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。本書には、FreeRTOS をベースとしたサンプルプロジェクトを添付しています。このサンプルプロジェクトは MbedTLS を含む FreeRTOS に TSIP ドライバを組み込んでおり、AWS との MQTT 通信をテストすることができます。

動作確認デバイス

本書に付属するサンプルプログラムは以下のデバイスで動作確認しています。

- RX72N: R5F572NDHDFB

動作環境

本書に付属するサンプルプログラムは以下の環境で動作確認しています。

IDE	e^{2}studio 2021-04
ツールチェーン	CCRX コンパイラ v3.0.3 GCC for Renesas 8.3.0.202002-GNURX
ターゲットボード	RX72N Envision Kit (製品型名: RTK5RX72N0C00000BJ)
デバッガ	E2 Lite エミュレータ (RX72N Envision Kit オンボード)
TSIP ドライバ	バージョン 1.11
Tera Term	バージョン 4.105
OpenSSL	1.1.1f

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(クライアント側のみ)。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	ユーザがデバイス上で暗号機能への入力として使う鍵。ユーザが生成する。
encrypted key	provisioning key 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。Renesas Secure Flash Programmer が生成する。
鍵生成情報 (key index)	user key などを TSIP ドライバで使用可能な形式に変換したデータ。 TSIP が生成する。
provisioning key	user key から encrypted key を生成するために必要な鍵。ユーザが生成する。
encrypted provisioning key	TSIP で encrypted key を復号し、key index に変換するための鍵情報。 DLM サーバが生成する。
Hidden 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）のみに存在する鍵。
DLM サーバ (https://dlm.renesas.com/)	Renesas 鍵管理サーバ。「DLM サーバ」は「Device Lifecycle Management サーバ」の略。provisioning key をラップ（暗号化）するのに使用する。

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。

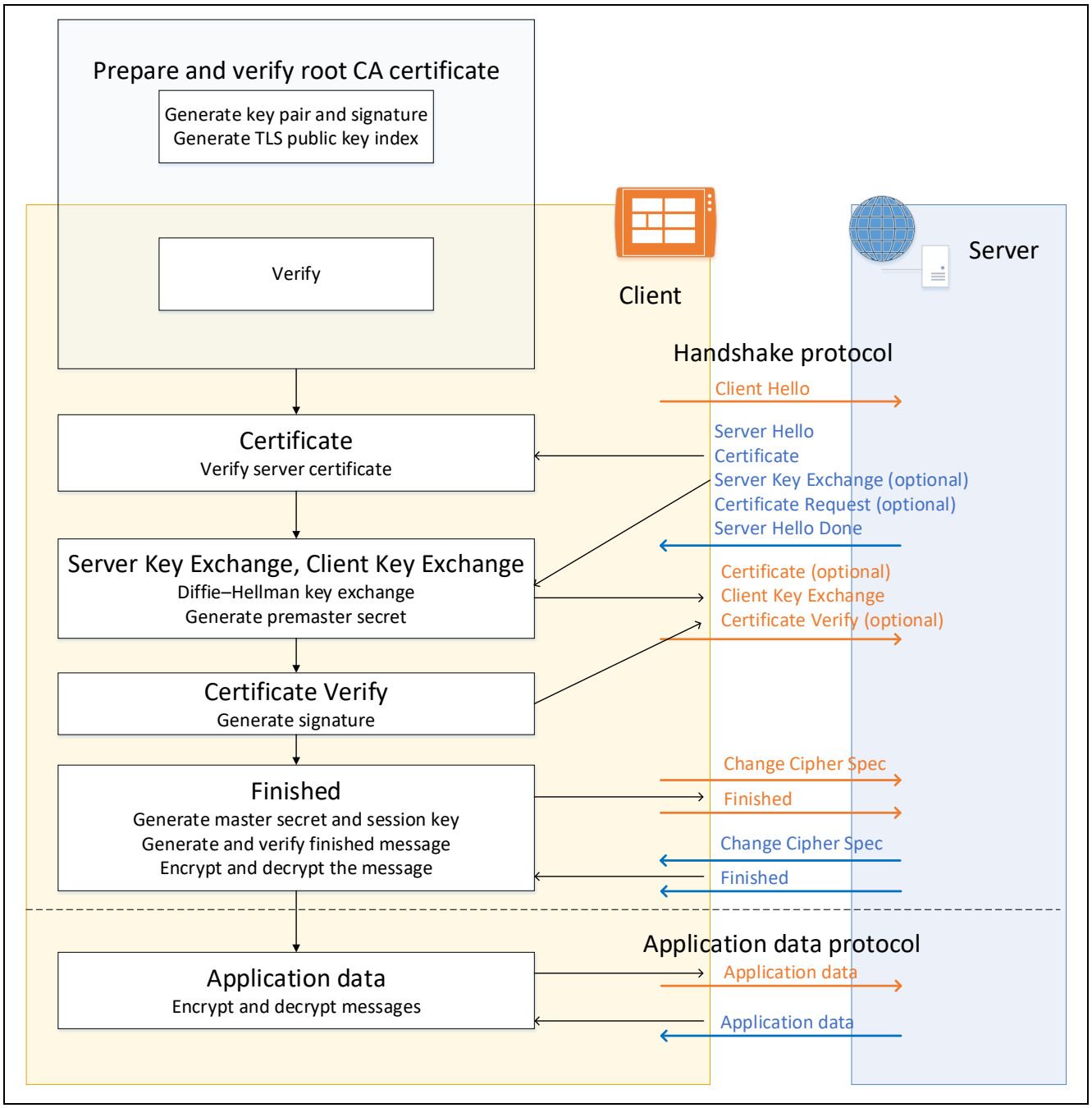


図 2-1　TLS 通信の流れと TSIP ドライバを使用する処理の概略

図 2-1 の TSIP ドライバ使用部分の詳細は、「6.2 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 6-1 および図 6-2 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形式に変換してプログラムに組み込む必要があります。TSIP ドライバで使用するユーザ鍵をラップする手順については、3.2.5 節で解説しています。

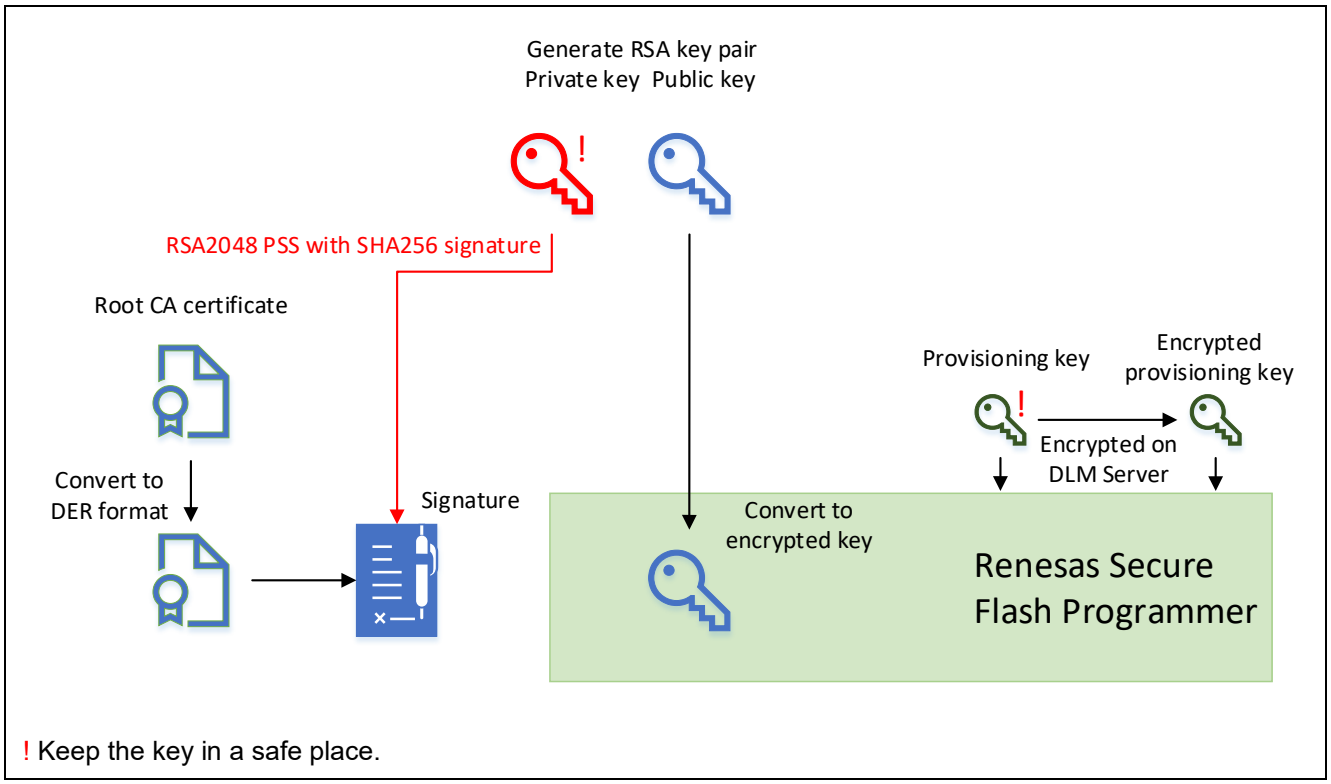


図 2-2　ルート CA 証明書の準備の流れ

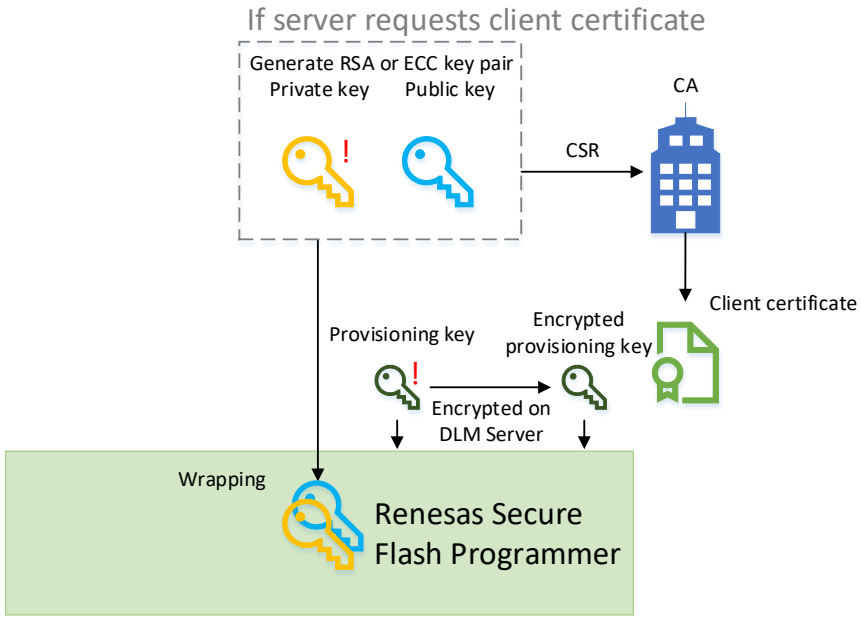
2.1.2　クライアント証明書の準備

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形式に変換してプログラムに組み込む必要があります。TSIP ドライバで使用するユーザ鍵をラップする手順については、3.2.5 節で解説しています。



