RX Family

TSIP (Trusted Secure IP) Module Firmware Integration Technology

Introduction

This document explains the usage of the software drivers for the Trusted Secure IP (TSIP) and TSIP-Lite modules on RX Family microcontrollers (MCUs). These software drivers are referred to collectively as the TSIP driver.

The TSIP 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/jp/ja/products/software-tools/software-os-middleware-driver/software-package/fit.html

The TSIP driver provides APIs for the cryptographic algorithms listed in Table 1 and Table 2 as well as for securely performing firmware updates.

Target Devices

TSIP: RX65N and RX651 groups, RX66N group, RX671 group, RX72M group, and RX72N group

TSIP-Lite: RX231 group, RX23W group, RX26T group, RX66T group, and RX72T group

For the specific product numbers of MCUs with TSIP functionality, refer to the user's manuals of the respective RX MCUs.

Table 1 TSIP Cryptographic Algorithms

Cipher Type		Algorithms
Asymmetric (public key) cryptography	Encryption/ decryption	RSAES-PKCS1-v1_5 (1024-/2048-/3072-/4096-bit)*1: RFC 8017
	Signature generation/ verification	RSASSA-PKCS1-v1_5 (1024-/2048-/3072-/4096-bit)*1: RFC 8017 ECDSA (ECC P-192/P-224/P-256/P-384): FIPS186-4
	Key generation	RSA (1024-/2048-bit) ECC P-192/P-224/P-256/P-384
Symmetric key cryptography	AES	AES (128-/256-bit) ECB/CBC/CTR: FIPS 197, SP800-38A
	DES	TDES*2 (56-/56x2-/56x3-bit) ECB/CBC: FIPS 46-3
	ARC4	ARC4*2 (2048-bit)
Hashing	SHA	SHA-1*2, SHA-256: FIPS 180-4
	MD5	MD5*2: RFC 1321
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 800-22.
TLS	TLS 1.2	TLS 1.2: RFC 5246 *3 Supported function: client function/server function Supported cipher suites (TLS 1.2): TLS_RSA_WITH_AES_128_CBC_SHA TLS_RSA_WITH_AES_256_CBC_SHA TLS_RSA_WITH_AES_128_CBC_SHA256 TLS_RSA_WITH_AES_256_CBC_SHA256 TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256 TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256 TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256 TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256
	TLS 1.3	TLS 1.3: RFC 8446 *4 Supported function: client function/server function Full Handshake/Resumption/0-RTT Supported cipher suites (TLS 1.3): TLS_AES_128_GCM_SHA256 TLS_AES_128_CCM_SHA256
Key update functions		AES, RSA, DES, ARC4, ECC, HMAC
Key exchange		ECDH P-256, ECDHE NIST P-512: SP800-56A, SP800-56C DH (2048-bit)
Key wrapping		AES (128-/256-bit)

Notes: 1. RSA (3072-/4096-bit) is supported for signature verification and modular exponentiation using public key only.
2. TDES, ARC4, SHA-1 and MD5 are obsoleted. It is not recommended to use these algorithms.
3. Applicable devices of the server function are the RX65N and RX651 groups, RX66N group, RX72M group, and RX72N group.
4. Applicable devices of the server function and Resumption and 0-RTT in the client function are the RX65N and RX651 groups, RX66N group, RX72M group, and RX72N group.

Table 2 TSIP-Lite Cryptographic Algorithms

Cipher Type		Algorithms
Symmetric key cryptography	AES	AES (128-/256-bit) ECB/CBC/CTR: FIPS 197, SP800-38A
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
Pseudo-random bit generation		SP 800-90A
Random number generation		Tested with SP 800-22.
Key update functions		AES
Key wrapping		AES (128-/256-bit)

Reference Information

RFC 2104: HMAC: Keyed-Hashing for Message Authentication (rfc-editor.org)
RFC 8017: PKCS #1: RSA Cryptography Specifications Version 2.2 (rfc-editor.org)
RFC 4253: The Secure Shell (SSH) Transport Layer Protocol
RFC 4543: The Use of Galois Message Authentication Code (GMAC) in IPsec ESP and AH (rfc-editor.org)
RFC 5246: The Transport Layer Security (TLS) Protocol Version 1.2 (rfc-editor.org)
RFC 8446: The Transport Layer Security (TLS) Protocol Version 1.3 (rfc-editor.org)

FIPS 46-3, Data Encryption Standard (DES) (withdrawn May 19, 2005) (nist.gov)
FIPS186-4: https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.186-4.pdf
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 800-38-C Recommendation for block cipher modes of operation: the CCM mode for authentication and confidentiality
NIST SP 800-38D, Recommendationfor Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC
NIST SP800-56A: Recommendation for Pair-Wise Key Establishment Schemes Using Discrete Lograrithm 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

Contents

1. Overview

1.1 Terminology

Terms used in this document are defined below. Note that the names used for some keys differ from those used in the “Diagram of Key Installation Process” appearing in the Trusted Secure IP section of the hardware manual of the MCU. Refer to “Diagram of Key Installation Process” (reproduced below as Figure 1.1) alongside the list below.

Table 1-1 Descriptions of Terms

Term	Description	Correspondence with Diagram of Key Installation Process
Key injection	Injecting a wrapped key into the device at the factory.	—
Key updating	Injecting a wraped 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, DES, ARC4, and HMAC, user keys are used as shared keys. For RSA and ECC, user keys are used as public keys and secret keys.	Key-1
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.	eKey-1
Wrapped key	Data consisting of an encrypted key that has been converted into a form that is usable by the TSIP 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.	Index-1 or Index-2
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.	Key-2
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.	Index-2
Key update key (KUK)	A key set by the user and used to generate an encrypted key from a user key during key updating. Thewrapped 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.	—

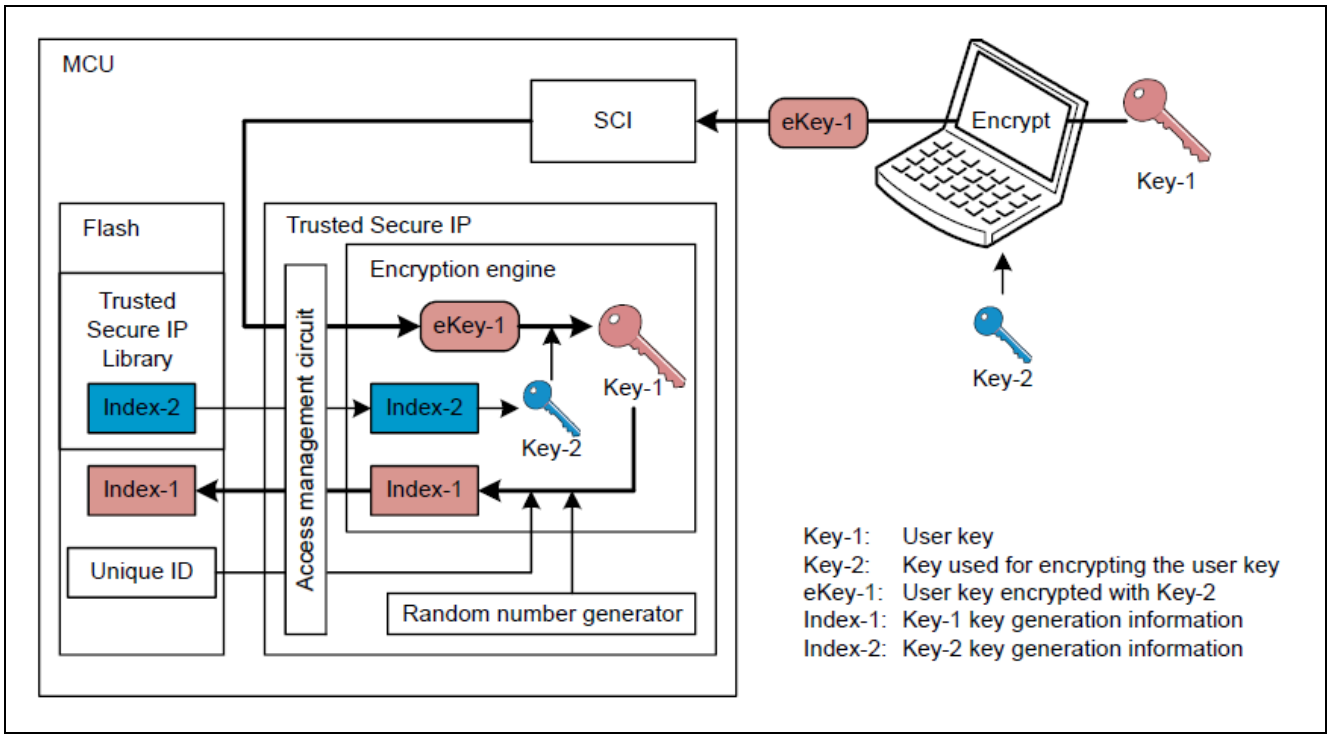


Figure 1.1 Diagram of Key Installation Process (Reproduced from Figure 52.4 in section 52, Trusted Secure IP (TSIP), in RX65N Group, RX651 Group, User's Manual: Hardware)

1.2 TSIP Overview

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

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

The TSIP restores from the wrapped key the encryption key used for cryptographic operations. This operation is performed internally by the TSIP. The wrapped key is tied to an HUK derived from a unique ID, making it device-specific. 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 TSIP driver.

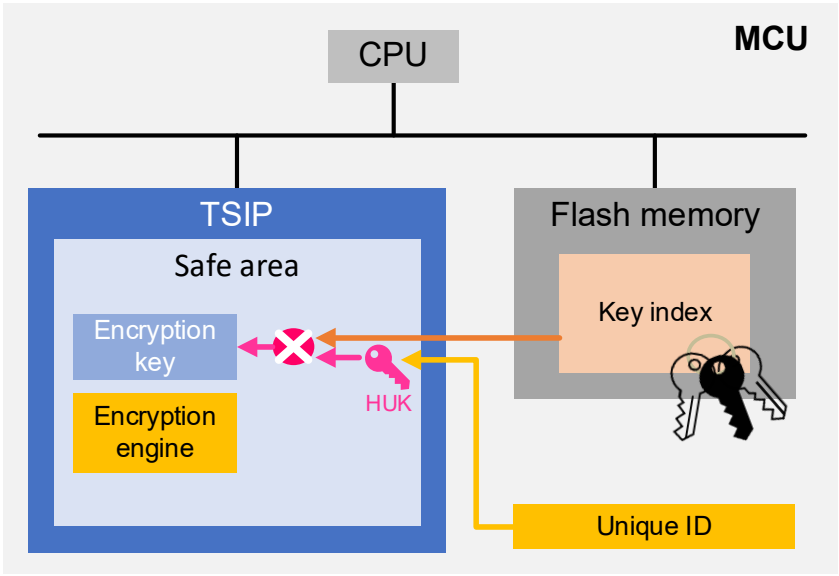


Figure 1.2 MCU Incorporating TSIP

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
Readme.txt	Readme
r20an0371jj0124-rx-tsip-security.pdf	TSIP driver application note (Japanese)
r20an0371ej0124-rx-tsip-security.pdf	TSIP driver application note (English)
r_tsip_rx_v1.24_sbom.spdx	SBOM information file(spdx)
r_tsip_rx_v1.24_sbom.json	SBOM information file(json)
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)
r01an5792jj0103-rx-tsip.pdf	AES cryptography project application note (Japanese)
r01an5880jj0104-rx-tsip.pdf	TLS cooperation function project application note (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)
r01an5792ej0103-rx-tsip.pdf	AES cryptography project application note (English)
r01an5880ej0104-rx-tsip.pdf	TLS cooperation function project application note (English)
FITModules	FIT module folder
r_tsip_rx_v1.24.zip	TSIP driver FIT module
r_tsip_rx_v1.24.xml	TSIP driver FIT module e^2 studio FIT plug-in XML file
r_tsip_rx_v1.24_extend.mdf	TSIP driver FIT module Smart Configurator configuration settings file
FITDemos	Demo project folder
rx*XXX*_*bbb*_tsip_sample*1*2**	Project showing how to write keys and use cryptographic APIs
rx*XXX*_*bbb*_tsip_secure_boot*1*2**	Secure boot/secure update implementation example
rx*XXX*_*bbb*_tsip_secure_boot*1*2**	Secure boot firmware
rx*XXX*_*bbb*_tsip_user_program*1*2**	User program following secure update
rx65n_2mb_rsk_tsip_aes_sample	AES cryptography project for RX65N
rx72n_ek_tsip_aes_sample	AES cryptography project for RX72N

Note: 1. "rx*XXX*" represents the RX group names.

2. "*bbb*" is the name of the supported board. Usually, USB and UART are available as IF for Update, but if the MCU does not support USB, only UART is supported. The project name which supports only UART includes _sci before _secure_boot and _user_program.
RSK : rsk (rsk(_tsip)_sci : only support UART)
MCB : mcb (mcb(_tsip)_sci : only support UART)
Envision Kit : ek

Table 1.3 lists the files and folders contained in the folder produced by unzipping r_tsip_rx_v.1.24.zip.

Table 1-3 File Structure

File/Directory (Bold) Names	Description
r_config	TSIP driver config file folder
r_tsip_rx_config.h	TSIP driver config file (default settings)
r_tsip_rx	TSIP driver FIT module folder
src	TSIP driver source code folder
targets	Folder containing program code dependent on specific MCU models
Group name*1	TSIP/TSIP-Lite driver folder There is a separate folder for each product group.
r_tsip_rx.c	TSIP system driver source code (required)
r_tsip_{algorithm}_rx.c	API source code for each cryptographic algorithm {algorithm} represents the name of the algorithm, such as aes or rsa.
r_tsip_rx_private.c	TSIP driver source code (Can be turned on/off with an option.)
r_tsip_rx_private.h	TSIP driver header file (Can be turned on/off with an option.)
iodefine	TSIP access header file storage folder
r_tsip_{Group name}_iodefine.h	TSIP access header file The folder name matches the name of the product group.
ip	TSIP access source code storage folder
r_tsip_rx_pxx.c	TSIP access source code “xx” in the file name represents a numeric value.
r_tsip_rx_subprcxx.c	TSIP access source code “xx” in the file name represents a numeric value.
r_tsip_rx_functionxxx.c	TSIP access source code (sub API) “xx” in the file name represents a numeric value.
s_flash.c	Key information file
doc	
ja	TSIP system driver source code (required)
r20an0371jj0124-rx-tsip-security.pdf	TSIP driver application note (Japanese)
en	TSIP driver source code (Can be turned on/off with an option.)
r20an0371ej0124-rx-tsip-security.pdf	TSIP driver application note (English)
r_tsip_rx_if.h	TSIP driver header file
readme.txt	Readme

1.4 Development Environment

The TSIP 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
   E1, E20, or 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. e^2 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 e^2 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_tsip_rx rev1.23

Compiler version: Renesas Electronics C/C++ Compiler Package for RX Family V3.07.00
(integrated development environment default settings with “-lang = c99” option added)

GCC for Renesas RX 14.2.0.202505
(integrated development environment default settings with “-std = gnu99” option added)

IAR C/C++ Compiler for Renesas RX version 5.10.01
(integrated development environment default settings)

Configuration options: default settings

ROM, RAM, and Stack Sizes				
		Memory Used		
Device	Category	Renesas Compiler	GCC	IAR Compiler
TSIP-Lite	ROM	56,600 bytes	64,712 bytes	60,159 bytes
	RAM	1,356 bytes	1,356 bytes	1,357 bytes
	Stack	184 bytes	196 bytes	164 bytes
TSIP	ROM	428,148 bytes	482,952 bytes	436,401 bytes
	RAM	7,980 bytes	8,064 bytes	7980 bytes
	Stack	1,644 bytes	1,120 bytes	1,376 bytes

The above table shows the code size which includes all the functions of TSIP driver. As a reference of the actual usage, ROM and RAM and the maximum stack usage for each functons are shown in below table. Note that sum of each size is different from actual build size because some of the APIs use same internal functios.

ROM, RAM, and Stack Sizes					
			Memory Used		
Device	Function	Category	Renesas Compiler	GCC	IAR Compiler
RX231	Common Function APIs	ROM	4,882 bytes	6,392 bytes	5,421 bytes
		RAM	1,356 bytes	1,356 bytes	1,356 bytes
		Stack	80 bytes	196 bytes	44 bytes
	Random Number Generation	ROM	362 bytes	224 bytes	429 bytes
		RAM	0 bytes	0 bytes	0 bytes
		Stack	28 bytes	36 bytes	20 bytes
	AES (128bit)	ROM	23,381 bytes	26,862 bytes	25,420 bytes
		RAM	112 bytes	112 bytes	108 bytes
		Stack	184 bytes	196 bytes	164 bytes
	AES (256bit)	ROM	24,413 bytes	28,032 bytes	26,379 bytes
		RAM	112 bytes	112 bytes	108 bytes
		Stack	184 bytes	196 bytes	164 bytes
	Key Wrap	ROM	6,370 bytes	6,624 bytes	6,350 bytes
		RAM	0 bytes	0 bytes	0 bytes
		Stack	112 bytes	80 bytes	72 bytes
	Firmware Update	ROM	2,577 bytes	2,848 bytes	2,552 bytes
		RAM	0 bytes	0 bytes	0 bytes
		Stack	40 bytes	60 bytes	12 bytes
RX65N	Common Function APIs	ROM	5,088 bytes	5,912 bytes	5,327 bytes
		RAM	1,415 bytes	1,400 bytes	1,400 bytes
		Stack	56 bytes	84 bytes	44 bytes
	Random Number Generation	ROM	286 bytes	216 bytes	458 bytes
		RAM	14 bytes	0 bytes	0 bytes
		Stack	28 bytes	4 bytes	20 bytes
	AES (128bit)	ROM	34,156 bytes	38,544 bytes	36,090 bytes
		RAM	206 bytes	192 bytes	192 bytes
		Stack	192 bytes	244 bytes	160 bytes
	AES (256bit)	ROM	27,145 bytes	30,104 bytes	28,262 bytes
		RAM	206 bytes	192 bytes	192 bytes
		Stack	192 bytes	188 bytes	160 bytes
	DES	ROM	6,816 bytes	7,592 bytes	6,244 bytes
		RAM	98 bytes	80 bytes	84 bytes
		Stack	64 bytes	64 bytes	48 bytes
	ARC4	ROM	5,442 bytes	6,376 bytes	5,965 bytes
		RAM	90 bytes	76 bytes	76 bytes
		Stack	68 bytes	60 bytes	40 bytes
	RSA	ROM	134,066 bytes	140,104 bytes	133,396 bytes
		RAM	6,723 bytes	6,724 bytes	6,724 bytes
		Stack	1,644 bytes	1,120 bytes	1,376 bytes
	ECC	ROM	40,569 bytes	51,688 bytes	43,120 bytes
		RAM	6,606 bytes	6,592 bytes	6,592 bytes
		Stack	360 bytes	272 bytes	344 bytes
	HASH	ROM	2,866 bytes	3,184 bytes	3,233 bytes

		RAM	14 bytes	0 bytes	0 bytes
		Stack	244 bytes	108 bytes	88 bytes
	HMAC	ROM	12,668 bytes	14,408 bytes	12,692 bytes
		RAM	178 bytes	164 bytes	164 bytes
		Stack	124 bytes	116 bytes	96 bytes
	DH	ROM	2,677 bytes	2,792 bytes	2,773 bytes
		RAM	14 bytes	0 bytes	0 bytes
		Stack	84 bytes	56 bytes	40 bytes
	ECDH	ROM	34,452 bytes	43,732 bytes	35,269 bytes
		RAM	6,630 bytes	6,616 bytes	6,616 bytes
		Stack	308 bytes	296 bytes	232 bytes
	Key Wrap	ROM	5,917 bytes	6,040 bytes	5,959 bytes
		RAM	14 bytes	0 bytes	0 bytes
		Stack	124 bytes	96 bytes	80 bytes
	TLS (Common to TLS 1.2 and TLS 1.3)	ROM	74,307 bytes	80,160 bytes	73,146 bytes
		RAM	7,806 bytes	7,792 bytes	7,792 bytes
		Stack	780 bytes	376 bytes	168 bytes
	TLS (TLS 1.2)	ROM	121,900 bytes	136,264 bytes	123,039 bytes
		RAM	7,806 bytes	7,792 bytes	7,792 bytes
		Stack	856 bytes	376 bytes	228 bytes
	TLS (TLS 1.3)	ROM	140,004 bytes	158,348 bytes	140,663 bytes
		RAM	6,642 bytes	6,628 bytes	6,628 bytes
		Stack	1,244 bytes	516 bytes	1,232 bytes
	Firmware Update	ROM	2,452 bytes	2,904 bytes	2,684 bytes
		RAM	14 bytes	0 bytes	0 bytes
		Stack	52 bytes	40 bytes	48 bytes

1.6 Sections

The TSIP driver uses the default sections.

In sample programs, the sections C_FIRMWARE_UPDATE_CONTROL_BLOCK, C_FIRMWARE_UPDATE_CONTROL_BLOCK_MIRROR and C_ENCRYPTED_KEY_BLOCK are used. If changes are required, edit the section setting.

When using the secure boot functionality, the sections BSECURE_BOOT*, PSECURE_BOOT, PSECURE_BOOT_ERASE, CSECURE_BOOT*, DSECURE_BOOT*, and RSECURE_BOOT* are used.

1.7 Performance

Performance information for TSIP-Lite drivers (RX231, RX23W, RX26T, RX66T, and RX72T) and TSIP drivers (RX65N, RX671, RX72M, and RX72N) for each device group is shown below.

Performance is measured in cycles of ICLK, the core clock. The operating clock (PCLKB) for TSIP-Lite and TSIP 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.

1.7.1 RX231

Table 1-4 Performance of Common APIs

API	Performance (Unit: Cycle)
R_TSIP_Open	7,400,000
R_TSIP_Close	430
R_TSIP_GetVersion	30
R_TSIP_GenerateAes128KeyIndex	4,000
R_TSIP_GenerateAes256KeyIndex	4,600
R_TSIP_GenerateAes128RandomKeyIndex	2,300
R_TSIP_GenerateAes256RandomKeyIndex	3,100
R_TSIP_GenerateRandomNumber	940
R_TSIP_GenerateUpdateKeyRingKeyIndex	4,500
R_TSIP_UpdateAes128KeyIndex	3,600
R_TSIP_UpdateAes256KeyIndex	4,100

Table 1-5 Firmware Verification Performance

API	Performance (Unit: Cycle)		
	2 KB Processing	4 KB Processing	6 KB Processing
R_TSIP_VerifyFirmwareMACInit	300	300	300
R_TSIP_VerifyFirmwareMACUpdate	11,000	23,000	34,000
R_TSIP_VerifyFirmwareMACFinal	710	710	710

Table 1-6 Performance of AES

API	Performance (Unit: Cycle)		
	16-Byte Processing	48-Byte Processing	80-Byte Processing
R_TSIP_Aes128EcbEncryptInit	1,400	1,400	1,400
R_TSIP_Aes128EcbEncryptUpdate	610	800	970
R_TSIP_Aes128EcbEncryptFinal	570	570	570
R_TSIP_Aes128EcbDecryptInit	1,400	1,400	1,400
R_TSIP_Aes128EcbDecryptUpdate	740	910	1,100
R_TSIP_Aes128EcbDecryptFinal	570	570	570
R_TSIP_Aes256EcbEncryptInit	1,600	1,600	1,600
R_TSIP_Aes256EcbEncryptUpdate	660	900	1,100
R_TSIP_Aes256EcbEncryptFinal	550	550	550
R_TSIP_Aes256EcbDecryptInit	1,700	1,700	1,700
R_TSIP_Aes256EcbDecryptUpdate	810	1,100	1,300
R_TSIP_Aes256EcbDecryptFinal	580	580	580
R_TSIP_Aes128CbcEncryptInit	1,400	1,400	1,400
R_TSIP_Aes128CbcEncryptUpdate	670	850	1,000
R_TSIP_Aes128CbcEncryptFinal	580	580	580
R_TSIP_Aes128CbcDecryptInit	1,400	1,400	1,400
R_TSIP_Aes128CbcDecryptUpdate	780	970	1,200
R_TSIP_Aes128CbcDecryptFinal	590	590	590
R_TSIP_Aes256CbcEncryptInit	1,700	1,700	1,700
R_TSIP_Aes256CbcEncryptUpdate	720	960	1,200
R_TSIP_Aes256CbcEncryptFinal	600	600	600
R_TSIP_Aes256CbcDecryptInit	1,700	1,700	1,700
R_TSIP_Aes256CbcDecryptUpdate	870	1,100	1,400
R_TSIP_Aes256CbcDecryptFinal	610	610	610

Table 1-7 Performance of AES-GCM

API	Performance (Unit: Cycle)		
	48-Byte Processing	64-Byte Processing	80-Byte Processing
R_TSIP_Aes128GcmEncryptInit	5,500	5,500	5,500
R_TSIP_Aes128GcmEncryptUpdate	2,900	3,400	3,900
R_TSIP_Aes128GcmEncryptFinal	1,300	1,300	1,300
R_TSIP_Aes128GcmDecryptInit	5,500	5,500	5,500
R_TSIP_Aes128GcmDecryptUpdate	2,500	2,600	2,700
R_TSIP_Aes128GcmDecryptFinal	2,100	2,100	2,100
R_TSIP_Aes256GcmEncryptInit	6,200	6,200	6,200
R_TSIP_Aes256GcmEncryptUpdate	3,000	3,500	4,000
R_TSIP_Aes256GcmEncryptFinal	1,400	1,400	1,400
R_TSIP_Aes256GcmDecryptInit	6,200	6,200	6,200
R_TSIP_Aes256GcmDecryptUpdate	2,500	2,700	2,800
R_TSIP_Aes256GcmDecryptFinal	2,200	2,100	2,100

Note: GCM performance was measured with parameters fixed as follows: ivec = 1,024 bits, AAD = 720 bits, and authentication tag = 128 bits.

Table 1-8 Performance of AES-CCM

API	Performance (Unit: Cycle)		
	48-Byte Processing	64-Byte Processing	80-Byte Processing
R_TSIP_Aes128CcmEncryptInit	2,700	2,700	2,700
R_TSIP_Aes128CcmEncryptUpdate	1,600	1,700	1,900
R_TSIP_Aes128CcmEncryptFinal	1,200	1,200	1,200
R_TSIP_Aes128CcmDecryptInit	2,500	2,500	2,500
R_TSIP_Aes128CcmDecryptUpdate	1,400	1,600	1,800
R_TSIP_Aes128CcmDecryptFinal	2,000	2,000	2,000
R_TSIP_Aes256CcmEncryptInit	3,000	3,000	3,000
R_TSIP_Aes256CcmEncryptUpdate	1,700	2,000	2,200
R_TSIP_Aes256CcmEncryptFinal	1,300	1,300	1,300
R_TSIP_Aes256CcmDecryptInit	3,000	3,000	3,000
R_TSIP_Aes256CcmDecryptUpdate	1,700	1,900	2,200
R_TSIP_Aes256CcmDecryptFinal	2,000	2,000	2,000

Note: CCM performance was measured with parameters fixed as follows: nonce = 104 bits, AAD = 880 bits, and MAC = 128 bits.

Table 1-9 Performance of AES-CMAC

API	Performance (Unit: Cycle)		
	48-Byte Processing	64-Byte Processing	80-Byte Processing
R_TSIP_Aes128CmacGenerateInit	910	910	910
R_TSIP_Aes128CmacGenerateUpdate	810	900	990
R_TSIP_Aes128CmacGenerateFinal	1,100	1,100	1,100
R_TSIP_Aes128CmacVerifyInit	900	910	910
R_TSIP_Aes128CmacVerifyUpdate	810	900	980
R_TSIP_Aes128CmacVerifyFinal	1,800	1,800	1,800
R_TSIP_Aes256CmacGenerateInit	1,300	1,300	1,200
R_TSIP_Aes256CmacGenerateUpdate	890	1,100	1,100
R_TSIP_Aes256CmacGenerateFinal	1,200	1,200	1,200
R_TSIP_Aes256CmacVerifyInit	1,300	1,300	1,300
R_TSIP_Aes256CmacVerifyUpdate	890	1,100	1,100
R_TSIP_Aes256CmacVerifyFinal	1,900	1,900	1,900

Table 1-10 Performance of AES Key Wrap

API	Performance (Unit: Cycle)	
	Wrap Target Key AES-128	Wrap Target Key AES-256
R_TSIP_Aes128KeyWrap	9,600	15,000
R_TSIP_Aes256KeyWrap	10,000	17,000
R_TSIP_Aes128KeyUnwrap	12,000	18,000
R_TSIP_Aes256KeyUnwrap	13,000	19,000

1.7.2 RX23W

Table 1-11 Performance of Common APIs

API	Performance (Unit: Cycle)
R_TSIP_Open	7,400,000
R_TSIP_Close	660
R_TSIP_GetVersion	38
R_TSIP_GenerateAes128KeyIndex	4,300
R_TSIP_GenerateAes256KeyIndex	5,000
R_TSIP_GenerateAes128RandomKeyIndex	2,400
R_TSIP_GenerateAes256RandomKeyIndex	3,300
R_TSIP_GenerateRandomNumber	1,100
R_TSIP_GenerateUpdateKeyRingKeyIndex	5,000
R_TSIP_UpdateAes128KeyIndex	3,900
R_TSIP_UpdateAes256KeyIndex	4,500

Table 1-12 Firmware Verification Performance

API	Performance (Unit: Cycle)		
	2 KB Processing	4 KB Processing	6 KB Processing
R_TSIP_VerifyFirmwareMACInit	340	340	340
R_TSIP_VerifyFirmwareMACUpdate	12,000	23,000	35,000
R_TSIP_VerifyFirmwareMACFinal	780	780	780

Table 1-13 Performance of AES

API	Performance (Unit: Cycle)		
	16-Byte Processing	48-Byte Processing	80-Byte Processing
R_TSIP_Aes128EcbEncryptInit	1,500	1,500	1,500
R_TSIP_Aes128EcbEncryptUpdate	730	910	1,100
R_TSIP_Aes128EcbEncryptFinal	660	660	660
R_TSIP_Aes128EcbDecryptInit	1,500	1,500	1,500
R_TSIP_Aes128EcbDecryptUpdate	850	1,000	1,200
R_TSIP_Aes128EcbDecryptFinal	670	670	670
R_TSIP_Aes256EcbEncryptInit	1,800	1,800	1,800
R_TSIP_Aes256EcbEncryptUpdate	760	1,000	1,300
R_TSIP_Aes256EcbEncryptFinal	650	650	650
R_TSIP_Aes256EcbDecryptInit	1,800	1,800	1,800
R_TSIP_Aes256EcbDecryptUpdate	930	1,200	1,400
R_TSIP_Aes256EcbDecryptFinal	670	670	670
R_TSIP_Aes128CbcEncryptInit	1,500	1,500	1,500
R_TSIP_Aes128CbcEncryptUpdate	810	990	1,200
R_TSIP_Aes128CbcEncryptFinal	680	680	680
R_TSIP_Aes128CbcDecryptInit	1,600	1,600	1,600
R_TSIP_Aes128CbcDecryptUpdate	920	1,100	1,300
R_TSIP_Aes128CbcDecryptFinal	700	700	700
R_TSIP_Aes256CbcEncryptInit	1,900	1,900	1,900
R_TSIP_Aes256CbcEncryptUpdate	850	1,100	1,300
R_TSIP_Aes256CbcEncryptFinal	680	580	580
R_TSIP_Aes256CbcDecryptInit	1,900	1,900	1,900
R_TSIP_Aes256CbcDecryptUpdate	1,000	1,200	1,500
R_TSIP_Aes256CbcDecryptFinal	700	700	700

Table 1-14 Performance of AES-GCM

API	Performance (Unit: Cycle)		
	48-Byte Processing	64-Byte Processing	80-Byte Processing
R_TSIP_Aes128GcmEncryptInit	6,200	6,200	6,200
R_TSIP_Aes128GcmEncryptUpdate	3,400	3,900	4,500
R_TSIP_Aes128GcmEncryptFinal	1,500	1,500	1,500
R_TSIP_Aes128GcmDecryptInit	6,200	6,200	6,200
R_TSIP_Aes128GcmDecryptUpdate	2,900	3,000	3,100
R_TSIP_Aes128GcmDecryptFinal	2,300	2,300	2,300
R_TSIP_Aes256GcmEncryptInit	6,900	6,900	6,900
R_TSIP_Aes256GcmEncryptUpdate	3,500	4,100	4,700
R_TSIP_Aes256GcmEncryptFinal	1,500	1,500	1,500
R_TSIP_Aes256GcmDecryptInit	6,900	6,900	6,900
R_TSIP_Aes256GcmDecryptUpdate	2,900	3,100	3,200
R_TSIP_Aes256GcmDecryptFinal	2,400	2,400	2,400

Note: GCM performance was measured with parameters fixed as follows: ivec = 1,024 bits, AAD = 720 bits, and authentication tag = 128 bits.

Table 1-15 Performance of AES-CCM

API	Performance (Unit: Cycle)		
	48-Byte Processing	64-Byte Processing	80-Byte Processing
R_TSIP_Aes128CcmEncryptInit	3,000	3,000	3,000
R_TSIP_Aes128CcmEncryptUpdate	1,800	1,900	2,100
R_TSIP_Aes128CcmEncryptFinal	1,400	1,400	1,400
R_TSIP_Aes128CcmDecryptInit	2,700	2,700	2,700
R_TSIP_Aes128CcmDecryptUpdate	1,700	1,800	2,000
R_TSIP_Aes128CcmDecryptFinal	2,200	2,200	2,200
R_TSIP_Aes256CcmEncryptInit	3,300	3,300	3,300
R_TSIP_Aes256CcmEncryptUpdate	2,000	2,200	2,500
R_TSIP_Aes256CcmEncryptFinal	1,500	1,500	1,500
R_TSIP_Aes256CcmDecryptInit	3,300	3,300	3,300
R_TSIP_Aes256CcmDecryptUpdate	1,800	2,100	2,300
R_TSIP_Aes256CcmDecryptFinal	2,300	2,300	2,300

Note: CCM performance was measured with parameters fixed as follows: nonce = 104 bits, AAD = 880 bits, and MAC = 128 bits.

Table 1-16 Performance of AES-CMAC

API	Performance (Unit: Cycle)		
	48-Byte Processing	64-Byte Processing	80-Byte Processing
R_TSIP_Aes128CmacGenerateInit	1,000	1,000	1,000
R_TSIP_Aes128CmacGenerateUpdate	950	1,000	1,100
R_TSIP_Aes128CmacGenerateFinal	1,300	1,300	1,300
R_TSIP_Aes128CmacVerifyInit	1,000	1,000	1,000
R_TSIP_Aes128CmacVerifyUpdate	940	1,000	1,100
R_TSIP_Aes128CmacVerifyFinal	2,000	2,000	2,000
R_TSIP_Aes256CmacGenerateInit	1,300	1,300	1,300
R_TSIP_Aes256CmacGenerateUpdate	1,000	1,100	1,300
R_TSIP_Aes256CmacGenerateFinal	1,300	1,300	1,300
R_TSIP_Aes256CmacVerifyInit	1,300	1,300	1,300
R_TSIP_Aes256CmacVerifyUpdate	1,000	1,100	1,300
R_TSIP_Aes256CmacVerifyFinal	2,100	2,100	2,100

Table 1-17 Performance of AES Key Wrap

API	Performance (Unit: Cycle)	
	Wrap Target Key AES-128	Wrap Target Key AES-256
R_TSIP_Aes128KeyWrap	11,000	17,000
R_TSIP_Aes256KeyWrap	11,000	17,000
R_TSIP_Aes128KeyUnwrap	13,000	20,000
R_TSIP_Aes256KeyUnwrap	14,000	21,000

1.7.3 RX26T

Table 1-18 Performance of Common APIs

API	Performance (Unit: Cycle)
R_TSIP_Open	7,400,000
R_TSIP_Close	270
R_TSIP_GetVersion	20
R_TSIP_GenerateAes128KeyIndex	4,000
R_TSIP_GenerateAes256KeyIndex	4,500
R_TSIP_GenerateAes128RandomKeyIndex	2,200
R_TSIP_GenerateAes256RandomKeyIndex	3,000
R_TSIP_GenerateRandomNumber	900
R_TSIP_GenerateUpdateKeyRingKeyIndex	4,500
R_TSIP_UpdateAes128KeyIndex	3,500
R_TSIP_UpdateAes256KeyIndex	4,100

Table 1-19 Firmware Verification Performance

API	Performance (Unit: Cycle)		
	2 KB Processing	4 KB Processing	6 KB Processing
R_TSIP_VerifyFirmwareMACInit	290	290	290
R_TSIP_VerifyFirmwareMACUpdate	11,000	23,000	34,000
R_TSIP_VerifyFirmwareMACFinal	650	650	650

Table 1-20 Performance of AES

API	Performance (Unit: Cycle)		
	16-Byte Processing	48-Byte Processing	80-Byte Processing
R_TSIP_Aes128EcbEncryptInit	1,300	1,300	1,300
R_TSIP_Aes128EcbEncryptUpdate	560	750	920
R_TSIP_Aes128EcbEncryptFinal	510	510	510
R_TSIP_Aes128EcbDecryptInit	1,300	1,300	1,300
R_TSIP_Aes128EcbDecryptUpdate	670	850	1,000
R_TSIP_Aes128EcbDecryptFinal	520	520	520
R_TSIP_Aes256EcbEncryptInit	1,500	1,600	1,600
R_TSIP_Aes256EcbEncryptUpdate	600	850	1,100
R_TSIP_Aes256EcbEncryptFinal	510	510	510
R_TSIP_Aes256EcbDecryptInit	1,600	1,600	1,600
R_TSIP_Aes256EcbDecryptUpdate	740	990	1,200
R_TSIP_Aes256EcbDecryptFinal	520	520	520
R_TSIP_Aes128CbcEncryptInit	1,300	1,300	1,300
R_TSIP_Aes128CbcEncryptUpdate	610	790	970
R_TSIP_Aes128CbcEncryptFinal	530	530	530
R_TSIP_Aes128CbcDecryptInit	1,300	1,300	1,300
R_TSIP_Aes128CbcDecryptUpdate	720	890	1,100
R_TSIP_Aes128CbcDecryptFinal	540	540	540
R_TSIP_Aes256CbcEncryptInit	1,600	1,700	1,600
R_TSIP_Aes256CbcEncryptUpdate	650	890	1,100
R_TSIP_Aes256CbcEncryptFinal	530	530	530
R_TSIP_Aes256CbcDecryptInit	1,600	1,600	1,600
R_TSIP_Aes256CbcDecryptUpdate	790	1,000	1,300
R_TSIP_Aes256CbcDecryptFinal	540	540	540

Table 1-21 Performance of AES-GCM

API	Performance (Unit: Cycle)		
	48-Byte Processing	64-Byte Processing	80-Byte Processing
R_TSIP_Aes128GcmEncryptInit	5,100	5,100	5,100
R_TSIP_Aes128GcmEncryptUpdate	2,600	3,100	3,600
R_TSIP_Aes128GcmEncryptFinal	1,300	1,300	1,300
R_TSIP_Aes128GcmDecryptInit	5,100	5,100	5,100
R_TSIP_Aes128GcmDecryptUpdate	2,200	2,300	2,400
R_TSIP_Aes128GcmDecryptFinal	2,000	2,000	2,000
R_TSIP_Aes256GcmEncryptInit	5,800	5,800	5,800
R_TSIP_Aes256GcmEncryptUpdate	2,700	3,200	3,700
R_TSIP_Aes256GcmEncryptFinal	1,300	1,300	1,300
R_TSIP_Aes256GcmDecryptInit	5,800	5,800	5,800
R_TSIP_Aes256GcmDecryptUpdate	2,300	2,400	2,500
R_TSIP_Aes256GcmDecryptFinal	2,000	2,000	2,000

Note: GCM performance was measured with parameters fixed as follows: ivec = 1,024 bits, AAD = 720 bits, and authentication tag = 128 bits.

Table 1-22 Performance of AES-CCM

API	Performance (Unit: Cycle)		
	48-Byte Processing	64-Byte Processing	80-Byte Processing
R_TSIP_Aes128CcmEncryptInit	2,500	2,500	2,500
R_TSIP_Aes128CcmEncryptUpdate	1,500	1,600	1,800
R_TSIP_Aes128CcmEncryptFinal	1,100	1,100	1,100
R_TSIP_Aes128CcmDecryptInit	2,300	2,300	2,300
R_TSIP_Aes128CcmDecryptUpdate	1,400	1,500	1,700
R_TSIP_Aes128CcmDecryptFinal	1,900	1,900	1,900
R_TSIP_Aes256CcmEncryptInit	2,800	2,800	2,800
R_TSIP_Aes256CcmEncryptUpdate	1,700	1,900	2,200
R_TSIP_Aes256CcmEncryptFinal	1,200	1,200	1,200
R_TSIP_Aes256CcmDecryptInit	2,900	2,900	2,900
R_TSIP_Aes256CcmDecryptUpdate	1,600	1,800	2,100
R_TSIP_Aes256CcmDecryptFinal	1,900	1,900	1,900

Note: CCM performance was measured with parameters fixed as follows: nonce = 104 bits, AAD = 880 bits, and MAC = 128 bits.

Table 1-23 Performance of AES-CMAC

API	Performance (Unit: Cycle)		
	48-Byte Processing	64-Byte Processing	80-Byte Processing
R_TSIP_Aes128CmacGenerateInit	870	870	870
R_TSIP_Aes128CmacGenerateUpdate	720	810	890
R_TSIP_Aes128CmacGenerateFinal	1,000	1,000	1,000
R_TSIP_Aes128CmacVerifyInit	880	880	880
R_TSIP_Aes128CmacVerifyUpdate	720	810	890
R_TSIP_Aes128CmacVerifyFinal	1,700	1,700	1,700
R_TSIP_Aes256CmacGenerateInit	1,200	1,200	1,200
R_TSIP_Aes256CmacGenerateUpdate	790	910	1,000
R_TSIP_Aes256CmacGenerateFinal	1,100	1,100	1,100
R_TSIP_Aes256CmacVerifyInit	1,200	1,200	1,200
R_TSIP_Aes256CmacVerifyUpdate	790	920	1,000
R_TSIP_Aes256CmacVerifyFinal	1,800	1,800	1,800

Table 1-24 Performance of AES Key Wrap

API	Performance (Unit: Cycle)	
	Wrap Target Key AES-128	Wrap Target Key AES-256
R_TSIP_Aes128KeyWrap	9,400	15,000
R_TSIP_Aes256KeyWrap	10,000	16,000
R_TSIP_Aes128KeyUnwrap	12,000	17,000
R_TSIP_Aes256KeyUnwrap	12,000	18,000

1.7.4 RX66T, RX72T

Table 1-25 Performance of Common APIs

API	Performance (Unit: Cycle)
R_TSIP_Open	7,400,000
R_TSIP_Close	290
R_TSIP_GetVersion	20
R_TSIP_GenerateAes128KeyIndex	3,900
R_TSIP_GenerateAes256KeyIndex	4,500
R_TSIP_GenerateAes128RandomKeyIndex	2,200
R_TSIP_GenerateAes256RandomKeyIndex	3,000
R_TSIP_GenerateRandomNumber	900
R_TSIP_GenerateUpdateKeyRingKeyIndex	4,500
R_TSIP_UpdateAes128KeyIndex	3,500
R_TSIP_UpdateAes256KeyIndex	4,000

Table 1-26 Firmware Verification Performance

API	Performance (Unit: Cycle)		
	2 KB Processing	4 KB Processing	6 KB Processing
R_TSIP_VerifyFirmwareMACInit	290	290	290
R_TSIP_VerifyFirmwareMACUpdate	11,000	23,000	34,000
R_TSIP_VerifyFirmwareMACFinal	650	650	650

Table 1-27 Performance of AES

API	Performance (Unit: Cycle)		
	16-Byte Processing	48-Byte Processing	80-Byte Processing
R_TSIP_Aes128EcbEncryptInit	1,300	1,300	1,300
R_TSIP_Aes128EcbEncryptUpdate	570	740	920
R_TSIP_Aes128EcbEncryptFinal	510	510	510
R_TSIP_Aes128EcbDecryptInit	1,300	1,300	1,300
R_TSIP_Aes128EcbDecryptUpdate	680	860	1,100
R_TSIP_Aes128EcbDecryptFinal	520	520	520
R_TSIP_Aes256EcbEncryptInit	1,600	1,600	1,600
R_TSIP_Aes256EcbEncryptUpdate	610	850	1,100
R_TSIP_Aes256EcbEncryptFinal	510	510	510
R_TSIP_Aes256EcbDecryptInit	1,600	1,600	1,600
R_TSIP_Aes256EcbDecryptUpdate	750	990	1,200
R_TSIP_Aes256EcbDecryptFinal	530	530	520
R_TSIP_Aes128CbcEncryptInit	1,300	1,300	1,300
R_TSIP_Aes128CbcEncryptUpdate	620	790	970
R_TSIP_Aes128CbcEncryptFinal	530	530	530
R_TSIP_Aes128CbcDecryptInit	1,300	1,300	1,300
R_TSIP_Aes128CbcDecryptUpdate	720	900	1,100
R_TSIP_Aes128CbcDecryptFinal	540	540	540
R_TSIP_Aes256CbcEncryptInit	1,600	1,600	1,600
R_TSIP_Aes256CbcEncryptUpdate	650	900	1,100
R_TSIP_Aes256CbcEncryptFinal	530	530	550
R_TSIP_Aes256CbcDecryptInit	1,600	1,700	1,600
R_TSIP_Aes256CbcDecryptUpdate	790	1,000	1,300
R_TSIP_Aes256CbcDecryptFinal	540	540	540

Table 1-28 Performance of AES-GCM

API	Performance (Unit: Cycle)		
	48-Byte Processing	64-Byte Processing	80-Byte Processing
R_TSIP_Aes128GcmEncryptInit	5,200	5,200	5,200
R_TSIP_Aes128GcmEncryptUpdate	2,600	3,100	3,600
R_TSIP_Aes128GcmEncryptFinal	1,300	1,300	1,200
R_TSIP_Aes128GcmDecryptInit	5,200	5,200	5,200
R_TSIP_Aes128GcmDecryptUpdate	2,200	2,300	2,400
R_TSIP_Aes128GcmDecryptFinal	2,000	2,000	2,000
R_TSIP_Aes256GcmEncryptInit	5,800	5,900	5,900
R_TSIP_Aes256GcmEncryptUpdate	2,700	3,200	3,700
R_TSIP_Aes256GcmEncryptFinal	1,300	1,300	1,300
R_TSIP_Aes256GcmDecryptInit	5,900	5,900	5,900
R_TSIP_Aes256GcmDecryptUpdate	2,300	2,400	2,600
R_TSIP_Aes256GcmDecryptFinal	2,100	2,100	2,100

Note: GCM performance was measured with parameters fixed as follows: ivec = 1,024 bits, AAD = 720 bits, and authentication tag = 128 bits.

Table 1-29 Performance of AES-CCM

API	Performance (Unit: Cycle)		
	48-Byte Processing	64-Byte Processing	80-Byte Processing
R_TSIP_Aes128CcmEncryptInit	2,500	2,500	2,500
R_TSIP_Aes128CcmEncryptUpdate	1,500	1,600	1,800
R_TSIP_Aes128CcmEncryptFinal	1,200	1,100	1,100
R_TSIP_Aes128CcmDecryptInit	2,300	2,300	2,300
R_TSIP_Aes128CcmDecryptUpdate	1,400	1,500	1,700
R_TSIP_Aes128CcmDecryptFinal	1,900	1,900	1,900
R_TSIP_Aes256CcmEncryptInit	2,900	2,900	2,900
R_TSIP_Aes256CcmEncryptUpdate	1,700	2,000	2,200
R_TSIP_Aes256CcmEncryptFinal	1,200	1,200	1,200
R_TSIP_Aes256CcmDecryptInit	2,900	2,900	2,900
R_TSIP_Aes256CcmDecryptUpdate	1,600	1,800	2,100
R_TSIP_Aes256CcmDecryptFinal	1,900	1,900	1,900

Note: CCM performance was measured with parameters fixed as follows: nonce = 104 bits, AAD = 880 bits, and MAC = 128 bits.

Table 1-30 Performance of AES-CMAC

API	Performance (Unit: Cycle)		
	48-Byte Processing	64-Byte Processing	80-Byte Processing
R_TSIP_Aes128CmacGenerateInit	880	880	880
R_TSIP_Aes128CmacGenerateUpdate	720	800	890
R_TSIP_Aes128CmacGenerateFinal	1,100	1,000	1,000
R_TSIP_Aes128CmacVerifyInit	880	880	880
R_TSIP_Aes128CmacVerifyUpdate	720	810	900
R_TSIP_Aes128CmacVerifyFinal	1,700	1,700	1,700
R_TSIP_Aes256CmacGenerateInit	1,200	1,200	1,200
R_TSIP_Aes256CmacGenerateUpdate	800	920	1,000
R_TSIP_Aes256CmacGenerateFinal	1,100	1,100	1,100
R_TSIP_Aes256CmacVerifyInit	1,200	1,200	1,200
R_TSIP_Aes256CmacVerifyUpdate	800	920	1,000
R_TSIP_Aes256CmacVerifyFinal	1,800	1,800	1,800

Table 1-31 Performance of AES Key Wrap

API	Performance (Unit: Cycle)	
	Wrap Target Key AES-128	Wrap Target Key AES-256
R_TSIP_Aes128KeyWrap	9,400	15,000
R_TSIP_Aes256KeyWrap	10,000	16,000
R_TSIP_Aes128KeyUnwrap	12,000	17,000
R_TSIP_Aes256KeyUnwrap	12,000	18,000

1.7.5 RX65N

Table 1-32 Performance of Common APIs

API	Performance (Unit: Cycle)
R_TSIP_Open	5,800,000
R_TSIP_Close	460
R_TSIP_GetVersion	34
R_TSIP_GenerateAes128KeyIndex	2,500
R_TSIP_GenerateAes256KeyIndex	2,700
R_TSIP_GenerateAes128RandomKeyIndex	1,400
R_TSIP_GenerateAes256RandomKeyIndex	2,100
R_TSIP_GenerateRandomNumber	650
R_TSIP_GenerateUpdateKeyRingKeyIndex	2,700
R_TSIP_UpdateAes128KeyIndex	2,300
R_TSIP_UpdateAes256KeyIndex	2,400

Table 1-33 Firmware Verification Performance

API	Performance (Unit: Cycle)		
	8 KB Processing	16 KB Processing	24 KB Processing
R_TSIP_VerifyFirmwareMACInit	230	230	230
R_TSIP_VerifyFirmwareMACUpdate	21,000	41,000	62,000
R_TSIP_VerifyFirmwareMACFinal	510	510	510

Table 1-34 Performance of AES

API	Performance (Unit: Cycle)		
	16-Byte Processing	48-Byte Processing	80-Byte Processing
R_TSIP_Aes128EcbEncryptInit	1,500	1,600	1,600
R_TSIP_Aes128EcbEncryptUpdate	510	650	830
R_TSIP_Aes128EcbEncryptFinal	430	430	430
R_TSIP_Aes128EcbDecryptInit	1,500	1,500	1,500
R_TSIP_Aes128EcbDecryptUpdate	570	710	890
R_TSIP_Aes128EcbDecryptFinal	460	460	460
R_TSIP_Aes256EcbEncryptInit	1,600	1,600	1,600
R_TSIP_Aes256EcbEncryptUpdate	510	660	840
R_TSIP_Aes256EcbEncryptFinal	450	450	450
R_TSIP_Aes256EcbDecryptInit	1,700	1,700	1,700
R_TSIP_Aes256EcbDecryptUpdate	580	730	920
R_TSIP_Aes256EcbDecryptFinal	460	460	460
R_TSIP_Aes128CbcEncryptInit	1,600	1,600	1,600
R_TSIP_Aes128CbcEncryptUpdate	570	710	890
R_TSIP_Aes128CbcEncryptFinal	460	460	470
R_TSIP_Aes128CbcDecryptInit	1,600	1,600	1,600
R_TSIP_Aes128CbcDecryptUpdate	630	770	950
R_TSIP_Aes128CbcDecryptFinal	480	480	480
R_TSIP_Aes256CbcEncryptInit	1,800	1,700	1,700
R_TSIP_Aes256CbcEncryptUpdate	590	740	920
R_TSIP_Aes256CbcEncryptFinal	490	480	480
R_TSIP_Aes256CbcDecryptInit	1,800	1,800	1,800
R_TSIP_Aes256CbcDecryptUpdate	660	800	990
R_TSIP_Aes256CbcDecryptFinal	490	490	490

Table 1-35 Performance of AES-GCM

API	Performance (Unit: Cycle)		
	48-Byte Processing	64-Byte Processing	80-Byte Processing
R_TSIP_Aes128GcmEncryptInit	5,400	5,400	5,400
R_TSIP_Aes128GcmEncryptUpdate	2,100	2,200	2,300
R_TSIP_Aes128GcmEncryptFinal	1,400	1,400	1,400
R_TSIP_Aes128GcmDecryptInit	5,300	5,300	5,300
R_TSIP_Aes128GcmDecryptUpdate	2,000	2,100	2,200
R_TSIP_Aes128GcmDecryptFinal	1,900	1,900	1,900
R_TSIP_Aes256GcmEncryptInit	5,300	5,300	5,300
R_TSIP_Aes256GcmEncryptUpdate	2,100	2,200	2,300
R_TSIP_Aes256GcmEncryptFinal	990	990	980
R_TSIP_Aes256GcmDecryptInit	5,300	5,300	5,300
R_TSIP_Aes256GcmDecryptUpdate	2,100	2,100	2,200
R_TSIP_Aes256GcmDecryptFinal	1,500	1,500	1,500

Note: GCM performance was measured with parameters fixed as follows: ivec = 1,024 bits, AAD = 720 bits, and authentication tag = 128 bits.

Table 1-36 Performance of AES-CCM

API	Performance (Unit: Cycle)		
	48-Byte Processing	64-Byte Processing	80-Byte Processing
R_TSIP_Aes128CcmEncryptInit	3,000	3,000	3,000
R_TSIP_Aes128CcmEncryptUpdate	1,100	1,200	1,300
R_TSIP_Aes128CcmEncryptFinal	910	910	910
R_TSIP_Aes128CcmDecryptInit	3,100	3,100	3,100
R_TSIP_Aes128CcmDecryptUpdate	1,000	1,100	1,200
R_TSIP_Aes128CcmDecryptFinal	1,500	1,500	1,500
R_TSIP_Aes256CcmEncryptInit	2,400	2,400	2,400
R_TSIP_Aes256CcmEncryptUpdate	1,200	1,300	1,400
R_TSIP_Aes256CcmEncryptFinal	950	950	950
R_TSIP_Aes256CcmDecryptInit	2,400	2,400	2,400
R_TSIP_Aes256CcmDecryptUpdate	1,100	1,200	1,200
R_TSIP_Aes256CcmDecryptFinal	1,600	1,600	1,600

Note: CCM performance was measured with parameters fixed as follows: nonce = 104 bits, AAD = 880 bits, and MAC = 128 bits.

Table 1-37 Performance of AES-CMAC

API	Performance (Unit: Cycle)		
	48-Byte Processing	64-Byte Processing	80-Byte Processing
R_TSIP_Aes128CmacGenerateInit	1,200	1,200	1,200
R_TSIP_Aes128CmacGenerateUpdate	640	690	730
R_TSIP_Aes128CmacGenerateFinal	780	780	780
R_TSIP_Aes128CmacVerifyInit	1,200	1,100	1,100
R_TSIP_Aes128CmacVerifyUpdate	650	700	740
R_TSIP_Aes128CmacVerifyFinal	1,400	1,400	1,400
R_TSIP_Aes256CmacGenerateInit	1,300	1,300	1,300
R_TSIP_Aes256CmacGenerateUpdate	680	720	770
R_TSIP_Aes256CmacGenerateFinal	820	820	820
R_TSIP_Aes256CmacVerifyInit	1,300	1,300	1,300
R_TSIP_Aes256CmacVerifyUpdate	670	720	760
R_TSIP_Aes256CmacVerifyFinal	1,400	1,400	1,400

Table 1-38 Performance of AES Key Wrap

API	Performance (Unit: Cycle)	
	Wrap Target Key AES-128	Wrap Target Key AES-256
R_TSIP_Aes128KeyWrap	6,900	11,000
R_TSIP_Aes256KeyWrap	7,200	11,000
R_TSIP_Aes128KeyUnwrap	7,700	12,000
R_TSIP_Aes256KeyUnwrap	8,000	12,000

Table 1-39 Performance of Common APIs (TDES Wrapped Key Generation)

API	Performance (Unit: Cycle)
R_TSIP_GenerateTdesKeyIndex	2,700
R_TSIP_GenerateTdesRandomKeyIndex	2,000
R_TSIP_UpdateTdesKeyIndex	2,400

Table 1-40 Performance of TDES

API	Performance (Unit: Cycle)		
	16-Byte Processing	48-Byte Processing	80-Byte Processing
R_TSIP_TdesEcbEncryptInit	990	990	990
R_TSIP_TdesEcbEncryptUpdate	550	790	1,000
R_TSIP_TdesEcbEncryptFinal	440	440	440
R_TSIP_TdesEcbDecryptInit	1,000	1,000	1,000
R_TSIP_TdesEcbDecryptUpdate	580	820	1,100
R_TSIP_TdesEcbDecryptFinal	450	450	450
R_TSIP_TdesCbcEncryptInit	1,100	1,100	1,100
R_TSIP_TdesCbcEncryptUpdate	620	870	1,200
R_TSIP_TdesCbcEncryptFinal	470	470	470
R_TSIP_TdesCbcDecryptInit	1,100	1,100	1,100
R_TSIP_TdesCbcDecryptUpdate	640	890	1,200
R_TSIP_TdesCbcDecryptFinal	490	480	490

Table 1-41 Performance of Common APIs (ARC4 Wrapped Key Generation)

API	Performance (Unit: Cycle)
R_TSIP_GenerateArc4KeyIndex	4,400
R_TSIP_GenerateArc4RandomKeyIndex	10,000
R_TSIP_UpdateArc4KeyIndex	4,200

Table 1-42 Performance of ARC4

API	Performance (Unit: Cycle)		
	16-Byte Processing	48-Byte Processing	80-Byte Processing
R_TSIP_Arc4EncryptInit	2,100	2,100	2,100
R_TSIP_Arc4EncryptUpdate	480	610	790
R_TSIP_Arc4EncryptFinal	330	330	330
R_TSIP_Arc4DecryptInit	2,100	2,100	2,100
R_TSIP_Arc4DecryptUpdate	480	620	800
R_TSIP_Arc4DecryptFinal	330	330	330

Table 1-43 Performance of Common APIs (RSA Wrapped Key Generation)

API	Performance (Unit: Cycle)
R_TSIP_GenerateRsa1024PublicKeyIndex	37,000
R_TSIP_GenerateRsa1024PrivateKeyIndex	39,000
R_TSIP_GenerateRsa2048PublicKeyIndex	140,000
R_TSIP_GenerateRsa2048PrivateKeyIndex	140,000
R_TSIP_GenerateRsa1024RandomKeyIndex*1	43,000,000
R_TSIP_GenerateRsa2048RandomKeyIndex*1	370,000,000
R_TSIP_UpdateRsa1024PublicKeyIndex	37,000
R_TSIP_UpdateRsa1024PrivateKeyIndex	38,000
R_TSIP_UpdateRsa2048PublicKeyIndex	140,000
R_TSIP_UpdateRsa2048PrivateKeyIndex	140,000

Note: 1. Average value over 10 runs.

Table 1-44 Performance of RSASSA-PKCS1-v1_5 Signature Generation/Verification (HASH = SHA1)

API	Performance (Unit: Cycle)		
	Message Size = 1 Byte	Message Size = 128 Bytes	Message Size = 256 Bytes
R_TSIP_RsassaPkcs1024SignatureGenerate	1,300,000	1,300,000	1,300,000
R_TSIP_RsassaPkcs1024SignatureVerification	17,000	19,000	19,000
R_TSIP_RsassaPkcs2048SignatureGenerate	26,000,000	26,000,000	26,000,000
R_TSIP_RsassaPkcs2048SignatureVerification	140,000	140,000	140,000

Table 1-45 Performance of RSASSA-PKCS1-v1_5 Signature Generation/Verification (HASH = SHA256)

API	Performance (Unit: Cycle)		
	Message Size = 1 Byte	Message Size = 128 Bytes	Message Size = 256 Bytes
R_TSIP_RsassaPkcs1024SignatureGenerate	1,300,000	1,300,000	1,300,000
R_TSIP_RsassaPkcs1024SignatureVerification	17,000	19,000	19,000
R_TSIP_RsassaPkcs2048SignatureGenerate	26,000,000	26,000,000	26,000,000
R_TSIP_RsassaPkcs2048SignatureVerification	140,000	140,000	140,000

Table 1-46 Performance of RSASSA-PKCS1-v1_5 Signature Generation/Verification (HASH = MD5)

API	Performance (Unit: Cycle)		
	Message Size = 1 Byte	Message Size = 128 Bytes	Message Size = 256 Bytes
R_TSIP_RsassaPkcs1024SignatureGenerate	1,300,000	1,300,000	1,300,000
R_TSIP_RsassaPkcs1024SignatureVerification	17,000	19,000	19,000
R_TSIP_RsassaPkcs2048SignatureGenerate	26,000,000	26,000,000	26,000,000
R_TSIP_RsassaPkcs2048SignatureVerification	140,000	140,000	140,000

Table 1-47 Performance of RSAES-PKCS1-v1_5 Encryption/Decryption with 1024-Bit Key Size

API	Performance (Unit: Cycle)	
	Message Size = 1 Byte	Message Size = 117 Bytes
R_TSIP_RsaesPkcs1024Encrypt	22,000	17,000
R_TSIP_RsaesPkcs1024Decrypt	1,300,000	1,300,000

Table 1-48 Performance of RSAES-PKCS1-v1_5 Encryption/Decryption with 2048-Bit Key Size

API	Performance (Unit: Cycle)	
	Message Size = 1 Byte	Message Size = 245 Bytes
R_TSIP_RsaesPkcs2048Encrypt	150,000	140,000
R_TSIP_RsaesPkcs2048Decrypt	26,000,000	26,000,000

Table 1-49 Performance of RSAES-OAEP Encryption/Decryption with 1024-Bit Key Size

API	Performance (Unit: Cycle)	
	Message Size = 1 Byte	Message Size = 62 Bytes
R_TSIP_RsaesOaep1024Encrypt	26,000	26,000
R_TSIP_RsaesOaep1024Decrypt	1,300,000	1,300,000

Table 1-50 Performance of RSAES-OAEP Encryption/Decryption with 2048-Bit Key Size

API	Performance (Unit: Cycle)	
	Message Size = 1 Byte	Message Size = 190 Bytes
R_TSIP_RsaesOaep2048Encrypt	150,000	150,000
R_TSIP_RsaesOaep2048Decrypt	26,000,000	26,000,000

Table 1-51 Performance of HASH (SHA1)

API	Performance (Unit: Cycle)		
	128-Byte Processing	192-Byte Processing	256-Byte Processing
R_TSIP_Sha1Init	130	130	130
R_TSIP_Sha1Update	1,600	1,800	2,000
R_TSIP_Sha1Final	830	830	830

Table 1-52 Performance of HASH (SHA256)

API	Performance (Unit: Cycle)		
	128-Byte Processing	192-Byte Processing	256-Byte Processing
R_TSIP_Sha256Init	140	140	140
R_TSIP_Sha256Update	1,600	1,800	2,000
R_TSIP_Sha256Final	840	840	840

Table 1-53 Performance of HASH (MD5)

API	Performance (Unit: Cycle)		
	128-Byte Processing	192-Byte Processing	256-Byte Processing
R_TSIP_Md5Init	120	120	120
R_TSIP_Md5Update	1,400	1,600	1,800
R_TSIP_Md5Final	780	780	780

Table 1-54 Performance of Common APIs (HMAC Wrapped Key Generation)

API	Performance (Unit: Cycle)
R_TSIP_GenerateSha1HmacKeyIndex	2,800
R_TSIP_GenerateSha256HmacKeyIndex	2,800
R_TSIP_UpdateSha1HmacKeyIndex	2,500
R_TSIP_UpdateSha256HmacKeyIndex	2,600

Table 1-55 Performance of HMAC (SHA1)

API	Performance (Unit: Cycle)		
	128-Byte Processing	192-Byte Processing	256-Byte Processing
R_TSIP_Sha1HmacGenerateInit	1,400	1,400	1,400
R_TSIP_Sha1HmacGenerateUpdate	970	1,200	1,500
R_TSIP_Sha1HmacGenerateFinal	2,000	2,000	2,000
R_TSIP_Sha1HmacVerifyInit	1,400	1,400	1,400
R_TSIP_Sha1HmacVerifyUpdate	970	1,300	1,500
R_TSIP_Sha1HmacVerifyFinal	3,000	3,000	3,000

Table 1-56 Performance of HMAC (SHA256)

API	Performance (Unit: Cycle)		
	128-Byte Processing	192-Byte Processing	256-Byte Processing
R_TSIP_Sha256HmacGenerateInit	1,800	1,800	1,800
R_TSIP_Sha256HmacGenerateUpdate	910	1,100	1,400
R_TSIP_Sha256HmacGenerateFinal	1,900	1,900	1,900
R_TSIP_Sha256HmacVerifyInit	1,800	1,800	1,800
R_TSIP_Sha256HmacVerifyUpdate	910	1,100	1,300
R_TSIP_Sha256HmacVerifyFinal	2,900	2,900	2,900

Table 1-57 Performance of Common APIs (ECC Wrapped Key Generation)

API	Performance (Unit: Cycle)
R_TSIP_GenerateEccP192PublicKeyIndex	3,200
R_TSIP_GenerateEccP224PublicKeyIndex	3,200
R_TSIP_GenerateEccP256PublicKeyIndex	3,200
R_TSIP_GenerateEccP384PublicKeyIndex	3,300
R_TSIP_GenerateEccP192PrivateKeyIndex	2,800
R_TSIP_GenerateEccP224PrivateKeyIndex	2,800
R_TSIP_GenerateEccP256PrivateKeyIndex	2,800
R_TSIP_GenerateEccP384PrivateKeyIndex	2,800
R_TSIP_GenerateEccP192RandomKeyIndex*1	140,000
R_TSIP_GenerateEccP224RandomKeyIndex*1	150,000
R_TSIP_GenerateEccP256RandomKeyIndex*1	150,000
R_TSIP_GenerateEccP384RandomKeyIndex*1	1,100,000
R_TSIP_UpdateEccP192PublicKeyIndex	2,800
R_TSIP_UpdateEccP224PublicKeyIndex	2,800
R_TSIP_UpdateEccP256PublicKeyIndex	2,900
R_TSIP_UpdateEccP384PublicKeyIndex	3,100
R_TSIP_UpdateEccP192PrivateKeyIndex	2,600
R_TSIP_UpdateEccP224PrivateKeyIndex	2,600
R_TSIP_UpdateEccP256PrivateKeyIndex	2,600
R_TSIP_UpdateEccP384PrivateKeyIndex	2,500

Note: 1. Average value over 10 runs.

Table 1-58 Performance of ECDSA Signature Generation/Verification

API	Performance (Unit: Cycle)		
	Message Size = 1 Byte	Message Size = 128 Bytes	Message Size = 256 Bytes
R_TSIP_EcdsaP192SignatureGenerate	180,000	170,000	180,000
R_TSIP_EcdsaP224SignatureGenerate	180,000	180,000	180,000
R_TSIP_EcdsaP256SignatureGenerate	180,000	180,000	180,000
R_TSIP_EcdsaP384SignatureGenerate*1	1,200,000		
R_TSIP_EcdsaP192SignatureVerification	320,000	330,000	330,000
R_TSIP_EcdsaP224SignatureVerification	350,000	350,000	350,000
R_TSIP_EcdsaP256SignatureVerification	350,000	350,000	360,000
R_TSIP_EcdsaP384SignatureVerification*1	2,200,000		

Note: 1. Does not include SHA384 calculation.

Table 1-59 Key Exchange Performance

API	Performance (Unit: Cycle)
R_TSIP_EcdhP256Init	56
R_TSIP_EcdhP256ReadPublicKey	360,000
R_TSIP_EcdhP256MakePublicKey	330,000
R_TSIP_EcdhP256CalculateSharedSecretIndex	380,000
R_TSIP_EcdhP256KeyDerivation	3,700
R_TSIP_EcdhP256SshKeyDerivation	51,000
R_TSIP_EcdheP512KeyAgreement	3,300,000
R_TSIP_Rsa2048DhKeyAgreement	53,000,000

Note: Key exchange performance (excluding KeyAgreement) was measured with parameters fixed as follows: key exchange format = ECDHE and derived key type = AES-128.

1.7.6 RX671

Table 1-60 Performance of Common APIs

API	Performance (Unit: Cycle)
R_TSIP_Open	5,500,000
R_TSIP_Close	320
R_TSIP_GetVersion	24
R_TSIP_GenerateAes128KeyIndex	2,000
R_TSIP_GenerateAes256KeyIndex	2,200
R_TSIP_GenerateAes128RandomKeyIndex	1,200
R_TSIP_GenerateAes256RandomKeyIndex	1,600
R_TSIP_GenerateRandomNumber	520
R_TSIP_GenerateUpdateKeyRingKeyIndex	2,200
R_TSIP_UpdateAes128KeyIndex	1,800
R_TSIP_UpdateAes256KeyIndex	2,000

Table 1-61 Firmware Verification Performance

API	Performance (Unit: Cycle)		
	8 KB Processing	16 KB Processing	24 KB Processing
R_TSIP_VerifyFirmwareMACInit	160	160	160
R_TSIP_VerifyFirmwareMACUpdate	18,000	35,000	53,000
R_TSIP_VerifyFirmwareMACFinal	390	390	390

Table 1-62 Performance of AES

API	Performance (Unit: Cycle)		
	16-Byte Processing	48-Byte Processing	80-Byte Processing
R_TSIP_Aes128EcbEncryptInit	1,200	1,200	1,200
R_TSIP_Aes128EcbEncryptUpdate	380	490	620
R_TSIP_Aes128EcbEncryptFinal	330	320	320
R_TSIP_Aes128EcbDecryptInit	1,200	1,200	1,200
R_TSIP_Aes128EcbDecryptUpdate	450	560	690
R_TSIP_Aes128EcbDecryptFinal	340	330	340
R_TSIP_Aes256EcbEncryptInit	1,300	1,300	1,300
R_TSIP_Aes256EcbEncryptUpdate	410	510	650
R_TSIP_Aes256EcbEncryptFinal	320	310	320
R_TSIP_Aes256EcbDecryptInit	1,300	1,300	1,300
R_TSIP_Aes256EcbDecryptUpdate	470	580	710
R_TSIP_Aes256EcbDecryptFinal	330	330	330
R_TSIP_Aes128CbcEncryptInit	1,200	1,200	1,200
R_TSIP_Aes128CbcEncryptUpdate	440	550	680
R_TSIP_Aes128CbcEncryptFinal	340	340	340
R_TSIP_Aes128CbcDecryptInit	1,200	1,200	1,200
R_TSIP_Aes128CbcDecryptUpdate	490	600	730
R_TSIP_Aes128CbcDecryptFinal	360	350	350
R_TSIP_Aes256CbcEncryptInit	1,300	1,300	1,300
R_TSIP_Aes256CbcEncryptUpdate	450	560	700
R_TSIP_Aes256CbcEncryptFinal	340	340	340
R_TSIP_Aes256CbcDecryptInit	1,300	1,300	1,300
R_TSIP_Aes256CbcDecryptUpdate	520	640	770
R_TSIP_Aes256CbcDecryptFinal	350	350	350

Table 1-63 Performance of AES-GCM

API	Performance (Unit: Cycle)		
	48-Byte Processing	64-Byte Processing	80-Byte Processing
R_TSIP_Aes128GcmEncryptInit	4,100	4,100	4,100
R_TSIP_Aes128GcmEncryptUpdate	1,600	1,700	1,800
R_TSIP_Aes128GcmEncryptFinal	1,000	1,000	1,000
R_TSIP_Aes128GcmDecryptInit	4,100	4,000	4,000
R_TSIP_Aes128GcmDecryptUpdate	1,600	1,600	1,700
R_TSIP_Aes128GcmDecryptFinal	1,500	1,500	1,500
R_TSIP_Aes256GcmEncryptInit	4,100	4,100	4,100
R_TSIP_Aes256GcmEncryptUpdate	1,600	1,600	1,700
R_TSIP_Aes256GcmEncryptFinal	780	780	760
R_TSIP_Aes256GcmDecryptInit	4,100	4,100	4,100
R_TSIP_Aes256GcmDecryptUpdate	1,800	1,600	1,700
R_TSIP_Aes256GcmDecryptFinal	1,200	1,200	1,200

Note: GCM performance was measured with parameters fixed as follows: ivec = 1,024 bits, AAD = 720 bits, and authentication tag = 128 bits.

Table 1-64 Performance of AES-CCM

API	Performance (Unit: Cycle)		
	48-Byte Processing	64-Byte Processing	80-Byte Processing
R_TSIP_Aes128CcmEncryptInit	2,300	2,300	2,300
R_TSIP_Aes128CcmEncryptUpdate	880	960	1,000
R_TSIP_Aes128CcmEncryptFinal	700	690	690
R_TSIP_Aes128CcmDecryptInit	2,400	2,300	2,300
R_TSIP_Aes128CcmDecryptUpdate	800	870	950
R_TSIP_Aes128CcmDecryptFinal	1,100	1,100	1,100
R_TSIP_Aes256CcmEncryptInit	1,900	1,900	1,900
R_TSIP_Aes256CcmEncryptUpdate	940	1,100	1,200
R_TSIP_Aes256CcmEncryptFinal	740	740	740
R_TSIP_Aes256CcmDecryptInit	1,900	1,900	1,900
R_TSIP_Aes256CcmDecryptUpdate	840	930	1,000
R_TSIP_Aes256CcmDecryptFinal	1,300	1,200	1,200

Note: CCM performance was measured with parameters fixed as follows: nonce = 104 bits, AAD = 880 bits, and MAC = 128 bits.

Table 1-65 Performance of AES-CMAC

API	Performance (Unit: Cycle)		
	48-Byte Processing	64-Byte Processing	80-Byte Processing
R_TSIP_Aes128CmacGenerateInit	850	850	850
R_TSIP_Aes128CmacGenerateUpdate	490	520	560
R_TSIP_Aes128CmacGenerateFinal	610	600	600
R_TSIP_Aes128CmacVerifyInit	850	850	850
R_TSIP_Aes128CmacVerifyUpdate	490	510	550
R_TSIP_Aes128CmacVerifyFinal	1,100	1,100	1,100
R_TSIP_Aes256CmacGenerateInit	980	970	970
R_TSIP_Aes256CmacGenerateUpdate	510	550	600
R_TSIP_Aes256CmacGenerateFinal	630	620	620
R_TSIP_Aes256CmacVerifyInit	960	960	960
R_TSIP_Aes256CmacVerifyUpdate	510	540	590
R_TSIP_Aes256CmacVerifyFinal	1,100	1,100	1,100

Table 1-66 Performance of AES Key Wrap

API	Performance (Unit: Cycle)	
	Wrap Target Key AES-128	Wrap Target Key AES-256
R_TSIP_Aes128KeyWrap	5,500	8,600
R_TSIP_Aes256KeyWrap	5,700	8,900
R_TSIP_Aes128KeyUnwrap	6,200	9,300
R_TSIP_Aes256KeyUnwrap	6,400	9,600

Table 1-67 Performance of Common APIs (TDES Wrapped Key Generation)

API	Performance (Unit: Cycle)
R_TSIP_GenerateTdesKeyIndex	2,200
R_TSIP_GenerateTdesRandomKeyIndex	1,700
R_TSIP_UpdateTdesKeyIndex	2,000

Table 1-68 Performance of TDES

API	Performance (Unit: Cycle)		
	16-Byte Processing	48-Byte Processing	80-Byte Processing
R_TSIP_TdesEcbEncryptInit	760	750	750
R_TSIP_TdesEcbEncryptUpdate	420	610	810
R_TSIP_TdesEcbEncryptFinal	320	310	310
R_TSIP_TdesEcbDecryptInit	770	770	770
R_TSIP_TdesEcbDecryptUpdate	440	640	830
R_TSIP_TdesEcbDecryptFinal	320	320	320
R_TSIP_TdesCbcEncryptInit	810	810	810
R_TSIP_TdesCbcEncryptUpdate	470	670	870
R_TSIP_TdesCbcEncryptFinal	330	340	340
R_TSIP_TdesCbcDecryptInit	820	820	820
R_TSIP_TdesCbcDecryptUpdate	490	690	890
R_TSIP_TdesCbcDecryptFinal	350	350	350

Table 1-69 Performance of Common APIs (ARC4 Wrapped Key Generation)

API	Performance (Unit: Cycle)
R_TSIP_GenerateArc4KeyIndex	3,900
R_TSIP_GenerateArc4RandomKeyIndex	8,500
R_TSIP_UpdateArc4KeyIndex	3,700

Table 1-70 Performance of ARC4

API	Performance (Unit: Cycle)		
	16-Byte Processing	48-Byte Processing	80-Byte Processing
R_TSIP_Arc4EncryptInit	1,700	1,700	1,700
R_TSIP_Arc4EncryptUpdate	360	470	600
R_TSIP_Arc4EncryptFinal	240	230	230
R_TSIP_Arc4DecryptInit	1,700	1,700	1,700
R_TSIP_Arc4DecryptUpdate	350	470	600
R_TSIP_Arc4DecryptFinal	240	240	240

Table 1-71 Performance of Common APIs (RSA Wrapped Key Generation)

API	Performance (Unit: Cycle)
R_TSIP_GenerateRsa1024PublicKeyIndex	37,000
R_TSIP_GenerateRsa1024PrivateKeyIndex	38,000
R_TSIP_GenerateRsa2048PublicKeyIndex	140,000
R_TSIP_GenerateRsa2048PrivateKeyIndex	140,000
R_TSIP_GenerateRsa1024RandomKeyIndex*1	73,000,000
R_TSIP_GenerateRsa2048RandomKeyIndex*1	400,000,000
R_TSIP_UpdateRsa1024PublicKeyIndex	36,000
R_TSIP_UpdateRsa1024PrivateKeyIndex	37,000
R_TSIP_UpdateRsa2048PublicKeyIndex	140,000
R_TSIP_UpdateRsa2048PrivateKeyIndex	140,000

Note: 1. Average value over 10 runs.

Table 1-72 Performance of RSASSA-PKCS1-v1_5 Signature Generation/Verification (HASH = SHA1)

API	Performance (Unit: Cycle)		
	Message Size = 1 Byte	Message Size = 128 Bytes	Message Size = 256 Bytes
R_TSIP_RsassaPkcs1024SignatureGenerate	1,200,000	1,200,000	1,200,000
R_TSIP_RsassaPkcs1024SignatureVerification	16,000	17,000	18,000
R_TSIP_RsassaPkcs2048SignatureGenerate	26,000,000	26,000,000	26,000,000
R_TSIP_RsassaPkcs2048SignatureVerification	140,000	140,000	140,000

Table 1-73 Performance of RSASSA-PKCS1-v1_5 Signature Generation/Verification (HASH = SHA256)

API	Performance (Unit: Cycle)		
	Message Size = 1 Byte	Message Size = 128 Bytes	Message Size = 256 Bytes
R_TSIP_RsassaPkcs1024SignatureGenerate	1,300,000	1,300,000	1,300,000
R_TSIP_RsassaPkcs1024SignatureVerification	16,000	17,000	18,000
R_TSIP_RsassaPkcs2048SignatureGenerate	26,000,000	26,000,000	26,000,000
R_TSIP_RsassaPkcs2048SignatureVerification	130,000	130,000	140,000

Table 1-74 Performance of RSASSA-PKCS1-v1_5 Signature Generation/Verification (HASH = MD5)

API	Performance (Unit: Cycle)		
	Message Size = 1 Byte	Message Size = 128 Bytes	Message Size = 256 Bytes
R_TSIP_RsassaPkcs1024SignatureGenerate	1,300,000	1,200,000	1,200,000
R_TSIP_RsassaPkcs1024SignatureVerification	16,000	17,000	17,000
R_TSIP_RsassaPkcs2048SignatureGenerate	26,000,000	27,000,000	27,000,000
R_TSIP_RsassaPkcs2048SignatureVerification	140,000	140,000	140,000

Table 1-75 Performance of RSAES-PKCS1-v1_5 Encryption/Decryption with 1024-Bit Key Size

API	Performance (Unit: Cycle)	
	Message Size = 1 Byte	Message Size = 117 Bytes
R_TSIP_RsaesPkcs1024Encrypt	20,000	15,000
R_TSIP_RsaesPkcs1024Decrypt	1,300,000	1,300,000

Table 1-76 Performance of RSAES-PKCS1-v1_5 Encryption/Decryption with 2048-Bit Key Size

API	Performance (Unit: Cycle)	
	Message Size = 1 Byte	Message Size = 245 Bytes
R_TSIP_RsaesPkcs2048Encrypt	150,000	150,000
R_TSIP_RsaesPkcs2048Decrypt	26,000,000	27,000,000

Table 1-77 Performance of RSAES-OAEP Encryption/Decryption with 1024-Bit Key Size

API	Performance (Unit: Cycle)	
	Message Size = 1 Byte	Message Size = 62 Bytes
R_TSIP_RsaesOaep1024Encrypt	23,000	23,000
R_TSIP_RsaesOaep1024Decrypt	1,300,000	1,300,000

Table 1-78 Performance of RSAES-OAEP Encryption/Decryption with 2048-Bit Key Size

API	Performance (Unit: Cycle)	
	Message Size = 1 Byte	Message Size = 190 Bytes
R_TSIP_RsaesPkcs2048Encrypt	140,000	150,000
R_TSIP_RsaesPkcs2048Decrypt	27,000,000	27,000,000

Table 1-79 Performance of HASH (SHA1)

API	Performance (Unit: Cycle)		
	128-Byte Processing	192-Byte Processing	256-Byte Processing
R_TSIP_Sha1Init	100	100	100
R_TSIP_Sha1Update	1,200	1,400	1,600
R_TSIP_Sha1Final	660	660	660

Table 1-80 Performance of HASH (SHA256)

API	Performance (Unit: Cycle)		
	128-Byte Processing	192-Byte Processing	256-Byte Processing
R_TSIP_Sha256Init	110	110	110
R_TSIP_Sha256Update	1,200	1,400	1,600
R_TSIP_Sha256Final	670	670	670

Table 1-81 Performance of HASH (MD5)

API	Performance (Unit: Cycle)		
	128-Byte Processing	192-Byte Processing	256-Byte Processing
R_TSIP_Md5Init	98	96	96
R_TSIP_Md5Update	1,200	1,300	1,500
R_TSIP_Md5Final	620	620	620

Table 1-82 Performance of Common APIs (HMAC Wrapped Key Generation)

API	Performance (Unit: Cycle)
R_TSIP_GenerateSha1HmacKeyIndex	2,300
R_TSIP_GenerateSha256HmacKeyIndex	2,300
R_TSIP_UpdateSha1HmacKeyIndex	2,100
R_TSIP_UpdateSha256HmacKeyIndex	2,100

Table 1-83 Performance of HMAC (SHA1)

API	Performance (Unit: Cycle)		
	128-Byte Processing	192-Byte Processing	256-Byte Processing
R_TSIP_Sha1HmacGenerateInit	1,000	1,000	1,000
R_TSIP_Sha1HmacGenerateUpdate	800	1,000	1,300
R_TSIP_Sha1HmacGenerateFinal	1,600	1,600	1,600
R_TSIP_Sha1HmacVerifyInit	1,000	1,000	1,000
R_TSIP_Sha1HmacVerifyUpdate	800	1,100	1,300
R_TSIP_Sha1HmacVerifyFinal	2,300	2,300	2,300

Table 1-84 Performance of HMAC (SHA256)

API	Performance (Unit: Cycle)		
	128-Byte Processing	192-Byte Processing	256-Byte Processing
R_TSIP_Sha256HmacGenerateInit	1,400	1,300	1,300
R_TSIP_Sha256HmacGenerateUpdate	740	910	1,100
R_TSIP_Sha256HmacGenerateFinal	1,600	1,600	1,60
R_TSIP_Sha256HmacVerifyInit	1,300	1,300	1,300
R_TSIP_Sha256HmacVerifyUpdate	740	910	1,100
R_TSIP_Sha256HmacVerifyFinal	2,300	2,200	2,200

Table 1-85 Performance of Common APIs (ECC Wrapped Key Generation)

API	Performance (Unit: Cycle)
R_TSIP_GenerateEccP192PublicKeyIndex	2,600
R_TSIP_GenerateEccP224PublicKeyIndex	2,600
R_TSIP_GenerateEccP256PublicKeyIndex	2,600
R_TSIP_GenerateEccP384PublicKeyIndex	2,700
R_TSIP_GenerateEccP192PrivateKeyIndex	2,300
R_TSIP_GenerateEccP224PrivateKeyIndex	2,300
R_TSIP_GenerateEccP256PrivateKeyIndex	2,300
R_TSIP_GenerateEccP384PrivateKeyIndex	2,300
R_TSIP_GenerateEccP192RandomKeyIndex*1	140,000
R_TSIP_GenerateEccP224RandomKeyIndex*1	140,000
R_TSIP_GenerateEccP256RandomKeyIndex*1	150,000
R_TSIP_GenerateEccP384RandomKeyIndex*1	1,100,000
R_TSIP_UpdateEccP192PublicKeyIndex	2,400
R_TSIP_UpdateEccP224PublicKeyIndex	2,300
R_TSIP_UpdateEccP256PublicKeyIndex	2,400
R_TSIP_UpdateEccP384PublicKeyIndex	2,500
R_TSIP_UpdateEccP192PrivateKeyIndex	2,100
sR_TSIP_UpdateEccP224PrivateKeyIndex	2,100
R_TSIP_UpdateEccP256PrivateKeyIndex	2,100
R_TSIP_UpdateEccP384PrivateKeyIndex	2,100

Note: 1. Average value over 10 runs.

Table 1-86 Performance of ECDSA Signature Generation/Verification

API	Performance (Unit: Cycle)		
	Message Size = 1 Byte	Message Size = 128 Bytes	Message Size = 256 Bytes
R_TSIP_EcdsaP192SignatureGenerate	170,000	160,000	160,000
R_TSIP_EcdsaP224SignatureGenerate	160,000	170,000	160,000
R_TSIP_EcdsaP256SignatureGenerate	170,000	170,000	170,000
R_TSIP_EcdsaP384SignatureGenerate*1	1,200,000		
R_TSIP_EcdsaP192SignatureVerification	300,000	310,000	310,000
R_TSIP_EcdsaP224SignatureVerification	330,000	330,000	330,000
R_TSIP_EcdsaP256SignatureVerification	340,000	330,000	340,000
R_TSIP_EcdsaP384SignatureVerification*1	2,100,000		

Note: 1. Does not include SHA384 calculation.

Table 1-87 Key Exchange Performance

API	Performance (Unit: Cycle)
R_TSIP_EcdhP256Init	40
R_TSIP_EcdhP256ReadPublicKey	330,000
R_TSIP_EcdhP256MakePublicKey	320,000
R_TSIP_EcdhP256CalculateSharedSecretIndex	350,000
R_TSIP_EcdhP256KeyDerivation	3,000
R_TSIP_EcdhP256SshKeyDerivation	40,000
R_TSIP_EcdheP512KeyAgreement	3,200,000
R_TSIP_Rsa2048DhKeyAgreement	53,000,000

Note: Key exchange performance (excluding KeyAgreement) was measured with parameters fixed as follows: key exchange format = ECDHE and derived key type = AES-128.

1.7.7 RX72M, RX72N

Table 1-88 Performance of Common APIs

API	Performance (Unit: Cycle)
R_TSIP_Open	6,300,000
R_TSIP_Close	300
R_TSIP_GetVersion	22
R_TSIP_GenerateAes128KeyIndex	2,100
R_TSIP_GenerateAes256KeyIndex	2,300
R_TSIP_GenerateAes128RandomKeyIndex	1,200
R_TSIP_GenerateAes256RandomKeyIndex	1,800
R_TSIP_GenerateRandomNumber	560
R_TSIP_GenerateUpdateKeyRingKeyIndex	2,300
R_TSIP_UpdateAes128KeyIndex	1,900
R_TSIP_UpdateAes256KeyIndex	2,000

Table 1-89 Firmware Verification Performance

API	Performance (Unit: Cycle)		
	8 KB Processing	16 KB Processing	24 KB Processing
R_TSIP_VerifyFirmwareMACInit	170	170	170
R_TSIP_VerifyFirmwareMACUpdate	19,000	37,000	56,000
R_TSIP_VerifyFirmwareMACFinal	400	400	400

Table 1-90 Performance of AES

API	Performance (Unit: Cycle)		
	16-Byte Processing	48-Byte Processing	80-Byte Processing
R_TSIP_Aes128EcbEncryptInit	1,200	1,200	1,200
R_TSIP_Aes128EcbEncryptUpdate	390	510	640
R_TSIP_Aes128EcbEncryptFinal	330	330	330
R_TSIP_Aes128EcbDecryptInit	1,300	1,300	1,300
R_TSIP_Aes128EcbDecryptUpdate	460	560	700
R_TSIP_Aes128EcbDecryptFinal	340	340	340
R_TSIP_Aes256EcbEncryptInit	1,400	1,400	1,400
R_TSIP_Aes256EcbEncryptUpdate	400	530	660
R_TSIP_Aes256EcbEncryptFinal	340	330	330
R_TSIP_Aes256EcbDecryptInit	1,300	1,400	1,400
R_TSIP_Aes256EcbDecryptUpdate	470	600	730
R_TSIP_Aes256EcbDecryptFinal	350	350	350
R_TSIP_Aes128CbcEncryptInit	1,300	1,300	1,300
R_TSIP_Aes128CbcEncryptUpdate	440	550	700
R_TSIP_Aes128CbcEncryptFinal	350	350	350
R_TSIP_Aes128CbcDecryptInit	1,300	1,300	1,300
R_TSIP_Aes128CbcDecryptUpdate	500	610	750
R_TSIP_Aes128CbcDecryptFinal	360	360	360
R_TSIP_Aes256CbcEncryptInit	1,400	1,400	1,400
R_TSIP_Aes256CbcEncryptUpdate	460	580	720
R_TSIP_Aes256CbcEncryptFinal	360	360	360
R_TSIP_Aes256CbcDecryptInit	1,400	1,500	1,400
R_TSIP_Aes256CbcDecryptUpdate	530	660	790
R_TSIP_Aes256CbcDecryptFinal	380	380	380

Table 1-91 Performance of AES-GCM

API	Performance (Unit: Cycle)		
	48-Byte Processing	64-Byte Processing	80-Byte Processing
R_TSIP_Aes128GcmEncryptInit	4,300	4,300	4,300
R_TSIP_Aes128GcmEncryptUpdate	1,600	1,700	1,800
R_TSIP_Aes128GcmEncryptFinal	1,100	1,100	1,100
R_TSIP_Aes128GcmDecryptInit	4,200	4,200	4,200
R_TSIP_Aes128GcmDecryptUpdate	1,600	1,700	1,800
R_TSIP_Aes128GcmDecryptFinal	1,500	1,500	1,500
R_TSIP_Aes256GcmEncryptInit	4,300	4,300	4,300
R_TSIP_Aes256GcmEncryptUpdate	1,700	1,700	1,800
R_TSIP_Aes256GcmEncryptFinal	820	810	810
R_TSIP_Aes256GcmDecryptInit	4,300	4,300	4,300
R_TSIP_Aes256GcmDecryptUpdate	1,600	1,700	1,700
R_TSIP_Aes256GcmDecryptFinal	1,300	1,300	1,300

Note: GCM performance was measured with parameters fixed as follows: ivec = 1,024 bits, AAD = 720 bits, and authentication tag = 128 bits.

Table 1-92 Performance of AES-CCM

API	Performance (Unit: Cycle)		
	48-Byte Processing	64-Byte Processing	80-Byte Processing
R_TSIP_Aes128CcmEncryptInit	2,400	2,400	2,400
R_TSIP_Aes128CcmEncryptUpdate	910	980	1,000
R_TSIP_Aes128CcmEncryptFinal	730	720	720
R_TSIP_Aes128CcmDecryptInit	2,500	2,500	2,500
R_TSIP_Aes128CcmDecryptUpdate	820	900	970
R_TSIP_Aes128CcmDecryptFinal	1,200	1,200	1,200
R_TSIP_Aes256CcmEncryptInit	2,000	2,000	2,000
R_TSIP_Aes256CcmEncryptUpdate	960	1,000	1,100
R_TSIP_Aes256CcmEncryptFinal	770	770	770
R_TSIP_Aes256CcmDecryptInit	2,000	2,000	2,000
R_TSIP_Aes256CcmDecryptUpdate	860	960	1,000
R_TSIP_Aes256CcmDecryptFinal	1,200	1,200	1,200

Note: CCM performance was measured with parameters fixed as follows: nonce = 104 bits, AAD = 880 bits, and MAC = 128 bits.

Table 1-93 Performance of AES-CMAC

API	Performance (Unit: Cycle)		
	48-Byte Processing	64-Byte Processing	80-Byte Processing
R_TSIP_Aes128CmacGenerateInit	890	890	890
R_TSIP_Aes128CmacGenerateUpdate	490	530	560
R_TSIP_Aes128CmacGenerateFinal	630	630	630
R_TSIP_Aes128CmacVerifyInit	890	890	890
R_TSIP_Aes128CmacVerifyUpdate	490	530	570
R_TSIP_Aes128CmacVerifyFinal	1,100	1,000	1,100
R_TSIP_Aes256CmacGenerateInit	1,100	1,100	1,100
R_TSIP_Aes256CmacGenerateUpdate	520	570	620
R_TSIP_Aes256CmacGenerateFinal	650	650	650
R_TSIP_Aes256CmacVerifyInit	1,100	1,000	1,000
R_TSIP_Aes256CmacVerifyUpdate	520	570	610
R_TSIP_Aes256CmacVerifyFinal	1,100	1,100	1,100

Table 1-94 Performance of AES Key Wrap

API	Performance (Unit: Cycle)	
	Wrap Target Key AES-128	Wrap Target Key AES-256
R_TSIP_Aes128KeyWrap	5,700	8,900
R_TSIP_Aes256KeyWrap	5,800	9,300
R_TSIP_Aes128KeyUnwrap	6,400	9,600
R_TSIP_Aes256KeyUnwrap	6,600	10,000

Table 1-95 Performance of Common APIs (TDES Wrapped Key Generation)

API	Performance (Unit: Cycle)
R_TSIP_GenerateTdesKeyIndex	2,300
R_TSIP_GenerateTdesRandomKeyIndex	1,800
R_TSIP_UpdateTdesKeyIndex	2,000

Table 1-96 Performance of TDES

API	Performance (Unit: Cycle)		
	16-Byte Processing	48-Byte Processing	80-Byte Processing
R_TSIP_TdesEcbEncryptInit	790	790	790
R_TSIP_TdesEcbEncryptUpdate	440	630	830
R_TSIP_TdesEcbEncryptFinal	330	330	320
R_TSIP_TdesEcbDecryptInit	810	810	810
R_TSIP_TdesEcbDecryptUpdate	460	660	860
R_TSIP_TdesEcbDecryptFinal	340	340	340
R_TSIP_TdesCbcEncryptInit	850	850	850
R_TSIP_TdesCbcEncryptUpdate	490	690	890
R_TSIP_TdesCbcEncryptFinal	360	360	360
R_TSIP_TdesCbcDecryptInit	860	860	860
R_TSIP_TdesCbcDecryptUpdate	510	710	910
R_TSIP_TdesCbcDecryptFinal	370	370	370

Table 1-97 Performance of Common APIs (ARC4 Wrapped Key Generation)

API	Performance (Unit: Cycle)
R_TSIP_GenerateArc4KeyIndex	4,000
R_TSIP_GenerateArc4RandomKeyIndex	9,100
R_TSIP_UpdateArc4KeyIndex	3,800

Table 1-98 Performance of ARC4

API	Performance (Unit: Cycle)		
	16-Byte Processing	48-Byte Processing	80-Byte Processing
R_TSIP_Arc4EncryptInit	1,900	1,900	1,900
R_TSIP_Arc4EncryptUpdate	360	480	610
R_TSIP_Arc4EncryptFinal	240	240	240
R_TSIP_Arc4DecryptInit	1,900	1,900	1,900
R_TSIP_Arc4DecryptUpdate	360	480	620
R_TSIP_Arc4DecryptFinal	240	240	240

Table 1-99 Performance of Common APIs (RSA Wrapped Key Generation)

API	Performance (Unit: Cycle)
R_TSIP_GenerateRsa1024PublicKeyIndex	37,000
R_TSIP_GenerateRsa1024PrivateKeyIndex	38,000
R_TSIP_GenerateRsa2048PublicKeyIndex	140,000
R_TSIP_GenerateRsa2048PrivateKeyIndex	140,000
R_TSIP_GenerateRsa1024RandomKeyIndex*1	76,000,000
R_TSIP_GenerateRsa2048RandomKeyIndex*1	370,000,000
R_TSIP_UpdateRsa1024PublicKeyIndex	37,000
R_TSIP_UpdateRsa1024PrivateKeyIndex	38,000
R_TSIP_UpdateRsa2048PublicKeyIndex	140,000
R_TSIP_UpdateRsa2048PrivateKeyIndex	140,000

Note: 1. Average value over 10 runs.

Table 1-100 Performance of RSASSA-PKCS1-v1_5 Signature Generation/Verification (HASH = SHA1)

API	Performance (Unit: Cycle)		
	Message Size = 1 Byte	Message Size = 128 Bytes	Message Size = 256 Bytes
R_TSIP_RsassaPkcs1024SignatureGenerate	1,200,000	1,200,000	1,200,000
R_TSIP_RsassaPkcs1024SignatureVerification	17,000	18,000	18,000
R_TSIP_RsassaPkcs2048SignatureGenerate	26,000,000	26,000,000	26,000,000
R_TSIP_RsassaPkcs2048SignatureVerification	130,000	140,000	140,000

Table 1-101 Performance of RSASSA-PKCS1-v1_5 Signature Generation/Verification (HASH = SHA256)

API	Performance (Unit: Cycle)		
	Message Size = 1 Byte	Message Size = 128 Bytes	Message Size = 256 Bytes
R_TSIP_RsassaPkcs1024SignatureGenerate	1,200,000	1,200,000	1,200,000
R_TSIP_RsassaPkcs1024SignatureVerification	17,000	18,000	18,000
R_TSIP_RsassaPkcs2048SignatureGenerate	27,000,000	27,000,000	27,000,000
R_TSIP_RsassaPkcs2048SignatureVerification	130,000	140,000	140,000

Table 1-102 Performance of RSASSA-PKCS1-v1_5 Signature Generation/Verification (HASH = MD5)

API	Performance (Unit: Cycle)		
	Message Size = 1 Byte	Message Size = 128 Bytes	Message Size = 256 Bytes
R_TSIP_RsassaPkcs1024SignatureGenerate	1,200,000	1,200,000	1,200,000
R_TSIP_RsassaPkcs1024SignatureVerification	16,000	18,000	18,000
R_TSIP_RsassaPkcs2048SignatureGenerate	26,000,000	26,000,000	27,000,000
R_TSIP_RsassaPkcs2048SignatureVerification	130,000	140,000	140,000

Table 1-103 Performance of RSAES-PKCS1-v1_5 Encryption/Decryption with 1024-Bit Key Size

API	Performance (Unit: Cycle)	
	Message Size = 1 Byte	Message Size = 117 Bytes
R_TSIP_RsaesPkcs1024Encrypt	20,000	16,000
R_TSIP_RsaesPkcs1024Decrypt	1,200,000	1,200,000

Table 1-104 Performance of RSAES-PKCS1-v1_5 Encryption/Decryption with 2048-Bit Key Size

API	Performance (Unit: Cycle)	
	Message Size = 1 Byte	Message Size = 245 Bytes
R_TSIP_RsaesPkcs2048Encrypt	140,000	130,000
R_TSIP_RsaesPkcs2048Decrypt	26,000,000	26,000,000

Table 1-105 Performance of RSAES-OAEP Encryption/Decryption with 1024-Bit Key Size

API	Performance (Unit: Cycle)	
	Message Size = 1 Byte	Message Size = 117 Bytes
R_TSIP_RsaesOaep1024Encrypt	23,000	23,000
R_TSIP_RsaesOaep1024Decrypt	1,300,000	1,300,000

Table 1-106 Performance of RSAES-OAEP Encryption/Decryption with 2048-Bit Key Size

API	Performance (Unit: Cycle)	
	Message Size = 1 Byte	Message Size = 245 Bytes
R_TSIP_RsaesOaep2048Encrypt	150,000	150,000
R_TSIP_RsaesOaep2048Decrypt	27,000,000	27,000,000

Table 1-107 Performance of HASH (SHA1)

API	Performance (Unit: Cycle)		
	128-Byte Processing	192-Byte Processing	256-Byte Processing
R_TSIP_Sha1Init	110	100	110
R_TSIP_Sha1Update	1,200	1,500	1,700
R_TSIP_Sha1Final	670	670	680

Table 1-108 Performance of HASH (SHA256)

API	Performance (Unit: Cycle)		
	128-Byte Processing	192-Byte Processing	256-Byte Processing
R_TSIP_Sha256Init	110	110	110
R_TSIP_Sha256Update	1,300	1,400	1,600
R_TSIP_Sha256Final	680	680	680

Table 1-109 Performance of HASH (MD5)

API	Performance (Unit: Cycle)		
	128-Byte Processing	192-Byte Processing	256-Byte Processing
R_TSIP_Md5Init	94	96	98
R_TSIP_Md5Update	1,200	1,300	1,500
R_TSIP_Md5Final	630	630	630

Table 1-110 Performance of Common APIs (HMAC Wrapped Key Generation)

API	Performance (Unit: Cycle)
R_TSIP_GenerateSha1HmacKeyIndex	2,400
R_TSIP_GenerateSha256HmacKeyIndex	2,400
R_TSIP_UpdateSha1HmacKeyIndex	2,100
R_TSIP_UpdateSha256HmacKeyIndex	2,100

Table 1-111 Performance of HMAC (SHA1)

API	Performance (Unit: Cycle)		
	128-Byte Processing	192-Byte Processing	256-Byte Processing
R_TSIP_Sha1HmacGenerateInit	1,100	1,100	1,100
R_TSIP_Sha1HmacGenerateUpdate	810	1,000	1,200
R_TSIP_Sha1HmacGenerateFinal	1,600	1,600	1,600
R_TSIP_Sha1HmacVerifyInit	1,100	1,100	1,100
R_TSIP_Sha1HmacVerifyUpdate	800	1,000	1,200
R_TSIP_Sha1HmacVerifyFinal	2,300	2,300	2,300

Table 1-112 Performance of HMAC (SHA256)

API	Performance (Unit: Cycle)		
	128-Byte Processing	192-Byte Processing	256-Byte Processing
R_TSIP_Sha256HmacGenerateInit	1,500	1,500	1,500
R_TSIP_Sha256HmacGenerateUpdate	740	900	1,100
R_TSIP_Sha256HmacGenerateFinal	1,500	1,500	1,600
R_TSIP_Sha256HmacVerifyInit	1,400	1,400	1,400
R_TSIP_Sha256HmacVerifyUpdate	730	900	1,100
R_TSIP_Sha256HmacVerifyFinal	2,300	2,300	2,300

Table 1-113 Performance of Common APIs (ECC Wrapped Key Generation)

API	Performance (Unit: Cycle)
R_TSIP_GenerateEccP192PublicKeyIndex	2,700
R_TSIP_GenerateEccP224PublicKeyIndex	2,700
R_TSIP_GenerateEccP256PublicKeyIndex	2,700
R_TSIP_GenerateEccP384PublicKeyIndex	2,800
R_TSIP_GenerateEccP192PrivateKeyIndex	2,400
R_TSIP_GenerateEccP224PrivateKeyIndex	2,400
R_TSIP_GenerateEccP256PrivateKeyIndex	2,400
R_TSIP_GenerateEccP384PrivateKeyIndex	2,400
R_TSIP_GenerateEccP192RandomKeyIndex*1	140,000
R_TSIP_GenerateEccP224RandomKeyIndex*1	140,000
R_TSIP_GenerateEccP256RandomKeyIndex*1	140,000
R_TSIP_GenerateEccP384RandomKeyIndex*1	1,100,000
R_TSIP_UpdateEccP192PublicKeyIndex	2,500
R_TSIP_UpdateEccP224PublicKeyIndex	2,500
R_TSIP_UpdateEccP256PublicKeyIndex	2,500
R_TSIP_UpdateEccP384PublicKeyIndex	2,600
R_TSIP_UpdateEccP192PrivateKeyIndex	2,100
R_TSIP_UpdateEccP224PrivateKeyIndex	2,200
R_TSIP_UpdateEccP256PrivateKeyIndex	2,200
R_TSIP_UpdateEccP384PrivateKeyIndex	2,100

Note: 1. Average value over 10 runs.

Table 1-114 Performance of ECDSA Signature Generation/Verification

API	Performance (Unit: Cycle)		
	Message Size = 1 Byte	Message Size = 128 Bytes	Message Size = 256 Bytes
R_TSIP_EcdsaP192SignatureGenerate	170,000	160,000	170,000
R_TSIP_EcdsaP224SignatureGenerate	160,000	170,000	160,000
R_TSIP_EcdsaP256SignatureGenerate	170,000	170,000	160,000
R_TSIP_EcdsaP384SignatureGenerate*1	1,200,000		
R_TSIP_EcdsaP192SignatureVerification	310,000	310,000	310,000
R_TSIP_EcdsaP224SignatureVerification	330,000	330,000	330,000
R_TSIP_EcdsaP256SignatureVerification	330,000	330,000	330,000
R_TSIP_EcdsaP384SignatureVerification*1	2,100,000		

Note: 1. Does not include SHA384 calculation.

Table 1-115 Key Exchange Performance

API	Performance (Unit: Cycle)
R_TSIP_EcdhP256Init	40
R_TSIP_EcdhP256ReadPublicKey	330,000
R_TSIP_EcdhP256MakePublicKey	310,000
R_TSIP_EcdhP256CalculateSharedSecretIndex	350,000
R_TSIP_EcdhP256KeyDerivation	3,100
R_TSIP_EcdhP256SshKeyDerivation	41,000
R_TSIP_EcdheP512KeyAgreement	3,200,000
R_TSIP_Rsa2048DhKeyAgreement	52,000,000

Note: Key exchange performance (excluding KeyAgreement) was measured with parameters fixed as follows: key exchange format = ECDHE and derived key type = AES-128.

2. API Information

2.1 Hardware Requirements

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

2.2 Software Requirements

The TSIP drivers are dependent on the following module:

r_bsp Use rev. 7.30 or later. (BSP stands for “board support package.”)

[RX231 and RX23W (On the RX231, portions of the comment below following “= Chip” differ.)]

