RX ファミリ

TSIP ドライバを用いた TLS 実装方法

要旨

Trusted Secure IP (TSIP) ドライバは SSL/TLS (本書では以後 TLS と表記) 通信用の API をサポートしています。本書では、TSIP ドライバの TLS 向け API およびその実装方法を解説します。

関連ドキュメント

- RX ファミリ TSIP(Trusted Secure IP)モジュール Firmware Integration Technology (R20AN0371)
- RX ファミリ RX65N における Amazon Web Services を利用した FreeRTOS OTA 実現方法 (R01AN5549)

目次

注：

- AWS™は Amazon.com, Inc. or its affiliates の商標です。(https://aws.amazon.com/trademark-guidelines)
- FreeRTOS™は Amazon Web Services, Inc.の商標です。(https://freertos.org/copyright.html)
- Git®は Software Freedom Conservancy, Inc.のトレードマークです。
  (https://www.git-scm.com/about/trademark)
- GitHub®は GitHub, Inc.のトレードマークです。(https://github.com/logos)
- Arm®は Arm Limited or its subsidiaries のトレードマークです。
  (https://www.arm.com/company/policies/trademarks/guidelines-trademarks)
- Mbed™は Arm Limited or its subsidiaries のトレードマークです。
  (https://www.arm.com/company/policies/trademarks/guidelines-trademarks)
- OpenSSL™は OpenSSL Software Foundation のトレードマークです。
  (https://www.openssl.org/policies/trademark.html)

1. 概要

1.1 TSIP を用いた TLS 通信のメリット

TSIP ドライバでは TLS 向けの API をサポートしています。本 API を利用することにより以下 2 点のメリットがあります。

- TLS プロトコル処理中で平文の鍵情報を扱わないため、デバイス内に格納されたお客様の鍵情報の漏洩リスクを減らすことができます。
- ハードウェアでアクセラレートすることにより、暗号処理を高速化できます。

1.2 TSIP ドライバでサポートしている Cipher Suite

TSIP ドライバは TLS1.2 に準拠した以下の Cipher Suite をサポートしています。

- 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 TSIP ドライバの TLS 向け API 一覧

表 1-1 に TLS 通信で使用する TSIP ドライバの API を示します。各 API の詳細は「2. TSIP を用いた TLS 通信の実装方法」およびアプリケーションノート「RX ファミリ TSIP(Trusted Secure IP)モジュール Firmware Integration Technology (R20AN0371)」を参照してください。

表 1-1　TLS 通信で使用する API 関数

使用場所	API 関数
証明書のインストール	R_TSIP_GenerateTlsRsaPublicKeyIndex R_TSIP_Close R_TSIP_Open R_TSIP_TlsRootCertificateVerification
Certificate	R_TSIP_TlsCertificateVerification
Server Key Exchange Client Key Exchange (鍵交換方式が ECDHE の場合)	R_TSIP_TlsServersEphemeralEcdhPublicKeyRetrieves R_TSIP_GenerateTlsP256EccKeyIndex R_TSIP_TlsGeneratePreMasterSecretWithEccP256Key
Client Key Exchange (鍵交換方式が RSA の場合)	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　用語の定義

本書で使用される用語の定義を以下に示します。

表 1-2　用語

用語	内容
ユーザ鍵、User Key	ユーザが使用する平文状態の鍵。デバイス上では使用しない。
Encrypted Key	UFPK を使用して User Key を AES128 で暗号化、MAC 付与することで生成される鍵情報。
Wrapped Key	User Key を TSIP ドライバで使用可能な形式に変換したデータ。 TSIP が Encrypted Key から生成する。
UFPK	User Factory Programming Key User Key から Encrypted Key を生成するために使用する、ユーザが設定する鍵。デバイス上では使用しない。
W-UFPK	Wrapped UFPK UFPK を使用して DLM サーバ上で生成する鍵情報。TSIP 内部にて HRK で UFPK に復号されて使用される。
Hardware Root Key (HRK)	TSIP 内部およびルネサスのセキュアルーム（DLM サーバなど）のみに存在する共通の暗号鍵。
Key Wrap サービス (https://dlm.renesas.com/)	Renesas 鍵管理サーバ。UFPK のラッピングに使用する。

2.　TSIP を用いた TLS 通信の実装方法

図 2-1 に TLS1.2 の通信の流れと TSIP ドライバを使用する処理の概略を示します。図中の白い四角は TSIP ドライバの実装が必要な処理です。TSIP ドライバの TLS 向け API を使用する際は、最初に TSIP ドライバを用いて、デバイスに格納したルート CA 証明書の完全性を検証する必要があります。そのために、TSIP ドライバが検証に使用する署名を、事前にルート CA 証明書に付加しておく必要があります。

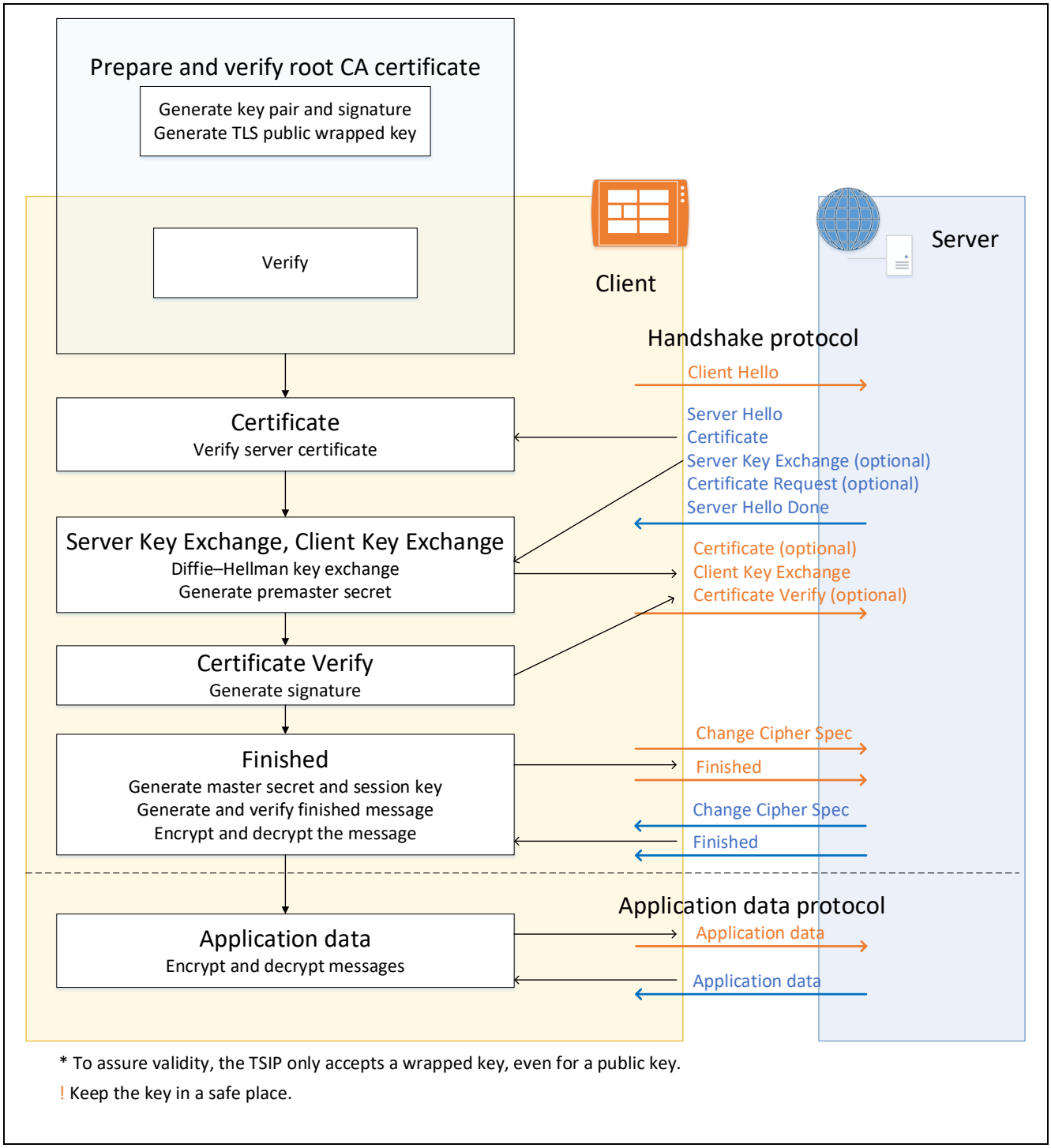


図 2-1　TLS 通信の流れと TSIP ドライバを使用する処理の概略

図 2-1 の TSIP ドライバ使用部分の詳細は、「4.1 TLS ネゴシエーションフローにおける TSIP ドライバの呼び出しフロー」の図 4-1 および図 4-2 を参照してください。