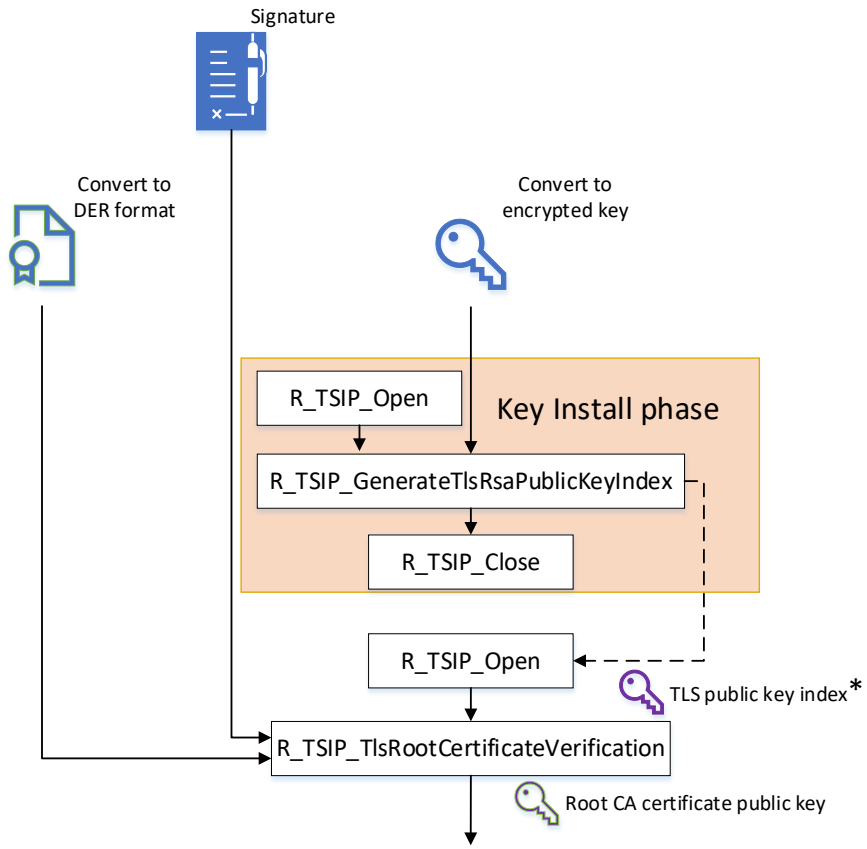
! Keep the key in a safe place.

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 key index, 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。

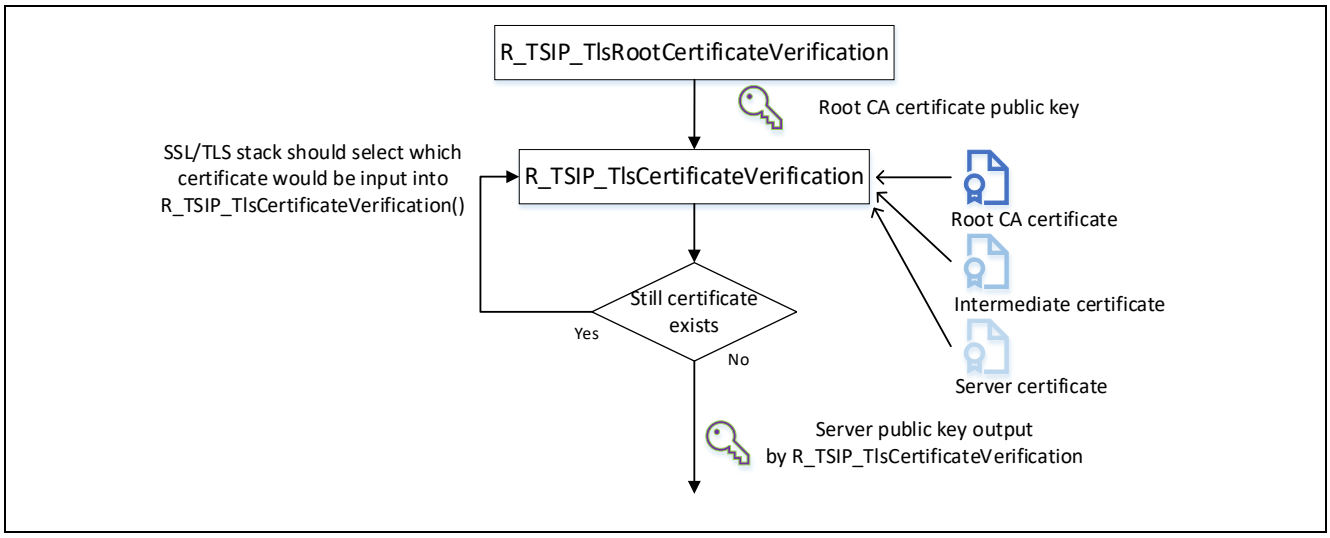


図 2-5　サーバ証明書の検証の流れ

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,	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**

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)	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。

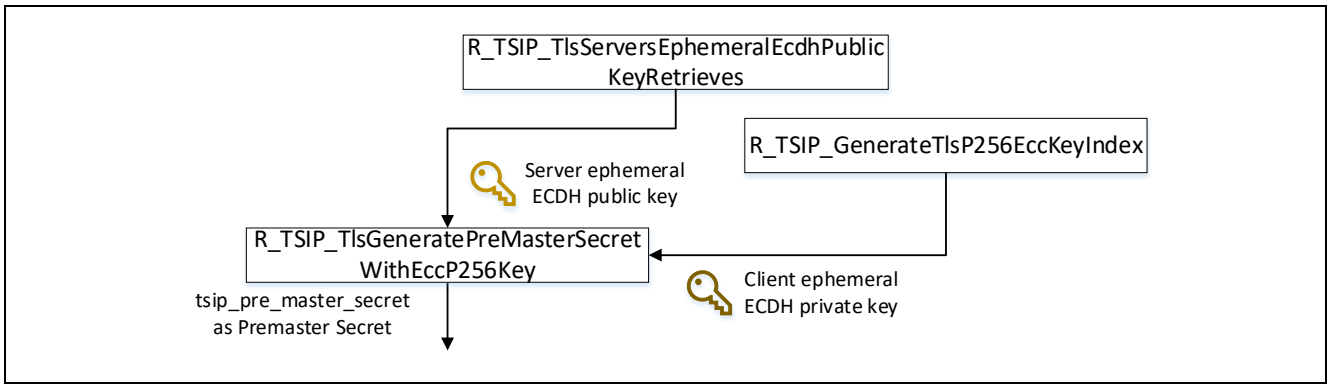


図 2-6　ECDHE による鍵交換の流れ

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, 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,	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_TlsGeneratePreMasterSecretWithEccP256Key()で使用する暗号化された ephemeral ECDH public key です。

uint32_t *encrypted_ephemeral_ecdh_public_key)	
e_tsip_err_t R_TSIP_GenerateTlsP256EccKeyIndex (uint32_t 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。

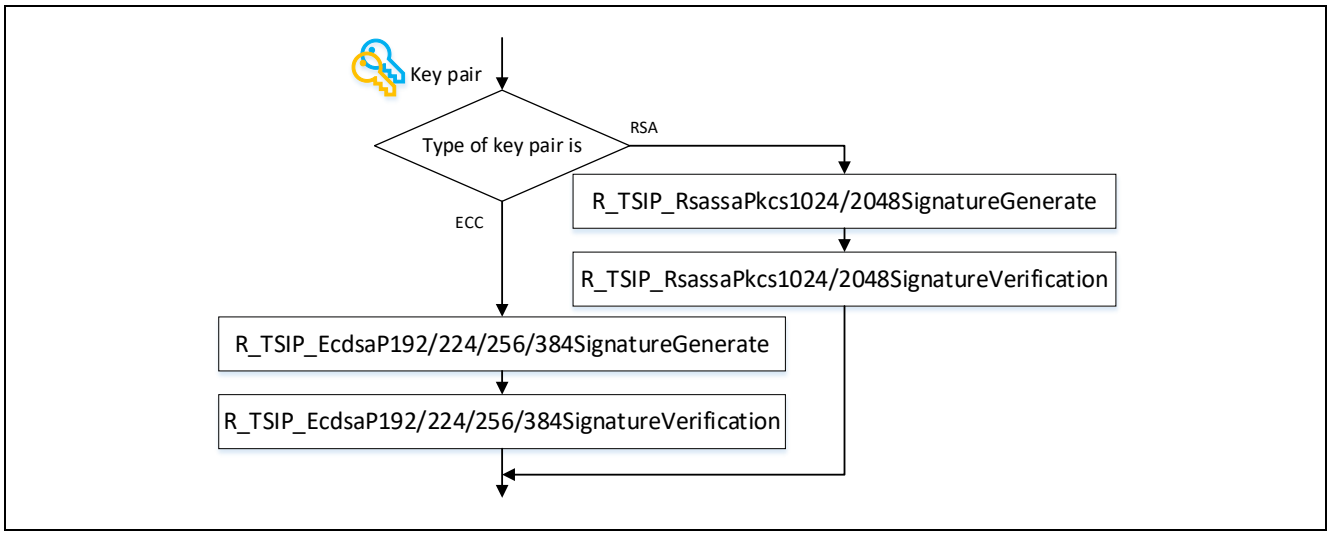


図 2-8　クライアント証明書の検証のための署名生成の流れ

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。

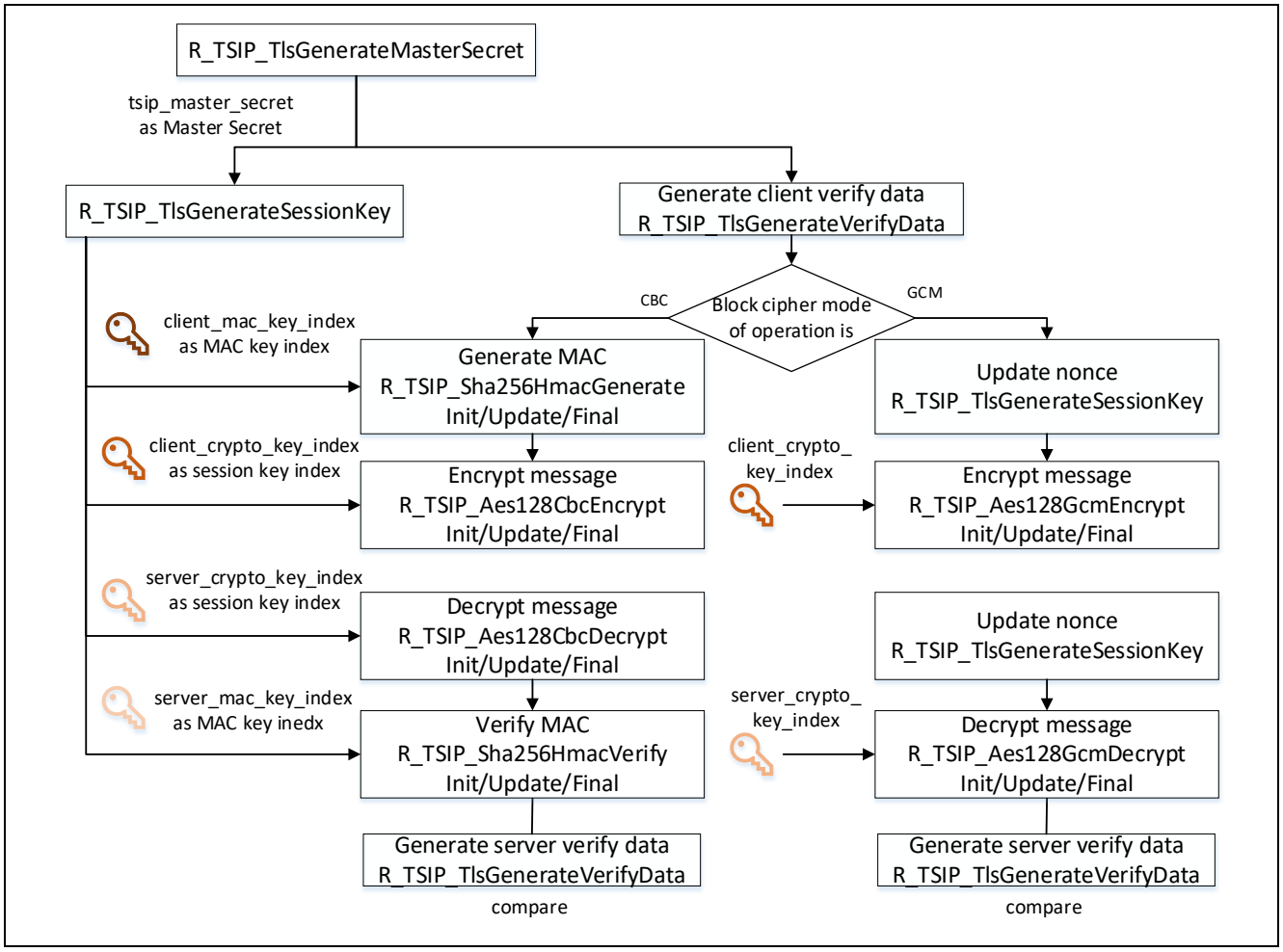


図 2-9　Finished メッセージの作成と検証の流れ

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.　サンプルプロジェクト

本サンプルプロジェクトは、RX72N Envision Kit を用いて AWS と TLS 接続し、MQTT 通信を行うデモです。

以下に RX72N Envision Kit で本サンプルプロジェクトを実行する際の接続情報を示します。デバッグとシリアル通信のため、RX72N Envision Kit と PC を 2 本の USB ケーブルで接続してください。また、インターネットに接続するために RX72N Envision Kit とルータを Ethernet ケーブルで接続してください。