Change the value in the following macro in r_bsp_config.h in the r_config folder to 0xB or 0xD (RX23W only).

```
/* Chip version.
   Character(s) = Value for macro =
   A            = 0xA             = Chip version A
                                  = Security function not included.
   B            = 0xB             = Chip version B
                                  = Security function included.
   C            = 0xC             = Chip version C
                                  = Security function not included.
   D            = 0xD             = Chip version D
                                  = Security function included.
*/
#define BSP_CFG_MCU_PART_VERSION        (0xB)
```

[RX26T]

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

```
/* Whether PGA differential input, Encryption and USB are included or not.
   Character(s) = Value for macro = Description
   A = 0xA  = Only CAN 2.0 protocol supported, without TSIP-Lite
   B = 0xB  = Only CAN 2.0 protocol supported, with TSIP-Lite
   C = 0xC  = CAN FD protocol supported, without TSIP-Lite
   D = 0xD  = CAN FD protocol supported, with TSIP-Lite
*/
#define BSP_CFG_MCU_PART_FUNCTION   (0xD)
```

[RX66T and RX72T (On the RX72T, portions of the comment below following “= PGA” differ.)]

Change the value in the following macro in r_bsp_config.h in the r_config folder to 0xE, 0xF, or 0x10.

```
/* Whether PGA differential input, Encryption and USB are included or not.
   Character(s) = Value for macro = Description
   A = 0xA  = PGA differential input included, Encryption module not included,
              USB module not included
   B = 0xB  = PGA differential input not included, Encryption module not
              included, USB module not included
   C = 0xC  = PGA differential input included, Encryption module not included,
              USB module included
   E = 0xE  = PGA differential input included, Encryption module included,
              USB module not included
   F = 0xF  = PGA differential input not included, Encryption module included,
              USB module not included
   G = 0x10 = PGA differential input included, Encryption module included,
              USB module included
*/
#define BSP_CFG_MCU_PART_FUNCTION    (0xE)
```

[RX66N, RX671, RX72M, and RX72N]

Change the value in the following macro in r_bsp_config.h in the r_config folder to 0x11.

```
/* Whether Encryption is included or not.
   Character(s) = Value for macro = Description
   D            = 0xD             = Encryption module not included
   H            = 0x11            = Encryption module included
*/
#define BSP_CFG_MCU_PART_FUNCTION       (0x11)
```

[RX65N]

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

```
/* Whether Encryption and SDHI/SDSI are included or not.
   Character(s) = Value for macro = Description
   A            = false = Encryption module not included, SDHI/SDSI module not included
   B            = false = Encryption module not included, SDHI/SDSI module included
   D            = false = Encryption module not included, SDHI/SDSI module included
   E            = true  = Encryption module included, SDHI/SDSI module not included
   F            = true  = Encryption module included, SDHI/SDSI module included
   H            = true  = Encryption module included, SDHI/SDSI module included
*/
#define BSP_CFG_MCU_PART_ENCRYPTION_INCLUDED    (true)
```

2.3 Supported Toolchain

The operation of the TSIP 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_tsip_rx_if.h.

2.5 Integer Types

The TSIP 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_tsip_rx_config.h you can turn the corresponding functions on or off.

Configuration options in r_config/r_tsip_rx_config.h		
Definition	Default Value	Remarks
TSIP_AES_128_ECB_ENCRYPT	(0)	—
TSIP_AES_256_ECB_ENCRYPT	(0)	—
TSIP_AES_128_ECB_DECRYPT	(0)	—
TSIP_AES_256_ECB_DECRYPT	(0)	—
TSIP_AES_128_CBC_ENCRYPT	(0)	—
TSIP_AES_256_CBC_ENCRYPT	(0)	—
TSIP_AES_128_CBC_DECRYPT	(0)	—
TSIP_AES_256_CBC_DECRYPT	(0)	—
TSIP_AES_128_CTR	(0)	—
TSIP_AES_256_CTR	(0)	—
TSIP_AES_128_GCM_ENCRYPT	(0)	—
TSIP_AES_256_GCM_ENCRYPT	(0)	—
TSIP_AES_128_GCM_DECRYPT	(0)	—
TSIP_AES_256_GCM_DECRYPT	(0)	—
TSIP_AES_128_CMAC	(0)	—
TSIP_AES_256_CMAC	(0)	—
TSIP_AES_128_CCM_ENCRYPT	(0)	—
TSIP_AES_256_CCM_ENCRYPT	(0)	—
TSIP_AES_128_CCM_DECRYPT	(0)	—
TSIP_AES_256_CCM_DECRYPT	(0)	—
TSIP_AES_128_KEY_WRAP	(0)	—
TSIP_AES_256_KEY_WRAP	(0)	—
TSIP_TDES_ECB_ENCRYPT	(0)	—
TSIP_TDES_ECB_DECRYPT	(0)	—
TSIP_TDES_CBC_ENCRYPT	(0)	—
TSIP_TDES_CBC_DECRYPT	(0)	—
TSIP_ARC4_ENCRYPT	(0)	—
TSIP_ARC4_DECRYPT	(0)	—
TSIP_SHA_1	(0)	—
TSIP_SHA_256	(0)	—
TSIP_MD5	(0)	—
TSIP_SHA_1_HMAC	(0)	—
TSIP_SHA_256_HMAC	(0)	—
TSIP_RSAES_1024	(0)	—
TSIP_RSAES_2048	(0)	—
TSIP_RSAES_3072	(0)	—
TSIP_RSAES_4096	(0)	—
TSIP_RSASSA_1024	(0)	—
TSIP_RSASSA_2048	(0)	—
TSIP_RSASSA_3072	(0)	—
TSIP_RSASSA_4096	(0)	—

Configuration options in r_config/r_tsip_rx_config.h		
TSIP_RSA_RETRY_COUNT_FOR_RSA_KEY_GENERATION	(5120*2)	The retry count for RSA key generation. The NIST FIPS 186-4 standard recommends a value of (5 * (key length / 2). The default value is calculated using this formula with a key length of 2,048 and then doubled to provide an extra margin.
TSIP_ECDSA_P192	(0)	—
TSIP_ECDSA_P224	(0)	—
TSIP_ECDSA_P256	(0)	—
TSIP_ECDSA_P384	(0)	—
TSIP_ECDH_P256	(0)	—
TSIP_USER_SHA_384_ENABLED*1	(0)	—
TSIP_TLS	(0)	—
TSIP_SECURE_BOOT	(0)	—
TSIP_FIRMWARE_UPDATE	(0)	—
TSIP_MULTI_THREADING*1	(0)	—

Note: 1. Refer to 4.3, User-Defined Functions, for instructions on defining user-defined functions.

Though each function can be disabled with setting the configuration, some unreferenced internal functions are not ommited in the building process when a specific compiler is used. In order to delete unreferenced internal finctions to reduce the code size, please set optimization settings of the compiler or linker adequately.

For example, in CC-RX V.3.07, unreferenced functions can be deleted with enabling -optimize=symbol_delete setting in optimization setting of the linker.

2.7 Data Structures

The data structures used by the TSIP driver are defined in r_tsip_rx_if.h.

2.8 Return Values

The return values of the TSIP driver API functions are listed below. The enumerated types of return values are defined in r_tsip_rx_if.h.

```
typedef enum e_tsip_err
{
   TSIP_SUCCESS=0,
   TSIP_ERR_FAIL,                //  Self-check failed to terminate normally,
                                 //  illegal MAC detected by R_TSIP_VerifyFirmwareMAC,
                                 //  or R_TSIP_ API function internal error.
   TSIP_ERR_RESOURCE_CONFLICT,   //  A resource conflict occurred because a resource required by
                                 //  the processing routine was in use by another processing routine.
   TSIP_ERR_RETRY,               //  Self-check terminated with an error. Run the function again.
   TSIP_ERR_KEY_SET,             //  An invalid wrapped key was input.
   TSIP_ERR_AUTHENTICATION,      //  Authentication failed,
                                 //  or signature verification using RSASSA-PKCS1-V.1.5 failed.
   TSIP_ERR_CALLBACK_UNREGIST,   //  Callback function not registered.
   TSIP_ERR_PARAMETER,           //  Invalid input date.
   TSIP_ERR_PROHIBIT_FUNCTION,   //  An invalid function call occurred.
   TSIP_RESUME_FIRMWARE_GENERATE_MAC,//  There is additional processing. It is necessary to call
                                     //  the API again.
   TSIP_ERR_VERIFICATION_FAIL,   //  Verification of TLS 1.3 handshaking failed.
   TSIP_ERR_ALREADY_OPEN,        //  TSIP drver is already open.
}e_tsip_err_t
```

2.9 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 e^2 studio
Using Smart Configurator in e^2 studio allows you to add the FIT module to your project automatically. Refer to the application note “Renesas e^2 studio Smart Configurator User Guide” (R20AN0451) for details.

(2) Adding the FIT module to your project using FIT Configurator in e^2 studio
Using FIT Configurator in e^2 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 e^2 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.

2.10 *for*, *while*, and *do while* Statements

The TSIP 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. TSIP Driver Usage

The TSIP driver for the RX Family provides the following functions:

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

The keys handled by the TSIP 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 TSIP. In the case of the RX TSIP driver, this type of opaque key is called a wrapped key. The TSIP driver implements secure key management by wrapping keys using the hardware unique key. This provides key confidentiality and detection of tampering outside of the TSIP.

The unauthorized access monitoring by the TSIP 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 TSIP 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-Final model, and APIs for single-part operations are provided for other ciphers.

3.1 Recovering after Unauthorized Access Detection

Unauthorized access monitoring by the TSIP 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 TSIP 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 TSIP driver once with R_TSIP_Close() and restart the TSIP with R_TSIP_Open(), or reset the device.

3.2 Avoiding TSIP Access Conflicts

All RX Family products provided with a TSIP allow use of only one channel of the TSIP. The TSIP driver, like many peripheral IP drivers, takes over the hardware resources of the TSIP while driver APIs are running.

Among the APIs that provide multi-part operations, the symmetric key cryptography and HMAC functions continue to occupy the TSIP hardware resources until the series of multi-part operations is complete.

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

1. While a TSIP driver API is being executed, other TSIP driver APIs must not be executed.
2. In the case of symmetric key cryptography and HMAC functions, other TSIP driver APIs cannot execute until the series of operations (Init/Update/Final) currently being processed is complete.

Note that the message digest generation function can execute other TSIP driver APIs while a series of multi-part operations is in progress.

If a TSIP driver API causes a TSIP hardware resource access conflict, the API returns TSIP_ERR_RESOURCE_CONFLICT or TSIP_ERR_PROHIBIT_FUNCTION.

Use one of the following methods to avoid TSIP access conflicts when using the TSIP driver:

- Use the APIs in an order that does not cause TSIP access conflicts.
- Use the TSIP access conflict avoidance function provided by the TSIP driver.
  Use system calls (mutexes, semaphores, etc.) for exclusive control in a real-time OS to implement a user_lock_function and user_unlock_function as user-defined functions for the TSIP driver. Enable TSIP_MULTI_THREADING in r_tsip_rx_config.h to turn on the TSIP access conflict avoidance function provided by the TSIP driver.
  It is also possible to have the TSIP driver use multiple threads in a real-time OS.

3.3 BSP FIT Module Integration

The TSIP driver uses the BSP FIT module internally as described in section 2.2. When using the TSIP driver, link to the following APIs. For details, refer to the application note “Board Support Package Module Using Firmware Integration Technology” (R01AN1685xJxxxxxx).

- R_BSP_RegisterProtectEnable()
- R_BSP_RegisterProtectDisable()
- R_BSP_InterruptsEnable()
- R_BSP_InterruptsDisable()

It is assumed that BSP startup has completed before these APIs are called. If BSP startup is not used, call R_BSP_StartupOpen() beforehand. Also initialize internal variables used within the above APIs.

3.4 Single-Part and Multi-Part Operations

There are two types of APIs provided by the TSIP 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 (message digest generation and HMAC functions), and APIs for single-part operations are provided for other ciphers.

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

All multi-part operations conform to the following pattern:

Init: Initialize and start the operation.
If initialization is successful, the operation is active. If initialization fails, the operation enters an error state.

Update: Update the operation.
The update function can provide additional parameters, supply data for processing, or generate output.
If updating is successful, the operation remains active. If updating fails, the operation enters an error state.

Final: Call the applicable finalizing function to end the operation.
This function accepts any final input, generates any final output, and then releases any resources associated with the operation.

If finalizing is successful, the operation returns to the inactive state. If updating fails, the operation enters an error state.

3.5 Initializing and Terminating the Driver

The driver provides APIs for the following driver management operations:

No.	API	Description
1	R_TSIP_Open	Opens the TSIP driver. Initializes the TSIP and performs a self-test of the TSIP's fault detection and random number generator circuits.
2	R_TSIP_Close	Closes the TSIP driver.
3	R_TSIP_SoftwareReset	Resets the TSIP driver.
4	R_TSIP_GetVersion	Gets the version number of the TSIP driver.

Applications using the driver must call R_TSIP_Open() to initialize the driver before using other functions. Also, when terminating use of the driver, R_TSIP_Close() must be called.

If problems occur while using the driver and there is a need to reset the driver and its control target, the TSIP, it is necessary to call R_TSIP_SoftwareReset() or R_TSIP_Open() after calling R_TSIP_Close(). Call R_TSIP_SoftwareReset() to apply a reset without resuming TSIP driver processing, or call R_TSIP_Open() to resume TSIP driver processing.

R_TSIP_Open() performs a self-test to detect hardware failure of the TSIP and to check for abnormalities in the random number generation 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.

3.6 Random Number Generation

The driver provides an API for generating random numbers.

No.	API	Description
1	R_TSIP_GenerateRandomNumber	Generates random numbers using the CTR-DRBG method described in NIST SP800-90A.

3.7 Key Management

The driver provides APIs for the following key management operations:

No.	API	Description
1	R_TSIP_GenerateUpdateKeyRingKeyIndex R_TSIP_GenerateAes*XXX*KeyIndex R_TSIP_GenerateTdesKeyIndex R_TSIP_GenerateArc4KeyIndex R_TSIP_GenerateSha*XXX*HmacKeyIndex R_TSIP_GenerateRsa*XXX*PublicKeyIndex R_TSIP_GenerateRsa*XXX*PrivateKeyIndex R_TSIP_GenerateEccP*XXX*PublicKeyIndex R_TSIP_GenerateEccP*XXX*PrivateKeyIndex R_TSIP_GenerateTlsRsaPublicKeyIndex	These key injection APIs use the Renesas Key Wrap service to convert a user key into a wrapped key wrapped using an HUK. They can be used for key injection at the factory. [Aes] XXX = 128, 256 [Hmac] XXX = 1, 256 [Rsa] XXX = 1024, 2048, 3072*1, 4096*1 [Ecc] XXX = 192, 224, 256, 384
2	R_TSIP_UpdateAes*XXX*KeyIndex R_TSIP_UpdateTdesKeyIndex R_TSIP_UpdateArc4KeyIndex R_TSIP_UpdateRsa*XXX*PublicKeyIndex R_TSIP_UpdateRsa*XXX*PrivateKeyIndex R_TSIP_UpdateEccP*XXX*PublicKeyIndex R_TSIP_UpdateEccP*XXX*PrivateKeyIndex	These key update APIs use an update keyring to convert a user key into a wrapped key wrapped using an HUK. They can be used to update keys in the field. [Aes] XXX = 128, 256 [Hmac] XXX = 1, 256 [Rsa] XXX = 1024, 2048, 3072*1, 4096*1 [Ecc] XXX = 192, 224, 256, 384
3	R_TSIP_GenerateAes*XXX*RandomKeyIndex R_TSIP_GenerateTdesRandomKeyIndex R_TSIP_GenerateArc4RandomKeyIndex R_TSIP_GenerateRsa*XXX*RandomKeyIndex R_TSIP_GenerateEccP*XXX*RandomKeyIndex	These APIs generate a random key and convert it to a wrapped key. [Aes] XXX = 128, 256 [Rsa] XXX = 1024, 2048 [Ecc] XXX = 192, 224, 256, 384

Note: 1. Only public keys are provided in these key lengths.

Figure 3.1 illustrates key handling in cryptographic operations performed by the TSIP driver.

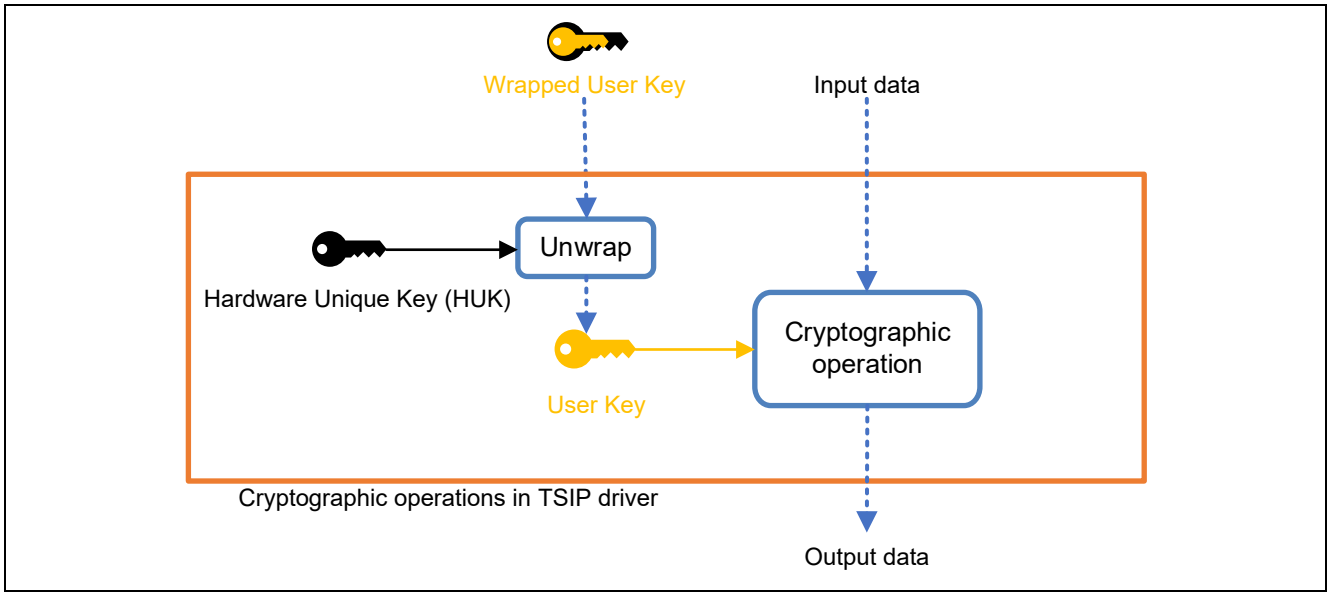


Figure 3.1 Key Handling in TSIP Driver Cryptographic Operations

The keys handled by the TSIP driver (input and output keys) are opaque keys wrapped using a device-specific key called an HUK, which is accessible only by the TSIP. In the case of the RX TSIP driver, this type of opaque key is called a wrapped key. The TSIP 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 TSIP. The wrapping of the warpped key can be unlocked only by the TSIP, and the unwrapped key exists only within the TSIP during cryptographic processing. Since the warpped key has been wrapped using a device-specific key, it cannot be unwrapped using a different device-specific key, even if the warpped key is copied from the nonvolatile memory of one device to another device.

3.7.1 Key Injection and Updating

Key injection and key updating provide a mechanism enabling secure delivery of user keys by converting them into warpped keys wrapped using an HUK. Figure 3.2 shows the key injection and key updating operation sequence, including use of the Renesas Key Wrap service.

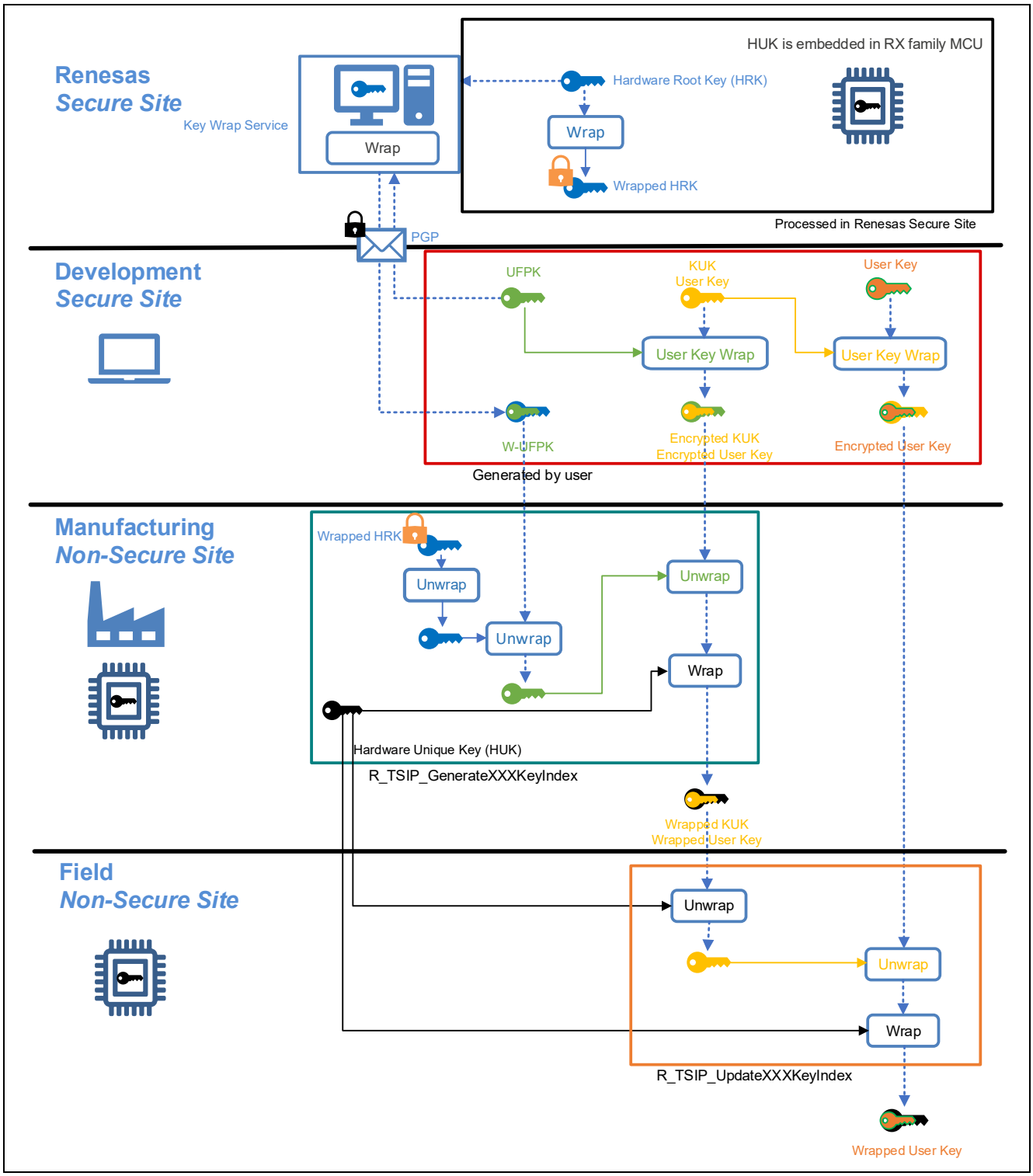


Figure 3.2 Key Injection and Key Updating Operation Sequence

Wrapping a secret key using an HUK involves encryption and adding a MAC, while wrapping a public key only involves adding a MAC. The public warpped key is not encrypted, so the plaintext public key can be extracted from it. For details, refer to 3.7.3, Plaintext Public Key Extraction.

Figure 3.3 shows the user key wrapping scheme used when a UFPK or update key (Key Update Key, KUK) is used for wrapping during key injection and key updating. The first 128 bits of the UFPK or KUK for wrapping is used as the key, and the user key is encrypted in AES-128 CBC mode. Then the trailing 128 bits of the of the provisioning key or update keyring for wrapping is used to calculate the MAC of the user key using AES-128 CBC-MAC. The MAC of the user key is concatenated to the user key and both are encrypted to generate an encrypted key.

For the data formats of user keys and wrapped keys (encrypted keys), refer to 7.3, User Key Encryption Formats.

