

Renesas RX Family

Injecting and Updating Secure User Keys

Introduction

Cryptography plays a critical role in providing the tools to achieve the essential elements of security: authenticity, confidentiality, and integrity. In modern cryptographic systems, the security of the system no longer depends on the confidentiality of the algorithm in use but rather on the confidentiality of the keys.

Renesas MCU security revolves around integrated security engines. Different types of security engines are used across the RX Family of MCUs. Users can find which specific engine is used in a particular microcontroller from its hardware user's manual.

The security engines run in two different modes, called protected mode and compatibility mode. This application note explains the definition of the two modes based on the following differences in operations.

- In compatibility mode, the security engines can inject not only plaintext keys but keys in a secure format that secures the confidentiality of the keys. Keys must be injected by using the RX Family's Flexible Software Package (FSP) API functions.
- In protected mode, the security engines can only inject keys in a secure format that secures the confidentiality of the keys. Keys must be injected by using the interface of the MCU's boot firmware.

This application note describes the following secure key injection processes.

- RSIP-E50D compatibility mode AES-128 secure key injection
- RSIP-E50D protected mode ECC secp256r1 public key and private key secure key injection and updating

Sample keys are provided with the projects. **These keys must not be used in an end-product**, as key reuse constitutes a major security risk. This application note describes how to modify the projects so they are usable with custom keys.

The following table describes some terms used in this application note.

Table Terminology

Term	Description	Section with Details
RSIP	Renesas Secure IP A security engine mounted in a Renesas microcontroller. The RSIP-E50D is mounted in the RX74M.	1.4
HUK	Hardware Unique Key A device-unique encryption key that is stored in the device and used to protect keys.	1.4
HRK	Hardware Root Key A key for wrapping the UFPK, which is stored only in the RSIP and on the Renesas Key Wrap server.	1.4
UFPK	User Factory Programming Key A set of keys that is set by the user and used for encrypting the user keys in key injection. This is not used in the device.	2.2
W-UFPK	Wrapped UFPK Key information generated by wrapping the UFPK with the HRK. This is decrypted into the UFPK by the HRK in the RSIP and used in that form.	2.2
KUK	Key Update Key A key that is set by the user and used for encrypting the user keys in key updating. To update keys in the device, a wrapped KUK needs to be generated through key injection in advance.	2.2
MRAM	Magneto-resistive Random Access Memory This is used as non-volatile RAM (NVRAM) instead of flash memory in process nodes above a certain level. Code MRAM and data MRAM are mounted in the RX74M.	5.3
RFP	Renesas Flash Programmer A tool for writing programs to NVRAM of Renesas microcontrollers.	5.3
SKMT	Security Key Management Tool A tool for supporting secure key injection or updating firmware.	4.2, 5.1, 5.2, and 6.1

Required Resources

Target Device

RX74M (with RSIP-E50D)

Development Tools and Software

- e² studio IDE v2026-07
- Renesas Flexible Software Package (FSP) v1.0.0
- Renesas Flash Programmer (RFP) v3.24
- Renesas Security Key Management Tool (SKMT) v1.12

The FSP and e² studio are bundled in a downloadable platform installer available on the FSP webpage (<https://www.renesas.com/rx-fsp>). The RFP can be downloaded from <https://www.renesas.com/en/software-tool/renesas-flash-programmer-programming-gui>. The free-of-charge edition can be used for the functionality required by this application note. The SKMT can be downloaded from <https://www.renesas.com/en/software-tool/security-key-management-tool>.

Hardware

- EK-RX74M, Evaluation Kit for RX74M MCU Group (<https://www.renesas.com/ek-rx74m>)
- PC running Windows® 11 OS
- One USB Type-C® cable Type-C® male to Type-C® male

Prerequisites and Intended Audience

This application note describes a model of development with the use of the e² studio, an integrated development environment (IDE) for Renesas microcontrollers. This development requires some knowledge of the security features of RX Family MCUs. As required, refer to the section “Security Features” in the *Renesas RX74M Group User’s Manual: Hardware* and read the documentation of development tools while proceeding through this application note.

The intended audience consists of product developers, product manufacturers, product support, or end users who are involved with any stage in the injecting and updating of secure keys by Renesas RX Family MCUs.

Contents

1. Protecting User Keys.....	6
1.1 Overview.....	6
1.2 Protecting User Keys and the Root of Trust.....	7
1.3 Overview of the Security Engine and Associated Keys	8
1.4 Advantages of Renesas Secure Key Injection	10
1.4.1 Advantages of Key Wrapping over Key Encryption	10
1.4.2 Advantages of Key Wrapping by Using the HUK.....	11
1.5 Differences between Protected Mode and Compatibility Mode	12
2. Secure Key Injection and Updating	13
2.1 Key Injection and Updating in Protected Mode.....	13
2.1.1 Key Injection	14
2.1.2 Key Updating.....	16
2.2 Key Injection in Compatibility Mode	18
2.3 Tools Used in Secure Key Injection and Updating.....	20
2.3.1 Renesas Key Wrap Service	20
2.3.2 Security Key Management Tool (SKMT).....	20
2.3.3 Renesas Flash Programmer (RFP).....	20
2.3.4 e ² studio or Renesas Flexible Software Package (FSP).....	20
3. Using the Renesas Key Wrap Service	21
3.1 Generating a PGP Key Pair	21
3.2 Registration with the Renesas Key Wrap Server	25
3.3 Exchanging PGP Public Keys of the User and Renesas	26
3.3.1 Registering the User PGP Public Keys	26
3.3.2 Obtaining the Renesas PGP Public Key	28
3.3.3 Importing the Renesas PGP Public Key	29
4. Generating a UFPK and Obtaining a W-UFPK.....	30
4.1 Generating a UFPK	30
4.1.1 Installing the SKMT	30
4.1.2 Generating a UFPK through the GUI	30
4.1.3 Generating a UFPK through the CLI	35
4.2 Obtaining a W-UFPK.....	36
4.2.1 Encrypting the UFPK with PGP.....	36
4.2.2 Obtaining a W-UFPK.....	39
4.2.3 Decrypting the W-UFPK.....	43
5. Secure Key Injection and Updating in RSIP Protected Mode and the Sample Project	44
5.1 Overview.....	44

5.2	Wrapping Keys with the UFPK and W-UFPK through the SKMT GUI.....	45
5.2.1	Wrapping the Initial ECC Key Pair with the UFPK.....	45
5.2.2	Wrapping a KUK with the UFPK	48
5.2.3	Wrapping a New ECC Key Pair with the KUK.....	51
5.3	Wrapping Keys with the UFPK and W-UFPK through the SKMT CLI.....	54
5.3.1	Wrapping the Initial ECC Key Pair with the UFPK.....	55
5.3.2	Generating and Wrapping a KUK with the UFPK.....	57
5.3.3	Wrapping the New ECC Key Pair with the KUK.....	58
5.4	Secure Key Injection through the MCU Boot Firmware Interface	60
5.4.1	Setting up the Hardware.....	60
5.4.2	Injecting the Initial User Keys and KUK	61
5.5	Sample Project for Protected Mode	64
5.5.1	Overview of the Sample Project.....	64
5.5.2	Injecting Keys in the Sample Project.....	65
5.5.2.1	Formatting the Injected Keys	65
5.5.2.2	Verifying the Injected and Updated Keys	65
5.5.3	Executing the Sample Project	66
6.	Secure Key Injection in RSIP Compatibility Mode and the Sample Project	67
6.1	Overview.....	67
6.2	Wrapping a Key with the UFPK and W-UFPK through the SKMT GUI.....	67
6.3	Wrapping the Key with the UFPK and W-UFPK through the SKMT CLI	70
6.4	Sample Project for Compatibility Mode	71
6.4.1	Using the Files Generated through the SKMT	71
6.4.2	Executing the Sample Project	71
7.	References	72
8.	Website and Support	73
	Revision History	74

1. Protecting User Keys

1.1 Overview

As well as the degree of confidentiality of the cryptographic algorithm itself, the security of a cryptographic system is largely dependent on whether encryption keys can be securely generated, stored, used, and updated. In embedded devices in particular, user keys injected at the time of manufacturing being exposed as plaintext keys in the program memory or general-purpose RAM can lead to the leakage of keys or reproduction of the device.

The Renesas Secure IP (RSIP-E50D) mounted in RX74M devices executes cryptographic operations in hardware as well as protecting keys in a way that does not allow direct reference by the CPU, that is, by handling user keys as wrapped keys. Since a wrapped key is protected based on the MCU-unique key information, even if a wrapped key that was generated for use in a specific MCU is copied to another MCU, it cannot be used as it is.

This application note, targeting the RSIP-E50D of the RX74M, explains how to securely inject user keys and update them as required. The RSIP-E50D has two operating modes: protected mode and compatibility mode. The interface used for key injection, the formats of keys that can be handled, and the method of updating keys differ in these two modes. The differences between the modes are explained in section 1.5, Differences between Protected Mode and Compatibility Mode.

In this application note, ECC secp256r1 public key and private key secure key injection and updating in protected mode and AES-128 secure key injection in compatibility mode are given as examples.

Note that the sample keys included in this application note are for use in confirming operation. These keys must not be used in an end-product because key reuse creates a major security risk.

1.2 Protecting User Keys and the Root of Trust

In IoT devices and embedded devices, many types of security functionality, such as confirmation of the device's authenticity, secure boot, firmware updating, and protection of communications data, depend on encryption keys. Leakage, falsification, or duplication of these keys may damage the authenticity of the entire system.

The roots of trust are hardware, firmware, and software components that are the base points of authenticity in the system. In an embedded device, a root of trust consists of a device-unique ID, device key, verification key for secure boot, and key for communications authentication. Therefore, mechanisms for securely injecting and storing these keys and preventing plaintext key exposure at the time of use are required to maintain the root of trust.

Secure boot is a typical functionality constituting the root of trust in many security systems. The application is authenticated by using public key encryption. The verification key used in secure boot and the private key and certificate used to identify the device are important assets for supporting the authenticity of the system.

From the above descriptions on root of trust, it is obvious that leakage of encryption keys can bring a secure system into a risky state. Protection of the root of trust involves limiting key accessibility within the cryptographic boundary only, with keys that are securely stored and preferably unclonable. The root of trust should also be protected from read and write accesses by unauthorized parties.

An RSIP mounted in an RX Family MCU made by Renesas handles user keys as wrapped keys so as to provide a mechanism for using keys without any plaintext key exposure in CPU-accessible areas. In addition, its method of secure key injection and updating supports key protection throughout the product lifecycle from provisioning during production to updating keys in the field.

1.3 Overview of the Security Engine and Associated Keys

This section gives an overview of the RSIP-E50D security engine mounted in RX74M devices and key information related to the protection, injection, and updating of user keys. The RSIP-E50D provides functionality for performing cryptographic operations as well as key protection functionality for handling user keys as wrapped keys.

The security engine (RSIP) is an isolated subsystem within the MCU. The security engine contains hardware accelerators for symmetric and asymmetric cryptographic algorithms, as well as various hash functions and message authentication codes. It also contains a true random number generator (TRNG), which provides the entropy source for cryptographic processing. The security engine is protected by an access management circuit, which can shut down the security engine in the event of an illicit external access attempt. Figure 1-1 is a schematic diagram of the security engine. For the cryptographic processing supported by the security engine, see Table 1-1.

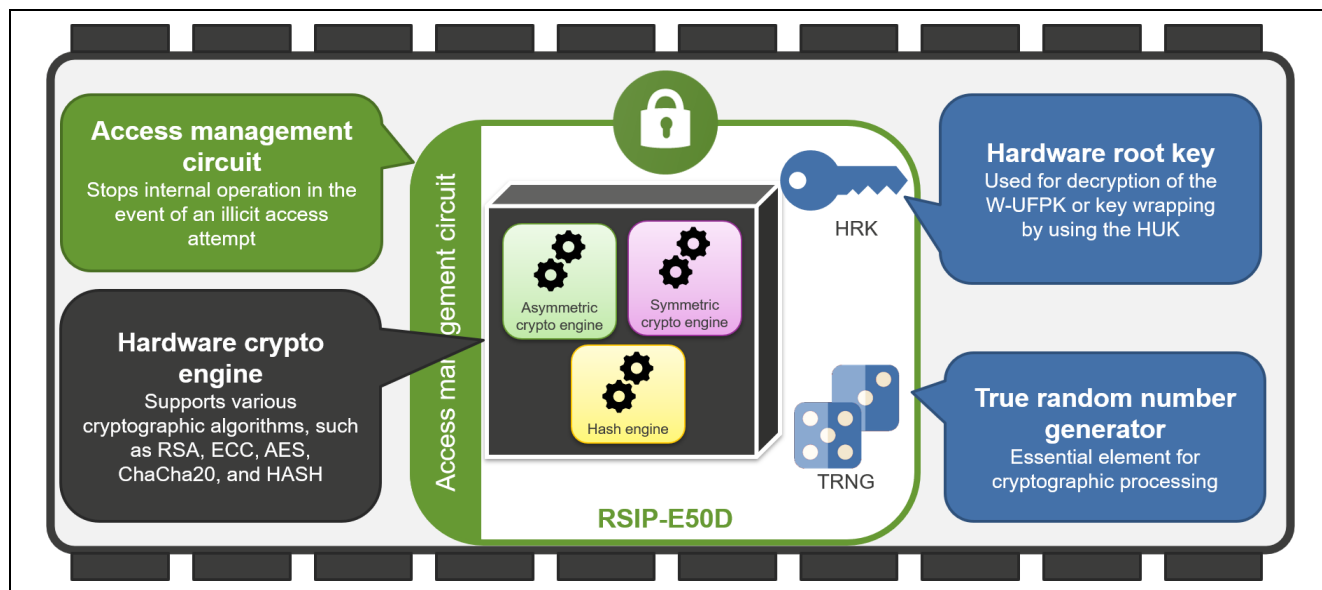


Figure 1-1 RSIP-E50D Security Engine

The hardware root key (HRK) is not a single key that is physically stored. It is represented in Figure 1-1 as such for simplifying the description of the concepts. The security engine contains internal RAM for operations that deal with sensitive material such as plaintext keys. This RAM is not accessible outside the security engine.

The security engine has its own dedicated internal RAM, enabling all cryptographic operations to be physically isolated within the security engine. This, combined with advanced key handling capability, means that it is possible to implement applications where there is no plaintext key exposure on any CPU-accessible bus.

Secure key storage and usage is accomplished by storing user keys in a wrapped format, encrypted by the hardware unique key (HUK) and tagged with a message authentication code. Since wrapped keys can only be unwrapped by the security engine within the specific MCU that wrapped them, the wrapping mechanism provides unclonable secure storage of user keys.

The RX Family also provides a secure key injection mechanism for the secure provisioning of devices. The security engine is packed full of types of cryptography functionality that are applicable to higher-level solutions, providing the option to use hardware acceleration to reduce both execution times and power consumption.

Table 1-1 lists the types of cryptographic functionality associated with the security engine.

Table 1-1 Renesas Secure IP and Types of Cryptographic Functionality Associated with the Security Engine

Functionality			RX74M
Mounted cryptographic IP module			
	Security engine		RSIP-E50D
Authentication and key exchange (encryption mode of public keys)			
	RSA	Key generation, sign/verification, encryption/decryption	Up to 4 Kbits
	ECC	Key generation, sign/verification, encryption/decryption	Up to 521 bits
Confidentiality protection (encryption mode of common keys)			
	AES	ECB, CBC, CTR	128, 192, 256 bits
		XTS	128, 256 bits
		CCM, GCM, CMAC, GMAC	128, 192, 256 bits
	ChaCha20	Poly1305	Yes
Data integrity			
		HMAC	SHA-2, SHA-3
		SHA-2	224, 256, 384, 512 bits
		SHA-3	224, 256, 384, 512 bits
Random number generation			
	TRNG	128 bits	Yes
Key handling			
	Wrapped key	Key protection by using the HUK	Yes
	Plaintext key	Injection of plaintext keys	Yes (only in compatibility mode)

The MCU-unique HUK is a 256-bit random key for the RSIP, which is injected at the Renesas factory. This key is stored in a wrapped format by using a wrapping mechanism unique to the MCU.

Since the HUK is unique to each MCU, even if an attacker was able to extract a stored user key, another MCU would not be able to use it.

In secure key injection, user keys are not written to the MCU in plaintext without change, but instead are passed to the MCU in a wrapped format from which secure user keys are generated inside the RSIP. In this processing, key information, such as the user factory programming key (UFPK) and wrapped UFPK (W-UFPK) are used. The UFPK is a key selected by the user to be used when injecting user keys. The W-UFPK is key information produced by wrapping the UFPK with Renesas key information.

When updating keys, use the key update key (KUK) to be used for securely updating new user keys when a device is deployed in the field. Handling of the KUK depends on the operating mode of the RSIP. The differences of key injection and updating in protected mode and compatibility mode are explained in section 1.5, Differences between Protected Mode and Compatibility Mode.

1.4 Advantages of Renesas Secure Key Injection

The wrapped key mechanism of the RSIP, which is described in the preceding section, with the method of secure key injection and updating, increases resistance to the leakage or duplication of keys, compared to the storage or transfer of user keys in plaintext without change.

Secure key injection and updating, combined with the security engine's support of wrapped keys, address many vulnerabilities associated with the use of plaintext keys.

- Plaintext keys are not stored in code MRAM. Even in the event of a program memory breach, the encryption keys that must be kept confidential are protected.
- Plaintext keys are not stored in commonly accessible RAM. In the event of malicious code executing on the system, the encryption keys that must be kept confidential are still protected.
- Wrapped keys can be securely stored in MRAM as well as copied into external memory, thus enabling unlimited storage of encryption keys.

In addition, the Renesas key wrapping techniques protect against device cloning, as described in the following subsections.

1.4.1 Advantages of Key Wrapping over Key Encryption

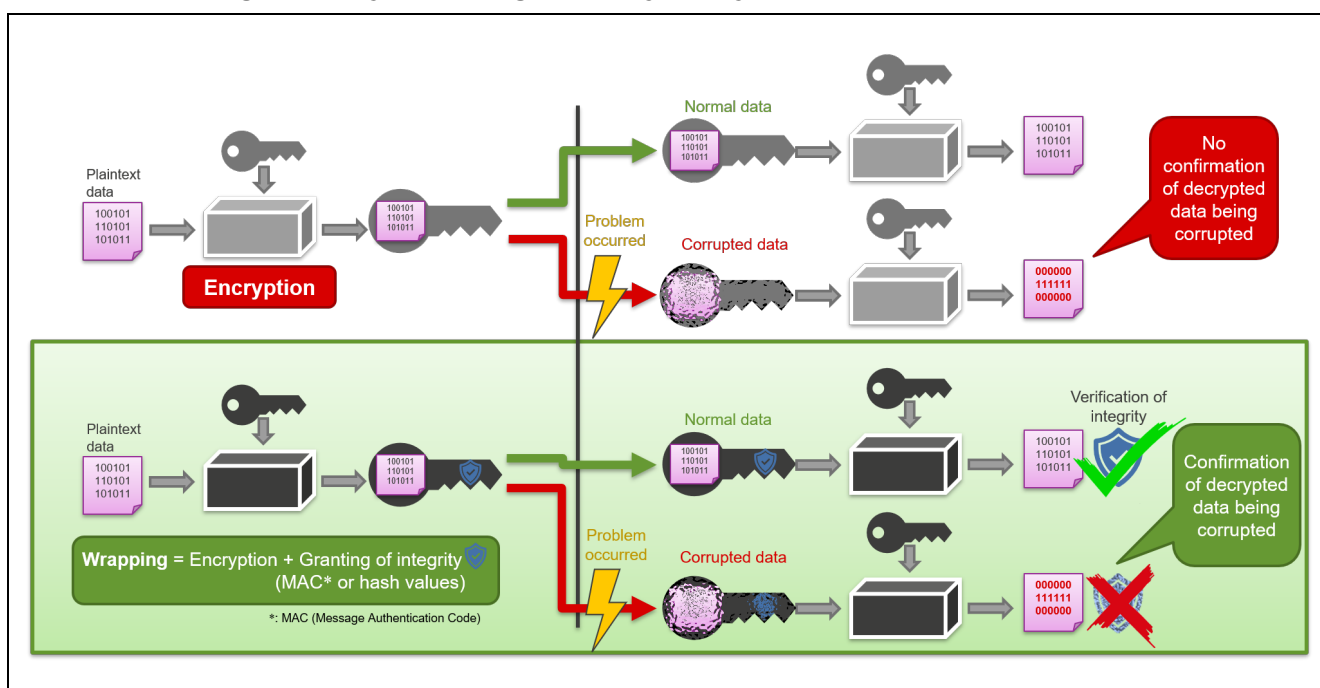


Figure 1-2 Wrapping and Encryption of Data

Understanding the difference between wrapping and encryption is important to secure asset storage.

When data are encrypted and sent to another recipient, if that recipient has the same key, the recipient will be able to decrypt the data. This results in an exchange of confidential information. However, what if there was a problem in 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 have been corrupted.

Wrapping solves this problem by appending a message authentication code for use in integrity checking to the encrypted output.

1.4.2 Advantages of Key Wrapping by Using the HUK

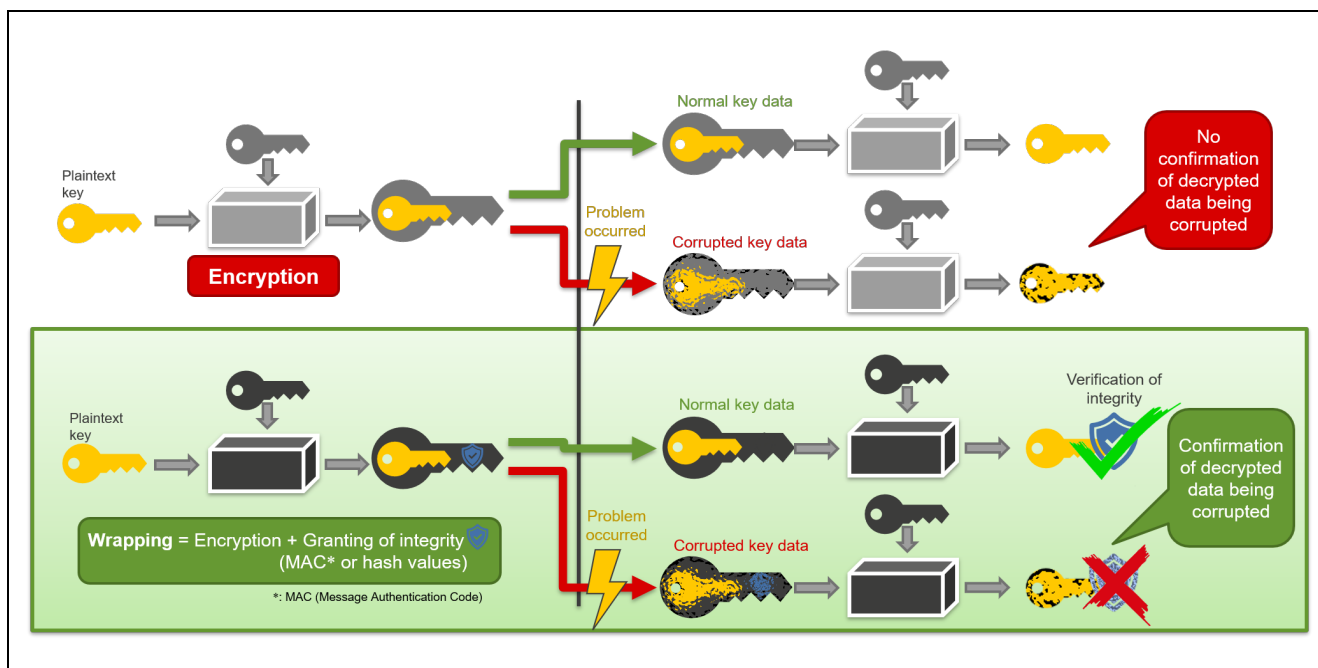


Figure 1-3 Wrapping and Encryption of Keys

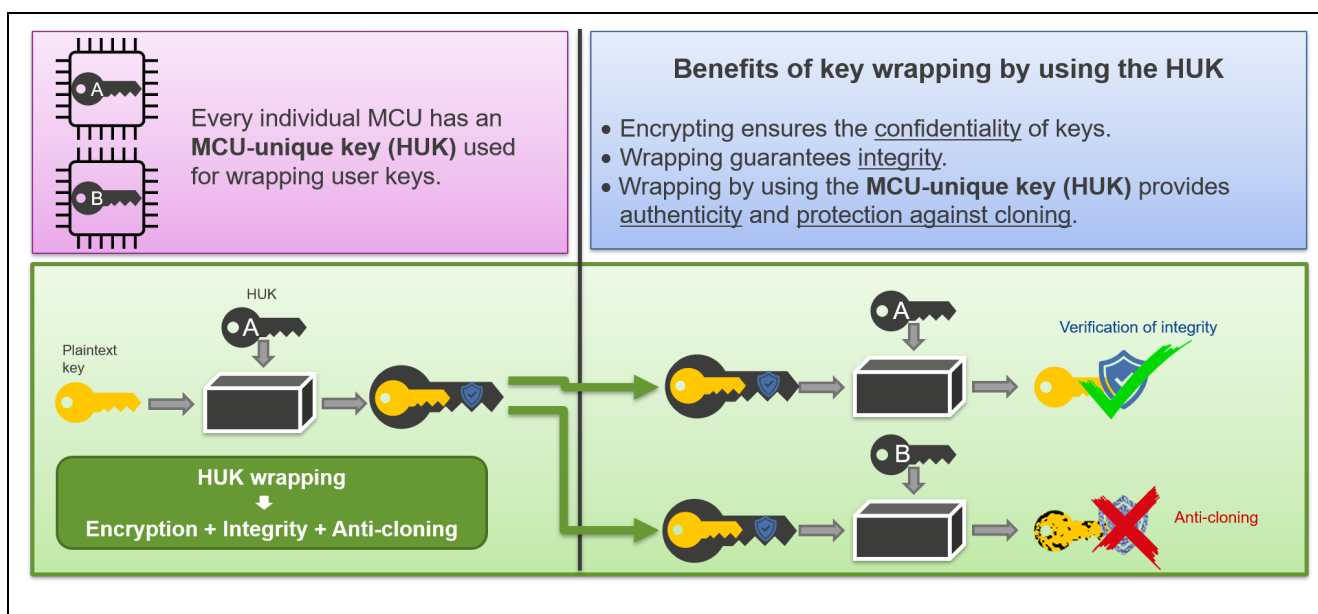


Figure 1-4 Key Wrapping by Using the HUK

Use of the HUK to wrap the stored keys adds another protection functionality called anti-cloning. If the wrapped key is transmitted or copied to another MCU, that MCU will not be able to either unwrap or use the copied key. Even if all of the MCU contents are copied onto another device, the keys are not exposed and cannot be used.

In the RSIP-E50D, protected mode and compatibility mode are available as operating modes when using the functionality to protect these keys. The differences of key injection and updating between the two modes are explained in the next section.

1.5 Differences between Protected Mode and Compatibility Mode

The RSIP-E50D has two operating modes: protected mode and compatibility mode. The interface used for key injection, the format of keys that can be handled, and the method of updating keys differ in the two modes. Since the preparation of the UFPK and W-UFPK, key data generated by the SKMT, method for injection to the MCU, and configuration of the sample projects differ according to the mode in use, the mode that is more appropriate for the target application has to be selected.

In protected mode, user keys are injected as secure keys. Keys should be injected through the interface of the MCU's boot firmware. Since the keys injected into the MCU are handled as if they are protected in the RSIP in this mode, a configuration in which plaintext keys are not stored in CPU-accessible areas can be achieved.

In protected mode, a KUK can be used to securely update the user keys when a device is deployed in the field. The KUKs have to be injected as secure keys just like user keys at the time of production provisioning. To update user keys in the field, the new user key must be wrapped in one of the previously injected KUKs and be updated as a secure key via the key update API function of the FSP.

Additional KUKs cannot be injected after the programming interface is disabled. Therefore, if keys might be updated during the product lifecycle, the injection of multiple KUKs at the time of production provisioning is recommended. This facilitates the handling of key rotation, KUK switchover, or response to a key exposure based on security policies.

In compatibility mode, the key injection API function of the FSP is used to inject keys. In this mode, plaintext keys as well as wrapped keys can be handled. This mode has compatibility with pre-existing software and is suitable for a configuration in which the application performs key injection by using the key injection API function of the FSP.

Compatibility mode, however, does not support key updating by using KUKs. For this reason, when updating keys in compatibility mode, keys have to be handled in a way that is different to that in protected mode. For details on key injection and updating in compatibility mode, refer to the user's manual of the FSP.

The W-UFPK and key injection data generated in protected mode are not compatible with those generated in compatibility mode. Therefore, correctly select the target MCU family, RSIP operating mode, and types of keys to be used when using the Renesas Key Wrap service and SKMT.

This application note describes ECC secp256r1 public key and private key secure key injection, and key updating by using the KUK as examples of operations in protected mode. In addition, AES-128 secure key injection is described as an example of operations in compatibility mode. For the entire procedures of specific information on key injection and updating processes, see section 2. For key injection, key updating, and a sample project in protected mode, see section 5. For key injection and a sample project in compatibility mode, see section 6.

2. Secure Key Injection and Updating

This section explains the steps for performing secure key injection and updating in the RSIP-E50D. The interface used and handling of keys differ between protected mode and compatibility mode. The general steps in each mode are explained in this section while implementation steps with specific key types as examples are explained in the following sections.

