RX Family

TSIP (Trusted Secure IP) Module Firmware Integration Technology (Binary version)

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.

Confirmed Devices

TSIP: RX65N and RX651 groups, 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.

There is an application note describing the details of the TSIP driver.

This application note will be explained using the key attached to the sample program. The key for mass production needs to be newly generated. An application note with the key details is available.

We will provide the product to customers who will be adopting or plan to adopt a Renesas microcontroller. Please contact your local Renesas Electronics sales office or distributor.

https://www.renesas.com/contact/

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 (56-/56x2-/56x3-bit) ECB/CBC: FIPS 46-3
	ARC4	ARC4 (2048-bit)
Hashing	SHA	SHA-1, SHA-256: FIPS 180-4
	MD5	MD5: RFC 1321
Authenticated encryption with associated data (AEAD)		GCM/CCM: FIPS 197, 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 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 Supported cipher suites (TLS 1.3)*2: 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 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. Applicable devices 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, SP800-38D
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)

Note: 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 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 800-38D, 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 key index into the device at the factory.	—
Key updating	Injecting a key index 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 encrypting a user key using a provisioning key or update keyring and adding a MAC value. An encrypted key corresponding to the same user key is a shared value on each device.	eKey-1
Key index	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 key index has been wrapped using an HUK, so the key index of the same encrypted key will be a unique value on each device.	Index-1 or Index-2
Provisioning 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
Encrypted provisioning key	Key information generated by wrapping a provisioning key using an HRK on the DLM server. The provisioning key is decrypted using the HRK internally by the TSIP.	Index-2
Update keyring	A keyring set by the user and used to generate an encrypted key from a user key during key updating. The key index for the update keyring must be generated beforehand by key injection in order to perform key updating on the device.	—
Hidden 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 provisioning keys.	—

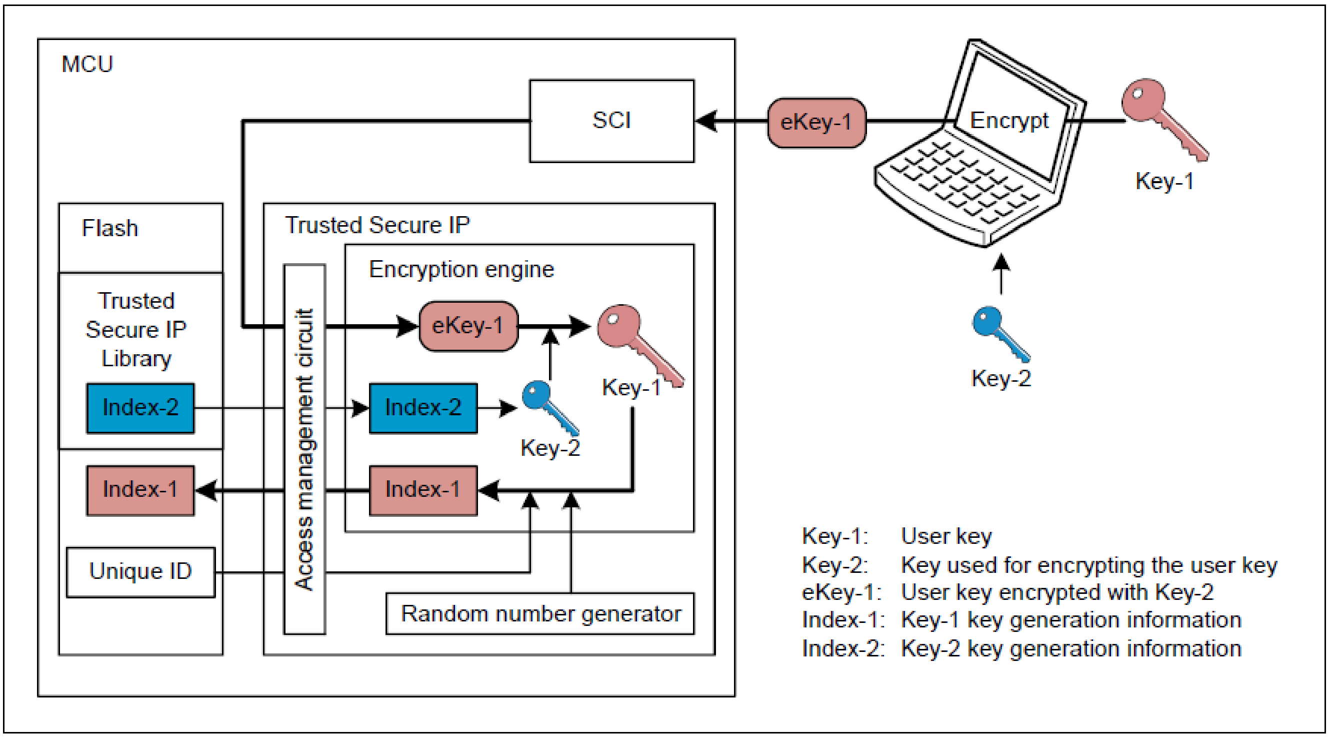

Figure 1.1 Diagram of Key Installation Process (Reproduced from Figure 52.4 in section 52, Trusted Secure IP (TSIP), in RX65N Group, RX651 Group, User's Manual: Hardware)

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 key index 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 key index the encryption key used for cryptographic operations. This operation is performed internally by the TSIP. The key index is tied to an HUK derived from a unique ID, making it device-specific. This means that even if a key index 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.

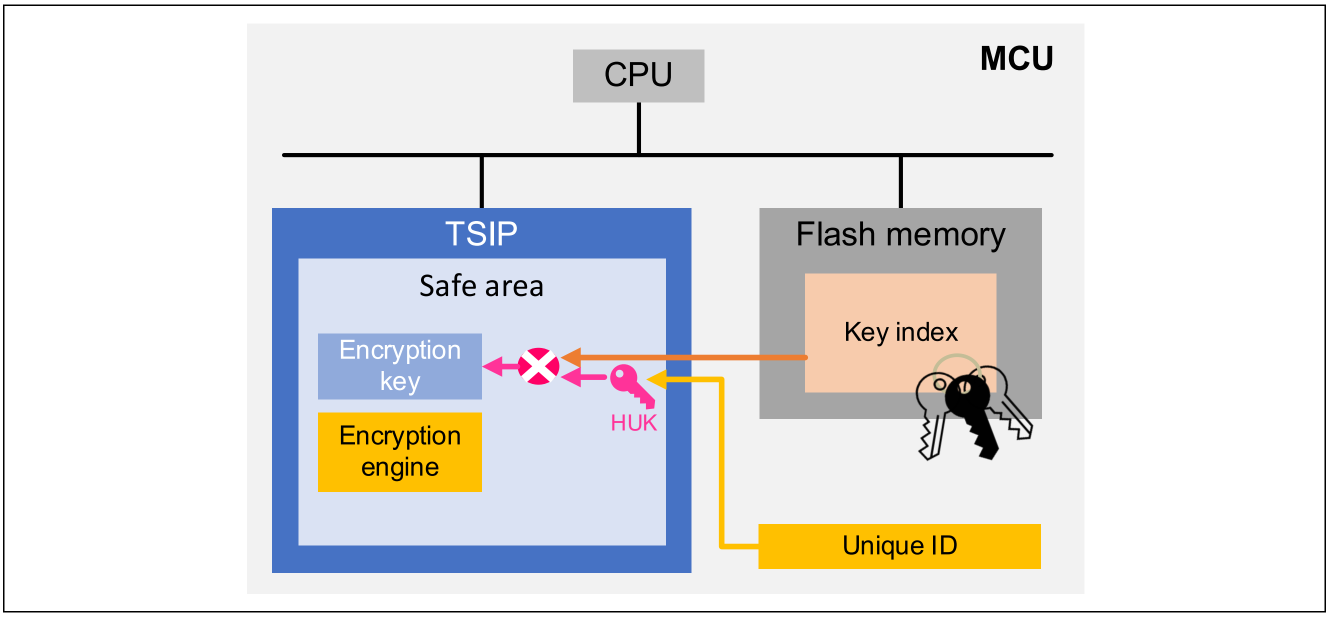

Figure 1.2 MCU Incorporating TSIP

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
RX_TSIP_SoftwareLicenseAgreement_ENG.pdf	Software License Agreement (English)
RX_TSIP_SoftwareLicenseAgreement_JPN.pdf	Software License Agreement (Japanese)
r20an0548jj0118-rx-tsip-security.pdf	TSIP driver application note (Japanese)
r20an0548ej0118-rx-tsip-security.pdf	TSIP driver application note (English)
reference_documents	Folder containing documentation of topics such as how to use the FIT module with various integrated development environments
ja	Folder containing documentation of topics such as how to use the FIT module with various integrated development environments (Japanese)
r01an1826jj0110-rx.pdf	How to add FIT modules to CS+ projects (Japanese)
r01an1723ju0121-rx.pdf	How to add FIT modules to e^2 studio projects (Japanese)
r20an0451js0140-e2studio-sc.pdf	Smart Configurator user's guide (Japanese)
r01an5792jj0101-rx-tsip.pdf	AES cryptography project application note (Japanese)
r01an5880jj0102-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)
r01an5792ej0101-rx-tsip.pdf	AES cryptography project application note (English)
r01an5880ej0102-rx-tsip.pdf	TLS cooperation function project application note (English)
FITModules	FIT module folder
r_tsip_rx_v1.18.l.zip	TSIP driver FIT module
r_tsip_rx_v1.18.l.xml	TSIP driver FIT module e^2 studio FIT plug-in XML file
r_tsip_rx_v1.18.l_extend.mdf	TSIP driver FIT module Smart Configurator configuration settings file
FITDemos	Demo project folder
rxXXX_rsk_tsip_sample*[1]	Project showing how to write keys and use cryptographic APIs
rx65n_2mb_rsk_tsip_aes_sample	AES cryptography project for RX65N
rx72n_ek_tsip_aes_sample	AES cryptography project for RX72N
rx_tsip_freertos_mbedtls_sample	TLS cooperation function project

File/Directory (Bold) Name			Description
tool			
	renesas_secure_flash_programmer		Tool for encrypting keys and user programs (includes source code)
		Renesas Secure Flash Programmer.exe	Tool for encrypting keys and user programs

Note: 1. "rxXXX" represents the RX group names of supported RSK boards.
Supported RSK boards: RX231, RX26T, RX65N-2MB, RX671, RX72M, RX72N, RX72T

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 rev1.18

Compiler version: Renesas Electronics C/C++ Compiler Package for RX Family V3.05.00
(integrated development environment default settings with “-lang = c99” option added)

GCC for Renesas RX 8.3.0.202204
(integrated development environment default settings with “-std = gnu99” option added)

IAR C/C++ Compiler for Renesas RX version 4.20.01
(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	56,506 bytes	55,484 bytes	58,494 bytes
	RAM	804 bytes	804 bytes	804 bytes
	Stack	184 bytes	—	164 bytes
TSIP	ROM	421,545 bytes	403,979 bytes	417,179 bytes
	RAM	7,428 bytes	7,428 bytes	7,428 bytes
	Stack	1,684 bytes	—	1,376 bytes

1.6 Sections

The TSIP driver uses the default sections.

If Renesas Secure Flash Programmer is used to generate key files, the sections C_FIRMWARE_UPDATE_CONTROL_BLOCK and C_FIRMWARE_UPDATE_CONTROL_BLOCK_MIRROR are used. Settings for these sections are configured automatically if CC-RX is set as the compiler and the TSIP driver is added to the project using Smart Configurator. If changes are required, edit the section setting.

When using the secure boot functionality, the sections BSECURE_BOOT*, PSECURE_BOOT, 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 1.3 Performance of Common APIs

API	Performance (Unit: Cycle)
R_TSIP_Open	7,400,000
R_TSIP_Close	450
R_TSIP_GetVersion	30
R_TSIP_GenerateAes128KeyIndex	4,000
R_TSIP_GenerateAes256KeyIndex	4,400
R_TSIP_GenerateAes128RandomKeyIndex	2,300
R_TSIP_GenerateAes256RandomKeyIndex	3,100
R_TSIP_GenerateRandomNumber	940
R_TSIP_GenerateUpdateKeyRingKeyIndex	4,400
R_TSIP_UpdaeteAes128KeyIndex	3,600
R_TSIP_UpdaeteAes256KeyIndex	3,900

Table 1.4 Firmware Verification Performance

API	Performance (Unit: Cycle)		
	2 KB Processing	4 KB Processing	6 KB Processing
R_TSIP_VerifyFirmwareMAC	13,000	24,000	35,000

Table 1.5 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	620	800	970
R_TSIP_Aes128EcbEncryptFinal	560	560	560
R_TSIP_Aes128EcbDecryptInit	1,400	1,400	1,400
R_TSIP_Aes128EcbDecryptUpdate	740	920	1,100
R_TSIP_Aes128EcbDecryptFinal	580	580	580
R_TSIP_Aes256EcbEncryptInit	1,700	1,700	1,700
R_TSIP_Aes256EcbEncryptUpdate	660	910	1,200
R_TSIP_Aes256EcbEncryptFinal	570	570	570
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	680	860	1,100
R_TSIP_Aes128CbcEncryptFinal	590	590	590
R_TSIP_Aes128CbcDecryptInit	1,400	1,400	1,400
R_TSIP_Aes128CbcDecryptUpdate	790	970	1,200
R_TSIP_Aes128CbcDecryptFinal	600	600	600
R_TSIP_Aes256CbcEncryptInit	1,700	1,700	1,700
R_TSIP_Aes256CbcEncryptUpdate	710	960	1,300
R_TSIP_Aes256CbcEncryptFinal	590	590	590
R_TSIP_Aes256CbcDecryptInit	1,700	1,700	1,700
R_TSIP_Aes256CbcDecryptUpdate	860	1,100	1,400
R_TSIP_Aes256CbcDecryptFinal	600	600	600

Table 1.6 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	4,100
R_TSIP_Aes256GcmEncryptFinal	1,400	1,400	1,400
R_TSIP_Aes256GcmDecryptInit	6,200	6,200	6,200
R_TSIP_Aes256GcmDecryptUpdate	2,600	2,700	2,800
R_TSIP_Aes256GcmDecryptFinal	2,200	2,200	2,200

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

Table 1.7 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	1,500	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	1,800	2,000	2,300
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 1.8 Performance of AES-CMAC

API	Performance (Unit: Cycle)		
	48-Byte Processing	64-Byte Processing	80-Byte Processing
R_TSIP_Aes128CmacGenerateInit	920	920	920
R_TSIP_Aes128CmacGenerateUpdate	820	900	990
R_TSIP_Aes128CmacGenerateFinal	1,100	1,100	1,100
R_TSIP_Aes128CmacVerifyInit	910	920	920
R_TSIP_Aes128CmacVerifyUpdate	820	910	990
R_TSIP_Aes128CmacVerifyFinal	1,800	1,800	1,800
R_TSIP_Aes256CmacGenerateInit	1,300	1,300	1,300
R_TSIP_Aes256CmacGenerateUpdate	880	1,100	1,200
R_TSIP_Aes256CmacGenerateFinal	1,200	1,200	1,200
R_TSIP_Aes256CmacVerifyInit	1,300	1,300	1,300
R_TSIP_Aes256CmacVerifyUpdate	880	1,100	1,200
R_TSIP_Aes256CmacVerifyFinal	1,900	1,900	1,900