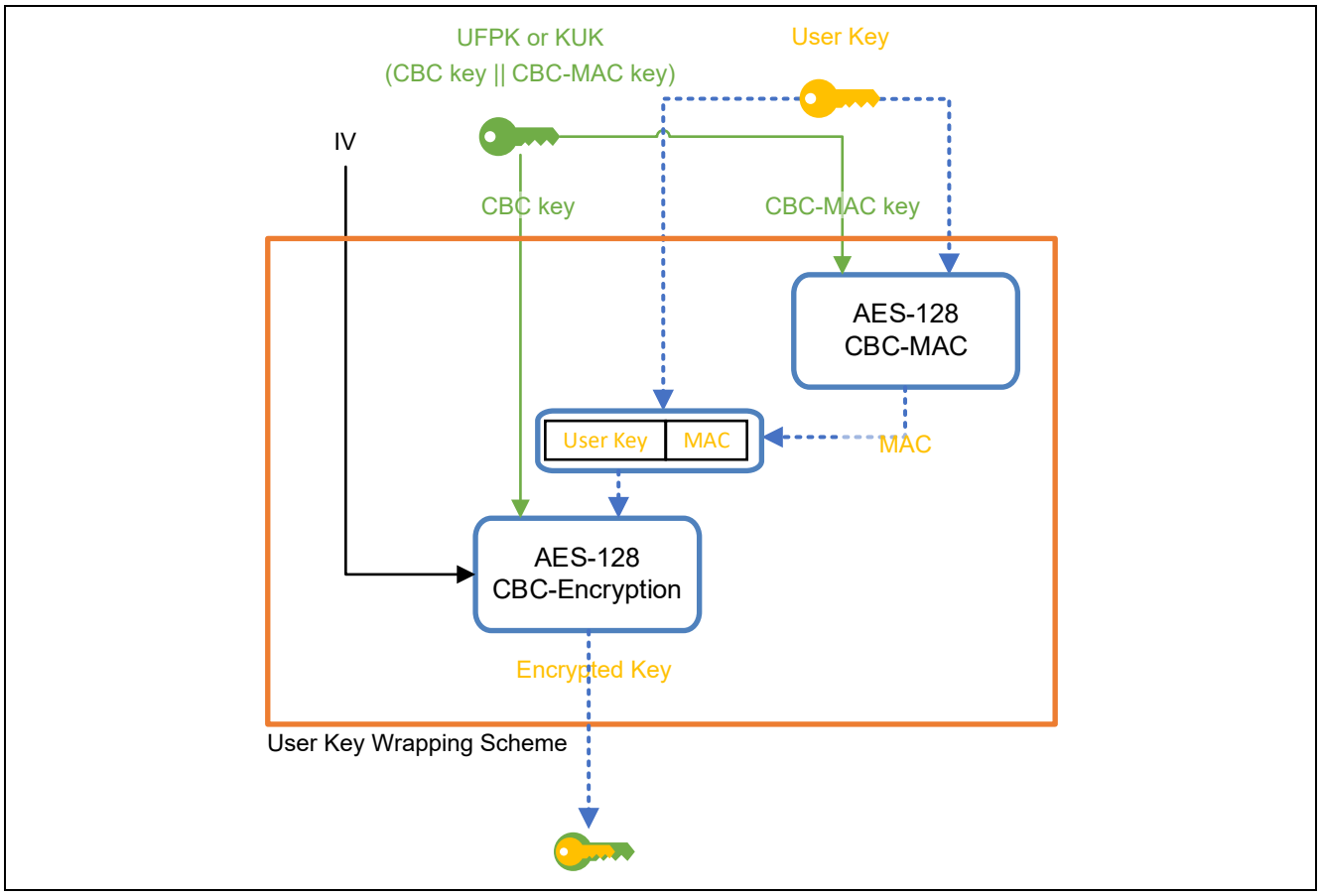


Figure 3.3 User Key Wrapping Scheme during Key Injection and Key Updating

The above method to generate an encrypted key is shown below with C language format.

```
-------------------------------------------------------------------------------------------------------------------------------------------------------
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 in above description mean below processes/

- AES_128_ENCRYPT(Key, Data) : Encrypt Data with AES128 ECB mode with using encrypting key Key
- xor_16byte(data1, data2) : XOR calculation between data1 and data2 where each value is 16 byte length

And the size of 1 element of each array(CBCkey[], CBCMACkey[], MAC[], iv[], user_key[], encrypted_key[]) is 4 byte.

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.

3.7.2 Key Generation

In key generation, the random number generation functionality of the TSIP is used to generate a new key, which is then output in an opaque format (wrapped key) that is usable by the TSIP driver.

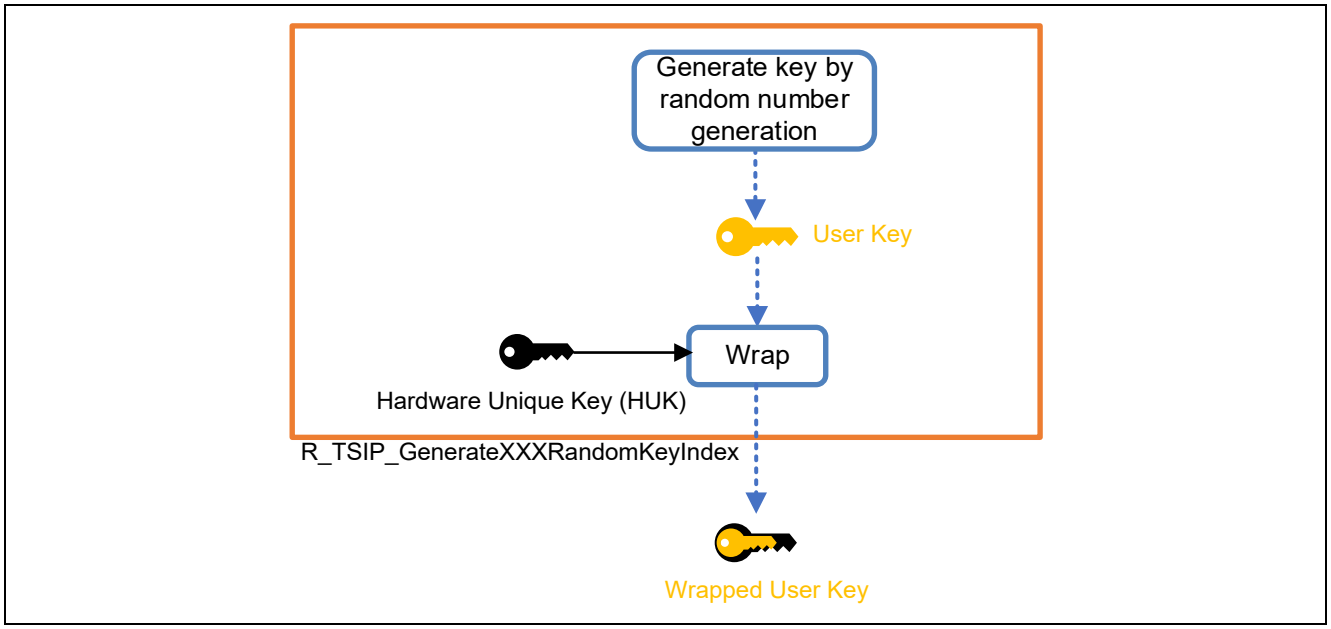


Figure 3.4 Key Generation Sequence

3.7.3 Plaintext Public Key Extraction

The value of public wrapped key is not encrypted, so the plaintext public key can be extracted from the format of the wrapped key.

The formats of the wrapped keys of public keys used by the various asymmetric cryptographic algorithms are listed in 7.4, Public Key Index Formats for Asymmetric Cryptography. Refer to this section when extracting the plaintext of public keys from their formats of wrapped keys.

3.8 Symmetric Key Cryptography

The driver provides APIs for the following types of symmetric cryptographic operations:

No.	API	Description
1	R_TSIP_AesXXX[Mode]Encrypt* R_TSIP_AesXXX[Mode]Decrypt* R_TSIP_AesXXXCtr* R_TSIP_Tdes[Mode]Encrypt* R_TSIP_Tdes[Mode]Decrypt* R_TSIP_Arc4Encrypt* R_TSIP_Arc4Decrypt*	Symmetric key cryptography AES 128-/256-bit: ECB, CBC, CTR encryption and decryption TDES: ECB, CBC encryption and decryption ARC4 XXX = 128, 256 Mode = Ecb, Cbc
2	R_TSIP_AesXXXGcmEncrypt* R_TSIP_AesXXXGcmDecrypt* R_TSIP_AesXXXCcmEncrypt* R_TSIP_AesXXXCcmDecrypt*	Authenticated encryption with associated data (AEAD) AES-GCM, AES-CCM 128-/256-bit encryption and decryption XXX = 128, 256
3	R_TSIP_AesXXXCmacGenerate* R_TSIP_AesXXXCmacVerify*	Message authentication codes (MAC) AES-CMAC 128-/256-bit MAC operation XXX = 128, 256
4	R_TSIP_AESXXXKeyWrap R_TSIP_AESXXXKeyUnwrap	AES key wrap/ unwrap XXX = 128, 256

* = Init, Update, Final

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 encryption operations for each AES mode operate as follows:

Call R_TSIP_AesXXX[Mode]EncryptInit() to specify the required key and initial vector. Call the R_TSIP_AesXXX[Mode]EncryptUpdate() function for the chunks of data comprising the plaintext message in consecutive block units. To complete the encryption operation, call R_TSIP_AesXXX[Mode]EncryptFinal().

The decryption operations for each AES mode operate as follows:

Call R_TSIP_AesXXX[Mode]DecryptInit() to specify the required key and initial vector. Call the R_TSIP_AesXXX[Mode]DecryptUpdate() function for the chunks of data comprising the ciphertext message in consecutive block units. To complete the decryption operation, call R_TSIP_AesXXX[Mode]DecryptFinal().

The TDES and ARC4 cryptographic APIs operate in the same way as those for AES.

Note: It is not recommended to use TDES and ARC4 because these algorithms are obsoleted.

3.8.2 Authenticated Encryption with Associated Data (AEAD)

The AES-GCM encryption operations operate as follows:

Call R_TSIP_AesXXXGcmEncryptInit() to specify the required key and initial vector.

Call the R_TSIP_AesXXXGcmEncryptUpdate() function for the chunks of data comprising the plaintext message in consecutive block units. To complete the encryption operation, call R_TSIP_AesXXXGcmEncryptFinal().

The AES-GCM decryption operations operate as follows:

Call R_TSIP_AesXXXGCMDecryptInit() to specify the required key and initial vector. Call the R_TSIP_AesXXXGCMDecryptUpdate() function for the chunks of data comprising the ciphertext message in consecutive block units. To complete the decryption operation, compute the authentication tag, and verify it against a reference value, call R_TSIP_AesXXXGcmDecryptFinal().

The AES-CCM cryptographic APIs operate in the same way as those for AES-GCM.

3.8.3 Message Authentication Code (MAC)

The MAC generation operations using AES-CMAC operate as follows:

Call R_TSIP_AesXXXCmacGenerateInit() to specify the required key. Call the R_TSIP_AesXXXCmacGenerateUpdate() function for the consecutive chunks of data comprising the message. To complete generating the MAC for the message, call R_TSIP_AesXXXCmacGenerateFinal().

AES-CMAC verification operates as follows:

Call R_TSIP_AesXXXCmacVerifyInit() to specify the required key.

Call the R_TSIP_AesXXXCmacVerifyUpdate() function for the chunk of data comprising the message. To verify the MAC of the message, call R_TSIP_AesXXXCmacVerifyFinal() and specify the MAC required for verification.

3.9 Asymmetric Cryptography

The driver provides APIs for the following asymmetric cryptographic operations:

No.	API	Description
1	R_TSIP_RsaesPkcsXXXEncrypt R_TSIP_RsaesPkcsXXXDecrypt R_TSIP_RsaesOaepXXXEncrypt R_TSIP_RsaesOaepXXXDecrypt	[RSAES-PKCS1-V1_5 encrypt] XXX = 1024, 2048, 3072, 4096 [RSAES-PKCS1-V1_5 decrypt] XXX = 1024, 2048 [RSAES-OAEP encrypt/decrypt] *XXX* = 1024, 2048
2	R_TSIP_RsassaPkcsXXXSignatureGenerate R_TSIP_RsassaPkcsXXXSignatureVerification R_TSIP_RsassaPssXXXSignatureGenerate R_TSIP_RsassaPssXXXSignatureVerification R_TSIP_EcdsaPXXXSignatureGenerate R_TSIP_EcdsaPXXXSignatureVerification	[RSASSA-PKCS1-V1_5 Sign] XXX = 1024, 2048 [RSASSA-PKCS1-V1_5 verify] XXX = 1024, 2048, 3072, 4096 [RSASSA-PSS sign/verify] XXX = 1024, 2048 [ECDSA sign/verify] XXX = 192, 224, 256, 384

Only APIs that implement encryption, decryption, signature generation, and verification as single-part operations are provided for asymmetric cryptographic operations.

3.10 Hash Functions

The driver provides APIs for the following hash operations:

No.	API	Description
1	R_TSIP_ShaXXX* R_TSIP_Md5* R_TSIP_GetCurrentHashDigestValue	Message digests (hash functions) SHA-1, SHA-256 XXX = 1, 256
2	R_TSIP_ShaXXXHmacGenerate* R_TSIP_ShaXXXHmacVerify*	Message authentication codes (MAC) HMAC: HMAC-SHA1, HMAC-SHA256 XXX = 1, 256

* = Init, Update, Final

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

3.10.1 Message Digest (Hash Function)

The hash operation APIs are used as follows:

Call R_TSIP_ShaXXXInit() to specify the newly allocated work area for the operation. Call R_TSIP_ShaXXXUpdate() for the consecutive chunks of data comprising the message. Call R_TSIP_ShaXXXFinal() to calculate the digest of the message. R_TSIP_GetCurrentHashDigestValue() can be called after R_TSIP_ShaXXXUpdate() to retrieve data while the hash operation is in progress.

The MD5 APIs are used in the same way as those for SHA.

Note: It is not recommended to use MD5 and SHA-1 because these algorithms are obsoleted.

3.10.2 Message Authentication Code (HMAC)

The HMAC generation APIs are used as follows:

Call R_TSIP_ShaXXXHmacGenerateInit() to specify the required key and newly allocated work area for the operation. Call R_TSIP_ShaXXXHmacGenerateUpdate() for the consecutive chunks of data comprising the message. To complete MAC generation for the message, call R_TSIP_ShaXXXHmacGenerateFinal().

The HMAC verification APIs are used as follows:

Call R_TSIP_ShaXXXHmacVerifyInit() to specify the required key and newly allocated work area for the operation. Call R_TSIP_ShaXXXHmacVerifyUpdate() for the consecutive chunks of data comprising the message. To verify the MAC of a message, call R_TSIP_ShaXXXHmacVerifyFinal() and specify the required MAC for verification.

3.11 ECDH Key Exchange

The driver provides APIs for the following ECDH key exchange operations:

No.	API	Description
1	R_TSIP_EcdhP256Init	Performs ECDH key exchange initialization.
2	R_TSIP_EcdhP256ReadPublicKey	Verifies the public key and signature of the other ECDH key exchange party and extracts the public key.
3	R_TSIP_EcdhP256MakePublicKey	Generates a temporary key (ephemeral key) and signature from a random number.
4	R_TSIP_EcdhP256CalculateSharedSecretIndex	Calculates the shared secret Z using the ECDH key exchange algorithm.
5	R_TSIP_EcdhP256KeyDerivation	Derives the key from the shared secret Z.
6	R_TSIP_EcdhP256SshKeyDerivation	Derives the key to use in SSH from the shared secret Z.

The ECDH key exchange APIs conform to NIST SP800-56A.

The way the APIs are used for key exchange differs depending on whether a previously injected key (static key) or a temporary key generated from a random number (ephemeral key) is used. Usage of the APIs with the Ephemeral Unified Model, One-Pass Diffie-Hellman, and Static Unified Model key exchange schemes defined by NIST SP800-56A is described below.

3.11.1 Ephemeral Unified Model (2e, 0s)

Under the Ephemeral Unified Model, an ephemeral key pair is generated between the other key exchange party (party U) and yourself (party V), and an ECDH key exchange takes place.

1. Execute R_TSIP_EcdhP256Init() to initialize the key exchange scheme. Specify 0 (ECDHE) for the key_type parameter of R_TSIP_EcdhP256Init().
2. Use R_TSIP_EcdhP256ReadPublicKey() to verify the public key (QeU) and signature (sig) received from party U. If the signature is correct, QeU_index is output.
3. Execute R_TSIP_EcdhP256MakePublicKey() to generate an ephemeral key pair (QeV and deV_index).
4. Input QeU_index and deV_index to R_TSIP_EcdhP256CalculateSharedSecretIndex() to generate shared_secret_index.
5. Input shared_secret_index and other information to R_TSIP_EcdhP256KeyDerivation() to derive the key. Party U and party V must use the same value for other information.
   When to derive the key to use in SSH, execute R_TSIP_EcdhP256SshKeyDerivation() instead of the function.

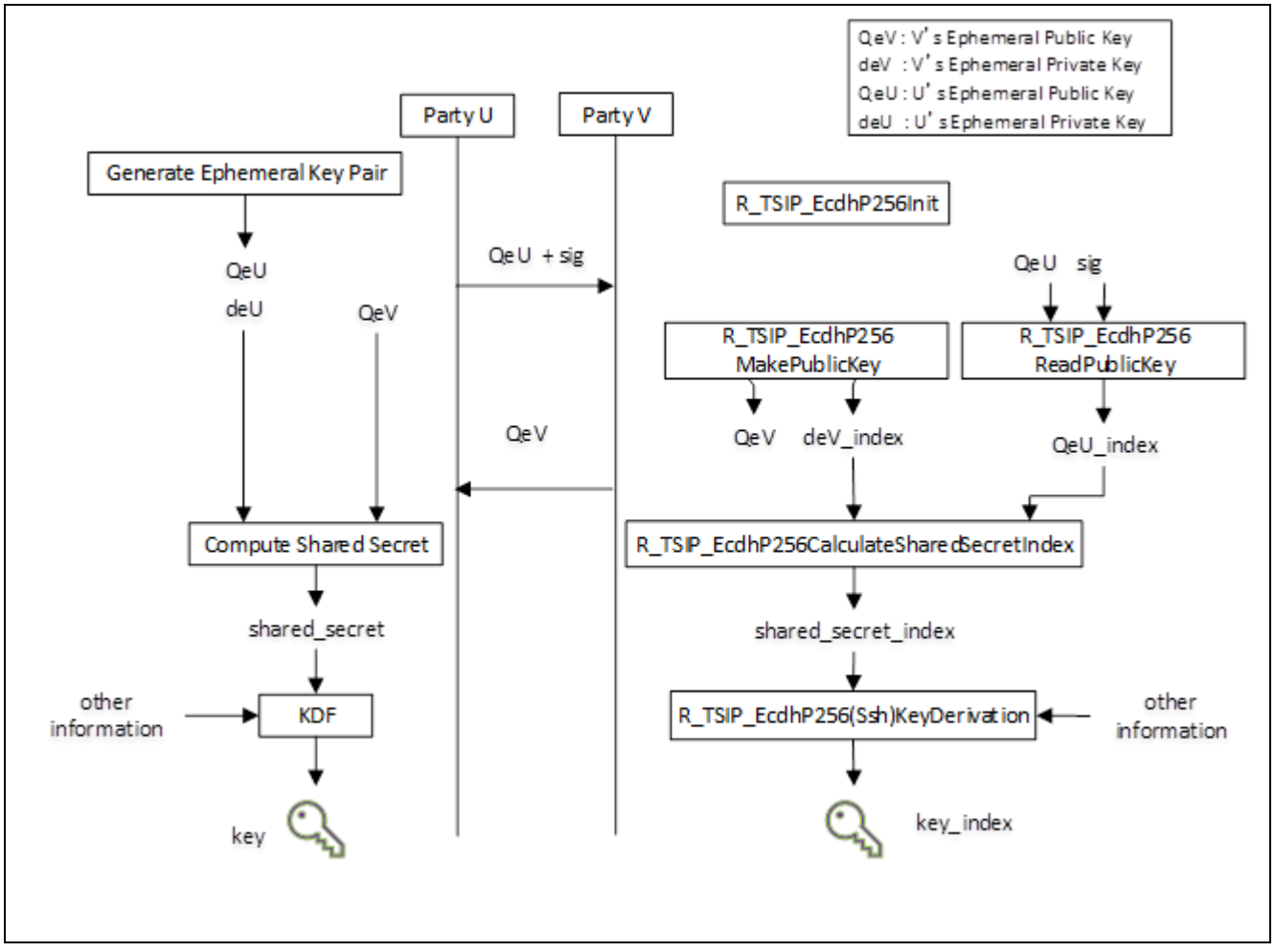


Figure 3.5 Key Exchange Scheme Using Ephemeral Unified Model

3.11.2 One-Pass Diffie-Hellman (1e, 1s)

Under One-Pass Diffie-Hellman, an ephemeral key generated by the other key exchange party (party U) and your own (party V's) static key are used for ECDH key exchange. Before commencing the key exchange scheme, party U must obtain party V's public key (QsV).

1. Execute R_TSIP_EcdhP256Init() to initialize the key exchange scheme. Specify 1 (ECDH) for the key_type parameter of R_TSIP_EcdhP256Init().
2. Use R_TSIP_EcdhP256ReadPublicKey() to verify the public key (QeU) and signature (sig) received from party U. If the signature is correct, QeU_index is output.
3. Generate a wrapped key (dsV_index) for the secret key (dsV) corresponding to the public key of party V (QsV), which is in the possession of party U.
4. Input QeU_index and deV_index to R_TSIP_EcdhP256CalculateSharedSecretIndex() to generate shared_secret_index.
5. Input shared_secret_index and other information to R_TSIP_EcdhP256KeyDerivation() to derive the key. Party U and party V must use the same value for other information.
   When to derive the key to use in SSH, execute R_TSIP_EcdhP256SshKeyDerivation() instead of the function.

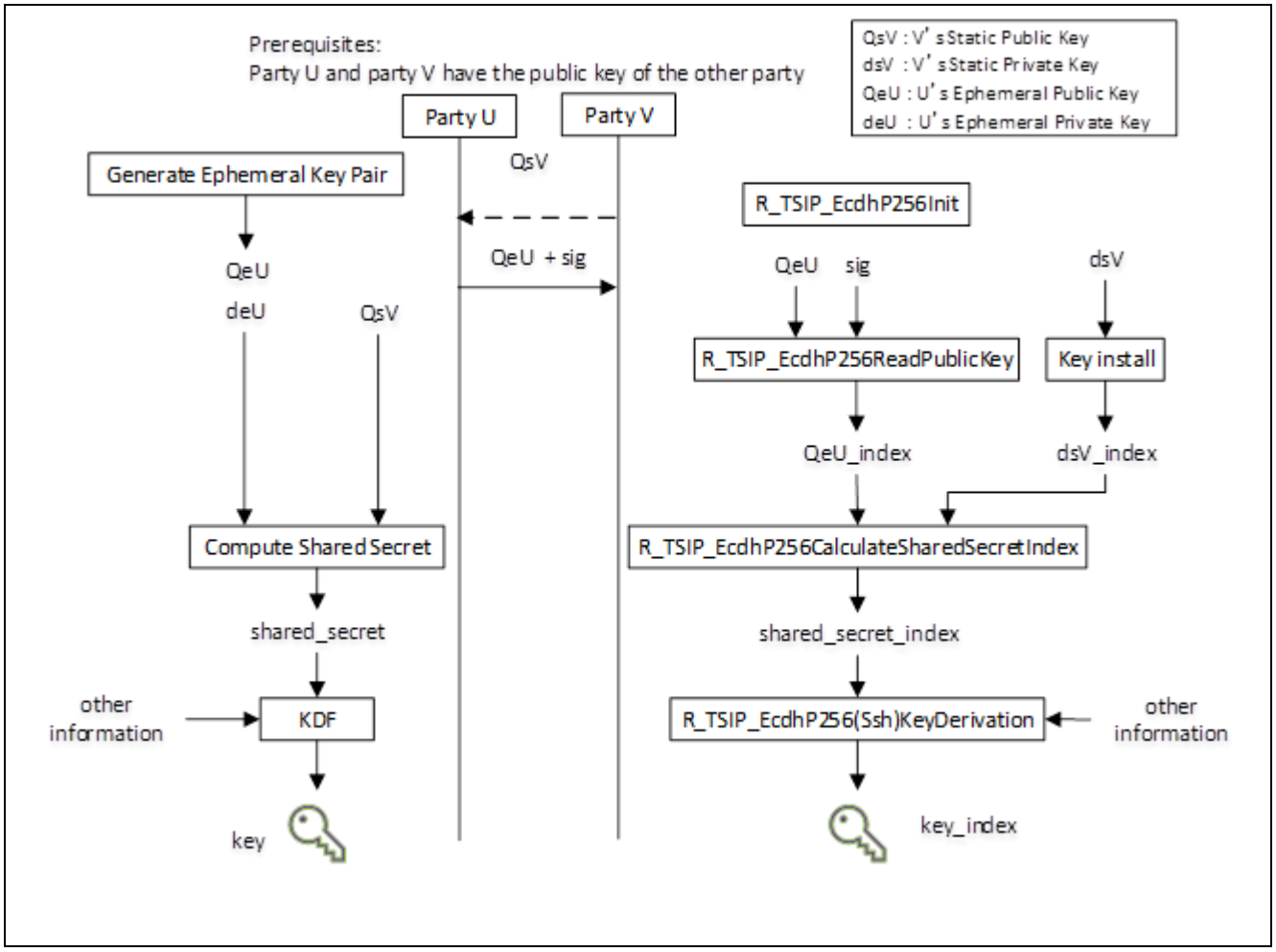


Figure 3.6 Key Exchange Scheme Using One-Pass Diffie-Hellman

3.11.3 Static Unified Model (0e, 2s)

Under the Static Unified Model, the static key of the other key exchange party (party U) and your own (party V's) static key are used for ECDH key exchange. Before commencing the key exchange scheme, both parties must obtain each other's public keys.

1. Execute R_TSIP_EcdhP256Init() to initialize the key exchange scheme. Specify 1 (ECDH) for the key_type parameter of R_TSIP_EcdhP256Init().
2. Use R_TSIP_EcdhP256ReadPublicKey() to verify the public key (QeU) and signature (sig) received from party U. If the signature is correct, QeU_index is output.
3. Generate a wrapped key (dsV_index) for the secret key (dsV) corresponding to the public key of party V (QsV), which is in the possession of party U.
4. Input QeU_index and deV_index to R_TSIP_EcdhP256CalculateSharedSecretIndex() to generate shared_secret_index.
5. Input shared_secret_index and other information to R_TSIP_EcdhP256KeyDerivation() to derive the key. Party U and party V must use the same value for other information. Also, include the nonce generated by party U in other information. This prevents the same identical key from being derived each time static keys are used for ECDH key exchange.
   When to derive the key to use in SSH, execute R_TSIP_EcdhP256SshKeyDerivation() instead of the function.

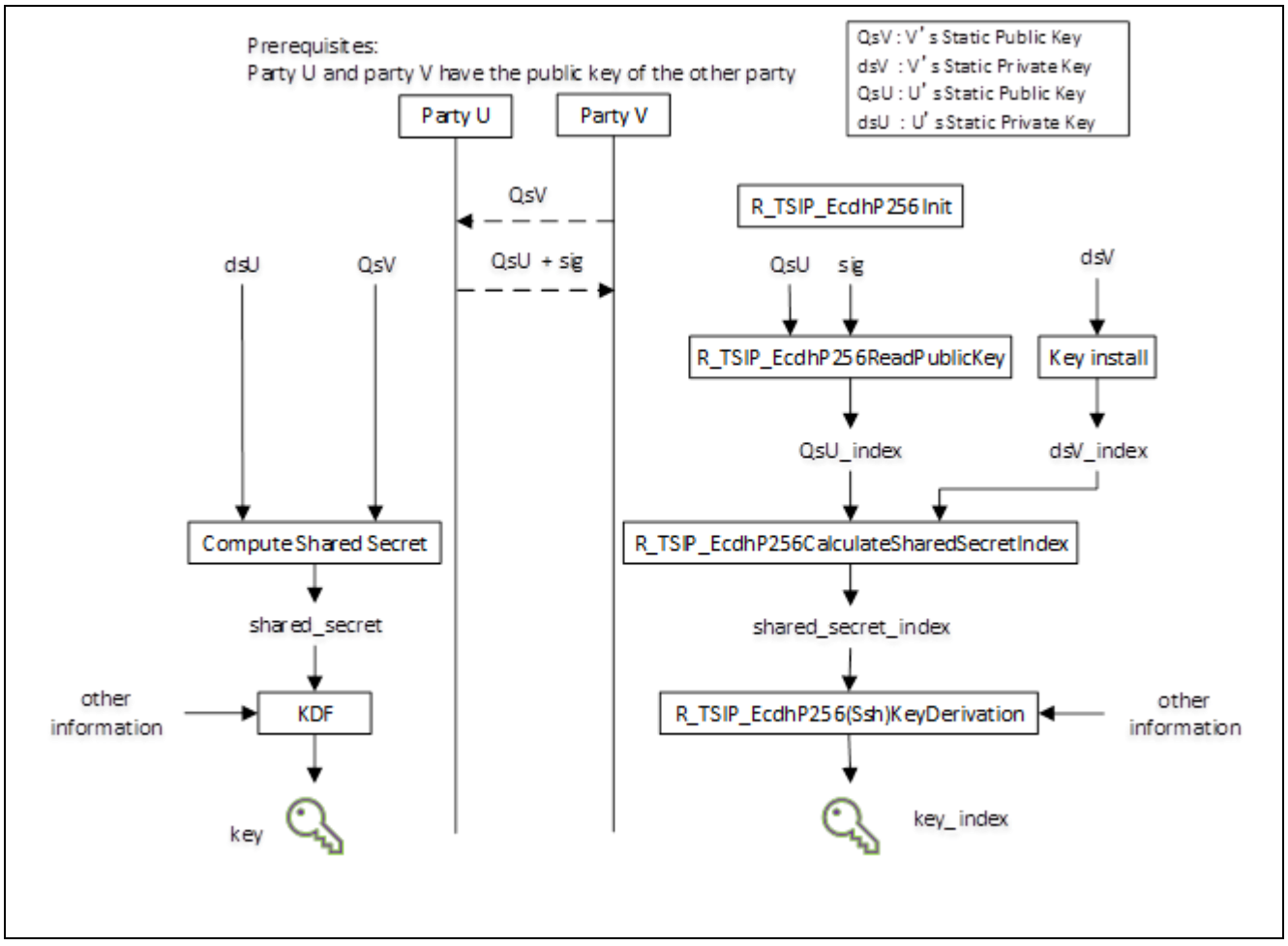


Figure 3.7 Key Exchange Scheme Using Static Unified Model

3.12 TLS Cooperation Function (TLS 1.2)

The TLS cooperation function (TLS 1.2) APIs conform to RFC 5246.

The provided TLS cooperation functionality supports the TLS 1.2 cooperation function in the form of a TLS 1.2 client function and TLS 1.2 server function.

3.12.1 TLS 1.2 Client Function

No.	API	Description
1	R_TSIP_TlsRegisterCaCertificationPublicKeyIndex	Used in common by all functions.
2	R_TSIP_GenerateTlsRsaPublicKeyIndex R_TSIP_UpdateTlsRsaPublicKeyIndex R_TSIP_TlsRootCertificateVerification R_TSIP_TlsCertificateVerification R_TSIP_TlsCertificateVerificationExtension	Used in common by both TLS 1.2 and TLS 1.3 Client functions.
3	R_TSIP_TlsGenerateServerRandom R_TSIP_TlsEncryptPreMasterSecretWithRsa2048PrivateKey R_TSIP_TlsGenerateMasterSecret R_TSIP_TlsGenerateSessionKey R_TSIP_TlsGenerateVerifyData R_TSIP_TlsServersEphemeralEcdhPublicKeyRetrieves R_TSIP_GenerateTlsP256EccKeyIndex R_TSIP_TlsGeneratePreMasterSecretWithEccP256Key R_TSIP_TlsGenerateExtendedMasterSecret R_TSIP_TlsCertificateVerifyVerification	Used by TLS 1.2 Client function.
4	R_TSIP_Aes128CbcEncryptInit/Update/Final R_TSIP_Aes256CbcEncryptInit/Update/Final R_TSIP_Aes128CbcDecryptInit/Update/Final R_TSIP_Aes256CbcDecryptInit/Update/Final R_TSIP_Aes128GcmEncryptInit/Update/Final R_TSIP_Aes256GcmEncryptInit/Update/Final R_TSIP_Aes128GcmDecryptInit/Update/Final R_TSIP_Aes256GcmDecryptInit/Update/Final R_TSIP_Sha1HmacGenerateInit/Update/Final R_TSIP_Sha1HmacVerifyInit/Update/Final R_TSIP_Sha256HmacGenerateInit/Update/Final R_TSIP_Sha256HmacVerifyInit/Update/Final R_TSIP_EcdsaP256SignatureGenerate R_TSIP_RsassaPkcs2048SignatureGenerate R_TSIP_RsassaPss2048SignatureGenerate	General APIs are used for TLS cooperation function.

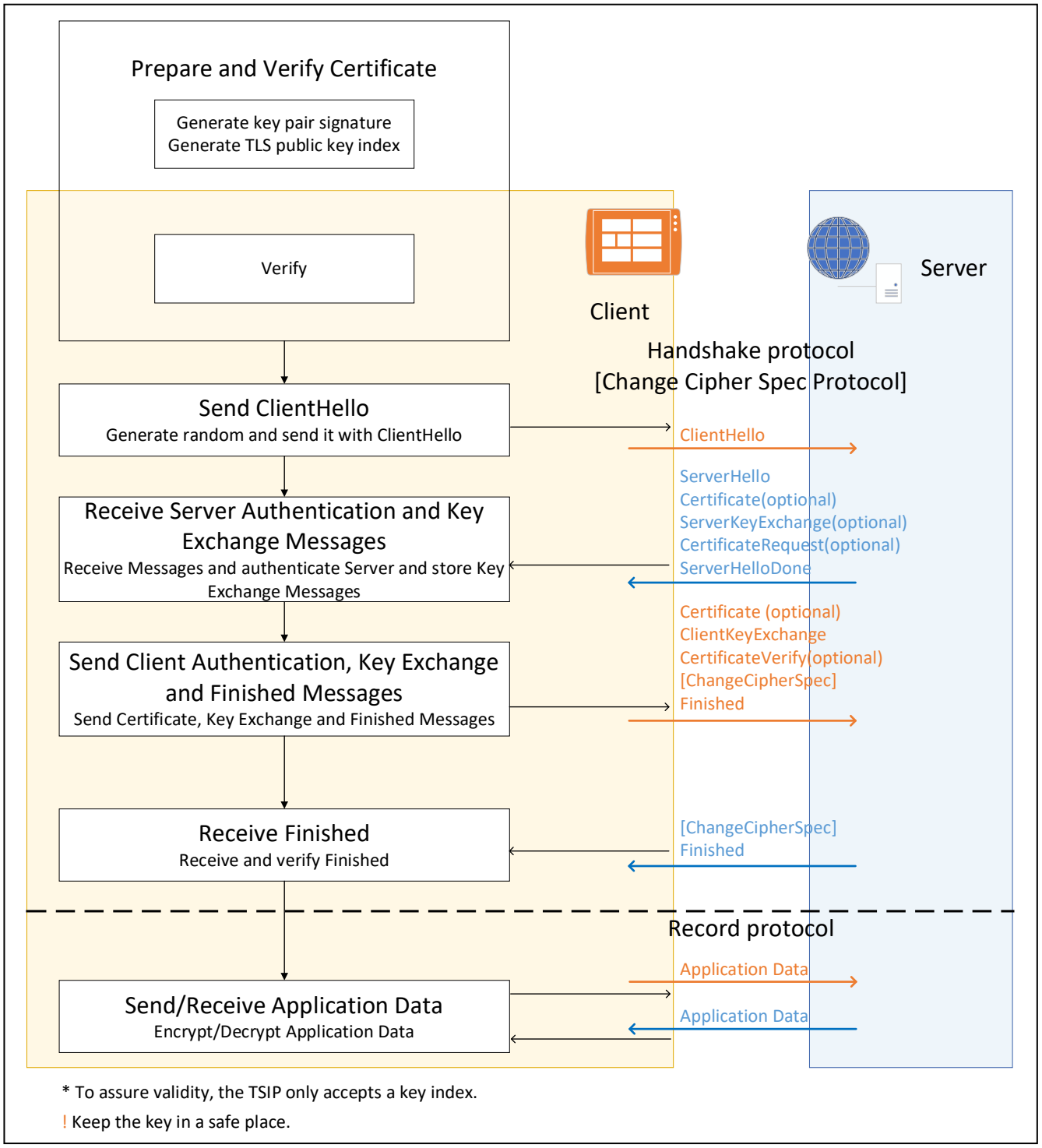


Figure 3.8 Implementation Using TLS 1.2 Client Function

1. Preparing and Verifying Certificates
   1.1 Before using TLS cooperation function, prepare the items below.
   - root CA certificate which the server connects to
   - RSA 2048bit key pair
   - client certificate.

   1.2 Generate signature value of the prepaired root CA certificate with using the private key of the key pair. The signature algorithm is RSA2048 PSS with SHA256. And generate Encrypted Key of the public key of the key pair with using UFPK. Note that the above procedure must be executed in a safe site.
   When there are multiple servers connects to and needed to register multiple root CA certificate, generate signature value of the root CA certificate bundle. It is possible to prevent making connection with an unintentional server with generating signature of the root CA certificate.

   1.3 Use R_TSIP_TlsRsaPublicKeyIndex with the Encrypted Key of the RSA public key as input to generate an RSA public wrapped key. Please refer to 3.7.1, Key Injection and Updating for more detail.

   1.4 Use R_TSIP_TlsRegisterCaCertificationPublicKey() or R_TSIP_Open() to register the RSA wrapped key to the TSIP driver. When R_TSIP_Open() is used to register the wrapped key, call the API after using R_TSIP_Close.

   1.5 Use R_TSIP_TlsRootCertificateVerification() to verify the root CA certificate bundle and generate an encrypted root CA certificate public key.

   1.6 Use R_TSIP_TlsCertificateVerification() or R_TSIP_TlsCertificateVerificationExtension() with the encrypted root CA certificate public key and client certificate public key as input to generate an encrypted client certificate public key.

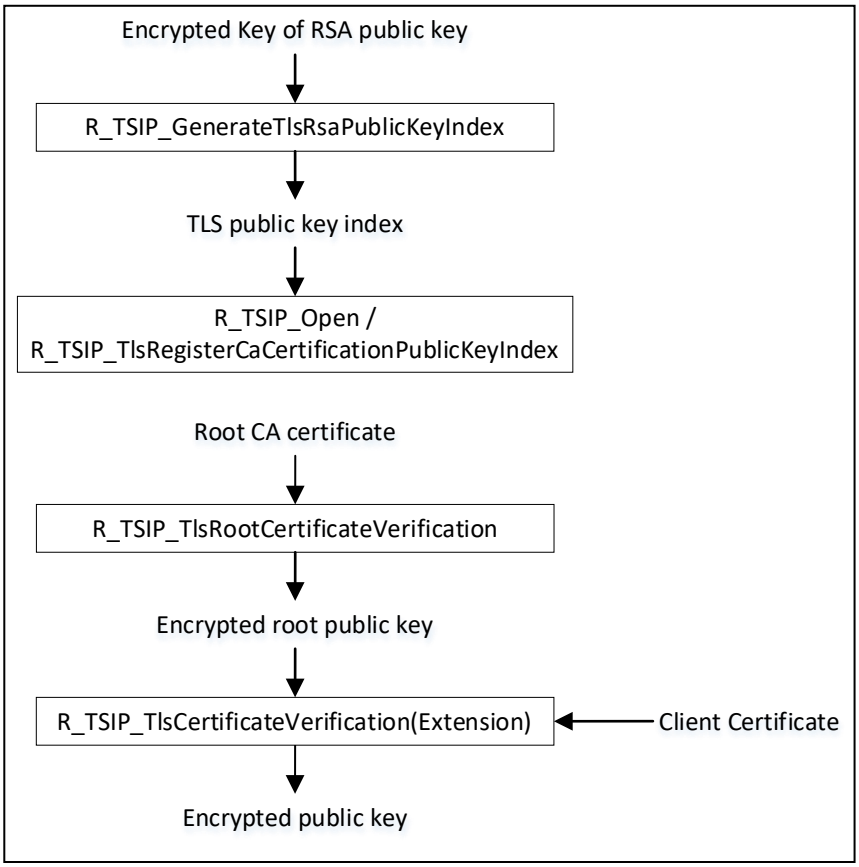


Figure 3.9 Preparing and Verifying Certificates

2. Sending ClientHello
   2.1 Send ClientHello.

3. Receiving Server Authentication and Key Exchange Messages
   3.1 Receive ServerHello and the random field in it.
   3.2 When (Server) certificate is received, use R_TSIP_TlsCertificateVerification() or R_TSIP_TlsCertificateVerificationExtension() to generate an encrypted public key.
   3.3-a When the key exchange method is ECDHE, extract the signature and ECDH public key from ServerKeyExchange and generate pre-master secret.

   3.3-a.1 Use R_TSIP_GenerateTlsP256EccKeyIndex() to generate an ECC key pair.

   3.3-a.2 Use R_TSIP_TlsServersEphemeralEcdhPublicKeyRetrieves() with the ECDH public key, (Server) certificate and handshake messages as input to generate encrypted ephemeral ECDH public key.

   3.3-a.3 Use R_TSIP_TlsGeneratePreMasterSecretWithEccP256Key() with the ephemeral ECDH public key and the wrapped ECC private key generated in 3.3-a.1 as input to generate encrypted ephemeral pre-master secret.

   3.3-b When the key exchange method is RSA, nothing to do here.

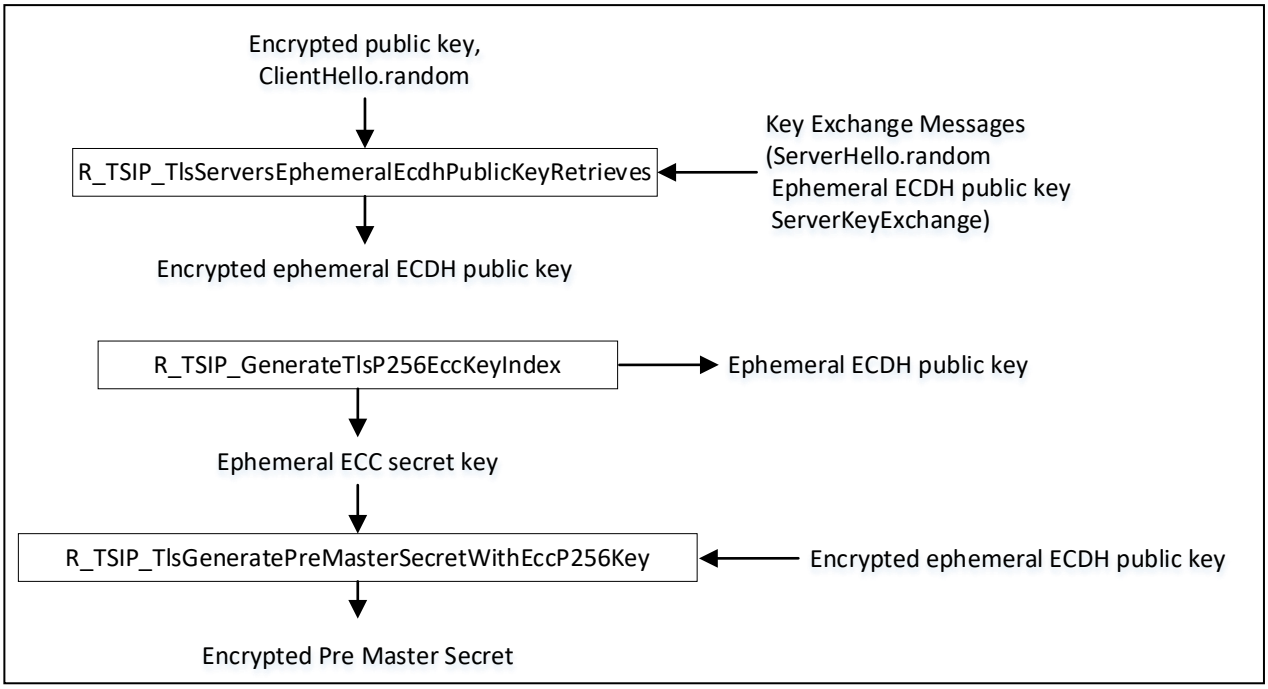


Figure 3.10 Receiving Server Authentication and Key Exchange Messages

4. Sending Client Authentication and Key Exchange Messages
   4.1 When CertificateRequest is received, send client certificate as (Client) Certificate to the server.
   4.2-a When the key exchange method is ECDHE, generate ClientKeyExchange with using ECDH public key.

   4.2-a.1 Use ECDH public key generated in 3.3.1 as ClientECDiffieHelmanPublic field in ClientKeyExchange.

   4.2-b When the key exchange method is RSA, generate ClientKeyExchange with generating pre-master secret.

   4.2-b.1 Use R_TSIP_TlsGeneratePreMasterSecret() to generate an encrypted ephemeral pre-master secret.

4.2-b.2 Use R_TSIP_TlsEncryptPreMasterSecretWithRsa2048PublicKey() with the encrypted ephemeral pre-master secret and (Server) Certificate as input to generate an encrypted ephemeral pre-master secret to use in ClientKeyExchange..

4.3 When CertificateVerify is needed, use R_TSIP_EcdsaP256SignatureGenerate(), R_TSIP_RsassaPkcs2048SignatureGenerate() or R_TSIP_RsassaPss24048SignatureGenerate() to generate it.

4.4 Use R_TSIP_TlsGenerateMasterSecret() with encrypted ephemeral pre-master secret as input to generate encrypted ephemeral master secret..

4.5 When extended master secret is to be generated, use R_TSIP_TlsGenerateExtendedMasterSecret() with encryted ephemeral pre-master secret as input to generate encrypted ephemeral extended master secret.

4.6 Use R_TSIP_TlsGenerateSessionKey() with encrypted ephemeral master secret or encrypted ephemeral extended master secret and each handshake message as input to generate each wrapped key to use in the TLS connection.

4.7 Generate (Client) Finished.

4.7.1 Use R_TSIP_TlsGenerateVerifyData() with encrypted ephemeral master secret or encrypted ephemeral extended master secret and hash value of the handshake messages as inout to generate (Client) VerifyData to use in (Client) Finished.

4.7.2 When AES CBC mode is used as the cipher suite, use APIs of AES CBC encryption and HMAC generation with wrapped AES key and wrapped MAC key for Client -> Server communication generated in 4.6 and (Client) VerifyData generated in 4.7.1 as input to generated (Client) Finished.

4.7.3 When AES GCM mode is used as the cipher suite, use APIs of AES GCM encryption with wrapped AES key for Client -> Server communication generated in 4.6 and (Client) VerifyData generated in 4.7.1 as input to generated (Client) Finished.

4.8 Send each authentication and key exchange messages.

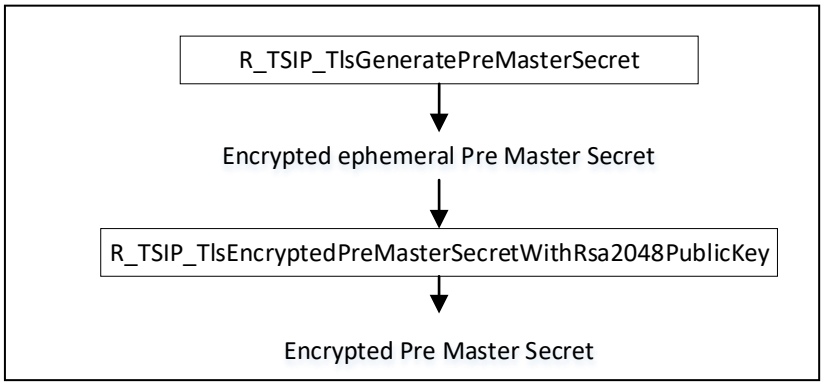


Figure 3.11 Sending Client Authentication and Key Exchange Messages (Generating pre-master secret for RSA)

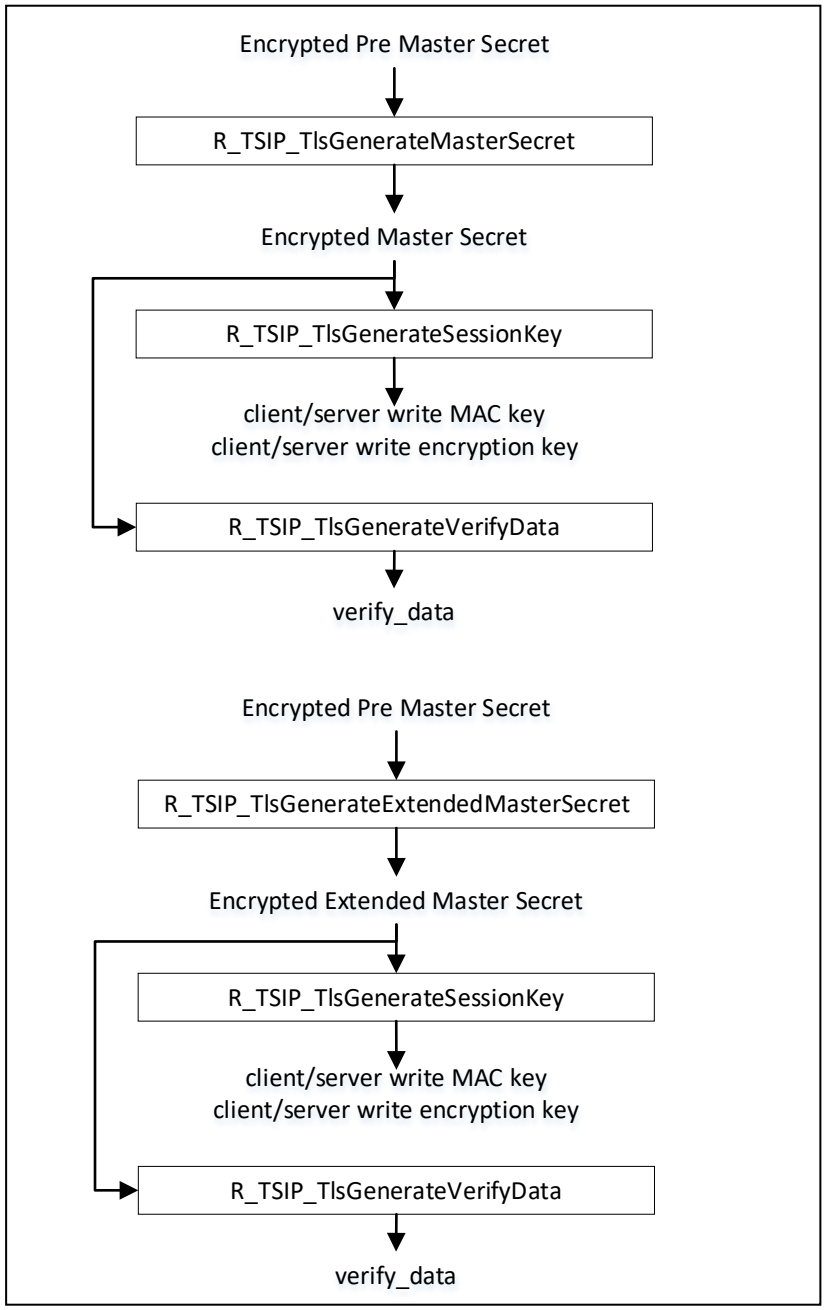


Figure 3.12 Sending Client Authentication and Key Exchange Messages (Generating VerifyData)

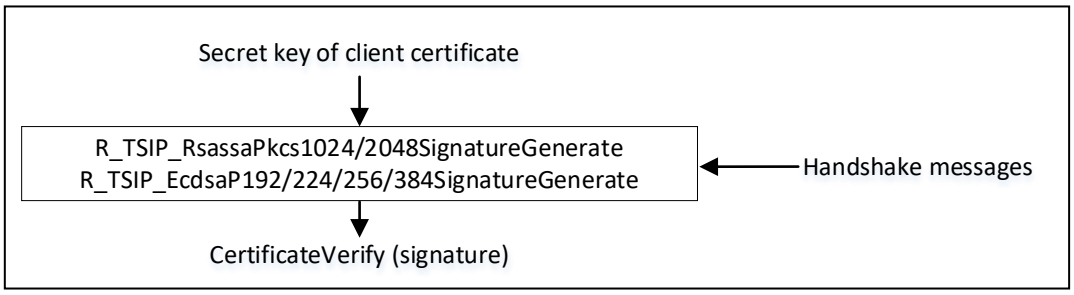


Figure 3.13 Sending Client Authentication and Key Exchange Messages (Generating CertificateVerify)

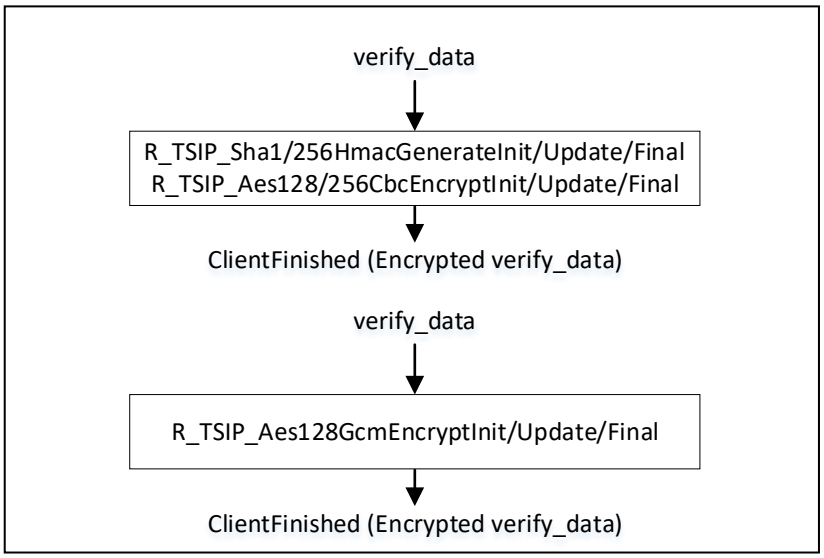


Figure 3.14 Sending Client Authentication and Key Exchange Messages (Generating ClientFinished)

5. Receiving Finished
   5.1 Use R_TSIP_TlsGenerateVerifyData() with encrypted ephemeral master secret or encrypted ephemeral extended master secret and hash value of the handshake messages as input to generate (Server) VerifyData.
   5.2 Verify (Server) Finished.

   5.2.1-a When AES CBC mode is used as the cipher suite, use APIs of AES CBC encryption and HMAC generation with wrapped AES key and wrapped MAC key for Server -> Client communication generated in 4.6 and (Server) Finished as input to decrypt VerifyData.

   5.2.1-b When AES GCM mode is used as the cipher suite, use APIs of AES GCM encryption with wrapped AES key for Server -> Client communication generated in 4.6 and (Server) Finished as input to decrypt VerifyData.

   5.2.2 Check the equality between the (Server) VerifyData generated in 5.1 and the VeriftData decrypted in 5.2.1-a or 5.2.1-b.

6. Sending/Receiving Application Data
   6.1-a When AES CBC mode is used as the cipher suite, use APIs of AES CBC encryption/decryption and HMAC generation/verification with wrapped AES key and wrapped MAC key generated in 4.6 to encrypt/decrypt Application Data.

   6.1-b When AES GCM mode is used as the cipher suite, use APIs of AES GCM encryption/decryption with wrapped AES key generated in 4.6 to encrypt/decrypt Application Data.

3.12.2 TLS 1.2 Server Function

No.	API	Description
1	R_TSIP_TlsRegisterCaCertificationPublicKeyIndex	Used in common by all functions.
2	R_TSIP_GenerateTlsSVRsaPublicKeyIndex R_TSIP_UpdateTlsSVRsaPublicKeyIndex R_TSIP_TlsSVRootCertificateVerification R_TSIP_TlsSVCertificateVerification R_TSIP_TlsSVCertificateVerificationExtension	Used in common by both TLS 1.2 and TLS 1.3 Server functions.
3	R_TSIP_TlsSVGenerateServerRandom R_TSIP_TlsSVDecryptPreMasterSecretWithRsa2048PrivateKey R_TSIP_TlsSVGenerateMasterSecret R_TSIP_TlsSVGenerateSessionKey R_TSIP_TlsSVGenerateVerifyData R_TSIP_GenerateTlsSVP256EccKeyIndex R_TSIP_TlsSVGeneratePreMasterSecretWithEccP256Key R_TSIP_TlsSVGenerateExtendedMasterSecret R_TSIP_TlsSVCertificateVerifyVerification	Used by TLS 1.2 Server function.
4	R_TSIP_Aes128CbcEncryptInit/Update/Final R_TSIP_Aes256CbcEncryptInit/Update/Final R_TSIP_Aes128CbcDecryptInit/Update/Final R_TSIP_Aes256CbcDecryptInit/Update/Final R_TSIP_Aes128GcmEncryptInit/Update/Final R_TSIP_Aes256GcmEncryptInit/Update/Final R_TSIP_Aes128GcmDecryptInit/Update/Final R_TSIP_Aes256GcmDecryptInit/Update/Final R_TSIP_Sha1HmacGenerateInit/Update/Final R_TSIP_Sha1HmacVerifyInit/Update/Final R_TSIP_Sha256HmacGenerateInit/Update/Final R_TSIP_Sha256HmacVerifyInit/Update/Final	General APIs are used for TLS cooperation function.

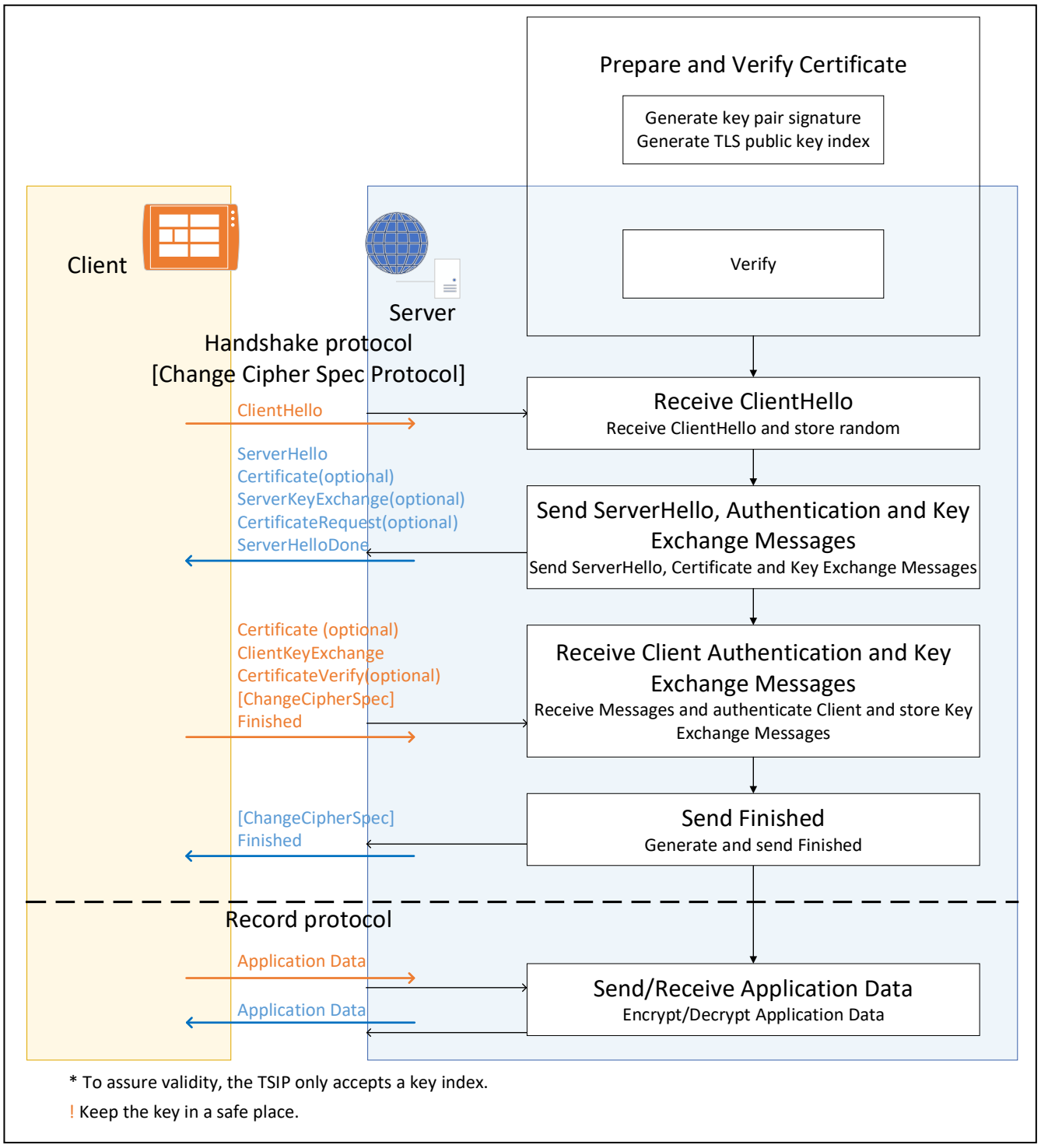


Figure 3.15 Implementation Using TLS 1.2 Server Function

1. Preparing and Verifying Certificates
   1.1 Before using TLS cooperation function, prepare the items below.
   - root CA certificate which the server connects to
   - RSA 2048bit key pair
   - server certificate.

   1.2 Generate signature value of the prepaired root CA certificate with using the private key of the key pair. The signature algorithm is RSA2048 PSS with SHA256. And generate Encrypted Key of the public key of the key pair with using UFPK. Note that the above procedure must be executed in a safe site.
   When multiple root CA certificate are needed to register, generate signature value of the root CA certificate bundle. It is possible to prevent making connection with using an unauthorized root CA certificate.

   1.3 Use R_TSIP_TlsSVRsaPublicKeyIndex with the Encrypted Key of the RSA public key as input to generate an RSA public wrapped key. Please refer to 3.7.1, Key Injection and Updating for more detail.

   1.4 Use R_TSIP_TlsRegisterCaCertificationPublicKey() to register the RSA wrapped key to the TSIP driver.

   1.5 Use R_TSIP_TlsSVRootCertificateVerification() to verify the root CA certificate bundle and generate an encrypted root CA certificate public key.

   1.6 Use R_TSIP_TlsSVCertificateVerification() or R_TSIP_TlsSVCertificateVerificationExtension() with the encrypted root CA certificate public key and server certificate public key as input to generate an encrypted server certificate public key.

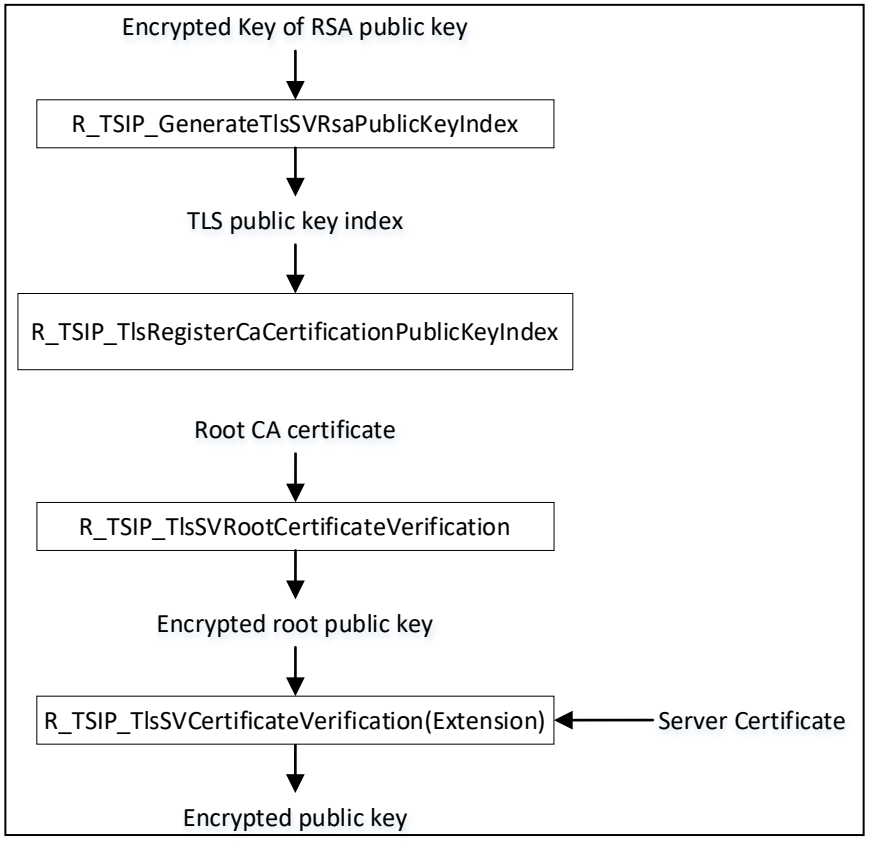


Figure 3.16 Preparing and Verifyng Certificates

2. Receiving ClientHello
   2.1 Receive ClientHello and the random field in it.

3. Sending ServerHello, Authentication and Key Exchange Messages
   3.1 Use R_TSIP_TlsSVGenerateServerRandom() to generate a random number value to use in ServerHello.
   3.2-a When the key exchange method is ECDHE, generate ServerKeyExchange.

   3.2-a.1 Use R_TSIP_GenerateTlsSVP256EccKeyIndex() to generate an ECC key pair. The ECDH public key of th ekey pair is used as public field of params in ServerKeyExchange.

   3.2-a.2 Use R_TSIP_EcdsaP256SignatureGenerate() with the wrapped private key of (server) certificate and hash value of the handshake messages as input to generate the signature. The signature is used as signed_params field in ServerKeyExchange.

   3.2-b When the key exchange method is RSA, nothing to do here.

   3.3 Send SerherHello, authentication and key exchange method is RSA, nothing to do here.

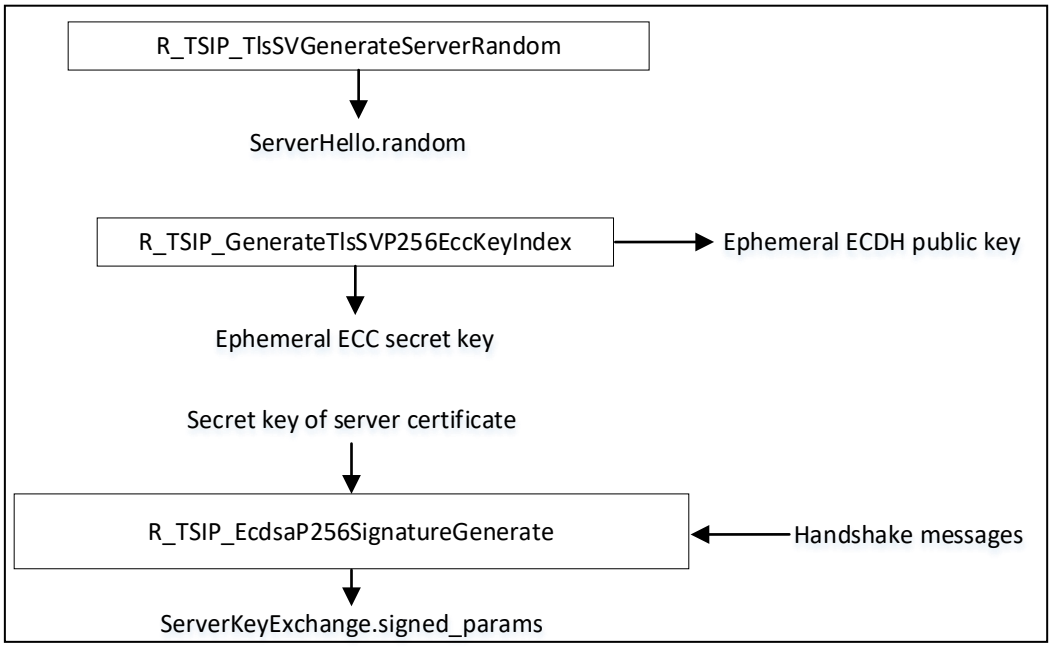


Figure 3.17 Sending ServerHello, Authentication and Key Exchange Messages

4. Receiving Client Authentication and Key Exchange Messages
   4.1 When (Client) Certificate is received, use R_TSIP_TlsSVCertificateVerification() or R_TSIP_TlsSVCertificateVerificationExtension() to generate an encrypted public key.
   4.2-a When the key exchange method is ECDHE, extract ECDH public key from ClientKeyExchange and generate pre-master secret.

   4.2-a.1 Use R_TSIP_TlsSVGeneratePreMasterSecretWithEccP256Key() with the ECDH public key in ClientKeyExchange and wrapped ECC private key generated in 3.3.1 as input to generate encrypted ephemeral pre-master secret.

   4.2-b When the key exchange method is RSA, extract pre-master secret from ClientKeyExchange.

   4.2-b.1 Use R_TSIP_TlsSVDecryptPreMasterSecretWithRsa2048PrivateKey() with encrypted pre-master secret in ClientKeyExchange and wrapped RSA private key of (Server) Certificate as input to generate the encrypted ephemeral pre-master secret.

4.3 Use R_TSIP_TlsSVGenerateMasterSecret() with encrypted ephemeral pre-master secret in ClientKeyExchange and wrapped RSA private key of (Server) Certificate as input to generate the encrypted ephemeral master secret.

4.4 When extended master secret is to be generated, use R_TSIP_TlsSVGenerateExtendedMasterSecret() with encryted ephemeral pre-master secret as input to generate encrypted ephemeral extended master secret.

4.5 When CertificateVerify is received, use R_TSIP_TlsSVCertificateVerifyVerification() to verify it.

4.6 Use R_TSIP_TlsSVGenerateSessionKey() with encrypted ephemeral master secret or encrypted ephemeral extended master secret and each handshake message as input to generate each wrapped key to use in the TLS connection.

4.7 Verify (Client) Finished.

4.7.1 Use R_TSIP_TlsSVGenerateVerifyData() with encrypted ephemeral master secret or encrypted ephemeral extended master secret and hash value of the handshake messages as inout to generate (Client) VerifyData to compare to (Client) Finished.

4.7.2-a When AES CBC mode is used as the cipher suite, use APIs of AES CBC decryption and HMAC verification with wrapped AES key and wrapped MAC key for Client -> Server communication generated in 4.6 and (Client) Finished as input to decrypt (Client) VerifyData.

4.7.2-b When AES GCM mode is used as the cipher suite, use APIs of AES GCM decryption with wrapped AES key for Client -> Server communication generated in 4.6 and (Client) Finished as input to decrypt (Client) VerifyData.

4.7.3 Check the equality between the (Client) VerifyData generated in 4.7.1 and the VeriftData decrypted in 4.7.2-a or 4.7.2-b.

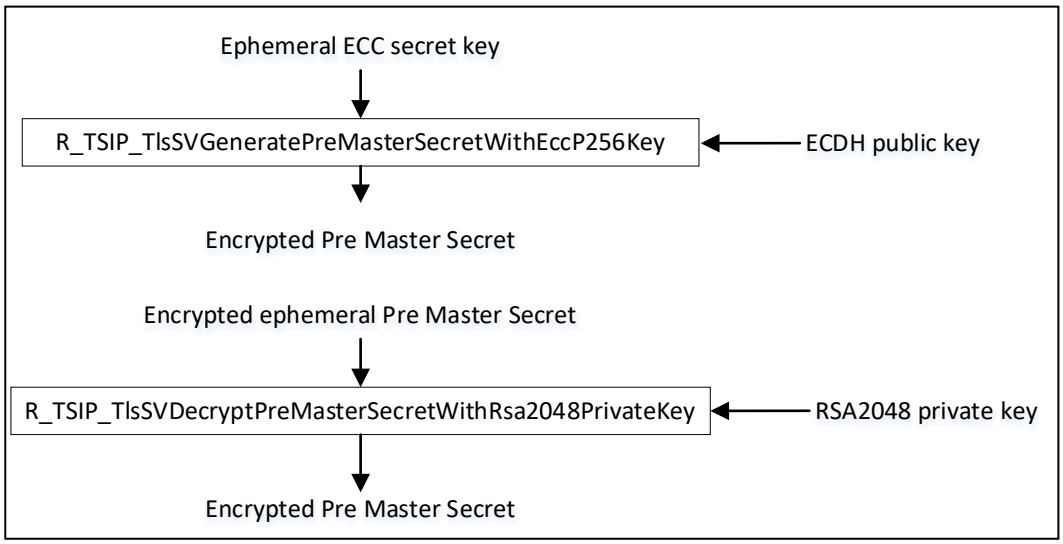


Figure 3.18 Receiving Client Authentication and Key Exchange Messages (Generating Pre Master Secret)

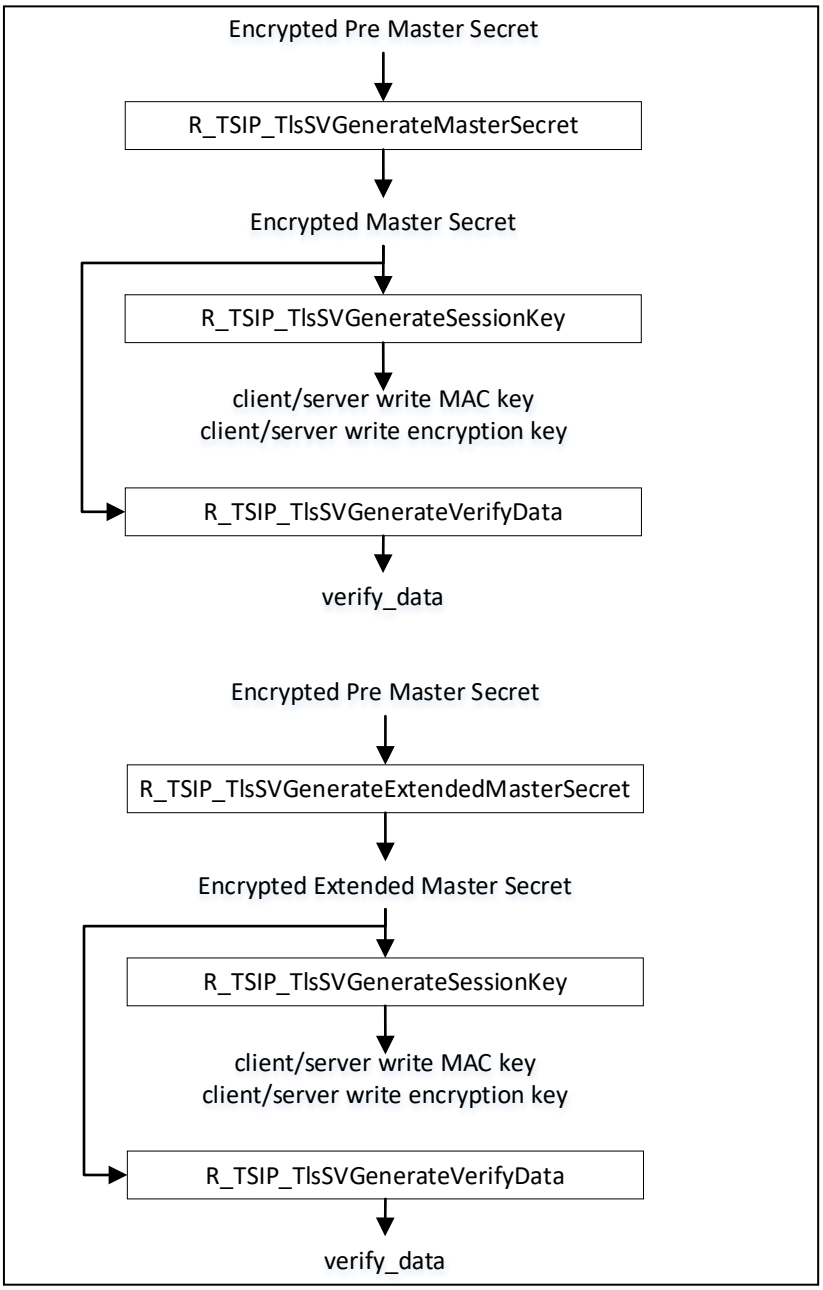


Figure 3.19 Receiving Client Authentication and Key Exchange Messages (Generating VerifyData)

5. Sending Finished
   5.1 Use R_TSIP_TlsSVGenerateVerifyData() with encrypted ephemeral master secret or encrypted ephemeral extended master secret and hash value of the handshake messages as input to generate (Server) VerifyData.
   5.2 Encrypt (Server) Finished.

   5.2-a When AES CBC mode is used as the cipher suite, use APIs of AES CBC encryption and HMAC generation with wrapped AES key and wrapped MAC key for Server -> Client communication generated in 4.6 and (Server) VerifyData as input to generate (Server) Finished.

   5.2-b When AES GCM mode is used as the cipher suite, use APIs of AES GCM encryption with wrapped AES key for Server -> Client communication generated in 4.6 and (Server) VerifyData as input to generate (Server) Finished.

   5.3 Send (Server) Finished.

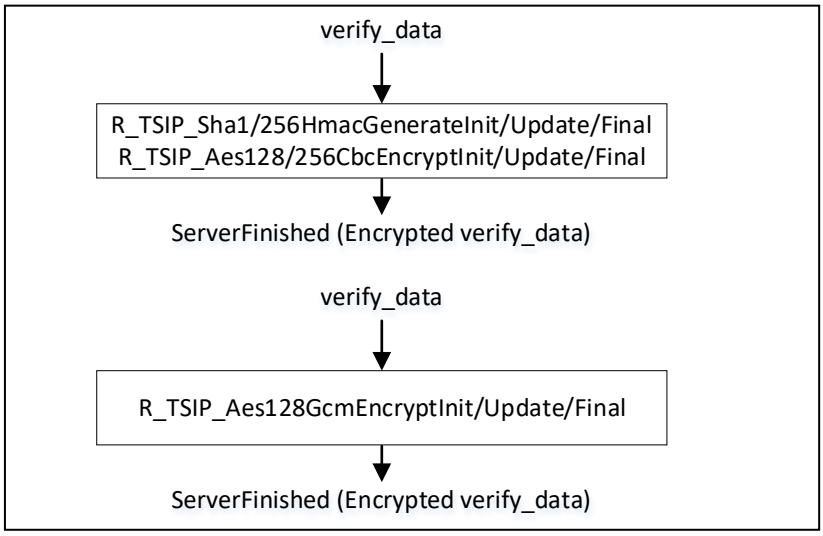


Figure 3.20 Sending Finished

6. Sending/Receiving Application Data
   6.1-a When AES CBC mode is used as the cipher suite, use APIs of AES CBC encryption/decryption and HMAC generation/verification with wrapped AES key and wrapped MAC key generated in 4.6 to encrypt/decrypt Application Data.

   6.1-b When AES GCM mode is used as the cipher suite, use APIs of AES GCM encryption/decryption with wrapped AES key generated in 4.6 to encrypt/decrypt Application Data.

3.13 TLS Cooperation Function (TLS 1.3)

The TLS cooperation function (TLS 1.3) APIs conform to RFC 8446.

The provided TLS cooperation functionality supports the TLS 1.3 cooperation function in the form of a TLS 1.3 client function and a TLS 1.3 server function. For details of these functions, refer to the TSIP driver usage instructions in 3.13.1.1, Full Handshake, 3.13.1.2, Resumption, and 3.13.1.3, 0-RTT.

3.13.1 TLS 1.3 Client Function

No.	API	Description
1	R_TSIP_GenerateTlsRsaPublicKeyIndex R_TSIP_UpdateTlsRsaPublicKeyIndex R_TSIP_TlsRootCertificateVerification R_TSIP_TlsCertificateVerification R_TSIP_TlsCertificateVerificationExtension R_TSIP_Tls13EncryptInit R_TSIP_Tls13EncryptUpdate R_TSIP_Tls13EncryptFinal R_TSIP_Tls13DecryptInit R_TSIP_Tls13DecryptUpdate R_TSIP_Tls13DecryptFinal R_TSIP_Tls13CertificateVerifyGenerate R_TSIP_Tls13CertificateVerifyVerification	Used in common by all functions.
2	R_TSIP_GenerateTls13P256EccKeyIndex R_TSIP_Tls13GenerateEcdheSharedSecret R_TSIP_Tls13GenerateHandshakeSecret R_TSIP_Tls13GenerateServerHandshakeTrafficKey R_TSIP_Tls13GenerateClientHandshakeTrafficKey R_TSIP_Tls13ServerHandshakeVerification R_TSIP_Tls13GenerateMasterSecret R_TSIP_Tls13GenerateApplicationTrafficKey R_TSIP_Tls13UpdateApplicationTrafficKey	Mainly used by full handshake.
3	R_TSIP_Tls13GenerateResumptionMasterSecret R_TSIP_Tls13GeneratePreSharedKey R_TSIP_Tls13GeneratePskBinderKey R_TSIP_Tls13GenerateResumptionHandshakeSecret	Mainly used by resumption.
4	R_TSIP_Tls13Generate0RttApplicationWriteKey	Used by 0-RTT.

3.13.1.1 Full Handshake

Figure 3.8 presents an overview of full handshake implementation using the TLS 1.3 client function.

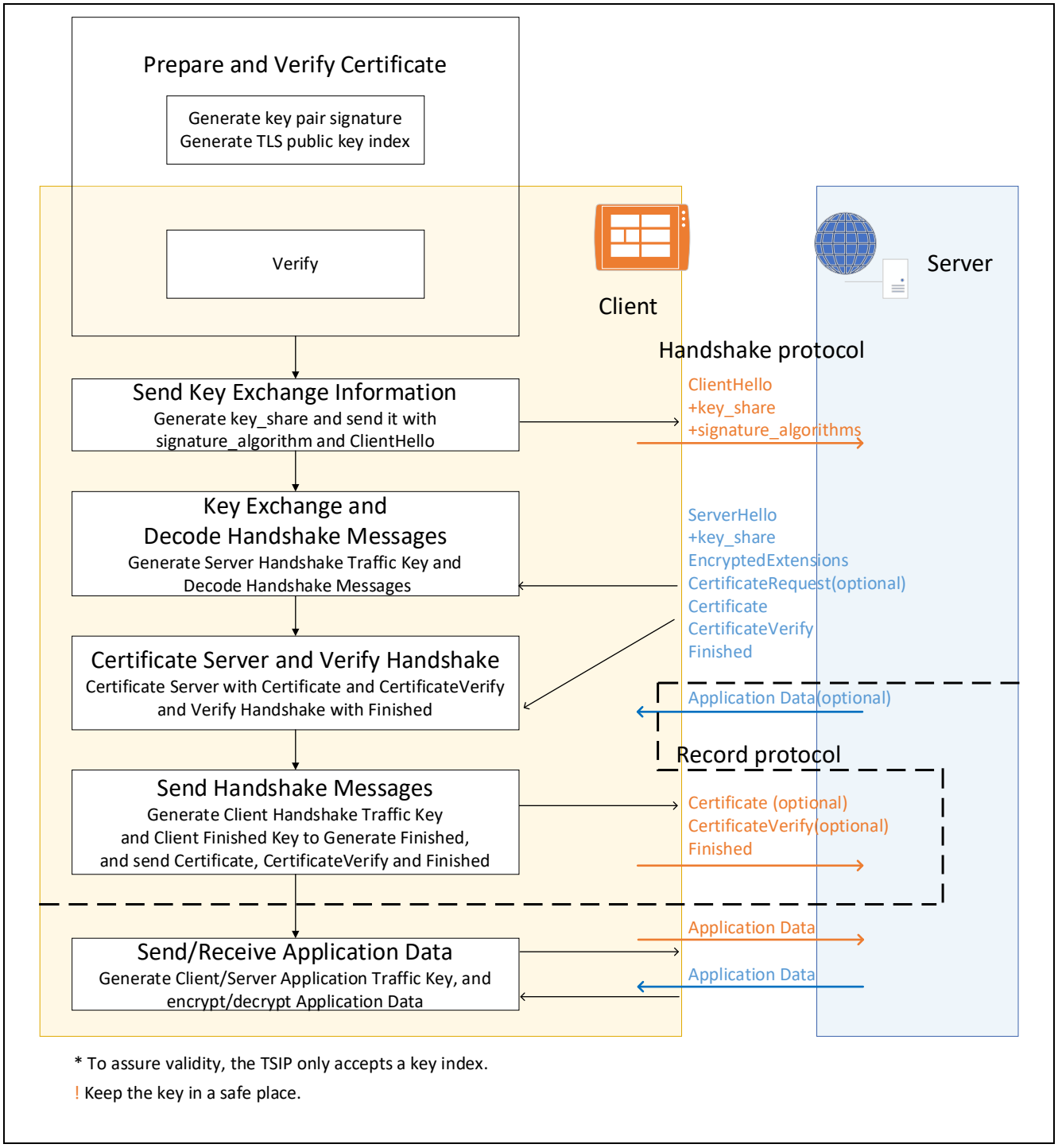


Figure 3.21 Full Handshake Implementation Using TLS 1.3 Client Function

1. Preparing and Verifying Certificates
   1.1 Before using TLS cooperation function, prepare the items below.
   - root CA certificate which the server connects to
   - RSA 2048bit key pair
   - client certificate.

   1.2 Generate signature value of the prepaired root CA certificate with using the private key of the key pair. The signature algorithm is RSA2048 PSS with SHA256. And generate Encrypted Key of the

public key of the key pair with using UFPK. Note that the above procedure must be executed in a safe site.
When there are multiple servers connects to and needed to register multiple root CA certificate, generate signature value of the root CA certificate bundle. It is possible to prevent making connection with an unintentional server with generating signature of the root CA certificate.

1.3 Use R_TSIP_TlsRsaPublicKeyIndex with the Encrypted Key of the RSA public key as input to generate an RSA public wrapped key. Please refer to 3.7.1, Key Injection and Updating for more detail.

1.4 Use R_TSIP_TlsRegisterCaCertificationPublicKey() or R_TSIP_Open() to register the RSA wrapped key to the TSIP driver. When R_TSIP_Open() is used to register the wrapped key, call the API after using R_TSIP_Close.

1.5 Use R_TSIP_TlsRootCertificateVerification() to verify the root CA certificate bundle and generate an encrypted root CA certificate public key.

1.6 Use R_TSIP_TlsCertificateVerification() or R_TSIP_TlsCertificateVerificationExtension() with the encrypted root CA certificate public key and client certificate public key as input to generate an encrypted client certificate public key.

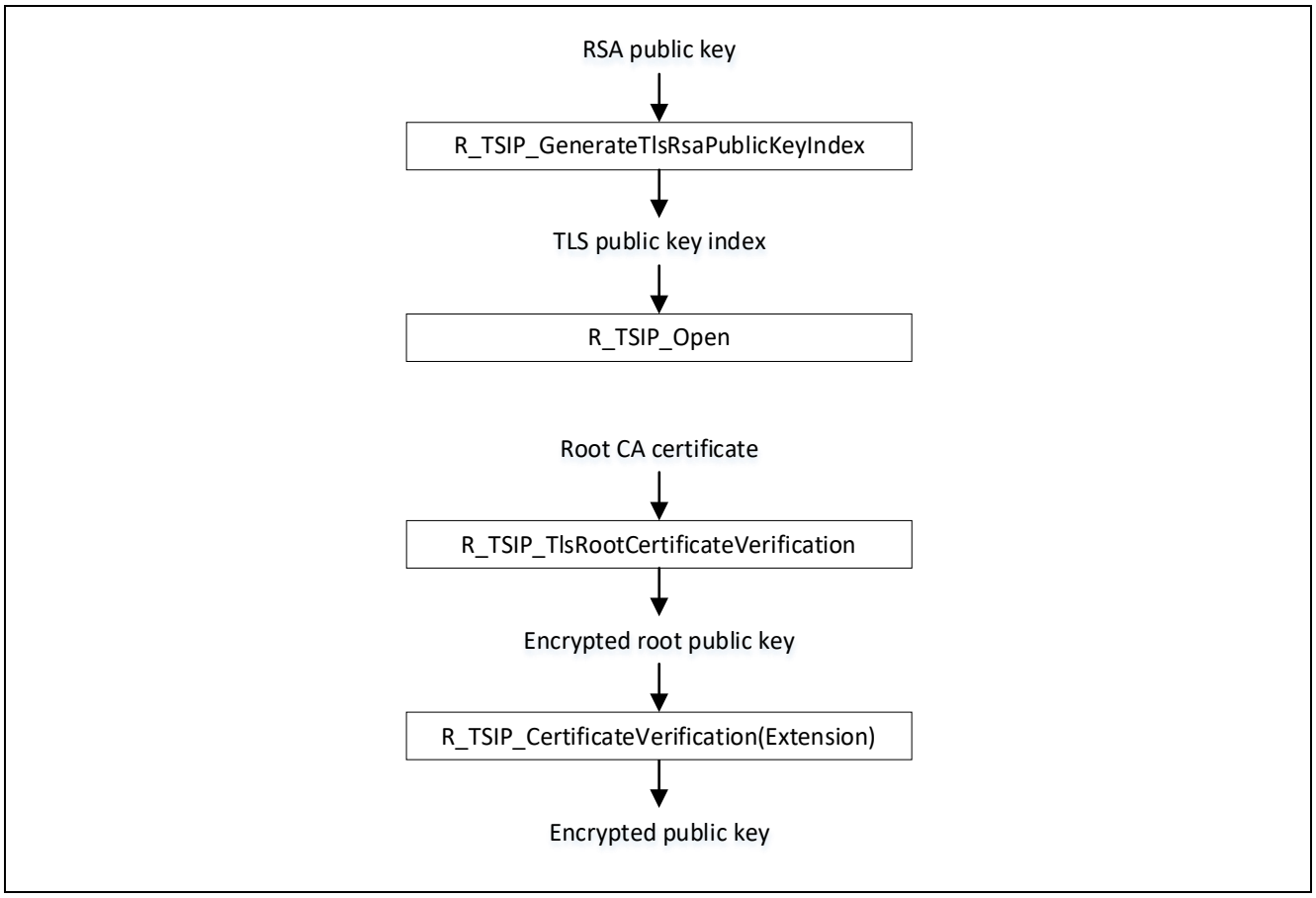


Figure 3.22 Preparing and Verifying Certificates

2. Sending Key Exchange Information

2.1 Use R_TSIP_GenerateTls13P256KeyIndex() to generate an ephemeral ECDH public key.

2.2 When ClientHello is sent, ECDH public key is sent to the server as the key_share field, together with the signature_algorithm field.

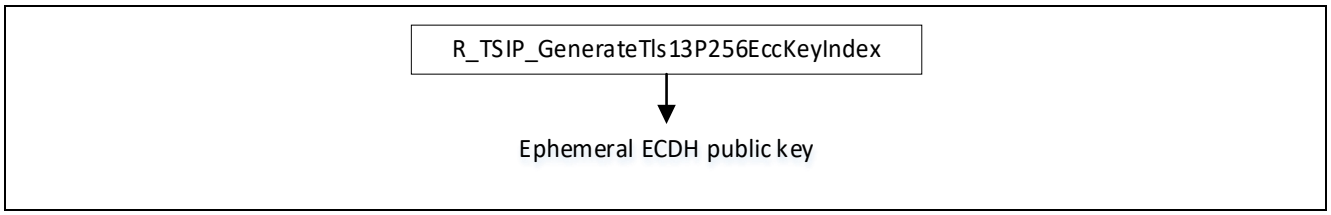


Figure 3.23 Sending Key Exchange Information

3. Exchanging Keys and Decoding Handshake Messages
   3.1 Use R_TSIP_Tls13GenerateEcdheSharedSecret() with the public key received from the server as input to generate a shared secret wrapped key.
   3.2 Use R_TSIP_Tls13GenerateHandshakeSecret() with the shared secret wrapped key as input to generate a handshake secret wrapped key.
   3.3 Use R_TSIP_Tls13GenerateServerHandshakeTrafficKey() with the handshake secret wrapped key as input to generate a server write key index and server finished wrapped key for use by the handshake protocol.
   3.4 Use R_TSIP_Tls13DecryptInit/Update/Final() with the server write wrapped key as input to decrypt encrypted handshake messages received from the server.

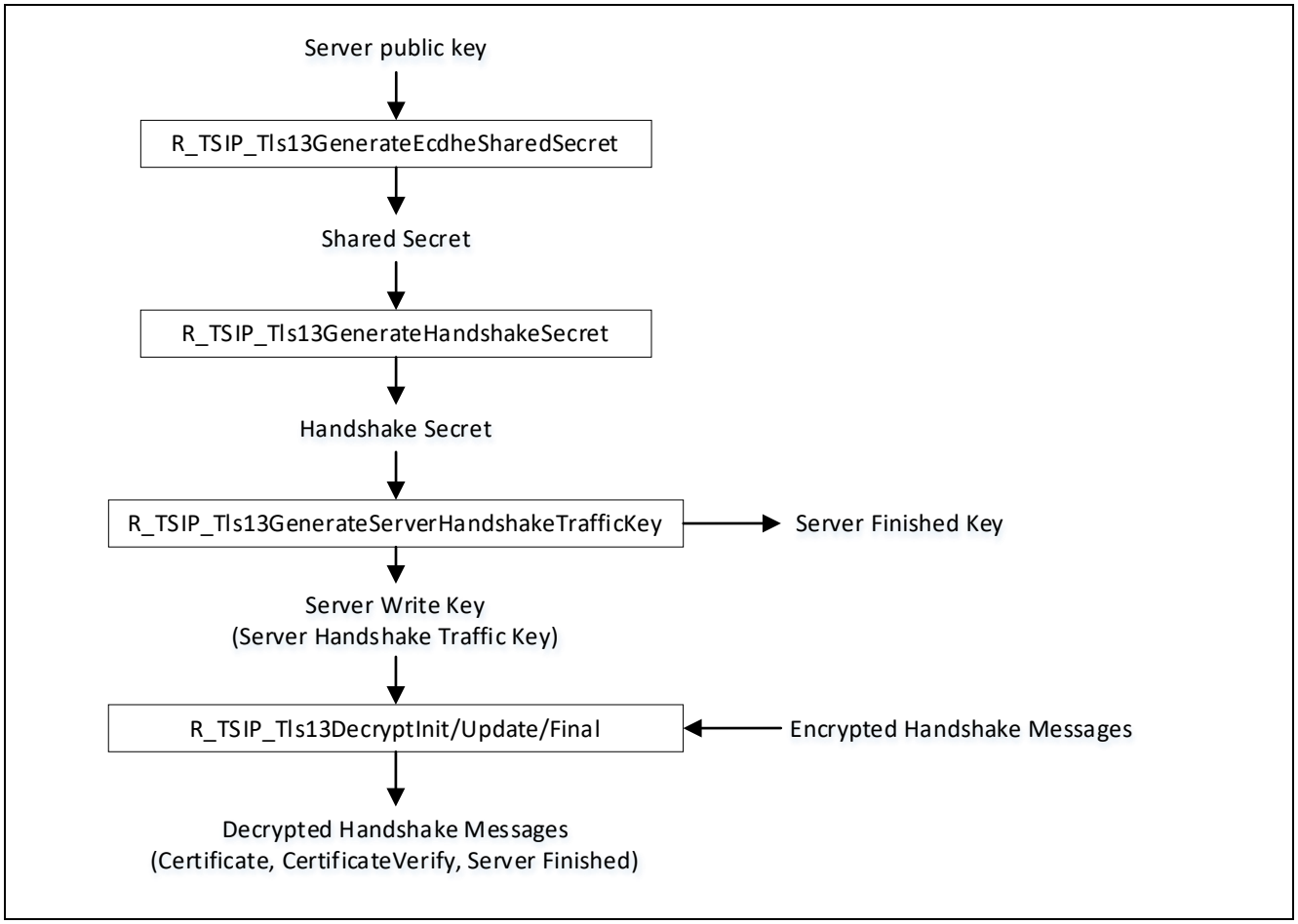


Figure 3.24 Exchanging Keys and Decoding Handshake Messages

4. Verifying Server and Handshake
   4.1 Use R_TSIP_TlsCertificateVerification() with the (Server) Certificate field from the handshake message as input to verify the signature of the signature.
   4.2 Use R_TSIP_Tls13CertificateVerifyVerification() with the (Server) CertificateVerify field from the handshake message as input to verify the (Server) CertificateVerify field.
   4.3 Use R_TSIP_Tls13ServerHandshakeVerification() with the (Server) Finished field from the handshake message and the server finished wrapped key as input to preform handshake verification. On the TSIP, the handshake verification result is output as verify_data_index.

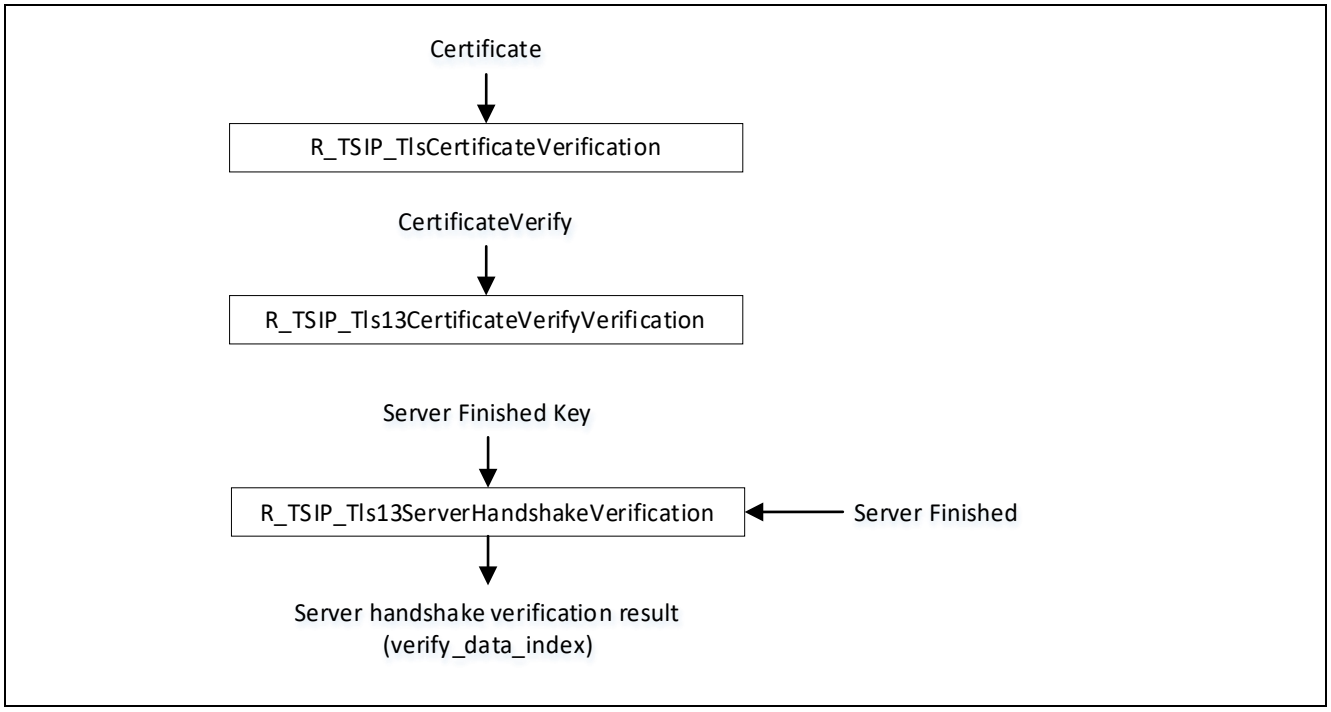


Figure 3.25 Verifying Server and Handshake

5. Sending Handshake Messages
   5.1 The (Client) Certificate field and (Client) CertificateVerify field are generated when CertificateRequest is received from the server. To generate a signature for the (Client) CertificateVerify field, use R_TSIP_Tls13CertificateVerifyGenerate() with the secret wrapped key of the certificate as input.
   5.2 Use R_TSIP_Tls13GenerateClientHandshakeTrafficKey() with the wrapped key of the handshake secret as input to generate the client write wrapped key and client finished key index used in the handshake phase.
   5.3 Use R_TSIP_Sha256HmacGenerateIniti/Update/Final() with the client finished wrapped key as input to create the (Client) Finished field.
   5.4 Use R_TSIP_Tls13EncryptInit/Update/Final() with the client write wrapped key as input to encrypt the handshake messages sent to the server.
   5.5 Send the encrypted handshake messages.

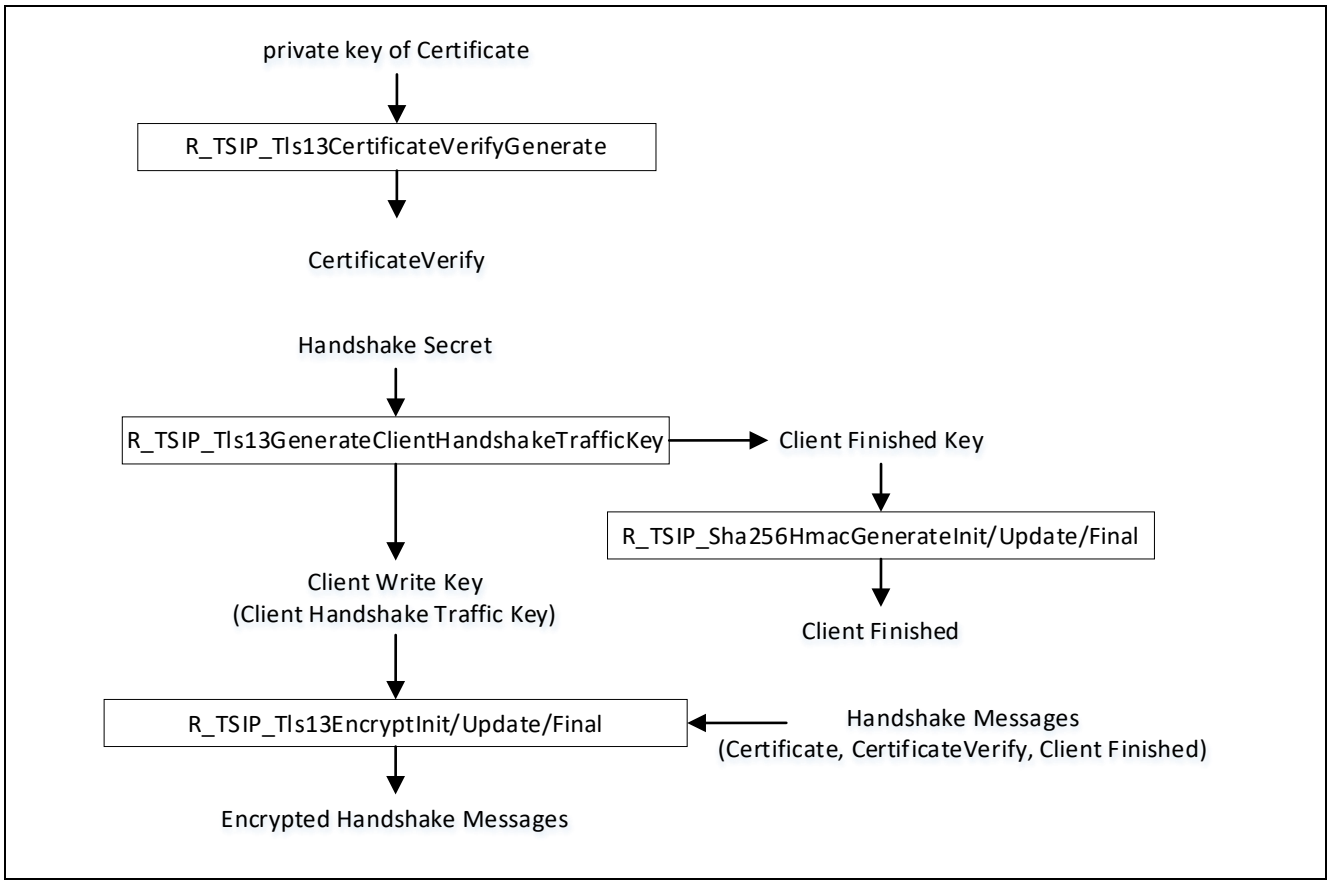


Figure 3.26 Sending Handshake Messages

6. Sending/Receiving Application Data
   6.1 Use R_TSIP_Tls13GenerateMasterSecret() with the handshake secret wrapped key and verify_data_index as input to generate the master secret wrapped key.
   6.2 Use R_TSIP_Tls13GenerateApplicationTrafficKey() with the master secret wrapped key as input to generate the server write wrapped key, client write wrapped key, and application secret wrapped key used by the record protocol.
   6.3 To update the server write key or client write key, use R_TSIP_Tls13UpdateApplicationTrafficKey() with the application secret wrapped key as input.
   6.4 To encrypt data to be sent to the server, use R_TSIP_Tls13EncryptInit/Update/Final() with the client write wrapped key as input.
   6.5 To decrypt data received from the server, use R_TSIP_Tls13DecryptInit/Update/Final() with the server write wrapped key as input.

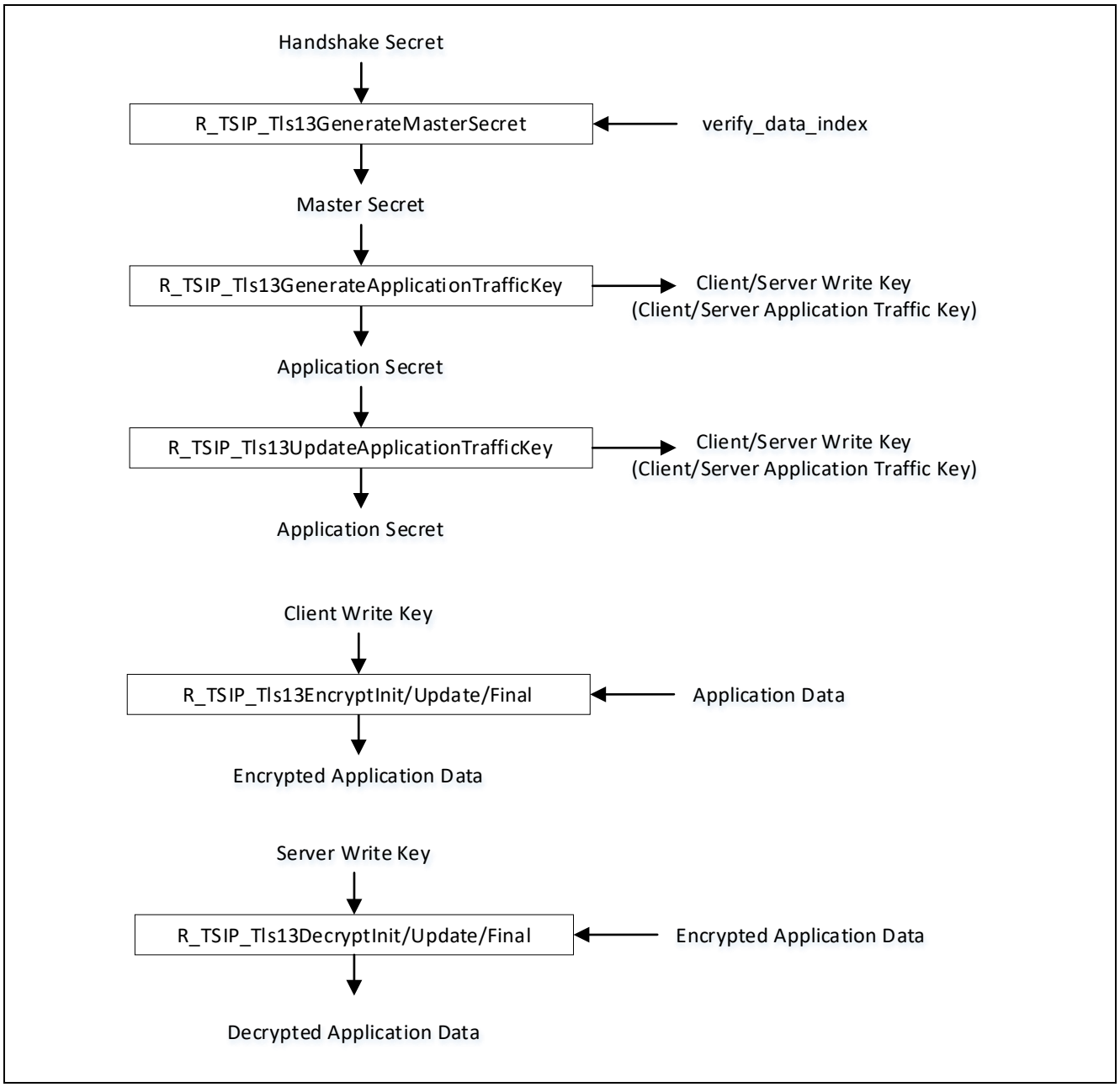


Figure 3.27 Sending/Receiving Application Data

3.13.1.2 Resumption

Figure 3.15 presents an overview of resumption implementation using the TLS 1.3 client function.

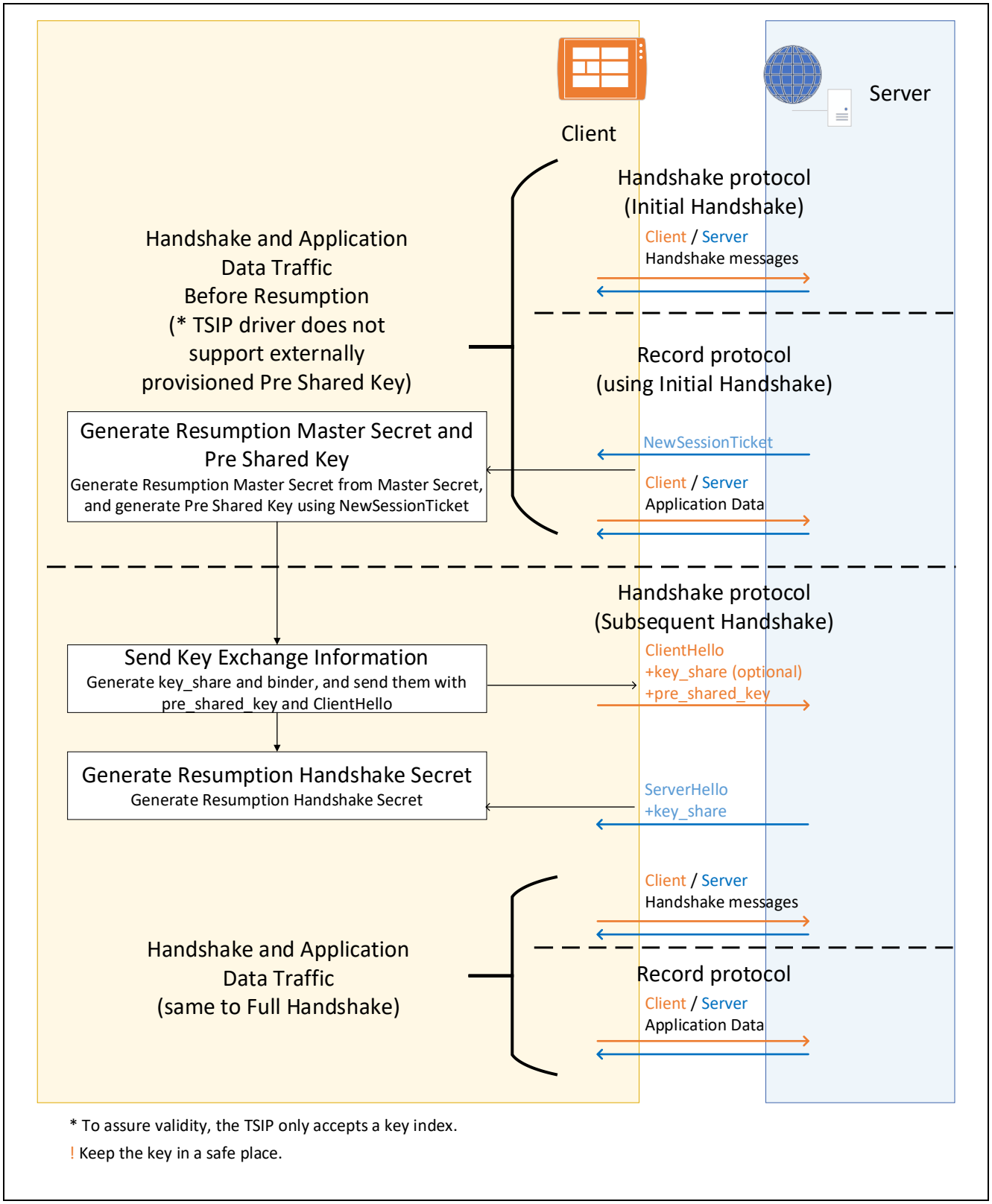


Figure 3.28 Resumption Implementation Using TLS 1.3 Client Function

1. Handshake and Application Data Communication before Resumption
   1.1 TLS 1.3 communication is established by performing a full handshake.
   1.2 A new session ticket is received from the server as application data.

Note: The TSIP driver does not allow use of other than the pre shared key generated after handshaking completes.

2. Generating Resumption Master Secret and Pre Shared Key
   2.1 Use R_TSIP_Tls13GenerateResumptionMasterSecret() with the master secret wrapped key used for the full handshake as input to generate the resumption master secret wrapped key.
   2.2 Use R_TSIP_Tls13GeneratePreSharedKey() with the resumption master secret wrapped key and the ticket nonce within the new session ticket received from the server as input to generate the pre shared wrapped key.

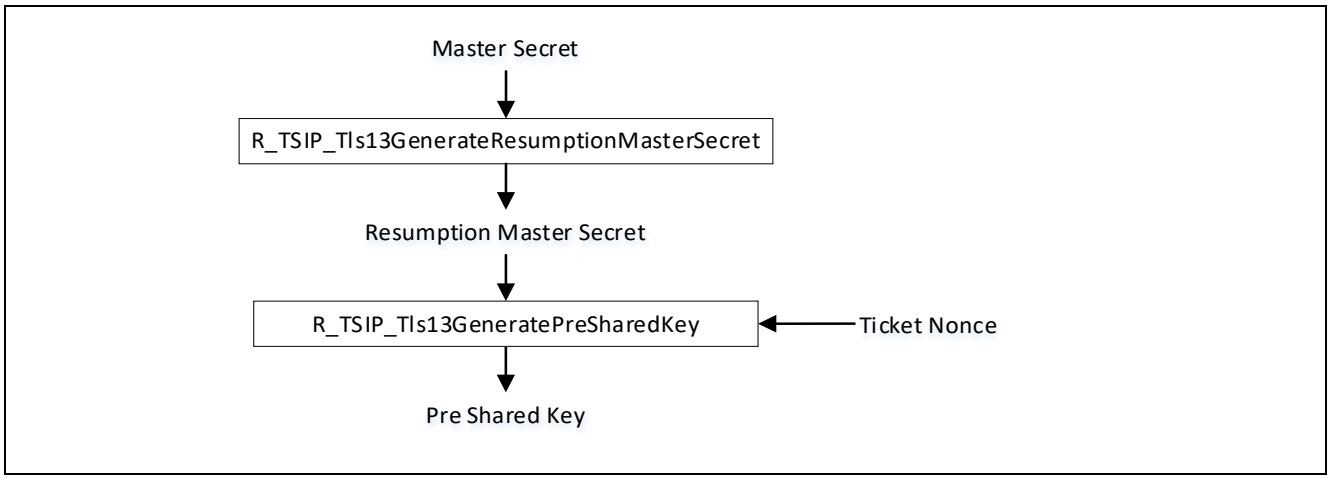


Figure 3.29 Generating Resumption Master Secret and Pre Shared Key

3. Sending Key Exchange Data
   3.1 Use R_TSIP_Tls13GeneratePskBinderKey() with the pre shared wrapped key as input to generate the binder wrapped key.
   3.2 Use R_TSIP_Sha256HmacGenerateInit/Update/Finish() with the binder wrapped key as input to generate the binder.
   3.3 As in item 2.1 of 3.13.1.1, use R_TSIP_GenerateTls13P256KeyIndex() to generate an ephemeral ECDH public key.
   3.4 When sending ClientHello to the server, also attach the pre shared key information including the binder as the pre_shared_key field and the ECDH public key as the key_share field.

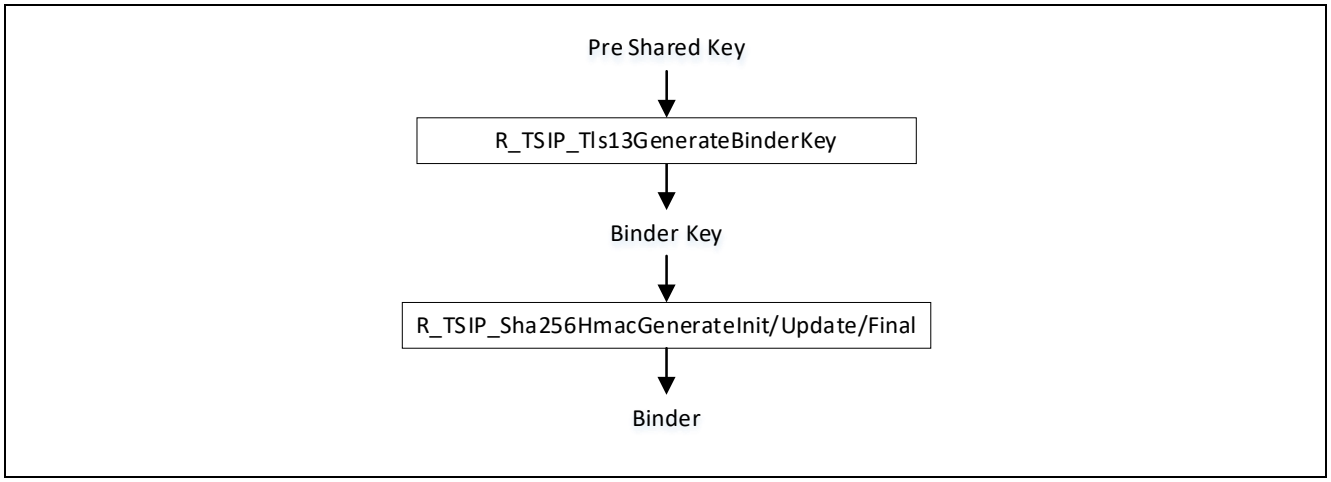


Figure 3.30 Sending Key Exchange Data

4. Generating Resumption Handshake Secret
   4.1 As in item 3.1 of 3.13.1.1, use R_TSIP_Tls13GenerateEcdheSharedSecret() with the public key received from the server as input to generate the shared secret wrapped key.
   4.2 Use R_TSIP_Tls13GenerateResumptionHandshakeSecret() with the pre shared wrapped key and shared secret wrapped key as input to generate the handshake secret wrapped key to be used for resumption.

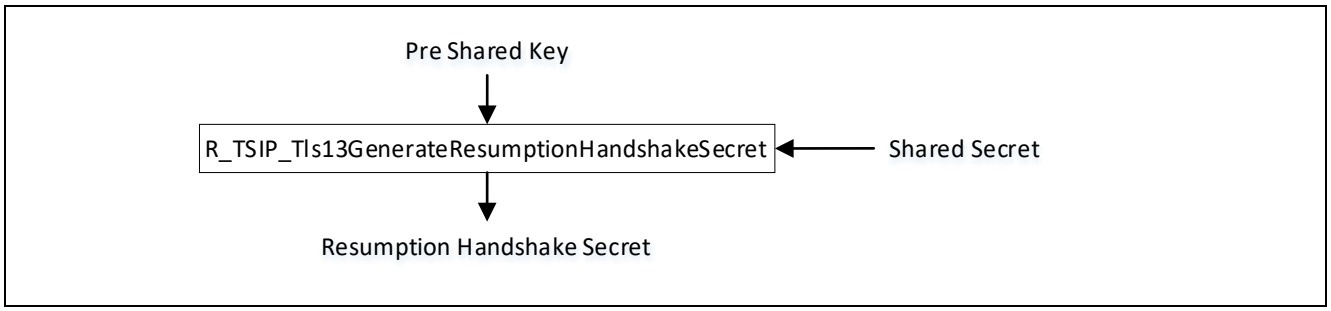


Figure 3.31 Generating Resumption Handshake Secret

5. Handshake and Application Data Communication
   5.1 After generating the handshake secret to be used for resumption, perform the same communication sequence as that for a full handshake to establish TLS 1.3 communication.
   5.2 After the handshake phase completes, application data communication takes place.

3.13.1.3 0-RTT

Figure 3.19 presents an overview of 0-RTT implementation using the TLS 1.3 client function.

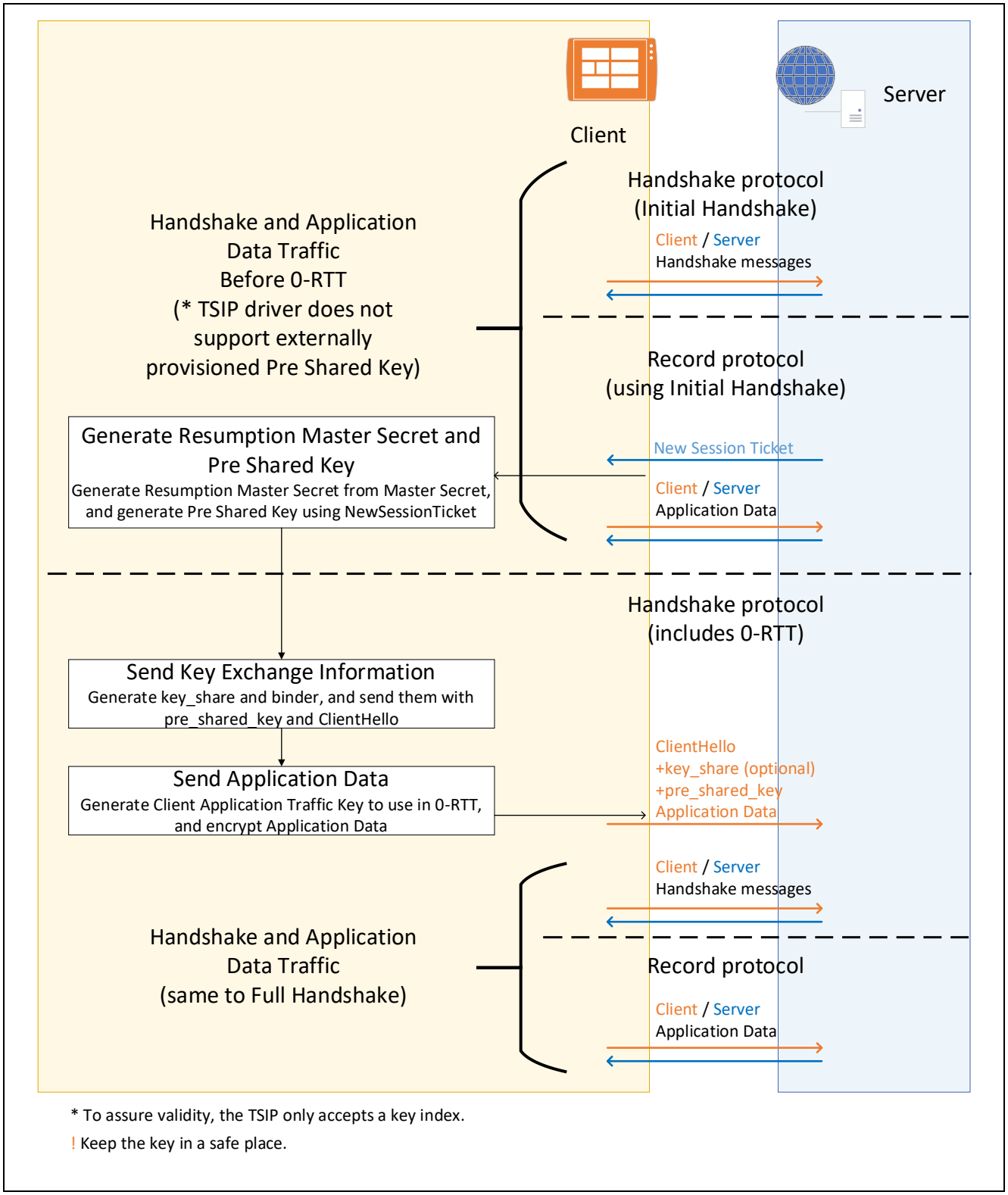


Figure 3.32 0-RTT Implementation Using TLS 1.3 Client Function

1. Handshake and Application Data Communication before Resumption
   1.1 TLS 1.3 communication is established by performing a full handshake.
   1.2 A new session ticket is received from the server as application data.

Note: The TSIP driver does not allow use of other than the pre shared key generated after handshaking completes.

2. Generating Resumption Master Secret and Pre Shared Key
   2.1 The subsequent processing is the same as item 2. of 3.13.1.2: Use R_TSIP_Tls13GenerateResumptionMasterSecret() with the master secret wrapped key used for the full handshake as input to generate the resumption master secret wrapped key.
   2.2 As in item 2.2 of 3.13.1.2, use R_TSIP_Tls13GeneratePreSharedKey() with the resumption master secret wrapped key and the ticket nonce within the new session ticket received from the server as input to generate the pre shared wrapped key.

3. Sending Key Exchange Data
   3.1 The subsequent processing is the same as item 3. of 3.13.1.2: Use R_TSIP_Tls13GeneratePskBinderKey() with the pre shared wrapped key as input to generate the binder wrapped key.
   3.2 Use R_TSIP_Sha256HmacGenerateInit/Update/Finish() with the binder wrapped key as input to generate the binder.
   3.3 As in item 2.1 of 3.13.1.1, use R_TSIP_GenerateTls13P256KeyIndex() to generate an ephemeral ECDH public key.
   3.4 When sending ClientHello to the server, also attach the pre shared key information including the binder as the pre_shared_key field and the ECDH public key as the key_share field.

4. Sending Application Data
   4.1 Use R_TSIP_Tls13Generate0RttApplicationWriteKey() with the pre shared wrapped key as input to generate the client write wrapped key to be used for 0-RTT.
   4.2 Use R_TSIP_Tls13EncryptInit/Update/Final() with the client write wrapped key as input to encrypt the application data.
   4.3 After sending ClientHello, send the encrypted application data to the server.

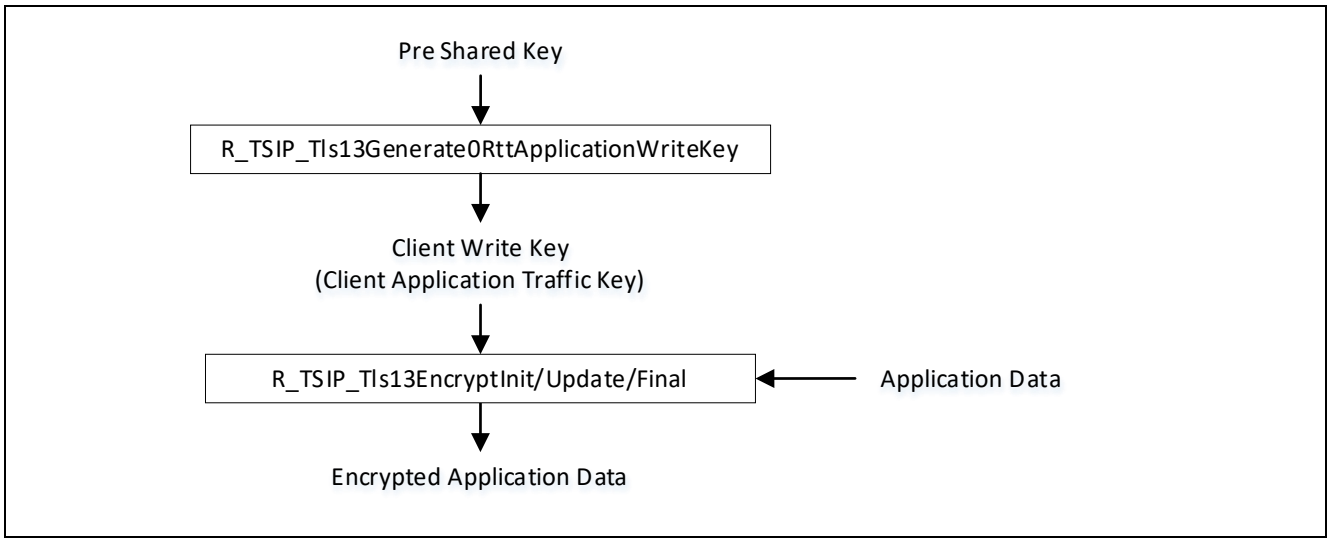


Figure 3.33 Sending Application Data

5. Handshake and Application Data Communication
   5.1 After sending the application data as 0-RTT, perform the same communication sequence as that for a full handshake to establish TLS 1.3 communication.
   5.2 After the handshake phase completes, application data communication takes place.

Note: As stated in section 2.3 of RFC 8446, when using 0-RTT the data is not forward secret and there are no guarantees of non-replay between connections. A judgment must be made with these risks in mind as to the use of this functionality.

3.13.2 TLS 1.3 Server Function

No.	API	Description
1	R_TSIP_GenerateTlsRsaPublicKeyIndex R_TSIP_UpdateTlsRsaPublicKeyIndex R_TSIP_TlsRootCertificateVerification R_TSIP_TlsCertificateVerification R_TSIP_TlsCertificateVerificationExtension R_TSIP_Tls13EncryptInit R_TSIP_Tls13EncryptUpdate R_TSIP_Tls13EncryptFinal R_TSIP_Tls13DecryptInit R_TSIP_Tls13DecryptUpdate R_TSIP_Tls13DecryptFinal R_TSIP_Tls13SVCertificateVerifyGenerate R_TSIP_Tls13SVCertificateVerifyVerification	Used in common by all functions.
2	R_TSIP_GenerateTls13SVP256EccKeyIndex R_TSIP_Tls13SVGenerateEcdheSharedSecret R_TSIP_Tls13SVGenerateHandshakeSecret R_TSIP_Tls13SVGenerateServerHandshakeTrafficKey R_TSIP_Tls13SVGenerateClientHandshakeTrafficKey R_TSIP_Tls13SVServerHandshakeVerification R_TSIP_Tls13SVGenerateMasterSecret R_TSIP_Tls13SVGenerateApplicationTrafficKey R_TSIP_Tls13SVUpdateApplicationTrafficKey	Mainly used by full handshake.
3	R_TSIP_Tls13SVGenerateResumptionMasterSecret R_TSIP_Tls13SVGeneratePreSharedKey R_TSIP_Tls13SVGeneratePskBinderKey R_TSIP_Tls13SVGenerateResumptionHandshakeSecret	Mainly used by resumption.
4	R_TSIP_Tls13SVGenerate0RttApplicationWriteKey	Used by 0-RTT.

3.13.2.1 Full Handshake

Figure 3.21 presents an overview of full handshake implementation using the TLS 1.3 server function.

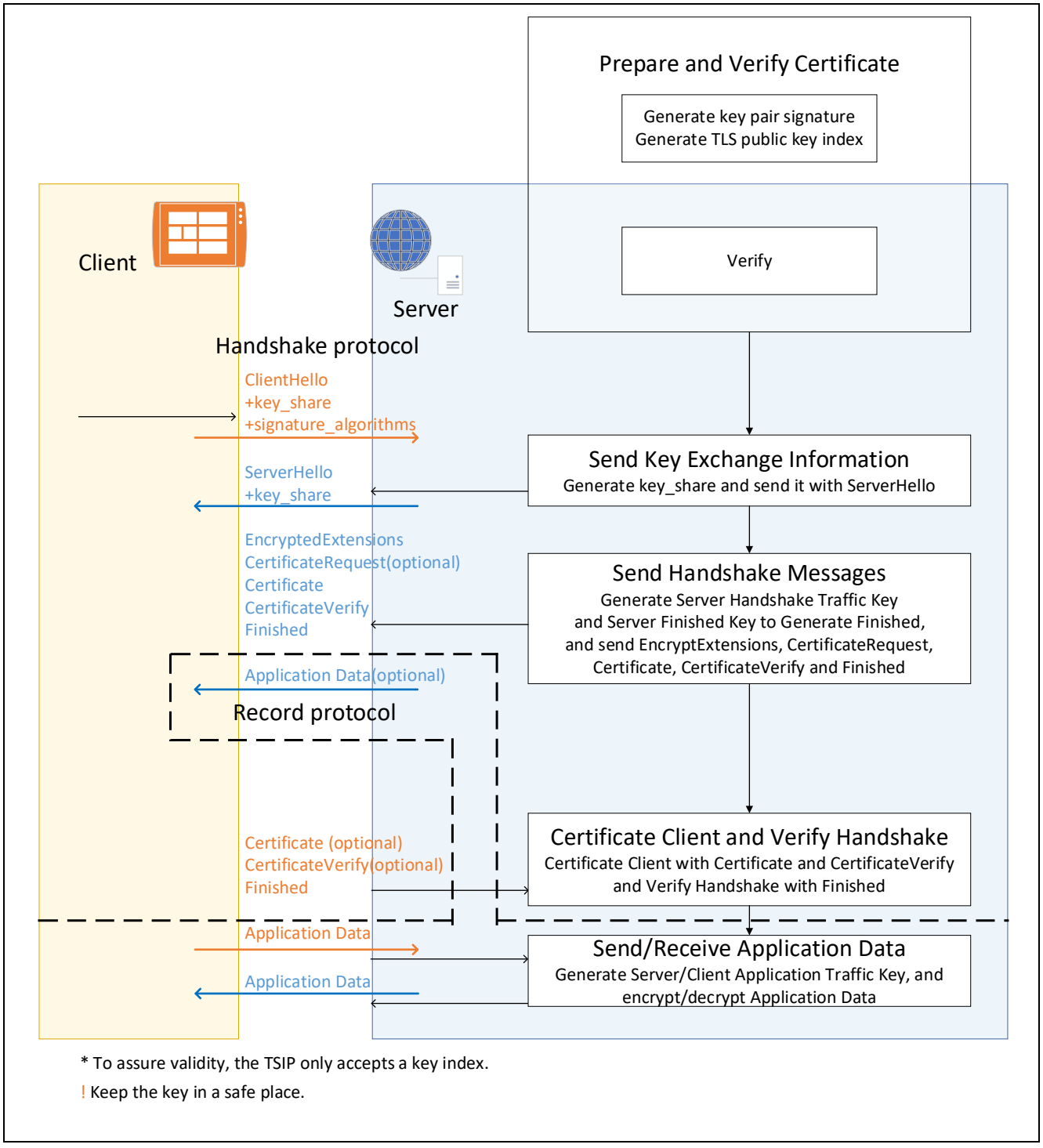


Figure 3.34 Full Handshake Implementation Using TLS 1.3 Server Function

1. Preparing and Verifying Certificates
   1.1 Before using TLS cooperation function, prepare the items below.
   - root CA certificate which the server connects to
   - RSA 2048bit key pair
   - server certificate.

   1.2 Generate signature value of the prepaired root CA certificate with using the private key of the key pair. The signature algorithm is RSA2048 PSS with SHA256. And generate Encrypted Key of the public key of the key pair with using UFPK. Note that the above procedure must be executed in a safe site.
   When multiple root CA certificate are needed to register, generate signature value of the root CA certificate bundle. It is possible to prevent making connection with using an unauthorized root CA certificate.

   1.3 Use R_TSIP_TlsSVRsaPublicKeyIndex with the Encrypted Key of the RSA public key as input to generate an RSA public wrapped key. Please refer to 3.7.1, Key Injection and Updating for more detail.

   1.4 Use R_TSIP_TlsRegisterCaCertificationPublicKey() to register the RSA wrapped key to the TSIP driver.

   1.5 Use R_TSIP_TlsSVRootCertificateVerification() to verify the root CA certificate bundle and generate an encrypted root CA certificate public key.

   1.6 Use R_TSIP_TlsSVCertificateVerification() or R_TSIP_TlsSVCertificateVerificationExtension() with the encrypted root CA certificate public key and server certificate public key as input to generate an encrypted server certificate public key.

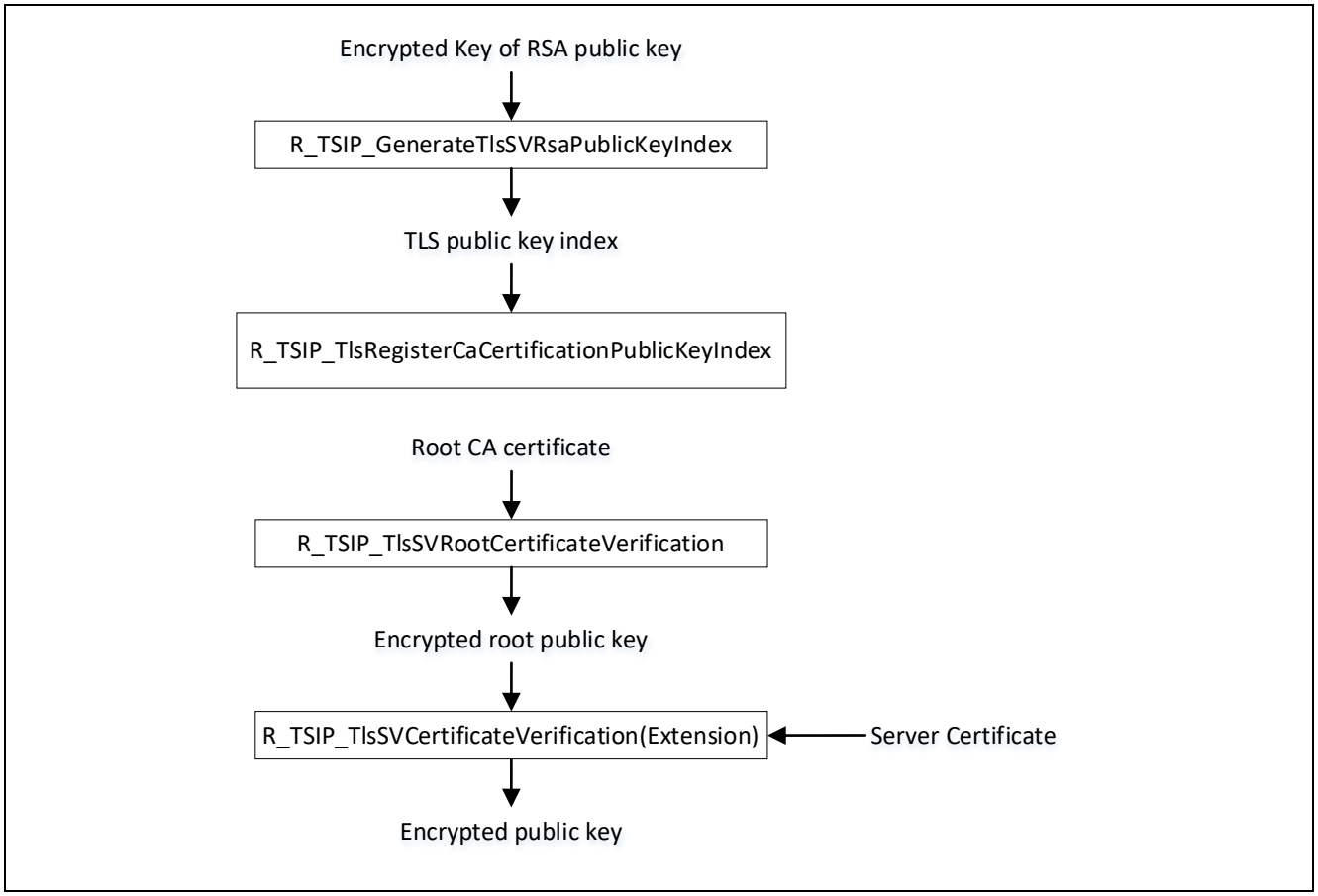


Figure 3.35 Preparing and Verifying Certificates

2. Sending Key Exchange Information
   2.1 Use R_TSIP_GenerateTls13P256KeyIndex() to generate an ephemeral ECDH public key.
   2.2 When ServerHello is sent, send the ECDH public key to the server as the key_share field.

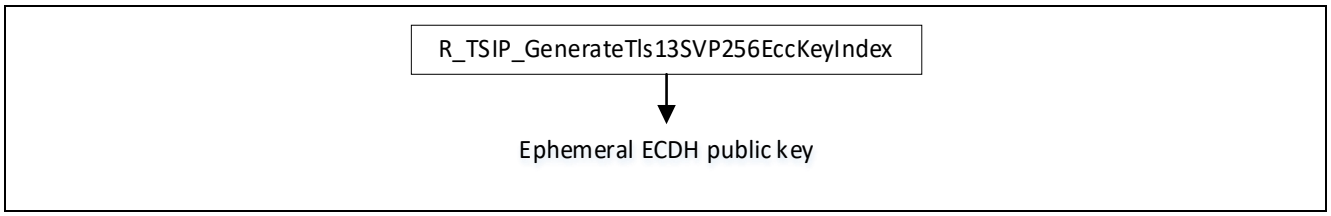


Figure 3.36 Sending Key Exchange Information

3. Sending Handshake Messages
   3.1 Use R_TSIP_Tls13SVCertificateVerifyGenerate() with the certificate's secret wrapped key as input to generate the CertificateVerify field.
   3.2 Use R_TSIP_Tls13SVGenerateEcdheSharedSecret() with the public key received from the client as input to generate the shared secret wrapped key.
   3.3 Use R_TSIP_Tls13SVGenerateHandshakeSecret() with the shared secret wrapped key as input to generate the handshake secret wrapped key.
   3.4 Use R_TSIP_Tls13SVGenerateServerHandshakeTrafficKey() with the handshake secret wrapped key as input to generate the server write wrapped key and server finished wrapped key to be used by the handshake protocol.
   3.5 Use R_TSIP_Tls13EncryptInit/Update/Final() with the server write wrapped key as input to encrypt the handshake messages to be sent to the client.
   3.6 Send the encrypted handshake messages.

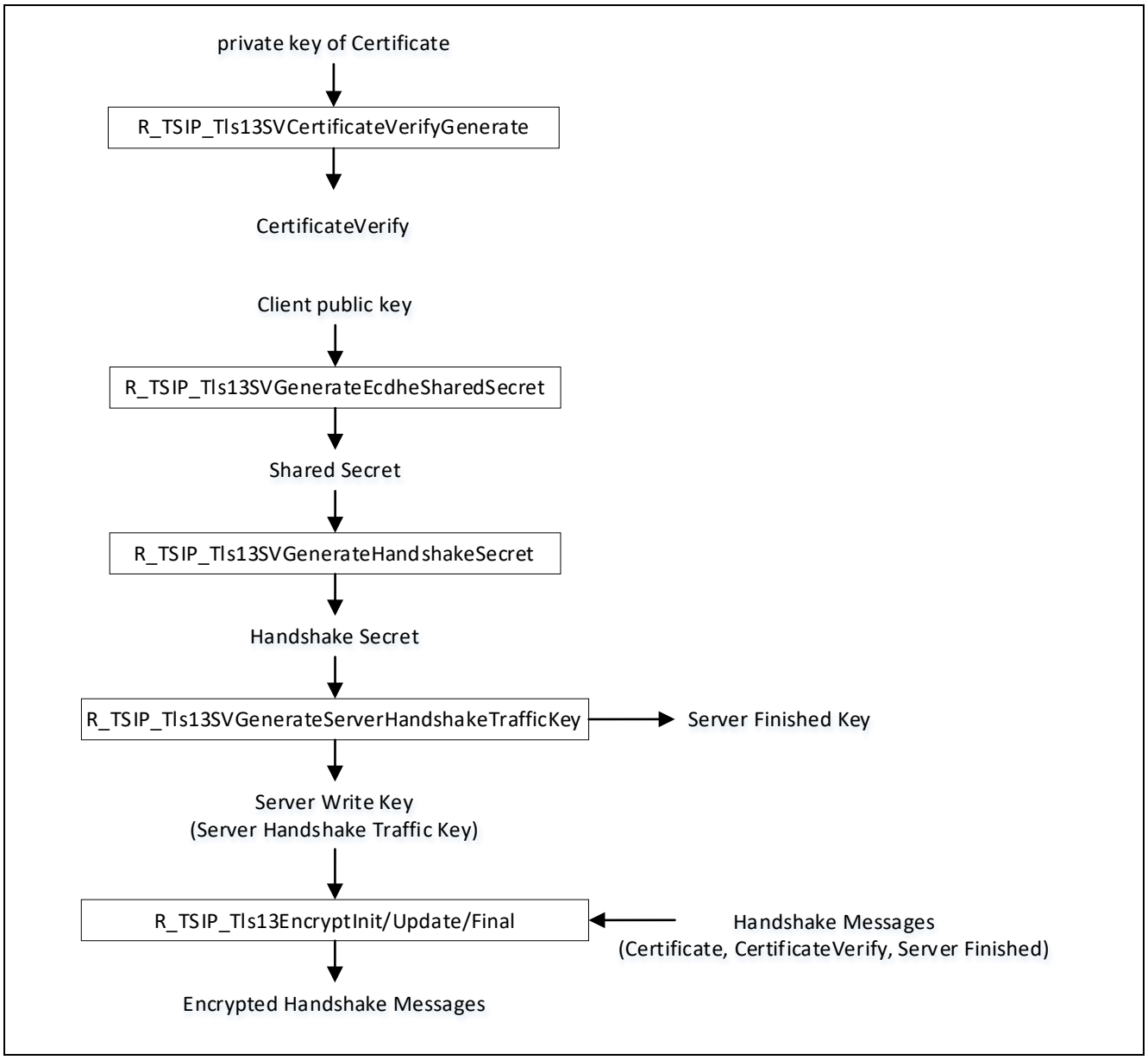


Figure 3.37 Sending Handshake Messages

4. Verifying Client and Handshake
   - 4.1 Use R_TSIP_Tls13SVGenerateClientHandshakeTrafficKey() with the handshake secret wrapped key as input to generate the client write wrapped key and client finished wrapped key to be used in the handshake phase.
   - 4.2 Use R_TSIP_Tls13DecryptInit/Update/Final() with the client write wrapped key as input to decrypt the encrypted handshake messages received from the client.
   - 4.3 Use R_TSIP_TlsCertificateVerification() with the (Client) Certificate field in the handshake message as input to verify the signature of the certificate.
   - 4.4 Use R_TSIP_Tls13SVCertificateVerifyVerification() with the (Client) CertificateVerify field in the handshake message as input to verify the (Client) CertificateVerify field.
   - 4.5 Use R_TSIP_Tls13SVClientHandshakeVerification() with the (Client) Finished field in the handshake message and the client finished wrapped key as input to perform handshake verification.

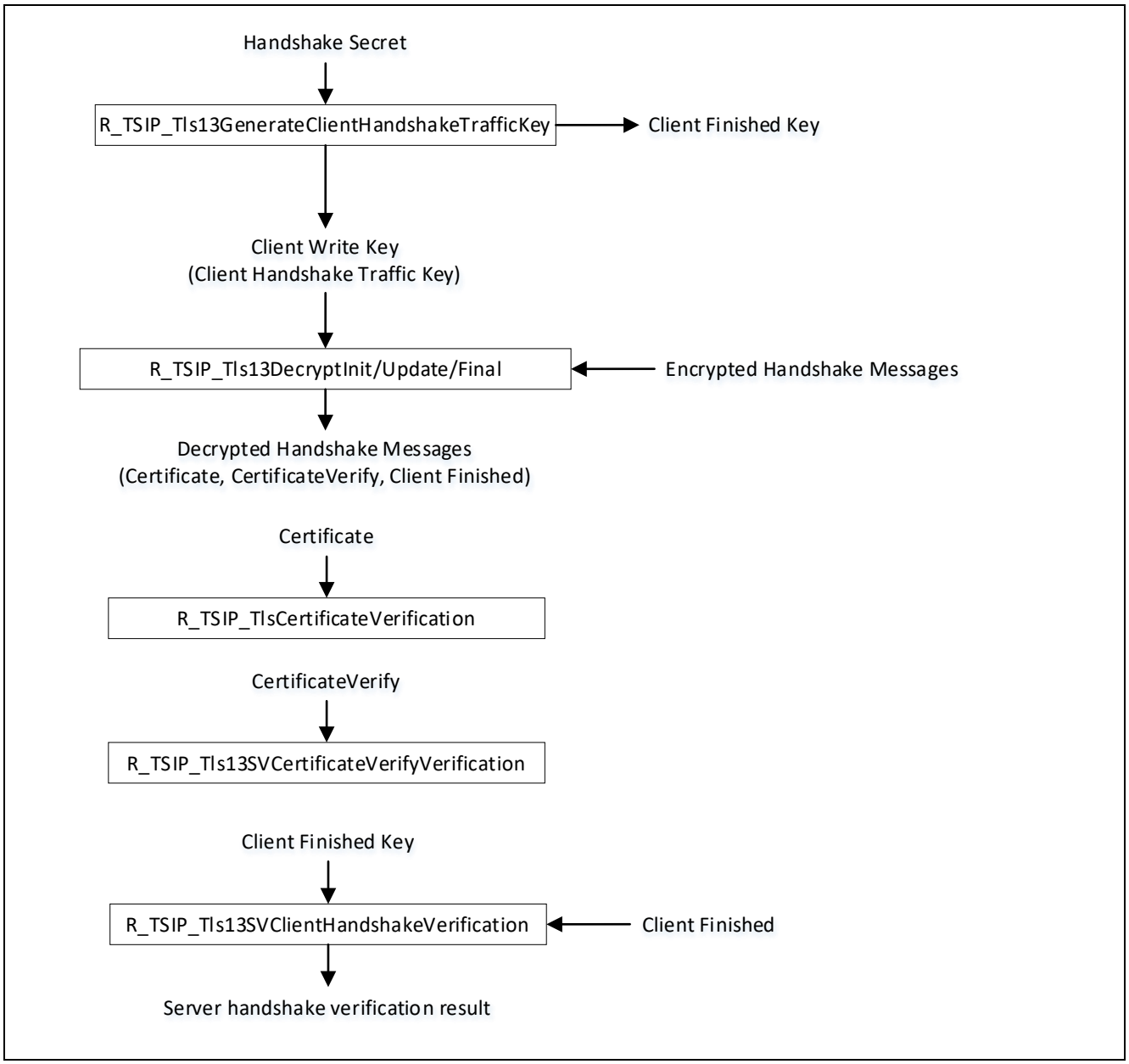


Figure 3.38 Verifying Client and Handshake

5. Sending/Receiving Application Data
   5.1 Use R_TSIP_Tls13SVGenerateMasterSecret() with the handshake secret wrapped key as input to generate the master secret wrapped key.
   5.2 Use R_TSIP_Tls13SVGenerateApplicationTrafficKey() with the master secret wrapped key as input to generate the server write wrapped key, client write wrapped key, and application secret wrapped key used by the record protocol.
   5.3 To update the server write key or client write key, use R_TSIP_Tls13SVUpdateApplicationTrafficKey() with the application secret wrapped key as input.
   5.4 To encrypt data to be sent to the server, use R_TSIP_Tls13EncryptInit/Update/Final() with the server write wrapped key as input.
   5.5 To decrypt data received from the server, use R_TSIP_Tls13DecryptInit/Update/Final() with the client write wrapped key as input.

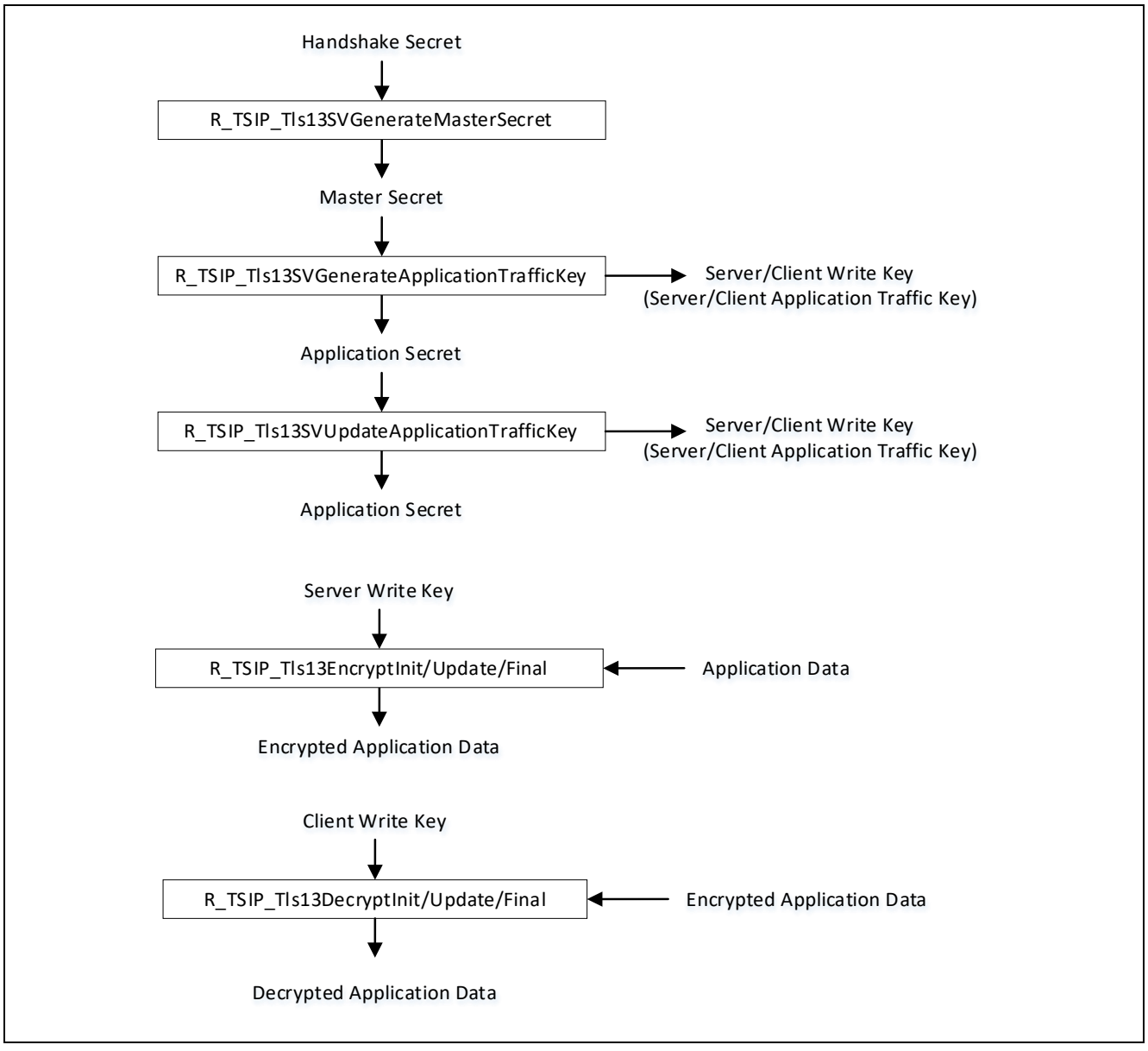


Figure 3.39 Sending/Receiving Application Data

Note: When application data is sent from the server without waiting to receive ClientFinished messages and a ClientFinished verification error occurs, the error can only be detected under conditions in which the server program has not been tampered with. A judgment must be made with these risks in mind as to the use of this functionality.

3.13.2.2 Resumption

Figure 3.27 presents an overview of resumption implementation using the TLS 1.3 server function.

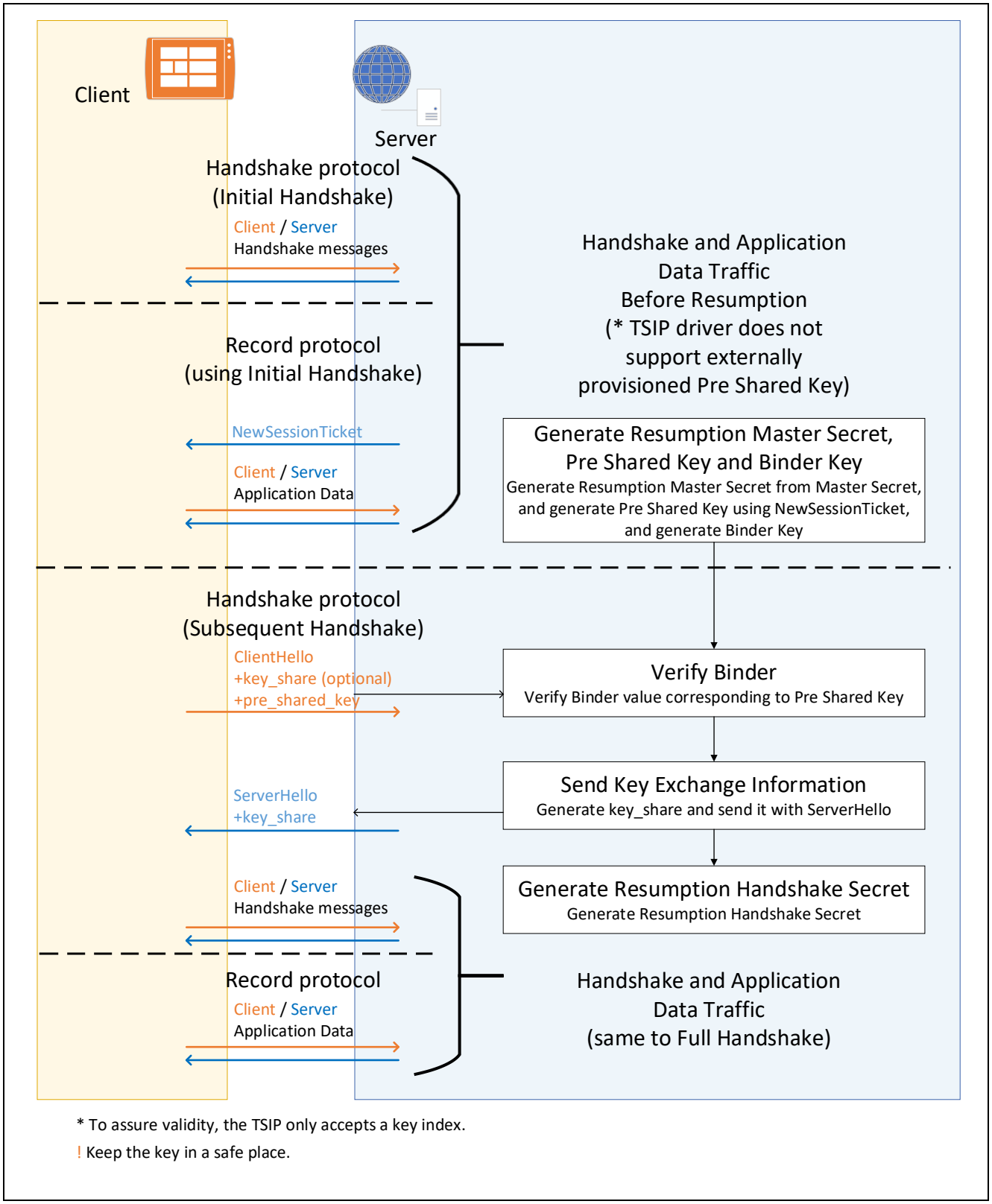


Figure 3.40 Resumption Implementation Using TLS 1.3 Server Function

1. Handshake and Application Data Communication before Resumption
   1.1 TLS 1.3 communication is established by performing a full handshake.

Note: The TSIP driver does not allow use of other than the pre shared key generated after handshaking completes.

2. Generating Resumption Master Secret, Pre Shared Key, and Binder Key
   2.1 Use R_TSIP_Tls13SVGenerateResumptionMasterSecret() with the master secret wrapped key used for the full handshake as input to generate the resumption master secret wrapped key.
   2.2 Use R_TSIP_Tls13SVGeneratePreSharedKey() with the resumption master secret wrapped key and the ticket nonce contained in the new session ticket sent to the client as input to generate the pre shared wrapped key.
   2.3 Use R_TSIP_Tls13SVGeneratePskBinderKey() with the pre shared wrapped key as input to generate the binder wrapped key.

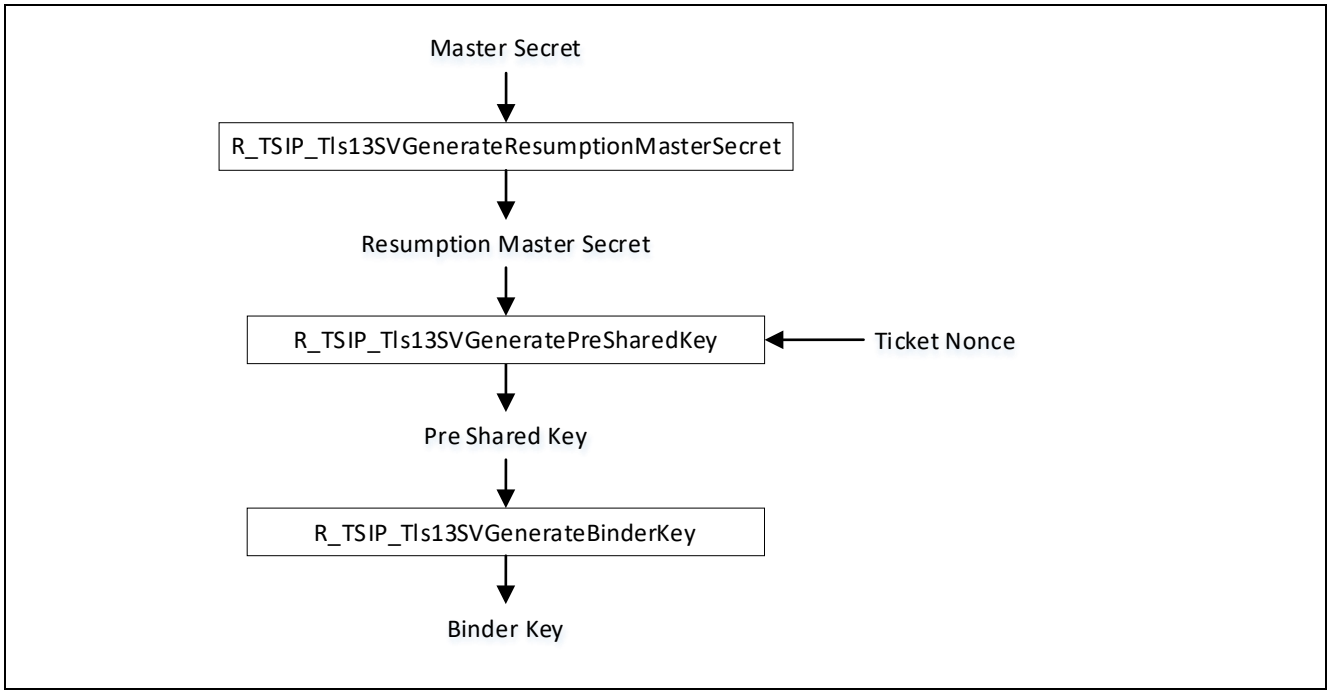


Figure 3.41 Generating Resumption Master Secret, Pre Shared Key, and Binder Key

3. Verifying Binder
   3.1 Use R_TSIP_Sha256HmacVerifyInit/Update/Final() with the binder key corresponding to the binder and pre shared key information received together with ClientHello as input to verify the binder.

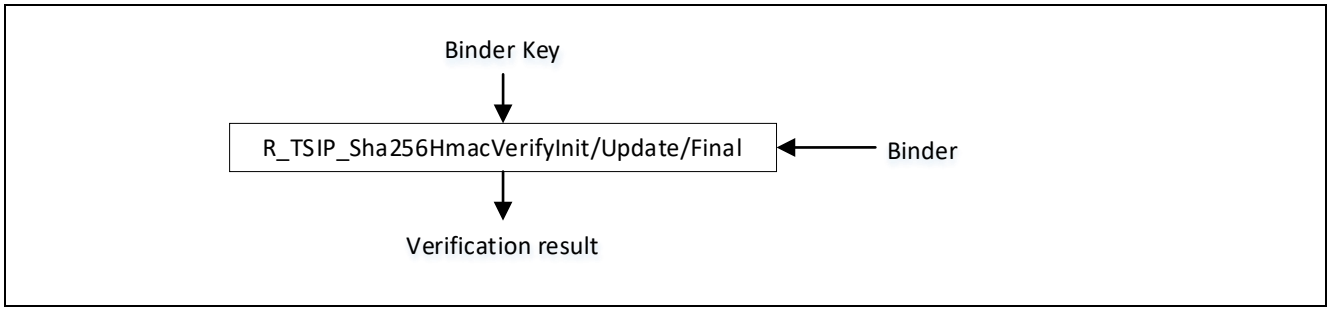


Figure 3.42 Verifying Binder

4. Generating Handshake Secret for Resumption
   4.1 As in item 3.2 of 3.13.2.1, use R_TSIP_Tls13SVGenerateEcdheSharedSecret() with the public key received from the server as input to generate the shared secret wrapped key.
   4.2 Use R_TSIP_Tls13SVGenerateResumptionHandshakeSecret() with the pre shared wrapped key and shared secret key index as input to generate the handshake secret wrapped key to be used for resumption.

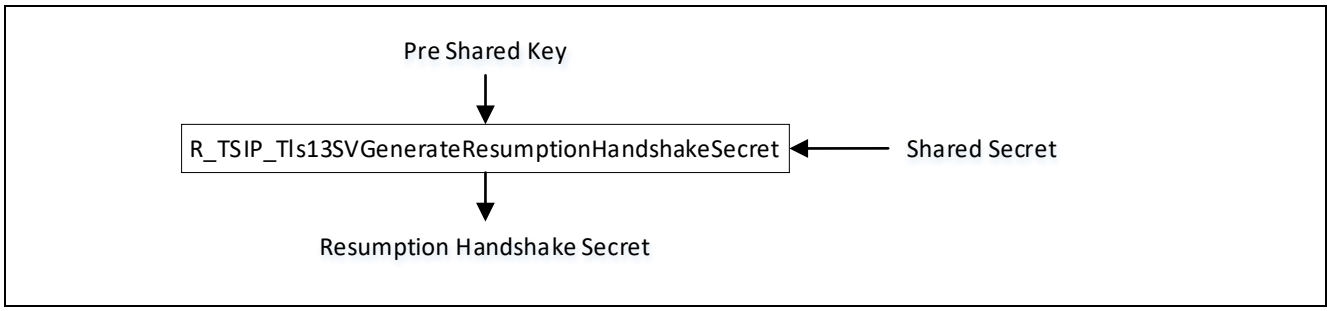


Figure 3.43 Generating Handshake Secret for Resumption

5. Handshake and Application Data Communication
   5.1 After generating the handshake secret to be used for resumption, perform the same communication sequence as that for a full handshake to establish TLS 1.3 communication.
   5.2 After the handshake phase completes, application data communication takes place.

3.13.2.3 0-RTT

Figure 3.31 presents an overview of 0-RTT implementation using the TLS 1.3 server function.

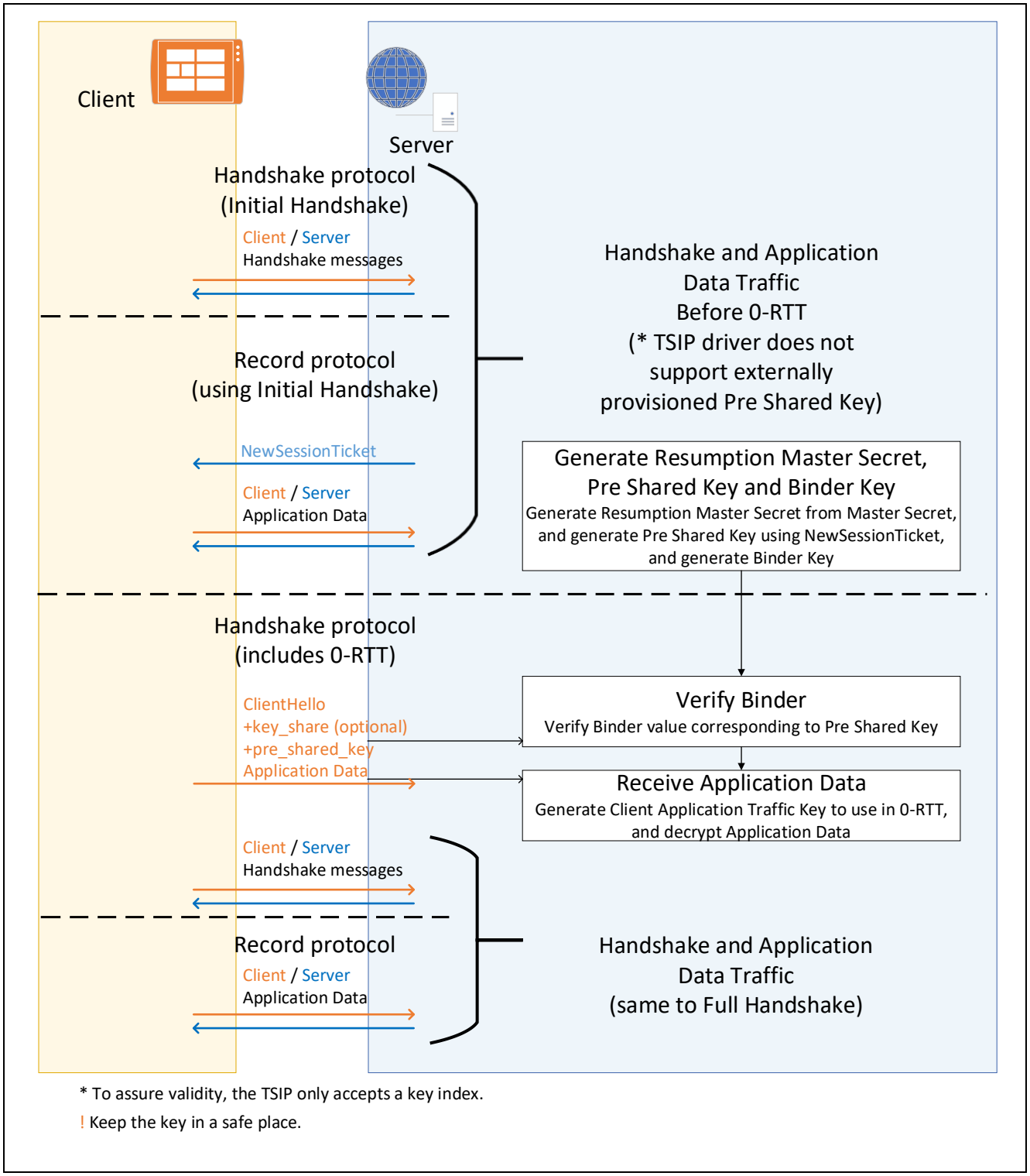


Figure 3.44 0-RTT Implementation Using TLS 1.3 Server Function

1. Handshake and Application Data Communication before Resumption
   1.1 TLS 1.3 communication is established by performing a full handshake.

Note: The TSIP driver does not allow use of other than the pre shared key generated after handshaking completes.

2. Generating Resumption Master Secret, Pre Shared Key, and Binder Key
   2.1 The subsequent processing is the same as item 2. of 3.13.2.2: Use R_TSIP_Tls13SVGenerateResumptionMasterSecret() with the master secret wrapped key used for the full handshake as input generate the resumption master secret wrapped key.
   2.2 Use R_TSIP_Tls13SVGeneratePreSharedKey() with the resumption master secret wrapped key and the ticket nonce contained in the new session ticket sent to the client as input to generate the pre shared wrapped key.
   2.3 Use R_TSIP_Tls13SVGeneratePskBinderKey() with the pre shared wrapped key as input to generate the binder key index.

3. Verifying Binder
   3.1 As in item 3.1 of 3.13.2.2, use R_TSIP_Sha256HmacVerifyInit/Update/Final() with the binder key corresponding to the binder and pre shared key information received together with ClientHello as input to verify the binder.

4. Receiving Application Data
   4.1 Use R_TSIP_Tls13SVGenerate0RttApplicationWriteKey() with the pre shared wrapped key as input to generate the client write wrapped key to be used for 0-RTT.
   4.2 Use R_TSIP_Tls13DecryptInit/Update/Final() with the client write wrapped key as input to decrypt the application data.

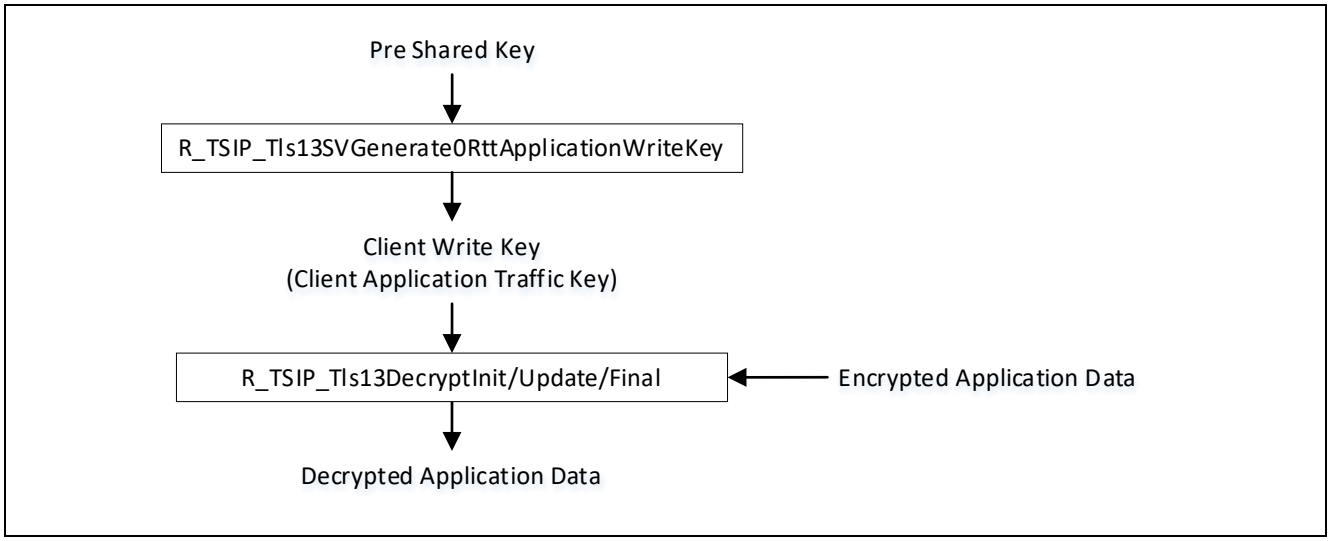


Figure 3.45 Receiving Application Data

5. Handshake and Application Data Communication
   5.1 After sending the application data as 0-RTT, perform the same communication sequence as that for a full handshake to establish TLS 1.3 communication.
   5.2 After the handshake phase completes, application data communication takes place.

Note: As stated in section 2.3 of RFC 8446, when using 0-RTT the data is not forward secret and there are no guarantees of non-replay between connections. A judgment must be made with these risks in mind as to the use of this functionality.

3.14 Firmware Update

The TSIP driver supports firmware update functionality to decrypt encrypted programs and perform MAC verification.

The driver provides APIs for the following firmware update operations:

No.	API	Description
1	R_TSIP_StartUpdateFirmware	Transitions the TSIP to a state in which the firmware update functionality can be used.
2	R_TSIP_GenerateFirmwareMAC*	Decrypts an encrypted program, performs MAC verification, and generates a MAC.
3	R_TSIP_VerifyFirmwareMAC*	Performs verification of a specified area using the MAC generated by R_TSIP_GenerateFirmwareMAC*.

* = Init, Update, Final

The firmware update functionality provides a mechanism for securely delivering an encrypted program and the key used to encrypt it. Figure 3.2 shows the program encryption, encryption key injection, and MAC verification operation sequence, including use of the Renesas Key Wrap service.

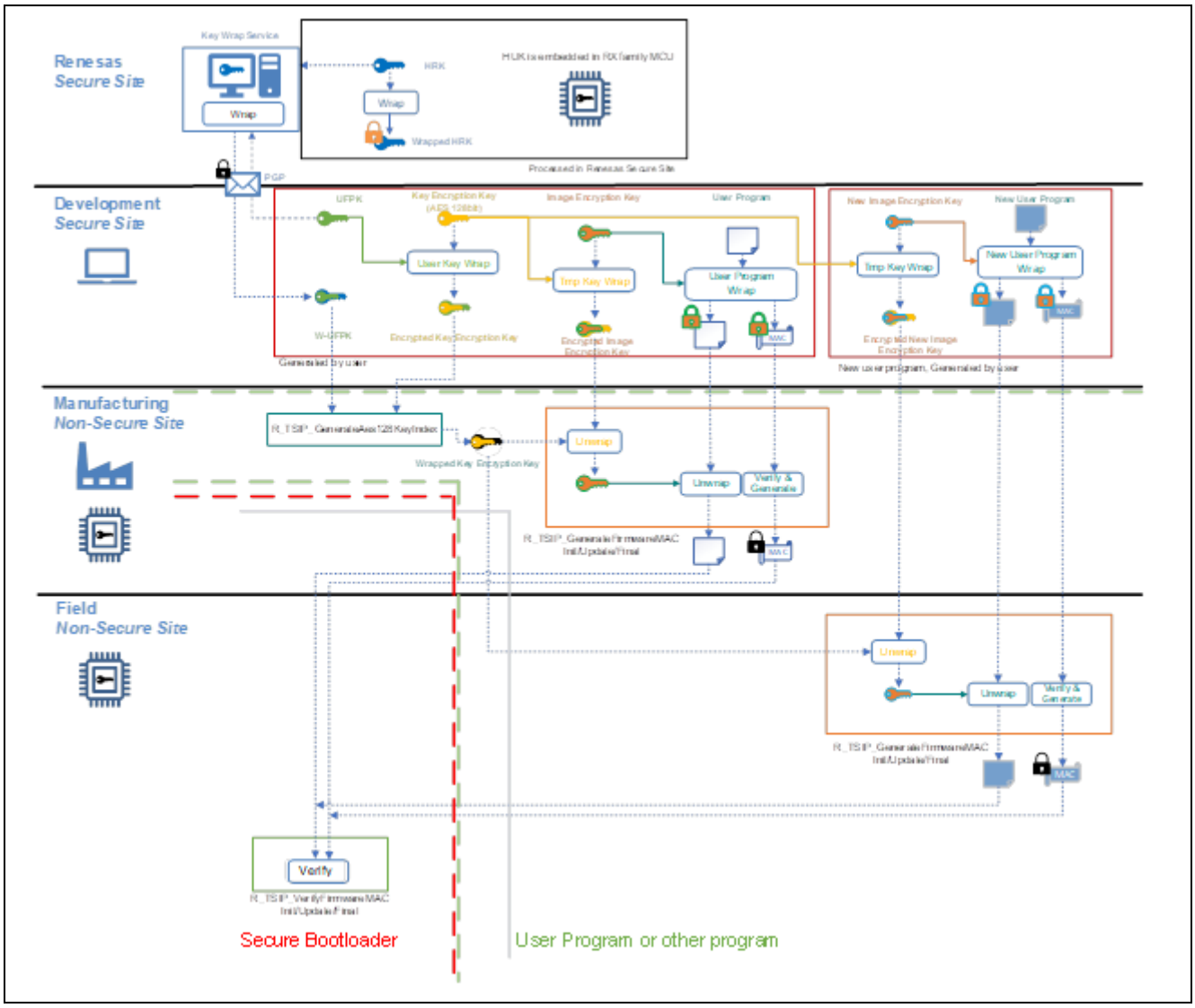


Figure 3.46 Firmware Update Sequence

3.14.1 Secure Boot

Secure boot refers to a program with functionality for detecting tampering with the user program. It is executed after a reset before the user program runs.

R_TSIP_VerifyFirmwareMACInit/Update/Final can be used as a secure boot program. Run R_TSIP_VerifyFirmwareMACInit/Update/Final with the MAC value output by R_TSIP_GenerateFirmwareMACInit/Update/Final as input.

3.14.2 Firmware Update

R_TSIP_GenerateFirmwareMACInit/Update/Final, which decrypts an encrypted program and performs MAC verification, can be used for firmware update operations. When MAC verification is successful, R_TSIP_GenerateFirmwareMACInit/Update/Final newly generates a MAC linked to the HRK.

R_TSIP_GenerateFirmwareMACInit/Update/Final can be run after first calling R_TSIP_StartUpdateFirmware to put the TSIP into firmware update status.

3.14.3 Encrypting the User Program

The method for encrypting the user program is shown below. AES-CBC is used as the firmware encryption algorithm and AES-CBCMAC as the MAC.

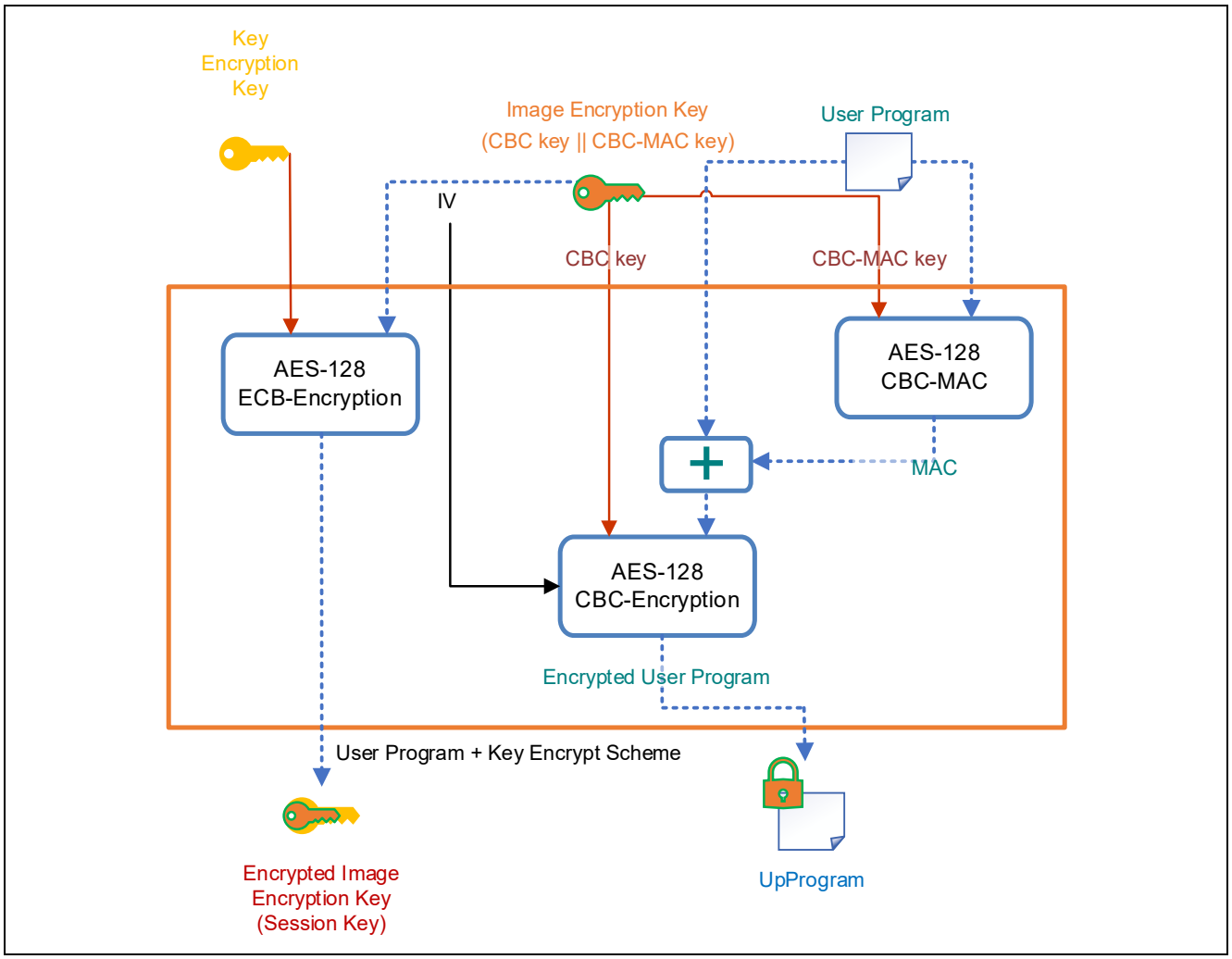


Figure 3.47 Firmware and Session Key Encryption Method

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 encrypt the image encryption key, and an encrypted image encryption key (session key) is generated.

4. API Functions

4.1 List of APIs

The TSIP driver implements the following APIs:

1. Common function APIs
2. Random number generation API
3. AES encryption/decryption APIs
4. DES encryption/decryption APIs
5. ARC4 encryption/decryption APIs
6. RSA operation APIs
7. ECC signature generation/verification APIs
8. HASH calculation APIs
9. HMAC generation/verification APIs
10. DH calculation API
11. ECDH key exchange APIs
12. Key wrap APIs
13. TLS function APIs
14. Firmware update/secure boot APIs

The APIs implemented in the TSIP 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	TSIP-Lite	TSIP
R_TSIP_Open	Enables TSIP functionality.	✓	✓
R_TSIP_Close	Disables TSIP functionality.	✓	✓
R_TSIP_SoftwareReset	Resets the TSIP module.	✓	✓
R_TSIP_GetVersion	Outputs the TSIP driver version.	✓	✓
R_TSIP_GenerateUpdateKeyRingKeyIndex	Generates a wrapped key for key updating.	✓	✓

Table 4-2 Random Number Generation API

API	Description	TSIP-Lite	TSIP
R_TSIP_GenrateRandomNumber	Generates random number.	✓	✓

Table 4-3 AES Encryption/Decryption APIs

API	Description	TSIP-Lite	TSIP
R_TSIP_GenerateAes*XXX*KeyIndex	Generates an AES wrapped key.	✓	✓
R_TSIP_UpdateAes*XXX*KeyIndex	Updates an AES wrapped key.	✓	✓
R_TSIP_GenerateAes*XXX*RandomKeyIndex	Generates an AES key from a random number and outputs the wrapped key.	✓	✓
R_TSIP_Aes*XXX*EcbEncryptInit R_TSIP_Aes*XXX*EcbEncryptUpdate R_TSIP_Aes*XXX*EcbEncryptFinal	Performs AES-ECB mode encryption.	✓	✓
R_TSIP_Aes*XXX*EcbDecryptInit R_TSIP_Aes*XXX*EcbDecryptUpdate R_TSIP_Aes*XXX*EcbDecryptFinal	Performs AES-ECB mode decryption.	✓	✓
R_TSIP_Aes*XXX*CbcEncryptInit R_TSIP_Aes*XXX*CbcEncryptUpdate R_TSIP_Aes*XXX*CbcEncryptFinal	Performs AES-CBC mode encryption.	✓	✓
R_TSIP_Aes*XXX*CbcDecryptInit R_TSIP_Aes*XXX*CbcDecryptUpdate R_TSIP_Aes*XXX*CbcDecryptFinal	Performs AES128-CBC mode decryption.	✓	✓
R_TSIP_Aes*XXX*CtrInit R_TSIP_Aes*XXX*CtrUpdate R_TSIP_Aes*XXX*CtrFinal	Performs AES-CTR mode encryption or decryption.	✓	✓
R_TSIP_Aes*XXX*GcmEncryptInit R_TSIP_Aes*XXX*GcmEncryptUpdate R_TSIP_Aes*XXX*GcmEncryptFinal	Performs AES-GCM encryption.	✓	✓
R_TSIP_Aes*XXX*GcmDecryptInit R_TSIP_Aes*XXX*GcmDecryptUpdate R_TSIP_Aes*XXX*GcmDecryptFinal	Performs AES-GCM decryption.	✓	✓
R_TSIP_Aes*XXX*CcmEncryptInit R_TSIP_Aes*XXX*CcmEncryptUpdate R_TSIP_Aes*XXX*CcmEncryptFinal	Performs AES-CCM encryption.	✓	✓
R_TSIP_Aes*XXX*CcmDecryptInit R_TSIP_Aes*XXX*CcmDecryptUpdate R_TSIP_Aes*XXX*CcmDecryptFinal	Performs AES-CCM decryption.	✓	✓
R_TSIP_Aes*XXX*CmacGenerateInit R_TSIP_Aes*XXX*CmacGenerateUpdate R_TSIP_Aes*XXX*CmacGenerateFinal	Performs AES-CMAC mode MAC generation.	✓	✓
R_TSIP_Aes*XXX*CmacVerifyInit R_TSIP_Aes*XXX*CmacVerifyUpdate R_TSIP_Aes*XXX*CmacVerifyFinal	Verifies a MAC generated in AES-CMAC mode.	✓	✓

Table 4-4 DES Encryption/Decryption APIs

API	Description	TSIP-Lite	TSIP
R_TSIP_GenerateTdesKeyIndex	Generates a TDES wrapped key.	—	✓
R_TSIP_UpdateTdesKeyIndex	Updates a TDES wrapped key.	—	✓
R_TSIP_GenerateTdesRandomKeyIndex	Generates a TDES key from a random number and outputs the wrapped key.	—	✓
R_TSIP_TdesEcbEncryptInit R_TSIP_TdesEcbEncryptUpdate R_TSIP_TdesEcbEncryptFinal	Performs TDES-ECB mode encryption.	—	✓
R_TSIP_TdesEcbDecryptInit R_TSIP_TdesEcbDecryptUpdate R_TSIP_TdesEcbDecryptFinal	Performs TDES-ECB mode decryption.	—	✓
R_TSIP_TdesCbcEncryptInit R_TSIP_TdesCbcEncryptUpdate R_TSIP_TdesCbcEncryptFinal	Performs TDES-CBC mode encryption.	—	✓
R_TSIP_TdesCbcDecryptInit R_TSIP_TdesCbcDecryptUpdate R_TSIP_TdesCbcDecryptFinal	Performs TDES-CBC mode decryption.	—	✓

Table 4-5 ARC4 Encryption/Decryption APIs

API	Description	TSIP-Lite	TSIP
R_TSIP_GenerateArc4KeyIndex	Generates an ARC4 wrapped key.	—	✓
R_TSIP_UpdateArc4KeyIndex	Updates an ARC4 wrapped key.	—	✓
R_TSIP_GenerateArc4RandomKeyIndex	Generates an ARC4 key from a random number and outputs the wrapped key.	—	✓
R_TSIP_Arc4EncryptInit R_TSIP_Arc4EncryptUpdate R_TSIP_Arc4EncryptFinal	Performs ARC4 encryption.	—	✓
R_TSIP_Arc4DecryptInit R_TSIP_Arc4DecryptUpdate R_TSIP_Arc4DecryptFinal	Performs ARC4 decryption.	—	✓

Table 4-6 RSA Operation APIs

API	Description	TSIP-Lite	TSIP
R_TSIP_GenerateRsa*XXX*PrivateKeyIndex	Generates an RSA secret wrapped key.	—	✓
R_TSIP_GenerateRsa*XXX*PublicKeyIndex	Generates an RSA public wrapped key.	—	✓
R_TSIP_UpdateRsa*XXX*PrivateKeyIndex	Updates an RSA secret wrapped key.	—	✓
R_TSIP_UpdateRsa*XXX*PublicKeyIndex	Updates an RSA public wrapped key.	—	✓
R_TSIP_GenerateRsa*XXX*RandomKeyIndex	Generates a pair of RSA wrapped keys from a random number. Exponent is fixed at 0x10001.	—	✓
R_TSIP_RsaesPkcs*XXX*Encrypt	Performs RSA encryption using RSAES-PKCS1-V1_5.	—	✓
R_TSIP_RsaesPkcs*XXX*Decrypt	Performs RSA decryption using RSAES-PKCS1-V1_5.	—	✓
R_TSIP_RsaesOaep*XXX*Encrypt	Performs RSA encryption using RSAES-OAEP.	—	✓
R_TSIP_RsaesOaep*XXX*Decrypt	Performs RSA decryption using RSAES-OAEP.	—	✓
R_TSIP_RsassaPkcs*XXX*SignatureGenerate	Generates a digital signature using RSASSA-PKCS1-V1_5.	—	✓
R_TSIP_RsassaPkcs*XXX*SignatureVerification	Verifies a digital signature using RSASSA-PKCS1-V1_5.	—	✓
R_TSIP_RsassaPss*XXX*SignatureGenerate	Generates a digital signature using RSASSA-PSS.	—	✓
R_TSIP_RsassaPss*XXX*SignatureVerification	Verifies a digital signature using RSASSA-PSS.	—	✓

Table 4-7 ECC Signature Generation/Verification APIs

API	Description	TSIP-Lite	TSIP
R_TSIP_GenerateEccP*XXX*PublicKeyIndex	Generates an ECC public wrapped key.	—	✓
R_TSIP_GenerateEccP*XXX*PrivateKeyIndex	Generates an ECC secret wrapped key.	—	✓
R_TSIP_UpdateEccP*XXX*PublicKeyIndex	Updates an ECC public wrapped key.	—	✓
R_TSIP_UpdateEccP*XXX*PrivateKeyIndex	Updates an ECC secret wrapped key.	—	✓
R_TSIP_GenerateEccP*XXX*RandomKeyIndex	Generates a pair of ECC wrapped keys from a random number.	—	✓
R_TSIP_EcdsaP*XXX*SignatureGenerate	Generates a digital signature using ECDSA.	—	✓
R_TSIP_EcdsaP*XXX*SignatureVerification	Verifies a digital signature using ECDSA.	—	✓

Table 4-8 HASH Calculation APIs

API	Description	TSIP-Lite	TSIP
R_TSIP_Sha*XXX*Init R_TSIP_Sha*XXX*Update R_TSIP_Sha*XXX*Final	Performs hash value operations using SHA.	—	✓
R_TSIP_Md5Init R_TSIP_Md5Update R_TSIP_Md5Final	Performs hash value operations using MD5.	—	✓
R_TSIP_GetCurrentHashDigestValue	Gets hash value for current input.	—	✓

Table 4-9 HMAC Generation/Verification APIs

API	Description	TSIP-Lite	TSIP
R_TSIP_GenerateSha*XXX*HmacKeyIndex	Generates an SHA-HMAC wrapped key.	—	✓
R_TSIP_UpdateSha*XXX*HmacKeyIndex	Updates an SHA-HMAC wrapped key.	—	✓
R_TSIP_Sha*XXX*HmacGenerateInit R_TSIP_Sha*XXX*HmacGenerateUpdate R_TSIP_Sha*XXX*HmacGenerateFinal	Performs SHA-HMAC generation.	—	✓
R_TSIP_Sha*XXX*HmacVerifyInit R_TSIP_Sha*XXX*HmacVerifyUpdate R_TSIP_Sha*XXX*HmacVerifyFinal	Performs SHA-HMAC verification.	—	✓

Table 4-10 DH Calculation API

API	Description	TSIP-Lite	TSIP
R_TSIP_Rsa2048DhKeyAgreement	Performs DH operations using RSA-2048.	—	✓

Table 4-11 ECDH Key Exchange APIs

API	Description	TSIP-Lite	TSIP
R_TSIP_EcdhP256Init	Prepares for performance of ECDH P-256 key exchange operations.	—	✓
R_TSIP_EcdhP256ReadPublicKey	Verifies the ECC P-256 public key signature of the other key exchange party.	—	✓
R_TSIP_EcdhP256MakePublicKey	Signs an ECC P-256 secret key.	—	✓
R_TSIP_EcdhP256CalculateSharedSecretIndex	Calculates the shared secret Z from the public key of the other key exchange party and your own secret key.	—	✓
R_TSIP_EcdhP256KeyDerivation	Derives Z from a shared key.	—	✓
R_TSIP_EcdheP512KeyAgreement	Performs ECDHE operations using Brainpool P512r1.	—	✓
R_TSIP_EcdhP256SshKeyDerivation	Derives Z from a shared key to use in SSH.	—	✓

Table 4-12 Key Exchange APIs

API	Description	TSIP-Lite	TSIP
R_TSIP_Aes*XXX*KeyWrap	Wraps a key using an AES key.	✓	✓
R_TSIP_Aes*XXX*KeyUnwrap	Unwraps a key wrapped with an AES key.	✓	✓

Table 4-13 TLS Function APIs

API	Description	TSIP -Lite	TSIP
R_TSIP_GenerateTls*XX*RsaPublicKeyIndex	Generates an RSA public wrapped key used in TLS cooperation.	—	✓
R_TSIP_UpdateTls*XX*RsaPublicKeyIndex	Updates an RSA public wrapped key used in TLS cooperation.	—	✓
R_TSIP_RegisterCaCertificationPublicKeyIndex		—	✓
R_TSIP_Tls*XX*RootCertificateVerification	Verifies a root CA certificate bundle.	—	✓
R_TSIP_Tls*XX*CertificateVerification	Verifies the signature of a server certificate or intermediate certificate.	—	✓
R_TSIP_Tls*XX*CertificateVerificationExtension	Verifies the signature of a server certificate or intermediate certificate.	—	✓
R_TSIP_TlsGeneratePreMasterSecret	Generates an encrypted pre-master secret.	—	✓
R_TSIP_TlsEncryptPreMasterSecretWithRsa2048PublicKey	Encrypts a pre-master secret using RSA-2048.	—	✓
R_TSIP_TlsSVGenerateServerRandom	Generates a random number value to use in ServerHello.	—	✓
R_TSIP_TlsSVDecryptPreMasterSecretWithRsa2048PrivateKey	De crypts a pre-master secret using RSA-2048.	—	✓
R_TSIP_Tls*XX*GenerateMasterSecret	Generates an encrypted master secret.	—	✓
R_TSIP_Tls*XX*GenerateSessionKey	Outputs TLS communication keys.	—	✓
R_TSIP_Tls*XX*GenerateVerifyData	Generates a VerifyData message.	—	✓
R_TSIP_TlsServersEphemeralEcdhPublicKeyRetrieves	Verifies a server key exchange signature.	—	✓
R_TSIP_GenerateTls*XX*P256EccKeyIndex	Generates a key pair from a random number used by the TLS cooperation function for elliptic curve cryptography over a 256-bit prime field.	—	✓
R_TSIP_Tls*XX*GeneratePreMasterSecretWithEcP256Key	Generates an ECC encrypted pre-master secret.	—	✓
R_TSIP_Tls*XX*GenerateExtendedMasterSecret	Generates an encrypted extended master secret.	—	✓
R_TSIP_TlsSVCertificateVerifyVerification	Verifies a CertificateVerify message received from the client.	—	✓
R_TSIP_GenerateTls13P256EccKeyIndex	Generates a key pair from a random number used by the TLS 1.3 cooperation function for elliptic curve cryptography over a 256-bit prime field.	—	✓
R_TSIP_Tls13GenerateEcdheSharedSecret	Generates a shared secret wrapped key.	—	✓
R_TSIP_Tls13GenerateHandshakeSecret	Generates a handshake secret wrapped key.	—	✓
R_TSIP_Tls13GenerateServerHandshakeTrafficKey	Generates a server write wrapped key and server finished wrapped key.	—	✓
R_TSIP_Tls13ServerHandshakeVerification	Verifies finished information provided by the server.	—	✓
R_TSIP_Tls13GenerateClientHandshakeTrafficKey	Generates a client write wrapped key and client finished wrapped key.	—	✓
R_TSIP_Tls13GenerateMasterSecret	Generates a master secret key index.	—	✓
R_TSIP_Tls13GenerateApplicationTrafficKey	Generates an application traffic secret wrapped key and an application traffic wrapped key.	—	✓

API	Description	TSIP -Lite	TSIP
R_TSIP_Tls13UpdateApplicationTrafficKey	Updates an application traffic secret wrapped key and an application traffic wrapped key.	—	✓
R_TSIP_Tls13GenerateResumptionMasterSecret	Generates a resumption master secret wrapped key.	—	✓
R_TSIP_Tls13GeneratePreSharedKey	Generates a pre shared wrapped key.	—	✓
R_TSIP_Tls13GeneratePskBinderKey	Generates a binder wrapped key.	—	✓
R_TSIP_Tls13GenerateResumptionHandshakeSecret	Generates a handshake secret wrapped key for resumption.	—	✓
R_TSIP_Tls13Generate0RttApplicationWriteKey	Generates a client write wrapped key for 0-RTT.	—	✓
R_TSIP_Tls13CertificateVerifyGenerate	Generates a CertificateVerify message to be sent to the server.	—	✓
R_TSIP_Tls13CertificateVerifyVerification	Verifies a CertificateVerify message received from the server.	—	✓
R_TSIP_GenerateTls13SVP256EccKeyIndex	Generates a key pair from a random number used by the TLS 1.3 cooperation function for elliptic curve cryptography over a 256-bit prime field.	—	✓
R_TSIP_Tls13SVGenerateEcdheSharedSecret	Generates a shared secret wrapped key.	—	✓
R_TSIP_Tls13SVGenerateHandshakeSecret	Generates a handshake secret wrapped key.	—	✓
R_TSIP_Tls13SVGenerateServerHandshakeTrafficKey	Generates a server write wrapped key and server finished wrapped key.	—	✓
R_TSIP_Tls13SVGenerateClientHandshakeTrafficKey	Generates a client write wrapped key and client finished wrapped key.	—	✓
R_TSIP_Tls13SVClientHandshakeVerification	Verifies finished information provided by the client.	—	✓
R_TSIP_Tls13SVGenerateMasterSecret	Generates a master secret wrapped key.	—	✓
R_TSIP_Tls13SVGenerateApplicationTrafficKey	Generates an application traffic secret wrapped key and application traffic wrapped key.	—	✓
R_TSIP_Tls13SVUpdateApplicationTrafficKey	Updates an application traffic secret wrapped key and application traffic wrapped key.	—	✓
R_TSIP_Tls13SVGenerateResumptionMasterSecret	Generates a resumption master secret wrapped key.	—	✓
R_TSIP_Tls13SVGeneratePreSharedKey	Generates a pre shared wrapped key.	—	✓
R_TSIP_Tls13SVGeneratePskBinderKey	Generates a binder wrapped key.	—	✓
R_TSIP_Tls13SVGenerateResumptionHandshakeSecret	Generates a handshake secret wrapped key for resumption.	—	✓
R_TSIP_Tls13SVGenerate0RttApplicationWriteKey	Generates a client write wrapped key for 0-RTT.	—	✓
R_TSIP_Tls13SVCertificateVerifyGenerate	Generates a CertificateVerify message to be sent to the client.	—	✓
R_TSIP_Tls13SVCertificateVerifyVerification	Verifies a CertificateVerify message received from the client.	—	✓
R_TSIP_Tls13EncryptInit R_TSIP_Tls13EncryptUpdate R_TSIP_Tls13EncryptFinal	Encrypts TLS 1.3 communication data.	—	✓

API	Description	TSIP -Lite	TSIP
R_TSIP_Tls13DecryptInit R_TSIP_Tls13DecryptUpdate R_TSIP_Tls13DecryptFinal	Decrypts TLS 1.3 communication data.	—	✓

Table 4-14 Firmware Update APIs

API	Description	TSIP-Lite	TSIP
R_TSIP_StartUpdateFirmware	Transitions to firmware update mode.	✓	✓
R_TSIP_GenerateFirmwareMACInit R_TSIP_GenerateFirmwareMACUpdate R_TSIP_GenerateFirmwareMACFinal	Decrypts and generates the MAC for encrypted firmware.	✓	✓
R_TSIP_VerifyFirmwareMACInit R_TSIP_VerifyFirmwareMACUpdate R_TSIP_VerifyFirmwareMACFinal	Performs a MAC check on the firmware.	✓	✓

4.2 Detailed Descriptions of API Functions

4.2.1 Common Function APIs

4.2.1.1 R_TSIP_Open

Format

