

RX Family

Renesas Secure IP (RSIP) Module Protected Mode Firmware Integration Technology

Introduction

This document explains the usage of the software drivers for the Renesas Secure IP (RSIP) modules Protected Mode on RX Family microcontrollers (MCUs). These software drivers are referred to collectively as the RSIP Protected Mode (PM) driver.

The RSIP PM driver is provided as a Firmware Integration Technology (FIT) module. Refer to the webpage linked to below for an overview of FIT.

<https://www.renesas.com/us/en/products/software-tools/software-os-middleware-driver/software-package/fit.html>

The RSIP PM driver provides APIs for the cryptographic algorithms listed in Table 1 as well as for securely performing firmware updates.

Confirmed Devices

RX261 group

Table 1 RSIP-E11A Cryptographic Algorithms

Cipher Type	Algorithms	
Asymmetric key cryptography	Signature generation/verification	ECDSA (secp256r1, brainpoolP256r1, secp256k1): RFC6979
	Key generation	secp256r1, brainpoolP256r1, secp256k1
Symmetric key cryptography	AES	AES (128-/256-bit) ECB/CBC/CTR: FIPS 197, SP800-38A
Hashing	SHA	SHA-224, SHA-256: FIPS 180-4
Authenticated encryption with associated data (AEAD)		GCM/CCM: FIPS 197, SP800-38C, SP800-38D
Message authentication		CMAC (AES): FIPS 197, SP800-38B GMAC: RFC 4543 HMAC (SHA): RFC 2104
Pseudo-random bit generation		SP 800-90A
Random number generation		Tested with SP 800-90B.
Key wrapping		RFC3394
KDF		SP 800-56A, SP 800-56C ECC (secp256r1)

Note: [RFC 2104: HMAC: Keyed-Hashing for Message Authentication \(rfc-editor.org\)](#)

[RFC 4543: The Use of Galois Message Authentication Code \(GMAC\) in IPsec ESP and AH \(rfc-editor.org\)](#)

[RFC 6979 - Deterministic Usage of the Digital Signature Algorithm \(DSA\) and Elliptic Curve Digital Signature Algorithm \(ECDSA\) \(ietf.org\)](#)

[NIST SP 800-38A, Recommendation for Block Cipher Modes of Operation Methods and Techniques](#)

NIST SP 800-38-B [Recommendation for Block Cipher Modes of Operation: The CMAC Mode for Authentication \(nist.gov\)](#)

NIST SP 800-38D, [Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode \(GCM\) and GMAC](#)

NIST SP800-56A: [Recommendation for Pair-Wise Key Establishment Schemes Using Discrete Logarithm Cryptography \(nist.gov\)](#)

NIST SP800-56C: [Recommendation for Key-Derivation Methods in Key-Establishment Schemes \(nist.gov\)](#)

NIST SP800-22: <https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-22r1a.pdf>

NIST SP800-90A: <https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-90Ar1.pdf>

NIST SP800-90B: <https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-90B.pdf>

FIPS 180-4: <https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.180-4.pdf>

Contents

1. Overview.....	7
1.1 Terminology.....	7
1.2 RSIP Overview	8
1.3 Structure of Product Files.....	9
1.4 Development Environment.....	12
1.5 Code Size	13
1.6 Performance	14
1.6.1 RX231.....	14
2. API Information.....	18
3. RSIP PM Driver Usage.....	25
3.1 Recovering after Unauthorized Access Detection.....	25
3.2 Avoiding RSIP Access Conflicts.....	25
3.3 BSP FIT Module Integration.....	26
3.4 Single-Part and Multi-Part Operations	26
3.5 Initialization, Termination, and State Transitions of the Driver.....	26
3.6 Key Management	28
3.6.1 Key Injection and Updating	29
3.6.1.1 Key Wrapping Algorithm.....	30
3.6.2 Key Generation	32
3.6.3 Plaintext Public Key Extraction.....	32
3.7 Random Number Generation	33
3.8 Symmetric Key Cryptography.....	33
3.8.1 Symmetric Key Cryptography.....	33
3.8.2 Authenticated Encryption with Associated Data (AEAD)	33
3.8.3 Message Authentication Code (MAC).....	34
3.9 Asymmetric Key Cryptography.....	34
3.10 Hash Functions.....	34
3.10.1 Message Digest.....	34
3.10.2 Message Authentication Code (HMAC)	35
3.11 Key Wrap.....	35
3.12 Key Exchange/Key Derivation.....	36
3.12.1 Receiving Public Key.....	36
3.12.1.1 To Handle Verified Public Key (Extracting Public Key from Certificate)	36
3.12.1.2 To Handle Unverified Public Key	37
3.12.2 Generate Wrapped Secret	37
3.12.2.1 Generate Wrapped Secret with Verified Public Key	37
3.12.2.2 Generate Wrapped Secret with Unverified Public Key	38
3.12.3 Key Derivation	39

3.13	Firmware Update/Secure Boot.....	40
3.13.1	Secure Boot.....	41
3.13.2	Firmware Update.....	41
3.13.3	Encrypting the User Program.....	41
4.	API Functions	43
4.2	Detailed Descriptions of API Functions.....	46
4.2.1	Common Functions	46
4.2.1.1	R_RSIP_Open.....	46
4.2.1.2	R_RSIP_Close	47
4.2.1.3	R_RSIP_GetVersion	48
4.2.2	Key Management	49
4.2.2.1	R_RSIP_InitialKeyWrap	49
4.2.2.2	R_RSIP_EncryptedKeyWrap	51
4.2.2.3	R_RSIP_KeyGenerate	53
4.2.2.4	R_RSIP_KeyPairGenerate.....	54
4.2.2.5	R_RSIP_PublicKeyExport.....	55
4.2.3	Random Number Generation	56
4.2.3.1	R_RSIP_RandomNumberGenerate.....	56
4.2.4	AES	57
4.2.4.1	R_RSIP_AES_Cipher_Init.....	57
4.2.4.2	R_RSIP_AES_Cipher_Update.....	59
4.2.4.3	R_RSIP_AES_Cipher_Finish.....	60
4.2.4.4	R_RSIP_AES_AEAD_Init	61
4.2.4.5	R_RSIP_AES_AEAD_LengthsSet.....	63
4.2.4.6	R_RSIP_AES_AEAD_AADUpdate	64
4.2.4.7	R_RSIP_AES_AEAD_Update.....	65
4.2.4.8	R_RSIP_AES_AEAD_Finish.....	67
4.2.4.9	R_RSIP_AES_AEAD_Verify	68
4.2.4.10	R_RSIP_AES_MAC_Init	69
4.2.4.11	R_RSIP_AES_MAC_Update	70
4.2.4.12	R_RSIP_AES_MAC_SignFinish	71
4.2.4.13	R_RSIP_AES_MAC_VerifyFinish	72
4.2.5	ECC	73
4.2.5.1	R_RSIP_ECDSA_Sign.....	73
4.2.5.2	R_RSIP_ECDSA_Verify.....	74
4.2.5.3	R_RSIP_PKI_ECDSA_CertVerify	75
4.2.5.4	R_RSIP_PKI_CertKeyImport	76
4.2.5.5	R_RSIP_PKI_VerifiedCertInfoExport	78
4.2.5.6	R_RSIP_PKI_VerifiedCertInfoImport	79
4.2.5.7	R_RSIP_ECDH_KeyAgree	80

4.2.5.8 R_RSIP_ECDH_PlainKeyAgree	81
4.2.6 Hash	82
4.2.6.1 R_RSIP_SHA_Compute	82
4.2.6.2 R_RSIP_SHA_Init	83
4.2.6.3 R_RSIP_SHA_Update	84
4.2.6.4 R_RSIP_SHA_Finish	85
4.2.6.5 R_RSIP_SHA_Suspend	86
4.2.6.6 R_RSIP_SHA_Resume	87
4.2.6.7 R_RSIP_HMAC_Compute	88
4.2.6.8 R_RSIP_HMAC_Verify	89
4.2.6.9 R_RSIP_HMAC_Init	91
4.2.6.10 R_RSIP_HMAC_Update	92
4.2.6.11 R_RSIP_HMAC_SignFinish	93
4.2.6.12 R_RSIP_HMAC_VerifyFinish	94
4.2.6.13 R_RSIP_HMAC_Suspend	95
4.2.6.14 R_RSIP_HMAC_Resume	96
4.2.7 Key Derivation Function (KDF)	97
4.2.7.1 R_RSIP_KDF_SHA_Init	97
4.2.7.2 R_RSIP_KDF_SHA_ECDHSecretUpdate	98
4.2.7.3 R_RSIP_KDF_SHA_Update	99
4.2.7.4 R_RSIP_KDF_SHA_Finish	100
4.2.7.5 R_RSIP_KDF_SHA_Suspend	101
4.2.7.6 R_RSIP_KDF_SHA_Resume	102
4.2.7.7 R_RSIP_KDF_DKMConcatenate	103
4.2.7.8 R_RSIP_KDF_DerivedKeyImport	104
4.2.7.9 R_RSIP_KDF_DerivedIVWrap	106
4.2.8 Key Wrap	108
4.2.8.1 R_RSIP_RFC3394_KeyWrap	108
4.2.8.2 R_RSIP_RFC3394_KeyUnwrap	110
4.2.9 Firmware Update/Secure Boot	112
4.2.9.1 R_RSIP_FWUP_StartUpdateFirmware	112
4.2.9.2 R_RSIP_FWUP_MAC_Sign_Init	113
4.2.9.3 R_RSIP_FWUP_MAC_Sign_Update	115
4.2.9.4 R_RSIP_FWUP_MAC_Sign_Finish	116
4.2.9.5 R_RSIP_SB_MAC_Verify_Init	117
4.2.9.6 R_RSIP_SB_MAC_Verify_Update	118
4.2.9.7 R_RSIP_SB_MAC_Verify_Finish	119
5. Key Injection and Updating	120
5.1 Key Injection	120
5.2 Key Updating	121

5.3	User Key cryptographic format.....	122
5.4	Generating an Encrypted Key Using Security Key Management Tool	124
5.4.1	Key Injection Procedure	124
5.4.1.1	Procedure for CLI Version.....	124
5.4.1.2	Procedure for GUI Version.....	125
5.4.2	Key Update Operation Procedure	128
5.4.2.1	Procedure for CLI Version.....	128
5.4.2.2	Procedure for GUI Version.....	130
6.	Sample Programs.....	135
6.1	Confirming Operation of How to Use Key Injection and Cryptography.....	135
6.1.1	Setting Up the Demo Project.....	136
6.1.2	Overview of Demo Project.....	137
6.1.2.1	Characteristics of Wrapped Keys and how to check them in the demo project.....	138
6.1.3	Demo Project Execution Example.....	138
6.2	Confirming Operation of the Secure Boot and Firmware Update	141
6.2.1	Setting Up the Demo Project.....	142
6.2.2	Overview of Secure Bootloader and Firmware Update Demo Projects	145
6.2.2.1	Secure Bootloader project.....	146
6.2.2.2	Firmware Update Project.....	148
6.2.3	Execution Example of the Demo Project.....	150
6.2.3.1	Execution Example of Two-Step Setup.....	150
6.2.3.2	Execution Example of Inclusive Setup	159
6.2.4	Debugging Firmware Update Project	167
6.2.5	Notes on Transition from Secure Bootloader Project to Firmware Update Project.....	168
7.	Appendix.....	169
7.1	Confirmed Operation Environment.....	169
7.2	Troubleshooting.....	170
7.3	Encrypted Key Generation in Dynamic Operation	171
8.	Reference Documents	173
	Revision History	174

1. Overview

1.1 Terminology

Terms used in this document are defined below.

Table 1-1 Descriptions of Terms

Term	Description
Key injection	Injecting a wrapped key into the device at the factory.
Key updating	Injecting a wrapped key into the device in the field.
User key	An encryption key in plaintext used by the user. Not used on the device. For AES and HMAC, user keys are used as shared keys. For ECC, user keys are used as public keys and secret keys.
Encrypted key	Key information generated by adding a MAC value and encrypting a user key using a UFPK or update key. An encrypted key corresponding to the same user key is a shared value on each device.
Wrapped key	Data consisting of an encrypted key that has been converted into a form that is usable by the RSIP PM driver by key injection or key updating. The wrapped key has been wrapped using an HUK, so the wrapped key of the same encrypted key will be a unique value on each device.
UFPK	User Factory Programming Key A keyring set by the user and used to generate an encrypted key from a user key during key injection. Not used on the device.
W-UFPK	Wrapped UFPK Key information generated by wrapping a UFPK using an HRK on the DLM server. The UFPK is decrypted using the HRK internally by the TSIP.
Key update key (KUK)	A key set by the user and used to generate an encrypted key from a user key during key updating. The wrapped KUK for the update key must be generated beforehand by key injection in order to perform key updating on the device.
Hardware root key (HRK)	A shared encryption key that exists only inside the TSIP and in secure rooms within Renesas.
Hardware unique key (HUK)	A device-specific encryption key that is derived internally by the TSIP and used to protect key data.
Device Lifecycle Management (DLM) server	A key management server operated by Renesas. It is used for wrapping UFPK.
Image Encryption Key	256-bit key used in firmware encryption for use with the firmware update API
Key Encryption Key	AES 128bit key used in Image Encryption Key wrap

1.2 RSIP Overview

The Renesas Secure IP (RSIP) block on RX Family MCUs creates a secure area inside the MCU by monitoring for unauthorized access attempts. This ensures that the RSIP can utilize the encryption engine and user key (encryption key) reliably and securely. The RSIP handles the encryption key in a format called a wrapped key that is secure and unreadable outside the RSIP block. This means that the encryption key, the most important element in reliable and secure encryption, can be stored in the flash memory.

The RSIP block has a safe area that contains the encryption engine and storage for the encryption key in plaintext format.

The RSIP restores from the wrapped key the encryption key used for cryptographic operations. The Wrapped Key is generated tied to the HUK, which is a device-specific value. This means that even if a wrapped key is copied from one device to a different device it cannot be used on the second device. To access the TSIP hardware an application must use the RSIP PM driver.

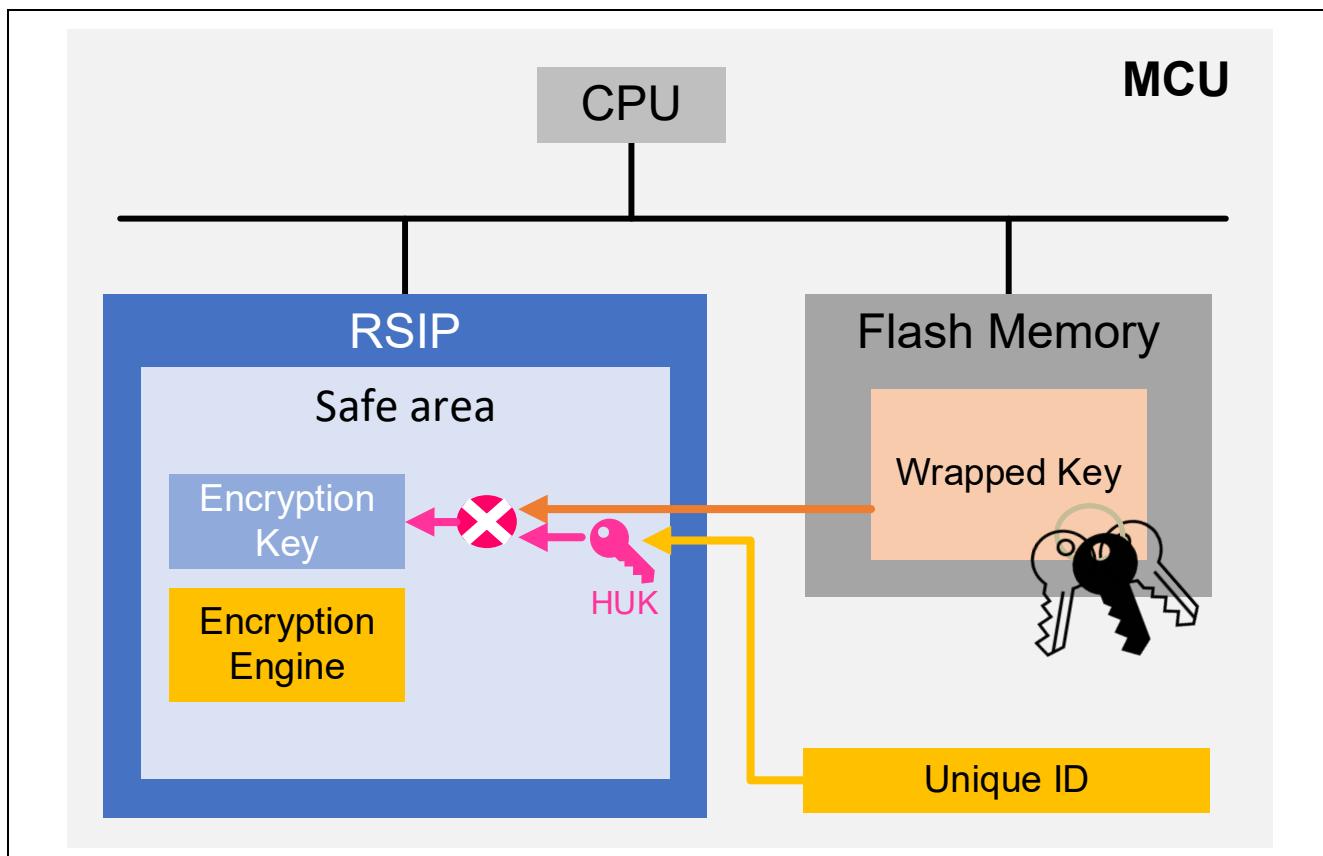


Figure 1.1 MCU Incorporating RSIP

1.3 Structure of Product Files

Table 1.2 below lists the files included in the product.

Table 1-2 Structure of Product Files

File/Directory (Bold) Name	Description
r20an0748jj0200-rx-rsip-security.pdf	RSIP PM driver application note (Japanese)
r20an0748ej0200-rx-rsip-security.pdf	RSIP PM driver application note (English)
reference_documents	Folder containing documentation of topics such as how to use the FIT module with various integrated development environments
ja	Folder containing documentation of topics such as how to use the FIT module with various integrated development environments (Japanese)
r01an1826jj0110-rx.pdf	How to add FIT modules to CS+ projects (Japanese)
r01an1723ju0121-rx.pdf	How to add FIT modules to e ² studio projects (Japanese)
r20an0451js0140-e2studio-sc.pdf	Smart Configurator user's guide (Japanese)
en	Folder containing documentation of topics such as how to use the FIT module with various integrated development environments (English)
r01an1826ej0110-rx.pdf	How to add FIT modules to CS+ projects (English)
r01an1723eu0121-rx.pdf	How to add FIT modules to e ² studio projects (English)
r20an0451es0140-e2studio-sc.pdf	Smart Configurator user's guide (English)
FITModules	FIT module folder
r_rsig_protected_rx_v2.00.zip	RSIP PM driver FIT module
r_rsig_protected_rx_v2.00.xml	RSIP PM driver FIT module e ² studio FIT plug-in XML file
r_rsig_protected_rx_v2.00_extend.mdf	RSIP PM driver FIT module Smart Configurator configuration settings file
FITDemos	Demo project folder
rx261_ek_rsig_sample	Project showing how to write keys and use cryptographic APIs
rx261_ek_rsig_secure_boot*^{1*2}	Secure boot/secure update implementation example
rx261_ek_rsig_secure_boot*^{1*2}	Secure boot firmware
rx261_ek_rsig_user_program*^{1*2}	User program following secure update

Table 1.3 lists the files and folders contained in the folder produced by unzipping r_rsip_protected_rx_v.2.00.zip.

Table 1-3 File Structure

File/Directory (Bold) Names	Description
r_config	RSIP PM driver config file folder
r_rsip_protected_rx_config.h	RSIP PM driver config file (default settings)
r_rsip_protected_rx	RSIP PM driver FIT module folder
src	RSIP PM driver source code folder
inc	RSIP PM driver header file folder
rx261	Folder containing program code dependent on specific MCU models
r_rsip.h	API header file to standardize model-independent portions
r_rsip_cfg.h	Config header file to standardize model-independent portions
primitive	RSIP PM driver source code folder
common	Folder for storing program code for microcontroller model-independent portions
r_rsip_err.h	Return value header file
r_rsip_util.h	Utility header file
rx261	Folder containing program code dependent on specific MCU models
r_rsip_rx261_iodefine.h	RSIP access header file
r_rsip_primitive.h	Header files for RSIP access source code
r_rsip_rx_finctionxxx.c	Source files for RSIP access source code xxx in the file name is a number.
r_rsip_rx_pxx.c	Source files for RSIP access source code xxx in the file name is a number.
s_flash.c	Key information file
private	Folder for storing program code of RSIP PM driver internal functions
common	Folder for storing program code for microcontroller model-independent portions
r_rsip_private.c	Source code for internal functions
r_rsip_private.h	Header files for internal functions
rx261	Folder for storing program code for microcontroller model-dependent portions
r_rsip_hal.c	Source code for RSIP PM driver HW dependent functions
r_rsip_wrapper.c	Source code for RSIP PM driver wrapper APIs
r_rsip_wrapper.h	Header files for RSIP PM driver wrapper APIs
public	Folder for storing programs for RSIP PM driver API
common	Folder for storing program code for microcontroller model-independent portions
r_rsip.c	Source code for RSIP PM driver common function API
r_rsip_aes.c	Source code for RSIP PM driver AES API
r_rsip_ecc.c	Source code for RSIP PM driver ECC API
r_rsip_kdf.c	Source code for RSIP PM driver KDF API
r_rsip_otf.c	Source code for RSIP PM driver OTF API
r_rsip_pk.c	Source code for RSIP PM driver PKI API
r_rsip_sha.c	Source code for RSIP PM driver HASH API
r_rsip_rsa.c	Source code for RSIP PM driver RSA API

File/Directory (Bold) Names		Description
	r_rsip_public.h	Header file for RSIP PM driver function API
	rx261	Folder for storing program code for microcontroller model-dependent portions
	r_rsip_rx.c	Source code for RSIP PM driver FIT Module API
	r_rsip_fwup.c	Source code for RSIP PM driver Firmware Update API
	r_rsip_key_injection.c	Source code for RSIP PM driver Key Injection API
	doc	Document folder
	ja	Document folder (Japanese)
	r20an0748jj0200-rx-rsip-security.pdf	RSIP PM driver application note (Japanese)
	en	Document folder (English)
	r20an0748ej0200-rx-rsip-security.pdf	RSIP PM driver application note (English)
	r_rsip_protected_rx_if.h	RSIP PM driver header file
	readme.txt	Readme

1.4 Development Environment

The RSIP PM driver was developed using the tools described below. When developing your own applications, use the versions of the software indicated below, or newer.

1. Integrated development environment

Refer to the “Integrated development environment” item under 7.1, Confirmed Operation Environment.

2. C compiler

Refer to the “C compiler” item under 7.1, Confirmed Operation Environment.

3. Emulator/debugger

E2 Lite

4. Evaluation boards

Refer to the “Board used” item under 7.1, Confirmed Operation Environment. All of the boards listed are special product versions with cryptographic functionality. Make sure to confirm the product model name before ordering. e² studio and CC-RX were used in combination for evaluation and to create the demo project.

The project conversion function can be used to convert projects from e² studio to CS+. If you encounter errors such as compiler errors, please contact your Renesas representative.

1.5 Code Size

The table below lists the ROM and RAM sizes and the maximum stack usage associated with this module.

The actual ROM (code and constants) and RAM (global data) sizes are determined by the configuration options listed in 2.6, Configuration.

The values listed in the table below have been confirmed under the following conditions:

Module revision: r_rsip_protected_rx rev2.00

Compiler version: Renesas Electronics C/C++ Compiler Package for RX Family V3.07.00
(Optimization Level 2 with “-lang = c99” option added)

GCC for Renesas RX 8.3.0.202311
(Optimization Level -Os with “-std = gnu99” option added)

IAR C/C++ Compiler for Renesas RX version 5.10.01
(Optimization level high(balance))

Configuration options:

Renesas Electronics C/C++ Compiler Package for RX Family: -isa=rxv3, Optimization Level 2

GCC for Renesas RX: RXv3, Optimization Level -Os

IAR C/C++ Compiler for Renesas RX: --core rxv3 -Oh, Optimization level high(balance)

Category	Memory Used		
	Renesas Compiler	GCC*	IAR Compiler
ROM	99,884 bytes	100,437 bytes	98,378 bytes
RAM	12 bytes	12 bytes	12 bytes
Stack	288 bytes	296 bytes	264 bytes

1.6 Performance

Performance is measured in cycles of ICLK, the core clock. The operating clock (PCLKB) for RSIP is set to ICLK : PCLKB = 2 : 1. The drivers are built using CC-RX with optimization level 2. Refer to 7.1, Confirmed Operation Environment, for version information. The configuration options are left in their default settings.

1.6.1 RX231

Table 1-4 Performance of Common APIs

API	Performance (Unit: Cycle)
R_RSIP_Open	790,000
R_RSIP_Close	450
R_RSIP_GetVersion	30

Table 1-5 Performance of Key management APIs

API	Performance (Unit: Cycle)
R_RSIP_EncryptedKeyWrap	11,000
R_RSIP_KeyGenerate	5,700
R_RSIP_KeyPairGenerate	9,300,000
R_RSIP_PublicKeyExport	130
R_RSIP_InitialKeyWrap	9,200

Table 1-6 Performance of Random Number Generate API

API	Performance (Unit: Cycle)
R_RSIP_RandomNumberGenerate	1,000

Table 1-7 Performance of AES APIs

Algorithm	API	Performance (Unit: Cycle)		
		48-Byte Processing	64-Byte Processing	80-Byte Processing
ECB Encryption	R_RSIP_AES_Cipher_Init	4,500	4,500	4,500
	R_RSIP_AES_Cipher_Update	840	950	1,100
	R_RSIP_AES_Cipher_Finish	460	460	460
ECB Decryption	R_RSIP_AES_Cipher_Init	4,500	4,500	4,500
	R_RSIP_AES_Cipher_Update	980	1,100	1,200
	R_RSIP_AES_Cipher_Finish	470	470	470
CBC Encryption	R_RSIP_AES_Cipher_Init	5,500	5,500	5,500
	R_RSIP_AES_Cipher_Update	880	990	1,100
	R_RSIP_AES_Cipher_Finish	480	480	480
CBC Decryption	R_RSIP_AES_Cipher_Init	5,500	5,500	5,500
	R_RSIP_AES_Cipher_Update	1,000	1,100	1,200
	R_RSIP_AES_Cipher_Finish	490	490	490
CTR	R_RSIP_AES_Cipher_Init	4,600	4,600	4,600
	R_RSIP_AES_Cipher_Update	930	1,000	1,100
	R_RSIP_AES_Cipher_Finish	500	500	500

Table 1-8 Performance of AES AEAD APIs

Algorithm	API	Performance (Unit: Cycle)		
		48-Byte Processing	64-Byte Processing	80-Byte Processing
GCM Encryption	R_RSIP_AES_AEAD_Init	5,300	5,300	5,300
	R_RSIP_AES_AEAD_AADUpdate	430	430	430
	R_RSIP_AES_AEAD_Update	1,600	1,800	2,000
	R_RSIP_AES_AEAD_Finish	1,300	1,300	1,300
GCM Decryption	R_RSIP_AES_AEAD_Init	5,300	5,300	5,300
	R_RSIP_AES_AEAD_AADUpdate	430	430	430
	R_RSIP_AES_AEAD_Update	1,700	1,900	2,100
	R_RSIP_AES_AEAD_Verify	1,700	1,700	1,700
CCM Encryption	R_RSIP_AES_AEAD_Init	240	240	240
	R_RSIP_AES_AEAD_LengthsSet	50	50	50
	R_RSIP_AES_AEAD_AADUpdate	4,900	4,900	4,900
	R_RSIP_AES_AEAD_Update	1,700	1,900	2,100
	R_RSIP_AES_AEAD_Finish	1,100	1,100	1,100
CCM Decryption	R_RSIP_AES_AEAD_Init	240	240	260
	R_RSIP_AES_AEAD_LengthsSet	60	60	60
	R_RSIP_AES_AEAD_AADUpdate	4,900	4,900	4,900
	R_RSIP_AES_AEAD_Update	1,600	1,800	2,000
	R_RSIP_AES_AEAD_Verify	1,600	1,600	1,600

Note: GCM performance was measured with parameters fixed as follows: ivec = 128 bits, AAD = 128 bits, and authentication tag = 128 bits.

CCM performance was measured with parameters fixed as follows: nonce = 56 bits, AAD = 64 bits, and MAC = 32 bits.

Table 1-9 Performance of AES MAC APIs

Algorithm	API	Performance (Unit: Cycle)		
		48-Byte Processing	64-Byte Processing	80-Byte Processing
Generate CMAC	R_RSIP_AES_MAC_Init	4,100	4,100	4,100
	R_RSIP_AES_MAC_Update	770	880	990
	R_RSIP_AES_MAC_SignFinish	980	990	990
Verify CMAC	R_RSIP_AES_MAC_Init	4,100	4,100	4,100
	R_RSIP_AES_MAC_Update	770	880	990
	R_RSIP_AES_MAC_VerifyFinish	1,500	1,500	1,500

Table 1-10 Performance of ECC APIs

Algorithm	API	Performance (Unit: Cycle)		
		48-Byte Processing	64-Byte Processing	80-Byte Processing
secp256r1	R_RSIP_ECDSA_Sign	9,400,000	9,400,000	9,400,000
	R_RSIP_ECDSA_Verify	4,400,000	4,600,000	4,500,000
secp256k1	R_RSIP_ECDSA_Sign	9,400,000	9,400,000	9,400,000
	R_RSIP_ECDSA_Verify	4,400,000	4,500,000	4,500,000
brainpoolP256r1	R_RSIP_ECDSA_Sign	9,400,000	9,400,000	9,400,000
	R_RSIP_ECDSA_Verify	4,400,000	4,400,000	4,400,000

Table 1-11 Performance of KDF APIs

API	Performance (Unit: Cycle)
R_RSIP_PKI_ECDSA_CertVerify	5,200,000
R_RSIP_PKI_CertKeyImport	810,000
R_RSIP_PKI_VerifiedCertInfoExport	70
R_RSIP_PKI_VerifiedCertInfoImport	80
R_RSIP_ECDH_KeyAgree	9,100,000
R_RSIP_ECDH_PlainKeyAgree	9,100,000
R_RSIP_KDF_SHA_Init	70
R_RSIP_KDF_SHA_ECDHSecretUpdate	9,600
R_RSIP_KDF_SHA_Update	90
R_RSIP_KDF_SHA_Suspend	230
R_RSIP_KDF_SHA_Resume	210
R_RSIP_KDF_SHA_Finish	6,900
R_RSIP_DKMC.concatenate	110
R_RSIP_KDF_DerivedKeyImport	30,000
R_RSIP_KDF_DerivedIVWrap	30,000

Table 1-12 Performance of HASH APIs

Algorithm	API	Performance (Unit: Cycle)		
		48-Byte Processing	64-Byte Processing	80-Byte Processing
SHA224	R_RSIP_SHA_Compute	2,000	3,300	3,300
	R_RSIP_SHA_Init	50	50	50
	R_RSIP_SHA_Update	130	140	1,900
	R_RSIP_SHA_Finish	2,000	3,300	3,300
SHA256	R_RSIP_SHA_Compute	2,000	3,300	3,300
	R_RSIP_SHA_Init	50	50	50
	R_RSIP_SHA_Update	130	140	1,900
	R_RSIP_SHA_Finish	2,000	3,300	2,000
-	R_RSIP_SHA_Suspend	230		
	R_RSIP_SHA_Resume	210		

Table 1-13 Performance of HMAC APIs

Algorithm	API	Performance (Unit: Cycle)		
		48-Byte Processing	64-Byte Processing	80-Byte Processing
Generate HMAC-SHA224	R_RSIP_HMAC_Compute	11,000	12,000	12,000
	R_RSIP_HMAC_Init	110	110	110
	R_RSIP_HMAC_Update	120	140	7,100
	R_RSIP_HMAC_SignFinish	11,000	12,000	5,300
Verify HMAC-SHA224	R_RSIP_HMAC_Verify	11,000	13,000	13,000
	R_RSIP_HMAC_Init	110	110	120
	R_RSIP_HMAC_Update	120	130	7,100
	R_RSIP_HMAC_VerifyFinish	11,000	13,000	6,100
Generate HMAC-SHA256	R_RSIP_HMAC_Compute	11,000	12,000	12,000
	R_RSIP_HMAC_Init	110	110	110
	R_RSIP_HMAC_Update	120	130	7,100
	R_RSIP_HMAC_SignFinish	11,000	12,000	5,100
Verify HMAC-SHA256	R_RSIP_HMAC_Verify	11,000	13,000	13,000
	R_RSIP_HMAC_Init	110	110	110
	R_RSIP_HMAC_Update	120	130	7,100
	R_RSIP_HMAC_VerifyFinish	11,000	13,000	6,100
-	R_RSIP_HMAC_Suspend	670		
	R_RSIP_HMAC_Resume	260		

Table 1-14 Performance of Key Wrap APIs

API	Performance (Unit: Cycle)	
	Wrapping Key AES-128	Wrapping Key AES-256
R_RSIP_RFC3394_KeyWrap	16,000	23,000
R_RSIP_RFC3394_KeyUnwrap	18,000	25,000

Table 1-15 Performance of Secure Boot APIs

API	Performance (Unit: Cycle)		
	2K-Byte Processing	4K-Byte Processing	6K-Byte Processing
R_RSIP_SB_MAC_Verify_Init	2,100	2,100	2,100
R_RSIP_SB_MAC_Verify_Update	18,000	36,000	53,000
R_RSIP_SB_MAC_Verify_Finish	18,000	36,000	54,000

2. API Information

2.1 Hardware Requirements

RSIP PM drivers can only be used with devices provided with a RSIP. Check the product number of the device to ensure that it incorporates a RSIP.

2.2 Software Requirements

The RSIP PM drivers are dependent on the following module:

r_bsp Use rev. 7.51 or later. (BSP stands for “board support package.”)

Change the value in the following macro in r_bsp_config.h in the r_config folder to 0xB.

```
/* Chip version.  
Character(s) = Value for macro =  
A           = 0xA           = Chip version A  
              = Encryption module not included, USB  
included, CAN FD included (only CAN 2.0 protocol supported)  
B           = 0xB           = Chip version B  
              = Encryption module and USB included, CAN FD  
included  
*/  
#define BSP_CFG_MCU_PART_VERSION      (0xB)
```

2.3 Supported Toolchain

The operation of the RSIP PM driver has been confirmed with the toolchain indicated in 7.1, Confirmed Operation Environment.

2.4 Header File

All API calls and their supported interface definitions are contained in r_rsip_protected_rx_if.h.

2.5 Integer Types

The RSIP PM driver uses ANSI C99 integer types defined in stdint.h.

2.6 Configuration

By setting the values of the following macros to 1 or 0 in /r_config/r_rsip_protected_rx_config.h you can turn the corresponding functions on or off.

Table 2-1 Configuration Options /r_rsip_protected_rx_config.h

Definition	Default Value	Meanings
RSIP_CFG_PARAM_CHECKING_ENABLE	1	Enable parameter check 1 : enabled 0: disabled
RSIP_CFG_AES_128_ENABLE	0	Enable AES 128bit 1 : enabled 0: disabled
RSIP_CFG_AES_256_ENABLE	0	Enable AES 256bit 1 : enabled 0: disabled
RSIP_CFG_AES_ECB_CBC_CTR_ENABLE	0	Enable AES ECB/CBC/CTR mode 1 : enabled 0: disabled
RSIP_CFG_AES_GCM_ENABLE	0	Enable AES GCM mode 1 : enabled 0: disabled
RSIP_CFG_AES_CCM_ENABLE	0	Enable AES CCM mode 1 : enabled 0: disabled
RSIP_CFG_AES_CMAC_ENABLE	0	Enable AES CMAC 1 : enabled 0: disabled
RSIP_CFG_ECC_SECP256R1_ENABLE	0	Enable ECC secp256r1 1 : enabled 0: disabled
RSIP_CFG_SHA224_ENABLE	0	Enable SHA224 1 : enabled 0: disabled
RSIP_CFG_SHA256_ENABLE	0	Enable SHA256 1 : enabled 0: disabled
RSIP_CFG_HMAC_SHA224_ENABLE	0	Enable HMAC-SHA224 1 : enabled 0: disabled
RSIP_CFG_HMAC_SHA256_ENABLE	0	Enable HMAC-SHA256 1 : enabled 0: disabled
RSIP_CFG_KDF_SHA256_ENABLE	0	Enable KDF-SHA256 1 : enabled 0: disabled
RSIP_CFG_SECURE_BOOT	0	Enable Secure Boot 1 : enabled 0: disabled
RSIP_CFG_FIRMWARE_UPDATE	0	Enable Firmware Update 1 : enabled 0: disabled

2.7 Type Definition

The following table shows the type definitions used in the RSIP PM driver.

Table 2-2 RSIP PM Driver Type Definition

Definition	Data Type	Remarks
rsip_ctrl_t	void	Type definitions for API arguments of the RSIP PM driver management structure Use rsip_instance_ctrl_t for the data type to be used for the arguments.

2.8 Data Structures

The following table shows the data structures used in the RSIP PM driver.

Table 2-3 RSIP PM Driver Data Structures

Definition	Remarks
rsip_instance_ctrl_t	Instance of RSIP PM driver management structure
rsip_cfg_t	Configuration Structure Not used in RX RSIP PM driver.
rsip_wrapped_key_t	Wrapped Key
rsip_sha_handle_t	Work area for SHA calculation
rsip_hmac_handle_t	Work area for HMAC calculation
rsip_wrapped_secret_t	Wrapped Secret
rsip_kdf_sha_handle_t	Work area for KDF SHA calculation
rsip_wrapped_dkm_t	Wrapped DKM
rsip_verified_cert_info_t	Verified Certificate Information

2.9 Enumerated Type

The following is the definition of the enumerated type used in the RSIP PM driver.

Table 2-4 Key Type enum rsip_key_type_t

Definition	Value	Remarks
RSIP_KEY_TYPE_INVALID	0x00000000	Invalid
RSIP_KEY_TYPE_AES_128	0x10000004	AES128bit key
RSIP_KEY_TYPE_AES_256	0x10000008	AES256bit key
RSIP_KEY_TYPE_ECC_SECP256R1_PUBLIC	0x04000008	secp256r1 public key
RSIP_KEY_TYPE_ECC_SECP256K1_PUBLIC	0x04030008	secp256k1 public key
RSIP_KEY_TYPE_ECC_BRAINPOOL_P256R1_PUBLIC	0x04030004	brainpoolP256r1 public key
RSIP_KEY_TYPE_ECC_SECP256R1_PRIVATE	0x05000008	secp256r1 private key
RSIP_KEY_TYPE_ECC_SECP256K1_PRIVATE	0x05030008	secp256k1 private key
RSIP_KEY_TYPE_ECC_BRAINPOOL_P256R1_PRIVATE	0x05030004	brainpoolP256r1 private key
RSIP_KEY_TYPE_HMAC_SHA224	0x08010008	HMAC SHA224 key
RSIP_KEY_TYPE_HMAC_SHA256	0x08020008	HMAC SHA256 key
RSIP_KEY_TYPE_KUK	0xff000008	Key Update Key

Table 2-5 HASH Type enum rsip_hash_type_t

Definition	Value	Remarks
RSIP_HASH_TYPE_SHA224	0	SHA224
RSIP_HASH_TYPE_SHA256	1	SHA256

Table 2-6 AES Mode Type enum rsip_aes_cipher_type_t

Definition	Value	Remarks
RSIP_AES_CIPHER_MODE_ECB_ENC	0	AES ECB mode encryption
RSIP_AES_CIPHER_MODE_ECB_DEC	1	AES ECB mode decryption
RSIP_AES_CIPHER_MODE_CBC_ENC	2	AES CBC mode encryption
RSIP_AES_CIPHER_MODE_CBC_DEC	3	AES CBC mode decryption
RSIP_AES_CIPHER_MODE_CTR	4	AES CTR mode
RSIP_AES_CIPHER_MODE_ECB_ENC_WWRAPPED_IV	7	AES ECB mode encryption with wrapped IV
RSIP_AES_CIPHER_MODE_ECB_DEC_WWRAPPED_IV	8	AES ECB mode decryption with wrapped IV

Table 2-7 AES AEAD Mode Type enum rsip_aes_aead_type_t

Definition	Value	Remarks
RSIP_AES_AEAD_MODE_GCM_ENC	0	AES GCM mode encryption
RSIP_AES_AEAD_MODE_GCM_DEC	1	AES GCM mode decryption
RSIP_AES_AEAD_MODE_CCM_ENC	2	AES CCM mode encryption
RSIP_AES_AEAD_MODE_CCM_DEC	3	AES CCM mode decryption
RSIP_AES_AEAD_MODE_GCM_ENC_WWRAPPED_IV	4	AES GCM mode encryption with wrapped IV
RSIP_AES_AEAD_MODE_GCM_DEC_WWRAPPED_IV	5	AES GCM mode decryption with wrapped IV

Table 2-8 AES MAC Mode Type enum rsip_mac_type_t

Definition	Value	Remarks
RSIP_AES_MAC_MODE_CMAC	0	AES CMAC mode

Table 2-9 Initial Vector Type enum rsip_initial_vector_type_t

Definition	Value	Remarks
RSIP_INITIAL_VECTOR_TYPE_AES_16_BYTE	0	16-byte initial vector for AES

Table 2-10 Byte Size of the wrapped_key_t Structure enum rsip_byte_size_wrapped_key_t

Definition	Value	Remarks
RSIP_BYTE_SIZE_WRAPPED_KEY_AES_128	36U	AES128 key
RSIP_BYTE_SIZE_WRAPPED_KEY_AES_256	52U	AES256 key
RSIP_BYTE_SIZE_WRAPPED_KEY_ECC_SECP256R1_P UBLIC	84U	secp256r1 public key
RSIP_BYTE_SIZE_WRAPPED_KEY_ECC_SECP256K1_P UBLIC	84U	secp256k1public key
RSIP_BYTE_SIZE_WRAPPED_KEY_ECC_BRAINPOOL25 6R1_PUBLIC	84U	brainpoolP256r1 public key
RSIP_BYTE_SIZE_WRAPPED_KEY_ECC_SECP256R1_P RIVATE	52U	secp256r1 private key
RSIP_BYTE_SIZE_WRAPPED_KEY_ECC_SECP256K1_P RIVATE	52U	secp256k1 private key
RSIP_BYTE_SIZE_WRAPPED_KEY_ECC_BRAINPOOL25 6R1_PRIVATE	52U	brainpoolP256r1 private key
RSIP_BYTE_SIZE_WRAPPED_KEY_HMAC_SHA224	52U	HMAC_SHA224 key
RSIP_BYTE_SIZE_WRAPPED_KEY_HMAC_SHA256	52U	HMAC_SHA256 key
RSIP_BYTE_SIZE_WRAPPED_KEY_KUK	52U	Key Update Key

2.10 Return Values

Below are the return values used in the API functions of the RSIP PM driver. The enumeration type of return values is defined as `fsp_err_t` in `/r_bsp/mcu/all/fsp_common_api.h`.

Table 2-11 Return Values enum `fsp_err_t`

Definition	Value	Remarks
<code>FSP_SUCCESS</code>	0x00000	SUCCESS
<code>FSP_ERR_ASSERTION</code>	0x00001	Pointer argument is NULL
<code>FSP_ERR_INVALID_ARGUMENT</code>	0x00003	An invalid value was entered.
<code>FSP_ERR_UNSUPPORTED</code>	0x00006	Selected mode is unsupported
<code>FSP_ERR_NOT_OPEN</code>	0x00007	RSIP driver is not open
<code>FSP_ERR_ALREADY_OPEN</code>	0x0000e	RSIP driver is already open
<code>FSP_ERR_NOT_ENABLED</code>	0x00013	Unable to perform the specified process.
<code>FSP_ERR_INVALID_SIZE</code>	0x00017	Input size incorrect
<code>FSP_ERR_INVALID_STATE</code>	0x0001e	Illegal call state
<code>FSP_ERR_CRYPTO_RSIP_RESOURCE_CONFLICT</code>	0x10100	HW resource conflict occurs
<code>FSP_ERR_CRYPTO_RSIP_FATAL</code>	0x10101	Fatal error
<code>FSP_ERR_CRYPTO_RSIP_FAIL</code>	0x10102	RSIP internal error
<code>FSP_ERR_CRYPTO_RSIP_KEY_SET_FAIL</code>	0x10103	Incorrect key value entered.
<code>FSP_ERR_CRYPTO_RSIP_AUTHENTICATION</code>	0x10104	Failed verification

2.11 Adding the FIT Module to Your Project

This module must be added to each project in which it is used. Renesas recommends using Smart Configurator as described in (1) or (3) below. However, Smart Configurator does not support all RX devices. If your RX device is not supported, use the method described in (2) or (4).

