

RX Family

TSIP (Trusted Secure IP) Module Firmware Integration Technology

Introduction

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

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

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

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

Confirmed Devices

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

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

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

Table 1 TSIP Cryptographic Algorithms

Cipher Type		Algorithms
Asymmetric (public key) cryptography	Encryption/ decryption	RSAES-PKCS1-v1_5 (1024-/2048-/3072-/4096-bit)*1: RFC 8017
	Signature generation/ verification	RSASSA-PKCS1-v1_5 (1024-/2048-/3072-/4096-bit)*1: RFC 8017 ECDSA (ECC P-192/P-224/P-256/P-384): FIPS186-4
	Key generation	RSA (1024-/2048-bit) ECC P-192/P-224/P-256/P-384
Symmetric key cryptography	AES	AES (128-/256-bit) ECB/CBC/CTR: FIPS 197, SP800-38A
	DES	TDES (56-/56x2-/56x3-bit) ECB/CBC: FIPS 46-3
	ARC4	ARC4 (2048-bit)
Hashing	SHA	SHA-1, SHA-256: FIPS 180-4
	MD5	MD5: RFC 1321
Authenticated encryption with associated data (AEAD)		GCM/CCM: FIPS 197, SP800-38C, SP800-38D
Message authentication		CMAC (AES): FIPS 197, SP800-38B GMAC: RFC 4543 HMAC (SHA): RFC 2104
Pseudo-random bit generation		SP 800-90A
Random number generation		Tested with SP 800-22.
TLS	TLS 1.2	TLS 1.2: RFC 5246 *2 Supported function: client function/server function Supported cipher suites (TLS 1.2): TLS_RSA_WITH_AES_128_CBC_SHA TLS_RSA_WITH_AES_256_CBC_SHA TLS_RSA_WITH_AES_128_CBC_SHA256 TLS_RSA_WITH_AES_256_CBC_SHA256 TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256 TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256 TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256 TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256
	TLS 1.3	TLS 1.3: RFC 8446 *3 Supported function: client function/server function Full Handshake/Resumption/0-RTT Supported cipher suites (TLS 1.3): TLS_AES_128_GCM_SHA256 TLS_AES_128_CCM_SHA256
Key update functions		AES, RSA, DES, ARC4, ECC, HMAC
Key exchange		ECDH P-256, ECDHE NIST P-512: SP800-56A, SP800-56C DH (2048-bit)
Key wrapping		AES (128-/256-bit)

Notes: 1. RSA (3072-/4096-bit) is supported for signature verification and modular exponentiation using public key only.

2. Applicable devices of the server function are the RX65N and RX651 groups, RX66N group, RX72M group, and RX72N group.

3. Applicable devices of the server function and Resumption and 0-RTT in the client function are the RX65N and RX651 groups, RX66N group, RX72M group, and RX72N group.

Table 2 TSIP-Lite Cryptographic Algorithms

Cipher Type		Algorithms
Symmetric key cryptography	AES	AES (128-/256-bit) ECB/CBC/CTR: FIPS 197, SP800-38A
Authenticated encryption with associated data (AEAD)		GCM/CCM: FIPS 197, SP800-38C, SP800-38D
Message authentication		CMAC (AES): FIPS 197, SP800-38B GMAC: RFC 4543
Pseudo-random bit generation		SP 800-90A
Random number generation		Tested with SP 800-22.
Key update functions		AES
Key wrapping		AES (128-/256-bit)

Note: [RFC 2104: HMAC: Keyed-Hashing for Message Authentication \(rfc-editor.org\)](https://rfc-editor.org/rfc/rfc2104.html)
[RFC 8017: PKCS #1: RSA Cryptography Specifications Version 2.2 \(rfc-editor.org\)](https://rfc-editor.org/rfc/rfc8017.html)
[RFC 4543: The Use of Galois Message Authentication Code \(GMAC\) in IPsec ESP and AH \(rfc-editor.org\)](https://rfc-editor.org/rfc/rfc4543.html)
[RFC 5246: The Transport Layer Security \(TLS\) Protocol Version 1.2 \(rfc-editor.org\)](https://rfc-editor.org/rfc/rfc5246.html)
[RFC 8446: The Transport Layer Security \(TLS\) Protocol Version 1.3 \(rfc-editor.org\)](https://rfc-editor.org/rfc/rfc8446.html)

[FIPS 46-3, Data Encryption Standard \(DES\) \(withdrawn May 19, 2005\) \(nist.gov\)](https://nvlpubs.nist.gov/nistpubs/FIPS/FIPS.46-3.pdf)

FIPS186-4: <https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.186-4.pdf>

[NIST SP 800-38A, Recommendation for Block Cipher Modes of Operation Methods and Techniques](https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-38A.pdf)

[NIST SP 800-38-B Recommendation for Block Cipher Modes of Operation: The CMAC Mode for Authentication \(nist.gov\)](https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-38B.pdf)

[NIST SP 800-38D, Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode \(GCM\) and GMAC](https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-38D.pdf)

[NIST SP800-56A: Recommendation for Pair-Wise Key Establishment Schemes Using Discrete Logarithm Cryptography \(nist.gov\)](https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-56A.pdf)

[NIST SP800-56C: Recommendation for Key-Derivation Methods in Key-Establishment Schemes \(nist.gov\)](https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-56C.pdf)

[NIST SP800-22: https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-22r1a.pdf](https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-22r1a.pdf)

[NIST SP800-90A: https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-90Ar1.pdf](https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-90Ar1.pdf)

Contents

1. Overview.....	12
1.1 Terminology.....	12
1.2 TSIP Overview.....	14
1.3 Structure of Product Files.....	15
1.4 Development Environment.....	17
1.5 Code Size.....	18
1.6 Sections.....	21
1.7 Performance.....	22
1.7.1 RX231.....	22
1.7.2 RX23W.....	25
1.7.3 RX26T.....	28
1.7.4 RX66T, RX72T.....	31
1.7.5 RX65N.....	34
1.7.6 RX671.....	42
1.7.7 RX72M, RX72N.....	50
2. API Information.....	58
3. TSIP Driver Usage.....	64
3.1 Recovering after Unauthorized Access Detection.....	64
3.2 Avoiding TSIP Access Conflicts.....	64
3.3 BSP FIT Module Integration.....	65
3.4 Single-Part and Multi-Part Operations.....	65
3.5 Initializing and Terminating the Driver.....	66
3.6 Random Number Generation.....	66
3.7 Key Management.....	67
3.7.1 Key Injection and Updating.....	68
3.7.2 Key Generation.....	70
3.7.3 Plaintext Public Key Extraction.....	70
3.8 Symmetric Key Cryptography.....	70
3.8.1 Symmetric Key Cryptography.....	71
3.8.2 Authenticated Encryption with Associated Data (AEAD).....	71
3.8.3 Message Authentication Code (MAC).....	71
3.9 Asymmetric Cryptography.....	72
3.10 Hash Functions.....	72
3.10.1 Message Digest (Hash Function).....	72
3.10.2 Message Authentication Code (HMAC).....	73
3.11 ECDH Key Exchange.....	73
3.11.1 Ephemeral Unified Model (2e, 0s).....	74
3.11.2 One-Pass Diffie-Hellman (1e, 1s).....	75

3.11.3	Static Unified Model (0e, 2s)	76
3.12	TLS Cooperation Function (TLS 1.2)	77
3.12.1	TLS 1.2 Client Function	77
3.12.2	TLS 1.2 Server Function	84
3.13	TLS Cooperation Function (TLS 1.3)	91
3.13.1	TLS 1.3 Client Function	91
3.13.1.1	Full Handshake	92
3.13.1.2	Resumption	98
3.13.1.3	0-RTT	101
3.13.2	TLS 1.3 Server Function	103
3.13.2.1	Full Handshake	104
3.13.2.2	Resumption	110
3.13.2.3	0-RTT	113
3.14	Firmware Update	115
3.14.1	Secure Boot	116
3.14.2	Firmware Update	116
3.14.3	Encrypting the User Program	116
4.	API Functions	117
4.2.1	Common Function APIs	125
4.2.1.1	R_TSIP_Open	125
4.2.1.2	R_TSIP_Close	126
4.2.1.3	R_TSIP_SoftwareReset	127
4.2.1.4	R_TSIP_GetVersion	128
4.2.1.5	R_TSIP_GenerateUpdateKeyRingKeyIndex	129
4.2.2	Random Number Generation	130
4.2.2.1	R_TSIP_GenerateRandomNumber	130
4.2.3	AES	131
4.2.3.1	R_TSIP_GenerateAesXXXKeyIndex	131
4.2.3.2	R_TSIP_UpdateAesXXXKeyIndex	132
4.2.3.3	R_TSIP_GenerateAesXXXRandomKeyIndex	133
4.2.3.4	R_TSIP_AesXXXEcbEncryptInit	134
4.2.3.5	R_TSIP_AesXXXEcbEncryptUpdate	135
4.2.3.6	R_TSIP_AesXXXEcbEncryptFinal	136
4.2.3.7	R_TSIP_AesXXXEcbDecryptInit	137
4.2.3.8	R_TSIP_AesXXXEcbDecryptUpdate	138
4.2.3.9	R_TSIP_AesXXXEcbDecryptFinal	139
4.2.3.10	R_TSIP_AesXXXCbcEncryptInit	140
4.2.3.11	R_TSIP_AesXXXCbcEncryptUpdate	141
4.2.3.12	R_TSIP_AesXXXCbcEncryptFinal	142
4.2.3.13	R_TSIP_AesXXXCbcDecryptInit	143

4.2.3.14	R_TSIP_AesXXXCbcDecryptUpdate	144
4.2.3.15	R_TSIP_AesXXXCbcDecryptFinal	145
4.2.3.16	R_TSIP_AesXXXCtrInit	146
4.2.3.17	R_TSIP_AesXXXCtrUpdate	147
4.2.3.18	R_TSIP_AesXXXCtrFinal	148
4.2.3.19	R_TSIP_AesXXXGcmEncryptInit	149
4.2.3.20	R_TSIP_AesXXXGcmEncryptUpdate	150
4.2.3.21	R_TSIP_AesXXXGcmEncryptFinal	152
4.2.3.22	R_TSIP_AesXXXGcmDecryptInit	153
4.2.3.23	R_TSIP_AesXXXGcmDecryptUpdate	154
4.2.3.24	R_TSIP_AesXXXGcmDecryptFinal	156
4.2.3.25	R_TSIP_AesXXXCcmEncryptInit	157
4.2.3.26	R_TSIP_AesXXXCcmEncryptUpdate	159
4.2.3.27	R_TSIP_AesXXXCcmEncryptFinal	160
4.2.3.28	R_TSIP_AesXXXCcmDecryptInit	161
4.2.3.29	R_TSIP_AesXXXCcmDecryptUpdate	163
4.2.3.30	R_TSIP_AesXXXCcmDecryptFinal	164
4.2.3.31	R_TSIP_AesXXXCmacGenerateInit	165
4.2.3.32	R_TSIP_AesXXXCmacGenerateUpdate	166
4.2.3.33	R_TSIP_AesXXXCmacGenerateFinal	167
4.2.3.34	R_TSIP_AesXXXCmacVerifyInit	168
4.2.3.35	R_TSIP_AesXXXCmacVerifyUpdate	169
4.2.3.36	R_TSIP_AesXXXCmacVerifyFinal	170
4.2.4	DES	171
4.2.4.1	R_TSIP_GenerateTdesKeyIndex	171
4.2.4.2	R_TSIP_UpdateTdesKeyIndex	172
4.2.4.3	R_TSIP_GenerateTdesRandomKeyIndex	173
4.2.4.4	R_TSIP_TdesEcbEncryptInit	174
4.2.4.5	R_TSIP_TdesEcbEncryptUpdate	175
4.2.4.6	R_TSIP_TdesEcbEncryptFinal	176
4.2.4.7	R_TSIP_TdesEcbDecryptInit	177
4.2.4.8	R_TSIP_TdesEcbDecryptUpdate	178
4.2.4.9	R_TSIP_TdesEcbDecryptFinal	179
4.2.4.10	R_TSIP_TdesCbcEncryptInit	180
4.2.4.11	R_TSIP_TdesCbcEncryptUpdate	181
4.2.4.12	R_TSIP_TdesCbcEncryptFinal	182
4.2.4.13	R_TSIP_TdesCbcDecryptInit	183
4.2.4.14	R_TSIP_TdesCbcDecryptUpdate	184
4.2.4.15	R_TSIP_TdesCbcDecryptFinal	185
4.2.5	ARC4	186
4.2.5.1	R_TSIP_GenerateArc4KeyIndex	186

4.2.5.2	R_TSIP_UpdateArc4KeyIndex.....	187
4.2.5.3	R_TSIP_GenerateArc4RandomKeyIndex.....	188
4.2.5.4	R_TSIP_Arc4EncryptInit	189
4.2.5.5	R_TSIP_Arc4EncryptUpdate	190
4.2.5.6	R_TSIP_Arc4EncryptFinal	191
4.2.5.7	R_TSIP_Arc4DecryptInit	192
4.2.5.8	R_TSIP_Arc4DecryptUpdate	193
4.2.5.9	R_TSIP_Arc4DecryptFinal	194
4.2.6	RSA	195
4.2.6.1	R_TSIP_GenerateRsaXXXPublicKeyIndex	195
4.2.6.2	R_TSIP_GenerateRsaXXXPrivateKeyIndex.....	197
4.2.6.3	R_TSIP_UpdateRsaXXXPublicKeyIndex.....	198
4.2.6.4	R_TSIP_UpdateRsaXXXPrivateKeyIndex	200
4.2.6.5	R_TSIP_GenerateRsaXXXRandomKeyIndex	201
4.2.6.6	R_TSIP_RsaesPkcsXXXEncrypt	203
4.2.6.7	R_TSIP_RsaesPkcsXXXDecrypt	205
4.2.6.8	R_TSIP_RsassaPkcsXXXSignatureGenerate	206
4.2.6.9	R_TSIP_RsassaPkcsXXXSignatureVerification	208
4.2.6.10	R_TSIP_RsassaPssXXXSignatureGenerate	210
4.2.6.11	R_TSIP_RsassaPssXXXSignatureVerification	212
4.2.7	ECC	214
4.2.7.1	R_TSIP_GenerateEccPXXXPublicKeyIndex	214
4.2.7.2	R_TSIP_GenerateEccPXXXPrivateKeyIndex.....	216
4.2.7.3	R_TSIP_UpdateEccPXXXPublicKeyIndex.....	218
4.2.7.4	R_TSIP_UpdateEccPXXXPrivateKeyIndex	220
4.2.7.5	R_TSIP_GenerateEccPXXXRandomKeyIndex	222
4.2.7.6	R_TSIP_EcdsaPXXXSignatureGenerate.....	224
4.2.7.7	R_TSIP_EcdsaPXXXSignatureVerification.....	226
4.2.8	HASH.....	228
4.2.8.1	R_TSIP_ShaXXXInit	228
4.2.8.2	R_TSIP_ShaXXXUpdate	229
4.2.8.3	R_TSIP_ShaXXXFinal	230
4.2.8.4	R_TSIP_Md5Init	231
4.2.8.5	R_TSIP_Md5Update	232
4.2.8.6	R_TSIP_Md5Final	233
4.2.8.7	R_TSIP_GetCurrentHashDigestValue	234
4.2.9	HMAC	235
4.2.9.1	R_TSIP_GenerateShaXXXHmacKeyIndex.....	235
4.2.9.2	R_TSIP_UpdateShaXXXHmacKeyIndex	236
4.2.9.3	R_TSIP_ShaXXXHmacGenerateInit.....	237
4.2.9.4	R_TSIP_ShaXXXHmacGenerateUpdate	238

4.2.9.5	R_TSIP_ShaXXXHmacGenerateFinal.....	239
4.2.9.6	R_TSIP_ShaXXXHmacVerifyInit.....	240
4.2.9.7	R_TSIP_ShaXXXHmacVerifyUpdate.....	241
4.2.9.8	R_TSIP_ShaXXXHmacVerifyFinal.....	242
4.2.10	DH.....	243
4.2.10.1	R_TSIP_Rsa2048DhKeyAgreement.....	243
4.2.11	ECDH.....	244
4.2.11.1	R_TSIP_EcdhP256Init.....	244
4.2.11.2	R_TSIP_EcdhP256ReadPublicKey.....	245
4.2.11.3	R_TSIP_EcdhP256MakePublicKey.....	246
4.2.11.4	R_TSIP_EcdhP256CalculateSharedSecretIndex.....	248
4.2.11.5	R_TSIP_EcdhP256KeyDerivation.....	249
4.2.11.6	R_TSIP_EcdheP512KeyAgreement.....	251
4.2.12	Key Wrap.....	252
4.2.12.1	R_TSIP_AesXXXKeyWrap.....	252
4.2.12.2	R_TSIP_AesXXXKeyUnwrap.....	254
4.2.13	TLS (Common to TLS 1.2 and TLS 1.3).....	256
4.2.13.1	R_TSIP_GenerateTlsXXRsaPublicKeyIndex.....	256
4.2.13.2	R_TSIP_UpdateTlsXXRsaPublicKeyIndex.....	258
4.2.13.3	R_TSIP_TlsRegisterCaCertificationPublicKeyIndex.....	259
4.2.13.4	R_TSIP_TlsXXRootCertificateVerification.....	260
4.2.13.5	R_TSIP_TlsXXCertificateVerification.....	262
4.2.13.6	R_TSIP_TlsXXCertificateVerificationExtension.....	265
4.2.14	TLS (TLS 1.2).....	268
4.2.14.1	R_TSIP_TlsGeneratePreMasterSecret.....	268
4.2.14.2	R_TSIP_TlsEncryptPreMasterSecretWithRsa2048PublicKey.....	269
4.2.14.3	R_TSIP_TlsSVGenerateServerRandom.....	270
4.2.14.4	R_TSIP_TlsSVDDecryptPreMasterSecretWithRsa2048PrivateKey.....	271
4.2.14.5	R_TSIP_TlsXXGenerateMasterSecret.....	272
4.2.14.6	R_TSIP_TlsXXGenerateSessionKey.....	274
4.2.14.7	R_TSIP_TlsXXGenerateVerifyData.....	276
4.2.14.8	R_TSIP_TlsServersEphemeralEcdhPublicKeyRetrieves.....	277
4.2.14.9	R_TSIP_GenerateTlsXXP256EccKeyIndex.....	279
4.2.14.10	R_TSIP_TlsXXGeneratePreMasterSecretWithEccP256Key.....	280
4.2.14.11	R_TSIP_TlsXXGenerateExtendedMasterSecret.....	281
4.2.14.12	R_TSIP_TlsSVCertificateVerifyVerification.....	283
4.2.15	TLS (TLS 1.3).....	285
4.2.15.1	R_TSIP_GenerateTls13P256EccKeyIndex.....	285
4.2.15.2	R_TSIP_Tls13GenerateEcdheSharedSecret.....	286
4.2.15.3	R_TSIP_Tls13GenerateHandshakeSecret.....	287
4.2.15.4	R_TSIP_Tls13GenerateServerHandshakeTrafficKey.....	288

4.2.15.5	R_TSIP_Tls13GenerateClientHandshakeTrafficKey	290
4.2.15.6	R_TSIP_Tls13ServerHandshakeVerification	292
4.2.15.7	R_TSIP_Tls13GenerateMasterSecret	294
4.2.15.8	R_TSIP_Tls13GenerateApplicationTrafficKey	295
4.2.15.9	R_TSIP_Tls13UpdateApplicationTrafficKey	297
4.2.15.10	R_TSIP_Tls13GenerateResumptionMasterSecret.....	299
4.2.15.11	R_TSIP_Tls13GeneratePreSharedKey	301
4.2.15.12	R_TSIP_Tls13GeneratePskBinderKey	303
4.2.15.13	R_TSIP_Tls13GenerateResumptionHandshakeSecret	304
4.2.15.14	R_TSIP_Tls13Generate0RttApplicationWriteKey	306
4.2.15.15	R_TSIP_Tls13CertificateVerifyGenerate	307
4.2.15.16	R_TSIP_Tls13CertificateVerifyVerification	309
4.2.15.17	R_TSIP_GenerateTls13SVP256EccKeyIndex	311
4.2.15.18	R_TSIP_Tls13SVGenerateEcdheSharedSecret	312
4.2.15.19	R_TSIP_Tls13SVGenerateHandshakeSecret.....	313
4.2.15.20	R_TSIP_Tls13SVGenerateServerHandshakeTrafficKey	314
4.2.15.21	R_TSIP_Tls13SVGenerateClientHandshakeTrafficKey	316
4.2.15.22	R_TSIP_Tls13SVClientHandshakeVerification	318
4.2.15.23	R_TSIP_Tls13SVGenerateMasterSecret	320
4.2.15.24	R_TSIP_Tls13SVGenerateApplicationTrafficKey.....	321
4.2.15.25	R_TSIP_Tls13SVUpdateApplicationTrafficKey	323
4.2.15.26	R_TSIP_Tls13SVGenerateResumptionMasterSecret.....	325
4.2.15.27	R_TSIP_Tls13SVGeneratePreSharedKey	327
4.2.15.28	R_TSIP_Tls13SVGeneratePskBinderKey	329
4.2.15.29	R_TSIP_Tls13SVGenerateResumptionHandshakeSecret.....	330
4.2.15.30	R_TSIP_Tls13SVGenerate0RttApplicationWriteKey.....	331
4.2.15.31	R_TSIP_Tls13SVCertificateVerifyGenerate	332
4.2.15.32	R_TSIP_Tls13SVCertificateVerifyVerification	334
4.2.15.33	R_TSIP_Tls13EncryptInit.....	336
4.2.15.34	R_TSIP_Tls13EncryptUpdate.....	338
4.2.15.35	R_TSIP_Tls13EncryptFinal.....	339
4.2.15.36	R_TSIP_Tls13DecryptInit	340
4.2.15.37	R_TSIP_Tls13DecryptUpdate	342
4.2.15.38	R_TSIP_Tls13DecryptFinal	343
4.2.16	Firmware Update	344
4.2.16.1	R_TSIP_StartUpdateFirmware	344
4.2.16.2	R_TSIP_GenerateFirmwareMAC.....	345
4.2.16.3	R_TSIP_VerifyFirmwareMAC	349
4.2.16.4	TSIP_GEN_MAC_CB_FUNC_T Type	350
4.3	User-Defined Functions.....	353

5.	Key Injection and Updating.....	356
5.1	Key Injection.....	356
5.2	Key Updating.....	357
5.3	Generating an Encrypted Key Using Security Key Management Tool	358
5.3.1	Key Injection Procedure	358
5.3.1.1	Procedure for CLI Version.....	358
5.3.1.2	Procedure for GUI Version.....	359
5.3.2	Key Update Operation Procedure	362
5.3.2.1	Procedure for CLI Version.....	362
5.3.2.2	Procedure for GUI Version.....	364
6.	Sample Programs.....	369
6.1	Confirming Operation of the Demo Projects	369
6.1.1	Setting Up the Demo Project.....	370
6.1.2	Overview of Demo Project.....	373
6.1.3	Demo Project Execution Example.....	380
6.2	Confirming Operation of the Secure Bootloader and Firmware Update Demo Projects.....	383
6.2.1	Demo Project Setup	384
6.2.2	Overview of Secure Bootloader and Firmware Update Projects.....	393
6.2.2.1	Secure Bootloader project.....	396
6.2.2.2	Firmware Update Project.....	407
6.2.3	Secure Bootloader and Firmware Update Execution Example.....	408
6.2.4	Execution Example of Factory Programming Using Motorola S Format File Containing Secure Bootloader and Encrypted User Program	419
6.2.5	Debugging User Program Projects	427
6.2.6	Notes on Transition from Secure Bootloader to User Program	428
6.2.7	Performance Information for Demo Project.....	429
7.	Appendix.....	430
7.1	Confirmed Operation Environment.....	430
7.2	Troubleshooting.....	431
7.3	User Key Encryption Formats	432
7.3.1	AES	432
7.3.1.1	AES 128-Bit Key.....	432
7.3.1.2	AES 256-Bit Key.....	432
7.3.2	DES	432
7.3.3	ARC4	433
7.3.4	RSA	433
7.3.4.1	RSA 1024-Bit Key	433
7.3.4.2	RSA 2048-Bit Key	434
7.3.4.3	RSA 3072-Bit Key	434
7.3.4.4	RSA 4096-Bit Key	435

RX Family	TSIP (Trusted Secure IP) Module Firmware Integration Technology
7.3.5	ECC..... 435
7.3.5.1	ECC P192 435
7.3.5.2	ECC P224 436
7.3.5.3	ECC P256 436
7.3.5.4	ECC P384 437
7.3.6	HMAC..... 437
7.3.6.1	SHA1-HMAC Key..... 437
7.3.6.2	SHA256-HMAC Key..... 437
7.3.7	KUK 437
7.4	Wrapped Key of Public Key Formats for Asymmetric Cryptography 438
7.4.1	RSA..... 438
7.4.2	ECC..... 438
7.4.2.1	ECC P 192-Bit Key..... 438
7.4.2.2	ECC P 224-Bit Key..... 438
7.4.2.3	ECC P 256-Bit Key..... 438
7.4.2.4	ECC P 384-Bit Key..... 438
7.5	Using Renesas Secure Flash Programmer..... 439
7.5.1	Usage Notes..... 439
7.5.2	provisioning key Tab 439
7.5.3	Key Wrap Tab 439
7.5.3.1	Key Data Input Formats 442
7.5.4	Firm Update Tab 445
7.5.4.1	Select MCU 446
7.5.4.2	Select Output Format 447
7.5.4.3	User Program Key..... 448
7.5.4.4	User Program File Path..... 448
7.5.4.5	Secure Boot Program File Path 448
7.5.5	Key Injection Procedure 449
7.5.6	Key Update Operation Procedure 451
7.5.7	Usage Notes..... 455
8.	Reference Documents..... 456
	Revision History..... 457

1. Overview

1.1 Terminology

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

Table 1-1 Descriptions of Terms

Term	Description	Correspondence with Diagram of Key Installation Process
Key injection	Injecting a wrapped key into the device at the factory.	—
Key updating	Injecting a wrapped key into the device in the field.	—
User key	An encryption key in plaintext used by the user. Not used on the device. For AES, DES, ARC4, and HMAC, user keys are used as shared keys. For RSA and ECC, user keys are used as public keys and secret keys.	Key-1
Encrypted key	Key information generated by adding a MAC value and encrypting a user key using a UFPK or update key. An encrypted key corresponding to the same user key is a shared value on each device.	eKey-1
Wrapped key	Data consisting of an encrypted key that has been converted into a form that is usable by the TSIP driver by key injection or key updating. The wrapped key has been wrapped using an HUK, so the wrapped key of the same encrypted key will be a unique value on each device.	Index-1 or Index-2
UFPK	User Factory Programming Key A keyring set by the user and used to generate an encrypted key from a user key during key injection. Not used on the device.	Key-2
W-UFPK	Wrapped UFPK Key information generated by wrapping a UFPK using an HRK on the DLM server. The UFPK is decrypted using the HRK internally by the TSIP.	Index-2
Key update key (KUK)	A key set by the user and used to generate an encrypted key from a user key during key updating. The wrapped KUK for the update key must be generated beforehand by key injection in order to perform key updating on the device.	—
Hardware root key (HRK)	A shared encryption key that exists only inside the TSIP and in secure rooms within Renesas.	—
Hardware unique key (HUK)	A device-specific encryption key that is derived internally by the TSIP and used to protect key data.	—
Device Lifecycle Management (DLM) server	A key management server operated by Renesas. It is used for wrapping UFPK.	—

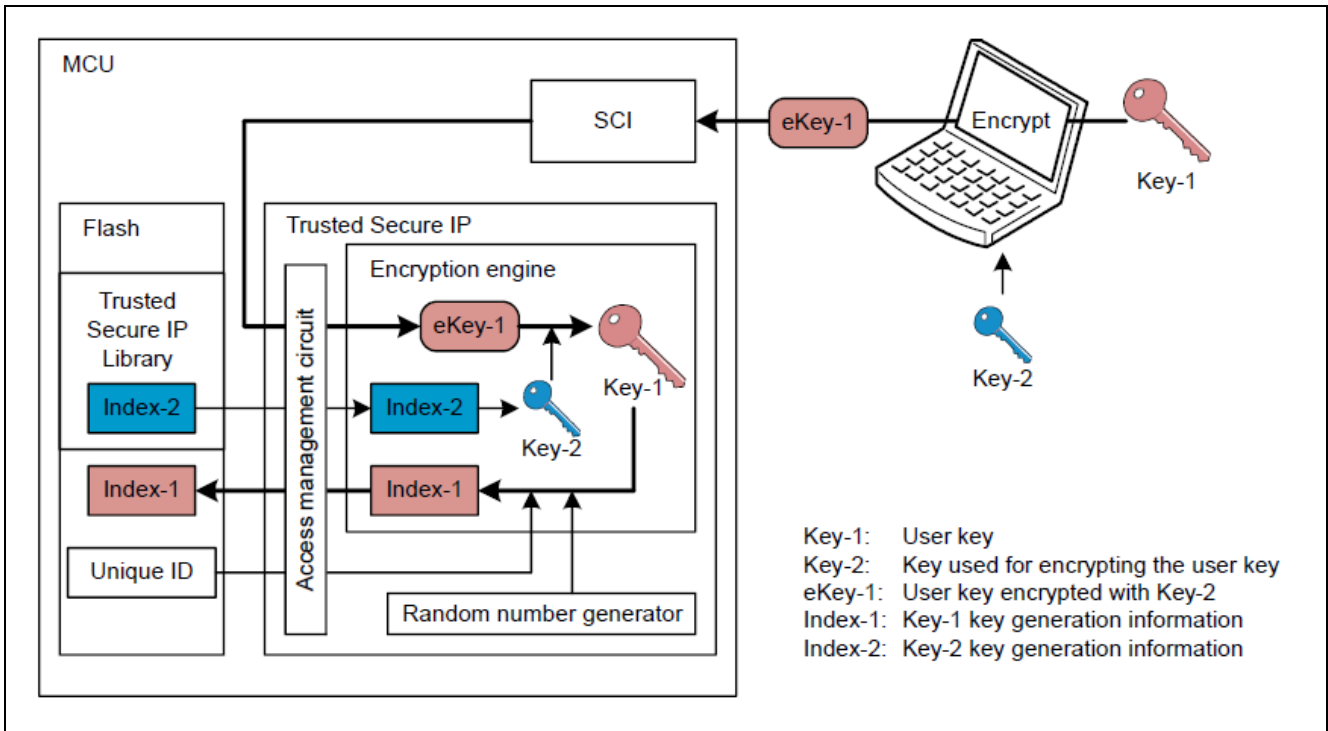


Figure 1.1 Diagram of Key Installation Process (Reproduced from Figure 52.4 in section 52, Trusted Secure IP (TSIP), in RX65N Group, RX651 Group, User’s Manual: Hardware)

1.2 TSIP Overview

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

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

The TSIP restores from the wrapped key the encryption key used for cryptographic operations. This operation is performed internally by the TSIP. The wrapped key is tied to an HUK derived from a unique ID, making it device-specific. This means that even if a wrapped key is copied from one device to a different device it cannot be used on the second device. To access the TSIP hardware an application must use the TSIP driver.

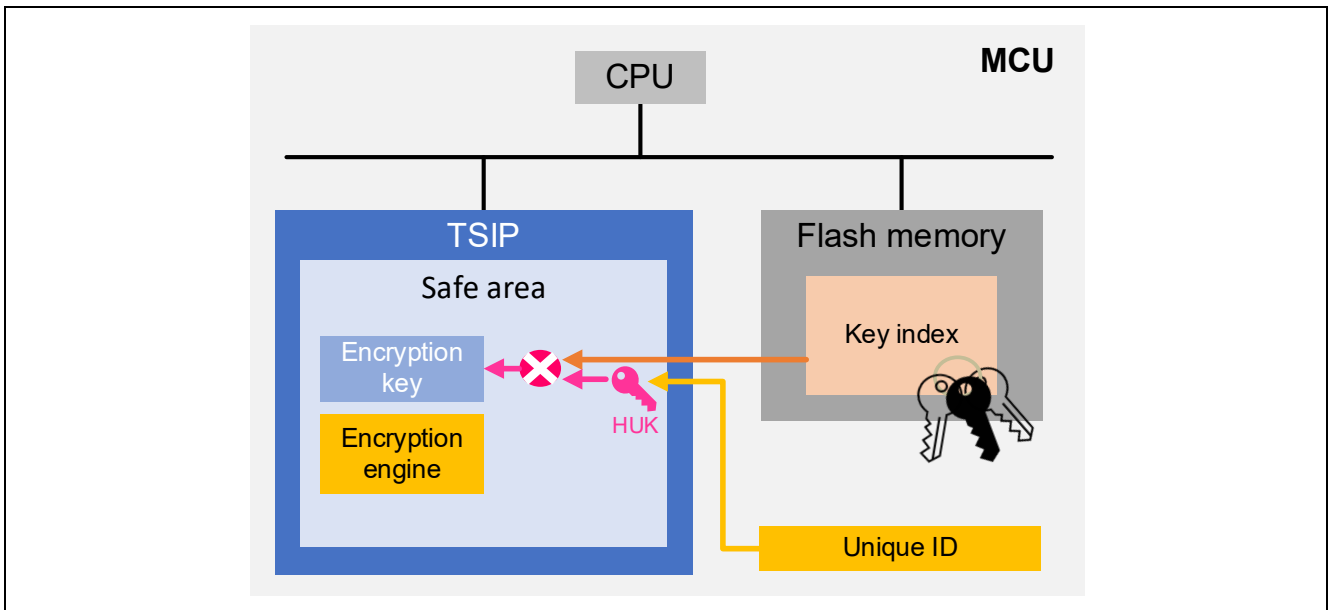


Figure 1.2 MCU Incorporating TSIP

1.3 Structure of Product Files

Table 1.2 below lists the files included in the product.

Table 1-2 Structure of Product Files

File/Directory (Bold) Name	Description
Readme.txt	Readme
RX_TSIP_SoftwareLicenseAgreement_JPN.pdf	Software License Agreement (Japanese)
RX_TSIP_SoftwareLicenseAgreement_ENG.pdf	Software License Agreement (English)
r20an0371jj0121-rx-tsip-security.pdf	TSIP driver application note (Japanese)
r20an0371ej0121-rx-tsip-security.pdf	TSIP driver application note (English)
reference_documents	Folder containing documentation of topics such as how to use the FIT module with various integrated development environments
ja	Folder containing documentation of topics such as how to use the FIT module with various integrated development environments (Japanese)
r01an1826jj0110-rx.pdf	How to add FIT modules to CS+ projects (Japanese)
r01an1723ju0121-rx.pdf	How to add FIT modules to e ² studio projects (Japanese)
r20an0451js0140-e2studio-sc.pdf	Smart Configurator user's guide (Japanese)
r01an5792jj0102-rx-tsip.pdf	AES cryptography project application note (Japanese)
r01an5880jj0103-rx-tsip.pdf	TLS cooperation function project application note (Japanese)
en	Folder containing documentation of topics such as how to use the FIT module with various integrated development environments (English)
r01an1826ej0110-rx.pdf	How to add FIT modules to CS+ projects (English)
r01an1723eu0121-rx.pdf	How to add FIT modules to e ² studio projects (English)
r20an0451es0140-e2studio-sc.pdf	Smart Configurator user's guide (English)
r01an5792ej0102-rx-tsip.pdf	AES cryptography project application note (English)
r01an5880ej0103-rx-tsip.pdf	TLS cooperation function project application note (English)
FITModules	FIT module folder
r_tsip_rx_v1.21.zip	TSIP driver FIT module
r_tsip_rx_v1.21.xml	TSIP driver FIT module e ² studio FIT plug-in XML file
r_tsip_rx_v1.21_extend.mdf	TSIP driver FIT module Smart Configurator configuration settings file
FITDemos	Demo project folder
rxXXX_bbb_tsip_sample *1*2	Project showing how to write keys and use cryptographic APIs
rxXXX_bbb_tsip_secure_boot *1*2	Secure boot/secure update implementation example
rxXXX_bbb_tsip_secure_boot *1*2	Secure boot firmware
rxXXX_bbb_tsip_user_program *1*2	User program following secure update
rx65n_2mb_rsk_tsip_aes_sample	AES cryptography project for RX65N
rx72n_ek_tsip_aes_sample	AES cryptography project for RX72N

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

Note: 1. "rxXXX" represents the RX group names.

2. "bbb" is the name of the supported board. Usually, USB and UART are available as IF for Update, but if the MCU does not support USB, only UART is supported. The project name which supports only UART includes _sci before _secure_boot and _user_program.

RSK : rsk (rsk(_tsip)_sci : only support UART)

MCB : mcb (mcb(_tsip)_sci : only support UART)

Envision Kit : ek

Table 1.3 lists the files and folders contained in the folder produced by unzipping r_tsip_rx_v.1.21.zip.

Table 1-3 File Structure

File/Directory (Bold) Names	Description
r_config	TSIP driver config file folder
r_tsip_rx_config.h	TSIP driver config file (default settings)
r_tsip_rx	TSIP driver FIT module folder
src	TSIP driver source code folder
targets	Folder containing program code dependent on specific MCU models
Group name*1	TSIP/TSIP-Lite driver folder There is a separate folder for each product group.
r_tsip_rx.c	TSIP system driver source code (required)
r_tsip_{algorithm}_rx.c	API source code for each cryptographic algorithm {algorithm} represents the name of the algorithm, such as aes or rsa.
r_tsip_rx_private.c	TSIP driver source code (Can be turned on/off with an option.)
r_tsip_rx_private.h	TSIP driver header file (Can be turned on/off with an option.)
iodefine	TSIP access header file storage folder
r_tsip_{Group name}_iodefine.h	TSIP access header file The folder name matches the name of the product group.
ip	TSIP access source code storage folder
r_tsip_rx_pxx.c	TSIP access source code "xx" in the file name represents a numeric value.
r_tsip_rx_subprcxx.c	TSIP access source code "xx" in the file name represents a numeric value.
r_tsip_rx_functionxxx.c	TSIP access source code (sub API) "xx" in the file name represents a numeric value.
s_flash.c	Key information file
doc	
ja	TSIP system driver source code (required)
r20an0371jj0121-rx-tsip-security.pdf	TSIP driver application note (Japanese)
en	TSIP driver source code (Can be turned on/off with an option.)

File/Directory (Bold) Names		Description
	r20an0371ej0121-rx-tsip-security.pdf	TSIP driver application note (English)
	r_tsip_rx_if.h	TSIP driver header file
	readme.txt	Readme

1.4 Development Environment

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

1. Integrated development environment
Refer to the “Integrated development environment” item under 7.1, Confirmed Operation Environment.
2. C compiler
Refer to the “C compiler” item under 7.1, Confirmed Operation Environment.
3. Emulator/debugger
E1, E20, or E2 Lite
4. Evaluation boards
Refer to the “Board used” item under 7.1, Confirmed Operation Environment. All of the boards listed are special product versions with cryptographic functionality. Make sure to confirm the product model name before ordering. e² studio and CC-RX were used in combination for evaluation and to create the demo project.
The project conversion function can be used to convert projects from e² studio to CS+. If you encounter errors such as compiler errors, please contact your Renesas representative.

1.5 Code Size

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

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

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

Module revision: r_tsip_rx rev1.21

Compiler version: Renesas Electronics C/C++ Compiler Package for RX Family V3.06.00
(integrated development environment default settings with “-lang = c99” option added)

GCC for Renesas RX 8.3.0.202311
(integrated development environment default settings with “-std = gnu99” option added)

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

Configuration options: default settings

ROM, RAM, and Stack Sizes				
Device	Category	Memory Used		
		Renesas Compiler	GCC*	IAR Compiler
TSIP-Lite	ROM	56,559 bytes	55,548 bytes	58,546 bytes
	RAM	1,364 bytes	1,364 bytes	1,364 bytes
	Stack	184 bytes	—	164 bytes
TSIP	ROM	453,129 bytes	435,069 bytes	451,349 bytes
	RAM	7,988 bytes	7,988 bytes	7,988 bytes
	Stack	1,648 bytes	—	1,376 bytes

Note: The items which stack analysis does not work are filled with “—”.

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

ROM, RAM, and Stack Sizes						
Device	Function	Category	Memory Used			
			Renesas Compiler	GCC*	IAR Compiler*	
RX231	Common Function APIs	ROM	5,201 bytes	5,617 bytes	5,382 bytes	
		RAM	1,356 bytes	1,356 bytes	1,356 bytes	
		Stack	80 bytes	—	—	
	Random Number Generation	ROM	902 bytes	427 bytes	419 bytes	
		RAM	0 bytes	0 bytes	0 bytes	
		Stack	28 bytes	—	20 bytes	
	AES (128bit)	ROM	23,700 bytes	23,402 bytes	25,034 bytes	
		RAM	112 bytes	112 bytes	112 bytes	
		Stack	184 bytes	—	164 bytes	
	AES (256bit)	ROM	24,733 bytes	24,433 bytes	26,125 bytes	
		RAM	112 bytes	192 bytes	112 bytes	
		Stack	184 bytes	—	164 bytes	
	Key Wrap	ROM	6,886 bytes	5,947 bytes	6,239 bytes	
		RAM	0 bytes	0 bytes	0 bytes	
		Stack	116 bytes	—	68 bytes	
	Firmware Update	ROM	3,267 bytes	2,427 bytes	2,560 bytes	
		RAM	8 bytes	8 bytes	8 bytes	
		Stack	92 bytes	—	—	
	RX65N	Common Function APIs	ROM	8,373 bytes	8,464 bytes	8,526 bytes
			RAM	1,452 bytes	1,400 bytes	1,400 bytes
			Stack	84 bytes	—	—
Random Number Generation		ROM	4,310 bytes	437 bytes	433 bytes	
		RAM	52 bytes	0 bytes	0 bytes	
		Stack	28 bytes	—	20 bytes	
AES (128bit)		ROM	37,456 bytes	36,634 bytes	39,346 bytes	
		RAM	244 bytes	192 bytes	192 bytes	
		Stack	192 bytes	—	160 bytes	
AES (256bit)		ROM	30,455 bytes	24,433 bytes	31,653 bytes	
		RAM	244 bytes	112 bytes	192 bytes	
		Stack	192 bytes	—	160 bytes	
DES		ROM	10,261 bytes	10,030 bytes	10,383 bytes	
		RAM	136 bytes	84 bytes	84 bytes	
		Stack	68 bytes	—	48 bytes	
ARC4		ROM	8,932 bytes	8,829 bytes	9,088 bytes	
		RAM	128 bytes	76 bytes	76 bytes	
		Stack	68 bytes	—	40 bytes	
RSA		ROM	136,236 bytes	122,378 bytes	133,269 bytes	
		RAM	6,724 bytes	6,724 bytes	6,724 bytes	
		Stack	1,632 bytes	—	1,376 bytes	
ECC		ROM	41,867 bytes	45,974 bytes	43,496 bytes	
		RAM	6,724 bytes	6,672 bytes	6,672 bytes	
		Stack	360 bytes	—	340 bytes	
HASH		ROM	6,818 bytes	2,922 bytes	3,152 bytes	

		RAM	52 bytes	0 bytes	0 bytes
		Stack	244 bytes	—	236 bytes
	HMAC	ROM	16,111 bytes	15,940 bytes	16,470 bytes
		RAM	216 bytes	164 bytes	164 bytes
		Stack	124 bytes	—	84 bytes
	DH	ROM	6,701 bytes	2,533 bytes	2,709 bytes
		RAM	52 bytes	0 bytes	0 bytes
		Stack	84 bytes	—	40 bytes
	ECDH	ROM	30,027 bytes	33,149 bytes	30,876 bytes
		RAM	6,668 bytes	6,616 bytes	6,616 bytes
		Stack	308 bytes	—	232 bytes
	Key Wrap	ROM	9,917 bytes	5,441 bytes	5,857 bytes
		RAM	52 bytes	0 bytes	0 bytes
		Stack	124 bytes	—	76 bytes
	TLS (Common to TLS 1.2 and TLS 1.3)	ROM	76,494 bytes	71,122 bytes	74,370 bytes
		RAM	7,844 bytes	7,792 bytes	7,792 bytes
		Stack	740 bytes	—	124 bytes
	TLS (TLS 1.2)	ROM	136,486 bytes	131,468 bytes	135,669 bytes
		RAM	7,844 bytes	7,792 bytes	7,792 bytes
		Stack	812 bytes	—	184 bytes
	TLS (TLS 1.3)	ROM	156,556 bytes	151,236 bytes	154,922 bytes
		RAM	6,680 bytes	6,628 bytes	6,628 bytes
		Stack	1,248 bytes	—	1,208 bytes
	Firmware Update	ROM	6,407 bytes	2,383 bytes	2,565 bytes
RAM		60 bytes	8 bytes	8 bytes	
Stack		84 bytes	—	—	

Note: The items which stack analysis does not work are filled with “—”.

1.6 Sections

The TSIP driver uses the default sections.

In sample programs, the sections `C_FIRMWARE_UPDATE_CONTROL_BLOCK` and `C_FIRMWARE_UPDATE_CONTROL_BLOCK_MIRROR` are used. Settings for these sections are configured automatically if CC-RX is set as the compiler and the TSIP driver is added to the project using Smart Configurator. Moreover, sample programs uses `C_ENCRYPTED_KEY_BLOCK`. If changes are required, edit the section setting.

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

1.7 Performance

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

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

1.7.1 RX231

Table 1-4 Performance of Common APIs

API	Performance (Unit: Cycle)
R_TSIP_Open	7,400,000
R_TSIP_Close	460
R_TSIP_GetVersion	30
R_TSIP_GenerateAes128KeyIndex	4,000
R_TSIP_GenerateAes256KeyIndex	4,600
R_TSIP_GenerateAes128RandomKeyIndex	2,300
R_TSIP_GenerateAes256RandomKeyIndex	3,100
R_TSIP_GenerateRandomNumber	950
R_TSIP_GenerateUpdateKeyRingKeyIndex	4,600
R_TSIP_UpdateAes128KeyIndex	3,600
R_TSIP_UpdateAes256KeyIndex	4,200

Table 1-5 Firmware Verification Performance

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

Table 1-6 Performance of AES

API	Performance (Unit: Cycle)		
	16-Byte Processing	48-Byte Processing	80-Byte Processing
R_TSIP_Aes128EcbEncryptInit	1,400	1,400	1,400
R_TSIP_Aes128EcbEncryptUpdate	620	800	970
R_TSIP_Aes128EcbEncryptFinal	570	570	570
R_TSIP_Aes128EcbDecryptInit	1,400	1,400	1,400
R_TSIP_Aes128EcbDecryptUpdate	740	920	1,100
R_TSIP_Aes128EcbDecryptFinal	580	580	580
R_TSIP_Aes256EcbEncryptInit	1,700	1,700	1,700
R_TSIP_Aes256EcbEncryptUpdate	660	910	1,200
R_TSIP_Aes256EcbEncryptFinal	580	580	580
R_TSIP_Aes256EcbDecryptInit	1,700	1,700	1,700
R_TSIP_Aes256EcbDecryptUpdate	810	1,100	1,300
R_TSIP_Aes256EcbDecryptFinal	580	580	580
R_TSIP_Aes128CbcEncryptInit	1,400	1,400	1,400
R_TSIP_Aes128CbcEncryptUpdate	680	860	1,100
R_TSIP_Aes128CbcEncryptFinal	590	590	590
R_TSIP_Aes128CbcDecryptInit	1,400	1,400	1,400
R_TSIP_Aes128CbcDecryptUpdate	800	970	1,200
R_TSIP_Aes128CbcDecryptFinal	600	600	600
R_TSIP_Aes256CbcEncryptInit	1,700	1,700	1,700
R_TSIP_Aes256CbcEncryptUpdate	720	960	1,300
R_TSIP_Aes256CbcEncryptFinal	600	600	600
R_TSIP_Aes256CbcDecryptInit	1,700	1,700	1,700
R_TSIP_Aes256CbcDecryptUpdate	860	1,100	1,400
R_TSIP_Aes256CbcDecryptFinal	610	610	610

Table 1-7 Performance of AES-GCM

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

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

Table 1-8 Performance of AES-CCM

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

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

Table 1-9 Performance of AES-CMAC

API	Performance (Unit: Cycle)		
	48-Byte Processing	64-Byte Processing	80-Byte Processing
R_TSIP_Aes128CmacGenerateInit	920	920	920
R_TSIP_Aes128CmacGenerateUpdate	820	900	990
R_TSIP_Aes128CmacGenerateFinal	1,100	1,100	1,100
R_TSIP_Aes128CmacVerifyInit	910	920	920
R_TSIP_Aes128CmacVerifyUpdate	820	910	1,000
R_TSIP_Aes128CmacVerifyFinal	1,800	1,800	1,800
R_TSIP_Aes256CmacGenerateInit	1,300	1,300	1,300
R_TSIP_Aes256CmacGenerateUpdate	880	1,100	1,200
R_TSIP_Aes256CmacGenerateFinal	1,200	1,200	1,200
R_TSIP_Aes256CmacVerifyInit	1,300	1,300	1,300
R_TSIP_Aes256CmacVerifyUpdate	890	1,100	1,200
R_TSIP_Aes256CmacVerifyFinal	1,900	1,900	1,900

Table 1-10 Performance of AES Key Wrap

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

1.7.2 RX23W

Table 1-11 Performance of Common APIs

API	Performance (Unit: Cycle)
R_TSIP_Open	7,400,000
R_TSIP_Close	690
R_TSIP_GetVersion	40
R_TSIP_GenerateAes128KeyIndex	4,400
R_TSIP_GenerateAes256KeyIndex	5,000
R_TSIP_GenerateAes128RandomKeyIndex	2,500
R_TSIP_GenerateAes256RandomKeyIndex	3,400
R_TSIP_GenerateRandomNumber	1,100
R_TSIP_GenerateUpdateKeyRingKeyIndex	5000
R_TSIP_UpdateAes128KeyIndex	3,900
R_TSIP_UpdateAes256KeyIndex	4,500

Table 1-12 Firmware Verification Performance

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

Table 1-13 Performance of AES

API	Performance (Unit: Cycle)		
	16-Byte Processing	48-Byte Processing	80-Byte Processing
R_TSIP_Aes128EcbEncryptInit	1,500	1,500	1,500
R_TSIP_Aes128EcbEncryptUpdate	750	930	1,200
R_TSIP_Aes128EcbEncryptFinal	660	660	660
R_TSIP_Aes128EcbDecryptInit	1,600	1,600	1,600
R_TSIP_Aes128EcbDecryptUpdate	860	1,100	1,300
R_TSIP_Aes128EcbDecryptFinal	670	670	670
R_TSIP_Aes256EcbEncryptInit	1,900	1,900	1,900
R_TSIP_Aes256EcbEncryptUpdate	780	1,100	1,300
R_TSIP_Aes256EcbEncryptFinal	670	670	670
R_TSIP_Aes256EcbDecryptInit	1,900	1,900	1,900
R_TSIP_Aes256EcbDecryptUpdate	930	1,200	1,500
R_TSIP_Aes256EcbDecryptFinal	690	690	690
R_TSIP_Aes128CbcEncryptInit	1,600	1,600	1,600
R_TSIP_Aes128CbcEncryptUpdate	820	1,100	1,200
R_TSIP_Aes128CbcEncryptFinal	690	690	690
R_TSIP_Aes128CbcDecryptInit	1,600	1,600	1,600
R_TSIP_Aes128CbcDecryptUpdate	930	1,200	1,300
R_TSIP_Aes128CbcDecryptFinal	700	700	700
R_TSIP_Aes256CbcEncryptInit	1,900	1,900	1,900
R_TSIP_Aes256CbcEncryptUpdate	860	1,100	1,400
R_TSIP_Aes256CbcEncryptFinal	700	700	700
R_TSIP_Aes256CbcDecryptInit	1,900	2,000	2,000
R_TSIP_Aes256CbcDecryptUpdate	1,000	1,300	1,500
R_TSIP_Aes256CbcDecryptFinal	720	720	720

Table 1-14 Performance of AES-GCM

API	Performance (Unit: Cycle)		
	48-Byte Processing	64-Byte Processing	80-Byte Processing
R_TSIP_Aes128GcmEncryptInit	6,300	6,300	6,300
R_TSIP_Aes128GcmEncryptUpdate	3,400	4,000	4,500
R_TSIP_Aes128GcmEncryptFinal	1,500	1,500	1,500
R_TSIP_Aes128GcmDecryptInit	6,300	6,300	6,300
R_TSIP_Aes128GcmDecryptUpdate	2,900	3,000	3,100
R_TSIP_Aes128GcmDecryptFinal	2,400	2,400	2,400
R_TSIP_Aes256GcmEncryptInit	7,000	7,000	7,000
R_TSIP_Aes256GcmEncryptUpdate	3,500	4,100	4,700
R_TSIP_Aes256GcmEncryptFinal	1,600	1,600	1,600
R_TSIP_Aes256GcmDecryptInit	7,000	7,000	7,000
R_TSIP_Aes256GcmDecryptUpdate	3,000	3,100	3,300
R_TSIP_Aes256GcmDecryptFinal	2,400	2,400	2,400

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

Table 1-15 Performance of AES-CCM

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

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

Table 1-16 Performance of AES-CMAC

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

Table 1-17 Performance of AES Key Wrap

API	Performance (Unit: Cycle)	
	Wrap Target Key AES-128	Wrap Target Key AES-256
R_TSIP_Aes128KeyWrap	11,000	17,000
R_TSIP_Aes256KeyWrap	12,000	18,000
R_TSIP_Aes128KeyUnwrap	14,000	20,000
R_TSIP_Aes256KeyUnwrap	15,000	21,000

1.7.3 RX26T

Table 1-18 Performance of Common APIs

API	Performance (Unit: Cycle)
R_TSIP_Open	7,400,000
R_TSIP_Close	280
R_TSIP_GetVersion	20
R_TSIP_GenerateAes128KeyIndex	4,000
R_TSIP_GenerateAes256KeyIndex	4,500
R_TSIP_GenerateAes128RandomKeyIndex	2,200
R_TSIP_GenerateAes256RandomKeyIndex	3,000
R_TSIP_GenerateRandomNumber	910
R_TSIP_GenerateUpdateKeyRingKeyIndex	4,500
R_TSIP_UpdateAes128KeyIndex	3,500
R_TSIP_UpdateAes256KeyIndex	4,100

Table 1-19 Firmware Verification Performance

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

Table 1-20 Performance of AES

API	Performance (Unit: Cycle)		
	16-Byte Processing	48-Byte Processing	80-Byte Processing
R_TSIP_Aes128EcbEncryptInit	1,300	1,300	1,300
R_TSIP_Aes128EcbEncryptUpdate	560	740	910
R_TSIP_Aes128EcbEncryptFinal	510	510	510
R_TSIP_Aes128EcbDecryptInit	1,300	1,300	1,300
R_TSIP_Aes128EcbDecryptUpdate	680	850	1,000
R_TSIP_Aes128EcbDecryptFinal	510	510	510
R_TSIP_Aes256EcbEncryptInit	1,600	1,600	1,600
R_TSIP_Aes256EcbEncryptUpdate	610	840	1,100
R_TSIP_Aes256EcbEncryptFinal	500	500	510
R_TSIP_Aes256EcbDecryptInit	1,600	1,600	1,600
R_TSIP_Aes256EcbDecryptUpdate	750	980	1,200
R_TSIP_Aes256EcbDecryptFinal	520	520	520
R_TSIP_Aes128CbcEncryptInit	1,300	1,300	1,300
R_TSIP_Aes128CbcEncryptUpdate	600	790	960
R_TSIP_Aes128CbcEncryptFinal	540	540	540
R_TSIP_Aes128CbcDecryptInit	1,300	1,300	1,300
R_TSIP_Aes128CbcDecryptUpdate	720	900	1,100
R_TSIP_Aes128CbcDecryptFinal	550	550	550
R_TSIP_Aes256CbcEncryptInit	1,600	1,600	1,600
R_TSIP_Aes256CbcEncryptUpdate	640	900	1,100
R_TSIP_Aes256CbcEncryptFinal	530	530	530
R_TSIP_Aes256CbcDecryptInit	1,600	1,600	1,600
R_TSIP_Aes256CbcDecryptUpdate	780	1,000	1,300
R_TSIP_Aes256CbcDecryptFinal	540	540	540

Table 1-21 Performance of AES-GCM

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

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

Table 1-22 Performance of AES-CCM

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

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

Table 1-23 Performance of AES-CMAC

API	Performance (Unit: Cycle)		
	48-Byte Processing	64-Byte Processing	80-Byte Processing
R_TSIP_Aes128CmacGenerateInit	870	870	870
R_TSIP_Aes128CmacGenerateUpdate	720	810	900
R_TSIP_Aes128CmacGenerateFinal	1,000	1,000	1,000
R_TSIP_Aes128CmacVerifyInit	870	880	880
R_TSIP_Aes128CmacVerifyUpdate	720	810	900
R_TSIP_Aes128CmacVerifyFinal	1,700	1,700	1,700
R_TSIP_Aes256CmacGenerateInit	1,200	1,200	1,200
R_TSIP_Aes256CmacGenerateUpdate	800	920	1,000
R_TSIP_Aes256CmacGenerateFinal	1,100	1,100	1,100
R_TSIP_Aes256CmacVerifyInit	1,200	1,200	1,200
R_TSIP_Aes256CmacVerifyUpdate	790	910	1,000
R_TSIP_Aes256CmacVerifyFinal	1,800	1,800	1,800

Table 1-24 Performance of AES Key Wrap

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

1.7.4 RX66T, RX72T

Table 1-25 Performance of Common APIs

API	Performance (Unit: Cycle)
R_TSIP_Open	7,400,000
R_TSIP_Close	290
R_TSIP_GetVersion	22
R_TSIP_GenerateAes128KeyIndex	4,000
R_TSIP_GenerateAes256KeyIndex	4,500
R_TSIP_GenerateAes128RandomKeyIndex	2,200
R_TSIP_GenerateAes256RandomKeyIndex	3,000
R_TSIP_GenerateRandomNumber	910
R_TSIP_GenerateUpdateKeyRingKeyIndex	4,500
R_TSIP_UpdateAes128KeyIndex	3,500
R_TSIP_UpdateAes256KeyIndex	4,100

Table 1-26 Firmware Verification Performance

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

Table 1-27 Performance of AES

API	Performance (Unit: Cycle)		
	16-Byte Processing	48-Byte Processing	80-Byte Processing
R_TSIP_Aes128EcbEncryptInit	1,300	1,300	1,300
R_TSIP_Aes128EcbEncryptUpdate	570	750	930
R_TSIP_Aes128EcbEncryptFinal	520	510	510
R_TSIP_Aes128EcbDecryptInit	1,300	1,300	1,300
R_TSIP_Aes128EcbDecryptUpdate	680	860	1,100
R_TSIP_Aes128EcbDecryptFinal	520	520	520
R_TSIP_Aes256EcbEncryptInit	1,600	1,600	1,600
R_TSIP_Aes256EcbEncryptUpdate	610	850	1,100
R_TSIP_Aes256EcbEncryptFinal	530	520	520
R_TSIP_Aes256EcbDecryptInit	1,600	1,600	1,600
R_TSIP_Aes256EcbDecryptUpdate	750	1,000	1,300
R_TSIP_Aes256EcbDecryptFinal	540	540	540
R_TSIP_Aes128CbcEncryptInit	1,400	1,400	1,400
R_TSIP_Aes128CbcEncryptUpdate	630	810	980
R_TSIP_Aes128CbcEncryptFinal	540	530	530
R_TSIP_Aes128CbcDecryptInit	1,400	1,400	1,400
R_TSIP_Aes128CbcDecryptUpdate	730	910	1,100
R_TSIP_Aes128CbcDecryptFinal	540	540	540
R_TSIP_Aes256CbcEncryptInit	1,700	1,700	1,700
R_TSIP_Aes256CbcEncryptUpdate	660	910	1,200
R_TSIP_Aes256CbcEncryptFinal	550	550	550
R_TSIP_Aes256CbcDecryptInit	1,700	1,700	1,700
R_TSIP_Aes256CbcDecryptUpdate	800	1,100	1,300
R_TSIP_Aes256CbcDecryptFinal	560	560	560

Table 1-28 Performance of AES-GCM

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

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

Table 1-29 Performance of AES-CCM

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

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

Table 1-30 Performance of AES-CMAC

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

Table 1-31 Performance of AES Key Wrap

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

1.7.5 RX65N

Table 1-32 Performance of Common APIs

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

Table 1-33 Firmware Verification Performance

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

Table 1-34 Performance of AES

API	Performance (Unit: Cycle)		
	16-Byte Processing	48-Byte Processing	80-Byte Processing
R_TSIP_Aes128EcbEncryptInit	1,700	1,700	1,700
R_TSIP_Aes128EcbEncryptUpdate	520	660	840
R_TSIP_Aes128EcbEncryptFinal	450	450	450
R_TSIP_Aes128EcbDecryptInit	1,700	1,700	1,700
R_TSIP_Aes128EcbDecryptUpdate	590	730	910
R_TSIP_Aes128EcbDecryptFinal	460	460	460
R_TSIP_Aes256EcbEncryptInit	1,800	1,800	1,800
R_TSIP_Aes256EcbEncryptUpdate	540	690	870
R_TSIP_Aes256EcbEncryptFinal	450	450	450
R_TSIP_Aes256EcbDecryptInit	1,800	1,800	1,800
R_TSIP_Aes256EcbDecryptUpdate	610	760	940
R_TSIP_Aes256EcbDecryptFinal	470	470	470
R_TSIP_Aes128CbcEncryptInit	1,700	1,700	1,700
R_TSIP_Aes128CbcEncryptUpdate	590	730	910
R_TSIP_Aes128CbcEncryptFinal	480	480	480
R_TSIP_Aes128CbcDecryptInit	1,800	1,800	1,800
R_TSIP_Aes128CbcDecryptUpdate	660	790	970
R_TSIP_Aes128CbcDecryptFinal	490	500	500
R_TSIP_Aes256CbcEncryptInit	1,900	1,900	1,900
R_TSIP_Aes256CbcEncryptUpdate	600	750	930
R_TSIP_Aes256CbcEncryptFinal	480	480	480
R_TSIP_Aes256CbcDecryptInit	1,900	1,900	1,900
R_TSIP_Aes256CbcDecryptUpdate	680	820	1,000
R_TSIP_Aes256CbcDecryptFinal	490	490	490

Table 1-35 Performance of AES-GCM

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

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

Table 1-36 Performance of AES-CCM

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

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

Table 1-37 Performance of AES-CMAC

API	Performance (Unit: Cycle)		
	48-Byte Processing	64-Byte Processing	80-Byte Processing
R_TSIP_Aes128CmacGenerateInit	1,200	1,200	1,200
R_TSIP_Aes128CmacGenerateUpdate	670	720	760
R_TSIP_Aes128CmacGenerateFinal	800	800	800
R_TSIP_Aes128CmacVerifyInit	1,200	1,200	1,200
R_TSIP_Aes128CmacVerifyUpdate	680	720	770
R_TSIP_Aes128CmacVerifyFinal	1,700	1,700	1,700
R_TSIP_Aes256CmacGenerateInit	1,300	1,300	1,300
R_TSIP_Aes256CmacGenerateUpdate	720	760	810
R_TSIP_Aes256CmacGenerateFinal	830	830	830
R_TSIP_Aes256CmacVerifyInit	1,300	1,300	1,300
R_TSIP_Aes256CmacVerifyUpdate	710	750	810
R_TSIP_Aes256CmacVerifyFinal	1,800	1,800	1,800

Table 1-38 Performance of AES Key Wrap

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

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

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

Table 1-40 Performance of TDES

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

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

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

Table 1-42 Performance of ARC4

API	Performance (Unit: Cycle)		
	16-Byte Processing	48-Byte Processing	80-Byte Processing
R_TSIP_Arc4EncryptInit	2,100	2,100	2,100
R_TSIP_Arc4EncryptUpdate	500	630	810
R_TSIP_Arc4EncryptFinal	330	330	330
R_TSIP_Arc4DecryptInit	2,100	2,100	2,100
R_TSIP_Arc4DecryptUpdate	490	630	810
R_TSIP_Arc4DecryptFinal	320	330	330

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

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

Note: 1. Average value over 10 runs.

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

API	Performance (Unit: Cycle)		
	Message Size = 1 Byte	Message Size = 128 Bytes	Message Size = 256 Bytes
R_TSIP_RsassaPkcs1024SignatureGenerate	1,300,000	1,300,000	1,300,000
R_TSIP_RsassaPkcs1024SignatureVerification	18,000	20,000	20,000
R_TSIP_RsassaPkcs2048SignatureGenerate	27,000,000	27,000,000	27,000,000
R_TSIP_RsassaPkcs2048SignatureVerification	140,000	140,000	140,000

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

API	Performance (Unit: Cycle)		
	Message Size = 1 Byte	Message Size = 128 Bytes	Message Size = 256 Bytes
R_TSIP_RsassaPkcs1024SignatureGenerate	1,300,000	1,300,000	1,300,000
R_TSIP_RsassaPkcs1024SignatureVerification	18,000	19,000	20,000
R_TSIP_RsassaPkcs2048SignatureGenerate	27,000,000	27,000,000	27,000,000
R_TSIP_RsassaPkcs2048SignatureVerification	140,000	140,000	140,000

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

API	Performance (Unit: Cycle)		
	Message Size = 1 Byte	Message Size = 128 Bytes	Message Size = 256 Bytes
R_TSIP_RsassaPkcs1024SignatureGenerate	1,300,000	1,300,000	1,300,000
R_TSIP_RsassaPkcs1024SignatureVerification	18,000	19,000	19,000
R_TSIP_RsassaPkcs2048SignatureGenerate	27,000,000	27,000,000	27,000,000
R_TSIP_RsassaPkcs2048SignatureVerification	140,000	140,000	140,000

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

API	Performance (Unit: Cycle)	
	Message Size = 1 Byte	Message Size = 117 Bytes
R_TSIP_RsaesPkcs1024Encrypt	23,000	17,000
R_TSIP_RsaesPkcs1024Decrypt	1,300,000	1,300,000

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

API	Performance (Unit: Cycle)	
	Message Size = 1 Byte	Message Size = 245 Bytes
R_TSIP_RsaesPkcs2048Encrypt	150,000	140,000
R_TSIP_RsaesPkcs2048Decrypt	27,000,000	27,000,000

Table 1-49 Performance of HASH (SHA1)

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

Table 1-50 Performance of HASH (SHA256)

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

Table 1-51 Performance of HASH (MD5)

API	Performance (Unit: Cycle)		
	128-Byte Processing	192-Byte Processing	256-Byte Processing
R_TSIP_Md5Init	120	120	120
R_TSIP_Md5Update	1,500	1,700	1,900
R_TSIP_Md5Final	790	790	790

Table 1-52 Performance of Common APIs (HMAC Wrapped Key Generation)

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

Table 1-53 Performance of HMAC (SHA1)

API	Performance (Unit: Cycle)		
	128-Byte Processing	192-Byte Processing	256-Byte Processing
R_TSIP_Sha1HmacGenerateInit	1,400	1,400	1,400
R_TSIP_Sha1HmacGenerateUpdate	980	1,300	1,500
R_TSIP_Sha1HmacGenerateFinal	2,000	2,000	2,000
R_TSIP_Sha1HmacVerifyInit	1,400	1,400	1,400
R_TSIP_Sha1HmacVerifyUpdate	980	1,300	1,500
R_TSIP_Sha1HmacVerifyFinal	3,700	3,700	3,700

Table 1-54 Performance of HMAC (SHA256)

API	Performance (Unit: Cycle)		
	128-Byte Processing	192-Byte Processing	256-Byte Processing
R_TSIP_Sha256HmacGenerateInit	1,900	1,900	1,900
R_TSIP_Sha256HmacGenerateUpdate	920	1,200	1,400
R_TSIP_Sha256HmacGenerateFinal	2,000	2,000	2,000
R_TSIP_Sha256HmacVerifyInit	1,900	1,900	1,900
R_TSIP_Sha256HmacVerifyUpdate	920	1,200	1,400
R_TSIP_Sha256HmacVerifyFinal	3,700	3,700	3,700

Table 1-55 Performance of Common APIs (ECC Wrapped Key Generation)

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

Note: 1. Average value over 10 runs.

Table 1-56 Performance of ECDSA Signature Generation/Verification

API	Performance (Unit: Cycle)		
	Message Size = 1 Byte	Message Size = 128 Bytes	Message Size = 256 Bytes
R_TSIP_EcdsaP192SignatureGenerate	180,000	180,000	180,000
R_TSIP_EcdsaP224SignatureGenerate	180,000	190,000	180,000
R_TSIP_EcdsaP256SignatureGenerate	190,000	190,000	190,000
R_TSIP_EcdsaP384SignatureGenerate* ¹	1,200,000		
R_TSIP_EcdsaP192SignatureVerification	340,000	340,000	340,000
R_TSIP_EcdsaP224SignatureVerification	360,000	360,000	360,000
R_TSIP_EcdsaP256SignatureVerification	360,000	360,000	360,000
R_TSIP_EcdsaP384SignatureVerification* ¹	2,300,000		

Note: 1. Does not include SHA384 calculation.

Table 1-57 Key Exchange Performance

API	Performance (Unit: Cycle)
R_TSIP_EcdhP256Init	60
R_TSIP_EcdhP256ReadPublicKey	360,000
R_TSIP_EcdhP256MakePublicKey	340,000
R_TSIP_EcdhP256CalculateSharedSecretIndex	380,000
R_TSIP_EcdhP256KeyDerivation	3,800
R_TSIP_EcdheP512KeyAgreement	3,400,000
R_TSIP_Rsa2048DhKeyAgreement	53,000,000

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

1.7.6 RX671

Table 1-58 Performance of Common APIs

API	Performance (Unit: Cycle)
R_TSIP_Open	5,400,000
R_TSIP_Close	310
R_TSIP_GetVersion	24
R_TSIP_GenerateAes128KeyIndex	2,100
R_TSIP_GenerateAes256KeyIndex	2,200
R_TSIP_GenerateAes128RandomKeyIndex	1,200
R_TSIP_GenerateAes256RandomKeyIndex	1,700
R_TSIP_GenerateRandomNumber	540
R_TSIP_GenerateUpdateKeyRingKeyIndex	2,200
R_TSIP_UpdateAes128KeyIndex	1,800
R_TSIP_UpdateAes256KeyIndex	2,000

Table 1-59 Firmware Verification Performance

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

Table 1-60 Performance of AES

API	Performance (Unit: Cycle)		
	16-Byte Processing	48-Byte Processing	80-Byte Processing
R_TSIP_Aes128EcbEncryptInit	1,300	1,200	1,200
R_TSIP_Aes128EcbEncryptUpdate	390	500	30
R_TSIP_Aes128EcbEncryptFinal	330	320	320
R_TSIP_Aes128EcbDecryptInit	1,300	1,300	1,300
R_TSIP_Aes128EcbDecryptUpdate	460	560	700
R_TSIP_Aes128EcbDecryptFinal	340	340	340
R_TSIP_Aes256EcbEncryptInit	1,400	1,400	1,400
R_TSIP_Aes256EcbEncryptUpdate	410	530	660
R_TSIP_Aes256EcbEncryptFinal	330	320	320
R_TSIP_Aes256EcbDecryptInit	1,400	1,400	1,400
R_TSIP_Aes256EcbDecryptUpdate	470	580	720
R_TSIP_Aes256EcbDecryptFinal	340	330	330
R_TSIP_Aes128CbcEncryptInit	1,300	1,300	1,300
R_TSIP_Aes128CbcEncryptUpdate	430	540	670
R_TSIP_Aes128CbcEncryptFinal	340	340	340
R_TSIP_Aes128CbcDecryptInit	1,300	1,300	1,300
R_TSIP_Aes128CbcDecryptUpdate	490	600	730
R_TSIP_Aes128CbcDecryptFinal	350	350	350
R_TSIP_Aes256CbcEncryptInit	1,400	1,400	1,400
R_TSIP_Aes256CbcEncryptUpdate	460	580	710
R_TSIP_Aes256CbcEncryptFinal	340	340	340
R_TSIP_Aes256CbcDecryptInit	1,400	1,400	1,400
R_TSIP_Aes256CbcDecryptUpdate	530	640	770
R_TSIP_Aes256CbcDecryptFinal	350	350	350

Table 1-61 Performance of AES-GCM

API	Performance (Unit: Cycle)		
	48-Byte Processing	64-Byte Processing	80-Byte Processing
R_TSIP_Aes128GcmEncryptInit	4,200	4,100	4,100
R_TSIP_Aes128GcmEncryptUpdate	1,600	1,700	1,700
R_TSIP_Aes128GcmEncryptFinal	950	950	950
R_TSIP_Aes128GcmDecryptInit	4,100	4,100	4,100
R_TSIP_Aes128GcmDecryptUpdate	1,600	1,600	1,700
R_TSIP_Aes128GcmDecryptFinal	1,500	1,500	1,500
R_TSIP_Aes256GcmEncryptInit	4,200	4,100	4,100
R_TSIP_Aes256GcmEncryptUpdate	1,600	1,700	1,800
R_TSIP_Aes256GcmEncryptFinal	830	820	820
R_TSIP_Aes256GcmDecryptInit	4,200	4,100	4,100
R_TSIP_Aes256GcmDecryptUpdate	1,600	1,700	1,700
R_TSIP_Aes256GcmDecryptFinal	1,500	1,500	1,500

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

Table 1-62 Performance of AES-CCM

API	Performance (Unit: Cycle)		
	48-Byte Processing	64-Byte Processing	80-Byte Processing
R_TSIP_Aes128CcmEncryptInit	2,300	2,300	2,300
R_TSIP_Aes128CcmEncryptUpdate	890	960	1,100
R_TSIP_Aes128CcmEncryptFinal	760	750	750
R_TSIP_Aes128CcmDecryptInit	2,400	2,400	2,400
R_TSIP_Aes128CcmDecryptUpdate	810	880	960
R_TSIP_Aes128CcmDecryptFinal	1,500	1,500	1,500
R_TSIP_Aes256CcmEncryptInit	1,900	1,900	1,900
R_TSIP_Aes256CcmEncryptUpdate	940	1,100	1,200
R_TSIP_Aes256CcmEncryptFinal	770	770	770
R_TSIP_Aes256CcmDecryptInit	1,900	1,900	1,900
R_TSIP_Aes256CcmDecryptUpdate	850	930	1,100
R_TSIP_Aes256CcmDecryptFinal	1,500	1,500	1,500

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

Table 1-63 Performance of AES-CMAC

API	Performance (Unit: Cycle)		
	48-Byte Processing	64-Byte Processing	80-Byte Processing
R_TSIP_Aes128CmacGenerateInit	880	870	870
R_TSIP_Aes128CmacGenerateUpdate	490	520	560
R_TSIP_Aes128CmacGenerateFinal	630	620	620
R_TSIP_Aes128CmacVerifyInit	870	870	870
R_TSIP_Aes128CmacVerifyUpdate	490	530	570
R_TSIP_Aes128CmacVerifyFinal	1,300	1,300	1,300
R_TSIP_Aes256CmacGenerateInit	1,000	1,000	1,000
R_TSIP_Aes256CmacGenerateUpdate	520	550	600
R_TSIP_Aes256CmacGenerateFinal	650	630	630
R_TSIP_Aes256CmacVerifyInit	990	990	990
R_TSIP_Aes256CmacVerifyUpdate	510	550	600
R_TSIP_Aes256CmacVerifyFinal	1,300	1,300	1,300

Table 1-64 Performance of AES Key Wrap

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

Table 1-65 Performance of Common APIs (TDES Wrapped Key Generation)

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

Table 1-66 Performance of TDES

API	Performance (Unit: Cycle)		
	16-Byte Processing	48-Byte Processing	80-Byte Processing
R_TSIP_TdesEcbEncryptInit	800	790	790
R_TSIP_TdesEcbEncryptUpdate	430	620	820
R_TSIP_TdesEcbEncryptFinal	330	320	320
R_TSIP_TdesEcbDecryptInit	810	810	810
R_TSIP_TdesEcbDecryptUpdate	450	640	840
R_TSIP_TdesEcbDecryptFinal	330	330	330
R_TSIP_TdesCbcEncryptInit	850	840	840
R_TSIP_TdesCbcEncryptUpdate	490	680	880
R_TSIP_TdesCbcEncryptFinal	340	340	340
R_TSIP_TdesCbcDecryptInit	850	850	850
R_TSIP_TdesCbcDecryptUpdate	500	700	890
R_TSIP_TdesCbcDecryptFinal	350	350	350

Table 1-67 Performance of Common APIs (ARC4 Wrapped Key Generation)

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

Table 1-68 Performance of ARC4

API	Performance (Unit: Cycle)		
	16-Byte Processing	48-Byte Processing	80-Byte Processing
R_TSIP_Arc4EncryptInit	1,800	1,800	1,800
R_TSIP_Arc4EncryptUpdate	360	480	610
R_TSIP_Arc4EncryptFinal	240	230	230
R_TSIP_Arc4DecryptInit	1,800	1,800	1,800
R_TSIP_Arc4DecryptUpdate	360	480	610
R_TSIP_Arc4DecryptFinal	230	230	230

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

API	Performance (Unit: Cycle)
R_TSIP_GenerateRsa1024PublicKeyIndex	37,000
R_TSIP_GenerateRsa1024PrivateKeyIndex	38,000
R_TSIP_GenerateRsa2048PublicKeyIndex	140,000
R_TSIP_GenerateRsa2048PrivateKeyIndex	140,000
R_TSIP_GenerateRsa1024RandomKeyIndex* ¹	67,000,000
R_TSIP_GenerateRsa2048RandomKeyIndex* ¹	460,000,000
R_TSIP_UpdateRsa1024PublicKeyIndex	37,000
R_TSIP_UpdateRsa1024PrivateKeyIndex	38,000
R_TSIP_UpdateRsa2048PublicKeyIndex	140,000
R_TSIP_UpdateRsa2048PrivateKeyIndex	140,000

Note: 1. Average value over 10 runs.

Table 1-70 Performance of RSASSA-PKCS1-v1_5 Signature Generation/Verification (HASH = SHA1)

API	Performance (Unit: Cycle)		
	Message Size = 1 Byte	Message Size = 128 Bytes	Message Size = 256 Bytes
R_TSIP_RsassaPkcs1024SignatureGenerate	1,300,000	1,300,000	1,300,000
R_TSIP_RsassaPkcs1024SignatureVerification	17,000	18,000	18,000
R_TSIP_RsassaPkcs2048SignatureGenerate	27,000,000	27,000,000	27,000,000
R_TSIP_RsassaPkcs2048SignatureVerification	140,000	140,000	140,000

Table 1-71 Performance of RSASSA-PKCS1-v1_5 Signature Generation/Verification (HASH = SHA256)

API	Performance (Unit: Cycle)		
	Message Size = 1 Byte	Message Size = 128 Bytes	Message Size = 256 Bytes
R_TSIP_RsassaPkcs1024SignatureGenerate	1,300,000	1,300,000	1,300,000
R_TSIP_RsassaPkcs1024SignatureVerification	17,000	18,000	18,000
R_TSIP_RsassaPkcs2048SignatureGenerate	27,000,000	27,000,000	27,000,000
R_TSIP_RsassaPkcs2048SignatureVerification	140,000	140,000	140,000

Table 1-72 Performance of RSASSA-PKCS1-v1_5 Signature Generation/Verification (HASH = MD5)

API	Performance (Unit: Cycle)		
	Message Size = 1 Byte	Message Size = 128 Bytes	Message Size = 256 Bytes
R_TSIP_RsassaPkcs1024SignatureGenerate	1,300,000	1,300,000	1,300,000
R_TSIP_RsassaPkcs1024SignatureVerification	17,000	18,000	18,000
R_TSIP_RsassaPkcs2048SignatureGenerate	27,000,000	27,000,000	27,000,000
R_TSIP_RsassaPkcs2048SignatureVerification	140,000	140,000	140,000

Table 1-73 Performance of RSAES-PKCS1-v1_5 Encryption/Decryption with 1024-Bit Key Size

API	Performance (Unit: Cycle)	
	Message Size = 1 Byte	Message Size = 117 Bytes
R_TSIP_RsaesPkcs1024Encrypt	20,000	16,000
R_TSIP_RsaesPkcs1024Decrypt	1,300,000	1,300,000

Table 1-74 Performance of RSAES-PKCS1-v1_5 Encryption/Decryption with 2048-Bit Key Size

API	Performance (Unit: Cycle)	
	Message Size = 1 Byte	Message Size = 245 Bytes
R_TSIP_RsaesPkcs2048Encrypt	150,000	140,000
R_TSIP_RsaesPkcs2048Decrypt	27,000,000	27,000,000

Table 1-75 Performance of HASH (SHA1)

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

Table 1-76 Performance of HASH (SHA256)

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

Table 1-77 Performance of HASH (MD5)

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

Table 1-78 Performance of Common APIs (HMAC Wrapped Key Generation)

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

Table 1-79 Performance of HMAC (SHA1)

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

Table 1-80 Performance of HMAC (SHA256)

API	Performance (Unit: Cycle)		
	128-Byte Processing	192-Byte Processing	256-Byte Processing
R_TSIP_Sha256HmacGenerateInit	1,400	1,400	1,400
R_TSIP_Sha256HmacGenerateUpdate	750	910	1,100
R_TSIP_Sha256HmacGenerateFinal	1,600	1,600	1,600
R_TSIP_Sha256HmacVerifyInit	1,400	1,300	1,300
R_TSIP_Sha256HmacVerifyUpdate	730	910	1,100
R_TSIP_Sha256HmacVerifyFinal	2,700	2,700	2,700

Table 1-81 Performance of Common APIs (ECC Wrapped Key Generation)

API	Performance (Unit: Cycle)
R_TSIP_GenerateEccP192PublicKeyIndex	2,600
R_TSIP_GenerateEccP224PublicKeyIndex	2,600
R_TSIP_GenerateEccP256PublicKeyIndex	2,600
R_TSIP_GenerateEccP384PublicKeyIndex	2,800
R_TSIP_GenerateEccP192PrivateKeyIndex	2,300
R_TSIP_GenerateEccP224PrivateKeyIndex	2,300
R_TSIP_GenerateEccP256PrivateKeyIndex	2,300
R_TSIP_GenerateEccP384PrivateKeyIndex	2,300
R_TSIP_GenerateEccP192RandomKeyIndex* ¹	140,000
R_TSIP_GenerateEccP224RandomKeyIndex* ¹	150,000
R_TSIP_GenerateEccP256RandomKeyIndex* ¹	150,000
R_TSIP_GenerateEccP384RandomKeyIndex* ¹	1,100,000
R_TSIP_UpdateEccP192PublicKeyIndex	2,400
R_TSIP_UpdateEccP224PublicKeyIndex	2,300
R_TSIP_UpdateEccP256PublicKeyIndex	2,300
R_TSIP_UpdateEccP384PublicKeyIndex	2,500
R_TSIP_UpdateEccP192PrivateKeyIndex	2,100
sR_TSIP_UpdateEccP224PrivateKeyIndex	2,100
R_TSIP_UpdateEccP256PrivateKeyIndex	2,100
R_TSIP_UpdateEccP384PrivateKeyIndex	2,100

Note: 1. Average value over 10 runs.

Table 1-82 Performance of ECDSA Signature Generation/Verification

API	Performance (Unit: Cycle)		
	Message Size = 1 Byte	Message Size = 128 Bytes	Message Size = 256 Bytes
R_TSIP_EcdsaP192SignatureGenerate	170,000	170,000	170,000
R_TSIP_EcdsaP224SignatureGenerate	170,000	170,000	170,000
R_TSIP_EcdsaP256SignatureGenerate	170,000	180,000	170,000
R_TSIP_EcdsaP384SignatureGenerate* ¹	1,200,000		
R_TSIP_EcdsaP192SignatureVerification	310,000	320,000	310,000
R_TSIP_EcdsaP224SignatureVerification	330,000	330,000	330,000
R_TSIP_EcdsaP256SignatureVerification	330,000	340,000	330,000
R_TSIP_EcdsaP384SignatureVerification* ¹	2,200,000		

Note: 1. Does not include SHA384 calculation.

Table 1-83 Key Exchange Performance

API	Performance (Unit: Cycle)
R_TSIP_EcdhP256Init	44
R_TSIP_EcdhP256ReadPublicKey	340,000
R_TSIP_EcdhP256MakePublicKey	320,000
R_TSIP_EcdhP256CalculateSharedSecretIndex	360,000
R_TSIP_EcdhP256KeyDerivation	3,000
R_TSIP_EcdheP512KeyAgreement	3,300,000
R_TSIP_Rsa2048DhKeyAgreement	53,000,000

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

1.7.7 RX72M, RX72N

Table 1-84 Performance of Common APIs

API	Performance (Unit: Cycle)
R_TSIP_Open	6,300,000
R_TSIP_Close	310
R_TSIP_GetVersion	22
R_TSIP_GenerateAes128KeyIndex	2,200
R_TSIP_GenerateAes256KeyIndex	2,300
R_TSIP_GenerateAes128RandomKeyIndex	1,300
R_TSIP_GenerateAes256RandomKeyIndex	1,800
R_TSIP_GenerateRandomNumber	570
R_TSIP_GenerateUpdateKeyRingKeyIndex	2,300
R_TSIP_UpdateAes128KeyIndex	1,900
R_TSIP_UpdateAes256KeyIndex	2,100

Table 1-85 Firmware Verification Performance

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

Table 1-86 Performance of AES

API	Performance (Unit: Cycle)		
	16-Byte Processing	48-Byte Processing	80-Byte Processing
R_TSIP_Aes128EcbEncryptInit	1,300	1,300	1,300
R_TSIP_Aes128EcbEncryptUpdate	390	510	640
R_TSIP_Aes128EcbEncryptFinal	340	340	340
R_TSIP_Aes128EcbDecryptInit	1,300	1,300	1,300
R_TSIP_Aes128EcbDecryptUpdate	460	570	700
R_TSIP_Aes128EcbDecryptFinal	350	350	350
R_TSIP_Aes256EcbEncryptInit	1,400	1,400	1,400
R_TSIP_Aes256EcbEncryptUpdate	410	530	660
R_TSIP_Aes256EcbEncryptFinal	330	330	330
R_TSIP_Aes256EcbDecryptInit	1,400	1,400	1,400
R_TSIP_Aes256EcbDecryptUpdate	480	600	740
R_TSIP_Aes256EcbDecryptFinal	340	340	340
R_TSIP_Aes128CbcEncryptInit	1,400	1,400	1,400
R_TSIP_Aes128CbcEncryptUpdate	450	570	710
R_TSIP_Aes128CbcEncryptFinal	360	360	360
R_TSIP_Aes128CbcDecryptInit	1,400	1,400	1,400
R_TSIP_Aes128CbcDecryptUpdate	510	620	750
R_TSIP_Aes128CbcDecryptFinal	370	370	370
R_TSIP_Aes256CbcEncryptInit	1,500	1,500	1,500
R_TSIP_Aes256CbcEncryptUpdate	460	590	720
R_TSIP_Aes256CbcEncryptFinal	360	360	360
R_TSIP_Aes256CbcDecryptInit	1,500	1,500	1,500
R_TSIP_Aes256CbcDecryptUpdate	540	660	800
R_TSIP_Aes256CbcDecryptFinal	370	370	370

Table 1-87 Performance of AES-GCM

API	Performance (Unit: Cycle)		
	48-Byte Processing	64-Byte Processing	80-Byte Processing
R_TSIP_Aes128GcmEncryptInit	4,400	4,400	4,400
R_TSIP_Aes128GcmEncryptUpdate	1,600	1,700	1,800
R_TSIP_Aes128GcmEncryptFinal	1,100	1,100	1,100
R_TSIP_Aes128GcmDecryptInit	4,300	4,300	4,300
R_TSIP_Aes128GcmDecryptUpdate	1,600	1,700	1,800
R_TSIP_Aes128GcmDecryptFinal	1,700	1,700	1,700
R_TSIP_Aes256GcmEncryptInit	4,400	4,400	4,400
R_TSIP_Aes256GcmEncryptUpdate	1,700	1,800	1,800
R_TSIP_Aes256GcmEncryptFinal	860	860	860
R_TSIP_Aes256GcmDecryptInit	1,700	1,700	1,700
R_TSIP_Aes256GcmDecryptUpdate	1,700	1,700	1,800
R_TSIP_Aes256GcmDecryptFinal	1,500	1,500	1,500

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

Table 1-88 Performance of AES-CCM

API	Performance (Unit: Cycle)		
	48-Byte Processing	64-Byte Processing	80-Byte Processing
R_TSIP_Aes128CcmEncryptInit	2,400	2,400	2,400
R_TSIP_Aes128CcmEncryptUpdate	910	980	1,100
R_TSIP_Aes128CcmEncryptFinal	760	760	760
R_TSIP_Aes128CcmDecryptInit	2,500	2,500	2,500
R_TSIP_Aes128CcmDecryptUpdate	830	900	980
R_TSIP_Aes128CcmDecryptFinal	1,500	1,500	1,500
R_TSIP_Aes256CcmEncryptInit	2,000	2,000	2,000
R_TSIP_Aes256CcmEncryptUpdate	960	1,100	1,200
R_TSIP_Aes256CcmEncryptFinal	800	800	800
R_TSIP_Aes256CcmDecryptInit	2,000	2,000	2,000
R_TSIP_Aes256CcmDecryptUpdate	860	960	1,100
R_TSIP_Aes256CcmDecryptFinal	1,600	1,600	1,600

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

Table 1-89 Performance of AES-CMAC

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

Table 1-90 Performance of AES Key Wrap

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

Table 1-91 Performance of Common APIs (TDES Wrapped Key Generation)

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

Table 1-92 Performance of TDES

API	Performance (Unit: Cycle)		
	16-Byte Processing	48-Byte Processing	80-Byte Processing
R_TSIP_TdesEcbEncryptInit	830	830	830
R_TSIP_TdesEcbEncryptUpdate	440	640	840
R_TSIP_TdesEcbEncryptFinal	330	330	330
R_TSIP_TdesEcbDecryptInit	850	850	850
R_TSIP_TdesEcbDecryptUpdate	460	660	860
R_TSIP_TdesEcbDecryptFinal	340	340	350
R_TSIP_TdesCbcEncryptInit	880	890	890
R_TSIP_TdesCbcEncryptUpdate	490	690	890
R_TSIP_TdesCbcEncryptFinal	360	360	360
R_TSIP_TdesCbcDecryptInit	890	890	890
R_TSIP_TdesCbcDecryptUpdate	510	720	910
R_TSIP_TdesCbcDecryptFinal	370	370	370

Table 1-93 Performance of Common APIs (ARC4 Wrapped Key Generation)

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

Table 1-94 Performance of ARC4

API	Performance (Unit: Cycle)		
	16-Byte Processing	48-Byte Processing	80-Byte Processing
R_TSIP_Arc4EncryptInit	1,900	1,900	1,900
R_TSIP_Arc4EncryptUpdate	370	490	620
R_TSIP_Arc4EncryptFinal	240	240	240
R_TSIP_Arc4DecryptInit	1,900	1,900	1,900
R_TSIP_Arc4DecryptUpdate	370	490	620
R_TSIP_Arc4DecryptFinal	240	240	240

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

API	Performance (Unit: Cycle)
R_TSIP_GenerateRsa1024PublicKeyIndex	37,000
R_TSIP_GenerateRsa1024PrivateKeyIndex	38,000
R_TSIP_GenerateRsa2048PublicKeyIndex	140,000
R_TSIP_GenerateRsa2048PrivateKeyIndex	140,000
R_TSIP_GenerateRsa1024RandomKeyIndex* ¹	61,000,000
R_TSIP_GenerateRsa2048RandomKeyIndex* ¹	450,000,000
R_TSIP_UpdateRsa1024PublicKeyIndex	37,000
R_TSIP_UpdateRsa1024PrivateKeyIndex	38,000
R_TSIP_UpdateRsa2048PublicKeyIndex	140,000
R_TSIP_UpdateRsa2048PrivateKeyIndex	140,000

Note: 1. Average value over 10 runs.

Table 1-96 Performance of RSASSA-PKCS1-v1_5 Signature Generation/Verification (HASH = SHA1)

API	Performance (Unit: Cycle)		
	Message Size = 1 Byte	Message Size = 128 Bytes	Message Size = 256 Bytes
R_TSIP_RsassaPkcs1024SignatureGenerate	1,300,000	1,300,000	1,300,000
R_TSIP_RsassaPkcs1024SignatureVerification	17,000	18,000	18,000
R_TSIP_RsassaPkcs2048SignatureGenerate	27,000,000	27,000,000	27,000,000
R_TSIP_RsassaPkcs2048SignatureVerification	140,000	140,000	140,000

Table 1-97 Performance of RSASSA-PKCS1-v1_5 Signature Generation/Verification (HASH = SHA256)

API	Performance (Unit: Cycle)		
	Message Size = 1 Byte	Message Size = 128 Bytes	Message Size = 256 Bytes
R_TSIP_RsassaPkcs1024SignatureGenerate	1,300,000	1,300,000	1,300,000
R_TSIP_RsassaPkcs1024SignatureVerification	17,000	18,000	18,000
R_TSIP_RsassaPkcs2048SignatureGenerate	27,000,000	27,000,000	27,000,000
R_TSIP_RsassaPkcs2048SignatureVerification	140,000	140,000	140,000

Table 1-98 Performance of RSASSA-PKCS1-v1_5 Signature Generation/Verification (HASH = MD5)

API	Performance (Unit: Cycle)		
	Message Size = 1 Byte	Message Size = 128 Bytes	Message Size = 256 Bytes
R_TSIP_RsassaPkcs1024SignatureGenerate	1,300,000	1,300,000	1,300,000
R_TSIP_RsassaPkcs1024SignatureVerification	17,000	18,000	18,000
R_TSIP_RsassaPkcs2048SignatureGenerate	27,000,000	27,000,000	27,000,000
R_TSIP_RsassaPkcs2048SignatureVerification	140,000	140,000	140,000

Table 1-99 Performance of RSAES-PKCS1-v1_5 Encryption/Decryption with 1024-Bit Key Size

API	Performance (Unit: Cycle)	
	Message Size = 1 Byte	Message Size = 117 Bytes
R_TSIP_RsaesPkcs1024Encrypt	21,000	16,000
R_TSIP_RsaesPkcs1024Decrypt	1,300,000	1,300,000

Table 1-100 Performance of RSAES-PKCS1-v1_5 Encryption/Decryption with 2048-Bit Key Size

API	Performance (Unit: Cycle)	
	Message Size = 1 Byte	Message Size = 245 Bytes
R_TSIP_RsaesPkcs2048Encrypt	150,000	140,000
R_TSIP_RsaesPkcs2048Decrypt	27,000,000	27,000,000

Table 1-101 Performance of HASH (SHA1)

API	Performance (Unit: Cycle)		
	128-Byte Processing	192-Byte Processing	256-Byte Processing
R_TSIP_Sha1Init	100	110	110
R_TSIP_Sha1Update	1,300	1,500	1,700
R_TSIP_Sha1Final	670	680	670

Table 1-102 Performance of HASH (SHA256)

API	Performance (Unit: Cycle)		
	128-Byte Processing	192-Byte Processing	256-Byte Processing
R_TSIP_Sha256Init	110	110	110
R_TSIP_Sha256Update	1,300	1,500	1,700
R_TSIP_Sha256Final	640	640	640

Table 1-103 Performance of HASH (MD5)

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

Table 1-104 Performance of Common APIs (HMAC Wrapped Key Generation)

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

Table 1-105 Performance of HMAC (SHA1)

API	Performance (Unit: Cycle)		
	128-Byte Processing	192-Byte Processing	256-Byte Processing
R_TSIP_Sha1HmacGenerateInit	1,100	1,100	1,100
R_TSIP_Sha1HmacGenerateUpdate	810	1,100	1,300
R_TSIP_Sha1HmacGenerateFinal	1,700	1,700	1,700
R_TSIP_Sha1HmacVerifyInit	1,100	1,100	1,100
R_TSIP_Sha1HmacVerifyUpdate	810	1,100	1,300
R_TSIP_Sha1HmacVerifyFinal	2,800	2,800	2,800

Table 1-106 Performance of HMAC (SHA256)

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

Table 1-107 Performance of Common APIs (ECC Wrapped Key Generation)

API	Performance (Unit: Cycle)
R_TSIP_GenerateEccP192PublicKeyIndex	2,700
R_TSIP_GenerateEccP224PublicKeyIndex	2,700
R_TSIP_GenerateEccP256PublicKeyIndex	2,700
R_TSIP_GenerateEccP384PublicKeyIndex	2,900
R_TSIP_GenerateEccP192PrivateKeyIndex	2,400
R_TSIP_GenerateEccP224PrivateKeyIndex	2,400
R_TSIP_GenerateEccP256PrivateKeyIndex	2,400
R_TSIP_GenerateEccP384PrivateKeyIndex	2,400
R_TSIP_GenerateEccP192RandomKeyIndex* ¹	140,000
R_TSIP_GenerateEccP224RandomKeyIndex* ¹	150,000
R_TSIP_GenerateEccP256RandomKeyIndex* ¹	150,000
R_TSIP_GenerateEccP384RandomKeyIndex* ¹	1,100,000
R_TSIP_UpdateEccP192PublicKeyIndex	2,500
R_TSIP_UpdateEccP224PublicKeyIndex	2,500
R_TSIP_UpdateEccP256PublicKeyIndex	2,500
R_TSIP_UpdateEccP384PublicKeyIndex	2,600
R_TSIP_UpdateEccP192PrivateKeyIndex	2,200
R_TSIP_UpdateEccP224PrivateKeyIndex	2,200
R_TSIP_UpdateEccP256PrivateKeyIndex	2,200
R_TSIP_UpdateEccP384PrivateKeyIndex	2,200

Note: 1. Average value over 10 runs.

Table 1-108 Performance of ECDSA Signature Generation/Verification

API	Performance (Unit: Cycle)		
	Message Size = 1 Byte	Message Size = 128 Bytes	Message Size = 256 Bytes
R_TSIP_EcdsaP192SignatureGenerate	170,000	170,000	170,000
R_TSIP_EcdsaP224SignatureGenerate	170,000	170,000	170,000
R_TSIP_EcdsaP256SignatureGenerate	180,000	170,000	170,000
R_TSIP_EcdsaP384SignatureGenerate* ¹	1,200,000		
R_TSIP_EcdsaP192SignatureVerification	310,000	310,000	310,000
R_TSIP_EcdsaP224SignatureVerification	330,000	330,000	340,000
R_TSIP_EcdsaP256SignatureVerification	340,000	340,000	340,000
R_TSIP_EcdsaP384SignatureVerification* ¹	2,100,000		

Note: 1. Does not include SHA384 calculation.

Table 1-109 Key Exchange Performance

API	Performance (Unit: Cycle)
R_TSIP_EcdhP256Init	42
R_TSIP_EcdhP256ReadPublicKey	340,000
R_TSIP_EcdhP256MakePublicKey	320,000
R_TSIP_EcdhP256CalculateSharedSecretIndex	360,000
R_TSIP_EcdhP256KeyDerivation	3,200
R_TSIP_EcdheP512KeyAgreement	3,300,000
R_TSIP_Rsa2048DhKeyAgreement	53,000,000

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

2. API Information

2.1 Hardware Requirements

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

2.2 Software Requirements

The TSIP drivers are dependent on the following module:

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

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

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

```

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

```

[RX26T]

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

```

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

```

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

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

```

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

```

[RX66N, RX671, RX72M, and RX72N]

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

```

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

```

[RX65N]

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

```

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

```

2.3 Supported Toolchain

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

2.4 Header File

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

2.5 Integer Types

The TSIP driver uses ANSI C99 integer types defined in `stdint.h`.