Table 1.9 Performance of AES Key Wrap

API	Performance (Unit: Cycle)	
	Wrap Target Key AES-128	Wrap Target Key AES-256
R_TSIP_Aes128KeyWrap	9,600	16,000
R_TSIP_Aes256KeyWrap	11,000	17,000
R_TSIP_Aes128KeyUnwrap	12,000	18,000
R_TSIP_Aes256KeyUnwrap	13,000	19,000

1.7.2 RX23W

Table 1.10 Performance of Common APIs

API	Performance (Unit: Cycle)
R_TSIP_Open	7,400,000
R_TSIP_Close	670
R_TSIP_GetVersion	40
R_TSIP_GenerateAes128KeyIndex	4,400
R_TSIP_GenerateAes256KeyIndex	4,700
R_TSIP_GenerateAes128RandomKeyIndex	2,500
R_TSIP_GenerateAes256RandomKeyIndex	3,400
R_TSIP_GenerateRandomNumber	1,100
R_TSIP_GenerateUpdateKeyRingKeyIndex	4,700
R_TSIP_UpdaeteAes128KeyIndex	3,900
R_TSIP_UpdaeteAes256KeyIndex	4,200

Table 1.11 Firmware Verification Performance

API	Performance (Unit: Cycle)		
	2 KB Processing	4 KB Processing	6 KB Processing
R_TSIP_VerifyFirmwareMAC	13,000	24,000	35,000

Table 1.12 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	750	920	1,200
R_TSIP_Aes128EcbEncryptFinal	650	650	650
R_TSIP_Aes128EcbDecryptInit	1,600	1,600	1,600
R_TSIP_Aes128EcbDecryptUpdate	860	1,100	1,300
R_TSIP_Aes128EcbDecryptFinal	670	670	670
R_TSIP_Aes256EcbEncryptInit	1,900	1,900	1,900
R_TSIP_Aes256EcbEncryptUpdate	780	1,100	1,300
R_TSIP_Aes256EcbEncryptFinal	670	670	670
R_TSIP_Aes256EcbDecryptInit	1,900	1,900	1,900
R_TSIP_Aes256EcbDecryptUpdate	930	1,200	1,500
R_TSIP_Aes256EcbDecryptFinal	690	690	690
R_TSIP_Aes128CbcEncryptInit	1,600	1,600	1,600
R_TSIP_Aes128CbcEncryptUpdate	820	1,000	1,200
R_TSIP_Aes128CbcEncryptFinal	690	690	690
R_TSIP_Aes128CbcDecryptInit	1,600	1,600	1,600
R_TSIP_Aes128CbcDecryptUpdate	930	1,200	1,300
R_TSIP_Aes128CbcDecryptFinal	700	700	700
R_TSIP_Aes256CbcEncryptInit	1,900	1,900	1,900
R_TSIP_Aes256CbcEncryptUpdate	860	1,100	1,400
R_TSIP_Aes256CbcEncryptFinal	700	700	700
R_TSIP_Aes256CbcDecryptInit	1,900	2,000	2,000
R_TSIP_Aes256CbcDecryptUpdate	1,000	1,300	1,500
R_TSIP_Aes256CbcDecryptFinal	720	720	720

Table 1.13 Performance of AES-GCM

API	Performance (Unit: Cycle)		
	48-Byte Processing	64-Byte Processing	80-Byte Processing
R_TSIP_Aes128GcmEncryptInit	6,300	6,300	6,300
R_TSIP_Aes128GcmEncryptUpdate	3,400	4,000	4,500
R_TSIP_Aes128GcmEncryptFinal	1,500	1,500	1,500
R_TSIP_Aes128GcmDecryptInit	6,300	6,300	6,300
R_TSIP_Aes128GcmDecryptUpdate	2,900	3,000	3,100
R_TSIP_Aes128GcmDecryptFinal	2,400	2,400	2,400
R_TSIP_Aes256GcmEncryptInit	7,000	7,000	7,000
R_TSIP_Aes256GcmEncryptUpdate	3,500	4,100	4,700
R_TSIP_Aes256GcmEncryptFinal	1,600	1,600	1,600
R_TSIP_Aes256GcmDecryptInit	7,000	7,000	7,000
R_TSIP_Aes256GcmDecryptUpdate	3,000	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 1.14 Performance of AES-CCM

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

API	Performance (Unit: Cycle)		
	48-Byte Processing	64-Byte Processing	80-Byte Processing
R_TSIP_Aes128CmacGenerateInit	1,100	1,100	1,100
R_TSIP_Aes128CmacGenerateUpdate	960	1,100	1,200
R_TSIP_Aes128CmacGenerateFinal	1,300	1,300	1,300
R_TSIP_Aes128CmacVerifyInit	1,100	1,100	1,100
R_TSIP_Aes128CmacVerifyUpdate	950	1,100	1,200
R_TSIP_Aes128CmacVerifyFinal	2,100	2,100	2,100
R_TSIP_Aes256CmacGenerateInit	1,400	1,400	1,400
R_TSIP_Aes256CmacGenerateUpdate	1,100	1,200	1,300
R_TSIP_Aes256CmacGenerateFinal	1,400	1,400	1,400
R_TSIP_Aes256CmacVerifyInit	1,400	1,400	1,400
R_TSIP_Aes256CmacVerifyUpdate	1,100	1,200	1,300
R_TSIP_Aes256CmacVerifyFinal	2,200	2,200	2,200

Table 1.16 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	12,000	18,000
R_TSIP_Aes128KeyUnwrap	14,000	20,000
R_TSIP_Aes256KeyUnwrap	15,000	21,000

1.7.3 RX26T

Table 1.17 Performance of Common APIs

API	Performance (Unit: Cycle)
R_TSIP_Open	7,400,000
R_TSIP_Close	280
R_TSIP_GetVersion	20
R_TSIP_GenerateAes128KeyIndex	3,900
R_TSIP_GenerateAes256KeyIndex	4,300
R_TSIP_GenerateAes128RandomKeyIndex	2,200
R_TSIP_GenerateAes256RandomKeyIndex	3,000
R_TSIP_GenerateRandomNumber	900
R_TSIP_GenerateUpdateKeyRingKeyIndex	4,300
R_TSIP_UpdaeteAes128KeyIndex	3,500
R_TSIP_UpdaeteAes256KeyIndex	3,800

Table 1.18 Firmware Verification Performance

API	Performance (Unit: Cycle)		
	2 KB Processing	4 KB Processing	6 KB Processing
R_TSIP_VerifyFirmwareMAC	12,000	23,000	34,000

Table 1.19 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	740	910
R_TSIP_Aes128EcbEncryptFinal	500	500	500
R_TSIP_Aes128EcbDecryptInit	1,300	1,300	1,300
R_TSIP_Aes128EcbDecryptUpdate	660	850	1,000
R_TSIP_Aes128EcbDecryptFinal	510	510	510
R_TSIP_Aes256EcbEncryptInit	1,600	1,600	1,600
R_TSIP_Aes256EcbEncryptUpdate	600	840	1,100
R_TSIP_Aes256EcbEncryptFinal	500	500	510
R_TSIP_Aes256EcbDecryptInit	1,600	1,600	1,600
R_TSIP_Aes256EcbDecryptUpdate	740	980	1,200
R_TSIP_Aes256EcbDecryptFinal	520	520	520
R_TSIP_Aes128CbcEncryptInit	1,300	1,300	1,300
R_TSIP_Aes128CbcEncryptUpdate	600	790	960
R_TSIP_Aes128CbcEncryptFinal	520	520	520
R_TSIP_Aes128CbcDecryptInit	1,300	1,300	1,300
R_TSIP_Aes128CbcDecryptUpdate	710	890	1,100
R_TSIP_Aes128CbcDecryptFinal	530	530	530
R_TSIP_Aes256CbcEncryptInit	1,600	1,600	1,600
R_TSIP_Aes256CbcEncryptUpdate	640	890	1,100
R_TSIP_Aes256CbcEncryptFinal	530	530	530
R_TSIP_Aes256CbcDecryptInit	1,600	1,600	1,600
R_TSIP_Aes256CbcDecryptUpdate	780	1,000	1,300
R_TSIP_Aes256CbcDecryptFinal	540	540	540

Table 1.20 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	1,200	1,200	1,200
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 1.21 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	1,400	1,600	1,800
R_TSIP_Aes128CcmEncryptFinal	1,100	1,100	1,100
R_TSIP_Aes128CcmDecryptInit	2,200	2,200	2,200
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	2,100
R_TSIP_Aes256CcmEncryptFinal	1,200	1,200	1,200
R_TSIP_Aes256CcmDecryptInit	2,800	2,800	2,800
R_TSIP_Aes256CcmDecryptUpdate	1,600	1,800	2,000
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 1.22 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	900
R_TSIP_Aes128CmacGenerateFinal	1,000	1,000	1,000
R_TSIP_Aes128CmacVerifyInit	870	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	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	910	1,000
R_TSIP_Aes256CmacVerifyFinal	1,800	1,800	1,800

Table 1.23 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 1.24 Performance of Common APIs

API	Performance (Unit: Cycle)
R_TSIP_Open	7,400,000
R_TSIP_Close	290
R_TSIP_GetVersion	22
R_TSIP_GenerateAes128KeyIndex	4,000
R_TSIP_GenerateAes256KeyIndex	4,300
R_TSIP_GenerateAes128RandomKeyIndex	2,200
R_TSIP_GenerateAes256RandomKeyIndex	3,000
R_TSIP_GenerateRandomNumber	910
R_TSIP_GenerateUpdateKeyRingKeyIndex	4,300
R_TSIP_UpdateAes128KeyIndex	3,500
R_TSIP_UpdateAes256KeyIndex	3,900

Table 1.25 Firmware Verification Performance

API	Performance (Unit: Cycle)		
	2 KB Processing	4 KB Processing	6 KB Processing
R_TSIP_VerifyFirmwareMAC	12,000	24,000	35,000

Table 1.26 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	520	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	520	510	510
R_TSIP_Aes256EcbDecryptInit	1,600	1,600	1,600
R_TSIP_Aes256EcbDecryptUpdate	750	1,000	1,300
R_TSIP_Aes256EcbDecryptFinal	530	520	520
R_TSIP_Aes128CbcEncryptInit	1,400	1,400	1,400
R_TSIP_Aes128CbcEncryptUpdate	630	810	980
R_TSIP_Aes128CbcEncryptFinal	540	530	530
R_TSIP_Aes128CbcDecryptInit	1,400	1,400	1,400
R_TSIP_Aes128CbcDecryptUpdate	730	910	1,100
R_TSIP_Aes128CbcDecryptFinal	540	540	540
R_TSIP_Aes256CbcEncryptInit	1,700	1,700	1,700
R_TSIP_Aes256CbcEncryptUpdate	660	910	1,200
R_TSIP_Aes256CbcEncryptFinal	540	540	540
R_TSIP_Aes256CbcDecryptInit	1,700	1,700	1,700
R_TSIP_Aes256CbcDecryptUpdate	800	1,100	1,300
R_TSIP_Aes256CbcDecryptFinal	550	550	550

Table 1.27 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	2,700	3,100	3,600
R_TSIP_Aes128GcmEncryptFinal	1,300	1,300	1,300
R_TSIP_Aes128GcmDecryptInit	5,200	5,200	5,200
R_TSIP_Aes128GcmDecryptUpdate	2,300	2,300	2,400
R_TSIP_Aes128GcmDecryptFinal	2,100	2,100	2,100
R_TSIP_Aes256GcmEncryptInit	5,900	5,900	5,900
R_TSIP_Aes256GcmEncryptUpdate	2,800	3,300	3,800
R_TSIP_Aes256GcmEncryptFinal	1,300	1,300	1,300
R_TSIP_Aes256GcmDecryptInit	5,900	5,900	5,900
R_TSIP_Aes256GcmDecryptUpdate	2,400	2,500	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 1.28 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	1,700	1,900
R_TSIP_Aes128CcmEncryptFinal	1,200	1,200	1,200
R_TSIP_Aes128CcmDecryptInit	2,300	2,300	2,300
R_TSIP_Aes128CcmDecryptUpdate	1,400	1,600	1,800
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	1,900	2,100
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 1.29 Performance of AES-CMAC

API	Performance (Unit: Cycle)		
	48-Byte Processing	64-Byte Processing	80-Byte Processing
R_TSIP_Aes128CmacGenerateInit	890	880	880
R_TSIP_Aes128CmacGenerateUpdate	730	810	900
R_TSIP_Aes128CmacGenerateFinal	1,100	1,100	1,100
R_TSIP_Aes128CmacVerifyInit	880	880	880
R_TSIP_Aes128CmacVerifyUpdate	720	810	900
R_TSIP_Aes128CmacVerifyFinal	1,800	1,800	1,800
R_TSIP_Aes256CmacGenerateInit	1,200	1,200	1,200
R_TSIP_Aes256CmacGenerateUpdate	800	930	1,100
R_TSIP_Aes256CmacGenerateFinal	1,200	1,200	1,200
R_TSIP_Aes256CmacVerifyInit	1,200	1,200	1,200
R_TSIP_Aes256CmacVerifyUpdate	800	920	1,100
R_TSIP_Aes256CmacVerifyFinal	1,800	1,800	1,800

Table 1.30 Performance of AES Key Wrap

API	Performance (Unit: Cycle)	
	Wrap Target Key AES-128	Wrap Target Key AES-256
R_TSIP_Aes128KeyWrap	9,400	16,000
R_TSIP_Aes256KeyWrap	11,000	17,000
R_TSIP_Aes128KeyUnwrap	12,000	18,000
R_TSIP_Aes256KeyUnwrap	13,000	19,000

1.7.5 RX65N

Table 1.31 Performance of Common APIs

API	Performance (Unit: Cycle)
R_TSIP_Open	5,700,000
R_TSIP_Close	460
R_TSIP_GetVersion	30
R_TSIP_GenerateAes128KeyIndex	2,700
R_TSIP_GenerateAes256KeyIndex	2,800
R_TSIP_GenerateAes128RandomKeyIndex	1,500
R_TSIP_GenerateAes256RandomKeyIndex	2,100
R_TSIP_GenerateRandomNumber	670
R_TSIP_GenerateUpdateKeyRingKeyIndex	2,800
R_TSIP_UpdateAes128KeyIndex	2,300
R_TSIP_UpdateAes256KeyIndex	2,400

Table 1.32 Firmware Verification Performance

API	Performance (Unit: Cycle)		
	8 KB Processing	16 KB Processing	24 KB Processing
R_TSIP_VerifyFirmwareMAC	22,000	42,000	63,000

