RX Family

Implementing TLS Using TSIP Driver

Introduction

The Trusted Secure IP (TSIP) driver supports APIs for SSL/TLS (referred to below as TLS) communication. This document describes the TLS APIs of the TSIP driver and how to implement them in user programs.

Related Documents

- RX Family TSIP (Trusted Secure IP) Module Firmware Integration Technology (R20AN0371)
- RX Family How to implement FreeRTOS OTA by using Amazon Web Services on RX65N (R01AN5549)

Contents

Notes:

- AWS™ is a trademark of Amazon.com, Inc. or its affiliates. (https://aws.amazon.com/trademark-guidelines)
- FreeRTOS™ is a trademark of Amazon Web Services, Inc. (https://freertos.org/copyright.html)
- Git® is a trademark of Software Freedom Conservancy, Inc. (https://www.git-scm.com/about/trademark)
- GitHub® is a trademark of GitHub, Inc. (https://github.com/logos)
- Arm® is a trademark of Arm Limited or its subsidiaries. (https://www.arm.com/company/policies/trademarks/guidelines-trademarks)
- Mbed™ is a trademark of Arm Limited or its subsidiaries. (https://www.arm.com/company/policies/trademarks/guidelines-trademarks)
- OpenSSL™ is a trademark of OpenSSL Software Foundation. (https://www.openssl.org/policies/trademark.html)

1. Overview

1.1 Advantages of TLS Communication Using TSIP

The TSIP driver supports APIs for TLS. These APIs provide the following two advantages.

- No keying information is handled as plaintext during TLS protocol processing, thereby reducing the risk that customer keying information stored on the device may leak.
- Hardware acceleration speeds up encryption processing.

1.2 Cipher Suites Supported by TSIP Driver

The TSIP driver supports the following cipher suites conforming to 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

1.3 TLS APIs of TSIP Driver

Table 1.1 lists the TSIP driver APIs used for TLS communication. For details of each API, refer to section 2, Implementing TLS Communication Using TSIP, and the application note RX Family TSIP (Trusted Secure IP) Module Firmware Integration Technology (R20AN0371).

Table 1.1 API Functions Used for TLS Communication

Where Used	API Function
Certificate installation	R_TSIP_GenerateTlsRsaPublicKeyIndex R_TSIP_Close R_TSIP_Open R_TSIP_TlsRootCertificateVerification
Certificate	R_TSIP_TlsCertificateVerification
Server Key Exchange Client Key Exchange (ECDHE key exchange algorithm)	R_TSIP_TlsServersEphemeralEcdhPublicKeyRetrieves R_TSIP_GenerateTlsP256EccKeyIndex R_TSIP_TlsGeneratePreMasterSecretWithEccP256Key
Client Key Exchange (RSA key exchange algorithm)	R_TSIP_TlsGeneratePreMasterSecret R_TSIP_TlsEncryptPreMasterSecretWithRsa2048PublicKey
Certificate Verify	R_TSIP_RsassaPkcs1024/2048SignatureGenerate R_TSIP_RsassaPkcs1024/2048SignatureVerification R_TSIP_EcdsaP192/224/256/384SignatureGenerate R_TSIP_EcdsaP192/224/256/384SignatureVerification
Finished	R_TSIP_TlsGenerateMasterSecret R_TSIP_TlsGenerateVerifyData R_TSIP_TlsGenerateSessionKey R_TSIP_Sha1HmacGenerateInit/Update/Final R_TSIP_Sha1HmacVerifyInit/Update/Final R_TSIP_Sha256HmacGenerateInit/Update/Final R_TSIP_Sha256HmacVerifyInit/Update/Final R_TSIP_Aes128CbcEncryptInit/Update/Final R_TSIP_Aes128CbcDecryptInit/Update/Final R_TSIP_Aes256CbcEncryptInit/Update/Final R_TSIP_Aes256CbcDecryptInit/Update/Final R_TSIP_Aes128GcmEncryptInit/Update/Final R_TSIP_Aes128GcmDecryptInit/Update/Final
Application Data	R_TSIP_TlsGenerateSessionKey R_TSIP_Sha1HmacGenerateInit/Update/Final R_TSIP_Sha1HmacVerifyInit/Update/Final R_TSIP_Sha256HmacGenerateInit/Update/Final R_TSIP_Sha256HmacVerifyInit/Update/Final R_TSIP_Aes128CbcEncryptInit/Update/Final R_TSIP_Aes128CbcDecryptInit/Update/Final R_TSIP_Aes256CbcEncryptInit/Update/Final R_TSIP_Aes256CbcDecryptInit/Update/Final R_TSIP_Aes128GcmEncryptInit/Update/Final R_TSIP_Aes128GcmDecryptInit/Update/Final

1.4 Definitions of Terms

Terms used in this document are defined below.

Table 1.2 Terms

Terms	Description
User key	The plain key used by the user. It is not used on the device.
Encrypted key	Keying information generated by using an UFPK to encrypt the user key with AES-128 and appending a MAC value.
Wrapped Key	User key converted to a format usable by the TSIP driver. Generated by the TSIP from the Encrypted Key.
UFPK	User Factory Programming Key A key to generate the Encrypted Key from The User Key. It is generated by the user and not used on the device.
W-UFPK	Wrapped UFPK Keying information generated by the DLM server by using UFPK. It is decrypted to the UFPK with HRK and used in the TSIP.
Hardware Root Key (HRK)	A key that exists only inside the TSIP and in a secure room (the DLM server, etc.) at Renesas.
Key Wrap service (https://dlm.renesas.com/)	The key administration server at Renesas. Used to perform key wrapping (encrypting) of UFPKs.

2. Implementing TLS Communication Using TSIP

Figure 2.1 shows an outline flowchart of TLS 1.2 communication and the processing performed using the TSIP driver. The processing enclosed in white boxes in the figure must be implemented using the TSIP driver. In order to make use of the TLS APIs of the TSIP driver, it is first necessary to use the TSIP driver to verify the integrity of the root CA certificate stored on the device. To do this it is necessary to append the signature to be used by the TSIP for verification to the root CA certificate beforehand.

