APPLICATION NOTE

RX Family

R20AN0548EJ0114
Rev.1.14
Oct. 22, 2021

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

Introduction

This application note describes the use of the software drivers for utilizing the TSIP (Trusted Secure IP) and TSIP-Lite capabilities on the RX Family of microcontrollers. This software is called the TSIP driver. The TSIP driver provides APIs for performing the cryptographic capabilities summarized in Table 1, as well as for securely performing firmware updates.

Table 1 Cryptographic Algorithms

		TSIP-Lite*1	TSIP*2
Public key cryptography	Encryption/ decryption	-	RSAES-PKCS1-v1_5
	Signature generation/ verification	-	RSASSA-PKCS1-v1_5, ECDSA
	Key generation	-	RSA (1024/2048 bit), ECC P-192/224/256/384
Common key cryptography	AES	AES (128/256 bit) ECB/CBC/GCM/ CCM	AES (128/256 bit) ECB/CBC/GCM/CCM
	DES	-	Triple-DES (56/56x2/56x3 bit) ECB/CBC
	ARC4	-	ARC4 (2048 bit)
Hashing	SHA	-	SHA-1, SHA-256
	MD5	-	MD5
Message authentication		CMAC (AES), GMAC	CMAC (AES), GMAC, HMAC (SHA)
Pseudo-random bit generation		SP 800-90A	SP 800-90A
Random number generation		Tested with SP 800-22	Tested with SP 800-22
SSL/TLS cooperation function		-	TLS1.2, TLS1.3 compliant Supporting cipher suite is below: 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 Supporting cipher suite for TLS1.3 is below*3: TLS_AES_128_GCM_SHA256
Key update function		AES	AES, RSA, DES, ARC4, ECC, HMAC
Key exchange		-	ECDH P-256, ECDHE P-512, DH (2048 bit)
Key Wrap		AES (128/256 bit)	AES (128/256 bit)

Notes: 1. Applicable devices are the RX231 Group, RX23W Group, RX66T Group, and RX72T Group.
2. Applicable devices are the RX65N Group, RX651 Group, RX66N Group, RX671 Group, RX72M Group, and RX72N Group.
3. Applicable devices are the RX65N Group, RX651 Group.

The TSIP driver is provided as a Firmware Integration Technology (FIT) module. For an overview of FIT, refer to the URL below.

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

Target Devices

RX231 Group, RX23W Group, RX65N, RX651 Group, RX66T Group, RX671 Group, RX72M Group, RX72N Group, and RX72T Group

For information regarding the model names of products that have TSIP capability, refer to the user's manuals of the respective RX microcontrollers.

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/

Contents

1. Overview

1.1 Terminology

Terms used in this document are defined below. For terms related to keys, refer to “Key Installation Process” (reproduced below as Figure 1.1) in the section on TSIP or security functions of the hardware manual of the MCU.

Table 1.1 Terminology

Term	Description	Key Installation Process
user key	Under AES, DES, ARC4, and HMAC a common key set by the user. Under RSA, ECC, a public key or secret key set by the user.	Key-1
encrypted key	Key information generated by AES128-encrypting the user key using a provisioning key.	eKey-1
key index	Data consisting of key information, such as the user key, that has been converted into a form that is usable by the TSIP driver. The user key is converted into the key index.	Index-1 or Index-2
provisioning key	An AES128 common keyring set by the user and used to encrypt the user key with AES128 and add a MAC value.	Key-2
encrypted provisioning key	Key information used by the TSIP to decrypt an encrypted key and convert it into a key index. The encrypted provisioning key is wrapped provis key by DLM server.	Index-2
DLM server	The Renesas key management server. “DLM server” is short for “device lifecycle management server.” It is used for provisioning key wrapping.	-

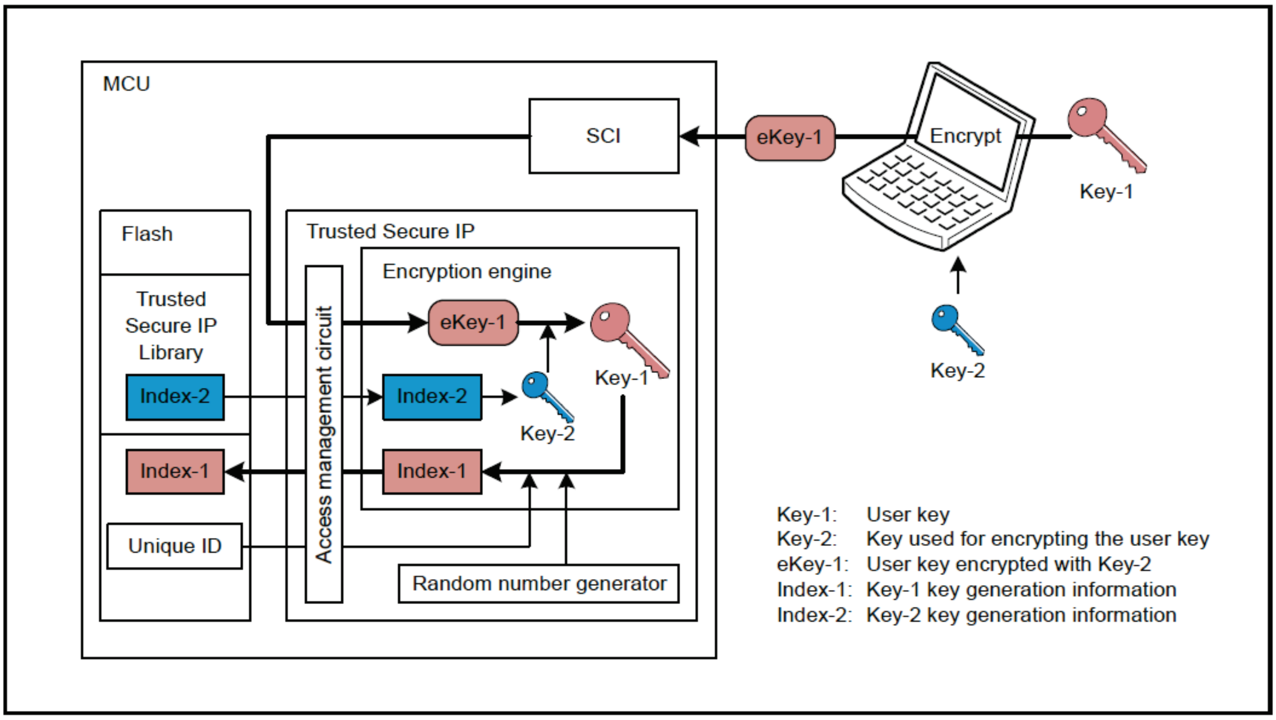


Figure 1.1 Key Installation Process
(RX65N Group, RX651 Group User's Manual: Hardware 52. Trusted Secure IP Figure 52.4)

1.2 Trusted Secure IP (TSIP)

The Trusted Secure IP (TSIP) block within the RX family creates a secure area inside the MCU by monitoring for unauthorized access attempts. It ensures that the encryption engine and encryption key can be utilized safely. The encryption key, the most important element in reliable and secure encryption, is linked to a unique ID and stored in the flash memory in a safe, undecipherable format.

Each TSIP devices include a safe area, which holds: an encryption engine, storage for raw keys, and a hidden root key, used to encrypt keys.

TSIP hardware generates Key Index from encrypted user key inside the TSIP which is device-specific, and tied to a unique ID. Hence, the key from one device will not work on a different device. The TSIP driver software allows applications access to the TSIP hardware.

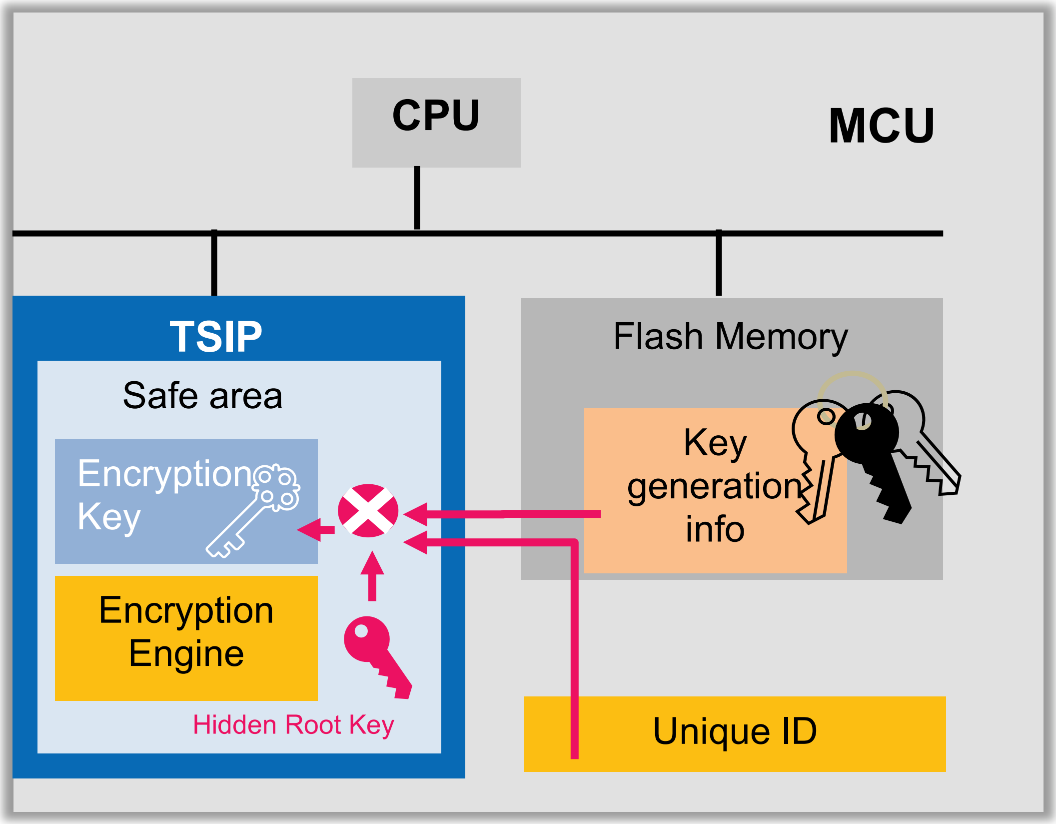


Figure 1.2 TSIP Hardware

1.3 Structure of Product Files

This product includes the files listed in Table1.2 below.

Table 1.2 Structure of Product Files

File/Directory **(Bold)** Names	Description
r20an0548ej0114-rx-tsip-security.pdf	TSIP driver Application Note (English)
r20an0548jj0114-rx-tsip-security.pdf	TSIP driver Application Note (Japanese)
reference_documents	Folder containing documentations such as how to use the FIT module with various integrated development environments
en	Folder containing documentations such as how to use the FIT module with various integrated development environments (English)
r01an1826ej0110-rx.pdf	How to add the FIT modules to CS+ Projects (English)
r01an1723eu0121-rx.pdf	How to add the FIT modules to e^2 studio Projects (English)
r20an0451es0140-e2studio-sc.pdf	Smart Configurator User Guide (English)
r01an5792ej0101-rx-tsip.pdf	Application note about how to use AES Cryptography with TSIP (English)
r01an5880ej0100-rx-tsip.pdf	Application note about how to implement TLS with TSIP (English)
ja	Folder containing documentations such as how to use the FIT module with various integrated development environments (Japanese)
r01an1826jj0110-rx.pdf	How to add the FIT modules to CS+ Projects (Japanese)
r01an1723ju0121-rx.pdf	How to add the FIT modules to e^2 studio Projects (Japanese)
r20an0451js0140-e2studio-sc.pdf	Smart Configurator User Guide (Japanese)
r01an5792jj0101-rx-tsip.pdf	Application note about how to use AES Cryptography with TSIP (Japanese)
r01an5880jj0100-rx-tsip.pdf	Application note about how to implement TLS with TSIP (Japanese)
FITModules	FIT module folder
r_tsip_rx_v1.14.l.zip	TSIP driver FIT Module
r_tsip_rx_v1.14.l.xml	TSIP driver FIT Module e^2 studio FIT plug-in XML file
r_tsip_rx_v1.14.l_extend.mdf	TSIP driver FIT Module Smart Configurator configuration file
FITDemos	Sample project folder
rx231_rsk_tsip_sample	RX231 project showing the methods for writing and updating keys
rx65n_2mb_rsk_tsip_sample	RX65N project showing the methods for writing and updating keys
rx66t_rsk_tsip_sample	RX66T project showing the methods for writing and updating keys
rx671_rsk_tsip_sample	RX671 project showing the methods for writing and updating keys
rx72m_rsk_tsip_sample	RX72M project showing the methods for writing and updating keys
rx72n_rsk_tsip_sample	RX72N project showing the methods for writing and updating keys
rx72t_rsk_tsip_sample	RX72T project showing the methods for writing and updating keys
rx65n_2mb_rsk_tsip_aes_sample	The sample indicates how to use AES cryptograpy in RX65N
rx72n_ek_tsip_aes_sample	The sample indicates how to use AES cryptograpy in RX72N
rx_tsip_freertos_mbedtls_sample	The sample indicates how to implement TLS

tool		
	Renesas Secure Flash Programmer.exe	The tool encrypts the key and user program.

1.4 Development Environment

The TSIP driver was developed using the environment shown 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 9.1, Confirmed Operation Environment.
2. C compiler
   Refer to the “C compiler” item under 9.1, Confirmed Operation Environment.
3. Emulator/debugger
   E1/E20/E2 Lite
4. Evaluation boards
   Refer to the “Board used” item under 9.1, Confirmed Operation Environment.
   All of the boards listed are special product versions with encryption 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 model 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 sizes of ROM, RAM and maximum stack usage associated with this module are listed below.

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

Module revision: r_tsip_rx rev1.14

Compiler version: Renesas Electronics C/C++ Compiler Package for RX Family V3.03.00
(integrated development environment default settings with “-lang = c99” option added)

GCC for Renesas RX 8.3.0.202102
(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)

ROM, RAM, and Stack Code Sizes				
Device	Category	Memory Used		
		Renesas Compiler	GCC	IAR Compiler
TSIP-Lite	ROM	54,779 bytes	55,310 bytes	54,150 bytes
	RAM	796 bytes	796 bytes	796 bytes
	STACK	184 bytes	-	164 bytes
TSIP	ROM	262,433 bytes	262,745 bytes	257,831 bytes
	RAM	1,176 bytes	1,176 bytes	1,176 bytes
	STACK	888 bytes	-	856 bytes

1.6 Sections

The TSIP driver uses the default sections.

1.7 Performance (RX231)

Information on the performance of the TSIP-Lite driver on the RX231 is shown below. Performance is measured using cycles of the core clock ICLK as the basic unit. The frequency of the TSIP-Lite operating clock PCLKB is set to ICLK:PCLKB = 2:1.

The Optimization level is level 2.

Table 1.3 Performance of each APIs

API	**Performance (Unit: cycle)**
R_TSIP_Open	7,359,124
R_TSIP_Close	448
R_TSIP_GetVersion	30
R_TSIP_GenerateAes128KeyIndex	3,966
R_TSIP_GenerateAes256KeyIndex	4,328
R_TSIP_GenerateAes128RandomKeyIndex	2,244
R_TSIP_GenerateAes256RandomKeyIndex	3,074
R_TSIP_GenerateRandomNumber	934
R_TSIP_GenerateUpdateKeyRingKeyIndex	4,322
R_TSIP_UpdateAes128KeyIndex	3,532
R_TSIP_UpdateAes256KeyIndex	3,880

Table 1.4 Performance of Firmware Verify APIs

API	Performance (Unit: cycle)		
	2 KB processing	4 KB processing	6 KB processing
R_TSIP_VerifyFirmwareMAC	12,004	23,266	34,528

Table 1.5 Performance of AES

API	Performance (Unit: cycle)		
	16-byte processing	48-byte processing	80-byte processing
R_TSIP_Aes128EcbEncryptInit	1,330	1,330	1,330
R_TSIP_Aes128EcbEncryptUpdate	610	788	960
R_TSIP_Aes128EcbEncryptFinal	550	550	550
R_TSIP_Aes128EcbDecryptInit	1,334	1,334	1,334
R_TSIP_Aes128EcbDecryptUpdate	728	906	1,078
R_TSIP_Aes128EcbDecryptFinal	566	566	566
R_TSIP_Aes256EcbEncryptInit	1,636	1,638	1,638
R_TSIP_Aes256EcbEncryptUpdate	662	900	1,142
R_TSIP_Aes256EcbEncryptFinal	556	556	556
R_TSIP_Aes256EcbDecryptInit	1,642	1,644	1,644
R_TSIP_Aes256EcbDecryptUpdate	802	1,040	1,292
R_TSIP_Aes256EcbDecryptFinal	574	574	574
R_TSIP_Aes128CbcEncryptInit	1,394	1,394	1,394
R_TSIP_Aes128CbcEncryptUpdate	682	860	1,032
R_TSIP_Aes128CbcEncryptFinal	578	578	578
R_TSIP_Aes128CbcDecryptInit	1,400	1,402	1,402
R_TSIP_Aes128CbcDecryptUpdate	798	976	1,148
R_TSIP_Aes128CbcDecryptFinal	592	592	592
R_TSIP_Aes256CbcEncryptInit	1,696	1,696	1,696
R_TSIP_Aes256CbcEncryptUpdate	732	970	1,212
R_TSIP_Aes256CbcEncryptFinal	582	582	582
R_TSIP_Aes256CbcDecryptInit	1,706	1,708	1,708
R_TSIP_Aes256CbcDecryptUpdate	874	1,112	1,364
R_TSIP_Aes256CbcDecryptFinal	592	592	592

Table 1.6 Performance of GCM

API	Performance (Unit: cycle)		
	48-byte processing	64-byte processing	80-byte processing
R_TSIP_Aes128GcmEncryptInit	5,462	5,462	5,462
R_TSIP_Aes128GcmEncryptUpdate	2,830	3,326	3,822
R_TSIP_Aes128GcmEncryptFinal	1,290	1,290	1,290
R_TSIP_Aes128GcmDecryptInit	5,454	5,456	5,456
R_TSIP_Aes128GcmDecryptUpdate	2,440	2,538	2,636
R_TSIP_Aes128GcmDecryptFinal	2,088	2,088	2,088
R_TSIP_Aes256GcmEncryptInit	6,158	6,160	6,160
R_TSIP_Aes256GcmEncryptUpdate	2,936	3,472	4,008
R_TSIP_Aes256GcmEncryptFinal	1,320	1,320	1,320
R_TSIP_Aes256GcmDecryptInit	6,158	6,160	6,160
R_TSIP_Aes256GcmDecryptUpdate	2,526	2,644	2,762
R_TSIP_Aes256GcmDecryptFinal	2,116	2,116	2,116

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	2,596	2,596	2,596
R_TSIP_Aes128CcmEncryptUpdate	1,518	1,686	1,864
R_TSIP_Aes128CcmEncryptFinal	1,178	1,178	1,178
R_TSIP_Aes128CcmDecryptInit	2,414	2,416	2,416
R_TSIP_Aes128CcmDecryptUpdate	1,430	1,598	1,776
R_TSIP_Aes128CcmDecryptFinal	1,928	1,928	1,928
R_TSIP_Aes256CcmEncryptInit	2,982	2,982	2,982
R_TSIP_Aes256CcmEncryptUpdate	1,734	1,982	2,220
R_TSIP_Aes256CcmEncryptFinal	1,216	1,216	1,216
R_TSIP_Aes256CcmDecryptInit	2,980	2,980	2,980
R_TSIP_Aes256CcmDecryptUpdate	1,646	1,894	2,132
R_TSIP_Aes256CcmDecryptFinal	1,972	1,972	1,972

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

Table 1.8 Performance of MAC (AES-CMAC)

API	Performance (Unit: cycle)		
	48-byte processing	64-byte processing	80-byte processing
R_TSIP_Aes128CmacGenerateInit	906	906	906
R_TSIP_Aes128CmacGenerateUpdate	804	892	980
R_TSIP_Aes128CmacGenerateFinal	1,088	1,088	1,088
R_TSIP_Aes128CmacVerifyInit	902	906	906
R_TSIP_Aes128CmacVerifyUpdate	806	890	978
R_TSIP_Aes128CmacVerifyFinal	1,784	1,784	1,784
R_TSIP_Aes256CmacGenerateInit	1,220	1,226	1,226
R_TSIP_Aes256CmacGenerateUpdate	872	1,000	1,118
R_TSIP_Aes256CmacGenerateFinal	1,156	1,156	1,156
R_TSIP_Aes256CmacVerifyInit	1,220	1,224	1,224
R_TSIP_Aes256CmacVerifyUpdate	878	1,002	1,130
R_TSIP_Aes256CmacVerifyFinal	1,852	1,852	1,852

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	9,538	15,264
R_TSIP_Aes256KeyWrap	10,330	16,536
R_TSIP_Aes128KeyUnwrap	11,922	17,680
R_TSIP_Aes256KeyUnwrap	12,694	18,932

1.8 Performance (RX23W)

Information on the performance of the TSIP-Lite driver on the RX23W is shown below. Performance is measured using cycles of the core clock ICLK as the basic unit. The frequency of the TSIP-Lite operating clock PCLKB is set to ICLK:PCLKB = 2:1.

The Optimization level is level 2.

Table 1.10 Performance of each APIs

API	**Performance (Unit: cycle)**
R_TSIP_Open	7,360,036
R_TSIP_Close	432
R_TSIP_GetVersion	30
R_TSIP_GenerateAes128KeyIndex	3,974
R_TSIP_GenerateAes256KeyIndex	4,334
R_TSIP_GenerateAes128RandomKeyIndex	2,248
R_TSIP_GenerateAes256RandomKeyIndex	3,054
R_TSIP_GenerateRandomNumber	930
R_TSIP_GenerateUpdateKeyRingKeyIndex	4,328
R_TSIP_UpdateAes128KeyIndex	3,526
R_TSIP_UpdateAes256KeyIndex	3,876

Table 1.11 Performance of Firmware Verify APIs

API	Performance (Unit: cycle)		
	2 KB processing	4 KB processing	6 KB processing
R_TSIP_VerifyFirmwareMAC	11,998	23,270	34,532

Table 1.12 Performance of AES

API	Performance (Unit: cycle)		
	16-byte processing	48-byte processing	80-byte processing
R_TSIP_Aes128EcbEncryptInit	1,314	1,314	1,314
R_TSIP_Aes128EcbEncryptUpdate	612	790	962
R_TSIP_Aes128EcbEncryptFinal	558	558	558
R_TSIP_Aes128EcbDecryptInit	1,324	1,324	1,324
R_TSIP_Aes128EcbDecryptUpdate	730	908	1,080
R_TSIP_Aes128EcbDecryptFinal	574	574	574
R_TSIP_Aes256EcbEncryptInit	1,622	1,624	1,624
R_TSIP_Aes256EcbEncryptUpdate	648	896	1,138
R_TSIP_Aes256EcbEncryptFinal	564	564	564
R_TSIP_Aes256EcbDecryptInit	1,630	1,632	1,632
R_TSIP_Aes256EcbDecryptUpdate	806	1,044	1,286
R_TSIP_Aes256EcbDecryptFinal	576	576	576
R_TSIP_Aes128CbcEncryptInit	1,380	1,380	1,380
R_TSIP_Aes128CbcEncryptUpdate	680	858	1,030
R_TSIP_Aes128CbcEncryptFinal	584	584	584
R_TSIP_Aes128CbcDecryptInit	1,390	1,392	1,392
R_TSIP_Aes128CbcDecryptUpdate	798	976	1,148
R_TSIP_Aes128CbcDecryptFinal	598	598	598
R_TSIP_Aes256CbcEncryptInit	1,692	1,692	1,692
R_TSIP_Aes256CbcEncryptUpdate	722	970	1,212
R_TSIP_Aes256CbcEncryptFinal	588	588	588
R_TSIP_Aes256CbcDecryptInit	1,696	1,698	1,698
R_TSIP_Aes256CbcDecryptUpdate	874	1,112	1,354
R_TSIP_Aes256CbcDecryptFinal	600	600	600

Table 1.13 Performance of GCM

API	Performance (Unit: cycle)		
	48-byte processing	64-byte processing	80-byte processing
R_TSIP_Aes128GcmEncryptInit	5,460	5,460	5,460
R_TSIP_Aes128GcmEncryptUpdate	2,850	3,348	3,846
R_TSIP_Aes128GcmEncryptFinal	1,288	1,288	1,288
R_TSIP_Aes128GcmDecryptInit	5,470	5,472	5,472
R_TSIP_Aes128GcmDecryptUpdate	2,428	2,526	2,624
R_TSIP_Aes128GcmDecryptFinal	2,082	2,082	2,082
R_TSIP_Aes256GcmEncryptInit	6,162	6,164	6,164
R_TSIP_Aes256GcmEncryptUpdate	2,954	3,490	4,026
R_TSIP_Aes256GcmEncryptFinal	1,328	1,328	1,328
R_TSIP_Aes256GcmDecryptInit	6,176	6,178	6,178
R_TSIP_Aes256GcmDecryptUpdate	2,536	2,654	2,772
R_TSIP_Aes256GcmDecryptFinal	2,114	2,114	2,114

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	2,596	2,596	2,596
R_TSIP_Aes128CcmEncryptUpdate	1,524	1,702	1,880
R_TSIP_Aes128CcmEncryptFinal	1,172	1,172	1,172
R_TSIP_Aes128CcmDecryptInit	2,408	2,410	2,410
R_TSIP_Aes128CcmDecryptUpdate	1,430	1,608	1,786
R_TSIP_Aes128CcmDecryptFinal	1,932	1,932	1,932
R_TSIP_Aes256CcmEncryptInit	2,978	2,978	2,978
R_TSIP_Aes256CcmEncryptUpdate	1,730	1,978	2,216
R_TSIP_Aes256CcmEncryptFinal	1,208	1,208	1,208
R_TSIP_Aes256CcmDecryptInit	2,982	2,982	2,982
R_TSIP_Aes256CcmDecryptUpdate	1,630	1,878	2,116
R_TSIP_Aes256CcmDecryptFinal	1,962	1,962	1,962

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

Table 1.15 Performance of MAC (AES-CMAC)

API	Performance (Unit: cycle)		
	48-byte processing	64-byte processing	80-byte processing
R_TSIP_Aes128CmacGenerateInit	902	902	902
R_TSIP_Aes128CmacGenerateUpdate	804	892	980
R_TSIP_Aes128CmacGenerateFinal	1,078	1,078	1,078
R_TSIP_Aes128CmacVerifyInit	896	900	900
R_TSIP_Aes128CmacVerifyUpdate	798	888	976
R_TSIP_Aes128CmacVerifyFinal	1,782	1,782	1,782
R_TSIP_Aes256CmacGenerateInit	1,214	1,220	1,220
R_TSIP_Aes256CmacGenerateUpdate	884	1,012	1,130
R_TSIP_Aes256CmacGenerateFinal	1,154	1,154	1,154
R_TSIP_Aes256CmacVerifyInit	1,212	1,216	1,216
R_TSIP_Aes256CmacVerifyUpdate	882	1,012	1,130
R_TSIP_Aes256CmacVerifyFinal	1,860	1,860	1,860

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	9,542	15,266
R_TSIP_Aes256KeyWrap	10,316	16,520
R_TSIP_Aes128KeyUnwrap	11,938	17,696
R_TSIP_Aes256KeyUnwrap	12,698	18,936

1.9 Performance (RX66T)

Information on the performance of the TSIP-Lite driver on the RX66T is shown below. Performance is measured using cycles of the core clock ICLK as the basic unit. The frequency of the TSIP-Lite operating clock PCLKB is set to ICLK:PCLKB = 2:1.

The Optimization level is level 2.

Table 1.17 Performance of each APIs

API	Performance (Unit: cycle)
R_TSIP_Open	7,353,142
R_TSIP_Close	288
R_TSIP_GetVersion	24
R_TSIP_GenerateAes128KeyIndex	3,890
R_TSIP_GenerateAes256KeyIndex	4,240
R_TSIP_GenerateAes128RandomKeyIndex	2,178
R_TSIP_GenerateAes256RandomKeyIndex	2,980
R_TSIP_GenerateRandomNumber	900
R_TSIP_GenerateUpdateKeyRingKeyIndex	4,242
R_TSIP_UpdateAes128KeyIndex	3,458
R_TSIP_UpdateAes256KeyIndex	3,804

Table 1.18 Performance of Firmware Verify APIs

API	Performance (Unit: cycle)		
	2 KB processing	4 KB processing	6 KB processing
R_TSIP_VerifyFirmwareMAC	11,940	23,202	34,466

Table 1.19 Performance of AES

API	Performance (Unit: cycle)		
	16-byte processing	48-byte processing	80-byte processing
R_TSIP_Aes128EcbEncryptInit	1,284	1,276	1,278
R_TSIP_Aes128EcbEncryptUpdate	560	740	916
R_TSIP_Aes128EcbEncryptFinal	512	508	508
R_TSIP_Aes128EcbDecryptInit	1,284	1,284	1,284
R_TSIP_Aes128EcbDecryptUpdate	666	846	1,022
R_TSIP_Aes128EcbDecryptFinal	516	516	516
R_TSIP_Aes256EcbEncryptInit	1,592	1,590	1,588
R_TSIP_Aes256EcbEncryptUpdate	606	848	1,088
R_TSIP_Aes256EcbEncryptFinal	510	510	510
R_TSIP_Aes256EcbDecryptInit	1,598	1,600	1,600
R_TSIP_Aes256EcbDecryptUpdate	746	990	1,230
R_TSIP_Aes256EcbDecryptFinal	520	520	520
R_TSIP_Aes128CbcEncryptInit	1,336	1,332	1,334
R_TSIP_Aes128CbcEncryptUpdate	614	796	972
R_TSIP_Aes128CbcEncryptFinal	528	528	528
R_TSIP_Aes128CbcDecryptInit	1,342	1,342	1,342
R_TSIP_Aes128CbcDecryptUpdate	722	904	1,080
R_TSIP_Aes128CbcDecryptFinal	540	540	540
R_TSIP_Aes256CbcEncryptInit	1,648	1,646	1,648
R_TSIP_Aes256CbcEncryptUpdate	668	910	1,150
R_TSIP_Aes256CbcEncryptFinal	532	532	532
R_TSIP_Aes256CbcDecryptInit	1,658	1,656	1,656
R_TSIP_Aes256CbcDecryptUpdate	810	1,058	1,298
R_TSIP_Aes256CbcDecryptFinal	542	542	542

Table 1.20 Performance of AES-GCM

API	Performance (Unit: cycle)		
	48-byte processing	64-byte processing	80-byte processing
R_TSIP_Aes128GcmEncryptInit	5,112	5,108	5,110
R_TSIP_Aes128GcmEncryptUpdate	2,580	3,070	3,558
R_TSIP_Aes128GcmEncryptFinal	1,244	1,240	1,238
R_TSIP_Aes128GcmDecryptInit	5,110	5,112	5,112
R_TSIP_Aes128GcmDecryptUpdate	2,204	2,298	2,386
R_TSIP_Aes128GcmDecryptFinal	2,010	2,008	2,008
R_TSIP_Aes256GcmEncryptInit	5,812	5,816	5,818
R_TSIP_Aes256GcmEncryptUpdate	2,698	3,218	3,740
R_TSIP_Aes256GcmEncryptFinal	1,276	1,278	1,278
R_TSIP_Aes256GcmDecryptInit	5,832	5,832	5,834
R_TSIP_Aes256GcmDecryptUpdate	2,304	2,426	2,546
R_TSIP_Aes256GcmDecryptFinal	2,060	2,060	2,060

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	2,446	2,446	2,446
R_TSIP_Aes128CcmEncryptUpdate	1,454	1,632	1,808
R_TSIP_Aes128CcmEncryptFinal	1,134	1,134	1,134
R_TSIP_Aes128CcmDecryptInit	2,240	2,240	2,240
R_TSIP_Aes128CcmDecryptUpdate	1,346	1,522	1,698
R_TSIP_Aes128CcmDecryptFinal	1,874	1,870	1,872
R_TSIP_Aes256CcmEncryptInit	2,818	2,816	2,814
R_TSIP_Aes256CcmEncryptUpdate	1,656	1,904	2,144
R_TSIP_Aes256CcmEncryptFinal	1,176	1,174	1,174
R_TSIP_Aes256CcmDecryptInit	2,810	2,808	2,808
R_TSIP_Aes256CcmDecryptUpdate	1,558	1,806	2,046
R_TSIP_Aes256CcmDecryptFinal	1,918	1,912	1,912

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	872	872	872
R_TSIP_Aes128CmacGenerateUpdate	718	808	898
R_TSIP_Aes128CmacGenerateFinal	1,024	1,022	1,022
R_TSIP_Aes128CmacVerifyInit	874	874	872
R_TSIP_Aes128CmacVerifyUpdate	716	806	896
R_TSIP_Aes128CmacVerifyFinal	1,716	1,714	1,714
R_TSIP_Aes256CmacGenerateInit	1,186	1,182	1,182
R_TSIP_Aes256CmacGenerateUpdate	790	916	1,036
R_TSIP_Aes256CmacGenerateFinal	1,098	1,094	1,094
R_TSIP_Aes256CmacVerifyInit	1,182	1,182	1,182
R_TSIP_Aes256CmacVerifyUpdate	788	916	1,036
R_TSIP_Aes256CmacVerifyFinal	1,788	1,786	1,786

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	9,354	15,000
R_TSIP_Aes256KeyWrap	10,046	16,076
R_TSIP_Aes128KeyUnwrap	11,668	17,352
R_TSIP_Aes256KeyUnwrap	12,396	18,466

1.10 Performance (RX72T)

Information on the performance of the TSIP-Lite driver on the RX72T is shown below. Performance is measured using cycles of the core clock ICLK as the basic unit. The frequency of the TSIP-Lite operating clock PCLKB is set to ICLK:PCLKB = 2:1.

The Optimization level is level 2.

Table 1.24 Performance of each APIs

API	**Performance (Unit: cycle)**
R_TSIP_Open	7,354,942
R_TSIP_Close	286
R_TSIP_GetVersion	20
R_TSIP_GenerateAes128KeyIndex	3,906
R_TSIP_GenerateAes256KeyIndex	4,260
R_TSIP_GenerateAes128RandomKeyIndex	2,182
R_TSIP_GenerateAes256RandomKeyIndex	2,982
R_TSIP_GenerateRandomNumber	908
R_TSIP_GenerateUpdateKeyRingKeyIndex	4,252
R_TSIP_UpdateAes128KeyIndex	3,460
R_TSIP_UpdateAes256KeyIndex	3,814

Table 1.25 Performance of Firmware Verify APIs

API	Performance (Unit: cycle)		
	2 KB processing	4 KB processing	6 KB processing
R_TSIP_VerifyFirmwareMAC	11,944	23,204	34,468

Table 1.26 Performance of AES

API	Performance (Unit: cycle)		
	16-byte processing	48-byte processing	80-byte processing
R_TSIP_Aes128EcbEncryptInit	1,290	1,282	1,280
R_TSIP_Aes128EcbEncryptUpdate	558	742	918
R_TSIP_Aes128EcbEncryptFinal	512	510	510
R_TSIP_Aes128EcbDecryptInit	1,288	1,286	1,286
R_TSIP_Aes128EcbDecryptUpdate	668	850	1,026
R_TSIP_Aes128EcbDecryptFinal	522	522	522
R_TSIP_Aes256EcbEncryptInit	1,594	1,592	1,590
R_TSIP_Aes256EcbEncryptUpdate	606	852	1,092
R_TSIP_Aes256EcbEncryptFinal	514	514	514
R_TSIP_Aes256EcbDecryptInit	1,600	1,602	1,600
R_TSIP_Aes256EcbDecryptUpdate	750	994	1,234
R_TSIP_Aes256EcbDecryptFinal	526	526	526
R_TSIP_Aes128CbcEncryptInit	1,340	1,338	1,338
R_TSIP_Aes128CbcEncryptUpdate	620	802	978
R_TSIP_Aes128CbcEncryptFinal	532	532	532
R_TSIP_Aes128CbcDecryptInit	1,344	1,344	1,344
R_TSIP_Aes128CbcDecryptUpdate	724	906	1,082
R_TSIP_Aes128CbcDecryptFinal	544	542	542
R_TSIP_Aes256CbcEncryptInit	1,650	1,648	1,648
R_TSIP_Aes256CbcEncryptUpdate	668	912	1,152
R_TSIP_Aes256CbcEncryptFinal	538	538	538
R_TSIP_Aes256CbcDecryptInit	1,660	1,660	1,658
R_TSIP_Aes256CbcDecryptUpdate	822	1,066	1,306
R_TSIP_Aes256CbcDecryptFinal	550	548	548

Table 1.27 Performance of GCM

API	Performance (Unit: cycle)		
	48-byte processing	64-byte processing	80-byte processing
R_TSIP_Aes128GcmEncryptInit	5,122	5,120	5,120
R_TSIP_Aes128GcmEncryptUpdate	2,590	3,080	3,570
R_TSIP_Aes128GcmEncryptFinal	1,244	1,240	1,238
R_TSIP_Aes128GcmDecryptInit	5,126	5,132	5,132
R_TSIP_Aes128GcmDecryptUpdate	2,202	2,292	2,380
R_TSIP_Aes128GcmDecryptFinal	2,014	2,014	2,014
R_TSIP_Aes256GcmEncryptInit	5,840	5,838	5,840
R_TSIP_Aes256GcmEncryptUpdate	2,696	3,216	3,738
R_TSIP_Aes256GcmEncryptFinal	1,288	1,286	1,286
R_TSIP_Aes256GcmDecryptInit	5,836	5,832	5,832
R_TSIP_Aes256GcmDecryptUpdate	2,302	2,422	2,542
R_TSIP_Aes256GcmDecryptFinal	2,050	2,050	2,050

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	2,452	2,452	2,452
R_TSIP_Aes128CcmEncryptUpdate	1,452	1,628	1,804
R_TSIP_Aes128CcmEncryptFinal	1,136	1,136	1,136
R_TSIP_Aes128CcmDecryptInit	2,246	2,248	2,248
R_TSIP_Aes128CcmDecryptUpdate	1,350	1,526	1,702
R_TSIP_Aes128CcmDecryptFinal	1,876	1,876	1,874
R_TSIP_Aes256CcmEncryptInit	2,818	2,818	2,816
R_TSIP_Aes256CcmEncryptUpdate	1,660	1,906	2,146
R_TSIP_Aes256CcmEncryptFinal	1,180	1,176	1,176
R_TSIP_Aes256CcmDecryptInit	2,812	2,810	2,810
R_TSIP_Aes256CcmDecryptUpdate	1,566	1,814	2,054
R_TSIP_Aes256CcmDecryptFinal	1,920	1,916	1,916

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

Table 1.29 Performance of MAC (AES-CMAC)

API	Performance (Unit: cycle)		
	48-byte processing	64-byte processing	80-byte processing
R_TSIP_Aes128CmacGenerateInit	886	886	886
R_TSIP_Aes128CmacGenerateUpdate	722	810	898
R_TSIP_Aes128CmacGenerateFinal	1,028	1,026	1,026
R_TSIP_Aes128CmacVerifyInit	882	882	882
R_TSIP_Aes128CmacVerifyUpdate	722	808	896
R_TSIP_Aes128CmacVerifyFinal	1,712	1,712	1,712
R_TSIP_Aes256CmacGenerateInit	1,190	1,188	1,188
R_TSIP_Aes256CmacGenerateUpdate	792	918	1,038
R_TSIP_Aes256CmacGenerateFinal	1,098	1,096	1,096
R_TSIP_Aes256CmacVerifyInit	1,186	1,186	1,186
R_TSIP_Aes256CmacVerifyUpdate	790	918	1,038
R_TSIP_Aes256CmacVerifyFinal	1,788	1,788	1,788

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	9,354	15,002
R_TSIP_Aes256KeyWrap	10,034	16,064
R_TSIP_Aes128KeyUnwrap	11,680	17,366
R_TSIP_Aes256KeyUnwrap	12,394	18,466

1.11 Performance (RX65N)

Information on the performance of the TSIP driver on the RX65N is shown below. Performance is measured using cycles of the core clock ICLK as the basic unit. The frequency of the TSIP operating clock PCLKB is set to ICLK:PCLKB = 2:1.

The Optimization level is level 2.

Table 1.31 Performance of each APIs

API	**Performance (Unit: cycle)**
R_TSIP_Open	5,681,682
R_TSIP_Close	444
R_TSIP_GetVersion	34
R_TSIP_GenerateAes128KeyIndex	2,610
R_TSIP_GenerateAes256KeyIndex	2,732
R_TSIP_GenerateAes128RandomKeyIndex	1,460
R_TSIP_GenerateAes256RandomKeyIndex	1,994
R_TSIP_GenerateRandomNumber	650
R_TSIP_GenerateUpdateKeyRingKeyIndex	2,750
R_TSIP_UpdateAes128KeyIndex	2,232
R_TSIP_UpdateAes256KeyIndex	2,356

Table 1.32 Performance of Firmware Verify APIs

API	Performance (Unit: cycle)		
	8 KB processing	16 KB processing	24 KB processing
R_TSIP_VerifyFirmwareMAC	21,040	41,520	62,000

Table 1.33 Performance of AES

API	Performance (Unit: cycle)		
	16-byte processing	48-byte processing	80-byte processing
R_TSIP_Aes128EcbEncryptInit	1,598	1,600	1,602
R_TSIP_Aes128EcbEncryptUpdate	506	652	832
R_TSIP_Aes128EcbEncryptFinal	436	436	436
R_TSIP_Aes128EcbDecryptInit	1,610	1,612	1,612
R_TSIP_Aes128EcbDecryptUpdate	576	718	900
R_TSIP_Aes128EcbDecryptFinal	462	462	460
R_TSIP_Aes256EcbEncryptInit	1,746	1,744	1,744
R_TSIP_Aes256EcbEncryptUpdate	524	674	854
R_TSIP_Aes256EcbEncryptFinal	428	428	428
R_TSIP_Aes256EcbDecryptInit	1,760	1,758	1,758
R_TSIP_Aes256EcbDecryptUpdate	602	750	930
R_TSIP_Aes256EcbDecryptFinal	448	448	448
R_TSIP_Aes128CbcEncryptInit	1,668	1,670	1,670
R_TSIP_Aes128CbcEncryptUpdate	596	740	920
R_TSIP_Aes128CbcEncryptFinal	468	470	470
R_TSIP_Aes128CbcDecryptInit	1,694	1,694	1,694
R_TSIP_Aes128CbcDecryptUpdate	652	794	974
R_TSIP_Aes128CbcDecryptFinal	482	482	482
R_TSIP_Aes256CbcEncryptInit	1,818	1,818	1,816
R_TSIP_Aes256CbcEncryptUpdate	610	762	942
R_TSIP_Aes256CbcEncryptFinal	464	462	464
R_TSIP_Aes256CbcDecryptInit	1,840	1,842	1,842
R_TSIP_Aes256CbcDecryptUpdate	678	826	1,006
R_TSIP_Aes256CbcDecryptFinal	480	480	480

Table 1.34 Performance of GCM

API	Performance (Unit: cycle)		
	48-byte processing	64-byte processing	80-byte processing
R_TSIP_Aes128GcmEncryptInit	5,236	5,236	5,236
R_TSIP_Aes128GcmEncryptUpdate	2,064	2,170	2,258
R_TSIP_Aes128GcmEncryptFinal	1,062	1,062	1,060
R_TSIP_Aes128GcmDecryptInit	5,252	5,250	5,252
R_TSIP_Aes128GcmDecryptUpdate	2,040	2,134	2,222
R_TSIP_Aes128GcmDecryptFinal	1,958	1,958	1,960
R_TSIP_Aes256GcmEncryptInit	5,402	5,402	5,402
R_TSIP_Aes256GcmEncryptUpdate	2,088	2,204	2,292
R_TSIP_Aes256GcmEncryptFinal	1,078	1,078	1,078
R_TSIP_Aes256GcmDecryptInit	5,396	5,396	5,398
R_TSIP_Aes256GcmDecryptUpdate	2,066	2,170	2,258
R_TSIP_Aes256GcmDecryptFinal	1,976	1,976	1,976

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	2,494	2,492	2,494
R_TSIP_Aes128CcmEncryptUpdate	1,118	1,218	1,308
R_TSIP_Aes128CcmEncryptFinal	958	958	958
R_TSIP_Aes128CcmDecryptInit	2,226	2,226	2,228
R_TSIP_Aes128CcmDecryptUpdate	1,040	1,130	1,218
R_TSIP_Aes128CcmDecryptFinal	2,014	2,012	2,014
R_TSIP_Aes256CcmEncryptInit	2,352	2,352	2,350
R_TSIP_Aes256CcmEncryptUpdate	1,166	1,264	1,352
R_TSIP_Aes256CcmEncryptFinal	984	984	984
R_TSIP_Aes256CcmDecryptInit	2,362	2,362	2,362
R_TSIP_Aes256CcmDecryptUpdate	1,074	1,160	1,250
R_TSIP_Aes256CcmDecryptFinal	2,024	2,024	2,024

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

Table 1.36 Performance of MAC (AES-CMAC)

API	Performance (Unit: cycle)		
	48-byte processing	64-byte processing	80-byte processing
R_TSIP_Aes128CmacGenerateInit	1,136	1,136	1,136
R_TSIP_Aes128CmacGenerateUpdate	672	718	762
R_TSIP_Aes128CmacGenerateFinal	788	788	788
R_TSIP_Aes128CmacVerifyInit	1,132	1,132	1,132
R_TSIP_Aes128CmacVerifyUpdate	672	716	760
R_TSIP_Aes128CmacVerifyFinal	1,666	1,666	1,666
R_TSIP_Aes256CmacGenerateInit	1,278	1,282	1,282
R_TSIP_Aes256CmacGenerateUpdate	688	732	776
R_TSIP_Aes256CmacGenerateFinal	812	812	812
R_TSIP_Aes256CmacVerifyInit	1,276	1,276	1,274
R_TSIP_Aes256CmacVerifyUpdate	690	736	780
R_TSIP_Aes256CmacVerifyFinal	1,688	1,688	1,688

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	8,254	12,982
R_TSIP_Aes256KeyWrap	8,406	13,134
R_TSIP_Aes128KeyUnwrap	9,288	13,970
R_TSIP_Aes256KeyUnwrap	9,426	14,106

Table 1.38 Performance of Common API (TDES User Key Index Generation)

API	Performance (Unit: cycle)
R_TSIP_GenerateTdesKeyIndex	2,736
R_TSIP_GenerateTdesRandomKeyIndex	2,040
R_TSIP_UpdateTdesKeyIndex	2,372

Table 1.39 Performance of TDES

API	Performance (Unit: cycle)		
	16-byte processing	48-byte processing	80-byte processing
R_TSIP_TdesEcbEncryptInit	1,046	1,046	1,046
R_TSIP_TdesEcbEncryptUpdate	546	790	1,030
R_TSIP_TdesEcbEncryptFinal	426	426	426
R_TSIP_TdesEcbDecryptInit	1,046	1,046	1,044
R_TSIP_TdesEcbDecryptUpdate	580	824	1,064
R_TSIP_TdesEcbDecryptFinal	444	446	444
R_TSIP_TdesCbcEncryptInit	1,108	1,112	1,112
R_TSIP_TdesCbcEncryptUpdate	628	870	1,110
R_TSIP_TdesCbcEncryptFinal	456	456	456
R_TSIP_TdesCbcDecryptInit	1,128	1,126	1,126
R_TSIP_TdesCbcDecryptUpdate	660	904	1,142
R_TSIP_TdesCbcDecryptFinal	474	474	474

Table 1.40 Performance of Common API (RSA User Key Index Generation)

API	Performance (Unit: cycle)
R_TSIP_GenerateRsa1024PublicKeyIndex	37,542
R_TSIP_GenerateRsa1024PrivateKeyIndex	38,612
R_TSIP_GenerateRsa2048PublicKeyIndex	137,516
R_TSIP_GenerateRsa2048PrivateKeyIndex	139,702
R_TSIP_GenerateRsa1024RandomKeyIndex *1	44,838,940
R_TSIP_GenerateRsa2048RandomKeyIndex *1	244,982,063
R_TSIP_UpdateRsa1024PublicKeyIndex	37,174
R_TSIP_UpdateRsa1024PrivateKeyIndex	38,244
R_TSIP_UpdateRsa2048PublicKeyIndex	137,138
R_TSIP_UpdateRsa2048PrivateKeyIndex	139,286

Note 1. Average value at 10 runs.

Table 1.41 Performance of RSASSA-PKCS-v1_5 Signature Generation/Verification (HASH = SHA1)

API	Performance (Unit: cycle)		
	Message size=1byte	Message size=128byte	Message size=256byte
R_TSIP_RsassaPkcs1024SignatureGenerate	1,266,386	1,267,802	1,268,282
R_TSIP_RsassaPkcs1024SignatureVerification	17,236	18,652	19,132
R_TSIP_RsassaPkcs2048SignatureGenerate	26,227,128	26,228,544	26,229,024
R_TSIP_RsassaPkcs2048SignatureVerification	135,572	136,988	137,470

Table 1.42 Performance of RSASSA-PKCS-v1_5 Signature Generation/Verification (HASH = SHA256)

API	Performance (Unit: cycle)		
	Message size=1byte	Message size=128byte	Message size=256byte
R_TSIP_RsassaPkcs1024SignatureGenerate	1,266,468	1,267,944	1,268,352
R_TSIP_RsassaPkcs1024SignatureVerification	17,320	18,796	19,204
R_TSIP_RsassaPkcs2048SignatureGenerate	26,227,220	26,228,696	26,229,104
R_TSIP_RsassaPkcs2048SignatureVerification	135,658	137,136	137,542

Table 1.43 Performance of RSASSA-PKCS-v1_5 Signature Generation/Verification (HASH = MD5)

API	Performance (Unit: cycle)		
	Message size=1byte	Message size=128byte	Message size=256byte
R_TSIP_RsassaPkcs1024SignatureGenerate	1,266,338	1,267,670	1,268,078
R_TSIP_RsassaPkcs1024SignatureVerification	17,200	18,518	18,926
R_TSIP_RsassaPkcs2048SignatureGenerate	26,227,092	26,228,412	26,228,820
R_TSIP_RsassaPkcs2048SignatureVerification	135,534	136,854	137,262

Table 1.44 Performance of RSAES-PKCS1-v1_5 Encryption/Decryption with 1,024-Bit Key Size

API	Performance (Unit: cycle)	
	Message size=1byte	Message size=117byte
R_TSIP_RsaesPkcs1024Encrypt	22,264	16,832
R_TSIP_RsaesPkcs1024Decrypt	1,265,488	1,265,488

Table 1.45 Performance of RSAES-PKCS1-v1_5 Encryption/Decryption with 2,048-Bit Key Size

API	Performance (Unit: cycle)	
	Message size=1byte	Message size=245byte
R_TSIP_RsaesPkcs2048Encrypt	146,720	135,154
R_TSIP_RsaesPkcs2048Decrypt	26,226,442	26,226,444

Table 1.46 Performance of HASH (SHA1)

API	Performance (Unit: cycle)		
	128-byte processing	192-byte processing	256-byte processing
R_TSIP_Sha1Init	128	128	128
R_TSIP_Sha1Update	1,510	1,750	1,990
R_TSIP_Sha1Final	822	822	822

Table 1.47 Performance of HASH (SHA256)

API	Performance (Unit: cycle)		
	128-byte processing	192-byte processing	256-byte processing
R_TSIP_Sha256Init	180	182	182
R_TSIP_Sha256Update	1,556	1,760	1,964
R_TSIP_Sha256Final	838	838	838

Table 1.48 Performance of HASH (MD5)

API	Performance (Unit: cycle)		
	128-byte processing	192-byte processing	256-byte processing
R_TSIP_Md5Init	124	124	124
R_TSIP_Md5Update	1,404	1,608	1,812
R_TSIP_Md5Final	776	774	774

Table 1.49 Performance of Common API (HMAC User Key Index Generation)

API	Performance (Unit: cycle)
R_TSIP_GenerateSha1HmacKeyIndex	2,950
R_TSIP_GenerateSha256HmacKeyIndex	2,950
R_TSIP_UpdateSha1HmacKeyIndex	2,586
R_TSIP_UpdateSha256HmacKeyIndex	2,580

Table 1.50 Performance of HMAC (SHA1)

API	Performance (Unit: cycle)		
	128-byte processing	192-byte processing	256-byte processing
R_TSIP_Sha1HmacGenerateInit	1,358	1,356	1,356
R_TSIP_Sha1HmacGenerateUpdate	964	1,204	1,444
R_TSIP_Sha1HmacGenerateFinal	1,972	1,972	1,972
R_TSIP_Sha1HmacVerifyInit	1,352	1,352	1,352
R_TSIP_Sha1HmacVerifyUpdate	966	1,206	1,446
R_TSIP_Sha1HmacVerifyFinal	3,620	3,620	3,620

Table 1.51 Performance of HMAC (SHA256)

API	Performance (Unit: cycle)		
	128-byte processing	192-byte processing	256-byte processing
R_TSIP_Sha256HmacGenerateInit	1,622	1,624	1,624
R_TSIP_Sha256HmacGenerateUpdate	912	1,116	1,320
R_TSIP_Sha256HmacGenerateFinal	1,950	1,950	1,950
R_TSIP_Sha256HmacVerifyInit	1,624	1,624	1,624
R_TSIP_Sha256HmacVerifyUpdate	904	1,108	1,310
R_TSIP_Sha256HmacVerifyFinal	3,590	3,590	3,588

Table 1.52 Performance of Common API (ECC User Key Index Generation)

API	Performance (Unit: cycle)
R_TSIP_GenerateEccP192PublicKeyIndex	3,302
R_TSIP_GenerateEccP224PublicKeyIndex	3,298
R_TSIP_GenerateEccP256PublicKeyIndex	3,298
R_TSIP_GenerateEccP384PublicKeyIndex	3,404
R_TSIP_GenerateEccP192PrivateKeyIndex	2,960
R_TSIP_GenerateEccP224PrivateKeyIndex	2,956
R_TSIP_GenerateEccP256PrivateKeyIndex	2,960
R_TSIP_GenerateEccP384PrivateKeyIndex	2,882
R_TSIP_GenerateEccP192RandomKeyIndex *1	145,134
R_TSIP_GenerateEccP224RandomKeyIndex *1	153,367
R_TSIP_GenerateEccP256RandomKeyIndex *1	155,429
R_TSIP_GenerateEccP384RandomKeyIndex *1	1,059,818
R_TSIP_UpdateEccP192PublicKeyIndex	2,902
R_TSIP_UpdateEccP224PublicKeyIndex	2,900
R_TSIP_UpdateEccP256PublicKeyIndex	2,902
R_TSIP_UpdateEccP384PublicKeyIndex	3,012
R_TSIP_UpdateEccP192PrivateKeyIndex	2,586
R_TSIP_UpdateEccP224PrivateKeyIndex	2,586
R_TSIP_UpdateEccP256PrivateKeyIndex	2,586
R_TSIP_UpdateEccP384PrivateKeyIndex	2,478

Note 1. Average value at 10 runs.

Table 1.53 Performance of ECDSA Signature Generation/Verification

API	Performance (Unit: cycle)		
	Message size=1byte	Message size=128byte	Message size=256byte
R_TSIP_EcdsaP192SignatureGenerate	172,720	174,196	173,392
R_TSIP_EcdsaP224SignatureGenerate	176,504	176,024	176,492
R_TSIP_EcdsaP256SignatureGenerate	179,420	181,004	183,418
R_TSIP_EcdsaP384SignatureGenerate*1	1,188,086		
R_TSIP_EcdsaP192SignatureVerification	327,692	331,328	330,260
R_TSIP_EcdsaP224SignatureVerification	349,612	342,856	349,276
R_TSIP_EcdsaP256SignatureVerification	350,704	352,920	358,764
R_TSIP_EcdsaP384SignatureVerification*1	2,202,578		

Note 1. Not include SHA384 calculation.

Table 1.54 Performance of Key Exchange

API	Performance (Unit: cycle)
R_TSIP_EcdhP256Init	56
R_TSIP_EcdhP256ReadPublicKey	357,840
R_TSIP_EcdhP256MakePublicKey	332,290
R_TSIP_EcdhP256CalculateSharedSecretIndex	375,382
R_TSIP_EcdhP256KeyDerivation	3,750
R_TSIP_EcdheP512KeyAgreement	3,270,996
R_TSIP_Rsa2048DhKeyAgreement	52,726,726

Key exchange performance (without KeyAgreement) was measured with parameters fixed as follows: key exchange format = ECDHE and derived key type = AES-128.

1.12 Performance (RX671)

Information on the performance of the TSIP driver on the RX671 is shown below. Performance is measured using cycles of the core clock ICLK as the basic unit. The frequency of the TSIP operating clock PCLKB is set to ICLK:PCLKB = 2:1.

The optimization level is level 2.