Table 1.33 Performance of AES

API	Performance (Unit: Cycle)		
	16-Byte Processing	48-Byte Processing	80-Byte Processing
R_TSIP_Aes128EcbEncryptInit	1,700	1,700	1,700
R_TSIP_Aes128EcbEncryptUpdate	520	660	840
R_TSIP_Aes128EcbEncryptFinal	450	450	450
R_TSIP_Aes128EcbDecryptInit	1,700	1,700	1,700
R_TSIP_Aes128EcbDecryptUpdate	590	730	910
R_TSIP_Aes128EcbDecryptFinal	460	460	460
R_TSIP_Aes256EcbEncryptInit	1,800	1,800	1,800
R_TSIP_Aes256EcbEncryptUpdate	540	690	870
R_TSIP_Aes256EcbEncryptFinal	440	440	440
R_TSIP_Aes256EcbDecryptInit	1,800	1,800	1,800
R_TSIP_Aes256EcbDecryptUpdate	610	750	930
R_TSIP_Aes256EcbDecryptFinal	470	470	470
R_TSIP_Aes128CbcEncryptInit	1,700	1,700	1,700
R_TSIP_Aes128CbcEncryptUpdate	590	730	900
R_TSIP_Aes128CbcEncryptFinal	480	480	480
R_TSIP_Aes128CbcDecryptInit	1,700	1,700	1,700
R_TSIP_Aes128CbcDecryptUpdate	660	790	970
R_TSIP_Aes128CbcDecryptFinal	490	500	500
R_TSIP_Aes256CbcEncryptInit	1,900	1,900	1,900
R_TSIP_Aes256CbcEncryptUpdate	590	740	920
R_TSIP_Aes256CbcEncryptFinal	480	480	480
R_TSIP_Aes256CbcDecryptInit	1,900	1,900	1,900
R_TSIP_Aes256CbcDecryptUpdate	680	820	1,000
R_TSIP_Aes256CbcDecryptFinal	490	490	490

Table 1.34 Performance of AES-GCM

API	Performance (Unit: Cycle)		
	48-Byte Processing	64-Byte Processing	80-Byte Processing
R_TSIP_Aes128GcmEncryptInit	5,600	5,600	5,600
R_TSIP_Aes128GcmEncryptUpdate	2,100	2,200	2,300
R_TSIP_Aes128GcmEncryptFinal	1,400	1,400	1,400
R_TSIP_Aes128GcmDecryptInit	5,500	5,500	5,500
R_TSIP_Aes128GcmDecryptUpdate	2,100	2,200	2,300
R_TSIP_Aes128GcmDecryptFinal	2,200	2,200	2,200
R_TSIP_Aes256GcmEncryptInit	5,500	5,500	5,500
R_TSIP_Aes256GcmEncryptUpdate	2,200	2,300	2,400
R_TSIP_Aes256GcmEncryptFinal	1,100	1,100	1,100
R_TSIP_Aes256GcmDecryptInit	5,500	5,500	5,500
R_TSIP_Aes256GcmDecryptUpdate	2,200	2,300	2,300
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 1.35 Performance of AES-CCM

API	Performance (Unit: Cycle)		
	48-Byte Processing	64-Byte Processing	80-Byte Processing
R_TSIP_Aes128CcmEncryptInit	3,100	3,100	3,100
R_TSIP_Aes128CcmEncryptUpdate	1,200	1,300	1,400
R_TSIP_Aes128CcmEncryptFinal	940	940	940
R_TSIP_Aes128CcmDecryptInit	3,200	3,200	3,200
R_TSIP_Aes128CcmDecryptUpdate	1,100	1,200	1,300
R_TSIP_Aes128CcmDecryptFinal	2,000	2,000	2,000
R_TSIP_Aes256CcmEncryptInit	2,400	2,400	2,400
R_TSIP_Aes256CcmEncryptUpdate	1,200	1,300	1,400
R_TSIP_Aes256CcmEncryptFinal	990	990	990
R_TSIP_Aes256CcmDecryptInit	2,400	2,400	2,400
R_TSIP_Aes256CcmDecryptUpdate	1,100	1,200	1,300
R_TSIP_Aes256CcmDecryptFinal	2,100	2,100	2,100

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

Table 1.36 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	670	720	760
R_TSIP_Aes128CmacGenerateFinal	800	800	800
R_TSIP_Aes128CmacVerifyInit	1,200	1,200	1,200
R_TSIP_Aes128CmacVerifyUpdate	680	720	770
R_TSIP_Aes128CmacVerifyFinal	1,700	1,700	1,700
R_TSIP_Aes256CmacGenerateInit	1,300	1,300	1,300
R_TSIP_Aes256CmacGenerateUpdate	720	760	810
R_TSIP_Aes256CmacGenerateFinal	830	830	830
R_TSIP_Aes256CmacVerifyInit	1,300	1,300	1,300
R_TSIP_Aes256CmacVerifyUpdate	710	750	810
R_TSIP_Aes256CmacVerifyFinal	1,700	1,700	1,700

Table 1.37 Performance of AES Key Wrap

API	Performance (Unit: Cycle)	
	Wrap Target Key AES-128	Wrap Target Key AES-256
R_TSIP_Aes128KeyWrap	8,300	13,000
R_TSIP_Aes256KeyWrap	8,400	14,000
R_TSIP_Aes128KeyUnwrap	9,300	14,000
R_TSIP_Aes256KeyUnwrap	9,500	15,000

Table 1.38 Performance of Common APIs (TDES Key Index Generation)

API	Performance (Unit: Cycle)
R_TSIP_GenerateTdesKeyIndex	2,800
R_TSIP_GenerateTdesRandomKeyIndex	2,100
R_TSIP_UpdateTdesKeyIndex	2,400

Table 1.39 Performance of TDES

API	Performance (Unit: Cycle)		
	16-Byte Processing	48-Byte Processing	80-Byte Processing
R_TSIP_TdesEcbEncryptInit	1,100	1,100	1,100
R_TSIP_TdesEcbEncryptUpdate	560	800	1,100
R_TSIP_TdesEcbEncryptFinal	450	450	450
R_TSIP_TdesEcbDecryptInit	1,100	1,100	1,100
R_TSIP_TdesEcbDecryptUpdate	590	830	1,100
R_TSIP_TdesEcbDecryptFinal	470	470	470
R_TSIP_TdesCbcEncryptInit	1,200	1,200	1,200
R_TSIP_TdesCbcEncryptUpdate	630	870	1,200
R_TSIP_TdesCbcEncryptFinal	480	480	480
R_TSIP_TdesCbcDecryptInit	1,200	1,200	1,200
R_TSIP_TdesCbcDecryptUpdate	650	900	1,200
R_TSIP_TdesCbcDecryptFinal	490	490	490

Table 1.40 Performance of Common APIs (ARC4 Key Index Generation)

API	Performance (Unit: Cycle)
R_TSIP_GenerateArc4KeyIndex	4,600
R_TSIP_GenerateArc4RandomKeyIndex	11,000
R_TSIP_UpdateArc4KeyIndex	4,200

Table 1.41 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	490	630	810
R_TSIP_Arc4EncryptFinal	330	330	330
R_TSIP_Arc4DecryptInit	2,100	2,100	2,100
R_TSIP_Arc4DecryptUpdate	490	630	810
R_TSIP_Arc4DecryptFinal	320	330	330

Table 1.42 Performance of Common APIs (RSA Key Index Generation)

API	Performance (Unit: Cycle)
R_TSIP_GenerateRsa1024PublicKeyIndex	38,000
R_TSIP_GenerateRsa1024PrivateKeyIndex	39,000
R_TSIP_GenerateRsa2048PublicKeyIndex	140,000
R_TSIP_GenerateRsa2048PrivateKeyIndex	140,000
R_TSIP_GenerateRsa1024RandomKeyIndex*[1]	75,000,000
R_TSIP_GenerateRsa2048RandomKeyIndex*[1]	540,000,000
R_TSIP_UpdateRsa1024PublicKeyIndex	38,000
R_TSIP_UpdateRsa1024PrivateKeyIndex	39,000
R_TSIP_UpdateRsa2048PublicKeyIndex	140,000
R_TSIP_UpdateRsa2048PrivateKeyIndex	140,000

Note: 1. Average value over 10 runs.

Table 1.43 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	18,000	19,000	20,000
R_TSIP_RsassaPkcs2048SignatureGenerate	27,000,000	27,000,000	27,000,000
R_TSIP_RsassaPkcs2048SignatureVerification	140,000	140,000	140,000

Table 1.44 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	18,000	19,000	20,000
R_TSIP_RsassaPkcs2048SignatureGenerate	27,000,000	27,000,000	27,000,000
R_TSIP_RsassaPkcs2048SignatureVerification	140,000	140,000	140,000

Table 1.45 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	18,000	19,000	19,000
R_TSIP_RsassaPkcs2048SignatureGenerate	27,000,000	27,000,000	27,000,000
R_TSIP_RsassaPkcs2048SignatureVerification	140,000	140,000	140,000

Table 1.46 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	23,000	17,000
R_TSIP_RsaesPkcs1024Decrypt	1,300,000	1,300,000

Table 1.47 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	27,000,000	27,000,000

Table 1.48 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 1.49 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 1.50 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	1,500	1,700	1,900
R_TSIP_Md5Final	780	780	780

Table 1.51 Performance of Common APIs (HMAC Key Index Generation)

API	Performance (Unit: Cycle)
R_TSIP_GenerateSha1HmacKeyIndex	3,000
R_TSIP_GenerateSha256HmacKeyIndex	3,000
R_TSIP_UpdateSha1HmacKeyIndex	2,700
R_TSIP_UpdateSha256HmacKeyIndex	2,700

Table 1.52 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	980	1,300	1,500
R_TSIP_Sha1HmacGenerateFinal	2,000	2,000	2,000
R_TSIP_Sha1HmacVerifyInit	1,400	1,400	1,400
R_TSIP_Sha1HmacVerifyUpdate	980	1,300	1,500
R_TSIP_Sha1HmacVerifyFinal	3,700	3,700	3,700

Table 1.53 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	920	1,200	1,400
R_TSIP_Sha256HmacGenerateFinal	2,000	2,000	2,000
R_TSIP_Sha256HmacVerifyInit	1,800	1,800	1,800
R_TSIP_Sha256HmacVerifyUpdate	920	1,200	1,400
R_TSIP_Sha256HmacVerifyFinal	3,700	3,700	3,700

Table 1.54 Performance of Common APIs (ECC Key Index Generation)

API	Performance (Unit: Cycle)
R_TSIP_GenerateEccP192PublicKeyIndex	3,300
R_TSIP_GenerateEccP224PublicKeyIndex	3,300
R_TSIP_GenerateEccP256PublicKeyIndex	3,300
R_TSIP_GenerateEccP384PublicKeyIndex	3,400
R_TSIP_GenerateEccP192PrivateKeyIndex	3,000
R_TSIP_GenerateEccP224PrivateKeyIndex	3,000
R_TSIP_GenerateEccP256PrivateKeyIndex	3,000
R_TSIP_GenerateEccP384PrivateKeyIndex	2,900
R_TSIP_GenerateEccP192RandomKeyIndex*1	150,000
R_TSIP_GenerateEccP224RandomKeyIndex*1	160,000
R_TSIP_GenerateEccP256RandomKeyIndex*1	160,000
R_TSIP_GenerateEccP384RandomKeyIndex*1	1,100,000
R_TSIP_UpdateEccP192PublicKeyIndex	3,000
R_TSIP_UpdateEccP224PublicKeyIndex	3,000
R_TSIP_UpdateEccP256PublicKeyIndex	3,000
R_TSIP_UpdateEccP384PublicKeyIndex	3,100
R_TSIP_UpdateEccP192PrivateKeyIndex	2,700
R_TSIP_UpdateEccP224PrivateKeyIndex	2,700
R_TSIP_UpdateEccP256PrivateKeyIndex	2,700
R_TSIP_UpdateEccP384PrivateKeyIndex	2,600

Note: 1. Average value over 10 runs.

Table 1.55 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	180,000	180,000
R_TSIP_EcdsaP224SignatureGenerate	180,000	180,000	180,000
R_TSIP_EcdsaP256SignatureGenerate	180,000	190,000	190,000
R_TSIP_EcdsaP384SignatureGenerate*1	1,200,000		
R_TSIP_EcdsaP192SignatureVerification	330,000	340,000	340,000
R_TSIP_EcdsaP224SignatureVerification	360,000	360,000	360,000
R_TSIP_EcdsaP256SignatureVerification	360,000	360,000	360,000
R_TSIP_EcdsaP384SignatureVerification*1	2,300,000		

Note: 1. Does not include SHA384 calculation.

Table 1.56 Key Exchange Performance

API	Performance (Unit: Cycle)
R_TSIP_EcdhP256Init	60
R_TSIP_EcdhP256ReadPublicKey	360,000
R_TSIP_EcdhP256MakePublicKey	340,000
R_TSIP_EcdhP256CalculateSharedSecretIndex	380,000
R_TSIP_EcdhP256KeyDerivation	3,800
R_TSIP_EcdheP512KeyAgreement	3,400,000
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 1.57 Performance of Common APIs

API	Performance (Unit: Cycle)
R_TSIP_Open	5,400,000
R_TSIP_Close	310
R_TSIP_GetVersion	22
R_TSIP_GenerateAes128KeyIndex	2,100
R_TSIP_GenerateAes256KeyIndex	2,200
R_TSIP_GenerateAes128RandomKeyIndex	1,200
R_TSIP_GenerateAes256RandomKeyIndex	1,700
R_TSIP_GenerateRandomNumber	540
R_TSIP_GenerateUpdateKeyRingKeyIndex	2,200
R_TSIP_UpdateAes128KeyIndex	1,800
R_TSIP_UpdateAes256KeyIndex	2,000

Table 1.58 Firmware Verification Performance

API	Performance (Unit: Cycle)		
	8 KB Processing	16 KB Processing	24 KB Processing
R_TSIP_VerifyFirmwareMAC	17,000	34,000	50,000

Table 1.59 Performance of AES

API	Performance (Unit: Cycle)		
	16-Byte Processing	48-Byte Processing	80-Byte Processing
R_TSIP_Aes128EcbEncryptInit	1,300	1,200	1,200
R_TSIP_Aes128EcbEncryptUpdate	390	490	620
R_TSIP_Aes128EcbEncryptFinal	320	310	310
R_TSIP_Aes128EcbDecryptInit	1,300	1,300	1,300
R_TSIP_Aes128EcbDecryptUpdate	450	560	690
R_TSIP_Aes128EcbDecryptFinal	320	320	320
R_TSIP_Aes256EcbEncryptInit	1,400	1,400	1,400
R_TSIP_Aes256EcbEncryptUpdate	400	510	640
R_TSIP_Aes256EcbEncryptFinal	320	310	310
R_TSIP_Aes256EcbDecryptInit	1,400	1,400	1,400
R_TSIP_Aes256EcbDecryptUpdate	470	580	710
R_TSIP_Aes256EcbDecryptFinal	330	330	330
R_TSIP_Aes128CbcEncryptInit	1,300	1,300	1,300
R_TSIP_Aes128CbcEncryptUpdate	430	540	670
R_TSIP_Aes128CbcEncryptFinal	340	330	330
R_TSIP_Aes128CbcDecryptInit	1,300	1,300	1,300
R_TSIP_Aes128CbcDecryptUpdate	490	600	730
R_TSIP_Aes128CbcDecryptFinal	340	340	340
R_TSIP_Aes256CbcEncryptInit	1,400	1,400	1,400
R_TSIP_Aes256CbcEncryptUpdate	450	570	700
R_TSIP_Aes256CbcEncryptFinal	340	340	340
R_TSIP_Aes256CbcDecryptInit	1,400	1,400	1,400
R_TSIP_Aes256CbcDecryptUpdate	520	640	770
R_TSIP_Aes256CbcDecryptFinal	350	350	350