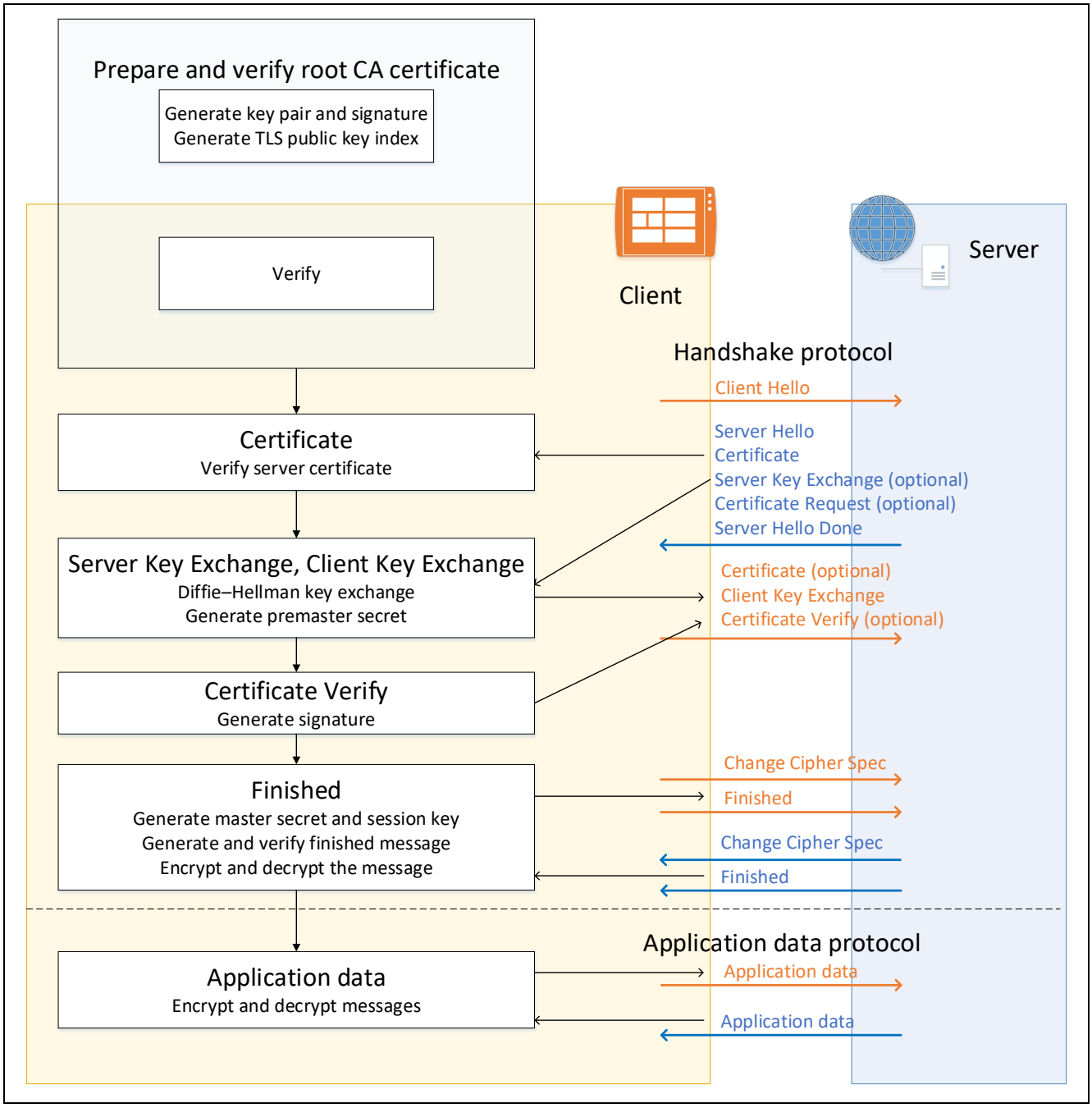


Figure 2.1 Outline of TLS Communication and Processing Performed Using TSIP Driver

For details of the parts of Figure 2.1 involving use of the TSIP driver, refer to Figure 6.1 and Figure 6.2 in section 6.2, Flowchart of TLS Negotiation and Calls to TSIP Driver.

Section 2.1 below describes preparation of the root CA certificate and client certificate as well as verification using the TSIP. Sections 2.3 and 2.4 describe the implementation of TLS protocols using the TSIP driver.

2.1 Preparation Beforehand

2.1.1 Preparation of Root CA Certificate

Before using the TLS APIs of the TSIP driver to extract the public key from the root CA certificate, it is necessary to verify the integrity of the root CA certificate.

Follow the steps below to obtain the root CA certificate and generate the signature to be verified by the TSIP driver. Refer to Figure 2.2 for the preparation sequence.

1. Obtain the root CA certificate.
2. Convert the root CA certificate to DER format.
3. Generate the signature of the root CA certificate and generate an RSA 2048-bit key pair to be used for signature verification.
4. Use the private key from the generated key pair to generate the signature corresponding to the root CA certificate. The signature format is “RSA2048 PSS with SHA256.”

The TSIP driver will not except input of user keys in plaintext, so the RSA 2048-bit public key used for signature verification must be converted to an Encrypted Key that will be accepted by the TSIP driver and embedded into a program. Generating the Encrypted Key can be done with Security Key Management Tool. For detail, please refer the application note RX Family TSIP (Trusted Secure IP) Module Firmware Integration Technology (R20AN0371).

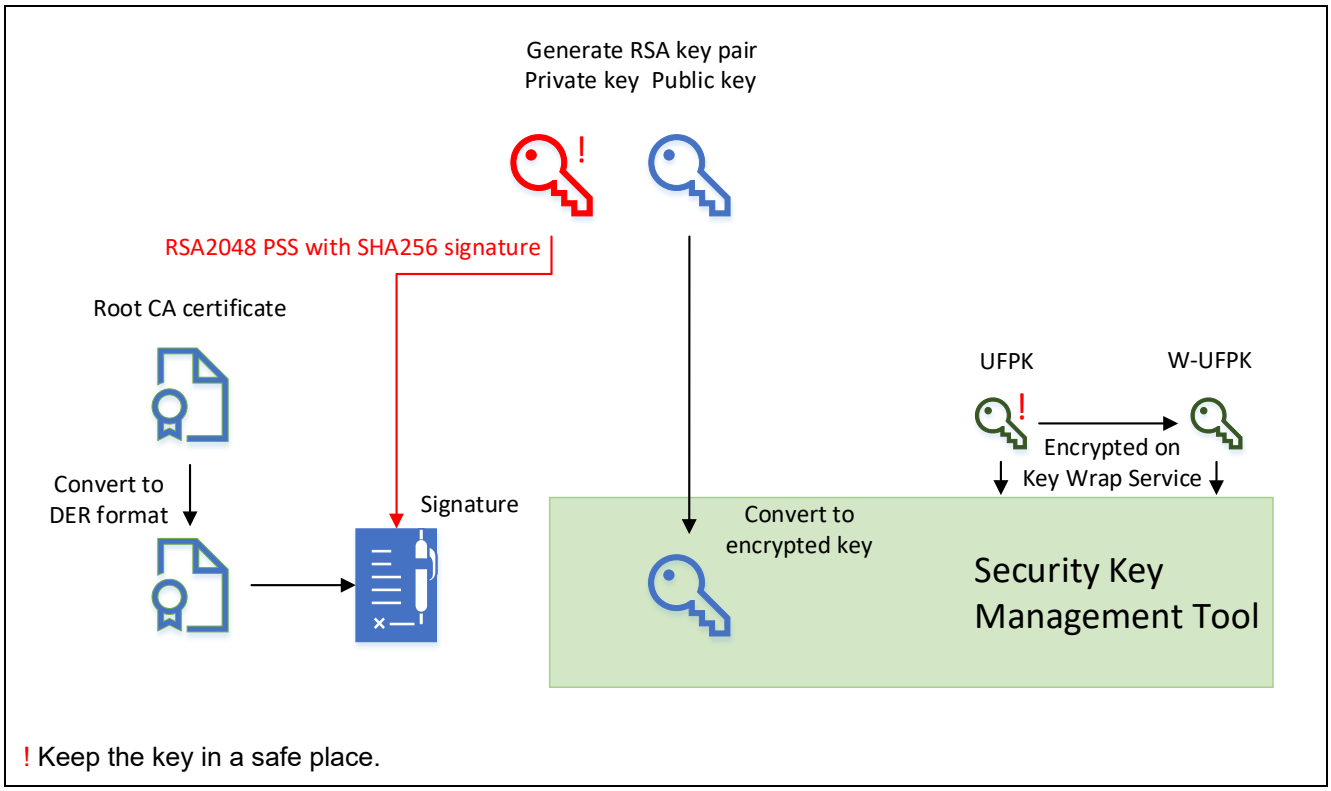


Figure 2.2 Root CA Certificate Preparation Sequence

2.1.2 Preparation of Client Certificate

This procedure involves generating the client key pair and preparing the client certificate.

Follow the steps below to generate the key pair and accept issuance of the client certificate. Refer to Figure 2.3 for the preparation sequence.

1. Generate an RSA and ECC key pair for use by the client.
2. Generate a certificate signing request (CSR) for the generated key pair.
3. Submit the CSR to the certificate authority (CA).
4. Obtain the client certificate issued by the CA based on the CSR.

The TSIP driver will not except input of user keys in plaintext, so the key pair used for signature generation and verification by the client must be converted to an Encrypted Key that will be accepted by the TSIP driver and embedded into a program. Generating the Encrypted Key can be done with Security Key Management Tool. For detail, please refer the application note RX Family TSIP (Trusted Secure IP) Module Firmware Integration Technology (R20AN0371).