```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_Open (
        tsip_tls_ca_certification_public_key_index_t *key_index_1,
        tsip_update_key_ring_t *key_index_2
)
```

Parameters

key_index_1	Input	TLS cooperation RSA public wrapped key for TLS Client Function
key_index_2	Input	Wrapped KUK

Return Values

TSIP_SUCCESS	Normal termination
TSIP_ERR_FAIL	Abnormal termination of self-diagnostics
TSIP_ERR_RESOURCE_CONFLICT	Occurrence of resource conflict because a hardware resource needed by the processing routine was in use by another processing routine
TSIP_ERR_RETRY	Abnormal termination of self-diagnostics Run the function again.
TSIP_ERR_ALREADY_OPEN	TSIP driver is already open.

Description

Enables use of TSIP functionality.

For key_index_1, input the “TLS cooperation RSA public wrapped key for TLS Client Function” generated by R_TSIP_GenerateTlsRsaPublicKeyIndex() or R_TSIP_UpdateTlsRsaPublicKeyIndex(). If the TLS cooperation function is not used, input a null pointer.

For key_index_2, input the “Wrapped KUK” generated by R_TSIP_GenerateUpdateKeyRingKeyIndex(). This registration process of the wrapped key can be executed either this API or R_TSIP_TlsRegisterCaCertificationPublicKeyIndex(). If the key update cooperation function is not used or the process is executed without this API, input a null pointer.

Note: to prevent the RX MCU from transitioning to standby mode while R_TSIP_Open() is running, R_TSIP_Open() internally calls the R_BSP_InterruptsDisable() API to disable interrupts and then the R_BSP_InterruptsEnable() API to enable interrupts.

Reentrancy

Not supported.

4.2.1.2 R_TSIP_Close

Format

```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_Close(void)
```

Parameters

None

Return Values

TSIP_SUCCESS	Normal termination

Description

Stops TSIP functionality.

Reentrancy

Not supported.

4.2.1.3 R_TSIP_SoftwareReset

Format

```
#include “r_tsip_rx_if.h”
void R_TSIP_SoftwareReset(void)
```

Parameters

None

Return Values

None

Description

Returns the TSIP to the initial state.

Reentrancy

Not supported.

4.2.1.4 R_TSIP_GetVersion

Format

```
#include “r_tsip_rx_if.h”
uint32_t R_TSIP_GetVersion(void)
```

Parameters

None

Return Values

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

Description

This function can be used to obtain the TSIP driver version.

Reentrancy

Not supported.

4.2.1.5 R_TSIP_GenerateUpdateKeyRingKeyIndex

Format

```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_GenerateUpdateKeyRingKeyIndex(
        uint8_t *encrypted_provisioning_key,
        uint8_t *iv,
        uint8_t *encrypted_key,
        tsip_update_key_ring_t *key_index
)
```

Parameters

encrypted_provisioning_key	Input	W-UFPK
iv	Input	Initialization vector when generating encrypted_key
encrypted_key	Input	Encrypted KUK
key_index	Output	Wrapped KUK

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_FAIL:	Occurrence of internal error
TSIP_ERR_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 outputs a wrapped key of KUK.

For encrypted_key, input the data indicated in 7.3.7, Update Keyring, encrypted using the UFPK.

Refer to 3.7.1, Key Injection and Updating, for an explanation of encrypted_provisioning_key, iv, and encrypted_key and how to use key_index.

Reentrancy

Not supported.

4.2.2 Random Number Generation

4.2.2.1 R_TSIP_GenerateRandomNumber

Format

```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_GenerateRandomNumber(
        uint32_t *random
)
```

Parameters

random	Output	4-word (16-byte) random number value

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_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 can be used to generate an NIST SP800-90A–compliant 4-word random number value.

Reentrancy

Not supported.

4.2.3 AES

4.2.3.1 R_TSIP_GenerateAes*XXX*KeyIndex

Format

(1) #include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_GenerateAes128KeyIndex(
uint8_t *encrypted_provisioning_key,
uint8_t *iv,
uint8_t *encrypted_key,
tsip_aes_key_index_t *key_index
)

(2) #include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_GenerateAes256KeyIndex(
uint8_t *encrypted_provisioning_key,
uint8_t *iv,
uint8_t *encrypted_key,
tsip_aes_key_index_t *key_index
)

Parameters

encrypted_provisioning_key	Input	W-UFPK
iv	Input	Initialization vector when generating encrypted_key
encrypted_key	Input	Encrypted key encrypted using UFPK
key_index	Output	Wrapped key

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_FAIL:	Occurrence of internal error
TSIP_ERR_RESOURCE_CONFLICT:	Occurrence of resource conflict because a hardware resource needed by the processing routine was in use by another processing routine

Description

R_TSIP_GenerateAes128KeyIndex is an API that outputs an AES 128-bit wrapped key.

R_TSIP_GenerateAes256KeyIndex is an API that outputs an AES 256-bit wrapped key.

For encrypted_key, input the data indicated in 7.3.1, AES, encrypted using the UFPK.

Refer to 3.7.1, Key Injection and Updating, for an explanation of encrypted_provisioning_key, iv, and encrypted_key and how to use key_index.

Reentrancy

Not supported.

4.2.3.2 R_TSIP_UpdateAes*XXX*KeyIndex

Format

(1)
```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_UpdateAes128KeyIndex(
        uint8_t *iv,
        uint8_t *encrypted_key,
        tsip_aes_key_index_t *key_index
        )
```

(2)
```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_UpdateAes256KeyIndex(
        uint8_t *iv,
        uint8_t *encrypted_key,
        tsip_aes_key_index_t *key_index
        )
```

Parameters

iv	Input	Initialization vector when generating encrypted_key
encrypted_key	Input	Encrypted key encrypted using KUK
key_index	Output	Wrapped key

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_FAIL:	Occurrence of internal error
TSIP_ERR_RESOURCE_CONFLICT:	Occurrence of resource conflict because a hardware resource needed by the processing routine was in use by another processing routine

Description

R_TSIP_UpdateAes128KeyIndex is an API that updates the wrapped key of an AES 128 key.

R_TSIP_UpdateAes256KeyIndex is an API that updates the wrapped key of an AES 256 key.

For encrypted_key, input the data indicated in 7.3.1, AES, encrypted using the KUK.

Refer to 3.7.1, Key Injection and Updating, for an explanation of iv and encrypted_key and how to use key_index.

Reentrancy

Not supported.

4.2.3.3 R_TSIP_GenerateAes*XXX*RandomKeyIndex

Format

(1) #include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_GenerateAes128RandomKeyIndex(
tsip_aes_key_index_t *key_index
)

(2) #include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_GenerateAes256RandomKeyIndex(
tsip_aes_key_index_t *key_index
)

Parameters

key_index	Output	(1) 128-bit AES wrapped key (2) 256-bit AES wrapped key

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_RESOURCE_CONFLICT:	Occurrence of resource conflict because a hardware resource needed by the processing routine was in use by another processing routine

Description

R_TSIP_GenerateAes128RandomKeyIndex is an API that outputs an AES 128-bit wrapped key.

R_TSIP_GenerateAes256RandomKeyIndex is an API that outputs an AES 256-bit wrapped key.

This API generates a user key from a random number within the TSIP. Therefore, no user key needs to be input. Encrypting data using the wrapped key output by this API makes it possible to prevent dead copying of data.

Refer to 3.7.1, Key Injection and Updating, for an explanation of how to use key_index.

Reentrancy

Not supported.

4.2.3.4 R_TSIP_Aes*XXX*EcbEncryptInit

Format

(1)
```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_Aes128EcbEncryptInit(
        tsip_aes_handle_t *handle,
        tsip_aes_key_index_t *key_index
        )
```

(2)
```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_Aes256EcbEncryptInit(
        tsip_aes_handle_t *handle,
        tsip_aes_key_index_t *key_index
        )
```

Parameters

handle	Output	AES handler (work area)
key_index	Input	Wrapped key

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_FAIL:	Occurrence of internal error (Only for TSIP)
TSIP_ERR_RESOURCE_CONFLICT:	Occurrence of resource conflict because a hardware resource needed by the processing routine was in use by another processing routine
TSIP_ERR_KEY_SET:	Invalid wrapped key input

Description

The R_TSIP_AesXXXEcbEncryptInit() function performs preparations for the execution of AES calculation and writes the result to the first parameter, handle. The parameter handle is used subsequently as a parameter by the R_TSIP_AesXXXEcbEncryptUpdate() and R_TSIP_AesXXXEcbEncryptFinal() functions.

Refer to 3.7.1, Key Injection and Updating, for an explanation of how to generate key_index.

Reentrancy

Not supported.

4.2.3.5 R_TSIP_Aes*XXX*EcbEncryptUpdate

Format

(1)
```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_Aes128EcbEncryptUpdate(
        tsip_aes_handle_t *handle,
        uint8_t *plain,
        uint8_t *cipher,
        uint32_t plain_length
        )
```

(2)
```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_Aes256EcbEncryptUpdate(
        tsip_aes_handle_t *handle,
        uint8_t *plain,
        uint8_t *cipher,
        uint32_t plain_length
        )
```

Parameters

handle	Input/output	AES handler (work area)
plain	Input	Plaintext data area
cipher	Output	Ciphertext data area
plain_length	Input	Byte length of plaintext data (Must be a multiple of 16.)

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_PARAMETER:	Invalid handle input
TSIP_ERR_PROHIBIT_FUNCTION:	Invalid function called

Description

Using the handle specified by the first parameter, handle, the R_TSIP_AesXXXEcbEncryptUpdate() function encrypts the second parameter, plain, using the key index specified by the R_TSIP_AesXXXEcbEncryptInit() function, and writes the encrypted result to the third parameter, cipher. After plaintext input completes, call R_TSIP_AesXXXEcbEncryptFinal().

Except in cases where the addresses are the same, specify areas for plain and cipher that do not overlap.

Reentrancy

Not supported.

4.2.3.6 R_TSIP_Aes*XXX*EcbEncrypFinal

Format

(1) #include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_Aes128EcbEncryptFinal(
tsip_aes_handle_t *handle,
uint8_t *cipher,
uint32_t *cipher_length
)

(2) #include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_Aes256EcbEncryptFinal(
tsip_aes_handle_t *handle,
uint8_t *cipher,
uint32_t *cipher_length
)

Parameters

handle	Input	AES handler (work area)
cipher	Output	Ciphertext data area (Nothing is ever written here.)
cipher_length	Output	Ciphertext data length (The write value is always 0.)

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_FAIL:	Occurrence of internal error
TSIP_ERR_PARAMETER:	Invalid handle input
TSIP_ERR_PROHIBIT_FUNCTION:	Invalid function called

Description

Using the handle specified by the first parameter, handle, the R_TSIP_AesXXXEcbEncryptFinal() function writes the calculation result to the second parameter, cipher, and writes the length of the calculation result to the third parameter, cipher_length. The original intent was for any portion of the encrypted result that was not a multiple of 16 bytes to be written to the second parameter. However, due to the restriction that only multiples of 16 can be input to the R_TSIP_AesXXXEcbEncryptUpdate() function, nothing is ever written to cipher, and 0 is always written to cipher_length. The parameters cipher and cipher_length are provided for future compatibility in anticipation of this restriction eventually being removed.

Reentrancy

Not supported.

4.2.3.7 R_TSIP_Aes*XXX*EcbDecryptInit

Format

(1)
```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_Aes128EcbDecryptInit(
        tsip_aes_handle_t *handle,
        tsip_aes_key_index_t *key_index
)
```
(2)
```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_Aes256EcbDecryptInit(
        tsip_aes_handle_t *handle,
        tsip_aes_key_index_t *key_index
)
```

Parameters

handle	Output	AES handler (work area)
key_index	Input	Wrapped key

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_FAIL:	Occurrence of internal error (Only for TSIP)
TSIP_ERR_RESOURCE_CONFLICT:	Occurrence of resource conflict because a hardware resource needed by the processing routine was in use by another processing routine
TSIP_ERR_KEY_SET:	Invalid wrapped key input

Description

The R_TSIP_AesXXXEcbDecryptInit() function performs preparations for the execution of AES calculation and writes the result to the first parameter, handle. The parameter handle is used subsequently as a parameter by the R_TSIP_AesXXXEcbDecryptUpdate() and R_TSIP_AesXXXEcbDecryptFinal() functions.

Refer to 3.7.1, Key Injection and Updating, for an explanation of how to generate key_index.

Reentrancy

Not supported.

4.2.3.8 R_TSIP_Aes*XXX*EcbDecryptUpdate

Format

(1)
```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_Aes128EcbDecryptUpdate(
        tsip_aes_handle_t *handle,
        uint8_t *cipher,
        uint8_t *plain,
        uint32_t cipher_length
        )
```

(2)
```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_Aes256EcbDecryptUpdate(
        tsip_aes_handle_t *handle,
        uint8_t *cipher,
        uint8_t *plain,
        uint32_t cipher_length
        )
```

Parameters

handle	Input/output	AES handler (work area)
cipher	Input	Ciphertext data area
plain	Output	Plaintext data area
cipher_length	Input	Byte length of ciphertext data (Must be a multiple of 16.)

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_PARAMETER:	Invalid handle input
TSIP_ERR_PROHIBIT_FUNCTION:	Invalid function called

Description

Using the handle specified by the first parameter, handle, the R_TSIP_AesXXXEcbDecryptUpdate() function decrypts the second parameter, cipher, using the key index specified by the R_TSIP_AesXXXEcbDecryptInit() function, and writes the decrypted result to the third parameter, plain. After ciphertext input completes, call R_TSIP_AesXXXEcbDecryptFinal().

Except in cases where the addresses are the same, specify areas for plain and cipher that do not overlap.

Reentrancy

Not supported.

4.2.3.9 R_TSIP_Aes*XXX*EcbDecryptFinal

Format

(1) #include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_Aes128EcbDecryptFinal(
tsip_aes_handle_t *handle,
uint8_t *plain,
uint32_t *plain_length
)

(2) #include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_Aes256EcbDecryptFinal(
tsip_aes_handle_t *handle,
uint8_t *plain,
uint32_t *plain_length
)

Parameters

handle	Input	AES handler (work area)
plain	Output	Plaintext data area (Nothing is ever written here.)
plain_length	Output	Plaintext data length (The write value is always 0.)

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_FAIL:	Occurrence of internal error
TSIP_ERR_PARAMETER:	Invalid handle input
TSIP_ERR_PROHIBIT_FUNCTION:	Invalid function called

Description

Using the handle specified by the first parameter, handle, the R_TSIP_AesXXXEcbDecryptFinal() function writes the calculation result to the second parameter, plain, and writes the length of the calculation result to the third parameter, plain_length. The original intent was for any portion of the decrypted result that was not a multiple of 16 bytes to be written to the second parameter. However, due to the restriction that only multiples of 16 can be input to the R_TSIP_AesXXXEcbDecryptUpdate() function, nothing is ever written to plain, and 0 is always written to plain_length. The parameters plain and plain_length are provided for future compatibility in anticipation of this restriction eventually being removed.

Reentrancy

Not supported.

4.2.3.10 R_TSIP_Aes*XXX*CbcEncryptInit

Format

(1)
```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_Aes128CbcEncryptInit(
        tsip_aes_handle_t *handle,
        tsip_aes_key_index_t *key_index,
        uint8_t *ivec
)
```

(2)
```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_Aes256CbcEncryptInit(
        tsip_aes_handle_t *handle,
        tsip_aes_key_index_t *key_index,
        uint8_t *ivec
)
```

Parameters

handle	Output	AES handler (work area)
key_index	Input	Wrapped key
ivec	Input	Initialization vector (16 bytes)

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_FAIL:	Occurrence of internal error (Only for TSIP)
TSIP_ERR_RESOURCE_CONFLICT:	Occurrence of resource conflict because a hardware resource needed by the processing routine was in use by another processing routine
TSIP_ERR_KEY_SET:	Invalid wrapped key input

Description

The R_TSIP_AesXXXCbcEncryptInit() function performs preparations for the execution of AES calculation, and writes the result to the first parameter, handle. The parameter handle is used subsequently as a parameter by the R_TSIP_AesXXXCbcEncryptUpdate() and R_TSIP_AesXXXCbcEncryptFinal() functions.

When using the TLS cooperation function, input client_crypto_key_index or server_crypto_key_index, generated by R_TSIP_TlsGenerateSessionKey(), as key_index.

Refer to 3.7.1, Key Injection and Updating, for an explanation of how to generate key_index.

Reentrancy

Not supported.

4.2.3.11 R_TSIP_Aes*XXX*CbcEncryptUpdate

Format

(1) #include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_Aes128CbcEncryptUpdate(
tsip_aes_handle_t *handle,
uint8_t *plain,
uint8_t *cipher,
uint32_t plain_length
)

(2) #include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_Aes256CbcEncryptUpdate(
tsip_aes_handle_t *handle,
uint8_t *plain,
uint8_t *cipher,
uint32_t plain_length
)

Parameters

handle	Input/output	AES handler (work area)
plain	Input	Plaintext data area
cipher	Output	Ciphertext data area
plain_length	Input	Byte length of plaintext data (Must be a multiple of 16.)

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_PARAMETER:	Invalid handle input
TSIP_ERR_PROHIBIT_FUNCTION:	Invalid function called

Description

Using the handle specified by the first parameter, handle, the R_TSIP_AesXXXCbcEncryptUpdate() function encrypts the second parameter, plain, using the key index specified by the R_TSIP_AesXXXCbcEncryptInit() function, and writes the encrypted result to the third parameter, cipher. After plaintext input completes, call R_TSIP_AesXXXCbcEncryptFinal().

Except in cases where the addresses are the same, specify areas for plain and cipher that do not overlap.

Reentrancy

Not supported.

4.2.3.12 R_TSIP_Aes*XXX*CbcEncryptFinal

Format

(1)
```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_Aes128CbcEncryptFinal(
        tsip_aes_handle_t *handle,
        uint8_t *cipher,
        uint32_t *cipher_length
        )
```
(2)
```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_Aes256CbcEncryptFinal(
        tsip_aes_handle_t *handle,
        uint8_t *cipher,
        uint32_t *cipher_length
        )
```

Parameters

handle	Input	AES handler (work area)
cipher	Output	Ciphertext data area (Nothing is ever written here.)
cipher_length	Output	Ciphertext data length (The write value is always 0.)

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_FAIL:	Occurrence of internal error
TSIP_ERR_PARAMETER:	Invalid handle input
TSIP_ERR_PROHIBIT_FUNCTION:	Invalid function called

Description

Using the handle specified by the first parameter, handle, the R_TSIP_AesXXXCbcEncryptFinal() function writes the calculation result to the second parameter, cipher, and writes the length of the calculation result to the third parameter, cipher_length. The original intent was for any portion of the encrypted result that was not a multiple of 16 bytes to be written to the second parameter. However, due to the restriction that only multiples of 16 can be input to the R_TSIP_AesXXXCbcEncryptUpdate() function, nothing is ever written to cipher, and 0 is always written to cipher_length. The parameters cipher and cipher_length are provided for future compatibility in anticipation of this restriction eventually being removed.

Reentrancy

Not supported.

4.2.3.13 R_TSIP_Aes*XXX*CbcDecryptInit

Format

(1)
```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_Aes128CbcDecryptInit(
        tsip_aes_handle_t *handle,
        tsip_aes_key_index_t *key_index,
        uint8_t *ivec
)
```
(2)
```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_Aes256CbcDecryptInit(
        tsip_aes_handle_t *handle,
        tsip_aes_key_index_t *key_index,
        uint8_t *ivec
)
```

Parameters

handle	Output	AES handler (work area)
key_index	Input	Wrapped key
ivec	Input	Initialization vector (16 bytes)

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_FAIL:	Occurrence of internal error (Only for TSIP)
TSIP_ERR_RESOURCE_CONFLICT:	Occurrence of resource conflict because a hardware resource needed by the processing routine was in use by another processing routine
TSIP_ERR_KEY_SET:	Invalid wrapped key input

Description

The R_TSIP_AesXXXCbcDecryptInit() function performs preparations for the execution of AES calculation, and writes the result to the first parameter, handle. The parameter handle is used subsequently as a parameter by the R_TSIP_AesXXXCbcDecryptUpdate() and R_TSIP_AesXXXCbcDecryptFinal() functions.

When using the TLS cooperation function, input client_crypto_key_index or server_crypto_key_index, generated by R_TSIP_TlsGenerateSessionKey(), as key_index.

Refer to 3.7.1, Key Injection and Updating, for an explanation of how to generate key_index.

Reentrancy

Not supported.

4.2.3.14 R_TSIP_Aes*XXX*CbcDecryptUpdate

Format

(1) #include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_Aes128CbcDecryptUpdate(
tsip_aes_handle_t *handle,
uint8_t *cipher,
uint8_t *plain,
uint32_t cipher_length
)

(2) #include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_Aes256CbcDecryptUpdate(
tsip_aes_handle_t *handle,
uint8_t *cipher,
uint8_t *plain,
uint32_t cipher_length
)

Parameters

handle	Input/output	AES handler (work area)
cipher	Input	Ciphertext data area
plain	Output	Plaintext data area
cipher_length	Input	Byte length of ciphertext data (Must be a multiple of 16.)

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_PARAMETER:	Invalid handle input
TSIP_ERR_PROHIBIT_FUNCTION:	Invalid function called

Description

Using the handle specified by the first parameter, handle, the R_TSIP_AesXXXCbcDecryptUpdate() function decrypts the second parameter, cipher, utilizing the key index specified by the R_TSIP_AesXXXCbcDecryptInit() function, and writes the decrypted result to the third parameter, plain. After ciphertext input completes, call R_TSIP_AesXXXCbcDecryptFinal().

Except in cases where the addresses are the same, specify areas for plain and cipher that do not overlap.

Reentrancy

Not supported.

4.2.3.15 R_TSIP_Aes*XXX*CbcDecryptFinal

Format

(1) #include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_Aes128CbcDecryptFinal(
tsip_aes_handle_t *handle,
uint8_t *plain,
uint32_t *plain_length
)

(2) #include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_Aes256CbcDecryptFinal(
tsip_aes_handle_t *handle,
uint8_t *plain,
uint32_t *plain_length
)

Parameters

handle	Input	AES handler (work area)
plain	Output	Plaintext data area (Nothing is ever written here.)
plain_length	Output	Plaintext data length (The write value is always 0.)

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_FAIL:	Occurrence of internal error
TSIP_ERR_PARAMETER:	Invalid handle input
TSIP_ERR_PROHIBIT_FUNCTION:	Invalid function called

Description

Using the handle specified by the first parameter, handle, the R_TSIP_AesXXXCbcDecryptFinal() function writes the calculation result to the second parameter, plain, and writes the length of the calculation result to the third parameter, plain_length. The original intent was for any portion of the decrypted result that was not a multiple of 16 bytes to be written to the second parameter. However, due to the restriction that only multiples of 16 can be input to the R_TSIP_AesXXXCbcDecryptUpdate() function, nothing is ever written to plain, and 0 is always written to plain_length. The parameters plain and plain_length are provided for future compatibility in anticipation of this restriction eventually being removed.

Reentrancy

Not supported.

4.2.3.16 R_TSIP_Aes*XXX*CtrInit

Format

(1)
```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_Aes128CtrInit(
        tsip_aes_handle_t *handle,
        tsip_aes_key_index_t *key_index,
        uint8_t *ictr
)
```

(2)
```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_Aes256CtrInit(
        tsip_aes_handle_t *handle,
        tsip_aes_key_index_t *key_index,
        uint8_t *ictr
)
```

Parameters

handle	Output	AES handler (work area)
key_index	Input	Wrapped key
ictr	Input	Initial counter (16 bytes)

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_FAIL:	Occurrence of internal error (Only for TSIP)
TSIP_ERR_RESOURCE_CONFLICT:	Occurrence of resource conflict because a hardware resource needed by the processing routine was in use by another processing routine
TSIP_ERR_KEY_SET:	Invalid wrapped key input

Description

This function performs preparations for the execution of an AES calculation and writes the result to the parameter handle. The parameter handle is used subsequently as a parameter by the R_TSIP_AesXXXCtrUpdate() and R_TSIP_AesXXXCtrFinal() functions.

Refer to 3.7.1, Key Injection and Updating, for an explanation of how to generate key_index.

Reentrancy

Not supported.

4.2.3.17 R_TSIP_Aes*XXX*CtrUpdate

Format

(1)
```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_Aes128CtrUpdate(
        tsip_aes_handle_t *handle,
        uint8_t *itext,
        uint8_t *otext,
        uint32_t itext_length
        )
```

(2)
```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_Aes256CtrUpdate(
        tsip_aes_handle_t *handle,
        uint8_t *itext,
        uint8_t *otext,
        uint32_t itext_length
        )
```

Parameters

handle	Input/output	AES handler (work area)
itext	Input	Input text (plaintext or ciphertext) data area
otext	Output	Output text (ciphertext or plaintext) data area
itext_length	Input	Byte length of input text data (Must be a multiple of 16.)

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_PARAMETER:	Invalid handle input
TSIP_ERR_PROHIBIT_FUNCTION:	Invalid function called

Description

Using the handle specified by the first parameter, handle, this function encrypts the second parameter, itext, utilizing key_index specified by the R_TSIP_AesXXXCtrInit() function, and writes the result to the third parameter, otext. After input of the final bock completes, call R_TSIP_AesXXXCtrFinal(). If the length of the last block is 1 to 127 bits, allocate areas in 16-byte units for itext and otext, and set an arbitrary value for the fractional remainder area of itext. In this case, ignore the value stored in the fractional remainder area of otext.

Except in cases where the addresses are the same, specify areas for itext and otext that do not overlap.

Reentrancy

Not supported.

4.2.3.18 R_TSIP_Aes*XXX*CtrFinal

Format

(1) #include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_Aes128CtrFinal(
tsip_aes_handle_t *handle
)

(2) #include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_Aes256CtrFinal(
tsip_aes_handle_t *handle
)

Parameters

handle	Input	AES handler (work area)

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_FAIL:	Occurrence of internal error
TSIP_ERR_PARAMETER:	Invalid handle input
TSIP_ERR_PROHIBIT_FUNCTION:	Invalid function called

Description

Using the handle specified by the first parameter, handle, this function completes the calculation.

Reentrancy

Not supported.

4.2.3.19 R_TSIP_Aes*XXX*GcmEncryptInit

Format

(1)
```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_Aes128GcmEncryptInit(
        tsip_gcm_handle_t *handle,
        tsip_aes_key_index_t *key_index,
        uint8_t *ivec,
        uint32_t ivec_len
)
```

(2)
```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_Aes256GcmEncryptInit(
        tsip_gcm_handle_t *handle,
        tsip_aes_key_index_t *key_index,
        uint8_t *ivec,
        uint32_t ivec_len
)
```

Parameters

handle	Output	AES-GCM handler (work area)
key_index	Input	Wrapped key
ivec	Input	Initialization vector area (iv_len bytes)*1
ivec_len	Input	Initialization vector length (1 or more bytes)

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_FAIL:	Occurrence of internal error
TSIP_ERR_RESOURCE_CONFLICT:	Occurrence of resource conflict because a hardware resource needed by the processing routine was in use by another processing routine
TSIP_ERR_KEY_SET:	Invalid wrapped key input
TSIP_ERR_PARAMETER:	Invalid input data

Description

The R_TSIP_AesXXXGcmEncryptInit() function performs preparations for the execution of GCM calculation and writes the result to the first parameter, handle. The parameter handle is used subsequently as a parameter by the R_TSIP_AesXXX1GcmEncryptUpdate() and R_TSIP_AesXXXGcmEncryptFinal() functions.

Note: 1. When key_index->type is TSIP_KEY_INDEX_TYPE_AES128_FOR_TLS
The key_index value generated by the R_TSIP_TlsGenerateSessionKey() function includes a 96-bit IV when a value of 6 or 7 has been specified for select_cipher. In this case, input a null pointer as the third parameter, ivec, and specify 0 as the fourth parameter, ivec_len.

Refer to 3.7.1, Key Injection and Updating, for an explanation of how to generate key_index.

Reentrancy

Not supported.

4.2.3.20 R_TSIP_Aes*XXX*GcmEncryptUpdate

Format

(1)
```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_Aes128GcmEncryptUpdate(
        tsip_gcm_handle_t *handle,
        uint8_t *plain,
        uint8_t *cipher,
        uint32_t plain_data_len,
        uint8_t *aad,
        uint32_t aad_len
)
```
(2)
```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_Aes256GcmEncryptUpdate(
        tsip_gcm_handle_t *handle,
        uint8_t *plain,
        uint8_t *cipher,
        uint32_t plain_data_len,
        uint8_t *aad,
        uint32_t aad_len
)
```

Parameters

handle	Input/output	AES handler (work area)
plain	Input	Plaintext data area
cipher	Output	Ciphertext data area
plain_data_len	Input	Byte length of plaintext data (Must be a multiple of 16.)
aad	Input	AAD (aad_len bytes)
aad_len	Input	AAD length (0 or more bytes)

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_PARAMETER:	Invalid handle input
TSIP_ERR_PROHIBIT_FUNCTION:	Invalid function called

Description

The R_TSIP_Aes128GcmEncryptUpdate() function encrypts the plaintext specified by the second parameter, plain, in GCM mode using the values specified for key_index and ivec in R_TSIP_Aes128GcmEncryptInit() and the value specified by the fifth parameter, aad. The function internally buffers the data input by the user until the input values of aad and plain exceed 16 bytes. Once the input data from plain reaches 16 bytes or more, the encrypted result is output to the area specified by the third parameter, cipher. The lengths of the plain and aad data to be input are specified by the fourth parameter, plain_data_len, and the sixth parameter, aad_len, respectively. For these, specify not the total byte count for the aad and plain input data, but rather the data length to be input when the user calls this function. If the input values of plain and aad are not divisible by 16 bytes, the function performs padding internally. First process the data to be input as aad, and then the data to be input as plain. If aad data is input after starting to input plain data, a root error occurs. If aad data and plain data are input to the function at the same time, the aad data is processed, and then the function transitions to the plain data input state. Except in cases where the addresses are the same, specify areas for plain and cipher that do not overlap.

Reentrancy

Not supported.

4.2.3.21 R_TSIP_Aes*XXX*GcmEncryptFinal

Format

(1)
```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_Aes128GcmEncryptFinal(
        tsip_gcm_handle_t *handle,
        uint8_t *cipher,
        uint32_t *cipher_data_len,
        uint8_t *atag
)
```

(2)
```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_Aes256GcmEncryptFinal(
        tsip_gcm_handle_t *handle,
        uint8_t *cipher,
        uint32_t *cipher_data_len,
        uint8_t *atag
)
```

Parameters

handle	Input	AES handler (work area)
cipher	Output	Ciphertext data area (Nothing is ever written here.)
cipher_data_len	Output	Ciphertext data length (The write value is always 0.)
atag	Output	Authentication tag area (16 bytes)

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_FAIL:	Occurrence of internal error
TSIP_ERR_PARAMETER:	Invalid handle input
TSIP_ERR_PROHIBIT_FUNCTION:	Invalid function called

Description

If there is 16-byte fractional remainder data indicated by the total data length of the value of plain input to R_TSIP_AesXXXGcmEncryptUpdate(), the R_TSIP_AesXXXGcmEncryptFinal() function outputs the result of encrypting the fractional remainder data to the area specified by the second parameter, cipher. In this case, the portion short of 16 bytes is padded with zeros. The authentication tag is output as the fourth parameter, atag.

Reentrancy

Not supported.

4.2.3.22 R_TSIP_Aes*XXX*GcmDecryptInit

Format

(1)
```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_Aes128GcmDecryptInit(
        tsip_gcm_handle_t *handle,
        tsip_aes_key_index_t *key_index,
        uint8_t *ivec,
        uint32_t ivec_len
)
```

(2)
```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_Aes256GcmDecryptInit(
        tsip_gcm_handle_t *handle,
        tsip_aes_key_index_t *key_index,
        uint8_t *ivec,
        uint32_t ivec_len
)
```

Parameters

handle	Output	AES handler (work area)
key_index	Input	Wrapped key
ivec	Input	Initialization vector area (iv_len bytes)*1
ivec_len	Input	Initialization vector length (1 or more bytes)

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_FAIL:	Occurrence of internal error
TSIP_ERR_RESOURCE_CONFLICT:	Occurrence of resource conflict because a hardware resource needed by the processing routine was in use by another processing routine
TSIP_ERR_KEY_SET:	Invalid wrapped key input
TSIP_ERR_PARAMETER:	Invalid input data

Description

The R_TSIP_AesXXXGcmDecryptInit() function performs preparations for the execution of GCM calculation and writes the result to the first parameter, handle. The parameter handle is used subsequently as a parameter by the R_TSIP_AesXXXGcmDecryptUpdate() and R_TSIP_AesXXXGcmDecryptFinal() functions.

Note: 1. When key_index->type is TSIP_KEY_INDEX_TYPE_AES128_FOR_TLS
The key_index value generated by the R_TSIP_TlsGenerateSessionKey() function includes a 96-bit IV when a value of 6 or 7 has been specified for select_cipher. In this case, input a null pointer as the third parameter, ivec, and specify 0 as the fourth parameter, ivec_len.

Refer to 3.7.1, Key Injection and Updating, for an explanation of how to generate key_index.

Reentrancy

Not supported.

4.2.3.23 R_TSIP_Aes*XXX*GcmDecryptUpdate

Format

(1) #include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_Aes128GcmDecryptUpdate(
tsip_gcm_handle_t *handle,
uint8_t *cipher,
uint8_t *plain,
uint32_t cipher_data_len,
uint8_t *aad,
uint32_t aad_len
)

(2) #include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_Aes256GcmDecryptUpdate(
tsip_gcm_handle_t *handle,
uint8_t *cipher,
uint8_t *plain,
uint32_t cipher_data_len,
uint8_t *aad,
uint32_t aad_len
)

Parameters

handle	Input/output	AES-GCM handler (work area)
cipher	Input	Ciphertext data area
plain	Output	Plaintext data area
cipher_data_len	Input	Ciphertext data length (0 or more bytes)
aad	Input	AAD (aad_len bytes)
aad_len	Input	AAD length (0 or more bytes)

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_PARAMETER:	Invalid handle input
TSIP_ERR_PROHIBIT_FUNCTION:	Invalid function called

Description

The R_TSIP_AesXXXGcmDecryptUpdate() function decrypts the ciphertext specified by the second parameter, cipher, in GCM mode using the values specified for key_index and ivec in R_TSIP_AesXXXGcmDecryptInit() and the value specified by the fifth parameter, aad. The function internally buffers the data input by the user until the input values of aad and cipher exceed 16 bytes. Once the input data from cypher reaches 16 bytes or more, the decrypted result is output to the area specified by the third parameter, plain. The lengths of the cipher and aad data to be input are specified by the fourth parameter, cipher_data_len, and the sixth parameter, aad_len, respectively. For these, specify not the total byte count for the aad and cipher input data, but rather the data length to be input when the user calls this function. If the input values of cipher and aad are not divisible by 16 bytes, the function performs padding internally. Inside of this API, the state is transited from aad data input state to cipher data input state. Although the aad data and cipher data can be input simultaneously, the last data of aad data must be input when cipher data is input. Except in cases where the addresses are the same, specify areas for plain and cipher that do not overlap.

Reentrancy.

Not supported.

4.2.3.24 R_TSIP_Aes*XXX*GcmDecryptFinal

Format

(1)
```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_Aes128GcmDecryptFinal(
        tsip_gcm_handle_t *handle,
        uint8_t *plain,
        uint32_t *plain_data_len,
        uint8_t *atag,
        uint32_t atag_len
)
```
(2)
```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_Aes256GcmDecryptFinal(
        tsip_gcm_handle_t *handle,
        uint8_t *plain,
        uint32_t *plain_data_len,
        uint8_t *atag,
        uint32_t atag_len
)
```

Parameters

handle	Input	AES-GCM handler (work area)
plain	Output	Plaintext data area (data_len bytes)
plain_data_len	Output	Plaintext data length (0 or more bytes)
atag	Input	Authentication tag area (atag_len bytes)
atag_len	Input	Authentication tag length (4, 8, 12, 13, 14, 15, or 16 bytes)

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_FAIL:	Occurrence of internal error
TSIP_ERR_AUTHENTICATION:	Authentication error
TSIP_ERR_PROHIBIT_FUNCTION:	Invalid function called
TSIP_ERR_PARAMETER:	Invalid input data

Description

The R_TSIP_AesXXXGcmDecryptFinal() function decrypts, in GCM mode, the fractional remainder of the ciphertext specified by R_TSIP_AesXXXGcmDecryptUpdate() that does not reach 16 bytes, and then terminates GCM decryption functionality. The decrypted data and authentication tag are output to the area specified by the second parameter, plain, and the area specified as the fourth parameter, atag, respectively. The total data length of the decrypted data is output to the third parameter, plain_data_len. If authentication fails, a value of TSIP_ERR_AUTHENTICATION is returned. For the fourth parameter, atag, input 16 bytes or less. If the data input is less than 16 bytes, it is padded with zeros by the function internally.

Reentrancy

Not supported.

4.2.3.25 R_TSIP_Aes*XXX*CcmEncryptInit

Format

(1)
```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_Aes128CcmEncryptInit(
        tsip_ccm_handle_t *handle,
        tsip_aes_key_index_t *key_index,
        uint8_t *nonce,
        uint32_t nonce_len,
        uint8_t *adata,
        uint8_t a_len,
        uint32_t payload_len,
        uint32_t mac_len
)
```

(2)
```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_Aes256CcmEncryptInit(
        tsip_ccm_handle_t *handle,
        tsip_aes_key_index_t *key_index,
        uint8_t *nonce,
        uint32_t nonce_len,
        uint8_t *adata,
        uint8_t a_len,
        uint32_t payload_len,
        uint32_t mac_len
)
```

Parameters

handle	Output	AES-CCM handler (work area)
key_index	Input	Wrapped key
nonce	Input	Nonce
nonce_len	Input	Nonce data length (7 to 13 bytes)
adata	Input	AAD
a_len	Input	AAD length (0 to 110 bytes)
payload_len	Input	Payload length (any number of bytes)
mac_len	Input	MAC length (4, 6, 8, 10, 12, 14, or 16 bytes)

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_FAIL:	Occurrence of internal error
TSIP_ERR_RESOURCE_CONFLICT:	Occurrence of resource conflict because a hardware resource needed by the processing routine was in use by another processing routine
TSIP_ERR_KEY_SET:	Invalid wrapped key input

Description

R_TSIP_AesXXXCcmEncryptInit() function performs preparations for the execution of CCM calculation and writes the result to the first parameter, handle. The parameter handle is used subsequently as a parameter by the R_TSIP_AesXXXCcmEncryptUpdate() and R_TSIP_AesXXXCcmEncryptFinal() functions.

Refer to 3.7.1, Key Injection and Updating, for an explanation of how to generate key_index.

Reentrancy

Not supported.

4.2.3.26 R_TSIP_Aes*XXX*CcmEncryptUpdate

Format

(1)
```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_Aes128CcmEncryptUpdate(
        tsip_ccm_handle_t *handle,
        uint8_t *plain,
        uint8_t *cipher,
        uint32_t plain_length
)
```

(2)
```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_Aes256CcmEncryptUpdate(
        tsip_ccm_handle_t *handle,
        uint8_t *plain,
        uint8_t *cipher,
        uint32_t plain_length
)
```

Parameters

handle	Input/output	AES handler (work area)
plain	Input	Plaintext data area
cipher	Output	Ciphertext data area
plain_length	Input	Plaintext data length

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_PARAMETER:	Invalid handle input
TSIP_ERR_PROHIBIT_FUNCTION:	Invalid function called

Description

The R_TSIP_AesXXXCcmEncryptUpdate() function encrypts the plaintext specified in the second argument, plain, in CCM mode using the values specified by key_index, nonce, and adata in R_TSIP_AesXXXCcmEncryptInit(). This function buffers internally the data input by the user until the input value of plain exceeds 16 bytes. Once the amount of plain input data is 16 bytes or greater, the encrypted result is output to cipher, which is specified in the third argument. Use payload_len in R_TSIP_AesXXXCcmEncryptInit() to specify the total data length of plain that will be input. Use plain_length in this function to specify the data length to be input when the user calls this function. If the input value of plain is less than 16 bytes, the function performs padding internally. Except in cases where the addresses are the same, specify areas for plain and cipher that do not overlap.

Reentrancy

Not supported.

4.2.3.27 R_TSIP_Aes*XXX*CcmEncryptFinal

Format

(1)
```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_Aes128CcmEncryptFinal(
        tsip_ccm_handle_t *handle,
        uint8_t *cipher,
        uint32_t *cipher_length,
        uint8_t *mac,
        uint32_t mac_length
)
```
(2)
```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_Aes256CcmEncryptFinal(
        tsip_ccm_handle_t *handle,
        uint8_t *cipher,
        uint32_t *cipher_length,
        uint8_t *mac,
        uint32_t mac_length
)
```

Parameters

handle	Input	AES handler (work area)
cipher	Output	Ciphertext data area (Nothing is ever written here.)
cipher_length	Output	Ciphertext data length (The write value is always 0.)
mac	Output	MAC area
mac_length	Input	MAC length (4, 6, 8, 10, 12, 14, or 16 bytes)

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_FAIL:	Occurrence of internal error
TSIP_ERR_PARAMETER:	Invalid handle input
TSIP_ERR_PROHIBIT_FUNCTION:	Invalid function called

Description

If there is 16-byte fractional remainder data indicated by the total data length of the value of plain input to R_TSIP_AesXXXCcmEncryptUpdate(), the R_TSIP_AesXXXCcmEncryptFinal() function outputs the result of encrypting the fractional remainder data to the area specified by the second parameter, cipher. The MAC value is output to the fourth parameter, mac. Set the fifth parameter, mac_length, to the same value as that specified for the parameter mac_length in R_TSIP_AesXXXCcmEncryptInit().

Reentrancy

Not supported.

4.2.3.28 R_TSIP_Aes*XXX*CcmDecryptInit

Format

(1)
```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_Aes128CcmDecryptInit(
        tsip_ccm_handle_t *handle,
        tsip_aes_key_index_t *key_index,
        uint8_t *nonce,
        uint32_t nonce_len,
        uint8_t *adata,
        uint8_t a_len,
        uint32_t payload_len,
        uint32_t mac_len
)
```

(2)
```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_Aes128CcmDecryptInit(
        tsip_ccm_handle_t *handle,
        tsip_aes_key_index_t *key_index,
        uint8_t *nonce,
        uint32_t nonce_len,
        uint8_t *adata,
        uint8_t a_len,
        uint32_t payload_len,
        uint32_t mac_len
)
```

Parameters

handle	Input	AES-CCM handler (work area)
key_index	Input	Wrapped key
nonce	Input	Nonce
nonce_len	Input	Nonce data length (7 to 13 bytes)
adata	Input	AAD
a_len	Input	AAD length (0 to 110 bytes)
payload_len	Input	Payload length (any number of bytes)
mac_len	Input	MAC length (4, 6, 8, 10, 12, 14, or 16 bytes)

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_FAIL:	Occurrence of internal error
TSIP_ERR_RESOURCE_CONFLICT:	Occurrence of resource conflict because a hardware resource needed by the processing routine was in use by another processing routine
TSIP_ERR_KEY_SET:	Invalid wrapped key input

Description

R_TSIP_AesXXXCcmDecryptInit() function performs preparations for the execution of CCM calculation and writes the result to the first parameter, handle. The parameter handle is used subsequently as a parameter by the R_TSIP_AesXXXCcmDecryptUpdate() and R_TSIP_AesXXXCcmDecryptFinal() functions.

Refer to 3.7.1, Key Injection and Updating, for an explanation of how to generate key_index.

Reentrancy

Not supported.

4.2.3.29 R_TSIP_Aes*XXX*CcmDecryptUpdate

Format

(1)
```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_Aes128CcmDecryptUpdate(
        tsip_ccm_handle_t *handle,
        uint8_t *cipher,
        uint8_t *plain,
        uint32_t cipher_length
)
```
(2)
```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_Aes256CcmDecryptUpdate(
        tsip_ccm_handle_t *handle,
        uint8_t *cipher,
        uint8_t *plain,
        uint32_t cipher_length
)
```

Parameters

handle	Input/output	AES-CCM handler (work area)
cipher	Input	Ciphertext data area
plain	Output	Plaintext data area
cipher_length	Input	Ciphertext data length

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_PARAMETER:	Invalid handle input
TSIP_ERR_PROHIBIT_FUNCTION:	Invalid function called

Description

The R_TSIP_AesXXXCcmDecryptUpdate() function decrypts the ciphertext specified by the second parameter, cipher, in CCM mode using the values specified for key_index, nonce, and adata in R_TSIP_AesXXXCcmDecryptInit(). The function internally buffers the data input by the user until the input value of cipher exceeds 16 bytes. Once the input data from cypher reaches 16 bytes or more, the decrypted result is output to the area specified by the third parameter, plain. Specify the total data length of the cipher data to be input in the payload_len parameter of R_TSIP_AesXXXCcmDecryptInit(). For the cipher_length parameter of this function, specify the data length to be input by the user when the function is called. If the input value of cipher is not divisible by 16 bytes, the function performs padding internally. Except in cases where the addresses are the same, specify areas for plain and cipher that do not overlap.

Reentrancy

Not supported.

4.2.3.30 R_TSIP_Aes*XXX*CcmDecryptFinal

Format

(1)
```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_Aes128CcmDecryptFinal(
        tsip_ccm_handle_t *handle,
        uint8_t *plain,
        uint32_t *plain_length,
        uint8_t *mac,
        uint32_t mac_length
        )
```

(2)
```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_Aes256CcmDecryptFinal(
        tsip_ccm_handle_t *handle,
        uint8_t *plain,
        uint32_t *plain_length,
        uint8_t *mac,
        uint32_t mac_length
        )
```

Parameters

handle	Input	AES-GCM handler (work area)
plain	Output	Plaintext data area
plain_length	Output	Plaintext data length
mac	Input	MAC area
mac_length	Input	MAC length (4, 6, 8, 10, 12, 14, or 16 bytes)

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_FAIL:	Occurrence of internal error
TSIP_ERR_PARAMETER:	Invalid handle input
TSIP_ERR_PROHIBIT_FUNCTION:	Invalid function called

Description

If the data length of cipher input in R_TSIP_AesXXXCcmDecryptUpdate() results in a fractional remainder after 16 bytes, the R_TSIP_AesXXXCcmDecryptFinal() function outputs the leftover decrypted data to the second parameter, cipher. In addition, the function verifies the fourth parameter, mac. Set the fifth parameter, mac_length, to the same value as that specified for the parameter mac_length in R_TSIP_AesXXXCcmDecryptInit().

Reentrancy

Not supported.

4.2.3.31 R_TSIP_Aes*XXX*CmacGenerateInit

Format

(1) #include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_Aes128CmacGenerateInit(
tsip_cmac_handle_t *handle,
tsip_aes_key_index_t *key_index
)

(2) #include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_Aes256CmacGenerateInit(
tsip_cmac_handle_t *handle,
tsip_aes_key_index_t *key_index
)

Parameters

handle	Output	AES-CMAC handler (work area)
key_index	Input	Wrapped key

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_FAIL:	Occurrence of internal error
TSIP_ERR_RESOURCE_CONFLICT:	Occurrence of resource conflict because a hardware resource needed by the processing routine was in use by another processing routine
TSIP_ERR_KEY_SET:	Invalid wrapped key input

Description

The R_TSIP_AesXXXCmacGenerateInit() function performs preparations for the execution of CMAC calculation, and writes the result to the first parameter, handle. The parameter handle is used subsequently as a parameter by the R_TSIP_AesXXXCmacGenerateUpdate() and R_TSIP_AesXXXCmacGenerateFinal() functions.

Refer to 3.7.1, Key Injection and Updating, for an explanation of how to generate key_index.

Reentrancy

Not supported.

4.2.3.32 R_TSIP_Aes*XXX*CmacGenerateUpdate

Format

(1) #include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_Aes128CmacGenerateUpdate(
tsip_cmac_handle_t *handle,
uint8_t *message,
uint32_t message_length
)

(2) #include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_Aes256CmacGenerateUpdate(
tsip_cmac_handle_t *handle,
uint8_t *message,
uint32_t message_length
)

Parameters

handle	Input/output	AES-CMAC handler (work area)
message	Input	Message data area (message_length bytes)
message_length	Input	Message data length (0 or more bytes)

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_PARAMETER:	Invalid handle input
TSIP_ERR_PROHIBIT_FUNCTION:	Invalid function called

Description

The R_TSIP_AesXXXCmacGenerateUpdate() function generates a MAC value from the message specified as the second parameter, message, using the value specified for key_index in R_TSIP_AesXXXCmacGenerateInit(). The function internally buffers the data input by the user until the input value of message exceeds 16 bytes. The length of the message data to be input is specified by the third parameter, message_len. For this, specify not the total byte count for the message input data, but rather the message data length to be input when the user calls this function. If the input value, message, is not a multiple of 16 bytes, it is padded with zeros by the function internally.

Reentrancy

Not supported.

4.2.3.33 R_TSIP_Aes*XXX*CmacGenerateFinal

Format

(1)
```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_Aes128CmacGenerateFinal(
        tsip_cmac_handle_t *handle,
        uint8_t *mac
)
```
(2)
```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_Aes256CmacGenerateFinal(
        tsip_cmac_handle_t *handle,
        uint8_t *mac
)
```

Parameters

handle	Input	AES-CMAC handler (work area)
mac	Output	MAC data area (16 bytes)

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_FAIL:	Occurrence of internal error
TSIP_ERR_PARAMETER:	Invalid handle input
TSIP_ERR_PROHIBIT_FUNCTION:	Invalid function called

Description

The R_TSIP_AesXXXCmacGenerateFinal() function outputs the MAC value to the MAC data area specified by the second parameter, mac, and then stops CMAC operation.

Reentrancy

Not supported.

4.2.3.34 R_TSIP_Aes*XXX*CmacVerifyInit

Format

(1)
```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_Aes128CmacVerifyInit(
        tsip_cmac_handle_t *handle,
        tsip_aes_key_index_t *key_index
)
```
(2)
```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_Aes256CmacVerifyInit(
        tsip_cmac_handle_t *handle,
        tsip_aes_key_index_t *key_index
)
```

Parameters

handle	Output	AES-CMAC handler (work area)
key_index	Input	Wrapped key

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_FAIL:	Occurrence of internal error
TSIP_ERR_RESOURCE_CONFLICT:	Occurrence of resource conflict because a hardware resource needed by the processing routine was in use by another processing routine
TSIP_ERR_KEY_SET:	Invalid wrapped key

Description

The R_TSIP_AesXXXCmacVerifyInit() function performs preparations for the execution of CMAC calculation, and writes the result to the first parameter, handle. The parameter handle is used subsequently as a parameter by the R_TSIP_AesXXXCmacVerifyUpdate() and R_TSIP_AesXXXCmacVerifyFinal() functions.

Refer to 3.7.1, Key Injection and Updating, for an explanation of how to generate key_index.

Reentrancy

Not supported.

4.2.3.35 R_TSIP_Aes*XXX*CmacVerifyUpdate

Format

(1)
```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_Aes128CmacVerifyUpdate(
        tsip_cmac_handle_t *handle,
        uint8_t *message,
        uint32_t message_length
)
```
(2)
```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_Aes256CmacVerifyUpdate(
        tsip_cmac_handle_t *handle,
        uint8_t *message,
        uint32_t message_length
)
```

Parameters

handle	Input/output	AES-CMAC handler (work area)
message	Input	Message data area (message_length bytes)
message_length	Input	Message data length (0 or more bytes)

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_PARAMETER:	Invalid handle input
TSIP_ERR_PROHIBIT_FUNCTION:	Invalid function called

Description

The R_TSIP_AesXXXCmacVerifyUpdate() function generates a MAC value from the message specified as the second parameter, message, using the value specified for key_index in R_TSIP_AesXXXCmacVerifyInit(). The function internally buffers the data input by the user until the input value of message exceeds 16 bytes. The length of the message data to be input is specified by the third parameter, message_len. For this, specify not the total byte count for the message input data, but rather the message data length to be input when the user calls this function. If the input value, message, is not a multiple of 16 bytes, it is padded with zeros by the function internally.

Reentrancy

Not supported.

4.2.3.36 R_TSIP_Aes*XXX*CmacVerifyFinal

Format

(1)
```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_Aes128CmacVerifyFinal(
        tsip_cmac_handle_t *handle,
        uint8_t *mac,
        uint32_t mac_length
)
```

(2)
```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_Aes256CmacVerifyFinal(
        tsip_cmac_handle_t *handle,
        uint8_t *mac,
        uint32_t mac_length
)
```

Parameters

handle	Input	AES-CMAC handler (work area)
mac	Input	MAC data area (16 bytes)
mac_length	Input	MAC data length (2 to 16 bytes)

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_FAIL:	Occurrence of internal error
TSIP_ERR_AUTHENTICATION:	Authentication failure
TSIP_ERR_PARAMETER:	Invalid handle input
TSIP_ERR_PROHIBIT_FUNCTION:	Invalid function called

Description

The R_TSIP_AesXXXCmacVerifyFinal() function inputs the MAC value to the data area specified by the second parameter, mac, and verifies the MAC value. If authentication fails, a value of TSIP_ERR_AUTHENTICATION is returned. If the MAC value is less than 16 bytes, it is padded with zeros by the function internally.

Reentrancy

Not supported.

4.2.4 DES

4.2.4.1 R_TSIP_GenerateTdesKeyIndex

Format

```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_GenerateTdesKeyIndex(
        uint8_t *encrypted_provisioning_key,
        uint8_t *iv,
        uint8_t *encrypted_key,
        tsip_tdes_key_index_t *key_index
)
```

Parameters

encrypted_provisioning_key	Input	W-UFPK
iv	Input	Initialization vector when generating encrypted_key
encrypted_key	Input	Encrypted key encrypted using UFPK
key_index	Output	Wrapped key

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_RESOURCE_CONFLICT:	Occurrence of resource conflict because a hardware resource needed by the processing routine was in use by another processing routine
TSIP_ERR_FAIL:	Occurrence of internal error

Description

This API outputs an DES wrapped key.

For encrypted_key, input the data indicated in 7.3.2, DES, encrypted using the UFPK. Refer to 3.7.1, Key Injection and Updating, for an explanation of encrypted_key, iv, and encrypted_provisioning_key and how to use key_index.

Reentrancy

Not supported.

4.2.4.2 R_TSIP_UpdateTdesKeyIndex

Format

```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_UpdateTdesKeyIndex(
        uint8_t *iv,
        uint8_t *encrypted_key,
        tsip_tdes_key_index_t *key_index
)
```

Parameters

iv	Input	Initialization vector when generating encrypted_key
encrypted_key	Input	Encrypted key encrypted using KUK
key_index	Output	Wrapped key

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_RESOURCE_CONFLICT:	Occurrence of resource conflict because a hardware resource needed by the processing routine was in use by another processing routine
TSIP_ERR_FAIL:	Occurrence of internal error

Description

This API outputs a TDES wrapped key.

For encrypted_key, input the data indicated in 7.3.2, DES, encrypted using the KUK. Refer to 3.7.1, Key Injection and Updating, for an explanation of iv and encrypted_key and how to use key_index.

Reentrancy

Not supported.

4.2.4.3 R_TSIP_GenerateTdesRandomKeyIndex

Format

```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_GenerateTdesRandomKeyIndex(
        tsip_tdes_key_index_t *key_index
)
```

Parameters

key_index	Output	Wrapped key

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_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 outputs a TDES wrapped key.

This API generates a user key from a random number within the TSIP. Therefore, no user key needs to be input. Encrypting data using the wrapped key output by this API makes it possible to prevent dead copying of data.

Refer to 3.7.1, Key Injection and Updating, for an explanation of how to use key_index.

Reentrancy

Not supported.

4.2.4.4 R_TSIP_TdesEcbEncryptInit

Format

```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_TdesEcbEncryptInit(
        tsip_tdes_handle_t *handle,
        tsip_tdes_key_index_t *key_index
)
```

Parameters

handle	Output	TDES handler (work area)
key_index	Input	Wrapped key

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_RESOURCE_CONFLICT:	Occurrence of resource conflict because a hardware resource needed by the processing routine was in use by another processing routine
TSIP_ERR_KEY_SET:	Invalid wrapped key

Description

The R_TSIP_TdesEcbEncryptInit() function performs preparations for the execution of DES calculation and writes the result to the first parameter, handle. The parameter handle is used subsequently as a parameter by the R_TSIP_TdesEcbEncryptUpdate() and R_TSIP_TdesEcbEncryptFinal() functions.

Refer to 3.7.1, Key Injection and Updating, for an explanation of how to generate key_index.

Reentrancy

Not supported.

4.2.4.5 R_TSIP_TdesEcbEncryptUpdate

Format

```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_TdesEcbEncryptUpdate(
        tsip_tdes_handle_t *handle,
        uint8_t *plain,
        uint8_t *cipher,
        uint32_t plain_length
)
```

Parameters

handle	Input/output	TDES handler (work area)
plain	Input	Plaintext data area
cipher	Output	Ciphertext data area
plain_length	Input	Byte length of plaintext data (Must be a multiple of 8.)

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_PARAMETER:	Invalid handle input
TSIP_ERR_PROHIBIT_FUNCTION:	Invalid function called

Description

Using the handle specified by the first parameter, handle, the R_TSIP_TdesEcbEncryptUpdate() function encrypts the second parameter, plain, using the key index specified by the R_TSIP_TdesEcbEncryptInit() function, and writes the encrypted result to the third parameter, cipher. After plaintext input completes, call R_TSIP_TdesEcbEncryptFinal().

Except in cases where the addresses are the same, specify areas for plain and cipher that do not overlap.

Reentrancy

Not supported.

4.2.4.6 R_TSIP_TdesEcbEncryptFinal

Format

```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_TdesEcbEncryptFinal(
        tsip_tdes_handle_t *handle,
        uint8_t *cipher,
        uint32_t *cipher_length
)
```

Parameters

handle	Input	TDES handler (work area)
cipher	Output	Ciphertext data area (Nothing is ever written here.)
cipher_length	Output	Ciphertext data length (The write value is always 0.)

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_FAIL:	Occurrence of internal error
TSIP_ERR_PARAMETER:	Invalid handle input
TSIP_ERR_PROHIBIT_FUNCTION:	Invalid function called

Description

Using the handle specified by the first parameter, handle, the R_TSIP_TdesEcbEncryptFinal() function writes the calculation result to the second parameter, cipher, and writes the length of the calculation result to the third parameter, cipher_length. The original intent was for any portion of the encrypted result that was not a multiple of 8 bytes to be written to the second parameter. However, due to the restriction that only multiples of 8 can be input to the R_TSIP_TdesEcbEncryptUpdate() function, nothing is ever written to cipher, and 0 is always written to cipher_length. The parameters cipher and cipher_length are provided for future compatibility in anticipation of this restriction eventually being removed.

Reentrancy

Not supported.

4.2.4.7 R_TSIP_TdesEcbDecryptInit

Format

```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_TdesEcbDecryptInit(
        tsip_tdes_handle_t *handle,
        tsip_tdes_key_index_t *key_index
)
```

Parameters

handle	Output	TDES handler (work area)
key_index	Input	Wrapped key

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_KEY_SET:	Invalid wrapped key
TSIP_ERR_RESOURCE_CONFLICT:	Occurrence of resource conflict because a hardware resource needed by the processing routine was in use by another processing routine

Description

The R_TSIP_TdesEcbDecryptInit() function performs preparations for the execution of DES calculation and writes the result to the first parameter, handle. The parameter handle is used subsequently as a parameter by the R_TSIP_TdesEcbDecryptUpdate() and R_TSIP_TdesEcbDecryptFinal() functions.

Refer to 3.7.1, Key Injection and Updating, for an explanation of how to generate key_index.

Reentrancy

Not supported.

4.2.4.8 R_TSIP_TdesEcbDecryptUpdate

Format

```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_TdesEcbDecryptUpdate(
        tsip_tdes_handle_t *handle,
        uint8_t *cipher,
        uint8_t *plain,
        uint32_t cipher_length
)
```

Parameters

handle	Input/output	TDES handler (work area)
cipher	Input	Ciphertext data area
plain	Output	Plaintext data area
cipher_length	Input	Byte length of ciphertext data (Must be a multiple of 8.)

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_PARAMETER:	Invalid handle input
TSIP_ERR_PROHIBIT_FUNCTION:	Invalid function called

Description

Using the handle specified by the first parameter, handle, the R_TSIP_TdesEcbDecryptUpdate() function decrypts the second parameter, cipher, using the key index specified by the R_TSIP_TdesEcbDecryptInit() function, and writes the decrypted result to the third parameter, plain. After ciphertext input completes, call R_TSIP_TdesEcbDecryptFinal().

Except in cases where the addresses are the same, specify areas for plain and cipher that do not overlap.

Reentrancy

Not supported.

4.2.4.9 R_TSIP_TdesEcbDecryptFinal

Format

```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_TdesEcbDecryptFinal(
        tsip_tdes_handle_t *handle,
        uint8_t *plain,
        uint32_t *plain_length
)
```

Parameters

handle	Input	TDES handler (work area)
plain	Output	Plaintext data area (Nothing is ever written here.)
plain_length	Output	Plaintext data length (The write value is always 0.)

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_FAIL:	Occurrence of internal error
TSIP_ERR_PARAMETER:	Invalid handle input
TSIP_ERR_PROHIBIT_FUNCTION:	Invalid function called

Description

Using the handle specified by the first parameter, handle, the R_TSIP_TdesEcbDecryptFinal() function writes the calculation result to the second parameter, plain, and writes the length of the calculation result to the third parameter, plain_length. The original intent was for any portion of the decrypted result that was not a multiple of 8 bytes to be written to the second parameter. However, due to the restriction that only multiples of 8 can be input to the R_TSIP_TdesEcbDecryptUpdate() function, nothing is ever written to plain, and 0 is always written to plain_length. The parameters plain and plain_length are provided for future compatibility in anticipation of this restriction eventually being removed.

Reentrancy

Not supported.

4.2.4.10 R_TSIP_TdesCbcEncryptInit

Format

```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_TdesCbcEncryptInit(
        tsip_tdes_handle_t *handle,
        tsip_tdes_key_index_t *key_index,
        uint8_t *ivec
)
```

Parameters

handle	Output	TDES handler (work area)
key_index	Input	Wrapped key
ivec	Input	Initialization vector (8 bytes)

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_KEY_SET:	Invalid wrapped key input
TSIP_ERR_RESOURCE_CONFLICT:	Occurrence of resource conflict because a hardware resource needed by the processing routine was in use by another processing routine

Description

The R_TSIP_TdesCbcEncryptInit() function performs preparations for the execution of DES calculation, and writes the result to the first parameter, handle. The parameter handle is used subsequently as a parameter by the R_TSIP_TdesCbcEncryptUpdate() and R_TSIP_TdesCbcEncryptFinal() functions.

Refer to 3.7.1, Key Injection and Updating, for an explanation of how to generate key_index.

Reentrancy

Not supported.

4.2.4.11 R_TSIP_TdesCbcEncryptUpdate

Format

```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_TdesCbcEncryptUpdate(
        tsip_tdes_handle_t *handle,
        uint8_t *plain,
        uint8_t *cipher,
        uint32_t plain_length
)
```

Parameters

handle	Input/output	TDES handler (work area)
plain	Input	Plaintext data area
cipher	Output	Ciphertext data area
plain_length	Input	Byte length of plaintext data (Must be a multiple of 8.)

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_PARAMETER:	Invalid handle input
TSIP_ERR_PROHIBIT_FUNCTION:	Invalid function called

Description

Using the handle specified by the first parameter, handle, the R_TSIP_TdesCbcEncryptUpdate() function encrypts the second parameter, plain, using the key index specified by the R_TSIP_TdesCbcEncryptInit() function, and writes the encrypted result to the third parameter, cipher. After plaintext input completes, call R_TSIP_TdesCbcEncryptFinal().

Except in cases where the addresses are the same, specify areas for plain and cipher that do not overlap.

Reentrancy

Not supported.

4.2.4.12 R_TSIP_TdesCbcEncryptFinal

Format

```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_TdesCbcEncryptFinal(
        tsip_tdes_handle_t *handle,
        uint8_t *cipher,
        uint32_t *cipher_length
)
```

Parameters

handle	Input	TDES handler (work area)
cipher	Output	Ciphertext data area (Nothing is ever written here.)
cipher_length	Output	Ciphertext data length (The write value is always 0.)

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_FAIL:	Occurrence of internal error
TSIP_ERR_PARAMETER:	Invalid handle input
TSIP_ERR_PROHIBIT_FUNCTION:	Invalid function called

Description

Using the handle specified by the first parameter, handle, the R_TSIP_TdesCbcEncryptFinal() function writes the calculation result to the second parameter, cipher, and writes the length of the calculation result to the third parameter, cipher_length. The original intent was for any portion of the encrypted result that was not a multiple of 8 bytes to be written to the second parameter. However, due to the restriction that only multiples of 8 can be input to the R_TSIP_TdesCbcEncryptUpdate() function, nothing is ever written to cipher, and 0 is always written to cipher_length. The parameters cipher and cipher_length are provided for future compatibility in anticipation of this restriction eventually being removed.

Reentrancy

Not supported.

4.2.4.13 R_TSIP_TdesCbcDecryptInit

Format

```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_TdesCbcDecryptInit(
        tsip_tdes_handle_t *handle,
        tsip_tdes_key_index_t *key_index,
        uint8_t *ivec
)
```

Parameters

handle	Output	TDES handler (work area)
key_index	Input	Wrapped key
ivec	Input	Initialization vector (8 bytes)

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_KEY_SET:	Invalid wrapped key input
TSIP_ERR_RESOURCE_CONFLICT:	Occurrence of resource conflict because a hardware resource needed by the processing routine was in use by another processing routine

Description

The R_TSIP_TdesCbcDecryptInit() function performs preparations for the execution of DES calculation, and writes the result to the first parameter, handle. The parameter handle is used subsequently as a parameter by the R_TSIP_TdesCbcDecryptUpdate() and R_TSIP_TdesCbcDecryptFinal() functions.

Refer to 3.7.1, Key Injection and Updating, for an explanation of how to generate key_index.

Reentrancy

Not supported.

4.2.4.14 R_TSIP_TdesCbcDecryptUpdate

Format

```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_TdesCbcDecryptUpdate(
        tsip_tdes_handle_t *handle,
        uint8_t *cipher,
        uint8_t *plain,
        uint32_t cipher_length
)
```

Parameters

handle	Input/output	TDES handler (work area)
cipher	Input	Ciphertext data area
plain	Output	Plaintext data area
cipher_length	Input	Byte length of ciphertext data (Must be a multiple of 8.)

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_PARAMETER:	Invalid handle input
TSIP_ERR_PROHIBIT_FUNCTION:	Invalid function called

Description

Using the handle specified by the first parameter, handle, the R_TSIP_TdesCbcDecryptUpdate() function decrypts the second parameter, cipher, utilizing the key index specified by the R_TSIP_TdesCbcDecryptInit() function, and writes the decrypted result to the third parameter, plain. After ciphertext input completes, call R_TSIP_TdesCbcDecryptFinal().

Except in cases where the addresses are the same, specify areas for plain and cipher that do not overlap.

Reentrancy

Not supported.

4.2.4.15 R_TSIP_TdesCbcDecryptFinal

Format

```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_TdesCbcDecryptFinal(
        tsip_tdes_handle_t *handle,
        uint8_t *plain,
        uint32_t *plain_length
)
```

Parameters

handle	Input	TDES handler (work area)
plain	Output	Plaintext data area (Nothing is ever written here.)
plain_length	Output	Plaintext data length (The write value is always 0.)

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_FAIL:	Occurrence of internal error
TSIP_ERR_PARAMETER:	Invalid handle input
TSIP_ERR_PROHIBIT_FUNCTION:	Invalid function called

Description

Using the handle specified by the first parameter, handle, the R_TSIP_TdesCbcDecryptFinal() function writes the calculation result to the second parameter, plain, and writes the length of the calculation result to the third parameter, plain_length. The original intent was for any portion of the decrypted result that was not a multiple of 8 bytes to be written to the second parameter. However, due to the restriction that only multiples of 8 can be input to the R_TSIP_TdesCbcDecryptUpdate() function, nothing is ever written to plain, and 0 is always written to plain_length. The parameters plain and plain_length are provided for future compatibility in anticipation of this restriction eventually being removed.

Reentrancy

Not supported.

4.2.5 ARC4

4.2.5.1 R_TSIP_GenerateArc4KeyIndex

Format

```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_GenerateArc4KeyIndex(
        uint8_t *encrypted_provisioning_key,
        uint8_t *iv,
        uint8_t *encrypted_key,
        tsip_arc4_key_index_t *key_index
)
```

Parameters

encrypted_provisioning_key	Input	W-UFPK
iv	Input	Initialization vector when generating encrypted_key
encrypted_key	Input	Encrypted key encrypted using UFPK
key_index	Output	Key index

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_FAIL:	Occurrence of internal error
TSIP_ERR_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 outputs an ARC4 wrapped key.

Refer to 7.3.3, ARC4, for the format of the data encrypted using the UFPK input as encrypted_key.

Refer to 3.7.1, Key Injection and Updating, for an explanation of encrypted_key, iv, and encrypted_provisioning_key and how to use key_index.

Reentrancy

Not supported.

4.2.5.2 R_TSIP_UpdateArc4KeyIndex

Format

```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_UpdateArc4KeyIndex(
        uint8_t *iv,
        uint8_t *encrypted_key,
        tsip_arc4_key_index_t *key_index
)
```

Parameters

iv	Input	Initialization vector when generating encrypted_key
encrypted_key	Input	Encrypted key encrypted using KUK
key_index	Output	Key index

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_FAIL:	Occurrence of internal error
TSIP_ERR_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 updates an ARC4 wrapped key.

Refer to 7.3.3, ARC4, for the format of the data encrypted using the KUK input as encrypted_key.

Refer to 3.7.1, Key Injection and Updating, for an explanation of iv and encrypted_key and how to use key_index.

Reentrancy

Not supported.

4.2.5.3 R_TSIP_GenerateArc4RandomKeyIndex

Format

```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_GenerateArc4RandomKeyIndex(
        tsip_arc4_key_index_t *key_index
)
```

Parameters

key_index	Output	Wrapped key

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_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 outputs an ARC4 wrapped key.

This API generates a user key from a random number within the TSIP. Therefore, no user key needs to be input. Encrypting data using the wrapped key output by this API makes it possible to prevent dead copying of data.

Refer to 3.7.1, Key Injection and Updating, for an explanation of how to use key_index.

Reentrancy

Not supported.

4.2.5.4 R_TSIP_Arc4EncryptInit

Format

```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_Arc4EncryptInit(
        tsip_arc4_handle_t *handle,
        tsip_arc4_key_index_t *key_index
)
```

Parameters

handle	Output	ARC4 handler (work area)
key_index	Input	Wrapped key

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_FAIL:	Occurrence of internal error
TSIP_ERR_RESOURCE_CONFLICT:	Occurrence of resource conflict because a hardware resource needed by the processing routine was in use by another processing routine
TSIP_ERR_KEY_SET:	Invalid wrapped key input

Description

The R_TSIP_Arc4EncryptInit() function performs preparations for the execution of ARC4 calculation, and writes the result to the first parameter, handle. The parameter handle is used subsequently as a parameter by the R_TSIP_Arc4EncryptUpdate() and R_TSIP_Arc4EncryptFinal() functions.

Refer to 3.7.1, Key Injection and Updating, for an explanation of how to generate key_index.

Reentrancy

Not supported.

4.2.5.5 R_TSIP_Arc4EncryptUpdate

Format

```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_Arc4EncryptUpdate(
        tsip_arc4_handle_t *handle,
        uint8_t *plain,
        uint8_t *cipher,
        uint32_t plain_length
)
```

Parameters

handle	Input/output	ARC4 handler (work area)
plain	Input	Plaintext data area
cipher	Output	Ciphertext data area
plain_length	Input	Byte length of plaintext data (Must be a multiple of 16.)

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_PARAMETER:	Invalid handle input
TSIP_ERR_PROHIBIT_FUNCTION:	Invalid function called

Description

The R_TSIP_Arc4EncryptUpdate() function encrypts the second parameter, plain, using the key index specified by the R_TSIP_Arc4EncryptInit() function, and writes the encrypted result to the third parameter, cipher. After plaintext input completes, call R_TSIP_Arc4EncryptFinal().

Except in cases where the addresses are the same, specify areas for plain and cipher that do not overlap.

Reentrancy

Not supported.

4.2.5.6 R_TSIP_Arc4EncryptFinal

Format

```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_Arc4EncryptFinal(
        tsip_arc4_handle_t *handle,
        uint8_t *cipher,
        uint32_t *cipher_length
)
```

Parameters

handle	Input	ARC4 handler (work area)
cipher	Output	Ciphertext data area (Nothing is ever written here.)
cipher_length	Output	Ciphertext data length (The write value is always 0.)

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_FAIL:	Occurrence of internal error
TSIP_ERR_PARAMETER:	Invalid handle input
TSIP_ERR_PROHIBIT_FUNCTION:	Invalid function called

Description

Using the handle specified by the first parameter, handle, the R_TSIP_Arc4EncryptFinal() function writes the calculation result to the second parameter, cipher, and writes the length of the calculation result to the third parameter, cipher_length. The original intent was for any portion of the encrypted result that was not a multiple of 16 bytes to be written to the second parameter. However, due to the restriction that only multiples of 16 can be input to the R_TSIP_Arc4EncryptUpdate() function, nothing is ever written to cipher, and 0 is always written to cipher_length. The parameters cipher and cipher_length are provided for future compatibility in anticipation of this restriction eventually being removed.

Reentrancy

Not supported.

4.2.5.7 R_TSIP_Arc4DecryptInit

Format

```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_Arc4DecryptInit(
        tsip_arc4_handle_t *handle,
        tsip_arc4_key_index_t *key_index
)
```

Parameters

handle	Output	ARC4 handler (work area)
key_index	Input	Wrapped key

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_RESOURCE_CONFLICT:	Occurrence of resource conflict because a hardware resource needed by the processing routine was in use by another processing routine
TSIP_ERR_KEY_SET:	Invalid wrapped key

Description

The R_TSIP_Arc4DecryptInit() function performs preparations for the execution of ARC4 calculation, and writes the result to the first parameter, handle. The parameter handle is used subsequently as a parameter by the R_TSIP_Arc4DecryptUpdate() and R_TSIP_Arc4DecryptFinal() functions.

Refer to 3.7.1, Key Injection and Updating, for an explanation of how to generate key_index.

Reentrancy

Not supported.

4.2.5.8 R_TSIP_Arc4DecryptUpdate

Format

```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_Arc4DecryptUpdate(
        tsip_arc4_handle_t *handle,
        uint8_t *cipher,
        uint8_t *plain,
        uint32_t cipher_length
)
```

Parameters

handle	Input/output	ARC4 handler (work area)
cipher	Input	Ciphertext data area
plain	Output	Plaintext data area
cipher_length	Input	Byte length of ciphertext data (Must be a multiple of 16.)

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_PARAMETER:	Invalid handle input
TSIP_ERR_PROHIBIT_FUNCTION:	Invalid function called

Description

Using the handle specified by the first parameter, handle, the R_TSIP_Arc4DecryptUpdate() function decrypts the second parameter, cipher, utilizing the key index specified by the R_TSIP_Arc4DecryptInit() function, and writes the decrypted result to the third parameter, plain. After ciphertext input completes, call R_TSIP_Arc4DecryptFinal().

Except in cases where the addresses are the same, specify areas for plain and cipher that do not overlap.

Reentrancy

Not supported.

4.2.5.9 R_TSIP_Arc4DecryptFinal

Format

