

Renesas RA Family

## Injecting and Updating Secure User Keys

## Introduction

Cryptography is important because it provides the tools to implement solutions for authenticity, confidentiality, and integrity, which are vital aspects of any security solution. In modern cryptographic systems, the security of the system no longer depends on the secrecy of the algorithm used but rather on the secrecy of the keys.

Renesas MCU security revolves around integrated security engines. There are different types of security engines across the RA MCU. Users can find the specific engine used in a particular MCU from its hardware user's manual.

- The following two types of security engines can operate in two different modes, called Compatibility Mode and Protected Mode. The application note Renesas SCE Operational Modes (R11AN0498) explains the definition of the two modes and their use cases. In Compatibility Mode, the following two security engines can inject secure keys as well as plaintext keys. In Protected Mode, these two security engines can inject only secure keys.
  - The Renesas Secure IP (RSIP) security engine, which is available on RA8 MCUs.
  - The Secure Crypto Engine 9 (SCE9) which is available on some RA6 and RA4 MCUs
- Other available security engines used in RA Family MCUs are Secure Crypto Engine 7 (SCE7), Secure Crypto Engine 5 (SCE5), and Secure Crypto Engine 5\_B (SCE5\_B). These security engines can only operate in Compatibility Mode and can inject secure keys as well as plaintext keys.

With this release, this application project demonstrates the following secure key injection processes:

- RSIP Compatibility mode AES-128 secure key injection using RA8M1 MCU
- SCE9 Protected Mode AES-256 and ECC secp256r1 public key secure key injection using RA6M4 MCU
- SCE7 Compatibility Mode AES-128 secure key injection using RA6M3 MCU. Compatibility Mode secure key injection for SCE5 and SCE5\_B uses APIs identical to those of SCE7 secure key injection.

Example keys are provided with the projects. This application note describes how to modify the projects to use custom keys.

#### **Required Resources**

#### Target Devices:

RA8M1/RA8D1/RA8T1 (with RSIP)

RA6M1/RA6M2/RA6M3 (with SCE7 Compatibility Mode only)

RA6M4/RA6M5/RA4M2/RA4M3 (with SCE9)

RA6T2 (with SCE5\_B Compatibility Mode only)

RA4M1/RA4W1 (with SCE5 Compatibility Mode only)

#### Development tools and software

- e<sup>2</sup> studio IDE v2024-07
- Renesas Flexible Software Package (FSP) v5.5.0
- SEGGER J-Link<sup>®</sup> USB driver and RTT Viewer
- Renesas Flash Programmer (RFP) v3.16
- Renesas Security Key Management Tool v1.0.7

The FSP, J-Link USB drivers, and e<sup>2</sup> studio are bundled in a downloadable platform installer available on the FSP webpage at <u>renesas.com/ra/fsp</u>. SEGGER RTT Viewer is available for download free-of-charge from <u>https://www.segger.com/products/debug-probes/j-link/tools/rtt-viewer/</u>. RFP is available for download from <u>https://www.renesas.com/software-tool/renesas-flash-programmer-programming-gui</u>. The free-of-charge



edition can be used for the functionality required by this Application Project. The Security Key Management Tool can be downloaded at <u>https://www.renesas.com/software-tool/security-key-management-tool</u>.

#### Hardware

- EK-RA8M1, Evaluation Kit for RA8M1 MCU Group (http://www.renesas.com/ra/ek-ra8m1)
- EK-RA6M4, Evaluation Kit for RA6M4 MCU Group (http://www.renesas.com/ra/ek-ra6m4)
- EK-RA6M3, Evaluation Kit for RA6M3 MCU Group (http://www.renesas.com/ra/ek-ra6m3)
- Workstation running Windows<sup>®</sup> 10
- One USB device cable (type-A male to micro-B male)

## **Prerequisites and Intended Audience**

This application note assumes you have some experience with the Renesas e<sup>2</sup> studio IDE and Arm<sup>®</sup>-TrustZone<sup>®</sup>-technology based development models with e<sup>2</sup> studio. In addition, the application note assumes that you have some knowledge of RA Family MCU security features. You can reference the section "Security Features" in the hardware user's manual for background knowledge preparation for the cryptographic key injection. The intended audience are product developers, product manufacturers, product support, or end users who are involved with any stage of injecting or updating secure keys with Renesas RA Family MCUs.



## Contents

1. Wrapped Key Creates Root of Trust	5
1.1 Introduction to Root of Trust	
1.2 Introduction to Security Engine and Associated Keys	5
1.3 Renesas Secure Key Injection Advantages	7
1.3.1 Advantages of Key Wrapping over Key Encryption	7
1.3.2 Advantages of Key Wrapping using MCU HUK	
1.4 Renesas RA MCU Factory Boot Firmware Limitations for SCE9	8
2. Wrapped Key Injection Use Cases and Injection Procedure Overview	9
2.1 Wrapped Key Types	9
2.2 General Steps for Secure Key Injection and Update	9
2.2.1 Key Injection	9
2.2.2 Key Update	
2.3 Overview of the Operations for Evaluating the Example Projects	
2.4 Tools Used in the Secure Key Injection and Update	
3. Using the Renesas Key Wrap Service	14
3.1 Create PGP Key Pair	
3.2 Registration with DLM Server	
3.3 Exchange User and Renesas PGP Public Keys	
4. Wrapping the User Factory Programming Key Using the Renesas Key V	/rap Service23
4.1 Renesas Security Key Management Tool	
4.2 Creating the User Factory Programming Key using the SKMT GUI Inte	face24
4.3 Creating the User Factory Programming Key using the CLI Interface	
4.4 Wrapping the UFPK	
5. Secure Key Injection for SCE9 Protected Mode	
5.1 Wrap Keys with the UFPK and W-UFPK	
5.1.1 Using the SKMT GUI Interface	
5.1.2 Using the SKMT CLI Interface	
5.2 Secure Key Injection via MCU Boot Interface	
5.2.1 Setting up the Hardware	
5.2.2 Inject the Initial User Key and Key-Update Key	
6. Secure Key Injection Preparation for RSIP and SCE7 Compatibility Mod	e51
6.1 Wrap an AES-128 User Key Using the UFPK for RSIP-E51A Compatib	ility Mode51
6.2 Wrap an AES-128 User Key Using the UFPK for SCE7	
7. Example Project for RA6M4 (SCE9 Protected Mode)	
7.1 Example Project Overview	



## Renesas RA Family

7.2.1	Formatting the Injected Keys	. 57
7.2.2	Verifying the Injected Key and the Updated Key	. 58
7.3	FSP Crypto Module Support for User Key Update	. 59
7.3.1	Save the New Wrapped Key to Data Flash	. 60
7.4	Import and Compile the Example Project	. 61
7.5	Running the Example Project	. 61
8.	Example Project for RA8M1 (RSIP Compatibility Mode)	. 63
8.1	Overview	. 63
8.2	Using the SKMT-Generated Files	. 64
8.3	RSIP Compatibility Mode Key Injection APIs	. 64
8.4	Import and Compile the Example Project	. 64
8.5	Running the Example Project	. 64
9.	Example Project for RA6M3 (SCE7 Compatibility Mode)	. 65
9.1	Overview	. 65
9.2	Using the SKMT-Generated Files	. 65
9.3	SCE7 Compatibility Mode Key Injection APIs	. 66
9.4	Import and Compile the Example Project	. 66
9.5	Running the Example Project	. 66
10.	References	. 67
11.	Website and Support	. 68
Revis	sion History	. 69



## 1. Wrapped Key Creates Root of Trust

## **1.1 Introduction to Root of Trust**

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

Secure boot is part of the services provided in the Root of Trust in many security systems. The application is authenticated using public key encryption. The associated keys are part of the Root of Trust in the system. Device Identity, which consists of Device Private Key and Device Certificate, is part of the Root of Trust for many IoT devices.

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

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

## 1.2 Introduction to Security Engine and Associated Keys

The security engine (RSIP, SEC9, SEC7, SCE5, or SCE5\_B) is an isolated subsystem within the MCU. The security engine contains hardware accelerators for symmetric and asymmetric cryptographic algorithms, as well as various hashes and message authentication codes. It also contains a True Random Number Generator (TRNG), providing an entropy source for cryptographic operations. The security engine is protected by an Access Management Circuit, which can shut down the security engine in the event of an illegal external access attempt. Figure 1 shows the conceptual diagram of the security engine. Refer to Table 1 for exactly what cryptographic operations are supported by each type of security engine.



Figure 1. Security Engine Capabilities

The Hardware Root Key (HRK) is not a single key that is physically stored. It is represented in this presentation 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 crypto 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 application keys in wrapped format, encrypted by the MCU's Hardware Unique Key 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 application keys. The RA Family also provides a secure key injection mechanism to securely provision your devices.

The security engine is packed full of cryptography features that you can leverage in your higher-level solutions, giving you the option to use hardware acceleration to reduce both execution time and power consumption. All the security engines offer AES, TRNG, and secure key storage and usage. The SCE7, SCE9, and RSIP expand this by offering both RSA and ECC for PKI solutions. The full complement of SCE9 Protected Mode crypto algorithms plus a selection of SCE7 crypto algorithms are NIST CAVP certified. Table 1 summarizes the different security engines and their associated cryptographic functionalities.

Functions		RA8x1	RA6M4, RA6M5 RA4M2, RA4M3	RA6M1, RA6M2 RA6M3, RA6T1	RA6T2	RA4M1, RA4W1	
		Cryptographic Iso	lation				
	Security Engines	Security Engine	RSIP-E51A	SCE9	SCE7	SCE5_B	SCE5
		Identity & Key Exc	change (Asymmetr	ic)			
	RSA	Key Gen, Sign/Verify	Up to 4K	Up to 4K	Up to 2K	-	-
	ECC	Key Gen, ECDSA, ECDH	Up to 521 bits	Up to 512 bits	Up to 512 bits Up to 384 bits		-
	Ed25519	EdDSA	Y	-	-	-	-
	DSA	Sign/Verify	-	-	Y	-	-
		Privacy (Symmetrie	c)				
		ECB, CBC, CTR	128/192/256	128/192/256	128/192/256	128/256	128/256
		GCTR	128/192/256	128/192/256	128/192/256	-	-
	AES	XTS	128/256	128/256	128/256	-	-
		CCM, GCM, CMAC	128/192/256	128/192/256	128/192/256	128/256	128/256
		Data Integrity				-	
		GHASH	Y	Y	Y	-	-
	Hash	HMAC	SHA224/256/ 384/512	SHA224/256	SHA224/256	-	-
		SHA-2 (224/256)	Y	Y	Y	-	-
		SHA-2 (384/512)	Y	-	-	-	-
	TRNG	HW Entropy, SP800-22A	Y	Y	Y	Y	Y
		Key Handling		·	·	•	
	Wrapped	Confidentiality, authenticity	Y	Y	Y	Y	Y
	Plaintext	Legacy compatibility	Y	Y	Y	Y	Y

Table 1	Ponosas Socuro IP	and Socurity	Engino Cr	votographic	Canabilities
Table I.	Reliesas Secure IP	and Security	Engine Cr	yptographic	Capabilities

The features of the various Security Engines are as follows:

- SCE5 provides hardware-accelerated symmetric encryption for confidentiality. The updated SCE5\_B uses enhanced secure key handling, leveraging an injected MCU-unique HUK.
- SCE7 adds asymmetric encryption and advanced hash functions for integrity and authentication.
- SCE9 expands upon the SCE7 by leveraging an injected MCU-unique HUK for secure key handling and increasing RSA support up to RSA-4K.
- RSIP expands upon the SCE9 by adding advanced cryptographic algorithms like EdDSA and ECC secp521r, SHA384, and SHA512.

The MCU-unique Hardware Unique Key (HUK) is a 256-bit random key for RSIP and SCE9 and a 128-bit random key for SCE5\_B, which is injected into the Renesas factory. This key is stored in a wrapped format using an MCU-unique key wrapping mechanism.

The MCU-unique Hardware Key (HUK) for SCE5 and SCE7 is a derived MCU unique key that serves the same purpose as the HUK for SCE9, RSIP, and SCE5\_B in terms of user key wrapping. The derived HUK for SCE7 and SCE5 is never stored and is accessible only by the SCE and not by application code.

Since, for all the security engines, the HUK is in a wrapped format unique to the MCU, even if an attacker were able to extract the stored key, another MCU would not be able to use it.



## **Renesas RA Family**

All security engines can inject a Key Update Key (KUK), which can be used to securely update the user keys when a device is deployed in the field. The KUKs are injected during end-product manufacturing via the MCU's programming interface or using FSP Crypto Driver. To update keys in a device that is deployed in the field, the new key must be wrapped with one of the previously injected KUKs. In addition to replacing keys that have been compromised, many security policies require key rotation or key update (re-keying) on a regular basis. It is recommended to consider injecting multiple KUKs.

## **1.3 Renesas Secure Key Injection Advantages**

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

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

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



## 1.3.1 Advantages of Key Wrapping over Key Encryption

Figure 2. Key Wrapping versus Key Encryption

It is important to understand the difference between wrapping and encrypting for secure asset storage.

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

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



## 1.3.2 Advantages of Key Wrapping using MCU HUK



Figure 3. Key Wrapping using the HUK

Using the MCU Hardware Unique Key (HUK) to wrap the stored keys adds another protection feature – clone protection. If the wrapped key is transmitted or copied to another MCU, that MCU's HUK 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 cannot be used or exposed.

## 1.4 Renesas RA MCU Factory Boot Firmware Limitations for SCE9

Secure key injection via the serial programming interface is not supported for RSA 3K, RSA 4K, ECC secp256k1, and Key-Update Keys on some older versions of the Renesas RA MCUs due to factory Boot Firmware limitations. The user needs to use a Renesas Flash Programmer (RFP) to read out the Boot Firmware version and confirm the support for the Secure Key Injection of the above-mentioned keys. Refer to the RFP user's manual Flow of Operations section to access the Bootloader Firmware version by using the **Read Device Information** menu.

- V1.2.04 WS1: secure user key inject command is not supported
- V1.3.10 WS2: user key inject command is not supported
- V1.5.22 CS: user key inject command is supported, but it does not support RSA 3K, RSA 4K, secp256k1, or KUK
- V1.6.25 and above MP: no limitations

The part information silkscreened on the device can also be checked, though it is recommended that the boot firmware version be confirmed as described above. Boot firmware limitations exist for the following MCUs:

- RA4M2 All WS and ES devices
- RA4M3 All WS, ES, and CS devices (date code 014AZ00)
- RA6M4 All WS, ES, and CS devices (date code 014AZ00). MP device with date codes 028AZ00, 031AZ00
- RA6M5 All WS and ES devices

Please note that some EK-RA6M4 and EK-RA4M3 Evaluation Kits may contain affected silicon. The following list shows the serial numbers of the affected kit. Note that all early adopter kits with WS or ES silicon are also affected.

- EK-RA4M3 Serial numbers 219243 219542
- EK-RA6M4 Serial numbers 215938 216237 and 218497 218996

If your application requires secure key injection of RSA 3K, RSA 4K, ECC secp256k1, or Key-Update Keys and your evaluation kit does not support it, please contact your local Renesas Sales representative.



## 2. Wrapped Key Injection Use Cases and Injection Procedure Overview

This section provides an overview of the wrapped key injection use cases and the general steps for the injection procedure of each use case. A step-by-step walkthrough of the wrapped key injection procedures is provided in later sections.

## 2.1 Wrapped Key Types

Table 2 summarizes the key types that can be directly injected into Renesas RA Family MCUs with the RSIP security engine. Injected keys are stored and wrapped by the MCU's HUK.

Table 2.	Supported Key Types for RSIP
----------	------------------------------

Lifecycle Transition Keys	SECDBG_KEY, NONSECDBG_KEY, RMA_KEY
AES	AES-128, AES-192, AES-256
RSA	RSA-1024, RSA-2048, RSA-3072, RSA-4096 (Public and Private)
ECC	secp192r1 (NIST P-192), secp224r1 (NIST P-224) (Public and Private)
	secp256r1 (NIST P-256), secp384r1 (NIST P-384) (Public and Private)
	secp256k1 (Public and Private)
	Brainpool P256r1, P384r1, and P512r1 (Public and Private)
HMAC	HMAC-SHA224, HMAC-SHA256
Utility Keys	Key-Update Keys

See Table 1 to understand the types of keys supported for other security engines based on the supported crypto algorithms and Device Lifecycle Management capability.