2.1 Key Injection and Updating in Protected Mode

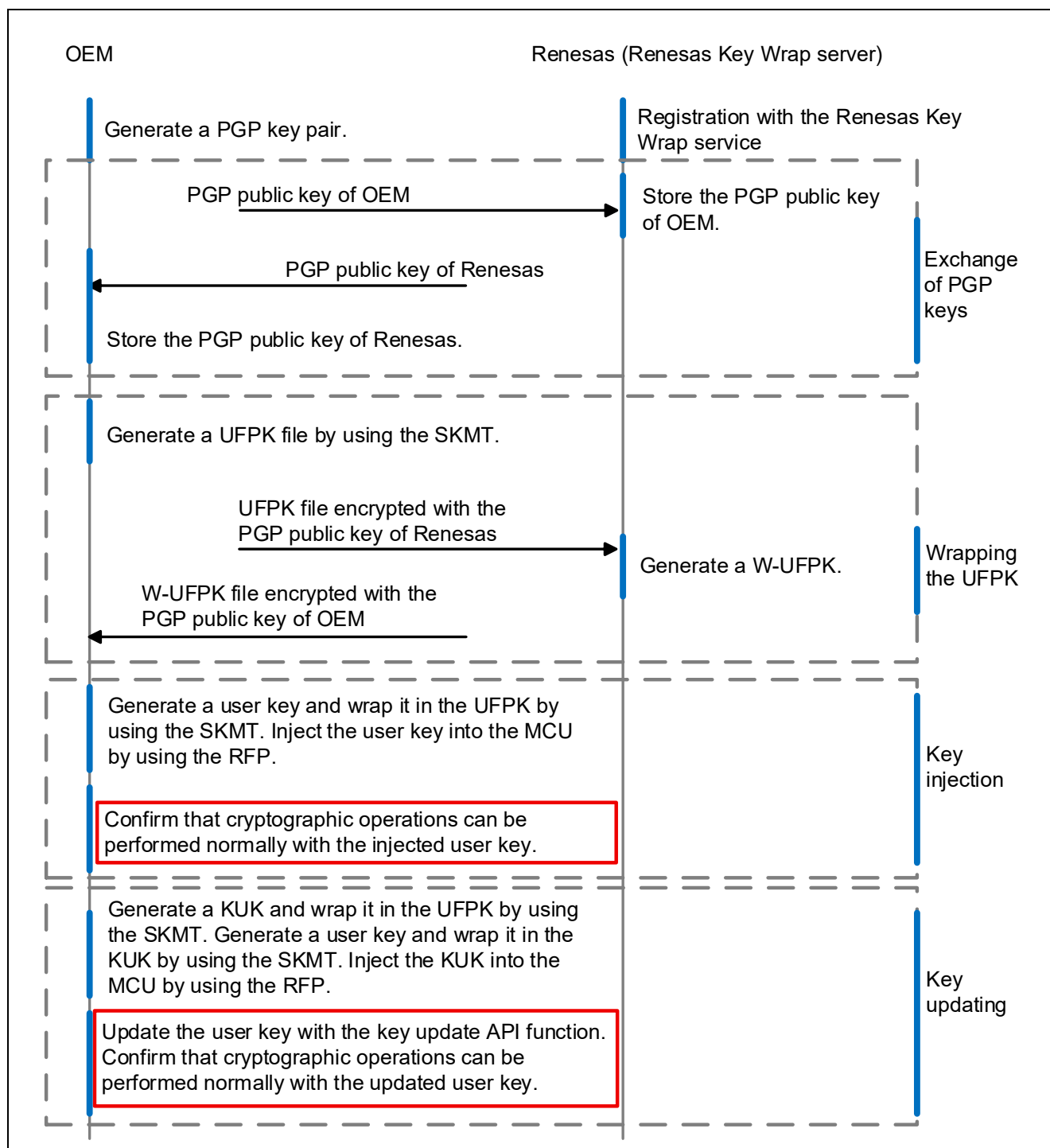


Figure 2-1 Flow of Key Injection and Updating in Protected Mode

Figure 2-1 shows the flow of key injection and updating in protected mode. Protected mode supports both secure key injection and key updating. Keys can be updated in the field by using a KUK. The detailed flow of steps is explained in sections 2.1.1 and 2.1.2.

The sample project for protected mode in this application note demonstrates the operations of key injection and updating in protected mode with the use of sample keys. These operations correspond to the blocks outlined in red in Figure 2-1. For details, see sections 3, 4, and 5.

2.1.1 Key Injection

Key injection generally requires the five steps listed and shown below. Among these steps, the steps of key preparation where encryption keys are handled in plaintext must be performed in a secure environment.

1. Generate a UFPK to be used for wrapping a user key. A UFPK can also be generated by using the SKMT. For details, see section 4.
2. Generate a W-UFPK by using the HRK (in blue) to wrap a UFPK (in green). Use the Renesas Key Wrap service to generate the W-UFPK. For details, see sections 3 and 4.

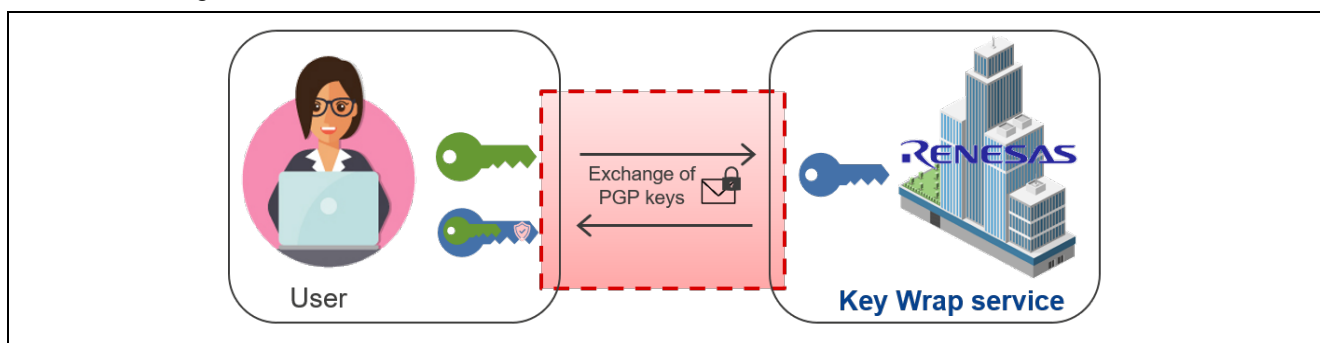


Figure 2-2 Wrapping the UFPK by Using the Renesas Key Wrap Server

3. Wrap the user key (in yellow) in the UFPK. The SKMT should be used in this step.

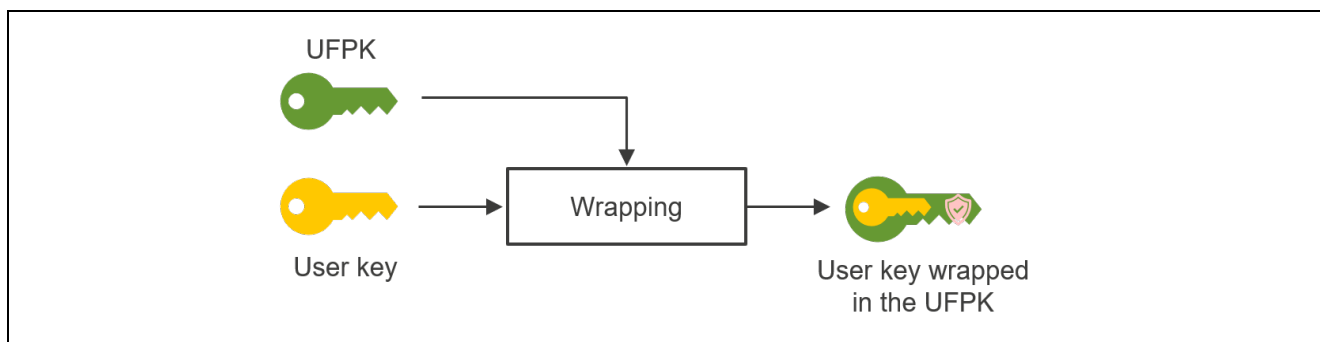


Figure 2-3 Wrapping the User Key in the UFPK

Key injection can be performed without exposure of a plaintext UFPK or plaintext user key by using a UFPK wrapped in the HRK and a user key wrapped in the UFPK. Generate each required type of wrapped key to execute this step.

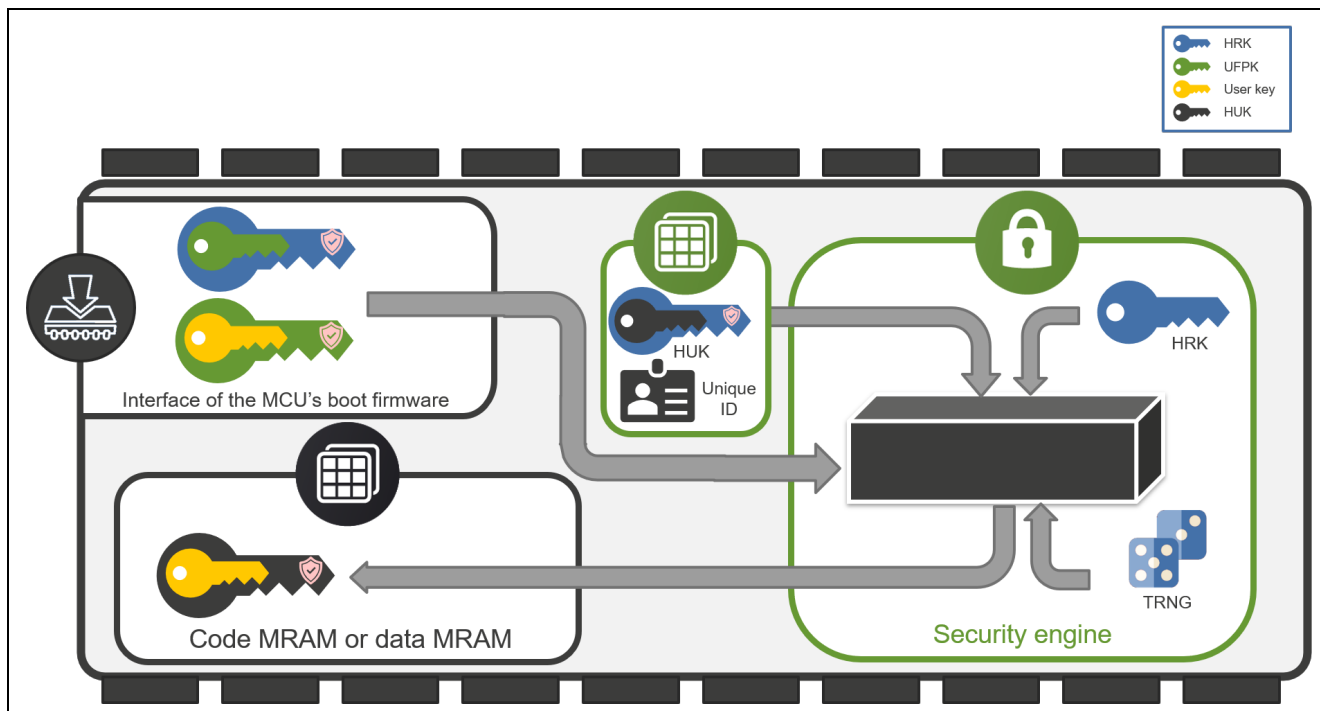


Figure 2-4 Injecting the User Key via the Serial Programming Interface (Protected Mode)

4. Generate key injection data by using the SKMT. Key injection data are generated as a file for a single operation of secure key injection including the wrapped user key and W-UFPK. Though the generated data differ according to the key type to be used, the basic procedure is common.
5. Finally, inject the key injection data by using the RFP. As shown in Figure 2-4, the security engine to which the key injection data were injected uses the HRK to unwrap the W-UFPK inside the security engine, then uses the unwrapped UFPK to unwrap the wrapped user key.

This procedure can be used in common with supported types of user key, such as an AES key, RSA key, or ECC key. Specific examples are given in section 5.

2.1.2 Key Updating

Since injecting new keys in the field is usually done by replacing older keys (key rotation or re-keying), this process is referred to as a “key updating”. In protected mode, to enable secure key updating in the field, one or more KUKs must be injected during programming or provisioning in production, as described above.

KUKs, like other cryptographic keys, can be stored in either code MRAM or data MRAM. Injection of the KUK uses the same procedure as injecting other user keys, as described in section 2.1.1. Since the KUK is the only mechanism by which new keys can be injected or wrapped, it is highly recommended that multiple KUKs be injected at provisioning during production. This enables the KUK to be rotated or revoked to adhere to an infrastructure security policy or to respond to a security breach caused by key exposure.

Additional KUKs **cannot** be injected after the programming interface is disabled. Once a product is deployed in the field with its programming interface disabled, new keys can **only** be injected via a pre-existing KUK.

The KUKs can be stored in any code MRAM or data MRAM area during production. The location will be used as an argument of the key update API function for the injection of the new user key. A user can inject multiple KUKs and provide a scheme to rotate the keys based on a timed schedule or key leakage event. For security reasons, it is recommended that users disable the programming interface prior to deployment in the field.

Key updating generally requires the four steps listed and shown below. Note that the KUK must already have been injected into the MCU.

1. Generate a KUK. The KUK needs to be injected into the device in advance following the steps in section 2.1.1.
2. Generate a new user key to be used in updating.
3. Use the KUK (in red) to wrap the new user key (in yellow). The SKMT should be used in this step.

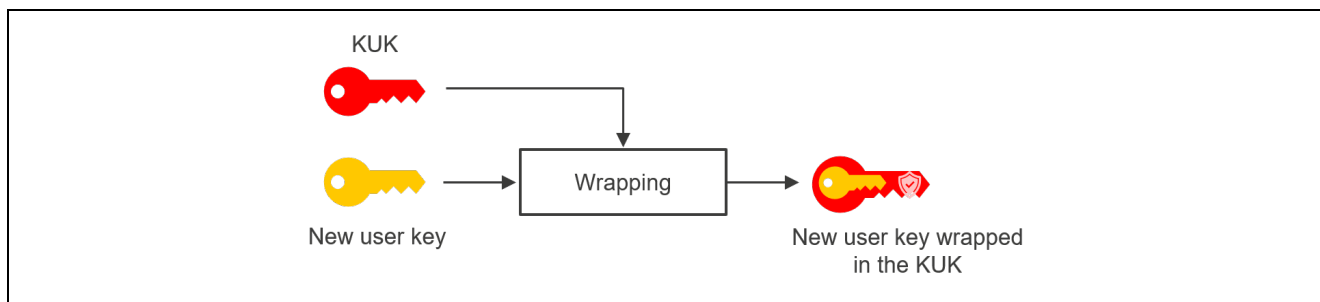


Figure 2-5 Wrapping the New User Key in a KUK

4. Use the key update API function of the FSP and the previously injected KUK to inject the new user key. The new user key is wrapped by the HUK (in black). Due to this, the user key after a key update is stored as a secure key.

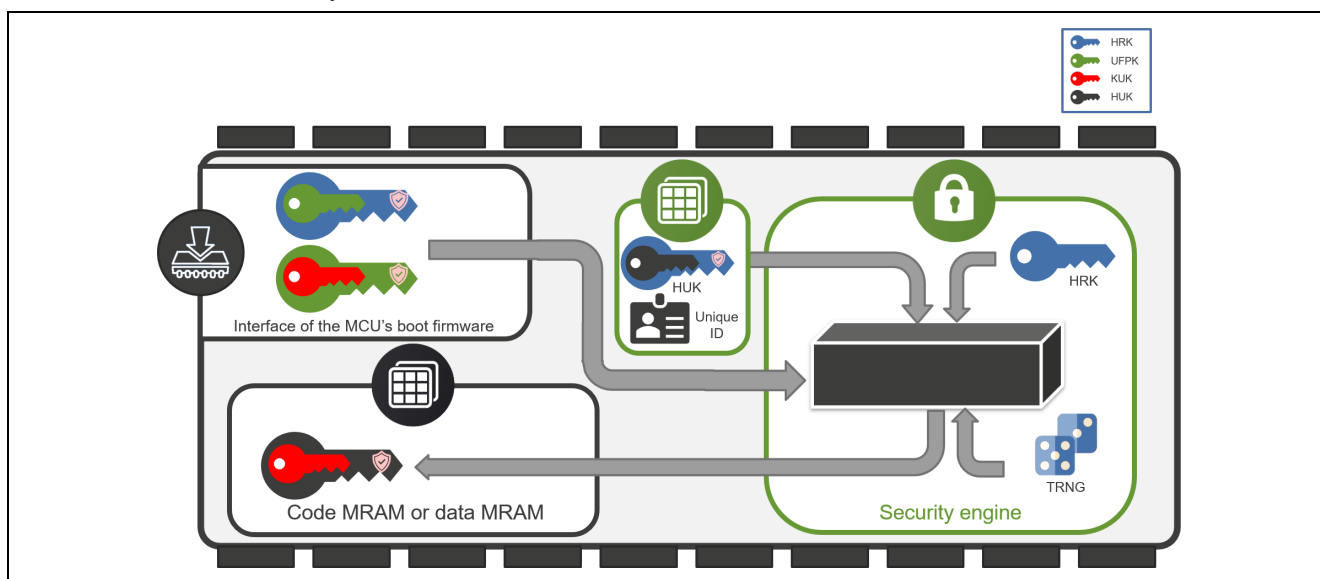


Figure 2-6 Updating the KUK

Though the generated data will differ according to the key type to be used, the basic flow of key updating is common with the types. Specific examples are given in section 5.

2.2 Key Injection in Compatibility Mode

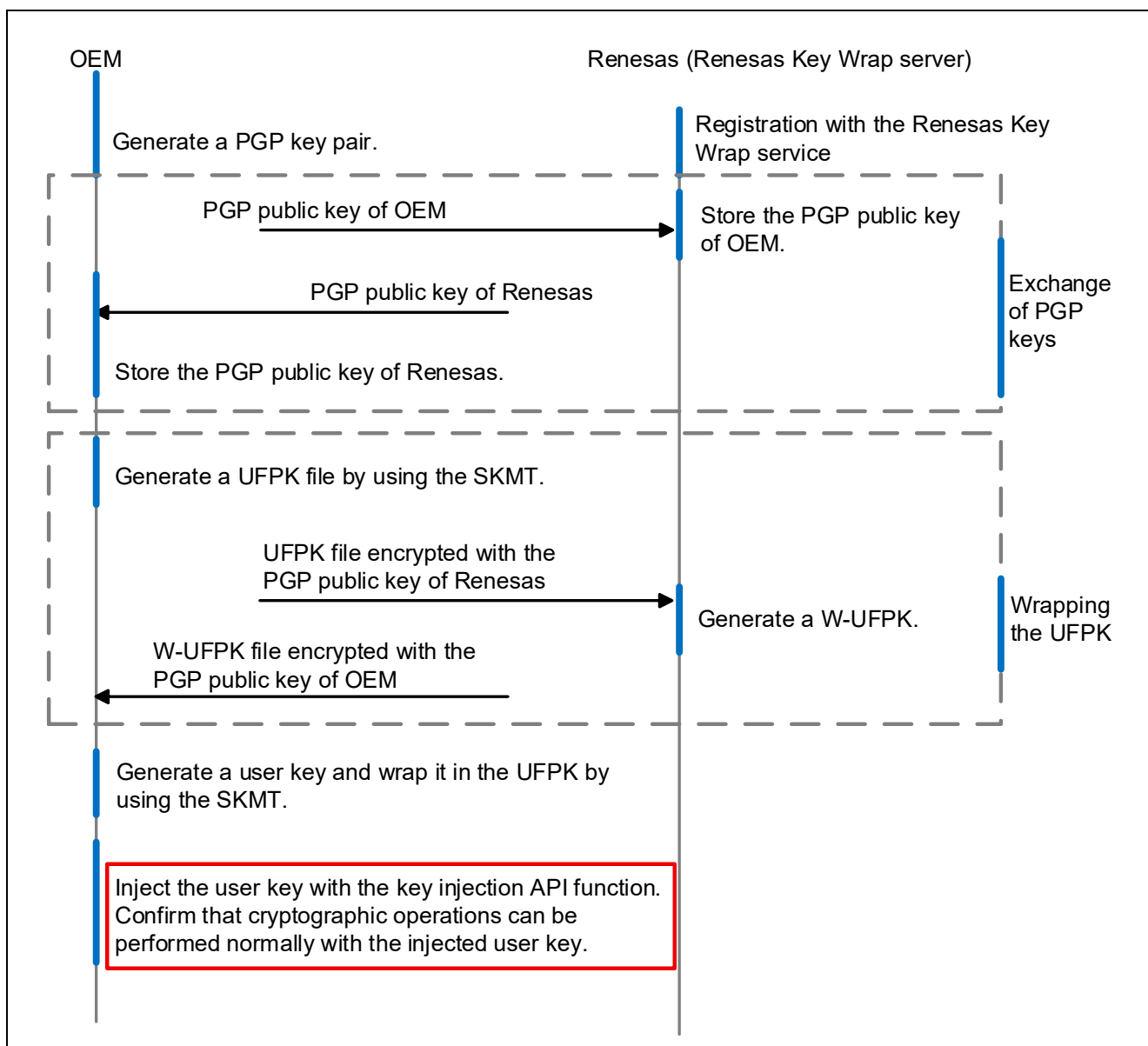


Figure 2-7 Flow of Key Injection in Compatibility Mode

Figure 2-7 shows the flow of key injection in compatibility mode. The sample project for compatibility mode in this application note demonstrates the operations of key injection in compatibility mode with the use of sample keys. This corresponds to the block outlined in red in Figure 2-7. For details, see sections 3, 4, and 6.

In compatibility mode, the key injection API function of the FSP is used to inject keys. Note that the interface for key injection is different from that in protected mode.

Key injection generally requires the four steps listed and shown below. Among these steps, steps 1 to 3 are common with the steps for key injection in protected mode. The steps of key preparation where encryption keys are handled in plaintext must be performed in a secure environment.

1. Generate a UFPK to be used for wrapping a user key. A UFPK can also be generated by using the SKMT. For details, see section 4.
2. Generate a W-UFPK by using the HRK to wrap a UFPK. Use the Renesas Key Wrap service to generate the W-UFPK. For details, see sections 3 and 4.
3. Wrap the user key in the UFPK.
4. Generate key injection data by using the SKMT. Key injection data are generated as a C source file including the wrapped user key and W-UFPK. This C source file is implemented so that its data are passed to the key injection API function of the FSP. Though the generated data differ according to the key type to be used, the basic procedure is common.

As shown in Figure 2-8, the security engine to which the key injection data were injected unwraps the W-UFPK inside the security engine by using the HRK, then unwraps the wrapped user key by using the unwrapped UFPK.

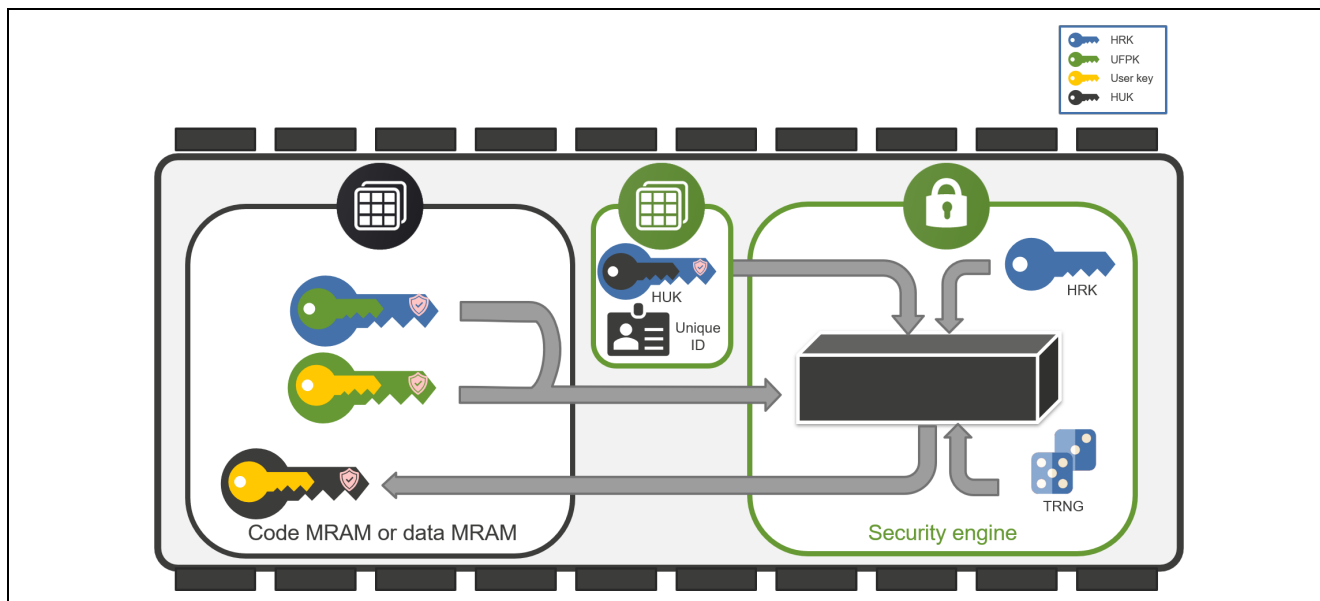


Figure 2-8 Injecting the User Key via an API Function (Compatibility Mode)

Key updating by using a KUK is not supported in compatibility mode.

This procedure can be applied to a variety of supported user keys. Specific examples are given in section 6.

2.3 Tools Used in Secure Key Injection and Updating

The tools used in secure key injection and updating are shown here. For details on obtaining, setting up, and using these tools, see the corresponding section mentioned below.

2.3.1 Renesas Key Wrap Service

This tool is used to obtain a W-UFPK. The UFPK registered by the user is wrapped at the Renesas Key Wrap server with the use of PGP encryption, and the generated W-UFPK is obtained through E-mail. For details, see section 3.

2.3.2 Security Key Management Tool (SKMT)

This tool is used for wrapping user keys and KUKs. It takes the UFPK and W-UFPK as well as the key data to be wrapped as inputs and produces a file that includes the data for injecting a wrapped key as the output. For details, see sections 4, 5, and 6.

2.3.3 Renesas Flash Programmer (RFP)

This tool is used to initialize the device and write key data to MRAM in protected mode. Using the key data file generated by the SKMT, it writes key data to the specified MRAM address. For details, see section 5.

2.3.4 e² studio or Renesas Flexible Software Package (FSP)

This tool is used for the building of sample projects and code generation. The sample projects in this application note use source code generated by using the FSP. e² studio, which is as an integrated development environment (IDE) is used for building and running sample projects. For details, see sections 5 and 6.

3. Using the Renesas Key Wrap Service

The Renesas Key Wrap service must be used to obtain a W-UFPK for use in secure key injection. To allow the secure exchange of key information, PGP encryption is used in the Renesas Key Wrap service. Therefore, the PGP public key must be exchanged between the user and the Renesas Key Wrap server prior to using the service. This section explains the steps involved in establishing the PGP-encrypted communications channel. This is a one-time process and does not need to be repeated for different MCUs.

3.1 Generating a PGP Key Pair

If you already have a PGP key pair, that key can be used for the key exchange process. Otherwise, the following procedure can be used to generate a PGP key pair.

The PGP software demonstrated here is Gpg4win, which can be downloaded from this URL:
<https://www.gpg4win.org/>.

The screenshots included in this application note are based on `gpg4win-4.0.0.exe`. There may be minor graphic interface updates with later versions; however, the functionality used in this application note should persist.

Download and install the Kleopatra installer. After installation is completed, click on the **Next** button, then the **Finish** button to terminate the installer.

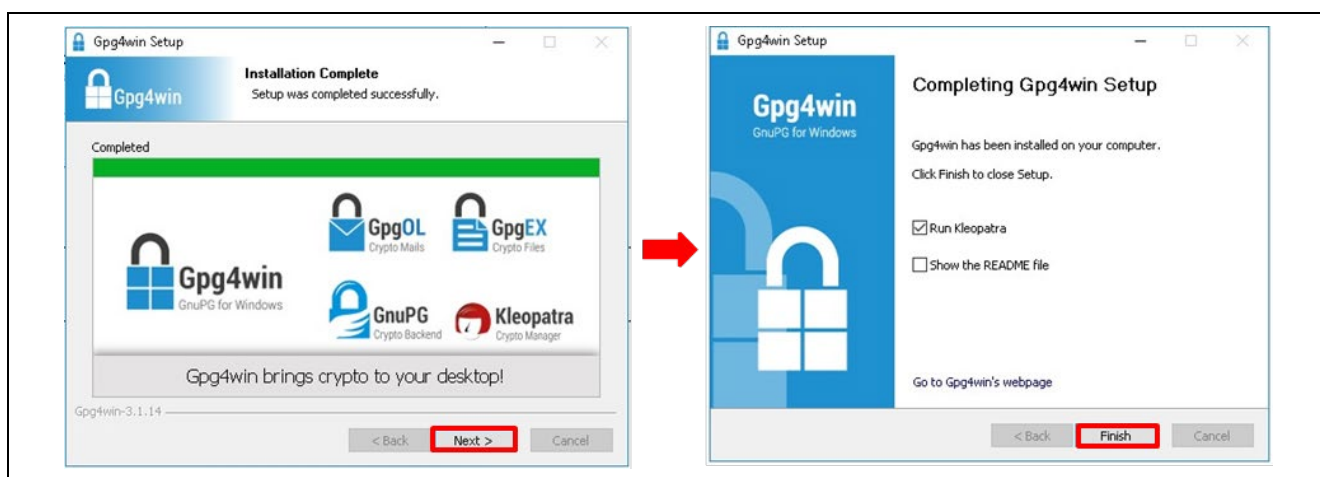


Figure 3-1 Downloading and Installing the Kleopatra Installer

Launch Kleopatra and generate a PGP key pair.

