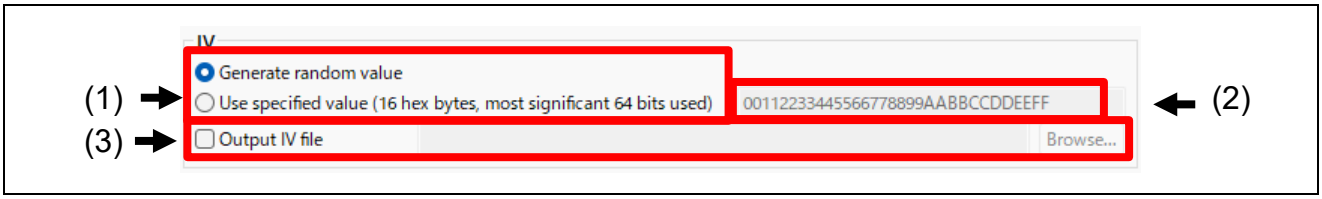


**Figure 3-33 IV Options**

| No | Items | Description |
|---|---|---|
| (1) | Input format | Select whether to use the input value from (2) for the IV value or to use a random number generated by the tool. The specific amount of randomness of the random values generated by this tool is not guaranteed. |
| (2) | Use specified value | When **Use specified value** is selected in (1), the upper 100 bits or 64bits of the value entered here will be used as the Initialization Vector during encryption. This value must be specified as 16 hex bytes in big-endian format. |
| (3) | Output IV file | Outputs the IV used during selection and encryption to the specified file.<br>For the output file format, see 4.8.4 **iv_tlv_output** option. |

### 3.9.5 Output Image Address and Contents

**Figure 3-34 Output Image Address and Contents Options**

| No | Items | Description |
|---|---|---|
| (1) | Specify output file address | Specifies the address of the output image file.<br>When **Retain original addresses** is selected, the starting address of the output file is set to the same address as the input file.<br>When **Start at address 0** is selected, the start address of the output file is set to 0. |
| (2) | Include Plaintext image | This option can be selected if **Retain original addresses** is selected in (1). If the check box is checked, data in the input file that is outside the encryption range will be included in the output file as plain text data. Otherwise, only the encrypted data within the specified address range will be included in the output file. |

## 3.10 [SFP] tab

This tab prepares the program and parameter information required by the Secure Factory Programming function and outputs it to a file.

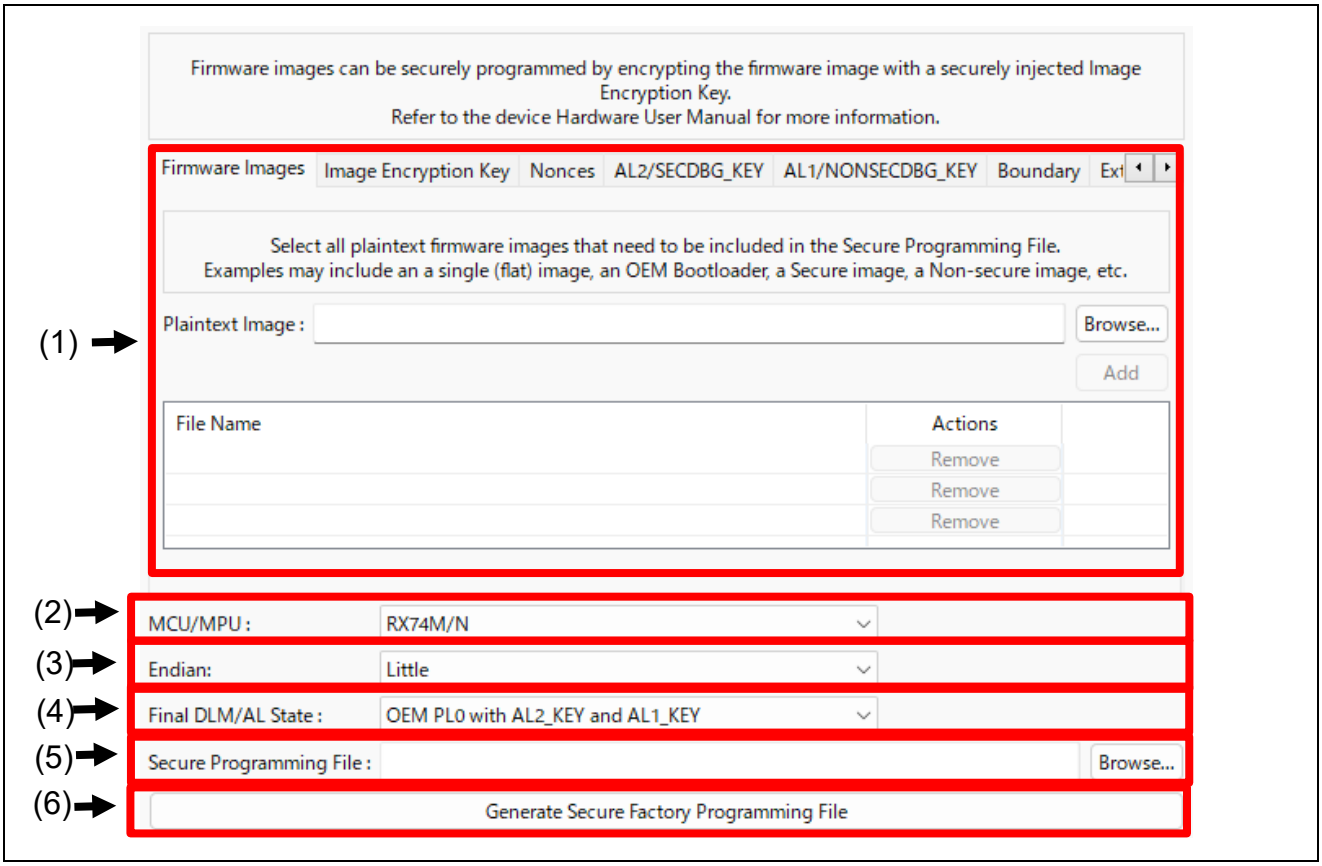


**Figure** 3-35 **[SFP] Tab**

| No | Items | Description |
|---|---|---|
| (1) | [**Firmware Images**]<br>[**Image Encryption Key**]<br>[**Nonces**]<br>[**AL2/SECDBG_KEY**]<br>[**AL1/NONSECDBG_KEY**]<br>[**Boundary**]<br>[**External Flash Memory Area**] tab | Enter the various data used to generate the Secure Factory Programming file.<br>For more information on each tab, see *Section 3.10.1[Firmware **Images]*** [Firmware Images] tab, *3.10.2* [Image Encryption Key] tab, *3.10.3* [Nonces] tab, *3.10.4* [AL2/SECDBG_KEY] tab, *3.10.5* [AL1/NONSECDBG_KEY] tab*, 3.10.6* [Boundary] tab*, 3.10.7* [External Flash Memory Area] tab. |
| (2) | MCU/MPU | Select the MCU/MPU that will use Secure Factory Programming. The MPU/MPU that can be used is determined by the cryptographic engine selected in the Overview tab. Please refer to *Table 3-14 MCU/MPU* for the MPU/MPU and cryptographic engines that can be selected. |
| (3) | Endian | Select the endianness (Little or Big) of the firmware image to be encrypted. |
| (4) | Final DLM/AL State | Specify the DLM transition destination information.<br>See *Table 3-15* for details on the available settings. |
| (5) | Secure Programming File | Specify the output path and name of the Secure Factory Programming file. |
| (6) | **Generate Secure Factory Programming File** | Generate the Secure Factory Programming file. |

Table 3-14 MCU/MPU

| Options | Overview tab | Final DLM/AL State | Description |
|---|---|---|---|
| RA8D1/M1/T1 | RA Family, RSIP-E51A Security functions and Protected Mode | - OEM PL0 with AL2_KEY<br>- LCK_BOOT” | The specified image is encrypted in the Secure Factory Programming format for RA8D1, RA8M1, and RA8T1. |
| RA8D2/M2/T2/P1 | RA Family, RSIP-E50D Security functions and Protected Mode | - OEM PL0 with AL2_KEY and AL1_KEY<br>- OEM PL0 with AL2_KEY<br>- LCK_BOOT” | The specified image is encrypted in the Secure Factory Programming format for RA8D2, RA8M2, RA8T2 and RA8P1. |
| RA8E1/E2 | RA Family, RSIP-E51A Security functions and Protected Mode | - OEM PL0 with AL2_KEY<br>- LCK_BOOT” | The specified image is encrypted in the Secure Factory Programming format for RA8E1 and RA8E2. |
| RA6B1/RA6U1 | RA Family, RSIP-E31B Security functions and Protected Mode | - OEM PL0 with AL2_KEY and AL1_KEY<br>- OEM PL0 with AL2_KEY<br>- LCK_BOOT” | The specified image is encrypted in the Secure Factory Programming format for RA6B1/RA6U1. |
| RA4C1 | RA Family, RSIP-E31A Security functions and Protected Mode | - DPL with SECDBG_KEY and NONSECDBG_KEY<br>- LCK_BOOT | The specified image is encrypted in the Secure Factory Programming format for RA4C1.<br>Be sure to set the [**Boundary**] tab. |
| RA4L1 | RA Family, RSIP-E11A Security functions and Protected Mode | - DPL with SECDBG_KEY and NONSECDBG_KEY<br>- LCK_BOOT | The specified image is encrypted in the Secure Factory Programming format for RA4L1.<br>Be sure to set the [**Boundary**] tab. |
| RX74M/N | RX Family, RSIP-E50D Security functions and Protected Mode | ・OEM PL0 with AL2_KEY and AL1_KEY<br>・OEM PL0 with AL2_KEY<br>・LCK_BOOT | The specified image is encrypted in the Secure Factory Programming format for RX74M and RX74T. |
| Not supported | A cryptographic engine other than the above was selected. | | A cryptographic engine that does not support the Secure Factory Programming function has been selected in the Overview tab. |

Table 3-15 Final DLM/AL State

| Options | Description |
|---|---|
| OEM PL0 with AL2_KEY and AL1_KEY | Transfer DLM to OEM and set protect mode to PL0 and write only AL2_Key and AL1_KEY. |
| OEM PL0 with AL2_KEY | Transfer DLM to OEM and set protect mode to PL0 and write only AL2_Key.<br>The AL1/NONSECDBG_KEY tab does not need to be filled in. |
| DPL with SECDBG_KEY and NONSECDBG_KEY | Transfer DLM to DPL and write SECDBG_KEY and NONSECDBG_KEY. |
| LCK_BOOT | Transfer DLM to LCK_BOOT.<br>The AL2/SECDBG_KEY, AL1/NONSECDBG_KEY tabs do not need to be filled in. |

**Note**: Not all MCUs/MPUs will support all available options.
Please refer to the Hardware Manual for the specific MCU/MPU for the list of supported options.

### 3.10.1 [Firmware Images] tab

Specify the firmware image file(s) to be encrypted.

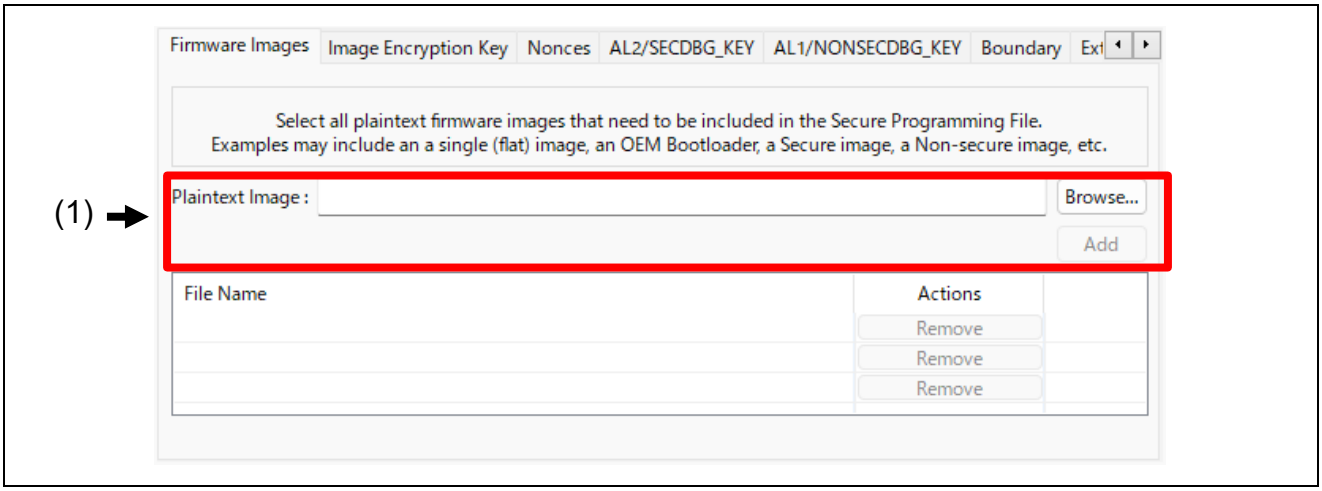


**Figure 3-36 [Firmware Images] Tab**

| No | Item | Description |
|---|---|---|
| (1) | **Plaintext Image** | Specify the user program(s) to be encrypted.<br>Select a file with the **Browse** button and add it to the list with the **Add** button.<br>Use the **Remove** action to delete a file. Up to three files can be specified. |

### 3.10.2 [Image Encryption Key] tab

Specify the key, IV, UFPK, and W-UFPK to be used when encrypting the Firmware Image.

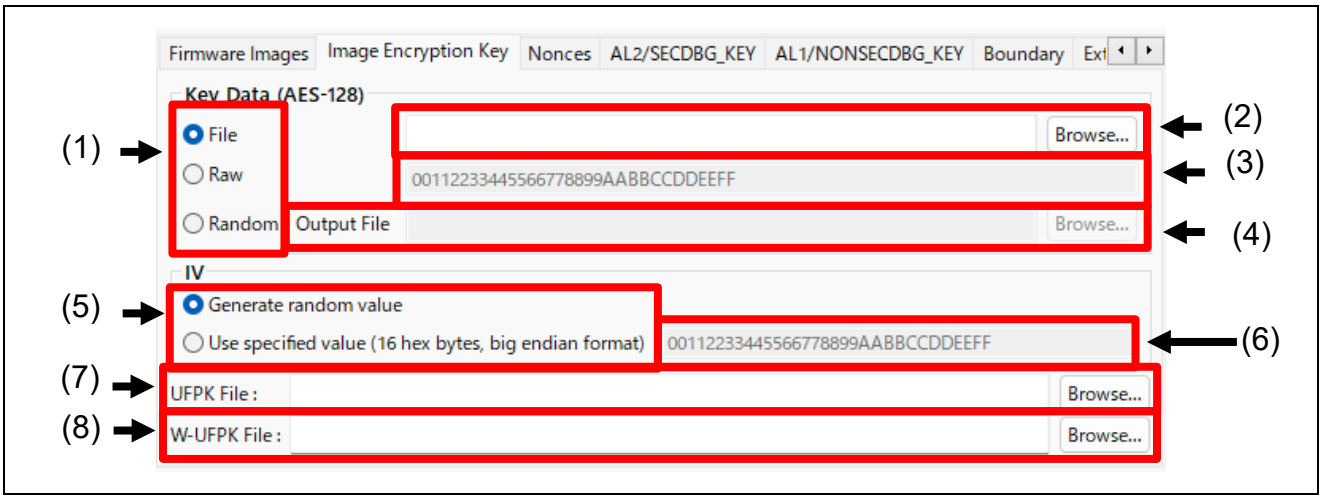


**Figure 3-37 [Image Encryption Key] Tab**

| No | Items | Description |
|---|---|---|
| (1) | Key Data Input format | Select whether to provide a key file containing the AES-128 key data in (2), to provide the raw plaintext data of the key in (3), or to have the tool generate a key with the output file name specified in (4). |
| (2) | File name | When **File** is selected in (1), select the AES-128 key file that contains the plaintext key. Key files must be binary files and have the extension `*.key`. |
| (3) | Raw data | When **Raw** is selected in (1), enter the plaintext AES-128 key data in hex format. |
| (4) | Output key file | When **Random** is selected in (1), the tool will generate the AES-128 key data. Specify the output file of the plaintext key data generated in the tool. The file can be a binary file with a `*.key` extension or a text file with a `*.txt` extension. **Note**: The specific amount of randomness of the random values generated by this tool is not guaranteed. Keys generated by this tool should be used for prototyping and test purposes only. |
| (5) | IV format | Select whether to use the input value from (6) for the IV value or to use a random number generated by the tool. The specific amount of randomness of the random values generated by this tool is not guaranteed. |
| (6) | IV value | When **Use specified value** is selected in (5), the value entered here will be used as the Initialization Vector during encryption. This value must be specified as 16 hex bytes in big-endian format. |
| (7) | UFPK File | Enter the UFPK `*.key` file. This file is generated in the [**Generate UFPK**] tab. |
| (8) | W-UFPK File | Enter the W-UFPK `*.key` file that corresponds to the specified UFPK. This file must be obtained from the Renesas Key Wrap service. |

### 3.10.3 [Nonces] tab

Specify the initialization vector values to be used for encryption of parameter information and the user program.

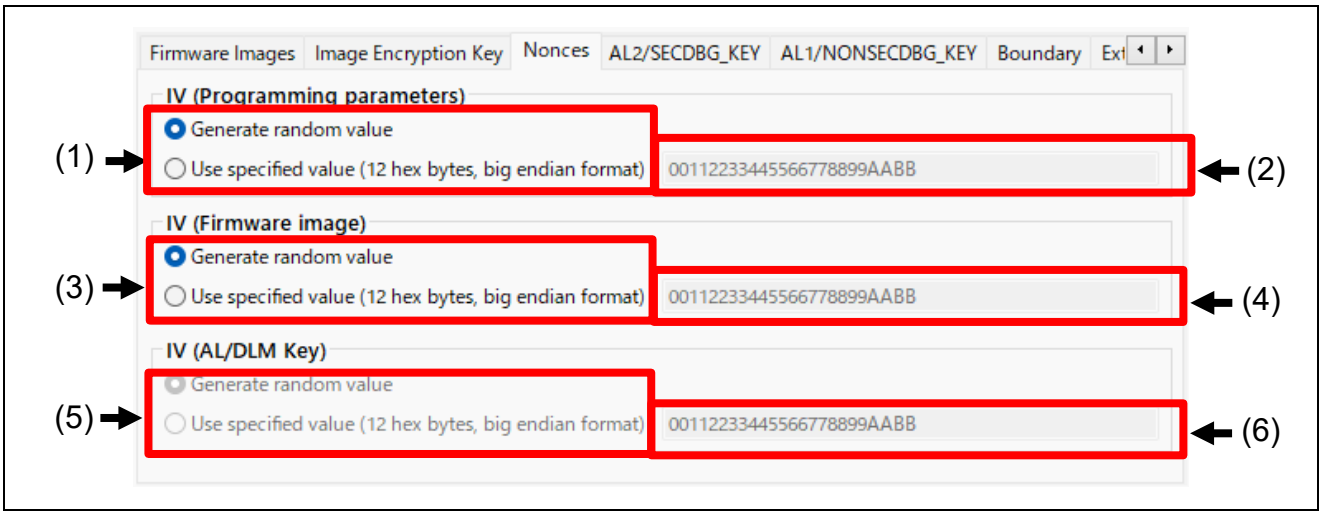


**Figure 3-38 [Nonces] Tab**

| No | Items | Description |
|---|---|---|
| (1) | IV format (Programming parameters) | Select whether to use the input value from (2) for the IV value or to use a random number generated by the tool. The specific amount of randomness of the random values generated by this tool is not guaranteed. |
| (2) | Specified value (Programming parameters) | When **Use specified value** is selected in (1), the value entered here will be used as the Initialization Vector during Programming parameters encryption. This value must be specified as 16 hex bytes in big-endian format. |
| (3) | IV format (Firmware Image) | Select whether to use the input value from (4) for the IV value or to use a random number generated by the tool. The specific amount of randomness of the random values generated by this tool is not guaranteed. |
| (4) | Specified value (Firmware Image) | When **Use specified value** is selected in (3), the value entered here will be used as the Initialization Vector during Firmware image encryption. This value must be specified as 16 hex bytes in big-endian format. |
| (5) | IV format (AL/DLM Key) | Select whether to use the input value from (6) for the IV value or to use a random number generated by the tool. The specific amount of randomness of the random values generated by this tool is not guaranteed. |
| (6) | Specified value (AL/DLM Key) | When **Use specified value** is selected in (5), the value entered here will be used as the Initialization Vector during Firmware image encryption. This value must be specified as 16 hex bytes in big-endian format. |

**Note**: The byte length of IV varies depending on the MCU/MPU. For information on which bytes are valid for each MCU/MPU, see Section 4.9.3 **nonce_trn, nonce_prg, nonce_key** option.

### 3.10.4 [AL2/SECDBG_KEY] tab

Specifies the key data to be used when encrypting the AL2_KEY or SECDBG_KEY.

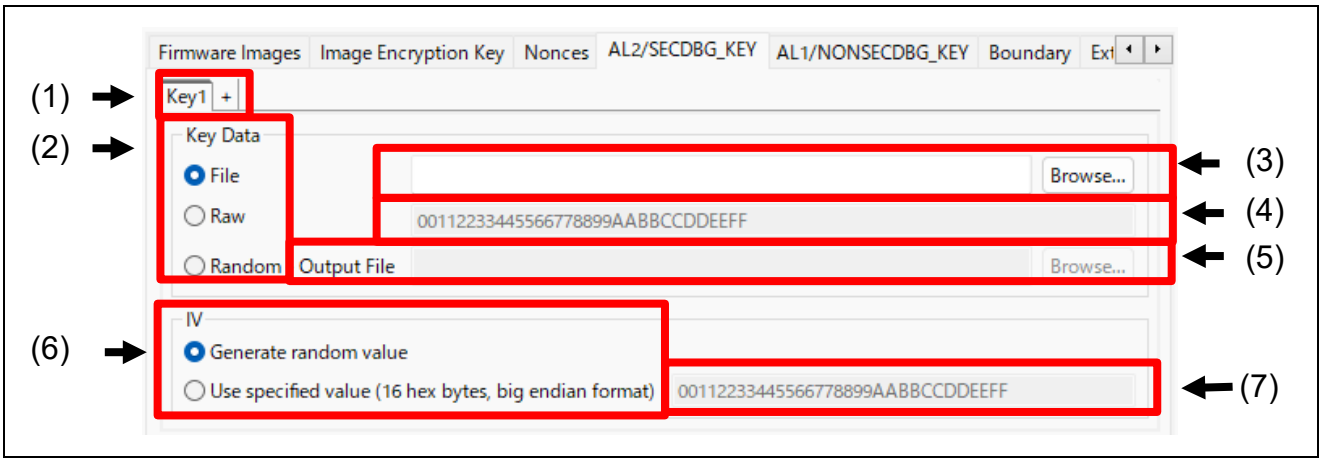


**Figure 3-39 [AL2/SECDBG_KEY] Tab**

| No | Items | Description |
|---|---|---|
| (1) | Key Tab | For MCUs/MPUs that support the injection of multiple AL2_KEY values, a "**+**" symbol will appear. Press the "**+**" button to configure multiple AL2_KEY values. Please refer to the MCU/MPU hardware manual for the maximum number of AL2_KEY values that can be injected. |
| (2) | Input format | Select whether to provide a key file containing the AL2_Key or SECDBG_KEY in (2), to provide the raw plaintext data of the key in (3), or to have the tool generate a key with the output file name specified in (4). |
| (3) | File name | When **File** is selected in (1), select the key file that contains the plaintext AL2_Key. Key files must be binary files and have the extension `*.key`. |
| (4) | Raw data | When **Raw** is selected in (1), enter the AL2_Key or SECDBG_KEY plaintext key data in hex format. |
| (5) | Output key file | When **Random** is selected in (1), the tool will generate the key data. Specify the output file of the plaintext AL2_Key or SECDBG_KEY key data generated by the tool. The file can be a binary file with a `*.key` extension or a text file with a `*.txt` extension. **Note**: The specific amount of randomness of the random values generated by this tool is not guaranteed. Keys generated by this tool should be used for prototyping and test purposes only. |
| (6) | IV format | Select whether to use the input value from (6) for the IV value or to use a random number generated by the tool. The specific amount of randomness of the random values generated by this tool is not guaranteed. |
| (7) | Specified value | When **Use specified value** is selected in (5), the value entered here will be used as the Initialization Vector during AL2_Key or SECDBG_KEY encryption. This value must be specified as 16 hex bytes in big-endian format. |

### 3.10.5 [AL1/NONSECDBG_KEY] tab

Specifies the key data to be used when encrypting with AL1_KEY or NONSECDBG_KEY.

**Note**: Not all MCUs/MPUs will support this option.
Please refer to the Hardware Manual for the specific MCU/MPU for the list of supported options.

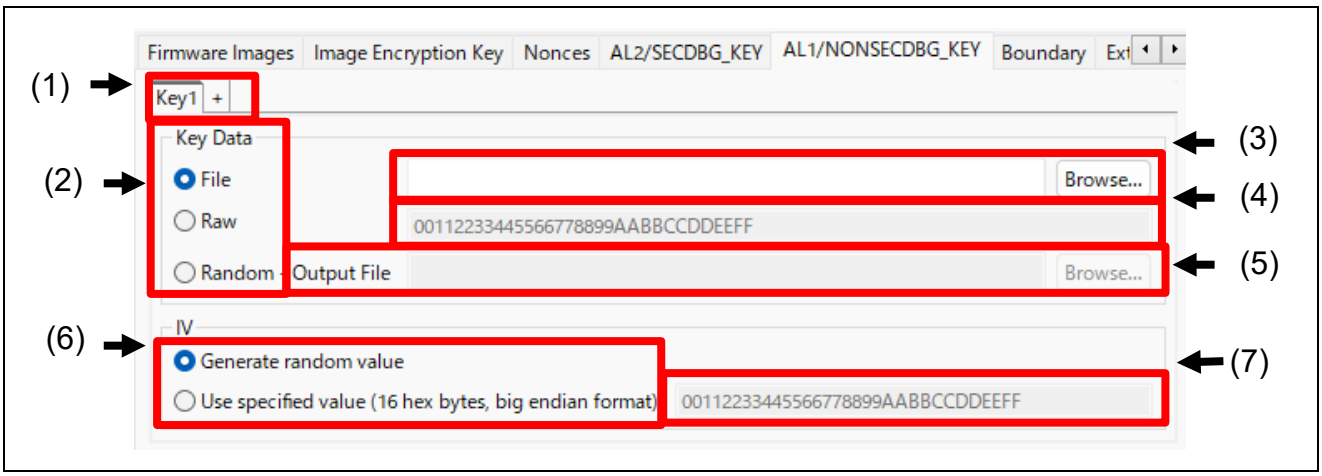


**Figure 3-40 [AL1/NONSECDBG_KEY] Tab**

| No | Items | Description |
|---|---|---|
| (1) | Key Tab | For MCUs/MPUs that support the injection of multiple AL1_KEY values, a "**+**" symbol will appear. Press the "**+**" button to configure multiple AL2_KEY values. Please refer to the MCU/MPU hardware manual for the maximum number of AL1_KEY values that can be injected. |
| (2) | Input format | Select whether to provide a key file containing the AL1_Key or NONSECDBG_KEY in (2), to provide the raw plaintext data of the key in (3), or to have the tool generate a key with the output file name specified in (4). |
| (3) | File name | When **File** is selected in (1), select the key file that contains the plaintext key. Key files must be binary files and have the extension `*.key`. |
| (4) | Raw data | When **Raw** is selected in (1), enter the AL1_Key or NONSECDBG_KEY plaintext key data in hex format. |
| (5) | Output key file | When **Random** is selected in (1), the tool will generate the key data. Specify the output file of the plaintext AL1_Key or NONSECDBG_KEY key data generated by the tool. The file can be a binary file with a `*.key` extension or a text file with a `*.txt` extension. **Note**: The specific amount of randomness of the random values generated by this tool is not guaranteed. Keys generated by this tool should be used for prototyping and test purposes only. |
| (6) | IV format | Select whether to use the input value from (6) for the IV value or to use a random number generated by the tool. The specific amount of randomness of the random values generated by this tool is not guaranteed. |
| (7) | Use specified value | When **Use specified value** is selected in (5), the value entered here will be used as the Initialization Vector during AL1_Key or NONSECDBG_KEY encryption. This value must be specified as 16 hex bytes in big-endian format. |

### 3.10.6 [Boundary] tab

Specify the Boundary information to be sent to the MCU/MPU by the Secure Factory Programming function.

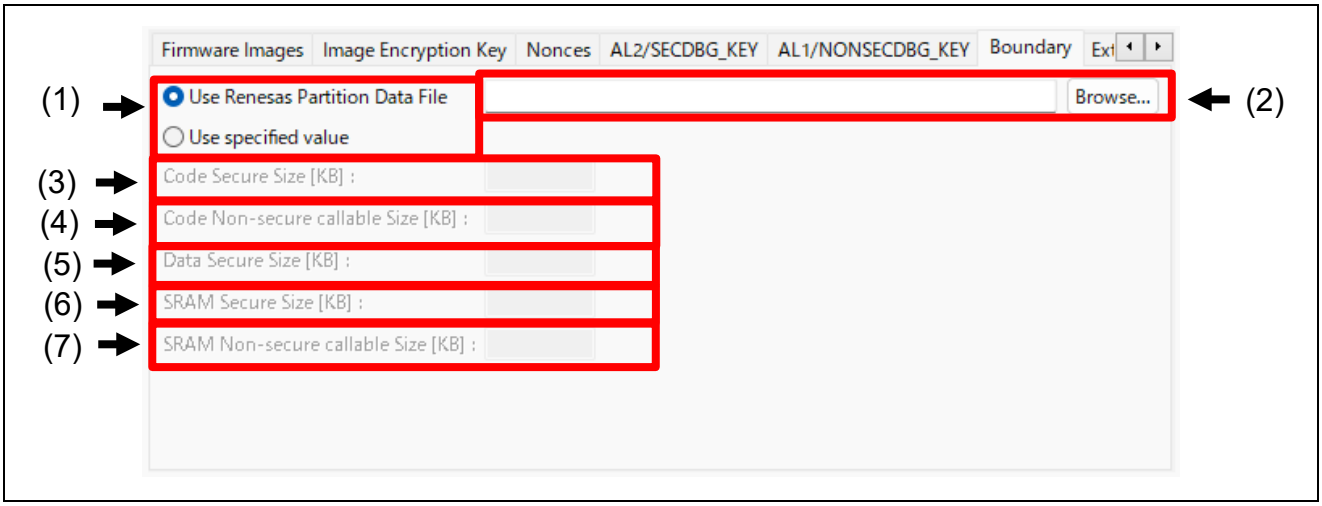


**Figure 3-41 [Boundary] Tab**

| No | Items | Description |
|---|---|---|
| (1) | Boundary setting | **Specify Renesas Partition Data File**.<br>: Specify the file in (2).<br>**Use specified value** : Specify the size of each area in (3) to (7). |
| (2) | Use Renesas Partition Data File | When **Specify Renesas Partition Data File** is selected in (1), the Renesas Partition Data File (`*.rpd`) output from e²studio can be specified. |
| (3) | Code Secure Size | When **Use specified value** is selected in (1), specify the size of Code flash secure area in KB. |
| (4) | Code Non-secure callable Size | When **Use specified value** is selected in (1), specify the size of Code flash Non-secure callable area in KB. |
| (5) | Data Secure Size | When **Use specified value** is selected in (1), specify the size of Data flash secure area in KB. |
| (6) | SRAM Secure Size | When **Use specified value** is selected in (1), specify the size of SRAM secure area in KB. |
| (7) | SRAM Non-secure callable Size | When **Use specified value** is selected in (1), specify the size of SRAM Non-secure callable area in KB. |

### 3.10.7 [External Flash Memory Area] tab

When using the Secure Factory Programming function to encrypt the image of the external flash memory area, specify the external flash memory area and the external flash memory write unit.

Two external flash memory areas can be specified. If you are only using one area, set External Flash Memory Area 0.

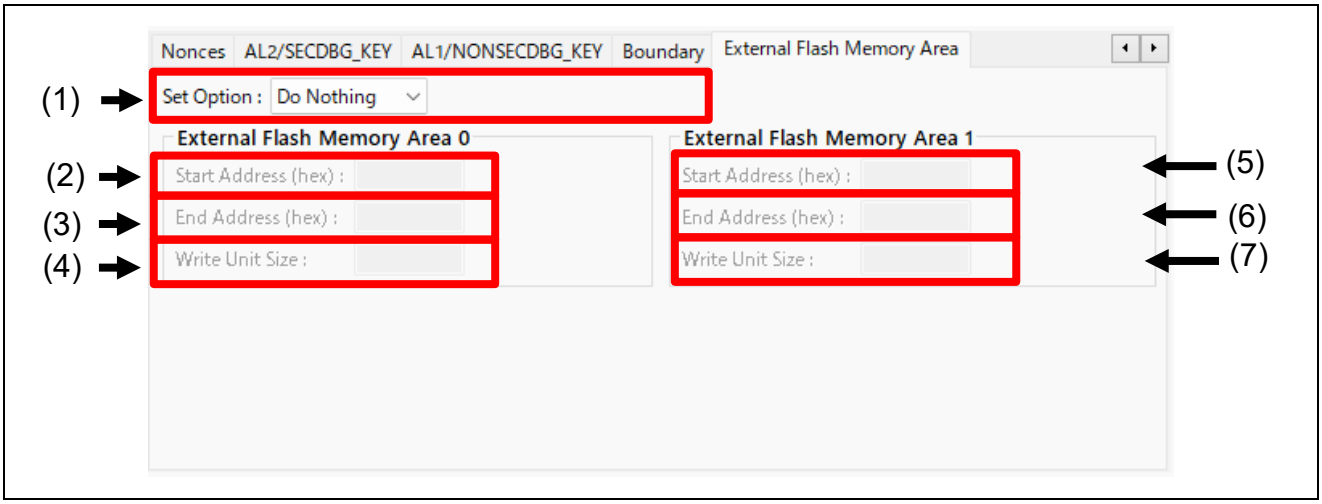


**Figure 3-42 [External Flash Memory Area] Tab**

| No | Items | Description |
|---|---|---|
| (1) | Set Option | For MCUs/MPUs that support the Secure Factory Programming function for external flash memory areas, specify whether or not to set the information for the external flash memory area that uses this function.<br>**Do nothing** : Do not set the External Flash Memory Area.<br>**Set one area** : 1 area external flash memory setting.<br>Specify the external flash area with (2) to (4).<br>**Set two areas** : 2 areas external flash memory setting.<br>Specify the external flash area with (2) to (7). |
| (2) | External Flash Memory Area 0 Start Address (hex) | When **Set one area** or **Set two areas** is selected in (1), specify the start address of external flash memory area 0. |
| (3) | External Flash Memory Area 0 End Address (hex) | When **Set one area** or **Set two areas** is selected in (1), specify the end address of external flash memory area 0. |
| (4) | External Flash Memory Area 0 Write Unit Size | When **Set one area** or **Set two areas** is selected in (1), specify the write unit size of external flash memory area 0.<br>Please specify the write unit in 16-byte units. |
| (5) | External Flash Memory Area 1 Start Address (hex) | When **Set two areas** is selected in (1), specify the start address of external flash memory area 1. |
| (6) | External Flash Memory Area 1 End Address (hex) | When **Set two areas** is selected in (1), specify the end address of external flash memory area 1. |
| (7) | External Flash Memory Area 1 Write Unit Size | When **Set two areas** is selected in (1), specify the write unit size of external flash memory area 1.<br>Please specify the write unit in 16-byte units. |

## 3.11 [RA6B1/RA6U1] tab

On this tab, you can attach headers, key certificates, and code certificates to the RA for wireless Provisioning SW, OEM_BL, or Application Firmware.

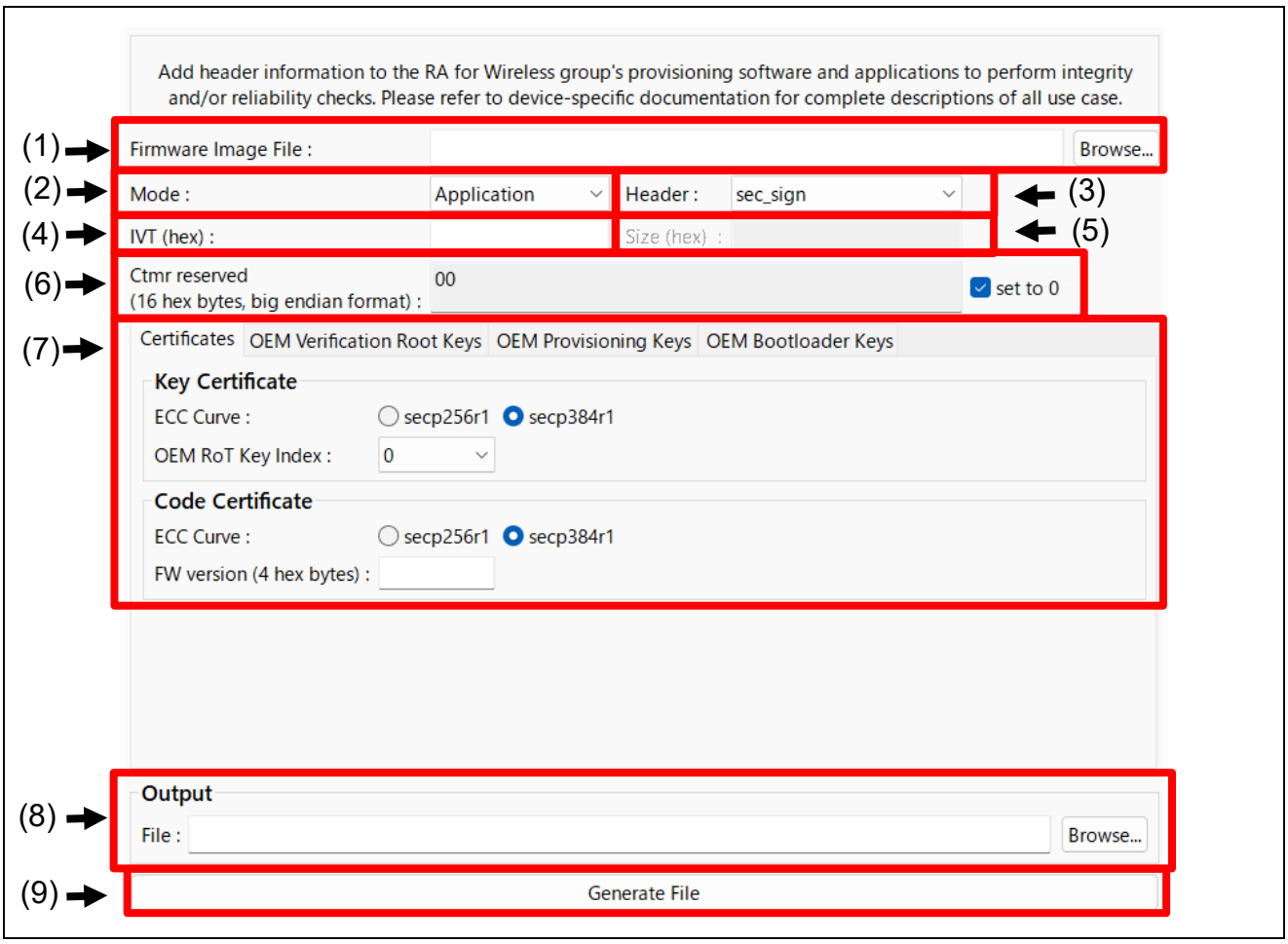


**Figure 3-43 [RA6B1/RA6U1] Tab**

| No | Items | Description |
|---|---|---|
| (1) | Firmware Image File | Specify the image file to which you want to add a header.<br>When specifying **Provisioning SW** in (2), only binary files can be specified.<br>When specifying **Application** you can specify either a binary file or a Motorola hex file. |
| (2) | Mode | Specify the target image to which you want to attach header information (Product Header, Image Header), key certificates, and code certificates.<br>**Provisioning SW** : Enter the Provisioning SW image in (1).<br>**Application** : Enter the OEM_BL or Application Firmware image in (1). |
| (3) | Header | **sec_sign** : Appends header information, Key Certificate, and Code Certificate to the image file specified in (1).<br>**sec** : Appends header information to the image file specified in (1). |
| (4) | IVT | If specified **Application** in (2), specify the IVT address of the image in (1). |
| (5) | Size | If specify **Application** in (2) and Motorola Hex file in (1), specify the size of the input file to be signed. |
| (6) | Ctmr_reserved | Set the binary value to be entered in the Ctmr_reserved field of the Image Header. |
| (7) | [**Certificates**]<br>[**OEM Verification Root Keys**]<br>[**OEM Provisioning Keys**]<br>[**OEM Bootloader Keys**] | [**Certificates**] tab: Configure the signing methods for key certificates and code certificates, and enter the relevant parameters.<br>[**OEM Verification Root Keys**] tab, [**OEM Provisioning Keys**] tab, [**OEM Bootloader Keys**] tab:<br>Enter the key information when **sec_sign** is specified in step (2).<br>For details on the [**Certificates**] tab, see *Section 3.11.1* [Certificates] tab.<br>For details on the [**OEM Verification Root Keys**] tab, refer to *Section 3.11.2* [OEM Verification Root Keys] tab.<br>For details on the [**OEM Provisioning Keys**] tab, refer to *Section 3.11.3* [OEM Provisioning Keys] tab.<br>For details on the [**OEM Bootloader Keys**] tab, refer to *Section 3.11.4* [OEM Bootloader Keys] tab. |
| (8) | Output | Specify the output file.<br>You can specify either a binary file (*.bin) or a Motorola Hex file (*.mot, *.srec) as the output file.<br>When specifying **Provisioning SW** in (2), only binary files can be specified. |
| (9) | **Generate File** | Generate provisioning SW or Application Firmware with headers and certificates attached. |

### 3.11.1 [Certificates] tab

Configure the signing methods for Key Certificate and Code Certificate, and enter the parameters.

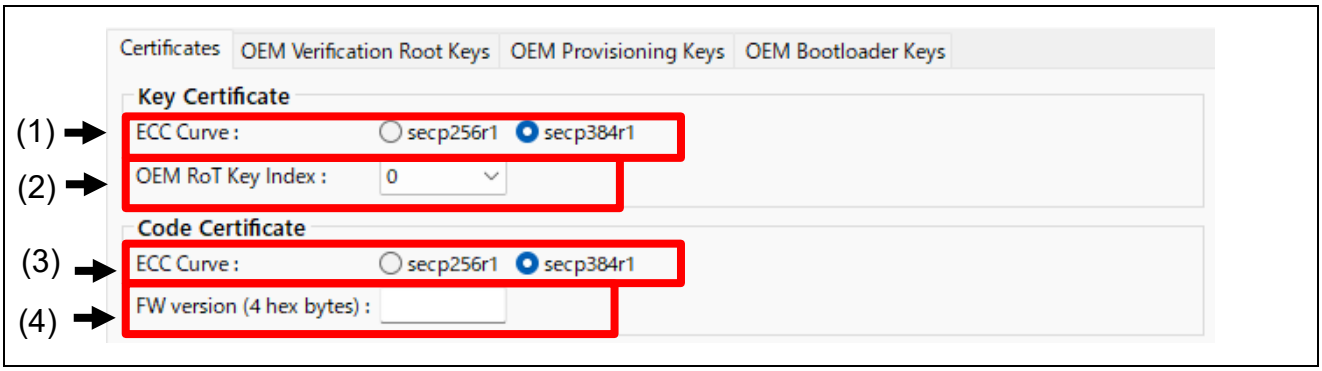


**Figure 3-44 [Certificates] Tab**

| No | Items | Description |
|---|---|---|
| (1) | ECC Curve | Select the type of ECC curve to use when signing the subject of Key Certificate. |
| (2) | OEM RoT Key Index | Specify the number of the OEM RoT Key to be used for Key Certificate verification. |
| (3) | ECC Curve | Select the type of ECC curve to use when signing the subject of Code Certificate. |
| (4) | FW version | When selecting mode **Application**, specify the firmware version of the signing target to be included in Code Certificate. |

### 3.11.2 [OEM Verification Root Keys] tab

Input OEM Verification Root Key.

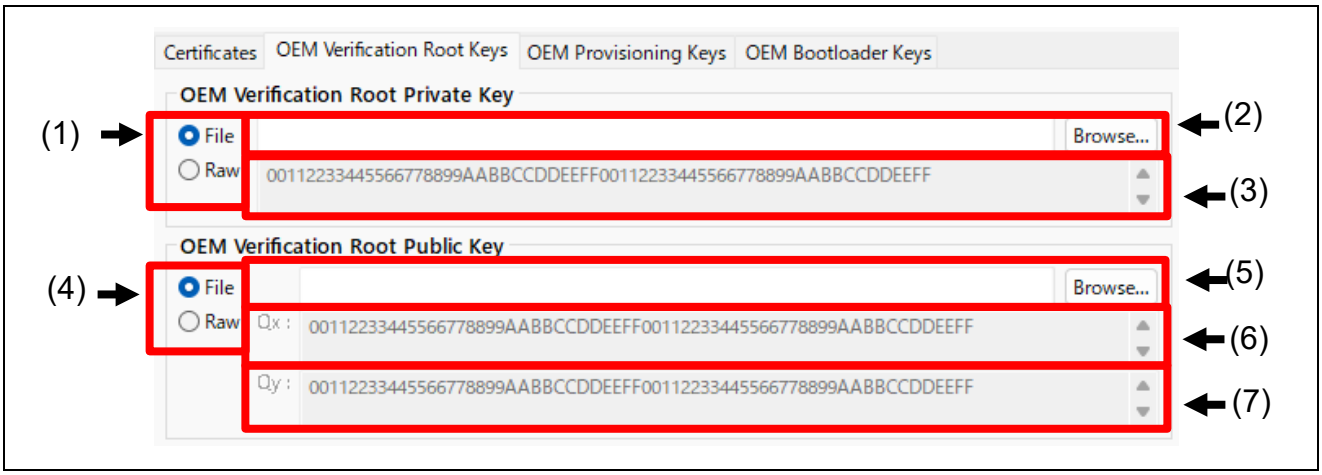


**Figure 3-45 [OEM Verification Root Key] Tab**

| No | Items | Description |
|---|---|---|
| (1) | Input format of OEM Verification Root Private Key | Select whether to provide a key file containing the key data in (2) or to provide the raw plaintext data of the key in (3). |
| (2) | File name | When **File** is selected in (1), select the key file that contains the plaintext key. The file can be a binary file with a `*.key` extension, a text file with a `*.txt` extension, or a PEM file with a `*.pem` extension.<br>Note that PEM files also contain public key information. When a PEM file is specified, the **OEM Verification Root Public Key** is not required. |
| (3) | Raw data | When **Raw** is selected in (1), enter the plain text key data in hexadecimal.<br>In **Certificates** tab, under **Key Certificates**, enter 32 bytes of private key data if you select the ECC curve **secp256r1**, or 48 bytes of private key data if you select the ECC curve **secp384r1**. |
| (4) | Input format of OEM Verification Root Public Key | Select whether to provide a key file containing the key data in (5) or to provide the raw plaintext data of the key in (6) and (7). |
| (5) | File name | When **File** is selected in (4), select the key file that contains the plaintext key. The file can be a binary file with a `*.key` extension or a text file with a `*.txt` extension. Note that if a PEM file was specified for the private key, that file also contains the public key. |
| (6) | Qx | When **Raw** is selected in (4), enter the plaintext data of the OEM Root public key Qx in hex format. |
| (7) | Qy | When **Raw** is selected in (4), enter the plaintext data of the OEM Root public key Qy in hex format. |

### 3.11.3 [OEM Provisioning Keys] tab

In Mode **Provisioning SW**, when header **sec_sign** is selected, enter the key pair for OEM provisioning.

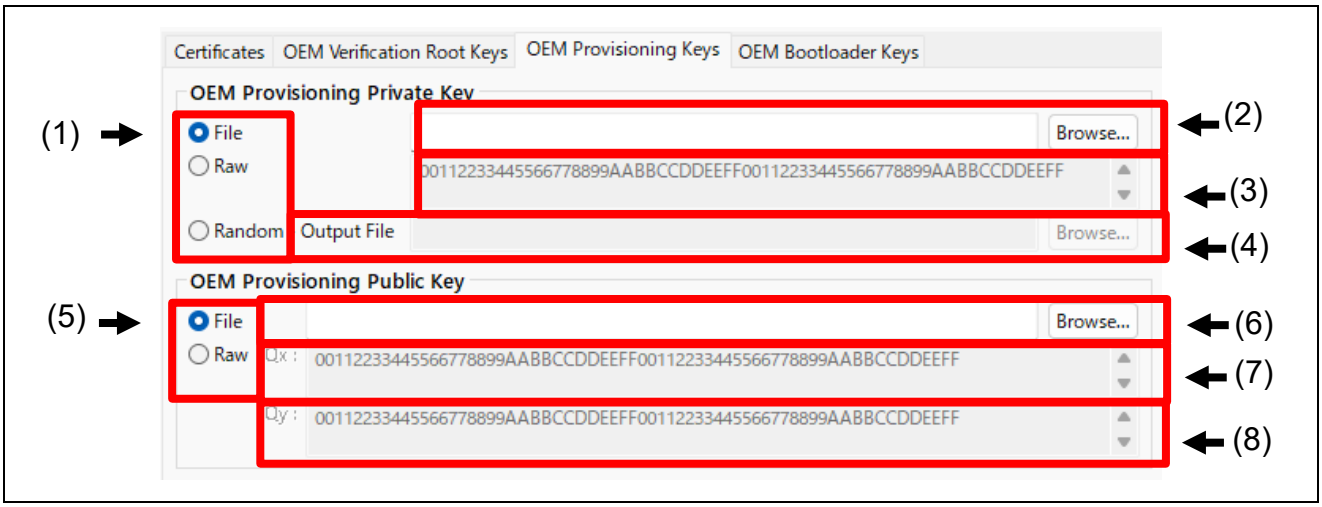


**Figure 3-46 [OEM Provisioning Keys] Tab**

| No | Items | Description |
|---|---|---|
| (1) | Input format of OEM Provisioning Private Key | Select whether to provide a key file containing the key data in (2), to provide the raw plaintext data of the key in (3), or to have the tool generate a key pair with the output file name specified in (4). |
| (2) | File name | When **File** is selected in (1), select the key file that contains the plaintext key. The file can be a binary file with a `*.key` extension, a text file with a `*.txt` extension, or a PEM file with a `*.pem` extension.<br>Note that PEM files also contain public key information. When a PEM file is specified, the **OEM Provisioning Public Key** is not required. |
| (3) | Raw data | When **Raw** is selected in (1), enter the plaintext key data in hexadecimal.<br>In **Certificates** tab, under **Code Certificates**, enter 32 bytes of private key data if you select the ECC curve **secp256r1**, or 48 bytes of private key data if you select the ECC curve **secp384r1**. |
| (4) | Output file name | When **Random** is selected in (1), the tool will generate the key data. Specify the output file of the plaintext key data generated by the tool. The output plaintext key file can be a text file (`*.txt`) or a binary file (`*.key`).<br>Both private and public keys are generated at the same time. The specified file name is appended with `_public` for public keys and `_private` for private keys. Injection/update files are generated only for the private key. The generated public key file can be used as input to create key injection files for the public key.<br>**Note**: The specific amount of randomness of the random values generated by this tool is not guaranteed. Keys generated by this tool should be used for prototyping and test purposes only |

| No | Items | Description |
|---|---|---|
| (5) | Input format of OEM Provisioning Public Key | Select whether to provide a key file containing the key data in (6) or to provide the raw plaintext data of the key in (7) and (8). |
| (6) | File name | When **File** is selected in (4), select the key file that contains the plaintext key. The file can be a binary file with a `*.key` extension or a text file with a `*.txt` extension. Note that if a PEM file was specified for the private key, that file also contains the public key. |
| (7) | Qx | When **Raw** is selected in (4), enter the plaintext data of the OEM Provisioning public key Qx in hex format. |
| (8) | Qy | When **Raw** is selected in (4), enter the plaintext data of the OEM Provisioning public key Qy in hex format. |

### 3.11.4 [OEM Bootloader Keys] tab

In Mode **Application**, when header **sec_sign** is selected, enter the key pair for OEM_BL.