```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_Aes128EcbDecryptFinal(
        tsip_aes_handle_t *handle,
        uint8_t *plain,
        uint32_t *plain_length
)
```

Parameters

handle	Input	ARC4 handler (work area)
plain	Output	Plaintext data area (Nothing is ever written here.)
plain_length	Output	Plaintext data length (The write value is always 0.)

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_FAIL:	Occurrence of internal error
TSIP_ERR_PARAMETER:	Invalid handle input
TSIP_ERR_PROHIBIT_FUNCTION:	Invalid function called

Description

Using the handle specified by the first parameter, handle, the R_TSIP_Arc4DecryptFinal() function writes the calculation result to the second parameter, plain, and writes the length of the calculation result to the third parameter, plain_length. The original intent was for any portion of the decrypted result that was not a multiple of 16 bytes to be written to the second parameter. However, due to the restriction that only multiples of 16 can be input to the R_TSIP_Arc4DecryptUpdate() function, nothing is ever written to plain, and 0 is always written to plain_length. The parameters plain and plain_length are provided for future compatibility in anticipation of this restriction eventually being removed.

Reentrancy

Not supported.

4.2.6 RSA

4.2.6.1 R_TSIP_GenerateRsa*XXX*PublicKeyIndex

Format

(1)
```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_GenerateRsa1024PublicKeyIndex(
        uint8_t *encrypted_provisioning_key,
        uint8_t *iv,
        uint8_t *encrypted_key,
        tsip_rsa1024_public_key_index_t *key_index
)
```
(2)
```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_GenerateRsa2048PublicKeyIndex(
        uint8_t *encrypted_provisioning_key,
        uint8_t *iv,
        uint8_t *encrypted_key,
        tsip_rsa2048_public_key_index_t *key_index
)
```
(3)
```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_GenerateRsa3072PublicKeyIndex(
        uint8_t *encrypted_provisioning_key,
        uint8_t *iv,
        uint8_t *encrypted_key,
        tsip_rsa3072_public_key_index_t *key_index
)
```
(4)
```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_GenerateRsa4096PublicKeyIndex(
        uint8_t *encrypted_provisioning_key,
        uint8_t *iv,
        uint8_t *encrypted_key,
        tsip_rsa4096_public_key_index_t *key_index
)
```

Parameters

encrypted_provisioning_key	Input	W-UFPK
iv	Input	Initialization vector when generating encrypted_key
encrypted_key	Input	Encrypted key encrypted using UFPK
key_index	Output	Public key with wrapped key format
value		Public key value
key_management_info1		Key management information
key_n		Modulus n (plaintext) (1) 1024-bit RSA (2) 2048-bit RSA (3) 3072-bit RSA (4) 4096-bit RSA

key_e	Exponent e (plaintext) (1) 1024-bit RSA (2) 2048-bit RSA (3) 3072-bit RSA (4) 4096-bit RSA
dummy	Dummy
key_management_info2	Key management information

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_RESOURCE_CONFLICT:	Occurrence of resource conflict because a hardware resource needed by the processing routine was in use by another processing routine
TSIP_ERR_FAIL	Occurrence of internal error

Description

This API outputs a 1024-bit, 2048-bit, 3072-bit, or 4096-bit RSA public key with wrapped key format.

Refer to 7.3.4, RSA, for the format of the data encrypted using the UFPK input as encrypted_key.

Ensure that the areas allocated for encrypted_key and key_index do not overlap.

Refer to 3.7.1, Key Injection and Updating, for an explanation of encrypted_provisioning_key, iv, and encrypted_key and how to use key_index.

Reentrancy

Not supported.

4.2.6.2 R_TSIP_GenerateRsa*XXX*PrivateKeyIndex

Format

(1)
```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_GenerateRsa1024PrivateKeyIndex(
        uint8_t *encrypted_provisioning_key,
        uint8_t *iv,
        uint8_t *encrypted_key,
        tsip_rsa1024_private_key_index_t *key_index
)
```

(2)
```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_GenerateRsa2048PrivateKeyIndex(
        uint8_t *encrypted_provisioning_key,
        uint8_t *iv,
        uint8_t *encrypted_key,
        tsip_rsa2048_private_key_index_t *key_index
)
```

Parameters

encrypted_provisioning_key	Input	W-UFPK
iv	Input	Initialization vector when generating encrypted_key
encrypted_key	Input	Encrypted key encrypted using UFPK
key_index	Output	Wrapped key (1) 1024-bit RSA (2) 2048-bit RSA

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_RESOURCE_CONFLICT:	Occurrence of resource conflict because a hardware resource needed by the processing routine was in use by another processing routine
TSIP_ERR_FAIL	Occurrence of internal error

Description

This API outputs a 1024-bit or 2048-bit RSA secret wrapped key.

Refer to 7.3.4, RSA, for the format of the data encrypted using the UFPK input as encrypted_key.

Ensure that the areas allocated for encrypted_key and key_index do not overlap.

Refer to 3.7.1, Key Injection and Updating, for an explanation of encrypted_provisioning_key, iv, and encrypted_key and how to use key_index.

Reentrancy

Not supported.

4.2.6.3 R_TSIP_UpdateRsa*XXX*PublicKeyIndex

Format

(1)
```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_UpdateRsa1024PublicKeyIndex(
        uint8_t *iv,
        uint8_t *encrypted_key,
        tsip_rsa1024_public_key_index_t *key_index
)
```

(2)
```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_UpdateRsa2048PublicKeyIndex(
        uint8_t *iv,
        uint8_t *encrypted_key,
        tsip_rsa2048_public_key_index_t *key_index
)
```

(3)
```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_UpdateRsa3072PublicKeyIndex(
        uint8_t *iv,
        uint8_t *encrypted_key,
        tsip_rsa3072_public_key_index_t *key_index
)
```

(4)
```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_UpdateRsa4096PublicKeyIndex(
        uint8_t *iv,
        uint8_t *encrypted_key,
        tsip_rsa4096_public_key_index_t *key_index
)
```

Parameters

iv	Input	Initialization vector when generating encrypted_key
encrypted_key	Input	Encrypted key encrypted using KUK
key_index	Output	Public key with wrapped key format
value		Public key value
key_management_info1		Key management information
key_n		Modulus n (plaintext) (1) 1024-bit RSA (2) 2048-bit RSA (3) 3072-bit RSA (4) 4096-bit RSA
key_e		Exponent e (plaintext) (1) 1024-bit RSA (2) 2048-bit RSA (3) 3072-bit RSA (4) 4096-bit RSA
dummy		Dummy
key_management_info2		Key management information

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_RESOURCE_CONFLICT:	Occurrence of resource conflict because a hardware resource needed by the processing routine was in use by another processing routine
TSIP_ERR_FAIL	Occurrence of internal error

Description

This API updates a 1024-bit, 2048-bit, 3072-bit, or 4096-bit RSA publickey with wrapped key format.

Refer to 7.3.4, RSA, for the format of the data encrypted using the KUK input as encrypted_key.

Refer to 3.7.1, Key Injection and Updating, for an explanation of iv and encrypted_key and how to use key_index.

Reentrancy

Not supported.

4.2.6.4 R_TSIP_UpdateRsa*XXX*PrivateKeyIndex

Format

(1)
```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_UpdateRsa1024PrivateKeyIndex(
        uint8_t *iv,
        uint8_t *encrypted_key,
        tsip_rsa1024_private_key_index_t *key_index
)
```

(2)
```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_UpdateRsa2048PrivateKeyIndex(
        uint8_t *iv,
        uint8_t *encrypted_key,
        tsip_rsa2048_private_key_index_t *key_index
)
```

Parameters

iv	Input	Initialization vector when generating encrypted_key
encrypted_key	Input	Encrypted key encrypted using KUK
key_index	Output	Wrapped key (1) 1024-bit RSA (2) 2048-bit RSA

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_RESOURCE_CONFLICT:	Occurrence of resource conflict because a hardware resource needed by the processing routine was in use by another processing routine
TSIP_ERR_FAIL	Occurrence of internal error

Description

This API updates a 1024-bit or 2048-bit RSA secret wrapped key. Refer to 7.3.4, RSA, for the format of the data encrypted using the UFPK input as encrypted_key.

Refer to 3.7.1, Key Injection and Updating, for an explanation of iv and encrypted_key and how to use key_index.

Reentrancy

Not supported.

4.2.6.5 R_TSIP_GenerateRsa*XXX*RandomKeyIndex

Format

(1) #include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_GenerateRsa1024RandomKeyIndex(
tsip_rsa1024_key_pair_index_t *key_pair_index
)

(2) #include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_GenerateRsa2048RandomKeyIndex(
tsip_rsa2048_key_pair_index_t *key_pair_index
)

Parameters

key_pair_index	Output	RSA key pair wrapped keys
public		RSA public key with wrapped key format
value		Public key value
key_management_info1		Key management information
key_n		Modulus n (plaintext) (1) 1024-bit RSA (2) 2048-bit RSA
key_e		Exponent e (plaintext) (1) 1024-bit RSA (2) 2048-bit RSA
dummy		Dummy
key_management_info2		Key management information
private		RSA secret wrapped key

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_RESOURCE_CONFLICT:	Occurrence of resource conflict because a hardware resource needed by the processing routine was in use by another processing routine
TSIP_ERR_FAIL	Occurrence of internal error

Description

This API outputs 1024-bit or 2048-bit wrapped keys for an RSA public key–secret key pair. The API generates a user key from a random number within the TSIP. Therefore, no user key needs to be input. Encrypting data using the wrapped keys output by this API makes it possible to prevent dead copying of data. The public key with wrapped key format is generated in key_pair_index->public, and the secret wrapped key is generated in key_pair_index->private. The only public key exponent generated is 0x00010001.

Refer to 3.7.1, Key Injection and Updating, for an explanation of how to use key_pair_index->public and key_pair_index->private. The parameter key_pair_index->public is utilized in the same manner as the public key with wrapped key format output by R_TSIP_GenerateRsaXXXPublicKeyIndex(), and key_pair_index->private is utilized in the same manner as the secret wrapped key output by R_TSIP_GenerateRsaXXXPrivateKeyIndex().

Reentrancy

Not supported.

4.2.6.6 R_TSIP_RsaesPkcs*XXX*Encrypt

Format

(1)
```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_RsaesPkcs1024Encrypt(
        tsip_rsa_byte_data_t *plain,
        tsip_rsa_byte_data_t *cipher,
        tsip_rsa1024_public_key_index_t *key_index
)
```
(2)
```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_RsaesPkcs2048Encrypt(
        tsip_rsa_byte_data_t *plain,
        tsip_rsa_byte_data_t *cipher,
        tsip_rsa2048_public_key_index_t *key_index
)
```
(3)
```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_RsaesPkcs3072Encrypt(
        tsip_rsa_byte_data_t *plain,
        tsip_rsa_byte_data_t *cipher,
        tsip_rsa3072_public_key_index_t *key_index
)
```
(4)
```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_RsaesPkcs4096Encrypt(
        tsip_rsa_byte_data_t *plain,
        tsip_rsa_byte_data_t *cipher,
        tsip_rsa4096_public_key_index_t *key_index
)
```

Parameters

plain	Input	Plaintext data to be encrypted
pdata		Specifies pointer to array containing plaintext.
data_length		Specifies valid data length of plaintext array. Data size <= modulus n size – 11
cipher	Output	Encrypted data
pdata		Specifies pointer to array containing ciphertext.
data_length		Inputs ciphertext buffer size, and outputs valid data length after encryption (modulus n size)
key_index	Input	Public key with wrapped key format (1) 1024-bit RSA (2) 2048-bit RSA (3) 3072-bit RSA (4) 4096-bit RSA

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_RESOURCE_CONFLICT:	Occurrence of resource conflict because a hardware resource needed by the processing routine was in use by another processing routine
TSIP_ERR_KEY_SET	Invalid wrapped key input
TSIP_ERR_PARAMETER	Invalid input data
TSIP_ERR_FAIL	Occurrence of internal error (Only for *XXX* = 3072, 4096)

Description

The R_TSIP_RsaesPkcsXXXEncrypt() function encrypts in RSA mode the plaintext input as the first parameter, plain, according to RSAES-PKCS1-V1_5. It then writes the encrypted result to the second parameter, cipher.

Refer to 3.7.1, Key Injection and Updating, for an explanation of how to generate key_index.

Reentrancy

Not supported.

4.2.6.7 R_TSIP_RsaesPkcs*XXX*Decrypt

Format

(1) #include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_RsaesPkcs1024Decrypt(
tsip_rsa_byte_data_t *cipher,
tsip_rsa_byte_data_t *plain,
tsip_rsa1024_private_key_index_t *key_index
)

(2) #include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_RsaesPkcs2048Decrypt(
tsip_rsa_byte_data_t *cipher,
tsip_rsa_byte_data_t *plain,
tsip_rsa2048_private_key_index_t *key_index
)

Parameters

cipher	Input	Encrypted data to be decrypted
pdata		Specifies pointer to array containing ciphertext.
data_length		Specifies valid data length of ciphertext array (modulus n size).
plain	Output	Decrypted plaintext data
pdata		Specifies pointer to array containing plaintext.
data_length		Specifies valid data length of plaintext array. Data size <= modulus n size – 11
key_index	Input	Wrapped key (1) 1024-bit RSA (2) 2048-bit RSA

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_RESOURCE_CONFLICT:	Occurrence of resource conflict because a hardware resource needed by the processing routine was in use by another processing routine
TSIP_ERR_KEY_SET	Invalid wrapped key input
TSIP_ERR_PARAMETER	Invalid input data

Description

The R_TSIP_RsaesPkcsXXXDecrypt() function decrypts in RSA mode the ciphertext input as the first parameter, cipher, according to RSAES-PKCS1-V1_5. It then writes the decrypted result to the second parameter, plain.

Refer to 3.7.1, Key Injection and Updating, for an explanation of how to generate key_index.

Reentrancy

Not supported.

4.2.6.8 R_TSIP_RsaesOaep*XXX*Encrypt

Format

(1) #include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_RsaesOaep1024Encrypt(
tsip_rsa_byte_data_t *plain,
tsip_rsa_byte_data_t *cipher,
tsip_rsa1024_public_key_index_t *key_index,
uint8_t hash_type,
uint8_t mgf_hash_type,
uint8_t *label,
uint32_t label_length
)

(2) #include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_RsaesOaep2048Encrypt(
tsip_rsa_byte_data_t *plain,
tsip_rsa_byte_data_t *cipher,
tsip_rsa2048_public_key_index_t *key_index
uint8_t hash_type,
uint8_t mgf_hash_type,
uint8_t *label,
uint32_t label_length
)

Parameters

plain	Input	Plaintext data to be encrypted
pdata		Specifies pointer to array containing plaintext.
data_length		Specifies valid data length of plaintext array. Data size <= modulus n size – 11
cipher	Output	Encrypted data
pdata		Specifies pointer to array containing ciphertext.
data_length		Inputs ciphertext buffer size, and outputs valid data length after encryption (modulus n size)
key_index	Input	Public key with wrapped key format (1) 1024-bit RSA (2) 2048-bit RSA
hash_type	Input	Hash type to use in encoding process R_TSIP_RSA_HASH_SHA1 R_TSIP_RSA_HASH_SHA256
mgf_hash_type	Input	Hash type to use in mask generation function (MGF1) R_TSIP_RSA_HASH_SHA1 R_TSIP_RSA_HASH_SHA256
label	Input	Label data
label_length	Input	Dsta length of label Data size <= 2^16-1

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_RESOURCE_CONFLICT:	Occurrence of resource conflict because a hardware resource needed by the processing routine was in use by another processing routine
TSIP_ERR_KEY_SET	Invalid wrapped key input
TSIP_ERR_PARAMETER	Invalid input data

Description

The R_TSIP_RsassaOaepXXXEncrypt() function encrypts, in accordance with RSAES-OAEP in section 7.1 of RFC 8017, plain text as the first parameter, plain, using the public wrapped key input as the third parameter, key_index, and writes the result to the second parameter, cipher. Hash algorithm used in RSAES-OAEP enceyption is specified by the fourth parameter, hash_type. Hash algorithm used in mask generation function is specified by fifth parameter, mgf_hash_type.

When a label is used in the encryption, the value is specified by sixth parameter, label, and the length is specified by the seventh parameter, label_length. When a label is not used in the encryption, specify NULL to the label and 0 to label_length.

Refer to 3.7.1, Key Injection and Updating, for an explanation of how to generate key_index.

Reentrancy

Not supported.

4.2.6.9 R_TSIP_RsaesOaep*XXX*Decrypt

Format

(1) #include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_RsaesOaep1024Decrypt(
tsip_rsa_byte_data_t *cipher,
tsip_rsa_byte_data_t *plain,
tsip_rsa1024_private_key_index_t *key_index,
uint8_t hash_type,
uint8_t mgf_hash_type,
uint8_t *label,
uint32_t label_length
)

(2) #include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_RsaesOaep2048Decrypt(
tsip_rsa_byte_data_t *cipher,
tsip_rsa_byte_data_t *plain,
tsip_rsa2048_private_key_index_t *key_index,
uint8_t hash_type,
uint8_t mgf_hash_type,
uint8_t *label,
uint32_t label_length
)

Parameters

cipher	Input	Encrypted data to be decrypted
pdata		Specifies pointer to array containing ciphertext.
data_length		Specifies valid data length of ciphertext array (modulus n size).
plain	Output	Decrypted plaintext data
pdata		Specifies pointer to array containing plaintext.
data_length		Specifies valid data length of plaintext array. Data size <= modulus n size – 11
key_index	Input	Wrapped key (1) 1024-bit RSA (2) 2048-bit RSA
hash_type	Input	Hash type to use in decoding process R_TSIP_RSA_HASH_SHA1 R_TSIP_RSA_HASH_SHA256
mgf_hash_type	Input	Hash type to use in mask generation function (MGF1) R_TSIP_RSA_HASH_SHA1 R_TSIP_RSA_HASH_SHA256
label	Input	Label data
label_length	Input	Dsta length of label Data size <= 2^16-1

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_RESOURCE_CONFLICT:	Occurrence of resource conflict because a hardware resource needed by the processing routine was in use by another processing routine
TSIP_ERR_KEY_SET	Invalid wrapped key input
TSIP_ERR_PARAMETER	Invalid input data
TSIP_ERR_FAIL	Decryption failed

Description

The R_TSIP_RsassaOaepXXXDecrypt() function decrypts, in accordance with RSAES-OAEP in section 7.1 of RFC 8017, cipher text as the first parameter, cipher, using the private wrapped key input as the third parameter, key_index, and writes the result to the second parameter, plain. Hash algorithm used in RSAES-OAEP enceyption is specified by the fourth parameter, hash_type. Hash algorithm used in mask generation function is specified by fifth parameter, mgf_hash_type.

When a label is used in the decryption, the value is specified by sixth parameter, label, and the length is specified by the seventh parameter, label_length. When a label is not used in the encryption, specify NULL to the label and 0 to label_length.

Refer to 3.7.1, Key Injection and Updating, for an explanation of how to generate key_index.

Reentrancy

Not supported.

4.2.6.10 R_TSIP_RsassaPkcs*XXX*SignatureGenerate

Format

(1)
```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_RsassaPkcs1024SignatureGenerate(
        tsip_rsa_byte_data_t *message_hash,
        tsip_rsa_byte_data_t *signature,
        tsip_rsa1024_private_key_index_t *key_index,
        uint8_t hash_type
)
```

(2)
```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_RsassaPkcs2048SignatureGenerate(
        tsip_rsa_byte_data_t *message_hash,
        tsip_rsa_byte_data_t *signature,
        tsip_rsa2048_private_key_index_t *key_index,
        uint8_t hash_type
)
```

Parameters

message_hash	Input	Message or hash value information to which to attach signature
pdata		Specifies pointer to array containing message or hash value.
data_length		Specifies valid data length of array (specified for message only). Maximum size is 2^31-1 byte.
data_type		Selects data type of message_hash. Message: 0 Hash value: 1
signature	Output	Signature text storage destination information
pdata		Specifies pointer to array containing signature text.
data_length		Data length (byte units)
key_index	Input	Wrapepd key (1) 1024-bit RSA (2) 2048-bit RSA
hash_type	Input	Hash type to attach to signature R_TSIP_RSA_HASH_MD5 R_TSIP_RSA_HASH_SHA1 R_TSIP_RSA_HASH_SHA256

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_RESOURCE_CONFLICT:	Occurrence of resource conflict because a hardware resource needed by the processing routine was in use by another processing routine
TSIP_ERR_KEY_SET	Invalid wrapped key input

TSIP_ERR_PARAMETER	Invalid input data

Description

The R_TSIP_RsassaPkcsXXXSignatureGenerate() function generates, in accordance with RSASSA-PKCS1-V1_5, signature text from the message text or hash value input as the first parameter, message_hash, using the secret wrapped key input as the third parameter, key_index, and writes the result to the second parameter, signature. When a message is specified as the first parameter, message_hash->data_type, a hash value is calculated from the message as specified by the fourth parameter, hash_type. When specifying a hash value in the first parameter, message_hash->data_type, a hash value calculated with the hash algorithm specified by the fourth parameter, hash_type, must be input as message_hash->pdata.

Refer to 3.7.1, Key Injection and Updating, for an explanation of how to generate key_index.

Reentrancy

Not supported.

4.2.6.11 R_TSIP_RsassaPkcs*XXX*SignatureVerification

Format

(1) #include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_RsassaPkcs1024SignatureVerification(
tsip_rsa_byte_data_t *signature,
tsip_rsa_byte_data_t *message_hash,
tsip_rsa1024_public_key_index_t *key_index,
uint8_t hash_type
)

(2) #include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_RsassaPkcs2048SignatureVerification(
tsip_rsa_byte_data_t *signature,
tsip_rsa_byte_data_t *message_hash,
tsip_rsa2048_public_key_index_t *key_index,
uint8_t hash_type
)

(3) #include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_RsassaPkcs3072SignatureVerification(
tsip_rsa_byte_data_t *signature,
tsip_rsa_byte_data_t *message_hash,
tsip_rsa3072_public_key_index_t *key_index,
uint8_t hash_type
)

(4) #include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_RsassaPkcs4096SignatureVerification(
tsip_rsa_byte_data_t *signature,
tsip_rsa_byte_data_t *message_hash,
tsip_rsa4096_public_key_index_t *key_index,
uint8_t hash_type
)

Parameters

signature	Input	Signature text information to be verified
pdata		Specifies pointer to array containing signature text.
message_hash	Input	Message text or hash value information to be verified
pdata		Specifies pointer to array containing message or hash value.
data_length		Specifies valid data length of array (specified for message only). Maximum size is 2^31-1 byte.
data_type		Selects data type of message_hash. Message: 0 Hash value: 1
key_index	Input	Public key with wrapped key format (1) 1024-bit RSA (2) 2048-bit RSA (3) 3072-bit RSA (4) 4096-bit RSA

hash_type	Input	Hash type R_TSIP_RSA_HASH_MD5 R_TSIP_RSA_HASH_SHA1 R_TSIP_RSA_HASH_SHA256

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_RESOURCE_CONFLICT:	Occurrence of resource conflict because a hardware resource needed by the processing routine was in use by another processing routine
TSIP_ERR_KEY_SET	Invalid wrapped key input
TSIP_ERR_AUTHENTICATION	Signature verification failure
TSIP_ERR_PARAMETER	Invalid input data
TSIP_ERR_FAIL	Occurrence of internal error (Only for *XXX* = 3072, 4096)

Description

R_TSIP_RsassaPkcsXXXSignatureVerification() function verifies, in accordance with RSASSAPKCS1-V1_5, the signature text input as the first parameter signature, and the message text or hash value input as the second parameter, message_hash, using the public key with wrapped key format input as the third parameter, key_index. When a message is specified by the second parameter, message_hash>data_type, a hash value is calculated using the public key with wrapped key format input as the third parameter, key_index, as specified by the fourth parameter, hash_type. When specifying a hash value in the second parameter, message_hash->data_type, a hash value calculated with the hash algorithm specified by the fourth parameter, hash_type, must be input as message_hash->pdata.

Refer to 3.7.1, Key Injection and Updating, for an explanation of how to generate key_index.

Reentrancy

Not supported.

4.2.6.12 R_TSIP_RsassaPss*XXX*SignatureGenerate

Format

(1)
```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_RsassaPss1024SignatureGenerate(
        tsip_rsa_byte_data_t *message_hash,
        tsip_rsa_byte_data_t *signature,
        tsip_rsa1024_private_key_index_t *key_index,
        uint8_t hash_type
)
```

(2)
```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_RsassaPss2048SignatureGenerate(
        tsip_rsa_byte_data_t *message_hash,
        tsip_rsa_byte_data_t *signature,
        tsip_rsa2048_private_key_index_t *key_index,
        uint8_t hash_type
)
```

Parameters

message_hash	Input	Message or hash value information to which to attach signature
pdata		Specifies pointer to array containing message or hash value.
data_length		Specifies valid data length of array (specified for message only). Maximum size is 2^31-1 byte.
data_type		Selects data type of message_hash. Message: 0 Hash value: 1
signature	Output	Signature text storage destination information
pdata		Specifies pointer to array containing signature text.
data_length		Data length (byte units)
key_index	Input	Wrapped key (1) 1024-bit RSA (2) 2048-bit RSA
hash_type	Input	Hash type to attach to signature R_TSIP_RSA_HASH_SHA1 R_TSIP_RSA_HASH_SHA256

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_RESOURCE_CONFLICT:	Occurrence of resource conflict because a hardware resource needed by the processing routine was in use by another processing routine
TSIP_ERR_KEY_SET	Invalid wrapped key input
TSIP_ERR_PARAMETER	Invalid input data

Description

The R_TSIP_RsassaPssXXXSignatureGenerate() function generates, in accordance with RSASSA-PSS in section 8.1 of RFC 8017, signature text from the message text or hash value input as the first parameter, message_hash, using the secret wrapped key input as the third parameter, key_index, and writes the result to the second parameter, signature. The salt length is fixed at 32 bytes.

When a message is specified as the first parameter, message_hash->data_type, a hash value is calculated from the message as specified by the fourth parameter, hash_type.

When specifying a hash value in the first parameter, message_hash->data_type, a hash value calculated with the hash algorithm specified by the fourth parameter, hash_type, must be input as message_hash->pdata.

Refer to 3.7.1, Key Injection and Updating, for an explanation of how to generate key_index.

Reentrancy

Not supported.

4.2.6.13 R_TSIP_RsassaPss*XXX*SignatureVerification

Format

(1) #include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_RsassaPss1024SignatureVerification(
tsip_rsa_byte_data_t *signature,
tsip_rsa_byte_data_t *message_hash,
tsip_rsa1024_public_key_index_t *key_index,
uint8_t hash_type
)

(2) #include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_RsassaPss2048SignatureVerification(
tsip_rsa_byte_data_t *signature,
tsip_rsa_byte_data_t *message_hash,
tsip_rsa2048_public_key_index_t *key_index,
uint8_t hash_type
)

Parameters

signature	Input	Signature text information to be verified
pdata		Specifies pointer to array containing signature text.
message_hash	Input	Message text or hash value information to be verified
pdata		Specifies pointer to array containing message or hash value.
data_length		Specifies valid data length of array (specified for message only). Maximum size is 2^31-1 byte.
data_type		Selects data type of message_hash. Message: 0 Hash value: 1
key_index	Input	Public key with wrapped key format (1) 1024-bit RSA (2) 2048-bit RSA
hash_type	Input	Hash type R_TSIP_RSA_HASH_SHA1 R_TSIP_RSA_HASH_SHA256

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_RESOURCE_CONFLICT:	Occurrence of resource conflict because a hardware resource needed by the processing routine was in use by another processing routine
TSIP_ERR_KEY_SET	Invalid wrapped key input
TSIP_ERR_AUTHENTICATION	Signature verification failure
TSIP_ERR_PARAMETER	Invalid input data

Description

The R_TSIP_RsassaPssXXXSignatureVerification() function verifies, in accordance with RSASSA-PSS, the signature text input as the first parameter signature, and the message text or hash value input as the second parameter, message_hash, using the public key with wrapped key format input as the third parameter, key_index. The salt length is fixed at 32 bytes.

When a message is specified by the second parameter, message_hash>data_type, a hash value is calculated using the public key with wrapped key format input as the third parameter, key_index, as specified by the fourth parameter, hash_type.

When specifying a hash value in the second parameter, message_hash->data_type, a hash value calculated with the hash algorithm specified by the fourth parameter, hash_type, must be input as message_hash->pdata.

Refer to 3.7.1, Key Injection and Updating, for an explanation of how to generate key_index.

Reentrancy

Not supported.

4.2.7 ECC

4.2.7.1 R_TSIP_GenerateEccP*XXX*PublicKeyIndex

Format

(1)
```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_GenerateEccP192PublicKeyIndex(
        uint8_t *encrypted_provisioning_key,
        uint8_t *iv,
        uint8_t *encrypted_key,
        tsip_ecc_public_key_index_t *key_index
)
```
(2)
```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_GenerateEccP224PublicKeyIndex(
        uint8_t *encrypted_provisioning_key,
        uint8_t *iv,
        uint8_t *encrypted_key,
        tsip_ecc_public_key_index_t *key_index
)
```
(3)
```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_GenerateEccP256PublicKeyIndex(
        uint8_t *encrypted_provisioning_key,
        uint8_t *iv,
        uint8_t *encrypted_key,
        tsip_ecc_public_key_index_t *key_index
)
```
(4)
```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_GenerateEccP384PublicKeyIndex(
        uint8_t *encrypted_provisioning_key,
        uint8_t *iv,
        uint8_t *encrypted_key,
        tsip_ecc_public_key_index_t *key_index
)
```

Parameters

encrypted_provisioning_key	Input	W-UFPK
iv	Input	Initialization vector when generating encrypted_key
encrypted_key	Input	Encrypted key encrypted using UFPK
key_index	Output	Public key with wrapped key format
value		Public key value
key_management_info		Key management information
key_q		(1) ECC P-192 public key Q (plaintext) (2) ECC P-224 public key Q (plaintext) (3) ECC P-256 public key Q (plaintext) (4) ECC P-384 public key Q (plaintext)

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_RESOURCE_CONFLICT:	Occurrence of resource conflict because a hardware resource needed by the processing routine was in use by another processing routine
TSIP_ERR_FAIL	Occurrence of internal error

Description

This API outputs an ECC P-192, P-224, P-256, or P-384 public key with wrapped key format.

Refer to 7.3.5, ECC, for an explanation of the method used to encrypt the public key using the provisioning key input as encrypted_key.

Ensure that the areas allocated for encrypted_key and key_index do not overlap.

A structure which includes the public key plaintext data is output as key_index->value.key_q. Refer to 7.4.2, ECC, for the format.

Refer to 3.7.1, Key Injection and Updating, for an explanation of encrypted_provisioning_key, iv, and encrypted_key and how to use key_index.

Reentrancy

Not supported.

4.2.7.2 R_TSIP_GenerateEccP*XXX*PrivateKeyIndex

Format

(1)
```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_GenerateEccP192PrivateKeyIndex(
        uint8_t *encrypted_provisioning_key,
        uint8_t *iv,
        uint8_t *encrypted_key,
        tsip_ecc_private_key_index_t *key_index
)
```
(2)
```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_GenerateEccP224PrivateKeyIndex(
        uint8_t *encrypted_provisioning_key,
        uint8_t *iv,
        uint8_t *encrypted_key,
        tsip_ecc_private_key_index_t *key_index
)
```
(3)
```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_GenerateEccP256PrivateKeyIndex(
        uint8_t *encrypted_provisioning_key,
        uint8_t *iv,
        uint8_t *encrypted_key,
        tsip_ecc_private_key_index_t *key_index
)
```
(4)
```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_GenerateEccP384PrivateKeyIndex(
        uint8_t *encrypted_provisioning_key,
        uint8_t *iv,
        uint8_t *encrypted_key,
        tsip_ecc_private_key_index_t *key_index
)
```

Parameters

encrypted_provisioning_key	Input	W-UFPK
iv	Input	Initialization vector when generating encrypted_key
encrypted_key	Input	Encrypted key encrypted using UFPK
key_index	Output	Wrapped key (1) ECC P-192 secret wrapped key (2) ECC P-224 secret wrapped key (3) ECC P-256 secret wrapped key (4) ECC P-384 secret wrapped key

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_RESOURCE_CONFLICT:	Occurrence of resource conflict because a hardware resource needed by the processing routine was in use by another processing routine
TSIP_ERR_FAIL	Occurrence of internal error

Description

This API outputs an ECC P-192, P-224, P-256, or P-384 secret wrapped key.

Refer to 7.4, Public Key Index Formats for Asymmetric Cryptography, for the format of the data encrypted using the UFPK input as encrypted_key.

Ensure that the areas allocated for encrypted_key and key_index do not overlap.

Refer to 3.7.1, Key Injection and Updating, for an explanation of encrypted_provisioning_key, iv, and encrypted_key and how to use key_index.

Reentrancy

Not supported.

4.2.7.3 R_TSIP_UpdateEccP*XXX*PublicKeyIndex

Format

(1)
```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_UpdateEccP192PublicKeyIndex(
        uint8_t *iv,
        uint8_t *encrypted_key,
        tsip_ecc_public_key_index_t *key_index
)
```

(2)
```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_UpdateEccP224PublicKeyIndex(
        uint8_t *iv,
        uint8_t *encrypted_key,
        tsip_ecc_public_key_index_t *key_index
)
```

(3)
```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_UpdateEccP256PublicKeyIndex(
        uint8_t *iv,
        uint8_t *encrypted_key,
        tsip_ecc_public_key_index_t *key_index
)
```

(4)
```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_UpdateEccP384PublicKeyIndex(
        uint8_t *iv,
        uint8_t *encrypted_key,
        tsip_ecc_public_key_index_t *key_index
)
```

Parameters

iv	Input	Initialization vector when generating encrypted_key
encrypted_key	Input	Encrypted key encrypted using KUK
key_index	Output	Public key with wrapped key format
value		Public key value
key_management_info		Key management information
key_q		Public key (Qx || Qy) (plaintext) (1) ECC P-192 (2) ECC P-224 (3) ECC P-256 (4) ECC P-384

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_RESOURCE_CONFLICT:	Occurrence of resource conflict because a hardware resource needed by the processing routine was in use by another processing routine
TSIP_ERR_FAIL	Occurrence of internal error

Description

This API updates an ECC P-192, P-224, P-256, or P-384 public key with wrapped key format.

Refer to 7.3.5, ECC, for an explanation of the method and format used to encrypt the public key using the KUK input as encrypted_key.

Refer to 7.4.2, ECC, for the format of the public key Q plaintext data output as key_index->value.key_q.

Refer to 3.7.1, Key Injection and Updating, for an explanation of iv and encrypted_key and how to use key_index.

Reentrancy

Not supported.

4.2.7.4 R_TSIP_UpdateEccP*XXX*PrivateKeyIndex

Format

(1)
```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_UpdateEccP192PrivateKeyIndex(
        uint8_t *iv,
        uint8_t *encrypted_key,
        tsip_ecc_private_key_index_t *key_index
)
```

(2)
```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_UpdateEccP224PrivateKeyIndex(
        uint8_t *iv,
        uint8_t *encrypted_key,
        tsip_ecc_private_key_index_t *key_index
)
```

(3)
```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_UpdateEccP256PrivateKeyIndex(
        uint8_t *iv,
        uint8_t *encrypted_key,
        tsip_ecc_private_key_index_t *key_index
)
```

(4)
```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_UpdateEccP384PrivateKeyIndex(
        uint8_t *iv,
        uint8_t *encrypted_key,
        tsip_ecc_private_key_index_t *key_index
)
```

Parameters

iv	Input	Initialization vector when generating encrypted_key
encrypted_key	Input	Encrypted key encrypted using KUK
key_index	Output	Wrapped key (1) ECC P-192 secret key (2) ECC P-224 secret key (3) ECC P-256 secret key (4) ECC P-384 secret key

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_RESOURCE_CONFLICT:	Occurrence of resource conflict because a hardware resource needed by the processing routine was in use by another processing routine
TSIP_ERR_FAIL	Occurrence of internal error

Description

This API updates an ECC P-192, P-224, P-256, or P-384 secret wrapped key.

Refer to 7.3.5, ECC, for an explanation of the method and format used to encrypt the private key using the KUK input as encrypted_key.

Refer to 3.7.1, Key Injection and Updating, for an explanation of iv and encrypted_key and how to use key_index.

Reentrancy

Not supported.

4.2.7.5 R_TSIP_GenerateEccP*XXX*RandomKeyIndex

Format

(1)
```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_GenerateEccP192RandomKeyIndex(
        tsip_ecc_key_pair_index_t *key_pair_index
)
```
(2)
```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_GenerateEccP224RandomKeyIndex(
        tsip_ecc_key_pair_index_t *key_pair_index
)
```
(3)
```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_GenerateEccP256RandomKeyIndex(
        tsip_ecc_key_pair_index_t *key_pair_index
)
```
(4)
```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_GenerateEccP384RandomKeyIndex(
        tsip_ecc_key_pair_index_t *key_pair_index
)
```

Parameters

key_pair_index	Output	ECC public key–secret key pair wrapped keys (1) ECC P-192 (2) ECC P-224 (3) ECC P-256 (4) ECC P-384
->public		Public key with wrapped key format
value		Public key value
key_management_info		Key management information
key_q		Public key (Qx || Qy) (plaintext)
->private		Secret wrapped key

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_RESOURCE_CONFLICT:	Occurrence of resource conflict because a hardware resource needed by the processing routine was in use by another processing routine
TSIP_ERR_FAIL	Occurrence of internal error

Description

This API outputs P-192, P-224, P-256, or P-384 wrapped keys for an ECC public key–secret key pair. The API generates a user key from a random number within the TSIP. Therefore, no user key needs to be input. Encrypting data using the wrapped keys output by this API makes it possible to prevent dead copying of data. The public key with wrapped key format is generated in key_pair_index->public, and the secret wrapped key is generated in key_pair_index->private. The plaintext public key is output to key_pair_index->public.value.key_q. Refer to 7.4, Public Key Index Formats for Asymmetric Cryptography, for the data format.

Refer to 3.7.1, Key Injection and Updating, for an explanation of how to use key_pair_index->public and key_pair_index->private. The parameter key_pair_index->public is utilized in the same manner as the public key with wrapped key format output by R_TSIP_GenerateEccPXXXPublicKeyIndex(), and key_pair_index->private is utilized in the same manner as the secret wrapped key output by R_TSIP_GenerateEccPXXXPrivateKeyIndex().

Reentrancy

Not supported.

4.2.7.6 R_TSIP_EcdsaP*XXX*SignatureGenerate

Format

(1)
```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_EcdsaP192SignatureGenerate(
        tsip_ecdsa_byte_data_t *message_hash,
        tsip_ecdsa_byte_data_t *signature,
        tsip_ecc_private_key_index_t *key_index
)
```

(2)
```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_EcdsaP224SignatureGenerate(
        tsip_ecdsa_byte_data_t *message_hash,
        tsip_ecdsa_byte_data_t *signature,
        tsip_ecc_private_key_index_t *key_index
)
```

(3)
```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_EcdsaP256SignatureGenerate(
        tsip_ecdsa_byte_data_t *message_hash,
        tsip_ecdsa_byte_data_t *signature,
        tsip_ecc_private_key_index_t *key_index
)
```

(4)
```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_EcdsaP384SignatureGenerate(
        tsip_ecdsa_byte_data_t *message_hash,
        tsip_ecdsa_byte_data_t *signature,
        tsip_ecc_private_key_index_t *key_index
)
```

Parameters

message_hash	Input	Message or hash value information to which to attach signature
pdata		Specifies pointer to array containing message or hash value.
data_length		Specifies valid data length of array (specified for message only). (1)~(3) Maximum size is 2^31-1 byte. (4) Maximum size is depends on user-definened function of SHA384
data_type		Selects data type of message_hash. Message: 0 Hash value: 1
signature	Output	Signature text storage destination information
pdata		Specifies pointer to array containing signature text. The signature format is as follows: (1) “0 padding (64 bits) || signature r (192 bits) || 0 padding (64 bits) || signature s (192 bits)” (2) “0 padding (32 bits) || signature r (224 bits) || 0 padding (32 bits) || signature s (224 bits)”

		(3) “signature r (256 bits) || signature s (256 bits)”
		(4) “signature r (384 bits) || signature s (384 bits)”
data_length		Data length (byte units)
key_index	Input	Wrapped key (1) ECC P-192 secret key (2) ECC P-224 secret key (3) ECC P-256 secret key (4) ECC P-384 secret key

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_RESOURCE_CONFLICT:	Occurrence of resource conflict because a hardware resource needed by the processing routine was in use by another processing routine
TSIP_ERR_KEY_SET	Invalid wrapped key index input
TSIP_ERR_FAIL	Occurrence of internal error
TSIP_ERR_PARAMETER	Invalid input data

Description

When using (1) R_TSIP_EcdsaP192SignatureGenerate, (2) R_TSIP_EcdsaP224SignatureGenerate, or (3) R_TSIP_EcdsaP256SignatureGenerate

When a message is specified by the first parameter, message_hash->data_type, an SHA-256 hash of the message text input as the first parameter, message_hash->pdata, is calculated, and the signature text is written to the second parameter, signature, in accordance with ECDSA P-192, P-224, or P-256 using the secret wrapped key input as the third parameter, key_index.

When a hash value is specified by the first parameter, message_hash->data_type, the signature text for the first XXX bits (XXX/8 bytes) of the SHA-256 hash value input as the first parameter, message_hash->pdata, is written to the second parameter, signature, in accordance with ECDSA P-192, P-224, or P-256 using the secret wrapped key input as the third parameter, key_index.

When using (4) R_TSIP_EcdsaP384SignatureGenerate

When a message is specified by the first parameter, message_hash->data_type, an SHA-384 hash of the message text input as the first parameter, message_hash->pdata, is calculated, and the signature text is written to the second parameter, signature, in accordance with ECDSA P-384 using the secret wrapped key input as the third parameter, key_index.

To use message input, refer to 4.3 and prepare a user-defined function for SHA384.

When a hash value is specified by the first parameter, message_hash->data_type, the signature text for the entire 48 bytes of the SHA-384 hash value input as the first parameter, message_hash->pdata, is written to the second parameter, signature, in accordance with ECDSA P-384 using the secret wrapped key input as the third parameter, key_index.

Refer to 3.7.1, Key Injection and Updating, for an explanation of how to generate key_index.

Reentrancy

Not supported.

4.2.7.7 R_TSIP_EcdsaP*XXX*SignatureVerification

Format

(1)
```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_EcdsaP192SignatureVerification(
        tsip_ecdsa_byte_data_t *signature,
        tsip_ecdsa_byte_data_t *message_hash,
        tsip_ecc_public_key_index_t *key_index
)
```
(2)
```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_EcdsaP224SignatureVerification(
        tsip_ecdsa_byte_data_t *signature,
        tsip_ecdsa_byte_data_t *message_hash,
        tsip_ecc_public_key_index_t *key_index
)
```
(3)
```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_EcdsaP256SignatureVerification(
        tsip_ecdsa_byte_data_t *signature,
        tsip_ecdsa_byte_data_t *message_hash,
        tsip_ecc_public_key_index_t *key_index
)
```
(4)
```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_EcdsaP384SignatureVerification(
        tsip_ecdsa_byte_data_t *signature,
        tsip_ecdsa_byte_data_t *message_hash,
        tsip_ecc_public_key_index_t *key_index
)
```

Parameters

signature	Input	Signature text information to be verified
pdata		Specifies pointer to array containing signature text. The signature format is as follows: (1) “0 padding (64 bits) || signature r (192 bits) || 0 padding (64 bits) || signature s (192 bits)” (2) “0 padding (32 bits) || signature r (224 bits) || 0 padding (32 bits) || signature s (224 bits)” (3) “signature r (256 bits) || signature s (256 bits)” (4) “signature r (384 bits) || signature s (384 bits)”
message_hash	Input	Message text or hash value information to be verified
pdata		Specifies pointer to array containing message or hash value.
data_length		Specifies valid data length of array (specified for message only). (1)~(3) Maximum size is 2^31-1 byte. (4) Maximum size is depends on user-definened function of SHA384

data_type		Selects data type of message_hash. Message: 0 Hash value: 1
key_index	Input	Public key with wrapped key format (1) ECC P-192 public key (2) ECC P-224 public key (3) ECC P-256 public key (4) ECC P-384 public key

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_RESOURCE_CONFLICT:	Occurrence of resource conflict because a hardware resource needed by the processing routine was in use by another processing routine
TSIP_ERR_KEY_SET	Invalid wrapped key index input
TSIP_ERR_FAIL	Internal error, or signature verification failure
TSIP_ERR_PARAMETER	Invalid input data

Description

When using (1) R_TSIP_EcdsaP192SignatureVerification, (2) R_TSIP_EcdsaP224SignatureVerification, or (3) R_TSIP_EcdsaP256SignatureVerification

When a message is specified by the second parameter, message_hash->data_type, an SHA-256 hash of the message text input as the second parameter, message_hash->pdata, is calculated, and the signature text input as the first parameter, signature, is validated in accordance with ECDSA P-192, P-224 or P-256 using the public key with wrapped key format input as the third parameter, key_index.

When a hash value is specified by the second parameter, message_hash->data_type, the signature text for the first XXX bits (XXX/8 bytes) of the SHA-256 hash value input as the second parameter, message_hash->pdata, and the signature text input as the first parameter, signature, is validated in accordance with ECDSA P-192, P-224 or P-256 using the public key with wrapped key format input as the third parameter, key_index.

When using (4) R_TSIP_EcdsaP384SignatureVerification

When a message is specified by the second parameter, message_hash->data_type, an SHA-384 hash of the message text input as the second parameter, message_hash->pdata, is calculated, and the signature text input as the first parameter, signature, is validated in accordance with ECDSA P-384 using the public key with wrapped key format input as the third parameter, key_index.

To use message input, refer to 4.3 and prepare a user-defined function for SHA384.

When a hash value is specified by the second parameter, message_hash->data_type, the signature text for the entire 48 bytes of the SHA-384 hash value input as the second parameter, message_hash->pdata, and the signature text input as the first parameter, signature, is validated in accordance with ECDSA P-384 using the public key with wrapped key format input as the third parameter, key_index.

Refer to 3.7.1, Key Injection and Updating, for an explanation of how to generate key_index.

Reentrancy

Not supported.

4.2.8 HASH

4.2.8.1 R_TSIP_Sha*XXX*Init

Format

(1)
```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_Sha1Init(
        tsip_sha_md5_handle_t *handle
)
```

(2)
```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_Sha256Init(
        tsip_sha_md5_handle_t *handle
)
```

Parameters

handle	Output	SHA handler (work area)

Return Values

TSIP_SUCCESS:	Normal termination

Description

The R_TSIP_ShaXXXInit() function performs preparations for the execution of an SHA1 or SHA256 hash calculation, and writes the result to the first parameter, handle. The parameter handle is used subsequently as a parameter by the R_TSIP_ShaXXXUpdate() and R_TSIP_ShaXXXFinal() functions.

Reentrancy

Not supported.

4.2.8.2 R_TSIP_Sha*XXX*Update

Format

(1)
```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_Sha1Update(
        tsip_sha_md5_handle_t *handle,
        uint8_t *message,
        uint32_t message_length
)
```

(2)
```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_Sha256Update(
        tsip_sha_md5_handle_t *handle,
        uint8_t *message,
        uint32_t message_length
)
```

Parameters

handle	Input/output	SHA handler (work area)
message	Input	Message area
message_length	Input	Byte length of message

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_RESOURCE_CONFLICT:	Occurrence of resource conflict because a hardware resource needed by the processing routine was in use by another processing routine
TSIP_ERR_PARAMETER:	Invalid handle input
TSIP_ERR_PROHIBIT_FUNCTION:	Invalid function called

Description

Using the handle specified by the first parameter, handle, the R_TSIP_ShaXXXUpdate() function calculates a hash value based on the second parameter, message, and the third parameter, message_length, and writes the ongoing status to the first parameter, handle (the value of which can be fetched using R_TSIP_GetCurrentHashDigestValue()). After message input completes, call R_TSIP_ShaXXXFinal().

Reentrancy

Not supported.

4.2.8.3 R_TSIP_Sha*XXX*Final

Format

(1)
```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_Sha1Final(
        tsip_sha_md5_handle_t *handle,
        uint8_t *digest,
        uint32_t *digest_length
)
```

(2)
```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_Sha256Final(
        tsip_sha_md5_handle_t *handle,
        uint8_t *digest,
        uint32_t *digest_length
)
```

Parameters

handle	Input	SHA handler (work area)
digest	Output	Hash data area
digest_length	Output	Byte length of hash data (1) SHA1: 20 bytes (2) SHA256: 32 bytes

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_RESOURCE_CONFLICT:	Occurrence of resource conflict because a hardware resource needed by the processing routine was in use by another processing routine
TSIP_ERR_PARAMETER:	Invalid handle input
TSIP_ERR_PROHIBIT_FUNCTION:	Invalid function called

Description

Using the handle specified as the first parameter, handle, the R_TSIP_ShaXXXFinal() function writes the calculation result to the second parameter, digest, and writes the length of the calculation result to the third parameter, digest_length.

Reentrancy

Not supported.

4.2.8.4 R_TSIP_Md5Init

Format

```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_Md5Init(
        tsip_sha_md5_handle_t *handle
)
```

Parameters

handle	Output	MD5 handler (work area)

Return Values

TSIP_SUCCESS:	Normal termination

Description

The R_TSIP_Md5Init() function performs preparations for the execution of hash calculation and writes the result to the first parameter, handle. The parameter handle is used subsequently as a parameter by the R_TSIP_Md5Update() and R_TSIP_Md5Final() functions.

Reentrancy

Not supported.

4.2.8.5 R_TSIP_Md5Update

Format

```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_Md5Update(
        tsip_sha_md5_handle_t *handle,
        uint8_t *message,
        uint32_t message_length
)
```

Parameters

handle	Input/output	MD5 handler (work area)
message	Input	Message area
message_length	Input	Byte length of message

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_RESOURCE_CONFLICT:	Occurrence of resource conflict because a hardware resource needed by the processing routine was in use by another processing routine
TSIP_ERR_PARAMETER:	Invalid handle input
TSIP_ERR_PROHIBIT_FUNCTION:	Invalid function called

Description

Using the handle specified by the first parameter, handle, the R_TSIP_Md5Update() function calculates a hash value based on the second parameter, message, and the third parameter, message_length, and writes the ongoing status to the first parameter, handle (the value of which can be fetched using R_TSIP_GetCurrentHashDigestValue()). After message input completes, call R_TSIP_Md5Final().

Reentrancy

Not supported.

4.2.8.6 R_TSIP_Md5Final

Format

```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_Sha1Final(
        tsip_sha_md5_handle_t *handle,
        uint8_t *digest,
        uint32_t *digest_length
)
```

Parameters

handle	Input	MD5 handler (work area)
digest	Output	Hash data area
digest_length	Output	Byte length of hash data (16 bytes)

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_RESOURCE_CONFLICT:	Occurrence of resource conflict because a hardware resource needed by the processing routine was in use by another processing routine
TSIP_ERR_PARAMETER:	Invalid handle input
TSIP_ERR_PROHIBIT_FUNCTION:	Invalid function called

Description

Using the handle specified as the first parameter, handle, the R_TSIP_Md5Final() function writes the calculation result to the second parameter, digest, and writes the length of the calculation result to the third parameter, digest_length.

Reentrancy

Not supported.

4.2.8.7 R_TSIP_GetCurrentHashDigestValue

Format

```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_GetCurrentHashDigestValue(
        tsip_sha_md5_handle_t *handle,
        uint8_t *digest,
        uint32_t *digest_length
)
```

Parameters

handle	Input	SHA or MD5 handler (work area)
digest	Output	Hash value for current input data area
digest_length	Output	Hash value for current input data length (16, 20, or 32 bytes)

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_PARAMETER:	Invalid handle input
TSIP_ERR_PROHIBIT_FUNCTION:	Invalid function called

Description

Using the handle specified as the parameter handle, this function outputs to the parameter digest the hash value for current input data after execution of an Update() function*1 and outputs to the parameter digest_length the data length.

Note: 1. R_TSIP_Sha1Update(), R_TSIP_Sha256Update(), or R_TSIP_Md5Update() function

Reentrancy

Not supported.

4.2.9 HMAC

4.2.9.1 R_TSIP_GenerateSha*XXX*HmacKeyIndex

Format

(1)
```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_GenerateSha1HmacKeyIndex (
        uint8_t *encrypted_provisioning_key,
        uint8_t *iv,
        uint8_t *encrypted_key,
        tsip_hmac_sha_key_index_t *key_index
)
```

(2)
```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_GenerateSha256HmacKeyIndex (
        uint8_t *encrypted_provisioning_key,
        uint8_t *iv,
        uint8_t *encrypted_key,
        tsip_hmac_sha_key_index_t *key_index
)
```

Parameters

encrypted_provisioning_key	Input	W-UFPK
iv	Input	Initialization vector when generating encrypted_key
encrypted_key	Input	Encrypted key encrypted using UFPK
key_index	Output	Wrapped key

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_FAIL:	Occurrence of internal error
TSIP_ERR_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 outputs an SHA1-HMAC or SHA256-HMAC wrapped key.

Refer to 7.3.6, SHA-HMAC, for the format of the data encrypted using the UFPK input as encrypted_key.

Refer to 3.7.1, Key Injection and Updating, for an explanation of encrypted_provisioning_key, iv, and encrypted_key and how to use key_index.

Reentrancy

Not supported.

4.2.9.2 R_TSIP_UpdateSha*XXX*HmacKeyIndex

Format

(1)
```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_UpdateSha1HmacKeyIndex (
        uint8_t *iv,
        uint8_t *encrypted_key,
        tsip_hmac_sha_key_index_t *key_index
)
```

(2)
```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_UpdateSha256HmacKeyIndex (
        uint8_t *iv,
        uint8_t *encrypted_key,
        tsip_hmac_sha_key_index_t *key_index
)
```

Parameters

iv	Input	Initialization vector when generating encrypted_key
encrypted_key	Input	Encrypted key encrypted using KUK
key_index	Output	Wrapped key

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_FAIL:	Occurrence of internal error
TSIP_ERR_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 updates a SHA-HMAC wrapped key.

Refer to 7.3.6, SHA-HMAC for the format of the data encrypted using the KUK input as encrypted_key.

Refer to 3.7.1, Key Injection and Updating, for an explanation of encrypted_provisioning_key, iv, and encrypted_key and how to use key_index.

Reentrancy

Not supported.

4.2.9.3 R_TSIP_Sha*XXX*HmacGenerateInit

Format

(1) #include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_Sha1HmacGenerateInit(
tsip_hmac_sha_handle_t *handle,
tsip_hmac_sha_key_index_t *key_index
)

(2) #include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_Sha256HmacGenerateInit(
tsip_hmac_sha_handle_t *handle,
tsip_hmac_sha_key_index_t *key_index
)

Parameters

handle	Output	SHA-HMAC handler (work area)
key_index	Input	Wrapped key

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_RESOURCE_CONFLICT:	Occurrence of resource conflict because a hardware resource needed by the processing routine was in use by another processing routine
TSIP_ERR_KEY_SET:	Invalid wrapped key input

Description

The R_TSIP_ShaXXXHmacGenerateInit() function uses the second parameter, key_index, to perform preparations for the execution of SHA1-HMAC or SHA256-HMAC calculation and writes the result to the first parameter, handle. When using the TLS cooperation function, use the wrapped key generated by the R_TSIP_TlsGenerateSessionKey() function as key_index. The parameter handle is used subsequently as a parameter by the R_TSIP_ShaXXXHmacGenerateUpdate() and R_TSIP_ShaXXXHmacGenerateFinal() functions.

Refer to 3.7.1, Key Injection and Updating, for an explanation of how to generate key_index.

Reentrancy

Not supported.

4.2.9.4 R_TSIP_Sha*XXX*HmacGenerateUpdate

Format

(1)
```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_Sha1HmacGenerateUpdate(
        tsip_hmac_sha_handle_t *handle,
        uint8_t *message,
        uint32_t message_length
)
```

(2)
```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_Sha256HmacGenerateUpdate(
        tsip_hmac_sha_handle_t *handle,
        uint8_t *message,
        uint32_t message_length
)
```

Parameters

handle	Input/output	SHA-HMAC handler (work area)
message	Input	Message area
message_length	Input	Byte length of message

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_PARAMETER:	Invalid handle input
TSIP_ERR_PROHIBIT_FUNCTION:	Invalid function called

Description

Using the handle specified by the first parameter, handle, the R_TSIP_ShaXXXHmacGenerateUpdate() function calculates a hash value based on the second parameter, message, and the third parameter, message_length, and writes the ongoing status to the first parameter, handle. After message input completes, call R_TSIP_ShaXXXHmacGenerateFinal().

Reentrancy

Not supported.

4.2.9.5 R_TSIP_Sha*XXX*HmacGenerateFinal

Format

(1)
```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_Sha1HmacGenerateFinal(
        tsip_hmac_sha_handle_t *handle,
        uint8_t *mac
)
```
(2)
```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_Sha256HmacGenerateFinal(
        tsip_hmac_sha_handle_t *handle,
        uint8_t *mac
)
```

Parameters

handle	Input	SHA-HMAC handler (work area)
mac	Output	HMAC area (1) SHA1-HMAC: 20 bytes (2) SHA256-HMAC: 32 bytes

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_FAIL:	Occurrence of internal error
TSIP_ERR_PARAMETER:	Invalid handle input
TSIP_ERR_PROHIBIT_FUNCTION:	Invalid function called

Description

The R_TSIP_ShaXXXHmacGenerateFinal() function uses the handle specified as the first parameter, handle, and writes the calculation result to the second parameter, mac.

Reentrancy

Not supported.

4.2.9.6 R_TSIP_Sha*XXX*HmacVerifyInit

Format

(1)
```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_Sha1HmacVerifyInit(
        tsip_hmac_sha_handle_t *handle,
        tsip_hmac_sha_key_index_t *key_index
)
```
(2)
```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_Sha256HmacVerifyInit(
        tsip_hmac_sha_handle_t *handle,
        tsip_hmac_sha_key_index_t *key_index
)
```

Parameters

handle	Output	SHA-HMAC handler (work area)
key_index	Input	Wrapped key

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_RESOURCE_CONFLICT:	Occurrence of resource conflict because a hardware resource needed by the processing routine was in use by another processing routine
TSIP_ERR_KEY_SET:	Invalid wrapped key input

Description

The R_TSIP_ShaXXXHmacVerifyInit() function uses the first parameter, key_index, to perform preparations for the execution of SHA1-HMAC or SHA256-HMAC calculation and writes the result to the first parameter, handle. When using the TLS cooperation function, use the wrapped key generated by the R_TSIP_TlsGenerateSessionKey() function as key_index. The parameter handle is used subsequently as a parameter by the R_TSIP_ShaXXXHmacVerifyUpdate() and R_TSIP_ShaXXXHmacVerifyFinal() functions.

Refer to 3.7.1, Key Injection and Updating, for an explanation of how to generate key_index.

Reentrancy

Not supported.

4.2.9.7 R_TSIP_Sha*XXX*HmacVerifyUpdate

Format

(1)
```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_Sha1HmacVerifyUpdate(
        tsip_hmac_sha_handle_t *handle,
        uint8_t *message,
        uint32_t message_length
)
```

(2)
```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_Sha256HmacVerifyUpdate(
        tsip_hmac_sha_handle_t *handle,
        uint8_t *message,
        uint32_t message_length
)
```

Parameters

handle	Input/output	SHA-HMAC handler (work area)
message	Input	Message area
message_length	Input	Byte length of message

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_PARAMETER:	Invalid handle input
TSIP_ERR_PROHIBIT_FUNCTION:	Invalid function called

Description

Using the handle specified by the first parameter, handle, the R_TSIP_ShaXXXHmacVerifyUpdate() function calculates a hash value based on the second parameter, message, and the third parameter, message_length, and writes the ongoing status to the first parameter, handle. After message input completes, call R_TSIP_ShaXXXHmacVerifyFinal().

Reentrancy

Not supported.

4.2.9.8 R_TSIP_Sha*XXX*HmacVerifyFinal

Format

(1)
```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_Sha1HmacVerifyFinal(
        tsip_hmac_sha_handle_t *handle,
        uint8_t *mac,
        uint32_t mac_length
)
```

(2)
```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_Sha256HmacVerifyFinal(
        tsip_hmac_sha_handle_t *handle,
        uint8_t *mac,
        uint32_t mac_length
)
```

Parameters

handle	Input	SHA-HMAC handler (work area)
mac	Input	HMAC area
mac_length	Input	HMAC length

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_FAIL:	Occurrence of internal error
TSIP_ERR_PARAMETER:	Invalid handle input
TSIP_ERR_PROHIBIT_FUNCTION:	Invalid function called

Description

Using the handle specified as the first parameter, handle, the R_TSIP_ShaXXXHmacVerifyFinal() function verifies the MAC value based on the second parameter, mac, and the third parameter, mac_length. The value of mac_length is specified in byte units. Input a value of 4 to 20 for SHA1-HMAC and 4 to 32 for SHA256-HMAC.

Reentrancy

Not supported.

4.2.10 DH

4.2.10.1 R_TSIP_Rsa2048DhKeyAgreement

Format

```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_Rsa2048DhKeyAgreement(
        tsip_aes_key_index_t *key_index,
        tsip_rsa2048_private_key_index_t *sender_private_key_index,
        uint8_t *message,
        uint8_t *receiver_modulus,
        uint8_t *sender_modulus
)
```

Parameters

key_index	Input	Wrapped key for AES-128 CMAC operation
sender_private_key_index	Input	Secret wrapped key for DH operation The secret key d included in the secret wrapped key is decrypted and used internally by the TSIP.
message	Input	Message (2048 bits) Set a value smaller than the prime number (d) included in sender_private_key_index.
receiver_modulus	Input	Modular exponentiation result calculated by the receiver + MAC 2048-bit modular exponentiation result || 128 bits
sender_modulus	Output	Modular exponentiation result calculated by the sender + MAC 2048-bit modular exponentiation result || 128 bits

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_KEY_SET:	Invalid wrapped key input
TSIP_ERR_RESOURCE_CONFLICT:	Occurrence of resource conflict because a hardware resource needed by the processing routine was in use by another processing routine
TSIP_ERR_FAIL:	Occurrence of internal error

Description

Performs a DH operation using RSA-2048.

The sender is the TSIP and the receiver is the other key exchange party.

Reentrancy

Not supported.

4.2.11 ECDH

4.2.11.1 R_TSIP_EcdhP256Init

Format

```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_EcdhP256Init (
        tsip_ecdh_handle_t *handle,
        uint32_t key_type,
        uint32_t use_key_id
)
```

Parameters

handle	Output	ECDH handler (work area)	
key_type	Input	Key exchange type	0: ECDHE 1: ECDH
user_key_id	Input	0: key_id not used 1: key_id used (for DLSM/COSEM)	

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_PARAMETER:	Invalid input data

Description

The R_TSIP_EcdhP256Init() function performs preparations for the execution of ECDH key exchange calculation and writes the result to the first parameter, handle. The parameter handle is used subsequently as a parameter by the R_TSIP_EcdhP256ReadPublicKey(), R_TSIP_EcdhP256MakePublicKey(), R_TSIP_EcdhP256CalculateSharedSecretIndex(), R_TSIP_EcdhP256KeyDerivation(), and R_TSIP_EcdhP256SshKeyDerivation() functions.

Use the second parameter, key_type, to select the type of ECDH key exchange. For ECDHE, the R_TSIP_EcdhP256MakePublicKey() function uses the TSIP’s random number generation functionality to generate an ECC P-256 key pair. For ECDH, keys injected beforehand are used for key exchange.

Input 1 as the third parameter, use_key_id, to use key_id when key exchange is performed. The key_id parameter is for applications conforming to the DLMS/COSEM standard for smart meters.

Refer to 3.7.1, Key Injection and Updating, for an explanation of how to generate key_index.

Reentrancy

Not supported.

4.2.11.2 R_TSIP_EcdhP256ReadPublicKey

Format

```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_EcdhP256ReadPublicKey (
        tsip_ecdh_handle_t *handle,
        tsip_ecc_public_key_index_t *public_key_index,
        uint8_t *public_key_data,
        tsip_ecdsa_byte_data_t *signature,
        tsip_ecc_public_key_index_t *key_index
)
```

Parameters

handle	Input/output	ECDH handler (work area)	
public_key_index	Input	Public wrapped key for signature verification	
public_key_data	Input	key_id not used	ECC P-256 public key (512 bits)
		key_id used	key_id (8 bits) || public key s (512 bits)
signature	Input	ECDSA P-256 signature of public_key_data	
key_index	Output	Wrapped key of public_key_data	

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_RESOURCE_CONFLICT:	Occurrence of resource conflict because a hardware resource needed by the processing routine was in use by another processing routine
TSIP_ERR_KEY_SET:	Invalid wrapped key input
TSIP_ERR_FAIL:	Internal error, or signature verification failure
TSIP_ERR_PARAMETER:	Invalid handle input
TSIP_ERR_PROHIBIT_FUNCTION:	Invalid function called

Description

The R_TSIP_EcdhP256ReadPublicKey() function verifies the signature of the ECC P-256 public key of the other ECDH key exchange party. If the signature is correct, it outputs the public_key_data wrapped key as the fifth parameter.

The first parameter, handle, is used subsequently as a parameter by the R_TSIP_EcdhP256CalculateSharedSecretIndex() function.

The second parameter, public_key_index, is a wrapped public key that is used to verify the fourth parameter, siganature, which is the signature of the third parameter, public_key_data.

The third parameter, public_key_data, is an ECC P-256 public key which is received from the other ECDH key exchange party. When key_id is used for DLSM/COSEM, concatenate key_id value to the head of the public key.

The fourth parameter, signature, is an ECDSA P-256 signature of the public_key_data. The hash function of the signature is SHA-256.

The R_TSIP_EcdhP256CalculateSharedSecretIndex() function uses key_index as input to calculate Z.

Refer to 3.7.1, Key Injection and Updating, for an explanation of how to generate public_key_index.

Reentrancy

Not supported.

4.2.11.3 R_TSIP_EcdhP256MakePublicKey

Format

```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_EcdhP256MakePublicKey (
        tsip_ecdh_handle_t *handle,
        tsip_ecc_public_key_index_t *public_key_index,
        tsip_ecc_private_key_index_t *private_key_index,
        uint8_t *public_key,
        tsip_ecdsa_byte_data_t *signature,
        tsip_ecc_private_key_index_t *key_index
)
```

Parameters

handle	Input/output	ECDH handler (work area) When using key_id, input handle->key_id after running R_TSIP_EcdhP256Init().
public_key_index	Input	For ECDHE, input a null pointer. For ECDH, input an ECC P-256 public wrapped key.
private_key_index	Input	ECC P-256 secret key for signature generation
public_key	Output	User public key (512 bits) for key exchange When using key_id, key_id (8 bits) || public key (512 bits) || 0 padding (24 bits)
signature ->pdata	Output	Signature text storage destination information : Specifies pointer to array containing signature text. Signature format: signature r (256 bits) || signature s (256 bits)”
->data_length		: Data length (byte units)
key_index	Output	For ECDHE, a secret wrapped key generated from a random number. Not output for ECDH.

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_RESOURCE_CONFLICT:	Occurrence of resource conflict because a hardware resource needed by the processing routine was in use by another processing routine
TSIP_ERR_KEY_SET:	Invalid wrapped key input
TSIP_ERR_FAIL:	Occurrence of internal error
TSIP_ERR_PARAMETER:	Invalid handle input
TSIP_ERR_PROHIBIT_FUNCTION:	Invalid function called

Description

The R_TSIP_EcdhP256MakePublicKey() function generates an ephemeral key pair and calculates a signature using a key that is either generated or input.

If ECDHE is specified by the key_type parameter of the R_TSIP_EcdhP256Init() function, the TSIP's random number generation functionality is used to generate an ECC P-256 key pair. The public key is output to public_key and the secret key is output to key_index.

If ECDH is specified by the key_type parameter of the R_TSIP_EcdhP256Init() function, the public key input as public_key_index is output to public_key and nothing is output to key_index.

The value of the first parameter, handle, is used subsequently as a parameter by the R_TSIP_EcdhP256CalculateSharedSecretIndex() function.

The second parameter, public_key_index, is an ECDH P-256 wrapped public key, this parameter is used when ECDH is used. When ECDHE is used, specify NULL to this parameter.

The third parameter, private_key_index, is an ECDH P-256 wrapped private key to use to generate the signature.

The fourth parameter, public_key, is an ECDH P-256 public key to send to the other ECDH key exchange party.

The fifth parameter, signature, is an ECDSA P-256 signature to send to the other ECDH key exchange party.

The R_TSIP_EcdhP256CalculateSharedSecretIndex() function uses the sixth parameter, key_index, as input to calculate Z.

Refer to 3.7.1, Key Injection and Updating, for an explanation of how to generate public_key_index and private_key_index.

Reentrancy

Not supported.

4.2.11.4 R_TSIP_EcdhP256CalculateSharedSecretIndex

Format

```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_EcdhP256CalculateSharedSecretIndex (
        tsip_ecdh_handle_t *handle,
        tsip_ecc_public_key_index_t *public_key_index,
        tsip_ecc_private_key_index_t *private_key_index,
        tsip_ecdh_key_index_t *shared_secret_index
)
```

Parameters

handle	Input/output	ECDH handler (work area)
public_key_index	Input	Public wrapped key whose signature was verified by R_TSIP_EcdhP256ReadPublicKey()
private_key_index	Input	Secret wrapped key
shared_secret_index	Output	Wrapped key of shared secret Z calculated during ECDH key exchange

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_RESOURCE_CONFLICT:	Occurrence of resource conflict because a hardware resource needed by the processing routine was in use by another processing routine
TSIP_ERR_KEY_SET:	Invalid wrapped key input
TSIP_ERR_FAIL:	Occurrence of internal error
TSIP_ERR_PARAMETER:	Invalid handle input
TSIP_ERR_PROHIBIT_FUNCTION:	Invalid function called

Description

The R_TSIP_EcdhP256CalculateSharedSecretIndex() function uses the ECDH key exchange algorithm to output the wrapped key of the shared secret Z derived from the public key of the other key exchange party and your own secret key.

As the second parameter, public_key_index, input the key_index, public wrapped key whose signature was verified by R_TSIP_EcdhP256ReadPublicKey() function.

If the value of the key_type parameter of R_TSIP_EcdhP256Init() function is 0 (ECDHE), input the key_index, secret wrapped key generated from a random number by R_TSIP_EcdhP256MakePublicKey() function as the third parameter, private_key_index, and if the value of the key_type is 1 (ECDH), input the secret wrapped key corresponding to the second parameter of R_TSIP_EcdhP256MakePublicKey() as the third parameter, private_key_index.

The R_TSIP_EcdhP256KeyDerivation() or R_TSIP_EcdhP256SshKeyDerivation() function subsequently uses shared_secret_index as key material for outputting the wrapped key.

Refer to 3.7.1, Key Injection and Updating, for an explanation of how to generate private_key_index to use in ECDH.

Reentrancy

Not supported.

4.2.11.5 R_TSIP_EcdhP256KeyDerivation

Format

```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_EcdhP256KeyDerivation (
        tsip_ecdh_handle_t *handle,
        tsip_ecdh_key_index_t *shared_secret_index,
        uint32_t key_type,
        uint32_t kdf_type,
        uint8_t *other_info,
        uint32_t other_info_length,
        tsip_hmac_sha_key_index_t *salt_key_index,
        tsip_aes_key_index_t *key_index
)
```

Parameters

handle	Input/output	ECDH handler (work area)	
shared_secret_index	Input	Z wrapped key calculated by R_TSIP_EcdhP256CalculateSharedSecretIndex	
key_type	Input	Derived key type	0: AES-128 1: AES-256 2: SHA256-HMAC
kdf_type	Input	Algorithm used for key derivation calculation	0: SHA256 1: SHA256-HMAC
other_info	Input	Additional data used for key derivation calculation Input a value of the concatenated data such as (AlgorithmID || PartyUInfo || PartyVInf || SuppPubInfo || SuppPrivInfo)	
other_info_length	Input	Byte length of other_info (from 23 to 147 byte units)	
salt_key_index	Input	Salt wrapped key (Input NULL when kdf_type is 0.)	
key_index	Output	Wrapped key index corresponding to key_type When the value of key_type is 2, an SHA256-HMAC wrapped key is output. The value of key_index can be specified by casting the start address of the area reserved beforehand by the tsip_hmac_sha_key_index_t type with the (tsip_aes_key_index_t*) type.	

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_RESOURCE_CONFLICT:	Occurrence of resource conflict because a hardware resource needed by the processing routine was in use by another processing routine
TSIP_ERR_KEY_SET:	Invalid wrapped key input
TSIP_ERR_PARAMETER:	Invalid handle input
TSIP_ERR_PROHIBIT_FUNCTION:	Invalid function called

Description

The R_TSIP_EcdhP256KeyDerivation() function uses the shared secret Z (shared_secret_index) calculated by the R_TSIP_EcdhP256CalculateSharedSecretIndex() function as key material to derive the wrapped key specified by the third parameter, key_type. The key derivation algorithm is one-step key derivation as defined in NIST SP800-56C. Any of AES-128, AES-256 or SHA-256 HMAC is specified by the fourth parameter, kdf_type. When SHA-256 HMAC is specified, the wrapped key output by the R_TSIP_GenerateSha256HmacKeyIndex() function or R_TSIP_UpdateSha256HmacKeyIndex() function is specified as the seventh parameter, salt_key_index.

Enter a fixed value for deriving a key shared with the other key exchange party as the fifth parameter, other_info. And enter the byte length of the other_info as the sixth parameter, other_info_length.

A key index corresponding to key_type is output as the eighth parameter, key_index. The correspondences between the types of derived key indexes and the functions with which they can be used as listed below.

Derived Wrapped Key	Compatible Functions
AES-128	All AES-128 Init functions and R_TSIP_Aes128KeyUnwrap()
AES-256	All AES-256 Init functions and R_TSIP_Aes256KeyUnwrap()
SHA256-HMAC	R_TSIP_Sha256HmacGenerateInit() and R_TSIP_Sha256HmacVerifyInit()

Reentrancy

Not supported.

4.2.11.6 R_TSIP_EcdheP512KeyAgreement

Format

```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_EcdheP512KeyAgreement(
        tsip_aes_key_index_t *key_index,
        uint8_t *receiver_public_key,
        uint8_t *sender_public_key
)
```

Parameters

key_index	Input	Wrapped key for AES-128 CMAC operation
receiver_public_key	Input	Brainpool P512r1 public key of receiver Q (1024 bits) || MAC (128 bits)
sender_public_key	Output	Brainpool P512r1 public key of sender Q (1024 bits) || MAC (128 bits)

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_RESOURCE_CONFLICT:	Occurrence of resource conflict because a hardware resource needed by the processing routine was in use by another processing routine
TSIP_ERR_KEY_SET:	Invalid wrapped key input
TSIP_ERR_FAIL:	Occurrence of internal error

Description

Performs an ECDHE operation after generation of a key pair using Brainpool P512r1.

The sender is the TSIP and the receiver is the other key exchange party.

Reentrancy

Not supported.

4.2.11.7 R_TSIP_EcdhP256SshKeyDerivation

Format

```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_EcdhP256SshKeyDerivation (
        tsip_ecdh_handle_t *handle,
        tsip_ecdh_key_index_t *shared_secret_index,
        uint32_t key_type,
        uint8_t *other_info,
        uint32_t other_info_length,
        tsip_aes_key_index_t *client_write_key_index,
        tsip_aes_key_index_t *server_write_key_index,
        tsip_aes_key_index_t *client_mac_key_index,
        tsip_aes_key_index_t *server_mac_key_index,
        uint8_t *client_iv,
        uint8_t *server_iv
)
```

Parameters

handle	Input/output	ECDH handler (work area)
shared_secret_index	Input	Z wrapped key calculated by R_TSIP_EcdhP256CalculateSharedSecretIndex
key_type	Input	Derived key type 0: AES-128 1: AES-256 2: SHA256-HMAC
other_info	Input	Additional data used for key derivation calculation V_C || V_S || I_C || I_S || K_S || e || f
other_info_length	Input	Byte length of other_info (2^24-1 or fewer byte units)
client_write_key_index	Output	Wrapped key to use in encryption by client side This output is ensbled when key_type = 0,1
server_write_key_index	Output	Wrapped key to use in encryption by server side This output is ensbled when key_type = 0,1
client_mac_key_index	Output	Wrapped key to generate integrity data by client side Wrapped key index corresponding to key_type When the value of key_type is 2, an SHA256-HMAC wrapped key is output. The value can be specified by casting the start address of the area reserved beforehand by the tsip_hmac_sha_key_index_t type with the (tsip_aes_key_index_t*) type.
server_mac_key_index	Output	Wrapped key to generate integrity data by server side Wrapped key index corresponding to key_type When the value of key_type is 2, an SHA256-HMAC wrapped key is output. The value can be specified by casting the start address of the area reserved beforehand by the tsip_hmac_sha_key_index_t type with the (tsip_aes_key_index_t*) type.

client_iv	Output	Initial vector to use by client side This output is ensbled when key_type = 0,1
server_iv	Output	Initial vector to use by server side This output is ensbled when key_type = 0,1

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_FAIL:	Internal error
TSIP_ERR_RESOURCE_CONFLICT:	Occurrence of resource conflict because a hardware resource needed by the processing routine was in use by another processing routine
TSIP_ERR_KEY_SET:	Invalid wrapped key input
TSIP_ERR_PARAMETER:	Invalid handle input
TSIP_ERR_PROHIBIT_FUNCTION:	Invalid function called

Description

The R_TSIP_EcdhP256SshKeyDerivation() function uses the shared secret Z (shared_secret_index) calculated by the R_TSIP_EcdhP256CalculateSharedSecretIndex() function as key material to derive the wrapped key specified by the third parameter, key_type. The key derivation algorithm is as defined in RFC4253.

Enter a fixed value for deriving a key shared with the other key exchange party as the fifth parameter, other_info. And enter the byte size to the fifth parameter, other_info_length.

Wrapped keys and initial vectors corresponding to key_type is output as from the sixth parameter to the eleventh parameter. The sixth parameter, client_write_key_index, is a wrapped key to use in encryption by client side. The seventh parameter, server_write_key_index, is a wrapped key to use in encryption by server side. The eighth parameter, client_mac_key_index, is a wrapped key to generate integrity data by client side. The nineth parameter, server_mac_key_index, is a wrapped key to generate integrity data by server side. The tenth parameter, client_iv, is an initial vector to use by client side. And the eleventh parameter, client_iv, is an initial vector to use by server side.

Reentrancy

Not supported.

4.2.12 Key Wrap

4.2.12.1 R_TSIP_Aes*XXX*KeyWrap

Format

(1)
```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_Aes128KeyWrap (
        tsip_aes_key_index_t *wrap_key_index,
        uint32_t target_key_type,
        tsip_aes_key_index_t *target_key_index,
        uint32_t *wrapped_key
)
```

(2)
```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_Aes256KeyWrap (
        tsip_aes_key_index_t *wrap_key_index,
        uint32_t target_key_type,
        tsip_aes_key_index_t *target_key_index,
        uint32_t *wrapped_key
)
```

Parameters

wrap_key_index	Input	(1) AES-128 wrapped key used for wrapping (2) AES-256 wrapped key used for wrapping
target_key_type	Input	Selects key to be wrapped. 0 (R_TSIP_KEYWRAP_AES128): AES-128 2 (R_TSIP_KEYWRAP_AES256): AES-256
target_key_index	Input	Wrapped key to be wrapped target_key_type 0: 13 word size target_key_type 2: 17 word size
wrapped_key	Output	Key which is wrapped target_key_type 0: 6 word size target_key_type 2: 10 word size

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_RESOURCE_CONFLICT:	Occurrence of resource conflict because a hardware resource needed by the processing routine was in use by another processing routine
TSIP_ERR_KEY_SET	Invalid wrapped key input
TSIP_ERR_FAIL	Occurrence of internal error

Description

The R_TSIP_AesXXXKeyWrap() function uses the first parameter, wrap_key_index, to wrap target_key_index, which is input as the third parameter. The wrapped key is written to the fourth parameter, wrapped_key. The processing conforms to the RFC 3394 wrapping algorithm. Use the second parameter, target_key_type, to select the key to be wrapped.

Use R_TSIP_Aes128KeyWrap() when the key length used for wrapping is 128 bits, and use R_TSIP_Aes256KeyWrap() when the key length is 256 bits.

Reentrancy

Not supported.

4.2.12.2 R_TSIP_Aes*XXX*KeyUnwrap

Format

(1)
```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_Aes128KeyUnwrap (
        tsip_aes_key_index_t *wrap_key_index,
        uint32_t target_key_type,
        uint32_t *wrapped_key,
        tsip_aes_key_index_t *target_key_index
)
```

(2)
```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_Aes256KeyUnwrap (
        tsip_aes_key_index_t *wrap_key_index,
        uint32_t target_key_type,
        uint32_t *wrapped_key,
        tsip_aes_key_index_t *target_key_index
)
```

Parameters

wrap_key_index	Input	(1) AES-128 wrapped key used for unwrapping (2) AES-256 wrapped key used for unwrapping
target_key_type	Input	Selects key to be unwrapped. 0 (R_TSIP_KEYWRAP_AES128): AES-128 2 (R_TSIP_KEYWRAP_AES256): AES-256
wrapped_key	Input	Key which is wrapped target_key_type 0: 6 word size target_key_type 2: 10 word size
target_key_index	Output	Wrapped Key target_key_type 0: 13 word size target_key_type 2: 17 word size

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_RESOURCE_CONFLICT:	Occurrence of resource conflict because a hardware resource needed by the processing routine was in use by another processing routine
TSIP_ERR_KEY_SET	Invalid wrapped key input
TSIP_ERR_FAIL	Occurrence of internal error

Description

The R_TSIP_AesXXXKeyUnwrap() function uses the first parameter, wrap_key_index, to unwrap wrapped_key, which is input as the third parameter. The unwrapped key is written to the fourth parameter, target_key_index. The processing conforms to the RFC 3394 unwrapping algorithm. Use the second parameter, target_key_type, to select the key to be unwrapped.

Use R_TSIP_Aes128KeyUnwrap() when the key length used for unwrapping is 128 bits, and use R_TSIP_Aes256KeyUnwrap() when the key length is 256 bits.

Reentrancy

Not supported.

4.2.13 TLS (Common to TLS 1.2 and TLS 1.3)

In this section, (1) means for Client and (2) means for Server in ths colimns of Format or other.

4.2.13.1 R_TSIP_GenerateTlsXXRsaPublicKeyIndex

Format

(1)
```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_GenerateTlsRsaPublicKeyIndex(
        uint8_t *encrypted_provisioning_key,
        uint8_t *iv,
        uint8_t *encrypted_key,
        tsip_tls_ca_certification_public_key_index_t *key_index
)
```

(2)
```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_GenerateTlsSVRsaPublicKeyIndex(
        uint8_t *encrypted_provisioning_key,
        uint8_t *iv,
        uint8_t *encrypted_key,
        tsip_tls_ca_certification_public_key_index_t *key_index
)
```

Parameters

encrypted_provisioning_key	Input	W-UFPK
iv	Input	Initialization vector when generating encrypted_key
encrypted_key	Input	Encrypted key encrypted using UFPK
key_index	Output	2048-bit RSA public wrapped key for use by TLS cooperation function Use this value as the key_index parameter input to R_TSIP_TlsRegistaeCaCertificationPublicKeyIndex When the wrapped key for client is generated with (1), it is possible to use this value as the key_index_1 parameter input to R_TSIP_Open.

Return Values

TSIP_SUCCESS	Normal termination
TSIP_ERR_FAIL	Occurrence of internal error
TSIP_ERR_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 outputs an RSA 2048-bit public wrapped key for use by the TLS cooperation function.

Refer to 7.3.4.2(1) for the format of the data encrypted using the UFPK input as encrypted_key.

Ensure that the areas allocated for encrypted_key and key_index do not overlap.

Refer to 3.7.1, Key Injection and Updating, for an explanation of encrypted_provisioning_key, iv, and encrypted_key and how to use key_index.

Reentrancy

Not supported.

4.2.13.2 R_TSIP_UpdateTlsXXRsaPublicKeyIndex

Format

(1)
```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_UpdateTlsRsaPublicKeyKeyIndex(
        uint8_t *iv,
        uint8_t *encrypted_key,
        tsip_tls_ca_public_key_index_t *key_index
)
```

(2)
```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_UpdateTlsSVRsaPublicKeyKeyIndex(
        uint8_t *iv,
        uint8_t *encrypted_key,
        tsip_tls_ca_public_key_index_t *key_index
)
```

Parameters

iv	Input	Initialization vector when generating encrypted_key
encrypted_key	Input	Encrypted key encrypted using KUK
key_index	Output	2048-bit RSA public wrapped key for use by TLS cooperation function Use this value as the key_index parameter input to R_TSIP_TlsRegistaeCaCertificationPublicKeyIndex When the wrapped key for client is generated with (1), it is possible to use this value as the key_index_1 parameter input to R_TSIP_Open.

Return Values

TSIP_SUCCESS	Normal termination
TSIP_ERR_FAIL	Occurrence of internal error
TSIP_ERR_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 updates a 2048-bit RSA public wrapped key for use by the TLS cooperation function.

Refer to 7.3.4.2(1) for the format of the data encrypted using the KUK input as encrypted_key.

Ensure that the areas allocated for encrypted_key and key_index do not overlap.

Refer to 3.7.1, Key Injection and Updating, for an explanation of encrypted_provisioning_key, iv, and encrypted_key and how to use key_index.

Reentrancy

Not supported.

4.2.13.3 R_TSIP_TlsRegisterCaCertificationPublicKeyIndex

Format

```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_TlsRegisterCaCertificationPublicKeyIndex(
        e_tsip_tls_mode_t mode,
        tsip_tls_ca_certification_public_key_index_t *key_index
)
```

Parameters

mode	Input	Entity type to use registering public key TSIP_TLS_MODE_CLIENT: Client, TSIP_TLS_MODE_SERVER: Server
key_index	Input	2048-bit RSA public wrapped key for use by TLS cooperation function Use the value of key_index output by R_TSIP_GenerateTlsRsaPublicKeyIndex, R_TSIP_UpdateTlsRsaPublicKeyIndex, R_TSIP_GenerateTlsSVRsaPublicKeyIndex or R_TSIP_UpdateTlsSVRsaPublicKeyIndex.

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_KEY_SET:	Invalid wrapped key input

Description

This API registers the public key for use by the TLS cooperation function.

The public key for Client can be registered by using R_TSIP_Open() function.

Reentrancy

Not supported.

4.2.13.4 R_TSIP_TlsXXRootCertificateVerification

Format

(1) #include “r_tsip_rx_if.h”
```
e_tsip_err_t R_TSIP_TlsRootCertificateVerification(
        uint32_t *public_key_type,
        uint8_t *certificate,
        uint32_t certificate_length,
        uint32_t public_key_n_start_position,
        uint32_t public_key_n_end_position,
        uint32_t public_key_e_start_position,
        uint32_t public_key_e_end_position,
        uint8_t *signature,
        uint32_t *encrypted_root_public_key
)
```
(2) #include “r_tsip_rx_if.h”
```
e_tsip_err_t R_TSIP_TlsSVRootCertificateVerification(
        uint32_t *public_key_type,
        uint8_t *certificate,
        uint32_t certificate_length,
        uint32_t public_key_n_start_position,
        uint32_t public_key_n_end_position,
        uint32_t public_key_e_start_position,
        uint32_t public_key_e_end_position,
        uint8_t *signature,
        uint32_t *encrypted_root_public_key
)
```

Parameters

public_key_type	Input	Public key type included in certificate 0: RSA 2048-bit, 2: ECC P-256
certificate	Input	Root CA certificate bundle (DER format)
certificate_length	Input	Byte length of root CA certificate bundle Maximum size is 2^31-1 byte.
public_key_n_start_position	Input	Public key start byte position relative to address specified by parameter certificate public_key_type 0: n, 2: Qx
public_key_n_end_position	Input	Public key end byte position relative to address specified by parameter certificate public_key_type 0: n, 2: Qx
public_key_e_start_position	Input	Public key start byte position relative to address specified by parameter certificate public_key_type 0: e, 2: Qy
public_key_e_end_position	Input	Public key end byte position relative to address specified by parameter certificate public_key_type 0: e, 2: Qy
signature	Input	Signature data for root CA certificate bundle The signature data size is 256 bytes. The signature format is “RSA2048 PSS with SHA256.”

encrypted_root_public_key	Output	Encrypted ECDSA P256 or RSA-2048 public key Use this value as the encrypted_input_public_key parameter input to R_TSIP_TlsXXCertificateVerification or R_TSIP_TlsXXCertificateVerificationExtension. If public_key_type is 0, 560 bytes are output, and if public_key_type is 2, 96 bytes are output.

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_FAIL:	Occurrence of internal error
TSIP_ERR_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 verifies a root CA certificate bundle for use by the TLS cooperation function.

Reentrancy

Not supported.

4.2.13.5 R_TSIP_TlsXXCertificateVerification

Format

(1) #include “r_tsip_rx_if.h”
```
e_tsip_err_t R_TSIP_TlsCertificateVerification(
        uint32_t *public_key_type,
        uint32_t *encrypted_input_public_key,
        uint8_t *certificate,
        uint32_t certificate_length,
        uint8_t *signature,
        uint32_t public_key_n_start_position,
        uint32_t public_key_n_end_position,
        uint32_t public_key_e_start_position,
        uint32_t public_key_e_end_position,
        uint32_t *encrypted_output_public_key
)
```
(2) #include “r_tsip_rx_if.h”
```
e_tsip_err_t R_TSIP_TlsSVCertificateVerification(
        uint32_t *public_key_type,
        uint32_t *encrypted_input_public_key,
        uint8_t *certificate,
        uint32_t certificate_length,
        uint8_t *signature,
        uint32_t public_key_n_start_position,
        uint32_t public_key_n_end_position,
        uint32_t public_key_e_start_position,
        uint32_t public_key_e_end_position,
        uint32_t *encrypted_output_public_key
)
```

Parameters

public_key_type	Input	Public key type included in certificate 0: RSA 2048-bit (for sha256WithRsaEncryption), 1: RSA 4096-bit (for sha256WithRsaEncryption), 2: ECC P-256 (for ecdsa-with-SHA256), 3: RSA 2048-bit (for RSASSA-PSS)
encrypted_input_public_key	Input	Encrypted public key Use the value of encrypted_root_public_key output by R_TSIP_TlsXXRootCertificateVerification or the value of encrypted_output_public_key output by R_TSIP_TlsXXCertificateVerification or R_TSIP_TlsXXCertificateVerificationExtension. Data size public_key_type 0, 1, or 3: 140 words (560 bytes), 2: 24 words (96 bytes)
certificate	Input	Certificate bundle (DER format)
certificate_length	Input	Byte length of certificate bundle Maximum size is 2^31-1 byte.

signature	Input	Signature data for certificate bundle public_key_type: 0 The data size is 256 bytes. Signature algorithm is sha256WithRSAEncryption public_key_type: 1 The data size is 512 bytes. Signature algorithm is sha256WithRSAEncryption public_key_type: 2 The data size is 64 bytes “r (256 bits) || s (256 bits)” Signature algorithm is ecdsa-with-SHA256 public_key_type: 3 The data size is 256 bytes. Signature algorithm is RSASSA-PSS {sha256, mgf1SHA256, 0x20, trailerFieldBC}
public_key_n_start_position	Input	Public key start byte position relative to address specified by parameter certificate public_key_type 0, 1, or 3: n, 2: Qx
public_key_n_end_position	Input	Public key end byte position relative to address specified by parameter certificate public_key_type 0, 1, or 3: n, 2: Qx
public_key_e_start_position	Input	Public key start byte position relative to address specified by parameter certificate public_key_type 0, 1, or 3: e, 2: Qy
public_key_e_end_position	Input	Public key end byte position relative to address specified by parameter certificate public_key_type 0, 1, or 3: e, 2: Qy
encrypted_output_public_key	Output	Encrypted public key Use this value as the encrypted_input_public_key parameter input to R_TSIP_TlsXXCertificateVerification or R_TSIP_TlsXXCertificateVerificationExtension, or as the encrypted_public_key parameter input to R_TSIP_TlsEncryptPreMasterSecretWithRsa2048PublicKey or R_TSIP_TlsServersEphemeralEcdhPublicKeyRetrives. However, it can be used only with R_TSIP_TlsXXCertificateVerification or R_TSIP_TlsXXCertificateVerificationExtension when public_key_type = 1 is selected. Data size public_key_type 0, 1, or 3: 140 words (560 bytes), 2: 24 words (96 bytes)

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_FAIL:	Occurrence of internal error
TSIP_ERR_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 verifies the signature of a server certificate or intermediate certificate used by the TLS cooperation function.

This API can be used for same purpose as the R_TSIP_TlsXXCertificateVerificationExtension() function, but make sure to use this function when the algorithm of the key used for signature verification is the same as the algorithm used to obtain the key from the certificate.

Reentrancy

Not supported.

4.2.13.6 R_TSIP_TlsXXCertificateVerificationExtension

Format

(1)
```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_TlsCertificateVerificationExtension(
        uint32_t *public_key_type,
        uint32_t *public_key_output_type,
        uint32_t *encrypted_input_public_key,
        uint8_t *certificate,
        uint32_t certificate_length,
        uint8_t *signature,
        uint32_t public_key_n_start_position,
        uint32_t public_key_n_end_position,
        uint32_t public_key_e_start_position,
        uint32_t public_key_e_end_position,
        uint32_t *encrypted_output_public_key
)
```

(2)
```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_TlsSVCertificateVerificationExtension(
        uint32_t *public_key_type,
        uint32_t *public_key_output_type,
        uint32_t *encrypted_input_public_key,
        uint8_t *certificate,
        uint32_t certificate_length,
        uint8_t *signature,
        uint32_t public_key_n_start_position,
        uint32_t public_key_n_end_position,
        uint32_t public_key_e_start_position,
        uint32_t public_key_e_end_position,
        uint32_t *encrypted_output_public_key
)
```

Parameters

public_key_type	Input	Type of public key included in input certificate 0: RSA 2048-bit (for sha256WithRsaEncryption), 1: RSA 4096-bit (for sha256WithRsaEncryption), 2: ECC P-256 (ecdsa-with-SHA256), 3: RSA 2048-bit (for RSASSA-PSS)
public_key_output_type	Input	Type of public key output from certificate 0: RSA 2048-bit (for sha256WithRsaEncryption), 1: RSA 4096-bit (for sha256WithRsaEncryption), 2: ECC P-256 (ecdsa-with-SHA256), 3: RSA 2048-bit (for RSASSA-PSS)
encrypted_input_public_key	Input	Encrypted public key Use the value of encrypted_root_public_key output by R_TSIP_TlsXXRootCertificateVerification or the value of encrypted_output_public_key output by R_TSIP_TlsXXCertificateVerification or R_TSIP_TlsXXCertificateVerificationExtension. Data size public_key_type 0, 1, or 3: 140 words (560 bytes), 2: 24 words (96 bytes)
certificate	Input	Certificate bundle (DER format)

certificate_length	Input	Byte length of certificate bundle Maximum size is 2^31-1 byte.
signature	Input	Signature data for certificate bundle public_key_type: 0 The data size is 256 bytes. Signature algorithm is sha256WithRSAEncryption public_key_type: 1 The data size is 512 bytes Signature algorithm is sha256WithRSAEncryption public_key_type: 2 The data size is 64 bytes “r (256 bits) || s (256 bits)” Signature algorithm is ecdsa-with-SHA256 public_key_type: 3 The data size is 256 bytes. Signature algorithm is RSASSA-PSS {sha256, mgf1SHA256, 0x20, trailerFieldBC}
public_key_n_start_position	Input	Public key start byte position relative to address specified by parameter certificate public_key_type 0, 1, or 3: n, 2: Qx
public_key_n_end_position	Input	Public key end byte position relative to address specified by parameter certificate public_key_type 0, 1, or 3: n, 2: Qx
public_key_e_start_position	Input	Public key start byte position relative to address specified by parameter certificate public_key_type 0, 1, or 3: e, 2: Qy
public_key_e_end_position	Input	Public key end byte position relative to address specified by parameter certificate public_key_type 0, 1, or 3: e, 2: Qy
encrypted_output_public_key	Output	Encrypted public key Use this value as the encrypted_input_public_key parameter input to R_TSIP_TlsXXCertificateVerification or R_TSIP_TlsXXCertificateVerificationExtension, or as the encrypted_public_key parameter input to R_TSIP_TlsEncryptPreMasterSecretWithRsa2048PublicKey or R_TSIP_TlsServersEphemeralEcdhPublicKeyRetrives. However, it can be used only with R_TSIP_TlsXXCertificateVerification or R_TSIP_TlsXXCertificateVerificationExtension when public_key_type = 1 is selected. Data size public_key_type 0, 1, or 3: 140 words (560 bytes), 2: 24 words (96 bytes)

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_FAIL:	Occurrence of internal error
TSIP_ERR_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 verifies the signature of a server certificate or intermediate certificate used by the TLS cooperation function.

This API can be used for same purpose as the R_TSIP_TlsXXCertificateVerification() function, but make sure to use this function when the algorithm of the key used for signature verification is different from the algorithm used to obtain the key from the certificate.

Reentrancy

Not supported.

4.2.14 TLS (TLS 1.2)

In this section, (1) means for Client and (2) means for Server in ths colimns of Format or other.

4.2.14.1 R_TSIP_TlsGeneratePreMasterSecret

Format

```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_TlsGeneratePreMasterSecret(
        uint32_t *tsip_pre_master_secret
)
```

Parameters

tsip_pre_master_secret	Output	Encrypted ephemeral pre-master secret data Use this value as the tsip_pre_master_secret parameter input to R_TSIP_TlsGenerateMasterSecret, R_TSIP_TlsEncryptPreMasterSecretWithRsa2048PublicKey, or R_TSIP_TlsGenerateExtendedMasterSecret. 20 words (80 bytes) of data is output.

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_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 generates an encrypted ephemeral pre-master secret for use by the TLS cooperation function.

Reentrancy

Not supported.

4.2.14.2 R_TSIP_TlsEncryptPreMasterSecretWithRsa2048PublicKey

Format

```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_TlsEncryptPreMasterSecretWithRsa2048PublicKey(
        uint32_t *encryted_public_key,
        uint32_t *tsip_pre_master_secret,
        uint8_t *encrypted_pre_master_secret
)
```

Parameters

encrypted_public_key	Input	Encrypted public key data Use the value of encrypted_output_public_key output by R_TSIP_TlsCertificateVerification or R_TSIP_TlsCertificateVerificationExtension. The data size is 140 words (560 bytes).
tsip_pre_master_secret	Input	Encrypted ephemeral pre-master secret data, output by R_TSIP_TlsGeneratePreMasterSecret.
encrypted_pre_master_secret	Output	Pre-master secret data that was encrypted in RSA-2048 mode using public_key

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_FAIL:	Occurrence of internal error
TSIP_ERR_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 encrypts a pre-master secret in RSA-2048 mode, using a public key from the input data, for use by the TLS cooperation function.

Reentrancy

Not supported.

4.2.14.3 R_TSIP_TlsSVGenerateServerRandom

Format

```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_TlsRegisterCaCertificationPublicKeyIndex(
        e_tsip_tls_version_t client_tls_version,
        e_tsip_tls_version_t server_tls_version,
        uint32_t gmx_unix_time,
        tsip_tls_sv_random_t *server_random
)
```

Parameters

client_tls_version	Input	Protocol version of TLS Client TSIP_TLS_VERSION_12: TLS 1.2, TSIP_TLS_VERSION_13: TLS 1.3
server_tls_version	Input	Protocol version of TLS Server TSIP_TLS_VERSION_12: TLS 1.2, TSIP_TLS_VERSION_13: TLS 1.3
gmt_unix_time	Input	Current GMT Unit time with 32 bits format
server_random	Output	Random number value to use in ServerHello and its MAC
random		Random number value to use in ServerHello The data size is 8 words (32 bytes).
mac		The MAC of random number value to use in ServerHello The data size is 4 words (12 bytes). The data does not used when TLS 1.3 is used.

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_FAIL:	Occurrence of internal error
TSIP_ERR_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 generates a random number value to use in ServerHello and its MAC used by the TLS cooperation function.

When the same random number value is used to ServerHello, TLS connection with same master secret can conducted. To prevent such attack and sure the security of the server system, this API provides random number generation function which can be used safely. And, its MAC is added to prevent the attack to change the generated random number value on purpose to use following process. Please call this API from user of the driver every time and use the latest output when the ServerHello is generating.

Reentrancy

Not supported.

4.2.14.4 R_TSIP_TlsSVDecryptPreMasterSecretWithRsa2048PrivateKey

Format

```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_TlsSVDecryptPreMasterSecretWithRsa2048PrivateKey(
        tsip_rsa2048_private_key_index_t *private_key_index,
        uint8_t *encrypted_pre_master_secret,
        uint32_t *tsip_pre_master_secret
)
```

Parameters

private_key_index	Input	Wrapped RSA private key type included in certificate Use the value of key_index output by R_TSIP_GenerateRsa2048PrivateKeyIndex.
encrypted_pre_master_secret	Input	Pre-master secret data that was encrypted in RSA-2048 mode
tsip_pre_master_secret	Output	Encrypted ephemeral ore-master secret The data size is 16 words (64 bytes).

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_KEY_SET::	Invalid wrapped key input
TSIP_ERR_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 decrypts a pre-master secret in RSA-2048 mode, using a private key from the input data, for use by the TLS cooperation function.

Reentrancy

Not supported.

4.2.14.5 R_TSIP_TlsXXGenerateMasterSecret

Format

(1) #include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_TlsGenerateMasterSecret(
uint32_t select_cipher_suite,
uint32_t *tsip_pre_master_secret,
uint8_t *client_random,
uint8_t *server_random,
uint32_t *tsip_master_secret
)

(2) #include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_TlsSVGenerateMasterSecret(
uint32_t select_cipher_suite,
uint32_t *tsip_pre_master_secret,
uint8_t *client_random,
tsip_tls_sv_random_t *server_random,
uint32_t *tsip_master_secret
)

Parameters

select_cipher_suite	Input	Cipher suite selection 0: R_TSIP_TLS_RSA_WITH_AES_128_CBC_SHA 1: R_TSIP_TLS_RSA_WITH_AES_256_CBC_SHA 2: R_TSIP_TLS_RSA_WITH_AES_128_CBC_SHA256 3: R_TSIP_TLS_RSA_WITH_AES_256_CBC_SHA256 4: R_TSIP_TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256 5: R_TSIP_TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256 6: R_TSIP_TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256 7: R_TSIP_TLS_ECDHE_RSSA_WITH_AES_128_GCM_SHA256
tsip_pre_master_secret	Input	Encrypted ephemeral pre-master secret data (1) Use the value of tsip_pre_master_secret output by R_TSIP_TlsGeneratePreMasterSecret or encrypted_pre_master_secret output by R_TSIP_TlsGeneratePreMasterSecretWithEccP256Key. (2) Use the value of tsip_pre_master_secret output by R_TSIP_TlsSVDecryptPreMasterSecretWithRsa2048PrivateKey or R_TSIP_TlsSVGeneratePreMasterSecretWithEccP256Key.
client_random	Input	32-byte random number value reported by ClientHello
server_random	Input	(1) 32-byte random number value reported by ServerHello (2) 32-byte random number value reported by ServerHello and its MAC
tsip_master_secret	Output	Encrypted ephemeral master secret data Use this value as the tsip_master_secret parameter input to R_TSIP_TlsXXGenerateSessionKey or R_TSIP_TlsXXGenerateVerifyData. 20 words (80 bytes) of data is output.

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_FAIL:	Occurrence of internal error
TSIP_ERR_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 generates an encrypted master secret for use by the TLS cooperation function.

Reentrancy

Not supported.

4.2.14.6 R_TSIP_TlsXXGenerateSessionKey

Format

(1) #include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_TlsGenerateSessionKey(
uint32_t select_cipher_suite,
uint32_t *tsip_master_secret,
uint8_t *client_random,
uint8_t *server_random,
uint8_t *nonce_explicit,
tsip_hmac_sha_key_index_t *client_mac_key_index,
tsip_hmac_sha_key_index_t *server_mac_key_index,
tsip_aes_key_index_t *client_crypto_key_index,
tsip_aes_key_index_t *server_crypto_key_index,
uint8_t *client_iv,
uint8_t *server_iv
)

(2) #include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_TlsSVGenerateSessionKey(
uint32_t select_cipher_suite,
uint32_t *tsip_master_secret,
uint8_t *client_random,
tsip_tls_sv_random_t *server_random,
uint8_t *nonce_explicit,
tsip_hmac_sha_key_index_t *client_mac_key_index,
tsip_hmac_sha_key_index_t *server_mac_key_index,
tsip_aes_key_index_t *client_crypto_key_index,
tsip_aes_key_index_t *server_crypto_key_index,
uint8_t *client_iv,
uint8_t *server_iv
)

Parameters

select_cipher_suite	Input	Cipher suite selection 0: R_TSIP_TLS_RSA_WITH_AES_128_CBC_SHA 1: R_TSIP_TLS_RSA_WITH_AES_256_CBC_SHA 2: R_TSIP_TLS_RSA_WITH_AES_128_CBC_SHA256 3: R_TSIP_TLS_RSA_WITH_AES_256_CBC_SHA256 4: R_TSIP_TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256 5: R_TSIP_TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256 6: R_TSIP_TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256 7: R_TSIP_TLS_ECDHE_RSSA_WITH_AES_128_GCM_SHA256
tsip_master_secret	Input	Encrypted ephemeral master secret data Use the value of tsip_master_secret output by R_TSIP_TlsXXGenerateMasterSecret.
client_random	Input	32-byte random number value reported by ClientHello
server_random	Input	(1) 32-byte random number value reported by ServerHello (2) 32-byte random number value reported by ServerHello and its MAC
nonce_explicit	Input	Nonce used by AES128 GCM cipher suite select_cipher_suite = 6-7: 8 bytes
client_mac_key_index	Output	MAC wrapped key for client to server communication

server_mac_key_index	Output	MAC wrapped key for server to client communication
client_crypt_key_index	Output	AES common wrapped key for client to server communication
server_crypt_key_index	Output	AES common wrapped key for server to client communication
client_iv	Output	IV used for transmission from client to server Output when the value of select_cipher_suite is 0 to 5. ((1) Used when NetX Duo is employed on the RX651 or RX65N.) Otherwise, nothing is output.
server_iv	Output	IV used for reception from server Output when the value of select_cipher_suite is 0 to 5. ((1) Used when NetX Duo is employed on the RX651 or RX65N.) Otherwise, nothing is output.

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_FAIL:	Occurrence of internal error
TSIP_ERR_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 outputs TLS common keys for use by the TLS cooperation function.

Nothing is output for the client_iv and server_iv parameters when the status is other than that specified in the parameter explanation.

Reentrancy

Not supported.

4.2.14.7 R_TSIP_TlsXXGenerateVerifyData

Format

(1)
```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_TlsGenerateVerifyData(
        uint32_t select_verify_data,
        uint32_t *tsip_master_secret,
        uint8_t *hand_shake_hash,
        uint8_t *verify_data
)
```

(2)
```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_TlsSVGenerateVerifyData(
        uint32_t select_verify_data,
        uint32_t *tsip_master_secret,
        uint8_t *hand_shake_hash,
        uint8_t *verify_data
)
```

Parameters

select_verify_data	Input	Client/server selection R_TSIP_TLS_GENERATE_CLIENT_VERIFY: Client verification data is generated. R_TSIP_TLS_GENERATE_SERVER_VERIFY: Server verify data is generated.
tsip_master_secret	Input	Encrypted ephemeral master secret data Use the value of tsip_master_secret output by R_TSIP_TlsXXGenerateMasterSecret.
hand_shake_hash	Input	SHA256 hash value for entire TLS handshake message
verify_data	Output	Verification data for Finished message

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_FAIL:	Occurrence of internal error
TSIP_ERR_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 generates verify data for use by the TLS cooperation function.

Reentrancy

Not supported.

4.2.14.8 R_TSIP_TlsServersEphemeralEcdhPublicKeyRetrieves

Format

```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_TlsServersEphemeralEcdhPublicKeyRetrieves(
        uint32_t public_key_type,
        uint8_t *client_random,
        uint8_t *server_random,
        uint8_t *server_ephemeral_ecdh_public_key,
        uint8_t *server_key_exchange_signature,
        uint32_t *encrypted_public_key,
        uint32_t *encrypted_ephemeral_ecdh_public_key
)
```

Parameters

public_key_type	Input	Public key type 0: RSA 2048-bit (rsa_pkcs1_sha256), 1: RSA 4096-bit (rsa_pkcs1_sha256), 2: ECDSA P-256 3: RSA 2048-bit (rsa_pss_rsae_sha256)
client_random	Input	32-byte random number value reported by ClientHello
server_random	Input	32-byte random number value reported by ServerHello
server_ephemeral_ecdh_public_key	Input	Ephemeral ECDH public key (uncompressed format) received from server 0 padding (24 bits) || 04 (8 bits) || Qx (256 bits) || Qy (256 bits)
server_key_exchange_signature	Input	ServerKeyExchange signature data public_key_type 0: 256 bytes, 2: 64 bytes
encrypted_public_key	Input	Encrypted public key for signature verification Use the value of encrypted_output_public_key output by R_TSIP_TlsCertificateVerification or R_TSIP_TlsCertificateVerificationExtension. public_key_type 0: 140 words (560 bytes), 2:24 words (96 bytes)
encrypted_ephemeral_ecdh_public_key	Output	Encrypted ephemeral ECDH public key used by R_TSIP_TlsGeneratePreMasterSecretWithEccP256Key The data size is 24 words (96 bytes) size

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_FAIL:	Occurrence of internal error
TSIP_ERR_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 verifies a ServerKeyExchange signature, using the input public key data, for use by the TLS cooperation function. If the signature is verified successfully, the ephemeral ECDH public key used by R_TSIP_TlsGeneratePreMasterSecretWithEccP256Key is encrypted and output.

Applicable cypher suites: TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256,
TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256,
TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256,
TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256

Reentrancy

Not supported.

4.2.14.9 R_TSIP_GenerateTlsXXP256EccKeyIndex

Format

(1) #include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_ GenerateTlsP256EccKeyIndex(
tsip_tls_p256_ecc_key_index_t *tls_p256_ecc_key_index,
uint8_t *ephemeral_ecdh_public_key
)

(2) #include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_ GenerateTlsSVP256EccKeyIndex(
tsip_tls_p256_ecc_key_index_t *tls_p256_ecc_key_index,
uint8_t *ephemeral_ecdh_public_key
)

Parameters

tls_p256_ecc_key_index	Output	Ephemeral ECC secret wrapped key Use this value as the tls_p256_ecc_key_index parameter input to R_TSIP_TlsXXGeneratePreMasterSecretWithEccP256Key.
ephemeral_ecdh_public_key	Output	(1) Ephemeral ECDH public key to be sent to the server (2) Ephemeral ECDH public key to be sent to the client Both are Qx (256 bits) || Qy (256 bits).

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_FAIL:	Occurrence of internal error
TSIP_ERR_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 generates a key pair from a random number for use by the TLS cooperation function for elliptic curve cryptography over a 256-bit prime field.

Applicable cypher suites: TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256,
TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256,
TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256,
TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256

Reentrancy

Not supported.

4.2.14.10 R_TSIP_TlsXXGeneratePreMasterSecretWithEccP256Key

Format

(1) #include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_TlsGeneratePreMasterSecretWithEccP256Key(
uint32_t *encrypted_public_key,
tsip_tls_p256_ecc_key_index_t *tls_p256_ecc_key_index,
uint32_t *tsip_pre_master_secret
)

(2) #include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_TlsSVGeneratePreMasterSecretWithEccP256Key(
uint32_t *ecdh_public_key,
tsip_tls_p256_ecc_key_index_t *tls_p256_ecc_key_index,
uint32_t *tsip_pre_master_secret
)

Parameters

(1) encrypted_public_key (2) ecdh_public_key	Input	Ephemeral ECDH public key (1) Use the value of encrypted_ephemeral_ecdh_public_key output by R_TSIP_TlsServersEphemeralEcdhPublicKeyRetrieves. (2) Use the value of ephemeral ECDH public key with its format is Qx (256 bits) || Qy (256 bits).
tls_p256_ecc_key_index	Input	Ephemeral ECC secret wrapped key Use the value of tls_p256_ecc_key_index output by R_TSIP_GenerateTlsXXP256EccKeyIndex.
tsip_pre_master_secret	Output	Encrypted ephemeral pre-master secret data 16 words (64 bytes) of data is output.

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_FAIL:	Occurrence of internal error
TSIP_ERR_RESOURCE_CONFLICT:	Occurrence of resource conflict because a hardware resource needed by the processing routine was in use by another processing routine
TSIP_ERR_KEY_SET:	Invalid wrapped key input

Description

This API generates an encrypted pre-master secret, using the input key data, for use by the TLS cooperation function.

Applicable cypher suites: TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256,
TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256,
TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256,
TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256

Reentrancy

Not supported.

4.2.14.11 R_TSIP_TlsXXGenerateExtendedMasterSecret

Format

(1) #include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_TlsGenerateExtendedMasterSecret(
uint32_t select_cipher_suite,
uint32_t *tsip_pre_master_secret,
uint8_t *digest,
uint32_t *extended_master_secret
)

(2) #include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_TlsSVGenerateExtendedMasterSecret(
uint32_t select_cipher_suite,
uint32_t *tsip_pre_master_secret,
uint8_t *digest,
uint32_t *extended_master_secret
)

Parameters

select_cipher_suite	Input	Cipher suite selection 2: R_TSIP_TLS_RSA_WITH_AES_128_CBC_SHA256 3: R_TSIP_TLS_RSA_WITH_AES_256_CBC_SHA256 4: R_TSIP_TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256 5: R_TSIP_TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256 6: R_TSIP_TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256 7: R_TSIP_TLS_ECDHE_RSSA_WITH_AES_128_GCM_SHA256
tsip_pre_master_secret	Input	Encrypted ephemeral pre-master secret data (1) Use the value of tsip_pre_master_secret output by R_TSIP_TlsGeneratePreMasterSecret or R_TSIP_TlsGeneratePreMasterSecretWithEccP256Key. (2) Use the value of tsip_pre_master_secret output by R_TSIP_TlsSVDecryptPreMasterSecretWithRsa2048PrivateKey or R_TSIP_TlsSVGeneratePreMasterSecretWithEccP256Key.
digest	Input	Message hash calculated using SHA256 Calculate and input a hash value of the value of concatenated handshake messages such as (ClientHello||ServerHello||Certificate||ServerKeyExchange ||CertificateRequest||ServerHelloDone||Certificate ||ClientKeyExchange). Use R_TSIP_Sha256Init/Update/Final to calculate the hash value and input as digest the value output by R_TSIP_Sha256Final.
extended_master_secret	Output	Encrypted ephemeral extended master secret data 20 words (80 bytes) of data is output. Use this value as the tsip_master_secret parameter input to R_TSIP_TlsXXGenerateSessionKey or R_TSIP_TlsXXGenerateVerifyData.

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_FAIL:	Occurrence of internal error
TSIP_ERR_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 generates encrypted extended master secret data, using encrypted pre-master secret data, for use by the TLS cooperation function.

Reentrancy

Not supported.

4.2.14.12 R_TSIP_TlsSVCertificateVerifyVerification

Format

```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_TlsSVCertificateVerifyVerification(
        uint32_t *key_index,
        e_tsip_tls_signature_scheme_type_t signature_scheme,
        uint8_t *handshake_message,
        uint32_t handshake_message_len,
        uint8_t *certificate_verify,
        uint32_t certificate_verify_len
)
```

Parameters

key_index	Input	Encrypted public key Use encrypted_output_public_key output by R_TSIP_TlsCertificateVerification or R_TSIP_TlsCertificateVerificationExtension.
signature_scheme	Input	Signature algorithm to be used TSIP_TLS_SIGNATURE_SCHEME_RSA_PKCS1_SHA256: rsa_pkcs1_sha256 TSIP_TLS_SIGNATURE_SCHEME_ECDSA_SECP256R1_SHA256: ecdsa_secp256r1_sha256 TSIP_TLS_SIGNATURE_SCHEME_RSA_PSS_RSAE_SHA256: rsa_pss_rsae_sha256
digest	Input	Handshake message Input a value of concatenated handshake messages such as (ClientHello||ServerHello||Certificate||ServerKeyExchange ||CertificateRequest||ServerHelloDone||CertificateClientKey Exchange). Use R_TSIP_Sha256Init/Update/Final to calculate the hash value and input as digest the value output by R_TSIP_Sha256Final.
handshake_message_len	Input	Byte length of handshake_message
certificate_verify	Input	CertificateVerify Input the start address of the buffer for storing the data.
certificate_verify_len	Input	Byte length of certificate_verify

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_FAIL:	Occurrence of internal error
TSIP_ERR_RESOURCE_CONFLICT:	Occurrence of resource conflict because a hardware resource needed by the processing routine was in use by another processing routine
TSIP_ERR_PARAMETER	Invalid input data

Description

This API verifies a CertificateVerify message received from the client for use by the server function of the TLS cooperation function.

This API can be used for same purpose as the R_TSIP_TlsCertificateVerificationExtension() function, but make sure to use this function when the algorithm of the key used for signature verification is the same as the algorithm used to obtain the key from the certificate.

Reentrancy

Not supported.

4.2.15 TLS (TLS 1.3)

4.2.15.1 R_TSIP_GenerateTls13P256EccKeyIndex

Format

```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_ TlsGenerateTls13P256EccKeyIndex(
        tsip_tls13_handle_t *handle,
        e_tsip_tls13_mode_t mode,
        tsip_tls_p256_ecc_key_index_t *key_index,
        uint8_t *ephemeral_ecdh_public_key
)
```

Parameters

handle	Input	Handle number (work area) indicating same session
mode	Input	Handshake protocol to use TSIP_TLS13_MODE_FULL_HANDSHAKE: Full handshake TSIP_TLS13_MODE_RESUMPTION: Resumption TSIP_TLS13_MODE_0_RTT: 0-RTT
key_index	Output	Ephemeral ECC secret wrapped key Use this value as the key_index parameter input to R_TSIP_Tls13GenerateEcdheSharedSecret.
ephemeral_ecdh_public_key	Output	Ephemeral ECDH public key to be sent to the server Qx (256 bits) || Qy (256 bits)

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_FAIL:	Occurrence of internal error
TSIP_ERR_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 generates a key pair from a random number for use by the TLS 1.3 cooperation function for elliptic curve cryptography over a 256-bit prime field.

Reentrancy

Not supported.

4.2.15.2 R_TSIP_Tls13GenerateEcdheSharedSecret

Format

```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_ Tls13GenerateEcdheSharedSecret(
        e_tsip_tls13_mode_t mode,
        uint8_t *server_public_key,
        tsip_tls_p256_ecc_key_index_t *key_index,
        tsip_tls13_ephemeral_shared_secret_key_index_t *shared_secret_key_index
)
```

Parameters

mode	Input	Handshake protocol to use TSIP_TLS13_MODE_FULL_HANDSHAKE: Full handshake TSIP_TLS13_MODE_RESUMPTION: Resumption TSIP_TLS13_MODE_0_RTT: 0-RTT
server_public_key	Input	Public key provided by the server Qx (256 bits) || Qy (256 bits)
key_index	Input	Ephemeral ECC secret wrapped key Use the value of key_index output by R_TSIP_GenerateTls13P256EccKeyIndex.
shared_secret_key_index	Output	Shared secret ephemeral wrapped key Use this value as the shared_secret_key_index parameter input to R_TSIP_Tls13GenerateHandshakeSecret and R_TSIP_Tls13GenerateResumptionHandshakeSecret.

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_FAIL:	Occurrence of internal error
TSIP_ERR_RESOURCE_CONFLICT:	Occurrence of resource conflict because a hardware resource needed by the processing routine was in use by another processing routine
TSIP_ERR_KEY_SET	Invalid wrapped key input

Description

This API calculates a shared secret, which is a shared key over a 256-bit prime field, using a public key provided by the server and a previously calculated secret key, and generates a wrapped key for use by the TLS 1.3 cooperation function.

Applicable cypher suites: TLS_AES_128_GCM_SHA256, TLS_AES_128_CCM_SHA256

Key exchange format: ECDHE NIST P-256

Reentrancy

Not supported.

4.2.15.3 R_TSIP_Tls13GenerateHandshakeSecret

Format

```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_ Tls13GenerateHandshakeSecret(
        tsip_tls13_ephemeral_shared_secret_key_index_t *shared_secret_key_index,
        tsip_tls13_ephemeral_handshake_secret_key_index_t *handshake_secret_key_index
)
```

Parameters

shared_secret_key_index	Input	Shared secret ephemeral wrapped key Use the value of shared_secret_key_index output by R_TSIP_Tls13GenerateEcdheSharedSecret.
handshake_secret_key_index	Output	Handshake secret ephemeral wrapped key Use this value as the handshake_secret_key_index parameter input to R_TSIP_Tls13GenerateServerHandshakeTrafficKey, R_TSIP_Tls13GenerateClientHandshakeTrafficKey, and R_TSIP_Tls13GenerateMasterSecret.

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_FAIL:	Occurrence of internal error
TSIP_ERR_RESOURCE_CONFLICT:	Occurrence of resource conflict because a hardware resource needed by the processing routine was in use by another processing routine
TSIP_ERR_KEY_SET	Invalid wrapped key input

Description

This API generates a handshake secret wrapped key, using a shared secret ephemeral key, for use by the TLS 1.3 cooperation function.

Reentrancy

Not supported.

4.2.15.4 R_TSIP_Tls13GenerateServerHandshakeTrafficKey

Format

```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_Tls13GenerateServerHandshakeTrafficKey(
        tsip_tls13_handle_t *handle,
        e_tsip_tls13_mode_t mode,
        tsip_tls13_ephemeral_handshake_secret_key_index_t *handshake_secret_key_index,
        uint8_t *digest,
        tsip_aes_key_index_t *server_write_key_index,
        tsip_tls13_ephemeral_server_finished_key_index_t *server_finished_key_index
)
```

Parameters

handle	Output	Handle number (work area) indicating same session
mode	Input	Handshake protocol to use TSIP_TLS13_MODE_FULL_HANDSHAKE: Full handshake TSIP_TLS13_MODE_RESUMPTION: Resumption TSIP_TLS13_MODE_0_RTT: 0-RTT
handshake_secret_key_index	Input	Handshake secret ephemeral wrapped key Use the value of handshake_secret_key_index output by R_TSIP_Tls13GenerateHandshakeSecret or R_TSIP_Tls13GenerateResumptionHandshakeSecret.
digest	Input	Message hash calculated using SHA256 Calculate and input the hash value of (ClientHello||ServerHello). Use R_TSIP_Sha256Init/Update/Final to calculate the hash value and input as digest the value output by R_TSIP_Sha256Final.
server_write_key_index	Output	Server write key ephemeral wrapped key Use this value as the key_index parameter input to R_TSIP_Tls13DecryptInit.
server_finished_key_index	Output	Server finished key ephemeral wrapped key Use this value as the server_finished_key_index parameter input to R_TSIP_Tls13ServerHandshakeVerification.

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_RESOURCE_CONFLICT:	Occurrence of resource conflict because a hardware resource needed by the processing routine was in use by another processing routine
TSIP_ERR_KEY_SET	Invalid wrapped key input

Description

This API generates, using the handshake secret output by R_TSIP_Tls13GenerateHandshakeSecret, a server write wrapped key and server finished wrapped key for use by the TLS 1.3 cooperation function.

Reentrancy

Not supported.

4.2.15.5 R_TSIP_Tls13GenerateClientHandshakeTrafficKey

Format

```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_Tls13GenerateClientHandshakeTrafficKey(
        tsip_tls13_handle_t *handle,
        e_tsip_tls13_mode_t mode,
        tsip_tls13_ephemeral_handshake_secret_key_index_t *handshake_secret_key_index,
        uint8_t *digest,
        tsip_aes_key_index_t *client_write_key_index,
        tsip_hmac_sha_key_index_t *client_finished_key_index
)
```

Parameters

handle	Output	Handle number (work area) indicating same session
mode	Input	Handshake protocol to use TSIP_TLS13_MODE_FULL_HANDSHAKE: Full handshake TSIP_TLS13_MODE_RESUMPTION: Resumption TSIP_TLS13_MODE_0_RTT: 0-RTT
handshake_secret_key_index	Input	Handshake secret ephemeral wrapped key Use the value of handshake_secret_key_index output by R_TSIP_Tls13GenerateHandshakeSecret or R_TSIP_Tls13GenerateResumptionHandshakeSecret.
digest	Input	Message hash calculated using SHA256 Calculate and input the hash value of (ClientHello||ServerHello). Use R_TSIP_Sha256Init/Update/Final to calculate the hash value and input as digest the value output by R_TSIP_Sha256Final.
client_write_key_index	Output	Client write key ephemeral wrapped key Use this value as the key_index parameter input to R_TSIP_Tls13EncryptInit.
client_finished_key_index	Output	Client finished key ephemeral wrapped key Use this value as the key_index parameter input to R_TSIP_Sha256HmacGenerateInit.

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_RESOURCE_CONFLICT:	Occurrence of resource conflict because a hardware resource needed by the processing routine was in use by another processing routine
TSIP_ERR_KEY_SET	Invalid wrapped key input

Description

This API generates, using the handshake secret output by R_TSIP_Tls13GenerateHandshakeSecret, a client write wrapped key and client finished wrapped key for use by the TLS 1.3 cooperation function.

Reentrancy

Not supported.

4.2.15.6 R_TSIP_Tls13ServerHandshakeVerification

Format

```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_ Tls13ServerHandshakeVerification(
        e_tsip_tls13_mode_t mode,
        tsip_tls13_ephemeral_server_finished_key_index_t *server_finished_key_index,
        uint8_t *digest,
        uint8_t *server_finished,
        uint32_t *verify_data_index
)
```

Parameters

mode	Input	Handshake protocol to use TSIP_TLS13_MODE_FULL_HANDSHAKE: Full handshake TSIP_TLS13_MODE_RESUMPTION: Resumption TSIP_TLS13_MODE_0_RTT: 0-RTT
server_finished_key_index	Input	Server finished key ephemeral wrapped key Use the value of server_finished_key_index output by R_TSIP_Tls13GenerateServerHandshakeTrafficKey.
digest	Input	Message hash calculated using SHA256 Calculate and input a hash value of the value of concatenated handshake messages such as (ClientHello||ServerHello||EncryptedExtensions ||CertificateRequest||Certificate||CertificateVerify). Use the value of R_TSIP_Sha256Init/Update/Final to calculate the hash value and input as digest the value output by R_TSIP_Sha256Final.
server_finished	Input	Finished information provided by the server Input the start address of the buffer for storing the ServerFinished data obtained from R_TSIP_Tls13DecryptUpdate/Final.
verify_data_index	Output	ServerHandshake verification result conforming to TSIP-specific specification Use this value as the as verify_data_index parameter input to R_TSIP_Tls13GenerateMasterSecret. Input the start address of the buffer to which data is to be output. The size must be 8 words (32 bytes).

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_FAIL	Occurrence of internal error
TSIP_ERR_RESOURCE_CONFLICT:	Occurrence of resource conflict because a hardware resource needed by the processing routine was in use by another processing routine

TSIP_ERR_KEY_SET	Invalid wrapped key input
TSIP_ERR_VERIFICATION_FAIL	Verification failure

Description

This API verifies handshake messages, using the Finished information provided by the server, for use by the TLS 1.3 cooperation function.

Reentrancy

Not supported.

4.2.15.7 R_TSIP_Tls13GenerateMasterSecret

Format