(1) Adding the FIT module to your project using Smart Configurator in e² studio

Using Smart Configurator in e² studio allows you to add the FIT module to your project automatically. Refer to the application note "Renesas e² studio Smart Configurator User Guide" (R20AN0451) for details.

(2) Adding the FIT module to your project using FIT Configurator in e² studio

Using FIT Configurator in e² studio allows you to add the FIT module to your project automatically. Refer to the application note "Adding Firmware Integration Technology Modules to Projects (R01AN1723)" for details.

(3) Adding the FIT module to your project using Smart Configurator in CS+

Using Smart Configurator Standalone Version in CS+ allows you to add the FIT module to your project automatically. Refer to the application note "Renesas e² studio Smart Configurator User Guide (R20AN0451)" for details.

(4) Adding the FIT module to your project in CS+

Manually add the FIT module to your project in CS+. Refer to the application note "Adding Firmware Integration Technology Modules to CS+ Projects (R01AN1826)" for details.

2.12 for, while, and do while Statements

The RSIP PM driver uses *for*, *while*, and *do while* statements (loop processing) to wait for registers to be updated, etc. Such loop processing is indicated in the comments with the keyword WAIT_LOOP. If you wish to incorporate fail-safe processing into loop processing, you can locate the corresponding processing by searching for the keyword WAIT_LOOP.

Devices for which WAIT_LOOP appears in the comments:

All device groups

A code sample is shown below.

Example *while* statement:

```
/* WAIT_LOOP */
while(0 == SYSTEM.OSCOVFSR.BIT.PLOVF)
{
    /* The delay period needed is to make sure that the PLL has stabilized. */
}
```

Example *for* statement:

```
/* Initialize reference counters to 0. */
/* WAIT_LOOP */
for (i = 0; i < BSP_REG_PROTECT_TOTAL_ITEMS; i++)
{
    g_protect_counters[i] = 0;
}
```

Example *do while* statement:

```
/* Reset completion waiting */
do
{
    reg = phy_read(ether_channel, PHY_REG_CONTROL);
    count++;
} while ((reg & PHY_CONTROL_RESET) && (count < ETHER_CFG_PHY_DELAY_RESET)); /* WAIT_LOOP */
```

3. RSIP PM Driver Usage

The RSIP PM driver for the RX family provides the following functions:

- Random number generation
- Secure key management
- Unauthorized access monitoring
- Acceleration of cryptographic operations

The keys handled by the RSIP PM driver (input and output keys) are opaque keys wrapped using a device-specific key called a hardware unique key (HUK), which is accessible only by the RSIP. In the case of the RX RSIP PM driver, this type of opaque key is called a wrapped key. The RSIP PM driver implements secure key management by wrapping keys using the HUK. This provides key confidentiality and detection of tampering outside of the RSIP.

Unauthorized access monitoring by the RSIP covers all cryptographic processing performed by the driver and is always enabled during cryptographic operations. If tampering with cryptographic operations is detected while the driver is in use, the driver stops operation.

There are two types of APIs provided by the RSIP PM driver for accelerating cryptographic operations: those that provide cryptographic operations with a single API and those that provide them with multiple APIs. In this document, the former are referred to as single-part operations and the latter as multi-part operations.

APIs for multi-part operations are provided for symmetric key cryptography and hashes split on the Init-Update-Finish model, and APIs for single-part operations are provided for other ciphers.

3.1 Recovering after Unauthorized Access Detection

Unauthorized access monitoring by the RSIP is always enabled during execution of all cryptographic APIs. If tampering with cryptographic operations is detected while the driver is in use, the driver enters an infinite loop to stop operation.

Whether or not the operation of the RSIP PM driver is stopped in an infinite loop due to unauthorized access must be detected by the user application using a watchdog timer or other means.

If unauthorized access is detected by the user application, appropriate measures should be taken to satisfy the system security policy, such as log recording or restarting the system.

To recover from unauthorized access detection, close the RSIP PM driver once with `R_RSIP_Close()` and restart the RSIP with `R_RSIP_Open()`, or reset the device.

3.2 Avoiding RSIP Access Conflicts

The RSIP PM driver occupies the hardware resources of the RSIP while APIs are running. Even the APIs that provide multi-part operations continue to occupy the RSIP hardware resources until the series of multi-part operations is complete.

Therefore, keep in mind the following two points to avoid RSIP access conflicts when using the RSIP PM driver in a user application program:

- 1) While an RSIP PM driver API is being executed, other RSIP PM driver APIs must not be executed.
- 2) In the case of the APIs that provide multi-part operations, other RSIP PM driver APIs cannot execute until the series of operations (Init to Finish or Verify) currently being processed is complete.

If an RSIP PM driver API causes a hardware resource access conflict, the API returns `FSP_ERR_INVALID_STATE` or `FSP_ERR_CRYPTO_RSIP_RESOURCE_CONFLICT`.

3.3 BSP FIT Module Integration

The RSIP PM driver uses the following APIs of the BSP FIT module in R_RSIP_Open() and R_RSIP_Close() for module stop release/setting of the RSIP. Even if the BSP FIT module is not used in your program, incorporate the BSP FIT module into your program.

- R_BSP_RegisterProtectEnable()
- R_BSP_RegisterProtectDisable()

For details, refer to the application note “Board Support Module Firmware Integration Technology” (R01AN1685xJxxxx).

For R_RSIP_Open(), it is assumed that BSP FIT startup has completed before it is called. If BSP FIT startup is not used, call R_BSP_StartupOpen() before calling R_RSIP_Open(). R_BSP_StartupOpen() initializes the internal variables used in R_RegisterProtectEnable() and R_RegisterProtectDisable().

3.4 Single-Part and Multi-Part Operations

There are two types of APIs provided by the RSIP PM driver: those that provide operations with a single API and those that provide them with multiple APIs. In this document, the former are referred to as single-part operations and the latter as multi-part operations.

In multi-part operations, a single cryptographic operation is split into a sequence of separate steps (Init-Update-Final) in the form of APIs. This enables fine control over the configuration of the cryptographic operation and allows message data to be processed intermittently instead of all at once.

Refer to section 4.2 for the API specifications for each multi-part operation.

When an API for multi-part operations is called, the API to be called next is managed depending on the internal state of the RSIP PM driver. If the appropriate API is not called, an error is returned as the return value.

3.5 Initialization, Termination, and State Transitions of the Driver

The driver provides the following common function APIs for driver management operations:

No.	API	Description
1	R_RSIP_Open	Opens the RSIP PM driver. Initializes the RSIP and performs a self-test of the RSIP's fault detection and random number generator circuits.
2	R_RSIP_Close	Closes the RSIP PM driver.
3	R_RSIP_GetVersion	Gets the version number of the RSIP PM driver.

Applications using the driver must call R_RSIP_Open() first to initialize the RSIP and the driver. Also, when terminating use of the driver, R_RSIP_Close() must be called.

If problems occur while using the RSIP PM driver and there is a need to reset the RSIP PM driver and its control target (RSIP) before resuming processing, it is necessary to call R_RSIP_Open() after calling R_RSIP_Close().

R_RSIP_Open() performs a self-test to detect hardware failure of the RSIP and to check for abnormalities in the random number generator circuit. The self-test of the random number generator circuit implements the health test described in NIST SP800-90B on the data generated by the physical random number generator, evaluates the entropy, and generates a random number seed.

To get the version number of the RSIP PM driver, call R_RSIP_GetVersion().

The RSIP PM driver retains five internal states to manage the availability of the RSIP.

State Name	Description
Close	State in which the RSIP PM driver is unavailable before calling R_RSIP_Open(). Calling R_RSIP_Close() from any state will cause a transition to the Close state.
Main	State in which the RSIP PM driver is available after calling R_RSIP_Open(). After a multi-part or single-part operation is completed, the driver returns to this state.
Each Algorithm	The driver transitions to this state after calling an “Init” API for multi-part operations. In this state, the multi-part operations of the algorithm called by the “Init” API can be executed.
Firmware Update	The driver transitions to this state after calling R_RSIP_FWUP_StartUpdateFirmware(). In this state, R_RSIP_FWUP_MAC_Sign_Init/Update/Finish() can be called.
Stop	The driver transitions to this state after execution of R_RSIP_FWUP_MAC_Sign_Finish() finishes.

Figure 3.1 shows the state transition diagram of the RSIP PM driver.

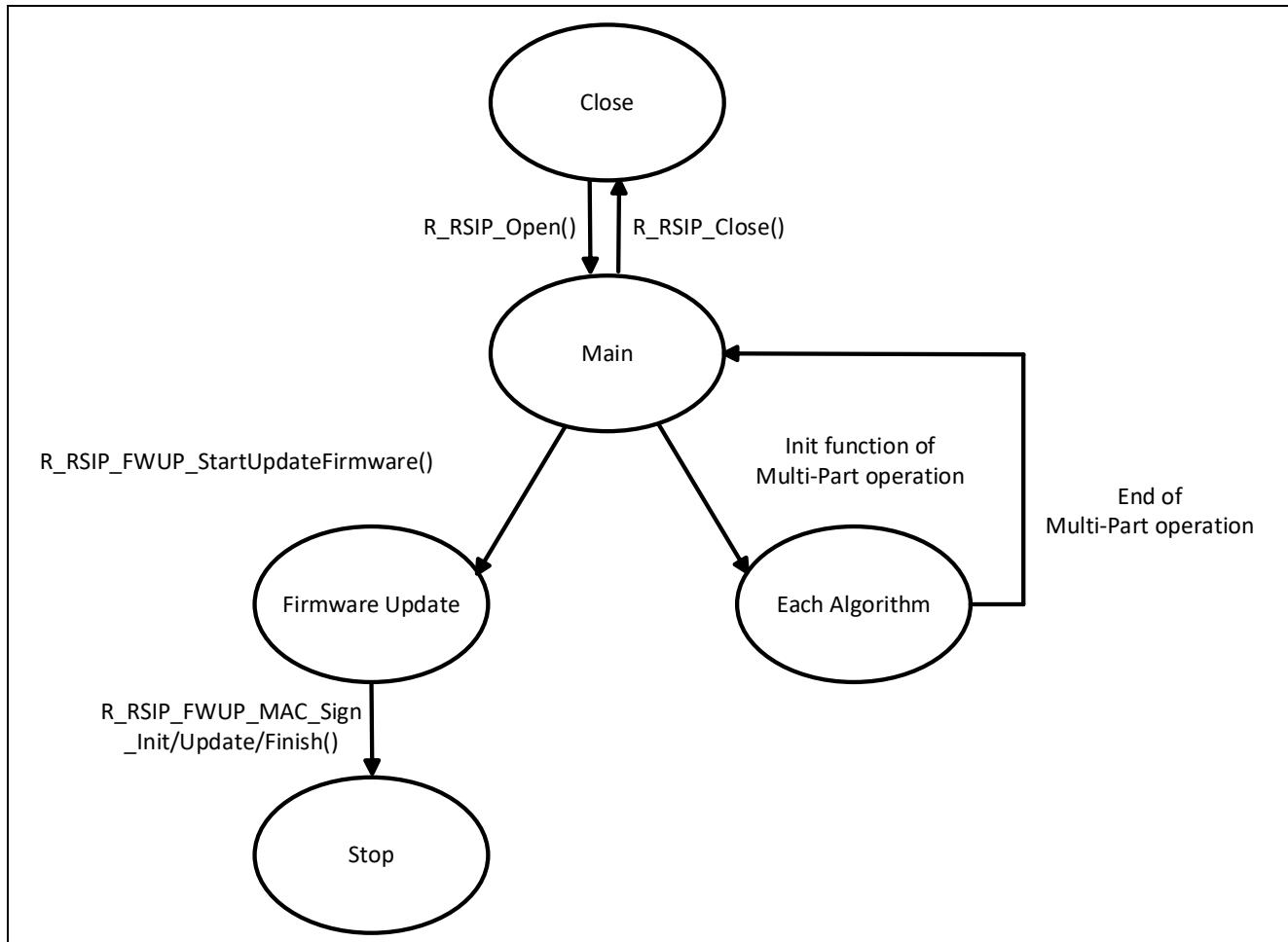


Figure 3.1 State Transition Diagram of the RSIP PM Driver

3.6 Key Management

The driver provides APIs for the following key management operations:

No.	API	Description
1	R_RSIP_EncryptedKeyWrap R_RSIP_InitialKeyWrap	Key updating and injection
2	R_RSIP_KeyGenerate R_RSIP_KeyPairGenerate	Key generation
3	R_RSIP_PublicKeyExport	Plaintext public key extraction

Figure 3.2 illustrates key handling in cryptographic operations performed by the RSIP PM driver.

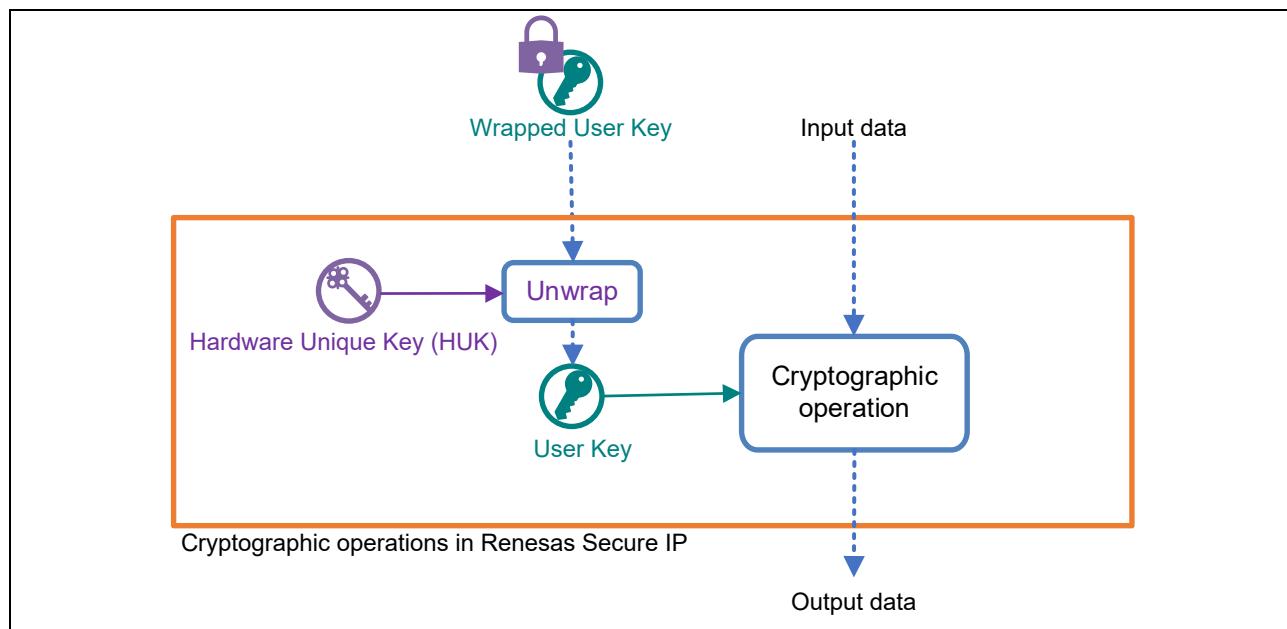


Figure 3.2 Key Handling in Cryptographic Operations by the RSIP PM Driver

The keys that are input and output in the cryptographic operations of the RSIP PM driver are opaque keys wrapped using a device-specific key called an HUK, which is accessible only by the RSIP. In the RSIP PM driver, this type of opaque key is called a wrapped key. Note here that the wrapped public key used in asymmetric key cryptography is in the form of a plaintext public key plus the key management information used in the RSIP PM driver.

The RSIP PM driver implements secure key management by wrapping user keys using the device-specific key. This provides key confidentiality and detection of tampering outside of the RSIP. The wrapping of the wrapped key can be unlocked only by the RSIP, and the unwrapped key exists only within the RSIP during cryptographic processing. Since the wrapped key has been wrapped using a device-specific key, it cannot be unwrapped using a different device-specific key, even if the wrapped key is copied from the nonvolatile memory of one device to another device.

3.6.1 Key Injection and Updating

Key injection and key updating provide a mechanism enabling secure delivery of user keys by converting them into wrapped keys wrapped using an HUK. Figure 3.3 shows the key injection and key updating operation sequence, including use of the Renesas Key Wrap Service.

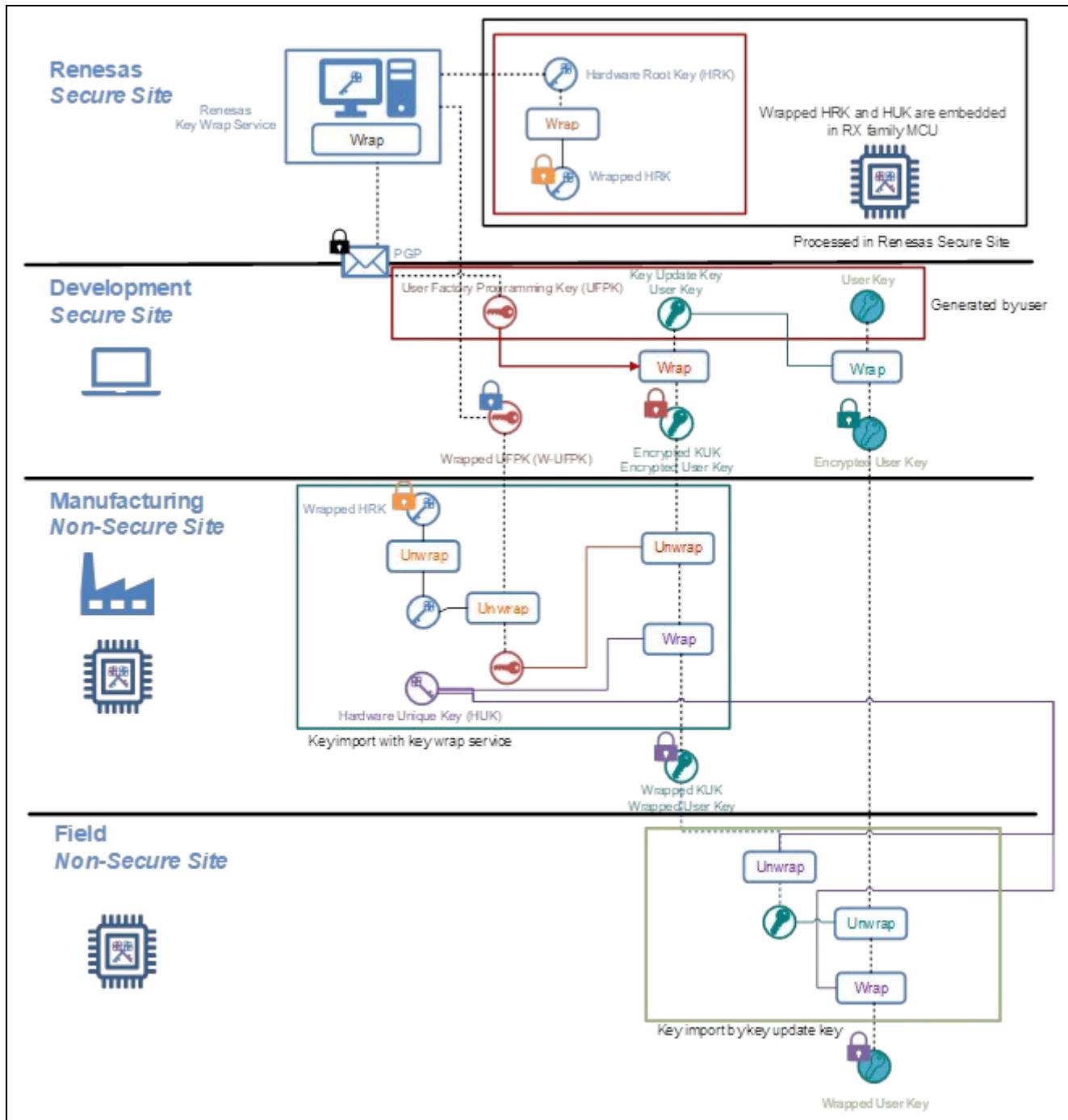


Figure 3.3 Key Injection and Key Updating Operation Sequence

When injecting keys, generate user keys, KUKs, and UFPKs at the customer's secure site (Figure 3.3 Development Secure Site), and wrap the user keys and KUKs with UFPKs. The UFPK used for wrapping should also be used to generate a W-UFPK using the Renesas Key Wrap Service. When updating keys, generate the user key to be updated at the customer's secure site ((Figure 3.3 Development Secure Site) and wrap the user key to be updated with KUK. The Security Key Management Tool can be used for key injection and key wrap for key renewal.

3.6.1.1 Key Wrapping Algorithm

Figure 3.4 shows the user key wrapping scheme used when a UFPK or KUK is used for wrapping during key injection and key updating. The first 128 bits of the UFPK or KUK are used as the CBC key and the trailing 128 bits as the CBC-MAC key for wrapping the user key or KUK. For the data formats of user keys and wrapped keys (encrypted user keys), refer to 5.3 User Key cryptographic format.

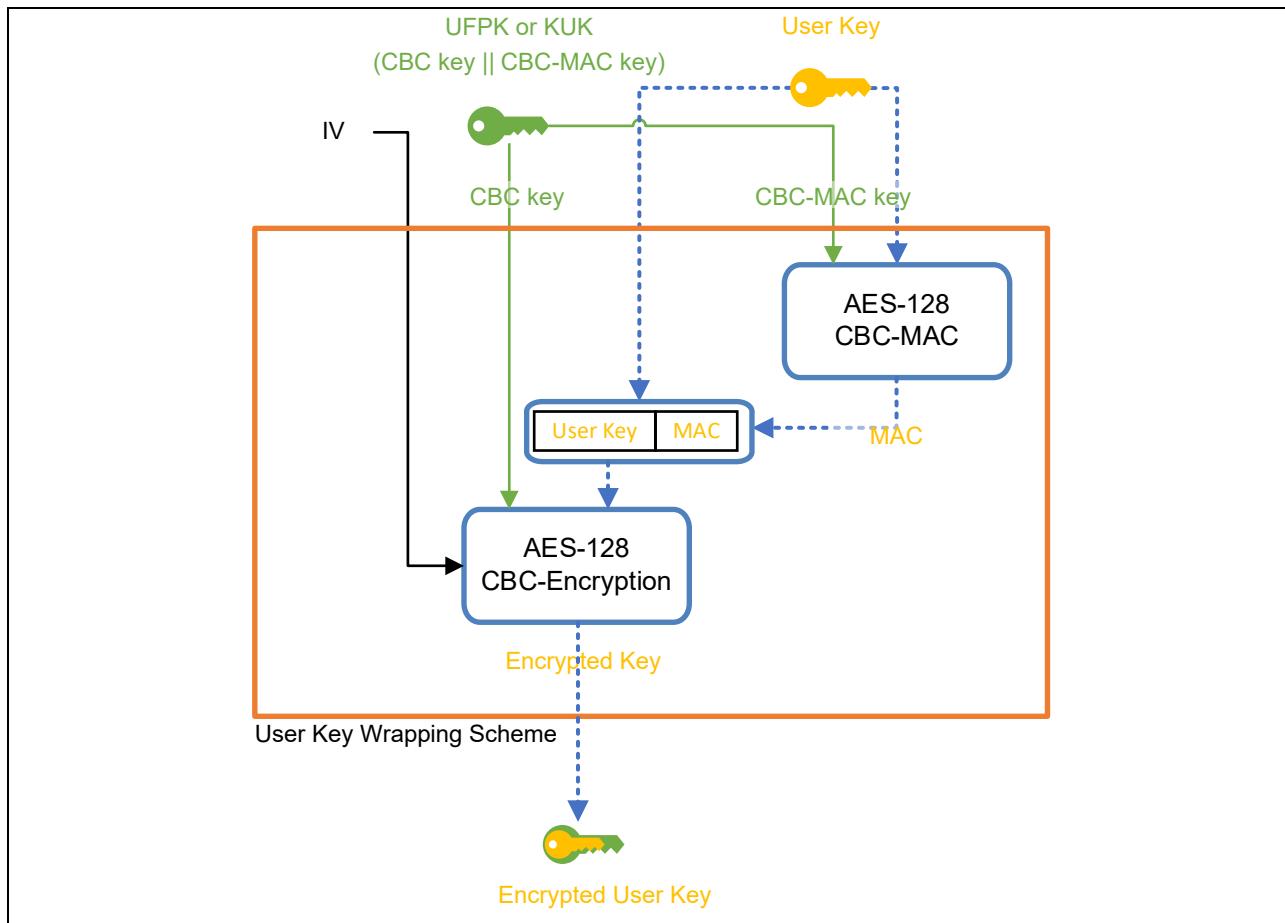


Figure 3.4 User Key Wrapping Scheme during Key Injection and Key Updating

The specific formula for wrapping the user key is as follows:

```
uint32_t user_key[len];
uint32_t MAC[4] = 0;
uint32_t iv[4] = IV;
for (i = 0; i < len; i += 4)
{
    MAC
        = AES_128_ENCRYPT(CBCMACkey[0: 3], xor_16byte(user_key[i: i+3], MAC[0: 3]));
    encrypted_key[i: i+3]
        = AES_128_ENCRYPT(CBCkey[0: 3], xor_16byte(user_key[i: i+3], iv[0: 3]));
    iv[0: 3] = encrypted_key [i: i+3];
}
encrypted_key[i: i+3] = AES_128_ENCRYPT(CBCkey[0: 3], xor_16byte(MAC[0: 3],
iv[0: 3]));
```

The functions used here mean the following processing:

- **AES_128_ENCRYPT(Key, Data):** Encryption of Data in AES128 ECB mode using the encryption Key

- xor_16byte(data1, data2): XOR operation of 16 bytes of data1 and data2

Also, one element of each array (CBCkey[], CBCMACkey[], MAC[], iv[], user_key[], encrypted_key[]) is 4 bytes in size.

The user key, including the KUK used for key updating, must be created by the user and injected into the device at the time of manufacture. For details of the procedure, refer to 5.1 Key Injection, and 5.2 Key Updating.

3.6.2 Key Generation

In key generation, the random number generation functionality of the RSIP is used to generate a key, which is then output in a wrapped key form that is usable by the RSIP PM driver.

To generate a wrapped key for use in symmetric key cryptography, call R_RSIP_KeyGenerate() with the type of key to be output.

To generate a pair of wrapped keys for use in asymmetric key cryptography, call R_RSIP_KeyPairGenerate() with the type of key to be output.

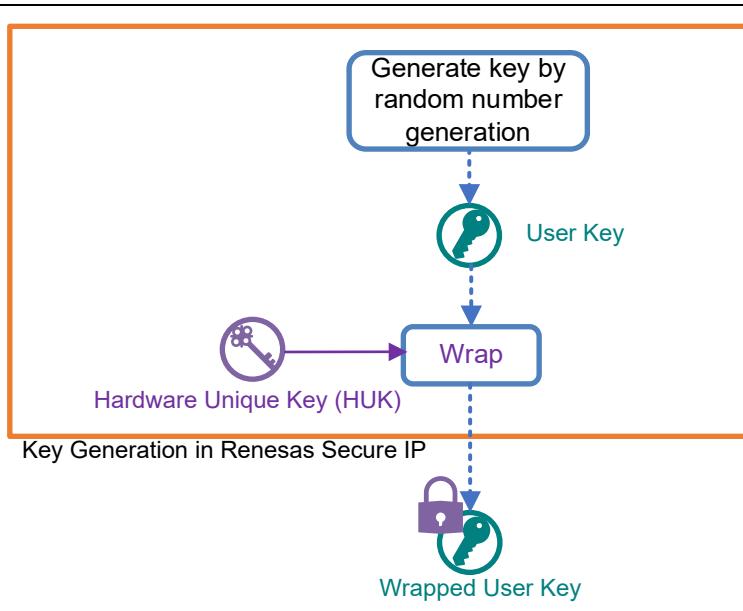


Figure 3.5 Key Generation Sequence

3.6.3 Plaintext Public Key Extraction

The wrapped public key is in the form of a plaintext public key plus the key management information used in the RSIP PM driver. To extract only the plaintext public key from a wrapped key, call R_RSIP_PublicKeyExport(). Refer to 4.2.2.5, R_RSIP_PublicKeyExport for the format of the extracted key.

3.7 Random Number Generation

The driver provides the following random number generation API.

No.	API	Description
1	R_RSIP_RandomNumberGenerate	Generates random numbers using CTR-DRBG as described in NIST SP800-90A.

3.8 Symmetric Key Cryptography

The driver provides APIs for the following symmetric cryptographic operations:

No.	API	Description
1	R_RSIP_AES_Cipher_Init R_RSIP_AES_Cipher_Update R_RSIP_AES_Cipher_Finish	Symmetric key cryptography AES 128-/256-bit ECB, CBC, CTR encryption and decryption
2	R_RSIP_AES_AEAD_Init R_RSIP_AES_AEAD_LengthsSet R_RSIP_AES_AEAD_AADUpdate R_RSIP_AES_AEAD_Update R_RSIP_AES_AEAD_Finish R_RSIP_AES_AEAD_Verify	Authenticated encryption with associated data (AEAD) AES 128-/256-bit GCM, CCM encryption and decryption
3	R_RSIP_AES_MAC_Init R_RSIP_AES_MAC_Update R_RSIP_AES_MAC_SignFinish R_RSIP_AES_MAC_VerifyFinish	Message authentication codes (MAC) AES-CMAC 128-/256-bit MAC generation and verification

A set of API functions that enable multi-part operations is provided for each type of symmetric cryptographic operation. For details on multi-part operations, refer to 3.4, Single-Part and Multi-Part Operations.

3.8.1 Symmetric Key Cryptography

The symmetric cryptographic operation is performed as follows:

Call R_RSIP_AES_Cipher_Init() to specify the encryption mode, required key and initialization vector.

Call the R_RSIP_AES_Cipher_Update() function for the chunks of data comprising the plaintext or ciphertext message in consecutive block units.

To complete the encryption operation, call R_RSIP_AES_Cipher_Finish().

3.8.2 Authenticated Encryption with Associated Data (AEAD)

The authenticated encryption with associated data is performed as follows:

Call R_RSIP_AES_AEAD_Init() to specify the encryption mode, required key and initialization vector.

When the encryption mode is CCM, call R_RSIP_AES_AEAD_LengthsSet() to specify the size of the input data.

Call R_RSIP_AES_AEAD_AADUpdate() to specify AAD.

Call the R_RSIP_AES_AEAD_Update() function for the chunks of data comprising the message in consecutive block units.

To complete the encryption operation and compute the authentication tag, call R_RSIP_AES_AEAD_Finish().

To complete the decryption operation, compute the authentication tag, and verify it against a reference value, call R_RSIP_AES_AEAD_Verify().

3.8.3 Message Authentication Code (MAC)

The message authentication code processing is performed as follows:

Call R_RSIP_AES_MAC_Init() to specify the MAC calculation mode and required key.

Call the R_RSIP_AES_MAC_Update() function for the consecutive chunks of data comprising the message.

In the case of a MAC generation operation, call R_RSIP_AES_MAC_SignFinish() to get the MAC data and complete the cryptographic operation.

To complete MAC verification for the message, call R_RSIP_AES_MAC_VerifyFinish().

3.9 Asymmetric Key Cryptography

The driver provides APIs for the following asymmetric cryptographic operations:

No.	API	Description
1	R_RSIP_ECDSA_Sign R_RSIP_ECDSA_Verify	Generates and verifies ECDSA signatures.

APIs for asymmetric key cryptography provide only single-part operations.

3.10 Hash Functions

The driver provides APIs for the following hash calculation:

No.	API	Description
1	R_RSIP_SHA_Compute R_RSIP_SHA_Init R_RSIP_SHA_Update R_RSIP_SHA_Finish R_RSIP_SHA_Suspend R_RSIP_SHA_Resume	Message digests SHA-224/256
2	R_RSIP_HMAC_Compute R_RSIP_HMAC_Verify R_RSIP_HMAC_Init R_RSIP_HMAC_Update R_RSIP_HMAC_SignFinish R_RSIP_HMAC_VerifyFinish R_RSIP_HMAC_Suspend R_RSIP_HMAC_Resume	Message authentication codes (HMAC) HMAC-SHA224/256

A set of API functions that enable multi-part operations is provided for each type of hash calculation. For details on multi-part operations, refer to 3.4, Single-Part and Multi-Part Operations

3.10.1 Message Digest

The message digest generation is performed as follows:

When multi-part operation is used, call R_RSIP_SHA_Compute().

When single-part operation is used, call R_RSIP_SHA_Init() to specify the hash calculation mode,

call R_RSIP_SHA_Update() for the consecutive chunks of data comprising the message, and

call R_RSIP_SHA_Finish() to calculate the digest of the message.

After calling R_RSIP_SHA_Update(), the multi-part operations for hash calculation can be suspended by calling R_RSIP_SHA_Suspend(). To resume hash calculation, call R_RSIP_SHA_Resume() and then call R_RSIP_SHA_Update() or R_RSIP_SHA_Finish() again to perform hash calculation.

R_RSIP_SHA_Finish() can be called after R_RSIP_SHA_Suspend() without calling R_RSIP_SHA_Resume() to retrieve calculation results of data while the hash calculation is in progress.

3.10.2 Message Authentication Code (HMAC)

The message authentication code processing is performed as follows:

When multi-part operation is used, call R_RSIP_HMAC_Compute() to generate HMAC or call R_RSIP_HMAC_Verify() to verify HMAC.

When single-part operation is used, call R_RSIP_HMAC_Init() to specify the required key, call R_RSIP_AES_HMAC_Update() for the consecutive chunks of data comprising the message, and call R_RSIP_HMAC_SignFinish() to complete HMAC generation for the message, or call R_RSIP_HMAC_VerifyFinish() to verify the HMAC of a message.

3.11 Key Wrap

The driver provides APIs for the following key wrapping operations:

No.	API	Description
1	R_RSIP_RFC3394_KeyWrap	Wraps a key with an RFC3394-compliant algorithm.
2	R_RSIP_RFC3394_KeyUnwrap	Unwraps a key with an RFC3394-compliant algorithm.

To wrap a key, call R_RSIP_RFC3394_KeyWrap() to specify the key to be used for wrapping and the key to be wrapped to obtain the wrapped key.

To unwrap a key, call R_RSIP_RFC3394_KeyWrap() to obtain the wrapped key of the unwrapped key, specifying the key to use for unwrapping, the key to wrap, and the type of wrapped key to output.

3.12 Key Exchange/Key Derivation

The driver provides APIs for the following key derivation and key derivation operations:

No.	API	Description
1	R_RSIP_PKI_ECDSA_CertVerify R_RSIP_PKI_CertKeyImport R_RSIP_PKI_VerifiedCertInfoExport R_RSIP_PKI_VerifiedCertInfoImport R_RSIP_ECDH_KeyAgree R_RSIP_ECDH_PlainKeyAgree	Key Exchange KDF-SHA256
2	R_RSIP_KDF_SHA_Init R_RSIP_KDF_SHA_ECDHSecretUpdate R_RSIP_KDF_SHA_Update R_RSIP_KDF_SHA_Finish R_RSIP_KDF_SHA_Suspend R_RSIP_KDF_SHA_Resume R_RSIP_KDF_DKMConcatenate R_RSIP_KDF_DerivedKeyImport R_RSIP_KDF_DerivedIVWrap	Key Derivation KDF-SHA256

A set of API functions is provided for key derivation and key derivation operations based on Public Key Infrastructure (PKI). First, the APIs for key exchange generates wrapped secret. And using the wrapped secret, the APIs for key derivation generates encryption key and initial vector.

3.12.1 Receiving Public Key

There are two methods to receive a public key from the peer of the key exchange, one is a method to receive a public key which is signed by Certification Authority (CA) or other (verified public key), the another is a method to receive unverified public key. Both methods can be performed with the functions.

3.12.1.1 To Handle Verified Public Key (Extracting Public Key from Certificate)

The sequence of the method to receive verified public key is described in Figure 3.6.

1. The peer of key exchange generates a key pair and sends a Certificate Signing Request (CSR) with the public key to CA. Then, CA issues the certificate.
2. The product with RSIP receives the wrapped public key of the CA, certificate and signature which are issued by the CA.
3. Call R_RSIP_PKI_ECDSA_CertVerify() with the wrapped public key of CA, certificate, and hash value of the certificate, which can be computed with R_RSIP_SHA_Compute(), as input to verify the signature.
Note: RSIP PM driver stores only the latest verified certificate information. When multiple certificates are used in the product, call R_RSIP_PKI_VerifiedCertInfoExport() to export the verified certificate information and call R_RSIP_PKI_VerifiedCertInfoImport() to import the exported verified certificate information.
4. Call R_RSIP_PKI_CertKeyImport() with the certificate as input to verify the certificate and generate wrapped public key.

Note: The wrapped public key obtained in this sequence can be used with asymmetric key cryptography APIs described in 3.9.

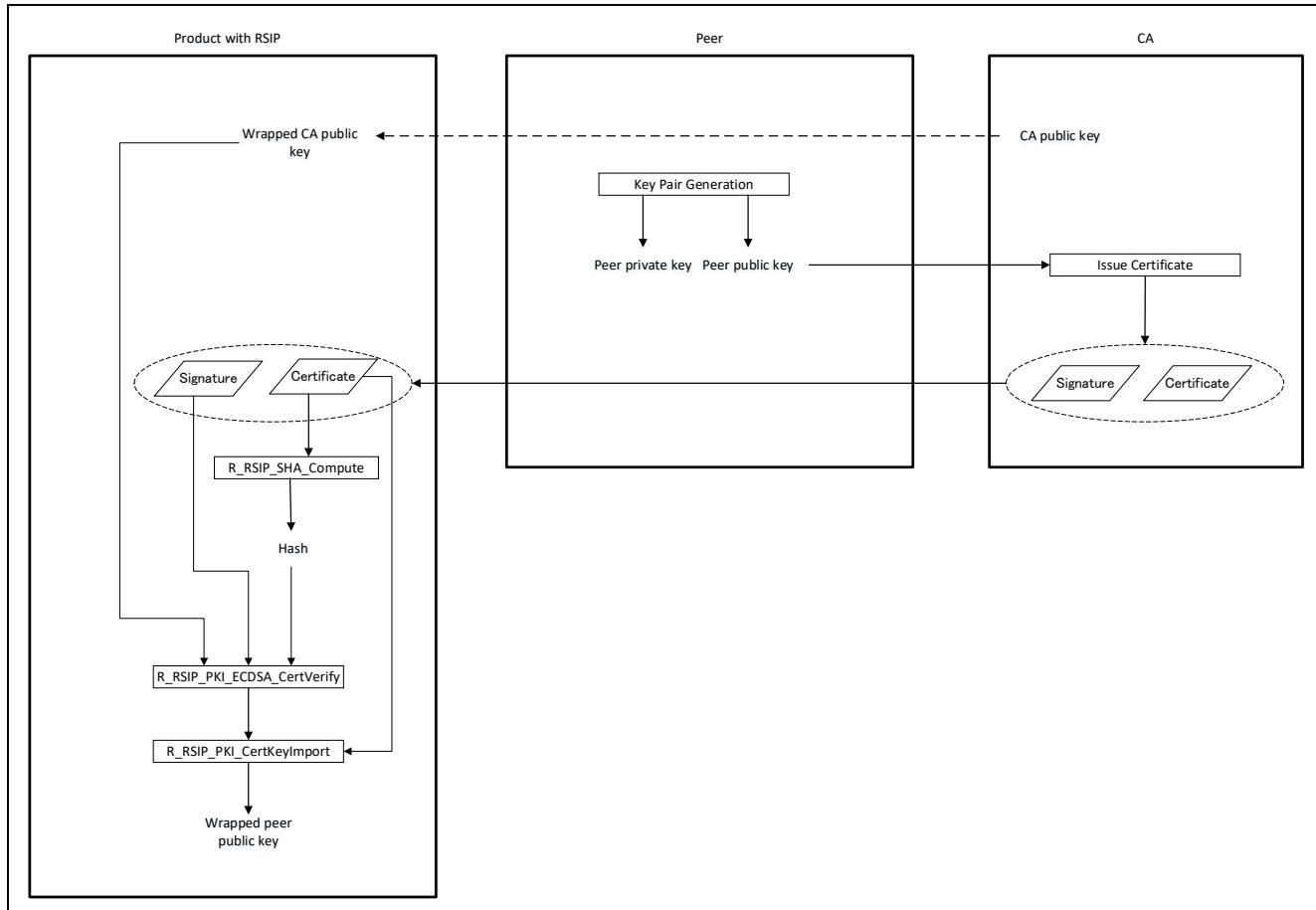


Figure 3.6 Sequence to Receive Verified Public Key

3.12.1.2 To Handle Unverified Public Key

To handle an unverified public key, it is not necessary to generate a wrapped public key. Refer 3.12.2.2 to generate a wrapped secret with an unverified public key.

Note that there is a risk of man-in-the-middle attacks, since peer public key is unverified. Therefore, it is recommended to use the verified public key as far as possible.

3.12.2 Generate Wrapped Secret

3.12.2.1 Generate Wrapped Secret with Verified Public Key

The sequence of generating a wrapped secret with verified wrapped public key is described in Figure 3.7.

1. Call R_RSIP_ECDH_KeyAgree() with the wrapped public key, which is generated in 3.12.1, and a wrapped private key, which is generated with R_RSIP_KeyPairGenerate(), as input to generate wrapped secret.
2. Call R_RSIP_PublicKeyExport() with the wrapped public key, which is generated with R_RSIP_KeyPairGenerate() in the above procedure, as input to generate a plain public key. Send the plain public key to the peer to generate the secret.

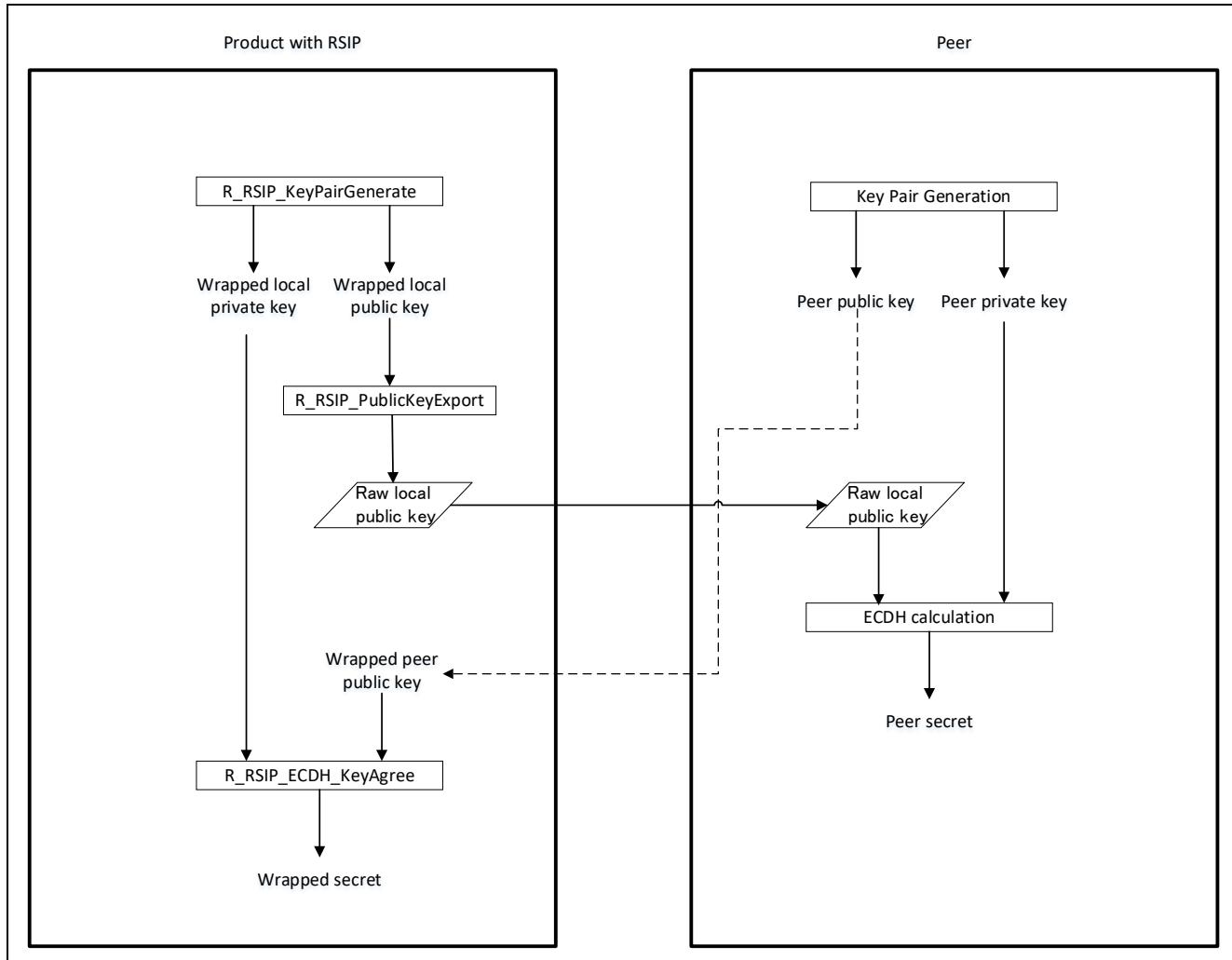


Figure 3.7 Sequence to Generate Wrapped Secret with Verified Public Key

3.12.2 Generate Wrapped Secret with Unverified Public Key

The sequence of generating a wrapped secret with plain public key is described in Figure 3.8.