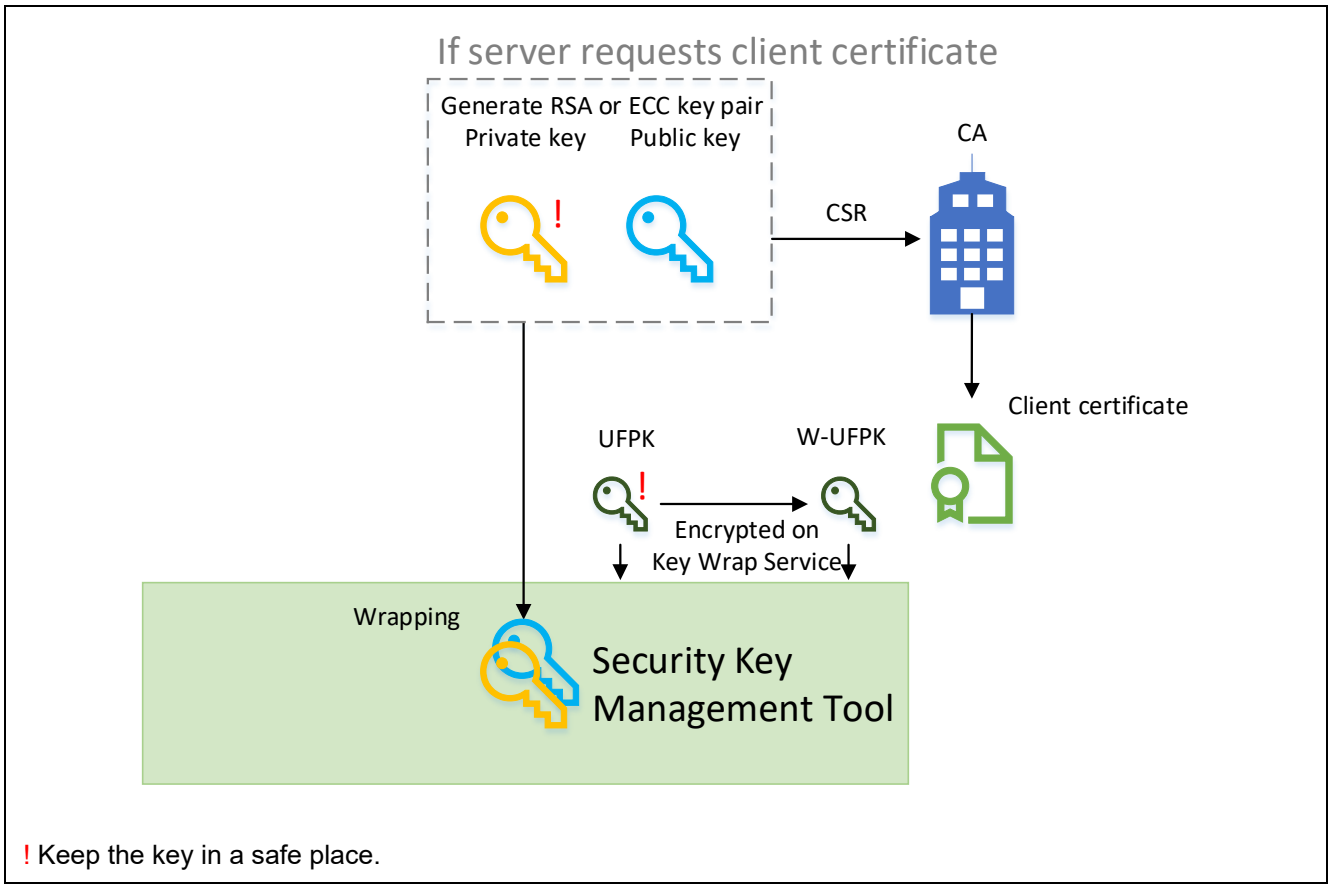


Figure 2.3 Key Pair and Client Certificate Generation Sequence

2.2 Verifying Root CA Certificate

Follow the steps below to verify the root CA certificate, using the DER format certificate, signature, and encrypted key created as described in section 2.1.1. Note that there can be a division of programs between steps 3 and 4. However, the TLS public key index must have been generated on the same device. Refer to Figure 2.4 for the processing sequence and to Table 2.1 for details of the TSIP driver APIs used.

1. Use the R_TSIP_Open() function to validate the TSIP.
2. Use the R_TSIP_GenerateTlsRsaPublicKeyIndex() function to generate the TLS public key index.
3. Use the R_TSIP_Close() function to halt operation of the TSIP.
4. Use the R_TSIP_Open() function to validate the TSIP once again and read in the TLS public key index.
5. Use the R_TSIP_TlsRootCertificateVerification() function to verify the root CA certificate.

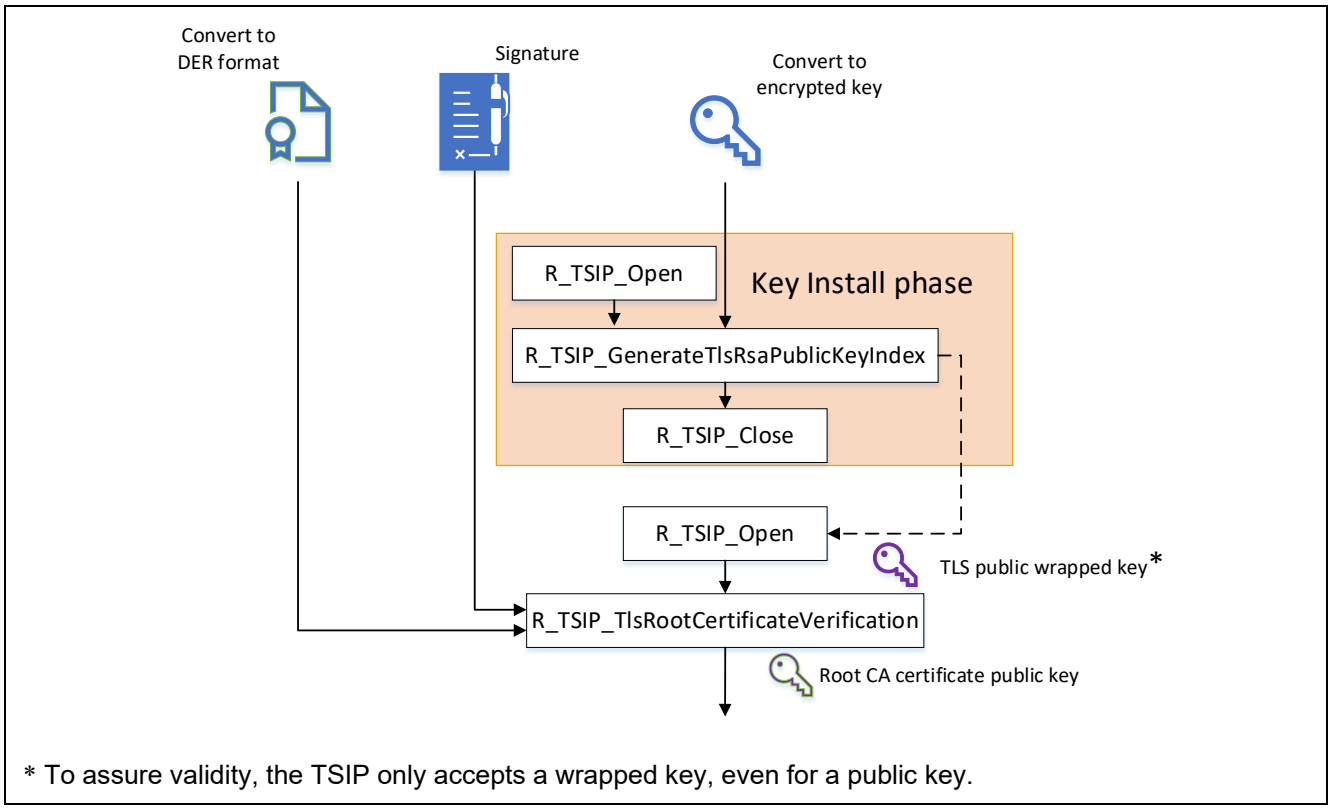


Figure 2.4 Root CA Certificate Verification Sequence

Table 2.1 API Functions Used for Root CA Certificate Preparation and Installation

API Function	Description
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)	Validates the TSIP. To validate the TLS APIs of the TSIP driver, it is necessary to input the **key_index** parameter of R_TSIP_GenerateTlsRsaPublicKeyIndex(). **Parameters** For **key_index_1**, input a null pointer the first time the function is called, and input the **key_index** value output by R_TSIP_GenerateTlsRsaPublicKeyIndex() the second time the function is called. If the key update function is not used, input a null pointer for **key_index_2**.
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)	Outputs the RSA public key key_index value used by R_TSIP_TlsRootCertificateVerification(). **Parameters** For **encrypted_provisioning_key**, **iv**, and **encrypted_key**, input the corresponding variables in key_data.c, output by Renesas Secure Flash Programmer.
e_tsip_err_t R_TSIP_Close (void)	Halts operation of the TSIP.
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);)	Verifies the root CA certificate prepared beforehand. **Parameters** For **public_key_type**, input the type of the public key contained in the certificate. For **certificate**, input the certificate in DER format, and for **certificate_length**, input the length of the certificate. For **public_key_*_*_position**, input the addresses corresponding to the start and end points of the public keying information of the root CA certificate obtained by decrypting the certificate. For **signature**, input the signature data corresponding to the certificate. For **encrypted_root_public_key**, keying information is output that is used to verify the server certificate in the next procedure.