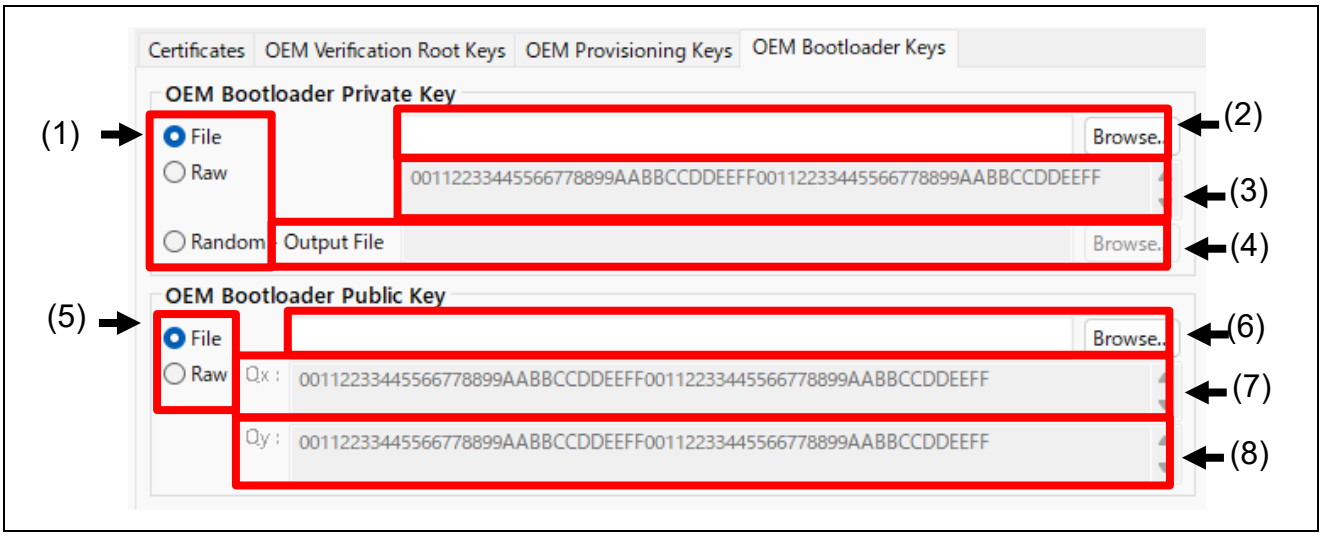


**Figure 3-47 [OEM Bootloader Keys] Tab**

| No | Items | Description |
|---|---|---|
| (1) | Input format of OEM Bootloader Private Key | Select whether to provide a key file containing the key data in (2), to provide the raw plaintext data of the key in (3), or to have the tool generate a key pair with the output file name specified in (4). |
| (2) | File name | When **File** is selected in (1), select the key file that contains the plaintext key. The file can be a binary file with a `*.key` extension, a text file with a `*.txt` extension, or a PEM file with a `*.pem` extension.<br>Note that PEM files also contain public key information. When a PEM file is specified, the **OEM Bootloader Public Key** is not required. |
| (3) | Raw data | When **Raw** is selected in (1), enter the plaintext key data in hexadecimal.<br>In **Certificates** tab, under **Code Certificates**, enter 32 bytes of private key data if you select the ECC curve **secp256r1**, or 48 bytes of private key data if you select the ECC curve **secp384r1**. |
| (4) | Output file name | When **Random** is selected in (1), the tool will generate the key data. Specify the output file of the plaintext key data generated by the tool. The output plaintext key file can be a text file (`*.txt`) or a binary file (`*.key`).<br>Both private and public keys are generated at the same time. The specified file name is appended with `_public` for public keys and `_private` for private keys. Injection/update files are generated only for the private key. The generated public key file can be used as input to create key injection files for the public key.<br>**Note**: The specific amount of randomness of the random values generated by this tool is not guaranteed. Keys generated by this tool should be used for prototyping and test purposes only |

| No | Items | Description |
|---|---|---|
| (5) | Input format of OEM Bootloader Public Key | Select whether to provide a key file containing the key data in (6) or to provide the raw plaintext data of the key in (7) and (8). |
| (6) | File name | When **File** is selected in (5), select the key file that contains the plaintext key. The file can be a binary file with a `*.key` extension or a text file with a `*.txt` extension. Note that if a PEM file was specified for the private key, that file also contains the public key. |
| (7) | Qx | When **Raw** is selected in (5), enter the plaintext data of the OEM Bootloader public Key Qx in hex format. |
| (8) | Qy | When **Raw** is selected in (5), enter the plaintext data of the OEM Bootloader public key Qy in hex format. |

# 4. Descriptions of CLI Functions

## 4.1 Command-line Syntax

skmt.exe [command] [options..]

Note: Command, options and parameters are not case sensitive.

Table 4-1 Command-line syntax

| Items | Description |
|---|---|
| skmt.exe | Executable file name. |
| command | Key generation commands. The command is preceded by “/” or “-“. |
| options .. | Zero or more options for the specified key generation command. Each option is preceded by “/” or “-“. |

## 4.2 Commands

Commands are shown in the table below.

Table 4-2 Command List

| Command | Description |
|---|---|
| **genufpk** | Generate UFPK file.<br>Upon success, the generated UFPK will be displayed on the console. |
| **genkuk** | Generate KUK file.<br>Upon success, the generated KUK will be displayed on the console. |
| **genkey** | Encrypt User key and output a file.<br>Upon success, the console will display the IV, W-UFPK, and the Encrypted User Key (including MAC). |
| **enctsip** | Encrypt a program for use with the TSIP's secure update function or factory programming. |
| **gencert** | Generates Key Certificate and Code Certificate that can be used in Renesas devices with First Stage Bootloader (FSBL) functionality. |
| **encdotf** | Encrypt user programs that can be used with Renesas devices equipped with the DOTF function. |
| **encsfp** | Encrypt user programs that can be used with Renesas devices equipped with Secure Factory Programming (SFP) functionality. |
| **genhdr** | Generate headers, key certificates, and code certificates for RA for Wireless. |
| **calcresponse** | Calculate a response value for challenge and response authentication and output it to the console. |
| **H** | Help |

This tool supports multiple file types. Specify the desired file type by using the file name extension as shown in *Table 4-3*. Both absolute and relative paths are supported.

Table 4-3 File Types and Extensions

| File type | Extension | Description |
|---|---|---|
| Binary key data | `*.key` | Binary data files for plaintext key material |
| Renesas key file | `*.rkey` | The file used by RFP for secure user and DLM key injection (if supported by the MCU/MPU). |
| Motorola hex file | `*.mot`,<br>`*.srec` | Motorola hex file |
| Binary data | `*.bin` | Binary data file |
| C source, header file | `*.c, *.h` | C source and header file |
| Text file | `*.txt` | Files written in ASCII characters |
| PEM file | `*.pem` | Base64-encoded file of x509 ASN.1 key, used by OpenSSL |
| RSU file | `*.rsu` | Image file for use as TSIP firmware update. |
| Secure Factory Programming file | `*.sfp` | Image file for use as Secure Flash Programming.<br>See appendix for Format. |
| Renesas Partition Data File | `*.rpd` | Boundary information file used by the Secure Factory Programming function. |

## 4.3 **genufpk** Command Options

This command generates a key file containing a User Factory Programming Key (UFPK). The UFPK can either be specified or generated by the tool.

Table 4-4 **genufpk** options

| Options | Parameters | Description |
|---|---|---|
| **ufpk** | Hex data | Specifies the 32-byte binary data for the UFPK.<br>This option is optional. If this option is omitted, a random value will be generated for the UFPK. |
| **output** | File Path | Specify the output file name.<br>This option is optional. If this option is omitted, the execution result will be output to the console. |
| **nooverwrite** | None | This option is optional. When this option is specified, an error will occur if the output file exists. |

Note: The specific amount of randomness of the random values generated by this tool is not guaranteed.

**Example of omitting ufpk option:**

**> skmt.exe /genufpk /output “D:\example\ufpk.key”**

**Example of using ufpk option:**

**> skmt.exe /genufpk**
**/ufpk “00112233445566778899aabbccddeeff00112233445566778899aabbccddeeff”**
**/output “D:\example\ufpk.key”**

## 4.4 **genkuk** Command Options

This command generates a key file containing a Key-Update Key (KUK). The KUK can either be specified or generated by the tool.

Table 4-5 **genkuk** options

| Options | Parameters | Description |
|---|---|---|
| **kuk** | Hex data | Specifies the 32-byte binary data used by KUK.<br>This option is optional. If this option is omitted, the random value generated in the tool will be used as KUK. |
| **output** | File Path | Specify the output file name.<br>This option is optional. If this option is omitted, the execution result will be output to the console. |
| **nooverwrite** | None | This option is optional. When this option is specified, an error will occur if the output file exists. |

Note: The specific amount of randomness of the random values generated by this tool is not guaranteed.

**Example of omitting kuk option:**

**> skmt.exe /genkuk /output “D:\example\kuk.key”**

**Example of using kuk option:**

**> skmt.exe /genkuk /output “D:\example\kuk.key”**
**/kuk “00112233445566778899aabbccddeeff00112233445566778899aabbccddeeff”**

## 4.5 **genkey** Command Options

This command generates files to use for secure key injection or update.

The following options can be used with the **genkey** command. As described in *Table 4-6*, some data inputs can be specified as hex data or by a binary file. When specifying a file, add "`file=`" at the beginning of the file path.

Table 4-6 **genkey** options(1)

| Options | Parameters | Description |
|---|---|---|
| **iv** | Hex data / File Path | Initialization Vector (IV) for encrypting the user or DLM key (16 bytes). This option is optional. If this option is omitted, a random value generated in the tool will be used as the IV. |
| **ufpk** | File Path | Key file containing a UFPK to be used for secure key injection. Note that this UFPK must correspond to the specified W-UFPK. When **kuk** is specified as an option, the **ufpk** option cannot be specified. |
| **wufpk** | File Path | Key file containing a wrapped UFPK provided by the Renesas Key Wrap service. When kuk is specified as an option, the wufpk option cannot be specified. |
| **kuk** | Hex data / File Path | KUK to be used for secure key update. When ufpk is specified as an option, the kuk option cannot be specified. |
| **mcu** | ASCII | Target MCU/MPU. This value must be one of the options listed in *Section 4.5.1 mcu Options*. |
| **keytype** | ASCII / Hex data | The type of key being prepared for injection or update. Either the key name or its numerical representation can be specified. This value must be one of the options listed in *Section 4.5.2 keytype Options*. |
| **key** | Hex data / File Path | DLM key data or User key data. For the format of the keyed data to be input, refer to *Section 4.5.3.1 Hex Data Direct Input*. This option is optional. If this option is omitted, the tool may be able to generate key data. However, there are some keys that cannot be generated in the case of public key cryptography. See *Section 4.5.3.3 key Option Omitted* for details. |
| **filetype** | ASCII | Output file type. See *Section 4.5.4 filetype Options*.<br>This option is optional. If this option is omitted and the output option is specified, it will be output as a bin file. If the filetype and output options are omitted, output will be to the console. |

Note: The specific amount of randomness of the random values generated by this tool is not guaranteed.

Table 4-7 **genkey** options(2)

| Options | Parameters | Description |
|---|---|---|
| **address** | Hex data | The address where the key is to be injected. This option must be specified when mot is specified for **filetype**. |
| **fileadd** | - | This option can be set when **csource**, **bin**, or **mot** is specified for **filetype**. When this option is specified, data is appended to the output file if it exists. |
| **bswap** | ASCII | This option can be set when **bin** or **mot** is specified for **filetype**. When this option is specified, output data is swapped. |
| **keyname** | ASCII | Specify the <**keyname**> portion of the structure, variable, and data size value defined in the C source and header files. This option must be specified when **csource** is specified in **filetype**. See *Section 4.5.4.2 csource Option for filetype* for a detailed description of how <**keyname**> is used. |
| **keyfileoutput** | File Path | This option is optional. If the **key** option is omitted, resulting in the DLM or user key being generated by the tool, use this option to specify the output path of the generated key data. Binary data is output. |
| **output** | File Path | Specify the output file name. This option is optional. If this option is omitted, the execution result will be output to the console. |
| **nooverwrite** | None | This option is optional. When this option is specified, an error will occur if the output file exists. |

Note: The specific amount of randomness of the random values generated by this tool is not guaranteed.

**Example of using ufpk option:**

**> skmt.exe /genkey /ufpk file=”D:\example\ufpk.key” /wufpk file=”D:\example\ufpk_enc.key” /mcu “RA-SCE9” /keytype “AES-128” /key “000102030405060708090A0B0C0D0E0F” /filetype “rfp” /output “D: \example\aes128.rkey”**

**Example of using kuk option:**

**> skmt.exe /genkey /kuk file=”D:\example\ufpk.key” /mcu “RA-SCE9” /keytype “AES-128” /key “000102030405060708090A0B0C0D0E0F” /filetype “csource” /output “D:\example\aes128.c”**

### 4.5.1 **mcu** Option

The **mcu** option specifes the MCU/MPU type and cryptographic engine.

Table 4-8 **mcu** Option

| ASCII | Description |
|---|---|
| RA-RSIP-E51A | Specified when using RA Family RSIP-E51A Security Function and Protected Mode |
| RA-RSIP-E51A-CM | Specified when using RA Family RSIP-E51A Compatibility Mode |
| RA-RSIP-E50D | Specified when using RA Family RSIP-E50D Security Function and Protected Mode |
| RA-RSIP-E50D-CM | Specified when using RA Family RSIP-E50D Compatibility Mode |
| RA-RSIP-E31A | Specified when using RA Family RSIP-E31A Security Function and Protected Mode |
| RA-RSIP-E31A-CM | Specified when using RA Family RSIP-E31A Compatibility Mode |
| RA-RSIP-E31B | Specified when using RA Family RSIP-E31B Security Function and Protected Mode |
| RA-RSIP-E31B-CM | Specified when using RA Family RSIP-E31B Compatibility Mode |
| RA-RSIP-E11A | Specified when using RA Family RSIP-E11A Security Function and Protected Mode |
| RA-RSIP-E11A-CM | Specified when using RA Family RSIP-E11A Compatibility Mode |
| RA-SCE9 | Specified when using RA Family SCE9 Security Function and Protected Mode |
| RA-SCE9-CM | Specified when using RA Family SCE9 Compatibility Mode |
| RA-SCE7 | Specified when using RA Family SCE7 |
| RA-SCE5_B | Specified when using RA Family SCE5_B |
| RA-SCE5 | Specified when using RA Family SCE5 |
| RX-RSIP-E50D | Specified when using RX Family RSIP-E50D Security Function and Protected Mode |
| RX-RSIP-E50D-CM | Specified when using RX Family RSIP-E50D Compatibility Mode |
| RX-RSIP-E11A | Specified when using RX Family RSIP-E11A Security Function and Protected Mode |
| RX-RSIP-E11A-CM | Specified when using RX Family RSIP-E11A Compatibility Mode |
| RX-TSIP | Specified when using RX Family TSIP |
| RX-TSIPLite | Specified when using RX Family TSIP-Lite |
| RZ-RSIP-T2M | Specified when using RZ Family RZ/T2M RSIP |
| RZ-RSIP-T2ME | Specified when using RZ Family RZ/T2ME RSIP |
| RZ-RSIP-T2L | Specified when using RZ Family RZ/T2L RSIP |
| RZ-RSIP-N2L | Specified when using RZ Family RZ/N2L RSIP |
| RZ-TSIP | Specified when using RZ Family TSIP |
| Synergy-SCE7 | Specified when using Synergy Platform SCE7 |
| Synergy-SCE5 | Specified when using Synergy Platform SCE5 |

### 4.5.2 **keytype** Option

The **keytype** option specifies the type of key that is being prepared for secure injection or update. Either the ASCII or the hex value can be used with this option. ASCII is recommended for readability.

Note that not all MCUs/MPUs support all key types.

Table 4-9 Authentication key for DLM transition

| ASCII | **keytype** value | Description |
|---|---|---|
| DLM-SSD | 0x01 | Authentication key used in transition to SSD state in DLM. |
| DLM-NSECSD | 0x02 | Authentication key used in transition to NSECSD state in DLM. |
| DLM-RMA-REQ | 0x03 | Authentication key used in transition to RMA_REQ state in DLM. |
| DLM-AL2 | 0x01 | Key for DLM authentication level AL2 transition (AL2_Key) |
| DLM-AL1 | 0x02 | Key for DLM authentication level AL1 transition (AL1_Key) |
| DLM-RMA | 0x03 | DLM RMA _REQ Key for state authentication (RMA_Key) |

Table 4-10 User Key AES

| ASCII | **keytype** value | Description |
|---|---|---|
| AES-128 | 0x05 | AES-128bit key |
| AES-192 | 0x06 | AES-192bit key |
| AES-256 | 0x07 | AES-256bit key |
| AES-128XTS | 0x08 | AES-128bit XTS key |
| AES-256XTS | 0x09 | AES-256bit XTS key |
| AES-256-DOTF | 0x07 | AES-256bit key<br>Export an Encrypted User Key file for RA for Wireless DOTF. For details, see 4.5.2.1 AES-256-DOTF Option |

Table 4-11 User Key RSA

| ASCII | **keytype** value | Description |
|---|---|---|
| RSA-1024-public | 0x0A | RSA 1024bit public key |
| RSA-1024-private | 0x0B | RSA 1024bit private key |
| RSA-2048-public | 0x0C | RSA 2048bit public key |
| RSA-2048-private | 0x0D | RSA 2048bit private key |
| RSA-3072-public | 0x0E | RSA 3072bit public key |
| RSA-3072-private | 0x0F | RSA 3072bit private key |
| RSA-4096-public | 0x10 | RSA 4096bit public key |
| RSA-4096-private | 0x11 | RSA 4096bit private key |
| RSA-2048-public-TLS | 0xFE | RSA 2048bit public key for TLS |

Note: “RSA-2048-public-TLS“ cannot be specified by using the **keytype** value. It must be specified using ASCII.

Table 4-12 User Key ECC

| ASCII | **keytype** value | Description |
|---|---|---|
| secp192r1-public | 0x12 | ECC NIST P-192 (secp192r1) public key |
| secp192r1-private | 0x13 | ECC NIST P-192 (secp192r1) private key |
| secp224r1-public | 0x14 | ECC NIST P-224 (secp224r1) public key |
| secp224r1-private | 0x15 | ECC NIST P-224 (secp224r1) private key |
| secp256r1-public | 0x16 | ECC NIST P-256 (secp256r1) public key |
| secp256r1-private | 0x17 | ECC NIST P-256 (secp256r1) private key |
| secp384r1-public | 0x18 | ECC NIST P-384 (secp384r1) public key |
| secp384r1-private | 0x19 | ECC NIST P-384 (secp384r1) private key |
| secp521r1-public | 0x24 | ECC NIST P-521 (secp521r1) public key |
| secp521r1-private | 0x25 | ECC NIST P-521 (secp521r1) private key |
| brainpoolP256r1-public | 0x1C | ECC brainpoolP256r1 public key |
| brainpoolP256r1-private | 0x1D | ECC brainpoolP256r1 private key |
| brainpoolP384r1-public | 0x1E | ECC brainpoolP384r1 public key |
| brainpoolP384r1-private | 0x1F | ECC brainpoolP384r1 private key |
| brainpoolP512r1-public | 0x20 | ECC brainpoolP512r1 public key |
| brainpoolP512r1-private | 0x21 | ECC brainpoolP512r1 private key |
| secp256k1-public | 0x22 | ECC Koblitz curve secp256k1 public key |
| secp256k1-private | 0x23 | ECC Koblitz curve secp256k1 private key |
| Ed25519-public | 0x26 | EdDSA Ed25519 public key |
| Ed25519-private | 0x27 | EdDSA Ed25519 private key |

Table 4-13 User Key SHA-HMAC

| ASCII | **keytype** value | Description |
|---|---|---|
| HMAC-SHA1 | 0x00 | HMAC-SHA1 key |
| HMAC-SHA224 | 0x1A | HMAC-SHA2-224 key |
| HMAC-SHA256 | 0x1B | HMAC-SHA2-256 key |
| HMAC-SHA384 | 0x28 | HMAC-SHA2-384 key |
| HMAC-SHA512 | 0x29 | HMAC-SHA2-512 key |
| HMAC-SHA512-224 | 0x2a | HMAC-SHA2-512/224 key |
| HMAC-SHA512-256 | 0x2b | HMAC-SHA2-512/256 key |
| HMAC-SHA3-224 | 0x2c | HMAC-SHA3-224 key |
| HMAC-SHA3-256 | 0x2d | HMAC-SHA3-256 key |
| HMAC-SHA3-384 | 0x2e | HMAC-SHA3-384 key |
| HMAC-SHA3-512 | 0x2f | HMAC-SHA3-512 key |

Note: “HMAC-SHA1“ cannot be specified by using the **keytype** value. It must be specified using ASCII.

Table 4-14 User Key ARC4

| ASCII | **keytype** value | Description |
|---|---|---|
| ARC4 | 0x00 | ARC4 Key |

Note: “ARC4“ cannot be specified by using the **keytype** value. It must be specified using ASCII.

Table 4-15 User Key DES

| ASCII | **keytype** value | Description |
|---|---|---|
| TDES | 0x00 | Triple DES key |

Note: “TDES“ cannot be specified by using the **keytype** value. It must be specified using ASCII.

Table 4-16 User Key ChaCha20-Poly1305

| ASCII | **keytype** value | Description |
|---|---|---|
| CHACHA20-POLY1305 | 0x30 | ChaCha20-Poly1305 key |

Table 4-17 OEM Root Public Key

| ASCII | **keytype** value | Description |
|---|---|---|
| OEM_ROOT_PK | 0xFD | OEM Root public key to be injected into the device when using FSBL<br>secp256r1 public key |
| OEM_ROOT_PK_SECP256R1_RA_WIRELESS | 0x16 | OEM RoT Key used with RA for Wireless<br>secp256r1 public key |
| OEM_ROOT_PK_SECP384R1_RA_WIRELESS | 0x18 | OEM RoT Key used with RA for Wireless<br>secp384r1 public key |

Table 4-18 Key Update Key

| ASCII | **keytype** value | Description |
|---|---|---|
| key-update-key | 0xFF | Key Update Key |

#### 4.5.2.1 AES-256-DOTF Option

When **keytype** "**AES-256-DOTF**" option is specified, AES-256-bit key is wrapped by UFPK and outputs the file in the format specified by **filetype** option. At the same time, it outputs a binary file in TLV format containing the Encrypted User Key for RA for Wireless DOTF.

The filename is modified by replacing the file extension specified in **output** option with the suffix "`_dotf_wrapped_key.bin`" followed by the original extension (`.bin`). The file is also output to the folder specified in **output** option at the same time.

The format of the Encrypted User Key binary file is as follows:

Table 4-19 Encrypted User Key File Format

| **Fields** | **Size(byte)** | **Value** | **Note** |
|---|---|---|---|
| Type | 2 | 0x00A0 | Output in little-endian order. |
| Length | 2 | 0x0030 | Output in little-endian order. |
| Value | 48 | Encrypted User Key | AES-256 key wrapped by UFPK |

### 4.5.3 **key** Option

The **key** option is used to specify the plaintext DLM or user key that needs to be prepared for secure injection or update. The plaintext can be specified by direct input of the hex data, or by a binary `*.key` file or `*.bin` file. Some key types can also be generated by the tool.

#### 4.5.3.1 Hex Data Direct Input

If the plaintext key is provided by direct input of the hex data, the required number of bytes as per the specified **keytype** option must be provided, as per the following tables.

Table 4-20 DLM-SSD, DLM-NSECSD, DLM-RMA-REQ, DLM-AL2, DLM-AL1, DLM-RMA

| Byte | Data |
|---|---|
| 0-15 | DLM key data |

Table 4-21 AES-128

| Byte | Data |
|---|---|
| 0-15 | AES-128bit key data |

Table 4-22 AES-192

| Byte | Data |
|---|---|
| 0-23 | AES-192bit key data |

Table 4-23 AES-256

| Byte | Data |
|---|---|
| 0-31 | AES-256bit key data |

Table 4-24 AES-128XTS

| Byte | Data |
|---|---|
| 0-15 | AES-128bit key1 |
| 16-31 | AES-128bit key2 |

Table 4-25 AES-256XTS

| Byte | Data |
|---|---|
| 0-31 | AES-256bit key1 |
| 32-63 | AES-256bit key2 |

Table 4-26 RSA-1024-public

| Byte | Data |
|---|---|
| 0-127 | RSA 1024bit Modulus n |
| 128-131 | RSA 1024bit Exponent e |

Table 4-27 RSA-1024-private

| Byte | Data |
|---|---|
| 0-127 | RSA 1024bit Modulus n |
| 128-255 | RSA 1024bit Decryption Exponent d |

Table 4-28 RSA-2048-public / RSA-2048-public-TLS

| Byte | Data |
|---|---|
| 0-255 | RSA 2048bit Modulus n |
| 256-259 | RSA 2048bit Exponent e |

Table 4-29 RSA-2048-private

| Byte | Data |
|---|---|
| 0-255 | RSA 2048bit Modulus n |
| 256-511 | RSA 2048bit Decryption Exponent d |

Table 4-30 RSA-3072-public

| Byte | Data |
|---|---|
| 0-383 | RSA 3072bit Modulus n |
| 384-387 | RSA 3072bit Exponent e |

Table 4-31 RSA-3072-private

| Byte | Data |
|---|---|
| 0-383 | RSA 3072bit Modulus n |
| 384-767 | RSA 3072bit Decryption Exponent d |

Table 4-32 RSA-4096-public

| Byte | Data |
|---|---|
| 0-511 | RSA 4096bit Modulus n |
| 512-515 | RSA 4096bit Exponent e |

Table 4-33 RSA-4096-private

| Byte | Data |
|---|---|
| 0-511 | RSA 4096bit Modulus n |
| 512-1023 | RSA 4096bit Decryption Exponent d |

Table 4-34 secp192r1-public

| Byte | Data |
|---|---|
| 0-23 | ECC Public key Qx |
| 24-47 | ECC Public key Qy |

Table 4-35 secp192r1-private

| Byte | Data |
|---|---|
| 0-23 | ECC Private key d |

Table 4-36 secp224r1-public

| Byte | Data |
|---|---|
| 0-27 | ECC Public key Qx |
| 28-55 | ECC Public key Qy |

Table 4-37 secp224r1-private

| Byte | Data |
|---|---|
| 0-27 | ECC Private key d |

Table 4-38 secp256r1-public / brainpoolP256r1-public / secp256k1-public / OEM_ROOT_PK / OEM_ROOT_PK_SECP256R1_RA_WIRELESS

| Byte | Data |
|---|---|
| 0-31 | ECC Public key Qx |
| 32-63 | ECC Public key Qy |

Table 4-39 secp256r1-private / brainpoolP256r1-private / secp256k1-private / Ed25519-private

| Byte | Data |
|---|---|
| 0-31 | ECC Private key d |

Table 4-40 secp384r1-public / brainpoolP384r1-public / OEM_ROOT_PK_SECP384R1_RA_WIRELESS

| Byte | Data |
|---|---|
| 0-47 | ECC Public key Qx |
| 48-95 | ECC Public key Qy |

Table 4-41 secp384r1-private / brainpoolP384r1-private

| Byte | Data |
|---|---|
| 0-47 | ECC Private d |

Table 4-42 brainpoolP512r1-public

| Byte | Data |
|---|---|
| 0-63 | ECC Public key Qx |
| 64-127 | ECC Public key Qy |

Table 4-43 brainpoolP512r1-private

| Byte | Data |
|---|---|
| 0-63 | ECC Private key d |

Table 4-44 secp521r1-public

| Byte | Data |
|---|---|
| 0-65 | 0 Padding(7bit) \|\| ECC Public key Qx |
| 66-127 | 0 Padding(7bit) \|\| ECC Public key Qy |

Note : || denotes bitwise concatenation.

Table 4-45 secp521r1-private

| Byte | Data |
|---|---|
| 0-65 | 0 Padding(7bit) \|\| ECC Private key d |

Note : || denotes bitwise concatenation.

Table 4-46 Ed25519-public

| Byte | Data |
|---|---|
| 0-31 | sign(1bit) \|\| ECC Private key Qy(255bit) |

Note : || denotes bitwise concatenation.

Table 4-47 HMAC-SHA1

| Byte | Data |
|---|---|
| 0-19 | HMAC-SHA-1 key |

Table 4-48 HMAC-SHA224 / HMAC-SHA3-224

| Byte | Data |
|---|---|
| 0-27 | HMAC-SHA2-224 / HMAC-SHA3-224key |

Table 4-49 HMAC-SHA256 / HMAC-SHA3-256

| Byte | Data |
|---|---|
| 0-31 | HMAC-SHA2-256 / HMAC-SHA3-256 key |

Table 4-50 HMAC-SHA384 / HMAC-SHA3-384

| Byte | Data |
|---|---|
| 0-47 | HMAC-SHA2-384 / HMAC-SHA3-384 key |

Table 4-51 HMAC-SHA512 / HMAC-SHA512-224 / HMAC-SHA512-256 / HMAC-SHA3-512

| Byte | Data |
|---|---|
| 0-63 | HMAC-SHA2-512 / 512-224 / 512-256 / HMAC-SHA3-512 key |

Table 4-52 ARC4

| Byte | Data |
|---|---|
| 0-255 | ARC4 key |

Table 4-53 TDES

| Byte | Data |
|---|---|
| 0-7 | 56bit DES key with odd parity 1 |
| 8-15 | 56bit DES key with odd parity 2 |
| 16-23 | 56bit DES key with odd parity 3 |

Note: Add odd parity for every 7 bits of key data.

Example:

- DES key data = 0x00000000000000 -> 0x0101010101010101
- DES key data = 0xFFFFFFFFFFFFFF -> 0xFEFEFEFEFEFEFEFE

Table 4-54 ChaCha20-Poly1305

| Byte | Data |
|---|---|
| 0-31 | ChaCha20-Poly1305 key |

Table 4-55 key-update-key

| Byte | Data |
|---|---|
| 0-31 | Key-Update Key |

**Example of usage with Hex Data:**

**> skmt.exe /genkey /ufpk file="D:\example\ufpk.key" /wufpk file="D:\example\ufpk_enc.key" /mcu "RA-SCE9" /keytype "AES-128" /key "000102030405060708090A0B0C0D0E0F" /filetype "rfp" /output "D:\example\aes128.rkey"**

#### 4.5.3.2 File Input

The **key** option supports file input for the following file formats

Table 4-56 **key** Option Input File

| File types | Extension | Format |
|---|---|---|
| binary | `*.key`, `*.bin` | Binary data file in the format shown in *Section 4.5.3.1 Hex Data Direct Input*. |
| Text | `*.txt` | Input File representing byte data in hexadecimal ASCII characters in the format shown in *Section 4.5.3.1 Hex Data Direct Input*. |
| PEM | `*.pem` | Key files in PEM file format for asymmetric keys generated by OpenSSL can be read. Only the key types specified in *Table 4-57 PEM file support for asymmetric keys* are supported |

Table 4-57 PEM file support for asymmetric keys

| Algorithms | keytypes |
|---|---|
| RSA | RSA-1024-private, RSA-1024-public, RSA-2048-private, RSA-2048-public, RSA-3072-private, RSA-3072-public, RSA-4096-private, RSA-4096-public, RSA-2048-public-TLS |
| ECC | secp256r1-private, secp256r1-public, secp384r1-private, secp384r1-public, secp521r1-private, secp521r1-public, brainpoolP256r1-private, brainpoolP256r1-public, brainpoolP384r1-private, brainpoolP384r1-public, brainpoolP512r1-private, brainpoolP512r1-public, Ed25519-private, Ed25519-public, OEM_ROOT_PK, OEM_ROOT_PK_SECP256R1_RA_WIRELESS, OEM_ROOT_PK_SECP384R1_RA_WIRELESS |

Note : The macOS version does not support PEM file input for brainpool curves.

**Example of specifying a binary file:**

**> skmt.exe /genkey /ufpk file="D:\example\ufpk.key" /wufpk file="D:\example\ufpk_enc.key" /mcu "RA-SCE9" /keytype "AES-128" /key file="D:\example\aes128.key" /filetype "rfp" /output "D:\example\aes128.rkey"**

**Example of specifying a text file:**

**> skmt.exe /genkey /ufpk file="D:\example\ufpk.key" /wufpk file="D:\example\ufpk_enc.key" /mcu "RA-SCE9" /keytype "AES-128" /key file="D:\example\aes128.txt" /filetype "rfp" /output "D:\example\aes128.rkey"**

**Example of specifying a PEM file:**

**> skmt.exe /genkey /ufpk file="D:\example\ufpk.key" /wufpk file="D:\example\ufpk_enc.key" /mcu "RA-SCE9" /keytype "RSA-2048-private" /key file="D:\example\rsa2048.pem" /filetype "rfp" /output "D:\example\rsa2048.rkey"**

To manually create key data, use a hex editor or binary editor, and create the data in big-endian order in the Hex Editor. An example is shown below.

- Example of manual creation of AES128-bit key file:
  AES 128-bit key `00112233445566778899AABBCCDDEEFF`

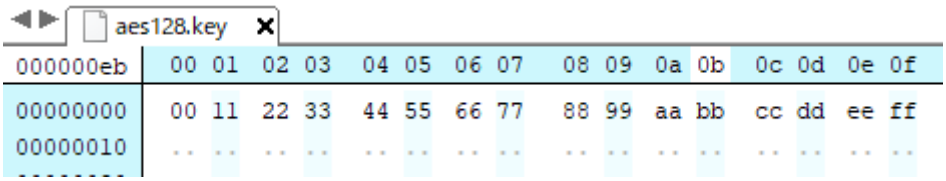


**Figure 4-1 Example of manual creation of AES 128-bit key file**

- Example of manual creation of RSA 2048 public key file:

Modulus
```
bad47a84c1782e4dbdd913f2a261fc8b
65838412c6e45a2068ed6d7f16e9cdf4
462b39119563cafb74b9cbf25cfd544b
dae23bff0ebe7f6441042b7e109b9a8a
faa056821ef8efaab219d21d67634847
85622d918d395a2a31f2ece8385a8131
e5ff143314a82e21afd713bae817cc0e
e3514d4839007ccb55d68409c97a18ab
62fa6f9f89b3f94a2777c47d6136775a
56a9a0127f682470bef831fbec4bcd7b
5095a7823fd70745d37d1bf72b63c4b1
b4a3d0581e74bf9ade93cc4614861755
3931a79d92e9e488ef47223ee6f6c061
884b13c9065b591139de13c1ea292749
1ed00fb793cd68f463f5f64baa53916b
46c818ab99706557a1c2d50d232577d1
```

Exponent `10001`

rsa-2048-public.key

| 00000157 | 00 01 | 02 03 | 04 05 | 06 07 | 08 09 | 0a 0b | 0c 0d | 0e 0f |
|---|---|---|---|---|---|---|---|---|
| 00000000 | ba d4 | 7a 84 | c1 78 | 2e 4d | bd d9 | 13 f2 | a2 61 | fc 8b |
| 00000010 | 65 83 | 84 12 | c6 e4 | 5a 20 | 68 ed | 6d 7f | 16 e9 | cd f4 |
| 00000020 | 46 2b | 39 11 | 95 63 | ca fb | 74 b9 | cb f2 | 5c fd | 54 4b |
| 00000030 | da e2 | 3b ff | 0e be | 7f 64 | 41 04 | 2b 7e | 10 9b | 9a 8a |
| 00000040 | fa a0 | 56 82 | 1e f8 | ef aa | b2 19 | d2 1d | 67 63 | 48 47 |
| 00000050 | 85 62 | 2d 91 | 8d 39 | 5a 2a | 31 f2 | ec e8 | 38 5a | 81 31 |
| 00000060 | e5 ff | 14 33 | 14 a8 | 2e 21 | af d7 | 13 ba | e8 17 | cc 0e |
| 00000070 | e3 51 | 4d 48 | 39 00 | 7c cb | 55 d6 | 84 09 | c9 7a | 18 ab |
| 00000080 | 62 fa | 6f 9f | 89 b3 | f9 4a | 27 77 | c4 7d | 61 36 | 77 5a |
| 00000090 | 56 a9 | a0 12 | 7f 68 | 24 70 | be f8 | 31 fb | ec 4b | cd 7b |
| 000000a0 | 50 95 | a7 82 | 3f d7 | 07 45 | d3 7d | 1b f7 | 2b 63 | c4 b1 |
| 000000b0 | b4 a3 | d0 58 | 1e 74 | bf 9a | de 93 | cc 46 | 14 86 | 17 55 |
| 000000c0 | 39 31 | a7 9d | 92 e9 | e4 88 | ef 47 | 22 3e | e6 f6 | c0 61 |
| 000000d0 | 88 4b | 13 c9 | 06 5b | 59 11 | 39 de | 13 c1 | ea 29 | 27 49 |
| 000000e0 | 1e d0 | 0f b7 | 93 cd | 68 f4 | 63 f5 | f6 4b | aa 53 | 91 6b |
| 000000f0 | 46 c8 | 18 ab | 99 70 | 65 57 | a1 c2 | d5 0d | 23 25 | 77 d1 |
| 00000100 | 00 01 | 00 01 | .. .. | .. .. | .. .. | .. .. | .. .. | .. .. |

**Figure 4-2 Example of manual creation of RSA 2048 public key file**

### 4.5.3.3 **key** Option Omitted

If the **key** option is omitted from the command line and the specified **keytype** is for either a symmetric algorithm or an asymmetric algorithm that is specified in the table below, the tool will generate a random value for the key.

For asymmetric keys, a key pair is generated. Both private and public keys are generated at the same time, and the specified file name is appended with `_public` for public keys and `_private` for private keys. This option supports the asymmetric **keytype** option shown in the table below.

It is recommended to use the **keyfileoutput** option to create a key file with the plaintext key(s).

The specific amount of randomness of the random values generated by this tool is not guaranteed. Keys generated by this tool should be used for prototyping and test purposes only.

Table 4-58 **keytype** that supports asymmetric key generation functionality

| Algorithms | **keytype** |
|---|---|
| RSA | RSA-1024-private, RSA-2048-private, RSA-3072-private, RSA-4096-private |
| ECC | secp256r1-private, secp384r1-private, secp521r1-private<br>brainpoolP256r1-private, brainpoolP384r1-private, brainpoolP512r1-private,<br>Ed25519-private, OEM_ROOT_PK,<br>OEM_ROOT_PK_SECP256R1_RA_WIRELESS,<br>OEM_ROOT_PK_SECP384R1_RA_WIRELESS |

Note : The macOS version does not support key generation for brainpool curves.

### 4.5.4 **filetype** Option

The **filetype** option allows you to specify the output file format of the prepared key. If this option is omitted, binary data is output. The tool will return an error if the specified **filetype** does not match the file name extension.

Table 4-59 **filetype** option

| ASCII | Description |
|---|---|
| **rfp** | Output a file in a file format that can be read by the Renesas Flash Programmer (RFP). The output file extension must be `*.rkey`. |
| **csource** | Output C source and header file. The output file extension must be `*.c`. |
| **bin** | Output binary data file. The output file extension must be `*.bin`. |
| **mot** | Output binary data in Motorola hex format. The output file extension must be `*.mot`.<br>Specify the starting address with 8 hexadecimal digits(32bits) in the address option. |

#### 4.5.4.1 **rfp** Option for **filetype**

This option outputs key data in Renesas Key File format to be used by the Renesas Flash Programmer.

**RFP file output example:**

**> skmt.exe /genkey /ufpk file="D: \example\ufpk.key" /wufpk file="D:\example\ufpk_enc.key" /mcu "RA-SCE9" /keytype "AES-128" /key file="D:\example\aes128.key" /filetype "rfp" /output "D:\example\abc.rkey"**

#### 4.5.4.2 **csource** Option for **filetype**

This option outputs C source files and header files.

When the **keyname** option is used, the specified <keyname> will be used as part of the structure name, the global variable name, and the byte size definition. If the **keyname** option is not used, default text will be used. These options are shown by the following table.

Table 4-60 **keyname** option usage

| | Setting **keyname** | Omitting **keyname** |
|---|---|---|
| Structure name | <keyname>_t | encrypted_user_key_data_t |
| Global variable name | g_<keyname> | g_encrypted_user_key_data |
| encrypted_user_key bytesize definition value | <KEYNAME>_SIZE * | ENCRYPTED_KEY_BYTE_SIZE |

* <keyname> will be converted to uppercase

The output C source structure is as follows, with <keyname> replaced as per *Table 4-60 **keyname** option usage*.

Table 4-61 Structure output by **csource** when the **ufpk** option is specified

| name | Type | Description |
|---|---|---|
| g_<keyname> | <keyname>_t | - |
| keytype | uint32_t | Outputs keytype value for security engines that use keytype values, or 0 for security engines that do not use keytype values. |
| shared_key_number | uint32_t | 0 is output. Not used when the **mcu** option is "RX-TSIP" or "RX-TSIPLite". |
| wufpk[32] | uint8_t | Output when the **wufpk** option is specified. If the **wufpk** option is not specified, 0 is output. |
| initial_vector[16] | uint8_t | The Initialization Vector used for user key encryption. |
| encrypted_user_key [<KEYNAME>_SIZE] | uint8_t | The encrypted user key. <KEYNAME>_SIZE is the size of the encrypted user key. See *Table 4-65-Table 4-75* for encrypted user key sizes. |
| crc[4] | uint8_t | CRC-32 value from keytype to encrypted_user_key. |

Table 4-62 Structure output by "**csource**" when the **kuk** option is specified

| name | Type | Description |
|---|---|---|
| g_<keyname> | <keyname>_t | - |
| keytype | uint32_t | Outputs keytype value for security engines that use keytype values, or 0 for security engines that do not use keytype values. |
| shared_key_number | uint32_t | 0 is output. Not used when the **mcu** option is "RX-TSIP" or "RX-TSIPLite". |
| initial_vector[16] | uint8_t | The Initialization Vector used for user key encryption. |
| encrypted_user_key [<KEYNAME>_SIZE] | uint8_t | The encrypted user key. <KEYNAME>_SIZE is the size of the encrypted user key. See *Table 4-65-Table 4-75* for encrypted user key sizes. |
| crc[4] | uint8_t | CRC-32 value from keytype to encrypted_user_key. |

**C source file output example:**

**> skmt.exe /genkey /ufpk file="D: \example\ufpk.key" /wufpk file="D: \example\ufpk_enc.key" /mcu "RX-TSIP" /keytype "AES-128" /key file="D:\example\aes128.key" /filetype "csource" /keyname testKey /output "D:\example\abc.c"**

#### 4.5.4.3 **bin** Option for **filetype**

This option outputs a file of binary data. The data array of the output binary data is as follows:

Table 4-63 Binary data output by **bin** when the **ufpk** option is specified

| Byte | Data |
|---|---|
| 0 | keytype<br>Outputs keytype value for security engines that use keytype values, or 0 for security engines that do not use keytype values. |
| 1 - 3 | Reserved (0 padding) |
| 4 | shared_key number<br>0 is output. Not used when the **mcu** option is "RX-TSIP" or "RX-TSIPLite". |
| 5 - 7 | Reserved (0 padding) |
| 8 - 39 | WUFPK<br>Output when the **wufpk** option is specified. If the **wufpk** option is not specified, 0 is output. |
| 40 - 55 | initial_vector |
| 56 - (56+N-1) | encrypted_user_key |
| (56+N) - 56+N +3 | crc<br>CRC-32 value from keytype to encrypted_user_key. |

Note: “N” is the same as the <KEYNAME>_SIZE value.

Table 4-64 Binary data output by **bin** when the **kuk** option is specified

| Byte | Data |
|---|---|
| 0 | keytype<br>Outputs keytype value for security engines that use keytype values, or 0 for security engines that do not use keytype values. |
| 1 - 3 | Reserved (0 padding) |
| 4 | shared_key number<br>0 is output. Not used when the **mcu** option is "RX-TSIP"or "RX-TSIPLite". |
| 5 - 7 | Reserved (0 padding) |
| 8 - 23 | initial_vector |
| 24 - (24+N-1) | encrypted_user_key |
| (24+N) - 24+N +3 | crc<br>CRC-32 value from keytype to encrypted_user_key. |

Note: “N” is the same as the <KEYNAME>_SIZE value.

**Binary file output example:**

**> skmt.exe /genkey /ufpk file="D:\example\ufpk.key" /wufpk file="D:\example\ufpk_enc.key" /mcu "RX-TSIP" /keytype "AES-128" /key file="D:\example\aes128.key" /filetype "bin" /output "D:\example\abc.bin"**

#### 4.5.4.4 **mot** Option for **filetype**

This option outputs a Motorola Hex format file. The format of the file is specified in *Table 4-63* and *Table 4-64*. The address for the data must be set by using the **address** option and specifying a hexadecimal 8-digit (32-bit) starting address.

**mot file output example:**

**> skmt.exe /genkey /ufpk file="D:\example\ufpk.key" /wufpk file="D:\example\ufpk_enc.key" /mcu "RX-TSIP" /keytype "AES-128" /key file="D:\example\aes128.key" /filetype "mot" /address "FFF00000" /output "D:\example\abc.mot"**

The encrypted_user_key size for each **keytype** option is shown below.

Table 4-65 encrypted_user_key size DLM Key

| **keytype** options | Byte size |
|---|---|
| DLM-SSD | 32 |
| DLM-NSECSD | 32 |
| DLM-RMA-REQ | 32 |
| DLM-AL2 | 32 |
| DLM-AL1 | 32 |
| DLM-RMA | 32 |

Table 4-66 encrypted_user_key size AES Key

| **keytype** options | Byte size |
|---|---|
| AES-128 | 32 |
| AES-192 | 48 |
| AES-256 | 48 |
| AES-128XTS | 48 |
| AES-256XTS | 80 |

Table 4-67 encrypted_user_key size RSA Key

| **keytype** options | Byte size |
|---|---|
| RSA-1024-public | 160 |
| RSA-1024-private | 272 |
| RSA-2048-public | 288 |
| RSA-2048-private | 528 |
| RSA-3072-public | 416 |
| RSA-3072-private | 784 |
| RSA-4096-public | 544 |
| RSA-4096-private | 1040 |
| RSA 2048bit public key for TLS | 288 |

Table 4-68 encrypted_user_key size ECC Key

| **keytype** options | Byte size |
|---|---|
| secp192r1-public | 80 |
| secp192r1-private | 48 |
| secp224r1-public | 80 |
| secp224r1-private | 48 |
| secp256r1-public | 80 |
| secp256r1-private | 48 |
| secp384r1-public | 112 |
| secp384r1-private | 64 |
| secp521r1-public | 132 |
| secp521r1-private | 66 |
| brainpoolP256r1-public | 80 |
| brainpoolP256r1-private | 48 |
| brainpoolP384r1-public | 112 |
| brainpoolP384r1-private | 64 |
| brainpoolP512r1-public | 144 |
| brainpoolP512r1-private | 80 |
| secp256k1-public | 80 |
| secp256k1-private | 48 |

Table 4-69 encrypted_user_key size Ed25519 Key

| **keytype** options | Byte size |
|---|---|
| Ed25519-public | 48 |
| Ed25519-private | 48 |

Table 4-70 encrypted_user_key size HMAC-SHA Key

| **keytype** options | Byte size |
|---|---|
| HMAC-SHA1 | 48 |
| HMAC-SHA224 | 48 |
| HMAC-SHA256 | 48 |
| HMAC-SHA384 | 64 |
| HMAC-SHA512 | 80 |
| HMAC-SHA512-224 | 80 |
| HMAC-SHA512-256 | 80 |
| HMAC-SHA3-224 | 48 |
| HMAC-SHA3-256 | 48 |
| HMAC-SHA3-384 | 64 |
| HMAC-SHA3-512 | 80 |

Table 4-71 encrypted_user_key size ARC4 Key

| **keytype** options | Byte size |
|---|---|
| ARC4 | 272 |

Table 4-72 encrypted_user_key size TDES Key

| **keytype** options | Byte size |
|---|---|
| TDES | 48 |

Table 4-73 encrypted_user_key size ChaCha20-Poly1305 Key

| **keytype** options | Byte size |
|---|---|
| ChaCha20-Poly1305 | 48 |

Table 4-74 encrypted_user_key size OEM_ROOT_PK Key

| **keytype** options | Byte size |
|---|---|
| OEM_ROOT_PK | 80 |
| OEM_ROOT_PK_SECP256R1_RA_WIRELESS | 80 |
| OEM_ROOT_PK_SECP384R1_RA_WIRELESS | 112 |

Table 4-75 encrypted_user_key size Key Update Key

| **keytype** options | Byte size |
|---|---|
| key-update-key | 48 |

### 4.5.5 **fileadd** Option

The **fileadd** option can be set when "**csource**", "**bin**" or "**mot**" is specified for the filetype option. The **fileadd** option can be set when the **filetype** option is "**csource**", "**bin**" or "**mot**".

When the same file name as the output file exists, the generated encrypted key information is appended to the existing data instead of overwriting it.

Note: When "**csource**" is specified in the **filetype** option, it does not check if the same definition values are defined for existing data and newly added data. Use the **keyname** option to ensure that they do not have the same definition value.

**fileadd example :**

```
> skmt.exe /genkey /ufpk file="D:\example\ufpk.key" /wufpk file="D:\example\ufpk_enc.key"
  /mcu "RX-TSIP" /keytype "AES-128" /key file="D:\example\aes128.key" /filetype "csource"
  /fileadd /keyname "aes128" /output "D:\example\abc.c"
```

### 4.5.6 **bswap** Option

The **bswap** option specifies the sequence of data to be output in ASCII characters. When omitted, the behavior is "**32-big**".

Table 4-76 **bswap** option

| ASCII | Description |
|---|---|
| **32-big** | Output each of the data in *Table 4-63* and *Table 4-64* to a file in big-endian data order. |
| **32-little** | Each of the data in *Table 4-63* and *Table 4-64* is output to a file in a 4-byte swapped data sequence. |

**bswap example :**

```
> skmt.exe /genkey /ufpk file="D:\example\ufpk.key" /wufpk file="D:\example\ufpk_enc.key"
  /mcu "RX-TSIP" /keytype "AES-128" /key file="D:\example\aes128.key" /filetype "csource"
  /bswap "32-little" /keyname "aes128" /output "D:\example\abc.c"
```

### 4.5.7 **keyfileoutput** Option

If the **key** option is omitted from the command line, the tool will generate a random key. The **keyfileoutput** option is used to save the generated key to a binary key file or a text file. The specified file extension may be either `*.key` or `*.txt`.

Only symmetric and some asymmetric key pairs can be generated by this tool. To generate an asymmetric key pair, specify the private key as the **keytype**. The following table shows the asymmetric key pair types that can be generated.

Table 4-77 Asymmetric key types that can be generated

| Algorithms | **keytype** |
|---|---|
| RSA | RSA-1024-private, RSA-2048-private, RSA-3072-private, RSA-4096-private |
| ECC | secp256r1-private, secp384r1-private, secp521r1-private<br>brainpool P256r1-private, brainpool P384r1-private, brainpool P512r1-private,<br>Ed25519-private, OEM_ROOT_PK,<br>OEM_ROOT_PK_SECP256R1_RA_WIRELESS,<br>OEM_ROOT_PK_SECP384R1_RA_WIRELESS |

The specific amount of randomness of the random values generated by this tool is not guaranteed. Keys generated by this tool should be used for prototyping and test purposes only.

When an asymmetric key pair is generated and **/keyfileoutput** is specified, “ `_private`” is appended to the file name for the private key and “`_public`” is appended to the file name for the public key. For example, if “`/keyfileoutput abc.key /output abc.rkey`” is specified, the following files are generated:

- `abc_private.key` containing the plaintext private key
- `abc_public.key` containing the plaintext public key
- `abc_private.rkey` containing the encrypted keys for injecting the private key
- `abc_public.rkey` containing the encrypted keys for injecting the public key

For the output key data format, see the format defined in *Section 4.5.3.1.Hex Data Direct Input*.

**Symmetric key file output example:**

```
> skmt.exe /genkey /ufpk file="D:\example\ufpk.key" /wufpk file="D:\example\ufpk_enc.key"
    /mcu "RA-SCE9" /keytype "AES-128" /filetype "rfp" /keyfileoutput file="D:\example\aes.key"
  /output "D:\example\abc.rkey"
```

**Asymmetric key file output example:**

```
> skmt.exe /genkey /ufpk file="D:/example/ufpk.key" /wufpk file="D:\example\ufpk_enc.key"
    /mcu "RX-TSIP" /keytype "RSA-1024-private" /filetype "bin"
    /keyfileoutput file="D:\example\rsa1024.key" /output "D:\example\rsa1024_private.bin"
```

## 4.6 **enctsip** Command Options

The options listed below can be used with the **enctsip** command. Input can be in the form of hexadecimal data or a binary file.

Table 4-78 enctsip Option

| Options | Parameter | Description |
|---|---|---|
| **mode** | ASCII | Specifies the data format of the output file. For details, refer to *Section 4.6.1 mode Option*. |
| **ver** | Decimal data | Specifies the version of the RSU header appended to the output file. Version 1 or 2 can be specified. If it is omitted, a value of **2** is used. For details, refer to *Section 4.6.2 ver Option*. |
| **prg** | File Path | Specifies the MOT file to be encrypted. |
| **prg_sb** | File Path | When **factory** is specified by the **mode** option, this option specifies the MOT file of the secure boot program to accompany the MOT file to be encrypted. |
| **enckey** | Hex data / File Path | Specifies the key to be used to encrypt the session key used to encrypt the data in the MOT file specified by the **prg** option (16 bytes). Specify 16-byte hex data or a binary file (`*.key`). |
| **session_key** | Hex data | Specifies the session key used to encrypt the data in the MOT file specified by the **prg** option (32 bytes). This option is optional. If it is omitted, a random value generated by the Security Key Management Tool is used as the key data.*1 |
| **iv_fw** | Hex data | Specifies the initialization vector (IV) used when encrypting the data in the MOT file specified by the **prg** option (16 bytes). This option is optional. If it is omitted, a random value generated by the Security Key Management Tool is used as the IV.*1 |
| **startaddr** | Hex data | Specifies the start address of the area of the MOT file specified by the **prg** option to be encrypted. The **address** must be aligned to a 16-byte boundary. |
| **endaddr** | Hex data | Specifies the end address of the area of the MOT file specified by the **prg** option to be encrypted. The encryption area specified by the start and end addresses must have a size that is aligned to 16-byte boundaries.<br>The maximum encryption range is 8MB. |
| **destaddr** | Hex data | When **mot** is specified by the **filetype** option, this option specifies the address to which the encrypted user program is output. |
| **imgflg** | ASCII / Hex data | Specifies the value input for the Image Flags RSU header. For details, refer to *4.6.3*, *imgflg Option*. |
| **filetype** | ASCII | Specifies the type of the output file. For details, refer to *Section 4.6.4*, *filetype Option*. |
| **flash_wsize** | Decimal data | Can only be specified for **ver** option 2.<br>Specify a value of 16 bytes or more in increments of 16 bytes.<br>This option can be omitted. If omitted, specify 256. |
| **df_ena** | ASCII | This option appends the data in Data Flash area contained in the mot file to be encrypted specified by the **prg** option to the output file.<br>See *Section 4.6.5* ***df_ena*** *Option* for details. |

Note: 1. Random values generated by the Security Key Management Tool are not guaranteed to provide sufficient randomness.