```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_ Tls13GenerateMasterSecret(
        tsip_tls13_handle_t *handle,
        e_tsip_tls13_mode_t mode,
        tsip_tls13_ephemeral_handshake_secret_key_index_t *handshake_secret_key_index,
        uint32_t *verify_data_index,
        tsip_tls13_ephemeral_master_secret_key_index_t *master_secret_key_index
)
```

Parameters

handle	Output	Handle number (work area) indicating same session
mode	Input	Handshake protocol to use TSIP_TLS13_MODE_FULL_HANDSHAKE: Full handshake TSIP_TLS13_MODE_RESUMPTION: Resumption TSIP_TLS13_MODE_0_RTT: 0-RTT
handshake_secret_key_index	Input	Handshake secret ephemeral wrapped key Use the value of handshake_secret_key_index output by R_TSIP_Tls13GenerateHandshakeSecret.
verify_data_index	Input	ServerHandshake verification result conforming to TSIP-specific specification Use the value of verify_data_index output by R_TSIP_Tls13ServerHandshakeVerification.
master_secret_key_index	Output	Master secret ephemeral wrapped key Use this value as the master_secret_key_index parameter input to R_TSIP_Tls13GenerateApplicationTrafficKey and R_TSIP_Tls13GenerateResumptionMasterSecret

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_FAIL	Occurrence of internal error
TSIP_ERR_RESOURCE_CONFLICT:	Occurrence of resource conflict because a hardware resource needed by the processing routine was in use by another processing routine
TSIP_ERR_KEY_SET	Invalid wrapped key input

Description

This API generates a master secret ephemeral wrapped key, using a handshake secret ephemeral key, for use by the TLS 1.3 cooperation function.

Reentrancy

Not supported.

4.2.15.8 R_TSIP_Tls13GenerateApplicationTrafficKey

Format

```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_Tls13GenerateApplicationTrafficKey(
        tsip_tls13_handle_t *handle,
        e_tsip_tls13_mode_t mode,
        tsip_tls13_ephemeral_master_secret_key_index_t *master_secret_key_index,
        uint8_t *digest,
        tsip_tls13_ephemeral_app_secret_key_index_t *server_app_secret_key_index,
        tsip_tls13_ephemeral_app_secret_key_index_t *client_app_secret_key_index,
        tsip_aes_key_index_t *server_write_key_index,
        tsip_aes_key_index_t *client_write_key_index
)
```

Parameters

handle	Input/output	Handle number (work area) indicating same session
mode	Input	Handshake protocol to use TSIP_TLS13_MODE_FULL_HANDSHAKE: Full handshake TSIP_TLS13_MODE_RESUMPTION: Resumption TSIP_TLS13_MODE_0_RTT: 0-RTT
master_secret_key_index	Input	Master secret ephemeral wrapped key Use the value of master_secret_key_index output by R_TSIP_Tls13GenerateMasterSecret.
digest	Input	Message hash calculated using SHA256 Calculate and input a hash value of the value of concatenated handshake messages such as (ClientHello||ServerHello||EncryptedExtensions ||CertificateRequest||Certificate||CertificateVerify ||ServerFinished). Use R_TSIP_Sha256Init/Update/Final to calculate the hash value and input as digest the value output by R_TSIP_Sha256Final.
server_app_secret_key_index	Output	Server application traffic secret ephemeral wrapped key Use this value as the input_app_secret_key_index parameter input to R_TSIP_Tls13UpdateApplicationTrafficKey.
client_app_secret_key_index	Output	Client application traffic secret ephemeral wrapped key Use this value as the input_app_secret_key_index parameter input to R_TSIP_Tls13UpdateApplicationTrafficKey.
server_write_key_index	Output	Server write key ephemeral wrapped key Use this value as the key_index parameter input to R_TSIP_Tls13DecryptInit.

client_write_key_index	Output	Client write key ephemeral wrapped key Use this value as the key_index parameter input to R_TSIP_Tls13EncryptInit.

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_RESOURCE_CONFLICT:	Occurrence of resource conflict because a hardware resource needed by the processing routine was in use by another processing routine
TSIP_ERR_KEY_SET	Invalid wrapped key input

Description

This API generates an application traffic secret wrapped key, using a master secret ephemeral key, for use by the TLS 1.3 cooperation function. It also generates server write key and client write key ephemeral wrapped keys.

Reentrancy

Not supported.

4.2.15.9 R_TSIP_Tls13UpdateApplicationTrafficKey

Format

```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_Tls13UpdateApplicationTrafficKey(
        tsip_tls13_handle_t *handle,
        e_tsip_tls13_mode_t mode,
        e_tsip_tls13_update_key_type_t key_type,
        tsip_tls13_ephemeral_app_secret_key_index_t *input_app_secret_key_index,
        tsip_tls13_ephemeral_app_secret_key_index_t *output_app_secret_key_index,
        tsip_aes_key_index_t *app_write_key_index
)
```

Parameters

handle	Input/output	Handle number (work area) indicating same session
mode	Input	Handshake protocol to use TSIP_TLS13_MODE_FULL_HANDSHAKE: Full handshake TSIP_TLS13_MODE_RESUMPTION: Resumption TSIP_TLS13_MODE_0_RTT: 0-RTT
key_type	Input	Type of key to be updated TSIP_TLS13_UPDATE_SERVER_KEY: Server application traffic secret TSIP_TLS13_UPDATE_CLIENT_KEY: Client application traffic secret
input_app_secret_key_index	Input	Ephemeral wrapped key of input server/client application traffic secret Use as input either server/client_app_secret_key_index output by R_TSIP_Tls13GenerateApplicationTrafficKey or output_app_secret_key_index output by R_TSIP_Tls13UpdateApplicationTrafficKey, whichever matches the type of key specified by key_type.
output_app_secret_key_index	Output	Ephemeral wrapped key of output server/client application traffic secret Output matching the type of key specified by key_type is obtained. Use this value as the input_app_secret_key_index parameter input to R_TSIP_Tls13UpdateApplicationTrafficKey.
app_write_key_index	Output	Server/client write key ephemeral wrapped key Output matching the type of key specified by key_type is obtained. Use ServerWriteKey as the key_index parameter input to R_TSIP_Tls13DecryptInit. Use ClientWriteKey as the key_index parameter input to R_TSIP_Tls13EncryptInit.

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_FAIL	Occurrence of internal error
TSIP_ERR_RESOURCE_CONFLICT:	Occurrence of resource conflict because a hardware resource needed by the processing routine was in use by another processing routine
TSIP_ERR_KEY_SET	Invalid wrapped key input
TSIP_ERR_PARAMETER	Invalid input data

Description

This API updates, using an application traffic secret, an encryption wrapped key corresponding to an application traffic secret wrapped key for use by the TLS 1.3 cooperation function.

Reentrancy

Not supported.

4.2.15.10 R_TSIP_Tls13GenerateResumptionMasterSecret

Format

```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_ Tls13GenerateResumptionMasterSecret(
        tsip_tls13_handle_t *handle,
        e_tsip_tls13_mode_t mode,
        tsip_tls13_ephemeral_master_secret_key_index_t *master_secret_key_index,
        uint8_t *digest,
        tsip_tls13_ephemeral_res_master_secret_key_index_t *res_master_secret_key_index
)
```

Parameters

handle	Input	Handle number (work area) indicating same session
mode	Input	Handshake protocol to use TSIP_TLS13_MODE_FULL_HANDSHAKE: Full handshake TSIP_TLS13_MODE_RESUMPTION: Resumption TSIP_TLS13_MODE_0_RTT: 0-RTT
master_secret_key_index	Input	Handshake secret ephemeral wrapped key Use the value of handshake_secret_key_index output by R_TSIP_Tls13GenerateHandshakeSecret.
digest	Input	Message hash calculated using SHA256 Calculate and input a hash value of the value of concatenated handshake messages such as (ClientHello||ServerHello||EncryptedExtensions ||CertificateRequest||Certificate||CertificateVerify ||ServerFinished||Certificate||CertificateVerify ||ClientFinished). Use R_TSIP_Sha256Init/Update/Final to calculate the hash value and input as digest the value output by R_TSIP_Sha256Final.
res_master_secret_key_index	Output	Resumption master secret ephemeral wrapped key Use this value as the res_master_secret_key_index parameter input to R_TSIP_Tls13GeneratePreSharedKey.

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_RESOURCE_CONFLICT:	Occurrence of resource conflict because a hardware resource needed by the processing routine was in use by another processing routine
TSIP_ERR_KEY_SET	Invalid wrapped key input

Description

This API generates a resumption master secret wrapped key, using a master secret ephemeral key, for use by the TLS 1.3 cooperation function.

As specified in RFC 8446, delete master_secret_key_index, the master secret ephemeral wrapped key, after generating a resumption master secret wrapped key using this API.

Reentrancy

Not supported.

4.2.15.11 R_TSIP_Tls13GeneratePreSharedKey

Format

```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_ Tls13GeneratePreSharedKey(
        tsip_tls13_handle_t *handle,
        e_tsip_tls13_mode_t mode,
        tsip_tls13_ephemeral_res_master_secret_key_index_t *res_master_secret_key_index,
        uint8_t *ticket_nonce,
        uint32_t ticket_nonce_len,
        tsip_tls13_ephemeral_pre_shared_key_index_t *pre_shared_key_index
)
```

Parameters

handle	Input	Handle number (work area) indicating same session
mode	Input	Handshake protocol to use TSIP_TLS13_MODE_FULL_HANDSHAKE: Full handshake TSIP_TLS13_MODE_RESUMPTION: Resumption TSIP_TLS13_MODE_0_RTT: 0-RTT
res_master_secret_key_index	Input	Resumtion master secret ephemeral wrapped key Use the value of res_master_secret_key_index output by R_TSIP_Tls13GenerateResumptionMasterSecret.
ticket_nonce	Input	Ticket nonce provided by the server If the size of the ticket nonce is not a multiple of 16 bytes, pad it with zeros to make it a multiple of 16 bytes before input.
ticket_nonce_len	Input	Byte length of ticket nonce Maximum size is 255 byte.
pre_shared_key_index	Output	Pre-shared key ephemeral wrapped key Use this value as the pre_shared_key_index parameter input to R_TSIP_Tls13GeneratePskBinderKey, R_TSIP_Tls13GenerateResumptionHandshakeSecret, and R_TSIP_Tls13Generate0RttApplicationWriteKey.

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_FAIL	Occurrence of internal error
TSIP_ERR_RESOURCE_CONFLICT:	Occurrence of resource conflict because a hardware resource needed by the processing routine was in use by another processing routine
TSIP_ERR_KEY_SET	Invalid wrapped key input

Description

This API generates, using a resumption master secret ephemeral key, a pre-shared wrapped key from new session ticket information for use by the TLS 1.3 cooperation function.

Reentrancy

Not supported.

4.2.15.12 R_TSIP_Tls13GeneratePskBinderKey

Format

```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_ Tls13GeneratePskBinderKey(
        tsip_tls13_handle_t *handle,
        tsip_tls13_ephemeral_pre_shared_key_index_t *pre_shared_key_index,
        tsip_hmac_sha_key_index_t *psk_binder_key_index
)
```

Parameters

handle	Input	Handle number (work area) indicating same session
pre_shared_key_index	Input	Pre-shared key ephemeral wrapped key Use the value of pre_shared_key_index output by R_TSIP_Tls13GeneratePreSharedKey.
psk_binder_key_index	Output	PSK binder key ephemeral wrapped key Use this value to generate PskBinder. Use this value as the key_index parameter input to R_TSIP_Sha256HmacGenerateInit.

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_FAIL	Occurrence of internal error
TSIP_ERR_RESOURCE_CONFLICT:	Occurrence of resource conflict because a hardware resource needed by the processing routine was in use by another processing routine
TSIP_ERR_KEY_SET	Invalid wrapped key input

Description

This API generates binder wrapped key for use by the TLS 1.3 cooperation function.

Reentrancy

Not supported.

4.2.15.13 R_TSIP_Tls13GenerateResumptionHandshakeSecret

Format

```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_ Tls13GenerateResumptionHandshakeSecret(
        tsip_tls13_handle_t *handle,
        e_tsip_tls13_mode_t mode,
        tsip_tls13_ephemeral_pre_shared_key_index_t *pre_shared_key_index,
        tsip_tls13_ephemeral_shared_secret_key_index_t *shared_secret_key_index,
        tsip_tls13_ephemeral_handshake_secret_key_index_t *handshake_secret_key_index
)
```

Parameters

handle	Input	Handle number (work area) indicating same session
mode	Input	Handshake protocol to use TSIP_TLS13_MODE_FULL_HANDSHAKE: Full handshake TSIP_TLS13_MODE_RESUMPTION: Resumption TSIP_TLS13_MODE_0_RTT: 0-RTT
pre_shared_key_index	Input	Pre-shared key ephemeral wrapped key Use the value of pre_shared_key_index output by R_TSIP_Tls13GeneratePreSharedKey.
shared_secret_key_index	Input	Shared secret ephemeral wrapped key Use the value of shared_secret_key_index output by R_TSIP_Tls13GenerateEcdheSharedSecret.
handshake_secret_key_index	Output	Handshake secret ephemeral wrapped key Use this value as the handshake_secret_key_index parameter input to R_TSIP_Tls13GenerateServerHandshakeTrafficKey, R_TSIP_Tls13GenerateClientHandshakeTrafficKey, and R_TSIP_Tls13GenerateMasterSecret.

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_FAIL	Occurrence of internal error
TSIP_ERR_RESOURCE_CONFLICT:	Occurrence of resource conflict because a hardware resource needed by the processing routine was in use by another processing routine
TSIP_ERR_KEY_SET	Invalid wrapped key input

Description

This API generates a handshake secret wrapped key, using the pre-shared wrapped key generated by R_TSIP_Tls13GeneratePreSharedKey, for use by the TLS 1.3 cooperation function.

Only pre-shared keys generated by the TSIP can be used. Other pre-shared keys are not supported.

Reentrancy

Not supported.

4.2.15.14 R_TSIP_Tls13Generate0RttApplicationWriteKey

Format

```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_ Tls13Generate0RttApplicationWriteKey(
        tsip_tls13_handle_t *handle,
        tsip_tls13_ephemeral_pre_shared_key_index_t *pre_shared_key_index,
        uint8_t *digest,
        tsip_aes_key_index_t *client_write_key_index
)
```

Parameters

handle	Input/output	Handle number (work area) indicating same session
pre_shared_key_index	Input	Pre-shared key ephemeral wrapped key Use the value of pre_shared_key_index output by R_TSIP_Tls13GeneratePreSharedKey.
digest	Input	Message hash calculated using SHA256 Calculate and input the hash value of ClientHello. Use R_TSIP_Sha256Init/Update/Final to calculate the hash value and input as digest the value output by R_TSIP_Sha256Final.
client_write_key_index	Output	Client write key ephemeral wrapped key Use this value as the key_index parameter input to R_TSIP_Tls13EncryptInit.

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_FAIL	Occurrence of internal error
TSIP_ERR_RESOURCE_CONFLICT:	Occurrence of resource conflict because a hardware resource needed by the processing routine was in use by another processing routine
TSIP_ERR_KEY_SET	Invalid wrapped key input

Description

This API generates a client write wrapped key for use as 0-RTT, using the pre-shared key generated by R_TSIP_Tls13GeneratePreSharedKey, for use by the TLS 1.3 cooperation function.

As stated in section 2.3 of RFC 8446, when using 0-RTT the data is not forward secret and there are no guarantees of non-replay between connections. A judgment must be made with these risks in mind as to the use of this functionality.

Reentrancy

Not supported.

4.2.15.15 R_TSIP_Tls13CertificateVerifyGenerate

Format

```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_ Tls13CertificateVerifyGenerate(
        uint32_t *key_index,
        e_tsip_tls13_signature_scheme_type_t signature_scheme,
        uint8_t *digest,
        uint8_t *certificate_verify,
        uint32_t *certificate_verify_len
)
```

Parameters

key_index	Input	Secret wrapped key for signature generation Use the value of key_pair_index or key_index output by R_TSIP_GenerateEccP256PrivateKeyIndex, R_TSIP_GenerateEccP256RandomKeyIndex, R_TSIP_UpdateEccP256PrivateKeyIndex, R_TSIP_GenerateRsa2048PrivateKeyIndex, R_TSIP_GenerateRsa2048RandomKeyIndex, or R_TSIP_UpdateRsa2048PrivateKeyIndex. Input this parameter after casting with uint32_t*.
signature_scheme	Input	Signature algorithm to be used TSIP_TLS13_SIGNATURE_SCHEME_ECDSA_SECP256R1_SHA256: ecdsa_secp256r1_sha256 TSIP_TLS13_SIGNATURE_SCHEME_RSA_PSS_RSAE_SHA256: rsa_pss_rsae_sha256
digest	Input	Message hash calculated using SHA256 Calculate and input a hash value of the value of concatenated handshake messages such as (ClientHello||ServerHello||EncryptedExtensions ||CertificateRequest||Certificate||CertificateVerify ||ServerFinished||Certificate). Use R_TSIP_Sha256Init/Update/Final to calculate the hash value and input as digest the value output by R_TSIP_Sha256Final.
certificate_verify	Output	CertificateVerify Data is output in the format specified in RFC 8446 section 4.4.3, Certificate Verify.
certificate_verify_len	Output	Byte length of certificate_verify

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_FAIL	Occurrence of internal error
TSIP_ERR_RESOURCE_CONFLICT:	Occurrence of resource conflict because a hardware resource needed by the processing routine was in use by another processing routine
TSIP_ERR_KEY_SET	Invalid wrapped key input
TSIP_ERR_PARAMETER	Invalid input data

Description

This API generates a CertificateVerify message to be sent to the server for use by the TLS 1.3 cooperation function. The algorithms used are ecdsa_secp256r1_sha256 and rsa_pss_rsae_sha256.

Reentrancy

Not supported.

4.2.15.16 R_TSIP_Tls13CertificateVerifyVerification

Format

```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_ Tls13CertificateVerifyVerification(
        uint32_t *key_index,
        e_tsip_tls13_signature_scheme_type_t signature_scheme,
        uint8_t *digest,
        uint8_t *certificate_verify,
        uint32_t certificate_verify_len
)
```

Parameters

key_index	Input	Encrypted public key Use the value of encrypted_output_public_key output by R_TSIP_TlsCertificateVerification or R_TSIP_TlsCertificateVerificationExtension.
signature_scheme	Input	Signature algorithm to be used TSIP_TLS13_SIGNATURE_SCHEME_ECDSA_SECP256R1_SHA256: ecdsa_secp256r1_sha256 TSIP_TLS13_SIGNATURE_SCHEME_RSA_PSS_RSAE_SHA256: rsa_pss_rsae_sha256
digest	Input	Message hash calculated using SHA256 Calculate and input a hash value of the value of concatenated handshake messages such as (ClientHello||ServerHello||EncryptedExtensions ||CertificateRequest||Certificate). Use R_TSIP_Sha256Init/Update/Final to calculate the hash value and input as digest the value output by R_TSIP_Sha256Final.
certificate_verify	Input	CertificateVerify Input the start address of the buffer for storing the data in the format specified in RFC 8446 section 4.4.3, Certificate Verify.
certificate_verify_len	Input	Byte length of certificate_verify

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_FAIL	Internal error, or signature verification failure
TSIP_ERR_RESOURCE_CONFLICT:	Occurrence of resource conflict because a hardware resource needed by the processing routine was in use by another processing routine
TSIP_ERR_KEY_SET	Invalid wrapped key input
TSIP_ERR_PARAMETER	Invalid input data

Description

This API verifies a CertificateVerify message received from the server for use by the TLS 1.3 cooperation function. The algorithms used are ecdsa_secp256r1_sha256 and rsa_pss_rsae_sha256.

Reentrancy

Not supported.

4.2.15.17 R_TSIP_GenerateTls13SVP256EccKeyIndex

Format

```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_ TlsGenerateTls13SVP256EccKeyIndex(
        tsip_tls13_handle_t *handle,
        e_tsip_tls13_mode_t mode,
        tsip_tls_p256_ecc_key_index_t *key_index,
        uint8_t *ephemeral_ecdh_public_key
)
```

Parameters

handle	Input	Handle number (work area) indicating same session
mode	Input	Handshake protocol to use TSIP_TLS13_MODE_FULL_HANDSHAKE: Full handshake TSIP_TLS13_MODE_RESUMPTION: Resumption TSIP_TLS13_MODE_0_RTT: 0-RTT
key_index	Output	Ephemeral ECC secret wrapped key Use this value as the key_index parameter input to R_TSIP_Tls13SVGenerateEcdheSharedSecret.
ephemeral_ecdh_public_key	Output	Ephemeral ECDH public key to be sent to the server Qx (256 bits) || Qy (256 bits)

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_FAIL:	Occurrence of internal error
TSIP_ERR_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 generates from a random number a key pair for use by the server function of the TLS 1.3 cooperation function in performing elliptic curve cryptography over a 256-bit prime field.

Reentrancy

Not supported.

4.2.15.18 R_TSIP_Tls13SVGenerateEcdheSharedSecret

Format

```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_ Tls13SVGenerateEcdheSharedSecret(
        e_tsip_tls13_mode_t mode,
        uint8_t *client_public_key,
        tsip_tls_p256_ecc_key_index_t *key_index,
        tsip_tls13_ephemeral_shared_secret_key_index_t *shared_secret_key_index
)
```

Parameters

mode	Input	Handshake protocol to use TSIP_TLS13_MODE_FULL_HANDSHAKE: Full handshake TSIP_TLS13_MODE_RESUMPTION: Resumption TSIP_TLS13_MODE_0_RTT: 0-RTT
client_public_key	Input	Public key provided by the client Qx (256 bits) || Qy (256 bits)
key_index	Input	Ephemeral ECC secret wrapped key Use the value of key_index output by R_TSIP_GenerateTls13SVP256EccKeyIndex.
shared_secret_key_index	Output	Shared secret ephemeral wrapped key Use this value as the shared_secret_key_index parameter input to R_TSIP_Tls13SVGenerateHandshakeSecret and R_TSIP_Tls13SVGenerateResumptionHandshake Secret.

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_FAIL:	Occurrence of internal error
TSIP_ERR_RESOURCE_CONFLICT:	Occurrence of resource conflict because a hardware resource needed by the processing routine was in use by another processing routine
TSIP_ERR_KEY_SET	Invalid wrapped key input

Description

This API calculates a shared secret, which is a shared key over a 256-bit prime field, using a public key provided by the server and a previously calculated secret key, and generates a wrapped key for use by the server function of the TLS 1.3 cooperation function.

Applicable cypher suites: TLS_AES_128_GCM_SHA256, TLS_AES_128_CCM_SHA256

Key exchange format: ECDHE NIST P-256

Reentrancy

Not supported.

4.2.15.19 R_TSIP_Tls13SVGenerateHandshakeSecret

Format

```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_ Tls13SVGenerateHandshakeSecret(
        tsip_tls13_ephemeral_shared_secret_key_index_t *shared_secret_key_index,
        tsip_tls13_ephemeral_handshake_secret_key_index_t *handshake_secret_key_index
)
```

Parameters

shared_secret_key_index	Input	Shared secret ephemeral wrapped key Use the value of shared_secret_key_index output by R_TSIP_Tls13SVGenerateEcdheSharedSecret.
handshake_secret_key_index	Output	Handshake secret ephemeral wrapped key Use this value as the handshake_secret_key_index parameter input to R_TSIP_Tls13SVGenerateServerHandshakeTrafficKey, R_TSIP_Tls13SVGenerateClientHandshakeTrafficKey, and R_TSIP_Tls13SVGenerateMasterSecret.

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_FAIL:	Occurrence of internal error
TSIP_ERR_RESOURCE_CONFLICT:	Occurrence of resource conflict because a hardware resource needed by the processing routine was in use by another processing routine
TSIP_ERR_KEY_SET	Invalid wrapped key input

Description

This API generates a handshake secret wrapped key, using a shared secret ephemeral key, for use by the server function of the TLS 1.3 cooperation function.

Reentrancy

Not supported.

4.2.15.20 R_TSIP_Tls13SVGenerateServerHandshakeTrafficKey

Format

```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_Tls13SVGenerateServerHandshakeTrafficKey(
        tsip_tls13_handle_t *handle,
        e_tsip_tls13_mode_t mode,
        tsip_tls13_ephemeral_handshake_secret_key_index_t *handshake_secret_key_index,
        uint8_t *digest,
        tsip_aes_key_index_t *server_write_key_index,
        tsip_hmac_sha_key_index_t *server_finished_key_index
)
```

Parameters

handle	Output	Handle number (work area) indicating same session
mode	Input	Handshake protocol to use TSIP_TLS13_MODE_FULL_HANDSHAKE: Full handshake TSIP_TLS13_MODE_RESUMPTION: Resumption TSIP_TLS13_MODE_0_RTT: 0-RTT
handshake_secret_key_index	Input	Handshake secret ephemeral wrapped key Use the value of handshake_secret_key_index output by R_TSIP_Tls13SVGenerateHandshakeSecret or R_TSIP_Tls13SVGenerateResumptionHandshakeSecret.
digest	Input	Message hash calculated using SHA256 Calculate and input the hash value of (ClientHello||ServerHello). Use R_TSIP_Sha256Init/Update/Final to calculate the hash value and input as digest the value output by R_TSIP_Sha256Final.
server_write_key_index	Output	Server write key ephemeral wrapped key Use this value as the key_index parameter input to R_TSIP_Tls13EncryptInit.
server_finished_key_index	Output	Server finished key ephemeral wrapped key Use this value as the key_index parameter input to R_TSIP_Sha256HmacGenerateInit.

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_RESOURCE_CONFLICT:	Occurrence of resource conflict because a hardware resource needed by the processing routine was in use by another processing routine
TSIP_ERR_KEY_SET	Invalid wrapped key input

Description

This API generates, using the handshake secret output by R_TSIP_Tls13SVGenerateHandshakeSecret, a server write wrapped key and server finished wrapped key for use by the server function of the TLS 1.3 cooperation function.

Reentrancy

Not supported.

4.2.15.21 R_TSIP_Tls13SVGenerateClientHandshakeTrafficKey

Format

```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_Tls13SVGenerateClientHandshakeTrafficKey(
        tsip_tls13_handle_t *handle,
        e_tsip_tls13_mode_t mode,
        tsip_tls13_ephemeral_handshake_secret_key_index_t *handshake_secret_key_index,
        uint8_t *digest,
        tsip_aes_key_index_t *client_write_key_index,
        tsip_tls13_ephemeral_client_finished_key_index_t *client_finished_key_index
)
```

Parameters

handle	Output	Handle number (work area) indicating same session
mode	Input	Handshake protocol to use TSIP_TLS13_MODE_FULL_HANDSHAKE: Full handshake TSIP_TLS13_MODE_RESUMPTION: Resumption TSIP_TLS13_MODE_0_RTT: 0-RTT
handshake_secret_key_index	Input	Handshake secret ephemeral wrapped key Use the value of handshake_secret_key_index output by R_TSIP_Tls13SVGenerateHandshakeSecret or R_TSIP_Tls13SVGenerateResumptionHandshakeSecret.
digest	Input	Message hash calculated using SHA256 Calculate and input the hash value of (ClientHello||ServerHello). Use R_TSIP_Sha256Init/Update/Final to calculate the hash value and input as digest the value output by R_TSIP_Sha256Final.
client_write_key_index	Output	Client write key ephemeral wrapped key Use this value as the key_index parameter input to R_TSIP_Tls13DecryptInit.
client_finished_key_index	Output	Client finished key ephemeral wrapped key Use this value as the client_finished_key_index parameter input to R_TSIP_Tls13SVClientHandshakeVerification.

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_RESOURCE_CONFLICT:	Occurrence of resource conflict because a hardware resource needed by the processing routine was in use by another processing routine
TSIP_ERR_KEY_SET	Invalid wrapped key input

Description

This API generates, using the handshake secret output by R_TSIP_Tls13SVGenerateHandshakeSecret, a client write wrapped key and client finished wrapped key for use by the server function of the TLS 1.3 cooperation function.

Reentrancy

Not supported.

4.2.15.22 R_TSIP_Tls13SVClientHandshakeVerification

Format

```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_ Tls13SVClientHandshakeVerification(
        e_tsip_tls13_mode_t mode,
        tsip_tls13_ephemeral_client_finished_key_index_t *client_finished_key_index,
        uint8_t *digest,
        uint8_t *client_finished
)
```

Parameters

mode	Input	Handshake protocol to use TSIP_TLS13_MODE_FULL_HANDSHAKE: Full handshake TSIP_TLS13_MODE_RESUMPTION: Resumption TSIP_TLS13_MODE_0_RTT: 0-RTT
client_finished_key_index	Input	Client finished key ephemeral wrapped key Use the value of client_finished_key_index output by R_TSIP_Tls13SVGenerateClientHandshakeTrafficKey
digest	Input	Message hash calculated using SHA256 Calculate and input a hash value of the value of concatenated handshake messages such as (ClientHello||ServerHello||EncryptedExtensions ||CertificateRequest||Certificate||CertificateVerify ||ServerFinished||Certificate||CertificateVerify). Use R_TSIP_Sha256Init/Update/Final to calculate the hash value and input as digest the value output by R_TSIP_Sha256Final.
client_finished	Input	Finished information provided by the client Input the start address of the buffer for storing the ClientFinished data obtained from R_TSIP_Tls13DecryptUpdate/Final.

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_FAIL	Occurrence of internal error
TSIP_ERR_RESOURCE_CONFLICT:	Occurrence of resource conflict because a hardware resource needed by the processing routine was in use by another processing routine
TSIP_ERR_KEY_SET	Invalid wrapped key input
TSIP_ERR_VERIFICATION_FAIL	Verification failure

Description

This API verifies handshake messages, using the Finished information provided by the client, for use by the server function of the TLS 1.3 cooperation function.

If this API returns a value of TSIP_ERR_VERIFICATION_FAIL, halt TLS communication including the verified handshake messages.

Reentrancy

Not supported.

4.2.15.23 R_TSIP_Tls13SVGenerateMasterSecret

Format

```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_ Tls13SVGenerateMasterSecret(
        tsip_tls13_handle_t *handle,
        e_tsip_tls13_mode_t mode,
        tsip_tls13_ephemeral_handshake_secret_key_index_t *handshake_secret_key_index,
        tsip_tls13_ephemeral_master_secret_key_index_t *master_secret_key_index
)
```

Parameters

handle	Output	Handle number (work area) indicating same session
mode	Input	Handshake protocol to use TSIP_TLS13_MODE_FULL_HANDSHAKE: Full handshake TSIP_TLS13_MODE_RESUMPTION: Resumption TSIP_TLS13_MODE_0_RTT: 0-RTT
handshake_secret_key_index	Input	Handshake secret ephemeral wrapped key Use the value of handshake_secret_key_index output by R_TSIP_Tls13SVGenerateHandshakeSecret.
master_secret_key_index	Output	Master secret ephemeral wrapped key Use this value as the master_secret_key_index parameter input to R_TSIP_Tls13SVGenerateApplicationTrafficKey and R_TSIP_Tls13SVGenerateResumptionMasterSecret.

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_FAIL	Occurrence of internal error
TSIP_ERR_RESOURCE_CONFLICT:	Occurrence of resource conflict because a hardware resource needed by the processing routine was in use by another processing routine
TSIP_ERR_KEY_SET	Invalid wrapped key input

Description

This API generates a master secret ephemeral wrapped key, using a handshake secret ephemeral key, for use by the server function of the TLS 1.3 cooperation function.

Reentrancy

Not supported.

4.2.15.24 R_TSIP_Tls13SVGenerateApplicationTrafficKey

Format

```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_Tls13SVGenerateApplicationTrafficKey(
        tsip_tls13_handle_t *handle,
        e_tsip_tls13_mode_t mode,
        tsip_tls13_ephemeral_master_secret_key_index_t *master_secret_key_index,
        uint8_t *digest,
        tsip_tls13_ephemeral_app_secret_key_index_t *server_app_secret_key_index,
        tsip_tls13_ephemeral_app_secret_key_index_t *client_app_secret_key_index,
        tsip_aes_key_index_t *server_write_key_index,
        tsip_aes_key_index_t *client_write_key_index
)
```

Parameters

handle	Input/output	Handle number (work area) indicating same session
mode	Input	Handshake protocol to use TSIP_TLS13_MODE_FULL_HANDSHAKE: Full handshake TSIP_TLS13_MODE_RESUMPTION: Resumption TSIP_TLS13_MODE_0_RTT: 0-RTT
master_secret_key_index	Input	Master secret ephemeral wrapped key Use the value of master_secret_key_index output by R_TSIP_Tls13SVGenerateMasterSecret.
digest	Input	Message hash calculated using SHA256 Calculate and input a hash value of the value of concatenated handshake messages such as (ClientHello||ServerHello||EncryptedExtensions ||CertificateRequest||Certificate||CertificateVerify ||ServerFinished). Use R_TSIP_Sha256Init/Update/Final to calculate the hash value and input as digest the value output by R_TSIP_Sha256Final.
server_app_secret_key_index	Output	Server application traffic secret ephemeral wrapped key Use this value as the input_app_secret_key_index parameter input to R_TSIP_Tls13SVUpdateApplicationTrafficKey.
client_app_secret_key_index	Output	Client application traffic secret ephemeral wrapped key Use this value as the input_app_secret_key_index parameter input to R_TSIP_Tls13SVUpdateApplicationTrafficKey.
server_write_key_index	Output	Server write key ephemeral wrapped key Use this value as the key_index parameter input to R_TSIP_Tls13EncryptInit
client_write_key_index	Output	Client write key ephemeral wrapped key Use this value as the key_index parameter input to R_TSIP_Tls13DecryptInit.

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_RESOURCE_CONFLICT:	Occurrence of resource conflict because a hardware resource needed by the processing routine was in use by another processing routine
TSIP_ERR_KEY_SET	Invalid wrapped key input

Description

This API generates an application traffic secret wrapped key, using a master secret ephemeral key, for use by the server function of the TLS 1.3 cooperation function. It also generates server write key and client write key ephemeral wrapped keys.

When application data is sent from the server without waiting to receive ClientFinished messages and a ClientFinished verification error occurs, the error can only be detected under conditions in which the server program has not been tampered with. A judgment must be made with these risks in mind as to the use of this functionality.

Reentrancy

Not supported.

4.2.15.25 R_TSIP_Tls13SVUpdateApplicationTrafficKey

Format

```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_Tls13SVUpdateApplicationTrafficKey(
        tsip_tls13_handle_t *handle,
        e_tsip_tls13_mode_t mode,
        e_tsip_tls13_update_key_type_t key_type,
        tsip_tls13_ephemeral_app_secret_key_index_t *input_app_secret_key_index,
        tsip_tls13_ephemeral_app_secret_key_index_t *output_app_secret_key_index,
        tsip_aes_key_index_t *app_write_key_index
)
```

Parameters

handle	Input/output	Handle number (work area) indicating same session
mode	Input	Handshake protocol to use TSIP_TLS13_MODE_FULL_HANDSHAKE: Full handshake TSIP_TLS13_MODE_RESUMPTION: Resumption TSIP_TLS13_MODE_0_RTT: 0-RTT
key_type	Input	Type of key to be updated TSIP_TLS13_UPDATE_SERVER_KEY: Server application traffic secret TSIP_TLS13_UPDATE_CLIENT_KEY: Client application traffic secret
input_app_secret_key_index	Input	Ephemeral wrapped key of input server/client application traffic secret Use as input either server/client_app_secret_key_index output by R_TSIP_Tls13SVGenerateApplicationTrafficKey or output_app_secret_key_index output by R_TSIP_Tls13SVUpdateApplicationTrafficKey, whichever matches the type of key specified by key_type.
output_app_secret_key_index	Output	Ephemeral wrapped key of output server/client application traffic secret Output matching the type of key specified by key_type is obtained. Use this value as the input_app_secret_key_index parameter input to R_TSIP_Tls13SVUpdateApplicationTrafficKey.
app_write_key_index	Output	Server/client write key ephemeral wrapped key Output matching the type of key specified by key_type is obtained. Use ServerWriteKey as the key_index parameter input to R_TSIP_Tls13EncryptInit. Use ClientWriteKey as the key_index parameter input to R_TSIP_Tls13DecryptInit.

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_FAIL	Occurrence of internal error
TSIP_ERR_RESOURCE_CONFLICT:	Occurrence of resource conflict because a hardware resource needed by the processing routine was in use by another processing routine
TSIP_ERR_KEY_SET	Invalid wrapped key input
TSIP_ERR_PARAMETER	Invalid input data

Description

This API updates, using an application traffic secret, an encryption wrapped key corresponding to an application traffic secret wrapped key for use by the server function of the TLS 1.3 cooperation function.

Reentrancy

Not supported.

4.2.15.26 R_TSIP_Tls13SVGenerateResumptionMasterSecret

Format

```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_ Tls13SVGenerateResumptionMasterSecret(
        tsip_tls13_handle_t *handle,
        e_tsip_tls13_mode_t mode,
        tsip_tls13_ephemeral_master_secret_key_index_t *master_secret_key_index,
        uint8_t *digest,
        tsip_tls13_ephemeral_res_master_secret_key_index_t *res_master_secret_key_index
)
```

Parameters

handle	Input	Handle number (work area) indicating same session
mode	Input	Handshake protocol to use TSIP_TLS13_MODE_FULL_HANDSHAKE: Full handshake TSIP_TLS13_MODE_RESUMPTION: Resumption TSIP_TLS13_MODE_0_RTT: 0-RTT
master_secret_key_index	Input	Handshake secret ephemeral wrapped key Use the value of handshake_secret_key_index output by R_TSIP_Tls13SVGenerateHandshakeSecret.
digest	Input	Message hash calculated using SHA256 Calculate and input a hash value of the value of concatenated handshake messages such as (ClientHello||ServerHello||EncryptedExtensions ||CertificateRequest||Certificate||CertificateVerify ||ServerFinished||Certificate||CertificateVerify ||ClientFinished). Use R_TSIP_Sha256Init/Update/Final to calculate the hash value and input as digest the value output by R_TSIP_Sha256Final.
res_master_secret_key_index	Output	Resumption master secret ephemeral wrapped key Use this value as the res_master_secret_key_index parameter input to R_TSIP_Tls13SVGeneratePreSharedKey.

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_RESOURCE_CONFLICT:	Occurrence of resource conflict because a hardware resource needed by the processing routine was in use by another processing routine
TSIP_ERR_KEY_SET	Invalid wrapped key input

Description

This API generates a resumption master secret wrapped key, using a master secret ephemeral key, for use by the server function of the TLS 1.3 cooperation function.

As specified in RFC 8446, delete master_secret_key_index, the master secret ephemeral wrapped key, after generating a resumption master secret wrapped key using this API.

Reentrancy

Not supported.

4.2.15.27 R_TSIP_Tls13SVGeneratePreSharedKey

Format

```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_ Tls13SVGeneratePreSharedKey(
        tsip_tls13_handle_t *handle,
        e_tsip_tls13_mode_t mode,
        tsip_tls13_ephemeral_res_master_secret_key_index_t *res_master_secret_key_index,
        uint8_t *ticket_nonce,
        uint32_t ticket_nonce_len,
        tsip_tls13_ephemeral_pre_shared_key_index_t *pre_shared_key_index
)
```

Parameters

handle	Input	Handle number (work area) indicating same session
mode	Input	Handshake protocol to use TSIP_TLS13_MODE_FULL_HANDSHAKE: Full handshake TSIP_TLS13_MODE_RESUMPTION: Resumption TSIP_TLS13_MODE_0_RTT: 0-RTT
res_master_secret_key_index	Input	Resumption master secret ephemeral wrapped key Use the value of res_master_secret_key_index output by R_TSIP_Tls13SVGenerateResumptionMasterSecret.
ticket_nonce	Input	Ticket nonce provided by the server If the size of the ticket nonce is not a multiple of 16 bytes, pad it with zeros to make it a multiple of 16 bytes before input.
ticket_nonce_len	Input	Byte length of ticket nonce Maximum size is 255 byte.
pre_shared_key_index	Output	Pre-shared key ephemeral wrapped key Use this value as the pre_shared_key_index parameter input to R_TSIP_Tls13SVGeneratePskBinderKey, R_TSIP_Tls13SVGenerateResumptionHandshakeSecret, and R_TSIP_Tls13SVGenerate0RttApplicationWriteKey.

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_FAIL	Occurrence of internal error
TSIP_ERR_RESOURCE_CONFLICT:	Occurrence of resource conflict because a hardware resource needed by the processing routine was in use by another processing routine
TSIP_ERR_KEY_SET	Invalid wrapped key input

Description

This API generates, using a resumption master secret ephemeral key, a pre-shared wrapped key from new session ticket information for use by the server function of the TLS 1.3 cooperation function.

Reentrancy

Not supported.

4.2.15.28 R_TSIP_Tls13SVGeneratePskBinderKey

Format

```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_ Tls13SVGeneratePskBinderKey(
        tsip_tls13_handle_t *handle,
        tsip_tls13_ephemeral_pre_shared_key_index_t *pre_shared_key_index,
        tsip_hmac_sha_key_index_t *psk_binder_key_index
)
```

Parameters

handle	Input	Handle number (work area) indicating same session
pre_shared_key_index	Input	Pre-shared key ephemeral wrapped key Use the value of pre_shared_key_index output by R_TSIP_Tls13SVGeneratePreSharedKey.
psk_binder_key_index	Output	Psk binder key ephemeral wrapped key Use this value to generate PskBinder. Use this value as the key_index parameter input to R_TSIP_Sha256HmacVerifyInit.

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_FAIL	Occurrence of internal error
TSIP_ERR_RESOURCE_CONFLICT:	Occurrence of resource conflict because a hardware resource needed by the processing routine was in use by another processing routine
TSIP_ERR_KEY_SET	Invalid wrapped key input

Description

This API generates a binder wrapped key for use by the server function of the TLS 1.3 cooperation function.

Reentrancy

Not supported.

4.2.15.29 R_TSIP_Tls13SVGenerateResumptionHandshakeSecret

Format

```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_ Tls13SVGenerateResumptionHandshakeSecret(
        tsip_tls13_handle_t *handle,
        e_tsip_tls13_mode_t mode,
        tsip_tls13_ephemeral_pre_shared_key_index_t *pre_shared_key_index,
        tsip_tls13_ephemeral_shared_secret_key_index_t *shared_secret_key_index,
        tsip_tls13_ephemeral_handshake_secret_key_index_t *handshake_secret_key_index
)
```

Parameters

handle	Input	Handle number (work area) indicating same session
mode	Input	Handshake protocol to use TSIP_TLS13_MODE_FULL_HANDSHAKE: Full handshake TSIP_TLS13_MODE_RESUMPTION: Resumption TSIP_TLS13_MODE_0_RTT: 0-RTT
pre_shared_key_index	Input	Pre-shared key ephemeral wrapped key Use the value of pre_shared_key_index output by R_TSIP_Tls13SVGeneratePreSharedKey.
shared_secret_key_index	Input	Shared secret ephemeral wrapped key Use the value of shared_secret_key_index output by R_TSIP_Tls13SVGenerateEcdheSharedSecret.
handshake_secret_key_index	Output	Handshake secret ephemeral wrapped key Use this value as the handshake_secret_key_index parameter input to R_TSIP_Tls13SVGenerateServerHandshakeTrafficKey, R_TSIP_Tls13SVGenerateClientHandshakeTrafficKey, and R_TSIP_Tls13SVGenerateMasterSecret.

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_FAIL	Occurrence of internal error
TSIP_ERR_RESOURCE_CONFLICT:	Occurrence of resource conflict because a hardware resource needed by the processing routine was in use by another processing routine
TSIP_ERR_KEY_SET	Invalid wrapped key input

Description

This API generates a handshake secret wrapped key, using the pre-shared wrapped key generated by R_TSIP_Tls13GeneratePreSharedKey, for use by the server function of the TLS 1.3 cooperation function.

Only pre-shared keys generated by the TSIP can be used. Other pre-shared keys are not supported.

Reentrancy

Not supported.

4.2.15.30 R_TSIP_Tls13SVGenerate0RttApplicationWriteKey

Format

```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_ Tls13SVGenerate0RttApplicationWriteKey(
        tsip_tls13_handle_t *handle,
        tsip_tls13_ephemeral_pre_shared_key_index_t *pre_shared_key_index,
        uint8_t *digest,
        tsip_aes_key_index_t *client_write_key_index
)
```

Parameters

handle	Input/output	Handle number (work area) indicating same session
pre_shared_key_index	Input	Pre-shared key ephemeral wrapped key Use the value of pre_shared_key_index output by R_TSIP_Tls13SVGeneratePreSharedKey.
digest	Input	Message hash calculated using SHA256 Calculate and input the hash value of ClientHello. Use R_TSIP_Sha256Init/Update/Final to calculate the hash value and input as digest the value output by R_TSIP_Sha256Final.
client_write_key_index	Output	Client write key ephemeral wrapped key Use this value as the key_index parameter input to R_TSIP_Tls13DecryptInit.

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_FAIL	Occurrence of internal error
TSIP_ERR_RESOURCE_CONFLICT:	Occurrence of resource conflict because a hardware resource needed by the processing routine was in use by another processing routine
TSIP_ERR_KEY_SET	Invalid wrapped key input

Description

This API generates a client write wrapped key for use as 0-RTT, using the pre-shared key generated by R_TSIP_Tls13GeneratePreSharedKey, for use by the server function of the TLS 1.3 cooperation function.

As stated in section 2.3 of RFC 8446, when using 0-RTT the data is not forward secret and there are no guarantees of non-replay between connections. A judgment must be made with these risks in mind as to the use of this functionality.

Reentrancy

Not supported.

4.2.15.31 R_TSIP_Tls13SVCertificateVerifyGenerate

Format

```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_ Tls13SVCertificateVerifyGenerate(
        uint32_t *key_index,
        e_tsip_tls13_signature_scheme_type_t signature_scheme,
        uint8_t *digest,
        uint8_t *certificate_verify,
        uint32_t *certificate_verify_len
)
```

Parameters

key_index	Input	Secret wrapped key for signature generation Use the value of key_pair_index or key_index output by R_TSIP_GenerateEccP256PrivateKeyIndex, R_TSIP_GenerateEccP256RandomKeyIndex, R_TSIP_UpdateEccP256PrivateKeyIndex, R_TSIP_GenerateRsa2048PrivateKeyIndex, R_TSIP_GenerateRsa2048RandomKeyIndex, or R_TSIP_UpdateRsa2048PrivateKeyIndex. Input this parameter after casting with uint32_t*.
signature_scheme	Input	Signature algorithm to be used TSIP_TLS13_SIGNATURE_SCHEME_ECDSA_SECP256R1_SHA256: ecdsa_secp256r1_sha256 TSIP_TLS13_SIGNATURE_SCHEME_RSA_PSS_RSAE_SHA256: rsa_pss_rsae_sha256
digest	Input	Message hash calculated using SHA256 Calculate and input a hash value of the value of concatenated handshake messages such as (ClientHello||ServerHello||EncryptedExtensions ||CertificateRequest||Certificate). Use R_TSIP_Sha256Init/Update/Final to calculate the hash value and input as digest the value output by R_TSIP_Sha256Final.
certificate_verify	Output	CertificateVerify Data is output in the format specified in RFC 8446 section 4.4.3, Certificate Verify.
certificate_verify_len	Output	Byte length of certificate_verify

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_FAIL	Occurrence of internal error
TSIP_ERR_RESOURCE_CONFLICT:	Occurrence of resource conflict because a hardware resource needed by the processing routine was in use by another processing routine
TSIP_ERR_KEY_SET	Invalid wrapped key input
TSIP_ERR_PARAMETER	Invalid input data

Description

This API generates a CertificateVerify message to be sent to the client for use by the server function of the TLS 1.3 cooperation function. The algorithms used are ecdsa_secp256r1_sha256 and rsa_pss_rsae_sha256.

Reentrancy

Not supported.

4.2.15.32 R_TSIP_Tls13SVCertificateVerifyVerification

Format

```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_ Tls13SVCertificateVerifyVerification(
        uint32_t *key_index,
        e_tsip_tls13_signature_scheme_type_t signature_scheme,
        uint8_t *digest,
        uint8_t *certificate_verify,
        uint32_t certificate_verify_len
)
```

Parameters

key_index	Input	Encrypted public key Use encrypted_output_public_key output by R_TSIP_TlsCertificateVerification or R_TSIP_TlsCertificateVerificationExtension.
signature_scheme	Input	Signature algorithm to be used TSIP_TLS13_SIGNATURE_SCHEME_ECDSA_SECP256R1_SHA256: ecdsa_secp256r1_sha256 TSIP_TLS13_SIGNATURE_SCHEME_RSA_PSS_RSAE_SHA256: rsa_pss_rsae_sha256
digest	Input	Message hash calculated using SHA256 Calculate and input a hash value of the value of concatenated handshake messages such as (ClientHello||ServerHello||EncryptedExtensions ||CertificateRequest||Certificate||CertificateVerify ||ServerFinished||Certificate). Use R_TSIP_Sha256Init/Update/Final to calculate the hash value and input as digest the value output by R_TSIP_Sha256Final.
certificate_verify	Input	CertificateVerify Input the start address of the buffer for storing the data in the format specified in RFC 8446 section 4.4.3, Certificate Verify.
certificate_verify_len	Input	Byte length of certificate_verify

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_FAIL	Internal error, or signature verification failure
TSIP_ERR_RESOURCE_CONFLICT:	Occurrence of resource conflict because a hardware resource needed by the processing routine was in use by another processing routine
TSIP_ERR_KEY_SET	Invalid wrapped key input
TSIP_ERR_PARAMETER	Invalid input data

Description

This API verifies a CertificateVerify message received from the client for use by the server function of the TLS 1.3 cooperation function. The algorithms used are ecdsa_secp256r1_sha256 and rsa_pss_rsae_sha256.

Reentrancy

Not supported.

4.2.15.33 R_TSIP_Tls13EncryptInit

Format

```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_ Tls13EncryptInit(
        tsip_tls13_handle_t *handle,
        e_tsip_tls13_phase_t phase,
        e_tsip_tls13_mode_t mode,
        e_tsip_tls13_cipher_suite_t cipher_suite,
        tsip_aes_key_index_t *key_index,
        uint32_t payload_length
)
```

Parameters

handle	Output	TLS 1.3 handler (work area)
phase	Input	Communication phase TSIP_TLS13_PHASE_HANDSHAKE: Handshake phase TSIP_TLS13_PHASE_APPLICATION: Application phase
mode	Input	Handshake protocol to use TSIP_TLS13_MODE_FULL_HANDSHAKE: Full handshake TSIP_TLS13_MODE_RESUMPTION: Resumption TSIP_TLS13_MODE_0_RTT: 0-RTT
cipher_suite	Input	Cypher suite TSIP_TLS13_CIPHER_SUITE_AES_128_GCM_SHA256: TLS_AES_128_GCM_SHA256 TSIP_TLS13_CIPHER_SUITE_AES_128_CCM_SHA256: TLS_AES_128_CCM_SHA256
key_index	Input	Ephemeral wrapped key of key used for encryption
payload_length	Input	Byte length of data to be encrypted

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_FAIL	Occurrence of internal error
TSIP_ERR_RESOURCE_CONFLICT:	Occurrence of resource conflict because a hardware resource needed by the processing routine was in use by another processing routine
TSIP_ERR_KEY_SET	Invalid wrapped key input

Description

The R_TSIP_Tls13EncryptInit() function performs preparations for the encryption of TLS 1.3 communication data and writes the result to the first parameter, handle. The parameter handle is used subsequently as a parameter by the R_TSIP_Tls13EncryptUpdate() and R_TSIP_Tls13EncryptFinal() functions.

Reentrancy

Not supported.

4.2.15.34 R_TSIP_Tls13EncryptUpdate

Format

```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_ Tls13EncryptUpdate(
        tsip_tls13_handle_t *handle,
        uint8_t *plain,
        uint8_t *cipher,
        uint32_t plain_length
)
```

Parameters

handle	Input/output	TLS 1.3 handler (work area)
plain	Input	Plaintext data area
cipher	Output	Ciphertext data area
plain_length	Input	Plaintext data length

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_PARAMETER	Invalid input data
TSIP_ERR_PROHIBIT_FUNCTION	Invalid function called

Description

The R_TSIP_Tls13EncryptUpdate() function encrypts the plaintext specified by the second parameter, plain, using the value specified for key_index in R_TSIP_Tls13EncryptInit() function. The function internally buffers the data input by the user until the input value of plain exceeds 16 bytes. Once the input data from plain reaches 16 bytes or more, the encrypted result is output to the area specified by the third parameter, cipher. Specify as the payload_length parameter of R_TSIP_Tls13EncryptInit() function the total data length of the data to be input as plain. For the plain_length parameter of this function, specify the data length to be input when the user calls the function. If the input value of plain is not divisible by 16 bytes, the function performs padding internally.

Except in cases where the addresses are the same, specify areas for plain and cipher that do not overlap.

Reentrancy

Not supported.

4.2.15.35 R_TSIP_Tls13EncryptFinal

Format

```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_ Tls13EncryptFinal(
        tsip_tls13_handle_t *handle,
        uint8_t *cipher,
        uint32_t *cipher_length
)
```

Parameters

handle	Input/output	TLS 1.3 handler (work area)
cipher	Output	Ciphertext data area
cipher_length	Output	Ciphertext data length

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_FAIL	Occurrence of internal error
TSIP_ERR_PARAMETER	Invalid input data
TSIP_ERR_PROHIBIT_FUNCTION	Invalid function called

Description

If there is 16-byte fractional remainder data indicated by the data length of the value of plain input to R_TSIP_Tls13EncryptUpdate() function, the R_TSIP_Tls13EncryptFinal() function outputs the result of encrypting the fractional remainder data to the area specified by the second parameter, cipher. At this time, if the data is less than 16 bytes, it is padded with zeros by the function internally.

Reentrancy

Not supported.

4.2.15.36 R_TSIP_Tls13DecryptInit

Format

```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_ Tls13DecryptInit(
        tsip_tls13_handle_t *handle,
        e_tsip_tls13_phase_t phase,
        e_tsip_tls13_mode_t mode,
        e_tsip_tls13_cipher_suite_t cipher_suite,
        tsip_aes_key_index_t *key_index,
        uint32_t payload_length
)
```

Parameters

handle	Output	TLS 1.3 handler (work area)
phase	Input	Communication phase TSIP_TLS13_PHASE_HANDSHAKE: Handshake phase TSIP_TLS13_PHASE_APPLICATION: Application phase
mode	Input	Handshake protocol to use TSIP_TLS13_MODE_FULL_HANDSHAKE: Full handshake TSIP_TLS13_MODE_RESUMPTION: Resumption TSIP_TLS13_MODE_0_RTT: 0-RTT
cipher_suite	Input	Cypher suite TSIP_TLS13_CIPHER_SUITE_AES_128_GCM_SHA256: TLS_AES_128_GCM_SHA256 TSIP_TLS13_CIPHER_SUITE_AES_128_CCM_SHA256: TLS_AES_128_CCM_SHA256
key_index	Input	Ephemeral wrapped key of key used for decryption
payload_length	Input	Byte length of data to be decrypted

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_FAIL	Occurrence of internal error
TSIP_ERR_RESOURCE_CONFLICT:	Occurrence of resource conflict because a hardware resource needed by the processing routine was in use by another processing routine
TSIP_ERR_KEY_SET	Invalid wrapped key input

Description

The R_TSIP_Tls13DecryptInit() function performs preparations for the decryption of TLS 1.3 communication data and writes the result to the first parameter, handle. The parameter handle is used subsequently as a parameter by the R_TSIP_Tls13DecryptUpdate() and R_TSIP_Tls13DecryptFinal() functions.

Reentrancy

Not supported.

4.2.15.37 R_TSIP_Tls13DecryptUpdate

Format

```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_ Tls13DecryptUpdate(
        tsip_tls13_handle_t *handle,
        uint8_t *cipher,
        uint8_t *plain,
        uint32_t cipher_length
)
```

Parameters

handle	Input/output	TLS 1.3 handler (work area)
cipher	Input	Ciphertext data area
plain	Output	Plaintext data area
cipher_length	Input	Ciphertext data length

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_PARAMETER	Invalid input data
TSIP_ERR_PROHIBIT_FUNCTION	Invalid function called

Description

The R_TSIP_Tls13DecryptUpdate() function decrypts the ciphertext specified by the second parameter, cipher, using the value specified for key_index in R_TSIP_Tls13DecryptInit() function. The function internally buffers the data input by the user until the input value of cipher exceeds 16 bytes. Once the input data from cypher reaches 16 bytes or more, the decrypted result is output to the area specified by the third parameter, plain. Specify as the payload_length parameter of R_TSIP_Tls13DecryptInit() function the total data length of the data to be input as cipher. For the cipher_length parameter of this function, specify the data length to be input when the user calls the function. If the input value of cipher is not divisible by 16 bytes, the function performs padding internally.

Except in cases where the addresses are the same, specify areas for plain and cipher that do not overlap.

Reentrancy

Not supported.

4.2.15.38 R_TSIP_Tls13DecryptFinal

Format

```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_ Tls13DecryptFinal(
        tsip_tls13_handle_t *handle,
        uint8_t *plain,
        uint32_t *plain_length
)
```

Parameters

handle	Input/output	TLS 1.3 handler (work area)
plain	Output	Plaintext data area
plain_length	Output	Plaintext data length

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_FAIL	Occurrence of internal error
TSIP_ERR_PARAMETER	Invalid input data
TSIP_ERR_PROHIBIT_FUNCTION	Invalid function called

Description

If the data length of cipher input in R_TSIP_Tls13DecryptUpdate() function results in a fractional remainder after 16 bytes, the R_TSIP_Tls13DecryptFinal() function outputs the leftover decrypted data to the second parameter, plain. At this time, if the data is less than 16 bytes, it is padded with zeros by the function internally. For plain, specify a RAM address that is a multiple of 4.

Reentrancy

Not supported.

4.2.16 Firmware Update

4.2.16.1 R_TSIP_StartUpdateFirmware

Format

e_tsip_err_t R_TSIP_StartUpdateFirmware(void)

Parameters

None

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_RESOURCE_CONFLICT:	Occurrence of resource conflict because a hardware resource needed by the processing routine was in use by another processing routine

Description

Transitions to the firmware update state.

Reentrancy

Not supported.

4.2.16.2 R_TSIP_GenerateFirmwareMACInit

Format

```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_GenerateFirmwareMACInit(
        uint32_t *wrapped_key_encryption_key,
        uint8_t *encrypted_image_encryption_key,
        uint8_t *initial_vector
)
```

Parameters

wrapped_key_encryption_key	Input	Wrapped key encryption key
encrypted_image_encryption_Key	Input	Encrypted image encryption key
initial_vector	Input	Initialization vector area for decryption of encrypted firmware.

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_RESOURCE_CONFLICT:	Occurrence of resource conflict because a hardware resource needed by the processing routine was in use by another processing routine
TSIP_ERR_KEY_SET	Invalid wrapped key input

Description

The R_TSIP_GenerateFirmwareMACInit() function performs preparations to decrypt encrypted firmware and to generate MAC of the decrypted plain firmware.

Specify the wrapped key which is used to wrap the image encryption key to the first parameter, wrapped_key_encryption_key. Specify the wrapped image encryption key which is wrapped with key encryption key to the second parameter, encrypted_image_encryption_key. And specify the initial vector which is used to encrypt firmware to the third parameter, initial_vector.

Reentrancy

Not supported.

4.2.16.3 R_TSIP_GenerateFirmwareMACUpdate

Format

```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_GenerateFirmwareMACUpdate(
        uint8_t *input,
        uint8_t *output,
        uint32_t input_length
)
```

Parameters

input	Input	Encrypted firmware
output	Output	Decrypted firmware data
input_length	Input	Encrypted firmware byte size Please specify with a multiple of 16 bytes.

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_PARAMETER	Invalid input data

Description

The R_TSIP_GenerateFirmwareMACUpdate() function decrypts the first parameter, input, with the size of the third parameter, input_length, and output firmware to the second parameter, output.

When the decryption process has to be executed by multiple times such as the encrypted firmware is located in multiple area, R_TSIP_GenerateFirmwareMACUpdate() is used. Note that the encrypted firmware which includes the last 16 bytes must be decrypted by R_TSIP_GenerateFirmwareMACFinal(). When the decryption process do not have to be multiple times, please use R_TSIP_GenerateFirmwareMACFinal().

Reentrancy

Not supported.

4.2.16.4 R_TSIP_GenerateFirmwareMACFinal

Format

```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_GenerateFirmwareMACFinal(
        uint8_t *input,
        uint8_t *input_mac,
        uint8_t *output,
        uint8_t *output_mac,
        uint32_t input_length
)
```

Parameters

input	Input	Input firmware
input_mac	Input	MAC value of the input firmware (16 bytes)
output	Output	Output firmware data
output_mac	Output	MAC value of the output firmware (16 bytes)
input_length	Input	Input firmware byte size Please specify with a multiple of 16 bytes.

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_FAIL:	Occurrence of internal error
TSIP_ERR_PARAMETER	Invalid input data

Description

The R_TSIP_GenerateFirmwareMACFinal() function decrypts the first parameter, input, with the size of the fifth parameter, input_length, and output firmware to the second parameter, output. And it verifies the MAC value specified as the second parameter, input_mac. When the verification of the MAC, it generates the MAC of the plain firmware and output to the fourth parameter, output_mac.

Reentrancy

Not supported.

4.2.16.5 R_TSIP_VerifyFirmwareMACInit

Format

#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_VerifyFirmwareMACInit(void)

Parameters

None

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_RESOURCE_CONFLICT:	Occurrence of resource conflict because a hardware resource needed by the processing routine was in use by another processing routine

Description

The R_TSIP_VerifyFirmwareMACInit() function performs preparations to verify MAC.

Reentrancy

Not supported.

4.2.16.6 R_TSIP_VerifyFirmwareMACUpdate

Format

```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_VerifyFirmwareMACUpdate(
        uint8_t *input,
        uint32_t input_length
)
```

Parameters

input	Input	Input firmware area
input_length	Input	Input firmware byte size Please specify with a multiple of 16 bytes.

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_PARAMETER	Invalid input data

Description

The R_TSIP_VerifyFirmwareMACUpdate() function verifys the first parameter, input, with the size of thesecond parameter, input_length, and process MAC generation.

When the MAC generation process has to be executed by multiple times such as the plain firmware is located in multiple area, R_TSIP_VerifyFirmwareMACUpdate() is used. Note that the plain firmware which includes the last 16 bytes must be verified by R_TSIP_VerifyFirmwareMACFinal(). When the verification process do not have to be multiple times, please use R_TSIP_VerifyFirmwareMACFinal().

Reentrancy

Not supported.

4.2.16.7 R_TSIP_VerifyFirmwareMACFinal

Format

```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_VerifyFirmwareMACFina(
        uint8_t *input,
        uint8_t mac,
        uint32_t input_length
)
```

Parameters

input	Input	Input firmware area
mac	Input	MAC value of the input firmware (16 bytes)
input_length	Input	Input firmware byte size Please specify with a multiple of 16 bytes.

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_FAIL:	Occurrence of internal error
TSIP_ERR_PARAMETER	Invalid input data

Description

The R_TSIP_VerifyFirmwareMACFinal() function verifys the MAC of a plain firmware.

Reentrancy

Not supported.

4.3 User-Defined Functions

This section describes the user-defined functions called by the TSIP driver.

4.3.1 user_sha384_fucntion

Format

```
#include “r_tsip_rx_config.h”
uint32_t user_sha384_function (
        uint8_t *message,
        uint8_t *digest,
        uint32_t message_length)
```

Parameters

message	Input	Start address of message
digest	Output	Hash calculation result storage address (48 bytes)
message_length	Input	Number of valid bytes in message

Return Values

0	Hash value storage success
Other than 0	Hash value storage failure

Description

The TSIP does not support SHA-384 in hardware, so the following API requires the user to create an SHA-384 function for signature generation and verification. To use the API, enable TSIP_USER_SHA_384_ENABLED in r_tsip_rx_config.h and prepare a function called user_sha384_function.

- R_TSIP_EcdsaP384SignatureGenerate
- R_TSIP_EcdsaP384SignatureVerification

This function can be defined when TSIP_USER_SHA_384_ENABLED is enabled in the configuration file. It performs an SHA-384 hash calculation for an area that starts from the address specified by the parameter message and extends for the number of bytes specified by the parameter message_length.

Store the calculation result at the address specified by the parameter digest.

4.3.2 user_lock_fucntion

Format

```
#include “r_tsip_rx_config.h”
void user_lock_function (
        void)
```

Parameters

None

Return Values

None

Description

To use the TSIP access conflict avoidance functionality described in section 3.2, the user must create a function to acquire exclusive control of a resource. Implement the function user_lock_function by enabling TSIP_MULTI_THREADING in r_tsip_rx_config.h.

To use access conflict avoidance functionality together with secure boot functionality, place this function in the secure boot area.

4.3.3 user_unlock_fucntion

Format

```
#include “r_tsip_rx_config.h”
void user_unlock_function (
        void)
```

Parameters

None

Return Values

None

Description

To use the TSIP access conflict avoidance functionality described in section 3.2, the user must create a function to acquire exclusive control of a resource. Implement the function user_unlock_function by enabling TSIP_MULTI_THREADING in r_tsip_rx_config.h.

To use access conflict avoidance functionality together with secure boot functionality, place this function in the secure boot area.

5. Key Injection and Updating

This section describes how to write encryption keys handled by the TSIP 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 TSIP 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 TSIP 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 Generating an Encrypted Key Using Security Key Management Tool

You can use Security Key Management Tool to generate encrypted 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.

5.3.1 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.

5.3.1.1 Procedure for CLI Version

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

   > skmt.exe /genufpk
   /ufpk "22222222222222222222222222222221111111111111111111111111111111"
   /output "C:\work\ufpk.key"

```
C:¥Renesas¥SecurityKeyMangementTool¥cli>skmt.exe /genufpk /ufpk "22222222222222222222222222222221
11111111111111111111111111111111" /output "C:¥work¥ufpk.key"
UFPK: 22222222222222222222222222222221111111111111111111111111111111
Output File: C:¥work¥ufpk.key
```

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-TSIP" /keytype "AES-128" /key
   "11111111222222223333333344444444" /filetype "csource" /keyname "euk_aes128"
   /output "C:\work\euk_aes128.c"

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

```
C:¥Renesas¥SecurityKeyMangementTool¥cli>skmt.exe /genkey /iv "55aa55aa55aa55aa55aa55aa55aa55aa" /
ufpk file="C:¥work¥ufpk.key" /wufpk file="C:¥work¥ufpk.key_enc.key" /mcu "RX-TSIP" /keytype "AES-
128" /key "11111111222222223333333344444444" /filetype "csource" /keyname "euk_aes128" /output "C
:¥work¥euk_aes128.c"
Output File: C:¥work¥euk_aes128.h
Output File: C:¥work¥euk_aes128.c
UFPK: 22222222222222222222222222222221111111111111111111111111111111
W-UFPK: 00000001DF741B6058AFDFE2DD186EA0AF168453F2B1F311ABAF6133AC68926B0A475130
IV: 55AA55AA55AA55AA55AA55AA55AA55AA
Encrypted key: 9206ACBC67367501A27EB48604AC77C6CB6CF84B4EBFF149B79DD6C0018FE0F9
```

Figure 5.2 genkey Command Execution Example

5.3.1.2 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.

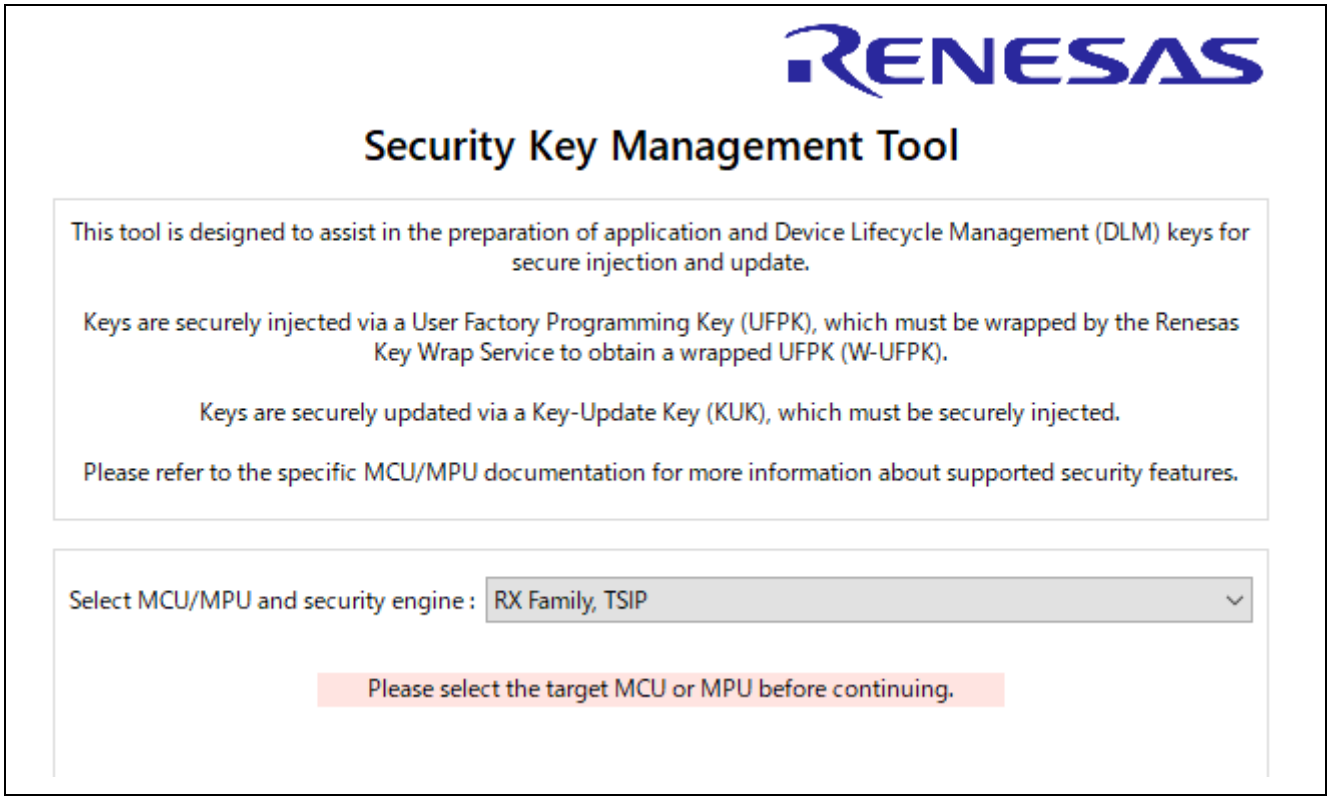


Figure 5.3 Overview Tab

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**.

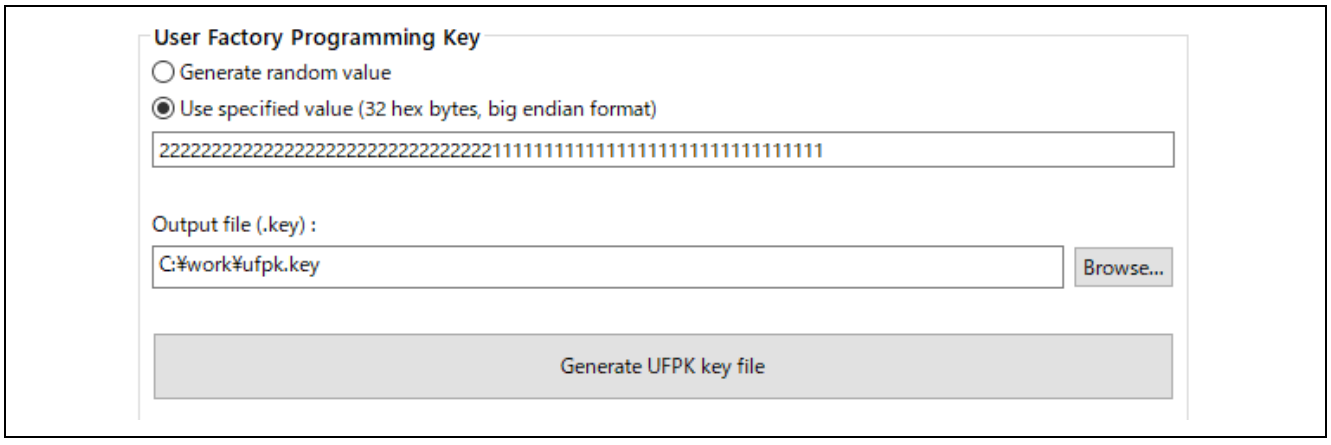


Figure 5.4 Example of UFPK Generation by Specifying Values on Generate UFPK Tab

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

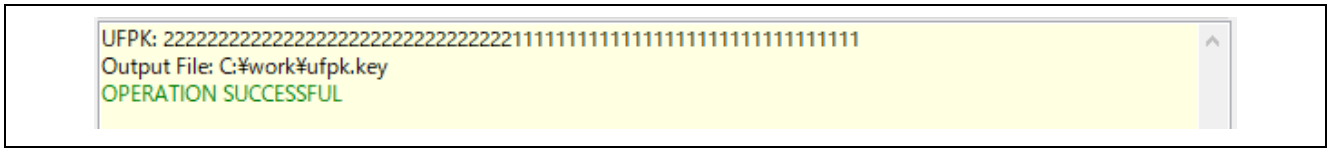


Figure 5.5 Generate UFPK Tab Execution Result

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**.

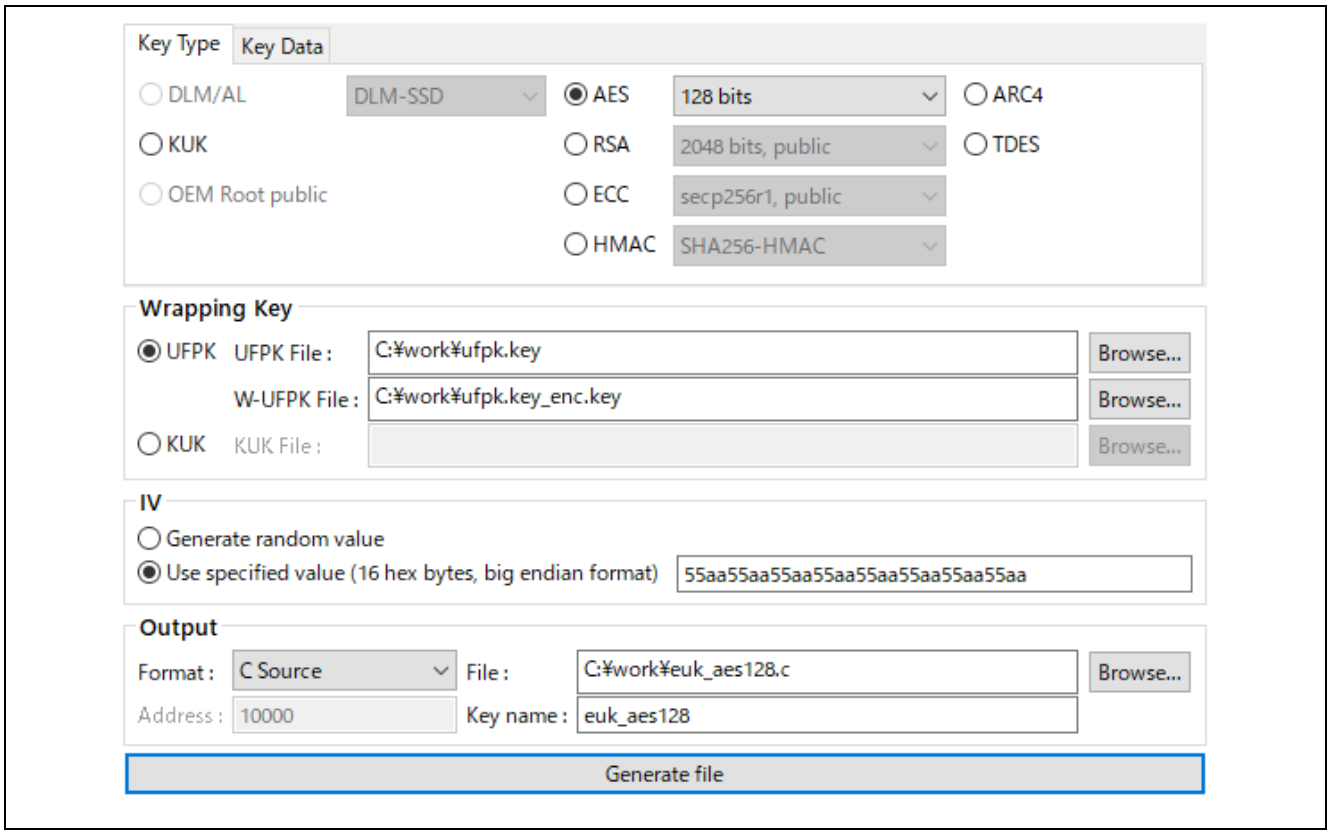


Figure 5.6 Example AES 128 Key File Output Settings on Wrap Key – Key Type Tab

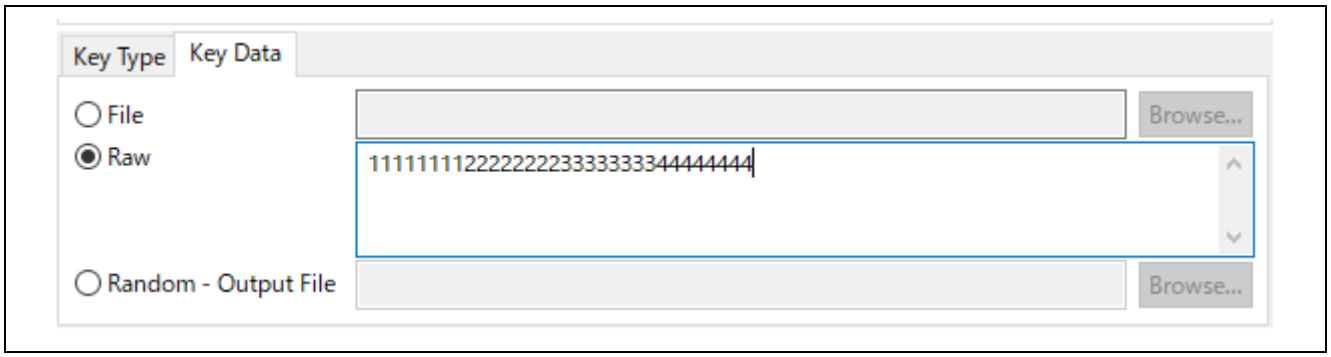


Figure 5.7 Example AES 128 Key C Source Output Settings on Wrap Key – Key Data Tab

The following is displayed when the operation completes successfully.

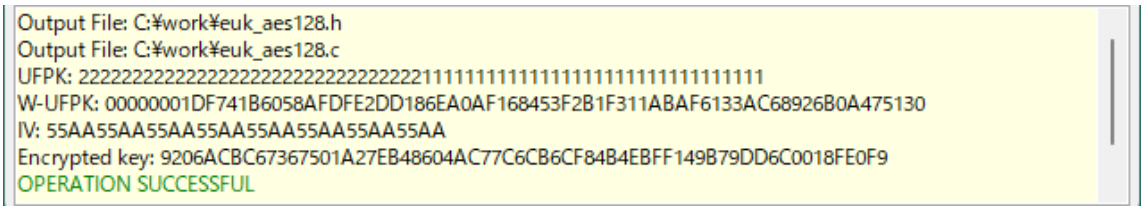


Figure 5.8 Wrap Key Tab Execution Result

Incorpolate the data in the output C source file into your project with the same method in 5.3.1.1.

5.3.2 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.

5.3.2.1 Procedure for CLI Version

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

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

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

Figure 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 /genkuk /output "C:\work\kuk.key" /kuk "d0aec19726cbc0e2fb40
3866b9b465a6c0d05b7a60362d5f435f9a3e98c79084"
KUK: D0AEC19726CBC0E2FB403866B9B465A6C0D05B7A60362D5F435F9A3E98C79084
Output File: C:\work\kuk.key
```

Figure 5.10 genkuk Command Execution Result

4. Generating an encrypted 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-TSIP" /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.

```
C:¥Renesas¥SecurityKeyMangementTool¥cli>skmt.exe /genkey /ufpk file="c:¥work¥ufpk.key" /wu
fpk file="C:¥work¥ufpk.key_enc.key" /mcu "RX-TSIP" /keytype "key-update-key" /key file="C:
¥work¥kuk.key" /filetype "csource" /output "C:¥work¥kuk.c"
Output File: C:¥work¥kuk.h
Output File: C:¥work¥kuk.c
UFPK: EC6B8FA5C0D5DA5142CCAF3A31AEBEAE2346CFE7EF644B9B6B70523CBA0F5C5C
W-UFPK: 00000001102D464621E307E4FF7027D346B9A2DEA8E35A6673DC9685DE0283CBED82DE1E
IV: 2F5FE32A536036EBC11BE0FA7BBBC572
Encrypted key: A3E508B735C32C4F122A931B78EE0F552FB42EEDA3CA1D955B1191CAF8FAC4D39462C5C1562
EB36EAB2D9331102DF699
```

Figure 5.11 genkey Command Execution Example

Use the data from the output C source file to inject the KUK by calling the R_TSIP_GenerateUpdateKeyRingKeyIndex() 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 an RSA-2048 public key
   In the terminal emulator, run the following genkey command.

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

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

```
C:¥Renesas¥SecurityKeyMangementTool¥cli>skmt.exe /genkey /kuk file="C:¥work¥kuk.key" /mcu
"RX-TSIP" /keytype "RSA-2048-public" /key "bad47a84c1782e4dbdd913f2a261fc8b65838412c6e45a2
068ed6d7f16e9cdf4462b39119563cafb74b9cbf25cfd544bdae23bff0ebe7f6441042b7e109b9a8afaa056821
ef8efaab219d21d6763484785622d918d395a2a31f2ece8385a8131e5ff143314a82e21afd713bae817cc0ee35
14d4839007ccb55d68409c97a18ab62fa6f9f89b3f94a2777c47d6136775a56a9a0127f682470bef831fbec4bc
d7b5095a7823fd70745d37d1bf72b63c4b1b4a3d0581e74bf9ade93cc46148617553931a79d92e9e488ef47223
ee6f6c061884b13c9065b591139de13c1ea2927491ed00fb793cd68f463f5f64baa53916b46c818ab99706557a
1c2d50d232577d100010001" /filetype "csource" /keyname "rsa2048public" /output "C:¥work¥rsa
2048public.c"
Output File: C:¥work¥rsa2048public.h
Output File: C:¥work¥rsa2048public.c
KUK: D0AEC19726CBC0E2FB403866B9B465A6C0D05B7A60362D5F435F9A3E98C79084
IV: 9A537E8D13E62700DAF2DB0A44E770A9
Encrypted key: EFEB764914ACE4DA87737EC90A2B1B14520A7C18992DCDB7E945540A3220165E3E022CBC11F
B1A0C39563C8702ECF4C931B1BE4198CCA0DCA373D1395F7046D9AB02AF7D93C5512E9438370DB2E5027F4BFB8
606409F81D7CDB138F446474C9C04BA9613DC928EDD09B9C5D2002F54CF739A94FD8C5CD694A80AD12D9B00067
2EBBEE9D24298717D47FB05092A8178D173B45EA2BA47FA0C742084783FF718C593F8F6E2AC8E6013A9CA4EA25
73F8619CEF0B97DDC2F184665E42FF89009B9E8C89C5115D3733273E1432BD97EABB9E9190589F225E287F5C79
25597810F07E7A478BA0A464590C53A881739B8F89CC372B7055DD02051105113E2179D39ADE17D3420BB1F506
F96A345A4B6FB131A96E089C2ADAF5772D7F8F16A7F5B703FD3
```

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_TSIP_Update2048PublicKeyIndex() function to output an updated RSA-2048 public key index.

5.3.2.2 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.

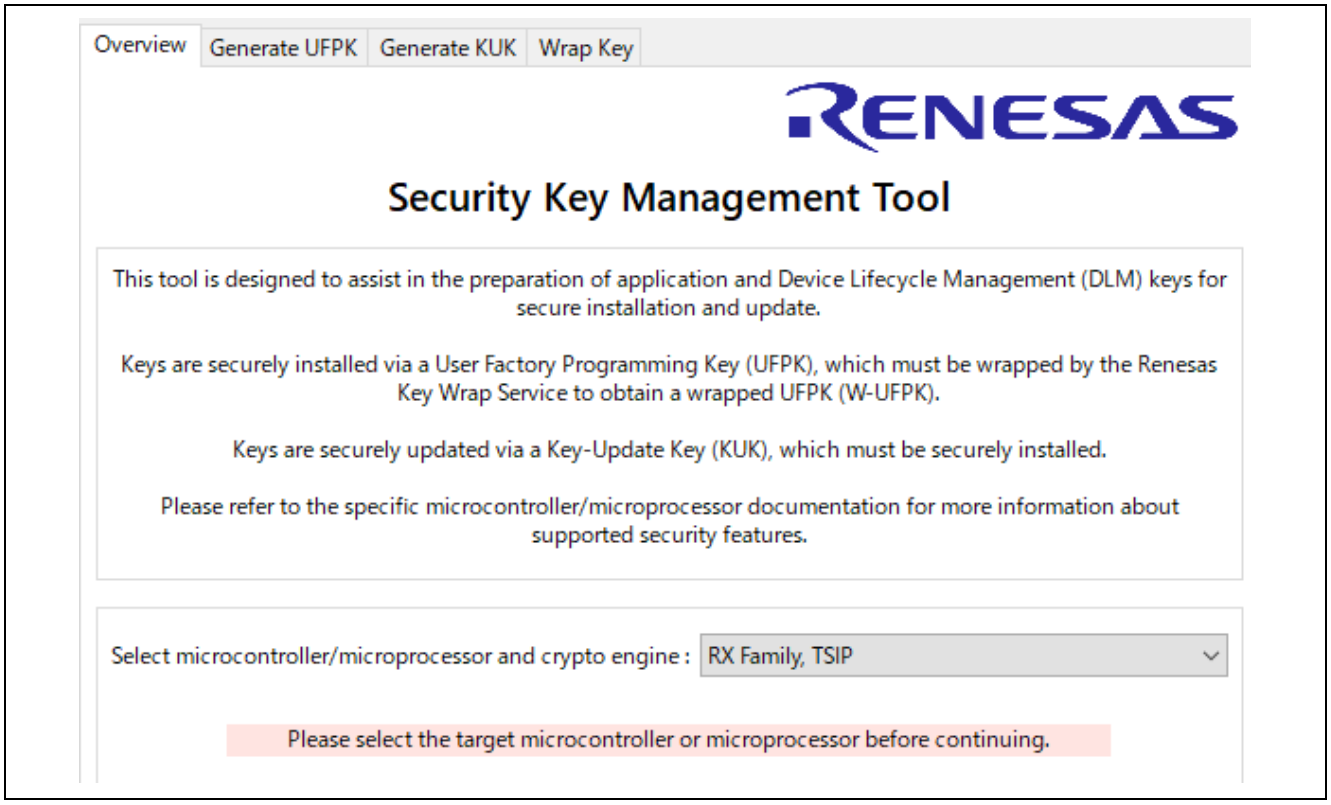


Figure 5.13 Overview Tab

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.

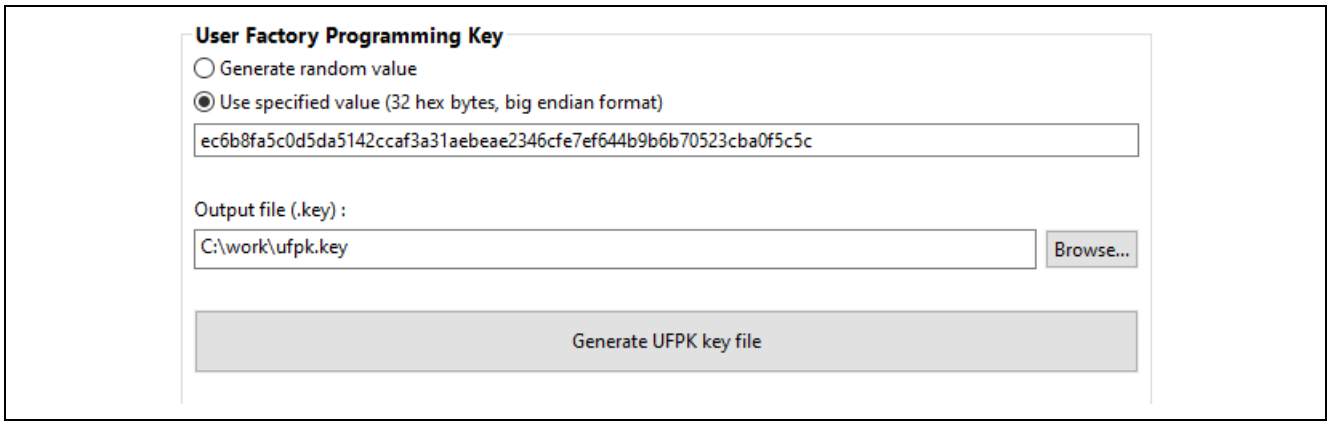


Figure 5.14 Example UFPK File Generation Settings on Generate UFPK Tab

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

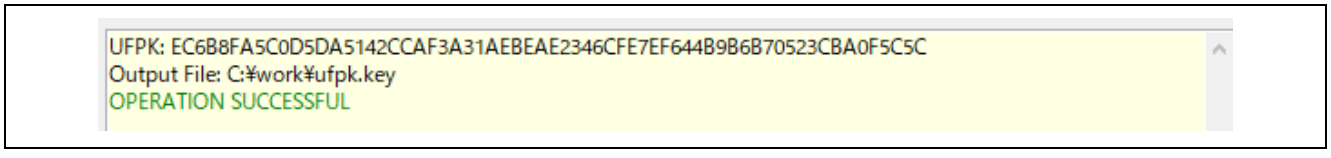


Figure 5.15 Generate UFPK Tab Execution Result

3. Obtaining a W-UFPK
   Send the 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 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.

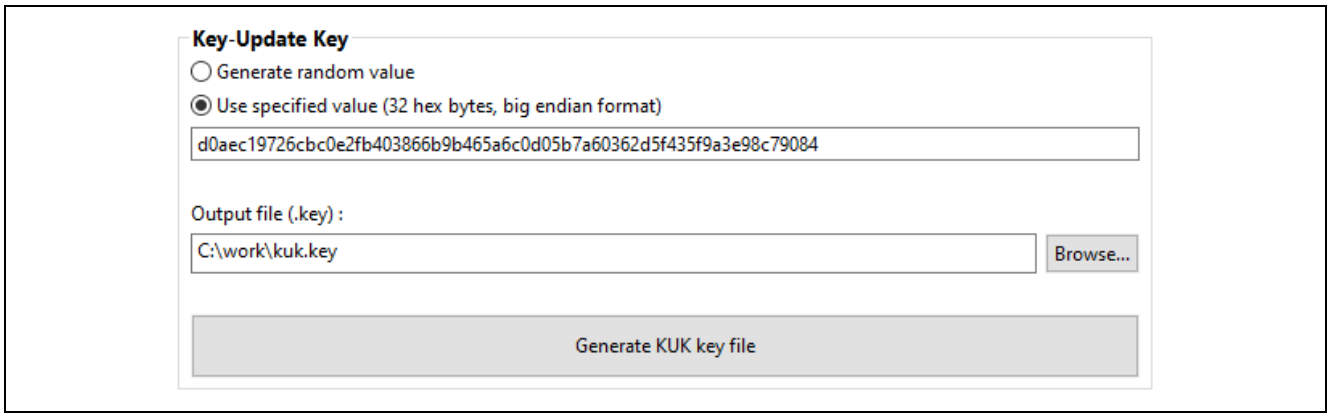


Figure 5.16 Example KUK File Generation Settings on Generate KUK Tab

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

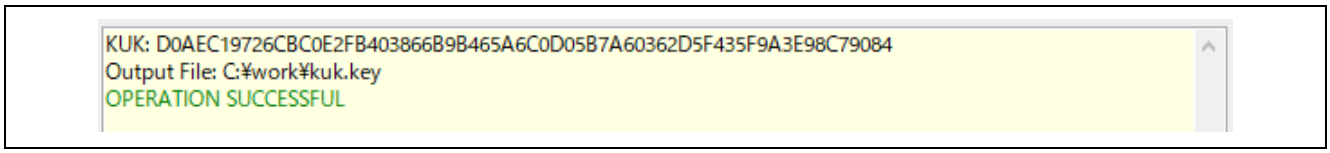


Figure 5.17 Generate KUK Tab Execution Result

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.

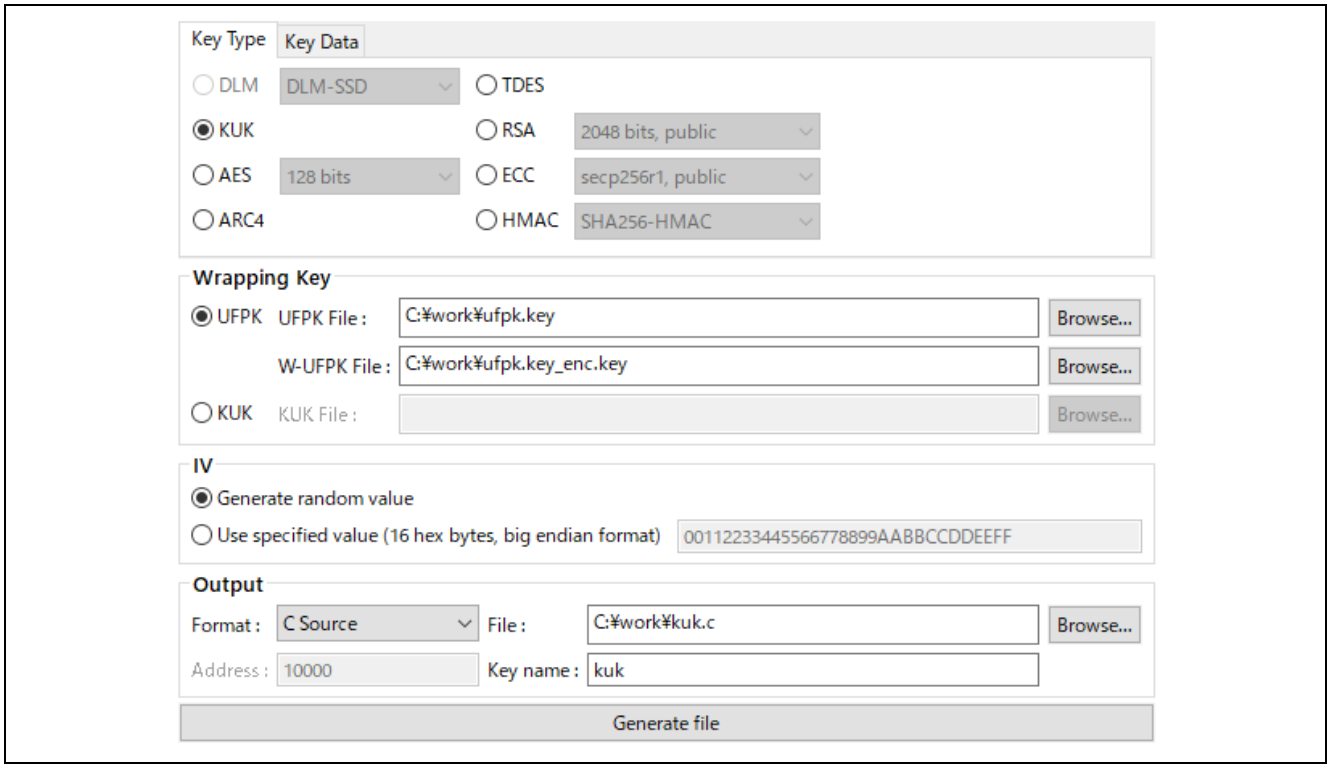


Figure 5.18 Example C Source File Output Settings for KUK File on Wrap Key – Key Type Tab

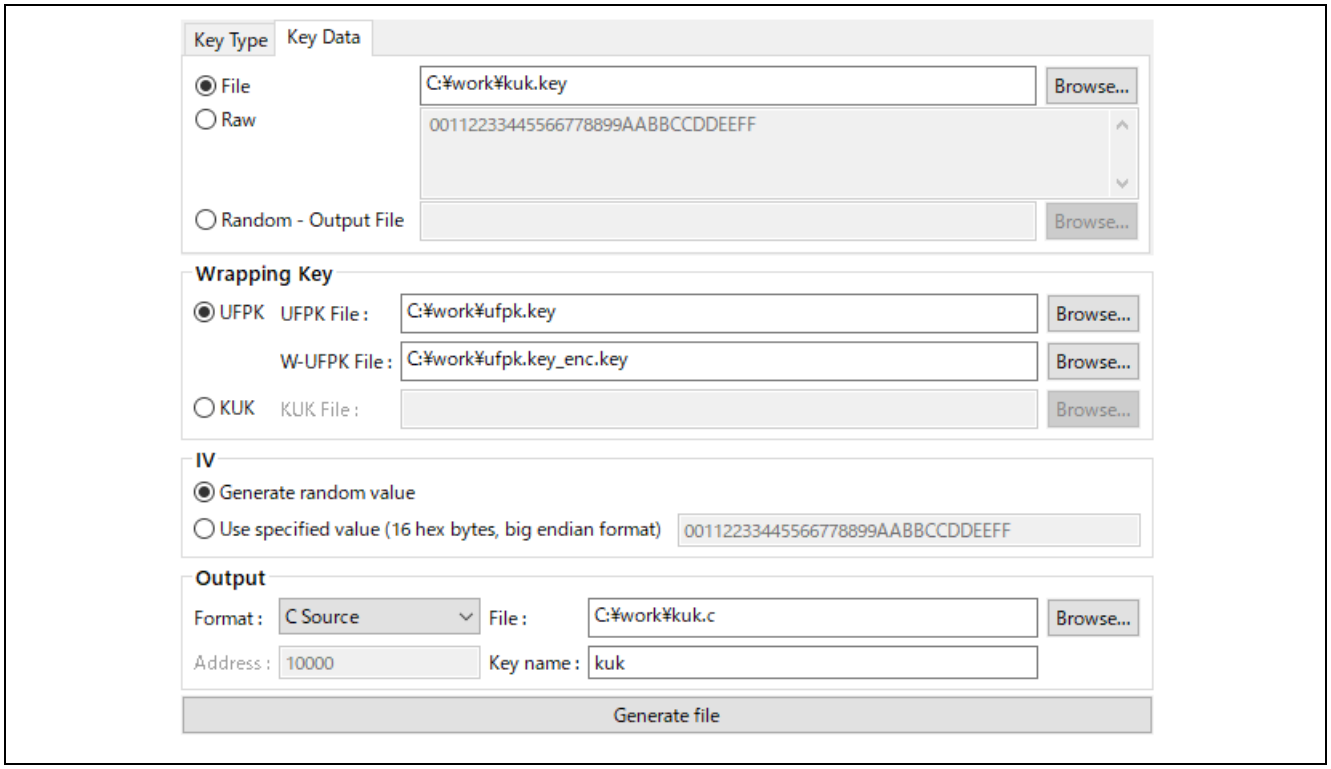


Figure 5.19 Example C Source File Output Settings for KUK File on Wrap Key – Key Data Tab

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

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

6. Generating a RSA 2048 public key file in C source file format
   On the Wrap Key tab, generate a RSA-2048 public key file in C source file format.
   On the Key Type tab, select **RSA** and **2048 bits, public**, and on the Key Data tab, enter the RSA 2048 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.

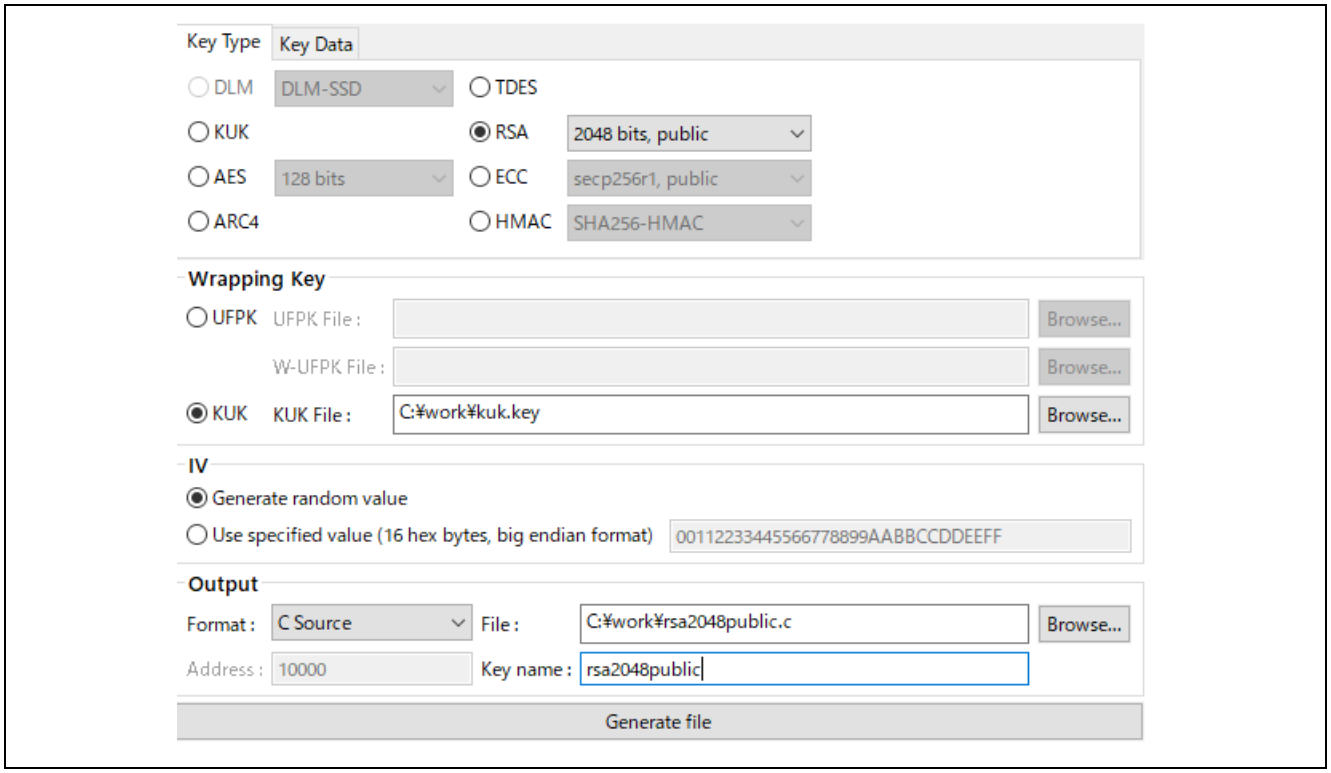


Figure 5.20 Example RSA 2048 Public Key C Source File Generation Settings on Wrap Key – Key Type Tab

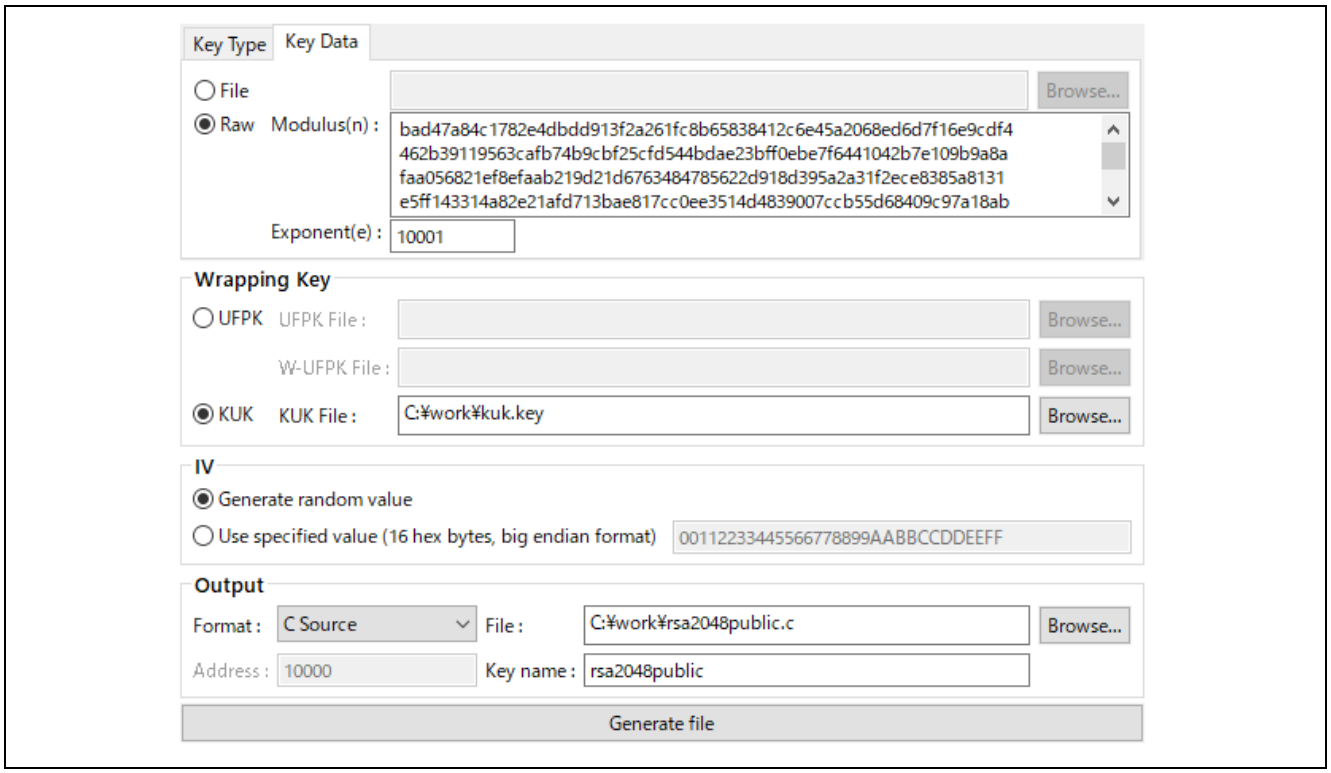


Figure 5.21 Example RSA 2048 Public Key C Source File Generation Settings on Wrap Key – Key Data Tab

The following is displayed when the operation completes successfully.

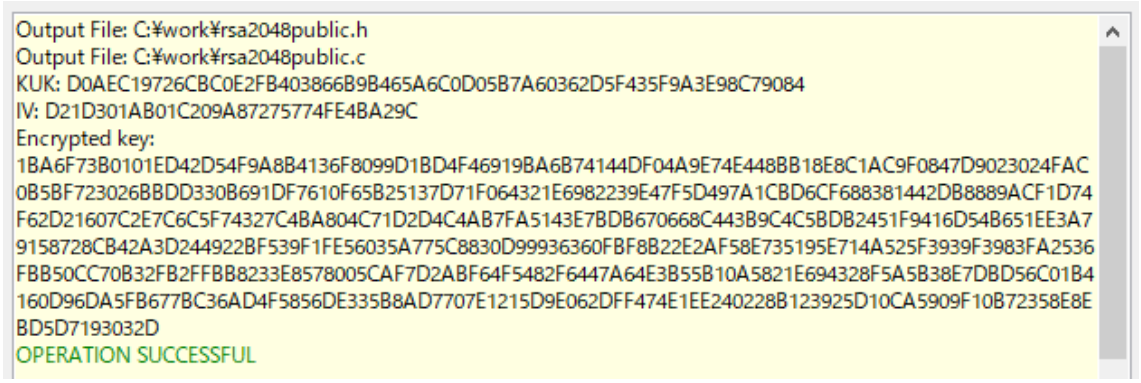


Figure 5.22 Wrap Key Tab Execution Result

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 the Demo Projects

With the exception of the AES cryptography project and TLS cooperation function project, the demo projects run on the MCUs listed in Table 6.1. The board which MCU is equipped is connected to the PC using a USB cable. Tera Term is used as the terminal emulator on the PC.