Table 1.60 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	1,700
R_TSIP_Aes128GcmEncryptFinal	950	940	940
R_TSIP_Aes128GcmDecryptInit	4,100	4,100	4,100
R_TSIP_Aes128GcmDecryptUpdate	1,600	1,600	1,700
R_TSIP_Aes128GcmDecryptFinal	1,500	1,500	1,500
R_TSIP_Aes256GcmEncryptInit	4,200	4,100	4,100
R_TSIP_Aes256GcmEncryptUpdate	1,600	1,700	1,800
R_TSIP_Aes256GcmEncryptFinal	830	820	820
R_TSIP_Aes256GcmDecryptInit	4,100	4,100	4,100
R_TSIP_Aes256GcmDecryptUpdate	1,600	1,700	1,700
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 1.61 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	870	950	1,100
R_TSIP_Aes128CcmEncryptFinal	760	750	750
R_TSIP_Aes128CcmDecryptInit	2,400	2,400	2,400
R_TSIP_Aes128CcmDecryptUpdate	810	870	950
R_TSIP_Aes128CcmDecryptFinal	1,500	1,500	1,500
R_TSIP_Aes256CcmEncryptInit	1,900	1,900	1,900
R_TSIP_Aes256CcmEncryptUpdate	940	1,100	1,200
R_TSIP_Aes256CcmEncryptFinal	770	770	770
R_TSIP_Aes256CcmDecryptInit	1,900	1,900	1,900
R_TSIP_Aes256CcmDecryptUpdate	850	930	1,100
R_TSIP_Aes256CcmDecryptFinal	1,500	1,500	1,500

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

Table 1.62 Performance of AES-CMAC

API	Performance (Unit: Cycle)		
	48-Byte Processing	64-Byte Processing	80-Byte Processing
R_TSIP_Aes128CmacGenerateInit	880	870	870
R_TSIP_Aes128CmacGenerateUpdate	490	520	560
R_TSIP_Aes128CmacGenerateFinal	630	620	620
R_TSIP_Aes128CmacVerifyInit	870	870	870
R_TSIP_Aes128CmacVerifyUpdate	490	530	570
R_TSIP_Aes128CmacVerifyFinal	1,300	1,300	1,300
R_TSIP_Aes256CmacGenerateInit	980	980	980
R_TSIP_Aes256CmacGenerateUpdate	520	550	600
R_TSIP_Aes256CmacGenerateFinal	650	630	630
R_TSIP_Aes256CmacVerifyInit	970	970	970
R_TSIP_Aes256CmacVerifyUpdate	510	550	600
R_TSIP_Aes256CmacVerifyFinal	1,300	1,300	1,300

Table 1.63 Performance of AES Key Wrap

API	Performance (Unit: Cycle)	
	Wrap Target Key AES-128	Wrap Target Key AES-256
R_TSIP_Aes128KeyWrap	6,400	10,000
R_TSIP_Aes256KeyWrap	6,600	11,000
R_TSIP_Aes128KeyUnwrap	7,200	11,000
R_TSIP_Aes256KeyUnwrap	7,400	12,000

Table 1.64 Performance of Common APIs (TDES Key Index Generation)

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

Table 1.65 Performance of TDES

API	Performance (Unit: Cycle)		
	16-Byte Processing	48-Byte Processing	80-Byte Processing
R_TSIP_TdesEcbEncryptInit	800	790	790
R_TSIP_TdesEcbEncryptUpdate	430	610	800
R_TSIP_TdesEcbEncryptFinal	320	300	300
R_TSIP_TdesEcbDecryptInit	800	800	800
R_TSIP_TdesEcbDecryptUpdate	450	640	830
R_TSIP_TdesEcbDecryptFinal	330	320	320
R_TSIP_TdesCbcEncryptInit	850	840	840
R_TSIP_TdesCbcEncryptUpdate	480	670	860
R_TSIP_TdesCbcEncryptFinal	320	320	320
R_TSIP_TdesCbcDecryptInit	850	850	850
R_TSIP_TdesCbcDecryptUpdate	500	700	890
R_TSIP_TdesCbcDecryptFinal	340	340	340

Table 1.66 Performance of Common APIs (ARC4 Key Index Generation)

API	Performance (Unit: Cycle)
R_TSIP_GenerateArc4KeyIndex	3,900
R_TSIP_GenerateArc4RandomKeyIndex	8,600
R_TSIP_UpdateArc4KeyIndex	3,700

Table 1.67 Performance of ARC4

API	Performance (Unit: Cycle)		
	16-Byte Processing	48-Byte Processing	80-Byte Processing
R_TSIP_Arc4EncryptInit	1,800	1,800	1,800
R_TSIP_Arc4EncryptUpdate	360	480	610
R_TSIP_Arc4EncryptFinal	230	230	230
R_TSIP_Arc4DecryptInit	1,800	1,800	1,800
R_TSIP_Arc4DecryptUpdate	360	480	610
R_TSIP_Arc4DecryptFinal	230	230	230

Table 1.68 Performance of Common APIs (RSA Key Index 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	67,000,000
R_TSIP_GenerateRsa2048RandomKeyIndex*1	360,000,000
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 1.69 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	16,000	18,000	18,000
R_TSIP_RsassaPkcs2048SignatureGenerate	27,000,000	27,000,000	27,000,000
R_TSIP_RsassaPkcs2048SignatureVerification	140,000	140,000	140,000

Table 1.70 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	18,000	18,000
R_TSIP_RsassaPkcs2048SignatureGenerate	27,000,000	27,000,000	27,000,000
R_TSIP_RsassaPkcs2048SignatureVerification	140,000	140,000	140,000

Table 1.71 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	16,000	17,000	18,000
R_TSIP_RsassaPkcs2048SignatureGenerate	27,000,000	27,000,000	27,000,000
R_TSIP_RsassaPkcs2048SignatureVerification	140,000	140,000	140,000

Table 1.72 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	1,300,000	1,300,000

Table 1.73 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	27,000,000	27,000,000

Table 1.74 Performance of HASH (SHA1)

API	Performance (Unit: Cycle)		
	128-Byte Processing	192-Byte Processing	256-Byte Processing
R_TSIP_Sha1Init	110	110	110
R_TSIP_Sha1Update	1,300	1,500	1,700
R_TSIP_Sha1Final	660	660	660

Table 1.75 Performance of HASH (SHA256)

API	Performance (Unit: Cycle)		
	128-Byte Processing	192-Byte Processing	256-Byte Processing
R_TSIP_Sha256Init	120	120	120
R_TSIP_Sha256Update	1,300	1,500	1,600
R_TSIP_Sha256Final	670	670	670

Table 1.76 Performance of HASH (MD5)

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

Table 1.77 Performance of Common APIs (HMAC Key Index Generation)

API	Performance (Unit: Cycle)
R_TSIP_GenerateSha1HmacKeyIndex	2,300
R_TSIP_GenerateSha256HmacKeyIndex	2,300
R_TSIP_UpdateSha1HmacKeyIndex	2,100
R_TSIP_UpdateSha256HmacKeyIndex	2,000

Table 1.78 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	1,100	1,300
R_TSIP_Sha1HmacGenerateFinal	1,600	1,600	1,600
R_TSIP_Sha1HmacVerifyInit	1,100	1,100	1,100
R_TSIP_Sha1HmacVerifyUpdate	800	1,100	1,300
R_TSIP_Sha1HmacVerifyFinal	2,800	2,800	2,800

Table 1.79 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	1,600
R_TSIP_Sha256HmacVerifyInit	1,300	1,300	1,300
R_TSIP_Sha256HmacVerifyUpdate	730	910	1,100
R_TSIP_Sha256HmacVerifyFinal	2,700	2,700	2,700

Table 1.80 Performance of Common APIs (ECC Key Index Generation)

API	Performance (Unit: Cycle)
R_TSIP_GenerateEccP192PublicKeyIndex	2,600
R_TSIP_GenerateEccP224PublicKeyIndex	2,600
R_TSIP_GenerateEccP256PublicKeyIndex	2,600
R_TSIP_GenerateEccP384PublicKeyIndex	2,800
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	150,000
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	2,300
R_TSIP_UpdateEccP384PublicKeyIndex	2,500
R_TSIP_UpdateEccP192PrivateKeyIndex	2,100
R_TSIP_UpdateEccP224PrivateKeyIndex	2,000
R_TSIP_UpdateEccP256PrivateKeyIndex	2,000
R_TSIP_UpdateEccP384PrivateKeyIndex	2,100

Note: 1. Average value over 10 runs.

Table 1.81 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	170,000	170,000
R_TSIP_EcdsaP224SignatureGenerate	170,000	170,000	170,000
R_TSIP_EcdsaP256SignatureGenerate	170,000	180,000	170,000
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	340,000	330,000
R_TSIP_EcdsaP384SignatureVerification*1	2,200,000		

Note: 1. Does not include SHA384 calculation.

Table 1.82 Key Exchange Performance

API	Performance (Unit: Cycle)
R_TSIP_EcdhP256Init	42
R_TSIP_EcdhP256ReadPublicKey	340,000
R_TSIP_EcdhP256MakePublicKey	310,000
R_TSIP_EcdhP256CalculateSharedSecretIndex	360,000
R_TSIP_EcdhP256KeyDerivation	3,000
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.

1.7.7 RX72M, RX72N

Table 1.83 Performance of Common APIs

API	Performance (Unit: Cycle)
R_TSIP_Open	6,300,000
R_TSIP_Close	310
R_TSIP_GetVersion	20
R_TSIP_GenerateAes128KeyIndex	2,200
R_TSIP_GenerateAes256KeyIndex	2,300
R_TSIP_GenerateAes128RandomKeyIndex	1,300
R_TSIP_GenerateAes256RandomKeyIndex	1,800
R_TSIP_GenerateRandomNumber	560
R_TSIP_GenerateUpdateKeyRingKeyIndex	2,300
R_TSIP_UpdateAes128KeyIndex	1,900
R_TSIP_UpdateAes256KeyIndex	2,100

Table 1.84 Firmware Verification Performance

API	Performance (Unit: Cycle)		
	8 KB Processing	16 KB Processing	24 KB Processing
R_TSIP_VerifyFirmwareMAC	19,000	38,000	56,000

Table 1.85 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	390	510	640
R_TSIP_Aes128EcbEncryptFinal	340	340	340
R_TSIP_Aes128EcbDecryptInit	1,300	1,300	1,300
R_TSIP_Aes128EcbDecryptUpdate	450	570	700
R_TSIP_Aes128EcbDecryptFinal	350	350	350
R_TSIP_Aes256EcbEncryptInit	1,400	1,400	1,400
R_TSIP_Aes256EcbEncryptUpdate	400	530	660
R_TSIP_Aes256EcbEncryptFinal	330	330	330
R_TSIP_Aes256EcbDecryptInit	1,400	1,400	1,400
R_TSIP_Aes256EcbDecryptUpdate	480	600	740
R_TSIP_Aes256EcbDecryptFinal	340	340	340
R_TSIP_Aes128CbcEncryptInit	1,400	1,400	1,400
R_TSIP_Aes128CbcEncryptUpdate	440	560	700
R_TSIP_Aes128CbcEncryptFinal	360	360	360
R_TSIP_Aes128CbcDecryptInit	1,400	1,400	1,400
R_TSIP_Aes128CbcDecryptUpdate	500	610	750
R_TSIP_Aes128CbcDecryptFinal	370	370	370
R_TSIP_Aes256CbcEncryptInit	1,500	1,500	1,500
R_TSIP_Aes256CbcEncryptUpdate	460	580	720
R_TSIP_Aes256CbcEncryptFinal	360	360	360
R_TSIP_Aes256CbcDecryptInit	1,500	1,500	1,500
R_TSIP_Aes256CbcDecryptUpdate	530	650	790
R_TSIP_Aes256CbcDecryptFinal	370	370	370

Table 1.86 Performance of AES-GCM

API	Performance (Unit: Cycle)		
	48-Byte Processing	64-Byte Processing	80-Byte Processing
R_TSIP_Aes128GcmEncryptInit	4,400	4,400	4,400
R_TSIP_Aes128GcmEncryptUpdate	1,600	1,700	1,800
R_TSIP_Aes128GcmEncryptFinal	1,100	1,100	1,100
R_TSIP_Aes128GcmDecryptInit	4,300	4,300	4,300
R_TSIP_Aes128GcmDecryptUpdate	1,600	1,700	1,800
R_TSIP_Aes128GcmDecryptFinal	1,700	1,700	1,700
R_TSIP_Aes256GcmEncryptInit	4,300	4,300	4,300
R_TSIP_Aes256GcmEncryptUpdate	1,600	1,700	1,800
R_TSIP_Aes256GcmEncryptFinal	860	860	860
R_TSIP_Aes256GcmDecryptInit	4,300	4,300	4,300
R_TSIP_Aes256GcmDecryptUpdate	1,700	1,700	1,800
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 1.87 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	900	970	1,100
R_TSIP_Aes128CcmEncryptFinal	750	750	750
R_TSIP_Aes128CcmDecryptInit	2,500	2,500	2,500
R_TSIP_Aes128CcmDecryptUpdate	820	900	980
R_TSIP_Aes128CcmDecryptFinal	1,500	1,500	1,500
R_TSIP_Aes256CcmEncryptInit	2,000	2,000	2,000
R_TSIP_Aes256CcmEncryptUpdate	960	1,100	1,200
R_TSIP_Aes256CcmEncryptFinal	800	800	800
R_TSIP_Aes256CcmDecryptInit	2,000	2,000	2,000
R_TSIP_Aes256CcmDecryptUpdate	860	950	1,100
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 1.88 Performance of AES-CMAC

API	Performance (Unit: Cycle)		
	48-Byte Processing	64-Byte Processing	80-Byte Processing
R_TSIP_Aes128CmacGenerateInit	920	910	920
R_TSIP_Aes128CmacGenerateUpdate	490	530	570
R_TSIP_Aes128CmacGenerateFinal	630	620	620
R_TSIP_Aes128CmacVerifyInit	910	920	920
R_TSIP_Aes128CmacVerifyUpdate	490	530	570
R_TSIP_Aes128CmacVerifyFinal	1,300	1,300	1,300
R_TSIP_Aes256CmacGenerateInit	1,100	1,100	1,100
R_TSIP_Aes256CmacGenerateUpdate	520	560	600
R_TSIP_Aes256CmacGenerateFinal	660	660	660
R_TSIP_Aes256CmacVerifyInit	1,100	1,100	1,100
R_TSIP_Aes256CmacVerifyUpdate	530	570	610
R_TSIP_Aes256CmacVerifyFinal	1,300	1,300	1,300

Table 1.89 Performance of AES Key Wrap

API	Performance (Unit: Cycle)	
	Wrap Target Key AES-128	Wrap Target Key AES-256
R_TSIP_Aes128KeyWrap	6,500	11,000
R_TSIP_Aes256KeyWrap	6,800	11,000
R_TSIP_Aes128KeyUnwrap	7,400	12,000
R_TSIP_Aes256KeyUnwrap	7,600	12,000

Table 1.90 Performance of Common APIs (TDES Key Index Generation)

API	Performance (Unit: Cycle)
R_TSIP_GenerateTdesKeyIndex	2,300
R_TSIP_GenerateTdesRandomKeyIndex	1,800
R_TSIP_UpdateTdesKeyIndex	2,100

Table 1.91 Performance of TDES

API	Performance (Unit: Cycle)		
	16-Byte Processing	48-Byte Processing	80-Byte Processing
R_TSIP_TdesEcbEncryptInit	820	820	820
R_TSIP_TdesEcbEncryptUpdate	440	640	840
R_TSIP_TdesEcbEncryptFinal	330	330	330
R_TSIP_TdesEcbDecryptInit	840	840	840
R_TSIP_TdesEcbDecryptUpdate	460	660	860
R_TSIP_TdesEcbDecryptFinal	340	340	340
R_TSIP_TdesCbcEncryptInit	880	880	880
R_TSIP_TdesCbcEncryptUpdate	490	690	890
R_TSIP_TdesCbcEncryptFinal	350	350	350
R_TSIP_TdesCbcDecryptInit	880	880	880
R_TSIP_TdesCbcDecryptUpdate	510	720	910
R_TSIP_TdesCbcDecryptFinal	370	370	370