2.1 節ではルート CA 証明書およびクライアント証明書の準備と TSIP を使用した検証方法を説明します。2.3 節、2.4 節では、TSIP ドライバを使用する TLS のプロトコルについて説明します。

2.1　事前準備

2.1.1　ルート CA 証明書の準備

TSIP ドライバの TLS 向け API ではルート CA 証明書から公開鍵を取り出す前にルート CA 証明書の完全性を検証します。

以下の手順に従いルート CA 証明書を入手し、TSIP ドライバが検証する署名を生成します。準備の流れは図 2-2 を参照してください。

1. ルート CA 証明書を入手します。
2. ルート CA 証明書を DER 形式に変換します。
3. ルート CA 証明書の署名生成と署名検証に使用する RSA-2048bit の鍵ペアを生成します。
4. 生成した鍵ペアの秘密鍵を使用して、ルート CA 証明書に対する署名を生成します。署名方式は”RSA2048 PSS with SHA256”です。

TSIP ドライバは、平文のユーザ鍵を入力として受け入れないため、署名検証に使用する RSA-2048bit の公開鍵は、TSIP ドライバが受け入れられる Encrypted Key に変換してプログラムに組み込む必要があります。Encrypted Key の生成は、Security Key Management Tool を使用して行うことができます。詳細は、アプリケーションノート「RX ファミリ TSIP(Trusted Secure IP)モジュール Firmware Integration Technology (R20AN0371)」を参照してください。

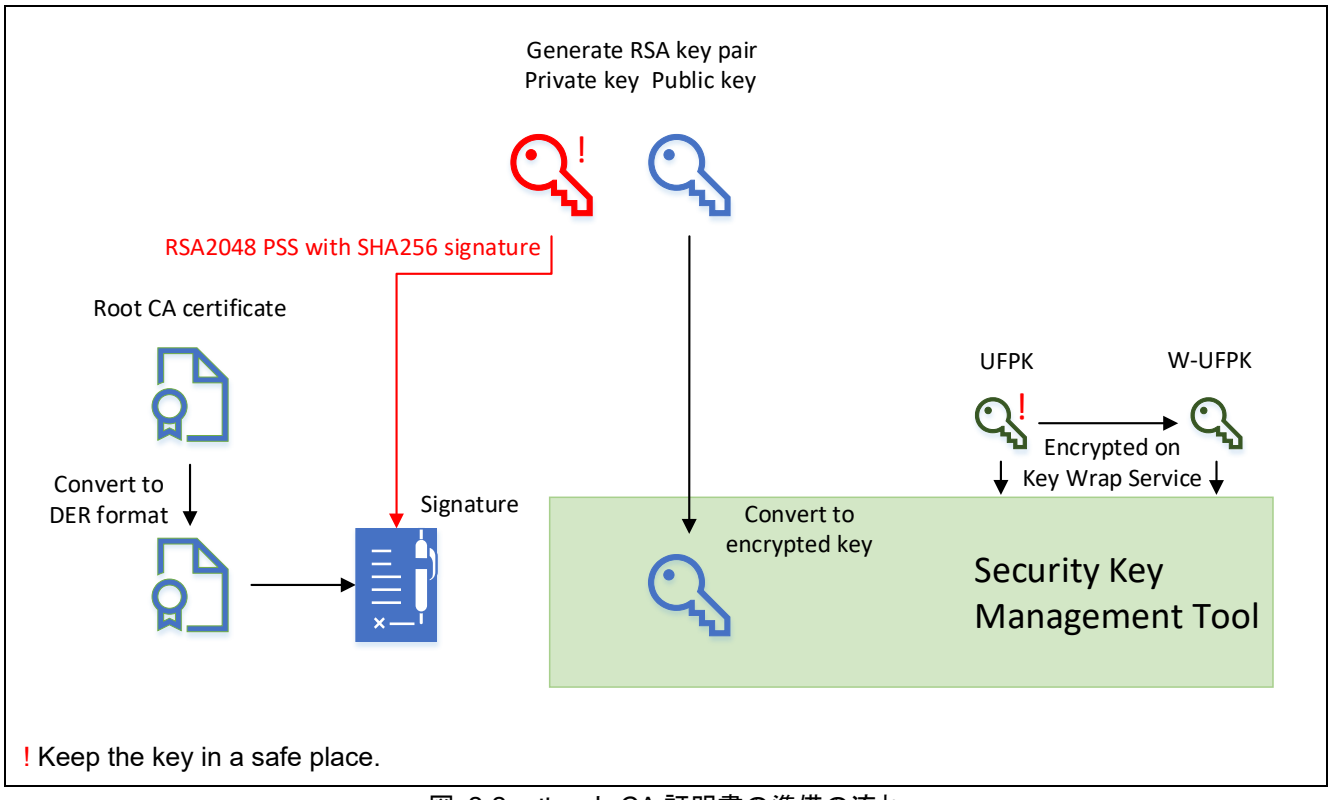


図 2-2　ルート CA 証明書の準備の流れ

2.1.2　クライアント証明書の準備

クライアントの鍵ペアを生成し、クライアント証明書を準備します。

以下の手順に従い、鍵ペアを生成し、クライアント証明書の発行を受けます。準備の流れは図 2-3 を参照してください。

1. クライアントが使用する RSA か ECC の鍵ペアを生成します。
2. 生成した鍵ペアの CSR (Certificate Signing Request: 証明書署名要求) を生成します。
3. CSR を CA (Certificate Authority: 認証局) に提出します。
4. CA が CSR を元に発行したクライアント証明書を入手します。

TSIP ドライバは、平文のユーザ鍵を入力として受け入れないため、クライアントが署名生成/署名検証に使用する鍵ペアは、TSIP ドライバが受け入れられる Encrypted Key に変換してプログラムに組み込む必要があります。Encrypted Key の生成は、Security Key Management Tool を使用して行うことができます。詳細は、アプリケーションノート「RX ファミリ TSIP(Trusted Secure IP)モジュール Firmware Integration Technology (R20AN0371)」を参照してください。