2.3 Handshake Protocol

The handshake protocol processing requiring implementation of the TSIP driver is described below.

2.3.1 Certificate

Follow the steps below to verify the certificate chain. Refer to Figure 2.5 for the processing sequence and to Table 2.2 for details of the TSIP driver API used.

1. Prepare the public key (the **encrypted_root_public_key** parameter of the R_TSIP_TlsRootCertificateVerification() function) extracted from the root CA certificate.
2. Use the R_TSIP_TlsCertificateVerification() function to verify the next certificate after the last certificate is verified.
3. If there is an unverified certificate remaining, the certificate verified in step 2 is an intermediate certificate. Prepare the public key (the **encrypted_output_public_key** parameter) contained in the verified certificate and return to step 2. If there are no unverified certificates remaining, the certificate verified in step 2 is the server certificate. Prepare the public key contained in the verified certificate and proceed to the next processing procedure.

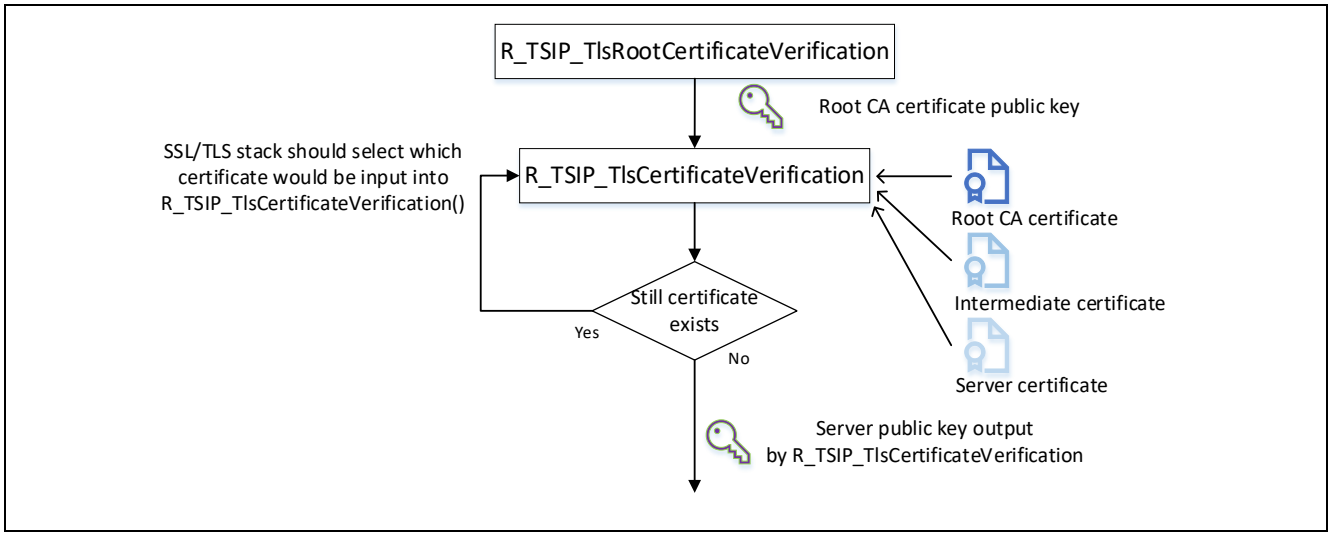


Figure 2.5 Server Certificate Verification Sequence

Table 2.2 API Function Used for Server Certificate Verification

API Function	Description
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)	Verifies the certificate chain. Call this function repeatedly to perform the necessary processing, starting with verification of the intermediate certificate after the root CA certificate and ending with verification of the server certificate. **Parameters** For **public_key_type**, input the type of the public key contained in the certificate. For **certificate**, input the certificate in DER format, and for **certificate_length**, input the length of the certificate. For **signature**, input the signature data from the certificate to be verified. For **public_key_*_*_position**, input the addresses corresponding to the public keying information to be verified. For **encrypted_input_public_key**, input the public key of the certificate verified immediately previously. The parameter **encrypted_output_public_key**, output when the server certificate is verified, is used when R_TSIP_TlsEncryptPreMasterSecretWithRsa2048PublicKey() or R_TSIP_TlsServersEphemeralEcdhPublicKeyRetrieves() is called to perform key exchange.

2.3.2 Server Key Exchange and Client Key Exchange

2.3.2.1 ECDHE Key Exchange

Follow the steps below to perform key exchange. Refer to Figure 2.6 for the processing sequence and to Table 2.3 for details of the TSIP driver APIs used.

1. Use the R_TSIP_TlsServersEphemeralEcdhPublicKeyRetrieves() function to verify the server ephemeral ECDH public key received in the Server Key Exchange message.
2. Use the R_TSIP_GenerateTlsP256EccKeyIndex() function to generate the client ephemeral ECDH key pair. Send the client ephemeral ECDH public key to the server in the Client Key Exchange message.
3. Use the R_TSIP_TlsGeneratePreMasterSecretWithEccP256Key() function to generate the premaster secret from the server ephemeral ECDH public key and client ephemeral ECDH private key.

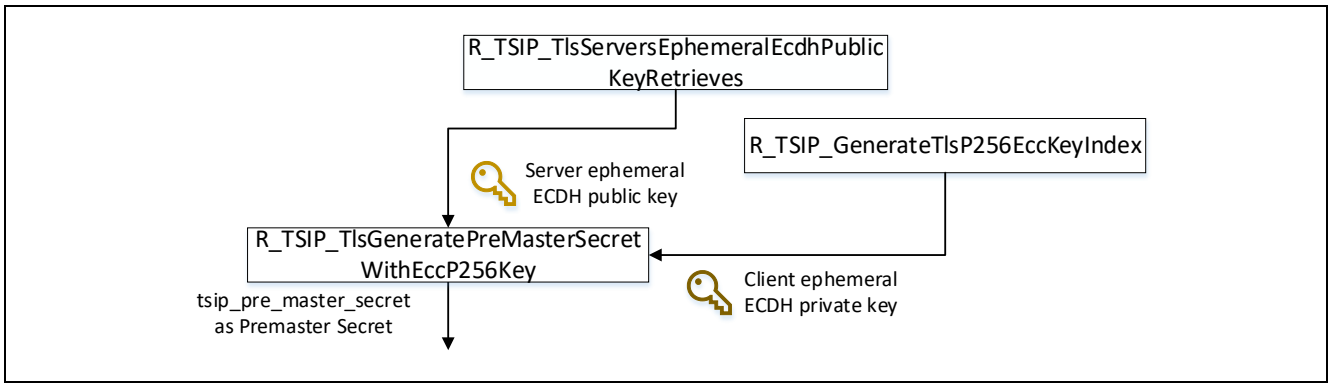


Figure 2.6 ECDHE Key Exchange Sequence

Table 2.3 API Functions Used for ECDHE Key Exchange

API Function	Description
e_tsip_err_t R_TSIP_TlsServersEphemeralEcdhPublicKeyRetrieves (uint32_t public_key_type, uint8_t *client_random, uint8_t *server_random, uint8_t *server_ephemeral_ecdh_public_key, uint8_t *server_key_exchange_signature, uint32_t *encrypted_public_key, uint32_t *encrypted_ephemeral_ecdh_public_key)	Verifies the Server Key Exchange signature based on the server public key. The output is an encrypted ephemeral ECDH public key used by R_TSIP_TlsGeneratePreMasterSecretWithEccP256Key().
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)	Generates a key pair using the ECDH algorithm. The output is the keying information used by R_TSIP_TlsGeneratePreMasterSecretWithEccP256Key() and the ephemeral ECDH public key sent to the server in the Client Key Exchange message.
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)	Outputs the premaster secret based on the values input from R_TSIP_TlsServersEphemeralEcdhPublicKeyRetrieves() and R_TSIP_GenerateTlsP256EccKeyIndex().