1. Click on **File -> Create a personal OpenPGP key pair**.

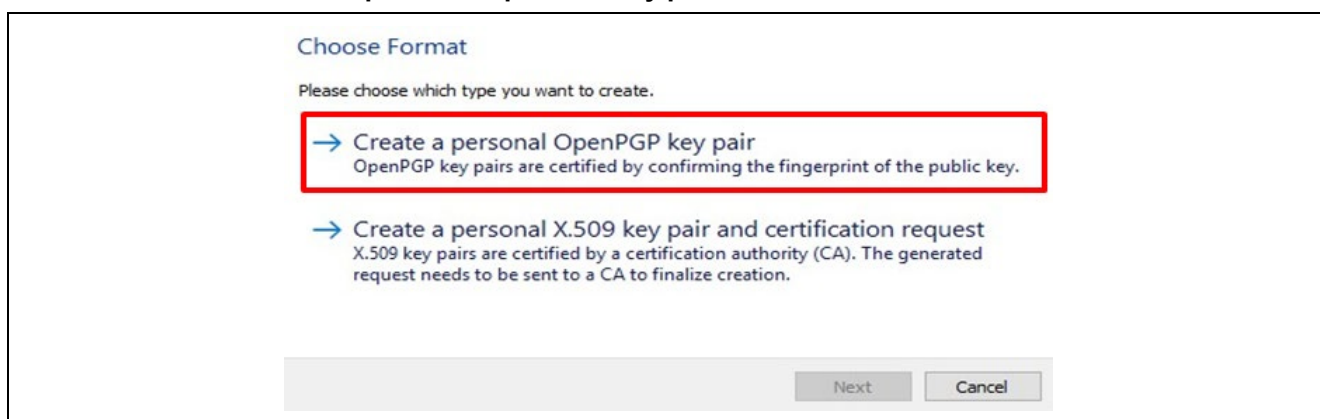
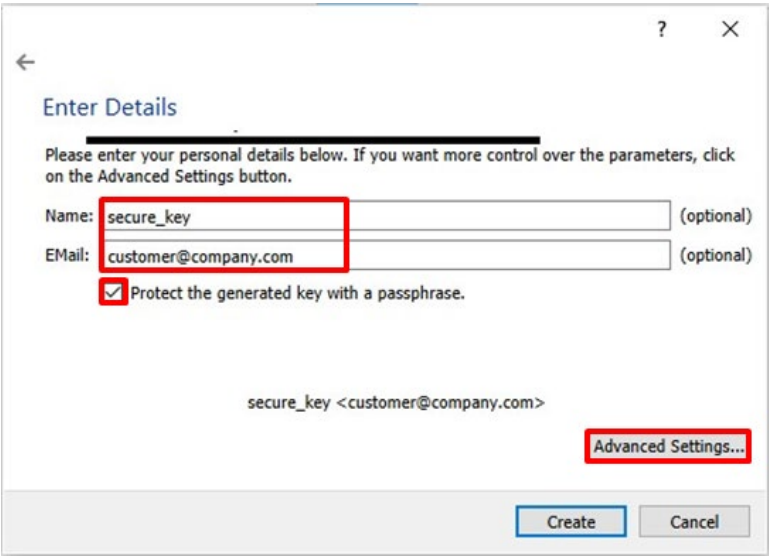


Figure 3-2 Generating a Personal OpenPGP Key Pair

- Specify a name and an E-mail address. **Note that even though these are marked as optional, at least one entity must be provided to move to the next stage.** Tick the **Protect the generated key with a passphrase** checkbox.



Enter Details

Please enter your personal details below. If you want more control over the parameters, click on the Advanced Settings button.

Name: (optional)

Email: (optional)

☒ Protect the generated key with a passphrase.

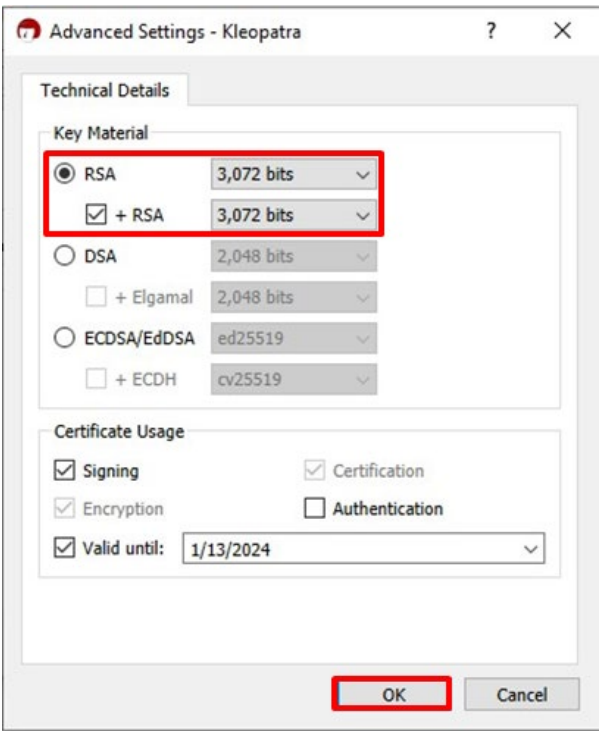
secure_key <customer@company.com>

[Advanced Settings...](#)

Create Cancel

Figure 3-3 Specifying a Name and E-mail Address

- Click on **Advanced Settings** and select **RSA** and **3,072 bits** as the type of key material. Click on **OK**.



Advanced Settings - Kleopatra

Technical Details

Key Material

☒ RSA 3,072 bits

☒ + RSA 3,072 bits

☐ DSA 2,048 bits

☐ + Elgamal 2,048 bits

☐ ECDSA/EdDSA ed25519

☐ + ECDH cv25519

Certificate Usage

☒ Signing ☒ Certification

☒ Encryption ☐ Authentication

☒ Valid until: 1/13/2024

OK Cancel

Figure 3-4 Selecting RSA Encryption

- Specify a passphrase twice to protect the private key. Then click on **OK**. **Be sure to save your passphrase.**

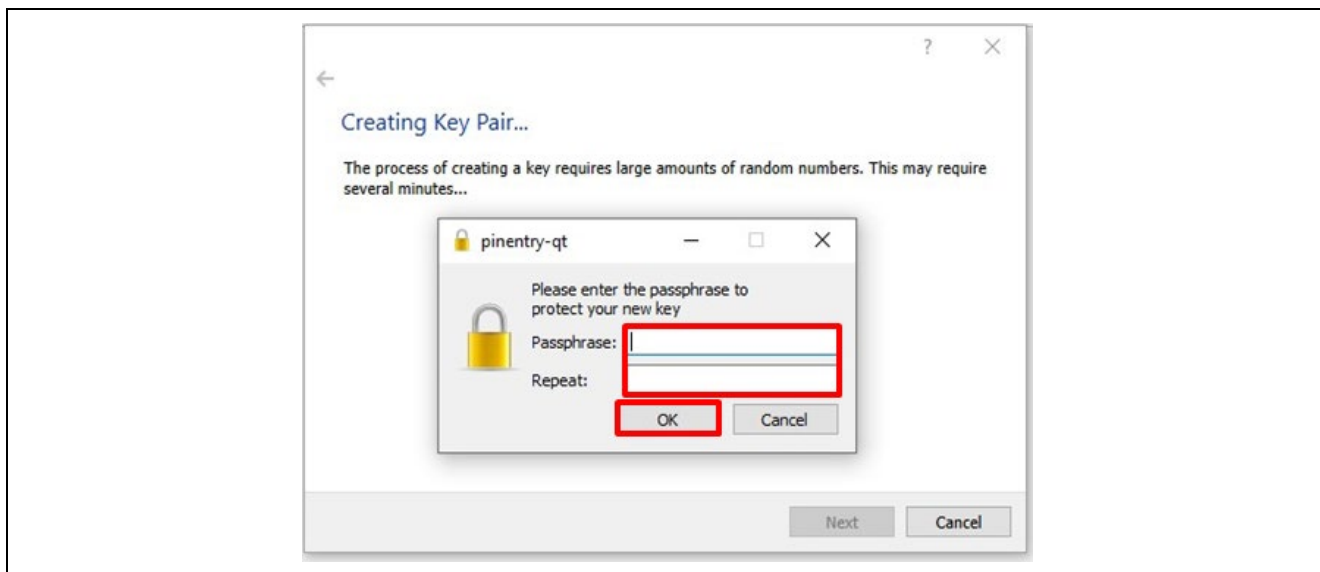


Figure 3-5 Setting a Passphrase

- The PGP key pair should be generated successfully. Click on **Finish**.

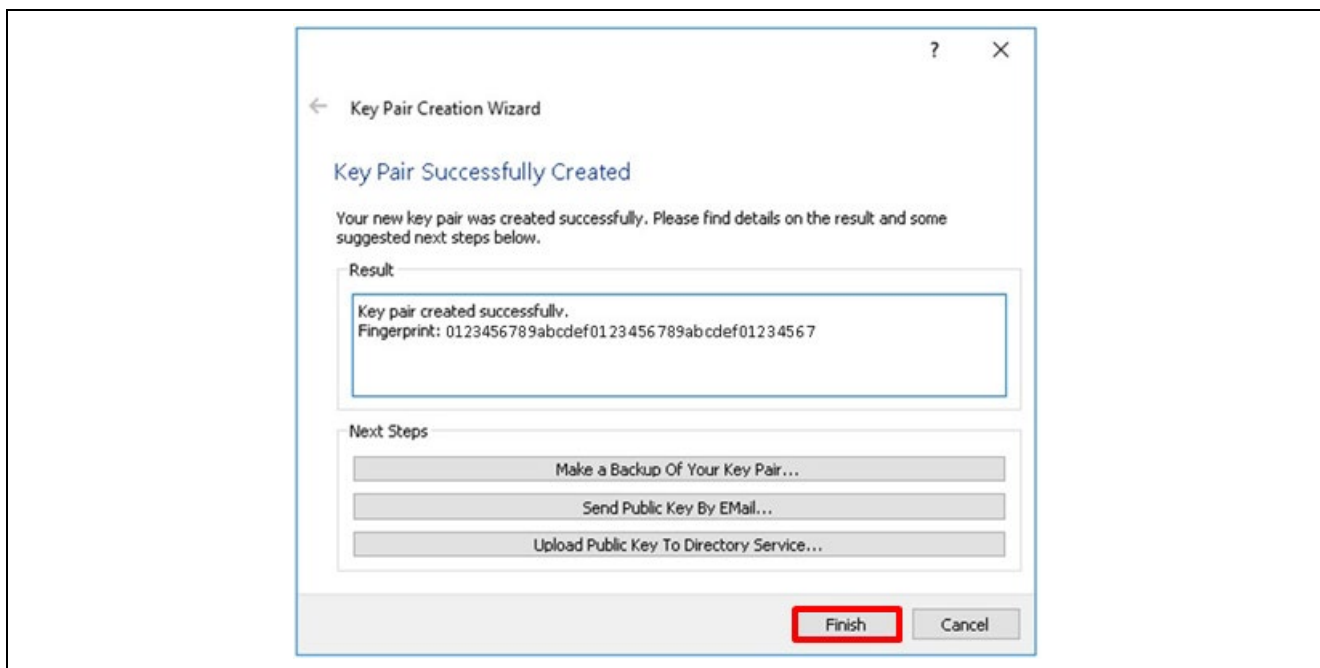


Figure 3-6 Generating a PGP Key Pair

6. A new item will be created in Kleopatra. Right-click on the key pair just generated and select **Export**.

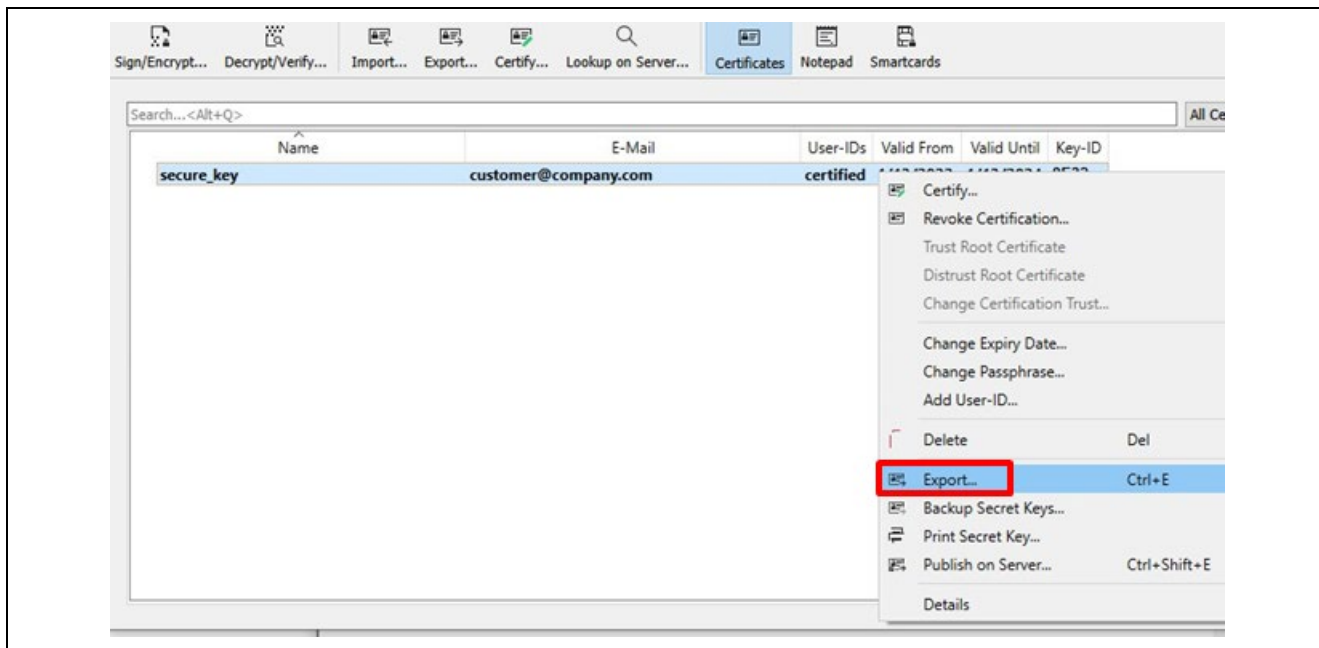


Figure 3-7 Exporting the User PGP Public Key

7. Save the public key to a file with an *.asc extension. In this example, this file is renamed to public_key.asc. Click on **Save**.

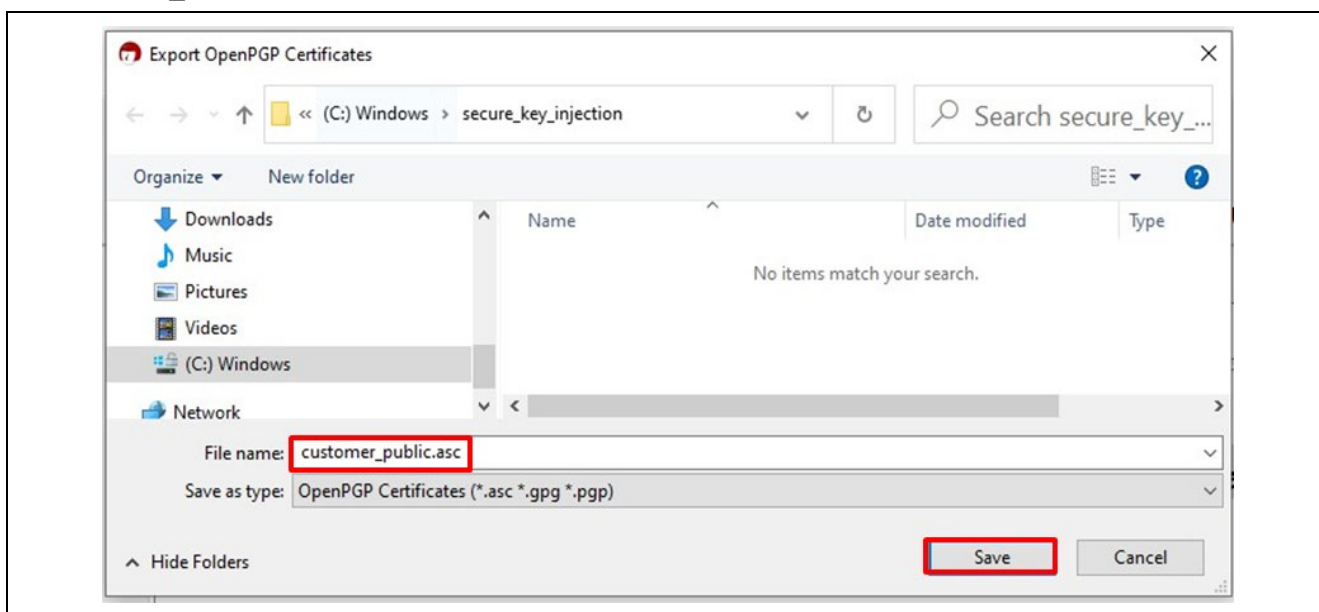
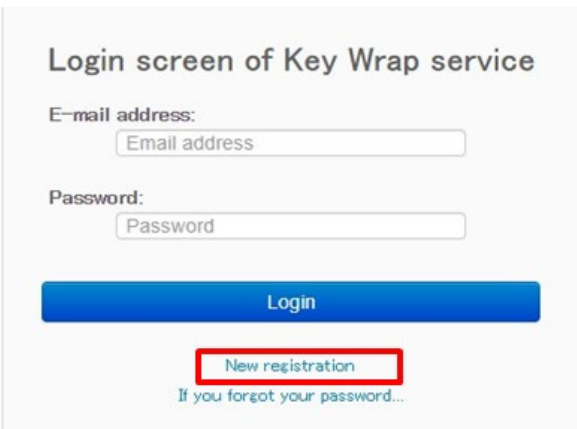


Figure 3-8 Saving the PGP Public Key in a Folder

3.2 Registration with the Renesas Key Wrap Server

The first time you use the Renesas Key Wrap service, you will have to register with the Renesas Key Wrap server.

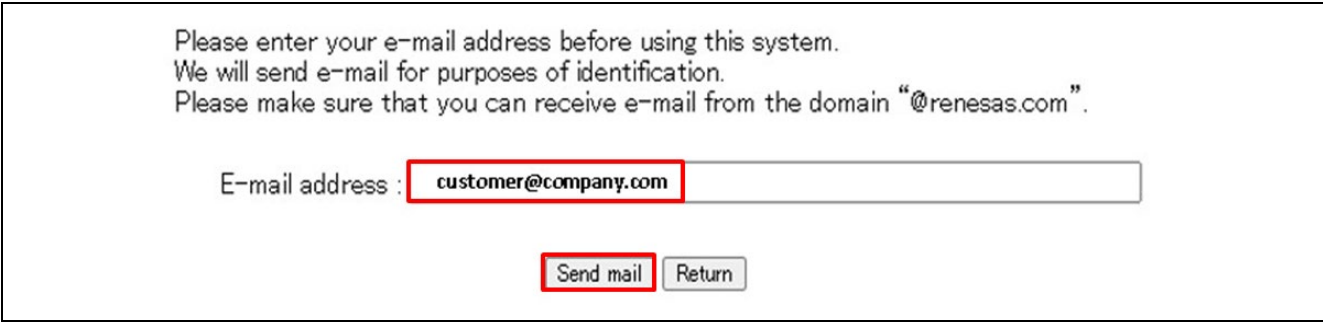
1. Open the URL <https://dlm.renesas.com/keywrap/toEnglish> in a browser and click on **New registration**.



The image shows the login screen of the Key Wrap service. It has a title "Login screen of Key Wrap service". Below the title, there are two input fields: "E-mail address:" and "Password:". Below these fields is a blue "Login" button. Below the "Login" button is a red-bordered box containing the text "New registration". Below the red-bordered box is the text "If you forgot your password...".

Figure 3-9 Starting Registration with the Renesas Key Wrap Server

2. Following the prompt, enter a **valid** E-mail address and click on **Send mail**.

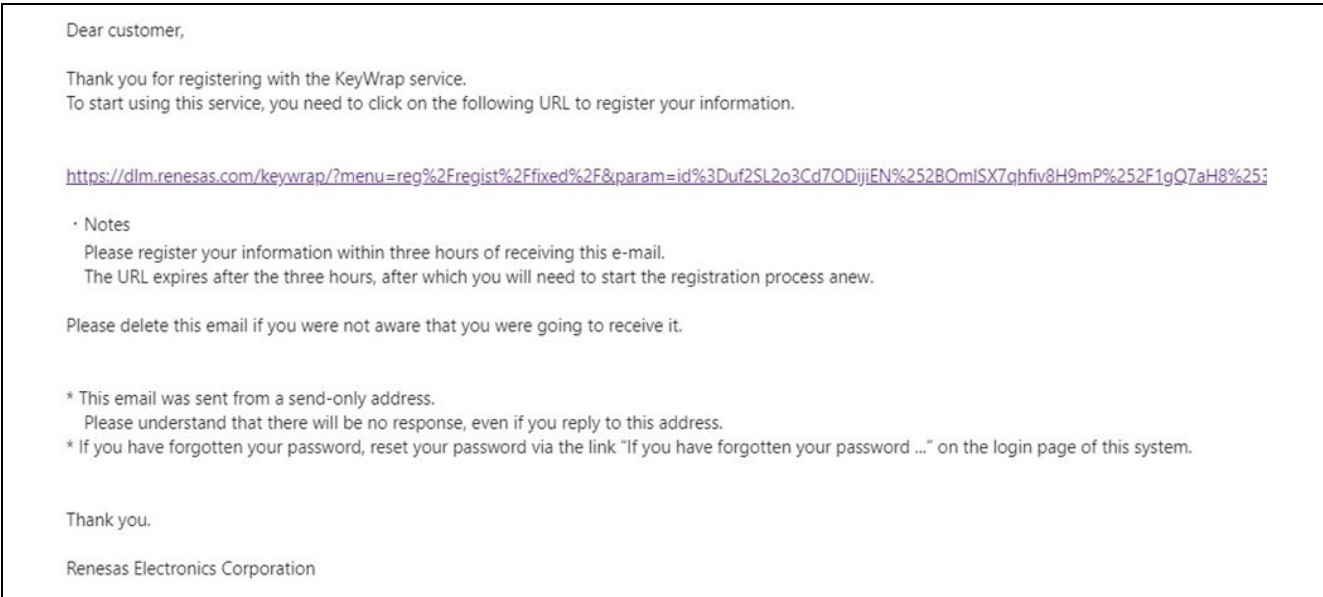


The image shows a registration prompt screen. It contains the following text: "Please enter your e-mail address before using this system. We will send e-mail for purposes of identification. Please make sure that you can receive e-mail from the domain '@renesas.com'". Below this text is an input field for "E-mail address:" with the text "customer@company.com" entered. Below the input field are two buttons: "Send mail" and "Return".

Figure 3-10 Registering the E-mail Address of the User

Clicking on **Send mail** displays the following message on the screen. Click on **Return**.

3. You should receive an E-mail similar to the one shown below. Click on the URL provided to confirm your registration.



The image shows a registration confirmation email. It contains the following text: "Dear customer, Thank you for registering with the KeyWrap service. To start using this service, you need to click on the following URL to register your information." Below this text is a URL: <https://dlm.renesas.com/keywrap/?menu=reg%2Fregist%2Ffixed%2F¶m=id%3Duf2SL2o3Cd7ODijjEN%252BOmlSX7ghfiv8H9mP%252F1gQ7aH8%253D>. Below the URL is a section titled "Notes" with the following text: "Please register your information within three hours of receiving this e-mail. The URL expires after the three hours, after which you will need to start the registration process anew." Below the notes is the text: "Please delete this email if you were not aware that you were going to receive it." Below this text is a section titled "Footnote" with the following text: "* This email was sent from a send-only address. Please understand that there will be no response, even if you reply to this address. * If you have forgotten your password, reset your password via the link 'If you have forgotten your password ...' on the login page of this system." Below the footnote is the text: "Thank you. Renesas Electronics Corporation".

Figure 3-11 Registration Confirmation E-mail

4. Follow the prompts to provide your name and company name and create a password. Click on the **Next (confirmation)** button. Note that the password must consist of 8 to 32 alphanumeric characters and may include the symbols "!" and "@".

Your information will be registered. Enter all of the following items.
The password is from 8 to 32 characters, which must be single-byte, and may include the symbols "!" and "@".

E-mail address :

Name :

Company Name :

Password :

Re-enter your password :

Figure 3-12 Confirming Registration

3.3 Exchanging PGP Public Keys of the User and Renesas

If you have not already exchanged the PGP keys with the Renesas Key Wrap server, follow the steps below.

3.3.1 Registering the User PGP Public Keys

1. After successfully registering the user information, the following screen will open. Click on the **Register** button to start using the Renesas Key Wrap service.

The following items will be registered. Are you sure?

E-mail address :

Name :

Company Name :

Password :

Re-enter your password :

Figure 3-13 Starting the Renesas Key Wrap Service

2. After confirming the terms of use shown below, accept the **Trusted Secure IP Key Wrap Service Agreement** by clicking on **I agree**. Note that this Agreement will come up whenever you log into the Renesas Key Wrap service.

---- CAUTION!!----

--- PLEASE READ THE FOLLOWING BEFORE USING THE SERVICE ---

This Trusted Secure IP Key Wrap Service Agreement (this "Agreement") is between you and Renesas Electronics Corporation. Please carefully note that this Agreement is legally valid agreement relating to Trusted Secure IP key encryption (the "Service").

Article 15 (ENTIRE AGREEMENT)

This Agreement sets forth the entire agreement of the parties with respect to the subject matter hereof and supersedes any prior or contemporaneous agreements, written or oral, concerning the subject matter hereof. Any change, modification or amendment of the terms of this Agreement shall not be effective unless reduced to writing and authorized by both parties.

I agree

I disagree.

[View PDF](#)

Figure 3-14 Agreement for Using the Renesas Key Wrap Service

3. When you log into the Renesas Key Wrap service, the window below appears. Click on **PGP key exchange**.

PGP key exchange

Display history

Refresh screen

Your PGP key has not been exchanged yet. Start by exchanging your PGP key.



The RZ family users.



The RX family users.



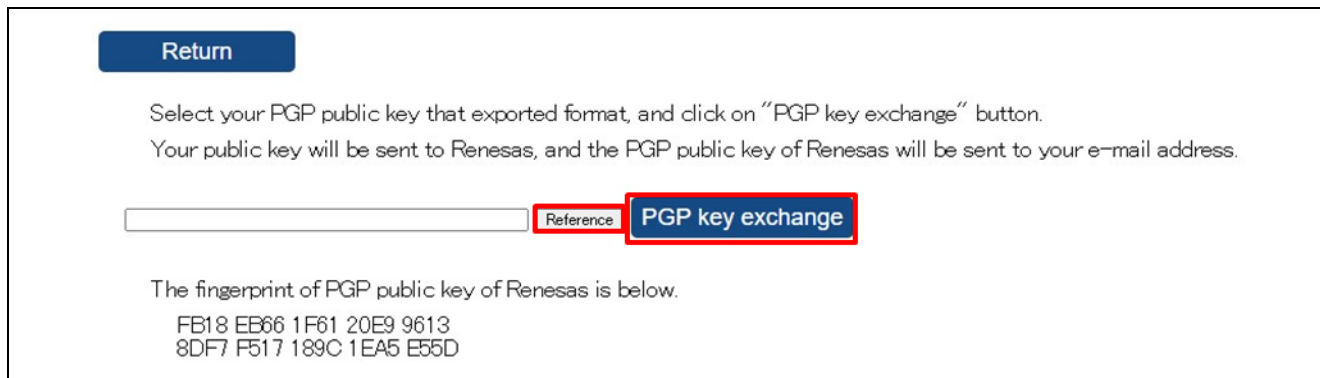
The RE family users.



The RA family users.

Figure 3-15 Starting the Exchange of PGP Keys

- Click on **Reference** and select the public key generated earlier (`public_key.asc`). Note that the fingerprint of the Renesas PGP public key is displayed. This will be used to certify the Renesas public key after you receive it. Then click on **PGP key exchange**.



Return

Select your PGP public key that exported format, and click on "PGP key exchange" button.
Your public key will be sent to Renesas, and the PGP public key of Renesas will be sent to your e-mail address.

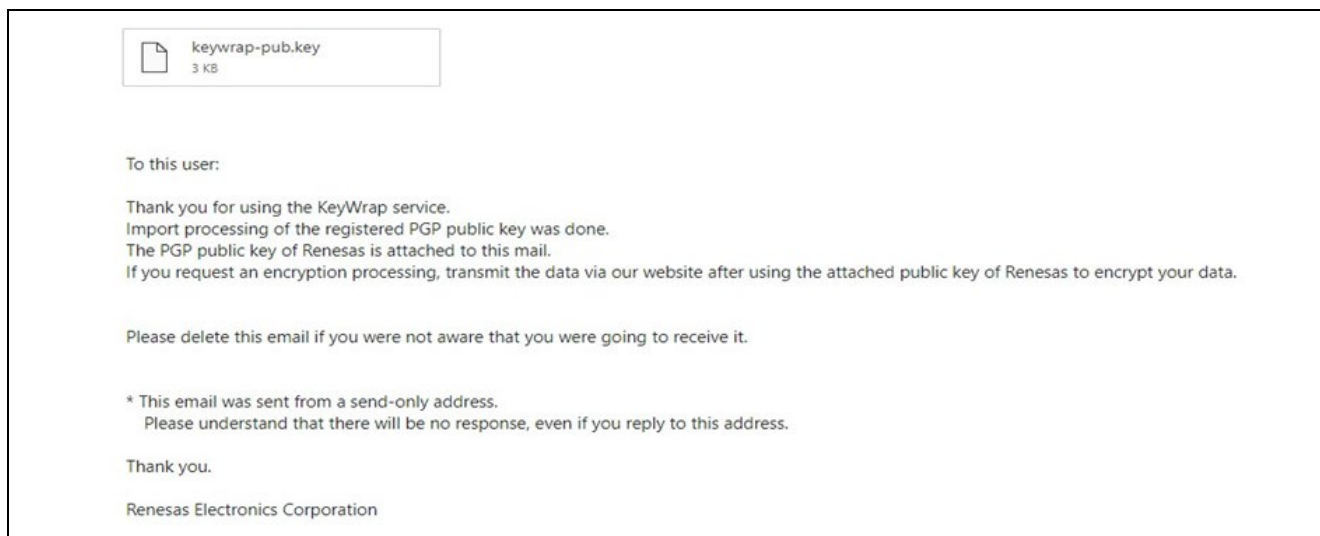
Reference **PGP key exchange**

The fingerprint of PGP public key of Renesas is below.
FB18 EB66 1F61 20E9 9613
8DF7 F517 189C 1EA5 E55D

Figure 3-16 Browsing the Customer's PGP Public Key

3.3.2 Obtaining the Renesas PGP Public Key

- If the key exchange is successful, you will receive an E-mail from Renesas at the E-mail address registered with the Renesas Key Wrap server with the contents as shown below. It typically takes about one to two minutes to receive this E-mail.
Note that a PGP public key can be registered any number of times. The latest PGP public key that has been registered successfully is used for encryption. All previously registered PGP public keys are discarded.



keywrap-pub.key
3 KB

To this user:

Thank you for using the KeyWrap service.
Import processing of the registered PGP public key was done.
The PGP public key of Renesas is attached to this mail.
If you request an encryption processing, transmit the data via our website after using the attached public key of Renesas to encrypt your data.

Please delete this email if you were not aware that you were going to receive it.

* This email was sent from a send-only address.
Please understand that there will be no response, even if you reply to this address.

Thank you.

Renesas Electronics Corporation

Figure 3-17 Receiving the Renesas PGP Public Key

Save the Renesas PGP public key file (`keywrap-pub.key`).