Table 4-79 enctsip Options(2)

| Options | Parameter | Description |
|---|---|---|
| **motoffset** | None | Can only be specified with **ver** option 2.<br>When **motoffset** option is added, the address offset is enabled when the rsu file is made into a mot file. it cannot be specified at the same time as the **df_ena** option. |
| **output** | File Path | Specifies the name of the output file. |
| **nooverwrite** | None | This option is optional. If it is specified and an output file exists, an error occurs. |

**Example use of mode factory setting:**

**> skmt.exe /enctsip /mode "factory" /ver "1" /prg "D:\example\userprog.mot"**
**/prg_sb "D:\example\secureboot.mot" /enckey "0123456789ABCDEF0123456789ABCDEF"**
**/startaddr "FFF80300" /endaddr "FFFEFFFF" /destaddr "FFF00300"**
**/filetype "mot" /output "D:\example\factory.mot"**

**Example use of mode update setting:**

**> skmt.exe /enctsip /mode "update" /ver "1" /prg "D:\example\userprog.mot"**
**/enckey "0123456789ABCDEF0123456789ABCDEF"**
**/startaddr "FFF80300" /endaddr "FFFEFFFF" /filetype "rsu" /output "D:\example\update.rsu"**

### 4.6.1 **ver** Option

The ver option specifies the version of the RSU file. version 1 (when the ver option is 1) and version 2 (when the ver option is 2) of the RSU file differ in the RSU header added to the output file and the method of creating the encrypted data.

#### 4.6.1.1 **ver** option 1

(1) Encrypted data creation method

Encrypts the entire area specified by the **startaddr** and **endaddr** options in the mot file specified by the **prg** option. Areas with no data are padded with 0xFF.

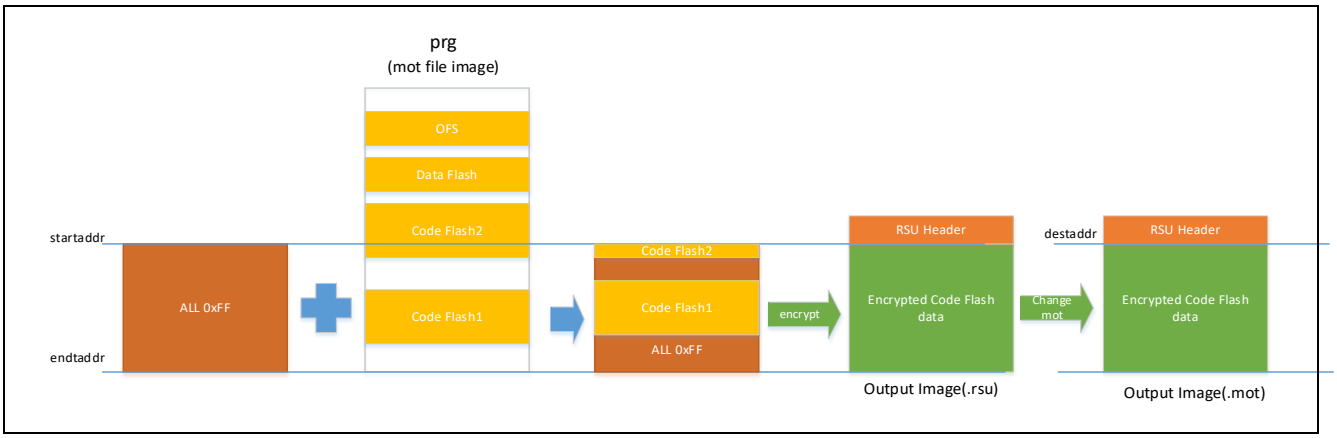


**Figure 4-3 Encryption flow when ver option 1 is specified**

### (2) RSU Header

*Table 4-80 RSU Header Version 1* shows the header of Version 1.

Table 4-80 RSU Header Version 1

| Offset | Component | Contents Name | Length | Note |
|---|---|---|---|---|
| 0x0000 | Header | Magic Code | 7 | “Renesas” |
| 0x0007 | | Image Flags | 1 | Value defined by imgflg option |
| 0x0008 | Signature | Firmware Verification Type | 32 | ASCII “mac-aes128-cmac-with-tsip” |
| 0x0028 | | Signature size | 4 | Not used (0x00). |
| 0x002C | | Signature | 256 | Not used (0x00). |
| 0x012C | Option | Dataflash Flag | 4 | Not used (0x00). |
| 0x0130 | | Dataflash Start Address | 4 | Not used (0x00). |
| 0x0134 | | Dataflash End Address | 4 | Not used (0x00). |
| 0x0138 | | Image Size | 4 | Not used (0x00). |
| 0x013C | | Reserved | 124 | All 0x00 |
| 0x01B8 | | Format Type*1 | 4 | Output file format<br>ASCII character string “rsu” (0x72, 0x73, 0x75, 0x00) or “mot” (0x6D, 0x6F, 0x74, 0x00) |
| 0x01BC | | IV*1 | 16 | IV used when encrypting the user program |
| 0x01CC | | SessionKey0*1 | 16 | Key, encrypted using enckey, that is used when encrypting the user program |
| 0x01DC | | SessionKey1*1 | 16 | Key, encrypted using enckey, that is used when encrypting the user program |
| 0x01EC | | Encrypted user program<br>+ MAC word size*1 | 4 | Total word size of the encrypted user program plus the MAC generated when encrypting the user program |
| 0x01F0 | | MAC*1 | 16 | MAC generated when encrypting the user program |
| 0x0200 | Descriptor | Sequence Number | 4 | Always 1 |
| 0x0204 | | Start Address | 4 | Start address of user program area |
| 0x0208 | | End Address | 4 | End address of user program area |
| 0x020C | | Execution Address | 4 | Storage address of user program execution start address<br>(Fixed Value 0xFFFEFFFC) |
| 0x0210 | | Hardware ID | 4 | Not used (0x00). |
| 0x0214 | | Reserved(0x00) | 236 | — |
| 0x0300 | Application Binary | | N | Encrypted user program |
| 0x0300+N | Dataflash Binary | | M | Not supported. |

Note: 1. This parameter is based on the *.rsu file format defined in version 1.x of RX firmware update FIT module and has been modified for the TSIP.

### 4.6.1.2 **ver** option 2

(1) Encrypted data creation method

The area specified by the **startaddr** and **endaddr** options in the mot file specified by the **prg** option is divided into blocks of the size specified by the **flash_wsize** option, and only the blocks in which data exists are encrypted.

If data exists only in a part of the block divided by the **flash_wsize** option, the area with no data is padded with 0xFF.

When outputting as a Motorola hex file, if **motoffset** option is not added, the rsu file (binary data) is output with **destaddr** address added. If **motoffset** option is added, **startaddr** address is converted to a **destaddr** address, and the address information in the input mot file is taken over and output.

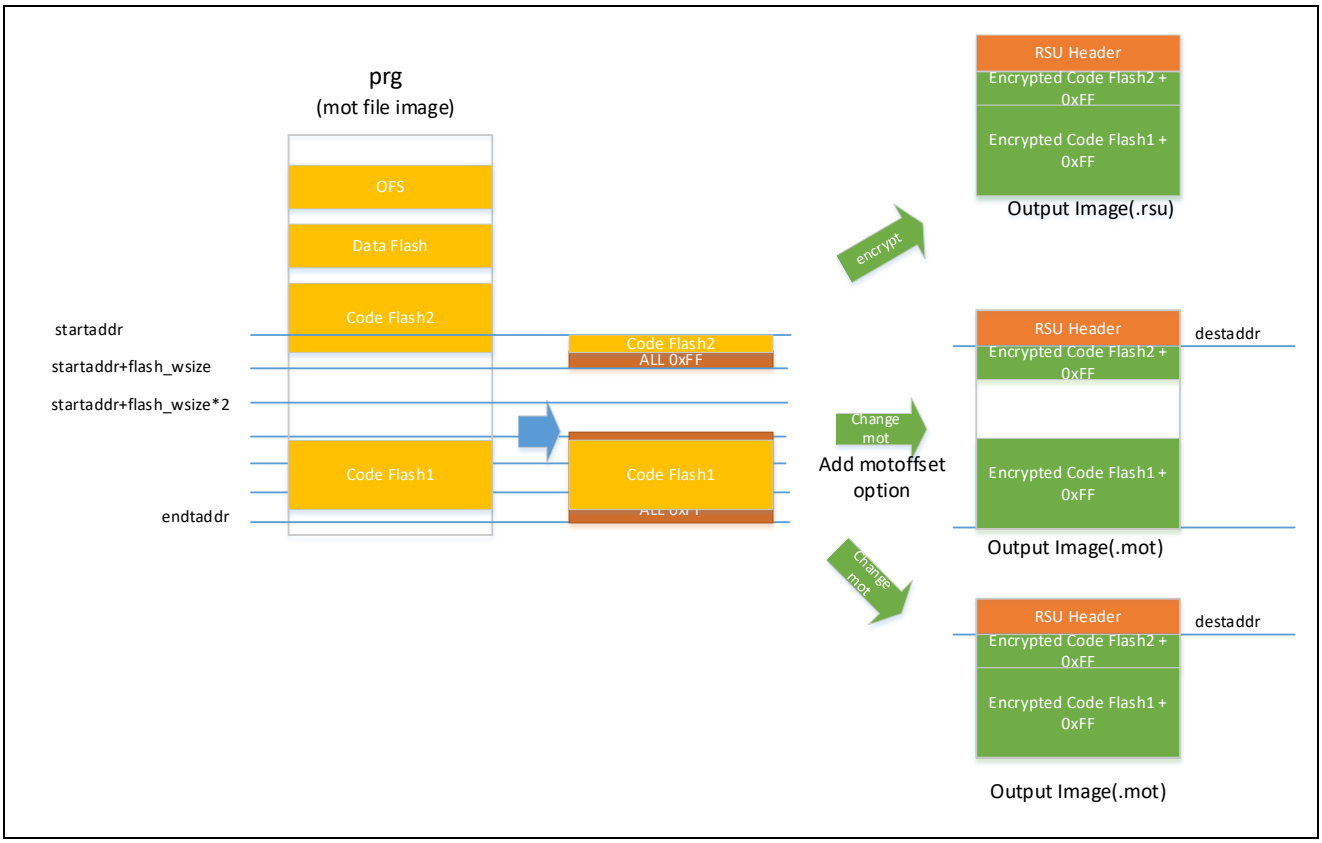


**Figure 4-4 Encryption flow when ver option 2 is specified**

### (2) RSU Header

*Table 4-81 RSU Header Version 2* shows the header of Version 2.

Table 4-81 RSU Header Version 2

| offset | Component | Contents name | Length | Note |
|---|---|---|---|---|
| 0x0000 | Header | Magic Code | 7 | “RELFWV2” |
| 0x0007 | | Reserved | 1 | Not used (0x00) |
| 0x0008 | Signature | Firmware Verification Type | 32 | ASCII “mac-aes128-cmac-with-tsip” |
| 0x0028 | | Signature size | 4 | Not used (0x00) |
| 0x002C | | Signature | 64 | Not used (0x00) |
| 0x006C | Option | Reserved(0x00) | 332 | - |
| 0x01B8 | | Format Type *1 | 4 | Output file format<br>ASCII character string “rsu” (0x72, 0x73, 0x75, 0x00) or “mot” (0x6D, 0x6F, 0x74, 0x00) |
| 0x01BC | | IV *1 | 16 | IV used when encrypting the user program |
| 0x01CC | | SessionKey0 *1 | 16 | Key, encrypted using enckey, that is used when encrypting the user program |
| 0x01DC | | SessionKey1 *1 | 16 | Key, encrypted using enckey, that is used when encrypting the user program |
| 0x01EC | | Encrypted user program + MAC word size *1 | 4 | Total word size of the encrypted user program plus the MAC generated when encrypting the user program |
| 0x01F0 | | MAC *1 | 16 | MAC generated when encrypting the user program |
| 0x0200 | Descriptor | Number of program data | 4 | Number of user program data (max. 31) |
| 0x0204 | | Start address[0] | 4 | Start address of user program area 1st |
| 0x0208 | | Data size[0] | 4 | Data size of user program area 1st item |
| 0x020C | | Start address[1] | 4 | Start address of user program area 2nd |
| 0x0210 | | Data size[1] | 4 | Data size of user program area 2nd item |
| .<br>.<br>. | | .<br>.<br>. | .<br>.<br>. | Start address of user program area 3-30th<br>Data size of user program area 3-30th item |
| 0x02F4 | | Start address[30] | 4 | Start address of user program area 31th |
| 0x02F8 | | Data size[30] | 4 | Data size of user program area 31th item |
| 0x02FC | | Reserved(0x00) | 4 | - |
| 0x0300 | Application Binary | | N | The encrypted user program |
| 0x0300 +N | Dataflash Binary | | M | Not support. |

Note:1.This parameter is based on the *.rsu file format defined in version 2.x of RX firmware update FIT module and has been modified for the TSIP.

### 4.6.2 **mode** Option

The **mode** option specifies the format of data output as ASCII characters.

Table 4-82 **mode** Option

| ASCII | Description |
|---|---|
| **factory** | Outputs a MOT file, to be used for programming at the **factory**, containing the secure boot program specified by the **prg_sb** option, the encrypted program, and an appended RSU header. Regarding the file image, refer to *Figure 4-3*, *File Image Generated when factory Is Specified for the mode Option*. |
| **update** | Outputs a binary file or MOT file, to be used as a firmware update in the field, containing the encrypted program and an appended RSU header. Regarding the file image, refer to *Figure 4-4*, *File Image Generated when update Is Specified for the mode Option and rsu for the filetype Option*, or *Figure 4-5*, *File Image Generated when update Is Specified for the mode Option and mot for the filetype Option*. |

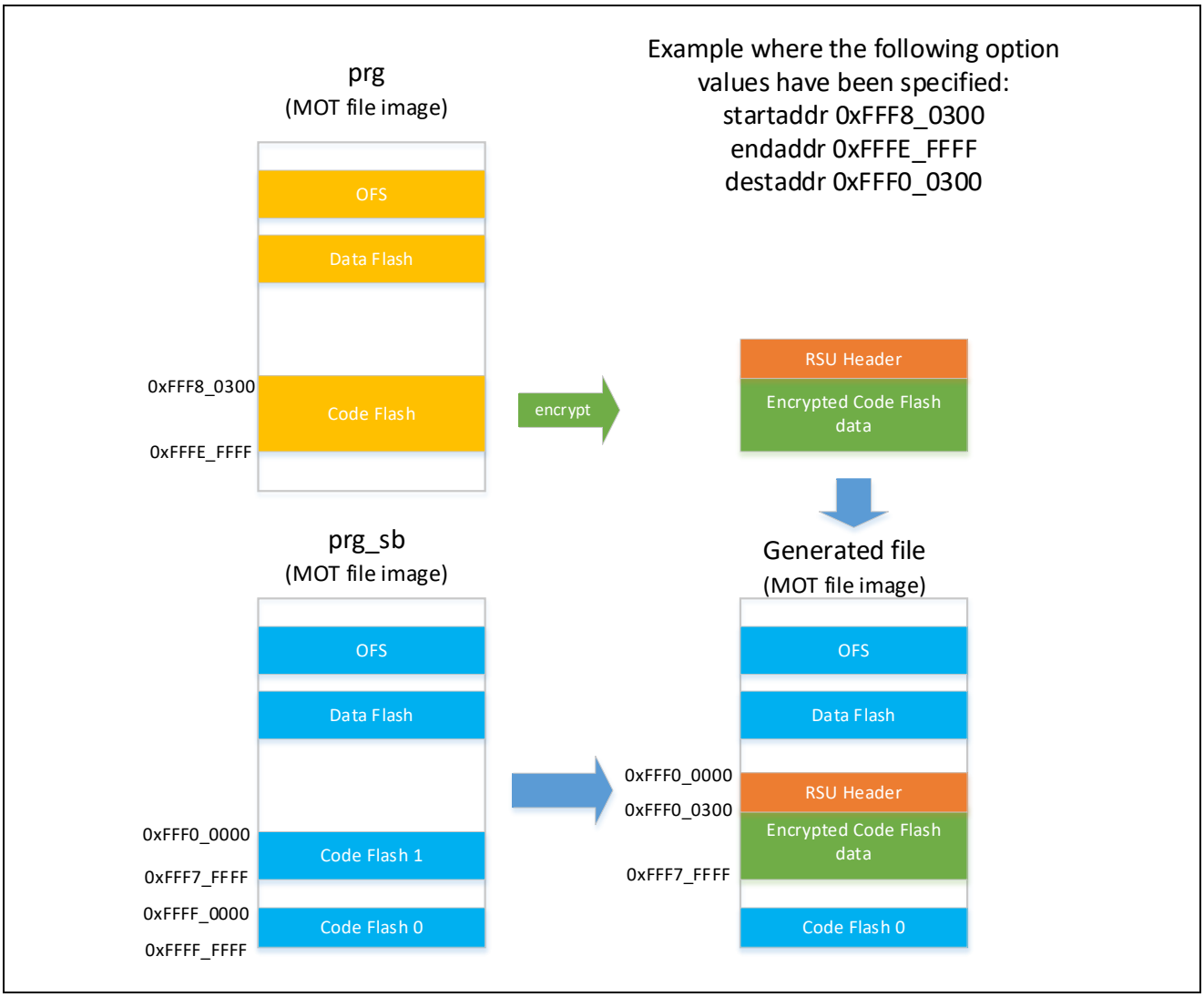


**Figure 4-5 File Image Generated when factory Is Specified for the mode Option**

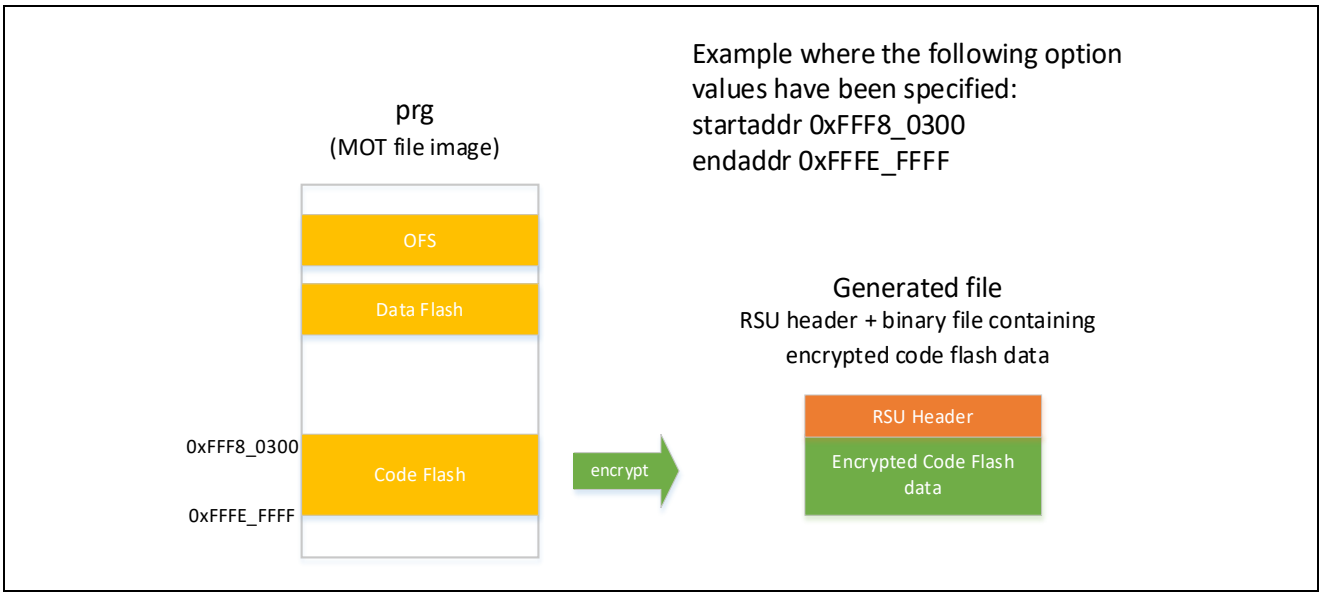


**Figure 4-6 File Image Generated when update Is Specified for the mode Option and rsu for the filetype Option**

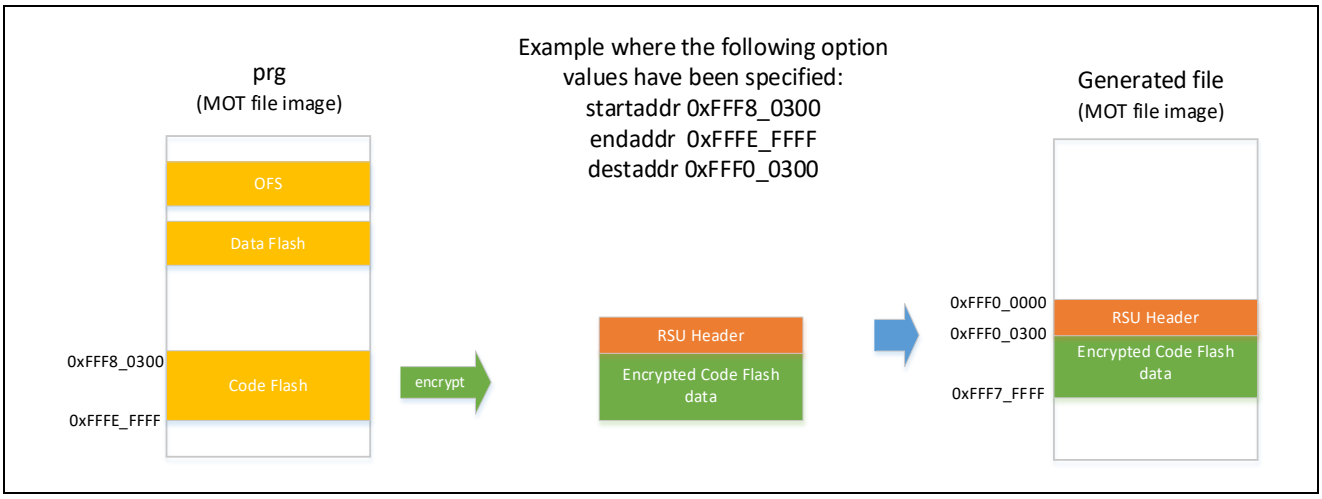


**Figure 4-7 File Image Generated when update Is Specified for the mode Option and mot for the filetype Option**

### 4.6.3 **imgflg** Option

The **imgflg** option specifies the value written to the **Image Flags** field of the RSU header appended to the output file. Either an ASCII or a hexadecimal value can be used for this option.

Table 4-83 **imgflg** Option

| ASCII | Value | Description |
|---|---|---|
| **blank** | 0xFF | No image written. |
| **testing** | 0xFE | Image being updated, tested. |
| **Installing** | 0xFC | Initial image being installed. |
| **valid** | 0xF8 | Application is valid. |
| **invalid** | 0xF0 | Application is invalid. |
| **end_of_life** | 0xE0 | Application has reached end of life. |

### 4.6.4 **filetype** Option

The **filetype** option specifies the format of the output file in ASCII characters. An error occurs if the specified file type and file name extension do not match.

Table 4-84 **filetype** Option

| ASCII | Extension | Description |
|---|---|---|
| **mot** | `*.mot` | Outputs a file in Motorola hex format. |
| **rsu** | `*.rsu` | Outputs binary data consisting of the encrypted user program and an appended RSU header. This value may only be specified when update is specified for the mode option. |

### 4.6.5 **df_ena** Option

The **df_ena** option specifies that the data in the data flash in the mot file specified by the **prg** option is appended to the output file or encrypted and appended to the output file.
When this option is used, data placed in 0x00100000 to 0x0100FFFF of the file specified by the **prg** option is treated as data in a data flash.
When **factory** is specified with the **mode** option, an error occurs if the address overlaps with the data in the Data Flash in the file specified with the **prg_sb** option.

Table 4-85 **df_ena** Option

| ASCII | Description |
|---|---|
| **add** | Appends to the output file the data flash data in the mot file specified by the **prg** option.<br>This can be specified only when **factory** is specified with the **mode** option. |
| **enc** | Encrypts and appends to the output file the data flash data in the mot file specified by the **prg** option.<br>This can be specified only when 2 is specified with the **ver** option. |

## 4.7 **gencert** command options

The following options are available for the **gencert** command. Data input can be specified as hexadecimal data or as a file. To specify a file, prefix the file path with "file=".

Table 4-86 **gencert** options(1)

| Options | Parameters | Description |
|---|---|---|
| **mode** | ASCII | Specify the method of signing the certificate to validate the OEM Bootloader.<br>This value must be one of the options listed in *Section 4.7.1* mode Option. |
| **loadaddr** | Hex data | Specifies the starting address of the OEM Bootloader. |
| **cfsize** | Hex data | Specifies the Code Flash size of the device to be used.<br>When **oembl_size** is omitted, data within the range from **loadaddr** option address to **cfsize** is targeted for signing.<br>Please refer to *Section 4.7.3 Area subject to OEM Bootloader signature or CRC calculation*. |
| **oembl_size** | Hex data | Specify the size of the OEM Bootloader. The size should be 16-byte aligned.<br>Please refer to *Section 4.7.3 Area subject to OEM Bootloader signature or CRC calculation.* |
| **ver** | Decimal data | This option is required only when mode is signature.<br>Specify the version of OEM Bootloader to be listed in Code Certificate.<br>1-4,294,967,295 can be specified.<br>The actual available version depends on the MCU/MPU specifications; please refer to the Hardware Manual or Application Notes of the MCU/MPU. |
| **oembl** | File Path | Specify the Motorola hex file of OEM Bootloader to be verified by FSBL. |
| **oembl_private** | Hex data / File Path | This option is required when mode is signature. Specify the OEM Bootloader Private Key.<br>If HEX data is specified, enter 32-byte HEX data.<br>For file input, you can specify a binary (`*.key`) or text file (`*.txt`) of the OEM Bootloader Private Key, or a PEM file (`*.pem`) containing private and public key data.<br>If this option is omitted, the tool internally generates the OEM Bootloader's secp256r1 key pair. |
| **oembl_public** | Hex data / File Path | This option is required when mode is signature. Specify the OEM Bootloader Public Key.<br>If HEX data is specified, enter 64-byte HEX data.<br>In case of file input, binary (`*.key`) or text file (`*.txt`) of Qx and Qy key data can be specified. For HEX data, binary files, or text files, enter the concatenated data of Qx and Qy (Qx \|\| Qy).<br>No input is required when entering PEM files in **oembl_private**. |

Table 4-87 **gencert** options(2)

| Options | Parameters | Description |
|---|---|---|
| **oemroot_private** | Hex data / File Path | This option is required when mode is signature. Specify the OEM Root Private Key.<br>If HEX data is specified, enter 32-byte HEX data.<br>For file input, you can specify a binary (`*.key`) or text file (`*.txt`) of the OEM Bootloader Private Key, or a PEM file (`*.pem`) containing private and public key data. |
| **oemroot_public** | Hex data / File Path | This option is required when mode is signature. Specify the OEM Root Public Key.<br>If HEX data is specified, enter 64-byte HEX data.<br>In case of file input, binary (`*.key`) or text file (`*.txt`) of Qx and Qy key data can be specified. For HEX data, binary files, or text files, enter the concatenated data of Qx and Qy (Qx \|\| Qy).<br>No input is required when entering PEM files in **oemroot_private**. |
| **measurement_report** | - | If this option is specified and the key certificate and code certificate are successfully generated, the measurement_report value is calculated from the OEM_BL and code certificate used for generation and output. |
| **oembl_endian** | ASCII | This option is used to swap the data during signature generation if the OEM_BL specified in **oembl** was built in big-endian format.<br>If omitted, the default setting is little-endian.<br>For details, see 4.7.2 **oembl_endian** option. |
| **output_codecert** | File Path | Specify the output file name of Code Certificate. |
| **output_keycert** | File Path | This option is required when mode is signature.<br>Specify the output file name of Key Certificate. |
| **keyfileoutput** | File Path | If mode is signature and the **oembl_private** option is omitted, the tool outputs the secp256r1 key pair file generated and used by the tool. Specify the output file name. |

Note: The specific amount of randomness of the random values generated by this tool is not guaranteed.

**Example of use when mode CRC is set :**

```
> skmt.exe /gencert /mode "CRC" /loadaddr "02000000" /oembl_size "1000"
/oembl "D:\example\user.mot" /output_codecert "D:\example\ccert.bin"
```

**Example of use when mode signature is set :**

```
> skmt.exe /gencert /mode "signature" /loadaddr "02000000" /cfsize "200000" /ver "1"
/oembl "D:\example\user.mot" /oembl_private file="D:\example\is_ec256_priv.pem"
/oemroot_private file="D:\example\ms_ec256_priv.pem"
/output_codecert "D:\example\ccert.bin" /output_keycert " D:\example\kcert.bin"
```

### 4.7.1 **mode** Option

The **mode** option specifies the type of Certificate to output in ASCII characters.

Table 4-88 **mode** options

| ASCII | Description |
|---|---|
| **signature** | Generate a Key Certificate and Code Certificate using the key to authenticate the OEM Bootloader (OEM Bootloader Key) and the key to authenticate the OEM Bootloader Key (OEM Root Key).<br>OEM ROOT key pair with **oemroot_private** and **oemroot_public** options, and OEM Bootloader key pair with **oembl_private** and **oembl_public** options. |
| **CRC** | Calculates the CRC value of the OEM Bootloader and generates only a Code Certificate. |

Note: When Code Certificate is generated with **CRC** specified, the CRC value is output as a dummy value in the Signer ID. If select "CRC + report measurement" in the FSBL operation mode and want to output a measurement report using the OEM Bootloader Key, select **signature** option to generate a Code Certificate.

### 4.7.2 **oembl_endian** option

The **oembl_endian** option is used to swap data during signature generation if the OEM_BL specified by the **oembl** option was built in big-endian format.

Table 4-89 **oembl_endian** option

| ASCII | Description |
|---|---|
| **32-big** | Configure this setting when building OEM_BL in big-endian format.<br>The signature is generated after the data order is swapped by 4 bytes. |
| **32-little** | Configure this setting when building OEM_BL in little-endian mode.<br>The signature is generated without swapping the data.<br>If omitted, the default setting is **32-little**. |

### 4.7.3 Area subject to OEM Bootloader signature or CRC calculation

The signature or CRC calculation target of OEM Bootloader is the mot file specified by **oembl** option and the image part extracted from the area specified by **oembl_size** and **cfsize** options.

#### 4.7.3.1 **oembl_size** option is specified

The image of **oembl_size** from the address specified by **loadaddr** option is signed or subjected to CRC operation. If there is no data for **oembl_size** in the input mot file, it is filled with 0xFF.

As an example, the following examples are illustrated when **loadaddr** "02000000" and **oembl_size** "4000" are specified.

example1: Input mot image is only up to 0x02002FFF

example2: Input mot image is up to 0x02004FFF

example3: No data exists from 0x02002000 to 0x02002FFF in the input mot image.

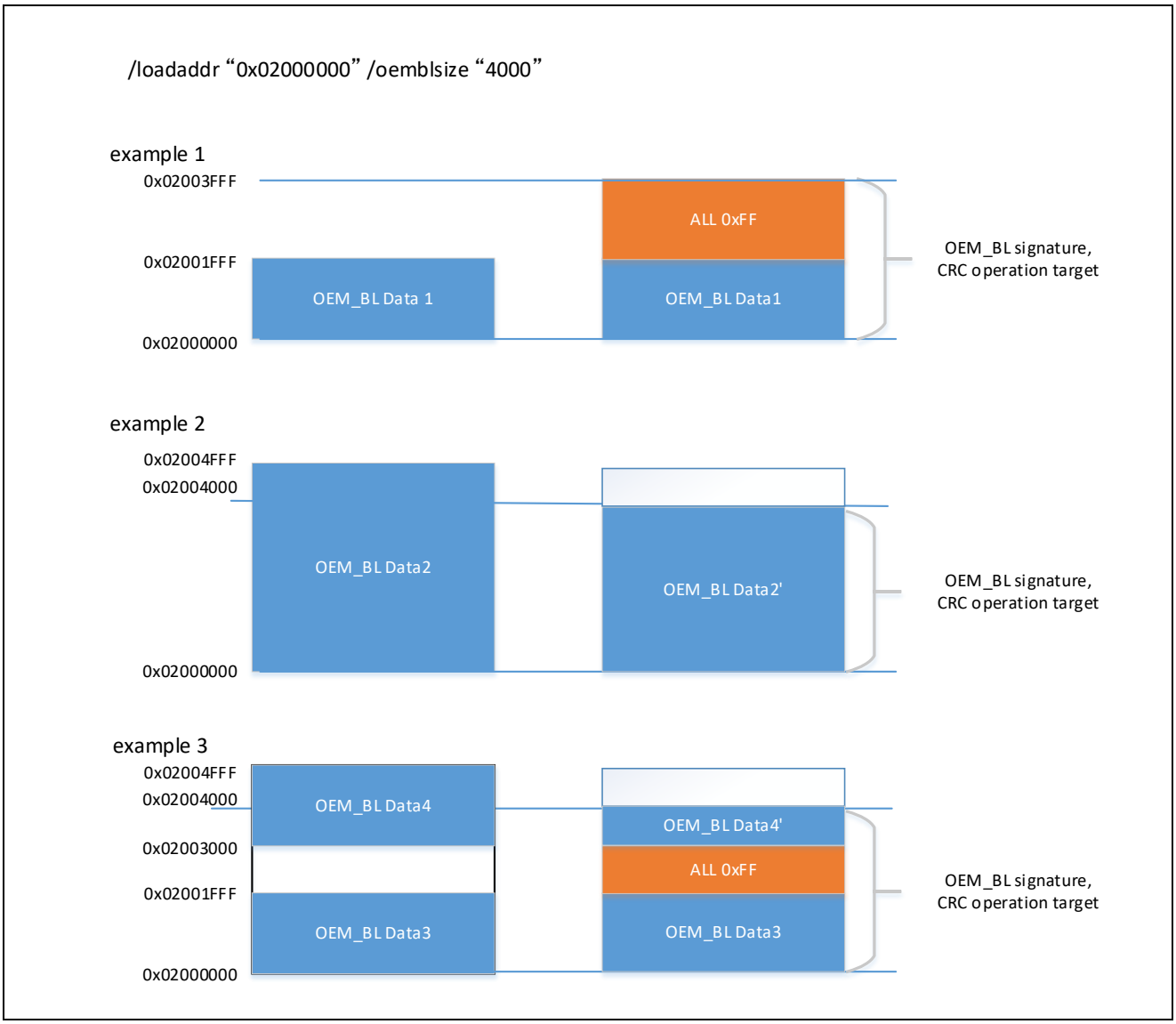


**Figure 4-8 Signature and CRC calculation target when oembl_size option is specified**

### 4.7.3.2 only **cfsize** option is specified

If **oembl_size** option is omitted, data in the mot file specified by **oembl** option, from the address specified by **loadaddr** option to the area specified by **cfsize**, is signed and subject to CRC calculation.

example 1: Input mot image is only up to 0x02001FF8

example 2: Input mot image is up to 0x020014FFF

example 3: Data also exists after 0x02010000 in the input mot image.

example 4: There is an area in the input mot image where no data exists up to 0x02010000.

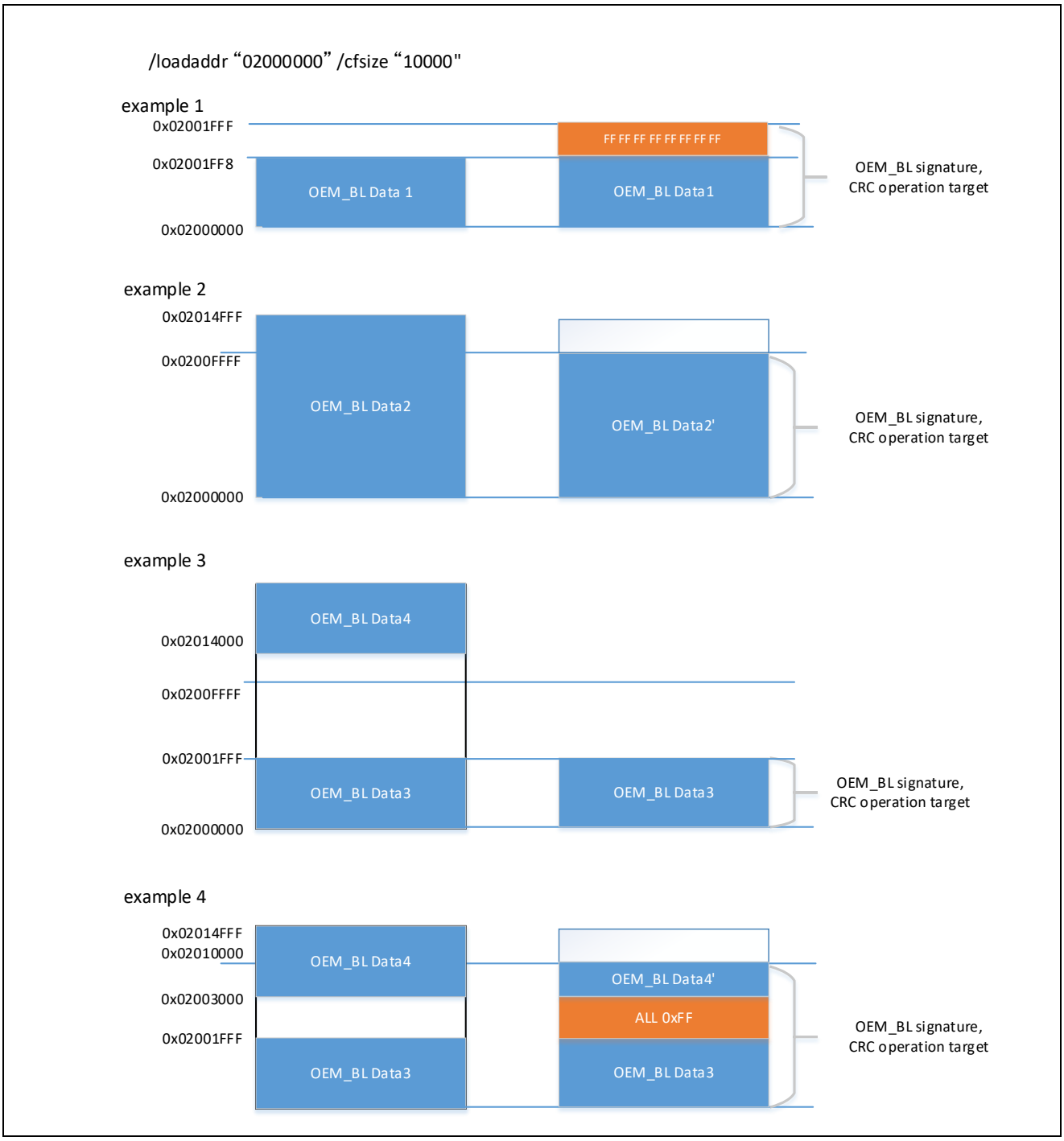


**Figure 4-9 Signature and CRC calculation target when only cfsize option is specified**

## 4.8 **encdotf** command options

The following options are available for **encdotf** command. Data input can be specified as hexadecimal data or as a file. To specify a file, prefix the file path with "file=".

Table 4-90 **encdotf** options

| Options | Parameters | Description |
|---|---|---|
| **keytype** | ASCII | Specify the bit length of the AES cipher to be used.<br>This value must be one of the option listed in *Section 4.8.1* keytype option. |
| **enckey** | Hex data/<br>File Path | Specifies the key data or key file with the bit length specified by **keytype** option.<br>For details, refer to *Section 4.8.2 enckey option*. |
| **nonce** | Hex data | Specify the nonce data to be used for encryption. Data size is 16 bytes. The upper 100 bits or the upper 64 bits of the input data are used as the nonce. The specifications for the nonce and counter used during encryption vary depending on MCU / MPU you are using.<br>This option is optional. If this option is omitted, a random value generated in the tool will be used as the nonce. |
| **startaddr** | Hex data | Specifies the starting address for encryption. The address size is 32 bits.<br>This option can be omitted. If this option is omitted, the entire file is encrypted. The address must be aligned to a 16-byte boundary. |
| **endaddr** | Hex data | Specifies the end address for encryption. The address size is 32 bits.<br>This option can be omitted. If this option is omitted, the entire file is encrypted.<br>Specify the address so that the size of the area specified by **startaddr** option to **endaddr** option is in units of 16 bytes. |
| **prg** | File Path | Specify the file to be encrypted. |
| **output** | File Path | Specifies the output file name. |
| **motaddr0** | None | When this option is specified, the start address of the output mot file address is set to 0.<br>When this option is specified, the **incplain** option or **destaddr** option cannot be specified. |
| **incplain** | None | When **startaddr** or **endaddr** option is specified, data other than that to be encrypted is attached to the output file and output to the file.<br>When this option is specified, **motaddr0** option cannot be specified. |
| **destaddr** | Hex data | Specifies the destination address of the data to be encrypted.<br>The destination address is used for the lower 28 bits of the counter during encryption.<br>This option can be omitted. If this option is omitted, the first address of the encryption range is used as the destination address.<br>When this option is specified, **motaddr0** option cannot be specified. |
| **dotftype** | Decimal data | Specifies the DOTF encryption specification.<br>For details, see *Section 4.8.3* ***dotftype*** *option*.<br>This option is optional. If omitted, the value **1** is selected. |
| **iv_tlv_output** | File Path | Outputs a file containing the IV values used during encryption.<br>For details, see section 4.8.4 **iv_tlv_output** option. |

Note: The specific amount of randomness of the random values generated by this tool is not guaranteed.

**Example of encryption range specification：**

```
> skmt.exe /encdotf /keytype "AES-128" /enckey "000102030405060708090A0B0C0D0E0F"
  /nonce "00112233445566778890000000" /startaddr "80000000" /endaddr "80000FFF"
  /prg "D:\work\program.srec" /incplain /output "D:\work\dotf_program.srec"
```

**Example of not specifying the encryption range：**

```
> skmt.exe /encdotf /keytype "AES-128" /enckey "000102030405060708090A0B0C0D0E0F"
  /nonce "00112233445566778890000000" /prg "D:\work\program.srec"
  /output "D:\work\dotf_program.srec"
```

### 4.8.1 **keytype** option

**keytype** option specifies the key length, in ASCII, of the key to be used for encryption.

Table 4-91 Key length for encryption

| ASCII | Description |
|---|---|
| **AES-128** | Set an AES key with a key length of 128 bits. |
| **AES-192** | Set an AES key with a key length of 192 bits. |
| **AES-256** | Set an AES key with a key length of 256 bits. |

### 4.8.2 **enckey** option

**enckey** option specifies the hexadata or file.

Input data or a binary file or text file in the following format according to the ASCII characters specified by the keytype option. For data or text file input, one byte is represented as a hex ASCII string.

Table 4-92 AES-128

| Bytes | Data |
|---|---|
| 0-15 | AES-128bit key data |

Table 4-93 AES-192

| Bytes | Data |
|---|---|
| 0-23 | AES-192bit key data |

Table 4-94 AES-256

| Bytes | Data |
|---|---|
| 0-31 | AES-256bit key data |

### 4.8.3 **dotftype** option

Because the DOTF IPs implemented in the RA (excluding RA for Wireless), RX, and RZ families differ from those in RA for Wireless, the encryption key length and the enabled bits for the nonce option vary. The **dotftype** option is used to determine the encryption method.

Table 4-95 **dotftype** option

| Value | MCU/MPU | AES key length | nonce valid bits |
|---|---|---|---|
| **1** | RA (excluding RA for Wireless), RX, and RZ families | 128 / 192 / 256 bits | The upper 100bits |
| **2** | RA for Wireless | 256 bits | The upper 64bits |

### 4.8.4 **iv_tlv_output** option

If this option is specified, the IV data will be output as a binary file in TLV format.
The file format is as follows:

Table 4-96 IV File Format

| Field | Size(byte) | Value | Note |
|---|---|---|---|
| Type | 2 | 0x00A1 | Output in little-endian order. |
| Length | 2 | 0x0010 | Output in little-endian order. |
| Value | 16 | IV | IV (counter) value used during DOTF encryption. |

## 4.9 **encsfp** command options

The following options are available for **encsfp** command. Data input can be specified as hexadecimal or decimal data or binary/text file or Renesas Partition Data file. To specify a file, prefix the file path with "file=".

Secure Factory Programming is not supported by all MCU/MPUs. Please refer to the Hardware Manual of the MCU/MPU and the application note of the boot firmware for support availability or encryption area.

Table 4-97 **encsfp** options(1)

| Options | Parameters | Description |
|---|---|---|
| **mcu** | ASCII | Specifies MCU map information that supports Secure Factory Programming.<br>This value must be one of the options listed in *Section 4.9.1* mcu . |
| **enckey** | Hex data / File Path | Specify the AES 128-bit key data to be used for encryption of user program, parameter information and the wrapped key (AL2_KEY, AL1_KEY, SECDBG_KEY and NONSECDBG_KEY). The data size is 16 bytes.<br>This option can be omitted. If this option is omitted, a random value generated in the tool is used as **enckey**. |
| **nonce_prm** | Hex data | Specify the nonce data to be used for encryption of parameter information. The data size is 12 or 11 bytes.<br>The byte length of the nonce that can be used varies depending on the **mcu** options set.<br>This option can be omitted. If this option is omitted, a random value generated in the tool is used as **nonce_prm**. |
| **nonce_prg** | Hex data | Specify the nonce data to be used for encryption of parameter information. The data size is 12 or 11 bytes.<br>The byte length of the nonce that can be used varies depending on the **mcu** options set.<br>This option can be omitted. If this option is omitted, a random value generated in the tool is used as **nonce_prg.** |
| **nonce_key** | Hex data | Specify the nonce data to be used in the encryption of the wrapped AL2_KEY, AL1_KEY, SECDBG_KEY and NONSECDBG_KEY. The data size is 12 or 11 bytes.<br>The byte length of the nonce that can be used varies depending on the **mcu** options set.<br>This option can be omitted. If this option is omitted, a random value generated in the tool is used as **nonce_key.** |
| **trn** | ASCII / Hex data | Specifies DLM transition destination information.<br>This value must be one of the options listed in *Section 4.9.2 trn option*. |
| **prg** | File Path | Specify the user program (mot file) to be encrypted.<br>Up to three user programs can be specified.<br>See *Section 4.9.4 prg option* for details. |

Table 4-98 **encsfp** options(2)

| Options | Parameters | Description |
|---|---|---|
| **al2key** | Hex data / File Path | Specify AL2_KEY. Data size is 16 bytes.<br>When **OEM_PL0_AL2** and **OEM_PL0_AL2_1** are specified with **trn** option, this option must be entered.<br>If this option is omitted when **OEM_PL0_AL2** and **OEM_PL0_AL2_1** are specified in **trn** option, a random value generated in the tool is used as AL2_KEY. |
| **al1key** | Hex data / File Path | Specify AL1_KEY. Data size is 16 bytes.<br>When **OEM_PL0_AL2_1** is specified with **trn** option, this option must be entered.<br>If this option is omitted when **OEM_PL0_AL2_1** is specified in **trn** option, a random value generated in the tool is used as AL1_KEY. |
| **secdbgkey** | Hex data / File Path | Specify SECDBG_KEY. Data size is 16 bytes.<br>When **DPL_SECDBG_NONSECDBG** is specified with **trn** option, this option must be entered.<br>If this option is omitted when **DPL_SECDBG_NONSECDBG** is specified in trn option, a random value generated in the tool is used as SECDBG_KEY. |
| **nonsecdbgkey** | Hex data / File Path | Specify NONSECDBG_KEY. Data size is 16 bytes.<br>When **DPL_SECDBG_NONSECDBG** is specified with **trn** option, this option must be entered.<br>If this option is omitted when **DPL_SECDBG_NONSECDBG** is specified in **trn** option, a random value generated in the tool is used as NONSECDBG_KEY. |
| **ufpk** | Hex data / File Path | Specify the UFPK value to be used when encrypting **enckey**, **al2key**, **al1key**, **secdbgkey** and **nonsecdbgkey** or a UFPK key file generated by the **genufpk** command. |
| **wufpk** | File Path | Specify the W-UFPK file containing the UFPK values wrapped by Renesas Key Wrapping service. |
| **iv_enckey** | Hex data | Specify the IV value to be used during **enckey** encryption. The data size is 16 bytes.<br>This option can be omitted. If this option is omitted, a random value generated in the tool is used as the IV. |
| **iv_al2key** | Hex data | When **OEM_PL0_AL2** is specified in **trn** option, specify the IV value to be used when encrypting **al2key**. The data size is 16 bytes.<br>This option can be omitted. If this option is omitted, a random value generated in the tool is used as IV. |
| **iv_secdbgkey** | Hex data | When **DPL_SECDBG_NONSECDBG** is specified in **trn** option, specify the IV value to be used when encrypting **secdbgkey**. The data size is 16 bytes.<br>This option can be omitted. If this option is omitted, a random value generated in the tool is used as IV. |
| **iv_ nonsecdbgkey** | Hex data | When **DPL_SECDBG_NONSECDBG** is specified in **trn** option, specify the IV value to be used when encrypting **nonsecdbgkey**. The data size is 16 bytes.<br>This option can be omitted. If this option is omitted, a random value generated in the tool is used as IV. |

Note: The specific amount of randomness of the random values generated by this tool is not guaranteed.