1. Call **R_RSIP_ECDH_PlainKeyAgree()** with the plain public key, which is received from the peer, and a wrapped private key, which is generated with **R_RSIP_KeyPairGenerate()**, as input to generate wrapped secret.
2. Call **R_RSIP_PublicKeyExport()** with the wrapped public key, which is generated with **R_RSIP_KeyPairGenerate()** in the above procedure, as input to generate a plain public key. Send the plain public key to the peer to generate the secret.

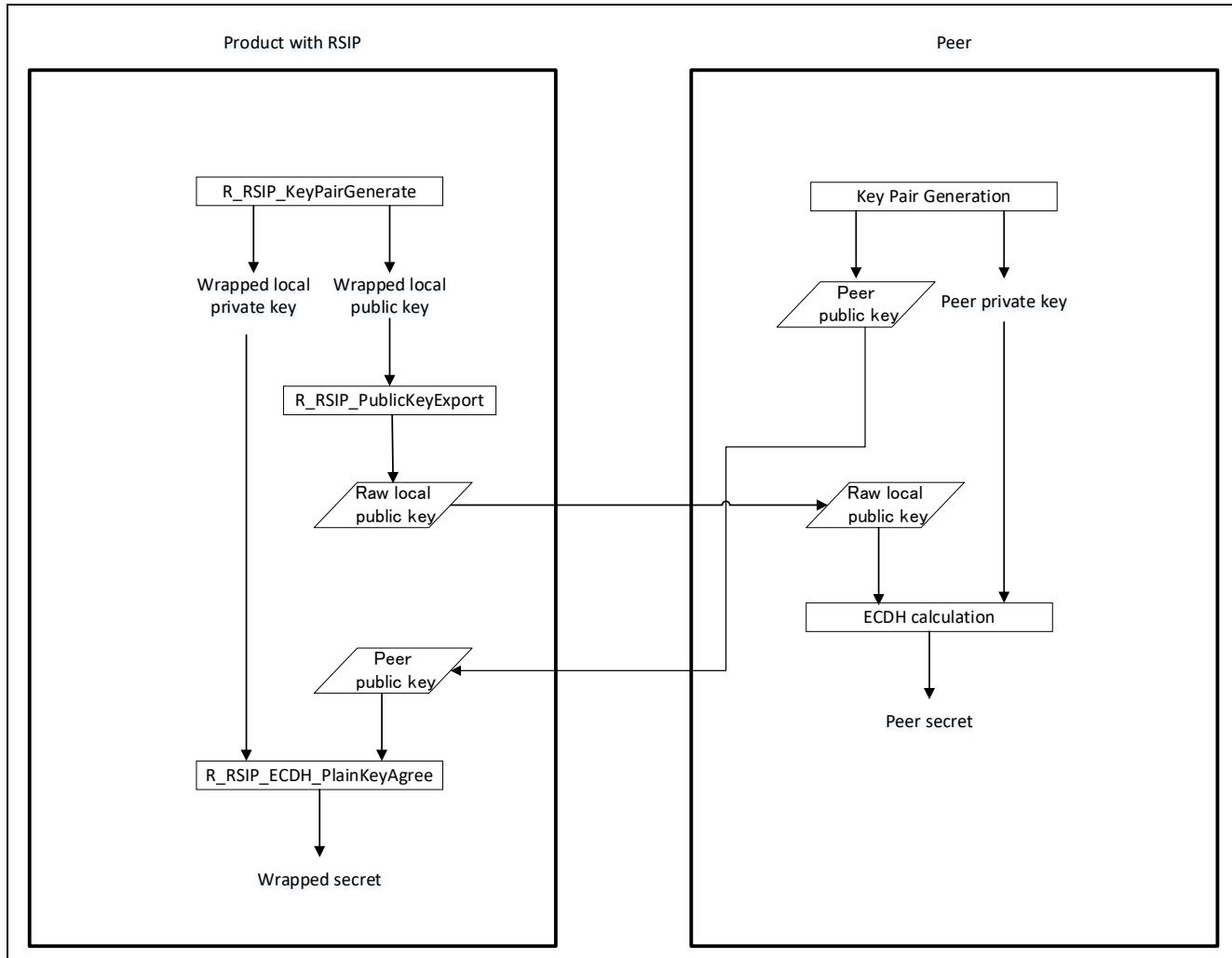


Figure 3.8 Sequence to Generate Wrapped Secret with Unverified Public Key

3.12.3 Key Derivation

Call R_RSIP_KDF_SHA_Init/ECDHSecretUpdate/Update/Finish() with the wrapped secret, which is generated with R_RSIP_ECDH_KeyAgree() in 3.12.2.1 or R_RSIP_ECDH_PlainKeyAgree() in 3.12.2.2, as input to generate Derived Keying Material (DKM). Call R_RSIP_KDF_DKMConcatenate() to concatenate DKM, call R_RSIP_KDF_DerivedKeyImport() to generate a wrapped key from the DKM, and call R_RSIP_KDF_DerivedIVWrap() to generate an initial vector from the DKM.

After calling R_RSIP_KDF_SHA_ECDHSecretUpdate/Update(), the KDF calculation can be suspended with calling R_RSIP_KDF_SHA_Suspend(). To resume the KDF calculation, call R_RSIP_KDF_SHA_Resume(), and call R_RSIP_KDF_SHA_Update() or R_RSIP_KDF_SHA_Finish() to continue the KDF calculation.

3.13 Firmware Update/Secure Boot

The driver provides APIs for the following firmware update and secure boot operations:

No.	API	Description
1	R_RSIP_FWUP_StartUpdateFirmware R_RSIP_FWUP_MAC_Sign_Init R_RSIP_FWUP_MAC_Sign_Update R_RSIP_FWUP_MAC_Sign_Finish	Decrypts encrypted firmware, performs MAC verification, and generates a MAC for decrypted plaintext firmware.
2	R_RSIP_SB_MAC_Verify_Init R_RSIP_SB_MAC_Verify_Update R_RSIP_SB_MAC_Verify_Finish	Performs verification of the MAC generated from the plaintext firmware decrypted by R_RSIP_FWUP_MAC_Sign_Update/Finish.

The firmware update functionality provides a mechanism for securely delivering an encrypted program and the key used to encrypt it. Firmware updates can be achieved in conjunction with secure boot. Figure 3.9 shows the program encryption, encryption key injection, and MAC verification operation sequence, including use of the Renesas Key Wrap Service.

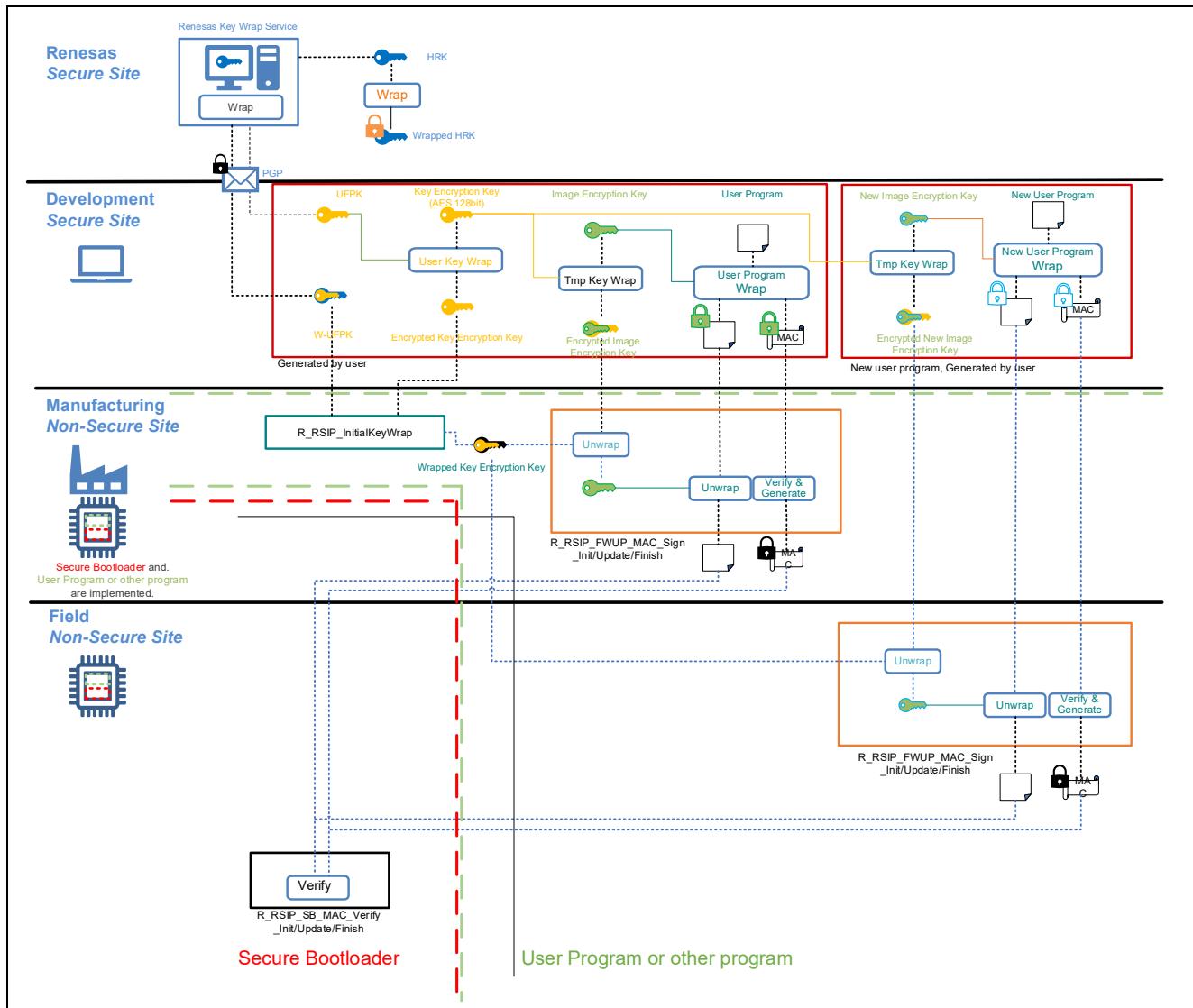


Figure 3.9 Firmware Update Sequence

At your secure site (Figure 3.9 Development Secure Site), generate the UFPK, the key used for encryption of the user program (Image Encryption Key), and the Key Encryption Key, and encrypt the user program and the encrypted program MAC. Please generate the key (Image Encryption Key) and Key Encryption Key for encryption of the user program. Also, wrap the Image Encryption Key with the Key Encryption Key. See 3.13.3 Encrypting the User Program for the algorithm of encryption of user program, MAC generation, and wrapping of the key used in encryption. The Security Key Management Tool can be used to encrypt the user program, generate the MAC, and wrap the key used for encryption.

3.13.1 Secure Boot

Secure boot refers to a functionality for detecting tampering with the user program. Before executing the user program after a reset, execute the secure boot program and verify the program to be executed after secure boot.

`R_RSIP_SB_MAC_VerifyInit/Update/Finish()` can be used to implement secure boot. Run `R_RSIP_SB_MAC_VerifyInit/Update/Finish()` with the plaintext firmware and MAC value output by `R_RSIP_FWUP_MAC_Sign_Update/Finish()` as input.

3.13.2 Firmware Update

Firmware update is a functionality that updates the firmware currently running to add new functionalities or to repair defects.

`R_RSIP_FWUP_MAC_Sign_Init/Update/Finish()`, which decrypts an encrypted program and performs MAC verification, can be used for firmware update implementation. When MAC verification is successful, `R_RSIP_FWUP_MAC_Sign_Finish()` newly generates a new MAC for plaintext firmware with a key newly linked to the HUK.

`R_RSIP_FWUP_MAC_Sign_Init/Update/Finish()` can be run after first calling `R_RSIP_StartUpdateFirmware()` to put the RSIP into the Firmware Update state.

3.13.3 Encrypting the User Program

Figure 3.10 shows the method for encrypting the user program.

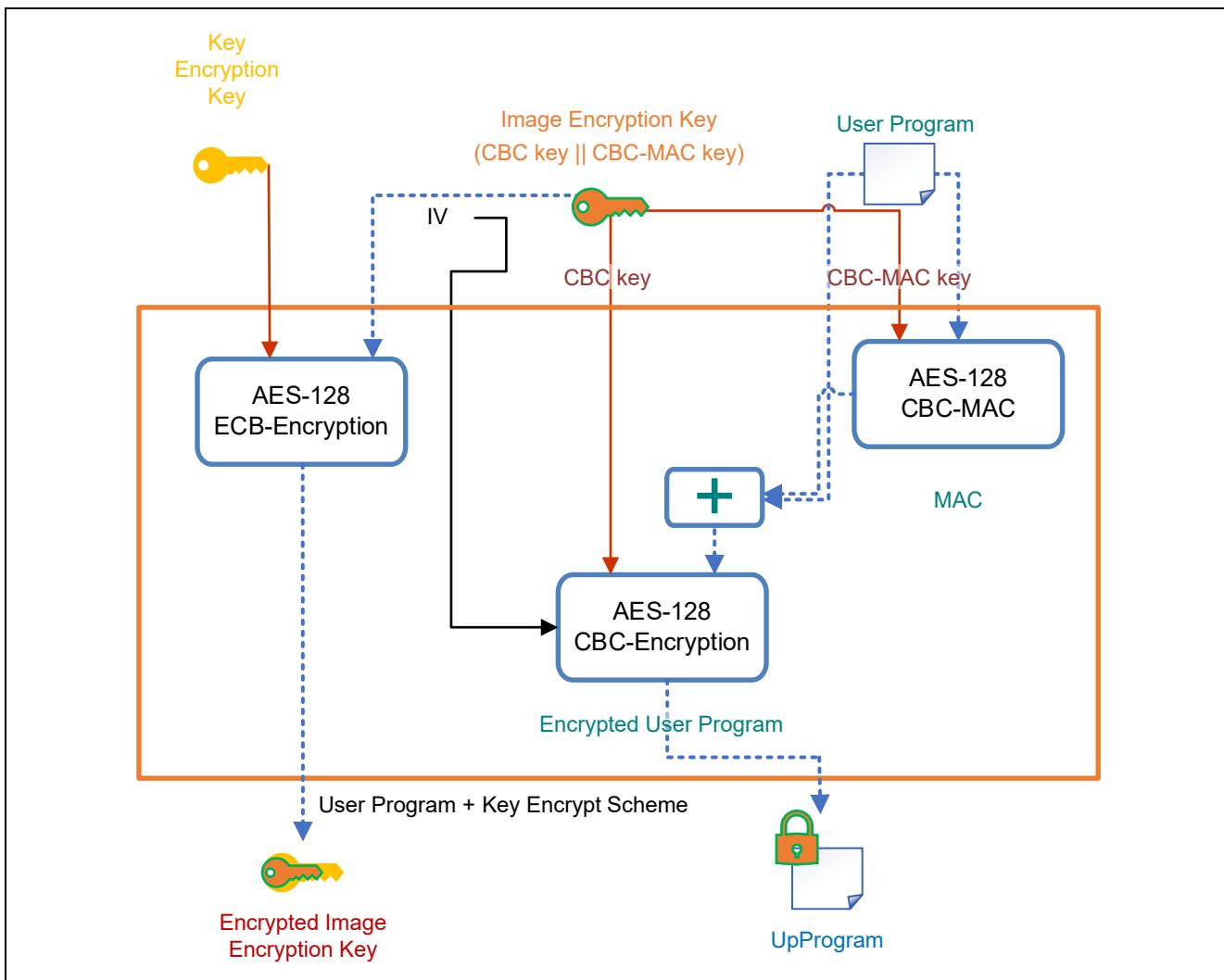


Figure 3.10 Firmware and Session Key Encryption Method

First, a key for encrypting the user program (image encryption key) and a user program key (key encryption key) for encrypting the image encryption key are prepared. Then the image encryption key is used to generate a MAC and encrypt the user program (UpProgram). The key encryption key is used to wrap the image encryption key, and an encrypted image encryption key (session key) is generated.

4. API Functions

4.1 List of APIs

The RSIP PM driver implements the following APIs:

1. Common function APIs
2. Key Management APIs
3. Random number generation API
4. AES encryption/decryption APIs
5. ECC signature generation/verification APIs
6. HASH calculation APIs
7. KDF (Key Derivation Function) APIs
8. Key wrap APIs
9. Firmware update/secure boot APIs

The APIs implemented in the RSIP PM driver are summarized in the tables below. “XXX” in the name of an API represents either the bit length or the SHA mode.

Table 4-1 Common Function APIs

API	Description
R_RSIP_Open	Exits RSIP from the module stop state and opens the RSIP PM driver.
R_RSIP_Close	Place RSIP in module stop state and close the RSIP PM driver.
R_RSIP_GetVersion	Outputs the version of the RSIP PM driver.

Table 4-2 Key Management APIs

API	Description
R_RSIP_InitialKeyWrap	Used during key injection. Generates the binary value of the Wrapped Key from the UFPK wrapped user key.
R_RSIP_EncryptedKeyWrap	Used for key update. Generates a Wrapped Key from a KUK-wrapped user key.
R_RSIP_KeyGenerate	Generates keys for symmetric key cryptography.
R_RSIP_KeyPairGenerate	Generates keys for asymmetric key cryptography.
R_RSIP_PublicKeyExport	Exports a plain-text public key from the Wrapped Key of the public key.

Table 4-3 Random Number Generation API

API	Description
R_RSIP_RandomNumberGenerate	Generates random number.

Table 4-4 AES Encryption/Decryption APIs

API	Description
R_RSIP_AES_Cipher_Init	Prepares to perform AES cryptographic operations.
R_RSIP_AES_Cipher_Update	Performs AES cryptographic operations.
R_RSIP_AES_Cipher_Finish	Terminates AES cryptographic operation.
R_RSIP_AES_AEAD_Init	Prepares to perform the AES AEAD operation.
R_RSIP_AES_AEAD_LengthsSet	Specifies the size of the data used in the AES AEAD operation.
R_RSIP_AES_AEAD_AADUpdate	Specifies additional authentication data to be used in the AES AEAD operation.
R_RSIP_AES_AEAD_Update	Performs AES AEAD operation.
R_RSIP_AES_AEAD_Finish	Terminates the encryption operation of the AES AEAD.
R_RSIP_AES_AEAD_Verify	Terminates the decryption operation of the AES AEAD.
R_RSIP_AES_MAC_Init	Prepares to perform the AES MAC operation.
R_RSIP_AES_MAC_Update	Performs AES MAC operation.
R_RSIP_AES_MAC_SignFinish	Terminates the generation operation of the AES MAC.
R_RSIP_AES_MAC_VerifyFinish	Terminates the verification operation of the AES MAC.

Table 4-5 ECC Signature Generation/Verification APIs

API	Description
R_RSIP_ECDSA_Sign	Performs ECDSA signature generation operations.
R_RSIP_ECDSA_Verify	Performs ECDSA signature verification operations.
R_RSIP_PKI_CertVerify	Verifies a public key certificate with ECDSA.
R_RSIP_PKI_CertKeyImport	Wraps public key in the verified public key certificate.
R_RSIP_PKI_VerifiedCertInfoExport	Exports verified certificate information.
R_RSIP_PKI_VerifiedCertInfoImport	Imports verified certificate information.
R_RSIP_ECDH_KeyAgree	Computes ECDH secret.
R_RSIP_ECDH_PlainKeyAgree	Computes ECDH secret.

Table 4-6 HASH Calculation APIs

API	Description
R_RSIP_SHA_Compute	Performs SHA calculation operations.
R_RSIP_SHA_Init	Prepares to perform SHA calculation operations.
R_RSIP_SHA_Update	Performs SHA calculation operations.
R_RSIP_SHA_Finish	Terminates SHA calculation operations.
R_RSIP_SHA_Suspend	Suspends SHA calculation operations.
R_RSIP_SHA_Resume	Resumes SHA calculation operations.
R_RSIP_HMAC_Compute	Performs HMAC generation operations.
R_RSIP_HMAC_Verify	Performs HMAC verification operations.
R_RSIP_HMAC_Init	Prepares to perform the HMAC operation.
R_RSIP_HMAC_Update	Performs HMAC operation.
R_RSIP_HMAC_SignFinish	Terminates the generation operation of the HMAC.
R_RSIP_HMAC_VerifyFinish	Terminates the verification operation of the HMAC.
R_RSIP_HMAC_Suspend	Suspends HMAC calculation operations.
R_RSIP_HMAC_Resume	Resumes HMAC calculation operations.

Table 4-7 KDF APIs

API	Description
R_RSIP_KDF_SHA_Init	Prepares to perform SHA calculation operations for KDF.
R_RSIP_KDF_SHA_ECDHSecretUpdate	Inputs a wrapped ECDH secret for KDF.
R_RSIP_KDF_SHA_Update	Performs SHA calculation operations for KDF.
R_RSIP_KDF_SHA_Finish	Terminates SHA calculation operations for KDF.
R_RSIP_KDF_SHA_Suspend	Suspends SHA calculation operations for KDF.
R_RSIP_KDF_SHA_Resume	Resumes SHA calculation operations for KDF.
R_RSIP_KDF_DKMConcatenate	Concatenates two wrapped DKMs.
R_RSIP_KDF_DerivedKeyImport	Outputs a wrapped key from KDF output.
R_RSIP_KDF_DerivedIVWrap	Outputs a wrapped initial vector from KDF output.

Table 4-8 Key Exchange APIs

API	Description
R_RSIP_RFC3394_KeyWrap	Wraps the key with an RFC3394 compliant algorithm.
R_RSIP_RFC3394_KeyUnwrap	Unwraps the key with an RFC3394 compliant algorithm.

Table 4-9 Firmware Update/Secure Boot APIs

API	Description
R_RSIP_FWUP_StartUpdateFirmware	Transitions to firmware update mode.
R_RSIP_FWUP_MAC_Sign_Init	Decrypt encrypted firmware and prepare for MAC generation.
R_RSIP_FWUP_MAC_Sign_Update	Decrypts encrypted firmware and outputs plain-text firmware.
R_RSIP_FWUP_MAC_Sign_Finish	Decrypts encrypted firmware and verifies the MAC to generate a plaintext firmware MAC.
R_RSIP_SB_MAC_Verify_Init	Prepare to verify the MAC of the plaintext firmware.
R_RSIP_SB_MAC_Verify_Update	Performs MAC operations on plaintext firmware.
R_RSIP_SB_MAC_Verify_Finish	Performs MAC verification of plaintext firmware.

4.2 Detailed Descriptions of API Functions

4.2.1 Common Functions

4.2.1.1 R_RSIP_Open

Format

```
#include "r_rsip_protected_rx_if.h"  
fsp_err_t R_RSIP_Open(  
    rsip_ctrl_t * const p_ctrl,  
    rsip_cfg_t const * const p_cfg  
)
```

Parameters

p_ctrl	Input/output	RSIP PM driver management structure
p_cfg	Input	Configuration structure

Return Values

FSP_SUCCESS	Normal termination
FSP_ERR_ASSERTION	Null pointer argument
FSP_ERR_ALREADY_OPEN	Driver already open
FSP_ERR_CRYPTO_RSIP_FAIL	Invalid input parameter
FSP_ERR_CRYPTO_RSIP_RESOURCE_CONFLICT	Occurrence of resource conflict because a hardware resource needed by the processing routine was in use by another processing routine
FSP_ERR_CRYPTO_RSIP_FATAL	Software corruption or hardware failure detected

Description

This API releases the RSIP from the module stop state, initializes the RSIP, and opens the RSIP PM driver.

For p_ctrl, input the management structure used commonly by the RSIP PM driver. p_ctrl should be held until R_RSIP_Close() is called.

For p_cfg, input the pin configuration structure of the device. It is not used in the RX RSIP PM driver, but a null cannot be specified. Input a pointer to any rsip_cfg_t structure.

Reentrant

Not supported.

4.2.1.2 R_RSIP_Close

Format

```
#include "r_rsip_protected_rx_if.h"  
fsp_err_t R_RSIP_Close (  
    rsip_ctrl_t * const p_ctrl  
)
```

Parameters

p_ctrl	Input/output	RSIP PM driver management structure
--------	--------------	-------------------------------------

Return Values

FSP_SUCCESS	Normal termination
FSP_ERR_ASSERTION	Null pointer argument
FSP_ERR_NOT_OPEN	RSIP PM driver not open
FSP_ERR_CRYPTO_RSIP_FATAL	Software corruption detected

Description

This API puts the RSIP into the module stop state and closes the RSIP PM driver.

For p_ctrl, specify the RSIP PM driver management structure used in the R_RSIP_Open() function.

Reentrant

Not supported.

4.2.1.3 R_RSIP_GetVersion

Format

```
#include "r_rsip_protected_rx_if.h"
uint32_t R_RSIP_GetVersion(void)
```

Parameters

None

Return Values

Upper 2 bytes:	Major version (decimal notation)
Lower 2 bytes:	Minor version (decimal notation)

Description

This API can be used to obtain the RSIP PM driver version.

Reentrant

Not supported.

4.2.2 Key Management

4.2.2.1 R_RSIP_InitialKeyWrap

Format

```
#include "r_rsip_protected_rx_if.h"
fsp_err_t R_RSIP_InitialKeyWrap(
    rsip_ctrl_t * const p_ctrl,
    void const * const p_wrapped_user_factory_programming_key,
    void const * const p_initial_vector,
    void const * const p_encrypted_key,
    rsip_wrapped_key_t * const p_wrapped_key
)
```

Parameters

p_ctrl	Input/output	RSIP	PM	driver	management structure
p_wrapped_user_factory_programming_key	Input				W-UFPK
p_initial_vector	Input				Initialization vector (16 bytes)
p_encrypted_key	Input				Encrypted key
p_wrapped_key	Input/output				Wrapped key (The wrapped key data size which is defined for each key type in Table 2-10 is output to p_value field)

Return Values

FSP_SUCCESS	Normal termination
FSP_ERR_ASSERTION	Null pointer argument
FSP_ERR_CRYPTO_RSIP_KEY_SET_FAIL	Occurrence of error in verification of the key specified in p_encrypted_key
FSP_ERR_CRYPTO_RSIP_FAIL	Invalid input parameter
FSP_ERR_CRYPTO_RSIP_RESOURCE_CONFLICT	Occurrence of resource conflict because a hardware resource needed by the processing routine was in use by another processing routine
FSP_ERR_CRYPTO_RSIP_FATAL	Software corruption or hardware failure detected

Description

This API generates binary data of the wrapped user key from the UFPK-wrapped user key. For p_wrapped_user_factory_programming_key, specify the W-UFPK of the UFPK used in wrapping the user key. For p_initial_vector, specify the initialization vector used in wrapping the user key. For p_encrypted_key, specify the user key wrapped with a UFPK. Binary data of the wrapped key is output to p_value field of p_wrapped_key.

For type field of p_wrapped_key, specify the value defined in Table 2-4. The byte size of the output wrapped key is the size of each value defined in Table 2-10. Allocate an enough area according to the algorithm specified in type and specify the address to p_value.

For p_ctrl, specify the pointer to the management structure used in the R_RSIP_Open() function.

Reentrant

Not supported.

4.2.2.2 R_RSIP_EncryptedKeyWrap

Format

```
#include "r_rsip_protected_rx_if.h"
fsp_err_t R_RSIP_EncryptedKeyWrap(
    rsip_ctrl_t * const p_ctrl,
    rsip_wrapped_key_t const * const p_key_update_key,
    void const * const p_initial_vector,
    void const * const p_encrypted_key,
    rsip_wrapped_key_t * const p_wrapped_key
)
```

Parameters

p_ctrl	Input/output	RSIP PM driver management structure
p_key_update_key	Input	Key update key
p_initial_vector	Input	Initialization vector (16 bytes)
p_encrypted_key	Input	Encrypted key
p_wrapped_key	Input/output	Wrapped key (The wrapped key data size which is defined for each key type in Table 2-10 is output to p_value field)

Return Values

FSP_SUCCESS	Normal termination
FSP_ERR_ASSERTION	Null pointer argument
FSP_ERR_NOT_OPEN	RSIP PM driver not open
FSP_ERR_INVALID_STATE	Invalid internal state of the RSIP PM driver
FSP_ERR_NOT_ENABLED	Unsupported key type specified for key_type
FSP_ERR_CRYPTO_RSIP_FAIL	Invalid input parameter
FSP_ERR_CRYPTO_RSIP_RESOURCE_CONFLICT	Occurrence of resource conflict because a hardware resource needed by the processing routine was in use by another processing routine
FSP_ERR_CRYPTO_RSIP_FATAL	Software corruption or hardware failure detected

Description

This API generates a wrapped key from the KUK-wrapped user key. It unwraps the KUK-wrapped user key input in p_encrypted_key using the wrapped key of the KUK input in p_key_update_key and the initialization vector p_initial_vector used when wrapping the user key with a KUK. It then outputs the wrapped key of the user key to p_value field of p_wrapped_key. For type field of p_wrapped_key, specify the value defined in Table 2-4. The byte size of the output wrapped key is the size of each value defined in Table 2-10. Allocate an enough area according to the algorithm specified in type and specify the address to p_value.

For p_ctrl, specify the pointer to the management structure used in the R_RSIP_Open() function.

Reentrant

Not supported.

4.2.2.3 R_RSIP_KeyGenerate

Format

```
#include "r_rsip_protected_rx_if.h"
fsp_err_t R_RSIP_KeyGenerate(
    rsip_ctrl_t * const p_ctrl,
    rsip_wrapped_key_t * const p_wrapped_key
)
```

Parameters

p_ctrl	Input/output	RSIP PM driver management structure
p_wrapped_key	Input/output	Wrapped key (The wrapped key data size which is defined for each key type in Table 2-10 is output to p_value field)

Return Values

FSP_SUCCESS	Normal termination
FSP_ERR_ASSERTION	Null pointer argument
FSP_ERR_NOT_OPEN	RSIP PM driver not open
FSP_ERR_INVALID_STATE	Invalid internal state of the RSIP PM driver
FSP_ERR_NOT_ENABLED	Unsupported key type specified for key_type
FSP_ERR_CRYPTO_RSIP_RESOURCE_CONFLICT	Occurrence of resource conflict because a hardware resource needed by the processing routine was in use by another processing routine
FSP_ERR_CRYPTO_RSIP_FATAL	Software corruption or hardware failure detected

Description

This API outputs the wrapped key of the symmetric key specified in type field of p_wrapped_key to p_value field of p_wrapped_key.

For type field of p_wrapped_key, specify the value defined in Table 2-4. The byte size of the output wrapped key is the size of each value defined in Table 2-10. Allocate an enough area according to the algorithm specified in type and specify the address to p_value.

For p_ctrl, specify the pointer to the management structure used in the R_RSIP_Open() function.

Reentrant

Not supported.

4.2.2.4 R_RSIP_KeyPairGenerate

Format

```
#include "r_rsip_protected_rx_if.h"
fsp_err_t R_RSIP_KeyPairGenerate(
    rsip_ctrl_t * const p_ctrl,
    rsip_wrapped_key_t *const p_wrapped_public_key,
    rsip_wrapped_key_t * const p_wrapped_private_key
)
```

Parameters

p_ctrl	Input/output	RSIP PM driver management structure
p_wrapped_public_key	Input/output	Wrapped public key (The wrapped key data size which is defined for each key type in Table 2-10 is output to p_value field)
p_wrapped_private_key	Input/output	Wrapped private key (The wrapped key data size which is defined for each key type in Table 2-10 is output to p_value field)

Return Values

FSP_SUCCESS	Normal termination
FSP_ERR_ASSERTION	Null pointer argument
FSP_ERR_NOT_OPEN	RSIP PM driver not open
FSP_ERR_INVALID_STATE	Invalid internal state of the RSIP PM driver
FSP_ERR_NOT_ENABLED	Unsupported key type specified for key_type
FSP_ERR_CRYPTO_RSIP_RESOURCE_CONFLICT	Occurrence of resource conflict because a hardware resource needed by the processing routine was in use by another processing routine
FSP_ERR_CRYPTO_RSIP_FATAL	Software corruption or hardware failure detected

Description

This API outputs the wrapped key of the asymmetric key specified in type fields of p_wrapped_public_key and p_wrapped_private_key to p_value fields of them.

For type field of p_wrapped_key, specify the value defined in Table 2-4. The byte size of the output wrapped key is the size of each value defined in Table 2-10. Allocate an enough area according to the algorithm specified in type and specify the address to p_value.

For p_ctrl, specify the pointer to the management structure used in the R_RSIP_Open() function.

Reentrant

Not supported.

4.2.2.5 R_RSIP_PublicKeyExport

Format

```
#include "r_rsip_protected_rx_if.h"
fsp_err_t R_RSIP_PublicKeyExport(
    rsip_wrapped_key_t const * const p_wrapped_public_key,
    uint8_t * const p_raw_public_key
)
```

Parameters

p_wrapped_public_key	Input	Wrapped public key
p_raw_public_key	Output	Plaintext public key (64 byte)

Return Values

FSP_SUCCESS	Normal termination
FSP_ERR_ASSERTION	Null pointer argument
FSP_ERR_CRYPTO_RSIP_KEY_SET_FAIL	Input of an unsupported key type for p_wrapped_public_key

Description

This API extracts a plaintext public key from the wrapped public key.

It outputs the plaintext public key from the wrapped key specified in p_wrapped_public_key to p_raw_public_key.

The public key output to p_raw_public_key will be in the following format:

secp256r1, brainpoolP256r1, secp256k1:
p_raw_public_key[0:31] = Public key Qx
p_raw_public_key[32:63] = Public key Qy

Reentrant

Not supported.

4.2.3 Random Number Generation

4.2.3.1 R_RSIP_RandomNumberGenerate

Format

```
#include "r_rsip_protected_rx_if.h"
fsp_err_t R_RSIP_RandomNumberGenerate(
    rsip_ctrl_t * const p_ctrl,
    uint8_t * const p_random
)
```

Parameters

p_ctrl	Input/output	RSIP PM driver management structure
p_random	Output	16-byte random number

Return Values

FSP_SUCCESS	Normal termination
FSP_ERR_ASSERTION	Null pointer argument
FSP_ERR_NOT_OPEN	RSIP PM driver not open
FSP_ERR_INVALID_STATE	Invalid internal state of the RSIP PM driver
FSP_ERR_CRYPTO_RSIP_RESOURCE_CONFLICT	Occurrence of resource conflict because a hardware resource needed by the processing routine was in use by another processing routine
FSP_ERR_CRYPTO_RSIP_FATAL	Software corruption or hardware failure detected

Description

This API outputs a 16-byte random number value compliant with NIST SP800-90A to p_random.

For p_ctrl, specify the pointer to the management structure used in the R_RSIP_Open() function.

Reentrant

Not supported.

4.2.4 AES

4.2.4.1 R_RSIP_AES_Cipher_Init

Format

```
#include "r_rsip_protected_rx_if.h"
fsp_err_t R_RSIP_AES_Cipher_Init(
    rsip_ctrl_t * const p_ctrl,
    rsip_aes_cipher_mode_t const mode,
    rsip_wrapped_key_t const * const p_wrapped_key,
    uint8_t const * const p_initial_vector
)
```

Parameters

p_ctrl	Input/output	RSIP PM driver management structure
mode	Input	Block encryption mode to be executed
p_wrapped_key	Input	Wrapped key
p_initial_vector	Input	Initialization vector (16 bytes)

Return Values

FSP_SUCCESS	Normal termination
FSP_ERR_ASSERTION	Null pointer argument
FSP_ERR_NOT_OPEN	RSIP PM driver not open
FSP_ERR_INVALID_STATE	Invalid internal state of the RSIP PM driver
FSP_ERR_NOT_ENABLED	Unsupported key type specified for p_wrapped_key
FSP_ERR_INVALID_ARGUMENT	Invalid input key type or mode
FSP_ERR_CRYPTO_RSIP_KEY_SET_FAIL	Occurrence of error in verification of the key specified in p_wrapped_key
FSP_ERR_CRYPTO_RSIP_RESOURCE_CONFLICT	Occurrence of resource conflict because a hardware resource needed by the processing routine was in use by another processing routine
FSP_ERR_CRYPTO_RSIP_FATAL	Software corruption detected

Description

This API performs preparations for execution of an AES cryptographic operation.

For mode, specify the AES block encryption mode to be executed. For p_wrapped_key, specify the wrapped key used for encryption and decryption. The meaning of the data specified for p_initial_vector varies depending on the mode: Initialization vector for CBC, nonce for CTR, or not used for ECB. Specify NULL to p_initial_vector for ECB.

For p_ctrl, specify the pointer to the management structure used in the R_RSIP_Open() function. The result of the R_RSIP_AES_Cipher_Init() function execution is stored in p_ctrl. The value output to p_ctrl is used by the R_RSIP_AES_Cipher_Update() and R_RSIP_AES_Cipher_Finish() functions.

Reentrant

Not supported.

4.2.4.2 R_RSIP_AES_Cipher_Update

Format

```
#include "r_rsip_protected_rx_if.h"
fsp_err_t R_RSIP_AES_Cipher_Update(
    rsip_ctrl_t * const p_ctrl,
    uint8_t const * const p_input,
    uint8_t * const p_output,
    uint32_t const length
)
```

Parameters

p_ctrl	Input/output	RSIP PM driver management structure
p_input	Input	Input data area
p_output	Output	Output data area (the size is same as the value of length)
length	Input	Byte length of input data (0 byte to any length of bytes, must be a multiple of 16)

Return Values

FSP_SUCCESS	Normal termination
FSP_ERR_ASSERTION	Null pointer argument
FSP_ERR_NOT_OPEN	RSIP PM driver not open
FSP_ERR_INVALID_STATE	Invalid internal state of the RSIP PM driver
FSP_ERR_INVALID_SIZE	Invalid input size

Description

This API performs AES cryptographic operations.

When encryption operation is specified in mode of R_RSIP_AES_Cipher_Init(), this API encrypts the plaintext input to p_input for the size specified by length and outputs it to p_output.

When decryption operation is specified in mode of R_RSIP_AES_Cipher_Init(), this API decrypts the ciphertext input to p_input for the size specified by length and outputs it to p_output.

For p_ctrl, specify p_ctrl used in the R_RSIP_AES_Cipher_Init() function.

Reentrant

Not supported.

4.2.4.3 R_RSIP_AES_Cipher_Finish

Format

```
#include "r_rsip_protected_rx_if.h"
fsp_err_t R_RSIP_AES_Cipher_Finish(
    rsip_ctrl_t *const p_ctrl
)
```

Parameters

p_ctrl	Input/output	RSIP PM driver management structure
--------	--------------	-------------------------------------

Return Values

FSP_SUCCESS	Normal termination
FSP_ERR_ASSERTION	Null pointer argument
FSP_ERR_NOT_OPEN	RSIP PM driver not open
FSP_ERR_INVALID_STATE	Invalid internal state of the RSIP PM driver
FSP_ERR_CRYPTO_RSIP_FATAL	Software corruption detected

Description

This API ends an AES cryptographic operation.

For p_ctrl, specify p_ctrl used in the R_RSIP_AES_Cipher_Init() and R_RSIP_AES_Cipher_Update() functions.

Reentrant

Not supported.

4.2.4.4 R_RSIP_AES_AEAD_Init

Format

```
#include "r_rsip_protected_rx_if.h"
fsp_err_t R_RSIP_AES_AEAD_Init(
    rsip_ctrl_t * const p_ctrl,
    rsip_aes_aead_mode_t const mode,
    rsip_wrapped_key_t const * const p_wrapped_key,
    uint8_t const * const p_nonce,
    uint32_t const nonce_length
)
```

Parameters

p_ctrl	Input/output	RSIP PM driver management structure
mode	Input	AEAD mode to be executed
p_wrapped_key	Input	Wrapped key
p_nonce	Input	Nonce
nonce_length	Input	Byte length of the nonce (must be greater than or equal to 1 for GCM, 7~11 byte for CCM)

Return Values

FSP_SUCCESS	Normal termination
FSP_ERR_ASSERTION	Null pointer argument
FSP_ERR_NOT_OPEN	RSIP PM driver not open
FSP_ERR_INVALID_STATE	Invalid internal state of the RSIP PM driver
FSP_ERR_NOT_ENABLED	Unsupported key type specified for p_wrapped_key
FSP_ERR_INVALID_ARGUMENT	Invalid input key type or mode
FSP_ERR_CRYPTO_RSIP_KEY_SET_FAIL	Occurrence of error in verification of the key specified in p_wrapped_key
FSP_ERR_CRYPTO_RSIP_RESOURCE_CONFLICT	Occurrence of resource conflict because a hardware resource needed by the processing routine was in use by another processing routine
FSP_ERR_CRYPTO_RSIP_FATAL	Software corruption detected

Description

The R_RSIP_AES_AEAD_Init() function performs preparations for execution of an AES AEAD operation.

For mode, specify the AEAD mode to be executed. For p_wrapped_key, specify the wrapped key used for AEAD. The meaning of the data specified for p_nonce varies depending on the mode: Initialization vector when the GCM mode was specified, or nonce when the CCM mode was specified. For nonce_length, input the initialization vector specified in p_nonce or the byte length of the nonce.

For p_ctrl, specify the pointer to the management structure used in the R_RSIP_OPEN() function. The result of the R_RSIP_AES_AEAD_Init() function is stored in p_ctrl. The value output to p_ctrl is used by

the R_RSIP_AES_AEAD_LengthsSet(), R_RSIP_AES_AEAD_AADUpdate(),
R_RSIP_AES_AEAD_Update(), R_RSIP_AES_AEAD_Finish(), and R_RSIP_AES_AEAD_Verify()
functions.

Reentrant

Not supported.

4.2.4.5 R_RSIP_AES_AEAD_LengthsSet

Format

```
#include "r_rsip_protected_rx_if.h"
fsp_err_t R_RSIP_AES_AEAD_LengthsSet(
    rsip_ctrl_t * const p_ctrl,
    uint32_t const total_aad_length,
    uint32_t const total_text_length,
    uint32_t const tag_length
)
```

Parameters

p_ctrl	Input/output	RSIP PM driver management structure
total_aad_length	Input	Byte length of AAD (110 bytes or less)
total_text_length	Input	Byte length of input/output data (0 byte to any length of bytes)
tag_length	Input	Byte length of the authentication tag (4, 6, 8, 10, 12, 14, or 16 bytes)

Return Values

FSP_SUCCESS	Normal termination
FSP_ERR_NOT_OPEN	RSIP PM driver not open
FSP_ERR_INVALID_SIZE	Invalid input size
FSP_ERR_INVALID_STATE	Invalid internal state of the RSIP PM driver

Description

This API is used when performing AES CCM operations. Specify the byte lengths of AAD, input/output data, and authentication tag.

For p_ctrl, specify p_ctrl used in the R_RSIP_AES_AEAD_Init() function.

Reentrant

Not supported.