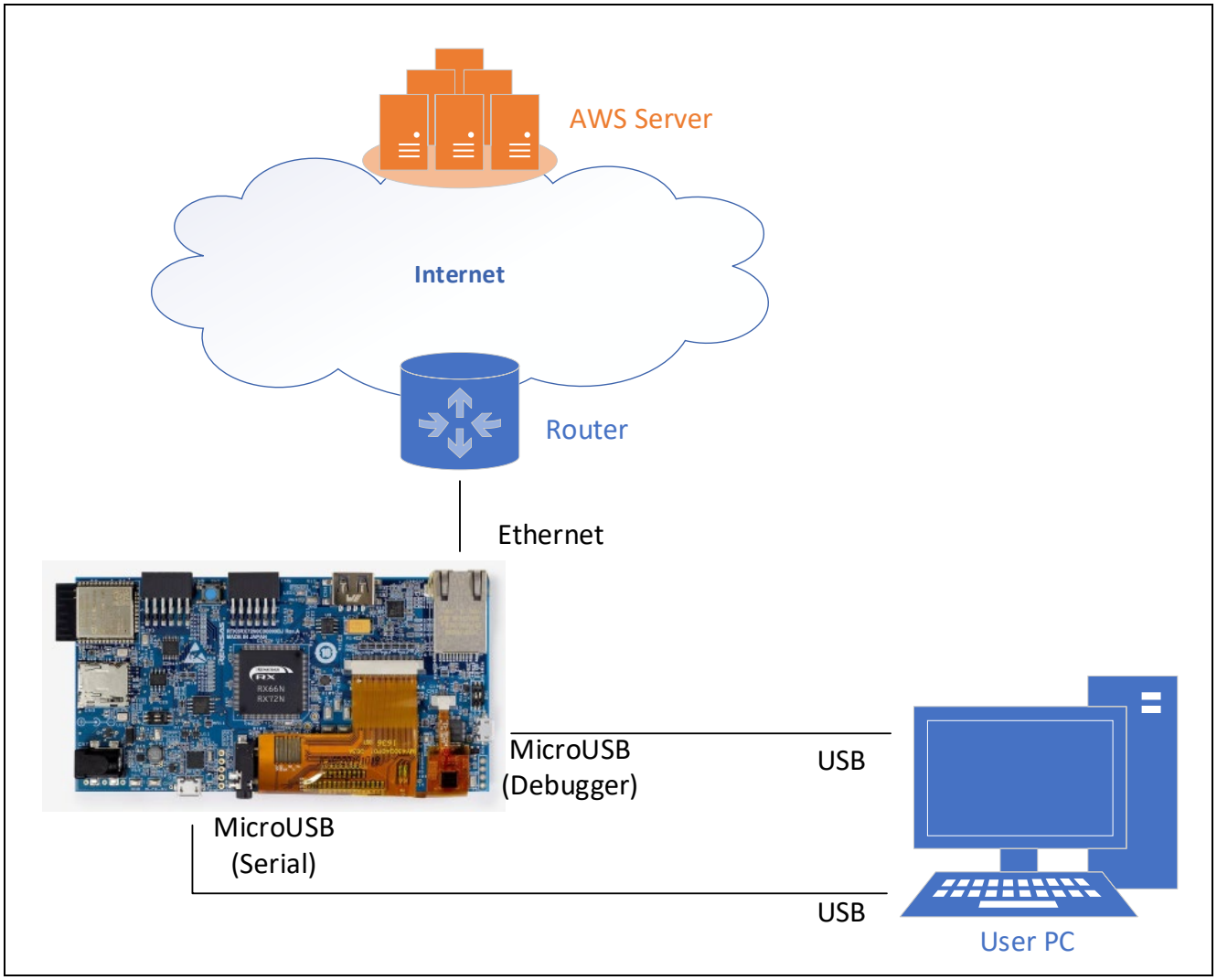


図 3-1　サンプルプロジェクトの接続関係

サンプルプロジェクトは FreeRTOS のプロジェクトをベースにしています。FreeRTOS は IoT 機器向けに必要なソースコードをまとめた IoT Libraries を提供しており、この中で暗号ライブラリとしてオープンソースの Mbed TLS を使用しています。本プロジェクトでは Mbed TLS ライブラリの一部の処理を TSIP ドライバの TLS 向け API で置き換えています。以下にサンプルプロジェクトのソフトウェア構造を示します。

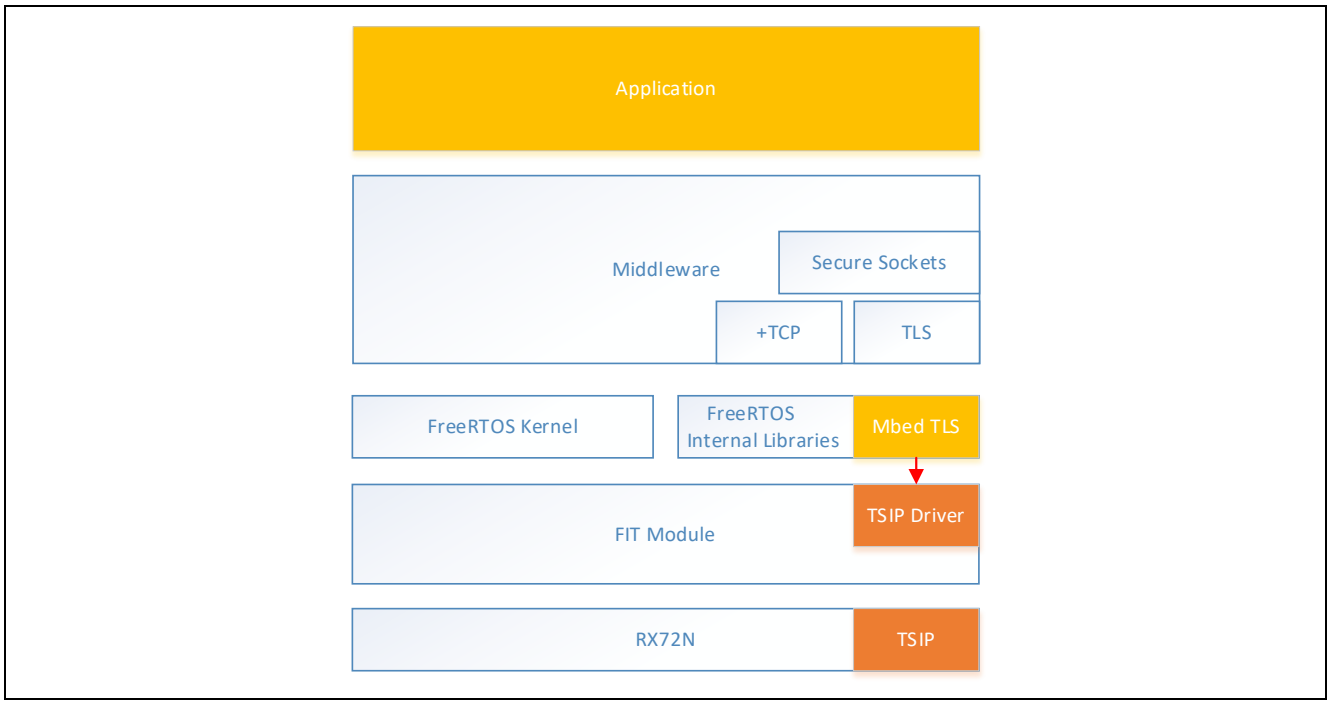


図 3-2　サンプルプロジェクトのソフトウェア構造

サンプルプロジェクトは、以下のリポジトリにある RX 向けの FreeRTOS プロジェクトをベースにしています。

https://github.com/renesas/amazon-freertos/releases/tag/v202002.00-rx-1.0.5

本章の説明では、以下のツールを使用しますので予めご用意ください。

- シェルスクリプト (bash) の実行環境
- OpenSSL
- Renesas Secure Flash Programmer

3.1　フォルダ構成

サンプルプロジェクトのフォルダ構成を以下に示します。**太字**はベースプロジェクトから修正したファイルの所在です。同一フォルダ内に多数の差分がある場合は代表的なフォルダのみを示しています。詳細は diff ツール等を使用して確認してください。

```
amazon-freertos
|--demos
|  |--dev_mode_key_provisioning/src/aws_dev_mode_key_provisioning.c
|--doc
|--freertos_kernel
|--libraries
|  |--3rdparty/mbedtls
|  |--abstractions/pkcs11/mbedtls/iot_pkcs11_mbedtls.c
|  |--freertos_plus/standard/tls
|--projects
|--tests
|--tools
|--vendors/renesas
  |--boards/rx_mcu_boards/(board name)/aws_demos/src/smc_gen
  |  |--general
  |  |--r_config/r_tsip_rx_config.h
  |--rx_driver_package/v125/r_tsip_rx
```

サンプルプロジェクトでは TSIP ドライバを使用するためにベースプロジェクトに以下のような改造をしています。

- プロジェクトに TSIP ドライバの FIT モジュールを追加
- FreeRTOS、Mbed TLS の一部の処理を TSIP ドライバの API で置き換え
- マルチタスク環境での TSIP へのアクセス衝突を回避するために、排他制御を追加
- 証明書とその署名を記述するファイル、設定用のファイルを新たに追加

3.2　鍵と証明書の準備

ここでは、サンプルプロジェクトで使用する鍵と証明書を入手し、TSIP ドライバで使用するために変換する方法について説明します。表 3-1 にサンプルプロジェクトで使用する鍵と証明書の入手方法の概要を示します。RSA のクライアント証明書を使用しない場合、3.2.3 節の操作は不要です。ECDSA のクライアント証明書を使用しない場合、3.2.3 節の操作は不要です。その後、3.2.4 節、3.2.5 節の内容に従って変換を行います。