2.6 Configuration

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

Configuration options in <code>r_config/r_tsip_rx_config.h</code>		
Definition	Default Value	Remarks
TSIP_AES_128_ECB_ENCRYPT	(1)	—
TSIP_AES_256_ECB_ENCRYPT	(1)	—
TSIP_AES_128_ECB_DECRYPT	(1)	—
TSIP_AES_256_ECB_DECRYPT	(1)	—
TSIP_AES_128_CBC_ENCRYPT	(1)	—
TSIP_AES_256_CBC_ENCRYPT	(1)	—
TSIP_AES_128_CBC_DECRYPT	(1)	—
TSIP_AES_256_CBC_DECRYPT	(1)	—
TSIP_AES_128_CTR	(1)	—
TSIP_AES_256_CTR	(1)	—
TSIP_AES_128_GCM_ENCRYPT	(1)	—
TSIP_AES_256_GCM_ENCRYPT	(1)	—
TSIP_AES_128_GCM_DECRYPT	(1)	—
TSIP_AES_256_GCM_DECRYPT	(1)	—
TSIP_AES_128_CMAC	(1)	—
TSIP_AES_256_CMAC	(1)	—
TSIP_AES_128_CCM_ENCRYPT	(1)	—
TSIP_AES_256_CCM_ENCRYPT	(1)	—
TSIP_AES_128_CCM_DECRYPT	(1)	—
TSIP_AES_256_CCM_DECRYPT	(1)	—
TSIP_AES_128_KEY_WRAP	(1)	—
TSIP_AES_256_KEY_WRAP	(1)	—
TSIP_TDES_ECB_ENCRYPT	(1)	—
TSIP_TDES_ECB_DECRYPT	(1)	—
TSIP_TDES_CBC_ENCRYPT	(1)	—
TSIP_TDES_CBC_DECRYPT	(1)	—
TSIP_ARC4_ENCRYPT	(1)	—
TSIP_ARC4_DECRYPT	(1)	—
TSIP_SHA_1	(1)	—
TSIP_SHA_256	(1)	—
TSIP_MD5	(1)	—
TSIP_SHA_1_HMAC	(1)	—
TSIP_SHA_256_HMAC	(1)	—
TSIP_RSAES_1024	(1)	—
TSIP_RSAES_2048	(1)	—
TSIP_RSAES_3072	(1)	—
TSIP_RSAES_4096	(1)	—
TSIP_RSASSA_1024	(1)	—
TSIP_RSASSA_2048	(1)	—
TSIP_RSASSA_3072	(1)	—
TSIP_RSASSA_4096	(1)	—

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

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

2.7 Data Structures

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

2.8 Return Values

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

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

2.9 Adding the FIT Module to Your Project

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

- (1) Adding the FIT module to your project using Smart Configurator in e² studio
Using Smart Configurator in e² studio allows you to add the FIT module to your project automatically. Refer to the application note “Renesas e² studio Smart Configurator User Guide” (R20AN0451) for details.
- (2) Adding the FIT module to your project using FIT Configurator in e² studio
Using FIT Configurator in e² studio allows you to add the FIT module to your project automatically. Refer to the application note “Adding Firmware Integration Technology Modules to Projects (R01AN1723)” for details.
- (3) Adding the FIT module to your project using Smart Configurator in CS+
Using Smart Configurator Standalone Version in CS+ allows you to add the FIT module to your project automatically. Refer to the application note “Renesas e² studio Smart Configurator User Guide (R20AN0451)” for details.
- (4) Adding the FIT module to your project in CS+
Manually add the FIT module to your project in CS+. Refer to the application note “Adding Firmware Integration Technology Modules to CS+ Projects (R01AN1826)” for details.

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

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

Devices for which `WAIT_LOOP` appears in the comments:

All device groups

A code sample is shown below.

Example *while* statement:

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

Example *for* statement:

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

Example *do while* statement:

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

3. TSIP Driver Usage

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

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

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

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

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

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

3.1 Recovering after Unauthorized Access Detection

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

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

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

To recover from unauthorized access detection, close the TSIP driver once with `R_TSIP_Close()` and restart the TSIP with `R_TSIP_Open()`, or reset the device.

3.2 Avoiding TSIP Access Conflicts

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

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

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

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

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

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

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

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

3.3 BSP FIT Module Integration

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

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

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

3.4 Single-Part and Multi-Part Operations

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

APIs for multi-part operations are provided for symmetric key cryptography and hashes (message digest generation and HMAC functions), and APIs for single-part operations are provided for other ciphers.

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

All multi-part operations conform to the following pattern:

Init: Initialize and start the operation.

If initialization is successful, the operation is active. If initialization fails, the operation enters an error state.

Update: Update the operation.

The update function can provide additional parameters, supply data for processing, or generate output.

If updating is successful, the operation remains active. If updating fails, the operation enters an error state.

Final: Call the applicable finalizing function to end the operation.

This function accepts any final input, generates any final output, and then releases any resources associated with the operation.

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

3.5 Initializing and Terminating the Driver

The driver provides APIs for the following driver management operations:

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

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

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

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

3.6 Random Number Generation

The driver provides an API for generating random numbers.

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

3.7 Key Management

The driver provides APIs for the following key management operations:

No.	API	Description
1	R_TSIP_GenerateUpdateKeyRingKeyIndex R_TSIP_GenerateAesXXXKeyIndex R_TSIP_GenerateTdesKeyIndex R_TSIP_GenerateArc4KeyIndex R_TSIP_GenerateShaXXXHmacKeyIndex R_TSIP_GenerateRsaXXXPublicKeyIndex R_TSIP_GenerateRsaXXXPrivateKeyIndex R_TSIP_GenerateEccPXXXPublicKeyIndex R_TSIP_GenerateEccPXXXPrivateKeyIndex R_TSIP_GenerateTlsRsaPublicKeyIndex	These key injection APIs use the Renesas Key Wrap service to convert a user key into a wrapped key wrapped using an HUK. They can be used for key injection at the factory. [Aes] XXX = 128, 256 [Hmac] XXX = 1, 256 [Rsa] XXX = 1024, 2048, 3072*1, 4096*1 [Ecc] XXX = 192, 224, 256, 384
2	R_TSIP_UpdateAesXXXKeyIndex R_TSIP_UpdateTdesKeyIndex R_TSIP_UpdateArc4KeyIndex R_TSIP_UpdateRsaXXXPublicKeyIndex R_TSIP_UpdateRsaXXXPrivateKeyIndex R_TSIP_UpdateEccPXXXPublicKeyIndex R_TSIP_UpdateEccPXXXPrivateKeyIndex	These key update APIs use an update keyring to convert a user key into a wrapped key wrapped using an HUK. They can be used to update keys in the field. [Aes] XXX = 128, 256 [Hmac] XXX = 1, 256 [Rsa] XXX = 1024, 2048, 3072*1, 4096*1 [Ecc] XXX = 192, 224, 256, 384
3	R_TSIP_GenerateAesXXXRandomKeyIndex R_TSIP_GenerateTdesRandomKeyIndex R_TSIP_GenerateArc4RandomKeyIndex R_TSIP_GenerateRsaXXXRandomKeyIndex R_TSIP_GenerateEccPXXXRandomKeyIndex	These APIs generate a random key and convert it to a wrapped key. [Aes] XXX = 128, 256 [Rsa] XXX = 1024, 2048 [Ecc] XXX = 192, 224, 256, 384

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

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

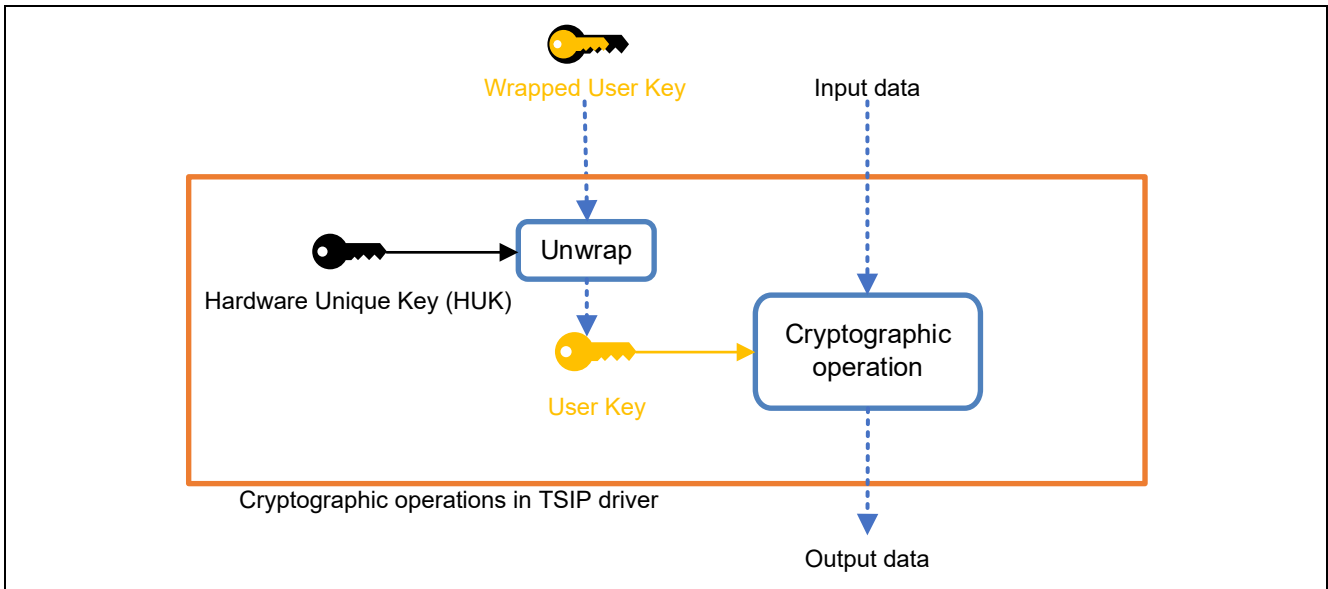


Figure 3.1 Key Handling in TSIP Driver Cryptographic Operations

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

of the TSIP. The wrapping of the wrapped key can be unlocked only by the TSIP, and the unwrapped key exists only within the TSIP during cryptographic processing. Since the wrapped key has been wrapped using a device-specific key, it cannot be unwrapped using a different device-specific key, even if the wrapped key is copied from the nonvolatile memory of one device to another device.

3.7.1 Key Injection and Updating

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

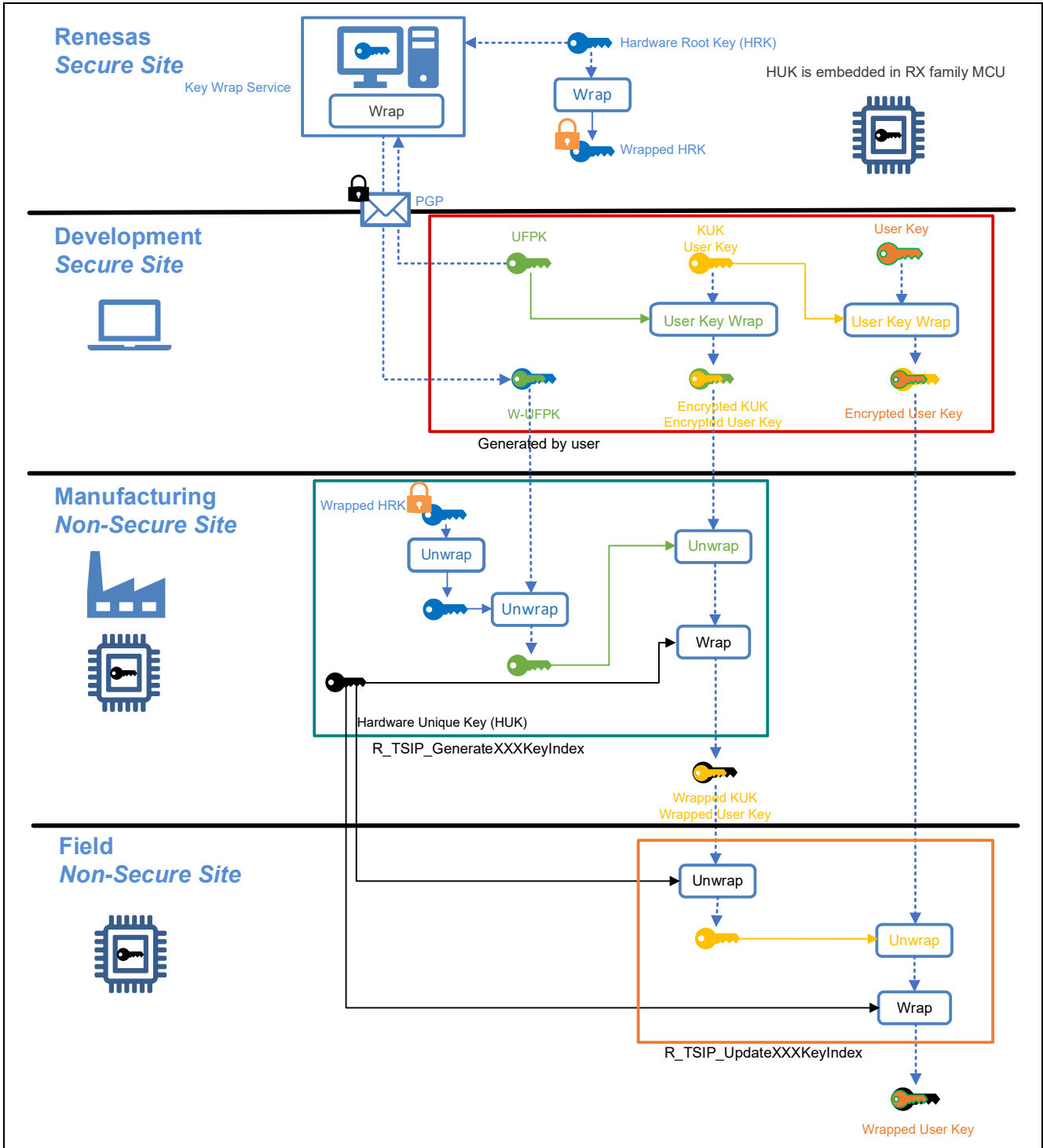


Figure 3.2 Key Injection and Key Updating Operation Sequence

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

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

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

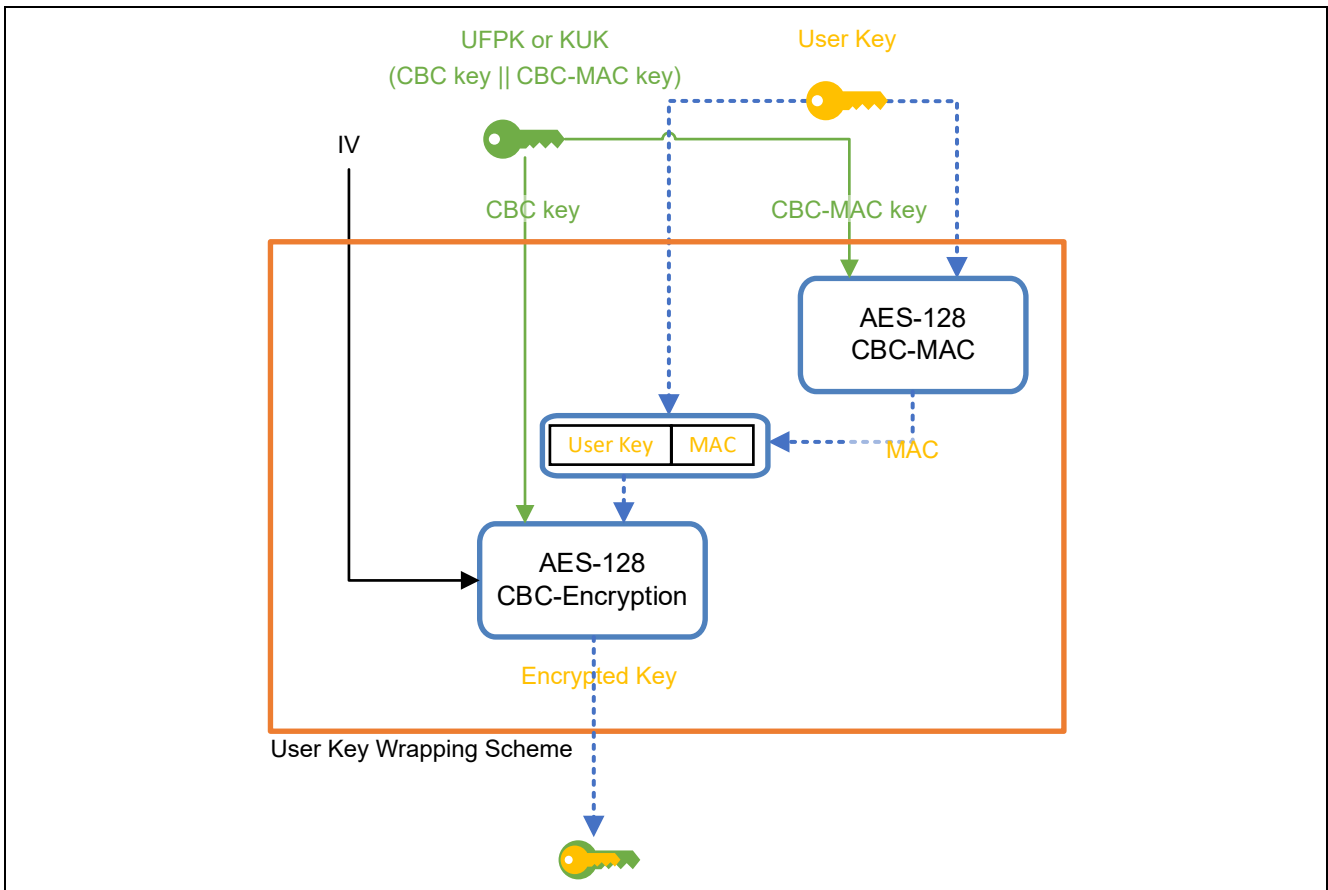


Figure 3.3 User Key Wrapping Scheme during Key Injection and Key Updating

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

3.7.2 Key Generation

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

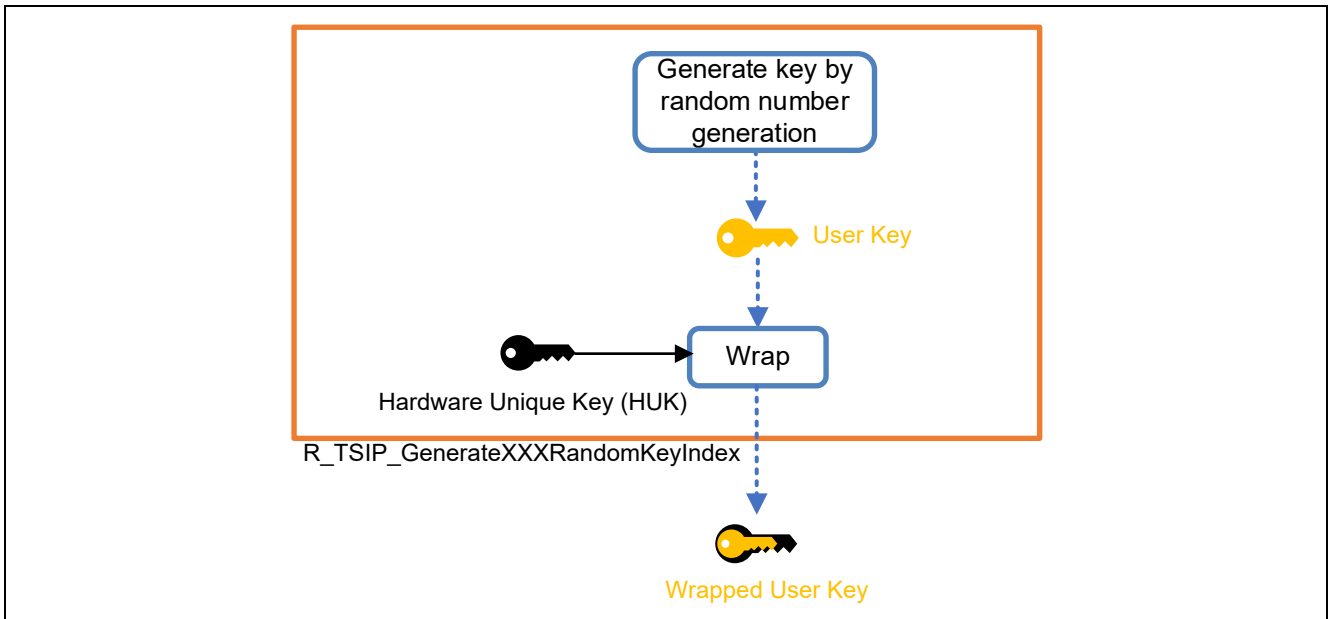


Figure 3.4 Key Generation Sequence

3.7.3 Plaintext Public Key Extraction

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

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

3.8 Symmetric Key Cryptography

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

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

* = Init, Update, Final

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

3.8.1 Symmetric Key Cryptography

The encryption operations for each AES mode operate as follows:

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

The decryption operations for each AES mode operate as follows:

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

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

3.8.2 Authenticated Encryption with Associated Data (AEAD)

The AES-GCM encryption operations operate as follows:

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

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

The AES-GCM decryption operations operate as follows:

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

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

3.8.3 Message Authentication Code (MAC)

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

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

AES-CMAC verification operates as follows:

Call `R_TSIP_AesXXXCmacVerifyInit()` to specify the required key.

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

3.9 Asymmetric Cryptography

The driver provides APIs for the following asymmetric cryptographic operations:

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

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

3.10 Hash Functions

The driver provides APIs for the following hash operations:

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

* = Init, Update, Final

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

3.10.1 Message Digest (Hash Function)

The hash operation APIs are used as follows:

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

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

3.10.2 Message Authentication Code (HMAC)

The HMAC generation APIs are used as follows:

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

The HMAC verification APIs are used as follows:

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

3.11 ECDH Key Exchange

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

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

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

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

3.11.1 Ephemeral Unified Model (2e, 0s)

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

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

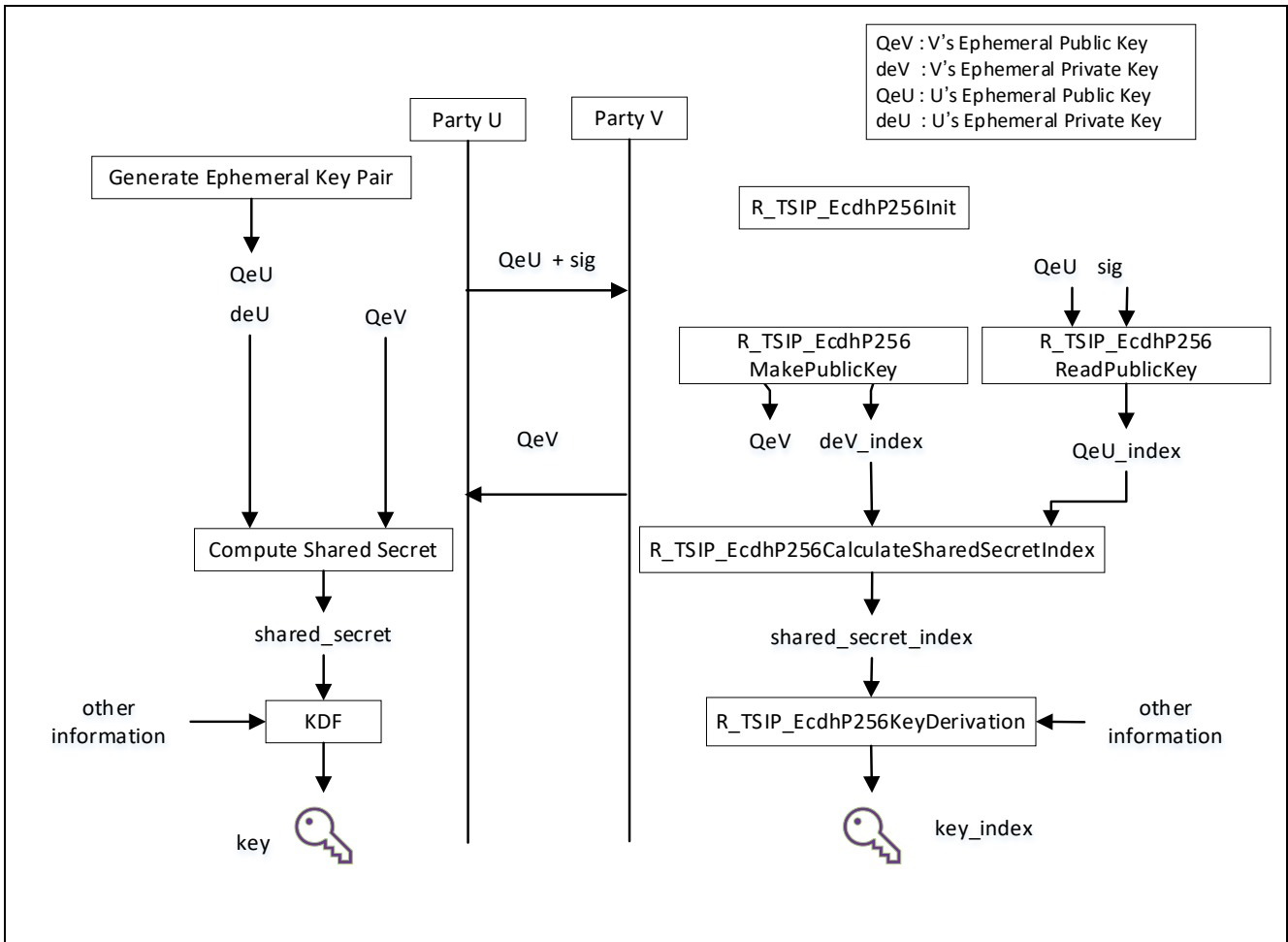


Figure 3.5 Key Exchange Scheme Using Ephemeral Unified Model

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

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

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

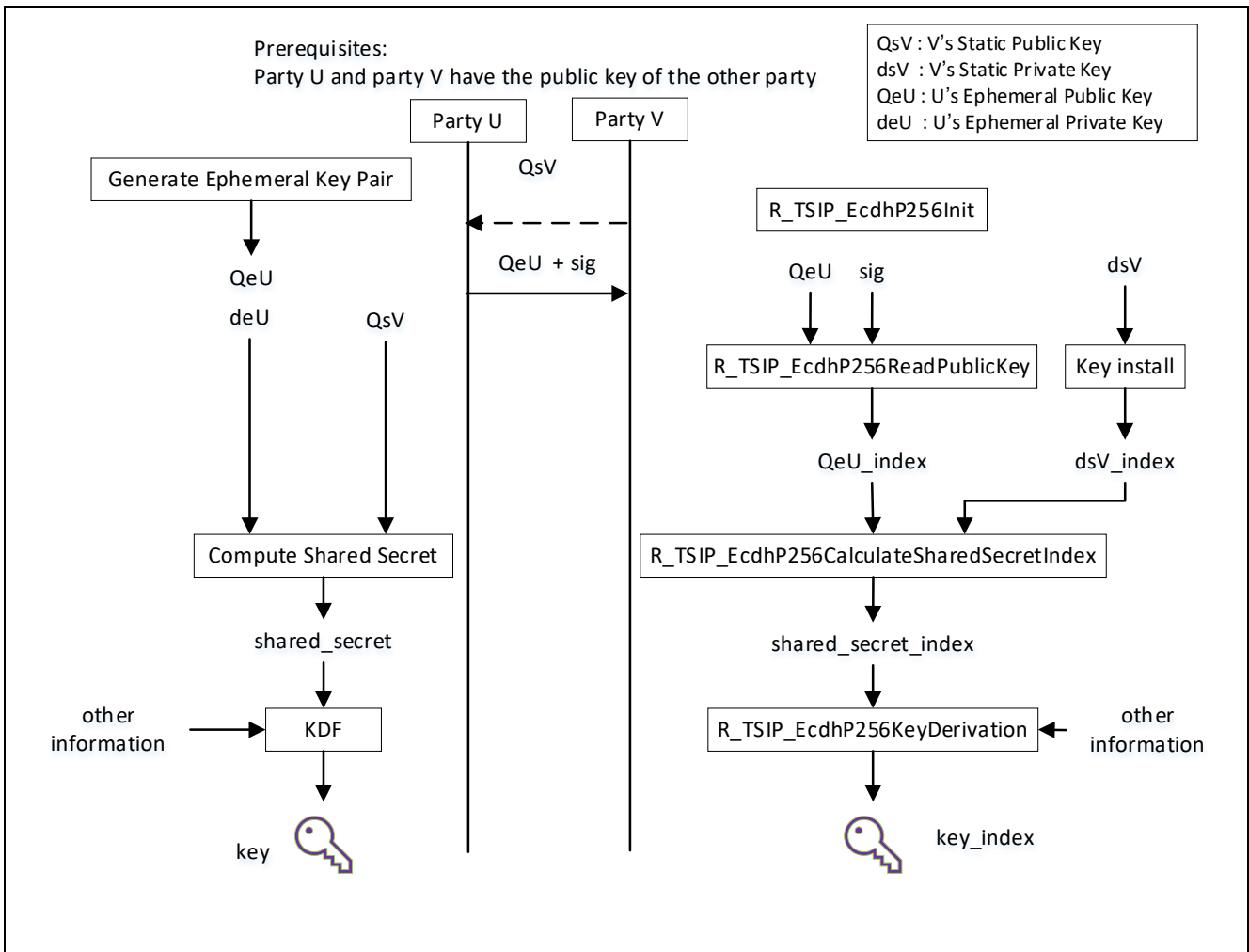


Figure 3.6 Key Exchange Scheme Using One-Pass Diffie-Hellman

3.11.3 Static Unified Model (0e, 2s)

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

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

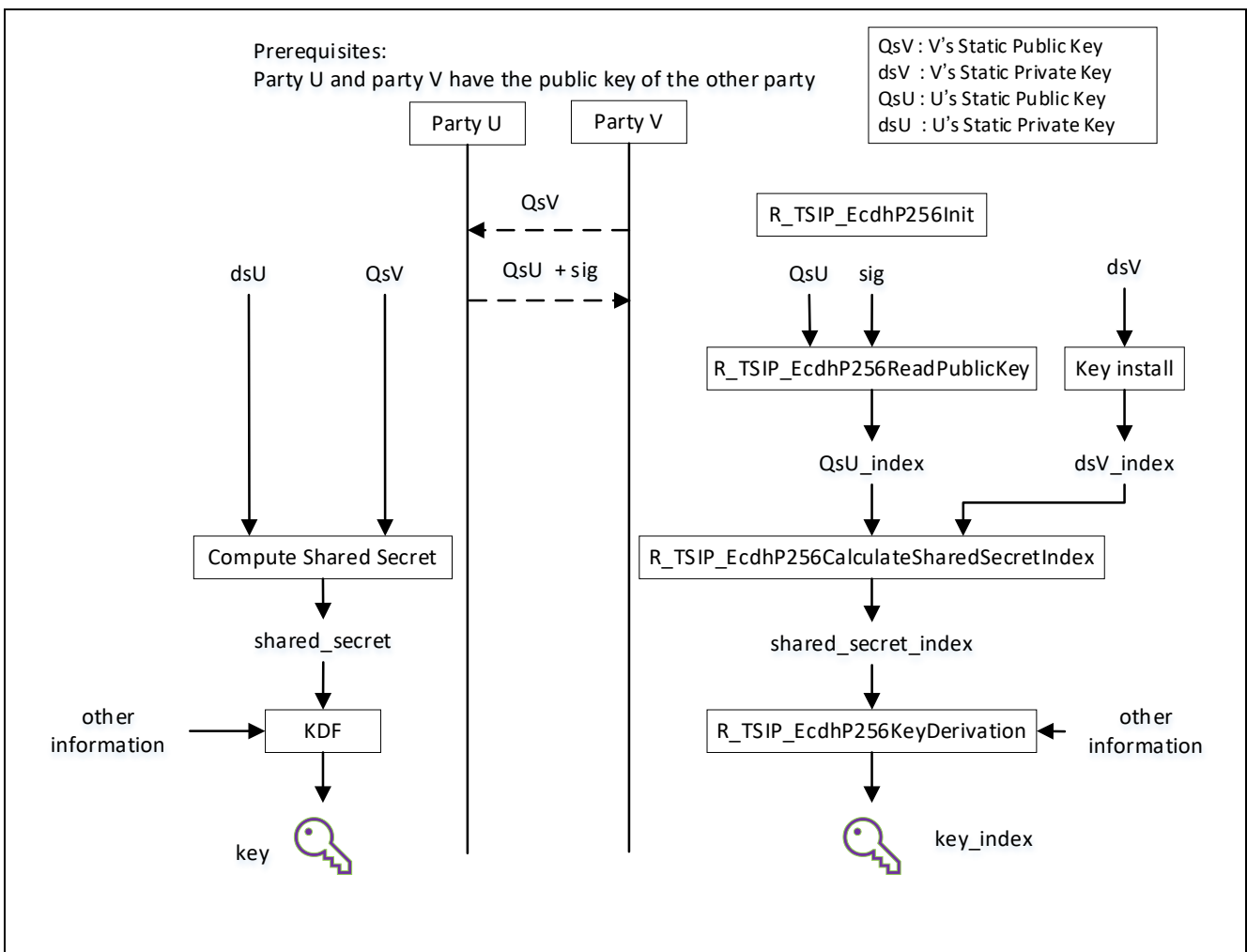


Figure 3.7 Key Exchange Scheme Using Static Unified Model

3.12 TLS Cooperation Function (TLS 1.2)

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

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

3.12.1 TLS 1.2 Client Function

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

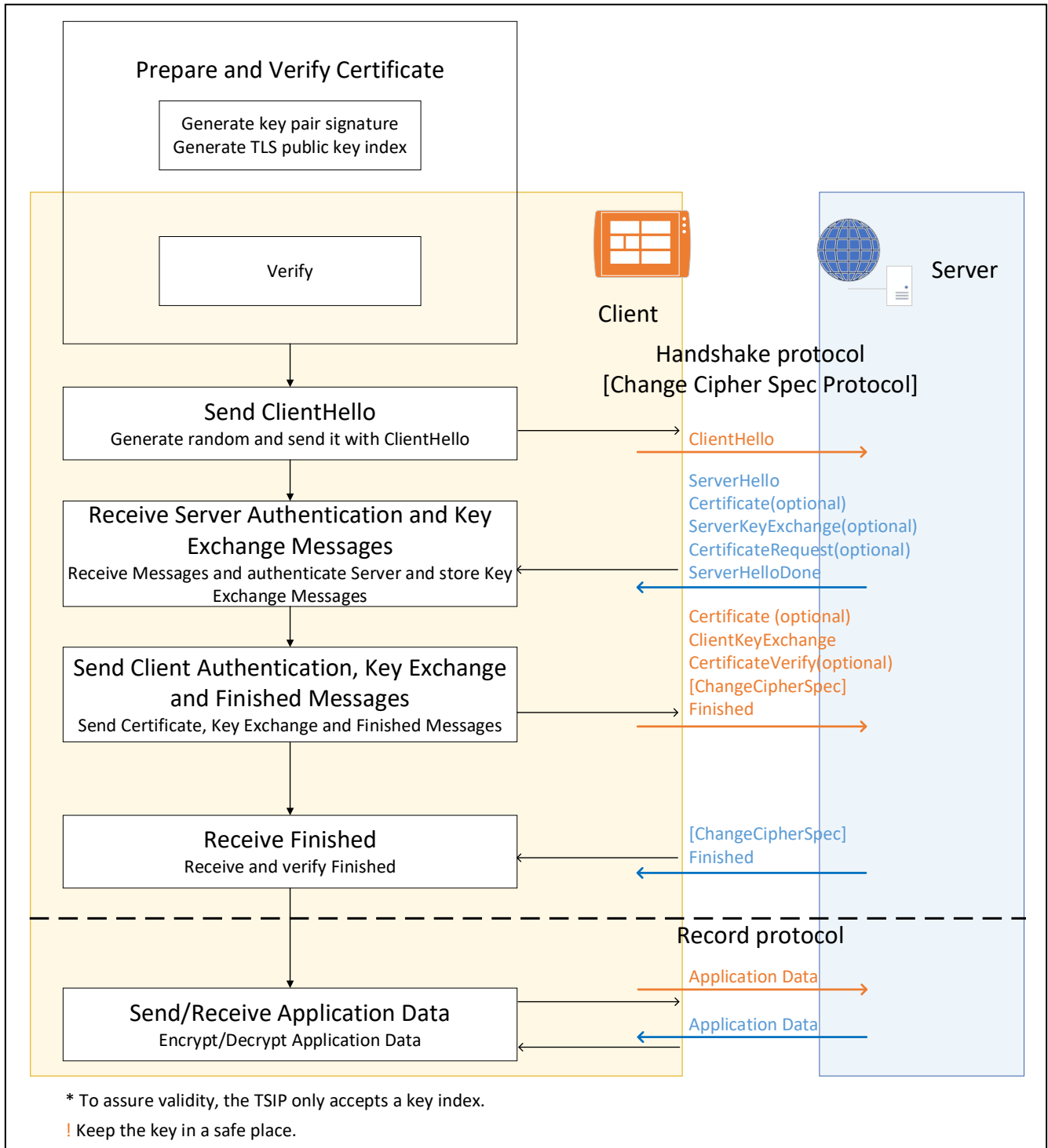


Figure 3.8 Implementation Using TLS 1.2 Client Function

1. Preparing and Verifying Certificates

1.1 Before using TLS cooperation function, prepare the items below.

- root CA certificate which the server connects to
- RSA 2048bit key pair
- client certificate.

1.2 Generate signature value of the prepared root CA certificate with using the private key of the key pair. The signature algorithm is RSA2048 PSS with SHA256. And generate Encrypted Key of the public key of the key pair with using UFPK. Note that the above procedure must be executed in a safe site.

When there are multiple servers connects to and needed to register multiple root CA certificate, generate signature value of the root CA certificate bundle. It is possible to prevent making connection with an unintentional server with generating signature of the root CA certificate.

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

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

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

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

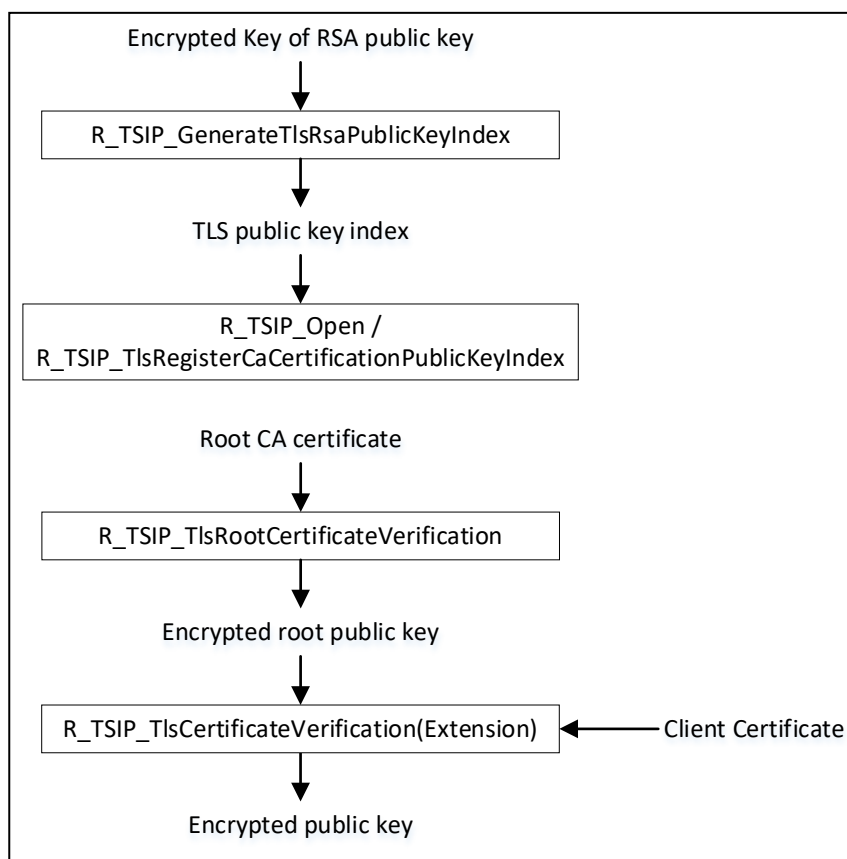


Figure 3.9 Preparing and Verifying Certificates

2. Sending ClientHello
 - 2.1 Send ClientHello.
3. Receiving Server Authentication and Key Exchange Messages
 - 3.1 Receive ServerHello and the random field in it.
 - 3.2 When (Server) certificate is received, use `R_TSIP_TlsCertificateVerification()` or `R_TSIP_TlsCertificateVerificationExtension()` to generate an encrypted public key.
 - 3.3-a When the key exchange method is ECDHE, extract the signature and ECDH public key from `ServerKeyExchange` and generate pre-master secret.
 - 3.3-a.1 Use `R_TSIP_GenerateTlsP256EccKeyIndex()` to generate an ECC key pair.
 - 3.3-a.2 Use `R_TSIP_TlsServersEphemeralEcdhPublicKeyRetrieves()` with the ECDH public key, (Server) certificate and handshake messages as input to generate encrypted ephemeral ECDH public key.
 - 3.3-a.3 Use `R_TSIP_TlsGeneratePreMasterSecretWithEccP256Key()` with the ephemeral ECDH public key and the wrapped ECC private key generated in 3.3-a.1 as input to generate encrypted ephemeral pre-master secret.
 - 3.3-b When the key exchange method is RSA, nothing to do here.

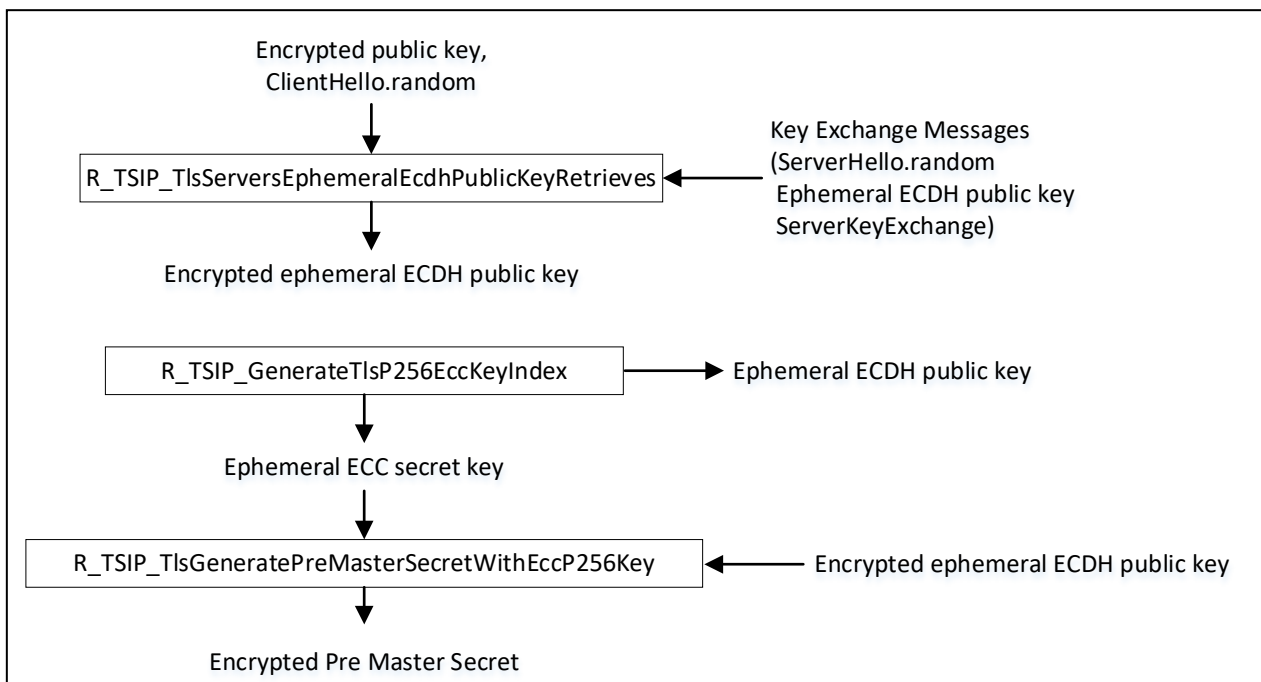
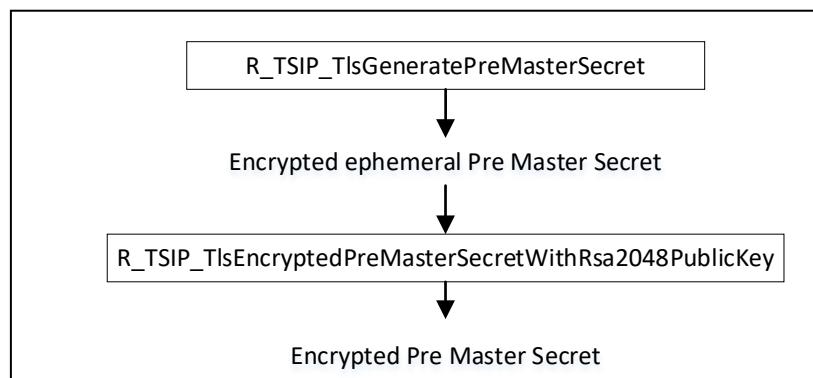


Figure 3.10 Receiving Server Authentication and Key Exchange Messages

4. Sending Client Authentication and Key Exchange Messages
 - 4.1 When `CertificateRequest` is received, send client certificate as (Client) Certificate to the server.
 - 4.2-a When the key exchange method is ECDHE, generate `ClientKeyExchange` with using ECDH public key.
 - 4.2-a.1 Use ECDH public key generated in 3.3.1 as `ClientECDiffieHelmanPublic` field in `ClientKeyExchange`.
 - 4.2-b When the key exchange method is RSA, generate `ClientKeyExchange` with generating pre-master secret.
 - 4.2-b.1 Use `R_TSIP_TlsGeneratePreMasterSecret()` to generate an encrypted ephemeral pre-master secret.

- 4.2-b.2 Use `R_TSIP_TlsEncryptPreMasterSecretWithRsa2048PublicKey()` with the encrypted ephemeral pre-master secret and (Server) Certificate as input to generate an encrypted ephemeral pre-master secret to use in `ClientKeyExchange`.
- 4.3 When `CertificateVerify` is needed, use `R_TSIP_EcdsaP256SignatureGenerate()`, `R_TSIP_RsassaPkcs2048SignatureGenerate()` or `R_TSIP_RsassaPss24048SignatureGenerate()` to generate it.
- 4.4 Use `R_TSIP_TlsGenerateMasterSecret()` with encrypted ephemeral pre-master secret as input to generate encrypted ephemeral master secret.
- 4.5 When extended master secret is to be generated, use `R_TSIP_TlsGenerateExtendedMasterSecret()` with encrypted ephemeral pre-master secret as input to generate encrypted ephemeral extended master secret.
- 4.6 Use `R_TSIP_TlsGenerateSessionKey()` with encrypted ephemeral master secret or encrypted ephemeral extended master secret and each handshake message as input to generate each wrapped key to use in the TLS connection.
- 4.7 Generate (Client) Finished.
- 4.7.1 Use `R_TSIP_TlsGenerateVerifyData()` with encrypted ephemeral master secret or encrypted ephemeral extended master secret and hash value of the handshake messages as input to generate (Client) `VerifyData` to use in (Client) Finished.
- 4.7.2 When AES CBC mode is used as the cipher suite, use APIs of AES CBC encryption and HMAC generation with wrapped AES key and wrapped MAC key for Client -> Server communication generated in 4.6 and (Client) `VerifyData` generated in 4.7.1 as input to generate (Client) Finished.
- 4.7.3 When AES GCM mode is used as the cipher suite, use APIs of AES GCM encryption with wrapped AES key for Client -> Server communication generated in 4.6 and (Client) `VerifyData` generated in 4.7.1 as input to generate (Client) Finished.
- 4.8 Send each authentication and key exchange messages.



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

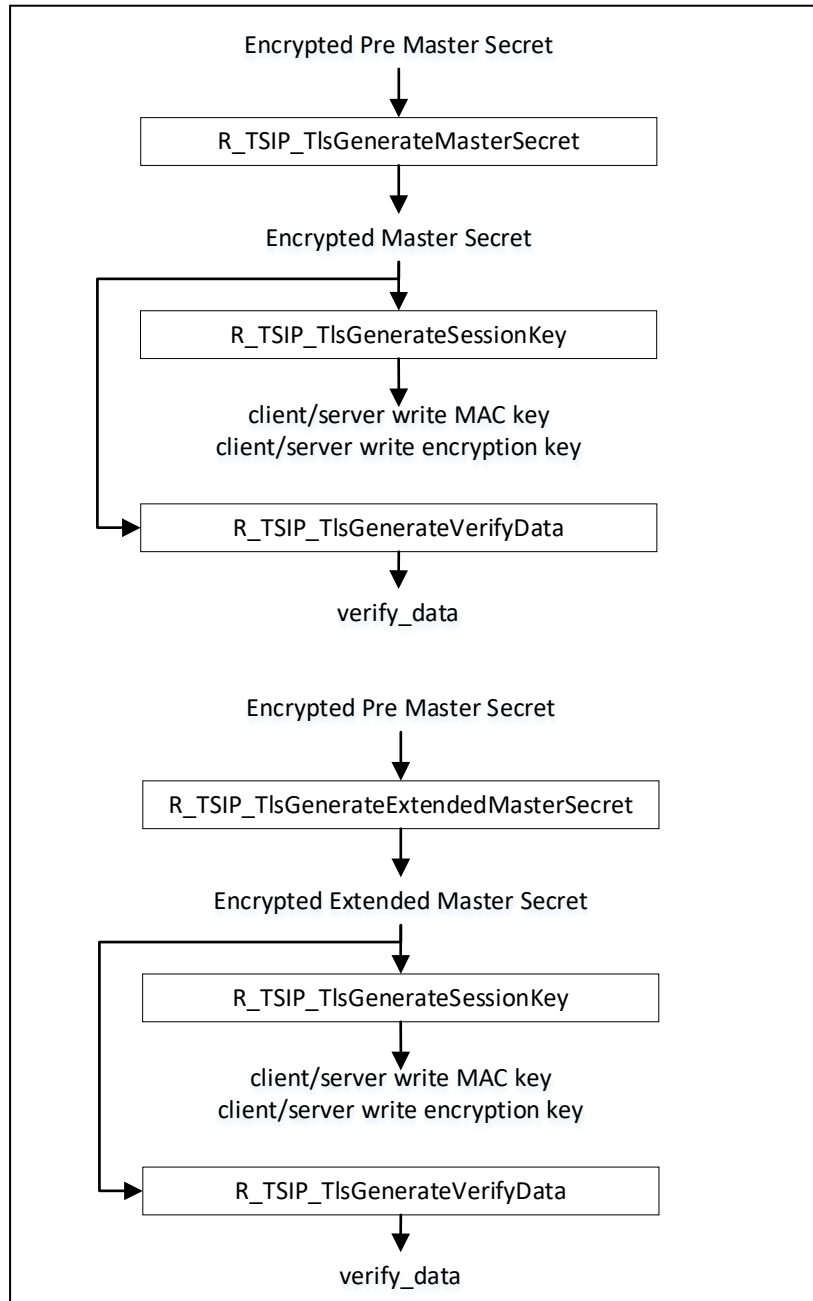


Figure 3.12 Sending Client Authentication and Key Exchange Messages (Generating VerifyData)

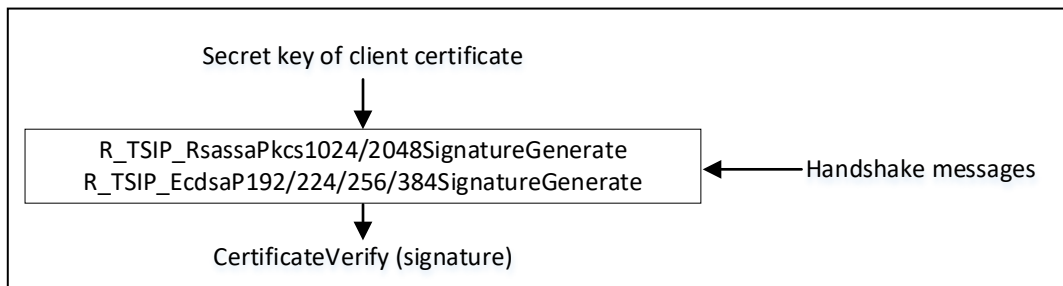


Figure 3.13 Sending Client Authentication and Key Exchange Messages (Generating CertificateVerify)

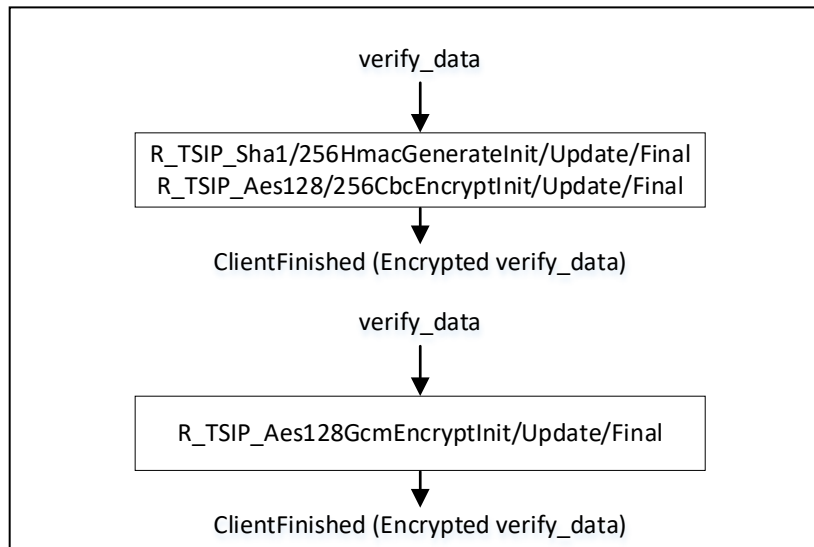


Figure 3.14 Sending Client Authentication and Key Exchange Messages (Generating ClientFinished)

5. Receiving Finished

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

5.2 Verify (Server) Finished.

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

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

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

6. Sending/Receiving Application Data

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

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

3.12.2 TLS 1.2 Server Function

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

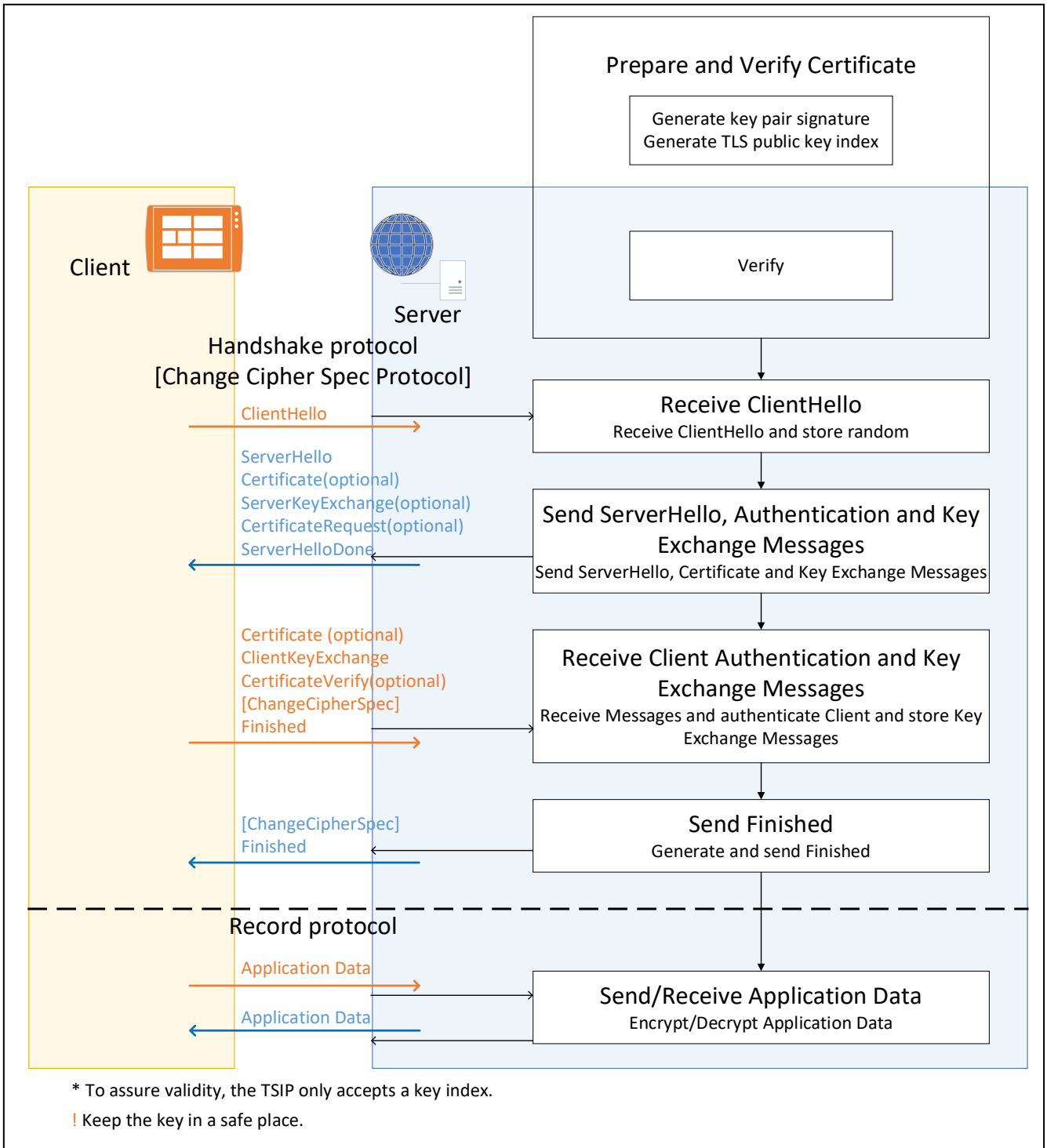


Figure 3.15 Implementation Using TLS 1.2 Server Function

1. Preparing and Verifying Certificates

1.1 Before using TLS cooperation function, prepare the items below.

- root CA certificate which the server connects to
- RSA 2048bit key pair
- server certificate.

1.2 Generate signature value of the prepared root CA certificate with using the private key of the key pair. The signature algorithm is RSA2048 PSS with SHA256. And generate Encrypted Key of the public key of the key pair with using UFPK. Note that the above procedure must be executed in a safe site.

When multiple root CA certificate are needed to register, generate signature value of the root CA certificate bundle. It is possible to prevent making connection with using an unauthorized root CA certificate.

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

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

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

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

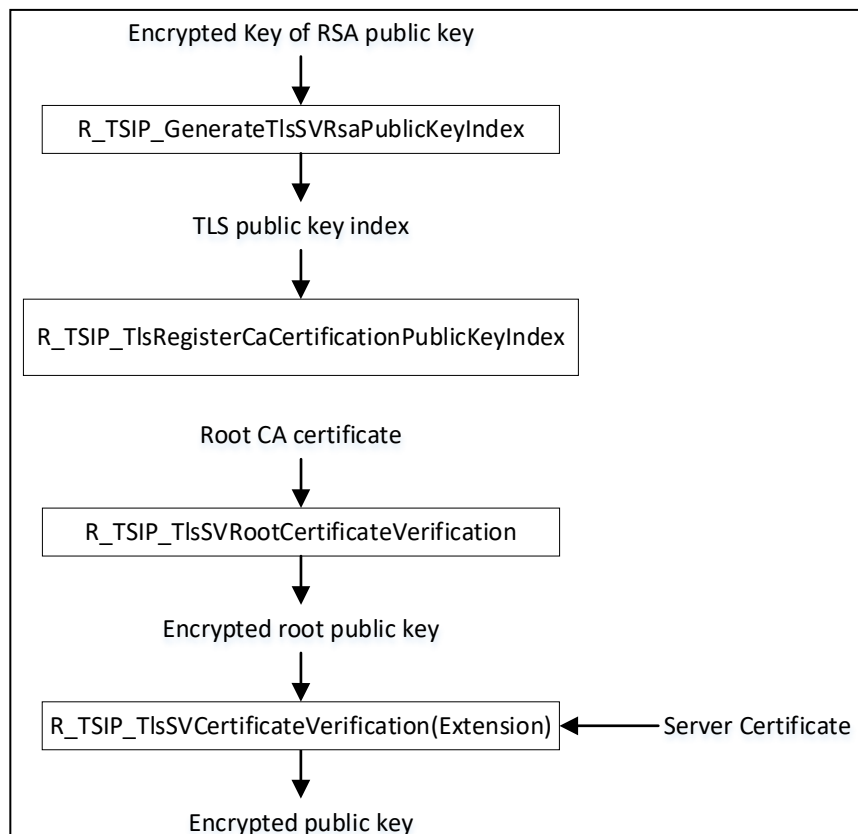


Figure 3.16 Preparing and Verifying Certificates

2. Receiving ClientHello
 - 2.1 Receive ClientHello and the random field in it.
3. Sending ServerHello, Authentication and Key Exchange Messages
 - 3.1 Use R_TSIP_TlsSVGenerateServerRandom() to generate a random number value to use in ServerHello.
 - 3.2-a When the key exchange method is ECDHE, generate ServerKeyExchange.
 - 3.2-a.1 Use R_TSIP_GenerateTlsSVP256EccKeyIndex() to generate an ECC key pair. The ECDH public key of the key pair is used as public field of params in ServerKeyExchange.
 - 3.2-a.2 Use R_TSIP_EcdsaP256SignatureGenerate() with the wrapped private key of (server) certificate and hash value of the handshake messages as input to generate the signature. The signature is used as signed_params field in ServerKeyExchange.
 - 3.2-b When the key exchange method is RSA, nothing to do here.
 - 3.3 Send ServerHello, authentication and key exchange method is RSA, nothing to do here.

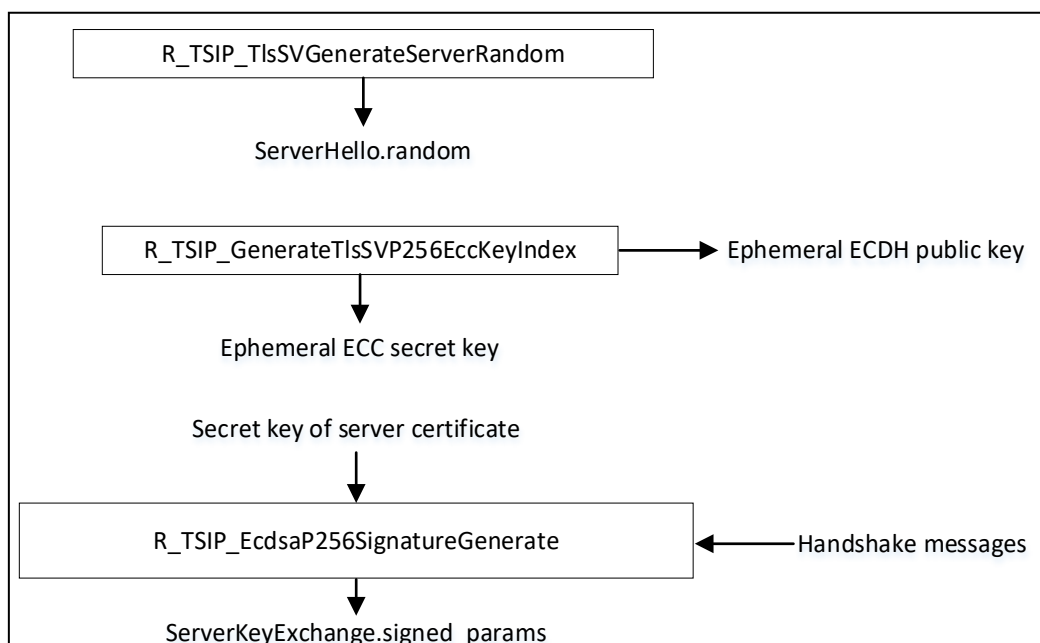


Figure 3.17 Sending ServerHello, Authentication and Key Exchange Messages

4. Receiving Client Authentication and Key Exchange Messages
 - 4.1 When (Client) Certificate is received, use R_TSIP_TlsSVCertificateVerification() or R_TSIP_TlsSVCertificateVerificationExtension() to generate an encrypted public key.
 - 4.2-a When the key exchange method is ECDHE, extract ECDH public key from ClientKeyExchange and generate pre-master secret.
 - 4.2-a.1 Use R_TSIP_TlsSVGeneratePreMasterSecretWithEccP256Key() with the ECDH public key in ClientKeyExchange and wrapped ECC private key generated in 3.3.1 as input to generate encrypted ephemeral pre-master secret.
 - 4.2-b When the key exchange method is RSA, extract pre-master secret from ClientKeyExchange.
 - 4.2-b.1 Use R_TSIP_TlsSVDecryptPreMasterSecretWithRsa2048PrivateKey() with encrypted pre-master secret in ClientKeyExchange and wrapped RSA private key of (Server) Certificate as input to generate the encrypted ephemeral pre-master secret.

- 4.3 Use `R_TSIP_TlsSVGenerateMasterSecret()` with encrypted ephemeral pre-master secret in `ClientKeyExchange` and wrapped RSA private key of (Server) Certificate as input to generate the encrypted ephemeral master secret.
- 4.4 When extended master secret is to be generated, use `R_TSIP_TlsSVGenerateExtendedMasterSecret()` with encrypted ephemeral pre-master secret as input to generate encrypted ephemeral extended master secret.
- 4.5 When `CertificateVerify` is received, use `R_TSIP_TlsSVCertificateVerifyVerification()` to verify it.
- 4.6 Use `R_TSIP_TlsSVGenerateSessionKey()` with encrypted ephemeral master secret or encrypted ephemeral extended master secret and each handshake message as input to generate each wrapped key to use in the TLS connection.
- 4.7 Verify (Client) Finished.
- 4.7.1 Use `R_TSIP_TlsSVGenerateVerifyData()` with encrypted ephemeral master secret or encrypted ephemeral extended master secret and hash value of the handshake messages as input to generate (Client) `VerifyData` to compare to (Client) Finished.
- 4.7.2-a When AES CBC mode is used as the cipher suite, use APIs of AES CBC decryption and HMAC verification with wrapped AES key and wrapped MAC key for Client -> Server communication generated in 4.6 and (Client) Finished as input to decrypt (Client) `VerifyData`.
- 4.7.2-b When AES GCM mode is used as the cipher suite, use APIs of AES GCM decryption with wrapped AES key for Client -> Server communication generated in 4.6 and (Client) Finished as input to decrypt (Client) `VerifyData`.
- 4.7.3 Check the equality between the (Client) `VerifyData` generated in 4.7.1 and the `VerifyData` decrypted in 4.7.2-a or 4.7.2-b.

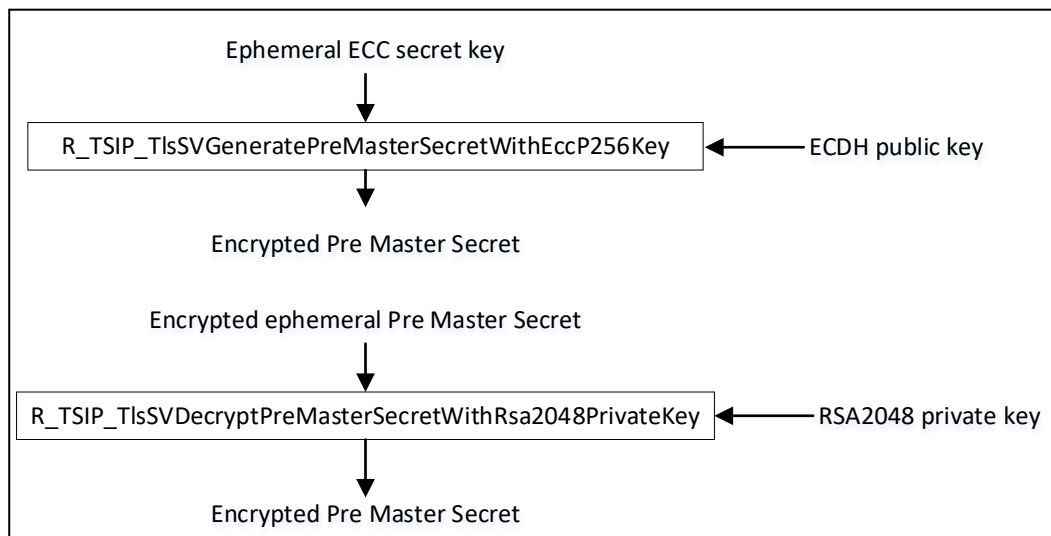


Figure 3.18 Receiving Client Authentication and Key Exchange Messages (Generating Pre Master Secret)

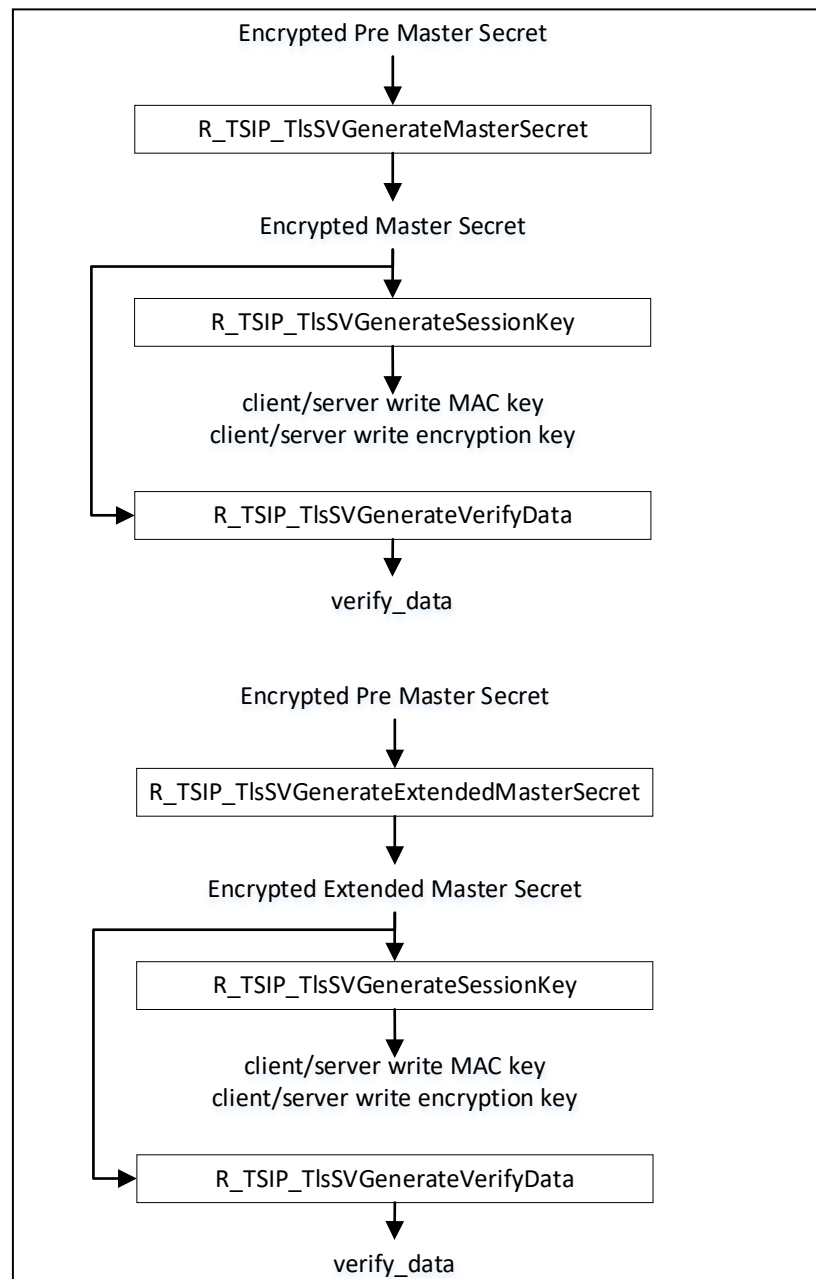


Figure 3.19 Receiving Client Authentication and Key Exchange Messages (Generating VerifyData)

5. Sending Finished

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

5.2 Encrypt (Server) Finished.

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

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

5.3 Send (Server) Finished.

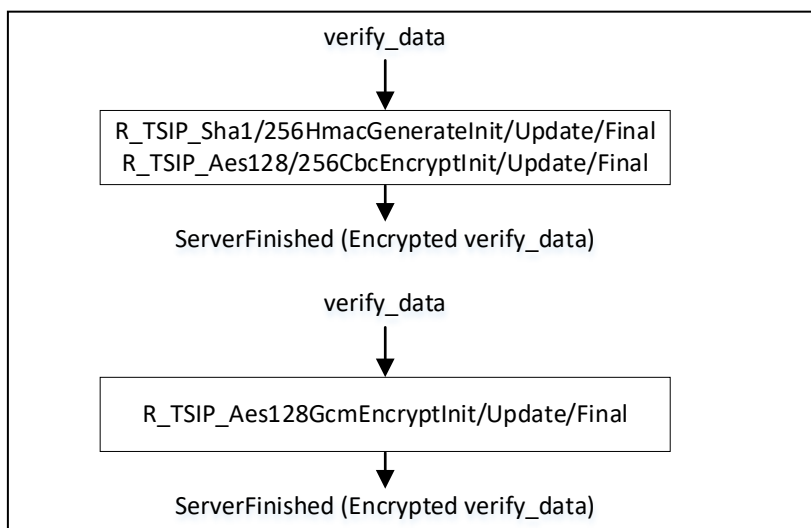


Figure 3.20 Sending Finished

6. Sending/Receiving Application Data

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

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

3.13 TLS Cooperation Function (TLS 1.3)

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

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

3.13.1 TLS 1.3 Client Function

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

3.13.1.1 Full Handshake

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

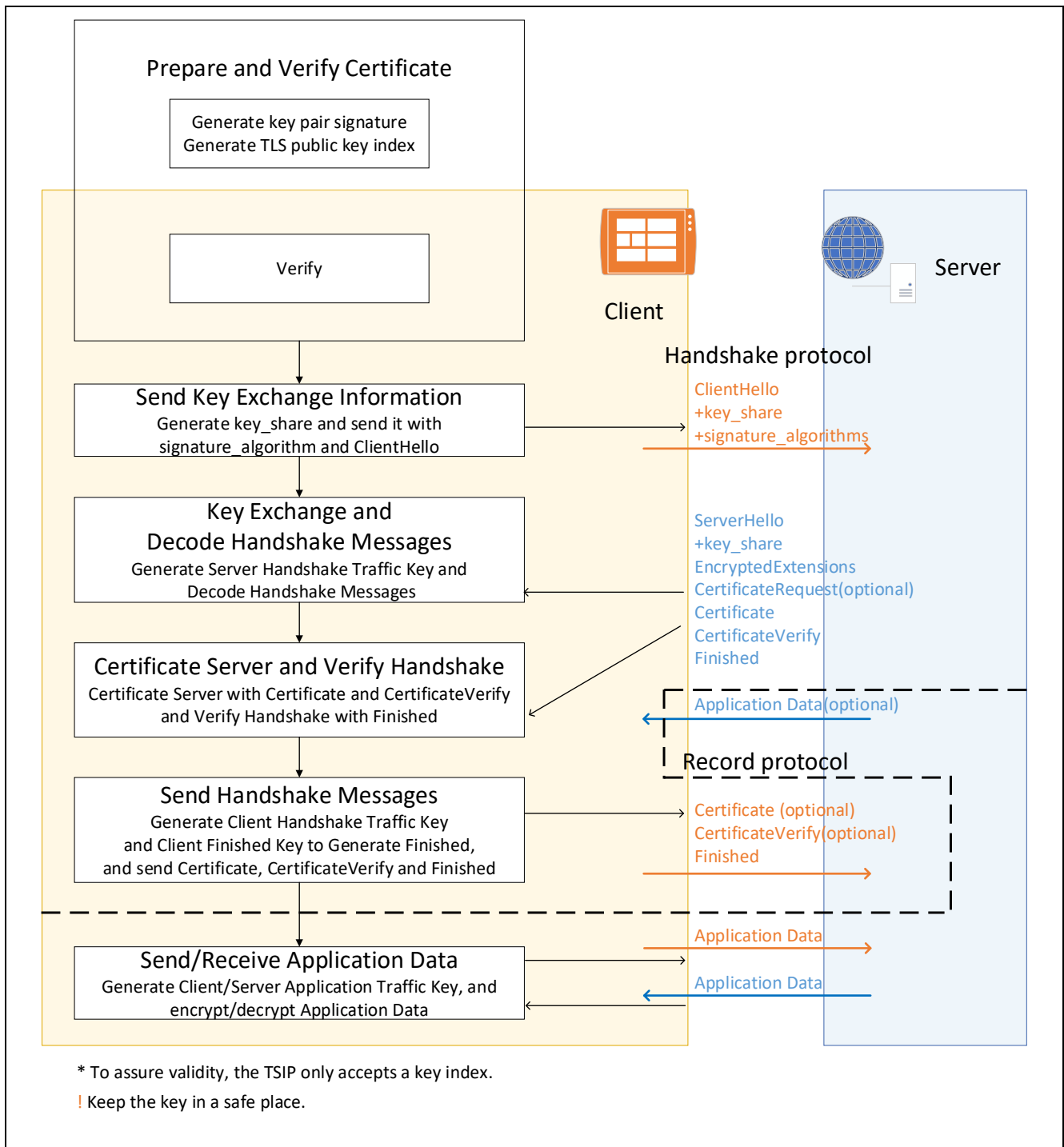


Figure 3.21 Full Handshake Implementation Using TLS 1.3 Client Function

1. Preparing and Verifying Certificates

1.1 Before using TLS cooperation function, prepare the items below.

- root CA certificate which the server connects to
- RSA 2048bit key pair
- client certificate.

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

public key of the key pair with using UFPK. Note that the above procedure must be executed in a safe site.

When there are multiple servers connects to and needed to register multiple root CA certificate, generate signature value of the root CA certificate bundle. It is possible to prevent making connection with an unintentional server with generating signature of the root CA certificate.

- 1.3 Use `R_TSIP_TlsRsaPublicKeyIndex` with the Encrypted Key of the RSA public key as input to generate an RSA public wrapped key. Please refer to 3.7.1, Key Injection and Updating for more detail.
- 1.4 Use `R_TSIP_TlsRegisterCaCertificationPublicKey()` or `R_TSIP_Open()` to register the RSA wrapped key to the TSIP driver. When `R_TSIP_Open()` is used to register the wrapped key, call the API after using `R_TSIP_Close`.
- 1.5 Use `R_TSIP_TlsRootCertificateVerification()` to verify the root CA certificate bundle and generate an encrypted root CA certificate public key.
- 1.6 Use `R_TSIP_TlsCertificateVerification()` or `R_TSIP_TlsCertificateVerificationExtension()` with the encrypted root CA certificate public key and client certificate public key as input to generate an encrypted client certificate public key.

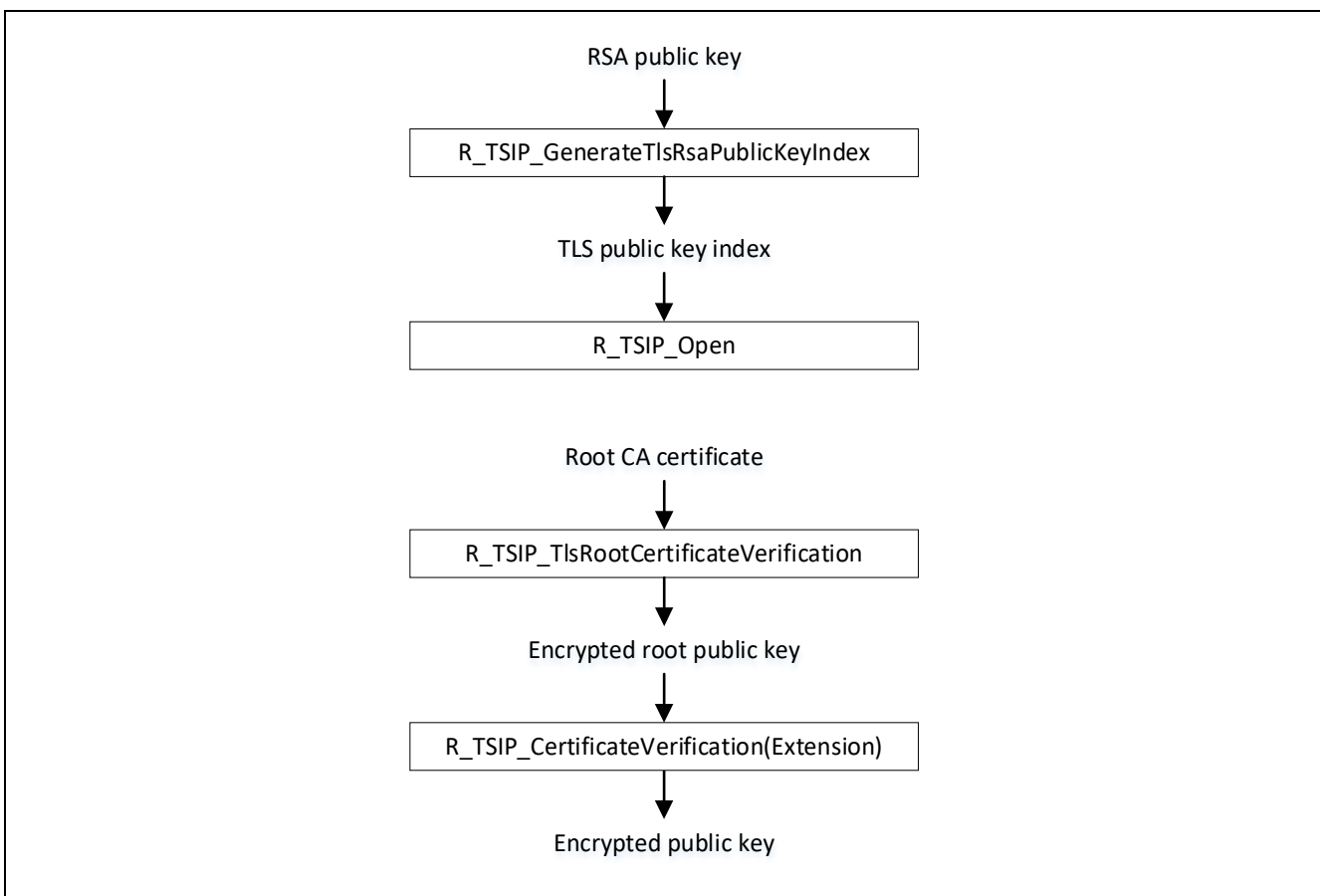


Figure 3.22 Preparing and Verifying Certificates

2. Sending Key Exchange Information

- 2.1 Use `R_TSIP_GenerateTls13P256KeyIndex()` to generate an ephemeral ECDH public key.
- 2.2 When ClientHello is sent, ECDH public key is sent to the server as the `key_share` field, together with the `signature_algorithm` field.

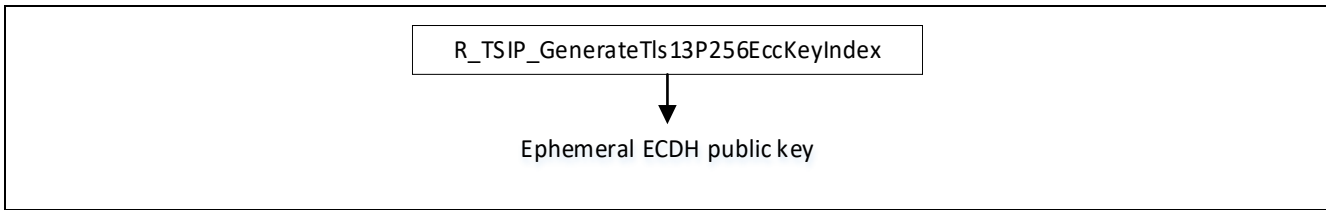


Figure 3.23 Sending Key Exchange Information

3. Exchanging Keys and Decoding Handshake Messages

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

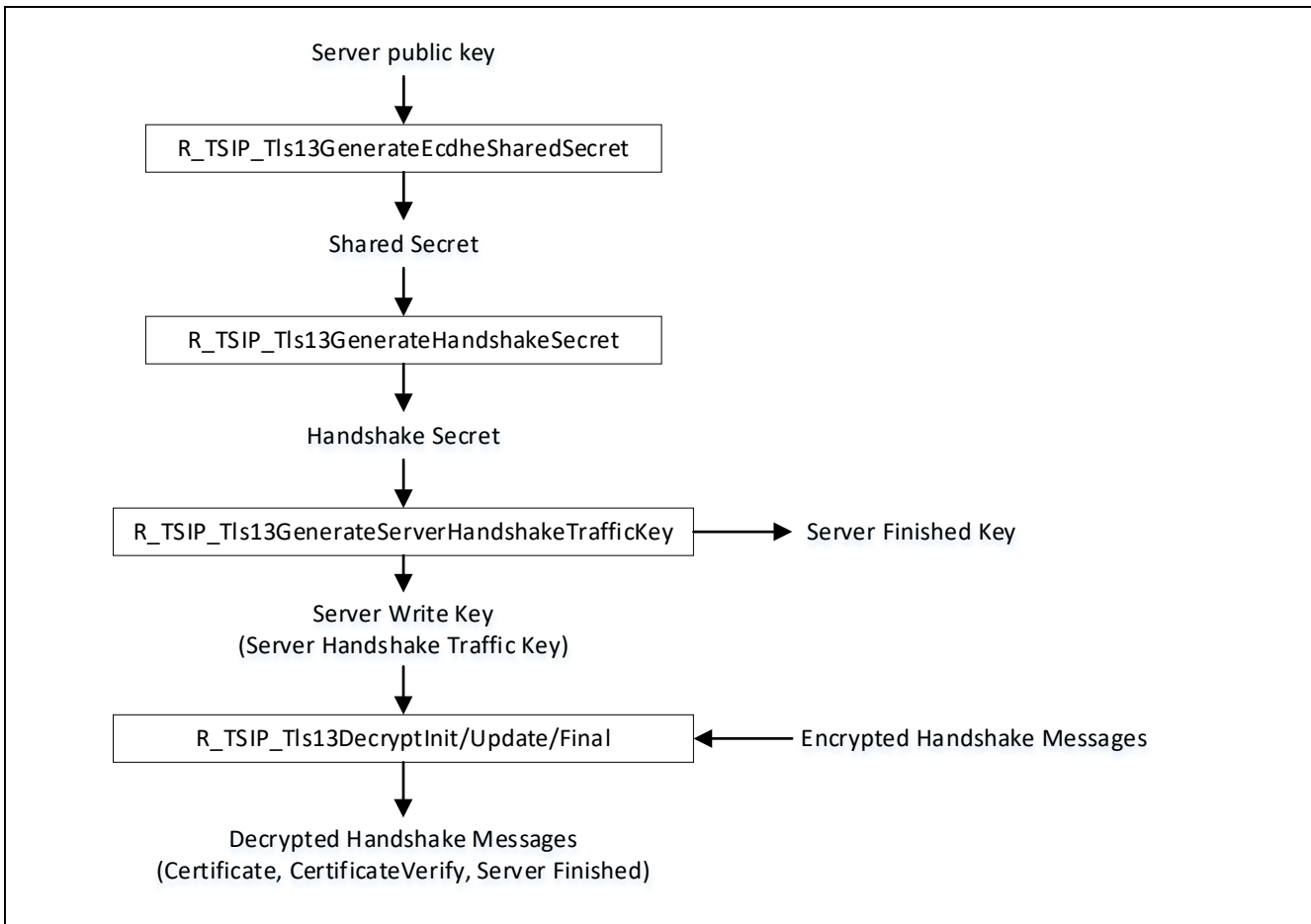
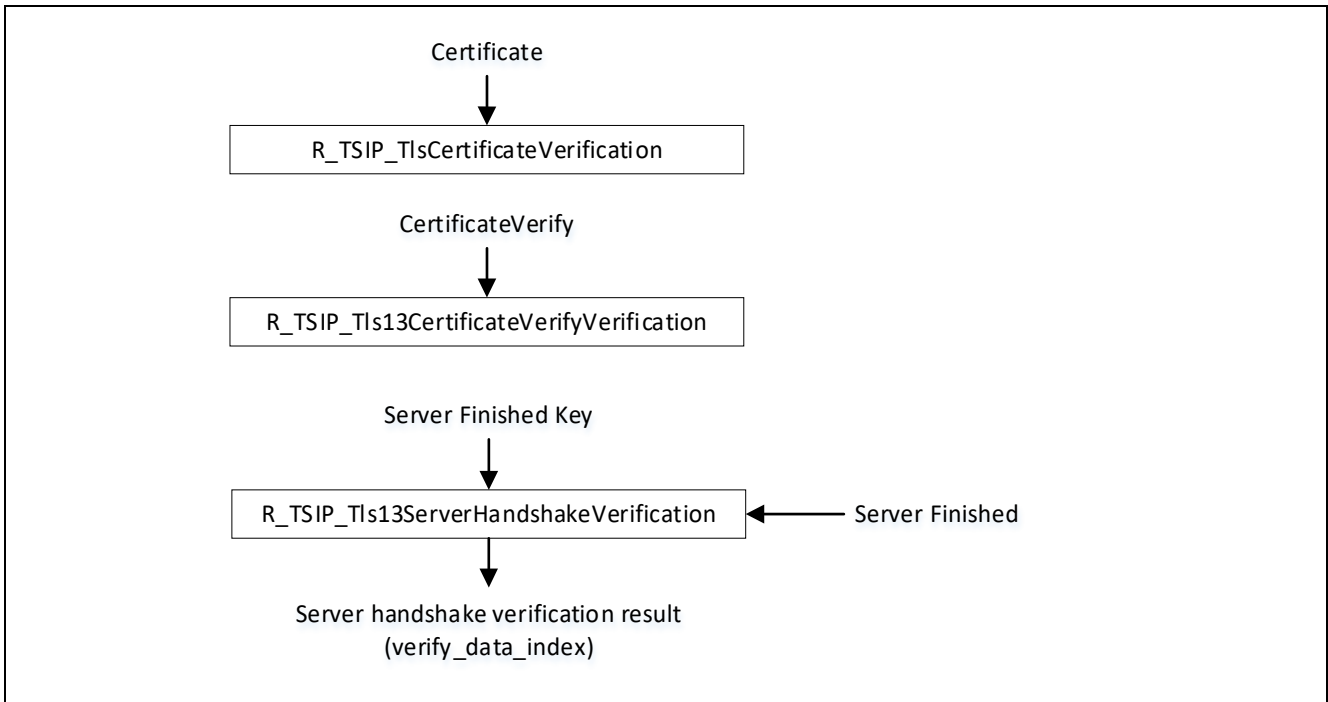


Figure 3.24 Exchanging Keys and Decoding Handshake Messages

4. Verifying Server and Handshake

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

**Figure 3.25 Verifying Server and Handshake**

5. Sending Handshake Messages

- 5.1 The (Client) Certificate field and (Client) CertificateVerify field are generated when CertificateRequest is received from the server. To generate a signature for the (Client) CertificateVerify field, use R_TSIP_Tls13CertificateVerifyGenerate() with the secret wrapped key of the certificate as input.
- 5.2 Use R_TSIP_Tls13GenerateClientHandshakeTrafficKey() with the wrapped key of the handshake secret as input to generate the client write wrapped key and client finished key index used in the handshake phase.
- 5.3 Use R_TSIP_Sha256HmacGenerateInit/Update/Final() with the client finished wrapped key as input to create the (Client) Finished field.
- 5.4 Use R_TSIP_Tls13EncryptInit/Update/Final() with the client write wrapped key as input to encrypt the handshake messages sent to the server.
- 5.5 Send the encrypted handshake messages.