Table 1.55 Performance of each APIs

API	**Performance (Unit: cycle)**
R_TSIP_Open	5,295,282
R_TSIP_Close	300
R_TSIP_GetVersion	24
R_TSIP_GenerateAes128KeyIndex	2,046
R_TSIP_GenerateAes256KeyIndex	2,162
R_TSIP_GenerateAes128RandomKeyIndex	1,162
R_TSIP_GenerateAes256RandomKeyIndex	1,622
R_TSIP_GenerateRandomNumber	520
R_TSIP_GenerateUpdateKeyRingKeyIndex	2,160
R_TSIP_UpdateAes128KeyIndex	1,792
R_TSIP_UpdateAes256KeyIndex	1,918

Table 1.56 Performance of Firmware Verify APIs

API	Performance (Unit: cycle)		
	8 KB processing	16 KB processing	24 KB processing
R_TSIP_VerifyFirmwareMAC	16,796	33,180	49,564

Table 1.57 Performance of AES

API	Performance (Unit: cycle)		
	16-byte processing	48-byte processing	80-byte processing
R_TSIP_Aes128EcbEncryptInit	1,220	1,200	1,200
R_TSIP_Aes128EcbEncryptUpdate	382	484	616
R_TSIP_Aes128EcbEncryptFinal	316	306	306
R_TSIP_Aes128EcbDecryptInit	1,216	1,216	1,216
R_TSIP_Aes128EcbDecryptUpdate	444	542	674
R_TSIP_Aes128EcbDecryptFinal	322	322	322
R_TSIP_Aes256EcbEncryptInit	1,340	1,326	1,328
R_TSIP_Aes256EcbEncryptUpdate	396	512	644
R_TSIP_Aes256EcbEncryptFinal	322	318	320
R_TSIP_Aes256EcbDecryptInit	1,338	1,336	1,338
R_TSIP_Aes256EcbDecryptUpdate	468	584	716
R_TSIP_Aes256EcbDecryptFinal	328	328	328
R_TSIP_Aes128CbcEncryptInit	1,266	1,260	1,260
R_TSIP_Aes128CbcEncryptUpdate	444	550	682
R_TSIP_Aes128CbcEncryptFinal	340	340	338
R_TSIP_Aes128CbcDecryptInit	1,284	1,282	1,284
R_TSIP_Aes128CbcDecryptUpdate	502	600	732
R_TSIP_Aes128CbcDecryptFinal	344	340	340
R_TSIP_Aes256CbcEncryptInit	1,394	1,392	1,394
R_TSIP_Aes256CbcEncryptUpdate	458	580	712
R_TSIP_Aes256CbcEncryptFinal	340	340	338
R_TSIP_Aes256CbcDecryptInit	1,408	1,410	1,408
R_TSIP_Aes256CbcDecryptUpdate	526	646	778
R_TSIP_Aes256CbcDecryptFinal	346	346	346

Table 1.58 Performance of GCM

API	Performance (Unit: cycle)		
	48-byte processing	64-byte processing	80-byte processing
R_TSIP_Aes128GcmEncryptInit	3,932	3,934	3,934
R_TSIP_Aes128GcmEncryptUpdate	1,540	1,624	1,688
R_TSIP_Aes128GcmEncryptFinal	826	822	822
R_TSIP_Aes128GcmDecryptInit	3,962	3,958	3,958
R_TSIP_Aes128GcmDecryptUpdate	1,528	1,590	1,654
R_TSIP_Aes128GcmDecryptFinal	1,432	1,428	1,428
R_TSIP_Aes256GcmEncryptInit	4,090	4,090	4,092
R_TSIP_Aes256GcmEncryptUpdate	1,560	1,648	1,712
R_TSIP_Aes256GcmEncryptFinal	840	828	828
R_TSIP_Aes256GcmDecryptInit	4,100	4,086	4,086
R_TSIP_Aes256GcmDecryptUpdate	1,560	1,620	1,692
R_TSIP_Aes256GcmDecryptFinal	1,446	1,430	1,430

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

Table 1.59 Performance of AES-CCM

API	Performance (Unit: cycle)		
	48-byte processing	64-byte processing	80-byte processing
R_TSIP_Aes128CcmEncryptInit	1,892	1,874	1,874
R_TSIP_Aes128CcmEncryptUpdate	872	952	1,024
R_TSIP_Aes128CcmEncryptFinal	752	744	744
R_TSIP_Aes128CcmDecryptInit	1,722	1,714	1,714
R_TSIP_Aes128CcmDecryptUpdate	794	858	938
R_TSIP_Aes128CcmDecryptFinal	1,458	1,450	1,450
R_TSIP_Aes256CcmEncryptInit	1,874	1,870	1,870
R_TSIP_Aes256CcmEncryptUpdate	928	1,020	1,108
R_TSIP_Aes256CcmEncryptFinal	762	752	752
R_TSIP_Aes256CcmDecryptInit	1,872	1,860	1,860
R_TSIP_Aes256CcmDecryptUpdate	836	914	1,010
R_TSIP_Aes256CcmDecryptFinal	1,478	1,474	1,472

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

Table 1.60 Performance of MAC (AES-CMAC)

API	Performance (Unit: cycle)		
	48-byte processing	64-byte processing	80-byte processing
R_TSIP_Aes128CmacGenerateInit	872	866	866
R_TSIP_Aes128CmacGenerateUpdate	484	514	556
R_TSIP_Aes128CmacGenerateFinal	610	602	602
R_TSIP_Aes128CmacVerifyInit	872	870	870
R_TSIP_Aes128CmacVerifyUpdate	488	516	558
R_TSIP_Aes128CmacVerifyFinal	1,220	1,218	1,218
R_TSIP_Aes256CmacGenerateInit	990	986	986
R_TSIP_Aes256CmacGenerateUpdate	502	532	582
R_TSIP_Aes256CmacGenerateFinal	636	626	626
R_TSIP_Aes256CmacVerifyInit	988	986	986
R_TSIP_Aes256CmacVerifyUpdate	492	536	586
R_TSIP_Aes256CmacVerifyFinal	1,244	1,244	1,244

Table 1.61 Performance of AES Key Wrap

API	Performance (Unit: cycle)	
	Wrap target key = AES-128	Wrap target key = AES-256
R_TSIP_Aes128KeyWrap	6,358	10,042
R_TSIP_Aes256KeyWrap	6,564	10,350
R_TSIP_Aes128KeyUnwrap	7,140	10,760
R_TSIP_Aes256KeyUnwrap	7,340	11,074

Table 1.62 Performance of Common API (TDES User Key Index Generation)

API	Performance (Unit: cycle)
R_TSIP_GenerateTdesKeyIndex	2,182
R_TSIP_GenerateTdesRandomKeyIndex	1,616
R_TSIP_UpdateTdesKeyIndex	1,918

Table 1.63 Performance of TDES

API	Performance (Unit: cycle)		
	16-byte processing	48-byte processing	80-byte processing
R_TSIP_TdesEcbEncryptInit	812	804	804
R_TSIP_TdesEcbEncryptUpdate	418	602	794
R_TSIP_TdesEcbEncryptFinal	320	306	306
R_TSIP_TdesEcbDecryptInit	810	810	810
R_TSIP_TdesEcbDecryptUpdate	430	624	816
R_TSIP_TdesEcbDecryptFinal	322	320	320
R_TSIP_TdesCbcEncryptInit	858	848	846
R_TSIP_TdesCbcEncryptUpdate	478	670	862
R_TSIP_TdesCbcEncryptFinal	336	334	334
R_TSIP_TdesCbcDecryptInit	860	858	858
R_TSIP_TdesCbcDecryptUpdate	504	700	892
R_TSIP_TdesCbcDecryptFinal	348	346	346

Table 1.64 Performance of Common API (RSA User Key Index Generation)

API	Performance (Unit: cycle)
R_TSIP_GenerateRsa1024PublicKeyIndex	36,602
R_TSIP_GenerateRsa1024PrivateKeyIndex	37,602
R_TSIP_GenerateRsa2048PublicKeyIndex	136,336
R_TSIP_GenerateRsa2048PrivateKeyIndex	138,342
R_TSIP_GenerateRsa1024RandomKeyIndex *1	50,778,587
R_TSIP_GenerateRsa2048RandomKeyIndex *1	457,710,545
R_TSIP_UpdateRsa1024PublicKeyIndex	36,350
R_TSIP_UpdateRsa1024PrivateKeyIndex	37,344
R_TSIP_UpdateRsa2048PublicKeyIndex	136,088
R_TSIP_UpdateRsa2048PrivateKeyIndex	138,056

Note 1. Average value at 10 runs.

Table 1.65 Performance of RSASSA-PKCS-v1_5 Signature Generation/Verification (HASH = SHA1)

API	Performance (Unit: cycle)		
	Message size=1byte	Message size=128byte	Message size=256byte
R_TSIP_RsassaPkcs1024SignatureGenerate	1,232,788	1,233,888	1,234,304
R_TSIP_RsassaPkcs1024SignatureVerification	15,970	17,094	17,512
R_TSIP_RsassaPkcs2048SignatureGenerate	26,094,768	26,095,884	26,096,300
R_TSIP_RsassaPkcs2048SignatureVerification	133,516	134,640	135,056

Table 1.66 Performance of RSASSA-PKCS-v1_5 Signature Generation/Verification (HASH = SHA256)

API	Performance (Unit: cycle)		
	Message size=1byte	Message size=128byte	Message size=256byte
R_TSIP_RsassaPkcs1024SignatureGenerate	1,232,850	1,233,996	1,234,348
R_TSIP_RsassaPkcs1024SignatureVerification	16,052	17,198	17,548
R_TSIP_RsassaPkcs2048SignatureGenerate	26,094,846	26,095,996	26,096,348
R_TSIP_RsassaPkcs2048SignatureVerification	133,604	134,754	135,106

Table 1.67 Performance of RSASSA-PKCS-v1_5 Signature Generation/Verification (HASH = MD5)

API	Performance (Unit: cycle)		
	Message size=1byte	Message size=128byte	Message size=256byte
R_TSIP_RsassaPkcs1024SignatureGenerate	1,232,732	1,233,754	1,234,106
R_TSIP_RsassaPkcs1024SignatureVerification	15,934	16,968	17,318
R_TSIP_RsassaPkcs2048SignatureGenerate	26,094,718	26,095,752	26,096,104
R_TSIP_RsassaPkcs2048SignatureVerification	133,486	134,516	134,868

Table 1.68 Performance of RSAES-PKCS1-v1_5 Encryption/Decryption with 1,024-Bit Key Size

API	Performance (Unit: cycle)	
	Message size=1byte	Message size=117byte
R_TSIP_RsaesPkcs1024Encrypt	19,646	15,484
R_TSIP_RsaesPkcs1024Decrypt	1,232,078	1,232,080

Table 1.69 Performance of RSAES-PKCS1-v1_5 Encryption/Decryption with 2,048-Bit Key Size

API	Performance (Unit: cycle)	
	Message size=1byte	Message size=245byte
R_TSIP_RsaesPkcs2048Encrypt	142,282	132,866
R_TSIP_RsaesPkcs2048Decrypt	26,094,270	26,094,272

Table 1.70 Performance of HASH (SHA1)

API	Performance (Unit: cycle)		
	128-byte processing	192-byte processing	256-byte processing
R_TSIP_Sha1Init	110	110	108
R_TSIP_Sha1Update	1,208	1,416	1,624
R_TSIP_Sha1Final	662	658	658

Table 1.71 Performance of HASH (SHA256)

API	Performance (Unit: cycle)		
	128-byte processing	192-byte processing	256-byte processing
R_TSIP_Sha256Init	158	154	154
R_TSIP_Sha256Update	1,232	1,408	1,584
R_TSIP_Sha256Final	670	672	672

Table 1.72 Performance of HASH (MD5)

API	Performance (Unit: cycle)		
	128-byte processing	192-byte processing	256-byte processing
R_TSIP_Md5Init	102	96	96
R_TSIP_Md5Update	1,116	1,284	1,460
R_TSIP_Md5Final	626	626	626

Table 1.73 Performance of Common API (HMAC User Key Index Generation)

API	Performance (Unit: cycle)
R_TSIP_GenerateSha1HmacKeyIndex	2,248
R_TSIP_GenerateSha256HmacKeyIndex	2,232
R_TSIP_UpdateSha1HmacKeyIndex	2,010
R_TSIP_UpdateSha256HmacKeyIndex	1,996

Table 1.74 Performance of HMAC (SHA1)

API	Performance (Unit: cycle)		
	128-byte processing	192-byte processing	256-byte processing
R_TSIP_Sha1HmacGenerateInit	1,032	1,030	1,030
R_TSIP_Sha1HmacGenerateUpdate	804	1,006	1,214
R_TSIP_Sha1HmacGenerateFinal	1,590	1,578	1,578
R_TSIP_Sha1HmacVerifyInit	1,026	1,026	1,028
R_TSIP_Sha1HmacVerifyUpdate	798	1,004	1,212
R_TSIP_Sha1HmacVerifyFinal	2,708	2,706	2,706

Table 1.75 Performance of HMAC (SHA256)

API	Performance (Unit: cycle)		
	128-byte processing	192-byte processing	256-byte processing
R_TSIP_Sha256HmacGenerateInit	1,232	1,226	1,226
R_TSIP_Sha256HmacGenerateUpdate	736	904	1,080
R_TSIP_Sha256HmacGenerateFinal	1,552	1,542	1,542
R_TSIP_Sha256HmacVerifyInit	1,220	1,228	1,226
R_TSIP_Sha256HmacVerifyUpdate	722	896	1,072
R_TSIP_Sha256HmacVerifyFinal	2,662	2,658	2,658

Table 1.76 Performance of Common API (ECC User Key Index Generation)

API	Performance (Unit: cycle)
R_TSIP_GenerateEccP192PublicKeyIndex	2,542
R_TSIP_GenerateEccP224PublicKeyIndex	2,528
R_TSIP_GenerateEccP256PublicKeyIndex	2,530
R_TSIP_GenerateEccP384PublicKeyIndex	2,688
R_TSIP_GenerateEccP192PrivateKeyIndex	2,252
R_TSIP_GenerateEccP224PrivateKeyIndex	2,240
R_TSIP_GenerateEccP256PrivateKeyIndex	2,242
R_TSIP_GenerateEccP384PrivateKeyIndex	2,292
R_TSIP_GenerateEccP192RandomKeyIndex *1	133,054
R_TSIP_GenerateEccP224RandomKeyIndex *1	141,140
R_TSIP_GenerateEccP256RandomKeyIndex *1	143,945
R_TSIP_GenerateEccP384RandomKeyIndex *1	1,012,834
R_TSIP_UpdateEccP192PublicKeyIndex	2,292
R_TSIP_UpdateEccP224PublicKeyIndex	2,278
R_TSIP_UpdateEccP256PublicKeyIndex	2,280
R_TSIP_UpdateEccP384PublicKeyIndex	2,448
R_TSIP_UpdateEccP192PrivateKeyIndex	2,000
R_TSIP_UpdateEccP224PrivateKeyIndex	1,988
R_TSIP_UpdateEccP256PrivateKeyIndex	1,988
R_TSIP_UpdateEccP384PrivateKeyIndex	2,030

Note 1. Average value at 10 runs.

Table 1.77 Performance of ECDSA Signature Generation/Verification

API	Performance (Unit: cycle)		
	Message size=1byte	Message size=128byte	Message size=256byte
R_TSIP_EcdsaP192SignatureGenerate	160,202	161,920	163,504
R_TSIP_EcdsaP224SignatureGenerate	161,104	161,076	165,228
R_TSIP_EcdsaP256SignatureGenerate	169,548	165,664	167,296
R_TSIP_EcdsaP384SignatureGenerate*1	1,103,548		
R_TSIP_EcdsaP192SignatureVerification	307,014	309,412	303,992
R_TSIP_EcdsaP224SignatureVerification	320,384	324,744	328,258
R_TSIP_EcdsaP256SignatureVerification	332,374	329,762	330,764
R_TSIP_EcdsaP384SignatureVerification*1	2,105,824		

Note 1. Not include SHA384 calculation.

Table 1.78 Performance of Key Exchange

API	Performance (Unit: cycle)
R_TSIP_EcdhP256Init	44
R_TSIP_EcdhP256ReadPublicKey	330,926
R_TSIP_EcdhP256MakePublicKey	306,630
R_TSIP_EcdhP256CalculateSharedSecretIndex	350,378
R_TSIP_EcdhP256KeyDerivation	3,010
R_TSIP_EcdheP512KeyAgreement	3,179,800
R_TSIP_Rsa2048DhKeyAgreement	52,461,482

Key exchange performance (without KeyAgreement) was measured with parameters fixed as follows: key exchange format = ECDHE and derived key type = AES-GCM-128.

1.13 Performance (RX72M)

Information on the performance of the TSIP driver on the RX72M is shown below. Performance is measured using cycles of the core clock ICLK as the basic unit. The frequency of the TSIP operating clock PCLKB is set to ICLK:PCLKB = 2:1.

The Optimization level is level 2.

Table 1.79 Performance of each APIs

API	**Performance (Unit: cycle)**
R_TSIP_Open	6,268,188
R_TSIP_Close	294
R_TSIP_GetVersion	22
R_TSIP_GenerateAes128KeyIndex	2,132
R_TSIP_GenerateAes256KeyIndex	2,246
R_TSIP_GenerateAes128RandomKeyIndex	1,234
R_TSIP_GenerateAes256RandomKeyIndex	1,722
R_TSIP_GenerateRandomNumber	552
R_TSIP_GenerateUpdateKeyRingKeyIndex	2,260
R_TSIP_UpdateAes128KeyIndex	1,872
R_TSIP_UpdateAes256KeyIndex	2,006

Table 1.80 Performance of Firmware Verify APIs

API	Performance (Unit: cycle)		
	8 KB processing	16 KB processing	24 KB processing
R_TSIP_VerifyFirmwareMAC	18,840	37,270	55,702

Table 1.81 Performance of AES

API	Performance (Unit: cycle)		
	16-byte processing	48-byte processing	80-byte processing
R_TSIP_Aes128EcbEncryptInit	1,264	1,264	1,264
R_TSIP_Aes128EcbEncryptUpdate	388	504	640
R_TSIP_Aes128EcbEncryptFinal	330	330	330
R_TSIP_Aes128EcbDecryptInit	1,274	1,276	1,276
R_TSIP_Aes128EcbDecryptUpdate	448	560	696
R_TSIP_Aes128EcbDecryptFinal	346	346	346
R_TSIP_Aes256EcbEncryptInit	1,382	1,380	1,380
R_TSIP_Aes256EcbEncryptUpdate	394	516	652
R_TSIP_Aes256EcbEncryptFinal	328	326	326
R_TSIP_Aes256EcbDecryptInit	1,392	1,394	1,394
R_TSIP_Aes256EcbDecryptUpdate	472	592	728
R_TSIP_Aes256EcbDecryptFinal	338	338	338
R_TSIP_Aes128CbcEncryptInit	1,326	1,324	1,322
R_TSIP_Aes128CbcEncryptUpdate	448	568	704
R_TSIP_Aes128CbcEncryptFinal	358	356	356
R_TSIP_Aes128CbcDecryptInit	1,338	1,336	1,338
R_TSIP_Aes128CbcDecryptUpdate	512	622	758
R_TSIP_Aes128CbcDecryptFinal	366	366	366
R_TSIP_Aes256CbcEncryptInit	1,444	1,444	1,446
R_TSIP_Aes256CbcEncryptUpdate	460	582	718
R_TSIP_Aes256CbcEncryptFinal	346	346	346
R_TSIP_Aes256CbcDecryptInit	1,458	1,460	1,462
R_TSIP_Aes256CbcDecryptUpdate	536	658	792
R_TSIP_Aes256CbcDecryptFinal	360	360	360

Table 1.82 Performance of AES-GCM

API	Performance (Unit: cycle)		
	48-byte processing	64-byte processing	80-byte processing
R_TSIP_Aes128GcmEncryptInit	4,120	4,120	4,120
R_TSIP_Aes128GcmEncryptUpdate	1,570	1,656	1,724
R_TSIP_Aes128GcmEncryptFinal	852	852	852
R_TSIP_Aes128GcmDecryptInit	4,140	4,140	4,140
R_TSIP_Aes128GcmDecryptUpdate	1,570	1,636	1,704
R_TSIP_Aes128GcmDecryptFinal	1,472	1,468	1,468
R_TSIP_Aes256GcmEncryptInit	4,264	4,268	4,268
R_TSIP_Aes256GcmEncryptUpdate	1,602	1,700	1,770
R_TSIP_Aes256GcmEncryptFinal	864	862	862
R_TSIP_Aes256GcmDecryptInit	4,284	4,278	4,278
R_TSIP_Aes256GcmDecryptUpdate	1,606	1,670	1,736
R_TSIP_Aes256GcmDecryptFinal	1,482	1,480	1,480

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

Table 1.83 Performance of AES-CCM

API	Performance (Unit: cycle)		
	48-byte processing	64-byte processing	80-byte processing
R_TSIP_Aes128CcmEncryptInit	1,934	1,936	1,936
R_TSIP_Aes128CcmEncryptUpdate	900	968	1,044
R_TSIP_Aes128CcmEncryptFinal	774	772	772
R_TSIP_Aes128CcmDecryptInit	1,760	1,758	1,758
R_TSIP_Aes128CcmDecryptUpdate	814	892	970
R_TSIP_Aes128CcmDecryptFinal	1,510	1,508	1,508
R_TSIP_Aes256CcmEncryptInit	1,930	1,928	1,930
R_TSIP_Aes256CcmEncryptUpdate	952	1,040	1,138
R_TSIP_Aes256CcmEncryptFinal	788	786	786
R_TSIP_Aes256CcmDecryptInit	1,922	1,920	1,920
R_TSIP_Aes256CcmDecryptUpdate	860	954	1,042
R_TSIP_Aes256CcmDecryptFinal	1,514	1,512	1,512

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

Table 1.84 Performance of AES-CMAC

API	Performance (Unit: cycle)		
	48-byte processing	64-byte processing	80-byte processing
R_TSIP_Aes128CmacGenerateInit	904	902	902
R_TSIP_Aes128CmacGenerateUpdate	486	522	558
R_TSIP_Aes128CmacGenerateFinal	634	624	624
R_TSIP_Aes128CmacVerifyInit	906	906	904
R_TSIP_Aes128CmacVerifyUpdate	488	522	560
R_TSIP_Aes128CmacVerifyFinal	1,254	1,254	1,254
R_TSIP_Aes256CmacGenerateInit	1,014	1,012	1,014
R_TSIP_Aes256CmacGenerateUpdate	512	558	596
R_TSIP_Aes256CmacGenerateFinal	654	650	650
R_TSIP_Aes256CmacVerifyInit	1,014	1,016	1,016
R_TSIP_Aes256CmacVerifyUpdate	514	562	600
R_TSIP_Aes256CmacVerifyFinal	1,280	1,284	1,284

Table 1.85 Performance of AES Key Wrap

API	Performance (Unit: cycle)	
	Wrap target key = AES-128	Wrap target key = AES-256
R_TSIP_Aes128KeyWrap	6,494	10,294
R_TSIP_Aes256KeyWrap	6,722	10,644
R_TSIP_Aes128KeyUnwrap	7,312	11,004
R_TSIP_Aes256KeyUnwrap	7,548	11,356

Table 1.86 Performance of Common API (TDES User Key Index Generation)

API	Performance (Unit: cycle)
R_TSIP_GenerateTdesKeyIndex	2,258
R_TSIP_GenerateTdesRandomKeyIndex	1,726
R_TSIP_UpdateTdesKeyIndex	2,008

Table 1.87 Performance of TDES

API	Performance (Unit: cycle)		
	16-byte processing	48-byte processing	80-byte processing
R_TSIP_TdesEcbEncryptInit	830	824	826
R_TSIP_TdesEcbEncryptUpdate	430	628	828
R_TSIP_TdesEcbEncryptFinal	332	328	328
R_TSIP_TdesEcbDecryptInit	836	834	832
R_TSIP_TdesEcbDecryptUpdate	450	650	850
R_TSIP_TdesEcbDecryptFinal	340	342	342
R_TSIP_TdesCbcEncryptInit	884	882	882
R_TSIP_TdesCbcEncryptUpdate	490	690	890
R_TSIP_TdesCbcEncryptFinal	350	350	350
R_TSIP_TdesCbcDecryptInit	892	890	892
R_TSIP_TdesCbcDecryptUpdate	514	716	916
R_TSIP_TdesCbcDecryptFinal	370	370	370

Table 1.88 Performance of Common API (RSA User Key Index Generation)

API	Performance (Unit: cycle)
R_TSIP_GenerateRsa1024PublicKeyIndex	36,744
R_TSIP_GenerateRsa1024PrivateKeyIndex	37,730
R_TSIP_GenerateRsa2048PublicKeyIndex	136,494
R_TSIP_GenerateRsa2048PrivateKeyIndex	138,478
R_TSIP_GenerateRsa1024RandomKeyIndex *1	35,231,688
R_TSIP_GenerateRsa2048RandomKeyIndex *1	439,079,858
R_TSIP_UpdateRsa1024PublicKeyIndex	36,490
R_TSIP_UpdateRsa1024PrivateKeyIndex	37,454
R_TSIP_UpdateRsa2048PublicKeyIndex	136,244
R_TSIP_UpdateRsa2048PrivateKeyIndex	138,192

Note 1. Average value at 10 runs.

Table 1.89 Performance of RSASSA-PKCS-v1_5 Signature Generation/Verification (HASH = SHA1)

API	Performance (Unit: cycle)		
	Message size=1byte	Message size=128byte	Message size=256byte
R_TSIP_RsassaPkcs1024SignatureGenerate	1,232,976	1,234,164	1,234,580
R_TSIP_RsassaPkcs1024SignatureVerification	16,082	17,274	17,680
R_TSIP_RsassaPkcs2048SignatureGenerate	26,095,160	26,096,350	26,096,760
R_TSIP_RsassaPkcs2048SignatureVerification	133,716	134,916	135,324

Table 1.90 Performance of RSASSA-PKCS-v1_5 Signature Generation/Verification (HASH = SHA256)

API	Performance (Unit: cycle)		
	Message size=1byte	Message size=128byte	Message size=256byte
R_TSIP_RsassaPkcs1024SignatureGenerate	1,233,054	1,234,254	1,234,608
R_TSIP_RsassaPkcs1024SignatureVerification	16,166	17,362	17,708
R_TSIP_RsassaPkcs2048SignatureGenerate	26,095,244	26,096,444	26,096,794
R_TSIP_RsassaPkcs2048SignatureVerification	133,798	134,996	135,344

Table 1.91 Performance of RSASSA-PKCS-v1_5 Signature Generation/Verification (HASH = MD5)

API	Performance (Unit: cycle)		
	Message size=1byte	Message size=128byte	Message size=256byte
R_TSIP_RsassaPkcs1024SignatureGenerate	1,232,948	1,234,024	1,234,372
R_TSIP_RsassaPkcs1024SignatureVerification	16,054	17,134	17,482
R_TSIP_RsassaPkcs2048SignatureGenerate	26,095,128	26,096,212	26,096,558
R_TSIP_RsassaPkcs2048SignatureVerification	133,688	134,768	135,116

Table 1.92 Performance of RSAES-PKCS1-v1_5 Encryption/Decryption with 1,024-Bit Key Size

API	Performance (Unit: cycle)	
	Message size=1byte	Message size=117byte
R_TSIP_RsaesPkcs1024Encrypt	20,114	15,636
R_TSIP_RsaesPkcs1024Decrypt	1,232,298	1,232,290

Table 1.93 Performance of RSAES-PKCS1-v1_5 Encryption/Decryption with 2,048-Bit Key Size

API	Performance (Unit: cycle)	
	Message size=1byte	Message size=245byte
R_TSIP_RsaesPkcs2048Encrypt	142,680	133,118
R_TSIP_RsaesPkcs2048Decrypt	26,094,668	26,094,670

Table 1.94 Performance of HASH (SHA1)

API	Performance (Unit: cycle)		
	128-byte processing	192-byte processing	256-byte processing
R_TSIP_Sha1Init	106	106	106
R_TSIP_Sha1Update	1,246	1,450	1,654
R_TSIP_Sha1Final	664	664	662

Table 1.95 Performance of HASH (SHA256)

API	Performance (Unit: cycle)		
	128-byte processing	192-byte processing	256-byte processing
R_TSIP_Sha256Init	152	152	154
R_TSIP_Sha256Update	1,272	1,444	1,618
R_TSIP_Sha256Final	682	682	682

Table 1.96 Performance of HASH (MD5)

API	Performance (Unit: cycle)		
	128-byte processing	192-byte processing	256-byte processing
R_TSIP_Md5Init	100	98	98
R_TSIP_Md5Update	1,156	1,328	1,500
R_TSIP_Md5Final	638	638	638

Table 1.97 Performance of Common API (HMAC User Key Index Generation)

API	Performance (Unit: cycle)
R_TSIP_GenerateSha1HmacKeyIndex	2,346
R_TSIP_GenerateSha256HmacKeyIndex	2,344
R_TSIP_UpdateSha1HmacKeyIndex	2,104
R_TSIP_UpdateSha256HmacKeyIndex	2,094

Table 1.98 Performance of HMAC (SHA1)

API	Performance (Unit: cycle)		
	128-byte processing	192-byte processing	256-byte processing
R_TSIP_Sha1HmacGenerateInit	1,082	1,080	1,080
R_TSIP_Sha1HmacGenerateUpdate	806	1,008	1,212
R_TSIP_Sha1HmacGenerateFinal	1,608	1,606	1,606
R_TSIP_Sha1HmacVerifyInit	1,080	1,080	1,080
R_TSIP_Sha1HmacVerifyUpdate	804	1,008	1,210
R_TSIP_Sha1HmacVerifyFinal	2,742	2,742	2,742

Table 1.99 Performance of HMAC (SHA256)

API	Performance (Unit: cycle)		
	128-byte processing	192-byte processing	256-byte processing
R_TSIP_Sha256HmacGenerateInit	1,280	1,278	1,280
R_TSIP_Sha256HmacGenerateUpdate	734	906	1,082
R_TSIP_Sha256HmacGenerateFinal	1,576	1,574	1,574
R_TSIP_Sha256HmacVerifyInit	1,272	1,274	1,274
R_TSIP_Sha256HmacVerifyUpdate	730	904	1,078
R_TSIP_Sha256HmacVerifyFinal	2,728	2,728	2,726

Table 1.100 Performance of Common API (ECC User Key Index Generation)

API	Performance (Unit: cycle)
R_TSIP_GenerateEccP192PublicKeyIndex	2,646
R_TSIP_GenerateEccP224PublicKeyIndex	2,638
R_TSIP_GenerateEccP256PublicKeyIndex	2,642
R_TSIP_GenerateEccP384PublicKeyIndex	2,812
R_TSIP_GenerateEccP192PrivateKeyIndex	2,340
R_TSIP_GenerateEccP224PrivateKeyIndex	2,336
R_TSIP_GenerateEccP256PrivateKeyIndex	2,338
R_TSIP_GenerateEccP384PrivateKeyIndex	2,374
R_TSIP_GenerateEccP192RandomKeyIndex *1	132,839
R_TSIP_GenerateEccP224RandomKeyIndex *1	141,935
R_TSIP_GenerateEccP256RandomKeyIndex *1	143,598
R_TSIP_GenerateEccP384RandomKeyIndex *1	1,007,046
R_TSIP_UpdateEccP192PublicKeyIndex	2,404
R_TSIP_UpdateEccP224PublicKeyIndex	2,400
R_TSIP_UpdateEccP256PublicKeyIndex	2,398
R_TSIP_UpdateEccP384PublicKeyIndex	2,560
R_TSIP_UpdateEccP192PrivateKeyIndex	2,094
R_TSIP_UpdateEccP224PrivateKeyIndex	2,090
R_TSIP_UpdateEccP256PrivateKeyIndex	2,092
R_TSIP_UpdateEccP384PrivateKeyIndex	2,130

Note 1. Average value at 10 runs.

Table 1.101 Performance of ECDSA Signature Generation/Verification

API	Performance (Unit: cycle)		
	Message size=1byte	Message size=128byte	Message size=256byte
R_TSIP_EcdsaP192SignatureGenerate	163,162	161,944	164,802
R_TSIP_EcdsaP224SignatureGenerate	164,732	164,794	163,864
R_TSIP_EcdsaP256SignatureGenerate	168,142	163,026	166,634
R_TSIP_EcdsaP384SignatureGenerate*1	1,110,656		
R_TSIP_EcdsaP192SignatureVerification	305,682	306,340	304,070
R_TSIP_EcdsaP224SignatureVerification	324,812	326,062	328,238
R_TSIP_EcdsaP256SignatureVerification	327,312	331,032	333,902
R_TSIP_EcdsaP384SignatureVerification*1	2,114,564		

Note 1. Not include SHA384 calculation.

Table 1.102 Performance of Key Exchange

API	Performance (Unit: cycle)
R_TSIP_EcdhP256Init	40
R_TSIP_EcdhP256ReadPublicKey	332,266
R_TSIP_EcdhP256MakePublicKey	309,756
R_TSIP_EcdhP256CalculateSharedSecretIndex	350,746
R_TSIP_EcdhP256KeyDerivation	3,116
R_TSIP_EcdheP512KeyAgreement	3,239,938
R_TSIP_Rsa2048DhKeyAgreement	52,462,438

Key exchange performance (without KeyAgreement) was measured with parameters fixed as follows: key exchange format = ECDHE and derived key type = AES-128.

1.14 Performance (RX72N)

Information on the performance of the TSIP driver on the RX72N is shown below. Performance is measured using cycles of the core clock ICLK as the basic unit. The frequency of the TSIP operating clock PCLKB is set to ICLK:PCLKB = 2:1.

The Optimization level is level 2.

Table 1.103 Performance of each APIs

API	**Performance (Unit: cycle)**
R_TSIP_Open	6,214,042
R_TSIP_Close	288
R_TSIP_GetVersion	20
R_TSIP_GenerateAes128KeyIndex	2,126
R_TSIP_GenerateAes256KeyIndex	2,258
R_TSIP_GenerateAes128RandomKeyIndex	1,242
R_TSIP_GenerateAes256RandomKeyIndex	1,726
R_TSIP_GenerateRandomNumber	550
R_TSIP_GenerateUpdateKeyRingKeyIndex	2,262
R_TSIP_UpdateAes128KeyIndex	1,870
R_TSIP_UpdateAes256KeyIndex	2,006

Table 1.104 Performance of Firmware Verify APIs

API	Performance (Unit: cycle)		
	8 KB processing	16 KB processing	24 KB processing
R_TSIP_VerifyFirmwareMAC	18,852	37,282	55,714

Table 1.105 Performance of AES

API	Performance (Unit: cycle)		
	16-byte processing	48-byte processing	80-byte processing
R_TSIP_Aes128EcbEncryptInit	1,268	1,266	1,266
R_TSIP_Aes128EcbEncryptUpdate	390	506	642
R_TSIP_Aes128EcbEncryptFinal	330	330	330
R_TSIP_Aes128EcbDecryptInit	1,280	1,282	1,282
R_TSIP_Aes128EcbDecryptUpdate	450	560	696
R_TSIP_Aes128EcbDecryptFinal	346	344	344
R_TSIP_Aes256EcbEncryptInit	1,378	1,374	1,374
R_TSIP_Aes256EcbEncryptUpdate	402	526	662
R_TSIP_Aes256EcbEncryptFinal	328	326	326
R_TSIP_Aes256EcbDecryptInit	1,388	1,388	1,388
R_TSIP_Aes256EcbDecryptUpdate	478	602	738
R_TSIP_Aes256EcbDecryptFinal	340	342	340
R_TSIP_Aes128CbcEncryptInit	1,336	1,334	1,334
R_TSIP_Aes128CbcEncryptUpdate	452	572	708
R_TSIP_Aes128CbcEncryptFinal	354	354	354
R_TSIP_Aes128CbcDecryptInit	1,350	1,350	1,350
R_TSIP_Aes128CbcDecryptUpdate	516	626	762
R_TSIP_Aes128CbcDecryptFinal	364	364	366
R_TSIP_Aes256CbcEncryptInit	1,440	1,442	1,442
R_TSIP_Aes256CbcEncryptUpdate	464	588	724
R_TSIP_Aes256CbcEncryptFinal	348	348	348
R_TSIP_Aes256CbcDecryptInit	1,454	1,456	1,456
R_TSIP_Aes256CbcDecryptUpdate	542	666	802
R_TSIP_Aes256CbcDecryptFinal	364	364	364

Table 1.106 Performance of AES-GCM

API	Performance (Unit: cycle)		
	48-byte processing	64-byte processing	80-byte processing
R_TSIP_Aes128GcmEncryptInit	4,122	4,122	4,122
R_TSIP_Aes128GcmEncryptUpdate	1,568	1,654	1,722
R_TSIP_Aes128GcmEncryptFinal	856	854	852
R_TSIP_Aes128GcmDecryptInit	4,142	4,144	4,144
R_TSIP_Aes128GcmDecryptUpdate	1,570	1,636	1,704
R_TSIP_Aes128GcmDecryptFinal	1,474	1,470	1,472
R_TSIP_Aes256GcmEncryptInit	4,266	4,274	4,274
R_TSIP_Aes256GcmEncryptUpdate	1,596	1,694	1,762
R_TSIP_Aes256GcmEncryptFinal	864	860	862
R_TSIP_Aes256GcmDecryptInit	4,282	4,276	4,276
R_TSIP_Aes256GcmDecryptUpdate	1,602	1,670	1,738
R_TSIP_Aes256GcmDecryptFinal	1,478	1,476	1,476

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

Table 1.107 Performance of AES-CCM

API	Performance (Unit: cycle)		
	48-byte processing	64-byte processing	80-byte processing
R_TSIP_Aes128CcmEncryptInit	1,936	1,938	1,938
R_TSIP_Aes128CcmEncryptUpdate	902	970	1,048
R_TSIP_Aes128CcmEncryptFinal	778	774	774
R_TSIP_Aes128CcmDecryptInit	1,758	1,758	1,758
R_TSIP_Aes128CcmDecryptUpdate	814	890	968
R_TSIP_Aes128CcmDecryptFinal	1,514	1,510	1,510
R_TSIP_Aes256CcmEncryptInit	1,920	1,920	1,920
R_TSIP_Aes256CcmEncryptUpdate	950	1,040	1,136
R_TSIP_Aes256CcmEncryptFinal	792	790	790
R_TSIP_Aes256CcmDecryptInit	1,924	1,922	1,922
R_TSIP_Aes256CcmDecryptUpdate	854	952	1,040
R_TSIP_Aes256CcmDecryptFinal	1,510	1,510	1,510

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

Table 1.108 Performance of AES-CMAC

API	Performance (Unit: cycle)		
	48-byte processing	64-byte processing	80-byte processing
R_TSIP_Aes128CmacGenerateInit	906	904	904
R_TSIP_Aes128CmacGenerateUpdate	480	514	550
R_TSIP_Aes128CmacGenerateFinal	636	624	626
R_TSIP_Aes128CmacVerifyInit	906	904	906
R_TSIP_Aes128CmacVerifyUpdate	482	518	554
R_TSIP_Aes128CmacVerifyFinal	1,254	1,254	1,254
R_TSIP_Aes256CmacGenerateInit	1,016	1,014	1,014
R_TSIP_Aes256CmacGenerateUpdate	510	556	602
R_TSIP_Aes256CmacGenerateFinal	652	648	648
R_TSIP_Aes256CmacVerifyInit	1,016	1,014	1,016
R_TSIP_Aes256CmacVerifyUpdate	510	558	604
R_TSIP_Aes256CmacVerifyFinal	1,282	1,280	1,280

Table 1.109 Performance of AES Key Wrap

API	Performance (Unit: cycle)	
	Wrap target key = AES-128	Wrap target key = AES-256
R_TSIP_Aes128KeyWrap	6,498	10,290
R_TSIP_Aes256KeyWrap	6,726	10,644
R_TSIP_Aes128KeyUnwrap	7,324	11,014
R_TSIP_Aes256KeyUnwrap	7,560	11,374

Table 1.110 Performance of Common API (TDES User Key Index Generation)

API	Performance (Unit: cycle)
R_TSIP_GenerateTdesKeyIndex	2,260
R_TSIP_GenerateTdesRandomKeyIndex	1,726
R_TSIP_UpdateTdesKeyIndex	2,012

Table 1.111 Performance of TDES

API	Performance (Unit: cycle)		
	16-byte processing	48-byte processing	80-byte processing
R_TSIP_TdesEcbEncryptInit	828	826	826
R_TSIP_TdesEcbEncryptUpdate	432	630	830
R_TSIP_TdesEcbEncryptFinal	328	326	326
R_TSIP_TdesEcbDecryptInit	836	832	832
R_TSIP_TdesEcbDecryptUpdate	454	656	856
R_TSIP_TdesEcbDecryptFinal	336	334	334
R_TSIP_TdesCbcEncryptInit	884	882	880
R_TSIP_TdesCbcEncryptUpdate	496	696	896
R_TSIP_TdesCbcEncryptFinal	352	350	350
R_TSIP_TdesCbcDecryptInit	892	892	892
R_TSIP_TdesCbcDecryptUpdate	522	724	922
R_TSIP_TdesCbcDecryptFinal	364	364	362

Table 1.112 Performance of Common API (RSA User Key Index Generation)

API	Performance (Unit: cycle)
R_TSIP_GenerateRsa1024PublicKeyIndex	36,744
R_TSIP_GenerateRsa1024PrivateKeyIndex	37,728
R_TSIP_GenerateRsa2048PublicKeyIndex	136,506
R_TSIP_GenerateRsa2048PrivateKeyIndex	138,468
R_TSIP_GenerateRsa1024RandomKeyIndex *1	55,982,401
R_TSIP_GenerateRsa2048RandomKeyIndex *1	295,173,049
R_TSIP_UpdateRsa1024PublicKeyIndex	36,492
R_TSIP_UpdateRsa1024PrivateKeyIndex	37,452
R_TSIP_UpdateRsa2048PublicKeyIndex	136,246
R_TSIP_UpdateRsa2048PrivateKeyIndex	138,200

Note 1. Average value at 10 runs.

Table 1.113 Performance of RSASSA-PKCS1-v1_5 Signature Generation/Verification (HASH = SHA1)

API	Performance (Unit: cycle)		
	Message size=1byte	Message size=128byte	Message size=256byte
R_TSIP_RsassaPkcs1024SignatureGenerate	1,232,990	1,234,178	1,234,588
R_TSIP_RsassaPkcs1024SignatureVerification	16,088	17,284	17,694
R_TSIP_RsassaPkcs2048SignatureGenerate	26,095,160	26,096,358	26,096,768
R_TSIP_RsassaPkcs2048SignatureVerification	133,720	134,914	135,328

Table 1.114 Performance of RSASSA-PKCS1-v1_5 Signature Generation/Verification (HASH = SHA256)

API	Performance (Unit: cycle)		
	Message size=1byte	Message size=128byte	Message size=256byte
R_TSIP_RsassaPkcs1024SignatureGenerate	1,233,054	1,234,256	1,234,608
R_TSIP_RsassaPkcs1024SignatureVerification	16,164	17,362	17,710
R_TSIP_RsassaPkcs2048SignatureGenerate	26,095,238	26,096,440	26,096,788
R_TSIP_RsassaPkcs2048SignatureVerification	133,798	134,996	135,344

Table 1.115 Performance of RSASSA-PKCS1-v1_5 Signature Generation/Verification (HASH = MD5)

API	Performance (Unit: cycle)		
	Message size=1byte	Message size=128byte	Message size=256byte
R_TSIP_RsassaPkcs1024SignatureGenerate	1,232,954	1,234,024	1,234,372
R_TSIP_RsassaPkcs1024SignatureVerification	16,058	17,136	17,484
R_TSIP_RsassaPkcs2048SignatureGenerate	26,095,126	26,096,208	26,096,556
R_TSIP_RsassaPkcs2048SignatureVerification	133,690	134,768	135,116

Table 1.116 Performance of RSAES-PKCS1-v1_5 Encryption/Decryption with 1,024-Bit Key Size

API	Performance (Unit: cycle)	
	Message size=1byte	Message size=117byte
R_TSIP_RsaesPkcs1024Encrypt	20,106	15,632
R_TSIP_RsaesPkcs1024Decrypt	1,232,298	1,232,288

Table 1.117 Performance of RSAES-PKCS1-v1_5 Encryption/Decryption with 2,048-Bit Key Size

API	Performance (Unit: cycle)	
	Message size=1byte	Message size=245byte
R_TSIP_RsaesPkcs2048Encrypt	142,650	133,110
R_TSIP_RsaesPkcs2048Decrypt	26,094,662	26,094,664

Table 1.118 Performance of HASH (SHA1)

API	Performance (Unit: cycle)		
	128-byte processing	192-byte processing	256-byte processing
R_TSIP_Sha1Init	108	108	106
R_TSIP_Sha1Update	1,248	1,452	1,656
R_TSIP_Sha1Final	668	668	668

Table 1.119 Performance of HASH (SHA256)

API	Performance (Unit: cycle)		
	128-byte processing	192-byte processing	256-byte processing
R_TSIP_Sha256Init	152	152	152
R_TSIP_Sha256Update	1,270	1,444	1,618
R_TSIP_Sha256Final	682	682	684

Table 1.120 Performance of HASH (MD5)

API	Performance (Unit: cycle)		
	128-byte processing	192-byte processing	256-byte processing
R_TSIP_Md5Init	100	98	98
R_TSIP_Md5Update	1,152	1,326	1,500
R_TSIP_Md5Final	638	638	638

Table 1.121 Performance of Common API (HMAC User Key Index Generation)

API	Performance (Unit: cycle)
R_TSIP_GenerateSha1HmacKeyIndex	2,344
R_TSIP_GenerateSha256HmacKeyIndex	2,342
R_TSIP_UpdateSha1HmacKeyIndex	2,102
R_TSIP_UpdateSha256HmacKeyIndex	2,098

Table 1.122 Performance of HMAC (SHA1)

API	Performance (Unit: cycle)		
	128-byte processing	192-byte processing	256-byte processing
R_TSIP_Sha1HmacGenerateInit	1,082	1,082	1,082
R_TSIP_Sha1HmacGenerateUpdate	798	1,000	1,204
R_TSIP_Sha1HmacGenerateFinal	1,612	1,610	1,608
R_TSIP_Sha1HmacVerifyInit	1,082	1,082	1,082
R_TSIP_Sha1HmacVerifyUpdate	802	1,006	1,210
R_TSIP_Sha1HmacVerifyFinal	2,748	2,746	2,746

Table 1.123 Performance of HMAC (SHA256)

API	Performance (Unit: cycle)		
	128-byte processing	192-byte processing	256-byte processing
R_TSIP_Sha256HmacGenerateInit	1,288	1,284	1,284
R_TSIP_Sha256HmacGenerateUpdate	730	902	1,076
R_TSIP_Sha256HmacGenerateFinal	1,580	1,576	1,576
R_TSIP_Sha256HmacVerifyInit	1,284	1,290	1,288
R_TSIP_Sha256HmacVerifyUpdate	728	902	1,078
R_TSIP_Sha256HmacVerifyFinal	2,732	2,732	2,730

Table 1.124 Performance of Common API (ECC User Key Index Generation)

API	Performance (Unit: cycle)
R_TSIP_GenerateEccP192PublicKeyIndex	2,654
R_TSIP_GenerateEccP224PublicKeyIndex	2,648
R_TSIP_GenerateEccP256PublicKeyIndex	2,644
R_TSIP_GenerateEccP384PublicKeyIndex	2,812
R_TSIP_GenerateEccP192PrivateKeyIndex	2,350
R_TSIP_GenerateEccP224PrivateKeyIndex	2,338
R_TSIP_GenerateEccP256PrivateKeyIndex	2,340
R_TSIP_GenerateEccP384PrivateKeyIndex	2,376
R_TSIP_GenerateEccP192RandomKeyIndex *1	134,428
R_TSIP_GenerateEccP224RandomKeyIndex *1	142,286
R_TSIP_GenerateEccP256RandomKeyIndex *1	143,597
R_TSIP_GenerateEccP384RandomKeyIndex *1	1,017,476
R_TSIP_UpdateEccP192PublicKeyIndex	2,404
R_TSIP_UpdateEccP224PublicKeyIndex	2,400
R_TSIP_UpdateEccP256PublicKeyIndex	2,400
R_TSIP_UpdateEccP384PublicKeyIndex	2,564
R_TSIP_UpdateEccP192PrivateKeyIndex	2,102
R_TSIP_UpdateEccP224PrivateKeyIndex	2,100
R_TSIP_UpdateEccP256PrivateKeyIndex	2,102
R_TSIP_UpdateEccP384PrivateKeyIndex	2,124

Note 1. Average value at 10 runs.

Table 1.125 Performance of ECDSA Signature Generation/Verification

API	Performance (Unit: cycle)		
	Message size=1byte	Message size=128byte	Message size=256byte
R_TSIP_EcdsaP192SignatureGenerate	163,192	161,976	164,204
R_TSIP_EcdsaP224SignatureGenerate	162,344	166,116	163,304
R_TSIP_EcdsaP256SignatureGenerate	166,352	170,716	165,332
R_TSIP_EcdsaP384SignatureGenerate*1	1,097,446		
R_TSIP_EcdsaP192SignatureVerification	306,456	305,734	302,274
R_TSIP_EcdsaP224SignatureVerification	325,532	324,166	324,466
R_TSIP_EcdsaP256SignatureVerification	332,486	331,068	331,416
R_TSIP_EcdsaP384SignatureVerification*1	2,116,204		

Note 1. Not include SHA384 calculation.

Table 1.126 Performance of Key Exchange

API	Performance (Unit: cycle)
R_TSIP_EcdhP256Init	40
R_TSIP_EcdhP256ReadPublicKey	332,310
R_TSIP_EcdhP256MakePublicKey	311,724
R_TSIP_EcdhP256CalculateSharedSecretIndex	350,778
R_TSIP_EcdhP256KeyDerivation	3,126
R_TSIP_EcdheP512KeyAgreement	3,165,030
R_TSIP_Rsa2048DhKeyAgreement	52,462,430

Key exchange performance (without 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

The TSIP driver depends upon the TSIP capabilities provided on the MCU. Use a model name from the RX231 Group, RX23W Group, RX65N, RX651 Group, RX66N Group, RX66T Group, RX671 Group, RX72M Group, RX72N Group, or RX72T Group that provides built-in TSIP.

2.2 Software Requirements

The TSIP driver is dependent on the following module:

r_bsp Use rev6.11 or later.

- When using the RX231 or RX23W (On the RX231, a portion of the comment below following "= Chip" differs.)

Change the following macro value to 0xB, or 0xD(Only RX23W) of the file r_bsp_config.h in the r_config folder.

```
/* 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)
```

- When using the RX66T or RX72T (On the RX72T, a portion of the comment below following "= PGA" differs.)

Change the value of 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)
```

- If using RX66N, RX671, RX72M, or RX72N

Change the value of the following macro of 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)
```

- If using RX65N

Change the value of the following macro of 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 with the following toolchain has been confirmed.

RX Family C/C++ Compiler Package V3.03.00

2.4 Header File

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

2.5 Integer Types

This project uses ANSI C99.

2.6 API Data Structure

For the data structures used in the TSIP driver, refer to r_tsip_rx_if.h.