## 2.2 General Steps for Secure Key Injection and Update

Secure Key Injection for RSIP, SCE9 Protected Mode, and SCE5\_B is performed via the MCU boot interface, demonstrated here with the Renesas Flash Programmer (RFP). Secure Key Injection for RSIP, SCE9 Compatibility Mode, SCE7, and SCE5 compatibility mode is performed through the FSP. Key preparation steps where key material is exposed in plaintext must be performed in a secure environment.

## 2.2.1 Key Injection

There are three high-level steps for key injection. Section 3 guides the user in establishing the PGP encrypted communication channel between the user and the Renesas DLM Server. Sections 4, 5, and 5.2 provides step-by-step walkthroughs of how to perform the three high-level steps for the secure key injection.

1. The first step in the secure key injection process is to use the Renesas Device Lifecycle Management (DLM) service to wrap an arbitrary User Factory Programming Key (UFPK) (in green) using the Renesas Hardware Root Key (HRK) (in blue). The UFPK is a 256-bit value selected by the user. The same UFPK can be used to inject any number of keys.



Figure 4. Wrapping the UFPK using DLM Server



#### 2. Next, the user key (in yellow) must be wrapped with the UFPK.





3. Finally, the user key is injected by providing the wrapped UFPK (W-UFPK) and the wrapped user key to the secure key injection mechanism of the security engine. Note that this is a conceptual representation of the secure user key injection. Once the wrapped user key is generated, the tool generates one secure key injection file, which includes the wrapped user key and the W-UFPK. This file (.rkey file) will be used in the secure key injection project.

For the protected mode, injecting the wrapped user key should be performed using the MCU boot interface. For compatibility mode, injecting the wrapped user key should be performed using the FSP key injection PSA Crypto API.



Figure 6. Inject User Key over the Serial Programing Interface

## 2.2.2 Key Update

Since injecting new keys in the field is usually done to replace older keys (key rotation or re-keying), this process is referred to as a "key update". To enable secure key updates in the field, one or more Key-Update Keys (KUK) must be injected during production programming/provisioning, as described above.

KUKs, like other cryptographic keys, can be stored in either code flash or data flash (if available on the MCU). Injection of the KUK uses the same procedure as injecting other user keys, as described in the section 2.2.1. Since the KUK is the only mechanism by which new keys can be injected/wrapped, it is highly recommended that multiple KUKs be injected during production provisioning. This enables the KUK to be rotated or revoked to adhere to an infrastructure security policy or to respond to a key exposure security breach.

For MCUs that support secure key injection over the MCU boot interface, additional KUKs CANNOT be injected after the programming interface is disabled. Once a product is in the field with its programming interface disabled, new keys can ONLY be injected via a pre-existing KUK.

The KUKs may be stored in any code or data flash location during production. This location will be passed to the key update API 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, we recommend that users disable the programming interface prior to deploying to the field.

There are two high-level steps for key updates. Note that the KUK must already reside in the MCU.

1. Use the KUK (in grey) to wrap the new user key (in yellow).





Figure 7. Wrap the New User Key with a KUK

2. Use the FSP and the previously injected KUK to inject the new user key. The new user key is wrapped by the MCU HUK (in black). Note that the APIs for the two modes are provided by different FSP modules.



Figure 8. Update the User Key

## 2.3 Overview of the Operations for Evaluating the Example Projects

The example projects in this application project demonstrate the secure key injection and update capabilities of Renesas RA Family MCUs using sample keys. Sections 3, 4, and 5 describe the steps needed to replace these sample keys with custom keys.

The following graphic shows the flow of this preparation work, plus the example project for SCE9 (RA6M4 example). The block outlined in red is the scope of the functionality of the example project.



OEM	Rer	nesas (DLM Server)	
Generate PGP key pair		Register with DLM	
OEM PGP public key		Store OEM PGP public	
◀ Store Renesas PGP public key	Renesas PGP public key	кеу	PGP Key Exchange
Generate UFPK file using skmt.exe			Wranning Llear Factory
Renesas PGP public key encrypted UFPK	with OEM PGP public key	Create HRK Wrapped UFPK (W-UFPK)	Programming Key
Generate an initial plaintext AES 256 user wrap it with UFPK using skmt.exe	r key and		
Generate a plaintext Key Update Key and with UFPK using skmt.exe	l wrap it		
Generate a new plaintext AES256 user ke wrap the new user key with the Key Upda using skmt.exe	ey and ate Key		
Inject the initial AES256 user key and the Update Key to MCU using RFP	Кеу		
Verify that the cryptographic operations with the RFP injected AES256 key. Update with the new AES256 user key using the F APIs. Verify that the cryptographic operat well with the updated AES256 key	work well e the MCU FSP Crypto tions work		

## Figure 9. Operational Flow Injecting and Updating an AES-256 Key for SCE9 Protected Mode

The following graphic shows the flow of this preparation work plus the example project for SCE7 (RA6M3 example). The block outlined in red is the scope of the functionality of the example project.



	F	Renesas (DLM Server)	
Generate PGP key pair		Register with DLM	
OEM PGP public key		Store OEM PGP public	-   
€ Store Renesas PGP public key	— Renesas PGP public key	кеу 	PGP Key Exchange
Generate UFPK file using skmt.exe			
Renesas PGP public key encrypted UF	РК	→ Create HRK Wrapped	Wrapping User Facto Programming Key
◀ W-UFPK encrypte	ed with Customer PGP public ke	UFPK (W-UFPK)	
Generate an initial plaintext AES 128	8 user key and		
encrypt it with UFPK using skmt.exe Generate a plaintext Key Update Ke encrypt it with UFPK using skmt.exe	y and		
encrypt it with UFPK using skmt.exe Generate a plaintext Key Update Ke encrypt it with UFPK using skmt.exe Generate a new plaintext AES128 us encrypt the new user key with the K Key using skmt.exe	y and ser key and Sey Update		
encrypt it with UFPK using skmt.exe Generate a plaintext Key Update Ke encrypt it with UFPK using skmt.exe Generate a new plaintext AES128 us encrypt the new user key with the K Key using skmt.exe Inject the initial AES128 user key to using FSP Key Injection for PSA Cryp	y and ser key and sey Update the MCU to API		

Figure 10. Operational Flow Injecting User Keys for SCE7 and RSIP-E51A Compatibility Mode

## 2.4 Tools Used in the Secure Key Injection and Update

There are three tools used in the secure key injection and update besides e<sup>2</sup> studio, which is used as the software project development environment. Refer to the corresponding section mentioned below for details on obtaining, setting up, and using these tools.

#### • Gpg4win

This tool is used in section 3 to establish a PGP encrypted communication channel between user and the Renesas Key Wrap server. Using this tool, the user can generate a user PGP key pair, perform key exchange with the Renesas DLM server, and assist the reception of the W-UFPK.

## Renesas Security Key Management Tool (SKMT)

This tool is used in section 4, section 5 and section 6 to generate the following three key files:

- User key: to be injected to MCU via RFP or FSP API
- Key update key: to be injected to MCU via RFP
- New user key wrapped using the KUK: to be updated by an FSP API

#### • Renesas Flash Programmer (RFP)

This tool is used in section 5.2 to inject the User key and KUK when using the security engine Protected Mode.



## 3. Using the Renesas Key Wrap Service

The Renesas Key Wrap Service must be used to obtain a wrapped UFPK (W-UFPK) for the specific MCU Group and security engine operational mode. All key material exchange is performed with PGP encryption. This section explains the steps to establish this PGP-encrypted communication channel between the user and the Renesas Key Wrap Server. This is a one-time process and does not need to be repeated for different MCUs.

## 3.1 Create PGP Key Pair

If you already have a PGP key pair, that key can be used for the key exchange process. Otherwise, the instructions below describe one method for creating a PGP key pair.

The PGP software demonstrated here is GPG4Win, which can be downloaded from this URL: <a href="http://www.gpg4win.org/">http://www.gpg4win.org/</a>

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 Kleopatra:



Figure 11. Download and Install Kleopatra

Launch Kleopatra and create a PGP Key Pair.

- 1. Click File > New Key Pair
- 2. Choose Create a personal OpenPGP key pair.

Please choose which type you want to create.	
<ul> <li>Create a personal OpenPGP</li> <li>OpenPGP key pairs are certified by control</li> </ul>	key pair nfirming the fingerprint of the public key.
→ Create a personal X.509 key X.509 key pairs are certified by a certif request needs to be sent to a CA to fi	pair and certification request ication authority (CA). The generated nalize creation.
	Next Cancel

Figure 12. Create a Personal Open PGP Key Pair



3. Provide a Name and Email. Note that even though these are marked as optional, at least one entity must be provided to move to the next stage. Check Protect the generated key with a passphase.

Enter Details Please enter your personal details below. If you want more control over the parameters, click on the Advanced Settings button.	
Name       secure_key       (optional)         EMail:       customer@company.com       (optional)         Image: Company Compan	
secure_key <customer@company.com> Advanced Settings</customer@company.com>	
Create Cancel	

Figure 13. Provide Name and Email

4. Click Advanced Settings and select RSA as the key type.

👦 Advanced Setti	ings - Kleopatra	? ×
Technical Details		
Key Haterial		
RSA	3,072 bits $\sim$	
🗹 + RSA	3,072 bits $\sim$	
O DSA	2,048 bits $\sim$	
🗌 + Elgan	nal 2,048 bits $\sim$	
O ECDSA/EdD	OSA ed25519 ~	
+ ECDH	cv25519 🗸	
Certificate Usag	je	
Signing	Certification	
Encryption	Authentication	
✓ Valid until:	1/13/2024	~
	ОК	Cancel

Figure 14. Select RSA Encryption



5. Click **Create** and provide a passphrase twice to protect the private key. Then click **OK**. **Be sure to save** your passphrase.

← Creating The process several minu	? × Key Pair of creating a key requires large amounts of random numbers. This may require tes
	Please enter the passphrase to protect your new key Passphrase: Repeat: OK Cancel
	Next Cancel

Figure 15. Define a Passphrase

6. The PGP key pair should be created successfully. Click **Finish**.

? ×	<
Key Pair Creation Wizard	
Key Pair Successfully Created	
Your new key pair was created successfully. Please find details on the result and some suggested next steps below.	
Result	
Key pair created successfullv. Fingerprint: 0123456789abcdef0123456789abcdef01234567	
Next Steps	
Make a Backup Of Your Key Pair	
Send Public Key By EMail	
Upload Public Key To Directory Service	
Finish Cancel	

## Figure 16. PGP Key Pair Created

7. A new item will be created in Kleopatra. Right-click on the keypair just created and select Export.

D.	w/	<b>A</b> =1		<b>A</b> =1	0		Ē	E	1			
Xá Sign/Encrypt	Liq Decrypt/Verify	Import	Export	Certify	Lookup on Server	Certificates		Smarte	ards			
Search <alt< td=""><td>:+Q&gt;</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td>All Ce</td></alt<>	:+Q>											All Ce
	Name				E-Mail		User-IDs	Valio	From	Valid Until	Key-ID	
secure_	key		cı	ustomer@	company.com		certified	- (BE	Certifi		0533	
								arr	Revok	, e Certificatio	on	
									Trust	Root Certifica	ate	
									Distru	ist Root Certi	ificate	
									Chang	ge Certificatio	on Trust	
									Chanc	ne Expiry Dat	e	
									Chanc	ge Passphras	e	
									Add L	Jser-ID		
								ī.	Delete	e		Del
								E,	Expor	t		Ctrl+E
									Backu	p Secret Key	S	
								Ţ.	Print S	Secret Key		
								E,	Publis	h on Server		Ctrl+Shift+E
								-	Detail	s		
								_				





8. Save the public key to a file with an \*.asc extension. In this example, this file is renamed to customer public.asc. Click **Save**.

$\leftrightarrow \rightarrow \uparrow \uparrow $ (C:) Windo	ws > secure_key_injection	5 V	$\mathcal{P}$ Search se	ecure_key
Organize 🔻 New folder				::: • ?
<ul> <li>Downloads</li> <li>Music</li> <li>Pictures</li> <li>Videos</li> <li>(C) Windows</li> </ul>	^ Name	∧ No items match y	Date modified	Туре
Network	v <			
File name: customer_put	olic.asc			~

Figure 18. Save the PGP Public Key to a Folder

## 3.2 Registration with DLM Server

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

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

Login screen of Key Wrap service	
E-mail address: Email address	
Password: Password	
Login	
New registration If you forgot your password	
	Login screen of Key Wrap service E-mail address: Email address Password: Password Login I you forgot your password

Figure 19. Start Registration with Renesas DLM Server

2. Follow the prompt to provide a **valid** email address and click **Send mail**.

Please make sure that you can receive e-mail from the domain "@renesas.com".	
E-mail address : <b>customer@company.com</b> Send mail Return	

Figure 20. Register User Email Address



#### After clicking **Send mail**, the following screen will appear. Click **Return**.

The e-mail has been sent.	
E-mail address : pcustomer@company.com	
Click on the link in the e-mail, and register your information.	
Unless you have registered within three hours, the link expires, so re-start the process from registration of your e-mail address.	
Return	

#### Figure 21. Acknowledge Email Transmission

## 3. You should receive an email similar to the one shown below. Click on the URL provided to confirm your registration.

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.
https://dlm.renesas.com/keywrap/?menu=reg%2Fregist%2Ffixed%2F&param=id%3Duf2SL2o3Cd7ODijiEN%252BOmISX7qhfiv8H9mP%252F1gQ7aH8%253D
<ul> <li>Notes</li> <li>Please register your information within three hours of receiving this e-mail.</li> <li>The URL expires after the three hours, after which you will need to start the registration process anew.</li> </ul>
Please delete this email if you were not aware that you were going to receive it.
<ul> <li>* This email was sent from a send-only address.</li> <li>Please understand that there will be no response, even if you reply to this address.</li> <li>* 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.</li> </ul>
Thank you.
Renesas Electronics Corporation

#### Figure 22. Registration Confirmation Email

4. Follow the prompts to provide your name and company name and create a password. Click 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 registere The password is from 8 to 32 cha the symbols "!" "@".	d. Enter all of the following items. aracters, which must be single-byte, and may include
E-mail address :	customer@company.com
Name	customer
Company Name	company
Password	
Re-enter your password	
	Next (confirmation)

Figure 23. Confirm Registration



After the confirmation screen is displayed, click on the **Register** button to complete the user registration.

The following items will be registe	ered. Are you sure?	
E-mail address :	customer@company.com	
Name :	customer	
Company Name :	company	
Password :	•••••	
Re-enter your password :	•••••	
	Register	

Figure 24. Finish the Registration

## 3.3 Exchange User and Renesas PGP Public Keys

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

1. After successfully registering the user information, the following screen will open. Click the **Start service** button to start using the key encryption system.

Registered		
E-mail address :	customer@company.com	
Name :	customer	
Company Name :	company	
	Start service	



2. When the agreement warning shows up, scroll down to the bottom of the **Trusted Secure IP Key Wrap Agreement** and click **I agree**. You will then be logged into the DLM server. Note that the Agreement will come up every time you log into the DLM server.

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 26. Agreement for Using the Renesas DLM Server



#### 3. When you log into the DLM system, the window below appears. Click **PGP key exchange**.

PGP key exchange	Display history	Refresh screen
Your PGP key has not been	exchanged yet. Start by excha	anging your PGP key.
RENESAS	The RZ fami	ly users.
RENESAS	The RX fami	ly users.
RENESAS	The RE fami	ly users.
RENESAS	The RA fami	ly users.

Figure 27. Start PGP Key Exchange

4. Click **Reference** and select the public key generated earlier (customer\_public.asc). Notice 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.

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 1 F61 20E9 9613 8DF7 F517 189C 1 EA5 E55D

#### Figure 28. Browse the Customer PGP Public Key

#### 5. Click the PGP key exchange.

PGP key exchange screen customer is already logged-in.	
Return	
Select your PGP public key that ex Your public key will be sent to Rene C¥fakepath¥customer_public.asc	<pre>sported format, and click on "PGP key exchange" button. esas, and the PGP public key of Renesas will be sent to your e-mail address. </pre> Reference PGP key exchange
The fingerprint of PGP public key o FB18 EB66 1F61 20E9 9613 8DF7 F517 189C 1EA5 E55D	if Renesas is below.

Figure 29. Exchange Keys



#### 6. Once the PGP public key is submitted, click Return.

Return
Your PGP public key submit is being processed in your application for registration.
After completion of registration, registration-completion e-mail to which the PGP public key of Renesas is attached will be sent. Please wait for a while.