3.3.3 Importing the Renesas PGP Public Key

- 1. Go back to the Kleopatra application and import the Renesas PGP public key to Kleopatra as shown below.

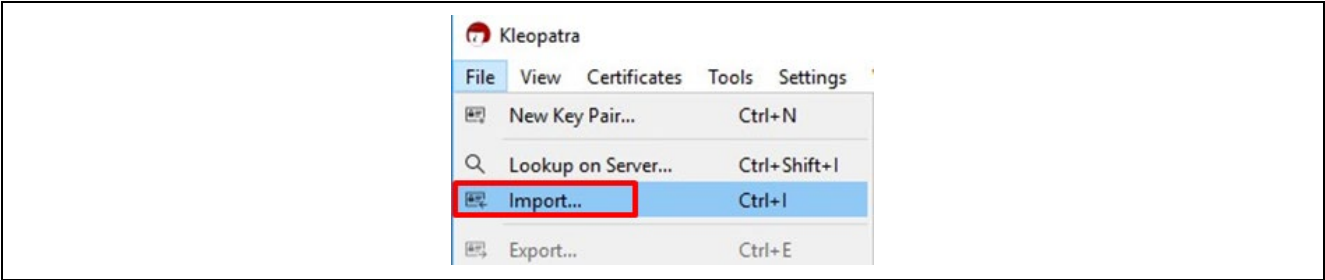


Figure 3-18 Importing the Renesas PGP Public Key

- 2. After having clicked on **Open**, the item below will have been added to the certificates imported to Kleopatra.

All Certificates						
Imported Certificates						
Name	E-Mail	User-IDs	Valid From	Valid Until	Key-ID	
keywrap	customer-key-encryption-system@lm.renesas.com	not certified	10/23/2018		F517 189C 1EA5 E55D	
secure_key	customer@company.com	certified	1/13/2022	1/13/2024	A58C 68B4 80A1 B869	

Figure 3-19 Renesas PGP Public Key

4. Generating a UFPK and Obtaining a W-UFPK

The Renesas Key Wrap service must be used to obtain a W-UFPK for secure key injection. This section describes the steps involved in using the Renesas Security Key Management Tool (SKMT) to generate a UFPK and in using the Renesas Key Wrap service to obtain a W-UFPK.

4.1 Generating a UFPK

4.1.1 Installing the SKMT

The SKMT performs several functions during the secure key injection process. Open the following link to access the latest SKMT.

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

From the above link, download the latest SKMT installer. This tool supports Windows, Linux, and macOS.

Once the executable file of the installer is downloaded, right-click on it and select **Run as administrator** to install this tool. Follow the prompt to select the **Setup Language**. Next, select the installation folder. By default, it will be installed into `C:\Renesas\SecurityKeyManagementTool\`. If an older version is installed, it will be overwritten.

The user's manual of this tool is located in the `\doc` folder. We recommend that you read through the user's manual before using the tool.

The SKMT provides two interfaces for users: a command-line interface (CLI) and a graphical user interface (GUI). The CLI is primarily intended for support of production. The GUI is primarily intended for development usage. This application note describes how to use both interfaces to inject and update keys.

4.1.2 Generating a UFPK through the GUI

Define a UFPK and convert it to a binary format that is compatible with the Renesas Key Wrap service. This can be done by using the SKMT.

The number of UFPKs to be generated and the method of UFPK usage and management depend on the security policies of the system. A W-UFPK is generated to correspond to the target MCU family and security engine mode, so select the correct MCU family and security engine mode when obtaining a W-UFPK.

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

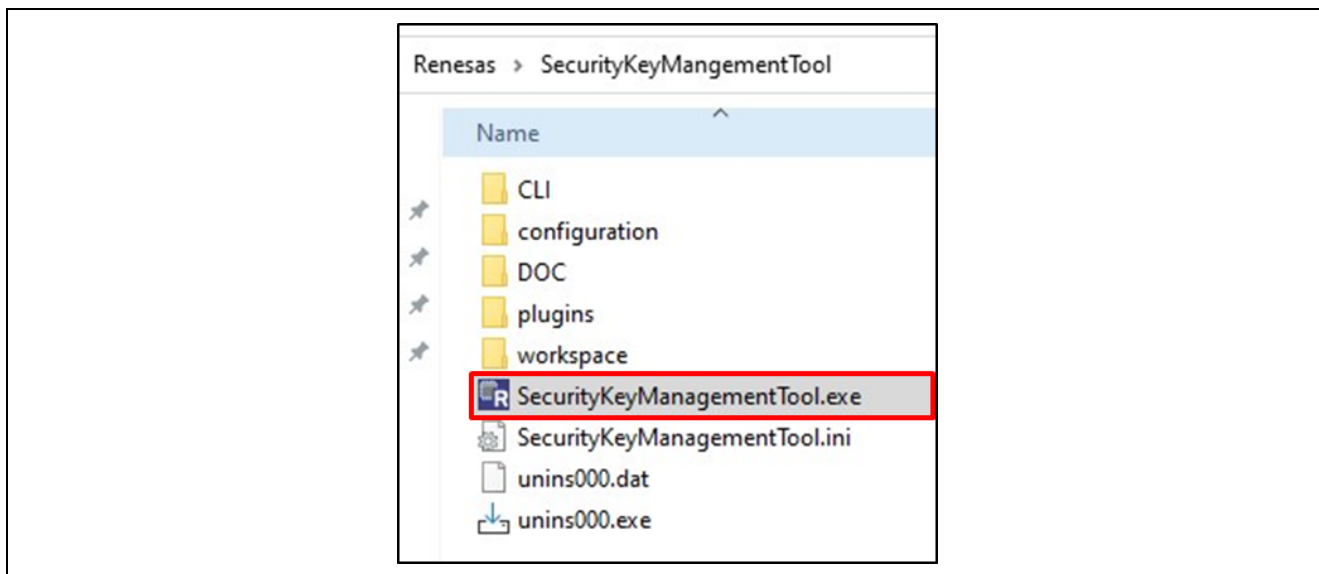


Figure 4-1 Launching the SKMT GUI

To use the sample projects included in this application project, set the UFPK to 000102030405060708090a0b0c0d0e0f000102030405060708090a0b0c0d0e0f.

Note that the 32-byte UFPK must be provided in big-endian format.

It is important to select the correct MCU family and security engine mode when using the SKMT tool.

- The RX74M incorporates the RSIP-E50D. To use sample project for RX74M protected mode, select **RX Family, RSIP-E50D Security Functions and Protected Mode** on the **Overview** tabbed page.

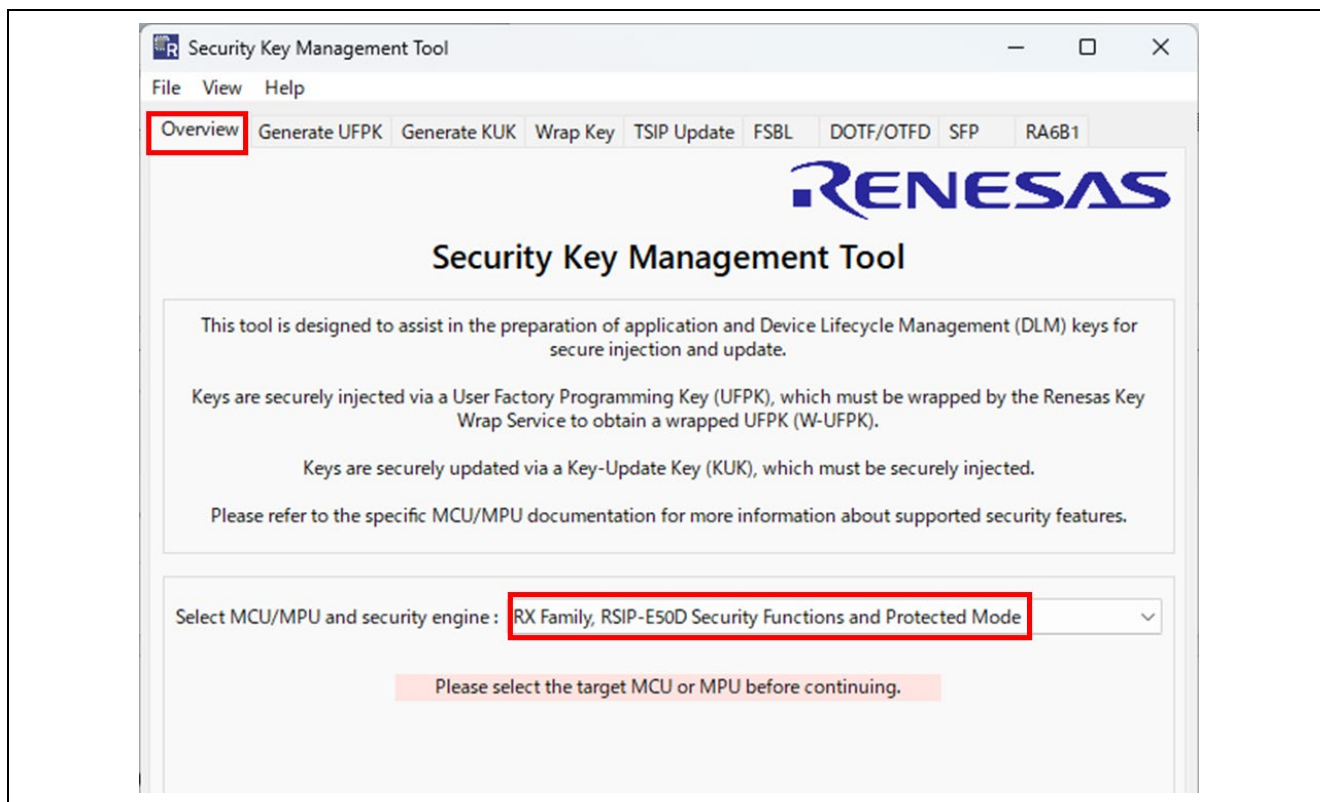


Figure 4-2 Selecting RX Family, RSIP-E50D Security Functions and Protected Mode

- To use the sample project for RX74M compatibility mode, select **RX Family, RSIP-E50D Compatibility Mode** on the **Overview** tabbed page.

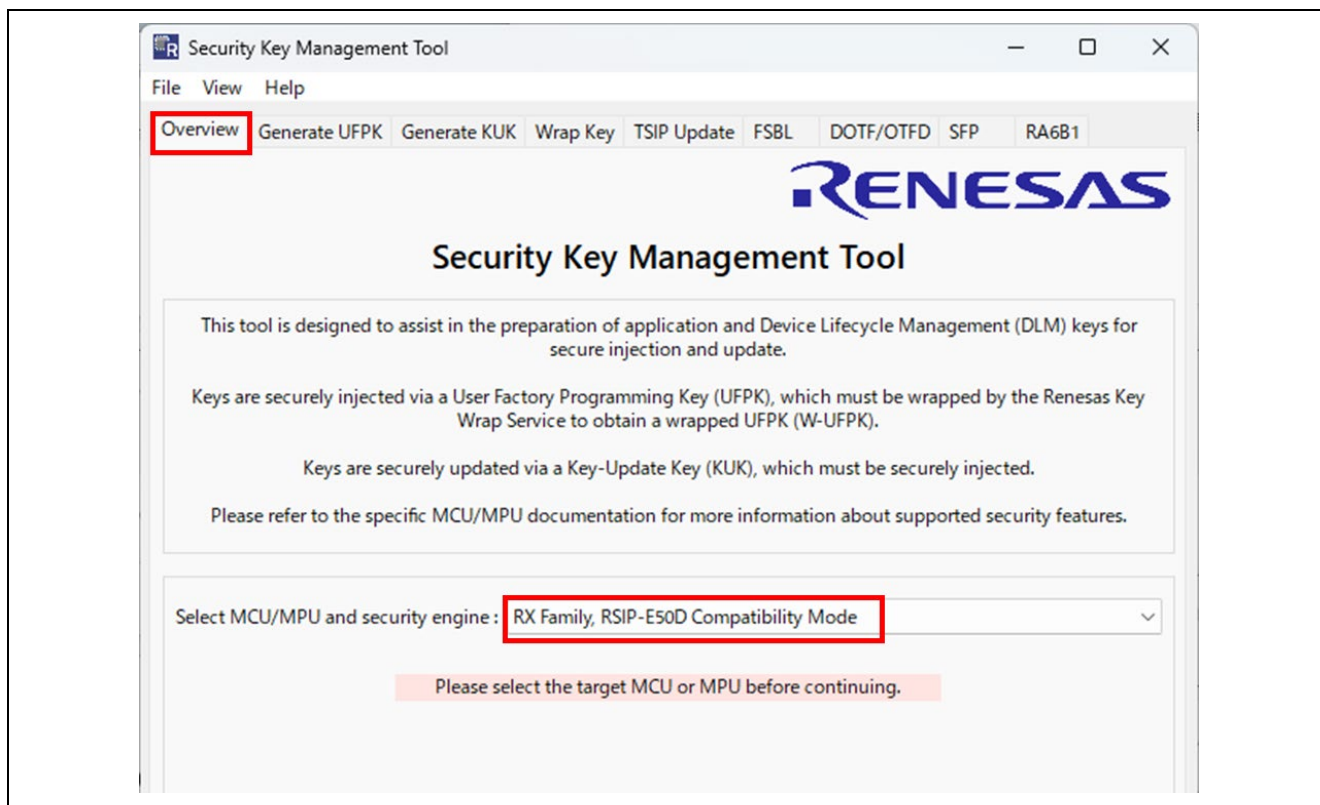


Figure 4-3 Selecting RX Family, RSIP-E50D Compatibility Mode

Once the correct MCU Family, security engine, and mode are selected, navigate to the **Generate UFPK** tabbed page.

- For **User Factory Programming Key**, select **Use specified value** and set the value. In this document, the value set to 000102030405060708090a0b0c0d0e0f000102030405060708090a0b0c0d0e0f.
- Click on the **Browse** button to select the folder for storing the key and name the resulting file. Choosing different file names that reflect different MCU families to avoid confusion at the UFPK wrapping stage is recommended. In this example, we name the file rx74m_ufpk.key.

Click on **Generate UFPK key file**. The rx74m_ufpk.key file will be generated.

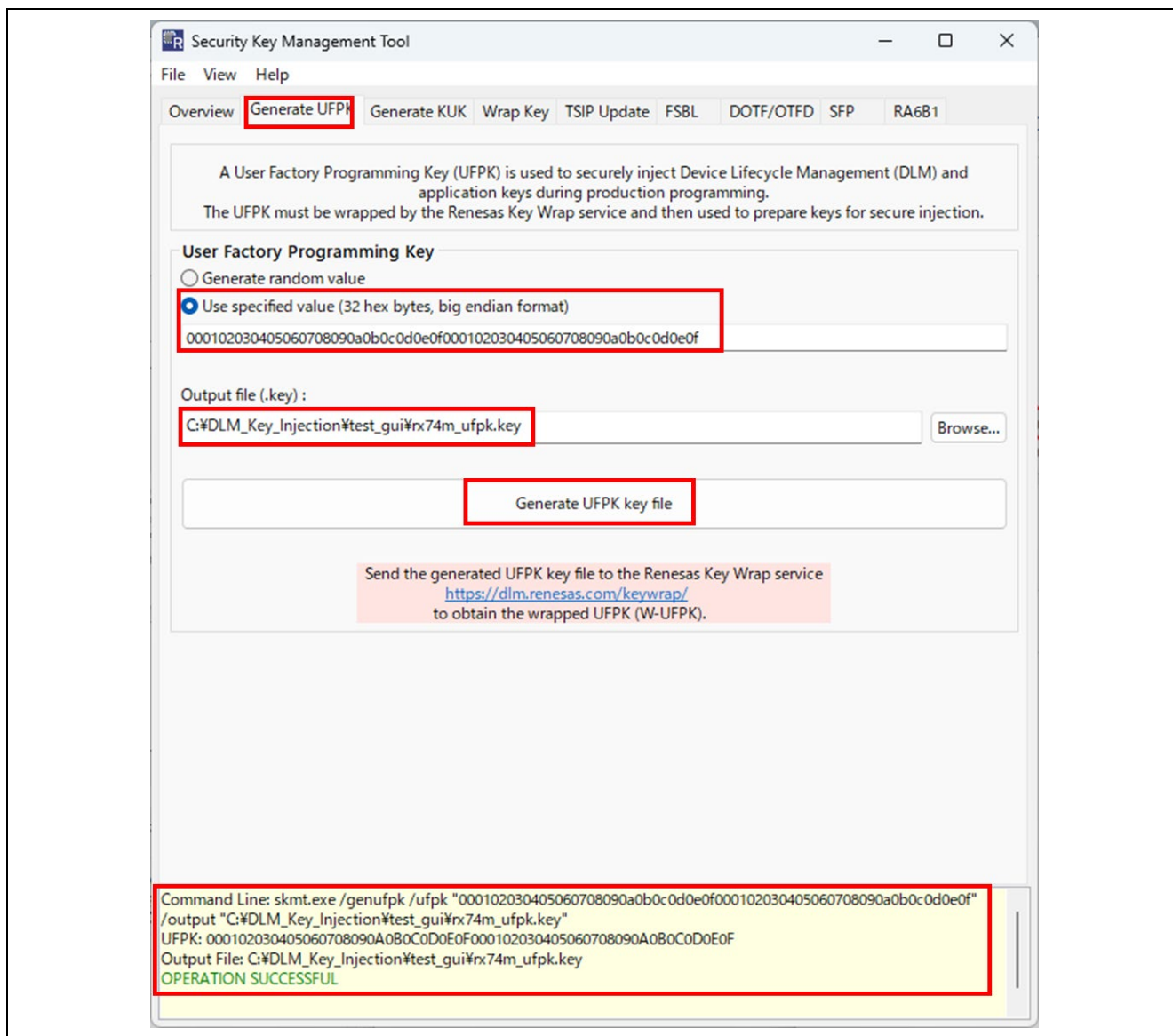


Figure 4-4 Generating a Fixed UFPK through the GUI for the RSIP-E50D

Alternatively, the user can generate a UFPK by choosing the **Generate random value** option.

4.1.3 Generating a UFPK through the CLI

Open a command prompt window and navigate to the folder where `skmt.exe` is stored. In the version of the SKMT used for the application note, this file is under `\Renesas\SecurityKeyManagementTool\cli\`.

Use the following command to generate a random UFPK and place it in a key file (`rx74m_ufpk.key`). If this is required, a complete file name including the path may be specified. For the usage of the `/genufpk` option, refer to the SKMT user's manual.

```
skmt.exe /genufpk /output  
"C:\Secure_Key_Injection\rx74m_protected_mode_key_info\rx74m_ufpk.key"
```

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

UFPK: E8AB23E99C9AD42823DA4215549A41496720F7243680A4715F4B944ACC94B691
Output File: C:\Secure_Key_Injection\rx74m_protected_mode_key_info\rx74m_ufpk.key

Figure 4-5 Generating a Random UFPK through the SKMT CLI

A specific UFPK can also be specified as shown in the following command:

```
skmt.exe /genufpk /ufpk  
"000102030405060708090a0b0c0d0e0f000102030405060708090a0b0c0d0e0f" /output  
"C:\Secure_Key_Injection\rx74m_protected_mode_key_info\rx74m_ufpk.key"
```

UFPK: 000102030405060708090a0b0c0d0e0f000102030405060708090a0b0c0d0e0f
Output File: C:\Secure_Key_Injection\rx74m_protected_mode_key_info\rx74m_ufpk.key

Figure 4-6 Generating a Fixed UFPK through the SKMT CLI

4.2 Obtaining a W-UFPK

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

4.2.1 Encrypting the UFPK with PGP

1. Launch the Kleopatra program.
2. Encrypt the UFPK with the Renesas public key. This key has already been imported to Kleopatra. Select **Sign/Encrypt** from Kleopatra and select the UFPK file. In the screenshot shown below, a sample file named `rx74m_ufpk.key` is used for demonstration purposes. Then click on **Open**.

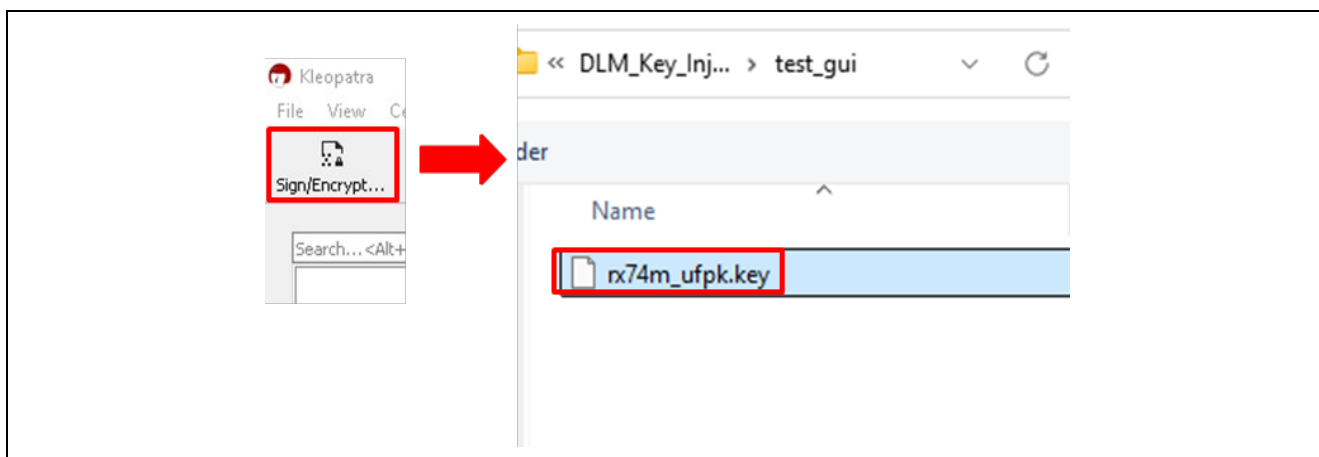


Figure 4-7 Encrypting the UFPK File for PGP Transfer

3. When asked who this file is to be encrypted for, deselect **Encrypt for me** if that is required and select **Sign as**, **Encrypt for others**, and **Encrypt/Sign each file separately**.

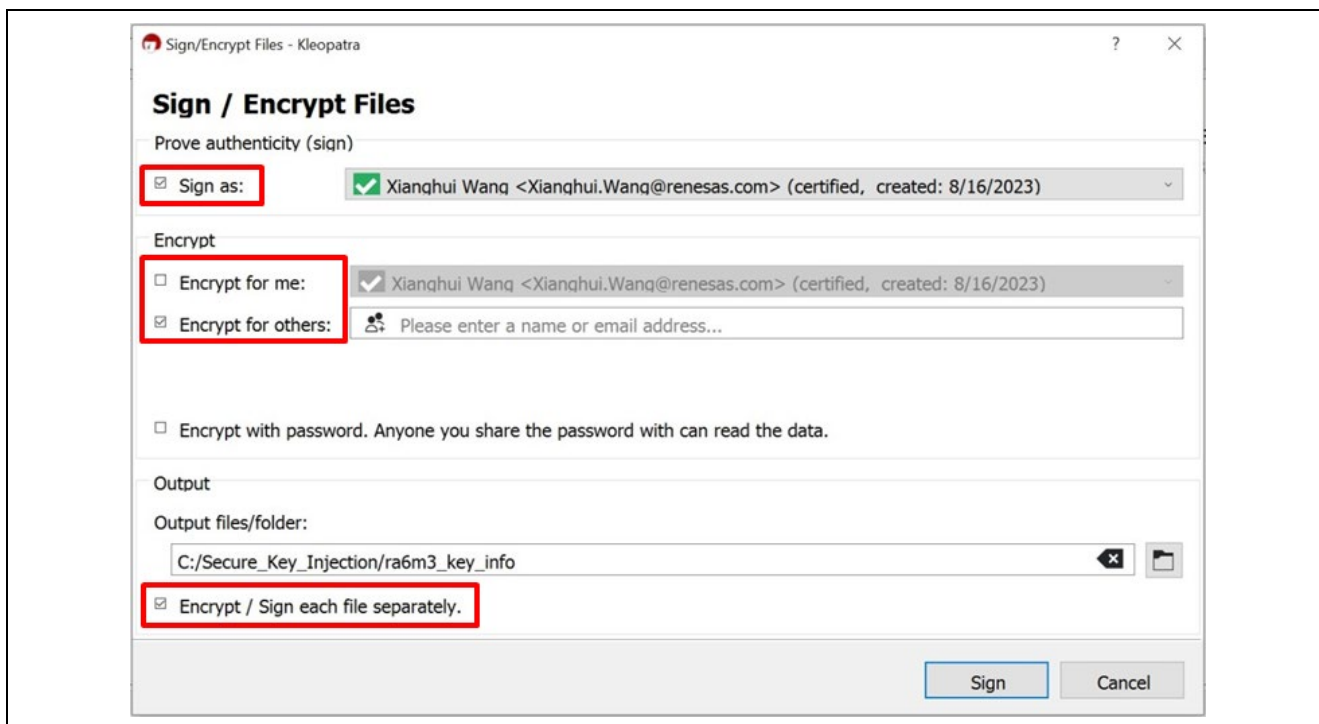


Figure 4-8 Selecting PGP Encryption Options

4. Click on the **Open selection dialog** (👤) icon for **Encrypt for others**. This will open a dialog box for selecting a certificate.

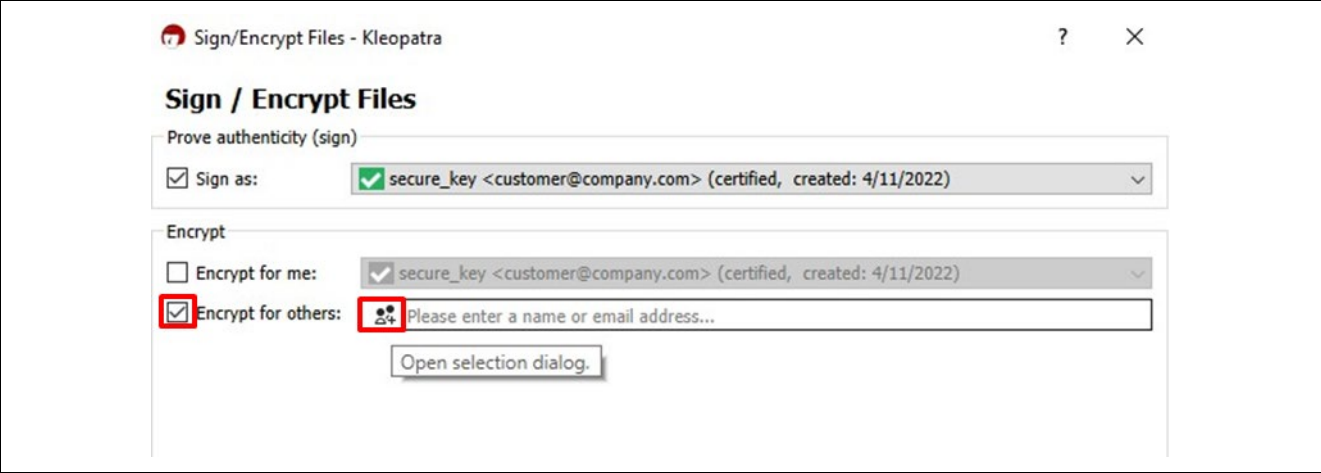


Figure 4-9 Opening the Certificate Selection Dialog Box

5. In this dialog box, select **keywrap** to choose the Renesas public key and then click on **OK**.

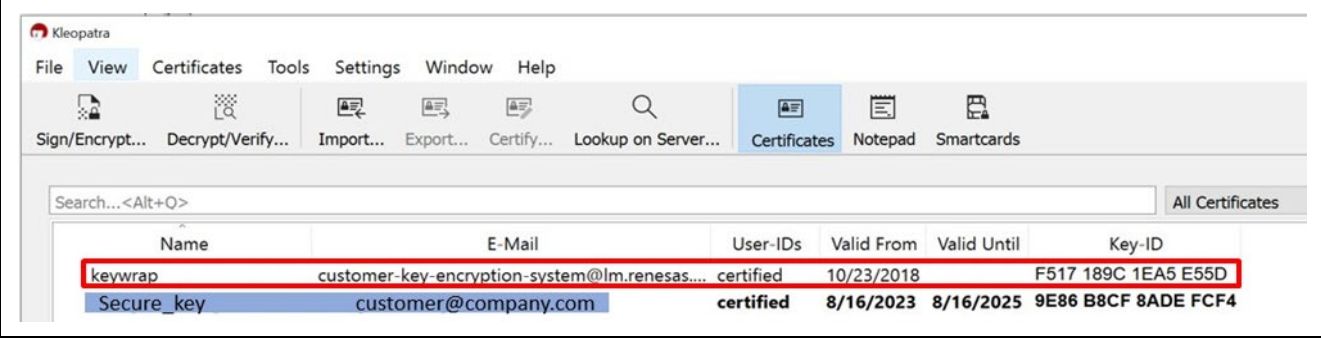


Figure 4-10 Selecting the Renesas PGP Public Key

6. Ensure that the correct folder where the encrypted key is to be stored is selected under **Output**. Finally, click on **Sign/Encrypt**. It is a good practice to keep the UFPKs and W-UFPKs for different MCU families in different folders and with distinctive names.