2.7 Return Values

This shows the different values API functions can return. This enum is found in r_tsip_rx_if.h along with the API function declarations.

```
typedef enum e_tsip_err
{
    TSIP_SUCCESS=0,
    TSIP_ERR_FAIL,                      // Self-check failed to terminate normally, or
                                        // Detected illegal MAC by using
                                        // R_TSIP_VerifyFirmwareMAC. or each R_TSIP_ 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,                     // Indicates that self-check terminated with an error. Run the
                                        // function again.
    TSIP_ERR_KEY_SET,                   // An error occuerd when setting the invalid key index.
    TSIP_ERR_AUTHENTICATION             // Authentication failed
    TSIP_ERR_CALLBACK_UNREGIST,         // Callback function is not registered.
    TSIP_ERR_PARAMETER,                 // Input date is illegal.
    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 TLS1.3 handshake 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" described in (1) or (3). However, "Smart Configurator" only supports some RX devices. Please use the methods of (2) or (4) for unsupported RX devices.

(1) Adding the FIT module to your project using "Smart Configurator" in e^2 studio
By using the "Smart Configurator" in e^2 studio, the FIT module is automatically added to your project. Refer to "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
By using the "FIT Configurator" in e^2 studio, the FIT module is automatically added to your project. Refer to "Adding Firmware Integration Technology Modules to Projects (R01AN1723)" for details.

(3) Adding the FIT module to your project using "Smart Configurator" on CS+
By using the "Smart Configurator Standalone version" in CS+, the FIT module is automatically added to your project. Refer to "Renesas e^2 studio Smart Configurator User Guide (R20AN0451)" for details.

(4) Adding the FIT module to your project in CS+
In CS+, please manually add the FIT module to your project. Refer to "Adding Firmware Integration Technology Modules to CS+ Projects (R01AN1826)" for details.

3. API Functions

3.1 List of API Functions

The TSIP driver implements the following API functions

(1) TSIP initialization-related API functions
(2) API to generate user key index data used in AES/DES/ARC4/RSA/ECC encryption and HMAC computation, API to generate key index data used for key updates, and API to update user key index data
(3) API functions for automatically generating AES/DES/ARC4/RSA/ECC user key index from random numbers
(4) API function for generating random numbers
(5) API for cryptographic algorithms
(6) API for securely updating firmware, booting up, etc.
(7) API for SSL/TLS cooperation function
(8) API for key exchange
(9) API for key wrap

Table 3.1 Table of APIs

Type	API	Description	TSIP -Lite	TSIP
(1)	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.	✔	✔
(2)	R_TSIP_GenerateAes128KeyIndex	Generates a 128-bit AES user key index.	✔	✔
	R_TSIP_GenerateAes256KeyIndex	Generates a 256-bit AES user key index.	✔	✔
	R_TSIP_GenerateUpdateKeyRingKeyIndex	Generates a keyring key index for key updating.	✔	✔
	R_TSIP_GenerateTdesKeyIndex	Generates a Triple-DES user key index.		✔
	R_TSIP_GenerateArc4KeyIndex	Generates a ARC4 user key index.		✔
	R_TSIP_GenerateRsa1024PrivateKeyIndex	Generates a 1024-bit RSA private key user key index.		✔
	R_TSIP_GenerateRsa1024PublicKeyIndex	Generates a 1024-bit RSA public key user key index.		✔
	R_TSIP_GenerateRsa2048PrivateKeyIndex	Generates a 2048-bit RSA private key user key index.		✔
	R_TSIP_GenerateRsa2048PublicKeyIndex	Generates a 2048-bit RSA public key user key index.		✔
	R_TSIP_GenerateTlsRsaPublicKeyIndex	Generates an RSA 2048-bit public key user key index used in TLS cooperation.		✔
	R_TSIP_GenerateEccP192PublicKeyIndex	Generates an ECC P-192 public key user key index.		✔
	R_TSIP_GenerateEccP224PublicKeyIndex	Generates an ECC P-224 public key user key index.		✔
	R_TSIP_GenerateEccP256PublicKeyIndex	Generates an ECC P-256 public key user key index.		✔
	R_TSIP_GenerateEccP384PublicKeyIndex	Generates an ECC P-384 public key user key index.		✔
	R_TSIP_GenerateEccP192PrivateKeyIndex	Generates an ECC P-192 private key user key index.		✔
	R_TSIP_GenerateEccP224PrivateKeyIndex	Generates an ECC P-224 private key user key index.		✔

Type	API	Description	TSIP -Lite	TSIP
	R_TSIP_GenerateEccP256PrivateKeyIndex	Generates an ECC P-256 private key user key index.		✔
	R_TSIP_GenerateEccP384PrivateKeyIndex	Generates an ECC P-384 private key user key index.		✔
	R_TSIP_GenerateSha1HmacKeyIndex	Generates a user key index for SHA1-HMAC computation.		✔
	R_TSIP_GenerateSha256HmacKeyIndex	Generates a user key index for SHA256-HMAC computation.		✔
(2)	R_TSIP_UpdateAes128KeyIndex	Updates an AES 128-bit user key index.	✔	✔
	R_TSIP_UpdateAes256KeyIndex	Updates an AES 256-bit user key index.	✔	✔
	R_TSIP_UpdateTdesKeyIndex	Updates a TDES user key index.		✔
	R_TSIP_UpdateArc4KeyIndex	Updates a ARC4 user key index.		✔
	R_TSIP_UpdateRsa1024PrivateKeyIndex	Updates the user key index for an RSA 1024-bit private key.		✔
	R_TSIP_UpdateRsa1024PublicKeyIndex	Updates the user key index for an RSA 1024-bit public key.		✔
	R_TSIP_UpdateRsa2048PrivateKeyIndex	Updates the user key index for an RSA 2048-bit private key.		✔
	R_TSIP_UpdateRsa2048PublicKeyIndex	Updates the user key index for an RSA 2048-bit public key.		✔
	R_TSIP_UpdateEccP192PublicKeyIndex	Updates the user key index for an ECC P-192 public key		✔
	R_TSIP_UpdateEccP224PublicKeyIndex	Updates the user key index for an ECC P-224 public key		✔
	R_TSIP_UpdateEccP256PublicKeyIndex	Updates the user key index for an ECC P-256 public key		✔
	R_TSIP_UpdateEccP384PublicKeyIndex	Updates the user key index for an ECC P-384 public key		✔
	R_TSIP_UpdateEccP192PrivateKeyIndex	Updates the user key index for an ECC P-192 private key		✔
	R_TSIP_UpdateEccP224PrivateKeyIndex	Updates the user key index for an ECC P-224 private key		✔
	R_TSIP_UpdateEccP256PrivateKeyIndex	Updates the user key index for an ECC P-256 private key		✔
	R_TSIP_UpdateEccP384PrivateKeyIndex	Updates the user key index for an ECC P-384 private key		✔
	R_TSIP_UpdateSha1HmacKeyIndex	Updates a user key index for SHA1-HMAC computation.		✔
	R_TSIP_UpdateSha256HmacKeyIndex	Updates a user key index for SHA256-HMAC computation.		✔
(3)	R_TSIP_GenerateAes128RandomKeyIndex	Generates a random128-bit AES user key index.	✔	✔
	R_TSIP_GenerateAes256RandomKeyIndex	Generates a random 256-bit AES user key index.	✔	✔
	R_TSIP_GenerateTdesRandomKeyIndex	Generates a random Triple-DES user key index.		✔
	R_TSIP_GenerateArc4RandomKeyIndex	Generates a random ARC4 user key index.		✔
	R_TSIP_GenerateRsa1024RandomKeyIndex	Generates a public key corresponding to the user key index for an RSA 1024-bit private key. The public key exponent is fixed at 0x10001.		✔

Type	API	Description	TSIP -Lite	TSIP
	R_TSIP_GenerateRsa2048RandomKeyIndex	Generates a public key corresponding to the user key index for an RSA 2048-bit private key. The public key exponent is fixed at 0x10001.		✔
	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_GenerateTls13P256EccKeyIndex	Generates a key pair from a random number used by the TLS1.3 cooperation function for elliptic curve cryptography over a 256-bit prime field.		✔
	R_TSIP_GenerateEccP192RandomKeyIndex	Generates a public key corresponding to the user key index for an ECC P-192 private key.		✔
	R_TSIP_GenerateEccP224RandomKeyIndex	Generates a public key corresponding to the user key index for an ECC P-224 private key.		✔
	R_TSIP_GenerateEccP256RandomKeyIndex	Generates a public key corresponding to the user key index for an ECC P-256 private key.		✔
	R_TSIP_GenerateEccP384RandomKeyIndex	Generates a public key corresponding to the user key index for an ECC P-384 private key.		✔
(4)	R_TSIP_GenerateRandomNumber	Generates a random number.	✔	✔
(5)	R_TSIP_Aes128EcbEncryptInit	Prepares to encrypt data in AES128-ECB mode using a 128-bit AES user key index.	✔	✔
	R_TSIP_Aes128EcbEncryptUpdate	Encrypts data in AES128-ECB mode.	✔	✔
	R_TSIP_Aes128EcbEncryptFinal	Performs final processing for encryption in AES128-ECB mode.	✔	✔
	R_TSIP_Aes128EcbDecryptInit	Prepares to decrypt data in AES128-ECB mode using a 128-bit AES user key index.	✔	✔
	R_TSIP_Aes128EcbDecryptUpdate	Decrypts data in AES128-ECB mode.	✔	✔
	R_TSIP_Aes128EcbDecryptFinal	Performs final processing for decryption in AES128-ECB mode.	✔	✔
	R_TSIP_Aes256EcbEncryptInit	Prepares to encrypt data in AES256-ECB mode using a 256-bit AES user key index.	✔	✔
	R_TSIP_Aes256EcbEncryptUpdate	Encrypts data in AES256-ECB mode.	✔	✔
	R_TSIP_Aes256EcbEncryptFinal	Performs final processing for encryption in AES256-ECB mode.	✔	✔
	R_TSIP_Aes256EcbDecryptInit	Prepares to decrypt data in AES256-ECB mode using a 256-bit AES user key index.	✔	✔
	R_TSIP_Aes256EcbDecryptUpdate	Decrypts data in AES256-ECB mode.	✔	✔
	R_TSIP_Aes256EcbDecryptFinal	Performs final processing for decryption in AES256-ECB mode.	✔	✔
	R_TSIP_Aes128CbcEncryptInit	Prepares to encrypt data in AES128-CBC mode using a 128-bit AES user key index.	✔	✔
	R_TSIP_Aes128CbcEncryptUpdate	Encrypts data in AES128-CBC mode.	✔	✔

Type	API	Description	TSIP -Lite	TSIP
	R_TSIP_Aes128CbcEncryptFinal	Performs final processing for encryption in AES128-CBC mode.	✔	✔
	R_TSIP_Aes128CbcDecryptInit	Prepares to decrypt data in AES128-CBC mode using a 128-bit AES user key index.	✔	✔
	R_TSIP_Aes128CbcDecryptUpdate	Decrypts data in AES128-CBC mode.	✔	✔
	R_TSIP_Aes128CbcDecryptFinal	Performs final processing for to decryption in AES128-CBC mode.	✔	✔
	R_TSIP_Aes256CbcEncryptInit	Prepares to encrypt data in AES256-CBC mode using a 256-bit AES user key index.	✔	✔
	R_TSIP_Aes256CbcEncryptUpdate	Encrypts data in AES256-CBC mode.	✔	✔
	R_TSIP_Aes256CbcEncryptFinal	Performs final processing for encryption in AES256-CBC mode.	✔	✔
	R_TSIP_Aes256CbcDecryptInit	Prepares to decrypt data in AES256-CBC mode using a 256-bit AES user key index.	✔	✔
	R_TSIP_Aes256CbcDecryptUpdate	Decrypts data in AES256-CBC mode.	✔	✔
	R_TSIP_Aes256CbcDecryptFinal	Performs final processing for decryption in AES256-CBC mode.	✔	✔
(5)	R_TSIP_Aes128GcmEncryptInit	Prepares to encrypt data in AES128-GCM mode using a 128-bit AES user key index.	✔	✔
	R_TSIP_Aes128GcmEncryptUpdate	Encrypts data in AES128-GCM mode.	✔	✔
	R_TSIP_Aes128GcmEncryptFinal	Prepares final processing for encryption in AES128-GCM mode.	✔	✔
	R_TSIP_Aes128GcmDecryptInit	Prepares to decrypt data in AES128-GCM mode using a 128-bit AES user key index.	✔	✔
	R_TSIP_Aes128GcmDecryptUpdate	Decrypts data in AES128-GCM mode.	✔	✔
	R_TSIP_Aes128GcmDecryptFinal	Prepares final processing for decryption in AES128-GCM mode.	✔	✔
	R_TSIP_Aes256GcmEncryptInit	Prepares to encrypt data in AES256-GCM mode using a 256-bit AES user key index.	✔	✔
	R_TSIP_Aes256GcmEncryptUpdate	Encrypts data in AES256-GCM mode.	✔	✔
	R_TSIP_Aes256GcmEncryptFinal	Prepares final processing for encryption in AES256-GCM mode.	✔	✔
	R_TSIP_Aes256GcmDecryptInit	Prepares to decrypt data in AES256-GCM mode using a 256-bit AES user key index.	✔	✔
	R_TSIP_Aes256GcmDecryptUpdate	Decrypts data in AES256-GCM mode.	✔	✔
	R_TSIP_Aes256GcmDecryptFinal	Prepares final processing for decryption in AES256-GCM mode.	✔	✔
	R_TSIP_Aes128CcmEncryptInit	Prepares to encrypt data in AES128-CCM mode using an AES 128-bit user key index.	✔	✔
	R_TSIP_Aes128CcmEncryptUpdate	Encrypts data in AES128-CCM mode.	✔	✔
	R_TSIP_Aes128CcmEncryptFinal	Performs final processing for encryption in AES128-CCM mode.	✔	✔
	R_TSIP_Aes128CcmDecryptInit	Prepares to decrypt data in AES128-CCM mode using an AES 128-bit user key index.	✔	✔

Type	API	Description	TSIP -Lite	TSIP
	R_TSIP_Aes128CcmDecryptUpdate	Decrypts data in AES128-CCM mode.	✔	✔
	R_TSIP_Aes128CcmDecryptFinal	Performs final processing for decryption in AES128-CCM mode.	✔	✔
	R_TSIP_Aes256CcmEncryptInit	Prepares to encrypt data in AES256-CCM mode using an AES 256-bit user key index.	✔	✔
	R_TSIP_Aes256CcmEncryptUpdate	Encrypts data in AES256-CCM mode.	✔	✔
	R_TSIP_Aes256CcmEncryptFinal	Performs final processing for encryption in AES256-CCM mode.	✔	✔
	R_TSIP_Aes256CcmDecryptInit	Prepares to decrypt data in AES256-CCM mode using an AES 256-bit user key index.	✔	✔
	R_TSIP_Aes256CcmDecryptUpdate	Decrypts data in AES256-CCM mode.	✔	✔
	R_TSIP_Aes256CcmDecryptFinal	Performs final processing for decryption in AES256-CCM mode.	✔	✔
	R_TSIP_Aes128CmacGenerateInit	Prepares to generate the AES128-MAC in CMAC mode using 128-bit AES user key index.	✔	✔
(5)	R_TSIP_Aes128CmacGenerateUpdate	Generates the MAC in AES128-CMAC mode.	✔	✔
	R_TSIP_Aes128CmacGenerateFinal	Performs final processing for MAC generation in AES128-CMAC mode.	✔	✔
	R_TSIP_Aes128CmacVerifyInit	Verifies the MAC generated in AES128-CMAC mode using 128-bit AES user key index.	✔	✔
	R_TSIP_Aes128CmacVerifyUpdate	Prepares to verify the MAC generated in AES128-CMAC mode.	✔	✔
	R_TSIP_Aes128CmacVerifyFinal	Performs final processing to verify the MAC generated in AES128-CMAC mode.	✔	✔
	R_TSIP_Aes256CmacGenerateInit	Prepares to generate the MAC in AES256-CMAC mode using 256-bit AES user key index.	✔	✔
	R_TSIP_Aes256CmacGenerateUpdate	Generates the MAC in AES256-CMAC.	✔	✔
	R_TSIP_Aes256CmacGenerateFinal	Performs final processing for MAC generation in AES256-CMAC mode.	✔	✔
	R_TSIP_Aes256CmacVerifyInit	Prepares to verify the MAC generated in AES256-CMAC mode using 256-bit AES user key index.	✔	✔
	R_TSIP_Aes256CmacVerifyUpdate	Verifies the MAC generated in AES256-CMAC mode.	✔	✔
	R_TSIP_Aes256CmacVerifyFinal	Performs final processing for MAC generation in AES256-CMAC mode.	✔	✔
	R_TSIP_TdesEcbEncryptInit	Prepares to encrypt data in TDES-ECB mode using a TDES user key index.		✔
	R_TSIP_TdesEcbEncryptUpdate	Encrypts data in TDES-ECB mode.		✔
	R_TSIP_TdesEcbEncryptFinal	Performs final processing for encryption in TDES-ECB mode.		✔
	R_TSIP_TdesEcbDecryptInit	Prepares to decrypt data in TDES- ECB mode using a TDES user key index.		✔
	R_TSIP_TdesEcbDecryptUpdate	Decrypts data in TDES-ECB mode.		✔
	R_TSIP_TdesEcbDecryptFinal	Performs final processing for decryption in TDES-ECB mode.		✔

Type	API	Description	TSIP -Lite	TSIP
	R_TSIP_TdesCbcEncryptInit	Prepares to encrypt data in TDES-CBC mode using a TDES user key index.		✔
	R_TSIP_TdesCbcEncryptUpdate	Encrypts data in TDES-CBC mode.		✔
	R_TSIP_TdesCbcEncryptFinal	Performs final processing for encryption in TDES-CBC mode.		✔
	R_TSIP_TdesCbcDecryptInit	Prepares to decrypt data in TDES- CBC mode using a TDES user key index.		✔
	R_TSIP_TdesCbcDecryptUpdate	Decrypts data in TDES-CBC mode.		✔
	R_TSIP_TdesCbcDecryptFinal	Performs final processing for decryption in TDES-CBC mode.		✔
	R_TSIP_Arc4EncryptInit	Prepares to encrypt data in ARC4 using a ARC4 user key index.		✔
	R_TSIP_Arc4EncryptUpdate	Encrypts data in ARC4.		✔
	R_TSIP_Arc4EncryptFinal	Performs final processing for encryption in ARC4.		✔
	R_TSIP_Arc4DecryptInit	Prepares to decrypt data in ARC4 using a ARC4 user key index.		✔
	R_TSIP_Arc4DecryptUpdate	Decrypts data in ARC4.		✔
	R_TSIP_Arc4DecryptFinal	Performs final processing for decryption in ARC4.		✔
	R_TSIP_RsaesPkcs1024Encrypt	Encrypts a 1024-bit key based on RSAES-PKCS1-V1_5.		✔
	R_TSIP_RsaesPkcs1024Decrypt	Decrypts a 1024-bit key based on RSAES-PKCS1-V1_5.		✔
	R_TSIP_RsaesPkcs2048Encrypt	Encrypts a 2048-bit key based on RSAES-PKCS1-V1_5.		✔
(5)	R_TSIP_RsaesPkcs2048Decrypt	Decrypts a 2048-bit key based on RSAES-PKCS1-V1_5.		✔
	R_TSIP_RsassaPkcs1024SignatureGenerate	Generates a 1024-bit digital signature based on RSASSA-PKCS1-V1_5.		✔
	R_TSIP_RsassaPkcs1024SignatureVerification	Verifies a 1024-bit digital signature based on RSASSA-PKCS1-V1_5.		✔
	R_TSIP_RsassaPkcs2048SignatureGenerate	Generates a digital signature based on RSASSA-PKCS1-V1_5.		✔
	R_TSIP_RsassaPkcs2048SignatureVerification	Verifies a digital signature based on RSASSA-PKCS1-V1_5.		✔
	R_TSIP_Sha1Init	Prepares to perform hash value generation based on SHA-1.		✔
	R_TSIP_Sha1Update	Performs hash value generation based on SHA-1.		✔
	R_TSIP_Sha1Final	Performs final processing for hash value generation based on SHA-1.		✔
	R_TSIP_Sha256Init	Prepares to perform hash value generation based on SHA-256.		✔
	R_TSIP_Sha256Update	Performs hash value generation based on SHA-256.		✔
	R_TSIP_Sha256Final	Performs final processing for hash value generation based on SHA-256.		✔
	R_TSIP_Sha1HmacGenerateInit	Prepares to perform SHA1-HMAC calculation.		✔
	R_TSIP_Sha1HmacGenerateUpdate	Performs SHA1-HMAC calculation.		✔

Type	API	Description	TSIP -Lite	TSIP
	R_TSIP_Sha1HmacGenerateFinal	Performs final processing for SHA1-HMAC calculation.		✔
	R_TSIP_Sha256HmacGenerateInit	Prepares to perform SHA256-HMAC calculation.		✔
	R_TSIP_Sha256HmacGenerateUpdate	Performs SHA256-HMAC calculation.		✔
	R_TSIP_Sha256HmacGenerateFinal	Performs final processing for SHA256-HMAC calculation.		✔
	R_TSIP_Sha1HmacVerifyInit	Prepares to verify SHA1-HMAC calculation.		✔
	R_TSIP_Sha1HmacVerifyUpdate	Verifies SHA1-HMAC calculation.		✔
	R_TSIP_Sha1HmacVerifyFinal	Performs final processing for verification of SHA1-HMAC calculation.		✔
	R_TSIP_Sha256HmacVerifyInit	Prepares to verify SHA256-HMAC calculation.		✔
	R_TSIP_Sha256HmacVerifyUpdate	Verifies SHA256-HMAC calculation.		✔
	R_TSIP_Sha256HmacVerifyFinal	Performs final processing for verification of SHA256-HMAC calculation.		✔
	R_TSIP_Md5Init	Prepares to perform hash value generation based on MD5.		✔
	R_TSIP_Md5Update	Performs hash value generation based on MD5.		✔
	R_TSIP_Md5Final	Performs final processing for hash value generation based on MD5.		✔
	R_TSIP_EcdsaP192SignatureGenerate	Generates a digital signature based on ECDSA P-192		✔
	R_TSIP_EcdsaP224SignatureGenerate	Generates a digital signature based on ECDSA P-224		✔
(5)	R_TSIP_EcdsaP256SignatureGenerate	Generates a digital signature based on ECDSA P-256		✔
	R_TSIP_EcdsaP384SignatureGenerate	Generates a digital signature based on ECDSA P-384		✔
	R_TSIP_EcdsaP192SignatureVerification	Verifies a digital signature based on ECDSA P-192		✔
	R_TSIP_EcdsaP224SignatureVerification	Verifies a digital signature based on ECDSA P-224		✔
	R_TSIP_EcdsaP256SignatureVerification	Verifies a digital signature based on ECDSA P-256		✔
	R_TSIP_EcdsaP384SignatureVerification	Verifies a digital signature based on ECDSA P-384		✔
(6)	R_TSIP_StartUpdateFirmware	Transitions to firmware update mode.	✔	✔
	R_TSIP_GenerateFirmwareMAC	Decrypts and generates the MAC for the encrypted firmware.	✔	✔
	R_TSIP_VerifyFirmwareMAC	Performs a MAC check on the firmware.	✔	✔
(7)	R_TSIP_TlsRootCertificateVerification	Verifies the root CA certificate bundle.		✔
	R_TSIP_TlsCertificateVerification	Verifies the server certificate and intermediate certificate.		✔
	R_TSIP_TlsGeneratePreMasterSecretWithRsa2048PublicKey	Generates the encrypted PreMasterSecret.		✔
	R_TSIP_TlsEncryptPreMasterSecret	Encrypts the PreMasterSecret using RSA-2048.		✔
	R_TSIP_TlsGenerateMasterSecret	Generates the encrypted MasterSecret.		✔
	R_TSIP_TlsGenerateSessionKey	Outputs TLS communication keys.		✔

Type	API	Description	TSIP -Lite	TSIP
	R_TSIP_TlsGenerateVerifyData	Generates VerifyData.		✔
	R_TSIP_TlsServersEphemeralEcdhPublicKeyRetrieves	Verifies a ServerKeyExchange signature.		✔
	R_TSIP_TlsGeneratePreMasterSecretWithEccP256Key	Generates an ECC encrypted PreMasterSecret.		✔
	R_TSIP_Tls13GenerateEcdheSharedSecret	Generates a SharedSecret key index.		✔
	R_TSIP_Tls13GenerateHandshakeSecret	Generates a HandshakeSecret key index.		✔
	R_TSIP_Tls13GenerateServerHandshakeTrafficKey	Generates a ServerWriteKey key index and a ServerFinishedKey key index.		✔
	R_TSIP_Tls13GenerateServerHandshakeVerification	Verifys Finished provided by the server.		✔
	R_TSIP_Tls13GenerateClientHandshakeTrafficKey	Generates a ClientWriteKey key index and a ClientFinishedKey key index.		✔
	R_TSIP_Tls13GenerateMasterSecret	Generates a MasterSecret key index.		✔
	R_TSIP_Tls13GenerateApplicationTrafficKey	Generates ApplicationTrafficSecret key indexes and AppicatonTrafficKey key indexes.		✔
	R_TSIP_Tls13UpdateApplicationTrafficKey	Updates an ApplicationTrafficSecret key index and an AppicatonTrafficKey key index.		✔
	R_TSIP_Tls13EncryptInit	Prepares to encrypt data used by the TLS1.3 cooperation function.		✔
	R_TSIP_Tls13EncryptUpdate	Encrypts data used by the TLS1.3 cooperation function.		✔
	R_TSIP_Tls13EncryptFinal	Prepares final processing for encryption used by the TLS1.3 cooperation function.		✔
	R_TSIP_Tls13DecryptInit	Prepares to decrypt data used by the TLS1.3 cooperation function.		✔
	R_TSIP_Tls13DecryptUpdate	Decrypts data used by the TLS1.3 cooperation function.		✔
	R_TSIP_Tls13DecryptFinal	Prepares final processing for decryption used by the TLS1.3 cooperation function.		✔
	R_TSIP_Tls13CertificateVerifyGenerate	Generates CertificateVerify used by the TLS1.3 cooperation function.		✔
	R_TSIP_Tls13CertificateVerifyVerification	Verifies CertificateVerify used by the TLS1.3 cooperation function.		✔
(8)	R_TSIP_EcdP256hInit	Prepares to perform ECDH P-256 key exchange computation.		✔
	R_TSIP_EcdhP256ReadPublicKey	Verifies the ECC P-256 public key signature of the other key exchange party.		✔
	R_TSIP_EcdhMakeP256PublicKey	Signs the ECC P-256 private key.		✔
	R_TSIP_EcdhP256CalculateSharedSecretIndex	Computes the shared secret Z from the public key of the other key exchange party and your own public key.		✔
	R_TSIP_EcdhP256KeyDerivation	Derives Z from the shared key.		✔
	R_TSIP_EcdheP512KeyAgreement	Calculate ECDHE key agreement using Brainpool P512r1		✔

Type	API	Description	TSIP -Lite	TSIP
	R_TSIP_Rsa2048DhKeyAgreement	Calculate DH key agreement using RSA-2048		✔
(9)	R_TSIP_Aes128KeyWrap	Wraps a key with an AES 128 key.	✔	✔
	R_TSIP_Aes256KeyWrap	Unwraps a key wrapped with an AES 128 key.	✔	✔
	R_TSIP_Aes128KeyUnwrap	Wraps a key with an AES 256 key.	✔	✔
	R_TSIP_Aes256KeyUnwrap	Unwraps a key wrapped with an AES 256 key.	✔	✔

3.2 State Transition Diagram

The TSIP monitors TSIP register access using software. The TSIP allows execution of an API function from the appropriate state transition source. If the TSIP detects illegal TSIP register access, it transitions to the TSIP illegal access detected state and infinite loop will be occurred in next R_TSIP_...() functions call. It is recommended to use the watch-dog timer to detect this infinite loop and reboot the system. The TSIP state transition diagram is shown below.

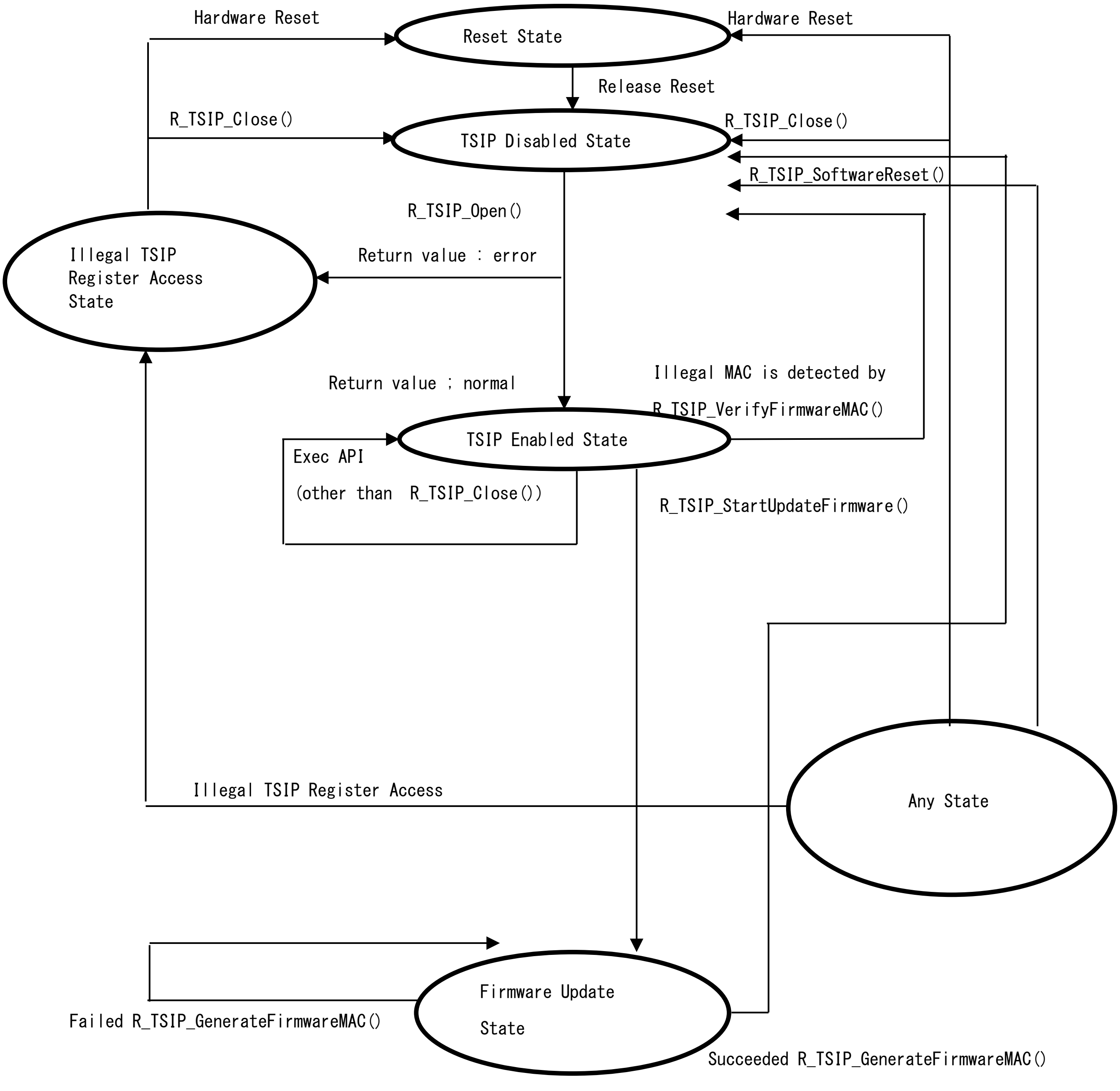


Figure 3.1 TSIP State Transition Diagram

Note: Always transition the RX to the standby mode from the TSIP operation halted state. Note that transitioning the RX to the standby mode from any state other than the TSIP operation halted state will increase current consumption. To avoid this, R_TSIP_Open() calls R_BSP_InterruptsDisable(), and R_BSP_InterruptsEnable().

3.3 Notes on API Usage

Each time one of the algorithm APIs of the TSIP driver is run, it is necessary to call the Init API, Update API, and Final API, in that order. It is not possible to use multiple algorithms at once. For example, it is not possible to call R_TSIP_Aes128EcbEncryptInit() and then, before calling R_TSIP_Aes128EcbEncryptFinal(), to call R_TSIP_Aes128EcbDecryptInit() in order to encrypt and decrypt AES-ECB 128 keys at the same time. If functions are not called in the correct order, a value of TSIP_ERR_RESOURCE_CONFLICT or TSIP_ERR_PROHIBIT_FUNCTION will be returned.

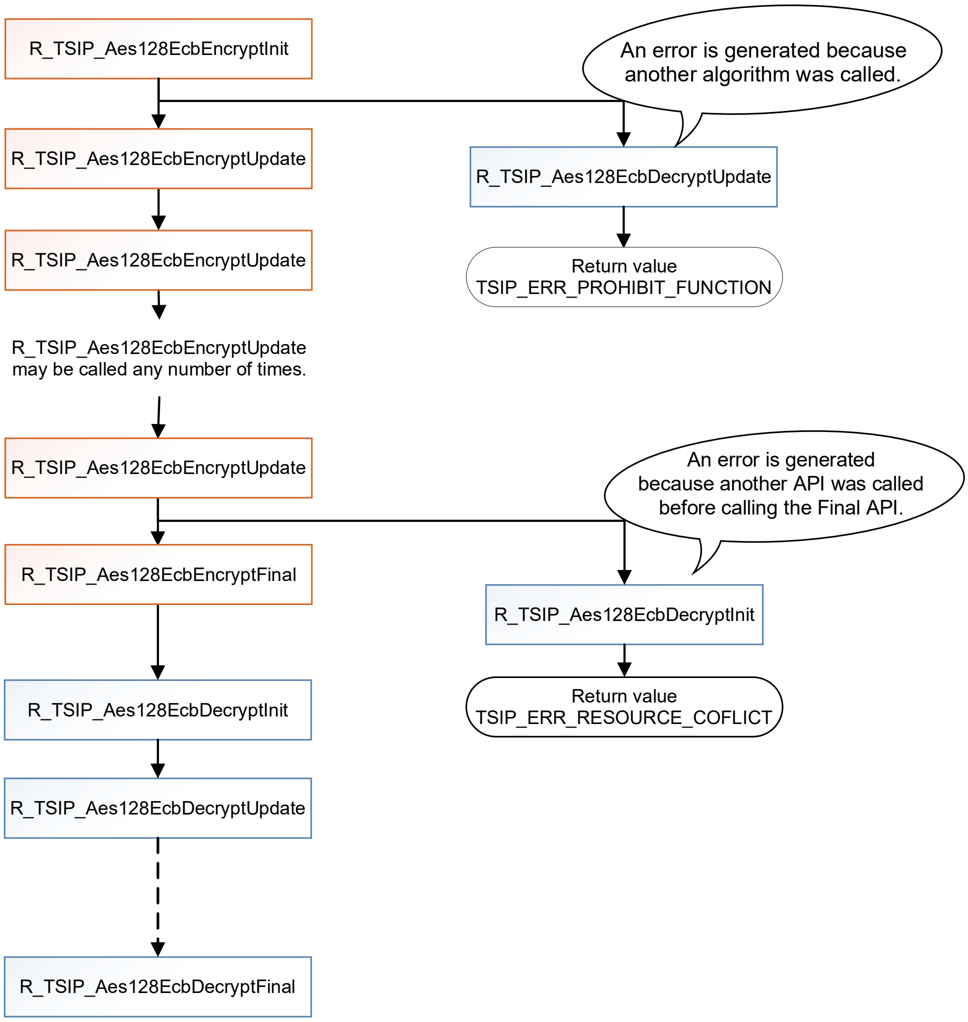


Figure 3.2 Example Use of AES-ECB 128 Encryption and Decryption Algorithms

4. Detailed Description of API Functions (for both TSIP and TSIP-Lite)

4.1 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	Key update keyring key index

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_FAIL:	The error-detection self-test failed to terminate normally.
TSIP_ERR_RESOURCE_CONFLICT:	A resource conflict occurred because a hardware resource needed by the processing routine was in use by another processing routine.
TSIP_ERR_RETRY:	Indicates that an entropy evaluation failure occurred. Run the function again.

Description

Enables use of TSIP functionality.

For key_index_1, input the "key index of TLS cooperation RSA public key" 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 "keyring key index for key update" generated by R_TSIP_GenerateUpdateKeyRingKeyIndex(). If the key update cooperation function is not used, input a null pointer.

<State transition>

The valid pre-run state is *TSIP disabled.*

The pre-run state is *TSIP Disabled State*.

After the function runs the state transitions to *TSIP Enabled State.*

Reentrant

Not supported

4.2 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 supply of power to the TSIP.

<State transition>

The pre-run state is *any* state.

After the function runs the state transitions to *TSIP Disabled State*.

Reentrant

Not supported

4.3 R_TSIP_SoftwareReset

Format

```
#include "r_tsip_rx_if.h"
void    R_TSIP_SoftwareReset (void);
```

Parameters

None.

Return Values

None.

Description

Reverts the state to the TSIP initial state.

<State transition>

The pre-run state is *any* state.

After the function runs the state transitions to *TSIP Disabled State*.

Reentrant

Not supported

4.4 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 get the TSIP driver version.

<State transition>

The pre-run state is *TSIP Enabled State*.

After the function runs the state transitions to *TSIP Enabled State*.

Reentrant

Not supported

4.5 R_TSIP_GenerateAes128KeyIndex

Format

#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);

Parameters

encrypted_provisioning_key	Input	Provisioning key wrapped by the DLM server
iv	Input	Initialization vector when generating encrypted_key
encrypted_key	Input	User key encryptedand MAC appended
key_index	Input/output	User key index

Return Values

TSIP_SUCCESS :	Normal termination
TSIP_ERR_RESOURCE_CONFLICT:	A resource conflict occurred because a hardware resource needed by the processing routine was in use by another processing routine.
TSIP_ERR_FAIL :	An internal error occurred.

Description

This API outputs 128-bit AES user key index.

Input data encrypted in the following format with the provisioning key as encrypted_key.

byte	128-bit			
	32-bit	32-bit	32-bit	32-bit
0-15	AES 128 key			
16-31	MAC			

<State transition>

The pre-run state is *TSIP Enabled State*.

After the function runs the state transitions to *TSIP Enabled State*.

For the method of generating encrypted_provisioning_key, iv, and encrypted_key, and instructions for using key_index, refer to Chapter 7, Key Data Operations.

Reentrant

Not supported

4.6 R_TSIP_GenerateAes256KeyIndex

Format

```
#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	User key encrypted and MAC appended
key_index	Input/output	User key index

Return Values

TSIP_SUCCESS :	Normal termination
TSIP_ERR_RESOURCE_CONFLICT:	A resource conflict occurred because a hardware resource needed by the processing routine was in use by another processing routine.
TSIP_ERR_FAIL:	An internal error occurred.

Description

This API outputs 256-bit AES user key index.

Input data encrypted in the following format with the provisioning key as encrypted_key.

byte	128-bit			
	32-bit	32-bit	32-bit	32-bit
0-15	AES 256 key			
16-31				
32-47	MAC			

<State transition>

The pre-run state is *TSIP Enabled State*.

After the function runs the state transitions to *TSIP Enabled State*.

For the method of generating encrypted_provisioning_key, iv, and encrypted_key, and instructions for using key_index, refer to Chapter 7, Key Data Operations.

Reentrant

Not supported

4.7 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	User key encrypted and MAC appended
key_index	Input/output	Key update keyring key index

Return Values

TSIP_SUCCESS:	Normal end
TSIP_ERR_RESOURCE_CONFLICT:	A resource conflict occurred because a hardware resource needed by this processing routine was in use by another processing routine.
TSIP_ERR_FAIL:	An internal error occurred.

Description

This API outputs a key index for the key update keyring.

Input data encrypted in the following format with the provisining key as encrypted_key.

byte	128-bit			
	32-bit	32-bit	32-bit	32-bit
0-15	Key update keyring			
16-31				
32-47	MAC			

<State transition>

The pre-run state is *TSIP Enabled State*.

After the function runs the state transitions to *TSIP Enabled State*.

For the method of generating encrypted_provisioning_key, iv, and encrypted_key, and instructions for using key_index, refer to Chapter 7, Key Data Operations.

Reentrant

Not supported

4.8 R_TSIP_UpdateAes128KeyIndex

Format

```
#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);
```

Parameters

iv	Input	Initialization vector when generating encrypted_key
encrypted_key	Input	User key encrypted with key update keyring with MAC appended
key_index	Input/output	User key index

Return Values

TSIP_SUCCESS:	Normal end
TSIP_ERR_RESOURCE_CONFLICT:	A resource conflict occurred because a hardware resource needed by this processing routine was in use by another processing routine.
TSIP_ERR_FAIL:	An internal error occurred.

Description

This API updates the key index of an AES 128 key.

Input data encrypted in the following format with the key update keyring as encrypted_key.

byte	128-bit			
	32-bit	32-bit	32-bit	32-bit
0-15	AES 128 key			
16-31	MAC			

<State transition>

The pre-run state is *TSIP Enabled State*.

After the function runs the state transitions to *TSIP Enabled State*.

For the method of generating iv and encrypted_key, and instructions for using key_index, refer to Chapter 7, Key Data Operations.

Reentrant

Not supported

4.9 R_TSIP_UpdateAes256KeyIndex

Format

```
#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 with key update keyring with MAC appended
key_index	Input/output	User key index

Return Values

TSIP_SUCCESS:	Normal end
TSIP_ERR_RESOURCE_CONFLICT:	A resource conflict occurred because a hardware resource needed by this processing routine was in use by another processing routine.
TSIP_ERR_FAIL:	An internal error occurred.

Description

This API updates the key index of an AES 256 key.

Input data encrypted in the following format with the key update keyring as encrypted_key.

byte	128-bit			
	32-bit	32-bit	32-bit	32-bit
0-15	AES 256 key			
16-31				
32-47	MAC			

<State transition>

The pre-run state is *TSIP Enabled State*.

After the function runs the state transitions to *TSIP Enabled State*.

For the method of generating iv and encrypted_key, and instructions for using key_index, refer to Chapter 7, Key Data Operations.

Reentrant

Not supported

4.10 R_TSIP_GenerateAes128RandomKeyIndex

Format

```
#include "r_tsip_rx_if.h"
e_tsip_err_t R_TSIP_GenerateAes128RandomKeyIndex(tsip_aes_key_index_t *key_index);
```

Parameters

key_index	input/output	128-bit AES user key index (9 words)

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_RESOURCE_CONFLICT:	A resource conflict occurred because a hardware resource needed by the processing routine was in use by another processing routine.

Description

This API outputs 128-bit AES user key index.

This API generates a user key from a random number in the TSIP. Accordingly, user key input is unnecessary. By encrypting data using the user key index that is output by this API, dead copying of data can be prevented.

<State transition>

The pre-run state is *TSIP Enabled State*.

After the function runs the state transitions to *TSIP Enabled State*.

Refer to the Section 7 to use key_index.

Reentrant

Not supported

4.11 R_TSIP_GenerateAes256RandomKeyIndex

Format

#include “r_tsip_rx_if.h”

e_tsip_err_t R_TSIP_GenerateAes256RandomKeyIndex(tsip_aes_key_index_t *key_index);

Parameters

key_index	input/output	256-bit AES user key index

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_RESOURCE_CONFLICT:	A resource conflict occurred because a hardware resource needed by the processing routine was in use by another processing routine.

Description

This API outputs 256-bit AES user key index.

This API generates a user key from a random number in the TSIP. Accordingly, user key input is unnecessary. By encrypting data using the user key index that is output by this API, dead copying of data can be prevented.

<State transition>

The pre-run state is *TSIP Enabled State*.

After the function runs the state transitions to *TSIP Enabled State*.

Refer to the Section 7 to use key_index.

Reentrant

Not supported

4.12 R_TSIP_GenerateRandomNumber

Format

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

Parameters

random	input/output	Stores 4words (16 bytes) random data.

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_RESOURCE_CONFLICT:	A resource conflict occurred because a hardware resource needed by the processing routine was in use by another processing routine.

Description

This API can generate NIST SP800-90A compliant word of 4 random number.

<State transition>

The pre-run state is *TSIP Enabled State*.

After the function runs the state transitions to *TSIP Enabled State*.

Reentrant

Not supported

4.13 R_TSIP_StartUpdateFirmware

Format

#include “r_tsip_rx_if.h”

e_tsip_err_t R_TSIP_StartUpdateFirmware(void);

Parameters

none

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_RESOURCE_CONFLICT:	A resource conflict occurred because a hardware resource needed by the processing routine was in use by another processing routine.

Description

State Transit to the *Firm Update State*.

<State transition>

The pre-run state is *TSIP Enabled State*.

After the function runs the state transitions to *Firm Update State*.

Reentrant

Not supported

4.14 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	User key index area for decrypting InData_SessionKey and generating firmware MAC values
InData_SessionKey	input	Session key area for decrypting encrypted firmware and verifying checksum values
InData_UpProgram	input	512 words (2048 bytes) area for temporarily storing encrypted firmware data.
InData_IV	input	Initial vector area for decrypting the encrypted firmware.
OutData_Program	input/output	512 words (2048 bytes) area for temporarily storing decrypted firmware data.
MAX_CNT	input	The word size for encrypted firmware+MAC word size. Encrypted firmware value should be a multiple of 4. MAC word size is 4 words (128bit). Encrypted firmware data minimum size is 16 words, so, MAX_CNT minimum size is 20.
p_callback	input/output	It is called multiple times when user's action is required. The contents of teh action is determined by teh enum TSIP_FW_CB_REQ_TYPE.
tsip_firmware_generate_mac_resume_handle	input/output	R_TSIP_GenerateFirmwaraMAC handler (work area)

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_FAIL:	An internal error occurred.
TSIP_ERR_RESOURCE_CONFLICT:	A resource conflict occurred because a hardware resource needed by the processing routine was in use by another processing routine.
TSIP_ERR_KEY_SET:	Input illegal user Key Index.
TSIP_ERR_CALLBACK_UNREGIST:	p_callback value is illlegal.
TSIP_ERR_PARAMETER:	Input data is illegal.
TSIP_RESUME_FIRMWARE_GENERATE_MAC	There is additional processing. It is necessary to call the API again.

Description

This function decrypts the firmware and generates new MAC for the encrypted firmware and the firmware checksum value. User can update the firmware by writing the decrypted firmware and new MAC value to the Flash ROM.

The encryption algorithm uses AES-CBC and the MAC uses AES-CMAC.This API is called in the following order.

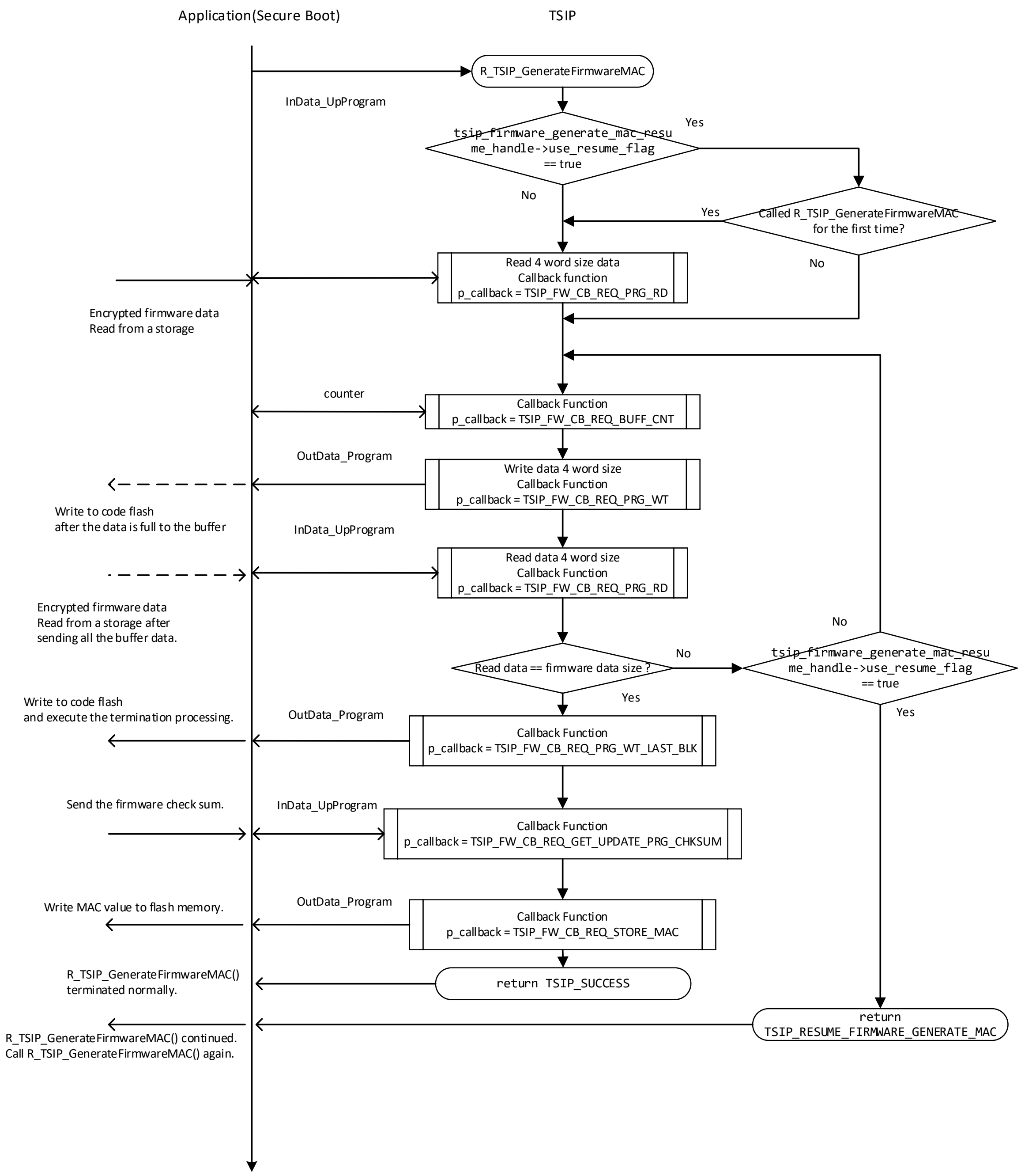


Figure 4.1 Flowchart of Calling of Callback Functions

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

1. Adjust increment (TSIP_FW_CB_REQ_BUFF_CNT).

2. Write decrypted firmware to storage destination (TSIP_FW_CB_REQ_PRG_WT).
3. Store 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 appropriate to the InData_Program and OutData_Program sizes that were reserved.

For example, if a 512-word buffer was reserved, adjust the increment to match the buffer position of 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).

For 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 of the all of the firmware has completed. Check that the 1st argument ”req_type”of the callback function p_callback is TSIP_FW_CB_REQ_GET_UPDATE_PRG_CHKSUM, then, pass the checksum value to the 4th argument ”InData_UpProgram” of p_callback. This API generates the firmware MAC value after reading the checksum value, when the checksum value is OK. MAC value is passed to the user using the 5th argument ”OutData_Program” when the 1st argument ”req_type” of callback function p_callback is TSIP_FW_CB_REQ_STORE_MAC. Store the MAC value in the flash 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 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.

<State transition>

The pre-run state is *Firm Update State*.

After the function runs the state transitions to *Firm Update State*.

Reentrant

Not supported

4.15 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	The word size for firmware+MAC word size. This value should be a multiple of 4. MAC word size is 4 words (16byte). Firmware data minimum size is 16 words, so, MAX_CNT minimum size is 20.
InData_MAC	input	MAC value to be compared (16byte)

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_FAIL:	Illegal MAC value
TSIP_ERR_RESOURCE_CONFLICT:	A resource conflict occurred because a hardware resource needed by the processing routine was in use by another processing routine.
TSIP_ERR_PARAMETER:	Input data is illegal.

Description

This function verifies the MAC value using firmware. This function will call firm_read_mac() function after all of firmware are read. Pass the MAC value that is generated by R_TSIP_GenerateFirmwareMAC(). For the 3rd argument “InData_Mac”, pass the MAC value generated by R_TSIP_GenerateFirmwareMAC().

The MAC verification algorithm uses AES-CMAC.

<State transition>

The pre-run state is *Firm Update State*.

After the function runs the state transitions to *Firm Update State*.

When illegal MAC value is detected, the state transitions to ***TSIP Illegal Access Detection State.***

4.16 R_TSIP_Aes128EcbEncryptInit

Format

#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);

Parameters

handle	input/output	AES handler (work area)
key_index	input	user key index area

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_KEY_SET:	Input illegal user key index.
TSIP_ERR_RESOURCE_CONFLICT:	A resource conflict occurred because a hardware resource needed by the processing routine was in use by another processing routine.
TSIP_ERR_FAIL:	An internal error occurred.

Description

The R_TSIP_Aes128EcbEncryptInit() function performs preparations for the execution of an AES calculation, and writes the result to the first argument, handle. The value of handle is used as an argument in the subsequent R_TSIP_Aes128EcbEncryptUpdate() function and R_TSIP_Aes128EcbEncryptFinal() function.

<State transition>

The pre-run state is *TSIP Enabled State*.

After the function runs the state transitions to *TSIP Enabled State*.

Refer to the Section 7 to generate key_index.

Reentrant

Not supported

4.17 R_TSIP_Aes128EcbEncryptUpdate

Format

#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	input/output	ciphertext data area
plain_length	input/output	plaintext data length (must be a multiple of 16)

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_PARAMETER:	Input data is illegal.
TSIP_ERR_PROHIBIT_FUNCTION:	An invalid function was called.

Description

The R_TSIP_Aes128EcbEncryptUpdate() function encrypts the second argument, plain, utilizing the key index stored in the handle specified in the first argument, handle, and writes the ongoing status to this first argument. In addition, it writes the encryption result to the third argument, cipher. After plaintext input is completed, call R_TSIP_Aes128EcbEncryptFinal().

Specify areas for plain and cipher that do not overlap. For plain and cipher, specify RAM addresses that are multiples of 4.

<State transition>

The pre-run state is *TSIP Enabled State*.

After the function runs the state transitions to *TSIP Enabled State*.

Refer to the Section 7 to generate key_index.

Reentrant

Not supported

4.18 R_TSIP_Aes128EcbEncryptFinal

Format

#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);

Parameters

handle	input/output	AES handler (work area)
cipher	input/output	ciphertext data area (nothing ever written here)
cipher_length	input/output	ciphertext data length (0 always written here)

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_FAIL:	An internal error occurred.
TSIP_ERR_PARAMETER:	Input data is illegal.
TSIP_ERR_PROHIBIT_FUNCTION:	An invalid function was called.

Description

Using the handle specified in the first argument, handle, the R_TSIP_Aes128EcbEncryptFinal() function writes the calculation result to the second argument, cipher, and writes the length of the calculation result to the third argument, cipher_length. The original intent was for a portion of the encryption result that was not a multiple of 16 bytes to be written to the second argument. However, as a result of the restriction that only multiples of 16 can be input to the Update function, nothing is ever written to cipher, and 0 is always written to cipher_length. The arguments cipher and cipher_length are provided for compatibility in anticipation of the time when this restriction is lifted.

<State transition>

The pre-run state is *TSIP Enabled State*.

After the function runs the state transitions to *TSIP Enabled State*.

Refer to the Section 7 to generate key_index.

Reentrant

Not supported

4.19 R_TSIP_Aes128EcbDecryptInit

Format

#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);

Parameters

handle	input/output	AES handler (work area)
key_index	input	user key index area

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_KEY_SET:	Input illegal user key index.
TSIP_ERR_RESOURCE_CONFLICT:	A resource conflict occurred because a hardware resource needed by the processing routine was in use by another processing routine.
TSIP_ERR_FAIL:	An internal error occurred.

Description

The R_TSIP_Aes128EcbDecryptInit() function performs preparations for the execution of an AES calculation, and writes the result to the first argument, handle. The value of handle is used as an argument in the subsequent R_TSIP_Aes128EcbDecryptUpdate() function and R_TSIP_Aes128EcbDecryptFinal() function.

<State transition>

The pre-run state is *TSIP Enabled State*.

After the function runs the state transitions to *TSIP Enabled State*.

Refer to the Section 7 to generate key_index.

Reentrant

Not supported

4.20 R_TSIP_Aes128EcbDecryptUpdate

Format

#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);

Parameters

handle	input/output	AES handler (work area)
cipher	input	ciphertext data area
plain	input/output	plaintext data area
cipher_length	input/output	ciphertext data length (must be a multiple of 16)

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_PARAMETER:	Input data is illegal.
TSIP_ERR_PROHIBIT_FUNCTION:	An invalid function was called.

Description

The R_TSIP_Aes128EcbDecryptUpdate() function decrypts the second argument, cipher, utilizing the key index stored in the handle specified in the first argument, handle, and writes the ongoing status to this first argument. In addition, it writes the decryption result to the third argument, plain. After ciphertext input is completed, call R_TSIP_Aes128EcbDecryptFinal().

Specify areas for plain and cipher that do not overlap. For plain and cipher, specify RAM addresses that are multiples of 4.

<State transition>

The pre-run state is *TSIP Enabled State*.

After the function runs the state transitions to *TSIP Enabled State*.

Refer to the Section 7 to generate key_index.

Reentrant

Not supported

4.21 R_TSIP_Aes128EcbDecryptFinal

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/output	AES handler (work area)
plain	input/output	plaintext data area (nothing ever written here)
plain_length	input/output	plaintext data length (0 always written here)

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_FAIL:	An internal error occurred.
TSIP_ERR_PARAMETER:	Input data is illegal.
TSIP_ERR_PROHIBIT_FUNCTION:	An invalid function was called.

Description

Using the handle specified in the first argument, handle, the R_TSIP_Aes128EcbDecryptFinal() function writes the calculation result to the second argument, plain, and writes the length of the calculation result to the third argument, plain_length. The original intent was for a portion of the decryption result that was not a multiple of 16 bytes to be written to the second argument. However, as a result of the restriction that only multiples of 16 can be input to the Update function, nothing is ever written to plain, and 0 is always written to plain_length. The arguments plain and plain_length are provided for compatibility in anticipation of the time when this restriction is lifted.

<State transition>

The pre-run state is *TSIP Enabled State*.

After the function runs the state transitions to *TSIP Enabled State*.

Refer to the Section 7 to generate key_index.

Reentrant

Not supported

4.22 R_TSIP_Aes256EcbEncryptInit

Format

#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	input/output	AES handler (work area)
key_index	input	user key index area

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_KEY_SET:	Input illegal user key index.
TSIP_ERR_RESOURCE_CONFLICT:	A resource conflict occurred because a hardware resource needed by the processing routine was in use by another processing routine.
TSIP_ERR_FAIL:	An internal error occurred.

Description

The R_TSIP_Aes256EcbEncryptInit() function performs preparations for the execution of an AES calculation, and writes the result to the first argument, handle. The value of handle is used as an argument in the subsequent R_TSIP_Aes256EcbEncryptUpdate() function and R_TSIP_Aes256EcbEncryptFinal() function.

<State transition>

The pre-run state is *TSIP Enabled State*.

After the function runs the state transitions to *TSIP Enabled State*.

Refer to the Section 7 to generate key_index.

Reentrant

Not supported

4.23 R_TSIP_Aes256EcbEncryptUpdate

Format

#include “r_tsip_rx_if.h”

e_tsip_err_t R_TSIP_Aes256EcbEncryptUpdate

(tsip_aes_handle_t *handle, uint8_t *plain, uint8_t *cipher, uint32_t plain_length);

Parameters

handle	input/output	AES handler (work area)
plain	input	plaintext data area
cipher	input/output	ciphertext data area
plain_length	input/output	plaintext data length (must be a multiple of 16)

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_PARAMETER:	Input data is illegal.
TSIP_ERR_PROHIBIT_FUNCTION:	An invalid function was called.

Description

The R_TSIP_Aes256EcbEncryptUpdate() function encrypts the second argument, plain, utilizing the key index stored in the handle specified in the first argument, handle, and writes the ongoing status to this first argument. In addition, it writes the encryption result to the third argument, cipher. After plaintext input is completed, call R_TSIP_Aes256EcbEncryptFinal().

Specify areas for plain and cipher that do not overlap. For plain and cipher, specify RAM addresses that are multiples of 4.

<State transition>

The pre-run state is *TSIP Enabled State*.

After the function runs the state transitions to *TSIP Enabled State*.

Refer to the Section 7 to generate key_index.

Reentrant

Not supported

4.24 R_TSIP_Aes256EcbEncryptFinal

Format

#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/output	AES handler (work area)
cipher	input/output	ciphertext data area (nothing ever written here)
cipher_length	input/output	ciphertext data length (0 always written here)

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_FAIL:	An internal error occurred.
TSIP_ERR_PARAMETER:	Input data is illegal.
TSIP_ERR_PROHIBIT_FUNCTION:	An invalid function was called.

Description

Using the handle specified in the first argument, handle, the R_TSIP_Aes256EcbEncryptFinal() function writes the calculation result to the second argument, cipher, and writes the length of the calculation result to the third argument, cipher_length. The original intent was for a portion of the encryption result that was not a multiple of 16 bytes to be written to the second argument. However, as a result of the restriction that only multiples of 16 can be input to the Update function, nothing is ever written to cipher, and 0 is always written to cipher_length. The arguments cipher and cipher_length are provided for compatibility in anticipation of the time when this restriction is lifted.

<State transition>

The pre-run state is *TSIP Enabled State*.

After the function runs the state transitions to *TSIP Enabled State*.

Refer to the Section 7 to generate key_index.

Reentrant

Not supported

4.25 R_TSIP_Aes256EcbDecryptInit

Format

#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	input/output	AES handler (work area)
key_index	input	user key index area

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_KEY_SET:	Input illegal user key index.
TSIP_ERR_RESOURCE_CONFLICT:	A resource conflict occurred because a hardware resource needed by the processing routine was in use by another processing routine.
TSIP_ERR_FAIL:	An internal error occurred.

Description

The R_TSIP_Aes256EcbDecryptInit() function performs preparations for the execution of an AES calculation, and writes the result to the first argument, handle. The value of handle is used as an argument in the subsequent R_TSIP_Aes256EcbDecryptUpdate() function and R_TSIP_Aes256EcbDecryptFinal() function.

<State transition>

The pre-run state is *TSIP Enabled State*.

After the function runs the state transitions to *TSIP Enabled State*.

Refer to the Section 7 to generate key_index.

Reentrant

Not supported

4.26 R_TSIP_Aes256EcbDecryptUpdate

Format

#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);

Parameters

handle	input/output	AES handler (work area)
cipher	input	ciphertext data area
plain	input/output	plaintext data area
cipher_length	input/output	ciphertext data length (must be a multiple of 16)

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_PARAMETER:	Input data is illegal.
TSIP_ERR_PROHIBIT_FUNCTION:	An invalid function was called.

Description

The R_TSIP_Aes256EcbDecryptUpdate() function decrypts the second argument, cipher, utilizing the key index stored in the handle specified in the first argument, handle, and writes the ongoing status to this first argument. In addition, it writes the decryption result to the third argument, plain. After ciphertext input is completed, call R_TSIP_Aes256EcbDecryptFinal().

Specify areas for plain and cipher that do not overlap. For plain and cipher, specify RAM addresses that are multiples of 4.

<State transition>

The pre-run state is *TSIP Enabled State*.

After the function runs the state transitions to *TSIP Enabled State*.

Refer to the Section 7 to generate key_index.

Reentrant

Not supported

4.27 R_TSIP_Aes256EcbDecryptFinal

Format

#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/output	AES handler (work area)
plain	input/output	plaintext data area (nothing ever written here)
plain_length	input/output	plaintext data length (0 always written here)

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_FAIL:	An internal error occurred.
TSIP_ERR_PARAMETER:	Input data is illegal.
TSIP_ERR_PROHIBIT_FUNCTION:	An invalid function was called.

Description

Using the handle specified in the first argument, handle, the R_TSIP_Aes256EcbDecryptFinal() function writes the calculation result to the second argument, plain, and writes the length of the calculation result to the third argument, plain_length. The original intent was for a portion of the decryption result that was not a multiple of 16 bytes to be written to the second argument. However, as a result of the restriction that only multiples of 16 can be input to the Update function, nothing is ever written to plain, and 0 is always written to plain_length. The arguments plain and plain_length are provided for compatibility in anticipation of the time when this restriction is lifted.

<State transition>

The pre-run state is *TSIP Enabled State*.

After the function runs the state transitions to *TSIP Enabled State*.

Refer to the Section 7 to generate key_index.

Reentrant

Not supported

4.28 R_TSIP_Aes128CbcEncryptInit

Format

#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);

Parameters

handle	input/output	AES handler (work area)
key_index	input	user key index area
ivec	input	initial vector area(16byte)

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_KEY_SET:	Input illegal user key index.
TSIP_ERR_RESOURCE_CONFLICT:	A resource conflict occurred because a hardware resource needed by the processing routine was in use by another processing routine.
TSIP_ERR_FAIL:	An internal error occurred.

Description

The R_TSIP_Aes128CbcEncryptInit() function performs preparations for the execution of an AES calculation, and writes the result to the first argument, handle. The value of handle is used as an argument in the subsequent R_TSIP_Aes128CbcEncryptUpdate() function and R_TSIP_Aes128CbcEncryptFinal() function.

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

<State transition>

The pre-run state is *TSIP Enabled State*.

After the function runs the state transitions to *TSIP Enabled State*.

Refer to the Section 7 to generate key_index.

Reentrant

Not supported

4.29 R_TSIP_Aes128CbcEncryptUpdate

Format