#### Figure 30. Wait for Renesas's PGP Public Key

7. You will receive an email from Renesas at the email address registered with the DLM server with the contents as shown below if the key exchange is successful. It typically takes about one to two minutes to receive this email.

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 31. Receive the Renesas PGP Public Key

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

8. Go back to the Kleopatra application and import the Renesas PGP Public key to Kleopatra as shown below.

			^ Name ^ Dat
			customer_public.asc 1/1
			keywrap-pub.key 1/1
	Klassatur		
	Rieopatra		✓ <
File	View Certificates	Tools Settings Ctrl+N	:ey Date modified: 1/15/2021 11:30 PM Size: 3.11 KB
Q	Lookup on Server	Ctrl+Shift+I	
	Import	Ctrl+I	me: keywrap-pub.key 🗸 Any files (*)
	mporem		

Figure 32. Import Renesas Public Key



9. After **Open** is clicked, a new item is added in Kleopatra as **not certified**.

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

Figure 33. Renesas Public Key is Imported

10. Confirm that the Fingerprint displayed is the same as what is shown on the screen represented in Figure 29. Click **Certify**.

customer_public.asc	1/13/2022 3:51 PM	OpenPGP Text File	3 KB				
keywrap-pub.key	1/13/2022 3:52 PM	KEY File	4 KB				
inning Tool						- п х	
dit Tools Help							
C Kleonatra							
M Gentificates Tasla	Cattings Window Hala						
- File View Certificates Tools	Settings Window Help						_
		Q 🗉					
Sign/Encrypt Decrypt/Verify	Import Export Certify L	ookup on Server Certificates	Notepad Smartcards				
Search < Alt+0>						All Certificates	
All Certificates Imported	Certificates				(	An Certificates	
Name	E-Mail	Urar-ID	r Valid From Valid	Intil	KavalD		
kanuran	tomer-key-encryntion-system@lu	m reneras com not certified	10/23/2018		Key-ID		
secure key cu	stomer@company.com	certified	1/13/2022 1/13/	20	Certify		
- 7				<b></b>	Revoke Certification		
					Trust Root Certificate		
					Distrust Root Certificate		
					Change Certification Tru:	st	
					Change Expiry Date		
-					Change Parcohrace		
					Add User-ID		
					Add Osci ID		_
				- É	Delete	Del	
				<b>e</b> ,	Export	Ctrl+E	
					Backup Secret Keys		
				7	Print Secret Key		
				<b>1</b> 55	Publish on Server	Ctrl+Shift+E	
					Details		

Figure 34. Confirm the Fingerprint and Certify the Renesas Public Key

11. Click Certify again from the following screen.

Fingerprint: I	FB18 EB66 1F61 20E9 9613 8DF7 F517 189C 1EA5 E55D	
Only the fing	erprint clearly identifies the key and its owner.	
Certify with:	secure_key <customer@company.com> (certified, created: 1/13/2022)</customer@company.com>	$\sim$
🗹 keywra	p <customer-key-encryption-system@lm.renesas.com></customer-key-encryption-system@lm.renesas.com>	
Advance	d	

Figure 35. Certify the Certificate



#### 12. Provide the passphrase to unlock the secure key.



Figure 36. Provide the Passphrase

#### 13. The following item will pop up upon successful certification. Click OK.

	👦 Certificati ? 🗙			
	Certification successful.	]		
	<b>↓</b>			
Search <alt+q> T Imported Certificates</alt+q>				
Search <alt+q> Timported Certificates Name</alt+q>	E-Mail	User-IDs Valid Fron	n Valid Until	Key-ID

Figure 37. Successful Certification

# 4. Wrapping the User Factory Programming Key Using the Renesas Key Wrap Service

If you do not already have a W-UFPK for your target MCU Group, follow the steps below to wrap a UFPK with the Renesas Hardware Root Key as described by Figure 4.

#### 4.1 Renesas Security Key Management Tool

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

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

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

All Types 🔹	Start typing to filter results by title Q	
Туре 🗢	Title 🔶	Date 🔶
Software & Tools - Other	Security Key Management Tool V1.07 for macOS 合 ZIP 209.92 MB <u>日本語</u>	Aug 30, 2024
Software & Tools - Other	Security Key Management Tool V1.07 for Windows 合 ZIP 150.53 MB <u>日本語</u>	Aug 30, 2024
Software & Tools - Other	Security Key Management Tool V1.07 for Linux	Aug 30, 2024

Figure 38. Download the Security Key Management Tool for Windows, Linux or macOS



Once the installer executable is downloaded, right-click on the installer and select **Run as administrator** to install this tool. Follow the prompt to select the **Setup Language**. Currently, both English and Japanese are supported. Next, select the installation folder. By default, it will be installed into

C:\Renesas\SecurityKeyManagementTool\. If a previous version is installed, the old version will be overwritten.

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

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

## 4.2 Creating the User Factory Programming Key using the SKMT GUI Interface

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

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

Renesas > SecurityKeyMangementTool

 Name

 CLI

 configuration

 DOC

 plugins

 workspace

 SecurityKeyManagementTool.exe

 SecurityKeyManagementTool.ini

 unins000.dat

 Luins000.exe

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

Figure 39. Launch SKMT GUI Interface

To use the example projects included 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.



 RA8M1 has RSIP-E51A, for the RA8M1 compatibility mode example project included, in the Overview window, select RA Family, RSIP-E51A Compatibility Mode.



Figure 40. Select RA Family, RSIP-E51A Compatibility Mode

 RA6M4 has SCE9, for the protected mode example project included, in the Overview window, select RA Family, SCE9 Security Functions, and Protected Mode.

	RENESAS
	Security Key Management Tool
This tool is designed to	assist in the preparation of application and Device Lifecycle Management (DLM) keys for secure injection and update.
Keys are securely inje	ted via a User Factory Programming Key (UFPK), which must be wrapped by the Renesas Key Wrap Service to obtain a wrapped UFPK (W-UFPK).
Keys are se	ecurely updated via a Key-Update Key (KUK), which must be securely injected.
Please refer to the spe	cific MCU/MPU documentation for more information about supported security features.
Select MCU/MPU and	security engine : RA Family, SCE9 Security Functions and Protected Mode 🗸 🗸
	Please select the target MCU or MPU before continuing.

Figure 41. Select RA Family, SCE9 Protected Mode



• RA6M3 has SCE7, for the SCE7 example project included, in the **Overview** window, select **RA Family**, **SCE7** 

	RENESAS
	Security Key Management Tool
This tool is designed to	assist in the preparation of application and Device Lifecycle Management (DLM) keys for secure injection and update.
Keys are securely injec	ted via a User Factory Programming Key (UFPK), which must be wrapped by the Renesas Key Wrap Service to obtain a wrapped UFPK (W-UFPK).
Keys are se	curely updated via a Key-Update Key (KUK), which must be securely injected.
Please refer to the spe	cific MCU/MPU documentation for more information about supported security features.
Select MCU/MPU and s	ecurity engine : RA Family, SCE7 🗸 🗸 🗸
	Please select the target MCU or MPU before continuing.

Figure 42. Select RA Family, SCE7



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

- For the User Factory Programming Key, select Use specified value.
- Click the **Browse** button to select a folder to store the key and name the resulting file. It is recommended that users choose different file names for the different MCU families to avoid confusion at the UFPK wrapping stage. In this example, we name the file ra8x1\_ufpk.key.

Click Generate UFPK key file. The  $ra8x1\_ufpk.key$  file will be generated. Similarly, the UFPK for RA6M4 and RA6M3 can also be generated.

Comment II									
Verview Generate U	Generate KUK	Wrap Key	TSIP UPDATE	FSBL	DOTF	SFP			
A User Factory Pro The UFPK must be w	gramming Key (UFP) applicatior rapped by the Renes	() is used to keys durin as Key Wrap	securely inject g production p service and th	t Device L programm en used t	ifecycle N iing. o prepare	Managen e keys for	nent (E r secur	DLM) an e injecti	d on
User Factory Prog	ramming Key								
Generate random	value				1				
Use specified valu	e (32 hex bytes, big e	ndian form	at)						
00010203040506070	8090A0B0C0D0E0F000	01020304050	)60708090a0b0	c0d0e0f					
	Send the generate <u>https://</u> to obtain	Generate d UFPK key (dlm.renesa the wrappe	EUFPK key file file to the Rene s.com/keywra ed UFPK (W-UF	esas Key V p <u>/</u> PK).	Vrap servi	ice			

#### Figure 43. Generate Fixed UFPK using GUI for RSIP-E51A

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



## 4.3 Creating the User Factory Programming Key using the CLI Interface

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

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

skmt.exe /genufpk /output "C:\User\_key\_injection\_protected\_mode\keys\ufpk.key"

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

```
UFPK: E8AB23E99C9AD42823DA4215549A41496720F7243680A4715F4B944ACC94B691
Output File: C:\User key injection protected mode\keys\ufpk.key
```

### Figure 44. Create a Random UFPK Using SKMT CLI

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

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

UFPK: 000102030405060708090A0A0C0D0E0F000102030405060708090a0b0c0d0e0f Output File: C:\User\_key\_injection\_protected\_mode\keys\ufpk.key

### Figure 45. Create a Fixed UFPK Using SKMT CLI

## 4.4 Wrapping the UFPK

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

- 1. Launch the Kleopatra program.
- 2. Encrypt the UFPK with the Renesas public key. This key was imported earlier to Kleopatra. Using Kleopatra, select Sign/Encrypt... and select the UFPK file. In this screenshot, an example file named ufpk ra6m3.key file is used for demonstration purposes. Then click **Open**.



Figure 46. Encrypt the UFPK File for PGP Transfer



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

Sign / Encryp  Prove authenticity (sign	ר Files ח)	
☑ Sign as:	Xianghui Wang <xianghui.wang@renesas.com> (certified, created: 8/16/2023)</xianghui.wang@renesas.com>	~
Encrypt	-	
Encrypt for me:	✓ Xianghui Wang <xianghui.wang@renesas.com> (certified, created: 8/16/2023)</xianghui.wang@renesas.com>	~
✓ Encrypt for others:	S Please enter a name or email address	
Encrypt for others:     Encrypt with passwer	Please enter a name or email address	
Encrypt for others:  Encrypt with passw Output Output Output files/folder:	Please enter a name or email address	
<ul> <li>Encrypt for others:</li> <li>Encrypt with passw</li> <li>Output</li> <li>Output files/folder:</li> <li>C:/Secure_Key_Inje</li> </ul>	Please enter a name or email address ord. Anyone you share the password with can read the data. ction/ra6m3_key_info	

Figure 47. Select PGP Encryption Options

4. Click the Open Selection Dialog (the <u>st</u> icon). This will open a Certificate Selection dialog box.

Sign / Encrypt Files	
Prove authenticity (sign)	
Sign as: Sign as: Sign as:	~
Encrypt	
Encrypt for me: secure_key <customer@company.com> (certified, created: 4/11/2022)</customer@company.com>	$\sim$
Encrypt for others:	
Open selection dialog.	

#### Figure 48. Open the Selection Dialog

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

🔊 Kleop	patra											
File	View	Certificates	Tools	Settings	Windo	w Help						
[				∎Ę.			Q	<b>a</b> =	Ē			
Sign/E	Encrypt	Decrypt/Veri	ify J	Import	Export	Certify	Lookup on Server.	Certificat	es Notepad	Smartcards		
Sea	rch <alt< td=""><td>t+Q&gt;</td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td>All Certificat</td></alt<>	t+Q>										All Certificat
Sea	arch <alt< td=""><td>t+Q&gt; Name</td><td></td><td></td><td></td><td>E-Mail</td><td></td><td>User-IDs</td><td>Valid From</td><td>Valid Until</td><td>Key-ID</td><td>All Certificat</td></alt<>	t+Q> Name				E-Mail		User-IDs	Valid From	Valid Until	Key-ID	All Certificat
Sea	irch <alt< td=""><td>t+Q&gt; Name</td><td></td><td>customer-</td><td>key-encry</td><td>E-Mail ption-syste</td><td>em@lm.renesas</td><td>User-IDs certified</td><td>Valid From 10/23/2018</td><td>Valid Until</td><td>Key-ID F517 189C 1EA5</td><td>All Certificat</td></alt<>	t+Q> Name		customer-	key-encry	E-Mail ption-syste	em@lm.renesas	User-IDs certified	Valid From 10/23/2018	Valid Until	Key-ID F517 189C 1EA5	All Certificat

Figure 49. Select the Renesas PGP Public key



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

Prove authenticity (sig	n)
☑ Sign as:	✓ Xianghui Wang <xianghui.wang@renesas.com> (certified, created: 8/16/2023) ×</xianghui.wang@renesas.com>
Encrypt	
Encrypt for me:	Xianghui Wang <xianghui.wang@renesas.com> (certified, created: 8/16/2023)</xianghui.wang@renesas.com>
☑ Encrypt for others:	customer-key-encryption-system@lm.renesas.com> (certified, OpenPGP, created: 10/23/2018)
	-
	Please enter a name or email address
<ul> <li>Encrypt with passw</li> <li>Output</li> <li>Output files/folder:</li> </ul>	Please enter a name or email address
Encrypt with passw Output Output C:/Secure_Key_Inju	Please enter a name or email address  vord. Anyone you share the password with can read the data.  ection/ra6m3_key_info
<ul> <li>Encrypt with passw</li> <li>Output</li> <li>Output files/folder:</li> <li>C:/Secure_Key_Inju</li> <li>Encrypt / Sign each</li> </ul>	Please enter a name or email address  rord. Anyone you share the password with can read the data.  ecction/ra6m3_key_info  file separately.

Figure 50. Encrypt UFPK using Renesas PGP Public Key

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



Figure 51. Start the UFPK Encryption process

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

🔒 piner	itry-qt	_		×
	Please enter the passphrase to unlock "secure_key <customer@company.cor 3072-bit RSA key, ID A58C68B480A1B created 2022-01-13. Passphrase:</customer@company.cor 	the OpenP n>" 869, DK	GP secre Canc	et key:

Figure 52. Provide Passphrase



9. The UFPK encrypted with the Renesas public key will be generated, with the.gpg added to the extension of the key. In this case, the file ufpk ra6m3.key.gpg is generated. Click Finish.

📅 Sign/Encrypt Files - Kleopatra ? X	
Results Status and progress of the crypto operations is shown here.	_
OpenPGP: All operations completed.	
$ufpk\_ra6m3.key \rightarrow ufpk\_ra6m3.key.gpg: \textbf{Signing and encryption succeeded.}$	
Finish Cancel	

Figure 53. Encrypted Key is Generated

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

PGP key exchange Display history Refresh screen
The RZ family users.
The RX family users.
The RE family users.
The RA family users.

Figure 54. Select the MCU Family

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

• To create a W-UFPK for the RA8M1 Compatible Mode secure key injection example project, select the Renesas RA Family and click Compatibility Mode **RA8M1 Encryption of customer's data**.





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

DLM and Protected Mode Installation via a device programmer, for DLM keys or use with the FSP Crypto APIs **RA6M4/RA6M5** Encryption of customer's data



#### Figure 56. Select the RA6M4/RA6M5 MCU Group DLM and Protected Mode

• To create a W-UFPK for the RA6M3 example project, select the **Renesas RA** Family and click Compatibility Mode **RA6M1/RA6M2/RA6M3/RA6T1 Encryption of customer's data**.



#### Figure 57. Select the RA6M1/RA6M2/RA6M3/RA6T1 MCU Group Compatibility Mode

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

RA6M1/RA6M2/RA6M3/RA6T1 Customer data selection screen Encryption service for products The Key2 (customer's key) generated by you will be encrypted by "HRK<sub>N</sub>" (the

The Key2 (customer's key) generated by you will be encrypted by "HRK<sub>M</sub>" (the embedded key of RA6M1/RA6M2/RA6M3/RA6T1 chip) and sent to you.

#### Figure 58. Choose Encryption service for products

12. Click Reference and select the corresponding encrypted UFPK; example shown is ufpk\_ra6m3.key.gpg created previously, and click Open. Note that in the DLM server description, Key2 refers to the UFPK.