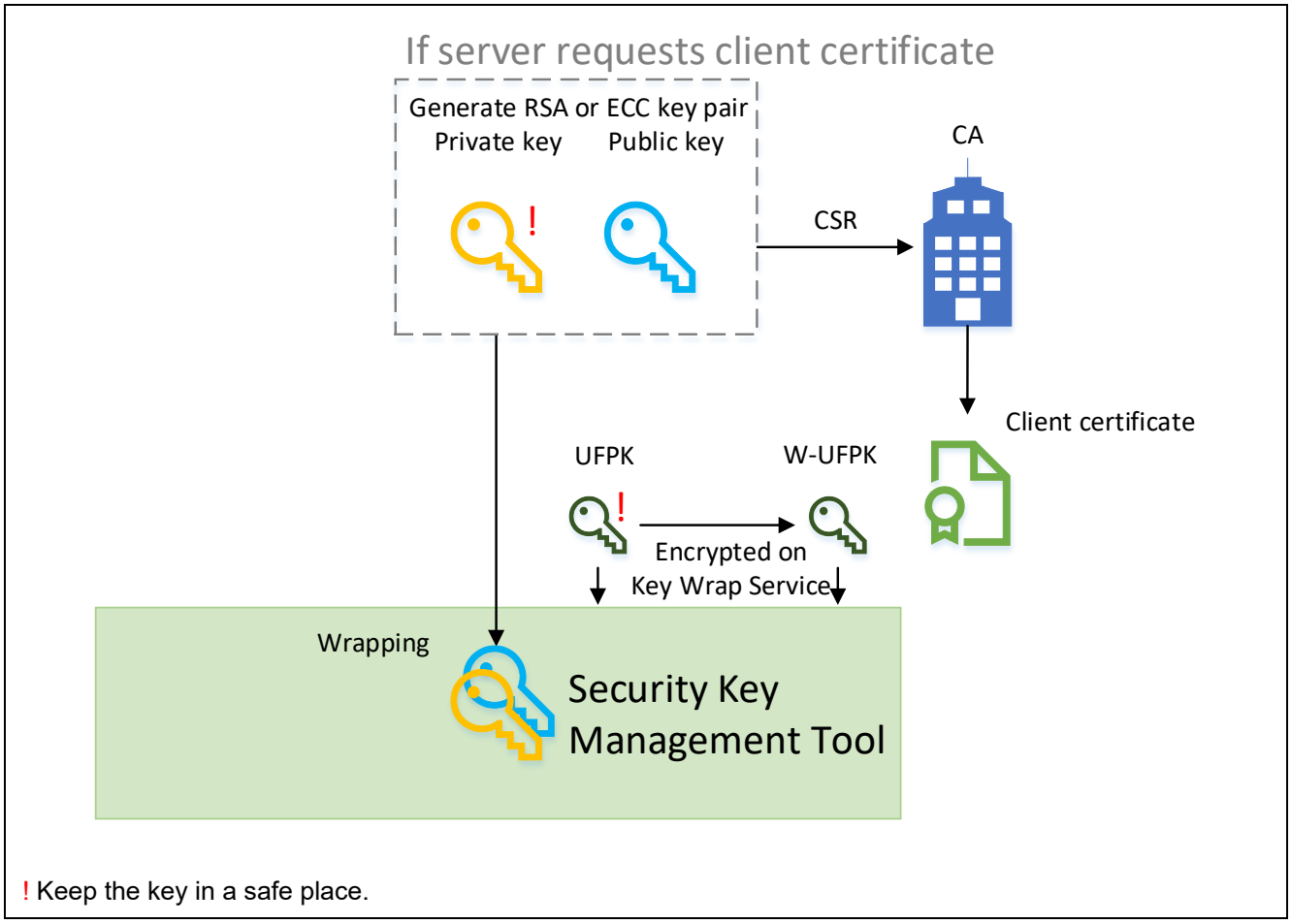
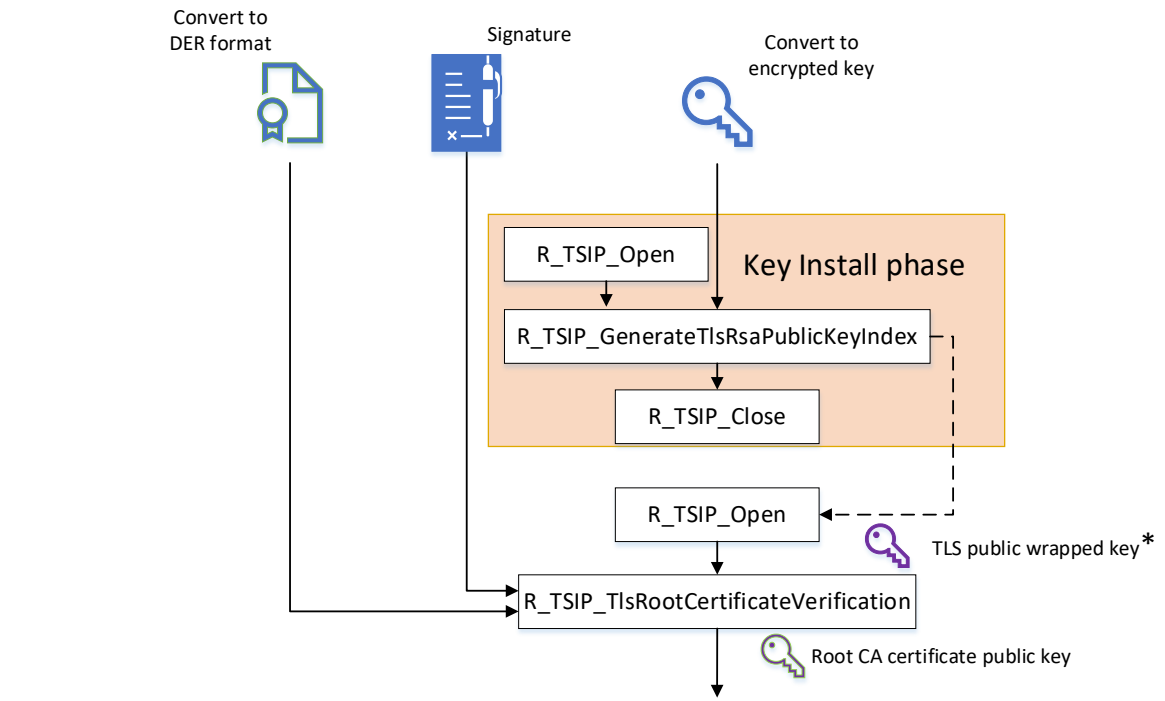


図 2-3　鍵ペアとクライアント証明書の生成の流れ

2.2　ルート CA 証明書の検証

2.1.1 節で作成した DER 形式の証明書、署名および encrypted key を使用して、以下の手順に従いルート CA 証明書を検証します。ここで、手順 3 と手順 4 の間でプログラムを分割することが可能です。ただし、TLS public key index は同一デバイス上で生成したもののみが使用できます。処理の流れは図 2-4 を、使用する TSIP ドライバの API の詳細は表 2-1 を参照してください。

1. R_TSIP_Open() 関数を使用して、TSIP を有効化します。
2. R_TSIP_GenerateTlsRsaPublicKeyIndex() 関数を使用して、TLS public key index を生成します。
3. R_TSIP_Close() 関数を使用して TSIP を停止します。
4. R_TSIP_Open() 関数を使用して TSIP を再度有効化するとともに、TLS public key index を読み込みます。
5. R_TSIP_TlsRootCertificateVerification() 関数を使用してルート CA 証明書を検証します。



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

図 2-4　ルート CA 証明書の検証の流れ

表 2-1　ルート CA 証明書の準備とインストールで使用する API 関数

API 関数	説明
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)	TSIP を有効化します。 TSIP ドライバの TLS 向け API を有効にするには R_TSIP_GenerateTlsRsaPublicKeyIndex()の key_index を入力する必要があります。 **引数** key_index_1 には、1 回目の呼び出しでは NULL ポインタを、2 回目の呼び出しでは R_TSIP_GenerateTlsRsaPublicKeyIndex()が出力した key_index を入力します。鍵更新機能を使用しない場合、key_index_2 には NULL ポインタを入力します。
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	R_TSIP_TlsRootCertificateVerification()で使用する RSA 公開鍵の key_index を出力します。 **引数** encrypted_provisioning_key, iv, encrypted_key には、Renesas Secure Flash Programmer から出力された key_data.c 内の変数を入力します。

)	
e_tsip_err_t R_TSIP_Close (void)	TSIP を停止します。
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);)	あらかじめ用意したルート CA 証明書を検証します。 **引数** public_key_type には、証明書に含まれている公開鍵の種類を入力します。certificate には証明書を DER 形式で入力し、certificate_length にはその長さを入力します。public_key_*_*_position には証明書を解読して得たルート CA 証明書の公開鍵情報の始点・終点の相対アドレスを入力します。signature には証明書に対する署名データを入力します。encrypted_root_public_key には鍵情報が出力され、次に行うサーバ証明書の検証に使用します。