```
#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);
```

Parameters

handle	input/output	AES handler (work area)
plain	input	plaintext data area
cipher	input/output	ciphertext data area
plain_length	input/output	plaintext data length (must be a multiple of 16)

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_PARAMETER:	Input data is illegal.
TSIP_ERR_PROHIBIT_FUNCTION:	An invalid function was called.

Description

The R_TSIP_Aes128CbcEncryptUpdate() function encrypts the second argument, plain, utilizing the key index stored in the handle specified in the first argument, handle, and writes the ongoing status to this first argument. In addition, it writes the encryption result to the third argument, cipher. After plaintext input is completed, call R_TSIP_Aes128CbcEncryptFinal().

Specify areas for plain and cipher that do not overlap. For plain and cipher, specify RAM addresses that are multiples of 4.

<State transition>

The pre-run state is *TSIP Enabled State*.

After the function runs the state transitions to *TSIP Enabled State*.

Refer to the Section 7 to generate key_index.

Reentrant

Not supported

4.30 R_TSIP_Aes128CbcEncryptFinal

Format

```
#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);
```

Parameters

handle	input/output	AES handler (work area)
cipher	input/output	ciphertext data area (nothing ever written here)
cipher_length	input/output	ciphertext data length (0 always written here)

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_FAIL:	An internal error occurred.
TSIP_ERR_PARAMETER:	Input data is illegal.
TSIP_ERR_PROHIBIT_FUNCTION:	An invalid function was called.

Description

Using the handle specified in the first argument, handle, the R_TSIP_Aes128CbcEncryptFinal() function writes the calculation result to the second argument, cipher, and writes the length of the calculation result to the third argument, cipher_length. The original intent was for a portion of the encryption result that was not a multiple of 16 bytes to be written to the second argument. However, as a result of the restriction that only multiples of 16 can be input to the Update function, nothing is ever written to cipher, and 0 is always written to cipher_length. The arguments cipher and cipher_length are provided for compatibility in anticipation of the time when this restriction is lifted.

<State transition>

The pre-run state is *TSIP Enabled State*.

After the function runs the state transitions to *TSIP Enabled State*.

Refer to the Section 7 to generate key_index.

Reentrant

Not supported

4.31 R_TSIP_Aes128CbcDecryptInit

Format

#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);

Parameters

handle	input/output	AES handler (work area)
key_index	input	user key index area
ivec	input	initial vector area(16byte)

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_KEY_SET:	Input illegal user key index.
TSIP_ERR_RESOURCE_CONFLICT:	A resource conflict occurred because a hardware resource needed by the processing routine was in use by another processing routine.
TSIP_ERR_FAIL:	An internal error occurred.

Description

The R_TSIP_Aes128CbcDecryptInit() function performs preparations for the execution of an AES calculation, and writes the result to the first argument, handle. The value of handle is used as an argument in the subsequent R_TSIP_Aes128CbcDecryptUpdate() function and R_TSIP_Aes128CbcDecryptFinal() function.

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

<State transition>

The pre-run state is *TSIP Enabled State*.

After the function runs the state transitions to *TSIP Enabled State*.

Refer to the Section 7 to generate key_index.

Reentrant

Not supported

4.32 R_TSIP_Aes128CbcDecryptUpdate

Format

```
#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);
```

Parameters

handle	input/output	AES handler (work area)
cipher	input	ciphertext data area
plain	input/output	plaintext data area
cipher_length	input/output	ciphertext data length (must be a multiple of 16)

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_FAIL:	An internal error occurred.
TSIP_ERR_RESOURCE_CONFLICT:	A resource conflict occurred because a hardware resource needed by the processing routine was in use by another processing routine.
TSIP_ERR_PARAMETER:	Input data is illegal.
TSIP_ERR_PROHIBIT_FUNCTION:	An invalid function was called.

Description

The R_TSIP_Aes128CbcDecryptUpdate() function decrypts the second argument, cipher, utilizing the key index stored in the handle specified in the first argument, handle, and writes the ongoing status to this first argument. In addition, it writes the decryption result to the third argument, plain. After ciphertext input is completed, call R_TSIP_Aes128CbcDecryptFinal().

Specify areas for plain and cipher that do not overlap. For plain and cipher, specify RAM addresses that are multiples of 4.

<State transition>

The pre-run state is *TSIP Enabled State*.

After the function runs the state transitions to *TSIP Enabled State*.

Refer to the Section 7 to generate key_index.

Reentrant

Not supported

4.33 R_TSIP_Aes128CbcDecryptFinal

Format

#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);

Parameters

handle	input/output	AES handler (work area)
plain	input/output	plaintext data area (nothing ever written here)
plain_length	input/output	plaintext data length (0 always written here)

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_FAIL:	An internal error occurred.
TSIP_ERR_PARAMETER:	Input data is illegal.
TSIP_ERR_PROHIBIT_FUNCTION:	An invalid function was called.

Description

Using the handle specified in the first argument, handle, the R_TSIP_Aes128CbcDecryptFinal() function writes the calculation result to the second argument, plain, and writes the length of the calculation result to the third argument, plain_length. The original intent was for a portion of the decryption result that was not a multiple of 16 bytes to be written to the second argument. However, as a result of the restriction that only multiples of 16 can be input to the Update function, nothing is ever written to plain, and 0 is always written to plain_length. The arguments plain and plain_length are provided for compatibility in anticipation of the time when this restriction is lifted.

<State transition>

The pre-run state is *TSIP Enabled State*.

After the function runs the state transitions to *TSIP Enabled State*.

Refer to the Section 7 to generate key_index.

Reentrant

Not supported

4.34 R_TSIP_Aes256CbcEncryptInit

Format

#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	input/output	AES handler (work area)
key_index	input	user key index area
ivec	input	initial vector area(16byte)

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_KEY_SET:	Input illegal user key index.
TSIP_ERR_RESOURCE_CONFLICT:	A resource conflict occurred because a hardware resource needed by the processing routine was in use by another processing routine.
TSIP_ERR_FAIL:	An internal error occurred.

Description

The R_TSIP_Aes256CbcEncryptInit() function performs preparations for the execution of an AES calculation, and writes the result to the first argument, handle. The value of handle is used as an argument in the subsequent R_TSIP_Aes256CbcEncryptUpdate() function and R_TSIP_Aes256CbcEncryptFinal() function.

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

<State transition>

The pre-run state is *TSIP Enabled State*.

After the function runs the state transitions to *TSIP Enabled State*.

Refer to the Section 7 to generate key_index.

Reentrant

Not supported

4.35 R_TSIP_Aes256CbcEncryptUpdate

Format

#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	input/output	ciphertext data area
plain_length	input/output	plaintext data length (must be a multiple of 16)

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_PARAMETER:	Input data is illegal.
TSIP_ERR_PROHIBIT_FUNCTION:	An invalid function was called.

Description

The R_TSIP_Aes256CbcEncryptUpdate() function encrypts the second argument, plain, utilizing the key index stored in the handle specified in the first argument, handle, and writes the ongoing status to this first argument. In addition, it writes the encryption result to the third argument, cipher. After plaintext input is completed, call R_TSIP_Aes256CbcEncryptFinal().

Specify areas for plain and cipher that do not overlap. For plain and cipher, specify RAM addresses that are multiples of 4.

<State transition>

The pre-run state is *TSIP Enabled State*.

After the function runs the state transitions to *TSIP Enabled State*.

Refer to the Section 7 to generate key_index.

Reentrant

Not supported

4.36 R_TSIP_Aes256CbcEncryptFinal

Format

```
#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/output	AES handler (work area)
cipher	input/output	ciphertext data area (nothing ever written here)
cipher_length	input/output	ciphertext data length (0 always written here)

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_FAIL:	An internal error occurred.
TSIP_ERR_PARAMETER:	Input data is illegal.
TSIP_ERR_PROHIBIT_FUNCTION:	An invalid function was called.

Description

Using the handle specified in the first argument, handle, the R_TSIP_Aes256CbcEncryptFinal() function writes the calculation result to the second argument, cipher, and writes the length of the calculation result to the third argument, cipher_length. The original intent was for a portion of the encryption result that was not a multiple of 16 bytes to be written to the second argument. However, as a result of the restriction that only multiples of 16 can be input to the Update function, nothing is ever written to cipher, and 0 is always written to cipher_length. The arguments cipher and cipher_length are provided for compatibility in anticipation of the time when this restriction is lifted.

<State transition>

The pre-run state is *TSIP Enabled State*.

After the function runs the state transitions to *TSIP Enabled State*.

Refer to the Section 7 to generate key_index.

Reentrant

Not supported

4.37 R_TSIP_Aes256CbcDecryptInit

Format

```
#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	input/output	AES handler (work area)
key_index	input	user key index area
ivec	input	initial vector area(16byte)

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_KEY_SET:	Input illegal user key index.
TSIP_ERR_RESOURCE_CONFLICT:	A resource conflict occurred because a hardware resource needed by the processing routine was in use by another processing routine.
TSIP_ERR_FAIL:	An internal error occurred.

Description

The R_TSIP_Aes256CbcDecryptInit() function performs preparations for the execution of an AES calculation, and writes the result to the first argument, handle. The value of handle is used as an argument in the subsequent R_TSIP_Aes256CbcDecryptUpdate() function and R_TSIP_Aes256CbcDecryptFinal() function.

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

<State transition>

The pre-run state is *TSIP Enabled State*.

After the function runs the state transitions to *TSIP Enabled State*.

Refer to the Section 7 to generate key_index.

Reentrant

Not supported

4.38 R_TSIP_Aes256CbcDecryptUpdate

Format

#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);

Parameters

handle	input/output	AES handler (work area)
cipher	input	ciphertext data area
plain	input/output	plaintext data area
cipher_length	input/output	ciphertext data length (must be a multiple of 16)

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_PARAMETER:	Input data is illegal.
TSIP_ERR_PROHIBIT_FUNCTION:	An invalid function was called.

Description

The R_TSIP_Aes256CbcDecryptUpdate() function decrypts the second argument, cipher, utilizing the key index stored in the handle specified in the first argument, handle, and writes the ongoing status to this first argument. In addition, it writes the decryption result to the third argument, plain. After ciphertext input is completed, call R_TSIP_Aes256CbcDecryptFinal().

Specify areas for plain and cipher that do not overlap. For plain and cipher, specify RAM addresses that are multiples of 4.

<State transition>

The pre-run state is *TSIP Enabled State*.

After the function runs the state transitions to *TSIP Enabled State*.

Refer to the Section 7 to generate key_index.

Reentrant

Not supported

4.39 R_TSIP_Aes256CbcDecryptFinal

Format

```
#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/output	AES handler (work area)
plain	input/output	plaintext data area (nothing ever written here)
plain_length	input/output	plaintext data length (0 always written here)

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_FAIL:	An internal error occurred.
TSIP_ERR_PARAMETER:	Input data is illegal.
TSIP_ERR_PROHIBIT_FUNCTION:	An invalid function was called.

Description

Using the handle specified in the first argument, handle, the R_TSIP_Aes256CbcDecryptFinal() function writes the calculation result to the second argument, plain, and writes the length of the calculation result to the third argument, plain_length. The original intent was for a portion of the decryption result that was not a multiple of 16 bytes to be written to the second argument. However, as a result of the restriction that only multiples of 16 can be input to the Update function, nothing is ever written to plain, and 0 is always written to plain_length. The arguments plain and plain_length are provided for compatibility in anticipation of the time when this restriction is lifted.

<State transition>

The pre-run state is *TSIP Enabled State*.

After the function runs the state transitions to *TSIP Enabled State*.

Refer to the Section 7 to generate key_index.

Reentrant

Not supported

4.40 R_TSIP_Aes128GcmEncryptInit

Format

#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);

Parameters

handle	input/output	AES-GCM handler (work area)
key_index	input	user key index area
ivec	input	initialization vector area (iv_len byte) [note]
ivec_len	input	initialization vector length (1 or more bytes)

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_RESOURCE_CONFLICT:	A resource conflict occurred because a hardware resource needed by the processing routine was in use by another processing routine.
TSIP_ERR_KEY_SET:	Invalid user key index was input.
TSIP_ERR_PARAMETER:	Input data is illegal.
TSIP_ERR_FAIL:	An internal error occurred.

Description

The R_TSIP_Aes128GcmEncryptInit() function performs preparations for the execution of an GCM calculation, and writes the result to the first argument, handle. The value of handle is used as an argument in the subsequent R_TSIP_Aes128GcmEncryptUpdate() function and R_TSIP_Aes128GcmEncryptFinal() function. Moreover, please set 4-byte aligned RAM address to ivec.

[note]

When key_index->type is TSIP_KEY_INDEX_TYPE_AES128_FOR_TLS

The key_index value generated by the R_TSIP_TlsGenerateSessionKey() function when 6 or 7 is specified for select_cipher includes a 96-bit IV. Input a null pointer as the third argument, ivec. Specify 0 as the fourth argument, ivec_len.

<State transition>

The pre-run state is *TSIP Enabled State*.

After the function runs the state transitions to *TSIP Enabled State*.

Refer to the Section 7 to generate key_index.

Reentrant

Not supported

4.41 R_TSIP_Aes128GcmEncryptUpdate

Format

#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);

Parameters

handle	input/output	AES-GCM handler (work area)
plain	input	plaintext data area
cipher	input/output	ciphertext data area
plain_data_len	input	plaintext data length (0 or more bytes)
aad	input	additional authentication data (aad_len byte)
aad_len	input	additional authentication data length (0 or more bytes)

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_PARAMETER:	After the data from plain was input, an invalid handle was input from aad.
TSIP_ERR_PROHIBIT_FUNCTION:	An invalid function was called.

Description

The R_TSIP_Aes128GcmEncryptUpdate() function encrypts the plaintext specified in the second argument, plain, in GCM mode using the values specified for key_index and ivec in R_TSIP_Aes128GcmEncryptInit(), along with the additional authentication data specified in the fifth argument, aad. Inside this function, the data that is input by the user is buffered until the input values of aad and plain exceed 16 bytes. After the input data from plain reaches 16 bytes or more, the encryption result is output to the ciphertext data area specified in the third argument, cipher. The lengths of the plain and aad data to input are respectively specified in the fourth argument, plain_data_len, and the sixth argument, aad_len. For these, specify not the total byte count for the aad and plain input data, but rather the data length to input when the user calls this function. If the input values plain and aad are not divisible by 16 bytes, they will be padded inside the function. First process the data that is input from aad, and then process the data that is input from plain. If aad data is input after starting to input plain data, an error will occur. If aad data and plain data are input to this function at the same time, the aad data will be processed, and then the function will transition to the plain data input state. Specify areas for plain and cipher that do not overlap. For plain, cipher, and aad, specify RAM addresses that are multiples of 4.

<State transition>

The pre-run state is *TSIP Enabled State*.

After the function runs the state transitions to *TSIP Enabled State*.

Refer to the Section 7 to generate key_index.

Reentrant

Not supported

4.42 R_TSIP_Aes128GcmEncryptFinal

Format

#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);

Parameters

handle	input/output	AES-GCM handler (work area)
cipher	input/output	ciphertext data area (data_len byte)
cipher_data_len	input/output	ciphertext data length (0 or more bytes)
atag	input/output	authentication tag area

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_FAIL:	An internal error occurred.
TSIP_ERR_PARAMETER:	An invalid handle was input.
TSIP_ERR_PROHIBIT_FUNCTION:	An invalid function was called.

Description

If there is 16-byte fractional data indicated by the total data length of the value of plain that was input by R_TSIP_Aes128GcmEncryptUpdate (), the R_TSIP_Aes128GcmEncryptFinal() function will output the result of encrypting that fractional data to the ciphertext data area specified in the second argument, cipher. Here, the portion that does not reach 16 bytes will be padded with zeros. The authentication tag is output to the fourth argument, atag. For cipher and atag, specify RAM addresses that are multiples of 4.

<State transition>

The pre-run state is *TSIP Enabled State*.

After the function runs the state transitions to *TSIP Enabled State*.

Refer to the Section 7 to generate key_index.

Reentrant

Not supported

4.43 R_TSIP_Aes128GcmDecryptInit

Format

```
#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);
```

Parameters

handle	input/output	AES-GCM handler (work area)
key_index	input	user key index area
ivec	input	initialization vector area (iv_len byte) [note]
ivec_len	input	initialization vector length (1 or more bytes)

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_RESOURCE_CONFLICT:	A resource conflict occurred because a hardware resource needed by the processing routine was in use by another processing routine.
TSIP_ERR_KEY_SET:	Invalid user key index was input.
TSIP_ERR_PARAMETER:	Input data is illegal.
TSIP_ERR_FAIL:	An internal error occurred.

Description

The R_TSIP_Aes128GcmDecryptInit() function performs preparations for the execution of an GCM calculation, and writes the result to the first argument, handle. The value of handle is used as an argument in the subsequent R_TSIP_Aes128GcmDecryptUpdate() function and R_TSIP_Aes128GcmDecryptFinal() function. Moreover, please set 4-byte aligned RAM address to ivec.

[note]

When key_index->type is TSIP_KEY_INDEX_TYPE_AES128_FOR_TLS.

The key_index value generated by the R_TSIP_TlsGenerateSessionKey() function when 6 or 7 is specified for select_cipher includes a 96-bit IV. Input a null pointer as the third argument, ivec. Specify 0 as the fourth argument, ivec_len.

<State transition>

The pre-run state is *TSIP Enabled State*.

After the function runs the state transitions to *TSIP Enabled State*.

Refer to the Section 7 to generate key_index.

Reentrant

Not supported

4.44 R_TSIP_Aes128GcmDecryptUpdate

Format

```
#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);
```

Parameters

handle	input/output	AES-GCM handler (work area)
cipher	input	ciphertext data area
plain	input/output	plaintext data area
cipher_data_len	input	ciphertext data length (0 or more bytes)
aad	input	additional authentication data (aad_len byte)
aad_len	input	additional authentication data length (0 or more bytes)

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_PARAMETER:	After the data from plain was input, an invalid handle was input from aad.
TSIP_ERR_PROHIBIT_FUNCTION:	An invalid function was called.

Description

The R_TSIP_Aes128GcmDecryptUpdate() function decrypts the ciphertext specified in the second argument, cipher, in GCM mode using the values specified for key_index and ivec in R_TSIP_Aes128GcmDecryptInit(), along with the additional authentication data specified in the fifth argument, aad. Inside this function, the data that is input by the user is buffered until the input values of aad and plain exceed 16 bytes. After the input data from cipher reaches 16 bytes or more, the decryption result is output to the plaintext data area specified in the third argument, plain. The lengths of the cipher and aad data to input are respectively specified in the fourth argument, cipher_data_len, and the sixth argument, aad_len. For these, specify not the total byte count for the aad and cipher input data, but rather the data length to input when the user calls this function. If the input values cipher and aad are not divisible by 16 bytes, they will be padded inside the function. First process the data that is input from aad, and then process the data that is input from cipher. If aad data is input after starting to input cipher data, an error will occur. If aad data and cipher data are input to this function at the same time, the aad data will be processed, and then the function will transition to the cipher data input state. Specify areas for plain and cipher that do not overlap. For plain, cipher, and aad, specify RAM addresses that are multiples of 4.

<State transition>

The pre-run state is *TSIP Enabled State*.

After the function runs the state transitions to *TSIP Enabled State*.

Refer to the Section 7 to generate key_index.

Reentrant

Not supported

4.45 R_TSIP_Aes128GcmDecryptFinal

Format

#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,

uint8_t atag_len);

Parameters

handle	input/output	AES-GCM handler (work area)
plain	input/output	plaintext data area (data_len byte)
plain_data_len	input/output	plaintext data length (0 or more bytes)
atag	input/output	authentication tag area (atag_len byte)
atag_len	input	authentication tag length (4,8,12,13,14,15,16byte)

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_FAIL:	An internal error occurred.
TSIP_ERR_AUTHENTICATION:	Authentication failed
TSIP_ERR_PARAMETER:	Input data is illegal..
TSIP_ERR_PROHIBIT_FUNCTION:	An invalid function was called.

Description

The R_TSIP_Aes128GcmDecryptFinal() function decrypts, in GCM mode, the fractional ciphertext specified by R_TSIP_Aes128GcmDecryptUpdate() that does not reach 16 bytes, and ends GCM decryption. The encryption data and authentication tag are respectively output to the plaintext data area specified in the second argument, plain, and the authentication tag area specified in the fourth argument, atag. The decoded data length is output to the third argument, plain_data_len. If authentication fails, the return value will be TSIP_ERR_AUTHENTICATION. For the fourth argument, atag, input 16 bytes or less. If it is less than 16 bytes, it will be padded with zeros inside the function. For plain and atag, specify RAM addresses that are multiples of 4.

<State transition>

The pre-run state is *TSIP Enabled State*.

After the function runs the state transitions to *TSIP Enabled State*.

Refer to the Section 7 to generate key_index.

Reentrant

Not supported

4.46 R_TSIP_Aes256GcmEncryptInit

Format

#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	input/output	AES-GCM handler (work area)
key_index	input	user key index area
ivec	input	initialization vector area (iv_len byte)
ivec_len	input	initialization vector length (1 or more bytes)

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_RESOURCE_CONFLICT:	A resource conflict occurred because a hardware resource needed by the processing routine was in use by another processing routine.
TSIP_ERR_KEY_SET:	Invalid user key index was input.
TSIP_ERR_PARAMETER:	Input data is illegal.
TSIP_ERR_FAIL:	An internal error occurred.

Description

The R_TSIP_Aes256GcmEncryptInit() function performs preparations for the execution of an GCM calculation, and writes the result to the first argument, handle. The value of handle is used as an argument in the subsequent R_TSIP_Aes256GcmEncryptUpdate() function and R_TSIP_Aes256GcmEncryptFinal() function. Moreover, please set 4-byte aligned RAM address to ivec.

<State transition>

The pre-run state is *TSIP Enabled State*.

After the function runs the state transitions to *TSIP Enabled State*.

Refer to the Section 7 to generate key_index.

Reentrant

Not supported

4.47 R_TSIP_Aes256GcmEncryptUpdate

Format

```
#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-GCM handler (work area)
plain	input	plaintext data area
cipher	input/output	ciphertext data area
plain_data_len	input	plaintext data length (0 or more bytes)
aad	input	additional authentication data (aad_len byte)
aad_len	input	additional authentication data length (0 or more bytes)

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_PARAMETER:	After the data from plain was input, an invalid handle was input from aad.
TSIP_ERR_PROHIBIT_FUNCTION:	An invalid function was called.

Description

The R_TSIP_Aes256GcmEncryptUpdate() function encrypts the plaintext specified in the second argument, plain, in GCM mode using the values specified for key_index and ivec in R_TSIP_Aes256GcmEncryptInit(), along with the additional authentication data specified in the fifth argument, aad. Inside this function, the data that is input by the user is buffered until the input values of aad and plain exceed 16 bytes. After the input data from plain reaches 16 bytes or more, the encryption result is output to the ciphertext data area specified in the third argument, cipher. The lengths of the plain and aad data to input are respectively specified in the fourth argument, plain_data_len, and the sixth argument, aad_len. For these, specify not the total byte count for the aad and plain input data, but rather the data length to input when the user calls this function. If the input values plain and aad are not divisible by 16 bytes, they will be padded inside the function. First process the data that is input from aad, and then process the data that is input from plain. If aad data is input after starting to input plain data, an error will occur. If aad data and plain data are input to this function at the same time, the aad data will be processed, and then the function will transition to the plain data input state. Specify areas for plain and cipher that do not overlap. For plain, cipher, and aad, specify RAM addresses that are multiples of 4.

<State transition>

The pre-run state is *TSIP Enabled State*.

After the function runs the state transitions to *TSIP Enabled State*.

Refer to the Section 7 to generate key_index.

Reentrant

Not supported

4.48 R_TSIP_Aes256GcmEncryptFinal

Format

#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/output	AES-GCM handler (work area)
cipher	input/output	ciphertext data area (data_len byte)
cipher_data_len	input/output	ciphertext data length (0 or more bytes)
atag	input/output	authentication tag area

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_FAIL:	An internal error occurred.
TSIP_ERR_PARAMETER:	An invalid handle was input.
TSIP_ERR_PROHIBIT_FUNCTION:	An invalid function was called.

Description

If there is 16-byte fractional data indicated by the total data length of the value of plain that was input by R_TSIP_Aes256GcmEncryptUpdate (), the R_TSIP_Aes256GcmEncryptFinal() function will output the result of encrypting that fractional data to the ciphertext data area specified in the second argument, cipher. Here, the portion that does not reach 16 bytes will be padded with zeros. The authentication tag is output to the fourth argument, atag. For cipher and atag, specify RAM addresses that are multiples of 4.

<State transition>

The pre-run state is *TSIP Enabled State*.

After the function runs the state transitions to *TSIP Enabled State*.

Refer to the Section 7 to generate key_index.

Reentrant

Not supported

4.49 R_TSIP_Aes256GcmDecryptInit

Format

```
#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	input/output	AES-GCM handler (work area)
key_index	input	user key index area
ivec	input	initialization vector area (iv_len byte)
ivec_len	input	initialization vector length (1 or more bytes)

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_RESOURCE_CONFLICT:	A resource conflict occurred because a hardware resource needed by the processing routine was in use by another processing routine.
TSIP_ERR_KEY_SET:	Invalid user key index was input.
TSIP_ERR_PARAMETER:	Input data is illegal.
TSIP_ERR_FAIL:	An internal error occurred.

Description

The R_TSIP_Aes256GcmDecryptInit() function performs preparations for the execution of an GCM calculation, and writes the result to the first argument, handle. The value of handle is used as an argument in the subsequent R_TSIP_Aes256GcmDecryptUpdate() function and R_TSIP_Aes256GcmDecryptFinal() function. Moreover, please set 4-byte aligned RAM address to ivec.

<State transition>

The pre-run state is *TSIP Enabled State*.

After the function runs the state transitions to *TSIP Enabled State*.

Refer to the Section 7 to generate key_index.

Reentrant

Not supported

4.50 R_TSIP_Aes256GcmDecryptUpdate

Format

```
#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	input/output	plaintext data area
cipher_data_len	input	ciphertext data length (0 or more bytes)
aad	input	additional authentication data (aad_len byte)
aad_len	input	additional authentication data length (0 or more bytes)

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_PARAMETER:	After the data from plain was input, an invalid handle was input from aad.
TSIP_ERR_PROHIBIT_FUNCTION:	An invalid function was called.

Description

The R_TSIP_Aes256GcmDecryptUpdate() function decrypts the ciphertext specified in the second argument, cipher, in GCM mode using the values specified for key_index and ivec in R_TSIP_Aes256GcmDecryptInit(), along with the additional authentication data specified in the fifth argument, aad. Inside this function, the data that is input by the user is buffered until the input values of aad and plain exceed 16 bytes. After the input data from cipher reaches 16 bytes or more, the decryption result is output to the plaintext data area specified in the third argument, plain. The lengths of the cipher and aad data to input are respectively specified in the fourth argument, cipher_data_len, and the sixth argument, aad_len. For these, specify not the total byte count for the aad and cipher input data, but rather the data length to input when the user calls this function. If the input values cipher and aad are not divisible by 16 bytes, they will be padded inside the function. First process the data that is input from aad, and then process the data that is input from cipher. If aad data is input after starting to input cipher data, an error will occur. If aad data and cipher data are input to this function at the same time, the aad data will be processed, and then the function will transition to the cipher data input state. Specify areas for plain and cipher that do not overlap. For plain, cipher, and aad, specify RAM addresses that are multiples of 4.

<State transition>

The pre-run state is *TSIP Enabled State*.

After the function runs the state transitions to *TSIP Enabled State*.

Refer to the Section 7 to generate key_index.

Reentrant

Not supported

4.51 R_TSIP_Aes256GcmDecryptFinal

Format

```
#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,
         uint8_t atag_len);
```

Parameters

handle	input/output	AES-GCM handler (work area)
plain	input/output	plaintext data area (data_len byte)
plain_data_len	input/output	plaintext data length (0 or more bytes)
atag	input/output	authentication tag area (atag_len byte)
atag_len	input	authentication tag length (4,8,12,13,14,15,16byte)

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_FAIL:	An internal error occurred.
TSIP_ERR_AUTHENTICATION:	Authentication failed
TSIP_ERR_PARAMETER:	Input data is illegal .
TSIP_ERR_PROHIBIT_FUNCTION:	An invalid function was called.

Description

The R_TSIP_Aes256GcmDecryptFinal() function decrypts, in GCM mode, the fractional ciphertext specified by R_TSIP_Aes256GcmDecryptUpdate() that does not reach 16 bytes, and ends GCM decryption. The encryption data and authentication tag are respectively output to the plaintext data area specified in the second argument, plain, and the authentication tag area specified in the fourth argument, atag. The decoded data length is output to the third argument, plain_data_len. If authentication fails, the return value will be TSIP_ERR_AUTHENTICATION. For the fourth argument, atag, input 16 bytes or less. If it is less than 16 bytes, it will be padded with zeros inside the function. For plain and atag, specify RAM addresses that are multiples of 4.

<State transition>

The pre-run state is *TSIP Enabled State*.

After the function runs the state transitions to *TSIP Enabled State*.

Refer to the Section 7 to generate key_index.

Reentrant

Not supported

4.52 R_TSIP_Aes128CcmEncryptInit

Format

```
#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
)
```

Parameters

handle	input/output	AES-CCM handler (work area)
key_index	input	user key index area
nonce	input	Nonce
nonce_len	input	Nonce data length (7 to 13 bytes)
adata	input	additional authentication data
a_len	input	additional authentication data 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_KEY_SET:	Invalid user key index was input.
TSIP_ERR_RESOURCE_CONFLICT:	A resource conflict occurred because a hardware resource needed by the processing routine was in use by another processing routine.
TSIP_ERR_FAIL:	An internal error occurred.

Description

The R_TSIP_Aes128CcmEncryptInit() function prepares to perform CCM computation and writes the result to the first argument, handle. The succeeding functions R_TSIP_Aes128CcmEncryptUpdate() and R_TSIP_Aes128CcmEncryptFinal() use handle as an argument.

<State transition>

The pre-run state is *TSIP Enabled State*.

After the function runs the state transitions to *TSIP Enabled State*.

Refer to the Section 7 to generate key_index.

Reentrant

Not supported

4.53 R_TSIP_Aes128CcmEncryptUpdate

Format

```
#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
)
```

Parameters

handle	input/output	AES-CCM handler (work area)
plain	input	plaintext data area
cipher	input/output	ciphertext data area
plain_length	input	plaintext data length

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_PROHIBIT_FUNCTION:	An invalid function was called.
TSIP_ERR_PARAMETER:	An invalid handle was input.

Description

The R_TSIP_Aes128CcmEncryptUpdate() 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_Aes128CcmEncryptInit(). 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_Aes128CcmEncryptInit() 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.

Ensure that the areas allocated to plain and cipher do not overlap. Also, specify RAM addresses that are multiples of 4 for plain and cipher.

<State transition>

The pre-run state is *TSIP Enabled State*.

After the function runs the state transitions to *TSIP Enabled State*.

Reentrant

Not supported

4.54 R_TSIP_Aes128CcmEncryptFinal

Format

```
#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
)
```

Parameters

handle	input/output	AES-CCM handler (work area)
cipher	input/output	ciphertext data area
cipher_length	input/output	ciphertext data length
mac	input/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_PROHIBIT_FUNCTION:	An invalid function was called.
TSIP_ERR_PARAMETER:	Input data is illegal
TSIP_ERR_FAIL:	An internal error occurred.

Description

If the data length of plain input in R_TSIP_Aes128CcmEncryptUpdate() results in leftover data after 16 bytes, the R_TSIP_Aes128CcmEncryptFinal() function outputs the leftover encrypted data to cipher, which is specified in the second argument. The MAC value is output to the fourth argument, mac. Set the fifth argument, mac_length, to the same value as that specified for the argument mac_len in Aes128CcmEncryptInit(). Also, specify RAM addresses that are multiples of 4 for cipher and mac.

<State transition>

The pre-run state is *TSIP Enabled State*.

After the function runs the state transitions to *TSIP Enabled State*.

Reentrant

Not supported

4.55 R_TSIP_Aes128CcmDecryptInit

Format

```
#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/output	AES-CCM handler (work area)
key_index	input	user key index area
nonce	input	Nonce
nonce_len	input	Nonce data length (7 to 13 bytes)
adata	input	additional authentication data
a_len	input	additional authentication data 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_KEY_SET:	Invalid user key index was input.
TSIP_ERR_RESOURCE_CONFLICT:	A resource conflict occurred because a hardware resource needed by the processing routine was in use by another processing routine.
TSIP_ERR_FAIL:	An internal error occurred.

Description

The R_TSIP_Aes128CcmDecryptInit() function prepares to perform CCM computation and writes the result to the first argument, handle. The succeeding functions R_TSIP_Aes128CcmDecryptUpdate() and R_TSIP_Aes128CcmDecryptFinal() use handle as an argument.

<State transition>

The pre-run state is *TSIP Enabled State*.

After the function runs the state transitions to *TSIP Enabled State*.

Reentrant

Not supported

4.56 R_TSIP_Aes128CcmDecryptUpdate

Format

```
#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
)
```

Parameters

handle	input/output	AES-CCM handler (work area)
cipher	input	plaintext data area
plain	input/output	ciphertext data area
cipher_length	input	ciphertext data length

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_PROHIBIT_FUNCTION:	An invalid function was called.
TSIP_ERR_PARAMETER:	An invalid handle was input.

Description

The R_TSIP_Aes128CcmDecryptUpdate() function decrypts the ciphertext specified by the second argument, cipher, in CCM mode using the values specified by key_index, nonce, and adata in in R_TSIP_Aes128CcmDecryptInit(). This function buffers internally the data input by the user until the input value of cipher exceeds 16 bytes. Once the amount of cipher input data is 16 bytes or greater, the decrypted result is output to plain, which is specified in the third argument. Use payload_len in R_TSIP_Aes128CcmDecryptInit() to specify the total data length of cipher that will be input. Use cipher_length in this function to specify the data length to be input when the user calls this function. If the input value of cipher is less than 16 bytes, the function performs padding internally.

Ensure that the areas allocated to cipher and plain do not overlap. Also, specify RAM addresses that are multiples of 4 for cipher and plain.

<State transition>

The pre-run state is *TSIP Enabled State*.

After the function runs the state transitions to *TSIP Enabled State*.

Reentrant

Not supported

4.57 R_TSIP_Aes128CcmDecryptFinal

Format

```
#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
)
```

Parameters

handle	input/output	AES-CCM handler (work area)
plain	input/output	plaintext data area
plain_length	input/output	plaintext data length
mac	input	MAC area
mac_length	input	MAC length (4, 6, 8, 10, 12, 14, or 16 bytes)

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_PROHIBIT_FUNCTION:	An invalid function was called.
TSIP_ERR_PARAMETER:	Input data is illegal
TSIP_ERR_FAIL	Internal error, or authentication failed.

Description

If the data length of cipher input in R_TSIP_Aes128CcmDecryptUpdate() results in leftover data after 16 bytes, the R_TSIP_Aes128CcmDecryptFinal() function outputs the leftover decrypted data to cipher, which is specified in the second argument. In addition, the function verifies the fourth argument, mac. Set the fifth argument, mac_length, to the same value as that specified for the argument mac_len in Aes128CcmDecryptInit().

<State transition>

The pre-run state is *TSIP Enabled State*.

After the function runs the state transitions to *TSIP Enabled State*.

Reentrant

Not supported

4.58 R_TSIP_Aes256CcmEncryptInit

Format

```
#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	input/output	AES-CCM handler (work area)
key_index	input	user key index area
nonce	input	Nonce
nonce_len	input	Nonce data length (7 to 13 bytes)
adata	input	additional authentication data
a_len	input	additional authentication data 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_KEY_SET:	Invalid user key index was input.
TSIP_ERR_RESOURCE_CONFLICT:	A resource conflict occurred because a hardware resource needed by the processing routine was in use by another processing routine.
TSIP_ERR_FAIL:	An internal error occurred.

Description

The R_TSIP_Aes256CcmEncryptInit() function prepares to perform CCM computation and writes the result to the first argument, handle. The succeeding functions R_TSIP_Aes256CcmEncryptUpdate() and R_TSIP_Aes256CcmEncryptFinal() use handle as an argument.

<State transition>

The pre-run state is *TSIP Enabled State*.

After the function runs the state transitions to *TSIP Enabled State*.

Refer to the Section 7 to generate key_index.

Reentrant

Not supported

4.59 R_TSIP_Aes256CcmEncryptUpdate

Format

```
#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-CCM handler (work area)
plain	input	plaintext data area
cipher	input/output	ciphertext data area
plain_length	input	plaintext data length

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_PROHIBIT_FUNCTION:	An invalid function was called.
TSIP_ERR_PARAMETER:	An invalid handle was input.

Description

The R_TSIP_Aes256CcmEncryptUpdate() 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_Aes256CcmEncryptInit(). 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_Aes256CcmEncryptInit() 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

Ensure that the areas allocated to plain and cipher do not overlap. Also, specify RAM addresses that are multiples of 4 for plain and cipher.

<State transition>

The pre-run state is *TSIP Enabled State*.

After the function runs the state transitions to *TSIP Enabled State*.

Reentrant

Not supported

4.60 R_TSIP_Aes256CcmEncryptFinal

Format

```
#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/output	AES-CCM handler (work area)
cipher	input/output	ciphertext data area
cipher_length	input/output	ciphertext data length
mac	input/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_PROHIBIT_FUNCTION:	An invalid function was called.
TSIP_ERR_PARAMETER:	Input data is illegal .
TSIP_ERR_FAIL:	An internal error occurred.

Description

If the data length of plain input in R_TSIP_Aes256CcmEncryptUpdate() results in leftover data after 16 bytes, the R_TSIP_Aes256CcmEncryptFinal() function outputs the leftover encrypted data to cipher, which is specified in the second argument. The MAC value is output to the fourth argument, mac. Set the fifth argument, mac_length, to the same value as that specified for the argument mac_len in Aes256CcmEncryptInit(). Also, specify RAM addresses that are multiples of 4 for cipher and mac.

<State transition>

The pre-run state is *TSIP Enabled State*.

After the function runs the state transitions to *TSIP Enabled State*.

Reentrant

Not supported

4.61 R_TSIP_Aes256CcmDecryptInit

Format

```
#include "r_tsip_rx_if.h"
e_tsip_err_t R_TSIP_Aes256CcmDecryptInit(
        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/output	AES-CCM handler (work area)
key_index	input	user key index area
nonce	input	Nonce
nonce_len	input	Nonce data length (7 to 13 bytes)
adata	input	additional authentication data
a_len	input	additional authentication data 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_KEY_SET:	Invalid user key index was input.
TSIP_ERR_RESOURCE_CONFLICT:	A resource conflict occurred because a hardware resource needed by the processing routine was in use by another processing routine.
TSIP_ERR_FAIL:	An internal error occurred.

Description

The R_TSIP_Aes256CcmDecryptInit() function prepares to perform CCM computation and writes the result to the first argument, handle. The succeeding functions R_TSIP_Aes256CcmDecryptUpdate() and R_TSIP_Aes256CcmDecryptFinal() use handle as an argument.

<State transition>

The pre-run state is *TSIP Enabled State*.

After the function runs the state transitions to *TSIP Enabled State*.

Reentrant

Not supported

4.62 R_TSIP_Aes256CcmDecryptUpdate

Format

```
#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	plaintext data area
plain	input/output	ciphertext data area
cipher_length	input	ciphertext data length

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_PROHIBIT_FUNCTION:	An invalid function was called.
TSIP_ERR_PARAMETER:	An invalid handle was input.

Description

The R_TSIP_Aes256CcmDecryptUpdate() function decrypts the ciphertext specified by the second argument, cipher, in CCM mode using the values specified by key_index, nonce, and adata in in R_TSIP_Aes256CcmDecryptInit(). This function buffers internally the data input by the user until the input value of cipher exceeds 16 bytes. Once the amount of cipher input data is 16 bytes or greater, the decrypted result is output to plain, which is specified in the third argument. Use payload_len in R_TSIP_Aes256CcmDecryptInit() to specify the total data length of cipher that will be input. Use cipher_length in this function to specify the data length to be input when the user calls this function. If the input value of cipher is less than 16 bytes, the function performs padding internally.

Ensure that the areas allocated to cipher and plain do not overlap. Also, specify RAM addresses that are multiples of 4 for cipher and plain.

<State transition>

The pre-run state is *TSIP Enabled State*.

After the function runs the state transitions to *TSIP Enabled State*.

Reentrant

Not supported

4.63 R_TSIP_Aes256CcmDecryptFinal

Format

```
#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/output	AES-CCM handler (work area)
plain	input/output	plaintext data area
plain_length	input/output	plaintext data length
mac	input	MAC area
mac_length	input	MAC length (4, 6, 8, 10, 12, 14, or 16 bytes)

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_PROHIBIT_FUNCTION:	An invalid function was called.
TSIP_ERR_PARAMETER:	Input data is illegal
TSIP_ERR_FAIL:	Internal error, or authentication failed.

Description

If the data length of cipher input in R_TSIP_Aes256CcmDecryptUpdate() results in leftover data after 16 bytes, the R_TSIP_Aes256CcmDecryptFinal() function outputs the leftover decrypted data to cipher, which is specified in the second argument. In addition, the function verifies the fourth argument, mac. Set the fifth argument, mac_length, to the same value as that specified for the argument mac_len in Aes256CcmDecryptInit().

<State transition>

The pre-run state is *TSIP Enabled State*.

After the function runs the state transitions to *TSIP Enabled State*.

Reentrant

Not supported

4.64 R_TSIP_Aes128CmacGenerateInit

Format

```
#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);
```

Parameters

handle	input/output	AES-CMAC handler (work area)
key_index	input	user key index area

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_RESOURCE_CONFLICT:	A resource conflict occurred because a hardware resource needed by the processing routine was in use by another processing routine.
TSIP_ERR_KEY_SET:	Invalid user key index was input.
TSIP_ERR_FAIL:	An internal error occurred.

Description

The R_TSIP_Aes128CmacGenerateInit() function performs preparations for the execution of an CMAC calculation, and writes the result to the first argument, handle. The value of handle is used as an argument in the subsequent R_TSIP_Aes128CmacGenerateUpdate() function and R_TSIP_Aes128CmacGenerateFinal() function.

<State transition>

The pre-run state is *TSIP Enabled State*.

After the function runs the state transitions to *TSIP Enabled State*.

Refer to the Section 7 to generate key_index.

Reentrant

Not supported

4.65 R_TSIP_Aes128CmacGenerateUpdate

Format

#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);

Parameters

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

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_PARAMETER:	An invalid handle was input.
TSIP_ERR_PROHIBIT_FUNCTION:	An invalid function was called.

Description

The R_TSIP_Aes128CmacGenerateUpdate() function performs MAC value generation based on the message specified in the second argument, message, using the value specified for key_index in R_TSIP_Aes128CmacGenerateInit(). Inside this function, the data that is input by the user is buffered until the input value of message exceeds 16 bytes. The length of the message data to input is specified in the third argument, message_len. For these, input not the total byte count for message input data, but rather the message data length to input when the user calls this function. If the input value, message, is not a multiple of 16 bytes, it will be padded within the function. For message, specify a RAM address that are multiples of 4.

<State transition>

The pre-run state is *TSIP Enabled State*.

After the function runs the state transitions to *TSIP Enabled State*.

Refer to the Section 7 to generate key_index.

Reentrant

Not supported

4.66 R_TSIP_Aes128CmacGenerateFinal

Format

#include "r_tsip_rx_if.h"

e_tsip_err_t R_TSIP_Aes128CmacGenerateFinal

(tsip_cmac_handle_t *handle, uint8_t *mac);

Parameters

handle	input/output	AES-CMAC handler (work area)
mac	input/output	MAC data area (16byte)

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_FAIL:	An internal error occurred.
TSIP_ERR_PARAMETER:	An invalid handle was input.
TSIP_ERR_PROHIBIT_FUNCTION:	An invalid function was called.

Description

The R_TSIP_Aes128CmacGenerateFinal() function outputs the MAC value to the MAC data area specified in the second argument, mac, and ends CMAC mode.

<State transition>

The pre-run state is *TSIP Enabled State*.

After the function runs the state transitions to *TSIP Enabled State*.

Reentrant

Not supported

4.67 R_TSIP_Aes256CmacGenerateInit

Format

```
#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	input/output	AES-CMAC handler (work area)
key_index	input	user key index area

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_RESOURCE_CONFLICT:	A resource conflict occurred because a hardware resource needed by the processing routine was in use by another processing routine.
TSIP_ERR_KEY_SET:	Invalid user key index was input.
TSIP_ERR_FAIL:	An internal error occurred.

Description

The R_TSIP_Aes256CmacGenerateInit() function performs preparations for the execution of a CMAC calculation, and writes the result to the first argument, handle. The value of handle is used as an argument in the subsequent R_TSIP_Aes256CmacGenerateUpdate() function and R_TSIP_Aes256CmacGenerateFinal() function.

<State transition>

The pre-run state is *TSIP Enabled State*.

After the function runs the state transitions to *TSIP Enabled State*.

Refer to the Section 7 to generate key_index.

Reentrant

Not supported

4.68 R_TSIP_Aes256CmacGenerateUpdate

Format

#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 byte)
message_length	input	message data length (0 or more bytes)

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_PARAMETER:	An invalid handle was input.
TSIP_ERR_PROHIBIT_FUNCTION:	An invalid function was called.

Description

The R_TSIP_Aes256CmacGenerateUpdate() function performs MAC value generation based on the message specified in the second argument, message, using the value specified for key_index in R_TSIP_Aes256CmacGenerateInit(). Inside this function, the data that is input by the user is buffered until the input value of message exceeds 16 bytes. The length of the message data to input is specified in the third argument, message_len. For these, input not the total byte count for message input data, but rather the message data length to input when the user calls this function. If the input value, message, is not a multiple of 16 bytes, it will be padded within the function. For message, specify a RAM address that are multiples of 4.

<State transition>

The pre-run state is *TSIP Enabled State*.

After the function runs the state transitions to *TSIP Enabled State*.

Reentrant

Not supported

4.69 R_TSIP_Aes256CmacGenerateFinal

Format

```
#include "r_tsip_rx_if.h"
e_tsip_err_t R_TSIP_Aes256CmacGenerateFinal
        (tsip_cmac_handle_t *handle, uint8_t *mac);
```

Parameters

handle	input/output	AES-CMAC handler (work area)
mac	input/output	MAC data area (16byte)

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_FAIL:	An internal error occurred.
TSIP_ERR_PARAMETER:	An invalid handle was input.
TSIP_ERR_PROHIBIT_FUNCTION:	An invalid function was called.

Description

The R_TSIP_Aes256CmacGenerateFinal() function outputs the MAC value to the MAC data area specified in the second argument, mac, and ends CMAC mode.

<State transition>

The pre-run state is *TSIP Enabled State*.

After the function runs the state transitions to *TSIP Enabled State*.

Reentrant

Not supported

4.70 R_TSIP_Aes128CmacVerifyInit

Format

```
#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);
```

Parameters

handle	input/output	AES-CMAC handler (work area)
key_index	input	user key index area

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_RESOURCE_CONFLICT:	A resource conflict occurred because a hardware resource needed by the processing routine was in use by another processing routine.
TSIP_ERR_KEY_SET:	Invalid user key index was input.
TSIP_ERR_FAIL:	An internal error occurred.

Description

The R_TSIP_Aes128CmacVerifyInit() function performs preparations for the execution of a CMAC calculation, and writes the result to the first argument, handle. The value of handle is used as an argument in the subsequent R_TSIP_Aes128CmacVerifyUpdate() function and R_TSIP_Aes128CmacVerifyFinal() function.

<State transition>

The pre-run state is *TSIP Enabled State*.

After the function runs the state transitions to *TSIP Enabled State*.

Refer to the Section 7 to generate key_index.

Reentrant

Not supported

4.71 R_TSIP_Aes128CmacVerifyUpdate

Format

#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);

Parameters

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

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_PARAMETER:	An invalid handle was input.
TSIP_ERR_PROHIBIT_FUNCTION:	An invalid function was called.

Description

The R_TSIP_Aes128CmacVerifyUpdate() function performs MAC value generation based on the message specified in the second argument, message, using the value specified for key_index in R_TSIP_Aes128CmacGenerateInit(). Inside this function, the data that is input by the user is buffered until the input value of message exceeds 16 bytes. The length of the message data to input is specified in the third argument, message_len. For these, input not the total byte count for message input data, but rather the message data length to input when the user calls this function. If the input value, message, is not a multiple of 16 bytes, it will be padded within the function. For message, specify a RAM address that are multiples of 4.

<State transition>

The pre-run state is *TSIP Enabled State*.

After the function runs the state transitions to *TSIP Enabled State*.

Reentrant

Not supported

4.72 R_TSIP_Aes128CmacVerifyFinal

Format

#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);

Parameters

handle	input/output	AES-CMAC handler (work area)
mac	input/output	MAC data area (mac_length byte)
mac_length	input/output	MAC data length (2 to 16 bytes)

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_FAIL:	An internal error occurred.
TSIP_ERR_AUTHENTICATION:	Authentication failed
TSIP_ERR_PARAMETER:	Input data is illegal .
TSIP_ERR_PROHIBIT_FUNCTION:	An invalid function was called.

Description

The R_TSIP_Aes128CmacVerifyFinal() function inputs the MAC value in the MAC data area specified in the second argument, mac, and verifies the MAC value. If authentication fails, the return value will be TSIP_ERR_AUTHENTICATION. If the MAC value is less than 16 bytes, it will be padded with zeros inside the function.

<State transition>

The pre-run state is *TSIP Enabled State*.

After the function runs the state transitions to *TSIP Enabled State*.

Reentrant

Not supported

4.73 R_TSIP_Aes256CmacVerifyInit

Format

```
#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	input/output	AES-CMAC handler (work area)
key_index	input	user key index area

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_RESOURCE_CONFLICT:	A resource conflict occurred because a hardware resource needed by the processing routine was in use by another processing routine.
TSIP_ERR_KEY_SET:	Invalid user key index was input.
TSIP_ERR_FAIL:	An internal error occurred.

Description

The R_TSIP_Aes256CmacVerifyInit() function performs preparations for the execution of a CMAC calculation, and writes the result to the first argument, handle. The value of handle is used as an argument in the subsequent R_TSIP_Aes256CmacVerifyUpdate() function and R_TSIP_Aes256CmacVerifyFinal() function.

<State transition>

The pre-run state is *TSIP Enabled State*.

After the function runs the state transitions to *TSIP Enabled State*.

Refer to the Section 7 to generate key_index.

Reentrant

Not supported

4.74 R_TSIP_Aes256CmacVerifyUpdate

Format

#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 byte)
message_length	input	message data length (0 or more bytes)

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_PARAMETER:	An invalid handle was input.
TSIP_ERR_PROHIBIT_FUNCTION:	An invalid function was called.

Description

The R_TSIP_Aes256CmacVerifyUpdate() function performs MAC value generation based on the message specified in the second argument, message, using the value specified for key_index in R_TSIP_Aes256CmacGenerateInit(). Inside this function, the data that is input by the user is buffered until the input value of message exceeds 16 bytes. The length of the message data to input is specified in the third argument, message_len. For these, input not the total byte count for message input data, but rather the message data length to input when the user calls this function. If the input value, message, is not a multiple of 16 bytes, it will be padded within the function. For message, specify a RAM address that are multiples of 4.

<State transition>

The pre-run state is *TSIP Enabled State*.

After the function runs the state transitions to *TSIP Enabled State*.

Reentrant

Not supported

4.75 R_TSIP_Aes256CmacVerifyFinal

Format

```
#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/output	AES-CMAC handler (work area)
mac	input	MAC data area (mac_length byte)
mac_length	input/output	MAC data length (2 to 16 byte)

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_FAIL:	An internal error occurred.
TSIP_ERR_AUTHENTICATION:	Authentication failed
TSIP_ERR_PARAMETER:	Input data is illegal
TSIP_ERR_PROHIBIT_FUNCTION:	An invalid function was called.

Description

The R_TSIP_Aes256CmacVerifyFinal() function inputs the MAC value in the MAC data area specified in the second argument, mac, and verifies the MAC value. If authentication fails, the return value will be TSIP_ERR_AUTHENTICATION. If the MAC value is less than 16 bytes, it will be padded with zeros inside the function.

<State transition>

The pre-run state is *TSIP Enabled State*.

After the function runs the state transitions to *TSIP Enabled State*.

Reentrant

Not supported

4.76 R_TSIP_Aes128KeyWrap

Format

```
#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
)
```

Parameters

wrap_key_index	Input	AES-128 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 Other: Reserved
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 end
TSIP_ERR_RESOURCE_CONFLICT:	A resource conflict occurred because a hardware resource required by the processing is in use by other processing.
TSIP_ERR_KEY_SET:	Invalid user key index was input.
TSIP_ERR_FAIL:	An internal error occurred.

Description

The R_TSIP_Aes128KeyWrap() function uses wrap_key_index, the first argument, to wrap target_key_index, which is input as the third argument. The wrapped key is written to the fourth argument, wrapped_key. This processing conforms to the RFC3394 wrapping algorithm. Use the second argument, target_key_type, to select the key to be wrapped.

<State transition>

The valid pre-run state is *TSIP Enabled State*.

After the function runs the state transitions to *TSIP Enabled State*.

For details on how to use key_index, refer to section 7, Key Data Operations.

Reentrant

Not supported

4.77 R_TSIP_Aes256KeyWrap

Format

```
#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	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 Other: Reserved
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 end
TSIP_ERR_RESOURCE_CONFLICT:	A resource conflict occurred because a hardware resource required by the processing is in use by other processing.
TSIP_ERR_KEY_SET:	Invalid user key index was input.
TSIP_ERR_FAIL:	An internal error occurred.