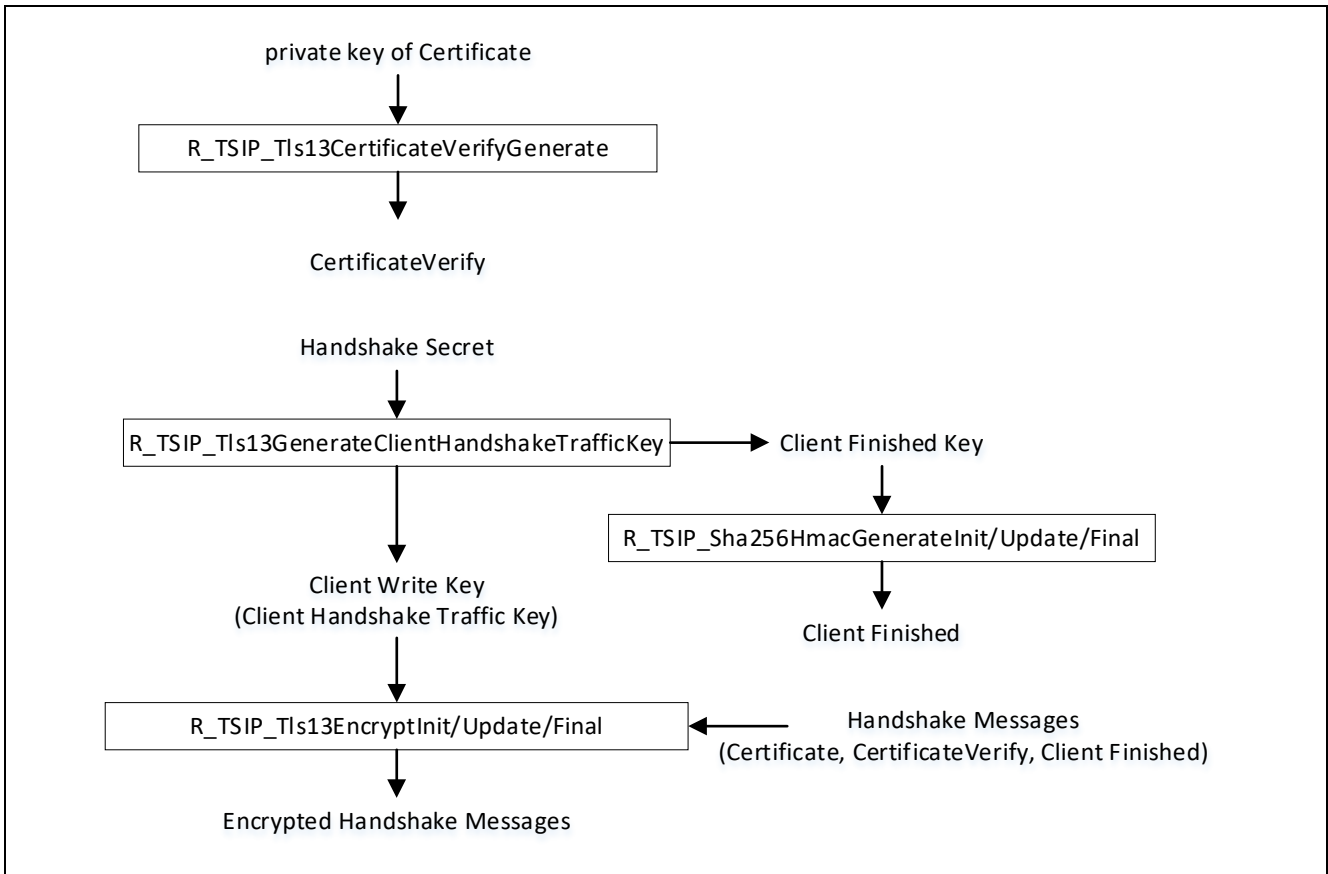


Figure 3.26 Sending Handshake Messages

6. Sending/Receiving Application Data

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

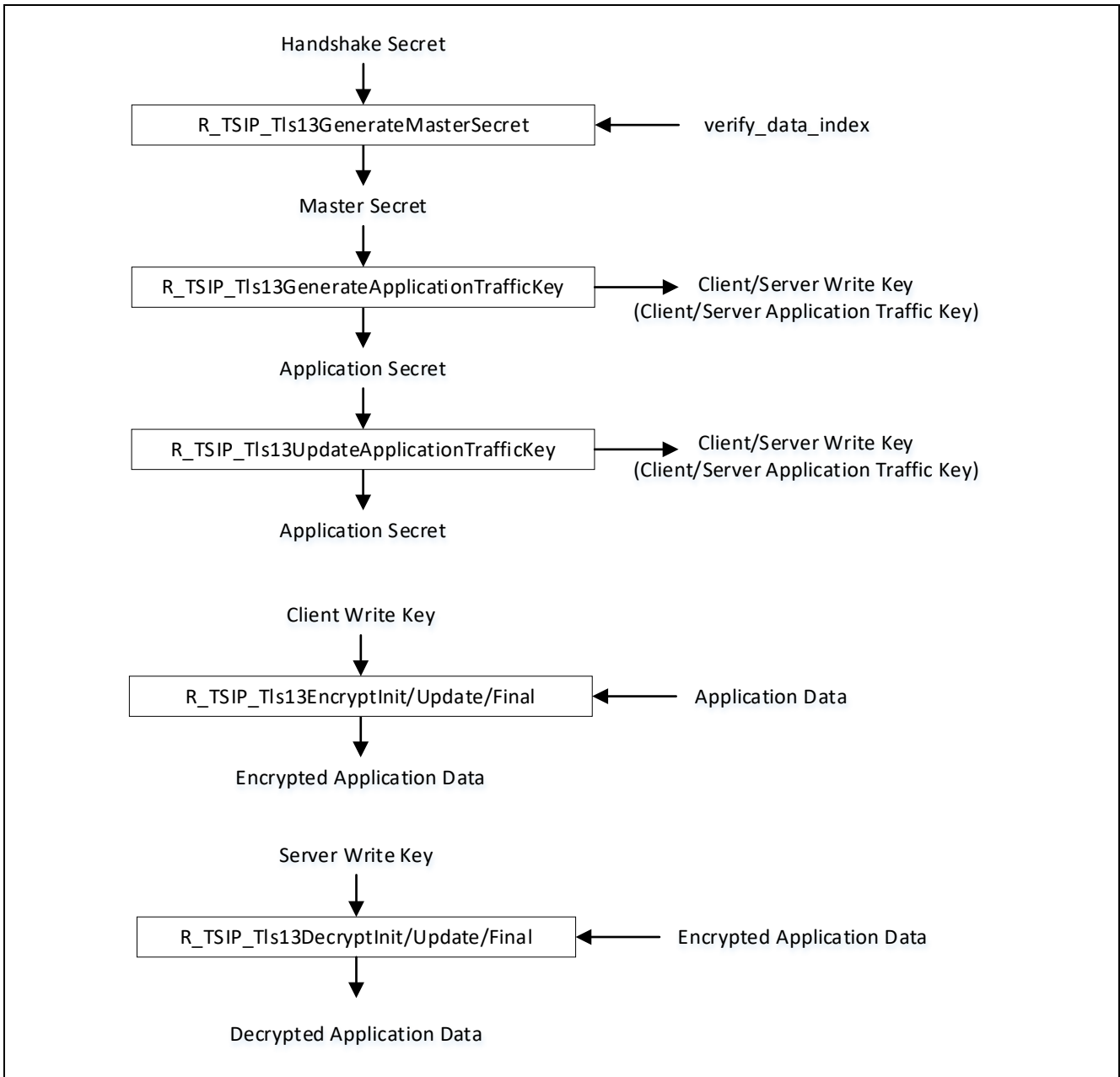


Figure 3.27 Sending/Receiving Application Data

3.13.1.2 Resumption

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

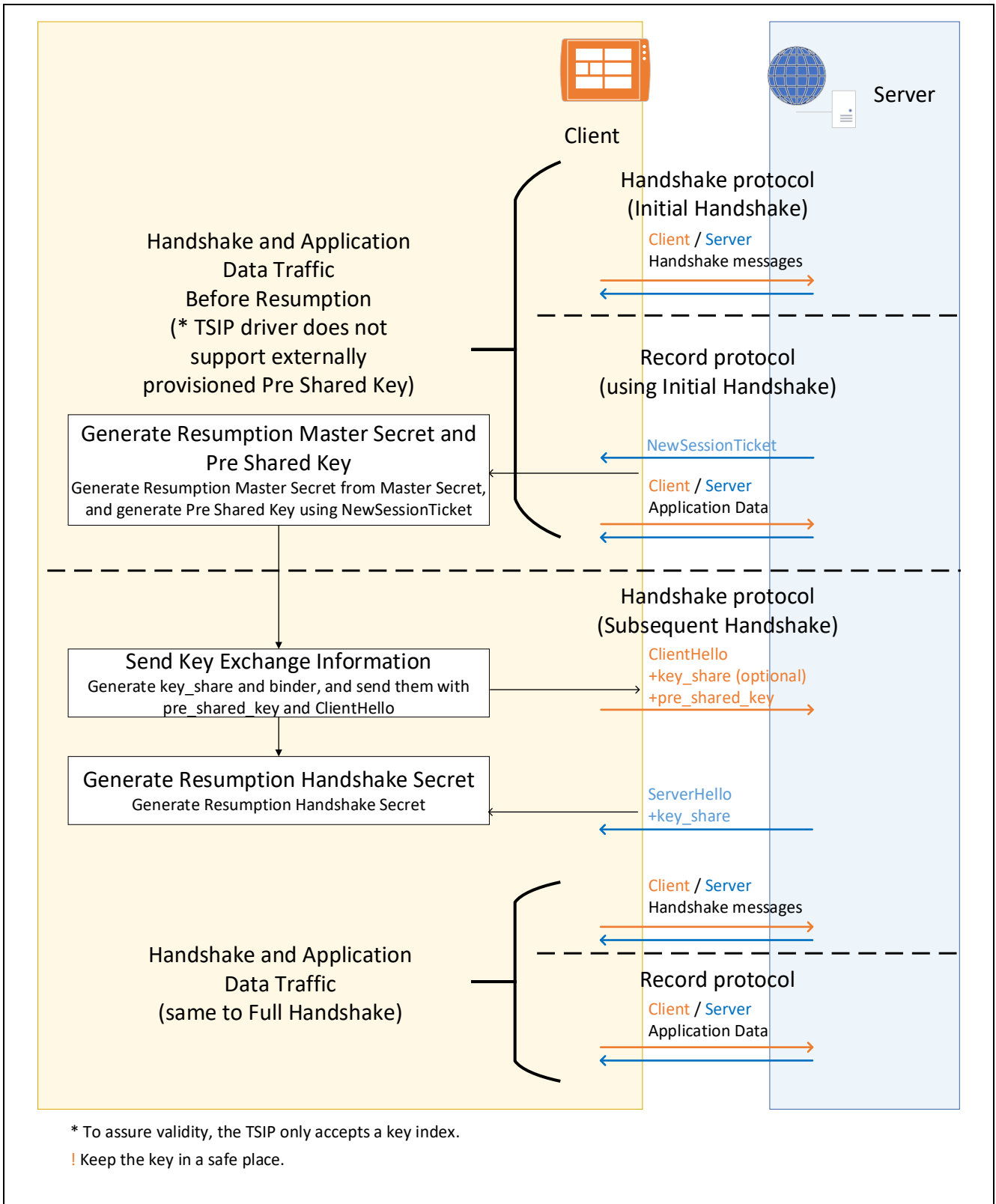


Figure 3.28 Resumption Implementation Using TLS 1.3 Client Function

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

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

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

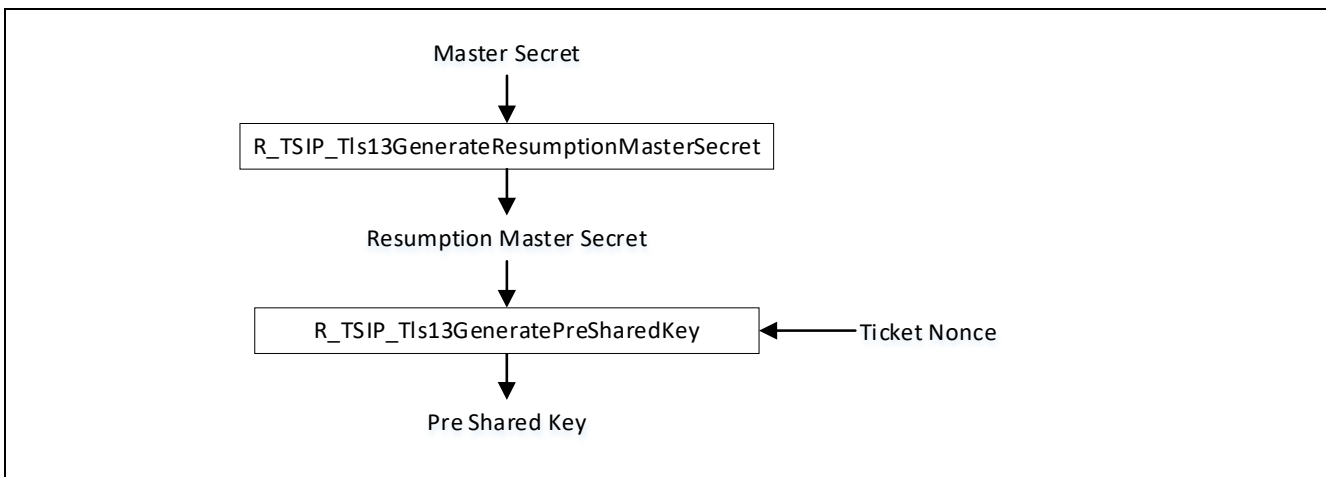


Figure 3.29 Generating Resumption Master Secret and Pre Shared Key

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

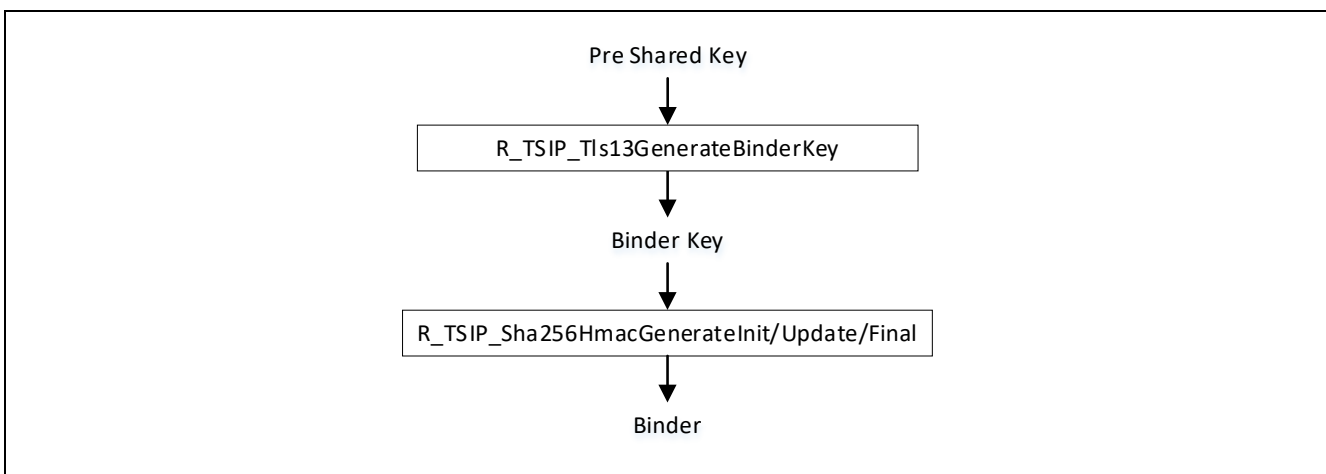


Figure 3.30 Sending Key Exchange Data

4. Generating Resumption Handshake Secret

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

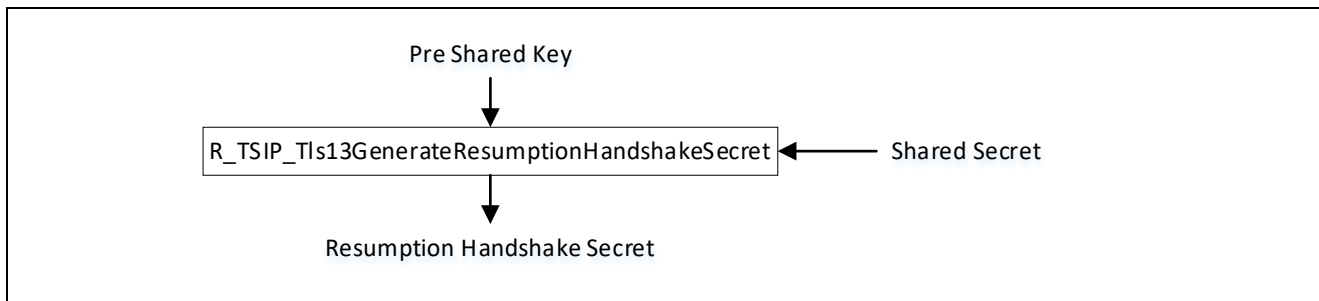


Figure 3.31 Generating Resumption Handshake Secret

5. Handshake and Application Data Communication

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

3.13.1.3 0-RTT

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

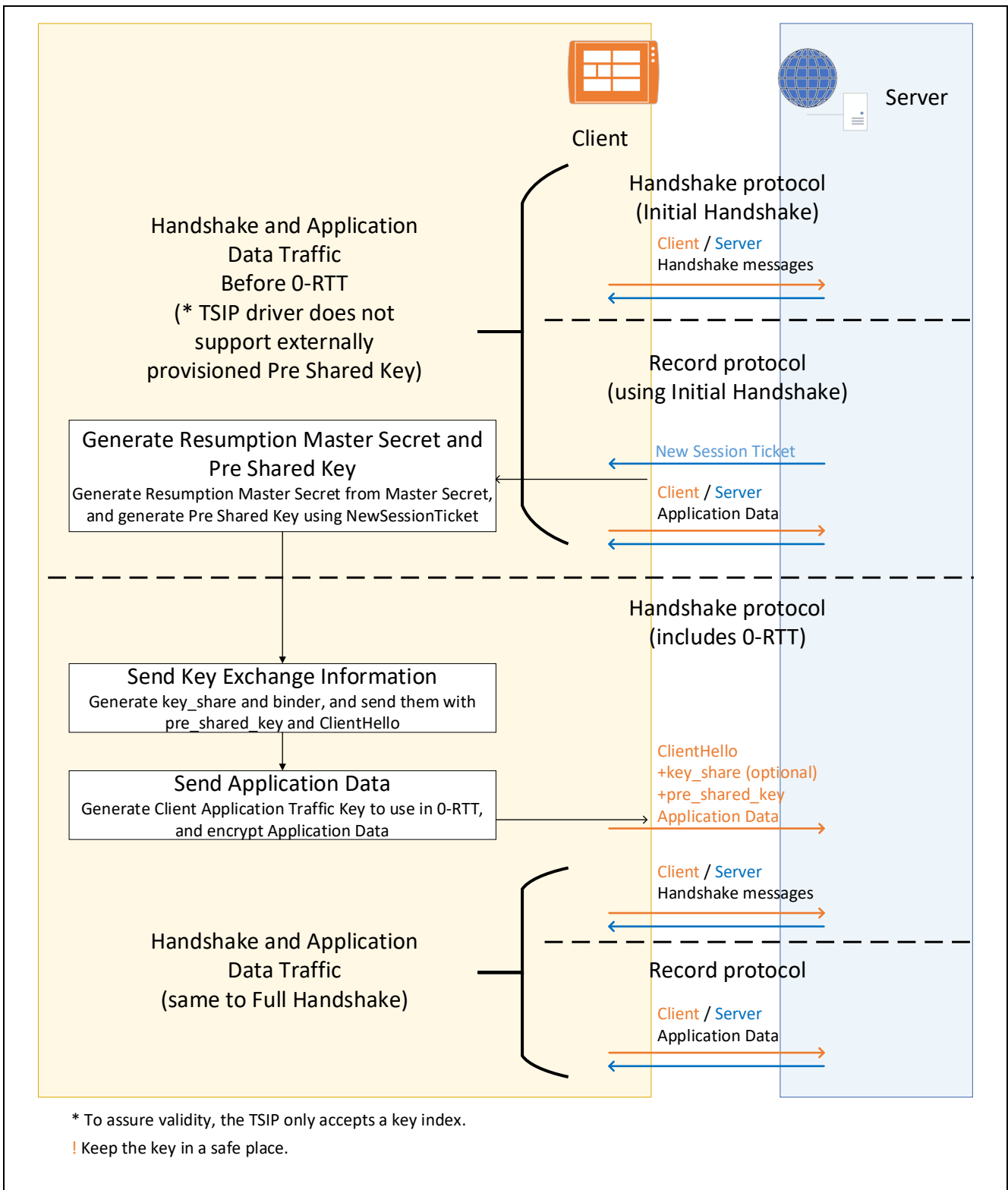


Figure 3.32 0-RTT Implementation Using TLS 1.3 Client Function

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

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

2. Generating Resumption Master Secret and Pre Shared Key
 - 2.1 The subsequent processing is the same as item 2. of 3.13.1.2: Use `R_TSIP_Tls13GenerateResumptionMasterSecret()` with the master secret wrapped key used for the full handshake as input to generate the resumption master secret wrapped key.
 - 2.2 As in item 2.2 of 3.13.1.2, use `R_TSIP_Tls13GeneratePreSharedKey()` with the resumption master secret wrapped key and the ticket nonce within the new session ticket received from the server as input to generate the pre shared wrapped key.
3. Sending Key Exchange Data
 - 3.1 The subsequent processing is the same as item 3. of 3.13.1.2: Use `R_TSIP_Tls13GeneratePskBinderKey()` with the pre shared wrapped key as input to generate the binder wrapped key.
 - 3.2 Use `R_TSIP_Sha256HmacGenerateInit/Update/Finish()` with the binder wrapped key as input to generate the binder.
 - 3.3 As in item 2.1 of 3.13.1.1, use `R_TSIP_GenerateTls13P256KeyIndex()` to generate an ephemeral ECDH public key.
 - 3.4 When sending `ClientHello` to the server, also attach the pre shared key information including the binder as the `pre_shared_key` field and the ECDH public key as the `key_share` field.
4. Sending Application Data
 - 4.1 Use `R_TSIP_Tls13Generate0RttApplicationWriteKey()` with the pre shared wrapped key as input to generate the client write wrapped key to be used for 0-RTT.
 - 4.2 Use `R_TSIP_Tls13EncryptInit/Update/Final()` with the client write wrapped key as input to encrypt the application data.
 - 4.3 After sending `ClientHello`, send the encrypted application data to the server.

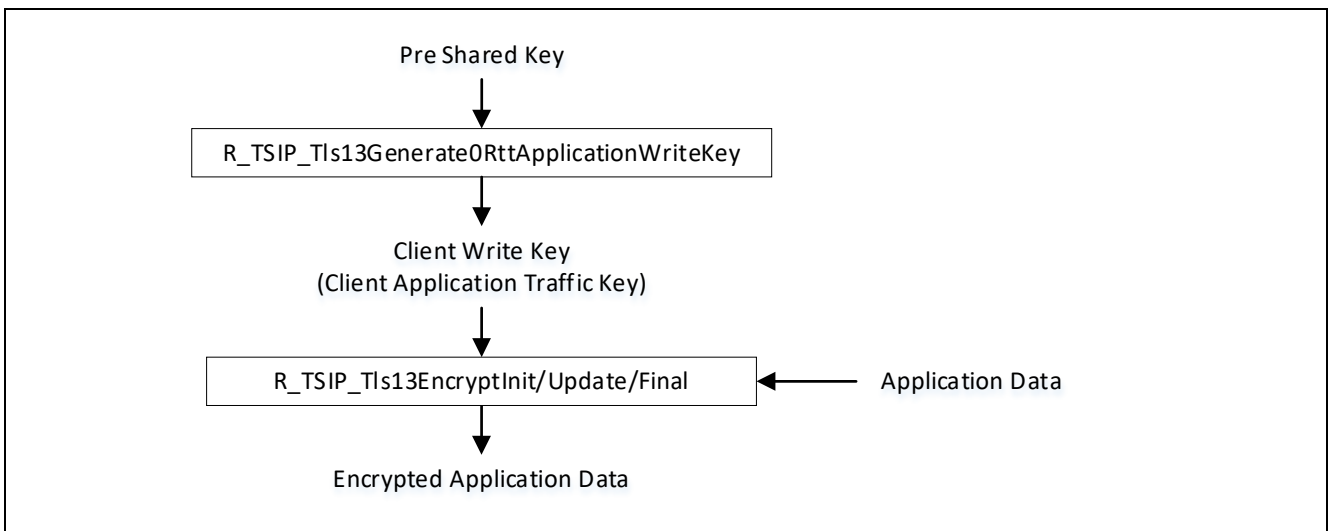


Figure 3.33 Sending Application Data

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

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

3.13.2 TLS 1.3 Server Function

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

3.13.2.1 Full Handshake

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

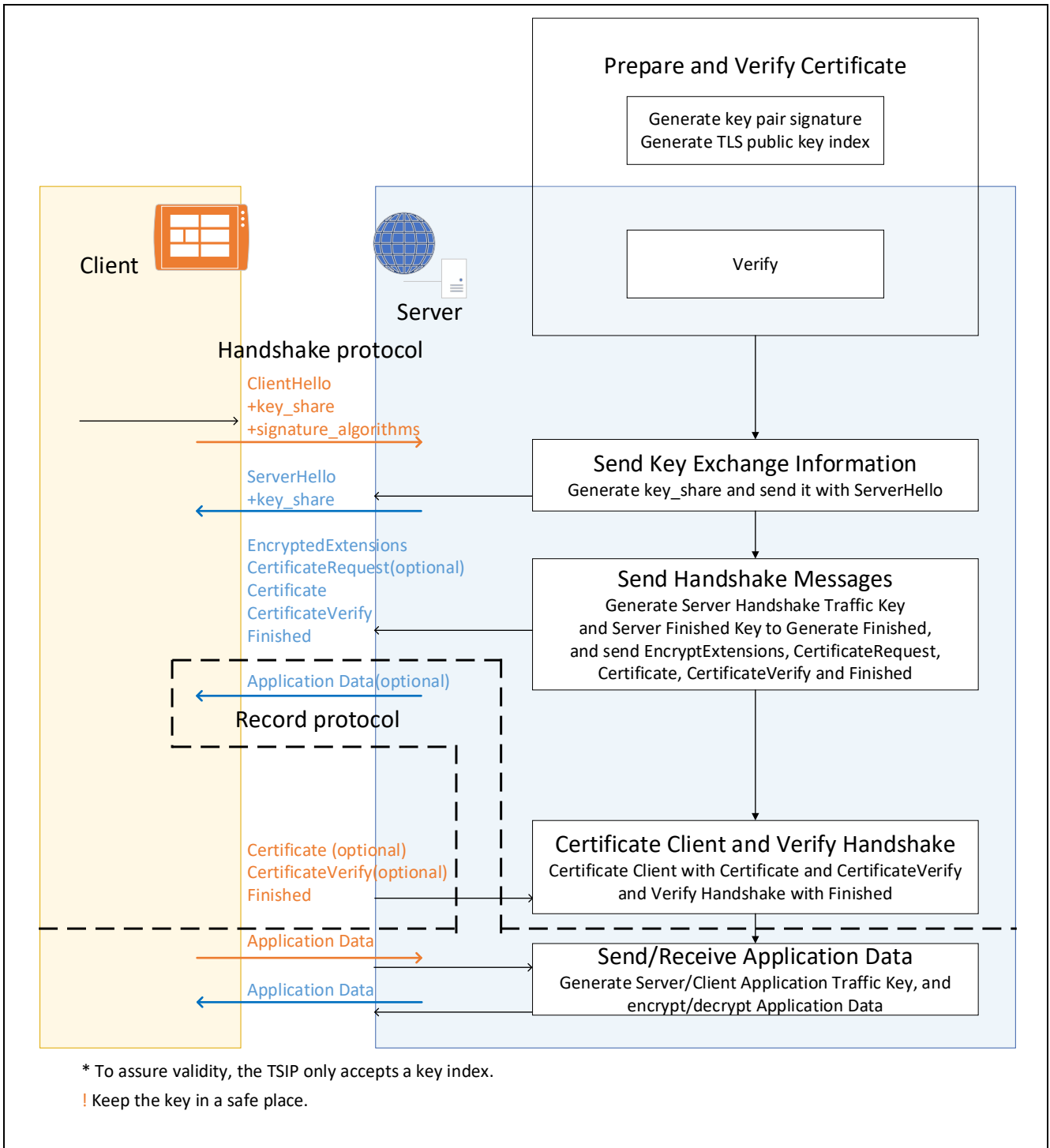


Figure 3.34 Full Handshake Implementation Using TLS 1.3 Server Function

1. Preparing and Verifying Certificates

1.1 Before using TLS cooperation function, prepare the items below.

- root CA certificate which the server connects to
- RSA 2048bit key pair
- server certificate.

1.2 Generate signature value of the prepared root CA certificate with using the private key of the key pair. The signature algorithm is RSA2048 PSS with SHA256. And generate Encrypted Key of the public key of the key pair with using UFPK. Note that the above procedure must be executed in a safe site.

When multiple root CA certificate are needed to register, generate signature value of the root CA certificate bundle. It is possible to prevent making connection with using an unauthorized root CA certificate.

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

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

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

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

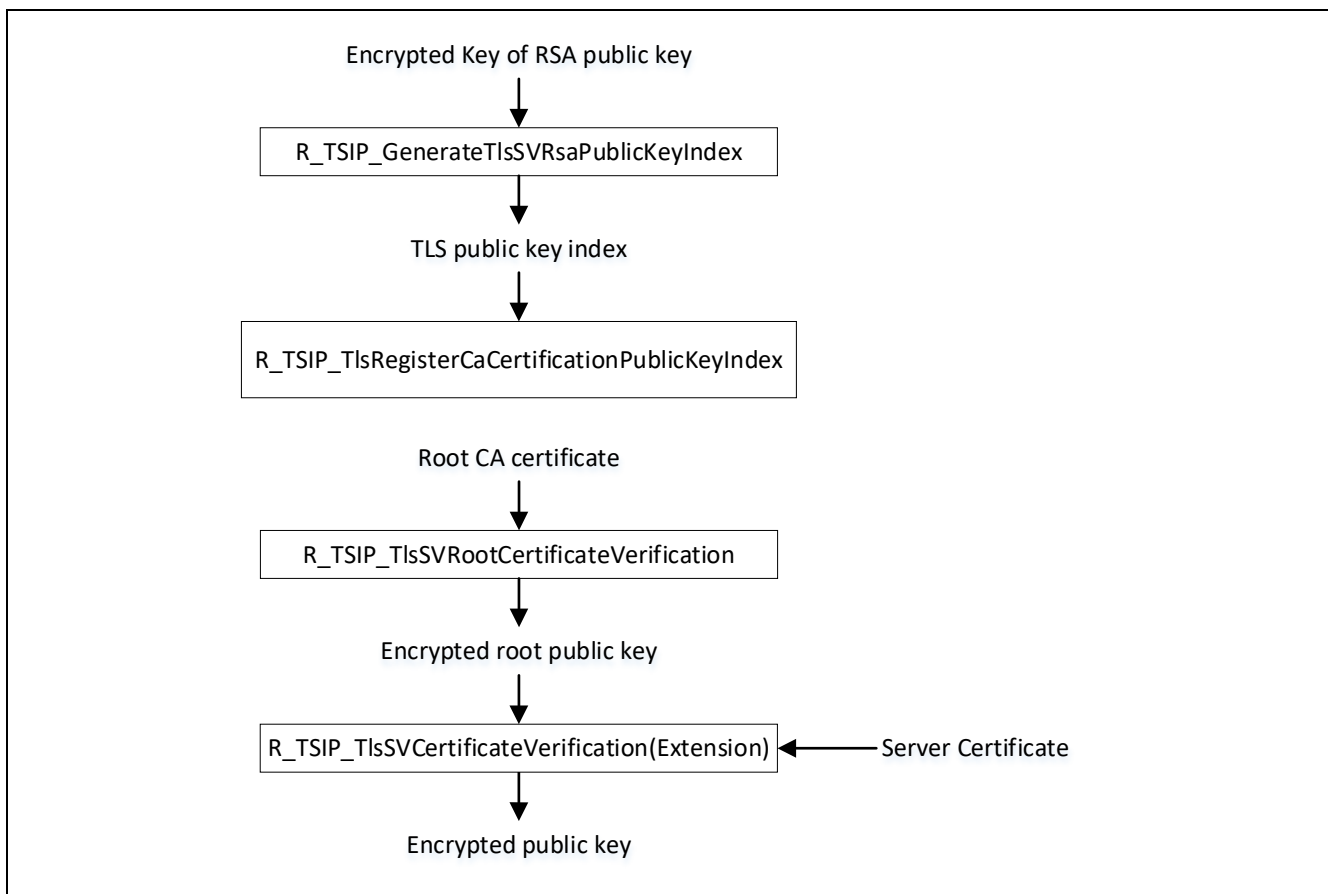


Figure 3.35 Preparing and Verifying Certificates

2. Sending Key Exchange Information

- 2.1 Use `R_TSIP_GenerateTls13P256KeyIndex()` to generate an ephemeral ECDH public key.
- 2.2 When `ServerHello` is sent, send the ECDH public key to the server as the `key_share` field.

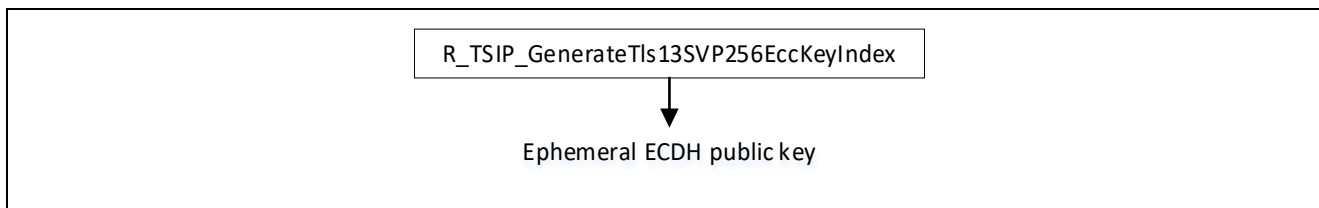


Figure 3.36 Sending Key Exchange Information

3. Sending Handshake Messages

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

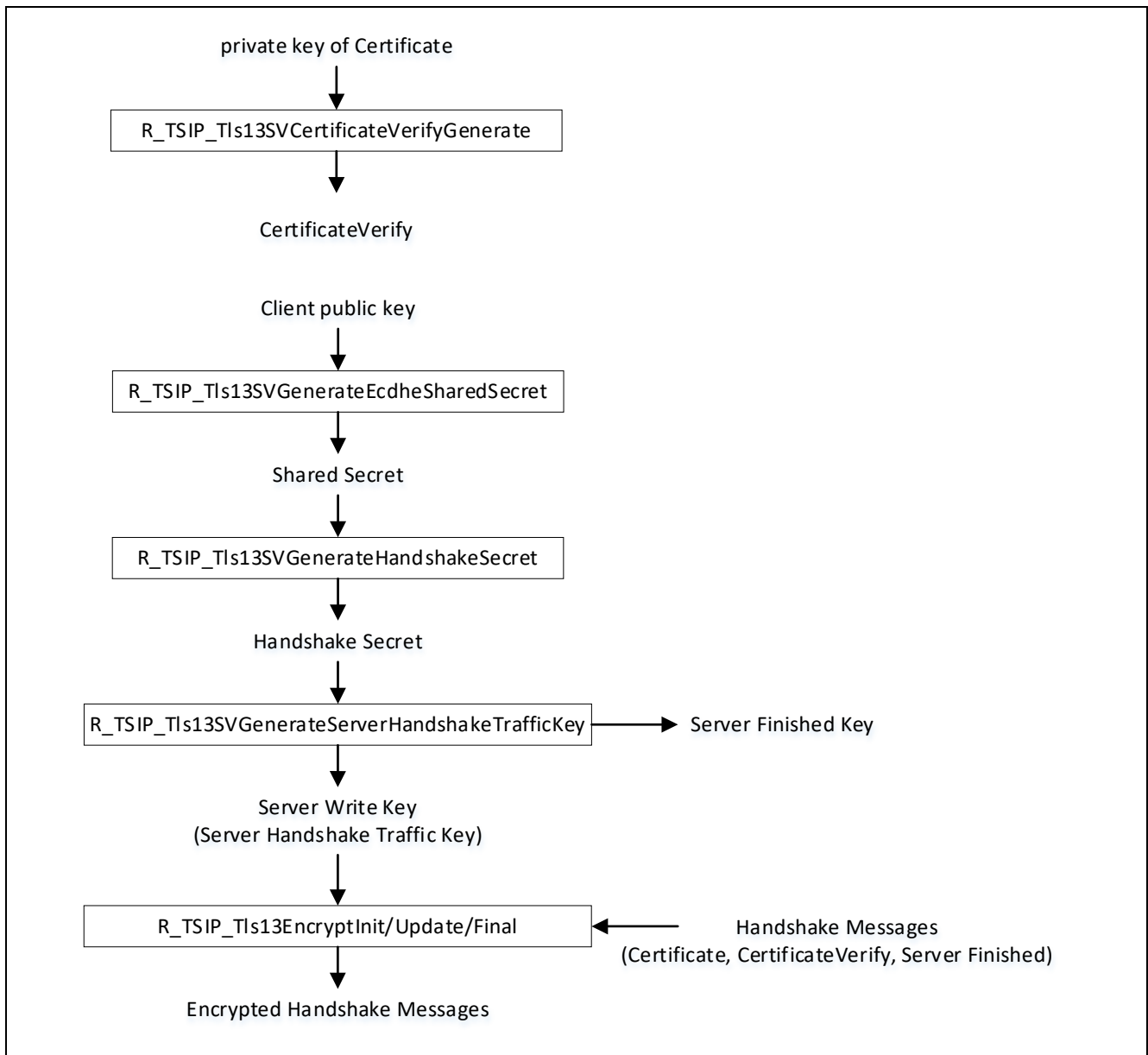


Figure 3.37 Sending Handshake Messages

4. Verifying Client and Handshake

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

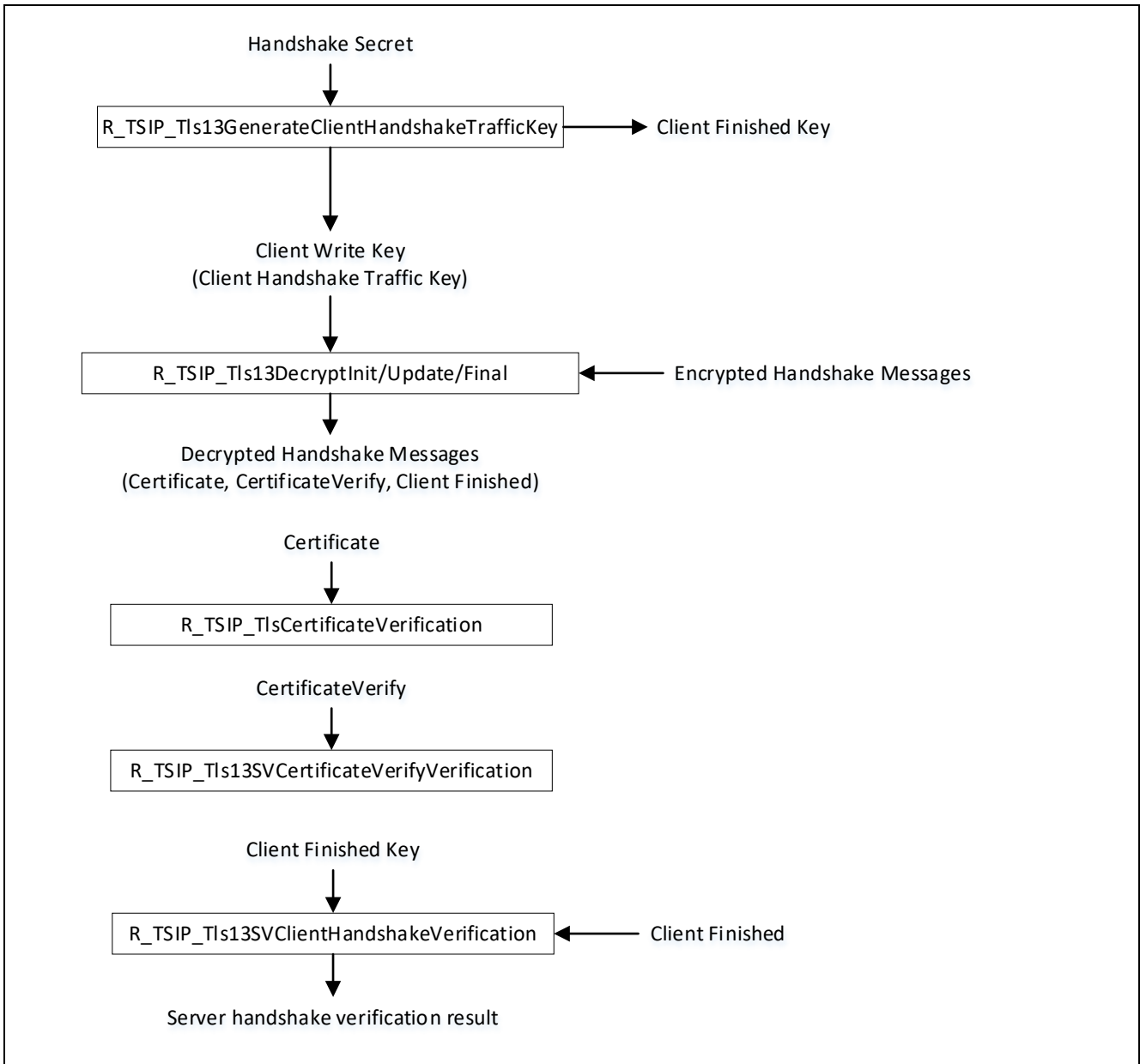


Figure 3.38 Verifying Client and Handshake

5. Sending/Receiving Application Data

- 5.1 Use `R_TSIP_TIs13SVGenerateMasterSecret()` with the handshake secret wrapped key as input to generate the master secret wrapped key.
- 5.2 Use `R_TSIP_TIs13SVGenerateApplicationTrafficKey()` with the master secret wrapped key as input to generate the server write wrapped key, client write wrapped key, and application secret wrapped key used by the record protocol.
- 5.3 To update the server write key or client write key, use `R_TSIP_TIs13SVUpdateApplicationTrafficKey()` with the application secret wrapped key as input.
- 5.4 To encrypt data to be sent to the server, use `R_TSIP_TIs13EncryptInit/Update/Final()` with the server write wrapped key as input.
- 5.5 To decrypt data received from the server, use `R_TSIP_TIs13DecryptInit/Update/Final()` with the client write wrapped key as input.

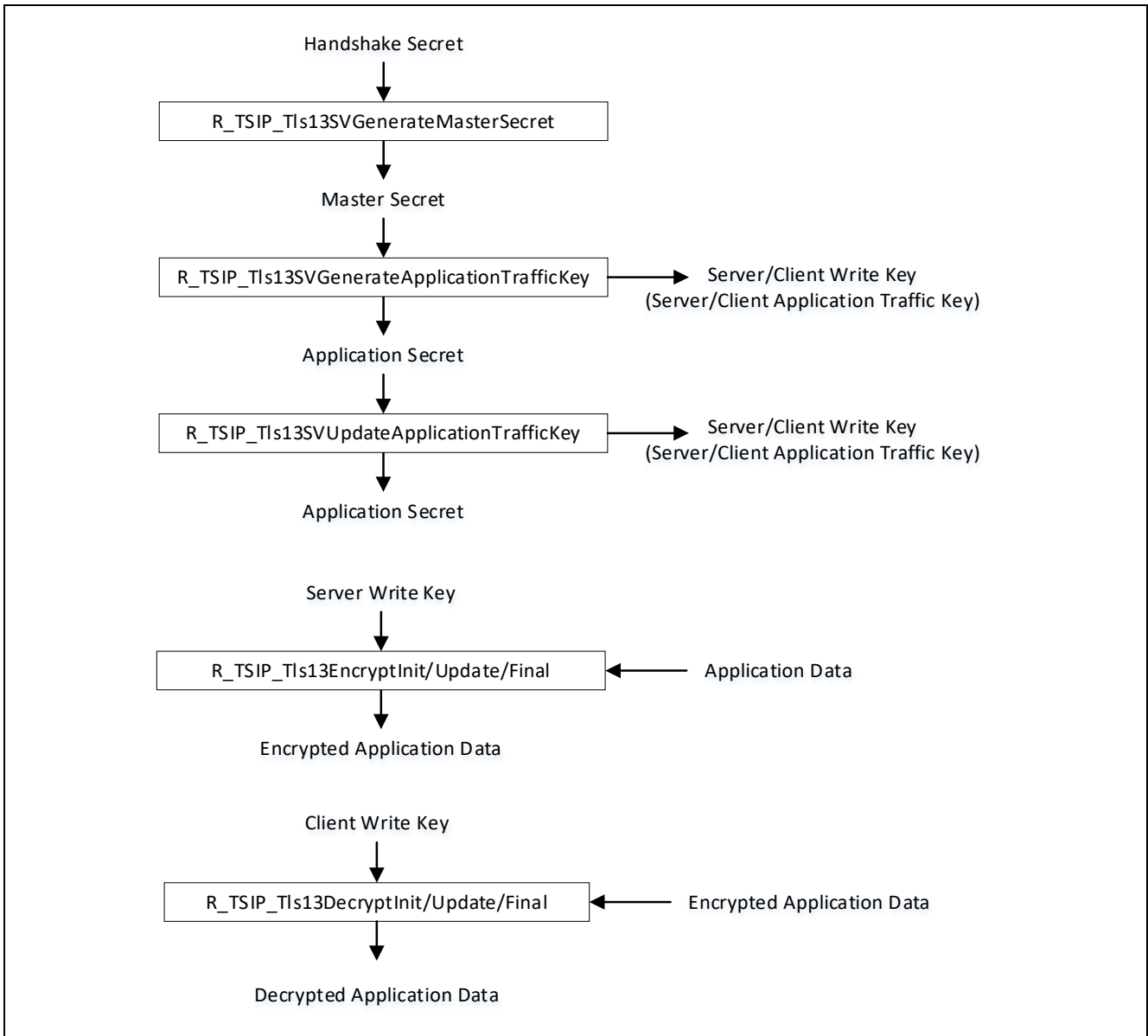


Figure 3.39 Sending/Receiving Application Data

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

3.13.2.2 Resumption

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

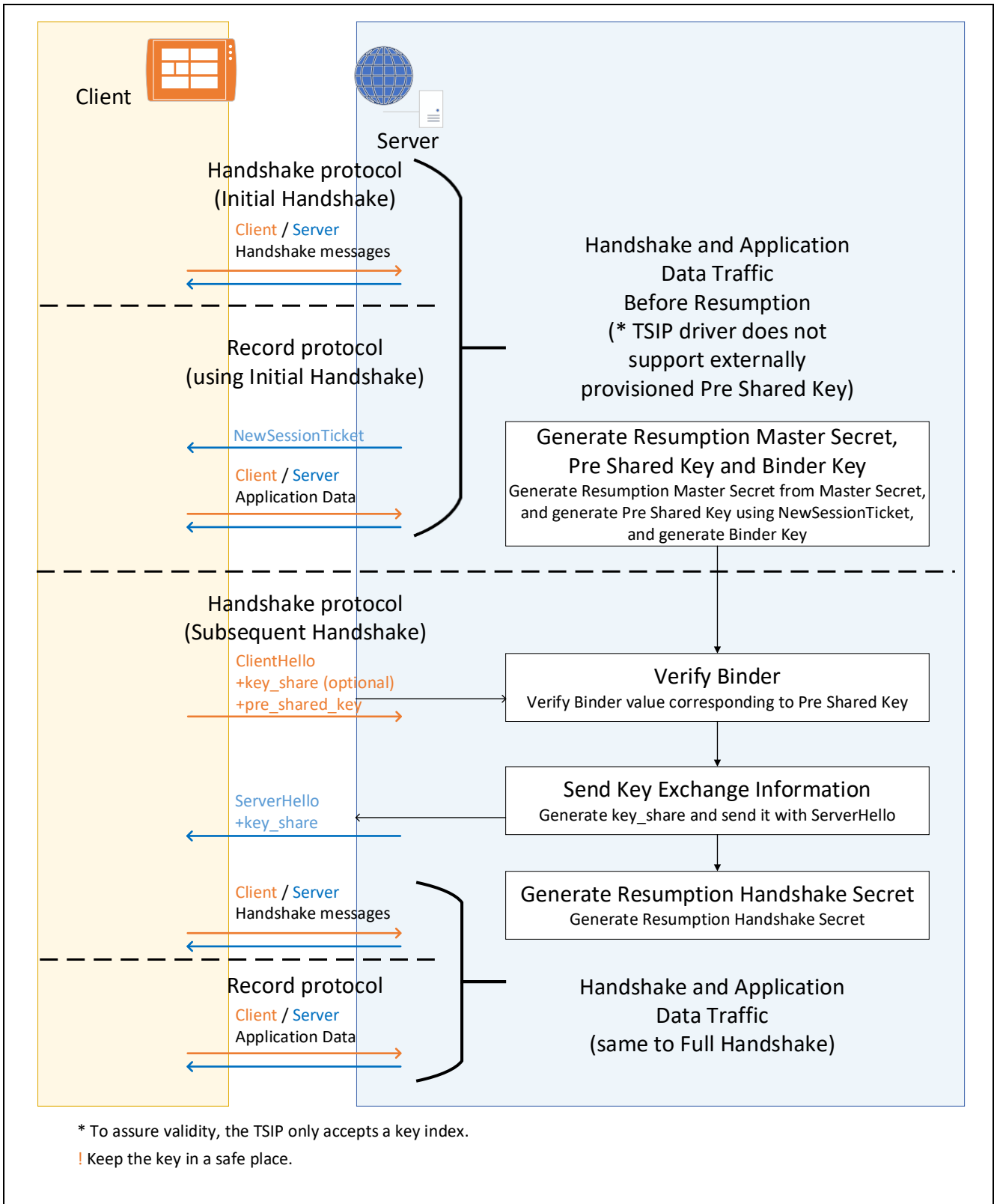


Figure 3.40 Resumption Implementation Using TLS 1.3 Server Function

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

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

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

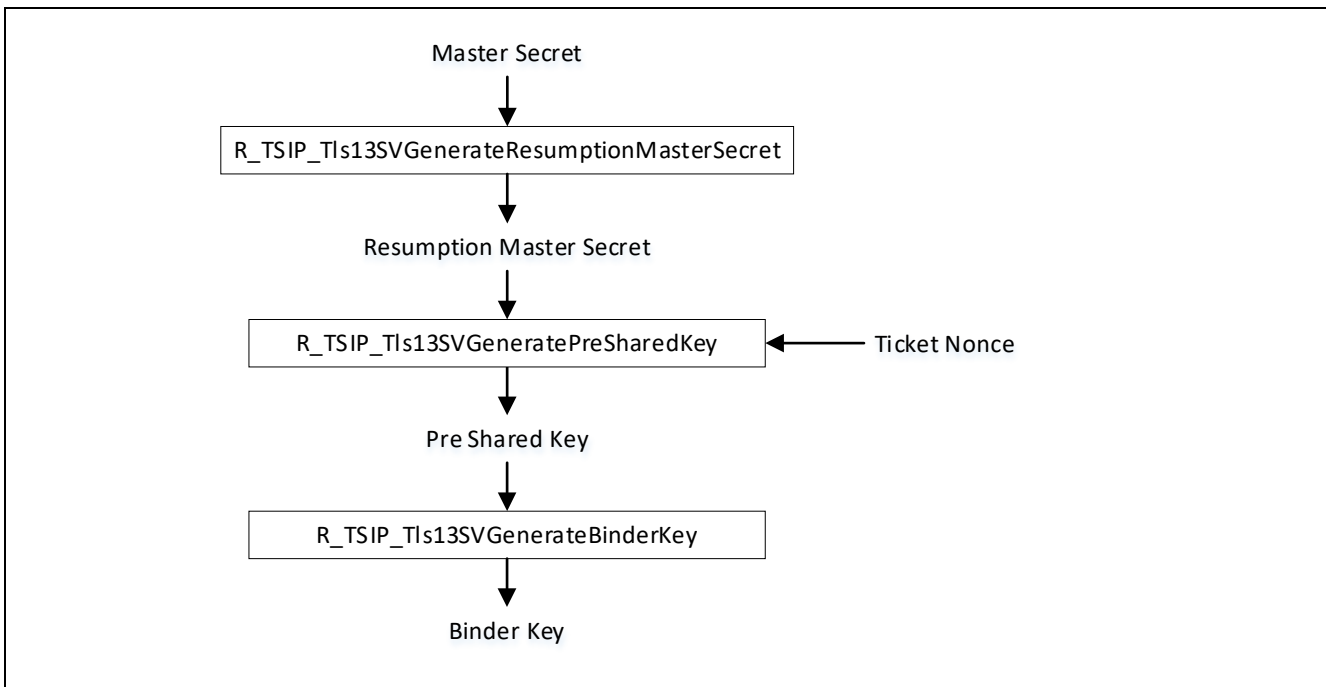


Figure 3.41 Generating Resumption Master Secret, Pre Shared Key, and Binder Key

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

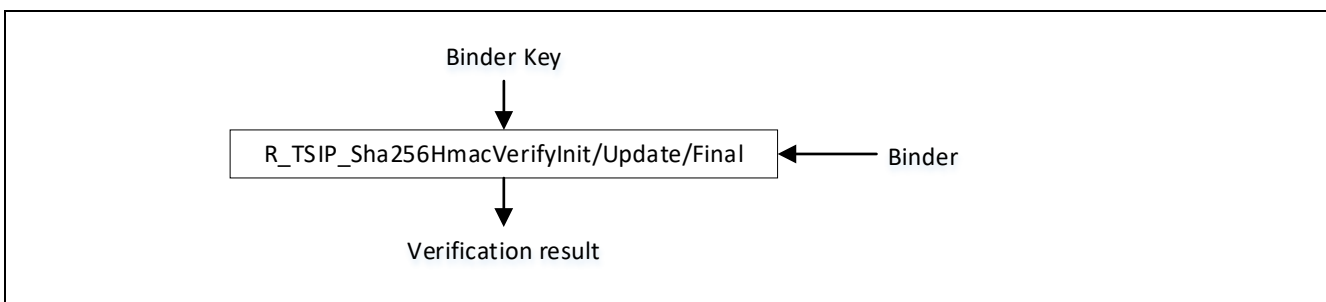


Figure 3.42 Verifying Binder

4. Generating Handshake Secret for Resumption

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

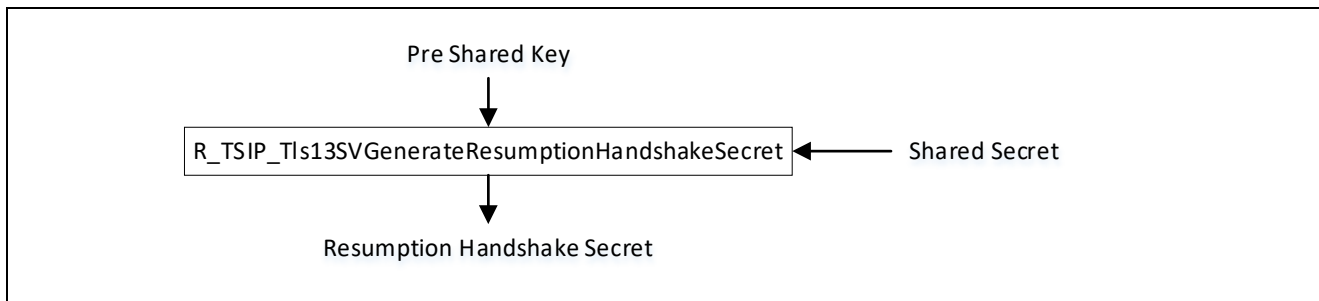


Figure 3.43 Generating Handshake Secret for Resumption

5. Handshake and Application Data Communication

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

3.13.2.3 0-RTT

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

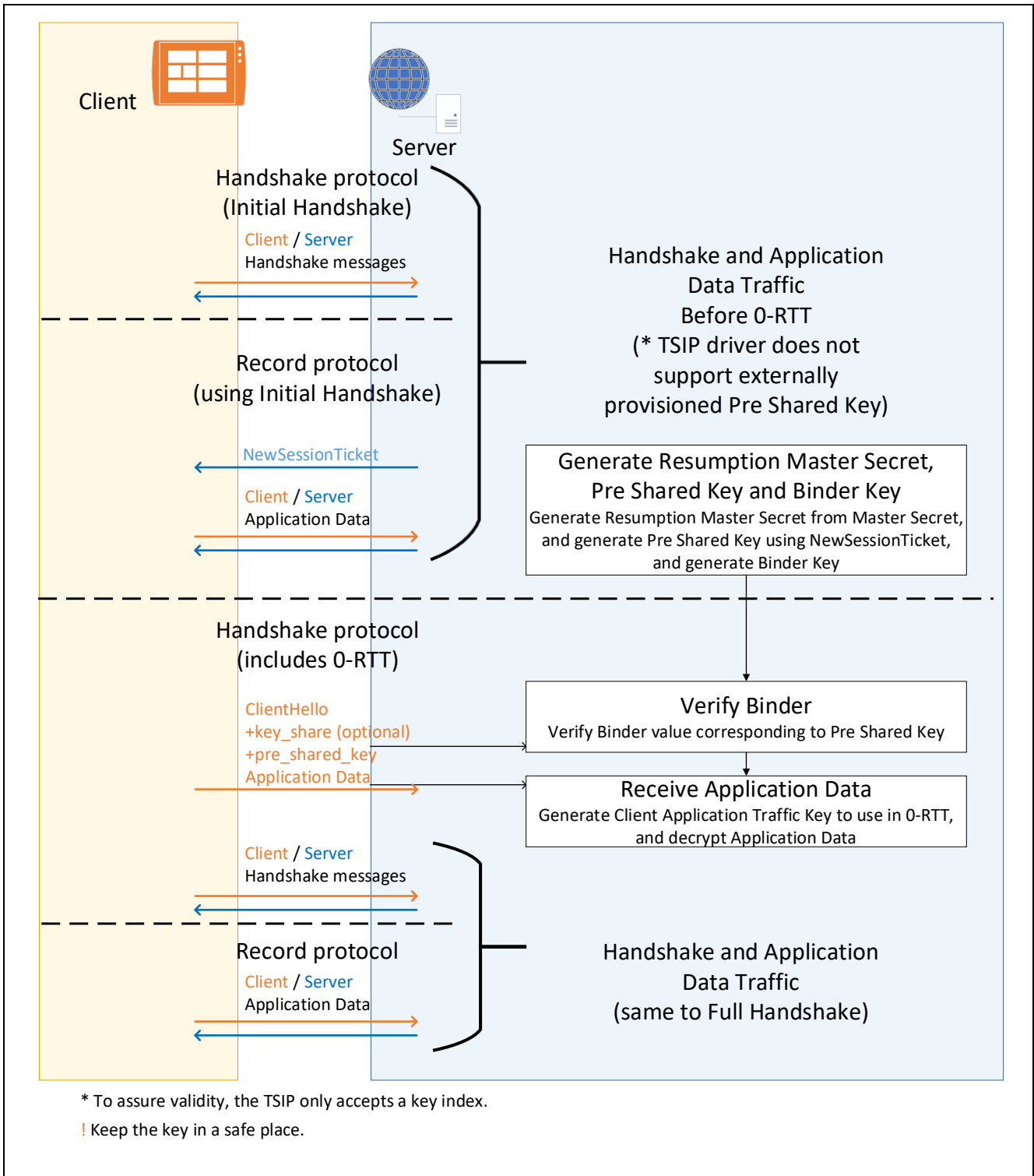


Figure 3.44 0-RTT Implementation Using TLS 1.3 Server Function

1. Handshake and Application Data Communication before Resumption

- 1.1 TLS 1.3 communication is established by performing a full handshake.

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

2. Generating Resumption Master Secret, Pre Shared Key, and Binder Key

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

3. Verifying Binder

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

4. Receiving Application Data

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

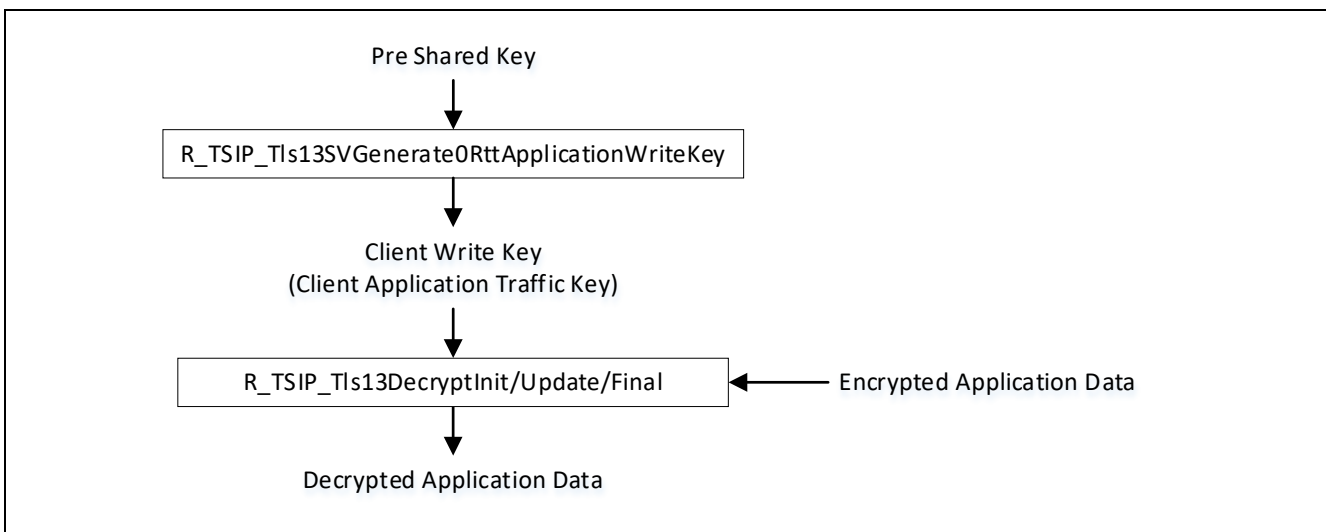


Figure 3.45 Receiving Application Data

5. Handshake and Application Data Communication

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

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

3.14 Firmware Update

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

The driver provides APIs for the following firmware update operations:

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

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

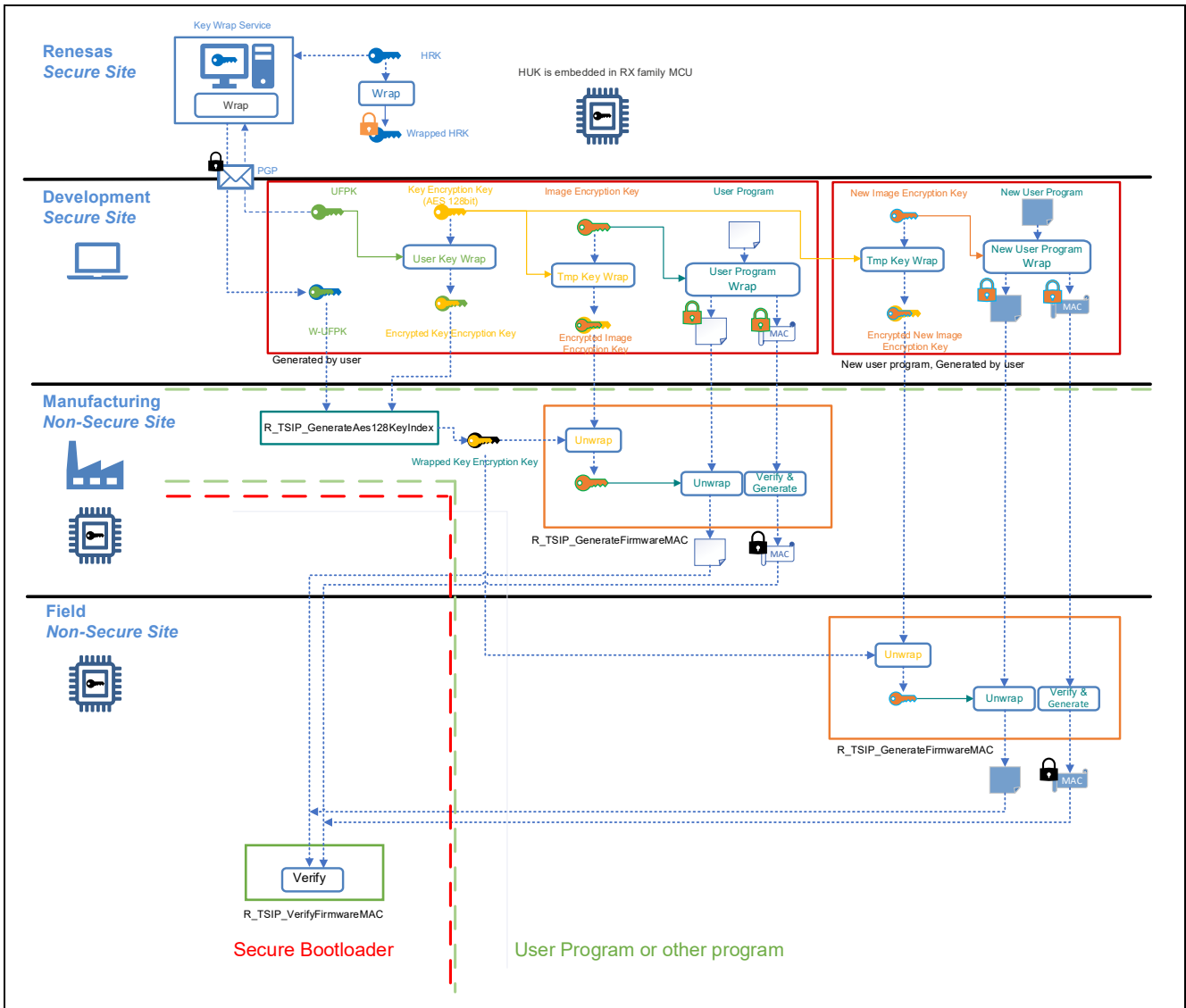


Figure 3.46 Firmware Update Sequence

3.14.1 Secure Boot

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

R_TSIP_VerifyFirmwareMAC can be used as a secure boot program. Run R_TSIP_VerifyFirmwareMAC with the MAC value output by R_TSIP_GenerateFirmwareMAC as input.

3.14.2 Firmware Update

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

R_TSIP_GenerateFirmwareMAC can be run after first calling R_TSIP_StartUpdateFirmware to put the TSIP into firmware update status.

3.14.3 Encrypting the User Program

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

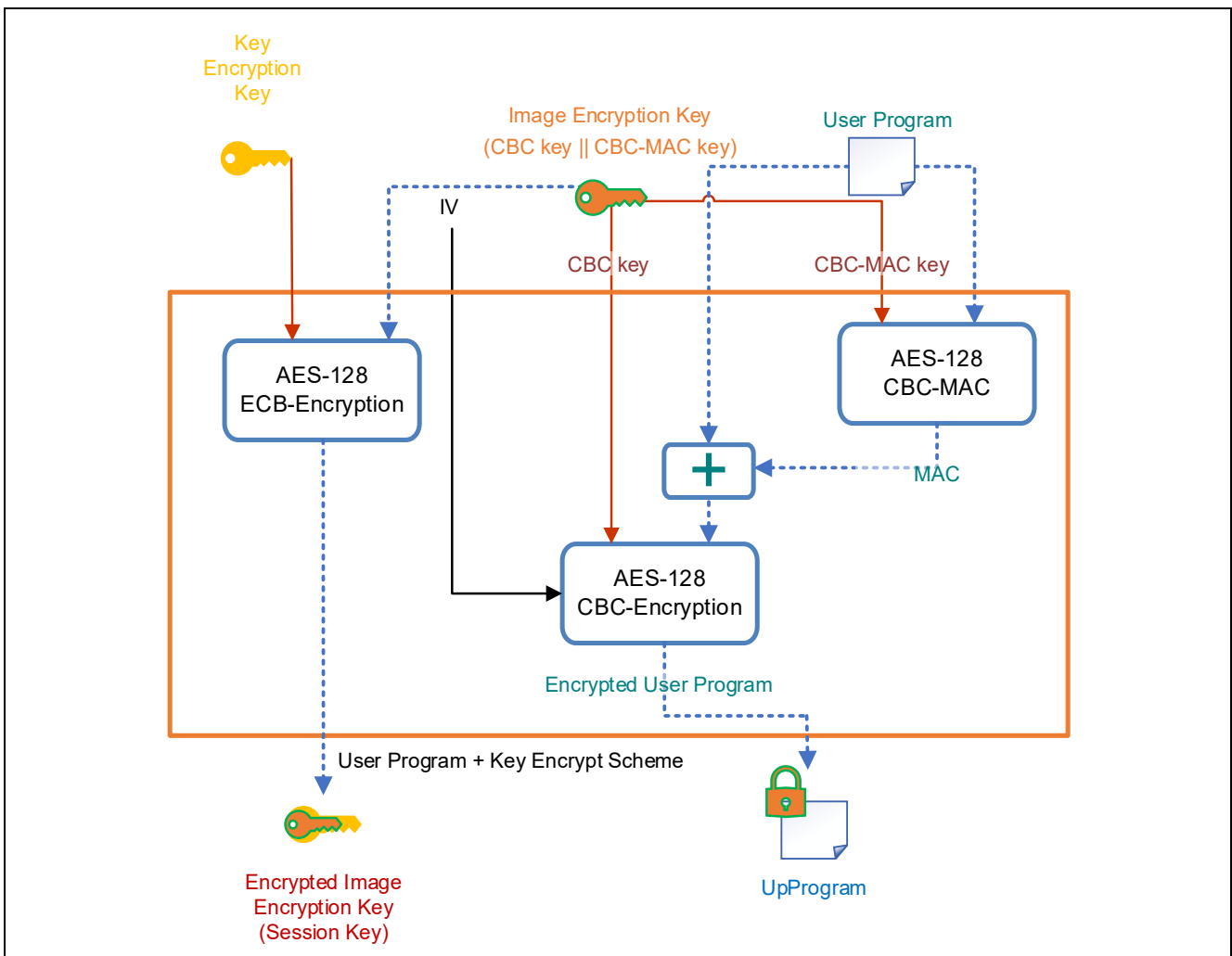


Figure 3.47 Firmware and Session Key Encryption Method

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

4. API Functions

4.1 List of APIs

The TSIP driver implements the following APIs:

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

The APIs implemented in the TSIP driver are summarized in the tables below. "XXX" in the name of an API represents either the bit length or the SHA mode.

Table 4-1 Common Function APIs

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

Table 4-2 Random Number Generation API

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

Table 4-3 AES Encryption/Decryption APIs

API	Description	TSIP-Lite	TSIP
R_TSIP_GenerateAesXXXKeyIndex	Generates an AES wrapped key.	✓	✓
R_TSIP_UpdateAesXXXKeyIndex	Updates an AES wrapped key.	✓	✓
R_TSIP_GenerateAesXXXRandomKeyIndex	Generates an AES key from a random number and outputs the wrapped key.	✓	✓
R_TSIP_AesXXXEcbEncryptInit R_TSIP_AesXXXEcbEncryptUpdate R_TSIP_AesXXXEcbEncryptFinal	Performs AES-ECB mode encryption.	✓	✓
R_TSIP_AesXXXEcbDecryptInit R_TSIP_AesXXXEcbDecryptUpdate R_TSIP_AesXXXEcbDecryptFinal	Performs AES-ECB mode decryption.	✓	✓
R_TSIP_AesXXXCbcEncryptInit R_TSIP_AesXXXCbcEncryptUpdate R_TSIP_AesXXXCbcEncryptFinal	Performs AES-CBC mode encryption.	✓	✓
R_TSIP_AesXXXCbcDecryptInit R_TSIP_AesXXXCbcDecryptUpdate R_TSIP_AesXXXCbcDecryptFinal	Performs AES128-CBC mode decryption.	✓	✓
R_TSIP_AesXXXCtrInit R_TSIP_AesXXXCtrUpdate R_TSIP_AesXXXCtrFinal	Performs AES-CTR mode encryption or decryption.	✓	✓
R_TSIP_AesXXXGcmEncryptInit R_TSIP_AesXXXGcmEncryptUpdate R_TSIP_AesXXXGcmEncryptFinal	Performs AES-GCM encryption.	✓	✓
R_TSIP_AesXXXGcmDecryptInit R_TSIP_AesXXXGcmDecryptUpdate R_TSIP_AesXXXGcmDecryptFinal	Performs AES-GCM decryption.	✓	✓
R_TSIP_AesXXXCcmEncryptInit R_TSIP_AesXXXCcmEncryptUpdate R_TSIP_AesXXXCcmEncryptFinal	Performs AES-CCM encryption.	✓	✓
R_TSIP_AesXXXCcmDecryptInit R_TSIP_AesXXXCcmDecryptUpdate R_TSIP_AesXXXCcmDecryptFinal	Performs AES-CCM decryption.	✓	✓
R_TSIP_AesXXXCmacGenerateInit R_TSIP_AesXXXCmacGenerateUpdate R_TSIP_AesXXXCmacGenerateFinal	Performs AES-CMAC mode MAC generation.	✓	✓
R_TSIP_AesXXXCmacVerifyInit R_TSIP_AesXXXCmacVerifyUpdate R_TSIP_AesXXXCmacVerifyFinal	Verifies a MAC generated in AES-CMAC mode.	✓	✓

Table 4-4 DES Encryption/Decryption APIs

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

Table 4-5 ARC4 Encryption/Decryption APIs

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

Table 4-6 RSA Operation APIs

API	Description	TSIP-Lite	TSIP
R_TSIP_GenerateRsaXXXPrivateKeyIndex	Generates an RSA secret wrapped key.	—	✓
R_TSIP_GenerateRsaXXXPublicKeyIndex	Generates an RSA public wrapped key.	—	✓
R_TSIP_UpdateRsaXXXPrivateKeyIndex	Updates an RSA secret wrapped key.	—	✓
R_TSIP_UpdateRsaXXXPublicKeyIndex	Updates an RSA public wrapped key.	—	✓
R_TSIP_GenerateRsaXXXRandomKeyIndex	Generates a pair of RSA wrapped keys from a random number. Exponent is fixed at 0x10001.	—	✓
R_TSIP_RsaesPkcsXXXEncrypt	Performs RSA encryption using RSAES-PKCS1-V1_5.	—	✓
R_TSIP_RsaesPkcsXXXDecrypt	Performs RSA decryption using RSAES-PKCS1-V1_5.	—	✓
R_TSIP_RsassaPkcsXXXSignatureGenerate	Generates a digital signature using RSASSA-PKCS1-V1_5.	—	✓
R_TSIP_RsassaPkcsXXXSignatureVerification	Verifies a digital signature using RSASSA-PKCS1-V1_5.	—	✓
R_TSIP_RsassaPssXXXSignatureGenerate	Generates a digital signature using RSASSA-PSS.	—	✓
R_TSIP_RsassaPssXXXSignatureVerification	Verifies a digital signature using RSASSA-PSS.	—	✓

Table 4-7 ECC Signature Generation/Verification APIs

API	Description	TSIP-Lite	TSIP
R_TSIP_GenerateEccPXXXPublicKeyIndex	Generates an ECC public wrapped key.	—	✓
R_TSIP_GenerateEccPXXXPrivateKeyIndex	Generates an ECC secret wrapped key.	—	✓
R_TSIP_UpdateEccPXXXPublicKeyIndex	Updates an ECC public wrapped key.	—	✓
R_TSIP_UpdateEccPXXXPrivateKeyIndex	Updates an ECC secret wrapped key.	—	✓
R_TSIP_GenerateEccPXXXRandomKeyIndex	Generates a pair of ECC wrapped keys from a random number.	—	✓
R_TSIP_EcdsaPXXXSignatureGenerate	Generates a digital signature using ECDSA.	—	✓
R_TSIP_EcdsaPXXXSignatureVerification	Verifies a digital signature using ECDSA.	—	✓

Table 4-8 HASH Calculation APIs

API	Description	TSIP-Lite	TSIP
R_TSIP_ShaXXXInit R_TSIP_ShaXXXUpdate R_TSIP_ShaXXXFinal	Performs hash value operations using SHA.	—	✓
R_TSIP_Md5Init R_TSIP_Md5Update R_TSIP_Md5Final	Performs hash value operations using MD5.	—	✓
R_TSIP_GetCurrentHashDigestValue	Gets hash value for current input.	—	✓

Table 4-9 HMAC Generation/Verification APIs

API	Description	TSIP-Lite	TSIP
R_TSIP_GenerateShaXXXHmacKeyIndex	Generates an SHA-HMAC wrapped key.	—	✓
R_TSIP_UpdateShaXXXHmacKeyIndex	Updates an SHA-HMAC wrapped key.	—	✓
R_TSIP_ShaXXXHmacGenerateInit R_TSIP_ShaXXXHmacGenerateUpdate R_TSIP_ShaXXXHmacGenerateFinal	Performs SHA-HMAC generation.	—	✓
R_TSIP_ShaXXXHmacVerifyInit R_TSIP_ShaXXXHmacVerifyUpdate R_TSIP_ShaXXXHmacVerifyFinal	Performs SHA-HMAC verification.	—	✓

Table 4-10 DH Calculation API

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

Table 4-11 ECDH Key Exchange APIs

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

Table 4-12 Key Exchange APIs

API	Description	TSIP-Lite	TSIP
R_TSIP_AesXXXKeyWrap	Wraps a key using an AES key.	✓	✓
R_TSIP_AesXXXKeyUnwrap	Unwraps a key wrapped with an AES key.	✓	✓

Table 4-13 TLS Function APIs

API	Description	TSIP -Lite	TSIP
R_TSIP_GenerateTlsXXRsaPublicKeyIndex	Generates an RSA public wrapped key used in TLS cooperation.	—	✓
R_TSIP_UpdateTlsXXRsaPublicKeyIndex	Updates an RSA public wrapped key used in TLS cooperation.	—	✓
R_TSIP_RegisterCaCertificationPublicKeyIndex		—	✓
R_TSIP_TlsXXRootCertificateVerification	Verifies a root CA certificate bundle.	—	✓
R_TSIP_TlsXXCertificateVerification	Verifies the signature of a server certificate or intermediate certificate.	—	✓
R_TSIP_TlsXXCertificateVerificationExtension	Verifies the signature of a server certificate or intermediate certificate.	—	✓
R_TSIP_TlsGeneratePreMasterSecret	Generates an encrypted pre-master secret.	—	✓
R_TSIP_TlsEncryptPreMasterSecretWithRsa2048PublicKey	Encrypts a pre-master secret using RSA-2048.	—	✓
R_TSIP_TlsSVGenerateServerRandom	Generates a random number value to use in ServerHello.	—	✓
R_TSIP_TlsSVDecryptPreMasterSecretWithRsa2048PrivateKey	Decrypts a pre-master secret using RSA-2048.	—	✓
R_TSIP_TlsXXGenerateMasterSecret	Generates an encrypted master secret.	—	✓
R_TSIP_TlsXXGenerateSessionKey	Outputs TLS communication keys.	—	✓
R_TSIP_TlsXXGenerateVerifyData	Generates a VerifyData message.	—	✓
R_TSIP_TlsServersEphemeralEcdhPublicKeyRetrieves	Verifies a server key exchange signature.	—	✓
R_TSIP_GenerateTlsXXP256EccKeyIndex	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_TlsXXGeneratePreMasterSecretWithEcp256Key	Generates an ECC encrypted pre-master secret.	—	✓
R_TSIP_TlsXXGenerateExtendedMasterSecret	Generates an encrypted extended master secret.	—	✓
R_TSIP_TlsSVCertificateVerifyVerification	Verifies a CertificateVerify message received from the client.	—	✓
R_TSIP_GenerateTls13P256EccKeyIndex	Generates a key pair from a random number used by the TLS 1.3 cooperation function for elliptic curve cryptography over a 256-bit prime field.	—	✓
R_TSIP_Tls13GenerateEcdheSharedSecret	Generates a shared secret wrapped key.	—	✓
R_TSIP_Tls13GenerateHandshakeSecret	Generates a handshake secret wrapped key.	—	✓
R_TSIP_Tls13GenerateServerHandshakeTrafficKey	Generates a server write wrapped key and server finished wrapped key.	—	✓
R_TSIP_Tls13ServerHandshakeVerification	Verifies finished information provided by the server.	—	✓
R_TSIP_Tls13GenerateClientHandshakeTrafficKey	Generates a client write wrapped key and client finished wrapped key.	—	✓
R_TSIP_Tls13GenerateMasterSecret	Generates a master secret key index.	—	✓
R_TSIP_Tls13GenerateApplicationTrafficKey	Generates an application traffic secret wrapped key and an application traffic wrapped key.	—	✓

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

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

Table 4-14 Firmware Update APIs

API	Description	TSIP-Lite	TSIP
R_TSIP_StartUpdateFirmware	Transitions to firmware update mode.	✓	✓
R_TSIP_GenerateFirmwareMAC	Decrypts and generates the MAC for encrypted firmware.	✓	✓
R_TSIP_VerifyFirmwareMAC	Performs a MAC check on the firmware.	✓	✓

4.2 Detailed Descriptions of API Functions

4.2.1 Common Function APIs

4.2.1.1 R_TSIP_Open

Format

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

Parameters

key_index_1	Input	TLS cooperation RSA public wrapped key
key_index_2	Input	Wrapped KUK

Return Values

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

Description

Enables use of TSIP functionality.

For key_index_1, input the "TLS cooperation RSA public wrapped 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 "Wrapped KUK" generated by R_TSIP_GenerateUpdateKeyRingKeyIndex(). If the key update cooperation function is not used, input a null pointer.

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

Reentrancy

Not supported.

4.2.1.2 R_TSIP_Close

Format

```
#include "r_tsip_rx_if.h"  
e_tsip_err_t R_TSIP_Close(void)
```

Parameters

None

Return Values

TSIP_SUCCESS

Normal termination

Description

Stops TSIP functionality.

Reentrancy

Not supported.

4.2.1.3 R_TSIP_SoftwareReset

Format

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

Parameters

None

Return Values

None

Description

Returns the TSIP to the initial state.

Reentrancy

Not supported.

4.2.1.4 R_TSIP_GetVersion

Format

```
#include "r_tsip_rx_if.h"
uint32_t R_TSIP_GetVersion(void)
```

Parameters

None

Return Values

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

Description

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

Reentrancy

Not supported.

4.2.1.5 R_TSIP_GenerateUpdateKeyRingKeyIndex

Format

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

Parameters

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

Return Values

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

Description

This API outputs a wrapped key of KUK.

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

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

Reentrancy

Not supported.

4.2.2 Random Number Generation

4.2.2.1 R_TSIP_GenerateRandomNumber

Format

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

Parameters

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

Return Values

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

Description

This API can be used to generate an NIST SP800-90A-compliant 4-word random number value.

Reentrancy

Not supported.

4.2.3 AES

4.2.3.1 R_TSIP_GenerateAesXXXKeyIndex

Format

- ```
(1) #include "r_tsip_rx_if.h"
 e_tsip_err_t R_TSIP_GenerateAes128KeyIndex(
 uint8_t *encrypted_provisioning_key,
 uint8_t *iv,
 uint8_t *encrypted_key,
 tsip_aes_key_index_t *key_index
)

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

##### Parameters

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

##### Return Values

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

##### Description

R\_TSIP\_GenerateAes128KeyIndex is an API that outputs an AES 128-bit wrapped key.

R\_TSIP\_GenerateAes256KeyIndex is an API that outputs an AES 256-bit wrapped key.

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

Refer to 3.7.1, Key Injection and Updating, for an explanation of encrypted\_provisioning\_key, iv, and encrypted\_key and how to use key\_index.

##### Reentrancy

Not supported.

### 4.2.3.2 R\_TSIP\_UpdateAesXXXKeyIndex

#### Format

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

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

Parameters

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

Return Values

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

Description

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

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

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

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

Reentrancy

Not supported.

4.2.3.3 R_TSIP_GenerateAesXXXRandomKeyIndex

Format

- ```
(1) #include "r_tsip_rx_if.h"
 e_tsip_err_t R_TSIP_GenerateAes128RandomKeyIndex(
 tsip_aes_key_index_t *key_index
)
(2) #include "r_tsip_rx_if.h"
 e_tsip_err_t R_TSIP_GenerateAes256RandomKeyIndex(
 tsip_aes_key_index_t *key_index
)
```

#### Parameters

|           |        |                                                            |
|-----------|--------|------------------------------------------------------------|
| key_index | Output | (1) 128-bit AES wrapped key<br>(2) 256-bit AES wrapped key |
|-----------|--------|------------------------------------------------------------|

#### Return Values

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

#### Description

R\_TSIP\_GenerateAes128RandomKeyIndex is an API that outputs an AES 128-bit wrapped key.

R\_TSIP\_GenerateAes256RandomKeyIndex is an API that outputs an AES 256-bit wrapped key.

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

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

#### Reentrancy

Not supported.

#### 4.2.3.4 R\_TSIP\_AesXXxEcbEncryptInit

##### Format

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

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

Parameters

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

Return Values

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

Description

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

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

Reentrancy

Not supported.

4.2.3.5 R_TSIP_AesXXxEcbEncryptUpdate

Format

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

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

#### Parameters

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

#### Return Values

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

#### Description

Using the handle specified by the first parameter, handle, the R\_TSIP\_AesXXxEcbEncryptUpdate() function encrypts the second parameter, plain, using the key index specified by the R\_TSIP\_AesXXxEcbEncryptInit() function, and writes the encrypted result to the third parameter, cipher. After plaintext input completes, call R\_TSIP\_AesXXxEcbEncryptFinal().

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

#### Reentrancy

Not supported.

### 4.2.3.6 R\_TSIP\_AesXXxEcbEncryptFinal

#### Format

- ```
(1) #include "r_tsip_rx_if.h"
    e_tsip_err_t R_TSIP_Aes128EcbEncryptFinal(
        tsip_aes_handle_t *handle,
        uint8_t *cipher,
        uint32_t *cipher_length
    )

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

Parameters

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

Return Values

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

Description

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

Reentrancy

Not supported.

4.2.3.7 R_TSIP_AesXXxEcbDecryptInit

Format

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

#### Parameters

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

#### Return Values

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

#### Description

The R\_TSIP\_AesXXxEcbDecryptInit() function performs preparations for the execution of AES calculation and writes the result to the first parameter, handle. The parameter handle is used subsequently as a parameter by the R\_TSIP\_AesXXxEcbDecryptUpdate() and R\_TSIP\_AesXXxEcbDecryptFinal() functions.

Refer to 3.7.1, Key Injection and Updating, for an explanation of how to generate key\_index.

#### Reentrancy

Not supported.

### 4.2.3.8 R\_TSIP\_AesXXxEcbDecryptUpdate

#### Format

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

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

Parameters

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

Return Values

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

Description

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

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

Reentrancy

Not supported.

4.2.3.9 R_TSIP_AesXXxEcbDecryptFinal

Format

- ```
(1) #include "r_tsip_rx_if.h"
 e_tsip_err_t R_TSIP_Aes128EcbDecryptFinal(
 tsip_aes_handle_t *handle,
 uint8_t *plain,
 uint32_t *plain_length
)

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

#### Parameters

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

#### Return Values

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

#### Description

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

#### Reentrancy

Not supported.

### 4.2.3.10 R\_TSIP\_AesXXxCbcEncryptInit

#### Format

```
(1) #include "r_tsip_rx_if.h"
 e_tsip_err_t R_TSIP_Aes128CbcEncryptInit(
 tsip_aes_handle_t *handle,
 tsip_aes_key_index_t *key_index,
 uint8_t *ivec
)
(2) #include "r_tsip_rx_if.h"
 e_tsip_err_t R_TSIP_Aes256CbcEncryptInit(
 tsip_aes_handle_t *handle,
 tsip_aes_key_index_t *key_index,
 uint8_t *ivec
)
```

#### Parameters

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

#### Return Values

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

#### Description

The R\_TSIP\_AesXXxCbcEncryptInit() function performs preparations for the execution of AES calculation, and writes the result to the first parameter, handle. The parameter handle is used subsequently as a parameter by the R\_TSIP\_AesXXxCbcEncryptUpdate() and R\_TSIP\_AesXXxCbcEncryptFinal() functions.

When using the TLS cooperation function, input client\_crypto\_key\_index or server\_crypto\_key\_index, generated by R\_TSIP\_TlsGenerateSessionKey(), as key\_index.

Refer to 3.7.1, Key Injection and Updating, for an explanation of how to generate key\_index.

#### Reentrancy

Not supported.

### 4.2.3.11 R\_TSIP\_AesXXxCbcEncryptUpdate

#### Format

- ```
(1) #include "r_tsip_rx_if.h"
    e_tsip_err_t R_TSIP_Aes128CbcEncryptUpdate(
        tsip_aes_handle_t *handle,
        uint8_t *plain,
        uint8_t *cipher,
        uint32_t plain_length
    )

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

Parameters

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

Return Values

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

Description

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

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

Reentrancy

Not supported.

4.2.3.12 R_TSIP_AesXXXCbcEncryptFinal

Format

- ```
(1) #include "r_tsip_rx_if.h"
 e_tsip_err_t R_TSIP_Aes128CbcEncryptFinal(
 tsip_aes_handle_t *handle,
 uint8_t *cipher,
 uint32_t *cipher_length
)

(2) #include "r_tsip_rx_if.h"
 e_tsip_err_t R_TSIP_Aes256CbcEncryptFinal(
 tsip_aes_handle_t *handle,
 uint8_t *cipher,
 uint32_t *cipher_length
)
```

#### Parameters

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

#### Return Values

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

#### Description

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

#### Reentrancy

Not supported.

### 4.2.3.13 R\_TSIP\_AesXXXCbcDecryptInit

#### Format

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

#### Parameters

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

#### Return Values

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

#### Description

The R\_TSIP\_AesXXXCbcDecryptInit() function performs preparations for the execution of AES calculation, and writes the result to the first parameter, handle. The parameter handle is used subsequently as a parameter by the R\_TSIP\_AesXXXCbcDecryptUpdate() and R\_TSIP\_AesXXXCbcDecryptFinal() functions.

When using the TLS cooperation function, input client\_crypto\_key\_index or server\_crypto\_key\_index, generated by R\_TSIP\_TlsGenerateSessionKey(), as key\_index.

Refer to 3.7.1, Key Injection and Updating, for an explanation of how to generate key\_index.

#### Reentrancy

Not supported.

### 4.2.3.14 R\_TSIP\_AesXXXCbcDecryptUpdate

#### Format

- ```
(1) #include "r_tsip_rx_if.h"
    e_tsip_err_t R_TSIP_Aes128CbcDecryptUpdate(
        tsip_aes_handle_t *handle,
        uint8_t *cipher,
        uint8_t *plain,
        uint32_t cipher_length
    )

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

Parameters

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

Return Values

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

Description

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

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

Reentrancy

Not supported.

4.2.3.15 R_TSIP_AesXXXCbcDecryptFinal

Format

- ```
(1) #include "r_tsip_rx_if.h"
 e_tsip_err_t R_TSIP_Aes128CbcDecryptFinal(
 tsip_aes_handle_t *handle,
 uint8_t *plain,
 uint32_t *plain_length
)

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

#### Parameters

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

#### Return Values

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

#### Description

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

#### Reentrancy

Not supported.

### 4.2.3.16 R\_TSIP\_AesXXXCtrInit

#### Format

```
(1) #include "r_tsip_rx_if.h"
 e_tsip_err_t R_TSIP_Aes128CtrInit(
 tsip_aes_handle_t *handle,
 tsip_aes_key_index_t *key_index,
 uint8_t *ictr
)
(2) #include "r_tsip_rx_if.h"
 e_tsip_err_t R_TSIP_Aes256CtrInit(
 tsip_aes_handle_t *handle,
 tsip_aes_key_index_t *key_index,
 uint8_t *ictr
)
```

#### Parameters

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

#### Return Values

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

#### Description

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

Refer to 3.7.1, Key Injection and Updating, for an explanation of how to generate key\_index.

#### Reentrancy

Not supported.

### 4.2.3.17 R\_TSIP\_AesXXXCtrUpdate

#### Format

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

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

Parameters

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

Return Values

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

Description

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

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

Reentrancy

Not supported.

4.2.3.18 R_TSIP_AesXXXCtrFinal

Format

- ```
(1) #include "r_tsip_rx_if.h"
 e_tsip_err_t R_TSIP_Aes128CtrFinal(
 tsip_aes_handle_t *handle
)
(2) #include "r_tsip_rx_if.h"
 e_tsip_err_t R_TSIP_Aes256CtrFinal(
 tsip_aes_handle_t *handle
)
```

**Parameters**

|        |       |                         |
|--------|-------|-------------------------|
| handle | Input | AES handler (work area) |
|--------|-------|-------------------------|

**Return Values**

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

**Description**

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

**Reentrancy**

Not supported.

### 4.2.3.19 R\_TSIP\_AesXXXGcmEncryptInit

#### Format

```
(1) #include "r_tsip_rx_if.h"
 e_tsip_err_t R_TSIP_Aes128GcmEncryptInit(
 tsip_gcm_handle_t *handle,
 tsip_aes_key_index_t *key_index,
 uint8_t *ivec,
 uint32_t ivec_len
)
(2) #include "r_tsip_rx_if.h"
 e_tsip_err_t R_TSIP_Aes256GcmEncryptInit(
 tsip_gcm_handle_t *handle,
 tsip_aes_key_index_t *key_index,
 uint8_t *ivec,
 uint32_t ivec_len
)
```

#### Parameters

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

#### Return Values

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

#### Description

The R\_TSIP\_AesXXXGcmEncryptInit() function performs preparations for the execution of GCM calculation and writes the result to the first parameter, handle. The parameter handle is used subsequently as a parameter by the R\_TSIP\_AesXXX1GcmEncryptUpdate() and R\_TSIP\_AesXXXGcmEncryptFinal() functions.

Note: 1. When key\_index->type is TSIP\_KEY\_INDEX\_TYPE\_AES128\_FOR\_TLS  
The key\_index value generated by the R\_TSIP\_TlsGenerateSessionKey() function includes a 96-bit IV when a value of 6 or 7 has been specified for select\_cipher. In this case, input a null pointer as the third parameter, ivec, and specify 0 as the fourth parameter, ivec\_len.

Refer to 3.7.1, Key Injection and Updating, for an explanation of how to generate key\_index.

#### Reentrancy

Not supported.

### 4.2.3.20 R\_TSIP\_AesXXXGcmEncryptUpdate

#### Format

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

#### Parameters

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

#### Return Values

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

#### Description

The R\_TSIP\_Aes128GcmEncryptUpdate() function encrypts the plaintext specified by the second parameter, plain, in GCM mode using the values specified for key\_index and ivec in R\_TSIP\_Aes128GcmEncryptInit() and the value specified by the fifth parameter, aad. The function internally buffers the data input by the user until the input values of aad and plain exceed 16 bytes. Once the input data from plain reaches 16 bytes or more, the encrypted result is output to the area specified by the third parameter, cipher. The lengths of the plain and aad data to be input are specified by the fourth parameter, plain\_data\_len, and the sixth parameter, aad\_len, respectively. For these, specify not the total byte count for the aad and plain input data, but rather the data length to be input when the user calls this function. If the input values of plain and aad are not divisible by 16 bytes, the function performs padding internally. First process the data to be input as aad, and then the data to be input as plain. If aad data is input after starting to input plain data, a root error occurs. If aad data and plain data are input to the function at the same time, the aad data is processed, and then the function transitions to the plain data input state. Except in cases where the addresses are the same, specify areas for plain and cipher that do not overlap.

**Reentrancy**

Not supported.

### 4.2.3.21 R\_TSIP\_AesXXXGcmEncryptFinal

#### Format

```
(1) #include "r_tsip_rx_if.h"
 e_tsip_err_t R_TSIP_Aes128GcmEncryptFinal(
 tsip_gcm_handle_t *handle,
 uint8_t *cipher,
 uint32_t *cipher_data_len,
 uint8_t *atag
)
(2) #include "r_tsip_rx_if.h"
 e_tsip_err_t R_TSIP_Aes256GcmEncryptFinal(
 tsip_gcm_handle_t *handle,
 uint8_t *cipher,
 uint32_t *cipher_data_len,
 uint8_t *atag
)
```

#### Parameters

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

#### Return Values

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

#### Description

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

#### Reentrancy

Not supported.



### 4.2.3.22 R\_TSIP\_AesXXXGcmDecryptInit

#### Format

```
(1) #include "r_tsip_rx_if.h"
 e_tsip_err_t R_TSIP_Aes128GcmDecryptInit(
 tsip_gcm_handle_t *handle,
 tsip_aes_key_index_t *key_index,
 uint8_t *ivec,
 uint32_t ivec_len
)
(2) #include "r_tsip_rx_if.h"
 e_tsip_err_t R_TSIP_Aes256GcmDecryptInit(
 tsip_gcm_handle_t *handle,
 tsip_aes_key_index_t *key_index,
 uint8_t *ivec,
 uint32_t ivec_len
)
```

#### Parameters

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

#### Return Values

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

#### Description

The R\_TSIP\_AesXXXGcmDecryptInit() function performs preparations for the execution of GCM calculation and writes the result to the first parameter, handle. The parameter handle is used subsequently as a parameter by the R\_TSIP\_AesXXXGcmDecryptUpdate() and R\_TSIP\_AesXXXGcmDecryptFinal() functions.

Note: 1. When key\_index->type is TSIP\_KEY\_INDEX\_TYPE\_AES128\_FOR\_TLS  
The key\_index value generated by the R\_TSIP\_TlsGenerateSessionKey() function includes a 96-bit IV when a value of 6 or 7 has been specified for select\_cipher. In this case, input a null pointer as the third parameter, ivec, and specify 0 as the fourth parameter, ivec\_len.

Refer to 3.7.1, Key Injection and Updating, for an explanation of how to generate key\_index.

#### Reentrancy

Not supported.

### 4.2.3.23 R\_TSIP\_AesXXXGcmDecryptUpdate

#### Format

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

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

Parameters

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

Return Values

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

Description

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

Reentrancy.

Not supported.

4.2.3.24 R_TSIP_AesXXXGcmDecryptFinal

Format

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

Parameters

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

Return Values

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

Description

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

Reentrancy

Not supported.

4.2.3.25 R_TSIP_AesXXXCcmEncryptInit

Format

```
(1) #include "r_tsip_rx_if.h"
    e_tsip_err_t R_TSIP_Aes128CcmEncryptInit(
        tsip_ccm_handle_t *handle,
        tsip_aes_key_index_t *key_index,
        uint8_t *nonce,
        uint32_t nonce_len,
        uint8_t *adata,
        uint8_t a_len,
        uint32_t payload_len,
        uint32_t mac_len
    )
(2) #include "r_tsip_rx_if.h"
    e_tsip_err_t R_TSIP_Aes256CcmEncryptInit(
        tsip_ccm_handle_t *handle,
        tsip_aes_key_index_t *key_index,
        uint8_t *nonce,
        uint32_t nonce_len,
        uint8_t *adata,
        uint8_t a_len,
        uint32_t payload_len,
        uint32_t mac_len
    )
```

Parameters

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

Return Values

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

Description

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

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

Reentrancy

Not supported.

4.2.3.26 R_TSIP_AesXXXCcmEncryptUpdate

Format

```
(1) #include "r_tsip_rx_if.h"
    e_tsip_err_t R_TSIP_Aes128CcmEncryptUpdate(
        tsip_ccm_handle_t *handle,
        uint8_t *plain,
        uint8_t *cipher,
        uint32_t plain_length
    )
(2) #include "r_tsip_rx_if.h"
    e_tsip_err_t R_TSIP_Aes256CcmEncryptUpdate(
        tsip_ccm_handle_t *handle,
        uint8_t *plain,
        uint8_t *cipher,
        uint32_t plain_length
    )
```

Parameters

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

Return Values

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

Description

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

Reentrancy

Not supported.

4.2.3.27 R_TSIP_AesXXXCcmEncryptFinal

Format

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

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

Parameters

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

Return Values

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

Description

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

Reentrancy

Not supported.

4.2.3.28 R_TSIP_AesXXXCcmDecryptInit

Format

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

Parameters

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

Return Values

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

Description

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

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

Reentrancy

Not supported.

4.2.3.29 R_TSIP_AesXXXCcmDecryptUpdate

Format

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

Parameters

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

Return Values

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

Description

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

Reentrancy

Not supported.

4.2.3.30 R_TSIP_AesXXXCcmDecryptFinal

Format

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

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

#### Parameters

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

#### Return Values

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

#### Description

If the data length of cipher input in R\_TSIP\_AesXXXCcmDecryptUpdate() results in a fractional remainder after 16 bytes, the R\_TSIP\_AesXXXCcmDecryptFinal() function outputs the leftover decrypted data to the second parameter, cipher. In addition, the function verifies the fourth parameter, mac. Set the fifth parameter, mac\_length, to the same value as that specified for the parameter mac\_length in R\_TSIP\_AesXXXCcmDecryptInit().

#### Reentrancy

Not supported.

### 4.2.3.31 R\_TSIP\_AesXXXCmacGenerateInit

#### Format

- ```
(1) #include "r_tsip_rx_if.h"
    e_tsip_err_t R_TSIP_Aes128CmacGenerateInit(
        tsip_cmac_handle_t *handle,
        tsip_aes_key_index_t *key_index
    )
(2) #include "r_tsip_rx_if.h"
    e_tsip_err_t R_TSIP_Aes256CmacGenerateInit(
        tsip_cmac_handle_t *handle,
        tsip_aes_key_index_t *key_index
    )
```

Parameters

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

Return Values

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

Description

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

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

Reentrancy

Not supported.

4.2.3.32 R_TSIP_AesXXXCmacGenerateUpdate

Format

```
(1) #include "r_tsip_rx_if.h"
    e_tsip_err_t R_TSIP_Aes128CmacGenerateUpdate(
        tsip_cmac_handle_t *handle,
        uint8_t *message,
        uint32_t message_length
    )
(2) #include "r_tsip_rx_if.h"
    e_tsip_err_t R_TSIP_Aes256CmacGenerateUpdate(
        tsip_cmac_handle_t *handle,
        uint8_t *message,
        uint32_t message_length
    )
```

Parameters

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

Return Values

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

Description

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

Reentrancy

Not supported.

4.2.3.33 R_TSIP_AesXXXCmacGenerateFinal

Format

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

**Parameters**

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

**Return Values**

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

**Description**

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

**Reentrancy**

Not supported.

### 4.2.3.34 R\_TSIP\_AesXXXCmacVerifyInit

#### Format

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

#### Parameters

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

#### Return Values

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

#### Description

The R\_TSIP\_AesXXXCmacVerifyInit() function performs preparations for the execution of CMAC calculation, and writes the result to the first parameter, handle. The parameter handle is used subsequently as a parameter by the R\_TSIP\_AesXXXCmacVerifyUpdate() and R\_TSIP\_AesXXXCmacVerifyFinal() functions.

Refer to 3.7.1, Key Injection and Updating, for an explanation of how to generate key\_index.

#### Reentrancy

Not supported.



### 4.2.3.35 R\_TSIP\_AesXXXCmacVerifyUpdate

#### Format

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

#### Parameters

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

#### Return Values

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

#### Description

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

#### Reentrancy

Not supported.

---

### 4.2.3.36 R\_TSIP\_AesXXXCmacVerifyFinal

---

**Format**

```
(1) #include "r_tsip_rx_if.h"
 e_tsip_err_t R_TSIP_Aes128CmacVerifyFinal(
 tsip_cmac_handle_t *handle,
 uint8_t *mac,
 uint32_t mac_length
)
(2) #include "r_tsip_rx_if.h"
 e_tsip_err_t R_TSIP_Aes256CmacVerifyFinal(
 tsip_cmac_handle_t *handle,
 uint8_t *mac,
 uint32_t mac_length
)
```

**Parameters**

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

**Return Values**

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

**Description**

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

**Reentrancy**

Not supported.

## 4.2.4 DES

### 4.2.4.1 R\_TSIP\_GenerateTdesKeyIndex

#### Format

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

#### Parameters

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

#### Return Values

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

#### Description

This API outputs an DES wrapped key.

For encrypted\_key, input the data indicated in 7.3.2, DES, encrypted using the UFPK. Refer to 3.7.1, Key Injection and Updating, for an explanation of encrypted\_key, iv, and encrypted\_provisioning\_key and how to use key\_index.

#### Reentrancy

Not supported.

---

#### 4.2.4.2 R\_TSIP\_UpdateTdesKeyIndex

---

**Format**

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

**Parameters**

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

**Return Values**

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

**Description**

This API outputs a TDES wrapped key.

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

**Reentrancy**

Not supported.

---

### 4.2.4.3 R\_TSIP\_GenerateTdesRandomKeyIndex

---

**Format**

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

**Parameters**

|           |        |             |
|-----------|--------|-------------|
| key_index | Output | Wrapped key |
|-----------|--------|-------------|

**Return Values**

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

**Description**

This API outputs a TDES wrapped key.

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

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

**Reentrancy**

Not supported.

---

#### 4.2.4.4 R\_TSIP\_TdesEcbEncryptInit

---

**Format**

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

**Parameters**

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

**Return Values**

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

**Description**

The R\_TSIP\_TdesEcbEncryptInit() function performs preparations for the execution of DES calculation and writes the result to the first parameter, handle. The parameter handle is used subsequently as a parameter by the R\_TSIP\_TdesEcbEncryptUpdate() and R\_TSIP\_TdesEcbEncryptFinal() functions.

Refer to 3.7.1, Key Injection and Updating, for an explanation of how to generate key\_index.

**Reentrancy**

Not supported.

#### 4.2.4.5 R\_TSIP\_TdesEcbEncryptUpdate

##### Format

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

##### Parameters

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

##### Return Values

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

##### Description

Using the handle specified by the first parameter, handle, the R\_TSIP\_TdesEcbEncryptUpdate() function encrypts the second parameter, plain, using the key index specified by the R\_TSIP\_TdesEcbEncryptInit() function, and writes the encrypted result to the third parameter, cipher. After plaintext input completes, call R\_TSIP\_TdesEcbEncryptFinal().

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

##### Reentrancy

Not supported.

#### 4.2.4.6 R\_TSIP\_TdesEcbEncryptFinal

##### Format

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

##### Parameters

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

##### Return Values

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

##### Description

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

##### Reentrancy

Not supported.



---

#### 4.2.4.7 R\_TSIP\_TdesEcbDecryptInit

---

**Format**

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

**Parameters**

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

**Return Values**

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

**Description**

The R\_TSIP\_TdesEcbDecryptInit() function performs preparations for the execution of DES calculation and writes the result to the first parameter, handle. The parameter handle is used subsequently as a parameter by the R\_TSIP\_TdesEcbDecryptUpdate() and R\_TSIP\_TdesEcbDecryptFinal() functions.

Refer to 3.7.1, Key Injection and Updating, for an explanation of how to generate key\_index.

**Reentrancy**

Not supported.

#### 4.2.4.8 R\_TSIP\_TdesEcbDecryptUpdate

##### Format

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

##### Parameters

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

##### Return Values

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

##### Description

Using the handle specified by the first parameter, handle, the R\_TSIP\_TdesEcbDecryptUpdate() function decrypts the second parameter, cipher, using the key index specified by the R\_TSIP\_TdesEcbDecryptInit() function, and writes the decrypted result to the third parameter, plain. After ciphertext input completes, call R\_TSIP\_TdesEcbDecryptFinal().

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

##### Reentrancy

Not supported.

#### 4.2.4.9 R\_TSIP\_TdesEcbDecryptFinal

##### Format

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

##### Parameters

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

##### Return Values

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

##### Description

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

##### Reentrancy

Not supported.

---

#### 4.2.4.10 R\_TSIP\_TdesCbcEncryptInit

---

**Format**

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

**Parameters**

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

**Return Values**

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

**Description**

The R\_TSIP\_TdesCbcEncryptInit() function performs preparations for the execution of DES calculation, and writes the result to the first parameter, handle. The parameter handle is used subsequently as a parameter by the R\_TSIP\_TdesCbcEncryptUpdate() and R\_TSIP\_TdesCbcEncryptFinal() functions.

Refer to 3.7.1, Key Injection and Updating, for an explanation of how to generate key\_index.