2.3　Handshake Protocol

ここでは、Handshake Protocol において TSIP ドライバの実装が必要な処理を解説します。

2.3.1　Certificate

以下の手順に従い証明書チェーンを検証します。処理の流れは図 2-5 を、使用する TSIP ドライバの API の詳細は表 2-2 を参照してください。

1. ルート CA 証明書から取り出した公開鍵（R_TSIP_TlsRootCertificateVerification() 関数の引数 encrypted_root_public_key）を用意します。
2. R_TSIP_TlsCertificateVerification() 関数を使用して、最後に検証した証明書の次の証明書を検証します。
3. 未検証の証明書が残っている場合、2.で検証した証明書は中間証明書です。検証した証明書に含まれている公開鍵（引数 encrypted_output_public_key）を用意し、2.に戻ります。未検証の証明書が残っていない場合、2.で検証した証明書はサーバ証明書です。検証したサーバ証明書に含まれている公開鍵を用意して次の処理に進みます。

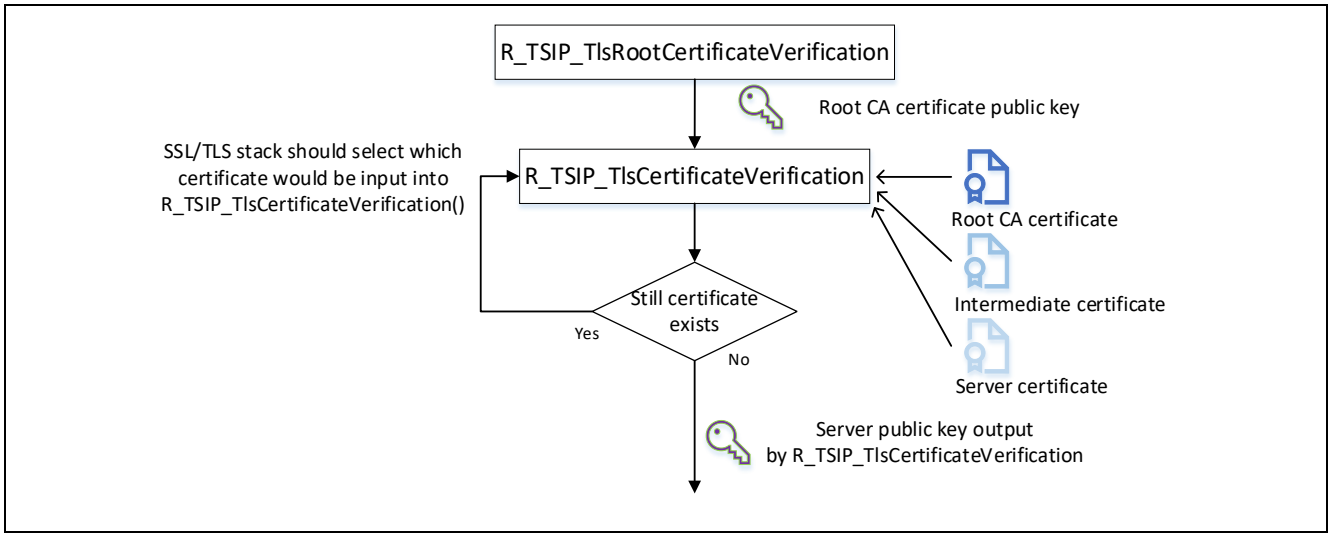


図 2-5　サーバ証明書の検証の流れ

表 2-2　サーバ証明書の検証で使用する API 関数

API 関数	説明
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)	証明書チェーンの検証を行います。ルート CA 証明書の次の中間証明書から検証を行い、サーバ証明書の検証が完了するまで本関数を繰り返し呼び出してください。 **引数** public_key_type には、証明書に含まれている公開鍵の種類を入力します。certificate には証明書を DER 形式で入力し、certificate_length にはその長さを入力します。signature には検証対象となる証明書内の署名データを入力します。public_key_*_*_position には検証対象の公開鍵情報の相対アドレスを入力します。encrypted_input_public_key には、直前に検証した証明書の公開鍵を入力します。サーバ証明書の検証時に出力された encrypted_output_public_key は鍵交換時に呼び出す R_TSIP_TlsEncryptPreMasterSecret WithRsa2048PublicKey()または R_TSIP_TlsServersEphemeralEcdhPublicKeyRetrieves()で使用します。

2.3.2　Server Key Exchange, Client Key Exchange

2.3.2.1　鍵交換方式が ECDHE の場合

以下の手順に従い鍵交換を行います。処理の流れは図 2-6 を、使用する TSIP ドライバの API の詳細は表 2-3 を参照してください。

1. R_TSIP_TlsServersEphemeralEcdhPublicKeyRetrieves() 関数を使用して Server Key Exchange メッセージで受け取ったサーバ ephemeral ECDH 公開鍵を検証します。
2. R_TSIP_GenerateTlsP256EccKeyIndex() 関数を使用してクライアント ephemeral ECDH 鍵ペアを生成します。クライアント ephemeral ECDH 公開鍵は Client Key Exchange メッセージでサーバに送信してください。
3. R_TSIP_TlsGeneratePreMasterSecretWithEccP256Key() 関数を使用してサーバ ephemeral ECDH 公開鍵とクライアント ephemeral ECDH 秘密鍵から premaster secret を生成します。

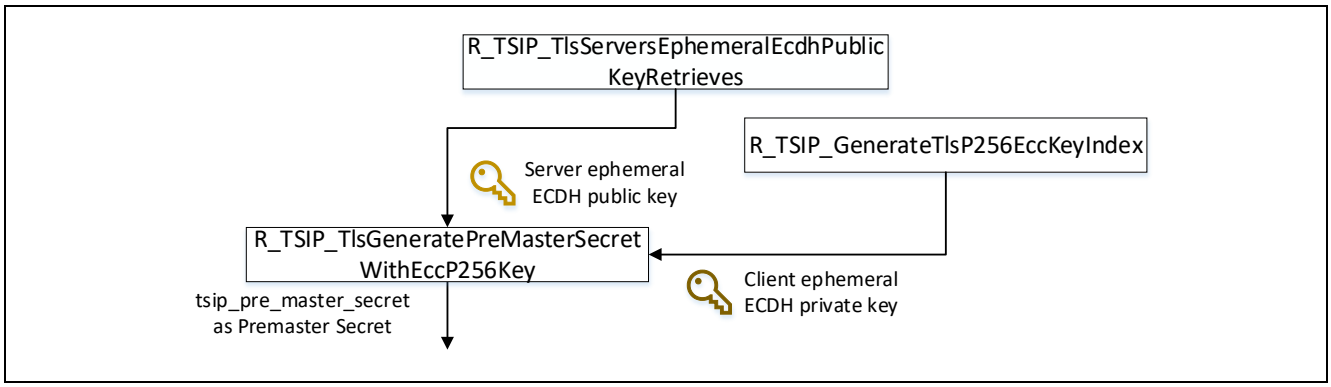


図 2-6　ECDHE による鍵交換の流れ

表 2-3　ECDHE による鍵交換で使用する API 関数

API 関数	説明
e_tsip_err_t R_TSIP_TlsServersEphemeralEcdhPublicKeyRetrieves (　uint32_t public_key_type,	サーバ公開鍵を元に Server Key Exchange の署名を検証します。出力は R_TSIP_TlsGeneratePreMasterSecretWithEccP256

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)	Key()で使用する暗号化された ephemeral ECDH public key です。
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)	ECDH アルゴリズムにおける鍵ペアを生成します。出力は R_TSIP_TlsGeneratePreMasterSecretWithEccP256Key()で使用する鍵情報と、Client Key Exchange メッセージでサーバに送信する ephemeral ECDH public key です。
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)	R_TSIP_TlsServersEphemeralEcdhPublicKeyRetrieves()および R_TSIP_GenerateTlsP256EccKeyIndex()からの入力を元に premaster secret を出力します。

2.3.2.2　鍵交換方式が RSA の場合

以下の手順に従い鍵交換を行います。処理の流れは図 2-7 を、使用する TSIP ドライバの API の詳細は表 2-4 を参照してください。

1. R_TSIP_TlsGeneratePreMasterSecret()関数を使用して premaster secret を生成します。
2. R_TSIP_TlsEncryptPreMasterSecretWithRsa2048PublicKey()関数で premaster secret を暗号化します。
   暗号化された premaster secret は Client Key Exchange メッセージでサーバに送信してください。