Table 1.92 Performance of Common APIs (ARC4 Key Index Generation)

API	Performance (Unit: Cycle)
R_TSIP_GenerateArc4KeyIndex	4,000
R_TSIP_GenerateArc4RandomKeyIndex	9,200
R_TSIP_UpdateArc4KeyIndex	3,800

Table 1.93 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	370	490	620
R_TSIP_Arc4EncryptFinal	240	240	240
R_TSIP_Arc4DecryptInit	1,900	1,900	1,900
R_TSIP_Arc4DecryptUpdate	370	490	620
R_TSIP_Arc4DecryptFinal	240	230	230

Table 1.94 Performance of Common APIs (RSA Key Index 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	59,000,000
R_TSIP_GenerateRsa2048RandomKeyIndex*1	450,000,000
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 1.95 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	18,000	18,000
R_TSIP_RsassaPkcs2048SignatureGenerate	27,000,000	27,000,000	27,000,000
R_TSIP_RsassaPkcs2048SignatureVerification	140,000	140,000	140,000

Table 1.96 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	18,000	18,000
R_TSIP_RsassaPkcs2048SignatureGenerate	27,000,000	27,000,000	27,000,000
R_TSIP_RsassaPkcs2048SignatureVerification	140,000	140,000	140,000

Table 1.97 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	18,000	18,000
R_TSIP_RsassaPkcs2048SignatureGenerate	27,000,000	27,000,000	27,000,000
R_TSIP_RsassaPkcs2048SignatureVerification	140,000	140,000	140,000

Table 1.98 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	21,000	16,000
R_TSIP_RsaesPkcs1024Decrypt	1,300,000	1,300,000

Table 1.99 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	27,000,000	27,000,000

Table 1.100 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	1,300	1,500	1,700
R_TSIP_Sha1Final	670	670	670

Table 1.101 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	1,500	1,700
R_TSIP_Sha256Final	640	640	640

Table 1.102 Performance of HASH (MD5)

API	Performance (Unit: Cycle)		
	128-Byte Processing	192-Byte Processing	256-Byte Processing
R_TSIP_Md5Init	94	94	94
R_TSIP_Md5Update	1,200	1,400	1,500
R_TSIP_Md5Final	630	630	630

Table 1.103 Performance of Common APIs (HMAC Key Index Generation)

API	Performance (Unit: Cycle)
R_TSIP_GenerateSha1HmacKeyIndex	2,400
R_TSIP_GenerateSha256HmacKeyIndex	2,400
R_TSIP_UpdateSha1HmacKeyIndex	2,200
R_TSIP_UpdateSha256HmacKeyIndex	2,200

Table 1.104 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	800	1,100	1,300
R_TSIP_Sha1HmacGenerateFinal	1,700	1,700	1,700
R_TSIP_Sha1HmacVerifyInit	1,100	1,100	1,100
R_TSIP_Sha1HmacVerifyUpdate	810	1,100	1,300
R_TSIP_Sha1HmacVerifyFinal	2,800	2,800	2,800

Table 1.105 Performance of HMAC (SHA256)

API	Performance (Unit: Cycle)		
	128-Byte Processing	192-Byte Processing	256-Byte Processing
R_TSIP_Sha256HmacGenerateInit	1,400	1,400	1,400
R_TSIP_Sha256HmacGenerateUpdate	740	910	1,100
R_TSIP_Sha256HmacGenerateFinal	1,600	1,600	1,600
R_TSIP_Sha256HmacVerifyInit	1,400	1,400	1,400
R_TSIP_Sha256HmacVerifyUpdate	730	910	1,100
R_TSIP_Sha256HmacVerifyFinal	2,800	2,800	2,800

Table 1.106 Performance of Common APIs (ECC Key Index Generation)

API	Performance (Unit: Cycle)
R_TSIP_GenerateEccP192PublicKeyIndex	2,700
R_TSIP_GenerateEccP224PublicKeyIndex	2,700
R_TSIP_GenerateEccP256PublicKeyIndex	2,700
R_TSIP_GenerateEccP384PublicKeyIndex	2,900
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	150,000
R_TSIP_GenerateEccP256RandomKeyIndex*1	150,000
R_TSIP_GenerateEccP384RandomKeyIndex*1	1,100,000
R_TSIP_UpdateEccP192PublicKeyIndex	2,500
R_TSIP_UpdateEccP224PublicKeyIndex	2,400
R_TSIP_UpdateEccP256PublicKeyIndex	2,500
R_TSIP_UpdateEccP384PublicKeyIndex	2,600
R_TSIP_UpdateEccP192PrivateKeyIndex	2,100
R_TSIP_UpdateEccP224PrivateKeyIndex	2,200
R_TSIP_UpdateEccP256PrivateKeyIndex	2,100
R_TSIP_UpdateEccP384PrivateKeyIndex	2,200

Note: 1. Average value over 10 runs.

Table 1.107 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	170,000	170,000
R_TSIP_EcdsaP224SignatureGenerate	170,000	170,000	170,000
R_TSIP_EcdsaP256SignatureGenerate	170,000	170,000	170,000
R_TSIP_EcdsaP384SignatureGenerate*1	1,200,000		
R_TSIP_EcdsaP192SignatureVerification	310,000	310,000	310,000
R_TSIP_EcdsaP224SignatureVerification	330,000	330,000	340,000
R_TSIP_EcdsaP256SignatureVerification	330,000	330,000	340,000
R_TSIP_EcdsaP384SignatureVerification*1	2,100,000		

Note: 1. Does not include SHA384 calculation.

Table 1.108 Key Exchange Performance

API	Performance (Unit: Cycle)
R_TSIP_EcdhP256Init	42
R_TSIP_EcdhP256ReadPublicKey	340,000
R_TSIP_EcdhP256MakePublicKey	310,000
R_TSIP_EcdhP256CalculateSharedSecretIndex	360,000
R_TSIP_EcdhP256KeyDerivation	3,200
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.

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. 7.10 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)
```

[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.

In the binary version of TSIP driver, double size is set to 64 bit.

2.6 Data Structures

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

2.7 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 key index 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.
}e_tsip_err_t
```

2.8 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.

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 key index. 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 key index 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 key index 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 key index. [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.

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 key indexes wrapped using an HUK.

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 key index is not encrypted, so the plaintext public key can be extracted from it.

The first 128 bits of the provisioning key or update keyring 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 user key.

This application note explains the provisioning key and encrypted provisioning key using the key attached to the sample program. These key for mass production needs to be newly generated. An application note with these key details is available.

We will provide the product to customers who will be adopting or plan to adopt a Renesas microcontroller. Please contact your local Renesas Electronics sales office or distributor.

https://www.renesas.com/contact/

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.

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	[RSAES-PKCS1-V1_5 encrypt] XXX = 1024, 2048, 3072, 4096 [RSAES-PKCS1-V1_5 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.

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 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.

The sample program to execute firmware update and an application note with details of this function is available.

We will provide the product to customers who will be adopting or plan to adopt a Renesas microcontroller. Please contact your local Renesas Electronics sales office or distributor.

https://www.renesas.com/contact/

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

Table 4.4 DES Encryption/Decryption APIs

API	Description	TSIP-Lite	TSIP
R_TSIP_GenerateTdesKeyIndex	Generates a TDES key index.	—	✓
R_TSIP_UpdateTdesKeyIndex	Updates a TDES key index.	—	✓
R_TSIP_GenerateTdesRandomKeyIndex	Generates a TDES key from a random number and outputs the key index.	—	✓
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 key index.	—	✓
R_TSIP_UpdateArc4KeyIndex	Updates an ARC4 key index.	—	✓
R_TSIP_GenerateArc4RandomKeyIndex	Generates an ARC4 key from a random number and outputs the key index.	—	✓
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 key index.	—	✓
R_TSIP_GenerateRsa*XXX*PublicKeyIndex	Generates an RSA public key index.	—	✓
R_TSIP_UpdateRsa*XXX*PrivateKeyIndex	Updates an RSA secret key index.	—	✓
R_TSIP_UpdateRsa*XXX*PublicKeyIndex	Updates an RSA public key index.	—	✓
R_TSIP_GenerateRsa*XXX*RandomKeyIndex	Generates a public key corresponding to an RSA secret key from a random number and outputs the key index. 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_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 key index.	—	✓
R_TSIP_GenerateEccP*XXX*PrivateKeyIndex	Generates an ECC secret key index.	—	✓
R_TSIP_UpdateEccP*XXX*PublicKeyIndex	Updates an ECC public key index.	—	✓
R_TSIP_UpdateEccP*XXX*PrivateKeyIndex	Updates an ECC secret key index.	—	✓
R_TSIP_GenerateEccP*XXX*RandomKeyIndex	Generates a public key corresponding to an ECC secret key from a random number and outputs the key index.	—	✓
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 key index.	—	✓
R_TSIP_UpdateSha*XXX*HmacKeyIndex	Updates an SHA-HMAC key index.	—	✓
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.	—	✓

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_GenerateTlsRsaPublicKeyIndex	Generates an RSA public key index used in TLS cooperation.	—	✓
R_TSIP_UpdateTlsRsaPublicKeyIndex	Updates an RSA public key index used in TLS cooperation.	—	✓
R_TSIP_TlsRootCertificateVerification	Verifies a root CA certificate bundle.	—	✓
R_TSIP_TlsCertificateVerification	Verifies the signature of a server certificate or intermediate certificate.	—	✓
R_TSIP_TlsCertificateVerificationExtension	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_TlsGenerateMasterSecret	Generates an encrypted master secret.	—	✓
R_TSIP_TlsGenerateSessionKey	Outputs TLS communication keys.	—	✓
R_TSIP_TlsGenerateVerifyData	Generates a VerifyData message.	—	✓
R_TSIP_TlsServersEphemeralEcdhPublicKeyRetrieves	Verifies a server key exchange signature.	—	✓
R_TSIP_GenerateTlsP256EccKeyIndex	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_TlsGeneratePreMasterSecretWithEccP256Key	Generates an ECC encrypted pre-master secret.	—	✓
R_TSIP_TlsGenerateExtendedMasterSecret	Generates an encrypted extended master secret.	—	✓
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 key index.	—	✓
R_TSIP_Tls13GenerateHandshakeSecret	Generates a handshake secret key index.	—	✓
R_TSIP_Tls13GenerateServerHandshakeTrafficKey	Generates a server write key index and server finished key index.	—	✓
R_TSIP_Tls13ServerHandshakeVerification	Verifies finished information provided by the server.	—	✓
R_TSIP_Tls13GenerateClientHandshakeTrafficKey	Generates a client write key index and client finished key index.	—	✓
R_TSIP_Tls13GenerateMasterSecret	Generates a master secret key index.	—	✓
R_TSIP_Tls13GenerateApplicationTrafficKey	Generates an application traffic secret key index and an application traffic key index.	—	✓
R_TSIP_Tls13UpdateApplicationTrafficKey	Updates an application traffic secret key index and an application traffic key index.	—	✓
R_TSIP_Tls13GenerateResumptionMasterSecret	Generates a resumption master secret key index.	—	✓
R_TSIP_Tls13GeneratePreSharedKey	Generates a pre shared key index.	—	✓
R_TSIP_Tls13GeneratePskBinderKey	Generates a binder key index.	—	✓

API	Description	TSIP -Lite	TSIP
R_TSIP_Tls13GenerateResumptionHandshakeSecret	Generates a handshake secret key index for resumption.	—	✓
R_TSIP_Tls13Generate0RttApplicationWriteKey	Generates a client write key index 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 key index.	—	✓
R_TSIP_Tls13SVGenerateHandshakeSecret	Generates a handshake secret key index.	—	✓
R_TSIP_Tls13SVGenerateServerHandshakeTrafficKey	Generates a server write key index and server finished key index.	—	✓
R_TSIP_Tls13SVGenerateClientHandshakeTrafficKey	Generates a client write key index and client finished key index.	—	✓
R_TSIP_Tls13SVClientHandshakeVerification	Verifies finished information provided by the client.	—	✓
R_TSIP_Tls13SVGenerateMasterSecret	Generates a master secret key index.	—	✓
R_TSIP_Tls13SVGenerateApplicationTrafficKey	Generates an application traffic secret key index and application traffic key index.	—	✓
R_TSIP_Tls13SVUpdateApplicationTrafficKey	Updates an application traffic secret key index and application traffic key index.	—	✓
R_TSIP_Tls13SVGenerateResumptionMasterSecret	Generates a resumption master secret key index.	—	✓
R_TSIP_Tls13SVGeneratePreSharedKey	Generates a pre shared key index.	—	✓
R_TSIP_Tls13SVGeneratePskBinderKey	Generates a binder key index.	—	✓
R_TSIP_Tls13SVGenerateResumptionHandshakeSecret	Generates a handshake secret key index for resumption.	—	✓
R_TSIP_Tls13SVGenerate0RttApplicationWriteKey	Generates a client write key index 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.	—	✓
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_GenerateFirmwareMAC	Decrypts and generates the MAC for encrypted firmware.	✓	✓
R_TSIP_VerifyFirmwareMAC	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 keyring key index
key_index_2	Input	Update keyring key index

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.

Description

Enables use of TSIP functionality.

For key_index_1, input the "TLS cooperation RSA public key index" 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 "update keyring key index" generated by R_TSIP_GenerateUpdateKeyRingKeyIndex(). If the key update cooperation function is not used, 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	Provisioning key wrapped by the DLM server
iv	Input	Initialization vector when generating encrypted_key
encrypted_key	Input	Encrypted user key with MAC appended
key_index	Output	Update keyring 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 a key update keyring key index.

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

4.2.2 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	Provisioning key wrapped by the DLM server
iv	Input	Initialization vector when generating encrypted_key
encrypted_key	Input	Encrypted user key with MAC appended
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

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

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

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

4.2.3.2 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	User key encrypted using update keyring with MAC appended
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

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

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

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

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 key index (2) 256-bit AES key index

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 key index.

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

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 key index 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	Key index area

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 key index 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_Aes128EcbEncryptUpdate(
        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	Key index area

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 key index 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	Key index area
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 key index 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	Key index area
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 key index 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	Key index area
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 key index 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	Key index area
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 key index 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	Key index area
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 key index 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	Key index area
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 key index 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	Key index area
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 key index 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	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 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	Key index area

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 key index 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	Key index area

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 key index input

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	Provisioning key wrapped by the DLM server
iv	Input	Initialization vector when generating encrypted_key
encrypted_key	Input	Encrypted TDES user key with MAC appended
key_index	Output	TDES key index

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

For encrypted_key, input the data indicated in 7.3.2, DES, encrypted using the provisioning 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.

4.2.4.2 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	User key encrypted using update keyring with MAC appended
key_index	Output	TDES key index

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 key index.

For encrypted_key, input the data indicated in 7.3.2, DES, encrypted using the provisioning 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.

4.2.4.3 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	TDES key index

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 key index.