**Reentrancy**

Not supported.

#### 4.2.4.11 R\_TSIP\_TdesCbcEncryptUpdate

##### Format

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

##### Parameters

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

##### Return Values

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

##### Description

Using the handle specified by the first parameter, handle, the R\_TSIP\_TdesCbcEncryptUpdate() function encrypts the second parameter, plain, using the key index specified by the R\_TSIP\_TdesCbcEncryptInit() function, and writes the encrypted result to the third parameter, cipher. After plaintext input completes, call R\_TSIP\_TdesCbcEncryptFinal().

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

##### Reentrancy

Not supported.

#### 4.2.4.12 R\_TSIP\_TdesCbcEncryptFinal

##### Format

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

##### Parameters

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

##### Return Values

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

##### Description

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

##### Reentrancy

Not supported.

---

#### 4.2.4.13 R\_TSIP\_TdesCbcDecryptInit

---

**Format**

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

**Parameters**

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

**Return Values**

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

**Description**

The R\_TSIP\_TdesCbcDecryptInit() function performs preparations for the execution of DES calculation, and writes the result to the first parameter, handle. The parameter handle is used subsequently as a parameter by the R\_TSIP\_TdesCbcDecryptUpdate() and R\_TSIP\_TdesCbcDecryptFinal() functions.

Refer to 3.7.1, Key Injection and Updating, for an explanation of how to generate key\_index.

**Reentrancy**

Not supported.

---

#### 4.2.4.14 R\_TSIP\_TdesCbcDecryptUpdate

---

**Format**

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

**Parameters**

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

**Return Values**

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

**Description**

Using the handle specified by the first parameter, handle, the R\_TSIP\_TdesCbcDecryptUpdate() function decrypts the second parameter, cipher, utilizing the key index specified by the R\_TSIP\_TdesCbcDecryptInit() function, and writes the decrypted result to the third parameter, plain. After ciphertext input completes, call R\_TSIP\_TdesCbcDecryptFinal().

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

**Reentrancy**

Not supported.



---

#### 4.2.4.15 R\_TSIP\_TdesCbcDecryptFinal

---

##### Format

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

##### Parameters

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

##### Return Values

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

##### Description

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

##### Reentrancy

Not supported.

## 4.2.5 ARC4

### 4.2.5.1 R\_TSIP\_GenerateArc4KeyIndex

#### Format

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

#### Parameters

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

#### Return Values

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

#### Description

This API outputs an ARC4 wrapped key.

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

Refer to 3.7.1, Key Injection and Updating, for an explanation of encrypted\_key, iv, and encrypted\_provisioning\_key and how to use key\_index.

#### Reentrancy

Not supported.

---

#### 4.2.5.2 R\_TSIP\_UpdateArc4KeyIndex

---

**Format**

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

**Parameters**

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

**Return Values**

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

**Description**

This API updates an ARC4 wrapped key.

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

Refer to 3.7.1, Key Injection and Updating, for an explanation of iv and encrypted\_key and how to use key\_index.

**Reentrancy**

Not supported.

---

### 4.2.5.3 R\_TSIP\_GenerateArc4RandomKeyIndex

---

**Format**

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

**Parameters**

|           |        |             |
|-----------|--------|-------------|
| key_index | Output | Wrapped key |
|-----------|--------|-------------|

**Return Values**

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

**Description**

This API outputs an ARC4 wrapped key.

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

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

**Reentrancy**

Not supported.

---

#### 4.2.5.4 R\_TSIP\_Arc4EncryptInit

---

**Format**

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

**Parameters**

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

**Return Values**

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

**Description**

The R\_TSIP\_Arc4EncryptInit() function performs preparations for the execution of ARC4 calculation, and writes the result to the first parameter, handle. The parameter handle is used subsequently as a parameter by the R\_TSIP\_Arc4EncryptUpdate() and R\_TSIP\_Arc4EncryptFinal() functions.

Refer to 3.7.1, Key Injection and Updating, for an explanation of how to generate key\_index.

**Reentrancy**

Not supported.

### 4.2.5.5 R\_TSIP\_Arc4EncryptUpdate

#### Format

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

#### Parameters

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

#### Return Values

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

#### Description

The R\_TSIP\_Arc4EncryptUpdate() function encrypts the second parameter, plain, using the key index specified by the R\_TSIP\_Arc4EncryptInit() function, and writes the encrypted result to the third parameter, cipher. After plaintext input completes, call R\_TSIP\_Arc4EncryptFinal().

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

#### Reentrancy

Not supported.

#### 4.2.5.6 R\_TSIP\_Arc4EncryptFinal

##### Format

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

##### Parameters

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

##### Return Values

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

##### Description

Using the handle specified by the first parameter, handle, the R\_TSIP\_Arc4EncryptFinal() function writes the calculation result to the second parameter, cipher, and writes the length of the calculation result to the third parameter, cipher\_length. The original intent was for any portion of the encrypted result that was not a multiple of 16 bytes to be written to the second parameter. However, due to the restriction that only multiples of 16 can be input to the R\_TSIP\_Arc4EncryptUpdate() function, nothing is ever written to cipher, and 0 is always written to cipher\_length. The parameters cipher and cipher\_length are provided for future compatibility in anticipation of this restriction eventually being removed.

##### Reentrancy

Not supported.

---

### 4.2.5.7 R\_TSIP\_Arc4DecryptInit

---

**Format**

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

**Parameters**

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

**Return Values**

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

**Description**

The R\_TSIP\_Arc4DecryptInit() function performs preparations for the execution of ARC4 calculation, and writes the result to the first parameter, handle. The parameter handle is used subsequently as a parameter by the R\_TSIP\_Arc4DecryptUpdate() and R\_TSIP\_Arc4DecryptFinal() functions.

Refer to 3.7.1, Key Injection and Updating, for an explanation of how to generate key\_index.

**Reentrancy**

Not supported.



#### 4.2.5.8 R\_TSIP\_Arc4DecryptUpdate

##### Format

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

##### Parameters

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

##### Return Values

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

##### Description

Using the handle specified by the first parameter, handle, the R\_TSIP\_Arc4DecryptUpdate() function decrypts the second parameter, cipher, utilizing the key index specified by the R\_TSIP\_Arc4DecryptInit() function, and writes the decrypted result to the third parameter, plain. After ciphertext input completes, call R\_TSIP\_Arc4DecryptFinal().

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

##### Reentrancy

Not supported.

---

#### 4.2.5.9 R\_TSIP\_Arc4DecryptFinal

---

**Format**

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

**Parameters**

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

**Return Values**

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

**Description**

Using the handle specified by the first parameter, handle, the R\_TSIP\_Arc4DecryptFinal() function writes the calculation result to the second parameter, plain, and writes the length of the calculation result to the third parameter, plain\_length. The original intent was for any portion of the decrypted result that was not a multiple of 16 bytes to be written to the second parameter. However, due to the restriction that only multiples of 16 can be input to the R\_TSIP\_Arc4DecryptUpdate() function, nothing is ever written to plain, and 0 is always written to plain\_length. The parameters plain and plain\_length are provided for future compatibility in anticipation of this restriction eventually being removed.

**Reentrancy**

Not supported.

## 4.2.6 RSA

### 4.2.6.1 R\_TSIP\_GenerateRsaXXXPublicKeyIndex

#### Format

- ```
(1) #include "r_tsip_rx_if.h"
    e_tsip_err_t R_TSIP_GenerateRsa1024PublicKeyIndex(
        uint8_t *encrypted_provisioning_key,
        uint8_t *iv,
        uint8_t *encrypted_key,
        tsip_rsa1024_public_key_index_t *key_index
    )

(2) #include "r_tsip_rx_if.h"
    e_tsip_err_t R_TSIP_GenerateRsa2048PublicKeyIndex(
        uint8_t *encrypted_provisioning_key,
        uint8_t *iv,
        uint8_t *encrypted_key,
        tsip_rsa2048_public_key_index_t *key_index
    )

(3) #include "r_tsip_rx_if.h"
    e_tsip_err_t R_TSIP_GenerateRsa3072PublicKeyIndex(
        uint8_t *encrypted_provisioning_key,
        uint8_t *iv,
        uint8_t *encrypted_key,
        tsip_rsa3072_public_key_index_t *key_index
    )

(4) #include "r_tsip_rx_if.h"
    e_tsip_err_t R_TSIP_GenerateRsa4096PublicKeyIndex(
        uint8_t *encrypted_provisioning_key,
        uint8_t *iv,
        uint8_t *encrypted_key,
        tsip_rsa4096_public_key_index_t *key_index
    )
```

Parameters

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

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

Return Values

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

Description

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

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

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

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

Reentrancy

Not supported.

4.2.6.2 R_TSIP_GenerateRsaXXXPrivateKeyIndex

Format

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

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

#### Parameters

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

#### Return Values

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

#### Description

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

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

Ensure that the areas allocated for encrypted\_key and key\_index do not overlap.

Refer to 3.7.1, Key Injection and Updating, for an explanation of encrypted\_provisioning\_key, iv, and encrypted\_key and how to use key\_index.

#### Reentrancy

Not supported.

### 4.2.6.3 R\_TSIP\_UpdateRsaXXXPublicKeyIndex

#### Format

- ```
(1) #include "r_tsip_rx_if.h"
    e_tsip_err_t R_TSIP_UpdateRsa1024PublicKeyIndex(
        uint8_t *iv,
        uint8_t *encrypted_key,
        tsip_rsa1024_public_key_index_t *key_index
    )
(2) #include "r_tsip_rx_if.h"
    e_tsip_err_t R_TSIP_UpdateRsa2048PublicKeyIndex(
        uint8_t *iv,
        uint8_t *encrypted_key,
        tsip_rsa2048_public_key_index_t *key_index
    )
(3) #include "r_tsip_rx_if.h"
    e_tsip_err_t R_TSIP_UpdateRsa3072PublicKeyIndex(
        uint8_t *iv,
        uint8_t *encrypted_key,
        tsip_rsa3072_public_key_index_t *key_index
    )
(4) #include "r_tsip_rx_if.h"
    e_tsip_err_t R_TSIP_UpdateRsa4096PublicKeyIndex(
        uint8_t *iv,
        uint8_t *encrypted_key,
        tsip_rsa4096_public_key_index_t *key_index
    )
```

Parameters

iv	Input	Initialization vector when generating encrypted_key
encrypted_key	Input	Encrypted key encrypted using KUK
key_index	Output	Wrapped key
value		Public key value
key_management_info1		Key management information
key_n		Modulus n (plaintext)
		(1) 1024-bit RSA
		(2) 2048-bit RSA
		(3) 3072-bit RSA
		(4) 4096-bit RSA
key_e		Exponent e (plaintext)
		(1) 1024-bit RSA
		(2) 2048-bit RSA
		(3) 3072-bit RSA
		(4) 4096-bit RSA
dummy		Dummy
key_management_info2		Key management information

Return Values

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

Description

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

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

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

Reentrancy

Not supported.

4.2.6.4 R_TSIP_UpdateRsaXXXPrivateKeyIndex

Format

```
(1) #include "r_tsip_rx_if.h"
    e_tsip_err_t R_TSIP_UpdateRsa1024PrivateKeyIndex(
        uint8_t *iv,
        uint8_t *encrypted_key,
        tsip_rsa1024_private_key_index_t *key_index
    )
(2) #include "r_tsip_rx_if.h"
    e_tsip_err_t R_TSIP_UpdateRsa2048PrivateKeyIndex(
        uint8_t *iv,
        uint8_t *encrypted_key,
        tsip_rsa2048_private_key_index_t *key_index
    )
```

Parameters

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

Return Values

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

Description

This API updates a 1024-bit or 2048-bit RSA secret wrapped key. Refer to 7.3.4, RSA, for the format of the data encrypted using the UFPK input as encrypted_key.

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

Reentrancy

Not supported.

4.2.6.5 R_TSIP_GenerateRsaXXXRandomKeyIndex

Format

- ```
(1) #include "r_tsip_rx_if.h"
 e_tsip_err_t R_TSIP_GenerateRsa1024RandomKeyIndex(
 tsip_rsa1024_key_pair_index_t *key_pair_index
)
(2) #include "r_tsip_rx_if.h"
 e_tsip_err_t R_TSIP_GenerateRsa2048RandomKeyIndex(
 tsip_rsa2048_key_pair_index_t *key_pair_index
)
```

##### Parameters

| key_pair_index       | Output | RSA key pair wrapped keys                                      |
|----------------------|--------|----------------------------------------------------------------|
| public               |        | RSA public wrapped key                                         |
| value                |        | Public key value                                               |
| key_management_info1 |        | Key management information                                     |
| key_n                |        | Modulus n (plaintext)<br>(1) 1024-bit RSA<br>(2) 2048-bit RSA  |
| key_e                |        | Exponent e (plaintext)<br>(1) 1024-bit RSA<br>(2) 2048-bit RSA |
| dummy                |        | Dummy                                                          |
| key_management_info2 |        | Key management information                                     |
| private              |        | RSA secret wrapped key                                         |

##### Return Values

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

##### Description

This API outputs 1024-bit or 2048-bit wrapped keys for an RSA public key–secret key pair. The API generates a user key from a random number within the TSIP. Therefore, no user key needs to be input. Encrypting data using the wrapped keys output by this API makes it possible to prevent dead copying of data. The public wrapped key is generated in `key_pair_index->public`, and the secret wrapped key is generated in `key_pair_index->private`. The only public key exponent generated is 0x00010001.

Refer to 3.7.1, Key Injection and Updating, for an explanation of how to use `key_pair_index->public` and `key_pair_index->private`. The parameter `key_pair_index->public` is utilized in the same manner as the public wrapped key output by `R_TSIP_GenerateRsaXXXPublicKeyIndex()`, and `key_pair_index->private` is utilized in the same manner as the secret wrapped key output by `R_TSIP_GenerateRsaXXXPrivateKeyIndex()`.

**Reentrancy**

Not supported.

#### 4.2.6.6 R\_TSIP\_RsaesPkcsXXXEncrypt

##### Format

- ```
(1) #include "r_tsip_rx_if.h"
    e_tsip_err_t R_TSIP_RsaesPkcs1024Encrypt(
        tsip_rsa_byte_data_t *plain,
        tsip_rsa_byte_data_t *cipher,
        tsip_rsa1024_public_key_index_t *key_index
    )
(2) #include "r_tsip_rx_if.h"
    e_tsip_err_t R_TSIP_RsaesPkcs2048Encrypt(
        tsip_rsa_byte_data_t *plain,
        tsip_rsa_byte_data_t *cipher,
        tsip_rsa2048_public_key_index_t *key_index
    )
(3) #include "r_tsip_rx_if.h"
    e_tsip_err_t R_TSIP_RsaesPkcs3072Encrypt(
        tsip_rsa_byte_data_t *plain,
        tsip_rsa_byte_data_t *cipher,
        tsip_rsa3072_public_key_index_t *key_index
    )
(4) #include "r_tsip_rx_if.h"
    e_tsip_err_t R_TSIP_RsaesPkcs4096Encrypt(
        tsip_rsa_byte_data_t *plain,
        tsip_rsa_byte_data_t *cipher,
        tsip_rsa4096_public_key_index_t *key_index
    )
```

Parameters

plain	Input	Plaintext data to be encrypted
pdata		Specifies pointer to array containing plaintext.
data_length		Specifies valid data length of plaintext array. Data size <= modulus n size – 11
cipher	Output	Encrypted data
pdata		Specifies pointer to array containing ciphertext.
data_length		Inputs ciphertext buffer size, and outputs valid data length after encryption (modulus n size)
key_index	Input	Wrapped key (1) 1024-bit RSA (2) 2048-bit RSA (3) 3072-bit RSA (4) 4096-bit RSA

Return Values

TSIP_SUCCESS:	Normal termination
TSIP_ERR_RESOURCE_CONFLICT:	Occurrence of resource conflict because a hardware resource needed by the processing routine was in use by another processing routine
TSIP_ERR_KEY_SET	Invalid wrapped key input
TSIP_ERR_PARAMETER	Invalid input data
TSIP_ERR_FAIL	Occurrence of internal error (Only for XXX = 3072, 4096)

Description

The R_TSIP_RsaesPkcsXXXEncrypt() function encrypts in RSA mode the plaintext input as the first parameter, plain, according to RSAES-PKCS1-V1_5. It then writes the encrypted result to the second parameter, cipher.

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

Reentrancy

Not supported.

4.2.6.7 R_TSIP_RsaesPkcsXXXDecrypt

Format

```
(1) #include "r_tsip_rx_if.h"
    e_tsip_err_t R_TSIP_RsaesPkcs1024Decrypt(
        tsip_rsa_byte_data_t *cipher,
        tsip_rsa_byte_data_t *plain,
        tsip_rsa1024_private_key_index_t *key_index
    )
(2) #include "r_tsip_rx_if.h"
    e_tsip_err_t R_TSIP_RsaesPkcs2048Decrypt(
        tsip_rsa_byte_data_t *cipher,
        tsip_rsa_byte_data_t *plain,
        tsip_rsa2048_private_key_index_t *key_index
    )
```

Parameters

cipher	Input	Encrypted data to be decrypted
pdata		Specifies pointer to array containing ciphertext.
data_length		Specifies valid data length of ciphertext array (modulus n size).
plain	Output	Decrypted plaintext data
pdata		Specifies pointer to array containing plaintext.
data_length		Specifies valid data length of plaintext array. Data size <= modulus n size – 11
key_index	Input	Wrapped key
		(1) 1024-bit RSA
		(2) 2048-bit RSA

Return Values

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

Description

The R_TSIP_RsaesPkcsXXXDecrypt() function decrypts in RSA mode the ciphertext input as the first parameter, cipher, according to RSAES-PKCS1-V1_5. It then writes the decrypted result to the second parameter, plain.

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

Reentrancy

Not supported.

4.2.6.8 R_TSIP_RsassaPkcsXXXSignatureGenerate

Format

```
(1) #include "r_tsip_rx_if.h"
    e_tsip_err_t R_TSIP_RsassaPkcs1024SignatureGenerate(
        tsip_rsa_byte_data_t *message_hash,
        tsip_rsa_byte_data_t *signature,
        tsip_rsa1024_private_key_index_t *key_index,
        uint8_t hash_type
    )
(2) #include "r_tsip_rx_if.h"
    e_tsip_err_t R_TSIP_RsassaPkcs2048SignatureGenerate(
        tsip_rsa_byte_data_t *message_hash,
        tsip_rsa_byte_data_t *signature,
        tsip_rsa2048_private_key_index_t *key_index,
        uint8_t hash_type
    )
```

Parameters

message_hash	Input	Message or hash value information to which to attach signature
pdata		Specifies pointer to array containing message or hash value.
data_length		Specifies valid data length of array (specified for message only).
data_type		Selects data type of message_hash. Message: 0 Hash value: 1
signature	Output	Signature text storage destination information
pdata		Specifies pointer to array containing signature text.
data_length		Data length (byte units)
key_index	Input	Wraped key (1) 1024-bit RSA (2) 2048-bit RSA
hash_type	Input	Hash type to attach to signature R_TSIP_RSA_HASH_MD5 R_TSIP_RSA_HASH_SHA1 R_TSIP_RSA_HASH_SHA256

Return Values

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

Description

The R_TSIP_RsassaPkcsXXXSignatureGenerate() function generates, in accordance with RSASSA-PKCS1-V1_5, signature text from the message text or hash value input as the first parameter, message_hash, using the secret wrapped key input as the third parameter, key_index, and writes the result to the second parameter, signature. When a message is specified as the first parameter, message_hash->data_type, a hash value is calculated from the message as specified by the fourth parameter, hash_type. When specifying a hash value in the first parameter, message_hash->data_type, a hash value calculated with the hash algorithm specified by the fourth parameter, hash_type, must be input as message_hash->pdata.

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

Reentrancy

Not supported.

4.2.6.9 R_TSIP_RsassaPkcsXXXSignatureVerification

Format

- ```
(1) #include "r_tsip_rx_if.h"
 e_tsip_err_t R_TSIP_RsassaPkcs1024SignatureVerification(
 tsip_rsa_byte_data_t *signature,
 tsip_rsa_byte_data_t *message_hash,
 tsip_rsa1024_public_key_index_t *key_index,
 uint8_t hash_type
)

(2) #include "r_tsip_rx_if.h"
 e_tsip_err_t R_TSIP_RsassaPkcs2048SignatureVerification(
 tsip_rsa_byte_data_t *signature,
 tsip_rsa_byte_data_t *message_hash,
 tsip_rsa2048_public_key_index_t *key_index,
 uint8_t hash_type
)

(3) #include "r_tsip_rx_if.h"
 e_tsip_err_t R_TSIP_RsassaPkcs3072SignatureVerification(
 tsip_rsa_byte_data_t *signature,
 tsip_rsa_byte_data_t *message_hash,
 tsip_rsa3072_public_key_index_t *key_index,
 uint8_t hash_type
)

(4) #include "r_tsip_rx_if.h"
 e_tsip_err_t R_TSIP_RsassaPkcs4096SignatureVerification(
 tsip_rsa_byte_data_t *signature,
 tsip_rsa_byte_data_t *message_hash,
 tsip_rsa4096_public_key_index_t *key_index,
 uint8_t hash_type
)
```

#### Parameters

|              |       |                                                                                             |
|--------------|-------|---------------------------------------------------------------------------------------------|
| signature    | Input | Signature text information to be verified                                                   |
| pdata        |       | Specifies pointer to array containing signature text.                                       |
| message_hash | Input | Message text or hash value information to be verified                                       |
| pdata        |       | Specifies pointer to array containing message or hash value.                                |
| data_length  |       | Specifies valid data length of array (specified for message only).                          |
| data_type    |       | Selects data type of message_hash.<br>Message: 0<br>Hash value: 1                           |
| key_index    | Input | Wrapped key<br>(1) 1024-bit RSA<br>(2) 2048-bit RSA<br>(3) 3072-bit RSA<br>(4) 4096-bit RSA |



|           |       |                                                                                    |
|-----------|-------|------------------------------------------------------------------------------------|
| hash_type | Input | Hash type<br>R_TSIP_RSA_HASH_MD5<br>R_TSIP_RSA_HASH_SHA1<br>R_TSIP_RSA_HASH_SHA256 |
|-----------|-------|------------------------------------------------------------------------------------|

**Return Values**

|                             |                                                                                                                                       |
|-----------------------------|---------------------------------------------------------------------------------------------------------------------------------------|
| TSIP_SUCCESS:               | Normal termination                                                                                                                    |
| TSIP_ERR_RESOURCE_CONFLICT: | Occurrence of resource conflict because a hardware resource needed by the processing routine was in use by another processing routine |
| TSIP_ERR_KEY_SET            | Invalid wrapped key input                                                                                                             |
| TSIP_ERR_AUTHENTICATION     | Signature verification failure                                                                                                        |
| TSIP_ERR_PARAMETER          | Invalid input data                                                                                                                    |
| TSIP_ERR_FAIL               | Occurrence of internal error (Only for XXX = 3072, 4096)                                                                              |

**Description**

R\_TSIP\_RsassaPkcsXXXSignatureVerification() function verifies, in accordance with RSASSAPKCS1-V1\_5, the signature text input as the first parameter signature, and the message text or hash value input as the second parameter, message\_hash, using the public wrapped key input as the third parameter, key\_index. When a message is specified by the second parameter, message\_hash->data\_type, a hash value is calculated using the public wrapped key input as the third parameter, key\_index, as specified by the fourth parameter, hash\_type. When specifying a hash value in the second parameter, message\_hash->data\_type, a hash value calculated with the hash algorithm specified by the fourth parameter, hash\_type, must be input as message\_hash->pdata.

Refer to 3.7.1, Key Injection and Updating, for an explanation of how to generate key\_index.

**Reentrancy**

Not supported.

#### 4.2.6.10 R\_TSIP\_RsassaPssXXXSignatureGenerate

##### Format

```
(1) #include "r_tsip_rx_if.h"
 e_tsip_err_t R_TSIP_RsassaPss1024SignatureGenerate(
 tsip_rsa_byte_data_t *message_hash,
 tsip_rsa_byte_data_t *signature,
 tsip_rsa1024_private_key_index_t *key_index,
 uint8_t hash_type
)
(2) #include "r_tsip_rx_if.h"
 e_tsip_err_t R_TSIP_RsassaPss2048SignatureGenerate(
 tsip_rsa_byte_data_t *message_hash,
 tsip_rsa_byte_data_t *signature,
 tsip_rsa2048_private_key_index_t *key_index,
 uint8_t hash_type
)
```

##### Parameters

|              |        |                                                                                    |
|--------------|--------|------------------------------------------------------------------------------------|
| message_hash | Input  | Message or hash value information to which to attach signature                     |
| pdata        |        | Specifies pointer to array containing message or hash value.                       |
| data_length  |        | Specifies valid data length of array (specified for message only).                 |
| data_type    |        | Selects data type of message_hash.<br>Message: 0<br>Hash value: 1                  |
| signature    | Output | Signature text storage destination information                                     |
| pdata        |        | Specifies pointer to array containing signature text.                              |
| data_length  |        | Data length (byte units)                                                           |
| key_index    | Input  | Wrapped key<br>(1) 1024-bit RSA<br>(2) 2048-bit RSA                                |
| hash_type    | Input  | Hash type to attach to signature<br>R_TSIP_RSA_HASH_SHA1<br>R_TSIP_RSA_HASH_SHA256 |

##### Return Values

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

**Description**

The R\_TSIP\_RsassaPssXXXSignatureGenerate() function generates, in accordance with RSASSA-PSS in section 8.1 of RFC 8017, signature text from the message text or hash value input as the first parameter, message\_hash, using the secret wrapped key input as the third parameter, key\_index, and writes the result to the second parameter, signature. The salt length is fixed at 32 bytes.

When a message is specified as the first parameter, message\_hash->data\_type, a hash value is calculated from the message as specified by the fourth parameter, hash\_type.

When specifying a hash value in the first parameter, message\_hash->data\_type, a hash value calculated with the hash algorithm specified by the fourth parameter, hash\_type, must be input as message\_hash->pdata.

Refer to 3.7.1, Key Injection and Updating, for an explanation of how to generate key\_index.

**Reentrancy**

Not supported.

#### 4.2.6.11 R\_TSIP\_RsassaPssXXXSignatureVerification

##### Format

```
(1) #include "r_tsip_rx_if.h"
 e_tsip_err_t R_TSIP_RsassaPss1024SignatureVerification(
 tsip_rsa_byte_data_t *signature,
 tsip_rsa_byte_data_t *message_hash,
 tsip_rsa1024_public_key_index_t *key_index,
 uint8_t hash_type
)
(2) #include "r_tsip_rx_if.h"
 e_tsip_err_t R_TSIP_RsassaPss2048SignatureVerification(
 tsip_rsa_byte_data_t *signature,
 tsip_rsa_byte_data_t *message_hash,
 tsip_rsa2048_public_key_index_t *key_index,
 uint8_t hash_type
)
```

##### Parameters

|              |       |                                                                    |
|--------------|-------|--------------------------------------------------------------------|
| signature    | Input | Signature text information to be verified                          |
| pdata        |       | Specifies pointer to array containing signature text.              |
| message_hash | Input | Message text or hash value information to be verified              |
| pdata        |       | Specifies pointer to array containing message or hash value.       |
| data_length  |       | Specifies valid data length of array (specified for message only). |
| data_type    |       | Selects data type of message_hash.<br>Message: 0<br>Hash value: 1  |
| key_index    | Input | Wraped key<br>(1) 1024-bit RSA<br>(2) 2048-bit RSA                 |
| hash_type    | Input | Hash type<br>R_TSIP_RSA_HASH_SHA1<br>R_TSIP_RSA_HASH_SHA256        |

##### Return Values

|                             |                                                                                                                                       |
|-----------------------------|---------------------------------------------------------------------------------------------------------------------------------------|
| TSIP_SUCCESS:               | Normal termination                                                                                                                    |
| TSIP_ERR_RESOURCE_CONFLICT: | Occurrence of resource conflict because a hardware resource needed by the processing routine was in use by another processing routine |
| TSIP_ERR_KEY_SET            | Invalid wrapped key input                                                                                                             |
| TSIP_ERR_AUTHENTICATION     | Signature verification failure                                                                                                        |
| TSIP_ERR_PARAMETER          | Invalid input data                                                                                                                    |

**Description**

The R\_TSIP\_RsassaPssXXXSignatureVerification() function verifies, in accordance with RSASSA-PSS, the signature text input as the first parameter signature, and the message text or hash value input as the second parameter, message\_hash, using the public wrapped key input as the third parameter, key\_index. The salt length is fixed at 32 bytes.

When a message is specified by the second parameter, message\_hash->data\_type, a hash value is calculated using the public wrapped key input as the third parameter, key\_index, as specified by the fourth parameter, hash\_type.

When specifying a hash value in the second parameter, message\_hash->data\_type, a hash value calculated with the hash algorithm specified by the fourth parameter, hash\_type, must be input as message\_hash->pdata.

Refer to 3.7.1, Key Injection and Updating, for an explanation of how to generate key\_index.

**Reentrancy**

Not supported.

## 4.2.7 ECC

### 4.2.7.1 R\_TSIP\_GenerateEccPXXXPublicKeyIndex

#### Format

- ```
(1) #include "r_tsip_rx_if.h"
    e_tsip_err_t R_TSIP_GenerateEccP192PublicKeyIndex(
        uint8_t *encrypted_provisioning_key,
        uint8_t *iv,
        uint8_t *encrypted_key,
        tsip_ecc_public_key_index_t *key_index
    )

(2) #include "r_tsip_rx_if.h"
    e_tsip_err_t R_TSIP_GenerateEccP224PublicKeyIndex(
        uint8_t *encrypted_provisioning_key,
        uint8_t *iv,
        uint8_t *encrypted_key,
        tsip_ecc_public_key_index_t *key_index
    )

(3) #include "r_tsip_rx_if.h"
    e_tsip_err_t R_TSIP_GenerateEccP256PublicKeyIndex(
        uint8_t *encrypted_provisioning_key,
        uint8_t *iv,
        uint8_t *encrypted_key,
        tsip_ecc_public_key_index_t *key_index
    )

(4) #include "r_tsip_rx_if.h"
    e_tsip_err_t R_TSIP_GenerateEccP384PublicKeyIndex(
        uint8_t *encrypted_provisioning_key,
        uint8_t *iv,
        uint8_t *encrypted_key,
        tsip_ecc_public_key_index_t *key_index
    )
```

Parameters

encrypted_provisioning_key	Input	W-UFPK
iv	Input	Initialization vector when generating encrypted_key
encrypted_key	Input	Encrypted key encrypted using UFPK
key_index	Output	Wrapped key
value		Public key value
key_management_info		Key management information
key_q		(1) ECC P-192 public key Q (plaintext) (2) ECC P-224 public key Q (plaintext) (3) ECC P-256 public key Q (plaintext) (4) ECC P-384 public key Q (plaintext)

Return Values

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

Description

This API outputs an ECC P-192, P-224, P-256, or P-384 public wrapped key.

Refer to 7.3.5, ECC, for an explanation of the method used to encrypt the public key using the provisioning key input as encrypted_key.

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

A structure which includes the public key plaintext data is output as key_index->value.key_q. Refer to 7.4.2, ECC, for the format.

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

Reentrancy

Not supported.

4.2.7.2 R_TSIP_GenerateEccPXXXPrivateKeyIndex

Format

- ```
(1) #include "r_tsip_rx_if.h"
 e_tsip_err_t R_TSIP_GenerateEccP192PrivateKeyIndex(
 uint8_t *encrypted_provisioning_key,
 uint8_t *iv,
 uint8_t *encrypted_key,
 tsip_ecc_private_key_index_t *key_index
)

(2) #include "r_tsip_rx_if.h"
 e_tsip_err_t R_TSIP_GenerateEccP224PrivateKeyIndex(
 uint8_t *encrypted_provisioning_key,
 uint8_t *iv,
 uint8_t *encrypted_key,
 tsip_ecc_private_key_index_t *key_index
)

(3) #include "r_tsip_rx_if.h"
 e_tsip_err_t R_TSIP_GenerateEccP256PrivateKeyIndex(
 uint8_t *encrypted_provisioning_key,
 uint8_t *iv,
 uint8_t *encrypted_key,
 tsip_ecc_private_key_index_t *key_index
)

(4) #include "r_tsip_rx_if.h"
 e_tsip_err_t R_TSIP_GenerateEccP384PrivateKeyIndex(
 uint8_t *encrypted_provisioning_key,
 uint8_t *iv,
 uint8_t *encrypted_key,
 tsip_ecc_private_key_index_t *key_index
)
```

#### Parameters

|                            |        |                                                                                                                                                             |
|----------------------------|--------|-------------------------------------------------------------------------------------------------------------------------------------------------------------|
| encrypted_provisioning_key | Input  | W-UFPK                                                                                                                                                      |
| iv                         | Input  | Initialization vector when generating encrypted_key                                                                                                         |
| encrypted_key              | Input  | Encrypted key encrypted using UFPK                                                                                                                          |
| key_index                  | Output | Wrapped key<br>(1) ECC P-192 secret wrapped key<br>(2) ECC P-224 secret wrapped key<br>(3) ECC P-256 secret wrapped key<br>(4) ECC P-384 secret wrapped key |

#### Return Values

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



**Description**

This API outputs an ECC P-192, P-224, P-256, or P-384 secret wrapped key.

Refer to 7.4, Public Key Index Formats for Asymmetric Cryptography, for the format of the data encrypted using the UFPK input as `encrypted_key`.

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

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

**Reentrancy**

Not supported.

### 4.2.7.3 R\_TSIP\_UpdateEccPXXXPublicKeyIndex

#### Format

- ```
(1) #include "r_tsip_rx_if.h"
    e_tsip_err_t R_TSIP_UpdateEccP192PublicKeyIndex(
        uint8_t *iv,
        uint8_t *encrypted_key,
        tsip_ecc_public_key_index_t *key_index
    )
(2) #include "r_tsip_rx_if.h"
    e_tsip_err_t R_TSIP_UpdateEccP224PublicKeyIndex(
        uint8_t *iv,
        uint8_t *encrypted_key,
        tsip_ecc_public_key_index_t *key_index
    )
(3) #include "r_tsip_rx_if.h"
    e_tsip_err_t R_TSIP_UpdateEccP256PublicKeyIndex(
        uint8_t *iv,
        uint8_t *encrypted_key,
        tsip_ecc_public_key_index_t *key_index
    )
(4) #include "r_tsip_rx_if.h"
    e_tsip_err_t R_TSIP_UpdateEccP384PublicKeyIndex(
        uint8_t *iv,
        uint8_t *encrypted_key,
        tsip_ecc_public_key_index_t *key_index
    )
```

Parameters

iv	Input	Initialization vector when generating encrypted_key
encrypted_key	Input	Encrypted key encrypted using KUK
key_index	Output	Wrapped key
value		Public key value
key_management_info		Key management information
key_q		Public key (Qx Qy) (plaintext)
		(1) ECC P-192
		(2) ECC P-224
		(3) ECC P-256
		(4) ECC P-384

Return Values

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

Description

This API updates an ECC P-192, P-224, P-256, or P-384 public wrapped key.

Refer to 7.3.5, ECC, for an explanation of the method and format used to encrypt the public key using the KUK input as `encrypted_key`.

Refer to 7.4.2, ECC, for the format of the public key Q plaintext data output as `key_index->value.key_q`.

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

Reentrancy

Not supported.

4.2.7.4 R_TSIP_UpdateEccPXXXPrivateKeyIndex

Format

- ```
(1) #include "r_tsip_rx_if.h"
 e_tsip_err_t R_TSIP_UpdateEccP192PrivateKeyIndex(
 uint8_t *iv,
 uint8_t *encrypted_key,
 tsip_ecc_private_key_index_t *key_index
)
(2) #include "r_tsip_rx_if.h"
 e_tsip_err_t R_TSIP_UpdateEccP224PrivateKeyIndex(
 uint8_t *iv,
 uint8_t *encrypted_key,
 tsip_ecc_private_key_index_t *key_index
)
(3) #include "r_tsip_rx_if.h"
 e_tsip_err_t R_TSIP_UpdateEccP256PrivateKeyIndex(
 uint8_t *iv,
 uint8_t *encrypted_key,
 tsip_ecc_private_key_index_t *key_index
)
(4) #include "r_tsip_rx_if.h"
 e_tsip_err_t R_TSIP_UpdateEccP384PrivateKeyIndex(
 uint8_t *iv,
 uint8_t *encrypted_key,
 tsip_ecc_private_key_index_t *key_index
)
```

##### Parameters

|               |        |                                                                                                                             |
|---------------|--------|-----------------------------------------------------------------------------------------------------------------------------|
| iv            | Input  | Initialization vector when generating encrypted_key                                                                         |
| encrypted_key | Input  | Encrypted key encrypted using KUK                                                                                           |
| key_index     | Output | Wrapped key<br>(1) ECC P-192 secret key<br>(2) ECC P-224 secret key<br>(3) ECC P-256 secret key<br>(4) ECC P-384 secret key |

##### Return Values

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

**Description**

This API updates an ECC P-192, P-224, P-256, or P-384 secret wrapped key.

Refer to 7.3.5, ECC, for an explanation of the method and format used to encrypt the private key using the KUK input as encrypted\_key.

Refer to 3.7.1, Key Injection and Updating, for an explanation of iv and encrypted\_key and how to use key\_index.

**Reentrancy**

Not supported.

### 4.2.7.5 R\_TSIP\_GenerateEccPXXXRandomKeyIndex

#### Format

- ```
(1) #include "r_tsip_rx_if.h"
    e_tsip_err_t R_TSIP_GenerateEccP192RandomKeyIndex(
        tsip_ecc_key_pair_index_t *key_pair_index
    )
(2) #include "r_tsip_rx_if.h"
    e_tsip_err_t R_TSIP_GenerateEccP224RandomKeyIndex(
        tsip_ecc_key_pair_index_t *key_pair_index
    )
(3) #include "r_tsip_rx_if.h"
    e_tsip_err_t R_TSIP_GenerateEccP256RandomKeyIndex(
        tsip_ecc_key_pair_index_t *key_pair_index
    )
(4) #include "r_tsip_rx_if.h"
    e_tsip_err_t R_TSIP_GenerateEccP384RandomKeyIndex(
        tsip_ecc_key_pair_index_t *key_pair_index
    )
```

Parameters

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

Return Values

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

Description

This API outputs P-192, P-224, P-256, or P-384 wrapped keys for an ECC public key–secret key pair. The API generates a user key from a random number within the TSIP. Therefore, no user key needs to be input. Encrypting data using the wrapped keys output by this API makes it possible to prevent dead copying of data. The public wrapped key is generated in `key_pair_index->public`, and the secret wrapped key is generated in `key_pair_index->private`. The plaintext public key is output to `key_pair_index->public.value.key_q`. Refer to 7.4, Public Key Index Formats for Asymmetric Cryptography, for the data format.

Refer to 3.7.1, Key Injection and Updating, for an explanation of how to use `key_pair_index->public` and `key_pair_index->private`. The parameter `key_pair_index->public` is utilized in the same manner as the public wrapped key output by `R_TSIP_GenerateEccPXXXPublicKeyIndex()`, and `key_pair_index->private` is utilized in the same manner as the secret wrapped key output by `R_TSIP_GenerateEccPXXXPrivateKeyIndex()`.

Reentrancy

Not supported.

4.2.7.6 R_TSIP_EcdsaPXXXSignatureGenerate

Format

- ```
(1) #include "r_tsip_rx_if.h"
 e_tsip_err_t R_TSIP_EcdsaP192SignatureGenerate(
 tsip_ecdsa_byte_data_t *message_hash,
 tsip_ecdsa_byte_data_t *signature,
 tsip_ecc_private_key_index_t *key_index
)

(2) #include "r_tsip_rx_if.h"
 e_tsip_err_t R_TSIP_EcdsaP224SignatureGenerate(
 tsip_ecdsa_byte_data_t *message_hash,
 tsip_ecdsa_byte_data_t *signature,
 tsip_ecc_private_key_index_t *key_index
)

(3) #include "r_tsip_rx_if.h"
 e_tsip_err_t R_TSIP_EcdsaP256SignatureGenerate(
 tsip_ecdsa_byte_data_t *message_hash,
 tsip_ecdsa_byte_data_t *signature,
 tsip_ecc_private_key_index_t *key_index
)

(4) #include "r_tsip_rx_if.h"
 e_tsip_err_t R_TSIP_EcdsaP384SignatureGenerate(
 tsip_ecdsa_byte_data_t *message_hash,
 tsip_ecdsa_byte_data_t *signature,
 tsip_ecc_private_key_index_t *key_index
)
```

#### Parameters

|              |        |                                                                                                                                                                                                                                                                                                                                                                                                                                  |
|--------------|--------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| message_hash | Input  | Message or hash value information to which to attach signature                                                                                                                                                                                                                                                                                                                                                                   |
| pdata        |        | Specifies pointer to array containing message or hash value.                                                                                                                                                                                                                                                                                                                                                                     |
| data_length  |        | Specifies valid data length of array (specified for message only).                                                                                                                                                                                                                                                                                                                                                               |
| data_type    |        | Selects data type of message_hash.<br>Message: 0<br>Hash value: 1                                                                                                                                                                                                                                                                                                                                                                |
| signature    | Output | Signature text storage destination information                                                                                                                                                                                                                                                                                                                                                                                   |
| pdata        |        | Specifies pointer to array containing signature text.<br>The signature format is as follows:<br>(1) "0 padding (64 bits)    signature r (192 bits)    0 padding (64 bits)    signature s (192 bits)"<br>(2) "0 padding (32 bits)    signature r (224 bits)    0 padding (32 bits)    signature s (224 bits)"<br>(3) "signature r (256 bits)    signature s (256 bits)"<br>(4) "signature r (384 bits)    signature s (384 bits)" |



|             |       |                                                                                                                             |
|-------------|-------|-----------------------------------------------------------------------------------------------------------------------------|
| data_length |       | Data length (byte units)                                                                                                    |
| key_index   | Input | Wrapped key<br>(1) ECC P-192 secret key<br>(2) ECC P-224 secret key<br>(3) ECC P-256 secret key<br>(4) ECC P-384 secret key |

### Return Values

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

### Description

When using (1) R\_TSIP\_EcdsaP192SignatureGenerate, (2) R\_TSIP\_EcdsaP224SignatureGenerate, or (3) R\_TSIP\_EcdsaP256SignatureGenerate

When a message is specified by the first parameter, message\_hash->data\_type, an SHA-256 hash of the message text input as the first parameter, message\_hash->pdata, is calculated, and the signature text is written to the second parameter, signature, in accordance with ECDSA P-192, P-224, or P-256 using the secret wrapped key input as the third parameter, key\_index.

When a hash value is specified by the first parameter, message\_hash->data\_type, the signature text for the first XXX bits (XXX/8 bytes) of the SHA-256 hash value input as the first parameter, message\_hash->pdata, is written to the second parameter, signature, in accordance with ECDSA P-192, P-224, or P-256 using the secret wrapped key input as the third parameter, key\_index.

When using (4) R\_TSIP\_EcdsaP384SignatureGenerate

When a message is specified by the first parameter, message\_hash->data\_type, an SHA-384 hash of the message text input as the first parameter, message\_hash->pdata, is calculated, and the signature text is written to the second parameter, signature, in accordance with ECDSA P-384 using the secret wrapped key input as the third parameter, key\_index.

To use message input, refer to 4.3 and prepare a user-defined function for SHA384.

When a hash value is specified by the first parameter, message\_hash->data\_type, the signature text for the entire 48 bytes of the SHA-384 hash value input as the first parameter, message\_hash->pdata, is written to the second parameter, signature, in accordance with ECDSA P-384 using the secret wrapped key input as the third parameter, key\_index.

Refer to 3.7.1, Key Injection and Updating, for an explanation of how to generate key\_index.

### Reentrancy

Not supported.

### 4.2.7.7 R\_TSIP\_EcdsaPXXXSignatureVerification

#### Format

- ```
(1) #include "r_tsip_rx_if.h"
    e_tsip_err_t R_TSIP_EcdsaP192SignatureVerification(
        tsip_ecdsa_byte_data_t *signature,
        tsip_ecdsa_byte_data_t *message_hash,
        tsip_ecc_public_key_index_t *key_index
    )

(2) #include "r_tsip_rx_if.h"
    e_tsip_err_t R_TSIP_EcdsaP224SignatureVerification(
        tsip_ecdsa_byte_data_t *signature,
        tsip_ecdsa_byte_data_t *message_hash,
        tsip_ecc_public_key_index_t *key_index
    )

(3) #include "r_tsip_rx_if.h"
    e_tsip_err_t R_TSIP_EcdsaP256SignatureVerification(
        tsip_ecdsa_byte_data_t *signature,
        tsip_ecdsa_byte_data_t *message_hash,
        tsip_ecc_public_key_index_t *key_index
    )

(4) #include "r_tsip_rx_if.h"
    e_tsip_err_t R_TSIP_EcdsaP384SignatureVerification(
        tsip_ecdsa_byte_data_t *signature,
        tsip_ecdsa_byte_data_t *message_hash,
        tsip_ecc_public_key_index_t *key_index
    )
```

Parameters

signature	Input	Signature text information to be verified
pdata		Specifies pointer to array containing signature text. The signature format is as follows: <ol style="list-style-type: none"> (1) "0 padding (64 bits) signature r (192 bits) 0 padding (64 bits) signature s (192 bits)" (2) "0 padding (32 bits) signature r (224 bits) 0 padding (32 bits) signature s (224 bits)" (3) "signature r (256 bits) signature s (256 bits)" (4) "signature r (384 bits) signature s (384 bits)"
message_hash	Input	Message text or hash value information to be verified
pdata		Specifies pointer to array containing message or hash value.
data_length		Specifies valid data length of array (specified for message only).

`data_type` Selects data type of `message_hash`.
 Message: 0
 Hash value: 1

`key_index` Input Wrapped key
 (1) ECC P-192 public key
 (2) ECC P-224 public key
 (3) ECC P-256 public key
 (4) ECC P-384 public key

Return Values

`TSIP_SUCCESS`: Normal termination
`TSIP_ERR_RESOURCE_CONFLICT`: Occurrence of resource conflict because a hardware resource needed by the processing routine was in use by another processing routine
`TSIP_ERR_KEY_SET`: Invalid wrapped key index input
`TSIP_ERR_FAIL`: Internal error, or signature verification failure
`TSIP_ERR_PARAMETER`: Invalid input data

Description

When using (1) `R_TSIP_EcdsaP192SignatureVerification`, (2) `R_TSIP_EcdsaP224SignatureVerification`, or (3) `R_TSIP_EcdsaP256SignatureVerification`

When a message is specified by the second parameter, `message_hash->data_type`, an SHA-256 hash of the message text input as the second parameter, `message_hash->pdata`, is calculated, and the signature text input as the first parameter, `signature`, is validated in accordance with ECDSA P-192, P-224 or P-256 using the public wrapped key input as the third parameter, `key_index`.

When a hash value is specified by the second parameter, `message_hash->data_type`, the signature text for the first XXX bits (XXX/8 bytes) of the SHA-256 hash value input as the second parameter, `message_hash->pdata`, and the signature text input as the first parameter, `signature`, is validated in accordance with ECDSA P-192, P-224 or P-256 using the public wrapped key input as the third parameter, `key_index`.

When using (4) `R_TSIP_EcdsaP384SignatureVerification`

When a message is specified by the second parameter, `message_hash->data_type`, an SHA-384 hash of the message text input as the second parameter, `message_hash->pdata`, is calculated, and the signature text input as the first parameter, `signature`, is validated in accordance with ECDSA P-384 using the public wrapped key input as the third parameter, `key_index`.

To use message input, refer to 4.3 and prepare a user-defined function for SHA384.

When a hash value is specified by the second parameter, `message_hash->data_type`, the signature text for the entire 48 bytes of the SHA-384 hash value input as the second parameter, `message_hash->pdata`, and the signature text input as the first parameter, `signature`, is validated in accordance with ECDSA P-384 using the public wrapped key input as the third parameter, `key_index`.

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

Reentrancy

Not supported.

4.2.8 HASH

4.2.8.1 R_TSIP_ShaXXXInit

Format

- ```
(1) #include "r_tsip_rx_if.h"
 e_tsip_err_t R_TSIP_Sha1Init(
 tsip_sha_md5_handle_t *handle
)

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

#### Parameters

|        |        |                         |
|--------|--------|-------------------------|
| handle | Output | SHA handler (work area) |
|--------|--------|-------------------------|

#### Return Values

|               |                    |
|---------------|--------------------|
| TSIP_SUCCESS: | Normal termination |
|---------------|--------------------|

#### Description

The R\_TSIP\_ShaXXXInit() function performs preparations for the execution of an SHA1 or SHA256 hash calculation, and writes the result to the first parameter, handle. The parameter handle is used subsequently as a parameter by the R\_TSIP\_ShaXXXUpdate() and R\_TSIP\_ShaXXXFinal() functions.

#### Reentrancy

Not supported.

### 4.2.8.2 R\_TSIP\_ShaXXXUpdate

#### Format

```
(1) #include "r_tsip_rx_if.h"
 e_tsip_err_t R_TSIP_Sha1Update(
 tsip_sha_md5_handle_t *handle,
 uint8_t *message,
 uint32_t message_length
)
(2) #include "r_tsip_rx_if.h"
 e_tsip_err_t R_TSIP_Sha256Update(
 tsip_sha_md5_handle_t *handle,
 uint8_t *message,
 uint32_t message_length
)
```

#### Parameters

|                |              |                         |
|----------------|--------------|-------------------------|
| handle         | Input/output | SHA handler (work area) |
| message        | Input        | Message area            |
| message_length | Input        | Byte length of message  |

#### Return Values

|                             |                                                                                                                                       |
|-----------------------------|---------------------------------------------------------------------------------------------------------------------------------------|
| TSIP_SUCCESS:               | Normal termination                                                                                                                    |
| TSIP_ERR_RESOURCE_CONFLICT: | Occurrence of resource conflict because a hardware resource needed by the processing routine was in use by another processing routine |
| TSIP_ERR_PARAMETER:         | Invalid handle input                                                                                                                  |
| TSIP_ERR_PROHIBIT_FUNCTION: | Invalid function called                                                                                                               |

#### Description

Using the handle specified by the first parameter, handle, the R\_TSIP\_ShaXXXUpdate() function calculates a hash value based on the second parameter, message, and the third parameter, message\_length, and writes the ongoing status to the first parameter, handle (the value of which can be fetched using R\_TSIP\_GetCurrentHashDigestValue()). After message input completes, call R\_TSIP\_ShaXXXFinal().

#### Reentrancy

Not supported.

### 4.2.8.3 R\_TSIP\_ShaXXXFinal

#### Format

```
(1) #include "r_tsip_rx_if.h"
 e_tsip_err_t R_TSIP_Sha1Final(
 tsip_sha_md5_handle_t *handle,
 uint8_t *digest,
 uint32_t *digest_length
)
(2) #include "r_tsip_rx_if.h"
 e_tsip_err_t R_TSIP_Sha256Final(
 tsip_sha_md5_handle_t *handle,
 uint8_t *digest,
 uint32_t *digest_length
)
```

#### Parameters

|               |        |                                                                        |
|---------------|--------|------------------------------------------------------------------------|
| handle        | Input  | SHA handler (work area)                                                |
| digest        | Output | Hash data area                                                         |
| digest_length | Output | Byte length of hash data<br>(1) SHA1: 20 bytes<br>(2) SHA256: 32 bytes |

#### Return Values

|                             |                                                                                                                                       |
|-----------------------------|---------------------------------------------------------------------------------------------------------------------------------------|
| TSIP_SUCCESS:               | Normal termination                                                                                                                    |
| TSIP_ERR_RESOURCE_CONFLICT: | Occurrence of resource conflict because a hardware resource needed by the processing routine was in use by another processing routine |
| TSIP_ERR_PARAMETER:         | Invalid handle input                                                                                                                  |
| TSIP_ERR_PROHIBIT_FUNCTION: | Invalid function called                                                                                                               |

#### Description

Using the handle specified as the first parameter, handle, the R\_TSIP\_ShaXXXFinal() function writes the calculation result to the second parameter, digest, and writes the length of the calculation result to the third parameter, digest\_length.

#### Reentrancy

Not supported.

#### 4.2.8.4 R\_TSIP\_Md5Init

---

**Format**

```
#include "r_tsip_rx_if.h"
e_tsip_err_t R_TSIP_Md5Init(
 tsip_sha_md5_handle_t *handle
)
```

**Parameters**

|        |        |                         |
|--------|--------|-------------------------|
| handle | Output | MD5 handler (work area) |
|--------|--------|-------------------------|

**Return Values**

|               |                    |
|---------------|--------------------|
| TSIP_SUCCESS: | Normal termination |
|---------------|--------------------|

**Description**

The R\_TSIP\_Md5Init() function performs preparations for the execution of hash calculation and writes the result to the first parameter, handle. The parameter handle is used subsequently as a parameter by the R\_TSIP\_Md5Update() and R\_TSIP\_Md5Final() functions.

**Reentrancy**

Not supported.

---

### 4.2.8.5 R\_TSIP\_Md5Update

---

**Format**

```
#include "r_tsip_rx_if.h"
e_tsip_err_t R_TSIP_Md5Update(
 tsip_sha_md5_handle_t *handle,
 uint8_t *message,
 uint32_t message_length
)
```

**Parameters**

|                |              |                         |
|----------------|--------------|-------------------------|
| handle         | Input/output | MD5 handler (work area) |
| message        | Input        | Message area            |
| message_length | Input        | Byte length of message  |

**Return Values**

|                             |                                                                                                                                       |
|-----------------------------|---------------------------------------------------------------------------------------------------------------------------------------|
| TSIP_SUCCESS:               | Normal termination                                                                                                                    |
| TSIP_ERR_RESOURCE_CONFLICT: | Occurrence of resource conflict because a hardware resource needed by the processing routine was in use by another processing routine |
| TSIP_ERR_PARAMETER:         | Invalid handle input                                                                                                                  |
| TSIP_ERR_PROHIBIT_FUNCTION: | Invalid function called                                                                                                               |

**Description**

Using the handle specified by the first parameter, handle, the R\_TSIP\_Md5Update() function calculates a hash value based on the second parameter, message, and the third parameter, message\_length, and writes the ongoing status to the first parameter, handle (the value of which can be fetched using R\_TSIP\_GetCurrentHashDigestValue()). After message input completes, call R\_TSIP\_Md5Final().

**Reentrancy**

Not supported.



---

#### 4.2.8.6 R\_TSIP\_Md5Final

---

**Format**

```
#include "r_tsip_rx_if.h"
e_tsip_err_t R_TSIP_Sha1Final(
 tsip_sha_md5_handle_t *handle,
 uint8_t *digest,
 uint32_t *digest_length
)
```

**Parameters**

|               |        |                                     |
|---------------|--------|-------------------------------------|
| handle        | Input  | MD5 handler (work area)             |
| digest        | Output | Hash data area                      |
| digest_length | Output | Byte length of hash data (16 bytes) |

**Return Values**

|                             |                                                                                                                                       |
|-----------------------------|---------------------------------------------------------------------------------------------------------------------------------------|
| TSIP_SUCCESS:               | Normal termination                                                                                                                    |
| TSIP_ERR_RESOURCE_CONFLICT: | Occurrence of resource conflict because a hardware resource needed by the processing routine was in use by another processing routine |
| TSIP_ERR_PARAMETER:         | Invalid handle input                                                                                                                  |
| TSIP_ERR_PROHIBIT_FUNCTION: | Invalid function called                                                                                                               |

**Description**

Using the handle specified as the first parameter, handle, the R\_TSIP\_Md5Final() function writes the calculation result to the second parameter, digest, and writes the length of the calculation result to the third parameter, digest\_length.

**Reentrancy**

Not supported.

---

### 4.2.8.7 R\_TSIP\_GetCurrentHashDigestValue

---

**Format**

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

**Parameters**

|               |        |                                                                |
|---------------|--------|----------------------------------------------------------------|
| handle        | Input  | SHA or MD5 handler (work area)                                 |
| digest        | Output | Hash value for current input data area                         |
| digest_length | Output | Hash value for current input data length (16, 20, or 32 bytes) |

**Return Values**

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

**Description**

Using the handle specified as the parameter handle, this function outputs to the parameter digest the hash value for current input data after execution of an Update() function\*1 and outputs to the parameter digest\_length the data length.

Note: 1. R\_TSIP\_Sha1Update(), R\_TSIP\_Sha256Update(), or R\_TSIP\_Md5Update() function

**Reentrancy**

Not supported.

## 4.2.9 HMAC

### 4.2.9.1 R\_TSIP\_GenerateShaXXXHmacKeyIndex

#### Format

```
(1) #include "r_tsip_rx_if.h"
 e_tsip_err_t R_TSIP_GenerateSha1HmacKeyIndex (
 uint8_t *encrypted_provisioning_key,
 uint8_t *iv,
 uint8_t *encrypted_key,
 tsip_hmac_sha_key_index_t *key_index
)
(2) #include "r_tsip_rx_if.h"
 e_tsip_err_t R_TSIP_GenerateSha256HmacKeyIndex (
 uint8_t *encrypted_provisioning_key,
 uint8_t *iv,
 uint8_t *encrypted_key,
 tsip_hmac_sha_key_index_t *key_index
)
```

#### Parameters

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

#### Return Values

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

#### Description

This API outputs an SHA1-HMAC or SHA256-HMAC wrapped key.

Refer to 7.3.6, SHA-HMAC, for the format of the data encrypted using the UFPK input as encrypted\_key.

Refer to 3.7.1, Key Injection and Updating, for an explanation of encrypted\_provisioning\_key, iv, and encrypted\_key and how to use key\_index.

#### Reentrancy

Not supported.

### 4.2.9.2 R\_TSIP\_UpdateShaXXXHmacKeyIndex

#### Format

```
(1) #include "r_tsip_rx_if.h"
 e_tsip_err_t R_TSIP_UpdateSha1HmacKeyIndex (
 uint8_t *iv,
 uint8_t *encrypted_key,
 tsip_hmac_sha_key_index_t *key_index
)
(2) #include "r_tsip_rx_if.h"
 e_tsip_err_t R_TSIP_UpdateSha256HmacKeyIndex (
 uint8_t *iv,
 uint8_t *encrypted_key,
 tsip_hmac_sha_key_index_t *key_index
)
```

#### Parameters

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

#### Return Values

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

#### Description

This API updates a SHA-HMAC wrapped key.

Refer to 7.3.6, SHA-HMAC for the format of the data encrypted using the KUK input as encrypted\_key.

Refer to 3.7.1, Key Injection and Updating, for an explanation of encrypted\_provisioning\_key, iv, and encrypted\_key and how to use key\_index.

#### Reentrancy

Not supported.

### 4.2.9.3 R\_TSIP\_ShaXXXHmacGenerateInit

#### Format

- ```
(1) #include "r_tsip_rx_if.h"
    e_tsip_err_t R_TSIP_Sha1HmacGenerateInit(
        tsip_hmac_sha_handle_t *handle,
        tsip_hmac_sha_key_index_t *key_index
    )
(2) #include "r_tsip_rx_if.h"
    e_tsip_err_t R_TSIP_Sha256HmacGenerateInit(
        tsip_hmac_sha_handle_t *handle,
        tsip_hmac_sha_key_index_t *key_index
    )
```

Parameters

handle	Output	SHA-HMAC handler (work area)
key_index	Input	Wrapped key

Return Values

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

Description

The R_TSIP_ShaXXXHmacGenerateInit() function uses the second parameter, key_index, to perform preparations for the execution of SHA1-HMAC or SHA256-HMAC calculation and writes the result to the first parameter, handle. When using the TLS cooperation function, use the wrapped key generated by the R_TSIP_TlsGenerateSessionKey() function as key_index. The parameter handle is used subsequently as a parameter by the R_TSIP_ShaXXXHmacGenerateUpdate() and R_TSIP_ShaXXXHmacGenerateFinal() functions.

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

Reentrancy

Not supported.

4.2.9.4 R_TSIP_ShaXXXHmacGenerateUpdate

Format

```
(1) #include "r_tsip_rx_if.h"
    e_tsip_err_t R_TSIP_Sha1HmacGenerateUpdate(
        tsip_hmac_sha_handle_t *handle,
        uint8_t *message,
        uint32_t message_length
    )
(2) #include "r_tsip_rx_if.h"
    e_tsip_err_t R_TSIP_Sha256HmacGenerateUpdate(
        tsip_hmac_sha_handle_t *handle,
        uint8_t *message,
        uint32_t message_length
    )
```

Parameters

handle	Input/output	SHA-HMAC handler (work area)
message	Input	Message area
message_length	Input	Byte length of message

Return Values

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

Description

Using the handle specified by the first parameter, handle, the R_TSIP_ShaXXXHmacGenerateUpdate() function calculates a hash value based on the second parameter, message, and the third parameter, message_length, and writes the ongoing status to the first parameter, handle. After message input completes, call R_TSIP_ShaXXXHmacGenerateFinal().

Reentrancy

Not supported.

4.2.9.5 R_TSIP_ShaXXXHmacGenerateFinal

Format

- ```
(1) #include "r_tsip_rx_if.h"
 e_tsip_err_t R_TSIP_Sha1HmacGenerateFinal(
 tsip_hmac_sha_handle_t *handle,
 uint8_t *mac
)
(2) #include "r_tsip_rx_if.h"
 e_tsip_err_t R_TSIP_Sha256HmacGenerateFinal(
 tsip_hmac_sha_handle_t *handle,
 uint8_t *mac
)
```

**Parameters**

|        |        |                                                                   |
|--------|--------|-------------------------------------------------------------------|
| handle | Input  | SHA-HMAC handler (work area)                                      |
| mac    | Output | HMAC area<br>(1) SHA1-HMAC: 20 bytes<br>(2) SHA256-HMAC: 32 bytes |

**Return Values**

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

**Description**

The R\_TSIP\_ShaXXXHmacGenerateFinal() function uses the handle specified as the first parameter, handle, and writes the calculation result to the second parameter, mac.

**Reentrancy**

Not supported.

### 4.2.9.6 R\_TSIP\_ShaXXXHmacVerifyInit

#### Format

- ```
(1) #include "r_tsip_rx_if.h"
    e_tsip_err_t R_TSIP_Sha1HmacVerifyInit(
        tsip_hmac_sha_handle_t *handle,
        tsip_hmac_sha_key_index_t *key_index
    )
(2) #include "r_tsip_rx_if.h"
    e_tsip_err_t R_TSIP_Sha256HmacVerifyInit(
        tsip_hmac_sha_handle_t *handle,
        tsip_hmac_sha_key_index_t *key_index
    )
```

Parameters

handle	Output	SHA-HMAC handler (work area)
key_index	Input	Wrapped key

Return Values

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

Description

The R_TSIP_ShaXXXHmacVerifyInit() function uses the first parameter, key_index, to perform preparations for the execution of SHA1-HMAC or SHA256-HMAC calculation and writes the result to the first parameter, handle. When using the TLS cooperation function, use the wrapped key generated by the R_TSIP_TlsGenerateSessionKey() function as key_index. The parameter handle is used subsequently as a parameter by the R_TSIP_ShaXXXHmacVerifyUpdate() and R_TSIP_ShaXXXHmacVerifyFinal() functions.

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

Reentrancy

Not supported.

4.2.9.7 R_TSIP_ShaXXXHmacVerifyUpdate

Format

```
(1) #include "r_tsip_rx_if.h"
    e_tsip_err_t R_TSIP_Sha1HmacVerifyUpdate(
        tsip_hmac_sha_handle_t *handle,
        uint8_t *message,
        uint32_t message_length
    )
(2) #include "r_tsip_rx_if.h"
    e_tsip_err_t R_TSIP_Sha256HmacVerifyUpdate(
        tsip_hmac_sha_handle_t *handle,
        uint8_t *message,
        uint32_t message_length
    )
```

Parameters

handle	Input/output	SHA-HMAC handler (work area)
message	Input	Message area
message_length	Input	Byte length of message

Return Values

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

Description

Using the handle specified by the first parameter, handle, the R_TSIP_ShaXXXHmacVerifyUpdate() function calculates a hash value based on the second parameter, message, and the third parameter, message_length, and writes the ongoing status to the first parameter, handle. After message input completes, call R_TSIP_ShaXXXHmacVerifyFinal().

Reentrancy

Not supported.

4.2.9.8 R_TSIP_ShaXXXHmacVerifyFinal

Format

- ```
(1) #include "r_tsip_rx_if.h"
 e_tsip_err_t R_TSIP_Sha1HmacVerifyFinal(
 tsip_hmac_sha_handle_t *handle,
 uint8_t *mac,
 uint32_t mac_length
)

(2) #include "r_tsip_rx_if.h"
 e_tsip_err_t R_TSIP_Sha256HmacVerifyFinal(
 tsip_hmac_sha_handle_t *handle,
 uint8_t *mac,
 uint32_t mac_length
)
```

#### Parameters

|            |       |                              |
|------------|-------|------------------------------|
| handle     | Input | SHA-HMAC handler (work area) |
| mac        | Input | HMAC area                    |
| mac_length | Input | HMAC length                  |

#### Return Values

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

#### Description

Using the handle specified as the first parameter, handle, the R\_TSIP\_ShaXXXHmacVerifyFinal() function verifies the MAC value based on the second parameter, mac, and the third parameter, mac\_length. The value of mac\_length is specified in byte units. Input a value of 4 to 20 for SHA1-HMAC and 4 to 32 for SHA256-HMAC.

#### Reentrancy

Not supported.

## 4.2.10 DH

### 4.2.10.1 R\_TSIP\_Rsa2048DhKeyAgreement

#### Format

```
#include "r_tsip_rx_if.h"
e_tsip_err_t R_TSIP_Rsa2048DhKeyAgreement(
 tsip_aes_key_index_t *key_index,
 tsip_rsa2048_private_key_index_t *sender_private_key_index,
 uint8_t *message,
 uint8_t *receiver_modulus,
 uint8_t *sender_modulus
)
```

#### Parameters

|                          |        |                                                                                                                                          |
|--------------------------|--------|------------------------------------------------------------------------------------------------------------------------------------------|
| key_index                | Input  | Wrapped key for AES-128 CMAC operation                                                                                                   |
| sender_private_key_index | Input  | Secret wrapped key for DH operation<br>The secret key d included in the secret wrapped key is decrypted and used internally by the TSIP. |
| message                  | Input  | Message (2048 bits)<br>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<br>2048-bit modular exponentiation result    128 bits                     |
| sender_modulus           | Output | Modular exponentiation result calculated by the sender + MAC<br>2048-bit modular exponentiation result    128 bits                       |

#### Return Values

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

#### Description

Performs a DH operation using RSA-2048.

The sender is the TSIP and the receiver is the other key exchange party.

#### Reentrancy

Not supported.

## 4.2.11 ECDH

### 4.2.11.1 R\_TSIP\_EcdhP256Init

#### Format

```
#include "r_tsip_rx_if.h"
e_tsip_err_t R_TSIP_EcdhP256Init (
 tsip_ecdh_handle_t *handle,
 uint32_t key_type,
 uint32_t use_key_id
)
```

#### Parameters

|             |        |                                         |
|-------------|--------|-----------------------------------------|
| handle      | Output | ECDH handler (work area)                |
| key_type    | Input  | Key exchange type   0: ECDHE<br>1: ECDH |
| user_key_id | Input  | 0: key_id not used<br>1: key_id used    |

#### Return Values

|                     |                    |
|---------------------|--------------------|
| TSIP_SUCCESS:       | Normal termination |
| TSIP_ERR_PARAMETER: | Invalid input data |

#### Description

The R\_TSIP\_EcdhP256Init() function performs preparations for the execution of ECDH key exchange calculation and writes the result to the first parameter, handle. The parameter handle is used subsequently as a parameter by the R\_TSIP\_EcdhP256ReadPublicKey(), R\_TSIP\_EcdhP256MakePublicKey(), R\_TSIP\_EcdhP256CalculateSharedSecretIndex(), and R\_TSIP\_EcdhP256KeyDerivation() functions.

Use the second parameter, key\_type, to select the type of ECDH key exchange. For ECDHE, the R\_TSIP\_EcdhP256MakePublicKey() function uses the TSIP's random number generation functionality to generate an ECC P-256 key pair. For ECDH, keys injected beforehand are used for key exchange.

Input 1 as the third parameter, use\_key\_id, to use key\_id when key exchange is performed. The key\_id parameter is for applications conforming to the DLMS/COSEM standard for smart meters.

Refer to 3.7.1, Key Injection and Updating, for an explanation of how to generate key\_index.

#### Reentrancy

Not supported.

### 4.2.11.2 R\_TSIP\_EcdhP256ReadPublicKey

#### Format

```
#include "r_tsip_rx_if.h"
e_tsip_err_t R_TSIP_EcdhP256ReadPublicKey (
 tsip_ecdh_handle_t *handle,
 tsip_ecc_public_key_index_t *public_key_index,
 uint8_t *public_key_data,
 tsip_ecdsa_byte_data_t *signature,
 tsip_ecc_public_key_index_t *key_index
)
```

#### Parameters

|                  |              |                                                                                                                     |
|------------------|--------------|---------------------------------------------------------------------------------------------------------------------|
| handle           | Input/output | ECDH handler (work area)                                                                                            |
| public_key_index | Input        | Public wrapped key for signature verification                                                                       |
| public_key_data  | Input        | key_id not used    ECC P-256 public key (512 bits)<br>key_id used        key_id (8 bits)    public key s (512 bits) |
| signature        | Input        | ECDSA P-256 signature of public_key_data                                                                            |
| key_index        | Output       | Wrapped key of public_key_data                                                                                      |

#### Return Values

|                             |                                                                                                                                       |
|-----------------------------|---------------------------------------------------------------------------------------------------------------------------------------|
| TSIP_SUCCESS:               | Normal termination                                                                                                                    |
| TSIP_ERR_RESOURCE_CONFLICT: | Occurrence of resource conflict because a hardware resource needed by the processing routine was in use by another processing routine |
| TSIP_ERR_KEY_SET:           | Invalid wrapped key input                                                                                                             |
| TSIP_ERR_FAIL:              | Internal error, or signature verification failure                                                                                     |
| TSIP_ERR_PARAMETER:         | Invalid handle input                                                                                                                  |
| TSIP_ERR_PROHIBIT_FUNCTION: | Invalid function called                                                                                                               |

#### Description

The R\_TSIP\_EcdhP256ReadPublicKey() function verifies the signature of the ECC P-256 public key of the other ECDH key exchange party. If the signature is correct, it outputs the public\_key\_data wrapped key as the fifth parameter.

The first parameter, handle, is used subsequently as a parameter by the R\_TSIP\_EcdhP256CalculateSharedSecretIndex() function.

The R\_TSIP\_EcdhP256CalculateSharedSecretIndex() function uses key\_index as input to calculate Z.

Refer to 3.7.1, Key Injection and Updating, for an explanation of how to generate key\_index.

#### Reentrancy

Not supported.

### 4.2.11.3 R\_TSIP\_EcdhP256MakePublicKey

#### Format

```
#include "r_tsip_rx_if.h"
e_tsip_err_t R_TSIP_EcdhP256MakePublicKey (
 tsip_ecdh_handle_t *handle,
 tsip_ecc_public_key_index_t *public_key_index,
 tsip_ecc_private_key_index_t *private_key_index,
 uint8_t *public_key,
 tsip_ecdsa_byte_data_t *signature,
 tsip_ecc_private_key_index_t *key_index
)
```

#### Parameters

| Parameter            | Direction    | Description                                                                                                                                                                      |
|----------------------|--------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| handle               | Input/output | ECDH handler (work area)<br>When using key_id, input handle->key_id after running R_TSIP_EcdhP256Init().                                                                         |
| public_key_index     | Input        | For ECDHE, input a null pointer.<br>For ECDH, input an ECC P-256 public wrapped key.                                                                                             |
| private_key_index    | Input        | ECC P-256 secret key for signature generation                                                                                                                                    |
| public_key           | Output       | User public key (512 bits) for key exchange<br>When using key_id, key_id (8 bits)    public key (512 bits)    0 padding (24 bits)                                                |
| signature<br>->pdata | Output       | Signature text storage destination information<br>: Specifies pointer to array containing signature text.<br>Signature format: signature r (256 bits)    signature s (256 bits)" |
| ->data_length        |              | : Data length (byte units)                                                                                                                                                       |
| key_index            | Output       | For ECDHE, a secret wrapped key generated from a random number.<br>Not output for ECDH.                                                                                          |

#### Return Values

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

**Description**

The R\_TSIP\_EcdhP256MakePublicKey() function generates an ephemeral key pair and calculates a signature using a key that is either generated or input. The generated signature is for applications conforming to the DLMS/COSEM standard for smart meters.

If ECDHE is specified by the key\_type parameter of the R\_TSIP\_EcdhP256Init() function, the TSIP's random number generation functionality is used to generate an ECC P-256 key pair. The public key is output to public\_key and the secret key is output to key\_index.

If ECDH is specified by the key\_type parameter of the R\_TSIP\_EcdhP256Init() function, the public key input as public\_key\_index is output to public\_key and nothing is output to key\_index.

The value of the first parameter, handle, is used subsequently as a parameter by the R\_TSIP\_EcdhP256CalculateSharedSecretIndex() function.

The R\_TSIP\_EcdhP256CalculateSharedSecretIndex() function uses key\_index as input to calculate Z.

Refer to 3.7.1, Key Injection and Updating, for an explanation of how to generate key\_index.

**Reentrancy**

Not supported.

#### 4.2.11.4 R\_TSIP\_EcdhP256CalculateSharedSecretIndex

##### Format

```
#include "r_tsip_rx_if.h"
e_tsip_err_t R_TSIP_EcdhP256CalculateSharedSecretIndex (
 tsip_ecdh_handle_t *handle,
 tsip_ecc_public_key_index_t *public_key_index,
 tsip_ecc_private_key_index_t *private_key_index,
 tsip_ecdh_key_index_t *shared_secret_index
)
```

##### Parameters

|                     |              |                                                                                   |
|---------------------|--------------|-----------------------------------------------------------------------------------|
| handle              | Input/output | ECDH handler (work area)                                                          |
| public_key_index    | Input        | Public wrapped key whose signature was verified by R_TSIP_EcdhP256ReadPublicKey() |
| private_key_index   | Input        | Secret wrapped key                                                                |
| shared_secret_index | Output       | Wrapped key of shared secret Z calculated during ECDH key exchange                |

##### Return Values

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

##### Description

The R\_TSIP\_EcdhP256CalculateSharedSecretIndex() function uses the ECDH key exchange algorithm to output the wrapped key of the shared secret Z derived from the public key of the other key exchange party and your own secret key.

As the second parameter, public\_key\_index, input the key\_index, public wrapped key whose signature was verified by R\_TSIP\_EcdhP256ReadPublicKey() function.

If the value of the key\_type parameter of R\_TSIP\_EcdhP256Init() function is 0, input the key\_index, secret wrapped key generated from a random number by R\_TSIP\_EcdhP256MakePublicKey() function as the third parameter, private\_key\_index, and if the value of the key\_type is other than 0, input the secret wrapped key corresponding to the second parameter of R\_TSIP\_EcdhP256MakePublicKey() as the third parameter, private\_key\_index.

The R\_TSIP\_EcdhP256KeyDerivation() function subsequently uses shared\_secret\_index as key material for outputting the wrapped key.

Refer to 3.7.1, Key Injection and Updating, for an explanation of how to generate key\_index.

##### Reentrancy

Not supported.



### 4.2.11.5 R\_TSIP\_EcdhP256KeyDerivation

#### Format

```
#include "r_tsip_rx_if.h"
e_tsip_err_t R_TSIP_EcdhP256KeyDerivation (
 tsip_ecdh_handle_t *handle,
 tsip_ecdh_key_index_t *shared_secret_index,
 uint32_t key_type,
 uint32_t kdf_type,
 uint8_t *other_info,
 uint32_t other_info_length,
 tsip_hmac_sha_key_index_t *salt_key_index,
 tsip_aes_key_index_t *key_index
)
```

#### Parameters

|                     |              |                                                                                                                                                                                                                                                                                                         |
|---------------------|--------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| handle              | Input/output | ECDH handler (work area)                                                                                                                                                                                                                                                                                |
| shared_secret_index | Input        | Z wrapped key calculated by R_TSIP_EcdhP256CalculateSharedSecretIndex                                                                                                                                                                                                                                   |
| key_type            | Input        | Derived key type<br>0: AES-128<br>1: AES-256<br>2: SHA256-HMAC                                                                                                                                                                                                                                          |
| kdf_type            | Input        | Algorithm used for key derivation calculation<br>0: SHA256<br>1: SHA256-HMAC                                                                                                                                                                                                                            |
| other_info          | Input        | Additional data used for key derivation calculation<br>AlgorithmID    PartyUInfo    PartyVInf                                                                                                                                                                                                           |
| other_info_length   | Input        | Byte length of other_info (147 or fewer byte units)                                                                                                                                                                                                                                                     |
| salt_key_index      | Input        | Salt wrapped key (Input NULL when kdf_type is 0.)                                                                                                                                                                                                                                                       |
| key_index           | Output       | Wrapped key index corresponding to key_type<br>When the value of key_type is 2, an SHA256-HMAC wrapped key is output. The value of key_index can be specified by casting the start address of the area reserved beforehand by the tsip_hmac_sha_key_index_t type with the (tsip_aes_key_index_t*) type. |

#### Return Values

|                             |                                                                                                                                       |
|-----------------------------|---------------------------------------------------------------------------------------------------------------------------------------|
| TSIP_SUCCESS:               | Normal termination                                                                                                                    |
| TSIP_ERR_RESOURCE_CONFLICT: | Occurrence of resource conflict because a hardware resource needed by the processing routine was in use by another processing routine |
| TSIP_ERR_KEY_SET:           | Invalid wrapped key input                                                                                                             |
| TSIP_ERR_PARAMETER:         | Invalid handle input                                                                                                                  |
| TSIP_ERR_PROHIBIT_FUNCTION: | Invalid function called                                                                                                               |

#### Description

The R\_TSIP\_EcdhP256KeyDerivation() function uses the shared secret Z (shared\_secret\_index) calculated by the R\_TSIP\_EcdhP256CalculateSharedSecretIndex() function as key material to derive the wrapped key specified by the third parameter, key\_type. The key derivation algorithm is one-step key derivation as defined in NIST SP800-56C. Any of AES-128, AES-256 or SHA-256 HMAC is specified by the fourth parameter, kdf\_type. When SHA-256 HMAC is specified, the wrapped key output by the R\_TSIP\_GenerateSha256HmacKeyIndex() function or R\_TSIP\_UpdateSha256HmacKeyIndex() function is specified as the seventh parameter, salt\_key\_index.

Enter a fixed value for deriving a key shared with the other key exchange party as the fifth parameter, other\_info.

A key index corresponding to key\_type is output as the eighth parameter, key\_index. The correspondences between the types of derived key indexes and the functions with which they can be used as listed below.

| Derived Wrapped Key | Compatible Functions                                              |
|---------------------|-------------------------------------------------------------------|
| AES-128             | All AES-128 Init functions and R_TSIP_Aes128KeyUnwrap()           |
| AES-256             | All AES-256 Init functions and R_TSIP_Aes256KeyUnwrap()           |
| SHA256-HMAC         | R_TSIP_Sha256HmacGenerateInit() and R_TSIP_Sha256HmacVerifyInit() |

### Reentrancy

Not supported.

---

### 4.2.11.6 R\_TSIP\_EcdheP512KeyAgreement

---

**Format**

```
#include "r_tsip_rx_if.h"
e_tsip_err_t R_TSIP_EcdheP512KeyAgreement(
 tsip_aes_key_index_t *key_index,
 uint8_t *receiver_public_key,
 uint8_t *sender_public_key
)
```

**Parameters**

|                     |        |                                                                            |
|---------------------|--------|----------------------------------------------------------------------------|
| key_index           | Input  | Wrapped key for AES-128 CMAC operation                                     |
| receiver_public_key | Input  | Brainpool P512r1 public key of receiver<br>Q (1024 bits)    MAC (128 bits) |
| sender_public_key   | Output | Brainpool P512r1 public key of sender<br>Q (1024 bits)    MAC (128 bits)   |

**Return Values**

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

**Description**

Performs an ECDHE operation after generation of a key pair using Brainpool P512r1.

The sender is the TSIP and the receiver is the other key exchange party.

**Reentrancy**

Not supported.

## 4.2.12 Key Wrap

### 4.2.12.1 R\_TSIP\_AesXXXKeyWrap

#### Format

```
(1) #include "r_tsip_rx_if.h"
 e_tsip_err_t R_TSIP_Aes128KeyWrap (
 tsip_aes_key_index_t *wrap_key_index,
 uint32_t target_key_type,
 tsip_aes_key_index_t *target_key_index,
 uint32_t *wrapped_key
)
(2) #include "r_tsip_rx_if.h"
 e_tsip_err_t R_TSIP_Aes256KeyWrap (
 tsip_aes_key_index_t *wrap_key_index,
 uint32_t target_key_type,
 tsip_aes_key_index_t *target_key_index,
 uint32_t *wrapped_key
)
```

#### Parameters

|                  |        |                                                                                                        |
|------------------|--------|--------------------------------------------------------------------------------------------------------|
| wrap_key_index   | Input  | (1) AES-128 wrapped key used for wrapping<br>(2) AES-256 wrapped key used for wrapping                 |
| target_key_type  | Input  | Selects key to be wrapped.<br>0 (R_TSIP_KEYWRAP_AES128): AES-128<br>2 (R_TSIP_KEYWRAP_AES256): AES-256 |
| target_key_index | Input  | Wrapped key to be wrapped<br>target_key_type 0: 13 word size<br>target_key_type 2: 17 word size        |
| wrapped_key      | Output | Key which is wrapped<br>target_key_type 0: 6 word size<br>target_key_type 2: 10 word size              |

#### Return Values

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

#### Description

The R\_TSIP\_AesXXXKeyWrap() function uses the first parameter, wrap\_key\_index, to wrap target\_key\_index, which is input as the third parameter. The wrapped key is written to the fourth parameter, wrapped\_key. The processing conforms to the RFC 3394 wrapping algorithm. Use the second parameter, target\_key\_type, to select the key to be wrapped.

Use R\_TSIP\_Aes128KeyWrap() when the key length used for wrapping is 128 bits, and use R\_TSIP\_Aes256KeyWrap() when the key length is 256 bits.

**Reentrancy**

Not supported.

### 4.2.12.2 R\_TSIP\_AesXXXKeyUnwrap

#### Format

```
(1) #include "r_tsip_rx_if.h"
 e_tsip_err_t R_TSIP_Aes128KeyUnwrap (
 tsip_aes_key_index_t *wrap_key_index,
 uint32_t target_key_type,
 uint32_t *wrapped_key,
 tsip_aes_key_index_t *target_key_index
)
(2) #include "r_tsip_rx_if.h"
 e_tsip_err_t R_TSIP_Aes256KeyUnwrap (
 tsip_aes_key_index_t *wrap_key_index,
 uint32_t target_key_type,
 uint32_t *wrapped_key,
 tsip_aes_key_index_t *target_key_index
)
```

#### Parameters

|                  |        |                                                                                                          |
|------------------|--------|----------------------------------------------------------------------------------------------------------|
| wrap_key_index   | Input  | (1) AES-128 wrapped key used for unwrapping<br>(2) AES-256 wrapped key used for unwrapping               |
| target_key_type  | Input  | Selects key to be unwrapped.<br>0 (R_TSIP_KEYWRAP_AES128): AES-128<br>2 (R_TSIP_KEYWRAP_AES256): AES-256 |
| wrapped_key      | Input  | Key which is wrapped<br>target_key_type 0: 6 word size<br>target_key_type 2: 10 word size                |
| target_key_index | Output | Unwrapped Key<br>target_key_type 0: 13 word size<br>target_key_type 2: 17 word size                      |

#### Return Values

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

#### Description

The R\_TSIP\_AesXXXKeyUnwrap() function uses the first parameter, wrap\_key\_index, to unwrap wrapped\_key, which is input as the third parameter. The unwrapped key is written to the fourth parameter, target\_key\_index. The processing conforms to the RFC 3394 unwrapping algorithm. Use the second parameter, target\_key\_type, to select the key to be unwrapped.

Use R\_TSIP\_Aes128KeyUnwrap() when the key length used for unwrapping is 128 bits, and use R\_TSIP\_Aes256KeyUnwrap() when the key length is 256 bits.

**Reentrancy**

Not supported.

### 4.2.13 TLS (Common to TLS 1.2 and TLS 1.3)

In this section, (1) means for Client and (2) means for Server in the columns of Format or other.

#### 4.2.13.1 R\_TSIP\_GenerateTlsXXRsaPublicKeyIndex

##### Format

```
(1) #include "r_tsip_rx_if.h"
 e_tsip_err_t R_TSIP_GenerateTlsRsaPublicKeyIndex(
 uint8_t *encrypted_provisioning_key,
 uint8_t *iv,
 uint8_t *encrypted_key,
 tsip_tls_ca_certification_public_key_index_t *key_index
)
(2) #include "r_tsip_rx_if.h"
 e_tsip_err_t R_TSIP_GenerateTlsSVRsaPublicKeyIndex(
 uint8_t *encrypted_provisioning_key,
 uint8_t *iv,
 uint8_t *encrypted_key,
 tsip_tls_ca_certification_public_key_index_t *key_index
)
```

##### Parameters

|                            |        |                                                                                                                                                                                                                                                                                                                         |
|----------------------------|--------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| encrypted_provisioning_key | Input  | W-UFPK                                                                                                                                                                                                                                                                                                                  |
| iv                         | Input  | Initialization vector when generating encrypted_key                                                                                                                                                                                                                                                                     |
| encrypted_key              | Input  | Encrypted key encrypted using UFPK                                                                                                                                                                                                                                                                                      |
| key_index                  | Output | 2048-bit RSA public wrapped key for use by TLS cooperation function<br>Use this value as the key_index parameter input to R_TSIP_TlsRegisterCaCertificationPublicKeyIndex<br>When the wrapped key for client is generated with (1), it is possible to use this value as the key_index_1 parameter input to R_TSIP_Open. |

##### Return Values

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

##### Description

This API outputs an RSA 2048-bit public wrapped key for use by the TLS cooperation function. Refer to 7.3.4.2(1) for the format of the data encrypted using the UFPK input as encrypted\_key. Ensure that the areas allocated for encrypted\_key and key\_index do not overlap.

Refer to 3.7.1, Key Injection and Updating, for an explanation of encrypted\_provisioning\_key, iv, and encrypted\_key and how to use key\_index.



**Reentrancy**

Not supported.

### 4.2.13.2 R\_TSIP\_UpdateTlsXXRsaPublicKeyIndex

#### Format

```
(1) #include "r_tsip_rx_if.h"
 e_tsip_err_t R_TSIP_UpdateTlsRsaPublicKeyKeyIndex(
 uint8_t *iv,
 uint8_t *encrypted_key,
 tsip_tls_ca_public_key_index_t *key_index
)
(2) #include "r_tsip_rx_if.h"
 e_tsip_err_t R_TSIP_UpdateTlsSVRsaPublicKeyKeyIndex(
 uint8_t *iv,
 uint8_t *encrypted_key,
 tsip_tls_ca_public_key_index_t *key_index
)
```

#### Parameters

| Parameter     | Direction | Description                                                                                                                                                                                                                                                                                                             |
|---------------|-----------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| iv            | Input     | Initialization vector when generating encrypted_key                                                                                                                                                                                                                                                                     |
| encrypted_key | Input     | Encrypted key encrypted using KUK                                                                                                                                                                                                                                                                                       |
| key_index     | Output    | 2048-bit RSA public wrapped key for use by TLS cooperation function<br>Use this value as the key_index parameter input to R_TSIP_TlsRegistaeCaCertificationPublicKeyIndex<br>When the wrapped key for client is generated with (1), it is possible to use this value as the key_index_1 parameter input to R_TSIP_Open. |

#### Return Values

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

#### Description

This API updates a 2048-bit RSA public wrapped key for use by the TLS cooperation function.

Refer to 7.3.4.2(1) for the format of the data encrypted using the KUK input as encrypted\_key.

Ensure that the areas allocated for encrypted\_key and key\_index do not overlap.

Refer to 3.7.1, Key Injection and Updating, for an explanation of encrypted\_provisioning\_key, iv, and encrypted\_key and how to use key\_index.

#### Reentrancy

Not supported.

### 4.2.13.3 R\_TSIP\_TlsRegisterCaCertificationPublicKeyIndex

#### Format

```
#include "r_tsip_rx_if.h"
e_tsip_err_t R_TSIP_TlsRegisterCaCertificationPublicKeyIndex(
 e_tsip_tls_mode_t mode,
 tsip_tls_ca_certification_public_key_index_t *key_index
)
```

#### Parameters

|           |       |                                                                                                                                                                                                                                                                   |
|-----------|-------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| mode      | Input | Entity type to use registering public key<br>TSIP_TLS_MODE_CLIENT: Client,<br>TSIP_TLS_MODE_SERVER: Server                                                                                                                                                        |
| key_index | Input | 2048-bit RSA public wrapped key for use by TLS cooperation function<br>Use the value of key_index output by R_TSIP_GenerateTlsRsaPublicKeyIndex, R_TSIP_UpdateTlsRsaPublicKeyIndex, R_TSIP_GenerateTlsSVRsaPublicKeyIndex or R_TSIP_UpdateTlsSVRsaPublicKeyIndex. |

#### Return Values

|                   |                           |
|-------------------|---------------------------|
| TSIP_SUCCESS:     | Normal termination        |
| TSIP_ERR_KEY_SET: | Invalid wrapped key input |

#### Description

This API registers the public key for use by the TLS cooperation function.

The public key for Client can be registered by using R\_TSIP\_Open() function.

#### Reentrancy

Not supported.

#### 4.2.13.4 R\_TSIP\_TlsXXRootCertificateVerification

##### Format

- ```
(1) #include "r_tsip_rx_if.h"
    e_tsip_err_t R_TSIP_TlsRootCertificateVerification(
        uint32_t *public_key_type,
        uint8_t *certificate,
        uint32_t certificate_length,
        uint32_t public_key_n_start_position,
        uint32_t public_key_n_end_position,
        uint32_t public_key_e_start_position,
        uint32_t public_key_e_end_position,
        uint8_t *signature,
        uint32_t *encrypted_root_public_key
    )

(2) #include "r_tsip_rx_if.h"
    e_tsip_err_t R_TSIP_TlsSVRootCertificateVerification(
        uint32_t *public_key_type,
        uint8_t *certificate,
        uint32_t certificate_length,
        uint32_t public_key_n_start_position,
        uint32_t public_key_n_end_position,
        uint32_t public_key_e_start_position,
        uint32_t public_key_e_end_position,
        uint8_t *signature,
        uint32_t *encrypted_root_public_key
    )
```

Parameters

public_key_type	Input	Public key type included in certificate 0: RSA 2048-bit, 2: ECC P-256
certificate	Input	Root CA certificate bundle (DER format)
certificate_length	Input	Byte length of root CA certificate bundle
public_key_n_start_position	Input	Public key start byte position relative to address specified by parameter certificate public_key_type 0: n, 2: Qx
public_key_n_end_position	Input	Public key end byte position relative to address specified by parameter certificate public_key_type 0: n, 2: Qx
public_key_e_start_position	Input	Public key start byte position relative to address specified by parameter certificate public_key_type 0: e, 2: Qy
public_key_e_end_position	Input	Public key end byte position relative to address specified by parameter certificate public_key_type 0: e, 2: Qy
signature	Input	Signature data for root CA certificate bundle The signature data size is 256 bytes. The signature format is "RSA2048 PSS with SHA256."

encrypted_root_public_key	Output	Encrypted ECDSA P256 or RSA-2048 public key Use this value as the encrypted_input_public_key parameter input to R_TSIP_TlsXXCertificateVerification or R_TSIP_TlsXXCertificateVerificationExtension. If public_key_type is 0, 560 bytes are output, and if public_key_type is 2, 96 bytes are output.
---------------------------	--------	---

Return Values

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

Description

This API verifies a root CA certificate bundle for use by the TLS cooperation function.

Reentrancy

Not supported.

4.2.13.5 R_TSIP_TlsXXCertificateVerification

Format

```
(1) #include "r_tsip_rx_if.h"
    e_tsip_err_t R_TSIP_TlsCertificateVerification(
        uint32_t *public_key_type,
        uint32_t *encrypted_input_public_key,
        uint8_t *certificate,
        uint32_t certificate_length,
        uint8_t *signature,
        uint32_t public_key_n_start_position,
        uint32_t public_key_n_end_position,
        uint32_t public_key_e_start_position,
        uint32_t public_key_e_end_position,
        uint32_t *encrypted_output_public_key
    )
(2) #include "r_tsip_rx_if.h"
    e_tsip_err_t R_TSIP_TlsSVCertificateVerification(
        uint32_t *public_key_type,
        uint32_t *encrypted_input_public_key,
        uint8_t *certificate,
        uint32_t certificate_length,
        uint8_t *signature,
        uint32_t public_key_n_start_position,
        uint32_t public_key_n_end_position,
        uint32_t public_key_e_start_position,
        uint32_t public_key_e_end_position,
        uint32_t *encrypted_output_public_key
    )
```

Parameters

public_key_type	Input	Public key type included in certificate 0: RSA 2048-bit (for sha256WithRsaEncryption), 1: RSA 4096-bit (for sha256WithRsaEncryption), 2: ECC P-256 (for ecdsa-with-SHA256), 3: RSA 2048-bit (for RSASSA-PSS)
encrypted_input_public_key	Input	Encrypted public key Use the value of encrypted_root_public_key output by R_TSIP_TlsXXRootCertificateVerification or the value of encrypted_output_public_key output by R_TSIP_TlsXXCertificateVerification or R_TSIP_TlsXXCertificateVerificationExtension. Data size public_key_type 0, 1, or 3: 140 words (560 bytes), 2: 24 words (96 bytes)
certificate	Input	Certificate bundle (DER format)
certificate_length	Input	Byte length of certificate bundle

signature	Input	Signature data for certificate bundle public_key_type: 0 The data size is 256 bytes. Signature algorithm is sha256WithRSAEncryption public_key_type: 1 The data size is 512 bytes. Signature algorithm is sha256WithRSAEncryption public_key_type: 2 The data size is 64 bytes “r (256 bits) s (256 bits)” Signature algorithm is ecdsa-with-SHA256 public_key_type: 3 The data size is 256 bytes. Signature algorithm is RSASSA-PSS {sha256, mgf1SHA256, 0x20, trailerFieldBC}
public_key_n_start_position	Input	Public key start byte position relative to address specified by parameter certificate public_key_type 0, 1, or 3: n, 2: Qx
public_key_n_end_position	Input	Public key end byte position relative to address specified by parameter certificate public_key_type 0, 1, or 3: n, 2: Qx
public_key_e_start_position	Input	Public key start byte position relative to address specified by parameter certificate public_key_type 0, 1, or 3: e, 2: Qy
public_key_e_end_position	Input	Public key end byte position relative to address specified by parameter certificate public_key_type 0, 1, or 3: e, 2: Qy
encrypted_output_public_key	Output	Encrypted public key Use this value as the encrypted_input_public_key parameter input to R_TSIP_TIsXXCertificateVerification or R_TSIP_TIsXXCertificateVerificationExtension, or as the encrypted_public_key parameter input to R_TSIP_TIsEncryptPreMasterSecretWithRsa2048PublicKey or R_TSIP_TIsServersEphemeralEcdhPublicKeyRetrives. However, it can be used only with R_TSIP_TIsXXCertificateVerification or R_TSIP_TIsXXCertificateVerificationExtension when public_key_type = 1 is selected. Data size public_key_type 0, 1, or 3: 140 words (560 bytes), 2: 24 words (96 bytes)

Return Values

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

Description

This API verifies the signature of a server certificate or intermediate certificate used by the TLS cooperation function.

This API can be used for same purpose as the R_TSIP_TlsXXCertificateVerificationExtension() function, but make sure to use this function when the algorithm of the key used for signature verification is the same as the algorithm used to obtain the key from the certificate.

Reentrancy

Not supported.

4.2.13.6 R_TSIP_TlsXXCertificateVerificationExtension

Format

- ```
(1) #include "r_tsip_rx_if.h"
 e_tsip_err_t R_TSIP_TlsCertificateVerificationExtension(
 uint32_t *public_key_type,
 uint32_t *public_key_output_type,
 uint32_t *encrypted_input_public_key,
 uint8_t *certificate,
 uint32_t certificate_length,
 uint8_t *signature,
 uint32_t public_key_n_start_position,
 uint32_t public_key_n_end_position,
 uint32_t public_key_e_start_position,
 uint32_t public_key_e_end_position,
 uint32_t *encrypted_output_public_key
)

(2) #include "r_tsip_rx_if.h"
 e_tsip_err_t R_TSIP_TlsSVCertificateVerificationExtension(
 uint32_t *public_key_type,
 uint32_t *public_key_output_type,
 uint32_t *encrypted_input_public_key,
 uint8_t *certificate,
 uint32_t certificate_length,
 uint8_t *signature,
 uint32_t public_key_n_start_position,
 uint32_t public_key_n_end_position,
 uint32_t public_key_e_start_position,
 uint32_t public_key_e_end_position,
 uint32_t *encrypted_output_public_key
)
```

#### Parameters

|                            |       |                                                                                                                                                                                                                                                                                                                                                              |
|----------------------------|-------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| public_key_type            | Input | Type of public key included in input certificate<br>0: RSA 2048-bit (for sha256WithRsaEncryption),<br>1: RSA 4096-bit (for sha256WithRsaEncryption),<br>2: ECC P-256 (ecdsa-with-SHA256),<br>3: RSA 2048-bit (for RSASSA-PSS)                                                                                                                                |
| public_key_output_type     | Input | Type of public key output from certificate<br>0: RSA 2048-bit (for sha256WithRsaEncryption),<br>1: RSA 4096-bit (for sha256WithRsaEncryption),<br>2: ECC P-256 (ecdsa-with-SHA256),<br>3: RSA 2048-bit (for RSASSA-PSS)                                                                                                                                      |
| encrypted_input_public_key | Input | Encrypted public key<br>Use the value of encrypted_root_public_key output by R_TSIP_TlsXXRootCertificateVerification or the value of encrypted_output_public_key output by R_TSIP_TlsXXCertificateVerification or R_TSIP_TlsXXCertificateVerificationExtension.<br>Data size<br>public_key_type 0, 1, or 3: 140 words (560 bytes),<br>2: 24 words (96 bytes) |
| certificate                | Input | Certificate bundle (DER format)                                                                                                                                                                                                                                                                                                                              |

|                             |        |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
|-----------------------------|--------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| certificate_length          | Input  | Byte length of certificate bundle                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| signature                   | Input  | Signature data for certificate bundle<br>public_key_type: 0<br>The data size is 256 bytes.<br>Signature algorithm is sha256WithRSAEncryption<br>public_key_type: 1<br>The data size is 512 bytes<br>Signature algorithm is sha256WithRSAEncryption<br>public_key_type: 2<br>The data size is 64 bytes “r (256 bits)    s (256 bits)”<br>Signature algorithm is ecdsa-with-SHA256<br>public_key_type: 3<br>The data size is 256 bytes.<br>Signature algorithm is RSASSA-PSS<br>{sha256, mgf1SHA256, 0x20, trailerFieldBC}                                                                                |
| public_key_n_start_position | Input  | Public key start byte position relative to address specified by parameter certificate<br>public_key_type 0, 1, or 3: n, 2: Qx                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| public_key_n_end_position   | Input  | Public key end byte position relative to address specified by parameter certificate<br>public_key_type 0, 1, or 3: n, 2: Qx                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| public_key_e_start_position | Input  | Public key start byte position relative to address specified by parameter certificate<br>public_key_type 0, 1, or 3: e, 2: Qy                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| public_key_e_end_position   | Input  | Public key end byte position relative to address specified by parameter certificate<br>public_key_type 0, 1, or 3: e, 2: Qy                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| encrypted_output_public_key | Output | Encrypted public key<br>Use this value as the encrypted_input_public_key parameter input to R_TSIP_TIsXXCertificateVerification or R_TSIP_TIsXXCertificateVerificationExtension, or as the encrypted_public_key parameter input to R_TSIP_TIsEncryptPreMasterSecretWithRsa2048PublicKey or R_TSIP_TIsServersEphemeralEcdhPublicKeyRetrives.<br>However, it can be used only with R_TSIP_TIsXXCertificateVerification or R_TSIP_TIsXXCertificateVerificationExtension when public_key_type = 1 is selected.<br>Data size<br>public_key_type 0, 1, or 3: 140 words (560 bytes),<br>2: 24 words (96 bytes) |

### Return Values

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

**Description**

This API verifies the signature of a server certificate or intermediate certificate used by the TLS cooperation function.

This API can be used for same purpose as the R\_TSIP\_TlsXXCertificateVerification() function, but make sure to use this function when the algorithm of the key used for signature verification is different from the algorithm used to obtain the key from the certificate.

**Reentrancy**

Not supported.

**4.2.14 TLS (TLS 1.2)**

In this section, (1) means for Client and (2) means for Server in the columns of Format or other.

**4.2.14.1 R\_TSIP\_TlsGeneratePreMasterSecret****Format**