2.3.2.2 RSA Key Exchange

Follow the steps below to perform key exchange. Refer to Figure 2.7 for the processing sequence and to Table 2.4 for details of the TSIP driver APIs used.

1. Use the R_TSIP_TlsGeneratePreMasterSecret() function to generate the premaster secret.
2. Use the R_TSIP_TlsEncryptPreMasterSecretWithRsa2048PublicKey() function to encrypt the premaster secret. Send the encrypted premaster secret to the server in the Client Key Exchange message.

Figure 2.7 RSA Key Exchange Sequence

Table 2.4 API Functions Used for RSA Key Exchange

API Function	Description
e_tsip_err_t R_TSIP_TlsGeneratePreMasterSecret (uint32_t *tsip_pre_master_secret)	Generates the premaster secret.
e_tsip_err_t R_TSIP_TlsEncryptPreMasterSecretwithRsaPublicKey ((uint32_t *encrypted_public_key, uint32_t *tsip_pre_master_secret, uint8_t *encrypted_pre_master_secret)	Outputs the premaster secret to be sent to the server in the Client Key Exchange message as data encrypted with the RSA-2048 public key.

2.3.3 Certificate Verify

This procedure involves generating a signature used to verify the client certificate on the server side.

The API functions used differ according to the type of public key contained in the client certificate. Refer to Figure 2.8 for the processing sequence and to Table 2.5, API Functions Used for Client Certificate Verification, for details of the TSIP driver APIs used.

If the public key type is RSA, the following sequence is used to generate the signature.

1. Use the R_TSIP_RsassaPkcs1024/2048SignatureGenerate() function to generate a signature for the message.
2. If necessary, generate a public key index from the public key with R_TSIP_GenerateRsa1024/2048PublicKeyIndex() function, and use the R_TSIP_RsassaPkcs1024/2048SignatureVerification() function to self-verify the generated signature.

If the public key type is ECC, the following sequence is used to generate the signature.

1. Use the R_TSIP_EcdsaP192/224/256/384SignatureGenerate() function to generate a signature for the message.
2. If necessary, generate a public key index from the public key with R_TSIP_GenerateEccP192/224/256/384PublicKeyIndex() function, and use the R_TSIP_EcdsaP192/224/256/384SignatureVerification() function to self-verify the generated signature.

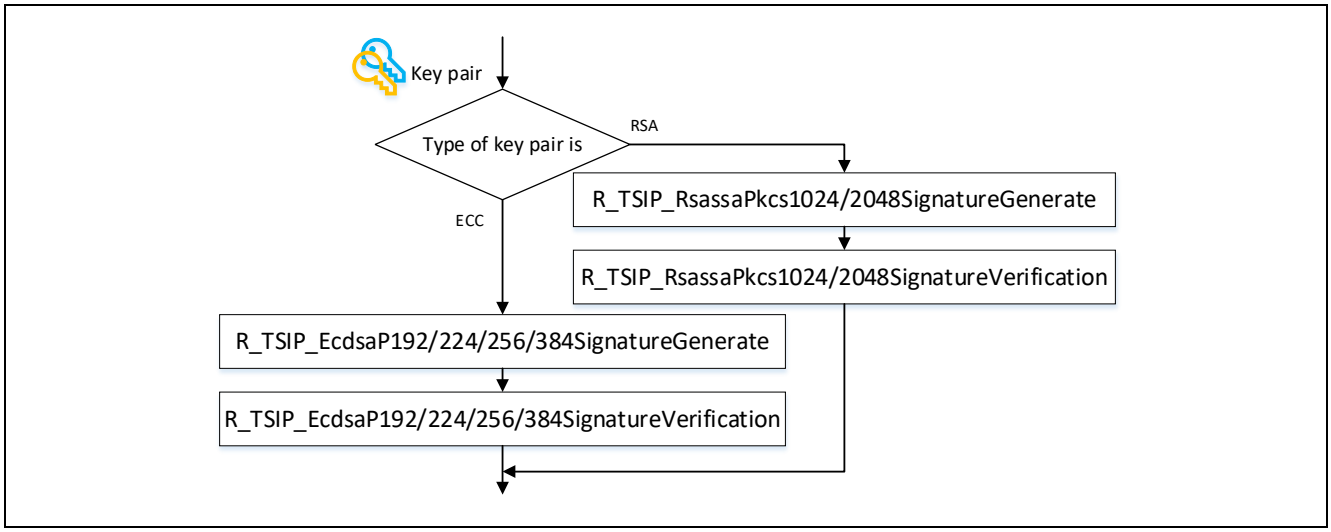


Figure 2.8 Signature Generation Sequence for Client Certificate Verification

Table 2.5 API Functions Used for Client Certificate Verification

API Function	Description
R_TSIP_RsassaPkcs1024/2048SignatureGenerate	Uses an RSA private key to generate an RSASSA-PKCS1-v1_5 signature.
R_TSIP_RsassaPkcs1024/2048SignatureVerification	Uses an RSA public key to verify an RSASSA-PKCS1-v1_5 signature.
R_TSIP_EcdsaP192/224/256/384SignatureGenerate	Uses an ECC private key to generate an ECDSA signature.
R_TSIP_EcdsaP192/224/256/384SignatureVerification	Uses an ECC public key to verify an ECDSA signature.

2.3.4 Finished

Follow the steps below to create and verify the Finished message. Refer to Figure 2.9 for the processing sequence and to Table 2.6, Table 2.7, Table 2.8, Table 2.9, and Table 2.10 for details of the TSIP driver APIs used.

1. Use the R_TSIP_TlsGenerateMasterSecret () function to generate the master secret from the premaster secret.
2. Use the R_TSIP_TlsGenerateSessionKey() function to generate four session keys (client write MAC key, server write MAC key, client write encryption key, and server write encryption key) and two IVs (client write IV and server write IV) from the master secret
3. Use the R_TSIP_TlsGenerateVerifyData() function to generate verify data from the content of the Finished message sent from the client.
4. Use the hash function and AES function supported by the cipher suite to generate and encrypt the signature of the Finished message.
5. Send the Finished message from the client to the server.
6. Receive the Finished message from the server.
7. Use the hash function and AES function supported by the cipher suite to decrypt and verify the signature of the Finished message.
8. Use the R_TSIP_TlsGenerateVerifyData() function to verify the verify data. This concludes the handshake protocol.