Use our PGP publi Specify the file for	c key to encrypt Key2 that you have made as the customer key. encryption by using the browse button, and click on the OK button.
	Reference Settle
	ecure_Key_I > ra6m3_key_info v V Search ra6
	der Name archive Lectronics Cor; Electronics Cor;
	ame: ufpk_ra6m3.key.gpg   All files (*.*)  Open

Figure 59. Select the PGP-Encrypted UFPK file

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



Figure 60. Return to the DLM Server Main Menu



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

KeyWrap service <customer-key-encryption-system@lm renesas.com=""></customer-key-encryption-system@lm>	$\odot$ $\leftarrow$
	L L
ufpk_ra6m3.key_enc.key.pgp ↓ 1 KB	
ro xyz	
Thank you for using the KeyWrap service. We have sent the encrypted data as an attachment. Save the attached file, and proceed with $\mathbb{P}($	GP decryption.
Product name: RA6M1/RA6M2/RA6M3/RA6T1 Processing mode: Products mode	
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.	

## Figure 61. Receiving the W-UFPK via Email

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



Figure 62. Decrypt the W-UFPK

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

7 Decrypt/Verify Files - Kleopatra		? ×
Output folder: C:/Secure_Key_Injection	on/ra6m3_key_infd	
All operations completed.		
		100%
ufpk_ra6m3.key_enc.key.pgp → n <b>Note:</b> You cannot be sure who e Recipient: customer@	ufpk_ra6m3.key_enc.key: <b>Decryption succeeded</b> . encrypted this message as it is not signed. ecompany.com	Show Audit Log
		Save All Discard

## Figure 63. Decrypting the Encrypted W-UFPK

**17.** Click **Save All** to save the decrypted W-UFPK key file ufpk\_ra6m3.key\_enc.key to the same folder as the UFPK key file. Both key files are required to generate key injection bundles.



## 5. Secure Key Injection for SCE9 Protected Mode

## 5.1 Wrap Keys with the UFPK and W-UFPK

This section walks the user through the wrapping process required for secure key injection and update. The SKMT tool is used to perform this key-wrapping process.

Step-by-step instructions for generating the three types of keys are provided using both the CLI and GUI interfaces of the SKMT.

- **User Key** wrapping with the UFPK for secure key injection of the user key
- Key-Update Key wrapping with the UFPK for secure key injection of the KUK
- User Key wrapping with the KUK for secure key update of the user key

This application project provides examples of user key wrapping of both AES-256 and ECC secp256r1 public keys.

## 5.1.1 Using the SKMT GUI Interface

To prepare a Protected Mode user key to inject using RFP, we need the UFPK, W-UFPK, and the user key as input to the SKMT GUI interface.

Launch the SKMT GUI and select **RA Family, SCE9 Security Functions, and Protected Mode** on the **Overview** tab. On the **Wrap Key** tab, open the submenu **Key Type**. This page can be used to choose which key type to prepare.

## 5.1.1.1 Wrap an Initial AES-256 Key with the UFPK

A NIST CAVP test vector is used for this purpose.

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

## Figure 64. NIST AES 256 Test Vector

In the Key Type area, choose Key Type and specify AES with 256 bits.



Figure 65. Choose AES 256 bits as the Key Type

Navigate to the **Key Data** page and input the **Raw** key data as shown below based on the NIST vector shown in Figure 64. The key data is duplicated here to easily copy and paste to the GUI interface.



Overview Gene	erate UFPK Generate KUK Wrap Key TSIP UPDATE FSBL DOTF SFP
Ke	ys must be wrapped by the UFPK for secure injection or by the KUK for secure update.
Key Type Key	Data
⊖ File	Browse
Raw	800000000000000000000000000000000000000
	~

Figure 66. Set up the Key Data

Under the **Wrapping Key** section, click the corresponding **Browse** buttons to select the **UFPK** and **W-UFPK** key pair that you generated for RA6M4 created in section 4.2 and 4.4. For the **IV**, select **Generate random value**. In the **Output** option, select **RFP**; then click the **Browse** button, choose the output folder, and name the output file.

C OII K	UFPK File :	C:\Secure_Key_Inject	tion\ra6m4_protected_mode_key_inf	fo\ufpk.key E	Browse
	W-UFPK File :	ecure_Key_Injection	.ra6m4_protected_mode_key_info\uf	fpk.key_enc.key	Browse
⊖ KUK	KUK File :	re_Key_Injection\ra6	m4_protected_mode_key_info\kuk_f	or_new_key.key [	Browse
IV					
Gener	ate random valı	Je			
⊖ Use sp	ecified value (1	6 hex bytes, big endia	n format) 00112233445566778899/	AABBCCDDEEFF	
Output					
Format :	RFP	✓ File:	or \ra6m4_protected_mode_key_in	fo\AES256.rkey	Browse
Address	10000	Key name :	NEW_AES256		
			Concerto Filo		
			Generate file		
utput File:	C:\Secure Key	Injection\ra6m4 prot	ected mode key info <mark>r AFS256.rkey</mark>		^
utput File: FPK: 00010	C:\Secure_Key_ 2030405060708	Injection\ra6m4_prot )90A0B0C0D0E0F0001	ected_mode_key_info <mark>.AES256.rkey</mark> 02030405060708090A0B0C0D0E0F		^
utput File: FPK: 00010 UFPK: 00	C:\Secure_Key_ 12030405060708 0000006FEE1503 384BB2CEAAE1	Injection\ra6m4_prot 90A080C0D0E0F0001 16A3B4E726F0B3F9E1F 16D628C0D459	ected_mode_key_info <mark>rAES256.rkey</mark> 02030405060708090A0B0C0D0E0F 74B7076FEE15036A3B4E726F0B3F9E1	F74B707	^
utput File: FPK: 00010 -UFPK: 00 : 771880D ncrypted k	C:\Secure_Key_ 120304050607081 0000006FEE1503 384BB2CEAAE11 ey:	Injection\ra6m4_prot 390A0B0C0D0E0F0001 6A3B4E726F0B3F9E1F E0D628C0D459	ected_mode_key_info <mark>.AES256.rkey</mark> 02030405060708090A0B0C0D0E0F 74B7076FEE15036A3B4E726F0B3F9E1	F74B707	^
utput File: FPK: 00010 -UFPK: 00 : 771880D hcrypted k 4B2476A2	C:\Secure_Key_ 120304050607084 0000006FEE1503 384BB2CEAAE1{ ey: 867B4F167553FI	Injection\ra6m4_prot )90A0B0C0D0E0F0001 16A3B4E726F0B3F9E1F E0D628C0D459	ected_mode_key_info <mark>.AES256.rkey</mark> 02030405060708090A0B0C0D0E0F 74B7076FEE15036A3B4E726F0B3F9E1 DBD7ADF685EB9D7E9F7B03A01A541	F74B707 2A62A4B8920127E0	5B3E710

Figure 67. Generate the AES 256 RFP Injection Key File

Now click Generate File. The  $\tt AES256.rkey$  file will be generated.

The plaintext AES-256 key and UFPK are NOT contained in the \*.rkey file, enabling confidential transfer of the key injection file contents.

## 5.1.1.2 Wrap an Initial ECC Public Key with the UFPK

A set of NIST test vectors are used in this application project. The CAVP NIST test vectors can be downloaded from the following link. The ECDSA vectors are what we will use.

Cryptographic Algorithm Validation Program | CSRC (nist.gov)



Use of these test vectors does	not replace validation of	ptained through the CAVP.	
The test vectors linked below ca 186-2 and FIPS 186-4) using the	an be used to informally ve validation systems <u>listed a</u>	erify the correctness of digital signature alg above.	orithm implementations (in FIPS
Response files (.rsp): the test vertex exactly.	vectors are properly format	tted in response (.rsp) files. Vendor respon	se files must match this format
Intermediate results files (.txt	:): files with intermediate r	esults (.txt) are supplied to help with debu	gging.
See the README file in each zip	file for details.		
	Publication	Algorithm Test Vectors	
	FIDS 195 4	DSA ECDSA BSA	
	FIPS 186-4		

Figure 68. ECDSA Test Vectors

After downloading the zip file 186-4ecdsatestvectors.zip, unzip it. The following vectors can be found in the plaintext file SigGen.txt.

Msg =
5905238877c77421f73e43ee3da6f2d9e2ccad5fc942dcec0cbd25482935faaf416983fe165b1a045ee2bcd2
e6dca3bdf46c4310a7461f9a37960ca672d3feb5473e253605fb1ddfd28065b53cb5858a8ad28175bf9bd386
a5e471ea7a65c17cc934a9d791e91491eb3754d03799790fe2d308d16146d5c9b0d0debd97d79ce8
Qx = 1ccbe91c075fc7f4f033bfa248db8fccd3565de94bbfb12f3c59ff46c271bf83
Qy = ce4014c68811f9a21a1fdb2c0e6113e06db7ca93b7404e78dc7ccd5ca89a4ca9
R = f3ac8061b514795b8843e3d6629527ed2afd6b1f6a555a7acabb5e6f79c8c2ac
S = 8bf77819ca05a6b2786c76262bf7371cef97b218e96f175a3ccdda2acc058903

Figure 69. NIST ECC secp256r1 Test Vector

Launch the SKMT GUI and select **RA Family, SCE9 Security Functions, and Protected Mode** on the **Overview** tab. On the **Wrap Key** tab, select the **Key Type** as **ECC** and **secp256r1, public**, as shown in Figure 70.

R Security Key Management Tool				_	×
File View Help					
Overview Generate UFPK Generate KUK	Wrap Key TS	SIP UPDATE FSBL DOTF	SFP		
Keys must be wranned by the II	EDK for secur	e injection or by the KIIK for	secure u	ndate	
Keys must be wrapped by the o	FPK for secur	e injection of by the Kok for	secure u	puate.	
Key Type Key Data					
$\bigcirc$ DLM/AL DLM-SSD $\lor$	⊖ AES	256 bits		C4	
⊖кик	⊖ RSA	2048 bits, public		ES	
OEM Root public	● ECC	secp256r1, public	·		
	OHMAC	SHA256-HMAC			

Figure 70. Choose secp256r1 Public Key

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

- Qx = 1ccbe91c075fc7f4f033bfa248db8fccd3565de94bbfb12f3c59ff46c271bf83
- Qy = ce4014c68811f9a21a1fdb2c0e6113e06db7ca93b7404e78dc7ccd5ca89a4ca9



Ke	ys must be wrapped by the UFPK for secure installation or by the KUK for secure	update.
Key Type	ey Data	Prowse
<ul> <li>Raw Qx</li> </ul>	Ccbe91c075fc7f4f033bfa248db8fccd3565de94bbfb12f3c59ff46c271bf83	A V
Qy	ce4014c68811f9a21a1fdb2c0e6113e06db7ca93b7404e78dc7ccd5ca89a4ca9	$\hat{}$

Figure 71. Provide the ECC Public Key data

Next, under the **Wrapping Key** section, click the corresponding **Browse** buttons to select the **UFPK** and **W-UFPK** key pair created in section 4.2 and 4.4. For the **IV**, select **Generate random value**. For the **Output** option, select **RFP**; then click the **Browse** button, choose the output folder, and name the output file.

UFPK	UFPK File :	C:\Secure_Key_Inject	on <sup>\</sup> ra6m4_protected_mode_key_inf	o\ufpk.key Browse.
	W-UFPK File :	ecure_Key_Injectior	a6m4_protected_mode_key_info\uf	pk.key_enc.key Browse.
⊖ KUK	KUK File :	re_Key_Injection\ra6r	h4_protected_mode_key_info\kuk_fo	or_new_key.key Browse
IV				
Gener	ate random valu	Je		
🔿 Use sp	ecified value (1	6 hex bytes, big endia	oformat) 001122334455667788994	ABBCCDDEEFF
Output				
Format :	RFP	✓ File :	4_protected_mode_key_info\ECC_	Public_Key.rkey Browse.
Address :	10000	Key name :	NEW_AES256	
			Generate file	
			Generate file	
tput File:	C:\Secure_Key_	Injection\ra6m4_prot	cted_mode_key_info\ECC_Public_K	ey.rkey
PK: 00010 UEPK: 00	2030405060708 000006FFF1503	090A0B0C0D0E0F0001	2030405060708090A0B0C0D0E0F 4B7076FFF15036A3B4F726F0B3F9F1	F74B707
F9737DD	6FB406496D295	BA2C49073BA9		
	ey:		A 5071000070070507001560550055	
crypted k	EEBOECOE35C2	10500882066077767570	// EX / TRU // I // I R // IEV / /X IEPESE /6 /	ETU/XEUXEE <e 60<="" leu="" td="" uu6=""></e>

## Figure 72. Generate the ECC Public Key RFP Injection Key File using GUI

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



## 5.1.1.3 Wrap a Key-Update Key with the UFPK

The SKMT can be used to generate a sample KUK. To generate the KUK key file, navigate to the **Generate KUK** tab and use: 000102030405060708090a0b0c0d0e0f000102030405060708090a0b0c0d0e0f.

Click the **Browse** button to select the folder and file name for the generated key file, here specified as kuk\_for\_new\_key.key. Next, click **Generate KUK key file**, and the kuk\_for\_new\_key.key file will be generated in the selected folder.

File View Help				
Overview Generate UFPK Generate KUK Wrap Key TSIP UPDATE FSBL	DOTF	SFP		
Key-Update Keys (KUKs) are used to securely update application keys aft The KUKs themselves must be securely injected	ter produc ed.	tion progr	amming.	
Key-Update Key				
Generate random value				
Use specified value (32 hex bytes, big endian format)				
000102030405060708090a0b0c0d0e0f000102030405060708090a0b0c0d0e0f				
Output file (.key) :				
rity\injecting-upqating-secure-user-keys\ra6m4_protected_mode_key_info\	kuk_for_n	ew_key.key	Brows	se
Generate KUK key file				
KUK: 000102030405060708090A0B0C0D0E0F000102030405060708090A0B0C0D0E0	)F			^
Output File: C:\RA8_Security\injecting-updating-secure-user-keys\ra6m4_prote	cted_mod	le_key_info		

Figure 73. Generate the KUK File used to Encrypt the User Key for SCE9



Next, we will wrap the KUK so it can be injected into the MCU. Navigate to the **Wrap Key** page and choose **KUK** from the **Key Type** area.

R Security Key Management Tool		– 🗆 ×
File View Help		
Overview Generate UFPK Generate KUK W	/rap Key TSIP UPDATE FSBL DOTF	SFP
Kevs must be wrapped by the UFF	PK for secure injection or by the KUK for	secure update.
Key lype Key Data		
O DLM/AL DLM-SSD ∨	○ AES 256 bits ~	ARC4
	○ RSA 2048 bits, public ~	⊖ TDES
OEM Root public	OECC secp256r1, public ∽	
	OHMAC SHA256-HMAC	

Figure 74. Choose KUK to Wrap

Navigate to the Key Data page, select the File option, and browse to the kuk\_for\_new\_key.key key file generated in Figure 73.

in occurry ricy man		$\sim$
File View Help		
Overview Generat	te UFPK Generate KUK Wrap Key TSIP UPDATE FSBL DOTF SFP	
12		
Keys n	nust be wrapped by the UFPK for secure injection or by the KUK for secure update.	
Keys n Key Type Key Da	nust be wrapped by the UFPK for secure injection or by the KUK for secure update. ata Key_Injecticn\ra6m4_protected_mode_key_info\kuk_for_new_key.key Browse.	

Figure 75. Provide the KUK .key File



Next, under the **Wrapping Key** section, click the corresponding **Browse** buttons to select the **UFPK** and **W-UFPK** key pair created in section 4.2 and 4.4. For the **IV**, select **Generate random value**. For the **Output** option, select **RFP**; then click the **Browse** button, choose the output folder, and name the output file.

Now click the Generate File button. The KUK.rkey file will be generated.