```
#include "r_tsip_rx_if.h"
e_tsip_err_t R_TSIP_TlsGeneratePreMasterSecret(
 uint32_t *tsip_pre_master_secret
)
```

**Parameters**

|                        |        |                                                                                                                                                                                                                                                                                       |
|------------------------|--------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| tsip_pre_master_secret | Output | Encrypted ephemeral pre-master secret data<br>Use this value as the tsip_pre_master_secret parameter input to R_TSIP_TlsGenerateMasterSecret, R_TSIP_TlsEncryptPreMasterSecretWithRsa2048PublicKey, or R_TSIP_TlsGenerateExtendedMasterSecret. 20 words (80 bytes) of data is output. |
|------------------------|--------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

**Return Values**

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

**Description**

This API generates an encrypted ephemeral pre-master secret for use by the TLS cooperation function.

**Reentrancy**

Not supported.

---

#### 4.2.14.2 R\_TSIP\_TIsEncryptPreMasterSecretWithRsa2048PublicKey

---

**Format**

```
#include "r_tsip_rx_if.h"
e_tsip_err_t R_TSIP_TIsEncryptPreMasterSecretWithRsa2048PublicKey(
 uint32_t *encrypted_public_key,
 uint32_t *tsip_pre_master_secret,
 uint8_t *encrypted_pre_master_secret
)
```

**Parameters**

|                             |        |                                                                                                                                                                                                              |
|-----------------------------|--------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| encrypted_public_key        | Input  | Encrypted public key data<br>Use the value of encrypted_output_public_key output by R_TSIP_TIsCertificateVerification or R_TSIP_TIsCertificateVerificationExtension. The data size is 140 words (560 bytes). |
| tsip_pre_master_secret      | Input  | Encrypted ephemeral pre-master secret data, output by R_TSIP_TIsGeneratePreMasterSecret.                                                                                                                     |
| encrypted_pre_master_secret | Output | Pre-master secret data that was encrypted in RSA-2048 mode using public_key                                                                                                                                  |

**Return Values**

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

**Description**

This API encrypts a pre-master secret in RSA-2048 mode, using a public key from the input data, for use by the TLS cooperation function.

**Reentrancy**

Not supported.

### 4.2.14.3 R\_TSIP\_TlsSVGenerateServerRandom

#### Format

```
#include "r_tsip_rx_if.h"
e_tsip_err_t R_TSIP_TlsRegisterCaCertificationPublicKeyIndex(
 e_tsip_tls_version_t client_tls_version,
 e_tsip_tls_version_t server_tls_version,
 uint32_t gmx_unix_time,
 tsip_tls_sv_random_t *server_random
)
```

#### Parameters

|                    |        |                                                                                                                                              |
|--------------------|--------|----------------------------------------------------------------------------------------------------------------------------------------------|
| client_tls_version | Input  | Protocol version of TLS Client<br>TSIP_TLS_VERSION_12: TLS 1.2,<br>TSIP_TLS_VERSION_13: TLS 1.3                                              |
| server_tls_version | Input  | Protocol version of TLS Server<br>TSIP_TLS_VERSION_12: TLS 1.2,<br>TSIP_TLS_VERSION_13: TLS 1.3                                              |
| gmt_unix_time      | Input  | Current GMT Unit time with 32 bits format                                                                                                    |
| server_random      | Output | Random number value to use in ServerHello and its MAC                                                                                        |
| random             |        | Random number value to use in ServerHello<br>The data size is 8 words (32 bytes).                                                            |
| mac                |        | The MAC of random number value to use in ServerHello<br>The data size is 4 words (12 bytes). The data does not used<br>when TLS 1.3 is used. |

#### Return Values

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

#### Description

This API generates a random number value to use in ServerHello and its MAC used by the TLS cooperation function.

When the same random number value is used to ServerHello, TLS connection with same master secret can be conducted. To prevent such attack and ensure the security of the server system, this API provides a random number generation function which can be used safely. And, its MAC is added to prevent the attack to change the generated random number value on purpose to use following process. Please call this API from user of the driver every time and use the latest output when the ServerHello is generating.

#### Reentrancy

Not supported.

#### 4.2.14.4 R\_TSIP\_TlsSVDcryptPreMasterSecretWithRsa2048PrivateKey

##### Format

```
#include "r_tsip_rx_if.h"
e_tsip_err_t R_TSIP_TlsSVDcryptPreMasterSecretWithRsa2048PrivateKey(
 tsip_rsa2048_private_key_index_t *private_key_index,
 uint8_t *encrypted_pre_master_secret,
 uint32_t *tsip_pre_master_secret
)
```

##### Parameters

|                             |        |                                                                                                                                        |
|-----------------------------|--------|----------------------------------------------------------------------------------------------------------------------------------------|
| private_key_index           | Input  | Wrapped RSA private key type included in certificate<br>Use the value of key_index output by<br>R_TSIP_GenerateRsa2048PrivateKeyIndex. |
| encrypted_pre_master_secret | Input  | Pre-master secret data that was encrypted in RSA-2048<br>mode                                                                          |
| tsip_pre_master_secret      | Output | Encrypted ephemeral pre-master secret<br>The data size is 16 words (64 bytes).                                                         |

##### Return Values

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

##### Description

This API decrypts a pre-master secret in RSA-2048 mode, using a private key from the input data, for use by the TLS cooperation function.

##### Reentrancy

Not supported.

#### 4.2.14.5 R\_TSIP\_TlsXXGenerateMasterSecret

##### Format

```
(1) #include "r_tsip_rx_if.h"
 e_tsip_err_t R_TSIP_TlsGenerateMasterSecret(
 uint32_t select_cipher_suite,
 uint32_t *tsip_pre_master_secret,
 uint8_t *client_random,
 uint8_t *server_random,
 uint32_t *tsip_master_secret
)
(2) #include "r_tsip_rx_if.h"
 e_tsip_err_t R_TSIP_TlsSVGenerateMasterSecret(
 uint32_t select_cipher_suite,
 uint32_t *tsip_pre_master_secret,
 uint8_t *client_random,
 tsip_tls_sv_random_t *server_random,
 uint32_t *tsip_master_secret
)
```

##### Parameters

|                        |        |                                                                                                                                                                                                                                                                                                                                                                                                                       |
|------------------------|--------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| select_cipher_suite    | Input  | Cipher suite selection<br>0: R_TSIP_TLS_RSA_WITH_AES_128_CBC_SHA<br>1: R_TSIP_TLS_RSA_WITH_AES_256_CBC_SHA<br>2: R_TSIP_TLS_RSA_WITH_AES_128_CBC_SHA256<br>3: R_TSIP_TLS_RSA_WITH_AES_256_CBC_SHA256<br>4: R_TSIP_TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256<br>5: R_TSIP_TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256<br>6: R_TSIP_TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256<br>7: R_TSIP_TLS_ECDHE_RSSA_WITH_AES_128_GCM_SHA256 |
| tsip_pre_master_secret | Input  | Encrypted ephemeral pre-master secret data<br>(1)<br>Use the value of tsip_pre_master_secret output by R_TSIP_TlsGeneratePreMasterSecret or encrypted_pre_master_secret output by R_TSIP_TlsGeneratePreMasterSecretWithEccP256Key.<br>(2)<br>Use the value of tsip_pre_master_secret output by R_TSIP_TlsSVDecryptPreMasterSecretWithRsa2048PrivateKey or R_TSIP_TlsSVGeneratePreMasterSecretWithEccP256Key.          |
| client_random          | Input  | 32-byte random number value reported by ClientHello                                                                                                                                                                                                                                                                                                                                                                   |
| server_random          | Input  | (1) 32-byte random number value reported by ServerHello<br>(2) 32-byte random number value reported by ServerHello and its MAC                                                                                                                                                                                                                                                                                        |
| tsip_master_secret     | Output | Encrypted ephemeral master secret data<br>Use this value as the tsip_master_secret parameter input to R_TSIP_TlsXXGenerateSessionKey or R_TSIP_TlsXXGenerateVerifyData. 20 words (80 bytes) of data is output.                                                                                                                                                                                                        |



**Return Values**

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

**Description**

This API generates an encrypted master secret for use by the TLS cooperation function.

**Reentrancy**

Not supported.

#### 4.2.14.6 R\_TSIP\_TlsXXGenerateSessionKey

##### Format

```
(1) #include "r_tsip_rx_if.h"
 e_tsip_err_t R_TSIP_TlsGenerateSessionKey(
 uint32_t select_cipher_suite,
 uint32_t *tsip_master_secret,
 uint8_t *client_random,
 uint8_t *server_random,
 uint8_t *nonce_explicit,
 tsip_hmac_sha_key_index_t *client_mac_key_index,
 tsip_hmac_sha_key_index_t *server_mac_key_index,
 tsip_aes_key_index_t *client_crypt_key_index,
 tsip_aes_key_index_t *server_crypt_key_index,
 uint8_t *client_iv,
 uint8_t *server_iv
)
(2) #include "r_tsip_rx_if.h"
 e_tsip_err_t R_TSIP_TlsSVGenerateSessionKey(
 uint32_t select_cipher_suite,
 uint32_t *tsip_master_secret,
 uint8_t *client_random,
 tsip_tls_sv_random_t *server_random,
 uint8_t *nonce_explicit,
 tsip_hmac_sha_key_index_t *client_mac_key_index,
 tsip_hmac_sha_key_index_t *server_mac_key_index,
 tsip_aes_key_index_t *client_crypt_key_index,
 tsip_aes_key_index_t *server_crypt_key_index,
 uint8_t *client_iv,
 uint8_t *server_iv
)
```

##### Parameters

|                      |        |                                                                                                                                                                                                                                                                                                                                                                                                                       |
|----------------------|--------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| select_cipher_suite  | Input  | Cipher suite selection<br>0: R_TSIP_TLS_RSA_WITH_AES_128_CBC_SHA<br>1: R_TSIP_TLS_RSA_WITH_AES_256_CBC_SHA<br>2: R_TSIP_TLS_RSA_WITH_AES_128_CBC_SHA256<br>3: R_TSIP_TLS_RSA_WITH_AES_256_CBC_SHA256<br>4: R_TSIP_TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256<br>5: R_TSIP_TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256<br>6: R_TSIP_TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256<br>7: R_TSIP_TLS_ECDHE_RSSA_WITH_AES_128_GCM_SHA256 |
| tsip_master_secret   | Input  | Encrypted ephemeral master secret data<br>Use the value of tsip_master_secret output by R_TSIP_TlsXXGenerateMasterSecret.                                                                                                                                                                                                                                                                                             |
| client_random        | Input  | 32-byte random number value reported by ClientHello                                                                                                                                                                                                                                                                                                                                                                   |
| server_random        | Input  | (1) 32-byte random number value reported by ServerHello<br>(2) 32-byte random number value reported by ServerHello and its MAC                                                                                                                                                                                                                                                                                        |
| nonce_explicit       | Input  | Nonce used by AES128 GCM cipher suite<br>select_cipher_suite = 6-7: 8 bytes                                                                                                                                                                                                                                                                                                                                           |
| client_mac_key_index | Output | MAC wrapped key for client to server communication                                                                                                                                                                                                                                                                                                                                                                    |

## RX Family TSIP (Trusted Secure IP) Module Firmware Integration Technology

|                        |        |                                                                                                                                                                                                        |
|------------------------|--------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| server_mac_key_index   | Output | MAC wrapped key for server to client communication                                                                                                                                                     |
| client_crypt_key_index | Output | AES common wrapped key for client to server communication                                                                                                                                              |
| server_crypt_key_index | Output | AES common wrapped key for server to client communication                                                                                                                                              |
| client_iv              | Output | IV used for transmission from client to server<br>Output when the value of select_cipher_suite is 0 to 5. ((1) Used when NetX Duo is employed on the RX651 or RX65N.)<br>Otherwise, nothing is output. |
| server_iv              | Output | IV used for reception from server<br>Output when the value of select_cipher_suite is 0 to 5. ((1) Used when NetX Duo is employed on the RX651 or RX65N.)<br>Otherwise, nothing is output.              |

### Return Values

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

### Description

This API outputs TLS common keys for use by the TLS cooperation function.

Nothing is output for the client\_iv and server\_iv parameters when the status is other than that specified in the parameter explanation.

### Reentrancy

Not supported.

#### 4.2.14.7 R\_TSIP\_TlsXXGenerateVerifyData

##### Format

```
(1) #include "r_tsip_rx_if.h"
 e_tsip_err_t R_TSIP_TlsGenerateVerifyData(
 uint32_t select_verify_data,
 uint32_t *tsip_master_secret,
 uint8_t *hand_shake_hash,
 uint8_t *verify_data
)
(2) #include "r_tsip_rx_if.h"
 e_tsip_err_t R_TSIP_TlsSVGenerateVerifyData(
 uint32_t select_verify_data,
 uint32_t *tsip_master_secret,
 uint8_t *hand_shake_hash,
 uint8_t *verify_data
)
```

##### Parameters

|                    |        |                                                                                                                                                                                   |
|--------------------|--------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| select_verify_data | Input  | Client/server selection<br>R_TSIP_TLS_GENERATE_CLIENT_VERIFY:<br>Client verification data is generated.<br>R_TSIP_TLS_GENERATE_SERVER_VERIFY:<br>Server verify data is generated. |
| tsip_master_secret | Input  | Encrypted ephemeral master secret data<br>Use the value of tsip_master_secret output by<br>R_TSIP_TlsXXGenerateMasterSecret.                                                      |
| hand_shake_hash    | Input  | SHA256 hash value for entire TLS handshake<br>message                                                                                                                             |
| verify_data        | Output | Verification data for Finished message                                                                                                                                            |

##### Return Values

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

##### Description

This API generates verify data for use by the TLS cooperation function.

##### Reentrancy

Not supported.

#### 4.2.14.8 R\_TSIP\_TIsServersEphemeralEcdhPublicKeyRetrieves

##### Format

```
#include "r_tsip_rx_if.h"
e_tsip_err_t R_TSIP_TIsServersEphemeralEcdhPublicKeyRetrieves(
 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<br>0: RSA 2048-bit (rsa_pkcs1_sha256),<br>1: RSA 4096-bit (rsa_pkcs1_sha256),<br>2: ECDSA P-256<br>3: RSA 2048-bit (rsa_pss_rsae_sha256)                                                                                                      |
| client_random                       | Input  | 32-byte random number value reported by ClientHello                                                                                                                                                                                                           |
| server_random                       | Input  | 32-byte random number value reported by ServerHello                                                                                                                                                                                                           |
| server_ephemeral_ecdh_public_key    | Input  | Ephemeral ECDH public key (uncompressed format) received from server<br>0 padding (24 bits)    04 (8 bits)    Qx (256 bits)    Qy (256 bits)                                                                                                                  |
| server_key_exchange_signature       | Input  | ServerKeyExchange signature data<br>public_key_type 0: 256 bytes, 2: 64 bytes                                                                                                                                                                                 |
| encrypted_public_key                | Input  | Encrypted public key for signature verification<br>Use the value of encrypted_output_public_key output by R_TSIP_TIsCertificateVerification or R_TSIP_TIsCertificateVerificationExtension.<br>public_key_type 0: 140 words (560 bytes), 2:24 words (96 bytes) |
| encrypted_ephemeral_ecdh_public_key | Output | Encrypted ephemeral ECDH public key used by R_TSIP_TIsGeneratePreMasterSecretWithEccP256Key<br>The data size is 24 words (96 bytes) size                                                                                                                      |

##### Return Values

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

##### Description

This API verifies a ServerKeyExchange signature, using the input public key data, for use by the TLS cooperation function. If the signature is verified successfully, the ephemeral ECDH public key used by R\_TSIP\_TIsGeneratePreMasterSecretWithEccP256Key is encrypted and output.

Applicable cypher suites: TLS\_ECDHE\_ECDSA\_WITH\_AES\_128\_CBC\_SHA256,  
TLS\_ECDHE\_RSA\_WITH\_AES\_128\_CBC\_SHA256,  
TLS\_ECDHE\_ECDSA\_WITH\_AES\_128\_GCM\_SHA256,  
TLS\_ECDHE\_RSA\_WITH\_AES\_128\_GCM\_SHA256

**Reentrancy**

Not supported.

#### 4.2.14.9 R\_TSIP\_GenerateTlsXXP256EccKeyIndex

##### Format

```
(1) #include "r_tsip_rx_if.h"
 e_tsip_err_t R_TSIP_GenerateTlsP256EccKeyIndex(
 tsip_tls_p256_ecc_key_index_t *tls_p256_ecc_key_index,
 uint8_t *ephemeral_ecdh_public_key
)
(2) #include "r_tsip_rx_if.h"
 e_tsip_err_t R_TSIP_GenerateTlsSVP256EccKeyIndex(
 tsip_tls_p256_ecc_key_index_t *tls_p256_ecc_key_index,
 uint8_t *ephemeral_ecdh_public_key
)
```

##### Parameters

|                           |        |                                                                                                                                                              |
|---------------------------|--------|--------------------------------------------------------------------------------------------------------------------------------------------------------------|
| tls_p256_ecc_key_index    | Output | Ephemeral ECC secret wrapped key<br>Use this value as the tls_p256_ecc_key_index parameter input to R_TSIP_TlsXXGeneratePreMasterSecretWithEccP256Key.       |
| ephemeral_ecdh_public_key | Output | (1) Ephemeral ECDH public key to be sent to the server<br>(2) Ephemeral ECDH public key to be sent to the client<br>Both are Qx (256 bits)    Qy (256 bits). |

##### Return Values

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

##### Description

This API generates a key pair from a random number for use by the TLS cooperation function for elliptic curve cryptography over a 256-bit prime field.

Applicable cypher suites: TLS\_ECDHE\_ECDSA\_WITH\_AES\_128\_CBC\_SHA256,  
 TLS\_ECDHE\_RSA\_WITH\_AES\_128\_CBC\_SHA256,  
 TLS\_ECDHE\_ECDSA\_WITH\_AES\_128\_GCM\_SHA256,  
 TLS\_ECDHE\_RSA\_WITH\_AES\_128\_GCM\_SHA256

##### Reentrancy

Not supported.

#### 4.2.14.10 R\_TSIP\_TlsXXGeneratePreMasterSecretWithEccP256Key

##### Format

```
(1) #include "r_tsip_rx_if.h"
 e_tsip_err_t R_TSIP_TlsGeneratePreMasterSecretWithEccP256Key(
 uint32_t *encrypted_public_key,
 tsip_tls_p256_ecc_key_index_t *tls_p256_ecc_key_index,
 uint32_t *tsip_pre_master_secret
)
(2) #include "r_tsip_rx_if.h"
 e_tsip_err_t R_TSIP_TlsSVGeneratePreMasterSecretWithEccP256Key(
 uint32_t *ecdh_public_key,
 tsip_tls_p256_ecc_key_index_t *tls_p256_ecc_key_index,
 uint32_t *tsip_pre_master_secret
)
```

##### Parameters

|                          |        |                                                                                                                            |
|--------------------------|--------|----------------------------------------------------------------------------------------------------------------------------|
| (1) encrypted_public_key | Input  | Ephemeral ECDH public key                                                                                                  |
| (2) ecdh_public_key      |        | (1)<br>Use the value of encrypted_ephemeral_ecdh_public_key output by R_TSIP_TlsServersEphemeralEcdhPublicKeyRetrieves.    |
|                          |        | (2)<br>Use the value of ephemeral ECDH public key with its format is Qx (256 bits)    Qy (256 bits).                       |
| tls_p256_ecc_key_index   | Input  | Ephemeral ECC secret wrapped key<br>Use the value of tls_p256_ecc_key_index output by R_TSIP_GenerateTlsXXP256EccKeyIndex. |
| tsip_pre_master_secret   | Output | Encrypted ephemeral pre-master secret data<br>16 words (64 bytes) of data is output.                                       |

##### Return Values

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

##### Description

This API generates an encrypted pre-master secret, using the input key data, for use by the TLS cooperation function.

Applicable cypher suites: TLS\_ECDHE\_ECDSA\_WITH\_AES\_128\_CBC\_SHA256,  
 TLS\_ECDHE\_RSA\_WITH\_AES\_128\_CBC\_SHA256,  
 TLS\_ECDHE\_ECDSA\_WITH\_AES\_128\_GCM\_SHA256,  
 TLS\_ECDHE\_RSA\_WITH\_AES\_128\_GCM\_SHA256

##### Reentrancy

Not supported.



#### 4.2.14.11 R\_TSIP\_TlsXXGenerateExtendedMasterSecret

##### Format

```
(1) #include "r_tsip_rx_if.h"
 e_tsip_err_t R_TSIP_TlsGenerateExtendedMasterSecret(
 uint32_t select_cipher_suite,
 uint32_t *tsip_pre_master_secret,
 uint8_t *digest,
 uint32_t *extended_master_secret
)
(2) #include "r_tsip_rx_if.h"
 e_tsip_err_t R_TSIP_TlsSVGenerateExtendedMasterSecret(
 uint32_t select_cipher_suite,
 uint32_t *tsip_pre_master_secret,
 uint8_t *digest,
 uint32_t *extended_master_secret
)
```

##### Parameters

|                        |        |                                                                                                                                                                                                                                                                                                                                                                                                                |
|------------------------|--------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| select_cipher_suite    | Input  | <p>Cipher suite selection</p> <p>2: R_TSIP_TLS_RSA_WITH_AES_128_CBC_SHA256<br/>         3: R_TSIP_TLS_RSA_WITH_AES_256_CBC_SHA256<br/>         4: R_TSIP_TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256<br/>         5: R_TSIP_TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256<br/>         6: R_TSIP_TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256<br/>         7: R_TSIP_TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256</p>                  |
| tsip_pre_master_secret | Input  | <p>Encrypted ephemeral pre-master secret data</p> <p>(1)<br/>         Use the value of tsip_pre_master_secret output by R_TSIP_TlsGeneratePreMasterSecret or R_TSIP_TlsGeneratePreMasterSecretWithEccP256Key.</p> <p>(2)<br/>         Use the value of tsip_pre_master_secret output by R_TSIP_TlsSVDecryptPreMasterSecretWithRsa2048PrivateKey or R_TSIP_TlsSVGeneratePreMasterSecretWithEccP256Key.</p>      |
| digest                 | Input  | <p>Message hash calculated using SHA256</p> <p>Calculate and input a hash value of the value of concatenated handshake messages such as (ClientHello  ServerHello  Certificate  ServerKeyExchange  CertificateRequest  ServerHelloDone  Certificate  ClientKeyExchange).</p> <p>Use R_TSIP_Sha256Init/Update/Final to calculate the hash value and input as digest the value output by R_TSIP_Sha256Final.</p> |
| extended_master_secret | Output | <p>Encrypted ephemeral extended master secret data</p> <p>20 words (80 bytes) of data is output.</p> <p>Use this value as the tsip_master_secret parameter input to R_TSIP_TlsXXGenerateSessionKey or R_TSIP_TlsXXGenerateVerifyData.</p>                                                                                                                                                                      |

**Return Values**

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

**Description**

This API generates encrypted extended master secret data, using encrypted pre-master secret data, for use by the TLS cooperation function.

**Reentrancy**

Not supported.

#### 4.2.14.12 R\_TSIP\_TlsSVCertificateVerifyVerification

##### Format

```
#include "r_tsip_rx_if.h"
e_tsip_err_t R_TSIP_TlsSVCertificateVerifyVerification(
 uint32_t *key_index,
 e_tsip_tls_signature_scheme_type_t signature_scheme,
 uint8_t *handshake_message,
 uint32_t handshake_message_len,
 uint8_t *certificate_verify,
 uint32_t certificate_verify_len
)
```

##### Parameters

|                        |       |                                                                                                                                                                                                                                                                                                                                            |
|------------------------|-------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| key_index              | Input | Encrypted public key<br>Use encrypted_output_public_key output by R_TSIP_TlsCertificateVerification or R_TSIP_TlsCertificateVerificationExtension.                                                                                                                                                                                         |
| signature_scheme       | Input | Signature algorithm to be used<br>TSIP_TLS_SIGNATURE_SCHEME_RSA_PKCS1_SHA256:<br>rsa_pkcs1_sha256<br>TSIP_TLS_SIGNATURE_SCHEME_ECDSA_SECP256R1_SHA256:<br>ecdsa_secp256r1_sha256<br>TSIP_TLS_SIGNATURE_SCHEME_RSA_PSS_RSAE_SHA256:<br>rsa_pss_rsae_sha256                                                                                  |
| digest                 | Input | Handshake message<br>Input a value of concatenated handshake messages such as (ClientHello  ServerHello  Certificate  ServerKeyExchange  CertificateRequest  ServerHelloDone  CertificateClientKeyExchange).<br>Use R_TSIP_Sha256Init/Update/Final to calculate the hash value and input as digest the value output by R_TSIP_Sha256Final. |
| handshake_message_len  | Input | Byte length of handshake_message                                                                                                                                                                                                                                                                                                           |
| certificate_verify     | Input | CertificateVerify<br>Input the start address of the buffer for storing the data.                                                                                                                                                                                                                                                           |
| certificate_verify_len | Input | Byte length of certificate_verify                                                                                                                                                                                                                                                                                                          |

##### Return Values

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

##### Description

This API verifies a CertificateVerify message received from the client for use by the server function of the TLS cooperation function.

This API can be used for same purpose as the `R_TSIP_TlsCertificateVerificationExtension()` function, but make sure to use this function when the algorithm of the key used for signature verification is the same as the algorithm used to obtain the key from the certificate.

**Reentrancy**

Not supported.

## 4.2.15 TLS (TLS 1.3)

### 4.2.15.1 R\_TSIP\_GenerateTls13P256EccKeyIndex

#### Format

```
#include "r_tsip_rx_if.h"
e_tsip_err_t R_TSIP_TlsGenerateTls13P256EccKeyIndex(
 tsip_tls13_handle_t *handle,
 e_tsip_tls13_mode_t mode,
 tsip_tls_p256_ecc_key_index_t *key_index,
 uint8_t *ephemeral_ecdh_public_key
)
```

#### Parameters

|                           |        |                                                                                                                                                                |
|---------------------------|--------|----------------------------------------------------------------------------------------------------------------------------------------------------------------|
| handle                    | Input  | Handle number (work area) indicating same session                                                                                                              |
| mode                      | Input  | Handshake protocol to use<br>TSIP_TLS13_MODE_FULL_HANDSHAKE:<br>Full handshake<br>TSIP_TLS13_MODE_RESUMPTION:<br>Resumption<br>TSIP_TLS13_MODE_0_RTT:<br>0-RTT |
| key_index                 | Output | Ephemeral ECC secret wrapped key<br>Use this value as the key_index parameter input to<br>R_TSIP_Tls13GenerateEcdheSharedSecret.                               |
| ephemeral_ecdh_public_key | Output | Ephemeral ECDH public key to be sent to the server<br>Qx (256 bits)    Qy (256 bits)                                                                           |

#### Return Values

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

#### Description

This API generates a key pair from a random number for use by the TLS 1.3 cooperation function for elliptic curve cryptography over a 256-bit prime field.

#### Reentrancy

Not supported.

### 4.2.15.2 R\_TSIP\_Tls13GenerateEcdheSharedSecret

#### Format

```
#include "r_tsip_rx_if.h"
e_tsip_err_t R_TSIP_Tls13GenerateEcdheSharedSecret(
 e_tsip_tls13_mode_t mode,
 uint8_t *server_public_key,
 tsip_tls_p256_ecc_key_index_t *key_index,
 tsip_tls13_ephemeral_shared_secret_key_index_t *shared_secret_key_index
)
```

#### Parameters

|                         |        |                                                                                                                                                                                                         |
|-------------------------|--------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| mode                    | Input  | Handshake protocol to use<br>TSIP_TLS13_MODE_FULL_HANDSHAKE:<br>Full handshake<br>TSIP_TLS13_MODE_RESUMPTION:<br>Resumption<br>TSIP_TLS13_MODE_0_RTT:<br>0-RTT                                          |
| server_public_key       | Input  | Public key provided by the server<br>Qx (256 bits)    Qy (256 bits)                                                                                                                                     |
| key_index               | Input  | Ephemeral ECC secret wrapped key<br>Use the value of key_index output by<br>R_TSIP_GenerateTls13P256EccKeyIndex.                                                                                        |
| shared_secret_key_index | Output | Shared secret ephemeral wrapped key<br>Use this value as the shared_secret_key_index<br>parameter input to<br>R_TSIP_Tls13GenerateHandshakeSecret and<br>R_TSIP_Tls13GenerateResumptionHandshakeSecret. |

#### Return Values

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

#### Description

This API calculates a shared secret, which is a shared key over a 256-bit prime field, using a public key provided by the server and a previously calculated secret key, and generates a wrapped key for use by the TLS 1.3 cooperation function.

Applicable cypher suites: TLS\_AES\_128\_GCM\_SHA256, TLS\_AES\_128\_CCM\_SHA256

Key exchange format: ECDHE NIST P-256

#### Reentrancy

Not supported.

---

### 4.2.15.3 R\_TSIP\_Tls13GenerateHandshakeSecret

---

**Format**

```
#include "r_tsip_rx_if.h"
e_tsip_err_t R_TSIP_Tls13GenerateHandshakeSecret(
 tsip_tls13_ephemeral_shared_secret_key_index_t *shared_secret_key_index,
 tsip_tls13_ephemeral_handshake_secret_key_index_t *handshake_secret_key_index
)
```

**Parameters**

|                            |        |                                                                                                                                                                                                                                                   |
|----------------------------|--------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| shared_secret_key_index    | Input  | Shared secret ephemeral wrapped key<br>Use the value of shared_secret_key_index output by R_TSIP_Tls13GenerateEcdheSharedSecret.                                                                                                                  |
| handshake_secret_key_index | Output | Handshake secret ephemeral wrapped key<br>Use this value as the handshake_secret_key_index parameter input to R_TSIP_Tls13GenerateServerHandshakeTrafficKey, R_TSIP_Tls13GenerateClientHandshakeTrafficKey, and R_TSIP_Tls13GenerateMasterSecret. |

**Return Values**

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

**Description**

This API generates a handshake secret wrapped key, using a shared secret ephemeral key, for use by the TLS 1.3 cooperation function.

**Reentrancy**

Not supported.

#### 4.2.15.4 R\_TSIP\_Tls13GenerateServerHandshakeTrafficKey

##### Format

```
#include "r_tsip_rx_if.h"
e_tsip_err_t R_TSIP_Tls13GenerateServerHandshakeTrafficKey(
 tsip_tls13_handle_t *handle,
 e_tsip_tls13_mode_t mode,
 tsip_tls13_ephemeral_handshake_secret_key_index_t *handshake_secret_key_index,
 uint8_t *digest,
 tsip_aes_key_index_t *server_write_key_index,
 tsip_tls13_ephemeral_server_finished_key_index_t *server_finished_key_index
)
```

##### Parameters

|                            |        |                                                                                                                                                                                                                                         |
|----------------------------|--------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| handle                     | Output | Handle number (work area) indicating same session                                                                                                                                                                                       |
| mode                       | Input  | Handshake protocol to use<br>TSIP_TLS13_MODE_FULL_HANDSHAKE:<br>Full handshake<br>TSIP_TLS13_MODE_RESUMPTION:<br>Resumption<br>TSIP_TLS13_MODE_0_RTT:<br>0-RTT                                                                          |
| handshake_secret_key_index | Input  | Handshake secret ephemeral wrapped key<br>Use the value of handshake_secret_key_index output by R_TSIP_Tls13GenerateHandshakeSecret or R_TSIP_Tls13GenerateResumptionHandshakeSecret.                                                   |
| digest                     | Input  | Message hash calculated using SHA256<br>Calculate and input the hash value of (ClientHello  ServerHello).<br>Use R_TSIP_Sha256Init/Update/Final to calculate the hash value and input as digest the value output by R_TSIP_Sha256Final. |
| server_write_key_index     | Output | Server write key ephemeral wrapped key<br>Use this value as the key_index parameter input to R_TSIP_Tls13DecryptInit.                                                                                                                   |
| server_finished_key_index  | Output | Server finished key ephemeral wrapped key<br>Use this value as the server_finished_key_index parameter input to R_TSIP_Tls13ServerHandshakeVerification.                                                                                |

##### Return Values

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

##### Description

This API generates, using the handshake secret output by R\_TSIP\_Tls13GenerateHandshakeSecret, a server write wrapped key and server finished wrapped key for use by the TLS 1.3 cooperation function.



**Reentrancy**

Not supported.

### 4.2.15.5 R\_TSIP\_Tls13GenerateClientHandshakeTrafficKey

#### Format

```
#include "r_tsip_rx_if.h"
e_tsip_err_t R_TSIP_Tls13GenerateClientHandshakeTrafficKey(
 tsip_tls13_handle_t *handle,
 e_tsip_tls13_mode_t mode,
 tsip_tls13_ephemeral_handshake_secret_key_index_t *handshake_secret_key_index,
 uint8_t *digest,
 tsip_aes_key_index_t *client_write_key_index,
 tsip_hmac_sha_key_index_t *client_finished_key_index
)
```

#### Parameters

|                            |        |                                                                                                                                                                                                                                         |
|----------------------------|--------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| handle                     | Output | Handle number (work area) indicating same session                                                                                                                                                                                       |
| mode                       | Input  | Handshake protocol to use<br>TSIP_TLS13_MODE_FULL_HANDSHAKE:<br>Full handshake<br>TSIP_TLS13_MODE_RESUMPTION:<br>Resumption<br>TSIP_TLS13_MODE_0_RTT:<br>0-RTT                                                                          |
| handshake_secret_key_index | Input  | Handshake secret ephemeral wrapped key<br>Use the value of handshake_secret_key_index output by R_TSIP_Tls13GenerateHandshakeSecret or R_TSIP_Tls13GenerateResumptionHandshakeSecret.                                                   |
| digest                     | Input  | Message hash calculated using SHA256<br>Calculate and input the hash value of (ClientHello  ServerHello).<br>Use R_TSIP_Sha256Init/Update/Final to calculate the hash value and input as digest the value output by R_TSIP_Sha256Final. |
| client_write_key_index     | Output | Client write key ephemeral wrapped key<br>Use this value as the key_index parameter input to R_TSIP_Tls13EncryptInit.                                                                                                                   |
| client_finished_key_index  | Output | Client finished key ephemeral wrapped key<br>Use this value as the key_index parameter input to R_TSIP_Sha256HmacGenerateInit.                                                                                                          |

#### Return Values

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

#### Description

This API generates, using the handshake secret output by R\_TSIP\_Tls13GenerateHandshakeSecret, a client write wrapped key and client finished wrapped key for use by the TLS 1.3 cooperation function.

**Reentrancy**

Not supported.

#### 4.2.15.6 R\_TSIP\_Tls13ServerHandshakeVerification

##### Format

```
#include "r_tsip_rx_if.h"
e_tsip_err_t R_TSIP_Tls13ServerHandshakeVerification(
 e_tsip_tls13_mode_t mode,
 tsip_tls13_ephemeral_server_finished_key_index_t *server_finished_key_index,
 uint8_t *digest,
 uint8_t *server_finished,
 uint32_t *verify_data_index
)
```

##### Parameters

|                           |        |                                                                                                                                                                                                                                                                                                                                                                                  |
|---------------------------|--------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| mode                      | Input  | Handshake protocol to use<br>TSIP_TLS13_MODE_FULL_HANDSHAKE:<br>Full handshake<br>TSIP_TLS13_MODE_RESUMPTION:<br>Resumption<br>TSIP_TLS13_MODE_0_RTT:<br>0-RTT                                                                                                                                                                                                                   |
| server_finished_key_index | Input  | Server finished key ephemeral wrapped key<br>Use the value of server_finished_key_index output by<br>R_TSIP_Tls13GenerateServerHandshakeTrafficKey.                                                                                                                                                                                                                              |
| digest                    | Input  | Message hash calculated using SHA256<br>Calculate and input a hash value of the value of concatenated handshake messages such as (ClientHello  ServerHello  EncryptedExtensions  CertificateRequest  Certificate  CertificateVerify).<br>Use the value of R_TSIP_Sha256Init/Update/Final to calculate the hash value and input as digest the value output by R_TSIP_Sha256Final. |
| server_finished           | Input  | Finished information provided by the server<br>Input the start address of the buffer for storing the ServerFinished data obtained from<br>R_TSIP_Tls13DecryptUpdate/Final.                                                                                                                                                                                                       |
| verify_data_index         | Output | ServerHandshake verification result conforming to TSIP-specific specification<br>Use this value as the as verify_data_index parameter input to R_TSIP_Tls13GenerateMasterSecret.<br>Input the start address of the buffer to which data is to be output. The size must be 8 words (32 bytes).                                                                                    |

##### Return Values

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

|                            |                           |
|----------------------------|---------------------------|
| TSIP_ERR_KEY_SET           | Invalid wrapped key input |
| TSIP_ERR_VERIFICATION_FAIL | Verification failure      |

**Description**

This API verifies handshake messages, using the Finished information provided by the server, for use by the TLS 1.3 cooperation function.

**Reentrancy**

Not supported.

### 4.2.15.7 R\_TSIP\_Tls13GenerateMasterSecret

#### Format

```
#include "r_tsip_rx_if.h"
e_tsip_err_t R_TSIP_Tls13GenerateMasterSecret(
 tsip_tls13_handle_t *handle,
 e_tsip_tls13_mode_t mode,
 tsip_tls13_ephemeral_handshake_secret_key_index_t *handshake_secret_key_index,
 uint32_t *verify_data_index,
 tsip_tls13_ephemeral_master_secret_key_index_t *master_secret_key_index
)
```

#### Parameters

|                            |        |                                                                                                                                                                                                  |
|----------------------------|--------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| handle                     | Output | Handle number (work area) indicating same session                                                                                                                                                |
| mode                       | Input  | Handshake protocol to use<br>TSIP_TLS13_MODE_FULL_HANDSHAKE:<br>Full handshake<br>TSIP_TLS13_MODE_RESUMPTION:<br>Resumption<br>TSIP_TLS13_MODE_0_RTT:<br>0-RTT                                   |
| handshake_secret_key_index | Input  | Handshake secret ephemeral wrapped key<br>Use the value of handshake_secret_key_index output by R_TSIP_Tls13GenerateHandshakeSecret.                                                             |
| verify_data_index          | Input  | ServerHandshake verification result conforming to TSIP-specific specification<br>Use the value of verify_data_index output by R_TSIP_Tls13ServerHandshakeVerification.                           |
| master_secret_key_index    | Output | Master secret ephemeral wrapped key<br>Use this value as the master_secret_key_index parameter input to R_TSIP_Tls13GenerateApplicationTrafficKey and R_TSIP_Tls13GenerateResumptionMasterSecret |

#### Return Values

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

#### Description

This API generates a master secret ephemeral wrapped key, using a handshake secret ephemeral key, for use by the TLS 1.3 cooperation function.

#### Reentrancy

Not supported.

#### 4.2.15.8 R\_TSIP\_Tls13GenerateApplicationTrafficKey

##### Format

```
#include "r_tsip_rx_if.h"
e_tsip_err_t R_TSIP_Tls13GenerateApplicationTrafficKey(
 tsip_tls13_handle_t *handle,
 e_tsip_tls13_mode_t mode,
 tsip_tls13_ephemeral_master_secret_key_index_t *master_secret_key_index,
 uint8_t *digest,
 tsip_tls13_ephemeral_app_secret_key_index_t *server_app_secret_key_index,
 tsip_tls13_ephemeral_app_secret_key_index_t *client_app_secret_key_index,
 tsip_aes_key_index_t *server_write_key_index,
 tsip_aes_key_index_t *client_write_key_index
)
```

##### Parameters

|                             |              |                                                                                                                                                                                                                                                                                                                                                                                     |
|-----------------------------|--------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| handle                      | Input/output | Handle number (work area) indicating same session                                                                                                                                                                                                                                                                                                                                   |
| mode                        | Input        | Handshake protocol to use<br>TSIP_TLS13_MODE_FULL_HANDSHAKE:<br>Full handshake<br>TSIP_TLS13_MODE_RESUMPTION:<br>Resumption<br>TSIP_TLS13_MODE_0_RTT:<br>0-RTT                                                                                                                                                                                                                      |
| master_secret_key_index     | Input        | Master secret ephemeral wrapped key<br>Use the value of master_secret_key_index output by R_TSIP_Tls13GenerateMasterSecret.                                                                                                                                                                                                                                                         |
| digest                      | Input        | Message hash calculated using SHA256<br>Calculate and input a hash value of the value of concatenated handshake messages such as (ClientHello  ServerHello  EncryptedExtensions  CertificateRequest  Certificate  CertificateVerify  ServerFinished).<br>Use R_TSIP_Sha256Init/Update/Final to calculate the hash value and input as digest the value output by R_TSIP_Sha256Final. |
| server_app_secret_key_index | Output       | Server application traffic secret ephemeral wrapped key<br>Use this value as the input_app_secret_key_index parameter input to R_TSIP_Tls13UpdateApplicationTrafficKey.                                                                                                                                                                                                             |
| client_app_secret_key_index | Output       | Client application traffic secret ephemeral wrapped key<br>Use this value as the input_app_secret_key_index parameter input to R_TSIP_Tls13UpdateApplicationTrafficKey.                                                                                                                                                                                                             |
| server_write_key_index      | Output       | Server write key ephemeral wrapped key<br>Use this value as the key_index parameter input to R_TSIP_Tls13DecryptInit.                                                                                                                                                                                                                                                               |

|                        |        |                                                                                                                       |
|------------------------|--------|-----------------------------------------------------------------------------------------------------------------------|
| client_write_key_index | Output | Client write key ephemeral wrapped key<br>Use this value as the key_index parameter input to R_TSIP_Tls13EncryptInit. |
|------------------------|--------|-----------------------------------------------------------------------------------------------------------------------|

**Return Values**

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

**Description**

This API generates an application traffic secret wrapped key, using a master secret ephemeral key, for use by the TLS 1.3 cooperation function. It also generates server write key and client write key ephemeral wrapped keys.

**Reentrancy**

Not supported.



### 4.2.15.9 R\_TSIP\_Tls13UpdateApplicationTrafficKey

#### Format

```
#include "r_tsip_rx_if.h"
e_tsip_err_t R_TSIP_Tls13UpdateApplicationTrafficKey(
 tsip_tls13_handle_t *handle,
 e_tsip_tls13_mode_t mode,
 e_tsip_tls13_update_key_type_t key_type,
 tsip_tls13_ephemeral_app_secret_key_index_t *input_app_secret_key_index,
 tsip_tls13_ephemeral_app_secret_key_index_t *output_app_secret_key_index,
 tsip_aes_key_index_t *app_write_key_index
)
```

#### Parameters

|                             |              |                                                                                                                                                                                                                                                                                                                                                    |
|-----------------------------|--------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| handle                      | Input/output | Handle number (work area) indicating same session                                                                                                                                                                                                                                                                                                  |
| mode                        | Input        | Handshake protocol to use<br>TSIP_TLS13_MODE_FULL_HANDSHAKE:<br>Full handshake<br>TSIP_TLS13_MODE_RESUMPTION:<br>Resumption<br>TSIP_TLS13_MODE_0_RTT:<br>0-RTT                                                                                                                                                                                     |
| key_type                    | Input        | Type of key to be updated<br>TSIP_TLS13_UPDATE_SERVER_KEY:<br>Server application traffic secret<br>TSIP_TLS13_UPDATE_CLIENT_KEY:<br>Client application traffic secret                                                                                                                                                                              |
| input_app_secret_key_index  | Input        | Ephemeral wrapped key of input server/client application traffic secret<br>Use as input either<br>server/client_app_secret_key_index output by<br>R_TSIP_Tls13GenerateApplicationTrafficKey or<br>output_app_secret_key_index output by<br>R_TSIP_Tls13UpdateApplicationTrafficKey,<br>whichever matches the type of key specified by<br>key_type. |
| output_app_secret_key_index | Output       | Ephemeral wrapped key of output server/client application traffic secret<br>Output matching the type of key specified by<br>key_type is obtained.<br>Use this value as the input_app_secret_key_index<br>parameter input to<br>R_TSIP_Tls13UpdateApplicationTrafficKey.                                                                            |
| app_write_key_index         | Output       | Server/client write key ephemeral wrapped key<br>Output matching the type of key specified by<br>key_type is obtained.<br>Use ServerWriteKey as the key_index parameter<br>input to R_TSIP_Tls13DecryptInit.<br>Use ClientWriteKey as the key_index parameter<br>input to R_TSIP_Tls13EncryptInit.                                                 |

**Return Values**

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

**Description**

This API updates, using an application traffic secret, an encryption wrapped key corresponding to an application traffic secret wrapped key for use by the TLS 1.3 cooperation function.

**Reentrancy**

Not supported.

#### 4.2.15.10 R\_TSIP\_Tls13GenerateResumptionMasterSecret

##### Format

```
#include "r_tsip_rx_if.h"
e_tsip_err_t R_TSIP_Tls13GenerateResumptionMasterSecret(
 tsip_tls13_handle_t *handle,
 e_tsip_tls13_mode_t mode,
 tsip_tls13_ephemeral_master_secret_key_index_t *master_secret_key_index,
 uint8_t *digest,
 tsip_tls13_ephemeral_res_master_secret_key_index_t *res_master_secret_key_index
)
```

##### Parameters

|                             |        |                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
|-----------------------------|--------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| handle                      | Input  | Handle number (work area) indicating same session                                                                                                                                                                                                                                                                                                                                                                                                               |
| mode                        | Input  | Handshake protocol to use<br>TSIP_TLS13_MODE_FULL_HANDSHAKE:<br>Full handshake<br>TSIP_TLS13_MODE_RESUMPTION:<br>Resumption<br>TSIP_TLS13_MODE_0_RTT:<br>0-RTT                                                                                                                                                                                                                                                                                                  |
| master_secret_key_index     | Input  | Handshake secret ephemeral wrapped key<br>Use the value of handshake_secret_key_index<br>output by R_TSIP_Tls13GenerateHandshakeSecret.                                                                                                                                                                                                                                                                                                                         |
| digest                      | Input  | Message hash calculated using SHA256<br><br>Calculate and input a hash value of the value of<br>concatenated handshake messages such as<br>(ClientHello  ServerHello  EncryptedExtensions<br>  CertificateRequest  Certificate  CertificateVerify<br>  ServerFinished  Certificate  CertificateVerify<br>  ClientFinished).<br>Use R_TSIP_Sha256Init/Update/Final to calculate<br>the hash value and input as digest the value output<br>by R_TSIP_Sha256Final. |
| res_master_secret_key_index | Output | Resumption master secret ephemeral wrapped key<br>Use this value as the res_master_secret_key_index<br>parameter input to<br>R_TSIP_Tls13GeneratePreSharedKey.                                                                                                                                                                                                                                                                                                  |

##### Return Values

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

**Description**

This API generates a resumption master secret wrapped key, using a master secret ephemeral key, for use by the TLS 1.3 cooperation function.

As specified in RFC 8446, delete `master_secret_key_index`, the master secret ephemeral wrapped key, after generating a resumption master secret wrapped key using this API.

**Reentrancy**

Not supported.

#### 4.2.15.11 R\_TSIP\_Tls13GeneratePreSharedKey

##### Format

```
#include "r_tsip_rx_if.h"
e_tsip_err_t R_TSIP_Tls13GeneratePreSharedKey(
 tsip_tls13_handle_t *handle,
 e_tsip_tls13_mode_t mode,
 tsip_tls13_ephemeral_res_master_secret_key_index_t *res_master_secret_key_index,
 uint8_t *ticket_nonce,
 uint32_t ticket_nonce_len,
 tsip_tls13_ephemeral_pre_shared_key_index_t *pre_shared_key_index
)
```

##### Parameters

|                             |        |                                                                                                                                                                                                                                                  |
|-----------------------------|--------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| handle                      | Input  | Handle number (work area) indicating same session                                                                                                                                                                                                |
| mode                        | Input  | Handshake protocol to use<br>TSIP_TLS13_MODE_FULL_HANDSHAKE:<br>Full handshake<br>TSIP_TLS13_MODE_RESUMPTION:<br>Resumption<br>TSIP_TLS13_MODE_0_RTT:<br>0-RTT                                                                                   |
| res_master_secret_key_index | Input  | Resumption master secret ephemeral wrapped key<br>Use the value of res_master_secret_key_index output by R_TSIP_Tls13GenerateResumptionMasterSecret.                                                                                             |
| ticket_nonce                | Input  | Ticket nonce provided by the server<br>If the size of the ticket nonce is not a multiple of 16 bytes, pad it with zeros to make it a multiple of 16 bytes before input.                                                                          |
| ticket_nonce_len            | Input  | Byte length of ticket nonce                                                                                                                                                                                                                      |
| pre_shared_key_index        | Output | Pre-shared key ephemeral wrapped key<br>Use this value as the pre_shared_key_index parameter input to<br>R_TSIP_Tls13GeneratePskBinderKey,<br>R_TSIP_Tls13GenerateResumptionHandshakeSecret,<br>and R_TSIP_Tls13Generate0RttApplicationWriteKey. |

##### Return Values

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

##### Description

This API generates, using a resumption master secret ephemeral key, a pre-shared wrapped key from new session ticket information for use by the TLS 1.3 cooperation function.

**Reentrancy**

Not supported.

---

#### 4.2.15.12 R\_TSIP\_Tls13GeneratePskBinderKey

---

**Format**

```
#include "r_tsip_rx_if.h"
e_tsip_err_t R_TSIP_Tls13GeneratePskBinderKey(
 tsip_tls13_handle_t *handle,
 tsip_tls13_ephemeral_pre_shared_key_index_t *pre_shared_key_index,
 tsip_hmac_sha_key_index_t *psk_binder_key_index
)
```

**Parameters**

|                      |        |                                                                                                                                                                    |
|----------------------|--------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| handle               | Input  | Handle number (work area) indicating same session                                                                                                                  |
| pre_shared_key_index | Input  | Pre-shared key ephemeral wrapped key<br>Use the value of pre_shared_key_index output by R_TSIP_Tls13GeneratePreSharedKey.                                          |
| psk_binder_key_index | Output | PSK binder key ephemeral wrapped key<br>Use this value to generate PskBinder.<br>Use this value as the key_index parameter input to R_TSIP_Sha256HmacGenerateInit. |

**Return Values**

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

**Description**

This API generates binder wrapped key for use by the TLS 1.3 cooperation function.

**Reentrancy**

Not supported.

**4.2.15.13 R\_TSIP\_Tls13GenerateResumptionHandshakeSecret****Format**

```
#include "r_tsip_rx_if.h"
e_tsip_err_t R_TSIP_Tls13GenerateResumptionHandshakeSecret(
 tsip_tls13_handle_t *handle,
 e_tsip_tls13_mode_t mode,
 tsip_tls13_ephemeral_pre_shared_key_index_t *pre_shared_key_index,
 tsip_tls13_ephemeral_shared_secret_key_index_t *shared_secret_key_index,
 tsip_tls13_ephemeral_handshake_secret_key_index_t *handshake_secret_key_index
)
```

**Parameters**

|                            |        |                                                                                                                                                                                                                                                   |
|----------------------------|--------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| handle                     | Input  | Handle number (work area) indicating same session                                                                                                                                                                                                 |
| mode                       | Input  | Handshake protocol to use<br>TSIP_TLS13_MODE_FULL_HANDSHAKE:<br>Full handshake<br>TSIP_TLS13_MODE_RESUMPTION:<br>Resumption<br>TSIP_TLS13_MODE_0_RTT:<br>0-RTT                                                                                    |
| pre_shared_key_index       | Input  | Pre-shared key ephemeral wrapped key<br>Use the value of pre_shared_key_index output by R_TSIP_Tls13GeneratePreSharedKey.                                                                                                                         |
| shared_secret_key_index    | Input  | Shared secret ephemeral wrapped key<br>Use the value of shared_secret_key_index output by R_TSIP_Tls13GenerateEcdheSharedSecret.                                                                                                                  |
| handshake_secret_key_index | Output | Handshake secret ephemeral wrapped key<br>Use this value as the handshake_secret_key_index parameter input to R_TSIP_Tls13GenerateServerHandshakeTrafficKey, R_TSIP_Tls13GenerateClientHandshakeTrafficKey, and R_TSIP_Tls13GenerateMasterSecret. |

**Return Values**

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

**Description**

This API generates a handshake secret wrapped key, using the pre-shared wrapped key generated by R\_TSIP\_Tls13GeneratePreSharedKey, for use by the TLS 1.3 cooperation function.

Only pre-shared keys generated by the TSIP can be used. Other pre-shared keys are not supported.



**Reentrancy**

Not supported.

#### 4.2.15.14 R\_TSIP\_Tls13Generate0RttApplicationWriteKey

##### Format

```
#include "r_tsip_rx_if.h"
e_tsip_err_t R_TSIP_Tls13Generate0RttApplicationWriteKey(
 tsip_tls13_handle_t *handle,
 tsip_tls13_ephemeral_pre_shared_key_index_t *pre_shared_key_index,
 uint8_t *digest,
 tsip_aes_key_index_t *client_write_key_index
)
```

##### Parameters

| Parameter              | Direction    | Description                                                                                                                                                                                                              |
|------------------------|--------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| handle                 | Input/output | Handle number (work area) indicating same session                                                                                                                                                                        |
| pre_shared_key_index   | Input        | Pre-shared key ephemeral wrapped key<br>Use the value of pre_shared_key_index output by R_TSIP_Tls13GeneratePreSharedKey.                                                                                                |
| digest                 | Input        | Message hash calculated using SHA256<br>Calculate and input the hash value of ClientHello.<br>Use R_TSIP_Sha256Init/Update/Final to calculate the hash value and input as digest the value output by R_TSIP_Sha256Final. |
| client_write_key_index | Output       | Client write key ephemeral wrapped key<br>Use this value as the key_index parameter input to R_TSIP_Tls13EncryptInit.                                                                                                    |

##### Return Values

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

##### Description

This API generates a client write wrapped key for use as 0-RTT, using the pre-shared key generated by R\_TSIP\_Tls13GeneratePreSharedKey, for use by the TLS 1.3 cooperation function.

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

##### Reentrancy

Not supported.

**4.2.15.15 R\_TSIP\_Tls13CertificateVerifyGenerate****Format**

```
#include "r_tsip_rx_if.h"
e_tsip_err_t R_TSIP_Tls13CertificateVerifyGenerate(
 uint32_t *key_index,
 e_tsip_tls13_signature_scheme_type_t signature_scheme,
 uint8_t *digest,
 uint8_t *certificate_verify,
 uint32_t *certificate_verify_len
)
```

**Parameters**

|                        |        |                                                                                                                                                                                                                                                                                                                                                                                                    |
|------------------------|--------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| key_index              | Input  | Secret wrapped key for signature generation<br>Use the value of key_pair_index or key_index output by R_TSIP_GenerateEccP256PrivateKeyIndex, R_TSIP_GenerateEccP256RandomKeyIndex, R_TSIP_UpdateEccP256PrivateKeyIndex, R_TSIP_GenerateRsa2048PrivateKeyIndex, R_TSIP_GenerateRsa2048RandomKeyIndex, or R_TSIP_UpdateRsa2048PrivateKeyIndex.<br>Input this parameter after casting with uint32_t*. |
| signature_scheme       | Input  | Signature algorithm to be used<br>TSIP_TLS13_SIGNATURE_SCHEME_ECDSA_SECP256R1_SHA256:<br>ecdsa_secp256r1_sha256<br>TSIP_TLS13_SIGNATURE_SCHEME_RSA_PSS_RSAE_SHA256:<br>rsa_pss_rsae_sha256                                                                                                                                                                                                         |
| digest                 | Input  | Message hash calculated using SHA256<br>Calculate and input a hash value of the value of concatenated handshake messages such as (ClientHello  ServerHello  EncryptedExtensions  CertificateRequest  Certificate  CertificateVerify  ServerFinished  Certificate).<br>Use R_TSIP_Sha256Init/Update/Final to calculate the hash value and input as digest the value output by R_TSIP_Sha256Final.   |
| certificate_verify     | Output | CertificateVerify<br>Data is output in the format specified in RFC 8446 section 4.4.3, Certificate Verify.                                                                                                                                                                                                                                                                                         |
| certificate_verify_len | Output | Byte length of certificate_verify                                                                                                                                                                                                                                                                                                                                                                  |

**Return Values**

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

**Description**

This API generates a CertificateVerify message to be sent to the server for use by the TLS 1.3 cooperation function. The algorithms used are ecdsa\_secp256r1\_sha256 and rsa\_pss\_rsae\_sha256.

**Reentrancy**

Not supported.

#### 4.2.15.16 R\_TSIP\_Tls13CertificateVerifyVerification

##### Format

```
#include "r_tsip_rx_if.h"
e_tsip_err_t R_TSIP_Tls13CertificateVerifyVerification(
 uint32_t *key_index,
 e_tsip_tls13_signature_scheme_type_t signature_scheme,
 uint8_t *digest,
 uint8_t *certificate_verify,
 uint32_t certificate_verify_len
)
```

##### Parameters

|                        |       |                                                                                                                                                                                                                                                                                                                                                  |
|------------------------|-------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| key_index              | Input | Encrypted public key<br>Use the value of encrypted_output_public_key output by R_TSIP_TlsCertificateVerification or R_TSIP_TlsCertificateVerificationExtension.                                                                                                                                                                                  |
| signature_scheme       | Input | Signature algorithm to be used<br>TSIP_TLS13_SIGNATURE_SCHEME_ECDSA_SECP256R1_SHA256:<br>ecdsa_secp256r1_sha256<br>TSIP_TLS13_SIGNATURE_SCHEME_RSA_PSS_RSAE_SHA256:<br>rsa_pss_rsae_sha256                                                                                                                                                       |
| digest                 | Input | Message hash calculated using SHA256<br>Calculate and input a hash value of the value of concatenated handshake messages such as (ClientHello  ServerHello  EncryptedExtensions  CertificateRequest  Certificate).<br>Use R_TSIP_Sha256Init/Update/Final to calculate the hash value and input as digest the value output by R_TSIP_Sha256Final. |
| certificate_verify     | Input | CertificateVerify<br>Input the start address of the buffer for storing the data in the format specified in RFC 8446 section 4.4.3, Certificate Verify.                                                                                                                                                                                           |
| certificate_verify_len | Input | Byte length of certificate_verify                                                                                                                                                                                                                                                                                                                |

##### Return Values

|                             |                                                                                                                                       |
|-----------------------------|---------------------------------------------------------------------------------------------------------------------------------------|
| TSIP_SUCCESS:               | Normal termination                                                                                                                    |
| TSIP_ERR_FAIL               | Internal error, or signature verification failure                                                                                     |
| TSIP_ERR_RESOURCE_CONFLICT: | Occurrence of resource conflict because a hardware resource needed by the processing routine was in use by another processing routine |
| TSIP_ERR_KEY_SET            | Invalid wrapped key input                                                                                                             |
| TSIP_ERR_PARAMETER          | Invalid input data                                                                                                                    |

##### Description

This API verifies a CertificateVerify message received from the server for use by the TLS 1.3 cooperation function. The algorithms used are ecdsa\_secp256r1\_sha256 and rsa\_pss\_rsae\_sha256.

**Reentrancy**

Not supported.

---

#### 4.2.15.17 R\_TSIP\_GenerateTls13SVP256EccKeyIndex

---

**Format**

```
#include "r_tsip_rx_if.h"
e_tsip_err_t R_TSIP_TlsGenerateTls13SVP256EccKeyIndex(
 tsip_tls13_handle_t *handle,
 e_tsip_tls13_mode_t mode,
 tsip_tls_p256_ecc_key_index_t *key_index,
 uint8_t *ephemeral_ecdh_public_key
)
```

**Parameters**

|                           |        |                                                                                                                                                                |
|---------------------------|--------|----------------------------------------------------------------------------------------------------------------------------------------------------------------|
| handle                    | Input  | Handle number (work area) indicating same session                                                                                                              |
| mode                      | Input  | Handshake protocol to use<br>TSIP_TLS13_MODE_FULL_HANDSHAKE:<br>Full handshake<br>TSIP_TLS13_MODE_RESUMPTION:<br>Resumption<br>TSIP_TLS13_MODE_0_RTT:<br>0-RTT |
| key_index                 | Output | Ephemeral ECC secret wrapped key<br>Use this value as the key_index parameter input to<br>R_TSIP_Tls13SVGenerateEcdheSharedSecret.                             |
| ephemeral_ecdh_public_key | Output | Ephemeral ECDH public key to be sent to the server<br>Qx (256 bits)    Qy (256 bits)                                                                           |

**Return Values**

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

**Description**

This API generates from a random number a key pair for use by the server function of the TLS 1.3 cooperation function in performing elliptic curve cryptography over a 256-bit prime field.

**Reentrancy**

Not supported.

#### 4.2.15.18 R\_TSIP\_Tls13SVGenerateEcdheSharedSecret

##### Format

```
#include "r_tsip_rx_if.h"
e_tsip_err_t R_TSIP_Tls13SVGenerateEcdheSharedSecret(
 e_tsip_tls13_mode_t mode,
 uint8_t *client_public_key,
 tsip_tls_p256_ecc_key_index_t *key_index,
 tsip_tls13_ephemeral_shared_secret_key_index_t *shared_secret_key_index
)
```

##### Parameters

| Parameter               | Direction | Description                                                                                                                                                                                                     |
|-------------------------|-----------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| mode                    | Input     | Handshake protocol to use<br>TSIP_TLS13_MODE_FULL_HANDSHAKE:<br>Full handshake<br>TSIP_TLS13_MODE_RESUMPTION:<br>Resumption<br>TSIP_TLS13_MODE_0_RTT:<br>0-RTT                                                  |
| client_public_key       | Input     | Public key provided by the client<br>Qx (256 bits)    Qy (256 bits)                                                                                                                                             |
| key_index               | Input     | Ephemeral ECC secret wrapped key<br>Use the value of key_index output by<br>R_TSIP_GenerateTls13SVP256EccKeyIndex.                                                                                              |
| shared_secret_key_index | Output    | Shared secret ephemeral wrapped key<br>Use this value as the shared_secret_key_index<br>parameter input to<br>R_TSIP_Tls13SVGenerateHandshakeSecret and<br>R_TSIP_Tls13SVGenerateResumptionHandshake<br>Secret. |

##### Return Values

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

##### Description

This API calculates a shared secret, which is a shared key over a 256-bit prime field, using a public key provided by the server and a previously calculated secret key, and generates a wrapped key for use by the server function of the TLS 1.3 cooperation function.

Applicable cypher suites: TLS\_AES\_128\_GCM\_SHA256, TLS\_AES\_128\_CCM\_SHA256

Key exchange format: ECDHE NIST P-256

##### Reentrancy

Not supported.



**4.2.15.19 R\_TSIP\_Tls13SVGenerateHandshakeSecret****Format**

```
#include "r_tsip_rx_if.h"
e_tsip_err_t R_TSIP_Tls13SVGenerateHandshakeSecret(
 tsip_tls13_ephemeral_shared_secret_key_index_t *shared_secret_key_index,
 tsip_tls13_ephemeral_handshake_secret_key_index_t *handshake_secret_key_index
)
```

**Parameters**

|                            |        |                                                                                                                                                                                                                                                         |
|----------------------------|--------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| shared_secret_key_index    | Input  | Shared secret ephemeral wrapped key<br>Use the value of shared_secret_key_index output by R_TSIP_Tls13SVGenerateEcdheSharedSecret.                                                                                                                      |
| handshake_secret_key_index | Output | Handshake secret ephemeral wrapped key<br>Use this value as the handshake_secret_key_index parameter input to R_TSIP_Tls13SVGenerateServerHandshakeTrafficKey, R_TSIP_Tls13SVGenerateClientHandshakeTrafficKey, and R_TSIP_Tls13SVGenerateMasterSecret. |

**Return Values**

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

**Description**

This API generates a handshake secret wrapped key, using a shared secret ephemeral key, for use by the server function of the TLS 1.3 cooperation function.

**Reentrancy**

Not supported.

**4.2.15.20 R\_TSIP\_Tls13SVGenerateServerHandshakeTrafficKey****Format**

```
#include "r_tsip_rx_if.h"
e_tsip_err_t R_TSIP_Tls13SVGenerateServerHandshakeTrafficKey(
 tsip_tls13_handle_t *handle,
 e_tsip_tls13_mode_t mode,
 tsip_tls13_ephemeral_handshake_secret_key_index_t *handshake_secret_key_index,
 uint8_t *digest,
 tsip_aes_key_index_t *server_write_key_index,
 tsip_hmac_sha_key_index_t *server_finished_key_index
)
```

**Parameters**

|                            |        |                                                                                                                                                                                                                                         |
|----------------------------|--------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| handle                     | Output | Handle number (work area) indicating same session                                                                                                                                                                                       |
| mode                       | Input  | Handshake protocol to use<br>TSIP_TLS13_MODE_FULL_HANDSHAKE:<br>Full handshake<br>TSIP_TLS13_MODE_RESUMPTION:<br>Resumption<br>TSIP_TLS13_MODE_0_RTT:<br>0-RTT                                                                          |
| handshake_secret_key_index | Input  | Handshake secret ephemeral wrapped key<br>Use the value of handshake_secret_key_index output by R_TSIP_Tls13SVGenerateHandshakeSecret or R_TSIP_Tls13SVGenerateResumptionHandshakeSecret.                                               |
| digest                     | Input  | Message hash calculated using SHA256<br>Calculate and input the hash value of (ClientHello  ServerHello).<br>Use R_TSIP_Sha256Init/Update/Final to calculate the hash value and input as digest the value output by R_TSIP_Sha256Final. |
| server_write_key_index     | Output | Server write key ephemeral wrapped key<br>Use this value as the key_index parameter input to R_TSIP_Tls13EncryptInit.                                                                                                                   |
| server_finished_key_index  | Output | Server finished key ephemeral wrapped key<br>Use this value as the key_index parameter input to R_TSIP_Sha256HmacGenerateInit.                                                                                                          |

**Return Values**

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

**Description**

This API generates, using the handshake secret output by R\_TSIP\_Tls13SVGenerateHandshakeSecret, a server write wrapped key and server finished wrapped key for use by the server function of the TLS 1.3 cooperation function.

**Reentrancy**

Not supported.

#### 4.2.15.21 R\_TSIP\_Tls13SVGenerateClientHandshakeTrafficKey

##### Format

```
#include "r_tsip_rx_if.h"
e_tsip_err_t R_TSIP_Tls13SVGenerateClientHandshakeTrafficKey(
 tsip_tls13_handle_t *handle,
 e_tsip_tls13_mode_t mode,
 tsip_tls13_ephemeral_handshake_secret_key_index_t *handshake_secret_key_index,
 uint8_t *digest,
 tsip_aes_key_index_t *client_write_key_index,
 tsip_tls13_ephemeral_client_finished_key_index_t *client_finished_key_index
)
```

##### Parameters

|                            |        |                                                                                                                                                                                                                                         |
|----------------------------|--------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| handle                     | Output | Handle number (work area) indicating same session                                                                                                                                                                                       |
| mode                       | Input  | Handshake protocol to use<br>TSIP_TLS13_MODE_FULL_HANDSHAKE:<br>Full handshake<br>TSIP_TLS13_MODE_RESUMPTION:<br>Resumption<br>TSIP_TLS13_MODE_0_RTT:<br>0-RTT                                                                          |
| handshake_secret_key_index | Input  | Handshake secret ephemeral wrapped key<br>Use the value of handshake_secret_key_index output by R_TSIP_Tls13SVGenerateHandshakeSecret or R_TSIP_Tls13SVGenerateResumptionHandshakeSecret.                                               |
| digest                     | Input  | Message hash calculated using SHA256<br>Calculate and input the hash value of (ClientHello  ServerHello).<br>Use R_TSIP_Sha256Init/Update/Final to calculate the hash value and input as digest the value output by R_TSIP_Sha256Final. |
| client_write_key_index     | Output | Client write key ephemeral wrapped key<br>Use this value as the key_index parameter input to R_TSIP_Tls13DecryptInit.                                                                                                                   |
| client_finished_key_index  | Output | Client finished key ephemeral wrapped key<br>Use this value as the client_finished_key_index parameter input to R_TSIP_Tls13SVClientHandshakeVerification.                                                                              |

##### Return Values

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

##### Description

This API generates, using the handshake secret output by R\_TSIP\_Tls13SVGenerateHandshakeSecret, a client write wrapped key and client finished wrapped key for use by the server function of the TLS 1.3 cooperation function.

**Reentrancy**

Not supported.

#### 4.2.15.22 R\_TSIP\_Tls13SVClientHandshakeVerification

##### Format

```
#include "r_tsip_rx_if.h"
e_tsip_err_t R_TSIP_Tls13SVClientHandshakeVerification(
 e_tsip_tls13_mode_t mode,
 tsip_tls13_ephemeral_client_finished_key_index_t *client_finished_key_index,
 uint8_t *digest,
 uint8_t *client_finished
)
```

##### Parameters

|                           |       |                                                                                                                                                                                                                                                                                                                                                                                                                     |
|---------------------------|-------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| mode                      | Input | Handshake protocol to use<br>TSIP_TLS13_MODE_FULL_HANDSHAKE:<br>Full handshake<br>TSIP_TLS13_MODE_RESUMPTION:<br>Resumption<br>TSIP_TLS13_MODE_0_RTT:<br>0-RTT                                                                                                                                                                                                                                                      |
| client_finished_key_index | Input | Client finished key ephemeral wrapped key<br>Use the value of client_finished_key_index output by R_TSIP_Tls13SVGenerateClientHandshakeTrafficKey                                                                                                                                                                                                                                                                   |
| digest                    | Input | Message hash calculated using SHA256<br>Calculate and input a hash value of the value of concatenated handshake messages such as (ClientHello  ServerHello  EncryptedExtensions  CertificateRequest  Certificate  CertificateVerify  ServerFinished  Certificate  CertificateVerify).<br>Use R_TSIP_Sha256Init/Update/Final to calculate the hash value and input as digest the value output by R_TSIP_Sha256Final. |
| client_finished           | Input | Finished information provided by the client<br>Input the start address of the buffer for storing the ClientFinished data obtained from R_TSIP_Tls13DecryptUpdate/Final.                                                                                                                                                                                                                                             |

##### Return Values

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

##### Description

This API verifies handshake messages, using the Finished information provided by the client, for use by the server function of the TLS 1.3 cooperation function.

If this API returns a value of TSIP\_ERR\_VERIFICATION\_FAIL, halt TLS communication including the verified handshake messages.

**Reentrancy**

Not supported.

**4.2.15.23 R\_TSIP\_Tls13SVGenerateMasterSecret****Format**

```
#include "r_tsip_rx_if.h"
e_tsip_err_t R_TSIP_Tls13SVGenerateMasterSecret(
 tsip_tls13_handle_t *handle,
 e_tsip_tls13_mode_t mode,
 tsip_tls13_ephemeral_handshake_secret_key_index_t *handshake_secret_key_index,
 tsip_tls13_ephemeral_master_secret_key_index_t *master_secret_key_index
)
```

**Parameters**

|                            |        |                                                                                                                                                                                                       |
|----------------------------|--------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| handle                     | Output | Handle number (work area) indicating same session                                                                                                                                                     |
| mode                       | Input  | Handshake protocol to use<br>TSIP_TLS13_MODE_FULL_HANDSHAKE:<br>Full handshake<br>TSIP_TLS13_MODE_RESUMPTION:<br>Resumption<br>TSIP_TLS13_MODE_0_RTT:<br>0-RTT                                        |
| handshake_secret_key_index | Input  | Handshake secret ephemeral wrapped key<br>Use the value of handshake_secret_key_index output by R_TSIP_Tls13SVGenerateHandshakeSecret.                                                                |
| master_secret_key_index    | Output | Master secret ephemeral wrapped key<br>Use this value as the master_secret_key_index parameter input to R_TSIP_Tls13SVGenerateApplicationTrafficKey and R_TSIP_Tls13SVGenerateResumptionMasterSecret. |

**Return Values**

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

**Description**

This API generates a master secret ephemeral wrapped key, using a handshake secret ephemeral key, for use by the server function of the TLS 1.3 cooperation function.

**Reentrancy**

Not supported.



**4.2.15.24 R\_TSIP\_Tls13SVGenerateApplicationTrafficKey****Format**

```
#include "r_tsip_rx_if.h"
e_tsip_err_t R_TSIP_Tls13SVGenerateApplicationTrafficKey(
 tsip_tls13_handle_t *handle,
 e_tsip_tls13_mode_t mode,
 tsip_tls13_ephemeral_master_secret_key_index_t *master_secret_key_index,
 uint8_t *digest,
 tsip_tls13_ephemeral_app_secret_key_index_t *server_app_secret_key_index,
 tsip_tls13_ephemeral_app_secret_key_index_t *client_app_secret_key_index,
 tsip_aes_key_index_t *server_write_key_index,
 tsip_aes_key_index_t *client_write_key_index
)
```

**Parameters**

|                             |              |                                                                                                                                                                                                                                                                                                                                                                                       |
|-----------------------------|--------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| handle                      | Input/output | Handle number (work area) indicating same session                                                                                                                                                                                                                                                                                                                                     |
| mode                        | Input        | Handshake protocol to use<br>TSIP_TLS13_MODE_FULL_HANDSHAKE:<br>Full handshake<br>TSIP_TLS13_MODE_RESUMPTION:<br>Resumption<br>TSIP_TLS13_MODE_0_RTT:<br>0-RTT                                                                                                                                                                                                                        |
| master_secret_key_index     | Input        | Master secret ephemeral wrapped key<br>Use the value of master_secret_key_index output by R_TSIP_Tls13SVGenerateMasterSecret.                                                                                                                                                                                                                                                         |
| digest                      | Input        | Message hash calculated using SHA256<br>Calculate and input a hash value of the value of concatenated handshake messages such as (ClientHello  ServerHello  EncryptedExtensions   CertificateRequest  Certificate  CertificateVerify   ServerFinished).<br>Use R_TSIP_Sha256Init/Update/Final to calculate the hash value and input as digest the value output by R_TSIP_Sha256Final. |
| server_app_secret_key_index | Output       | Server application traffic secret ephemeral wrapped key<br>Use this value as the input_app_secret_key_index parameter input to R_TSIP_Tls13SVUpdateApplicationTrafficKey.                                                                                                                                                                                                             |
| client_app_secret_key_index | Output       | Client application traffic secret ephemeral wrapped key<br>Use this value as the input_app_secret_key_index parameter input to R_TSIP_Tls13SVUpdateApplicationTrafficKey.                                                                                                                                                                                                             |
| server_write_key_index      | Output       | Server write key ephemeral wrapped key<br>Use this value as the key_index parameter input to R_TSIP_Tls13EncryptInit                                                                                                                                                                                                                                                                  |
| client_write_key_index      | Output       | Client write key ephemeral wrapped key<br>Use this value as the key_index parameter input to R_TSIP_Tls13DecryptInit.                                                                                                                                                                                                                                                                 |

**Return Values**

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

**Description**

This API generates an application traffic secret wrapped key, using a master secret ephemeral key, for use by the server function of the TLS 1.3 cooperation function. It also generates server write key and client write key ephemeral wrapped keys.

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

**Reentrancy**

Not supported.

**4.2.15.25 R\_TSIP\_Tls13SVUpdateApplicationTrafficKey****Format**

```
#include "r_tsip_rx_if.h"
e_tsip_err_t R_TSIP_Tls13SVUpdateApplicationTrafficKey(
 tsip_tls13_handle_t *handle,
 e_tsip_tls13_mode_t mode,
 e_tsip_tls13_update_key_type_t key_type,
 tsip_tls13_ephemeral_app_secret_key_index_t *input_app_secret_key_index,
 tsip_tls13_ephemeral_app_secret_key_index_t *output_app_secret_key_index,
 tsip_aes_key_index_t *app_write_key_index
)
```

**Parameters**

|                             |              |                                                                                                                                                                                                                                                                                                                                                        |
|-----------------------------|--------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| handle                      | Input/output | Handle number (work area) indicating same session                                                                                                                                                                                                                                                                                                      |
| mode                        | Input        | Handshake protocol to use<br>TSIP_TLS13_MODE_FULL_HANDSHAKE:<br>Full handshake<br>TSIP_TLS13_MODE_RESUMPTION:<br>Resumption<br>TSIP_TLS13_MODE_0_RTT:<br>0-RTT                                                                                                                                                                                         |
| key_type                    | Input        | Type of key to be updated<br>TSIP_TLS13_UPDATE_SERVER_KEY:<br>Server application traffic secret<br>TSIP_TLS13_UPDATE_CLIENT_KEY:<br>Client application traffic secret                                                                                                                                                                                  |
| input_app_secret_key_index  | Input        | Ephemeral wrapped key of input server/client application traffic secret<br>Use as input either<br>server/client_app_secret_key_index output by<br>R_TSIP_Tls13SVGenerateApplicationTrafficKey or<br>output_app_secret_key_index output by<br>R_TSIP_Tls13SVUpdateApplicationTrafficKey,<br>whichever matches the type of key specified by<br>key_type. |
| output_app_secret_key_index | Output       | Ephemeral wrapped key of output server/client application traffic secret<br>Output matching the type of key specified by<br>key_type is obtained.<br>Use this value as the input_app_secret_key_index<br>parameter input to<br>R_TSIP_Tls13SVUpdateApplicationTrafficKey.                                                                              |
| app_write_key_index         | Output       | Server/client write key ephemeral wrapped key<br>Output matching the type of key specified by<br>key_type is obtained.<br>Use ServerWriteKey as the key_index parameter<br>input to R_TSIP_Tls13EncryptInit.<br>Use ClientWriteKey as the key_index parameter<br>input to R_TSIP_Tls13DecryptInit.                                                     |

**Return Values**

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

**Description**

This API updates, using an application traffic secret, an encryption wrapped key corresponding to an application traffic secret wrapped key for use by the server function of the TLS 1.3 cooperation function.

**Reentrancy**

Not supported.

**4.2.15.26 R\_TSIP\_Tls13SVGenerateResumptionMasterSecret****Format**

```
#include "r_tsip_rx_if.h"
e_tsip_err_t R_TSIP_Tls13SVGenerateResumptionMasterSecret(
 tsip_tls13_handle_t *handle,
 e_tsip_tls13_mode_t mode,
 tsip_tls13_ephemeral_master_secret_key_index_t *master_secret_key_index,
 uint8_t *digest,
 tsip_tls13_ephemeral_res_master_secret_key_index_t *res_master_secret_key_index
)
```

**Parameters**

|                             |        |                                                                                                                                                                                                                                                                                                                                                                                                                                        |
|-----------------------------|--------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| handle                      | Input  | Handle number (work area) indicating same session                                                                                                                                                                                                                                                                                                                                                                                      |
| mode                        | Input  | Handshake protocol to use<br>TSIP_TLS13_MODE_FULL_HANDSHAKE:<br>Full handshake<br>TSIP_TLS13_MODE_RESUMPTION:<br>Resumption<br>TSIP_TLS13_MODE_0_RTT:<br>0-RTT                                                                                                                                                                                                                                                                         |
| master_secret_key_index     | Input  | Handshake secret ephemeral wrapped key<br>Use the value of handshake_secret_key_index output by R_TSIP_Tls13SVGenerateHandshakeSecret.                                                                                                                                                                                                                                                                                                 |
| digest                      | Input  | Message hash calculated using SHA256<br>Calculate and input a hash value of the value of concatenated handshake messages such as (ClientHello  ServerHello  EncryptedExtensions   CertificateRequest  Certificate  CertificateVerify   ServerFinished  Certificate  CertificateVerify   ClientFinished).<br>Use R_TSIP_Sha256Init/Update/Final to calculate the hash value and input as digest the value output by R_TSIP_Sha256Final. |
| res_master_secret_key_index | Output | Resumption master secret ephemeral wrapped key<br>Use this value as the res_master_secret_key_index parameter input to R_TSIP_Tls13SVGeneratePreSharedKey.                                                                                                                                                                                                                                                                             |

**Return Values**

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

**Description**

This API generates a resumption master secret wrapped key, using a master secret ephemeral key, for use by the server function of the TLS 1.3 cooperation function.

As specified in RFC 8446, delete `master_secret_key_index`, the master secret ephemeral wrapped key, after generating a resumption master secret wrapped key using this API.

**Reentrancy**

Not supported.

#### 4.2.15.27 R\_TSIP\_Tls13SVGeneratePreSharedKey

##### Format

```
#include "r_tsip_rx_if.h"
e_tsip_err_t R_TSIP_Tls13SVGeneratePreSharedKey(
 tsip_tls13_handle_t *handle,
 e_tsip_tls13_mode_t mode,
 tsip_tls13_ephemeral_res_master_secret_key_index_t *res_master_secret_key_index,
 uint8_t *ticket_nonce,
 uint32_t ticket_nonce_len,
 tsip_tls13_ephemeral_pre_shared_key_index_t *pre_shared_key_index
)
```

##### Parameters

|                             |        |                                                                                                                                                                                                                                               |
|-----------------------------|--------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| handle                      | Input  | Handle number (work area) indicating same session                                                                                                                                                                                             |
| mode                        | Input  | Handshake protocol to use<br>TSIP_TLS13_MODE_FULL_HANDSHAKE:<br>Full handshake<br>TSIP_TLS13_MODE_RESUMPTION:<br>Resumption<br>TSIP_TLS13_MODE_0_RTT:<br>0-RTT                                                                                |
| res_master_secret_key_index | Input  | Resumption master secret ephemeral wrapped key<br>Use the value of res_master_secret_key_index output by R_TSIP_Tls13SVGenerateResumptionMasterSecret.                                                                                        |
| ticket_nonce                | Input  | Ticket nonce provided by the server<br>If the size of the ticket nonce is not a multiple of 16 bytes, pad it with zeros to make it a multiple of 16 bytes before input.                                                                       |
| ticket_nonce_len            | Input  | Byte length of ticket nonce                                                                                                                                                                                                                   |
| pre_shared_key_index        | Output | Pre-shared key ephemeral wrapped key<br>Use this value as the pre_shared_key_index parameter input to R_TSIP_Tls13SVGeneratePskBinderKey, R_TSIP_Tls13SVGenerateResumptionHandshakeSecret, and R_TSIP_Tls13SVGenerate0RttApplicationWriteKey. |

##### Return Values

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

##### Description

This API generates, using a resumption master secret ephemeral key, a pre-shared wrapped key from new session ticket information for use by the server function of the TLS 1.3 cooperation function.

**Reentrancy**

Not supported.



**4.2.15.28 R\_TSIP\_Tls13SVGeneratePskBinderKey****Format**

```
#include "r_tsip_rx_if.h"
e_tsip_err_t R_TSIP_Tls13SVGeneratePskBinderKey(
 tsip_tls13_handle_t *handle,
 tsip_tls13_ephemeral_pre_shared_key_index_t *pre_shared_key_index,
 tsip_hmac_sha_key_index_t *psk_binder_key_index
)
```

**Parameters**

|                      |        |                                                                                                                                                                  |
|----------------------|--------|------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| handle               | Input  | Handle number (work area) indicating same session                                                                                                                |
| pre_shared_key_index | Input  | Pre-shared key ephemeral wrapped key<br>Use the value of pre_shared_key_index output by R_TSIP_Tls13SVGeneratePreSharedKey.                                      |
| psk_binder_key_index | Output | Psk binder key ephemeral wrapped key<br>Use this value to generate PskBinder.<br>Use this value as the key_index parameter input to R_TSIP_Sha256HmacVerifyInit. |

**Return Values**

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

**Description**

This API generates a binder wrapped key for use by the server function of the TLS 1.3 cooperation function.

**Reentrancy**

Not supported.

**4.2.15.29 R\_TSIP\_Tls13SVGenerateResumptionHandshakeSecret****Format**

```
#include "r_tsip_rx_if.h"
e_tsip_err_t R_TSIP_Tls13SVGenerateResumptionHandshakeSecret(
 tsip_tls13_handle_t *handle,
 e_tsip_tls13_mode_t mode,
 tsip_tls13_ephemeral_pre_shared_key_index_t *pre_shared_key_index,
 tsip_tls13_ephemeral_shared_secret_key_index_t *shared_secret_key_index,
 tsip_tls13_ephemeral_handshake_secret_key_index_t *handshake_secret_key_index
)
```

**Parameters**

|                            |        |                                                                                                                                                                                                                                                                     |
|----------------------------|--------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| handle                     | Input  | Handle number (work area) indicating same session                                                                                                                                                                                                                   |
| mode                       | Input  | Handshake protocol to use<br>TSIP_TLS13_MODE_FULL_HANDSHAKE:<br>Full handshake<br>TSIP_TLS13_MODE_RESUMPTION:<br>Resumption<br>TSIP_TLS13_MODE_0_RTT:<br>0-RTT                                                                                                      |
| pre_shared_key_index       | Input  | Pre-shared key ephemeral wrapped key<br>Use the value of pre_shared_key_index output by<br>R_TSIP_Tls13SVGeneratePreSharedKey.                                                                                                                                      |
| shared_secret_key_index    | Input  | Shared secret ephemeral wrapped key<br>Use the value of shared_secret_key_index output by<br>R_TSIP_Tls13SVGenerateEcdheSharedSecret.                                                                                                                               |
| handshake_secret_key_index | Output | Handshake secret ephemeral wrapped key<br>Use this value as the handshake_secret_key_index<br>parameter input to<br>R_TSIP_Tls13SVGenerateServerHandshakeTrafficKey,<br>R_TSIP_Tls13SVGenerateClientHandshakeTrafficKey,<br>and R_TSIP_Tls13SVGenerateMasterSecret. |

**Return Values**

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

**Description**

This API generates a handshake secret wrapped key, using the pre-shared wrapped key generated by R\_TSIP\_Tls13GeneratePreSharedKey, for use by the server function of the TLS 1.3 cooperation function.

Only pre-shared keys generated by the TSIP can be used. Other pre-shared keys are not supported.

**Reentrancy**

Not supported.

**4.2.15.30 R\_TSIP\_Tls13SVGenerate0RttApplicationWriteKey****Format**

```
#include "r_tsip_rx_if.h"
e_tsip_err_t R_TSIP_Tls13SVGenerate0RttApplicationWriteKey(
 tsip_tls13_handle_t *handle,
 tsip_tls13_ephemeral_pre_shared_key_index_t *pre_shared_key_index,
 uint8_t *digest,
 tsip_aes_key_index_t *client_write_key_index
)
```

**Parameters**

|                        |              |                                                                                                                                                                                                                          |
|------------------------|--------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| handle                 | Input/output | Handle number (work area) indicating same session                                                                                                                                                                        |
| pre_shared_key_index   | Input        | Pre-shared key ephemeral wrapped key<br>Use the value of pre_shared_key_index output by R_TSIP_Tls13SVGeneratePreSharedKey.                                                                                              |
| digest                 | Input        | Message hash calculated using SHA256<br>Calculate and input the hash value of ClientHello.<br>Use R_TSIP_Sha256Init/Update/Final to calculate the hash value and input as digest the value output by R_TSIP_Sha256Final. |
| client_write_key_index | Output       | Client write key ephemeral wrapped key<br>Use this value as the key_index parameter input to R_TSIP_Tls13DecryptInit.                                                                                                    |

**Return Values**

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

**Description**

This API generates a client write wrapped key for use as 0-RTT, using the pre-shared key generated by R\_TSIP\_Tls13GeneratePreSharedKey, for use by the server function of the TLS 1.3 cooperation function.

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

**Reentrancy**

Not supported.

### 4.2.15.31 R\_TSIP\_Tls13SVCertificateVerifyGenerate

#### Format

```
#include "r_tsip_rx_if.h"
e_tsip_err_t R_TSIP_Tls13SVCertificateVerifyGenerate(
 uint32_t *key_index,
 e_tsip_tls13_signature_scheme_type_t signature_scheme,
 uint8_t *digest,
 uint8_t *certificate_verify,
 uint32_t *certificate_verify_len
)
```

#### Parameters

|                        |        |                                                                                                                                                                                                                                                                                                                                                                                                    |
|------------------------|--------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| key_index              | Input  | Secret wrapped key for signature generation<br>Use the value of key_pair_index or key_index output by R_TSIP_GenerateEccP256PrivateKeyIndex, R_TSIP_GenerateEccP256RandomKeyIndex, R_TSIP_UpdateEccP256PrivateKeyIndex, R_TSIP_GenerateRsa2048PrivateKeyIndex, R_TSIP_GenerateRsa2048RandomKeyIndex, or R_TSIP_UpdateRsa2048PrivateKeyIndex.<br>Input this parameter after casting with uint32_t*. |
| signature_scheme       | Input  | Signature algorithm to be used<br>TSIP_TLS13_SIGNATURE_SCHEME_ECDSA_SECP256R1_SHA256:<br>ecdsa_secp256r1_sha256<br>TSIP_TLS13_SIGNATURE_SCHEME_RSA_PSS_RSAE_SHA256:<br>rsa_pss_rsae_sha256                                                                                                                                                                                                         |
| digest                 | Input  | Message hash calculated using SHA256<br>Calculate and input a hash value of the value of concatenated handshake messages such as (ClientHello  ServerHello  EncryptedExtensions   CertificateRequest  Certificate).<br>Use R_TSIP_Sha256Init/Update/Final to calculate the hash value and input as digest the value output by R_TSIP_Sha256Final.                                                  |
| certificate_verify     | Output | CertificateVerify<br>Data is output in the format specified in RFC 8446 section 4.4.3, Certificate Verify.                                                                                                                                                                                                                                                                                         |
| certificate_verify_len | Output | Byte length of certificate_verify                                                                                                                                                                                                                                                                                                                                                                  |

#### Return Values

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

#### Description

This API generates a CertificateVerify message to be sent to the client for use by the server function of the TLS 1.3 cooperation function. The algorithms used are ecdsa\_secp256r1\_sha256 and rsa\_pss\_rsae\_sha256.

**Reentrancy**

Not supported.

### 4.2.15.32 R\_TSIP\_Tls13SVCertificateVerifyVerification

#### Format

```
#include "r_tsip_rx_if.h"
e_tsip_err_t R_TSIP_Tls13SVCertificateVerifyVerification(
 uint32_t *key_index,
 e_tsip_tls13_signature_scheme_type_t signature_scheme,
 uint8_t *digest,
 uint8_t *certificate_verify,
 uint32_t certificate_verify_len
)
```

#### Parameters

|                        |       |                                                                                                                                                                                                                                                                                                                                                                                                  |
|------------------------|-------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| key_index              | Input | Encrypted public key<br>Use encrypted_output_public_key output by R_TSIP_TlsCertificateVerification or R_TSIP_TlsCertificateVerificationExtension.                                                                                                                                                                                                                                               |
| signature_scheme       | Input | Signature algorithm to be used<br>TSIP_TLS13_SIGNATURE_SCHEME_ECDSA_SECP256R1_SHA256:<br>ecdsa_secp256r1_sha256<br>TSIP_TLS13_SIGNATURE_SCHEME_RSA_PSS_RSAE_SHA256:<br>rsa_pss_rsae_sha256                                                                                                                                                                                                       |
| digest                 | Input | Message hash calculated using SHA256<br>Calculate and input a hash value of the value of concatenated handshake messages such as (ClientHello  ServerHello  EncryptedExtensions  CertificateRequest  Certificate  CertificateVerify  ServerFinished  Certificate).<br>Use R_TSIP_Sha256Init/Update/Final to calculate the hash value and input as digest the value output by R_TSIP_Sha256Final. |
| certificate_verify     | Input | CertificateVerify<br>Input the start address of the buffer for storing the data in the format specified in RFC 8446 section 4.4.3, Certificate Verify.                                                                                                                                                                                                                                           |
| certificate_verify_len | Input | Byte length of certificate_verify                                                                                                                                                                                                                                                                                                                                                                |

#### Return Values

|                             |                                                                                                                                       |
|-----------------------------|---------------------------------------------------------------------------------------------------------------------------------------|
| TSIP_SUCCESS:               | Normal termination                                                                                                                    |
| TSIP_ERR_FAIL               | Internal error, or signature verification failure                                                                                     |
| TSIP_ERR_RESOURCE_CONFLICT: | Occurrence of resource conflict because a hardware resource needed by the processing routine was in use by another processing routine |
| TSIP_ERR_KEY_SET            | Invalid wrapped key input                                                                                                             |
| TSIP_ERR_PARAMETER          | Invalid input data                                                                                                                    |

#### Description

This API verifies a CertificateVerify message received from the client for use by the server function of the TLS 1.3 cooperation function. The algorithms used are ecdsa\_secp256r1\_sha256 and rsa\_pss\_rsae\_sha256.

**Reentrancy**

Not supported.

### 4.2.15.33 R\_TSIP\_Tls13EncryptInit

#### Format

```
#include "r_tsip_rx_if.h"
e_tsip_err_t R_TSIP_Tls13EncryptInit(
 tsip_tls13_handle_t *handle,
 e_tsip_tls13_phase_t phase,
 e_tsip_tls13_mode_t mode,
 e_tsip_tls13_cipher_suite_t cipher_suite,
 tsip_aes_key_index_t *key_index,
 uint32_t payload_length
)
```

#### Parameters

|                |        |                                                                                                                                                                |
|----------------|--------|----------------------------------------------------------------------------------------------------------------------------------------------------------------|
| handle         | Output | TLS 1.3 handler (work area)                                                                                                                                    |
| phase          | Input  | Communication phase<br>TSIP_TLS13_PHASE_HANDSHAKE:<br>Handshake phase<br>TSIP_TLS13_PHASE_APPLICATION:<br>Application phase                                    |
| mode           | Input  | Handshake protocol to use<br>TSIP_TLS13_MODE_FULL_HANDSHAKE:<br>Full handshake<br>TSIP_TLS13_MODE_RESUMPTION:<br>Resumption<br>TSIP_TLS13_MODE_0_RTT:<br>0-RTT |
| cipher_suite   | Input  | Cypher suite<br>TSIP_TLS13_CIPHER_SUITE_AES_128_GCM_SHA256:<br>TLS_AES_128_GCM_SHA256<br>TSIP_TLS13_CIPHER_SUITE_AES_128_CCM_SHA256:<br>TLS_AES_128_CCM_SHA256 |
| key_index      | Input  | Ephemeral wrapped key of key used for encryption                                                                                                               |
| payload_length | Input  | Byte length of data to be encrypted                                                                                                                            |

#### Return Values

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

#### Description

The R\_TSIP\_Tls13EncryptInit() function performs preparations for the encryption of TLS 1.3 communication data and writes the result to the first parameter, handle. The parameter handle is used subsequently as a parameter by the R\_TSIP\_Tls13EncryptUpdate() and R\_TSIP\_Tls13EncryptFinal() functions.



**Reentrancy**

Not supported.

---

#### 4.2.15.34 R\_TSIP\_Tls13EncryptUpdate

---

**Format**

```
#include "r_tsip_rx_if.h"
e_tsip_err_t R_TSIP_Tls13EncryptUpdate(
 tsip_tls13_handle_t *handle,
 uint8_t *plain,
 uint8_t *cipher,
 uint32_t plain_length
)
```

**Parameters**

|              |              |                             |
|--------------|--------------|-----------------------------|
| handle       | Input/output | TLS 1.3 handler (work area) |
| plain        | Input        | Plaintext data area         |
| cipher       | Output       | Ciphertext data area        |
| plain_length | Input        | Plaintext data length       |

**Return Values**

|                            |                         |
|----------------------------|-------------------------|
| TSIP_SUCCESS:              | Normal termination      |
| TSIP_ERR_PARAMETER         | Invalid input data      |
| TSIP_ERR_PROHIBIT_FUNCTION | Invalid function called |

**Description**

The R\_TSIP\_Tls13EncryptUpdate() function encrypts the plaintext specified by the second parameter, plain, using the value specified for key\_index in R\_TSIP\_Tls13EncryptInit() function. The function internally buffers the data input by the user until the input value of plain exceeds 16 bytes. Once the input data from plain reaches 16 bytes or more, the encrypted result is output to the area specified by the third parameter, cipher. Specify as the payload\_length parameter of R\_TSIP\_Tls13EncryptInit() function the total data length of the data to be input as plain. For the plain\_length parameter of this function, specify the data length to be input when the user calls the function. If the input value of plain is not divisible by 16 bytes, the function performs padding internally.

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

**Reentrancy**

Not supported.

---

#### 4.2.15.35 R\_TSIP\_Tls13EncryptFinal

---

**Format**

```
#include "r_tsip_rx_if.h"
e_tsip_err_t R_TSIP_Tls13EncryptFinal(
 tsip_tls13_handle_t *handle,
 uint8_t *cipher,
 uint32_t *cipher_length
)
```

**Parameters**

|               |              |                             |
|---------------|--------------|-----------------------------|
| handle        | Input/output | TLS 1.3 handler (work area) |
| cipher        | Output       | Ciphertext data area        |
| cipher_length | Output       | Ciphertext data length      |

**Return Values**

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

**Description**

If there is 16-byte fractional remainder data indicated by the data length of the value of plain input to R\_TSIP\_Tls13EncryptUpdate() function, the R\_TSIP\_Tls13EncryptFinal() function outputs the result of encrypting the fractional remainder data to the area specified by the second parameter, cipher. At this time, if the data is less than 16 bytes, it is padded with zeros by the function internally.

**Reentrancy**

Not supported.

### 4.2.15.36 R\_TSIP\_Tls13DecryptInit

#### Format

```
#include "r_tsip_rx_if.h"
e_tsip_err_t R_TSIP_Tls13DecryptInit(
 tsip_tls13_handle_t *handle,
 e_tsip_tls13_phase_t phase,
 e_tsip_tls13_mode_t mode,
 e_tsip_tls13_cipher_suite_t cipher_suite,
 tsip_aes_key_index_t *key_index,
 uint32_t payload_length
)
```

#### Parameters

|                |        |                                                                                                                                                                |
|----------------|--------|----------------------------------------------------------------------------------------------------------------------------------------------------------------|
| handle         | Output | TLS 1.3 handler (work area)                                                                                                                                    |
| phase          | Input  | Communication phase<br>TSIP_TLS13_PHASE_HANDSHAKE:<br>Handshake phase<br>TSIP_TLS13_PHASE_APPLICATION:<br>Application phase                                    |
| mode           | Input  | Handshake protocol to use<br>TSIP_TLS13_MODE_FULL_HANDSHAKE:<br>Full handshake<br>TSIP_TLS13_MODE_RESUMPTION:<br>Resumption<br>TSIP_TLS13_MODE_0_RTT:<br>0-RTT |
| cipher_suite   | Input  | Cypher suite<br>TSIP_TLS13_CIPHER_SUITE_AES_128_GCM_SHA256:<br>TLS_AES_128_GCM_SHA256<br>TSIP_TLS13_CIPHER_SUITE_AES_128_CCM_SHA256:<br>TLS_AES_128_CCM_SHA256 |
| key_index      | Input  | Ephemeral wrapped key of key used for decryption                                                                                                               |
| payload_length | Input  | Byte length of data to be decrypted                                                                                                                            |

#### Return Values

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

#### Description

The R\_TSIP\_Tls13DecryptInit() function performs preparations for the decryption of TLS 1.3 communication data and writes the result to the first parameter, handle. The parameter handle is used subsequently as a parameter by the R\_TSIP\_Tls13DecryptUpdate() and R\_TSIP\_Tls13DecryptFinal() functions.