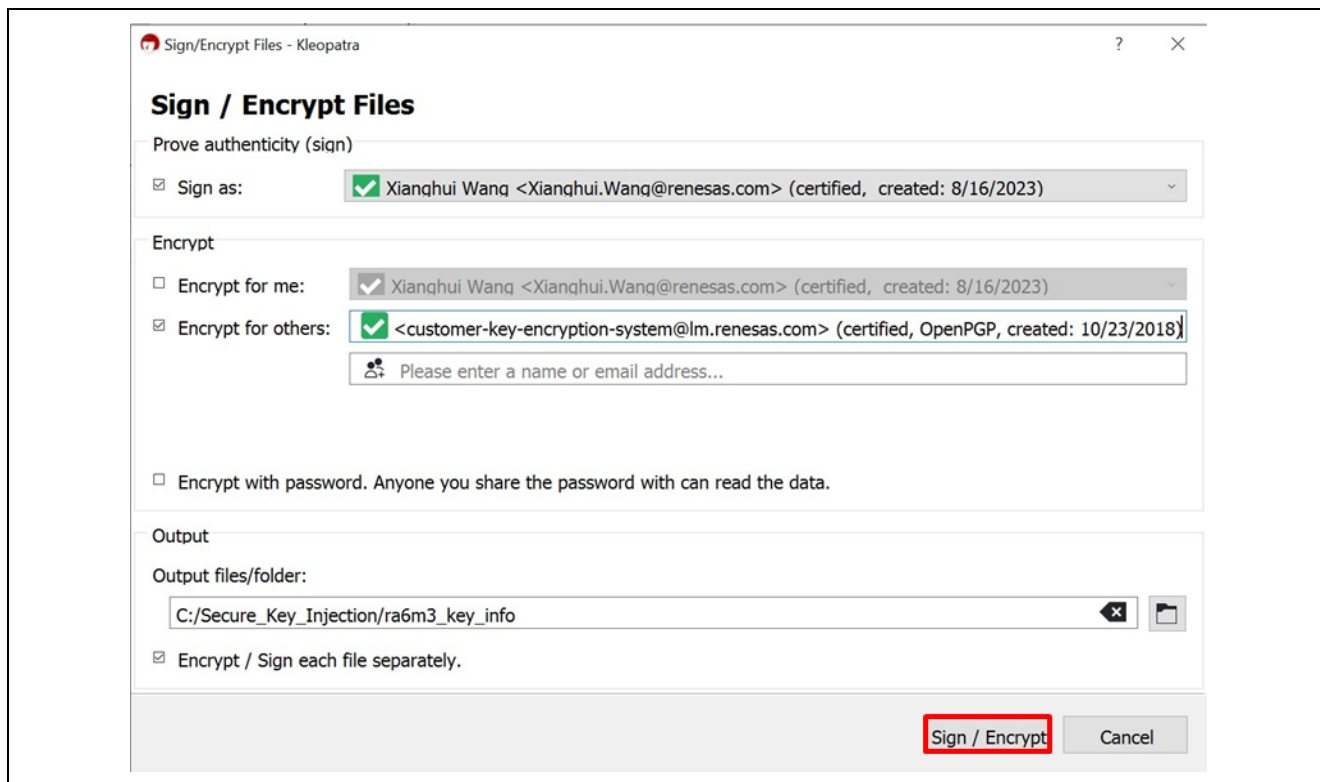


Figure 4-11 Encrypting the UFPK with the Renesas PGP Public Key

7. If you have not selected **Encrypt for me**, you will get an **Encrypt-To-Self Warning** to state that you will be unable to decrypt the data. Ignore the warning and click on **Continue** if you will not need to decrypt the data.

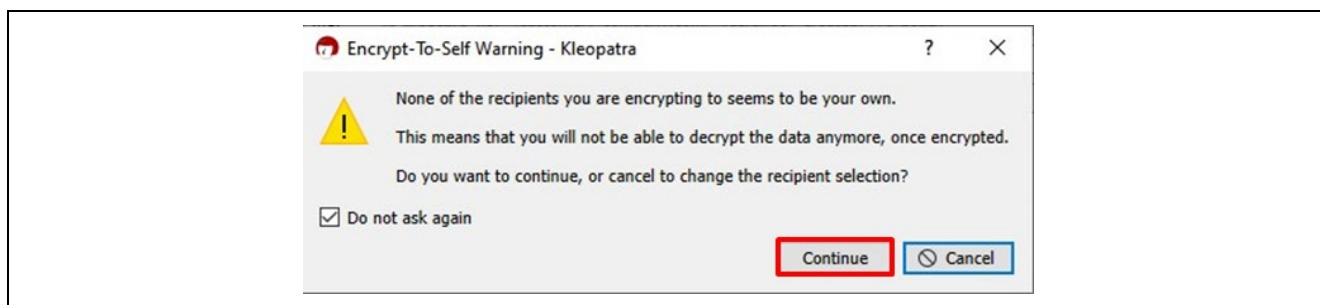


Figure 4-12 Starting the UFPK Encryption Process

8. The UFPK encrypted with the Renesas public key will be generated with `.gpg` added as the extension of the key file name. In this example, the `rx74m_ufpk.key.gpg` file is generated. Click on **Finish**.

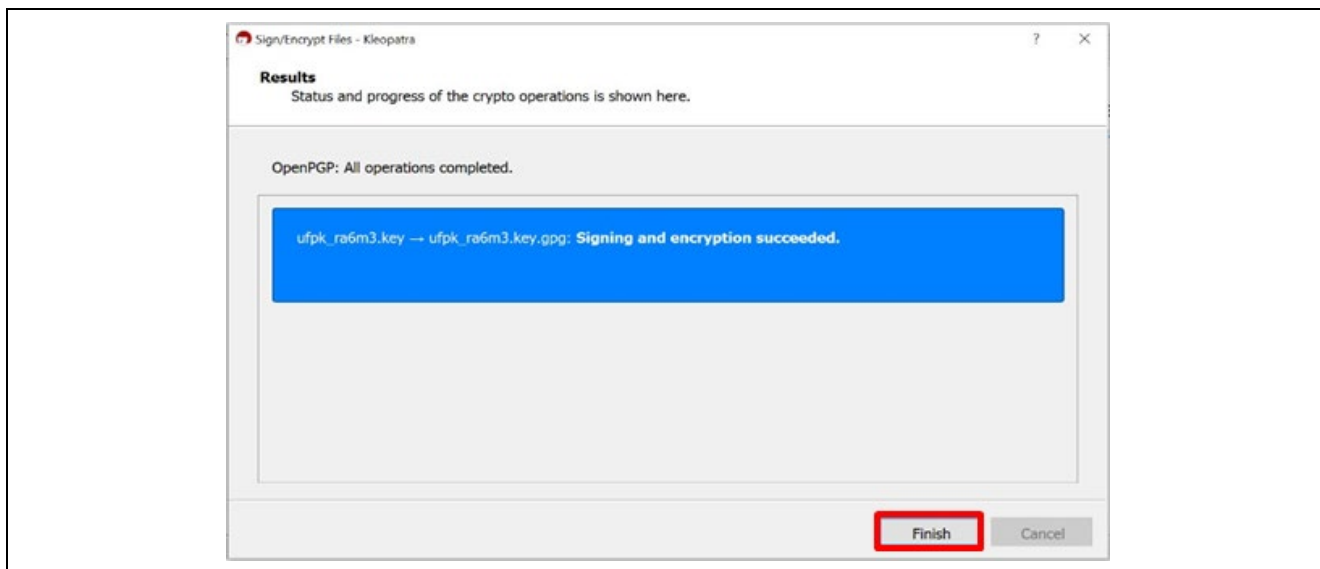


Figure 4-13 Encrypted Key was Generated

4.2.2 Obtaining a W-UFPK

1. Now, we can send the UFPK encrypted with the Renesas public key to the Renesas Key Wrap server to have it wrapped. Return to the Renesas Key Wrap server Web page and select **Renesas RX**.

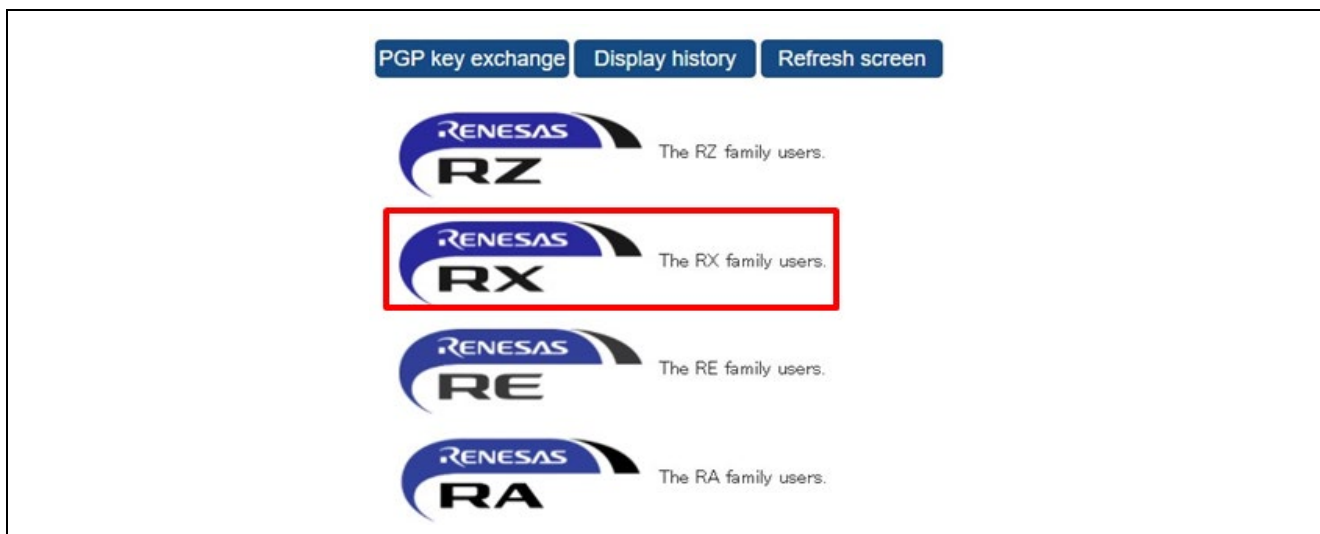


Figure 4-14 Selecting the MCU Family

When generating a W-UFPPK, it is important to select the correct MCU family and security engine mode.

Note: A W-UFPPK generated for compatibility mode cannot be used for DLM keys or key injection in protected mode and vice versa.

- To generate a W-UFPPK for the sample project for injecting secure keys in RX74M protected mode, select **Renesas RX Family** and click on **RX74M Wrap UFPPK** under **RSIP Protected Mode**.

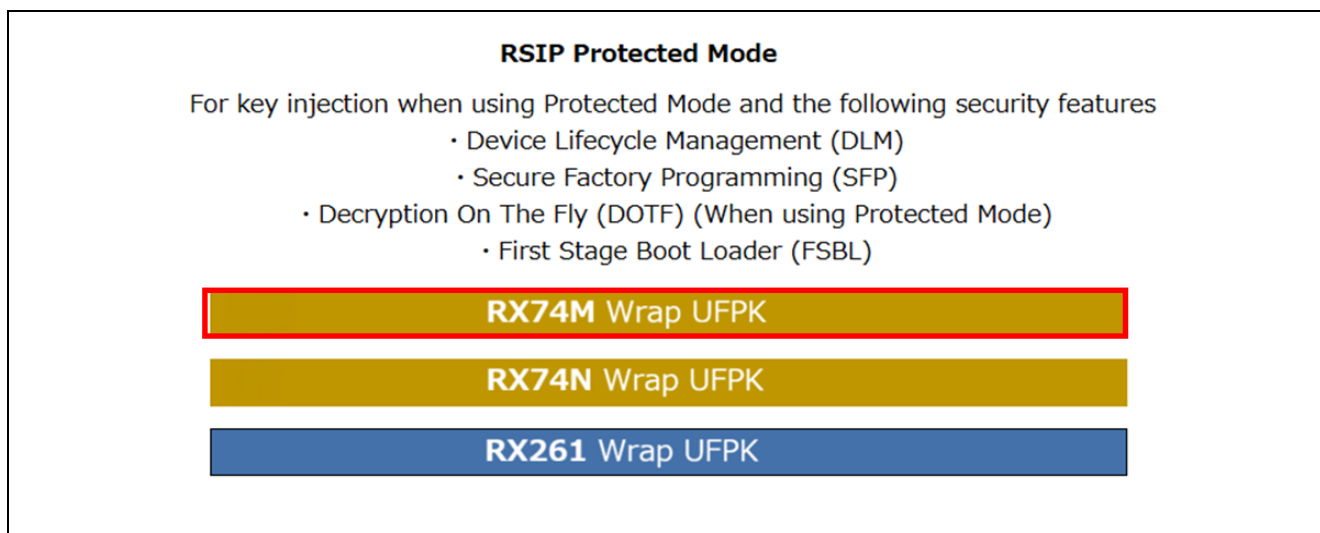


Figure 4-15 Selecting the RX74M Family DLM and Protected Mode

- To generate a W-UFPPK for the sample project for injecting secure keys in RX74M compatibility mode, select **Renesas RX Family** and click on **RX74M Wrap UFPPK** under **RSIP Compatibility Mode**.

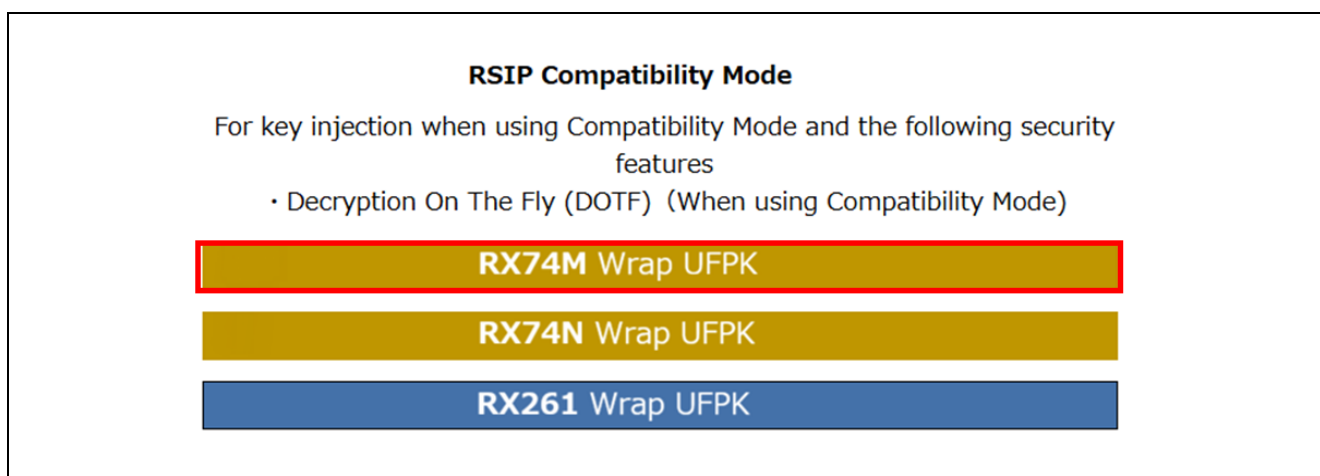


Figure 4-16 Selecting RX74M Family Compatibility Mode

- Click on **Encryption service for products** on the next screen.

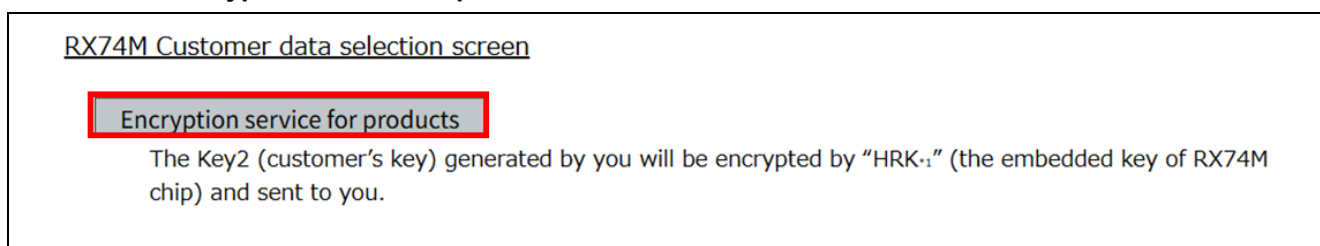


Figure 4-17 Selecting Encryption Service for Products

3. Click on **Browse** and select the UFPK encrypted with the Renesas public key. For example, select `rx74m_ufpk.key.gpg` that was previously generated and click on **Open**. Note that in the descriptions displayed on the Renesas Key Wrap service screen, **Key2** refers to a UFPK.

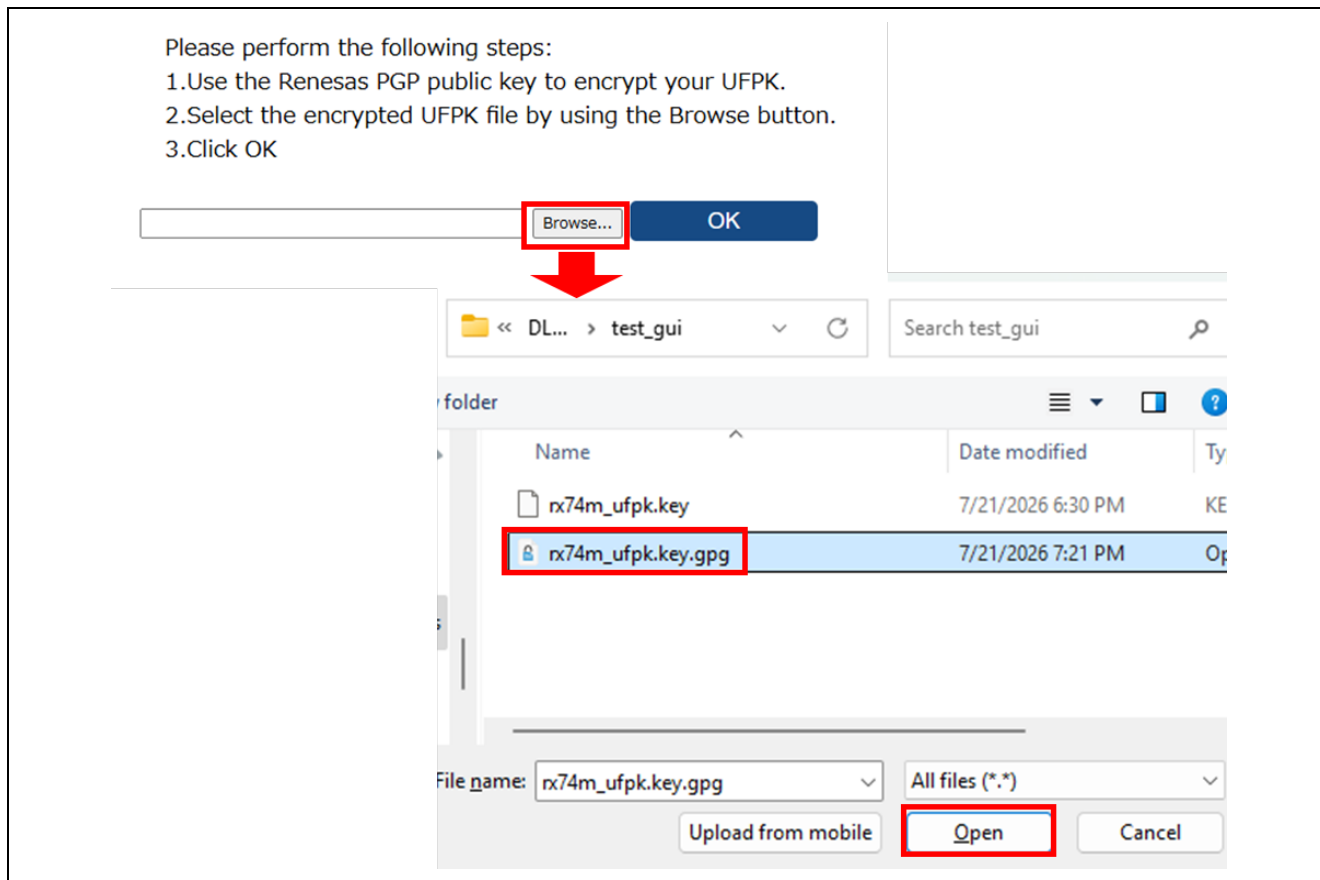


Figure 4-18 Selecting the UFPK File Encrypted with PGP

4. Click on **Settle** and the following message will appear. Then click on **Return to the menu**. You can now log out of the Renesas Key Wrap service.



Figure 4-19 Returning to the Main Menu of the Renesas Key Wrap Service

5. The W-UFPK encrypted with the PGP public key will usually arrive via E-mail in 1 to 2 minutes. Save the attached file.

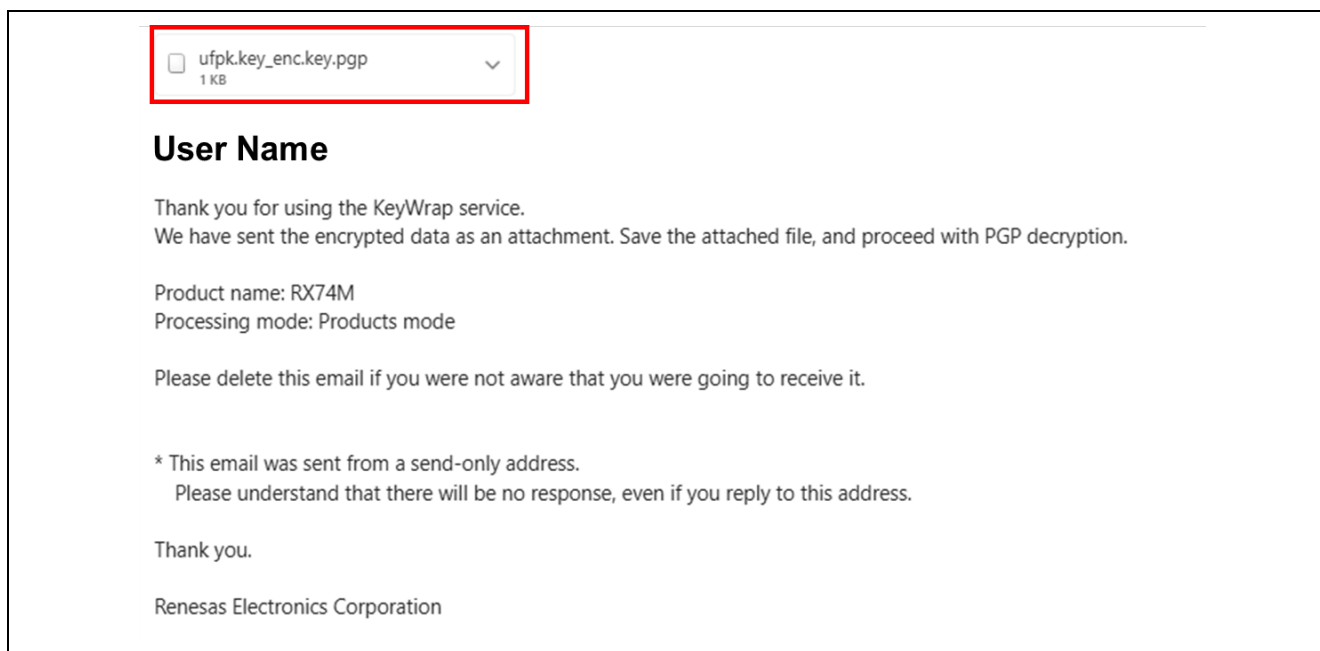


Figure 4-20 Receiving the W-UFPK via E-mail

Note: The obtained W-UFPK corresponds to the target MCU family and security engine mode. The W-UFPK obtained for compatibility mode cannot be used in protected mode and the W-UFPK obtained for protected mode cannot be used in compatibility mode.

4.2.3 Decrypting the W-UFPK

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

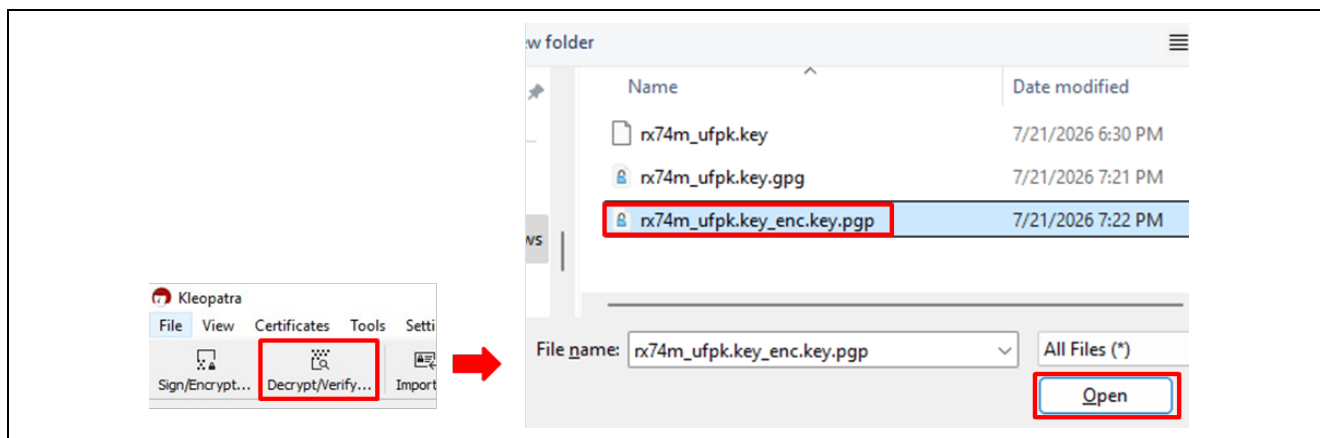


Figure 4-21 Decrypting the W-UFPK

2. Follow the prompt to enter the PGP private key passphrase and click on **OK**. The decrypted W-UFPK will be generated in the specified folder.

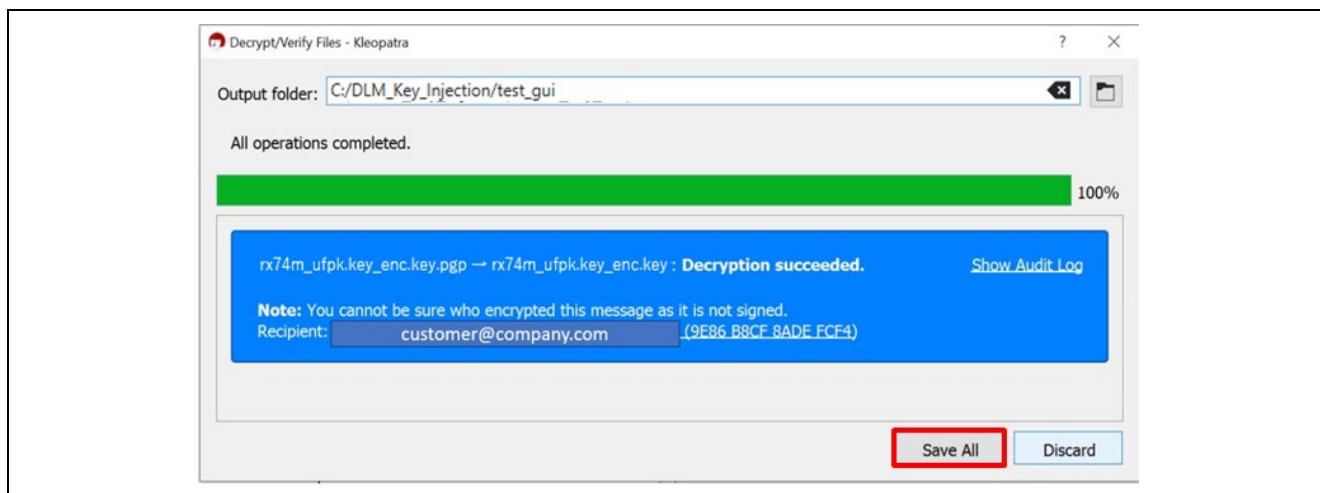


Figure 4-22 Decrypting the W-UFPK Encrypted with PGP

3. Click on **Save All** to save the decrypted W-UFPK key file `rx74m_ufpk.key_enc.key`. Both the UFPK and W-UFPK files are used in the generation of key injection data, which is described in the next sections.

5. Secure Key Injection and Updating in RSIP Protected Mode and the Sample Project

5.1 Overview

This section describes the procedures for secure key injection and updating in RSIP-E50D protected mode and a sample project for using the keys.

Only secure keys can be injected in protected mode. The MCU boot firmware interface should be used for injection of the initial user keys and KUK. To update a user key in the field, a previously injected KUK is used to wrap a new user key and FSP Crypto API functions are used for the execution of key updating.

Step-by-step instructions for the following procedures are given in this section.

- Wrapping of an ECC key pair and KUK through the SKMT GUI
- Wrapping of an ECC key pair and KUK through the SKMT CLI
- Injection of the initial user keys and KUK through the MCU boot firmware interface
- Execution of the sample project in protected mode
- Replacing of the source files when custom keys are to be used

This application note provides an example of user key wrapping for an ECC secp256r1 key pair (private and public keys) in RSIP-E50D protected mode of the RX74M.

The sample project in this application note uses the NIST test vectors as examples. To check the test vectors, download the related files from the following link.

[Cryptographic Algorithm Validation Program | CSRC \(nist.gov\)](https://csrc.nist.gov/ cryptographic-algorithm-validation-program)

5.2 Wrapping Keys with the UFPK and W-UFPK through the SKMT GUI

5.2.1 Wrapping the Initial ECC Key Pair with the UFPK

To wrap an ECC public key, make sure that **RX Family**, **RSIP-E50D Security Functions**, and **Protected Mode** is selected on the **Overview** tabbed page of the SKMT GUI. Next, open the **Wrap Key** tabbed page and set **Key Type** to **ECC** and **secp256r1, public** as shown in Figure 5-1.

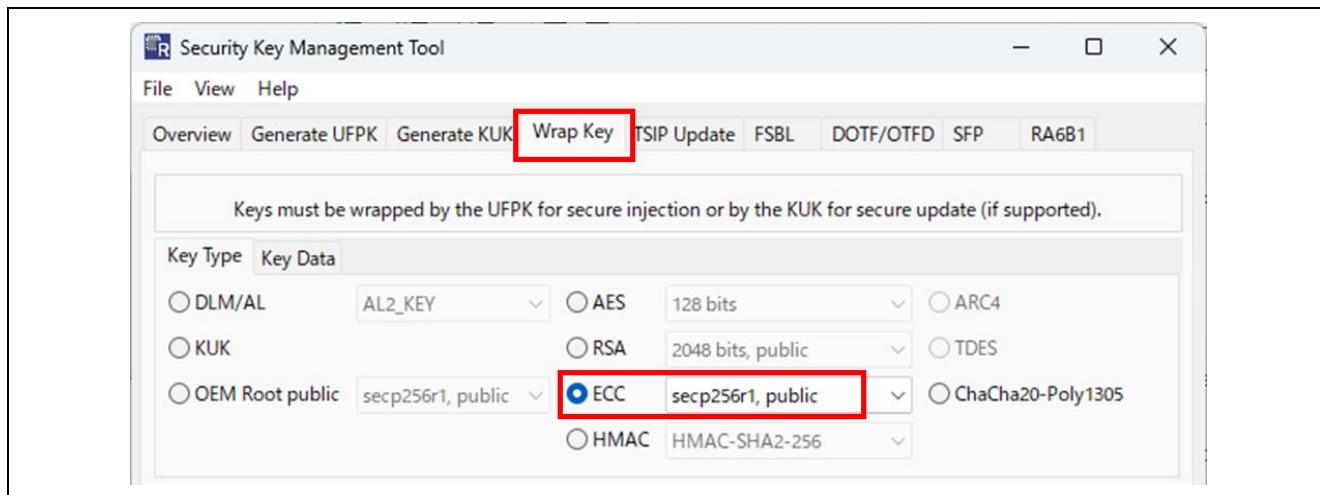


Figure 5-1 Selecting the secp256r1 Public Key

Next, configure the key data. On the **Key Data** tabbed page, select **Raw** and specify **Qx** and **Qy** as shown below. You can easily copy and paste the following data through the GUI.