4.2.4.6 R_RSIP_AES_AEAD_AADUpdate

Format

```
#include "r_rsip_protected_rx_if.h"
fsp_err_t R_RSIP_AES_AEAD_AADUpdate(
    rsip_ctrl_t * const p_ctrl,
    uint8_t const * const p_aad,
    uint32_t const aad_length
)
```

Parameters

p_ctrl	Input/output	RSIP PM driver management structure
p_aad	Input	AAD area
aad_length	Input	Byte length of AAD (0 byte to any length of bytes)

Return Values

FSP_SUCCESS	Normal termination
FSP_ERR_ASSERTION	Null pointer argument
FSP_ERR_NOT_OPEN	RSIP PM driver not open
FSP_ERR_INVALID_STATE	Invalid internal state of the RSIP PM driver
FSP_ERR_CRYPTO_RSIP_KEY_SET_FAIL	Occurrence of error in verification of p_wrapped_key which is input in R_RSIP_AES_AEAD_Init
FSP_ERR_CRYPTO_RSIP_FAIL	Internal error
FSP_ERR_CRYPTO_RSIP_RESOURCE_CONFLICT	Occurrence of resource conflict because a hardware resource needed by the processing routine was in use by another processing routine
FSP_ERR_CRYPTO_RSIP_FATAL	Software corruption detected

Description

This API specifies the AAD used in an AES AEAD operation.

For p_aad, input the AAD. For aad_length, input the byte size of the AAD.

For p_ctrl, specify p_ctrl used in the R_RSIP_AES_AEAD_Init() and R_RSIP_AES_AEAD_LengthsSet() functions.

Before calling the R_RSIP_AES_AEAD_Update() function, input AAD with this function.

Reentrant

Not supported.

4.2.4.7 R_RSIP_AES_AEAD_Update

Format

```
#include "r_rsip_protected_rx_if.h"
fsp_err_t R_RSIP_AES_AEAD_Update(
    rsip_ctrl_t * const p_ctrl,
    uint8_t const * const p_input,
    uint32_t const input_length,
    uint8_t * const p_output,
    uint32_t * const p_output_length
)
```

Parameters

p_ctrl	Input/output	RSIP PM driver management structure
p_input	Input	Input data area
input_length	Input	Byte length of input data (0 byte to any length of bytes)
p_output	Output	Output data area
p_output_length	Output	Byte length of output data

Return Values

FSP_SUCCESS	Normal termination
FSP_ERR_ASSERTION	Null pointer argument
FSP_ERR_NOT_OPEN	RSIP PM driver not open
FSP_ERR_INVALID_STATE	Invalid internal state of the RSIP PM driver
FSP_ERR_INVALID_SIZE	Invalid byte size specified for Input_length

Description

This API performs an AES AEAD operation.

When encryption operation is specified in mode of R_RSIP_AES_AEAD_Init(), the API encrypts the plaintext input to p_input for the size specified by input_length and outputs encrypted data to p_output. If the size specified in input_length is not 16-byte aligned, the API encrypts the fraction of 16-byte alignment in the R_RSIP_AES_AEAD_Update(), R_RSIP_AES_AEAD_Finish(), or R_RSIP_AES_AEAD_Verify() which is called next. The size which is encrypted and then output to p_output is output to p_output_length. When decryption operation is specified in mode of R_RSIP_AES_AEAD_Init(), this API decrypts the ciphertext input to p_input for the size specified by length and outputs decrypted data to p_output. If the size specified in input_length is not 16-byte aligned, the API decrypts the fraction of 16-byte alignment in the R_RSIP_AES_AEAD_Update(), R_RSIP_AES_AEAD_Finish(), or R_RSIP_AES_AEAD_Verify() which is called next. The size which is decrypted and then output to p_output is output to p_output_length. Except in cases where the addresses are the same, specify areas for p_input and p_output that do not overlap.

For p_ctrl, specify p_ctrl used in the R_RSIP_AES_AEAD_Init(), R_RSIP_AES_AEAD_LengthsSet(), and R_RSIP_AES_AEAD_AADUpdate() functions.

Reentrant

Not supported.

4.2.4.8 R_RSIP_AES_AEAD_Finish

Format

```
#include "r_rsip_protected_rx_if.h"
fsp_err_t R_RSIP_AES_AEAD_Finish(
    rsip_ctrl_t * const p_ctrl,
    uint8_t * const p_output,
    uint32_t * const p_output_length,
    uint8_t * const p_tag
)
```

Parameters

p_ctrl	Input/output	RSIP PM driver management structure
p_output	Output	Output data area (all output data which includes buffered data)
p_output_length	Output	Byte length of output data
p_tag	Output	Authentication tag area (16 bytes)

Return Values

FSP_SUCCESS	Normal termination
FSP_ERR_ASSERTION	Null pointer argument
FSP_ERR_NOT_OPEN	RSIP PM driver not open
FSP_ERR_INVALID_STATE	Invalid internal state of the RSIP PM driver
FSP_ERR_CRYPTO_RSIP_FAIL	Occurrence of internal error
FSP_ERR_CRYPTO_RSIP_FATAL	Software corruption detected

Description

This API performs AES AEAD encryption and authentication tag generation.

If the total data size input by R_RSIP_AES_AEAD_Update() is not 16-byte aligned, the API outputs the ciphertext data for the fraction of 16-byte alignment in p_output of the R_RSIP_AES_AEAD_Finish() function, and the size of the ciphertext data output from this API in p_output_length. An authentication tag will be output to p_tag.

Use R_RSIP_AES_AEAD_Verify() for decryption and tag verification.

For p_ctrl, specify p_ctrl used in the R_RSIP_AES_AEAD_Init(), R_RSIP_AES_AEAD_LengthsSet(), R_RSIP_AES_AEAD_AADUpdate(), and R_RSIP_AES_AEAD_Update() functions.

Reentrant

Not supported.

4.2.4.9 R_RSIP_AES_AEAD_Verify

Format

```
#include "r_rsip_protected_rx_if.h"
fsp_err_t R_RSIP_AES_AEAD_Verify(
    rsip_ctrl_t * const p_ctrl,
    uint8_t * const p_output,
    uint32_t * const p_output_length,
    uint8_t const * const p_tag,
    uint32_t const tag_length
)
```

Parameters

p_ctrl	Input/output	RSIP PM driver management structure
p_output	Output	Output data area (all output data which includes buffered data)
p_output_length	Output	Byte length of output data
p_tag	Input	Authentication tag area
tag_length	Input	Byte length of the authentication tag (1 to 16 bytes)

Return Values

FSP_SUCCESS	Normal termination
FSP_ERR_ASSERTION	Null pointer argument
FSP_ERR_NOT_OPEN	RSIP PM driver not open
FSP_ERR_INVALID_STATE	Invalid internal state of the RSIP PM driver
FSP_ERR_INVALID_SIZE	Invalid tag_length value
FSP_ERR_CRYPTO_RSIP_FAIL	Occurrence of internal error
FSP_ERR_CRYPTO_RSIP_AUTHENTICATION	Authentication error
FSP_ERR_CRYPTO_RSIP_FATAL	Software corruption detected

Description

This API performs decryption and tag verification.

If the total data size input by R_RSIP_AES_AEAD_Update() is not 16-byte aligned, the API outputs the plaintext data for the fraction in p_output of the R_RSIP_AES_AEAD_Verify() function, and the size of the plaintext data output from this API in p_output_length. For p_tag, input the authentication tag value to be used for authentication. For tag_length, input the byte length of the tag to be used for tag authentication. Use R_RSIP_AES_AEAD_Finish() for encryption and tag generation.

For p_ctrl, specify p_ctrl used in the R_RSIP_AES_AEAD_Init(), R_RSIP_AES_AEAD_LengthsSet(), R_RSIP_AES_AEAD_AADUpdate(), and R_RSIP_AES_AEAD_Update() functions.

Reentrant

Not supported.

4.2.4.10 R_RSIP_AES_MAC_Init

Format

```
#include "r_rsip_protected_rx_if.h"
fsp_err_t R_RSIP_AES_MAC_Init(
    rsip_ctrl_t * const p_ctrl,
    rsip_aes_mac_mode_t const mode,
    rsip_wrapped_key_t const * const p_wrapped_key
)
```

Parameters

p_ctrl	Input/output	RSIP PM driver management structure
mode	Input	MAC calculation mode to be executed
p_wrapped_key	Input	Wrapped key

Return Values

FSP_SUCCESS	Normal termination
FSP_ERR_ASSERTION	Null pointer argument
FSP_ERR_NOT_OPEN	RSIP PM driver not open
FSP_ERR_INVALID_STATE	Invalid internal state of the RSIP PM driver
FSP_ERR_NOT_ENABLED	Unsupported key type specified for p_wrapped_key
FSP_ERR_CRYPTO_RSIP_KEY_SET_FAIL	Occurrence of error in verification of p_wrapped_key
FSP_ERR_CRYPTO_RSIP_RESOURCE_CONFLICT	Occurrence of resource conflict because a hardware resource needed by the processing routine was in use by another processing routine
FSP_ERR_CRYPTO_RSIP_FATAL	Software corruption detected

Description

This API performs preparations for execution of an AES MAC calculation.

For mode, specify the MAC mode to be executed. For p_wrapped_key, specify the wrapped key used for MAC calculation.

For p_ctrl, specify the pointer to the management structure used in the R_RSIP_Open() function. The preparation result for AES MAC calculation execution is stored in p_ctrl. The value output to p_ctrl is used by the R_RSIP_AES_MAC_Update(), R_RSIP_AES_MAC_Finish(), and R_RSIP_AES_MAC_Verify() functions.

Reentrant

Not supported.

4.2.4.11 R_RSIP_AES_MAC_Update

Format

```
#include "r_rsip_protected_rx_if.h"
fsp_err_t R_RSIP_AES_MAC_Update(
    rsip_ctrl_t * const p_ctrl,
    uint8_t const * const p_message,
    uint32_t const message_length
)
```

Parameters

p_ctrl	Input/output	RSIP PM driver management structure
p_message	Input	Message data area
message_length	Input	Byte length of message data (0 byte to any length of bytes)

Return Values

FSP_SUCCESS	Normal termination
FSP_ERR_ASSERTION	Null pointer argument
FSP_ERR_NOT_OPEN	RSIP PM driver not open
FSP_ERR_INVALID_STATE	Invalid internal state of the RSIP PM driver

Description

This API performs an AES MAC calculation. After the calculation is complete, the API calls R_RSIP_AES_MAC_SignFinish() or R_RSIP_AES_MAC_VerifyFinish().
For p_message, specify the message to be MAC-verified or generated. For message_length, specify the length of the message to be input.

For p_ctrl, specify p_ctrl used in the R_RSIP_AES_MAC_Init() function.

Reentrant

Not supported.

4.2.4.12 R_RSIP_AES_MAC_SignFinish

Format

```
#include "r_rsip_protected_rx_if.h"
fsp_err_t R_RSIP_AES_MAC_SignFinish(
    rsip_ctrl_t * const p_ctrl,
    uint8_t * const p_mac
)
```

Parameters

p_ctrl	Input/output	RSIP PM driver management structure
p_mac	Output	MAC data area (16 bytes)

Return Values

FSP_SUCCESS	Normal termination
FSP_ERR_ASSERTION	Null pointer argument
FSP_ERR_NOT_OPEN	RSIP PM driver not open
FSP_ERR_INVALID_STATE	Invalid internal state of the RSIP PM driver
FSP_ERR_CRYPTO_RSIP_FAIL	Internal error
FSP_ERR_CRYPTO_RSIP_FATAL	Software corruption detected

Description

This API generates MAC generation.
It outputs the MAC value of the message input by the R_RSIP_AES_MAC_Update() function to p_mac.
Use R_RSIP_AES_MAC_VerifyFinish() to perform MAC verification.

For p_ctrl, specify p_ctrl used in the R_RSIP_AES_MAC_Init() and R_RSIP_AES_MAC_Update()
functions.

Reentrant

Not supported.

4.2.4.13 R_RSIP_AES_MAC_VerifyFinish

Format

```
#include "r_rsip_protected_rx_if.h"
fsp_err_t R_RSIP_AES_MAC_VerifyFinish(
    rsip_ctrl_t * const p_ctrl,
    uint8_t const * const p_mac,
    uint32_t const mac_length
)
```

Parameters

p_ctrl	Input/output	RSIP PM driver management structure
p_mac	Input	MAC data area
mac_length	Input	Byte length of MAC data (2 to 16 bytes)

Return Values

FSP_SUCCESS	Normal termination
FSP_ERR_ASSERTION	Null pointer argument
FSP_ERR_NOT_OPEN	RSIP PM driver not open
FSP_ERR_INVALID_STATE	Invalid internal state of the RSIP PM driver
FSP_ERR_INVALID_SIZE	Invalid size specified for mac_length
FSP_ERR_CRYPTO_RSIP_FAIL	Internal error
FSP_ERR_CRYPTO_RSIP_AUTHENTICATION	MAC verification error
FSP_ERR_CRYPTO_RSIP_FATAL	Software corruption detected

Description

This API performs MAC verification.

It verifies the MAC value of the message input in the R_RSIP_AES_MAC_Update() function against the MAC input in p_mac. It verifies the MAC value input in p_mac as a valid MAC value for the number of bytes specified in mac_length.

Use R_RSIP_AES_MAC_SignFinish() to perform MAC generation.

For p_ctrl, specify p_ctrl used in the R_RSIP_AES_MAC_Init() and R_RSIP_AES_MAC_Update() functions.

Reentrant

Not supported.

4.2.5 ECC

4.2.5.1 R_RSIP_ECDSA_Sign

Format

```
#include "r_rsip_protected_rx_if.h"
fsp_err_t R_RSIP_ECDSA_Sign(
    rsip_ctrl_t * const p_ctrl,
    rsip_wrapped_key_t const * const p_wrapped_private_key,
    uint8_t const * const p_hash,
    uint8_t *const p_signature
)
```

Parameters

p_ctrl	Input/output	RSIP PM driver management structure
p_wrapped_private_key	Input	Wrapped key
p_hash	Input	Hash value A hash value of the same size as the key length is input.
p_signature	Output	Signature A signature twice the size of the key length is output. (64 byte)

Return Values

FSP_SUCCESS	Normal termination
FSP_ERR_ASSERTION	Null pointer argument
FSP_ERR_NOT_OPEN	RSIP PM driver not open
FSP_ERR_INVALID_STATE	Invalid internal state of the RSIP PM driver
FSP_ERR_NOT_ENABLED	Unsupported key type specified for p_wrapped_key
FSP_ERR_CRYPTO_RSIP_KEY_SET_FAIL	Occurrence of error in verification of p_wrapped_key
FSP_ERR_CRYPTO_RSIP_FAIL	Invalid input parameter or occurrence of error in signature generation
FSP_ERR_CRYPTO_RSIP_RESOURCE_CONFLICT	Occurrence of resource conflict because a hardware resource needed by the processing routine was in use by another processing routine
FSP_ERR_CRYPTO_RSIP_FATAL	Software corruption detected

Description

This API generates an ECDSA signature.
It outputs the ECDSA signature of the hash value input in p_hash to p_signature using the private key input in p_wrapped_private_key.

For p_ctrl, specify the pointer to the management structure used in the R_RSIP_Open() function.

Reentrant

Not supported.

4.2.5.2 R_RSIP_ECDSA_Verify

Format

```
#include "r_rsip_protected_rx_if.h"
fsp_err_t R_RSIP_ECDSA_Verify(
    rsip_ctrl_t *const p_ctrl,
    rsip_wrapped_key_t const * const p_wrapped_public_key,
    uint8_t const * const p_hash,
    uint8_t const * const p_signature
)
```

Parameters

p_ctrl	Input/output	RSIP PM driver management structure
p_wrapped_public_key	Input	Wrapped key
p_hash	Input	Hash value to be verified A hash value whose byte length is the same size as the key length is input.
p_signature	Input	Signature to be verified A signature twice the size of the key length is input.

Return Values

FSP_SUCCESS	Normal termination
FSP_ERR_ASSERTION	Null pointer argument
FSP_ERR_NOT_OPEN	RSIP PM driver not open
FSP_ERR_INVALID_STATE	Invalid internal state of the RSIP PM driver
FSP_ERR_NOT_ENABLED	Unsupported key type specified for p_wrapped_key
FSP_ERR_CRYPTO_RSIP_KEY_SET_FAIL	Occurrence of error in verification of p_wrapped_key
FSP_ERR_CRYPTO_RSIP_FAIL	Invalid input parameter or occurrence of error in signature verification
FSP_ERR_CRYPTO_RSIP_RESOURCE_CONFLICT	Occurrence of resource conflict because a hardware resource needed by the processing routine was in use by another processing routine
FSP_ERR_CRYPTO_RSIP_FATAL	Software corruption detected

Description

This API performs ECDSA signature verification.
It uses the public key input in p_wrapped_public_key to verify the hash value input in p_hash and the ECDSA signature value input in p_signature.

For p_ctrl, specify the pointer to the management structure used in the R_RSIP_Open() function.

Reentrant

Not supported.

4.2.5.3 R_RSIP_PKI_ECDSA_CertVerify

Format

```
#include "r_rsip_protected_rx_if.h"
fsp_err_t R_RSIP_ECDSA_CertVerify(
    rsip_ctrl_t *const p_ctrl,
    rsip_wrapped_key_t const * const p_wrapped_public_key,
    uint8_t const * const p_hash,
    uint8_t const * const p_signature
)
```

Parameters

p_ctrl	Input/output	RSIP PM driver management structure
p_wrapped_public_key	Input	Wrapped key
p_hash	Input	Hash value to be verified A hash value whose byte length is the same size as the key length is input.
p_signature	Input	Signature to be verified A signature twice the size of the key length is input.

Return Values

FSP_SUCCESS	Normal termination
FSP_ERR_ASSERTION	Null pointer argument
FSP_ERR_NOT_OPEN	RSIP PM driver not open
FSP_ERR_INVALID_STATE	Invalid internal state of the RSIP PM driver
FSP_ERR_NOT_ENABLED	Unsupported key type specified for p_wrapped_key
FSP_ERR_CRYPTO_RSIP_KEY_SET_FAIL	Occurrence of error in verification of p_wrapped_key
FSP_ERR_CRYPTO_RSIP_FAIL	Invalid input parameter or occurrence of error in signature verification
FSP_ERR_CRYPTO_RSIP_RESOURCE_CONFLICT	Occurrence of resource conflict because a hardware resource needed by the processing routine was in use by another processing routine
FSP_ERR_CRYPTO_RSIP_FATAL	Software corruption detected

Description

Verifies a public key certificate with ECDSA.

Message hash p_hash should be computed in advance. In the case of hash length is less than the key length, padding is required to make it the same as the key length. Signature to be verified is input to p_signature.

For p_ctrl, specify the pointer to the management structure used in the R_RSIP_Open() function.

Reentrant

Not supported.

4.2.5.4 R_RSIP_PKI_CertKeyImport

Format

```
#include "r_rsip_protected_rx_if.h"
fsp_err_t R_RSIP_ECDSA_CertKeyImport(
    rsip_ctrl_t *const p_ctrl,
    uint8_t const * const p_cert,
    uint32_t const cert_length,
    uint8_t const * const p_key_param1,
    uint32_t const key_param1_length,
    uint8_t const * const p_key_param2,
    uint32_t const key_param2_length,
    rsip_hash_type_t const hash_function,
    rsip_wrapped_key_t * const p_wrapped_public_key
)
```

Parameters

p_ctrl	Input/output	RSIP PM driver management structure
p_cert	Input	Certificate
cert_length	Input	Byte length of certificate (0 byte to any length of bytes)
p_key_param1	Input	Public key parameter in certificate It means Qx when ECC is used.
key_param1_length	Input	Byte length public key parameter in certificate (p_key_param1)
p_key_param2	Input	Public key parameter in certificate It means Qy when ECC is used.
key_param2_length	Input	Byte length public key parameter in certificate (p_key_param2)
hash_function	Input	The hash function used when verifying certificate signature
p_wrapped_public_key	Input/output	Wrapped Key (The wrapped key data size which is defined for each key type in Table 2-10 is output to p_value field)

Return Values

FSP_SUCCESS	Normal termination
FSP_ERR_ASSERTION	Null pointer argument
FSP_ERR_NOT_OPEN	RSIP PM driver not open
FSP_ERR_INVALID_STATE	Invalid internal state of the RSIP PM driver
FSP_ERR_NOT_ENABLED	Unsupported key type specified for p_wrapped_key
FSP_ERR_CRYPTO_RSIP_FAIL	Invalid input parameter or occurrence of error in signature verification
FSP_ERR_CRYPTO_RSIP_RESOURCE_CONFLICT	Occurrence of resource conflict because a hardware resource needed by the processing routine was in use by another processing routine

FSP_ERR_CRYPTO_RSIP_FATAL

Software corruption detected

Description

Wraps the public key in the verified public key certificate.

Certificate and its byte length are input to p_cert and cert_length, key parameter Qx in the certificate and its byte length are input to p_key_param1 and key_param1_length, key parameter Qy in the certificate and its byte length are input to p_key_param2 and key_param2_length, and hash function used when verifying certificate signature is input to hash_function.

Wrapped public key is output to p_wrapped_public_key.

For type field of p_wrapped_public_key, specify the value defined in Table 2-4. The byte size of the output wrapped key is the size of each value defined in Table 2-10. Allocate an enough area according to the algorithm specified in type and specify the address to p_value.

For p_ctrl, specify the pointer to the management structure used in the R_RSIP_Open() function.

Reentrant

Not supported.

4.2.5.5 R_RSIP_PKI_VerifiedCertInfoExport

Format

```
#include "r_rsip_protected_rx_if.h"
fsp_err_t R_RSIP_ECDSA_VerifiedCertInfoExport(
    rsip_ctrl_t *const p_ctrl,
    rsip_verified_cert_info_t * const p_verified_cert_info
)
```

Parameters

p_ctrl	Input/output	RSIP PM driver management structure
p_verified_cert_info	Output	Certificate to be signed (52 byte)

Return Values

FSP_SUCCESS	Normal termination
FSP_ERR_ASSERTION	Null pointer argument

Description

Exports verified certificate information stored in this driver.
Certificate to be signed is output to p_verified_cert_info.

For p_ctrl, specify the pointer to the management structure used in the R_RSIP_Open() function.

Reentrant

Not supported.

4.2.5.6 R_RSIP_PKI_VerifiedCertInfoImport

Format

```
#include "r_rsip_protected_rx_if.h"
fsp_err_t R_RSIP_ECDSA_VerifiedCertInfoImport(
    rsip_ctrl_t *const p_ctrl,
    rsip_verified_cert_info_t * const p_verified_cert_info
)
```

Parameters

p_ctrl	Input/output	RSIP PM driver management structure
p_verified_cert_info	Input	Certificate to be signed

Return Values

FSP_SUCCESS	Normal termination
FSP_ERR_ASSERTION	Null pointer argument

Description

Imports verified certificate information.
Certificate to be signed is input to p_verified_cert_info.
For p_ctrl, specify the pointer to the management structure used in the R_RSIP_Open() function.

Reentrant

Not supported.

4.2.5.7 R_RSIP_ECDH_KeyAgree

Format

```
#include "r_rsip_protected_rx_if.h"
fsp_err_t R_RSIP_ECDH_KeyAgree(
    rsip_ctrl_t *const p_ctrl,
    rsip_wrapped_key_t const * const p_wrapped_private_key,
    rsip_wrapped_key_t const * const p_wrapped_public_key,
    rsip_wrapped_secret_t * const p_wrapped_secret
)
```

Parameters

p_ctrl	Input/output	RSIP PM driver management structure
p_wrapped_private_key	Input	Wrapped local private key
p_wrapped_public_key	Input	Wrapped peer public key
p_wrapped_secret	Output	Wrapped secret (56 byte)

Return Values

FSP_SUCCESS	Normal termination
FSP_ERR_ASSERTION	Null pointer argument
FSP_ERR_NOT_OPEN	RSIP PM driver not open
FSP_ERR_INVALID_STATE	Invalid internal state of the RSIP PM driver
FSP_ERR_NOT_ENABLED	Unsupported key type specified for p_wrapped_key
FSP_ERR_CRYPTO_RSIP_KEY_SET_FAIL	Occurrence of error in verification of p_wrapped_key
FSP_ERR_CRYPTO_RSIP_FAIL	Invalid input parameter or occurrence of error in signature verification
FSP_ERR_CRYPTO_RSIP_RESOURCE_CONFLICT	Occurrence of resource conflict because a hardware resource needed by the processing routine was in use by another processing routine
FSP_ERR_CRYPTO_RSIP_FATAL	Software corruption detected

Description

Computes ECDH secret with wrapped private key and wrapped public key.
For p_wrapped_private_key, specify wrapped local private key. For p_wrapped_public_key, specify wrapped peer public key.
It outputs the wrapped secret to p_wrapped_secret.

For p_ctrl, specify the pointer to the management structure used in the R_RSIP_Open() function.

Reentrant

Not supported.

4.2.5.8 R_RSIP_ECDH_PlainKeyAgree

Format

```
#include "r_rsip_protected_rx_if.h"
fsp_err_t R_RSIP_ECDH_PlainKeyAgree(
    rsip_ctrl_t *const p_ctrl,
    rsip_wrapped_key_t const * const p_wrapped_private_key,
    uint8_t const * const p_plain_public_key,
    rsip_wrapped_secret_t * const p_wrapped_secret
)
```

Parameters

p_ctrl	Input/output	RSIP PM driver management structure
p_wrapped_private_key	Input	Wrapped local private key
p_wrapped_public_key	Input	Plain peer public key
p_wrapped_secret	Output	Wrapped secret (56 byte)

Return Values

FSP_SUCCESS	Normal termination
FSP_ERR_ASSERTION	Null pointer argument
FSP_ERR_NOT_OPEN	RSIP PM driver not open
FSP_ERR_INVALID_STATE	Invalid internal state of the RSIP PM driver
FSP_ERR_NOT_ENABLED	Unsupported key type specified for p_wrapped_key
FSP_ERR_CRYPTO_RSIP_KEY_SET_FAIL	Occurrence of error in verification of p_wrapped_key
FSP_ERR_CRYPTO_RSIP_FAIL	Invalid input parameter or occurrence of error in signature verification
FSP_ERR_CRYPTO_RSIP_RESOURCE_CONFLICT	Occurrence of resource conflict because a hardware resource needed by the processing routine was in use by another processing routine
FSP_ERR_CRYPTO_RSIP_FATAL	Software corruption detected

Description

Computes ECDH secret with wrapped private key and plain public key.

For p_wrapped_private_key, specify wrapped local private key. For p_plain_public_key, specify plain peer public key.

It outputs the wrapped secret to p_wrapped_secret.

For p_ctrl, specify the pointer to the management structure used in the R_RSIP_Open() function.

Note that there is a risk of man-in-the-middle attacks, since peer public key is in plain text. Therefore, it is recommended to use R_RSIP_ECDH_KeyAgree() as far as possible.

Reentrant

Not supported.

4.2.6 Hash

4.2.6.1 R_RSIP_SHA_Compute

Format

```
#include "r_rsip_protected_rx_if.h"
fsp_err_t R_RSIP_SHA_Init(
    rsip_ctrl_t * const p_ctrl,
    rsip_hash_type_t const hash_type,
    uint8_t const * const p_message,
    uint32_t const message_length,
    uint8_t * const p_digest
)
```

Parameters

p_ctrl	Input/output	RSIP PM driver management structure
hash_type	Input	Hash calculation mode to be executed
p_message	Input	Message data area
message_length	Input	Byte length of message data (0 byte to any length of bytes)
p_digest	Output	Hash data area SHA224: 28 byte SHA256: 32 byte

Return Values

FSP_SUCCESS	Normal termination
FSP_ERR_ASSERTION	Null pointer argument
FSP_ERR_NOT_OPEN	RSIP PM driver not open
FSP_ERR_INVALID_STATE	Invalid internal state of the RSIP PM driver
FSP_ERR_NOT_ENABLED	Unsupported hash calculation mode specified for hash_type
FSP_ERR_INVALID_ARGUMENT	Invalid input key type or mode

Description

This API performs preparations for execution of a hash calculation.
For hash_type, specify the hash calculation mode to be executed.

For p_ctrl, specify the pointer to the management structure used in the R_RSIP_Open() function. For p_message, input the message to be hash-calculated. For message_length, specify the length of the message to be input in p_message.

It outputs the hash value of the message to p_digest.

Reentrant

Not supported.

4.2.6.2 R_RSIP_SHA_Init

Format

```
#include "r_rsip_protected_rx_if.h"
fsp_err_t R_RSIP_SHA_Init(
    rsip_ctrl_t * const p_ctrl,
    rsip_hash_type_t const hash_type
)
```

Parameters

p_ctrl	Input/output	RSIP PM driver management structure
hash_type	Input	Hash calculation mode to be executed

Return Values

FSP_SUCCESS	Normal termination
FSP_ERR_ASSERTION	Null pointer argument
FSP_ERR_NOT_OPEN	RSIP PM driver not open
FSP_ERR_INVALID_STATE	Invalid internal state of the RSIP PM driver
FSP_ERR_NOT_ENABLED	Unsupported hash calculation mode specified for hash_type
FSP_ERR_INVALID_ARGUMENT	Invalid input key type or mode

Description

This API performs preparations for execution of a hash calculation.
For hash_type, specify the hash calculation mode to be executed.

For p_ctrl, specify the pointer to the management structure used in the R_RSIP_Open() function. The preparation result for hash calculation execution is stored in p_ctrl. The value output to p_ctrl is used by the R_RSIP_SHA_Update() and R_RSIP_SHA_Finish() functions.

Reentrant

Not supported.

4.2.6.3 R_RSIP_SHA_Update

Format

```
#include "r_rsip_protected_rx_if.h"
fsp_err_t R_RSIP_SHA_Update(
    rsip_ctrl_t * const p_ctrl,
    uint8_t const * const p_message,
    uint32_t const message_length
)
```

Parameters

p_ctrl	Input/output	RSIP PM driver management structure
p_message	Input	Message data area
message_length	Input	Byte length of message data (0 byte to any length of bytes)

Return Values

FSP_SUCCESS	Normal termination
FSP_ERR_ASSERTION	Null pointer argument
FSP_ERR_NOT_OPEN	RSIP PM driver not open
FSP_ERR_INVALID_STATE	Invalid internal state of the RSIP PM driver
FSP_ERR_CRYPTO_RSIP_RESOURCE_CONFLICT	Occurrence of resource conflict because a hardware resource needed by the processing routine was in use by another processing routine
FSP_ERR_CRYPTO_RSIP_FATAL	Software corruption detected

Description

This API performs a hash calculation.

For p_message, input the message to be hash-calculated. For message_length, specify the length of the message to be input in p_message.

For p_ctrl, specify p_ctrl used in the R_RSIP_SHA_Init() function.

Reentrant

Not supported.

4.2.6.4 R_RSIP_SHA_Finish

Format

```
#include "r_rsip_protected_rx_if.h"
fsp_err_t R_RSIP_SHA_Finish(
    rsip_ctrl_t * const p_ctrl,
    uint8_t * const p_digest
)
```

Parameters

p_ctrl	Input/output	RSIP PM driver management structure
p_digest	Output	Hash data area SHA224: 28 byte SHA256: 32 byte

Return Values

FSP_SUCCESS	Normal termination
FSP_ERR_ASSERTION	Null pointer argument
FSP_ERR_NOT_OPEN	RSIP PM driver not open
FSP_ERR_INVALID_STATE	Invalid internal state of the RSIP PM driver
FSP_ERR_CRYPTO_RSIP_RESOURCE_CONFLICT	Occurrence of resource conflict because a hardware resource needed by the processing routine was in use by another processing routine
FSP_ERR_CRYPTO_RSIP_FATAL	Software corruption detected

Description

This API outputs the result of a hash calculation.
It outputs the hash value of the message input in the R_RSIP_SHA_Update() function to p_digest.
For p_ctrl, specify p_ctrl used in the R_RSIP_SHA_Init() and R_RSIP_SHA_Update() functions.

Reentrant

Not supported.

4.2.6.5 R_RSIP_SHA_Suspend

Format

```
#include "r_rsip_protected_rx_if.h"
fsp_err_t R_RSIP_SHA_Suspend(
    rsip_ctrl_t * const p_ctrl,
    rsip_sha_handle_t * const p_handle
)
```

Parameters

p_ctrl	Input/output	RSIP PM driver management structure
p_handle	Output	Control information for hash calculation (128 byte)

Return Values

FSP_SUCCESS	Normal termination
FSP_ERR_ASSERTION	Null pointer argument
FSP_ERR_NOT_OPEN	RSIP PM driver not open
FSP_ERR_INVALID_STATE	Invalid internal state of the RSIP PM driver

Description

After execution with the R_RSIP_SHA_Init() and R_RSIP_SHA_Update() functions, this API suspends a hash calculation midway so that other multi-part or single-part operations can be performed. It outputs the intermediate value of the suspended hash calculation to p_handle.

If R_RSIP_SHA_Finish() function is called after R_RSIP_SHA_Suspend() function is called, this API outputs the hash value of the message input to R_RSIP_SHA_Update() function up to that point.
To resume the hash calculation, call the R_RSIP_SHA_Resume() function.

For p_ctrl, specify p_ctrl used in the R_RSIP_SHA_Init() and R_RSIP_SHA_Update() functions.

Reentrant

Not supported.

4.2.6.6 R_RSIP_SHA_Resume

Format

```
#include "r_rsip_protected_rx_if.h"
fsp_err_t R_RSIP_SHA_Suspend(
    rsip_ctrl_t * const p_ctrl,
    rsip_sha_handle_t * const p_handle
)
```

Parameters

p_ctrl	Input/output	RSIP PM driver management structure
p_handle	Input	Control information for hash calculation

Return Values

FSP_SUCCESS	Normal termination
FSP_ERR_ASSERTION	Null pointer argument
FSP_ERR_NOT_OPEN	RSIP PM driver not open
FSP_ERR_INVALID_STATE	Invalid internal state of the RSIP PM driver

Description

This API resumes the hash calculation suspended by the R_RSIP_SHA_Suspend() function.
For p_handle, input p_handle output by the R_RSIP_SHA_Suspend() function.

For p_ctrl, specify the pointer to the management structure used in the R_RSIP_Open() function. The intermediate value of the hash calculation held by the R_RSIP_SHA_Suspend() function is set in p_ctrl and used in the subsequent R_RSIP_SHA_Update() and R_RSIP_SHA_Finish() functions.

Reentrant

Not supported.

4.2.6.7 R_RSIP_HMAC_Compute

Format

```
#include "r_rsip_protected_rx_if.h"
fsp_err_t R_RSIP_HMAC_Init(
    rsip_ctrl_t * const p_ctrl,
    rsip_wrapped_key_t const * const p_wrapped_key,
    uint8_t const * const p_message,
    uint32_t const message_length,
    uint8_t * const p_mac
)
```

Parameters

p_ctrl	Input/output	RSIP PM driver management structure
p_wrapped_key	Input	Wrapped key
p_message	Input	Message data area
message_length	Input	Byte length of message data (0 byte to any length of bytes)
p_mac	Output	HMAC data area HMAC-SHA224: 28 byte HMAC-SHA256: 32 byte

Return Values

FSP_SUCCESS	Normal termination
FSP_ERR_ASSERTION	Null pointer argument
FSP_ERR_NOT_OPEN	RSIP PM driver not open
FSP_ERR_INVALID_STATE	Invalid internal state of the RSIP PM driver
FSP_ERR_NOT_ENABLED	Unsupported key type specified for p_wrapped_key
FSP_ERR_CRYPTO_RSIP_KEY_SET_FAIL	Occurrence of error in verification of p_wrapped_key
FSP_ERR_CRYPTO_RSIP_RESOURCE_CONFLICT	Occurrence of resource conflict because a hardware resource needed by the processing routine was in use by another processing routine
FSP_ERR_CRYPTO_RSIP_FATAL	Software corruption detected

Description

This API performs preparations for execution of an HMAC calculation.

For p_wrapped_key, specify the wrapped key to be used for HMAC generation and verification.

For p_ctrl, specify the pointer to the management structure used in the R_RSIP_Open() function. For p_message, input the message to be hash-calculated. For message_length, specify the length of the message to be input in p_message.

It outputs the HMAC value of the message to p_mac.

Reentrant

Not supported.

4.2.6.8 R_RSIP_HMAC_Verify

Format

```
#include "r_rsip_protected_rx_if.h"
fsp_err_t R_RSIP_HMAC_Init(
    rsip_ctrl_t * const p_ctrl,
    rsip_wrapped_key_t const * const p_wrapped_key,
    uint8_t const * const p_message,
    uint32_t const message_length,
    uint8_t const * const p_mac,
    uint32_t const mac_length
)
```

Parameters

p_ctrl	Input/output	RSIP PM driver management structure
p_wrapped_key	Input	Wrapped key
p_message	Input	Message data area
message_length	Input	Byte length of message data (0 byte to any length of bytes)
p_mac	Input	HMAC data area to be verified
mac_length	Input	Byte length of HMAC data to be verified

Return Values

FSP_SUCCESS	Normal termination
FSP_ERR_ASSERTION	Null pointer argument
FSP_ERR_NOT_OPEN	RSIP PM driver not open
FSP_ERR_INVALID_STATE	Invalid internal state of the RSIP PM driver
FSP_ERR_NOT_ENABLED	Unsupported key type specified for p_wrapped_key
FSP_ERR_CRYPTO_RSIP_KEY_SET_FAIL	Occurrence of error in verification of p_wrapped_key
FSP_ERR_CRYPTO_RSIP_RESOURCE_CONFLICT	Occurrence of resource conflict because a hardware resource needed by the processing routine was in use by another processing routine
FSP_ERR_CRYPTO_RSIP_FATAL	Software corruption detected

Description

This API performs preparations for execution of an HMAC calculation.

For p_wrapped_key, specify the wrapped key to be used for HMAC generation and verification.

For p_ctrl, specify the pointer to the management structure used in the R_RSIP_Open() function. For p_message, input the message to be hash-calculated. For message_length, specify the length of the message to be input in p_message. This function verifies the MAC value input in p_mac as a valid MAC value for the number of bytes specified in mac_length.

Reentrant

Not supported.

4.2.6.9 R_RSIP_HMAC_Init

Format

```
#include "r_rsip_protected_rx_if.h"
fsp_err_t R_RSIP_HMAC_Init(
    rsip_ctrl_t * const p_ctrl,
    rsip_wrapped_key_t const * const p_wrapped_key
)
```

Parameters

p_ctrl	Input/output	RSIP PM driver management structure
p_wrapped_key	Input	Wrapped key

Return Values

FSP_SUCCESS	Normal termination
FSP_ERR_ASSERTION	Null pointer argument
FSP_ERR_NOT_OPEN	RSIP PM driver not open
FSP_ERR_INVALID_STATE	Invalid internal state of the RSIP PM driver
FSP_ERR_NOT_ENABLED	Unsupported key type specified for p_wrapped_key
FSP_ERR_CRYPTO_RSIP_KEY_SET_FAIL	Occurrence of error in verification of p_wrapped_key
FSP_ERR_CRYPTO_RSIP_RESOURCE_CONFLICT	Occurrence of resource conflict because a hardware resource needed by the processing routine was in use by another processing routine
FSP_ERR_CRYPTO_RSIP_FATAL	Software corruption detected

Description

This API performs preparations for execution of an HMAC calculation.

For p_wrapped_key, specify the wrapped key to be used for HMAC generation and verification.

For p_ctrl, specify the pointer to the management structure used in the R_RSIP_Open() function. The preparation result for HMAC calculation execution is stored in p_ctrl. The value output to p_ctrl is used by the R_RSIP_HMAC_Update(), R_RSIP_HMAC_SignFinish(), and R_RSIP_HMAC_VerifyFinish() functions.

Reentrant

Not supported.

4.2.6.10 R_RSIP_HMAC_Update

Format

```
#include "r_rsip_protected_rx_if.h"
fsp_err_t R_RSIP_HMAC_Update(
    rsip_ctrl_t * const p_ctrl,
    uint8_t const * const p_message,
    uint32_t const message_length
)
```

Parameters

p_ctrl	Input/output	RSIP PM driver management structure
p_message	Input	Message data area
message_length	Input	Byte length of message data (0 byte to any length of bytes)

Return Values

FSP_SUCCESS	Normal termination
FSP_ERR_ASSERTION	Null pointer argument
FSP_ERR_NOT_OPEN	RSIP PM driver not open
FSP_ERR_INVALID_STATE	Invalid internal state of the RSIP PM driver
FSP_ERR_NOT_ENABLED	Unsupported key type specified for p_wrapped_key which is input in R_RSIP_HMAC_Init
FSP_ERR_CRYPTO_RSIP_KEY_SET_FAIL	Occurrence of error in verification of p_wrapped_key which is input in R_RSIP_HMAC_Init
FSP_ERR_CRYPTO_RSIP_RESOURCE_CONFLICT	Occurrence of resource conflict because a hardware resource needed by the processing routine was in use by another processing routine
FSP_ERR_CRYPTO_RSIP_FATAL	Software corruption detected

Description

This API performs an HMAC calculation.

For p_message, input the message to be hash-calculated. For message_length, specify the length of the message to be input in p_message.

For p_ctrl, specify p_ctrl used in the R_RSIP_HMAC_Init() function.

Reentrant

Not supported.

4.2.6.11 R_RSIP_HMAC_SignFinish

Format

```
#include "r_rsip_protected_rx_if.h"
fsp_err_t R_RSIP_HMAC_SignFinish(
    rsip_ctrl_t *const p_ctrl,
    uint8_t * const p_mac
)
```

Parameters

p_ctrl	Input/output	RSIP PM driver management structure
p_mac	Output	HMAC data area HMAC-SHA224: 28 byte HMAC-SHA256: 32 byte

Return Values

FSP_SUCCESS	Normal termination
FSP_ERR_ASSERTION	Null pointer argument
FSP_ERR_NOT_OPEN	RSIP PM driver not open
FSP_ERR_INVALID_STATE	Invalid internal state of the RSIP PM driver
FSP_ERR_NOT_ENABLED	Unsupported key type specified for p_wrapped_key which is input in R_RSIP_HMAC_Init
FSP_ERR_CRYPTO_RSIP_KEY_SET_FAIL	Occurrence of error in verification of p_wrapped_key which is input in R_RSIP_HMAC_Init
FSP_ERR_CRYPTO_RSIP_RESOURCE_CONFLICT	Occurrence of resource conflict because a hardware resource needed by the processing routine was in use by another processing routine
FSP_ERR_CRYPTO_RSIP_FATAL	Software corruption detected

Description

This API generates an HMAC.

It outputs the HMAC value of the message input by the R_RSIP_HMAC_Update() function to p_mac.

Use R_RSIP_HMAC_VerifyFinish() to perform HMAC verification.

For p_ctrl, specify p_ctrl used in the R_RSIP_HMAC_Init() and R_RSIP_HMAC_Update() functions.

Reentrant

Not supported.

4.2.6.12 R_RSIP_HMAC_VerifyFinish

Format

```
#include "r_rsip_protected_rx_if.h"
fsp_err_t R_RSIP_HMAC_VerifyFinish(
    rsip_ctrl_t * const p_ctrl,
    uint8_t const * const p_mac,
    uint32_t const mac_length
)
```

Parameters

p_ctrl	Input/output	RSIP PM driver management structure
p_mac	Input	HMAC data area to be verified
mac_length	Input	Byte length of HMAC data to be verified

Return Values

FSP_SUCCESS	Normal termination
FSP_ERR_ASSERTION	Null pointer argument
FSP_ERR_NOT_OPEN	RSIP PM driver not open
FSP_ERR_INVALID_STATE	Invalid internal state of the RSIP PM driver
FSP_ERR_INVALID_SIZE	Invalid mac_length
FSP_ERR_NOT_ENABLED	Unsupported key type specified for p_wrapped_key which is input in R_RSIP_HMAC_Init
FSP_ERR_CRYPTO_RSIP_KEY_SET_FAIL	Occurrence of error in verification of p_wrapped_key which is input in R_RSIP_HMAC_Init
FSP_ERR_CRYPTO_RSIP_FAIL	MAC verification error
FSP_ERR_CRYPTO_RSIP_RESOURCE_CONFLICT	Occurrence of resource conflict because a hardware resource needed by the processing routine was in use by another processing routine
FSP_ERR_CRYPTO_RSIP_FATAL	Software corruption detected

Description

This API performs HMAC verification.

It verifies the HMAC value of the message input in the R_RSIP_HMAC_Update() function against the HMAC value input in p_mac. It verifies the MAC value input in p_mac as a valid MAC value for the number of bytes specified in mac_length.

R_RSIP_HMAC_SignFinish() is used for HMAC generation.

For p_ctrl, specify p_ctrl used in the R_RSIP_HMAC_Init() and R_RSIP_HMAC_Update() functions.

Reentrant

Not supported.