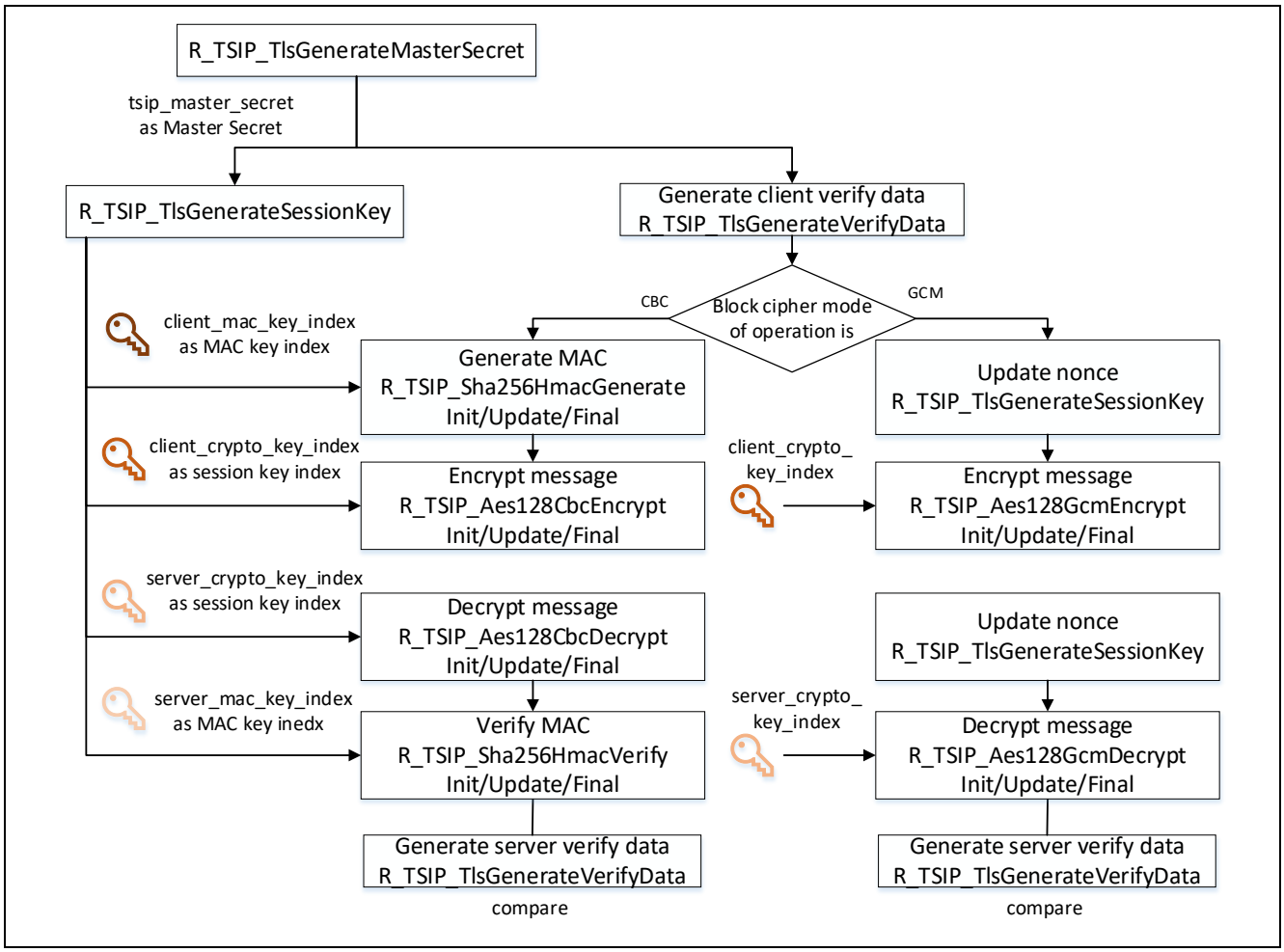


Figure 2.9 Finished Message Generation and Verification Sequence

Table 2.6 API Functions Used for Master Secret Generation and Finished Message Generation and Verification

API Function	Description
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)	Generates the master secret based on the premaster secret.
e_tsip_err_t R_TSIP_TlsGenerateSessionKey (uint32_t select_cipher_suite, uint32_t * tsip_master_secret, uint8_t *client_random, uint8_t *server_random, uint8_t *nonce_explict, tsip_hmac_sha_key_index_t *client_mac_key_index, tsip_hmac_sha_key_index_t *server_mac_key_index, tsip_aes_key_index_t *client_crypto_key_index, tsip_aes_key_index_t *server_crypto_key_index, uint8_t *client_iv, uint8_t *server_iv)	Outputs the session key **key_index** (**client_mac_key_index**, **server_mac_key_index**, **client_crypto_key_index**, and **server_crypto_key_index**) based on the master secret. The IVs are contained in **client_crypto_key_index** and **server_crypto_key_index**. When using CBC mode, input NULL for **nonce_explicit**. When using GCM mode, input a nonce value.
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)	Generates verify data for the Finished message.

Table 2.7 API Functions Used for Encryption in CBC Mode

API Function	Description
R_TSIP_Sha1HmacGenerateInit/Update/Final R_TSIP_Sha256HmacGenerateInit/Update/Final	Uses **client_mac_key_index** to generate the MAC value of the data to be sent to the server.
R_TSIP_Aes128CbcEncryptInit/Update/Final R_TSIP_Aes256CbcEncryptInit/Update/Final	Uses **client_crypto_key_index** to encrypt the data to be sent to the server.

Table 2.8 API Functions Used for Decryption in CBC Mode

API Function	Description
R_TSIP_Aes128CbcDecryptInit/Update/Final R_TSIP_Aes256CbcDecryptInit/Update/Final	Uses **server_crypto_key_index** to decrypt cipher text received from the server.
R_TSIP_Sha1HmacVerifyInit/Update/Final R_TSIP_Sha256HmacVerifyInit/Update/Final	Uses **server_mac_key_index** to verify the MAC value of decrypted data received from the server.

Table 2.9 API Functions Used for Encryption in GCM Mode

API Function	Description
R_TSIP_TlsGenerateSessionKey	Updates the nonce in the TSIP. For **nonce_explicit**, input a different nonce for each packet.
R_TSIP_Aes128GcmEncryptInit/Update/Final	Uses **client_crypto_key_index** to encrypt and generate a certification tag for data to be sent to the server. Input NULL for **Ivec** and 0 for **ivec_len**.

Table 2.10 API Functions Used for Decryption in GCM Mode

API Function	Description
R_TSIP_TlsGenerateSessionKey	Updates the nonce in the TSIP. For **nonce_explicit**, input the nonce contained in the packet.
R_TSIP_Aes128GcmDecryptInit/Update/Final	Uses **server_crypto_key_index** to decrypt and verify the certification tag of data received from the server.

2.4 Application Data Protocol

Like the Finished message of the handshake protocol, the application data protocol uses TSIP driver APIs to carry out encryption and decryption processing and to perform encrypted communication. Refer to Table 2.7 and Table 2.8 for the APIs used in CBC mode and to Table 2.9 and Table 2.10 for the APIs used in GCM mode.

3. Sample Code

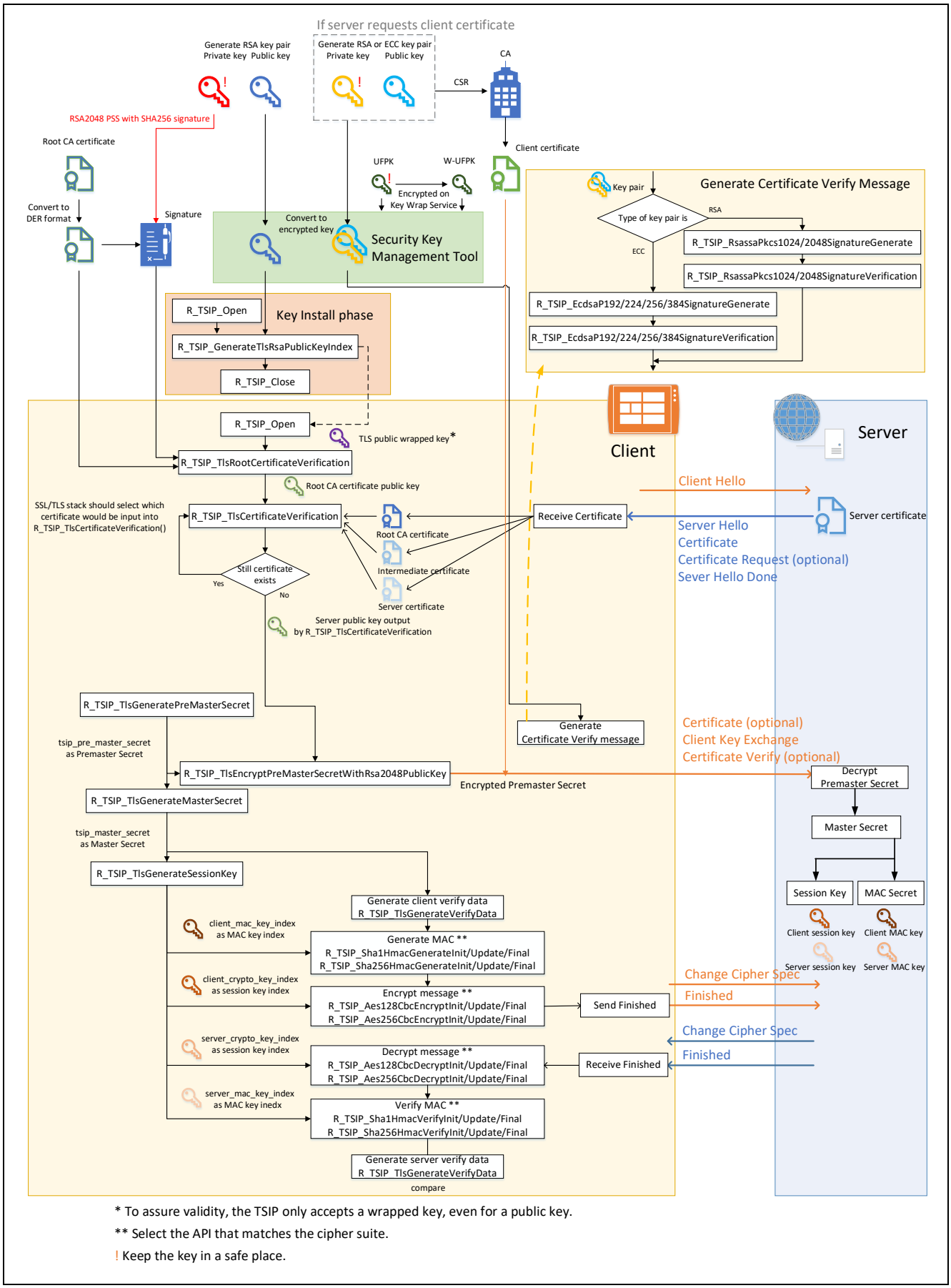
To refer the sample code, please access to the link **iot-reference-rx** and refer the links described in the Getting Started Guide of the latest Release.