**Reentrancy**

Not supported.

**4.2.15.37 R\_TSIP\_Tls13DecryptUpdate****Format**

```
#include "r_tsip_rx_if.h"
e_tsip_err_t R_TSIP_Tls13DecryptUpdate(
 tsip_tls13_handle_t *handle,
 uint8_t *cipher,
 uint8_t *plain,
 uint32_t cipher_length
)
```

**Parameters**

|               |              |                             |
|---------------|--------------|-----------------------------|
| handle        | Input/output | TLS 1.3 handler (work area) |
| cipher        | Input        | Ciphertext data area        |
| plain         | Output       | Plaintext data area         |
| cipher_length | Input        | Ciphertext data length      |

**Return Values**

|                            |                         |
|----------------------------|-------------------------|
| TSIP_SUCCESS:              | Normal termination      |
| TSIP_ERR_PARAMETER         | Invalid input data      |
| TSIP_ERR_PROHIBIT_FUNCTION | Invalid function called |

**Description**

The R\_TSIP\_Tls13DecryptUpdate() function decrypts the ciphertext specified by the second parameter, cipher, using the value specified for key\_index in R\_TSIP\_Tls13DecryptInit() function. The function internally buffers the data input by the user until the input value of cipher exceeds 16 bytes. Once the input data from cypher reaches 16 bytes or more, the decrypted result is output to the area specified by the third parameter, plain. Specify as the payload\_length parameter of R\_TSIP\_Tls13DecryptInit() function the total data length of the data to be input as cipher. For the cipher\_length parameter of this function, specify the data length to be input when the user calls the function. If the input value of cipher is not divisible by 16 bytes, the function performs padding internally.

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

**Reentrancy**

Not supported.

**4.2.15.38 R\_TSIP\_Tls13DecryptFinal****Format**

```
#include "r_tsip_rx_if.h"
e_tsip_err_t R_TSIP_Tls13DecryptFinal(
 tsip_tls13_handle_t *handle,
 uint8_t *plain,
 uint32_t *plain_length
)
```

**Parameters**

|              |              |                             |
|--------------|--------------|-----------------------------|
| handle       | Input/output | TLS 1.3 handler (work area) |
| plain        | Output       | Plaintext data area         |
| plain_length | Output       | Plaintext data length       |

**Return Values**

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

**Description**

If the data length of cipher input in R\_TSIP\_Tls13DecryptUpdate() function results in a fractional remainder after 16 bytes, the R\_TSIP\_Tls13DecryptFinal() function outputs the leftover decrypted data to the second parameter, plain. At this time, if the data is less than 16 bytes, it is padded with zeros by the function internally. For plain, specify a RAM address that is a multiple of 4.

**Reentrancy**

Not supported.

---

## 4.2.16 Firmware Update

---

### 4.2.16.1 R\_TSIP\_StartUpdateFirmware

---

**Format**

e\_tsip\_err\_t R\_TSIP\_StartUpdateFirmware(void)

**Parameters**

None

**Return Values**

TSIP\_SUCCESS:

Normal termination

TSIP\_ERR\_RESOURCE\_CONFLICT:

Occurrence of resource conflict because a hardware resource needed by the processing routine was in use by another processing routine

**Description**

Transitions to the firmware update state.

**Reentrancy**

Not supported.



### 4.2.16.2 R\_TSIP\_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  | Wrapped key encryption key                                                                                                                                                                                                                                                             |
| InData_SessionKey                        | Input  | Encrypted image encryption key                                                                                                                                                                                                                                                         |
| InData_UpProgram                         | Input  | Area (one block size of code flash memory in the demo project) for temporary storage of encrypted firmware data                                                                                                                                                                        |
| InData_IV                                | Input  | Initialization vector area for decryption of encrypted firmware.                                                                                                                                                                                                                       |
| OutData_Program                          | Output | Area (one block size of code flash memory in the demo project) for temporary storage of decrypted firmware data                                                                                                                                                                        |
| MAX_CNT                                  | Input  | Encrypted firmware word size + MAC word size<br>The firmware word size must be a multiple of 4 words. The MAC size is fixed at 4 words (128 bits), so input the firmware word size + 4. The minimum size of the encrypted firmware is 16 words, so the minimum value of MAX_CNT is 20. |
| p_callback                               | Input  | This callback function is called multiple times when action by the user is required. The type of action is determined by the enum TSIP_FW_CB_REQ_TYPE.                                                                                                                                 |
| tsip_firmware_generate_mac_resume_handle | Input  | R_TSIP_GenerateFirmwareMAC handler (work area)                                                                                                                                                                                                                                         |

**Return Values**

|                                   |                                                                                                                                       |
|-----------------------------------|---------------------------------------------------------------------------------------------------------------------------------------|
| TSIP_SUCCESS:                     | Normal termination                                                                                                                    |
| TSIP_ERR_FAIL:                    | Occurrence of internal error                                                                                                          |
| TSIP_ERR_RESOURCE_CONFLICT:       | Occurrence of resource conflict because a hardware resource needed by the processing routine was in use by another processing routine |
| TSIP_ERR_KEY_SET                  | Invalid wrapped key input                                                                                                             |
| TSIP_ERR_CALLBACK_UNREGIST        | Invalid p_callback value                                                                                                              |
| TSIP_ERR_PARAMETER                | Invalid input data                                                                                                                    |
| TSIP_RESUME_FIRMWARE_GENERATE_MAC | There is additional processing. It is necessary to call the API again.                                                                |

**Description**

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

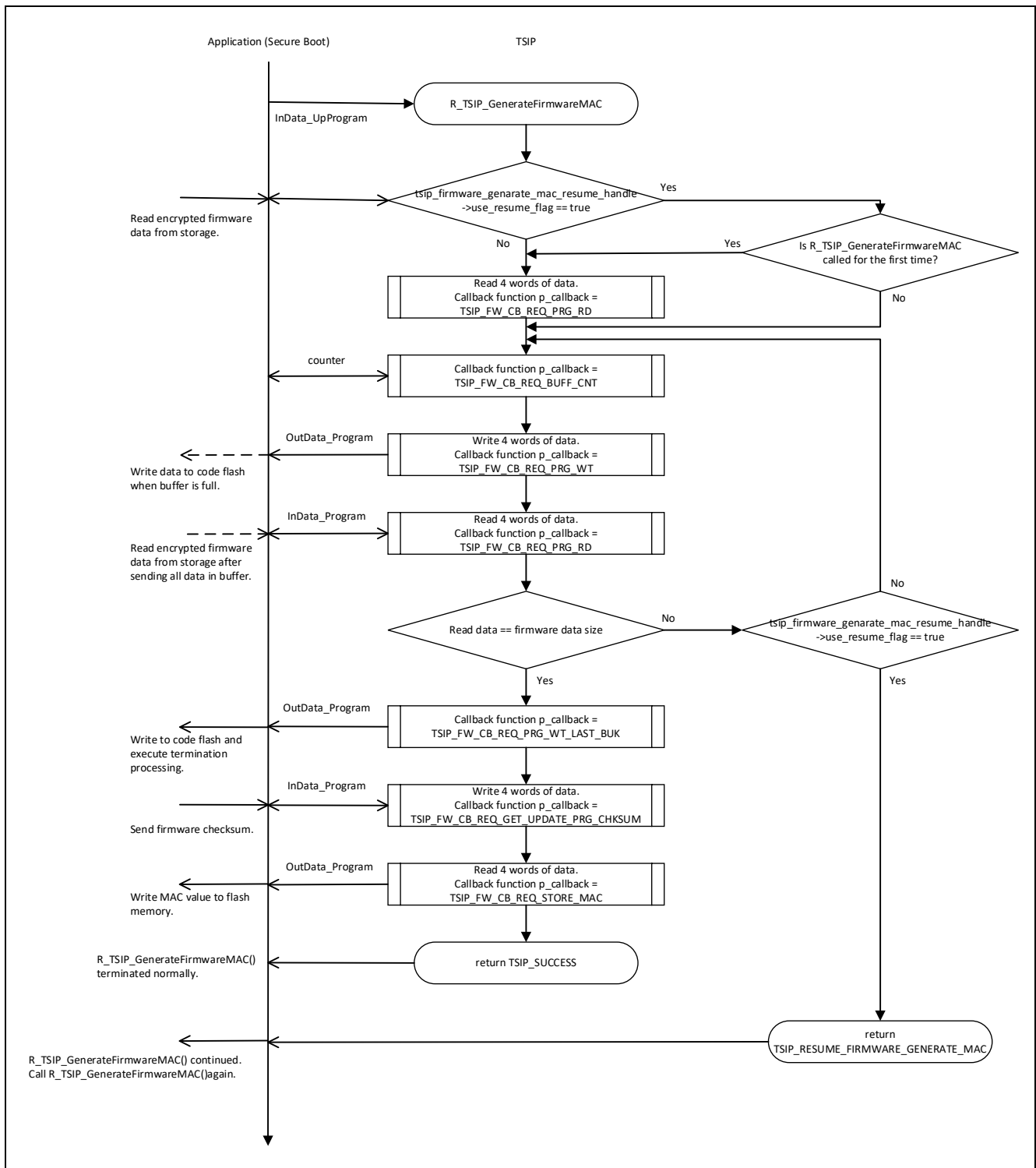


Figure 4.1 Flowchart of Calls to Callback Function

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

1. Increment adjustment (TSIP\_FW\_CB\_REQ\_BUFF\_CNT)
2. Writing of decrypted firmware to storage destination (TSIP\_FW\_CB\_REQ\_PRG\_WT)
3. Storage of encrypted firmware in InData\_UpProgram (TSIP\_FW\_CB\_REQ\_PRG\_RD)

It is not necessary to perform the processing in the callback function every time. Perform processing as appropriate for the sizes reserved for InData\_Program and OutData\_Program.

For example, if a 512-word buffer has been reserved, adjust the increment to match the buffer position on the  $512 / 4 = 128$ th time (TSIP\_FW\_CB\_REQ\_BUFF\_CNT), write to the storage destination (TSIP\_FW\_CB\_REQ\_PRG\_WT), and store the encrypted firmware in InData\_UpProgram (TSIP\_FW\_CB\_REQ\_PRG\_RD).

As the write request to the final storage destination, specify req\_type = TSIP\_FW\_CB\_REQ\_PRG\_WT\_LAST\_BLK (not TSIP\_FW\_CB\_REQ\_PRG\_WT).

This API is called again by the callback function p\_callback after reading and writing all of the firmware data has completed. After confirming that the value of the first parameter, req\_type, of the callback function p\_callback is TSIP\_FW\_CB\_REQ\_GET\_UPDATE\_PRG\_CHKSUM, pass the checksum value to the fourth parameter, InData\_UpProgram, of p\_callback. This API generates a firmware MAC value after reading and verifying the checksum value. After this, the MAC value is passed to the user using the fifth parameter, OutData\_Program, when the first parameter, req\_type, of callback function p\_callback is TSIP\_FW\_CB\_REQ\_STORE\_MAC. Store the MAC value in the flash memory area.

If called when tsip\_firmware\_generate\_mac\_resume\_handle->use\_resume\_flag is set to true, this API operates as a firmware update start and update function but does not perform the firmware update processing in its entirety. If there is additional processing remaining, a value of TSIP\_RESUME\_FIRMWARE\_GENERATE\_MAC is returned. Continue to call R\_TSIP\_GenerateFirmwareMAC() until a value of TSIP\_SUCCESS is returned. A return value of TSIP\_SUCCESS indicates that firmware update processing has completed successfully.

### Reentrancy

Not supported.

### 4.2.16.3 R\_TSIP\_VerifyFirmwareMAC

#### Format

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

#### Parameters

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

#### Return Values

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

#### Description

This function accepts firmware data and a MAC value, and verifies the MAC value. As the third parameter, InData\_Mac, pass the MAC value generated by R\_TSIP\_GenerateFirmwareMAC().

AES-CMAC is used as the MAC verification algorithm.

#### Reentrancy

Not supported.

#### 4.2.16.4 TSIP\_GEN\_MAC\_CB\_FUNC\_T Type

##### Format

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

##### Parameters

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

##### Return Values

None

##### Description

This function is used by the R\_TSIP\_GenerateFirmwareMAC function and is registered as its seventh parameter.

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

The size of the InData\_UpProgram and OutData\_Program areas must be a multiple of 4 and no fewer than 4 words. InData\_UpProgram and OutData\_Program should be the same size. The demo project uses the block size of code flash write unit.

This callback function is called by the R\_TSIP\_GenerateFirmwareMAC function for multiple types of requests. The type of request is stored in the first parameter, req\_type.

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

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

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

<req\_type = TSIP\_FW\_CB\_REQ\_PRG\_WT>

This is a request to store decrypted firmware.

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

The total decrypted data size is stored in the second parameter, `iLoop`. The maximum value of `iLoop` in this request is the value of the sixth parameter, `MAX_CNT`, minus 4 words. The last 4 words and the unstored firmware are handled by the request <req\_type = TSIP\_FW\_CB\_REQ\_PRG\_WT\_LAST\_BLK>.

<req\_type = TSIP\_FW\_CB\_REQ\_PRG\_RD>

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

The TSIP module issues this request each time before performing decryption in 4-word units. It works in the same manner as <req\_type = TSIP\_FW\_CB\_REQ\_PRG\_WT>. Store the decrypted firmware, as appropriate for the areas secured on the user side, in the fourth parameter, `InData_UpProgram`.

<req\_type = TSIP\_FW\_CB\_REQ\_BUFF\_CNT>

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

<req\_type = TSIP\_FW\_CB\_REQ\_PRG\_WT\_LAST\_BLK>

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

<req\_type = TSIP\_FW\_CB\_REQ\_GET\_UPDATE\_PRG\_CHKSUM>

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

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

<req\_type = req\_type = TSIP\_FW\_CB\_REQ\_STORE\_MAC>

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

The MAC is stored in the fifth parameter, OutData\_Program. The MAC size is 16 bytes.

The value of the sixth parameter, MAX\_CNT, is the same as that of the sixth parameter of R\_TSIP\_GenerateFirmwareMAC(), MAX\_CNT.



### 4.3 User-Defined Functions

This section describes the user-defined functions called by the TSIP driver.

#### 4.3.1 user\_sha384\_fucntion

##### Format

```
#include "r_tsip_rx_config.h"
uint32_t user_sha384_function (
 uint8_t *message,
 uint8_t *digest,
 uint32_t message_length)
```

##### Parameters

|                |        |                                                    |
|----------------|--------|----------------------------------------------------|
| message        | Input  | Start address of message                           |
| digest         | Output | Hash calculation result storage address (48 bytes) |
| message_length | Input  | Number of valid bytes in message                   |

##### Return Values

|              |                            |
|--------------|----------------------------|
| 0            | Hash value storage success |
| Other than 0 | Hash value storage failure |

##### Description

The TSIP does not support SHA-384 in hardware, so the following API requires the user to create an SHA-384 function for signature generation and verification. To use the API, enable TSIP\_USER\_SHA\_384\_ENABLED in r\_tsip\_rx\_config.h and prepare a function called user\_sha384\_function.

- R\_TSIP\_EcdsaP384SignatureGenerate
- R\_TSIP\_EcdsaP384SignatureVerification

This function can be defined when TSIP\_USER\_SHA\_384\_ENABLED is enabled in the configuration file. It performs an SHA-384 hash calculation for an area that starts from the address specified by the parameter message and extends for the number of bytes specified by the parameter message\_length.

Store the calculation result at the address specified by the parameter digest.

### 4.3.2 user\_lock\_fuction

---

**Format**

```
#include "r_tsip_rx_config.h"
void user_lock_function (
 void)
```

**Parameters**

None

**Return Values**

None

**Description**

To use the TSIP access conflict avoidance functionality described in section 3.2, the user must create a function to acquire exclusive control of a resource. Implement the function `user_lock_function` by enabling `TSIP_MULTI_THREADING` in `r_tsip_rx_config.h`.

To use access conflict avoidance functionality together with secure boot functionality, place this function in the secure boot area.

### 4.3.3 user\_unlock\_fucntion

---

**Format**

```
#include "r_tsip_rx_config.h"
void user_unlock_function (
 void)
```

**Parameters**

None

**Return Values**

None

**Description**

To use the TSIP access conflict avoidance functionality described in section 3.2, the user must create a function to acquire exclusive control of a resource. Implement the function `user_unlock_function` by enabling `TSIP_MULTI_THREADING` in `r_tsip_rx_config.h`.

To use access conflict avoidance functionality together with secure boot functionality, place this function in the secure boot area.

## 5. Key Injection and Updating

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

### 5.1 Key Injection

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

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

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

The process for implementing user key injection is as follows:

1. Preparing the key data necessary for user key injection

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

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

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

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

2. Creating a user key injection program

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

3. Key injection

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

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

## 5.2 Key Updating

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

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

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

1. Creating a user application incorporating key update functionality  
Create a user application incorporating key update functionality by making use of a key update API. Implementing key update functionality requires processing to receive the encrypted key using a communication interface, to convert it to a wrapped key using a key update API, and then to write it to the flash memory.  
Refer to 3.7.1, Key Injection and Updating, for an explanation of how to use key update APIs.
2. Injecting the KUK and user key into the device  
Refer to section 5.1 and create a user key injection program including the KUK, then inject the KUK and user key into the flash memory. It is recommended that the key injection data included in the user key injection program be deleted after key injection finishes.
3. Programming the user application program incorporating key update functionality to the device  
Use a programming method of your choice to program to user application program incorporating key update functionality to the flash memory.
4. Creating the user key data to be used for the update  
As described in section 3.7.1, which explains the user key wrapping scheme (Figure 3.3, User Key Wrapping Scheme during Key Injection and Key Updating), generate an encrypted key by wrapping the user key using the KUK.
5. Updating the user key  
Pass the encrypted key to the user application program incorporating key update functionality running on the RX Family MCU. The operation of the key update functionality of the user application program implements updating of the key stored in the on-chip flash memory of the RX Family MCU. Depending on the functionality implemented in the user application program, it is also possible to inject a new user key.

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



5.3.1.2 Procedure for GUI Version

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

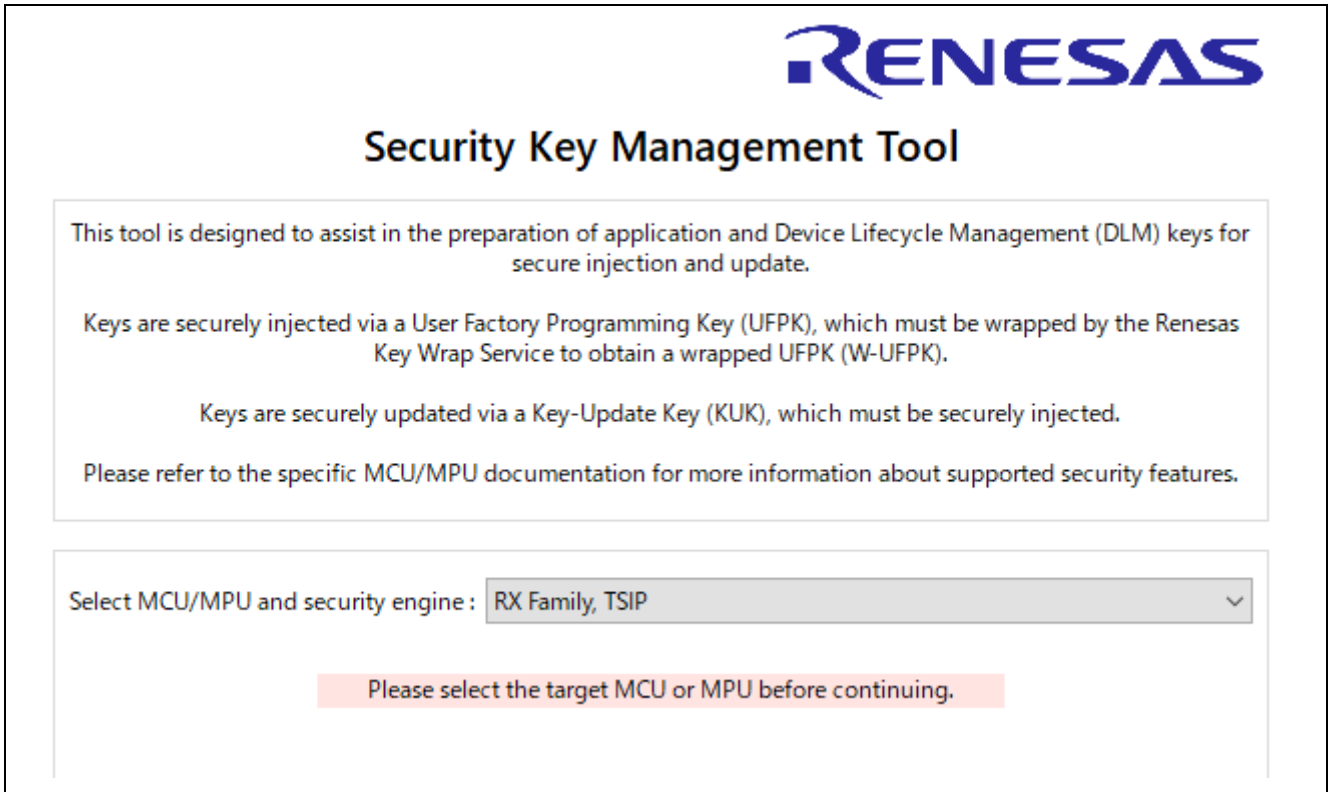


Figure 5.3 Overview Tab

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

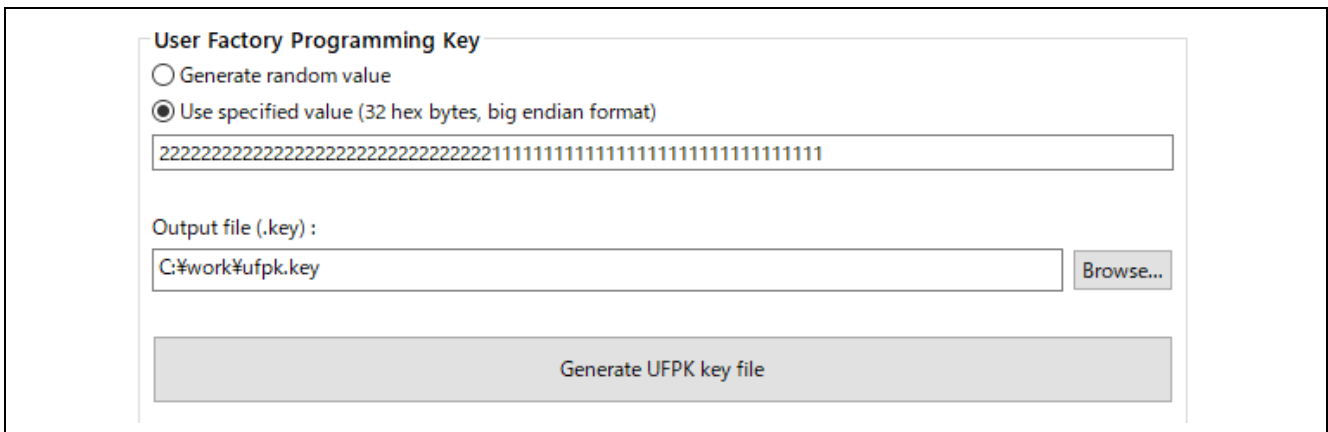


Figure 5.4 Example of UFPK Generation by Specifying Values on Generate UFPK Tab

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

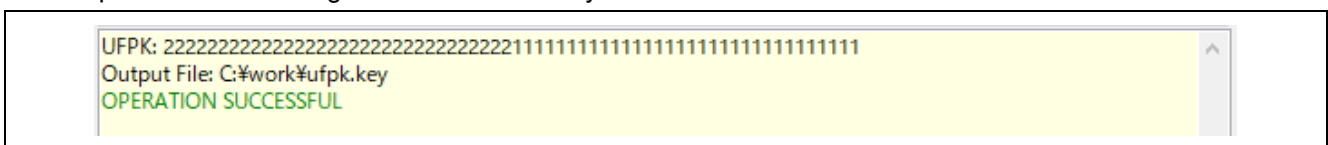


Figure 5.5 Generate UFPK Tab Execution Result

3. Obtaining a W-UFPK

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

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

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

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

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



The following is displayed when the operation completes successfully.

```

Output File: C:\work\%euk_aes128.h
Output File: C:\work\%euk_aes128.c
UFPK: 22211111111111111111111111111111111111
W-UFPK: 00000001DF741B6058AFDFE2DD186EA0AF168453F2B1F311ABAF6133AC68926B0A475130
IV: 55AA55AA55AA55AA55AA55AA55AA55AA
Encrypted key: 9206ACBC67367501A27EB48604AC77C6CB6CF84B4EBFF149B79DD6C0018FE0F9
OPERATION SUCCESSFUL

```

Figure 5.8 Wrap Key Tab Execution Result

Incorporate the data in the output C source file into your project with the same method in 5.3.1.1.

### 5.3.2 Key Update Operation Procedure

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

#### 5.3.2.1 Procedure for CLI Version

##### 1. Generating a UFPK

In a terminal emulator, run the following command:

```
> skmt.exe /genufpk
 /ufpk "ec6b8fa5c0d5da5142ccaf3a31aebeae2346cfe7ef644b9b6b70523cba0f5c5c"
 /output "C:\work\ufpk.key"
```

```
C:\work\skmt\tool>skmt.exe /genufpk /ufpk "ec6b8fa5c0d5da5142ccaf3a31aebeae2346cfe7ef644b9b6b70523cba0f5c5c" /output "C:\work\ufpk.key"
UFPK: EC6B8FA5C0D5DA5142CCAF3A31AEBEAE2346CFE7EF644B9B6B70523CBA0F5C5C
Output File: C:\work\ufpk.key
```

Figure 5.9 genufpk Command Execution Result

##### 2. Obtaining a W-UFPK

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

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

##### 3. Generating a KUK

In the terminal emulator, run the following genkuk command.

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

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

Figure 5.10 genkuk Command Execution Result

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

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

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

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

```
C:\work\skmt\tool>skmt.exe /genkey /ufpk file="c:\work\ufpk.key" /wufpk file="C:\work\ufpk.key_enc.key" /mcu "RX-TSIP" /keytype "key-update-key" /key file="C:\work\kuk.key" /filetype "csource" /output "C:\work\kuk.c"
Output File: C:\work\kuk.h
Output File: C:\work\kuk.c
UFPK: EC6B8FA5C0D5DA5142CCAF3A31AEBEAE2346CFE7EF644B9B6B70523CBA0F5C5C
W-UFPK: 0000001102D464621E307E4FF7027D346B9A2DEA8E35A6673DC9685DE0283CBED82DE1E
IV: 2F5FE32A536036EBC11BE0FA7BBBC572
Encrypted key: A3E508B735C32C4F122A931B78EE0F552FB42EEDA3CA1D955B1191CAF8FAC4D39462C5C1562
EB36EAB2D9331102DF699
```

Figure 5.11 genkey Command Execution Example

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

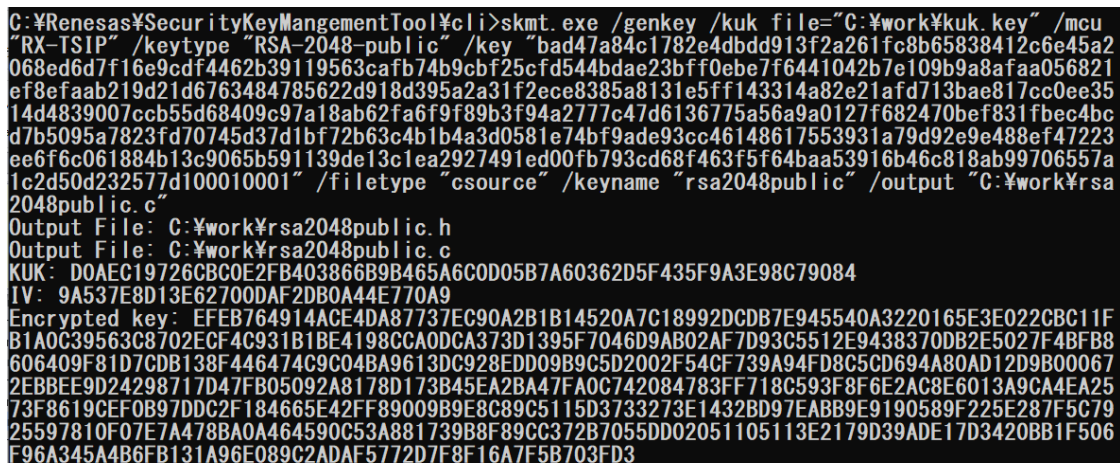
Next, the wrapping method using a KUK for key updating in the field is described.

5. Generating an encrypted file containing an RSA-2048 public key

In the terminal emulator, run the following `genkey` command.

```
> skmt.exe /genkey /kuk file="C:\work\kuk.key" /mcu "RX-TSIP" /keytype "RSA-2048-public"
/key "bad47a84c1782e4dbdd913f2a261fc8b65838412c6e45a2068ed6d7f16e9cdf4
462b39119563cafb74b9cbf25cfd544bdae23bff0ebe7f6441042b7e109b9a8a
faa056821ef8efaab219d21d6763484785622d918d395a2a31f2ece8385a8131
e5ff143314a82e21afd713bae817cc0ee3514d4839007ccb55d68409c97a18ab
62fa6f9f89b3f94a2777c47d6136775a56a9a0127f682470bef831fbec4bcd7b
5095a7823fd70745d37d1bf72b63c4b1b4a3d0581e74bf9ade93cc4614861755
3931a79d92e9e488ef47223ee6f6c061884b13c9065b591139de13c1ea292749
1ed00fb793cd68f463f5f64baa53916b46c818ab99706557a1c2d50d232577d1
00010001"
/filetype "csource" /keyname "rsa2048public"
/output "C:\work\rsa2048public.bin"
```

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



```
C:\Renesas\SecurityKeyManagementTool\cli>skmt.exe /genkey /kuk file="C:\work\kuk.key" /mcu
"RX-TSIP" /keytype "RSA-2048-public" /key "bad47a84c1782e4dbdd913f2a261fc8b65838412c6e45a2
068ed6d7f16e9cdf4462b39119563cafb74b9cbf25cfd544bdae23bff0ebe7f6441042b7e109b9a8a
faa056821ef8efaab219d21d6763484785622d918d395a2a31f2ece8385a8131e5ff143314a82e21afd713bae817cc0ee35
14d4839007ccb55d68409c97a18ab62fa6f9f89b3f94a2777c47d6136775a56a9a0127f682470bef831fbec4bc
d7b5095a7823fd70745d37d1bf72b63c4b1b4a3d0581e74bf9ade93cc46148617553931a79d92e9e488ef47223
ee6f6c061884b13c9065b591139de13c1ea2927491ed00fb793cd68f463f5f64baa53916b46c818ab99706557a
1c2d50d232577d100010001" /filetype "csource" /keyname "rsa2048public" /output "C:\work\rsa
2048public.c"
Output File: C:\work\rsa2048public.h
Output File: C:\work\rsa2048public.c
KUK: DOAEC19726CBC0E2FB403866B9B465A6C0D05B7A60362D5F435F9A3E98C79084
IV: 9A537E8D13E62700DAF2DB0A44E770A9
Encrypted key: EFEB764914ACE4DA87737EC90A2B1B14520A7C18992DCDB7E945540A3220165E3E022CBC11F
B1A0C39563C8702ECF4C931B1BE4198CCA0DCA373D1395F7046D9AB02AF7D93C5512E9438370DB2E5027F4BFB8
606409F81D7CDB138F446474C9C04BA9613DC928EDD09B9C5D2002F54CF739A94FD8C5CD69A80AD12D9B00067
2EBBEE9D24298717D47FB05092A8178D173B45EA2BA47FA0C742084783FF718C593F8F6E2AC8E6013A9CA4EA25
73F8619CEFB97DDC2F184665E42FF89009B9E8C89C5115D3733273E1432BD97EABB9E9190589F225E287F5C79
25597810F07E7A478BA0A464590C53A881739B8F89CC372B7055DD02051105113E2179D39ADE17D3420BB1F506
F96A345A4B6FB131A96E089C2ADAF5772D7F8F16A7F5B703FD3
```

Figure 5.12 `genkey` Command Execution Example

Input the data from the output C source file to the external interface of the device, and pass the data to the parameters of the `R_TSIP_Update2048PublicKeyIndex()` function to output an updated RSA-2048 public key index.

5.3.2.2 Procedure for GUI Version

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

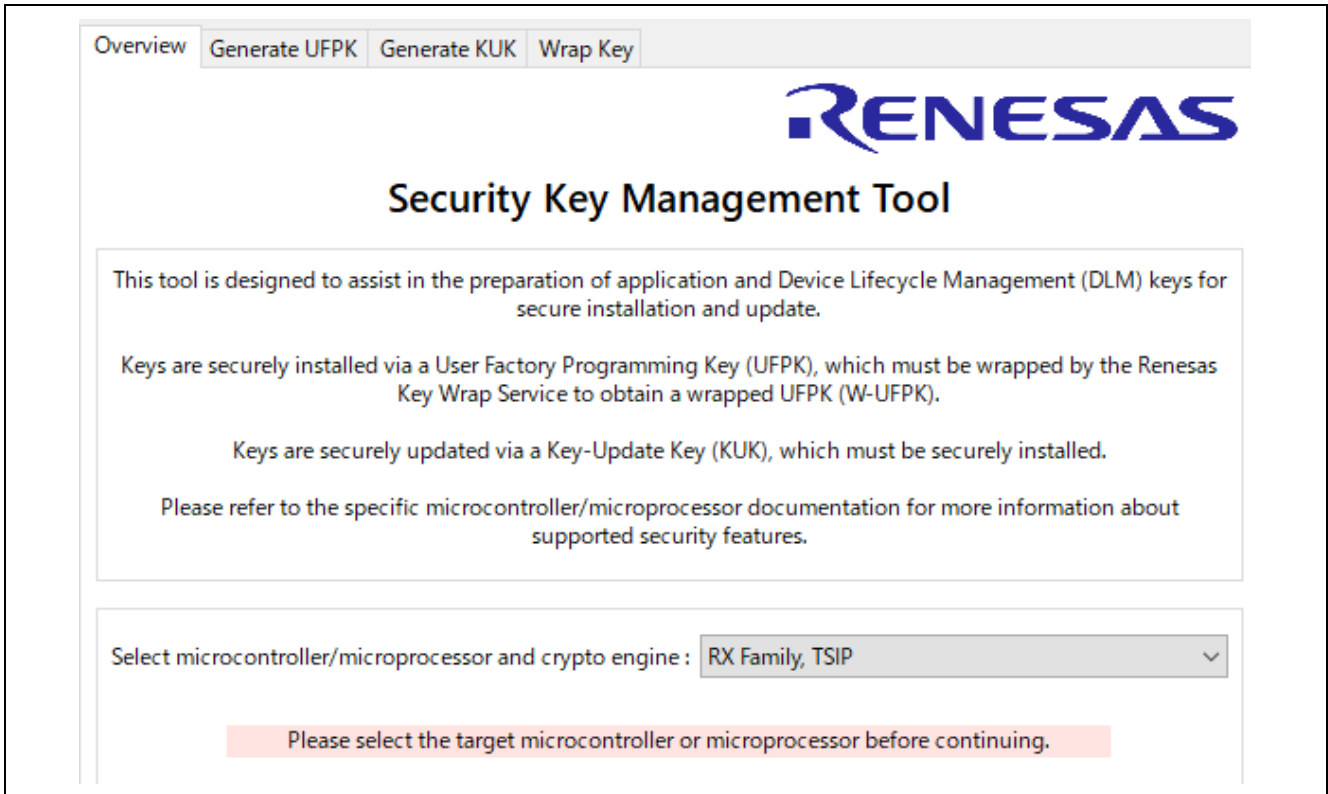


Figure 5.13 Overview Tab

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

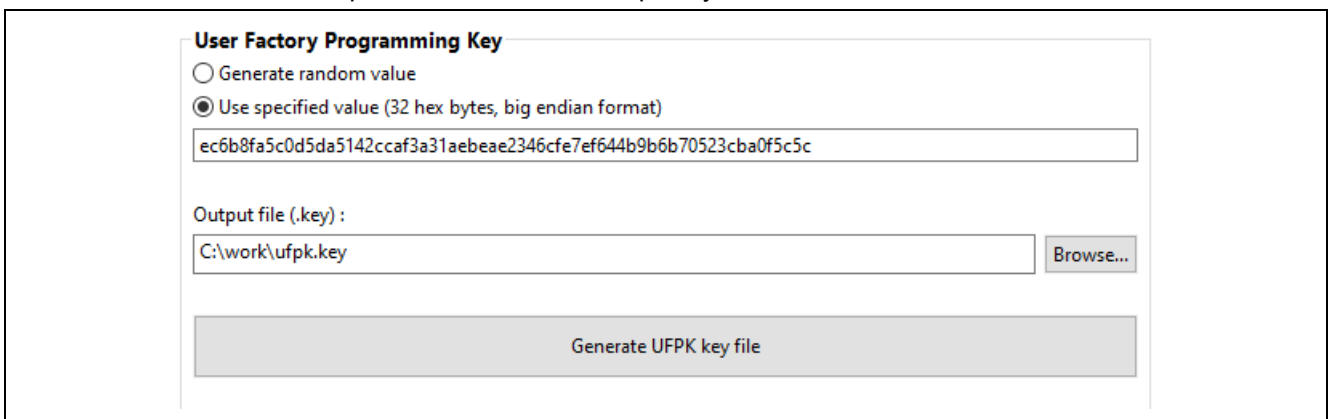


Figure 5.14 Example UFPK File Generation Settings on Generate UFPK Tab

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

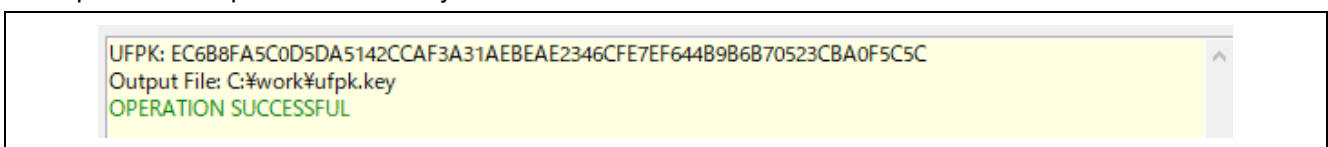


Figure 5.15 Generate UFPK Tab Execution Result

## 3. Obtaining a W-UFPK

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

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

## 4. Generating a KUK

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

**Figure 5.16 Example KUK File Generation Settings on Generate KUK Tab**

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

**Figure 5.17 Generate KUK Tab Execution Result**

5. Generating a KUK file in C source file format

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

The screenshot shows the 'Key Type' tab with the following settings:

- Key Type:** KUK (selected), DLM-SSD (128 bits), RSA (2048 bits, public), ECC (secp256r1, public), HMAC (SHA256-HMAC).
- Wrapping Key:** UFPK (selected). UFPK File: C:\work\ufpk.key. W-UFPK File: C:\work\ufpk.key\_enc.key.
- IV:** Generate random value (selected). Use specified value (16 hex bytes, big endian format): 00112233445566778899AABBCCDDEEFF.
- Output:** Format: C Source. File: C:\work\kuk.c. Address: 10000. Key name: kuk.

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

The screenshot shows the 'Key Type' tab with the following settings:

- Key Type:** File (selected). Key Data: C:\work\kuk.key.
- Wrapping Key:** UFPK (selected). UFPK File: C:\work\ufpk.key. W-UFPK File: C:\work\ufpk.key\_enc.key.
- IV:** Generate random value (selected). Use specified value (16 hex bytes, big endian format): 00112233445566778899AABBCCDDEEFF.
- Output:** Format: C Source. File: C:\work\kuk.c. Address: 10000. Key name: kuk.

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

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

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

6. Generating a RSA 2048 public key file in C source file format

On the Wrap Key tab, generate a RSA-2048 public key file in C source file format.

On the Key Type tab, select **RSA** and **2048 bits, public**, and on the Key Data tab, enter the RSA 2048 public key data.

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

The screenshot shows a software configuration window for generating a key file. It has two tabs: 'Key Type' and 'Key Data'. Under 'Key Type', there are radio buttons for DLM, KUK, AES, and ARC4. Under 'Key Data', there are radio buttons for DLM-SSD, TDES, RSA, ECC, and HMAC. The 'RSA' option is selected, and its dropdown menu shows '2048 bits, public'. Below this is the 'Wrapping Key' section with radio buttons for UFPK and KUK. The 'KUK' option is selected, and its 'KUK File' field contains 'C:\work\kuk.key'. Below that is the 'IV' section with radio buttons for 'Generate random value' and 'Use specified value (16 hex bytes, big endian format)'. The 'Generate random value' option is selected. Below that is the 'Output' section with a 'Format' dropdown set to 'C Source', a 'File' field containing 'C:\work\rsa2048public.c', an 'Address' field set to '10000', and a 'Key name' field containing 'rsa2048public'. A 'Generate file' button is at the bottom of the window.

Figure 5.20 Example RSA 2048 Public Key C Source File Generation Settings on Wrap Key – Key Type Tab

**Figure 5.21 Example RSA 2048 Public Key C Source File Generation Settings on Wrap Key – Key Data Tab**

The following is displayed when the operation completes successfully.

```

Output File: C:\work\rsa2048public.h
Output File: C:\work\rsa2048public.c
KUK: D0AEC19726CBC0E2FB403866B9B465A6C0D05B7A60362D5F435F9A3E98C79084
IV: D21D301AB01C209A87275774FE4BA29C
Encrypted key:
1BA6F73B0101ED42D54F9A8B4136F8099D1BD4F46919BA6B74144DF04A9E74E448BB18E8C1AC9F0847D9023024FAC
0B5BF723026BBD330B691DF7610F65B25137D71F064321E6982239E47F5D497A1CBD6CF688381442DB8889ACF1D74
F62D21607C2E7C6C5F74327C4BA804C71D2D4C4AB7FA5143E7BDB670668C443B9C4C5BDB2451F9416D54B651EE3A7
9158728CB42A3D244922BF539F1FE56035A775C8830D99936360FBF8B22E2AF58E735195E714A525F3939F3983FA2536
FBB50CC70B32FB2FFBB8233E8578005CAF7D2ABF64F5482F6447A64E3B55B10A5821E694328F5A5B38E7DBD56C01B4
160D96DA5FB677BC36AD4F5856DE335B8AD7707E1215D9E062DFF474E1EE240228B123925D10CA5909F10B72358E8E
BD5D7193032D
OPERATION SUCCESSFUL

```

**Figure 5.22 Wrap Key Tab Execution Result**

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



## 6. Sample Programs

### 6.1 Confirming Operation of the Demo Projects

With the exception of the AES cryptography project and TLS cooperation function project, the demo projects run on the MCUs listed in Table 6.1. The board which MCU is equipped is connected to the PC using a USB cable. Tera Term is used as the terminal emulator on the PC.

**Table 6-1 Demo Projects to used on each MCU**

| MCU   | Demo Project              |
|-------|---------------------------|
| RX231 | rx231_rsk_tsip_sample     |
| RX26T | rx26t_mcb_tsip_sample     |
| RX65N | rx65n_2mb_rsk_tsip_sample |
| RX66T | rx66t_rsk_tsip_sample     |
| RX671 | rx671_rsk_tsip_sample     |
| RX72M | rx72m_rsk_tsip_sample     |
| RX72N | rx72n_rsk_tsip_sample     |
| RX72T | rx72t_rsk_tsip_sample     |

Projects for MCUs which support Dual Bank mode make use of code flash linear mode.

Operation of the demo projects has been confirmed using little-endian byte ordering.

The description of demo project execution results, etc., in this section applies to the operation results on the RSK RX231 board. The terminal emulator software used is Tera Term.

#### Additional Notes

Demo projects for MCUs provided with a TSIP module require a link size larger than 128 KB. It is therefore necessary to purchase the product version of the CC-RX compiler in order to compile these projects.

There is no RSK currently available for the RX66N, so no demo projects are provided for this MCU. To confirm operation on the RX66N, refer to section 4 in the application note "RX Smart Configurator: User's Guide: e<sup>2</sup> studio" (R20AN0451ES0140) and change the selected device to the RX66N.

The RX72N (R5F572NNHDBD) and RX66N (R5F566NNHDBD) are pin-compatible, so equivalent operation can be achieved simply by changing the device.

The TSIP driver uses the same target folder, so operation is possible using the same UFPK.

### 6.1.1 Setting Up the Demo Project

Wiring connections between the PC and board are shown below.

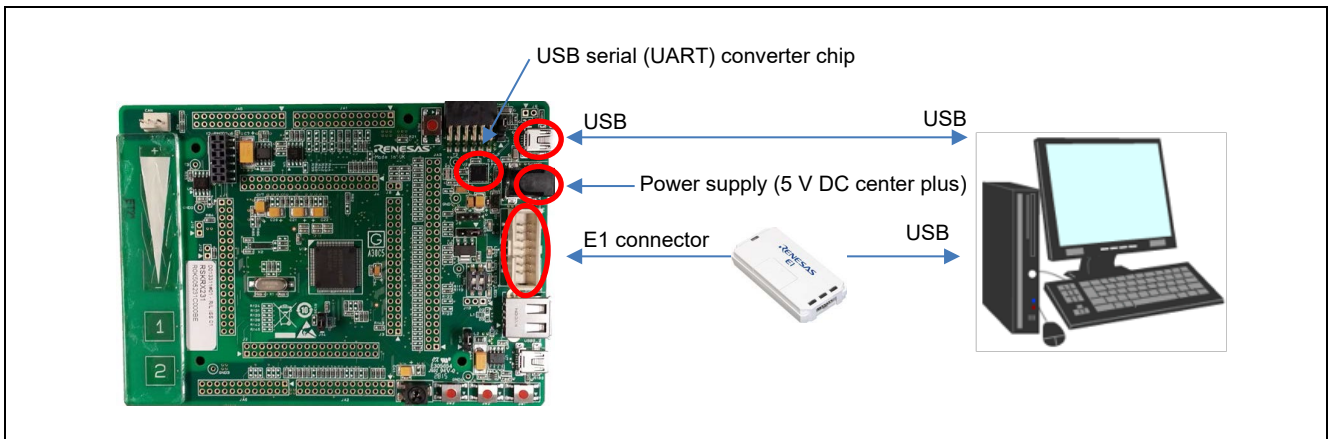


Figure 6.1 Wiring Connections between PC and RSK RX231 Board

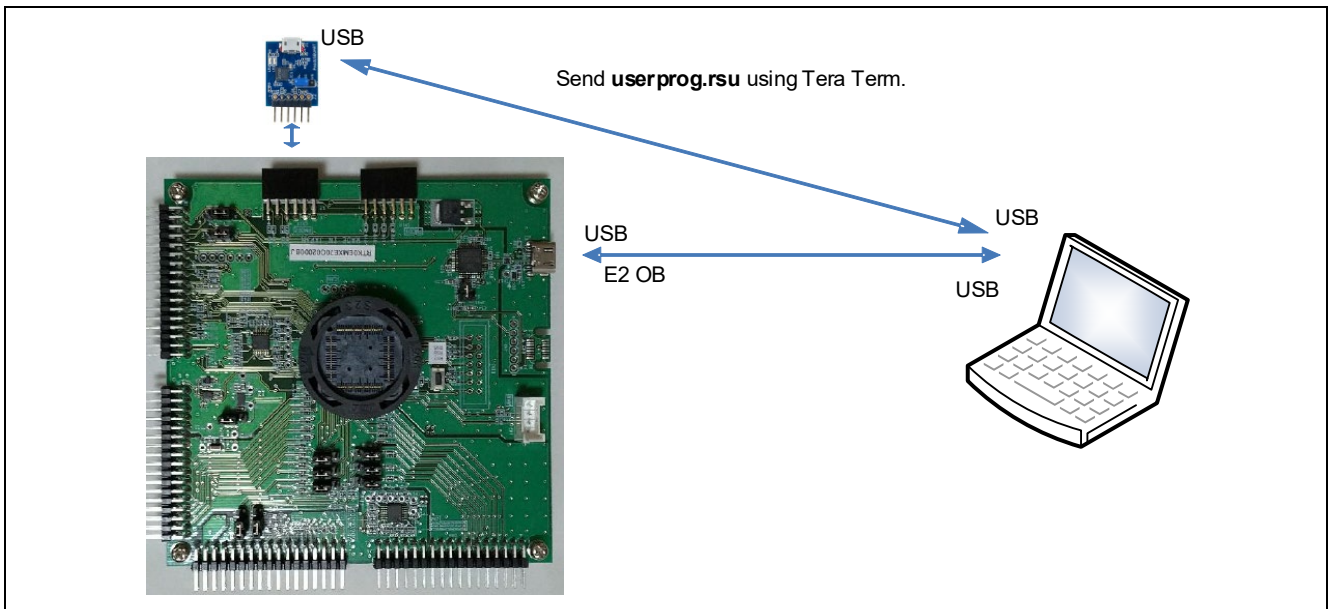


Figure 6.2 Wiring Connections between PC and MCB RX26T Board

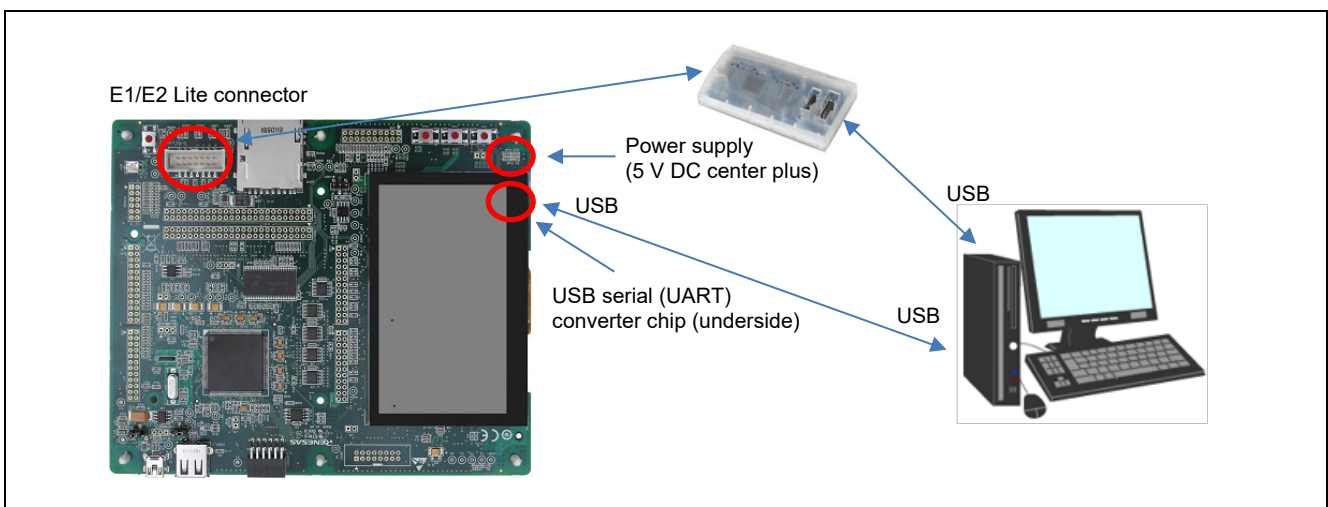


Figure 6.3 Wiring Connections between PC and RSK RX65N-2MB Board

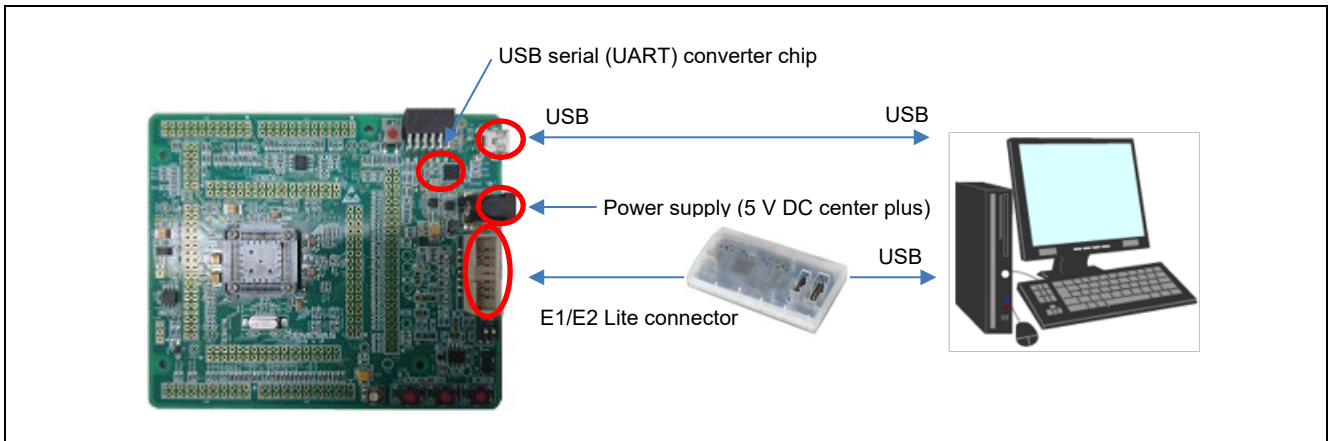


Figure 6.4 Wiring Connections between PC and RSK RX66T Board

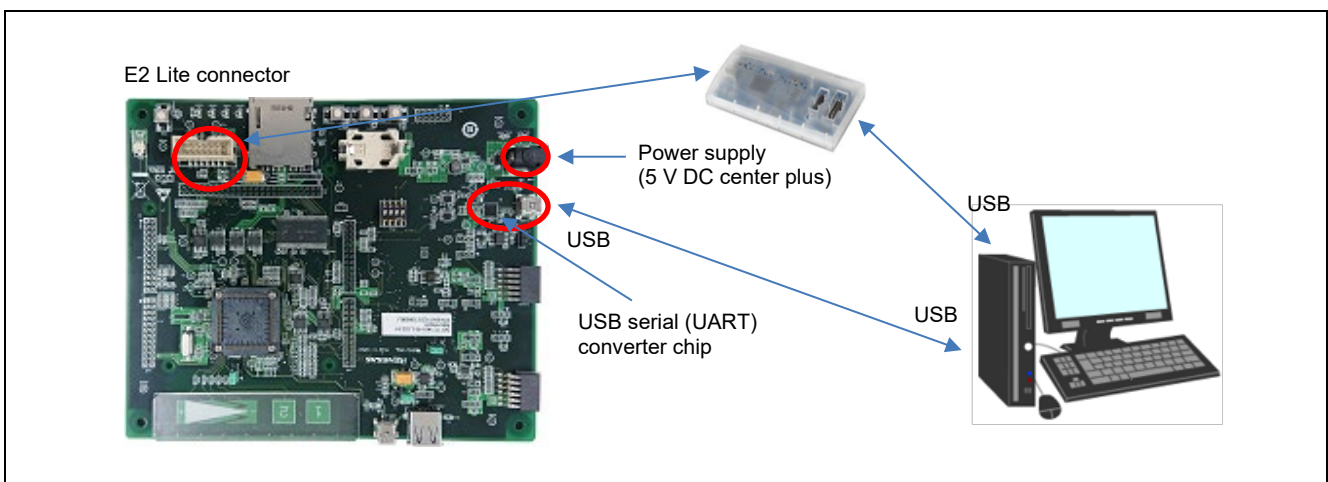


Figure 6.5 Wiring Connections between PC and RSK RX671 Board

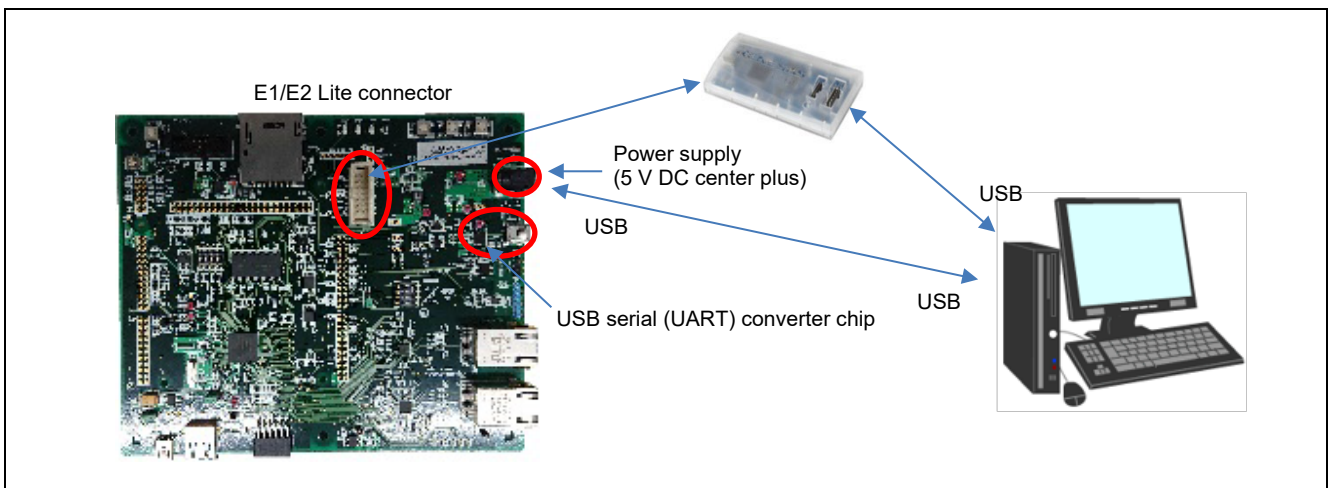


Figure 6.6 Wiring Connections between PC and RSK RX72M Board

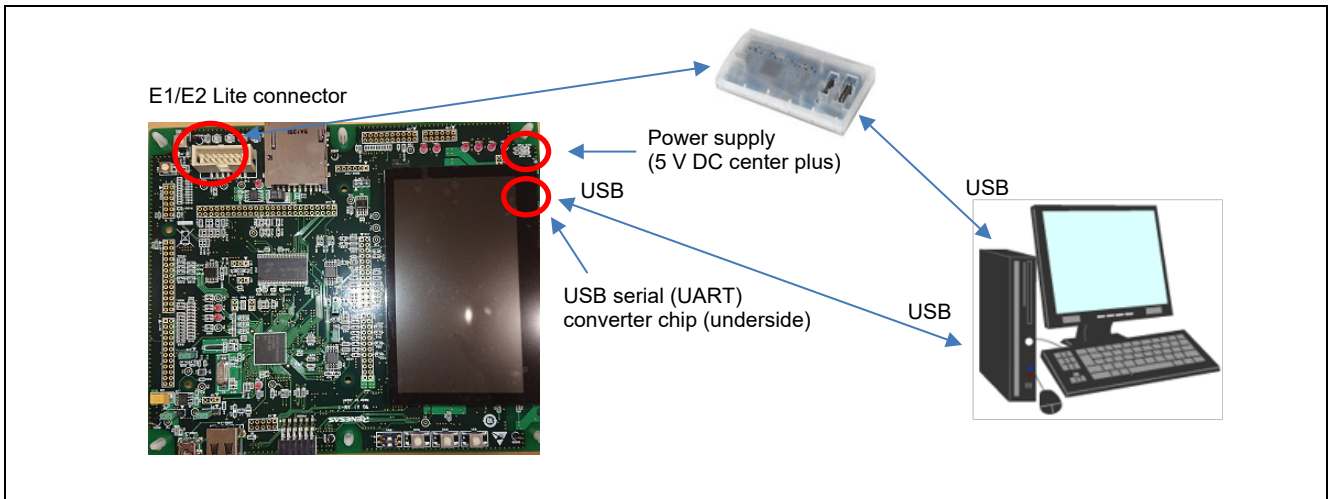


Figure 6.7 Wiring Connections between PC and RSK RX72N Board

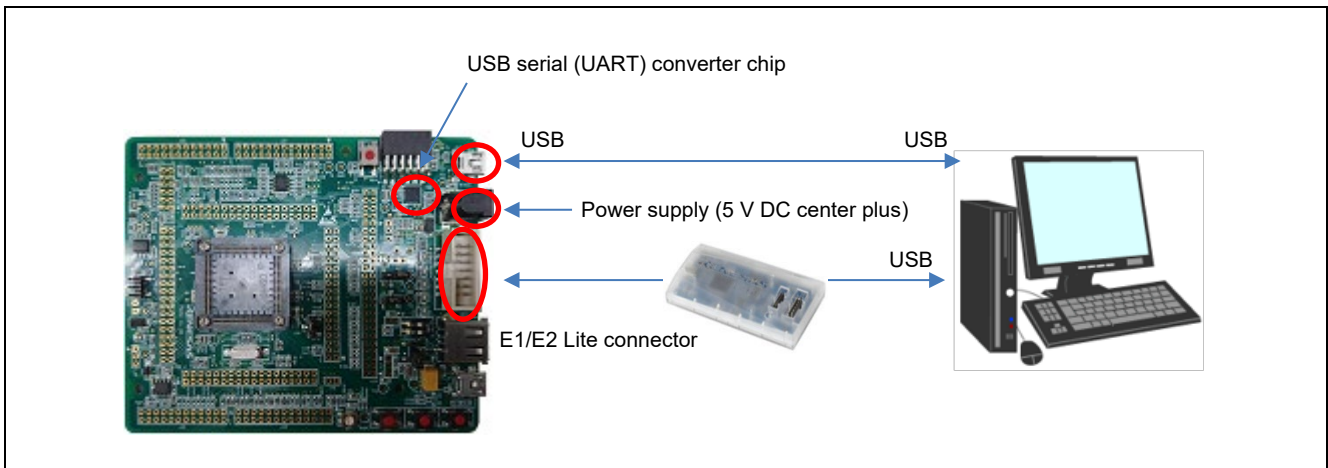


Figure 6.8 Wiring Connections between PC and RSK RX72T Board

The serial settings in Tera Term are as follows:

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

6.1.2 Overview of Demo Project

The key data structures used by the demo project are listed below.

Table 6-2 Key Data Structures of Demo Project

| Name                                  | Type         | Description                                                                                                                                                                                                         |
|---------------------------------------|--------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| st_key_block_data_t                   | —            | Key data structure located in C_FIRMWARE_UPDATE_CONTROL_BLOCK and C_FIRMWARE_UPDATE_CONTROL_BLOCK_MIRROR                                                                                                            |
| firmware_update_control_data          | —            | Structure for storing mac information during firmware updating                                                                                                                                                      |
| user_program_max_cnt                  | uint32_t     | Firmware byte size of firmware update                                                                                                                                                                               |
| lifecycle_state                       | uint32_t     | Key data and user program*1 status<br>There are three statuses:<br>LIFECYCLE_STATE_BLANK,<br>LIFECYCLE_STATE_UPDATING, and<br>LIFECYCLE_STATE_ON_THE_MARKET.                                                        |
|                                       |              | 1. LIFECYCLE_STATE_BLANK:                                                                                                                                                                                           |
|                                       | Key          | Injection incomplete.                                                                                                                                                                                               |
|                                       | User program | Injection incomplete.                                                                                                                                                                                               |
|                                       |              | 2. LIFECYCLE_STATE_UPDATING:                                                                                                                                                                                        |
|                                       | Key          | Injection complete, key data hash not set.                                                                                                                                                                          |
|                                       | User program | UPDATE_TEMPORARY_AREA:<br>Writing of update program complete.<br>UPDATE_FIRMWARE_AREA:<br>Program has not been updated.                                                                                             |
|                                       |              | 3. LIFECYCLE_STATE_ON_THE_MARKET:                                                                                                                                                                                   |
|                                       | Key          | Injection complete, key data hash setting complete, hash verification complete.                                                                                                                                     |
|                                       | User program | UPDATE_TEMPORARY_AREA:<br>Writing of update program complete.<br>UPDATE_FIRMWARE_AREA:<br>Copying of data from<br>UPDATE_TEMPORARY_AREA<br>complete. MAC verification complete.<br>User program operation possible. |
| program_mac0[]                        | uint32_t     | MAC value of UPDATE_FIRMWARE_AREA                                                                                                                                                                                   |
| program_mac1[]                        | uint32_t     | MAC value of UPDATE_TEMPORARY_AREA                                                                                                                                                                                  |
| key_data                              | —            | Key data storage structure                                                                                                                                                                                          |
| encrypted_user_aes128_key[]           | uint8_t      | Encrypted key of AES 128 shared key                                                                                                                                                                                 |
| encrypted_user_aes256_key[]           | uint8_t      | Encrypted key of AES 256 shared key                                                                                                                                                                                 |
| encrypted_user_tdes_key[]*2           | uint8_t      | Encrypted key of TDES shared key                                                                                                                                                                                    |
| encrypted_user_arc4_key[]*2           | uint8_t      | Encrypted key of ARC4 shared key                                                                                                                                                                                    |
| encrypted_user_sha1hmac_key[]*2       | uint8_t      | Encrypted key of SHA-1 HMAC shared key                                                                                                                                                                              |
| encrypted_user_sha256hmac_key[]*2     | uint8_t      | Encrypted key of SHA-256 HMAC shared key                                                                                                                                                                            |
| encrypted_user_rsa1024_ne_key[]*2     | uint8_t      | Encrypted key of RSA 1024 public key                                                                                                                                                                                |
| encrypted_user_rsa1024_nd_key[]*2     | uint8_t      | Encrypted key of RSA 1024 secret key                                                                                                                                                                                |
| encrypted_user_rsa2048_ne_key[]*2     | uint8_t      | Encrypted key of RSA 2048 public key                                                                                                                                                                                |
| encrypted_user_rsa2048_nd_key[]*2     | uint8_t      | Encrypted key of RSA 2048 secret key                                                                                                                                                                                |
| encrypted_user_ecc192_public_key[]*2  | uint8_t      | Encrypted key of ECC 192 public key                                                                                                                                                                                 |
| encrypted_user_ecc192_private_key[]*2 | uint8_t      | Encrypted key of ECC 192 secret key                                                                                                                                                                                 |
| encrypted_user_ecc224_public_key[]*2  | uint8_t      | Encrypted key of ECC 224 public key                                                                                                                                                                                 |
| encrypted_user_ecc224_private_key[]*2 | uint8_t      | Encrypted key of ECC 224 secret key                                                                                                                                                                                 |
| encrypted_user_ecc256_public_key[]    | uint8_t      | Encrypted key of ECC 256 public key                                                                                                                                                                                 |



| Name                                | Type                             | Description                                                  |
|-------------------------------------|----------------------------------|--------------------------------------------------------------|
| *2                                  |                                  |                                                              |
| encrypted_user_ecc256_private_key[] | uint8_t                          | Encrypted key of ECC 256 secret key                          |
| *2                                  |                                  |                                                              |
| encrypted_user_update_key[]         | uint8_t                          | Encrypted key of update keyring                              |
| encrypted_provisioning_key[]        | uint8_t                          | Encrypted provisioning key generated by DLM server           |
| iv[]                                | uint8_t                          | IV used for key injection                                    |
| user_aes128_key_index               | tsip_aes_key_index_t             | AES 128 key index                                            |
| user_aes256_key_index               | tsip_aes_key_index_t             | AES 256 key index                                            |
| user_tdes_key_index*2               | tsip_tdes_key_index_t            | TDES key index                                               |
| user_arc4_key_index*2               | tsip_arc4_key_index_t            | ARC4 key index                                               |
| user_sha1hmac_key_index*2           | tsip_hmac_sha_key_index_t        | SHA-1 HMAC key index                                         |
| user_sha256hmac_key_index*2         | tsip_hmac_sha_key_index_t        | SHA-256 HMAC key index                                       |
| user_rsa1024_ne_key_index*2         | tsip_rsa1024_public_key_index_t  | RSA 1024 public key index                                    |
| user_rsa1024_nd_key_index*2         | tsip_rsa1024_private_key_index_t | RSA 1024 secret key index                                    |
| user_rsa2048_ne_key_index*2         | tsip_rsa2048_public_key_index_t  | RSA 2048 public key index                                    |
| user_rsa2048_nd_key_index*2         | tsip_rsa2048_private_key_index_t | RSA 2048 secret key index                                    |
| user_ecc192_public_key_index*2      | tsip_ecc_public_key_index_t      | ECC 192 public key index                                     |
| user_ecc192_private_key_index*2     | tsip_ecc_private_key_index_t     | ECC 192 secret key index                                     |
| user_ecc224_public_key_index*2      | tsip_ecc_public_key_index_t      | ECC 224 public key index                                     |
| user_ecc224_private_key_index*2     | tsip_ecc_private_key_index_t     | ECC 224 secret key index                                     |
| user_ecc256_public_key_index*2      | tsip_ecc_public_key_index_t      | ECC 256 public key index                                     |
| user_ecc256_private_key_index*2     | tsip_ecc_private_key_index_t     | ECC 256 secret key index                                     |
| hash_sha1[]                         | uint8_t                          | SHA1 hash value of firmware_update_control_data and key_data |

Notes: 1. Key data status only in the demo project.

The user program status is used by the secure boot project (see 6.2 Confirming Operation of the Secure Bootloader and Firmware Update Demo Projects).

- When a device provided with a TSIP-Lite module is used, TDES, ARC4, HMAC, RSA, and ECC key information is not used. In addition, this data is not present in the key\_data.c and key\_data.h files appended to the demo project versions for devices provided with a TSIP-Lite module. The key\_data.c and key\_data.h files appended to the demo project for the RX671 do not contain ARC4 and RSA-2048 data.

Main data and mirror data blocks are provided in the data flash for the demo project’s key information structures. (In the figure below, the main data block is C\_FIRMWARE\_UPDATE\_CONTROL\_BLOCK and the mirror data block is C\_FIRMWARE\_UPDATE\_CONTROL\_BLOCK\_MIRROR.) This prevents loss of the key information in case of a write failure due to circumstances such as a power interruption while updating the key information.

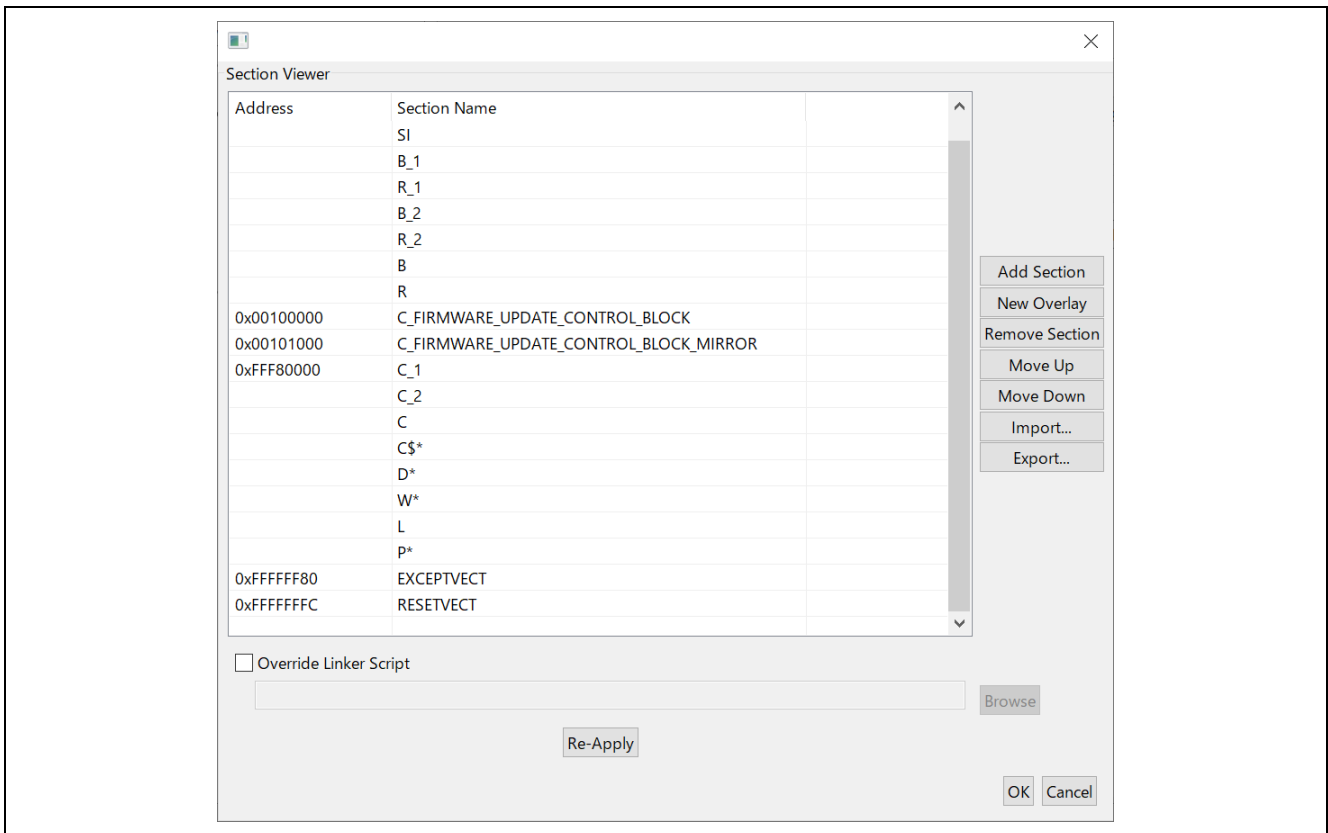


Figure 6.9 RX231 Demo Project Sections

In key\_data.c and key\_data.h, definition file of encrypted key in “demo project folder”/src/genkey is included. Refer to 5.3.1 Key Injection Procedure to see how to generate the files.

The demo project implements the following commands.

**Table 6-3 Demo Project Commands**

| Command        | Operation                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
|----------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| display        | Displays the wrapped key.                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
| encdemo [Arg1] | Encrypts the Arg1 value in AES 128-bit mode.                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| function       | Performs the following tests:*1<br>AES-128/256 ECB, CBC, CTR, GCM, the CCM encryption and decryption<br>AES-128/256 CMAC and SHA-HMAC generation and verification<br>TDES encryption and decryption<br>ARC4 encryption and decryption<br>(RSA-1024 and RSA-2048 only) RSAES-PKCS1_v1_5 encryption and decryption<br>(RSA-1024 and RSA-2048 only) RSASSA-PKCS1_v1_5 signature generation and verification<br>ECDSA signature generation and verification tests (excluding P-384) |
| random         | Generates a random value using R_TSIP_GenerateRandomNumber().                                                                                                                                                                                                                                                                                                                                                                                                                   |

Note: 1. SHA-HMAC generation and verification, TDES encryption and decryption, ARC4 encryption and decryption, RSAES-PKCS1\_v1\_5 encryption and decryption, RSASSA-PKCS1\_v1\_5 signature generation and verification, and ECDSA signature generation and verification are not implemented in the demo projects for devices provided with a TSIP-Lite module. And ARC4 and RSA-2048 are not supported in RX671 demo project.

The default values of keys are as follows and IV is "55aa55aa55aa55aa55aa55aa55aa55aa" in the demo project.

**Table 6-4 Demo Project Default Key Values**

| Key Type                     | Value                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
|------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| UFPK                         | 222222222222222222222222222211111111111111111111111111111111111111111111111111111111111                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
| AES 128-bit                  | 111111111222222222333333333444444444                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| AES 256-bit                  | ffffffffeeeeeddddcccccbbbbbbaaaaaaa000000099999999                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 3 TDES*2                     | 010101010101010102020202020202020404040404040404                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| ARC4 2048-bit*2              | 29041972fb42ba5fc7127712f13829c929041972fb42ba5fc7127712f13829c9<br>29041972fb42ba5fc7127712f13829c929041972fb42ba5fc7127712f13829c9<br>29041972fb42ba5fc7127712f13829c929041972fb42ba5fc7127712f13829c9<br>29041972fb42ba5fc7127712f13829c929041972fb42ba5fc7127712f13829c9<br>29041972fb42ba5fc7127712f13829c929041972fb42ba5fc7127712f13829c9<br>29041972fb42ba5fc7127712f13829c929041972fb42ba5fc7127712f13829c9<br>29041972fb42ba5fc7127712f13829c929041972fb42ba5fc7127712f13829c9<br>29041972fb42ba5fc7127712f13829c929041972fb42ba5fc7127712f13829c9 |
| SHA-1 HMAC*2                 | 000102030405060708090a0b0c0d0e0f10111213                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| SHA-256 HMAC*2               | 000102030405060708090a0b0c0d0e0f101112131415161718191a1b1c1d1e1f                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| RSA 1024-bit public<br>*1*2  | ccd6cb86f59ffea97c278b7cf395fa56f3709a958bce1e6e3ae196b471f4517f<br>64d32d81d10bcea5b55b9ea659c3b9db1854a696b801a8e72439265e85bc6138<br>cf874f45fe22a2477fe8337671357db67180b1ac9a0e84c098dbcdca8a3e6a72<br>dfcee7e6907ede2d53aa726e8e97ea26fcb3bbfa7bf7fbfbb70f1ec1d153125<br>00010001                                                                                                                                                                                                                                                                      |
| RSA 1024-bit private<br>*1*2 | ccd6cb86f59ffea97c278b7cf395fa56f3709a958bce1e6e3ae196b471f4517f<br>64d32d81d10bcea5b55b9ea659c3b9db1854a696b801a8e72439265e85bc6138<br>cf874f45fe22a2477fe8337671357db67180b1ac9a0e84c098dbcdca8a3e6a72<br>dfcee7e6907ede2d53aa726e8e97ea26fcb3bbfa7bf7fbfbb70f1ec1d153125<br>096ed2cc923f1df11c208e11e0fdc51b7ff66d87f97a32788d099a7110d65972<br>6e68332e493c2bf6019608864c97f0d52017b5dc36f90c982858e16574ef29e2<br>e8b6f4386bdbb1beae60390aac0160ce787f98ba34c83d6612badbc99d17666<br>04e72b394d1991fc4ded87e11cf165caae0cc652771b8395ed5b9a99a9fa5781   |



| Key Type                                      | Value                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
|-----------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| RSA 2048-bit public<br>*1*2                   | bad47a84c1782e4dbdd913f2a261fc8b65838412c6e45a2068ed6d7f16e9cdf4<br>462b39119563cafb74b9cbf25cfd544bdae23bff0ebe7f6441042b7e109b9a8a<br>faa056821ef8efaab219d21d6763484785622d918d395a2a31f2ece8385a8131<br>e5ff143314a82e21afd713bae817cc0ee3514d4839007ccb55d68409c97a18ab<br>62fa6f9f89b3f94a2777c47d6136775a56a9a0127f682470bef831fbec4bcd7b<br>5095a7823fd70745d37d1bf72b63c4b1b4a3d0581e74bf9ade93cc4614861755<br>3931a79d92e9e488ef47223ee6f6c061884b13c9065b591139de13c1ea292749<br>1ed00fb793cd68f463f5f64baa53916b46c818ab99706557a1c2d50d232577d1<br>00010001                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| RSA 2048-bit private<br>*1*2                  | bad47a84c1782e4dbdd913f2a261fc8b65838412c6e45a2068ed6d7f16e9cdf4<br>462b39119563cafb74b9cbf25cfd544bdae23bff0ebe7f6441042b7e109b9a8a<br>faa056821ef8efaab219d21d6763484785622d918d395a2a31f2ece8385a8131<br>e5ff143314a82e21afd713bae817cc0ee3514d4839007ccb55d68409c97a18ab<br>62fa6f9f89b3f94a2777c47d6136775a56a9a0127f682470bef831fbec4bcd7b<br>5095a7823fd70745d37d1bf72b63c4b1b4a3d0581e74bf9ade93cc4614861755<br>3931a79d92e9e488ef47223ee6f6c061884b13c9065b591139de13c1ea292749<br>1ed00fb793cd68f463f5f64baa53916b46c818ab99706557a1c2d50d232577d1<br>40d60f24b61d76783d3bb1dc00b55f96a2a686f59b3750fdb15c40251c370c65<br>cada222673811bc6b305ed7c90ffc3abdddc8336612ff13b42a75cb7c88fb93<br>6291b523d80acce5a0842c724ed85a1393faf3d470bda8083fa84dc5f3149984<br>4f0c7c1e93fb1f734a5a29fb31a35c8a0822455f1c850a49e8629714ec6a2657<br>efe75ec1ca6e62f9a3756c9b20b4855bdc9a3ab58c43d8af85b837a7fd15aa11<br>49c119cfe960c05a9d4cea69c9fb6a897145674882bf57241d77c054dc4c94e8<br>349d376296137eb421686159cb878d15d171eda8692834afc871988f203fc822<br>c5dcee7f6c48df663ea3dc755e7dc06aebd41d05f1ca2891e2679783244d068f |
| ECC 192-bit public*2<br>(top: Qx, bottom: Qy) | fba2aac647884b504eb8cd5a0a1287babcc62163f606a9a2<br>dae6d4cc05ef4f27d79ee38b71c9c8ef4865d98850d84aa5                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| ECC 192-bit private*2                         | 7891686032fd8057f636b44b1f47cce564d2509923a7465b                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| ECC 224-bit public*2<br>(top: Qx, bottom: Qy) | 4c741e4d20103670b7161ae72271082155838418084335338ac38fa4<br>db7919151ac28587b72bad7ab180ec8e95ab9e2c8d81d9b9d7e2e383                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| ECC 224-bit private*2                         | 888fc992893bdd8aa02c80768832605d020b81ae0b25474154ec89aa                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| ECC 256-bit public*2<br>(top: Qx, bottom: Qy) | 1ccbe91c075fc7f4f033bfa248db8fccd3565de94bbfb12f3c59ff46c271bf83<br>ce4014c68811f9a21a1fdb2c0e6113e06db7ca93b7404e78dc7ccd5ca89a4ca9                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| ECC 256-bit private*2                         | 519b423d715f8b581f4fa8ee59f4771a5b44c8130b4e3eacca54a56dda72b464                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |

Notes: 1. The RSA 1024-bit key pair has the following values:

Modulus n:

ccd6cb86f59f9ea97c278b7cf395fa56f3709a958bce1e6e3ae196b471f4517f  
64d32d81d10bcea5b55b9ea659c3b9db1854a696b801a8e72439265e85bc6138  
cf874f45fe22a2477fe8337671357db67180b1ac9a0e84c098dbcdca8a3e6a72  
dfcee7e6907ede2d53aa726e8e97ea26fcb3bbfa7b7fbfbb70f1ec1d153125

Exponent e: 00010001

Decryption Exponent d:

096ed2cc923f1df11c208e11e0fdc51b7ff66d87f97a32788d099a7110d65972  
6e68332e493c2bf6019608864c97f0d52017b5dc36f90c982858e16574ef29e2  
e8b6f4386bddd1beae60390aaac0160ce787f98ba34c83d6612badbc99d17666  
04e72b394d1991fc4ded87e11cf165caae0cc652771b8395ed5b9a99a9fa5781

The RSA 2048-bit key pair has the following values:

Modulus n:

bad47a84c1782e4dbdd913f2a261fc8b65838412c6e45a2068ed6d7f16e9cdf4  
462b39119563cafb74b9cbf25cfd544bdae23bff0ebe7f6441042b7e109b9a8a  
faa056821ef8efaab219d21d6763484785622d918d395a2a31f2ece8385a8131  
e5ff143314a82e21afd713bae817cc0ee3514d4839007ccb55d68409c97a18ab



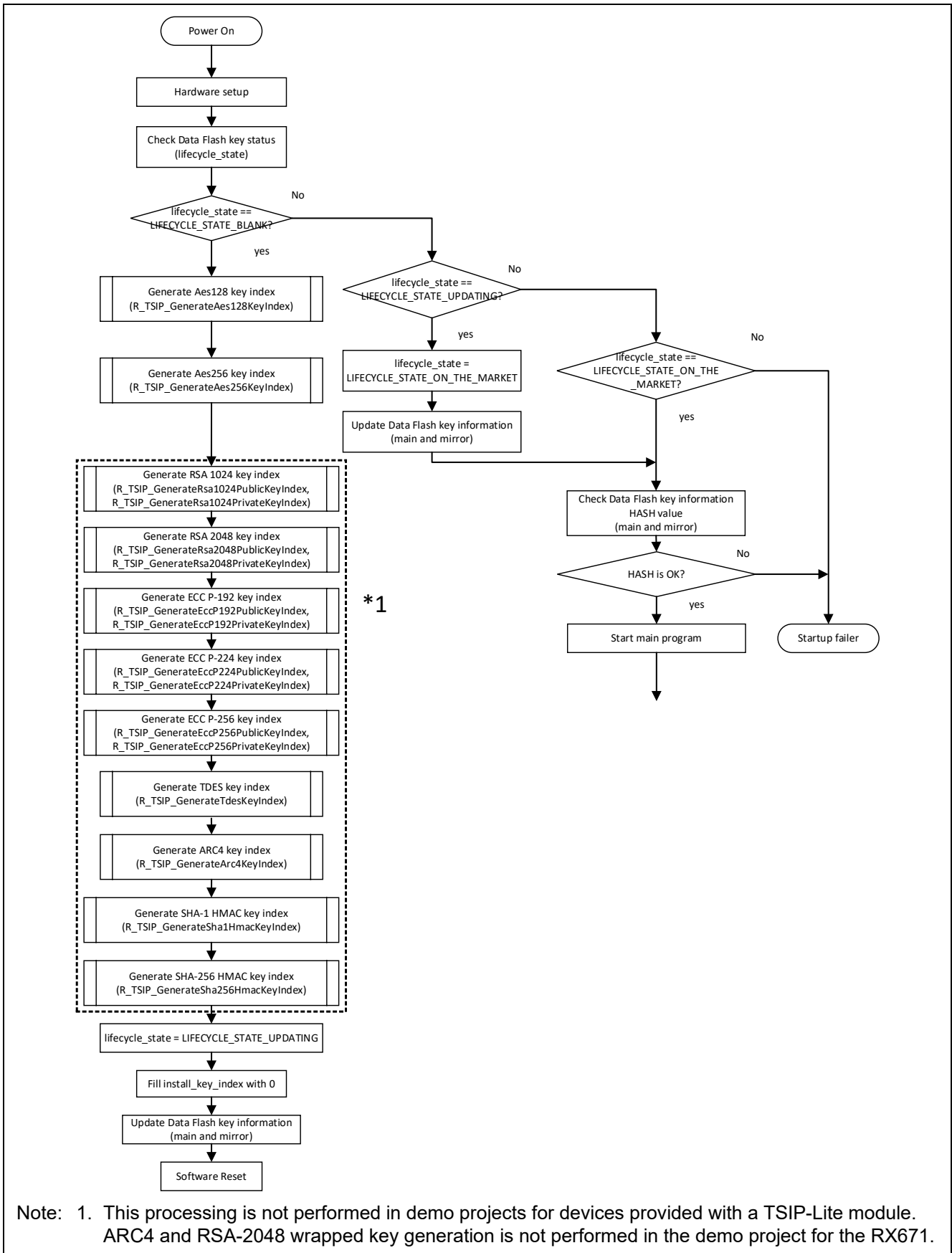
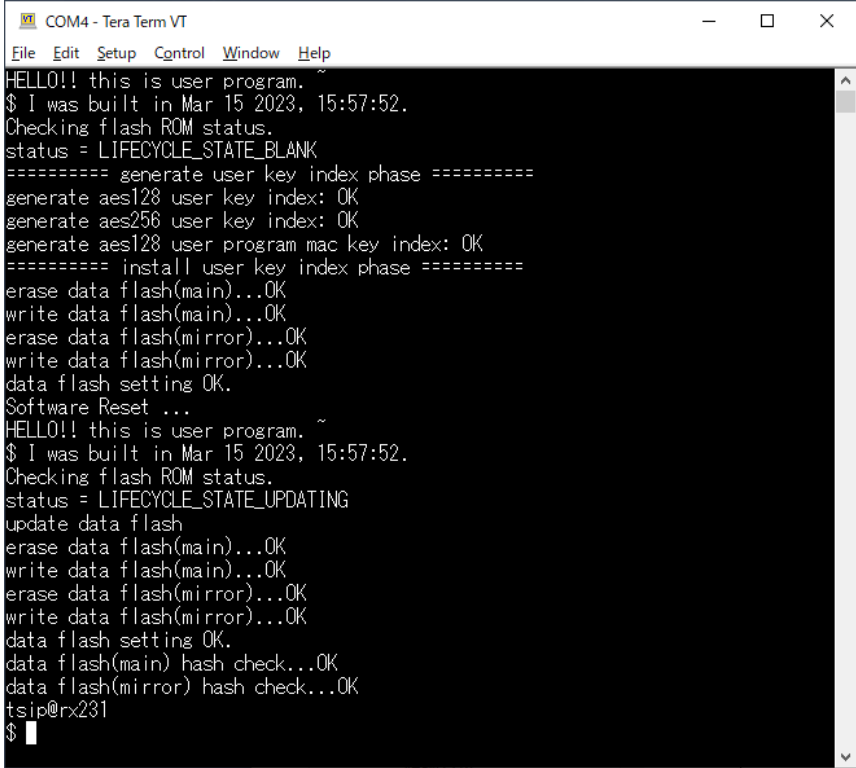


Figure 6.10 Flowchart of Key Injection at Program Startup

### 6.1.3 Demo Project Execution Example

Figure 6.10 shows the terminal output when key installation completes successfully after the program is downloaded to the MCU and run for the first time, and Figure 6.11 shows the terminal output when the program is run from a state other than when it has been newly downloaded to the MCU. After the above output is displayed, the program is ready to accept the commands listed in Table 6.3.



```
COM4 - Tera Term VT
File Edit Setup Control Window Help
HELLO!! this is user program.
$ I was built in Mar 15 2023, 15:57:52.
Checking flash ROM status.
status = LIFECYCLE_STATE_BLANK
===== generate user key index phase =====
generate aes128 user key index: OK
generate aes256 user key index: OK
generate aes128 user program mac key index: OK
===== install user key index phase =====
erase data flash(main)...OK
write data flash(main)...OK
erase data flash(mirror)...OK
write data flash(mirror)...OK
data flash setting OK.
Software Reset ...
HELLO!! this is user program. ~
$ I was built in Mar 15 2023, 15:57:52.
Checking flash ROM status.
status = LIFECYCLE_STATE_UPDATING
update data flash
erase data flash(main)...OK
write data flash(main)...OK
erase data flash(mirror)...OK
write data flash(mirror)...OK
data flash setting OK.
data flash(main) hash check...OK
data flash(mirror) hash check...OK
tsip@rx231
$ █
```

Figure 6.11 Tera Term Display (Initial Execution on RSK RX231 Board)

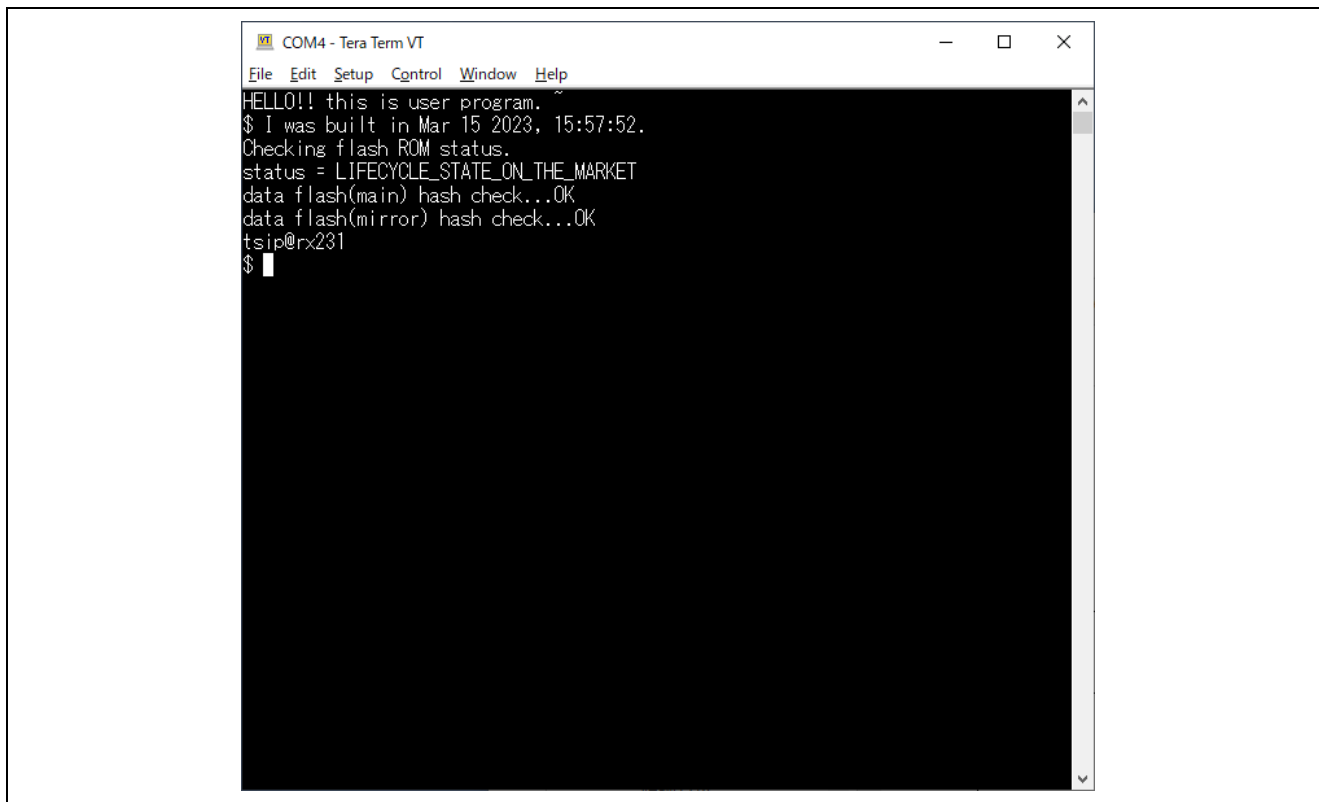


Figure 6.12 Tera Term Display (2nd and Subsequent Executions on RSK RX231 Board)

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

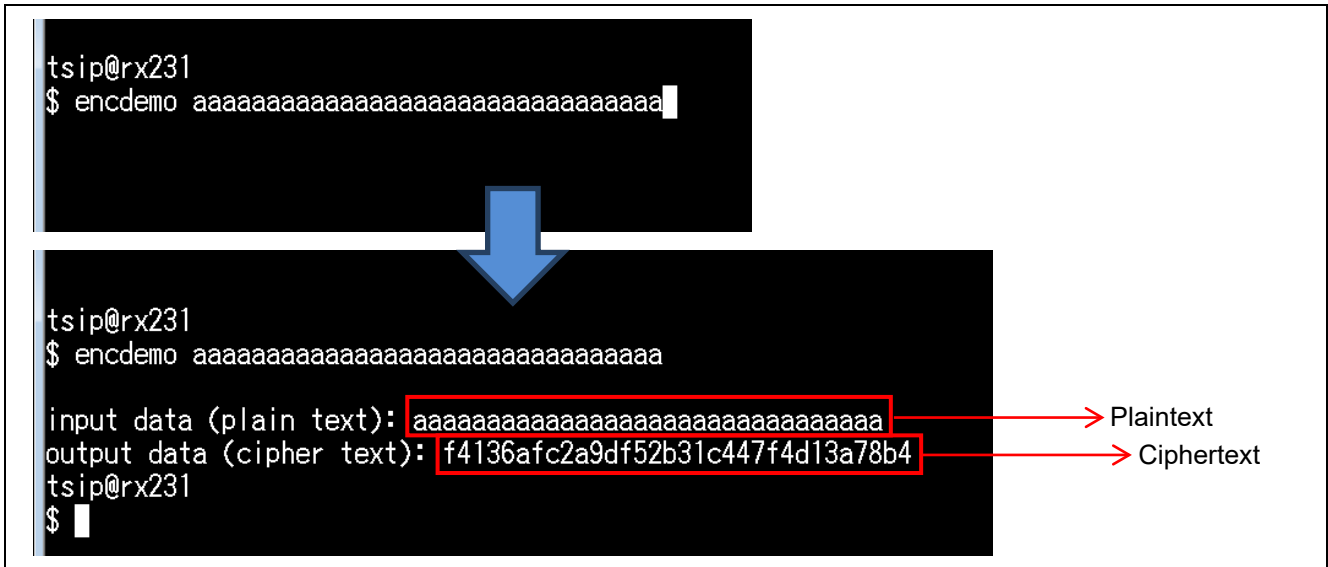


Figure 6.13 Tera Term Display (encdemo Command)

**[Key Point 1]**

The user key is stored in the data flash area (0x00100000), but make sure the user key specified in Security Key Management Tool (default value: 0x11, 0x11, 0x11, 0x11, 0x22, 0x22, 0x22, 0x22, 0x33, 0x33, 0x33, 0x33, 0x44, 0x44, 0x44, 0x44) is not exposed.

This is important to ensure that the user key cannot be discovered by reverse engineering the software.

**[Key Point 2]**

The wrapped key is tied to the HUK. It is not possible to use a wrapped key generated on one chip on another chip by copying it over. This is because the TSIP driver verifies the wrapped key by checking it against the HUK.

This is important to prevent dead copying of software.

**[Key Point 3]**

The wrapped key is always output as a different value even when the same user key is input. This is because a random number is used when generating the wrapped key.

This is important to reduce the risk of the user key being guessed based on the wrapped key.

## 6.2 Confirming Operation of the Secure Bootloader and Firmware Update Demo Projects

The secure bootloader and firmware update demo projects use the API functions listed in 4.2.16, Firmware Update, to decrypt on the device a user program encrypted as described in 3.14.3, Encrypting the User Program. The term firmware update refers to the process of writing the encrypted user program to the on-chip flash ROM following verification. Secure boot refers to the process of verifying the user program before execution. The secure bootloader and firmware update demo projects are sample programs that implement firmware update and secure boot operations.

Two demo projects are provided, a secure bootloader project and a firmware update project. The secure bootloader project includes firmware update functionality to write the initial firmware to the device.

Secure bootloader project: rxXXX\_bbb\_tsip\_secure\_boot\*<sup>1</sup>

Firmware update project: rxXXX\_bbb\_tsip\_user\_program\*<sup>1</sup>

The firmware update functionality of the demo projects supports use of a UART\*<sup>2</sup> or USB\*<sup>3</sup> interface to send the initial program or the updated program to the device. In addition, Security Key Management Tool can be used to generate a file in Motorola S format\*<sup>4</sup> containing both the secure bootloader and the encrypted user program for use when performing programming at the factory (factory programming function).

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

- Notes:
1. The versions of projects for devices without USB support only provide support for UART. Project versions that do not support USB have `_sci` in the project name. Also, the letters `bbb` are replaced by the board name. RSK is indicated by `rsk`, MCB by `mcb`, and Envision Kit by `ek`.
  2. For devices that are incompatible with dual bank mode (the RX231, RX66T, and RX72T), use the PMOD USB-UART converter board for UART flow control. It is also necessary to use the PMOD USB-UART converter board with boards lacking USB-serial conversion capability (the MCB RX26T).
  3. The RX66T and RX26T do not have USB interface support, so there is no demo project that uses USB provided for these devices.
  4. Motorola S format files containing a secure bootloader and an encrypted user program can be decrypted by a corresponding RX TSIP-equipped device if the Motorola S format file itself is stolen. It is therefore necessary to ensure that programming to devices of encrypted user programs with secure bootloaders takes place in a secure facility.

### 6.2.1 Demo Project Setup

The demo projects will run on the boards listed in Table 6.5, Boards Used in Demo Projects and Their Settings. To use the USB interface, set the jumper on the RSK board to the USB-HOST side to enable USB Host mode on the board. The factory default setting is the USB-FUNCTION side. For devices that do not support dual bank mode, use the PMOD USB-UART converter board for status indications and transfer of the update program.