図 2-7　RSA による鍵交換の流れ

表 2-4　RSA による鍵交換で使用する API 関数

API 関数	説明
e_tsip_err_t R_TSIP_TlsGeneratePreMasterSecret (uint32_t *tsip_pre_master_secret)	premaster secret を生成します。
e_tsip_err_t R_TSIP_TlsEncryptPreMasterSecretwithRsaPublicKey ((uint32_t *encrypted_public_key, uint32_t *tsip_pre_master_secret, uint8_t *encrypted_pre_master_secret)	Client Key Exchange メッセージでサーバに送信する、premaster secret を RSA-2048 公開鍵で暗号化したデータを出力します。

2.3.3　Certificate Verify

クライアント証明書をサーバ側で検証するための署名を生成します。

クライアント証明書に含まれる公開鍵の種類によって使用する関数が異なります。処理の流れは図 2-8 を、使用する TSIP ドライバの API の詳細は表 2-5　クライアント証明書の検証で使用する API 関数を参照してください。

公開鍵が RSA であれば、以下の流れで署名を生成します。

1. R_TSIP_RsassaPkcs1024/2048SignatureGenerate()関数でメッセージに対する署名を生成します。
2. 必要に応じて、公開鍵から R_TSIP_GenerateRsa1024/2048PublicKeyIndex()関数でユーザ鍵生成情報を生成し、R_TSIP_RsassaPkcs1024/2048SignatureVerification()関数を用いて、生成した署名を自己検証します。

公開鍵が ECC であれば、以下の流れで署名を生成します。

1. R_TSIP_EcdsaP192/224/256/384SignatureGenerate()関数でメッセージに対する署名を生成します。
2. 必要に応じて、公開鍵から R_TSIP_GenerateEccP192/224/256/384PublicKeyIndex()関数でユーザ鍵生成情報を生成し、R_TSIP_EcdsaP192/224/256/384SignatureVerification()関数を用いて、生成した署名を自己検証します。

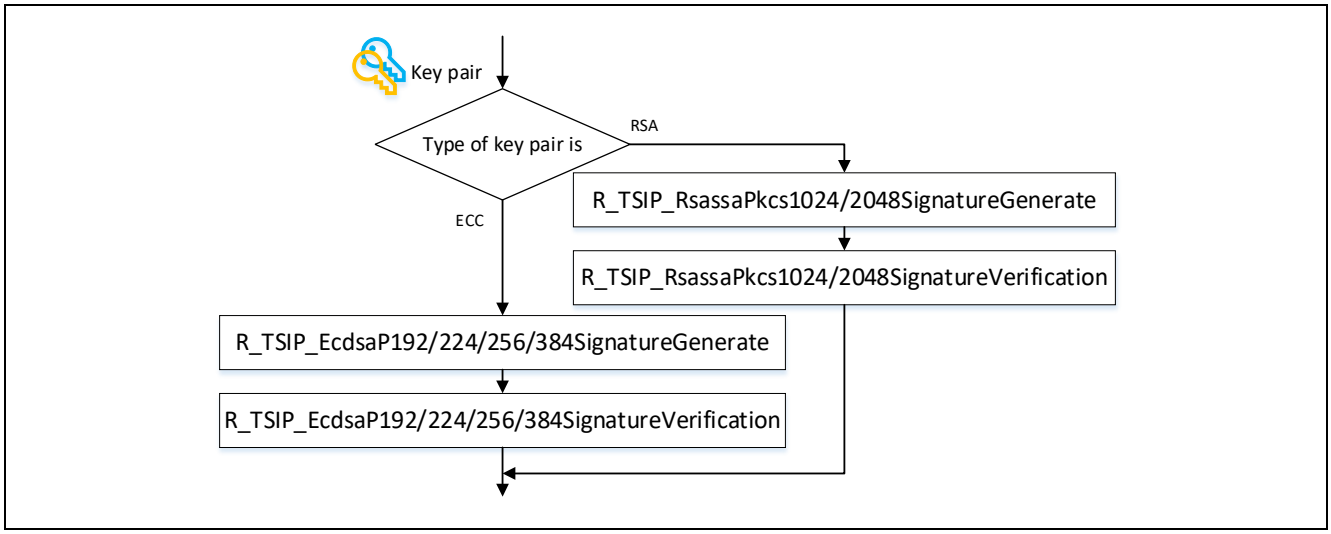


図 2-8　クライアント証明書の検証のための署名生成の流れ

表 2-5　クライアント証明書の検証で使用する API 関数

API 関数	説明
R_TSIP_RsassaPkcs1024/2048SignatureGenerate	RSA の秘密鍵を用いて、RSASSA-PKCS1-v1_5 の署名を生成します。
R_TSIP_RsassaPkcs1024/2048SignatureVerification	RSA の公開鍵を用いて、RSASSA-PKCS1-v1_5 の署名を検証します。
R_TSIP_EcdsaP192/224/256/384SignatureGenerate	ECC の秘密鍵を用いて、ECDSA の署名を生成します。
R_TSIP_EcdsaP192/224/256/384SignatureVerification	ECC の公開鍵を用いて、ECDSA の署名を検証します。

2.3.4　Finished

以下の手順に従い Finished メッセージの作成と検証を行います。処理の流れは図 2-9 を、使用する TSIP ドライバの API の詳細は表 2-6、表 2-7、 表 2-8、表 2-9、表 2-10 を参照してください。

1. R_TSIP_TlsGenerateMasterSecret ()関数で premaster secret から master secret を生成します。
2. R_TSIP_TlsGenerateSessionKey()関数で master secret から 4 本の session key (client write MAC key, server write MAC key, client write encryption key, server write encryption key) と 2 つの IV (client write IV, server write IV) を生成します。
3. R_TSIP_TlsGenerateVerifyData()関数でクライアントから送信する Finished メッセージ内の Verify Data を生成します。
4. Cipher Suite に対応したハッシュ関数と AES 関数を使用して、Finished メッセージの署名の生成と暗号化を行います。
5. クライアントからサーバに Finished メッセージを送信します。
6. サーバから Finished メッセージを受信します。
7. Cipher Suite に対応したハッシュ関数と AES 関数を使用して、Finished メッセージの復号と署名の検証を行います。
8. R_TSIP_TlsGenerateVerifyData()関数で Verify Data を検証し、Handshake Protocol を終了します。