4.2.6.13 R_RSIP_HMAC_Suspend

Format

```
#include "r_rsip_protected_rx_if.h"
fsp_err_t R_RSIP_SHA_Suspend(
    rsip_ctrl_t * const p_ctrl,
    rsip_hmac_handle_t * const p_handle
)
```

Parameters

p_ctrl	Input/output	RSIP PM driver management structure
p_handle	Output	Control information for HMAC calculation

Return Values

FSP_SUCCESS	Normal termination
FSP_ERR_ASSERTION	Null pointer argument
FSP_ERR_NOT_OPEN	RSIP PM driver not open
FSP_ERR_INVALID_STATE	Invalid internal state of the RSIP PM driver
FSP_ERR_CRYPTO_RSIP_FATAL	Software corruption detected

Description

After execution with the R_RSIP_HMAC_Init() and R_RSIP_HMAC_Update() functions, this API suspends an HMAC calculation midway so that other multi-part or single-part operations can be performed. It outputs the intermediate value of the suspended HMAC calculation to p_handle.

If R_RSIP_HMAC_SignFinish() or R_RSIP_HMAC_VerifyFinish() function is called after R_RSIP_HMAC_Suspend() function is called, this API outputs the HMAC value of the message input to R_RSIP_HMAC_Update() function up to that point.

To resume the HMAC calculation, call the R_RSIP_HMAC_Suspend() function.

For p_ctrl, specify p_ctrl used in the R_RSIP_HMAC_Init() and R_RSIP_HMAC_Update() functions.

Reentrant

Not supported.

4.2.6.14 R_RSIP_HMAC_Resume

Format

```
#include "r_rsip_protected_rx_if.h"
fsp_err_t R_RSIP_SHA_Suspend(
    rsip_ctrl_t * const p_ctrl,
    rsip_hmac_handle_t * const p_handle
)
```

Parameters

p_ctrl	Input/output	RSIP PM driver management structure
p_handle	Input	Control information for HMAC calculation

Return Values

FSP_SUCCESS	Normal termination
FSP_ERR_ASSERTION	Null pointer argument
FSP_ERR_NOT_OPEN	RSIP PM driver not open
FSP_ERR_INVALID_STATE	Invalid internal state of the RSIP PM driver

Description

This API resumes the HMAC calculation suspended by the R_RSIP_HMAC_Suspend() function. For p_handle, input p_handle output by the R_RSIP_HMAC_Suspend() function.

For p_ctrl, specify the pointer to the management structure used in the R_RSIP_Open() function. The intermediate value of the HMAC calculation held by the R_RSIP_HMAC_Suspend() function is set in p_ctrl and used in the subsequent R_RSIP_HMAC_Update() and R_RSIP_HMAC_SignFinish() or R_RSIP_HMAC_VerifyFinish() functions.

Reentrant

Not supported.

4.2.7 Key Derivation Function (KDF)

4.2.7.1 R_RSIP_KDF_SHA_Init

Format

```
#include "r_rsip_protected_rx_if.h"
fsp_err_t R_RSIP_KDF_SHA_Init(
    rsip_ctrl_t * const p_ctrl,
    rsip_hash_type_t const hash_type
)
```

Parameters

p_ctrl	Input/output	RSIP PM driver management structure
hash_type	Input	Hash type

Return Values

FSP_SUCCESS	Normal termination
FSP_ERR_ASSERTION	Null pointer argument
FSP_ERR_NOT_OPEN	RSIP PM driver not open
FSP_ERR_INVALID_STATE	Invalid internal state of the RSIP PM driver
FSP_ERR_NOT_ENABLED	Unsupported key type in this functionality (depending on the setting)

Description

This API performs preparations for execution of a hash calculation for KDF.

For hash_type, specify the hash calculation mode to be executed.

For p_ctrl, specify the pointer to the management structure used in the R_RSIP_Open() function. The preparation result for hash calculation execution is stored in p_ctrl. The value output to p_ctrl is used by the R_RSIP_KDF_SHA_ECDHSecretUpdate(), R_RSIP_KDF_SHA_Update() and R_RSIP_KDF_SHA_Finish() functions.

Reentrant

Not supported.

4.2.7.2 R_RSIP_KDF_SHA_ECDHSecretUpdate

Format

```
#include "r_rsip_protected_rx_if.h"
fsp_err_t R_RSIP_KDF_SHA_ECDHSecretUpdate(
    rsip_ctrl_t * const p_ctrl,
    rsip_wrapped_key_t const * const p_wrapped_secret
)
```

Parameters

p_ctrl	Input/output	RSIP PM driver management structure
p_wrapped_secret	Input	Wrapped secret

Return Values

FSP_SUCCESS	Normal termination
FSP_ERR_ASSERTION	Null pointer argument
FSP_ERR_NOT_OPEN	RSIP PM driver not open
FSP_ERR_INVALID_STATE	Invalid internal state of the RSIP PM driver
FSP_ERR_NOT_ENABLED	Unsupported key type in this functionality (depending on the setting)
FSP_ERR_CRYPTO_RSIP_FAIL	Invalid input parameter
FSP_ERR_CRYPTO_RSIP_RESOURCE_CONFLICT	Occurrence of resource conflict because a hardware resource needed by the processing routine was in use by another processing routine
FSP_ERR_CRYPTO_RSIP_FATAL	Software corruption or hardware failure detected

Description

This API inputs a wrapped ECDH secret for KDF.

For p_wrapped_secret, input the wrapped secret

For p_ctrl, specify p_ctrl used in the R_RSIP_KDF_SHA_Init() function.

Reentrant

Not supported.

4.2.7.3 R_RSIP_KDF_SHA_Update

Format

```
#include "r_rsip_protected_rx_if.h"
fsp_err_t R_RSIP_KDF_SHA_Update(
    rsip_ctrl_t * const p_ctrl,
    uint8_t const * const p_message,
    uint32_t const message_length
)
```

Parameters

p_ctrl	Input/output	RSIP PM driver management structure
p_message	Input	Message data area
message_length	Input	Byte length of message data (0 byte to any length of bytes)

Return Values

FSP_SUCCESS	Normal termination
FSP_ERR_ASSERTION	Null pointer argument
FSP_ERR_NOT_OPEN	RSIP PM driver not open
FSP_ERR_INVALID_STATE	Invalid internal state of the RSIP PM driver
FSP_ERR_INVALID_SIZE	Invalid input size
FSP_ERR_CRYPTO_RSIP_FATAL	Software corruption or hardware failure detected

Description

This API performs a hash calculation for KDF.

For p_message, input the message to be hash-calculated. For message_length, specify the length of the message to be input in p_message.

For p_ctrl, specify p_ctrl used in the R_RSIP_KDF_SHA_Init() function.

Reentrant

Not supported.

4.2.7.4 R_RSIP_KDF_SHA_Finish

Format

```
#include "r_rsip_protected_rx_if.h"
fsp_err_t R_RSIP_RFC3394_KeyUnwrap(
    rsip_ctrl_t * const p_ctrl,
    rsip_wrapped_key_t * const p_wrapped_dkm
)
```

Parameters

p_ctrl	Input/output	RSIP PM driver management structure
p_wrapped_dkm	Output	Wrapped DKM (Derived Keying Material) (64 byte)

Return Values

FSP_SUCCESS	Normal termination
FSP_ERR_ASSERTION	Null pointer argument
FSP_ERR_NOT_OPEN	RSIP PM driver not open
FSP_ERR_INVALID_STATE	Invalid internal state of the RSIP PM driver
FSP_ERR_CRYPTO_RSIP_FAIL	Invalid input parameter
FSP_ERR_CRYPTO_RSIP_RESOURCE_CONFLICT	Occurrence of resource conflict because a hardware resource needed by the processing routine was in use by another processing routine
FSP_ERR_CRYPTO_RSIP_FATAL	Software corruption or hardware failure detected

Description

This API outputs the result of a hash calculation for KDF.
It outputs the wrapped DKM to p_wrapped_dkm.

For p_ctrl, specify p_ctrl used in the R_RSIP_KDF_SHA_Init(), R_RSIP_KDF_SHA_ECDHSecretUpdate() and R_RSIP_KDF_SHA_Update() functions.

Reentrant

Not supported.

4.2.7.5 R_RSIP_KDF_SHA_Suspend

Format

```
#include "r_rsip_protected_rx_if.h"
fsp_err_t R_RSIP_KDF_SHA_Suspend(
    rsip_ctrl_t * const p_ctrl,
    rsip_kdf_sha_handle_t * const p_handle
)
```

Parameters

p_ctrl	Input/output	RSIP PM driver management structure
p_handle	Output	KDF SHA control block (232 byte)

Return Values

FSP_SUCCESS	Normal termination
FSP_ERR_ASSERTION	Null pointer argument
FSP_ERR_NOT_OPEN	RSIP PM driver not open
FSP_ERR_INVALID_STATE	Invalid internal state of the RSIP PM driver

Description

After execution with the R_RSIP_KDF_SHA_Init(), R_RSIP_KDF_SHA_ECDHSecretUpdate() and R_RSIP_KDF_SHA_Update() functions, this API suspends a hash calculation for KDF midway so that other multi-part or single-part operations can be performed. It outputs the intermediate value of the suspended hash calculation to p_handle.

To resume the hash calculation for KDF, call the R_RSIP_KDF_SHA_Resume() function.

For p_ctrl, specify p_ctrl used in the R_RSIP_KDF_SHA_Init(), R_RSIP_KDF_SHA_ECDHSecretUpdate() and R_RSIP_KDF_SHA_Update() functions.

Reentrant

Not supported.

4.2.7.6 R_RSIP_KDF_SHA_Resume

Format

```
#include "r_rsip_protected_rx_if.h"
fsp_err_t R_RSIP_RFC3394_KeyUnwrap(
    rsip_ctrl_t * const p_ctrl,
    rsip_kdf_sha_handle_t const * const p_handle
)
```

Parameters

p_ctrl	Input/output	RSIP PM driver management structure
p_handle	Input	KDF SHA control block

Return Values

FSP_SUCCESS	Normal termination
FSP_ERR_ASSERTION	Null pointer argument
FSP_ERR_NOT_OPEN	RSIP PM driver not open
FSP_ERR_INVALID_STATE	Invalid internal state of the RSIP PM driver

Description

This API resumes the hash calculation for KDF suspended by the R_RSIP_KDF_SHA_Suspend() function. For p_handle, input p_handle output by the R_RSIP_KDF_SHA_Suspend() function.

For p_ctrl, specify the pointer to the management structure used in the R_RSIP_Open() function. The intermediate value of the hash calculation for KDF held by the R_RSIP_KDF_SHA_Suspend() function is set in p_ctrl and used in the subsequent R_RSIP_KDF_SHA_Update() and R_RSIP_KDF_SHA_Finish() functions.

Reentrant

Not supported.

4.2.7.7 R_RSIP_KDF_DKMConcatenate

Format

```
#include "r_rsip_protected_rx_if.h"
fsp_err_t R_RSIP_KDF_DKMConcatenate(
    rsip_wrapped_dkm_t * const p_wrapped_dkm1,
    rsip_wrapped_dkm_t const * const p_wrapped_dkm2,
    uint32_t const wrapped_dkm1_buffer_length
)
```

Parameters

p_wrapped_dkm1	Input/output	First DKM (DKM1)
p_wrapped_dkm2	Input	Second DKM (DKM2)
dkm1_buffer_length	Input	Length of wrapped_dkm1 buffer

Return Values

FSP_SUCCESS	Normal termination
FSP_ERR_ASSERTION	Null pointer argument
FSP_ERR_INVALID_SIZE	Invalid input size
FSP_ERR_CRYPTO_RSIP_FAIL	Invalid input parameter

Description

This API concatenates two wrapped DKMs.

For p_wrapped_dkm1, input the first DKM. For p_wrapped_dkm2, input the second DKM. For dkm1_buffer_length, specify the buffer size of p_wrapped_dkm1. The concatenated DKM is output to p_wrapped_dkm1.

Reentrant

Not supported.

4.2.7.8 R_RSIP_KDF_DerivedKeyImport

Format

```
#include "r_rsip_protected_rx_if.h"
fsp_err_t R_RSIP_RFC3394_KeyUnwrap(
    rsip_ctrl_t * const p_ctrl,
    rsip_wrapped_dkm_t const * const p_wrapped_dkm,
    uint32_t const position,
    rsip_wrapped_key_t * const p_wrapped_key
)
```

Parameters

p_ctrl	Input/output	RSIP PM driver management structure
p_wrapped_dkm	Input	Wrapped data of concatenated DKM
position	Input	Start position of output key value in concatenated DKM
p_wrapped_key	Output	Wrapped key (The wrapped key data size which is defined for each key type in Table 2-10 is output to p_value field)

Return Values

FSP_SUCCESS	Normal termination
FSP_ERR_ASSERTION	Null pointer argument
FSP_ERR_NOT_OPEN	RSIP PM driver not open
FSP_ERR_INVALID_STATE	Invalid internal state of the RSIP PM driver
FSP_ERR_NOT_ENABLED	Unsupported key type in this functionality (depending on the setting)
FSP_ERR_CRYPTO_RSIP_FAIL	Invalid input parameter
FSP_ERR_CRYPTO_RSIP_RESOURCE_CONFLICT	Occurrence of resource conflict because a hardware resource needed by the processing routine was in use by another processing routine
FSP_ERR_CRYPTO_RSIP_FATAL	Software corruption or hardware failure detected

Description

This API generates a wrapped key from DKM.

For p_wrapped_dkm, specify the wrapped data of concatenated DKM. For position, specify the start position of output key value in concatenated DKM. Binary data of the wrapped key is output to p_value field of p_wrapped_key.

For type field of p_wrapped_key, specify the value defined in Table 2-4. The byte size of the output wrapped key is the size of each value defined in Table 2-10. Allocate an enough area according to the algorithm specified in type and specify the address to p_value.

For p_ctrl, specify p_ctrl used in the R_RSIP_KDF_SHA_Init(), R_RSIP_KDF_SHA_ECDHSecretUpdate(), R_RSIP_KDF_SHA_Update() and R_RSIP_KDF_SHA_Finish() functions.

Reentrant

Not supported.

4.2.7.9 R_RSIP_KDF_DerivedIVWrap

Format

```
#include "r_rsip_protected_rx_if.h"
fsp_err_t R_RSIP_RFC3394_KeyUnwrap(
    rsip_ctrl_t * const p_ctrl,
    rsip_wrapped_dkm_t const * const p_wrapped_dkm,
    rsip_initial_vector_type_t const initial_vector_type,
    uint32_t const position,
    uint8_t const * const p_tls_sequence_num,
    uint8_t * const p_wrapped_initial_vector
)
```

Parameters

p_ctrl	Input/output	RSIP PM driver management structure
p_wrapped_dkm	Input	Wrapped data of concatenated DKM
initial_vector_type	Input	Initial vector type
position	Input	Start position of output key value in concatenated DKM
p_tls_sequence_num	Input	Used only for TLS, must be set to NULL.
p_wrapped_key	Output	Wrapped initial vector (36 byte)

Return Values

FSP_SUCCESS	Normal termination
FSP_ERR_ASSERTION	Null pointer argument
FSP_ERR_NOT_OPEN	RSIP PM driver not open
FSP_ERR_INVALID_STATE	Invalid internal state of the RSIP PM driver
FSP_ERR_NOT_ENABLED	Unsupported key type in this functionality (depending on the setting)
FSP_ERR_CRYPTO_RSIP_FAIL	Invalid input parameter
FSP_ERR_CRYPTO_RSIP_RESOURCE_CONFLICT	Occurrence of resource conflict because a hardware resource needed by the processing routine was in use by another processing routine
FSP_ERR_CRYPTO_RSIP_FATAL	Software corruption or hardware failure detected

Description

This API generates a wrapped initial vector from DKM.

For p_wrapped_dkm, specify the wrapped data of concatenated DKM. For initial_vector_type, specify the type of initial vector. For position, specify the start position of output key value in concatenated DKM. For p_tls_sequence_num, specify NULL. Binary data of the wrapped initial vector is output to p_wrapped_initial_vector.

For p_ctrl, specify p_ctrl used in the R_RSIP_KDF_SHA_Init(), R_RSIP_KDF_SHA_ECDHSecretUpdate(), R_RSIP_KDF_SHA_Update() and R_RSIP_KDF_SHA_Finish() functions.

Reentrant

Not supported.

4.2.8 Key Wrap

4.2.8.1 R_RSIP_RFC3394_KeyWrap

Format

```
#include "r_rsip_protected_rx_if.h"
fsp_err_t R_RSIP_RFC3394_KeyWrap(
    rsip_ctrl_t * const p_ctrl,
    rsip_wrapped_key_t const * const p_wrapped_kek,
    rsip_wrapped_key_t const * const p_wrapped_target_key,
    uint8_t * const p_rfc3394_wrapped_target_key
)
```

Parameters

p_ctrl	Input/output	RSIP PM driver management structure
p_wrapped_kek	Input	Key used for wrapping
p_wrapped_target_key	Input	Key to be wrapped
p_rfc3394_wrapped_target_key	Output	Wrapped key AES128 key: 24 byte AES256 key: 40 byte

Return Values

FSP_SUCCESS	Normal termination
FSP_ERR_ASSERTION	Null pointer argument
FSP_ERR_NOT_OPEN	RSIP PM driver not open
FSP_ERR_INVALID_STATE	Invalid internal state of the RSIP PM driver
FSP_ERR_NOT_ENABLED	Unsupported key type in this functionality (depending on the setting)
FSP_ERR_INVALID_ARGUMENT	Invalid input key type or mode
FSP_ERR_CRYPTO_RSIP_FAIL	Invalid input parameter
FSP_ERR_CRYPTO_RSIP_KEY_SET_FAIL	Invalid key input
FSP_ERR_CRYPTO_RSIP_RESOURCE_CONFLICT	Occurrence of resource conflict because a hardware resource needed by the processing routine was in use by another processing routine
FSP_ERR_CRYPTO_RSIP_FATAL	Software corruption or hardware failure detected

Description

This API wraps a key with an RFC3394-compliant algorithm.

It wraps p_wrapped_target_key with the key specified in p_wrapped_kek. The wrapped key is output to p_rfc3394_wrapped_target_key.

For p_ctrl, specify the pointer to the management structure used in the R_RSIP_Open() function.

Reentrant

Not supported.

4.2.8.2 R_RSIP_RFC3394_KeyUnwrap

Format

```
#include "r_rsip_protected_rx_if.h"
fsp_err_t R_RSIP_RFC3394_KeyUnwrap(
    rsip_ctrl_t * const p_ctrl,
    rsip_wrapped_key_t const * const p_wrapped_kek,
    rsip_key_type_t const key_type,
    uint8_t const * const p_rfc3394_wrapped_target_key,
    rsip_wrapped_key_t * const p_wrapped_target_key
)
```

Parameters

p_ctrl	Input/output	RSIP PM driver management structure
p_wrapped_kek	Input	Key used for unwrapping
key_type	Input	Type of output wrapped key
p_rfc3394_wrapped_target_key	Input	Wrapped key
p_wrapped_target_key	Input/output	Unwrapped key (The wrapped key data size which is defined for each key type in Table 2-10 is output to p_value field)

Return Values

FSP_SUCCESS	Normal termination
FSP_ERR_ASSERTION	Null pointer argument
FSP_ERR_NOT_OPEN	RSIP PM driver not open
FSP_ERR_INVALID_STATE	Invalid internal state of the RSIP PM driver
FSP_ERR_NOT_ENABLED	Unsupported key type in this functionality (depending on the setting)
FSP_ERR_INVALID_ARGUMENT	Invalid input key type or mode
FSP_ERR_CRYPTO_RSIP_FAIL	Invalid input parameter
FSP_ERR_CRYPTO_RSIP_KEY_SET_FAIL	Invalid key input
FSP_ERR_CRYPTO_RSIP_RESOURCE_CONFLICT	Occurrence of resource conflict because a hardware resource needed by the processing routine was in use by another processing routine
FSP_ERR_CRYPTO_RSIP_FATAL	Software corruption or hardware failure detected

Description

This API unwraps a key with an RFC3394-compliant algorithm.

It unwraps p_rfc3394_wrapped_target_key with the key specified in p_wrapped_kek. The unwrapped key is output to p_wrapped_target_key in the wrapped key format. For key_type, specify the type of key to be unwrapped.

For p_ctrl, specify the pointer to the management structure used in the R_RSIP_Open() function.

Reentrant

Not supported.

4.2.9 Firmware Update/Secure Boot

4.2.9.1 R_RSIP_FWUP_StartUpdateFirmware

Format

```
#include "r_rsip_protected_rx_if.h"
fsp_err_t R_RSIP_FWUP_StartUpdateFirmware(
    rsip_ctrl_t * const p_ctrl
)
```

Parameters

p_ctrl	Input/output	RSIP PM driver management structure
--------	--------------	-------------------------------------

Return Values

FSP_SUCCESS	Normal termination
FSP_ERR_ASSERTION	Null pointer argument
FSP_ERR_NOT_OPEN	RSIP PM driver not open
FSP_ERR_INVALID_STATE	Invalid internal state of the RSIP PM driver
FSP_ERR_CRYPTO_RSIP_RESOURCE_CONFLICT	Occurrence of resource conflict because a hardware resource needed by the processing routine was in use by another processing routine

Description

This API transitions the RSIP PM driver to the Firmware Update state.

For p_ctrl, specify the pointer to the management structure used in the R_RSIP_Open() function.

Reentrant

Not supported.

4.2.9.2 R_RSIP_FWUP_MAC_Sign_Init

Format

```
#include "r_rsip_protected_rx_if.h"
fsp_err_t R_RSIP_FWUP_MAC_Sign_Init(
    rsip_ctrl_t * const p_ctrl,
    rsip_wrapped_key_t const * const p_wrapped_key_encryption_key,
    uint8_t const * const p_encrypted_image_encryption_key,
    uint8_t const * const p_initial_vector,
    uint32_t const firmware_size
)
```

Parameters

p_ctrl	Input/output	RSIP PM driver management structure
p_wrapped_key_encryption_key	Input	Wrapped key of the key used for wrapping the image encryption key
p_encrypted_image_encryption_key	Input	Image encryption key wrapped with a key encryption key
p_initial_vector	Input	Initialization vector used in firmware encryption
firmware_size	Input	Total byte length of encrypted firmware (0 byte to any length of bytes)

Return Values

FSP_SUCCESS	Normal termination
FSP_ERR_ASSERTION	Null pointer argument
FSP_ERR_NOT_OPEN	RSIP PM driver not open
FSP_ERR_INVALID_STATE	Invalid internal state of the RSIP PM driver
FSP_ERR_CRYPTO_RSIP_RESOURCE_CONFLICT	Occurrence of resource conflict because a hardware resource needed by the processing routine was in use by another processing routine
FSP_ERR_CRYPTO_RSIP_KEY_SET_FAIL	Invalid key input

Description

This API decrypts encrypted firmware, performs MAC verification, and performs preparations for generation of a MAC for plaintext firmware.

For p_wrapped_key_encryption_key, input a wrapped key of the key encryption key used for wrapping the key used for encrypting firmware (image encryption key). For p_encrypted_image_encryption_key, input the image encryption key wrapped with the key encryption key. For p_initial_vector, input the initialization vector used for firmware encryption. For firmware_size, input the byte size of the encrypted firmware to be decrypted.

For p_ctrl, specify the pointer to the management structure used in the R_RSIP_Open() function. The result of the R_RSIP_FWUP_MAC_Sign_Init() function execution is stored in p_ctrl. The value output to p_ctrl is used by the R_RSIP_FWUP_MAC_Sign_Update() and R_RSIP_FWUP_MAC_Sign_Finish() functions.

Reentrant

Not supported.

4.2.9.3 R_RSIP_FWUP_MAC_Sign_Update

Format

```
#include "r_rsip_protected_rx_if.h"
fsp_err_t R_RSIP_FWUP_MAC_Sign_Update(
    rsip_ctrl_t * const p_ctrl,
    uint8_t const * const p_input,
    uint8_t * const p_output,
    uint32_t const input_length
)
```

Parameters

p_ctrl	Input/output	RSIP PM driver management structure
p_input	Input	Encrypted firmware
p_output	Output	Plaintext firmware (the size is same as the value of length)
input_length	Input	Byte length of input encrypted firmware (0 byte to any length of bytes, must be a multiple of 16)

Return Values

FSP_SUCCESS	Normal termination
FSP_ERR_ASSERTION	Null pointer argument
FSP_ERR_NOT_OPEN	RSIP PM driver not open
FSP_ERR_INVALID_STATE	Invalid internal state of the RSIP PM driver
FSP_ERR_INVALID_ARGUMENT	Invalid input data

Description

This API decrypts the encrypted firmware input to p_input for the size specified by length, and outputs the plaintext firmware to p_output.

Use R_RSIP_FWUP_MAC_Sign_Update() for decryption if the decryption process needs to be performed more than once, for example, because the encrypted firmware exists in multiple areas. This API, however, decrypts the encrypted firmware including the last 16 bytes using R_RSIP_FWUP_MAC_Sign_Finish(). If there is no need to perform the decryption process more than once, do not use R_RSIP_FWUP_MAC_Sign_Update(), but use R_RSIP_FWUP_MAC_Sign_Finish().

For p_ctrl, specify p_ctrl used in the R_RSIP_FWUP_MAC_Sign_Init() function.

Reentrant

Not supported.

4.2.9.4 R_RSIP_FWUP_MAC_Sign_Finish

Format

```
#include "r_rsip_protected_rx_if.h"
fsp_err_t R_RSIP_FWUP_MAC_Sign_Finish(
    rsip_ctrl_t * const p_ctrl,
    uint8_t const * const p_input,
    uint8_t const * const p_input_mac,
    uint32_t const length,
    uint8_t * const p_output,
    uint8_t * const p_output_mac
)
```

Parameters

p_ctrl	Input/output	RSIP PM driver management structure
p_input	Input	Input firmware area
p_input_mac	Input	MAC value of input firmware (16 bytes)
p_output	Output	Output firmware area
p_output_mac	Output	MAC value of output firmware (16 bytes)
input_length	Input	Byte length of input firmware. (0 byte to any length of bytes, must be a multiple of 16)

Return Values

FSP_SUCCESS	Normal termination
FSP_ERR_ASSERTION	Null pointer argument
FSP_ERR_NOT_OPEN	RSIP PM driver not open
FSP_ERR_INVALID_STATE	Invalid internal state of the RSIP PM driver
FSP_ERR_CRYPTO_RSIP_FAIL	Abnormal termination

Description

This API decrypts the encrypted firmware input to p_input for the size specified by length, and outputs the plaintext firmware to p_output. It then performs MAC verification of the encrypted firmware input to p_input_mac. If the MAC verification of the encrypted firmware is successful, the API generates a MAC value of the plaintext firmware and outputs it to p_output_mac.

For p_ctrl, specify p_ctrl used in the R_RSIP_FWUP_MAC_Sign_Init() and R_RSIP_FWUP_MAC_Sign_Update() functions.

Reentrant

Not supported.

4.2.9.5 R_RSIP_SB_MAC_Verify_Init

Format

```
#include "r_rsip_protected_rx_if.h"
fsp_err_t R_RSIP_SB_MAC_Verify_Init(
    rsip_ctrl_t * const p_ctrl
)
```

Parameters

p_ctrl	Input/output RSIP PM driver management structure
--------	--

Return Values

FSP_SUCCESS	Normal termination
FSP_ERR_ASSERTION	Null pointer argument
FSP_ERR_NOT_OPEN	RSIP PM driver not open
FSP_ERR_INVALID_STATE	Invalid internal state of the RSIP PM driver

Description

This API performs preparations for MAC verification for plaintext firmware.

For p_ctrl, specify the pointer to the management structure used in the R_RSIP_Open() function. The result of the R_RSIP_SB_MAC_Verify_Init() function execution is stored in p_ctrl. The value output to p_ctrl is used by the R_RSIP_SB_MAC_Verify_Update() and R_RSIP_SB_MAC_Verify_Finish() functions.

Reentrant

Not supported.

4.2.9.6 R_RSIP_SB_MAC_Verify_Update

Format

```
#include "r_rsip_protected_rx_if.h"
fsp_err_t R_RSIP_SB_MAC_Verify_Update(
    rsip_ctrl_t * const p_ctrl,
    uint8_t const * const p_input,
    uint32_t const input_length
)
```

Parameters

p_ctrl	Input/output	RSIP PM driver management structure
p_input	Input	Input firmware area
input_length	Input	Byte length of input firmware. (0 byte to any length of bytes, must be a multiple of 16)

Return Values

FSP_SUCCESS	Normal termination
FSP_ERR_ASSERTION	Null pointer argument
FSP_ERR_NOT_OPEN	RSIP PM driver not open
FSP_ERR_INVALID_STATE	Invalid internal state of the RSIP PM driver
FSP_ERR_CRYPTO_RSIP_RESOURCE_CONFLICT	Occurrence of resource conflict because a hardware resource needed by the processing routine was in use by another processing routine
FSP_ERR_CRYPTO_RSIP_FAIL	Abnormal termination

Description

This API performs MAC calculation on plaintext firmware which is input to p_input for the size specified by input_length.

Use R_RSIP_SB_MAC_Verify_Update() if the plaintext firmware needs to be input more than once, for example, because the plaintext firmware exists in multiple areas. Input the plaintext firmware including the last 16 bytes by using R_RSIP_SB_MAC_Verify_Finish(). If there is no need to divide plaintext firmware input, use R_RSIP_SB_MAC_Verify_Finish() instead of R_RSIP_SB_MAC_Verify_Update().

For p_ctrl, specify p_ctrl used in the R_RSIP_SB_MAC_Verify_Init() function.

Reentrant

Not supported.

4.2.9.7 R_RSIP_SB_MAC_Verify_Finish

Format

```
#include "r_rsip_protected_rx_if.h"
fsp_err_t R_RSIP_SB_MAC_Verify_Finish(
    rsip_ctrl_t * const p_ctrl,
    uint8_t const * const p_input,
    uint8_t const * const p_mac,
    uint32_t const input_length
)
```

Parameters

p_ctrl	Input/output	RSIP PM driver management structure
p_input	Input	Input firmware area
p_mac	Input	MAC value of input firmware (16 bytes)
input_length	Input	Byte length of input firmware. (0 byte to any length of bytes, must be a multiple of 16)

Return Values

FSP_SUCCESS	Normal termination
FSP_ERR_ASSERTION	Null pointer argument
FSP_ERR_NOT_OPEN	RSIP PM driver not open
FSP_ERR_INVALID_STATE	Invalid internal state of the RSIP PM driver
FSP_ERR_CRYPTO_RSIP_RESOURCE_CONFLICT	Occurrence of resource conflict because a hardware resource needed by the processing routine was in use by another processing routine
FSP_ERR_CRYPTO_RSIP_FAIL	Abnormal termination

Description

This API performs MAC verification for input plaintext firmware.

For p_ctrl, specify the argument used in the R_RSIP_SB_MAC_Verify_Init() and R_RSIP_SB_MAC_Verify_Update() functions.

Reentrant

Not supported.

5. Key Injection and Updating

This section describes how to write encryption keys handled by the RSIP PM driver to nonvolatile memory such as the on-chip flash memory.

5.1 Key Injection

The procedure used to safely inject keys into products as part of the customer's manufacturing process is presented below. Refer to 3.7.1, Key Injection and Updating, for an explanation of the RSIP PM driver's key injection mechanism.

The Renesas Key Wrap Service provided by Renesas and a key injection program running on the RX Family MCU are required for user key injection. Supplementary tools such as Security Key Management Tool can be used to simplify the process.

The demo project accompanying this application note includes a sample key injection program that can be used for reference.

The process for implementing user key injection is as follows:

1. Preparing the key data necessary for user key injection

Use a tool of your choice to prepare a 256-bit UFPK and 128-bit IV to use for wrapping the user key to be injected. The example below uses OpenSSL to generate a UFPK and IV.

```
> openssl rand 32 > ufpk.bin  
> openssl rand 16 > iv.bin
```

Use the Renesas Key Wrap Service (<https://dlm.renesas.com/keywrap>) to generate a W-UFPK by wrapping ufpk.bin using the HRK. For detailed information, refer to the Renesas Key Wrap Service FAQ.

As described in section 3.7.1, which explains the user key wrapping scheme (Figure 3.3, User Key Wrapping Scheme during Key Injection and Key Updating), generate an encrypted key by wrapping the user key using the UFPK (ufpk.bin).

2. Creating a user key injection program

Input the encrypted key, ufpk.bin, and iv.bin generated in step 1 to the key injection API for the cryptographic algorithm being used to generate a user wrapped key, and create a program to write it to nonvolatile memory. Refer to 3.7.1, Key Injection and Updating, for a description of how to use the key injection APIs.

3. Key injection

Run the user key injection program on the RX Family MCU to inject the user key into the flash memory. It is recommended that the key injection data included in the user key injection program be deleted after key injection finishes.

Secure Key Management Tool is available as a supplementary tool for performing steps 1 and 2. Refer to sections 5.3 and 5.4 for details of this tool.

5.2 Key Updating

The procedure used to safely inject or update keys in products in the field is presented below. Refer to 3.7.1, Key Injection and Updating, for an explanation of the RSIP PM driver's key update mechanism.

To be able to inject or update keys in the field, the application program of the customer's product must be provided with key update functionality beforehand.

Key update functionality is implemented using a KUK and key update API. The KUK must be written to the flash memory during the product's manufacturing process using the method described in section 5.1, Key Injection. The process for implementing user key updating is as follows:

1. Creating a user application incorporating key update functionality

Create a user application incorporating key update functionality by making use of a key update API.

Implementing key update functionality requires processing to receive the encrypted key using a communication interface, to convert it to a wrapped key using a key update API, and then to write it to the flash memory.

Refer to 3.7.1, Key Injection and Updating, for an explanation of how to use key update APIs.

2. Injecting the KUK and user key into the device

Refer to section 5.1 and create a user key injection program including the KUK, then inject the KUK and user key into the flash memory. It is recommended that the key injection data included in the user key injection program be deleted after key injection finishes.

3. Programming the user application program incorporating key update functionality to the device

Use a programming method of your choice to program to user application program incorporating key update functionality to the flash memory.

4. Creating the user key data to be used for the update

As described in section 3.7.1, which explains the user key wrapping scheme (Figure 3.3, User Key Wrapping Scheme during Key Injection and Key Updating), generate an encrypted key by wrapping the user key using the KUK.

5. Updating the user key

Pass the encrypted key to the user application program incorporating key update functionality running on the RX Family MCU. The operation of the key update functionality of the user application program implements updating of the key stored in the on-chip flash memory of the RX Family MCU. Depending on the functionality implemented in the user application program, it is also possible to inject a new user key.

Secure Key Management Tool is available as a supplementary tool for performing step 4. Refer to sections 5.3 and 5.4 for details of this tool.

5.3 User Key cryptographic format

The format of the input when encrypting a user key/KUK is shown below: Encrypted Key is generated by key wrapping in the manner shown in 3.6.1.1 Key Wrapping Algorithm.

- AES 128bit key

Input (User Key)

byte	16			
	4	4	4	4
0-15	128 bit AES key			

- AES 256bit key

Input (User Key)

byte	16			
	4	4	4	4
0-31	256 bit AES key			

- ECC secp256r1, brainpoolP256r1, secp256k1 public key

Input (User Key)

byte	16			
	4	4	4	4
0-31	ECC 256 bit public key Qx			
32-63	ECC 256 bit public key Qy			

- ECC secp256r1, brainpoolP256r1, secp256k1 private key

Input (User Key)

byte	16			
	4	4	4	4
0-31	ECC 256 bit private key d			

- HMAC-SHA224 key

Input (User Key)

byte	16			
	4	4	4	4
0-15	HMAC-SHA224 key			
16-31	0 padding			

- HMAC-SHA256 key

Input (User Key)

byte	16			
	4	4	4	4
0-31	HMAC-SHA256 key			

- KUK

Input (User Key)

byte	16			
	4	4	4	4
0-15	AES 128bit CBC key			
16-31	AES 128bit CBCMAC key			

5.4 Generating an Encrypted Key Using Security Key Management Tool

You can use Security Key Management Tool to generate encrypted user keys. Security Key Management Tool is also available in a command line interface (CLI) version that can easily be deployed in settings such as the production line in a factory.

Security Key Management Tool

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

Refer to the user's manual for detailed instructions on how to use Security Key Management Tool.

5.4.1 Key Injection Procedure

The procedure for generating a key file to be used for key injection is described below. In the following example, C source code for AES is generated.

5.4.1.1 Procedure for CLI Version

1. Generating a UFPK
In a terminal emulator, run the following genufpk command.

Figure 5.1 genufpk Command Execution Result

- ## 2. Obtaining a W-UFPK

Send the ufpk.key file generated in step 1 to the Renesas Key Wrap service (<https://dlm.renesas.com/keywrap>) to obtain a W-UFPK. For detailed information, refer to the Renesas Key Wrap Service FAQ.

3. Generating an AES 128 key file as a C source file

In the terminal emulator, run the following genkey command.

```
> skmt.exe /genkey /iv "55aa55aa55aa55aa55aa55aa55aa55aa" /ufpk file="C:\work\ufpk.key"  
    /wufpk file="C:\work\ufpk.key_enc.key" /mcu "RX-RSIP-E11A" /keytype "AES-128" /key  
    "11111111222222233333344444444" /filetype "csource" /keyname "euk_aes128"  
    /output "C:\work\leuk_aes128.c"
```

Use the UFPK file generated in step 1 and the W-UFPK file generated in step 2.

Figure 5.2 genkey Command Execution Example

5.4.1.2 Procedure for GUI Version

- #### 1. Selecting an MCU or MPU and an encryption engine

On the **Overview** tab, select an MCU or MPU and an encryption engine.

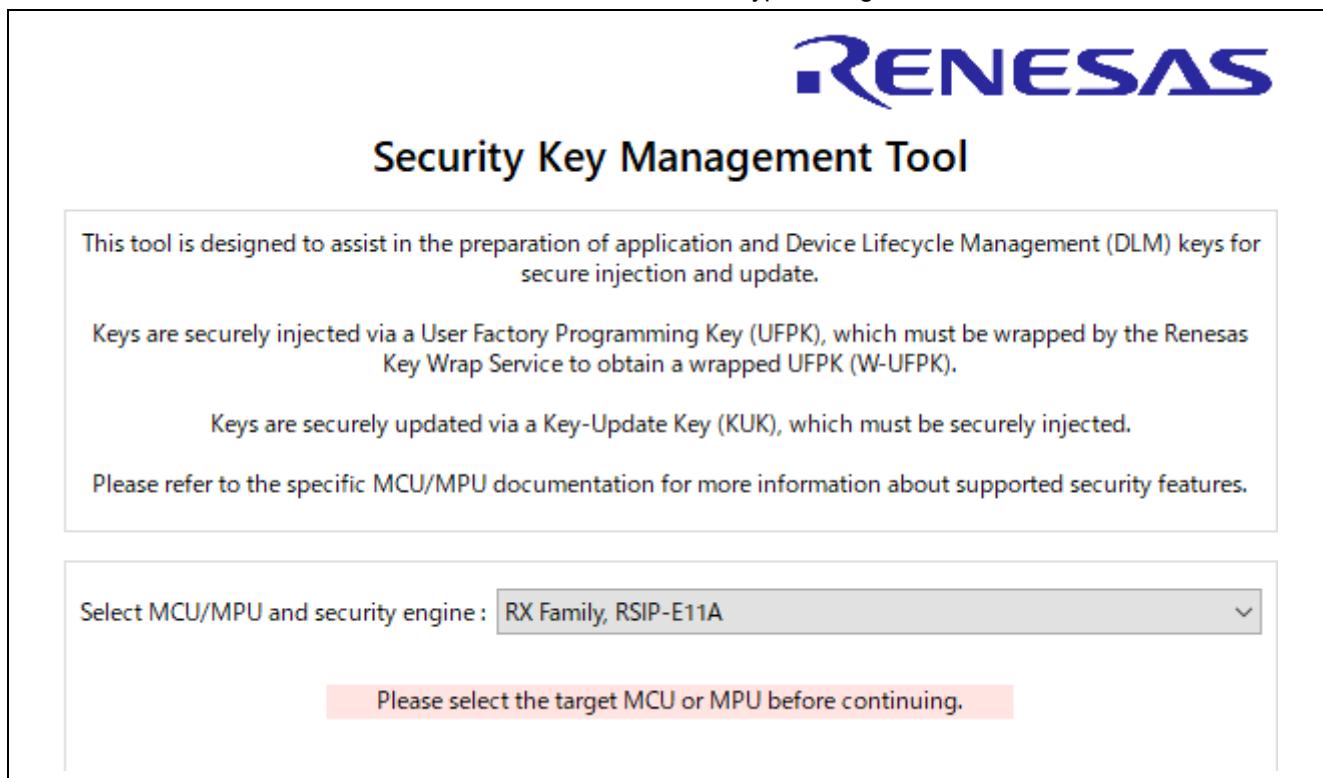


Figure 5.3 Overview Tab

- ## 2. Generating a UFPK

On the **Generate UFPK** tab, specify a UFPK value and a file name with the extension *.key to generate an UFPK file. This example uses the file name **ufpk.key**.

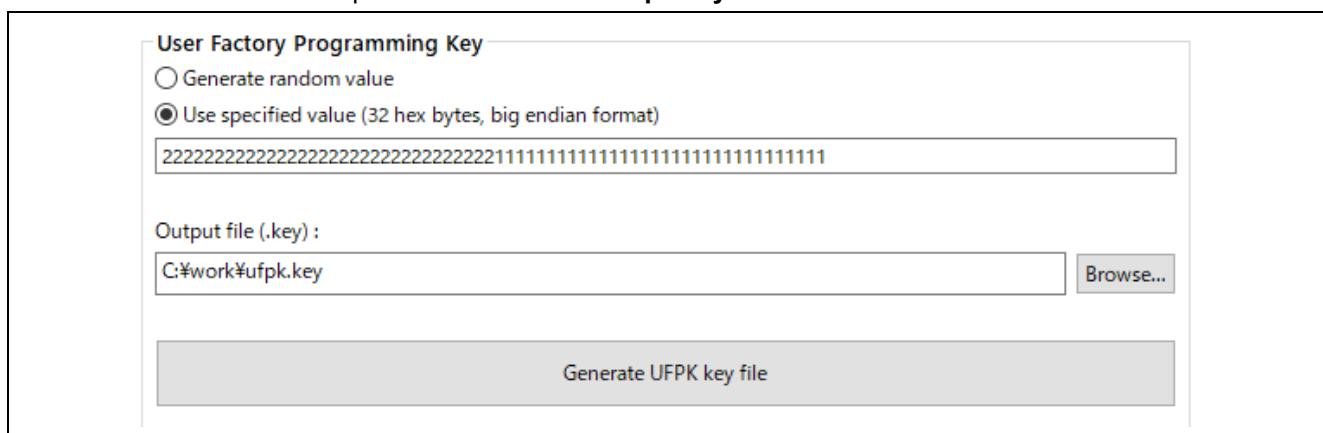


Figure 5.4 Example of UFPK Generation by Specifying Values on Generate UFPK Tab

Click the **Generate UFPK key file** button to generate an UFPK file. The following execution result is output when the file is generated successfully.



Figure 5.5 Generate UFPK Tab Execution Result

3. Obtaining a W-UFPK

Send the ufpk.key file generated in step 2 to the Renesas Key Wrap service (<https://dlm.renesas.com/keywrap>) to obtain a W-UFPK. For detailed information, refer to the Renesas Key Wrap Service FAQ.

4. Generating an AES 128 key file as a C source file

On the **Wrap Key** tab, generate an AES 128 key file. On the **Key Type** tab, select **AES** and **128 bits**, and on the **Key Data** tab, enter the AES 128 key data. For **Wrapping Key**, specify the UFPK file generated in step 2 and the W-UFPK file obtained in step 3. For **Format**: select **C Source**.