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

| Boards                | Jumper setting                    | Flash Mode     | Note                                                                                                                                |
|-----------------------|-----------------------------------|----------------|-------------------------------------------------------------------------------------------------------------------------------------|
| RSK RX231             | J15 : 1-2 short                   | -              | When using UART as the interface, the 0 Ω resistors on the RSK board must be replaced. A PMOD USB-UART converter board is required. |
| MCB RX26T             | -                                 | Dual Bank Mode | A PMOD USB-UART converter board is required. Supports UART only.                                                                    |
| RSK RX65N             | J7 : 1-2 short<br>J16 : 2-3 short | Dual Bank Mode | —                                                                                                                                   |
| RSK RX66T             | -                                 | -              | A PMOD USB-UART converter board is required. Supports UART only.                                                                    |
| RSK RX671             | J8 : 2-3 short<br>J13 : 2-3 short | Dual Bank Mode | —                                                                                                                                   |
| RSK RX72M             | J8 : 2-3 short<br>J10 : 2-3 short | Dual Bank Mode | —                                                                                                                                   |
| RSK RX72N             | J7 : 2-3 short<br>J8 : 2-3 short  | Dual Bank Mode | —                                                                                                                                   |
| RX72N<br>Envision Kit | -                                 | Dual Bank Mode | —                                                                                                                                   |
| RSK RX72T             | J13 : 1-2 short                   | -              | A PMOD USB-UART converter board is required.                                                                                        |

The figures showing execution results show results for the RX65N. The terminal emulator used is Tera Term.

The operation of the demo projects has been confirmed using little-endian byte ordering. RX Firmware Update FIT V.1.05 is used with some modifications. Changes are indicated by the comment `/* modified from original fwup code */`.

The link size of the demo projects for TSIP-equipped devices is larger than 128 KB. You will need to purchase the commercial version of the CC-RX compiler in order to use them.



Connections between the PC and board are shown below.

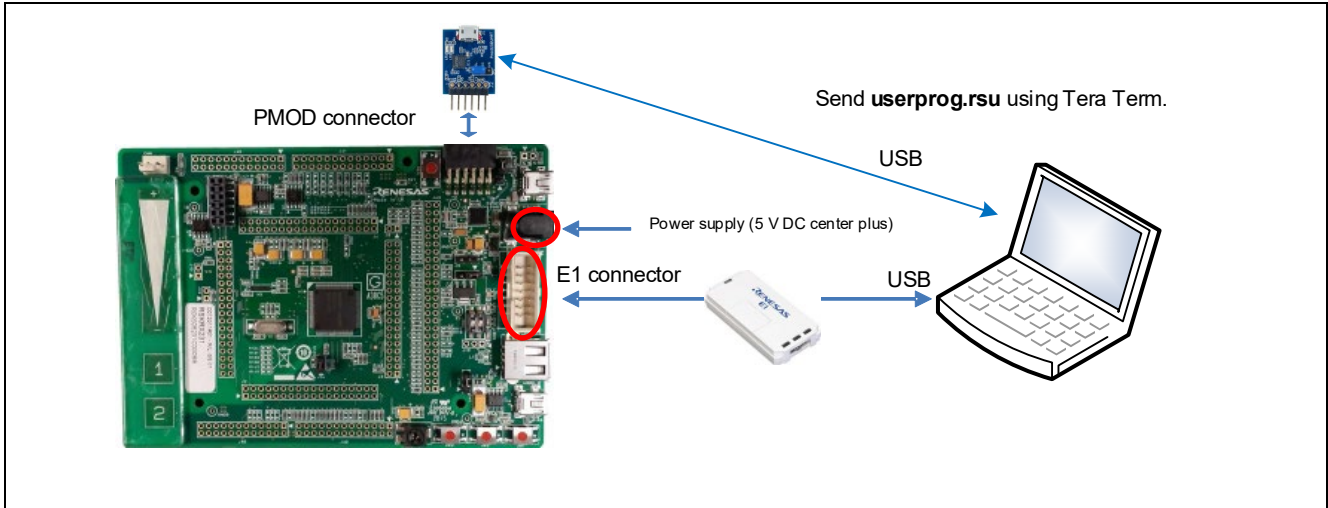


Figure 6.14 Connections between RSK RX231\*2 and PC for Firmware Updates Using UART\*1

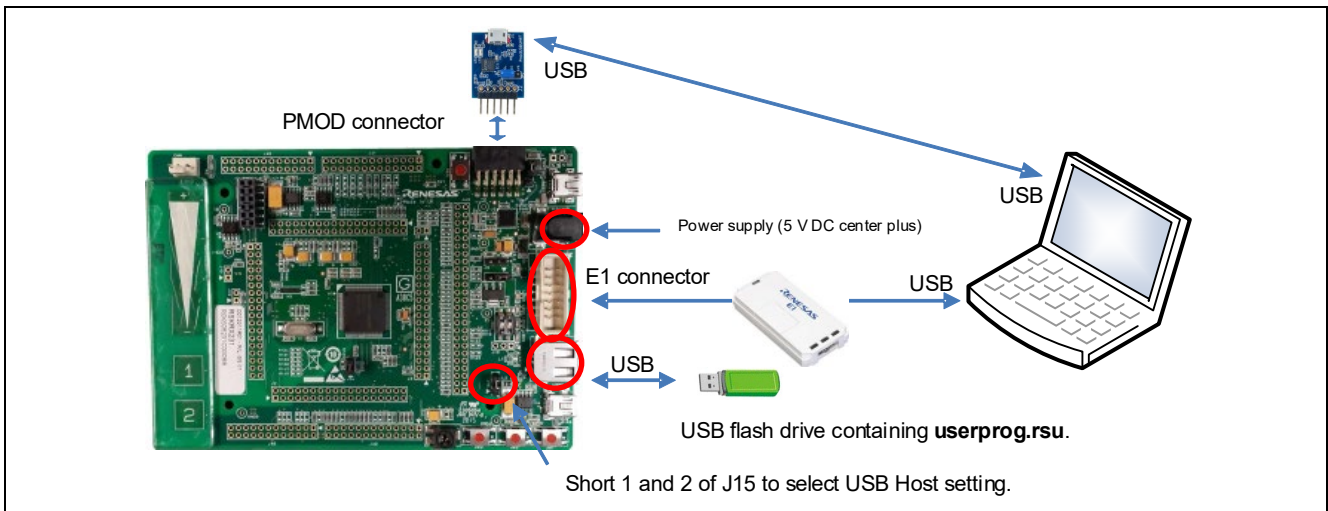


Figure 6.15 Connections between RSK RX231 and PC for Firmware Updates Using USB

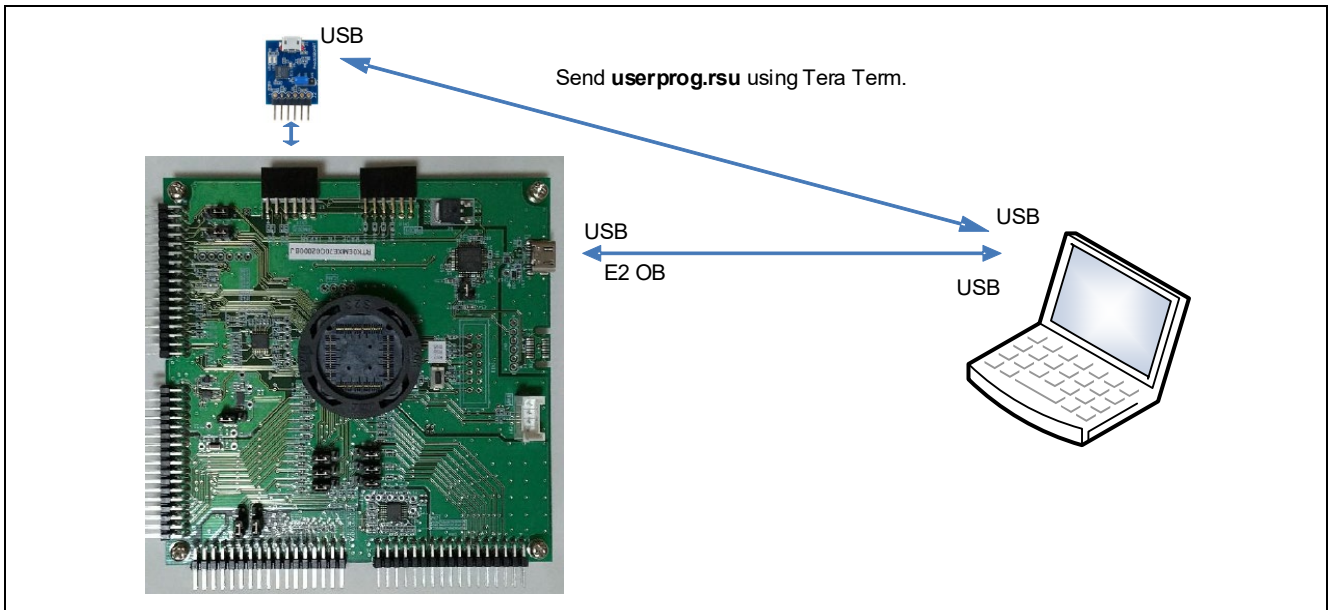


Figure 6.16 Connections between MCB RX26T and PC for Firmware Updates Using UART

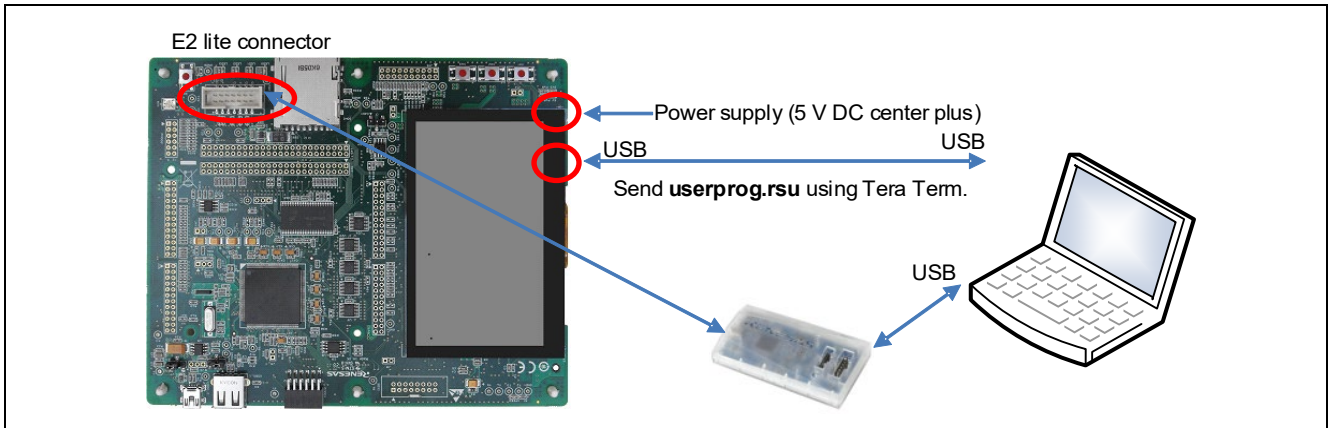


Figure 6.17 Connections between RSK RX65N-2MB and PC for Firmware Updates Using UART

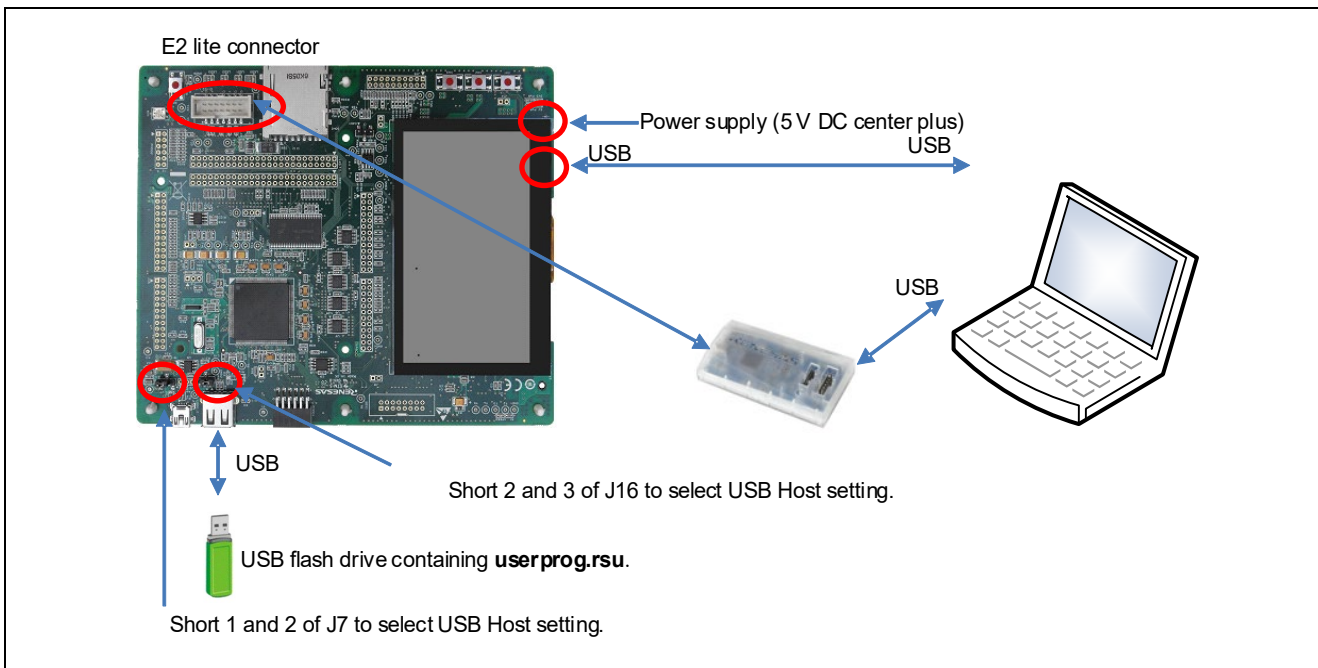


Figure 6.18 Connections between RSK RX65N-2MB and PC for Firmware Updates Using USB

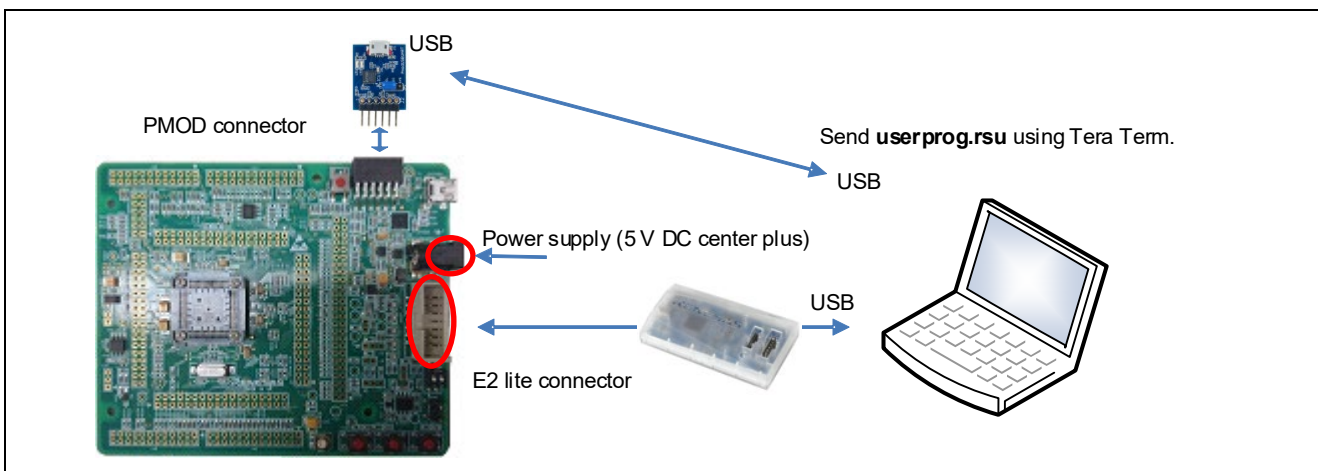


Figure 6.19 Connections between RSK RX66T and PC for Firmware Updates Using UART\*1

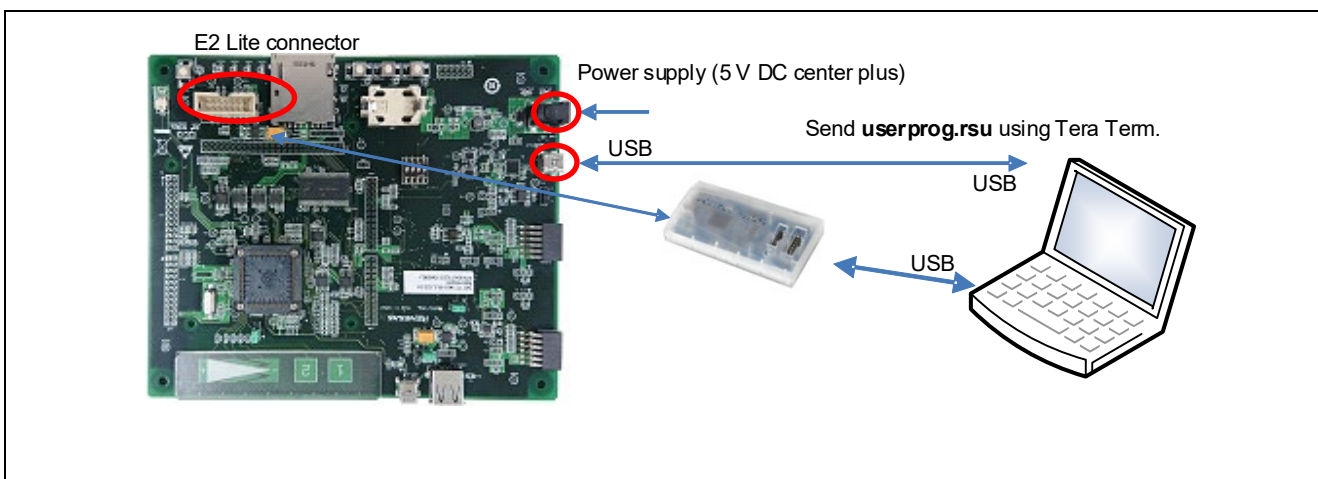


Figure 6.20 Connections between RSK RX671 and PC for Firmware Updates Using UART

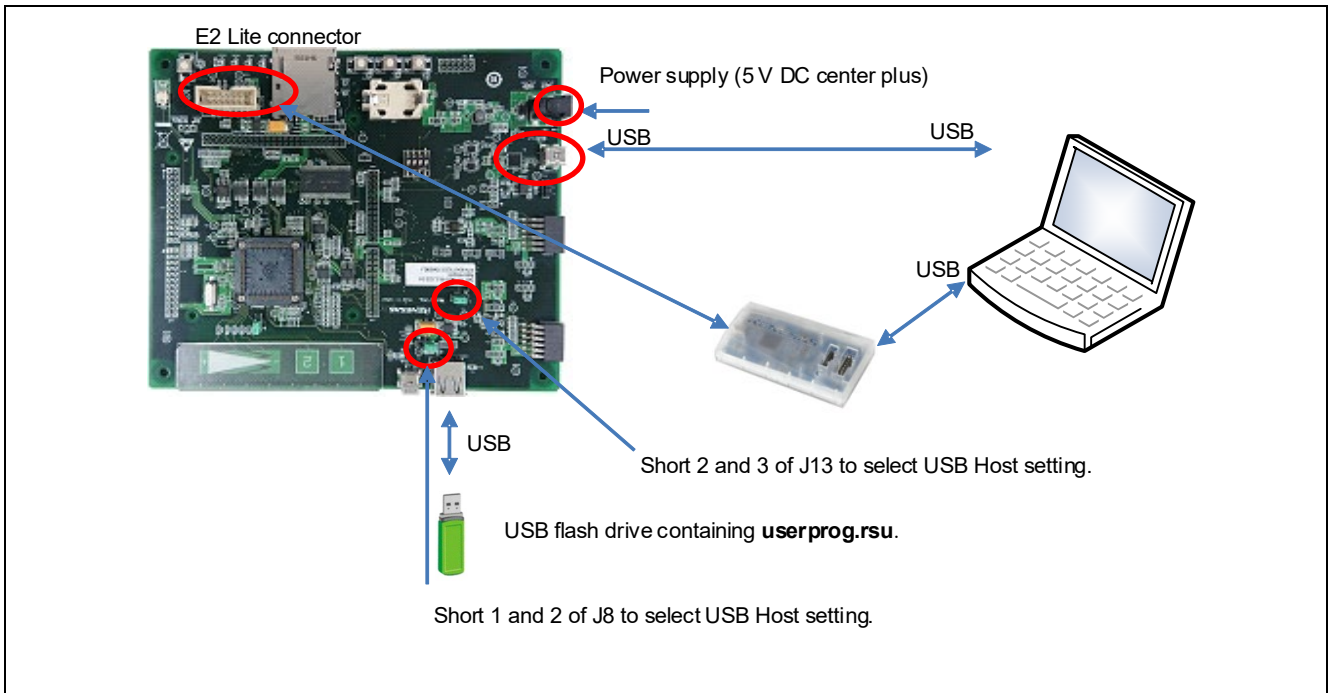


Figure 6.21 Connections between RSK RX671 and PC for Firmware Updates Using USB

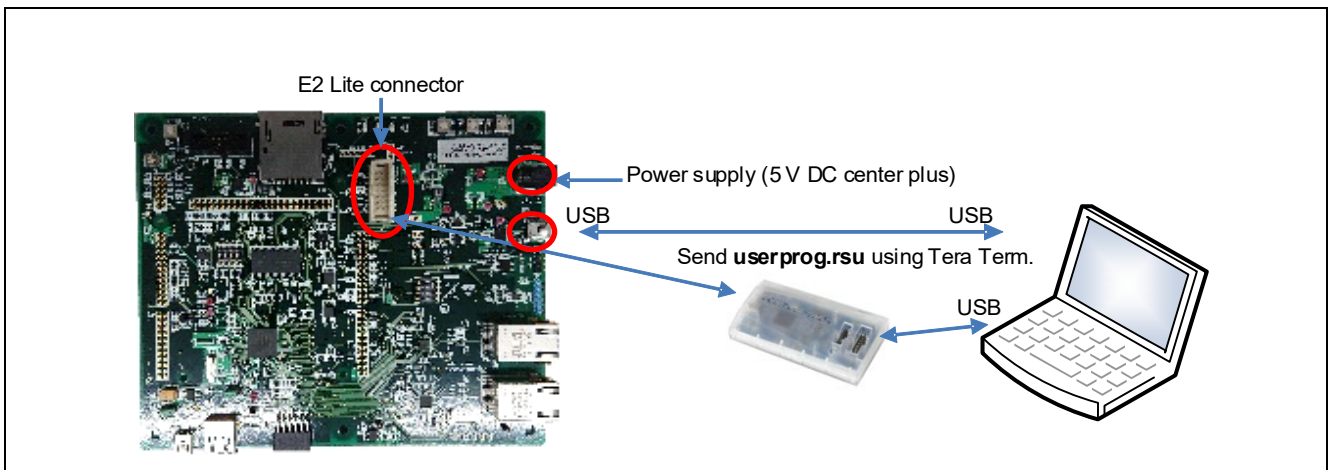


Figure 6.22 Connections between RSK RX72M and PC for Firmware Updates Using UART



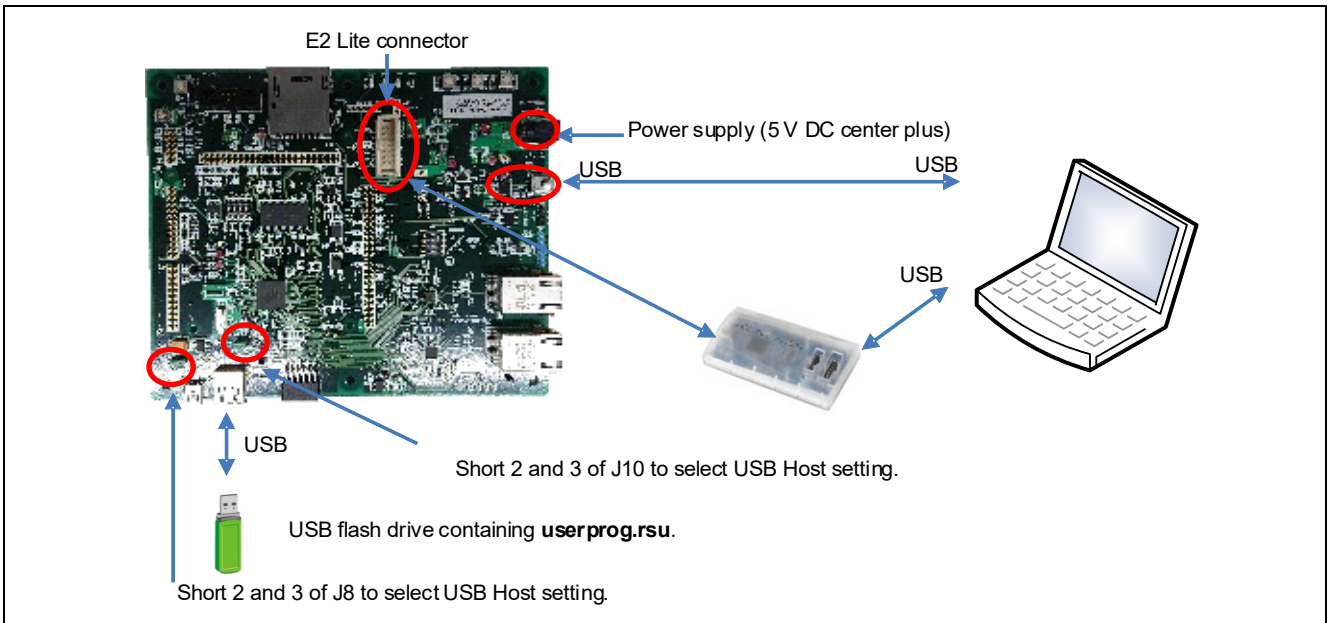


Figure 6.23 Connections between RSK RX72M and PC for Firmware Updates Using USB

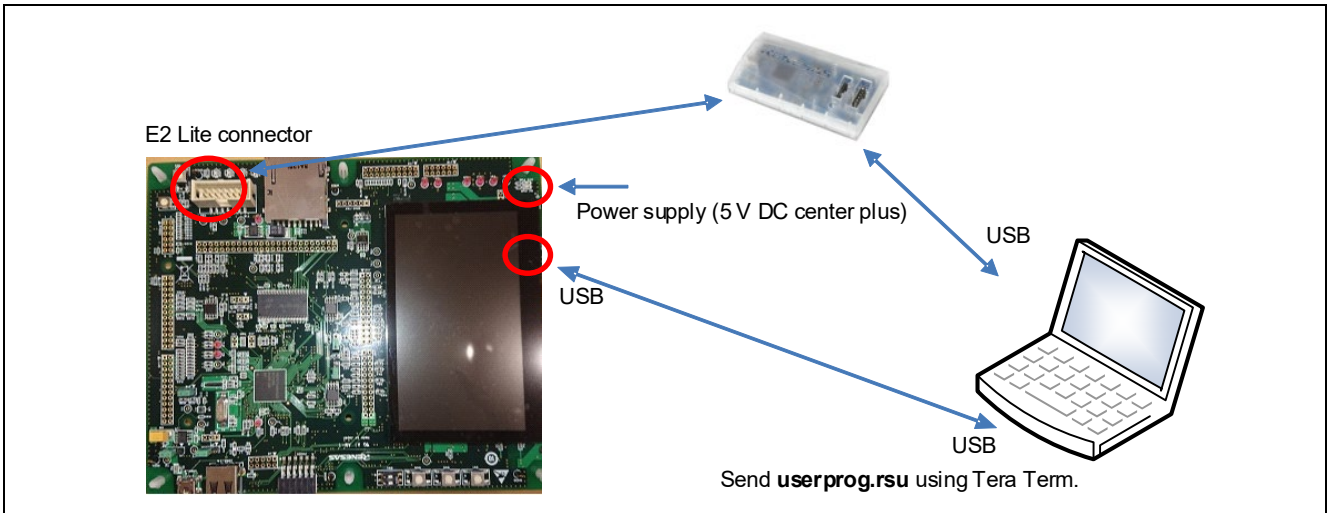


Figure 6.24 Connections between RSK RX72N and PC for Firmware Updates Using UART

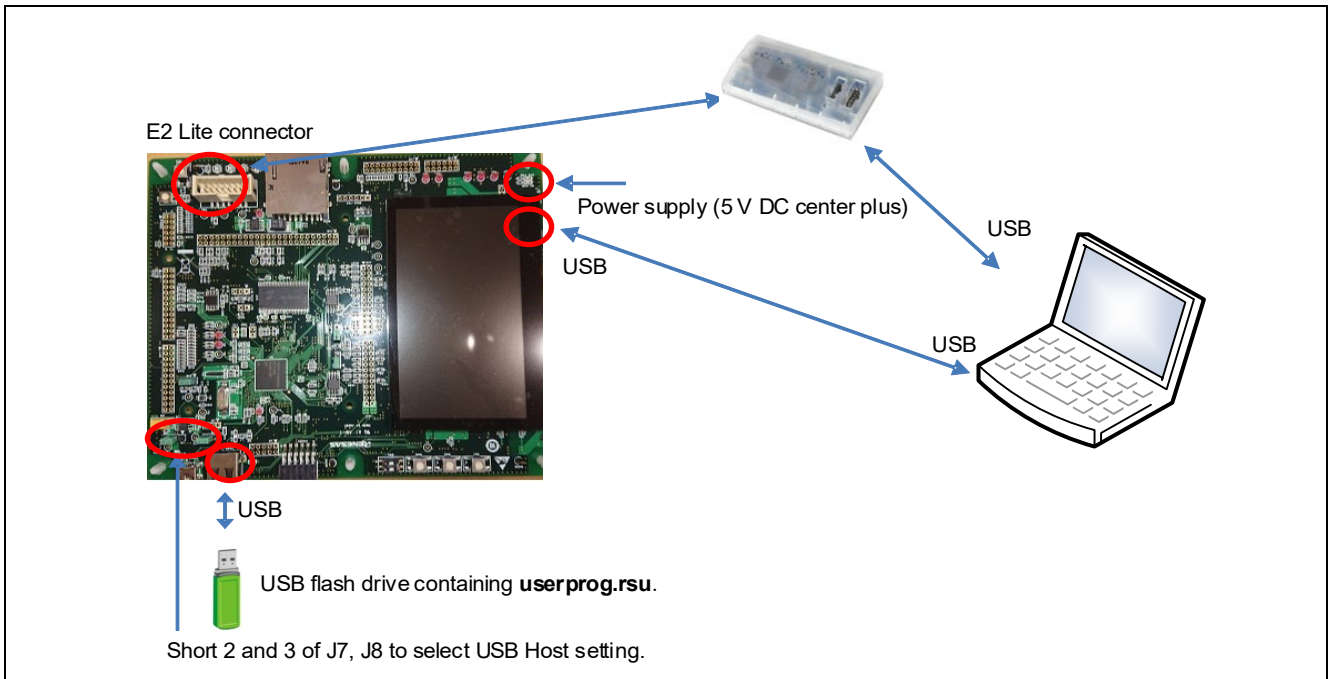


Figure 6.25 Connections between RSK RX72N and PC for Firmware Updates Using USB

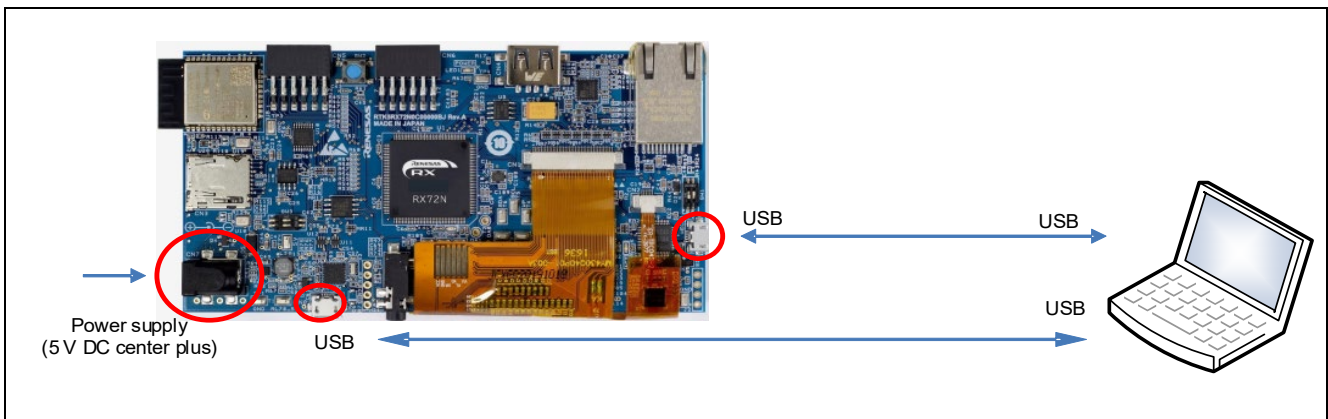


Figure 6.26 Connections between RX72N Envision Kit and PC for Firmware Updates Using UART

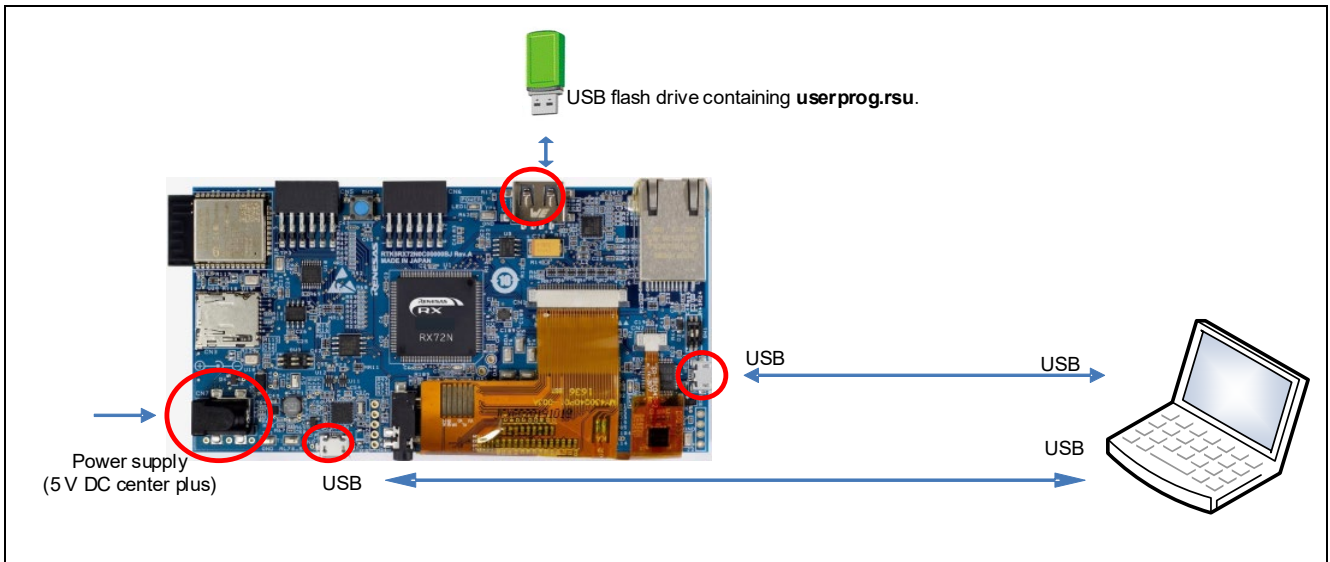


Figure 6.27 Connections between RX72N Envision Kit and PC for Firmware Updates Using USB

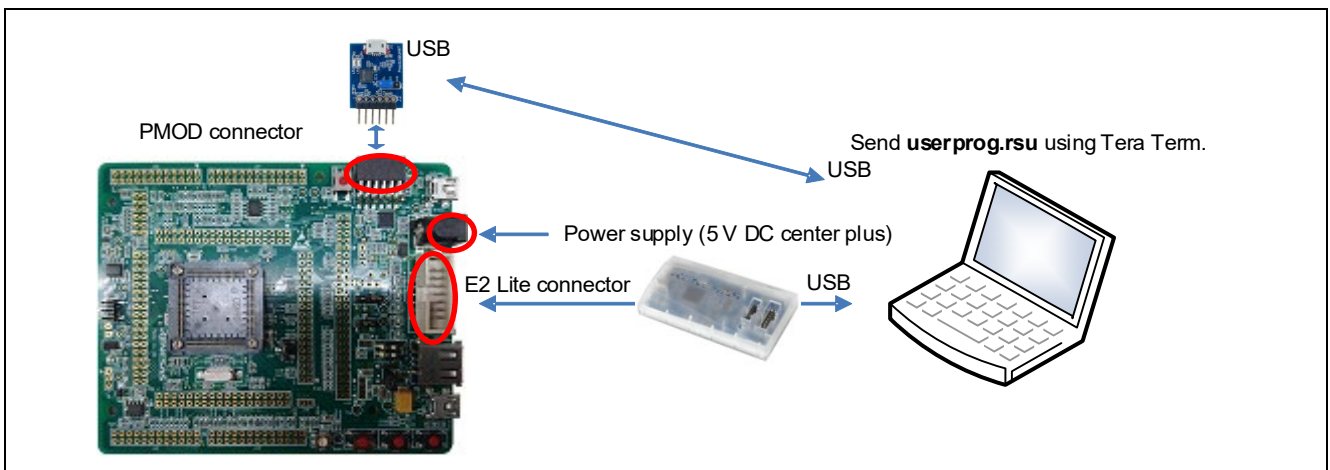


Figure 6.28 Connections between RSK RX72T and PC for Firmware Updates Using UART\*1

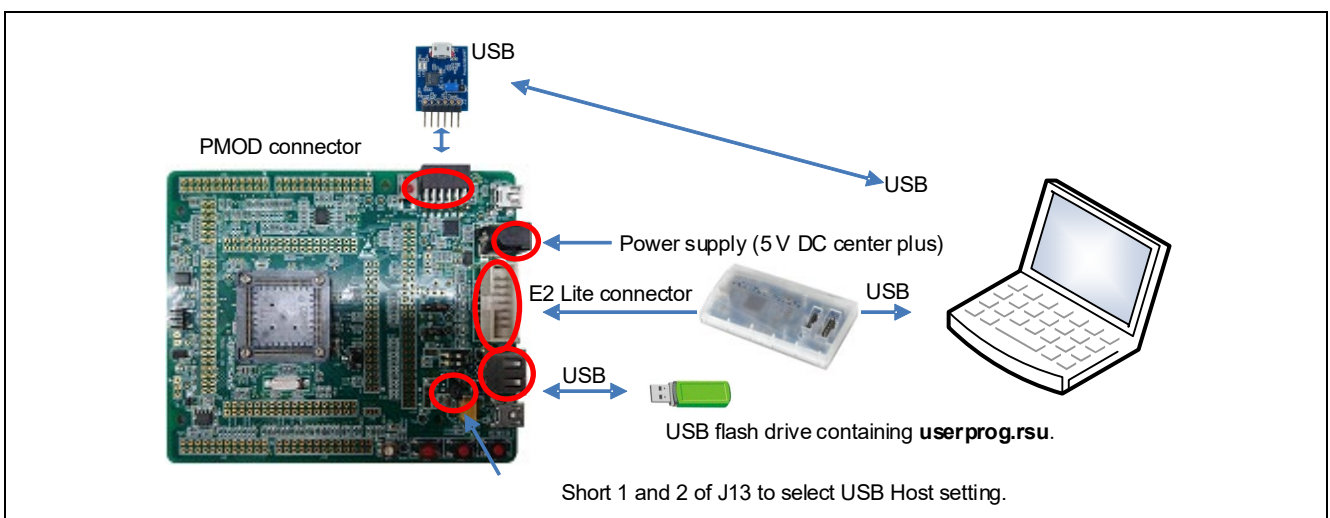


Figure 6.29 Connections between RSK RX72T and PC for Firmware Updates Using USB

Notes: 1. The operation of the demo projects has been confirmed using a Pmod USBUART from Digilent. The connection to the PMOD connector on the RX RSK board must be made as shown below.

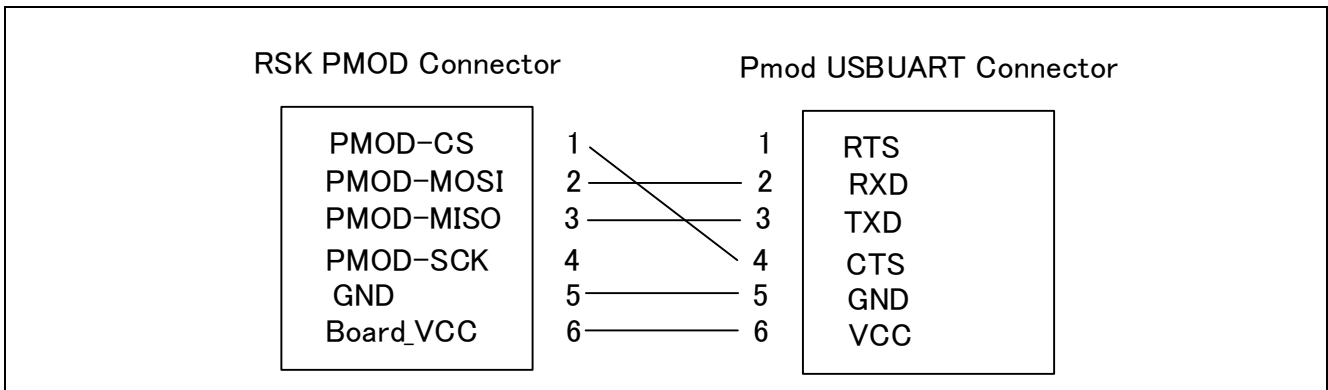


Figure 6.30 Connection diagram of RX RSK and Pmod USBUART

2. To perform flow control of the Pmod USBUART board connected to the PMOD1 connector on the RSK RX231 board, it is necessary to replace the previously mounted 0 Ω resistors with a pattern with no devices mounted.

| Mounted |   | Not Mounted |
|---------|---|-------------|
| R125    | → | R126        |
| R128    | → | R127        |
| R135    | → | R134        |

Tera Term serial settings are as follows.

Table 6-6 Using RSK RX231, RSK RX66T or RSK RX72T

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

Table 6-7 Using RSK RX65N-2MB, RSK RX671, RSK RX72M, RSK RX72N, RX72N Envision Kit or MCB RX26T

| Item         | Value    |
|--------------|----------|
| Speed        | 57600bps |
| Data         | 8bit     |
| Parity       | none     |
| Stop bit     | 1bit     |
| New-line     | CR       |
| Local echo   | OFF      |
| Flow control | none     |



## 6.2.2 Overview of Secure Bootloader and Firmware Update Projects

This section provides an overview of the secure bootloader project and firmware update project. The directory structure is described in Table 6-8.

**Table 6-8 Directory structure of secure bootloader project and firmware update project**

| Directory Name                     | Description                                               |
|------------------------------------|-----------------------------------------------------------|
| FITDemos                           | Demo project folder                                       |
| rxXXX_bbb_tsip_secure_update *1 *2 | Secure boot/secure update implementation example          |
| rxXXX_bbb_tsip_secure_boot *1 *2   | Secure boot firmware                                      |
| key                                | UFPK and W-UFPK file which can be used to execute example |
| skmt                               | Settign file of Security Key Management Tool              |
| src                                | Source code of the secure boot firmware                   |
| rxXXX_bbb_tsip_user_program *1 *2  | User program following secure update                      |
| src                                | Source code of the user program                           |

Note: 1. "rxXXX" represents the RX group names.

2. "bbb" is the name of the supported board. Usually, USB and UART are available as IF for Update, but if the MCU does not support USB, only UART is supported. The director name of project which support only UART includes \_sci before \_secure\_boot and \_user\_program.

RSK : rsk (rsk(\_tsip)\_sci : only support UART)

MCB : mcb (mcb(\_tsip)\_sci : only support UART)

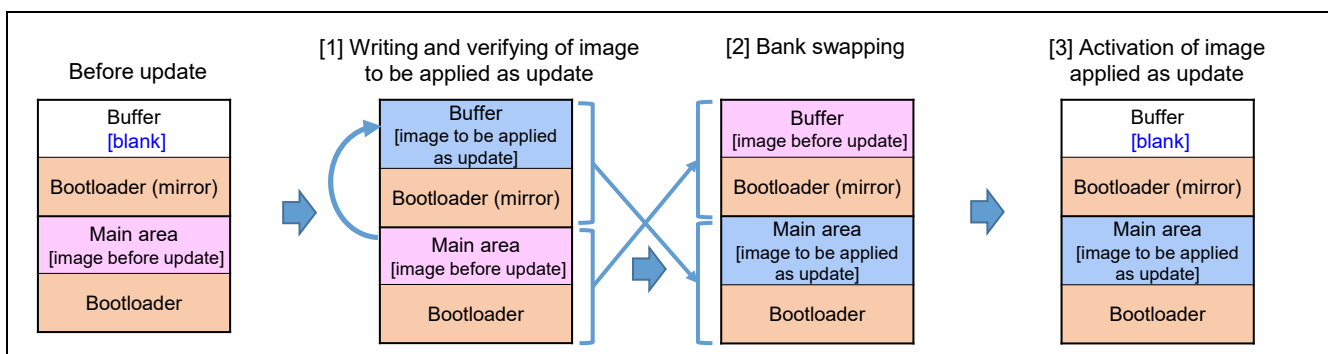
Envision Kit : ek

The RX TSIP FIT secure bootloader and firmware update projects are sample programs that divide the flash ROM area into two sections, which are used as the main area and buffer area.

Main area: Storage area for user program to be executed

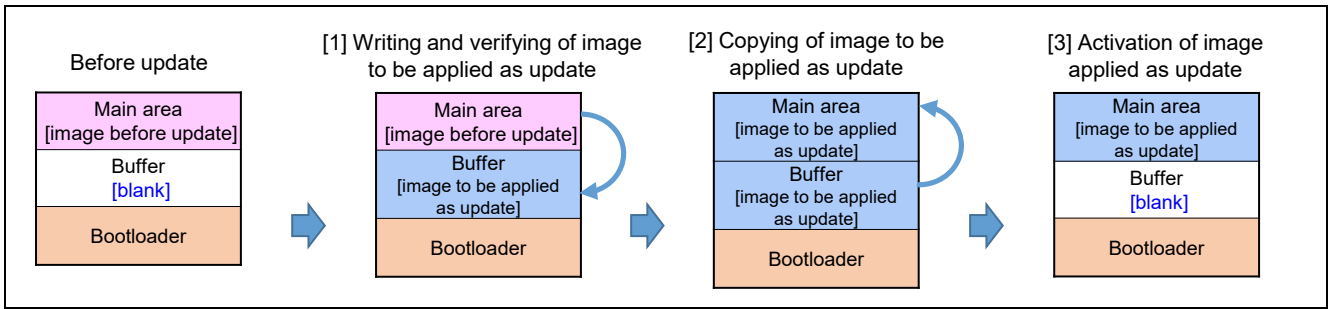
Buffer area: Storage area for image to be applied as an update

On devices with flash ROM that supports the dual bank function, this functionality is utilized in the firmware update operation.



**Figure 6.31 Firmware Update Operation Utilizing Dual Bank Function**

On devices with flash ROM that does not support the dual bank function, the flash ROM area is divided into two sections, a main area and buffer area, and a partial updating method is used to apply the firmware update.



**Figure 6.32 Firmware Update Operation Utilizing Partial Updating Method**

The key\_block\_data located in the data flash is used to manage the MAC values used for status management and validity verification as well as the keys used for decryption. As data management for key injection or countermeasures in case of power interruption while an update is in progress, two copies of the same data are maintained in a main area and mirror area in the data flash.

Table 6-9 key\_block\_data Structure

| Name                         | Type                 | Description                                                                                              |
|------------------------------|----------------------|----------------------------------------------------------------------------------------------------------|
| st_key_block_data_t          | —                    | Key data structure located in C_FIRMWARE_UPDATE_CONTROL_BLOCK and C_FIRMWARE_UPDATE_CONTROL_BLOCK_MIRROR |
| firmware_update_control_data | —                    | Structure for storing MAC information during firmware update                                             |
| user_program_max_cnt         | uint32_t             | Byte size of firmware update                                                                             |
| lifecycle_state              | uint32_t             | See Table 6.9, Description of lifecycle_state                                                            |
| program_mac0[]               | uint32_t             | MAC value of buffer area                                                                                 |
| program_mac1[]               | uint32_t             | MAC value of main area                                                                                   |
| key_data                     | —                    | Key data storage structure                                                                               |
| user_aes128_key_index        | tsip_aes_key_index_t | Key index used to decrypt encrypted user program                                                         |
| hash_sha1[]                  | uint8_t              | SHA1 hash value of firmware_update_control_data and key_data                                             |

Table 6-10 Description of lifecycle\_state

| Status        | Description |                                                                                                                |
|---------------|-------------|----------------------------------------------------------------------------------------------------------------|
| BLANK         | Key         | Injection incomplete                                                                                           |
|               | Program     | Buffer area: Injection incomplete.<br>Main area: Injection incomplete.                                         |
| UPDATING      | Key         | Injection complete, key data hash not set.                                                                     |
|               | Program     | Buffer area: Writing of program update complete.<br>Main area: Program has not been updated.                   |
| ON_THE_MARKET | Key         | Injection complete, key data hash setting complete.                                                            |
|               | Program     | Buffer area: Program has not been updated.<br>Main area: Program has been updated, program execution possible. |

6.2.2.1 Secure Bootloader project

The secure bootloader project performs key injection used to decrypt the user program and verifies the user program. It also contains program code to install the initial user program (the firmware update program in the case of this sample project).

A flowchart of the secure bootloader project is shown below.

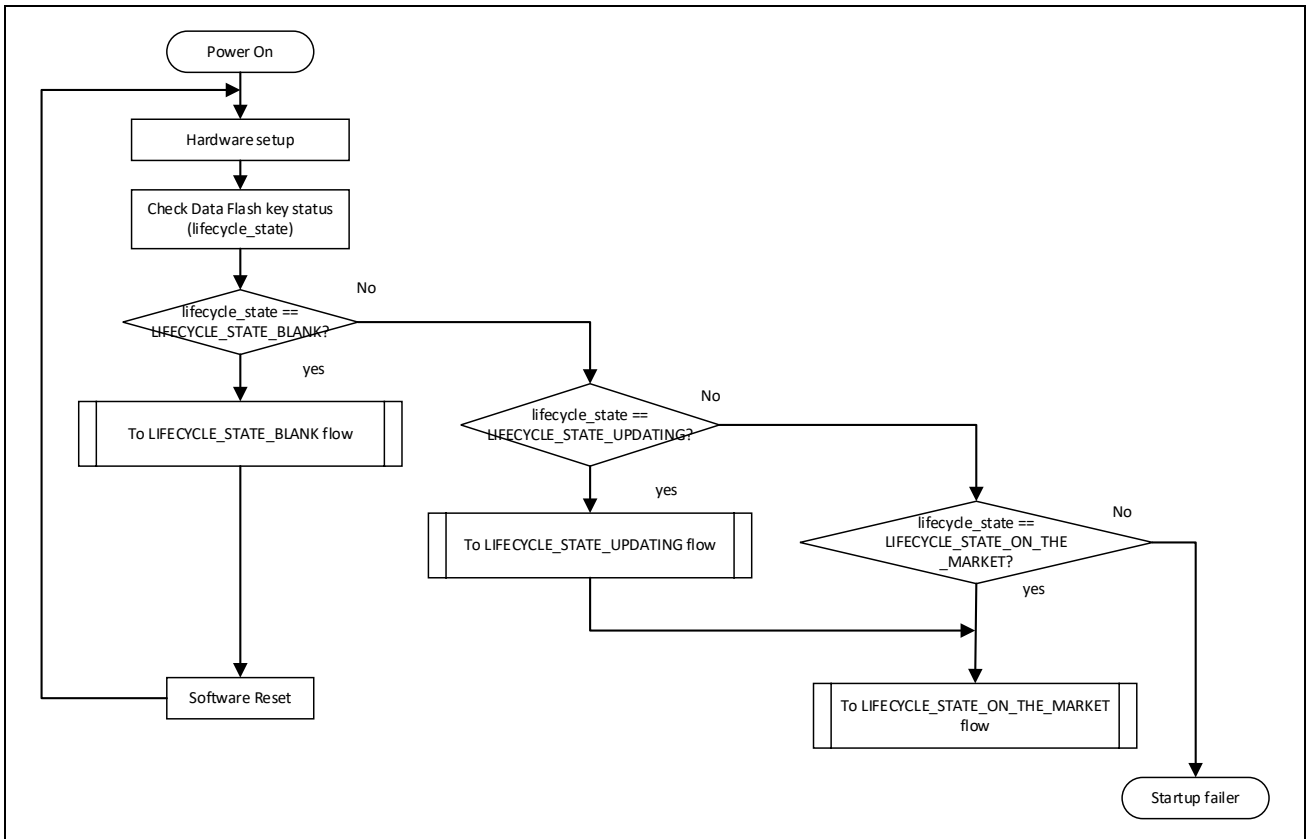


Figure 6.33 The MAIN flow of Secure Bootloader

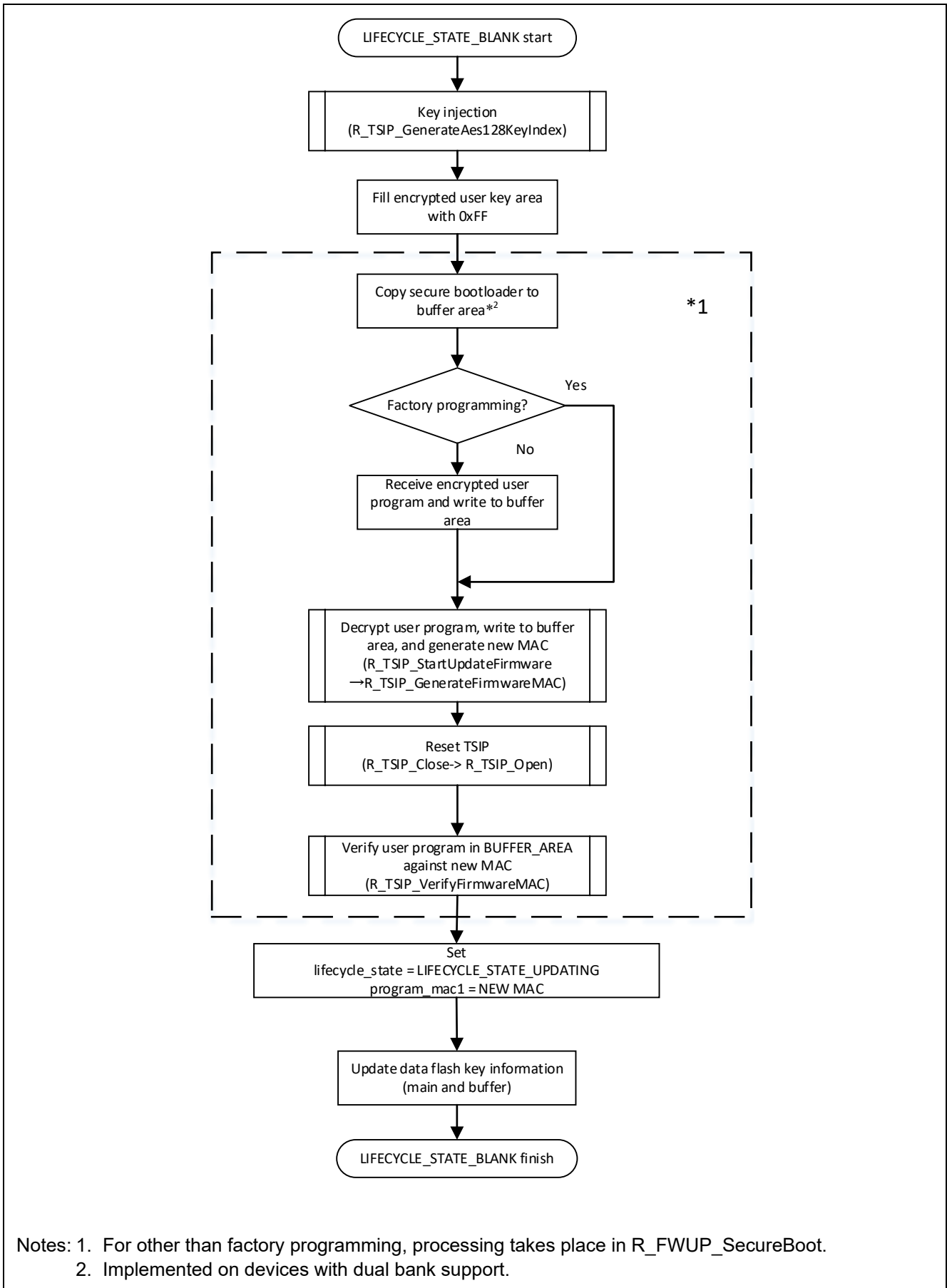
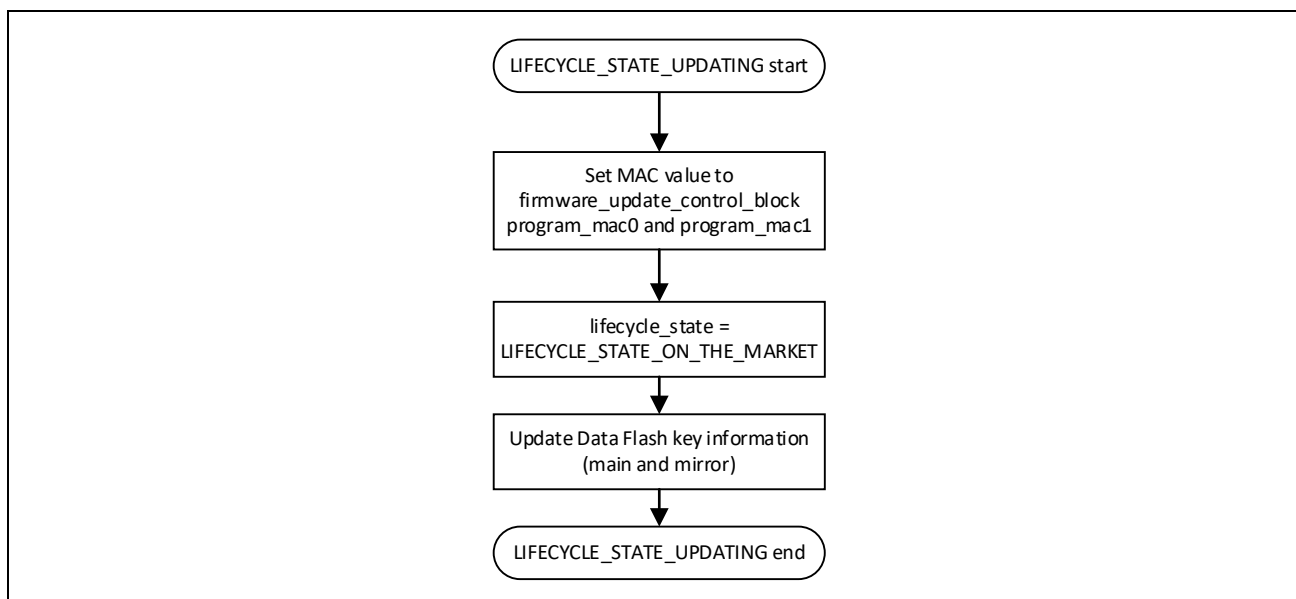


Figure 6.34 The LIFECYCLE\_STATE\_BLANK flow of Secure Bootloader



**Figure 6.35 The LIFECYCLE\_STATE\_UPDATING flow of Secure Bootloader**

It is recommended that the API which is used to inject key is deleted after the key injection. Though the API is deleted in the demo project, the process of the demo project continues.

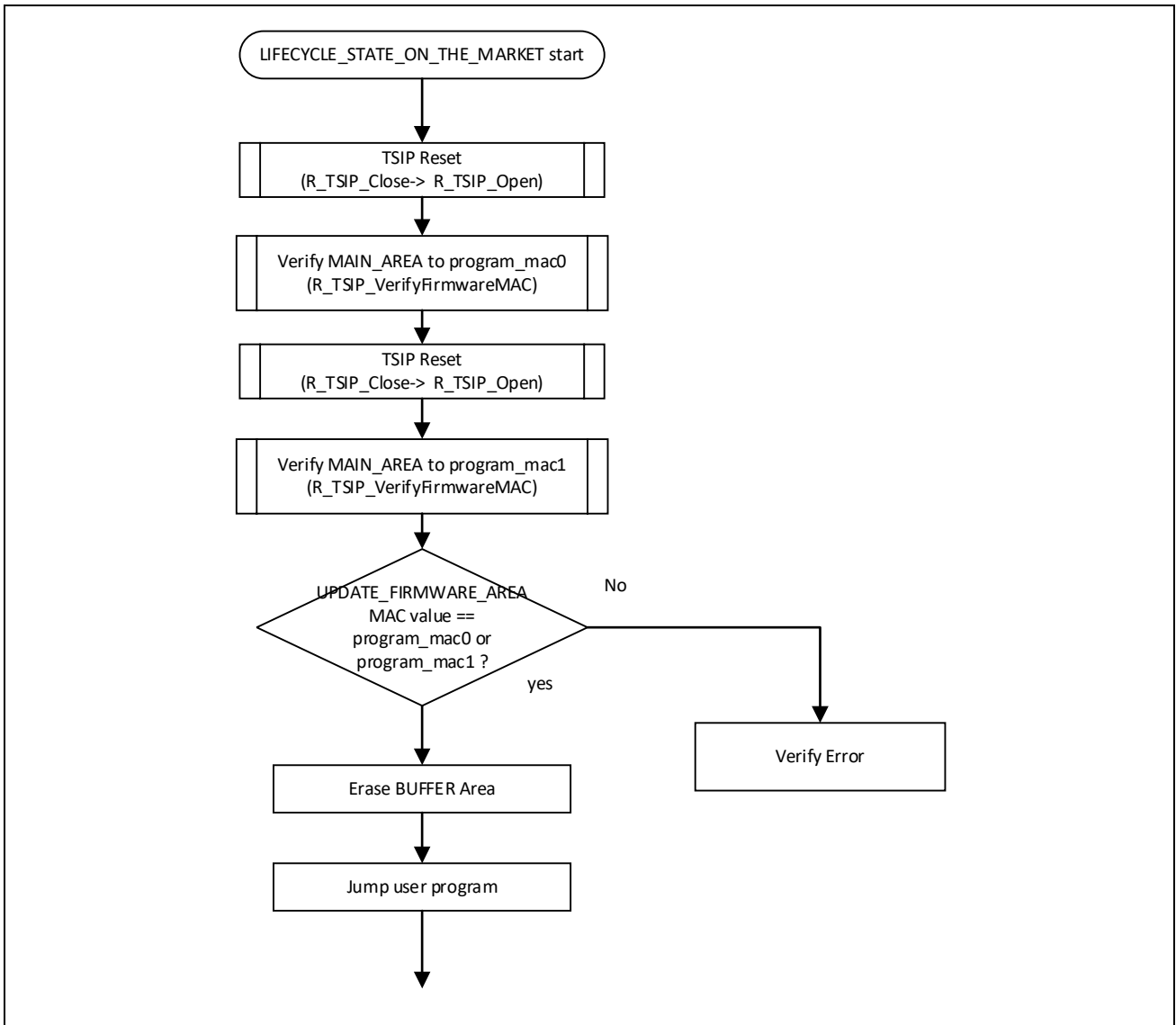


Figure 6.36 The LIFECYCLE\_STATE\_ON\_THE\_MARKET flow of Secure Bootloader

Before “Jump to user program,” run R\_USB\_Close, R\_TSIP\_Close, R\_FLASH\_Close, and R\_SCI\_Close.

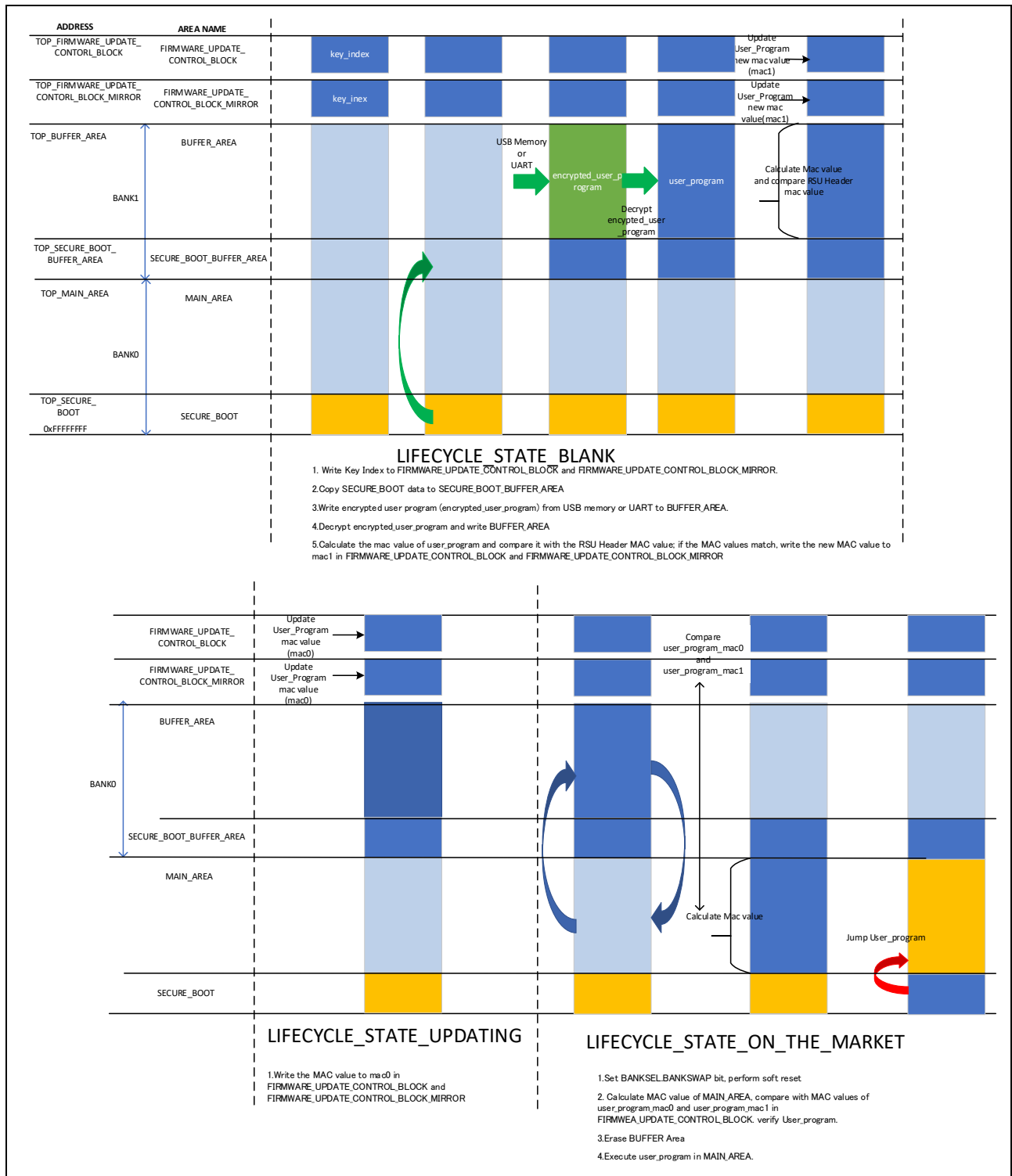
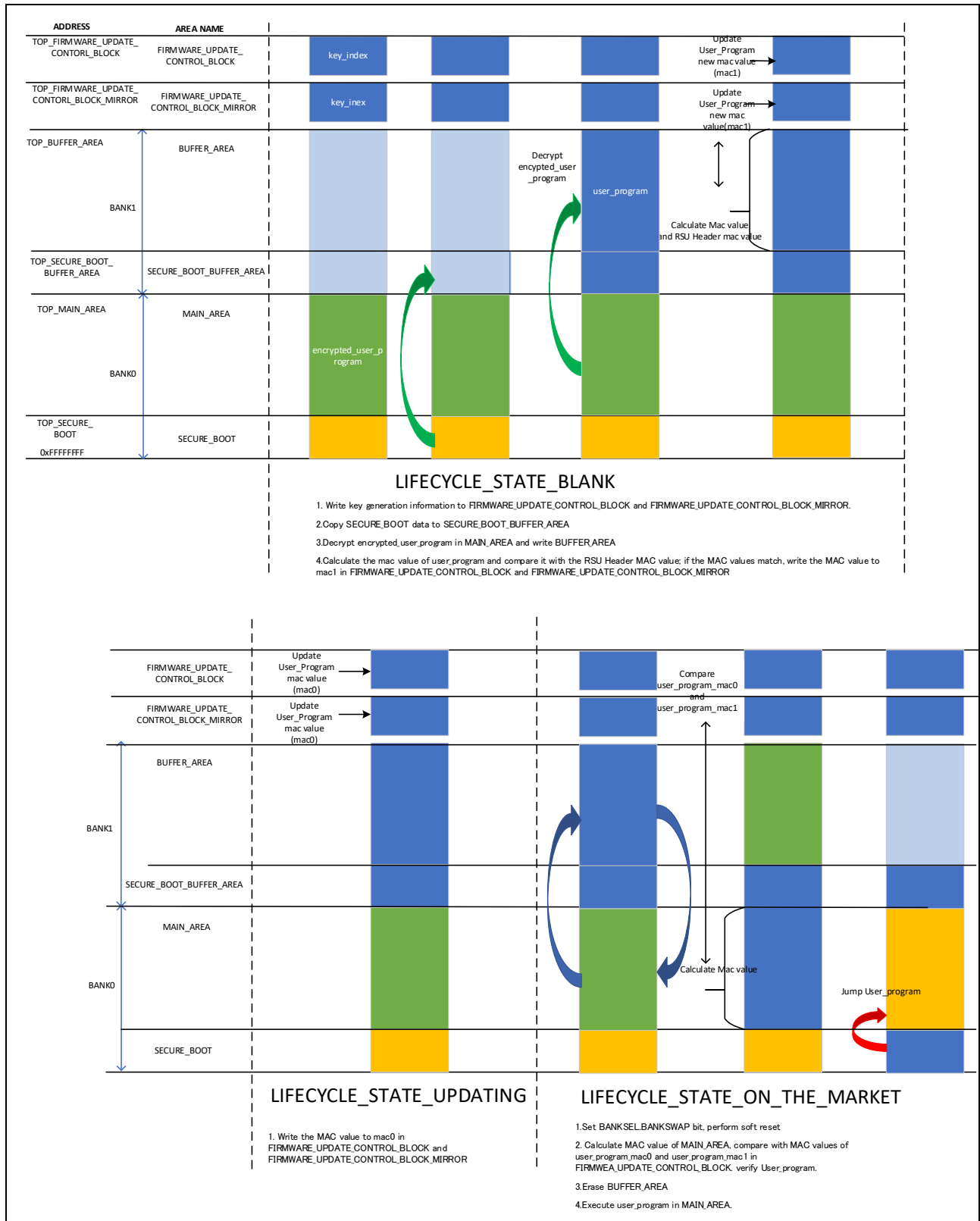


Figure 6.37 Secure Bootloader On-Chip Flash Memory Status in Each lifecycle\_state (Device with Dual Bank Support)





**Figure 6.38 Secure Bootloader On-Chip Flash Memory Status in Each lifecycle\_state (Factory Programming of Device with Dual Bank Support)**

**Table 6-11 Addresses of Areas Set in Secure Bootloader Project (Device with Dual Bank Support)**

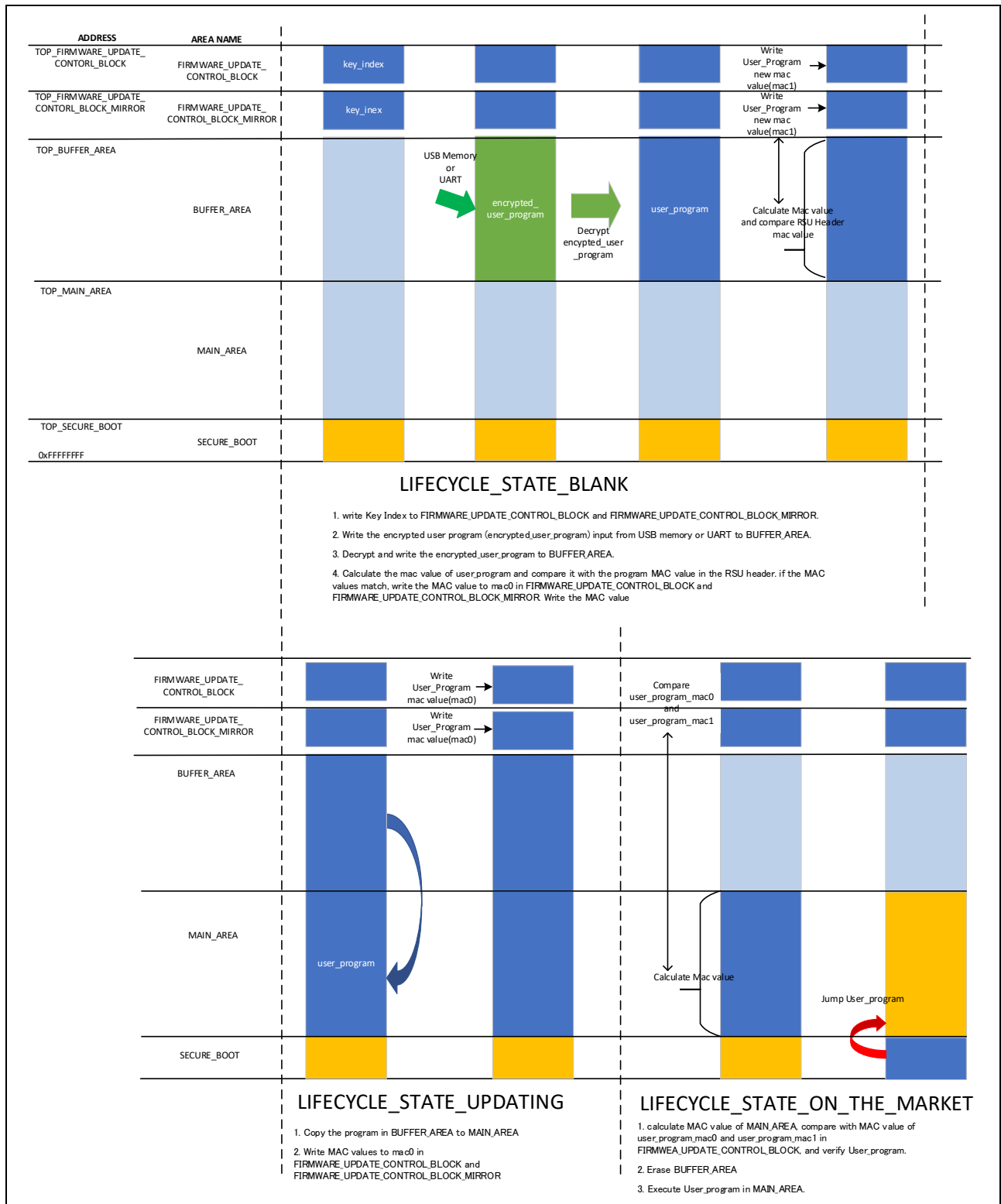
|                                                      | <b>RX26T</b> | <b>RX65N</b> | <b>RX671</b> | <b>RX72M,RX72N</b> |
|------------------------------------------------------|--------------|--------------|--------------|--------------------|
| TOP_FIRMWARE_UPD<br>ATE_<br>CONTORL_BLOCK            | 0x00100000   | 0x00100000   | 0x00100000   | 0x00100000         |
| TOP_FIRMWARE_UPD<br>ATE_<br>CONTORL_BLOCK_MIR<br>ROR | 0x00102000   | 0x00104000   | 0x00101000   | 0x00104000         |
| TOP_BUFFER_AREA                                      | 0xFFF80000   | 0xFFE00000   | 0xFFE00000   | 0xFFC00000         |
| TOP_SECURE_BOOT_<br>BUFFER_AREA                      | 0xFFFB0000   | 0xFFEF0000   | 0xFFEF0000   | 0xFFDF0000         |
| TOP_MAIN_AREA                                        | 0xFFFC0000   | 0xFFF00000   | 0xFFF00000   | 0xFFE00000         |
| TOP_SECURE_<br>BOOT_AREA                             | 0xFFFF0000   | 0xFFFF0000   | 0xFFFF0000   | 0xFFFF0000         |

The secure bootloader project places modules described in the SECURE\_BOOT section. For this reason, changes have been made to the source code of each FIT module.

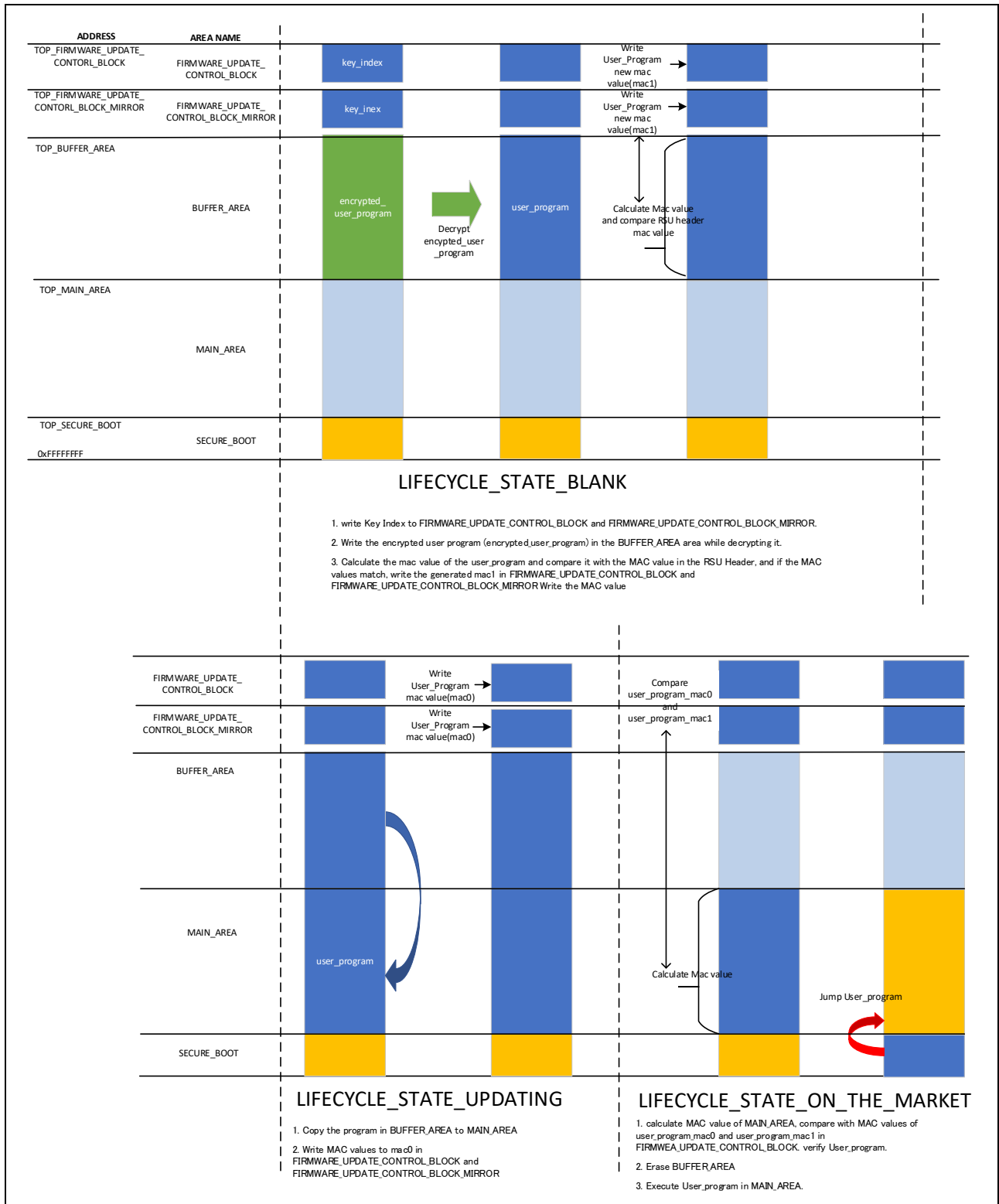
Table 6-12 FIT Modules and Source Code Requiring Changes

| Module  | Folder                     | File                                                    |                                                |
|---------|----------------------------|---------------------------------------------------------|------------------------------------------------|
| bsp     | r_bsp\board\generic_rxXXX  | hwsetup.c                                               |                                                |
|         |                            | r_bsp\mcu\all                                           | dbstc.c                                        |
|         |                            | lowlvl.c                                                |                                                |
|         |                            | lowsrc.c                                                |                                                |
|         |                            | mcu_locks.c                                             |                                                |
|         |                            | r_bsp_common.c                                          |                                                |
|         |                            | r_bsp_cpu.c                                             |                                                |
|         |                            | r_bsp_interrupts.c                                      |                                                |
|         |                            | r_bsp_locking.c                                         |                                                |
|         |                            | r_bsp_mcu_startup.c                                     |                                                |
|         |                            | r_bsp_software_interrupt.c                              |                                                |
|         |                            | sbrk.c                                                  |                                                |
|         |                            | r_bsp\mcu\rxXXX                                         | mcu_clocks.c                                   |
|         |                            |                                                         | mcu_init.c                                     |
|         |                            |                                                         | mcu_interrupts.h<br>(only RX231, RX66T, RX72T) |
|         |                            |                                                         | mcu_interrupts.c                               |
|         |                            |                                                         | mcu_mapped_interrupts.c<br>(Excluding RX231)   |
|         |                            | r_bsp_vbatt.c<br>(only RX231)                           |                                                |
| byteq   | r_byteq\src                | r_byteq.c                                               |                                                |
| flash   | r_flash\src                | r_flash_fc.c                                            |                                                |
|         |                            | r_flash_group.c                                         |                                                |
|         |                            | r_flash_nofc.c                                          |                                                |
|         |                            | r_flash_rx.c                                            |                                                |
| r_fwup  | r_fwup                     | r_fwup_if.h                                             |                                                |
|         | r_fwup\src                 | r_fwup_boot_loader.c                                    |                                                |
|         |                            | r_fwup.c                                                |                                                |
|         |                            | r_fwup_private.h                                        |                                                |
| sci     | r_sci_rx\src               | r_sci_rx.c                                              |                                                |
|         | r_sci_rx\src\targets\rxXXX | r_sci_rxXXX.c                                           |                                                |
|         |                            | r_sci_rxXXX_data.c                                      |                                                |
| general | general                    | r_cg_hardware_setup.c                                   |                                                |
|         |                            | r_smc_cg.c                                              |                                                |
|         |                            | r_smc_cg_user.c                                         |                                                |
|         |                            | r_smc_interrupt.c                                       |                                                |
| pincfg  | r_pincfg                   | r_sci_rx_pinset.c                                       |                                                |
|         |                            | r_usb_basic_pinset.c<br>(Excluding RX231, RX26T, RX66T) |                                                |
|         |                            | r_usb_basic_mini_pinset.c<br>(only RX231)               |                                                |

\*Note : rxXXX contains the RX group name.



**Figure 6.39 Secure bootloader Internal Flash memory status for each lifecycle\_state (Device without Dual Bank Support)**



**Figure 6.40 Secure bootloader Internal Flash memory status for each lifecycle\_state (Factory Programming of Device with Dual Bank Support)**

**Table 6-13 Addresses of Areas Set in Secure Bootloader Project (Device without Dual Bank Support)**

|                                              | <b>RX231</b> | <b>RX66T</b> | <b>RX72T</b> |
|----------------------------------------------|--------------|--------------|--------------|
| TOP_FIRMWARE_UPDATE_<br>CONTORL_BLOCK        | 0x00100000   | 0x00100000   | 0x00100000   |
| TOP_FIRMWARE_UPDATE_<br>CONTORL_BLOCK_MIRROR | 0x00101000   | 0x00104000   | 0x00104000   |
| TOP_BUFFER_AREA                              | 0xFFFF80000  | 0xFFFF80000  | 0xFFFF00000  |
| TOP_MAIN_AREA                                | 0xFFFFB8000  | 0xFFFFB8000  | 0xFFFF78000  |
| TOP_SECURE_BOOT_AREA                         | 0xFFFFF0000  | 0xFFFFF0000  | 0xFFFFF0000  |

The secure bootloader project places modules described in the SECURE\_BOOT section. For this reason, changes have been made to the source code of each FIT module.

6.2.2.2 Firmware Update Project

The firmware update project decrypts and verifies the user program. A flowchart of the firmware update project after receiving the update command is shown below.

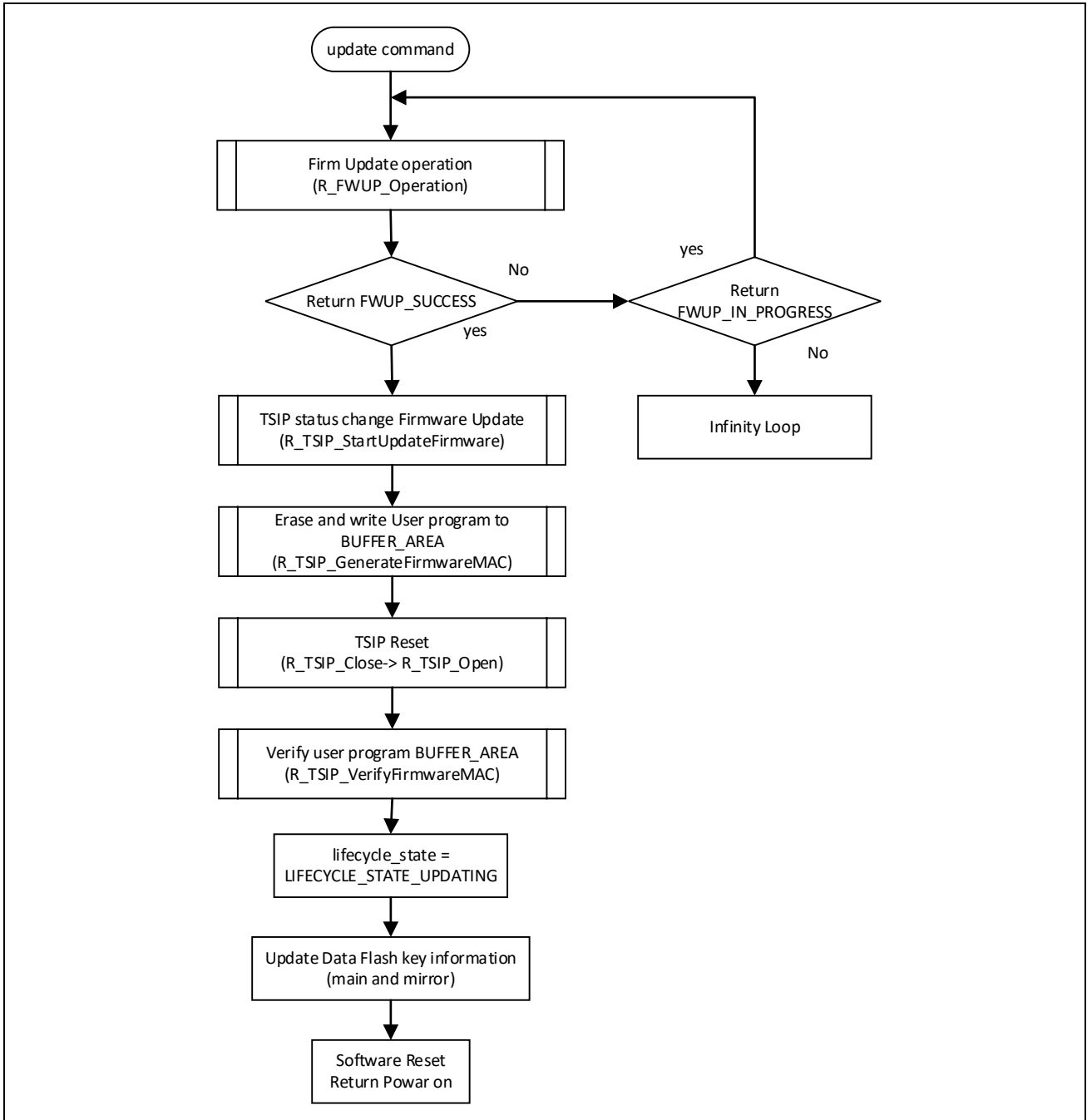


Figure 6.41 Firmware Update Flowchart

### 6.2.3 Secure Bootloader and Firmware Update Execution Example

Firmware update operation using UART is supported on all the board. In addition, firmware update operation using a USB flash drive is supported on boards populated with a device with USB HOST functionality and equipped with a USB host socket.

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

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

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

UFPK = 22222222222222222222222222221111111111111111111111111111111111111111

Key Encryption Key = 0123456789abcdef0123456789abcdef

- Command line version

Run the following command.

```
> skmt.exe /genkey /iv "55aa55aa55aa55aa55aa55aa55aa55aa55aa55aa55aa55aa55aa" /ufpk
file="{skmt_loc}\..\key\sample.key"
 /wufpk file="{skmt_loc}\key\sample.key_enc.key"
 /mcu "RX-TSIP" /keytype "AES-128" /key "0123456789ABCDEF0123456789ABCDEF"
 /filetype "csource" /keyname "euk_aes128" /output "{skmt_loc}\genkey\euk_aes128.c"
```

For {skmt\_loc}, substitute the path of the secure bootloader project folder.

- Standalone version

Enter the following values.

— **Key Type** tab

Select **AES – 128 bits**.

— **Key Data** tab

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

— **Wrapping Key**

UFPK File: **sample\_key.key** included in the sample

W-UFPK File: **sample\_key\_enc.key** included in the sample

— **IV**

Select **Use specified value** and enter “**55aa55aa55aa55aa55aa55aa55aa55aa55aa**”.

— **Output**

Format: C source

File: euk\_aes128.c

Key name: euk\_aes128

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

The file rxXXX\_SecureUpdate.skmt is in xml format and can be edited as text. For {skmt\_loc} in rxXXX\_SecureUpdate.skmt, substitute the path of the secure bootloader project folder.



The screenshot shows the 'Wrap Key Tab' with the following configuration:

- Key Type:** AES (selected), 128 bits, DLM-SSD (dropdown).
- Wrapping Key:** UFPK (selected). UFPK File: `{$skmt_loc}%key%sample.key`. W-UFPK File: `{$skmt_loc}%key%sample.key_enc.key`.
- IV:** Generate random value (selected). Use specified value: `00112233445566778899AABBCCDDEEFF`.
- Output:** Format: C Source. File: `{$skmt_loc}%src%genkey%euk_aes128.c`. Address: 10000. Key name: `euk_aes128`.

A 'Generate file' button is located at the bottom of the tab.

Figure 6.42 Generating a Key Data File (Wrap Key Tab)

The screenshot shows the 'Key Data Tab' with the following configuration:

- Key Type:** Raw (selected).
- Key Data:** `0123456789abcdef0123456789abcdef`

Figure 6.43 Generating a Key Data File (Wrap Key Tab – Key Data Tab)

## (2) Building the Secure Bootloader Project

After importing the secure bootloader project (rxXXX\_bbb\_tsip\_secure\_boot) into the e<sup>2</sup> studio workspace, place euk\_aes128.c and euk\_aes128.h generated in step (1) in the src folder of the rxXXX\_bbb\_tsip\_secure\_boot project.

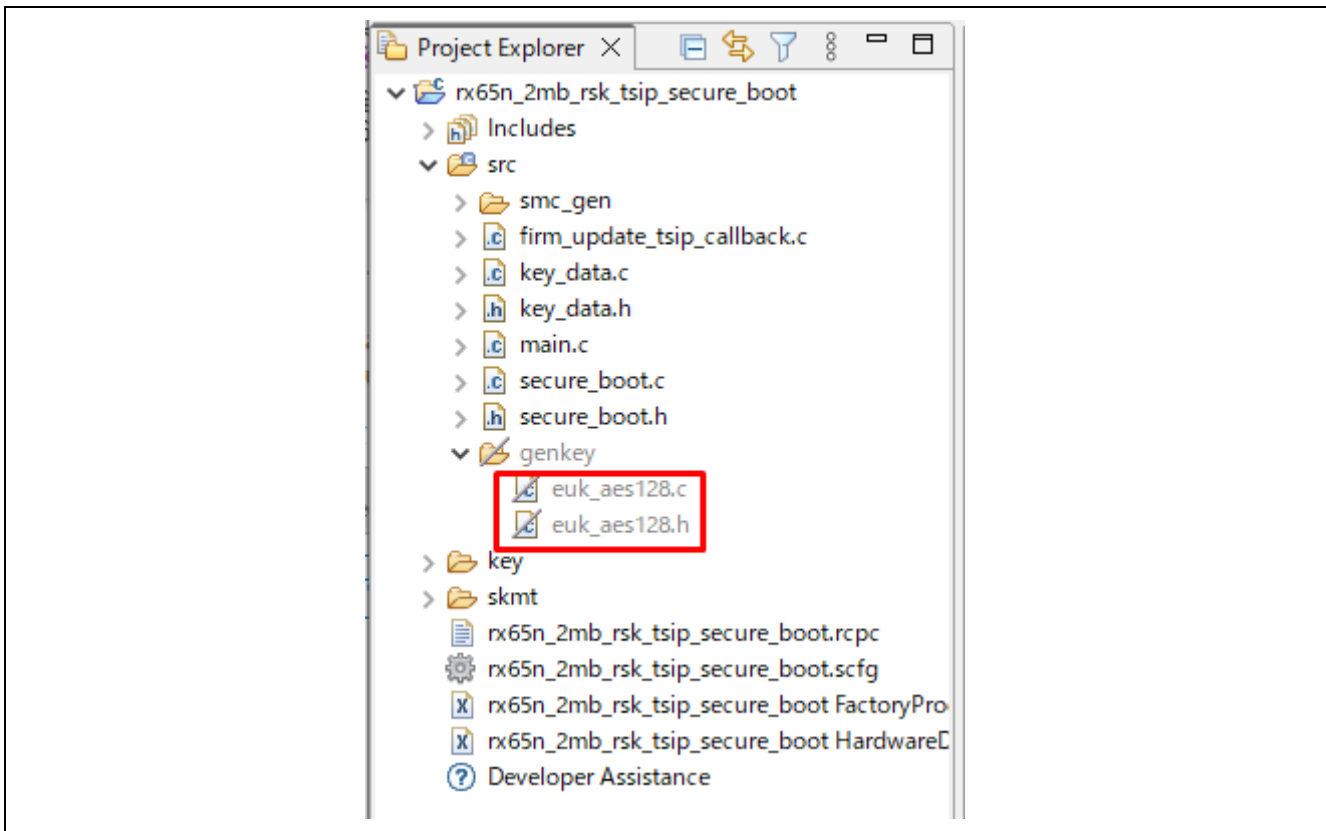


Figure 6.44 e<sup>2</sup> studio Project Explorer

Also, to use USB flash drive with the supported board, in r\_fwup\_config.h in src\smc\_gen\r\_config, set

```
#define FWUP_CFG_COMMUNICATION_FUNCTION
```

to (2). After making the changes, build the project in e<sup>2</sup> studio.

## (3) Building the Firmware Update Project

After importing the firmware update project (rxXXX\_bbb\_tsip\_user\_program) into the e<sup>2</sup> studio workspace, use USB flash drive with the supported board, as in step (2), in r\_fwup\_config.h in src\smc\_gen\r\_config, set

```
#define FWUP_CFG_COMMUNICATION_FUNCTION
```

to (2). After making the changes, build the project in e<sup>2</sup> studio.

## (4) Encrypting the Firmware Update Program

- Command line version  
Run the following command.

```
> skmt.exe /enctsip /mode "update" /ver "1" /prg
"{skmt_loc}\..\rx65n_2mb_rsk_tsip_user_program\Release_rx65n\rx65n_2mb_rsk_ts
ip_user_program.mot"
/enckey "0123456789abcdef0123456789abcdef"
/startaddr "FFE00300" /endaddr "FFFFFFF" /imgflg "testing" /filetype "bin"
/output ""{skmt_loc}\userprog.rsu"
```

For {skmt\_loc}, substitute the path of the secure bootloader project folder.

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

**Table 6-14 Selections for Each MCU**

| MCU   | Parameter     |            | Address  |
|-------|---------------|------------|----------|
|       | GUI           | CLI        |          |
| RX231 | Start address | /startaddr | FFFB8300 |
| RX66T | End address   | /endaddr   | FFFFFFF  |
| RX72T | Start address | /startaddr | FFF78300 |
|       | End address   | /endaddr   | FFFFFFF  |
| RX26T | Start address | /startaddr | FFFC0300 |
|       | End address   | /endaddr   | FFFFFFF  |
| RX65N | Start address | /startaddr | FFF00300 |
| RX671 | End address   | /endaddr   | FFFFFFF  |
| RX72M | Start address | /startaddr | FFE00300 |
| RX72N | End address   | /endaddr   | FFFFFFF  |

- Standalone version
  - Enter the following values.
    - **Output Image**  
Select **Secure Update**.
    - **Firmware Image**  
MOT file output by update project (rxXXX\_bbb\_tsip\_user\_program.mot)
    - **Encrypted Address Range** tab  
The values differ depending on the MCU. Refer to Table 6.13 and select the appropriate values to match the MCU.
    - **IV** tab  
Select **Use specified value** and enter “55aa55aa55aa55aa55aa55aa55aa”.
    - **RSU Header** tab  
Image Flag: TESTING  
RSU header Ver: 1
    - **Output**  
Format: Binary  
File: userprog.rsu

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

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

The screenshot shows a software interface for generating an encrypted file. At the top, there are three main input fields: 'Output Image' with a dropdown menu set to 'Secure Update', 'Firmware Image' with a text box containing a path and a 'Browse...' button, and 'Secure Boot Image' with a text box and a 'Browse...' button. Below these is a tabbed interface with four tabs: 'Encryption address range', 'Image Encryption Key', 'IV', and 'RSU header'. The 'Encryption address range' tab is selected and contains three input fields: 'start address' (FFF00300), 'end address' (FFFEFFFF), and 'Encrypted image output address'. Below the tabs is an 'Output' section with a 'Format' dropdown set to 'Binary' and a 'File' text box with a path and a 'Browse...' button. At the bottom of the interface is a large 'Generate file' button.

Figure 6.45 Generating an Encrypted File (TSIP UPDATE Tab)

Encryption address range | Image Encryption Key | IV | RSU header

**Key Encryption Key**

0123456789abcdef0123456789abcdef

**Image Encryption Key**

Generate random value

Use specified value (32 hex bytes, big endian format)

**Figure 6.46** Generating an Encrypted File (TSIP UPDATE Tab – Image Encryption Key Tab)

Encryption address range | Image Encryption Key | IV | RSU header

Generate random value

Use specified value (16 hex bytes, big endian format)

**Figure 6.47** Generating an Encrypted File (TSIP UPDATE Tab – IV Tab)

Encryption address range | Image Encryption Key | IV | RSU header

Image Flag: TESTING

RSU header Ver: 1

**Figure 6.48** Generating an Encrypted File (TSIP UPDATE Tab – RSU Header Tab)

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

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

### (5) Starting the Terminal Emulator (Tera Term)

Make connections as shown in the applicable connection diagram in 6.2.1, Demo Project Setup, then launch Tera Term. Refer to Table 6-6, Using RSK RX231, RSK RX66T or RSK RX72T or Table 6-7, Using RSK RX65N-2MB, RSK RX671, RSK RX72M, RSK RX72N, RX72N Envision Kit or MCB RX26T to make serial setting.

### (6) Running the Secure Bootloader Project

Select `rxXXX_bbb_tsip_secure_boot` in e<sup>2</sup> studio's Project Explorer, then click the  button to execute the Secure Bootloader project.