https://github.com/renesas/iot-reference-rx

4. Appendix

4.1 Flowchart of TLS Negotiation and Calls to TSIP Driver

Below are flowcharts of TLS negotiation overall and associated calls to the TSIP driver. Figure 4.1 applies when the key exchange algorithm is RSA and Figure 4.2 when it is ECDHE.

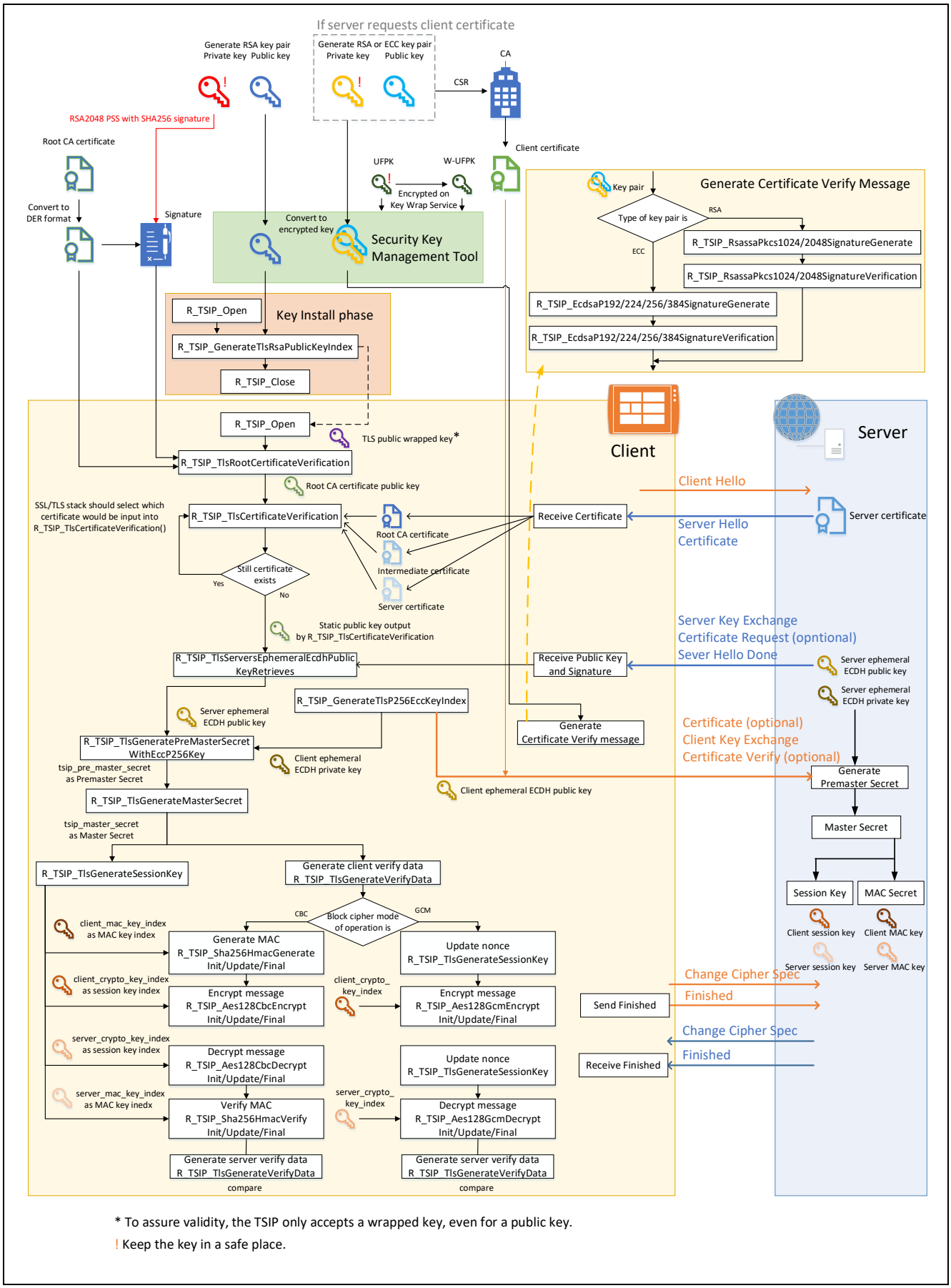


* To assure validity, the TSIP only accepts a wrapped key, even for a public key.

** Select the API that matches the cipher suite.

! Keep the key in a safe place.

Figure 4.1 Flowchart of TLS Negotiation and Associated Calls to TSIP Driver (RSA Key Exchange Algorithm)



* To assure validity, the TSIP only accepts a wrapped key, even for a public key.

! Keep the key in a safe place.

Figure 4.2 Flowchart of TLS Negotiation and Associated Calls to TSIP Driver (ECDHE Key Exchange Algorithm)

Revision History

Rev.	Date	Description	
		Page	Summary
1.00	Jun. 30, 2021	—	First edition issued
1.01	Mar. 31, 2022	—	Error correction
1.02	Sep. 15, 2022	16	2.3.3 Added explanation about generating Certificate Verify
1.03	Jun. 28, 2024	—	Replacement of descriptions about sample project

General Precautions in the Handling of Microprocessing Unit and Microcontroller Unit Products

The following usage notes are applicable to all Microprocessing unit and Microcontroller unit products from Renesas. For detailed usage notes on the products covered by this document, refer to the relevant sections of the document as well as any technical updates that have been issued for the products.

1. Precaution against Electrostatic Discharge (ESD)

   A strong electrical field, when exposed to a CMOS device, can cause destruction of the gate oxide and ultimately degrade the device operation. Steps must be taken to stop the generation of static electricity as much as possible, and quickly dissipate it when it occurs. Environmental control must be adequate. When it is dry, a humidifier should be used. This is recommended to avoid using insulators that can easily build up static electricity. Semiconductor devices must be stored and transported in an anti-static container, static shielding bag or conductive material. All test and measurement tools including work benches and floors must be grounded. The operator must also be grounded using a wrist strap. Semiconductor devices must not be touched with bare hands. Similar precautions must be taken for printed circuit boards with mounted semiconductor devices.
2. Processing at power-on

   The state of the product is undefined at the time when power is supplied. The states of internal circuits in the LSI are indeterminate and the states of register settings and pins are undefined at the time when power is supplied. In a finished product where the reset signal is applied to the external reset pin, the states of pins are not guaranteed from the time when power is supplied until the reset process is completed. In a similar way, the states of pins in a product that is reset by an on-chip power-on reset function are not guaranteed from the time when power is supplied until the power reaches the level at which resetting is specified.
3. Input of signal during power-off state

   Do not input signals or an I/O pull-up power supply while the device is powered off. The current injection that results from input of such a signal or I/O pull-up power supply may cause malfunction and the abnormal current that passes in the device at this time may cause degradation of internal elements. Follow the guideline for input signal during power-off state as described in your product documentation.
4. Handling of unused pins

   Handle unused pins in accordance with the directions given under handling of unused pins in the manual. The input pins of CMOS products are generally in the high-impedance state. In operation with an unused pin in the open-circuit state, extra electromagnetic noise is induced in the vicinity of the LSI, an associated shoot-through current flows internally, and malfunctions occur due to the false recognition of the pin state as an input signal become possible.
5. Clock signals