Qx = 1ccb91c075fc7f4f033bfa248db8fccd3565de94bbfb12f3c59ff46c271bf83

Qy = ce4014c68811f9a21a1fdb2c0e6113e06db7ca93b7404e78dc7ccd5ca89a4ca9

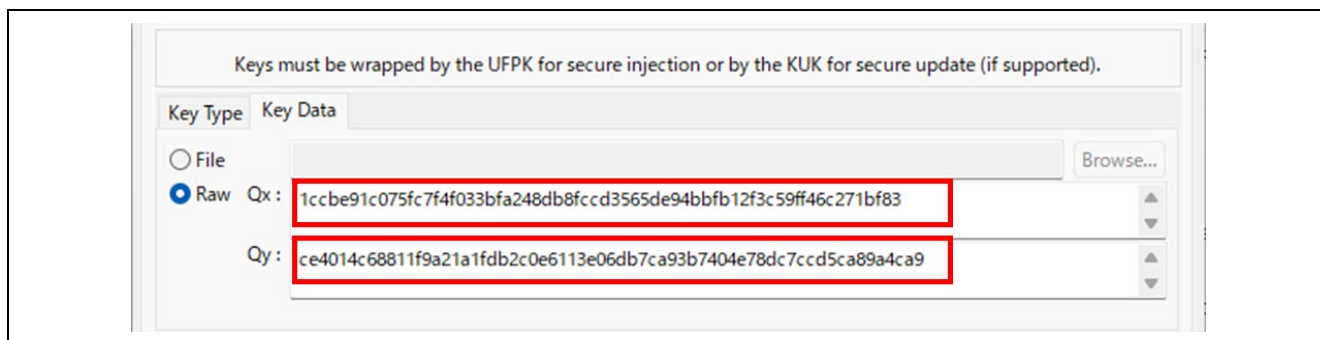


Figure 5-2 Specifying the ECC Public Key Data

Under **Wrapping Key**, click on the **Browse** buttons corresponding to the UFPK and W-UFPK and select the key pair generated in sections 4.1.2 and 4.2. For **IV**, select **Generate random value**. For the **Format** option, select **RFP**. After that, click on the **Browse** button to choose the output folder and name the output file. Click on **Generate File** and a *.rkey file will be generated.

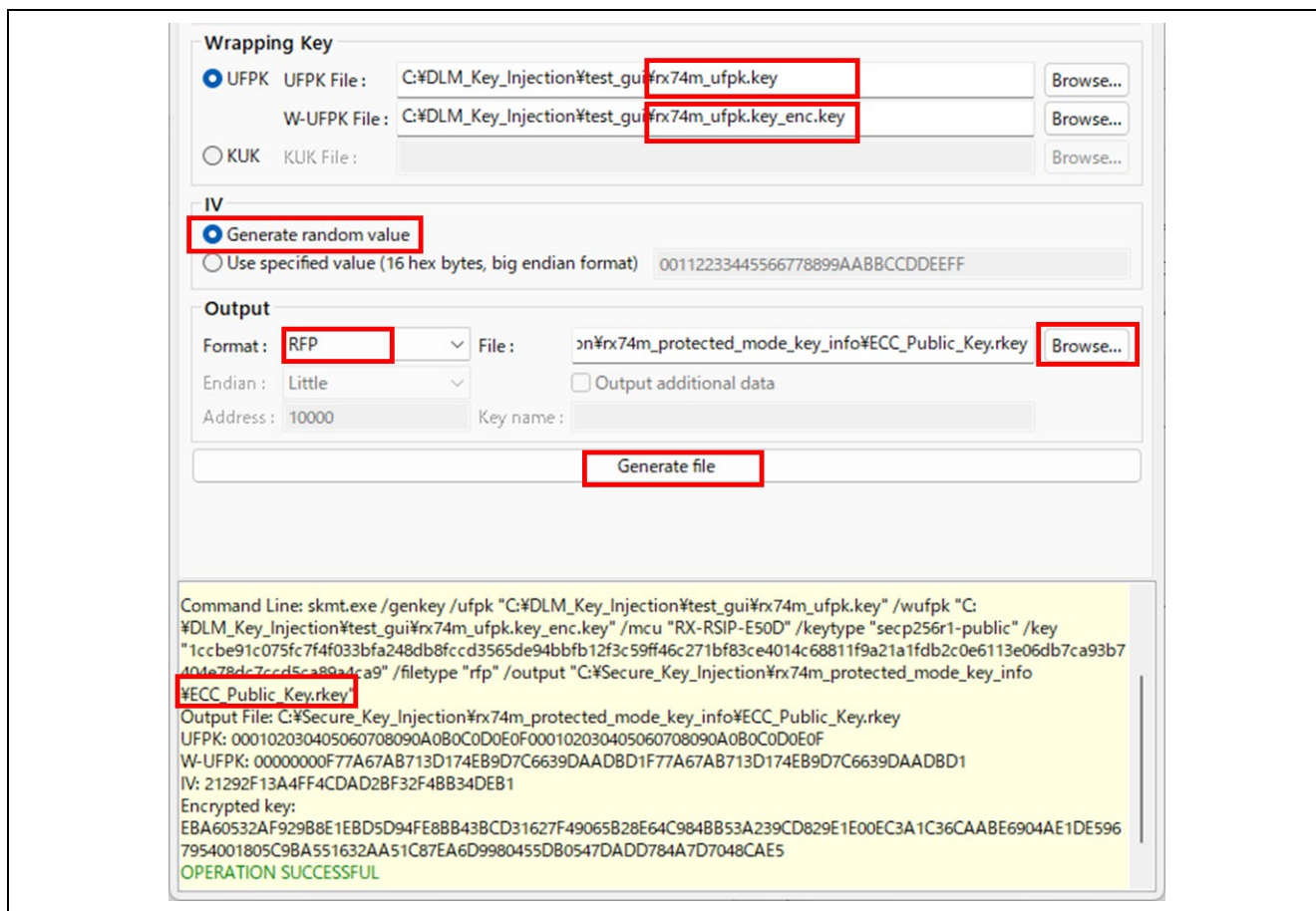


Figure 5-3 Generating the RFP Key Injection File for the ECC Public Key through the GUI

The *.rkey file does NOT contain the plaintext ECC key and UFPK, enabling confidential transfer of the key injection file.

Next, follow the above procedure in the same way to wrap the private key corresponding to the wrapped public key with the UFPK. On the **Wrap Key** tabbed page, set **Key Type** to **ECC** and **secp256r1, private** as shown in Figure 5-4.

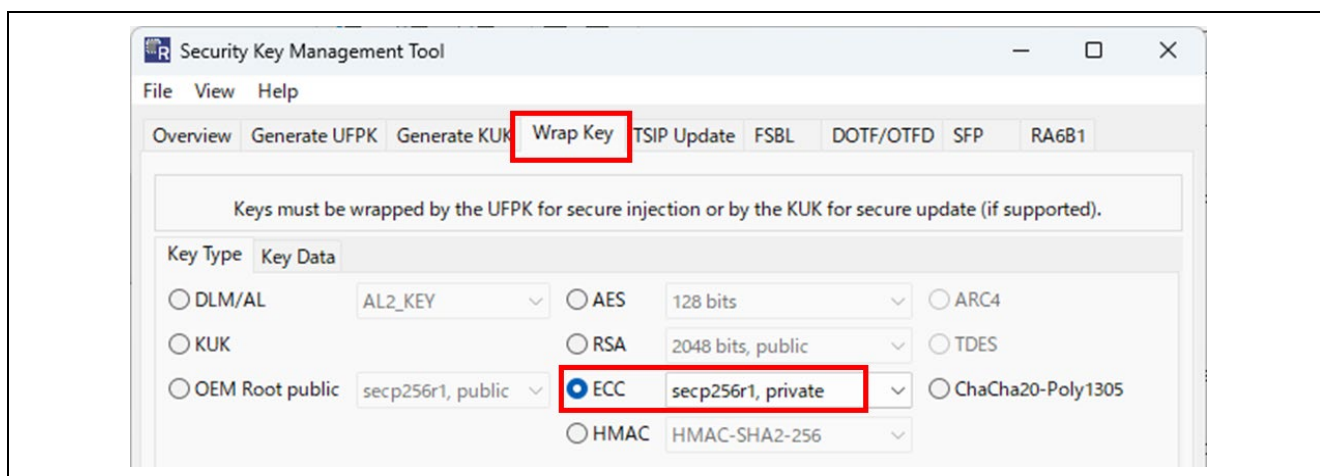


Figure 5-4 Selecting the secp256r1 Private Key

Next, configure the key data. On the **Key Data** tabbed page, select **Raw** and specify **d** as shown below. You can easily copy and paste the following key data through the GUI.

d = 519b423d715f8b581f4fa8ee59f4771a5b44c8130b4e3eacca54a56dda72b464

Figure 5-5 Specifying the ECC Private Key Data

Under **Wrapping Key**, click on the **Browse** buttons corresponding to the UFPK and W-UFPK and select the key pair generated in sections 4.1.2 and 4.2. For **IV**, select **Generate random value**. For the **Format** option, select **RFP**. After that, click on the **Browse** button to choose the output folder and name the output file. Click on **Generate file** and a *.rkey file will be generated.

```

Command Line: skmt.exe /genkey /ufpk "C:\DLM_Key_Injection\test_gui\rx74m_ufpk.key" /wufpk "C:\DLM_Key_Injection\test_gui\rx74m_ufpk.key_enc.key" /mcu "RX-RSIP-E50D" /keytype "secp256r1-private" /key "519b423d715f8b581f4fa8ee59f4771a5b44c8130b4e3eacca54a56dda72b464" /filetype "rfp" /output "C:\Secure_Key_Injection\rx74m_protected_mode_key_info\ECC_Private_Key.rkey"
Output File: C:\Secure_Key_Injection\rx74m_protected_mode_key_info\ECC_Private_Key.rkey
UFPK: 000102030405060708090A0B0C0D0E0F000102030405060708090A0B0C0D0E0F
W-UFPK: 00000000F77A67AB713D174EB9D7C6639DAADB8D1F77A67AB713D174EB9D7C6639DAADB8D1
IV: 9D9BA0F4B980E422923694CDF85BE5A8
Encrypted key:
0EAAF98A39E51AD7637BA642369A13B4C88DB0264E0A17CD0EDFD8834A536D0BCD614E222B834ECF02DC0CFDDE0A0FBA
OPERATION SUCCESSFUL
  
```

Figure 5-6 Generating the RFP Key Injection File for the ECC Private Key through the GUI

The *.rkey file does NOT contain the plaintext ECC key and UFPK, enabling confidential transfer of the key injection file.

5.2.2 Wrapping a KUK with the UFPK

The SKMT can be used to generate a sample KUK. To generate a KUK file, open the **Generate KUK** tabbed page, select **Use specified value**, and enter the following value.

```
000102030405060708090a0b0c0d0e0f000102030405060708090a0b0c0d0e0f
```

Click on the **Browse** button to select the output folder and file name for the key file to be generated; here `kuk_for_new_key.key` is specified as the file name. Then, click on **Generate KUK key file** and the `kuk_for_new_key.key` file will be generated in the selected folder.

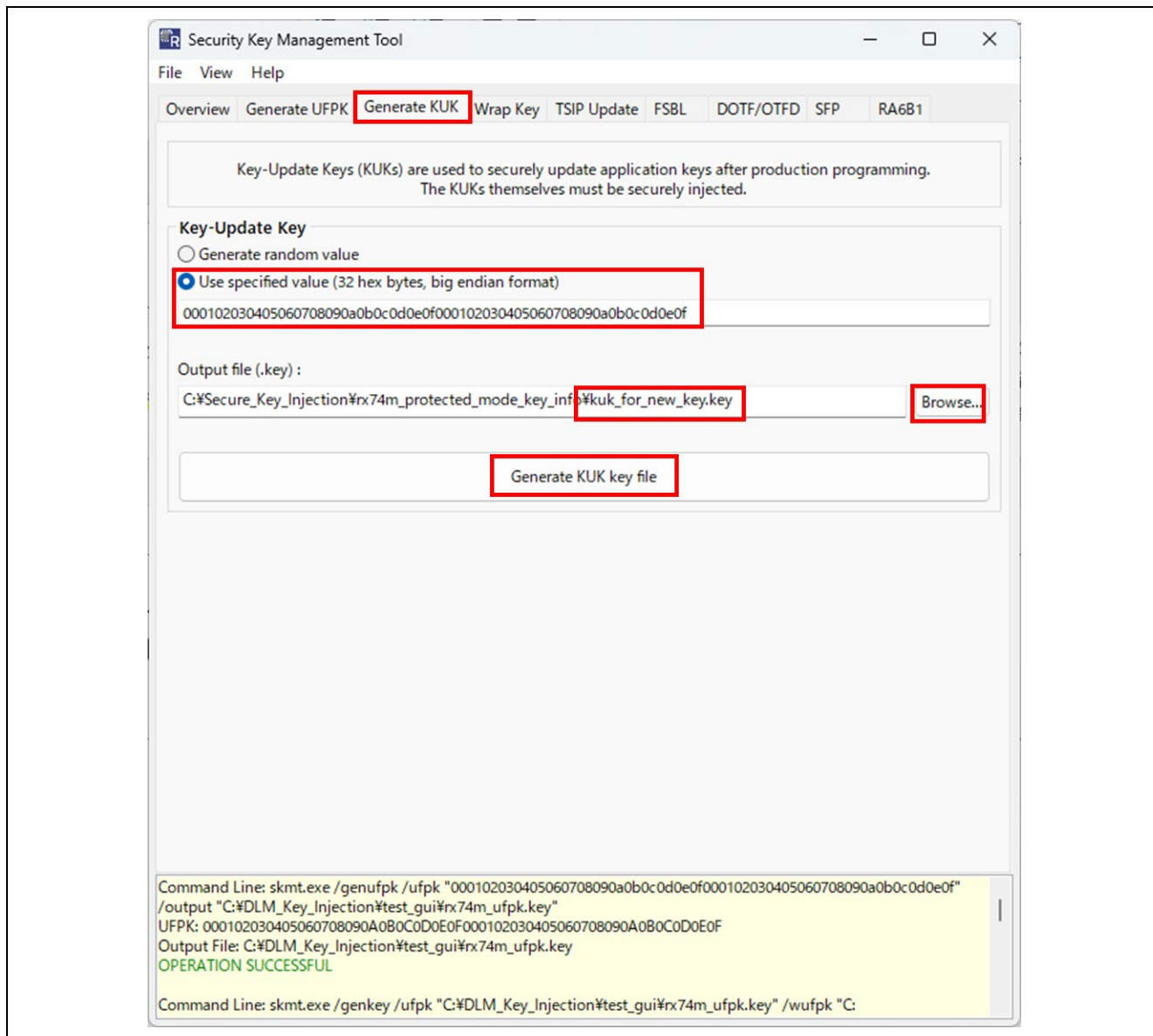


Figure 5-7 Generating the KUK File for Encryption of the User Keys for the RSIP-E50D

Wrap the KUK such that it can be injected into the MCU. Open the **Wrap Key** tabbed page and choose **KUK** for **Key Type**.

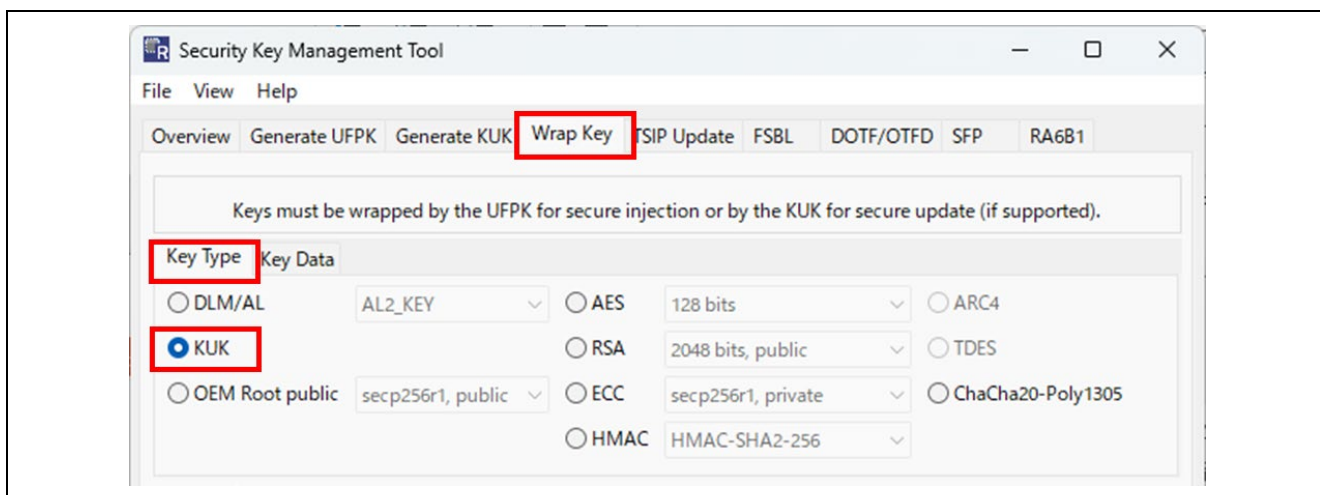


Figure 5-8 Selecting KUK for Wrapping

Open the **Key Data** tabbed page, select the **File** option, and browse to find the `kuk_for_new_key.key` file generated in Figure 5-7.

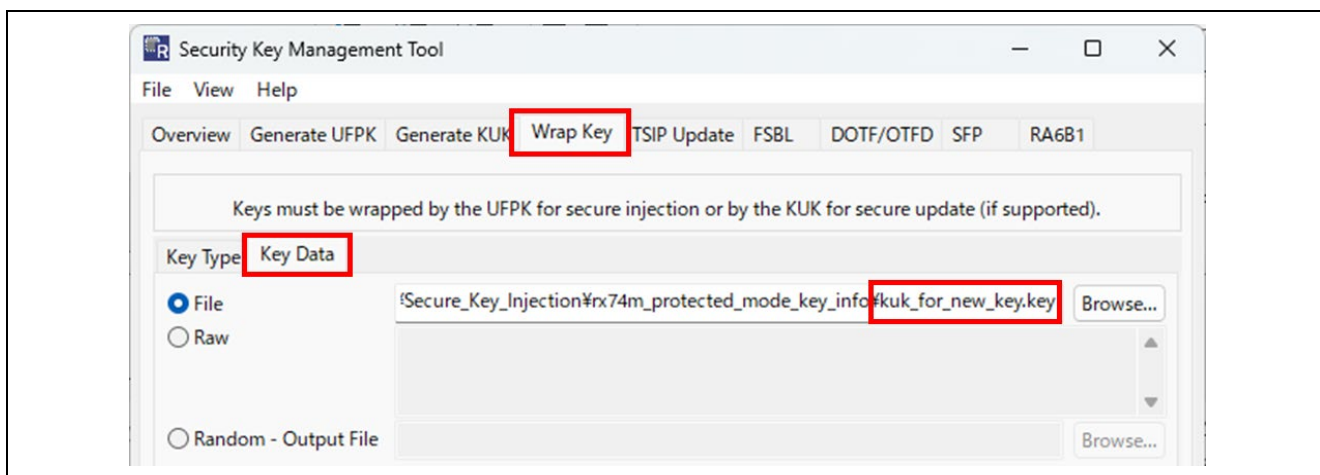


Figure 5-9 Specifying the KUK .key File

Under **Wrapping Key**, click on the **Browse** buttons corresponding to the UFPK and W-UFPK and select the key pair generated in sections 4.1.2 and 4.2. For **IV**, select **Generate random value**. For the **Format** option, select **RFP**. After that, click on the **Browse** button to choose the output folder and name the output file.

Next, click on **Generate file** and the KUK.rkey file will be generated.

Wrapping Key

☒ UFPK UFPK File: C:\DLM_Key_Injection\test_gui\rx74m_ufpk.key Browse...

W-UFPK File: C:\DLM_Key_Injection\test_gui\rx74m_ufpk.key_enc.key Browse...

☐ KUK KUK File: Browse...

IV

☒ Generate random value

☐ Use specified value (16 hex bytes, big endian format) 00112233445566778899AABBCCDDEEFF

Output

Format: RFP File: Key_Injection\rx74m_protected_mode_key_info\KUK.rkey Browse...

Endian: Little ☐ Output additional data

Address: 10000 Key name:

Generate file

Command Line: skmt.exe /genkey /ufpk "C:\DLM_Key_Injection\test_gui\rx74m_ufpk.key" /wufpk "C:\DLM_Key_Injection\test_gui\rx74m_ufpk.key_enc.key" /mcu "RX-RSIP-E50D" /keytype "key-update-key" /key file="C:\Secure_Key_Injection\rx74m_protected_mode_key_info\kuk_for_new_key.key" /filetype "rfp" /output "C:\Secure_Key_Injection\rx74m_protected_mode_key_info\KUK.rkey"

Output File: C:\Secure_Key_Injection\rx74m_protected_mode_key_info\KUK.rkey

UFPK: 000102030405060708090A0B0C0D0E0F000102030405060708090A0B0C0D0E0F

W-UFPK: 00000000F77A67AB713D174EB9D7C6639DAADBD1F77A67AB713D174EB9D7C6639DAADBD1

IV: 620D280E3B380EED68A2B0F3C7433B6

Encrypted key:

443776A3729F4D80C54D67AD14EBB3B936BFC35EF4EEC5ACC6424A3B9C8C2B5EA5B2FFDC80398B20A00D6B00B209490A

OPERATION SUCCESSFUL

Figure 5-10 Generating the KUK Injection File through the RSIP-E50D GUI

The *.rkey file does NOT contain the plaintext KUK and UFPK, enabling confidential transfer of the key injection file.

5.2.3 Wrapping a New ECC Key Pair with the KUK

This section describes the procedure for wrapping a new ECC key pair by using `kuk_for_new_key.key` generated in Figure 5-7.

Use the SKMT GUI and select **RX Family, RSIP-E50D Security Functions, and Protected Mode** on the **Overview** tabbed page. Then open the **Wrap Key** tabbed page. Set **Key Type** to **ECC** and **secp256r1, public** as shown in Figure 5-1

On the **Key Data** tabbed page, select **Raw** and specify **Qx** and **Qy** as shown below. You can easily copy and paste the following key data through the GUI.

Qx = e266ddfdc12668db30d4ca3e8f7749432c416044f2d2b8c10bf3d4012aeffa8a

Qy = bfa86404a2e9ffe67d47c587ef7a97a7f456b863b4d02cfc6928973ab5b1cb39

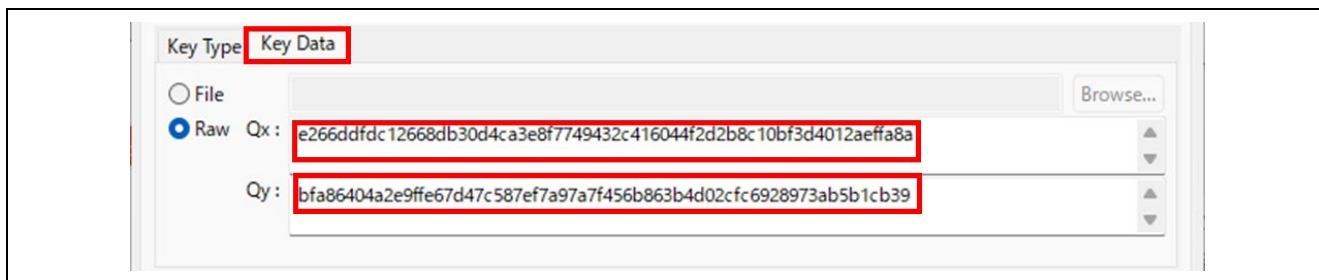


Figure 5-11 Specifying the New ECC Public Key Data

Under **Wrapping Key**, click on the **Browse** button corresponding to the KUK and select the KUK generated in section 5.2.2. For **IV**, select **Generate random value**. For the **Format** option, select **C Source**, and name the output file as `new_ecc_public_key.c`. Set **Key name** to `NEW_ECC_PUB`.

Finally, click on **Generate file** and both the `new_ecc_public_key.c` and `new_ecc_public_key.h` files will be generated.

Wrapping Key

☐ UFPK UFPK File: C:\DLM_Key_Injection\test_gui\rx74m_ufpk.key Browse...

W-UFPK File: C:\DLM_Key_Injection\test_gui\rx74m_ufpk.key_enc.key Browse...

☒ KUK KUK File: C:\Secure_Key_Injection\rx74m_protected_mode_key_info\kuk_for_new_key.key Browse...

IV

☒ Generate random value

☐ Use specified value (16 hex bytes, big endian format) 00112233445566778899AABBCCDDEEFF

Output

Format: C Source File: \rx74m_protected_mode_key_info\new_ecc_public_key.c Browse...

Endian: Little ☐ Output additional data

Address: 10000 Key name: NEW_ECC_PUB

Generate file

Command Line: skmt.exe /genkey /kuk file="C:\Secure_Key_Injection\rx74m_protected_mode_key_info\kuk_for_new_key.key" /mcu "RX-RSIP-E50D" /keytype "secp256r1-public" /key "e266ddfdc12668db30d4ca3e8f7749432c416044f2d2b8c10bf3d4012aeffa8abfa86404a2e9ffe67d47c587ef7a97a7f456b863b4d02cfc6928973ab5b1cb39" /filetype "csource" /keyname "NEW_ECC_PUB" /output "C:\Secure_Key_Injection\rx74m_protected_mode_key_info\new_ecc_public_key.c"

Output File: C:\Secure_Key_Injection\rx74m_protected_mode_key_info\new_ecc_public_key.h

Output File: C:\Secure_Key_Injection\rx74m_protected_mode_key_info\new_ecc_public_key.c

KUK: 000102030405060708090A0B0C0D0E0F000102030405060708090A0B0C0D0E0F

IV: 1DE66320E33A437EF88EB468EE125788

Encrypted key: 543442A8429FAE9C2680BC48D668700D4135A19488449ECF7449AE44376223A4CDAAA4E8DD1E310AE23C48D22D384FA1E50311F00E92E591389B42E53DAFE586DBC6CB063783F991F105736330CD7BC7

OPERATION SUCCESSFUL

Figure 5-12 Generating the ECC Public Key Wrapped with the KUK

Follow the above procedure in the same way to wrap the private key corresponding to the new wrapped public key with the KUK. On the **Wrap Key** tabbed page, set **Key Type** to **ECC** and **secp256r1, private** as shown in Figure 5-4.

Next, configure the key data. On the **Key Data** tabbed page, select **Raw** and specify **d** as shown below. You can easily copy and paste the following key data through the GUI.

`d = 0f56db78ca460b055c500064824bed999a25aaf48ebb519ac201537b85479813`

Key Type **Key Data**

☐ File Browse...

☒ Raw

0f56db78ca460b055c500064824bed999a25aaf48ebb519ac201537b85479813

☐ Random - Output File Browse...

Figure 5-13 Specifying the New ECC Private Key Data

Under **Wrapping Key**, click on the **Browse** button corresponding to the KUK and select the KUK generated in section 5.2.2. For **IV**, select **Generate random value**. For the **Format** option, select **C Source** and name the output file as `new_ecc_private_key.c`. Set **Key name** to `NEW_ECC_PRI`.

Finally, click on **Generate file** and both the `new_ecc_private_key.c` and `new_ecc_private_key.h` files will be generated.

Wrapping Key

☐ UFPK UFPK File: C:\DLM_Key_Injection\test_gui\rx74m_ufpk.key Browse...

W-UFPK File: C:\DLM_Key_Injection\test_gui\rx74m_ufpk.key_enc.key Browse...

☒ KUK KUK File: C:\Secure_Key_Injection\rx74m_protected_mode_key_info\kuk_for_new_key.key Browse...

IV

☒ Generate random value

☐ Use specified value (16 hex bytes, big endian format) 00112233445566778899AABBCCDDEEFF

Output

Format: C Source File: \rx74m_protected_mode_key_info\new_ecc_private_key.c Browse...

Endian: Little ☐ Output additional data

Address: 10000 Key name: NEW_ECC_PRI

Generate file

Command Line: skmt.exe /genkey /kuk file="C:\Secure_Key_Injection\rx74m_protected_mode_key_info\kuk_for_new_key.key" /mcu "RX-RSIP-E50D" /keytype "secp256r1-private" /key "0f56db78ca460b055c500064824bed999a25aaf48ebb519ac201537b85479813" /filetype "csource" /keyname "NEW_ECC_PRI" /output "C:\Secure_Key_Injection\rx74m_protected_mode_key_info\new_ecc_private_key.c"

Output File: C:\Secure_Key_Injection\rx74m_protected_mode_key_info\new_ecc_private_key.h

Output File: C:\Secure_Key_Injection\rx74m_protected_mode_key_info\new_ecc_private_key.c

KUK: 000102030405060708090A0B0C0D0E0F000102030405060708090A0B0C0D0E0F

IV: 11D4432BA070659D30EA62C5B96FCDC9

Encrypted key:
465ED2EC54FBD2406469BB4F8B23F5363262E0CF06D88699C4DD31BB320568A32C03014EEAFB22ED18136DBF35AFA874

OPERATION SUCCESSFUL

Figure 5-14 Generating the ECC Private Key Wrapped with the KUK

5.3 Wrapping Keys with the UFPK and W-UFPK through the SKMT CLI

This section describes the procedures for wrapping keys through the SKMT CLI.

The `/genkey` command of the SKMT command-line tool `skmt.exe` is used to prepare keys for secure injection and updating. The options used with this command are listed below.