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

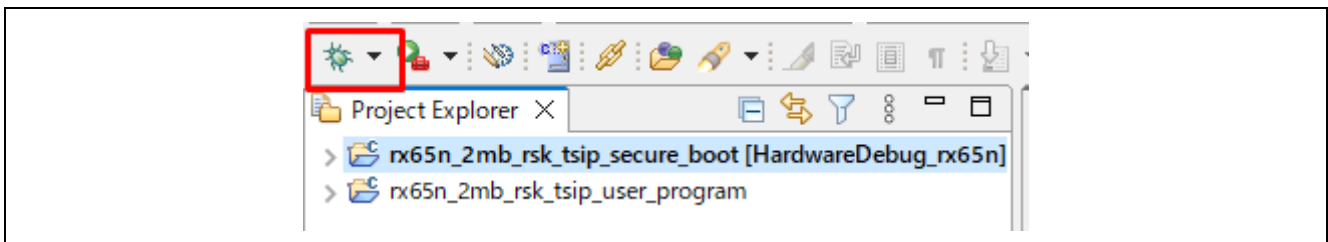


Figure 6.49 Running the Secure Bootloader Project (RSK RX65N)

## (7) Installing the Initial User Program

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

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

```
HELLO!! this is boot program. ~
$ I was built in May 10 2023, 13:37:17.
Checking flash ROM status.
status = LIFECYCLE_STATE_BLANK
bank info = 1. (start bank = 0)
===== generate user key index phase =====
generate aes128 user program mac key index: OK

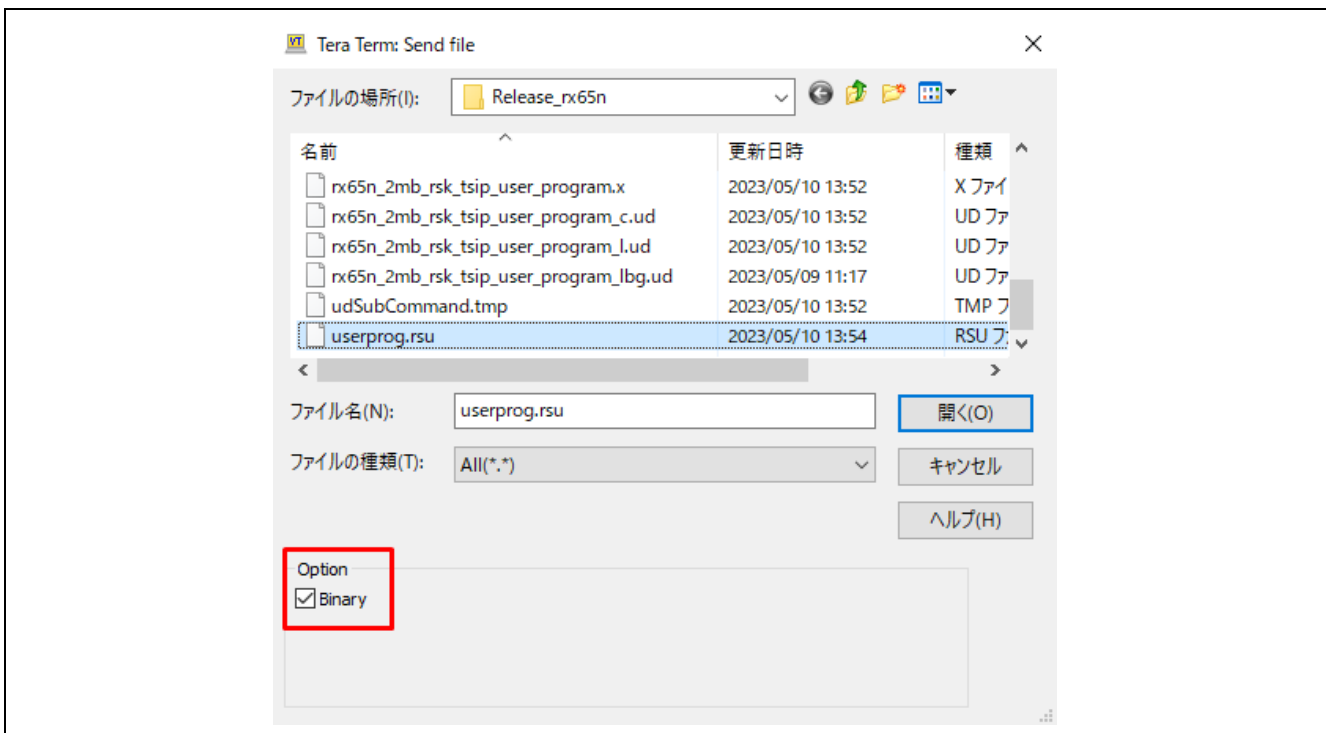
RX65N secure boot program

Checking flash ROM status.
bank 0 status = 0xff [LIFECYCLE_STATE_BLANK]
bank 1 status = 0xff [LIFECYCLE_STATE_BLANK]
bank info = 1. (start bank = 0)
start installing user program.
erase bank1 secure boot mirror area...OK
copy secure boot from bank0 to bank1...OK
erase install area (code flash): OK
send "userprog.rsu" via UART.
```

**Figure 6.50** Log Output of Secure Bootloader up to Wait for Initial User Program Installation (RSK RX65N)

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

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



**Figure 6.51** Tera Term: Send file Dialog Box

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

```

installing firmware...100%(960/960KB).
completed installing firmware.
integrity check scheme = mac-aes128-cmac-with-tsip
bank1(temporary area) on code flash integrity check...transit TSIP status to UpdateFirmware:
OK
extract update file parameters: OK
decrypt program:
100%... OK
code flash MAC check...OK
===== install user key index phase =====
erase data flash(main)...OK
write data flash(main)...OK
erase data flash(mirror)...OK
write data flash(mirror)...OK
data flash setting OK.
software reset...
HELLO!! this is boot program. ~
$ I was built in May 10 2023, 13:37:17.
Checking flash ROM status.
status = LIFECYCLE_STATE_UPDATING
bank info = 1. (start bank = 0)
update data flash
erase data flash(main)...OK
write data flash(main)...OK
erase data flash(mirror)...OK
write data flash(mirror)...OK
data flash setting OK.

RX65N secure boot program

Checking flash ROM status.
bank 0 status = 0xff [LIFECYCLE_STATE_BLANK]
bank 1 status = 0xfe [LIFECYCLE_STATE_TESTING]
bank info = 1. (start bank = 0)
integrity check scheme = mac-aes128-cmac-with-tsip
bank1(temporary area) on code flash integrity check...OK
update LIFECYCLE_STATE from [LIFECYCLE_STATE_TESTING] to [LIFECYCLE_STATE_VALID]
bank1(temporary area) block0 erase (to update LIFECYCLE_STATE)...OK
bank1(temporary area) block0 write (to update LIFECYCLE_STATE)...OK
swap bank...[FWUP_STATE_BANK1_UPDATE_LIFECYCLE_WRITE_COMPLETE and LIFECYCLE_STATE_VALID]
HELLO!! this is boot program. ~
$ I was built in May 10 2023, 13:37:17.
Checking flash ROM status.
status = LIFECYCLE_STATE_ON_THE_MARKET
bank info = 0. (start bank = 1)

RX65N secure boot program

Checking flash ROM status.
bank 0 status = 0xf8 [LIFECYCLE_STATE_VALID]
bank 1 status = 0xff [LIFECYCLE_STATE_BLANK]
bank info = 0. (start bank = 1)
integrity check scheme = mac-aes128-cmac-with-tsip
bank0(execute area) on code flash integrity check...OK
erase install area (code flash): OK
data flash(main) hash check...OK
data flash(mirror) hash check...OK
secure boot sequence: success.
□HELLO!! this is user program. ~
$ I was built in May 9 2023, 17:19:52.
Version ver 1.00.
tsip@rx65n
$

```

**Figure 6.52 Installation Log Message Output for Secure Bootloader Initial Firmware (RSK RX65N)**



### (8) Firmware Update Operation

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

```
/* Command prompt related */
#define PROMPT ("tsip@rx65n\r\n$ ")
#define VERSION ("ver 1.01")
#define HELLO_MESSAGE ("HELLO!! this is user program. ~\r\n$ ")
```

Figure 6.53 Modified Location in main.c (rx65n\_2mb\_rsk\_tsip\_user\_program)

Execute the **update** command. Install the user program that has been updated to Ver 1.01 as described in step (7). After the upload finishes, the system reboots.

```

tsip@rx65n
$
install start ...
[INFO] Receive file created.
[INFO] -----
[INFO] Firmware update user program
[INFO] -----
[INFO] Send Update firmware via UART.
[INFO] Flash Write: Address = 0xFFE00000, length = 1024byte ... OK
[INFO] Flash Write: Address = 0xFFE00400, length = 1024byte ... OK
--- (Omitted) ---
[INFO] Flash Write: Address = 0xFFEEF400, length = 1024byte ... OK
[INFO] Flash Write: Address = 0xFFEEF800, length = 1024byte ... OK
[INFO] Flash Write: Address = 0xFFEEFC00, length = 1024byte ... OK
[INFO] Firmware update to Flash is complete.
[INFO] Firmware update to Flash is complete.
[INFO] Check signature of update firmware is complete.
[INFO] Testing.
[FWUP_main] Firmware update completely.
transit TSIP status to UpdateFirmware: OK
extract update file parameters: OK
decrypt program:
: 100%... OK
code flash MAC check...OK
 100 % 100 % ... install complete
HELLO!! this is boot program. ~
$ I was built in May 10 2023, 13:37:17.
Checking flash ROM status.
status = LIFECYCLE_STATE_UPDATING
bank info = 0. (start bank = 1)
update data flash
erase data flash(main)...OK
write data flash(main)...OK
erase data flash(mirror)...OK
write data flash(mirror)...OK
data flash setting OK.

RX65N secure boot program

Checking flash ROM status.
bank 0 status = 0xf8 [LIFECYCLE_STATE_VALID]
bank 1 status = 0xfe [LIFECYCLE_STATE_TESTING]
bank info = 0. (start bank = 1)
integrity check scheme = mac-aes128-cmac-with-tsip
bank1(temporary area) on code flash integrity check...OK
update LIFECYCLE_STATE from [LIFECYCLE_STATE_TESTING] to [LIFECYCLE_STATE_VALID]
bank1(temporary area) block0 erase (to update LIFECYCLE_STATE)...OK
bank1(temporary area) block0 write (to update LIFECYCLE_STATE)...OK
swap bank...[FWUP_STATE_BANK1_UPDATE_LIFECYCLE_WRITE_COMPLETE and LIFECYCLE_STATE_VALID]
HELLO!! this is boot program. ~
$ I was built in May 10 2023, 13:37:17.
Checking flash ROM status.
status = LIFECYCLE_STATE_ON_THE_MARKET
bank info = 1. (start bank = 0)

RX65N secure boot program

Checking flash ROM status.
bank 0 status = 0xf8 [LIFECYCLE_STATE_VALID]
bank 1 status = 0xf8 [LIFECYCLE_STATE_VALID]
bank info = 1. (start bank = 0)
integrity check scheme = mac-aes128-cmac-with-tsip
bank0(execute area) on code flash integrity check...OK
erase install area (code flash): OK
data flash(main) hash check...OK
data flash(mirror) hash check...OK
secure boot sequence: success.
☐HELLO!! this is user program. ~
$ I was built in May 9 2023, 17:19:52.
Version ver 1.01.
tsip@rx65n
$

```

Figure 6.54 Firmware Update Log Message Output (RSK RX65N)

## 6.2.4 Execution Example of Factory Programming Using Motorola S Format File Containing Secure Bootloader and Encrypted User Program

This section describes how to create a Motorola S format file containing a secure bootloader and encrypted user program and how to program it to a device using Renesas Flash Programmer (factory programming).

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

Generate a key file to encrypt the image. The procedure is same to 6.2.3, Secure Bootloader and Firmware Update Execution Example.

### (2) Building the Secure Bootloader Project

After importing the secure bootloader project (rxXXX\_bbb\_tsip\_secure\_boot) into the e<sup>2</sup> studio workspace, place euk\_aes128.c and euk\_aes128.h generated in step (1) in the src folder of the rxXXX\_bbb\_tsip\_secure\_boot project.

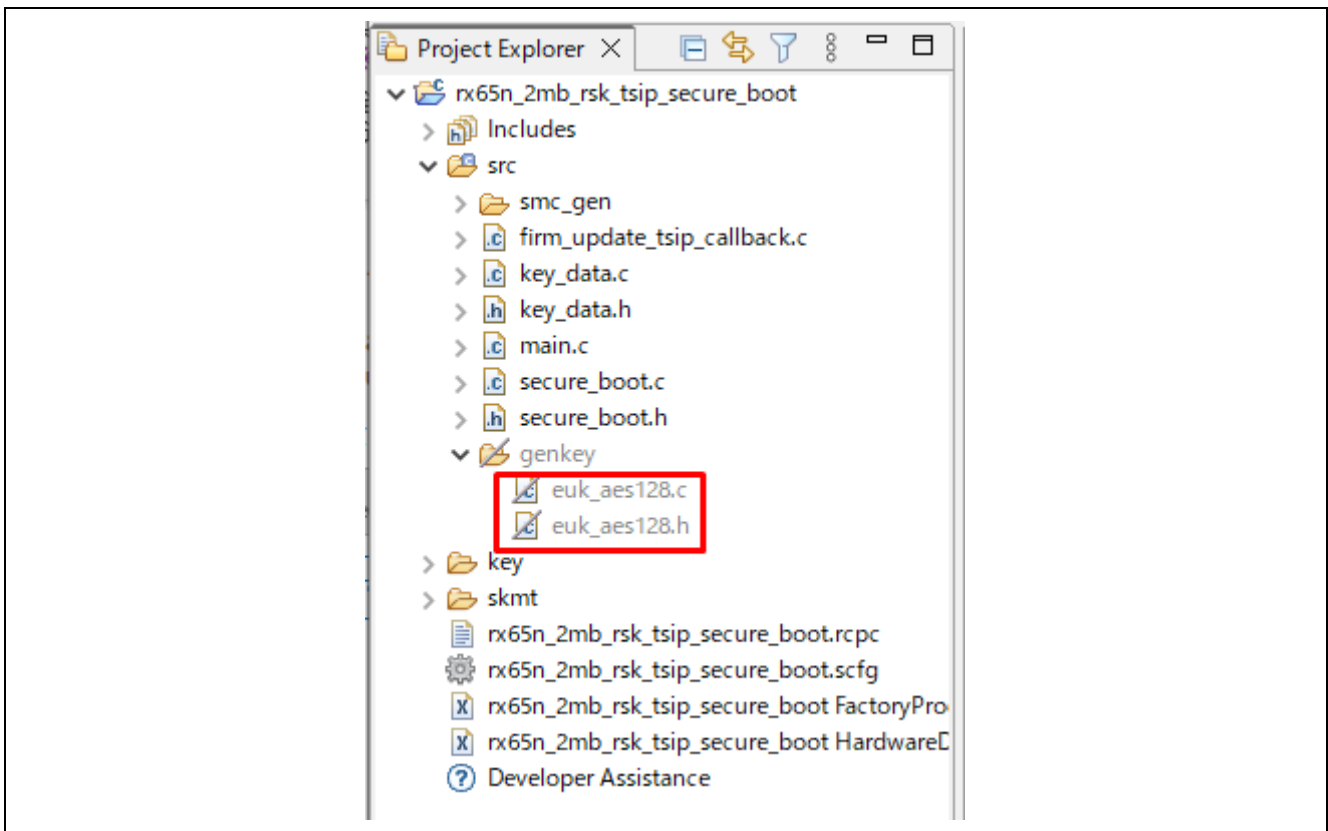
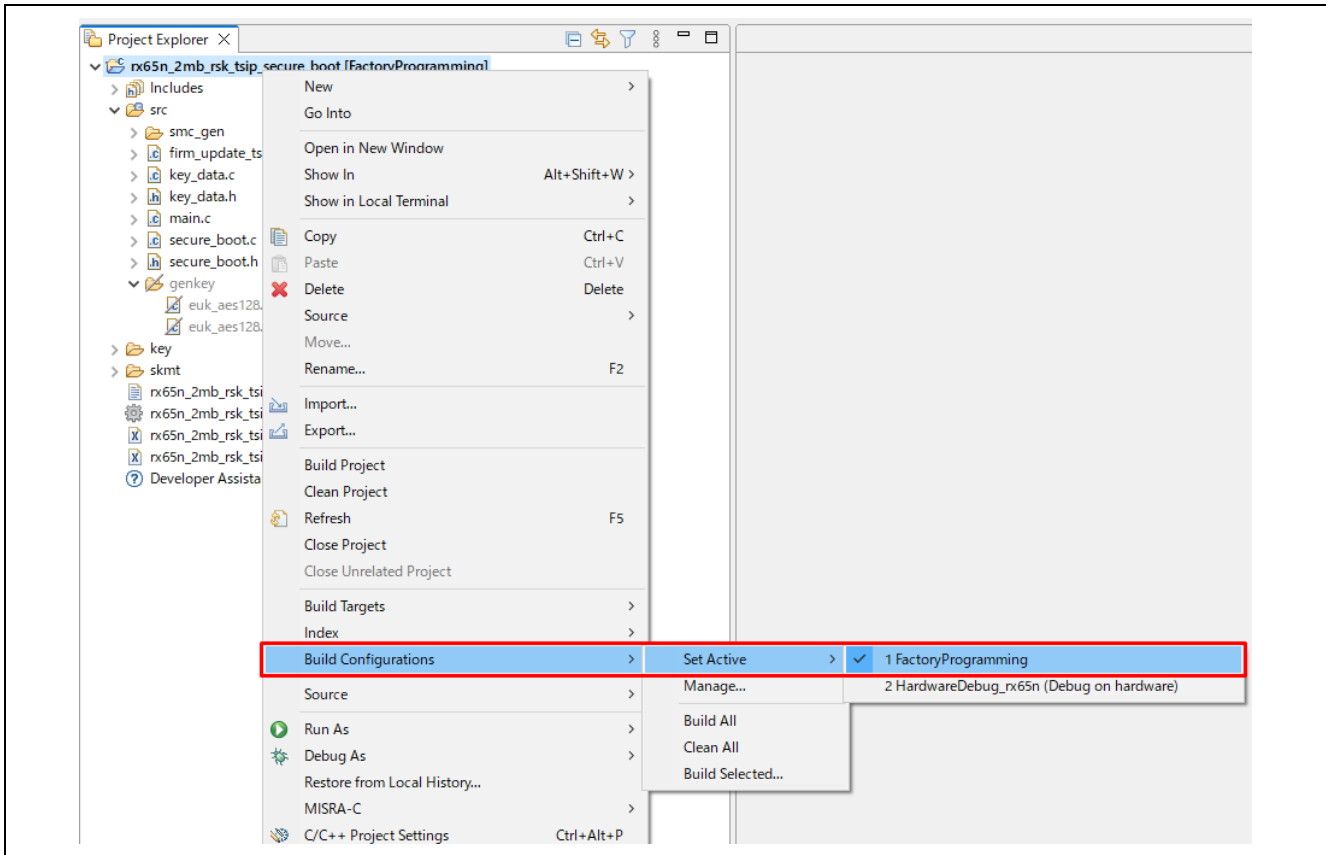


Figure 6.55 e<sup>2</sup> studio Project Explorer

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

In e<sup>2</sup> studio's Project Explorer, right-click the secure update project and on the menu that appears select **Build Configuration > Activate > Factory Programming**.



**Figure 6.56 Changing Build Configuration to Factory Programming**

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

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

### (3) Building the Firmware Update Project

Build the project with the same operation of 6.2.3, Secure Bootloader and Firmware Update Execution Example.

**(4) Encrypting the Firmware Update Program**

- Command line version

Run the following command.

```
> skmt.exe /enctsip /mode "factory" /ver "1" /prg
 "${skmt_loc}\..\rx65n_2mb_rsk_tsip_user_program\Release_rx65n\rx65n_2mb_rsk_tsip
 _user_program.mot"
 /prg_sb "${skmt_loc}\FactoryProgramming\rx65n_2mb_rsk_tsip_secure_boot.mot "
 /enckey "0123456789abcdef0123456789abcdef"
 /startaddr "FFE00300" /endaddr "FFFFFFF" /destaddr "FFE00300"
 /imgflg "valid" /filetype "mot"
 /output "" "${skmt_loc}\userprog.mot"
```

For {skmt\_loc}, substitute the path of the secure bootloader project folder.

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

**Table 6-15 Selections for Each MCU**

| MCU            | Parameter                      |            | Address  |
|----------------|--------------------------------|------------|----------|
|                | GUI                            | CLI        |          |
| RX231<br>RX66T | Start address                  | /startaddr | FFFB8300 |
|                | End address                    | /endaddr   | FFFFFFF  |
|                | Encrypted image output address | /destaddr  | FFFB8300 |
| RX72T          | Start address                  | /startaddr | FFF78300 |
|                | End address                    | /endaddr   | FFFFFFF  |
|                | Encrypted image output address | /destaddr  | FFF78300 |
| RX26T          | Start address                  | /startaddr | FFFC0300 |
|                | End address                    | /endaddr   | FFFFFFF  |
|                | Encrypted image output address | /destaddr  | FFFC0300 |
| RX65N<br>RX671 | Start address                  | /startaddr | FFF00300 |
|                | End address                    | /endaddr   | FFFFFFF  |
|                | Encrypted image output address | /destaddr  | FFF00300 |
| RX72M<br>RX72N | Start address                  | /startaddr | FFE00300 |
|                | End address                    | /endaddr   | FFFFFFF  |
|                | Encrypted image output address | /destaddr  | FFE00300 |

- Standalone version

Enter the following values.

— **Output Image**

Select **Secure Update**.

— **Firmware Image**

MOT file output by update project (rxXXX\_bbb\_tsip\_user\_program.mot)

— **Encrypted Address Range** tab

The values differ depending on the MCU. Refer to Table 6.15 and select the appropriate values to match the MCU.

— **IV** tab

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

— **RSU Header** tab

Image Flag: TESTING

RSU header Ver: 1

— **Output**

Format: Binary

File: userprog.mot

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

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

**Figure 6.57** Generating an Encrypted File (TSIP UPDATE Tab)

**Figure 6.58** Generating an Encrypted File (TSIP UPDATE Tab – Image Encryption Key Tab)

The screenshot shows a software interface with four tabs: "Encryption address range", "Image Encryption Key", "IV", and "RSU header". The "IV" tab is active. It contains two radio button options: "Generate random value" (which is selected) and "Use specified value (16 hex bytes, big endian format)". A text input field is visible next to the second option.

**Figure 6.59 Generating an Encrypted File (TSIP UPDATE Tab – IV Tab)**

The screenshot shows the same software interface with the "RSU header" tab active. It features two dropdown menus: "Image Flag:" with the value "VALID" selected, and "RSU header Ver:" with the value "1" selected.

**Figure 6.60 Generating an Encrypted File (TSIP UPDATE Tab – RSU Header Tab)**

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

### (5) Starting the Terminal Emulator (Tera Term)

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

### (6) Writing Data Using Renesas Flash Programmer

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

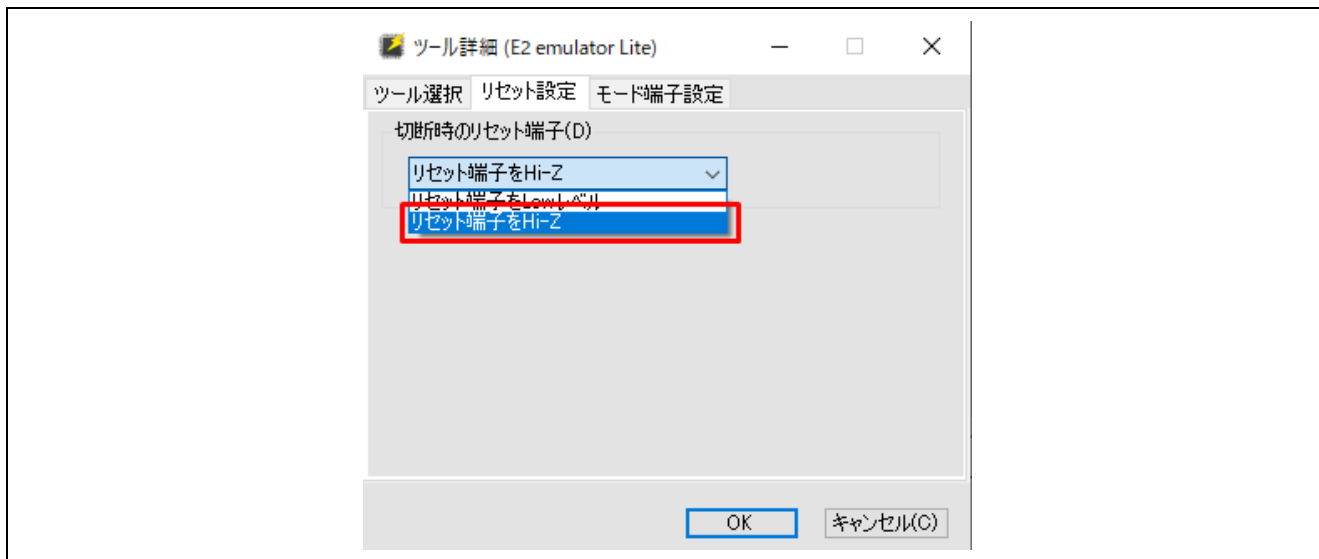


Figure 6.61 Renesas Flash Programmer Reset Settings Tab



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

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

```
HELLO!! this is boot program. ~
$ I was built in May 9 2023, 17:15:53.
Checking flash ROM status.
status = LIFECYCLE_STATE_BLANK
bank info = 1. (start bank = 0)
===== generate user key index phase =====
generate aes128 user program mac key index: OK
Copy Secure Boot...OK
Decrypt User Program.
transit TSIP status to UpdateFirmware: OK
extract update file parameters: OK
decrypt program:
100%... OK
code flash MAC check...OK
===== install user key index phase =====
erase data flash(main)...OK
write data flash(main)...OK
erase data flash(mirror)...OK
write data flash(mirror)...OK
data flash setting OK.
software reset...
HELLO!! this is boot program. ~
$ I was built in May 9 2023, 17:15:53.
Checking flash ROM status.
status = LIFECYCLE_STATE_UPDATING
bank info = 1. (start bank = 0)
update data flash
erase data flash(main)...OK
write data flash(main)...OK
erase data flash(mirror)...OK
write data flash(mirror)...OK
data flash setting OK.
```

**Figure 6.62 Log Output when Using Renesas Flash Programmer to Write Encrypted User Program (1) (RSK RX65N)**

```

RX65N secure boot program

Checking flash ROM status.
bank 0 status = 0xf8 [LIFECYCLE_STATE_VALID]
bank 1 status = 0xfe [LIFECYCLE_STATE_TESTING]
bank info = 1. (start bank = 0)
integrity check scheme = mac-aes128-cmac-with-tsip
bank1(temporary area) on code flash integrity check...OK
update LIFECYCLE_STATE from [LIFECYCLE_STATE_TESTING] to [LIFECYCLE_STATE_VALID]
bank1(temporary area) block0 erase (to update LIFECYCLE_STATE)...OK
bank1(temporary area) block0 write (to update LIFECYCLE_STATE)...OK
swap bank...[FWUP_STATE_BANK1_UPDATE_LIFECYCLE_WRITE_COMPLETE and LIFECYCLE_STATE_VALID]
HELLO!! this is boot program. ~
$ I was built in May 9 2023, 17:15:53.
Checking flash ROM status.
status = LIFECYCLE_STATE_ON_THE_MARKET
bank info = 0. (start bank = 1)

RX65N secure boot program

Checking flash ROM status.
bank 0 status = 0xf8 [LIFECYCLE_STATE_VALID]
bank 1 status = 0xf8 [LIFECYCLE_STATE_VALID]
bank info = 0. (start bank = 1)
integrity check scheme = mac-aes128-cmac-with-tsip
bank0(execute area) on code flash integrity check...OK
erase install area (code flash): OK
data flash(main) hash check...OK
data flash(mirror) hash check...OK
secure boot sequence: success.
□HELLO!! this is user program. ~
$ I was built in May 10 2023, 13:52:34.
Version ver 1.00.
tsip@rx65n
$

```

**Figure 6.63 Log Output when Using Renesas Flash Programmer to Write Encrypted User Program (2) (RSK RX65N)**

### (7) Firmware Update Operation

For firmware update operation, refer to step (8), Firmware Update Operation, under 6.2.3, Secure Bootloader and Firmware Update Execution Example.


### 6.2.5 Debugging User Program Projects

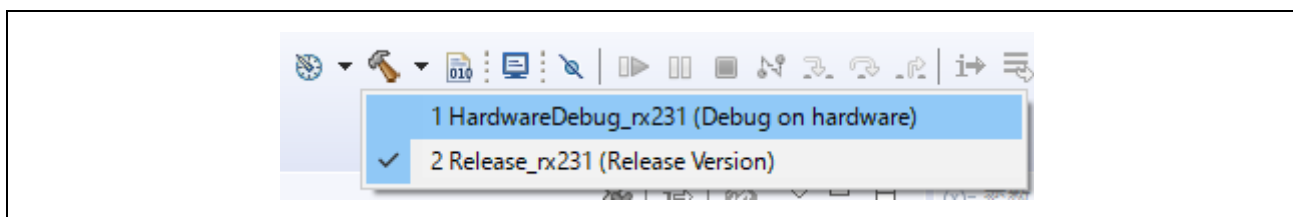
Since the user program project is started via Secure Boot, the reset vector (RESETVECT) of the project is placed at 0xFFFF7FFFC. For this reason, the user program project cannot be debugged as is. To debug the user program project, switch to the debugging configuration shown in Debugging Configuration of Each User Program.

**Table 6-16 Debugging configuration for each user program**

| Board              | User Program                    | Debug Configuration |
|--------------------|---------------------------------|---------------------|
| RSK RX231          | rx231_rsk_tsip_user_program     | HardwareDebug_rx231 |
| MCB RX26T          | rx26t_mcb_tsip_sci_user_program | HardwareDebug_rx26t |
| RSK RX65N          | rx65n_2mb_rsk_tsip_user_program | HardwareDebug_rx65n |
| RSK RX66T          | rx66t_rsk_tsip_sci_user_program | HardwareDebug_rx66t |
| RSK RX671          | rx671_rsk_tsip_user_program     | HardwareDebug_rx671 |
| RSK RX72M          | rx72m_rsk_tsip_user_program     | HardwareDebug_rx72m |
| RSK RX72N          | rx72n_rsk_tsip_user_program     | HardwareDebug_rx72n |
| RX72N Envision Kit | rx72n_ek_tsip_user_program      | HardwareDebug_rx72n |
| RSK RX72T          | rx72t_rsk_tsip_user_program     | HardwareDebug_rx72t |

- How to switch debugger connection configuration:

1. Press ▼ next to e<sup>2</sup>studio menu build button  Select [HardwareDebug\_rx231(Debug on hardware)] to start the build.



**Figure 6.64 e<sup>2</sup>studio project menu(for RX231)**

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

Difference between "Release\_rxXXX" configuration for release and "HardwareDebug\_rxXXX" configuration for debugging

The addresses of the reset vector (RESETVECT) and interrupt vector (EXCEPTVECT) at build time are different between the "Release\_rxXXX" and "HardwareDebug\_rxXXX" configurations for each user program project.

**Table 6-17 User program project Vector setting address for each configuration**

| Symbol     | Release_rxXXX | HardwareDebug_rxXXX |
|------------|---------------|---------------------|
| RESETVECT  | 0xFFFFEFFF8   | 0xFFFFFFFF8         |
| EXCEPTVECT | 0xFFFFEFFF0   | 0xFFFFFFFF0         |

### 6.2.6 Notes on Transition from Secure Bootloader to User Program

When the transition from the secure bootloader program to the user program takes place, the peripheral function settings of the secure bootloader program are inherited by the application. In the bootloader sample program this is implemented as follows.

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

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

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

### 6.2.7 Performance Information for Demo Project

Performance information for each Secure Bootloader demo project is shown below.

Performance is measured in cycles of ICLK, the core clock. The drivers are built using CC-RX with optimization level 2. Other conditions are shown below.

Module revision: r\_tsip\_rx rev1.21

Compiler version: Renesas Electronics C/C++ Compiler Package for RX Family V3.06.00  
(integrated development environment default settings with “-lang = c99” option added)

As a performance information of the demo project, the processing time from reset to MAC verification for each Secure Bootloader demo project included in the package of this application note is shown in Table 6-18.

**Table 6-18 Performance of Secure Bootloader Demo Project (Unit: Cycle)**

| Board              | Performance<br>(Reset ~ MAC Verification) |
|--------------------|-------------------------------------------|
| RSK RX231          | 300,000,000                               |
| MCB RX26T          | 280,000,000                               |
| RSK RX65N          | 280,000,000                               |
| RSK RX66T          | 380,000,000                               |
| RSK RX671          | 280,000,000                               |
| RSK RX72M          | 560,000,000                               |
| RSK RX72N          | 560,000,000                               |
| RX72N Envision Kit | 560,000,000                               |
| RSK RX72T          | 450,000,000                               |

## 7. Appendix

### 7.1 Confirmed Operation Environment

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

**Table 7-1 Confirmed Operation Environment**

| Item                               | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
|------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Integrated development environment | Renesas Electronics e <sup>2</sup> studio 2024-04<br>IAR Embedded Workbench for Renesas RX 5.10.01                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| C compiler                         | Renesas Electronics C/C++ Compiler for RX Family (CC-RX) V3.06.00<br>Compile options:<br>The following option has been added to the default settings of the integrated development environment.<br>-lang = c99                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
|                                    | GCC for Renesas RX 8.3.0.202311<br>Compile options:<br>The following option has been added to the default settings of the integrated development environment.<br>-std = gnu99                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
|                                    | IAR C/C++ Compiler for Renesas RX version 5.10.01<br>Compiler options: Default settings of the integrated development environment                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| Endian order                       | Big-endian or little-endian                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
| Module version                     | Ver. 1.21                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
| Board used                         | Renesas Starter Kit for RX231 (B version) (product No.: R0K505231S020BE)<br>Renesas Solution Starter Kit for RX23W (with TSIP)<br>(product No.: RTK5523W8BC00001BJ)<br>MCB-RX26T Type B (product No.: RTK0EMXE70C02000BJ)<br>Renesas Starter Kit+ for RX65N-2MB (with TSIP)<br>(product No.: RTK50565N2S10010BE)<br>Renesas Starter Kit for RX66T (with TSIP) (product No.: RTK50566T0S00010BE)<br>Renesas Starter Kit+ for RX671 (product No.: RTK55671xxxxxxxxxx)<br>Renesas Starter Kit+ for RX72M (with TSIP) (product No.: RTK5572MNHSxxxxxxxx)<br>Renesas Starter Kit+ for RX72N (with TSIP) (product No.: RTK5572NNHCxxxxxxxx)<br>Renesas Starter Kit for RX72T (with TSIP) (product No.: RTK5572TKCS00010BE) |

## 7.2 Troubleshooting

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

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

- Using CS+  
Application Note: Adding Firmware Integration Technology Modules to CS+ Projects (R01AN1826)
- Using e<sup>2</sup> studio  
Application Note: Adding Firmware Integration Technology Modules to Projects (R01AN1723)

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

(2) Q: I want to use the FIT Demos e<sup>2</sup> studio sample project on CS+.

A: Visit the following webpage for instructions:

Porting from the e<sup>2</sup> studio to CS+

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

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

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

## 7.3 User Key Encryption Formats

For key injection the user key is wrapped using a UFPK and IV, and for key updating the user key is wrapped using a KUK and IV. The format of the key data to be wrapped depends on the cryptographic algorithm. This section lists the data formats for the user key to be encrypted (user key) and for the wrapped key (encrypted user key).

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

### 7.3.1 AES

#### 7.3.1.1 AES 128-Bit Key

Input (User key)

|       |                 |   |   |   |
|-------|-----------------|---|---|---|
| Bytes | 16              |   |   |   |
|       | 4               | 4 | 4 | 4 |
| 0-15  | 128-bit AES key |   |   |   |

Output (Encrypted key)

|       |                                         |   |   |   |
|-------|-----------------------------------------|---|---|---|
| Bytes | 16                                      |   |   |   |
|       | 4                                       | 4 | 4 | 4 |
| 0-15  | encrypted_user_key<br>(128-bit AES key) |   |   |   |
| 16-31 | MAC                                     |   |   |   |

#### 7.3.1.2 AES 256-Bit Key

Input (User key)

|       |                 |   |   |   |
|-------|-----------------|---|---|---|
| Bytes | 16              |   |   |   |
|       | 4               | 4 | 4 | 4 |
| 0-31  | 256-bit AES key |   |   |   |

Output (Encrypted key)

|       |                                         |   |   |   |
|-------|-----------------------------------------|---|---|---|
| Bytes | 16                                      |   |   |   |
|       | 4                                       | 4 | 4 | 4 |
| 0-31  | encrypted_user_key<br>(256-bit AES key) |   |   |   |
| 32-47 | MAC                                     |   |   |   |

### 7.3.2 DES

Input (User key)

|       |                                    |   |   |   |
|-------|------------------------------------|---|---|---|
| Bytes | 16                                 |   |   |   |
|       | 4                                  | 4 | 4 | 4 |
| 0-7   | 56-bit DES key with odd parity 1*1 |   |   |   |
| 8-15  | 56-bit DES key with odd parity 2*1 |   |   |   |
| 16-23 | 56-bit DES key with odd parity 3*1 |   |   |   |

Output (Encrypted key)

|       |                                                                                                                                        |   |   |   |
|-------|----------------------------------------------------------------------------------------------------------------------------------------|---|---|---|
| Bytes | 16                                                                                                                                     |   |   |   |
|       | 4                                                                                                                                      | 4 | 4 | 4 |
| 0-23  | encrypted_user_key<br>(56-bit DES key with odd parity 1   <br>56-bit DES key with odd parity 2   <br>56-bit DES key with odd parity 3) |   |   |   |
| 24-39 | MAC                                                                                                                                    |   |   |   |

Note: 1. Append an odd-parity bit to each 7 bits of key data.

Example: DES key data = 0x0000000000000000 → 0x0101010101010101

DES key data = 0xFFFFFFFFFFFFFFFF → 0xFEFEFEFEFEFEFEFEFE

For 2-DES, insert the same key in the 56-bit DES key with odd parity 1 and the 56-bit DES key with odd parity 3.

For DES, insert the same value in the 56-bit DES key with odd parity 1, the 56-bit DES key with odd parity 2, and the 56-bit DES key with odd parity 3.



### 7.3.3 ARC4

Input (User key)

|       |      |   |   |   |
|-------|------|---|---|---|
| Bytes | 16   |   |   |   |
|       | 4    | 4 | 4 | 4 |
| 0-255 | ARC4 |   |   |   |

Output (Encrypted key)

|         |                           |   |   |   |
|---------|---------------------------|---|---|---|
| Bytes   | 16                        |   |   |   |
|         | 4                         | 4 | 4 | 4 |
| 0-255   | encrypted_user_key (ARC4) |   |   |   |
| 256-272 | MAC                       |   |   |   |

### 7.3.4 RSA

#### 7.3.4.1 RSA 1024-Bit Key

##### (1) Public Key

Input (User key)

|         |                           |           |   |   |
|---------|---------------------------|-----------|---|---|
| Bytes   | 16                        |           |   |   |
|         | 4                         | 4         | 4 | 4 |
| 0-127   | RSA 1024-bit public key n |           |   |   |
| 128-143 | RSA 1024-bit public key e | 0 padding |   |   |

Output (Encrypted key)

|         |                                                                        |   |   |   |
|---------|------------------------------------------------------------------------|---|---|---|
| Bytes   | 16                                                                     |   |   |   |
|         | 4                                                                      | 4 | 4 | 4 |
| 0-143   | encrypted_user_key<br>(RSA 1024-bit public key n    e   <br>0 padding) |   |   |   |
| 144-159 | MAC                                                                    |   |   |   |

##### (2) Secret Key

Input (User key)

|         |                           |   |   |   |
|---------|---------------------------|---|---|---|
| Bytes   | 16                        |   |   |   |
|         | 4                         | 4 | 4 | 4 |
| 0-127   | RSA 1024-bit public key n |   |   |   |
| 128-255 | RSA 1024-bit secret key d |   |   |   |

Output (Encrypted key)

|         |                                                                      |   |   |   |
|---------|----------------------------------------------------------------------|---|---|---|
| Bytes   | 16                                                                   |   |   |   |
|         | 4                                                                    | 4 | 4 | 4 |
| 0-255   | encrypted_user_key<br>(RSA 1024-bit public key n    secret<br>key d) |   |   |   |
| 256-271 | MAC                                                                  |   |   |   |

### 7.3.4.2 RSA 2048-Bit Key

#### (1) Public Key

Input (User key)

|         |                           |           |   |   |
|---------|---------------------------|-----------|---|---|
| Bytes   | 16                        |           |   |   |
|         | 4                         | 4         | 4 | 4 |
| 0-255   | RSA 2048-bit public key n |           |   |   |
| 256-271 | RSA 2048-bit public key e | 0 padding |   |   |

Output (Encrypted key)

|         |                                                                     |   |   |   |
|---------|---------------------------------------------------------------------|---|---|---|
| Bytes   | 16                                                                  |   |   |   |
|         | 4                                                                   | 4 | 4 | 4 |
| 0-271   | encrypted_user_key<br>(RSA 2048-bit public key n    e    0 padding) |   |   |   |
| 272-287 | MAC                                                                 |   |   |   |

#### (2) Secret Key

Input (User key)

|         |                           |   |   |   |
|---------|---------------------------|---|---|---|
| Bytes   | 16                        |   |   |   |
|         | 4                         | 4 | 4 | 4 |
| 0-255   | RSA 2048-bit public key n |   |   |   |
| 256-511 | RSA 2048-bit secret key d |   |   |   |

Output (Encrypted key)

|         |                                                                   |   |   |   |
|---------|-------------------------------------------------------------------|---|---|---|
| Bytes   | 16                                                                |   |   |   |
|         | 4                                                                 | 4 | 4 | 4 |
| 0-511   | encrypted_user_key<br>(RSA 2048-bit public key n    secret key d) |   |   |   |
| 512-527 | MAC                                                               |   |   |   |

### 7.3.4.3 RSA 3072-Bit Key

#### (1) Public Key

Input (User key)

|         |                           |           |   |   |
|---------|---------------------------|-----------|---|---|
| Bytes   | 16                        |           |   |   |
|         | 4                         | 4         | 4 | 4 |
| 0-383   | RSA 3072-bit public key n |           |   |   |
| 384-399 | RSA 3072-bit public key e | 0 padding |   |   |

Output (Encrypted key)

|         |                                                                     |   |   |   |
|---------|---------------------------------------------------------------------|---|---|---|
| Bytes   | 16                                                                  |   |   |   |
|         | 4                                                                   | 4 | 4 | 4 |
| 0-399   | encrypted_user_key<br>(RSA 3072-bit public key n    e    0 padding) |   |   |   |
| 400-415 | MAC                                                                 |   |   |   |

**7.3.4.4 RSA 4096-Bit Key**

**(1) Public Key**

Input (User key)

|         |                           |           |   |   |
|---------|---------------------------|-----------|---|---|
| Bytes   | 16                        |           |   |   |
|         | 4                         | 4         | 4 | 4 |
| 0-511   | RSA 4096-bit public key n |           |   |   |
| 512-527 | RSA 4096-bit public key e | 0 padding |   |   |

Output (Encrypted key)

|         |                                                                     |   |   |   |
|---------|---------------------------------------------------------------------|---|---|---|
| Bytes   | 16                                                                  |   |   |   |
|         | 4                                                                   | 4 | 4 | 4 |
| 0-527   | encrypted_user_key<br>(RSA 4096-bit public key n    e    0 padding) |   |   |   |
| 528-543 | MAC                                                                 |   |   |   |

**7.3.5 ECC**

**7.3.5.1 ECC P192**

**(1) Public Key**

Input (User key)

|       |           |   |   |                             |
|-------|-----------|---|---|-----------------------------|
| Bytes | 16        |   |   |                             |
|       | 4         | 4 | 4 | 4                           |
| 0-31  | 0 padding |   |   | ECC P 192-bit public key Qx |
|       |           |   |   |                             |
| 32-63 | 0 padding |   |   | ECC P 192-bit public key Qy |
|       |           |   |   |                             |

Output (Encrypted key)

|       |                                                                                                              |   |   |   |
|-------|--------------------------------------------------------------------------------------------------------------|---|---|---|
| Bytes | 16                                                                                                           |   |   |   |
|       | 4                                                                                                            | 4 | 4 | 4 |
| 0-63  | encrypted_user_key<br>(0 padding    ECC P 192-bit public key Qx    0 padding    ECC P 192-bit public key Qy) |   |   |   |
| 64-79 | MAC                                                                                                          |   |   |   |

**(2) Secret Key**

Input (User key)

|       |           |   |   |                            |
|-------|-----------|---|---|----------------------------|
| Bytes | 16        |   |   |                            |
|       | 4         | 4 | 4 | 4                          |
| 0-31  | 0 padding |   |   | ECC P 192-bit secret key d |
|       |           |   |   |                            |

Output (Encrypted key)

|       |                                                                 |   |   |   |
|-------|-----------------------------------------------------------------|---|---|---|
| Bytes | 16                                                              |   |   |   |
|       | 4                                                               | 4 | 4 | 4 |
| 0-31  | encrypted_user_key<br>(0 padding    ECC P 192-bit secret key d) |   |   |   |
| 32-47 | MAC                                                             |   |   |   |

**7.3.5.2 ECC P224**

**(1) Public Key**

Input (User key)

|       |                             |                             |   |   |
|-------|-----------------------------|-----------------------------|---|---|
| Bytes | 16                          |                             |   |   |
|       | 4                           | 4                           | 4 | 4 |
| 0-31  | 0 padding                   | ECC P 224-bit public key Qx |   |   |
|       | ECC P 224-bit public key Qx |                             |   |   |
| 32-63 | 0 padding                   | ECC P 224-bit public key Qy |   |   |
|       | ECC P 224-bit public key Qy |                             |   |   |

Output (Encrypted key)

|       |                                                                                                              |   |   |   |
|-------|--------------------------------------------------------------------------------------------------------------|---|---|---|
| Bytes | 16                                                                                                           |   |   |   |
|       | 4                                                                                                            | 4 | 4 | 4 |
| 0-63  | encrypted_user_key<br>(0 padding    ECC P 224-bit public key Qx    0 padding    ECC P 224-bit public key Qy) |   |   |   |
|       | encrypted_user_key<br>(0 padding    ECC P 224-bit public key Qx    0 padding    ECC P 224-bit public key Qy) |   |   |   |
| 64-79 | MAC                                                                                                          |   |   |   |

**(2) Secret Key**

Input (User key)

|       |                            |                            |   |   |
|-------|----------------------------|----------------------------|---|---|
| Bytes | 16                         |                            |   |   |
|       | 4                          | 4                          | 4 | 4 |
| 0-31  | 0 padding                  | ECC P 224-bit secret key d |   |   |
|       | ECC P 224-bit secret key d |                            |   |   |

Output (Encrypted key)

|       |                                                                 |   |   |   |
|-------|-----------------------------------------------------------------|---|---|---|
| Bytes | 16                                                              |   |   |   |
|       | 4                                                               | 4 | 4 | 4 |
| 0-31  | encrypted_user_key<br>(0 padding    ECC P 224-bit secret key d) |   |   |   |
|       | encrypted_user_key<br>(0 padding    ECC P 224-bit secret key d) |   |   |   |
| 32-47 | MAC                                                             |   |   |   |

**7.3.5.3 ECC P256**

**(1) Public Key**

Input (User key)

|       |                             |   |   |   |
|-------|-----------------------------|---|---|---|
| Bytes | 16                          |   |   |   |
|       | 4                           | 4 | 4 | 4 |
| 0-31  | ECC P 256-bit public key Qx |   |   |   |
| 32-63 | ECC P 256-bit public key Qy |   |   |   |

Output (Encrypted key)

|       |                                                                                    |   |   |   |
|-------|------------------------------------------------------------------------------------|---|---|---|
| Bytes | 16                                                                                 |   |   |   |
|       | 4                                                                                  | 4 | 4 | 4 |
| 0-63  | encrypted_user_key<br>(ECC P 256-bit public key Qx    ECC P 256-bit public key Qy) |   |   |   |
|       | encrypted_user_key<br>(ECC P 256-bit public key Qx    ECC P 256-bit public key Qy) |   |   |   |
| 64-79 | MAC                                                                                |   |   |   |

**(2) Secret Key**

Input (User key)

|       |                            |   |   |   |
|-------|----------------------------|---|---|---|
| Bytes | 16                         |   |   |   |
|       | 4                          | 4 | 4 | 4 |
| 0-31  | ECC P 256-bit secret key d |   |   |   |

Output (Encrypted key)

|       |                                                    |   |   |   |
|-------|----------------------------------------------------|---|---|---|
| Bytes | 16                                                 |   |   |   |
|       | 4                                                  | 4 | 4 | 4 |
| 0-31  | encrypted_user_key<br>(ECC P 256-bit secret key d) |   |   |   |
|       | encrypted_user_key<br>(ECC P 256-bit secret key d) |   |   |   |
| 32-47 | MAC                                                |   |   |   |

**7.3.5.4 ECC P384**

**(1) Public Key**

Input (User key)

|       |                             |   |   |   |
|-------|-----------------------------|---|---|---|
| Bytes | 16                          |   |   |   |
|       | 4                           | 4 | 4 | 4 |
| 0-47  | ECC P 384-bit public key Qx |   |   |   |
| 48-95 | ECC P 384-bit public key Qy |   |   |   |

Output (Encrypted key)

|        |                                                                                       |   |   |   |
|--------|---------------------------------------------------------------------------------------|---|---|---|
| Bytes  | 16                                                                                    |   |   |   |
|        | 4                                                                                     | 4 | 4 | 4 |
| 0-95   | encrypted_user_key<br>(ECC P 384-bit public key Qx   <br>ECC P 384-bit public key Qy) |   |   |   |
| 96-111 | MAC                                                                                   |   |   |   |

**(2) Secret Key**

Input (User key)

|       |                            |   |   |   |
|-------|----------------------------|---|---|---|
| Bytes | 16                         |   |   |   |
|       | 4                          | 4 | 4 | 4 |
| 0-47  | ECC P 384-bit secret key d |   |   |   |

Output (Encrypted key)

|       |                                                    |   |   |   |
|-------|----------------------------------------------------|---|---|---|
| Bytes | 16                                                 |   |   |   |
|       | 4                                                  | 4 | 4 | 4 |
| 0-47  | encrypted_user_key<br>(ECC P 384-bit secret key d) |   |   |   |
| 48-63 | MAC                                                |   |   |   |

**7.3.6 HMAC**

**7.3.6.1 SHA1-HMAC Key**

Input (User key)

|       |               |   |   |   |
|-------|---------------|---|---|---|
| Bytes | 16            |   |   |   |
|       | 4             | 4 | 4 | 4 |
| 0-31  | HMAC-SHA1 key |   |   |   |
|       | 0 padding     |   |   |   |

Output (Encrypted key)

|       |                                                |   |   |   |
|-------|------------------------------------------------|---|---|---|
| Bytes | 16                                             |   |   |   |
|       | 4                                              | 4 | 4 | 4 |
| 0-31  | encrypted_user_key<br>(HMAC-SHA1    0 padding) |   |   |   |
| 32-47 | MAC                                            |   |   |   |

**7.3.6.2 SHA256-HMAC Key**

Input (User key)

|       |                 |   |   |   |
|-------|-----------------|---|---|---|
| Bytes | 16              |   |   |   |
|       | 4               | 4 | 4 | 4 |
| 0-31  | HMAC-SHA256 key |   |   |   |

Output (Encrypted key)

|       |                                     |   |   |   |
|-------|-------------------------------------|---|---|---|
| Bytes | 16                                  |   |   |   |
|       | 4                                   | 4 | 4 | 4 |
| 0-31  | encrypted_user_key<br>(HMAC-SHA256) |   |   |   |
| 32-47 | MAC                                 |   |   |   |

**7.3.7 KUK**

Input (User key)

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

Output (Encrypted key)

|       |                                                              |   |   |   |
|-------|--------------------------------------------------------------|---|---|---|
| Bytes | 16                                                           |   |   |   |
|       | 4                                                            | 4 | 4 | 4 |
| 0-31  | encrypted_user_key<br>(AES 128-bit CBC key    CBCMAC<br>key) |   |   |   |
| 32-47 | MAC                                                          |   |   |   |

## 7.4 Wrapped Key of Public Key Formats for Asymmetric Cryptography

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

### 7.4.1 RSA

The key index structure `tsip_rsaXXXX_public_key_index_t` of the RSA public key contains the plaintext data of the public key in members `value.key_n` and `value_e`.

The modulus and exponent values are output in big-endian byte ordering as `key_n` and `key_e`, respectively.

### 7.4.2 ECC

The member `value.key_q` of the wrapped key structure `tsip_ecc_public_key_index_t` of the ECC public key contains the plaintext data of the public key. The format of `key_q` is as shown below.

#### 7.4.2.1 ECC P 192-Bit Key

|       |                                            |         |                         |         |
|-------|--------------------------------------------|---------|-------------------------|---------|
| Bytes | 128 bits                                   |         |                         |         |
|       | 32 bits                                    | 32 bits | 32 bits                 | 32 bits |
| 0-15  | 0 padding                                  |         | ECC P-192 public key Qx |         |
| 16-31 | ECC P 192-bit public key Qx (continuation) |         |                         |         |
| 32-47 | 0 padding                                  |         | ECC P-192 public key Qy |         |
| 48-63 | ECC P 192-bit public key Qy (continuation) |         |                         |         |
| 64-79 | Wrapped key management information         |         |                         |         |

#### 7.4.2.2 ECC P 224-Bit Key

|       |                                            |         |                         |         |
|-------|--------------------------------------------|---------|-------------------------|---------|
| Bytes | 128 bits                                   |         |                         |         |
|       | 32 bits                                    | 32 bits | 32 bits                 | 32 bits |
| 0-15  | 0 padding                                  |         | ECC P-224 public key Qx |         |
| 16-31 | ECC P 224-bit public key Qx (continuation) |         |                         |         |
| 32-47 | 0 padding                                  |         | ECC P-224 public key Qy |         |
| 48-63 | ECC P 224-bit public key Qy (continuation) |         |                         |         |
| 64-79 | Wrapped key management information         |         |                         |         |

#### 7.4.2.3 ECC P 256-Bit Key

|       |                                    |         |         |         |
|-------|------------------------------------|---------|---------|---------|
| Bytes | 128 bits                           |         |         |         |
|       | 32 bits                            | 32 bits | 32 bits | 32 bits |
| 0-31  | ECC P-256-bit public key Qx        |         |         |         |
| 32-63 | ECC P 256-bit public key Qy        |         |         |         |
| 64-79 | Wrapped key management information |         |         |         |

#### 7.4.2.4 ECC P 384-Bit Key

|        |                                    |         |         |         |
|--------|------------------------------------|---------|---------|---------|
| Bytes  | 128 bits                           |         |         |         |
|        | 32 bits                            | 32 bits | 32 bits | 32 bits |
| 0-47   | ECC P 384-bit public key Qx        |         |         |         |
| 48-95  | ECC P 384-bit public key Qy        |         |         |         |
| 96-111 | Wrapped key management information |         |         |         |

## 7.5 Using Renesas Secure Flash Programmer

### 7.5.1 Usage Notes

Renesas Secure Flash Programmer is a tool included with RX TSIP FIT that can generate Encrypted Keys and encrypted user programs (Note). When using Renesas Secure Flash Programmer, please read the following table as a replacement.

Note : Development of Renesas Secure Flash Programmer was completed with V.1.19.

**Table 7-2 Terminology table Security Key Management Tool and Renesas Secure Flash Programmer**

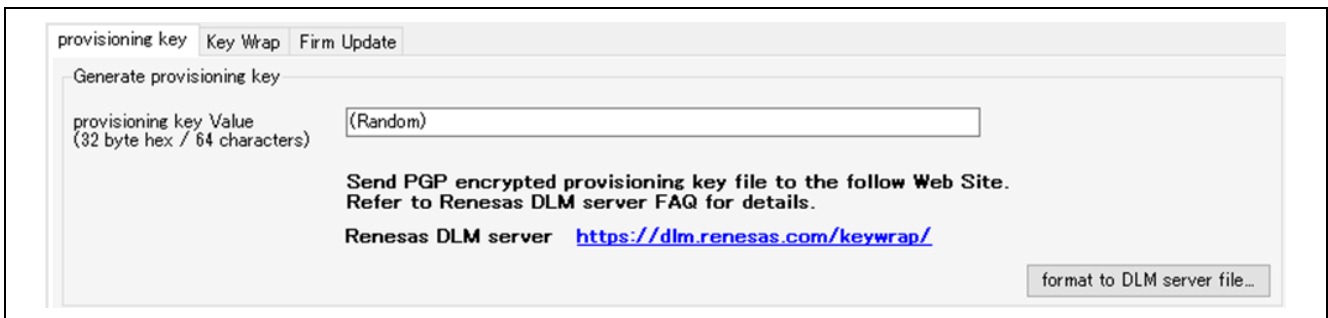
| Security Key Management Tool                 | Renesas Secure Flash Programmer |
|----------------------------------------------|---------------------------------|
| UFPK(User Factory Programming Key)           | Provisioning Key                |
| W-UFPK(Wrapped User Factory Programming Key) | Encrypted Provisioning Key      |
| KUK(Key Update Key)                          | Update Key Ring                 |
| Encrypted Key                                | Encrypted Key                   |
| Wrapped Key                                  | Key Index                       |
| Key Encryption Key                           | Prog User Key, User Program Key |

### 7.5.2 provisioning key Tab

Figure 7.1 shows the provisioning key tab of Renesas Secure Flash Programmer. The provisioning key tab is used to create a provisioning key file to be sent to the DLM server.

Enter a provisioning key value in hexadecimal 32-byte format for **provisioning key Value** and click the **format to DLM server file...** button.

It is possible to generate a provisioning key file from a random number by pressing the **format to DLM server file...** button when **(Random)** is displayed, but the result cannot be used in actual products because the random number used lacks sufficient precision.



**Figure 7.1 provisioning key Tab of Renesas Secure Flash Programmer**

### 7.5.3 Key Wrap Tab

Figure 7.2 shows the Key Wrap tab of Renesas Secure Flash Programmer. Table 7-3 contains descriptions of the setting values that can be configured on the Key Wrap tab. Based on the descriptions in Table 7-3, enter appropriate setting values and click the **Generate Key Files...** button to generate encrypted key files (key\_data.c and key\_data.h). Refer to Table 7-4 for descriptions of the buttons.

provisioning key | Key Wrap | Firm Update

MCU  
Select MCU (select MCU) v

Key Setting  
Key Type AES-128bit v Key Data Register  
Key Type Key Data Delete

provisioning key  
provisioning key File Path Browse...  
encrypted provisioning key File Path Browse...  
IV (16 byte hex / 32 characters) (Random)  
Generate Key Files...

Figure 7.2 Key Wrap Tab of Renesas Secure Flash Programmer



Table 7-3 Description of Key Wrap Tab Setting Values

| Parameter                            | Setting Value                                                                                                                                                                                                                                                                                                                                                                 | Description                                                                                                                                                                                                                    |
|--------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Select MCU                           | TSIP-Lite (DF Memory 8KB)<br>TSIP-Lite (DF Memory 16KB)<br>TSIP-Lite (DF Memory 32KB)<br>TSIP (DF Memory 8KB)<br>TSIP (DF Memory 32KB)                                                                                                                                                                                                                                        | Selects the MCU (TSIP type and data flash memory size) to be used.                                                                                                                                                             |
| Key Type                             | AES-128bit<br>AES-256bit<br>DES<br>2Key-TDES<br>Triple-DES<br>ARC4-2048bit<br>SHA1-HMAC<br>SHA256-HMAC<br>RSA-1024bit Public/Private/All<br>RSA-2048bit Public/Private/All<br>RSA-3072bit Public<br>RSA-4096bit Public<br>ECC-192bit Public/Private/All<br>ECC-224bit Public/Private/All<br>ECC-256bit Public/Private/All<br>ECC-384bit Public/Private/All<br>Update Key Ring | Specifies the type of user key to be generated.                                                                                                                                                                                |
| Key Data                             | User key<br>Refer to 7.5.3.1, Key Data Input Formats, for the key data formats that can be entered.                                                                                                                                                                                                                                                                           | Enter the user key data to be generated.                                                                                                                                                                                       |
| provisioning key File Path           | File path of provisioning key file                                                                                                                                                                                                                                                                                                                                            | Specify the file path of the plaintext provisioning key file to be used when encrypting the user key. Refer to 7.5.2, provisioning key Tab, for instructions for generating a provisioning key file.                           |
| encrypted provisioning key File Path | File path of wrapped provisioning key file                                                                                                                                                                                                                                                                                                                                    | Specify the file path to which the wrapped C language provisioning key file will be output. Refer to 7.5.2, provisioning key Tab, for instructions for generating a wrapped provisioning key file.                             |
| IV (16 byte hex / 32 characters)     | IV value                                                                                                                                                                                                                                                                                                                                                                      | Enter a 16-byte IV value. If the <b>Generate Key File...</b> button is pressed with the input value left as <b>(Random)</b> , a random number generated internally by Renesas Secure Flash Programmer is used as the IV value. |
| Generate Key Files...                | Button for generating C language files                                                                                                                                                                                                                                                                                                                                        | Outputs C language encrypted key files.                                                                                                                                                                                        |

**Table 7-4 Description of Key Wrap Tab Buttons**

| Button                | Description                                                                                                                                                                                                                                                                            |
|-----------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Register              | Registers the user key data specified in the <b>Key Data</b> field. For <b>Key Data</b> , enter user key data that matches <b>Key Type</b> , then click this button to register it.                                                                                                    |
| Delete                | Deletes the registered user key data. Make sure the entry field containing the user key data is selected, then click this button to delete it.                                                                                                                                         |
| Browse...             | Click these buttons to specify the file paths of the provisioning key file and wrapped provisioning key file, respectively, from the File Explorer interface. You can also enter file paths directly.                                                                                  |
| Generate Key Files... | Generates encrypted key files (key_data.c and key_data.h). Enter appropriate values into the various fields, then click this button. When the button is clicked, a window appears for specifying the folder to which the encrypted key files will be output, and the files are output. |

### 7.5.3.1 Key Data Input Formats

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

(1) AES 128-bit data format

|       |                  |
|-------|------------------|
| Bytes | 128 bits         |
| 0-15  | AES 128 key data |

(2) AES 256-bit data format

|       |                  |
|-------|------------------|
| Bytes | 256 bits         |
| 0-31  | AES 256 key data |

(3) TDES data format

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

(4) 2Key-TDES data format

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

(5) DES data format

|       |                |
|-------|----------------|
| Bytes | DES user key 1 |
| 0-7   | DES key data   |

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

DES key data format is shown below.

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

Example: When the parity bit is added, the DES user key 0x0000000000000000 becomes 0x0101010101010101, 0xFFFFFFFFFFFFFFFF becomes 0xFEFEFEFEFEFEFEFEFE, 0x01020304050607 becomes 0x018080614029190E.

(6) ARC4 data format

|       |               |
|-------|---------------|
| Bytes | 2048 bits     |
| 0-255 | ARC4 key data |

## (7) SHA1-HMAC data format

|       |                    |
|-------|--------------------|
| Bytes | 160 bits           |
| 0-19  | SHA1-HMAC key data |

## (8) SHA256-HMAC data format

|       |                      |
|-------|----------------------|
| Bytes | 256 bits             |
| 0-31  | SHA256-HMAC key data |

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

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

Note: Public key

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

## (26) ECC 384-bit public data format (96 bytes)

| Bytes | ECC 384-bit public key Qx      | ECC 384-bit public key Qy      |
|-------|--------------------------------|--------------------------------|
| 0-95  | 48-byte ECC public key Qx data | 48-byte ECC public key Qy data |

## (27) ECC 384-bit private data format (48 bytes)

| Bytes | ECC 384-bit private key     |
|-------|-----------------------------|
| 0-47  | 48-byte ECC secret key data |

## (28) ECC 384-bit all data format (144 bytes)

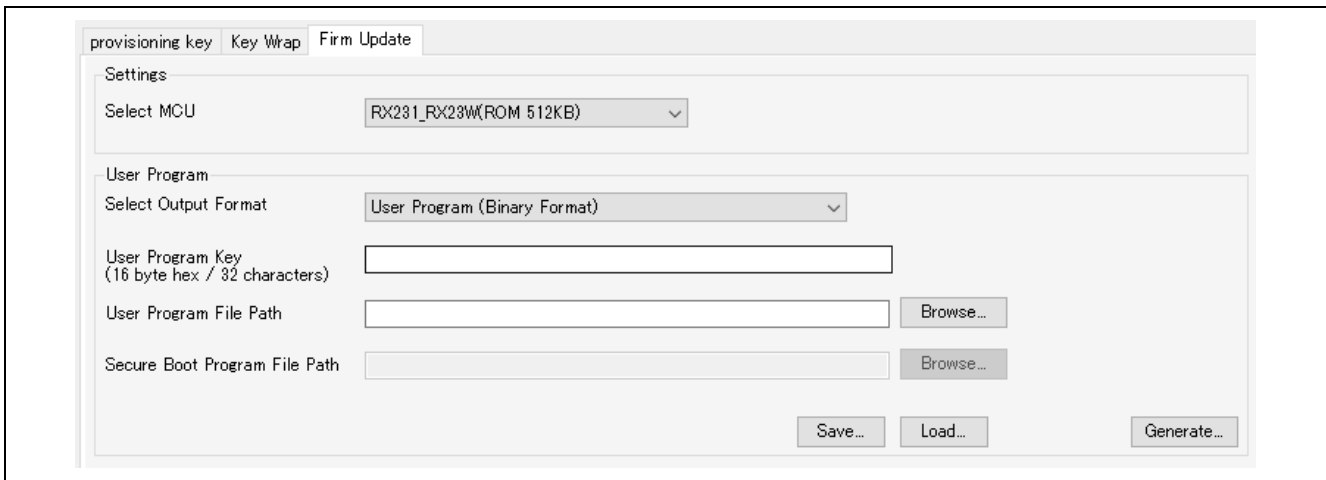
| Bytes | ECC 384-bit public key Qx      | ECC 384-bit public key Qy      | ECC 384-bit private key     |
|-------|--------------------------------|--------------------------------|-----------------------------|
| 0-143 | 48-byte ECC public key Qx data | 48-byte ECC public key Qy data | 48-byte ECC secret key data |

## (29) Update keyring data format

| Bytes | 256 bits            |
|-------|---------------------|
| 0-31  | Update keyring data |

### 7.5.4 Firm Update Tab

Figure 7.3 shows the Firm Update tab of Renesas Secure Flash Programmer, and Table 7-5 contains descriptions of the setting values that can be configured on the Firm Update tab. Based on the descriptions in Table 7-5, enter appropriate setting values and click the **Generate...** button to generate an encrypted user program file (extension: \*.rsu or \*.mot). Refer to Table 7-6 for descriptions of the buttons.



**Figure 7.3 Firm Update Tab of Renesas Secure Flash Programmer**

**Table 7-5 Description of Firm Update Tab Setting Values**

| Parameter                     | Setting Value                                                                                                                                                                                                                                                                                                                   | Description                                                                                                                                                                              |
|-------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Select MCU                    | RX231_RX23W (ROM 512KB)<br>RX231_RX23W (ROM 384KB)<br>RX231 with WiFi (ROM 512KB)<br>RX66T_RX72T (ROM 1MB)<br>RX66T_RX72T (ROM 512KB)<br>RX66T (ROM 256KB)<br>RX26T (ROM512KB)<br>RX65N_RX651_RX671 (ROM 2MB)<br>RX65N_RX651_RX671 (ROM 1.5MB)<br>RX671 (ROM 1MB)<br>RX72M_RX72N_RX66N (ROM 4MB)<br>RX72M_RX72N_RX66N (ROM 2MB) | Select the option that matches the group name and ROM size of the device to be used. Refer to 7.5.4.1, Select MCU, for details.                                                          |
| Select Output Format          | User Program (Binary Format),<br>User Program + Secure Boot Program(Motorola S Format)                                                                                                                                                                                                                                          | Select the option that matches the file format to output. Refer to 7.5.4.2, Select Output Format, for details.                                                                           |
| User Program Key              | AES-128 key ASCII data                                                                                                                                                                                                                                                                                                          | Enter the AES 128-bit key to be injected for the firmware update. Refer to 7.5.4.3, User Program Key, for details.                                                                       |
| User Program File Path        | File path of Motorola hex file containing program to be encrypted and to which an MAC will be added                                                                                                                                                                                                                             | Specify the file path of the Motorola hex file to be encrypted and to which an MAC will be added. Refer to 7.5.4.4, User Program File Path, for details.                                 |
| Secure Boot Program File Path | File path of Motorola hex file containing secure boot program                                                                                                                                                                                                                                                                   | Specify the file path of the Motorola hex file containing secure boot program to combine with the program to be encrypted. Refer to 7.5.4.5, Secure Boot Program File Path, for details. |

**Table 7-6 Description of Firm Update Tab Buttons**

| Button      | Description                                                                                                                                                                                                                                                                                                      |
|-------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Browse...   | Click this button to specify, from the File Explorer interface, the file path of the user program to be encrypted. You can also enter the file path directly.                                                                                                                                                    |
| Generate... | Click this button to generate an encrypted user program file (extension: rsu). Enter appropriate values into the various fields, then click this button. When the button is clicked, a window appears for specifying the folder to which the encrypted user program file will be output, and the file is output. |
| Save...     | Save Renesas Secure Flash Programmer configuration information in a *.ini file.                                                                                                                                                                                                                                  |
| Load...     | The "Save..." button loads the saved configuration information of Renesas Secure Flash Programmer.                                                                                                                                                                                                               |

### 7.5.4.1 Select MCU

Select according to the MCU family used.

MAIN\_AREA, BUFFER\_AREA, and SECURE\_BOOT areas in the Secure Boot and Firm Update sample are set to the following areas.

**Table 7-7 Area address of Select MCU**

| Setting value                    | SECURE_BOOT                | MAIN_AREA                    | BUFFER_AREA                  |
|----------------------------------|----------------------------|------------------------------|------------------------------|
| RX231_RX23W<br>(ROM 512KB)       | 0xFFFF0000 –<br>0xFFFFFFFF | 0xFFFFB8000 -<br>0xFFFFEFFFF | 0xFFFF80000 -<br>0xFFFFB7FFF |
| RX231_RX23W<br>(ROM 384KB)       |                            | 0xFFFFC8000 -<br>0xFFFFEFFFF | 0xFFFFA0000 -<br>0xFFFFC7FFF |
| RX66T_RX72T<br>(ROM 1MB)         |                            | 0xFFFF78000 –<br>0xFFFFEFFFF | 0xFFFF00000 -<br>0xFFFF77FFF |
| RX66T_RX72T<br>(ROM 512KB)       |                            | 0xFFFFB8000 -<br>0xFFFFEFFFF | 0xFFFF80000 -<br>0xFFFFB7FFF |
| RX66T<br>(ROM 256KB)             |                            | 0xFFFFD8000 -<br>0xFFFFEFFFF | 0xFFFFC0000 -<br>0xFFFFD7FFF |
| RX26T<br>(ROM 512KB)             |                            | 0xFFFFC0000 -<br>0xFFFFEFFFF | 0xFFFF80000 -<br>0xFFFFAFFFF |
| RX65N_RX651_RX671<br>(ROM 2MB)   |                            | 0xFFFF00000 –<br>0xFFFFEFFFF | 0xFFFE00000 -<br>0xFFFEFFFF  |
| RX65N_RX651_RX671<br>(ROM 1.5MB) |                            | 0xFFFF40000 –<br>0xFFFFEFFFF | 0xFFFE40000 -<br>0xFFFEFFFF  |
| RX671<br>(ROM 1MB)               |                            | 0xFFFF80000 –<br>0xFFFFEFFFF | 0xFFFE80000 -<br>0xFFFEFFFF  |
| RX72M_RX72N_RX66N<br>(ROM 4MB)   |                            | 0xFFFE00000 –<br>0xFFFFEFFFF | 0xFFFC00000 -<br>0xFFDEFFFF  |
| RX72M_RX72N_RX66N<br>(ROM 2MB)   |                            | 0xFFFF00000 –<br>0xFFFFEFFFF | 0xFFFD00000 -<br>0xFFDEFFFF  |

### 7.5.4.2 Select Output Format

Select Output Format allows you to select the format of the file to be generated.

**Table 7-8 RX Firmware Update FIT for TSIP V.1.0x RSU Format**

| offset    | Component          | Contents name                                        | Length          | Note                                                                                                                |
|-----------|--------------------|------------------------------------------------------|-----------------|---------------------------------------------------------------------------------------------------------------------|
| 0x0000    | Header             | Magic Code                                           | 7               | Renesas                                                                                                             |
| 0x0007    |                    | Image Flags                                          | 1               | User Program (Binary Format) : 0xFE(TESTING)<br>User Program + Secure Boot Program(Motorola S Format) : 0xf8(VALID) |
| 0x0008    | Signature          | Firmware Verification Type                           | 32              | ASCII "mac-aes128-cmac-with-tsip"                                                                                   |
| 0x0028    |                    | Signature size                                       | 4               | Not use. (0x00)                                                                                                     |
| 0x002C    |                    | Signature                                            | 256             | Not use. (0x00)                                                                                                     |
| 0x012C    | Option             | Dataflash Flag                                       | 4               | Not use. (0x00)                                                                                                     |
| 0x0130    |                    | Dataflash Start Address                              | 4               | Not use. (0x00)                                                                                                     |
| 0x0134    |                    | Dataflash End Address                                | 4               | Not use. (0x00)                                                                                                     |
| 0x0138    |                    | Image Size                                           | 4               | Not use. (0x00)                                                                                                     |
| 0x013C    |                    | Reserved                                             | 123             | All 0x00                                                                                                            |
| 0x01B7    |                    | Decrypt Flag <sup>*1</sup>                           | 1               | 0x00 : Decryption not performed                                                                                     |
| 0x01B8    |                    | Format Type <sup>*1</sup>                            | 4               | Output file format.<br>ASCII "rsu"(0x72, 0x73, 0x75, 0x00) or<br>"mot"(0x6D, 0x6F, 0x74, 0x00)                      |
| 0x01BC    |                    | IV <sup>*1</sup>                                     | 16              | IV used during user program encryption. <sup>*2</sup>                                                               |
| 0x01CC    |                    | SessionKey0 <sup>*1</sup>                            | 16              | Key used during user program encryption encrypted with prog user key <sup>*2</sup>                                  |
| 0x01DC    |                    | SessionKey1 <sup>*1</sup>                            | 16              | Key used during user program encryption encrypted with prog user key <sup>*2</sup>                                  |
| 0x01EC    |                    | Encrypted user program + MAC word size <sup>*1</sup> | 4               | Total word size of encrypted user program + MAC generated during user program encryption                            |
| 0x01F0    |                    | MAC <sup>*1</sup>                                    | 16              | MAC generated during user program encryption. <sup>*2</sup>                                                         |
| 0x0200    |                    | Descriptor                                           | Sequence Number | 4                                                                                                                   |
| 0x0204    | Start Address      |                                                      | 4               | Start address of user program area.                                                                                 |
| 0x0208    | End Address        |                                                      | 4               | End address of user program area.                                                                                   |
| 0x020C    | Execution Address  |                                                      | 4               | Address for storing user program execution start address.                                                           |
| 0x0210    | Hardware ID        |                                                      | 4               | Not use(0x00)                                                                                                       |
| 0x0214    | Reserved(0x00)     |                                                      | 236             | -                                                                                                                   |
| 0x0300    | Application Binary |                                                      | N               | Encrypted user program.                                                                                             |
| 0x0300 +N | Dataflash Binary   |                                                      | M               | Not supported                                                                                                       |

Note: 1. Parameters changed for TSIP from \*.rsu file format defined in RX Firmware Update FIT V.1.0x.  
2. Refer to Figure 3.47 Firmware and Session Key Encryption Method.

**(1) Select User Program (Binary Format) :**

User Program Only the data in MAIN\_AREA of the input file is encrypted and a binary file with RSU header is output. Table 5 6 RX Firmware Update FIT for TSIP V.1.0x RSU Format.

**(2) Select User Program + Secure Boot Program(Motorola S Format) :**

Encrypts only the data in the MAIN\_AREA of the User Program input file, adds an RSU header to the first 0 to 0x300 of the main area, and then encrypts the data in the \*.rsu file format defined in the RX Firmware Update FIT V.1.0x specified in the "Secure Boot Program File Path". The \*.rsu file format defined in RX Firmware Update FIT V.1.0x is placed in the UPDATE\_TEMPORARY\_AREA area in the Motorola S format file, with the "Secure Boot Program The data in the OFS area and the Data Flash area is the data of the file specified in "Secure Boot Program File Path". The data in the OFS area and Data Flash area uses the data in the file specified in "Secure Boot Program File Path".

Motorola S format files containing both secure boot and encrypted user programs can be decrypted if the Motorola S format file itself is stolen by writing to and executing the corresponding RX TSIP-equipped device. When writing an encrypted user program that contains a secure boot, please write it in a secure factory.

**7.5.4.3 User Program Key**

Specify the User Prog Key to encrypt the Tmp Key indicated by Figure 3.47 Firmware and Session Key Encryption Method.

**7.5.4.4 User Program File Path**

Specify the user program to be encrypted.

**7.5.4.5 Secure Boot Program File Path**

When "User Program + Secure Boot Program (Motorola S Format)" is selected in Select Output Format, specify the Motorola S format file of the secure boot program to be combined with the encrypted user program.

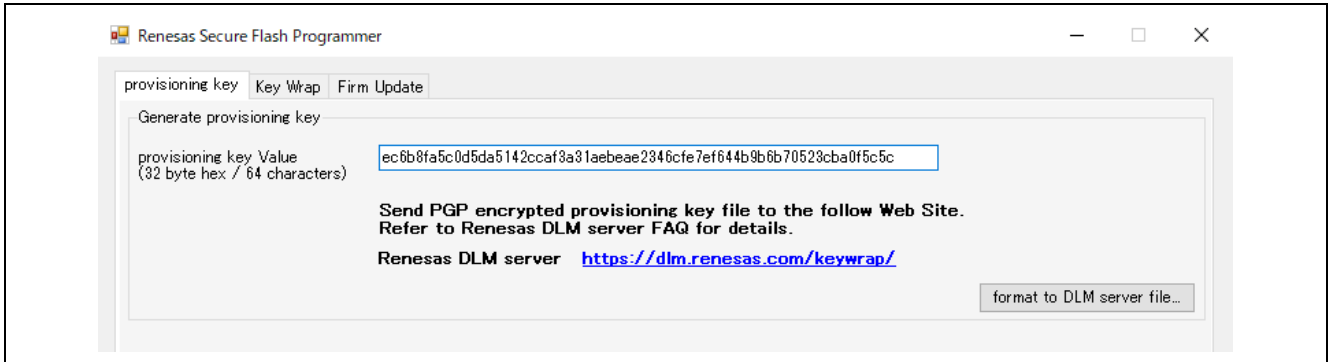


### 7.5.5 Key Injection Procedure

The procedure for generating a key file used for key injection is described here. The example below shows the procedure for generating an AES Motorola hex file.

#### 1. Generating a provisioning key

On the provisioning key tab, specify a provisioning key value and generate a provisioning key file having a file name with the extension \*.key. The file name used in the example below is sample.key.



**Figure 7.4 Generating a Provisioning Key by Specifying a Value on the provisioning key Tab**

Click the **format to DLM server file...** button to generate a provisioning key file.

#### 2. Obtaining an encrypted provisioning key

Send the sample.key file generated in step 1 to the Renesas Key Wrap service (<https://dlm.renesas.com/keywrap>) to obtain an encrypted provisioning key.

For detailed information, refer to the Renesas Key Wrap Service FAQ or the relevant application note.

### 3. Generating AES 128 key files

On the Key Wrap tab, generate AES 128 key files.

Choose the appropriate option for **Select MCU** based on whether the device to be used is provided with a TSIP or TSIP-Lite module and according to the data flash size.

Select the key type for **Key Type** and enter key data in the **Key Data** field, then click the **Register** button.

For **provisioning key File Path**, enter the path of the provisioning key file generated in step 1. For **encrypted provisioning key File Path**, enter the path of the encrypted provisioning key file obtained as described in step 2. In this example, **(Random)** is selected for **IV** in the interest of simplicity.

After configuring all the settings, click the **Generate Key Files...** button.

You can output the key data files (key\_data.c and key\_data.h) to a folder of your choice.

In this example, the settings are configured for an AES 128-bit key only, but it is possible to encrypt multiple keys in a single set of key files by configuring **Key Type** and **Key Data** settings, and clicking the **Register** button, for additional keys after the first one.

The screenshot shows the 'Key Wrap' tab in a software interface. At the top, there are three tabs: 'provisioning key', 'Key Wrap', and 'Firm Update'. The 'Key Wrap' tab is active. Below the tabs, there is a section for 'MCU' with a dropdown menu set to 'TSIP(DF Memory 32KB)'. Underneath is the 'Key Setting' section, which includes a 'Key Type' dropdown set to 'AES-128bit' and a 'Key Data' text field containing the hexadecimal string '000102030405060708090a0b0c0d0e0f'. To the right of the 'Key Data' field are 'Register' and 'Delete' buttons. Below this is a table with two columns: 'Key Type' and 'Key Data'. The table contains one entry: 'AES-128bit' and '000102030405060708090a0b0c0d0e0f'. Below the table is a horizontal scrollbar. At the bottom of the 'Key Setting' section is a 'Generate Key Files...' button. Below the 'Key Setting' section is the 'provisioning key' section, which includes three text fields: 'provisioning key File Path' (containing 'C:\work\samplekey'), 'encrypted provisioning key File Path' (containing 'C:\work\samplekey\_enckey'), and 'IV (16 byte hex / 32 characters)' (containing '(Random)'). Each of the first two fields has a 'Browse...' button to its right. At the bottom right of the entire interface is a 'Generate Key Files...' button.

**Figure 7.5 Example AES 128 Key File Output Settings on Key Wrap Tab**

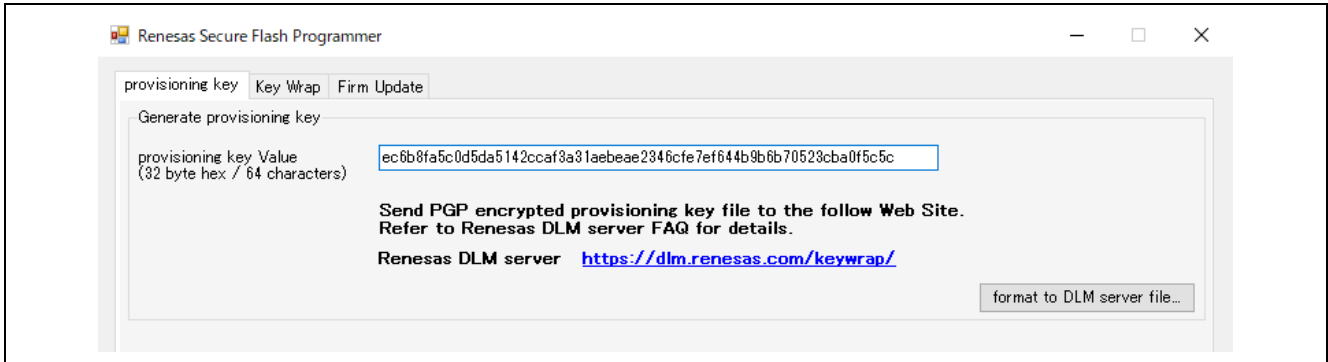
Incorporate the output C source files into your project. The key index can be output from the TSIP by passing the key data to the `R_TSIP_GenerateAes128KeyIndex()` function in your program.

### 7.5.6 Key Update Operation Procedure

If an update keyring has been injected beforehand, it is not necessary to obtain a new provisioning key when performing a key update.

#### 1. Generating a provisioning key

On the provisioning key tab, specify a provisioning key value and generate a provisioning key file having a file name with the extension \*.key. The file name used in the example below is sample.key.



**Figure 7.6 Generating a Provisioning Key by Specifying a Value on the provisioning key Tab**

Click the **format to DLM server file...** button to generate a provisioning key file.

#### 2. Obtaining an encrypted provisioning key

Send the sample.key file generated in step 1 to the Renesas Key Wrap service (<https://dlm.renesas.com/keywrap>) to obtain an encrypted provisioning key.

For detailed information, refer to the Renesas Key Wrap Service FAQ or the relevant application note.

## 3. Generating update keyring files

On the Key Wrap tab, generate update keyring files.

Choose the appropriate option for **Select MCU** based on whether the device to be used is provided with a TSIP or TSIP-Lite module and according to the data flash size.

Select the key type for **Key Type** and enter key data in the **Key Data** field, then click the **Register** button.

For **provisioning key File Path**, enter the path of the provisioning key file generated in step 1. For **encrypted provisioning key File Path**, enter the path of the encrypted provisioning key file obtained as described in step 2. In this example, **(Random)** is selected for **IV** in the interest of simplicity.

After configuring all the settings, click the **Generate Key Files...** button.

You can output the key data files (key\_data.c and key\_data.h) to a folder of your choice.

In this example, the settings are configured to generate files for both an AES 128-bit key and an update keyring.

The screenshot shows the 'Key Wrap' tab with the following settings:

- MCU:** Select MCU: TSIP(DF Memory 32KB)
- Key Setting:**
  - Key Type: Update Key Ring
  - Key Data: d0aec19726cbc0e2fb403866b9b465a6c0d05b7a60362d5f435f9a3e98c79084
  - Register button
- Key List:**

| Key Type        | Key Data                                                         | Delete |
|-----------------|------------------------------------------------------------------|--------|
| AES-128bit      | 000102030405060708090a0b0c0d0e0f                                 |        |
| Update Key Ring | d0aec19726cbc0e2fb403866b9b465a6c0d05b7a60362d5f435f9a3e98c79084 |        |
- provisioning key:**
  - provisioning key File Path: C:\work\samplekey (Browse...)
  - encrypted provisioning key File Path: C:\work\samplekey\_enckey (Browse...)
  - IV (16 byte hex / 32 characters): (Random)
  - Generate Key Files... button

**Figure 7.7 Example AES 128 Key and Update Keyring File Output Settings on Key Wrap Tab**

Incorporate the output C source files into your project. The update keyring key index can be output from the TSIP by passing the key data to the `R_TSIP_GenerateUpdateKeyRingKeyIndex()` function in your program. Pass the output update keyring key index to the second parameter of `R_TSIP_Open()`.

Next, the wrapping method using an update keyring for key updating in the field is described.

## 4. Generating an update keyring \*.key file

Enter the **Update Key Ring** value setting configured in step 3 as the **provisioning key Value** on the provisioning key tab and click the **format to DLM server file...** button to generate an update keyring \*.key file.

Figure 7.8 provisioning key Tab of Renesas Secure Flash Programmer

## 5. Generating key files for updating key data

On the Key Wrap tab, generate key files for updating the key data.

Choose the appropriate option for **Select MCU** based on whether the device to be used is provided with a TSIP or TSIP-Lite module and according to the data flash size.

Select the key type for **Key Type** and enter key data in the **Key Data** field, then click the **Register** button.

For **provisioning key File Path**, enter the path to the update keyring provisioning key file generated in step 4. For **encrypted provisioning key File Path**, you can enter the path to any encrypted provisioning key file of your choice because the value will not actually be used. In this example, the path to the encrypted provisioning key file (UpdateKeyRing.key) obtained as described in step 2 is specified.

In this example, **(Random)** is selected for **IV** in the interest of simplicity.

After configuring all the settings, click the **Generate Key Files...** button.

You can output the key data files (key\_data.c and key\_data.h) to a folder of your choice.

Figure 7.9 Update Key Encryption Settings Using Update Keyring in Key Wrap Tab of Renesas Secure Flash Programmer

6. Input `key_data.encrypted_user_xxx_key[]` and `key_data.iv[]` from the output key file as the `encrypted_key` and `iv` parameters of `R_TSIP_UpdatexxxKeyIndex()`. (The cryptographic algorithm and key length are represented by `xxx`.)  
For AES 128, input `key_data.encrypted_user_aes128_key[]` as the `encrypted_key` parameter of `R_TSIP_UpdateAes128KeyIndex()` and `key_data.iv` as the `iv` parameter of `R_TSIP_UpdateAes128KeyIndex()`.
7. The key index output by `R_TSIP_UpdatexxxKeyIndex()` can be used to generate a new key. In this case, delete the previous key generation information.

## 7.5.7 Usage Notes

### (1) Output File Formats

When Renesas Secure Flash Programmer is used to output a C source file containing key data, the member names in the output structures differ from those output by Security Key Management Tool. For the correspondences between the member names, refer to `key_data.c` and `key_data.h` in the `\src` folder of the demo projects, the files in the `\genkey` folder, and the table below.

**Table 7-9 Correspondences between Structure Members in C Source Files**

| Renesas Secure Flash Programmer* <sup>1</sup>            | Security Key Management Tool* <sup>2</sup>   | Description                     |
|----------------------------------------------------------|----------------------------------------------|---------------------------------|
| <code>encrypted_user_xxxx_key[]</code>                   | <code>encrypted_user_key[]</code>            | Encrypted user key              |
| <code>encrypted_provisioning_key[]</code> * <sup>3</sup> | <code>wufpk[]</code> * <sup>3</sup>          | Encrypted provisioning key      |
| <code>iv[]</code> * <sup>3</sup>                         | <code>initial_vector[]</code> * <sup>3</sup> | Initialization vector           |
| <code>user_xxxx_key_index[]</code>                       | —                                            | Key index                       |
| —                                                        | <code>keytype</code>                         | Not used by the RX TSIP driver. |
| —                                                        | <code>shared_key_number</code>               | Not used by the RX TSIP driver. |
| —                                                        | <code>crc[]</code>                           | Not used by the RX TSIP driver. |

Notes: 1. Names of members of the `st_key_block_data_t.key_data` structure.

2. Names of members of the `g_<keyname>` structure. For details, refer to 4.5.4.2, **csource** Option for **filetype**, in Security Key Management Tool: User's Manual.

3. Use the sample values for **wufpk** and **initial\_vector** in all keys.

## 8. Reference Documents

User's Manual: Hardware

(The latest version can be downloaded from the Renesas Electronics website.)

Technical Updates/Technical News

(The latest information can be downloaded from the Renesas Electronics website.)

User's Manual: Development Environment

RX Family CC-RX Compiler User's Manual (R20UT3248)

(The latest version can be downloaded from the Renesas Electronics website.)



## Revision History

| Rev. | Date          | Description |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
|------|---------------|-------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|      |               | Page        | Summary                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| 1.00 | Sep. 27, 2015 | —           | First edition issued                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
| 1.01 | Jun. 27, 2016 | —           | Added firmware update functionality                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| 1.02 | May 31, 2017  | —           | <ul style="list-style-type: none"> <li>Added AES-CMAC functionality</li> <li>Combined R_TSIP_SelfCheck1(), R_TSIP_SelfCheck2(), and R_TSIP_SoftwareReset into R_TSIP_Open()</li> <li>Added parameters for R_TSIP_InstallAes128UserKey() and R_TSIP_InstallAes256UserKey()</li> <li>Changed content on firmware updating using a USB flash drive</li> </ul>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |
| 1.03 | Sep. 30, 2017 | —           | <ul style="list-style-type: none"> <li>Added following flags for SHA and RSA: TSIP_SHA1, TSIP_SHA256, TSIP_RSA_1024, TSIP_RSA_2048</li> <li>Added R_TSIP_ERROR_PROHIBIT_FUNCTION error</li> </ul>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| 1.04 | Feb. 28, 2018 | —           | <ul style="list-style-type: none"> <li>Added following flags for TLS and SECURE BOOT: TSIP_TLS, TSIP_SHA_1_HMAC, TSIP_SHA_256_HMAC, SECURE_BOOT</li> <li>Changed return value type to enum e_tsip_err_t</li> <li>Deleted R_TSIP_Rsa1024ModularExponent</li> <li>Deleted R_TSIP_Rsa2048ModularExponent</li> <li>Added TLS APIs</li> </ul>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
| 1.05 | Apr. 30, 2018 | —           | <ul style="list-style-type: none"> <li>Added following flags for TDES and MD5: TSIP_TDES_ECB_ENCRYPT, TSIP_TDES_ECB_DECRYPT, TSIP_TDES_CBC_ENCRYPT, TSIP_TDES_CBC_DECRYPT, TSIP_MD5</li> <li>Changed following flags for RSA: RSASSA_1024, RSASSA_2048, RSAES_1024, RSAES_2048</li> <li>Added MD5 APIs</li> <li>Added TDES APIs</li> <li>Added RSAES-PKCS1-v1_5 APIs</li> <li>Added TLS APIs</li> </ul>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| 1.06 | Sep. 28, 2018 | —           | <ul style="list-style-type: none"> <li>Added support for RX66T</li> <li>Deleted R_TSIP_TlsAes128CbcEncryptInit/Update/Final, R_TSIP_TlsAes128CbcDecryptInit/Update/Final, R_TSIP_TlsAes256CbcEncryptInit/Update/Final, R_TSIP_TlsAes256CbcDecryptInit/Update/Final, R_TSIP_TlsSha1HmacGenerateInit/Update/Final, R_TSIP_TlsSha256HmacGenerateInit/Update/Final, R_TSIP_TlsSha1HmacVerifyInit/Update/Final, R_TSIP_TlsSha256HmacVerifyInit/Update/Final</li> <li>Changed parameter key index type in Init API of each algorithm</li> <li>Added key update APIs</li> <li>R_TSIP_Aes128CbcEncryptInit/Update/Final<br/>R_TSIP_Aes128CbcDecryptInit/Update/Final<br/>R_TSIP_Aes256CbcEncryptInit/Update/Final<br/>R_TSIP_Aes256CbcDecryptInit/Update/Final</li> <li>Added description of support for TLS</li> <li>Changed parameter key index type in Init API of each algorithm</li> <li>Added key update APIs</li> <li>Added RSA key generation API</li> <li>Amended SHA-HMAC APIs</li> </ul> |
| 1.07 | Feb. 28, 2019 | —           | Added support for RX72T                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |

| Rev. | Date          | Description |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
|------|---------------|-------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|      |               | Page        | Summary                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| 1.08 | Sep. 30, 2019 | —           | <ul style="list-style-type: none"> <li>Added support for RX23W, RX72M, and elliptic curve cryptography</li> <li>Changed SECURE_BOOT configuration to TSIP_SECURE_BOOT, and deleted TSIP_INSTALL_KEY_RING_INDEX</li> <li>Changed R_TSIP_GenerateTdesUserKeyIndex to R_TSIP_GenerateTdesKeyIndex</li> </ul>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| 1.09 | Mar. 31, 2020 | —           | <ul style="list-style-type: none"> <li>Added CCM and HMAC key generation and ECDH and key wrap functionality</li> <li>Added support for RX66N and RX72N</li> <li>Deleted R_TSIP_Open parameter and terminology explanation of s_flash, and newly added files_flash.c</li> <li>Added TSIP_USER_HASH_ENABLED configuration and user-defined function R_TSIP_RSA_IF_HASH used by RSA signature generation/verification function</li> <li>Added TSIP_ERR_FAIL to return values of GCM calculation preparation functions and unified the order in which the return values are listed</li> <li>Changed description of return value TSIP_ERR_PARAMETER to “Invalid input data” for cases other than handles</li> <li>Added description of members of key index to Parameters for functions that generate RSA and ECC public keys</li> <li>Amended description in Parameters for RSA encryption and decryption functions</li> <li>Added functionality for selecting message and hash value as data types for signature generation and verification functions</li> <li>Corrected spelling of “character” and “mirror”</li> <li>Changed s_inst1/2 to key_index_1/2, including type, and added R_TSIP_UpdateTlsRsaPublicKeyIndex as target of key_index_1</li> </ul> |
| 1.10 | Jun. 30, 2020 | —           | <ul style="list-style-type: none"> <li>Added ECC P-384 key installation, key generation, and key update functionality</li> <li>Added ARC4 and ECDSA P-384 functionality</li> <li>Added ECDH P-256 functionality support for RX72M, RX66N, and RX72N</li> <li>Added key wrap functionality support for RX72M, RX66N, and RX72N</li> <li>Added ARC4 functionality and support for RX72N to demo project</li> <li>Deleted TSIP_USER_HASH_ENABLED configuration and user-defined function R_TSIP_RSA_IF_HASH used by RSA signature generation and verification functions</li> <li>Added TSIP_ECDSA_P384, TSIP_ECDH_P256, and TSIP_USER_SHA_384_ENABLED configurations</li> <li>Changed name of R_TSIP_EcdhXXX() ECDH key exchange function to R_TSIP_EcdhP256XXX()</li> <li>Changed ECC public key structure tsip_ecc_public_key_index_t</li> </ul>                                                                                                                                                                                                                                                                                                                                                                                                           |

| Rev. | Date          | Description |                                                                                                                                                                                                                                                                                                                                                                                                                                                      |
|------|---------------|-------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|      |               | Page        | Summary                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| 1.11 | Sep. 30, 2020 | —           | <ul style="list-style-type: none"> <li>Added DH 2048-bit and ECDHE 512-bit functionality</li> <li>Unified the description of iv in Parameters for R_TSIP_GenerateXXXKeyIndex() and R_TSIP_UpdateXXXKeyIndex()</li> <li>Deleted ECDH (AES GCM 128 with IV) functionality from R_TSIP_EcdhP256Init()</li> <li>Changed R_TSIP_AesXXXKeyWrap() and R_TSIP_AesXXXKeyUnwrap() into API functions common to MCUs with TSIP-Lite and TSIP modules</li> </ul> |
| 1.12 | Jun. 30, 2021 | —           | Added AES cryptography project and TLS cooperation function project                                                                                                                                                                                                                                                                                                                                                                                  |
| 1.13 | Aug. 31, 2021 | —           | Added support for RX671                                                                                                                                                                                                                                                                                                                                                                                                                              |
| 1.14 | Oct. 22, 2021 | —           | Added support for TLS 1.3 (RX65N only)                                                                                                                                                                                                                                                                                                                                                                                                               |
| 1.15 | Mar. 31, 2022 | —           | <ul style="list-style-type: none"> <li>Added support for TLS 1.3 (RX66N, RX72M, and RX72N)</li> <li>Added support for TLS 1.2 RSA 4096-bit</li> <li>Added hash calculation-in-progress acquisition function</li> <li>Deleted reffolder, r_tsip_md5_rx.c, and r_tsip_sha_rx.c, and added r_tsip_hash_rx.c</li> </ul>                                                                                                                                  |
| 1.16 | Sep. 15, 2022 | —           | <ul style="list-style-type: none"> <li>Added support for TLS 1.3 (Resumption/0-RTT)</li> <li>Added support for AES-CTR</li> <li>Added support for RSA3072 and RSA4096</li> </ul>                                                                                                                                                                                                                                                                     |
| 1.17 | Jan. 20, 2023 | —           | <ul style="list-style-type: none"> <li>Revised section structure of application note</li> <li>Added support for TLS 1.3 server (RX65N and RX72N)</li> </ul>                                                                                                                                                                                                                                                                                          |
| 1.18 | May 24, 2023  | —           | Added support for RX26T                                                                                                                                                                                                                                                                                                                                                                                                                              |
| 1.19 | Nov. 30, 2023 | —           | Update example of Secure Bootloader / Firmware Update                                                                                                                                                                                                                                                                                                                                                                                                |
| 1.20 | Feb. 28, 2024 | —           | Applied software workaround to avoid the HW issue of AES-CCM decryption (TSIP-Lite)                                                                                                                                                                                                                                                                                                                                                                  |
| 1.21 | Jun. 28, 2024 | —           | <ul style="list-style-type: none"> <li>Added support for TLS 1.2 server (RX65N, RX66N, RX72M, and RX72N)</li> <li>Added supported public key type of R_TSIP_TlsServersEphemeralEcdhPublicKeyRetrieves</li> </ul>                                                                                                                                                                                                                                     |

## 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](http://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/](http://www.renesas.com/contact/).