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 key index 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	TDES key index area

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 key index input

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	TDES key index area

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_KEY_SET:	Invalid key index 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_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	TDES key index area
ivec	Input	Initialization vector (8 bytes)

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_KEY_SET:	Invalid key index 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	TDES key index area
ivec	Input	Initialization vector (8 bytes)

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_KEY_SET:	Invalid key index 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	Provisioning key wrapped by the DLM server
iv	Input	Initialization vector when generating encrypted_key
encrypted_key	Input	Encrypted user key with MAC appended
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 key index.

Refer to 7.3.3, ARC4, for the format of the data encrypted using the provisioning key 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	User key encrypted using update keyring with MAC appended
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 key index.

Refer to 7.3.3, ARC4, for the format of the data encrypted using the key update keyring 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	Key index

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 key index.

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 key index 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	Key index area

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 key index 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	Key index area

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 key index input

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	Provisioning key wrapped by the DLM server
iv	Input	Initialization vector when generating encrypted_key
encrypted_key	Input	Encrypted RSA public key with MAC appended
key_index	Output	RSA public key index
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 index.

Refer to 7.3.4, RSA, for the format of the data encrypted using the provisioning key 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	Provisioning key wrapped by the DLM server
iv	Input	Initialization vector when generating encrypted_key
encrypted_key	Input	Encrypted RSA public key with MAC appended
key_index	Output	RSA secret key index (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 key index.

Refer to 7.3.4, RSA, for the format of the data encrypted using the provisioning key 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	Public key encrypted using update keyring with MAC appended
key_index	Output	RSA public key index
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 public key index.

Refer to 7.3.4, RSA, for the format of the data encrypted using the update keyring 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	Secret key encrypted using update keyring with MAC appended
key_index	Output	RSA secret key index (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 key index. Refer to 7.3.4, RSA, for the format of the data encrypted using the provisioning key 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 key indexes
public		RSA public key index
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 key index

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 key indexes 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 key indexes output by this API makes it possible to prevent dead copying of data. The public key index is generated in key_pair_index->public, and the secret key index 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 index output by R_TSIP_GenerateRsaXXXPublicKeyIndex(), and key_pair_index->private is utilized in the same manner as the secret key index 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	RSA public key index (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 key index 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	RSA secret key index (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 key index 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_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).
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	RSA secret key index (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 key index 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 key index 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.

4.2.6.9 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).
data_type		Selects data type of message_hash. Message: 0 Hash value: 1
key_index	Input	Inputs RSA public key index. (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 key index 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 index 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 index 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.

4.2.6.10 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).
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	RSA secret key index (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 key index 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 key index 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_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).
data_type		Selects data type of message_hash. Message: 0 Hash value: 1
key_index	Input	Inputs RSA public key index. (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 key index 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 index 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 index 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_GenerateEccP386PublicKeyIndex(
        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	Provisioning key wrapped by the DLM server
iv	Input	Initialization vector when generating encrypted_key
encrypted_key	Input	Encrypted ECC public key with MAC appended
key_index	Output	ECC public key index
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-386 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 index.

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_GenerateEccP386PrivateKeyIndex(
        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	Provisioning key wrapped by the DLM server
iv	Input	Initialization vector when generating encrypted_key
encrypted_key	Input	Encrypted ECC secret key with MAC appended
key_index	Output	ECC secret key index (1) ECC P-192 secret key index (2) ECC P-224 secret key index (3) ECC P-256 secret key index (4) ECC P-386 secret key index

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-386 secret key index.

Refer to 7.4, Public Key Index Formats for Asymmetric Cryptography, for the format of the data encrypted using the provisioning key 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	Public key encrypted using update keyring with MAC appended
key_index	Output	ECC public key index
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 index.

Refer to 7.3.5, ECC, for an explanation of the method and format used to encrypt the public key using the update keyring 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	Secret key encrypted using update keyring with MAC appended
key_index	Output	ECC secret key index (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 secret key index.

Refer to 7.3.5, ECC, for an explanation of the method and format used to encrypt the private key using the update keyring 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_GenerateEccP386RandomKeyIndex(
        tsip_ecc_key_pair_index_t *key_pair_index
)
```

Parameters

key_pair_index	Output	ECC public key–secret key pair key indexes (1) ECC P-192 (2) ECC P-224 (3) ECC P-256 (4) ECC P-384
->public		Public key index
value		Public key value
key_management_info		Key management information
key_q		Public key (Qx || Qy) (plaintext)
->private		Secret key index

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 key indexes 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 key indexes output by this API makes it possible to prevent dead copying of data. The public key index is generated in key_pair_index->public, and the secret key index 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 index output by R_TSIP_GenerateEccPXXXPublicKeyIndex(), and key_pair_index->private is utilized in the same manner as the secret key index 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).
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	ECC secret key index (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_KEY_SET	Invalid 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 key index 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 key index 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 key index 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 key index 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).

data_type		Selects data type of message_hash. Message: 0 Hash value: 1
key_index	Input	ECC public key index (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_KEY_SET	Invalid 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 using the public key index 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 using the public key index 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 index 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 index 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	Provisioning key wrapped by the DLM server
iv	Input	Initialization vector when generating encrypted_key
encrypted_key	Input	Encrypted user key with MAC appended
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 SHA1-HMAC or SHA256-HMAC key index.

Refer to 7.3.6, SHA-HMAC, for the format of the data encrypted using the provisioning key 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	User key encrypted using update keyring with MAC appended
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 HMAC key index.

Refer to 7.3.6, SHA-HMAC for the format of the data encrypted using the update keyring 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	Key index area

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 key index 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 MAC key index 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	Key index area

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 key index 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 MAC key index 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	Key index area for AES-128 CMAC operation
sender_private_key_index	Input	Secret key index for DH operation The secret key d included in the secret key index 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 key index 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

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(), and R_TSIP_EcdhP256KeyDerivation() 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 key index area 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	Key index 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 key index 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 key index as the fifth parameter.

The first parameter, handle, is used subsequently as a parameter by the R_TSIP_EcdhP256CalculateSharedSecretIndex() function.

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 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_Ecdh256Init().
public_key_index	Input	For ECDHE, input a null pointer. For ECDH, input an ECC P-256 public key index.
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 key index 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 key index 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. The generated signature is for applications conforming to the DLMS/COSEM standard for smart meters.

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 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 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 key index whose signature was verified by R_TSIP_EcdhP256ReadPublicKey()
private_key_index	Input	Secret key index
shared_secret_index	Output	Key index 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 key index 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 key index 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 key index whose signature was verified by R_TSIP_EcdhP256ReadPublicKey() function.

If the value of the key_type parameter of R_TSIP_EcdhP256Init() function is 0, input the key_index, secret key index 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 other than 0, input the secret key index corresponding to the second parameter of R_TSIP_EcdhP256MakePublicKey() as the third parameter, private_key_index.

The R_TSIP_EcdhP256KeyDerivation() function subsequently uses shared_secret_index as key material for outputting the key index.

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

Reentrancy

Not supported.

4.2.11.5 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 key index 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: AES-128 1: AES-256 2: SHA256-HMAC
other_info	Input	Additional data used for key derivation calculation AlgorithmID || PartyUInfo || PartyVInf
other_info_length	Input	Byte length of other_info (147 or fewer byte units)
salt_key_index	Input	Salt key index (Input NULL when kdf_type is 0.)
key_index	Output	Key index corresponding to key_type When the value of key_type is 2, an SHA256-HMAC key index 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 key index 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 key index 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 key index 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.

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 Key Index	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	Key index area 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 key index 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.

4.2.12 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 key index used for wrapping (2) AES-256 key index 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	Key index to be wrapped target_key_type 0: 13 word size target_key_type 2: 17 word size
wrapped_key	Output	Wrapped key 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 key index 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 key index used for unwrapping (2) AES-256 key index 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	Wrapped key target_key_type 0: 6 word size target_key_type 2: 10 word size
target_key_index	Output	Key index 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 key index 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)

4.2.13.1 R_TSIP_GenerateTlsRsaPublicKeyIndex

Format

```
#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
)
```

Parameters

encrypted_provisioning_key	Input	Provisioning key wrapped by the DLM server
iv	Input	Initialization vector when generating encrypted_key
encrypted_key	Input	2048-bit RSA public key encrypted in AES128-ECB mode
key_index	Output	2048-bit RSA public key index for use by TLS cooperation function 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 key index for use by the TLS cooperation function.

Refer to 7.3.4.2(1) for the format of the data encrypted using the provisioning key 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_UpdateTlsRsaPublicKeyIndex

Format

```
#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
)
```

Parameters

iv	Input	Initialization vector when generating encrypted_key
encrypted_key	Input	Public key encrypted using update keyring with MAC appended
key_index	Output	2048-bit RSA public key index for use by TLS cooperation function 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 key index for use by the TLS cooperation function.

Refer to 7.3.4.2(1) for the format of the data encrypted using the provisioning key 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_TlsRootCertificateVerification

Format

```
#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
)
```

Parameters

public_key_type	Input	Public key type included in certificate 0: RSA 2048-bit, 2: ECC P-256, other: reserved
certificate	Input	Root CA certificate bundle (DER format)
certificate_length	Input	Byte length of root CA certificate bundle
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_TlsCertificateVerification or R_TSIP_TlsCertificateVerificationExtension. 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.

4.2.13.4 R_TSIP_TlsCertificateVerification

Format

```
#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
)
```

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), other: reserved
encrypted_input_public_key	Input	Encrypted public key Use the value of encrypted_root_public_key output by R_TSIP_TlsRootCertificateVerification or the value of encrypted_output_public_key output by R_TSIP_TlsCertificateVerification or R_TSIP_TlsCertificateVerificationExtension. 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
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_TlsCertificateVerification or R_TSIP_TlsCertificateVerificationExtension, 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_TlsCertificateVerification or R_TSIP_TlsCertificateVerificationExtension 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_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.

4.2.13.5 R_TSIP_TlsCertificateVerificationExtension

Format

```
#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
)
```

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), other: reserved
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), other: reserved
encrypted_input_public_key	Input	Encrypted public key Use the value of encrypted_root_public_key output by R_TSIP_TlsRootCertificateVerification or the value of encrypted_output_public_key output by R_TSIP_TlsCertificateVerification or R_TSIP_TlsCertificateVerificationExtension. 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

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_TlsCertificateVerification or R_TSIP_TlsCertificateVerificationExtension, 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_TlsCertificateVerification or R_TSIP_TlsCertificateVerificationExtension 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_TlsCertificateVerification() 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)

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	Pre-master secret data on which TSIP-specific conversion has been performed 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 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	Pre-master secret data on which TSIP-specific conversion has been performed, 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_TlsGenerateMasterSecret

Format

```
#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
)
```

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	Pre-master secret data on which TSIP-specific conversion has been performed Use the value of tsip_pre_master_secret output by R_TSIP_TlsGeneratePreMasterSecret or encrypted_pre_master_secret output by R_TSIP_TlsGeneratePreMasterSecretWithEccP256Key.
client_random	Input	32-byte random number value reported by ClientHello
server_random	Input	32-byte random number value reported by ServerHello
tsip_master_secret	Output	Master secret data on which TSIP-specific conversion has been performed Use this value as the tsip_master_secret parameter input to R_TSIP_TlsGenerateSessionKey or R_TSIP_TlsGenerateVerifyData. 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.

4.2.14.4 R_TSIP_TlsGenerateSessionKey

Format

```
#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_crypt_key_index,
        tsip_aes_key_index_t *server_crypt_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	Master secret data on which TSIP-specific conversion has been performed Use the value of tsip_master_secret output by R_TSIP_TlsGenerateMasterSecret.
client_random	Input	32-byte random number value reported by ClientHello
server_random	Input	32-byte random number value reported by ServerHello
nonce_explicit	Input	Nonce used by AES128 GCM cipher suite select_cipher_suite = 6-7: 8 bytes
client_mac_key_index	Output	MAC key index for client to server communication
server_mac_key_index	Output	MAC key index for server to client communication
client_crypt_key_index	Output	AES common key index for client to server communication
server_crypt_key_index	Output	AES common key index 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. (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. (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.

Key information used for communication is retained internally by the TSIP.

Reentrancy

Not supported.

4.2.14.5 R_TSIP_TlsGenerateVerifyData

Format

```
#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
)
```

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	Master secret data on which TSIP-specific conversion has been performed Use the value of tsip_master_secret output by R_TSIP_TlsGenerateMasterSecret.
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 verification data for use by the TLS cooperation function.

Reentrancy

Not supported.

4.2.14.6 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, 1: Reserved, 2: ECDSA P-256
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.

4.2.14.7 R_TSIP_GenerateTlsP256EccKeyIndex

Format

```
#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
)
```

Parameters

tls_p256_ecc_key_index	Output	Ephemeral ECC secret key index Use this value as the tls_p256_ecc_key_index parameter input to R_TSIP_TlsGeneratePreMasterSecretWithEccP256Key.
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 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.

4.2.14.8 R_TSIP_TlsGeneratePreMasterSecretWithEccP256Key

Format

```
#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
)
```

Parameters

encrypted_public_key	Input	Encrypted ephemeral ECDH public key Use the value of encrypted_ephemeral_ecdh_public_key output by R_TSIP_TlsServersEphemeralEcdhPublicKeyRetrieves.
tls_p256_ecc_key_index	Input	Ephemeral ECC secret key index Use the value of tls_p256_ecc_key_index output by R_TSIP_GenerateTlsP256EccKeyIndex.
tsip_pre_master_secret	Output	Pre-master secret data on which TSIP-specific conversion has been performed 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

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.

4.2.14.9 R_TSIP_TlsGenerateExtendedMasterSecret

Format

```
#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
)
```

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	Pre-master secret data on which TSIP-specific conversion has been performed Use the value of tsip_pre_master_secret output by R_TSIP_TlsGeneratePreMasterSecret or R_TSIP_TlsGeneratePreMasterSecretWithEccP256Key.
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	Extended master secret data on which TSIP-specific conversion has been performed 20 words (80 bytes) of data is output. Use this value as the tsip_master_secret parameter input to R_TSIP_TlsGenerateSessionKey or R_TSIP_TlsGenerateVerifyData.

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.

4.2.15 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 key index 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 key index Use the value of key_index output by R_TSIP_GenerateTls13P256EccKeyIndex.
shared_secret_key_index	Output	Shared secret ephemeral key index 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 key index 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 key index 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 key index Use the value of shared_secret_key_index output by R_TSIP_Tls13GenerateEcdheSharedSecret.
handshake_secret_key_index	Output	Handshake secret ephemeral key index 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 key index input

Description

This API generates a handshake secret key index, 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 key index 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 key index Use this value as the key_index parameter input to R_TSIP_Tls13DecryptInit.
server_finished_key_index	Output	Server finished key ephemeral key index 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 key index input

Description

This API generates, using the handshake secret output by R_TSIP_Tls13GenerateHandshakeSecret, a server write key index and server finished key index 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 key index 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 key index Use this value as the key_index parameter input to R_TSIP_Tls13EncryptInit.
client_finished_key_index	Output	Client finished key ephemeral key index 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 key index input

Description

This API generates, using the handshake secret output by R_TSIP_Tls13GenerateHandshakeSecret, a client write key index and client finished key index 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 key index 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 key index 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 key index 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 key index 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 key index input

Description

This API generates a master secret ephemeral 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 key index 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 key index 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 key index 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 key index Use this value as the key_index parameter input to R_TSIP_Tls13DecryptInit.

client_write_key_index	Output	Client write key ephemeral key index 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 key index input

Description

This API generates an application traffic secret key index, 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 key indexes.