Table 4-99 **encsfp** options(3)

| Options | Parameters | Description |
|---|---|---|
| **num_al2key** | Decimal data | Specify the number of keys when specifying multiple AL2_KEY entries.<br>If omitted, the value is set to 1.<br>For details, see 4.9.7 **num_al2key**/**num_al1key** options. |
| **num_al1key** | Decimal data | Specify the number of keys when specifying multiple AL1_KEY entries.<br>If omitted, the value is set to 1.<br>For details, see 4.9.7 **num_al2key**/**num_al1key** options. |
| **boundary** | Decimal data / File Path | Specify the boundary information for code flash(secure, non-secure-callable), data flash and SRAM (secure, non-secure callable). Specify five decimal arguments or a Renesas Partition Data File (`*.rpd`).<br>This option can be omitted. If omitted, the boundary information will not be set. For details, see *Section 4.9.5* ***boundary*** *option* |
| **extarea0** | Hex data, Decimal data | Specify the address and write unit of the external flash memory area. Following this option, enter the “start address (hexadecimal)”, “end address (hexadecimal)” and “write unit (decimal)” in that order. Specify the external flash memory area and write unit in 16-byte units. This option is not required if the external flash memory is not included in the encryption target. For details, see *Section 4.9.6* ***extarea0/extarea1*** *options*. |
| **extarea1** | Hex data, Decimal data | Specify the address and write unit of the external flash memory area. Following this option, enter the “start address (hexadecimal)”, “end address (hexadecimal)” and “write unit (decimal)” in that order. Specify the external flash memory area and write unit in 16-byte units. This option is not required if the external flash memory is not included in the encryption target. For details, see *Section 4.9.6* ***extarea0/extarea1*** *options*. |
| **endian** | ASCII | Configure the settings according to the MCU/MPU's operating mode. This option is optional. If omitted, “little” is set by default.<br>**little** : Configures the system for little-endian operation.<br>**big** : Configures the system for big-endian operation. |
| **output** | File Path | Specify the output file name (sfp file).<br>For the format of the sfp file, see Appendix Secure Factory Programming File Format. |
| **output_al2key** | File Path | If **OEM_PL0_AL2_1 and OEM_PL0_AL2** are specified with **trn** option and the **al2key** option is omitted, the random value generated in the tool used as the key is output as the key file (key file). |
| **output_al1key** | File Path | If **OEM_PL0_AL2_1** is specified with **trn** option and the **al1key** option is omitted, the random value generated in the tool used as the key is output as the key file (key file). |
| **output_secdbgkey** | File Path | If **DPL_SECDBG_NONSECDBG** is specified with **trn** option and the **secdbgkey** option is omitted, the random value generated in the tool used as the key is output as the key file (key file). |
| **output _nonsecdbgkey** | File Path | If **DPL_SECDBG_NONSECDBG** is specified with **trn** option and the **nonsecdbgkey** option is omitted, the random value generated in the tool used as the key is output as the key file (key file). |
| **output_enckey** | File Path | If **enckey** option is omitted, the random value generated within the tool used as the key is output as a key file (key file). |

**Specified mcu option RA8D1_M1_T1 and trn option OEM_PL0_AL2 :**

```
> skmt /encsfp /mcu "RA8D1_M1_T1" /enckey "000102030405060708090a0b0c0d0e0f"
  /nonce_prg "111111111111222222222222" /nonce_prm "333333333333444444444444"
  /trn "OEM_PL0_AL2" /prg "D:\work\program.mot"
  /al2key "77777777777777778888888888888888"
  /ufpk file="ufpk.key" /wufpk "ufpk.key_enc.key"
  /iv_enckey "9999999999999999aaaaaaaaaaaaaaaa"
  /iv_al2key "dddddddddddddddddeeeeeeeeeeeeeeee"
  /output " D:\work\ program.sfp"
```

**Specified mcu option RA4L1 and trn option DPL_SECDEBG_NONSECDBG :**

```
> skmt /encsfp /mcu "RA4L1" /enckey "000102030405060708090a0b0c0d0e0f"
  /nonce_prg "111111111111222222222222" /nonce_prm "333333333333444444444444"
  /nonce_key "555555555555666666666666"
  /trn "DPL_SECDEBG_NONSECDBG" /prg "D:\work\program.mot"
  /secdbgkey "77777777777777778888888888888888"
  /nonsecdbgkey "9999999999999999aaaaaaaaaaaaaaaa"
  /ufpk file="ufpk.key" /wufpk "ufpk.key_enc.key"
  /iv_enckey " dddddddddddddddddeeeeeeeeeeeeeeee"
  /iv_secdbgkey "ffffffffffffffff0000000000000000"
  /iv_nonsecdbgkey "00000000000000001111111111111111"
  /boundary "3" "29" "0" "6" "2"
  /output " D:\work\ program.sfp"
```

**Specified mcu option RA8D1_M1_T1 and trn option LCK_BOOT :**

```
> skmt /encsfp /mcu "RA8D1_M1_T1" /enckey "101112131415161718191a1b1c1d1e1f"
  /nonce_prg "111111111111222222222222" /nonce_prm "333333333333444444444444"
  /trn "LCK_BOOT" /prg "D:\work\program.mot"
  /ufpk file="ufpk.key" /wufpk "ufpk.key_enc.key"
  /iv_enckey "9999999999999999aaaaaaaaaaaaaaaa" /output "D:\work\ program.sfp"
```

### 4.9.1 **mcu** option

**mcu** option specifies MCU information in ASCII characters.

Table 4-100 **mcu** option

| ASCII | Description |
|---|---|
| **RA8D1_M1_T1** | Specifies map information for RA8D1/M1/T1. |
| **RA8D2_M2_T2_P1** | Specifies map information for RA8D2/M2/T2/P1. |
| **RA8E1_E2** | Specifies map information for RA8E1/E2. |
| **RA6B1_U1** | Specifies map information for RA6B1/RA6U1. The boundary option can be specified. |
| **RA4C1** | Specifies map information for RA4C1. The boundary option can be specified. |
| **RA4L1** | Specifies map information for RA4L1. The boundary option can be specified. |
| **RX74M_N** | Specifies map information for RX74M/N. |

### 4.9.2 **trn** option

**trn** option specifies DLM transition information in ASCII characters.

Table 4-101 DLM Transition Destination and Authentication Key

| ASCII | Description |
|---|---|
| **OEM_PL0_AL2_1** | Write AL2_KEY and AL1_KEY, and transfer DLM to OEM, Protection Level to PL0. |
| **OEM_PL0_AL2** | Write AL2_KEY and transfer DLM to OEM, Protection Level to PL0. |
| **DPL_SECDBG_NONSECDBG** | Write SECDBG_KEY and NONSECDBG_Key and transfer DLM to DPL. |
| **LCK_BOOT** | Transfer DLM to LCK_BOOT |

Table 4-102 **trn** option and **mcu** option that can be specified

| **mcu** option | **trn** option | | | |
|---|---|---|---|---|
| | **OEM_PL0_AL2_1** | **OEM_PL0_AL2** | **DPL_SECDBG_NONSECDBG** | **LCK_BOOT** |
| **RA8D1_M1_T1** | - | ✓ | - | ✓ |
| **RA8D2_M2_T2_P1** | ✓ | ✓ | - | ✓ |
| **RA8E1_E2** | - | ✓ | - | ✓ |
| **RA6B1_U1** | ✓ | ✓ | - | ✓ |
| **RA4C1** | - | - | ✓ | ✓ |
| **RA4L1** | - | - | ✓ | ✓ |
| **RX74M_N** | ✓ | ✓ | - | ✓ |

✓ : available - : N/A

### 4.9.3 **nonce_trn, nonce_prg, nonce_key** options

The byte lengths that can be set for **nonce_trn**, **nonce_prg**, and **nonce_key** options vary depending on the options specified in **mcu** option. The size of the image that can be encrypted depends on the byte length that can be set for **nonce_prg** option. For details, refer to *Section 7.5.1 Image Size Limitation for Secure Factory Programming Function*.

Table 4-103 **trn** option and configurable byte length for **nonce_trn, nonce_prg, and nonce_key**

| **mcu** option | Byte length | | |
|---|---|---|---|
| | **nonce_trn** | **nonce_prg** | **nonce_key** |
| **RA8D1_M1_T1, RA8D2_M2_T2_P1, RA8E1_E2, RA4C1, RA4L1, RX74M_N** | 12 | 12 | 12 |
| **RA6B1_U1** | 11 | 11 | 11 |

### 4.9.4 **prg** option

The **prg** option specifies the user program to be encrypted by the Secure Factory Programming function. Up to three **prg** options can be specified.

**prg option 3 file specification example :**

```
> skmt /encsfp /mcu "RA8D1_M1_T1" /enckey "101112131415161718191a1b1c1d1e1f"
  /nonce_prg "111111111111222222222222" /nonce_prm "333333333333444444444444"
  /trn "LCK_BOOT"
  /prg "D:\work\program1.mot" "D:\work\program2.mot" "D:\work\program3.mot"
  /ufpk file="ufpk.key" /wufpk "ufpk.key_enc.key"
  /iv_enckey "9999999999999999aaaaaaaaaaaaaaaa" /output " D:\work\ program.sfp"
```

### 4.9.5 **boundary** option

Specifies boundary information for Secure, and Non-Secure Callables for Code NVM area , SRAM and Data Flash. The method of specification can be input by argument or file input.

Table 4-104 **boundary** option and **mcu** option that can be specified

| **mcu** option | **boundary** option | | | | |
|---|---|---|---|---|---|
| | Code Secure Size | Code Non-secure Callable Size | Data Secure Size | SRAM Secure Size | SRAM Non-secure Callable Size |
| **RA6B1_U1** | ✓ | - | - | | - |
| **RA4C1** | ✓ | ✓ | ✓ | ✓ | ✓ |
| **RA4L1** | ✓ | ✓ | ✓ | ✓ | ✓ |
| other | - | - | - | - | - |

✓ : available - : N/A

#### 4.9.5.1 Argument input

Specify the size of each area in KB units in the following order from the top.
Code Secure Size, Code Non-secure Callable Size, Data Secure Size, SRAM Secure Size, SRAM Non-secure Callable Size.

For example :
/boundary “128” “64” “0” “16” If “8” is specified, the following settings are used, respectively.
Code Secure Size : 128KB
Code Non-secure Callable Size : 64KB
Data Secure Size : 0KB
SRAM Secure Size : 16KB
SRAM Non-secure Callable Size: 8KB

#### 4.9.5.2 File input

The following parameters are read from the Renesas Partition Data File (`*.rpd`) generated when a project is generated in e²studio and specified as boundary information, The parameters in the Renesas Partition Data File are in Bytes, so they are converted to KB units.

Table 4-105 Parameter for Renesas Partition Data File

| Renesas Partition Data File Parameter [Byte] | Boundary information [KB] |
|---|---|
| FLASH_S_SIZE | Code Secure Size |
| FLASH_C_SIZE | Code Non-secure Callable Size |
| DATA_FLASH_S_SIZE | Data Secure Size |
| RAM_S_SIZE | SRAM Secure Size |
| RAM_C_SIZE | SRAM Non-secure Callable Size |

### 4.9.6 **extarea0/extarea1** options

The Security Key Management Tool supports encryption in 16-byte units for external flash memory areas defined in the boot firmware application note for MCU/MPUs that support the Secure Factory Programming function. With this option, the external flash memory area address and the write unit of the flash memory to be used can be set according to the write unit of the external flash memory to be used. Two areas can be specified using the extarea0 and extarea1 options.
The designation method is as follows: “start address (hex),” “end address (hex),” and “write unit (decimal). Enter “Start address (hex)”, “End address (hex)” and “Write unit (decimal)” in this order. To specify only one external flash memory area, use the extarea0 option.

The Security Key Management Tool divides the specified external flash memory area by the specified write and encrypts the write unit block with data. If only a part of the write unit block has data, the address with no data is complemented with 0xFF to encrypt the data.

**External flash memory 1 area setting**

**Area 0 : 0x90000000-0x903FFFFF 64-byte write unit is specified :**

```
> skmt /encsfp /mcu "RA8D1_M1_T1" /enckey "101112131415161718191a1b1c1d1e1f"
  /nonce_prg "111111111111222222222222" /nonce_prm "333333333333444444444444"
  /trn "LCK_BOOT"
  /prg "D:\work\program.mot" "D:\work\program_ext.mot"
  /ufpk file="ufpk.key" /wufpk "ufpk.key_enc.key"
  /extarea0 "90000000" "903FFFFF" "64"
  /iv_enckey "9999999999999999aaaaaaaaaaaaaaaa" /output " D:\work\ program.sfp"
```

**External flash memory 2 area setting**

**Area 0 : 0x80000000-0x800FFFFF 128-byte write unit is specified**

**Area1 : 0x90000000-0x903FFFFF 64-byte write unit is specified :**

```
> skmt /encsfp /mcu "RA8D1_M1_T1" /enckey "101112131415161718191a1b1c1d1e1f"
  /nonce_prg "111111111111222222222222" /nonce_prm "333333333333444444444444"
  /trn "LCK_BOOT"
  /prg "D:\work\program.mot" "D:\work\program_ext.mot"
  /ufpk file="ufpk.key" /wufpk "ufpk.key_enc.key"
  /extarea0 "80000000" "800FFFFF" "128"
  /extarea1 "90000000" "903FFFFF" "64"
  /iv_enckey "9999999999999999aaaaaaaaaaaaaaaa" /output " D:\work\ program.sfp"
```

### 4.9.7 **num_al2key**/**num_al1key** options

Depending on the MCU/MPU, it may be possible to inject multiple AL2_KEY and AL1_KEY keys. Use this option to specify the number of AL2_KEY and AL1_KEY keys to be injected. When specifying multiple keys, set multiple values for each of the **al2key**, **al1key**, **iv_al2key**, **iv_al1key**, **output_al2key**, and **output_al1key** options. When specifying multiple keys, use the same method for all entries. You cannot configure settings such as specifying the first key from a file, the second key as a value, and the third key as one generated by the tool. Please refer to the example below for configuration instructions. The example shows how to configure AL2_KEY, but please use the same method for AL1_KEY.

Please refer to the MCU/MPU hardware manual for the maximum number of AL2_KEY and AL1_KEY keys that can be injected.

**Example of specifying the 4 AL2_KEY entries (keys specified via file) and specifying IV values :**

```
> skmt /encsfp /mcu "RX74M_N" /enckey "101112131415161718191a1b1c1d1e1f"
/nonce_prg "111111111111222222222222" /nonce_prm "333333333333444444444444"
/trn "OEM_PL0_AL2" /prg "D:\work\program.mot" "D:\work\program_ext.mot"
/ufpk file="ufpk.key" /wufpk "ufpk.key_enc.key"
/num_al2key "4"
/al2_key file=“key1.key” file=“key2.key” file=“key3.key” file=“key4.key”
/iv_al2key “00000000000000001111111111111111” “22222222222222223333333333333333”
“44444444444444445555555555555555” “66666666666666667777777777777777”
/iv_enckey "9999999999999999aaaaaaaaaaaaaaaa" /output " D:\work\ program.sfp"
```

**Example of specifying the 3 AL2_KEY entries (keys specified via value) and omitting IV values :**

```
> skmt /encsfp /mcu "RX74M_N" /enckey "101112131415161718191a1b1c1d1e1f"
/nonce_prg "111111111111222222222222" /nonce_prm "333333333333444444444444"
/trn "OEM_PL0_AL2" /prg "D:\work\program.mot" "D:\work\program_ext.mot"
/ufpk file="ufpk.key" /wufpk "ufpk.key_enc.key"
/num_al2key "3"
/al2_key "00000000000000001111111111111111" "22222222222222223333333333333333"
"44444444444444445555555555555555"
/iv_enckey "9999999999999999aaaaaaaaaaaaaaaa" /output " D:\work\ program.sfp"
```

**Example of specifying the 2 AL2_KEY entries (keys omitting key value) and omitting IV values :**

```
> skmt /encsfp /mcu "RX74M_N" /enckey "101112131415161718191a1b1c1d1e1f"
/nonce_prg "111111111111222222222222" /nonce_prm "333333333333444444444444"
/trn "OEM_PL0_AL2" /prg "D:\work\program.mot" "D:\work\program_ext.mot"
/ufpk file="ufpk.key" /wufpk "ufpk.key_enc.key"
/num_al2key "2"
/iv_enckey "9999999999999999aaaaaaaaaaaaaaaa" /output " D:\work\ program.sfp"
/output_al2key "output1.key" "output2.key"
```

## 4.10 genhdr options

The following options are available with **genhdr** command. Data can be specified as hexadecimal or decimal values, or as a file. When specifying a file, prefix the file path with “file=”.

Table 4-106 **genhdr** options(1)

| Options | Parameters | Description |
|---|---|---|
| **mode** | ASCII | **provisioning** : Generates headers, code certificates, and key certificates for Provisioning SW.<br>**app** : Generates headers, code certificates, and key certificates for OEM_BL and Application Firmware.<br>For details, see *Section 4.10.1* ***mode*** . |
| **header** | ASCII | **sec_sign** : Generates headers, as well as code certificates and key certificates.<br>**sec** : Generates headers.<br>For details, see *Section 4.10.2* ***header*** *option*. |
| **key_sign** | ASCII | Specify the ECC key to be used when generating Key Certificate signature.<br>For details, see *Section 4.10.3* ***key_sign / code_sign*** *option*. |
| **code_sign** | ASCII | Specify the ECC key to be used when generating Code Certificate signature.<br>For details, see *Section 4.10.3* ***key_sign / code_sign*** *option*. |
| **fw_ver** | Hex data | Specify the firmware version of the code certificate in 4 bytes.<br>If **mode** is set to **app**, please specify this option. |
| **ctmr_reserved** | Hex data | Specifies the 16-byte data to be entered into the ctmr_reserved field of the Image Header. If omitted, enter 0. |
| **oem_rot_key_num** | Hex data | Specify the number of the OEM RoT Key to be used in the oem_rot_key field of the key certificate.<br>Valid values range from 0 to 3. |
| **oembl_private_key** | Hex data / File Path | This option is required when **mode** is **app** and **header** is **sec_sign**. Specify OEM_BL verification private key.<br>When specifying HEX data, enter 48 bytes of HEX data if **code_sign** is set to **secp384r1**, and 32 bytes of HEX data if **code_sign** is set to **secp256r1**.<br>When specifying a file, you can provide a binary file (`*.key`) containing the OEM_BL verification private key, a text file (`*.txt`), or a PEM file (`*.pem`) containing the public key data.<br>If this option is omitted, the tool will generate the OEM_BL verification key pair internally. |
| **oembl_public_key** | Hex data / File Path | This option is required when **mode** is set to **app** and **header** is set to **sec_sign**. Specify the OEM_BL verification public key.<br>When specifying HEX data, enter 96 bytes of HEX data if **secp384r1** is specified for **code_sign**, and 64 bytes of HEX data if **secp256r1** is specified for **code_sign**.<br>When specifying a file, you can select either a binary key file (`*.key`) or a text file (`*.txt`).<br>For HEX data, binary files, or text files, enter the concatenated data of Qx and Qy (Qx \|\| Qy).<br>This specification is not required when using **oembl_private_key** PEM file. |

Table 4-107 **genhdr** options(2)

| Options | Parameters | Description |
|---|---|---|
| **oemprov_private_key** | Hex data / File Path | This option is required when **mode** is **provisioning** and **header** is **sec_sign**. Specify OEM provisioning private key.<br>When specifying HEX data, enter 48 bytes of HEX data if **code_sign** is set to **secp384r1**, and 32 bytes of HEX data if **code_sign** is set to **secp256r1**.<br>When specifying a file, you can provide a binary file (`*.key`) containing the OEM provisioning private key, a text file (`*.txt`), or a PEM file (`*.pem`) containing the public key data.<br>If this option is omitted, the tool will generate the OEM provisioning key pair internally. |
| **oemprov_public_key** | Hex data / File Path | This option is required when **mode** is set to **provisioning** and **header** is set to **sec_sign**. Specify the OEM provisioning public key.<br>When specifying HEX data, enter 96 bytes of HEX data if **secp384r1** is specified for **code_sign**, and 64 bytes of HEX data if **secp256r1** is specified for **code_sign**.<br>When specifying a file, you can select either a binary key file (`*.key`) or a text file (`*.txt`).<br>For HEX data, binary files, or text files, enter the concatenated data of Qx and Qy (Qx \|\| Qy).<br>This specification is not required when using **oemprov_private_key** PEM file. |
| **oemroot_private_key** | Hex data / File Path | This option is required when the **header** is set to **sec_sign**. Specify the OEM verification root private key.<br>When specifying HEX data, enter 48 bytes of HEX data if **key_sign** is set to **secp384r1**, and 32 bytes of HEX data if **key_sign** is set to **secp256r1**.<br>When specifying a file, you can specify the binary file (`*.key`) of the OEM verification root private key, a text file (`*.txt`), or a PEM file (`*.pem`) containing the public key data. |
| **oemroot_public_key** | Hex data / File Path | This option is required when the **header** is **sec_sign**. Specify the OEM verification root public key.<br>When specifying HEX data, enter 96 bytes of HEX data if **key_sign** is set to **secp384r1**, and 64 bytes of HEX data if **key_sign** is set to **secp256r1**. When specifying a file, you can use either a binary key file (`*.key`) or a text file (`*.txt`).<br>For HEX data, binary files, or text files, enter the data resulting from concatenating Qx and Qy (Qx \|\| Qy).<br>This specification is not required when entering a PEM file via **oemroot_private_key**. |

Note: The specific amount of randomness of the random values generated by this tool is not guaranteed.

Table 4-108 **genhdr** options(3)

| Options | Parameters | Description |
|---|---|---|
| **ivt** | Hex data | This option is required when **mode** is set to **app**.<br>Specify the IVT (Initial Vector Table) address of the input file.<br>If the input file is a binary file, specify the starting address of the binary file. If the input file is a Motorola Hex file, specify the IVT address (the starting address of the data to be signed). |
| **size** | Hex data | This option is required when **mode** is **app** and the input file is a Motorola Hex file.<br>It specifies the portion of the file up to the size specified by **ivt** as the signing target. If this option is omitted, all data from **ivt** onward will be included in the signature.<br>For details, see *Section 4.10.5* Regarding the Subject of Code Certificate. |
| **prg** | File Path | Specify Provisioning SW, OEM_BL, or Application Firmware to which the headers will be appended.<br>You can specify either a binary file (`*.bin`) or a Motorola Hex file (`*.mot, *.srec`).<br>If **mode** is set to **provisioning**, only binary files can be specified. |
| **output** | File Path | Specify the output file.<br>You can specify either a binary file (`*.bin`) or a Motorola Hex file (`*.mot, *.srec`) as the output file. If **mode** is set to **provisioning**, only binary files can be specified. |
| **keyfileoutput** | File Path | If **header** is **sec_sign** and the **oemprov_private_key** or **oembl_private_key** option is omitted, the tool will output the files containing secp384r1 or secp256r1 key pairs generated and used within the tool. Please specify the output file name. |

**Specified mode provisioning and header sec_sign :**

```
> skmt /genhdr /mode "provisioning" /header "sec_sign" /key_sign "secp384r1"
  /code_sign "secp384r1" /oemroot_private_key file="ecc384_private.pem"
  /oemprov_private_key file="ecc384_private.pem" /oem_rot_key_num "0"
  /prg "provSW.bin" /output "provSW_secsign.bin"
```

**Specified mode provisioning and header sec :**

```
> skmt /genhdr /mode "provisioning" /header "sec"
   /prg "provSW.bin" /output "provSW_sec.bin"
```

**Specified mode app and header sec_sign :**

```
> skmt /genhdr /mode "app" /header "sec_sign" /key_sign "secp384r1" /code_sign "secp384r1"
  /oemroot_private_key file="ecc384_private.pem"
  /oembl_private_key file="ecc384_private.pem" /oem_rot_key_num "0"
```

**/ivt "04002400" /size "10000" /fw_ver "00000001" /prg "sample.mot"**
**/output "sample_sec_sign.mot"**

**Specified mode app and header sec :**

**> skmt /genhdr /mode "app" /header "sec" /ivt "04002400" /size "10000"**
**/fw_ver "00000001" /prg "sample.mot" /output "sample_sec.mot"**

### 4.10.1 **mode** option

**mode** option specifies the type of program to be entered using ASCII characters.

Table 4-109 **mode** option

| ASCII | Description |
|---|---|
| **provisioning** | Generate the header for Provisioning SW |
| **app** | Generate the header for OEM_BL, Application Firmware |

### 4.10.2 **header** option

**header** option specifies the header or certificate to be appended to a program entered in ASCII characters.

Table 4-110 **header** options

| ASCII | Description |
|---|---|
| **sec_sign** | Add Product Header, Image Header, key certificate, and code certificate. |
| **sec** | Add Product Header, and Image. |

Note : The Product Header is added when **mode app** option is specified.

### 4.10.3 **key_sign / code_sign** options

When **sec_sign** is specified in **header**, specify in ASCII the type of key to be used for signing during the generation of key certificates and code certificates.

Table 4-111 **key_sign / code_sign** options

| ASCII | Description |
|---|---|
| **secp256r1** | Generate a signature using the secp256r1 key. |
| **secp384r1** | Generate a signature using the secp384r1 key. |

### 4.10.4 **prg** Option

When **mode** “**app**” is specified, if the beginning of a file specified with the **prg** option starts with 0x70, 0x72, 0x6F, or 0x64 (ASCII for “prod”), the system determines that a product header has been appended. It deletes the data from the beginning of the file up to the address specified by the **IVT** option and re-appends the header.

### 4.10.5 Regarding the Subject of Code Certificate

When **header** "**sec_sign**" header is specified, the signature value of the code certificate covers a portion of the code certificate plus the input firmware image.

If the input firmware image is a binary file, the signature covers a portion of the code certificate plus the entire binary data of the input firmware image.

If the input firmware image is a Motorola Hex file, the behavior differs depending on whether the **size** option is specified or not.

・If the **size** option is not specified

The signature covers the portion of the code certificate plus the address specified by the **ivt** option, up to the end of the data. If there are areas where no data exists, those areas are filled with 0xFF before the signature is applied. Additionally, the output file will be filled with 0xFF.

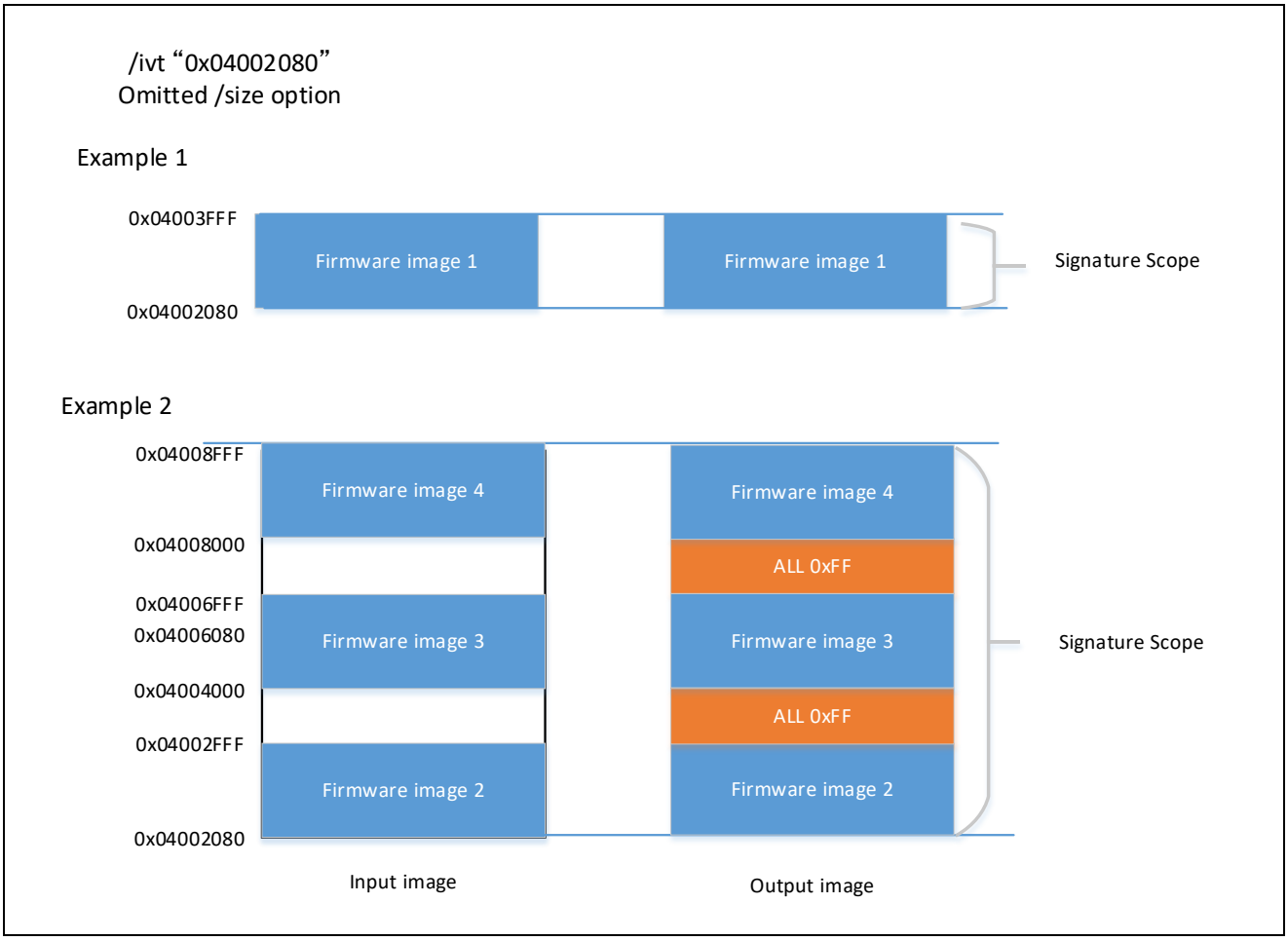


**Figure 4-10 Signature Scope and Output Images When the size Option Is Omitted When Specifying a Motorola Hex File for the Input Firmware Image**

・When specifying the **size** option

The area specified by **size**—starting from the address specified by the **ivt** option and including a portion of the code certificate—will be signed. If there are areas where no data exists, those areas will be filled with 0xFF, and then the area specified by **size** will be signed. Additionally, the output file will be a file filled with 0xFF.

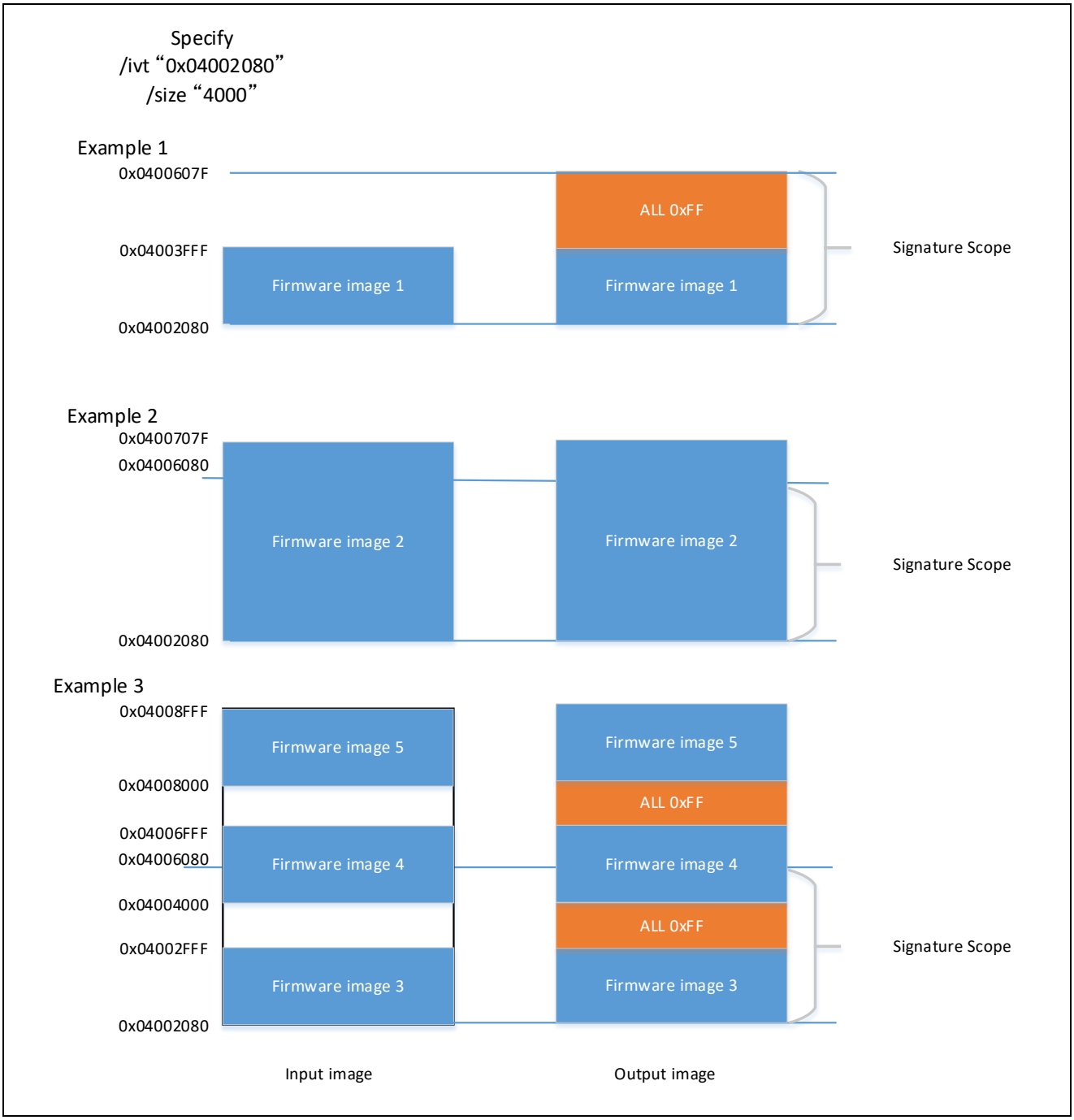


**Figure 4-11 Signature Scope and Output Images When the size Option Is Specified for a Motorola Hex File in the Input Firmware Image**

## 4.11 **calcresponse** Options

Table 4-112 **calcresponse** Options

| Option | Parameter | Description |
|---|---|---|
| **challenge** | Hex data | Specifies the challenge value. (This is usually set to the unique ID of the device.) The data size is 16 bytes. |
| **key** | Hex data | Specifies the DLM key data. The data size is 16 bytes. |
| **algorithm** | Name | Specifies the calculation algorithm. Select either of the following:<br>**HMAC-SHA256**<br>**AES-128-CMAC** |

**Example use of calcresponse options:**

```
> skmt.exe /calcresponse /challenge "ABCDEFGHIJKLMNOPQRSTUVWXYZ012345"
        /key "000102030405060708090A0B0C0D0E0F" /algorithm HMAC-SHA256
```

# 5. Operating Procedure

## 5.1 Standalone

### 5.1.1 Windows

Run the installer SecurityKeyManagementTool_installer_vXXX.exe and install it in any folder.

Note: Install into a folder with as shallow a hierarchy as possible, with write permission, and with no "blanks". If the hierarchy is too deep or the folder name is too long, it may not run.

#### 5.1.1.1 GUI version

SecurityKeyManagementTool.exe in the installation folder is the executable file.

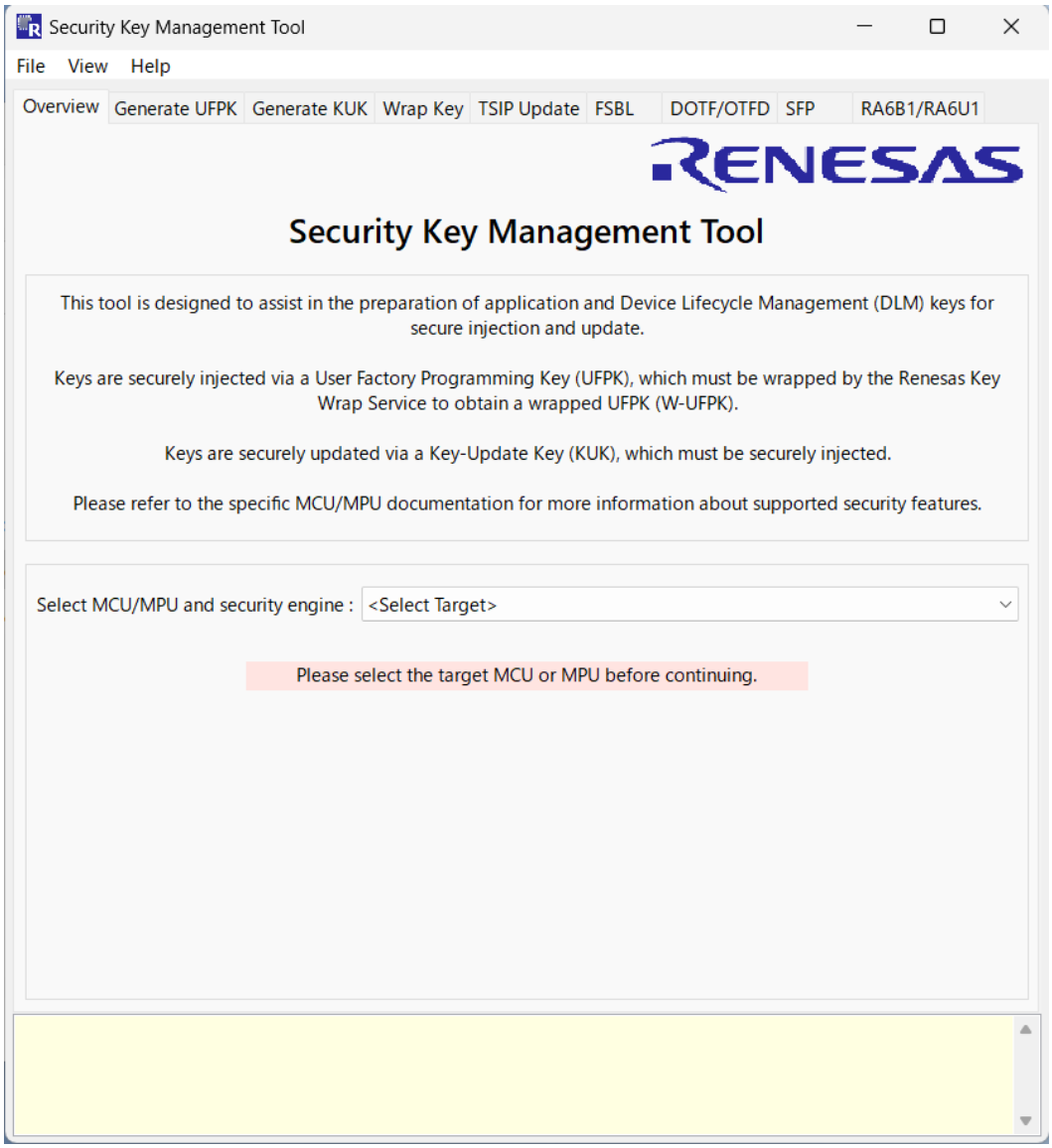


**Figure 5-1 Security Key Management Tool – GUI Dialog at startup from Windows**

#### 5.1.1.2 CLI version

The files stored in the installed **CLI** folder are the complete set of files required to execute the CLI version of the Security Key Management Tool. If only the CLI version of the tool will be used (for example, in a production environment), these files and folder(**Bold**) can be moved or copied to another folder for convenience. The following is the complete list of files needed to execute the CLI version.

- skmt.exe
- libsodium.dll
- **device**
- **address**

Enter the `skmt.exe` command from command prompt.

```
C:\work\skmt\tool>skmt.exe /genkey /ufpk file="C:\work\ufpk.key" /wufpk file="C:\work\u
fpk_enc.key" /mcu "RA-SCE9" /keytype "AES-128" /key "000102030405060708090A0B0C0D0E0F"
/filetype "rfp" /output "c:\work\aes128.rkey"
Output File: c:\work\aes128.rkey
UFPK: EC6B8FA5C0D5DA5142CCAF3A31AEBEAE2346CFE7EF644B9B6B70523CBA0F5C5C
W-UFPK: 000000004F4C172DEF2789DE4F041294C6B1218D11DC81DC5B37FB72DB899DCFF4BE7158
IV: 46EB124312C83FA226994D17A3F5616A
Encrypted key: DEAE066D5845A01E3FBC0445EC1141F60195D3B3F159D2B624A259BE4965A41D


C:\work\skmt\tool>
```

**Figure 5-2 Security Key Management Tool - CLI execution example from command prompt**

### 5.1.2 Linux version

#### 5.1.2.1 GUI version

After unzipping the Linux version package, place the **SecurityKeyManagementTool** folder in any folder. When decompressing, use the command **tar xvzfp xxxxx.tar.gz** to maintain execute permissions.

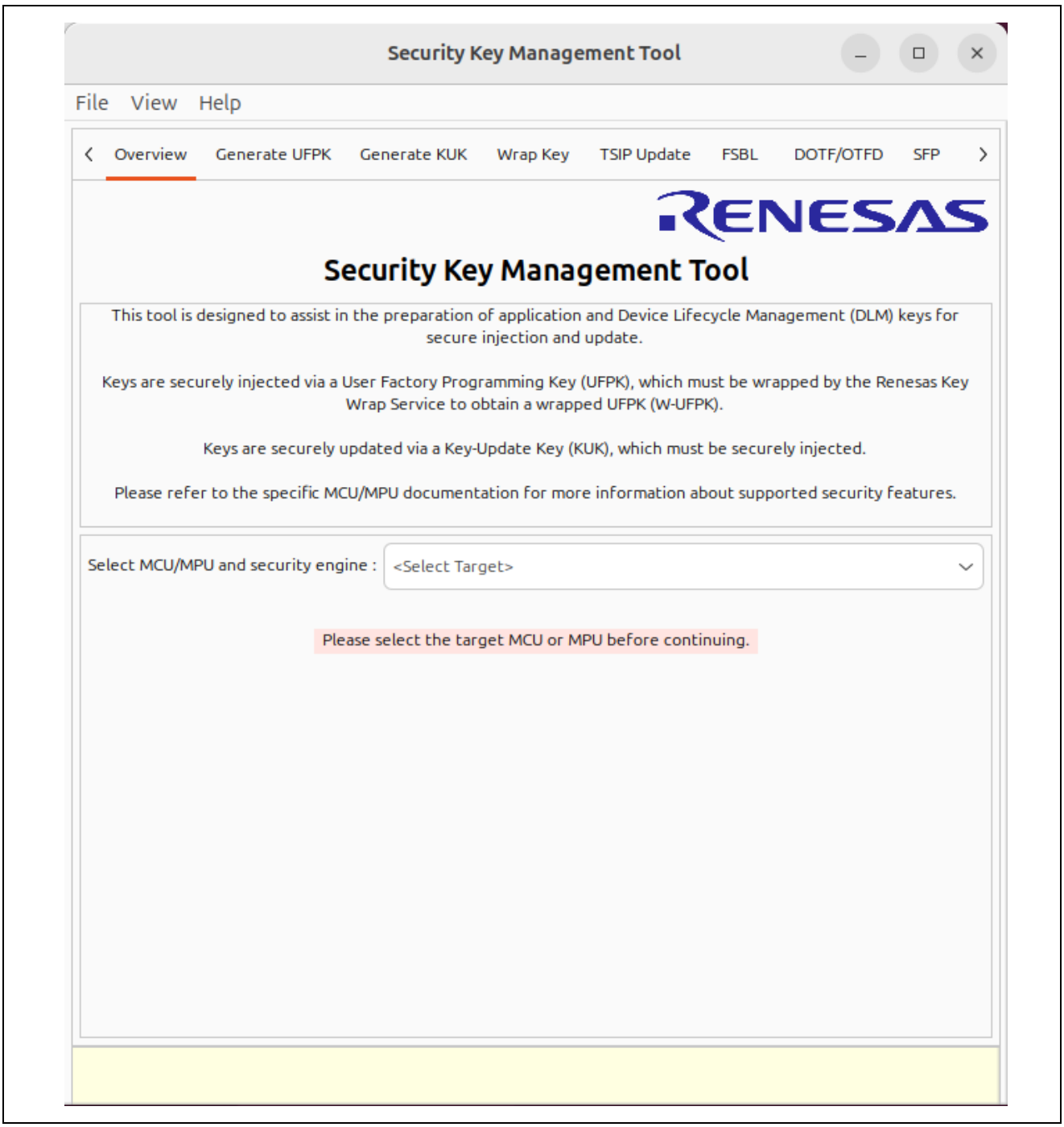


**Figure 5-3 Security Key Management Tool – GUI Dialog at startup from Linux**

### 5.1.2.2 CLI version

The files stored in the installed **CLI** folder are the complete set of files required to execute the CLI version of the Security Key Management Tool. If only the CLI version of the tool will be used (for example, in a production environment), these files can be moved or copied to another folder for convenience. The following is the complete list of files needed to execute the CLI version.

- skmt
- **device**
- **address**

Enter the `skmt` command from the terminal software.

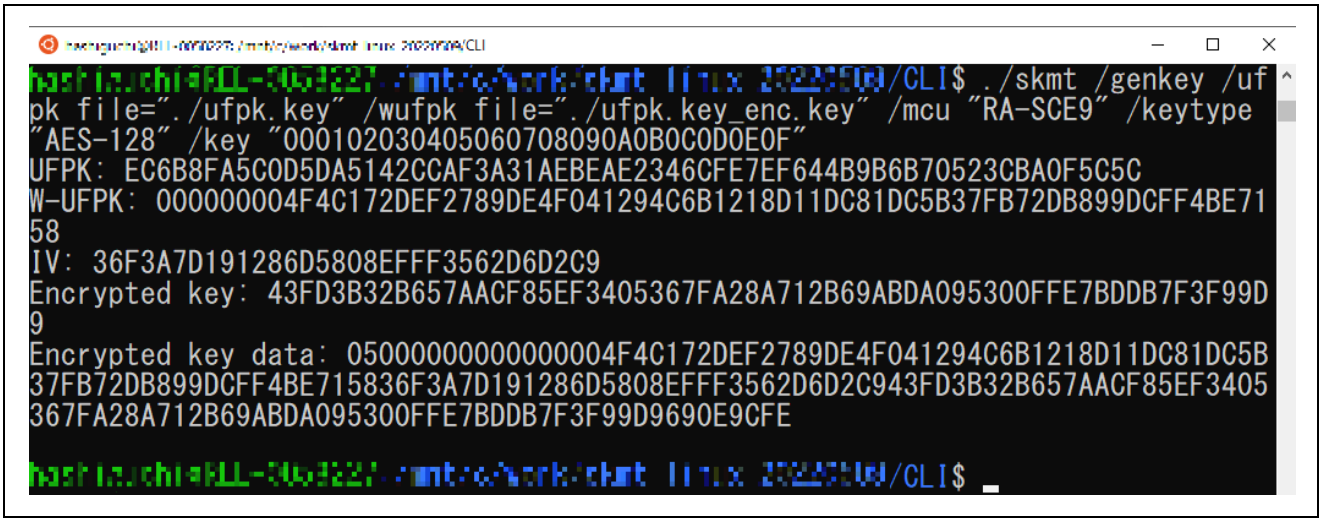


**Figure 5-4 Security Key Management Tool - CLI execution example from Linux Terminal software**

### 5.1.3 macOS version

Unzip the downloaded package in any folder.
After unzipping, the file structure is as follows
SecurityKeyManagementTool.app.zip : GUI version
SKMT-CLI.zip : CLI version

#### 5.1.3.1 GUI version

Unzip SecurityKeyManagementTool.app.zip in any folder, where SecurityKeyManagementTool.app is the executable file.

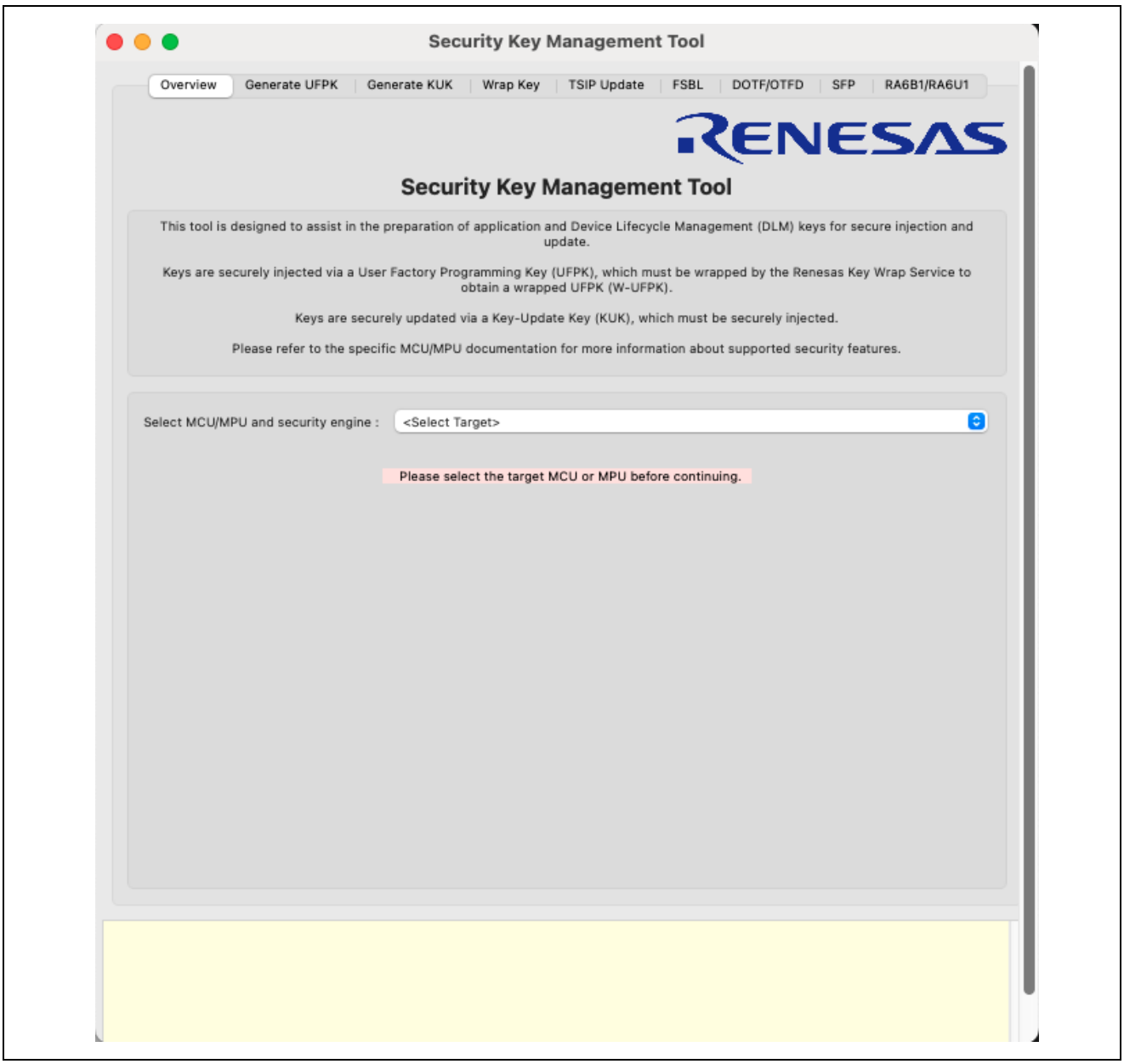


**Figure 5-5 Security Key Management Tool – GUI Dialog at startup from macOS**

### 5.1.3.2 CLI version

Unzip the file SKMT_CLI.zip in the extracted folder to a folder of your choice. Place the following set of files in the folder in the folder of your choice.

- skmt
- **device**
- **address**

Enter the `skmt` command from the terminal software.

**Figure 5-6 Security Key Management Tool - CLI execution example from macOS Terminal software**

## 5.2 e²studio plugin

The Windows, Linux and macOS versions follow the same installation and uninstallation procedure. Please follow the steps below to install and uninstall.

### 5.2.1 Installing e²studio plugin version

1. Download SecurityKeyManagementTool_Plugin_vXXX_Windows.zip SecurityKeyManagementTool_Plugin_vXXX_mac.zip or SecurityKeyManagementTool_Plugin_vXXX_Linux.zip from the Renesas Web site and place it in any folder of your choice. vXXX is this tool version.
2. Start e²studio
3. Select e²studio menu "**Help(H)**" – "**Install New Software...**" menu to open the "**Install**" dialog.

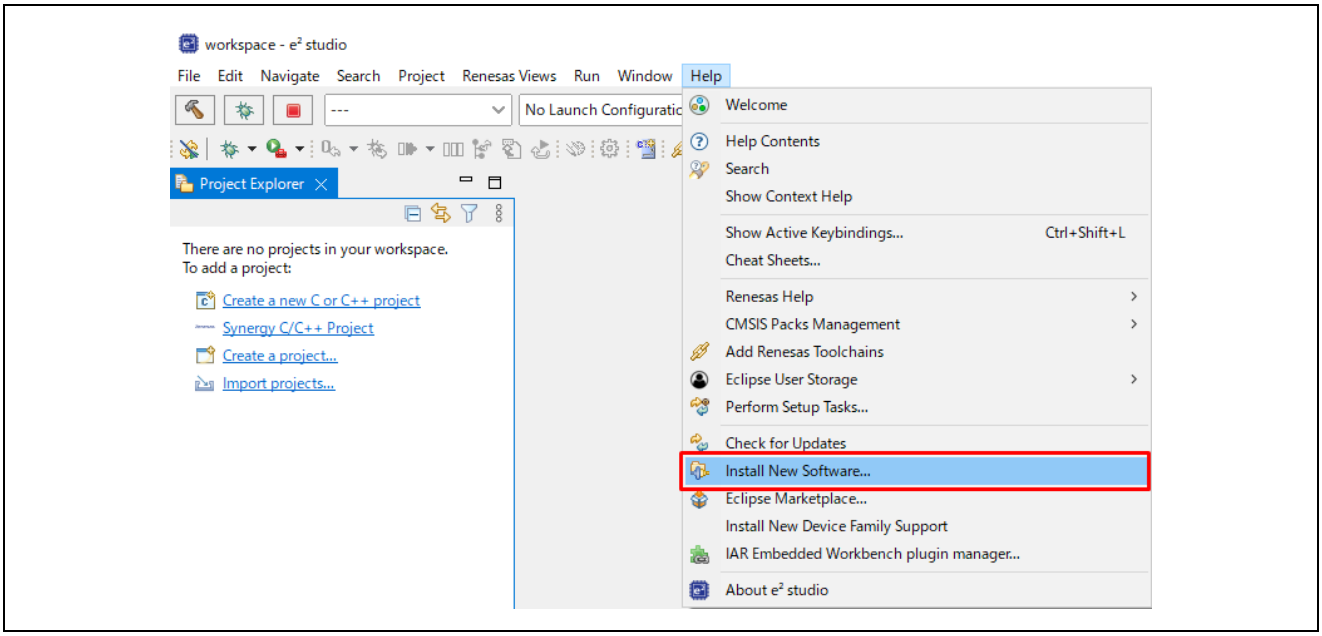


**Figure 5-7 e²studio "Help(H)" – "Install New Software..."**

4. Press the “**Add**” button in the “**Install**” dialog to open the “**Add Repository**” dialog.

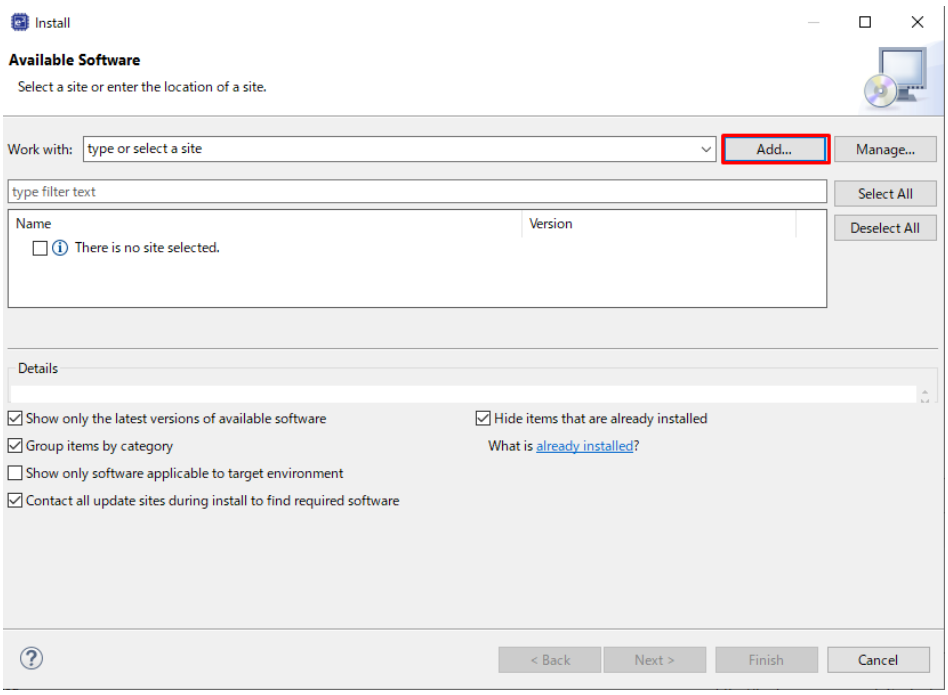


**Figure 5-8 “Install” Dialog**

5. Click the "**Archive(A)…**" button on the "**Add Repository**" dialog, specify the SecurityKeyManagementTool_Plugin_vXXX_Window/Linux/mac.zip file prepared in step 1, and then click the "**Add**" button.