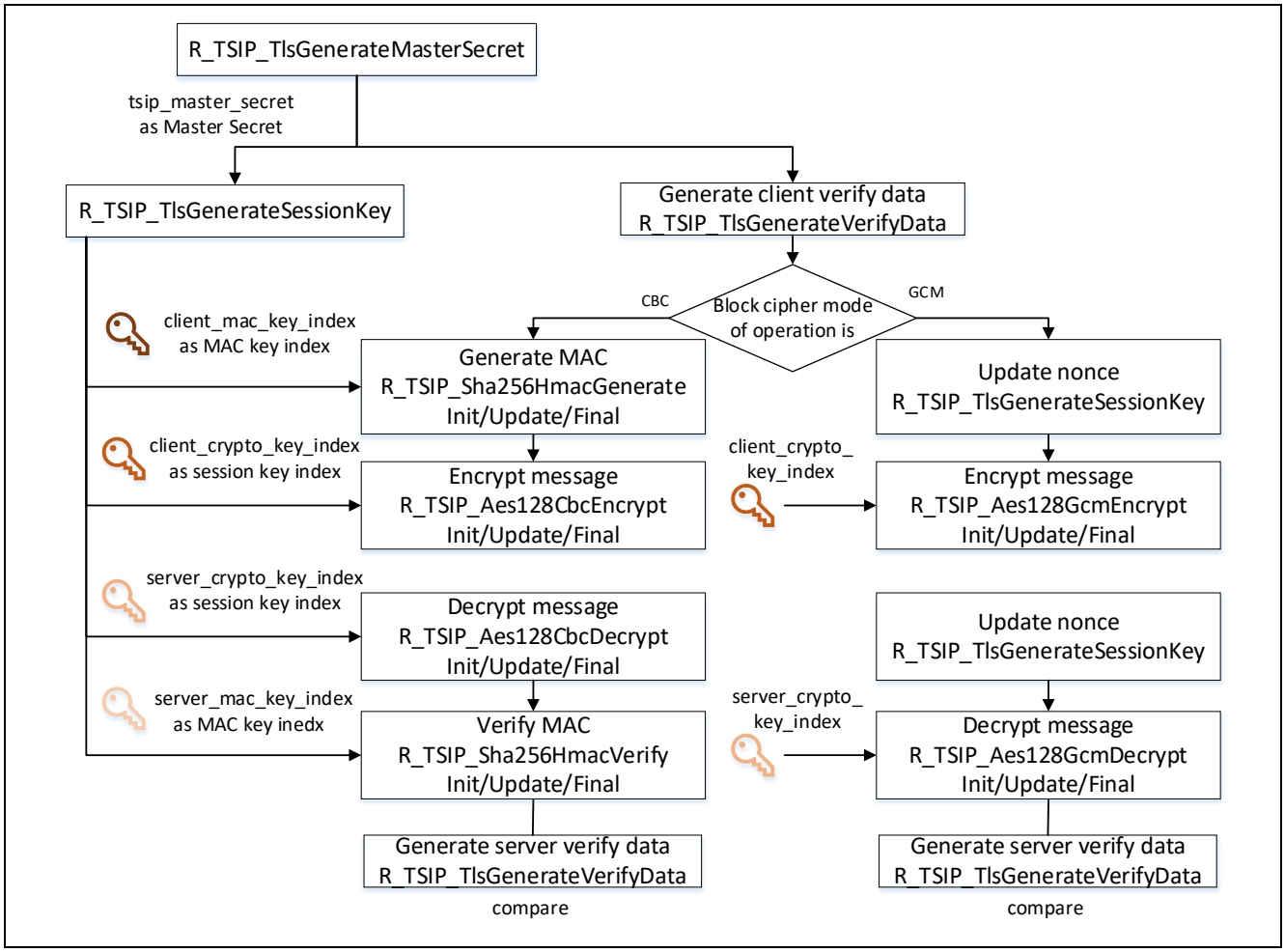


図 2-9　Finished メッセージの作成と検証の流れ

表 2-6 master secret の生成と Finished メッセージの生成・検証で使用する API 関数

API 関数	説明
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)	premaster secret を元に master secret を生成します。

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)	master secret を元に session key の key_index (client_mac_key_index, server_mac_key_index, client_crypto_key_index, server_crypto_key_index)を出力します。 IV は client_crypto_key_index と server_crypto_key_index に含まれています。 CBC モードを使用する場合、nonce_explicit には NULL を入力します。 GCM モードを使用する場合はノンスを入力します。
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)	Finished メッセージ用の VerifyData を生成します。

表 2-7 CBC モードでの暗号化に使用する API 関数

API 関数	説明
R_TSIP_Sha1HmacGenerateInit/Update/Final R_TSIP_Sha256HmacGenerateInit/Update/Final	client_mac_key_index を使用して、サーバに送信するデータの MAC 値を生成します。
R_TSIP_Aes128CbcEncryptInit/Update/Final R_TSIP_Aes256CbcEncryptInit/Update/Final	client_crypto_key_index を使用して、サーバに送信するデータを暗号化します。

表 2-8 CBC モードでの復号に使用する API 関数

API 関数	説明
R_TSIP_Aes128CbcDecryptInit/Update/Final R_TSIP_Aes256CbcDecryptInit/Update/Final	server_crypto_key_index を使用して、サーバから受信した暗号文を復号します。
R_TSIP_Sha1HmacVerifyInit/Update/Final R_TSIP_Sha256HmacVerifyInit/Update/Final	server_mac_key_index を使用して、サーバから受信した復号済みのデータの MAC 値を検証します。

表 2-9 GCM モードでの暗号化に使用する API 関数

API 関数	説明
R_TSIP_TlsGenerateSessionKey	TSIP 上のノンスを更新します。nonce_explicit には 1 パケットごとに異なるノンスを入力します。
R_TSIP_Aes128GcmEncryptInit/Update/Final	client_crypto_key_index を使用して、サーバに送信するデータの暗号化と認証タグの生成を行います。Ivec には NULL を、ivec_len には 0 を入力します。

表 2-10 GCM モードでの復号に使用する API 関数

API 関数	説明
R_TSIP_TlsGenerateSessionKey	TSIP 上のノンスを更新します。nonce_explicit にはパケットに含まれているノンスを入力します。
R_TSIP_Aes128GcmDecryptInit/Update/Final	server_crypto_key_index を使用して、サーバから受信したデータの復号と認証タグの検証を行います。

2.4　Application Data Protocol

Application Data Protocol では、Handshake Protocol の Finished メッセージと同様に TSIP ドライバの API を用いて暗号化・復号処理をして暗号通信を行います。使用する API は CBC モードの場合は表 2-7 と表 2-8、GCM モードの場合は表 2-9 と表 2-10 を参照してください。

3. サンプルコード

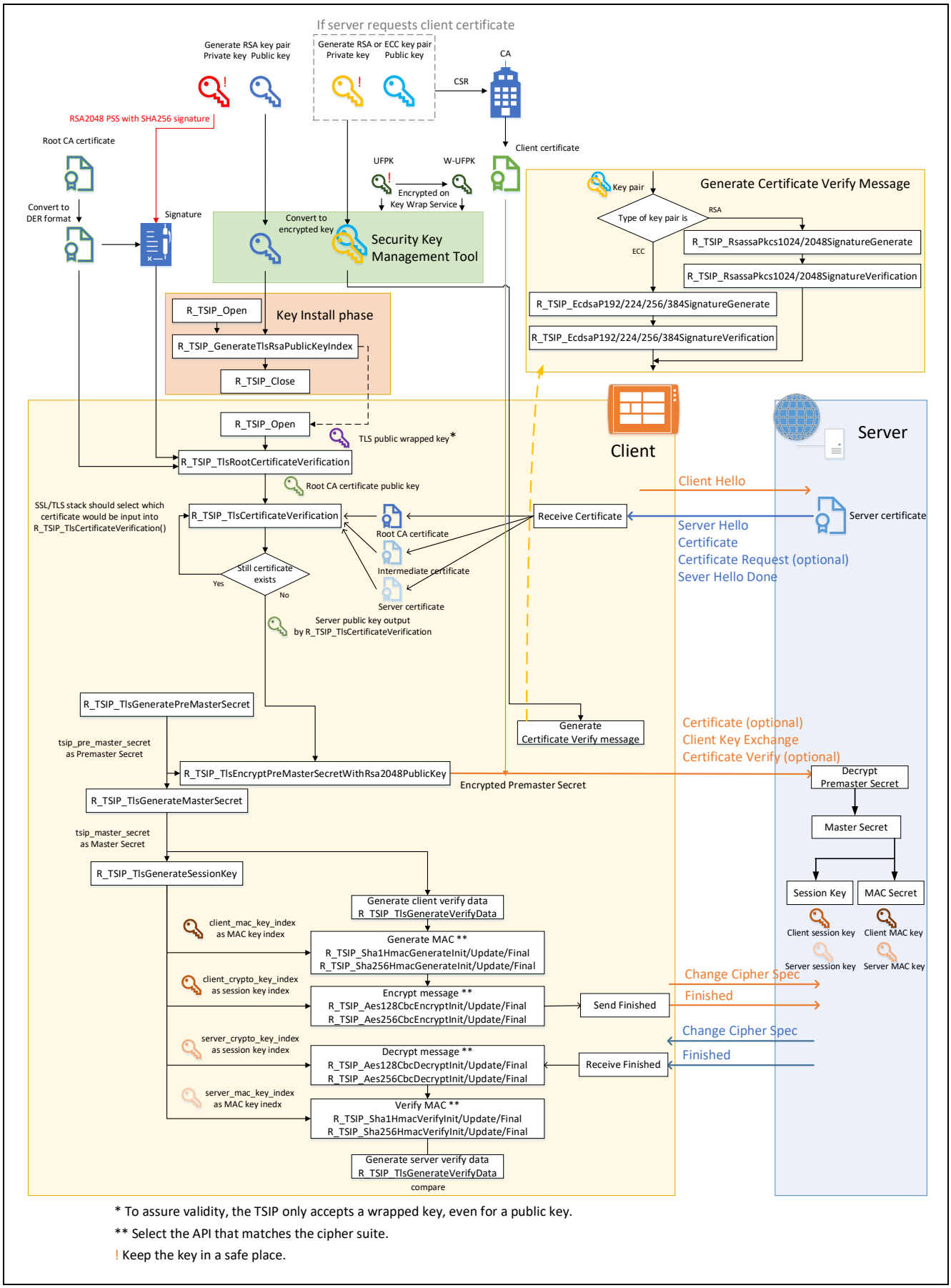
サンプルコードは以下リンク先の **iot-reference-rx** 最新 Release の Getting Started Guide に記載されているリンクを参照してください。