Description

The R_TSIP_Aes256KeyWrap() function uses wrap_key_index, the first argument, to wrap target_key_index, which is input as the third argument. The wrapped key is written to the fourth argument, wrapped_key. This processing conforms to the RFC3394 wrapping algorithm. Use the second argument, target_key_type, to select the key to be wrapped.

<State transition>

The valid pre-run state is *TSIP Enabled State*.

After the function runs the state transitions to *TSIP Enabled State*.

For details on how to use key_index, refer to section 7, Key Data Operations.

Reentrant

Not supported

4.78 R_TSIP_Aes128KeyUnwrap

Format

```
#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
)
```

Parameters

wrap_key_index	Input	AES-128 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 Other: Reserved
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 end
TSIP_ERR_RESOURCE_CONFLICT:	A resource conflict occurred because a hardware resource required by the processing is in use by other processing.
TSIP_ERR_KEY_SET:	Invalid user key index was input.
TSIP_ERR_FAIL:	An internal error occurred.

Description

The R_TSIP_Aes128KeyUnwrap function uses wrap_key_index, the first argument, to unwrap wrapped_key, which is input as the third argument. The unwrapped key is written to the fourth argument, target_key_index. This processing conforms to the RFC3394 unwrapping algorithm. Use the second argument, target_key_type, to select the key to be unwrapped.

<State transition>

The valid pre-run state is *TSIP Enabled State*.

After the function runs the state transitions to *TSIP Enabled State*.

For details on how to use key_index, refer to section 7, Key Data Operations.

Reentrant

Not supported

4.79 R_TSIP_Aes256KeyUnwrap

Format

```
#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	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 Other: Reserved
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 end
TSIP_ERR_RESOURCE_CONFLICT:	A resource conflict occurred because a hardware resource required by the processing is in use by other processing.
TSIP_ERR_KEY_SET:	Invalid user key index was input.
TSIP_ERR_FAIL:	An internal error occurred.

Description

The R_TSIP_Aes256KeyUnwrap function uses wrap_key_index, the first argument, to unwrap wrapped_key, which is input as the third argument. The unwrapped key is written to the fourth argument, target_key_index. This processing conforms to the RFC3394 unwrapping algorithm. Use the second argument, target_key_type, to select the key to be unwrapped.

<State transition>

The valid pre-run state is *TSIP Enabled State*.

After the function runs the state transitions to *TSIP Enabled State*.

For details on how to use key_index, refer to section 7, Key Data Operations.

Reentrant

Not supported

5. Detailed Description of API Functions (for TSIP)

5.1 R_TSIP_Sha1Init

Format

#include “r_tsip_rx_if.h”

e_tsip_err_t R_TSIP_Sha1Init (tsip_sha_md5_handle_t *handle);

Parameters

handle	input/output	SHA handler (work area)

Return Values

TSIP_SUCCESS:	Normal termination

Description

The R_TSIP_Sha1Init() function performs preparations for the execution of an SHA1 hash calculation, and writes the result to the first argument, handle. The value of handle is used as an argument in the subsequent R_TSIP_Sha1Update() function and R_TSIP_Sha1Final() function.

<State transition>

The pre-run state is *TSIP Enabled State*.

After the function runs the state transitions to *TSIP Enabled State*.

Reentrant

Not supported

5.2 R_TSIP_Sha1Update

Format

#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);

Parameters

handle	input/output	SHA handler (work area)
message	input	message data area
message_length	input	message data length

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_RESOURCE_CONFLICT:	A resource conflict occurred because a hardware resource needed by the processing routine was in use by another processing routine.
TSIP_ERR_PARAMETER:	An invalid handle was input.
TSIP_ERR_PROHIBIT_FUNCTION:	An invalid function was called.

Description

The R_TSIP_Sha1Update() function calculates a hash value based on the second argument, message, and the third argument, message_length, utilizing in the first argument, handle, and writes the ongoing status to this first argument. After message input is completed, call R_TSIP_Sha1Final().

<State transition>

The pre-run state is *TSIP Enabled State*.

After the function runs the state transitions to *TSIP Enabled State*.

Reentrant

Not supported

5.3 R_TSIP_Sha1Final

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/output	SHA handler (work area)
digest	input/output	hash data area
digest_length	input/output	hash data length (20 bytes)

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_RESOURCE_CONFLICT:	A resource conflict occurred because a hardware resource needed by the processing routine was in use by another processing routine.
TSIP_ERR_PARAMETER:	An invalid handle was input.
TSIP_ERR_PROHIBIT_FUNCTION:	An invalid function was called.

Description

Using the handle specified in the first argument, handle, the R_TSIP_Sha1Final() function writes the calculation result to the second argument, digest, and writes the length of the calculation result to the third argument, digest_length.

<State transition>

The pre-run state is *TSIP Enabled State*.

After the function runs the state transitions to *TSIP Enabled State*.

Reentrant

Not supported

5.4 R_TSIP_Sha256Init

Format

```
#include "r_tsip_rx_if.h"
e_tsip_err_t R_TSIP_Sha256Init (tsip_sha_md5_handle_t *handle);
```

Parameters

handle	input/output	SHA handler (work area)

Return Values

TSIP_SUCCESS:	Normal termination

Description

The R_TSIP_Sha256Init() function performs preparations for the execution of an SHA-256 hash calculation, and writes the result to the first argument, handle. The value of handle is used as an argument in the subsequent R_TSIP_Sha256Update() function and R_TSIP_Sha256Final() function.

<State transition>

The pre-run state is *TSIP Enabled State*.

After the function runs the state transitions to *TSIP Enabled State*.

Reentrant

Not supported

5.5 R_TSIP_Sha256Update

Format

#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 data area
message_length	input	message data length

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_RESOURCE_CONFLICT:	A resource conflict occurred because a hardware resource needed by the processing routine was in use by another processing routine.
TSIP_ERR_PARAMETER:	An invalid handle was input.
TSIP_ERR_PROHIBIT_FUNCTION:	An invalid function was called.

Description

The R_TSIP_Sha256Update() function calculates a hash value based on the second argument, message, and the third argument, message_length, utilizing in the first argument, handle, and writes the ongoing status to this first argument. After message input is completed, call R_TSIP_Sha256Final().

<State transition>

The pre-run state is *TSIP Enabled State*.

After the function runs the state transitions to *TSIP Enabled State*.

Reentrant

Not supported

5.6 R_TSIP_Sha256Final

Format

#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/output	SHA handler (work area)
digest	input/output	hash data area
digest_length	input/output	hash data length (32bytes)

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_RESOURCE_CONFLICT:	A resource conflict occurred because a hardware resource needed by the processing routine was in use by another processing routine.
TSIP_ERR_PARAMETER:	An invalid handle was input.
TSIP_ERR_PROHIBIT_FUNCTION:	An invalid function was called.

Description

Using the handle specified in the first argument, handle, the R_TSIP_Sha256Final() function writes the calculation result to the second argument, digest, and writes the length of the calculation result to the third argument, digest_length.

<State transition>

The pre-run state is *TSIP Enabled State*.

After the function runs the state transitions to *TSIP Enabled State*.

Reentrant

Not supported

5.7 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	input/output	MD5 handler (work area)

Return Values

TSIP_SUCCESS:	Normal termination

Description

The R_TSIP_Md5Init() function prepares to calculate the MD5 hash and writes the result to the first argument, handle. The subsequent functions R_TSIP_Md5Update() and R_TSIP_Md5Final() also use handle as an argument.

< State transition >

The state before a valid run is *TSIP Enabled State*.

After the function runs the state is *TSIP Enabled State*.

Reentrant

Not supported

5.8 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 data area
message_length	input	message data length in bytes

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_RESOURCE_CONFLICT:	A resource conflict occurred because a hardware resource required for processing is in use by another processing routine.
TSIP_ERR_PARAMETER:	An illegal handle was input.
TSIP_ERR_PROHIBIT_FUNCTION:	An illegal function was called.

Description

The R_TSIP_Md5Update() function uses the handle specified by the first argument, handle, and calculates a hash value from the second argument, message, and the third argument, message_length, writing the progress along the way to the first argument, handle. After message input completes, call R_TSIP_Md5Final().

< State transition >

The pre-run state is *TSIP Enabled State*.

After the function runs the state is *TSIP Enabled State*.

Reentrant

Not supported

5.9 R_TSIP_Md5Final

Format

```
#include "r_tsip_rx_if.h"
e_tsip_err_t R_TSIP_Md5Final
        (tsip_sha_md5_handle_t *handle, uint8_t *digest, uint32_t *digest_length);
```

Parameters

handle	input/output	MD5 handler (work area)
digest	input/output	hash data area
digest_length	input/output	hash data length

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_RESOURCE_CONFLICT:	A resource conflict occurred because a hardware resource required for processing is in use by another processing routine.
TSIP_ERR_PARAMETER:	An illegal handle was input.
TSIP_ERR_PROHIBIT_FUNCTION:	An illegal function was called.

Description

The R_TSIP_Md5Final() function writes the calculation result to the second argument, digest, and the length of the calculation result to the third argument, digest_length, using the handle specified by the first argument handle.

< State transition >

The pre-run state is *TSIP Enabled State*.

After the function runs the state is *TSIP Enabled State*.

Reentrant

Not supported

5.10 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	Initial vector when generating encrypted_key
encrypted_key	Input	Encrypted Triple-DES user key with MAC appended
key_index	Input/output	Triple-DES user key index

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_RESOURCE_CONFLICT:	A resource conflict occurred because a hardware resource required for processing is in use by another processing routine.
TSIP_ERR_FAIL:	An internal error occurred.

Description

This API outputs Triple-DES user key index.

Input data in the following format as encrypted_key.

byte	128-bit			
	32-bit	32-bit	32-bit	32-bit
0-15	Encrypted Triple-DES key			
16-31				
32-47	MAC			

For instructions for inputting a key for use as a DES or 2TDES (2-key TDES) key, refer to Chapter 7, Key Data Operations.

< State transition >

The state before a valid run is *TSIP Enabled State*.

After the function runs the state transitions to *TSIP Enabled State*.

For the method of generating encrypted_key, iv, and encrypted_provisioning_key, and instructions for using key_index, refer to Chapter 7, Key Data Operations.

Reentrant

Not supported

5.11 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	input/output	Triple-DES user key index (13 words)

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_RESOURCE_CONFLICT:	A resource conflict occurred because a hardware resource required for processing is in use by another processing routine.

Description

This API outputs Triple-DES user key index.

This API is used to generate a user key from a random number internally in the TSIP. Consequentially, there is no need to input a user key. The user key index output by this API can be used to encrypt data and thereby prevent dead copying.

< State transition >

The state before a valid run is *TSIP Enabled State*.

After the function runs the state transitions to *TSIP Enabled State*.

For instructions for using key_index, refer to "7. Key Data Operations."

Reentrant

Not supported

5.12 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 with key update keyring with MAC appended
key_index	Input/output	Triple-DES user key index

Return Values

TSIP_SUCCESS:	Normal end
TSIP_ERR_RESOURCE_CONFLICT:	A resource conflict occurred because a hardware resource needed by this processing routine was in use by another processing routine.
TSIP_ERR_FAIL:	An internal error occurred.

Description

This API updates the Triple-DES key index.

Input data encrypted in the following format with the key update keyring as encrypted_key.

byte	128-bit			
	32-bit	32-bit	32-bit	32-bit
0-15	Triple-DES key			
16-31				
32-47	MAC			

< State transition >

The pre-run state is *TSIP Enabled State*.

After the function runs the state transitions to *TSIP Enabled State*.

For the method of generating iv and encrypted_key, and instructions for using key_index, refer to Chapter 7, Key Data Operations.

Reentrant

Not supported

5.13 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	input/output	TDES handler (work area)
key_index	input	user key index area

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_RESOURCE_CONFLICT:	A resource conflict occurred because a hardware resource required for processing is in use by another processing routine.
TSIP_ERR_KEY_SET:	Incorrect user key index was input.

Description

The R_TSIP_TdesEcbEncryptInit() function prepares to perform DES calculation and writes the result to the first argument, handle. The subsequent functions R_TSIP_TdesEcbEncryptUpdate() function and R_TSIP_TdesEcbEncryptFinal() also use handle as an argument.

< State transition >

The state before a valid run is *TSIP Enabled State*.

After the function runs the state is *TSIP Enabled State*.

For instructions for using key_index, refer to "7. Key Data Operations."

Reentrant

Not supported

5.14 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	input/output	ciphertext data area
plain_length	input	plaintext data length (Must be a multiple of 8.)

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_PARAMETER:	An illegal handle was input.
TSIP_ERR_PROHIBIT_FUNCTION:	An illegal function was called.

Description

The R_TSIP_TdesEcbEncryptUpdate() function uses the handle specified by the first argument, handle, and encrypts the contents of the second argument, plain, using the key_index stored in handle, writing the progress along the way to the first argument, handle. It also writes the encrypted result to the third argument, cipher. After plaintext input finishes, call R_TSIP_TdesEcbEncryptFinal().

Ensure that plain and cipher are not assigned to overlapping areas. Also, specify RAM addresses for plain and cipher that are multiples of 4.

< State transition >

The pre-run state is *TSIP Enabled State*.

After the function runs the state is *TSIP Enabled State*.

For instructions for using key_index, refer to “7. Key Data Operations.”

Reentrant

Not supported

5.15 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/output	TDES handler (work area)
cipher	input/output	ciphertext data area (Nothing is ever written to this area.)
cipher_length	input/output	ciphertext data length (Zero is always written to this area.)

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_FAIL :	An internal error occurred.
TSIP_ERR_PARAMETER:	An illegal handle was input.
TSIP_ERR_PROHIBIT_FUNCTION:	An illegal function was called.

Description

The R_TSIP_TdesEcbEncryptFinal() function writes the calculation result to the second argument, cipher, and the length of the calculation result to the third argument, cipher_length, using the handle specified by the first argument, handle. The leftover amount less than a multiple of 8 bytes was originally supposed to be encrypted and the result written to the second argument, but the Update function has a restriction that only allows it to handle values that are multiples of 8 bytes. Therefore, this function never actually writes anything to cipher and it always writes 0 to cipher_length. The arguments cipher and cipher_length are provided to ensure compatibility in case this restriction is removed in future.

< State transition >

The pre-run state is *TSIP Enabled State*.

After the function runs the state is *TSIP Enabled State*.

Reentrant

Not supported

5.16 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	input/output	TDES handler (work area)
key_index	input	user key index area

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_KEY_SET:	Incorrect user key index was input.
TSIP_ERR_RESOURCE_CONFLICT:	A resource conflict occurred because a hardware resource required for processing is in use by another processing routine.

Description

The R_TSIP_TdesEcbDecryptInit() function prepares to perform DES calculation and writes the result to the first argument, handle. The subsequent functions R_TSIP_TdesEcbDecryptUpdate() function and R_TSIP_TdesEcbDecryptFinal() also use handle as an argument.

< State transition >

The state before a valid run is *TSIP Enabled State*.

After the function runs the state is *TSIP Enabled State*.

For instructions for using key_index, refer to “7. Key Data Operations.”

Reentrant

Not supported

5.17 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	input/output	plaintext data area
cipher_length	input	ciphertext data length (Must be a multiple of 8.)

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_PARAMETER:	An illegal handle was input.
TSIP_ERR_PROHIBIT_FUNCTION:	An illegal function was called.

Description

The R_TSIP_TdesEcbDecryptUpdate() function uses the handle specified by the first argument, handle, and decrypts the contents of the second argument, cipher, using the key_index stored in handle, writing the progress along the way to the first argument, handle. It also writes the encrypted result to the third argument, plain. After ciphertext input finishes, call R_TSIP_TdesEcbDecryptFinal().

Ensure that plain and cipher are not assigned to overlapping areas. Also, specify RAM addresses for plain and cipher that are multiples of 4.

< State transition >

The pre-run state is *TSIP Enabled State*.

After the function runs the state is *TSIP Enabled State*.

Reentrant

Not supported

5.18 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/output	TDES handler (work area)
plain	input/output	plaintext data area (Nothing is ever written to this area.)
plain_length	input/output	plaintext data length (Zero is always written to this area.)

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_FAIL:	An internal error occurred.
TSIP_ERR_PARAMETER:	An illegal handle was input.
TSIP_ERR_PROHIBIT_FUNCTION:	An illegal function was called.

Description

The R_TSIP_TdesEcbDecryptFinal() function writes the calculation result to the second argument, plain, and the length of the calculation result to the third argument, plain_length, using the handle specified by the first argument, handle. The leftover amount less than a multiple of 8 bytes was originally supposed to be encrypted and the result written to the second argument, but the Update function has a restriction that only allows it to handle values that are multiples of 8 bytes. Therefore, this function never actually writes anything to plain and it always writes 0 to plain_length. The arguments plain and plain_length are provided to ensure compatibility in case this restriction is removed in future.

< State transition >

The pre-run state is *TSIP Enabled State*.

After the function runs the state is *TSIP Enabled State*.

Reentrant

Not supported

5.19 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	input/output	TDES handler (work area)
key_index	input	user key index area
ivec	input	initialization vector(8byte)

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_KEY_SET:	Incorrect user key index was input.
TSIP_ERR_RESOURCE_CONFLICT:	A resource conflict occurred because a hardware resource required for processing is in use by another processing routine.

Description

The R_TSIP_TdesCbcEncryptInit() function prepares to perform DES calculation and writes the result to the first argument, handle. The subsequent functions R_TSIP_TdesCbcEncryptUpdate() function and R_TSIP_TdesCbcEncryptFinal() also use handle as an argument.

< State transition >

The state before a valid run is *TSIP Enabled State*.

After the function runs the state is *TSIP Enabled State*.

For instructions for using key_index, refer to “7. Key Data Operations.”

Reentrant

Not supported

5.20 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	input/output	ciphertext data area
plain_length	input	plaintext data length (Must be a multiple of 8.)

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_PARAMETER:	An illegal handle was input.
TSIP_ERR_PROHIBIT_FUNCTION:	An illegal function was called.

Description

The R_TSIP_TdesCbcEncryptUpdate() function uses the handle specified by the first argument, handle, and encrypts the contents of the second argument, plain, using the key_index stored in handle, writing the progress along the way to the first argument, handle. It also writes the encrypted result to the third argument, cipher. After plaintext input finishes, call R_TSIP_TdesCbcEncryptFinal().

Ensure that plain and cipher are not assigned to overlapping areas. Also, specify RAM addresses for plain and cipher that are multiples of 4.

< State transition >

The pre-run state is *TSIP Enabled State*.

After the function runs the state is *TSIP Enabled State*.

For instructions for using key_index, refer to “7. Key Data Operations.”

Reentrant

Not supported

5.21 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/output	TDES handler (work area)
cipher	input/output	ciphertext data area (Nothing is ever written to this area.)
cipher_length	input/output	ciphertext data length (Zero is always written to this area.)

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_FAIL:	An internal error occurred.
TSIP_ERR_PARAMETER:	An illegal handle was input.
TSIP_ERR_PROHIBIT_FUNCTION:	An illegal function was called.

Description

The R_TSIP_TdesCbcEncryptFinal() function writes the calculation result to the second argument, cipher, and the length of the calculation result to the third argument, cipher_length, using the handle specified by the first argument, handle. The leftover amount less than a multiple of 8 bytes was originally supposed to be encrypted and the result written to the second argument, but the Update function has a restriction that only allows it to handle values that are multiples of 8 bytes. Therefore, this function never actually writes anything to cipher and it always writes 0 to cipher_length. The arguments cipher and cipher_length are provided to ensure compatibility in case this restriction is removed in future.

< State transition >

The pre-run state is *TSIP Enabled State*.

After the function runs the state is *TSIP Enabled State*.

Reentrant

Not supported

5.22 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	input/output	TDES handler (work area)
key_index	input	user key index area
ivec	input	initialization vector(16byte)

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_KEY_SET:	Incorrect user key index was input.
TSIP_ERR_RESOURCE_CONFLICT:	A resource conflict occurred because a hardware resource required for processing is in use by another processing routine.

Description

The R_TSIP_TdesCbcDecryptInit() function prepares to perform DES calculation and writes the result to the first argument, handle. The subsequent functions R_TSIP_TdesCbcDecryptUpdate() function and R_TSIP_TdesCbcDecryptFinal() also use handle as an argument.

< State transition >

The state before a valid run is *TSIP Enabled State*.

After the function runs the state is *TSIP Enabled State*.

For instructions for using key_index, refer to “7. Key Data Operations.”

Reentrant

Not supported

5.23 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	input/output	plaintext data area
cipher_length	input	ciphertext data length (Must be a multiple of 16.)

Return Values

TSIP_SUCCESS :	Normal termination
TSIP_ERR_PARAMETER:	An illegal handle was input.
TSIP_ERR_PROHIBIT_FUNCTION:	An illegal function was called.

Description

The R_TSIP_TdesCbcDecryptUpdate() function uses the handle specified by the first argument, handle, and decrypts the contents of the second argument, cipher, using the key_index stored in handle, writing the progress along the way to the first argument, handle. It also writes the encrypted result to the third argument, plain. After ciphertext input finishes, call R_TSIP_TdesCbcDecryptFinal().

Ensure that plain and cipher are not assigned to overlapping areas. Also, specify RAM addresses for plain and cipher that are multiples of 4.

< State transition >

The pre-run state is *TSIP Enabled State*.

After the function runs the state is *TSIP Enabled State*.

Reentrant

Not supported

5.24 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/output	TDES handler (work area)
plain	input/output	plaintext data area (Nothing is ever written to this area.)
plain_length	input/output	plaintext data length (Zero is always written to this area.)

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_FAIL:	An internal error occurred.
TSIP_ERR_PARAMETER:	An illegal handle was input.
TSIP_ERR_PROHIBIT_FUNCTION:	An illegal function was called.

Description

The R_TSIP_TdesCbcDecryptFinal() function writes the calculation result to the second argument, plain, and the length of the calculation result to the third argument, plain_length, using the handle specified by the first argument, handle. The leftover amount less than a multiple of 8 bytes was originally supposed to be encrypted and the result written to the second argument, but the Update function has a restriction that only allows it to handle values that are multiples of 8 bytes. Therefore, this function never actually writes anything to plain and it always writes 0 to plain_length. The arguments plain and plain_length are provided to ensure compatibility in case this restriction is removed in future.

< State transition >

The pre-run state is *TSIP Enabled State*.

After the function runs the state is *TSIP Enabled State*.

Reentrant

Not supported

5.25 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 used when generating encrypted_key
encrypted_key	Input	ARC4 user key with encrypted MAC appended
key_index	Input/output	ARC4 user key index

Return Values

TSIP_SUCCESS:	Normal end
TSIP_ERR_RESOURCE_CONFLICT:	A resource conflict occurred because a hardware resource needed by this processing routine was in use by another processing routine.
TSIP_ERR_FAIL:	An internal error occurred.

Description

This API outputs an ARC4 user key index.

Input data in the following format as the encrypted_key.

byte	128 bit			
	32bit	32bit	32bit	32bit
0-255	Encrypted ARC4 key			
256-271	MAC			

< State transition >

The valid pre-run state is *TSIP Enabled State*.

After the function runs the state transitions to *TSIP Enabled State*.

For the method of generating encrypted_key, iv, and encrypted_provisioning_key, and instructions for using key_index, refer to Chapter 7, Key Data Operations.

Reentrant

Not supported

5.26 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	Input/output	ARC4 user key index

Return Values

TSIP_SUCCESS:	Normal end
TSIP_ERR_RESOURCE_CONFLICT:	A resource conflict occurred because a hardware resource needed by this processing routine was in use by another processing routine.

Description

This API outputs an ARC4 user key index.

This API generates a user key from a random number internally in the TSIP. Accordingly, user key input is unnecessary. By encrypting data using the user key index that is output by this API, dead copying of data can be prevented.

< State transition >

The valid pre-run state is *TSIP Enabled State*.

After the function runs the state transitions to *TSIP Enabled State*.

For instructions for using key_index, refer to Chapter 7, Key Data Operations.

Reentrant

Not supported

5.27 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 with MAC encrypted with key update keyring appended
key_index	Input/output	ARC4 user key index

Return Values

TSIP_SUCCESS:	Normal end
TSIP_ERR_RESOURCE_CONFLICT:	A resource conflict occurred because a hardware resource needed by this processing routine was in use by another processing routine.
TSIP_ERR_FAIL:	An internal error occurred.

Description

This API updates the key index of an ARC4 key.

Input data encrypted in the following format with the key update keyring as encrypted_key.

byte	128 bit			
	32bit	32bit	32bit	32bit
0-255	ARC4 key			
256-271	MAC			

< State transition >

The valid pre-run state is *TSIP Enabled State*.

After the function runs the state transitions to *TSIP Enabled State*.

For the method of generating iv and encrypted_key, and instructions for using key_index, refer to Chapter 7, Key Data Operations.

Reentrant

Not supported

5.28 R_TSIP_Arc4EncryptInit

Format

```
#include "r_tsip_rx_if.h"
e_tsip_err_t R_TSIP_Arc4EcbEncryptInit(
    tsip_arc4_handle_t *handle,
        tsip_arc4_key_index_t *key_index
)
```

Parameters

handle	Input/output	ARC4 handler (work area)
key_index	Input	ARC4 user key index area

Return Values

TSIP_SUCCESS:	Normal end
TSIP_ERR_RESOURCE_CONFLICT:	A resource conflict occurred because a hardware resource needed by this processing routine was in use by another processing routine.
TSIP_ERR_KEY_SET:	An invalid user key index was input.

Description

The R_TSIP_Arc4EncryptInit() function performs preparations for the execution of an ARC4 calculation, and writes the result to the first argument, handle. The value of handle is used as an argument in the subsequent R_TSIP_Arc4EncryptUpdate() function and R_TSIP_Arc4EncryptFinal() function.

< State transition >

The valid pre-run state is *TSIP Enabled State*.

After the function runs the state transitions to *TSIP Enabled State*.

For instructions for using key_index, refer to Chapter 7, Key Data Operations.

Reentrant

Not supported

5.29 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	Input/output	Ciphertext data area
plain_length	Input	Plaintext data length (must be a multiple of 16)

Return Values

TSIP_SUCCESS:	Normal end
TSIP_ERR_PARAMETER:	An invalid handle was input.
TSIP_ERR_PROHIBIT_FUNCTION:	An invalid function was called.

Description

The R_TSIP_Arc4EncryptUpdate() function encrypts the second argument, plain, utilizing the key index stored in the handle specified in the first argument, handle, and writes the ongoing status to this first argument. In addition, it writes the encryption result to the third argument, cipher. After plaintext input is completed, call R_TSIP_Arc4EncryptFinal().

Specify areas for plain and cipher that do not overlap. For plain and cipher, specify RAM addresses that are multiples of 4.

< State transition >

The pre-run state is *TSIP Enabled State*.

After the function runs the state transitions to *TSIP Enabled State*.

For instructions for using key_index, refer to Chapter 7, Key Data Operations.

Reentrant

Not supported

5.30 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/output	ARC4 handler (work area)
cipher	Input/output	Ciphertext data area (nothing ever written here)
cipher_length	Input/output	Ciphertext data length (0 always written here)

Return Values

TSIP_SUCCESS:	Normal end
TSIP_ERR_FAIL:	An internal error occurred.
TSIP_ERR_PARAMETER:	An invalid handle was input.
TSIP_ERR_PROHIBIT_FUNCTION:	An invalid function was called.

Description

Using the handle specified in the first argument, handle, the R_TSIP_Arc4EncryptFinal() function writes the calculation result to the second argument, cipher, and writes the length of the calculation result to the third argument, cipher_length. The original intent was for the portion of the encryption result that was not a multiple of 16 bytes to be written to the second argument. However, as a result of the restriction that only multiples of 16 can be input to the Update function, nothing is ever written to cipher, and 0 is always written to cipher_length. The arguments cipher and cipher_length are provided for compatibility in anticipation of the time when this restriction is lifted.

< State transition >

The pre-run state is *TSIP Enabled State*.

After the function runs the state transitions to *TSIP Enabled State*.

Reentrant

Not supported

5.31 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	Input/output	ARC4 handler (work area)
key_index	Input	ARC4 user key index area

Return Values

TSIP_SUCCESS:	Normal end
TSIP_ERR_KEY_SET:	An invalid user key index was input.
TSIP_ERR_RESOURCE_CONFLICT:	A resource conflict occurred because a hardware resource needed by this processing routine was in use by another processing routine.

Description

The R_TSIP_Arc4DecryptInit() function performs preparations for the execution of an ARC4 calculation, and writes the result to the first argument, handle. The value of handle is used as an argument in the subsequent R_TSIP_Arc4DecryptUpdate() function and R_TSIP_Arc4DecryptFinal() function.

< State transition >

The valid pre-run state is *TSIP Enabled State*.

After the function runs the state transitions to *TSIP Enabled State*.

For instructions for using key_index, refer to Chapter 7, Key Data Operations.

Reentrant

Not supported

5.32 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	Input/output	Plaintext data area
cipher_length	Input	Ciphertext data length (must be a multiple of 16)

Return Values

TSIP_SUCCESS:	Normal end
TSIP_ERR_PARAMETER:	An invalid handle was input.
TSIP_ERR_PROHIBIT_FUNCTION:	An invalid function was called.

Description

The R_TSIP_Arc4DecryptUpdate() function decrypts the second argument, cipher, utilizing the key index stored in the handle specified in the first argument, handle, and writes the ongoing status to this first argument. In addition, it writes the decryption result to the third argument, plain. After ciphertext input is completed, call R_TSIP_Arc4DecryptFinal().

Specify areas for plain and cipher that do not overlap. For plain and cipher, specify RAM addresses that are multiples of 4.

< State transition >

The pre-run state is *TSIP Enabled State*.

After the function runs the state transitions to *TSIP Enabled State*.

Reentrant

Not supported

5.33 R_TSIP_Arc4DecryptFinal

Format

```
#include "r_tsip_rx_if.h"
e_tsip_err_t R_TSIP_Arc4DecryptFinal(
    tsip_arc4_handle_t *handle,
        uint8_t *plain,
        uint32_t *plain_length
)
```

Parameters

handle	Input/output	ARC4 handler (work area)
plain	Input/output	Plaintext data area (nothing ever written here)
plain_length	Input/output	Plaintext data length (0 always written here)

Return Values

TSIP_SUCCESS:	Normal end
TSIP_ERR_FAIL:	An internal error occurred.
TSIP_ERR_PARAMETER:	An invalid handle was input.
TSIP_ERR_PROHIBIT_FUNCTION:	An invalid function was called.

Description

Using the handle specified in the first argument, handle, the R_TSIP_Arc4DecryptFinal() function writes the calculation result to the second argument, plain, and writes the length of the calculation result to the third argument, plain_length. The original intent was for the portion of the decryption result that was not a multiple of 16 bytes to be written to the second argument. However, as a result of the restriction that only multiples of 16 can be input to the Update function, nothing is ever written to plain, and 0 is always written to plain_length. The arguments plain and plain_length are provided for compatibility in anticipation of the time when this restriction is lifted.

< State transition >

The pre-run state is *TSIP Enabled State*.

After the function runs the state transitions to *TSIP Enabled State*.

Reentrant

Not supported

5.34 R_TSIP_GenerateRsa1024PublicKeyIndex

Format

```
#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);
```

Parameters

encrypted_provisioning_key	Input	Provisioning key wrapped by the DLM server
iv	Input	Initial vector used when generating encrypted_key
encrypted_key	Input	Encrypted RSA 1024-bit public key with MAC appended
key_index	Input/output	RSA 1024-bit public key user key index
key_index->value.key_management_info1		: Key management information
key_index->value.key_n		: RSA 1024-bit public key n (plaintext)
key_index->value.key_e		: RSA 1024-bit public key e (plaintext)
key_index->value.dummy		: Dummy
key_index->value.key_management_info2		: Key management information

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_RESOURCE_CONFLICT:	A resource conflict occurred because a hardware resource needed by the processing routine was in use by another processing routine.
TSIP_ERR_FAIL:	An internal error occurred.

Description

This API outputs a 1024-bit RSA public key user key index.

Input data encrypted in the following format with the provisining key as encrypted_key.

byte	128-bit			
	32-bit	32-bit	32-bit	32-bit
0-127	RSA 1024-bit public key n			
128-143	RSA 1024-bit public key e	0 padding		
144-159	MAC			

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

<State transition>

The pre-run state is *TSIP Enabled State*.

After the function runs the state transitions to *TSIP Enabled State*.

For the method of generating encrypted_provisioning_key, iv, and encrypted_key, and instructions for using key_index, refer to Chapter 7, Key Data Operations.

Reentrant

Not supported

5.35 R_TSIP_GenerateRsa1024PrivateKeyIndex

Format

#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);

Parameters

encrypted_provisioning_key	Input	Provisioning key wrapped by the DLM server
iv	Input	Initial vector used when generating encrypted_key
encrypted_key ppended	Input	Encrypted RSA 1024-bit private key with MAC
key_index	Input/output	RSA 1024-bit private key user key index

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_RESOURCE_CONFLICT:	A resource conflict occurred because a hardware resource needed by the processing routine was in use by another processing routine.
TSIP_ERR_FAIL:	An internal error occurred.

Description

This API outputs a 1024-bit RSA private user key user key index.

Input data encrypted in the following format with the provisioning key as encrypted_key.

byte	128-bit			
	32-bit	32-bit	32-bit	32-bit
0-127	RSA 1024-bit public key n			
128-255	RSA 1024-bit private key d			
256-271	MAC			

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

<State transition>

The pre-run state is *TSIP Enabled State*.

After the function runs the state transitions to *TSIP Enabled State*.

For the method of generating encrypted_provisioning_key, iv, and encrypted_key, and instructions for using key_index, refer to Chapter 7, Key Data Operations.

Reentrant

Not supported

5.36 R_TSIP_GenerateRsa2048PublicKeyIndex

Format

```
#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);
```

Parameters

encrypted_provisioning_key	Input	Provisioning key wrapped by the DLM server
iv	Input	Initial vector used when generating encrypted_key
encrypted_key	Input	Encrypted RSA 2048-bit public key with MAC appended
key_index	Input/output	RSA 2048-bit public key user key index
key_index->value.key_management_info1		: Key management information
key_index->value.key_n		: RSA 2048-bit public key n (plaintext)
key_index->value.key_e		: RSA 2048-bit public key e (plaintext)
key_index->value.dummy		: Dummy
key_index->value.key_management_info2		: Key management information

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_RESOURCE_CONFLICT:	A resource conflict occurred because a hardware resource needed by the processing routine was in use by another processing routine.
TSIP_ERR_FAIL:	An internal error occurred.

Description

This API outputs a 2048-bit RSA public key user key index.

Input data encrypted in the following format with the provisioioning key as encrypted_key.

byte	128-bit			
	32-bit	32-bit	32-bit	32-bit
0-255	RSA 2048-bit public key n			
256-272	RSA 2048-bit public key e	0 padding		
272-287	MAC			

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

<State transition>

The pre-run state is *TSIP Enabled State*.

After the function runs the state transitions to *TSIP Enabled State*.

For the method of generating encrypted_provisioning_key, iv, and encrypted_key, and instructions for using key_index, refer to Chapter 7, Key Data Operations.

Reentrant

Not supported

5.37 R_TSIP_GenerateRsa2048PrivateKeyIndex

Format

```
#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	Initial vector used when generating encrypted_key
encrypted_key	Input	Encrypted RSA 2048-bit private key with MAC ppended
key_index	Input/output	RSA 2048-bit private key user key index

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_RESOURCE_CONFLICT:	A resource conflict occurred because a hardware resource needed by the processing routine was in use by another processing routine.
TSIP_ERR_FAIL:	An internal error occurred.

Description

This API outputs a 2048-bit RSA private key user key index.

Input data encrypted in the following format with the provisioning key as encrypted_key.

byte	128-bit			
	32-bit	32-bit	32-bit	32-bit
0-255	RSA 2048-bit public key n			
256-511	RSA 2048-bit private key d			
512-527	MAC			

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

<State transition>

The pre-run state is *TSIP Enabled State*.

After the function runs the state transitions to *TSIP Enabled State*.

For the method of generating encrypted_provisioning_key, iv, and encrypted_key, and instructions for using key_index and install_key_index, refer to Chapter 7, Key Data Operations.

Reentrant

Not supported

5.38 R_TSIP_GenerateRsa1024RandomKeyIndex

Format

```
#include "r_tsip_rx_if.h"
e_tsip_err_t R_TSIP_GenerateRsa1024RandomKeyIndex
        (tsip_rsa1024_key_pair_index_t *key_pair_index);
```

Parameters

key_pair_index Input/output User key index for RSA 1024-bit public key and private key pair

key_pair_index->public : RSA 1024-bit public key user key index
key_pair_index->public.value.key_management_info1 : Key management information
key_pair_index->public.value.key_n : RSA 1024-bit public key n (plaintext)
key_pair_index->public.value.key_e : RSA 1024-bit public key e (plaintext)
key_pair_index->public.value.dummy : Dummy
key_pair_index->public.value.key_management_info2 : Key management information
key_pair_index->private : RSA 1024-bit private key user key index

Return Values

TSIP_SUCCESS:	Normal end
TSIP_ERR_RESOURCE_CONFLICT:	A resource conflict occurred because a hardware resource needed by this processing routine was in use by another processing routine.
TSIP_ERR_FAIL:	An internal error occurred. Key generation failed.

Description

This API outputs a user key index for a 1024-bit RSA public key and private key pair. The API generates a user key from a random value produced internally by the TSIP. Consequently, there is no need to input a user key. Dead copying of data can be prevented by encrypting the data using the user key index output by this API. A public key user key index is generated by key_pair_index->public, and a private key user key index is generated by key_pair_index->private. As the public key exponent, only 0x00010001 is generated.

<State transition>

The valid pre-run state is *TSIP enabled.*

The pre-run state is *TSIP Disabled State*.

For the method of using key_pair_index->public and key_pair_index->private, refer to Chapter 7, Key Data Operations.

key_pair_index->public is the same operation as the public key user key index output from R_TSIP_GenerateRsa1024PublicKeyIndex(), and Key_pair_index->private is the same operation as the private key user key index output from R_TSIP_GenerateRsa1024PrivateKeyIndex().

Reentrant

Not supported

5.39 R_TSIP_GenerateRsa2048RandomKeyIndex

Format

#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 Input/output User key index for RSA 2048-bit public key and private key pair

key_pair_index->public	: RSA 2048-bit public key user key index
key_pair_index->public.value.key_management_info1	: Key management information
key_pair_index->public.value.key_n	: RSA 2048-bit public key n (plaintext)
key_pair_index->public.value.key_e	: RSA 2048-bit public key e (plaintext)
key_pair_index->public.value.dummy	: Dummy
key_pair_index->public.value.key_management_info2	: Key management information
key_pair_index->private	: RSA 2048-bit private key user key index

Return Values

TSIP_SUCCESS:	Normal end
TSIP_ERR_RESOURCE_CONFLICT:	A resource conflict occurred because a hardware resource needed by this processing routine was in use by another processing routine.
TSIP_ERR_FAIL:	An internal error occurred. Key generation failed.

Description

This API outputs a user key index for a 2048-bit RSA public key and private key pair. The API generates a user key from a random value produced internally by the TSIP. Consequently, there is no need to input a user key. Dead copying of data can be prevented by encrypting the data using the user key index output by this API. A public key user key index is generated by key_pair_index->public, and a private key user key index is generated by key_pair_index->private. As the public key exponent, only 0x00010001 is generated.

<State transition>

The valid pre-run state is *TSIP enabled.*

The pre-run state is *TSIP Disabled State*.

For the method of using key_pair_index->public and key_pair_index->private, refer to Chapter 7, Key Data Operations.

key_pair_index->public is the same operation as the public key user key index output from R_TSIP_GenerateRsa2048PublicKeyIndex(), and Key_pair_index->private is the same operation as the private key user key index output from R_TSIP_GenerateRsa2048PrivateKeyIndex().

Reentrant

Not supported

5.40 R_TSIP_UpdateRsa1024PublicKeyIndex

Format

```
#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);
```

Parameters

iv	Input	Initialization vector when generating encrypted_key
encrypted_key	Input	Public key encrypted with key update keyring with MAC appended
key_index	Input/output	RSA 1024-bit public key user key index

key_index->value.key_management_info1 : Key management information
key_index->value.key_n : RSA 1024-bit public key n (plaintext)
key_index->value.key_e : RSA 1024-bit public key e (plaintext)
key_index->value.dummy : Dummy
key_index->value.key_management_info2 : Key management information

Return Values

TSIP_SUCCESS:	Normal end
TSIP_ERR_RESOURCE_CONFLICT:	A resource conflict occurred because a hardware resource needed by this processing routine was in use by another processing routine.
TSIP_ERR_FAIL	An internal error occurred.

Description

This API updates an RSA 1024-bit public key user key index.

Input data encrypted in the following format with the key update keyring as encrypted_key.

byte	128-bit			
	32-bit	32-bit	32-bit	32-bit
0-127	RSA 1024-bit public key n			
128-143	RSA 1024-bit public key e	0 padding		
144-159	MAC			

<State transition>

The valid pre-run state is *TSIP enabled.*

After the function runs the state transitions to *TSIP Enabled State*.

For the method of generating iv and encrypted_key, and instructions for using key_index, refer to Chapter 7, Key Data Operations.

Reentrant

Not supported

5.41 R_TSIP_UpdateRsa1024PrivateKeyIndex

Format

```
#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);
```

Parameters

iv	Input	Initialization vector when generating encrypted_key
encrypted_key	Input	Private key encrypted with key update keyring with MAC appended
key_index	Input/output	RSA 1024-bit private key user key index

Return Values

TSIP_SUCCESS:	Normal end
TSIP_ERR_RESOURCE_CONFLICT:	A resource conflict occurred because a hardware resource needed by this processing routine was in use by another processing routine.
TSIP_ERR_FAIL	An internal error occurred.

Description

This API updates an RSA 1024-bit private key user key index.

Input data encrypted in the following format with the key update keyring as encrypted_key.

byte	128-bit			
	32-bit	32-bit	32-bit	32-bit
0-127	RSA 1024-bit public key n			
128-255	RSA 1024-bit private key d			
256-271	MAC			

<State transition>

The valid pre-run state is *TSIP enabled.*

After the function runs the state transitions to *TSIP Enabled State*.

For the method of generating iv and encrypted_key, and instructions for using key_index, refer to Chapter 7, Key Data Operations.

Reentrant

Not supported

5.42 R_TSIP_UpdateRsa2048PublicKeyIndex

Format

```
#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);
```

Parameters

iv	Input	Initialization vector when generating encrypted_key
encrypted_key	Input	Public key encrypted with key update keyring with MAC appended
key_index	Input/output	RSA 2048-bit public key user key index

key_index->value.key_management_info1 : Key management information
key_index->value.key_n : RSA 2048-bit public key n (plaintext)
key_index->value.key_e : RSA 2048-bit public key e (plaintext)
key_index->value.dummy : Dummy
key_index->value.key_management_info2 : Key management information

Return Values

TSIP_SUCCESS:	Normal end
TSIP_ERR_RESOURCE_CONFLICT:	A resource conflict occurred because a hardware resource needed by this processing routine was in use by another processing routine.
TSIP_ERR_FAIL	An internal error occurred.

Description

This API updates an RSA 2048-bit public key user key index.

Input data encrypted in the following format with the key update keyring as encrypted_key.

byte	128-bit			
	32-bit	32-bit	32-bit	32-bit
0-255	RSA 2048-bit public key n			
256-271	RSA 2048-bit public key e	0 padding		
272-287	MAC			

<State transition>

The valid pre-run state is *TSIP enabled.*

After the function runs the state transitions to *TSIP Enabled State*.

For the method of generating iv and encrypted_key, and instructions for using key_index, refer to Chapter 7, Key Data Operations.

Reentrant

Not supported

5.43 R_TSIP_UpdateRsa2048PrivateKeyIndex

Format

```
#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	Private key encrypted with key update keyring with MAC appended
key_index	Input/output	RSA 2048-bit private key user key index

Return Values

TSIP_SUCCESS:	Normal end
TSIP_ERR_RESOURCE_CONFLICT:	A resource conflict occurred because a hardware resource needed by this processing routine was in use by another processing routine.
TSIP_ERR_FAIL	An internal error occurred.

Description

This API updates an RSA 2048-bit private key user key index.

Input data encrypted in the following format with the key update keyring as encrypted_key.

Word	128-bit			
	32-bit	32-bit	32-bit	32-bit
0-63	RSA 2048-bit public key n			
64-127	RSA 2048-bit private key d			
128-131	MAC			

<State transition>

The valid pre-run state is *TSIP enabled.*

After the function runs the state transitions to *TSIP Enabled State*.

For the method of generating iv and encrypted_key, and instructions for using key_index, refer to Chapter 7, Key Data Operations.

Reentrant

Not supported

5.44 R_TSIP_RsaesPkcs1024Encrypt

Format

#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);

Parameters

plain	input	plaintext	
plain->pdata			: Specifies pointer to array containing plaintext.
plain->data_length			: Specifies valid data length of plaintext array. data size ≤ public key n size – 11
cipher	input/output	ciphertext	
cipher->pdata			: Specifies pointer to array containing ciphertext.
cipher->data_length			: Inputs ciphertext buffer size. Outputs valid data length after encryption (public key n size).
key_index	input	key data area	: Inputs the 1024-bit RSA public key user key index.

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_RESOURCE_CONFLICT:	A resource conflict occurred because a hardware resource required for processing is in use by another processing routine.
TSIP_ERR_KEY_SET	Incorrect user key index was input.
TSIP_ERR_PARAMETER:	Input data is illegal.

Description

The R_TSIP_RsaesPkcs1024Encrypt() function RSA-encrypts the plaintext input to the first argument, plain, according to RSAES-PKCS1-V1_5. It writes the encryption result to the second argument, cipher.

< State transition >

The state before a valid run is *TSIP Enabled State*.

After the function runs the state is *TSIP Enabled State*.

For instructions for using key_index, refer to “7. Key Data Operations.”

Reentrant

Not supported

5.45 R_TSIP_RsaesPkcs1024Decrypt

Format

```
#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);
```

Parameters

cipher	input	ciphertext	
cipher->pdata			: Specifies pointer to array containing ciphertext.
cipher->data_length			: Specifies valid data length of ciphertext array. (public key n size)
plain	input/output	plaintext	
plain->pdata			: Specifies pointer to array containing plaintext.
plain->data_length			: Inputs plaintext buffer size. The following size is required. Plaintext buffer size >= public key n size -11 Outputs valid data length after decryption.
key_index	input	key data area	: Inputs the 1024-bit RSA private key user key index.

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_RESOURCE_CONFLICT:	A resource conflict occurred because a hardware resource required for processing is in use by another processing routine.
TSIP_ERR_KEY_SET:	Incorrect user key index was input.
TSIP_ERR_PARAMETER:	Input data is illegal.

Description

The R_TSIP_RsaesPkcs1024Decrypt() function RSA-decrypts the ciphertext input to the first argument, cipher, according to RSAES-PKCS1-V1_5. It writes the decryption result to the second argument, plain.

< State transition >

The state before a valid run is *TSIP Enabled State*.

After the function runs the state is *TSIP Enabled State*.

For instructions for using key_index, refer to “7. Key Data Operations.”

Reentrant

Not supported

5.46 R_TSIP_RsaesPkcs2048Encrypt

Format

```
#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);
```

Parameters

plain	input	plaintext	
plain->pdata			: Specifies pointer to array containing plaintext.
plain->data_length			: Specifies valid data length of plain text array. data size ≤ public key n size – 11
cipher	input/output	ciphertext	
cipher->pdata			: Specifies pointer to array that stores ciphertext.
cipher->data_length			: Inputs ciphertext buffer size Outputs valid data length of ciphertext (public key n size).
key_index	input	key data area	: Inputs the 2048-bit RSA public key user key index.

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_RESOURCE_CONFLICT:	A resource conflict occurred because a hardware resource required for processing is in use by another processing routine.
TSIP_ERR_KEY_SET:	Incorrect user key index was input.
TSIP_ERR_PARAMETER:	Input data is illegal.

Description

The R_TSIP_RsaesPkcs2048Encrypt() function RSA-encrypts the plaintext input to the first argument, plain, according to RSAES-PKCS1-V1_5. It writes the encryption result to the second argument, cipher.

< State transition >

The state before a valid run is *TSIP Enabled State*.

After the function runs the state is *TSIP Enabled State*.

For instructions for using key_index, refer to “7. Key Data Operations.”

Reentrant

Not supported

5.47 R_TSIP_RsaesPkcs2048Decrypt

Format

```
#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	ciphertext	
cipher->pdata			: Specifies pointer to array containing ciphertext.
cipher->data_length			: Specifies valid data length of ciphertext array. (public key n size)
plain	input/output	plaintext	
plain->pdata			: Specifies pointer to array containing plaintext
plain->data_length			: Inputs plaintext buffer size. The following size is required. Plaintext buffer size >= public key n size -11 Outputs valid data length after decryption.
key_index	input	key data area	: Inputs the 2048-bit RSA private key user key index.

Return Values

TSIP_SUCCESS :	Normal termination
TSIP_ERR_RESOURCE_CONFLICT:	A resource conflict occurred because a hardware resource required for processing is in use by another processing routine.
TSIP_ERR_KEY_SET	Incorrect user key index was input.
TSIP_ERR_PARAMETER:	Input data is illegal.

Description

The R_TSIP_RsaesPkcs2048Decrypt() function RSA-decrypts the ciphertext input to the first argument, cipher, according to RSAES-PKCS1-V1_5. It writes the decryption result to the second argument, plain.

< State transition >

The state before a valid run is *TSIP Enabled State*.

After the function runs the state is *TSIP Enabled State*.

For instructions for using key_index, refer to “7. Key Data Operations.”

Reentrant

Not supported

5.48 R_TSIP_RsassaPkcs1024SignatureGenerate

Format

```
#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
        )
```

Parameters

message_hash	input	Message or hash value to which to attach signature	
message_hash->pdata			: Specifies pointer to array storing the message or hash value
message_hash->data_length			: Specifies effective data length of the array (Specify only when Message is selected)
message_hash->data_type			: Selects the data type of message_hash Message: 0 Hash value: 1
signature	input/output	Signature text storage destination information	
signature->pdata			: Specifies pointer to array storing the signature text
signature->data_length			: data length
key_index	input	Key data area	: Inputs the 1024-bit RSA private key user key index.
hash_type	input	Hash type	: RSA_HASH_MD5, RSA_HASH_SHA1 or RSA_HASH_SHA256

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_RESOURCE_CONFLICT:	A resource conflict occurred because a hardware resource needed by the processing routine was in use by another processing routine.
TSIP_ERR_KEY_SET:	Invalid user key index was input.
TSIP_ERR_PARAMETER:	Input data is invalid.
Other then the above Return Values	Return value from an internal function that performs a hash operation.

Description

The R_TSIP_RsassaPkcs1024SignatureGenerate() function generates, in accordance with RSASSA-PKCS1-V1_5, a signature from the message text or hash value that is input in the first argument, message_hash, using the private key user key index input to the third argument, key_index, and writes the signature text to the second argument, signature. When a message is specified in the first argument, message_hash->data_type, a hash value is calculated for the message as specified by the fourth argument, hash_type. When specifying a hash value in the first argument, message_hash->data_type, a hash value calculated with a hash algorithm as specified by the fourth argument, hash_type, must be input to message_hash->pdata.

<State transition>

The pre-run state is *TSIP Enabled State*.

After the function runs the state transitions to *TSIP Enabled State*.

Refer to the Section 7 to generate key_index.

Reentrant

Not supported

5.49 R_TSIP_RsassaPkcs1024SignatureVerification

Format

```
#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
)
```

Parameters

signature	input	Signature text information to verify	
signature->pdata			: Specifies pointer to array storing the signature text
signature->data_length			: Specifies effective data length of the array
message_hash	input	Message text or hash value to verify	
message_hash->pdata			: Specifies pointer to array storing the message or hash value
message_hash->data_length			: Specifies effective data length of the array (Specify only when Message is selected)
message_hash->data_type			: Selects the data type of message_hash Message: 0 Hash value: 1
key_index	input	Key data area	: Inputs the 1024-bit RSA public key user key index.
hash_type	input	Hash type	: RSA_HASH_MD5, RSA_HASH_SHA1 or RSA_HASH_SHA256

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_RESOURCE_CONFLICT:	A resource conflict occurred because a hardware resource needed by the processing routine was in use by another processing routine.
TSIP_ERR_KEY_SET:	Invalid user key index was input.
TSIP_ERR_AUTHENTICATION:	Authentication failed
TSIP_ERR_PARAMETER:	Input data is invalid.
Other then the above Return Values	Return value from an internal function that performs a hash operation.

Description

R_TSIP_RsassaPkcs1024SignatureVerification() function verifies, in accordance with RSASSA-PKCS1-V1_5, the signature text input to the first argument signature, and the message text or hash value input to the second argument, message_hash, using the public key user key index input to the third argument, key_index. When a message is specified in the second argument, message_hash->data_type, a hash value is calculated using the public key user key index input to the third argument, key_index, and as specified by the fourth argument, hash_type. When specifying a hash value in the second argument, message_hash->data_type, a hash value calculated with a hash algorithm as specified by the fourth argument, hash_type, must be input to message_hash->pdata.

<State transition>

The pre-run state is *TSIP Enabled State*.

After the function runs the state transitions to *TSIP Enabled State*.

Refer to the Section 7 to generate key_index.