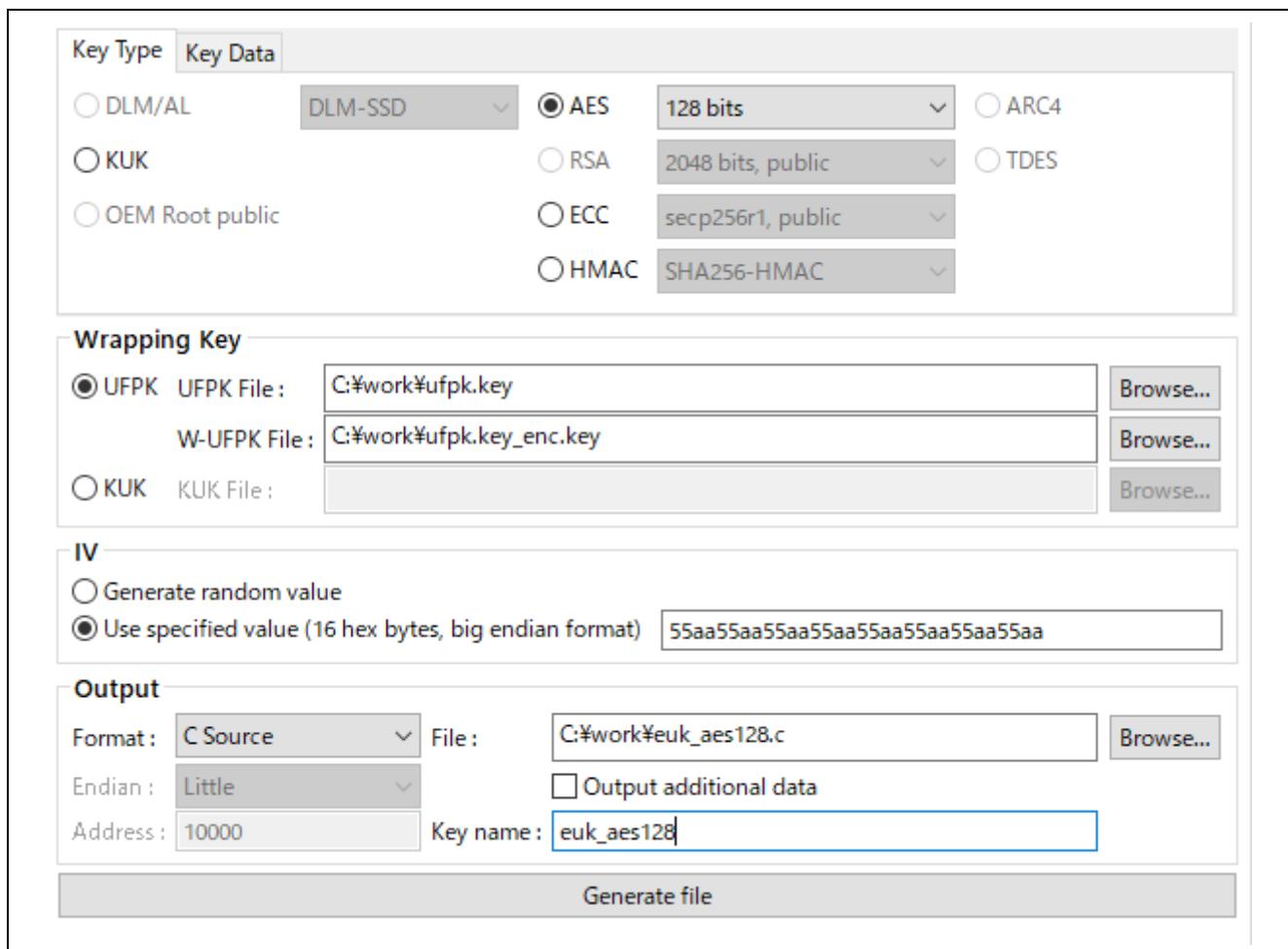


Figure 5.6 Example AES 128 Key File Output Settings on Wrap Key – Key Type Tab

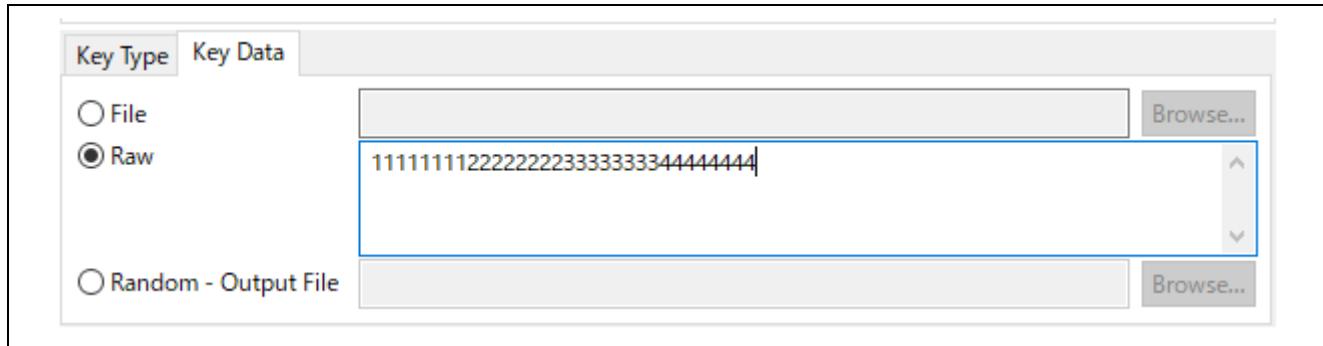


Figure 5.7 Example AES 128 Key C Source Output Settings on Wrap Key – Key Data Tab

The following is displayed when the operation completes successfully.

Figure 5.8 Wrap Key Tab Execution Result

Incorporate the data in the output C source file into your project with the same method in 5.4.1.1.

5.4.2 Key Update Operation Procedure

If a KUK has been injected beforehand, it is not necessary to obtain a new UFPK when performing a key update.

5.4.2.1 Procedure for CLI Version

1. Generating a UFPK

In a terminal emulator, run the following command:

Figure 5.9 genufpk Command Execution Result

2. Obtaining a W-UFPK

Send the ufpk.key file generated in step 1 to the Renesas Key Wrap service (<https://dlm.renesas.com/keywrap>) to obtain a W-UFPK.

For detailed information, refer to the Renesas Key Wrap Service FAQ or the relevant application note.

3. Generating a KUK

In the terminal emulator, run the following genkuk command.

```
> skmt.exe /genkuk /output "C:\work\kuk.key"  
    /kuk "d0aec19726cbc0e2fb403866b9b465a6c0d05b7a60362d5f435f9a3e98c79084"
```

```
C:\work\skmt\tool>skmt.exe /genuk /output "C:\work\kuk.key" /kuk "d0aec19726cbc0e2fb403866b9b465a6c0d05b7a60362d5f435f9a3e98c79084"  
KUK: D0AEC19726CBC0E2FB403866B9B465A6C0D05B7A60362D5F435F9A3E98C79084  
Output file: C:\work\kuk.key
```

Figure 5.10 genkuk Command Execution Result

4. Generating an encrypted user key file containing a KUK encrypted using a UFPK

In the terminal emulator, run the genkey command to encrypt a KUK using a UFPK.

```
> skmt.exe /genkey /ufpk file="c:\work\ufpk.key" /wufpk file="C:\work\ufpk.key_enc.key"  
    /mcu "RX-RSIP-E11A" /keytype "key-update-key" /key file="C:\work\kuk.key" /filetype  
"csource"  
    /output "C:\work\kuk.c"
```

Use the UFPK file generated in step 1 and the W-UFPK file generated as described in step 2.

Figure 5.11 genkey Command Execution Example

Use the data from the output C source file to inject the KUK by calling the R_RSIP_InitialKeyWrap() function in your program.

Next, the wrapping method using a KUK for key updating in the field is described.

5. Generating an encrypted file containing a secp256r1 public key

In the terminal emulator, run the following genkey command.

```
> skmt.exe /genkey /kuk file="C:\work\kuk.key" /mcu "RX-RSIP-E11A"
/keytype "secp256r1-public" /key
"19b3f37e35d0a5448983bfc91f69b8e167c135fa0f863d6d0efb99fce34f593823b8eb34f45ae0197aef66
426a08459019d63b04bc5eccf3b428181a92f3ff9c"
/filetype "csource" /keyname "secp256r1public"
/output "C:\work\secp256r1public.c"
```

Use the file generated in step 3 as the KUK file.

```
C:\Renesas\SecurityKeyManagementTool\cli>skmt.exe /genkey /kuk file="C:\work\kuk.key" /mcu "RX-RSIP-E11A" /keytype "secp256r1-public" /key "19b3f37e35d0a5448983bfc91f69b8e167c135fa0f86
3d6d0efb99fce34f593823b8eb34f45ae0197aef66426a08459019d63b04bc5eccf3b428181a92f3ff9c" /filet
ype "csource" /keyname "secp256r1public" /output "C:\work\secp256r1public.c"
Output File: C:\work\secp256r1public.h
Output File: C:\work\secp256r1public.c
KUK: D0AEC19726CBC0E2FB403866B9B465A6C0D05B7A60362D5F435F9A3E98C79084
IV: 86077570BC362CBFD707CA84C5D118C2
Encrypted key: 89FF8126D500B5C25ADB98A96552809F32E1B7B5177427FFC9A91A6935F7F5CBECE6E16636ADD
86E371A30A4D3D8355AC5669F0C18ED8CC8E6A4795D8A8C12B4C397302906A54064BE344698D723797F
```

Figure 5.12 genkey Command Execution Example

Input the data from the output C source file to the external interface of the device, and pass the data to the parameters of the R_RSIP_EncryptedKeyWrap() function to output an updated secp256r1 public key wrapped key.

5.4.2.2 Procedure for GUI Version

- #### 1. Selecting an MCU or MPU and an encryption engine

On the **Overview** tab, select an MCU or MPU and an encryption engine.

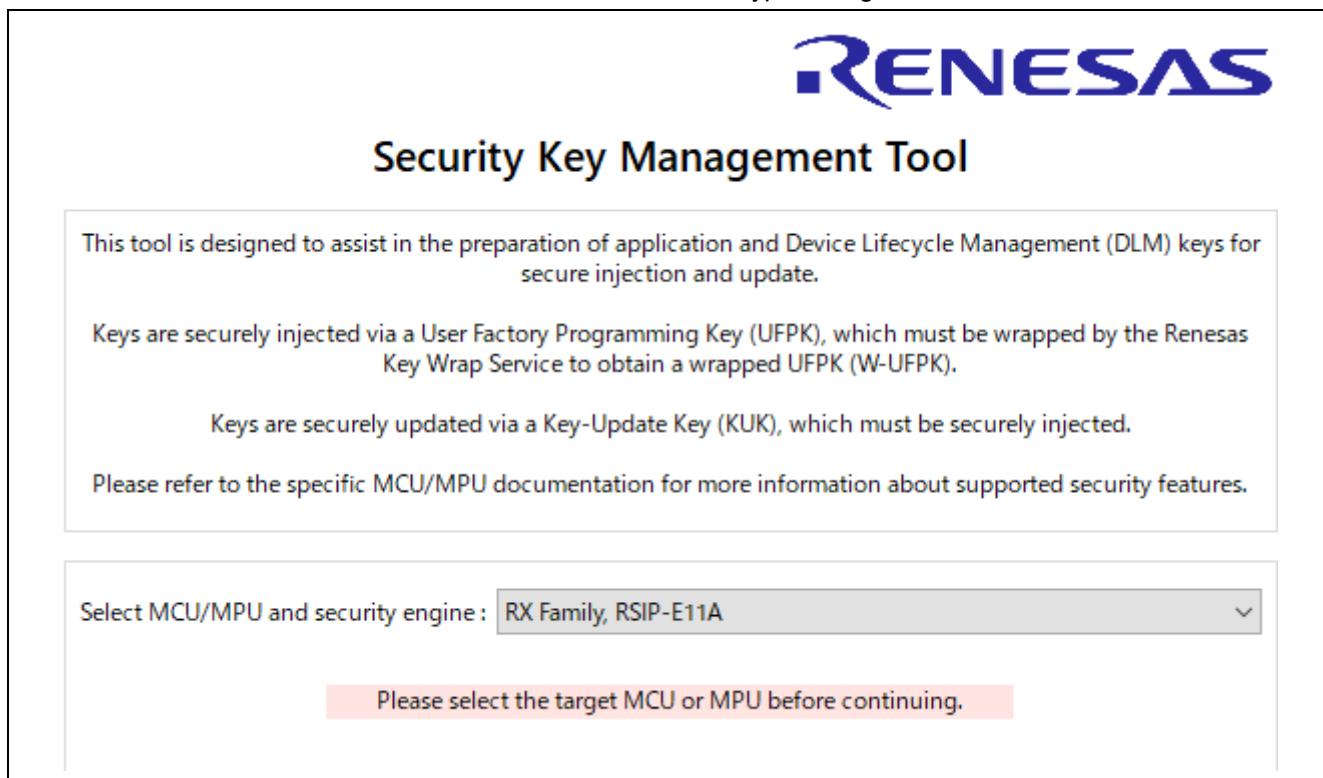


Figure 5.13 Overview Tab

- ## 2. Generating a UFPK

On the Generate UFPK tab, specify an UFPK value and a file name with the extension *.key to generate an UFPK file. This example uses the file name `ufpk.key`.

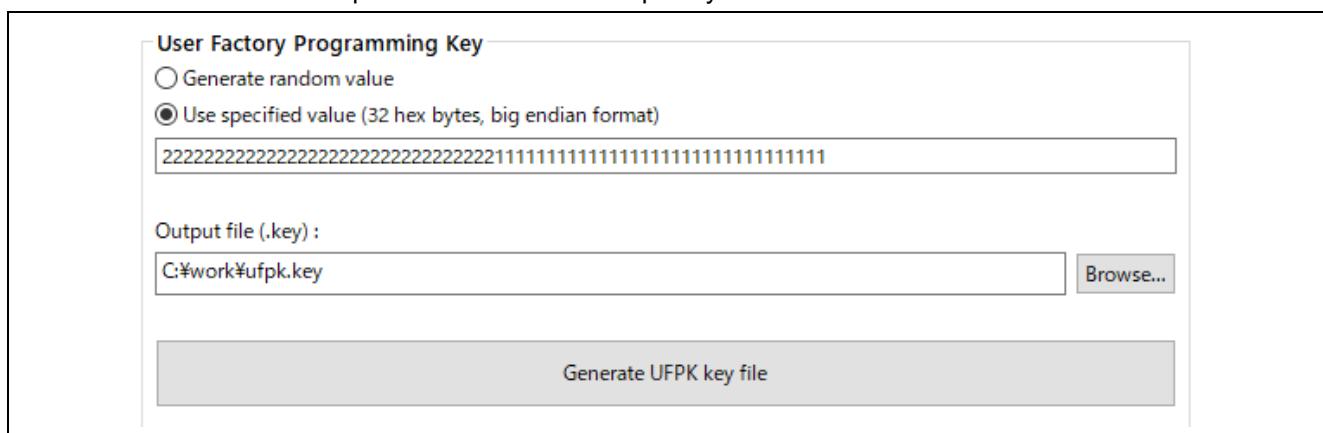


Figure 5.14 Example UFPK File Generation Settings on Generate UFPK Tab

Click the **Generate UFPK key file** button to generate an UFPK file. The following is displayed when the operation completes successfully.

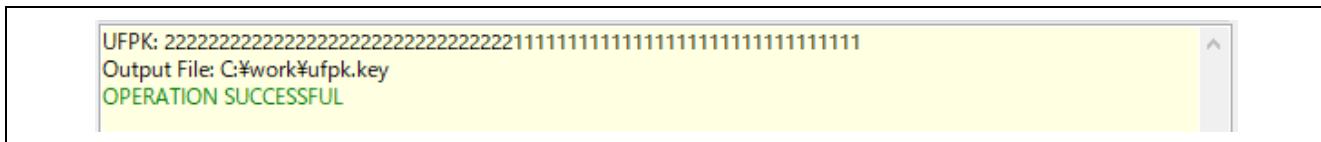


Figure 5.15 Generate UFPK Tab Execution Result

3. Obtaining a W-UFPK

Send the upfk.key file generated in step 2 to the Renesas Key Wrap service (<https://dlm.renesas.com/keywrap>) to obtain a W-UFPK.

For detailed information, refer to the Renesas Key Wrap Service FAQ or the relevant application note.

4. Generating a KUK

On the Generate KUK tab, select whether to use the tool to generate a random value for the KUK or to enter a 256-bit value for the KUK. Enter a file name with the extension *.key. This example uses the file name `kuk.key`.

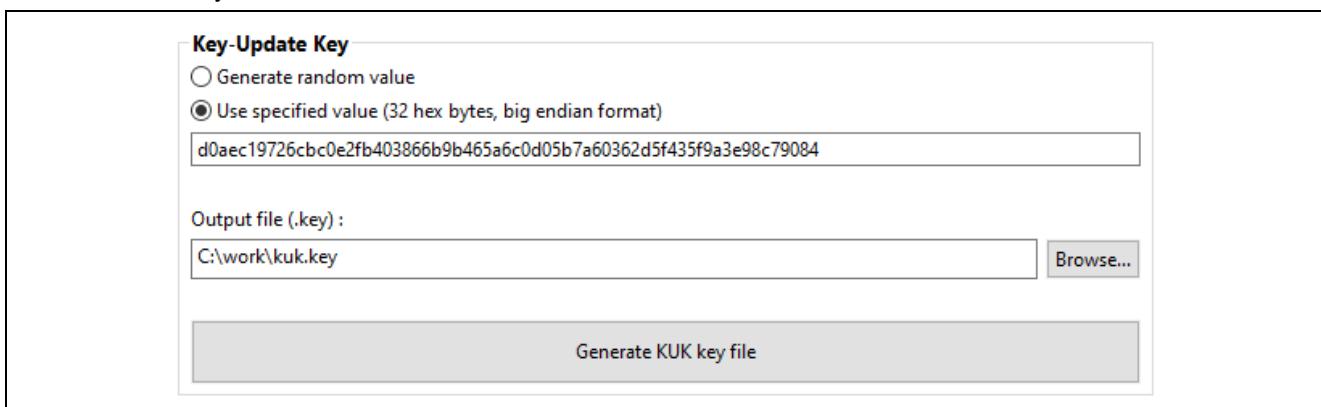


Figure 5.16 Example KUK File Generation Settings on Generate KUK Tab

Click the **Generate KUK key file** button to generate a KUK file. The following is displayed when the operation completes successfully.



Figure 5.17 Generate KUK Tab Execution Result

5. Generating a KUK file in C source file format

Select **KUK** on the Key Type tab, which is on the Wrap Key tab. On the Key Data tab, enter the file name of the KUK file generated in the preceding step (**kuk.key** in this example). For **Wrapping Key**, select **UFPK** and select the UFPK file generated in step 2 and the W-UFPK file obtained as described in step 3. In this example, **Generate random value** is selected for **IV** in the interest of simplicity. In the **Output** panel, select **C Source** for **Format**: and enter the file name of the C source file.

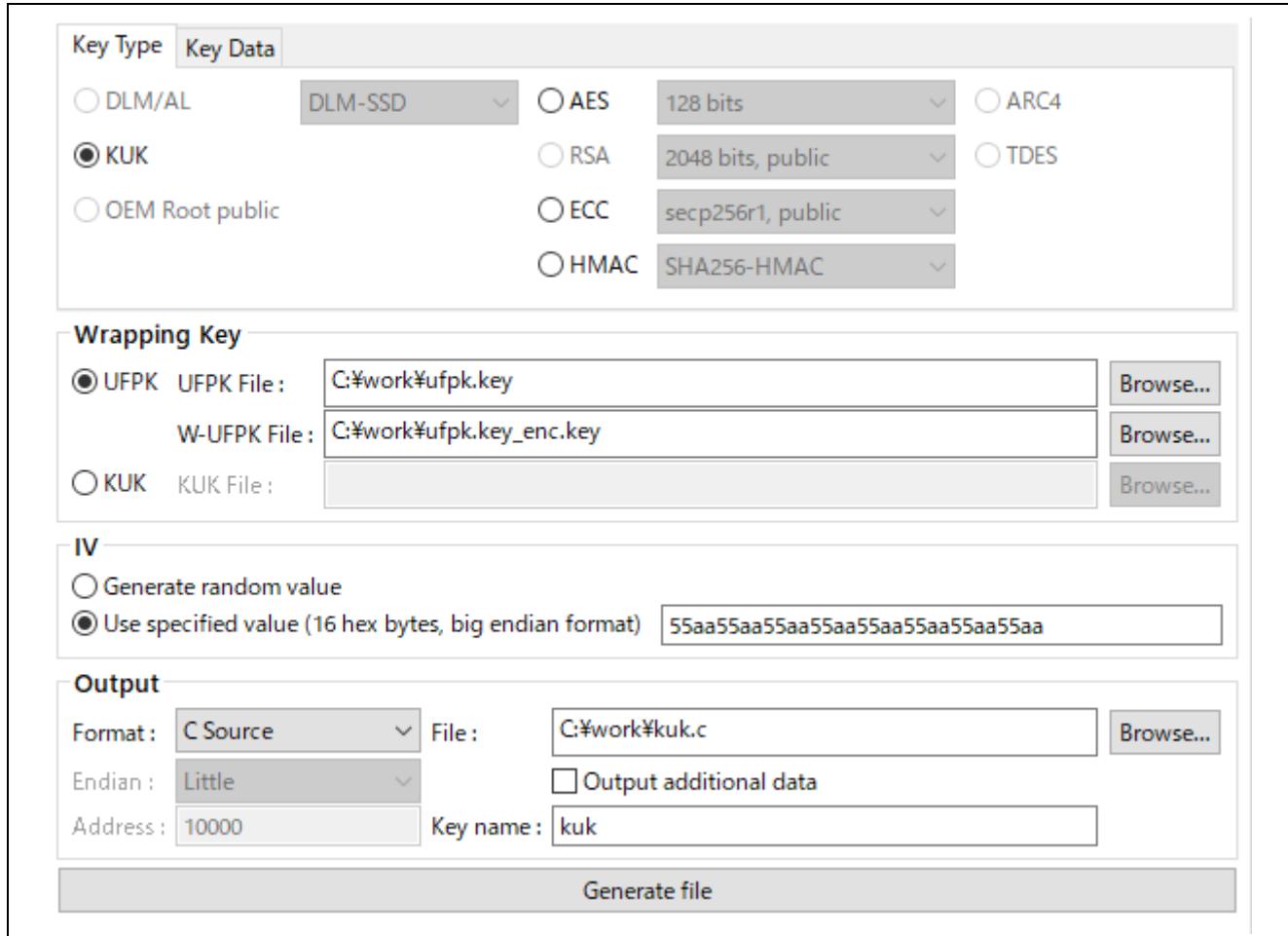


Figure 5.18 Example C Source File Output Settings for KUK File on Wrap Key – Key Type Tab

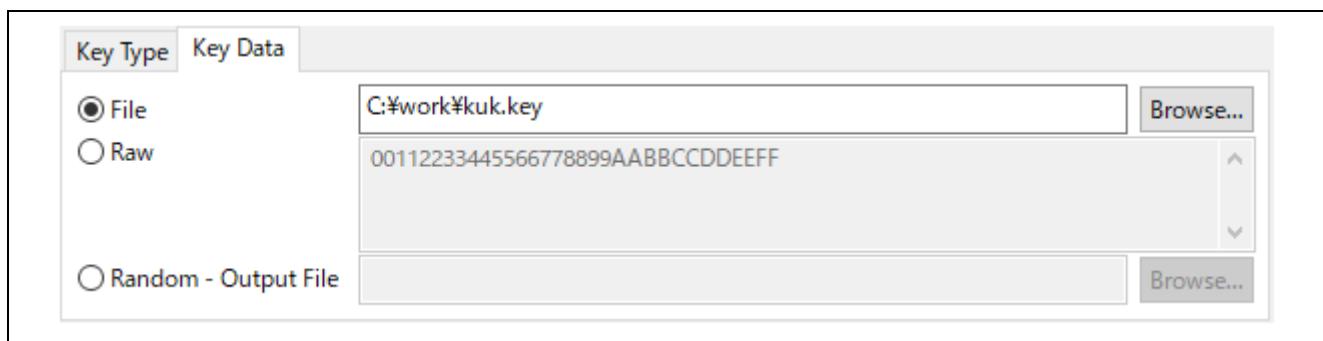


Figure 5.19 Example C source File Output Settings for KUK File on Wrap Key – Key Type Tab

Use the data from the output C source file to inject the KUK by calling the R_RSIP_InitialKeyWrap() function in your program.

Next, the wrapping method using a KUK for key updating in the field is described.

6. Generating a secp256r1 public key file in C source file format

On the Wrap Key tab, generate a secp256r1 public key file in C source file format.

On the Key Type tab, select **ECC** and **256 bits, public**, and on the Key Data tab, enter the secp256r1 256 public key data.

For **Wrapping Key**, specify the KUK file generated in 4. In this example, **Generate random value** is selected for **IV** in the interest of simplicity. Under **Output**, select **C Source** for **Format**: and enter a file name with the extension *.c.

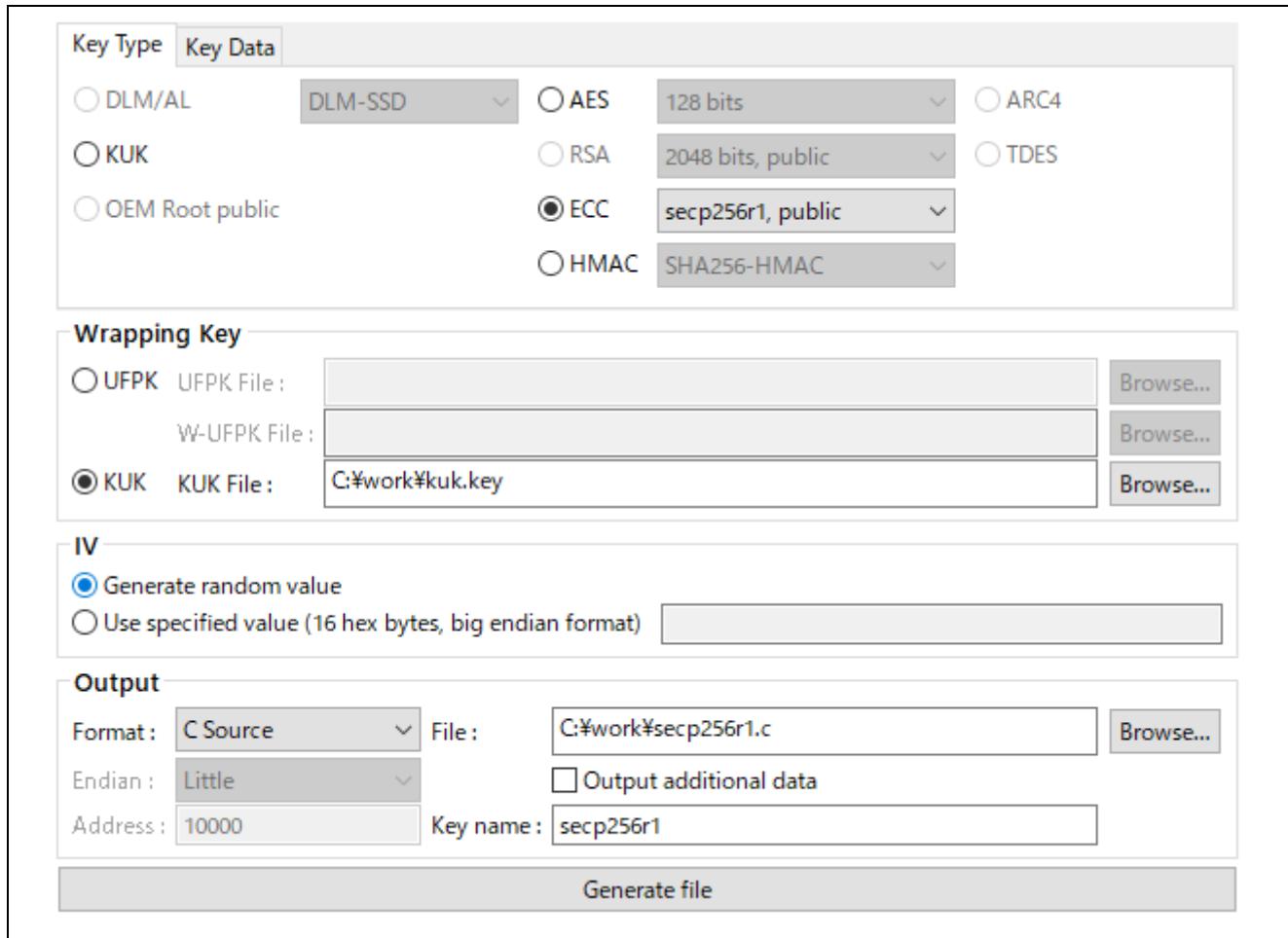


Figure 5.20 Example secp256r1 Public Key C Source File Generation Settings on Wrap Key – Key Type Tab

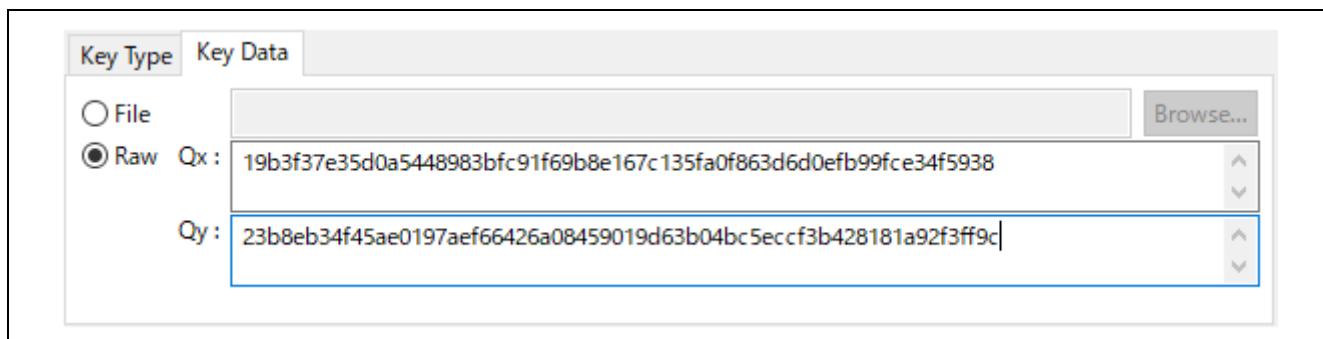


Figure 5.21 Example secp256r1 Public Key C Source File Generation Settings on Wrap Key – Key Data Tab

The following is displayed when the operation completes successfully.

```
Output File: C:\work\secp256r1.h
Output File: C:\work\secp256r1.c
KUK: D0AEC19726CBC0E2FB403866B9B465A6C0D05B7A60362D5F435F9A3E98C79084
IV: B8E884BADF11B9A1D51ED7E016AB1B99
Encrypted key:
296A6DAD17F922660DA1A8BC5729095376D51DF9D9F2EF4235AC9C8216D69598962F42648F31ECC1F3FD955567570B
DA34CEA831F56E6DC7A072F2C88EA93936094957BBADD808CFA5AF2A12597A25C1
OPERATION SUCCESSFUL
```

Figure 5.22 Wrap Key Tab Execution Result

You can perform key updating in the field by incorporating the generated C source file into your project and using it to create update data.

6. Sample Programs

6.1 Confirming Operation of How to Use Key Injection and Cryptography

The demo project listed in Table 6.1 shows the usages of cryptographic operation and random number generation APIs provided by RSIP PM driver.

Table 6-1 Demo Projects to used on each MCU

MCU	Demo Project
RX261	rx261_ek_rsip_sample

The demo project outputs the operation result with UART. So the running board is connected with PC which terminal software is installed.

The demo project operates with using little-endian byte ordering.

6.1.1 Setting Up the Demo Project

Wiring connections between the PC and board are shown below.

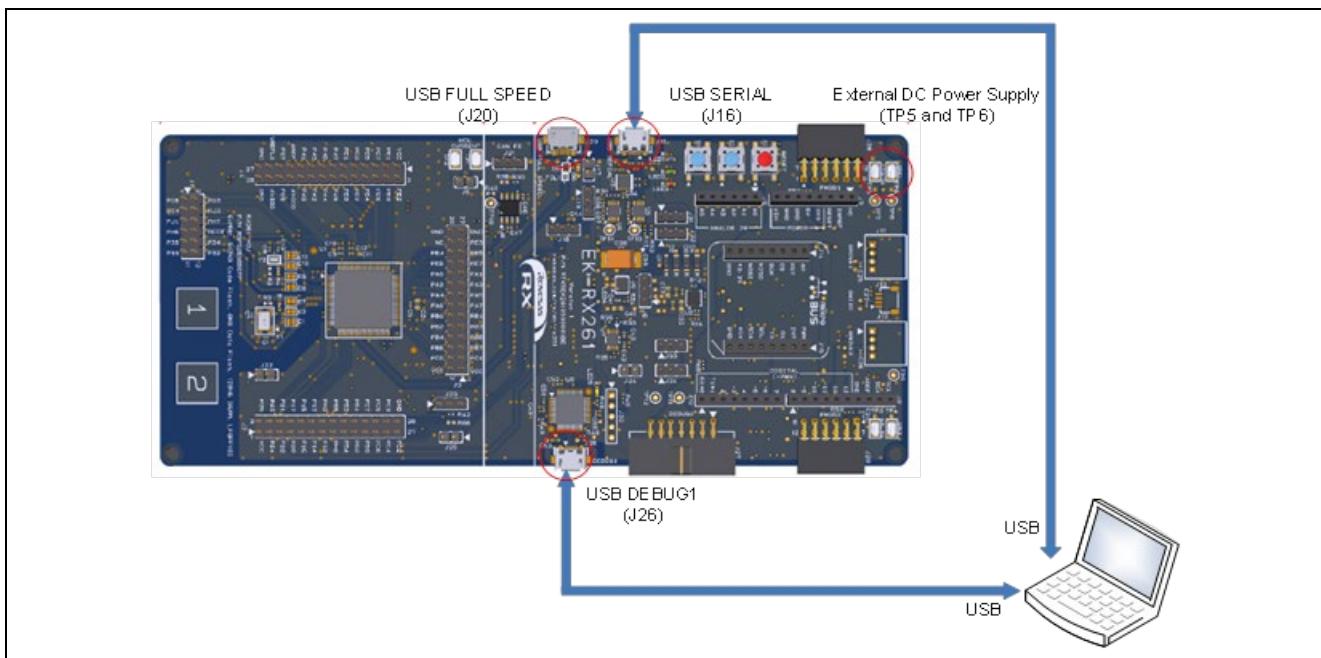


Figure 6.1 Wiring Connections between PC and EK-RX261 Board

The USB DEBUG1, USB FULL SPEED, USB SERIAL and external DS power supply can be used to supply electricity to EK-RX261 Board. Please supply electricity with either port.

The serial settings in Tera Term are as follows:

- Speed: 115200 bps
- Data: 8 bit
- Parity: None
- Stop bits: 1 bit
- New-line code (Transmit): CR

6.1.2 Overview of Demo Project

The flow chart of the demo project is shown in Figure 6.2.

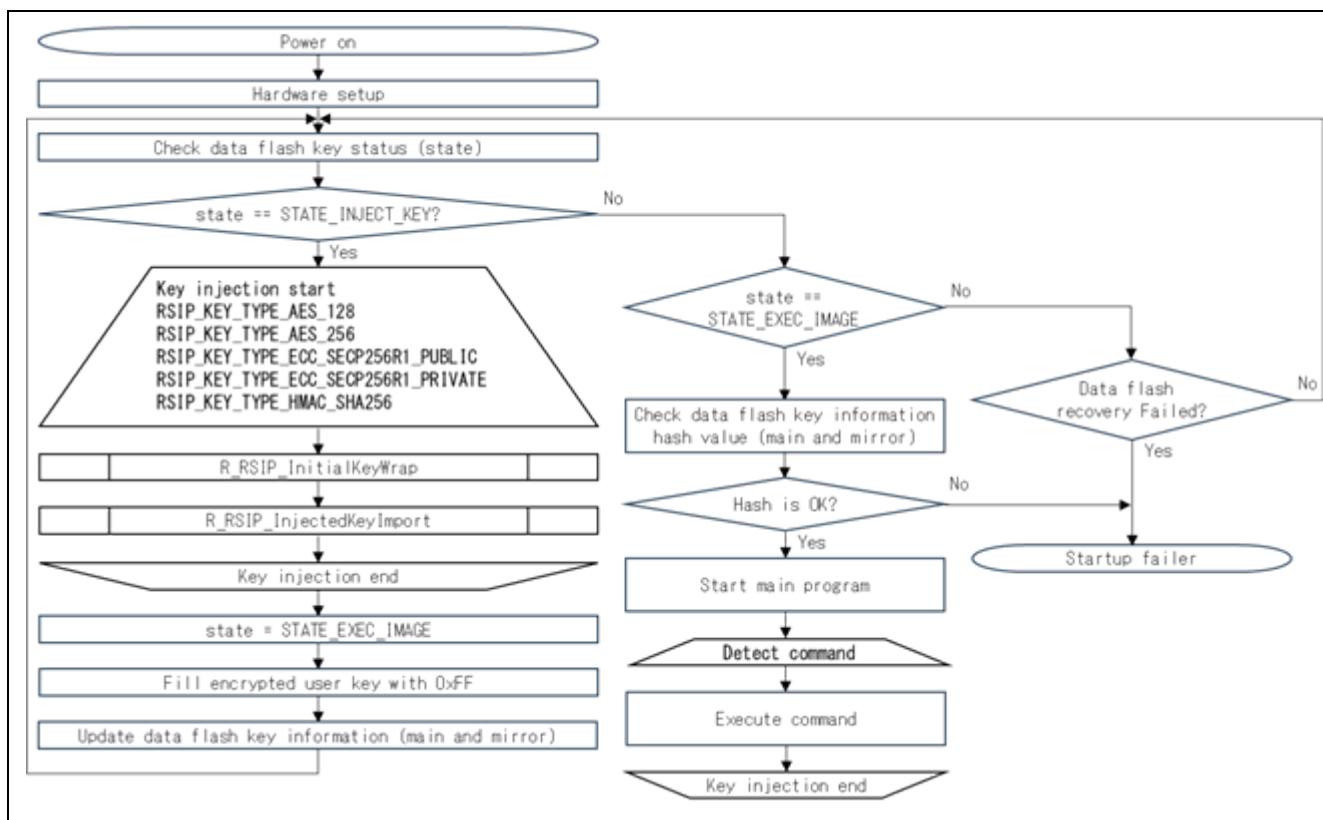


Figure 6.2 Flow chart of Confirming Operation of How to Use Key Injection and Cryptography

The behavior of the demo project is managed with the state transition and the state is managed with the flag in the data flash.

Table 6-2 State of the demo project

State	Operation content
STATE_INJECT_KEY	<p>Inject the wrapped keys to the data flash. In the demo project, the area to store the keys in the data flash is divided to main area and mirror area. The key data can be recovered with the structure if the power shutoff has occurred while writing the keys. After injecting the keys, transit the state to STATE_EXEC_IMAGE. The key injection is executed only in the first time of the start operation. After the time, the state begins with the state STATE_EXEC_IMAGE.</p>
STATE_EXEC_IMAGE	Confirm the key data in the data flash with checking its hash value. When the validity is confirmed, the project begins accepting the commands.

The commands implemented in the demo project is shown below.

Table 6-3 Commands of the demo project

Command	Operation
display	Displays the wrapped keys.
encdemo [Arg1]	Encrypts the Arg1 value in AES 128-bit ECB mode.
function	Performs the following tests. - AES128/256 ECB, CBC, CTR, CCM, GCM encryption/decryption - AES128/256 CMAC generation/verification - SHA-224/256 HMAC generation/verification - ECDSA P256 signature generation/verification
random	Generates a pseudo random value.

6.1.2.1 Characteristics of Wrapped Keys and how to check them in the demo project

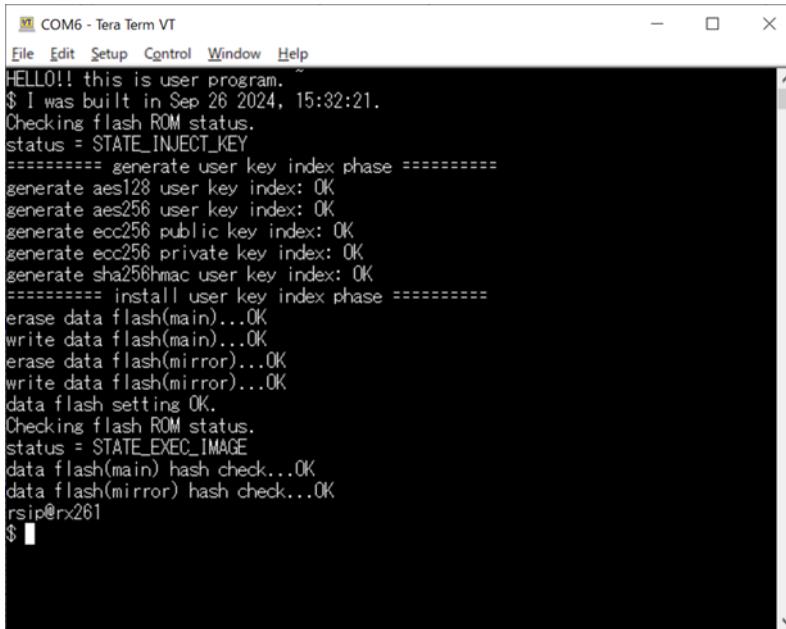
The RSIP PM driver does not use user keys in plain text. This means that when the software is reverse engineered, the plain-text user key is not exposed. The demo project uses the Wrapped Key of the user key injected into the data flash. When reading the data flash area (0x00100000) where the Wrapped Key of the user key is placed, we can confirm that the plain text of the user key is not written.

Since the Wrapped Key is generated in conjunction with the HUK, the Wrapped Key generated on one chip cannot be copied to another chip for use. This is meant to prevent dead copies of user keys. If you try to use a Wrapped Key generated by another device in your demo project, you can be sure that the RSIP PM driver will fail.

Also, since Wrapped Keys contain random numbers, even if the same device generates a Wrapped Key for the same user key, it will generate a different value. This makes it difficult to guess the user key from the Wrapped Key. These Wrapped Key features can be seen in the demo project. Download the demo project again and you will see that the Wrapped Key value changes each time you run the display command.

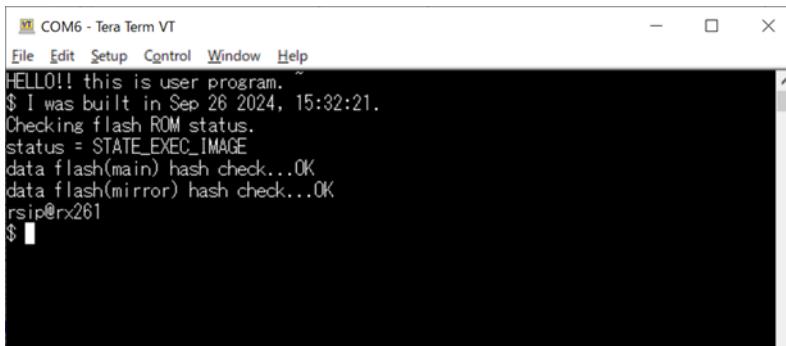
6.1.3 Demo Project Execution Example

Figure 6.3 shows the terminal output when key installation completes successfully after the program is downloaded to the MCU and run for the first time, and Figure 6.4 shows the terminal output when the program is run from a state other than when it has been newly downloaded to the MCU. After the above output is displayed, the program is ready to accept the commands listed in Table 6-3.



```
COM6 - Tera Term VT
File Edit Setup Control Window Help
HELLO!! this is user program.
$ I was built in Sep 26 2024, 15:32:21.
Checking flash ROM status.
status = STATE_INJECT_KEY
===== generate user key index phase =====
generate aes128 user key index: OK
generate aes256 user key index: OK
generate ecc256 public key index: OK
generate ecc256 private key index: OK
generate sha256hmac user key index: OK
===== install user key index phase =====
erase data flash(main)...OK
write data flash(main)...OK
erase data flash(mirror)...OK
write data flash(mirror)...OK
data flash setting OK.
Checking flash ROM status.
status = STATE_EXEC_IMAGE
data flash(main) hash check...OK
data flash(mirror) hash check...OK
rsip@rx261
$
```

Figure 6.3 Tera Term Display (Initial Execution)



```
COM6 - Tera Term VT
File Edit Setup Control Window Help
HELLO!! this is user program.
$ I was built in Sep 26 2024, 15:32:21.
Checking flash ROM status.
status = STATE_EXEC_IMAGE
data flash(main) hash check...OK
data flash(mirror) hash check...OK
rsip@rx261
$
```

Figure 6.4 Tera Term Display (2nd and Subsequent Executions)

As an example of the command usage, the usage of encdemo command is shown below.

The encdemo command performs the encryption of the data inputted as the argument of the command by AES ECB mode with using AES 128-bit key injected before the procedure.