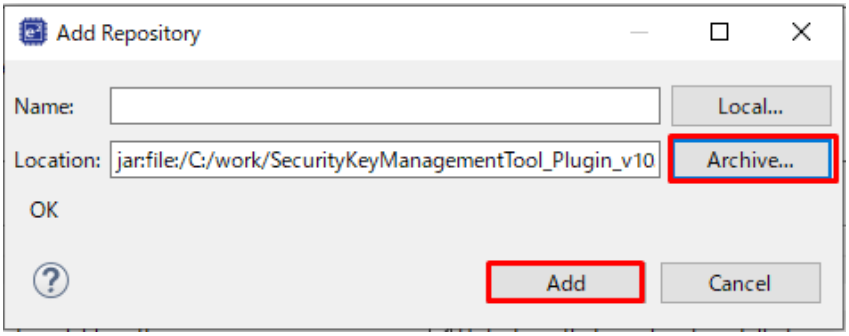


**Figure 5-9 “Add Repository” Dialog**

6. "**Renesas Solution Toolkit**" will be added to the "**Install**" dialog box. Check the checkbox and press the "**Next >**" button. Uncheck box "Contact all sites during install to find required software.". If left checked, installation will take longer.

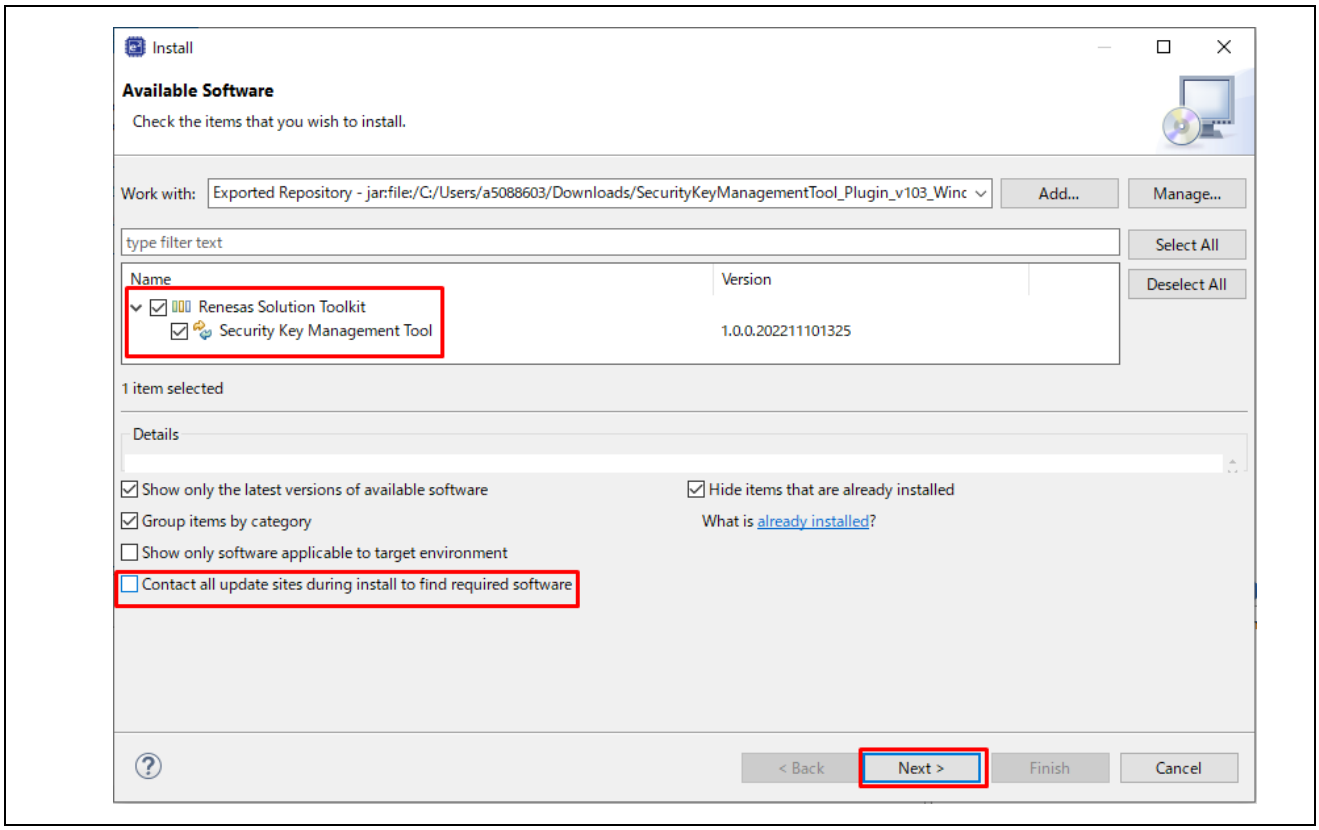


**Figure 5-10 "Install" Dialog – Select "Security Key Management Tool"**

7. When the "**Install**" dialog box appears, confirm that the "**Security Key Management Tool**" is installed and press the "**Next >**" button.

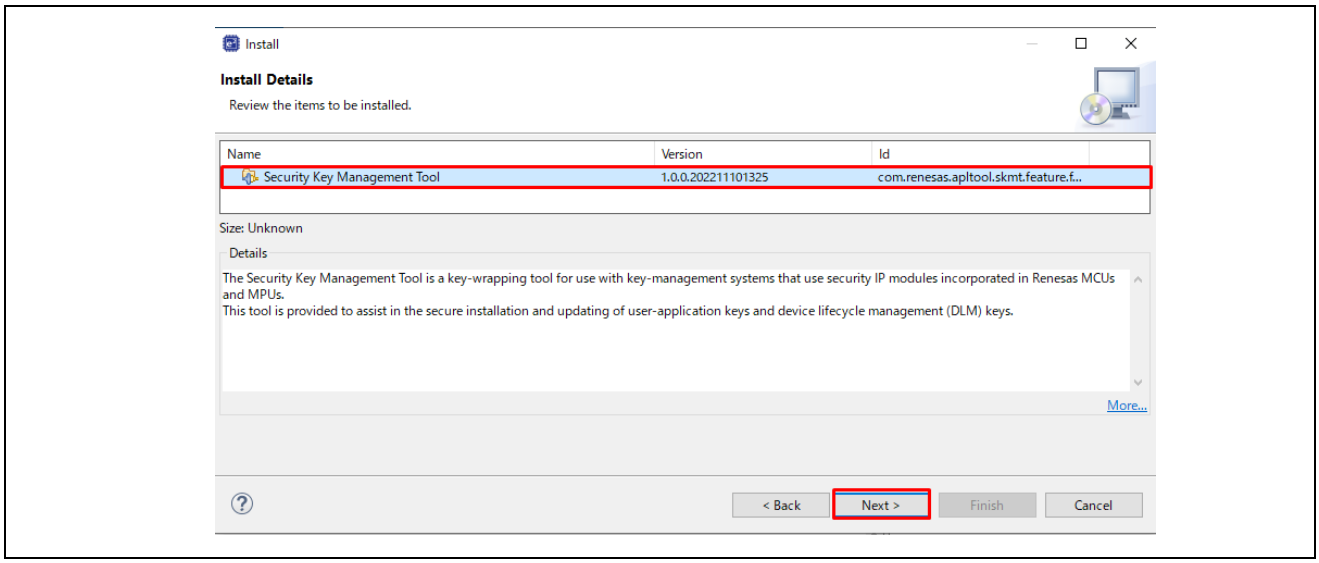


**Figure 5-11 "Install" Dialog – Install Details**

8. This is followed by the license confirmation screen. Accept the terms of the license, which was presented during the Plugin download and can be reviewed at the URL displayed in the dialog box. Select "I accept the terms of the license agreement" and press the Finish button.

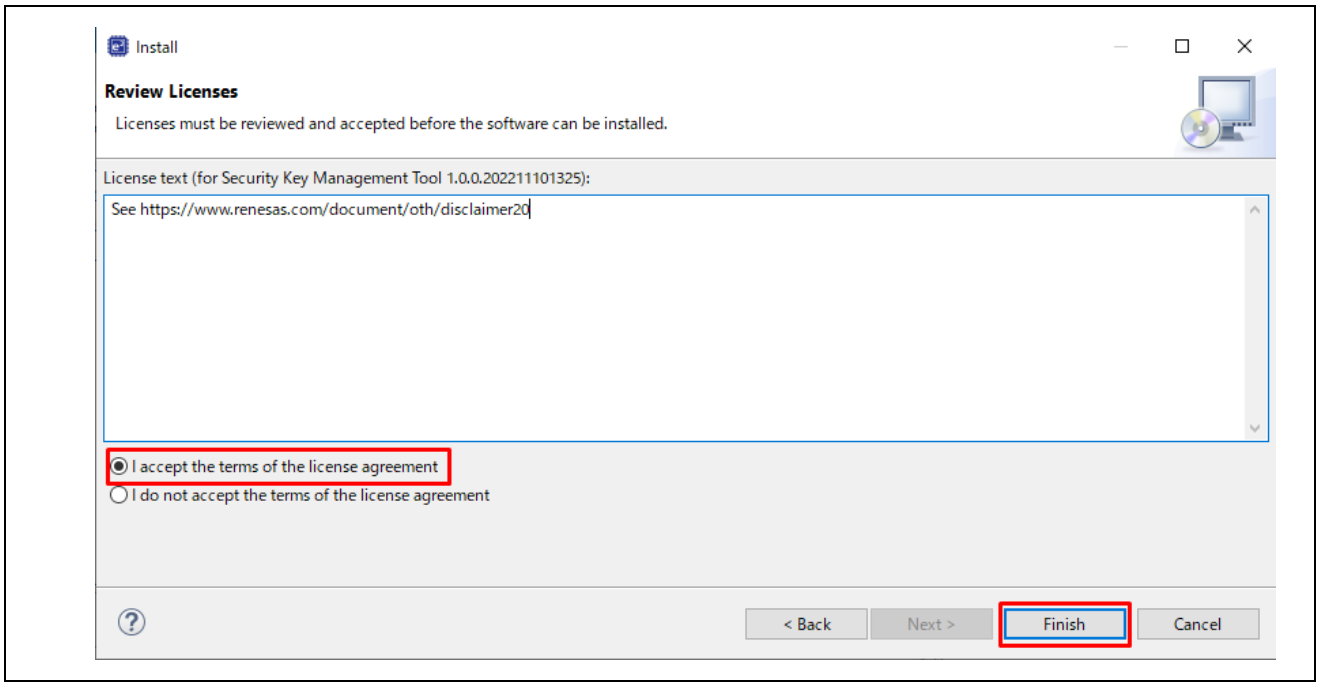


**Figure 5-12 “Install” Dialog – Review License**

9. When the "**Trust**" dialog box appears, check the displayed certificate and press the "**Trust Selected**" button to continue the installation.

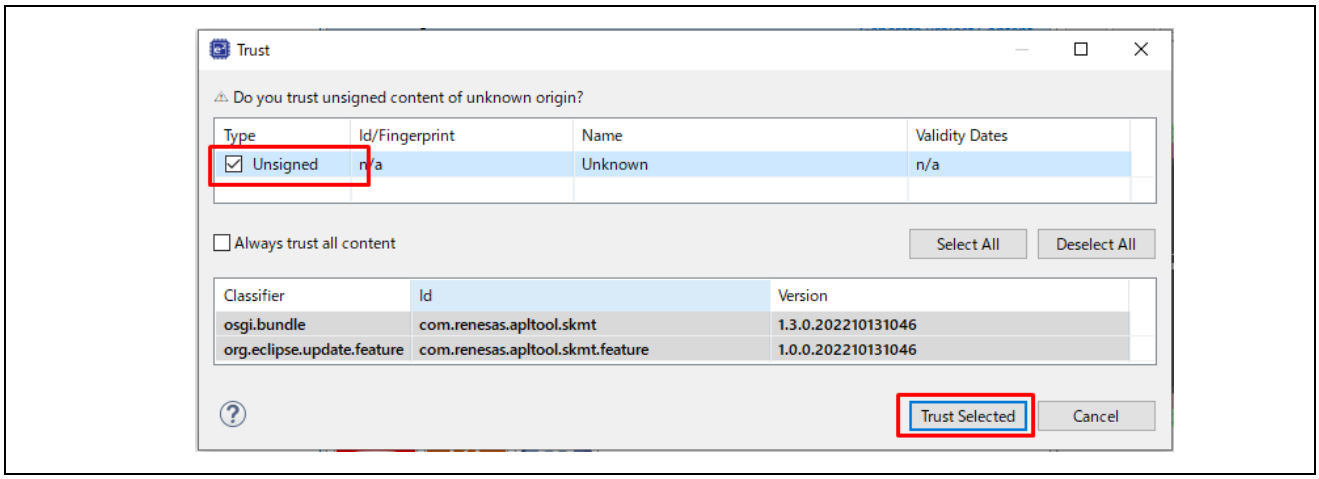


**Figure 5-13 “Trust” Dialog**

10. Restart e² studio when prompted.

11. After installation is complete, the Security Key Management Tool will be added to the "**Properties**" dialog of the project.

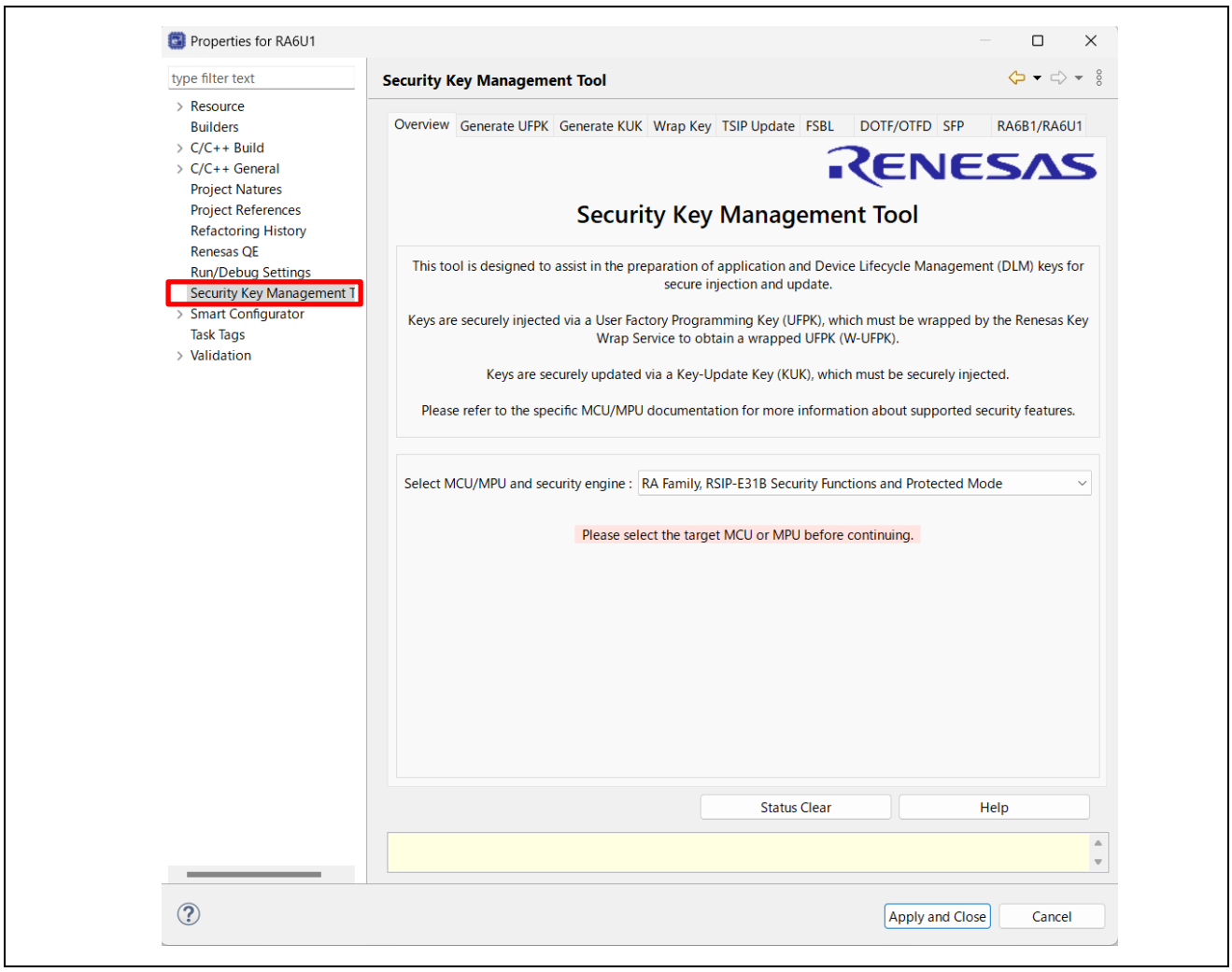


**Figure 5-14 Project "Properties" Dialog**

### 5.2.2 Uninstalling e²studio plugin version

1. Select e² studio menu “**Help(H)**” – “**About e² studio**” menu to open the “**About e² studio**” dialog.
2. Press the "**Installation Details**" button in the "**About e² studio**" dialog.

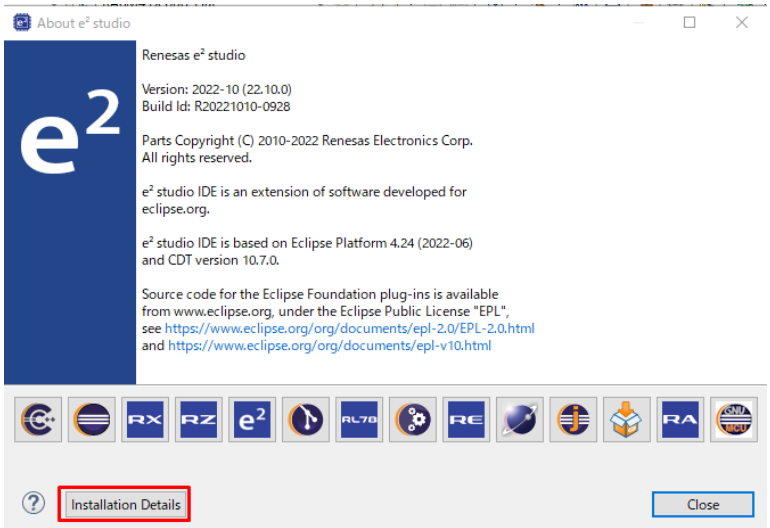


**Figure 5-15 “About e²studio” Dialog**

3. In the "**e² studio Installation Details**" dialog, select the "**Installed Software**" tab - Security Key Management Tool and press the "**Uninstall...**" button.

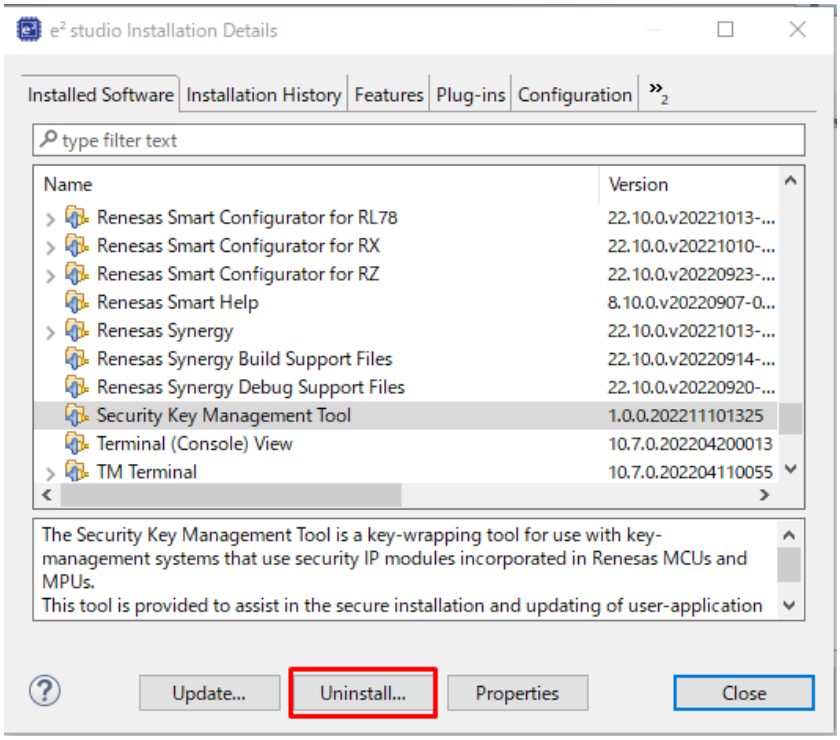


**Figure 5-16 "e² studio Installation Details" Dialog**

4. When the "**Uninstall**" dialog box appears, select the Security Key Management Tool and press the "**Finish**" button.

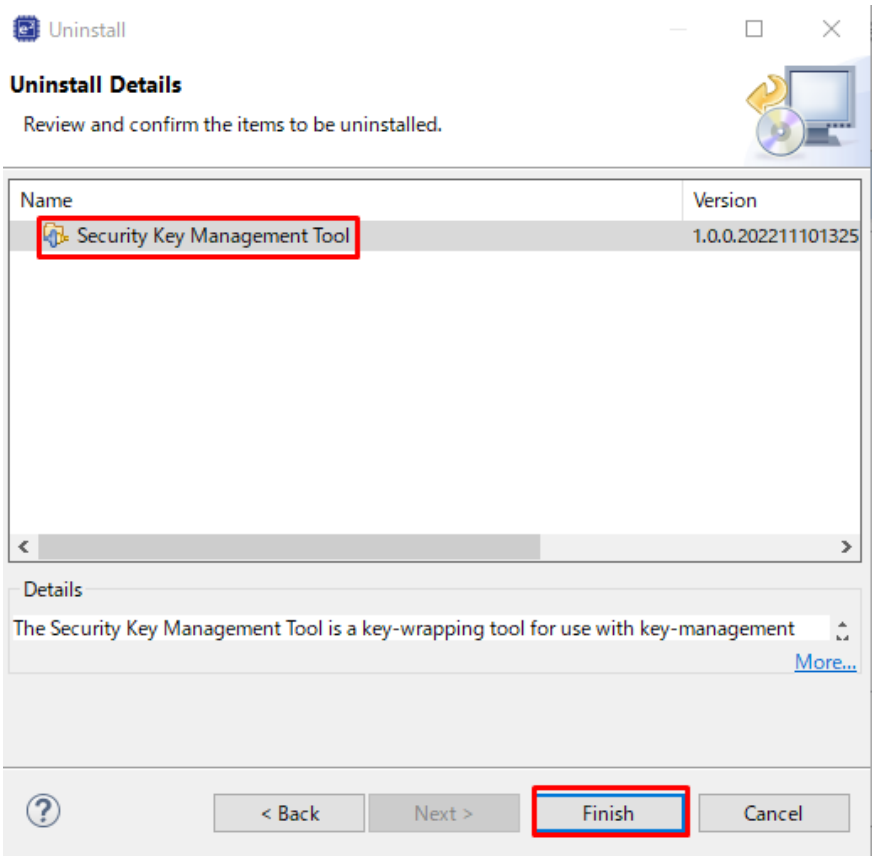


**Figure 5-17 "Uninstall" Dialog**

5. You will be prompted to restart e$^2$ studio.

# 6. Usage Examples

## 6.1 Example 1 – Inject an AES128 Key on an RX Family MCU with TSIP

The RX Family TSIP supports secure key injection via firmware executing on the MCU. The required drivers can be found in the TSIP library, as per *Table 1-1 MCU/MPU Related Information*.

During production, it is often necessary to program groups of devices with identical keys. The key injection information can be embedded in provisioning code, but if there are multiple groups with different keys, it may be desirable to separate the key information from the provisioning code. These steps can be used to create a Motorola Hex file that can be programmed in conjunction with provisioning code to securely inject one or more keys. This example creates a file to inject one AES128 key.

### 6.1.1 Using the GUI version

1. Select the MCU/MPU and security engine in the [**Overview**] tab.

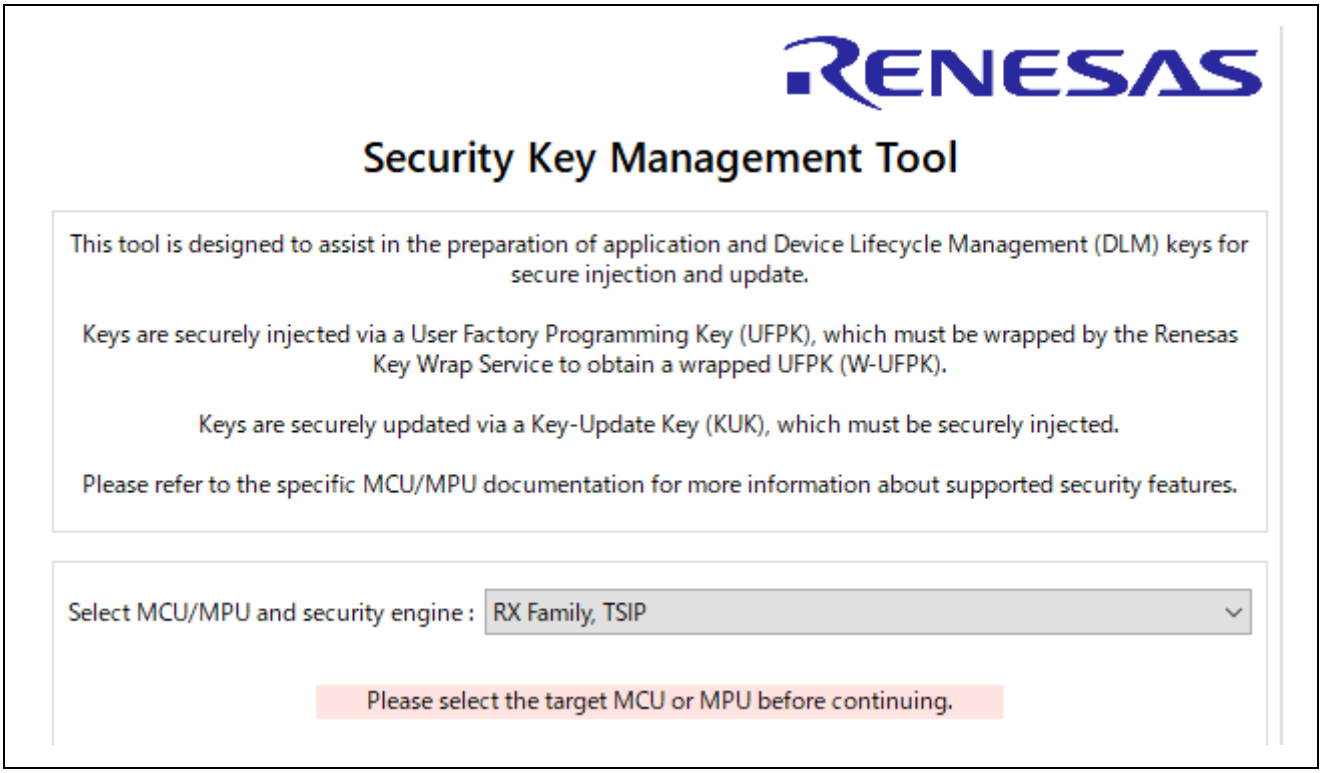


**Figure 6-1 [Overview] Tab, Injecting an AES128 Key with RX Family TSIP**

2. Create a UFPK. In the [**Generate UFPK**] tab, enter the desired UFPK value and enter the output file name with a `*.key` extension. This example uses the file name `ufpk.key`.

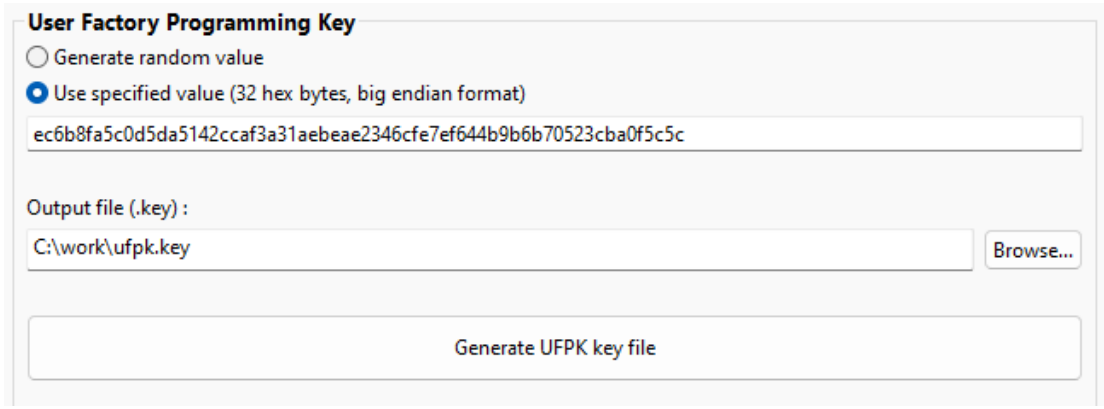


**Figure 6-2 [Generate UFPK] Tab, Example of UFPK generation using specified value**

Press the "**Generate UFPK key file**" button to generate the UFPK file. If the file is generated successfully, the tool will output the following execution result.

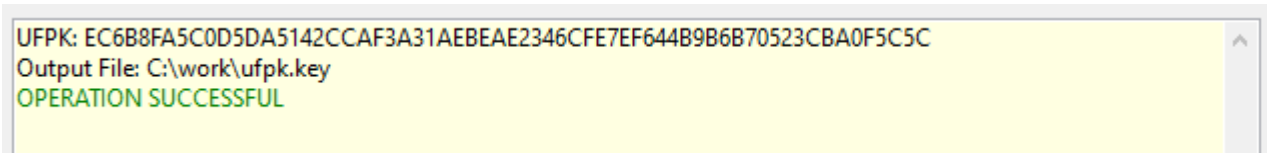


**Figure 6-3 Example of execution result, UFPK generation using specified value**

3. Get the W-UFPK. Send the `ufpk.key` file generated in step 2 to the Renesas Key Wrap service to get the W-UFPK. For more information, refer to the Renesas Key Wrap Service FAQ or the device-specific application note. The URL for the Renesas Key Wrap service is https://dlm.renesas.com/keywrap.

4. Create the AES128 key file as a Motorola Hex file. In the [**Wrap Key**] tab, select **AES128** in the [**Key Type**] tab and enter the AES128 data in the [**Key Data**] tab. For the **Wrapping Key**, select **UFPK** and select the UFPK file generated in step 2 and the W-UFPK file obtained in step 3. For simplicity, this example selects **Generate random value** for the **IV**. In the **Output** panel, select **Motorola Hex** as the **Format**, enter a file name with a `*.mot` extension, and set the **Address** field to FFFF0000. Specify **Little** as the **Endian**.

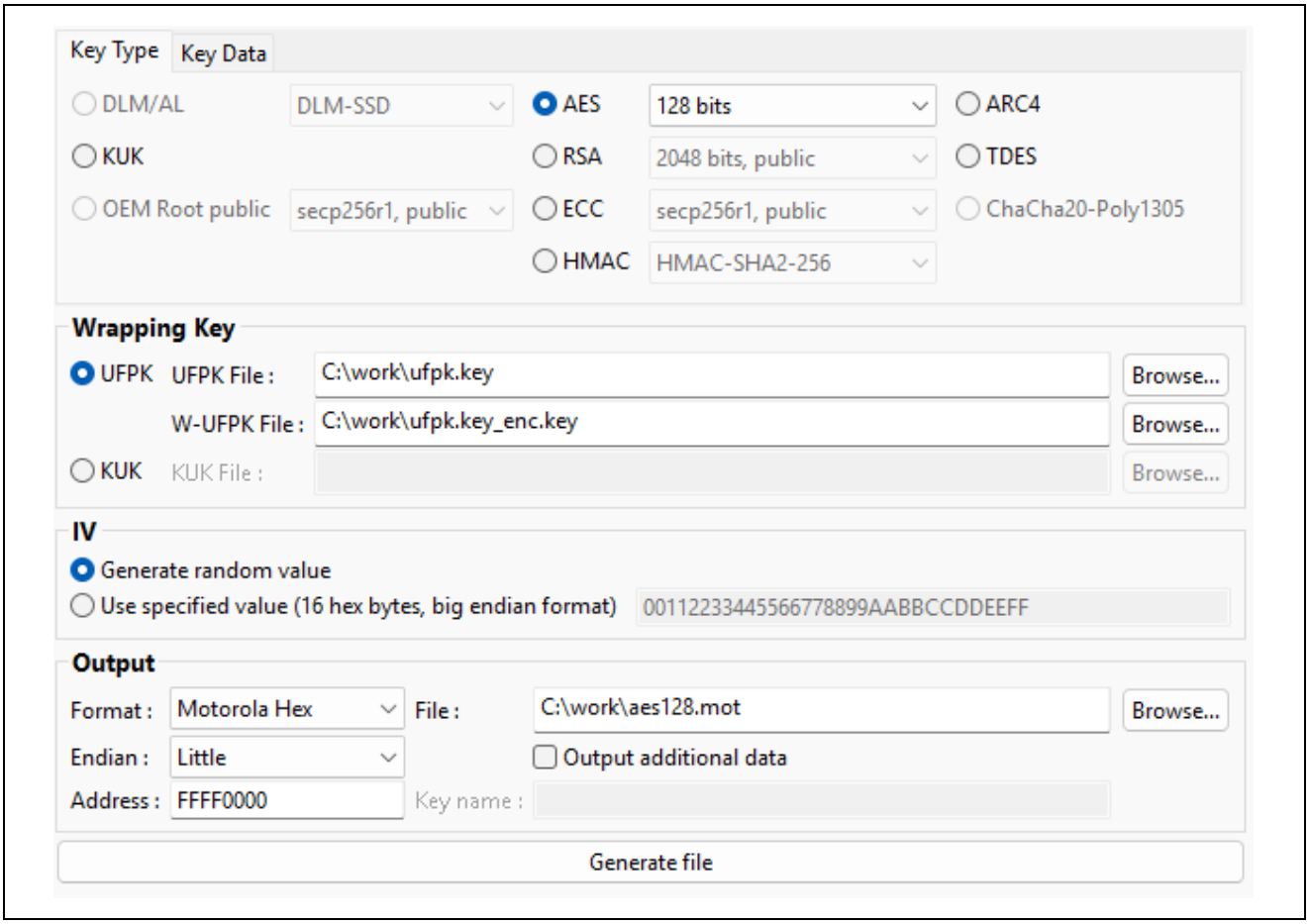


**Figure 6-4 [Wrap Key] - [Key Type] Tab, Example of creating a AES128 key file as Motorola Hex**

**Figure 6-5 [Wrap Key] - [Key Data] Tab, Example of creating an AES128 key file as Motorola Hex**

Click the **Generate file** button to generate the specified output file. If the file is generated successfully, the tool will output the following execution result.

```
Output File: C:\work\aes128.mot
UFPK: EC6B8FA5C0D5DA5142CCAF3A31AEBEAE2346CFE7EF644B9B6B70523CBA0F5C5C
W-UFPK: 00000001102D464621E307E4FF7027D346B9A2DEA8E35A6673DC9685DE0283CBED82DE1E
IV: DBDD88AC0378B58AF7471FEBE17AF392
Encrypted key: 23D335F4BA2BFF6D581F0412C73E8599429BB2AEDBFBBBF18A4E374C39507AEA
OPERATION SUCCESSFUL
```

**Figure 6-6 Example of execution result, creating an AES128 key file as Motorola Hex**

### 6.1.2 Using the CLI version

1. Create the UFPK using the **genufpk** command. This example shows a known value being used for the UFPK:

**> skmt.exe /genufpk**
**/ufpk "ec6b8fa5c0d5da5142ccaf3a31aebeae2346cfe7ef644b9b6b70523cba0f5c5c"**
**/output "C:\work\ufpk.key"**

```
C:\work\skmt\tool>skmt.exe /genufpk /ufpk "ec6b8fa5c0d5da5142ccaf3a31aebeae2346cfe7ef64
4b9b6b70523cba0f5c5c" /output "C:\work\ufpk.key"
UFPK: EC6B8FA5C0D5DA5142CCAF3A31AEBEAE2346CFE7EF644B9B6B70523CBA0F5C5C
Output File: C:\work\ufpk.key
```

**Figure 6-7 Execution result of CLI genufpk command**

2. Get the W-UFPK. Send the `ufpk.key` file generated in step 1 to the Renesas Key Wrap service to get the W-UFPK. For more information, refer to the Renesas Key Wrap Service FAQ or the device-specific application note. The URL for the Renesas Key Wrap service is https://dlm.renesas.com/keywrap.

3. Create the AES128 key file as Motorola Hex by using the **genkey** command using the UFPK file generated in step 1 and the W-UFPK file obtained in step 2:

**> skmt.exe /genkey /ufpk file="c:\work\ufpk.key" /wufpk file="C:\work\ufpk.key_enc.key"**
**/mcu "RX-TSIP" /keytype "AES-128" /key "000102030405060708090A0B0C0D0E0F"**
**/filetype "mot" /address "FFFF0000" /output "C:\work\aes128.mot"**

```
C:\work\skmt\tool>skmt.exe /genkey /ufpk file="c:\work\ufpk.key" /wufpk file="C:\work\u
fpk.key_enc.key" /mcu "RX-TSIP" /keytype "AES-128" /key "000102030405060708090A0B0C0D0E
0F" /filetype "mot" /address "FFFF0000" /output "C:\work\aes128.mot"
Output File: C:\work\aes128.mot
UFPK: EC6B8FA5C0D5DA5142CCAF3A31AEBEAE2346CFE7EF644B9B6B70523CBA0F5C5C
W-UFPK: 000000004F4C172DEF2789DE4F041294C6B1218D11DC81DC5B37FB72DB899DCFF4BE7158
IV: 9232A2FD6FC74A7DC0A501397A1A34F3
Encrypted key: 7E66CEB3CA203748A669EDB6B17EF8D231244EA70EF72B839F54CBE41CC1CCC6
```

**Figure 6-8 CLI example of creating an AES128 key file as Motorola Hex**

## 6.2 Example 2 – Inject a Key-Update Key on an RA Family MCU with SCE9 Security Functions and Protected Mode

The RA Family MCUs with SCE9 in Protected Mode support secure key injection via the programming interface. The Renesas Flash Programmer (RFP) supports this capability.

The advantage of using the programming interface to securely inject keys is that no special provisioning code is needed on the MCU. The keys and application code can be installed as part of the same programming process. This example injects one KUK, enabling key update in the field.

### 6.2.1 Using the GUI version

1. Select the MCU/MPU and security engine in the [**Overview**] tab.

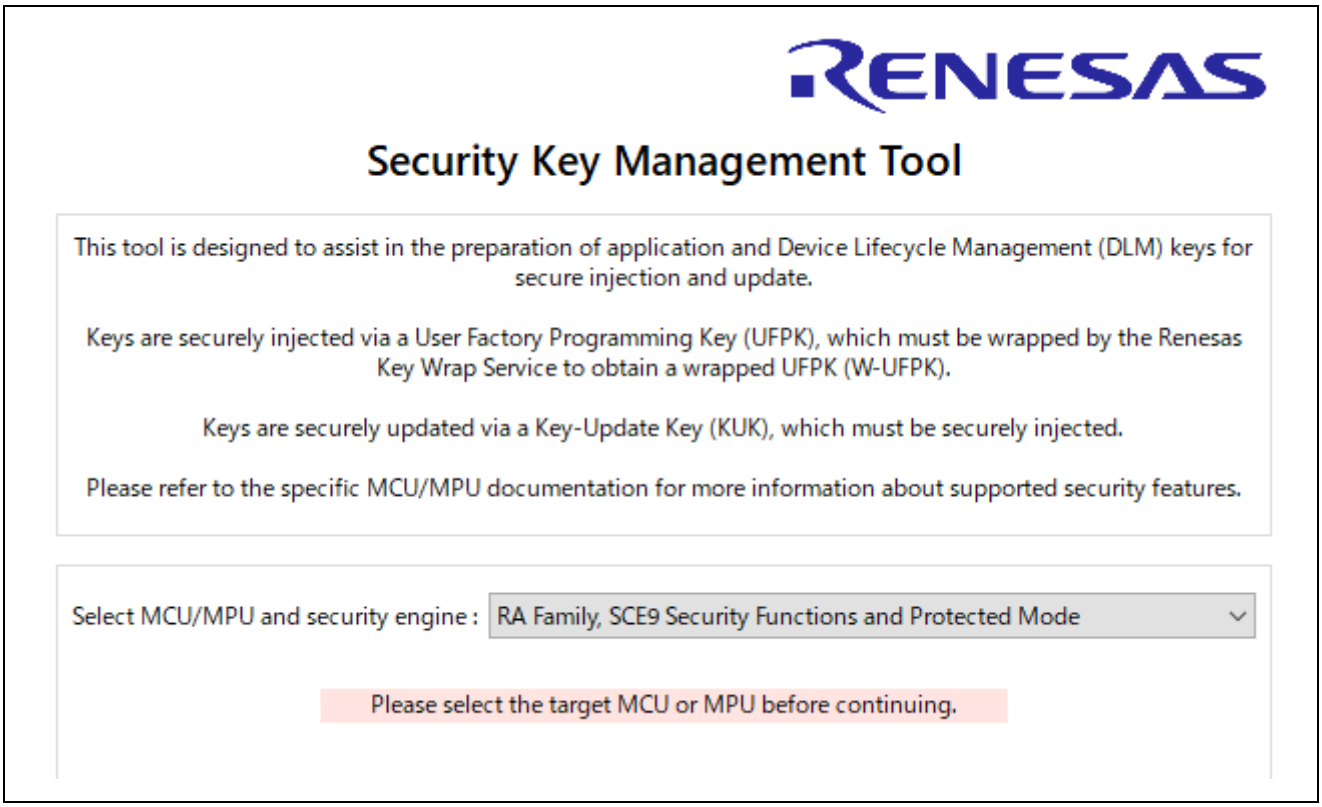


**Figure 6-9 [Overview] Tab, Injecting a KUK with RA Family SCE9 Security Functions and Protected Mode**

2. Create a UFPK. In the [**Generate UFPK**] tab, enter the desired UFPK value and enter the output file name with a `*.key` extension. This example uses the file name `ufpk.key`.

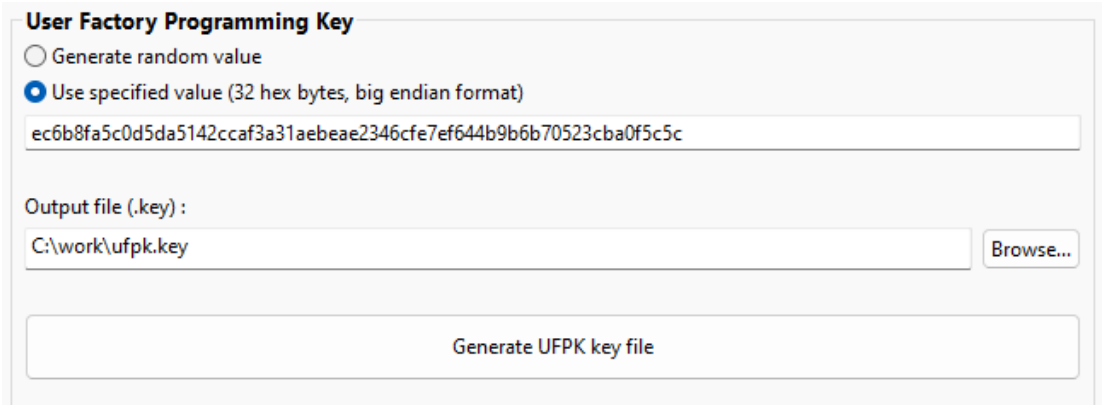


**Figure 6-10 [Generate UFPK] tab Example of UFPK generation using specified value**

Press the **Generate UFPK key file** button to generate the UFPK file. If the file is generated successfully, the tool will output the following execution result.:

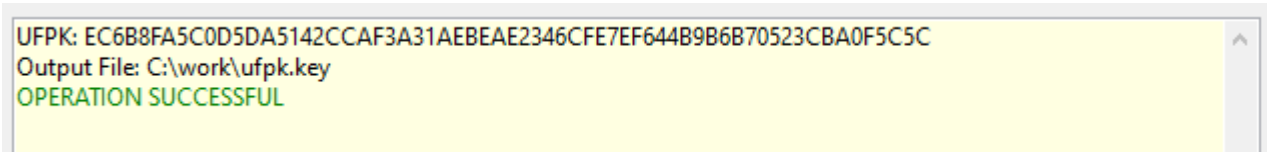


**Figure 6-11 Example execution result, UFPK generation using specified value**

3. Get the W-UFPK. Send the `ufpk.key` file generated in step 2 to the Renesas Key Wrap service to get the W-UFPK. For more information, refer to the Renesas Key Wrap Service FAQ or the device-specific application note. The URL for the Renesas Key Wrap service is https://dlm.renesas.com/keywrap.

4. Create the KUK key file. In the [**Generate KUK**] tab, select to either use the tool to generate a random value for the KUK or to enter a 256-bit value for the KUK. Enter the output file name with a `*.key` extension. This example uses the file name `kuk.key`.

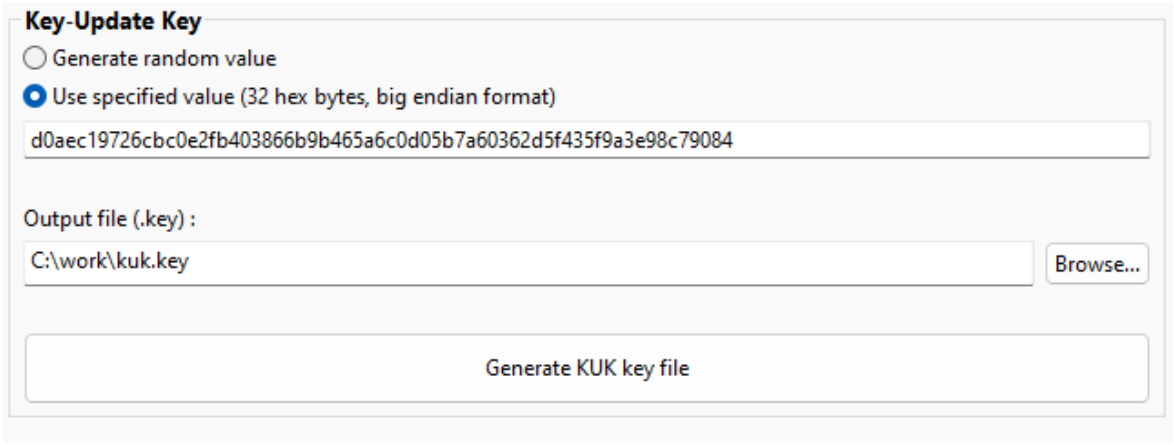


**Figure 6-12 [Generate KUK] Tab, Example of creating a KUK key file**

Click the **Generate KUK key file** button to generate the KUK key file. If the file is generated successfully, the tool will output the following execution result.

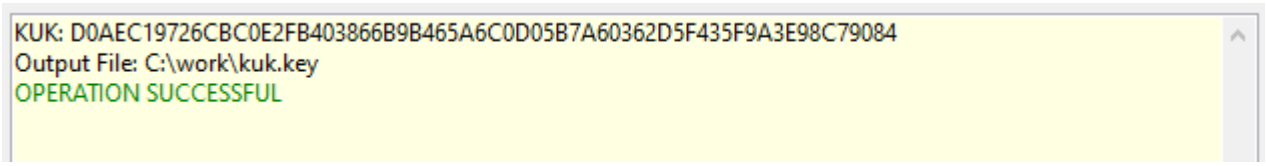


**Figure 6-13 Example execution result, creating a KUK file**

5. Create an RFP file for KUK injection. In the [**Wrap Key**] tab, select **KUK** in the [**Key Type**] tab. In the [**Key Data**] tab, enter the KUK key file name that was generated in the previous step (`kuk.key` in this example). For the **Wrapping key**, select **UFPK** and select the UFPK file generated in step 2 and the W-UFPK file obtained in step 3. For simplicity, this example selects **Generate random value** for the **IV**. In the **Output** panel, select **RFP** as the **Format** and enter a file name with a `*.rkey` extension.

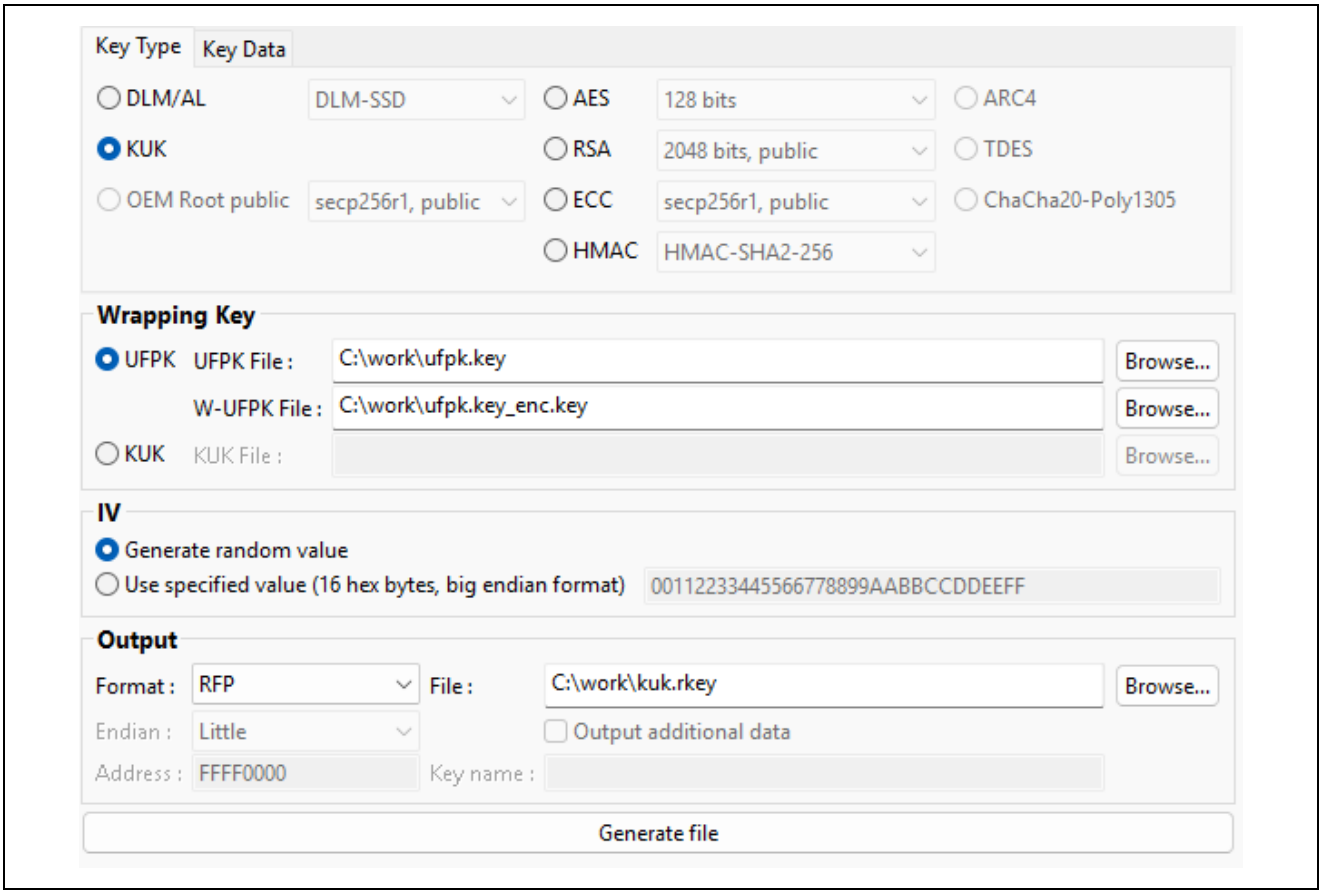


**Figure 6-14 [Wrap Key] – [Key Type] Tab, Example of creating a KUK file as an RFP file**

**Figure 6-15 [Wrap Key] – [Key Data] Tab, Example of creating a KUK file as an RFP file**

Click the **Generate file** button to generate the specified output file. If the file is generated successfully, the tool will output the following execution result.

```
W-UFPK: 00000000971BF948953C5A92626AC7D70CFD8C4803B7FC978A9028CF1CD0CC32BD2F7F97
IV: A7136C9E53E737BF2B5864AEEDA43D58
Encrypted key:
9B00211834DB5492DB9CE4C8707D1C38D5032B5538CBFDEC8A0D3B65FD802F81475D14B70DE4C89DE8A8ABCF28FA
899F
OPERATION SUCCESSFUL
```

**Figure 6-16 Example execution result, creating a KUK file as an RFP file**

### 6.2.2 Using the CLI version

1. Create the UFPK by using the **genufpk** command. This example shows a specified value being used for the UFPK:

**> skmt.exe /genufpk**
**/ufpk “ec6b8fa5c0d5da5142ccaf3a31aebeae2346cfe7ef644b9b6b70523cba0f5c5c”**
**/output “C:\work\ufpk.key”**

```
C:\work\skmt\tool>skmt.exe /genufpk /ufpk "ec6b8fa5c0d5da5142ccaf3a31aebeae2346cfe7ef64
4b9b6b70523cba0f5c5c" /output "C:\work\ufpk.key"
UFPK: EC6B8FA5C0D5DA5142CCAF3A31AEBEAE2346CFE7EF644B9B6B70523CBA0F5C5C
Output File: C:\work\ufpk.key
```

**Figure 6-17 Execution result of CLI genufpk command**

2. Get the W-UFPK. Send the `ufpk.key` file generated in step 1 to the Renesas Key Wrap service to get the W-UFPK. For more information, refer to the Renesas Key Wrap Service FAQ or the device-specific application note. The URL for the Renesas Key Wrap service is https://dlm.renesas.com/keywrap.