Reentrant

Not supported

5.50 R_TSIP_RsassaPkcs2048SignatureGenerate

Format

```
#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 to which to attach signature	
message_hash->pdata			: Specifies pointer to array storing the message or hash value
message_hash->data_length			: Specifies effective data length of the array (Specify only when Message is selected)
message_hash->data_type			: Selects the data type of message_hash Message: 0 Hash value: 1
signature	input/output	Signature text storage destination information	
signature->pdata			: Specifies pointer to array storing the signature text
signature->data_length			: data length
key_index	input	Key data area	: Inputs the 2048-bit RSA private key user key index.
hash_type	input	Hash type	: RSA_HASH_MD5, RSA_HASH_SHA1 or RSA_HASH_SHA256

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_RESOURCE_CONFLICT:	A resource conflict occurred because a hardware resource needed by the processing routine was in use by another processing routine.
TSIP_ERR_KEY_SET:	Invalid user key index was input.
TSIP_ERR_PARAMETER:	Input data is invalid.
Other then the above Return Values	Return value from an internal function that performs a hash operation.

Description

The R_TSIP_RsassaPkcs2048SignatureGenerate() function generates, in accordance with RSASSA-PKCS1-V1_5, a signature from the message text or hash value that is input in the first argument, message_hash, using the private key user key index input to the third argument, key_index, and writes the signature text to the second argument, signature. When a message is specified in the first argument, message_hash->data_type, a hash value is calculated for the message as specified by the fourth argument, hash_type. When specifying a hash value in the first argument, message_hash->data_type, a hash value calculated with a hash algorithm as specified by the fourth argument, hash_type, must be input to message_hash->pdata.

<State transition>

The pre-run state is *TSIP Enabled State*.

After the function runs the state transitions to *TSIP Enabled State*.

Refer to the Section 7 to generate key_index.

Reentrant

Not supported

5.51 R_TSIP_RsassaPkcs2048SignatureVerification

Format

```
#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
)
```

Parameters

signature	input	Signature text information to verify	
signature->pdata			: Specifies pointer to array storing the signature text
signature->data_length			: Specifies effective data length of the array
message_hash	input	Message or hash value to verify	
message_hash->pdata			: Specifies pointer to array storing the message or hash value
message_hash->data_length			: Specifies effective data length of the array (Specify only when Message is selected)
message_hash->data_type			: Selects the data type of message_hash Message: 0 Hash value: 1
key_index	input	Key data area	: Inputs the 1024-bit RSA public key user key index.
hash_type	input	Hash type	: RSA_HASH_MD5, RSA_HASH_SHA1 or RSA_HASH_SHA256

Return Values

TSIP_SUCCESS :	Normal termination
TSIP_ERR_RESOURCE_CONFLICT:	A resource conflict occurred because a hardware resource needed by the processing routine was in use by another processing routine.
TSIP_ERR_KEY_SET:	Invalid user key index was input.
TSIP_ERR_AUTHENTICATION:	Authentication failed
TSIP_ERR_PARAMETER:	Input data is invalid.
Other then the above Return Values	Return value from an internal function that performs a hash operation.

Description

R_TSIP_RsassaPkcs2048SignatureVerification() function verifies, in accordance with RSASSA-PKCS1-V1_5, the signature text input to the first argument signature, and the message text or hash value input to the second argument, message_hash, using the public key user key index input to the third argument, key_index. When a message is specified in the second argument, message_hash->data_type, a hash value is calculated using the public key user key index input to the third argument, key_index, and as specified by the fourth argument, hash_type. When specifying a hash value in the second argument, message_hash->data_type, a hash value calculated with a hash algorithm as specified by the fourth argument, hash_type, must be input to message_hash->pdata.

<State transition>

The pre-run state is *TSIP Enabled State*.

After the function runs the state transitions to *TSIP Enabled State*.

Refer to the Section 7 to generate key_index.

Reentrant

Not supported

5.52 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	User key index area for AES-128 CMAC operation
sender_private_key_index	Input	Private key generation information used in DH operation The private key d included in the private key generation information is decrypted and used internally in 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-bit MAC
sender_modulus	Input/output	Modular exponentiation result calculated by the sender + MAC 2048-bit modular exponentiation result || 128-bit MAC

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_KEY_SET:	Invalid user key index was input.
TSIP_ERR_RESOURCE_CONFLICT:	A resource conflict occurred because a hardware resource needed by the processing routine was in use by another processing routine.
TSIP_ERR_FAIL:	An internal error occurred.

Description

Performs DH operation using RSA-2048.

Note that the sender is the TSIP and the receiver is the other key exchange party.

<State transition>

The valid pre-run state is *TSIP Enabled State*.

After the function runs the state transitions to *TSIP Enabled State*.

Reentrant

Not supported

5.53 R_TSIP_Sha1HmacGenerateInit

Format

#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);

Parameters

handle	Input/output	SHA-HMAC handler (work area)
key_index	Input	MAC key index area

Return Values

TSIP_SUCCESS:	Normal end
TSIP_ERR_KEY_SET:	An invalid MAC key index was input.
TSIP_ERR_RESOURCE_CONFLICT:	A resource conflict occurred because a hardware resource needed by this processing routine was in use by another processing routine.

Description

The R_TSIP_Sha1HmacGenerateInit() function uses the second argument key_index to prepare for execution of SHA1-HMAC calculation, then writes the result to the first argument handle. When using the TLS cooperation function, use the MAC key index generated by the R_TSIP_TlsGenerateSessionKey() function as key_index. The argument handle is used by the subsequent R_TSIP_Sha1HmacGenerateUpdate() function or R_TSIP_Sha1HmacGenerateFinal() function.

The pre-run state is *TSIP enabled*.

After the function runs the state transitions to *TSIP enabled*.

Reentrant

Not supported

5.54 R_TSIP_Sha1HmacGenerateUpdate

Format

#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);

Parameters

handle	Input/output	SHA-HMAC handle (work area)
message	Input	Message area
message_length	Input	Message length

Return Values

TSIP_SUCCESS:	Normal end
TSIP_ERR_PARAMETER:	An invalid handle was input.
TSIP_ERR_PROHIBIT_FUNCTION:	An invalid function was called.

Description

The R_TSIP_Sha1HmacGenerateUpdate() function uses the handle specified by the first argument handle, calculates a hash value from the second argument message and third argument message_length, then writes the intermediate result to the first argument handle. After message input finishes, call the R_TSIP_Sha1HmacGenerateFinal() function.

The pre-run state is *TSIP enabled*.

After the function runs the state transitions to *TSIP enabled*.

Reentrant

Not supported

5.55 R_TSIP_Sha1HmacGenerateFinal

Format

```
#include "r_tsip_rx_if.h"
e_tsip_err_t R_TSIP_Sha1HmacGenerateFinal
        (tsip_hmac_sha_handle_t *handle, uint8_t *mac);
```

Parameters

handle	Input/output	SHA-HMAC handle (work area)
mac	Input/output	HMAC area (20 bytes)

Return Values

TSIP_SUCCESS:	Normal end
TSIP_ERR_FAIL:	An internal error occurred.
TSIP_ERR_PARAMETER:	An invalid handle was input.
TSIP_ERR_PROHIBIT_FUNCTION:	An invalid function was called.

Description

The R_TSIP_Sha1HmacGenerateFinal() function uses the handle specified by the first argument handle and writes the calculation result to the second argument mac.

The pre-run state is *TSIP enabled*.

After the function runs the state transitions to *TSIP enabled*.

Reentrant

Not supported

5.56 R_TSIP_Sha256HmacGenerateInit

Format

```
#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	Input/output	SHA-HMAC handler (work area)
key_index	Input	MAC key index area

Return Values

TSIP_SUCCESS:	Normal end
TSIP_ERR_KEY_SET:	An invalid MAC key index was input.
TSIP_ERR_RESOURCE_CONFLICT:	A resource conflict occurred because a hardware resource needed by this processing routine was in use by another processing routine.

Description

The R_TSIP_Sha256HmacGenerateInit() function uses the second argument key_index to prepare for execution of SHA256-HMAC calculation, then writes the result to the first argument handle. When using the TLS cooperation function, use the MAC key index generated by the R_TSIP_TlsGenerateSessionKey() function as key_index. The argument handle is used by the subsequent R_TSIP_Sha256HmacGenerateUpdate() function or R_TSIP_Sha256HmacGenerateFinal() function.

The pre-run state is *TSIP enabled*.

After the function runs the state transitions to *TSIP enabled*.

Reentrant

Not supported

5.57 R_TSIP_Sha256HmacGenerateUpdate

Format

#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 handle (work area)
message	Input	Message area
message_length	Input	Message length

Return Values

TSIP_SUCCESS:	Normal end
TSIP_ERR_PARAMETER:	An invalid handle was input.
TSIP_ERR_PROHIBIT_FUNCTION:	An invalid function was called.

Description

The R_TSIP_Sha256HmacGenerateUpdate() function uses the handle specified by the first argument handle, calculates a hash value from the second argument message and third argument message_length, then writes the intermediate result to the first argument handle. After message input finishes, call the R_TSIP_Sha256HmacGenerateFinal() function.

The pre-run state is *TSIP enabled*.

After the function runs the state transitions to *TSIP enabled*.

Reentrant

Not supported

5.58 R_TSIP_Sha256HmacGenerateFinal

Format

```
#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/output	SHA-HMAC handle (work area)
mac	Input/output	HMAC area (32 bytes)

Return Values

TSIP_SUCCESS:	Normal end
TSIP_ERR_FAIL:	An internal error occurred.
TSIP_ERR_PARAMETER:	An invalid handle was input.
TSIP_ERR_PROHIBIT_FUNCTION:	An invalid function was called.

Description

The R_TSIP_Sha256HmacGenerateFinal() function uses the handle specified by the first argument handle and writes the calculation result to the second argument mac.

The pre-run state is *TSIP enabled*.

After the function runs the state transitions to *TSIP enabled*.

Reentrant

Not supported

5.59 R_TSIP_Sha1HmacVerifyInit

Format

```
#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);
```

Parameters

handle	Input/output	SHA-HMAC handler (work area)
key_index	Input	MAC key index area

Return Values

TSIP_SUCCESS:	Normal end
TSIP_ERR_KEY_SET:	An invalid MAC key index was input.
TSIP_ERR_RESOURCE_CONFLICT:	A resource conflict occurred because a hardware resource needed by this processing routine was in use by another processing routine.

Description

The R_TSIP_Sha1HmacVerifyInit() function uses the first argument key_index to prepare for execution of SHA1-HMAC calculation, then writes the result to the first argument handle. When using the TLS cooperation function, use the MAC key index generated by the R_TSIP_TlsGenerateSessionKey() function as key_index. The argument handle is used by the subsequent R_TSIP_Sha1HmacVerifyUpdate() function or R_TSIP_Sha1HmacVerifyFinal() function.

The pre-run state is *TSIP enabled*.

After the function runs the state transitions to *TSIP enabled*.

Reentrant

Not supported

5.60 R_TSIP_Sha1HmacVerifyUpdate

Format

```
#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);
```

Parameters

handle	Input/output	SHA-HMAC handle (work area)
message	Input	Message area
message_length	Input	Message length

Return Values

TSIP_SUCCESS:	Normal end
TSIP_ERR_PARAMETER:	An invalid handle was input.
TSIP_ERR_PROHIBIT_FUNCTION:	An invalid function was called.

Description

The R_TSIP_Sha1HmacVerifyUpdate() function uses the handle specified by the first argument handle, calculates a hash value from the second argument message and third argument message_length, then writes the intermediate result to the first argument handle. After message input finishes, call the R_TSIP_Sha1HmacVerifyFinal() function.

The pre-run state is *TSIP enabled*.

After the function runs the state transitions to *TSIP enabled*.

Reentrant

Not supported

5.61 R_TSIP_Sha1HmacVerifyFinal

Format

#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);

Parameters

handle	Input/output	SHA-HMAC handle (work area)
mac	Input	HMAC area
mac_length	Input	HMAC length

Return Values

TSIP_SUCCESS:	Normal end
TSIP_ERR_FAIL:	An internal error occurred, or verification failed.
TSIP_ERR_PARAMETER:	An invalid handle was input.
TSIP_ERR_PROHIBIT_FUNCTION:	An invalid function was called.

Description

The R_TSIP_Sha1HmacVerifyFinal() function uses the handle specified by the first argument handle and verifies the mac value from the second argument mac and third argument mac_length. Input a value in bytes from 4 to 20 as mac_length.

The pre-run state is *TSIP enabled*.

After the function runs the state transitions to *TSIP enabled*.

Reentrant

Not supported

5.62 R_TSIP_Sha256HmacVerifyInit

Format

```
#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	Input/output	SHA-HMAC handler (work area)
key_index	Input	MAC key index area

Return Values

TSIP_SUCCESS:	Normal end
TSIP_ERR_KEY_SET:	An invalid MAC key index was input.
TSIP_ERR_RESOURCE_CONFLICT:	A resource conflict occurred because a hardware resource needed by this processing routine was in use by another processing routine.

Description

The R_TSIP_Sha256HmacVerifyInit() function uses the second argument key_index to prepare for execution of SHA256-HMAC calculation, then writes the result to the first argument handle. When using the TLS cooperation function, use the MAC key index generated by the R_TSIP_TlsGenerateSessionKey() function as key_index. The argument handle is used by the subsequent R_TSIP_Sha256HmacVerifyUpdate() function or R_TSIP_Sha256HmacVerifyFinal() function.

The pre-run state is *TSIP enabled*.

After the function runs the state transitions to *TSIP enabled*.

Reentrant

Not supported

5.63 R_TSIP_Sha256HmacVerifyUpdate

Format

#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 handle (work area)
message	Input	Message area
message_length	Input	Message length

Return Values

TSIP_SUCCESS:	Normal end
TSIP_ERR_PARAMETER:	An invalid handle was input.
TSIP_ERR_PROHIBIT_FUNCTION:	An invalid function was called.

Description

The R_TSIP_Sha256HmacVerifyUpdate() function uses the handle specified by the first argument handle, calculates a hash value from the second argument message and third argument message_length, then writes the intermediate result to the first argument handle. After message input finishes, call the R_TSIP_Sha256HmacVerifyFinal() function.

The pre-run state is *TSIP enabled*.

After the function runs the state transitions to *TSIP enabled*.

Reentrant

Not supported

5.64 R_TSIP_Sha256HmacVerifyFinal

Format

```
#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/output	SHA-HMAC handle (work area)
mac	Input	HMAC area
mac_length	Input	HMAC length

Return Values

TSIP_SUCCESS:	Normal end
TSIP_ERR_FAIL:	An internal error occurred, or verification failed.
TSIP_ERR_PARAMETER:	An invalid handle was input.
TSIP_ERR_PROHIBIT_FUNCTION:	An invalid function was called.

Description

The R_TSIP_Sha256HmacVerifyFinal() function uses the handle specified by the first argument handle and verifies the mac value from the second argument mac and third argument mac_length. Input a value in bytes from 4 to 32 as mac_length.

The pre-run state is *TSIP enabled*.

After the function runs the state transitions to *TSIP enabled*.

Reentrant

Not supported

5.65 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	Initial vector used when generating encrypted_key
encrypted_key	Input	2048-bit RSA public key encrypted in AES 128 ECB mode
key_index	Input/output	2048-bit RSA public key user key index used by TLS cooperation function

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_RESOURCE_CONFLICT:	A resource conflict occurred because a hardware resource needed by the processing routine was in use by another processing routine.
TSIP_ERR_FAIL:	An internal error occurred.

Description

This API outputs a 2048-bit RSA public key user key index used by the TLS cooperation function.

Input data in the following format as encrypted_key.

byte	128-bit			
	32-bit	32-bit	32-bit	32-bit
0-255	RSA 2048-bit public key n			
256-271	RSA 2048-bit public key e	0 padding		
272-287	MAC			

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

<State transition>

The pre-run state is *TSIP Enabled State*.

After the function runs the state transitions to *TSIP Enabled State*.

For the method of generating encrypted_provisioning_key, iv, and key_index, and instructions for using key_index, refer to Chapter 7, Key Data Operations.

Reentrant

Not supported

5.66 R_TSIP_UpdateTlsRsaPublicKeyIndex

Format

#include “r_tsip_rx_if.h”

e_tsip_err_t R_TSIP_UpdateTlsRsaPublicKeyIndex

(uint8_t *iv, uint8_t *encrypted_key, tsip_tls_ca_certification_public_key_index_t *key_index);

Parameters

iv	Input	Initialization vector when generating encrypted_key
encrypted_key	Input	Public key encrypted key update keyring with MAC appended
key_index	Input/output	RSA 2048-bit public key user key index used by TLS cooperation function

Return Values

TSIP_SUCCESS:	Normal end
TSIP_ERR_RESOURCE_CONFLICT:	A resource conflict occurred because a hardware resource needed by this processing routine was in use by another processing routine.
TSIP_ERR_FAIL:	An internal error occurred.

Description

This API outputs a 2048-bit RSA public key user key index used by the TLS cooperation function.

Input data in the following format as encrypted_key.

byte	128-bit			
	32-bit	32-bit	32-bit	32-bit
0-255	RSA 2048-bit public key n			
256-271	RSA 2048-bit public key e	0 padding		
272-287	MAC			

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

<State transition>

The pre-run state is *TSIP Enabled State*.

After the function runs the state transitions to *TSIP Enabled State*.

For the method of generating iv and encrypted_key, and instructions for using key_index, refer to Chapter 7, Key Data Operations.

Reentrant

Not supported

5.67 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 the 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 originating at the address specified by argument certificate Public key public_key_type 0: n, 2: Qx
public_key_n_end_position	Input	Public key end byte position originating at the address specified by argument certificate Public key public_key_type 0: n, 2: Qx
public_key_e_start_position	Input	Public key start byte position originating at the address specified by argument certificate Public key public_key_type 0: e, 2: Qy
public_key_e_end_position	Input	Public key end byte position originating at the address specified by argument certificate Public key public_key_type 0: e, 2: Qy
signature	Input	Signature data for root CA certificate bundle Input 256 bytes of signature data. The signature format is “RSA2048 PSS with SHA256”.
encrypted_root_public_key	Input/output	Encrypted ECDSA P256 or RSA2048 public key used by R_TSIP_TlsCertificateVerification If the value of public_key_type is 0 then 560 bytes are output, and if 2 then 96 bytes.

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_FAIL:	An internal error occurred.
TSIP_ERR_RESOURCE_CONFLICT:	A resource conflict occurred because a hardware resource needed by the processing routine was in use by another processing routine.

Description

This API verifies the root CA certificate bundle.

<State transition>

The pre-run state is *TSIP Enabled State*.

After the function runs the state transitions to *TSIP Enabled State*.

Reentrant

Not supported

5.68 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 the certificate 0: RSA 2048-bit, 2: ECC P-256, other: reserved
encrypted_input_public_key	Input	RSA-2048 public key output by R_TSIP_TlsRootCertificateVerification or R_TSIP_TlsCertificateVerification Data size public_key_type 0: 140 words, 2: 24 words
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 Data size is 256 byte Algorithm is sha256 With RSA Encryption public_key_type:2 Data size is 64 byte "r(256bit) || s(256bit)" Algorithm is sha256 With ECDSA P-256 Encryption
public_key_n_start_position	Input	Public key start byte position originating at the address specified by argument certificate Public key public_key_type 0: n, 2: Qx
public_key_n_end_position	Input	Public key end byte position originating at the address specified by argument certificate Public key public_key_type 0: n, 2: Qx
public_key_e_start_position	Input	Public key start byte position originating at the address specified by argument certificate Public key public_key_type 0: n, 2: Qx
public_key_e_end_position	Input	Public key end byte position originating at the address specified by argument certificate Public key public_key_type 0: n, 2: Qx
encrypted_output_public_key	Input/output	R_TSIP_TlsCertificateVerification or R_TSIP_TlsEncryptPreMasterSecret Encrypted public key used by WithRsa2048PublicKey Data size public_key_type 0: 140 words, 2: 24 words

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_FAIL:	An internal error occurred.
TSIP_ERR_RESOURCE_CONFLICT:	A resource conflict occurred because a hardware resource needed by the processing routine was in use by another processing routine.

Description

This API verifies the server certificate or intermediate certificate.

<State transition>

The pre-run state is *TSIP Enabled State*.

After the function runs the state transitions to *TSIP Enabled State*.

Reentrant

Not supported

5.69 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	input/output	pre-master secret data with TSIP-specific conversion This data length is 80 bytes.

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_RESOURCE_CONFLICT:	A resource conflict occurred because a hardware resource needed by the processing routine was in use by another processing routine.

Description

This API generates the encrypted PreMasterSecret.

<State transition>

The pre-run state is *TSIP Enabled State*.

After the function runs the state transitions to *TSIP Enabled State*.

Reentrant

Not supported

5.70 R_TSIP_TlsEncryptPreMasterSecretWithRsa2048PublicKey

Format

```
#include "r_tsip_rx_if.h"
e_tsip_err_t R_TSIP_TlsEncryptPreMasterSecretwithRsaPublicKey
        (uint32_t *encrypted_public_key, uint32_t *tsip_pre_master_secret,
         uint8_t *encrypted_pre_master_secret);
```

Parameters

encrypted_public_key	input	Public key data output by R_TSIP_TlsCertificateVerification 140 word size
tsip_pre_master_secret	input	pre-master secret data with TSIP-specific conversion output by R_TSIP_TlsGeneratePreMasterSecret
encrypted_pre_master_secret	input/output	pre-master secret data that was RSA-2048 encrypted using public_key

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_FAIL:	An internal error occurred.
TSIP_ERR_RESOURCE_CONFLICT:	A resource conflict occurred because a hardware resource needed by the processing routine was in use by another processing routine.

Description

This API RSA-2048 encrypts PreMasterSecret using the public key from the input data.

<State transition>

The pre-run state is *TSIP Enabled State*.

After the function runs the state transitions to *TSIP Enabled State*.

Reentrant

Not supported

5.71 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

selet_cipher_suite	input	Selected cipher suite R_TSIP_TLS_RSA_WITH_AES_128_CBC_SHA :0 R_TSIP_TLS_RSA_WITH_AES_256_CBC_SHA :1 R_TSIP_TLS_RSA_WITH_AES_128_CBC_SHA256 :2 R_TSIP_TLS_RSA_WITH_AES_256_CBC_SHA256 :3 R_TSIP_TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256 :4 R_TSIP_TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256 :5 R_TSIP_TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256 :6 R_TSIP_TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 :7
tsip_pre_master_secret	input	Value output by R_TSIP_TlsGeneratePreMasterSecret or R_TSIP_TlsGeneratePreMasterSecretWithEccP256Key
client_random	input	Value of 32-byte random number reported by ClientHello
server_random	input	32-byte random number value reported by ServerHello
tsip_master_secret	input/output	20 words of master secret data with TSIP-specific conversion is output.

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_FAIL:	An internal error occurred.
TSIP_ERR_RESOURCE_CONFLICT:	A resource conflict occurred because a hardware resource needed by the processing routine was in use by another processing routine.

Description

This API is used to generate the encrypted MasterSecret.

<State transition>

The pre-run state is *TSIP Enabled State*.

After the function runs the state transitions to *TSIP Enabled State*.

Reentrant

Not supported

5.72 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_explict,
         tsip_hmac_sha_key_index_t *client_mac_key_index,
         tsip_hmac_sha_key_index_t *server_mac_key_index,
         tsip_aes_key_index_t *client_crypto_key_index,
         tsip_aes_key_index_t *server_crypto_key_index,
         uint8_t *client_iv,
         uint8_t *server_iv);
```

Parameters

select_cipher_suite	input	cipher_suite number selection	
	R_TSIP_TLS_RSA_WITH_AES_128_CBC_SHA		:0
	R_TSIP_TLS_RSA_WITH_AES_256_CBC_SHA		:1
	R_TSIP_TLS_RSA_WITH_AES_128_CBC_SHA256		:2
	R_TSIP_TLS_RSA_WITH_AES_256_CBC_SHA256		:3
	R_TSIP_TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256		:4
	R_TSIP_TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256		:5
	R_TSIP_TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256		:6
	R_TSIP_TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256		:7
tsip_master_secret	input	master secret data with TSIP-specific conversion output by R_TSIP_TlsGenerateMasterSecret	
client_random	input	Value of 32-byte random number reported by ClientHello	
server_random	input	32-byte random number value reported by ServerHello	
nonce_explict	input	Nonce used by cipher suite AES128GCM select_cipher_suite=6-7: 8 bytes	
client_mac_key_index	input/output	MAC key index for client -> server communication select_cipher_suite=0-5: 17 words	
server_mac_key_index	input/output	MAC key index for server -> client communication select_cipher_suite=0-5: 17 words	
client_crypto_key_index	input/output	Common key index for client -> server communication select_cipher_suite=0, 2, 4, 5: 13 words select_cipher_suite=1, 3, 6, 7: 17 words	
server_crypto_key_index	input/output	Common key index for server -> client communication select_cipher_suite=0, 2, 4, 5: 13 words select_cipher_suite=1, 3, 6, 7: 17 words	
client_iv	input/output	Nothing is output.	
server_iv	input/output	Nothing is output.	

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_FAIL:	An internal error occurred.
TSIP_ERR_RESOURCE_CONFLICT:	A resource conflict occurred because a hardware resource needed by the processing routine was in use by another processing routine.

Description

This API is used to output keys for TLS communication.

Nothing is output for the client_iv or server_iv argument. The key information used for communication is retained internally by TSIP.

<State transition>

The pre-run state is *TSIP Enabled State*.

After the function runs the state transitions to *TSIP Enabled State*.

Reentrant

Not supported

5.73 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 type selection 0: R_TSIP_TLS_GENERATE_CLIENT_VERIFY Generate ClientVerifyData. 1: R_TSIP_TLS_GENERATE_SERVER_VERIFY Generate ServerVerifyData
tsip_master_secret	input	master secret data with TSIP-specific conversion output by R_TSIP_TlsGenerateMasterSecret
hand_shake_hash	input	SHA256 HASH value for entire TLS handshake message
verify_data	input/output	VerifyData for Finished message

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_FAIL:	An internal error occurred.
TSIP_ERR_RESOURCE_CONFLICT:	A resource conflict occurred because a hardware resource needed by the processing routine was in use by another processing routine.

Description

This API is used to generate Verify data.

<State transition>

The pre-run state is *TSIP Enabled State*.

After the function runs the state transitions to *TSIP Enabled State*.

Reentrant

Not supported

5.74 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	Random number value (32 bytes) reported by ClientHello
server_random	Input	Random number value (32 bytes) reported by ServerHello
server_ephemeral_ecdh_public_key	Input	Ephemeral ECDH public key (uncompressed format) received by server 0 padding (24-bit) || 04 (8-bit) || Qx (256-bit) || Qy (256-bit)
server_key_exchange_signature	Input	ServerKeyExchange signature data Public key: 256 bytes for RSA 2048-bit 64 bytes for ECDSA P-256 Output encrypted ephemeral ECDH public key
encrypted_public_key	Input	Encrypted public key for signature verification Encrypted public key data output by R_TSIP_CertificateVerification Public key: 140-word size for RSA 2048-bit 24-word size for ECDSA P-256
encrypted_ephemeral_ecdh_public_key	Input/output	Encrypted ephemeral ECDH public key Input to R_TSIP_TlsGeneratePreMasterSecretWithEccP256Key (24-word size).

Return Values

TSIP_SUCCESS:	Normal end
TSIP_ERR_RESOURCE_CONFLICT:	A resource conflict occurred because a hardware resource required by the processing is in use by other processing.
TSIP_ERR_FAIL:	An internal error occurred.

Description

Verifies the ServerKeyExchange signature using the input public key data. If the signature is verified successfully, the ephemeral ECDH public key used by R_TSIP_TlsGeneratePreMasterSecretWithEccP256Key is encrypted and output.

Relevant 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

The pre-run state is *TSIP Enabled State*.

After the function runs the state transitions to *TSIP Enabled State*.

Reentrant

Not supported

5.75 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 output by R_TSIP_TlsServersEphemeralEcdhPublicKey Retrieves
tls_p256_ecc_key_index	Input	Key information output by R_TSIP_GenerateTlsP256EccKeyIndex
tsip_pre_master_secret	Input/output	Outputs 64 bytes of pre-master secret data on which TSIP-specific conversion has been performed.

Return Values

TSIP_SUCCESS:	Normal end
TSIP_ERR_RESOURCE_CONFLICT:	A resource conflict occurred because a hardware resource required by the processing is in use by other processing.
TSIP_ERR_FAIL:	An internal error occurred.

Description

This is an API for generating an encrypted PreMasterSecret using the input data.

Relevant 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

The pre-run state is *TSIP Enabled State*.

After the function runs the state transitions to *TSIP Enabled State*.

Reentrant

Not supported

5.76 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	Key information for generating PreMasterSecret Input to R_TSIP_TlsGeneratePreMasterSecretWithEccP256Key
ephemeral_ecdh_public_key	Output	Ephemeral ECDH public key Public key Qx (256-bit) || public key Qy (256-bit)

Return Values

TSIP_SUCCESS:	Normal end
TSIP_ERR_RESOURCE_CONFLICT:	A resource conflict occurred because a hardware resource required by the processing is in use by other processing.
TSIP_ERR_FAIL:	An internal error occurred.

Description

This is an API for generating a key pair from a random number used by the TLS cooperation function for elliptic curve cryptography over a 256-bit prime field.

The valid pre-run state is *TSIP Enabled State*.

After the function runs the state transitions to *TSIP Enabled State*.

Reentrant

Not supported

5.77 R_TSIP_GenerateTls13P256EccKeyIndex

Format

```
#include "r_tsip_rx_if.h"
e_tsip_err_t R_TSIP_GenerateTls13P256EccKeyIndex(
        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/Output	Handler to indicate the session (work area)
mode	Input	Handshake protocol to use TSIP_TLS13_MODE_FULL_HANDSHAKE : Full Handshake
key_index	Output	Ephemeral ECC secret key key index Input to R_TSIP_Tls13GenerateEcdhSharedSecret
ephemeral_ecdh_public_key	Output	Ephemeral ECDH public key Public key Qx (256-bit) || public key Qy (256-bit)

Return Values

TSIP_SUCCESS:	Normal end
TSIP_ERR_RESOURCE_CONFLICT:	A resource conflict occurred because a hardware resource required by the processing is in use by other processing.
TSIP_ERR_FAIL:	An internal error occurred.

Description

This is an API for generating a key pair from a random number used by the TLS1.3 cooperation function for elliptic curve cryptography over a 256-bit prime field.

The valid pre-run state is *TSIP Enabled State*.

After the function runs the state transitions to *TSIP Enabled State*.

Reentrant

Not supported

5.78 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
ephemeral_ecdh_public_key	Input	Public key provided by the server Qx (256-bit) || public key Qy (256-bit)
key_index	Input	Ephemeral ECC secret key key index Output by R_TSIP_Tls13GenerateEcdhSharedSecret
shared_secret_key_index	Output	Ephemeral SharedSecret key index Input to R_TSIP_Tls13GenerateHandshakeSecret

Return Values

TSIP_SUCCESS:	Normal end
TSIP_ERR_RESOURCE_CONFLICT:	A resource conflict occurred because a hardware resource required by the processing is in use by other processing.
TSIP_ERR_FAIL:	An internal error occurred.
TSIP_ERR_KEY_SET:	Incorrect user key index was input.

Description

This is an API for generating a SharedSecret key index from elliptic curve cryptography over a 256-bit prime field with using public key provided by the server and prepared private key used by the TLS1.3 cooperation function.

Cipher Suite : TLS_AES_128_GCM_SHA256

Key Exchange : ECDHE NIST P-256

The valid pre-run state is *TSIP Enabled State*.

After the function runs the state transitions to *TSIP Enabled State*.

Reentrant

Not supported

5.79 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	Ephemeral SharedSecret key index Output by R_TSIP_Tls13GenerateHandshakeSecret
handshake_secret_key_index	Output	Ephemeral HandshakeSecret key index Input to R_TSIP_Tls13GenerateClientHandshakeTrafficKey, R_TSIP_Tls13GenerateClientHandshakeTrafficKey and R_TSIP_Tls13GenerateMasterSecret

Return Values

TSIP_SUCCESS:	Normal end
TSIP_ERR_RESOURCE_CONFLICT:	A resource conflict occurred because a hardware resource required by the processing is in use by other processing.
TSIP_ERR_KEY_SET:	Incorrect user key index was input.

Description

This is an API for generating a HandshakeSecret key index with using the SharedSecret key index used by the TLS1.3 cooperation function.

The valid pre-run state is *TSIP Enabled State*.

After the function runs the state transitions to *TSIP Enabled State*.

Reentrant

Not supported

5.80 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	Input/Output	Handler to indicate the session (work area)
mode	Input	Handshake protocol to use TSIP_TLS13_MODE_FULL_HANDSHAKE : Full Handshake
handshake_secret_key_index	Input	Ephemeral HandshakeSecret key index Output by R_TSIP_Tls13GenerateHandshakeSecret
digest	Input	Message hash calculated with SHA256 Output by R_TSIP_Sha256Final to calculate concatenated handshake message such as (ClientHello||ServerHello)
server_write_key_index	Output	Ephemeral ServerWriteKey key index Input to R_TSIP_Tls13DecryptInit
server_finished_key_index	Output	Ephemeral ServerFinishedKey key index Input to R_TSIP_Tls13ServerHandshakeVerification

Return Values

TSIP_SUCCESS:	Normal end
TSIP_ERR_RESOURCE_CONFLICT:	A resource conflict occurred because a hardware resource required by the processing is in use by other processing.
TSIP_ERR_KEY_SET:	Incorrect user key index was input.

Description

This is an API for generating a ServerWriteKey key index and a ServerFinishedKey key index with using the HandshakeSecret key index used by the TLS1.3 cooperation function.

The valid pre-run state is *TSIP Enabled State*.

After the function runs the state transitions to *TSIP Enabled State*.

Reentrant

Not supported

5.81 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
server_finished_key_index	Input	Ephemeral ServerFinishedKey key index Output by R_TSIP_Tls13ServerHandshakeVerification
digest	Input	Message hash calculated with SHA256 Output by R_TSIP_Sha256Final to calculate concatenated handshake message such as (ClientHello||ServerHello||EncryptedExtensions ||CertificateRequest||Certificate||CertificateVerify)
server_finished	Input	Finished provided by the server Input to R_TSIP_Tls13DecryptInit
server_finished_key_index	Output	Ephemeral ServerFinishedKey key index Output by R_TSIP_Tls13DecryptFinal
verify_data_index	Output	Result of server handshake verification Input to R_TSIP_Tls13GenerateMasterSecret 8 words (32 bytes)

Return Values

TSIP_SUCCESS:	Normal end
TSIP_ERR_RESOURCE_CONFLICT:	A resource conflict occurred because a hardware resource required by the processing is in use by other processing.
TSIP_ERR_FAIL:	An internal error occurred.
TSIP_ERR_KEY_SET:	Incorrect user key index was input.
TSIP_ERR_VERIFICATION_FAIL:	Handshake verification failed.

Description

This is an API for verifying the Finished provided from the server used by the TLS1.3 cooperation function.

The valid pre-run state is *TSIP Enabled State*.

After the function runs the state transitions to *TSIP Enabled State*.

Reentrant

Not supported

5.82 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	Input/Output	Handler to indicate the session (work area)
mode	Input	Handshake protocol to use TSIP_TLS13_MODE_FULL_HANDSHAKE : Full Handshake
handshake_secret_key_index	Input	Ephemeral HandshakeSecret key index Output by R_TSIP_Tls13GenerateHandshakeSecret
digest	Input	Message hash calculated with SHA256 Output by R_TSIP_Sha256Final to calculate concatenated handshake message such as (ClientHello||ServerHello)
client_write_key_index	Output	Ephemeral ClientWriteKey key index Input to R_TSIP_Tls13EncryptInit
client_finished_key_index	Output	Ephemeral ClientFinishedKey key index Input to R_TSIP_Sha256HmacGenerateInit

Return Values

TSIP_SUCCESS:	Normal end
TSIP_ERR_RESOURCE_CONFLICT:	A resource conflict occurred because a hardware resource required by the processing is in use by other processing.
TSIP_ERR_KEY_SET:	Incorrect user key index was input.

Description

This is an API for generating a ClientWriteKey key index and a ClientFinishedKey key index with using the HandshakeSecret key index used by the TLS1.3 cooperation function.

The valid pre-run state is *TSIP Enabled State*.

After the function runs the state transitions to *TSIP Enabled State*.

Reentrant

Not supported

5.83 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	Input/Output	Handler to indicate the session (work area)
mode	Input	Handshake protocol to use TSIP_TLS13_MODE_FULL_HANDSHAKE : Full Handshake
handshake_secret_key_index	Input	Ephemeral HandshakeSecret key index Output by R_TSIP_Tls13GenerateHandshakeSecret
verify_data_index	Input	Result of server handshake verification Output by R_TSIP_Tls13GenerateMasterSecret
master_secret_key_index	Output	Ephemeral MasterSecret key index Input to R_TSIP_Tls13GenerateApplicationTrafficKey

Return Values

TSIP_SUCCESS:	Normal end
TSIP_ERR_RESOURCE_CONFLICT:	A resource conflict occurred because a hardware resource required by the processing is in use by other processing.
TSIP_ERR_FAIL:	An internal error occurred.
TSIP_ERR_KEY_SET:	Incorrect user key index was input.

Description

This is an API for generating a MasterSecret key index with using the HandshakeSecret key index used by the TLS1.3 cooperation function.

The valid pre-run state is *TSIP Enabled State*.

After the function runs the state transitions to *TSIP Enabled State*.

Reentrant

Not supported

5.84 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	Handler to indicate the session (work area)
mode	Input	Handshake protocol to use TSIP_TLS13_MODE_FULL_HANDSHAKE : Full Handshake
master_secret_key_index	Input	Ephemeral MasterSecret key index Output by R_TSIP_Tls13GenerateMasterSecret
digest	Input	Message hash calculated with SHA256 Output by R_TSIP_Sha256Final to calculate concatenated handshake message such as (ClientHello||ServerHello||EncryptedExtensions ||CertificateRequest||Certificate||CertificateVerify ||ServerFinished)
server_app_secret_key_index	Output	Ephemeral ServerApplicationTrafficSecret key index Input to R_TSIP_Tls13UpdateApplicationTrafficKey
client_app_secret_key_index	Output	Ephemeral ClientApplicationTrafficSecret key index Input to R_TSIP_Tls13UpdateApplicationTrafficKey
server_write_key_index	Output	Ephemeral ServerWriteKey key index Input to R_TSIP_Tls13DecryptInit
client_write_key_index	Output	Ephemeral ClientWriteKey key index Input to R_TSIP_Tls13EncryptInit

Return Values

TSIP_SUCCESS:	Normal end
TSIP_ERR_RESOURCE_CONFLICT:	A resource conflict occurred because a hardware resource required by the processing is in use by other processing.
TSIP_ERR_KEY_SET:	Incorrect user key index was input.

Description

This is an API for generating a ServerWriteKey key index, a ClientWriteKey key index and each ApplicationTrafficSecret key indexes with using the MasterSecret key index used by the TLS1.3 cooperation function.

The valid pre-run state is *TSIP Enabled State*.

After the function runs the state transitions to *TSIP Enabled State*.

Reentrant

Not supported

5.85 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	Handler to indicate the session (work area)
mode	Input	Handshake protocol to use TSIP_TLS13_MODE_FULL_HANDSHAKE : Full Handshake
key_type	Input	Key type to update TSIP_TLS13_UPDATE_SERVER_KEY : Server ApplicationTrafficSecret/WriteKey TSIP_TLS13_UPDATE_CLIENT_KEY : Client ApplicationTrafficSecret/WriteKey
input_app_secret_key_index	Input	Ephemeral Server/Client ApplicationTrafficSecret key index Output by R_TSIP_Tls13GenerateApplicationTrafficKey or R_TSIP_Tls13UpdateApplicationTrafficKey
output_app_secret_key_index	Output	Ephemeral Server/Client ApplicationTrafficSecret key index Input to R_TSIP_Tls13UpdateApplicationTrafficKey
app_write_key_index	Output	Ephemeral Server/ClientWriteKey key index Input to R_TSIP_Tls13EncryptInit orx R_TSIP_Tls13DecryptInit

Return Values

TSIP_SUCCESS:	Normal end
TSIP_ERR_FAIL:	An internal error occurred.
TSIP_ERR_RESOURCE_CONFLICT:	A resource conflict occurred because a hardware resource required by the processing is in use by other processing.
TSIP_ERR_KEY_SET:	Incorrect user key index was input.
TSIP_ERR_PARAMETER:	Input data is illegal.

Description

This is an API for updating an ApplicationTrafficSecret key index and corresponding WriteKey key index with using the previous ApplicationTrafficSecret key index used by the TLS1.3 cooperation function.

The valid pre-run state is *TSIP Enabled State*.

After the function runs the state transitions to *TSIP Enabled State*.

Reentrant

Not supported

5.86 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 *client_write_key_index,
        uint32_t payload_length
)
```

Parameters

handle	Input/Output	Handler to indicate the session (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
cipher_suite	Input	Cipher suite TSIP_TLS13_CIPHER_SUITE_AES_128_GCM_SHA256 : TLS_AES_128_GCM_SHA256
client_write_key_index	Input	Ephemeral ClientWriteKey key index
payload_length	Input	Payload length

Return Values

TSIP_SUCCESS:	Normal end
TSIP_ERR_FAIL:	An internal error occurred.
TSIP_ERR_RESOURCE_CONFLICT:	A resource conflict occurred because a hardware resource required by the processing is in use by other processing.
TSIP_ERR_KEY_SET:	Incorrect user key index was input.

Description

The R_TSIP_TLS13EncryptInit() function performs preparations for the execution of an encrypt calculation used by the TLS1.3 cooperation function, and writes the result to the first argument, handle. The value of handle is used as an argument in the subsequent R_TSIP_Tls13EncryptUpdate() function and R_TSIP_Tls13EncryptFinal() function.

The valid pre-run state is *TSIP Enabled State*.

After the function runs the state transitions to *TSIP Enabled State*.

Reentrant

Not supported

5.87 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	Handler to indicate the session (work area)
plain	Input	Plaintext data area
cipher	Output	Ciphertext data area
plain_length	Input	Plaintext data length

Return Values

TSIP_SUCCESS:	Normal end
TSIP_ERR_PROHIBIT_FUNCTION:	An invalid function was called.
TSIP_ERR_PARAMETER:	Input data is illegal.

Description

The R_TSIP_Tls13EncryptUpdate() function encrypts the plaintext specified in the second argument, plain, using the values specified for client_write_key_index in R_TSIP_Tls13EncryptInit(). Inside this function, the data that is input by the user is buffered until the input values of plain exceed 16 bytes. After the input data from plain reaches 16 bytes or more, the encryption result is output to the ciphertext data area specified in the third argument, cipher. The length of the plain to input is specified in the fourth argument, payload_length. For this, specify not the total byte count for the plain input data, but rather the data length to input when the user calls this function. If the input value plain is not divisible by 16 bytes, that will be padded inside the function. Specify areas for plain and cipher that do not overlap. For plain and cipher, specify RAM addresses that are multiples of 4.

The valid pre-run state is *TSIP Enabled State*.

After the function runs the state transitions to *TSIP Enabled State*.

Reentrant

Not supported

5.88 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	Handler to indicate the session (work area)
cipher	Output	Ciphertext data area
cipher_length	Output	Ciphertext data length

Return Values

TSIP_SUCCESS:	Normal end
TSIP_ERR_FAIL:	An internal error occurred.
TSIP_ERR_PROHIBIT_FUNCTION:	An invalid function was called.
TSIP_ERR_PARAMETER:	Input data is illegal.

Description

If there is 16-byte fractional data indicated by the total data length of the value of plain that was input by R_TSIP_Tls13EncryptUpdate(), the R_TSIP_Tls13EncryptFinal() function will output the result of encrypting that fractional data to the ciphertext data area specified in the second argument, cipher. Here, the portion that does not reach 16 bytes will be padded with zeros. For cipher, specify RAM address that are multiples of 4.

The valid pre-run state is *TSIP Enabled State*.

After the function runs the state transitions to *TSIP Enabled State*.

Reentrant

Not supported

5.89 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 *server_write_key_index,
        uint32_t payload_length
)
```

Parameters

handle	Input/Output	Handler to indicate the session (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
cipher_suite	Input	Cipher suite TSIP_TLS13_CIPHER_SUITE_AES_128_GCM_SHA256 : TLS_AES_128_GCM_SHA256
server_write_key_index	Input	Ephemeral ServerWriteKey key index
payload_length	Input	Payload length

Return Values

TSIP_SUCCESS:	Normal end
TSIP_ERR_FAIL:	An internal error occurred.
TSIP_ERR_RESOURCE_CONFLICT:	A resource conflict occurred because a hardware resource required by the processing is in use by other processing.
TSIP_ERR_KEY_SET:	Incorrect user key index was input.

Description

The R_TSIP_TLS13DecryptInit() function performs preparations for the execution of a decrypt calculation used by the TLS1.3 cooperation function, and writes the result to the first argument, handle. The value of handle is used as an argument in the subsequent R_TSIP_Tls13DecryptUpdate() function and R_TSIP_Tls13DecryptFinal() function.

The valid pre-run state is *TSIP Enabled State*.

After the function runs the state transitions to *TSIP Enabled State*.

Reentrant

Not supported

5.90 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	Handler to indicate the session (work area)
cipher	Input	Ciphertext data area
plain	Output	Plaintext data area
cipher_length	Input	Ciphertext data length

Return Values

TSIP_SUCCESS:	Normal end
TSIP_ERR_PROHIBIT_FUNCTION:	An invalid function was called.
TSIP_ERR_PARAMETER:	Input data is illegal.

Description

The R_TSIP_Tls13DecryptUpdate() function decrypts the ciphertext specified in the second argument, cipher, using the values specified for server_write_key_index in R_TSIP_Tls13DecryptInit(). Inside this function, the data that is input by the user is buffered until the input values of cipher exceed 16 bytes. After the input data from cipher reaches 16 bytes or more, the decryption result is output to the plaintext data area specified in the third argument, plain. The length of the cipher to input is specified in the fourth argument, cipher_length. For this, specify not the total byte count for the cipher input data, but rather the data length to input when the user calls this function. If the input value cipher is not divisible by 16 bytes, that will be padded inside the function. Specify areas for cipher and plain that do not overlap. For cipher and plain, specify RAM addresses that are multiples of 4.

The valid pre-run state is *TSIP Enabled State*.

After the function runs the state transitions to *TSIP Enabled State*.

Reentrant

Not supported

5.91 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	Handler to indicate the session (work area)
plain	Output	Plaintext data area
plain_length	Output	Plaintext data length

Return Values

TSIP_SUCCESS:	Normal end
TSIP_ERR_FAIL:	An internal error occurred.
TSIP_ERR_PROHIBIT_FUNCTION:	An invalid function was called.
TSIP_ERR_PARAMETER:	Input data is illegal.
TSIP_ERR_PROHIBIT_FUNCTION:	An invalid function was called.

Description

If there is 16-byte fractional data indicated by the total data length of the value of cipher that was input by R_TSIP_Tls13DecryptUpdate(), the R_TSIP_Tls13DecryptFinal() function will output the result of decrypting that fractional data to the plaintext data area specified in the second argument, plain. Here, the portion that does not reach 16 bytes will be padded with zeros. For plain, specify RAM address that are multiples of 4.

The valid pre-run state is *TSIP Enabled State*.

After the function runs the state transitions to *TSIP Enabled State*.

Reentrant

Not supported

5.92 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	ECC P-256 private key user key index Output by R_TSIP_GenerateEccP256PrivateKeyIndex with casting uint32_t *
signature_scheme	Input	Signature Algorithm TSIP_TLS13_SIGNATURE_SCHEME_ECDSA_SECP256R1_SHA256 : ecdsa_secp256r1_sha256
digest	Input	Message hash calculated with SHA256 Output by R_TSIP_Sha256Final to calculate concatenated handshake message such as (ClientHello||ServerHello||EncryptedExtensions ||CertificateRequest||Certificate||CertificateVerify ||ServerFinished||Certificate)
certificate_verify	Output	CertificateVerify Output format is described in RFC8446 section 4.4.3. Enough area to store data must be allocated.
certificate_verify_len	Output	Byte length of certificate_verify

Return Values

TSIP_SUCCESS:	Normal end
TSIP_ERR_RESOURCE_CONFLICT:	A resource conflict occurred because a hardware resource required by the processing is in use by other processing.
TSIP_ERR_FAIL:	An internal error occurred.
TSIP_ERR_KEY_SET:	Incorrect user key index was input.
TSIP_ERR_PARAMETER:	Input data is illegal.

Description

This is an API for generating the CertifucateVerify sending to the server used by the TLS1.3 cooperation function. Supporting signature algorithm is ECDSA P-256 and hash algorithm is SHA256.

The valid pre-run state is *TSIP Enabled State*.

After the function runs the state transitions to *TSIP Enabled State*.

Reentrant

Not supported

5.93 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	ECC P-256 public key user key index Output by R_TSIP_GenerateEccP256PublicKeyIndex with casting uint32_t *
signature_scheme	Input	Signature Algorithm TSIP_TLS13_SIGNATURE_SCHEME_ECDSA_SECP256R1_SHA256 : ecdsa_secp256r1_sha256
digest	Input	Message hash calculated with SHA256 Output by R_TSIP_Sha256Final to calculate concatenated handshake message such as (ClientHello||ServerHello||EncryptedExtensions ||CertificateRequest||Certificate)
certificate_verify	Input	CertificateVerify Input format must be described in RFC8446 section 4.4.3.
certificate_verify_len	Input	Byte length of certificate_verify

Return Values

TSIP_SUCCESS:	Normal end
TSIP_ERR_RESOURCE_CONFLICT:	A resource conflict occurred because a hardware resource required by the processing is in use by other processing.
TSIP_ERR_FAIL:	An internal error occurred, or signature verification failed.
TSIP_ERR_KEY_SET:	Incorrect user key index was input.
TSIP_ERR_PARAMETER:	Input data is illegal.

Description

This is an API for verifying the CertifucateVerify received from the server used by the TLS1.3 cooperation function. Supporting signature algorithm is ECDSA P-256 and hash algorithm is SHA256.

The valid pre-run state is *TSIP Enabled State*.

After the function runs the state transitions to *TSIP Enabled State*.

Reentrant

Not supported

5.94 R_TSIP_GenerateEccP192PublicKeyIndex

Format

```
#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
)
```

Parameters

encrypted_provisioning_key	Input	Provisioning key wrapped by the DLM server
iv	Input	Initial vector used when generating encrypted_key
encrypted_key	Input	Encrypted ECC P-192 public key with MAC value added
key_index	Output	ECC P-192 public key user key index
key_index->value.key_management_info		: Key management information
key_index->value.key_q		: ECC P-192 public key Q (plaintext)

Return Values

TSIP_SUCCESS:	Normal end
TSIP_ERR_RESOURCE_CONFLICT:	A resource conflict occurred because a hardware resource required by the processing is in use by other processing.
TSIP_ERR_FAIL:	An internal error occurred.

Description

This is an API for outputting an ECC P-192 public key user key index.

For encrypted_key, input data in the following format that has been encrypted with the provisioning 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 public key Qx (continuation)			
32-47	0 padding		ECC P-192 public key Qy	
48-63	ECC P-192 public key Qy (continuation)			
64-79	MAC			

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

<State transition>

The valid pre-run state is *TSIP Enabled State*.

After the function runs the state transitions to *TSIP Enabled State*.

For the method of generating encrypted_provisioning_key, iv, and encrypted_key, and for how to use key_index, refer to section 7, Key Data Operations.

Reentrant

Not supported

5.95 R_TSIP_GenerateEccP224PublicKeyIndex

Format

```
#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
)
```

Parameters

encrypted_provisioning_key	Input	Provisioning key wrapped by the DLM server
iv	Input	Initial vector used when generating encrypted_key
encrypted_key	Input	Encrypted ECC P-224 public key with MAC value added
key_index	Output	ECC P-224 public key user key index
key_index->value.key_management_info		: Key management information
key_index->value.key_q		: ECC P-224 public key Q (plaintext)

Return Values

TSIP_SUCCESS:	Normal end
TSIP_ERR_RESOURCE_CONFLICT:	A resource conflict occurred because a hardware resource required by the processing is in use by other processing.
TSIP_ERR_FAIL:	An internal error occurred.

Description

This is an API for outputting an ECC P-224 public key user key index.

For encrypted_key, input data in the following format that has been encrypted with the provisioning 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 public key Qx (continuation)			
32-47	0 padding	ECC P-224 public key Qy		
48-63	ECC P-224 public key Qy (continuation)			
64-79	MAC			

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

<State transition>

The valid pre-run state is *TSIP Enabled State*.

After the function runs the state transitions to *TSIP Enabled State*.

For the method of generating encrypted_provisioning_key, iv, and encrypted_key, and for how to use key_index, refer to section 7, Key Data Operations.

Reentrant

Not supported

5.96 R_TSIP_GenerateEccP256PublicKeyIndex

Format

```
#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
)
```

Parameters

encrypted_provisioning_key	Input	Provisioning key wrapped by the DLM server
iv	Input	Initial vector used when generating encrypted_key
encrypted_key	Input	Encrypted ECC P-256 public key with MAC value added
key_index	Output	ECC P-256 public key user key index
key_index->value.key_management_info		: Key management information
key_index->value.key_q		: ECC P-256 public key Q (plaintext)

Return Values

TSIP_SUCCESS:	Normal end
TSIP_ERR_RESOURCE_CONFLICT:	A resource conflict occurred because a hardware resource required by the processing is in use by other processing.
TSIP_ERR_FAIL:	An internal error occurred.

Description

This is an API for outputting an ECC P-256 public key user key index.

For encrypted_key, input data in the following format that has been encrypted with the provisioning key.

Bytes	128 bits			
	32 bits	32 bits	32 bits	32 bits
0-31	ECC P-256 public key Qx			
32-63	ECC P-256 public key Qy			
64-79	MAC			

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

<State transition>

The valid pre-run state is *TSIP Enabled State*.

After the function runs the state transitions to *TSIP Enabled State*.

For the method of generating encrypted_provisioning_key, iv, and encrypted_key, and for how to use key_index, refer to section 7, Key Data Operations.

Reentrant

Not supported

5.97 R_TSIP_GenerateEccP384PublicKeyIndex

Format

```
#include "r_tsip_rx_if.h"
e_tsip_err_t R_TSIP_GenerateEccP384PublicKeyIndex(
        uint8_t *encrypted_provisioning_key,
        uint8_t *iv,
        uint8_t *encrypted_key,
        tsip_ecc_public_key_index_t *key_index
)
```

Parameters

encrypted_provisioning_key	Input	Provisioning key wrapped by the DLM server
iv	Input	Initial vector used when generating encrypted_key
encrypted_key	Input	Encrypted ECC P-384 public key with MAC value added
key_index	Output	ECC P-384 public key user key index
key_index->value.key_management_info		: Key management information
key_index->value.key_q		: ECC P-384 public key Q (plaintext)

Return Values

TSIP_SUCCESS:	Normal end
TSIP_ERR_RESOURCE_CONFLICT:	A resource conflict occurred because a hardware resource required by the processing is in use by other processing.
TSIP_ERR_FAIL:	An internal error occurred.

Description

This is an API for outputting an ECC P-384 public key user key index.

For encrypted_key, input data in the following format that has been encrypted with the provisioning key.

Bytes	128 bits			
	32 bits	32 bits	32 bits	32 bits
0-47	ECC P-384 public key Qx			
48-95	ECC P-384 public key Qy			
96-111	MAC			

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

<State transition>

The valid pre-run state is *TSIP Enabled State*.

After the function runs the state transitions to *TSIP Enabled State*.

For the method of generating encrypted_provisioning_key, iv, and encrypted_key, and for how to use key_index, refer to section 7, Key Data Operations.

Reentrant

Not supported

5.98 R_TSIP_GenerateEccP192PrivateKeyIndex

Format

```
#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
)
```

Parameters

encrypted_provisioning_key	Input	Provisioning key wrapped by the DLM server
iv	Input	Initial vector used when generating encrypted_key
encrypted_key	Input	Encrypted ECC P-192 private key with MAC value added
key_index	Output	ECC P-192 private key user key index

Return Values

TSIP_SUCCESS:	Normal end
TSIP_ERR_RESOURCE_CONFLICT:	A resource conflict occurred because a hardware resource required by the processing is in use by other processing.
TSIP_ERR_FAIL:	An internal error occurred.

Description

This is an API for outputting an ECC P-192 private key user key index.

For encrypted_key, input data in the following format that has been encrypted with the provisioning key.

Bytes	128 bits			
	32 bits	32 bits	32 bits	32 bits
0-15	0 padding		ECC P-192 private key	
16-31	ECC P-192 private key (continuation)			
32-47	MAC			

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

<State transition>

The valid pre-run state is *TSIP Enabled State*.

After the function runs the state transitions to *TSIP Enabled State*.

For the method of generating encrypted_provisioning_key, iv, and encrypted_key, and for how to use key_index, refer to section 7, Key Data Operations.

Reentrant

Not supported

5.99 R_TSIP_GenerateEccP224PrivateKeyIndex

Format

```
#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
)
```

Parameters

encrypted_provisioning_key	Input	Provisioning key wrapped by the DLM server
iv	Input	Initial vector used when generating encrypted_key
encrypted_key	Input	Encrypted ECC P-224 private key with MAC value added
key_index	Output	ECC P-224 private key user key index

Return Values

TSIP_SUCCESS:	Normal end
TSIP_ERR_RESOURCE_CONFLICT:	A resource conflict occurred because a hardware resource required by the processing is in use by other processing.
TSIP_ERR_FAIL:	An internal error occurred.

Description

This is an API for outputting an ECC P-224 private key user key index.

For encrypted_key, input data in the following format that has been encrypted with the provisioning key.

Bytes	128 bits			
	32 bits	32 bits	32 bits	32 bits
0-15	0 padding	ECC P-224 private key		
16-31	ECC P-224 private key (continuation)			
32-47	MAC			

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

<State transition>

The valid pre-run state is *TSIP Enabled State*.

After the function runs the state transitions to *TSIP Enabled State*.

For the method of generating encrypted_provisioning_key, iv, and encrypted_key, and for how to use key_index, refer to section 7, Key Data Operations.

Reentrant

Not supported

5.100 R_TSIP_GenerateEccP256PrivateKeyIndex

Format

```
#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
)
```

Parameters

encrypted_provisioning_key	Input	Provisioning key wrapped by the DLM server
iv	Input	Initial vector used when generating encrypted_key
encrypted_key	Input	Encrypted ECC P-256 private key with MAC value added
key_index	Output	ECC P-256 private key user key index

Return Values

TSIP_SUCCESS:	Normal end
TSIP_ERR_RESOURCE_CONFLICT:	A resource conflict occurred because a hardware resource required by the processing is in use by other processing.
TSIP_ERR_FAIL:	An internal error occurred.

Description

This is an API for outputting an ECC P-256 private key user key index.

For encrypted_key, input data in the following format that has been encrypted with the provisioning key.

Bytes	128 bits			
	32 bits	32 bits	32 bits	32 bits
0-31	ECC P-256 private key			
32-47	MAC			

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

<State transition>

The valid pre-run state is *TSIP Enabled State*.

After the function runs the state transitions to *TSIP Enabled State*.

For the method of generating encrypted_provisioning_key, iv, and encrypted_key, and for how to use key_index, refer to section 7, Key Data Operations.

Reentrant

Not supported

5.101 R_TSIP_GenerateEccP384PrivateKeyIndex

Format

```
#include "r_tsip_rx_if.h"
e_tsip_err_t R_TSIP_GenerateEccP384PrivateKeyIndex(
        uint8_t *encrypted_provisioning_key,
        uint8_t *iv,
        uint8_t *encrypted_key,
        tsip_ecc_private_key_index_t *key_index
)
```

Parameters

encrypted_provisioning_key	Input	Provisioning key wrapped by the DLM server
iv	Input	Initial vector used when generating encrypted_key
encrypted_key	Input	Encrypted ECC P-384 private key with MAC value added
key_index	Output	ECC P-384 private key user key index

Return Values

TSIP_SUCCESS:	Normal end
TSIP_ERR_RESOURCE_CONFLICT:	A resource conflict occurred because a hardware resource required by the processing is in use by other processing.
TSIP_ERR_FAIL:	An internal error occurred.

Description

This is an API for outputting an ECC P-384 private key user key index.

For encrypted_key, input data in the following format that has been encrypted with the provisioning key.

Bytes	128 bits			
	32 bits	32 bits	32 bits	32 bits
0-37	ECC P-384 private key			
48-63	MAC			

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

<State transition>

The valid pre-run state is *TSIP Enabled State*.

After the function runs the state transitions to *TSIP Enabled State*.

For the method of generating encrypted_provisioning_key, iv, and encrypted_key, and for how to use key_index, refer to section 7, Key Data Operations.

Reentrant

Not supported

5.102 R_TSIP_GenerateEccP192RandomKeyIndex

Format