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

4.　付録

4.1　TLS ネゴシエーションフローにおける TSIP ドライバの呼び出しフロー

本節では TLS ネゴシエーションフロー全体における TSIP ドライバの呼び出しフローをまとめます。鍵交換方式が RSA の場合を図 4-1 に、ECDHE の場合を図 4-2 に示します。

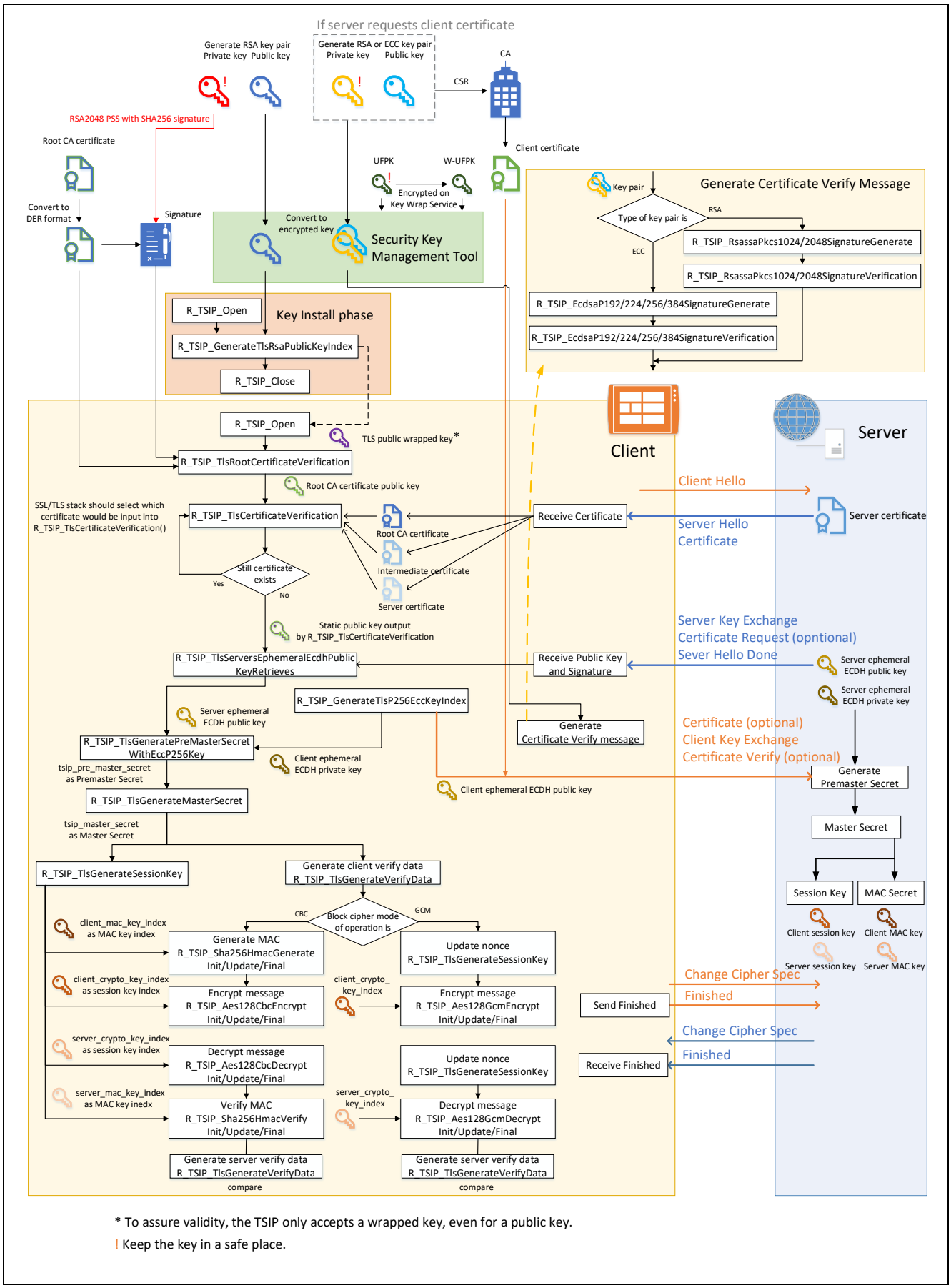


* To assure validity, the TSIP only accepts a wrapped key, even for a public key.

** Select the API that matches the cipher suite.

! Keep the key in a safe place.

図 4-1　TLS ネゴシエーションフローと TSIP ドライバの対応(鍵交換方式が RSA の場合)



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

! Keep the key in a safe place.

図 4-2　TLS ネゴシエーションフローと TSIP ドライバの対応(鍵交換方式が ECDHE の場合)

ホームページとサポート窓口

ルネサス エレクトロニクスホームページ
https://www.renesas.com/jp/ja/

お問合せ先
https://www.renesas.com/jp/ja/support/contact.html

改訂記録

Rev.	発行日	改訂内容	
		ページ	ポイント
1.00	Jun.30.21	—	初版発行
1.01	Mar.31.22	—	記載の修正
1.02	Sep.15.22	14	2.3.3 Certificate Verify 生成の説明追記
1.03	Jun.28.24	—	サンプルプログラムに関する記載の差し替え
1.04	Apr.10.25	—	用語に関する変更を適用

製品ご使用上の注意事項

ここでは、マイコン製品全体に適用する「使用上の注意事項」について説明します。個別の使用上の注意事項については、本ドキュメントおよびテクニカルアップデートを参照してください。

1. 静電気対策
   CMOS 製品の取り扱いの際は静電気防止を心がけてください。CMOS 製品は強い静電気によってゲート絶縁破壊を生じることがあります。運搬や保存の際には、当社が出荷梱包に使用している導電性のトレーやマガジンケース、導電性の緩衝材、金属ケースなどを利用し、組み立て工程にはアースを施してください。プラスチック板上に放置したり、端子を触ったりしないでください。また、CMOS 製品を実装したボードについても同様の扱いをしてください。
2. 電源投入時の処置
   電源投入時は、製品の状態は不定です。電源投入時には、LSI の内部回路の状態は不確定であり、レジスタの設定や各端子の状態は不定です。外部リセット端子でリセットする製品の場合、電源投入からリセットが有効になるまでの期間、端子の状態は保証できません。同様に、内蔵パワーオンリセット機能を使用してリセットする製品の場合、電源投入からリセットのかかる一定電圧に達するまでの期間、端子の状態は保証できません。
3. 電源オフ時における入力信号
   当該製品の電源がオフ状態のときに、入力信号や入出力プルアップ電源を入れないでください。入力信号や入出力プルアップ電源からの電流注入により、誤動作を引き起こしたり、異常電流が流れ内部素子を劣化させたりする場合があります。資料中に「電源オフ時における入力信号」についての記載のある製品は、その内容を守ってください。
4. 未使用端子の処理
   未使用端子は、「未使用端子の処理」に従って処理してください。CMOS 製品の入力端子のインピーダンスは、一般に、ハイインピーダンスとなっています。未使用端子を開放状態で動作させると、誘導現象により、LSI 周辺のノイズが印加され、LSI 内部で貫通電流が流れたり、入力信号と認識されて誤動作を起こす恐れがあります。
5. クロックについて
   リセット時は、クロックが安定した後、リセットを解除してください。プログラム実行中のクロック切り替え時は、切り替え先クロックが安定した後に切り替えてください。リセット時、外部発振子（または外部発振回路）を用いたクロックで動作を開始するシステムでは、クロックが十分安定した後、リセットを解除してください。また、プログラムの途中で外部発振子（または外部発振回路）を用いたクロックに切り替える場合は、切り替え先のクロックが十分安定してから切り替えてください。
6. 入力端子の印加波形
   入力ノイズや反射波による波形歪みは誤動作の原因になりますので注意してください。CMOS 製品の入力がノイズなどに起因して、V_{IL}（Max.）から V_{IH}（Min.）までの領域にとどまるような場合は、誤動作を引き起こす恐れがあります。入力レベルが固定の場合はもちろん、V_{IL}（Max.）から V_{IH}（Min.）までの領域を通過する遷移期間中にチャタリングノイズなどが入らないように使用してください。
7. リザーブアドレス（予約領域）のアクセス禁止
   リザーブアドレス（予約領域）のアクセスを禁止します。アドレス領域には、将来の拡張機能用に割り付けられている リザーブアドレス（予約領域）があります。これらのアドレスをアクセスしたときの動作については、保証できませんので、アクセスしないようにしてください。
8. 製品間の相違について
   型名の異なる製品に変更する場合は、製品型名ごとにシステム評価試験を実施してください。同じグループのマイコンでも型名が違うと、フラッシュメモリ、レイアウトパターンの相違などにより、電気的特性の範囲で、特性値、動作マージン、ノイズ耐量、ノイズ幅射量などが異なる場合があります。型名が違う製品に変更する場合は、個々の製品ごとにシステム評価試験を実施してください。