- `/keytype:` This option can take either an ASCII string or one-byte hexadecimal value as an input parameter to indicate the key type.
- `/ufpk:` User factory programming key
- `/wufpk:` UFPK wrapped with the Renesas HRK
- `/kuk:` Key update key for secure key updating
- `/mcu:` Target MCU and security engine
- `/output:` Output of the command

Refer to the SKMT user's manual for more information about these commands, including the valid values for the individual parameters.

In this application note, an ECC secp256r1 key pair is used in describing the secure key injection and updating processes.

The examples in this section use the UFPK and W-UFPK generated earlier.

5.3.1 Wrapping the Initial ECC Key Pair with the UFPK

In this section, the same ECC key pair as that shown in Figure 5-2 and Figure 5-5 is used in describing an example of preparing an ECC key pair for secure key injection.

In the Command Prompt window opened earlier (section 4.1.3), use the following command to generate an ECC public key injection file (ECC_Public_Key_CLI.rkey). Refer to the SKMT user's manual for details on how to construct the command.

```
skmt.exe /genkey /ufpk
file="C:\Secure_Key_Injection\rx74m_protected_mode_key_info\rx74m_ufpk.key"
/wufpk
file="C:\Secure_Key_Injection\rx74m_protected_mode_key_info\rx74m_ufpk.key_enc.k
ey" /mcu "RX-RSIP-E50D" /keytype "secp256r1-public" /key
"1ccb91c075fc7f4f033bfa248db8fcd3565de94bbfb12f3c59ff46c271bf83ce4014c68811f9a
21a1fdb2c0e6113e06db7ca93b7404e78dc7ccd5ca89a4ca9" /filetype "rfp" /output
"C:\Secure_Key_Injection\rx74m_protected_mode_key_info\ECC_Public_Key_CLI.rkey"
```

Note the following in this example.

- 1ccb91c075fc7f4f033bfa248db8fcd3565de94bbfb12f3c59ff46c271bf83ce4014c68811f9a21a1fdb2c0e6113e06db7ca93b7404e78dc7ccd5ca89a4ca9 is the ECC public key of the NIST test vector.
- The key type is specified as "secp256r1-public".
- "RX-RSIP-E50D" is used for the /mcu option.
- A randomly generated initialization vector (IV) is used. The IV is updated with each instance of encryption.
- In this example, UFPK file and W-UFPK file are stored in C:\Secure_Key_Injection\rx74m_protected_mode_key_info folder.
- The command option /output defines the location and name of the output file.

```
Output File: C:\Secure_Key_Injection\rx74m_protected_mode_key_info\ECC_Public_Key_CLI.rkey
UFPK: 000102030405060708090A0B0C0D0E0F000102030405060708090A0B0C0D0E0F
W-UFPK: 1234567890ABCDEF1234567890ABCDEF1234567890ABCDEF1234567890ABCDEF12345678
IV: 0273B7277508F33491F2BA569B092535
Encrypted key:
1234567890ABCDEF1234567890ABCDEF1234567890ABCDEF1234567890ABCDEF1234567890ABCDEF
1234567890ABCDEF1234567890ABCDEF1234567890ABCDEF1234567890ABCDEF
```

Figure 5-15 Generating an ECC Public Key Injection File through the CLI

Use the following command to generate an ECC private key injection file (ECC_Private_Key_CLI.rkey).

```
skmt.exe /genkey /ufpk  
file="C:\Secure_Key_Injection\rx74m_protected_mode_key_info\rx74m_ufpk.key"  
/wufpk  
file="C:\Secure_Key_Injection\rx74m_protected_mode_key_info\rx74m_ufpk.key_enc.k  
ey" /mcu "RX-RSIP-E50D" /keytype "secp256r1-private" /key  
"519b423d715f8b581f4fa8ee59f4771a5b44c8130b4e3eacca54a56dda72b464" /filetype  
"rfp" /output  
"C:\Secure_Key_Injection\rx74m_protected_mode_key_info\ECC_Private_Key_CLI.rkey"
```

Note the following in this example.

- 519b423d715f8b581f4fa8ee59f4771a5b44c8130b4e3eacca54a56dda72b464 is the ECC private key of the NIST test vector.
- The key type is specified as "secp256r1-private".
- "RX-RSIP-E50D" is used for the /mcu option.
- A randomly generated IV is used. The IV is updated with each instance of encryption.
- The command option /output defines the location and name of the output file.

```
Output File: C:\Secure_Key_Injection\rx74m_protected_mode_key_info\ECC_Private_Key_CLI.rkey  
UFPK: 000102030405060708090A0B0C0D0E0F000102030405060708090A0B0C0D0E0F  
W-UFPK: 1234567890ABCDEF1234567890ABCDEF1234567890ABCDEF1234567890ABCDEF12345678  
IV: 0273B7277508F33491F2BA569B092535  
Encrypted key:  
1234567890ABCDEF1234567890ABCDEF1234567890ABCDEF1234567890ABCDEF1234567890ABCDEF
```

Figure 5-16 Generating an ECC Private Key Injection File through the CLI

5.3.2 Generating and Wrapping a KUK with the UFPK

The SKMT can be used to generate a KUK key file. Use the following command.

```
skmt.exe /genkuk /kuk
"000102030405060708090a0b0c0d0e0f000102030405060708090a0b0c0d0e0f" /output
"C:\Secure_Key_Injection\rx74m_protected_mode_key_info\kuk_for_new_key_cli.key"
```

Note the following in this example.

- The complete file path for the key file is specified.
- The same KUK as was used in section 5.2.2 must be used.

```
KUK: 000102030405060708090A0B0C0D0E0F000102030405060708090A0B0C0D0E0F
Output File:
C:\Secure_Key_Injection\rx74m_protected_mode_key_info\kuk_for_new_key_cli.key
```

Figure 5-17 Generating a KUK File

The generated key file `kuk_for_new_key_cli.key` now contains the KUK. Retain this key file and use it to wrap new user keys for secure key updating.

To enable secure key updating, you must first securely inject the KUK. Use the SKMT to wrap the KUK with the UFPK and generate a key injection file for use with the RFP through the following command.

```
skmt.exe /genkey /ufpk
file="C:\Secure_Key_Injection\rx74m_protected_mode_key_info\rx74m_ufpk.key"
/wufpk
file="C:\Secure_Key_Injection\rx74m_protected_mode_key_info\rx74m_ufpk.key_enc.k
ey" /mcu "RX-RSIP-E50D" /keytype "key-update-key" /key
file="C:\Secure_Key_Injection\rx74m_protected_mode_key_info\kuk_for_new_key_cli.
key" /filetype "rfp" /output
"C:\Secure_Key_Injection\rx74m_protected_mode_key_info\KUK_CLI.rkey"
```

Note the following in this example.

- The KUK file generated before is used.
- The key type is specified as "key-update-key".
- A randomly generated IV is used. The IV is updated each time this command is executed.
- The complete file path for the key file (`KUK_CLI.rkey`) is specified.

```
Output File: C:\Secure_Key_Injection\rx74m_protected_mode_key_info\KUK_CLI.rkey
UFPK: 000102030405060708090A0B0C0D0E0F000102030405060708090A0B0C0D0E0F
W-UFPK: 1234567890ABCDEF1234567890ABCDEF1234567890ABCDEF1234567890ABCDEF12345678
IV: 1234567890ABCDEF1234567890ABCDEF
Encrypted key:
1234567890ABCDEF1234567890ABCDEF1234567890ABCDEF1234567890ABCDEF1234567890ABCDEF
```

Figure 5-18 Generating a KUK Injection File through the CLI

The generated key file `KUK_CLI.rkey` now contains the wrapped KUK and W-UFPK. The `*.rkey` file does NOT contain the plaintext KUK and UFPK, enabling confidential transfer of the key injection file.

5.3.3 Wrapping the New ECC Key Pair with the KUK

Use the following command to wrap the new ECC public key shown in Figure 5-11.

```
skmt.exe /genkey /kuk
file="C:\Secure_Key_Injection\rx74m_protected_mode_key_info\kuk_for_new_key_cli.
key" /mcu "RX-RSIP-E50D" /keytype "secp256r1-public" /key
"e266dddfdc12668db30d4ca3e8f7749432c416044f2d2b8c10bf3d4012aeffa8abfa86404a2e9ffe
67d47c587ef7a97a7f456b863b4d02cfc6928973ab5b1cb39" /filetype "csource" /keyname
"NEW_ECC_PUB" /output
"C:\Secure_Key_Injection\rx74m_protected_mode_key_info\new_ecc_public_key_cli.c"
```

Note the following in this example.

- e266dddfdc12668db30d4ca3e8f7749432c416044f2d2b8c10bf3d4012aeffa8abfa86404a2e9ffe67d47c587ef7a97a7f456b863b4d02cfc6928973ab5b1cb39 is the ECC public key of the NIST test vector.
- The value for key type "secp256r1-public" is one of the available options prescribed in the SKMT user's manual.
- "RX-RSIP-E50D" is used for the /mcu option.
- A randomly generated IV is used. The IV is updated each time this command is executed.
- The command option /output defines the location and name of the output file.
- /keyname is used to create an identifiable key structure name that is unique within the software project. This resolves potential confusion when two or more sets of new user keys are to be generated. If this option is not specified, a key structure named encrypted_user_key_data is generated.
- The generated new_ecc_public_key_cli.c and new_ecc_public_key_cli.h files contain the output information in a key structure. This information can be checked in the files the sample project contains.

```
Output File: C:\Secure_Key_Injection\rx74m_protected_mode_key_info\new_ecc_public_key_cli.h
Output File: C:\Secure_Key_Injection\rx74m_protected_mode_key_info\new_ecc_public_key_cli.c
KUK: 000102030405060708090A0B0C0D0E0F000102030405060708090A0B0C0D0E0F
IV: 36E763D5A82924B4888732D50C93B602
Encrypted key:
9B0A7F8C91C038704A4F2C758EAC3DDD1372B4DC6AA4F22667D7D0E41218A1DEDBB8337E557B59B91100225BC8BBE280
72214FF3C729D953AEFA9E997C3989967C831DC6501E9528715ADA30FA0D0402
```

Figure 5-19 Encrypting the New ECC Public Key with the KUK

Use the following command to wrap the new ECC private key shown in Figure 5-13.

```
skmt.exe /genkey /kuk  
file="C:\Secure_Key_Injection\rx74m_protected_mode_key_info\kuk_for_new_key_cli.  
key" /mcu "RX-RSIP-E50D" /keytype "secp256r1-private" /key  
"0f56db78ca460b055c500064824bed999a25aaf48ebb519ac201537b85479813" /filetype  
"csource" /keyname "NEW_ECC_PRI" /output  
"C:\Secure_Key_Injection\rx74m_protected_mode_key_info\new_ecc_private_key_cli.c  
"
```

Note the following in this example.

- 0f56db78ca460b055c500064824bed999a25aaf48ebb519ac201537b85479813 is the ECC private key of the NIST test vector.
- The value for key type "secp256r1-private" is one of the available options prescribed in the SKMT user's manual.
- "RX-RSIP-E50D" is used for the /mcu option.
- A randomly generated IV is used. The IV is updated each time this command is executed.
- The command option /output defines the location and name of the output file
- /keyname is used to create an identifiable key structure name that is unique within the software project. This resolves potential confusion when two or more sets of new user keys are to be generated. If this option is not specified, a key structure named encrypted_user_key_data is generated.
- The generated new_ecc_private_key_cli.c and new_ecc_private_key_cli.h files contain the output information in a key structure. This information can be checked in the files the sample project contains.

```
Output File: C:\Secure_Key_Injection\rx74m_protected_mode_key_info\new_ecc_private_key_cli.h  
Output File: C:\Secure_Key_Injection\rx74m_protected_mode_key_info\new_ecc_private_key_cli.c  
KUK: 000102030405060708090A0B0C0D0E0F000102030405060708090A0B0C0D0E0F  
IV: 36E763D5A82924B4888732D50C93B602  
Encrypted key:  
9B0A7F8C91C038704A4F2C758EAC3DDD1372B4DC6AA4F22667D7D0E41218A1DEDBB8337E557B59B91100225BC8BBE280
```

Figure 5-20 **Encrypting the New ECC Private Key with the KUK**

5.4 Secure Key Injection through the MCU Boot Firmware Interface

Follow the steps in this section to inject the ECC public key, ECC private key, and KUK that were prepared in section 5.2 or section 5.3. This functionality is supported by the MCUs of the RX Family that incorporate the RSIP (protected mode) security engine.

5.4.1 Setting up the Hardware

Set up the hardware as follows when the EK-RX74M evaluation board is used.

- Set switches 1 and 2 of SW6 off on the EK-RX74M to set the MCU operating mode to debug on-board mode. For details, refer to the EK-RX74M user's manual.
- Connect the J8 connector on the EK-RX74M to the PC you are using for development through a USB Type-C® cable to supply power and make the debugging facilities available.

Erase the entire MRAM and ensure that the MCU is in the OEM device lifecycle state. This can be done by using the RFP as shown below.

1. Unzip `rfp_resources_rx74m.zip`.
2. Launch the RFP GUI.
3. Select **File** -> **Open Project** as shown in Figure 5-21 and select `rx74m_secure_key_inject.rpj`.
4. Select **Target Device** -> **Initialize Device** as shown in Figure 5-21.

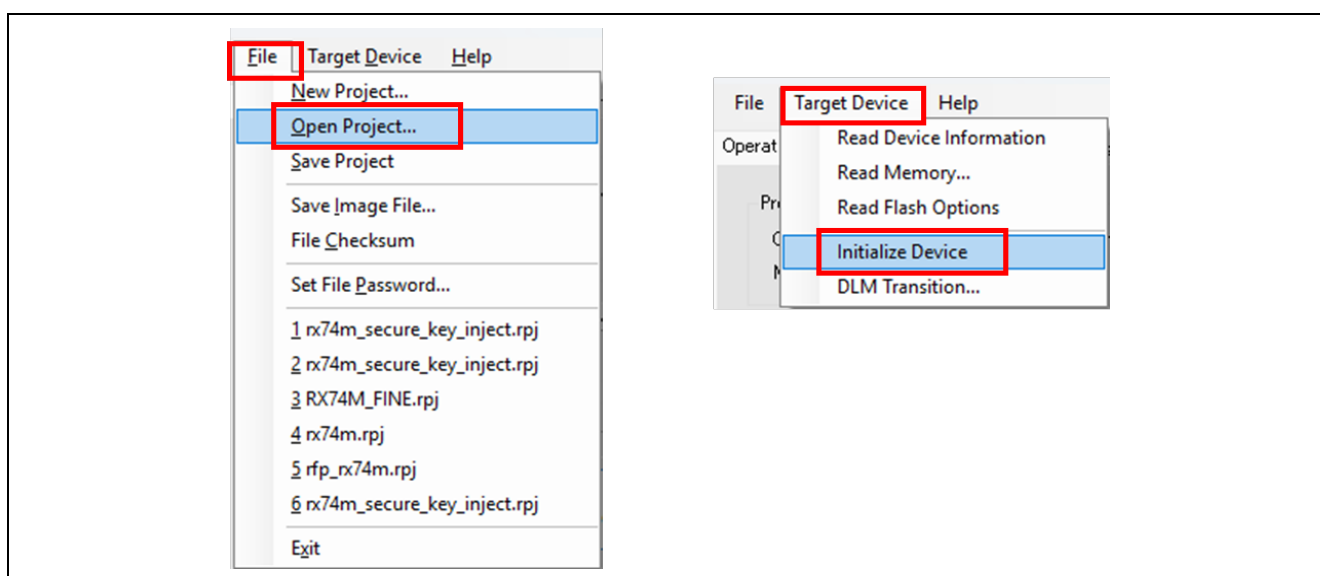


Figure 5-21 Opening the RFP Project and Initializing the Device

The message shown in Figure 5-22 will appear upon successful initialization.

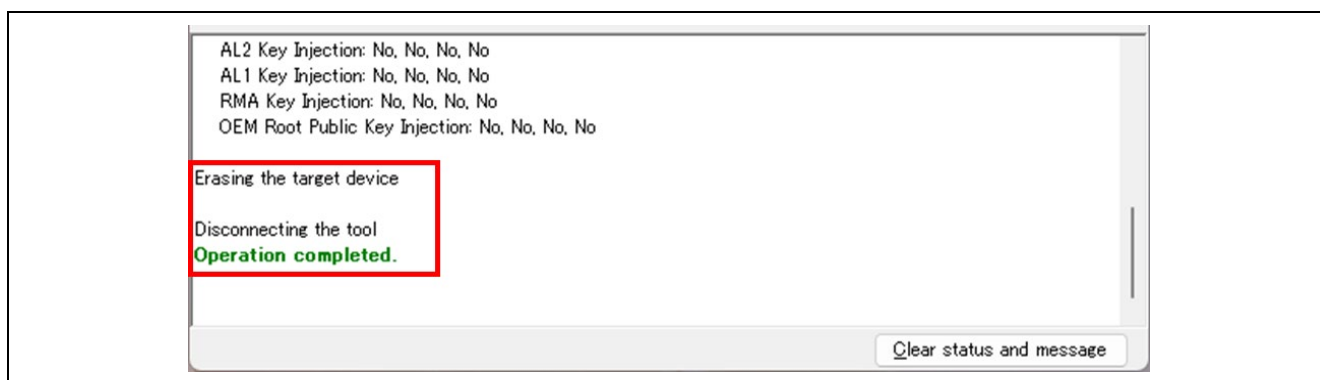


Figure 5-22 RX74M Initialization

5.4.2 Injecting the Initial User Keys and KUK

After having set up the hardware, turn power to the board off and then on, and follow the steps below to inject the ECC key pair and KUK. The set of injection keys used in this section is that generated through the GUI interface.

To simplify the execution of these steps, the .rkey files that match the sample project are included in the rfp_resources_rx74m.zip file. The screen captures in this section are with these files in use for demonstration purposes. To use different keys, update the corresponding .rkey files to match those keys.

The following describes how to inject keys in the EK-RX74M sample project.

On the **Operation** tabbed page, click on **Add/Remove Files**. Next, click on **Add File(s)** and add the .rkey file containing the ECC private key, which for this example is ECC_Private_Key.rkey (Figure 5-6). Set the **Address** property to a data MRAM or code MRAM address that is applicable to your specific application. In this example, the ECC private key is injected into address 0xFFFFD0000 of the data MRAM.

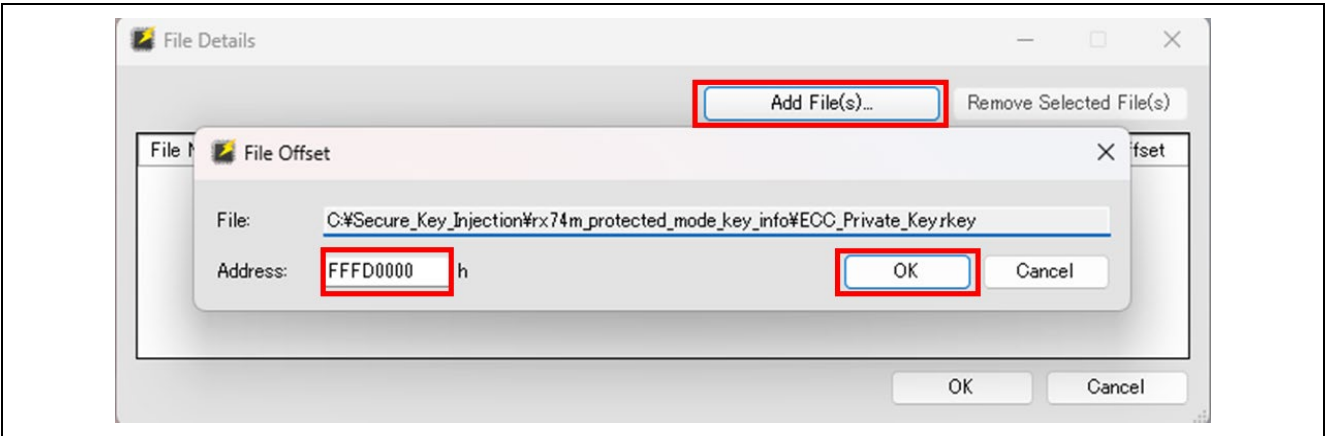


Figure 5-23 Adding ECC_Private_Key.rkey to the RFP Configuration

Click on **OK** and the ECC_Private_Key.rkey file will be configured at the corresponding load address.

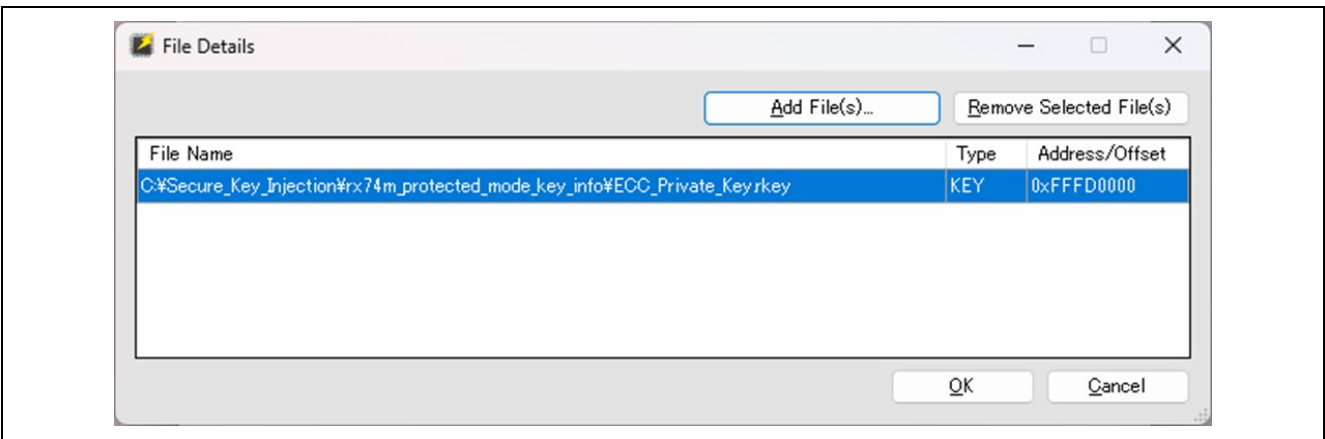


Figure 5-24 ECC_Private_Key.rkey Added to the RFP Configuration

Click on **Add File(s)** again to add `ECC_Public_Key.rkey`. Browse to find `ECC_Public_Key.rkey` (Figure 5-3). Set the **Address** property to a data MRAM or code MRAM address that is applicable to your specific application. In this example, the ECC public key is injected into address `0xFFFD0100` of the data MRAM.

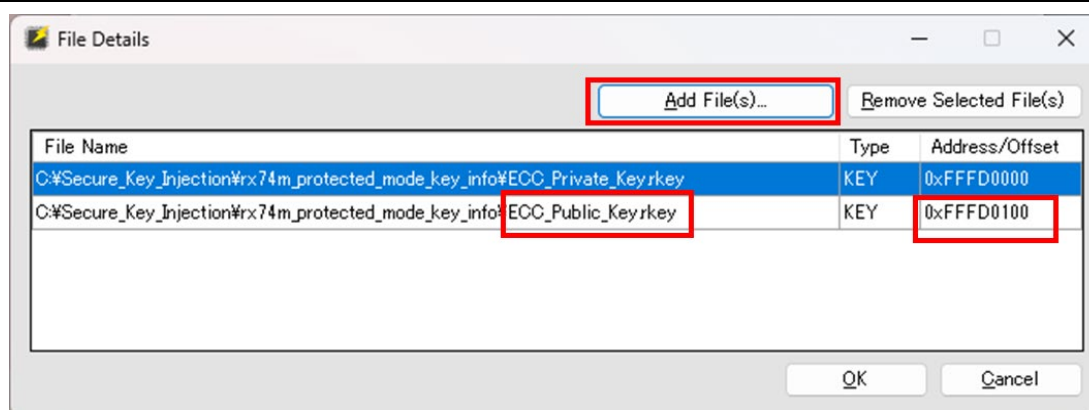


Figure 5-25 Selecting the ECC Public Key and Specifying the Injection Address

Click on **Add File(s)** again to add `KUK.rkey`. Browse to find `KUK.rkey` (Figure 5-10). Set the **Address** property to a data MRAM or code MRAM address that is applicable to your specific application. In this example, the KUK is injected into address `0xFFFD0200` of the code MRAM.

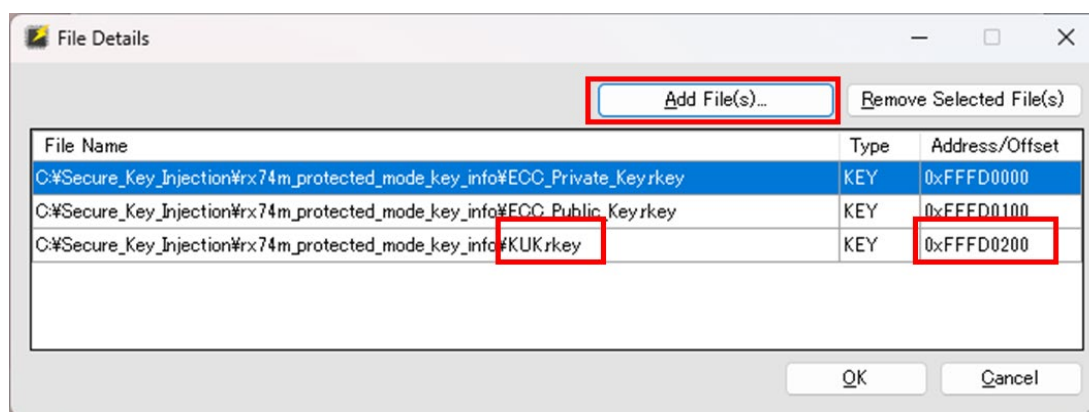


Figure 5-26 Selecting the KUK and Specifying the Injection Address

Click on **OK** and open the **Operation Settings** tabbed page. Check that **Erase**, **Program**, and **Verify** are selected.

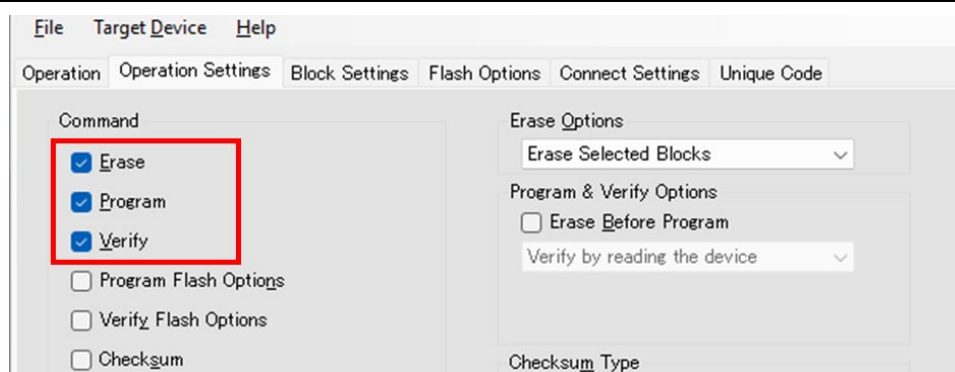
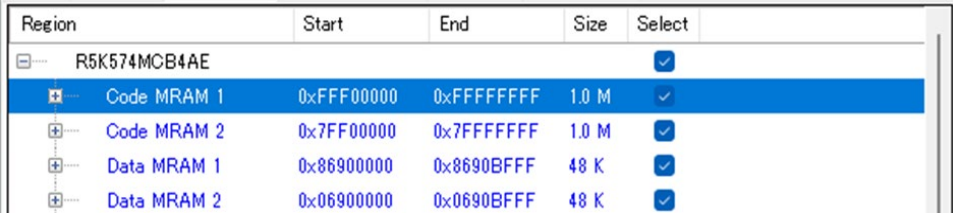


Figure 5-27 Selecting Execution of Erasing, Programming, and Verifying the MRAM

- Open the **Block Settings** tabbed page and check that the entire MRAM region is selected for erasure.



Region	Start	End	Size	Select
R5K574MCB4AE				☒
Code MRAM 1	0xFFFF0000	0xFFFFFFFF	1.0 M	☒
Code MRAM 2	0x7FF00000	0x7FFFFFFF	1.0 M	☒
Data MRAM 1	0x86900000	0x8690BFFF	48 K	☒
Data MRAM 2	0x06900000	0x0690BFFF	48 K	☒

Figure 5-28 Entire MRAM Region Selected for Erasure

- Open the **Operation** tabbed page. Click on **Start** to inject the ECC private key, ECC public key, and KUK. A similar output message to that shown below will be displayed for the selected MRAM addresses upon successful completion of injection.