```
#include "r_tsip_rx_if.h"
e_tsip_err_t R_TSIP_GenerateEccP192RandomKeyIndex(
        tsip_ecc_key_pair_index_t *key_pair_index
)
```

Parameters

key_pair_index Output User key indexes for ECC P-192 public key and private key pair

key_pair_index->public : ECC P-192 public key user key index
key_pair_index->public.value.key_management_info : Key management information
key_pair_index->public.value.key_q : ECC P-192 public key Q (plaintext)
key_pair_index->private : ECC P-192 private key user key index

Return Values

TSIP_SUCCESS:	Normal end
TSIP_ERR_RESOURCE_CONFLICT:	A resource conflict occurred because a hardware resource required by the processing is in use by other processing.
TSIP_ERR_FAIL:	An internal error occurred.

Description

This is an API for outputting user key indexes for an ECC P-192 public key and private key pair. This API generates a user key from a random number value internally within the TSIP. There is therefore no need to input a user key. It is possible to prevent dead copying of data by using the user key index output by this API to encrypt the data. The public key index is generated in key_pair_index->public, and the private key user key index is generated in key_pair_index->private.

<State transition>

The valid pre-run state is *TSIP Enabled State*.

After the function runs the state transitions to *TSIP Enabled State*.

For the method of using key_pair_index->public and key_pair_index->private, refer to Chapter 7, Key Data Operations.

key_pair_index->public is the same operation as the public key user key index output from R_TSIP_GenerateEccP192PublicKeyIndex(), and Key_pair_index->private is the same operation as the private key user key index output from R_TSIP_GenerateEccP192PrivateKeyIndex().

Reentrant

Not supported

5.103 R_TSIP_GenerateEccP224RandomKeyIndex

Format

```
#include "r_tsip_rx_if.h"
e_tsip_err_t R_TSIP_GenerateEccP224RandomKeyIndex(
        tsip_ecc_key_pair_index_t *key_pair_index
)
```

Parameters

key_pair_index	Output	User key indexes for ECC P-224 public key and private key pair

key_pair_index->public : ECC P-224 public key user key index
key_pair_index->public.value.key_management_info : Key management information
key_pair_index->public.value.key_q : ECC P-224 public key Q (plaintext)
key_pair_index->private : ECC P-224 private key user key index

Return Values

TSIP_SUCCESS:	Normal end
TSIP_ERR_RESOURCE_CONFLICT:	A resource conflict occurred because a hardware resource required by the processing is in use by other processing.
TSIP_ERR_FAIL:	An internal error occurred.

Description

This is an API for outputting user key indexes for an ECC P-224 public key and private key pair. This API generates a user key from a random number value internally within the TSIP. There is therefore no need to input a user key. It is possible to prevent dead copying of data by using the user key index output by this API to encrypt the data. The public key user key index is generated in key_pair_index->public, and the private key user key index is generated in key_pair_index->private.

<State transition>

The valid pre-run state is *TSIP Enabled State*.

After the function runs the state transitions to *TSIP Enabled State*.

For the method of using key_pair_index->public and key_pair_index->private, refer to Chapter 7, Key Data Operations.

key_pair_index->public is the same operation as the public key user key index output from R_TSIP_GenerateEccP224PublicKeyIndex(), and Key_pair_index->private is the same operation as the private key user key index output from R_TSIP_GenerateEccP224PrivateKeyIndex().

Reentrant

Not supported

5.104 R_TSIP_GenerateEccP256RandomKeyIndex

Format

```
#include "r_tsip_rx_if.h"
e_tsip_err_t R_TSIP_GenerateEccP256RandomKeyIndex(
        tsip_ecc_key_pair_index_t *key_pair_index
)
```

Parameters

key_pair_index	Output	User key indexes for ECC P-256 public key and private key pair

key_pair_index->public : ECC P-256 public key user key index
key_pair_index->public.value.key_management_info : Key management information
key_pair_index->public.value.key_q : ECC P-256 public key Q (plaintext)
key_pair_index->private : ECC P-256 private key user key index

Return Values

TSIP_SUCCESS:	Normal end
TSIP_ERR_RESOURCE_CONFLICT:	A resource conflict occurred because a hardware resource required by the processing is in use by other processing.
TSIP_ERR_FAIL:	An internal error occurred.

Description

This is an API for outputting user key indexes for an ECC P-256 public key and private key pair. This API generates a user key from a random number value internally within the TSIP. There is therefore no need to input a user key. It is possible to prevent dead copying of data by using the user key index output by this API to encrypt the data. The public key user key index is generated in key_pair_index->public, and the private key user key index is generated in key_pair_index->private.

<State transition>

The valid pre-run state is *TSIP Enabled State*.

After the function runs the state transitions to *TSIP Enabled State*.

For the method of using key_pair_index->public and key_pair_index->private, refer to Chapter 7, Key Data Operations.

key_pair_index->public is the same operation as the public key user key index output from R_TSIP_GenerateEccP256PublicKeyIndex(), and Key_pair_index->private is the same operation as the private key user key index output from R_TSIP_GenerateEccP256PrivateKeyIndex().

Reentrant

Not supported

5.105 R_TSIP_GenerateEccP384RandomKeyIndex

Format

```
#include "r_tsip_rx_if.h"
e_tsip_err_t R_TSIP_GenerateEccP384RandomKeyIndex(
        tsip_ecc_key_pair_index_t *key_pair_index
)
```

Parameters

key_pair_index	Output	User key indexes for ECC P-384 public key and private key pair

key_pair_index->public : ECC P-384 public key user key index
key_pair_index->public.value.key_management_info : Key management information
key_pair_index->public.value.key_q : ECC P-384 public key Q (plaintext)
key_pair_index->private : ECC P-384 private key user key index

Return Values

TSIP_SUCCESS:	Normal end
TSIP_ERR_RESOURCE_CONFLICT:	A resource conflict occurred because a hardware resource required by the processing is in use by other processing.
TSIP_ERR_FAIL:	An internal error occurred.

Description

This is an API for outputting user key indexes for an ECC P-384 public key and private key pair. This API generates a user key from a random number value internally within the TSIP. There is therefore no need to input a user key. It is possible to prevent dead copying of data by using the user key index output by this API to encrypt the data. The public key user key index is generated in key_pair_index->public, and the private key user key index is generated in key_pair_index->private.

<State transition>

The valid pre-run state is *TSIP Enabled State*.

After the function runs the state transitions to *TSIP Enabled State*.

For the method of using key_pair_index->public and key_pair_index->private, refer to Chapter 7, Key Data Operations.

key_pair_index->public is the same operation as the public key user key index output from R_TSIP_GenerateEccP384PublicKeyIndex(), and Key_pair_index->private is the same operation as the private key user key index output from R_TSIP_GenerateEccP384PrivateKeyIndex().

Reentrant

Not supported

5.106 R_TSIP_GenerateSha1HmacKeyIndex

Format

```
#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
)
```

Parameters

encrypted_provisioning_key	input	Provisioning key wrapped by the DLM server
iv	input	Initialization vector when generating encrypted_key
encrypted_key	input	User key with encrypted MAC appended
key_index	input/output	User key index

Return Values

TSIP_SUCCESS:	Normal end
TSIP_ERR_RESOURCE_CONFLICT:	A resource conflict occurred because a hardware resource required by the processing is in use by other processing.
TSIP_ERR_FAIL:	An internal error occurred.

Description

This API outputs an SHA1-HMAC user key index.

Input data encrypted in the following format with the provisioning key as encrypted_key.

Bytes	128 bits			
	32 bits	32 bits	32 bits	32 bits
0-15	SHA1-HMAC 160-bit key			
16-31		0 padding		
32-47	MAC			

<State transition>

The valid pre-run state is *TSIP Enabled State*.

After the function runs the state transitions to *TSIP Enabled State*.

For the method of generating encrypted_provisioning_key, iv, and encrypted_key, and instructions for using key_index, refer to Chapter 7, Key Data Operations.

Reentrant

Not supported

5.107 R_TSIP_GenerateSha256HmacKeyIndex

Format

```
#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	User key with encrypted MAC appended
key_index	input/output	User key index

Return Values

TSIP_SUCCESS:	Normal end
TSIP_ERR_RESOURCE_CONFLICT:	A resource conflict occurred because a hardware resource required by the processing is in use by other processing.
TSIP_ERR_FAIL:	An internal error occurred.

Description

This API outputs an SHA256-HMAC user key index.

Input data encrypted in the following format with the provisioning key as encrypted_key.

Bytes	128 bits			
	32 bits	32 bits	32 bits	32 bits
0-15	SHA256-HMAC 256-bit key			
16-31				
32-47	MAC			

<State transition>

The valid pre-run state is *TSIP Enabled State*.

After the function runs the state transitions to *TSIP Enabled State*.

For the method of generating encrypted_provisioning_key, iv, and encrypted_key, and instructions for using key_index, refer to Chapter 7, Key Data Operations.

Reentrant

Not supported

5.108 R_TSIP_UpdateEccP192PublicKeyIndex

Format

```
#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
)
```

Parameters

iv	Input	Initialization vector when generating encrypted_key
encrypted_key	Input	Public key encrypted with key update keyring with MAC value added
key_index	Output	ECC P-192 public key user key index

key_index->value.key_management_info : Key management information
key_index->value.key_q : ECC P-192 public key Q (plaintext)

Return Values

TSIP_SUCCESS:	Normal end
TSIP_ERR_RESOURCE_CONFLICT:	A resource conflict occurred because a hardware resource required by the processing is in use by other processing.
TSIP_ERR_FAIL:	An internal error occurred.

Description

This is an API for updating the key index of an ECC P-192 public key.

For encrypted_key, input data in the following format that has been encrypted with the key update keyring.

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 public key Qx (continuation)			
32-47	0 padding		ECC P-192 public key Qy	
48-63	ECC P-192 public key Qy (continuation)			
64-79	MAC			

<State transition>

The valid pre-run state is *TSIP Enabled State*.

After the function runs the state transitions to *TSIP Enabled State*.

For the method of generating iv and encrypted_key, and for how to use key_index, refer to section 7, Key Data Operations.

Reentrant

Not supported

5.109 R_TSIP_UpdateEccP224PublicKeyIndex

Format

```
#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
)
```

Parameters

iv	Input	Initialization vector when generating encrypted_key
encrypted_key	Input	Public key encrypted with key update keyring with MAC value added
key_index	Output	ECC P-224 public key user key index

key_index->value.key_management_info : Key management information
key_index->value.key_q : ECC P-224 public key Q (plaintext)

Return Values

TSIP_SUCCESS:	Normal end
TSIP_ERR_RESOURCE_CONFLICT:	A resource conflict occurred because a hardware resource required by the processing is in use by other processing.
TSIP_ERR_FAIL:	An internal error occurred.

Description

This is an API for updating the key index of an ECC P-224 public key.

For encrypted_key, input data in the following format that has been encrypted with the key update keyring.

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 public key Qx (continuation)			
32-47	0 padding	ECC P-224 public key Qy		
48-63	ECC P-224 public key Qy (continuation)			
64-79	MAC			

<State transition>

The valid pre-run state is *TSIP Enabled State*.

After the function runs the state transitions to *TSIP Enabled State*.

For the method of generating iv and encrypted_key, and for how to use key_index, refer to section 7, Key Data Operations.

Reentrant

Not supported

5.110 R_TSIP_UpdateEccP256PublicKeyIndex

Format

```
#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
)
```

Parameters

iv	Input	Initialization vector when generating encrypted_key
encrypted_key	Input	Public key encrypted with key update keyring with MAC value added
key_index	Output	ECC P-256 public key user key index

key_index->value.key_management_info : Key management information
key_index->value.key_q : ECC P-256 public key Q (plaintext)

Return Values

TSIP_SUCCESS:	Normal end
TSIP_ERR_RESOURCE_CONFLICT:	A resource conflict occurred because a hardware resource required by the processing is in use by other processing.
TSIP_ERR_FAIL:	An internal error occurred.

Description

This is an API for updating the key index of an ECC P-256 public key.

For encrypted_key, input data in the following format that has been encrypted with the key update keyring.

Bytes	128 bits			
	32 bits	32 bits	32 bits	32 bits
0-31	ECC P-256 public key Qx			
32-63	ECC P-256 public key Qy			
64-79	MAC			

<State transition>

The valid pre-run state is *TSIP Enabled State*.

After the function runs the state transitions to *TSIP Enabled State*.

For the method of generating iv and encrypted_key, and for how to use key_index, refer to section 7, Key Data Operations.

Reentrant

Not supported

5.111 R_TSIP_UpdateEccP384PublicKeyIndex

Format

```
#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 with key update keyring with MAC value added
key_index	Output	ECC P-384 public key user key index

key_index->value.key_management_info : Key management information
key_index->value.key_q : ECC P-384 public key Q (plaintext)

Return Values

TSIP_SUCCESS:	Normal end
TSIP_ERR_RESOURCE_CONFLICT:	A resource conflict occurred because a hardware resource required by the processing is in use by other processing.
TSIP_ERR_FAIL:	An internal error occurred.

Description

This is an API for updating the key index of an ECC P-384 public key.

For encrypted_key, input data in the following format that has been encrypted with the key update keyring.

Bytes	128 bits			
	32 bits	32 bits	32 bits	32 bits
0-47	ECC P-384 public key Qx			
48-95	ECC P-384 public key Qy			
96-111	MAC			

<State transition>

The valid pre-run state is *TSIP Enabled State*.

After the function runs the state transitions to *TSIP Enabled State*.

For the method of generating iv and encrypted_key, and for how to use key_index, refer to section 7, Key Data Operations.

Reentrant

Not supported

5.112 R_TSIP_UpdateEccP192PrivateKeyIndex

Format

```
#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
)
```

Parameters

iv	Input	Initialization vector when generating encrypted_key
encrypted_key	Input	Private key encrypted with key update keyring with MAC value added
key_index	Output	ECC P-192 private key user key index

Return Values

TSIP_SUCCESS:	Normal end
TSIP_ERR_RESOURCE_CONFLICT:	A resource conflict occurred because a hardware resource required by the processing is in use by other processing.
TSIP_ERR_FAIL:	An internal error occurred.

Description

This is an API for updating the key index of an ECC P-192 private key.

For encrypted_key, input data in the following format that has been encrypted with the key update keyring.

Bytes	128 bits			
	32 bits	32 bits	32 bits	32 bits
0-15	0 padding		ECC P-192 private key	
16-31	ECC P-192 private key (continuation)			
32-47	MAC			

<State transition>

The valid pre-run state is *TSIP Enabled State*.

After the function runs the state transitions to *TSIP Enabled State*.

For the method of generating iv and encrypted_key, and for how to use key_index, refer to section 7, Key Data Operations.

Reentrant

Not supported

5.113 R_TSIP_UpdateEccP224PrivateKeyIndex

Format

```
#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
)
```

Parameters

iv	Input	Initialization vector when generating encrypted_key
encrypted_key	Input	Private key encrypted with key update keyring with MAC value added
key_index	Output	ECC P-224 private key user key index

Return Values

TSIP_SUCCESS:	Normal end
TSIP_ERR_RESOURCE_CONFLICT:	A resource conflict occurred because a hardware resource required by the processing is in use by other processing.
TSIP_ERR_FAIL:	An internal error occurred.

Description

This is an API for updating the key index of an ECC P-224 private key.

For encrypted_key, input data in the following format that has been encrypted with the key update keyring.

Bytes	128 bits			
	32 bits	32 bits	32 bits	32 bits
0-15	0 padding	ECC P-224 private key		
16-31	ECC P-224 private key (continuation)			
32-47	MAC			

<State transition>

The valid pre-run state is *TSIP Enabled State*.

After the function runs the state transitions to *TSIP Enabled State*.

For the method of generating iv and encrypted_key, and for how to use key_index, refer to section 7, Key Data Operations.

Reentrant

Not supported

5.114 R_TSIP_UpdateEccP256PrivateKeyIndex

Format

```
#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
)
```

Parameters

iv	Input	Initialization vector when generating encrypted_key
encrypted_key	Input	Private key encrypted with key update keyring with MAC value added
key_index	Output	ECC P-256 private key user key index

Return Values

TSIP_SUCCESS:	Normal end
TSIP_ERR_RESOURCE_CONFLICT:	A resource conflict occurred because a hardware resource required by the processing is in use by other processing.
TSIP_ERR_FAIL:	An internal error occurred.

Description

This is an API for updating the key index of an ECC P-256 private key.

For encrypted_key, input data in the following format that has been encrypted with the key update keyring.

Bytes	128 bits			
	32 bits	32 bits	32 bits	32 bits
0-31	ECC P-256 private key			
32-47	MAC			

<State transition>

The valid pre-run state is *TSIP Enabled State*.

After the function runs the state transitions to *TSIP Enabled State*.

For the method of generating iv and encrypted_key, and for how to use key_index, refer to section 7, Key Data Operations.

Reentrant

Not supported

5.115 R_TSIP_UpdateEccP384PrivateKeyIndex

Format

```
#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	Private key encrypted with key update keyring with MAC value added
key_index	Output	ECC P-384 private key user key index

Return Values

TSIP_SUCCESS:	Normal end
TSIP_ERR_RESOURCE_CONFLICT:	A resource conflict occurred because a hardware resource required by the processing is in use by other processing.
TSIP_ERR_FAIL:	An internal error occurred.

Description

This is an API for updating the key index of an ECC P-384 private key.

For encrypted_key, input data in the following format that has been encrypted with the key update keyring.

Bytes	128 bits			
	32 bits	32 bits	32 bits	32 bits
0-47	ECC P-384 private key			
48637	MAC			

<State transition>

The valid pre-run state is *TSIP Enabled State*.

After the function runs the state transitions to *TSIP Enabled State*.

For the method of generating iv and encrypted_key, and for how to use key_index, refer to section 7, Key Data Operations.

Reentrant

Not supported

5.116 R_TSIP_UpdateSha1HmacKeyIndex

Format

```
#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
)
```

Parameters

iv	Input	Initialization vector when generating encrypted_key
encrypted_key	Input	User key encrypted with key update keyring with MAC value added
key_index	Input/output	User key index

Return Values

TSIP_SUCCESS:	Normal end
TSIP_ERR_RESOURCE_CONFLICT:	A resource conflict occurred because a hardware resource required by the processing is in use by other processing.
TSIP_ERR_FAIL:	An internal error occurred.

Description

This API updates the user key index of an SHA1-HMAC key.

Input data encrypted in the following format with the key update keyring as encrypted_key.

Bytes	128 bits			
	32 bits	32 bits	32 bits	32 bits
0-15	SHA1-HMAC 160-bit key			
16-31		0 padding		
32-47	MAC			

<State transition>

The valid pre-run state is *TSIP Enabled State*.

After the function runs the state transitions to *TSIP Enabled State*.

For the method of generating encrypted_provisioning_key, iv, and encrypted_key, and instructions for using key_index, refer to Chapter 7, Key Data Operations.

Reentrant

Not supported

5.117 R_TSIP_UpdateSha256HmacKeyIndex

Format

```
#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 with key update keyring with MAC value added
key_index	Input/output	User key index

Return Values

TSIP_SUCCESS:	Normal end
TSIP_ERR_RESOURCE_CONFLICT:	A resource conflict occurred because a hardware resource required by the processing is in use by other processing.
TSIP_ERR_FAIL:	An internal error occurred.

Description

This API updates the user key index of an SHA256-HMAC key.

Input data encrypted in the following format with the key update keyring as encrypted_key.

Bytes	128 bits			
	32 bits	32 bits	32 bits	32 bits
0-15	SHA256-HMAC 256-bit key			
16-31				
32-47	MAC			

<State transition>

The valid pre-run state is *TSIP Enabled State*.

After the function runs the state transitions to *TSIP Enabled State*.

For the method of generating encrypted_provisioning_key, iv, and encrypted_key, and instructions for using key_index, refer to Chapter 7, Key Data Operations.

Reentrant

Not supported

5.118 R_TSIP_EcdsaP192SignatureGenerate

Format

```
#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
)
```

Parameters

message_hash	Input	Message or hash value to which to attach signature	
message_hash->pdata			: Specifies pointer to array storing the message or hash value
message_hash->data_length			: Specifies effective data length of the array (Specify only when Message is selected)
message_hash->data_type			: Selects the data type of message_hash Message: 0 Hash value: 1
signature	Output	Signature text storage destination information	
signature->pdata			: Specifies pointer to array storing signature text The signature format is "0 padding (64 bits) || signature r (192 bits) || 0 padding (64 bits) || signature s (192 bits)".
signature->data_length			: Data length (byte units)
key_index	Input	Key data area	: Input user key index of ECC P-192 private key.

Return Values

TSIP_SUCCESS:	Normal end
TSIP_ERR_RESOURCE_CONFLICT:	A resource conflict occurred because a hardware resource required by the processing is in use by other processing.
TSIP_ERR_KEY_SET:	Invalid user key index was input.
TSIP_ERR_FAIL:	An internal error occurred.
TSIP_ERR_PARAMETER:	Input data is invalid.
Other then the above Return Values	Return value from an internal function that performs a hash operation.

Description

When a message is specified in the first argument, message_hash->data_type, a SHA-256 hash of the message text input as the first argument, message_hash->pdata, is calculated, and the signature text is written to the second argument, signature, in accordance with ECDSA P-192 using the private key user key index input as the third argument, key_index.

When a hash value is specified in the first argument, message_hash->data_type, the signature text for the first 24 bytes of the SHA-256 hash value input to the first argument, message_hash->pdata, is written to the second argument, signature, in accordance with ECDSA P-192 using the private key user key index input as the third argument, key_index.

<State transition>

The valid pre-run state is *TSIP Enabled State*.

After the function runs the state transitions to *TSIP Enabled State*.

For details on how to use key_pair_index, refer to section 7, Key Data Operations.

Reentrant

Not supported

5.119 R_TSIP_EcdsaP224SignatureGenerate

Format

```
#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
)
```

Parameters

message_hash	Input	Message or hash value to which to attach signature	
message_hash->pdata			: Specifies pointer to array storing the message or hash value
message_hash->data_length			: Specifies effective data length of the array (Specify only when Message is selected)
message_hash->data_type			: Selects the data type of message_hash Message: 0 Hash value: 1
signature	Output	Signature text storage destination information	
signature->pdata			: Specifies pointer to array storing signature text The signature format is "0 padding (32 bits) || signature r (224 bits) || 0 padding (32 bits) || signature s (224 bits)".
signature->data_length			: Data length (byte units)
key_index	Input	Key data area	: Input user key index of ECC P-224 private key.

Return Values

TSIP_SUCCESS:	Normal end
TSIP_ERR_RESOURCE_CONFLICT:	A resource conflict occurred because a hardware resource required by the processing is in use by other processing.
TSIP_ERR_KEY_SET:	Invalid user key index was input.
TSIP_ERR_FAIL:	An internal error occurred.
TSIP_ERR_PARAMETER:	Input data is invalid.
Other then the above Return Values	Return value from an internal function that performs a hash operation.

Description

When a message is specified in the first argument, message_hash->data_type, a SHA-256 hash of the message text input as the first argument, message_hash->pdata, is calculated, and the signature text is written to the second argument, signature, in accordance with ECDSA P-224 using the private key user key index input as the third argument, key_index.

When a hash value is specified in the first argument, message_hash->data_type, the signature text for the first 28 bytes of the SHA-256 hash value input to the first argument, message_hash->pdata, is written to the second argument, signature, in accordance with ECDSA P-224 using the private key user key index input as the third argument, key_index.

<State transition>

The valid pre-run state is *TSIP Enabled State*.

After the function runs the state transitions to *TSIP Enabled State*.

For details on how to use key_pair_index, refer to section 7, Key Data Operations.

Reentrant

Not supported

5.120 R_TSIP_EcdsaP256SignatureGenerate

Format

```
#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
)
```

Parameters

message_hash	Input	Message or hash value to which to attach signature
message_hash->pdata		: Specifies pointer to array storing the message or hash value
message_hash->data_length		: Specifies effective data length of the array (Specify only when Message is selected)
message_hash->data_type		: Selects the data type of message_hash Message: 0 Hash value: 1
signature	Output	Signature text storage destination information
signature->pdata		: Specifies pointer to array storing signature text The signature format is signature r (256 bits) || signature s (256 bits)
signature->data_length		: Data length (byte units)
key_index	Input	Key data area : Input user key index of ECC P-256 private key.

Return Values

TSIP_SUCCESS:	Normal end
TSIP_ERR_RESOURCE_CONFLICT:	A resource conflict occurred because a hardware resource required by the processing is in use by other processing.
TSIP_ERR_KEY_SET:	Invalid user key index was input.
TSIP_ERR_FAIL:	An internal error occurred.
TSIP_ERR_PARAMETER:	Input data is invalid.
Other then the above Return Values	Return value from an internal function that performs a hash operation.

Description

When a message is specified in the first argument, message_hash->data_type, a SHA-256 hash of the message text input as the first argument, message_hash->pdata, is calculated, and the signature text is written to the second argument, signature, in accordance with ECDSA P-256 using the private key user key index input as the third argument, key_index.

When a hash value is specified in the first argument, message_hash->data_type, the signature text for the entire 32 bytes of the SHA-256 hash value input to the first argument, message_hash->pdata, is written to the second argument, signature, in accordance with ECDSA P-256 using the private key user key index input as the third argument, key_index.

<State transition>

The valid pre-run state is *TSIP Enabled State*.

After the function runs the state transitions to *TSIP Enabled State*.

For details on how to use key_pair_index, refer to section 7, Key Data Operations.

Reentrant

Not supported

5.121 R_TSIP_EcdsaP384SignatureGenerate

Format

```
#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

Hash value to which to attach signature			
message_hash->pdata			: Specifies pointer to array storing the hash value
message_hash->data_length		: Specifies effective data length of the array (Nonuse)	
message_hash->data_type			: Only 1 can be specified
signature	Output	Signature text storage destination information	
signature->pdata			: Specifies pointer to array storing signature text The signature format is signature r (384 bits) || signature s (384 bits)
signature->data_length			: Data length (byte units)
key_index	Input	Key data area	: Input user key index of ECC P-384 private key.

Return Values

TSIP_SUCCESS:	Normal end
TSIP_ERR_RESOURCE_CONFLICT:	A resource conflict occurred because a hardware resource required by the processing is in use by other processing.
TSIP_ERR_KEY_SET:	Invalid user key index was input.
TSIP_ERR_FAIL:	An internal error occurred.
TSIP_ERR_PARAMETER:	Input data is invalid.

Description

The signature text for the first 48 bytes of the SHA-384 hash value input to the first argument, message_hash->pdata, is written to the second argument, signature, in accordance with ECDSA P-384 using the private key user key index input as the third argument, key_index.

<State transition>

The valid pre-run state is *TSIP Enabled State*.

After the function runs the state transitions to *TSIP Enabled State*.

For details on how to use key_pair_index, refer to section 7, Key Data Operations.

Reentrant

Not supported

5.122 R_TSIP_EcdsaP192SignatureVerification

Format

```
#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
)
```

Parameters

signature	Input	Signature text information to be verified	
signature->pdata			: Specifies pointer to array storing signature text The signature format is "0 padding (64 bits) || signature r (192 bits) || 0 padding (64 bits) || signature s (192 bits)".
signature->data_length			: Specifies the data length (byte units) (nonuse)
message_hash	Input	Message or hash value to be verified	
message_hash->pdata			: Specifies pointer to array storing the message or hash value
message_hash->data_length			: Specifies effective data length of the array (Specify only when Message is selected)
message_hash->data_type			: Selects the data type of message_hash Message: 0 Hash value: 1
key_index	Input	Key data area	: Input user key index of ECC P-192 public key.

Return Values

TSIP_SUCCESS:	Normal end
TSIP_ERR_RESOURCE_CONFLICT:	A resource conflict occurred because a hardware resource required by the processing is in use by other processing.
TSIP_ERR_KEY_SET:	Invalid user key index was input.
TSIP_ERR_FAIL:	An internal error occurred, or signature verification failed.
TSIP_ERR_PARAMETER:	Input data is invalid.
Other then the above Return Values	Return value from an internal function that performs a hash operation.

Description

When a message is specified in the second argument, message_hash->data_type, a SHA-256 hash of the message text input as the second argument, message_hash->pdata, is calculated, and the signature text input to the first argument, signature, is validated in accordance with ECDSA P-192 using the public key user key index input as the third argument, key_index.

When a hash value is specified in the second argument, message_hash->data_type, the signature text for the first 24 bytes of the SHA-256 hash value input to the second argument, message_hash->pdata, input to the first argument, signature, is validated in accordance with ECDSA P-192 using the public key user key index input as the third argument, key_index.

<State transition>

The valid pre-run state is *TSIP Enabled State*.

After the function runs the state transitions to *TSIP Enabled State*.

For details on how to use key_pair_index, refer to section 7, Key Data Operations.

Reentrant

Not supported

5.123 R_TSIP_EcdsaP224SignatureVerification

Format

```
#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
)
```

Parameters

signature	Input	Signature text information to be verified
signature->pdata		: Specifies pointer to array storing signature text The signature format is "0 padding (32 bits) || signature r (224 bits) || 0 padding (32 bits) || signature s (224 bits)".
signature->data_length		: Specifies the data length (byte units) (nonuse)
message_hash	Input	Message or hash value to be verified
message_hash->pdata		: Specifies pointer to array storing the message or hash value
message_hash->data_length		: Specifies effective data length of the array (Specify only when Message is selected)
message_hash->data_type		: Selects the data type of message_hash Message: 0 Hash value: 1
key_index	Input	Key data area : Input user key index of ECC P-224 public key.

Return Values

TSIP_SUCCESS:	Normal end
TSIP_ERR_RESOURCE_CONFLICT:	A resource conflict occurred because a hardware resource required by the processing is in use by other processing.
TSIP_ERR_KEY_SET:	Invalid user key index was input.
TSIP_ERR_FAIL:	An internal error occurred, or signature verification failed.
TSIP_ERR_PARAMETER:	Input data is invalid.
Other then the above Return Values	Return value from an internal function that performs a hash operation.

Description

When a message is specified in the second argument, message_hash->data_type, a SHA-256 hash of the message text input as the second argument, message_hash->pdata, is calculated, and the signature text input to the first argument, signature, is validated in accordance with ECDSA P-224 using the public key user key index input as the third argument, key_index.

When a hash value is specified in the second argument, message_hash->data_type, the signature text for the first 28 bytes of the SHA-256 hash value input to the second argument, message_hash->pdata, input to the first argument, signature, is validated in accordance with ECDSA P-224 using the public key user key index input as the third argument, key_index.

<State transition>

The valid pre-run state is *TSIP Enabled State*.

After the function runs the state transitions to *TSIP Enabled State*.

For details on how to use key_pair_index, refer to section 7, Key Data Operations.

Reentrant

Not supported

5.124 R_TSIP_EcdsaP256SignatureVerification

Format

```
#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
)
```

Parameters

signature	Input	Signature text information to be verified
signature->pdata		: Specifies pointer to array storing signature text The signature format is signature r (256 bits) || signature s (256 bits)"
signature->data_length		: Specifies the data length (byte units) (nonuse)
message_hash	Input	Message or hash value to be verified
message_hash->pdata		: Specifies pointer to array storing the message or hash value
message_hash->data_length		: Specifies effective data length of the array (Specify only when Message is selected)
message_hash->data_type		: Selects the data type of message_hash Message: 0 Hash value: 1
key_index	Input	Key data area : Input user key index of ECC P-256 public key.

Return Values

TSIP_SUCCESS:	Normal end
TSIP_ERR_RESOURCE_CONFLICT:	A resource conflict occurred because a hardware resource required by the processing is in use by other processing.
TSIP_ERR_KEY_SET:	Invalid user key index was input.
TSIP_ERR_FAIL:	An internal error occurred, or signature verification failed.
TSIP_ERR_PARAMETER:	Input data is invalid.
Other then the above Return Values	Return value from an internal function that performs a hash operation.

Description

When a message is specified in the second argument, message_hash->data_type, a SHA-256 hash of the message text input as the second argument, message_hash->pdata, is calculated, and the signature text input to the first argument, signature, is validated in accordance with ECDSA P-256 using the public key user key index input as the third argument, key_index.

When a hash value is specified in the second argument, message_hash->data_type, the signature text for the entire 32 bytes of the SHA-256 hash value input to the second argument, message_hash->pdata, input to the first argument, signature, is validated in accordance with ECDSA P-256 using the public key user key index input as the third argument, key_index.

<State transition>

The valid pre-run state is *TSIP Enabled State*.

After the function runs the state transitions to *TSIP Enabled State*.

For details on how to use key_pair_index, refer to section 7, Key Data Operations.

Reentrant

Not supported

5.125 R_TSIP_EcdsaP384SignatureVerification

Format

```
#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
signature->pdata		: Specifies pointer to array storing signature text The signature format is signature r (384 bits) || signature s (384 bits)"
signature->data_length		: Specifies the data length (byte units) (nonuse)
message_hash	Input	Hash value to be verified
message_hash->pdata		: Specifies pointer to array storing the message or hash value
message_hash->data_length		: Specifies effective data length of the array (Nonuse)
message_hash->data_type		: Only 1 can be specified
key_index	Input	Key data area : Input user key index of ECC P-384 public key.

Return Values

TSIP_SUCCESS:	Normal end
TSIP_ERR_RESOURCE_CONFLICT:	A resource conflict occurred because a hardware resource required by the processing is in use by other processing.
TSIP_ERR_KEY_SET:	Invalid user key index was input.
TSIP_ERR_FAIL:	An internal error occurred, or signature verification failed.
TSIP_ERR_PARAMETER:	Input data is invalid.

Description

The signature text for the entire 48 bytes of the SHA-384 hash value input to the second argument, message_hash->pdata, and the signature text input to the first argument, signature, is validated in accordance with ECDSA P-384 using the public key user key index input as the third argument, key_index.

<State transition>

The valid pre-run state is *TSIP Enabled State*.

After the function runs the state transitions to *TSIP Enabled State*.

For details on how to use key_pair_index, refer to section 7, Key Data Operations.

Reentrant

Not supported

5.126 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	Input/output	ECDH handler (work area)
key_type	Input	Key exchange type 0: ECDHE 1: ECDH
use_key_id	Input	0: key_id not used, 1: key_id used

Return Values

TSIP_SUCCESS:	Normal end
TSIP_ERR_PARAMETER:	Input data is invalid.

Description

The R_TSIP_EcdhP256Init function prepares to perform ECDH key exchange computation and writes the result to the first argument, handle. The succeeding functions R_TSIP_EcdhP256ReadPublicKey, R_TSIP_EcdhP256MakePublicKey, R_TSIP_EcdhP256CalculateSharedSecretIndex, and R_TSIP_EcdhP256KeyDerivation use handle as an argument.

Use the second argument, key_type, to select the type of ECDH key exchange. When ECDHE is selected, the R_TSIP_EcdhP256MakePublicKey function uses the TSIP's random number generation functionality to generate an ECC P-256 key pair. When ECDH is selected, keys installed beforehand are used for key exchange.

Input 1 as the third argument, use_key_id, to use key_id when key exchange is performed. key_id is for applications conforming to the DLMS/COSEM standard for smart meters.

<State transition>

The valid pre-run state is *TSIP Enabled State*.

After the function runs the state transitions to *TSIP Enabled State*.

For details on how to use key_index, refer to section 7, Key Data Operations.

Reentrant

Not supported

5.127 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 user key index area for signature verification
public_key_data	Input	ECC P-256 public key (512-bit) When key_id is used: key_id (8-bit) || public key (512-bit)
signature	Input	ECDSA P-256 signature of public_key_data
key_index	Output	Key index of public_key_data

Return Values

TSIP_SUCCESS:	Normal end
TSIP_ERR_RESOURCE_CONFLICT:	A resource conflict occurred because a hardware resource required by the processing is in use by other processing.
TSIP_ERR_KEY_SET:	Invalid user key index was input.
TSIP_ERR_FAIL:	An internal error occurred, or signature verification failed.
TSIP_ERR_PARAMETER:	An invalid handle was input.
TSIP_ERR_PROHIBIT_FUNCTION:	An invalid function was 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 to the fifth argument.

The first argument, handle, is used as an argument in the subsequent function R_TSIP_EcdhP256CalculateSharedSecretIndex().

R_TSIP_EcdhP256CalculateSharedSecretIndex uses key_index as input to calculate Z.

<State transition>

The valid pre-run state is *TSIP Enabled State*.

After the function runs the state transitions to *TSIP Enabled State*.

For details on how to use key_index, refer to section 7, Key Data Operations.

Reentrant

Not supported

5.128 R_TSIP_EcdhP256MakePublicKey

Format

```
#include "r_tsip_rx_if.h"
e_tsip_err_t R_TSIP_EcdhP256MakePublicKey (
        tsip_ecdh_handle_t *handle,
        tsip_ecc_public_key_index_t *public_key_index,
        tsip_ecc_private_key_index_t *private_key_index,
        uint8_t *public_key,
        tsip_ecdsa_byte_data_t *signature,
        tsip_ecc_private_key_index_t *key_index
)
```

Parameters

handle	Input/output	ECDH handler (work area) When using key_id, input handle->key_id after running R_TSIP_EcdhP256Init().
public_key_index	Input	For ECDHE, input a null pointer. For ECDH, input the key index of a ECC P-256 public key.
private_key_index	Input	ECC P-256 private key for signature generation
public_key	Output	User public key (512-bit) for key exchange When using key_id, key_id (8-bit) || user public key (512-bit) || 0 padding (24-bit)
signature	Output	Signature text storage destination information
->pdata		: Specifies pointer to array for storing signature text Signature format: signature r (256-bit) || signature s (256-bit)”
->data_length		: Data length (in byte units)
key_index	Output	For ECDHE, a private key user key index generated from a random number. Not output for ECDH.

Return Values

TSIP_SUCCESS:	Normal end
TSIP_ERR_RESOURCE_CONFLICT:	A resource conflict occurred because a hardware resource required by the processing is in use by other processing.
TSIP_ERR_KEY_SET:	Invalid user key index was input.
TSIP_ERR_FAIL:	An internal error occurred.
TSIP_ERR_PARAMETER:	An invalid handle was input.
TSIP_ERR_PROHIBIT_FUNCTION:	An invalid function was called.

Description

The R_TSIP_EcdhP256MakePublicKey() function calculates a signature for a public key user key index used for ECDH key exchange.

If ECDHE is specified by the key_type argument 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 private key is output to key_index.

If ECDH is specified by the key_type argument 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 succeeding function R_TSIP_EcdhP256CalculateSharedSecretIndex() uses the first argument, handle, as an argument.

The R_TSIP_EcdhP256CalculateSharedSecretIndex() function uses key_index as input to calculate Z.

<State transition>

The valid pre-run state is *TSIP Enabled State*.

After the function runs the state transitions to *TSIP Enabled State*.

For details on how to use key_index, refer to section 7, Key Data Operations.

Reentrant

Not supported

5.129 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 user key index whose signature was verified by R_TSIP_EcdhP256ReadPublicKey()
private_key_index	Input	Private key user key index
shared_secret_index	Output	Key index of shared secret Z calculated by ECDH key exchange

Return Values

TSIP_SUCCESS:	Normal end
TSIP_ERR_RESOURCE_CONFLICT:	A resource conflict occurred because a hardware resource required by the processing is in use by other processing.
TSIP_ERR_KEY_SET:	Invalid user key index was input.
TSIP_ERR_FAIL:	An internal error occurred.
TSIP_ERR_PARAMETER:	An invalid handle was input.
TSIP_ERR_PROHIBIT_FUNCTION:	An invalid function was 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 private key.

Input as the second argument, public_key_index, the public key user key index whose signature was verified by R_TSIP_EcdhP256ReadPublicKey().

When key_type of R_TSIP_EcdhP256Init() is 0, input as the third argument, private_key_index, the private key user key index generated from a random number by R_TSIP_EcdhP256MakePublicKey(), and when key_type is other than 0, input the private key user key index that forms a pair with the second argument of R_TSIP_EcdhP256MakePublicKey().

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

<State transition>

The valid pre-run state is *TSIP Enabled State*.

After the function runs the state transitions to *TSIP Enabled State*.

For details on how to use key_index, refer to section 7, Key Data Operations.

Reentrant

Not supported

5.130 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: SHA-256 1: SHA-256 HMAC
other_info	Input	Additional data used for key derivation calculation AlgorithmID || PartyUInfo || PartyVInfo
other_info_length	Input	Data length of other_info (up to 147 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. 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 end
TSIP_ERR_RESOURCE_CONFLICT:	A resource conflict occurred because a hardware resource required by the processing is in use by other processing.
TSIP_ERR_KEY_SET:	Invalid user key index was input.
TSIP_ERR_PARAMETER:	An invalid handle was input.
TSIP_ERR_PROHIBIT_FUNCTION:	An invalid function was called.

Description

The R_TSIP_EcdhP256KeyDerivation() function uses the shared secret "Z (shared_secret_index)" calculated by the R_TSIP_EcdhP256CalculateSharedSecretIndex() function as the key material to derive the key index specified by the third argument, key_type.The key derivation algorithm is one-step key derivation as defined in NIST SP800-56C. Either SHA-256 or SHA-256 HMAC is specified by the fourth argument, 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 argument, salt_key_index.

Enter a fixed value for deriving a key shared with the key exchange partner in the fifth argument, other_info.

A key index corresponding to key_type is output as the eighth argument, key_index. The correspondences between the types of derived key_index 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()

<State transition>

The valid pre-run state is *TSIP Enabled State*.

After the function runs the state transitions to *TSIP Enabled State*.

For details on how to use key_index, refer to section 7, Key Data Operations.

Reentrant

Not supported

5.131 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	User key index area for AES-128 CMAC operation
receiver_public_key	Input	Receiver's Brainpool P512r1 public key Q (1024-bit) || MAC (128-bit)
sender_public_key	Input/output	Sender's Brainpool P512r1 public key Q (1024-bit) || MAC (128-bit)

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_KEY_SET:	Invalid user key index was input.
TSIP_ERR_RESOURCE_CONFLICT:	A resource conflict occurred because a hardware resource needed by the processing routine was in use by another processing routine.
TSIP_ERR_FAIL:	An internal error occurred.

Description

Performs an ECDHE operation after generation of a key pair using Brainpool P512r1.

Note that the sender is the TSIP and the receiver is the other key exchange party.

<State transition>

The valid pre-run state is *TSIP Enabled State*.

After the function runs the state transitions to *TSIP Enabled State*.

Reentrant

Not supported

6. Callback Function

6.1 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	request contents (TSIP_FW_CB_REQ_TYPE)
iLoop	input	loop counts (WORD unit)
counter	input/output	offset for the area references
InData_UpProgram	input/output	same address as the 3rd argument "InData_UpProgram" of R_TSIP_GenerateFirmwareMAC()
OutData_Program	input/output	same address as the 5th argument "OutData_Program" of R_TSIP_GenerateFirmwareMAC()
MAX_CNT	input	same value as the 6th argument "MAX_CNT" of R_TSIP_GenerateFirmwareMAC()

Return Values

None

Description

This function is used in the R_TSIP_GenerateFirmwareMAC and is registered in the 7th argument of this function.

This is used to store the decrypted firmware and MAC at user side.
The area size of InData_UpProgram and OutData_Program should be the multiple of 4, and require at least 4 words. InData_UpProgram and OutData_Program should be the same size. The enclosed sample program is the size of the minimum code flash write unit.

This callback function is called in the R_TSIP_GenerateFirmwareMAC for multiple applications.
The application is stored in the 1st argument "req_type".

The 1st argument ”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;
```

According to this value, the user takes necessary actions.

<req_type = TSIP_FW_CB_REQ_PRG_WT>

This is the storage request of the decrypted firmware.

TSIP Module makes this request accordingly after storing the data in the 5th argument “OutData_Program” by 4-word unit.

The processing is not required on each request.

Store the decrypted firmware according to the area secured at user side. For example, when the areas are secured for 8 words, store the firmware decrypted when noticed twice.

The sum of the size decrypted is stored in the 2nd argument ”iLoop”.

The maximum value of the ”iLoop” in this request is the value subtracting 4 words from the 6th argument “MAX_CNT”. The last 4 words and the firmware not stored are handled in the request of <req_type = TSIP_FW_CB_REQ_PRG_WT_LAST_BLK>.

<req_type = TSIP_FW_CB_REQ_PRG_RD>

This is the request for obtaining the firmware checksum value for the firmware to be updated.

TSIP Module makes this request accordingly before processing the decryption by 4-word unit.

The system is the same as <req_type = TSIP_FW_CB_REQ_PRG_WT>.

Store the firmware in the 4th argument ”InData_UpProgram” according to the area secured at user side.

<req_type = TSIP_FW_CB_REQ_BUFF_CNT,>

This is the offset value request when referring to the 4th argument ”InData_UpProgram” and the 5th argument ”OutData_Program”.

Return the value with 4-word increment for the 3rd argument “counter” to the 3rd argument ”counter”.

When exceeding the size secured in the 4th argument “InData_UpProgram” and the 5th argument “OutData_Program”, restore the 3rd argument “counter” to its default settings.

<req_type = TSIP_FW_CB_REQ_PRG_WT_LAST_BLK>

This request is made 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 the request for obtaining the firmware checksum value for the firmware to be updated.

Store the checksum value in the 4th argument “InData_UpProgram”. The checksum is 16byte in length.

Checksum value is shown as CHECKSUM in the description of Section 7.1.

<req_type = req_type = TSIP_FW_CB_REQ_STORE_MAC>

The MAC for the decrypted firmware is output.

The MAC (for 16bytes) is stored in the 5th argument “OutData_Program”.

The 6th argument ”MAX_CNT” is the same value as the R_TSIP_GenerateFirmwareMAC()’s.

7. Key Data Operations

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/

7.1 AES User Key Operation

7.1.1 AES User Key Installation Overview

The method of installing AES user keys is described below.

An AES user key is an arbitrary byte sequence (128 or 256 bits in length) that is generated on a user PC.

The AES user key is unique for each user.

Install a user key in accordance with this installation procedure. In addition, until the user key is written to the RX microcontroller's internal data flash memory in the course of following the processing flow below, be sure to perform all processing in a safe location (for example, a factory under the direct management of the user's company).

The user key is written to the data flash in the form of user key index. Recovering a user key from this user key index is only possible from within TSIP. It cannot be accessed in purely software form.

By inputting the user key index to the respective APIs, the user key is recovered from within TSIP. Since user key index is encrypted using device-specific information, if the user key index in data flash memory is copied to and used on a different RX microcontroller with built-in TSIP, it will not yield correct encryption and decryption results. In addition, if invalid user key index is input to TSIP, it will not operate properly.

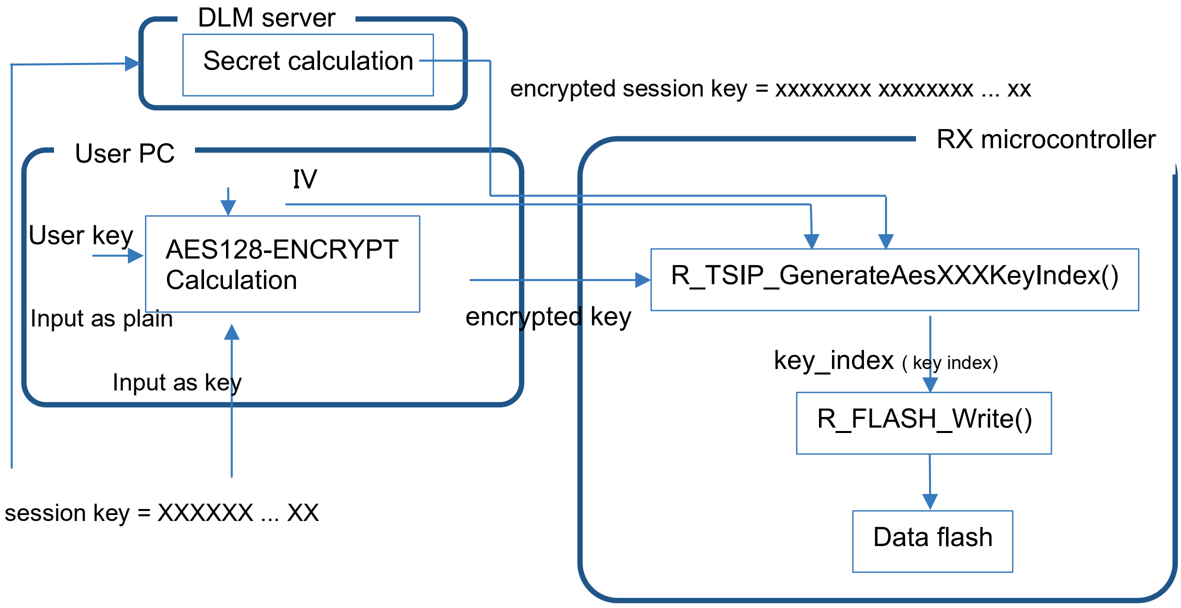


Figure 7.1 Scheme of Install the AES User Key

An example of generation of user key on the user PC is presented on the following pages assumed that the user's PCis running Microsoft Windows.

Renesas Secure Flash Programmer is used to generate the user key.

7.1.2 AES User Key “encrypted key” Creation Method

Launch the Renesas Secure Flash Programmer.

Figure 7.2 Renesas Secure Flash Programmer (Key Wrap Tab, AES 128-bit Key Setting)

Enter user key settings in the Key Wrap tab.

Here we will make settings for outputting keys (AES 128-bit and 256-bit) that an AES user can use freely and keys (AES 128-bit) used for firmware updates.

Select AES 128-bit or AES 256-bit under “Key Type” on the Key Wrap tab.

If you selected AES 128-bit, input 16 bytes of key information in the “Key Data” field, and if you selected AES 256-bit, input 32 bytes of key information. Click the “Register” button to register the key information entered in the key list. The format of the data entered in the key list is as follows.

- AES 128-bit Data Format

bytes	128-bit
0-15	AES 128 key data

- AES 256-bit Data Format

Bytes	256-bit
0-31	AES 256 key data

Set information in "provisioning key File Path" and "encrypted provisioning key File Path" of "provisioning key". Set "provisioning key File Path" to **sample.key** in the FITDemos folder and "encrypted provisioning key File Path" to **sample.key_enc.key**.

After specifying the provisioning key file and encrypted provisioning key file, as well as the IV value if necessary, click the [Generate Key File …] button to generate the encrypted key (encrypted user key) data files key_data.c and key_data.h for input to the R_TSIP_GenerateAesXXXKeyIndex() function.

7.2 TDES User Key Operation

7.2.1 TDES User Key Installation Overview

The TDES user key installation procedure is described below.

The TDES user key comprises three keys, each consisting of 56 bits of data generated on the user's PC.

Each user's TDES user key has a unique value.

Follow the procedure described below to install the user key. Also, ensure that all processing shown in the flowchart below for writing the user key to the on-chip flash memory of the RX MCU is performed in a secure site (such as a plant operated directly by the customer).

The user key is written to the data flash in the form of user key index. Recovering the user key from the user key index can only be performed internally by the TSIP. This data is not software accessible.

The user key is recovered internally by the TSIP when the user key index is input via the various API functions. Since the user key index has been encrypted using device-specific information, it is not possible to generate correct decryption or encryption results by copying the user key index in the data flash to another TSIP-equipped RX MCU. In addition, the TSIP will not operate correctly if an incorrect user key index is input to the TSIP.

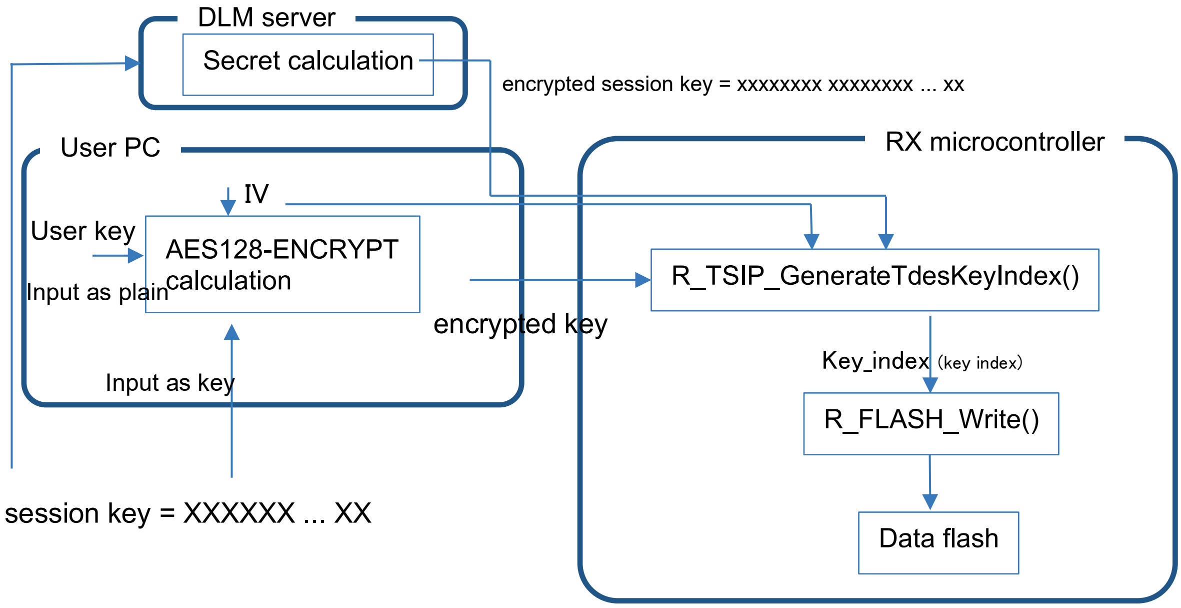


Figure 7.3 TDES User Key Installation

TDES user key data format

bytes	128-bit			
	32-bit	32-bit	32-bit	32-bit
0-15	DES user key1*		DES user key2	
16-31	DES user key3		0 padding	

* DES user key n

The data length of the DES user key is 56 bits. An odd parity bit is appended to each 7 bits of key data, so the DES user key comprises 64 bits of data.

The format of DES user key n is shown below.

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

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

- Use as DES
  Enter values such that DES user key 1 = DES user key 2 = DES user key 3.
- Use as 2-Key TDES
  Enter values such that DES user key 1 = DES user key 3 and DES user key 1 not equal DES user key 2.

An example of generation of a user key on the user’s PC is presented on the following pages. It is assumed that the user’s PC is running Microsoft Windows.

Renesas Secure Flash Programmer is used to generate the user key.

7.2.2 TDES User Key “encrypted key” Creation Method

Launch Renesas Secure Flash Programmer.

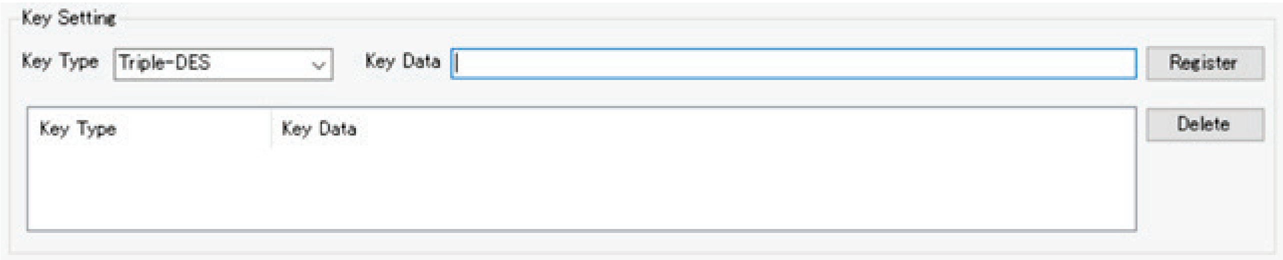


Figure 7.4 Renesas Secure Flash Programmer (Key Wrap Tab, Triple-DES Key Setting)

Enter user key settings in the Key Wrap tab.

Here we will make settings for outputting keys (Triple-DES, 2-Key TDES, and DES) that a TDES user can use freely.

Select Triple-DES, 2-Key TDES, or DES under “Key Type” on the Key Wrap tab.

If you selected Triple-DES, input 24 bytes of key information in the “Key Data” field, if you selected 2-Key TDES, input 16 bytes of key information, and if you selected DES, input 8 bytes of key information. Click the “Register” button to register the key information entered in the key list. The format of the data entered in the key list is as follows.

- Triple-DES Data Format

Bytes	64-bit	64-bit	64-bit
0-23	DES key data 1	DES key data 2	DES key data 3

- 2-Key TDES Data Format

Bytes	64-bit	64-bit
0-15	DES key data 1	DES key data 2

- DES Data Format

Bytes	64-bit
0-7	DES key data 1

Set information in "provisioning key File Path" and "encrypted provisioning key File Path" of "provisioning key". Set "provisioning key File Path" to **sample.key** in the FITDemos folder and "encrypted provisioning key File Path" to **sample.key_enc.key**.