3. Create the KUK key file by using the **genkuk** command. This example shows a specified value being used for the KUK.

**> skmt.exe /genkuk /output “C:\work\kuk.key”**
**/kuk “d0aec19726cbc0e2fb403866b9b465a6c0d05b7a60362d5f435f9a3e98c79084”**

```
C:\work\skmt\tool>skmt.exe /genkuk /output "C:\work\kuk.key" /kuk "d0aec19726cbc0e2fb40
3866b9b465a6c0d05b7a60362d5f435f9a3e98c79084"
KUK: D0AEC19726CBC0E2FB403866B9B465A6C0D05B7A60362D5F435F9A3E98C79084
Output File: C:\work\kuk.key
```

**Figure 6-18 Execution result of CLI genkuk command**

4. Create the RFP file to use for KUK injection by using the **genkey** command, using the UFPK file generated in step 1, the W-UFPK file obtained in step 2, and the KUK file generated in step 3:

**> skmt.exe /genkey /ufpk file=”c:\work\ufpk.key” /wufpk file=”C:\work\ufpk.key_enc.key” /mcu “RA-SCE9” /keytype “key-update-key” /key file=”C:\work\kuk.key” /filetype “rfp” /output “C:\work\kuk.rkey”**

```
C:\work\skmt\tool>skmt.exe /genkey /ufpk file="c:\work\ufpk.key" /wufpk file="C:\work\u
fpk.key_enc.key" /mcu "RA-SCE9" /keytype "key-update-key" /key file="C:\work\kuk.key" /
filetype "rfp" /output "C:\work\kuk.rkey"
Output File: C:\work\kuk.rkey
UFPK: EC6B8FA5C0D5DA5142CCAF3A31AEBEAE2346CFE7EF644B9B6B70523CBA0F5C5C
W-UFPK: 000000004F4C172DEF2789DE4F041294C6B1218D11DC81DC5B37FB72DB899DCFF4BE7158
IV: 4B44A010D7BDF764200BC2424650F976
Encrypted key: 043565AE00E185EBAFF505290DF1976B1A00132FE1F0B25F32C3C83BF92C83A7644968FF
5869821C3EBAD7EB41AA509B
```

**Figure 6-19 CLI example of creating an AES128 key file as an RFP file**

## 6.3 Example 3 – Update an RSA 2048 Public Key on an RA Family MCU with SCE9 Protected Mode

Keys can be updated in the field using a previously injected KUK. The required drivers can be found in the device's supporting software drivers, as per *Table 1-1 MCU/MPU Related Information*.

Key update in the field can be performed in a variety of ways. One way is to include the KUK-wrapped key as part of a firmware update. A simple way to do this is to embed the data as a C source file. This example updates one RSA 2048 public key using a previously injected KUK, such as the one created and injected in the previous example.

### 6.3.1 Using the GUI version

1. Select the MCU/MPU and security engine in the [**Overview**] tab.

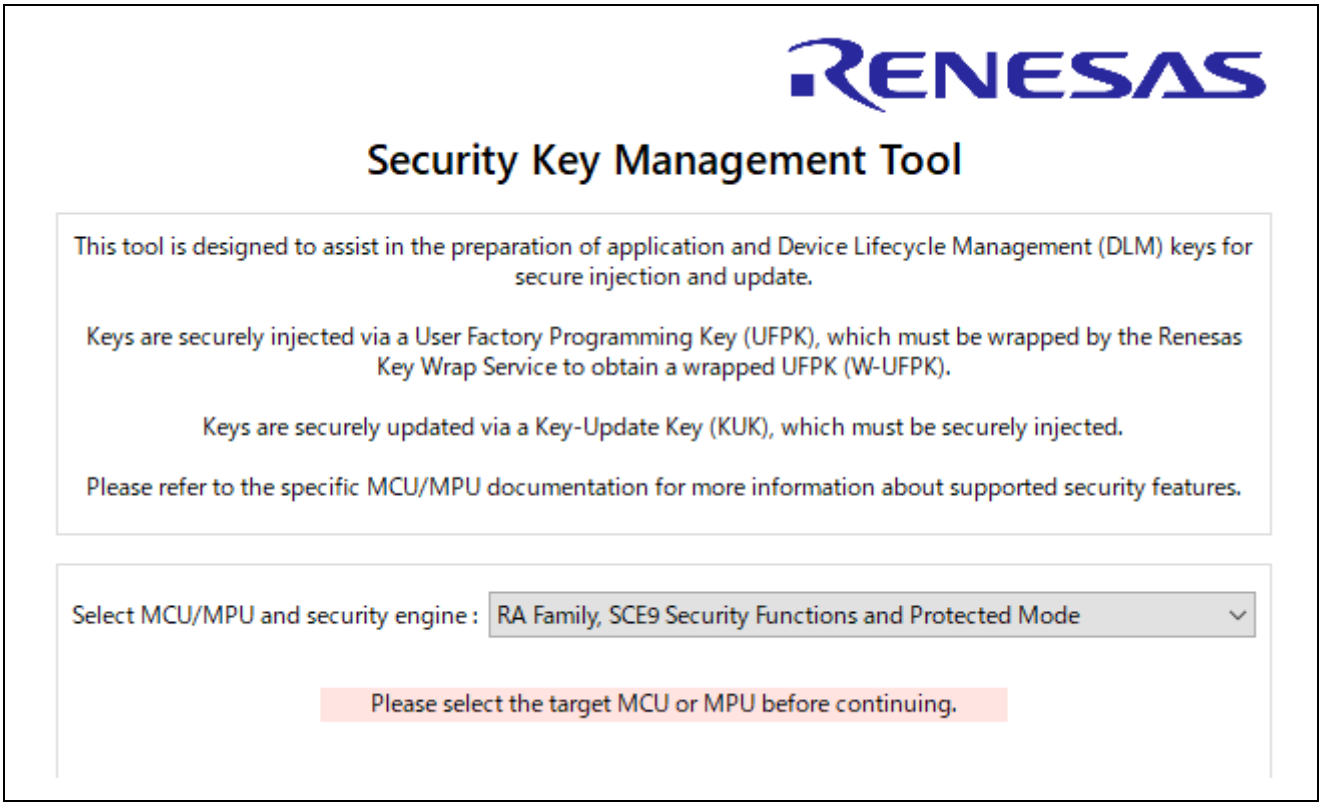


**Figure 6-20 [Overview] Tab, Updating an RSA 2048 Public Key, RA Family SCE9 Security Functions and Protected Mode**

2. Create the RSA 2048 Public key file as a C source file. In the [**Wrap Key]** tab, select **RSA** and **2048 bits, public** in the [**Key Type**] tab and enter the RSA 2048 public key data in the [**Key Data**] tab. For the **Wrapping key**, select **KUK** set **KUK file** to the file that was generated in section 6.2 Example 2 – Inject a Key-Update Key on an RA Family MCU with SCE9 Protected Mode. For simplicity, this example selects **Generate random value** for the **IV**. In the **Output** panel, select **C Source** as the **Format** and enter a file name with a `*.c` extension. In this example, enter “rsa2048public” as the **key name**.

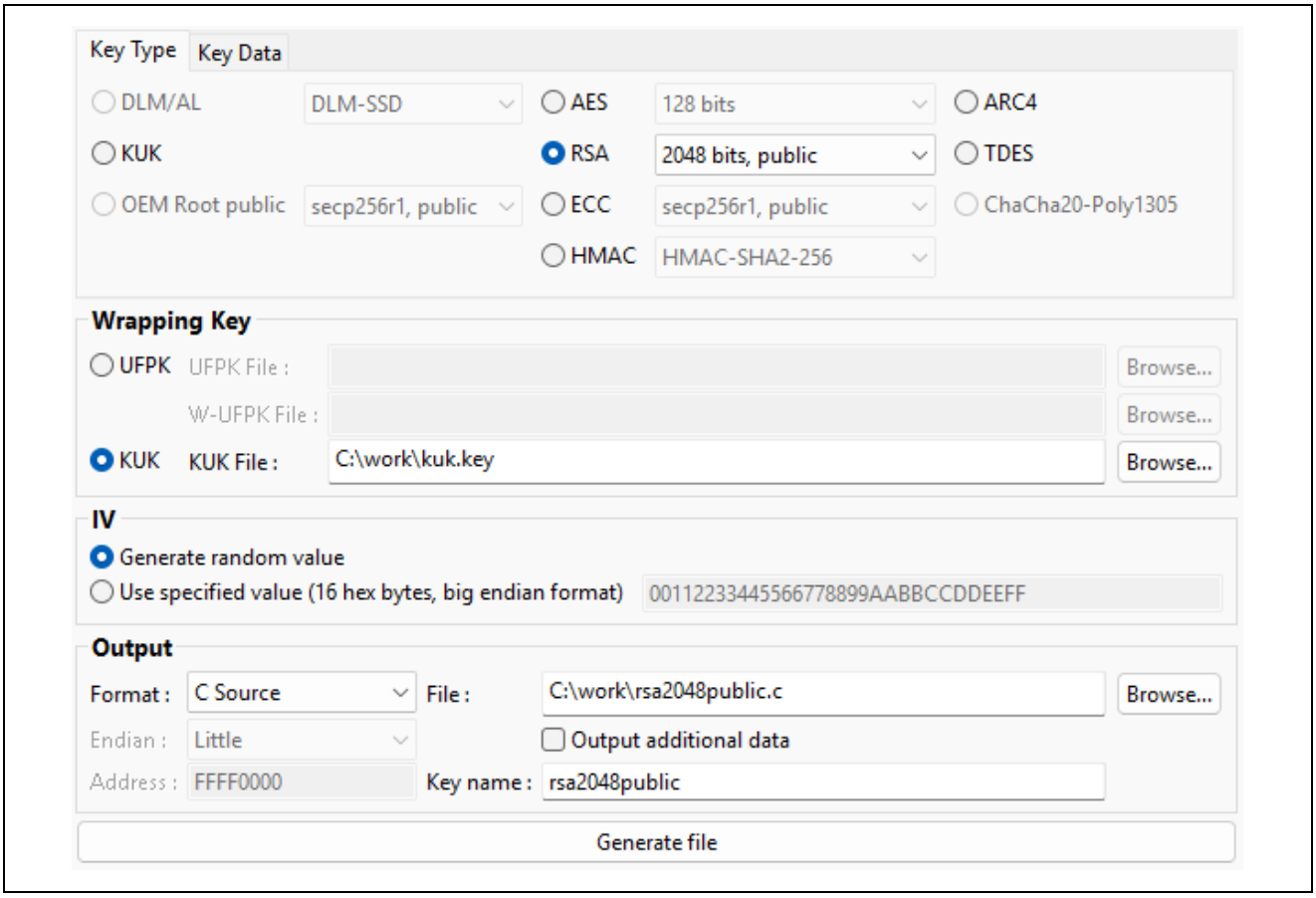


**Figure 6-21 [Wrap Key] - [Key Type] Tab, Example of creating an RSA 2048 Public key file as C source file**

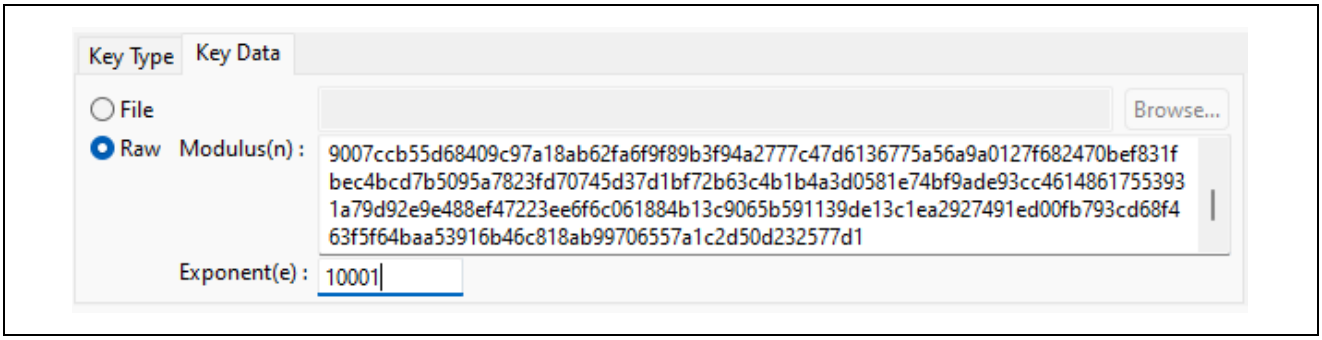


Figure 6-22 [Wrap Key] – [Key Data]Tab, Example of creating an RSA 2048 Public key file as C source file

Click the **Generate file** button to generate the specified output file. If the file is generated successfully, the tool will output the following execution result.

```
Output File: C:\work\rsa2048public.h
Output File: C:\work\rsa2048public.c
KUK: D0AEC19726CBC0E2FB403866B9B465A6C0D05B7A60362D5F435F9A3E98C79084
IV: E16B94C67F90F4D193021D54B939FF78
Encrypted key:
47AEE77D4E11BBB8B3A9B8D4C669C57D4B8A3B0F4BC0416835AEB7260BAA90ADECB99AD68D8BD1233B8D90EDA
94F41608497DBA7644716C241EFF3807260503749DB43632A84365C1E8F7DCC08690923F6C8A7C119643C412EE9B93
C34A0F3554791D4FC39E62ED6899556FC1A62163EEAB2E93EEE8CCFF485FEF6662C1EDF4037D1D796A2BA2F616FB
C5C61074E0E589165929973EE38C29698ED08B4FDCD52ADE1D85BAC8B740FB6F51F171CDBC9563EC932A35052CD
993E8FDFA6B98B4534414F46DF5DBF61A0A272F8A39CC3C010F0F1073B5F1BC4C71B00C3D79DC85E0FC5E2C97117
73E219CE49A67366DC7DB85085111949D4C115D387005F1B0EBF0959A873E9F6B21DAC16F8A24E3826A668377A562
DFEF011137E8258E84AD2C883
OPERATION SUCCESSFUL
```

**Figure 6-23 Example execution result, creating an RSA 2048 Public key file as C source file**

### 6.3.2 Using the CLI version

1. Create the RSA 2048 Public key file as a C source file by using the **genkey** command:

```
> skmt.exe /genkey /kuk file="C:\work\kuk.key" /mcu "RA-SCE9" /keytype "RSA-2048-public"
        /key "bad47a84c1782e4dbdd913f2a261fc8b65838412c6e45a2068ed6d7f16e9cdf4
              462b39119563cafb74b9cbf25cfd544bdae23bff0ebe7f6441042b7e109b9a8a
              faa056821ef8efaab219d21d6763484785622d918d395a2a31f2ece8385a8131
              e5ff143314a82e21afd713bae817cc0ee3514d4839007ccb55d68409c97a18ab
              62fa6f9f89b3f94a2777c47d6136775a56a9a0127f682470bef831fbec4bcd7b
              5095a7823fd70745d37d1bf72b63c4b1b4a3d0581e74bf9ade93cc4614861755
              3931a79d92e9e488ef47223ee6f6c061884b13c9065b591139de13c1ea292749
              1ed00fb793cd68f463f5f64baa53916b46c818ab99706557a1c2d50d232577d1
              00010001"
        /filetype "csource" /output "C:\work\rsa2048public.c" /keyname “rsa2048public”
```

Use the KUK file generated in section 6.2 Example 2 – Inject a Key-Update Key on an RA Family MCU with SCE9 Protected Mode.

```
C:\work\skmt\tool>skmt.exe /genkey /kuk file="C:\work\kuk.key" /mcu "RA-SCE9" /keytype
"RSA-2048-public" /key "bad47a84c1782e4dbdd913f2a261fc8b65838412c6e45a2068ed6d7f16e9cdf
4462b39119563cafb74b9cbf25cfd544bdae23bff0ebe7f6441042b7e109b9a8afaa056821ef8efaab219d2
1d6763484785622d918d395a2a31f2ece8385a8131e5ff143314a82e21afd713bae817cc0ee3514d4839007
ccb55d68409c97a18ab62fa6f9f89b3f94a2777c47d6136775a56a9a0127f682470bef831fbec4bcd7b5095
a7823fd70745d37d1bf72b63c4b1b4a3d0581e74bf9ade93cc46148617553931a79d92e9e488ef47223ee6f
6c061884b13c9065b591139de13c1ea2927491ed00fb793cd68f463f5f64baa53916b46c818ab99706557a1
c2d50d232577d100010001" /filetype "csource" /output "C:\work\rsa2048public.c" /keyname
"rsa2048public"
Output File: C:\work\rsa2048public.h
Output File: C:\work\rsa2048public.c
KUK: D0AEC19726CBC0E2FB403866B9B465A6C0D05B7A60362D5F435F9A3E98C79084
IV: BED48489C13FF9173A6F7649DEB6EB47
Encrypted key: FBB110AAAA2260E6033B2D9839A71121C137ABE34D1FE59D50C7DB2F0B568AA2D5C0A0CF
92CA7D093A624E931BFC6115A09DBED02CA3E85909940D2FE2921C5589D4D858A349F54FFFEBF8A18D94139
909BF57A0DEB219A99C640993990B3B837132F404CCCCB821AF1D28C8895C968863E293E22A87365BEC0748
1B856275BEC8E44EC9BFA392F586CEBB674C73230834C0B7989011E8DCEA6C71DE314D381788A129A42C541
27CA2FBD315A3F8BA9960B25C7B6A999D915443358C755A52D77608CFE4A48B9EEB46CFDA0AB96CEC209EB2
01F2EA2C2382564C30A3622E57071247B2E386160C5D21A00119ED4DD118B07554E72BB844FFA2AE9EF7C3D
28D9D0C116490388A554EC39D7D515C89C8459A42FD4495B6DF14ADB69D082D356DED
```

**Figure 6-24 CLI example of creating an RSA 2048 Public key file as C source file**

## 6.4 Example 4 – Use RX Family TSIP Secure Update

A secure firmware update solution employing the TSIP can be used to encrypt the user program, send the encrypted user program to the target device, and then decrypt and update the user program on the device. For how to make use of the required drivers and sample code, refer to the RX TSIP documentation linked to in *Table 1-1*, *MCU/MPU Related Information*.

### 6.4.1 Using the GUI Version

1. Selecting the MCU/MPU and Security Engine
   Select the MCU/MPU and security engine on the **[Overview]** tab.

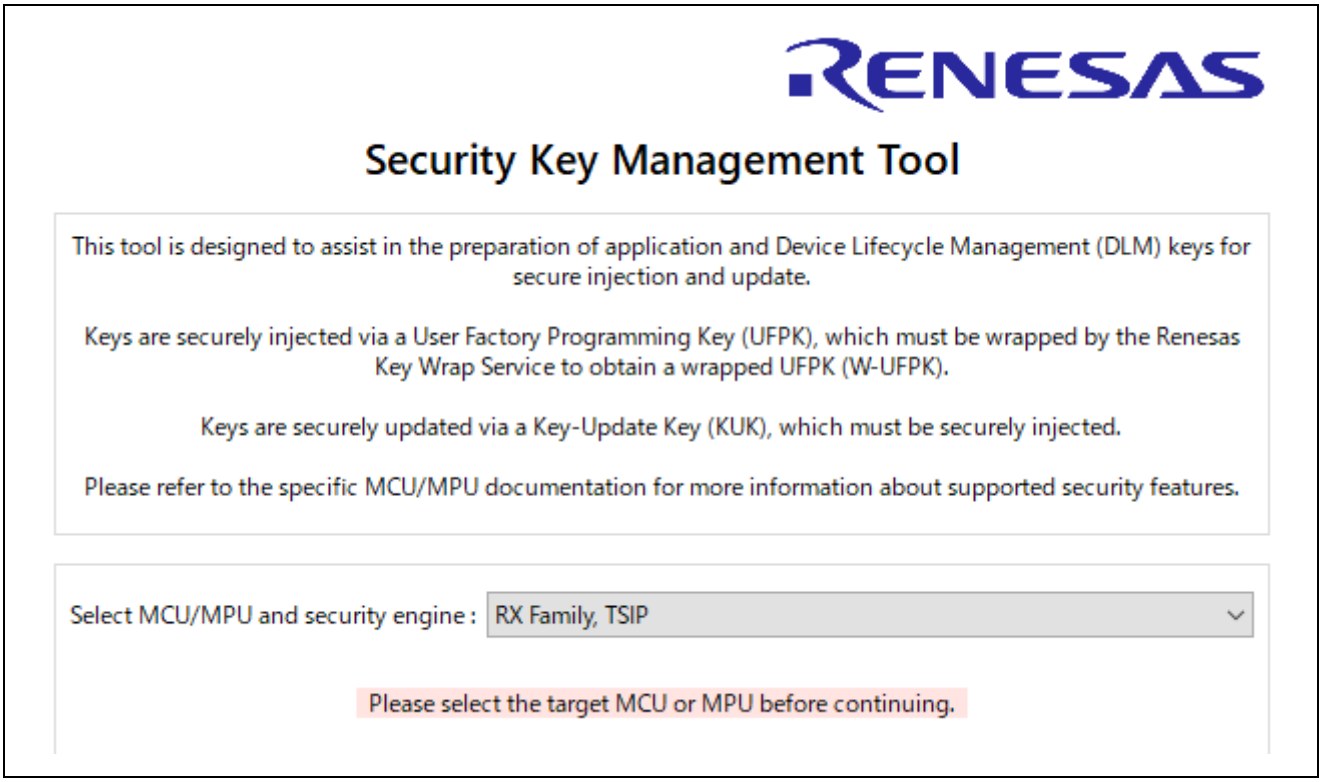


**Figure 6-25 [Overview] Tab**

2. Generating a Firmware Image File
   On the **[TSIP Update]** tab, generate a firmware image with RSU header appended. For **Output image** select **Secure Update**, and for **Firmware image** specify the firmware MOT file to be encrypted.
   On the **[RSU Header]** tab, set **Image flags** to **TESTING** and **RSU header version** to 1.
   On the **[Encrypted Address Range]** tab, set the address range to be encrypted and whether or not Data Flash data will be added. In this example, **Data Flash** is set to **Do not add**.
   For **Session Key Encryption Key** on the **[Image Encryption Key]** tab, specify the key for encrypting the encryption key set in the secure boot program. For simplicity, **Use random number generator** is selected for **Image Encryption Session Key** in this example.
   **Use random number generator** is also selected on the **[IV]** tab in this example for simplicity.
   Select **Binary** as the format under **Output**, and specify a file name with the extension *.rsu.

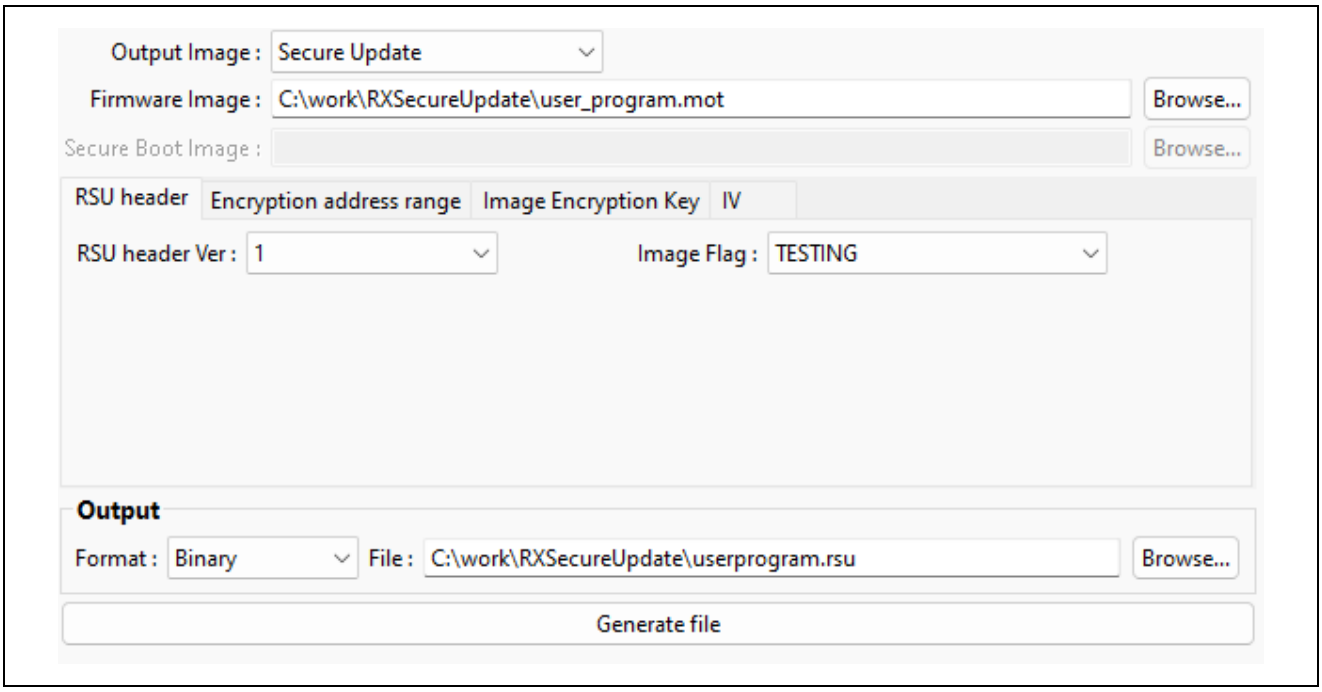


**Figure 6-26 [TSIP Update] – [RSU Header] Tab: Example Settings**

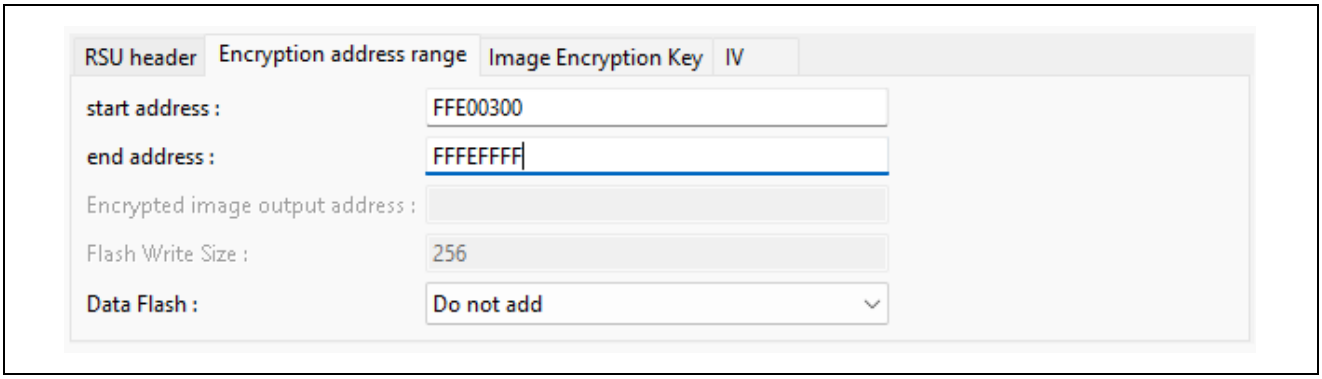


**Figure 6-27 [TSIP Update] – [Encrypted Address Range] Tab: Other Example Settings**

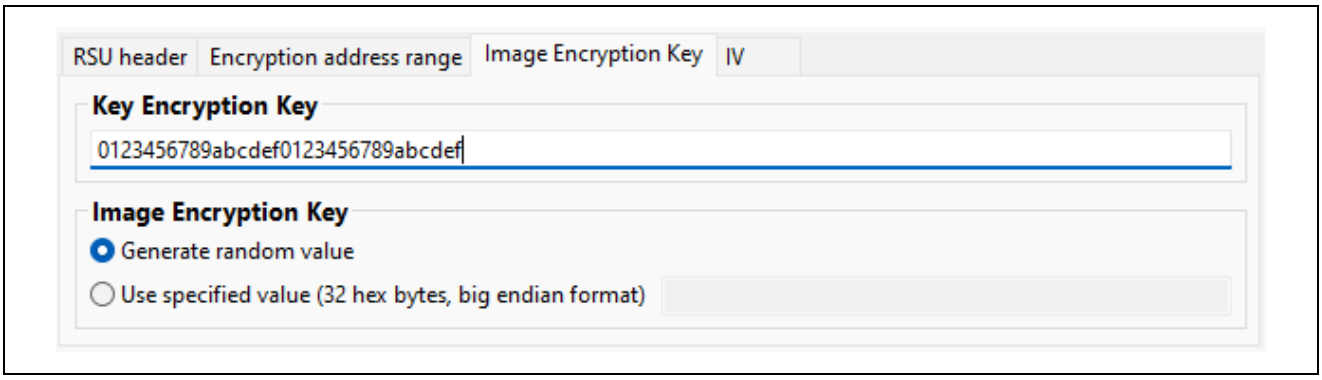


**Figure 6-28 [TSIP Update] – [Image Encryption Key] Tab: Example Settings**

**Figure 6-29 [TSIP Update] – [IV] Tab: Example Settings**

Output similar to the following appears when the operation completes successfully.

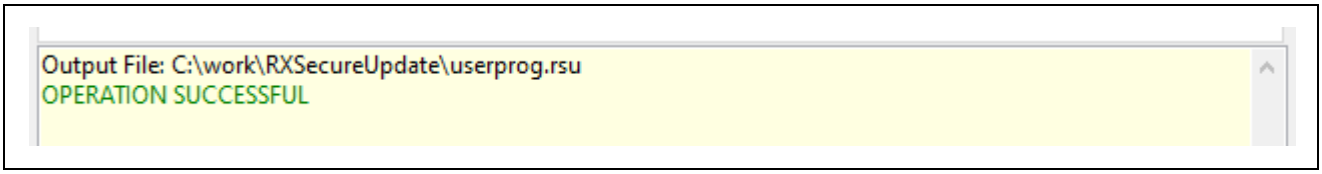


**Figure 6-30 [TSIP Update] Tab: Execution Example**

### 6.4.2 Using the CLI Version

1. Run the **enctsip** command in the terminal emulator program.

**> skmt.exe /enctsip /mode "update" /ver "1" /prg "C:\work\RXSecureUpdate\user_program.mot"**
**/enckey "0123456789abcdef0123456789abcdef"**
**/startaddr "FFE00300" /endaddr "FFFEFFFF" /imgflg "testing" /filetype "bin"**
**/output "C:\work\RXSecureUpdate\userprog.rsu"**

```
C:\work\skmt\tool>skmt.exe /enctsip /mode "update" /ver "1" /prg "C:\work\RXSecureUpdate\use
r_program.mot" /enckey "0123456789abcdef0123456789abcdef" /startaddr "FFE00300" /endaddr "FF
FEFFFF" /imgflg "testing" /filetype "bin" /output "C:\work\RXSecureUpdate\userprog.rsu"
Output File: C:\work\RXSecureUpdate\userprog.rsu
```

**Figure 6-31 enctsip Command Execution Example**

## 6.5 Example 5 – RA Family Generate FSBL Key Certificate / Code Certificate

Generates Key Certificate and Code Certificate that can be used in Renesas devices with First Stage Bootloader (FSBL) functionality. The Key Certificate and Code Certificate are used to verify the validity of the OEM Bootloader when it is written to the device using the Renesas Flash Programmer.

Please refer to *Table 1-1* MCU/MPU-related website documents for information on how to utilize the required tools and samples.

### 6.5.1 Using the GUI Version

1. Selecting the MCU/MPU and Security Engine
   Select the MCU/MPU and security engine on the **[Overview]** tab.

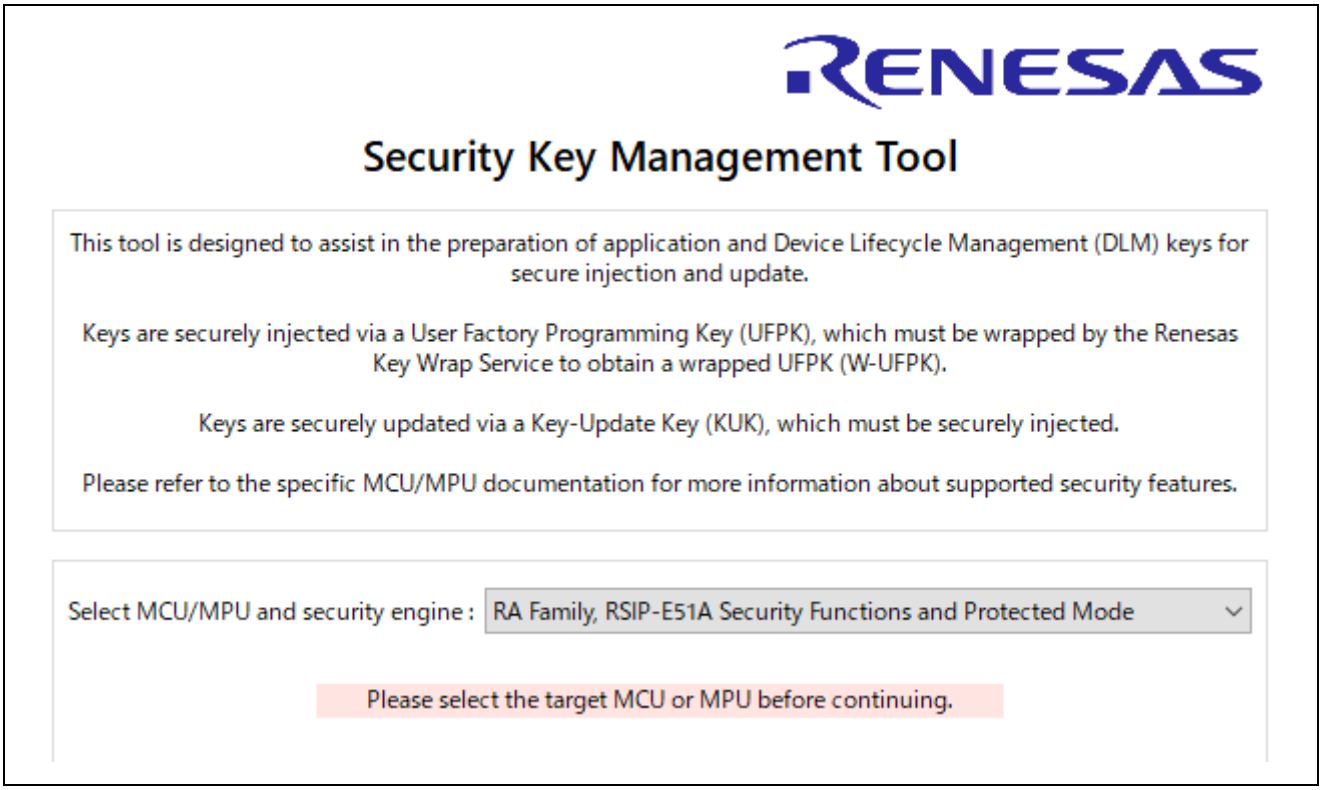


**Figure 6-32 [Overview] Tab, Generate Key Certificate and Code Certificate with RA Family RSIP-E51A Security Functions and Protected Mode**

2. Generate Key Certificate and Code Certificate

   Use [**FSBL**] tab to generate a Key Certificate and Code Certificate.

   In this example, the procedure is described for the case where the **Programming Verification Method** is **Signature**.

   Specify the OEM Bootloader mot file to be signed in **OEM Bootloader Image**.

   Select **Signature** in **Programming Verification Method**.

   Set **Image Version** to **1**.

   Configure the settings in [**Certificates**] tab.

   Set the starting address of the OEM Bootloader to **Code Flash Start Address (hex)**. In our example, set 02000000.

   Select the size of the Code Flash for the device to be used in **Device Code Flash Size**. In our example, select **2MB**.

   Under **OEM Bootloader Size (16-byte aligned)**, select **Calculate automatically**.

   Specify the file name of the Key Certificate in **Key Certificate**.

   Specify the file name of the Code Certificate in **Code Certificate**.

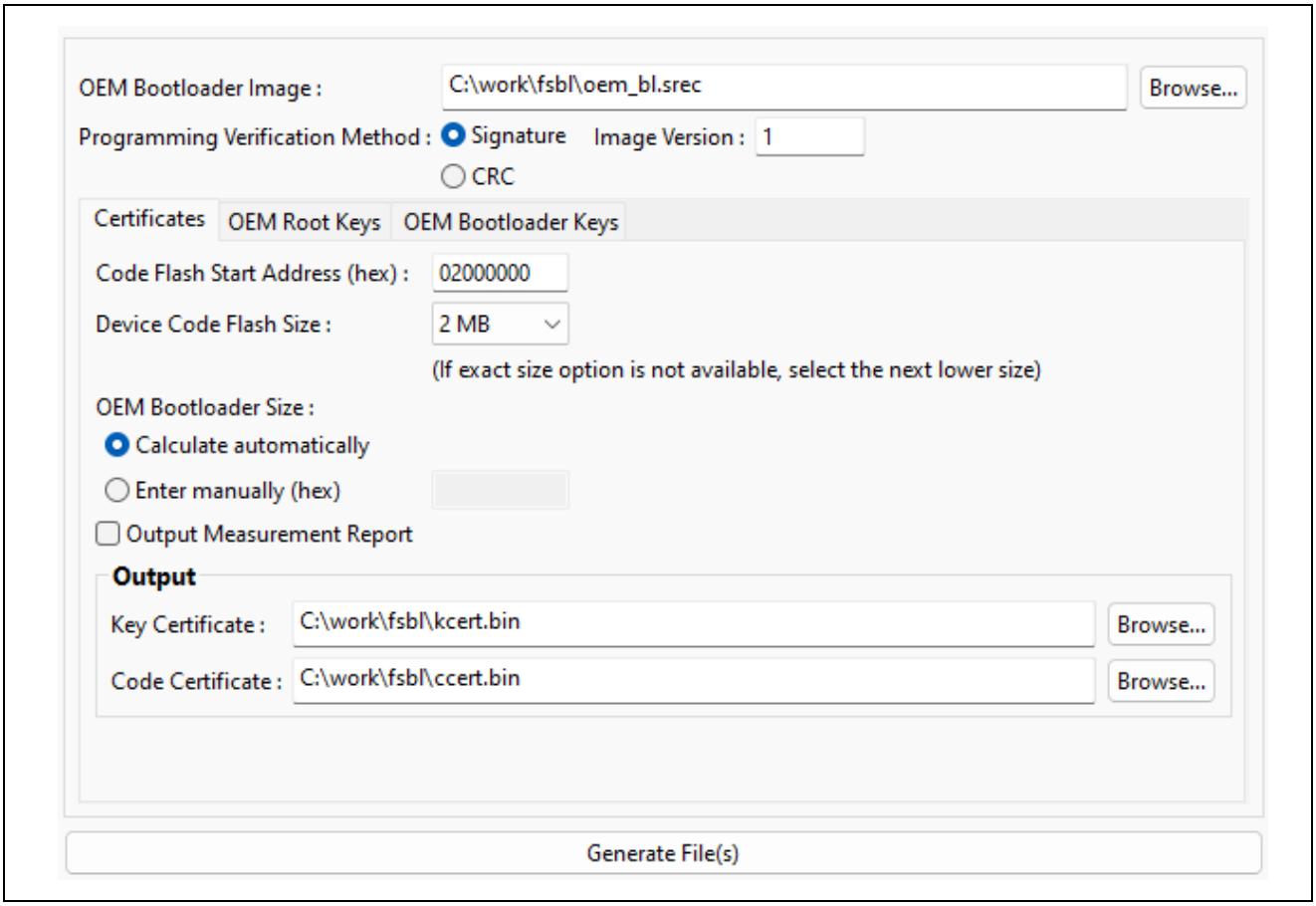


**Figure 6-33 [FSBL] – [Certificates] Tab, Setting Example**

3. Setting [**OEM Root Keys**] tab.
   Select **File** for **OEM Root Private Key** and specify a PEM file containing the public key of the OEM Bootloader Private Key. Since the PEM file is specified in **OEM Root Private Key**, it is not necessary to specify **OEM Root Public Key**.

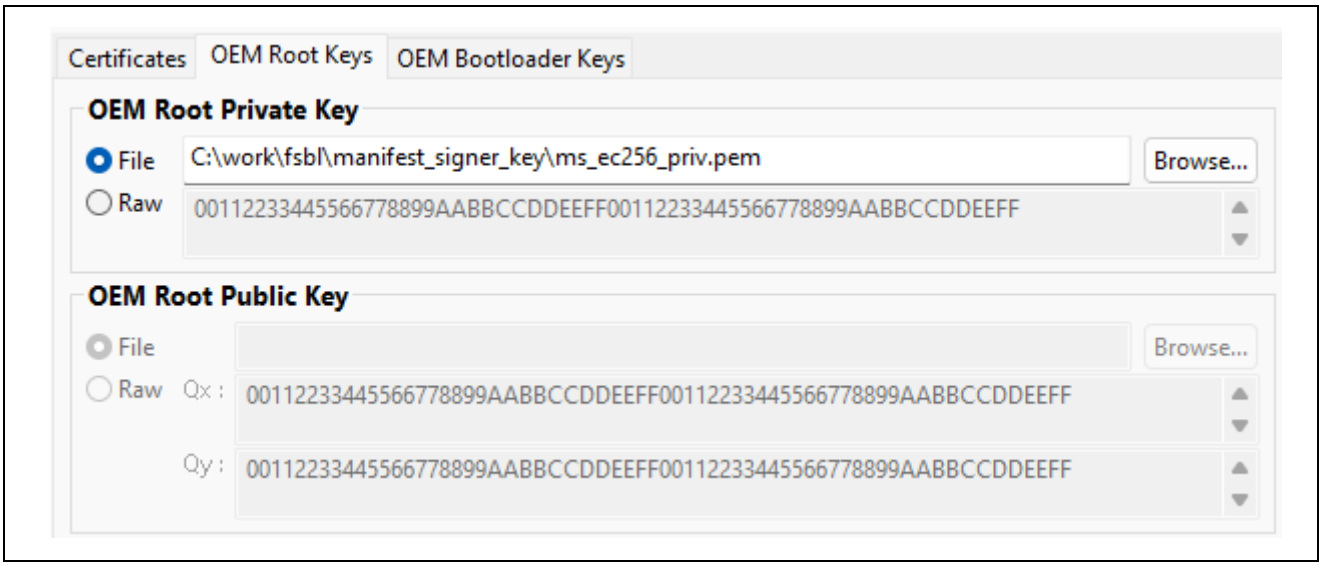


**Figure 6-34 [FSBL] – [OEM Root Keys] Tab, Setting Example**

4. Setting [**OEM Bootloader Keys**] tab.
   Select **File** for **OEM Bootloader Private Key** and specify a PEM file containing the public key of the OEM Bootloader Private Key. Since the PEM file is specified in **OEM Bootloader Private Key**, it is not necessary to specify **OEM Bootloader Public Key**.

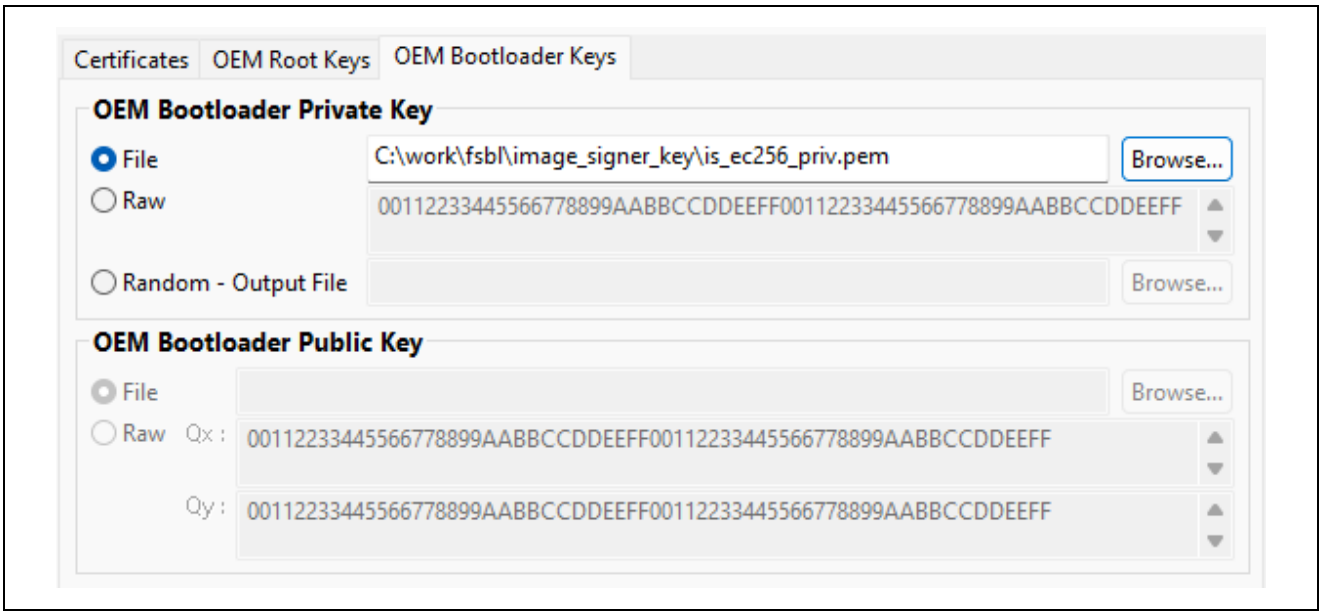


**Figure 6-35 [FSBL] – [OEM Bootloader Keys] Tab, Setting Example**

5. Generate Key Certificate and Code Certificate
   Click the **Generate File(s)** button to generate the Key Certificate and Code Certificate. If the files are successfully generated, the following results will be output.

```
Output File: C:\work\fsbl\kcert.bin
Output File: C:\work\fsbl\ccert.bin
OPERATION SUCCESSFUL
```

**Figure 6-36 Example of execution result, generate Key Certificate and Code Certificate**

### 6.5.2 Using the CLI version

1. Create Key Certificate and Code Certificate file by using the **gencert** command:

**> skmt.exe /gencert /mode "signature" /loadaddr "02000000" /cfsize "200000" /ver "1"**
**/oembl "C:\work\fsbl\oembl.srec"**
**/oembl_private file="C:\work\fsbl\manifest_signer_key\ms_ec256_priv.pem"**
**/oemroot_private file="C:\work\fsbl\image_signer_key\is_ec256_priv.pem"**
**/output_codecert "C:\work\fsbl\ccert.bin" /output_keycert " C:\work\fsbl\kcert.bin"**

```
C:\work\skmt\tool>skmt.exe /gencert /mode "signature" /loadaddr "02000000" /cfsize "200000"
/ver "1" /oembl "C:\work\fsbl\oembl.srec" /oembl_private file="C:\work\fsbl\manifest_signer_
key\ms_ec256_priv.pem" /oemroot_private file="C:\work\fsbl\image_signer_key\is_ec256_priv.pe
m" /output_codecert "C:\work\fsbl\ccert.bin" /output_keycert " C:\work\fsbl\kcert.bin"
Output File: C:\work\fsbl\kcert.bin
Output File: C:\work\fsbl\ccert.bin
```

**Figure 6-37 Execution result of CLI gencert command**

## 6.6 Example 6 – Using DOTF with RA Family

This example shows how to encrypt a portion of a user program that has been configured to use the DOTF functionality of an RA Family MCU.

Please refer to *Table 1-1* MCU/MPU-related website documents for information on how to utilize the required tools and samples.

### 6.6.1 Using the GUI version

1. Selecting the MCU/MPU and Security Engine
   Select the MCU/MPU and security engine on the **[Overview]** tab.

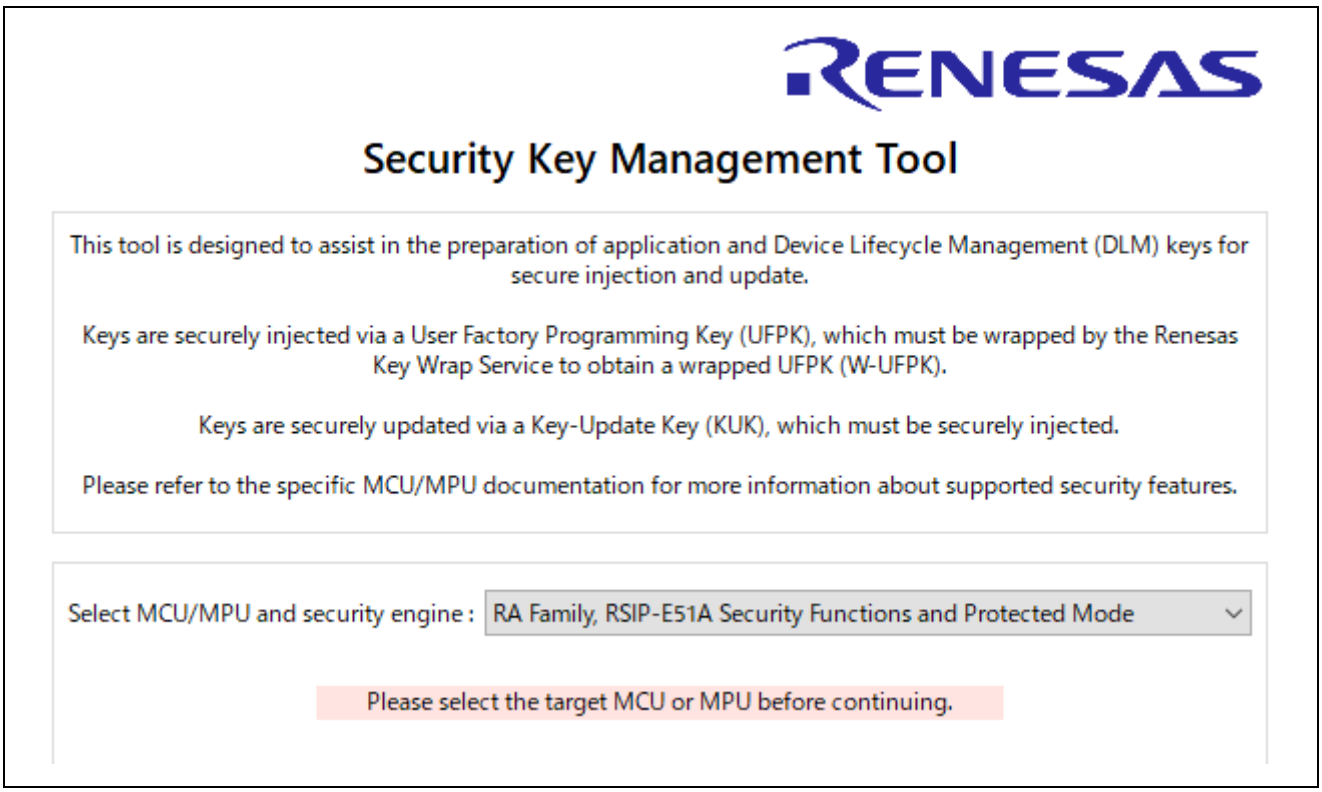


**Figure 6-38 [Overview] Tab, Decryption On-The-Fly Encryption with RA Family RSIP-E51A Security Functions and Protected Mode**

2. Setting Firmware image
   Use [**DOTF/OTFD**] tab to encrypt the firmware image.
   Set the firmware mot file to be encrypted in **Plaintext Image** field.
   Specify the address range of the encrypted image entered in **Plaintext Image** in **Encrypted Address Range**.
   Set **Destination Address** if the address to encrypt and the address to be placed are different.
   In the configuration example, set **Encryption address range** from 90000000 to 90000FFF
   **Destination address** is set **Same as Plaintext Image**.