表 3-1　サンプルプロジェクトで使用する鍵と証明書の入手方法

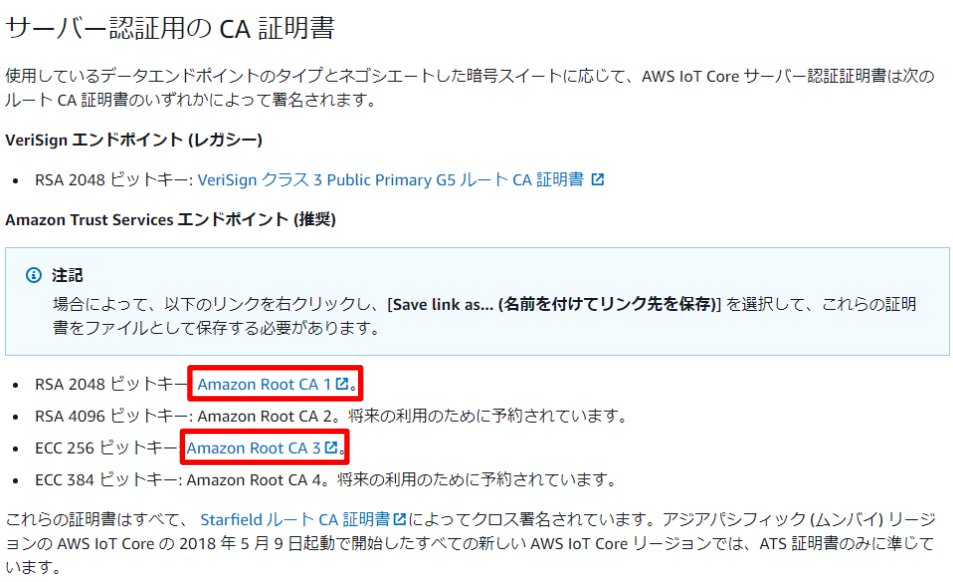
鍵/証明書	入手方法	節
RSA ルート CA 証明書	AWS からダウンロードする	3.2.1
ECDSA ルート CA 証明書	AWS からダウンロードする	3.2.1
RSA 鍵ペア	AWS 上で自動生成されたものをダウンロードする	3.2.2
RSA クライアント証明書	AWS からダウンロードする	3.2.2
ECC 鍵ペア	OpenSSL 等のツールを使って作成する	3.2.3.1
ECDSA クライアント証明書	OpenSSL 等のツールを使って CSR (Certificate Signing Request: 証明書署名要求) を作成し、AWS にアップロードする。その後、AWS から証明書をダウンロードする。	3.2.3.2
ルート CA 証明書の署名生成/署名検証用鍵ペア	OpenSSL 等のツールを使ってユーザが作成する	3.2.4.1 3.2.4.2

3.2.1　ルート CA 証明書の入手

以下の URL からルート CA 証明書を入手します。

https://docs.aws.amazon.com/iot/latest/developerguide/server-authentication.html#server-authentication-certs

RSA の証明書を使用する場合は「Amazon Root CA 1」を、ECDSA の証明書を使用する場合は「Amazon Root CA 3」をダウンロードしてください。



ダウンロードした証明書は、サンプルプロジェクトの key_crt_sig_generator フォルダの内部に以下のように配置してください。

```
key_crt_sig_generator
|-- ca
|   |-- AmazonRootCA1.pem
|   |-- AmazonRootCA3.pem
|-- ca-sign-keypair-rsa2048
|-- client-ecc256
|-- client-rsa2048
|-- output
|--1_rsa2048_convertCrt.sh
|--2_1_ecc256_generateKeyPair.sh
|--2_2_ecc256_convertCrt.sh
|--3_showkeyValues.sh
|--convertCrt.sh
```

3.2.2　RSA の鍵ペアとクライアント証明書の生成

RSA の鍵ペアとクライアント証明書は AWS サーバ上で自動生成することができます。アプリケーションノート「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)」の「1.1 コンソールへのサインイン」に従ってモノの登録をした上で、RSA-2048 のクライアント証明書、パブリックキー、およびプライベートキーを入手してください。

ダウンロードした証明書と鍵は、サンプルプロジェクトの key_crt_sig_generator フォルダに以下のように配置してください。

```
key_crt_sig_generator
|-- ca
|   |-- AmazonRootCA1.pem
|   |-- AmazonRootCA3.pem
|-- ca-sign-keypair-rsa2048
|-- client-ecc256
|-- client-rsa2048
|   |-- *-certificate.pem.crt
|   |-- *-private.pem.key
|   |-- *-public.pem.key
|-- output
|--1_rsa2048_convertCrt.sh
|--2_1_ecc256_generateKeyPair.sh
|--2_2_ecc256_convertCrt.sh
|--3_showkeyValues.sh
|--convertCrt.sh
```

3.2.3 ECDSA のクライアント証明書と鍵ペアの生成

ECDSA のクライアント証明書を AWS で生成するためには、ECC の鍵ペアと CSR（Certificate Signing Request: 証明書署名要求）を生成し、AWS にアップロードする必要があります。

3.2.3.1 ECC 鍵ペア生成

key_crt_sig_generator フォルダにある 2_1_ecc256_generateKeyPair.sh を実行してください。

以下は 2_1_ecc256_generateKeyPair.sh の内容です。

```
#!/bin/sh

# Create a key pair and CSR
openssl ecparam -genkey -name prime256v1 -out client-ecc256/prime256v1-
private.pem.key
openssl req -new -sha256 -key client-ecc256/prime256v1-private.pem.key -out
client-ecc256/prime256v1-csr.pem.csr
echo -e "¥nPlease upload ¥"prime256v1-csr.pem.csr¥" to AWS IoT Core and
download ¥"*-certificate.pem.crt¥"."
```

実行後、client-ecc256 フォルダに ECC の鍵ペアとそれに対する CSR が出力されます。

```
key_crt_sig_generator
|-- ca
|   |-- AmazonRootCA1.pem
|   |-- AmazonRootCA3.pem
|-- ca-sign-keypair-rsa2048
|-- client-ecc256
|   |-- prime256v1-csr.pem.csr
|   |-- prime256v1-private.pem.key
|-- client-rsa2048
|-- output
|--1_rsa2048_convertCrt.sh
|--2_1_ecc256_generateKeyPair.sh
|--2_2_ecc256_convertCrt.sh
|--3_showkeyValues.sh
|--convertCrt.sh
```

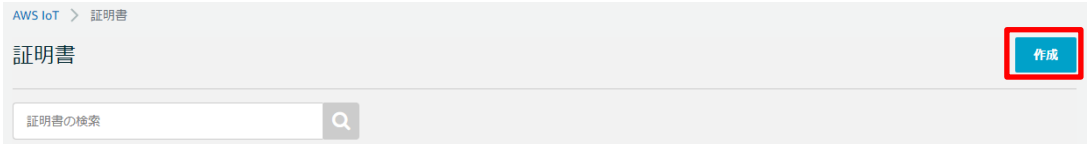
3.2.3.2　AWS への鍵の登録

出力された CSR を AWS IoT にアップロードします。
「3.2.2 RSA の鍵ペアとクライアント証明書の生成」の操作を行っていない場合、最初に AWS 上でモノの作成を行う必要があります。アプリケーションノート「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)」の「1.1 コンソールへのサインイン」に従ってモノを登録してください。この際、以下の「モノに証明書を追加」の画面では「証明書なしでモノを作成」を選択してください。

AWS マネジメントコンソール (https://aws.amazon.com/console/)にサインインし、「すべてのサービス」→「IoT」→「IoT コア」をクリックしてください。

左端のメニューから「安全性」→「証明書」を選択します。

画面右上の「作成」をクリックします。



「CSR による作成」をクリックします。

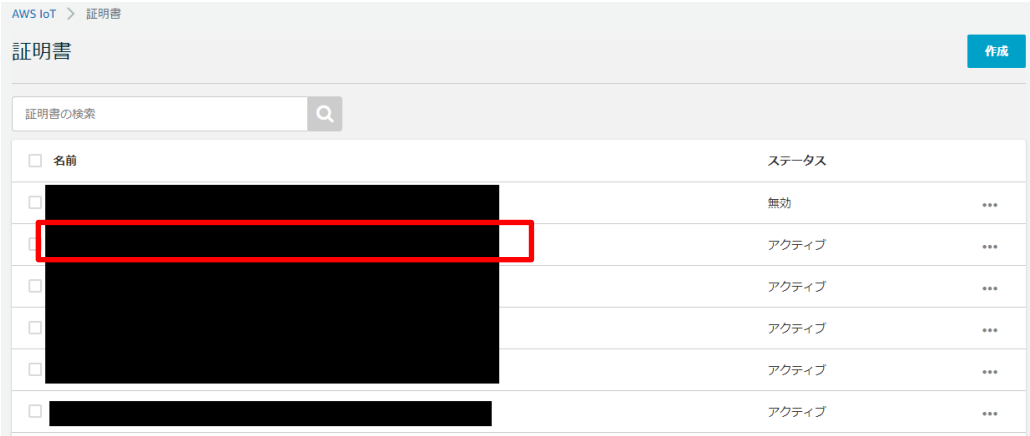
ファイルオープンのダイアログが表示されるので、3.2.3.1 で作成した prime256v1-csr.pem.csr を選択します。その後、「ファイルのアップロード」をクリックします。

AWS IoT が ECC 鍵ペアに対する証明書を発行します。「ダウンロード」、「有効化」、「ポリシーにアタッチ」をクリックします。

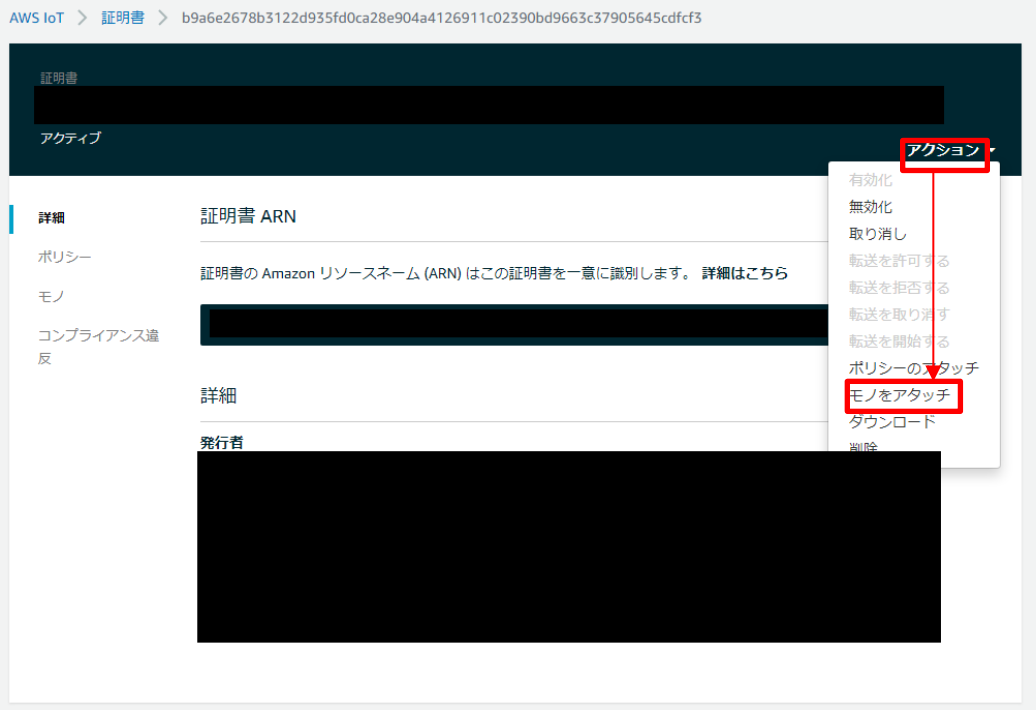
モノの登録時に作成したポリシーを選択し、「完了」をクリックします。

左端のメニューから「安全性」→「証明書」を選択します。

証明書のリストの中から先ほど作成した証明書を選択します。



「アクション」→「モノをアタッチ」を選択します。



先ほど作成したモノを選択し、「アタッチ」をクリックします。

以上で証明書とモノの紐づけが完了しました。

ダウンロードした証明書は、サンプルプロジェクトの key_crt_sig_generator フォルダに以下のように配置してください。

```
key_crt_sig_generator
|-- ca
|   |-- AmazonRootCA1.pem
|   |-- AmazonRootCA3.pem
|-- ca-sign-keypair-rsa2048
|-- client-ecc256
|   |-- *-certificate.pem.crt
|   |-- prime256v1-csr.pem.csr
|   |-- prime256v1-private.pem.key
|-- client-rsa2048
|-- output
|--1_rsa2048_convertCrt.sh
|--2_1_ecc256_generateKeyPair.sh
|--2_2_ecc256_convertCrt.sh
|--3_showkeyValues.sh
|--convertCrt.sh
```