The demo project

Figure 6.12 shows a usage example in which the command and argument **encdemo aaaaaaaaaaaaaaaaaaaaaaaa** are entered and encryption is performed using an AES 128-bit key index. The key used for encryption is the user key (16 bytes (128 bits)) input to Security Key Management Tool. When the command is run using the default value of **0x11, 0x11, 0x11, 0x11, 0x22, 0x22, 0x22, 0x22, 0x33, 0x33, 0x33, 0x44, 0x44, 0x44, 0x44**, the result is as shown in the screenshot below.

```
rsip@rx261
$ encdemo aaaaaaaaaaaaaaaaaaaaaaaa
input data (plain text): aaaaaaaaaaaaaaaaaaaaaaaa
output data (cipher text): f4136afc2a9df52b31c447f4d13a78b4
rsip@rx261
$
```

Figure 6.5 Tera Term Display (encdemo Command)

6.2 Confirming Operation of the Secure Boot and Firmware Update

The secure bootloader and firmware update demo projects show how to realize the secure boot and firmware update. In this demo project, two methods to set up the program of secure boot and firmware update to the device are shown below.

1. A method that firstly only the secure boot program is downloaded, secondly the firmware update program is written to the flash memory by the function of the secure boot program (two-step setup)
2. A method to make a Motorola S format file which includes both secure boot program and firmware update program and write it to the flash memory (inclusive setup)

The secure bootloader and firmware update demo projects decrypt the firmware update program which is encrypted with the method described in 4.2.9 Firmware Update/Secure Boot with using the APIs described in 3.13.3 Encrypting the User Program. Secure firmware update can be executed by starting after verifying encrypted firmware update program and writing it to the flash memory. And secure boot can be executed by verifying the firmware update program before starting.

The demo project has two projects, secure boot project and firmware update project. The secure boot project executes not only secure boot but also firmware update to write firmware update program to the device.

Secure bootloader project: rx261_ek_rsip_secure_boot

Firmware update project: rx261_ek_rsip_user_program

The firmware update functionality of the demo projects supports use of a UART or USB interface to send the initial program or the updated program to the device. In addition, Security Key Management Tool can be used to generate a file in Motorola S format^{*1} containing both the secure bootloader and the encrypted user program (factory programming function).

The demo projects can use dual bank mode for firmware update operation on devices with on-chip flash ROM that supports dual bank mode.

Notes: 1. Motorola S format files containing a secure bootloader and an encrypted user program can be decrypted by a corresponding RX RSIP-equipped device if the Motorola S format file itself is stolen. It is therefore necessary to ensure that programming to devices of encrypted user programs with secure bootloaders takes place in a secure facility.

The demo projects use a UART to output the result and operate firmware update via UART. Please connect a PC which terminal software is installed to the board to operate the demo projects. In the explanation below, Tera Term is used as the terminal software.

The demo projects operate using little-endian byte ordering.

The version of the FIT modules which is used in the demo projects are shown below. Some of the internal functions which are used to execute secure boot procedure are allocated to the SECURE_BOOT section.

Table 6-4 FIT modules used in the demo projects

FIT modules	version
r_bsp [Note]	7.53
r_byteq	2.11
r_cmt_rx	5.71
r_flash_rx	5.22
r_fwup	2.04
r_sci_rx	5.41
r_sha_rx	1.06
r_sys_time_rx	1.02
r_tfat_driver_rx	2.61
r_tfat_rx	4.14
r_usb_basic_mini	1.31
r_usb_hmsc_mini	1.31

Note: BSP FIT module is modified to use USB memory for firmware update in bootloader project.

6.2.1 Setting Up the Demo Project

The demo projects will run on the boards listed in Table 6.5. To use the board, please set the board for the interface to use.

Table 6-5 Boards used in the demo project and their settings

Boards	Jumper setting to use UART	Jumper setting to use USB
EK-RX261	-	J17 : open J18 : open J19 : 1-2 short

Connections between the PC and board are shown below.

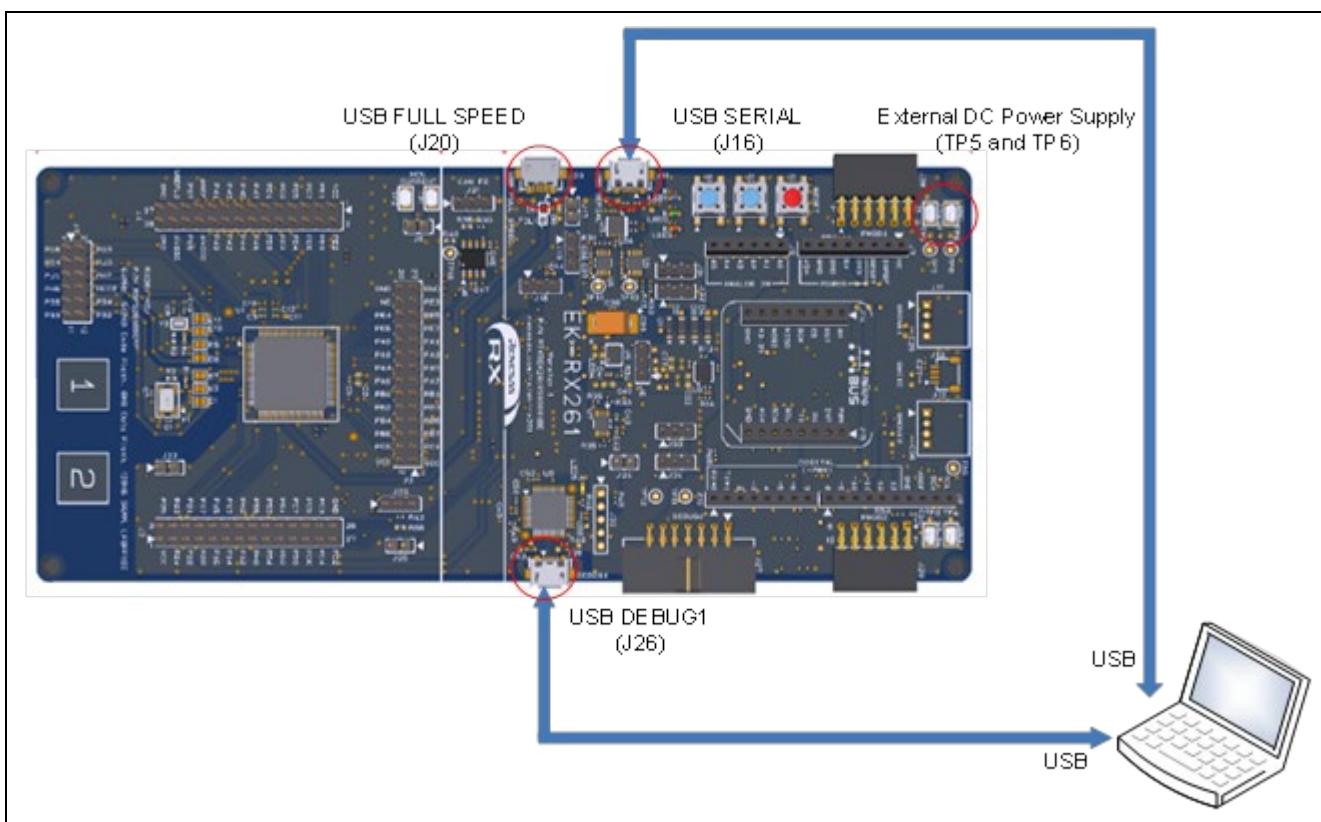


Figure 6.6 Connections between EK-RX261 and PC for Firmware Updates Using UART

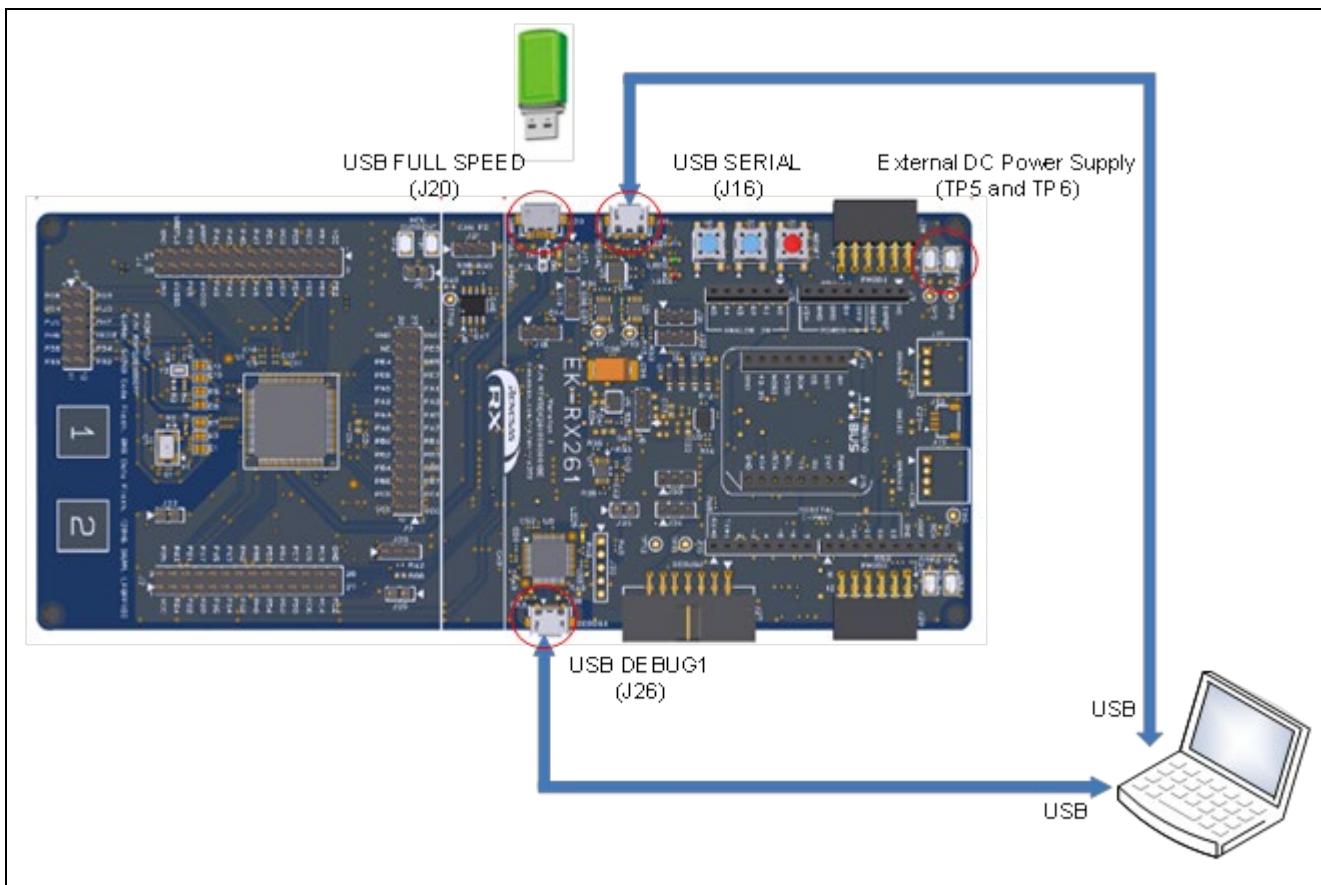


Figure 6.7 Connections between EK-RX261 and PC for Firmware Updates Using USB

Tera Term serial settings are as follows.

Table 6-6 Tera Term settings

Item	Value
Speed	115200bps
Data	8bit
Parity	none
Stop bit	1bit
Flow control	RTS/CTS
New-line code (Transmit)	CR
Local echo	OFF

6.2.2 Overview of Secure Bootloader and Firmware Update Demo Projects

The directory structure of the secure bootloader project and firmware update demo project is described in Table 6-7.

Table 6-7 Directory structure of secure bootloader project and firmware update project

Directory Name	Description
rx261_ek_rsip_secure_update	Secure boot/secure update implementation example
rx261_ek_rsip_secure_boot	Secure boot firmware
key	UFPK and W-UFPK file which can be used to execute example
skmt	Setting file of Security Key Management Tool
src	Source code of the secure boot firmware
rx261_ek_rsip_user_program	Firmware update program
src	Source code of the firmware update program

The secure bootloader and firmware update demo projects handle flash memory area with dividing into two sections, which are used as the main area and buffer area.

Main area: Storage area for program to be executed

Buffer area: Storage area for program to be applied as an update

6.2.2.1 Secure Bootloader project

A flowchart of the secure bootloader project is shown below.

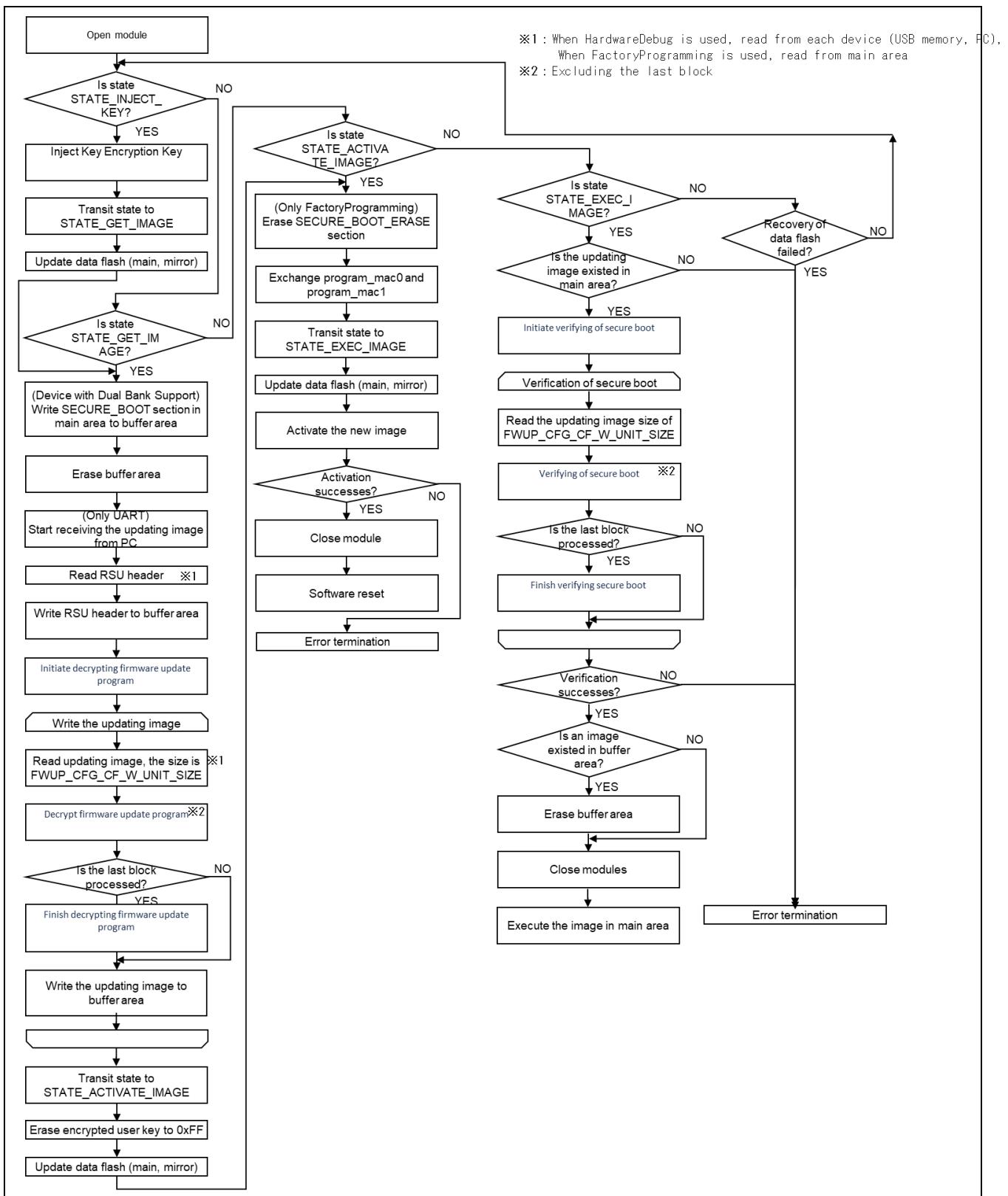


Figure 6.8 Secure Bootloader project flowchart

Secure bootloader project manages the operation with using state transition. The state is managed by the flag which is stored in the data flash. Each state and their operation contents are shown in .

Table 6-8 State of the Secure Boot project

State	Operation Content
STATE_INJECT_KEY	Store wrapped keys to the data flash. Then, the wrapped keys are stored both main area and buffer area to prevent an elimination of the wrapped keys which is caused by a failure of writing to the memory because of cutoff energy while updating. After key injection is finished, the state transits to STATE_GET_IMAGE.
STATE_GET_IMAGE	Get encrypted firmware update program and decrypt it. Firstly, erase the buffer area. While getting an encrypted firmware update program thorough UART or USB memory, decrypt it and write to the buffer area. After erasing the encrypted key, the state transits to STATE_ACTIVATE_IMAGE.
STATE_ACTIVATE_IMAGE	Activate the updating image. When the inclusive setup is used, the process to inject key is erased firstly. After copying the firmware update program to the main area, the state transits to STATE_EXEC_IMAGE and software reset is executed.
STATE_EXEC_IMAGE	Verify MAC of the program stored in main area. When an image is stored in the buffer area, erase it. And start the program in the main area.

6.2.2.2 Firmware Update Project

A flowchart of the firmware update project is shown below.

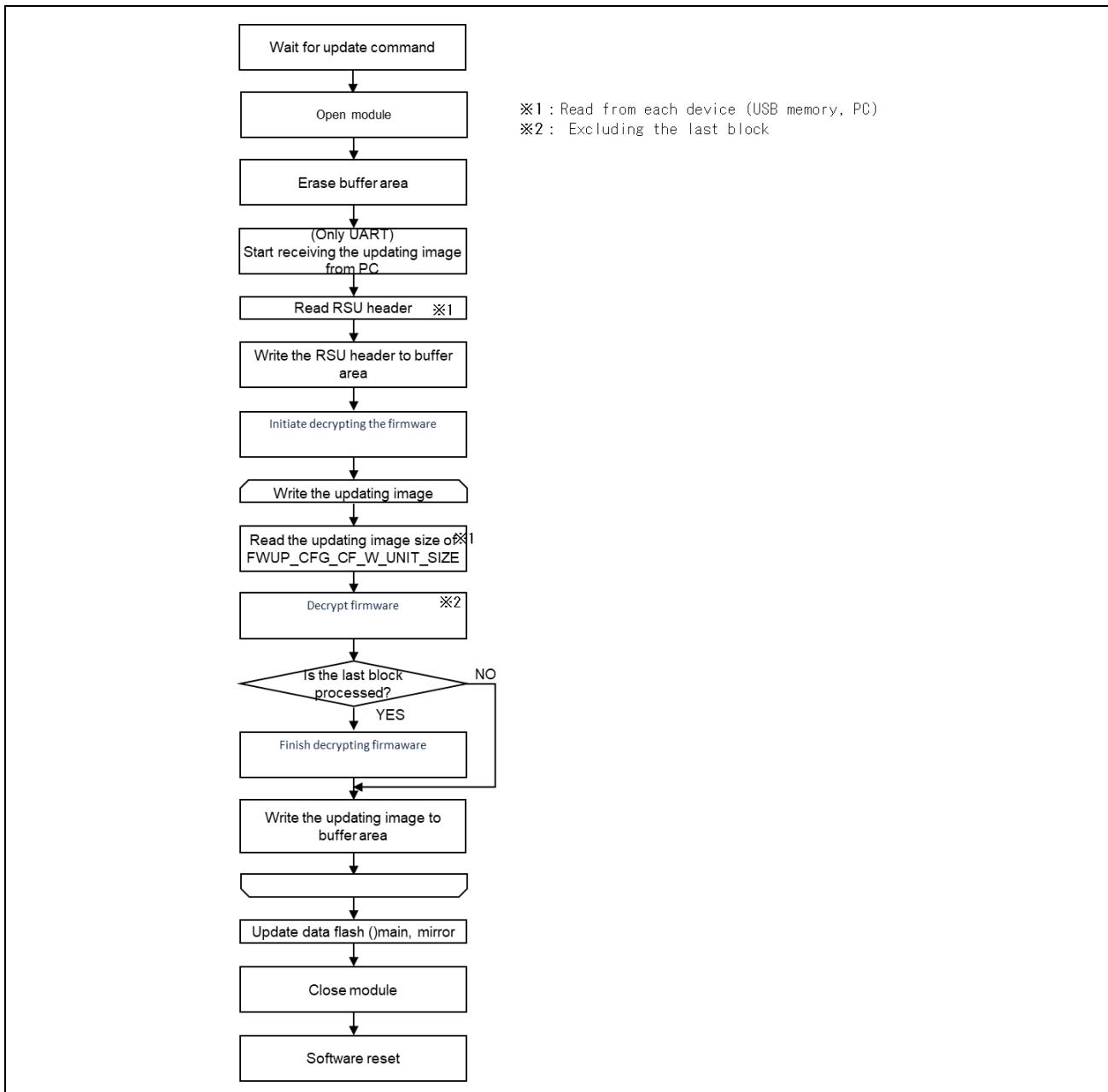


Figure 6.9 Firmware Update Flowchart

In the firmware update project, a command is implemented described below.

Table 6-9 Command of the eFirmware Update project

Command	Operation
update	Execute firmware update. Firstly, erase the buffer area. While getting an encrypted user program thorough UART or USB memory, decrypt it and write to the buffer area. After the firmware update is terminated normally, transits the state to STATE_ACTIVATE_IMAGE and execute software reset.

6.2.3 Execution Example of the Demo Project

6.2.3.1 Execution Example of Two-Step Setup

(1) Generating an Encryption Key File for the Firmware Update

First, on the **Wrap Key** tab of Security Key Management Tool, generate a key file to encrypt the image encryption key used to encrypt the firmware update program.

The secure bootloader project uses the following values for **UFPK** and **Key Encryption Key**.

Key Encryption Key = 0123456789abcdef0123456789abcdef

- Command line version

Run the following command.

```
> skmt.exe /genkey /iv "55aa55aa55aa55aa55aa55aa55aa55aa55aa" /ufpk  
file="{$skmt_loc}\..\key\sample.key"  
    /wufpk file="{$skmt_loc}\key\sample.key_enc.key"  
    /mcu "RX-RSIP-E11A" /keytype "AES-128" /key "0123456789ABCDEF0123456789ABCDEF"  
    /filetype "csource" /keyname "euk_aes128" /output "{$skmt_loc}\genkey\euk_aes128.c"
```

For {\$skmt_loc}, substitute the path of the secure bootloader project folder.

- Standalone version

Enter the following values.

— Key Type tab

Select AES – 128 bits.

— Key Data tab

Enter the string **0123456789abcdef0123456789abcdef** as plaintext data.

— Wrapping Key

UFPK File: **sample_key.key** included in the sample

W-UFPK File: **sample_key_enc.key** included in the sample

— IV

— Output

Format: C source

File: euk aes128.c

Key name: euk_aes128

To use the above settings, you can load the Security Key Management Tool settings file rx261_SecureUpdate.skmt included in the sample.

The file rx261_SecureUpdate.skmt is in xml format and can be edited as text. For **{\$skmt_loc}** in rx261_SecureUpdate.skmt, substitute the path of the secure bootloader project folder.

Key Type **Key Data**

<input type="radio"/> DLM/AL	DLM-SSD	<input checked="" type="radio"/> AES	128 bits	<input type="radio"/> ARC4
<input type="radio"/> KUK		<input type="radio"/> RSA	2048 bits, public	<input type="radio"/> TDES
<input type="radio"/> OEM Root public		<input type="radio"/> ECC	secp256r1, public	
		<input type="radio"/> HMAC	SHA256-HMAC	

Wrapping Key

<input checked="" type="radio"/> UFPK	UFPK File :	<code>{\$skmt_loc}¥key¥sample.key</code>	<input type="button" value="Browse..."/>
	W-UFPK File :	<code>{\$skmt_loc}¥key¥sample.key_enc.key</code>	<input type="button" value="Browse..."/>
<input type="radio"/> KUK	KUK File :		<input type="button" value="Browse..."/>

IV

Generate random value
 Use specified value (16 hex bytes, big endian format) `00112233445566778899AABBCCDDEEFF`

Output

Format :	C Source	File :	<code>{\$skmt_loc}¥src¥genkey¥euk_aes128.c</code>	<input type="button" value="Browse..."/>
Address :	10000	Key name :	euk_aes128	

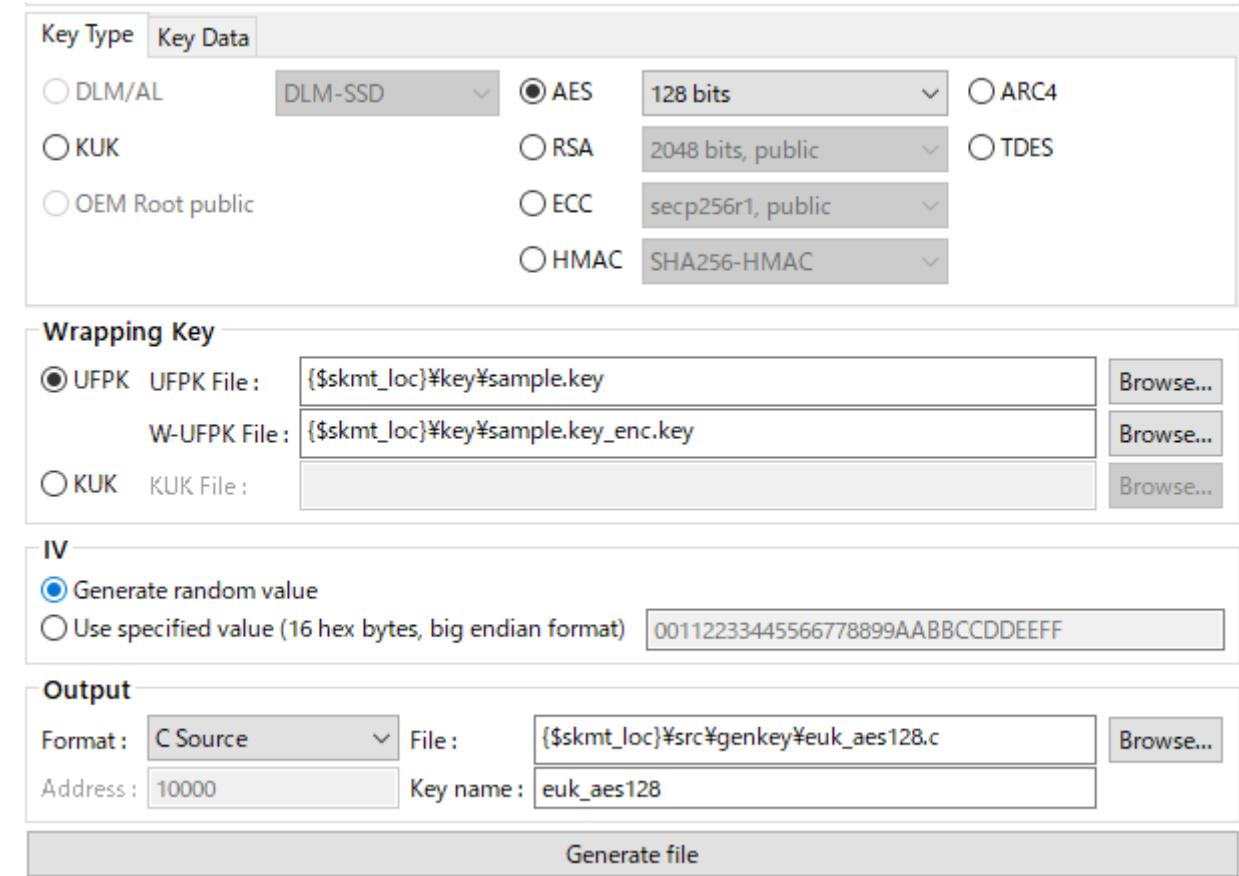


Figure 6.10 Generating a Key Data File (Wrap Key Tab)

Key Type **Key Data**

<input type="radio"/> File	<input type="button" value="Browse..."/>
<input checked="" type="radio"/> Raw	<code>0123456789abcdef0123456789abcdef</code>
<input type="radio"/> Random - Output File	<input type="button" value="Browse..."/>

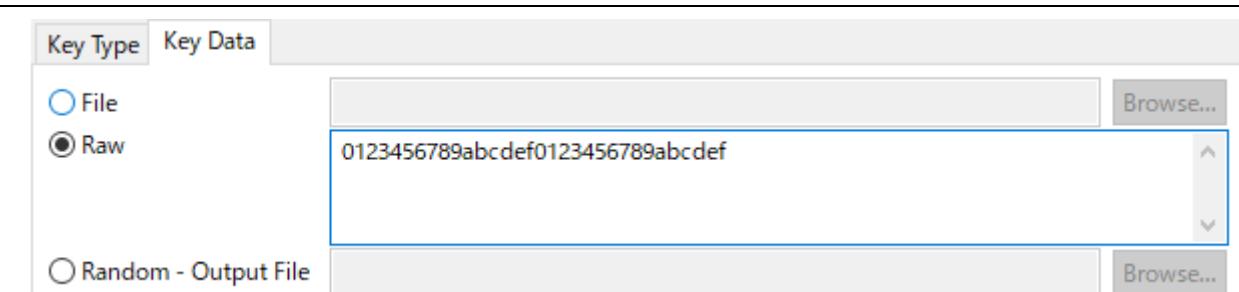


Figure 6.11 Generating a Key Data File (Wrap Key Tab – Key Data Tab)

(2) Building the Secure Bootloader Project

After importing the secure bootloader project (rx261_ek_rsip_secure_boot) into the e² studio workspace, place euk_aes128.c and euk_aes128.h generated in step (1) in the src folder of the rx261_ek_tsip_secure_boot project.

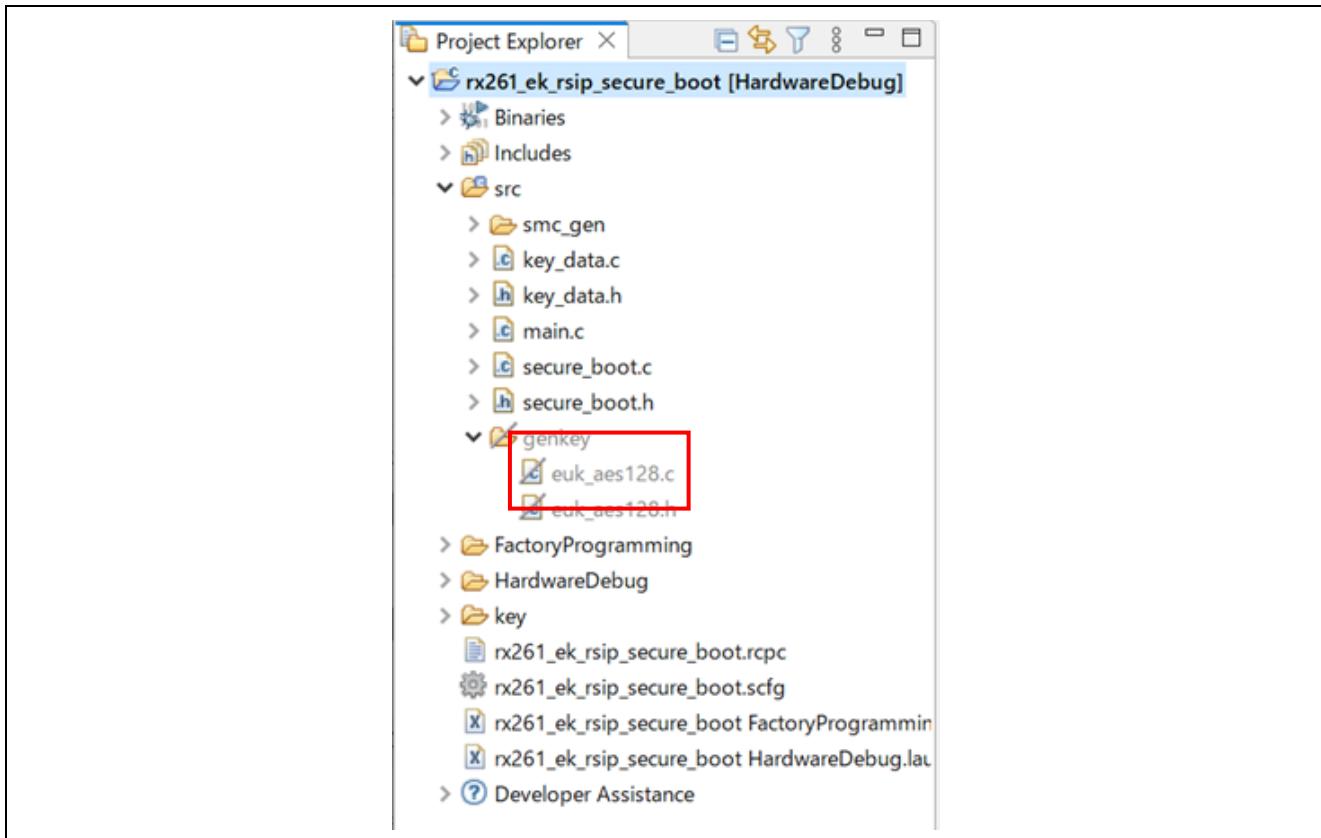


Figure 6.12 e²studio Project Explorer

Before building the secure boot project, build libraries to use in the project. Select "Library Key Injection" and "Library USB Memory" from the pull-down menu.

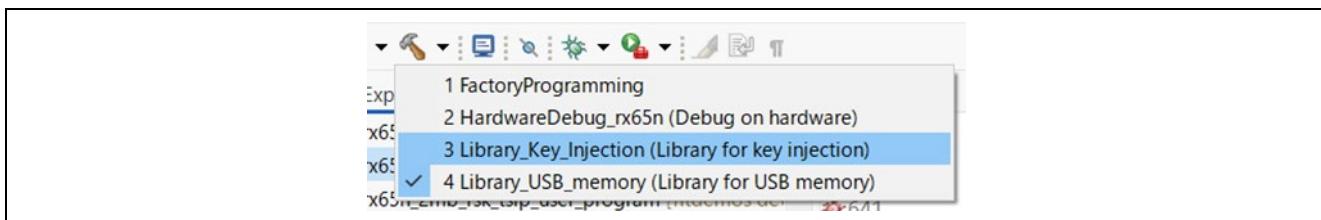


Figure 6.13 Building library files

Also, to use USB flash drive with the supported board, in Macro definition, set as below.

```
ENABLE_USB=1
```

After making the changes, build the project in e² studio.

(3) Building the Firmware Update Project

After importing the firmware update project (rx261_ek_rsip_user_program) into the e² studio workspace, use USB flash drive with the supported board, as in step (2), in Macro definition, set as below.

```
ENABLE_USB=1
```

After making the changes, build the project in e² studio.

(4) Encrypting the Firmware Update Program

- Command line version

Run the following command.

```
> skmt.exe /enctsip /mode "update" /ver "2" /prg
    "{$skmt_loc}..\rx261_ek_rsip_user_program\Release\rx261_ek_rsip_user_program.mot"
    /enckey "0123456789abcdef0123456789abcdef" /session_key
    "fedcba9876543210fedcba98765432100123456789abcdef0123456789abcdef"
    /iv_fw "55aa55aa55aa55aa55aa55aa55aa55aa"
    /startaddr "FFEB8300" /endaddr "FFFFFFFFFF" /filetype "bin" /flash_wsize 128
    /output "{$skmt_loc}\userprog.rsu"
```

For {\$skmt_loc}, substitute the path of the secure bootloader project folder.

The settings values of /startaddr and /endaddr differ depending on the MCU. Refer to the table below and enter the appropriate values.

Table 6-10 Specifying address for the MCU

MCU	Parameter		Address
	GUI	CLI	
RX261	Start address	/startaddr	FFFB8300
	End address	/endaddr	FFFFFFFFFF

- Standalone version

Enter the following values.

— **Output Image**

Select **Secure Update**.

— **Firmware Image**

Motorola S format file output by firmware update project (rx261_ek_rsip_user_program.mot)

— **Encrypted Address Range tab**

The values differ depending on the MCU. Refer to Table 6-10 and select the appropriate values to match the MCU.

— **IV tab**

Select **Use specified value** and enter “**55aa55aa55aa55aa55aa55aa55aa55aa**”.

— **RSU Header tab**

RSU header Ver: e

— **Output**

Format: Binary

File: userprog.rsu

To use the above settings, you can load the Security Key Management Tool settings file rx261_SecureUpdate.skmt included in the sample.

The file rx261_SecureUpdate.skmt is in xml format and can be edited as text. For **{\$skmt_loc}** in rx261_SecureUpdate.skmt, substitute the path of the secure bootloader project folder.

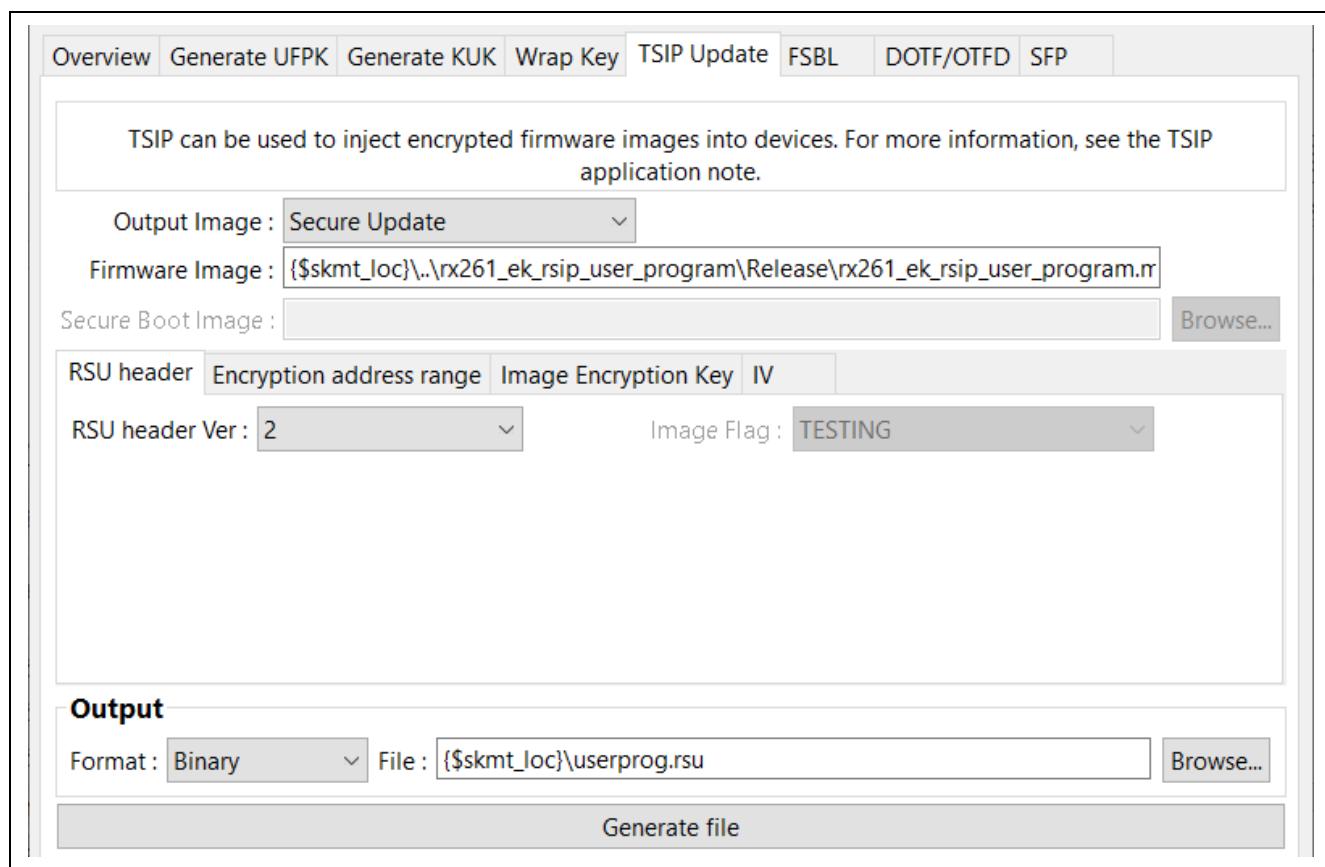


Figure 6.14 Generating an Encrypted File (TSIP UPDATE Tab – RSU header Tab)

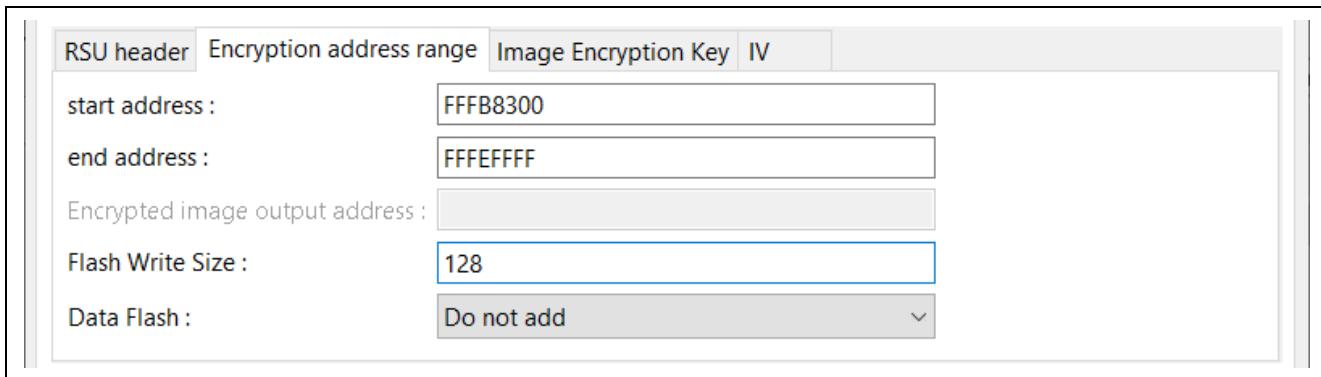


Figure 6.15 Generating an Encrypted File (TSIP UPDATE Tab – Encryption address range Tab)

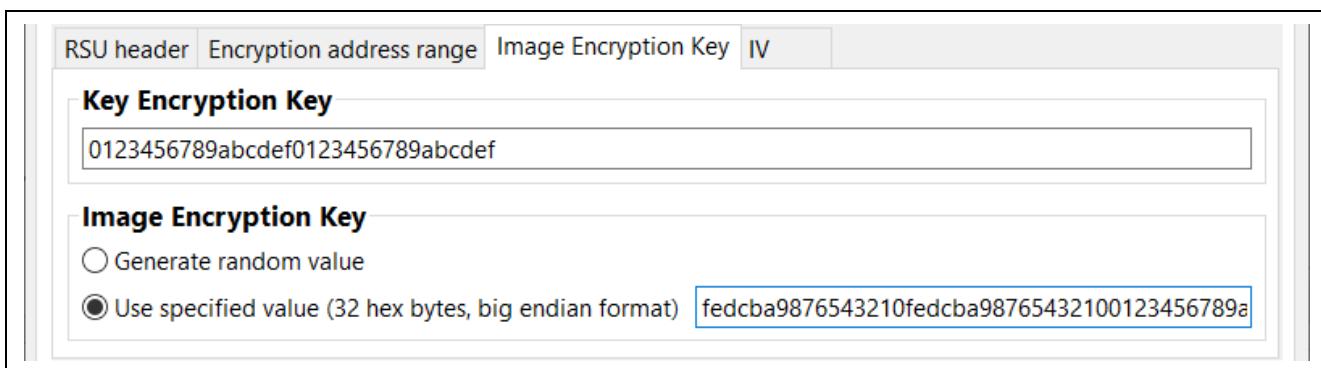


Figure 6.16 Generating an Encrypted File (TSIP UPDATE Tab – Image Encryption Key Tab)

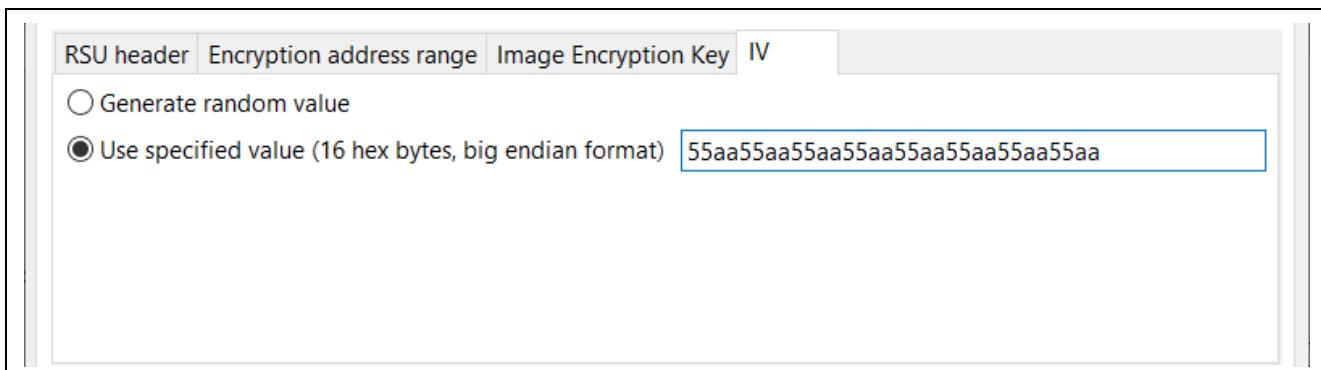


Figure 6.17 Generating an Encrypted File (TSIP UPDATE Tab – IV Tab)

Use the generated file (default file name: userprog.rsu) as shown below.