	UFPK File :	C:\Secure_Key_Inject	ion\ra6m4_prote	cted_mode_key_i	nfo\ufpk.key	Browse
	W-UFPK File :	ecure_Key_Injection\	ra6m4_protected	_mode_key_info\	ufpk.key_enc.key	Browse
⊖ KUK	KUK File :					Browse
IV						
Gene	ate random val	ue	6 - 13 - 1994			
() Use s	pecified value (1	6 hex bytes, big endia	n format) 0011	223344556677889	9AABBCCDDEEFF	
Output		<b></b>				
Format :	RFP	✓ File:	:ction\ra6m4_p	rotected_mode_k	ey_in <sup>f</sup> o\KUK.rkey	Browse
Address	10000	Key name :				
			Generate file			
utput File	C:\Secure_Key	_Injection\ra6m4_prote	cted_mode_key_	info\KUK.rkey		
Output File IFPK: 0001 V-UFPK: 00	C:\Secure_Key )2030405060708 0000006FEE150	Injection\ra6m4_prote 090A0B0C0D0E0F00010 36A3B4E726F0B3F9E1F	ected_mode_key )20304050607080 74B7076FEE15036	info\KUK.rkey 90A0B0C0D0E0F A3B4E726F0B3F9	E1F74B707	
Output File IFPK: 0001 V-UFPK: 00 /: 9EDC350 ncrypted I	C:\Secure_Key )2030405060708 0000006FEE150 ;F3CC946C3940 ey:	_Injection\ra6m4_prote 090A080C0D0E0F0001( 36A3B4E726F0B3F9E1F 61D70B89FF8D0	ected_mode_key )20304050607080 74B7076FEE15036	info\KUK.rkey 90A0B0C0D0E0F A3B4E726F0B3F9	E1F74B707	
Dutput File JFPK: 0001 V-UFPK: 00 /: 9EDC350 ncrypted J 3FECF7931	C:\Secure_Key )2030405060708 )0000006FEE150 ;F3CC946C3940 ;ey; )D3F967F24D0F	Injection\ra6m4_prote 090A0B0C0D0E0F0001( 36A3B4E726F0B3F9E1F 61D70B89FF8D0 57FC0493B1D46EAA0D	ected_mode_key )20304050607080 74B7076FEE15036 4D1D0D86882100	info\KUK.rkey 90A0B0C0D0E0F jA3B4E726F0B3F9 29D506A98F41594	E1F74B707 \145A6F85C939B21	19736BBEF

#### Figure 76. Generate the Key-Update Key Injection File using GUI for SCE9

#### 5.1.1.4 Wrap a New AES-256 User Key with the KUK

In the section, we will use the kuk\_for\_new\_key.key generated in Figure 73 to wrap a new AES-256 key.

We will use a second NIST test vector to demonstrate secure key updates using the KUK.

#### Figure 77. NIST Test Vector as New AES-256 Key Test Data

Navigate to the SKMT Wrap Key tab. In the Key Type area, select AES-256 with 256 bits.

Overview	Generate UF	PK Generate	кик М	'rap Key	TSIP UPDATE FSBI	DOTF	SFP	
Key Type	Keys must	be wrapped by	y the UFF	PK for sec	ure injection or by t	he KUK for s	ecure update.	
ODLM	/AL	DLM-SSD	$\sim$	AES	256 bits	~	O ARC4	
Окик				⊖ RSA	2048 bits, public	· · · ·		
OEM	Root public			OECC	secp256r1, publi	ic V		

Figure 78. Choose AES 256bit New User Key



In the **Key Data** area, provide the key data from the NIST vector based on Figure 77. The key data is duplicated here to copy and paste into the GUI interface.

Key Type Key Data	
◯ File	Browse
Raw     c000000000000000000000000000000000	^
	~

Figure 79. Provide the New AES 256-bit Key Data

In the Wrapping Key area, select KUK as the wrapping key and click Browse to locate the kuk\_for\_new\_key.key file generated in Figure 73. For the IV, choose Generate random value. For the Output option, choose C Source and name the output file as new\_aes\_key.c. Name the Key name property as NEW\_AES256. This name will be used in the source files for key-specific definitions.

Finally, click Generate file. Both the new\_aes\_key.c and the new\_aes\_key.h files will be generated.

<b>○ UFPK</b>	UFPK File :	C:\Secure_Key_Injection\ra6m4_protected_mode_key_info\ufpk.key	Browse
	W-UFPK File :	Secure_Key_Injection\ra6m4_protected_mode_key_info\ufpk.key_er	nc Browse
© KUK	KUK File :	<pre></pre>	y Browse
IV			
Gener	ate random valu	ue	
⊖ Use sp	pecified value (1	6 hex bytes, big endian format) 00112233445566778899AABBCCDD	EEFF
Output			
Format :	C Source	<ul> <li>File : jm4_protected_mode_key_info\new_aes_key.</li> </ul>	C Browse
Endian :	Little	Output additional data	
Address :	10000	Key name : NEW_AES256	
		Generate file	
utput File:	C:\Secure_Key_	Injection\ra6m4_protected_mode_key_info\new_aes_key.h	
utput File:	C:\Secure_Key_	Injection\ra6m4_protected_mode_key_info\new_aes_key.c	
: 94D28DC	96BE7C8185E37	/155541C4D966	
crypted k	ey: `854D740D0532`	77800282670D688580862456034538822058024D2847058870853DC553	8C34B06E54E
480D	05RD140D05521	11055202010D000000002A10054150022010024D20A1020010015DC255	00340000341
	CHOCECCEU		

Figure 80. Generate KUK-Wrapped AES-256 Key



## 5.1.1.5 Wrap a New ECC Public Key with the KUK

In the section, we will use the kuk\_for\_new\_key.key generated in Figure 73 to wrap a new ECC Public key.

To demonstrate updating the ECC public key, another NIST ECC secp256r1 test vector is used in this application project. This test vector can be found in SigGen.txt, downloaded based on Figure 68.

Msg = c35e2f092553c55772926bdbe87c9796827d17024dbb9233a545366e2e5987dd344deb72df987144b8c6c43b c41b654b94cc856e16b96d7a821c8ec039b503e3d86728c494a967d83011a0e090b5d54cd47f4e366c0912bc 808fbb2ea96efac88fb3ebec9342738e225f7c7c2b011ce375b56621a20642b4d36e060db4524af1 Qx = e266ddfdc12668db30d4ca3e8f7749432c416044f2d2b8c10bf3d4012aeffa8a Qy = bfa86404a2e9ffe67d47c587ef7a97a7f456b863b4d02cfc6928973ab5b1cb39 R = 976d3a4e9d23326dc0baa9fa560b7c4e53f42864f508483a6473b6a11079b2db S = 1b766e9ceb71ba6c01dcd46e0af462cd4cfa652ae5017d4555b8eeefe36e1932

#### Figure 81. New Set of NIST ECC Test Vectors

Follow the procedure below to wrap the new ECC public key using the KUK file generated in Figure 73.

From the SKMT GUI, make sure **RA Family**, **SCE9 Security Functions**, and **Protected Mode are** selected from the **Overview** page. Next, navigate to the **Wrap Key** page. Select the **Key Type** as **secp256r1**, **public** as shown in Figure 70.

Under the **Key Data** area, select **Raw** and provide **Qx** and **Qy** as shown below. The key data is duplicated here so the user can copy and paste it to the GUI interface.

Qx = e266ddfdc12668db30d4ca3e8f7749432c416044f2d2b8c10bf3d4012aeffa8a

Qy = bfa86404a2e9ffe67d47c587ef7a97a7f456b863b4d02cfc6928973ab5b1cb39



Figure 82. Provide the New ECC Public Key Data



Next, under the **Wrapping Key** section, click the corresponding **Browse** button to select the KUK generated in section 5.1.1.2. For the **IV**, select **Generate random value**. In the **Output** option, choose **C Source** and name the output as new ecc public key.c. set the **Key name** to NEW\_ECC\_PUB.

Finally, click Generate file. Both the new\_ecc\_public\_key.c and the new\_ecc\_public\_key.h files will be generated.

Wrappi	ng Key				
⊖ UFPK	UFPK File :	C:\Secure_Key_Injection\ra6m4_protected_mode_key_info\ufpk.key Brows	se		
	W-UFPK File :	ecure_Key_Injection\ra6m4_protected_mode_key_info\ufpk.key_enc.key Brows	se		
<b>● KUK</b>	KUK File :	re_Key_Injection\ra6m4_protected_mode_key_info\kuk_for_new_key.key	Browse		
IV					
Gener	ate random valı	Je			
() Use sp	ecified value (1	6 hex bytes, big endian format) 00112233445566778899AABBCCDDEEFF			
Output					
Format :	C Source	File : _protected_mode_key_info\new_ecc_public_key.d Brows	se		
Address	10000	Key name : NEW_ECC_PUB			
		Generate file			
		Generate nie			
			^		
utput File:	C:\Secure_Key_	Injection\ra6m4_protected_mode_key_info\new_ecc_public_key.h			
utput File: utput File: UK: 000102	C:\Secure_Key_ C:\Secure_Key_ 03040506070800	Injection\ra6m4_protected_mode_key_info\new_ecc_public_key.h Injection\ra6m4_protected_mode_key_info\new_ecc_public_key.c \a0abacc_nDbcbecond1a203406566708090A080c_0D0505			
utput File: utput File: UK: 000102 (: 3CDB949	C:\Secure_Key_ C:\Secure_Key_ 03040506070809 DA8CAB62937F	Injection\ra6m4_protected_mode_key_info\new_ecc_public_key.h Injection\ra6m4_protected_mode_key_info\new_ecc_public_key.c 30A080C0D0E0F000102030405060708090A080C0D0E0F FBFBF643D7F20			
utput File: utput File: UK: 000102 /: 3CDB949 ncrypted k CE08D3D2	C:\Secure_Key_ C:\Secure_Key_ 03040506070809 DA8CAB62937F ey: -3E270B9B49397	Injection\ra6m4_protected_mode_key_info\new_ecc_public_key.h Injection\ra6m4_protected_mode_key_info\new_ecc_public_key.c 90A0B0C0D0E0F000102030405060708090A0B0C0D0E0F FBFBF643D7F20 -60889865C3F93C2016D6CAF48A1AFF85FF884D6F1DDF3FC5A8A4526859878CDF27	02B		
utput File: utput File: UK: 000102 (: 3CDB949 ncrypted k CF08D3D2 986E5B03B	C:\Secure_Key_ C:\Secure_Key_ 03040506070809 DA8CAB62937F ey: F3F270B9B49390 C2498CE60FCC	Injection\ra6m4_protected_mode_key_info\new_ecc_public_key.h Injection\ra6m4_protected_mode_key_info\new_ecc_public_key.c 90A08DC0D0E0F000102030405060708090A08DC0D0E0F FBFBF643D7F20 260889865C3E93C2016D6CAE48A1AEF85FE884D6F1DDE3FC5A8A4526859878CDF27 504186532A79C3F23956BCB057BF79DB003562101A33E8	'02B		

Figure 83. Generate KUK-Wrapped ECC Public Key

## 5.1.2 Using the SKMT CLI Interface

This section describes how to perform the actions described above using the SKMT CLI interface. These examples use SCE9 Protected mode, but SCE7 support is fundamentally the same.

The /genkey command of the Security Key Management Tool command line tool skmt.exe will be used to prepare keys for secure injection and update. These are the options for this command:

- /keytype This input can take either ASCII or a one-byte hexadecimal input parameter indicating the key type.
- /ufpk The User Factory Programming Key.
- /wufpk The Renesas HRK-wrapped UFPK.
- /kuk The Key-Update Key for secure key update.
- /mcu The target MCU and security engine.
- /output The output of the command.

Refer to the Security Key Management Tool user's manual for more information about these commands, including the valid values for each parameter.

This application project uses an AES-256 key and an ECC secp256r1 public key to illustrate the secure key injection and update processes.

For these examples, we will use the UFPK and W-UPFK created earlier.



## 5.1.2.1 Wrap an Initial AES-256 Key with the UFPK

In the Command Prompt window opened earlier (section 4.3), use the following command to create the AES-256 key injection file (AES256\_CLI.rkey). Refer to the Security Key Management Tool user manual for more information on how to construct the command.

Note that in this example:

- We have specified the key type "AES-256".
- "RA-SCE9" is used for the /mcu option.
- We are using a randomly generated IV. The IV changes each time this command is executed.
- In this example, we have specified the complete file path for the key file AES256 CLI.rkey.

```
Output File: C:\Secure_Key_Injection\ra6m4_protected_mode_key_info\AES256_CLI.rkey
UFPK: 000102030405060708090A0B0C0D0E0F000102030405060708090A0B0C0D0E0F
W-UFPK: 000000006FEE15036A3B4E726F0B3F9E1F74B7076FEE15036A3B4E726F0B3F9E1F74B707
IV: 0B730F4F7194A9CB67E284A1B0D2A370
Encrypted key:
1D6612F7F276BFBBEBEB05410151C43E74E0368D3FB0688FB7A5D2D35E2B286A9963C14F3FE16A4529AAC7E8B0650EB72
```

#### Figure 84. Create the AES-256 User Key Injection File

The generated key file AES256\_CLI.rkey now contains the encrypted user key along with the W-UFPK. The plaintext AES-256 key and UFPK are NOT contained in the \*.rkey file, enabling confidential transfer of the key injection file contents.

## 5.1.2.2 Wrap an Initial ECC Public Key with the UFPK

In this section, we will use the ECC key pair in Figure 69 as an example of preparing an ECC public key for secure key injection.

In the Command Prompt window opened earlier (section 4.3), use the following command to create the ECC public key injection file (ECC\_Public\_Key\_CLI.rkey). Refer to the Security Key Management Tool user manual for more information on how to construct the command.

```
Skmt.exe /genkey /ufpk
file="C:\Secure_Key_Injection\ra6m4_protected_mode_key_info\ufpk.key" /wufpk
file="C:\Secure_Key_Injection\ra6m4_protected_mode_key_info\ufpk.key_enc.key"
/mcu "RA-SCE9" /keytype "secp256r1-public" /key
"1ccbe91c075fc7f4f033bfa248db8fccd3565de94bbfb12f3c59ff46c271bf83
ce4014c68811f9a21a1fdb2c0e6113e06db7ca93b7404e78dc7ccd5ca89a4ca9" /filetype
"rfp" /output
"C:\Secure_Key_Injection\ra6m4_protected_mode_key_info\ECC_Public_Key_CLI.rkey
"
```

#### Note that in this example:

- 1ccbe91c075fc7f4f033bfa248db8fccd3565de94bbfb12f3c59ff46c271bf83
   ce4014c68811f9a21a1fdb2c0e6113e06db7ca93b7404e78dc7ccd5ca89a4ca9 is the NIST ECC
   public key from Figure 69.
- We have specified the key type "secp256r1-public".
- "RA-SCE9" is used for the /mcu option.



- We are using a randomly generated IV. The IV is updated in each encryption instance.
- $\bullet$  The command option /output defines the locations and name of the output file.

```
Output File: C:\Secure_Key_Injection\ra6m4_protected_mode_key_info\ECC_Public_Key_CLI.rkey
UFFK: 000102030405060708090A0B0C0D0E0F000102030405060708090A0B0C0D0E0F
W-UFFK: 1234567890ABCDEF1234567890ABCDEF1234567890ABCDEF1234567890ABCDEF12345678
IV: 0273B7277508F33491F2BA569B092535
Encrypted key:
1234567890ABCDEF1234567890ABCDEF1234567890ABCDEF1234567890ABCDEF1234567890ABCDEF1234567890ABCDEF1234567890ABCDEF1234567890ABCDEF1234567890ABCDEF1234567890ABCDEF1234567890ABCDEF1234567890ABCDEF1234567890ABCDEF1234567890ABCDEF1234567890ABCDEF1234567890ABCDEF1234567890ABCDEF1234567890ABCDEF1234567890ABCDEF1234567890ABCDEF1234567890ABCDEF1234567890ABCDEF1234567890ABCDEF1234567890ABCDEF1234567890ABCDEF1234567890ABCDEF1234567890ABCDEF1234567890ABCDEF1234567890ABCDEF1234567890ABCDEF1234567890ABCDEF1234567890ABCDEF1234567890ABCDEF1234567890ABCDEF1234567890ABCDEF1234567890ABCDEF1234567890ABCDEF1234567890ABCDEF1234567890ABCDEF1234567890ABCDEF1234567890ABCDEF1234567890ABCDEF1234567890ABCDEF1234567890ABCDEF1234567890ABCDEF1234567890ABCDEF1234567890ABCDEF1234567890ABCDEF1234567890ABCDEF1234567890ABCDEF1234567890ABCDEF1234567890ABCDEF1234567890ABCDEF1234567890ABCDEF1234567890ABCDEF1234567890ABCDEF1234567890ABCDEF1234567890ABCDEF1234567890ABCDEF1234567890ABCDEF1234567890ABCDEF1234567890ABCDEF1234567890ABCDEF1234567890ABCDEF1234567890ABCDEF1234567890ABCDEF1234567890ABCDEF1234567890ABCDEF1234567890ABCDEF1234567890ABCDEF1234567890ABCDEF1234567890ABCDEF1234567890ABCDEF1234567890ABCDEF1234567890ABCDEF1234567890ABCDEF1234567890ABCDEF1234567890ABCDEF1234567890ABCDEF1234567890ABCDEF1234567890ABCDEF1234567890ABCDEF1234567890ABCDEF1234567890ABCDEF1234567890ABCDEF1234567890ABCDEF1234567890ABCDEF1234567890ABCDEF1234567890ABCDEF1234567890ABCDEF1234567890ABCDEF1234567890ABCDEF1234567890ABCDEF1234567890ABCDEF1234567890ABCDEF1234567890ABCDEF1234567890ABCDEF1234567890ABCDEF1234567890ABCDEF1234567890ABCDEF1234567890ABCDEF1234567890ABCDEF1234567890ABCDEF1234567890ABCDEF1234567890ABCDEF1234567890ABCDEF1234567890ABCDEF1234567890ABCDEF1234567890ABCDEF1234567
```

#### Figure 85. Create the ECC Public Key Injection File Using CLI

## 5.1.2.3 Create and Wrap a Key-Update Key with the UFPK

We can use the SKMT to create a key file for a KUK. This is done with the following command:

```
skmt.exe /genkuk /kuk
"000102030405060708090A0B0C0D0E0F000102030405060708090a0b0c0d0e0f" /output
"C:\Secure_Key_Injection\ra6m4_protected_mode_key_info\kuk_for_new_key_cli.key
"
```

#### Note that in this example:

- We have specified the complete file path for the key file.
- We need to use the same Key-Update Key as used in section 5.1.2.3.

```
KUK: 000102030405060708090A0B0C0D0E0F000102030405060708090A0B0C0D0E0F
Output File:
C:\Secure_Key_Injection\ra6m4_protected_mode_key_info\kuk_for_new_key_cli.key
```

#### Figure 86. Create the KUK Key File

The generated key file kuk\_for\_new\_key\_cli.key now contains the KUK. Retain this key file to use for wrapping new user keys for secure key updates.