3.2.4　ルート CA 証明書の署名生成と証明書ファイル形式の変換

TLS で使用する証明書は一般的に PEM 形式で提供されます。TSIP ドライバで使用するためには証明書を PEM 形式から DER 形式へと変換する必要があります。使用する Cipher Suite に応じて RSA、ECDSA のいずれか、または両方の証明書の変換を行ってください。また、ルート CA 証明書には署名を付加します。

3.2.4.1　RSA の証明書

RSA のルート CA 証明書 (AmazonRootCA1.pem) とクライアント証明書を DER 形式に変換します。また、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、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。

key_crt_sig_generator フォルダ内の 1_rsa2048_convertCrt.sh を実行してください。

以下は 1_rsa2048_convertCrt.sh の内容です。

```
#!/bin/sh

# If the format of client certificate is RSA, this project utilize RSA root
CA certificate.

INPUT_CRT_CLIENT_PEM="./client-rsa2048/*-certificate.pem.crt"
OUTPUT_CRT_CLIENT_DER="./output/client_rsa2048_crt.der"
OUTPUT_CRT_CLIENT_TXT="./output/client_rsa2048_crt_array.txt"

INPUT_CRT_ROOT_PEM="./ca/AmazonRootCA1.pem"
OUTPUT_CRT_ROOT_DER="./output/AmazonRootCA1_crt.der"
OUTPUT_CRT_ROOT_TXT="./output/AmazonRootCA1_crt_array.txt"

INPUT_KEY_PRIVATE_PEM="./ca-sign-keypair-rsa2048/rsa2048-private.pem"
INPUT_KEY_PUBLIC_PEM="./ca-sign-keypair-rsa2048/rsa2048-public.pem"

OUTPUT_CRT_ROOT_SIG_BIN="./output/AmazonRootCA1_sig.sig"
OUTPUT_CRT_ROOT_SIG_TXT="./output/AmazonRootCA1_sig_array.txt"

. ./convertCrt.sh
```

以下は 1_rsa2048_convertCrt.sh の内部から呼び出される convertCrt.sh の内容です。

```
#!/bin/sh

# 1.Convert PEM format client certicicate to DER
if [ -e $INPUT_CRT_CLIENT_PEM ]
then
    openssl x509 -in $INPUT_CRT_CLIENT_PEM -out $OUTPUT_CRT_CLIENT_DER -
outform der
    hexdump -v -e '/1 "0x%02x, "' $OUTPUT_CRT_CLIENT_DER >
$OUTPUT_CRT_CLIENT_TXT
else
    echo "Client certificate is not found"
    exit 1
fi

# 2.Convert PEM format root CA certicicate to DER
if [ -e $INPUT_CRT_ROOT_PEM ]
then
    openssl x509 -in $INPUT_CRT_ROOT_PEM -out $OUTPUT_CRT_ROOT_DER -outform
der
    hexdump -v -e '/1 "0x%02x, "' $OUTPUT_CRT_ROOT_DER > $OUTPUT_CRT_ROOT_TXT
else
    echo "Root CA certificate is not found"
    exit 1
fi

# 3.Generate RSA-2048 key pair for signature if it is not exist
if [ ! -e $INPUT_KEY_PRIVATE_PEM ] || [ ! -e $INPUT_KEY_PUBLIC_PEM ]
then
    openssl genrsa -out $INPUT_KEY_PRIVATE_PEM 2048
    openssl rsa -in $INPUT_KEY_PRIVATE_PEM -pubout -out $INPUT_KEY_PUBLIC_PEM
fi

# 4.Create a signature of the Root CA certificate
if [ -e $INPUT_KEY_PRIVATE_PEM ] && [ -e $OUTPUT_CRT_ROOT_DER ]
then
    openssl dgst -sha256 -sigopt rsa_padding_mode:pss -sigopt
rsa_pss_saltlen:-1 -sign $INPUT_KEY_PRIVATE_PEM -out
$OUTPUT_CRT_ROOT_SIG_BIN $OUTPUT_CRT_ROOT_DER
    openssl dgst -sha256 -sigopt rsa_padding_mode:pss -verify
$INPUT_KEY_PUBLIC_PEM -signature $OUTPUT_CRT_ROOT_SIG_BIN
$OUTPUT_CRT_ROOT_DER
    hexdump -v -e '/1 "0x%02x, "' $OUTPUT_CRT_ROOT_SIG_BIN >
$OUTPUT_CRT_ROOT_SIG_TXT
fi
```

スクリプトの実行後、output フォルダに以下の 6 つのファイルが生成されます。このうち、赤字のファイルをサンプルプロジェクトで使用します。これらのファイルは、生成されたバイナリデータを C 言語の uint8_t 型配列の形式で記述したものです。サンプルプロジェクトの amazon-freertos/vendors/renesas/boards/rx_mcu_boards/(board_name)/aws_demos/src/smc_gen/general 内に上書き保存してください。

- AmazonRootCA1_crt_array.txt
- AmazonRootCA1_sig_array.txt
- client_rsa2048_crt_array.txt
- AmazonRootCA1_crt.der
- AmazonRootCA1_sig.sig
- client_rsa2048_crt.der

3.2.4.2 ECDSA の証明書

ECDSA のルート CA 証明書 (AmazonRootCA3.pem) とクライアント証明書を DER 形式に変換します。また、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、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。

2_2_ecc256_convertCrt.sh を実行してください。

以下は 2_2_ecc256_convertCrt.sh の内容です。

```
#!/bin/sh

# If the format of client certificate is ECDSA, this project utilize ECDSA
root CA certificate.

INPUT_CRT_CLIENT_PEM="./client-ecc256/*-certificate.pem.crt"
OUTPUT_CRT_CLIENT_DER="./output/client_ecc256_crt.der"
OUTPUT_CRT_CLIENT_TXT="./output/client_ecc256_crt_array.txt"

INPUT_CRT_ROOT_PEM="./ca/AmazonRootCA3.pem"
OUTPUT_CRT_ROOT_DER="./output/AmazonRootCA3_crt.der"
OUTPUT_CRT_ROOT_TXT="./output/AmazonRootCA3_crt_array.txt"

INPUT_KEY_PRIVATE_PEM="./ca-sign-keypair-rsa2048/rsa2048-private.pem"
INPUT_KEY_PUBLIC_PEM="./ca-sign-keypair-rsa2048/rsa2048-public.pem"

OUTPUT_CRT_ROOT_SIG_BIN="./output/AmazonRootCA3_sig.sig"
OUTPUT_CRT_ROOT_SIG_TXT="./output/AmazonRootCA3_sig_array.txt"

. ./convertCrt.sh
```

スクリプトの実行後、output フォルダに以下の 6 つのファイルが生成されます。このうち、赤字のファイルをサンプルプロジェクトで使用します。これらのファイルは、生成されたバイナリデータを C 言語の uint8_t 型配列の形式で記述したものです。サンプルプロジェクトの amazon-freertos/vendors/renesas/boards/rx_mcu_boards/(board_name)/aws_demos/src/smc_gen/general 内に上書き保存してください。

- AmazonRootCA3_crt_array.txt
- AmazonRootCA3_sig_array.txt
- client_ecc256_crt_array.txt
- AmazonRootCA3_crt.der
- AmazonRootCA3_sig.sig
- client_ecc256_crt.der

3.2.5 鍵のラップ

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、鍵をラップして TSIP ドライバが受け入れられる形式に変換する必要があります。ここでは、TLS で使用する鍵を TSIP ドライバが受け入れられる形式に変換してデバイスに書き込む方法について説明します。

TLS で使用する鍵も証明書と同様、一般に PEM 形式で提供されます。TSIP ドライバで使用するためには PEM 形式の鍵ファイルから鍵データを抽出します。その後、Renesas DLM サーバ（https://dlm.renesas.com/）および Renesas Secure Flash Programmer を使用して、以下の手順でラップします。

1. PEM 形式の鍵ファイルから、ルート CA 証明書の署名検証用鍵データおよびクライアント証明書の鍵データを抽出する。
   3_showkeyValues.sh を実行します。

```
#!/bin/sh

KEY_ROOT_SIGNATURE_RSA2048="./ca-sign-keypair-rsa2048/rsa2048-private.pem"
KEY_CLIENT_RSA2048="./client-rsa2048/*-private.pem.key"
KEY_CLIENT_ECC256="./client-ecc256/prime256v1-private.pem.key"

echo Root CA signature RSA-2048bit Public:
if [ -e $KEY_ROOT_SIGNATURE_RSA2048 ]
then

   openssl asn1parse -in $KEY_ROOT_SIGNATURE_RSA2048 | awk -F:
'NR==3{print $4} NR==4{printf("%08d¥n", $4)}' | tr -d "¥n"
else
   echo "Not found"
fi

echo -e "¥n¥nClient RSA-2048bit All:"
if [ -e $KEY_CLIENT_RSA2048 ]
then
   openssl asn1parse -in $KEY_CLIENT_RSA2048 | awk -F: 'NR==3{print $4}
NR==4{printf("%08d¥n", $4)} NR==5{print $4}' | tr -d "¥n"
else
   echo "Not found"
fi

echo -e "¥n¥nClient ECC-256bit All: "
if [ -e $KEY_CLIENT_ECC256 ]
then
   TEMP_PRIVATE=`openssl ec -in $KEY_CLIENT_ECC256 -text -noout | sed -n
3,5p`
   TEMP_PUBLIC=`openssl ec -in $KEY_CLIENT_ECC256 -text -noout | sed -n
7,11p`
   echo $TEMP_PUBLIC $TEMP_PRIVATE | sed "s/^04//" | tr -d " :"
else
   echo "Not found"
fi
```

   スクリプトを実行すると、ルート CA 証明書の署名検証用鍵データおよびクライアント証明書の鍵データが Renesas Secure Flash Programmer の Key Wrap タブ、Key Data に入力できる形式でコンソール上に出力されます。