- When using UART, send the file with Tera Term.
- When using USB flash drive, store the file into the USB memory and attach it to the board.

(5) Starting the Terminal Emulator (Tera Term)

Make connections as shown in the applicable connection diagram in 6.2.1, Demo Project Setup, then launch Tera Term. Refer to Table 6-6, Tera Term settings to make serial setting.

(6) Running the Secure Bootloader Project

Select **rx261_ek_rsip_secure_boot** in e² studio's Project Explorer, then click the  button to execute the Secure Bootloader project.

If the on-chip flash memory of the board to be used has not been completely erased, the project will not run properly. Use the **Erase Chip** option in Renesas Flash Programmer to completely erase the flash memory.

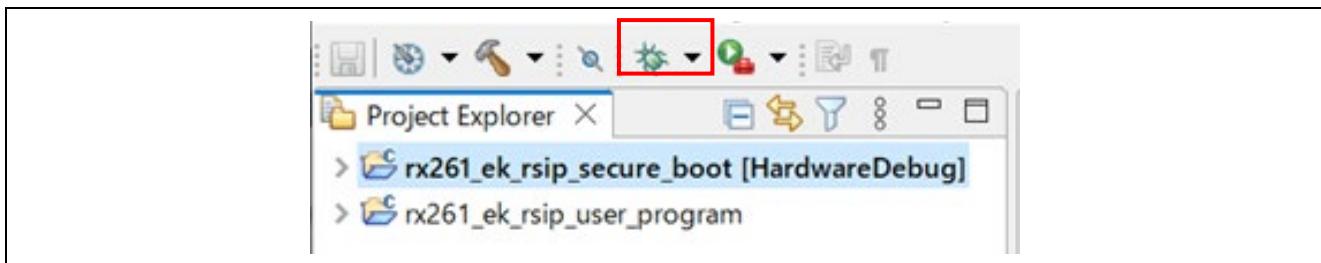


Figure 6.18 Running the Secure Bootloader Project

(7) Installing the Initial User Program

When the secure bootloader project is run, the encrypted user program is decrypted, and then the user program is executed.

If this works correctly, similar to the following is displayed in Tera Term.

```
HELLO!! this is boot program. ~
$ I was built in Sep 27 2024, 09:29:33.
Checking flash ROM status.
status = STATE_INJECT_KEY
===== generate user key index phase =====
generate aes128 user program mac key index: OK
===== install user key index phase =====
erase data flash(main)...OK
write data flash(main)...OK
erase data flash(mirror)...OK
write data flash(mirror)...OK
data flash setting OK.
Checking flash ROM status.
status = STATE_GET_IMAGE

==== Image updater [with buffer] ====
Erase buffer area...OK
send image(*.rsu) via UART.
```

Figure 6.19 Log Output of Secure Bootloader up to Wait for Updating Image

When using USB flash drive, attach USB memory to the board.

When using UART, in Tera Term, select **File > Send file...** and specify the firmware update program (**userprog.rsu** in the sample) that was encrypted in step (3). Then check the box for **Binary** under **Option** and click the **Open** button.

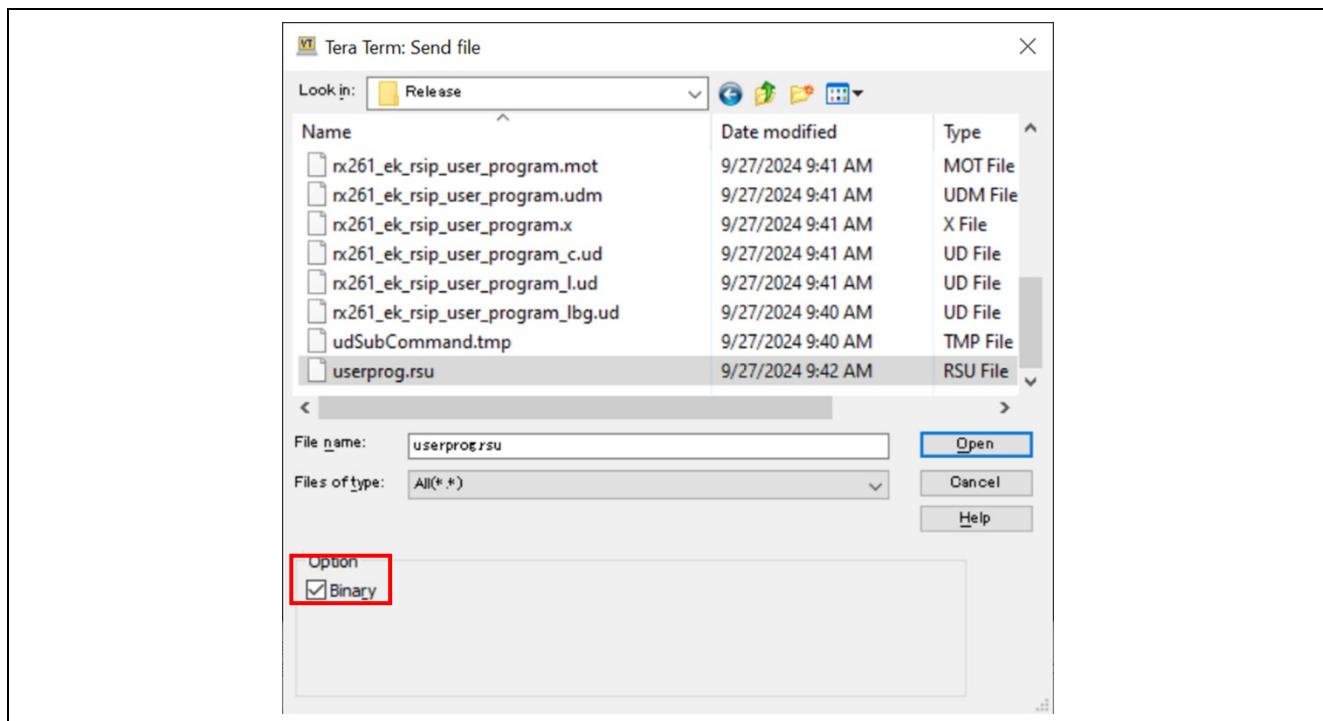


Figure 6.20 Tera Term: Send file Dialog Box

If the firmware update program is successfully programmed to the flash memory, output similar to the following is displayed in Tera Term.

```
erase data flash(main)...OK
write data flash(main)...OK
erase data flash(mirror)...OK
write data flash(mirror)...OK
data flash setting OK.
Checking flash ROM status.
status = STATE_ACTIVATE_IMAGE
update data flash
erase data flash(main)...OK
write data flash(main)...OK
erase data flash(mirror)...OK
write data flash(mirror)...OK
data flash setting OK.
activating image ... OK
software reset...
HELLO!! this is boot program. ~
$ I was built in Sep 27 2024, 09:29:33.
Checking flash ROM status.
status = STATE_EXEC_IMAGE

===== BootLoader [with buffer] =====
secure boot sequence: success.
execute image ...
HELLO!! this is user program. ~
$ I was built in Sep 27 2024, 09:40:38.
Version ver 1.00.
rsip@rx261
$
```

Figure 6.21 Log Output of Secure Bootloader When Writing Firmware Update Program

(8) Firmware Update Operation

Next, change some of the source code of the user program project and update the firmware. Change the version number in the user program main.c of the user program project from **ver 1.00** to **ver 1.01**. After building the project and generating a Motorola S format file as described in step (3), encrypt it as described in step (4).

```
/* Command prompt related */
#define PROMPT      ("rsip@rx261\r\n$ ")
#define VERSION     ("ver 1.00")
#define HELLO_MESSAGE ("HELLO!! this is user program. ~\r\n$ ")
```

Figure 6.22 Modified Location in main.c

Execute the **update** command. Install the firmware update program that has been updated to Ver 1.01 as described in step (7). After the upload finishes, the system reboots. According to the log output, a part of the output which is described in Figure 6.21 with boldfaced type is updated.

6.2.3.2 Execution Example of Inclusive Setup

This section describes how to create a Motorola S format file containing a secure bootloader and encrypted firmware update program and how to program it to a device using Renesas Flash Programmer.

(1) Generating an Encryption Key File for the Firmware Update

Generate a key file to encrypt the image. The procedure is same to 6.2.3.1 (1), Execution Example of Two-Step Setup.

(2) Building the Secure Bootloader Project

After importing the secure bootloader project (rx261_ek_rsip_secure_boot) into the e² studio workspace, place euk_aes128.c and euk_aes128.h generated in step (1) in the src folder of the rx261_ek_rsip_secure_boot project.

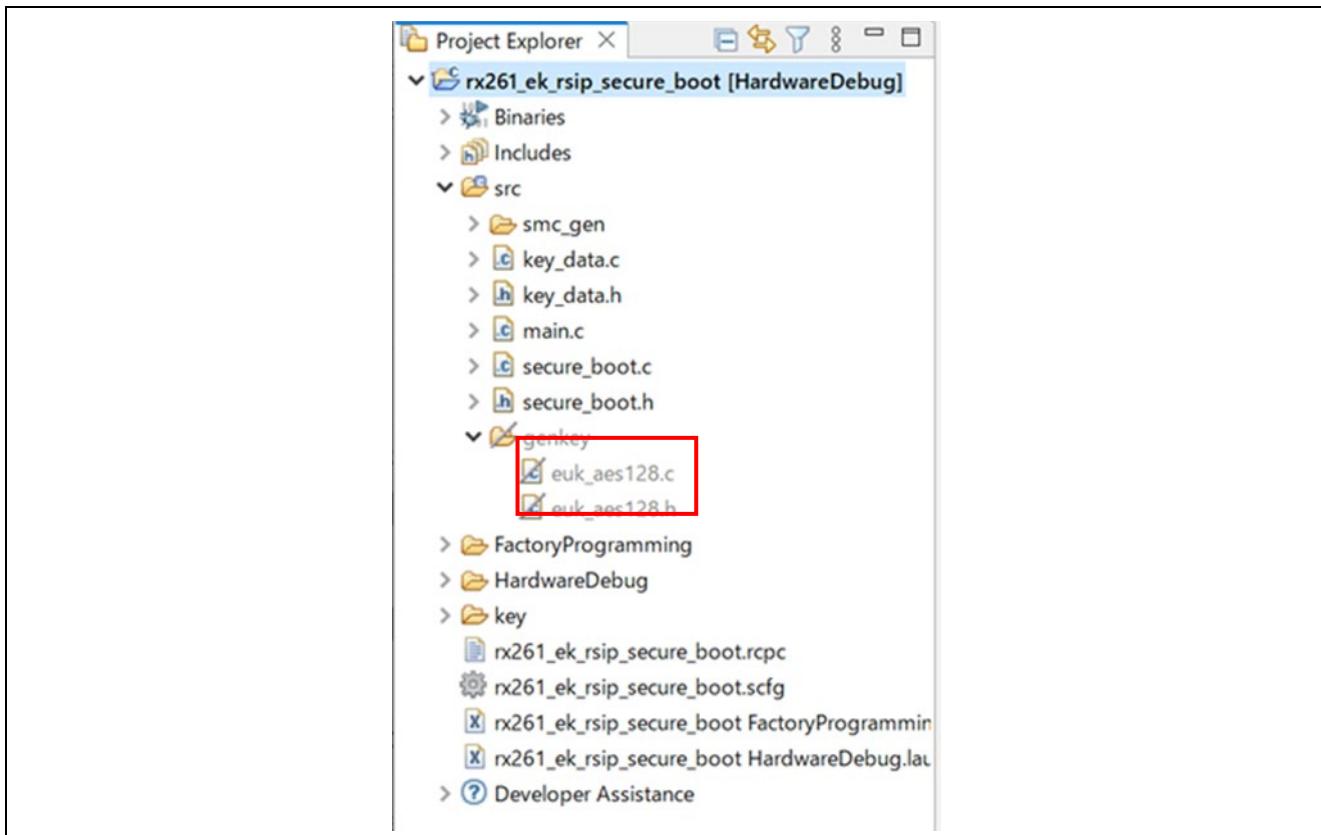


Figure 6.23 e²studio Project Explorer

Next, change the setting of **Build Configuration** to **Factory Programming**.

In e² studio's Project Explorer, right-click the secure update project and on the menu that appears select **Build Configuration > Activate > Factory Programming**.

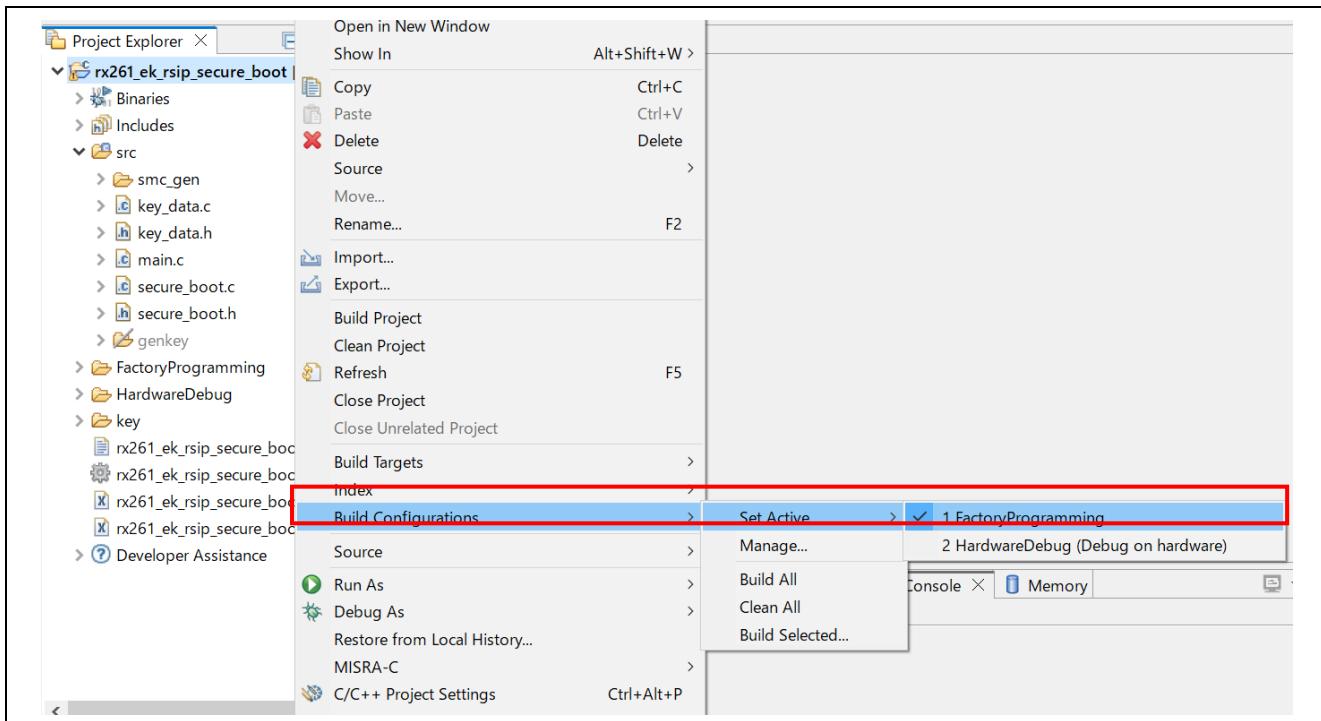


Figure 6.24 Changing Build Configuration to Factory Programming

After changing the **Build Configuration** setting to **Factory Programming**, build the secure bootloader project.

A definition of the build macro FACTORY_PROGRAMMING has been added from **Build Configuration: HardwareDebug**. Also, the section information is different.

(3) Building the Firmware Update Project

Build the project with the same operation of 6.2.3.1, Execution Example of Two-Step Setup (3).

(4) Encrypting the Firmware Update Program

- Command line version

Run the following command.

```
> skmt.exe /enctsip /mode "factory" /ver "2" /prg
  "{$skmt_loc}..\rx261_ek_rsig_user_program\Release\rx261_ek_rsig_user_program.mot"
  /prg_sb "{$skmt_loc}\FactoryProgramming\rx261_ek_rsig_secure_boot.mot "
  /enckey "0123456789abcdef0123456789abcdef" /session_key
  "fedcba9876543210fedcba98765432100123456789abcdef0123456789abcdef"
  /iv_fw "55aa55aa55aa55aa55aa55aa55aa55aa"
  /startaddr "FFFB8300" /endaddr "FFFEFFFF" /destaddr "FFFB8300"
  /filetype "mot" /flash_wsize 128 /output "{$skmt_loc}\userprog.mot"
```

For {\$skmt_loc}, substitute the path of the secure bootloader project folder.

The setting values of /startaddr, /endaddr, and /destaddr differ depending on the MCU. Refer to the table below when entering values.

Table 6-11 Specifying address for Each MCU

MCU	Parameter		Address
	GUI	CLI	
RX261	Start address	/startaddr	FFFB8300
	End address	/endaddr	FFFEFFFF
	Encrypted image output address	/destaddr	FFFB8300

- Standalone version

Enter the following values.

— **Output Image**

Select **Secure Update**.

— **Firmware Image**

Motorola S format file output by update project (rx261_ek_rsig_user_program.mot)

— **Encrypted Address Range tab**

The values differ depending on the MCU. Refer to Table 6-11 and select the appropriate values to match the MCU.

— **IV tab**

Select **Use specified value** and enter “55aa55aa55aa55aa55aa55aa55aa55aa”.

— **RSU Header tab**

RSU header Ver: 1

— Output

Format: Binary

File: userprog.mot

To use the above settings, you can load the Security Key Management Tool settings file rx261_SecureUpdate.skmt included in the sample.

The file rx261_SecureUpdate.skmt is in xml format and can be edited as text. For `{$skmt_loc}` in rx261_SecureUpdate.skmt, substitute the path of the secure bootloader project folder.

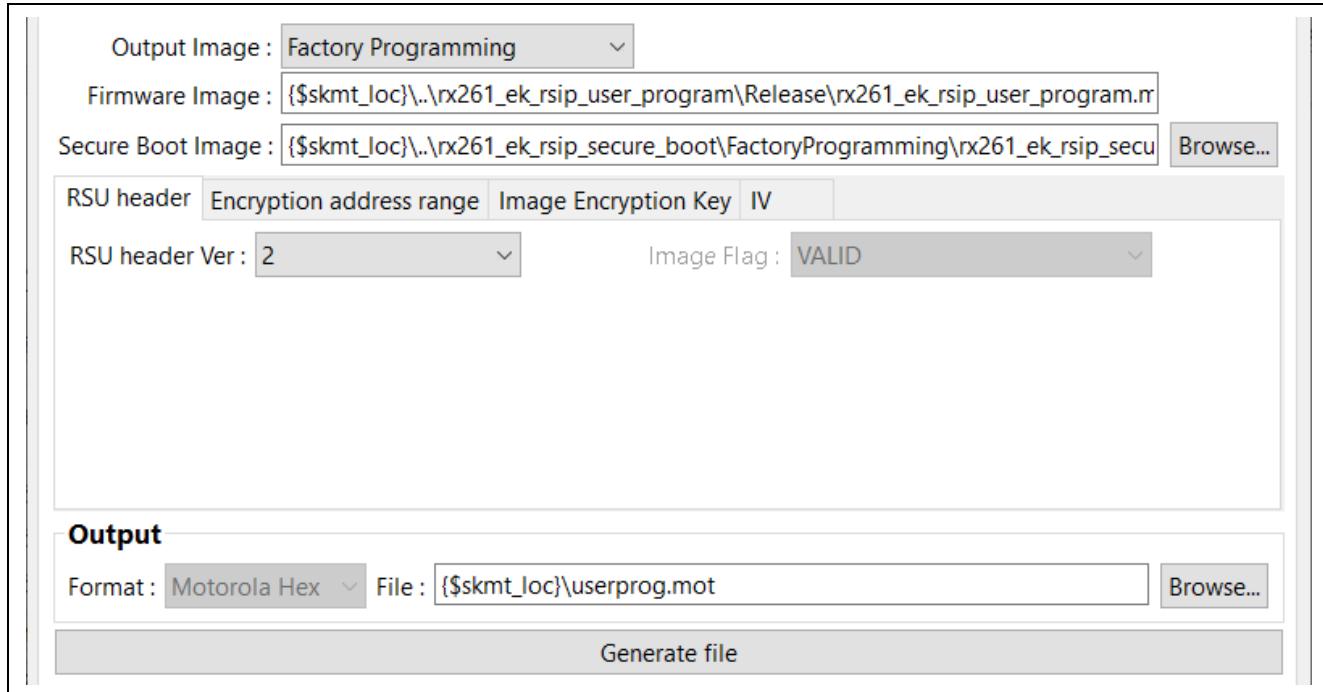


Figure 6.25 Generating an Encrypted File (TSIP UPDATE Tab – RSU header Tab)

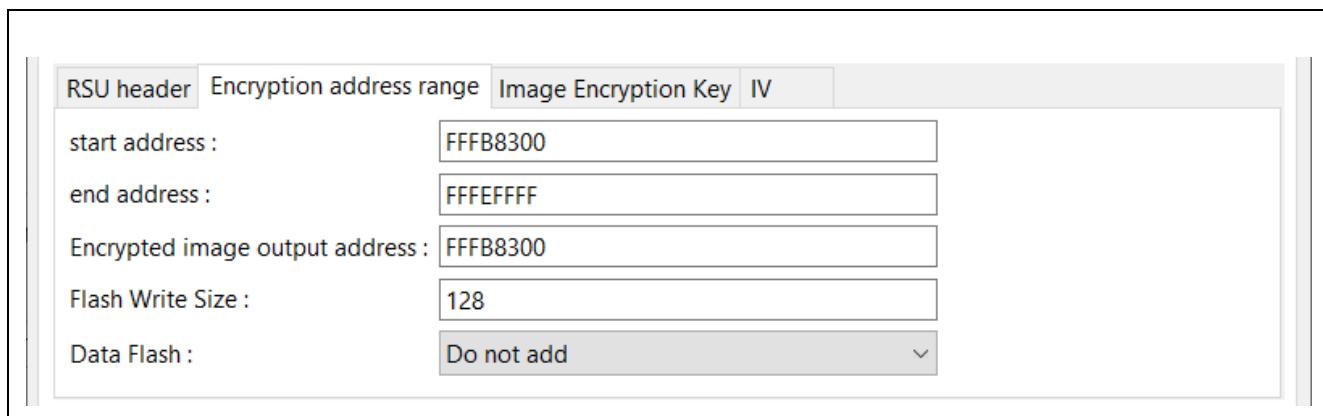


Figure 6.26 Generating an Encrypted File (TSIP UPDATE Tab – Encryption address range Tab)

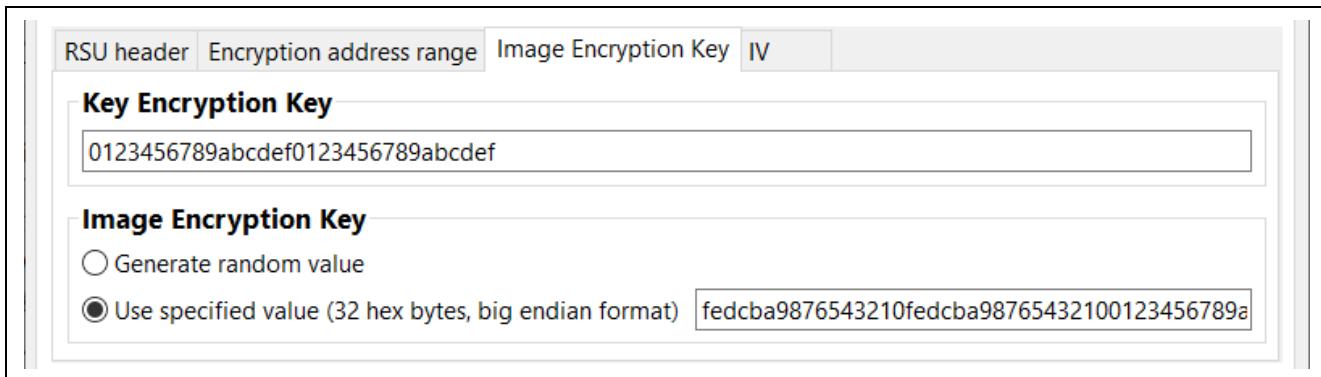


Figure 6.27 Generating an Encrypted File (TSIP UPDATE Tab – Image Encryption Key Tab)

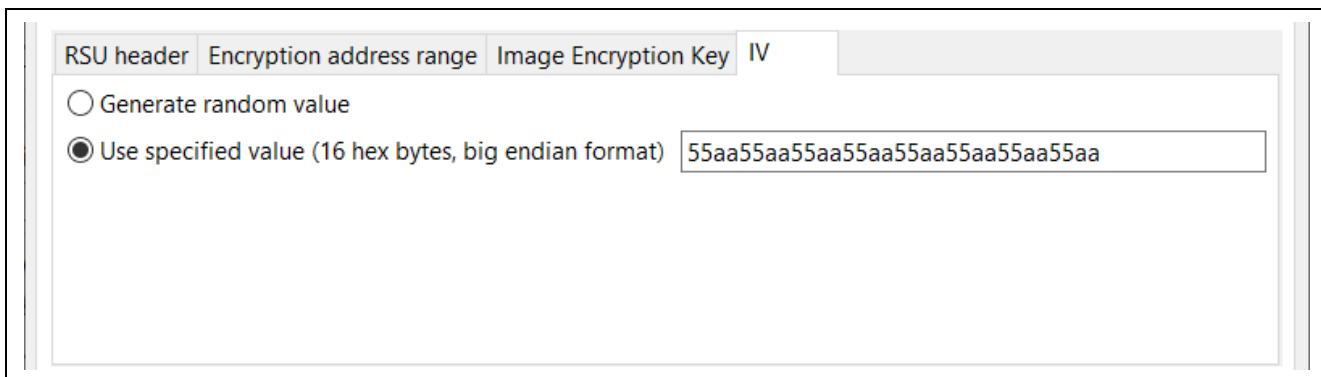


Figure 6.28 Generating an Encrypted File (TSIP UPDATE Tab – IV Tab)

Use Renesas Flash Programmer to write the Motorola S format file (default file name: userprog.mot) containing the encrypted user program and plaintext secure bootloader to the device.

(5) Starting the Terminal Emulator (Tera Term)

Make connections as shown in the applicable connection diagram in 6.2.1, Demo Project Setup, and launch Tera Term before writing the data using Renesas Flash Programmer.

(6) Writing Data Using Renesas Flash Programmer

Start Renesas Flash Programmer and create a project. After creating the project, click the **Tool Details** button on the **Connection Settings** tab. Then select the **Reset Settings** tab and set **Reset signal at Disconnection** to **Reset Pin as Hi-Z**.

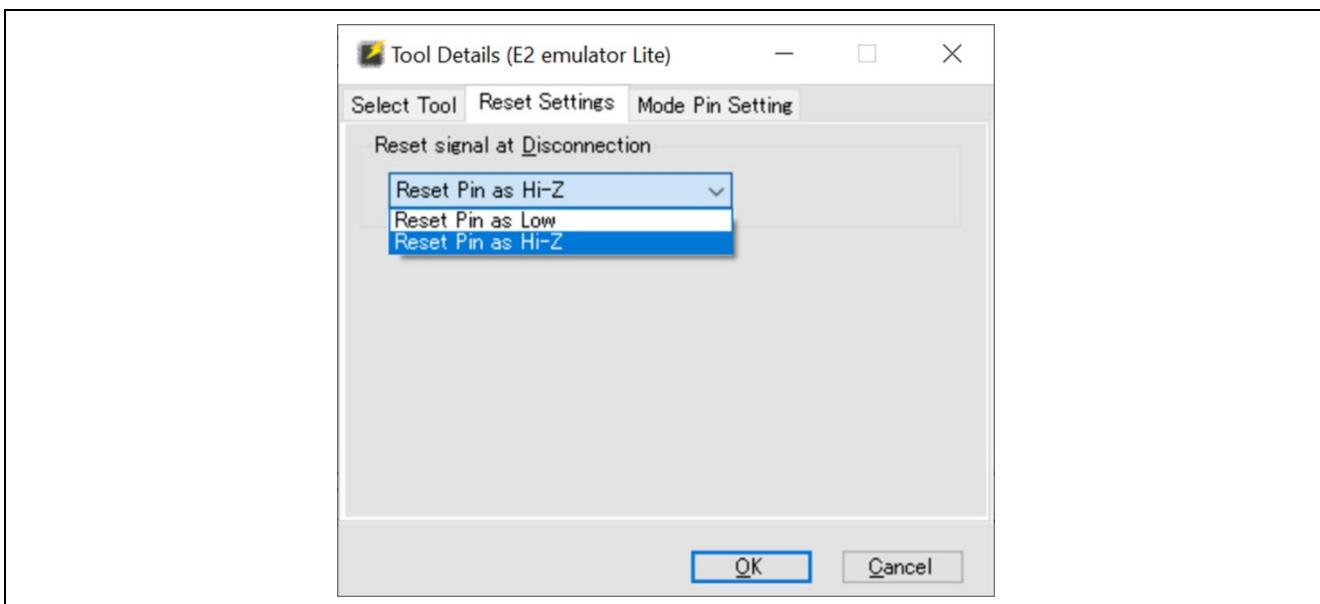


Figure 6.29 Renesas Flash Programmer Reset Settings Tab

If the data was successfully written to the device using Renesas Flash Programmer, the secure bootloader program runs on the device, decrypts the encrypted user program, and then runs the user program.

If the operation is successful, similar to the following is displayed.

```
HELLO!! this is boot program. ~  
$ I was built in Oct 2 2024, 14:10:22.  
Checking flash ROM status.  
status = STATE_INJECT_KEY  
===== generate user key index phase =====  
generate aes128 user program mac key index: OK  
===== install user key index phase =====  
erase data flash(main)...OK  
write data flash(main)...OK  
erase data flash(mirror)...OK  
write data flash(mirror)...OK  
data flash setting OK.  
Checking flash ROM status.  
status = STATE_GET_IMAGE
```

Figure 6.30 Log Output when Using Renesas Flash Programmer to Write Encrypted Firmware Update Program

After starting the device, the device executes the secure bootloader program and decrypt the encrypted firmware update program. Then, firmware update program is executed. In addition, decryption of the encrypted firmware update program is executed only in the first starting after writing the inclusive setup program.

If the operation is successful, similar to the following is displayed.

```
erase data flash(main)...OK
write data flash(main)...OK
erase data flash(mirror)...OK
write data flash(mirror)...OK
data flash setting OK.

Checking flash ROM status.
status = STATE_ACTIVATE_IMAGE
erase secure boot erase area...OK
update data flash
erase data flash(main)...OK
write data flash(main)...OK
erase data flash(mirror)...OK
write data flash(mirror)...OK
data flash setting OK.

activating image ... OK
software reset...

HELLO!! this is boot program. ~
$ I was built in Oct 2 2024, 14:10:22.

Checking flash ROM status.
status = STATE_EXEC_IMAGE

===== BootLoader [with buffer] =====
secure boot sequence: success.
execute image ...

HELLO!! this is user program. ~
$ I was built in Oct 2 2024, 14:13:48.

Version ver 1.00.

rsip@rx261
$
```

Figure 6.31 Log Output of First Starting after Inclusive Setup Program is Written

(7) Firmware Update Operation

For firmware update operation, refer to 6.2.3.1, Execution Example of Two-Step Setup (8).

6.2.4 Debugging Firmware Update Project

Since the firmware update project is started via Secure Boot, the reset vector (RESETVECT) of the project is placed at 0xFFFF7FFFC. For this reason, the firmware update project cannot be debugged as is. To debug the firmware update project, switch the debugging configuration to the HardwareDebug.

- How to switch debugger connection configuration:

1. Press ▼ next to e²studio menu build button  Select [HardwareDebug (Debug on hardware)] to start the build.

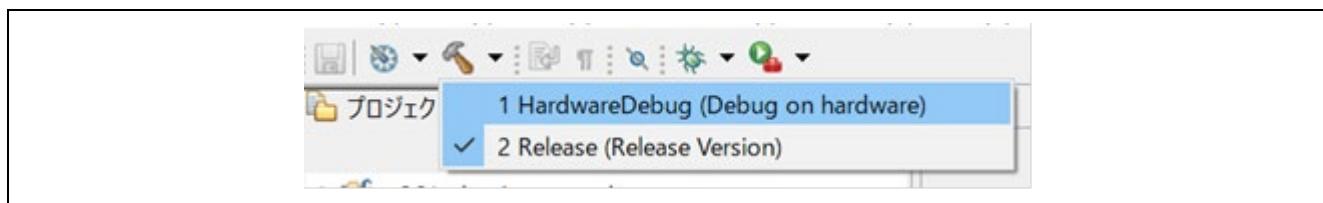


Figure 6.32 e²studio project menu

2. Once the build is complete, debugging with the debugger becomes possible.

Difference between "Release" configuration for release and "HardwareDebug" configuration for debugging
The addresses of the reset vector (RESETVECT) and interrupt vector (EXCEPTVECT) at build time are different between the "Release" and "HardwareDebug" configurations.

Table 6-12 User program project Vector setting address for each configuration

Symbol	Release	HardwareDebug
RESETVECT	0xFFFFEFFF	0xFFFFFFFF
EXCEPTVECT	0xFFFFEFF80	0xFFFFFFF80

6.2.5 Notes on Transition from Secure Bootloader Project to Firmware Update Project

When the transition from the secure bootloader program to the firmware update program takes place, the peripheral function settings of the secure bootloader program are inherited by the application. Therefore, in the secure bootloader project of the demo project is implemented as follows.

The API functions of the FIT modules (SCI, flash, TSIP, and USB) used by the secure bootloader program are closed when the bootloader terminates. All other settings are returned to their initial values when Smart Configurator is used.

If the user modifies the secure bootloader sample program for their own use, the peripheral function settings configured in the secure bootloader program are inherited by the application. It is therefore recommended either that the peripheral function settings be initialized before the transition from the secure bootloader program to the user program, or that common peripheral function settings be used for the application and the secure bootloader.

In other words, it is necessary to also consider the implementation of the secure bootloader program when creating applications.

7. Appendix

7.1 Confirmed Operation Environment

The operation of the driver has been confirmed in the following environment.

Table 7-1 Confirmed Operation Environment

Item	Description
Integrated development environment	Renesas Electronics e ² studio 2025-04 IAR Embedded Workbench for Renesas RX 5.10.01
C compiler	Renesas Electronics C/C++ Compiler for RX Family (CC-RX) V3.07.00 Compile options: The following option has been added to the default settings of the integrated development environment. -lang = c99
	GCC for Renesas RX 8.3.0.202311 Compile options: The following option has been added to the default settings of the integrated development environment. -std = gnu99
	IAR C/C++ Compiler for Renesas RX version 5.10.01 Compiler options: Default settings of the integrated development environment
Endian order	Big-endian or little-endian
Module version	Ver. 2.00
Board used	Renesas Evaluation Kit for RX261 (product No.: RTK5EK2610Sxxxxxx)

7.2 Troubleshooting

(1) Q: I added the FIT module to my project, but when I build it I get the error “Could not open source file ‘platform.h’.”

A: The FIT module may not have been added to the project properly. Refer to the documents listed below to confirm the method for adding FIT modules:

- Using CS+
Application Note: Adding Firmware Integration Technology Modules to CS+ Projects (R01AN1826)
- Using e² studio
Application Note: Adding Firmware Integration Technology Modules to Projects (R01AN1723)

When using the FIT module, the board support package FIT module (BSP module) must also be added to the project. Refer to the application note “RX Family: Board Support Package Module Using Firmware Integration Technology” (R01AN1685) for instructions for adding the BSP module.

(2) Q: I want to use the FIT Demos e² studio sample project on CS+.

A: Visit the following webpage for instructions:

Porting from the e² studio to CS+

> Convert an Existing Project to Create a New Project With CS+

<https://www.renesas.com/jp/ja/products/software-tools/tools/migration-tools/migration-e2studio-to-csplus.html>

Note: In step 5, the [Q0268002] dialog box may appear if the box next to “Backup the project composition files after conversion” is checked. If you click the **Yes** button in the [Q0268002] dialog box, you must then re-input the compiler include path.

7.3 Encrypted Key Generation in Dynamic Operation

The operations to generate encrypted key which is described in the 0 in Figure 3.4 User Key Wrapping Scheme during Key Injection and Key Updating, can be executed in dynamic operation on the device with using TSIP driver. The overview of the operation is shown in Figure 7.1.

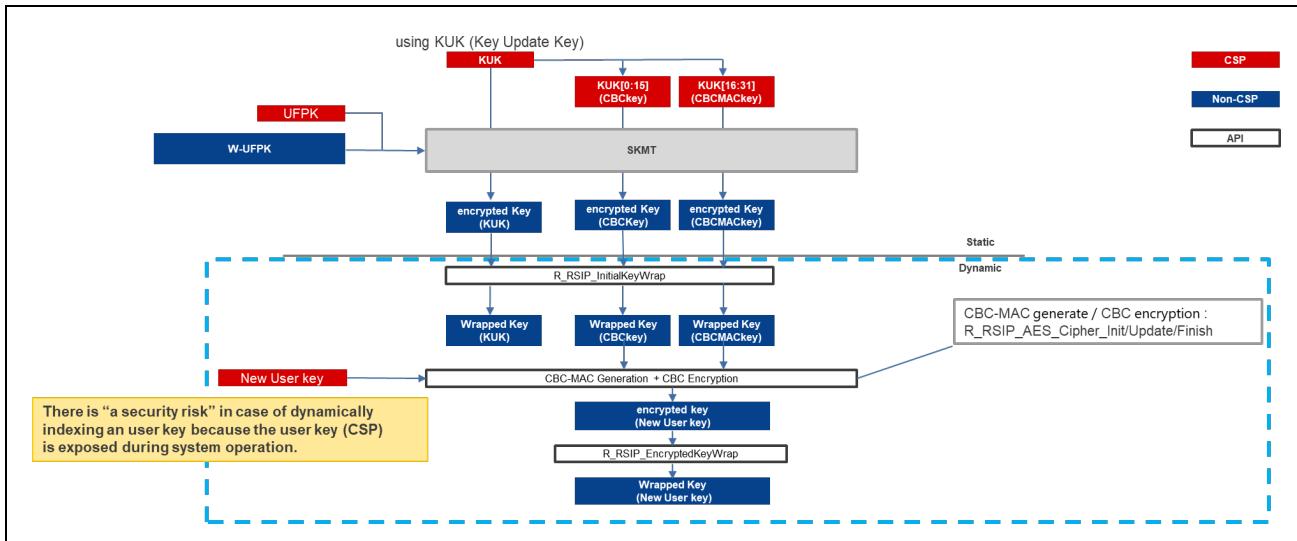


Figure 7.1 Encrypted Key Generation in Dynamic Operation

The specific operation which is described in light blue dotted line frame in Figure 7.1 is shown below with C language format.

```

uint8_t new_user_key[LEN] = "New User key"; /* LEN is byte size of User Key */
uint8_t encrypted_key[LEN+16];
uint32_t MAC[4] = 0; /* Zero initialization */

r_instance_ctrl_t p_ctrl;
rsip_wrapped_key_t CBCMACkey;
rsip_wrapped_key_t CBCkey;
rsip_wrapped_key_t KUK;
rsip_wrapped_key_t wrapped_key;

uint32_t i, j;
uint8_t work_input[16];
uint8_t work_output[16];
uint8_t work_dummy[16];
uint32_t dummy;

/*Inject KUK as CBCMACkey and CBCkey*/
CBCkey.type = RSIP_KEY_TYPE_AES_128;
CBCkey.p_value = "Address to store wrapped key value";
R_RSIP_InitialKeyWrap((rsip_ctrl_t *)p_ctrl, 'W-UFPK', 'Initial vector',
    'Encrypted key for 1st half of KUK as CBCkey', &CBCkey);
CBCMACkey.type = RSIP_KEY_TYPE_AES_128;
CBCMACkey.p_value = "Address to store wrapped key value";
R_RSIP_InitialKeyWrap((rsip_ctrl_t *)p_ctrl, 'W-UFPK', 'Initial vector',
    'Encrypted key for 2nd half of KUK as CBCMACkey', &CBCMACkey);

/* Inject KUK */
KUK.type = RSIP_KEY_TYPE_KUK;

```

```
KUK.p_value = "Address to store wrapped key value";
R_RSIP_InitialKeyWrap((rsip_ctrl_t *)p_ctrl, 'W-UFPK', 'Initial vector',
    'Encrypted key for KUK', &KUK);

/* AES-128 CBC-MAC */
R_RSIP_AES_Cipher_Init((rsip_ctrl_t *)p_ctrl, RSIP_AES_CIPHER_MODE_ECB_ENC,
&CBCMACkey, NULL);
for (i = 0; i < LEN; i += 16)
{
    for (j=0; j<16; j++)
    {
        work_input[j] = new_user_key[i+j] ^ mac[j];
    }
    R_RSIP_AES_Cipher_Update((rsip_ctrl_t *)p_ctrl, work_input, work_output, 16);
    memcpy(mac, work_output, 16);
}
R_RSIP_AES_Cipher_Finish((rsip_ctrl_t *)p_ctrl);

/* AES-128 CBC-Encryption */
R_RSIP_AES_Cipher_tInit((rsip_ctrl_t *)p_ctrl, RSIP_AES_CIPHER_MODE_CBC_ENC, &CBCkey, iv);
R_RSIP_AES_Cipher_Update((rsip_ctrl_t *)p_ctrl, new_user_key, encrypted_key, len);
R_RSIP_AES_Cipher_Update((rsip_ctrl_t *)p_ctrl , mac, &encrypted_key[len], 16);
R_RSIP_AES_Cipher_Finish((rsip_ctrl_t *)p_ctrl);

/* Generate wrapped key */
wrapped_key.type = "Key type of the wrapped key";
wrapped_key.p_value = "Address to store wrapped key value";
R_TSIP_EncryptedKeyWrap((rsip_ctrl_t *)p_ctrl , &KUK, iv, encrypted_key, &wrapped_key);
```

In above description, the words with single quotation marks such as 'W-UFPK' and 'Initial vector' mean the value assigned from key data files generated by SKMT, and the words with double quotation marks such as "New User Key" means the value assigned prepared by the user.

However, in the case of processing the above operation, unwrapped plain user key is exposed in the non secure area. Therefore, the operation is not recommended to implement. If you want to implement above operation, please examine the security risk and use only when the risk is allowable. In addition, please examine risk control methods such as deleting plain user key instantly after wrapping process is finished.

8. Reference Documents

User's Manual: Hardware

(The latest version can be downloaded from the Renesas Electronics website.)

Technical Updates/Technical News

(The latest information can be downloaded from the Renesas Electronics website.)

User's Manual: Development Environment

RX Family CC-RX Compiler User's Manual (R20UT3248)

(The latest version can be downloaded from the Renesas Electronics website.)

Revision History

Rev.	Date	Description	
		Page	Summary
1.00	Oct. 15, 2024	—	First edition issued
2.00	Jul. 31, 2025	—	<ul style="list-style-type: none">Changed specification of Key ManagementAdded support for ECDH KDFAdded HMAC Suspend/Resume APIs

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

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

1. Precaution against Electrostatic Discharge (ESD)

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

2. Processing at power-on

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

3. Input of signal during power-off state

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

4. Handling of unused pins

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

5. Clock signals

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

6. Voltage application waveform at input pin

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

7. Prohibition of access to reserved addresses

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

8. Differences between products

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

Notice

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

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

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

(Rev.5.0-1 October 2020)

Corporate Headquarters

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

Trademarks

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

Contact information

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