To enable secure key updates, we must first securely inject the KUK. Use the SKMT to wrap the KUK with the UFPK and create a key injection file for use with RFP with the following command:

```
skmt.exe /genkey /ufpk
file="C:\Secure_Key_Injection\ra6m4_protected_mode_key_info\ufpk.key" /wufpk
file="C:\Secure_Key_Injection\ra6m4_protected_mode_key_info\ufpk.key_enc.key"
/mcu "RA-SCE9" /keytype "key-update-key" /key
file="C:\Secure_Key_Injection\ra6m4_protected_mode_key_info\kuk_for_new_key_cl
i.key" /filetype "rfp" /output
"C:\Secure Key Injection\ra6m4 protected mode key info\KUK CLI.rkey"
```

#### Note that in this example:

- We are using the KUK key file created above.
- We have specified the key type "key-update-key".
- We are using a randomly generated IV. The IV changes each time this command is executed.
- In this example, we have specified a complete file path for the key file (KUK\_CLI.rkey).

#### Figure 87. Create the Key-Update Key Injection File Using CLI



The generated key file  $KUK\_CLI.rkey$  now contains the wrapped KUK along with the W-UFPK. The plaintext KUK and UFPK are NOT contained in the \*.rkey file, enabling confidential transfer of the key injection file contents.

## 5.1.2.4 Wrap a New AES-256 Key with the KUK

The user can use the following command to wrap the new AES key defined in Figure 77 using the KUK. This is done with the following command.

Note that in this example:

- We are using a randomly generated IV. The IV changes each time this command is executed.
- We use the /keyname to create an identifiable key structure name that is unique in the software project. This resolves confusions when more than one set of new user keys are to be generated. If this option is not provided, a key structure name of encrypted user key data is generated for the key structure.
- The generated new\_aes\_key\_cli.c and new\_aes\_key\_cli.h files include the output information in
   a data structure. The user can directly include these two files in the application project. This is
   demonstrated in the example project included.

```
Output File: C:\Secure_Key_Injection\ra6m4_protected_mode_key_info\new_aes_key_cli.h
Output File: C:\Secure_Key_Injection\ra6m4_protected_mode_key_info\new_aes_key_cli.c
KUK: 000102030405060708090A0B0C0D0E0F000102030405060708090A0B0C0D0E0F
IV: 3C8841F6E6AE05B7625098EC70C542C1
Encrypted key:
03FE218ABCD0AD2F5A5634833ABD7F4D6F4CF8BF2CAC737CE1BE56C28DF0ADAD52536EED8DF405031230F935B087ECA0
```

## Figure 88. Encrypt the New User Key with the KUK

## 5.1.2.5 Wrap a New ECC Public Key With the KUK

Use the following command to wrap the new ECC public key shown in Figure 81.

```
skmt.exe /genkey /kuk
file="C:\Secure_Key_Injection\ra6m4_protected_mode_key_info\kuk_for_new_key.ke
y" /mcu "RA-SCE9" /keytype "secp256r1-public" /key
"e266ddfdc12668db30d4ca3e8f7749432c416044f2d2b8c10bf3d4012aeffa8abfa86404a2e9f
fe67d47c587ef7a97a7f456b863b4d02cfc6928973ab5b1cb39" /filetype "csource"
/keyname "NEW_ECC_PUB" /output
"C:\Secure_Key_Injection\ra6m4_protected_mode_key_info\new_ecc_public_key_cli.
c"
```

#### Note that in this example:

- e266ddfdc12668db30d4ca3e8f7749432c416044f2d2b8c10bf3d4012aeffa8a
   bfa86404a2e9ffe67d47c587ef7a97a7f456b863b4d02cfc6928973ab5b1cb39 is the ECC public key from the NIST test vector shown in Figure 81.
- The key type "secp256r1-public" is one of the available options specified in the Security Key Management Tool user's manual.
- "RA-SCE9" is used for the /mcu option.
- We are using a randomly generated IV. The IV changes each time this command is executed.
- The command option /output defines the locations and name of the output file.



- We use the /keyname to create an identifiable key structure name that is unique in the software project. This resolves confusions when more than one set of new user keys are to be generated. If this option is not provided, a key structure named encrypted user key data is generated for the key structure.
- The generated new\_ecc\_public\_key\_cli.c and new\_ecc\_public\_key\_cli.h files include the
  output information in a data structure. OThis is demonstrated in the example project included.

```
Output File: C:\Secure_Key_Injection\ra6m4_protected_mode_key_info\new_ecc_public_key_cli.h
Output File: C:\Secure_Key_Injection\ra6m4_protected_mode_key_info\new_ecc_public_key_cli.c
KUK: 000102030405060708090A0B0C0D0E0F000102030405060708090A0B0C0D0E0F
IV: 36E763D5A82924B4888732D50C93B602
Encrypted key:
9B0A7F8C91c038704A4F2C758EAC3DDD1372B4DC6AA4F22667D7D0E41218A1DEDBB8337E557B59B91100225BC8BBE2807221
4FF3C729D953AEFA9E997C3989967C831Dc6501E9528715ADA30FA0D0402
```

## Figure 89. Encrypt the New ECC Public Key with the KUK

## 5.2 Secure Key Injection via MCU Boot Interface

Follow this section to inject the AES-256 key, the ECC public key, and the Key-Update Key (KUK) that were prepared in section 5.1.1 or section 5.1.2. This capability is supported by RA Family MCUs that incorporate the SCE9 (Protected Mode) or SCE5\_B security engine.

### 5.2.1 Setting up the Hardware

Set up the EK-RA6M4 evaluation board as follows.

- Connect the jumper setting to J16 to put the device in boot mode. Refer to the EK-RA6M4 User's Manual for details.
- Connect the EK-RA6M4 J10 connector to the development PC using a USB micro-B cable to provide power and a debug connection using the onboard debugger.

Erase the entire MCU flash and ensure that the MCU is in the SSD Device Lifecycle State. This can be done using the Renesas Flash Programmer, as shown here.

- 1. Unzip rfp resources ra6m4.zip
- 2. Launch the Renesas Flash Programmer GUI executable.
- 3. Select File > Open Project and select ra6m4\_secure\_key\_inject.rpj.
- 4. Select Target Device -> Initialize Device.



#### Figure 90. Open RFP Project and Initialize the Device

Upon successful initialization, the following message will be printed.



Target device : R7FA6M4AF3CFB	
Connecting the tool Tool: J-Link (J-Link OB-S124 compiled Feb 2 2021 16:57:21), Interf. Connecting to the target device Setting the target device Communication speed: 9600bps Setting the target device	ace : 2 wire UART
Erasing the target device Disconnecting the tool <b>Operation completed</b>	
	~

Figure 91. RA6M4 Initialization

Unless there are permanently locked flash blocks, the entire flash will be erased, and the RA6M4 will be set to SSD state through the above steps.

## 5.2.2 Inject the Initial User Key and Key-Update Key

After initializing the RA6M4, power-cycle the board and follow the steps below to inject the AES-256 key, the ECC public key, and the Key-Update Key. This section uses the set of injection keys generated from the GUI interface.

To simplify duplicating this example, the .rkey files that match the example project are included in the rfp\_resources\_ra6m4.zip file. If the user intends to use the NIST vectors included in this application project for verification purposes, they can use the included .rkey files for system verification. The screen captures included in this section use these files for demonstration purposes. If different keys are used, then the corresponding .rkey files must be updated to match those keys.

Under the **Operation** tab, click **Add/Remove Files**. Next, click **Add Files**, and then add the .rkey file containing the AES256 key, which for this example is

 $rfp_resources_ra6m4\user_keys\AES256.rkey$  (Figure 67). Set the Address property to a data flash or code flash address applicable to your specific application. In this example, the AES key will be injected into the first block of Data Flash at 0x08000000.

File Details	— — ×
	Add File(s) Remove Selected File(s)
File Name	Type Address/Offset
File Offset File: C:\Secure_Key_hjection\ra6m4_protected_n Address: 08000000 h	mode_key_info\AES256.rkey OK Cancel OK Cancel

Figure 92. Add the AES256.rkey to RFP Configuration

Click **OK**, the AES256.rkey file will be configured to the corresponding load address.



🎽 File Details		12	- 🗆 X
	Add File(s)	Remo	ve Selected File(s)
File Name		Туре	Address/Offset
C:\Secure_Key_Injection\ra6m4_protected_mode_key_info\AB	ES256.rkey	KEY	0x0800000

Figure 93. AES256.rkey is added to the RFP Configuration

Click Add Files again and add ECC\_Public\_Key.rkey. Browse to the ECC\_Public\_Key.rkey (Figure 72). Set the Address property to a data flash or code flash address applicable to your specific application. In this example, the ECC public key will be injected into the third block of Data Flash at 0x08000080.

		- ~
Add File(s)	Remo	ove Selected File(s)
File Name	Туре	Address/Offset
C:\Secure_Key_Injection\ra6m4_protected_mode_key_info\AES256.rkey	KEY	0x0800000
C:\Secure_Key_Injection\ra6m4_protected_mode_key_info ECC_Public_Key.rkey	KEY	0x08000080
C:\Secure_Key_Injection\ra6m4_protected_mode_key_info	KEY	0x08000080

Figure 94. Configure the ECC Public Key Selection and Injection Address

Click **Add Files** again and add KUK.rkey. Browse to the KUK.rkey (Figure 76). Set the **Address** property to a data flash or code flash address applicable to your specific application. In this example, the Key-Update Key will be injected into the code flash at 0x00040000.

Add File(s)		
//ad 110(0)	lemove Sel	ected File(s)
File Name Typ	e Addr	ress/Offset
C:\Secure_Key_Injection\ra6m4_protected_mode_key_info\AES256.rkey KEY	0x08	000000
C:\Secure_Key_Injection\ra6m4_protected_mode_key_info\ECC_Public_Key.rkey KEY	0x080	080000
C:\Secure_Key_Injection\va6m4_protected_mode_key_infc\KUK.rkey KEY	0×00	040000

Figure 95. Configure the Key-Update Key Selection and Injection Address

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



Operation	Operation Settings	Block Settings	Flash Options	Connect Settings	Unique Code	
Comm	and			Erase Options		
	Erase			Erase Selected B	ocks	~
	Program			Program & Verify Op	tions	
	Verify			Erase Before F	Program	
	Program Flash Option			Verify by reading t	he device	
	Verify Flash Options					
	Chackeum					

Figure 96. Select to Perform Flash Erase, Program, and Verify

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

<b></b>	R7FA6M4AF3CFB				$\checkmark$
Ħ	Code Flash 1	0×00000000	0x000FFFFF	1.0 M	$\checkmark$
ŧ	····· Data Flash 1	0x08000000	0x08001FFF	8 K	$\checkmark$
+	····· Config Area 1	0x0100A100	0x0100A2FF	512	$\checkmark$

Figure 97. Entire Flash Region is Selected for Erase

• Browse to the **Operation** tab. Click **Start** to inject the AES-256, the ECC public key, and the Key-Update Key. The injection should succeed with a similar output message as shown below at the selected flash addresses.

F	ile Target Device Help	
Op	Operation Settings Block Settings Rash Options Connect Settings	Unique Code
	Project Information Current Project: ra&m4_secure_key_inject.rpj Microcontroller: R7FA6M4AF3CFB	
	Program and User Key Files	
	AES256.rkey, ECC_Public_Key.rkey, KUK.rkey	
	3 files selected	Add/Remove Files
	Command	
	Erase >> Program >> Verify	
	Start	ок
1] Wiri D	Data Flash 1] 0x08000000 - 0x08001FFF size :8 K ting data to the target device Jser Keys] 0x00040000	^
	Jser Keys] 0x08000000 Jser Keys] 0x0800080 Jser Keys] 0x00040000 Jser Keys] 0x08000000 Jser Keys] 0x08000000	
Dis Op	connecting the tool eration completed.	v

Figure 98. Secure Keys Successfully Injected

In this example code, no application is programmed since we are interested only in the key injection. In a production flow, it is possible to program the application and user keys together. This operation can also be performed using the command line function of RFP.



## 6. Secure Key Injection Preparation for RSIP and SCE7 Compatibility Mode

This section shows how to generate the .c and .h files, which can be used in an application project that uses the FSP APIs to inject keys into the PSA Crypto APIs using the security engine in Compatibility Mode. This key injection method must be used for both user keys and Key-Update Keys.

## 6.1 Wrap an AES-128 User Key Using the UFPK for RSIP-E51A Compatibility Mode

A NIST CAVP test vector is used for the demonstration.

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

#### Figure 99. NIST AES-128 Test Vector

Using the SKMT GUI interface, on the Overview tab, select RA Family, RSIP-E51A Compatibility Mode.

Select MCU/MPU and secu	rity engine : RA Family, RSIP-E51A Compatibility Mode	~
	Please select the target MCU or MPU before continuing.	

#### Figure 100. Choose RA Family, RSIP-E51A Compatibility Mode

On the Wrap Key tab, in the Key Type area, choose AES and 128 bits.

Overview	Generate Uf	PK Generate KUK	Wrap Key	TSIP UPDATE	FSBL	DOTF	SFP	
	Keys mus	t be wrapped by the	UFPK for sec	ure injection o	r by the	KUK for s	ecure up	date.
Кеу Туре	Key Data							
	AL	DLM-SSD	🗸 🖲 AES	128 bits		~		4
⊖кик			⊖ RSA	2048 bits, p	ublic	$\sim$		S
	Root public		OECC	secp256r1,	public	$\sim$		
				SHA256-HM	MAC	$\sim$		

Figure 101. Choose AES-128 bits as the Key Type

Select the **Key Data** tab and input the **Raw** Key Data as shown below based on the NIST vector as shown in Figure 99.



Keys must be	wrapped by the UFPK for secure injection or by the KU	K for secure update.
Key Type Key Data		
⊖ File		Browse
● Raw	e0000000000000000000000000000000000000	^

#### Figure 102. Set up the Initial AES-128 Key Data

Under the **Wrapping Key** section, click the corresponding **Browse** buttons to select the **UFPK** and **W-UFPK** key pair. Choose the **Generate random value** option for the **IV** data. For the **Output** option, select **C Source**; then click the **Browse** button, choose the output folder and file name, and name the key. This name will be reflected in the definitions generated for the C source files.

Now click the Generate File button. The source files to inject the AES key will be generated.