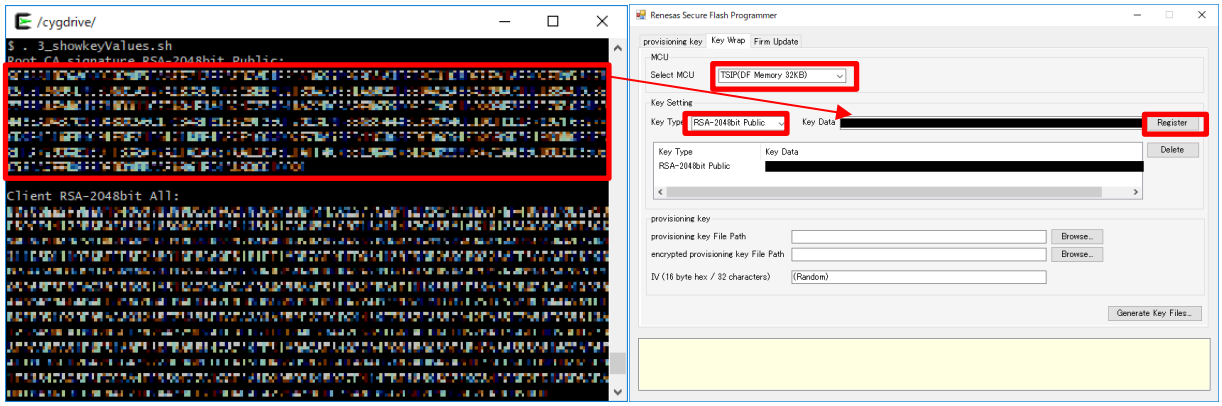
2. 任意の provisioning key を生成し、DLM サーバにアップロードして encrypted provisioning key を生成します。provisioning key は署名検証用の公開鍵をラップするために使用する鍵です。
   Renesas Secure Flash Programmer を使用して、DLM サーバが受け付けるフォーマットの provisioning key ファイルを作成することができます。provisioning key ファイルの作成方法は、5.1 節を参照してください。DLM サーバの使用方法は、DLM サーバのトップページにある FAQ より、操作マニュアルを参照してください。

3. その後、encrypted provisioning key、平文の provisioning key、1 の手順で抽出したルート CA 証明書の署名検証用鍵データ、クライアント証明書の鍵データの 4 つを Renesas Secure Flash Programmer に入力し、暗号化鍵ファイル（key_data.c および key_data.h）を生成します。暗号化鍵ファイルには、encrypted provisioning key と provisioning key でラップされた鍵（encrypted key）として、ルート CA 証明書の署名検証用鍵、クライアント証明書鍵が出力されます。詳細な手順については、5.2 節を参照してください。ルート CA 証明書の署名検証用鍵とクライアント証明書の鍵ペアは以下の手順で Renesas Secure Flash Programmer に入力します。

3-1. ルート CA 証明書の署名検証用鍵の入力

Key Wrap タブ、Key Type には、登録する鍵ペアの種類として RSA-2048bit Public を指定してください。

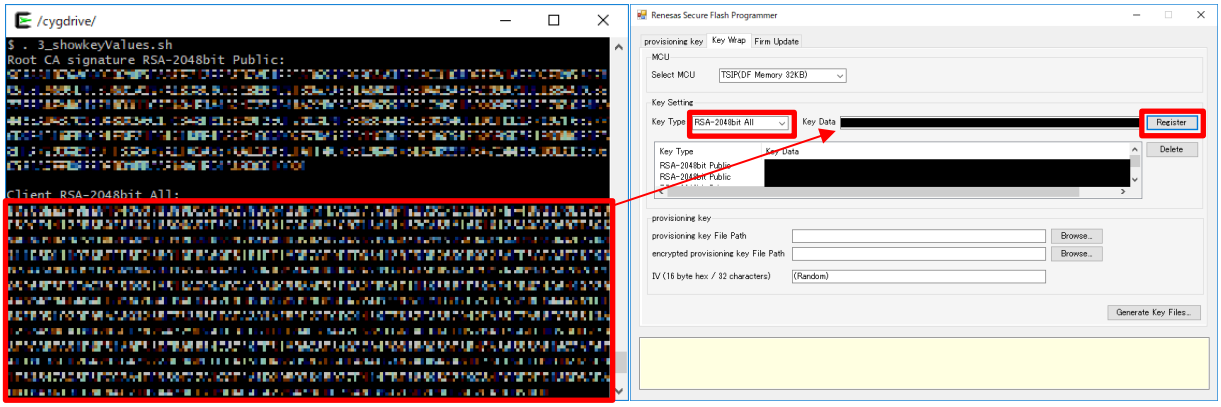
Renesas Secure Flash Programmer の Key Wrap タブ、Key Data に、1 の手順で抽出したルート CA 証明書の署名検証用鍵を入力し、Register をクリックします。



3-2. クライアント証明書の鍵ペアの入力

Key Wrap タブ、Key Type には登録する鍵ペアの種類を指定します。RSA のクライアント証明書を使用する場合は RSA-2048bit All、ECDSA のクライアント証明書を使用する場合は ECC-256bit All を指定してください。

Key Data に、1 の手順で抽出したクライアント証明書の鍵ペアを入力し、Register をクリックします。



生成された暗号化鍵ファイル（key_data.c および key_data.h）は、ファイルをサンプルプロジェクトの amazon-freertos/vendors/renesas/boards/rx_mcu_boards/(board_name)/aws_demos/src/smc_gen/general 内に上書き保存してください。

3.3　AWS との通信設定

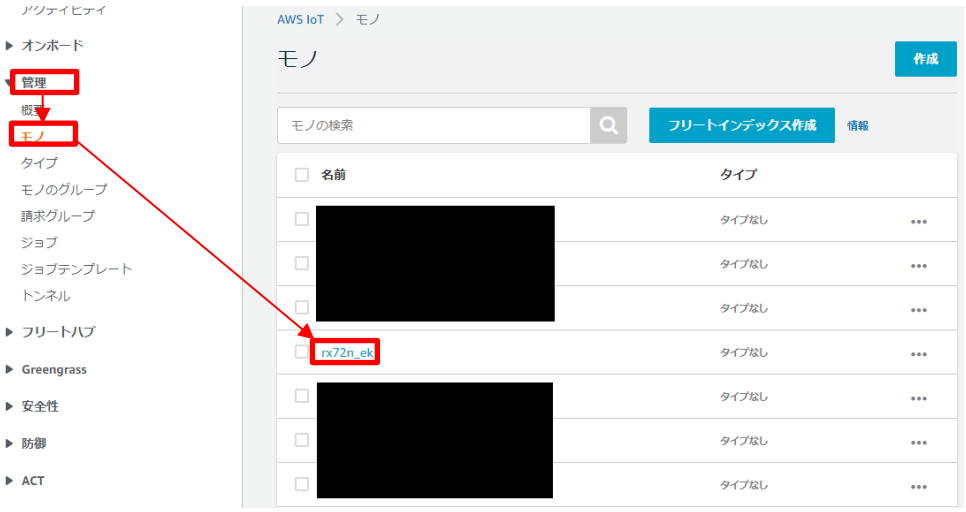
FreeRTOS で用意されている設定ファイルと本サンプルプログラムの設定ファイルで各種設定を行います。

3.3.1　AWS IoT の設定

amazon-freertos フォルダ内の demos/include/aws_clientcredential.h を開き、以下の項目を設定してください。

- #define clientcredentialMQTT_BROKER_ENDPOINT に Rest API エンドポイントを記入する。
- #define clientcredentialIOT_THING_NAME にモノの名前を記入する。

Rest API エンドポイントとモノの名前を確認するには AWS マネジメントコンソール (https://aws.amazon.com/console/)にサインインし、「すべてのサービス」→「IoT」→「IoT コア」をクリックしてください。「管理」「モノ」をクリックし、3.2 節で作成したモノを選択します。



「相互作用」をクリックして表示される画面の左上に表示されているのがモノの名前、「HTTPS」の項目に記載されているのが Rest API エンドポイントです。

3.3.2　IP の設定

デフォルトの設定では DHCP を使用します。ターゲットボードを接続するルータの DHCP 機能が無効の場合、以下の設定を行ってください。

amazon-freertos フォルダ内の vendors/renesas/boards/rx_mcu_boards/(board_name)/aws_demos/config_files/FreeRTOSIPConfig.h を開き、#define ipconfigUSE_DHCP を 0 に設定する。

amazon-freertos フォルダ内の vendors/renesas/boards/rx_mcu_boards(board_name)/aws_demos/config_files/FreeRTOSConfig.h を開き、IP アドレス、デフォルトゲートウェイ、DNS サーバアドレス、サブネットマスクを入力する。

3.3.3　クライアント証明書の形式の選択

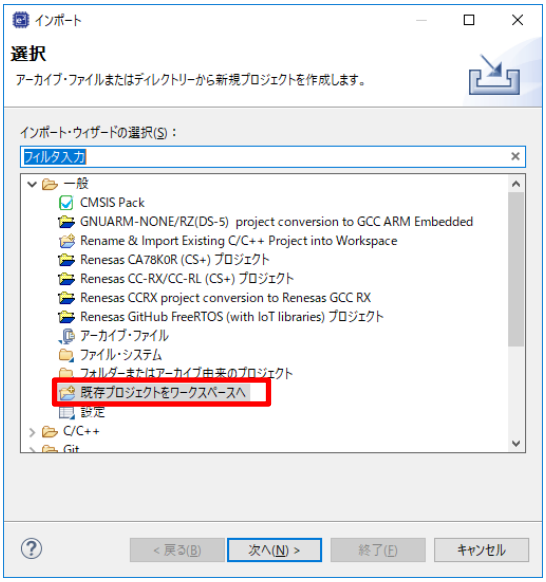
デフォルトでは RSA 証明書が有効になっています。ECDSA 証明書に切り替えるには、amazon-freertos/vendors/renesas/boards/rx_mcu_boards/(board_name)/aws_demos/src/smc_gen/general/r_trust_certificate_data.h を開き、以下のように編集してください。

```
// RSA or ECDSA
//#define TSIP_CLIENT_CERTIFICATE_TYPE   R_TSIP_TLS_PUBLIC_KEY_TYPE_RSA2048
// RSA
#define TSIP_CLIENT_CERTIFICATE_TYPE R_TSIP_TLS_PUBLIC_KEY_TYPE_ECDSA_P256
// ECDSA
```

4.　プロジェクトのビルドおよび実行

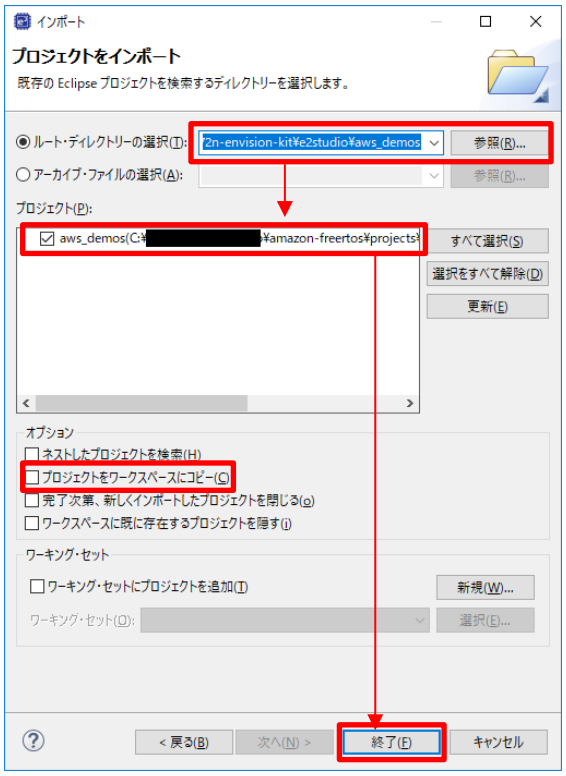
4.1　プロジェクトのインポート

e²studio を開いて任意のワークスペースに移動し、[File](ファイル)→[Import](インポート)をクリックします。次に、下に示すように[General](一般)→[Existing Projects into Workspkace](既存プロジェクトをワークスペースへ)を選択します。



[Select root directory](ルート・ディレクトリーの参照)の、[Browse…](参照)ボタンを押して、amazon-freertos/projects/renesas/(board_name)/e2studio/aws_demos を選択します。aws_demos という名前のプロジェクトが選択可能になっていることを確認し、[Finish](終了)をクリックします。