After specifying the provisioning key file and encrypted provisioning key file, as well as the IV value if necessary, click the [Generate Key File …] button to generate the encrypted key data files key_data.c and key_data.h for input to the R_TSIP_GenerateTdesKeyIndex() function.

7.3 ARC4 User Key Operation

7.3.1 ARC4 User Key Installation Overview

The ARC4 user key installation procedure is described below.

The ARC4 user key comprises three keys, each consisting of 56 bits of data generated on the user's PC.

Each user's ARC4 user key has a unique value.

Follow the procedure described below to install the user key. Also, ensure that all processing shown in the flowchart below for writing the user key to the on-chip flash memory of the RX MCU is performed in a secure site (such as a plant operated directly by the customer).

The user key is written to the data flash in the form of user key index. Recovering the user key from the user key index can only be performed internally by the TSIP. This data is not software accessible.

The user key is recovered internally by the TSIP when the user key index is input via the various API functions. Since the user key index has been encrypted using device-specific information, it is not possible to generate correct decryption or encryption results by copying the user key index in the data flash to another TSIP-equipped RX MCU. In addition, the TSIP will not operate correctly if an incorrect user key index is input to the TSIP.

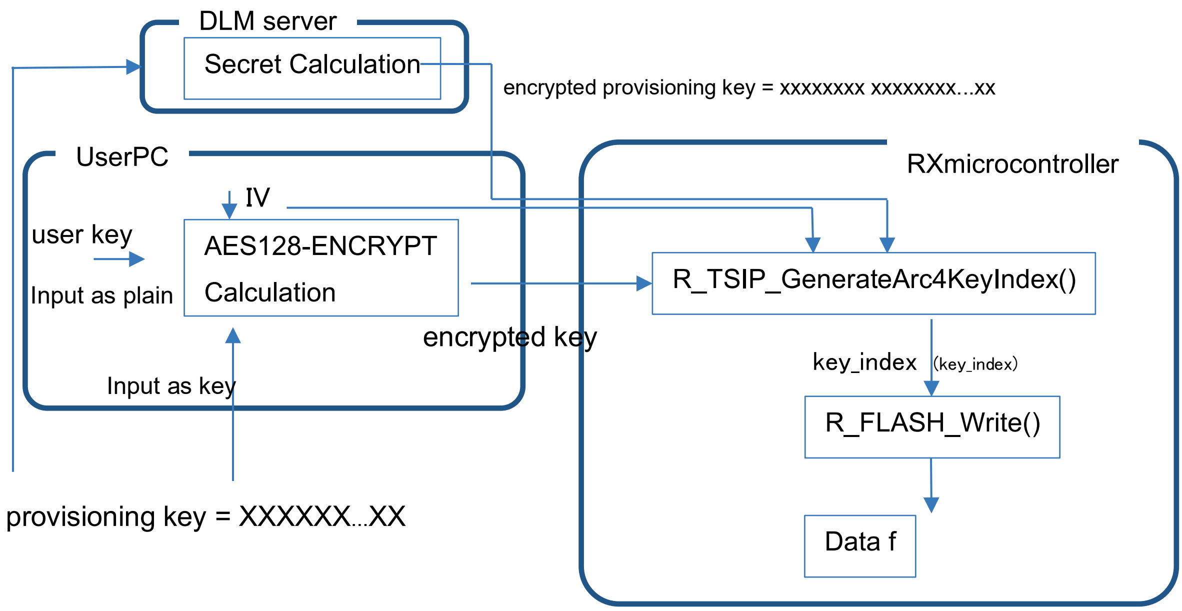


Figure 7.5 ARC4 User Key Installation

An example of generation of a user key on the user's PC is presented on the following pages. It is assumed that the user's PC is running Microsoft Windows.

Renesas Secure Flash Programmer is used to generate the user key.

7.3.2 ARC4 User Key “encrypted key” Creation Method

Launch Renesas Secure Flash Programmer.

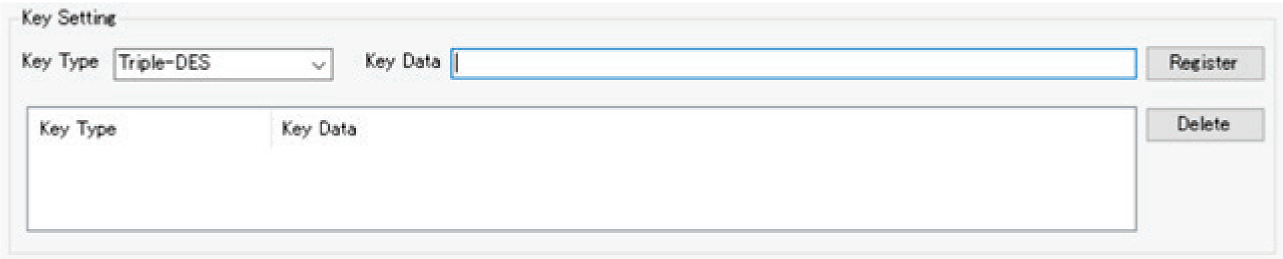


Figure 7.6 Renesas Secure Flash Programmer (Key Wrap Tab, ARC4 Key Setting)

Enter user key settings in the Key Wrap tab.

Here we will make settings for outputting keys (ARC4) that a TDES user can use freely.

Select ARC4-2048bit under “Key Type” on the Key Wrap tab.

Input 256 bytes of key information in the “Key Data” field. Click the “Register” button to register the key information entered in the key list. The format of the data entered in the key list is as follows.

- ARC4 Data Format

Bytes	2048-bit
0-255	ARC4 key data 1

Set information in "provisioning key File Path" and "encrypted provisioning key File Path" of "provisioning key". Set "provisioning key File Path" to **sample.key** in the FITDemos folder and "encrypted provisioning key File Path" to **sample.key_enc.key**.

After specifying the provisioning key file and encrypted provisioning key file, as well as the IV value if necessary, click the [Generate Key File …] button to generate the encrypted key data files key_data.c and key_data.h for input to the R_TSIP_GenerateArc4KeyIndex() function.

7.4 HMAC User Key Utilization

7.4.1 HMAC User Key Installation Overview

The HMAC user key installation procedure is described below.

The HMAC user key comprises three keys, each consisting of 256 bits of data generated on the user's PC.

Each user's HAMC user key has a unique value.

Follow the procedure described below to install the user key. Also, ensure that all processing shown in the flowchart below for writing the user key to the on-chip flash memory of the RX MCU is performed in a secure site (such as a plant operated directly by the customer).

The user key is written to the data flash in the form of user key index. Recovering the user key from the user key index can only be performed internally by the TSIP. This data is not software accessible.

The user key is recovered internally by the TSIP when the user key index is input via the various API functions. Since the user key index has been encrypted using device-specific information, it is not possible to generate correct decryption or encryption results by copying the user key index in the data flash to another TSIP-equipped RX MCU. In addition, the TSIP will not operate correctly if an incorrect user key index is input to the TSIP.

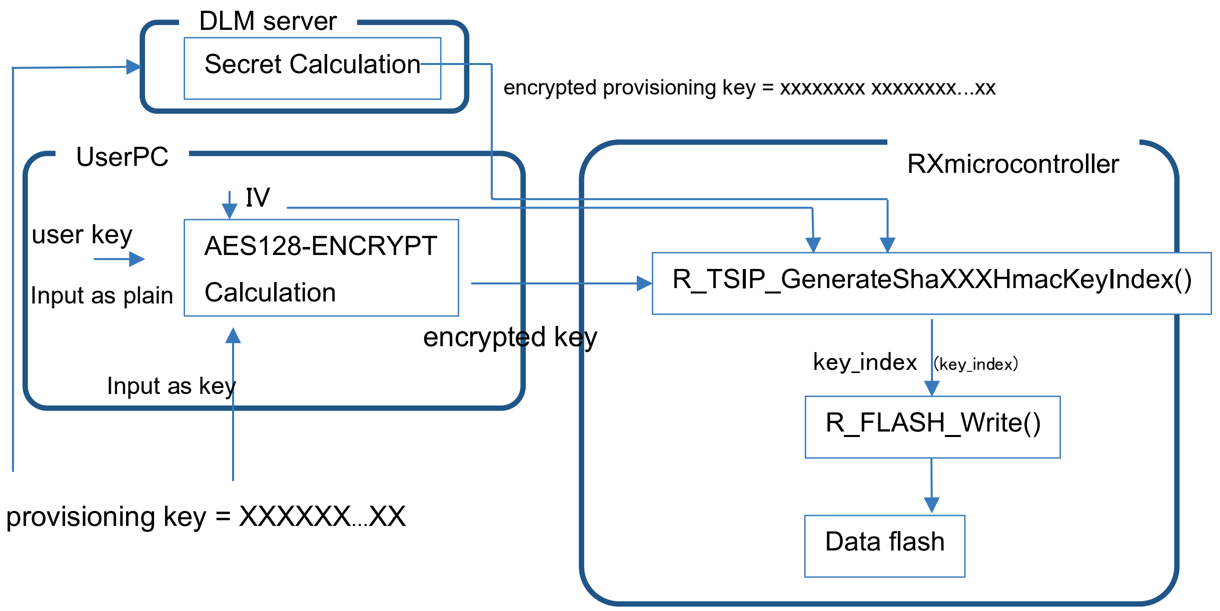


Figure 7.1 HMAC User Key Installation

An example of generation of a user key on the user's PC is presented on the following pages. It is assumed that the user's PC is running Microsoft Windows.

Renesas Secure Flash Programmer is used to generate the user key.

7.4.2 HMAC User Key (encrypted key) Generation

Launch Renesas Secure Flash Programmer.

Figure 7.2 Renesas Secure Flash Programmer (Key Wrap Tab, SHA256-HMAC Key Setting)

Enter user key settings in the Key Wrap tab.

Here we will make settings for outputting keys (SHA-1,SHA-256) that a TDES user can use freely.

Select SHA1-HAMC or SHA256-HMACunder “Key Type” on the Key Wrap tab.

Input 20 bytes of key information in the “Key Data” field for SHA1-HAMC. Input 32 bytes of key information in the “Key Data” field for SHA1-HAMC. Click the “Register” button to register the key information entered in the key list. The format of the data entered in the key list is as follows.

- SHA1-HMAC Data Format

bytes	160bit
0-19	SHA1-HMAC key data

- SHA256-HMAC Data Format

bytes	256bit
0-31	SHA256-HMAC key data

Set information in "provisioning key File Path" and "encrypted provisioning key File Path" of "provisioning key". Set "provisioning key File Path" to **sample.key** in the FITDemos folder and "encrypted provisioning key File Path" to **sample.key_enc.key**.

After specifying the provisioning key file and encrypted provisioning key file, as well as the IV value if necessary, click the [Generate Key File …] button to generate the encrypted key data files key_data.c and key_data.h for input to the R_TSIP_GenerateShaXXXHmacKeyIndex() function.

7.5 RSA Public Key and Private Key Operation

7.5.1 RSA Public Key and Private Key Installation Overview

The method of installing RSA public and private keys is shown below.

Install public and private keys in accordance with this installation procedure. In addition, until the public and private keys are written to the RX microcontroller's internal data flash memory in the course of following the processing flow below, be sure to perform all processing in a safe location (for example, a factory under the direct management of the user's company).

The user key is written to the data flash in the form of user key index. Recovering a private key from this private key user key index is only possible from within TSIP. It cannot be accessed in purely software form.

By inputting the public key user key index and private key user key index to the respective APIs, user keys are recovered from within TSIP. Since private key user key index is encrypted using device-specific information, if the private key user key index in data flash memory is copied to and used on a different RX microcontroller with built-in TSIP, it will not yield correct encryption and decryption results. In addition, if invalid private key user key index is input to TSIP, it will not operate properly.

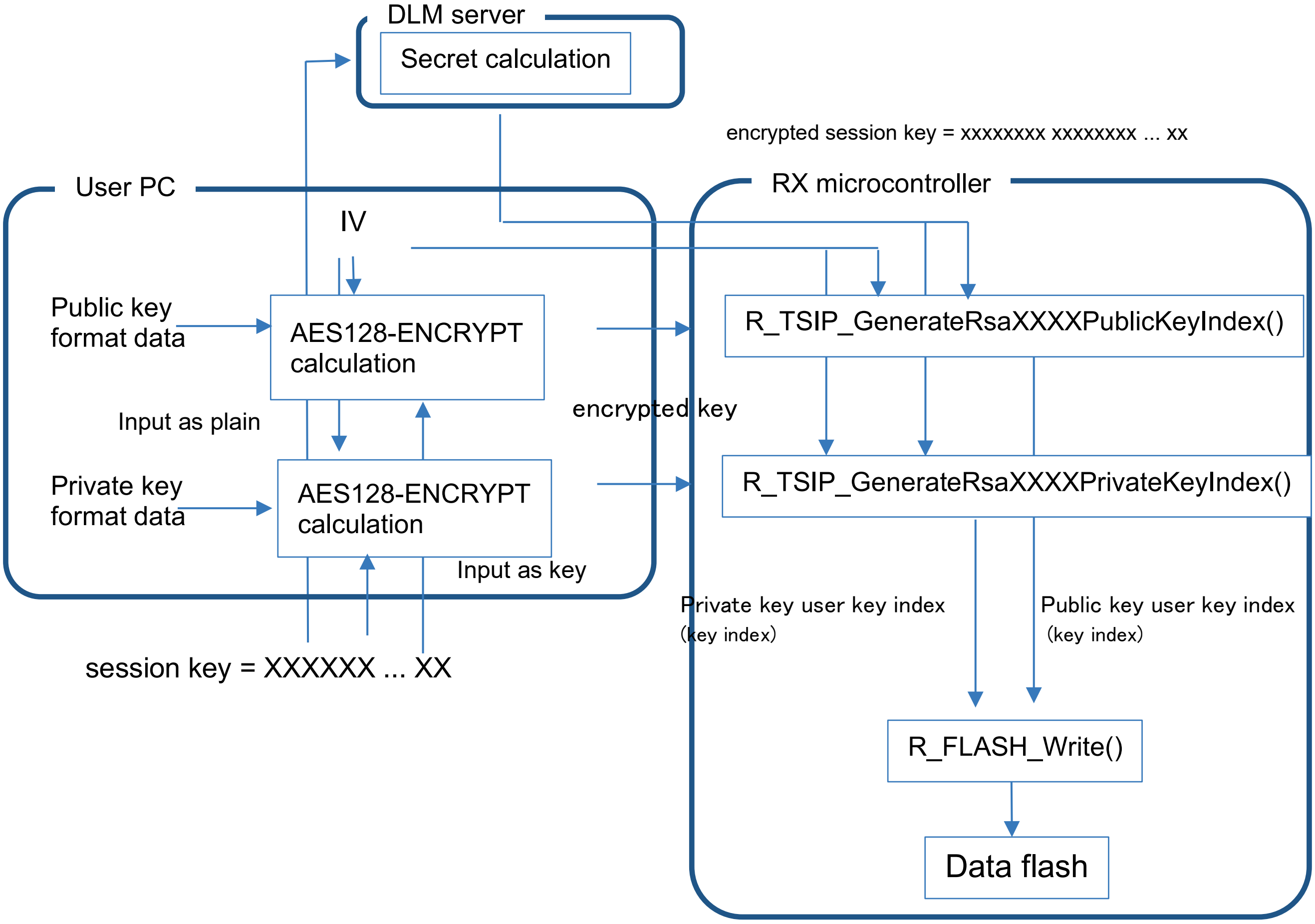


Figure 7.7 RSA Public Key and Private Key Installation Method

- public key format data

byte	128-bit			
	32-bit	32-bit	32-bit	32-bit
1024-bit: 0 to 127 2048-bit: 0 to 255	1024/2048-bit RSA public key n			
1024-bit: 128 to 143 2048-bit: 256 to 271	1024/2048-bit RSA public key e	Zero-padding		

- private key format data

	128-bit			
	32-bit	32-bit	32-bit	32-bit
1024-bit: 0 to 127 2048-bit: 0 to 255	1024/2048-bit RSA public key n			
1024-bit: 128 to 255 2048-bit: 256 to 511	1024/2048-bit RSA private key d			

An example of the method in which public and private key information is generated on a user PC is shown on the next page. The user PC being used is a Windows PC.

Renesas Secure Flash Programmer is used to generate the public and private keys.

7.5.2 RSA Public Key and Private Key “encrypted key” Creation Method

Launch the Renesas Secure Flash Programmer at the path below.

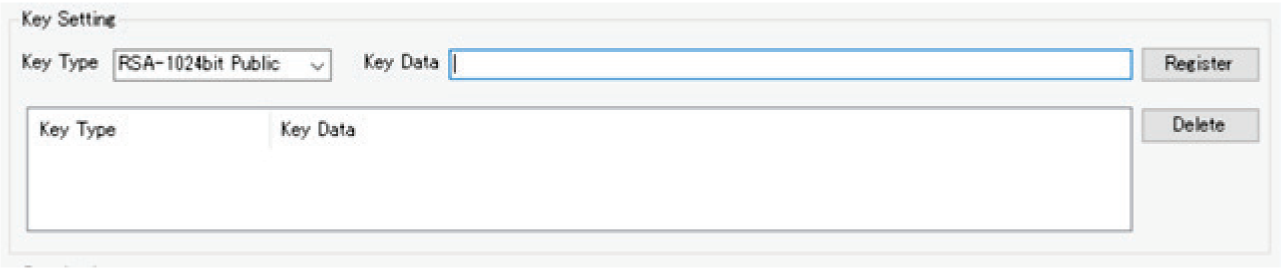


Figure 7.8 Renesas Secure Flash Programmer (Key Wrap Tab, RSA 1024-bit Public Key Setting)

Enter user key settings in the Key Wrap tab.

Here we will make settings for outputting keys (RSA 1024-bit public/private/All and RSA 2048-bit public/private/All) that an RSA user can use freely.

Select RSA 1024-bit public, RSA 1024-bit private, RSA 1024-bit All, RSA 2048-bit public, RSA 2048-bit private, or RSA-2048 bit All under “Key Type” on the Key Wrap tab.

In the Key Data field, enter 132 bytes of key information for RSA 1024-bit public, 256 bytes of key information for RSA 1024-bit private, 260 bytes of key information for RSA 1024-bit all, 260 bytes of key information for RSA 2048-bit public, 512 bytes of key information for RSA 2048-bit private, or 516 bytes of key information for RSA 2048-bit all. Click the Register button to register the key information input in the key list. (When RSA XXXX-bit all is selected, RSA XXXX-bit public and RSA XXXX-bit private are registered separately.) The data formats for inputting data to the key list are shown below. If the key data is of less than the specified bit length, use 0 padding of the higher-order bits. For example, to use a value of 0x10001 for public key e, input 0x00, 0x01, 0x00, 0x01.

- RSA 1024-Bit Public Data Format

Bytes	1024-bit	32-bit
0-131	128-byte RSA public key n data	4-byte RSA public key e data

- RSA 1024-Bit Private Data Format

Bytes	1024-bit	1024-bit
0-255	128-byte RSA public key n data	128-byte RSA private key d data

- RSA 1024-Bit All Data Format

Bytes	RSA 1024-bit Public key n	RSA 1024-bit Public key e	RSA 1024-bit Private key d
0-259	128-byte RSA public key n data	4-byte RSA public key e data	128-byte RSA private key d data

- RSA 2048-bit Public Data Format

Byte	2048-bit	32-bit
0-259	256-byte RSA public key n data	4-byte RSA public key e data

- RSA 2048-bit Private Data Format

Byte	2048-bit	2048-bit
0-511	256-byte RSA public key n data	256-byte RSA private key d data

- RSA 2048-Bit All Data Format

Bytes	RSA 2048-bit Public key n	RSA 2048-bit Public key e	RSA 2048-bit Private key d
0-515	256-byte RSA public key n data	4-byte RSA public key e data	256-byte RSA private key d data

Set information in "provisioning key File Path" and "encrypted provisioning key File Path" of "provisioning key". Set "provisioning key File Path" to **sample.key** in the FITDemos folder and "encrypted provisioning key File Path" to **sample.key_enc.key**.

After specifying the provisioning key file and encrypted provisioning key file, as well as the IV value if necessary, click the [Generate Key File] button to generate the encrypted key (encrypted key) data files key_data.c and key_data.h for input to the R_TSIP_GenerateRsaXXXXPublic/PrivateKeyIndex() function.

7.6 ECC Public Key and Private Key Operation

7.6.1 ECC Public Key and Private Key Installation Overview

The method of installing ECC public and private keys is shown below.

Install public and private keys in accordance with this installation procedure. In addition, be sure to perform all processing in a safe location (for example, a factory under the direct management of the user's company) until the public and private keys are written to the RX microcontroller's internal data flash memory in the course of the processing sequence shown below.

The user key is written to the data flash in the form of user key index. Recovering a private or public key from the user key index is only possible internally within the TSIP. These cannot be accessed by software.

By inputting a user key index to the appropriate API, a user key is recovered from within the TSIP. Since the user key index is encrypted using device-specific information, if the user key index in the data flash memory is copied to and used on a different RX microcontroller with a built-in TSIP, it will not yield correct encryption and decryption results. In addition, if invalid private key user key index is input to the TSIP, it will not operate properly.

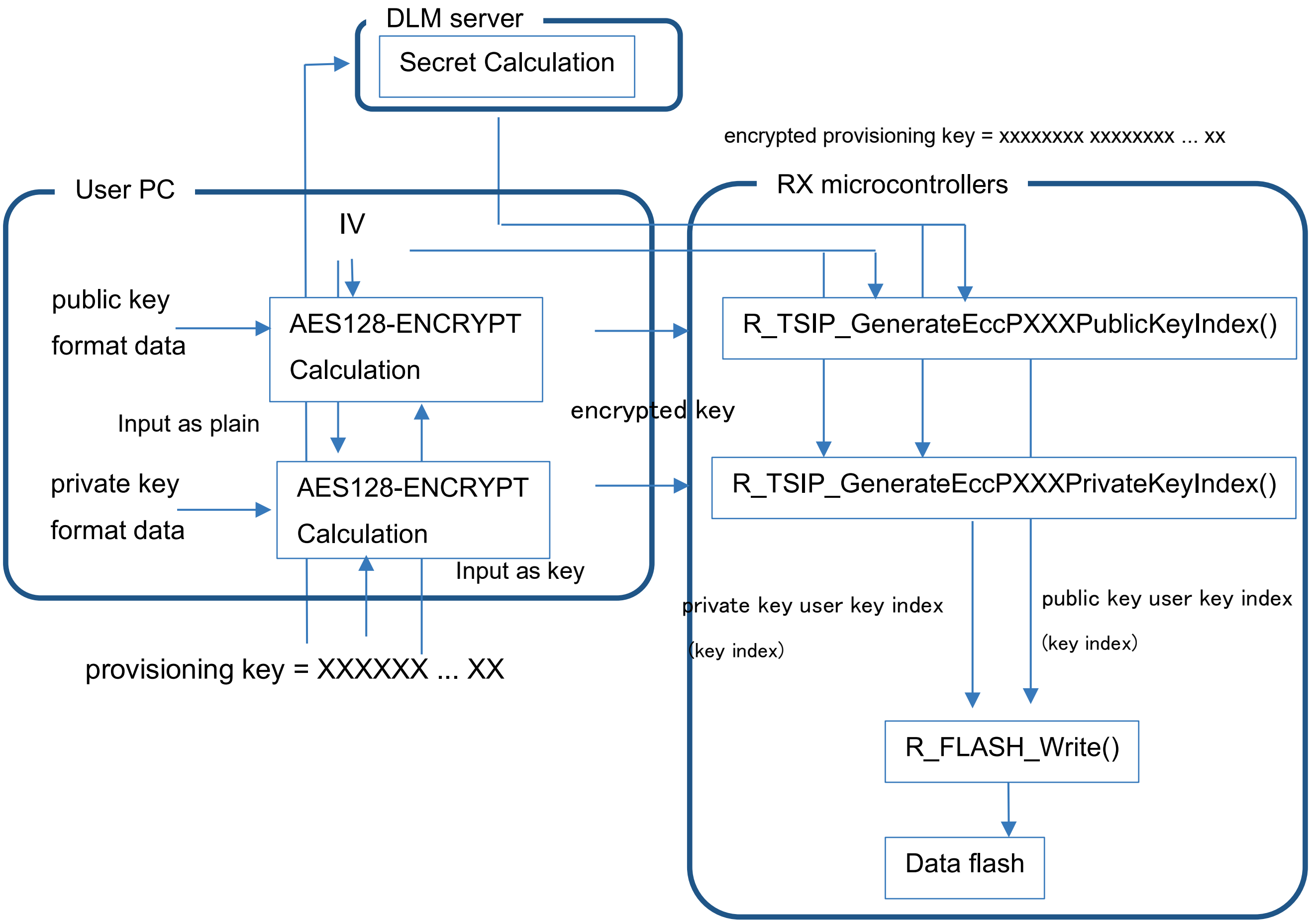


Figure 7.9 ECC Public Key and Private Key Installation Method

- Public key format data

Bytes	128 bits			
	32 bits	32 bits	32 bits	32 bits
0-31 Note 1	0 padding (required for 192 or 224 bits) || ECC 192-, 224, 256, or 384-bit public key Qx			
32-63 Note 2	0 padding (required for 192 or 224 bits) || ECC 192-, 224, 256,or 384-bit public key Qy			

Notes: 1. Applies to ECC-192, ECC-224, and ECC-256. Bytes 0–47 for ECC-384.
2. Applies to ECC-192, ECC-224, and ECC-256. Bytes 48–95 for ECC-384.

- Private key format data

Bytes	128 bits			
	32 bits	32 bits	32 bits	32 bits
0-31 Note 1	0 padding (required for 192 or 224 bits) || ECC 192-, 224, 256, or 384-bit private key			

An example of the method whereby public and private key information is generated on a user PC is shown on the next page. The user PC used is running Microsoft Windows.

Renesas Secure Flash Programmer is used to generate the public and private keys.

7.6.2 ECC Public Key and Private Key “encrypted key” Creation Method

Launch Renesas Secure Flash Programmer.

Figure 7.10 Renesas Secure Flash Programmer (Key Wrap Tab, ECC 256-bit public Key Setting)

Enter user key settings in the Key Wrap tab.

Here we will make settings for outputting keys (ECC 192-bit public/private/all, ECC 224-bit public/private/all, ECC 256-bit public/private/all and , ECC-384bit Public/Private/All) that an ECC user can use freely.

Select ECC 192-bit public, ECC 192-bit private, ECC 192-bit all, ECC 224-bit public, ECC 224-bit private, ECC 224-bit all, ECC 256-bit public, ECC 256-bit private, ECC 256-bit all, ECC-384bit Public, ECC-384bit Private and ECC-384bit All on the Key Wrap tab.

As key data, input key information with the number of bytes listed below for the appropriate data format. Click the Register button to register the entered key information in the key list. (The registered key information is divided between ECC-XXXbit Public and ECC-XXXbit Private when ECC-XXXbit All is selected.) The supported data formats for key list input are shown below.

- 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

- ECC 192-Bit Pravate Data Format (24 bytes)

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

- 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 private key data

- ECC 224-Bit Public Data Format (56 bytes)

byte	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

- ECC 224-Bit Private Data Format (28 bytes)

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

- ECC 224-Bit All Data Format (84 bytes)

byte	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 private key data

- ECC 256-Bit Public Data Format (64 bytes)

byte	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

- ECC 256-Bit Private Data Format (32 bytes)

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

- ECC 256-Bit All Data Format (96 bytes)

byte	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 private key data

- ECC 384-Bit Public Data Format (96 bytes)

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

- ECC 256-Bit Private Data Format (48 bytes)

Byte	ECC-384bit Private key
0-47	48-byte ECC private key data

- ECC 256-Bit All Data Format (144 bytes)

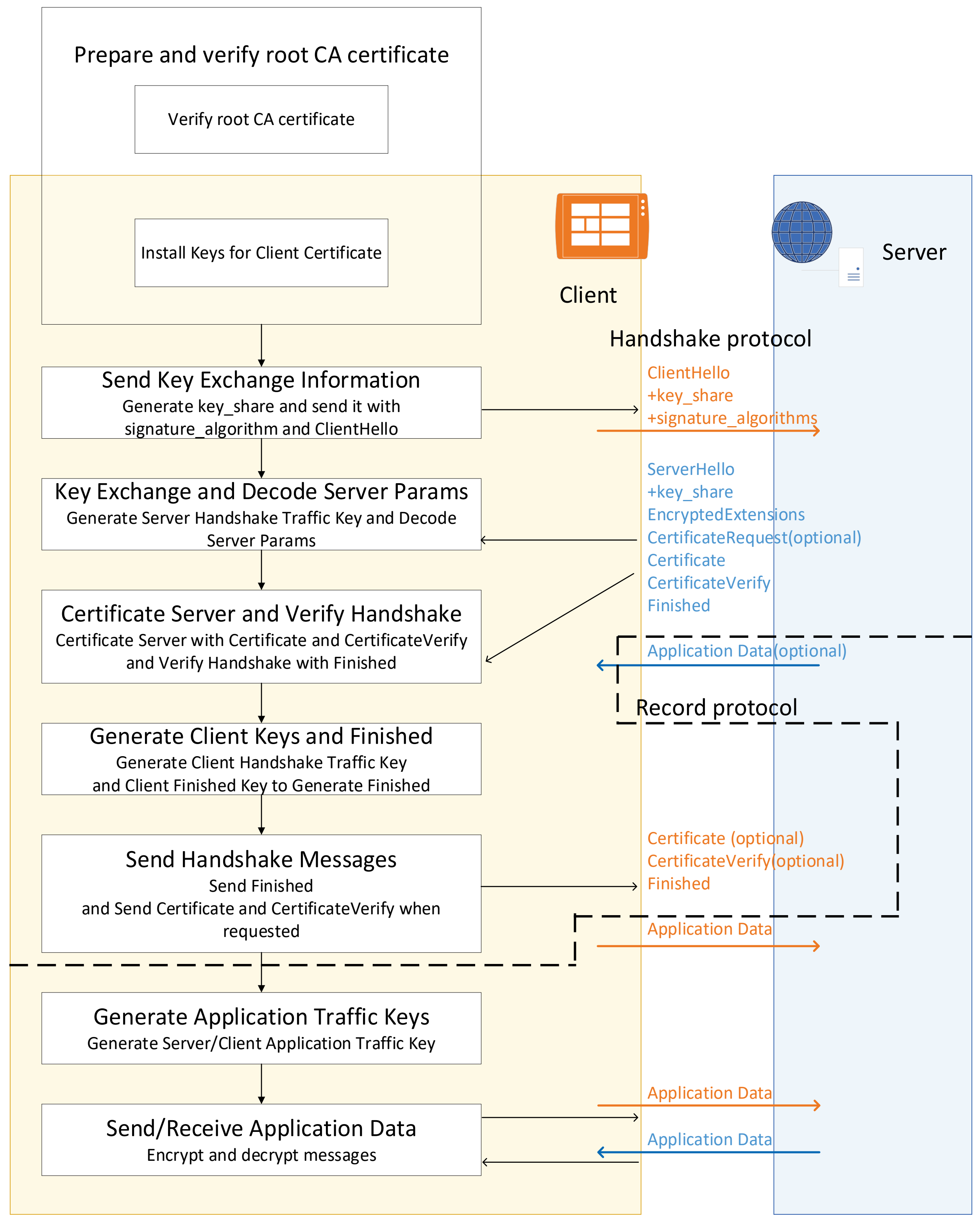
byte	ECC-384bit Public key Qx	ECC-384bit Public key Qy	ECC-384bit Private key
0-143	48-byte ECC public key Qx data	48-byte ECC public key Qy data	48-byte ECC private key data

Set information in "provisioning key File Path" and "encrypted provisioning key File Path" of "provisioning key". Set "provisioning key File Path" to **sample.key** in the FITDemos folder and "encrypted provisioning key File Path" to **sample.key_enc.key**.

After specifying the provisioning key file and encrypted provisioning key file, as well as the IV value if necessary, click the [Generate Key File…] button to generate the encrypted key (encrypted key) data files key_data.c and key_data.h for input to the R_TSIP_GenerateEccXXXXPublic/PrivateKeyIndex() function.

8. TLS Cooperation Function : Scheme to Use TLS Cooperation Function (TLS1.3)

The method to use TLS1.3 of TLS cooperation function is shown below.



* To assure validity, the TSIP only accepts a key index, except a public key (this feature is going to be updated).

! Keep the key in a safe place.

Figure 10-1 Scheme to use TLS1.3 cooperation function

8.1 Prepare and verify root CA certificate

The method to prepare root CA certificate and install key of the certificate is same to TLS1.2 cooperation function. Please refer to “How to implement TLS with TSIP driver (R01AN5880xJxxxx)”.

8.2 Send Key Exchange Information

1. Generate ECDH public key with R_TSIP_GenerateTls13P256EccKeyIndex().
2. Send ECDH public key to the server as key_share field with signature_algorithm field when sending ClientHello.

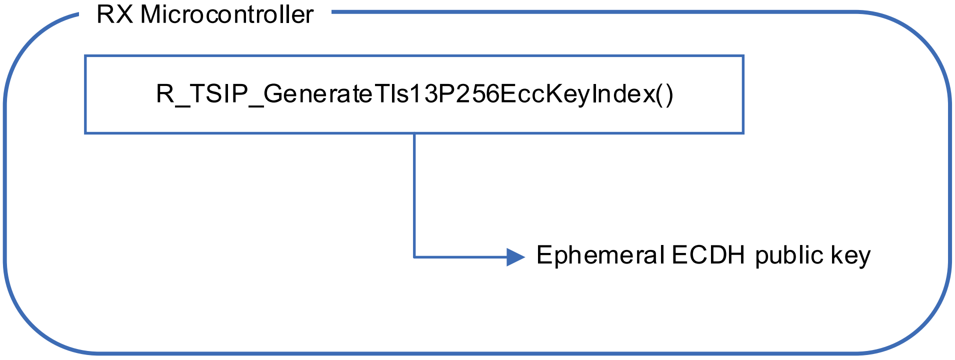


Figure 10-2 Send Key Exchange Information

8.3 Key Exchange and Decode Server Params

1. Generate a SharedSecret key index with inputting the public key received from the server to R_TSIP_Tls13GenerateSharedSecret().
2. Generate a HandshakeSecret key index with inputting the SharedSecret key index to R_TSIP_Tls13GenerateHandshakeSecret().
3. Generate a ServerWriteKey key index and a ServerFinishedKey key index with inputting the HandshakeSecret key index to R_TSIP_Tls13GenerateServerHandshakeTrafficKey().
4. Decode the encrypted handshake messages received from the server with inputting the ServerWriteKey key index to R_TSIP_Tls13DecryptInit/Update/Final().

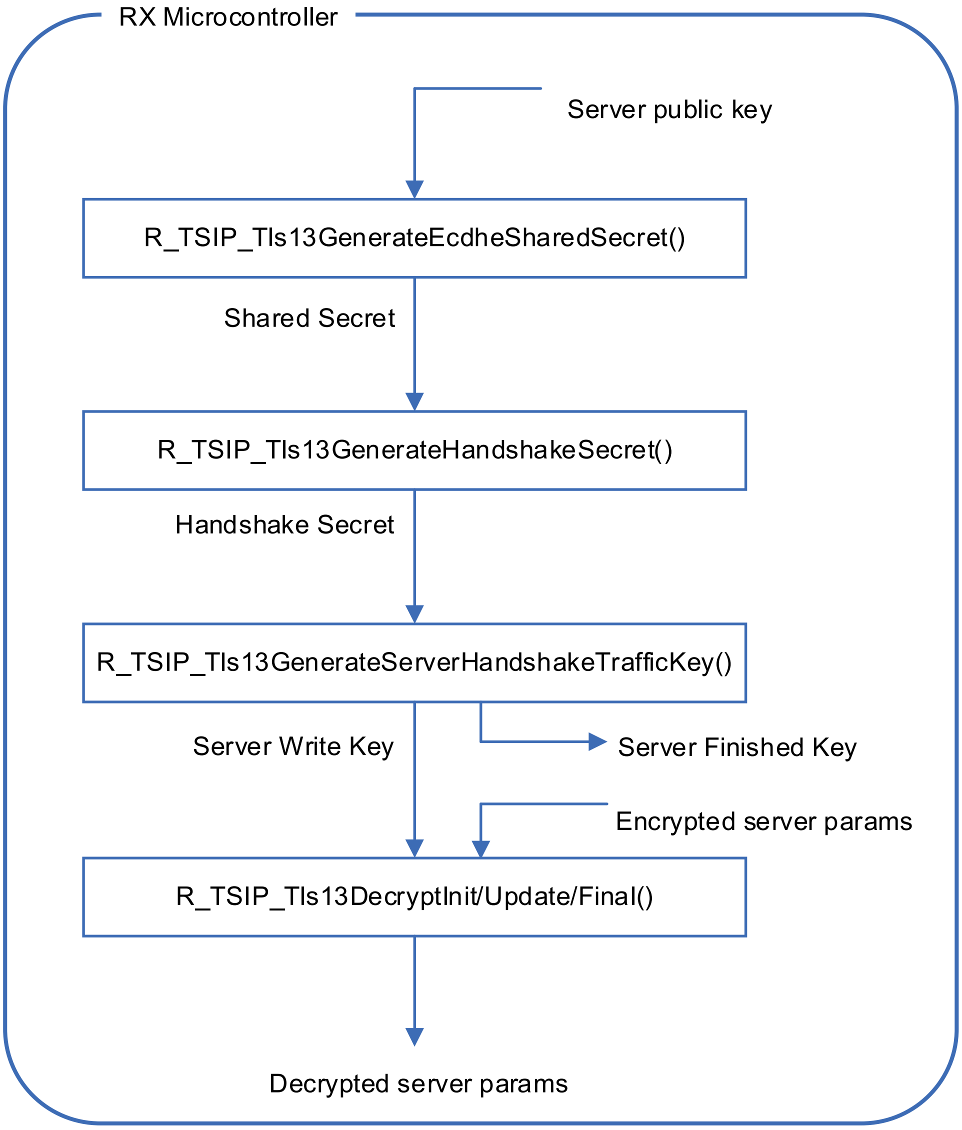


Figure 10-3 Key Exchange and Decode Server Params

8.4 Certificate Server and Verify Handshake

1. Certificate the server by verifying (Server) Certificate field and (Server) CertificateVerify field obtained by decoding with R_TSIP_Tls13DecryptInit/Update/Final(). R_TSIP_Tls13CertificateVerifyVerification() can be used to verify signature of (Server) CertificateVerify field.
2. Verify the handshake with inputting (Server) Finished field obtained by decoding with R_TSIP_Tls13DecryptInit/Update/Final() and the ServerFinishedKey key index to R_TSIP_Tls13ServerHandshakeVerification(). The result of verifying the handshake is outputted as verify_data_index.

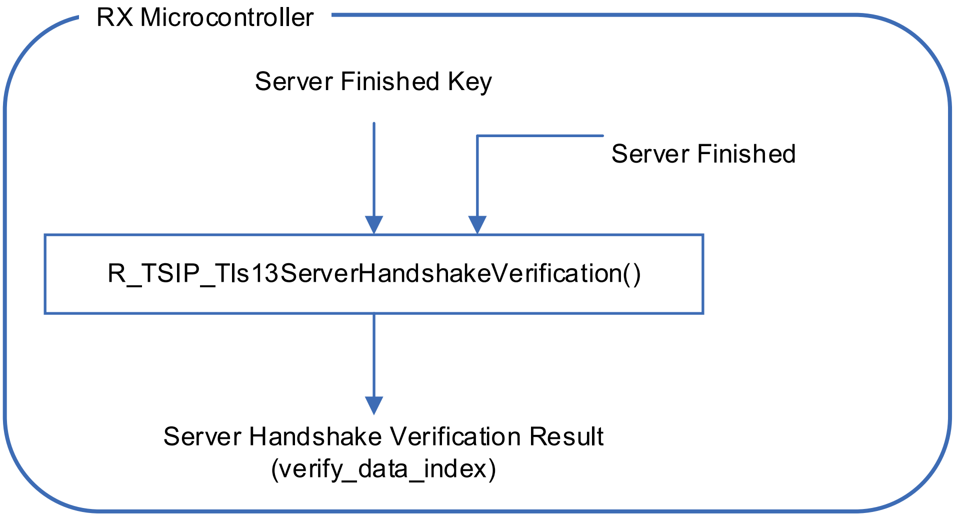


Figure 10-4 Certificate Server and Verify Handshake

8.5 Generate Client Keys and Finished

1. Generate a ClientWriteKey key index and a ClientFiishedKey key index with inputting the HandshakeSecret key index to R_TSIP_Tls13GenerateClientHandshakeTrafficKey(). These keys are used in handshake phase.
2. Generate (Client) Certificate field and (Client) CertificateVerify field when received CertificateRequest from the server. R_TSIP_Tls13CertificateVerifyGenerate() can be used to generate signature of (Client) CertificateVerify field.
3. Generate (Client) Finished field with inputting the ClientFinishedKey key index to R_TSIP_Sha256HmacGenerateInit/Update/Final().

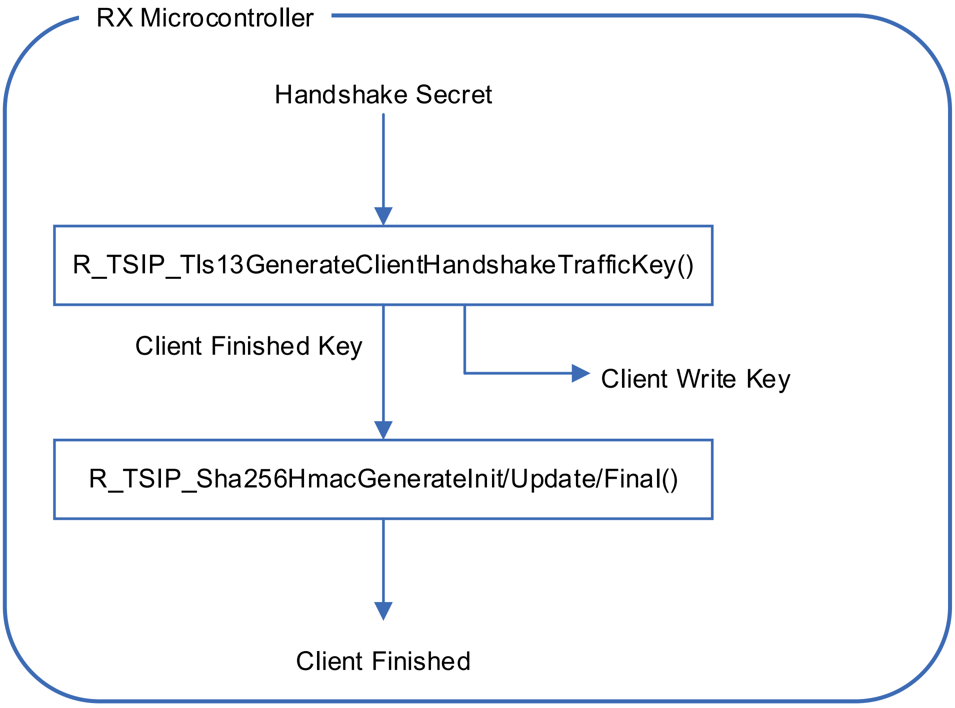


Figure 10-5 Generate Client Keys and Finished

8.6 Send Handshake Message

1. Encode handshake messages to send the server with inputting the ClientWriteKey key index to R_TSIP_Tls13EncryptInit/Update/Final().
2. Send encrypted handshake messages.

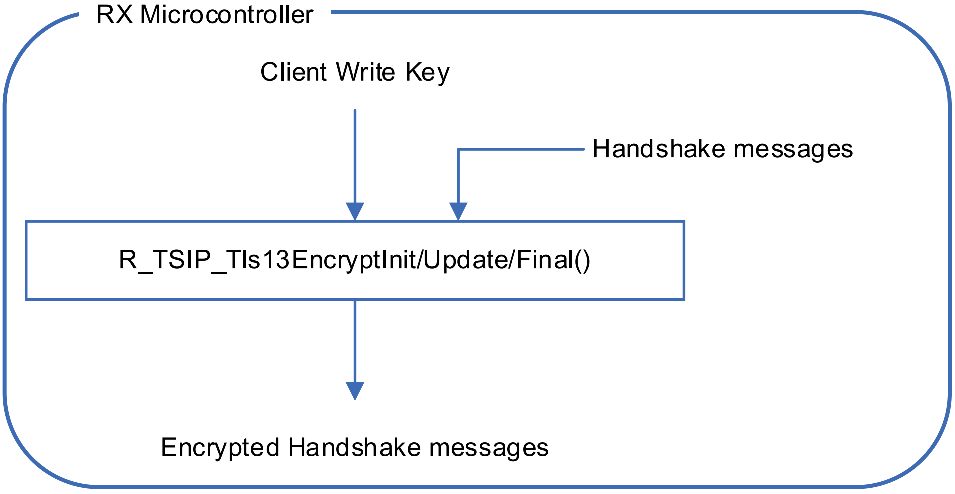


Figure 10-6 Send Handshake Messages

8.7 Generate Application Traffic Keys

1. Generate a MasterSecret key index with inputting the HandshakeSecret key index and verify_data_index to R_TSIP_Tls13GenerateMasterSecret().
2. Generate a ServerWriteKey key index, a ClientWriteKey key index and each ApplicationSecrets key index with inputting the MasterSecret key index to R_TSIP_Tls13GenerateApplicationTrafficKey().
3. R_TSIP_Tls13UpdateApplicationTrafficKey() can be used when updating a ServerWriteKey or a ClientWriteKey.

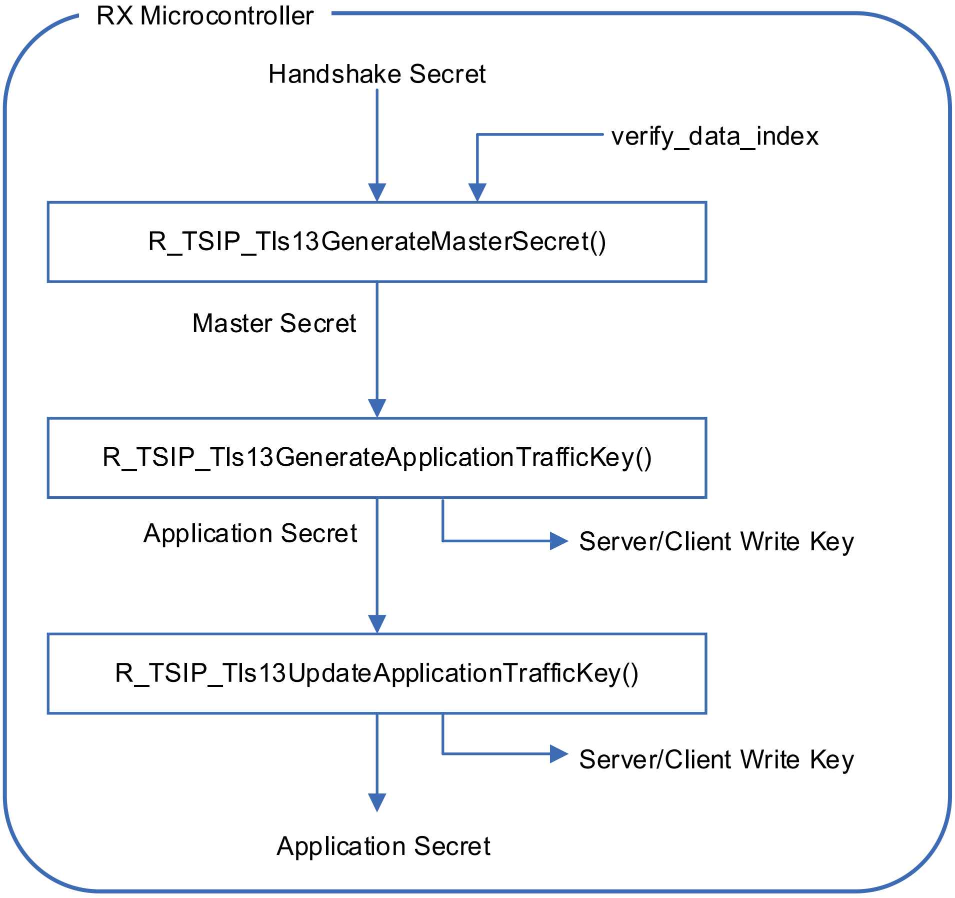


Figure 10-7 Generate Application Traffic Keys

8.8 Send/Receive Application Data

1. R_TSIP_Tls13DecryptInit/Update/Final() is used with the ServerWriteKey key index when decoding data received from the server.
2. R_TSIP_Tls13EncryptInit/Update/Final() is used with the ClientWriteKey key index when encoding data sending to the server.

9. Appendix

9.1 Confirmed Operation Environment

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

Table 9.1 Confirmed Operation Environment

Item	Description
Integrated development environment	Renesas Electronics e^2 studio 2021-07 IAR Embedded Workbench for Renesas RX 4.20.01
C compiler	Renesas Electronics C/C++ Compiler for RX Family (CC-RX) V3.03.00 Compile options: The following option has been added to the default settings of the integrated development environment. -lang = c99
	GCC for Renesas RX 8.3.0.202102 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 Compile 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/little endian
Module version	Ver.1.14
Board used	Renesas Starter Kit for RX231 (B version) (product No.: R0K505231S020BE) Renesas Solution Starter Kit for RX23W (with TSIP) (product No.: RTK5523W8BC00001BJ) 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.: RTK55671xxxxxxxxx) Renesas Starter Kit+ for RX72M (with TSIP) (product No.: RTK5572NNHC00000BJ) Renesas Starter Kit+ for RX72N (with TSIP) (product No.: RTK5572NNHC00000BJ) Renesas Starter Kit for RX72T (with TSIP) (product No.: RTK5572TKCS00010BE)

9.2 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 if the method for adding FIT modules:

- Using CS+
  Application note: “RX Family: Adding Firmware Integration Technology Modules to CS+ Projects” (R01AN1826)
- Using e^2 studio
  Application note: “RX Family: 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 “Yes” in the [Q0268002] dialog box, you must then re-input the compiler include path.

10. Reference Documents

User's Manual: Hardware
User's Manual: Hardware
(The latest versions can be downloaded from the Renesas Electronics website.)

Technical Update/Technical News
(The latest versions can be downloaded from the Renesas Electronics website.)

User's Manual: Development Environment
RX Family CC-RX Compiler User's Manual (R20UT3248)
(The latest versions can be downloaded from the Renesas Electronics website.)

Website and Support

Renesas Electronics Website
https://www.renesas.com/jp/ja/
Inquiries
https://www.renesas.com/jp/ja/support/contact.html

Revision History

Rev.	Date	Description Page	Summary
1.00	Jul 10, 2020	-	First release.
1.11	Dec. 31, 2020		• Added ECC P-384 key installation, key generation, and key update functions • Added ECDSA P-384 functions • Added support for RX72M, RX66N, and RX72N to key exchange function • Changed name of ECDH key exchange function R_TSIP_EcdhXXX() to R_TSIP_EcdhP256XXX() • Modified ECC public key structure tsip_ecc_public_key_index_t • Changed R_TSIP_AesXXXKeyWrap() and R_TSIP_AesXXXKeyUnwrap() to common APIs to both TSIP and TSIP-Lite • Deleted configuration description • Unified descriptions of iv parameter of R_TSIP_GenerateXXXKeyIndex() and R_TSIP_UpdateXXXKeyIndex() • Listed TSIP_ERR_FAIL in return values of all AES Init functions • Deleted text related to TSIP_USER_HASH_ENABLED • Changed the version numbers of the development environments to those used during development • Changed the order in which device names are listed 1.2 In the product configuration table, removed the mdf file, secure_boot projects, rsk_tsip_rfp_project, and rsk_usb_serial_driver, and added RX72N project 1.4 to 1.12 Listed current version information 1.5 Removed secure boot description 2.2 Changed version number of r_bsp 3.4 Corrected spelling of TSIP_ERR_RESOURCE_CONFLICT 4.14 Removed examples of implementing secure updates using USB memory 4.40, 4.43 Added information on differences in handling of IV for different key_index->type values 5.29 Change plain_length description of arguments 5.32 Change cipher_length description of arguments 5.52 Description of the R_TSIP_Rsa2048DhKeyAgreement function was relocated. 5.113 Changed the name of argument algorithm_id to key_type, that include setting value change, and added the kdf_type and salt_key_index to argument. Deleted TSIP_ERR_FAIL in return value.
1.12	Jun. 31, 2021		• Updated version of development environment to the used version in development • Revised the explanation of AES-GCM and RSA decryption 1.2 Added the sample indicates how to use AES cryptograpy and how to implement TLS in the table of Structure of Product Files 1.4 to 1.12 Listed current version information

1.13	Aug. 31, 2021	• Added support for RX671 • Updated version of development environment to the used version in development • Added HMAC user key. 1.2 Added Trusted Secure IP(TSIP) 1.3 Updated Structure of Produte File Table. 1.5~1.14 Updated the information to this version 2.2 Updated r_bsp version. 3.2 Updated State Transition Diagram 5.38, 5.39, 5.85, 5.86, 5.87, 5.88 Updated description. 7.1.1, 7.2.1, 7.3.1, 7.4.1, 7.5.1, 7.6.1 Updated description.
1.14	Oct. 22, 2021	• Added support fot TLS1.3 cooperation function (only RX65N)

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

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

1. Precaution against Electrostatic Discharge (ESD)
   A strong electrical field, when exposed to a CMOS device, can cause destruction of the gate oxide and ultimately degrade the device operation. Steps must be taken to stop the generation of static electricity as much as possible, and quickly dissipate it when it occurs. Environmental control must be adequate. When it is dry, a humidifier should be used. This is recommended to avoid using insulators that can easily build up static electricity. Semiconductor devices must be stored and transported in an anti-static container, static shielding bag or conductive material. All test and measurement tools including work benches and floors must be grounded. The operator must also be grounded using a wrist strap. Semiconductor devices must not be touched with bare hands. Similar precautions must be taken for printed circuit boards with mounted semiconductor devices.
2. Processing at power-on
   The state of the product is undefined at the time when power is supplied. The states of internal circuits in the LSI are indeterminate and the states of register settings and pins are undefined at the time when power is supplied. In a finished product where the reset signal is applied to the external reset pin, the states of pins are not guaranteed from the time when power is supplied until the reset process is completed. In a similar way, the states of pins in a product that is reset by an on-chip power-on reset function are not guaranteed from the time when power is supplied until the power reaches the level at which resetting is specified.
3. Input of signal during power-off state
   Do not input signals or an I/O pull-up power supply while the device is powered off. The current injection that results from input of such a signal or I/O pull-up power supply may cause malfunction and the abnormal current that passes in the device at this time may cause degradation of internal elements. Follow the guideline for input signal during power-off state as described in your product documentation.
4. Handling of unused pins
   Handle unused pins in accordance with the directions given under handling of unused pins in the manual. The input pins of CMOS products are generally in the high-impedance state. In operation with an unused pin in the open-circuit state, extra electromagnetic noise is induced in the vicinity of the LSI, an associated shoot-through current flows internally, and malfunctions occur due to the false recognition of the pin state as an input signal become possible.
5. Clock signals
   After applying a reset, only release the reset line after the operating clock signal becomes stable. When switching the clock signal during program execution, wait until the target clock signal is stabilized. When the clock signal is generated with an external resonator or from an external oscillator during a reset, ensure that the reset line is only released after full stabilization of the clock signal. Additionally, when switching to a clock signal produced with an external resonator or by an external oscillator while program execution is in progress, wait until the target clock signal is stable.
6. Voltage application waveform at input pin
   Waveform distortion due to input noise or a reflected wave may cause malfunction. If the input of the CMOS device stays in the area between V_{IL} (Max.) and V_{IH} (Min.) due to noise, for example, the device may malfunction. Take care to prevent chattering noise from entering the device when the input level is fixed, and also in the transition period when the input level passes through the area between V_{IL} (Max.) and V_{IH} (Min.).
7. Prohibition of access to reserved addresses
   Access to reserved addresses is prohibited. The reserved addresses are provided for possible future expansion of functions. Do not access these addresses as the correct operation of the LSI is not guaranteed.
8. Differences between products
   Before changing from one product to another, for example to a product with a different part number, confirm that the change will not lead to problems. The characteristics of a microprocessing unit or microcontroller unit products in the same group but having a different part number might differ in terms of internal memory capacity, layout pattern, and other factors, which can affect the ranges of electrical characteristics, such as characteristic values, operating margins, immunity to noise, and amount of radiated noise. When changing to a product with a different part number, implement a system-evaluation test for the given product.

Notice

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

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

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

(Rev.5.0-1 October 2020)

Corporate Headquarters

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

Trademarks

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

Contact information

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