Table 6-1 Demo Projects to used on each MCU

MCU	Demo Project
RX231	rx231_rsk_tsip_sample
RX26T	rx26t_mcb_tsip_sample
RX65N	rx65n_2mb_rsk_tsip_sample
RX66T	rx66t_rsk_tsip_sample
RX671	rx671_rsk_tsip_sample
RX72M	rx72m_rsk_tsip_sample
RX72N	rx72n_rsk_tsip_sample
RX72T	rx72t_rsk_tsip_sample

Projects for MCUs which support Dual Bank mode make use of code flash linear mode.

Operation of the demo projects has been confirmed using little-endian byte ordering.

The description of demo project execution results, etc., in this section applies to the operation results on the RSK RX231 board. The terminal emulator software used is Tera Term.

Additional Notes

Demo projects for MCUs provided with a TSIP module require a link size larger than 128 KB. It is therefore necessary to purchase the product version of the CC-RX compiler in order to compile these projects.

There is no RSK currently available for the RX66N, so no demo projects are provided for this MCU. To confirm operation on the RX66N, refer to section 4 in the application note “RX Smart Configurator: User’s Guide: e^2 studio” (R20AN0451ES0140) and change the selected device to the RX66N.

The RX72N (R5F572NNHDBD) and RX66N (R5F566NNHDBD) are pin-compatible, so equivalent operation can be achieved simply by changing the device.

The TSIP driver uses the same target folder, so operation is possible using the same UFPK.

6.1.1 Setting Up the Demo Project

Wiring connections between the PC and board are shown below.

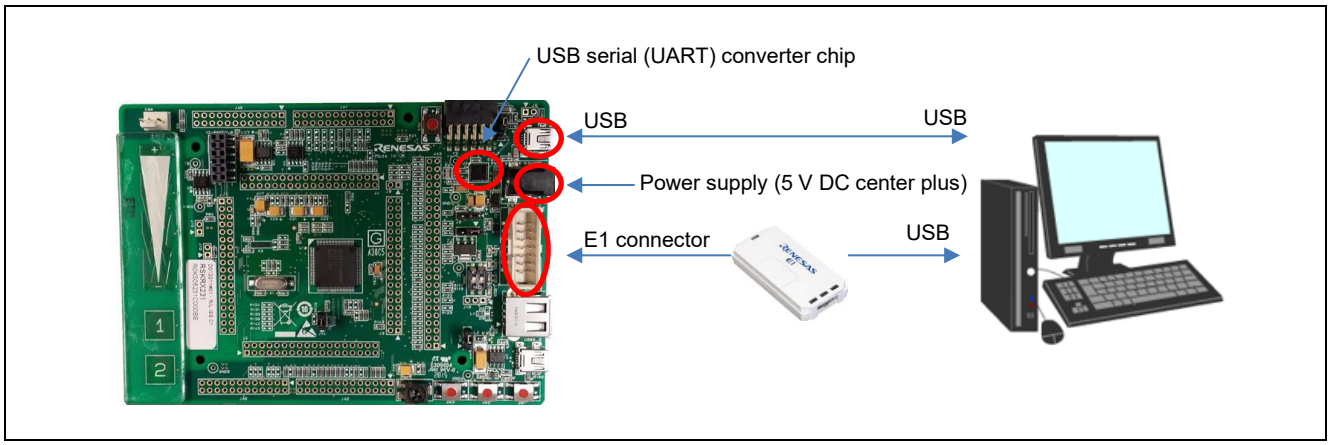


Figure 6.1 Wiring Connections between PC and RSK RX231 Board

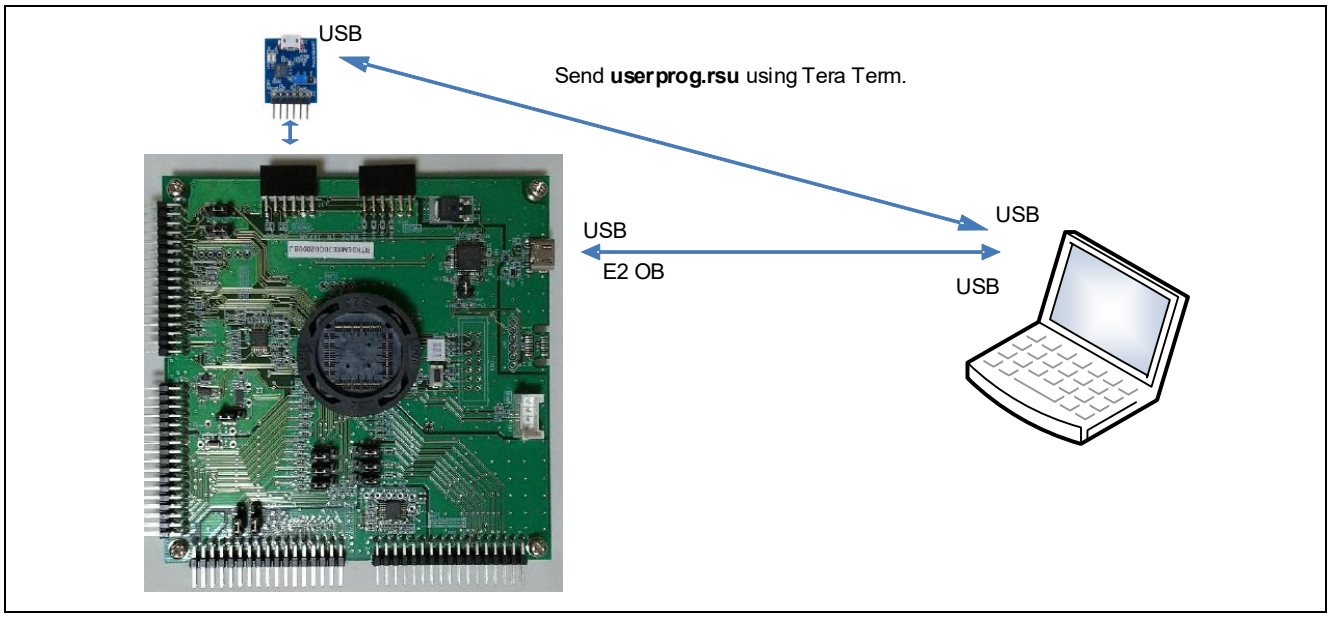


Figure 6.2 Wiring Connections between PC and MCB RX26T Board

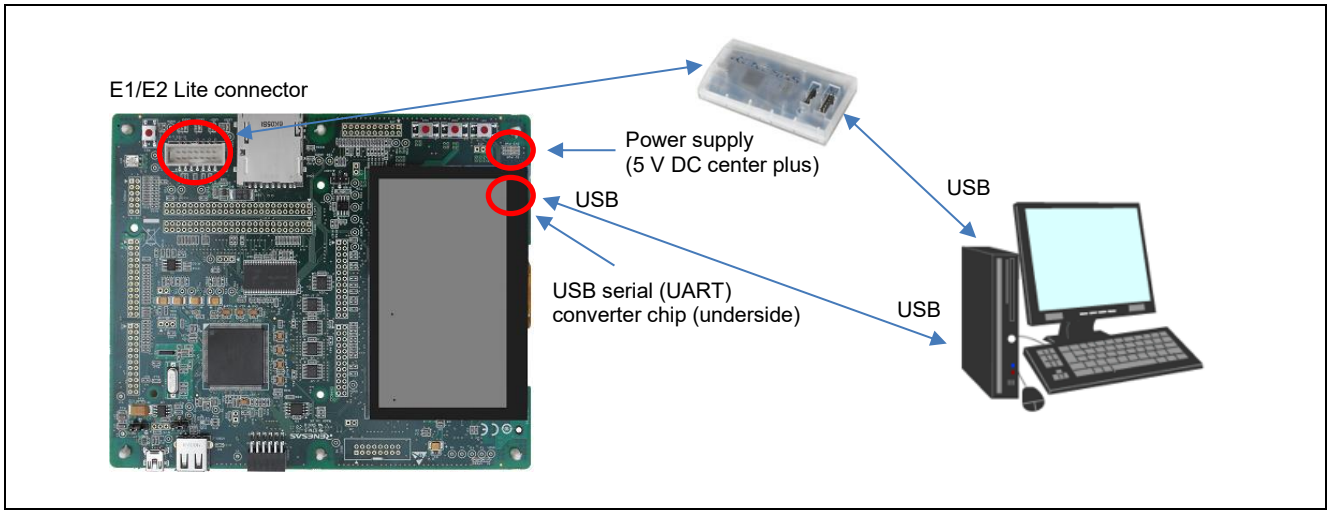


Figure 6.3 Wiring Connections between PC and RSK RX65N-2MB Board

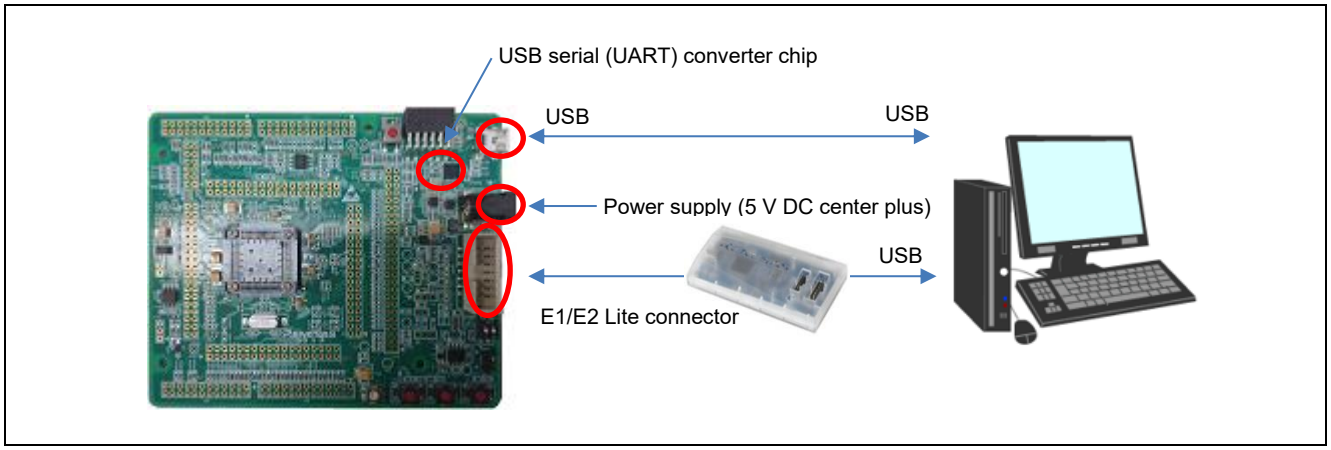


Figure 6.4 Wiring Connections between PC and RSK RX66T Board

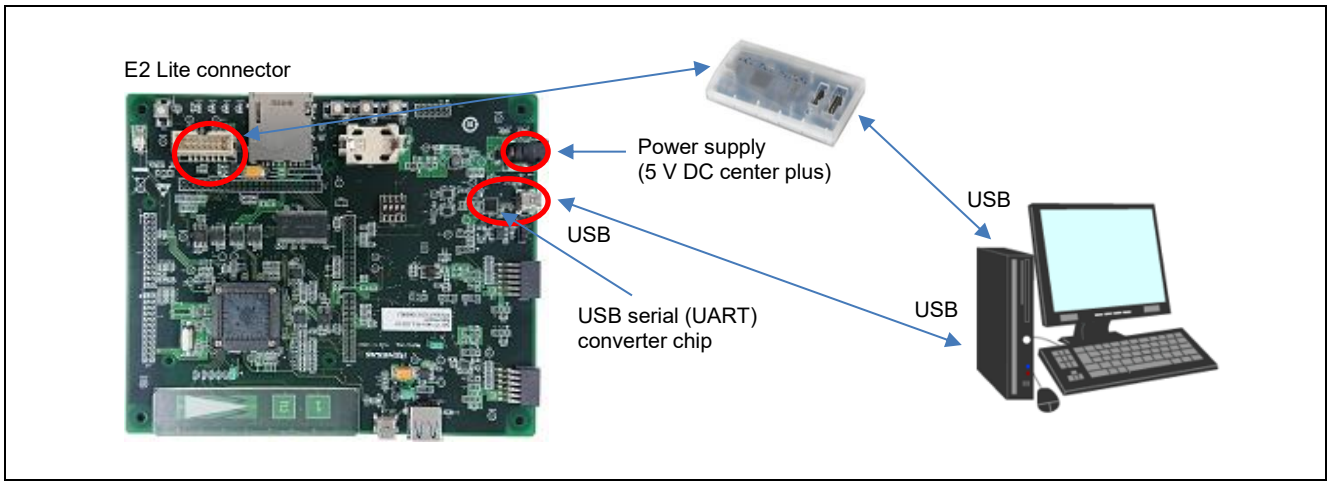


Figure 6.5 Wiring Connections between PC and RSK RX671 Board

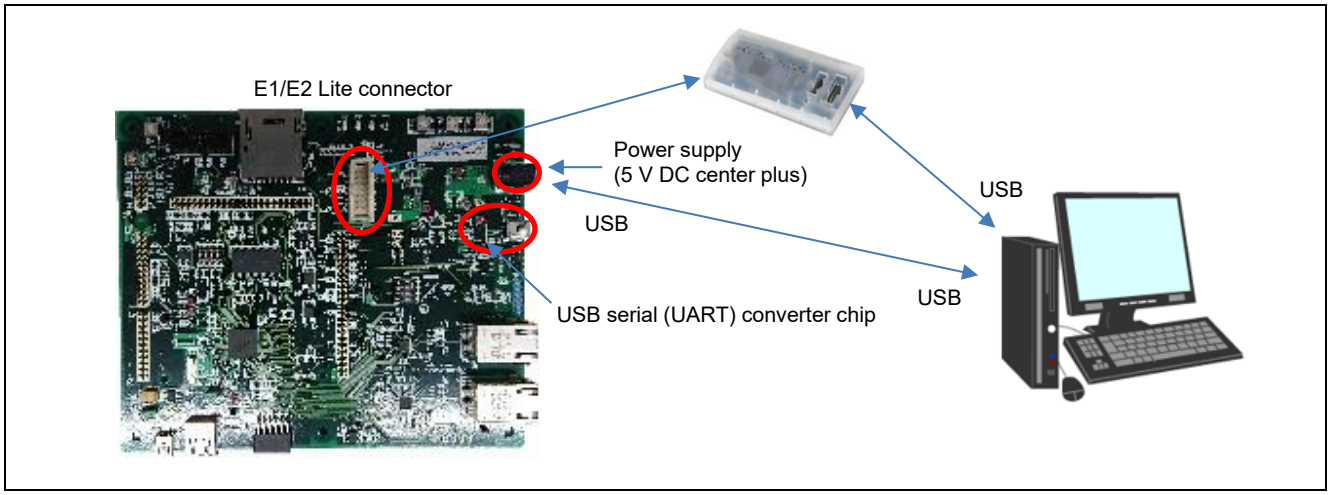


Figure 6.6 Wiring Connections between PC and RSK RX72M Board

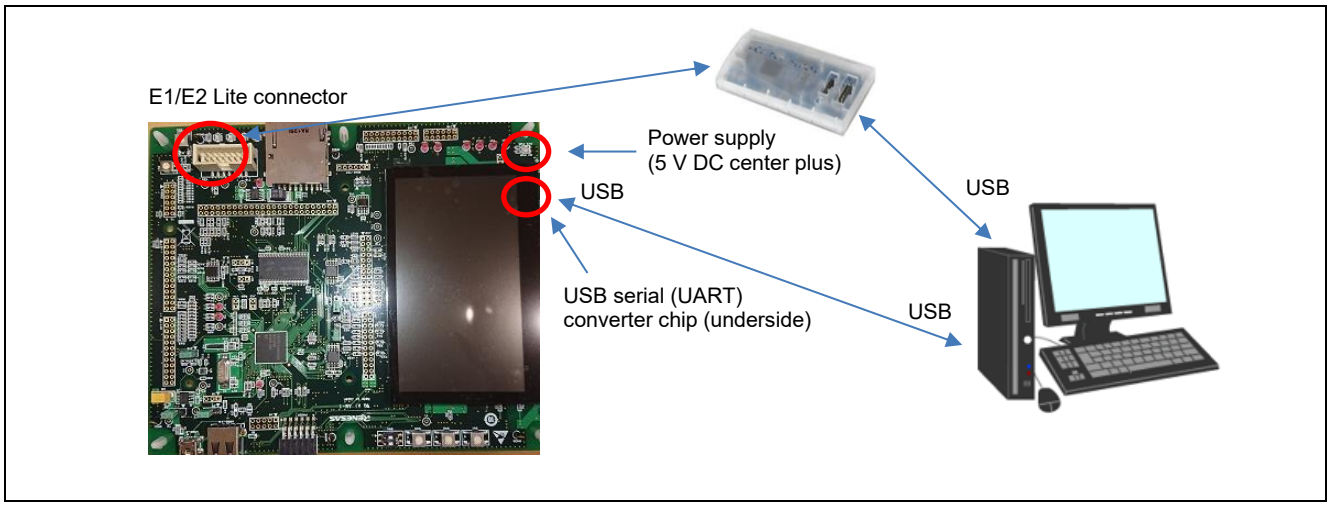


Figure 6.7 Wiring Connections between PC and RSK RX72N Board

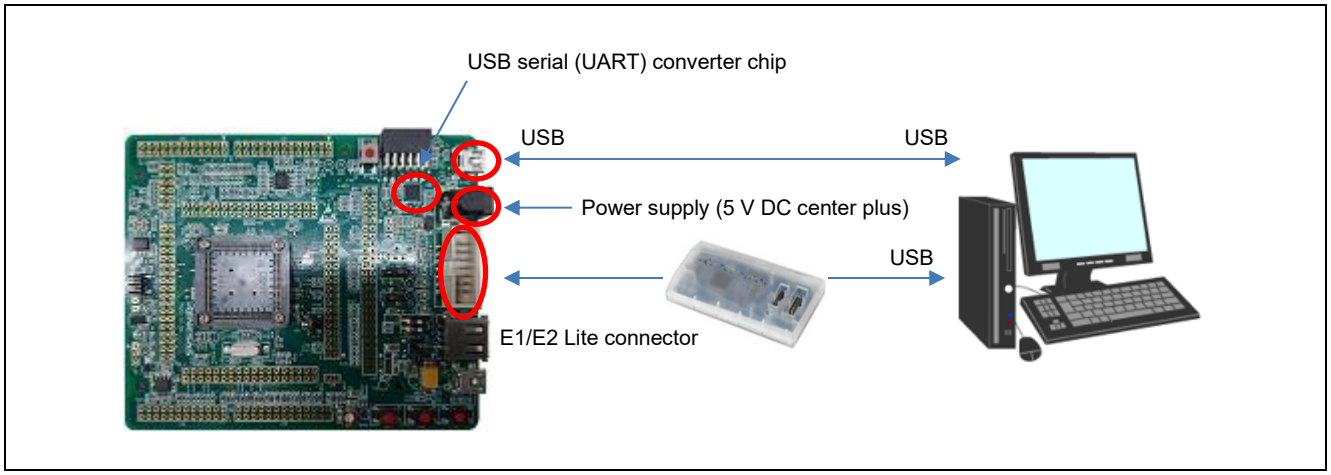


Figure 6.8 Wiring Connections between PC and RSK RX72T Board

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 key data structures used by the demo project are listed below.

Table 6-2 Key Data Structures of Demo Project

Name	Type	Description
st_key_block_data_t	—	Key data structure located in C_FIRMWARE_UPDATE_CONTROL_BLOCK and C_FIRMWARE_UPDATE_CONTROL_BLOCK _MIRROR
firmware_update_control_data	—	Structure for storing mac information during firmware updating
user_program_max_cnt	uint32_t	Firmware byte size of firmware update
lifecycle_state	uint32_t	Key data and user program*1 status There are two statuses: STATE_KEY_INJECT and STATE_EXEC_IMAGE. STATE_KEY_INJECT : Inject wrapped keys STATE_EXEC_IMAGE : Verify wrapped keys and execute program
program_mac0[]	uint32_t	MAC value of UPDATE_FIRMWARE_AREA
program_mac1[]	uint32_t	MAC value of UPDATE_TEMPORARY_AREA
key_data	—	Key data storage structure
encrypted_user_aes128_key[]	uint8_t	Encrypted key of AES 128 shared key
encrypted_user_aes256_key[]	uint8_t	Encrypted key of AES 256 shared key
encrypted_user_tdes_key[]*2	uint8_t	Encrypted key of TDES shared key
encrypted_user_arc4_key[]*2	uint8_t	Encrypted key of ARC4 shared key
encrypted_user_sha1hmac_key[]*2	uint8_t	Encrypted key of SHA-1 HMAC shared key
encrypted_user_sha256hmac_key[]*2	uint8_t	Encrypted key of SHA-256 HMAC shared key
encrypted_user_rsa1024_ne_key[]*2	uint8_t	Encrypted key of RSA 1024 public key
encrypted_user_rsa1024_nd_key[]*2	uint8_t	Encrypted key of RSA 1024 secret key
encrypted_user_rsa2048_ne_key[]*2	uint8_t	Encrypted key of RSA 2048 public key
encrypted_user_rsa2048_nd_key[]*2	uint8_t	Encrypted key of RSA 2048 secret key
encrypted_user_ecc192_public_key[]*2	uint8_t	Encrypted key of ECC 192 public key
encrypted_user_ecc192_private_key[]*2	uint8_t	Encrypted key of ECC 192 secret key
encrypted_user_ecc224_public_key[]*2	uint8_t	Encrypted key of ECC 224 public key
encrypted_user_ecc224_private_key[]*2	uint8_t	Encrypted key of ECC 224 secret key
encrypted_user_ecc256_public_key[]*2	uint8_t	Encrypted key of ECC 256 public key
encrypted_user_ecc256_private_key[]*2	uint8_t	Encrypted key of ECC 256 secret key
encrypted_user_update_key[]	uint8_t	Encrypted key of update keyring
encrypted_provisioning_key[]	uint8_t	Encrypted provisioning key generated by DLM server
iv[]	uint8_t	IV used for key injection
user_aes128_key_index	tsip_aes_key_index_t	AES 128 key index
user_aes256_key_index	tsip_aes_key_index_t	AES 256 key index
user_tdes_key_index*2	tsip_tdes_key_index_t	TDES key index
user_arc4_key_index*2	tsip_arc4_key_index_t	ARC4 key index
user_sha1hmac_key_index*2	tsip_hmac_sha_key_index_t	SHA-1 HMAC key index
user_sha256hmac_key_index*2	tsip_hmac_sha_key_index_t	SHA-256 HMAC key index
user_rsa1024_ne_key_index*2	tsip_rsa1024_public_key_index_t	RSA 1024 public key index
user_rsa1024_nd_key_index*2	tsip_rsa1024_private_key_index_t	RSA 1024 secret key index
user_rsa2048_ne_key_index*2	tsip_rsa2048_public_key_index_t	RSA 2048 public key index
user_rsa2048_nd_key_index*2	tsip_rsa2048_private_key_index_t	RSA 2048 secret key index
user_ecc192_public_key_index*2	tsip_ecc_public_key_index_t	ECC 192 public key index
user_ecc192_private_key_index*2	tsip_ecc_private_key_index_t	ECC 192 secret key index
user_ecc224_public_key_index*2	tsip_ecc_public_key_index_t	ECC 224 public key index

Name		Type	Description
	user_ecc224_private_key_index*2	tsip_ecc_private_key_index_t	ECC 224 secret key index
	user_ecc256_public_key_index*2	tsip_ecc_public_key_index_t	ECC 256 public key index
	user_ecc256_private_key_index*2	tsip_ecc_private_key_index_t	ECC 256 secret key index
hash_sha1[]		uint8_t	SHA1 hash value of firmware_update_control_data and key_data

Notes: 1. Key data status only in the demo project.
The user program status is used by the secure boot project (see 6.2 Confirming Operation of the Secure Bootloader and Firmware Update Demo Projects).

2. When a device provided with a TSIP-Lite module is used, TDES, ARC4, HMAC, RSA, and ECC key information is not used. In addition, this data is not present in the key_data.c and key_data.h files appended to the demo project versions for devices provided with a TSIP-Lite module. The key_data.c and key_data.h files appended to the demo project for the RX671 do not contain ARC4 and RSA-2048 data.

Main data and mirror data blocks are provided in the data flash for the demo project's key information structures. (In the figure below, the main data block is C_FIRMWARE_UPDATE_CONTROL_BLOCK and the mirror data block is C_FIRMWARE_UPDATE_CONTROL_BLOCK_MIRROR.) This prevents loss of the key information in case of a write failure due to circumstances such as a power interruption while updating the key information.

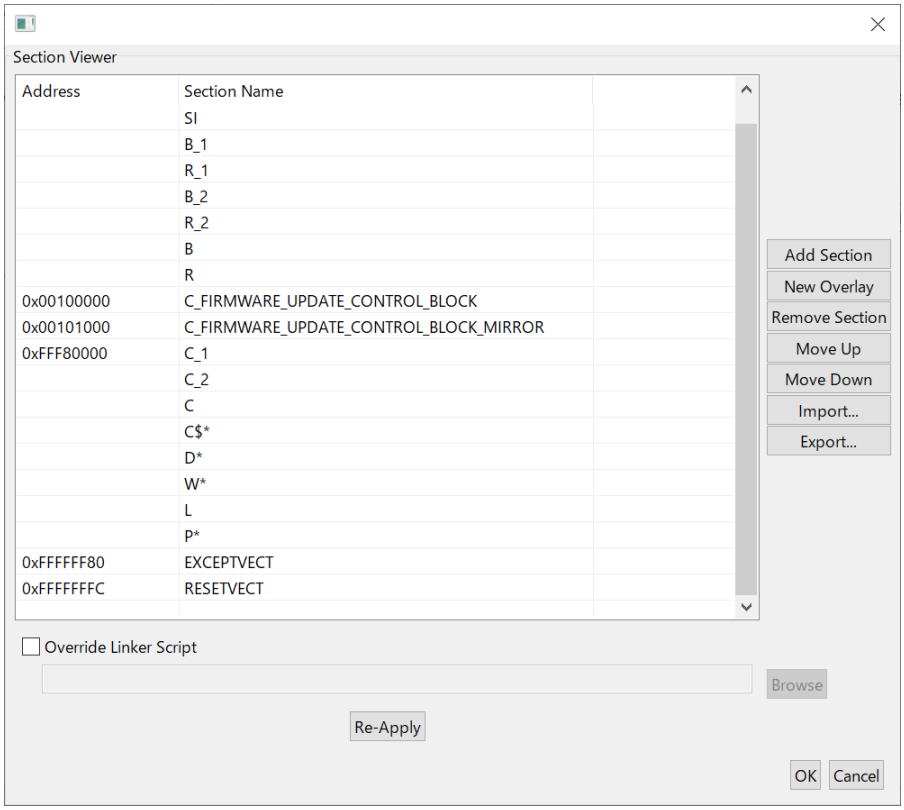


Figure 6.9 RX231 Demo Project Sections

In key_data.c and key_data.h, definition file of encrypted key in "demo project folder"/src/genkey is included. Refer to 5.3.1 Key Injection Procedure to see haw to generate the files.

The demo project implements the following commands.

Table 6-3 Demo Project Commands

Command	Operation
display	Displays the wrapped key.
encdemo [Arg1]	Encrypts the Arg1 value in AES 128-bit mode.
function	Performs the following tests:*1 AES-128/256 ECB, CBC, CTR, GCM, the CCM encryption and decryption AES-128/256 CMAC and SHA-HMAC generation and verification TDES encryption and decryption ARC4 encryption and decryption (RSA-1024 and RSA-2048 only) RSAES-PKCS1_v1_5 encryption and decryption (RSA-1024 and RSA-2048 only) RSASSA-PKCS1_v1_5 signature generation and verification ECDSA signature generation and verification tests (excluding P-384)
random	Generates a random value using R_TSIP_GenerateRandomNumber().

Note: 1. SHA-HMAC generation and verification, TDES encryption and decryption, ARC4 encryption and decryption, RSAES-PKCS1_v1_5 encryption and decryption, RSASSA-PKCS1_v1_5 signature generation and verification, and ECDSA signature generation and verification are not implemented in the demo projects for devices provided with a TSIP-Lite module. And ARC4 and RSA-2048 are not supported in RX671 demo project.

The default values of keys are as follows and IV is “55aa55aa55aa55aa55aa55aa55aa55aa” in the demo project.

Table 6-4 Demo Project Default Key Values

Key Type	Value
UFPK	22222222222222222222222222222221111111111111111111111111111111
AES 128-bit	11111111222222223333333344444444
AES 256-bit	ffffffffeeeeeeeeddddddddccccccccbbbbbbbbaaaaaaaa0000000099999999
3 TDES*2	01010101010101010202020202020202040404040404040404
ARC4 2048-bit*2	29041972fb42ba5fc7127712f13829c929041972fb42ba5fc7127712f13829c9 29041972fb42ba5fc7127712f13829c929041972fb42ba5fc7127712f13829c9 29041972fb42ba5fc7127712f13829c929041972fb42ba5fc7127712f13829c9 29041972fb42ba5fc7127712f13829c929041972fb42ba5fc7127712f13829c9 29041972fb42ba5fc7127712f13829c929041972fb42ba5fc7127712f13829c9 29041972fb42ba5fc7127712f13829c929041972fb42ba5fc7127712f13829c9 29041972fb42ba5fc7127712f13829c929041972fb42ba5fc7127712f13829c9 29041972fb42ba5fc7127712f13829c929041972fb42ba5fc7127712f13829c9
SHA-1 HMAC*2	000102030405060708090a0b0c0d0e0f10111213
SHA-256 HMAC*2	000102030405060708090a0b0c0d0e0f101112131415161718191a1b1c1d1e1f
RSA 1024-bit public *1*2	ccd6cb86f59ffea97c278b7cf395fa56f3709a958bce1e6e3ae196b471f4517f 64d32d81d10bcea5b55b9ea659c3b9db1854a696b801a8e72439265e85bc6138 cf874f45fe22a2477fe8337671357db67180b1ac9a0e84c098dbcdca8a3e6a72 dfcee7e6907ede2d53aa726e8e97ea26fcba3bbfa7bf7bfbbb70f1ec1d153125 00010001
RSA 1024-bit private *1*2	ccd6cb86f59ffea97c278b7cf395fa56f3709a958bce1e6e3ae196b471f4517f 64d32d81d10bcea5b55b9ea659c3b9db1854a696b801a8e72439265e85bc6138 cf874f45fe22a2477fe8337671357db67180b1ac9a0e84c098dbcdca8a3e6a72 dfcee7e6907ede2d53aa726e8e97ea26fcba3bbfa7bf7bfbbb70f1ec1d153125 096ed2cc923f1df11c208e11e0fdc51b7ff66d87f97a32788d099a7110d65972 6e68332e493c2bf6019608864c97f0d52017b5dc36f90c982858e16574ef29e2 e8b6f4386bdbb1beae60390aaac0160ce787f98ba34c83d6612badbc99d17666 04e72b394d1991fc4ded87e11cf165caae0cc652771b8395ed5b9a99a9fa5781

Key Type	Value
RSA 2048-bit public *1*2	bad47a84c1782e4dbdd913f2a261fc8b65838412c6e45a2068ed6d7f16e9cdf4 462b39119563cafb74b9cbf25cfd544bdae23bff0ebe7f6441042b7e109b9a8a faa056821ef8efaab219d21d6763484785622d918d395a2a31f2ece8385a8131 e5ff143314a82e21afd713bae817cc0ee3514d4839007ccb55d68409c97a18ab 62fa6f9f89b3f94a2777c47d6136775a56a9a0127f682470bef831fbec4bcd7b 5095a7823fd70745d37d1bf72b63c4b1b4a3d0581e74bf9ade93cc4614861755 3931a79d92e9e488ef47223ee6f6c061884b13c9065b591139de13c1ea292749 1ed00fb793cd68f463f5f64baa53916b46c818ab99706557a1c2d50d232577d1 00010001
RSA 2048-bit private *1*2	bad47a84c1782e4dbdd913f2a261fc8b65838412c6e45a2068ed6d7f16e9cdf4 462b39119563cafb74b9cbf25cfd544bdae23bff0ebe7f6441042b7e109b9a8a faa056821ef8efaab219d21d6763484785622d918d395a2a31f2ece8385a8131 e5ff143314a82e21afd713bae817cc0ee3514d4839007ccb55d68409c97a18ab 62fa6f9f89b3f94a2777c47d6136775a56a9a0127f682470bef831fbec4bcd7b 5095a7823fd70745d37d1bf72b63c4b1b4a3d0581e74bf9ade93cc4614861755 3931a79d92e9e488ef47223ee6f6c061884b13c9065b591139de13c1ea292749 1ed00fb793cd68f463f5f64baa53916b46c818ab99706557a1c2d50d232577d1 40d60f24b61d76783d3bb1dc00b55f96a2a686f59b3750fdb15c40251c370c65 cada222673811bc6b305ed7c90ffcb3abdddc8336612ff13b42a75cb7c88fb93 6291b523d80acce5a0842c724ed85a1393faf3d470bda8083fa84dc5f3149984 4f0c7c1e93fb1f734a5a29fb31a35c8a0822455f1c850a49e8629714ec6a2657 efe75ec1ca6e62f9a3756c9b20b4855bdc9a3ab58c43d8af85b837a7fd15aa11 49c119cfe960c05a9d4cea69c9fb6a897145674882bf57241d77c054dc4c94e8 349d376296137eb421686159cb878d15d171eda8692834afc871988f203fc822 c5dcee7f6c48df663ea3dc755e7dc06aebd41d05f1ca2891e2679783244d068f
ECC 192-bit public*2 (top: Qx, bottom: Qy)	fba2aac647884b504eb8cd5a0a1287babcc62163f606a9a2 dae6d4cc05ef4f27d79ee38b71c9c8ef4865d98850d84aa5
ECC 192-bit private*2	7891686032fd8057f636b44b1f47cce564d2509923a7465b
ECC 224-bit public*2 (top: Qx, bottom: Qy)	4c741e4d20103670b7161ae72271082155838418084335338ac38fa4 db7919151ac28587b72bad7ab180ec8e95ab9e2c8d81d9b9d7e2e383
ECC 224-bit private*2	888fc992893bdd8aa02c80768832605d020b81ae0b25474154ec89aa
ECC 256-bit public*2 (top: Qx, bottom: Qy)	1ccbe91c075fc7f4f033bfa248db8fccd3565de94bbfb12f3c59ff46c271bf83 ce4014c68811f9a21a1fdb2c0e6113e06db7ca93b7404e78dc7ccd5ca89a4ca9
ECC 256-bit private*2	519b423d715f8b581f4fa8ee59f4771a5b44c8130b4e3eacca54a56dda72b464

Notes: 1. The RSA 1024-bit key pair has the following values:

Modulus n:

ccd6cb86f59ffea97c278b7cf395fa56f3709a958bce1e6e3ae196b471f4517f
64d32d81d10bcea5b55b9ea659c3b9db1854a696b801a8e72439265e85bc6138
cf874f45fe22a2477fe8337671357db67180b1ac9a0e84c098dbcdca8a3e6a72
dfcee7e6907ede2d53aa726e8e97ea26fcba3bbfa7bf7bfbbb70f1ec1d153125

Exponent e: 00010001

Decryption Exponent d:

096ed2cc923f1df11c208e11e0fdc51b7ff66d87f97a32788d099a7110d65972
6e68332e493c2bf6019608864c97f0d52017b5dc36f90c982858e16574ef29e2
e8b6f4386bdbb1beae60390aaac0160ce787f98ba34c83d6612badbc99d17666
04e72b394d1991fc4ded87e11cf165caae0cc652771b8395ed5b9a99a9fa5781

The RSA 2048-bit key pair has the following values:

Modulus n:

bad47a84c1782e4dbdd913f2a261fc8b65838412c6e45a2068ed6d7f16e9cdf4
462b39119563cafb74b9cbf25cfd544bdae23bff0ebe7f6441042b7e109b9a8a
faa056821ef8efaab219d21d6763484785622d918d395a2a31f2ece8385a8131
e5ff143314a82e21afd713bae817cc0ee3514d4839007ccb55d68409c97a18ab

62fa6f9f89b3f94a2777c47d6136775a56a9a0127f682470bef831fbec4bcd7b
5095a7823fd70745d37d1bf72b63c4b1b4a3d0581e74bf9ade93cc4614861755
3931a79d92e9e488ef47223ee6f6c061884b13c9065b591139de13c1ea292749
1ed00fb793cd68f463f5f64baa53916b46c818ab99706557a1c2d50d232577d1

Exponent e: 00010001

Decryption Exponent d:

40d60f24b61d76783d3bb1dc00b55f96a2a686f59b3750fdb15c40251c370c65
cada222673811bc6b305ed7c90ffcb3abdddc8336612ff13b42a75cb7c88fb93
6291b523d80acce5a0842c724ed85a1393faf3d470bda8083fa84dc5f3149984
4f0c7c1e93fb1f734a5a29fb31a35c8a0822455f1c850a49e8629714ec6a2657
efe75ec1ca6e62f9a3756c9b20b4855bdc9a3ab58c43d8af85b837a7fd15aa11
49c119cfe960c05a9d4cea69c9fb6a897145674882bf57241d77c054dc4c94e8
349d376296137eb421686159cb878d15d171eda8692834afc871988f203fc822
c5dcee7f6c48df663ea3dc755e7dc06aebd41d05f1ca2891e2679783244d068f

2. The demo projects for devices provided with a TSIP-Lite module do not use TDES, ARC4, HMAC, RSA, and ECC key information. In addition, this data is not present in the key_data.c and key_data.h files appended to the demo project versions for devices provided with a TSIP-Lite module. The key_data.c and key_data.h files appended to the demo project for the RX671 do not contain ARC4 and RSA-2048 data.

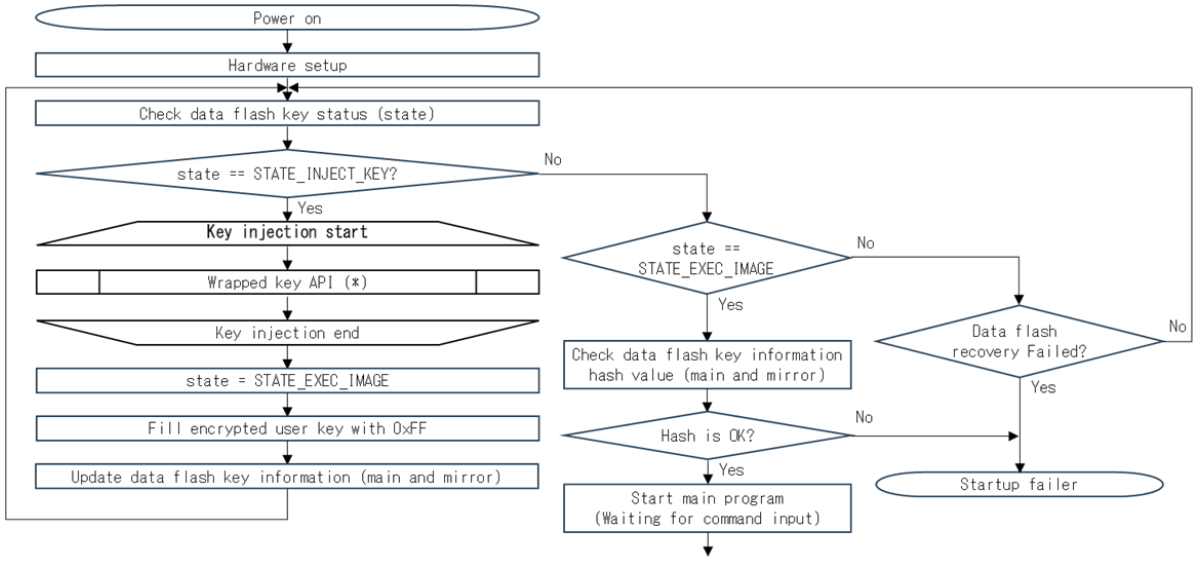
After the program is downloaded to the MCU, it performs key injection the first time it is run. Key injection converts a user key into wrapped key for use by the TSIP and writes it to the on-chip data flash. Figure 6.9 is a flowchart of the key injection processing that takes place when the program starts. When key injection completes successfully, the program is ready for command/response execution. The information displayed at this time is shown in section 6.1.3.

The demo project uses the following value as the UFPK:

UFPK = 22222222222222222222222222222222111111111111111111111111111111111

A UFPK file, sample.key, and W-UFPK file, sample.key_enc.key, that are usable by Security Key Management Tool are provided in the key folder of the demo project.

When the program is run from a state other than when it has been newly downloaded to the MCU, it merely checks the key information. The information displayed at this time is shown in section 6.1.3.



(*) Injecting wrapped keys described in Table 6-4.

Figure 6.10 Flowchart of Key Injection at Program Startup

6.1.3 Demo Project Execution Example

Figure 6.10 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 6.11 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.

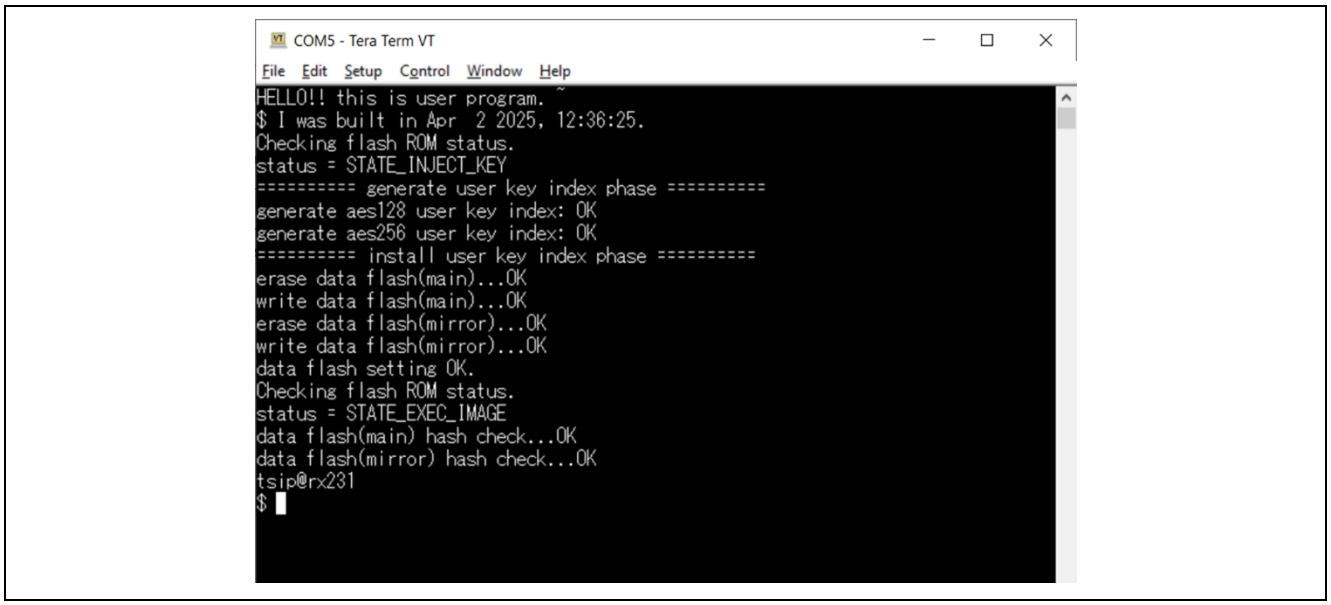


Figure 6.11 Tera Term Display (Initial Execution on RSK RX231 Board)

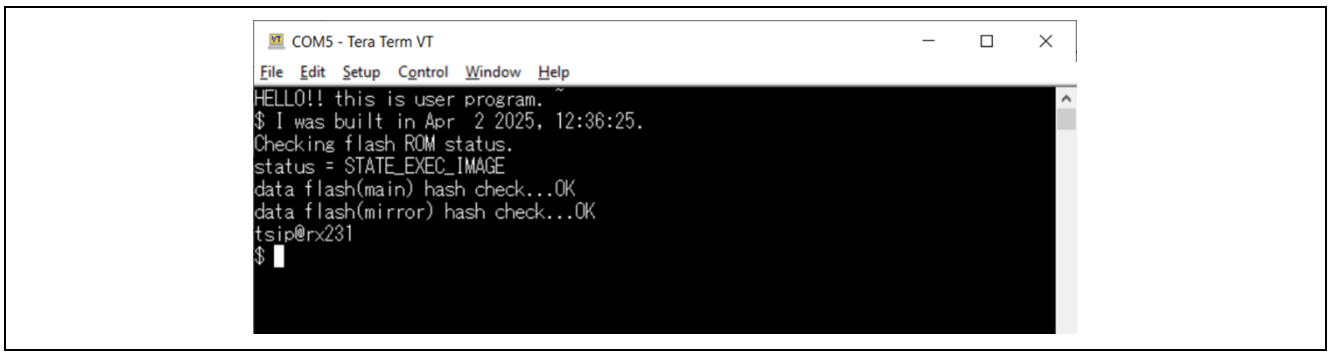


Figure 6.12 Tera Term Display (2nd and Subsequent Executions on RSK RX231 Board)

Figure 6.12 shows a usage example in which the command and argument **encdemo aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa** 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, 0x33, 0x44, 0x44, 0x44, 0x44**, the result is as shown in the screenshot below.

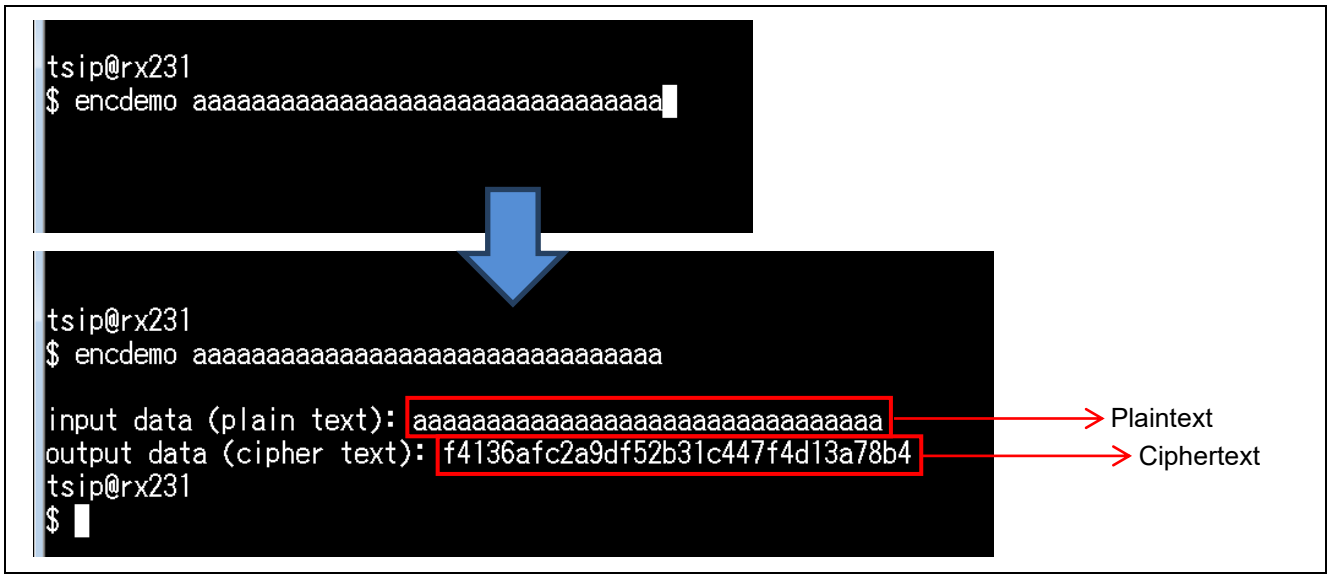


Figure 6.13 Tera Term Display (encdemo Command)

[Key Point 1]

The user key is stored in the data flash area (0x00100000), but make sure the user key specified in Security Key Management Tool (default value: 0x11, 0x11, 0x11, 0x11, 0x22, 0x22, 0x22, 0x22, 0x33, 0x33, 0x33, 0x33, 0x44, 0x44, 0x44, 0x44) is not exposed.

This is important to ensure that the user key cannot be discovered by reverse engineering the software.

[Key Point 2]

The wrapped key is tied to the HUK. It is not possible to use a wrapped key generated on one chip on another chip by copying it over. This is because the TSIP driver verifies the wrapped key by checking it against the HUK.

This is important to prevent dead copying of software.

[Key Point 3]

The wrapped key is always output as a different value even when the same user key is input. This is because a random number is used when generating the wrapped key.

This is important to reduce the risk of the user key being guessed based on the wrapped key.

6.2 Confirming Operation of the Secure Bootloader and Firmware Update Demo Projects

The secure bootloader and firmware update demo projects use the API functions listed in 4.2.16, Firmware Update, to decrypt on the device a user program encrypted as described in 3.14.3, Encrypting the User Program. The term firmware update refers to the process of writing the encrypted user program to the on-chip flash ROM following verification. Secure boot refers to the process of verifying the user program before execution. The secure bootloader and firmware update demo projects are sample programs that implement firmware update and secure boot operations.

Two demo projects are provided, a secure bootloader project and a firmware update project. The secure bootloader project includes firmware update functionality to write the initial firmware to the device.

Secure bootloader project: rx*XXX*_*bbb*_tsip_secure_boot*[1]

Firmware update project: rx*XXX*_*bbb*_tsip_user_program*[1]

The firmware update functionality of the demo projects supports use of a UART*[2,3] or USB*[4] 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*[5] containing both the secure bootloader and the encrypted user program for use when performing programming at the factory (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. The versions of projects for devices without USB support only provide support for UART. Project versions that do not support USB have _sci in the project name. Also, the letters *bbb* are replaced by the board name. RSK is indicated by rsk, MCB by mcb, and Envision Kit by ek.
2. For devices that are incompatible with dual bank mode (the RX231, RX66T, and RX72T), use the PMOD USB-UART converter board for UART flow control. It is also necessary to use the PMOD USB-UART converter board with boards lacking USB-serial conversion capability (the MCB RX26T).
3. For devices that are compatible with dual bank mode (the RX65N, RX671, RX72M, and RX72N) and when to use RSU header Ver.1, can use 1 area update function and which firmware update operation can be executed without using PMOD USB-UART converter board. However, when to use RSU header Ver.2, the 1 area update function can not be used and PMOD USB-UART converter board for UART flow control is required.
4. The RX66T and RX26T do not have USB interface support, so there is no demo project that uses USB provided for these devices.
5. Motorola S format files containing a secure bootloader and an encrypted user program can be decrypted by a corresponding RX TSIP–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 version of the FIT modules which is used in the dem oprojects are shown below. Some of the internal functions which are used to execute secure boot procedure are allocated to the SECURE_BOOT section.

Table 6-5 FIT modules used in the demo projects

FIT modules	version
r_bsp [Note]	7.54
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	1.44
r_usb_hmsc	1.44
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 of RX231. Please refer to 6.2.6 for details.

6.2.1 Demo Project Setup

The demo projects will run on the boards listed in Table 6.5, Boards Used in Demo Projects and Their Settings. To use the USB interface, set the jumper on the RSK board to the USB-HOST side to enable USB Host mode on the board. The factory default setting is the USB-FUNCTION side. For devices that do not support dual bank mode and projects using RSU header Ver.2, use the PMOD USB-UART converter board for status indications and transfer of the update program.

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

Boards	Jumper setting	Flash Mode	Note
RSK RX231	J15 : 1-2 short	-	When using UART as the interface, the 0 Ω resistors on the RSK board must be replaced. A PMOD USB-UART converter board is required.
MCB RX26T	-	Dual Bank Mode	A PMOD USB-UART converter board is required. Supports UART only.
RSK RX65N	J7 : 1-2 short J16 : 2-3 short	Dual Bank Mode	When using RSU header Ver.2, a PMOD USB-UART converter board is required.
RSK RX66T	-	-	A PMOD USB-UART converter board is required. Supports UART only.
RSK RX671	J8 : 2-3 short J13 : 2-3 short	Dual Bank Mode	When using RSU header Ver.2, a PMOD USB-UART converter board is required.
RSK RX72M	J8 : 2-3 short J10 : 2-3 short	Dual Bank Mode	When using RSU header Ver.2, a PMOD USB-UART converter board is required.
RSK RX72N	J7 : 2-3 short J8 : 2-3 short	Dual Bank Mode	When using RSU header Ver.2, a PMOD USB-UART converter board is required.
RX72N Envision Kit	-	Dual Bank Mode	When using RSU header Ver.2, a PMOD USB-UART converter board is required.
RSK RX72T	J13 : 1-2 short	-	A PMOD USB-UART converter board is required.

The figures showing execution results show results for the RX65N. The terminal emulator used is Tera Term.

The operation of the demo projects has been confirmed using little-endian byte ordering. RX Firmware Update FIT V.1.05 is used with some modifications. Changes are indicated by the comment /* modified from original fwup code */.

The link size of the demo projects for TSIP-equipped devices is larger than 128 KB. You will need to purchase the commercial version of the CC-RX compiler in order to use them.

Connections between the PC and board are shown below.

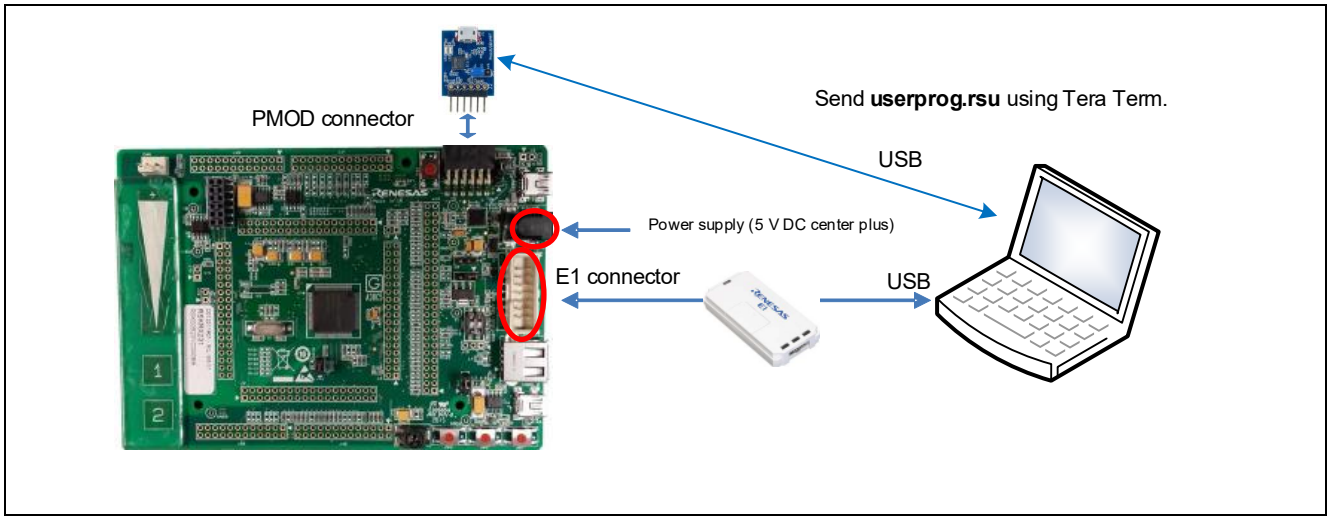


Figure 6.14 Connections between RSK RX231*2 and PC for Firmware Updates Using UART*1

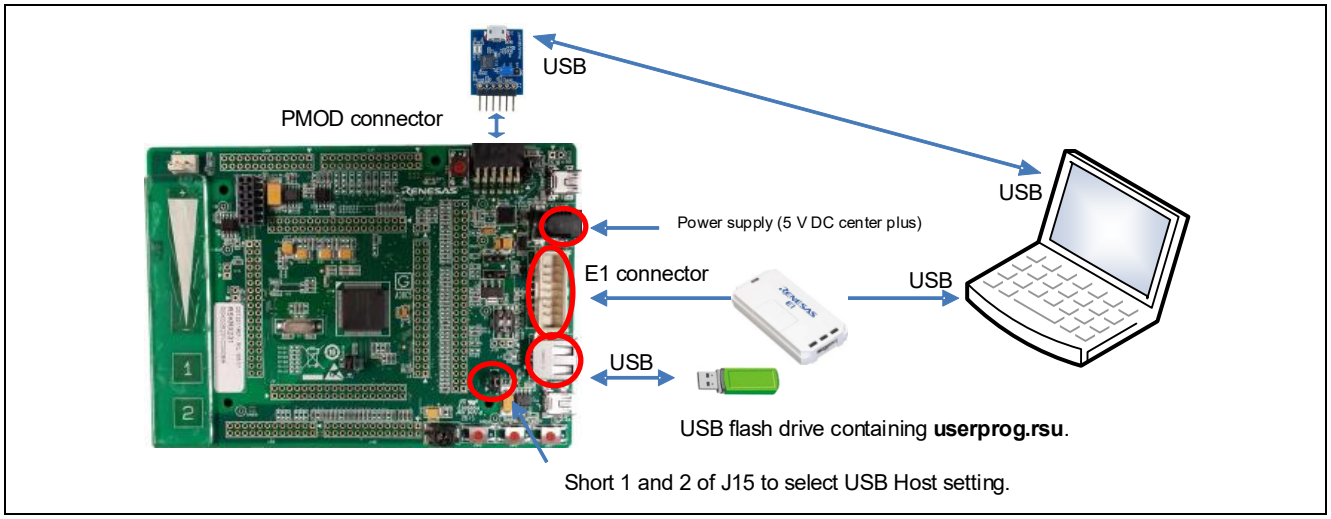


Figure 6.15 Connections between RSK RX231 and PC for Firmware Updates Using USB

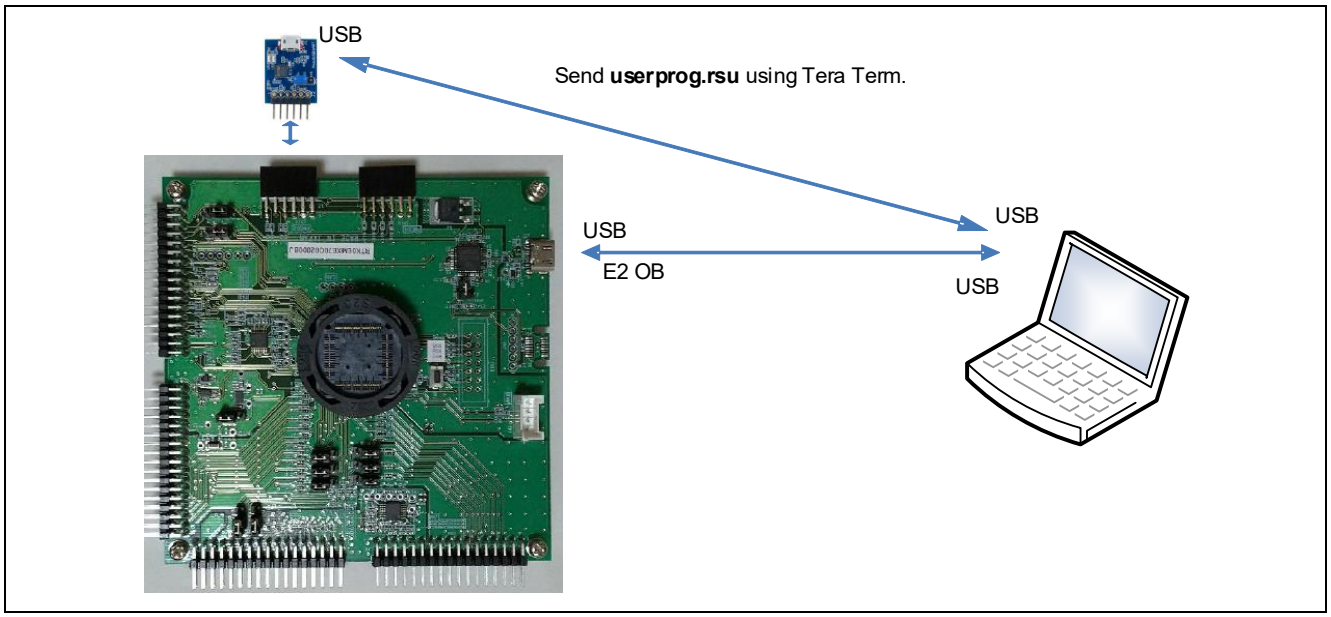


Figure 6.16 Connections between MCB RX26T and PC for Firmware Updates Using UART

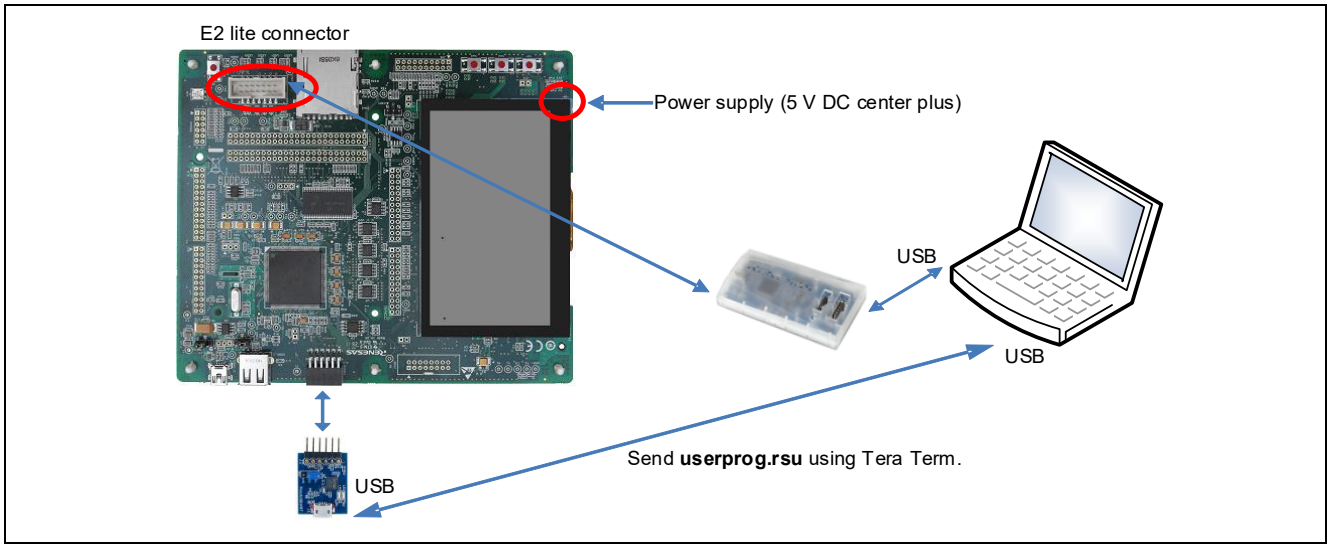


Figure 6.17 Connections between RSK RX65N-2MB and PC for Firmware Updates Using UART

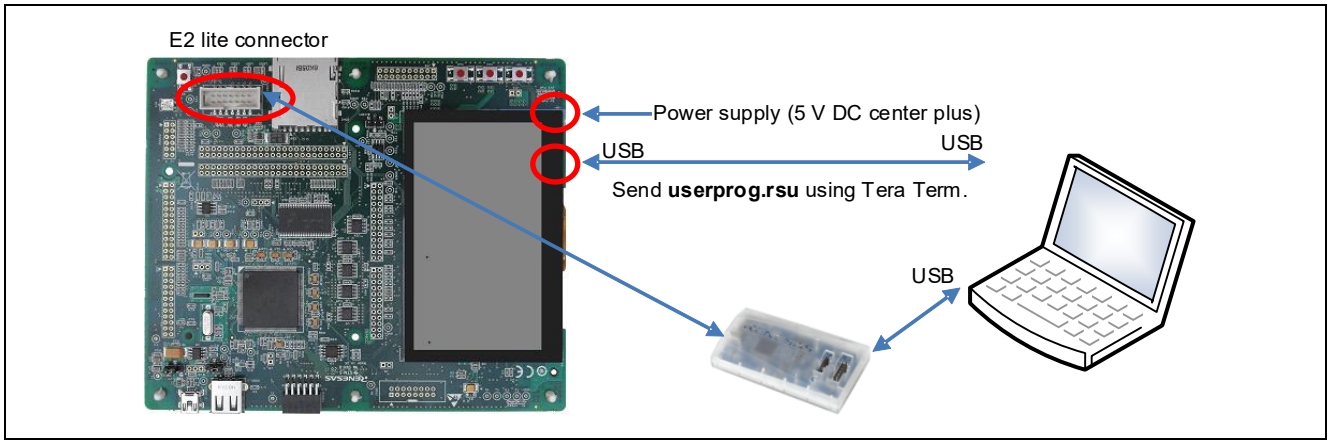


Figure 6.18 Connections between RSK RX65N-2MB and PC for Firmware Updates Using UART (1 area update)

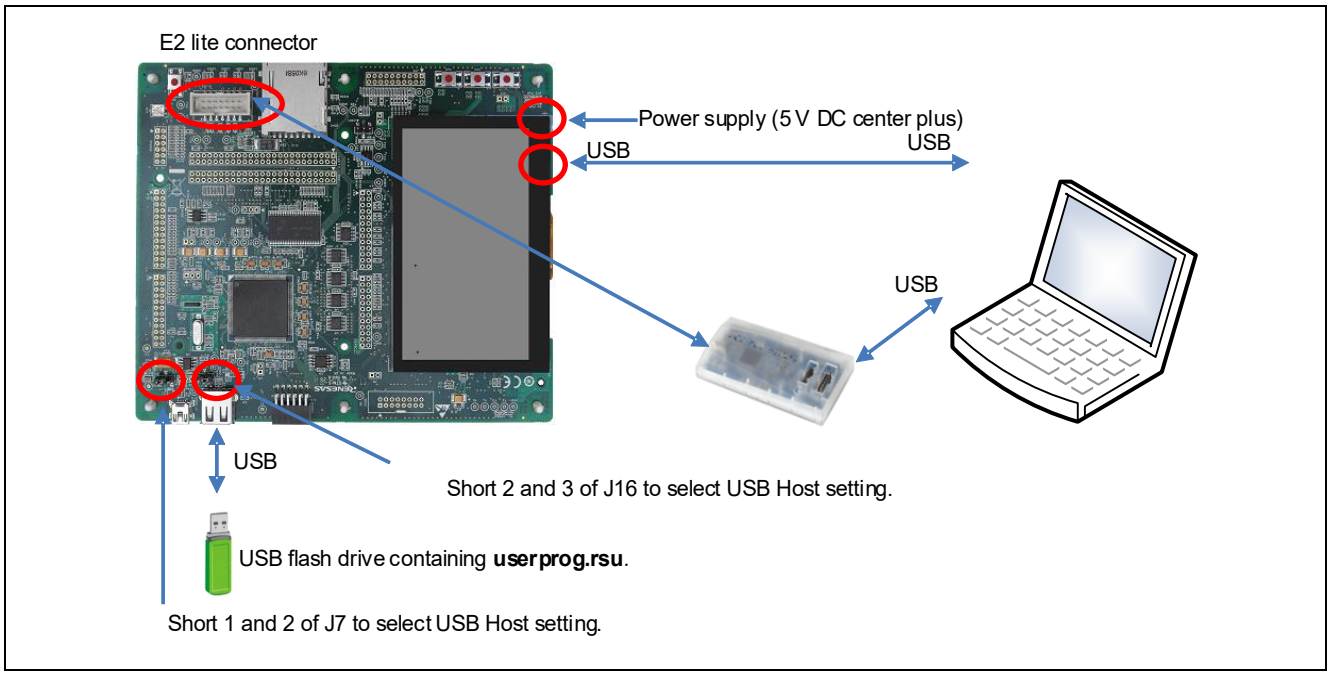


Figure 6.19 Connections between RSK RX65N-2MB and PC for Firmware Updates Using USB

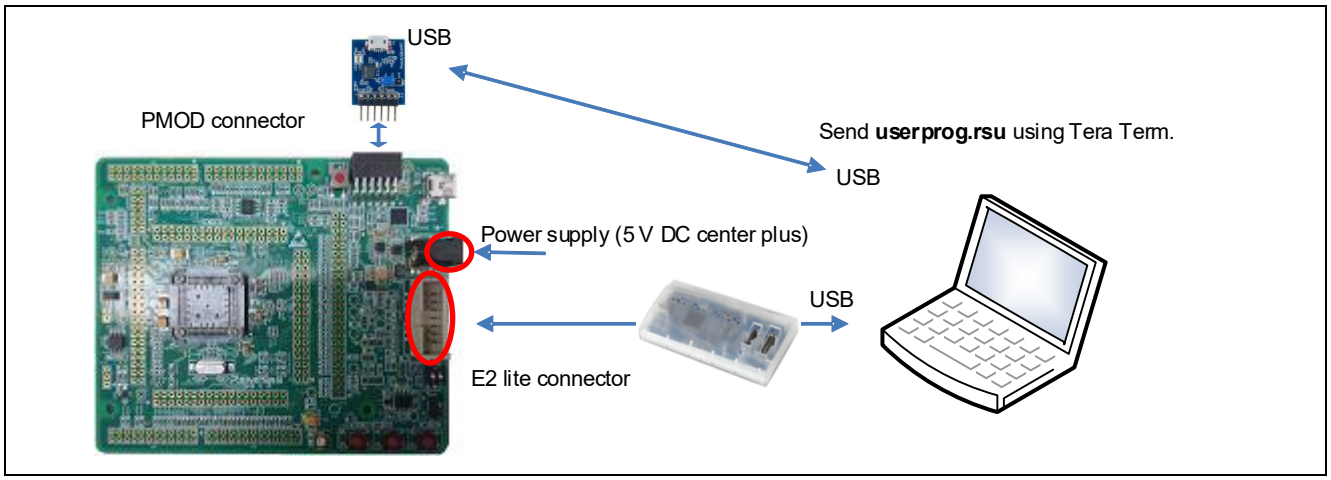


Figure 6.20 Connections between RSK RX66T and PC for Firmware Updates Using UART*1

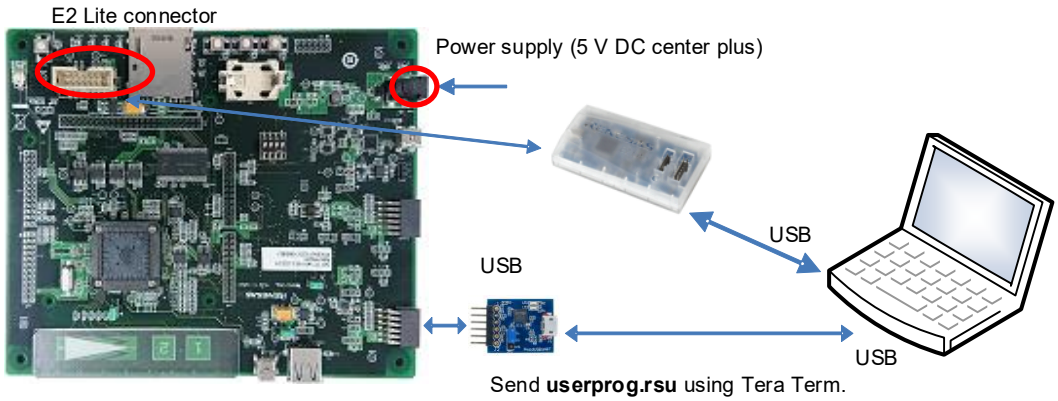


Figure 6.21 Connections between RSK RX671 and PC for Firmware Updates Using UART

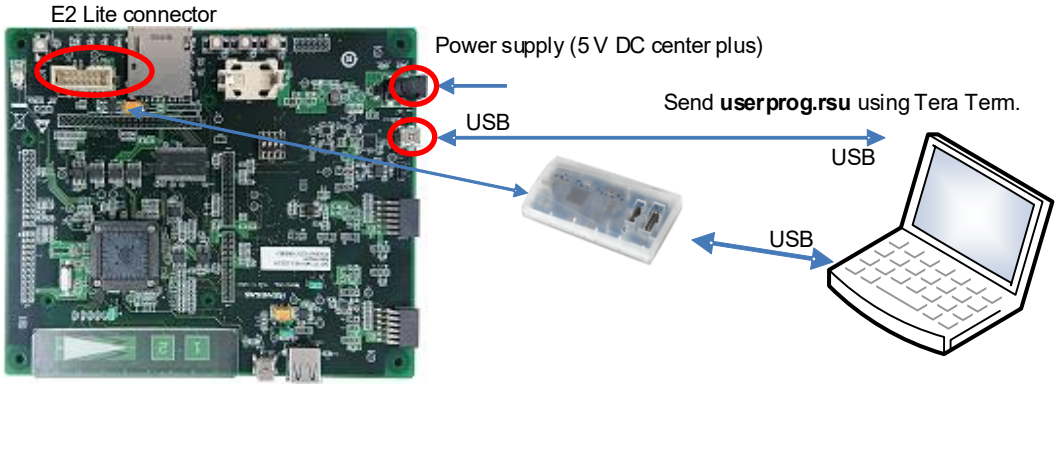


Figure 6.22 Connections between RSK RX671 and PC for Firmware Updates Using UART (1 area update)

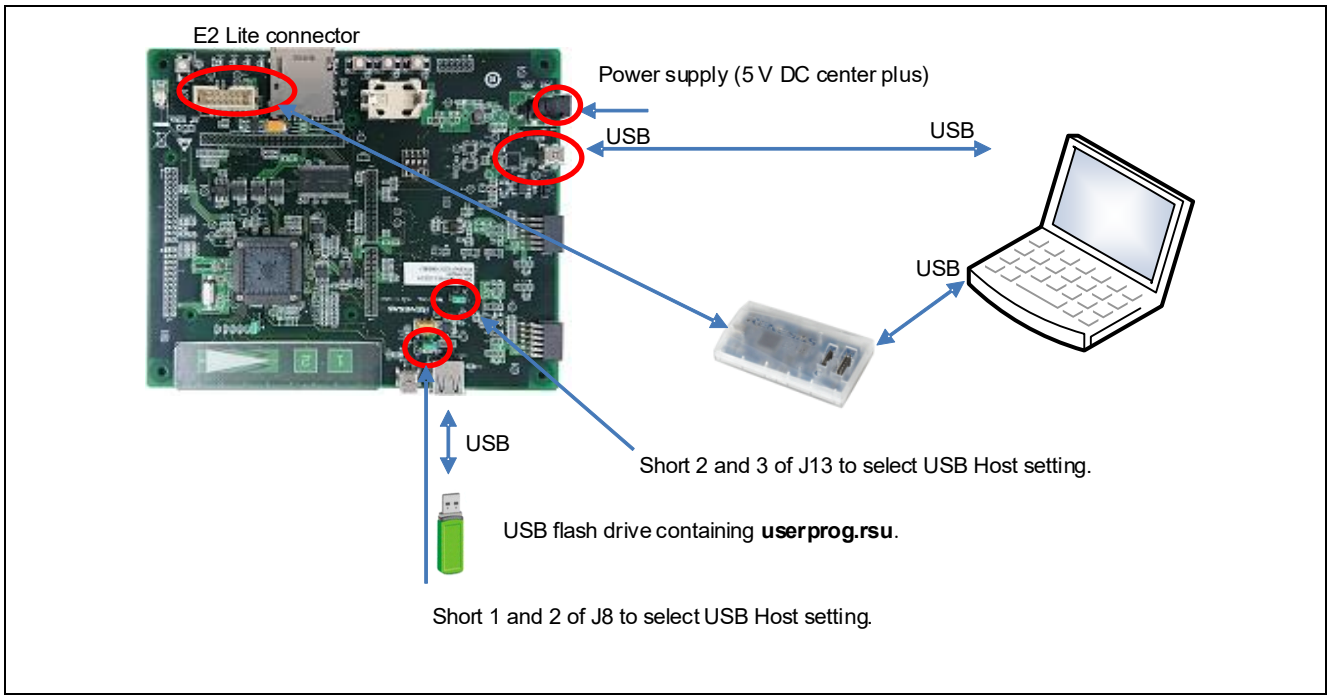


Figure 6.23 Connections between RSK RX671 and PC for Firmware Updates Using USB

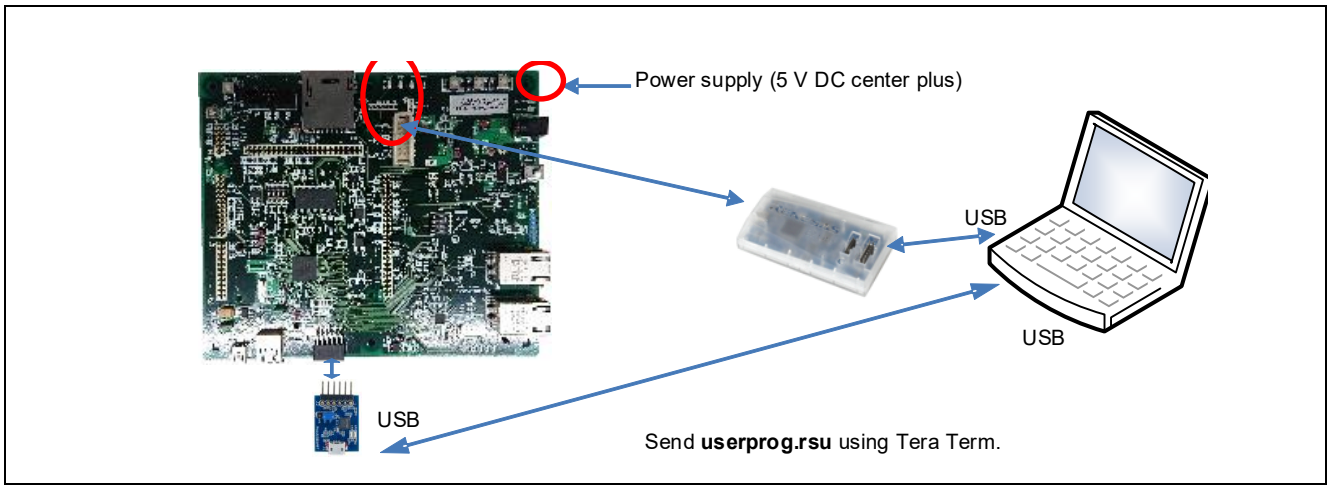


Figure 6.24 Connections between RSK RX72M and PC for Firmware Updates Using UART

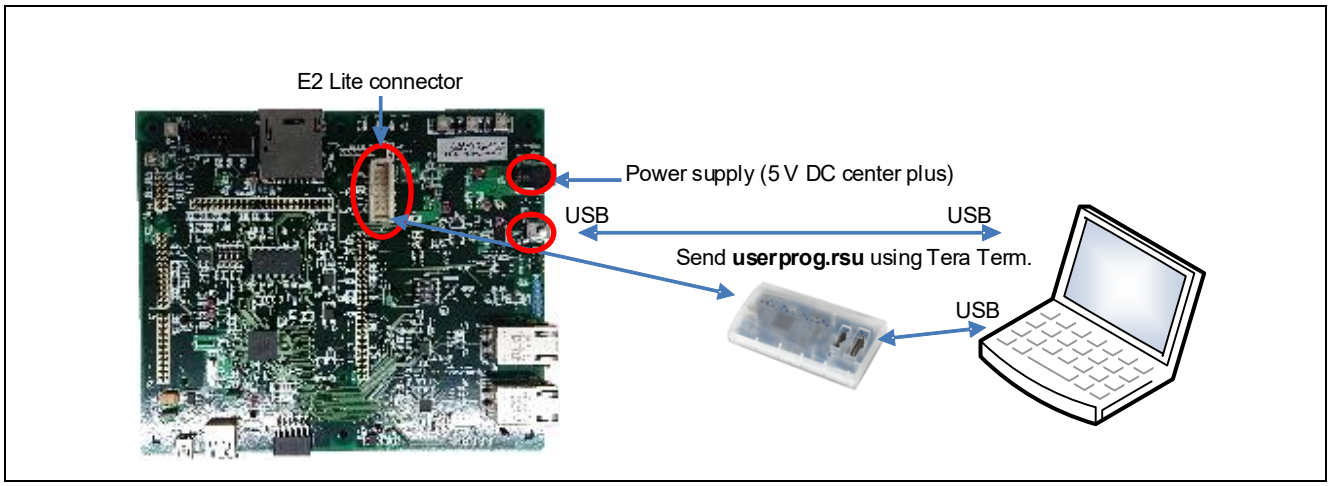


Figure 6.25 Connections between RSK RX72M and PC for Firmware Updates Using UART (1 area update)

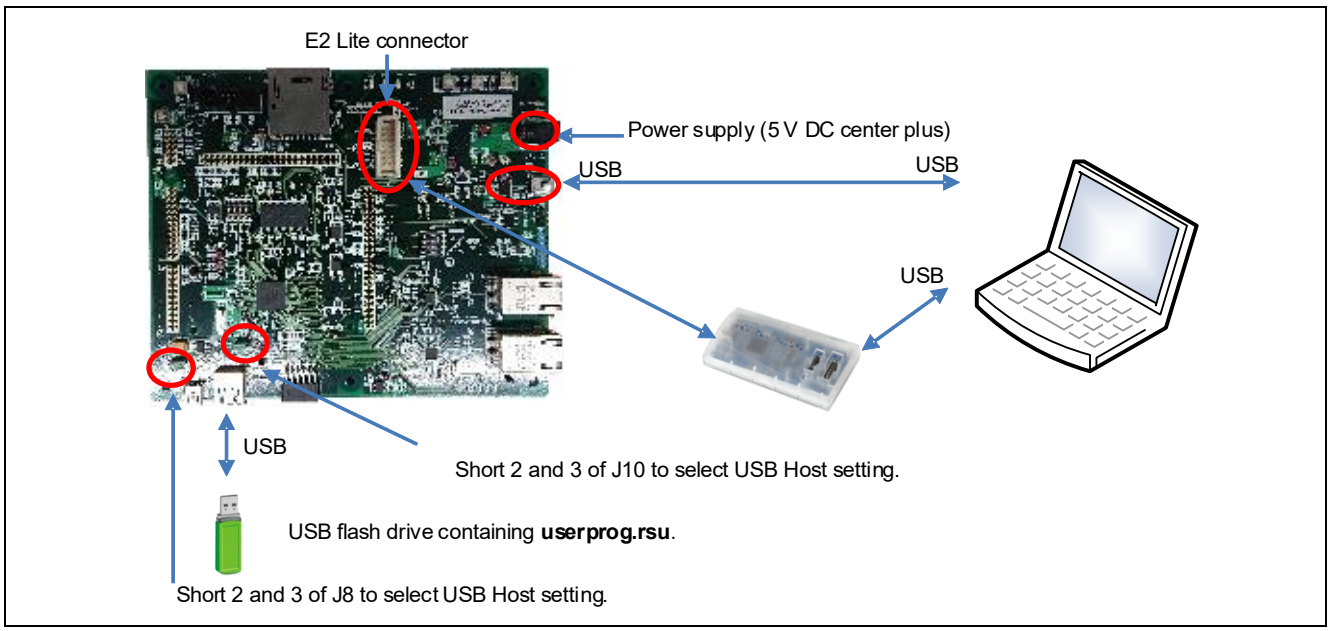


Figure 6.26 Connections between RSK RX72M and PC for Firmware Updates Using USB

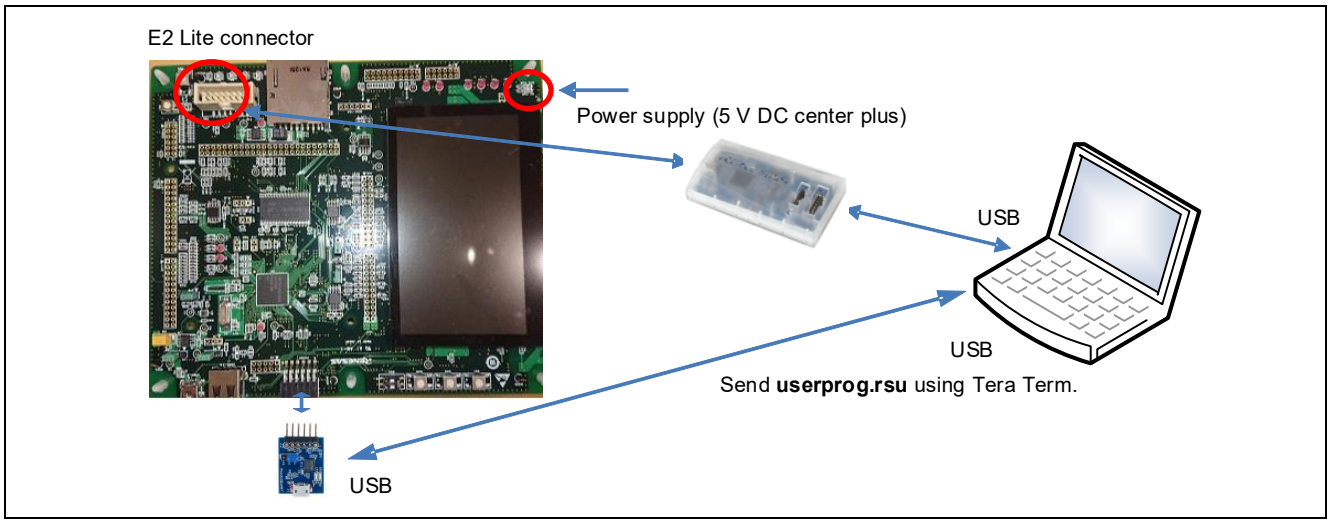


Figure 6.27 Connections between RSK RX72N and PC for Firmware Updates Using UART

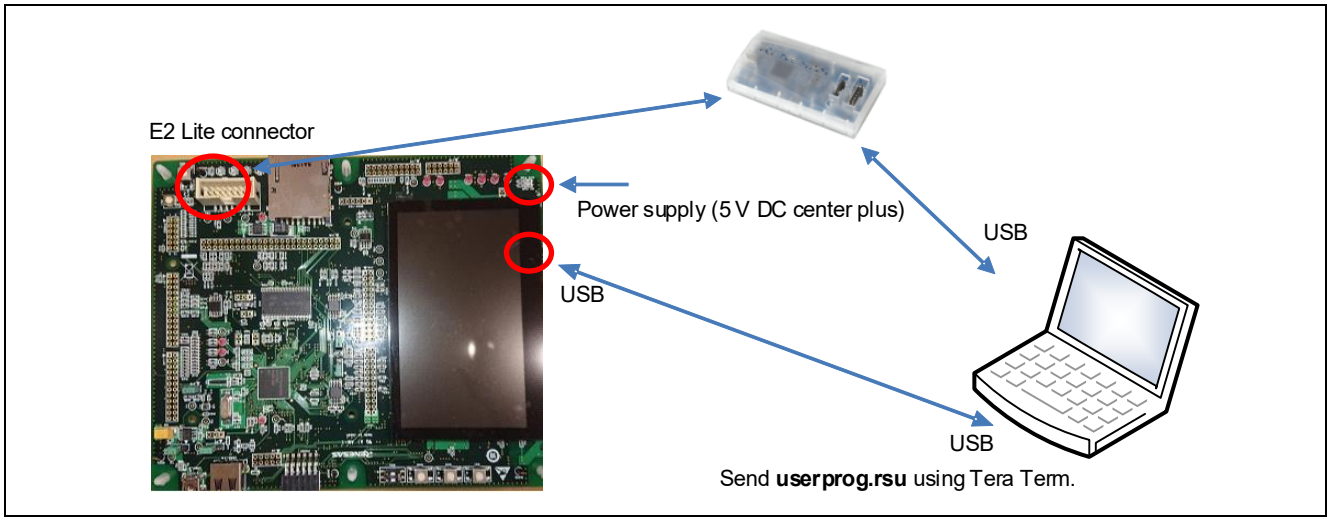


Figure 6.28 Connections between RSK RX72N and PC for Firmware Updates Using UART (1 area update)

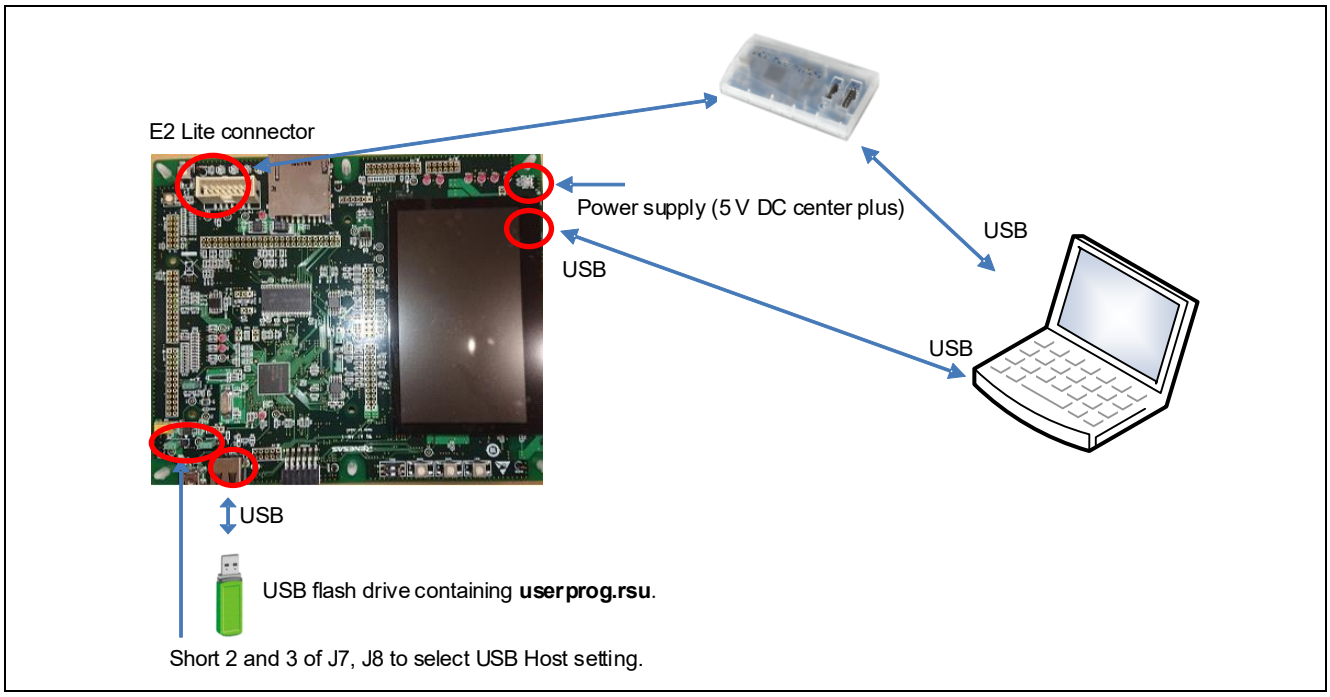


Figure 6.29 Connections between RSK RX72N and PC for Firmware Updates Using USB

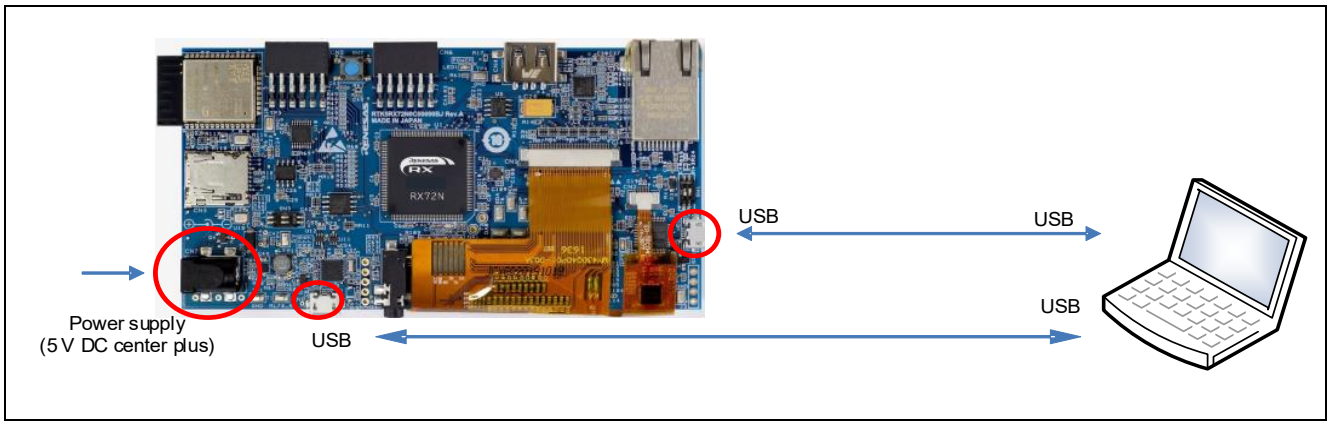


Figure 6.30 Connections between RX72N Envision Kit and PC for Firmware Updates Using UART

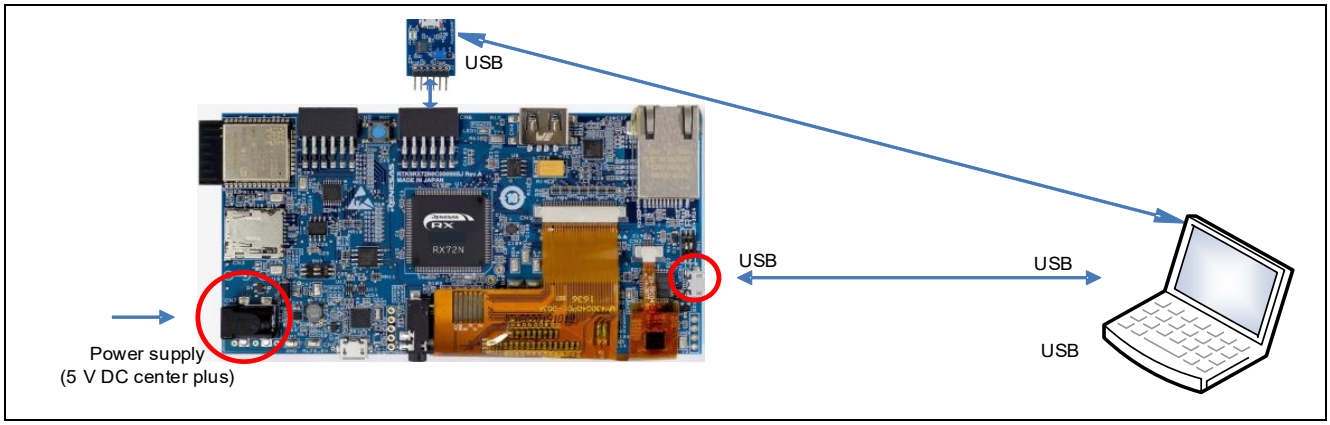


Figure 6.31 Connections between RX72N Envision Kit and PC for Firmware Updates Using UART (1 area update)

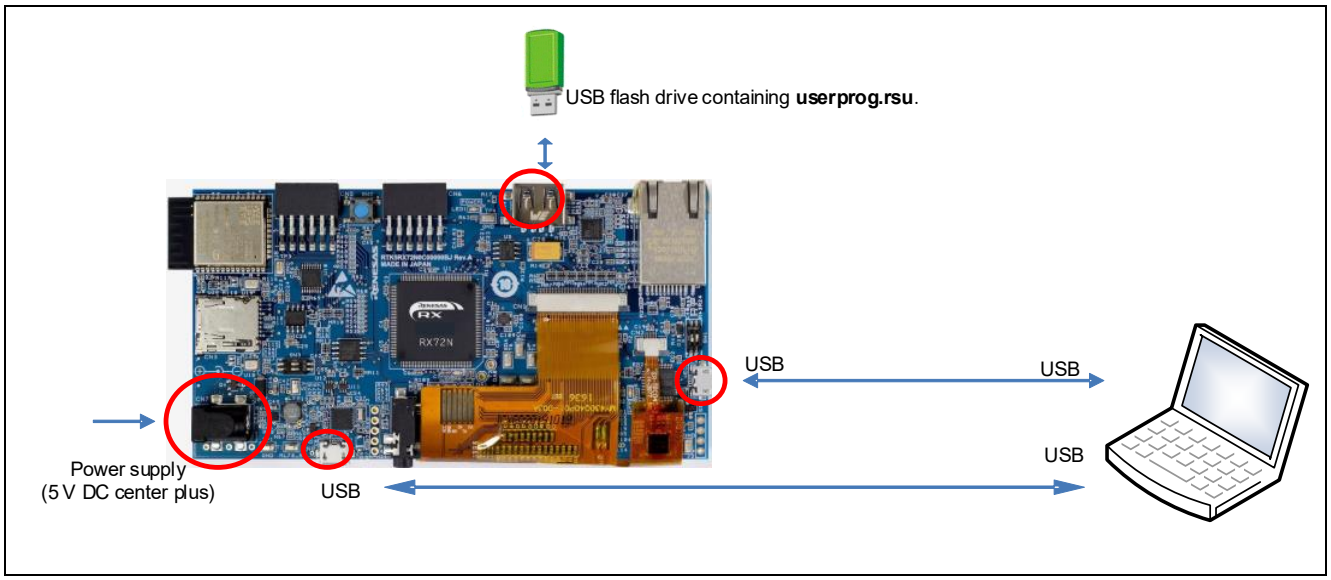


Figure 6.32 Connections between RX72N Envision Kit and PC for Firmware Updates Using USB

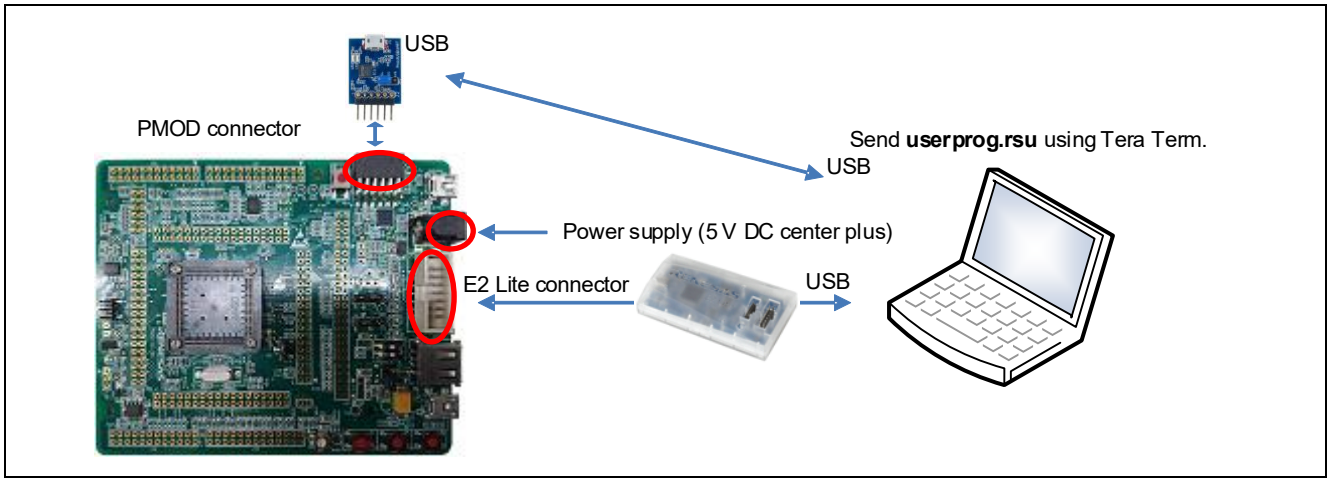


Figure 6.33 Connections between RSK RX72T and PC for Firmware Updates Using UART*1

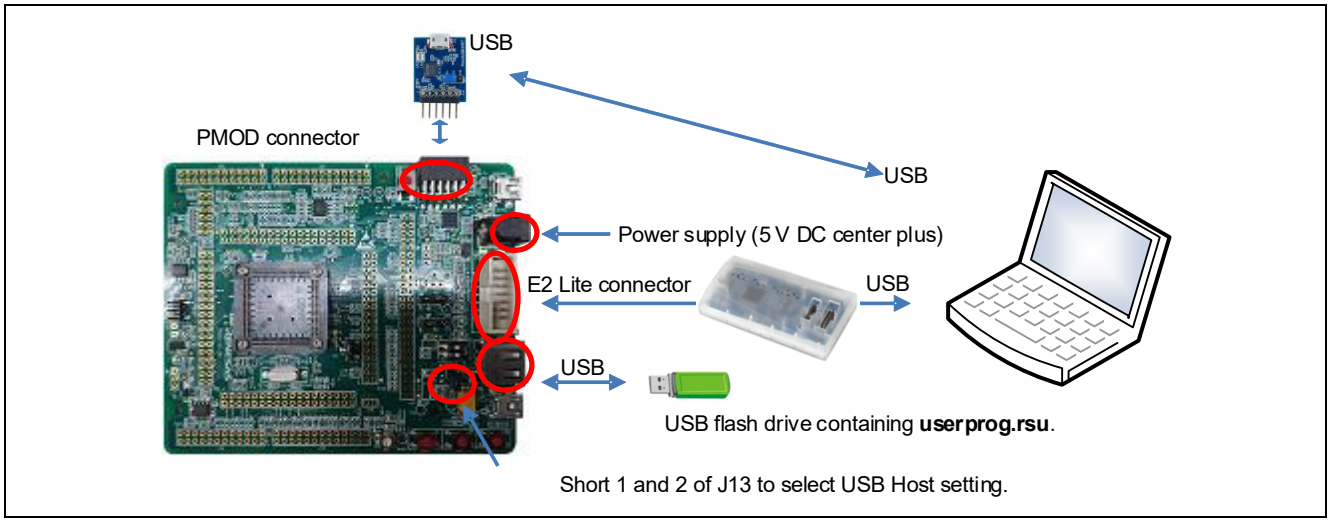


Figure 6.34 Connections between RSK RX72T and PC for Firmware Updates Using USB

Notes: 1. The operation of the demo projects has been confirmed using a Pmod USBUART from Digilent. The connection to the PMOD connector on the RX RSK board must be made as shown below.

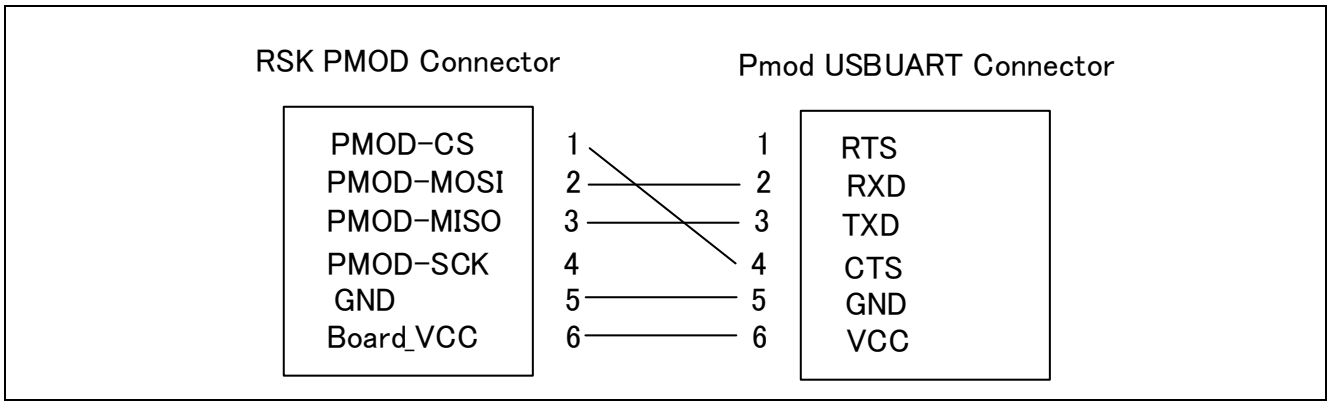


Figure 6.35 Connection diagram of RX RSK and Pmod USBUART

2. To perform flow control of the Pmod USBUART board connected to the PMOD1 connector on the RSK RX231 board, it is necessary to replace the previously mounted 0 Ω resistors with a pattern with no devices mounted.

Mounted		Not Mounted
R125	→	R126
R128	→	R127
R135	→	R134

Tera Term serial settings are as follows.

Table 6-7 Tera Term serial settings

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

Table 6-8 Tera Term settings (1 area update)

Item	Value
Speed	57600bps
Data	8bit
Parity	none
Stop bit	1bit
New-line	CR
Local echo	OFF
Flow control	none

6.2.2 Overview of Secure Bootloader and Firmware Update Projects

This section provides an overview of the secure bootloader project and firmware update project. The directory structure is described in Table 6-9.

Table 6-9 Directory structure of secure bootloader project and firmware update project

Directory Name	Description
FITDemos	Demo project folder
rx*XXX*_*bbb*_tsip_secure_update *1 *2	Secure boot/secure update implementation example
rx*XXX*_*bbb*_tsip_secure_boot *1 *2	Secure boot firmware
key	UFPK and W-UFPK file which can be used to execute example
skmt	Settign file of Security Key Management Tool
src	Source code of the secure boot firmware
rx*XXX*_*bbb*_tsip_user_program *1 *2	User program following secure update
src	Source code of the user program

Note: 1. "rx*XXX*" represents the RX group names.
2. "*bbb*" is the name of the supported board. Usually, USB and UART are available as IF for Update, but if the MCU does not support USB, only UART is supported. The director name of project which support only UART includes _sci before _secure_boot and _user_program.
RSK : rsk (rsk(_tsip)_sci : only support UART)
MCB : mcb (mcb(_tsip)_sci : only support UART)
Envision Kit : ek

The RX TSIP FIT secure bootloader and firmware update projects are sample programs that divide the flash ROM area into two sections, which are used as the main area and buffer area.

Main area: Storage area for user program to be executed

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

On devices with flash ROM that supports the dual bank function, this functionality is utilized in the firmware update operation.

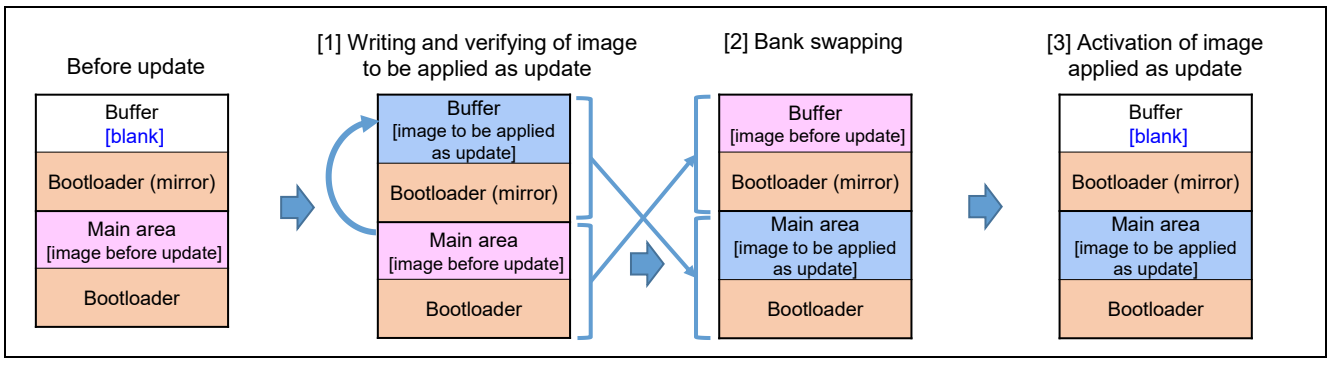


Figure 6.36 Firmware Update Operation Utilizing Dual Bank Function

On devices with flash ROM that does not support the dual bank function, the flash ROM area is divided into two sections, a main area and buffer area, and a partial updating method is used to apply the firmware update.