	Key Manageme	ent Tool							_		×
e View	Help										
verview	Generate UFPK	Generate	KUK Wr	ap Key TSIP	UPDATE	FSBL	DOTF	SFP			
Keys must be wrapped by the UFPK for secure injection or by the KUK for secure update.											
Key Type Key Data											
◯ File	O File e00000000000000000000000000000000000									e	
Raw		e000000	000000000	000000000000000000000000000000000000000	00000						^
						-					
◯ Rando	om - Output File									Brows	e
Wranni	na Kev										
UFPK	UFPK File :	ecure Kev	Injection	ra8m1 com	pitibility	mode ke	ey info∖ra	8x1 ufr	ok.key	Brows	
0	W-UFPK File :	:y_Injectio	- , n\ra8m1_c	ompatibility	node ke	- y_info\ra	a8x1_ufpl	k.key_er	nc.key	Brows	se
⊖кик	KUK File :					-			-	Brows	se
IV.											
Gener	ate random valu	e									
O Use specified value (16 hex bytes, big endian format)         00112233445566778899AABBCCDDEEFF									EEFF		
🔿 Use sp								Deebb			
OUse sp								beebb			
OUse sp Output Format :	C Source	~ F	ile :	atibility_m	ode_key_i	nfo\ra8n	n1_initial_	_aes128_	_key.c	Brows	;e
OUse sp Output Format : Address :	C Source	~ F	ile : (ey name :	atibility_m RA8M1_AE	ode_key_i S128	nfo\ra8n	n1_initial_	_aes128_	key.c	Brows	;e
O Use sp Output Format : Address :	C Source	~ F	ile : (ey name :	atibility_m RA8M1_AE Generate	ode_key_i S128 file	nfo\ra8n	n1_initial_	_aes128	_key.c	Brows	;e
O Use sp Output Format : Address :	C Source 10000	V F	ile : Key name :	atibility_m RA8M1_AE Generate	ode_key_i S128 file	nfo\ra8n	n1_initial_	_aes128_	_key.c	Brows	;e
O Use sp Output Format : Address :	C Source 10000	~ F	ïle : ćey name :	atibility_m RA8M1_AE Generate	ode_key_i S128 file	nfo\ra8n	n1_initial_	aes128	key.c	Brows	;e

### Figure 103. Generate the Initial AES-128 Encrypted Key File

Note that the generated <code>ra8m1\_initial\_aes\_128.c</code> and <code>ra8m1\_initial\_aes\_128.h</code> are used in the RA8M1 secure key injection example project.



## 6.2 Wrap an AES-128 User Key Using the UFPK for SCE7

The same NIST CAVP test vector is shown in Figure 99 is used for the demonstration.

Using the SKMT GUI interface, on the **Overview** tab, select **RA Family, SCE7**. On the **Wrap Key** tab, in the **Key Type** area, choose **AES** and **128 bits**.

enerate UFPK	Generate KUK	Wrap Key	TSIP UPDATE	FSBL	DOTF	SFP			
Keys must be wrapped by the UFPK for secure injection or by the KUK for secure update									
Key Type Key Data									
DLI	M-SSD	AES	128 bits		~		C4		
		⊖ RSA	2048 bits, j	public	~		S		
ot public		OECC	secp256r1,	, public	$\sim$				
			C SHA256-H	IMAC	$\sim$				
	Keys must be ey Data DL	Keys must be wrapped by the ey Data DLM-SSD	Keys must be wrapped by the UFPK for ser ey Data DLM-SSD RSA of public HMA	Keys must be wrapped by the UFPK for secure injection of ey Data DLM-SSD AES AES 128 bits CRSA 2048 bits, t public ECC SHA256-H	Keys must be wrapped by the UFPK for secure injection or by the ey Data DLM-SSD • AES 128 bits O RSA 2048 bits, public ot public • ECC secp256r1, public O HMAC SHA256-HMAC	Keys must be wrapped by the UFPK for secure injection or by the KUK for secure injection or by the KUK for secure Jack	Keys must be wrapped by the UFPK for secure injection or by the KUK for secure up         ey Data         DLM-SSD <ul> <li>AES</li> <li>128 bits</li> <li>AR</li> <li>RSA</li> <li>2048 bits, public</li> <li>TDB</li> <li>t public</li> <li>ECC</li> <li>secp256r1, public</li> <li>HMAC</li> <li>SHA256-HMAC</li> </ul>		

### Figure 104. Choose AES-128 bits as the Key Type

Select the **Key Data** tab and input the **Raw** Key Data as shown below based on the NIST vector as shown in Figure 103.

Overview	Generate UFPK	Generate KUK	Wrap Key	TSIP UPDATE	FSBL	DOTF	SFP			
	Keys must be wrapped by the UFPK for secure injection or by the KUK for secure update.									
Кеу Туре	Key Data									
⊖ File		e0000000000	0000000000	0000000000				Browse		
Raw		e0000000000	0000000000	000000000				^		
								~		
◯ Rando	om - Output File							Browse		

Figure 105. Set up the AES-128 Key Data

Under the **Wrapping Key** section, click the corresponding **Browse** buttons to select the **UFPK** and **W-UFPK** key pair for RA6M3. Choose the **Generate random value** option for the **IV** data. For the **Output** option, select **C Source**; then click the **Browse** button, choose the output folder and file name, and name the key. This name will be reflected in the definitions generated for the C source files.

Now click the Generate File button. The source files to inject the AES key will be generated.



Ovenview Generate LIE	DK Generate KUK Wrap Key, TSID LIDDATE ESRI DOTE SED	
Overview Generate OF	Senerale KOK WHEP KEY TSIP OPDATE FSBL DOTF SPP	
Keys mus	be wrapped by the UFPK for secure injection or by the KUK for secure	e update.
Key Type Key Data		
◯ File	e0000000000000000000000000000000000000	Browse
Raw	e0000000000000000000000000000000000000	^
Random - Output	ile	Browse
	C:\Secure Key Injection\ra6m3 key info\ufpk ra6m3.key	Browse
W-UEPK File	C:\Secure Key Injection\ra6m3 key info\ufpk ra6m3.key enc.key	Browse
		Browse
		Diotizein
Generate random v	alue	
0	(16 hex bytes, big endian format) 00112233445566778899AABBCCD	DEEFF
O Use specified value		
O Use specified value		
Ouse specified value Output Format : C Source	File : ure_Key_Injection\ra6m3_key_info\initial_ae	es_128.c Browse
Output Format : C Source Address : 10000	File :     ure_Key_Injection\ra6m3_key_info\initial_ae       Key name :     AES128	Browse
Ouse specified value Output Format: C Source Address: 10000	File : ure_Key_Injection\ra6m3_key_info\initial_ae Key name : AES128 Generate file	es_128.c Browse
O Use specified value Output Format: C Source Address: 10000	File :       ure_Key_Injection\ra6m3_key_info\initial_ae         Key name :       AES128         Generate file       Generate file	es_128.c Browse
O Use specified value Output Format: C Source Address: 10000	File : ure_Key_Injection\ra6m3_key_info\initial_ae Key name : AES128 Generate file	es_128.c Browse
O Use specified value Output Format: C Source Address: 10000	File : ure_Key_Injection\ra6m3_key_info\initial_ad	es_128.c Browse
O Use specified value Output Format : C Source Address : 10000 Output File: C:\Secure_Ke	File : ure_Key_Injection\ra6m3_key_info\initial_ae Key name : AES128 Generate file y_Injection\ra6m3_key_info\initial_aes_128.h y_Injection\ra6m3_key_info\initial_aes_128.c	es_128.c Browse

Figure 106. Generate the AES-128 Encrypted Key File

## 7. Example Project for RA6M4 (SCE9 Protected Mode)

To exercise the example projects as is, users can follow the steps below:

- Inject the included example RFP injection keys (AES256.rkey, KUK.rkey, and ECC\_Public\_Key.rkey which are included in rfp\_resource\_ra6m4.zip) by following section 5.2.2.
- A set of new user keys (AES256 as well as ECC Public Key) generated using the example KUK is
  already provisioned in the example projects. The user can then directly proceed to exercise the example
  project.
- Please do not use the example keys for production support.

To use the example projects with customized keys, the user can follow below steps:



- To test customized RFP injection keys and new user update keys (generated by the following section 5.1.1 or 5.1.2 rather than using the ones included in rfp\_resources\_ra6m4.zip), the user needs to follow section 5.2.2 to inject the keys to the MCU. User also needs to generate customized new user key files (new\_aes\_key.c/.h and new\_ecc\_public\_key.c/.h) with the same key name to replace the corresponding files used in the example project. Once the example projects are updated, the user can proceed to run the example projects to verify the operations.
- To test the new user key update procedure only, the user can use the included RFP KUK.rkey file to generate new source files to replace the corresponding files in the example project. Once the example projects are updated, the user can then proceed to the verification of the operations.

## 7.1 Example Project Overview

This pair of TrustZone-based secure and non-secure example projects provides the following functions:

#### Secure project (secure\_key\_inject\_update\_ra6m4\_s):

- Uses the injected AES-256 key to perform cryptographic operation using AES256-CBC.
- The injected Key-Update Key (KUK) is used to inject the new AES-256 key and store this new AES-256 key for data flash.
- Uses the new AES-256 to perform cryptographic operation using AES256-CBC.
- Uses the injected ECC public key to verify the NIST test signature shown in Figure 69.
- The injected Key-Update Key (KUK) is used to inject the newly wrapped ECC public key and store this new ECC public key for data flash.
- Uses the new ECC public key to verify the NIST test signature shown in Figure 81.

### Non-secure project (secure\_key\_inject\_update\_ra6m4\_ns):

- Establishes an RTT Viewer interface to allow users to select the intended Secure Crypto Engine and flash operation.
- Calls the non-secure callable APIs provided from the secure project based on user selection from the RTT Viewer interface.
- Prints the user operation results on the RTT Viewer.





Figure 107. Software Block Diagram

The FSP modules used in this pair of example projects are:

- r\_sce\_protected: This module is used in the secure region and provides services to the non-secure region via non-secure callable APIs
- r\_flash\_hp: This module is used in the secure region and provides services to the non-secure region via non-secure callable APIs

For more information on designing applications with TrustZone<sup>®</sup> support, refer to the application project *Renesas RA Family MCU Security Design with TrustZone – IP Protection.* 

## 7.2 Using the RFP Injected Keys

## 7.2.1 Formatting the Injected Keys

The keys that are injected into the MCU flash using RFP cannot be used directly by the FSP Crypto APIs. A minor formatting change is required.

## 7.2.1.1 Formatting the Injected AES Key

The following code snippet reads the AES-256 key from Flash. The destination buffer can then be used for cryptographic operations. Replace the macro DIRECT\_AES\_KEY\_ADDRESS with the actual injection address.

```
static sce_aes_wrapped_key_t injected_key;
injected_key.type = SCE_KEY_INDEX_TYPE_AES256;
memcpy(injected_key.value, (uint32_t *)DIRECT_AES_KEY_ADDRESS,
HW SCE AES256 KEY INDEX WORD SIZE*4);
```



## 7.2.1.2 Formatting the Injected ECC Public Key

The following code snippet reads the ECC public key from Flash. The destination buffer can then be used for cryptographic operations. Replace the macro <code>DIRECT\_ECC\_PUB\_KEY\_ADDRESS</code> with the actual injection address.

```
static sce_ecc_public_wrapped_key_t ecc_public_key_injected;
ecc_public_key_injected.type = SCE_KEY_INDEX_TYPE_ECC_P256_PUBLIC;
wrapped_ecc_public_key_size = sizeof(ecc_public_key_injected.value);
memcpy((uint8_t *)(&(ecc_public_key_injected.value)), (uint8_t *)DIRECT_ECC_PUB_KEY_ADDRESS,
wrapped ecc public key size);
```

## 7.2.1.3 Formatting the Injected KUK

The following code snippet reads the injected KUK from the flash. The destination buffer can then be used for secure key updates. Replace the macro KUK ADDRESS with the actual injection address.

```
static sce_key_update_key_t kuk_key;
kuk_key.type = SCE_KEY_INDEX_TYPE_UPDATE_KEY_RING;
memcpy(kuk key.value, (uint32 t *)(KUK ADDRESS),HW SCE UPDATE KEY RING INDEX WORD SIZE*4);
```

## 7.2.1.4 Formatting an Injected RSA Public Key

This application project does not include an example usage for RSA secure key injection and update, but the principles are identical. The following code snippet can be used to format an injected RSA public key. Replace the macro RSA 2048 PUB KEY ADDRESS with the actual injection address

```
static sce_rsa2048_public_wrapped_key_t injected_rsa_public_key;
injected_rsa_public_key.type = SCE_KEY_INDEX_TYPE_RSA2048_PUBLIC;
uint32_t wrapped_rsa_2048_public_key_size = sizeof(injected_rsa_public_key.value);
memcpy((uint8_t *)(&(injected_rsa_public_key.balur)), (uint32_t *)RSA_2048_PUB_KEY_ADDRESS,
wrapped rsa 2048 public key size);
```

## 7.2.2 Verifying the Injected Key and the Updated Key

To verify the AES injection, provide the plaintext message and the expected cipher text for the injected AES key and the updated AES key to the software project. For example, based on the NIST vectors presented in Figure 64 and Figure 77, use the plaintext data below in <code>aes\_crypto\_operations.c</code>:

```
#define BLOCK
                                                      16
/* NIST vector plaintext message used for both directly injected AES key and updated AES key*/
static uint8 t plain text[BLOCK] = {
         0x00, 0
};
/* NIST vector initialization vector for the directly injected AES key and the AES key update*/
static uint8 t iv[BLOCK] = {
            0x00, 0x00
};
/* NIST cipher to match directly injected AES key*/
static uint8_t cipher_expected[BLOCK] = {
         0xe3, 0x5a, 0x6d, 0xcb, 0x19, 0xb2, 0x01, 0xa0, 0x1e, 0xbc, 0xfa, 0x8a, 0xa2, 0x2b, 0x57, 0x59
};
/* NIST cipher to match new AES key */
static uint8 t cipher expected new[BLOCK] = {
     0xb2, 0x91, 0x69, 0xcd, 0xcf, 0x2d, 0x83, 0xe8, 0x38, 0x12, 0x5a, 0x12, 0xee, 0x6a, 0xa4, 0x00
};
```



To verify the ECC public key injection, the expected signature using the ECC private key, which matches the injected ECC public key (see Figure 69) is provided in the array ECC\_SECP256R1ExpectedSignature in ecc\_crypto\_operation.c.

#### Figure 108. Provision the ECC\_SECP256R1ExpectedSignaure Array

Similarly, the expected signature using the ECC private key which matches the updated ECC public key (see Figure 81) is provided in the array ECC\_SECP256R1ExpectedSignature\_New in ecc crypto operation.c.

#### Figure 109. Provision the ECC\_SECP256R1ExpectedSignaure\_New Array

There is no action needed from the user if the same sets of keys and plaintext messages are used. If new sets of keys and messages are used, the user needs to update the project with the new keys and messages.

## 7.3 FSP Crypto Module Support for User Key Update

This section introduces the FSP Crypto APIs for SCE Protected Mode that are used for secure user key updates. For a complete description of all FSP Crypto APIs, refer to the FSP User's Manual.

To use keys that have been injected via the secure key injection process using the MCUboot interfaces, the application must refer to those keys at the address where they were injected. If you inject keys at addresses other than those demonstrated above, be sure to change your application code to reflect those addresses. See instructions in section 7.4.

To perform a secure AES key update, use the following API to MCU-uniquely wrap a new AES key using a previously injected Key-Update Key:

```
fsp_err_t R_SCE_AES256_EncryptedKeyWrap (
    uint8_t *initial_vector,
    uint8_t *encrypted_key,
    sce_key_update_key_t *key_update_key,
    sce_aes_wrapped_key_t *wrapped_key )
```



#### The API parameters are:

- [in] initial\_vector: Pointer to a buffer that holds the initialization vector that was used to wrap the new key. This must be the IV that was used during the key wrap process shown in section 5.1.1.4 or section. This value will be included in the generated new\_aes\_key.c and new\_aes\_key.h.
- [in] encrypted\_key: Pointer to a buffer that holds the new key, wrapped by the KUK. In this example, it is the KUK-wrapped AES-256 key that was output during the key wrap process shown in section 5.1.1.4 or section 5.1.2.4. This value will be included in the generated new\_aes\_key.c and new aes key.h.
- [in] key\_update\_key: Pointer to the Key-Update Key that was previously injected on the MCU. This address must match the address used when injecting the KUK into section 5.2.2. The user needs to update the macro definition KUK\_ADDRESS defined in flash\_storage.h to match the injection address.
- [in, out] wrapped\_key: This is the SRAM buffer to store the wrapped new user key. For security considerations, it is recommended to erase this buffer right after the wrapped key is saved to flash. In this application project, the newly generated wrapped key is stored in data flash and used in the example project.