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 key index 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 key index 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 key index 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 key index input
TSIP_ERR_PARAMETER	Invalid input data

Description

This API updates, using an application traffic secret, an encryption key index corresponding to an application traffic secret key index 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 key index 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 key index 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 key index input

Description

This API generates a resumption master secret key index, 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 key index, after generating a resumption master secret key index 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 key index 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
pre_shared_key_index	Output	Pre-shared key ephemeral key index 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 key index input

Description

This API generates, using a resumption master secret ephemeral key, a pre-shared key index 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 key index Use the value of pre_shared_key_index output by R_TSIP_Tls13GeneratePreSharedKey.
psk_binder_key_index	Output	PSK binder key ephemeral key index 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 key index input

Description

This API generates binder key index 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 key index Use the value of pre_shared_key_index output by R_TSIP_Tls13GeneratePreSharedKey.
shared_secret_key_index	Input	Shared secret ephemeral key index Use the value of shared_secret_key_index output by R_TSIP_Tls13GenerateEcdheSharedSecret.
handshake_secret_key_index	Output	Handshake secret ephemeral key index 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 key index input

Description

This API generates a handshake secret key index, using the pre-shared key index 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 key index 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 key index 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 key index input

Description

This API generates a client write key index 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 key index 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
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 key index 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
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 key index 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 key index 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 key index Use the value of key_index output by R_TSIP_GenerateTls13SVP256EccKeyIndex.
shared_secret_key_index	Output	Shared secret ephemeral key index 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 key index 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 key index 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 key index Use the value of shared_secret_key_index output by R_TSIP_Tls13SVGenerateEcdheSharedSecret.
handshake_secret_key_index	Output	Handshake secret ephemeral key index 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 key index input

Description

This API generates a handshake secret key index, 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 key index 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 key index Use this value as the key_index parameter input to R_TSIP_Tls13EncryptInit.
server_finished_key_index	Output	Server finished key ephemeral key index 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 key index input

Description

This API generates, using the handshake secret output by R_TSIP_Tls13SVGenerateHandshakeSecret, a server write key index and server finished key index 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 key index 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 key index Use this value as the key_index parameter input to R_TSIP_Tls13DecryptInit.
client_finished_key_index	Output	Client finished key ephemeral key index 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 key index input

Description

This API generates, using the handshake secret output by R_TSIP_Tls13SVGenerateHandshakeSecret, a client write key index and client finished key index 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 key index 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 key index 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 key index Use the value of handshake_secret_key_index output by R_TSIP_Tls13SVGenerateHandshakeSecret.
master_secret_key_index	Output	Master secret ephemeral key index 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 key index input

Description

This API generates a master secret ephemeral key index, 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 key index 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 key index 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 key index 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 key index Use this value as the key_index parameter input to R_TSIP_Tls13EncryptInit
client_write_key_index	Output	Client write key ephemeral key index 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 key index input

Description

This API generates an application traffic secret key index, 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 key indexes.

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 key index 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 key index 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 key index 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 key index input
TSIP_ERR_PARAMETER	Invalid input data

Description

This API updates, using an application traffic secret, an encryption key index corresponding to an application traffic secret key index 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 key index 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 key index 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 key index input

Description

This API generates a resumption master secret key index, 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 key index, after generating a resumption master secret key index 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 key index 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
pre_shared_key_index	Output	Pre-shared key ephemeral key index 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 key index input

Description

This API generates, using a resumption master secret ephemeral key, a pre-shared key index 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 key index Use the value of pre_shared_key_index output by R_TSIP_Tls13SVGeneratePreSharedKey.
psk_binder_key_index	Output	Psk binder key ephemeral key index 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 key index input

Description

This API generates a binder key index 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 key index Use the value of pre_shared_key_index output by R_TSIP_Tls13SVGeneratePreSharedKey.
shared_secret_key_index	Input	Shared secret ephemeral key index Use the value of shared_secret_key_index output by R_TSIP_Tls13SVGenerateEcdheSharedSecret.
handshake_secret_key_index	Output	Handshake secret ephemeral key index 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 key index input

Description

This API generates a handshake secret key index, using the pre-shared key index 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 key index 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 key index 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 key index input

Description

This API generates a client write key index 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 key index 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
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 key index input
TSIP_ERR_PARAMETER	Invalid input data

Description

This API generates a CertificateVerify message to be sent to the server 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
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 key index input
TSIP_ERR_PARAMETER	Invalid input data

Description

This API verifies a CertificateVerify message received from the server 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 key index 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 key index 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 key index 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 key index 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_GenerateFirmwareMAC

Format

```
#include "r_tsip_rx_if.h"
e_tsip_err_t R_TSIP_GenerateFirmwareMAC(
        uint32_t *InData_KeyIndex,
        uint32_t *InData_SessionKey,
        uint32_t *InData_UpProgram,
        uint32_t *InData_IV,
        uint32_t *OutData_Program,
        uint32_t MAX_CNT,
        TSIP_GEN_MAC_CB_FUNC_T p_callback,
        tsip_firmware_generate_mac_resume_handle_t *tsip_firmware_generate_mac_resume_handle
)
```

Parameters

InData_KeyIndex	Input	Key index area for InData_SessionKey decryption and generation of firmware MAC values
InData_SessionKey	Input	Session key area for decryption of encrypted firmware and verification of checksum values
InData_UpProgram	Input	Area (512 words (2048 bytes) in the demo project) for temporary storage of encrypted firmware data
InData_IV	Input	Initialization vector area for decryption of encrypted firmware.
OutData_Program	Output	Area (512 words (2048 bytes) in the demo project) for temporary storage of decrypted firmware data
MAX_CNT	Input	Encrypted firmware word size + MAC word size The firmware word size must be a multiple of 4 words. The MAC size is fixed at 4 words (128 bits), so input the firmware word size + 4. The minimum size of the encrypted firmware is 16 words, so the minimum value of MAX_CNT is 20.
p_callback	Input	This callback function is called multiple times when action by the user is required. The type of action is determined by the enum TSIP_FW_CB_REQ_TYPE.
tsip_firmware_generate_mac_resume_handle	Input	R_TSIP_GenerateFirmwareMAC handler (work area)

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 key index input
TSIP_ERR_CALLBACK_UNREGIST	Invalid p_callback value
TSIP_ERR_PARAMETER	Invalid input data
TSIP_RESUME_FIRMWARE_GENERATE_MAC	There is additional processing. It is necessary to call the API again.

Description

This function accepts encrypted firmware data and a firmware checksum value, decrypts the firmware, and generates a new MAC value. The user can update the firmware by writing the decrypted firmware and new MAC value to the flash ROM. Refer to section 3.14, Firmware Update, for details of the firmware update functionality.

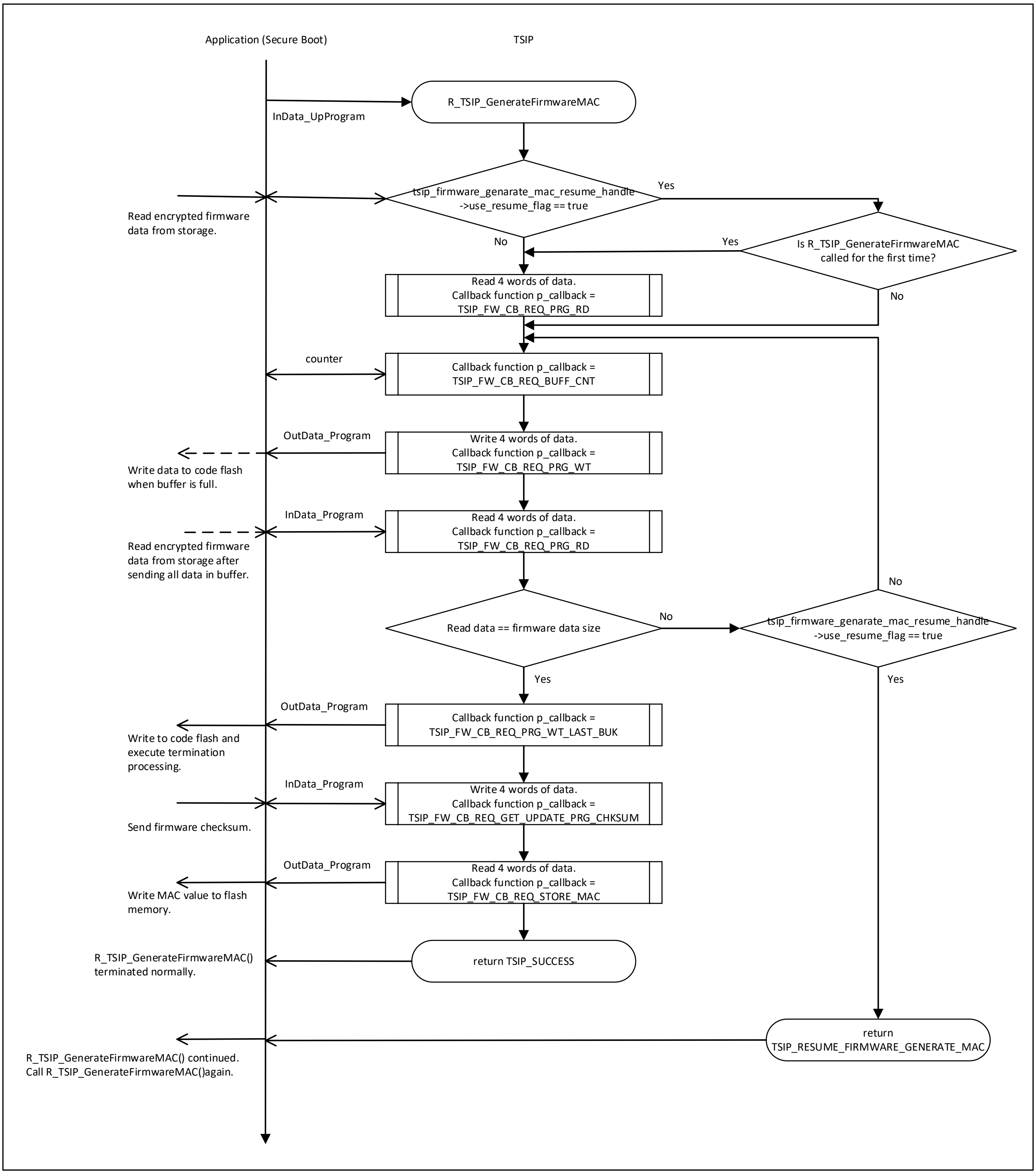

Figure 4.1 Flowchart of Calls to Callback Function

Processing to read and write firmware data is performed in 4-word units. The following steps are used to call the callback function registered as the seventh parameter, p_callback. The string in parentheses () is the type of processing specified by the first parameter, req_type, of the callback function p_callback.

1. Increment adjustment (TSIP_FW_CB_REQ_BUFF_CNT)
2. Writing of decrypted firmware to storage destination (TSIP_FW_CB_REQ_PRG_WT)
3. Storage of encrypted firmware in InData_UpProgram (TSIP_FW_CB_REQ_PRG_RD)

It is not necessary to perform the processing in the callback function every time. Perform processing as appropriate for the sizes reserved for InData_Program and OutData_Program.

For example, if a 512-word buffer has been reserved, adjust the increment to match the buffer position on the 512 / 4 = 128th time (TSIP_FW_CB_REQ_BUFF_CNT), write to the storage destination (TSIP_FW_CB_REQ_PRG_WT), and store the encrypted firmware in InData_UpProgram (TSIP_FW_CB_REQ_PRG_RD).

As the write request to the final storage destination, specify req_type = TSIP_FW_CB_REQ_PRG_WT_LAST_BLK (not TSIP_FW_CB_REQ_PRG_WT).

This API is called again by the callback function p_callback after reading and writing all of the firmware data has completed. After confirming that the value of the first parameter, req_type, of the callback function p_callback is TSIP_FW_CB_REQ_GET_UPDATE_PRG_CHKSUM, pass the checksum value to the fourth parameter, InData_UpProgram, of p_callback. This API generates a firmware MAC value after reading and verifying the checksum value. After this, the MAC value is passed to the user using the fifth parameter, OutData_Program, when the first parameter, req_type, of callback function p_callback is TSIP_FW_CB_REQ_STORE_MAC. Store the MAC value in the flash memory area.

If called when tsip_firmware_generate_mac_resume_handle->use_resume_flag is set to true, this API operates as a firmware update start and update function but does not perform the firmware update processing in its entirety. If there is additional processing remaining, a value of TSIP_RESUME_FIRMWARE_GENERATE_MAC is returned. Continue to call R_TSIP_GenerateFirmwareMAC() until a value of TSIP_SUCCESS is returned. A return value of TSIP_SUCCESS indicates that firmware update processing has completed successfully.

Reentrancy

Not supported.

4.2.16.3 R_TSIP_VerifyFirmwareMAC

Format

```
#include "r_tsip_rx_if.h"
e_tsip_err_t R_TSIP_VerifyFirmwareMAC(
        uint32_t *InData_Program,
        uint32_t MAX_CNT,
        uint32_t *InData_MAC
)
```

Parameters

InData_Program	Input	Firmware
MAX_CNT	Input	Firmware word size + MAC size The firmware word size must be a multiple of 4 words. The MAC size is fixed at 4 words (16 bytes), so input the firmware word size + 4. The minimum size of the encrypted firmware is 16 words, so the minimum value of MAX_CNT is 20.
InData_MAC	Input	MAC value to be compared (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_PARAMETER	Invalid input data

Description

This function accepts firmware data and a MAC value, and verifies the MAC value. As the third parameter, InData_Mac, pass the MAC value generated by R_TSIP_GenerateFirmwareMAC().

AES-CMAC is used as the MAC verification algorithm.

Reentrancy

Not supported.

4.2.16.4 TSIP_GEN_MAC_CB_FUNC_T Type

Format

```
#include "r_tsip_rx_if.h"
typedef void (*TSIP_GEN_MAC_CB_FUNC_T)(
        TSIP_FW_CB_REQ_TYPE req_type,
        uint32_t iLoop,
        uint32_t *counter,
        uint32_t *InData_UpProgram,
        uint32_t *OutData_Program,
        uint32_t MAX_CNT)
```

Parameters

req_type	Input	Contents of request (TSIP_FW_CB_REQ_TYPE)
iLoop	Input	Loop count (word units)
counter	Input	Offset for area references
InData_UpProgram	Input	Same address as third parameter, InData_UpProgram, of R_TSIP_GenerateFirmwareMAC()
OutData_Program	Input/output	Same address as fifth parameter, OutData_Program, of R_TSIP_GenerateFirmwareMAC()
MAX_CNT	Input	Same value as sixth parameter, MAX_CNT, of R_TSIP_GenerateFirmwareMAC()

Return Values

None

Description

This function is used by the R_TSIP_GenerateFirmwareMAC function and is registered as its seventh parameter.

The function is used to store the decrypted firmware and MAC on the user side.

The size of the InData_UpProgram and OutData_Program areas must be a multiple of 4 and no fewer than 4 words. InData_UpProgram and OutData_Program should be the same size. The demo project uses the size of the minimum code flash write unit.

This callback function is called by the R_TSIP_GenerateFirmwareMAC function for multiple types of requests. The type of request is stored in the first parameter, req_type.