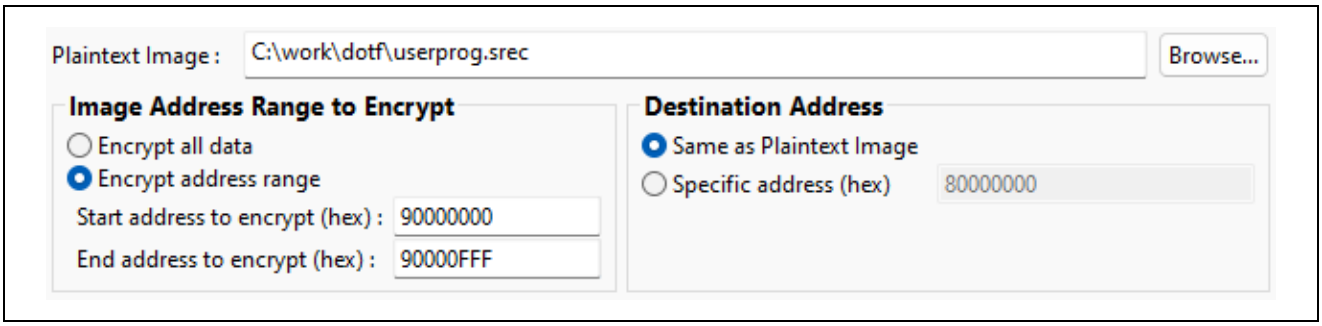


**Figure 6-39 [DOTF/OTFD] tab - Example settings for Plaintext Image, Encryption Range, and Destination Address**

3. Image Encryption Key and IV setting
   Set **Image Encryption Key** to the key to be used for encryption. In this example, an AES128 key is used.
   Specify the address range of the encrypted image entered in Plaintext Image in **IV** field. In this example, for simplicity, select **Use Random Number Generator** for IV.

**Figure 6-40 [DOTF/OTFD] tab - Image Encryption Key, IV Setting example**

4. Setting output file
   In **Encrypted image** specify the name of the output file.
   **Output Image Address and Contents** specifies the address of the output Motorola hexadecimal and whether to include the unencrypted plaintext data.
   For Output Image Address and Contents, select **Retain Original Address** and check the check box of **Include plaintext image data that is outside the encrypted address range**.

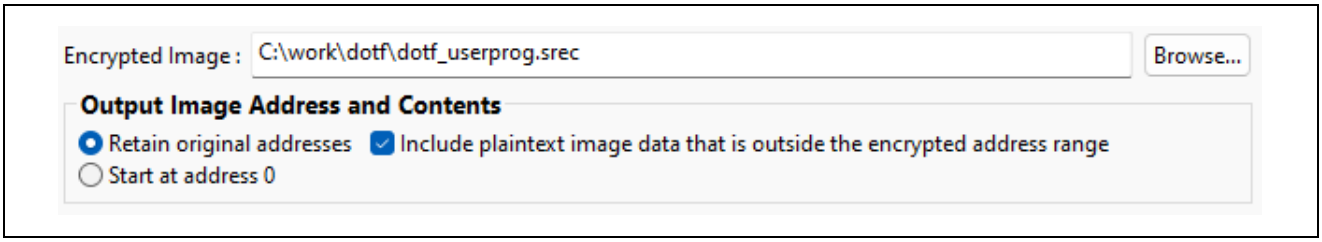


**Figure 6-41 [DOTF/OTFD] tab – Encrypted Image and Output Image Address and Contents**

Click the **Generate Encrypted Firmware Image** button to generate the specified output file. If the file is generated successfully, the tool will output the following execution result.

**Figure 6-42 [DOTF/OTFD] tab – Example of execution result**

### 6.6.2 Using the CLI version

1. Encrypt user programs using **encdotf** command.

**> skmt.exe /encdotf /keytype "AES-128" /enckey "000102030405060708090A0B0C0D0E0F" /startaddr "90000000" /endaddr "90000FFF" /incplain /prg "C:\work\dotf\userprog.srec" /output "C:\work\dotf\dotf_userprog.srec"**

```
C:\work\skmt\tool>skmt.exe /encdotf /keytype "AES-128" /enckey "000102030405060708090A0B0C0D
0E0F" /startaddr "90000000" /endaddr "90000FFF" /incplain /prg "C:\work\dotf\userprog.srec"
/output "C:\work\dotf\dotf_userprog.srec"
Output File: C:\work\dotf\dotf_userprog.srec
Key: 000102030405060708090A0B0C0D0E0F
Counter: E3B9489DD945947EBC3BE98389000000
```

**Figure 6-43 Execution result of CLI encdotf command**

## 6.7 Example 7 – RA Family using Secure Factory Programming

Outputs a Secure Factory Programming file (`*.sfp`) for use with the Secure Factory Programming function.

Please note, the Secure Factory Programming file does **not** contain the information shown in the table below. This information must be provided as a package to enable secure production programming.

Table 6-1 Additional Files Needed for Secure Production Programming

| Items | When Needed | How to Provide |
|---|---|---|
| User Key Injection Files | If the application requires keys that have been securely injected. | Generate all required `*.rkey` files using the **[Wrap Key]** tab of the GUI (*Section 3.6 [Wrap Key] Tab*) or the **genkey** CLI command (*Section 4.5* **genkey** Command Options). Note that the UFPK used for user key injection does not need to be the same UFPK used for Secure Factory Programming. |
| Key Certificate | If the application will use the First Stage Bootloader with signature authentication. | Generate any required binary Key Certificate files using the **[FSBL]** tab of the GUI (*Section 3.8 [FSBL] tab*) or the **gencert** CLI command (*Section 4.7* **gencert** command option) |
| Code Certificate | If the application will use the First Stage Bootloader with either signature authentication or CRC verification. | Generate any required binary Code Certificate files using the **[FSBL]** tab of the GUI (*Section 3.8 [FSBL] tab*) or the **gencert** CLI command (*Section 4.7* **gencert** command option) |

### 6.7.1 Using the GUI version

1. Selecting the MCU/MPU and Security Engine
   Select the MCU/MPU and security engine on the **[Overview]** tab.

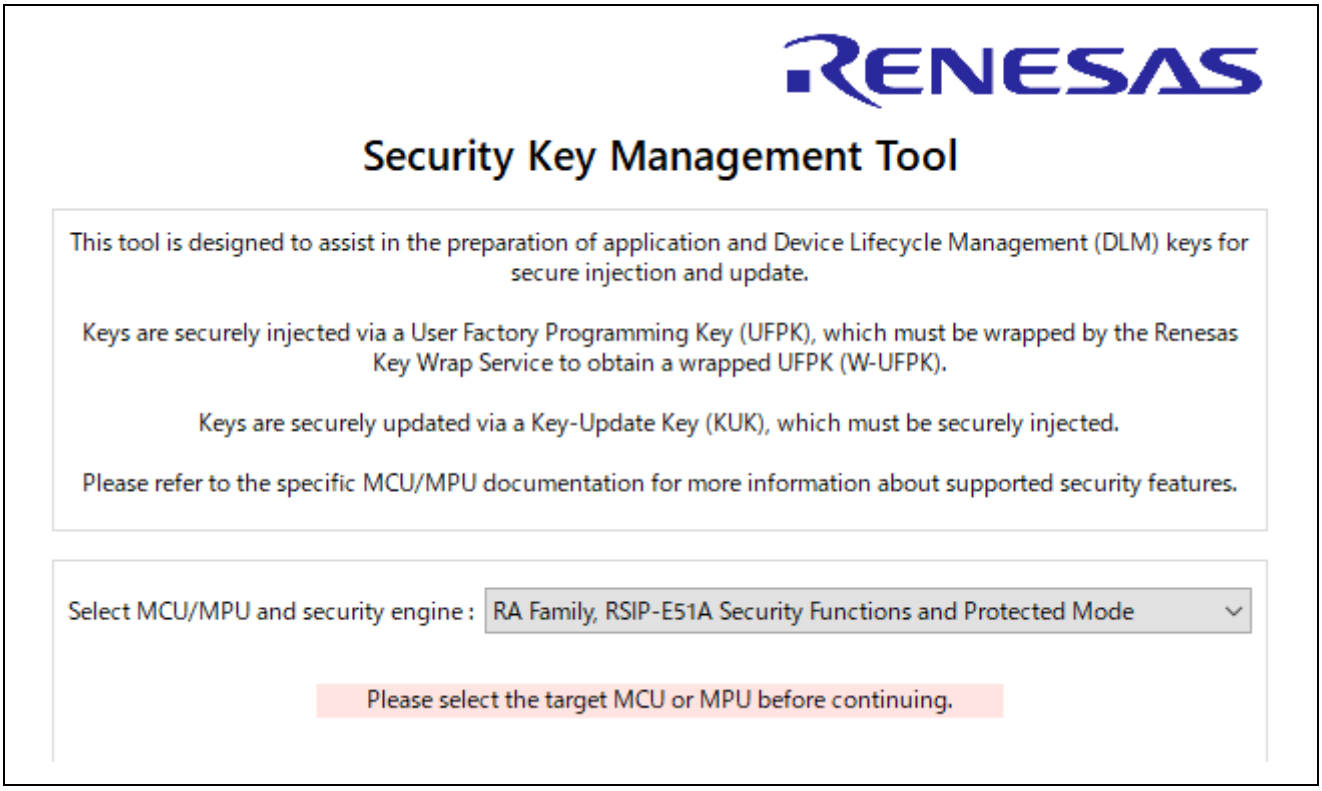


**Figure 6-44 [Overview] Tab, Secure Factory Programming with RA Family RSIP-E51A Security Functions and Protected Mode**

2. Setting [**Firmware Images**] tab

   Specify the user program to be encrypted in [**Firmware Images**] tab **Plaintext Image**. Select the file and click the Add button to add the file.

   Select the MCU/MPU information to be used in **MCU/MPU**. In this example, **RA8D1/M1/T1** is selected.

   Select the destination DLM/AL state after writing the firmware image in **Final DLM/AL State**.

   In this example, select **OEM PL0 with AL2_KEY**.

   Specify the output file name in **Secure Programming File(`*.sfp`)**.

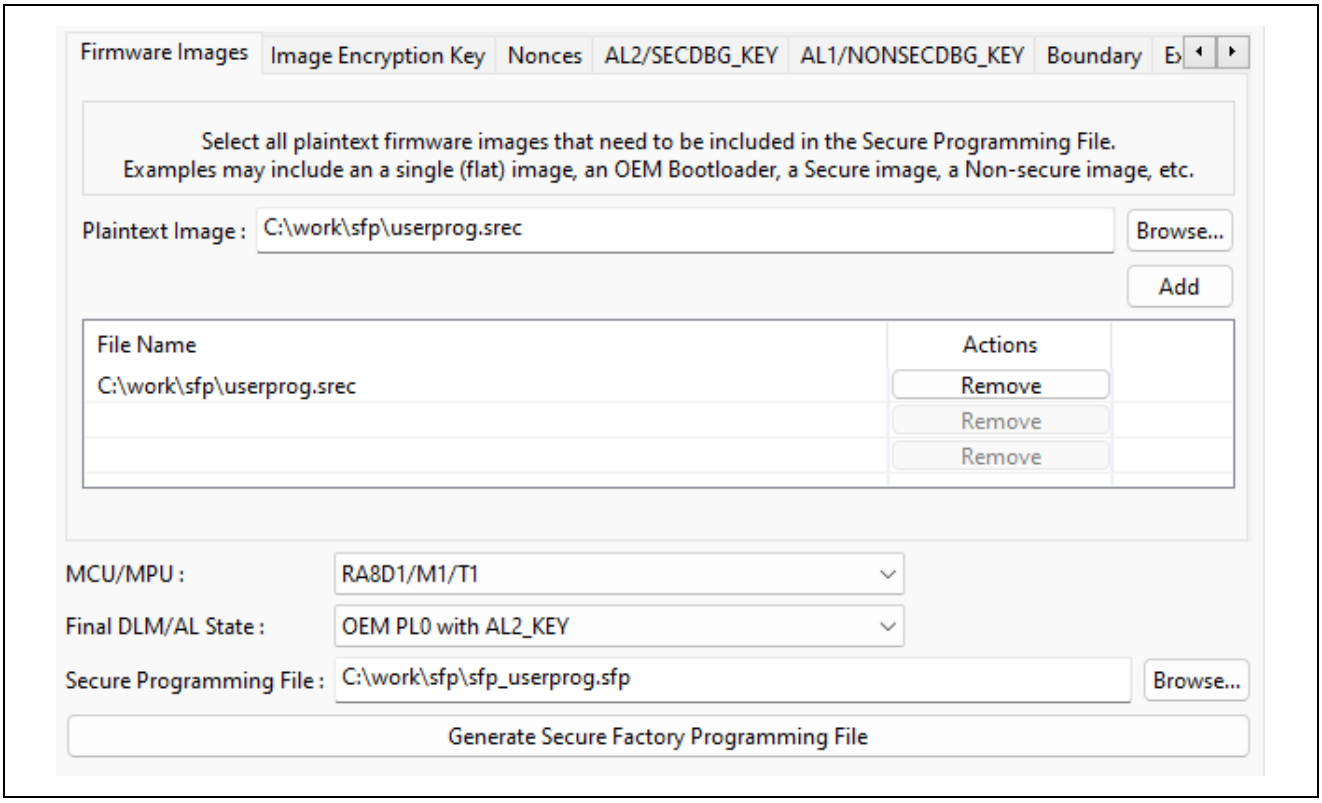


**Figure 6-45 [SFP] – [Firmware Images] Tab, Setting Example**

3. Setting [**Image Encryption Key**] tab
   Specify the AES 128-bit key data to be used for encryption of the user program and parameter information in **Key Data (AES-128)**.
   For simplicity, **IV** is set to **Generate random value** in this example.
   Specify UFPK file for **UFPK File** and W-UFPK file for **W-UFPK File**.
   Refer to *6.1.1* Steps 1-3 for how to generate UFPK and W-UFPK files.

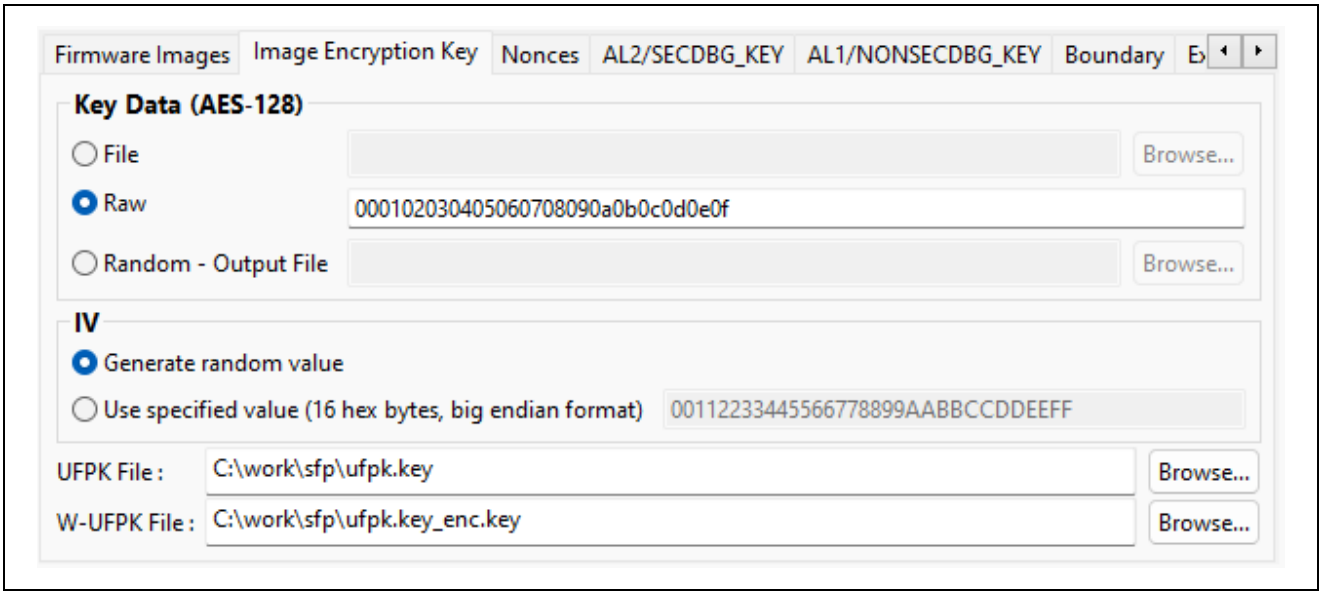


**Figure 6-46 [SFP] – [Image Encryption Key] Tab, Setting Example**

4. Setting [**Nonces**] tab.
   **IV (Programming parameters)** and **IV (Firmware image)** are for simplicity. **Generate random value** is selected in this example. **IV (AL/DLM key)** is not required to be selected in this example.

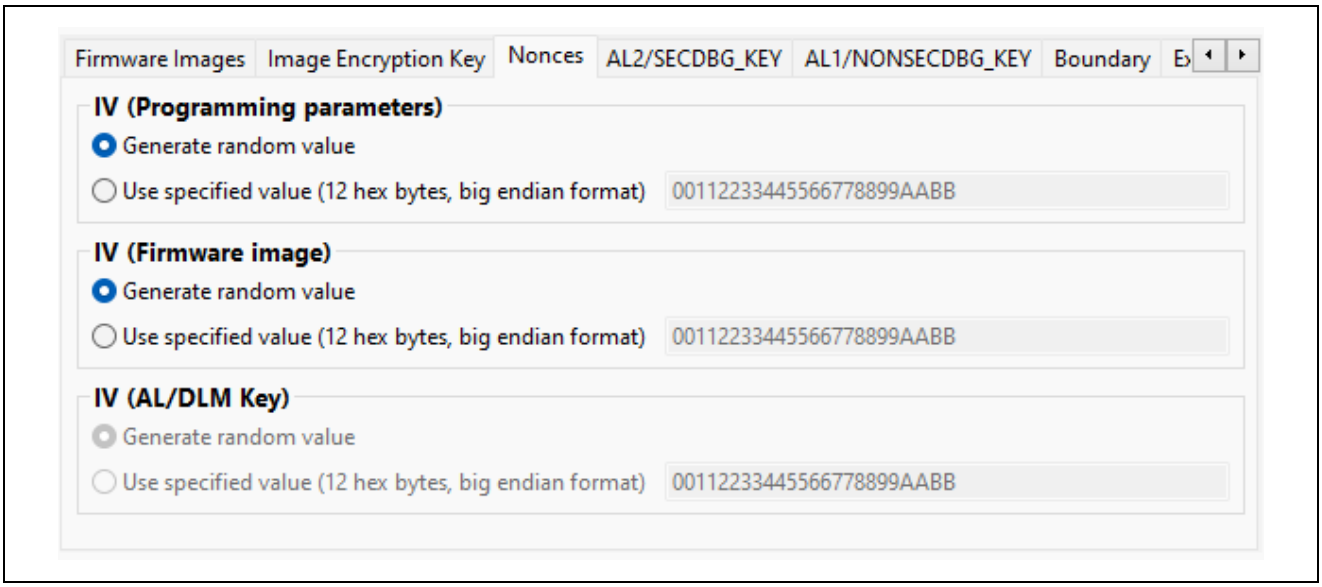


**Figure 6-47 [SFP] – [Nonces] Tab, Setting Example**

5. Setting [**AL2/SECDBG_KEY**] tab.
   Specify the AES128bit key data to be used as AL2_KEY in **Key Data**.
   For simplicity, **IV** is set to **Generate random value** in this example.

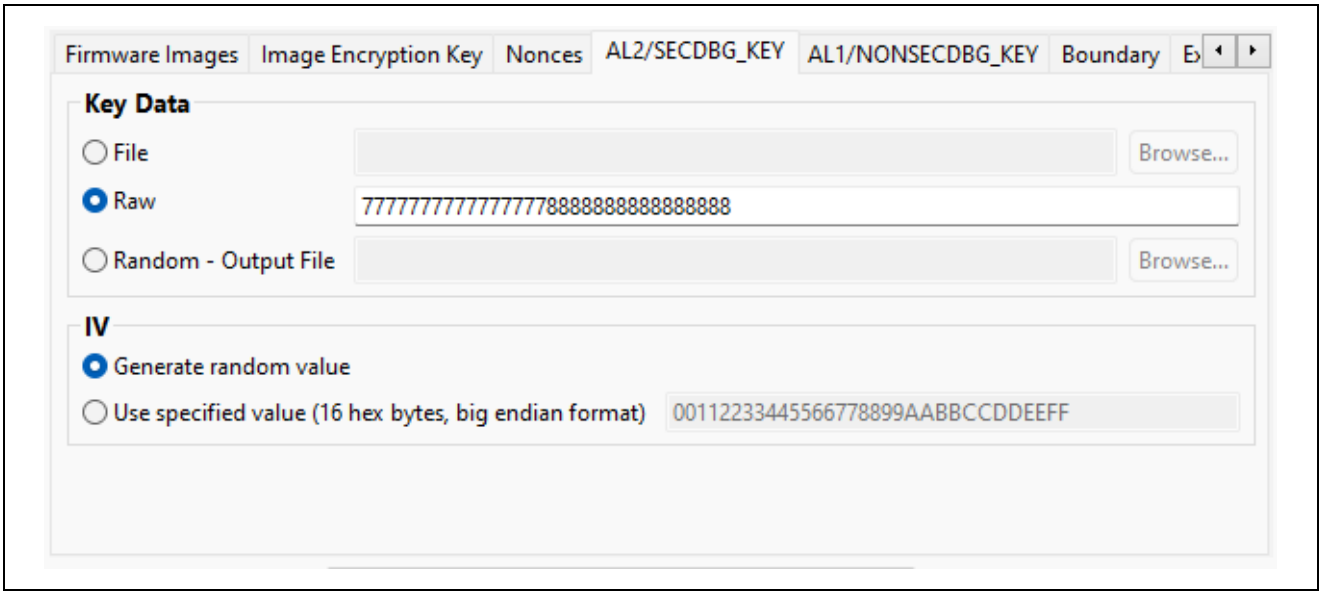


**Figure 6-48 [SFP] – [AL2/SECDBG_KEY] Tab, Setting Example**

Click the **Output File** button to generate the Secure Factory Programming file. If the file is successfully generated, the following results will be output.

```
Output File: C:\work\sfp\sfp_userprog.sfp
OPERATION SUCCESSFUL
```

**Figure 6-49 [SFP] tab – Example of execution result**

### 6.7.2 Using the CLI version

1. Encrypt user programs using **encsfp** command.

**>skmt /encsfp /mcu "RA8D1_M1_T1" /enckey "000102030405060708090a0b0c0d0e0f"**
**/trn "OEM_PL0_AL2" /prg "C:\work\sfp\userprog.srec"**
**/al2key "77777777777777778888888888888888"**
**/ufpk file="C:\work\sfp\ufpk.key" /wufpk "C:\work\sfp\ufpk.key_enc.key"**
**/output "C:\work\sfp\sfp_userprog.sfp"**

```
C:\work\skmt\tool>skmt /encsfp /mcu "RA8D1_M1_T1" /enckey "000102030405060708090a0b0c0d0e0f"
 /trn "OEM_PL0_AL2" /prg "C:\work\sfp\userprog.srec" /al2key "77777777777777778888888888888888
" /ufpk file="C:\work\sfp\ufpk.key" /wufpk "C:\work\sfp\ufpk.key_enc.key" /output "C:\work
\sfp\sfp_userprog.sfp"
Output File: C:\work\sfp\sfp_userprog.sfp


C:\work\skmt\tool>
```

**Figure 6-50 Execution result of CLI encsfp command**

## 6.8 Example 8 – Generate RA for Wireless Header

This tool includes functionality to embed headers, key certificates, and code certificates into Provisioning SW, OEM_BL or Application Firmware for RA for Wireless. This example demonstrates how to generate headers, key certificates, and code certificates for Application Firmware.

### 6.8.1 Using the GUI version

1. Selecting the MCU/MPU and Security Engine
   Select the MCU/MPU and security engine on the **[Overview]** tab.

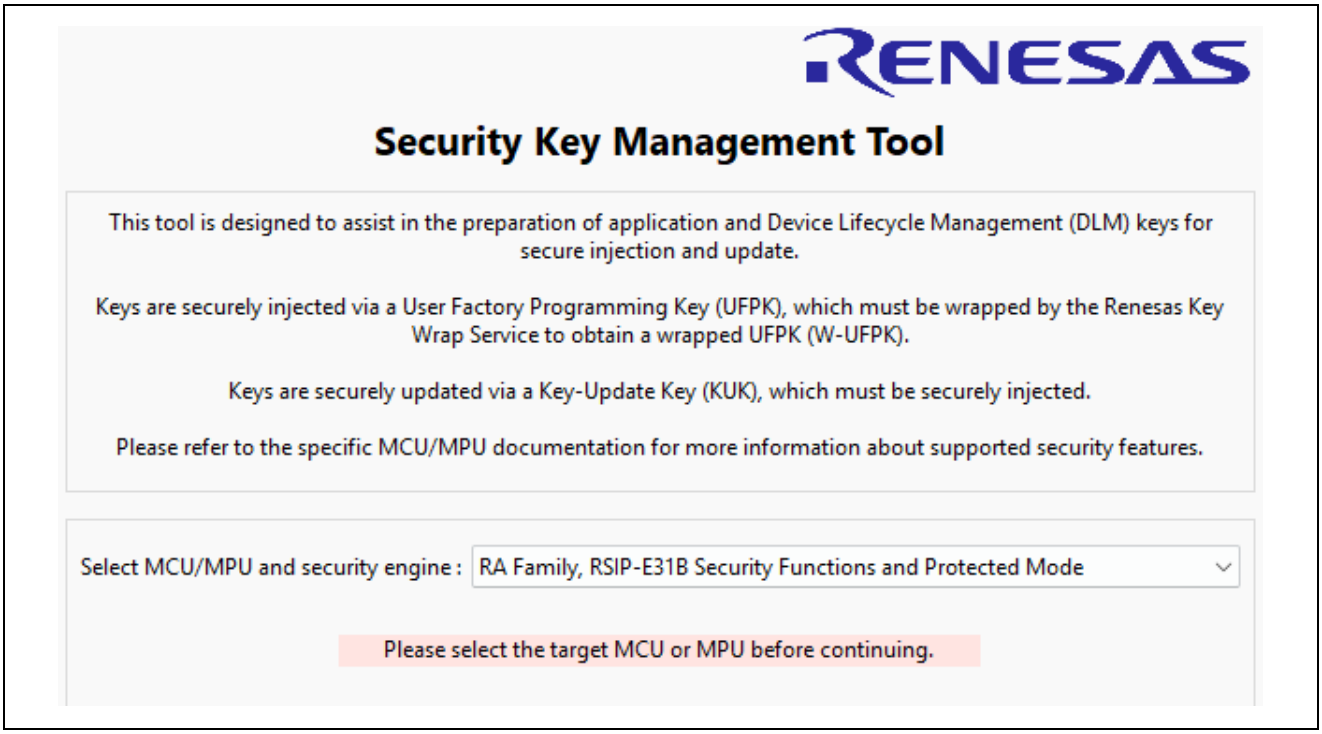


**Figure 6-51 [Overview] Tab, RA Family RSIP-E31B Security Functions and Protected Mode**

2. Setting Firmware Image

Use [**RA6B1/RA6U1**] tab to add a header to the Application Firmware file.

Specify the Motorola Hex file for the Application Firmware in **Firmware Image File**.
Set **Mode** to **Application** and **Header** to **sec_sign**.
Enter the IVT address from the Motorola Hex file in **IVT** field.
Enter the size of the Application Firmware to be signed in **Size** field.

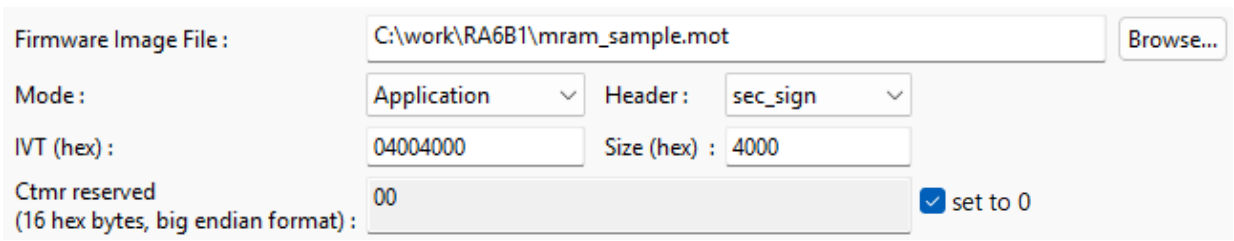


**Figure 6-52 [RA6B1/RA6U1] Tab, Setting Example of Firmware Image Configuration**

3. Setting [**Certificates**] Tab

Specify the ECDSA curve used for signing the key certificate and code certificate, as well as the OEM RoT key number for the key certificate and the firmware version for the code certificate.
In this example:
Specify **secp384r1** for the **Key Certificate** signature and **OEM RoT key number 0**.
Specify **secp384r1** for the **Code Certificate** signature and **FW version 1**.

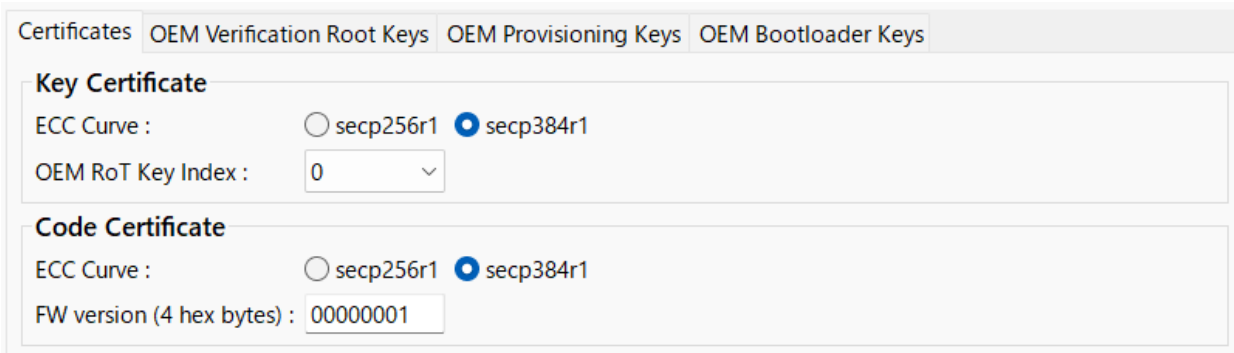


**Figure 6-53 [RA6B1/RA6U1] Tab – [Certificates] Tab, Setting Example**

4. Setting [**OEM Verification Root Keys**] Tab

Under **OEM Verification Root Private Key**, select **File** and specify a PEM file containing the OEM Verification Root Public Key. If you specify a PEM file under **OEM Verification Root Private Key**, there is no need to specify **OEM Verification Root Public Key**.

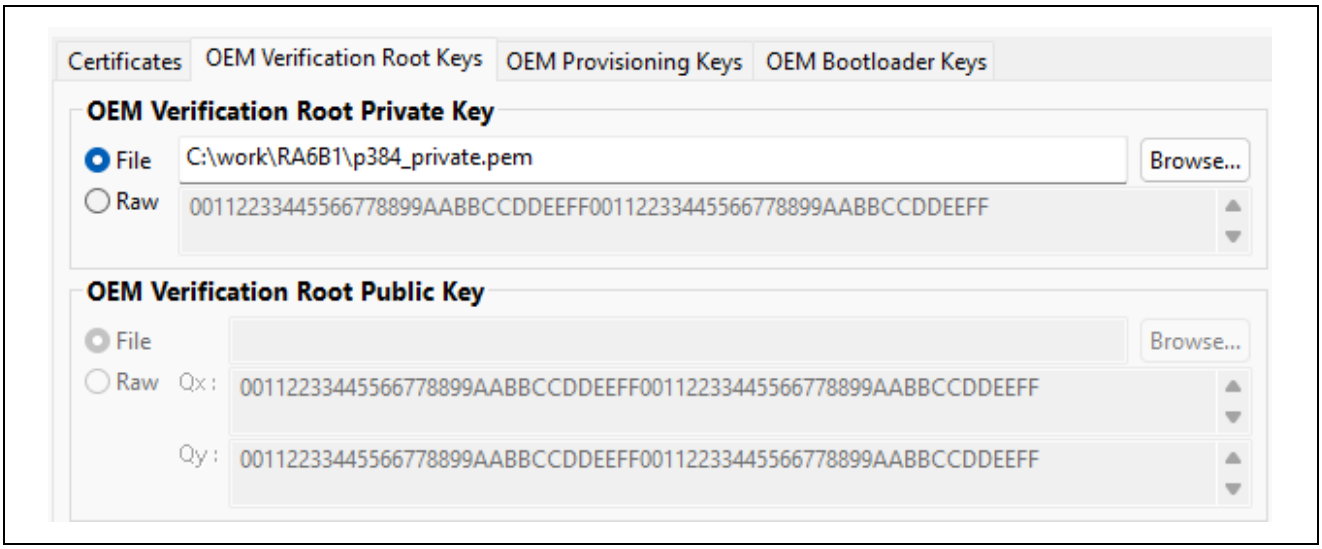


**Figure 6-54 [RA6B1/RA6U1] Tab – [OEM Verification Root Keys] Tab, Setting Example**

5. Setting [**OEM Bootloader Keys**] Tab

Select a file under **OEM Bootloader Private Key** and specify a PEM file containing the OEM Bootloader Public Key. If you specify a PEM file under **OEM_BL Verification Private Key**, there is no need to specify the **OEM Bootloader Public Key**.

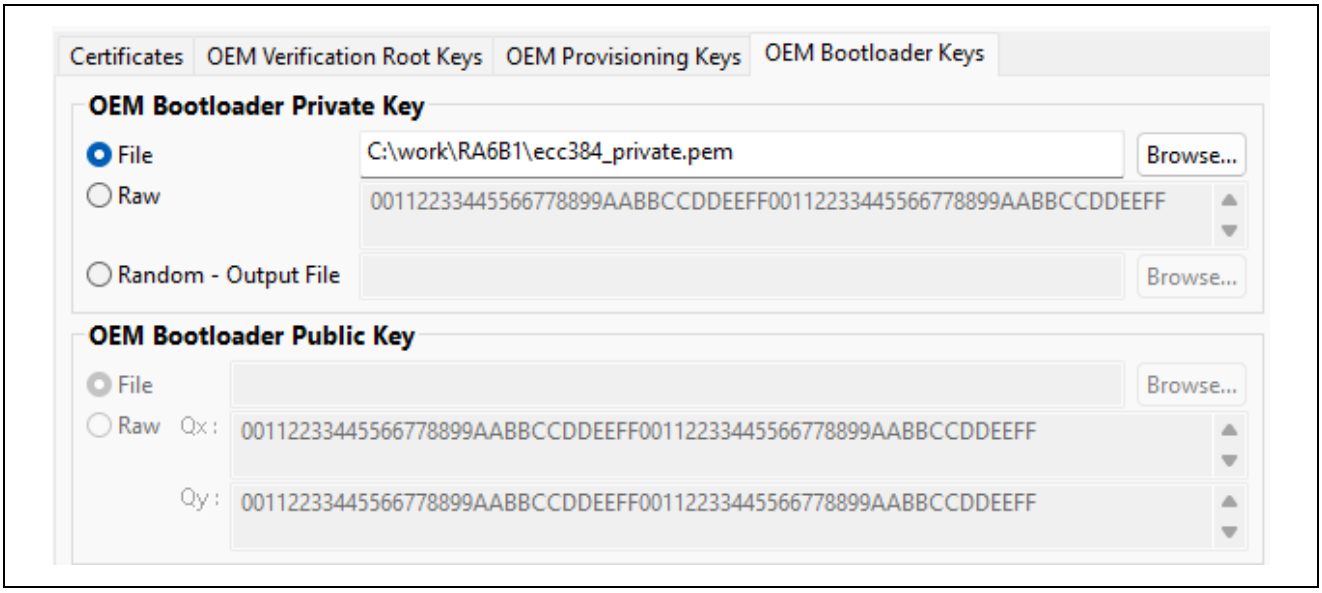


**Figure 6-55 [RA6B1/RA6U1] Tab – [OEM Bootloader Keys] Tab, Setting Example**

6. Setting Output file

Specify **file** for the output file.
In this example, we specify the output as a Motorola Hex file.

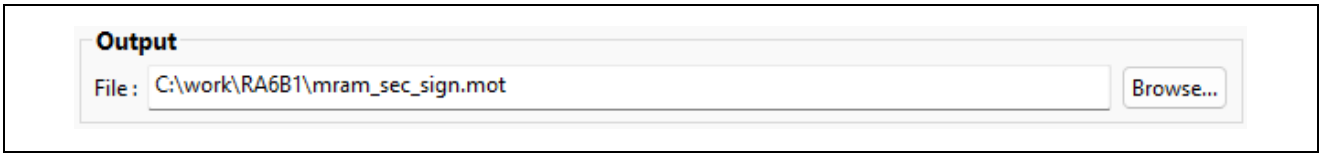


**Figure 6-56 [RA6B1/RA6U1] Tab – Specified the output filename**

When you click the **Generate File** button, a file will be generated that includes the header, key certificate, and code certificate appended to Application Firmware file you entered. If the file is generated successfully, the following output will be displayed.

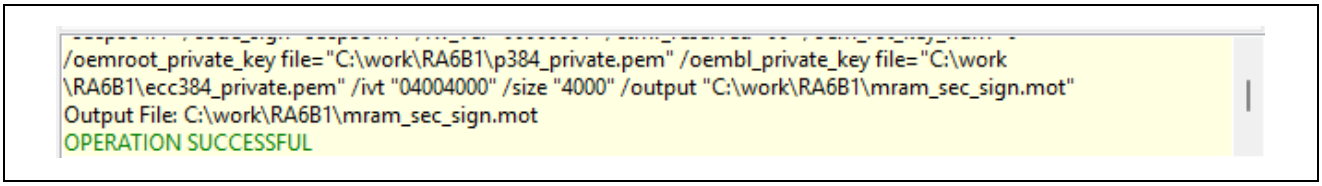


**Figure 6-57 [RA6B1/RA6U1] Tab – Example of execution result**

### 6.8.2 Using the CLI version

1. Create Key Certificate, Code Certificate file and headers by using the **genhdr** command:

**>skmt /genhdr /prg "C:\work\RA6B1\mram_sample.mot" /mode "app" /header "sec_sign" /key_sign "secp384r1" /code_sign "secp384r1" /fw_ver "00000001" /ctmr_reserved "00" /oem_rot_key_num "0" /oemroot_private_key file="C:\work\RA6B1\p384_private.pem" /oembl_private_key file="C:\work\RA6B1\ecc384_private.pem" /ivt "04004000" /size "4000" /output "C:\work\RA6B1\mram_sec_sign.mot"**

```
c:\work\skmt\tool>skmt.exe /genhdr /prg "C:\work\RA6B1\mram_sample.mot" /mode "app" /header "sec_sign" /key_sign "secp38
4r1" /code_sign "secp384r1" /fw_ver "00000001" /ctmr_reserved "00" /oem_rot_key_num "0" /oemroot_private_key file="C:\wo
rk\RA6B1\p384_private.pem" /oembl_private_key file="C:\work\RA6B1\ecc384_private.pem" /ivt "04004000" /size "4000" /outp
ut "C:\work\RA6B1\mram_sec_sign.mot"
Output File: C:\work\RA6B1\mram_sec_sign.mot


c:\work\skmt\tool>
```

**Figure 6-58 Execution result of CLI genhdr command**

# 7. Notes

## 7.1 Display settings when using Windows environment

In the Windows environment, if the standalone GUI or the e² studio plug-in version is set to a display setting other than the recommended setting, the entire dialog may not be displayed.

The display can be improved by the following ways.

1. Right-click on the executable file:
   Standalone version : `SecurityKeyManagementTool.exe`
   e² studio plugin version : `e2studio.exe`
   `e2studio.exe` is located in the **eclipse** folder of e² studio installation.
2. Select **Properties**, and then select the **Compatibility** tab. On the **Compatibility** tab, click the **Change high DPI settings** button.
3. In the **High DPI scaling override** section, check the **Override high DPI scaling behaviour** checkbox and select **System** in the pull-down list. Then click **OK** on both dialog boxes.

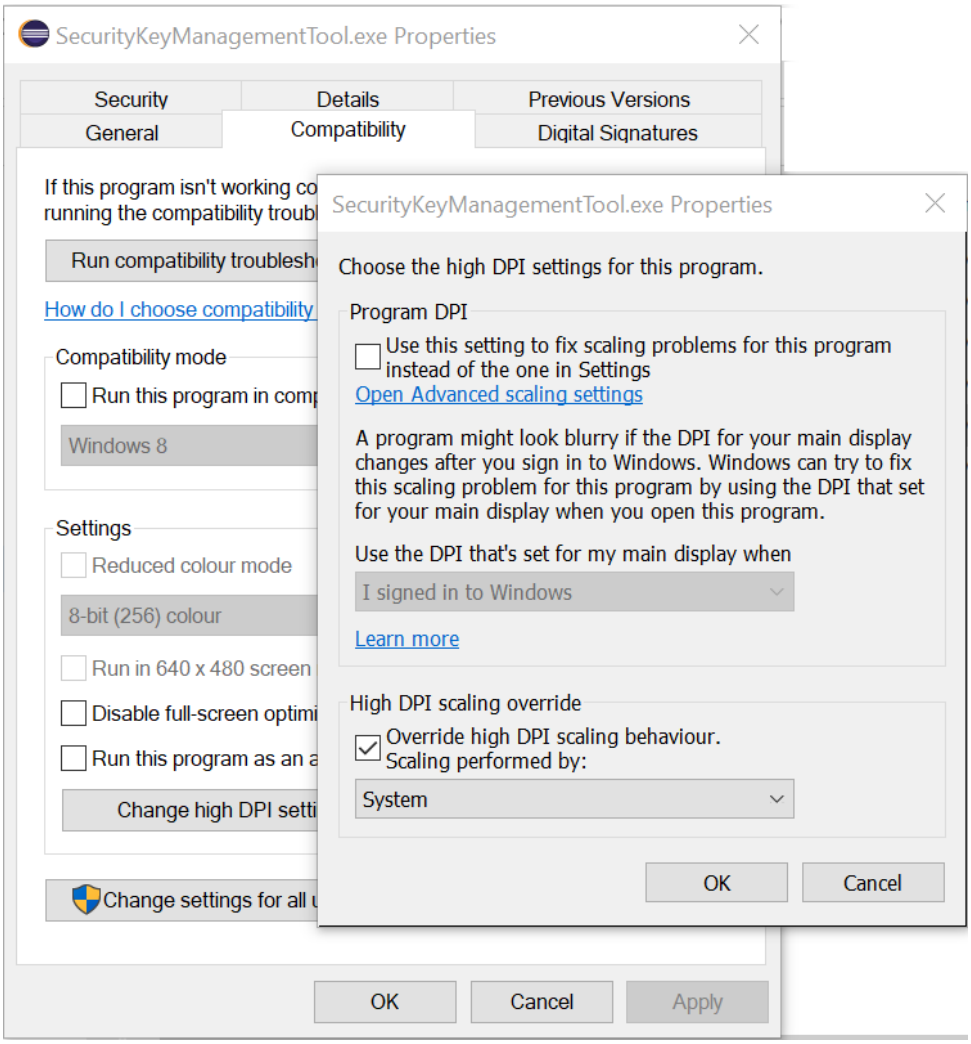


Figure 7-1 SecurityKeyManagementTool.exe Properties “High DPI scaling override” settings

## 7.2 Note on using e²studio plugin version in a Linux environment

If you are using the e²studio plug-in version in a Linux environment, you must set execution permissions on the command line executable file after installation. After installing the plug-in version of the Security Key Management Tool in e²studio, grant execution rights to the following files in the e²studio installation folder.

/eclipse/plugins/com.renesas.apltool.skmt_X.X.X.XXXXXXXXXXXX/cli/skmt

## 7.3 Note on using e²studio plugin version in a macOS environment

If you are using the e²studio plug-in version in a macOS environment, you must set execution permissions on the command line executable file after installation. After installing the plug-in version of the Security Key Management Tool in e²studio, grant execution rights to the following files in the e²studio installation folder.

/eclipse/plugins/com.renesas.apltool.skmt_X.X.X.XXXXXXXXXXXX/cli/skmt

## 7.4 Note The macOS version limitation

The macOS version has limitations on the following features of the command line, standalone, and e2studio plug-in versions

Table 7-1 Limitations of the command line version

| Command | Options | Limitations |
|---|---|---|
| **genkey** | Key generation by omitting the /key option when the following specific /keytype option is specified.<br>”brainpoolP256r1-private”<br>”brainpoolP384r1-private”<br>”brainpoolP512r1-private” | Not available.<br>The following error message appears<br>“Error: For BRAINPOOLPxxxR1-PRIVATE, key option cannot be omitted when using MAC OS.” |
| | PEM file input with /key option when the following specific /keytype option is specified.<br>”brainpoolP256r1-public”、<br>”brainpoolP256r1-private”、<br>”brainpoolP384r1-public”、<br>”brainpoolP384r1-private”、<br>”brainpoolP512r1-public”、<br>”brainpoolP512r1-private”、 | Not available.<br>The following error message appears<br>“Error: For BRAINPOOLPxxxR1-PRIVATE/PUBLIC, PEM file cannot be specified when using MAC OS.” |

Table 7-2 Limitations of the standalone and the e²studio plugin version

| Tab | Options | Limitations |
|---|---|---|
| Wrap Key | When the following key types are selected in the [KeyType] tab, select "Random" in the [KeyData] tab for key generation.<br>”brainpoolP256r1-private”<br>”brainpoolP384r1-private”<br>”brainpoolP512r1-private” | Not available.<br>When the Generate file button is pressed, the following error message is displayed.<br>“Error: For BRAINPOOLPxxxR1-PRIVATE, key option cannot be omitted when using MAC OS.” |
| | When the following key types are selected on the [KeyType] tab, select "File" on the [KeyData] tab to use a PEM file as input.<br>”brainpoolP256r1-public”、<br>”brainpoolP256r1-private”、<br>”brainpoolP384r1-public”、<br>”brainpoolP384r1-private”、<br>”brainpoolP512r1-public”、<br>”brainpoolP512r1-private”、 | Not available.<br>When the Generate file button is pressed, the following error message is displayed.<br>“Error: For BRAINPOOLPxxxR1-PRIVATE/PUBLIC, PEM file cannot be specified when using MAC OS.” |

## 7.5 Notes of Secure Factory Programming Function

### 7.5.1 Image Size Limitation for Secure Factory Programming Function

The Secure Factory Programming feature encrypts user program images using AES-CCM mode. As a result, the maximum size of user program images that can be encrypted using this feature is limited by the byte length of the nonce (the byte length specified by the **nonce_prg** option).

Table 7-3 Maximum user program image size

| Maximum image size | Byte size length of **nonce_prg** | **mcu** option |
|---|---|---|
| 0xFFFFFF | 12 | **RA8D1_M1_T1, RA8D2_M2_T2_P1, RA4C1, RA4L1, RX74M_N** |
| 0xFFFFFFFF | 11 | **RA6B1_U1** |

In addition, the user program image is divided according to the format defined in the Boot Firmware application note “Encrypted Data Write Command - Data Packet [Encrypted User Data] - DAT - User data and write address/size.” For this reason, the maximum size of the user program image available for use with the Secure Factory Programming feature is slightly smaller than the following size.

If a user program image larger than the maximum size is entered in the Security Key Management Tool, the following error will be displayed.
Error: The user program to be encrypted is too large. The size of the user program depends on the nonce size used during encryption. For details, refer to the SKMT Software Manual.

#### 7.5.1.1 How to create User data and write address/size in Security Key Management Tool

The Security Key Management Tool adds write address/size information to each write unit (minimum 16 bytes) of each area of the code flash, data flash, and OFS areas of the user program image. When an external flash memory area is specified with extarea0/1, write address/size information is added for each write unit set. The maximum size of the user data section is 1024 bytes.

Example: Image to be encrypted Data exists in the external flash memory area 0x90000000 to 0x9000040F
When **extarea0** “0x90000000” “0x9000FFFF” “16” is set
Encrypted Data Write Command - Data Packet [Encrypted User Data] - as data for DAT
The Security Key Management Tool is
Encrypt the first packet DAT: SAD+SIZE+Reserved+(data in 0x90000000-0x900003FF)
Encrypt the second packet DAT: SAD+SIZE+Reserved+(data in 0x90000400-0x9000040F)

## 7.6 Standalone and Plugin versions Notes on changing MCU/MPU and Crypto Engine in the Overview tab

- Detail :

When using the Standalone or Plugin version, [**Key Data**] tab within [**WrapKey**] tab may not display properly if the MCU/MPU and cipher engine are re-selected on [**Overview**] tab.

- Procedure for issue :

1. Under [**Overview**] tab MCU/MPU and security engine, select the desired MCU/MPU and security engine (in our example, RX Family, TSIP)

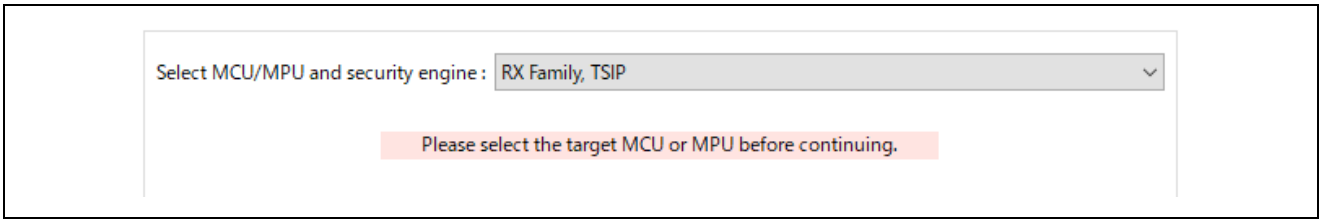


**Figure 7-2 [Overview] Tab, Select RX Family, TSIP for MCU/MPU and Security engine**

2. In [**Wrap Key**] tab, select the type of key to be used in [**Key Type**] tab. (In the example, select RSA 2048bits, public)

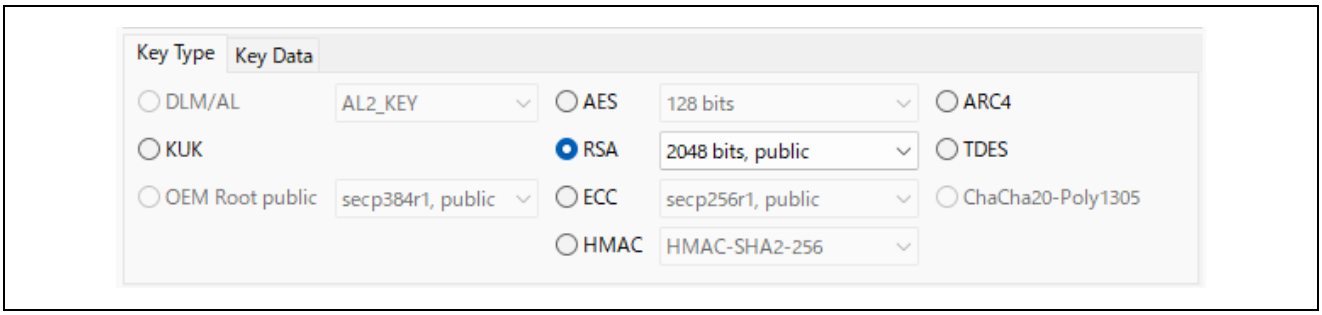


**Figure 7-3 [Wrap Key] Tab, Select RSA 2048bits, public in [Key Type] tab**

3. [**Overview**] tab Reselect MCU/MPU and security engine. (In the example, select RX Family, TSIP Lite)

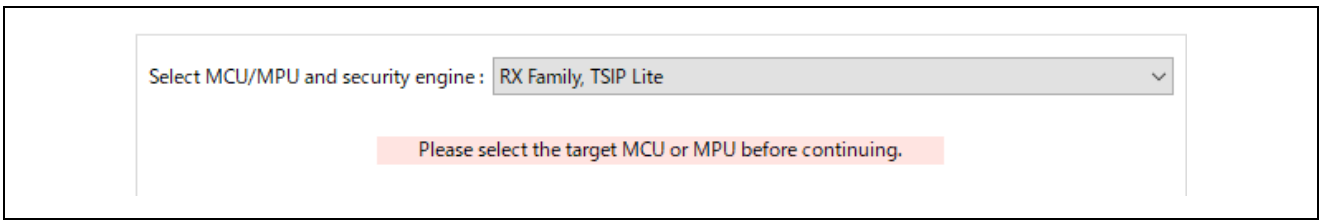


**Figure 7-4 [Overview] Tab, Select RX Family, TSIP Lite for MCU/MPU and Security engine**

4. [**Key Type**] tab in the [**WrapKey**] tab has AES 128bits selected, but the [Key Data] tab displays the input display of the RSA public key.