To perform a secure ECC public key update, use the following API to MCU-uniquely wrap a new ECC public key using a previously injected Key-Update Key:

```
fsp_err_t R_SCE_ECC_secp256r1_EncryptedPublicKeyWrap (
   uint8_t * initial_vector,
   uint8_t *encrypted_key,
   sce_key_update_key_t *key_update_key,
   sce_ecc_public_wrapped_key_t *wrapped_key )
```

#### The API parameters are:

- [in] initial\_vector: Pointer to a buffer that holds the initialization vector that was used to wrap the new key. This must be the IV that was used during the key wrap process shown in section 5.1.1.5 or section 5.1.2.5. This value will be included in the generated new\_ecc\_public\_key.c and new\_ecc\_public\_key.h.
- [in] encrypted\_key: Pointer to a buffer that holds the new key, wrapped by the HUK. In this example, it is the KUK-wrapped ECC private key that was output during the key wrap process shown in section 5.1.1.5 or section 5.1.2.5. This value will be included in the generated new\_ecc\_public\_key.c and new\_ecc\_public\_key.h.
- [in] key\_update\_key: Pointer to the Key-Update Key that was previously injected on the MCU. This address must match the address used when injecting the KUK into section 5.2.2. The user needs to update the macro definition KUK\_ADDRESS defined in flash\_storage.h to match the injection address.
- [in, out] wrapped\_key: This is the SRAM buffer to store the wrapped new user key. For security considerations, it is recommended to erase this buffer right after the wrapped key is saved to flash. In this application project, the newly generated wrapped key is stored in data flash and used in the example project.

## 7.3.1 Save the New Wrapped Key to Data Flash

Once a new key is wrapped, the user needs to use the flash driver  $r_flash_hp$  to manually store it in the data flash.

```
sce_aes_wrapped_key_t wrapped_new_user_key;
error = R_SCE_AES256_EncryptedKeyWrap (
    iv_encrypt_new_key, encrypted_new_key, &kuk_key, &wrapped_new_user_key );
```

Refer to function store new aes key to data flash() and function

store\_new\_ecc\_pub\_key\_to\_data\_flash() for the operations of storing the new wrapped keys to data flash.



## 7.4 Import and Compile the Example Project

Follow the steps below to exercise the example project. Note that there are sections of the code that must be updated using the secure key injection results generated above prior to compiling and running the project. Note that if the user has used the NIST vectors included in this application project for verification purposes, steps 4 to 5 can be skipped.

- 1. Launch e<sup>2</sup> studio and import secure\_key\_inject\_update\_ra6m4.zip file to a workspace.
- 2. At the bottom of flash\_storage.h, find the macro definitions DIRECT\_AES\_KEY\_ADDRESS, DIRECT\_ECC\_PUB\_KEY\_ADDRESS, and KUK\_ADDRESS based on Figure 98.
- 3. Replace new\_aes\_key.h and new\_aes\_key.c with the new sets of files generated in section 5.1.1.4 or section 5.1.2.4 located in folder \secure key inject update ra6m4 s\src\.
- 4. Replace new\_ecc\_public\_key.c and new\_ecc\_public\_key.h generated in section 5.1.1.5 or section 5.1.2.5 located in folder \secure\_key\_inject\_update\_ra6m4\_s\src\.
- 5. If different file names are used, update the #include definition in aes\_crypto\_operations.c on
  this line to reflect the new file name.



## Figure 110. Include the Generated Header File for AES Operation

6. If different file names are used, update the #include definition in ecc\_crypto\_operations.c on this line to reflect the new file name.



## Figure 111. Include the Generated Header File for ECC Operation

- 7. Next, double-click configuration.xml from the secure project. Once the configurator is opened, click **Generate Project Content** and then compile the secure project.
- 8. Expand the non-secure project and double-click the configuration.xml file. Once the configurator is opened, click Generate Project Content and compile the non-secure project.

## 7.5 Running the Example Project

Prior to running the example project, the user is requested to remove Jumper J16 to put the MCU in Normal execution mode.

Once the source code compilation is successful, follow the steps below to exercise the example projects:

- Choose to debug from the non-secure application. Right-click on secure\_key\_inject\_update\_ra6m4\_ns and select Debug As > Renesas GDB Hardware Debugging.
- 2. Execution will halt at the secure project reset handler.



Figure 112. Running to the Secure Project Reset Handler



- 3. Click **Resume** II twice to run the project.
- 4. Open the J-Link RTT Viewer with the settings shown below.

J-Link RTT Viewer V7.98b   Configuration X
Connection to J-Link
● USB
○ Existing Session
Specify Target Device
R7FA6M4AF
Force go on connect
Script file (optional)
Target Interface & Speed
SWD 🔻 4000 kHz 🔻
RTT Control Block
O Auto Detection O Address O Search Range
Enter one or more address range(s) the RTT Control block can be located in. Syntax: <rangestart [hex]=""> <rangesize>[, <range1start [hex]=""> <range1size>,] Example: 0x10000000 0x1000, 0x2000000 0x1000</range1size></range1start></rangesize></rangestart>
0x20000000 0x8000
OK Cancel

Figure 113. RTT Viewer Setting

5. Click **OK**. The following menu should be printed.

00> M	NENU to Select
00> Pr	ress 1 to exercise the cryptographic operation with directly injected AES key
00> Pr	ress 2 to create updated new AES key using FSP
00> Pr	ress 3 to exercise the cryptographic operation with newly updated AES key
00> Pr	ress 4 to exercise the cryptographic operation with directly injected ECC public key,
00> Pr	ress 5 to create updated new ECC public key using FSP
00> Pr	ress 6 to exercise the cryptographic operation with newly updated ECC public key

Figure 114. Main RTT User Menu

A. Input **1** to confirm that the cipher text for the first AES key is successfully decrypted by the injected AES-256 key.

Cryptographic operation with directly injected AES key, which is injected via the serial interface. 00> 00> 00> Result: Cryptographic operation is successful with directly injected AES key

Figure 115. Crypto Operation with Injected AES-256 Key

B. Input 2 to perform a key update to wrap the new AES-256 key and save the new key to data flash. Note that the SCE handles the wrapping of the new key internally without exposing the plaintext key. It is not possible to extract the plaintext key. The wrapped AES key in SRAM is deleted after storing it to the data flash. Note that if menu option '1' is rerun after menu item '2' is run, it will fail because the new AES key will not generate the same cipher text as the original key.

	2							0		•				, 1							
00>	Update	the	new	AES	key	enci	ryp	ted	with	key	update	key	and	store	the	new	wrapped	key	in	data	flash.
00>																					
00>	Result:	AES	Key	is u	pdat	ted a	and	st	ored	to D	ata Fla	sh									

Figure 116. Update the AES Key and Store to Data Flash



C. Input **3** to confirm that the cipher text for the second AES key is successfully decrypted by the updated AES-256 key.



#### Figure 117. Crypto Operation with the New AES Key

D. Input **4** to confirm that the signature generated using the first ECC private key is successfully verified by the injected ECC public key.

00> Cryptographic operation with directly injected ECC public key, which is injected via the serial interface. 00> 00> Result: Cryptographic operation is successful with directly injected ECC public key

#### Figure 118. Crypto Operation with Injected ECC Public Key

E. Input **5** to perform a key update to wrap the new ECC public key and save the new key to data flash. Note that the SCE handles the wrapping of the new key internally without exposing the plaintext key. It is not possible to extract the plaintext key. The wrapped ECC public key in SRAM is deleted after storing to data flash. Note that if menu option '4' is rerun after menu item '5' is run, it will fail because the new ECC public key cannot verify a signature that was generated by the first key's private key.

```
< 5
00> Update the new ECC public key encrypted with key update key and store the new wrapped key in data flash.
00>
00> Result: ECC public Key is updated and stored to Data Flash
```

### Figure 119. Update the ECC Public Key and Store to Data Flash

F. Input **6** to confirm that the signature generated using the second ECC private key is successfully verified by the updated ECC public key.



#### Figure 120. Crypto Operation with the New ECC Public Key

Successful operations of the above menu items conclude the demonstration of the secure key injection and update in this application project.

## 8. Example Project for RA8M1 (RSIP Compatibility Mode)

This section introduces RSIP Compatibility Mode with an example of AES-128 user key injection and update.

## 8.1 Overview

This example project demonstrates the following functionalities of the compatibility mode of RSIP-E51A.

- AES-128 key injection using the files generated in section 6.1.
- Verifying the injected AES-128 key using PSA Crypto APIs and a NIST AES test vector.



## 8.2 Using the SKMT-Generated Files

The source files generated from Figure 103 are included in the example project. These files provide the UFPK-wrapped user key information used to demonstrate the functionality described in section 8.1.

✓ 🚑 src	
> 🔁 SEGGER_RTT	
> h app_definitions.h	
> h common_utils.h	
> 🔂 hal_entry.c	
> c ra8m1_initial_aes128_key.c	
> h ra8m1_initial_aes128_key.h	



## 8.3 RSIP Compatibility Mode Key Injection APIs

This demonstration uses the APIs in the Key Injection module (r\_rsip\_key\_injection) to perform key injection. Refer to the FSP User Manual for the complete list of key injection APIs and their parameters.

## 8.4 Import and Compile the Example Project

Note that if AES keys other than the NIST vectors are used, then those new source files need to replace the existing files in the example project prior to compiling and running the example project. If the NIST vectors included in this application project are being used for verification purposes, step 2 can be skipped.

- 1. Launch e<sup>2</sup> studio and import secure\_key\_inject\_ra8m1.zip file to a workspace.
- 2. Replace ra8m1\_initial\_aes128\_key.h and ra8m1\_initial\_aes128\_key.c with the new set of files generated in Figure 106.
- 3. If different file names are used, update the #include definition in hal\_entry.c on this line to reflect the new file name.

<pre>#include</pre>	"hal_data.h"
<pre>#include</pre>	"common_utils.h"
<pre>#include</pre>	"app_definitions.h"
<pre>#include</pre>	"hw sce ra private.h"
#include	"ra8m1 initial aes128 key.h"

#### Figure 122. Include the Generated Header File for AES Operation

4. Next, double-click configuration.xml. Once the Configurator is opened, click Generate Project Content and then compile the secure project.

## 8.5 Running the Example Project

Follow the steps below to exercise the example projects:

- Right-click on secure\_key\_inject\_ra8m1 and select Debug As > Renesas GDB Hardware Debugging.
- 2. Execution will halt at the reset handler.



Figure 123. Running to the Project Reset Handler



- 3. Click **Resume** I twice to run the project.
- 4. Open the J-Link RTT Viewer with the settings shown below.

J-Link RTT Viewer V7.98b   Configuration	×
	~
Connection to J-Link	
O <u>U</u> SB	
○ <u>T</u> CP/IP	
Existing Session     Auto E	econnect
Use non-default port	
Specify Target Device	
R7FA8M1AH	×
Force go on connect	
Script file (optional)	
Target Interface & Speed	
SWD	▼ 4000 kHz ▼
RTT Control Block	
○ Auto <u>D</u> etection ○ <u>A</u> ddress	Search <u>R</u> ange
Enter one or more address range(s) the RTT Control blo Syntax: <rangestart [hex]=""> <rangesize>[, <range? Example: 0x1000000 0x1000, 0x2000000 0x1000</range? </rangesize></rangestart>	ock can be located in. IStart [Hex]> <range1size>,]</range1size>
0x22000000 0x8000	
	OK Cancel

#### Figure 124. RTT Viewer Setting

5. Click **OK**. The following execution result should be printed. Users can step into the code to understand the code execution flow.

00> 00>	Result:	Initial AES 128 Key Wrap is successful	
00> 00> 00>	Result:	Cryptographic operation is successful with initial wrapped AES 128 key	
00>			

Figure 125. Execution Result - Secure Key Injection Example Project for RA8M1

## 9. Example Project for RA6M3 (SCE7 Compatibility Mode)

This section introduces SCE7 Compatibility Mode with an example of AES-128 user key injection and update.

## 9.1 Overview

This example project demonstrates the following functionalities of the compatibility mode of SCE7:

- AES-128 key injection using the files generated in section 6.2.
- Verifying the injected AES-128 key using PSA Crypto APIs and a NIST AES test vector.

## 9.2 Using the SKMT-Generated Files

The source files generated Section 6.2 from Figure 106 are included in the example project. These files provide the UFPK-wrapped AES key source files used to demonstrate the functionality described above.



✓ <sup>™</sup> secure_key_inject_ra	i6m3 [Debug]
> 🌮 Binaries	
> 🔊 Includes	
> 🔑 ra	
> 😕 ra_gen	
✓ 2 src	
> 📂 SEGGER_RTT	
> h app_definitions	.h
> h common_utils.	h
> h crypto_ep.h	
> 🖻 hal_entry.c	
> 💼 initial_aes_128.	source files generated
> h initial_aes_128.	using SKMT

Figure 126. RA6M3 Example Project Source Code

## 9.3 SCE7 Compatibility Mode Key Injection APIs

This demonstration uses the APIs in the Key Injection module (r\_sce\_key\_injection) to perform key injection. Refer to the FSP User Manual for the complete list of key injection APIs and their parameters.

## 9.4 Import and Compile the Example Project

Note that if AES keys other than the NIST vectors are used, then those new source files need to replace the existing files in the example project prior to compiling and running the example project. If the NIST vectors included in this application project are being used for verification purposes, steps 2 to 5 can be skipped.

- 1. Launch  $e^2$  studio and import secure\_key\_inject\_ra6m3.zip file to a workspace.
- 2. Replace initial\_aes\_128.h and initial\_aes\_128.c with the new set of files generated in Figure 106.
- 3. If different file names are used, update the #include definition in hal\_entry.c on this line to reflect the new file name.



Figure 127. Include the Generated Header File for AES Operation

4. Next, double-click configuration.xml. Once the configurator is opened, click Generate Project Content and then compile the project.

## 9.5 Running the Example Project

Follow the steps below to exercise the example projects:

- 6. Right-click on secure\_key\_injection\_ra6m3 and select Debug As > Renesas GDB Hardware Debugging.
- 7. Execution will halt at the reset handler.

64 00016a18	SystemInit();
65	
66	<pre>/* Call user application. */</pre>
67 00016a1e	<pre>main();</pre>
68	
69 😑	while (1)
70	{
71	/* Infinite Loop. */
72 00016a22	}
73	}

#### Figure 128. Running to the Project Reset Handler



- 8. Click **Resume** II twice to run the project.
- 9. Open the J-Link RTT Viewer with the settings shown below.

Connection to J-Link	Serial No
○ TCP/IP	
O Existing Session	
Specify Target Device	~
Script file (optional)	
Target Interface & Speed	
SWD	▼ 4000 kHz ▼
RTT Control Block      Auto Detection     A	uddress 🔿 Search Range
J-Link automatically detects	the RTT control block.
	OK Cancel

#### Figure 129. RTT Viewer Setting

10. Click **OK**. The following execution result should be printed. Users can step into the code to understand the code execution flow.

00> 00> Result: AES 128 Key Wrap is successful 00>	
00> Result: Cryptographic operation is successful with wrapped AES 128 key	

Figure 130. Execution Result - Secure Key Injection for Example Project RA6M3

## 10. References

- 1. Renesas RA Family Device Lifecycle Management Key Injection Application Note (R11AN0469)
- 2. Renesas RA Family Secure Crypto Engine Operational Modes Application Note (R11AN0498)
- 3. Renesas RA Family MCU Security Design with TrustZone® IP Protection (R11AN0467)
- 4. Renesas RA Family MCU Plaintext Key Injection (R11AN0473)



## 11. Website and Support

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

EK-RA6M4 Resources EK-RA8M1 Resources EK-RA6M3 Resources RA Product Information Flexible Software Package (FSP) RA Product Support Forum Renesas Support renesas.com/ra/ek-ra6m4 renesas.com/ra/ek-ra8m1 renesas.com/ra/ek-ra6m3 renesas.com/ra renesas.com/ra/fsp renesas.com/ra/forum renesas.com/support



## **Revision History**

		Description	
Rev.	Date	Page	Summary
1.00	May.19.21	-	First release document
1.10	Jan.27.22	-	Update to use Security Key Management Tool CLI V1.0.0
1.20	Mar.25.22	-	Updated to add SKMT GUI support
1.30	Oct.25.22	-	Update to support SCE7 with FSP v4.0.0
2.00	Jan.03.24	-	Update to FSP v5.1.0
2.10	Oct.15.24	-	Update to FSP v5.5.0



## 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 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

- 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
  - "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, 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.
- This document shall not be reprinted, reproduced or duplicated in any form, in whole or in part, without prior written consent of Renesas Electronics.
   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: <u>www.renesas.com/contact/</u>.