The first parameter, req_type, has the value defined by the enum TSIP_FW_CB_REQ_TYPE.

```
typedef enum
{
    TSIP_FW_CB_REQ_PRG_WT = 0u,
    TSIP_FW_CB_REQ_PRG_RD,
    TSIP_FW_CB_REQ_BUFF_CNT,
    TSIP_FW_CB_REQ_PRG_WT_LAST_BLK,
    TSIP_FW_CB_REQ_GET_UPDATE_PRG_CHKSUM,
    TSIP_FW_CB_REQ_STORE_MAC,
}TSIP_FW_CB_REQ_TYPE;
```

Based on this value, the user takes the necessary actions.

<req_type = TSIP_FW_CB_REQ_PRG_WT>

This is a request to store decrypted firmware.

The TSIP module issues this request each time after storing data in the fifth parameter, OutData_Program, in 4-word units. It is not necessary to process every request. Store the decrypted firmware as appropriate in the areas secured on the user side. For example, if 8-word areas have been secured, store the decrypted firmware once every two requests.

The total decrypted data size is stored in the second parameter, iLoop. The maximum value of iLoop in this request is the value of the sixth parameter, MAX_CNT, minus 4 words. The last 4 words and the unstored firmware are handled by the request <req_type = TSIP_FW_CB_REQ_PRG_WT_LAST_BLK>.

<req_type = TSIP_FW_CB_REQ_PRG_RD>

This is a request to obtain encrypted firmware to be applied as an update.

The TSIP module issues this request each time before performing decryption in 4-word units. It works in the same manner as <req_type = TSIP_FW_CB_REQ_PRG_WT>. Store the decrypted firmware, as appropriate for the areas secured on the user side, in the fourth parameter, InData_UpProgram.

<req_type = TSIP_FW_CB_REQ_BUFF_CNT>

This is a request for the offset value used when referencing the fourth parameter, InData_UpProgram, and the fifth parameter, OutData_Program. Return a value in 4-word increments to the third parameter, counter. If the size secured by the fourth parameter, InData_UpProgram, and the fifth parameter, OutData_Program, is exceeded, restore the third parameter, counter, to its default value.

<req_type = TSIP_FW_CB_REQ_PRG_WT_LAST_BLK>

This is a request that is issued when the last block of the encrypted firmware is decrypted. Store the areas that cannot be stored by the decrypted firmware at this time.

<req_type = TSIP_FW_CB_REQ_GET_UPDATE_PRG_CHKSUM>

This is a request to obtain the firmware checksum value for the firmware to be applied as an update.

Store the checksum value in the fourth parameter, InData_UpProgram. The checksum size is 16 bytes.

<req_type = req_type = TSIP_FW_CB_REQ_STORE_MAC>

This is a request to output the MAC for the decrypted firmware.

The MAC is stored in the fifth parameter, OutData_Program. The MAC size is 16 bytes.

The value of the sixth parameter, MAX_CNT, is the same as that of the sixth parameter of R_TSIP_GenerateFirmwareMAC(), MAX_CNT.

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.

4.4 Using Renesas Secure Flash Programmer

4.4.1 Key Wrap Tab

Figure 5.2 shows the Key Wrap tab of Renesas Secure Flash Programmer. Table 5.1 contains descriptions of the setting values that can be configured on the Key Wrap tab. Based on the descriptions in Table 5.1, enter appropriate setting values and click the **Generate Key Files…** button to generate encrypted key files (key_data.c and key_data.h). Refer to Table 5.2 for descriptions of the buttons.

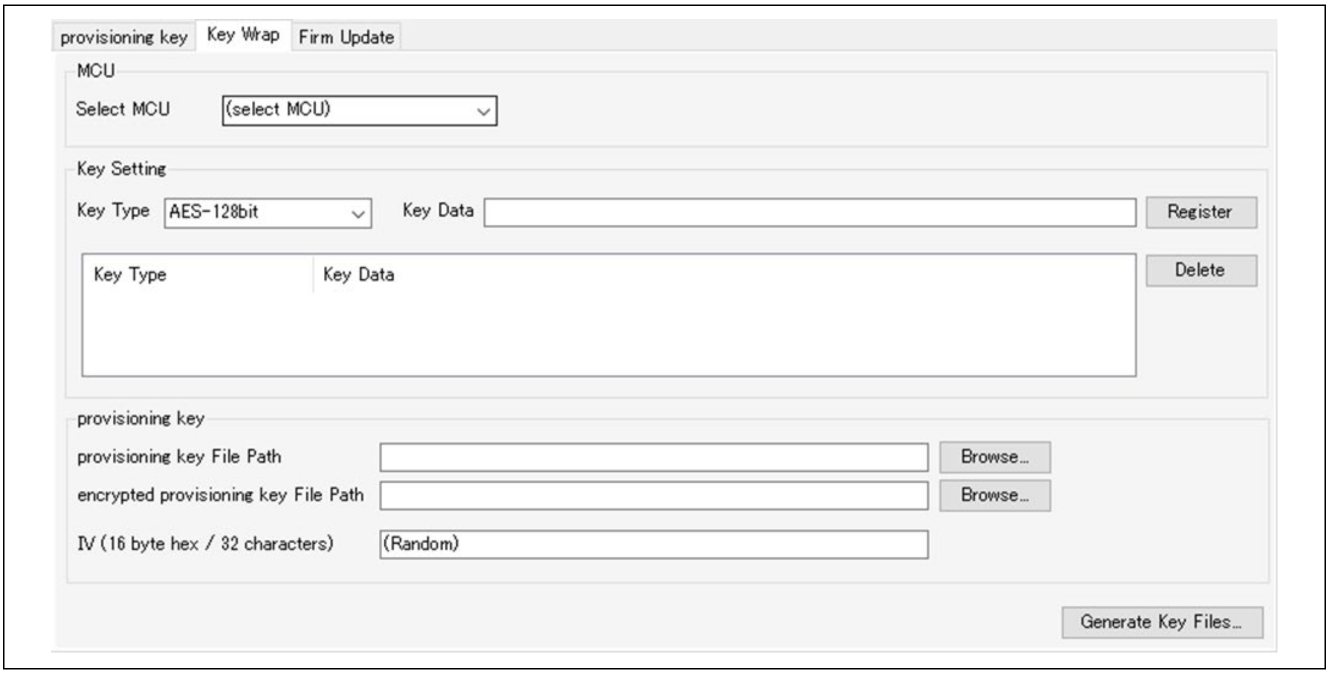

Figure 4.2 Key Wrap Tab of Renesas Secure Flash Programmer

4.4.1.1 Key Data Input Formats

Enter the following data in big-endian order in the **Key Data** field of the Key Wrap tab.

(1) AES 128-bit data format

Bytes	128 bits
0-15	AES 128 key data

(2) AES 256-bit data format

Bytes	256 bits
0-31	AES 256 key data

(3) TDES data format

Bytes	DES user key 1	DES user key 2	DES user key 3
0-23	DES key data	DES key data	DES key data

(4) 2Key-TDES data format

Bytes	DES user key 1	DES user key 2
0-15	DES key data	DES key data

(5) DES data format

Bytes	DES user key 1
0-7	DES key data

The DES key data is 8 bits long and consists of 7 bits of key data and 1 odd parity bit.

DES key data format is shown below.

DES user key n							
Byte No.	0		1		…	8	
Bits	7-1	0	7-1	0	…	7-1	0
Data	Key data	Odd parity	Key data	Odd parity	…	Key data	Odd parity

Example: When the parity bit is added, the DES user key 0x00000000000000 becomes 0x0101010101010101, 0xFFFFFFFFFFFFFF becomes 0xFEFEFEFEFEFEFEFE, 0x01020304050607 becomes 0x018080614029190E.

(6) ARC4 data format

Bytes	2048 bits
0-255	ARC4 key data

(7) SHA1-HMAC data format

Bytes	160 bits
0-19	SHA1-HMAC key data

(8) SHA256-HMAC data format

Bytes	256 bits
0-31	SHA256-HMAC key data

(9) RSA 1024-bit public data format (132 bytes)

Bytes	RSA 1024-bit modulus n	RSA 1024-bit exponent e
0-131	128-byte RSA modulus n data	4-byte RSA exponent e data

Note: Public key

(10) RSA 1024-bit private data format (256 bytes)

Bytes	RSA 1024-bit modulus n	RSA 1024-bit decryption exponent d
0-255	128-byte RSA modulus n data	128-byte RSA decryption exponent d data

(11) RSA 1024-bit all data format (260 bytes)

Bytes	RSA 1024-bit modulus n	RSA 1024-bit exponent e	RSA 1024-bit decryption exponent d
0-259	128-byte RSA modulus n data	4-byte RSA public key data	128-byte RSA decryption exponent d data

(12) RSA 2048-bit public data format (260 bytes)

Bytes	RSA 2048-bit modulus n	RSA 2048-bit exponent e
0-259	256-byte RSA modulus n data	4-byte RSA exponent e data

(13) RSA 2048-bit private data format (512 bytes)

Bytes	RSA 2048-bit modulus n	RSA 2048-bit decryption exponent d
0-511	256-byte RSA modulus n data	256-byte RSA decryption exponent d data

(14) RSA 2048-bit all data format (516 bytes)

Bytes	RSA 2048-bit modulus n	RSA 2048-bit exponent e	RSA 2048-bit decryption exponent d
0-515	256-byte RSA modulus n data	4-byte RSA exponent e data	256-byte RSA decryption exponent d data

(15) RSA 3072-bit public data format (388 bytes)

Bytes	RSA 3072-bit modulus n	RSA 3072-bit exponent e
0-387	384-byte RSA modulus n data	4-byte RSA exponent e data

(16) RSA 4096-bit public data format (516 bytes)

Bytes	RSA 4096-bit modulus n	RSA 4096-bit exponent e
0-515	512-byte RSA modulus n data	4-byte RSA exponent e data

(17) ECC 192-bit public data format (48 bytes)

Bytes	ECC 192-bit public key Qx	ECC 192-bit public key Qy
0-47	24-byte ECC public key Qx data	24-byte ECC public key Qy data

(18) ECC 192-bit private data format (24 bytes)

Bytes	ECC 192-bit private key
0-23	24-byte ECC secret key data

(19) ECC 192-bit all data format (72 bytes)

Bytes	ECC 192-bit public key Qx	ECC 192-bit public key Qy	ECC 192-bit private key
0-71	24-byte ECC public key Qx data	24-byte ECC public key Qy data	24-byte ECC secret key data

(20) ECC 224-bit public data format (56 bytes)

Bytes	ECC 224-bit public key Qx	ECC 224-bit public key Qy
0-55	28-byte ECC public key Qx data	28-byte ECC public key Qy data

(21) ECC 224-bit private data format (28 bytes)

Bytes	ECC 224-bit private key
0-27	28-byte ECC secret key data

(22) ECC 224-bit all data format (84 bytes)

Bytes	ECC 224-bit public key Qx	ECC 224-bit public key Qy	ECC 224-bit private key
0-83	28-byte ECC public key Qx data	28-byte ECC public key Qy data	28-byte ECC secret key data

(23) ECC 256-bit public data format (64 bytes)

Bytes	ECC 256-bit public key Qx	ECC 256-bit public key Qy
0-63	32-byte ECC public key Qx data	32-byte ECC public key Qy data

(24) ECC 256-bit private data format (32 bytes)

Bytes	ECC 256-bit private key
0-31	32-byte ECC secret key data

(25) ECC 256-bit all data format (96 bytes)

Bytes	ECC 256-bit public key Qx	ECC 256-bit public key Qy	ECC 256-bit private key
0-95	32-byte ECC public key Qx data	32-byte ECC public key Qy data	32-byte ECC secret key data

(26) ECC 384-bit public data format (96 bytes)

Bytes	ECC 384-bit public key Qx	ECC 384-bit public key Qy
0-95	48-byte ECC public key Qx data	48-byte ECC public key Qy data

(27) ECC 384-bit private data format (48 bytes)

Bytes	ECC 384-bit private key
0-47	48-byte ECC secret key data

(28) ECC 384-bit all data format (144 bytes)

Bytes	ECC 384-bit public key Qx	ECC 384-bit public key Qy	ECC 384-bit private key
0-143	48-byte ECC public key Qx data	48-byte ECC public key Qy data	48-byte ECC secret key data

5. Appendix

5.1 Confirmed Operation Environment

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

Table 5.1 Confirmed Operation Environment

Item	Description
Integrated development environment	Renesas Electronics e² studio 2023-01 IAR Embedded Workbench for Renesas RX 4.20.01
C compiler	Renesas Electronics C/C++ Compiler for RX Family (CC-RX) V3.05.00 Compile options: The following option has been added to the default settings of the integrated development environment. -lang = c99
	GCC for Renesas RX 8.3.0.202204 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 4.20.01 Compiler options: Default settings of the integrated development environment
Renesas Secure Flash Programmer (GUI tool)	The following software is required: Microsoft .NET Framework 4.5 or later
Endian order	Big-endian or little-endian
Module version	Ver. 1.18
Board used	Renesas Starter Kit for RX231 (B version) (product No.: R0K505231S020BE) Renesas Solution Starter Kit for RX23W (with TSIP) (product No.: RTK5523W8BC00001BJ) Motor Control Kit for RX26T (product No.: RTK0EMXE70S00020BJ) 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) Renesas Starter Kit for RX72T (with TSIP) (product No.: RTK5572TKCS00010BE)

5.2 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.

5.3 User Key Encryption Formats

For key injection the user key is wrapped using a provisioning key and IV, and for key updating the user key is wrapped using an update keyring 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 user key).

Refer to 3.7.1, Key Injection and Updating, for information on encryption methods.

5.3.1 AES

5.3.1.1 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			

5.3.1.2 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			

5.3.2 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.

5.3.3 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			

5.3.4 RSA

5.3.4.1 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			

5.3.4.2 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			

5.3.4.3 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			

(2) Secret Key

Input (User key)

Bytes	16			
	4	4	4	4
0-383	RSA 3072-bit public key n			
384-767	RSA 3072-bit secret key d			

Output (Encrypted key)

Bytes	16			
	4	4	4	4
0-511	encrypted_user_key (RSA 3072-bit public key n || secret key d)			
512-527	MAC			

5.3.4.4 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			

(2) Secret Key

Input (User key)

Bytes	16			
	4	4	4	4
0-511	RSA 4096-bit public key n			
512-1023	RSA 4096-bit public key d			

Output (Encrypted key)

Bytes	16			
	4	4	4	4
0-1023	encrypted_user_key (RSA 4096-bit public key n || secret key d)			
1024-1039	MAC			

5.3.5 ECC

5.3.5.1 ECC P 192-Bit Key

(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			

5.3.5.2 ECC P 224-Bit Key

(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			

5.3.5.3 ECC P 256-Bit Key

(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			

5.3.5.4 ECC P 384-Bit Key

(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 (0 padding || ECC P 384-bit secret key d)			
48-63	MAC			

5.3.6 HMAC

5.3.6.1 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-SHA224 || 0 padding)			
32-47	MAC			

5.3.6.2 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			

5.3.7 Update Keyring

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			

5.4 Public Key Index Formats for Asymmetric Cryptography

Public keys for asymmetric cryptography contain plaintext information in the key index. This enables plaintext information to be extracted from the key index using the TSIP's key generation functionality. The data format of each cryptographic algorithm is described below.

5.4.1 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.

5.4.2 ECC

The member value.key_q of the key index 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.

5.4.2.1 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	Key index management information			

5.4.2.2 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	Key index management information			

5.4.2.3 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	Key index management information			

5.4.2.4 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	Key index management information			

6. 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

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

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/.

Trademarks

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