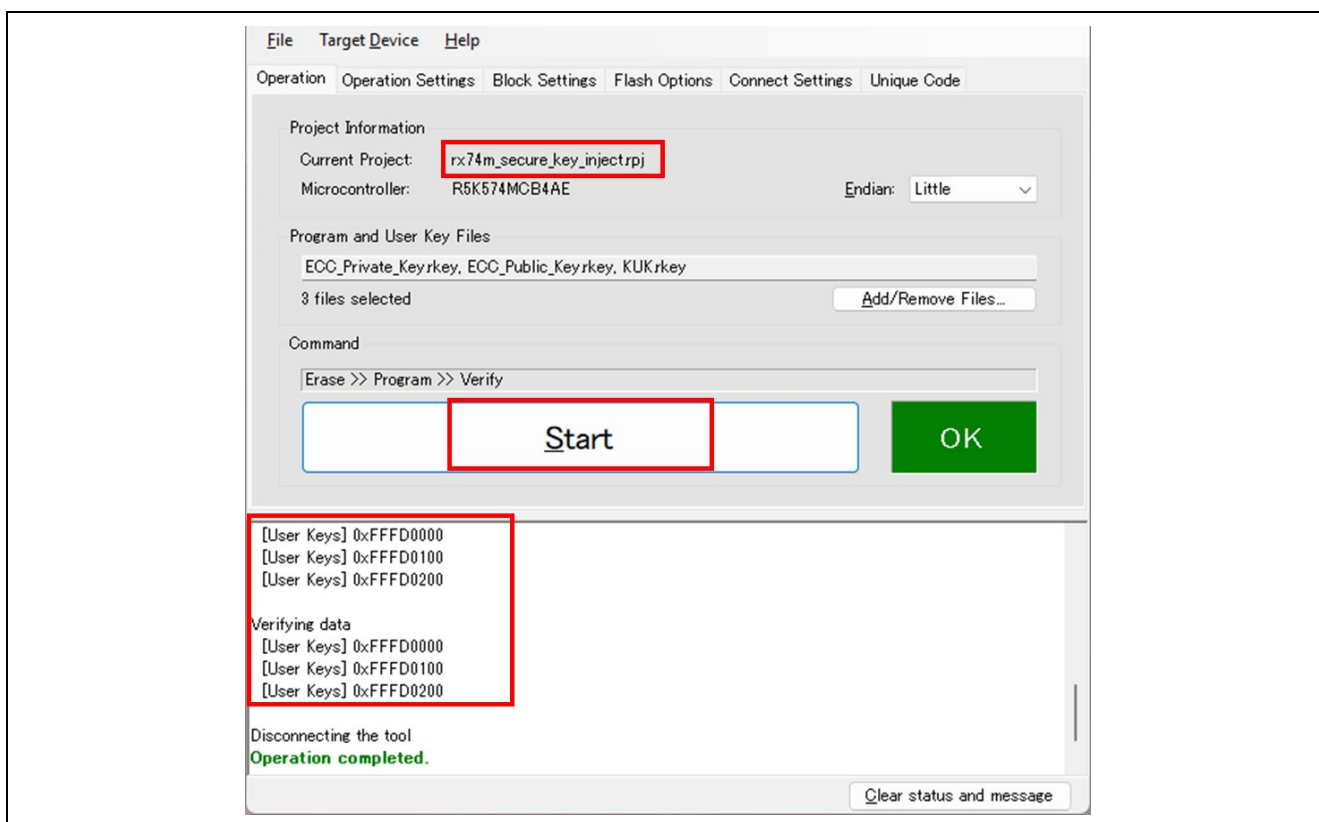


Figure 5-29 Secure Key Injection was Successful

In this sample code, no application is programmed because it only shows how to inject keys. In an actual production flow, an application and user keys can be programmed together in the code. The command-line facility of the RFP can also be used for this purpose.

5.5 Sample Project for Protected Mode

This section introduces RSIP protected mode with an example of wrapped ECC key pair injection and updating.

To execute the sample project as is, follow the steps below.

- Inject the provided sample RFP injection keys (KUK.rkey, ECC_Public_Key.rkey, and ECC_Private_Key.rkey that are contained in rfp_resource_rx74m.zip) by following the procedure in section 5.4.2.
- A set of new user keys (ECC public and private keys) generated by using the sample KUK is already provisioned in the sample project. This allows the user to directly proceed with execution of the sample project.

Note: Do not use the sample keys for mass production.

To use the sample project with new keys generated by the user, follow the steps below.

- To test newly generated injection keys and user update keys (keys generated by following the procedure in section 5.2 or 5.3 rather than those contained in rfp_resources_rx74m.zip), the user needs to follow section 5.4 to inject the keys to the MCU. The user also needs to generate new user key files (new_ecc_public_key.c/.h and new_ecc_private_key.c/.h) with the same key name to replace the corresponding files used in the sample project. Once the sample project is updated, the user can proceed with execution of the sample project to verify operations.
- To only test the procedure for updating with new user keys, the user can use the provided KUK.rkey file to generate new source files to replace the corresponding files in the sample project. Once the sample project has been updated, the user can proceed with verification of operations.

5.5.1 Overview of the Sample Project

This sample project executes the following processing in protected mode of the RSIP-E50D.

- The injected ECC private key is used to sign the NIST test vector message, after which the injected ECC public key is used to verify the signature created with the corresponding private key.
- The injected KUK is used to inject a newly wrapped ECC key pair and this new ECC key pair is stored in MRAM.
- The new ECC key pair is used to create a signature and then verify the signature.

5.5.2 Injecting Keys in the Sample Project

5.5.2.1 Formatting the Injected Keys

The keys injected into the MRAM of the MCU by using the RFP cannot be used directly by the FSP Crypto API functions. A minor change in the settings for formatting is required.

(1) Formatting the Injected ECC Key Pair

The information of the ECC public key and ECC private key is stored in the following section of the source code (`hal_entry.c`). The variables in which the information is stored can then be used for cryptographic operations. Replace the macros `ECC_PUB_KEY_ADDRESS` and `ECC_PRI_KEY_ADDRESS` with the actual injection addresses and specify the key type so that the information matches the format that can be used by the FSP Crypto API functions.

```
static rsip_wrapped_key_t ecc_public_key_injected =
{
    .type = RSIP_KEY_TYPE_ECC_SECP256R1_PUBLIC, .p_value = (void *)ECC_PUB_KEY_ADDRESS
};

static rsip_wrapped_key_t ecc_private_key_injected =
{
    .type = RSIP_KEY_TYPE_ECC_SECP256R1_PRIVATE, .p_value = (void *) ECC_PRI_KEY_ADDRESS
};
```

(2) Formatting the Injected KUK

The information of the injected KUK is stored in the following section of the source code (`hal_entry.c`). The variable in which the information is stored can then be used for secure key updating. Replace the macro `KUK_ADDRESS` with the actual injection address and specify the key type so that the information matches the format that can be used by the FSP Crypto API functions.

```
static rsip_wrapped_key_t kuk_key =
{
    .type = RSIP_KEY_TYPE_KUK, .p_value = (void *) KUK_ADDRESS
};
```

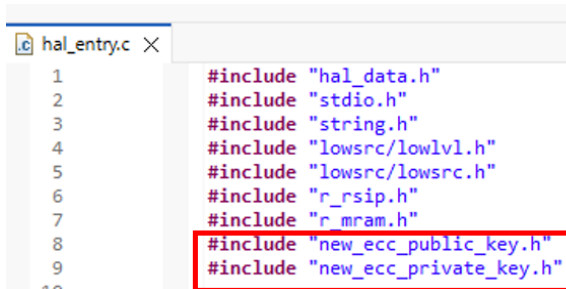
5.5.2.2 Verifying the Injected and Updated Keys

To check whether the wrapped ECC key pair has been correctly injected, compare the result of injection with the expected value in the NIST test vector that uses the same key pair. Specifically, check that the signature generated with the ECC private key matches the expected value in the test vector and that the signature can be successfully verified by the ECC public key.

5.5.3 Executing the Sample Project

Follow the steps below to execute the sample project. Note that some sections of the code must be updated by using the secure key injection results generated with the above steps before building and execution of the project.

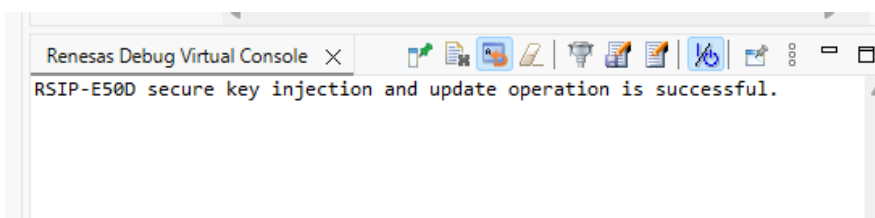
1. Launch the e² studio and import `secure_key_inject_update_rx74m_pm_ccrx` or `secure_key_inject_update_rx74m_pm_gcc` to a workspace.
2. To generate and use a new ECC key pair, replace the files located in the folder `\secure_key_inject_update_rx74m\src\` with `new_ecc_public_key.c` and `new_ecc_public_key.h` generated through the method in section 5.2.3 or 5.3.
3. If different file names are used, update the include file statements in `hal_entry.c` to reflect the new file names.



```
1 #include "hal_data.h"
2 #include "stdio.h"
3 #include "string.h"
4 #include "lowsrc/lowlvl.h"
5 #include "lowsrc/lowsrc.h"
6 #include "r_rsip.h"
7 #include "r_mram.h"
8 #include "new_ecc_public_key.h"
9 #include "new_ecc_private_key.h"
```

Figure 5-30 Including the Header Files Generated for ECC Operations

4. Next, double-click on `configuration.xml`. Once the configurator is opened, click on **Generate Project Content** to build the project.
5. Select **Renesas Views -> Debug -> Renesas Debug Virtual Console** to open the console view.
6. Execute debugging.
7. Confirm that the following message appears in the console view.



```
Renesas Debug Virtual Console
RSIP-E50D secure key injection and update operation is successful.
```

Figure 5-31 Message in the Console View

6. Secure Key Injection in RSIP Compatibility Mode and the Sample Project

6.1 Overview

This section describes preparing for secure key injection in RSIP-E50D compatibility mode and a sample project for using the prepared key.

The key injection API functions of the FSP are used for key injection to PSA Crypto API functions in compatibility mode. This section describes the procedures for using the SKMT to generate C source and header files that can be used in the sample project.

This sample project demonstrates the following functions in compatibility mode of the RSIP-E50D.

- Injection of an AES-128 key
- AES-128 encryption and decryption by using the injected AES-128 key

The sample project in this application note uses the NIST CAVP test vectors as examples. To check the test vectors, download the related files from the following link.

<https://csrc.nist.gov/Projects/Cryptographic-Algorithm-Validation-Program/Block-Ciphers>

6.2 Wrapping a Key with the UFPK and W-UFPK through the SKMT GUI

Use the SKMT GUI and select **RX Family, RSIP-E50D Compatibility Mode** on the **Overview** tabbed page.

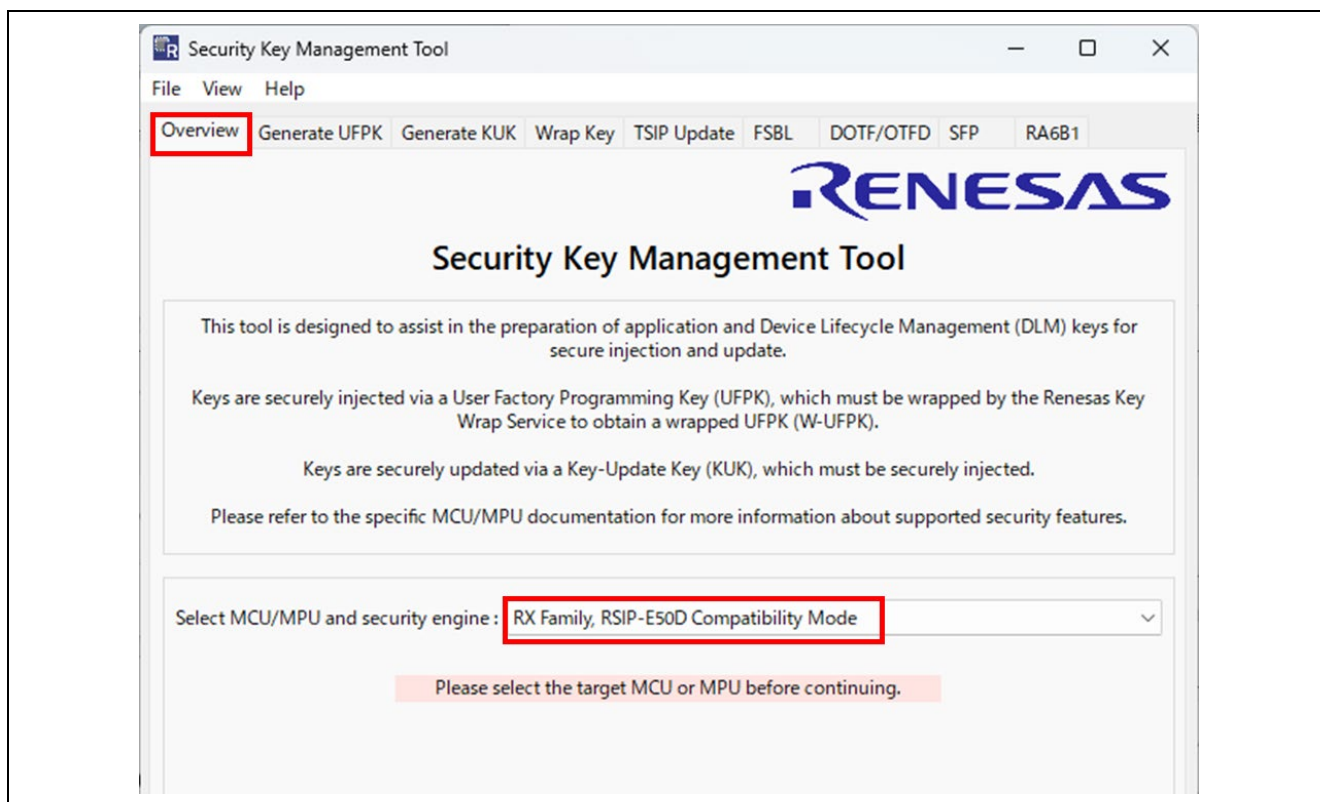
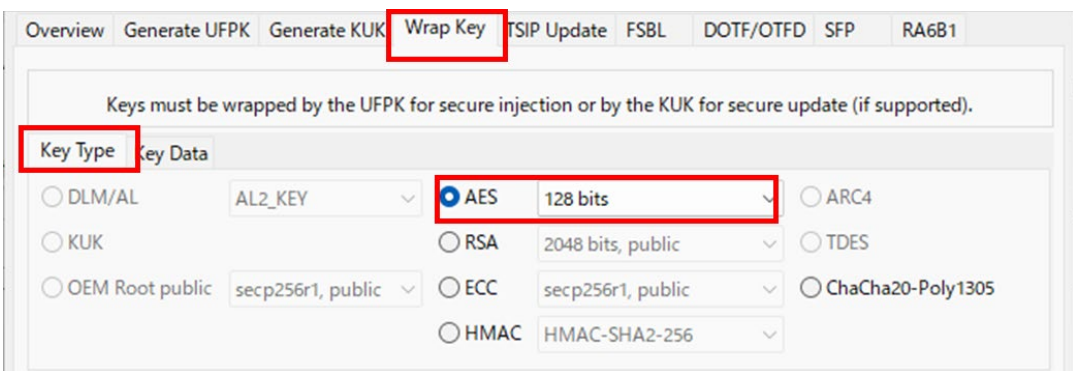


Figure 6-1 Selecting RX Family, RSIP-E50D Compatibility Mode

Set **Key Type** to **AES** and **128 bits** on the **Wrap Key** tabbed page.



Overview Generate UFPK Generate KUK **Wrap Key** TSIP Update FSBL DOTF/OTFD SFP RA6B1

Keys must be wrapped by the UFPK for secure injection or by the KUK for secure update (if supported).

Key Type Key Data

☐ DLM/AL AL2_KEY ☒ **AES** 128 bits ☐ ARC4

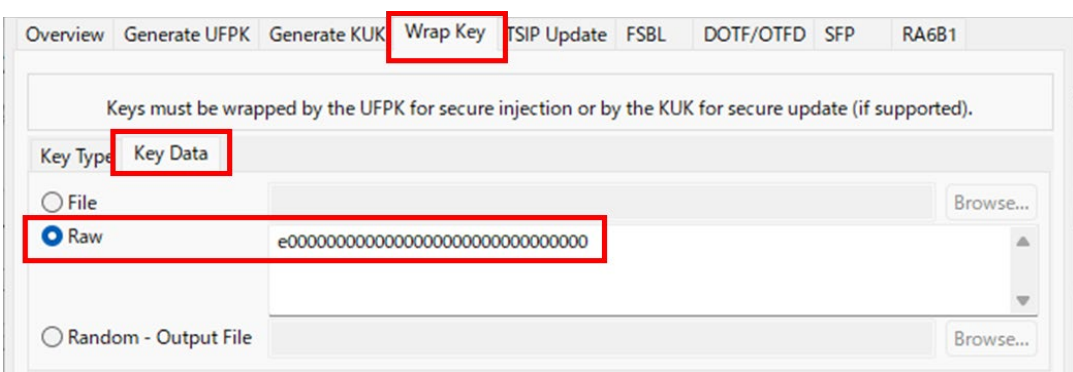
☐ KUK ☐ RSA 2048 bits, public ☐ TDES

☐ OEM Root public secp256r1, public ☐ ECC secp256r1, public ☐ ChaCha20-Poly1305

☐ HMAC HMAC-SHA2-256

Figure 6-2 Selecting AES and 128 bits as the Key Type

Select the **Key Data** tab and enter key data in the **Raw** field as shown below and set the value. In this document, the value is set to e0000000000000000000000000000000.



Overview Generate UFPK Generate KUK **Wrap Key** TSIP Update FSBL DOTF/OTFD SFP RA6B1

Keys must be wrapped by the UFPK for secure injection or by the KUK for secure update (if supported).

Key Type **Key Data**

☐ File ☒ **Raw** e0000000000000000000000000000000 ☐ Random - Output File

Browse... Browse...

Figure 6-3 Setting the Initial AES-128 Key Data

Under **Wrapping Key**, click on the **Browse** buttons corresponding to the UFPK and W-UFPK to select a key pair. For **IV** data, select **Generate random value**. For the **Format** option, select **C Source**. After that, click on the **Browse** button to choose the output folder and file and name the key. This name will be reflected in the definitions generated for the C source file.

Next, click on **Generate file** and the source files for AES key injection will be generated.

Wrapping Key

☒ UFPK UFPK File: C:\DLM_Key_Injection\test_gui\#rx74m_ufpk.key Browse...

W-UFPK File: C:\DLM_Key_Injection\test_gui\#rx74m_ufpk.key_enc.key Browse...

☐ KUK KUK File: Browse...

IV

☒ Generate random value

☐ Use specified value (16 hex bytes, big endian format) 00112233445566778899AABBCCDDEEFF

Output

Format: C Source File: 4m_protected_mode_key_info#rx74m_initial_aes128_key.c Browse...

Endian: Little ☐ Output additional data

Address: 10000 Key name: RX74M_AES128

Generate file

Command Line: skmt.exe /genkey /ufpk "C:\DLM_Key_Injection\test_gui\#rx74m_ufpk.key" /wufpk "C:\DLM_Key_Injection\test_gui\#rx74m_ufpk.key_enc.key" /mcu "RX-RSIP-E50D-CM" /keytype "AES-128" /key "e000000000000000000000000000000000" / filetype "csource" /keyname "RX74M_AES128" /output "C:\Secure_Key_Injection\#rx74m_protected_mode_key_info\#rx74m_initial_aes128_key.c"

Output File: C:\Secure_Key_Injection\#rx74m_protected_mode_key_info\#rx74m_initial_aes128_key.h

Output File: C:\Secure_Key_Injection\#rx74m_protected_mode_key_info\#rx74m_initial_aes128_key.c

UFPK: 000102030405060708090A0B0C0D0E0F000102030405060708090A0B0C0D0E0F

W-UFPK: 00000000F77A67AB713D174EB9D7C6639DAADB1F77A67AB713D174EB9D7C6639DAADB1

IV: 4FDF578174ECACDF9C5BEC28221A8288

Encrypted key: 0D5260D25A077FE2C745E3BFF108A273A204CFEE60903842B96FF2651E9B65AC

OPERATION SUCCESSFUL

Figure 6-4 Generating the Initial AES-128 Encryption Key File

Note that the data in the generated `rx74m_initial_aes128_key.c` are sample data and are only intended for use in the sample project. Do not use them in your finished products.

6.3 Wrapping the Key with the UFPK and W-UFPK through the SKMT CLI

Use the SKMT to create the source files of the wrapped key through the following command.

```
skmt.exe /genkey /ufpk  
file="C:\Secure_Key_Injection\rx74m_compatibility_mode_key_info\rx74m_ufpk.key"  
/wufpk  
file="C:\Secure_Key_Injection\rx74m_compatibility_mode_key_info\rx74m_ufpk.key_e  
nc.key" /mcu "RX-RSIP-E50D-CM" /keytype "AES-128"  
/key "e0000000000000000000000000000000" /filetype "csource" /output  
"C:\Secure_Key_Injection\rx74m_compatibility_mode_key_info\rx74m_initial_aes128_  
key_cli.c"
```

Note the following in this example.

- The key type is specified as "AES-128".
- A randomly generated IV value is used. The IV is updated each time this command is executed.
- In this example, UFPK file and W-UFPK file are stored in
C:\Secure_Key_Injection\rx74m_compatibility_mode_key_info folder.
- The complete file path for the source file (rx74m_initial_aes128_key_cli.c) is specified.

```
Output File:  
C:\Secure_Key_Injection\rx74m_compatibility_mode_key_info\rx74m_initial_aes128_key_cli.c  
UFPK: 000102030405060708090A0B0C0D0E0F000102030405060708090A0B0C0D0E0F  
W-UFPK: 1234567890ABCDEF1234567890ABCDEF1234567890ABCDEF1234567890ABCDEF12345678  
IV: 1234567890ABCDEF1234567890ABCDEF  
Encrypted key:  
1234567890ABCDEF1234567890ABCDEF1234567890ABCDEF1234567890ABCDEF1234567890ABCDEF
```

Figure 6-5 Generating the KUK Injection File through the CLI

6.4 Sample Project for Compatibility Mode

This section introduces RSIP compatibility mode with an example of AES-128 user key injection.

6.4.1 Using the Files Generated through the SKMT

The source files generated through the method in section 6.2 are included in the sample project. To use newly generated source files, replace the files in `src\` with the new ones.

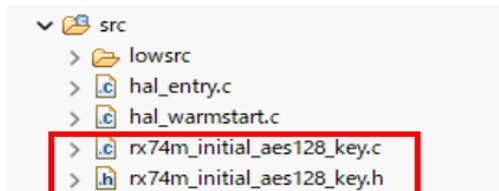


Figure 6-6 Source Code of the RX74M Sample Project

6.4.2 Executing the Sample Project

Follow the steps below to execute the sample project.

1. Launch the e² studio and import `secure_key_inject_rx74m_cm_ccrx` or `secure_key_inject_rx74m_cm_gcc` to a workspace.
2. To generate and use a new AES-128 key, replace the files located in the folder `src\` with `rx74m_initial_aes128_key.h` and `rx74m_initial_aes128_key.c` generated through the method in section 6.2.
3. If a different file name is used, update the include file statement in `hal_entry.c` to reflect the new file name.

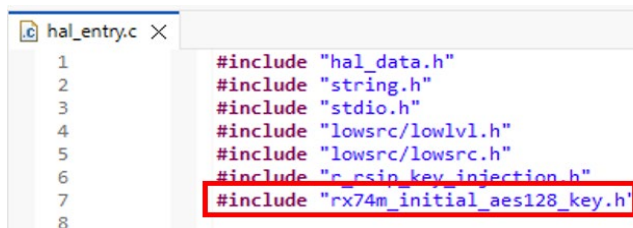


Figure 6-7 Including the Header File Generated for AES Operations

4. Next, double-click on `configuration.xml`. Once the configurator is opened, click on **Generate Project Content** to build the project.
5. Select **Renesas Views -> Debug -> Renesas Debug Virtual Console** to open the console view.
6. Execute debugging.
7. Confirm that the following message appears in the console view.

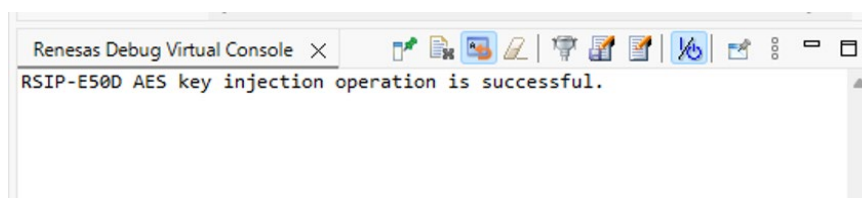


Figure 6-8 Message in the Console View

7. References

1. RX74M Group User's Manual: Hardware (R01UH1055)
2. Security Key Management Tool Software Manual (R20UT5349)
3. Renesas Flash Programmer V3.23 User's Manual (R20UT5784)

8. Website and Support

Visit the following URLs to learn about microcontrollers of the RX Family, download tools and related documentation, and get support.

EK-RX74M Resources	renesas.com/ek-rx74m
RX Product Information	renesas.com/rx
Flexible Software Package (FSP)	renesas.com/rx-fsp
RX Product Support Forum	renesas.com/rx/forum
Renesas Support	renesas.com/support

Revision History

Rev.	Date	Description	
		Page	Summary
1.00	Oct.22.26	—	Initial release

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

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

1. Precaution against Electrostatic Discharge (ESD)

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

2. Processing at power-on

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

3. Input of signal during power-off state

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

4. Handling of unused pins

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

5. Clock signals

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

6. Voltage application waveform at input pin

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

7. Prohibition of access to reserved addresses

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

8. Differences between products

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

Notice

1. Descriptions of circuits, software and other related information in this document are provided only to illustrate the operation of semiconductor products and application examples. You are fully responsible for the incorporation or any other use of the circuits, software, and information in the design of your product or system. Renesas Electronics disclaims any and all liability for any losses and damages incurred by you or third parties arising from the use of these circuits, software, or information.
2. Renesas Electronics hereby expressly disclaims any warranties against and liability for infringement or any other claims involving patents, copyrights, or other intellectual property rights of third parties, by or arising from the use of Renesas Electronics products or technical information described in this document, including but not limited to, the product data, drawings, charts, programs, algorithms, and application examples.
3. No license, express, implied or otherwise, is granted hereby under any patents, copyrights or other intellectual property rights of Renesas Electronics or others.
4. You shall be responsible for determining what licenses are required from any third parties, and obtaining such licenses for the lawful import, export, manufacture, sales, utilization, distribution or other disposal of any products incorporating Renesas Electronics products, if required.
5. You shall not alter, modify, copy, or reverse engineer any Renesas Electronics product, whether in whole or in part. Renesas Electronics disclaims any and all liability for any losses or damages incurred by you or third parties arising from such alteration, modification, copying or reverse engineering.
6. Renesas Electronics products are classified according to the following two quality grades: "Standard" and "High Quality". The intended applications for each Renesas Electronics product depends on the product's quality grade, as indicated below.
 - "Standard": Computers; office equipment; communications equipment; test and measurement equipment; audio and visual equipment; home electronic appliances; machine tools; personal electronic equipment; industrial robots; etc.
 - "High Quality": Transportation equipment (automobiles, trains, ships, etc.); traffic control (traffic lights); large-scale communication equipment; key financial terminal systems; safety control equipment; etc.
- Unless expressly designated as a high reliability product or a product for harsh environments in a Renesas Electronics data sheet or other Renesas Electronics document, Renesas Electronics products are not intended or authorized for use in products or systems that may pose a direct threat to human life or bodily injury (artificial life support devices or systems; surgical implantations; etc.), or may cause serious property damage (space system; undersea repeaters; nuclear power control systems; aircraft control systems; key plant systems; military equipment; etc.). Renesas Electronics disclaims any and all liability for any damages or losses incurred by you or any third parties arising from the use of any Renesas Electronics product that is inconsistent with any Renesas Electronics data sheet, user's manual or other Renesas Electronics document.
7. No semiconductor product is absolutely secure. Notwithstanding any security measures or features that may be implemented in Renesas Electronics hardware or software products, Renesas Electronics shall have absolutely no liability arising out of any vulnerability or security breach, including but not limited to any unauthorized access to or use of a Renesas Electronics product or a system that uses a Renesas Electronics product. RENESAS ELECTRONICS DOES NOT WARRANT OR GUARANTEE THAT RENESAS ELECTRONICS PRODUCTS, OR ANY SYSTEMS CREATED USING RENESAS ELECTRONICS PRODUCTS WILL BE INVULNERABLE OR FREE FROM CORRUPTION, ATTACK, VIRUSES, INTERFERENCE, HACKING, DATA LOSS OR THEFT, OR OTHER SECURITY INTRUSION ("Vulnerability Issues"). RENESAS ELECTRONICS DISCLAIMS ANY AND ALL RESPONSIBILITY OR LIABILITY ARISING FROM OR RELATED TO ANY VULNERABILITY ISSUES. FURTHERMORE, TO THE EXTENT PERMITTED BY APPLICABLE LAW, RENESAS ELECTRONICS DISCLAIMS ANY AND ALL WARRANTIES, EXPRESS OR IMPLIED, WITH RESPECT TO THIS DOCUMENT AND ANY RELATED OR ACCOMPANYING SOFTWARE OR HARDWARE, INCLUDING BUT NOT LIMITED TO THE IMPLIED WARRANTIES OF MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE.
8. When using Renesas Electronics products, refer to the latest product information (data sheets, user's manuals, application notes, "General Notes for Handling and Using Semiconductor Devices" in the reliability handbook, etc.), and ensure that usage conditions are within the ranges specified by Renesas Electronics with respect to maximum ratings, operating power supply voltage range, heat dissipation characteristics, installation, etc. Renesas Electronics disclaims any and all liability for any malfunctions, failure or accident arising out of the use of Renesas Electronics products outside of such specified ranges.
9. Although Renesas Electronics endeavors to improve the quality and reliability of Renesas Electronics products, semiconductor products have specific characteristics, such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Unless designated as a high reliability product or a product for harsh environments in a Renesas Electronics data sheet or other Renesas Electronics document, Renesas Electronics products are not subject to radiation resistance design. You are responsible for implementing safety measures to guard against the possibility of bodily injury, injury or damage caused by fire, and/or danger to the public in the event of a failure or malfunction of Renesas Electronics products, such as safety design for hardware and software, including but not limited to redundancy, fire control and malfunction prevention, appropriate treatment for aging degradation or any other appropriate measures. Because the evaluation of microcomputer software alone is very difficult and impractical, you are responsible for evaluating the safety of the final products or systems manufactured by you.
10. Please contact a Renesas Electronics sales office for details as to environmental matters such as the environmental compatibility of each Renesas Electronics product. You are responsible for carefully and sufficiently investigating applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive, and using Renesas Electronics products in compliance with all these applicable laws and regulations. Renesas Electronics disclaims any and all liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.
11. Renesas Electronics products and technologies shall not be used for or incorporated into any products or systems whose manufacture, use, or sale is prohibited under any applicable domestic or foreign laws or regulations. You shall comply with any applicable export control laws and regulations promulgated and administered by the governments of any countries asserting jurisdiction over the parties or transactions.
12. It is the responsibility of the buyer or distributor of Renesas Electronics products, or any other party who distributes, disposes of, or otherwise sells or transfers the product to a third party, to notify such third party in advance of the contents and conditions set forth in this document.
13. This document shall not be reprinted, reproduced or duplicated in any form, in whole or in part, without prior written consent of Renesas Electronics.
14. Please contact a Renesas Electronics sales office if you have any questions regarding the information contained in this document or Renesas Electronics products.

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

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

(Rev.5.0-1 October 2020)

Corporate Headquarters

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

Trademarks

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

Contact information

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