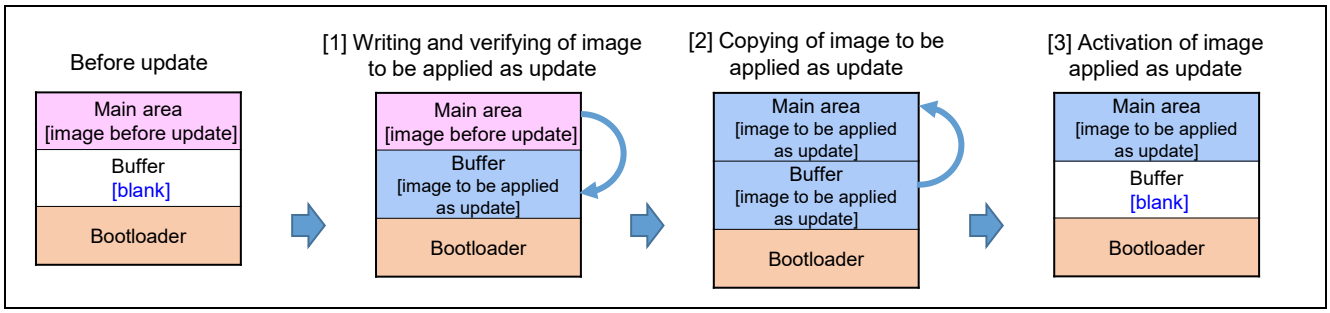


Figure 6.37 Firmware Update Operation Utilizing Partial Updating Method

The key_block_data located in the data flash is used to manage the MAC values used for status management and validity verification as well as the keys used for decryption. As data management for key injection or countermeasures in case of power interruption while an update is in progress, two copies of the same data are maintained in a main area and mirror area in the data flash.

Table 6-10 key_block_data Structure

Name	Type	Description
st_key_block_data_t	—	Key data structure located in C_FIRMWARE_UPDATE_CONTROL_BLOCK and C_FIRMWARE_UPDATE_CONTROL_BLOCK_MIRROR
firmware_update_control_data	—	Structure for storing MAC information during firmware update
user_program_max_cnt	uint32_t	Byte size of firmware update
state	uint32_t	See **Table 6-11**
program_mac0[]	uint32_t	MAC value of buffer area
program_mac1[]	uint32_t	MAC value of main area
key_data	—	Key data storage structure
user_aes128_key_index	tsip_aes_key_index_t	Key index used to decrypt encrypted user program
hash_sha1[]	uint8_t	SHA1 hash value of firmware_update_control_data and key_data

Table 6-11 **Description of state**

Status	Description
STATE_INJECT_KEY	Inject Key Encryption Key
STATE_GET_IMAGE	Receive and decrypt user program Write user program to code flash
STATE_ACTIVATE_IMAGE	Activate user program
STATE_EXEC_IMAGE	Verify and execute user program

6.2.2.1 Secure Bootloader project

The secure bootloader project performs key injection used to decrypt the user program and verifies the user program. It also contains program code to install the initial user program (the firmware update program in the case of this sample project).

A flowchart of the secure bootloader project is shown below.

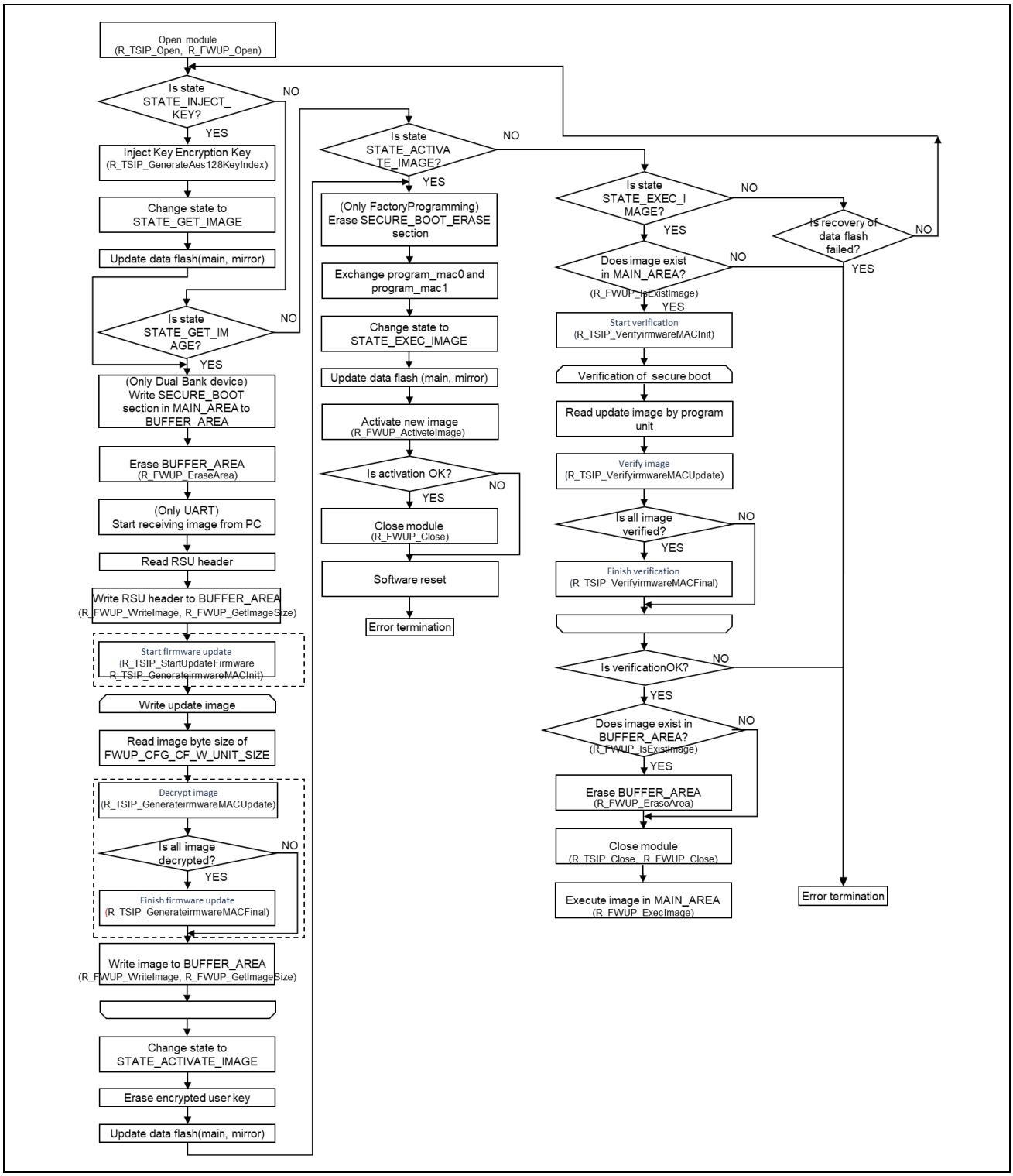


Figure 6.38 The flow of Secure Bootloader

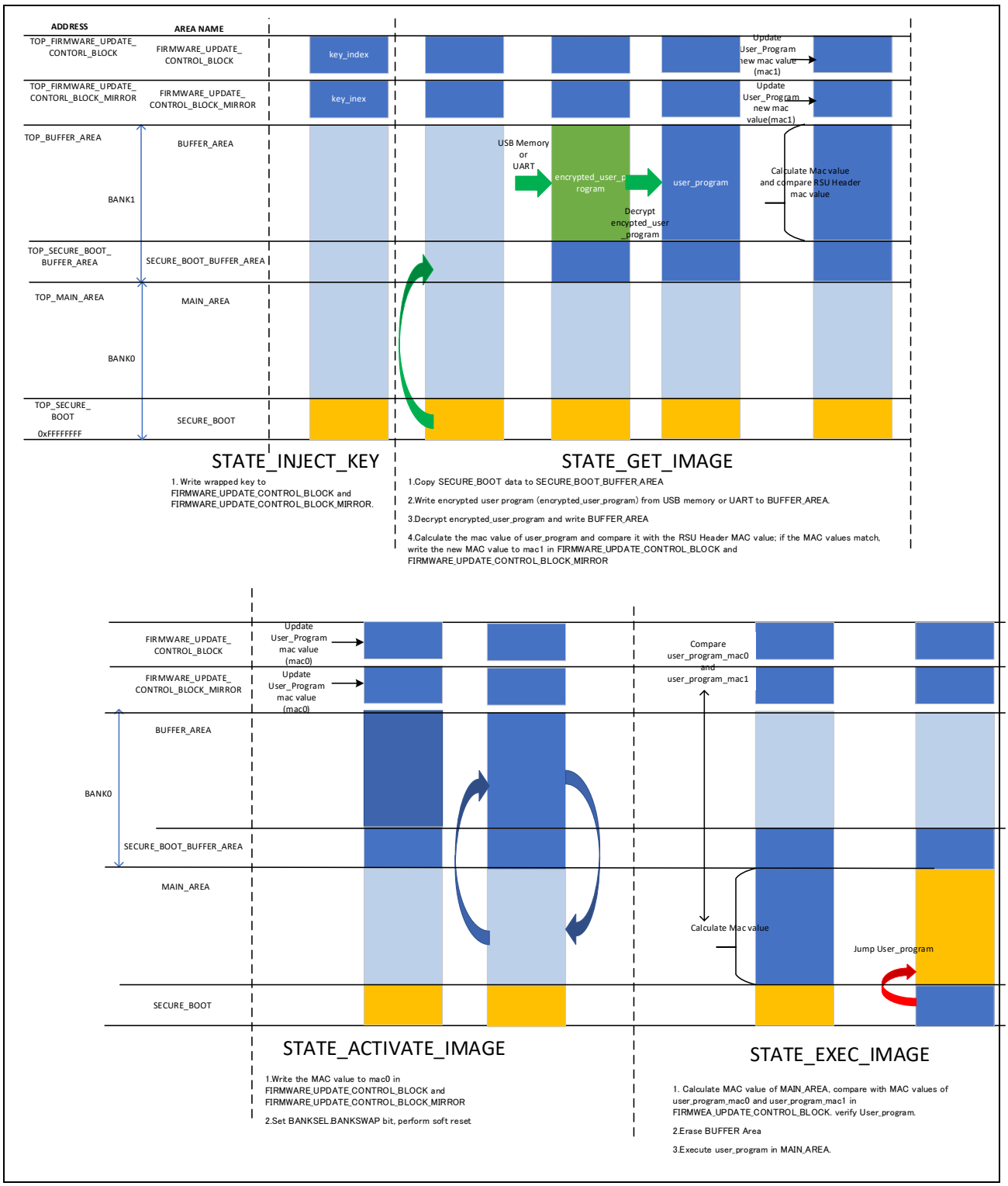


Figure 6.39 Secure Bootloader On-Chip Flash Memory Status in Each state (Device with Dual Bank Support)

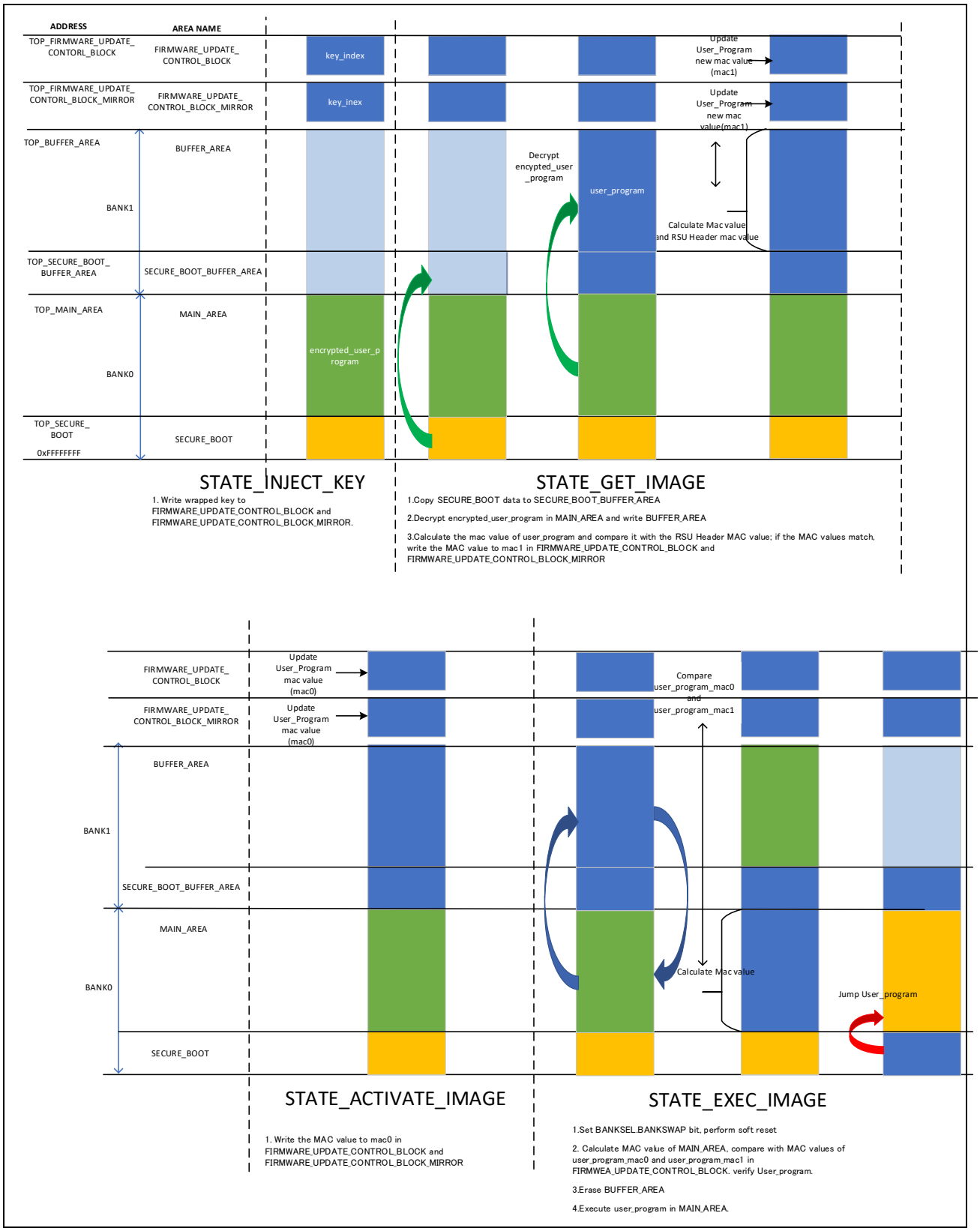


Figure 6.40 Secure Bootloader On-Chip Flash Memory Status in Each state (Factory Programming of Device with Dual Bank Support)

Table 6-12 Addresses of Areas Set in Secure Bootloader Project (Device with Dual Bank Support)

	RX26T	RX65N	RX671	RX72M,RX72N
TOP_FIRMWARE_UPDATE_ CONTORL_BLOCK	0x00100000	0x00100000	0x00100000	0x00100000
TOP_FIRMWARE_UPDATE_ CONTORL_BLOCK_MIRROR	0x00102000	0x00104000	0x00101000	0x00104000
TOP_BUFFER_AREA	0xFFF80000	0xFFE00000	0xFFE00000	0xFFC00000
TOP_SECURE_BOOT_ BUFFER_AREA	0xFFFB0000	0xFFEF0000	0xFFEF0000	0xFFDF0000
TOP_MAIN_AREA	0xFFFC0000	0xFFF00000	0xFFF00000	0xFFE00000
TOP_SECURE_ BOOT_AREA	0xFFFF0000	0xFFFF0000	0xFFFF0000	0xFFFF0000

The secure bootloader project places the key injection APIs and the modules for USB operation in the SECURE_BOOT_ERASE section to delete after using. For this reason, these source codes are built to library files which are allocated to the specified section before linking other part of the project.

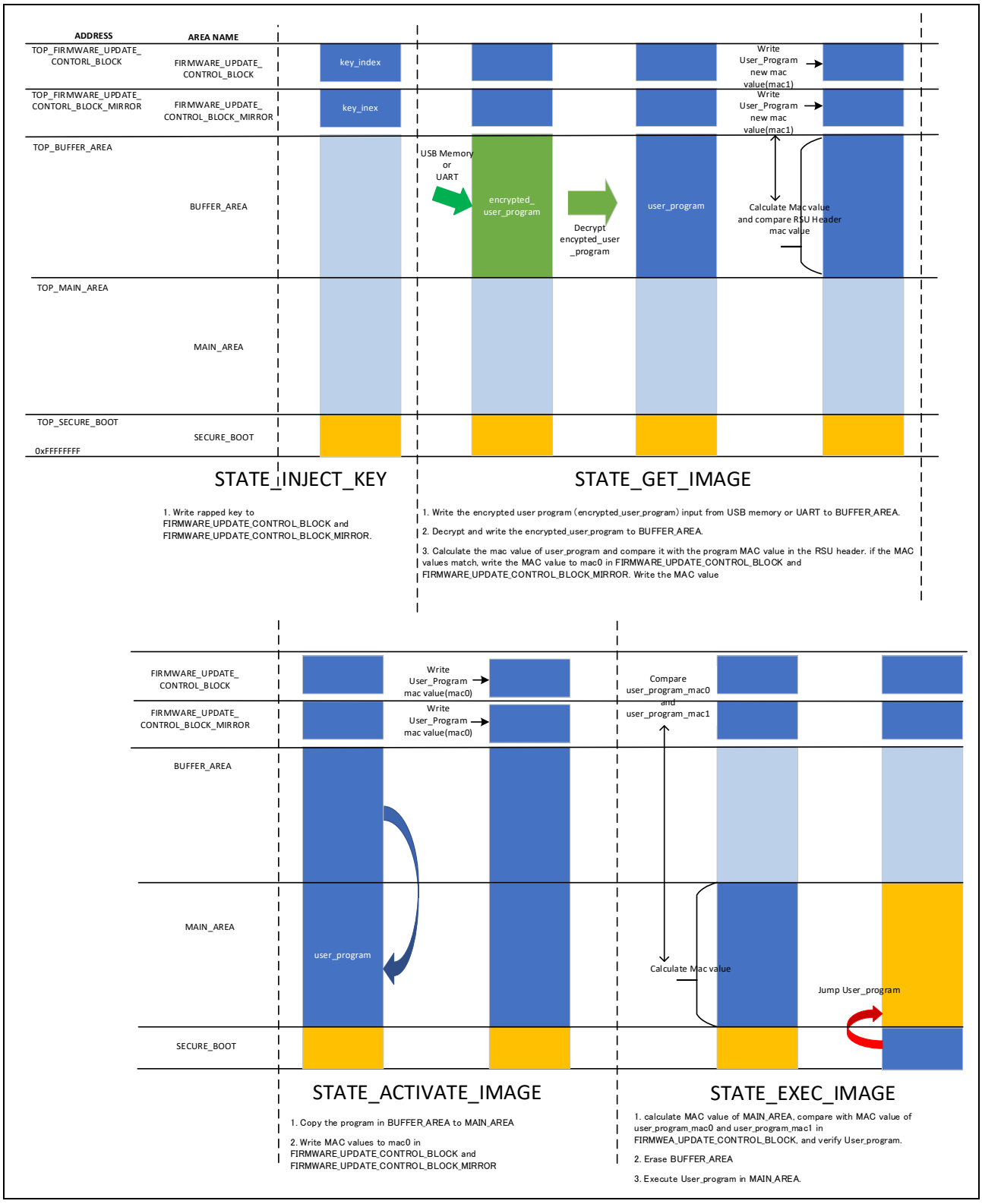


Figure 6.41 Secure bootloader Internal Flash memory status for each state (Device without Dual Bank Support)

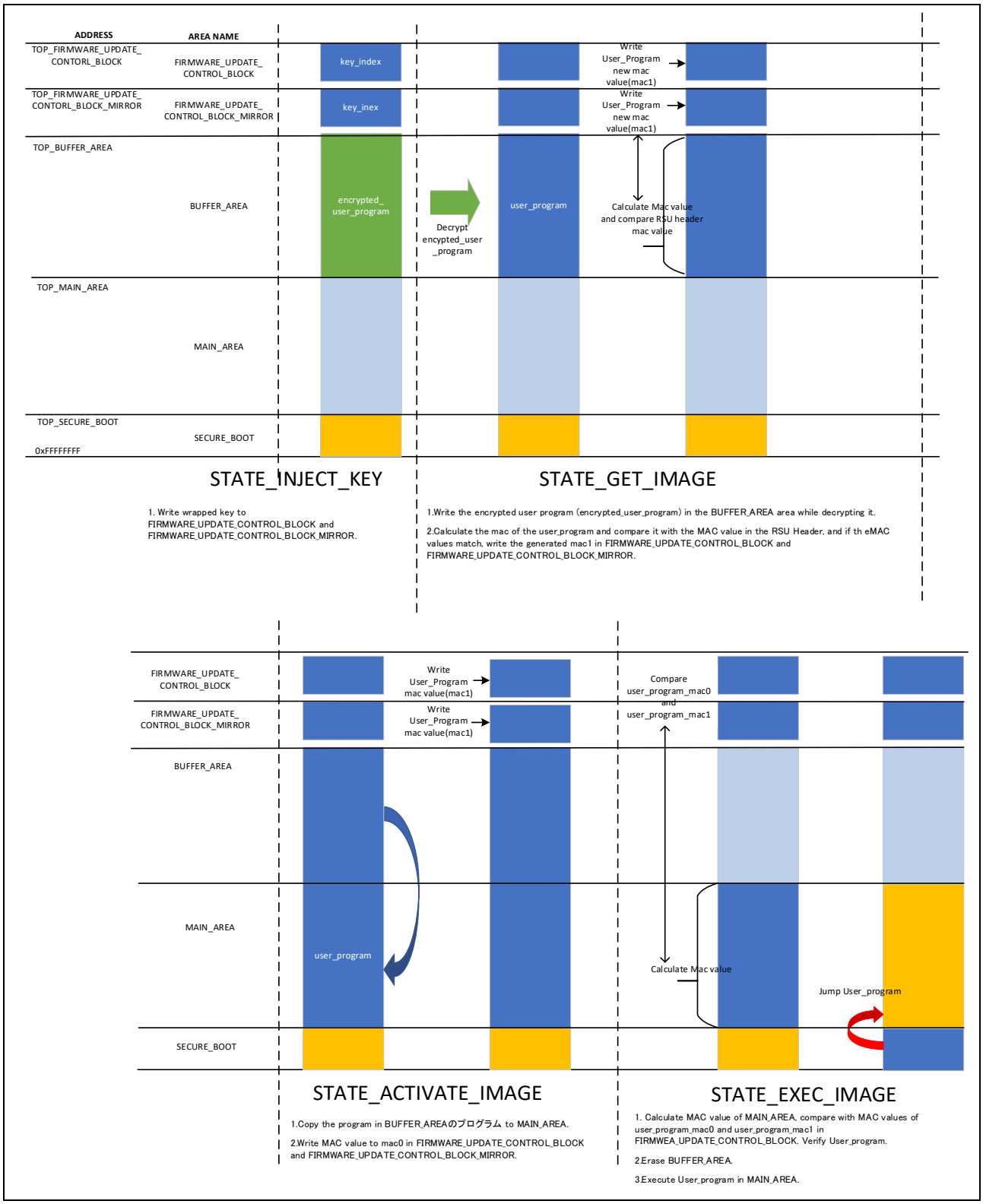


Figure 6.42 Secure bootloader Internal Flash memory status for each state (Factory Programming of Device with Dual Bank Support)

Table 6-13 Addresses of Areas Set in Secure Bootloader Project (Device without Dual Bank Support)

	RX231	RX66T	RX72T
TOP_FIRMWARE_UPDATE_ CONTORL_BLOCK	0x00100000	0x00100000	0x00100000
TOP_FIRMWARE_UPDATE_ CONTORL_BLOCK_MIRROR	0x00101000	0x00104000	0x00104000
TOP_BUFFER_AREA	0xFFF80000	0xFFF80000	0xFFF00000
TOP_MAIN_AREA	0xFFFB8000	0xFFFB8000	0xFFF78000
TOP_SECURE_BOOT_AREA	0xFFFF0000	0xFFFF0000	0xFFFF0000

The secure bootloader project places modules described in the SECURE_BOOT section. For this reason, changes have been made to the source code of each FIT module.

6.2.2.2 Firmware Update Project

The firmware update project decrypts and verifies the user program. A flowchart of the firmware update project after receiving the update command is shown below.

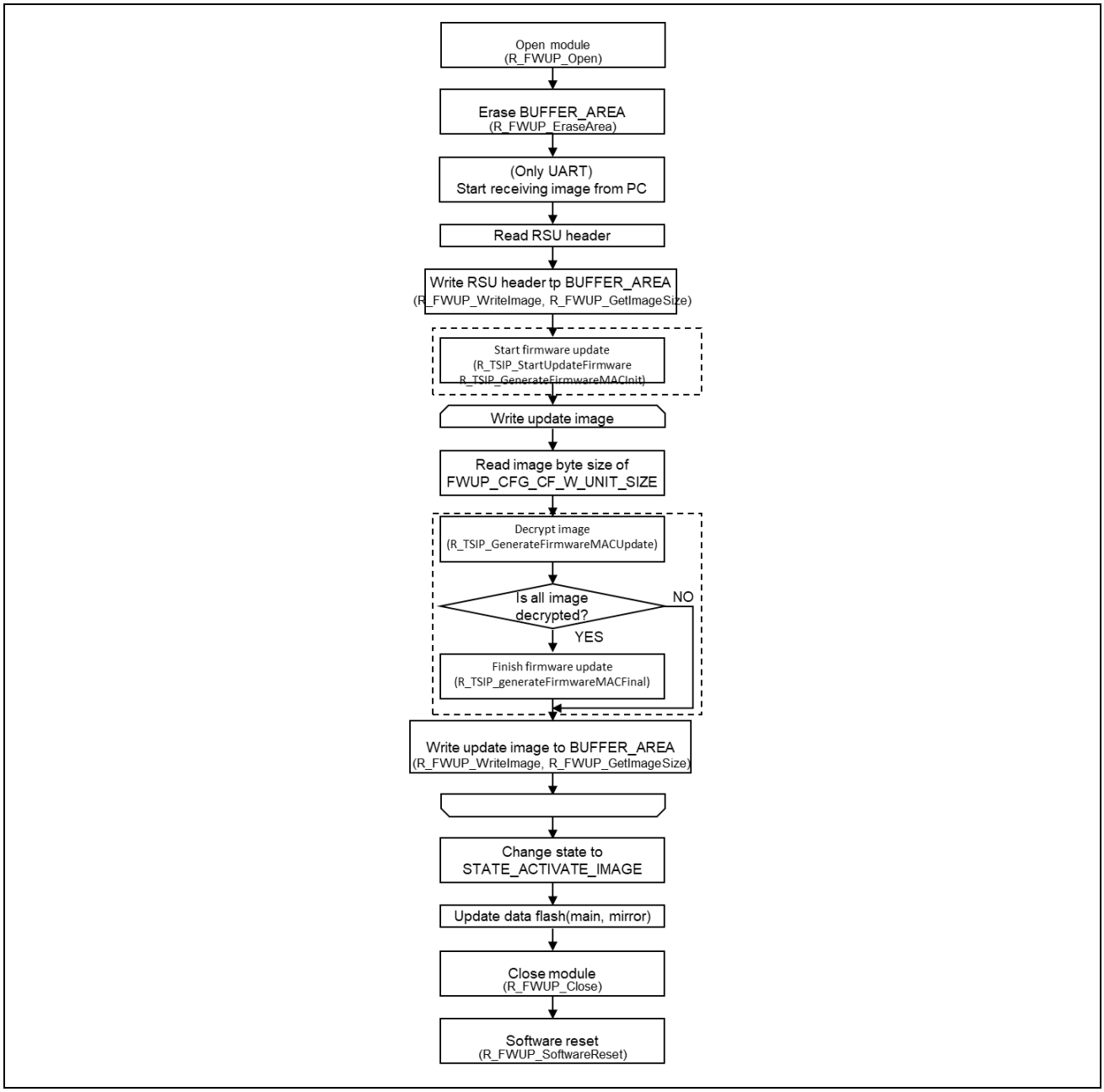


Figure 6.43 Firmware Update Flowchart

6.2.3 Secure Bootloader and Firmware Update Execution Example

Firmware update operation using UART is supported on all the board. In addition, firmware update operation using a USB flash drive is supported on boards populated with a device with USB HOST functionality and equipped with a USB host socket.

(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 user program.

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

UFPK = 22222222222222222222222222222222111111111111111111111111111111111

Key Encryption Key = 0123456789abcdef0123456789abcdef

- Command line version
  Run the following command.

  > skmt.exe /genkey /iv "55aa55aa55aa55aa55aa55aa55aa55aa" /ufpk file="{$skmt_loc}\..\key\sample.key"
  /wufpk file="{$skmt_loc}\key\sample.key_enc.key"
  /mcu "RX-TSIP" /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**
  Select **Use specified value** and enter **"55aa55aa55aa55aa55aa55aa55aa55aa"**.
  — **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 rx*XXX*_SecureUpdate.skmt included in the sample.

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

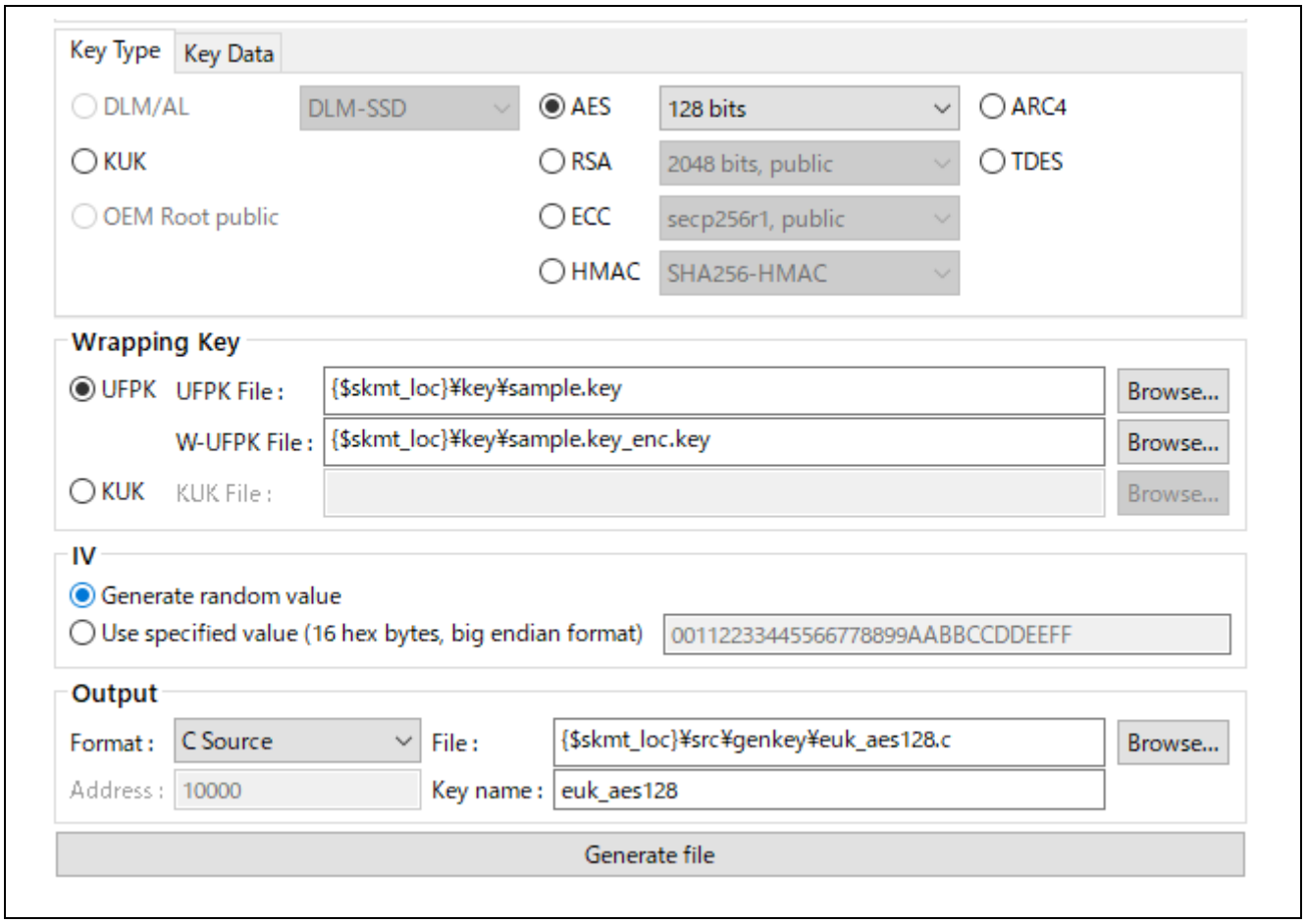


Figure 6.44 Generating a Key Data File (Wrap Key Tab)

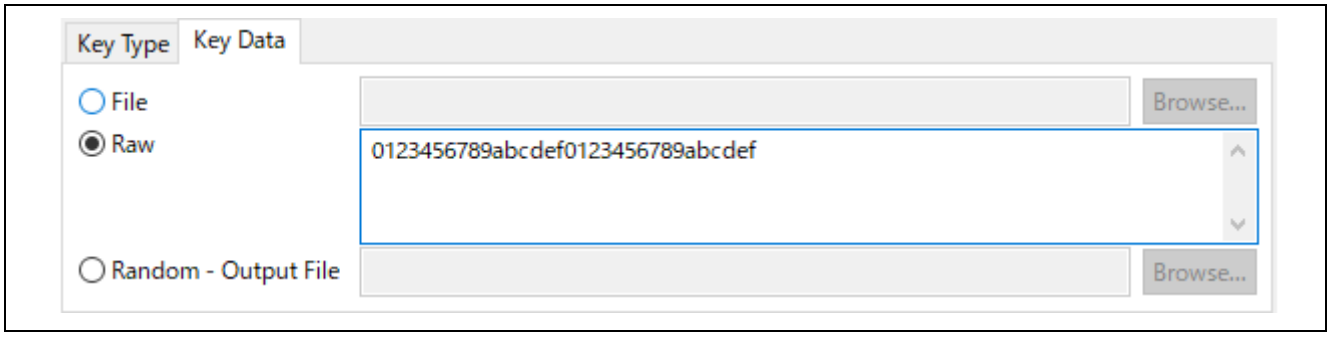


Figure 6.45 Generating a Key Data File (Wrap Key Tab – Key Data Tab)

(2) Building the Secure Bootloader Project

After importing the secure bootloader project (rx*XXX*_bbb_tsip_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 rxXXX_bbb_tsip_secure_boot project.

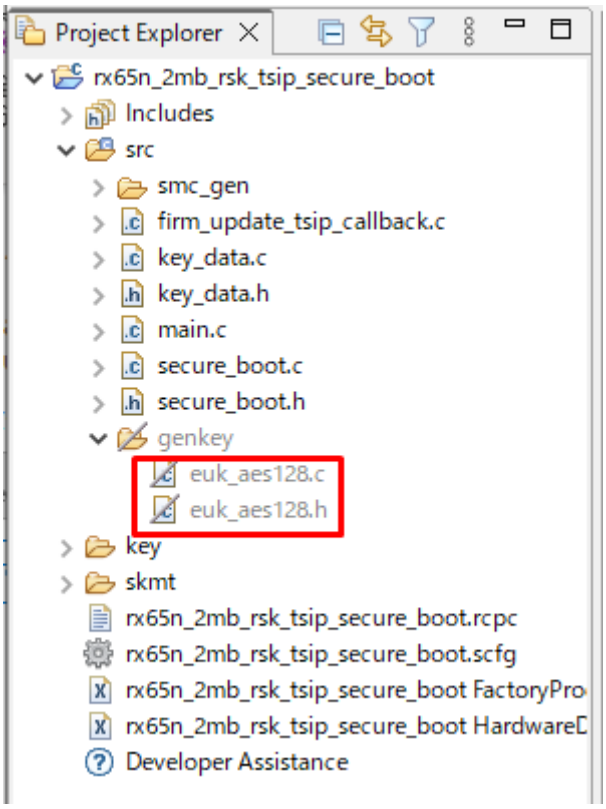


Figure 6.46 e² studio Project Explorer

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.

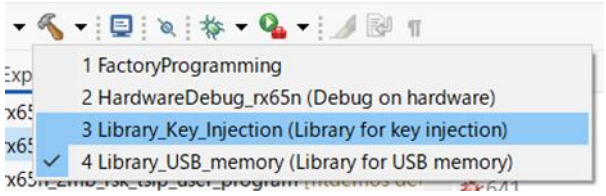


Figure 6.47 Building library files

Also, set the macro definitions to adjust the interface to send program to the device. These can be set from “C/C++ Build” > “Settings” > “Tool Settings” > “Compiler” > “Source” in the project properties menu.

To use USB flash drive with the supported board, set “ENABLE_USB=1”.

To use UART, set “ENABLE_USB=0”.

To use 1 area update function, set “UPDATE_1AREA=1” and set FWUP_CFG_FWUPV1_COMPATIBLE value in r_fwup_config.h to 1. In addition, change BSP_CFG_UART_TERMINAL_CHANNEL and BSP_CFG_SCI_UART_TERMINAL_BITRATE values in r_bsp_config.h to the serial port number

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

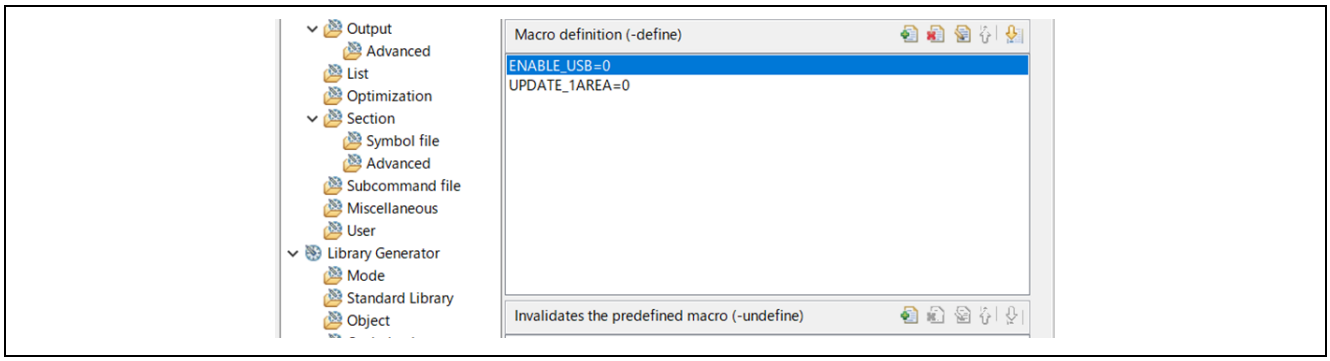


Figure 6.48 Setting macro definitions

(3) Building the Firmware Update Project

After importing the firmware update project (rx*XXX*_bbb_tsip_user_program) into the e² studio workspace, as in step (2), set Macro definitions. After making the changes, build the project in e² studio.

(4) Encrypting the Firmware Update Program

- Command line version
  When to use RSU header Ver.2, run the following command.

```
> skmt.exe /enctsip /mode "update"  /ver "2" /prg
    "{$skmt_loc}\..\rx65n_2mb_rsk_tsip_user_program\Release_rx65n\rx65n_2mb_rsk_tsip_user
    _program.mot" /enckey "0123456789ABCDEF0123456789ABCDEF" /session_key
  "FEDCBA9876543210FEDCBA98765432100123456789ABCDEF0123456789ABCDEF"
  /iv_fw "55AA55AA55AA55AA55AA55AA55AA55AA" /startaddr "FFF00300" /endaddr
  "FFFEFFFF" /filetype "bin" /flash_wsize 128 /output "{$skmt_loc}\userprog.rsu"
```

  When to use RSU header Ver.1, run the following command.

```
> skmt.exe /enctsip /mode "update" /ver "1" /prg
    "{$skmt_loc}\..\rx65n_2mb_rsk_tsip_user_program\Release_rx65n\rx65n_2mb_rsk_ts
    ip_user_program.mot"
    /enckey "0123456789abcdef0123456789abcdef"
    /startaddr "FFE00300" /endaddr "FFFEFFFF" /imgflg "testing" /filetype "bin"
    /flash_wsize 128 /output ""{$skmt_loc}\userprog.rsu"
```

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

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

Table 6-14 Selections for Each MCU

MCU	Parameter		Address
	GUI	CLI	
RX231 RX66T	Start address	/startaddr	FFFB8300
	End address	/endaddr	FFFEFFFF
RX72T	Start address	/startaddr	FFF78300
	End address	/endaddr	FFFEFFFF
RX26T	Start address	/startaddr	FFFC0300
	End address	/endaddr	FFFEFFFF
RX65N RX671	Start address	/startaddr	FFF00300
	End address	/endaddr	FFFEFFFF
RX72M RX72N	Start address	/startaddr	FFE00300
	End address	/endaddr	FFFEFFFF

- Standalone version
  Enter the following values.
  — **Output Image**
  Select **Secure Update**.
  — **Firmware Image**
  MOT file output by update project (rx*XXX*_*bbb*_tsip_user_program.mot)
  — **Encrypted Address Range** tab
  The values differ depending on the MCU. Refer to Table 6.13 and select the appropriate values to match the MCU.
  — **IV** tab
  Select **Use specified value** and enter **“55aa55aa55aa55aa55aa55aa55aa55aa”**.
  — **RSU Header** tab
  When 1 area update is not used, set below setting
  RSU header Ver: 2
  When 1 area update is used, set below setting
  Image Flag: TESTING
  RSU header Ver: 1
  — **Output**
  Format: Binary
  File: userprog.rsu

To use the above settings, you can load the Security Key Management Tool settings file rx*XXX*_SecureUpdate.skmt included in the sample.

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

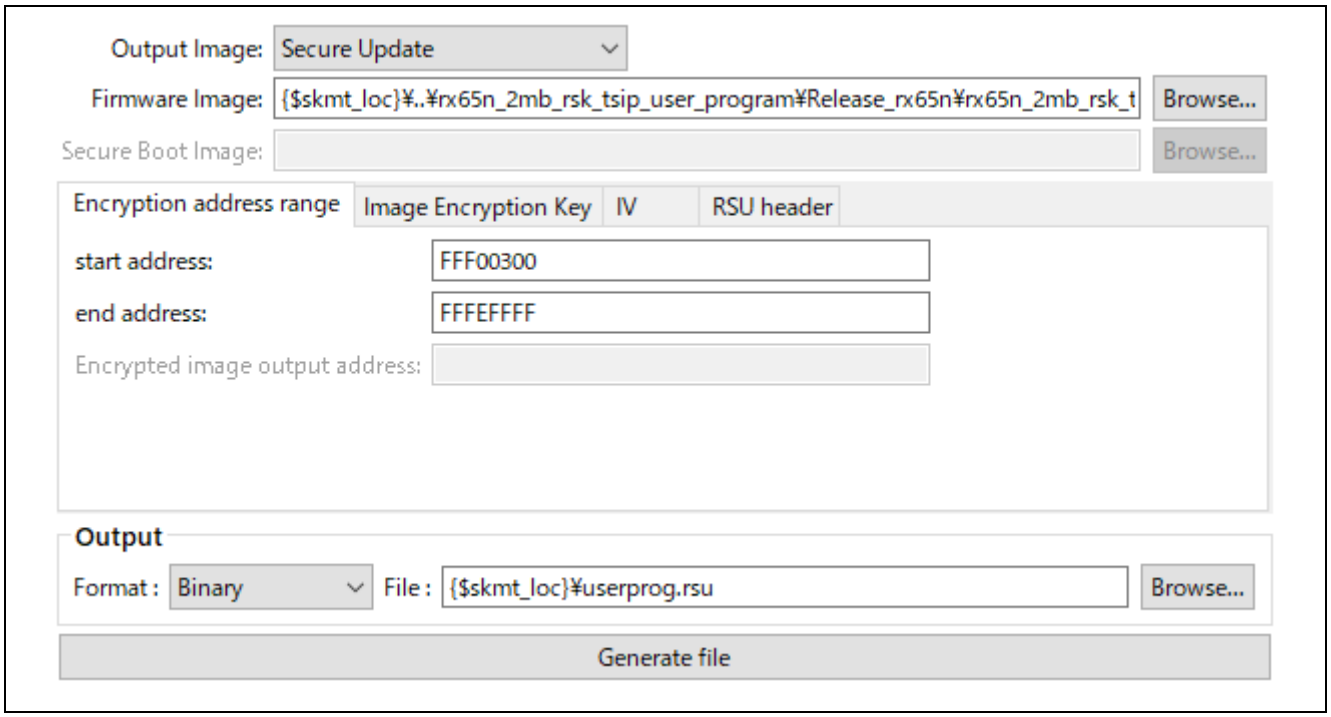


Figure 6.49 Generating an Encrypted File (TSIP UPDATE Tab)

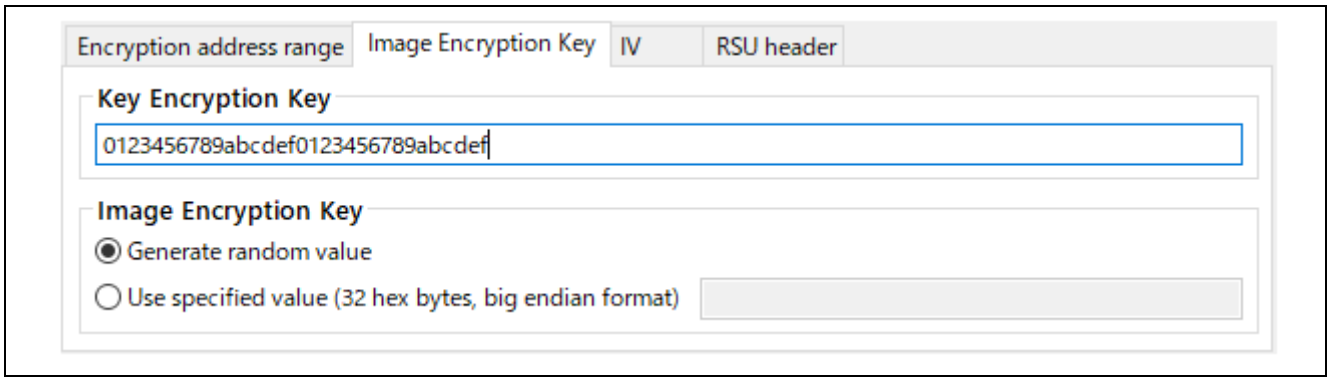


Figure 6.50 Generating an Encrypted File (TSIP UPDATE Tab – Image Encryption Key Tab)

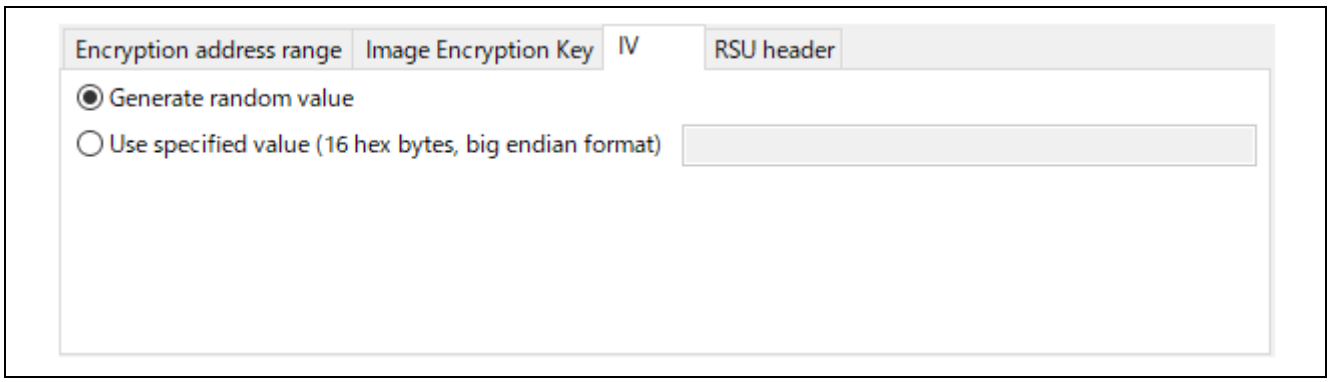


Figure 6.51 Generating an Encrypted File (TSIP UPDATE Tab – IV Tab)

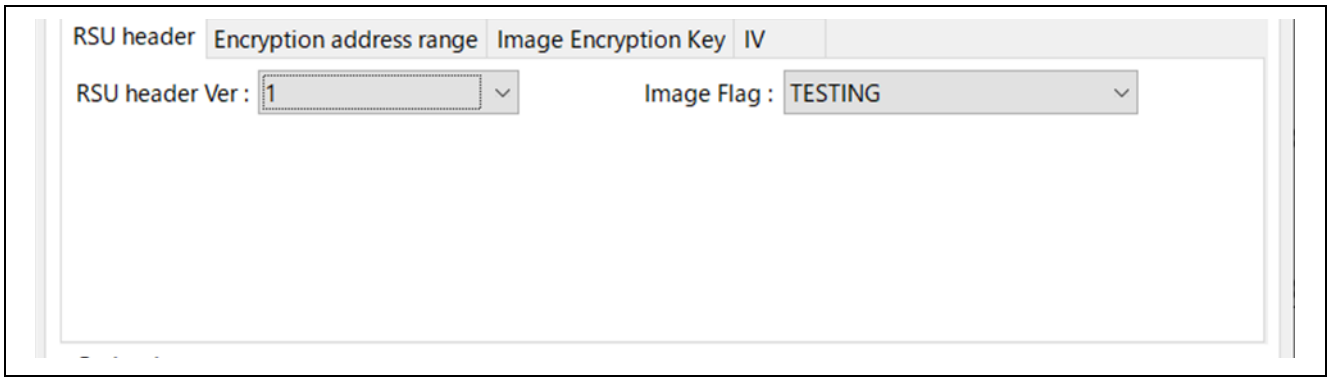


Figure 6.52 Generating an Encrypted File (TSIP UPDATE Tab – RSU Header Tab)

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 6-7, Tera Term serial settings or Table 6-8, Tera Term settings (1 area update) to make serial setting.

(6) Running the Secure Bootloader Project

Select **rx*XXX*_*bbb*_tsip_secure_boot** in e^2 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.

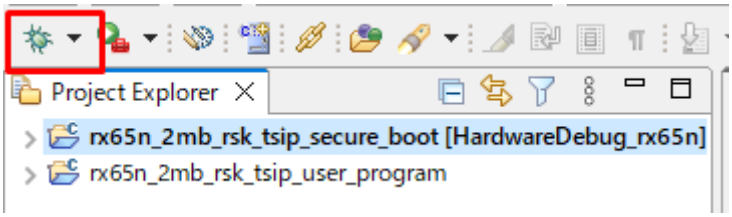


Figure 6.53 Running the Secure Bootloader Project (RSK RX65N)

(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, silillar to the following is displayed in Tera Term.

```
HELLO!! this is boot program. ~
$ I was built in Mar 31 2025, 14:54:11.
bank info = 1. (start bank = 0)
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.
bank info = 1. (start bank = 0)
Checking flash ROM status.
status = STATE_GET_IMAGE
Copy Secure Boot...OK

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

Figure 6.54 Log Output of Secure Bootloader up to Wait for Initial User Program Installation (RSK RX65N)

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.

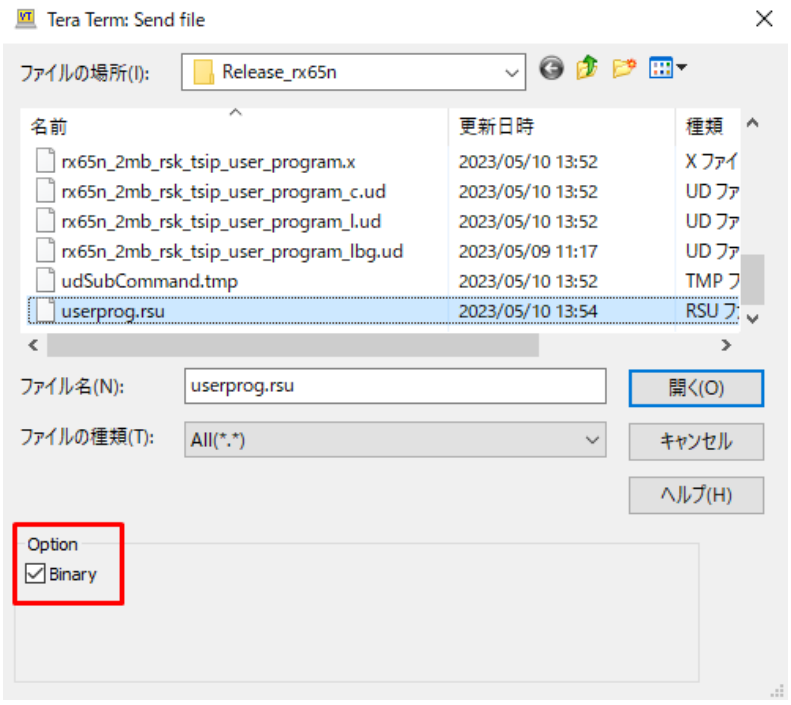


Figure 6.55 Tera Term: Send file Dialog Box

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

```
W 0xFFE00000, 128 ... OK
W 0xFFE00080, 128 ... OK
W 0xFFE00100, 128 ... OK
W 0xFFE00180, 128 ... OK
W 0xFFE00200, 128 ... OK
W 0xFFE00280, 128 ... OK
transit TSIP status to UpdateFirmware: OK
extract update file parameters: OK
decrypt program:
W 0xFFE00300, 128 ... OK
W 0xFFE00380, 128 ... OK
W 0xFFE00400, 128 ... OK

...

W 0xFFE16280, 128 ... OK
W 0xFFE16300, 128 ... OK
W 0xFFEEFF80, 128 ... OK
erase data flash(main)...OK
write data flash(main)...OK
erase data flash(mirror)...OK
write data flash(mirror)...OK
data flash setting OK.
bank info = 1. (start bank = 0)
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 Mar 31 2025, 14:54:11.
bank info = 0. (start bank = 1)
Checking flash ROM status.
status = STATE_EXEC_IMAGE

==== BootLoader [dual bank] ====
secure boot sequence: success.
execute image ...
HELLO!! this is user program. ~
$ I was built in Mar 31 2025, 14:12:25.
Version ver 1.00.
tsip@rx65n
$
```

Figure 6.56 Installation Log Message Output for Secure Bootloader Initial Firmware (RSK RX65N)

(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 MOT file as described in step (3), encrypt it as described in step (4).

```
/* Command prompt related */
#define PROMPT                  ("tsip@rx65n\r\n$ ")
#define VERSION                 ("ver 1.01")
#define HELLO_MESSAGE           ("HELLO!! this is user program. ~\r\n$ ")
```

Figure 6.57 Modified Location in main.c (rx65n_2mb_rsk_tsip_user_program)

Execute the **update** command. Install the user program that has been updated to Ver 1.01 as described in step (7). After the upload finishes, the system reboots.

```
HELLO!! this is boot program. ~
$ I was built in Mar 31 2025, 16:24:24.
bank info = 1. (start bank = 0)
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.
bank info = 1. (start bank = 0)
Checking flash ROM status.
status = STATE_GET_IMAGE
Copy Secure Boot...OK

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

...

W 0xFFE0D780, 128 ... OK
W 0xFFE0D800, 128 ... OK
W 0xFFEEFF80, 128 ... OK
erase data flash(main)...OK
write data flash(main)...OK
erase data flash(mirror)...OK
write data flash(mirror)...OK
data flash setting OK.
bank info = 1. (start bank = 0)
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 Mar 31 2025, 16:24:24.
bank info = 0. (start bank = 1)
Checking flash ROM status.
status = STATE_EXEC_IMAGE

==== BootLoader [dual bank] ====
secure boot sequence: success.
execute image ...
HELLO!! this is user program. ~
$ I was built in Mar 31 2025, 15:05:48.
Version ver 1.01.
tsip@rx65n
$
install start ...

==== Update from User [dual bank]  ver 1.0.0 ====
Erase buffer area...OK
send image(*.rsu) via UART.
```

Figure 6.58 Firmware Update Log Message Output (RSK RX65N)

6.2.4 Execution Example of Factory Programming Using Motorola S Format File Containing Secure Bootloader and Encrypted User Program

This section describes how to create a Motorola S format file containing a secure bootloader and encrypted user program and how to program it to a device using Renesas Flash Programmer (factory programming).

(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, Secure Bootloader and Firmware Update Execution Example.

(2) Building the Secure Bootloader Project

After importing the secure bootloader project (rx*XXX*_bbb_tsip_secure_boot) into the e^2 studio workspace, place euk_aes128.c and euk_aes128.h generated in step (1) in the src folder of the rx*XXX*_bbb_tsip_secure_boot project.

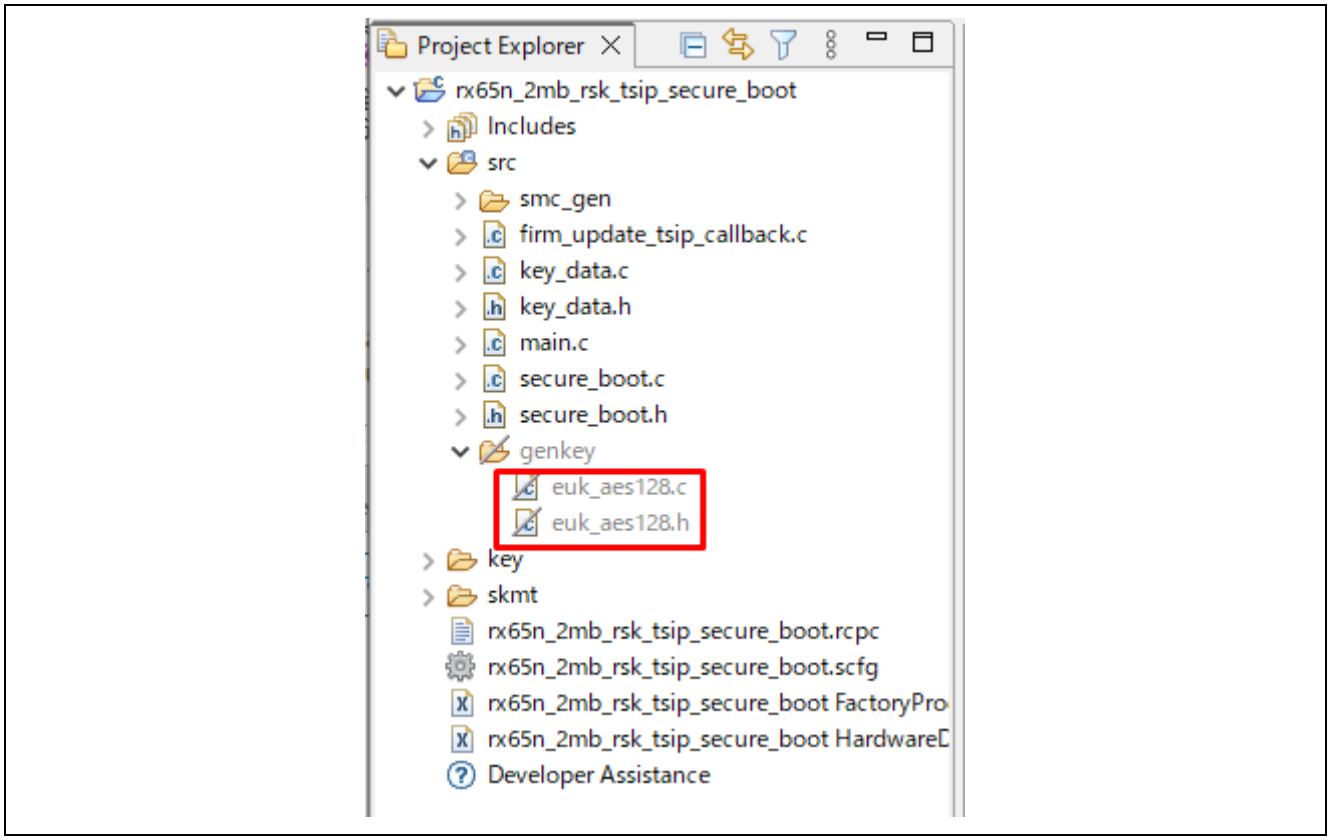


Figure 6.59 e^2 studio Project Explorer

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

In e^2 studio's Project Explorer, right-click the secure update project and on the menu that appears select **Build Configuration** > **Activate** > **Factory Programming**.

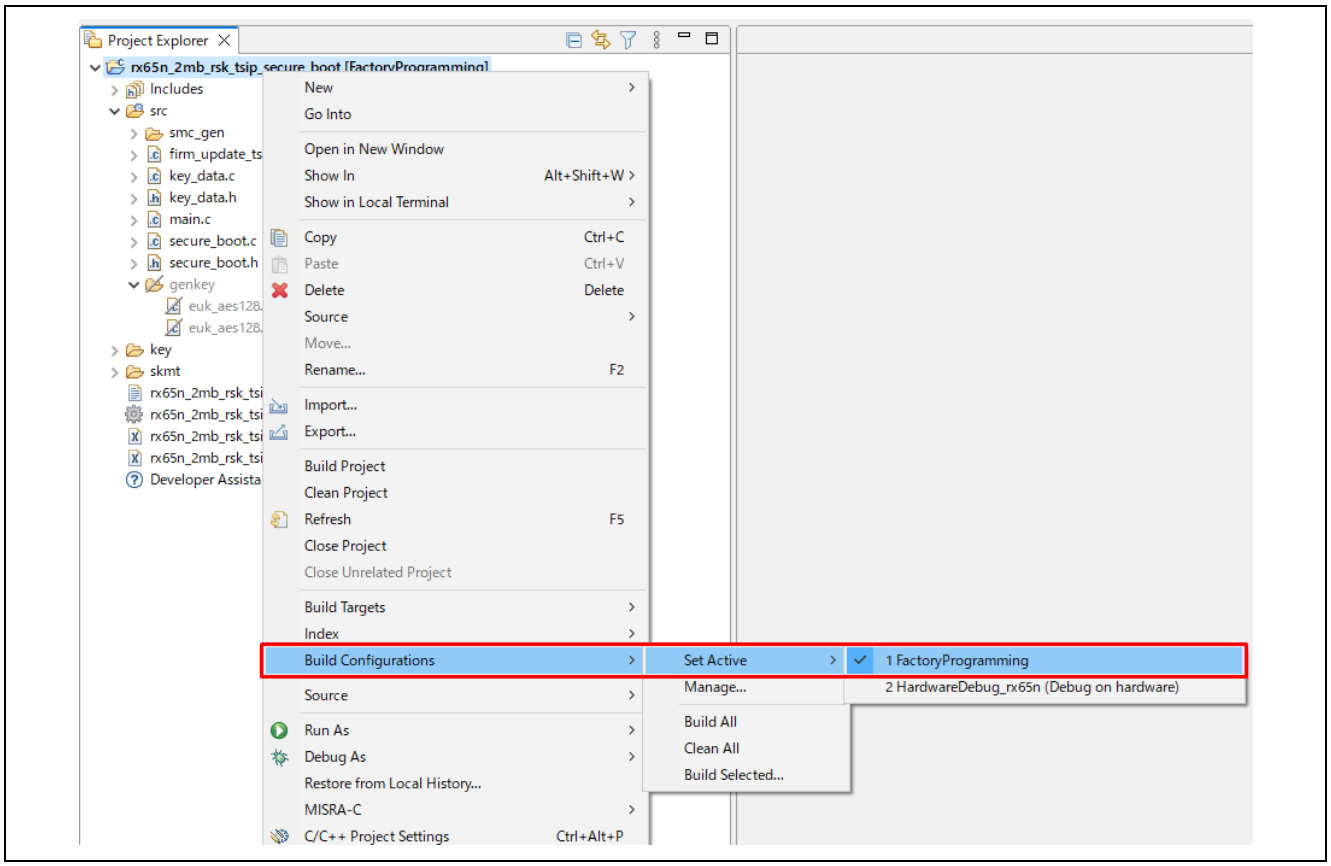


Figure 6.60 Changing Build Configuration to Factory Programming

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, Secure Bootloader and Firmware Update Execution Example.

(4) Encrypting the Firmware Update Program

- Command line version
  When to use RSU header Ver.2, run the following command.

```
> skmt.exe /enctsip /mode "factory" /ver "2" /prg
    ""{$skmt_loc}\..\rx65n_2mb_rsk_tsip_user_program\Release_rx65n\rx65n_2mb_rsk_tsip_user
    _program.mot"
    /prg_sb "{$skmt_loc}\FactoryProgramming\rx65n_2mb_rsk_tsip_secure_boot.mot" /enckey
    "0123456789ABCDEF0123456789ABCDEF" /session_key
    "FEDCBA9876543210FEDCBA98765432100123456789ABCDEF0123456789ABCDEF"
    /iv_fw "55AA55AA55AA55AA55AA55AA55AA55AA" /startaddr "FFF00300" /endaddr
    "FFFEFFFF" /filetype "mot" /flash_wsize 128 /output "{$skmt_loc}\userprog.mot"
```

  When to use RSU header Ver.2, run the following command.

```
> skmt.exe /enctsip /mode "factory" /ver "1" /prg
    "{$skmt_loc}\..\rx65n_2mb_rsk_tsip_user_program\Release_rx65n\rx65n_2mb_rsk_tsip
    _user_program.mot"
    /prg_sb "{$skmt_loc}\FactoryProgramming\rx65n_2mb_rsk_tsip_secure_boot.mot "
    /enckey "0123456789abcdef0123456789abcdef"
    /startaddr "FFE00300" /endaddr "FFFEFFFF" /destaddr "FFE00300"
    /imgflg "valid" /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 **/stardaddr**, **/endaddr**, and **/destaddr** differ depending on the MCU. Refer to the table below when entering values.

Table 6-15 Selections for Each MCU

MCU	Parameter		Address
	GUI	CLI	
RX231 RX66T	Start address	/startaddr	FFFB8300
	End address	/endaddr	FFFEFFFF
	Encrypted image output address	/destaddr	FFFB8300
RX72T	Start address	/startaddr	FFF78300
	End address	/endaddr	FFFEFFFF
	Encrypted image output address	/destaddr	FFF78300
RX26T	Start address	/startaddr	FFFC0300
	End address	/endaddr	FFFEFFFF
	Encrypted image output address	/destaddr	FFFC0300
RX65N RX671	Start address	/startaddr	FFF00300
	End address	/endaddr	FFFEFFFF
	Encrypted image output address	/destaddr	FFF00300
RX72M RX72N	Start address	/startaddr	FFE00300
	End address	/endaddr	FFFEFFFF
	Encrypted image output address	/destaddr	FFE00300

- Standalone version
  Enter the following values.
  — **Output Image**
  Select **Secure Update**.
  — **Firmware Image**
  MOT file output by update project (rx*XXX_bbb*_tsip_user_program.mot)
  — **Encrypted Address Range** tab

The values differ depending on the MCU. Refer to Table 6.15 and select the appropriate values to match the MCU.

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

— **RSU Header** tab
RSU header Ver: 2

— **Output**
Format: Binary
File: userprog.mot

To use the above settings, you can load the Security Key Management Tool settings file rx*XXX*_SecureUpdate.skmt included in the sample.

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

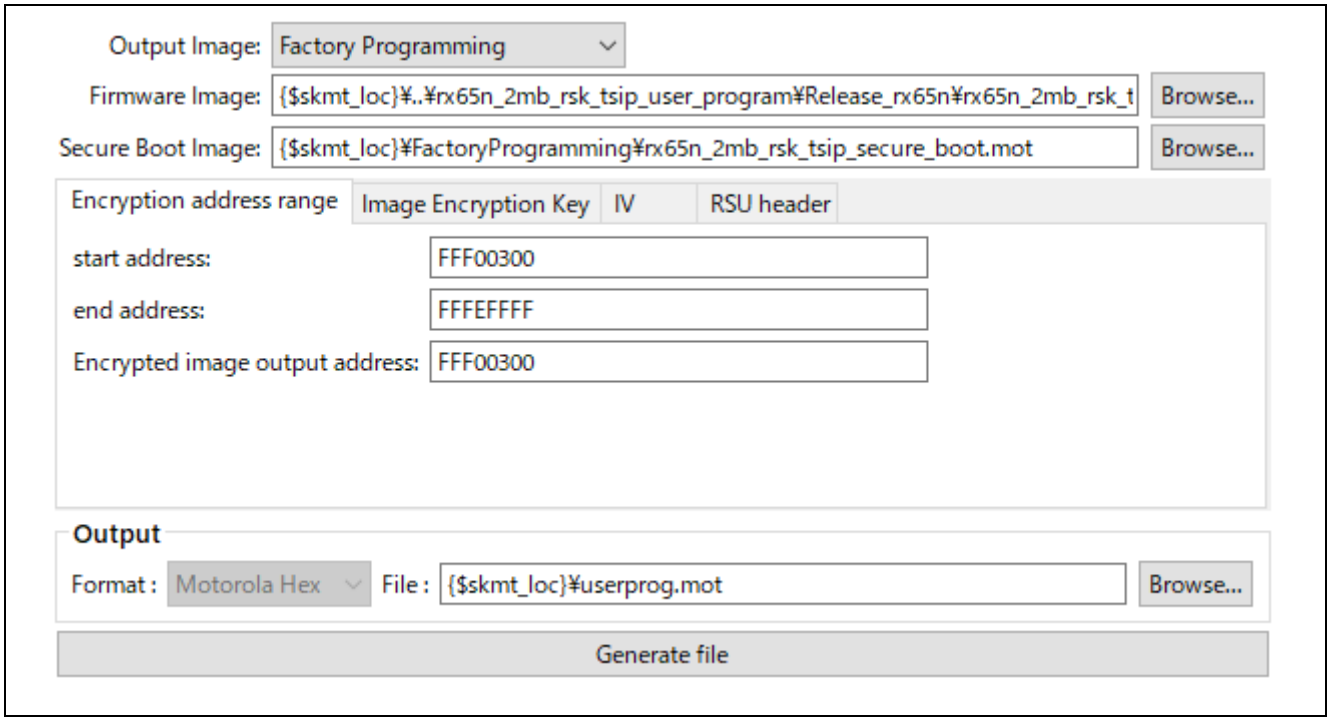


Figure 6.61 Generating an Encrypted File (TSIP UPDATE Tab)

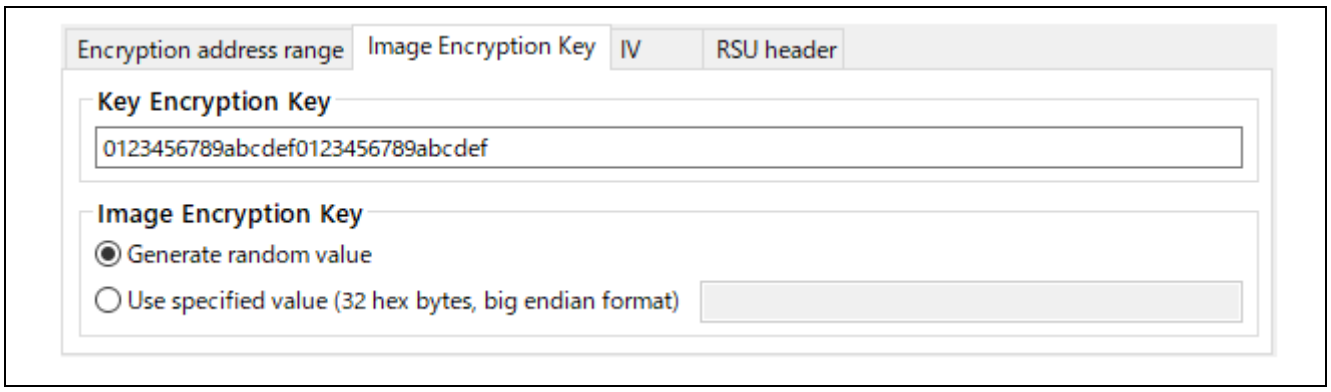


Figure 6.62 Generating an Encrypted File (TSIP UPDATE Tab – Image Encryption Key Tab)

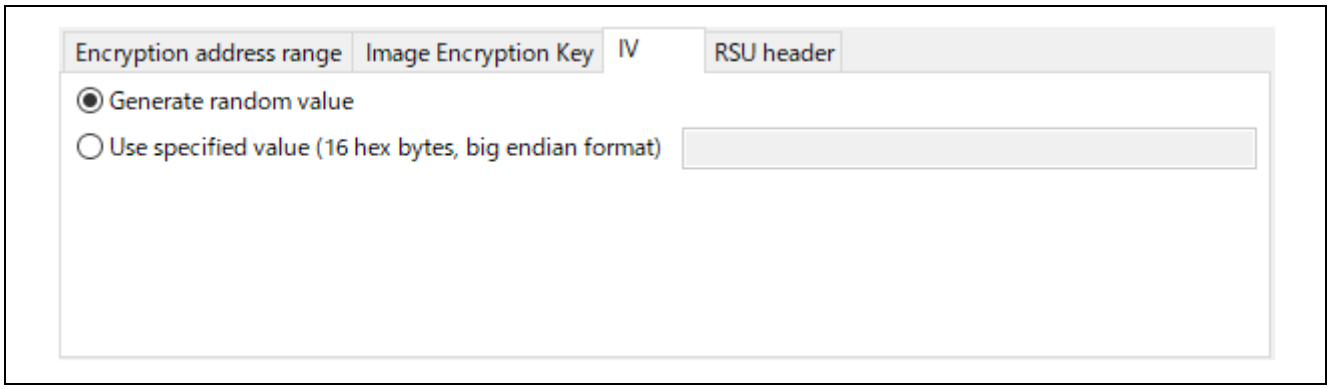


Figure 6.63 Generating an Encrypted File (TSIP UPDATE Tab – IV Tab)

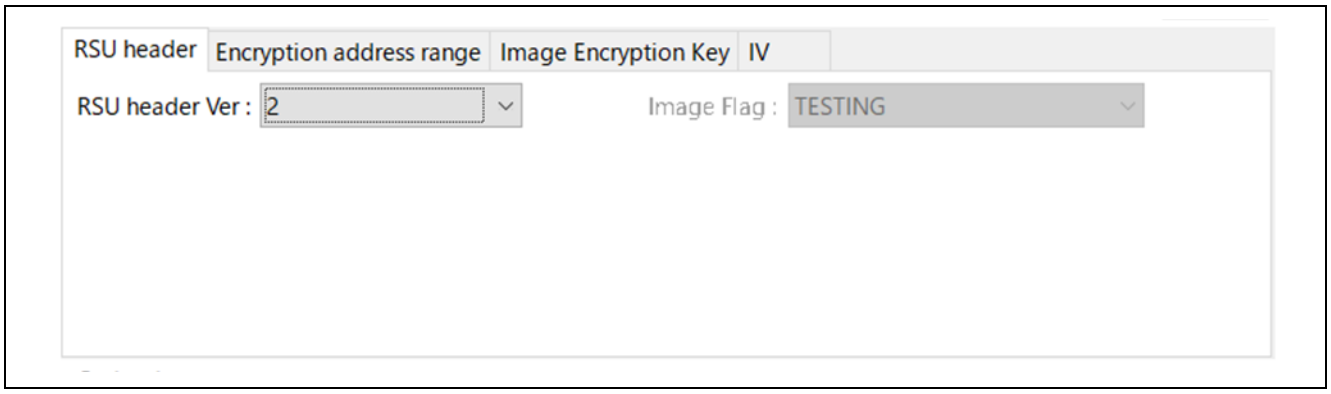


Figure 6.64 Generating an Encrypted File (TSIP UPDATE Tab – RSU Header Tab)

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**.

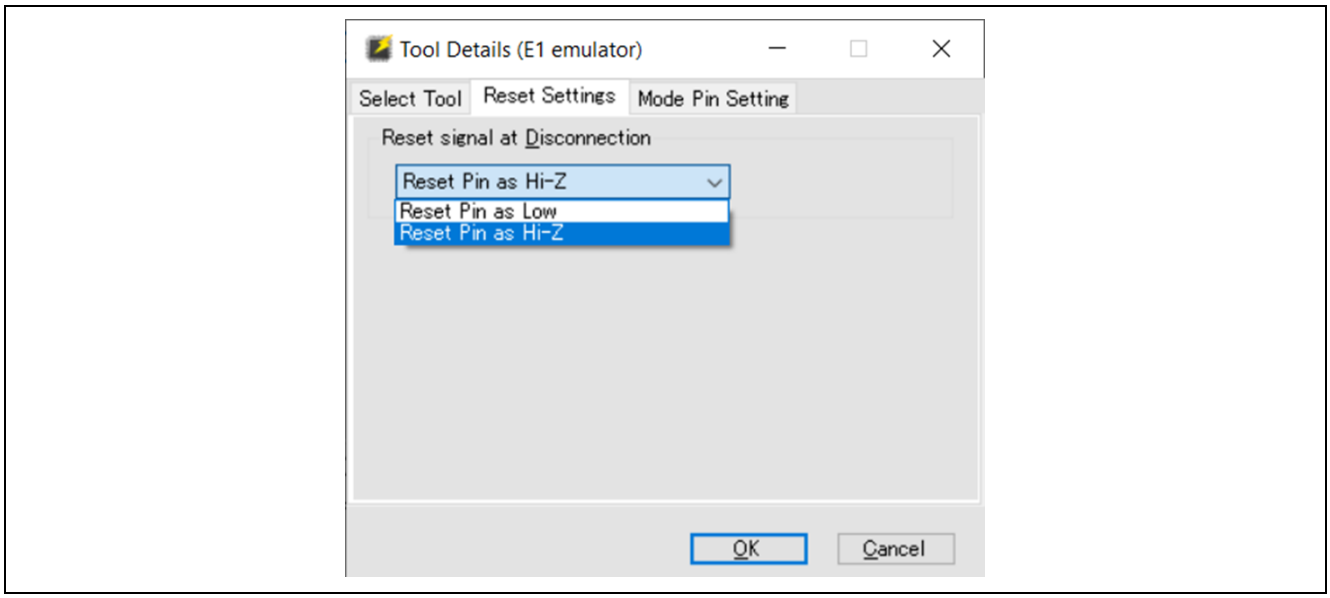


Figure 6.65 Renesas Flash Programmer Reset Settings Tab

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 Mar 31 2025, 15:10:47.
bank info = 1. (start bank = 0)
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.
bank info = 1. (start bank = 0)
Checking flash ROM status.
status = STATE_GET_IMAGE
Copy Secure Boot...OK

==== Image updater [dual bank] ====
Erase buffer area...OK
W 0xFFE00000, 128 ... OK
W 0xFFE00080, 128 ... OK
W 0xFFE00100, 128 ... OK
W 0xFFE00180, 128 ... OK
W 0xFFE00200, 128 ... OK
W 0xFFE00280, 128 ... OK
transit TSIP status to UpdateFirmware: OK
extract update file parameters: OK
decrypt program:
W 0xFFE00300, 128 ... OK
W 0xFFE00380, 128 ... OK

...

W 0xFFE0D800, 128 ... OK
W 0xFFEEFF80, 128 ... OK
erase data flash(main)...OK
write data flash(main)...OK
erase data flash(mirror)...OK
write data flash(mirror)...OK
data flash setting OK.
bank info = 1. (start bank = 0)
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 Mar 31 2025, 15:10:47.
bank info = 0. (start bank = 1)
Checking flash ROM status.
status = STATE_EXEC_IMAGE

==== BootLoader [dual bank] ====
secure boot sequence: success.
execute image ...
HELLO!! this is user program. ~
$ I was built in Mar 31 2025, 15:11:36.
Version ver 1.00.
tsip@rx65n
$
```

Figure 6.66 **Log Output when Using Renesas Flash Programmer to Write Encrypted User Program (RSK RX65N)**

(7) Firmware Update Operation

For firmware update operation, refer to step (8), Firmware Update Operation, under 6.2.3, Secure Bootloader and Firmware Update Execution Example.

6.2.5 Debugging User Program Projects

Since the user program project is started via Secure Boot, the reset vector (RESETVECT) of the project is placed at 0xFFF7FFFC. For this reason, the user program project cannot be debugged as is. To debug the user program project, switch to the debugging configuration shown in Debugging Configuration of Each User Program.

Table 6-16 Debugging configuration for each user program

Board	User Program	Debug Configuration
RSK RX231	rx231_rsk_tsip_user_program	HardwareDebug_rx231
MCB RX26T	rx26t_mcb_tsip_sci_user_program	HardwareDebug_rx26t
RSK RX65N	rx65n_2mb_rsk_tsip_user_program	HardwareDebug_rx65n
RSK RX66T	rx66t_rsk_tsip_sci_user_program	HardwareDebug_rx66t
RSK RX671	rx671_rsk_tsip_user_program	HardwareDebug_rx671
RSK RX72M	rx72m_rsk_tsip_user_program	HardwareDebug_rx72m
RSK RX72N	rx72n_rsk_tsip_user_program	HardwareDebug_rx72n
RX72N Envision Kit	rx72n_ek_tsip_user_program	HardwareDebug_rx72n
RSK RX72T	rx72t_rsk_tsip_user_program	HardwareDebug_rx72t

- How to switch debugger connection configuration:

1.Press ▼ next to e²studio menu build button 
Select [HardwareDebug_rx231(Debug on hardware)] to start the build.

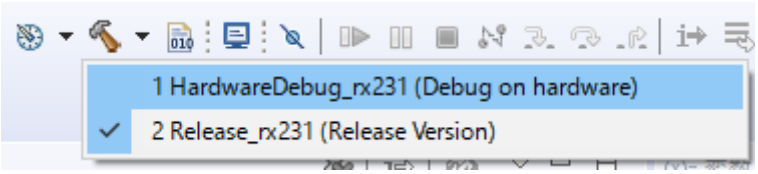


Figure 6.67 e²studio project menu(for RX231)

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

Difference between "Release_rxXXX" configuration for release and "HardwareDebug_rxXXX" configuration for debugging

The addresses of the reset vector (RESETVECT) and interrupt vector (EXCEPTVECT) at build time are different between the "Release_rxXXX" and "HardwareDebug_rxXXX" configurations for each user program project.

Table 6-17 User program project Vector setting address for each configuration

Symbol	Release_rx*XXX*	HardwareDebug_rx*XXX*
RESETVECT	0xFFFEFFFC	0xFFFFFFFC
EXCEPTVECT	0xFFFEFF80	0xFFFFFF80

6.2.6 BSP Modification of RX231 Secure Bootloader Demo Project

In the demo projects of secure bootloader, the configuration BSP_CFG_BOOTLOADER_PROJECT is set to 1 because the projects behave as bootloader projects. When the configuration is set to 1, functions in the BSP FIT module which are not supposed to be used are disabled.
However, USB interface is disabled in BSP FIT module of RX231 when the configuration BSP_CFG_BOOTLOADER_PROJECT is set to 1. Therefore, the BSP FIT module of RX231 secure bootloader project is modified to enable the firmware update with using USB memory. When implementing the secure bootloader which supports USB interface to RX231, please refer to the modifications in this BSP FIT module.

6.2.7 Notes on Transition from Secre Bootloader to User Program

When the transition from the secure bootloader program to the user program takes place, the peripheral function settings of the secure bootloader program are inherited by the application. In the bootloader sample program this 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.

6.2.8 Performance Information for Demo Project

Performance information for each Secure Bootloader demo project is shown below.

Performance is measured in cycles of ICLK, the core clock. The drivers are built using CC-RX with optimization level 2. Othe conditions are shown below.

Module revision: r_tsip_rx rev1.23

Operating clock: ICLK : PCLKB = 2 : 1

Compiler version: Renesas Electronics C/C++ Compiler Package for RX Family V3.07.00
(integrated development environment default settings with “-lang = c99” option added)

As a performance information of the demo project, the processing time from reset to MAC verifcation for each Secure Bootloader demo project included in the package of this application note is shown in Table 6-18.

Table 6-18 Performance of Secure Bootloader Demo Project (Unit: Cycle)

Board	Performance (Reset ~ MAC Verification)
RSK RX231	11,000,000
MCB RX26T	16,000,000
RSK RX65N	14,000,000
RSK RX66T	26,000,000
RSK RX671	14,000,000
RSK RX72M	29,000,000
RSK RX72N	30,000,000
RX72N Envision Kit	29,000,000
RSK RX72T	28,000,000

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 2025-07 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 14.2.0.202505 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. 1.23
Board used	Renesas Starter Kit for RX231 (B version) (product No.: R0K505231S020BE) Renesas Solution Starter Kit for RX23W (with TSIP) (product No.: RTK5523W8BC00001BJ) MCB-RX26T Type B (product No.: RTK0EMXE70C02000BJ) Renesas Starter Kit+ for RX65N-2MB (with TSIP) (product No.: RTK50565N2S10010BE) Renesas Starter Kit for RX66T (with TSIP) (product No.: RTK50566T0S00010BE) Renesas Starter Kit+ for RX671 (product No.: RTK55671xxxxxxxxxx) Renesas Starter Kit+ for RX72M (with TSIP) (product No.: RTK5572MNHSxxxxxxx) Renesas Starter Kit+ for RX72N (with TSIP) (product No.: RTK5572NNHCxxxxxxx) RX72N Envision Kit (product No.: RTK5RX72N0CxxxxxBJ) Renesas Starter Kit for RX72T (with TSIP) (product No.: RTK5572TKCS00010BE)

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 e^2 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 e^2 studio sample project on CS+.

A: Visit the following webpage for instructions:
Porting from the e^2 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 User Key Encryption Formats

For key injection the user key is wrapped using a UFPK and IV, and for key updating the user key is wrapped using a KUK and IV. The format of the key data to be wrapped depends on the cryptographic algorithm. This section lists the data formats for the user key to be encrypted (user key) and for the wrapped key (encrypted key).

Refer to 3.7.1, Key Injection and Updating, for information on encryption methods.

7.3.1 AES

7.3.1.1 AES 128-Bit Key

Input (User key)

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

Output (Encrypted key)

Bytes	16			
	4	4	4	4
0-15	encrypted_user_key (128-bit AES key)			
16-31	MAC			

7.3.1.2 AES 256-Bit Key

Input (User key)

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

Output (Encrypted key)

Bytes	16			
	4	4	4	4
0-31	encrypted_user_key (256-bit AES key)			
32-47	MAC			

7.3.2 DES

Input (User key)

Bytes	16			
	4	4	4	4
0-7	56-bit DES key with odd parity 1*1			
8-15	56-bit DES key with odd parity 2*1			
16-23	56-bit DES key with odd parity 3*1			

Output (Encrypted key)

Bytes	16			
	4	4	4	4
0-23	encrypted_user_key (56-bit DES key with odd parity 1 || 56-bit DES key with odd parity 2 || 56-bit DES key with odd parity 3)			
24-39	MAC			

Note: 1. Append an odd-parity bit to each 7 bits of key data.
Example: DES key data = 0x00000000000000 → 0x0101010101010101
DES key data = 0xFFFFFFFFFFFFFF → 0xFEFEFEFEFEFEFEFE

For 2-DES, insert the same key in the 56-bit DES key with odd parity 1 and the 56-bit DES key with odd parity 3.

For DES, insert the same value in the 56-bit DES key with odd parity 1, the 56-bit DES key with odd parity 2, and the 56-bit DES key with odd parity 3.

7.3.3 ARC4

Input (User key)

Bytes	16			
	4	4	4	4
0-255	ARC4			

Output (Encrypted key)

Bytes	16			
	4	4	4	4
0-255	encrypted_user_key (ARC4)			
256-272	MAC			

7.3.4 RSA

7.3.4.1 RSA 1024-Bit Key

(1) Public Key

Input (User key)

Bytes	16			
	4	4	4	4
0-127	RSA 1024-bit public key n			
128-143	RSA 1024-bit public key e	0 padding		

Output (Encrypted key)

Bytes	16			
	4	4	4	4
0-143	encrypted_user_key (RSA 1024-bit public key n || e || 0 padding)			
144-159	MAC			

(2) Secret Key

Input (User key)

Bytes	16			
	4	4	4	4
0-127	RSA 1024-bit public key n			
128-255	RSA 1024-bit secret key d			

Output (Encrypted key)

Bytes	16			
	4	4	4	4
0-255	encrypted_user_key (RSA 1024-bit public key n || secret key d)			
256-271	MAC			

7.3.4.2 RSA 2048-Bit Key

(1) Public Key

Input (User key)

Bytes	16			
	4	4	4	4
0-255	RSA 2048-bit public key n			
256-271	RSA 2048-bit public key e	0 padding		

Output (Encrypted key)

Bytes	16			
	4	4	4	4
0-271	encrypted_user_key (RSA 2048-bit public key n || e || 0 padding)			
272-287	MAC			

(2) Secret Key

Input (User key)

Bytes	16			
	4	4	4	4
0-255	RSA 2048-bit public key n			
256-511	RSA 2048-bit secret key d			

Output (Encrypted key)

Bytes	16			
	4	4	4	4
0-511	encrypted_user_key (RSA 2048-bit public key n || secret key d)			
512-527	MAC			

7.3.4.3 RSA 3072-Bit Key

(1) Public Key

Input (User key)

Bytes	16			
	4	4	4	4
0-383	RSA 3072-bit public key n			
384-399	RSA 3072-bit public key e	0 padding		

Output (Encrypted key)

Bytes	16			
	4	4	4	4
0-399	encrypted_user_key (RSA 3072-bit public key n || e || 0 padding)			
400-415	MAC			

7.3.4.4 RSA 4096-Bit Key

(1) Public Key

Input (User key)

Bytes	16			
	4	4	4	4
0-511	RSA 4096-bit public key n			
512-527	RSA 4096-bit public key e	0 padding		

Output (Encrypted key)

Bytes	16			
	4	4	4	4
0-527	encrypted_user_key (RSA 4096-bit public key n || e || 0 padding)			
528-543	MAC			

7.3.5 ECC

7.3.5.1 ECC P192

(1) Public Key

Input (User key)

Bytes	16			
	4	4	4	4
0-31	0 padding			
	ECC P 192-bit public key Qx			
32-63	0 padding			
	ECC P 192-bit public key Qy			

Output (Encrypted key)

Bytes	16			
	4	4	4	4
0-63	encrypted_user_key (0 padding || ECC P 192-bit public key Qx || 0 padding || ECC P 192-bit public key Qy)			
64-79	MAC			

(2) Secret Key

Input (User key)

Bytes	16			
	4	4	4	4
0-31	0 padding			
	ECC P 192-bit secret key d			

Output (Encrypted key)

Bytes	16			
	4	4	4	4
0-31	encrypted_user_key (0 padding || ECC P 192-bit secret key d)			
32-47	MAC			

7.3.5.2 ECC P224

(1) Public Key

Input (User key)

Bytes	16			
	4	4	4	4
0-31	0 padding ECC P 224-bit public key Qx			
32-63	0 padding ECC P 224-bit public key Qy			

Output (Encrypted key)

Bytes	16			
	4	4	4	4
0-63	encrypted_user_key (0 padding || ECC P 224-bit public key Qx || 0 padding || ECC P 224-bit public key Qy)			
64-79	MAC			

(2) Secret Key

Input (User key)

Bytes	16			
	4	4	4	4
0-31	0 padding ECC P 224-bit secret key d			

Output (Encrypted key)

Bytes	16			
	4	4	4	4
0-31	encrypted_user_key (0 padding || ECC P 224-bit secret key d)			
32-47	MAC			

7.3.5.3 ECC P256

(1) Public Key

Input (User key)

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

Output (Encrypted key)

Bytes	16			
	4	4	4	4
0-63	encrypted_user_key (ECC P 256-bit public key Qx || ECC P 256-bit public key Qy)			
64-79	MAC			

(2) Secret Key

Input (User key)

Bytes	16			
	4	4	4	4
0-31	ECC P 256-bit secret key d			

Output (Encrypted key)

Bytes	16			
	4	4	4	4
0-31	encrypted_user_key (ECC P 256-bit secret key d)			
32-47	MAC			

7.3.5.4 ECC P384

(1) Public Key

Input (User key)

Bytes	16			
	4	4	4	4
0-47	ECC P 384-bit public key Qx			
48-95	ECC P 384-bit public key Qy			

Output (Encrypted key)

Bytes	16			
	4	4	4	4
0-95	encrypted_user_key (ECC P 384-bit public key Qx || ECC P 384-bit public key Qy)			
96-111	MAC			

(2) Secret Key

Input (User key)

Bytes	16			
	4	4	4	4
0-47	ECC P 384-bit secret key d			

Output (Encrypted key)

Bytes	16			
	4	4	4	4
0-47	encrypted_user_key (ECC P 384-bit secret key d)			
48-63	MAC			

7.3.6 HMAC

7.3.6.1 SHA1-HMAC Key

Input (User key)

Bytes	16			
	4	4	4	4
0-31	HMAC-SHA1 key			
		0 padding		

Output (Encrypted key)

Bytes	16			
	4	4	4	4
0-31	encrypted_user_key (HMAC-SHA1 || 0 padding)			
32-47	MAC			

7.3.6.2 SHA256-HMAC Key

Input (User key)

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

Output (Encrypted key)

Bytes	16			
	4	4	4	4
0-31	encrypted_user_key (HMAC-SHA256)			
32-47	MAC			

7.3.7 KUK

Input (User key)

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

Output (Encrypted key)

Bytes	16			
	4	4	4	4
0-31	encrypted_user_key (AES 128-bit CBC key || CBCMAC key)			
32-47	MAC			

7.4 Wrapped Key of Public Key Formats for Asymmetric Cryptography

Public keys for asymmetric cryptography contain plaintext information in the wrapped key. This enables plaintext information to be extracted from the wrapped key using the TSIP's key generation functionality. The data format of each cryptographic algorithm is described below.

7.4.1 RSA

The key index structure tsip_rsaXXXX_public_key_index_t of the RSA public key contains the plaintext data of the public key in members value.key_n and value_e.

The modulus and exponent values are output in big-endian byte ordering as key_n and key_e, respectively.

7.4.2 ECC

The member value.key_q of the wrapped key structure tsip_ecc_public_key_index_t of the ECC public key contains the plaintext data of the public key. The format of key_q is as shown below.

7.4.2.1 ECC P 192-Bit Key

Bytes	128 bits			
	32 bits	32 bits	32 bits	32 bits
0-15	0 padding		ECC P-192 public key Qx	
16-31	ECC P 192-bit public key Qx (continuation)			
32-47	0 padding		ECC P-192 public key Qy	
48-63	ECC P 192-bit public key Qy (continuation)			
64-79	Wrapped key management information			

7.4.2.2 ECC P 224-Bit Key

Bytes	128 bits			
	32 bits	32 bits	32 bits	32 bits
0-15	0 padding	ECC P-224 public key Qx		
16-31	ECC P 224-bit public key Qx (continuation)			
32-47	0 padding	ECC P-224 public key Qy		
48-63	ECC P 224-bit public key Qy (continuation)			
64-79	Wrapped key management information			

7.4.2.3 ECC P 256-Bit Key

Bytes	128 bits			
	32 bits	32 bits	32 bits	32 bits
0-31	ECC P-256-bit public key Qx			
32-63	ECC P 256-bit public key Qy			
64-79	Wrapped key management information			

7.4.2.4 ECC P 384-Bit Key

Bytes	128 bits			
	32 bits	32 bits	32 bits	32 bits
0-47	ECC P 384-bit public key Qx			
48-95	ECC P 384-bit public key Qy			
96-111	Wrapped key management information			

7.5 Encrypted Key Generation in Dynamic Operation

The operations to generate encrypted key which is described in the Figure 3.3 User Key Wrapping Scheme during Key Injection and Key Updating in 3.7.1 Key Injection and 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.

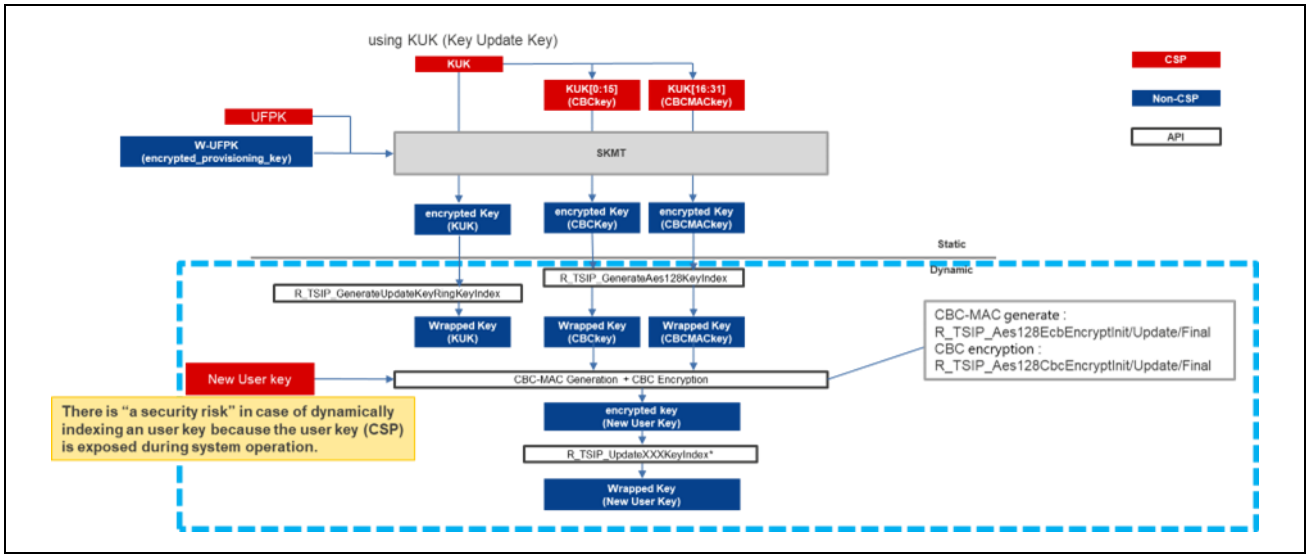


Figure 7.1 Encrypted Key Generation in Dynamic Operation

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 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 */
uint8_t iv[16] = “Initial vector”;

tsip_aes_key_index_t CBCMACkey;
tsip_aes_key_index_t CBCkey;
tsip_update_key_ring_t KUK;
tsip_XXX_key_index* wrapped_key;

tsip_aes_handle_t aes_handle;
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*/
R_TSIP_GenerateAes128KeyIndex(‘W-UFPK’, ‘Initial vector’,
        ‘Encrypted key for 1st half of KUK as CBCkey’, &CBCkey);
R_TSIP_GenerateAes128KeyIndex(‘W-UFPK’, ‘Initial vector’,
        ‘Encrypted key for 2nd half of KUK as CBCMACkey’, &CBCMACkey);

/* Inject KUK */
R_TSIP_GenerateUpdateKeyRingKeyIndex(‘W-UFPK’, ‘Initial vector’,
        ‘Encrypted key for KUK’, &KUK);
R_TSIP_Close();
R_TSIP_Open(NULL, &KUK);
```

```
/* AES-128 CBC-MAC */
R_TSIP_Aes128EcbEncryptInit(&aes_handle, &CBCMACkey);
for (i = 0; i < LEN; i += 16)
{
        for (j=0; j<16; j++)
        {
                work_input[j] = user_key[i+j] ^ mac[j];
        }
        R_TSIP_AES128EcbEncryptUpdate(&aes_handle, work_input, work_output, 16);
        memcpy(mac, work_output, 16);
}
R_TSIP_Aes128EncryptFinal(&aes_handle, work_dummy, &dummy);

/* AES-128 CBC-Encryption */
R_TSIP_Aes128CbcEncryptInit(&aes_handle, &CBCkey, iv);
R_TSIP_Aes128CbcEncryptUpdate(&aes_handle, user_key, encrypted_key, len);
R_TSIP_Aes128CbcEncryptUpdate(&aes_handle, mac, &encrypted_key[len], 16);
R_TSIP_Aes128CbcEncryptFinal(&aes_handle, work_dummy, &dummy);

/* Generate wrapped key */
R_TSIP_UpdateXXXKeyIndex*(iv, encrypted_key, &“Wrapped key structure for the User 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.

In addition, R_TSIP_UpdateXXXKeyIndex* means the API which updates the wrapped key which type is determined by the user key type.

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	Sep. 27, 2015	—	First edition issued
1.01	Jun. 27, 2016	—	Added firmware update functionality
1.02	May 31, 2017	—	• Added AES-CMAC functionality • Combined R_TSIP_SelfCheck1(), R_TSIP_SelfCheck2(), and R_TSIP_SoftwareReset into R_TSIP_Open() • Added parameters for R_TSIP_InstallAes128UserKey() and R_TSIP_InstallAes256UserKey() • Changed content on firmware updating using a USB flash drive
1.03	Sep. 30, 2017	—	• Added following flags for SHA and RSA: TSIP_SHA1, TSIP_SHA256, TSIP_RSA_1024, TSIP_RSA_2048 • Added R_TSIP_ERROR_PROHIBIT_FUNCTION error
1.04	Feb. 28, 2018	—	• Added following flags for TLS and SECURE BOOT: TSIP_TLS, TSIP_SHA_1_HMAC, TSIP_SHA_256_HMAC, SECURE_BOOT • Changed return value type to enum e_tsip_err_t • Deleted R_TSIP_Rsa1024ModularExponent • Deleted R_TSIP_Rsa2048ModularExponent • Added TLS APIs
1.05	Apr. 30, 2018	—	• Added following flags for TDES and MD5: TSIP_TDES_ECB_ENCRYPT, TSIP_TDES_ECB_DECRYPT, TSIP_TDES_CBC_ENCRYPT, TSIP_TDES_CBC_DECRYPT, TSIP_MD5 Changed following flags for RSA RSASSA_1024, RSASSA_2048, RSAES_1024, RSAES_2048 • Added MD5 APIs • Added TDES APIs • Added RSAES-PKCS1-v1_5 APIs • Added TLS APIs
1.06	Sep. 28, 2018	—	• Added support for RX66T • Deleted R_TSIP_TlsAes128CbcEncryptInit/Update/Final, R_TSIP_TlsAes128CbcDecryptInit/Update/Final, R_TSIP_TlsAes256CbcEncryptInit/Update/Final, R_TSIP_TlsAes256CbcDecryptInit/Update/Final, R_TSIP_TlsSha1HmacGenerateInit/Update/Final, R_TSIP_TlsSha256HmacGenerateInit/Update/Final, R_TSIP_TlsSha1HmacVerifyInit/Update/Final, R_TSIP_TlsSha256HmacVerifyInit/Update/Final • Changed parameter key index type in Init API of each algorithm • Added key update APIs • R_TSIP_Aes128CbcEncryptInit/Update/Final R_TSIP_Aes128CbcDecryptInit/Update/Final R_TSIP_Aes256CbcEncryptInit/Update/Final R_TSIP_Aes256CbcDecryptInit/Update/Final Added description of support for TLS • Changed parameter key index type in Init API of each algorithm • Added key update APIs • Added RSA key generation API • Amended SHA-HMAC APIs
1.07	Feb. 28, 2019	—	Added support for RX72T

Rev.	Date	Description	
		Page	Summary
1.08	Sep. 30, 2019	—	• Added support for RX23W, RX72M, and elliptic curve cryptography • Changed SECURE_BOOT configuration to TSIP_SECURE_BOOT, and deleted TSIP_INSTALL_KEY_RING_INDEX • Changed R_TSIP_GenerateTdesUserKeyIndex to R_TSIP_GenerateTdesKeyIndex
1.09	Mar. 31, 2020	—	• Added CCM and HMAC key generation and ECDH and key wrap functionality • Added support for RX66N and RX72N • Deleted R_TSIP_Open parameter and terminology explanation of s_flash, and newly added files_flash.c • Added TSIP_USER_HASH_ENABLED configuration and user-defined function R_TSIP_RSA_IF_HASH used by RSA signature generation/verification function • Added TSIP_ERR_FAIL to return values of GCM calculation preparation functions and unified the order in which the return values are listed • Changed description of return value TSIP_ERR_PARAMETER to “Invalid input data” for cases other than handles • Added description of members of key index to Parameters for functions that generate RSA and ECC public keys • Amended description in Parameters for RSA encryption and decryption functions • Added functionality for selecting message and hash value as data types for signature generation and verification functions • Corrected spelling of “character” and “mirror” • Changed s_inst1/2 to key_index_1/2, including type, and added R_TSIP_UpdateTlsRsaPublicKeyIndex as target of key_index_1
1.10	Jun. 30, 2020	—	• Added ECC P-384 key installation, key generation, and key update functionality • Added ARC4 and ECDSA P-384 functionality • Added ECDH P-256 functionality support for RX72M, RX66N, and RX72N • Added key wrap functionality support for RX72M, RX66N, and RX72N • Added ARC4 functionality and support for RX72N to demo project • Deleted TSIP_USER_HASH_ENABLED configuration and user-defined function R_TSIP_RSA_IF_HASH used by RSA signature generation and verification functions • Added TSIP_ECDSA_P384, TSIP_ECDH_P256, and TSIP_USER_SHA_384_ENABLED configurations • Changed name of R_TSIP_EcdhXXX() ECDH key exchange function to R_TSIP_EcdhP256XXX() • Changed ECC public key structure tsip_ecc_public_key_index_t

Rev.	Date	Description	
		Page	Summary
1.11	Sep. 30, 2020	—	• Added DH 2048-bit and ECDHE 512-bit functionality • Unified the description of iv in Parameters for R_TSIP_GenerateXXXKeyIndex() and R_TSIP_UpdateXXXKeyIndex() • Deleted ECDH (AES GCM 128 with IV) functionality from R_TSIP_EcdhP256Init() • Changed R_TSIP_AesXXXKeyWrap() and R_TSIP_AesXXXKeyUnwrap() into API functions common to MCUs with TSIP-Lite and TSIP modules
1.12	Jun. 30, 2021	—	Added AES cryptography project and TLS cooperation function project
1.13	Aug. 31, 2021	—	Added support for RX671
1.14	Oct. 22, 2021	—	Added support for TLS 1.3 (RX65N only)
1.15	Mar. 31, 2022	—	• Added support for TLS 1.3 (RX66N, RX72M, and RX72N) • Added support for TLS 1.2 RSA 4096-bit • Added hash calculation-in-progress acquisition function • Deleted reffolder, r_tsip_md5_rx.c, and r_tsip_sha_rx.c, and added r_tsip_hash_rx.c
1.16	Sep. 15, 2022	—	• Added support for TLS 1.3 (Resumption/0-RTT) • Added support for AES-CTR • Added support for RSA3072 and RSA4096
1.17	Jan. 20, 2023	—	• Revised section structure of application note • Added support for TLS 1.3 server (RX65N and RX72N)
1.18	May 24, 2023	—	Added support for RX26T
1.19	Nov. 30, 2023	—	Update example of Secure Bootloader / Firmware Update
1.20	Feb. 28, 2024	—	Applied software workaround to avoid the HW issue of AES-CCM decryption (TSIP-Lite)
1.21	Jun. 28, 2024	—	• Added support for TLS 1.2 server (RX65N, RX66N, RX72M, and RX72N • Added supported public key type of R_TSIP_TlsServersEphemeralEcdhPublicKeyRetrieves
1.22	Apr. 10, 2025	—	• Added API functions for RSAES-OAEP • Added an API function for key derivation to use in SSH • Changed specification of Firmware Update APIs • Updated explanation of Confirmation Operation of the Secure Bootloader and Firmware Update Demo Projects • Deleted explanation of Renesas Secure Flash Programmer
1.23	Oct. 15, 2025	—	• Update description of R_TSIP_Open • Update description of ECDH API
1.24	Mar. 20, 2026	—	Included spdx file in the package

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/.