この時に、[Copy projects into workspace](プロジェクトをワークスペースにコピー)にチェックが入っていないことを確認してください。



4.2　プロジェクトのビルド

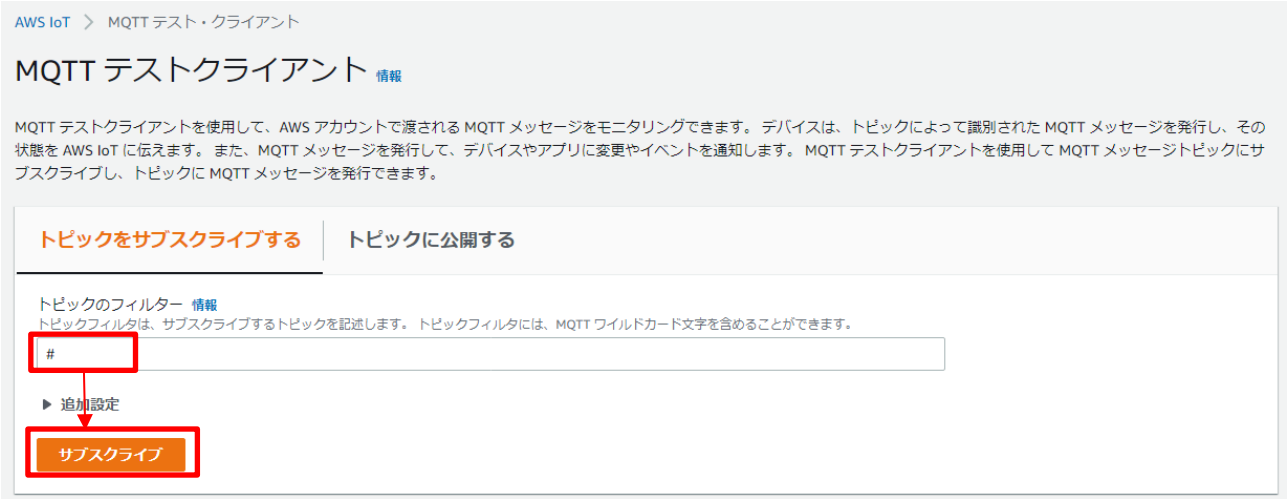
[Project](プロジェクト)→[Build All](すべてをビルド)をクリックしてプロジェクトをビルドします。この際、Warning が出現しますが動作に問題はありません。

ビルド完了後、図 3-1 を参考にターゲットボードを PC とルータに接続し、[Run](実行)→[Debug](デバッグ(D))をクリックしてデバッグを開始してください。

4.3　AWS IoT との接続

AWS マネジメントコンソール (https://aws.amazon.com/console/)にサインインし、「すべてのサービス」→「IoT」→「IoT」コアをクリックしてください。左端のメニューから「テスト」を選択し、MQTT テストクライアントを開きます。

「トピックのフィルター」にワイルドカードの"#"を入力し、サブスクライブをクリックします。

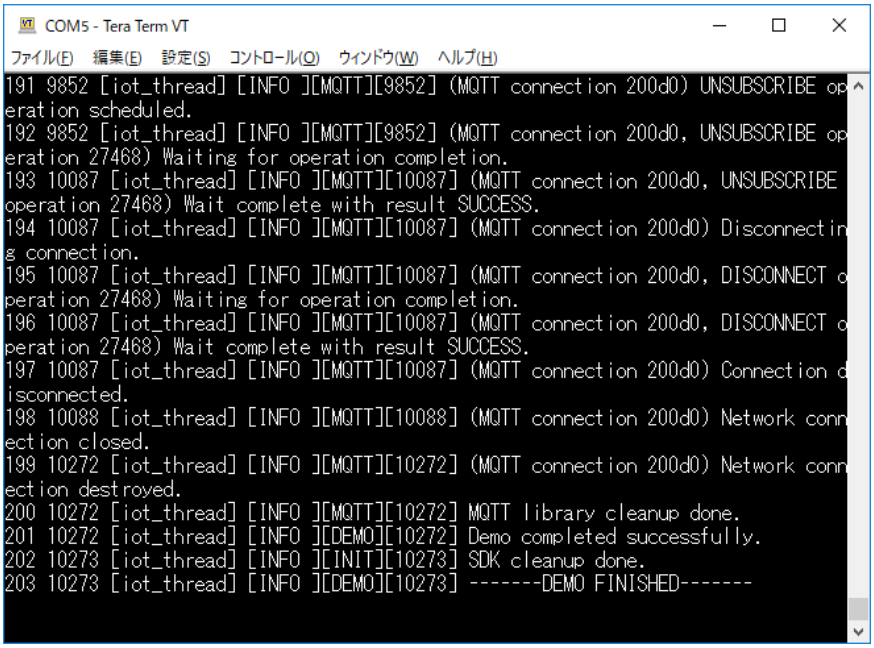


以下のように画面下部に空のコンソールが表示されることを確認してください。

Tera Term 等、任意のターミナルエミュレータを起動してターゲットボードとのシリアル通信を開始します。その後、e²studio の[Run](実行) → [Resume](再開)ボタンをクリックしプログラムを実行すると、AWS 接続を行うプログラムが実行されます。

AWS との接続に成功すると、MQTT クライアントに通信ログが出力されます。

また、ターミナルエミュレータにはタスクのログが表示されます。


COM5 - Tera Term VT
ファイル(F) 編集(E) 設定(S) コントロール(O) ウィンドウ(W) ヘルプ(H)
191 9852 [iot_thread] [INFO][MQTT][9852] (MQTT connection 200d0) UNSUBSCRIBE op
eration scheduled.
192 9852 [iot_thread] [INFO][MQTT][9852] (MQTT connection 200d0, UNSUBSCRIBE op
eration 27468) Waiting for operation completion.
193 10087 [iot_thread] [INFO][MQTT][10087] (MQTT connection 200d0, UNSUBSCRIBE
operation 27468) Wait complete with result SUCCESS.
194 10087 [iot_thread] [INFO][MQTT][10087] (MQTT connection 200d0) Disconnectin
g connection.
195 10087 [iot_thread] [INFO][MQTT][10087] (MQTT connection 200d0, DISCONNECT o
peration 27468) Waiting for operation completion.
196 10087 [iot_thread] [INFO][MQTT][10087] (MQTT connection 200d0, DISCONNECT o
peration 27468) Wait complete with result SUCCESS.
197 10087 [iot_thread] [INFO][MQTT][10087] (MQTT connection 200d0) Connection d
isconnected.
198 10088 [iot_thread] [INFO][MQTT][10088] (MQTT connection 200d0) Network conn
ection closed.
199 10272 [iot_thread] [INFO][MQTT][10272] (MQTT connection 200d0) Network conn
ection destroyed.
200 10272 [iot_thread] [INFO][MQTT][10272] MQTT library cleanup done.
201 10272 [iot_thread] [INFO][DEMO][10272] Demo completed successfully.
202 10273 [iot_thread] [INFO][INIT][10273] SDK cleanup done.
203 10273 [iot_thread] [INFO][DEMO][10273] -------DEMO FINISHED-------

5.　Renesas Secure Flash Programmer の使用方法

5.1　provisioning key ファイルの生成

図 5.1 に Renesas Secure Flash Programmer の provisioning key タブを示します。16 進数 32byte のフォーマットに沿う provisioning key の値を「provisioning key Value」に入力し、「format to DLM server file…」を押して DLM サーバに送るための provisioning key ファイルを作成してください。

「(Random)」が表示された状態で「format to DLM server file…」を押すことで、乱数で生成した provisioning key ファイルを作成することができますが、この乱数は十分な精度持つものではないため、実製品では使用することはできません。

図 5.1 Renesas Secure Flash Programmer の Provisioning Key タブ

5.2　暗号化鍵ファイルの生成

図 5.2 に Renesas Secure Flash Programmer の Key Wrap タブを、表 5-1 に Key Wrap タブの設定値の説明を示します。表 5-1 の説明に沿って設定値を入力し、「Generate Key Files…」を押して暗号化鍵ファイル（key_data.c および key_data.h）を生成してください。各ボタンの説明は、表 5-2 を参照してください。

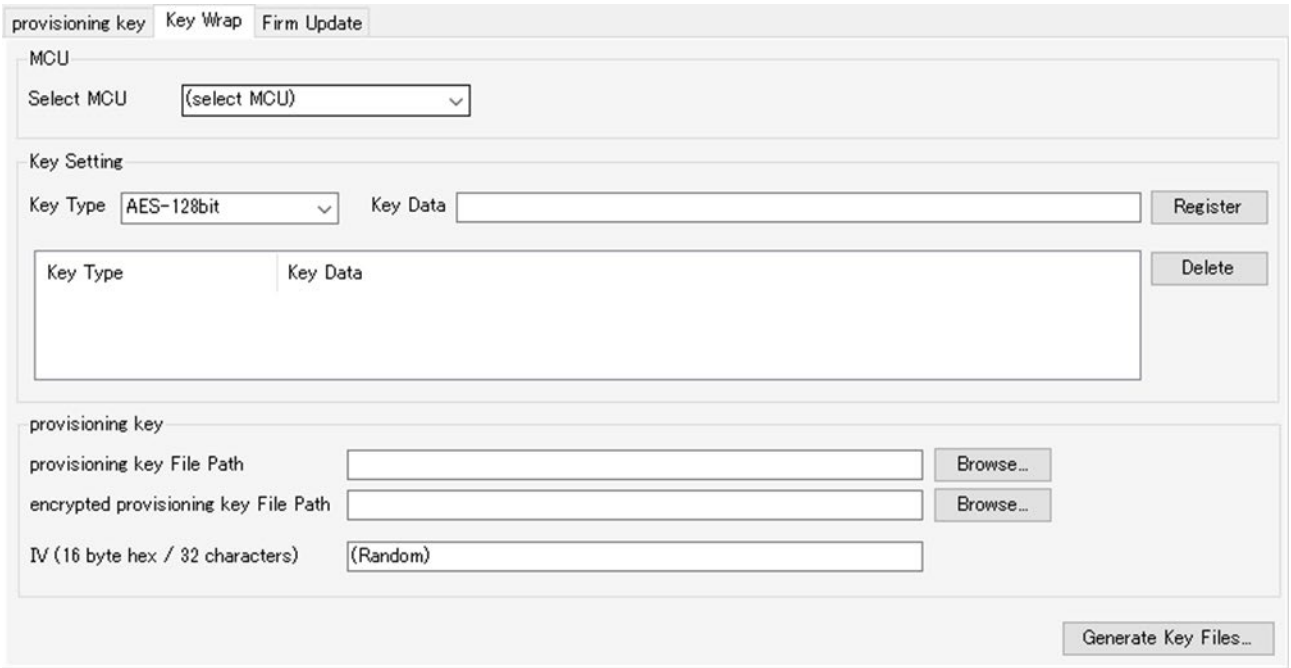


図 5.2 Renesas Secure Flash Programmer の Key Wrap タブ

表 5-1 Key Wrap タブ設定値の説明

パラメータ	設定値	説明
Select MCU	• TSIP-Lite(DF Memory 8KB) • TSIP-Lite(DF Memory 32KB) • TSIP(DF Memory 8KB) • TSIP(DF Memory 32KB)	使用する MCU(TSIP 種別およびデータフラッシュメモリサイズ)を選択します。このサンプルプログラムでは TSIP(DF Memory 32KB)を指定してください。
Key Type	• AES-128bit • AES-256bit • DES • 2Key-TDES • Triple-DES • ARC4-2048bit • SHA1-HMAC • SHA256-HMAC • RSA-1024bit Public/Private/All • RSA-2048bit Public/Private/All • RSA-3072bit Public • RSA-4096bit Public • ECC-192bit Public/Private/All • ECC-224bit Public/Private/All • ECC-256bit Public/Private/All • ECC-384bit Public/Private/All • Update Key Ring	生成するユーザ鍵の種類を指定します。このサンプルプログラムでは以下を指定してください。 • RSA-2048bit Public Key Data がルート CA 証明書の署名検証用鍵データの場合。 • RSA-2048bit All Key Data が RSA のクライアント証明書の鍵データの場合。 • ECC-256bit All Key Data が ECDSA のクライアント証明書の鍵データの場合。
Key Data	ユーザ鍵 鍵フォーマットは、RX ファミリ TSIP(Trusted Secure IP)モジュール Firmware Integration Technology (R20AN0371)の 7 章を参照。	生成するユーザ鍵データを指定してください。このサンプルプログラムでは上記 Key Type に示す鍵を使用します。
provisioning key File Path	provisioning key ファイルのファイルパス	ユーザ鍵を暗号化する際に使用する、平文の provisioning key ファイルのファイルパスを指定してください。provisioning key ファイルの作成方法は、5.1 節を参照してください。
encrypted provisioning key File Path	ラップされた provisioning key ファイルのファイルパス	C 言語ファイルに出力する、ラップされた provisioning key ファイルのパスを指定してください。ラップされた provisioning key ファイルは、DLM サーバで生成してください。
IV (16 byte hex / 32 characters)	IV 値	16 バイトの IV 値を入力してください。TSIP に入力するユーザ鍵は、暗号化され MAC 値が付加されている必要がありますが、その演算の際の初期化ベクタとして使用します。
Generate Key File…	C 言語ファイルを生成するボタン	C 言語の暗号化鍵ファイルを出力します。