   After applying a reset, only release the reset line after the operating clock signal becomes stable. When switching the clock signal during program execution, wait until the target clock signal is stabilized. When the clock signal is generated with an external resonator or from an external oscillator during a reset, ensure that the reset line is only released after full stabilization of the clock signal. Additionally, when switching to a clock signal produced with an external resonator or by an external oscillator while program execution is in progress, wait until the target clock signal is stable.
6. Voltage application waveform at input pin

   Waveform distortion due to input noise or a reflected wave may cause malfunction. If the input of the CMOS device stays in the area between V_{IL} (Max.) and V_{IH} (Min.) due to noise, for example, the device may malfunction. Take care to prevent chattering noise from entering the device when the input level is fixed, and also in the transition period when the input level passes through the area between V_{IL} (Max.) and V_{IH} (Min.).
7. Prohibition of access to reserved addresses

   Access to reserved addresses is prohibited. The reserved addresses are provided for possible future expansion of functions. Do not access these addresses as the correct operation of the LSI is not guaranteed.
8. Differences between products

   Before changing from one product to another, for example to a product with a different part number, confirm that the change will not lead to problems. The characteristics of a microprocessing unit or microcontroller unit products in the same group but having a different part number might differ in terms of internal memory capacity, layout pattern, and other factors, which can affect the ranges of electrical characteristics, such as characteristic values, operating margins, immunity to noise, and amount of radiated noise. When changing to a product with a different part number, implement a system-evaluation test for the given product.

Notice

1. Descriptions of circuits, software and other related information in this document are provided only to illustrate the operation of semiconductor products and application examples. You are fully responsible for the incorporation or any other use of the circuits, software, and information in the design of your product or system. Renesas Electronics disclaims any and all liability for any losses and damages incurred by you or third parties arising from the use of these circuits, software, or information.
2. Renesas Electronics hereby expressly disclaims any warranties against and liability for infringement or any other claims involving patents, copyrights, or other intellectual property rights of third parties, by or arising from the use of Renesas Electronics products or technical information described in this document, including but not limited to, the product data, drawings, charts, programs, algorithms, and application examples.
3. No license, express, implied or otherwise, is granted hereby under any patents, copyrights or other intellectual property rights of Renesas Electronics or others.
4. You shall be responsible for determining what licenses are required from any third parties, and obtaining such licenses for the lawful import, export, manufacture, sales, utilization, distribution or other disposal of any products incorporating Renesas Electronics products, if required.
5. You shall not alter, modify, copy, or reverse engineer any Renesas Electronics product, whether in whole or in part. Renesas Electronics disclaims any and all liability for any losses or damages incurred by you or third parties arising from such alteration, modification, copying or reverse engineering.
6. Renesas Electronics products are classified according to the following two quality grades: “Standard” and “High Quality”. The intended applications for each Renesas Electronics product depends on the product’s quality grade, as indicated below.
   "Standard": Computers; office equipment; communications equipment; test and measurement equipment; audio and visual equipment; home electronic appliances; machine tools; personal electronic equipment; industrial robots; etc.
   "High Quality": Transportation equipment (automobiles, trains, ships, etc.); traffic control (traffic lights); large-scale communication equipment; key financial terminal systems; safety control equipment; etc.

   Unless expressly designated as a high reliability product or a product for harsh environments in a Renesas Electronics data sheet or other Renesas Electronics document, Renesas Electronics products are not intended or authorized for use in products or systems that may pose a direct threat to human life or bodily injury (artificial life support devices or systems; surgical implantations; etc.), or may cause serious property damage (space system; undersea repeaters; nuclear power control systems; aircraft control systems; key plant systems; military equipment; etc.). Renesas Electronics disclaims any and all liability for any damages or losses incurred by you or any third parties arising from the use of any Renesas Electronics product that is inconsistent with any Renesas Electronics data sheet, user’s manual or other Renesas Electronics document.
7. No semiconductor product is absolutely secure. Notwithstanding any security measures or features that may be implemented in Renesas Electronics hardware or software products, Renesas Electronics shall have absolutely no liability arising out of any vulnerability or security breach, including but not limited to any unauthorized access to or use of a Renesas Electronics product or a system that uses a Renesas Electronics product. RENESAS ELECTRONICS DOES NOT WARRANT OR GUARANTEE THAT RENESAS ELECTRONICS PRODUCTS, OR ANY SYSTEMS CREATED USING RENESAS ELECTRONICS PRODUCTS WILL BE INVULNERABLE OR FREE FROM CORRUPTION, ATTACK, VIRUSES, INTERFERENCE, HACKING, DATA LOSS OR THEFT, OR OTHER SECURITY INTRUSION (“Vulnerability Issues”). RENESAS ELECTRONICS DISCLAIMS ANY AND ALL RESPONSIBILITY OR LIABILITY ARISING FROM OR RELATED TO ANY VULNERABILITY ISSUES. FURTHERMORE, TO THE EXTENT PERMITTED BY APPLICABLE LAW, RENESAS ELECTRONICS DISCLAIMS ANY AND ALL WARRANTIES, EXPRESS OR IMPLIED, WITH RESPECT TO THIS DOCUMENT AND ANY RELATED OR ACCOMPANYING SOFTWARE OR HARDWARE, INCLUDING BUT NOT LIMITED TO THE IMPLIED WARRANTIES OF MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE.
8. When using Renesas Electronics products, refer to the latest product information (data sheets, user’s manuals, application notes, “General Notes for Handling and Using Semiconductor Devices” in the reliability handbook, etc.), and ensure that usage conditions are within the ranges specified by Renesas Electronics with respect to maximum ratings, operating power supply voltage range, heat dissipation characteristics, installation, etc. Renesas Electronics disclaims any and all liability for any malfunctions, failure or accident arising out of the use of Renesas Electronics products outside of such specified ranges.
9. Although Renesas Electronics endeavors to improve the quality and reliability of Renesas Electronics products, semiconductor products have specific characteristics, such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Unless designated as a high reliability product or a product for harsh environments in a Renesas Electronics data sheet or other Renesas Electronics document, Renesas Electronics products are not subject to radiation resistance design. You are responsible for implementing safety measures to guard against the possibility of bodily injury, injury or damage caused by fire, and/or danger to the public in the event of a failure or malfunction of Renesas Electronics products, such as safety design for hardware and software, including but not limited to redundancy, fire control and malfunction prevention, appropriate treatment for aging degradation or any other appropriate measures. Because the evaluation of microcomputer software alone is very difficult and impractical, you are responsible for evaluating the safety of the final products or systems manufactured by you.
10. Please contact a Renesas Electronics sales office for details as to environmental matters such as the environmental compatibility of each Renesas Electronics product. You are responsible for carefully and sufficiently investigating applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive, and using Renesas Electronics products in compliance with all these applicable laws and regulations. Renesas Electronics disclaims any and all liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.
11. Renesas Electronics products and technologies shall not be used for or incorporated into any products or systems whose manufacture, use, or sale is prohibited under any applicable domestic or foreign laws or regulations. You shall comply with any applicable export control laws and regulations promulgated and administered by the governments of any countries asserting jurisdiction over the parties or transactions.
12. It is the responsibility of the buyer or distributor of Renesas Electronics products, or any other party who distributes, disposes of, or otherwise sells or transfers the product to a third party, to notify such third party in advance of the contents and conditions set forth in this document.
13. This document shall not be reprinted, reproduced or duplicated in any form, in whole or in part, without prior written consent of Renesas Electronics.
14. Please contact a Renesas Electronics sales office if you have any questions regarding the information contained in this document or Renesas Electronics products.

(Note1) “Renesas Electronics” as used in this document means Renesas Electronics Corporation and also includes its directly or indirectly controlled subsidiaries.

(Note2) “Renesas Electronics product(s)” means any product developed or manufactured by or for Renesas Electronics.

(Rev.5.0-1 October 2020)

Corporate Headquarters

TOYOSU FORESIA, 3-2-24 Toyosu,
Koto-ku, Tokyo 135-0061, Japan

www.renesas.com

Trademarks

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

Contact information

For further information on a product, technology, the most up-to-date version of a document, or your nearest sales office, please visit:
www.renesas.com/contact/.