ご注意書き

1. 本資料に記載された回路、ソフトウェアおよびこれらに関連する情報は、半導体製品の動作例、応用例を説明するものです。回路、ソフトウェアおよびこれらに関連する情報を使用する場合、お客様の責任において、お客様の機器・システムを設計ください。これらの使用に起因して生じた損害（お客様または第三者いずれに生じた損害も含みます。以下同じです。）に関し、当社は、一切その責任を負いません。
2. 当社製品または本資料に記載された製品データ、図、表、プログラム、アルゴリズム、応用回路例等の情報の使用に起因して発生した第三者の特許権、著作権その他の知的財産権に対する侵害またはこれらに関する紛争について、当社は、何らの保証を行うものではなく、また責任を負うものではありません。
3. 当社は、本資料に基づき当社または第三者の特許権、著作権その他の知的財産権を何ら許諾するものではありません。
4. 当社製品を組み込んだ製品の輸出入、製造、販売、利用、配布その他の行為を行うにあたり、第三者保有の技術の利用に関するライセンスが必要となる場合、当該ライセンス取得の判断および取得はお客様の責任において行ってください。
5. 当社製品を、全部または一部を問わず、改造、改変、複製、リバースエンジニアリング、その他、不適切に使用しないでください。かかる改造、改変、複製、リバースエンジニアリング等により生じた損害に関し、当社は、一切その責任を負いません。
6. 当社は、当社製品の品質水準を「標準水準」および「高品質水準」に分類しており、各品質水準は、以下に示す用途に製品が使用されることを意図しております。
   標準水準：　コンピュータ、OA 機器、通信機器、計測機器、AV 機器、家電、工作機械、パーソナル機器、産業用ロボット等
   高品質水準：輸送機器（自動車、電車、船舶等）、交通制御（信号）、大規模通信機器、金融端末基幹システム、各種安全制御装置等
   当社製品は、データシート等により高信頼性、Harsh environment 向け製品と定義しているものを除き、直接生命・身体に危害を及ぼす可能性のある機器・システム（生命維持装置、人体に埋め込み使用するもの等）、もしくは多大な物的損害を発生させるおそれのある機器・システム（宇宙機器と、海底中継器、原子力制御システム、航空機制御システム、プラント基幹システム、軍事機器等）に使用されることを意図しておらず、これらの用途に使用することは想定していません。たとえ、当社が想定していない用途に当社製品を使用したことにより損害が生じても、当社は一切その責任を負いません。
7. あらゆる半導体製品は、外部攻撃からの安全性を 100%保証されているわけではありません。当社ハードウェア／ソフトウェア製品にはセキュリティ対策が組み込まれているものもありますが、これによって、当社は、セキュリティ脆弱性または侵害（当社製品または当社製品が使用されているシステムに対する不正アクセス・不正使用を含みますが、これに限りません。）から生じる責任を負うものではありません。当社は、当社製品または当社製品が使用されたあらゆるシステムが、不正な改変、攻撃、ウイルス、干渉、ハッキング、データの破壊または窃盗その他の不正な侵入行為（「脆弱性問題」といいます。）によって影響を受けないことを保証しません。当社は、脆弱性問題に起因しまたはこれに関連して生じた損害について、一切責任を負いません。また、法令において認められる限りにおいて、本資料および当社ハードウェア／ソフトウェア製品について、商品性および特定目的との合致に関する保証ならびに第三者の権利を侵害しないことの保証を含め、明示または黙示のいかなる保証も行いません。
8. 当社製品をご使用の際は、最新の製品情報（データシート、ユーザーズマニュアル、アプリケーションノート、信頼性ハンドブックに記載の「半導体デバイスの使用上の一般的な注意事項」等）をご確認の上、当社が指定する最大定格、動作電源電圧範囲、放熱特性、実装条件その他指定条件の範囲内でご使用ください。指定条件の範囲を超えて当社製品をご使用された場合の故障、誤動作の不具合および事故につきましては、当社は、一切その責任を負いません。
9. 当社は、当社製品の品質および信頼性の向上に努めていますが、半導体製品はある確率で故障が発生したり、使用条件によっては誤動作したりする場合があります。また、当社製品は、データシート等において高信頼性、Harsh environment 向け製品と定義しているものを除き、耐放射線設計を行っておりません。仮に当社製品の故障または誤動作が生じた場合であっても、人身事故、火災事故その他社会的損害等を生じさせないよう、お客様の責任において、冗長設計、延焼対策設計、誤動作防止設計等の安全設計およびエージング処理等、お客様の機器・システムとしての出荷保証を行ってください。特に、マイコンソフトウェアは、単独での検証は困難なため、お客様の機器・システムとしての安全検証をお客様の責任で行ってください。
10. 当社製品の環境適合性等の詳細につきましては、製品個別に必ず当社営業窓口までお問合せください。ご使用に際しては、特定の物質の含有・使用を規制する RoHS 指令等、適用される環境関連法令を十分調査のうえ、かかる法令に適合するようご使用ください。かかる法令を遵守しないことにより生じた損害に関して、当社は、一切その責任を負いません。
11. 当社製品および技術を国内外の法令および規則により製造・使用・販売を禁止されている機器・システムに使用することはできません。当社製品および技術を輸出、販売または移転等する場合は、「外国為替及び外国貿易法」その他日本国および適用される外国の輸出管理関連法規を遵守し、それらの定めるところに従い必要な手続きを行ってください。
12. お客様が当社製品を第三者に転売等される場合には、事前に当該第三者に対して、本ご注意書き記載の諸条件を通知する責任を負うものといたします。
13. 本資料の全部または一部を当社の文書による事前の承諾を得ることなく転載または複製することを禁じます。
14. 本資料に記載されている内容または当社製品についてご不明な点がございましたら、当社の営業担当者までお問合せください。

注 1. 本資料において使用されている「当社」とは、ルネサス エレクトロニクス株式会社およびルネサス エレクトロニクス株式会社が直接的、間接的に支配する会社をいいます。

注 2. 本資料において使用されている「当社製品」とは、注 1 において定義された当社の開発、製造製品をいいます。

(Rev.5.0-1　2020.10)

本社所在地

〒135-0061 東京都江東区豊洲 3-2-24（豊洲フォレシア）

www.renesas.com

お問合せ窓口

弊社の製品や技術、ドキュメントの最新情報、最寄の営業お問合せ窓口に関する情報などは、弊社ウェブサイトをご覧ください。

www.renesas.com/contact/

商標について

ルネサスおよびルネサスロゴはルネサス エレクトロニクス株式会社の商標です。すべての商標および登録商標は、それぞれの所有者に帰属します。