表 5-2 Key Wrap タブのボタンの説明

ボタン	説明
Register	Key Data 欄に指定したユーザ鍵データを登録します。Key Data には Key Type に合わせたユーザ鍵データを入力して、登録してください。
Delete	登録されているユーザ鍵データを削除します。削除するユーザ鍵データをウィンドウで選択した状態でボタンを押して、削除してください。
Browse…	provisioning key ファイル、ラップされた provisioning key ファイルのファイルパスをエクスプローラーから指定する際に使用します。ファイルパスは、直接入力することもできます。

Generate Key Files…	暗号化鍵ファイル（key_data.c および key_data.h）を生成します。各欄に適切な値を入力した状態で押してください。ボタンを押すと、暗号化鍵ファイルを出力するフォルダを指定する画面に切り替わり、ファイルが出力されます。

6. 付録

6.1 TSIP ドライバを用いた TLS 通信の性能

表 6-1 に RX72N Envision kit で TLS 通信をした際の Application Data の通信速度の例を示します。1MB 分のデータの転送時間を MCU 内蔵のタイマで測定し、5 回分の転送時間の平均値から算出しています。この例では、TSIP ドライバを利用することにより、通信速度が 3 – 7Mbps から 20 – 30Mbps に向上しています。

表 6-1　TSIP ドライバを用いた TLS 通信速度の例

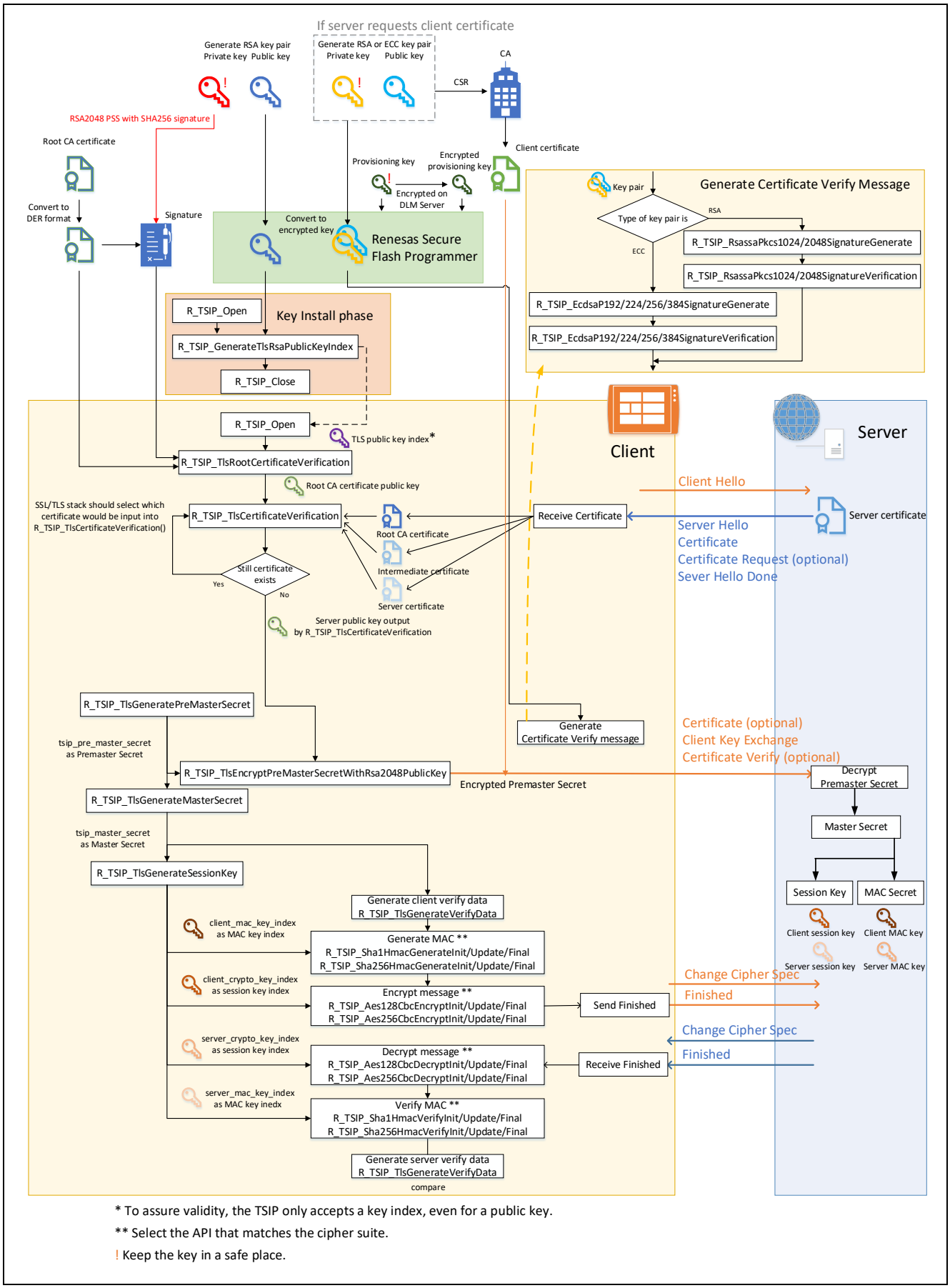
Cipher Suite	Block Cipher	Mbed TLS 注 1	Mbed TLS w/ TSIP 注 2
TLS_RSA_WITH_AES_128_CBC_SHA	128bit AES-CBC	Up: **6.4**Mbps Down: **6.6**Mbps	Up: **25.0**Mbps Down: **28.3**Mbps
TLS_RSA_WITH_AES_256_CBC_SHA	256bit AES-CBC	Up: **5.5**Mbps Down: **5.6**Mbps	Up: **24.2**Mbps Down: **27.2**Mbps
TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256	128bit AES-GCM	Up: **3.7**Mbps Down: **3.8**Mbps	Up: **22.4**Mbps Down: **29.5**Mbps

【注】 システムクロック (ICLK): 240MHz
TSIP 動作クロック (PCLKB): 60MHz

【注】 1. Mbed TLS: ソフトウェア処理
2. Mbed TLS w/ TSIP: TSIP ドライバの TLS 向け API を使用

6.2 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 6-1 に、ECDHE の場合を図 6-2 に示します。

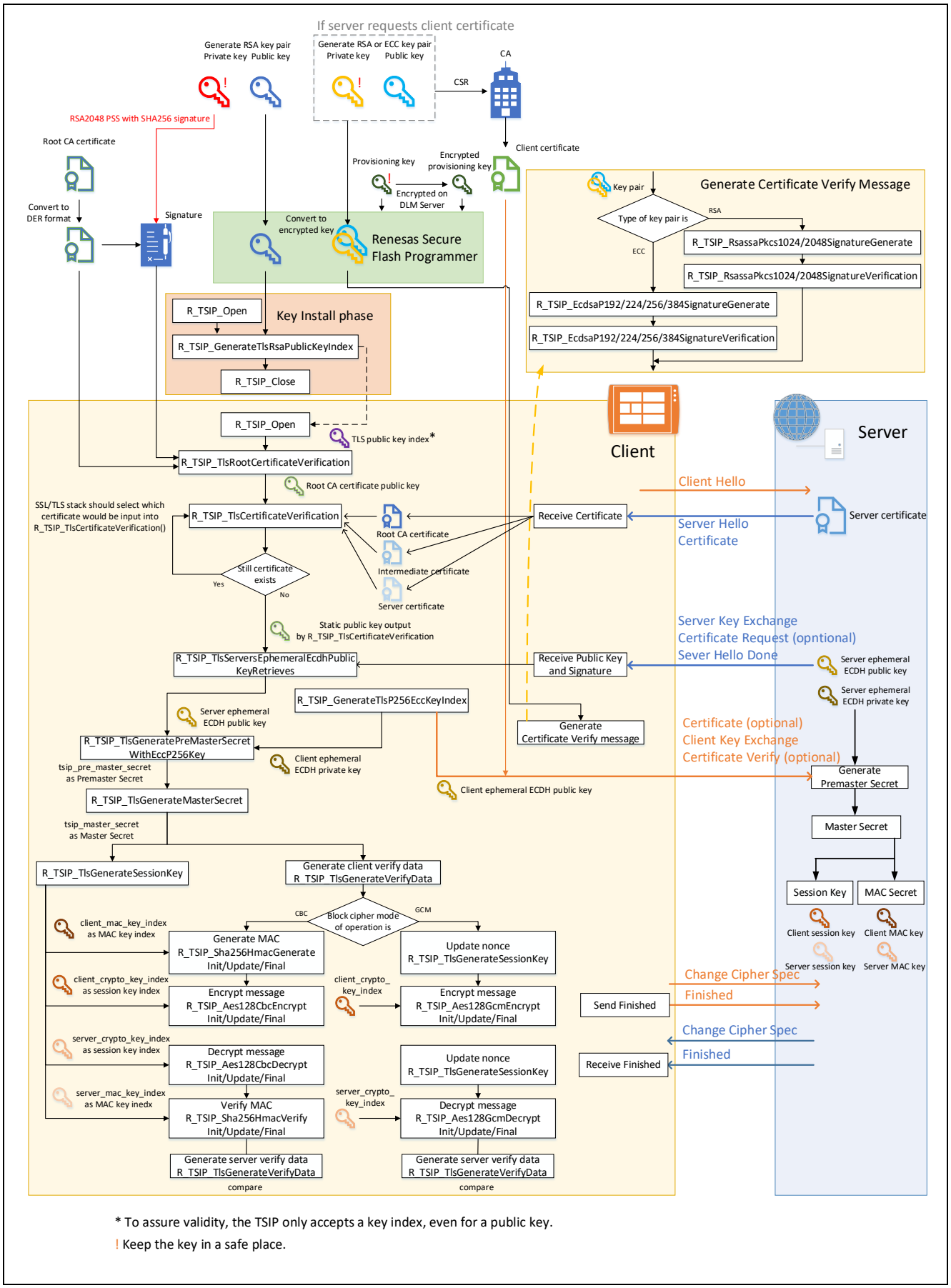


* To assure validity, the TSIP only accepts a key index, even for a public key.

** Select the API that matches the cipher suite.

! Keep the key in a safe place.

図 6-1　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 key index, even for a public key.

! Keep the key in a safe place.

図 6-2　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

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。