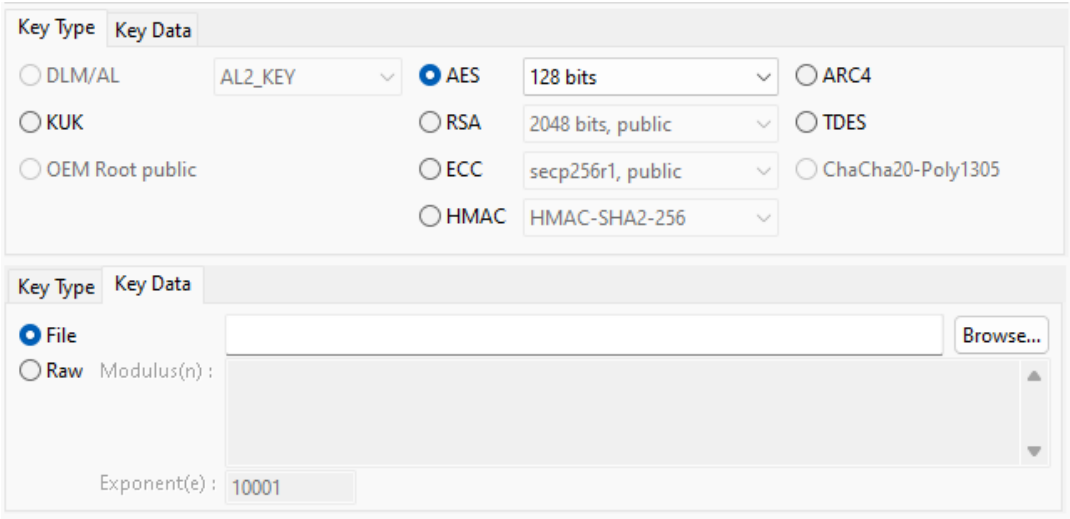


**Figure 7-5 [Wrap Key] Tab, Display AES 128bits in [Key Type] tab and the input display RSA public key in [Key Data] tab**

- Workaround :

If this problem occurs, select a different key type on the [**Key Type**] tab in [**WrapKey**] tab, and [**Key Data**] tab will return to normal. If you want to select AES, select another key type and then select AES again, and the display on [**Key Data**] tab will return to normal.

# 8. Appendix

## 8.1 License

### (a) .NET

This tool uses .NET, which is open-source software provided by .NET Foundation.
See below for .NET license.

.NET License :
https://github.com/dotnet/runtime/blob/main/LICENSE.TXT
https://github.com/dotnet/blazor/blob/master/LICENSE.txt

.NET Foundation :
https://dotnetfoundation.org/

### (b) Inno Setup

The installer for the windows version of this tool was created using Inno Setup.
See below for Inno Setup license.

Inno Setup License :
https://jrsoftware.org/files/is/license.txt

Inno Setup :
https://jrsoftware.org/isinfo.php

### (c) NSec

The Ed25519 key generation function of this software uses the NSec library.
Please refer to the following for NSec license.

NSec License :
https://nsec.rocks/license

NSec :
https://nsec.rocks/

### (d) Bouncy Castle

The macOS version of this software uses Bouncy Castle in the encsfp command.

For the license of Bouncy Castle, please refer to the following.

Bouncy Castle License :

https://www.bouncycastle.org/about/license

Bouncy Castle :

https://www.bouncycastle.org/

## 8.2 Renesas Key File Format

### 8.2.1 Text format

Table 8-1 Renesas Key File Format : Text format

| Items | Specification |
|---|---|
| Character | ASCII code only<br>(Unicode and multi-byte characters cannot be used.) |
| Space Chapter | TAB(0x09) / Space(0x20) |
| Maximum length of a line | 80 bytes |
| Line break | CRLF(0x0D,0x0A) / CR(0x0D) / LF(0x0A) |
| Base64 | Chars = Alpha / Digit / "+" / "/"<br>Pad = "="<br>Line length = 64 chars (except the final line) |

### 8.2.2 File Structure

The file structure is constructed with the following three items.

```
<Header>
<Base64Lines>
<Footer>
```

<Header> and <Footer> are the following fixed strings.

Header = "-----BEGIN RENESAS KEY-----"

Footer = "-----END RENESAS KEY-----"

<Base64Lines> comprises multi-line strings that represent the key data structure, described below. The key data structure is Base64 encoded binary.

Lines after the footer, blank lines, and white space at the end of lines are ignored.

### 8.2.3 File Extension

`".rkey"`

## 8.2.4 Key Data

### 8.2.4.1 Structure

Key Data is stored in the following order and size. The byte order is big-endian.

Table 8-2 Renesas Key File Format : Key data structure

| Name | Type | Size | Description |
|---|---|---|---|
| Magic code | Char[4] | 4 Bytes | "REK1" |
| Suite Version | Integer | 4 Bytes | Data format version.<br>Currently Must be 1. |
| Reserved | Byte[7] | 7 Bytes | 0. |
| Key Type | Byte | 1 Bytes | [For DLM key]<br>0.<br>[For user key]<br>Keytype value constant<br>(Refer to 4.5.2 **keytype** Options) |
| Encrypted Key Size | Integer | 4 Bytes | Size of "Encrypted Key" ( = N bytes) |
| W-UFPK | Byte[36] | 36 Bytes | Value of the W-UFPK file sent from the Renesas DLM server<br>The first 4 bytes are Shared Key Number<br>The remaining 32 bytes are the WUFPK value |
| Initialization Vector | Byte[16] | 16 Bytes | Initialization vector value used to Wrap user key. |
| Encrypted Key | Byte[N] | N Bytes | User key encrypted with UFPK value + MAC value |
| Data CRC | Byte[4] | 4 Bytes | Calculated CRC for all data except this CRC data.<br>Initial Value = 0xFFFFFFFF<br>Magic number = 0x04C11DB7 |

### 8.2.5 Encrypted Key Wrapping Formula

The following formula is used for encryption and MAC generation when wrapping a user key with UFPK or KUK.

- Encrypted Key Wrapping Formula for RA Family, RX Family, and Synergy Platform

```
uint32_t i = 0;
uint32_t n            = User key byte size;
uint8_t IV[16];                                  // Initial Vector(128bit)
uint8_t CBCKey[16]     = UFPK[0:15] or KUK[0:15];   // CBCKEY in either UFPK or KUK
uint8_t CBCMACKey[16] = UFPK[16:31] or KUK[16:31]; // CBCMACKEY in either UFPK or KUK
uint8_t User_Key[n];                             // Plain text User key
uint8_t MAC[16] = {0};
uint32_t encrypted_key[n];                       // Encrypted Key

for (i = 0; i < n; i += 16)
{
   MAC[0:15]    =
      AES128-Enc(CBCMACkey[0: 15], xor_16byte(User_Key[i: i+15], MAC[0: 15]));
   encrypted_key[i: i+15] =
      AES128-Enc(CBCkey[0: 15],    xor_16byte(User_Key [i: i+15], IV[0: 15]));
   IV[0: 15]      = encrypted_key [i: i+15];
}
encrypted_key[i: +15] = AES128-Enc(CBCkey[0: 15], xor_16byte(MAC[0: 15], IV[0: 15]));
```

* : AES128-Enc() indicates encryption in AES ECB mode with a key length of 128 bits.
First Argument: Encryption Key, Second Argument: Plaintext Data

- Encrypted Key Wrapping Formula for RZ family

```
uint32_t n            = User key byte size;
uint8_t IV[16];                                  // Initial Vector(128bit)
uint8_t CBCKey[16]     = UFPK[0:15] or KUK[0:15];   // CBCKEY in either UFPK or KUK
uint8_t CMACKey[16]    = UFPK[16:31] or KUK[16:31]; // CMACKEY in either UFPK or KUK
uint8_t User_Key[n];                             // Plain text User key
uint8_t MAC[16] = {0};
uint32_t encrypted_key[n];                       // Encrypted Key

MAC[0:15]           = AES128-CMAC(CMACkey[0: 15], User_Key[0: n-1]); *1
encrypted_key[0: n-1] = AES128-CBC(CBCkey[0: 15],   IV[0: 15], User_Key[0: n-1]); *2
encrypted_key[n: n+15] = MAC[0:15];
```

*1 : AES128-CMAC() indicates encryption in AES CMAC mode with a key length of 128 bits.
First Argument: Encryption Key, Second Argument: Plaintext Data

*2 : AES128-CBC() indicates encryption in AES CBC mode with a key length of 128 bits.
First Argument: Encryption Key, Second Argument: Initial Vector Third Argument: Plaintext Data

## 8.3 Secure Factory Programming File Format

### 8.3.1 File Format

A binary file consists of a “Pre-Data Field” that stores unique information other than the TLV field, a “TLV Length Field” that stores the size of the TLV field, and a “TLV (Type-Length-Value) Field” that stores encrypted data.
The format of the “Pre-Data Field” file differs depending on the MCU/MPU. Refer to Table 8 3 to see which file format is used for which MCU/MPU.
Data is arranged in big-endian order.

Table 8-3 MCU/MPU and Pre-Data Field Correspondence Table

| Format Version | MCU/MPU | Format Image |
|---|---|---|
| V1 | RA8D1_M1_T1, RA8E1_E2 | Figure 8-1 |
| V2 | RA8D2_M2_T2_P1, RA6B1_U1, RA4L1, RA4C1, RX74M_N | Figure 8-2 |

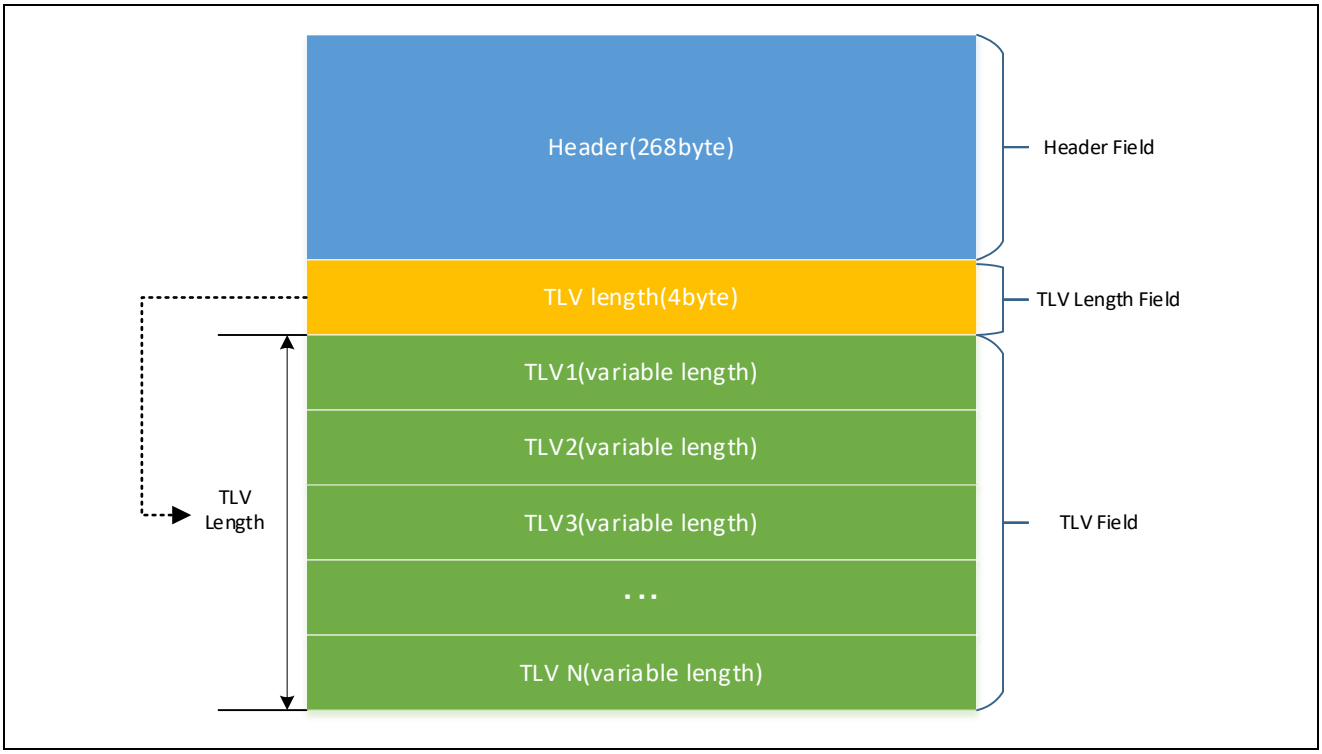


**Figure 8-1 Secure Factory Programming File V1 Format**

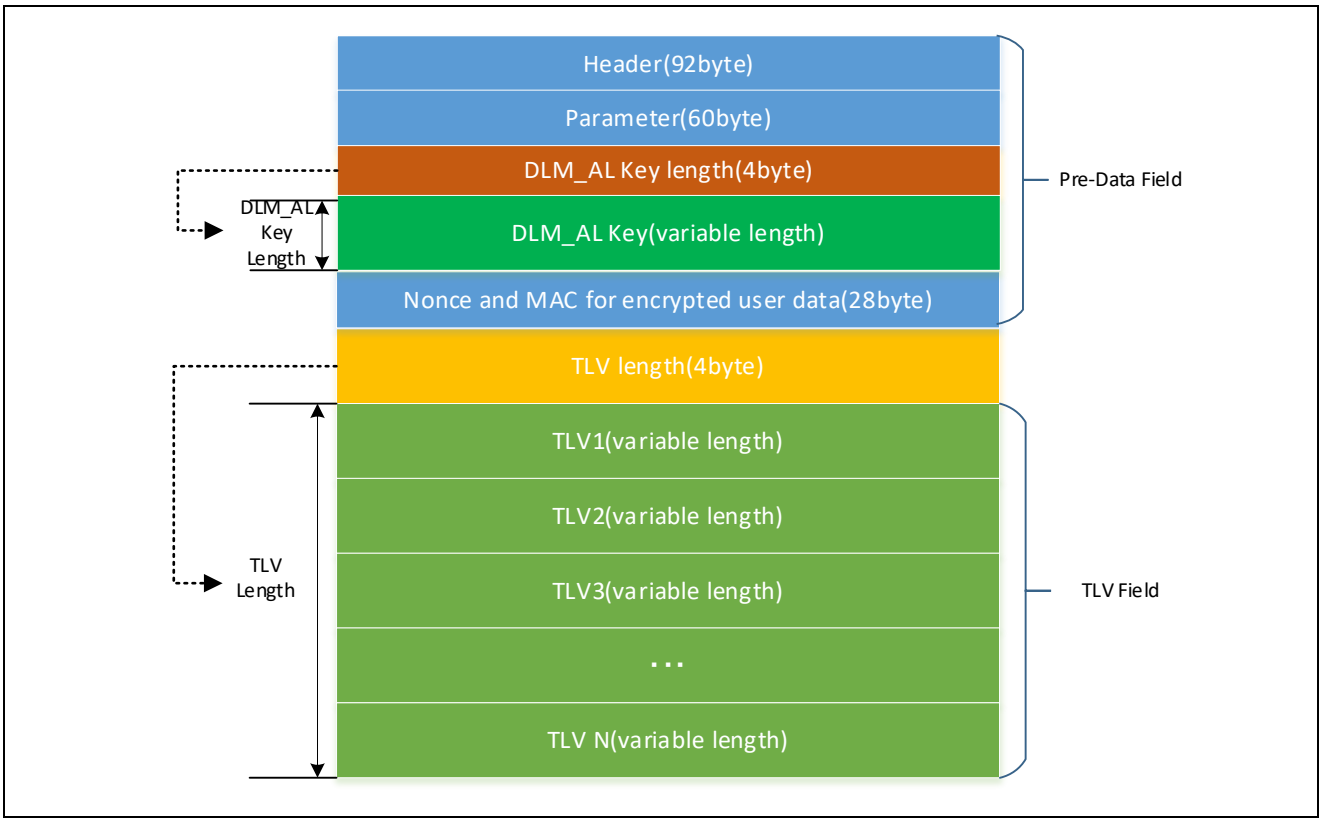


**Figure 8-2 Secure Factory Programming File V2 Format**

#### 8.3.1.1 Pre-Data Field

(1) Pre-Data Field V1 format

The Pre-Data field in V1 format is a 268-byte fixed-length field that stores unique information.

Table 8-4 Secure Factory Programming File V1 Format : Pre-Data field

| Field | Size [byte] | Description |
|---|---|---|
| Magic | 4 | Sets "0x73667072", which is an unsigned 4-byte integer representation of the "sfpr" ASCII code, as the unique magic number. |
| Version | 4 | Format Version<br>The upper 2 bytes represent the major version and the lower 2 bytes the minor version.<br>In V.1.00, "0x00010000" is set. |
| W-UFPK | 36 | Value of the W-UFPK file sent from the Renesas DLM server<br>The first 4 bytes are Shared Key Number<br>The remaining 32 bytes are the WUFPK value |
| Initialization Vector used for encrypting ENKY | 16 | Initialization vector used to wrap the key used to encrypt the user program and parameters |
| ENKY | 32 | UFPK-encrypted data with parameter values and the key used to encrypt the user program |
| Nonce used for encrypting parameters | 12 | Nonce data used to encrypt parameters |
| Encrypted Parameter | 16 | Data with parameter values encrypted with Encryption key |
| MAC for Encrypted Parameter | 16 | MAC value generated when the parameter value is encrypted with Encryption key |
| Reserved | 3 | 0xFF |
| Encrypted AL2 Key Enable | 1 | Initialization Vector for used encrypted AL2 Key field and Encrypted AL2 Key with MAC field enable/disable flag<br>0: Disabled<br>Other: Enabled |
| Initial Vector used for encrypting AL2 Key | 16 | Initialization Vector value used during AL2 Key encryption |
| Encrypted AL2 Key | 32 | AL2 Key encrypted with UFPK + MAC value |
| Reserved | 3 | 0xFF |
| Fixed Value | 1 | 0x00 |
| Reserved | 48 | 0xFF |
| Nonce used for encrypting user data | 12 | Nonce value used to encrypt the user program |
| MAC for encrypted user data | 16 | MAC value generated when encrypting the user program |

### (2) Pre-Data Field V2 format

The Pre-Data field in V2 format is a field that contains the Header, Parameter, DLM_AL key (including DLM_AL key length), Nonce and MAC for encrypted user data.
DLM_AL key is a variable length field.

#### (a) Header Field

The Header field is a 92-byte fixed-length field containing the Magic Number, Version, W-UFPK, IV of ENKY and ENKY data.

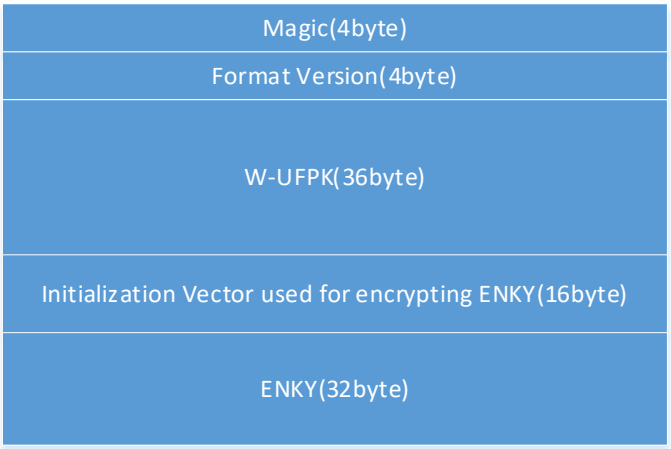


**Figure 8-3 Secure Factory Programming File V2 Format : Header Field**

Table 8-5 Secure Factory Programming File V2 Format : Header field

| Field | Size [byte] | Description |
|---|---|---|
| Magic | 4 | Sets "0x73667072", which is an unsigned 4-byte integer representation of the "sfpr" ASCII code, as the unique magic number. |
| Version | 4 | Format Version<br>The upper 2 bytes represent the major version and the lower 2 bytes the minor version.<br>In V.2.00, "0x00020000" is set. |
| W-UFPK | 36 | Value of the W-UFPK file sent from the Renesas DLM server<br>The first 4 bytes are Shared Key Number<br>The remaining 32 bytes are the WUFPK value |
| Initialization Vector used for encrypting ENKY | 16 | Initialization vector used to wrap the key used to encrypt the user program and parameters |
| ENKY | 32 | UFPK-encrypted data with parameter values and the key used to encrypt the user program |

### (b) Parameter Field

The Parameter field is a 60-byte fixed-length field that contains the encrypted Parameter information, the Nonce and MAC information used during Parameter encryption.

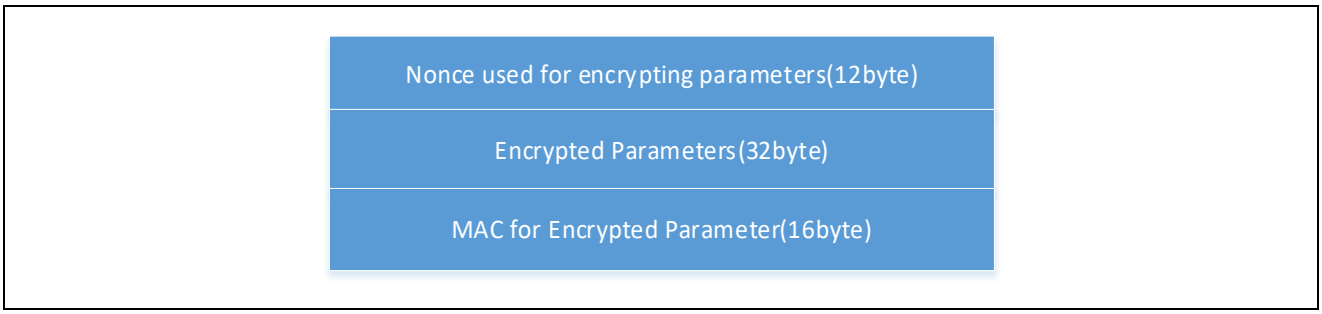


**Figure 8-4 Secure Factory Programming File V2 Format : Parameter Field**

Table 8-6 Secure Factory Programming File V2 Format : Parameter field

| Field | Size [byte] | Description |
|---|---|---|
| Nonce used for encrypting parameter | 12 | Nonce used to encrypt Parameter information<br>If the nonce is 11 bytes long, enter 0xFF in the last byte. |
| Encrypted Parameters | 32 | Parameter information encrypted with Encryption key |
| MAC for Encrypted Parameter | 16 | MAC value generated when Parameter information is Encryption key encrypted. |

### (c) DLM_AL Key Field

The DLM_AL Key field is a variable length field that contains the encrypted data of the wrapped DLM_AL key (Encrypted DLM key data), the Nonce used for encrypting the wrapped DLM_AL key (Nonce used for encrypting DLM key data), and the MAC (MAC for Encrypting DLM key data).
DLM_AL Key length is a fixed length field of 4 bytes that stores the number of bytes in the DLM_AL Key field.

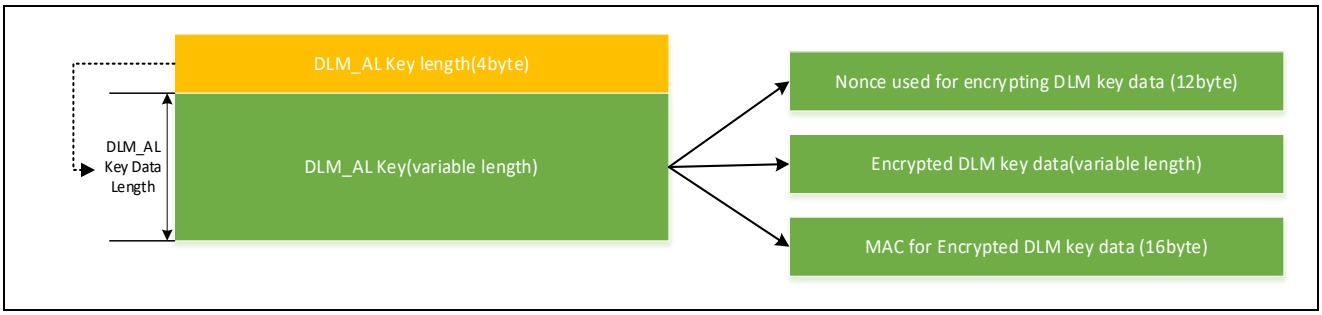


**Figure 8-5 Secure Factory Programming File V2 Format : DLM_AL Key Field**

Table 8-7 Secure Factory Programming File V2 Format : Parameter field

| Field | Size [byte] | Description |
|---|---|---|
| DLM_AL Key length | 4 | DLM_AL Total byte size of Key field |
| Nonce used for encrypting DLM key data | 12 | Nonce used to encrypt the Encrypted DLM_AL key data (Encrypted DLM key data) of the wrapped DLM_AL key<br>If the nonce is 11 bytes long, enter 0xFF in the last byte. |
| Encrypted DLM key data | variable length | Encrypted data of wrapped DLM_AL key |
| MAC for Encrypted DLM key data | 16 | MAC used to encrypt the Encrypted DLM_AL key data (Encrypted DLM key data) of the wrapped DLM_AL key. |

### (3) Nonce and MAC for encrypted user data Field

The Nonce and MAC for encrypted user data field is a 28-byte fixed-length field that stores the nonce used during encryption of the user program and the generated MAC information.

**Figure 8-6 Secure Factory Programming File V2 Format : Nonce and MAC for encrypted user data Field**

Table 8-8 Secure Factory Programming File V2 Format : Nonce and MAC for encrypted user data field

| Field | Size [byte] | Description |
|---|---|---|
| Nonce used for encrypting user data | 12 | Nonce value used to encrypt the user program<br>If the nonce is 11 bytes long, enter 0xFF in the last byte. |
| MAC for Encrypted user data | 16 | MAC value generated when encrypting the user program |

### 8.3.1.2 TLV Length Field

TLV Length is a 4-byte fixed-length field that describes the size of the TLV field.

Table 8-9 Secure Factory Programming File Format : TLV Length field

| Field | Size [byte] | Description |
|---|---|---|
| TLV Length | 4 | Total byte size of TLV field |

### 8.3.1.3 TLV Field

TLV is a variable-length field that is set to one of the following values in Type-Length-Value format.

- Encrypted User Program

The format of TLV is shown in Figure 8-7 TLV Format.

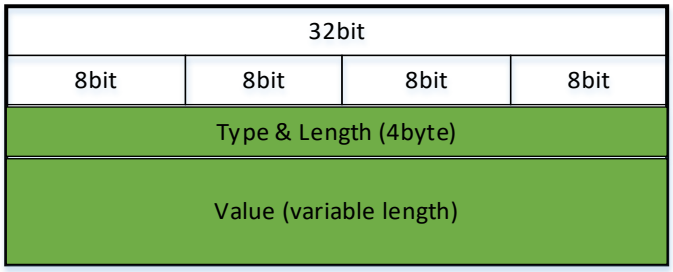


**Figure 8-7 TLV Format**

Table 8-10 Secure Factory Programming File Format : TLV field

| Field | Size [byte] | Description |
|---|---|---|
| Type & Length | 4 | Value type and word (32bit) size |
| Value | variable length | Data per type<br>Size is 4 times the value specified by Length<br>(Length is for word size specification) |

### (1) Detail for Type&Length Field

Type&Length field is defined as a 32-bit bit field.

Table 8-11 Secure Factory Programming File Format : TLV Type&Length field

| Field | Bit Field | Size(bit) | Bit position | Description |
|---|---|---|---|---|
| Type | Class | 4 | 31:28 | Value Classification<br>bit 31 28<br>0 0 0 0 : Encrypted User Data<br>*Others are reserved |
| | <Class unique> | 4 | 27:24 | Fields for each Class |
| Length | word size | 24 | 23:0 | The word size of Value (1 word = 4 byte)<br>[value]0～16,777,215 |

### (2) Unique field for each Class

This section describes the value of the Class unique field for each Class.

If the Class field is "Encrypted User Data”

Table 8-12 Secure Factory Programming File Format : The Class field is “Encrypted User Data”

| Bit Field | Size (bit) | Bit position | Description |
|---|---|---|---|
| Use type | 4 | 27:24 | Usage Type<br>[value]<br>bit 27 24<br>0 0 0 0 : Encrypted User Data<br>*Others are reserved |

## 8.3.2 File extensions

".sfp"

## 8.4 Key Certificate / Code Certificate

### 8.4.1 Key Certificate File Format

This is a binary file with the following data structure.
For details, please refer to the Hardware Manual of the device that supports the FSBL function.

Table 8-13 Key Certificate data format

| Field | | Size [byte] | Description |
|---|---|---|---|
| Header | Magic | 4 | 0x6B657963 |
| | Manifest Version | 4 | 0x00010000 |
| | Flags | 4 | Reserved (0x00000000) |
| | Reserved | 20 | Reserved (ALL 0) |
| TLV Length | | 4 | Data byte length of the TLV field in the Key Certificate 0x000000AC |
| TLV ECC PUBKEY | Type&Length | 4 | 0x00088010 |
| | Value | 64 | OEM ROOT public key specified by /gencert /oemroot_public option |
| TLV KEYHASH | Type&Length | 4 | 0x10144008 |
| | Value | 32 | SHA2-256 HASH value of the OEM BL public key specified by the /gencert /oembl_public option |
| TLV EXPECTED_SIG | Type&Length | 4 | 0x20088410 |
| | Value | 64 | Signature |

### 8.4.2 Code Certificate file format generated by **mode "signature"**

This is a binary file with the following data structure.
For details, please refer to the Hardware Manual of the device that supports the FSBL function.

Table 8-14 Code Certificate data format generated by **mode** “**signature**”

| Field | | Size [byte] | Description |
|---|---|---|---|
| Header | Magic | 4 | 0x636F6463 |
| | Manifest Version | 4 | 0x00010000 |
| | Flags | 4 | 0x00000000 |
| | Load Addr | 4 | OEM BL starting address specified by /**gencert** /**loadaddr** option |
| | Dest Addr | 4 | OEM BL starting address specified by /**gencert** /**loadaddr** option |
| | Image size | 4 | OEM BL size specified by /**gencert** /**oembl_size** option OEM BL size calculated from OEM BL file entered by /**cfsize** option and /**oembl** when /**oembl_size** is omitted |
| | Image version | 4 | Version information specified by /**gencert** /**ver** option |
| | Build number | 4 | Reserved (ALL 0) |
| TLV Length | | 4 | Data byte length of TLV field in Code Certificate 0x000000AC |
| TLV ECC PUBKEY | Type&Length | 4 | 0x01088010 |
| | Value | 64 | OEM ROOT public key |
| TLV EXPECTED_CRC | Type&Length | 4 | 0x40000001 |
| | Value | 4 | CRC32 value of OEM BL |
| TLV SIGNER_ID | Type&Length | 4 | 0x10144008 |
| | Value | 32 | SHA2-256 HASH value of the OEM BL public key |
| TLV EXPECTED_SIG | Type&Length | 4 | 0x25088410 |
| | Value | 64 | Signature |

### 8.4.3 Code Certificate file format generated by **mode "crc"**

This is a binary file with the following data structure.
For details, please refer to the Hardware Manual of the device that supports the FSBL function.

Table 8-15 Code Certificate data format generated by **mode** "**crc**"

| Field | | Size [byte] | Description |
|---|---|---|---|
| Header | Magic | 4 | 0x636F6463 |
| | Manifest Version | 4 | 0x00010000 |
| | Flags | 4 | 0x00000000 |
| | Load Addr | 4 | OEM BL starting address specified by /**gencert** /**loadaddr** option |
| | Dest Addr | 4 | OEM BL starting address specified by /**gencert** /**loadaddr** option |
| | Image size | 4 | OEM BL size specified by /**gencert** /**oembl_size** option OEM BL size calculated from OEM BL file entered by /**cfsize** option and /**oembl** when /**oembl_size** is omitted |
| | Image version | 4 | 0 |
| | Build number | 4 | Reserved (ALL 0) |
| TLV Length | | 4 | Data byte length of TLV field in Code Certificate 0x000000AC |
| TLV EXPECTED_CRC | Type&Length | 4 | 0x40000001 |
| | Value | 4 | CRC32 value of OEM BL |
| TLV SIGNER_ID | Type&Length | 4 | 0x10144008 |
| | Value | 32 | FILL with CRC32 value of OEM BL. |

### 8.4.4 CRC32 calculation

The formula for calculating TLV EXPECTED_CRC in Code Certificate is as follows.

- CRC32 (x32 + x26 + x23 + x22 + x16 + x12 + x11 + x10 + x8 + x7 + x5 + x4 + x2 + x + 1)

Polynomial : 0xEDB88320(Bit reversed)

Shift direction : right shift

Input bit invert : No

output bit inversion: Yes

Initial Value : 0xFFFFFFFF

## 8.5 TSIP Update

### 8.5.1 User program encryption formula

```
uint32_t i = 0;
uint32_t n            = User program byte size;
uint8_t IV[16]        = IV[0:15];                              // Initial Vector(128bit)
uint8_t IEkey[16]     = Image Encryption Key[0:15];
uint8_t IEMACKey[16] = Image Encryption Key[16:31];
uint8_t KEKey[16]     = Key Encryption Key[0:15];
uint8_t User_Program[n];
uint8_t MAC[16] = {0};
uint32_t encrypted_user_program[n];                           // Encrypted user program

for (i = 0; i < (n-16); i += 16)
{
   MAC[0:15]    =
      AES128-Enc(IEMACkey[0: 15], xor_16byte(User_Program[i: i+15], MAC[0: 15]));
   encrypted_user_program[i: i+15] =
      AES128-Enc(IEkey[0: 15],    xor_16byte(User_Program[i: i+15], IV[0: 15]));
   IV[0: 15]    = encrypted_user_program[i: i+15];
}
encrypted_user_program[i: i+15]
                  = AES128-Enc(IEkey[0: 15], xor_16byte(MAC[0: 15], IV[0: 15]));
SessionKey0[0:15] = AES128-Enc(KEkey[0: 15], IEKey[0: 15]);
SessionKey1[0:15] = AES128-Enc(KEkey[0: 15], IEMACKey[0: 15]);
```

* : AES128-Enc() indicates encryption in AES ECB mode with a key length of 128 bits.
First Argument: Encryption Key, Second Argument: Plaintext Data

## 8.6 RA6B1/RA6U1 Header format

For RA6B1/RA6U1, using the RA6B1/RA6U1 tab or the **genhdr** command, you can generate a Product Header, Image Header, Key Certificate, and Code Certificate by combining **mode** and **header** options.

The combinations for generating each header and certificate are as follows:

Table 8-16 **mode**, **header** options, and the generated headers and certificates

| Options | | Header | | Certificate | |
|---|---|---|---|---|---|
| **mode** | **header** | Product Header | Image Header | Key Certificate | Code Certificate |
| **provisioning** | **sec_sign** | - | ✓ | ✓ | ✓ |
| | **sec** | - | ✓ | - | - |
| **app** | **sec_sign** | ✓ | ✓ | ✓ | ✓ |
| | **sec** | ✓ | ✓ | - | - |

✓ : Generate - : Not generate

### 8.6.1 Product Header

Table 8-17 Product Header

| | Field | Value | Size(byte) |
|---|---|---|---|
| Product header | Identifier | “prod” | 4 |
| | Active image pointer | Start address of Image Header | 4 |
| | Update image pointer | Start address of Image Header | 4 |
| | CRC | CRC-16-CCITT value from “Identifier” to “Update image pointer”; the default value is 0x1021 | 2 |
| | Reserved | 0 | 18 |
| Product header (backup) | Identifier | Same value as Product Header | 4 |
| | Active image pointer | Same value as Product Header | 4 |
| | Update image pointer | Same value as Product Header | 4 |
| | CRC | Same value as Product Header | 2 |
| | Reserved | 0 | 18 |

### 8.6.2 Image Header

Table 8-18 Image Header

| Field | Value | Size(byte) |
|---|---|---|
| Image Header Identifier | “imag” | 4 |
| Version | 0x00000000 | 4 |
| ctmr reserved | An area where users can enter any data<br>Enter the value specified with **ctmr** option | 16 |
| IVT offset | The offset size from the Image Header to the IVT (the beginning of the input file for Provisioning SW, or the address specified in the IVT for Application) | 4 |

### 8.6.3 Key Certificate

Table 8-19 Key Certificate

| Field | Value | Size(byte) |
|---|---|---|
| Key Certification Identifier | “kcer” | 4 |
| Signature | Using the OEM Root Key, the signature value from the Key Cert Version of the key certificate to the OEM RoT Key index<br>If secp256r1 is specified for the OEM RoT key, the bytes starting from the 64th byte are padded with zeros | 96 |
| Key Cert version | 00.00.01.xx is set. The value of xx depends on the ECC curve used to sign the key certificate.<br>secp384r1: 00<br>secp256r1: 01 | 4 |
| Code size value | The byte size of the firmware image signed by the code certificate. | 4 |
| SHA384 of OEM Provisioning SW PK/OEM BL PK | For **mode provisioning**, SHA-384 hash value of the OEM Provisioning SW PK.<br>For **mode app**, SHA-384 hash value of the OEM BL PK. | 48 |
| OEM RoT Key Index | The OEM RoT Key number used for verification | 4 |

### 8.6.4 Code Certificate

Table 8-20 Code Certificate

| Field | Value | Size(byte) |
|---|---|---|
| Code Certification Identifier | “ccer” | 4 |
| Signature | The signature value from the Code Cert Version field to the end of the code certificate, plus the signature value from the IVT field to the Code Size field in the application firmware<br>If secp256r1 is specified in the OEM Provisioning SW or OEM BL PK, the bytes starting from the 64th byte are padded with zeros. | 96 |
| Code Cert version | 00.00.01.xx is set. The value of xx depends on the ECC curve used to sign the code certificate.<br>secp384r1: 00<br>secp256r1: 01 | 4 |
| OEM Provisioning SW / OEM BL PK | For **mode provisioning**, the public key of the OEM PROV Key<br>For **mode app**, the public key of the OEM BL Key | 96 |
| FW version | Application Firmware Version<br>This field is not present in **mode provisioning**. | 4 |

| Revision History | Security Key Management Tool Software Manual |
|---|---|

| Rev. | Date | Description | |
|---|---|---|---|
| | | Page | Summary |
| 1.00 | Dec.28.21 | — | First Edition issued |
| 1.01 | Mar.31.22 | — | 3 GUI function description added<br>5 GUI description added<br>6 GUI operation method added |
| 1.02 | Jun.30.22 | — | 4.5.1 mcu option RA-SCE5_B added<br>4.5.3.2 File input pem file added<br>5.2.1 Usage when using Linux version GUI Change |
| 1.03 | Dec.29.22 | — | 4.7 e$^2$studio plugin version description added<br>Appendix Renesas Key File Format added |
| 1.04 | Sep.29.23 | — | 3.7 [TSIP UPDATE] tab added<br>4.5.1 mcu option RE-TSIPLite deleted<br>4.6 ecntsip command option added<br>4.7 calcresponse command option added |
| 1.05 | Dec.22.23 | — | 1.5 Renesas Security Functions added<br>3.8 [FSBL] tab added<br>3.9 [DOTF] tab added<br>3.10 [SFP] tab added<br>4.7 gencert command option added<br>4.8 encdotf command option added<br>4.9 encsfp command option added<br>6.5 Generate FSBL Key Certificate/Code Certificate added<br>6.6 Using Decryption On-The-Fly with RA Family added<br>6.7 RA Family using Secure Factory Programming added |
| 1.06 | Mar.29.24 | P.26<br>P.49<br>P.84<br>P.94<br><br><br>P.98<br><br><br><br>P.101 | 2.2.2 Update e2studio version 2024-01<br>3.7.6 RSU header ver 2 can be specified change<br>4.5.3.2 Added RSA-2048-public-TLS to pem file support asymmetric keys<br>4.6 ver option 2 added<br>Changed to 2 when omitted<br>endaddr option The upper limit of the range to be encrypted is 8MB added<br>4.7.2 Sequence Number in ver option RSU header V1 is always 1 Corrected<br>Execution Address fixed value 0xFFFEFFFC appended<br>RSU header V2 added<br>4.7 ver option 1 to 4,294,967,295 can be specified added |

| Rev. | Date | Description | |
|---|---|---|---|
| | | Page | Summary |
| 1.07 | Aug.30.24 | P.6 | Terminology Encrypted KUK Add |
| | | P.12 | 1.2 Table 1-1 Updated |
| | | P.25 | 2.1 Table 2-1 and Table 2-2<br>support RX RSIP-E11A RZ/T2M,RZ/T2ME,RZ/T2L, RZ/N2L |
| | | P.27 | 2.2.1 OSes supported Ubuntu 22.04 and macOS added<br>Update e2studio version 2024-07 |
| | | P.39 | 3.6.2.1 Table 3-6 Note regarding macOS version added |
| | | P.46 | 3.6.5 Table 3-8 Addition |
| | | P.48 | 3.7 TSIP UPDATE -> TSIP Update tab name change |
| | | P.61 | 3.9 DOTF -> DOTF/OTFD tab name change |
| | | P.69 | 3.10 Final DLM/AL State “OEM PL0 with AL2_KEY and AL1_KEY” option deleted<br>Notes added |
| | | P.72 | 3.10.5 Note added |
| | | P.78 | 4.5.1 Table 4-8 update support RX RSIP-E11A,RZ/T2M,RZ/T2ME,RZ/T2L, RZ/N2L |
| | | P.86 | 4.5.3.1 Table 4-44 Qx → Qy Correction |
| | | P.87 | 4.5.3.2 Table 4-53,4-54 Note regarding macOS version added |
| | | P.96 | 4.5.5 fileadd Options added |
| | | P.96 | 4.5.6 bswap Options added |
| | | P.115 | 4.9 al1_key option removed |
| | | P.118 | 4.9.2 trn “OEM_PL_AL2_1” option deleted |
| | | P.125<br>-126 | 5.1.3 macOS version added |
| | | P.171<br>-172 | 7.3 macOS version Notes on using e2studio plugin added<br>7.4 macOS version Notes added |
| | | P.175 | 8 C Renesas Key File Format 5) Encrypted Key Wrapping Formula added |
| | | P.183 | F TSIP Update 1) User program encryption formula added |
| 1.08 | Feb.19 25 | P.4 | RSIP-Exxx added to description |
| | | P.25,26 | Table2-1,2-2 added RSIP-E11A PM, CM |
| | | P.27 | 2.2.2 Updated Software Environment |
| | | P.39 | 3.6.2.1 Table 3-6 Added Ed25519 |
| | | P.48 | 3.7 Updated Figure3.16 |
| | | P.50 | 3.7.3 Updated Figure3.19 |
| | | P.51 | 3.7.4 Updated Figure3.20<br>Added Flash Write Size, Data Flash |
| | | P.66<br>-68 | 3.10 Updated Figure3.35<br>Added Table 3-13<br>Table 3-14 Added DPL with SECDBG_KEY and NONSECDBG_KEY |
| | | P.71 | 3.10.3 Updated Figure3.38<br>Added IV(AL/DLM key) |
| | | P.72 | 3.10.4 Change AL2_KEY -> AL2/SECDBG_KEY tab |
| | | P.73 | 3.10.5 Change AL1_KEY -> AL1/NONSECDBG_KEY tab |
| | | P.74 | Added 3.10.6 Boundary tab |
| | | P.75 | Added 3.10.7 External Flash Memory Area tab |
| | | P.77 | 4.2 Table4-3 Added Renesa Partition Data File |
| | | P.82 | 4.5.1 Table 4-8 Added RSIP-E11A PM, CM |
| | | P.91 | 4.5.3.2 Table 4-54 Added Ed25519 |
| | | P.93 | 4.5.3.3 Table 4-55 Added Ed25519 |
| | | P.101 | 4.5.7 Table 4-73 Added Ed25519 |
| | | P.102 | 4.6 Table 4-74 Added flash_wsize and df_ena optionsUpdated 4.6.1 ver option |
| | | P.107 | 4.6.1 Table 4-76 Delete “only one in this version” |

| | | | |
|---|---|---|---|
| | | P.110<br>P.119<br>-121<br><br><br><br>P.123<br><br><br><br><br>P.124<br>P.125<br>P.157<br>-158<br>P.170<br>-175<br><br><br><br>P.177,<br>178<br>P.189,<br>190<br>P.181<br>-197 | Added 4.6.5 df_ena option<br>4.9 Deleted Note for macOS version<br>Table 4-89 Added nonce_key, secdbgkey and nonsecdbgkey, options<br>Table 4-90 Added boundary iv_secdbgkey, iv_nonsecdbgkey, output_secdbgkey, and output_nonsecdbgkey option<br>4.9.1 Change RA8x1 to RA8D1_M1_T1<br>4.9.2 Table 4-92 Added RA4L1<br>Table 4-93 Added DPL_SECDBG_NONSECDBG<br>Added Table 4-94<br>Added 4.9.4 boundary option<br>Added 4.9.5 extarea0/extarea1 options<br>6.4.1 Update from Figure 6-26 to 6-29<br><br>6.7.1 Update from Figure 6-45 to 6-48<br>Added MCU/MPU selection explanation<br>Added description of IV (AL/DLM key)<br>Updated tab name<br>7.4 Delete limitation of encsfp<br><br>Added 7.5 Limitations of Secure Factory Programming Function.<br>Added 7.6 GUI Version [SFP] tab Limitations when setting files<br>8 Modified chapter structure.<br>8.1 Added NSec and Bouncy Castle<br>8.3 Added Format Version 2 |
| 1.09 | Jul.1 25 | P,25,26<br><br><br><br><br><br><br>P.27<br><br><br><br>P.34<br>P.37<br><br><br><br>P.57<br>P.67<br>P.68<br>P.74<br>P.82<br>P.84<br><br>P.85<br>P.89<br><br><br><br><br><br>P.90<br>P.99 | 2.1 Table 2-1, 2-2<br>Added RA Family RSIP-E50D Security Function and Protected Mode、<br>RSIP-E50D Compatibility Mode<br>RX Family RSIP-E11A Security Function and Protected Mode、<br>RSIP-E11A Compatibility Mode<br>Delete RX Family RSIP-E11A<br>2.2.2 OSes supported<br>Added Windows11 and Ubuntu 24.04<br>Deleted Ubuntu 20.04<br>Updated e2sudio 2025-01 to e2studio 2025-04<br>3.6 Added ChaCha20-Poly1305<br>3.6.1 Table 3-5<br>Change SHAxxx-HMAC to HMAC-SHA-1 or HMAC-SHA2-xxxx<br>Added HMAC- SHA3-224, HMAC- SHA3-256, HMAC- SHA3-384, HMAC- SHA3-512<br>3.8.2 Added Output Measurement Report<br>3.10 Table 3-13 Added RA8D2/M2/T2/P1<br>Table 3-14 Added OEM PL0 with AL2_KEY and AL1_KEY<br>3.10.6 Delete Set Option<br>4.5.1 Table 4-8 Added RA-RSIP-E50D, RA-RSIP-E50D-CM, RX-RSIP-E11A-CM<br>4.5.2 Table 4-13 Added SHA3-HMAC-224, SHA3-HMAC-256, SHA3-HMAC-384, SHA3-HMAC-512<br>Table 4-16 Added ChaCha20-Poly1305<br>4.5.3.1 added<br>Table 4-47 Added HMAC-SHA3-224<br>Table 4-48 Added HMAC-SHA3-256<br>Table 4-49 Added HMAC-SHA3-384<br>Table 4-50 Added HMAC-SHA3-512<br>Added Table 4-53 ChaCha20-Poly1305<br>4.5.4 Table 4-69 Added<br>HMAC-SHA3-224, HMAC-SHA3-256, HMAC-SHA3-384, HMAC-SHA3-512 |

| | | | |
|---|---|---|---|
| | | | Added Table 4-72 ChaCha20-Poly1305 |
| | | P.103 | 4.6 Table 4-78 Added motoffset option |
| | | P.106 | 4.6.1.2 Added description of motoffset option |
| | | P.112 | 4.7 Table 4-86 Added measurementreport option |
| | | P.123 | 4.9.1 Table 4-96 Added RA8D2_M2_T2_P1 |
| | | | 4.9.2 Table 4-97,Table4-98 AddedOEM_PL0_AL2_1 |
| | | | Changed Chapter 7.6 |
| | | P.184 | 8.3.1 Table 8-3 Added RA8D2_M2_T2_P1 |
| 1.10 | Aug.06 25 | P,25,26 | 2.1 Table 2-1, 2-2<br>Added RA Family RSIP-E31A Security Function and Protected Mode、<br>RSIP-E31A Compatibility Mode |
| | | P.67 | 3.10 Table 3-13 Added RA4C1 |
| | | P.82 | 4.5.1 Table 4-8 Added RA-RSIP-E31A, RA-RSIP-E31A-CM |
| | | P.123 | 4.9.1 Table 4-96, Table 4-98 Added RA4C1 |
| | | P.179 | 7.5 Changed chapter structure |
| | | P.180 | 7.5.2 Added Notes of Secure Factory Programming function. |
| | | P.189 | 8.3.1 Table 8-3 Added RA4C1 |
| | | P.191 | 8.3.1.1 Table 8-4 Changed 192 byte 0xFF to 0x00 Fixed Value |
| 1.11 | Nov.28 25 | P.27 | 2.2.2 Deleted Windows10 from Support OS<br>Updated e$^2$studio plugin version |
| | | P.30 | 3.2.1 Added [Log Start…] [Log End]<br>3.2.2 Updated [Status Clear] description.<br>Added [Display Command Line] |
| | | P.31 | 3.2.4 Added Status area |
| | | P.58 | 3.8.3 Figure 3.24 Updated: Key Certificate and Code Certificate with Output Frame Added |
| | | - | Delete 7.5.2 |
| 1.12 | Sep.30 26 | | Change this manual title form User's Manual to Software Manual |
| | | P.25-28 | 2.1 Table 2-1、Table 2-2<br>Added RA Family RSIP-E31B Security Function and Protected Mode、<br>RSIP-E31B Compatibility Mode<br>RX Family RSIP-E50D Security Function and Protected Mode、<br>RSIP-E50D Compatibility Mode<br>RZ Family RZ/T2N |
| | | P.39 | 3.6.1 Added 256bits, DOTF to Table 3-2 |
| | | P.40 | Added Table 3-6 |
| | | P.55 | 3.7.5 Supported Key Encryption Key file input |
| | | P.57,58 | 3.8 Added OEM Bootloader Endian Configuration Option |
| | | P.67 | 3.9.3 Updated Key length specification Description |
| | | P.67 | 3.9.4 Updated No(2) Description<br>Added Output IV file |
| | | P.69,70 | 3.10 Added Endian Configuration Option |
| | | P.71 | Added RA6B1/RA6U1, RA8E1/E2 and RX74M/N to Table 3-14 |
| | | P.79 | 3.11 Added [RA6B1/RA6U1] tab |
| | | P.88 | 4.2 Added genhdr to Table 4-2 |
| | | P.93 | 4.5.1 Added RX-RSIP-E50D, RX-RSIP-E50D-CM,<br>Added RA RSIP-E31B and RA RSIP-E31B-CM<br>Added RZ/T2N to Table 4-8 |
| | | P.94 | 4.5.2 Added AES-256-DOTF to Table 4-10 |
| | | P.96 | 4.5.2 Added OEM_ROOT_PK_SECP256R1_RA_WIRELESS and<br>OEM_ROOT_PK_SECP384R1_RA_WIRELESS to Table 4-17 |
| | | P.96 | 4.5.2.1 Added AES-256-DOTF Option |

| | | | |
|---|---|---|---|
| | | P.99 | 4.5.3 Added OEM_ROOT_PK_SECP256R1_RA_WIRELESS to Table 4-38<br>Added OEM_ROOT_PK_SECP384R1_RA_WIRELESS to Table 4-40 |
| | | P.102 | 4.5.3.2 Table 4-56 Added *.bin in Extension |
| | | P.102 | 4.5.3.2 Table 4-57 |
| | | P.104 | 4.5.3.3 Table 4-58 |
| | | P.112 | 4.5.7 Table 4-77<br>Added OEM_ROOT_PK_SECP256R1_RA_WIRELESS<br>and OEM_ROOT_PK_SECP384R1_RA_WIRELESS |
| | | P.113 | 4.6 Supported enckey file input to Table 4-78. |
| | | P.123 | 4.7 Added oembl_endian option to Table 4-87. |
| | | P.124 | Added 4.7.2 oembl_endian option |
| | | P.127 | 4.8 Update nonce Description in Table 4-90<br>Added dotftype and iv_tlv_output to Table 4-90 |
| | | P.129 | 4.8.3 Added 4.8.3 dotftype option |
| | | P.130 | 4.8.4 Added iv_tlv_output option |
| | | P.131 | 4.9 Updated nonce_prm, nonce_prg and nonce_key description in Table 4-97 |
| | | P.133 | Added num_al2key num_al1key and endian options to Table 4-99 |
| | | P.135 | 4.9 Added RA8E1_E2, RA6B1_U1 and RX74M_N to Table 4-100, Table 4-102 and Table 4-103 |
| | | P.136 | Added 4.9.3 nonce_prm, nonce_prg and nonce_key options |
| | | P.136 | Update 4.9.5 description. |
| | | P.139 | Added 4.9.7 num_al2key/num_al1key options |
| | | P.140 | Added 4.10 genhdr Options |
| | | P.197 | Added 6.8 Example 8 – Generate RA for Wireless header |
| | | P.205 | 7.5.1 Updated the size information |
| | | P.212 | 8.2.5 Encrypted Key Wrapping Formula for RZ family<br>AES128-CMAC() The IV argument is unnecessary and has been removed |
| | | P.222 | 8.4.2 Table 8-14 TLV ECC PUBKEY Type & Length Corrected from 0x00088010 0x01088010 |
| | | P.225 | Added 8.6 RA6B1/RA6U1 Header format |

Security Key Management Tool Software Manual

Publication Date: Rev.1.12 Sep.30.26

Published by: Renesas Electronics Corporation

# Security Key Management Tool

Renesas Electronics Corporation

R20UT5349EJ0112