RX ファミリ

R20AN0371JJ0121
Rev.1.21
2024.06.28

TSIP(Trusted Secure IP)モジュール Firmware Integration Technology

要旨

本資料は、RX ファミリ搭載の TSIP(Trusted Secure IP) および TSIP-Lite を活用するためのソフトウェア・ドライバの使用方法を記します。このソフトウェア・ドライバは TSIP ドライバと呼びます。

TSIP ドライバは、Firmware Integration Technology(FIT)モジュールとして提供されます。FIT の概念については以下 URL を参照してください。

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

TSIP ドライバは 表 1、表 2 にまとめた暗号機能、およびファームウェアアップデートをセキュアに行うための API を持ちます。

動作確認デバイス

TSIP : RX65N, RX651 グループ、RX671 グループ、RX72M グループ、RX72N グループ

TSIP-Lite : RX231 グループ、RX23W グループ、RX26T グループ、RX66T グループ、RX72T グループ

TSIP 機能がある製品型名については各 RX マイコンのユーザーズマニュアルを参照してください。

表 1　TSIP 暗号アルゴリズム

暗号種別		アルゴリズム
非対称鍵暗号	暗号化/復号	RSAES-PKCS1-v1_5(1024/2048/3072/4096 bit)【注 1】：RFC8017
	署名生成/検証	RSASSA-PKCS1-v1_5(1024/2048/3072/4096 bit)【注 1】：RFC8017 ECDSA(ECC P-192/224/256/384)：FIPS186-4
	鍵生成	RSA(1024/2048 bit) ECC P-192/224/256/384
対象鍵暗号	AES	AES(128/256 bit) ECB/CBC/CTR：FIPS 197, SP800-38A
	DES	TDES(56/56x2/56x3 bit) ECB/CBC：FIPS 46-3
	ARC4	ARC4(2048 bit)
ハッシュ	SHA	SHA-1, SHA-256：FIPS 180-4
	MD5	MD5：RFC1321
認証付き暗号(AEAD)		GCM/CCM：FIPS 197, SP800-38C, SP800-38D
メッセージ認証		CMAC(AES)：FIPS 197, SP800-38B GMAC：RFC4543 HMAC(SHA)：RFC2104
疑似乱数ビット生成		SP 800-90A
乱数生成		SP 800-22 で検定済み
TLS	TLS1.2	TLS1.2：RFC5246【注 2】 サポートしている機能: クライアント機能/サーバ機能 サポートしている cipher suite(TLS1.2): 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
	TLS1.3	TLS1.3：RFC8446【注 3】 サポートしている機能: クライアント機能/サーバ機能 Full Handshake/Resumption/0-RTT サポートしている cipher suite(TLS1.3): TLS_AES_128_GCM_SHA256 TLS_AES_128_CCM_SHA256
鍵更新機能		AES, RSA, DES, ARC4, ECC, HMAC
鍵共有		ECDH P-256, ECDHE NIST P-512：SP800-56A, SP800-56C DH(2048 bit)
Key Wrap		AES(128/256 bit)

【注】1. RSA(3072/4096 bit)は、署名検証ならびに公開鍵を使用したべき乗剰余演算のみのサポートです。
2. サーバ機能の対象デバイスは RX65N, RX651 グループ、RX66N グループ、RX72M グループ、RX72N グループです。
3. クライアント機能の Resumption と 0-RTT、ならびにサーバ機能の対象デバイスは RX65N, RX651 グループ、RX66N グループ、RX72M グループ、RX72N グループです。

表 2　TSIP-Lite 暗号アルゴリズム

暗号種別		アルゴリズム
対象鍵暗号	AES	AES(128/256 bit) ECB/CBC/CTR：FIPS 197, SP800-38A
認証付き暗号(AEAD)		GCM/CCM：FIPS 197, SP800-38C, SP800-38D
メッセージ認証		CMAC(AES)：FIPS 197, SP800-38B GMAC：RFC4543
疑似乱数ビット生成		SP 800-90A
乱数生成		SP 800-22 で検定済み
鍵更新機能		AES
Key Wrap		AES(128/256 bit)

注

RFC 2104: HMAC: Keyed-Hashing for Message Authentication (rfc-editor.org)
RFC 8017: PKCS #1: RSA Cryptography Specifications Version 2.2 (rfc-editor.org)
RFC 4543: The Use of Galois Message Authentication Code (GMAC) in IPsec ESP and AH (rfc-editor.org)
RFC 5246: The Transport Layer Security (TLS) Protocol Version 1.2 (rfc-editor.org)
RFC 8446: The Transport Layer Security (TLS) Protocol Version 1.3 (rfc-editor.org)

FIPS 46-3, Data Encryption Standard (DES) (withdrawn May 19, 2005) (nist.gov)
FIPS186-4: https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.186-4.pdf
NIST SP 800-38A, Recommendation for Block Cipher Modes of Operation Methods and Techniques
NIST SP 800-38-B Recommendation for Block Cipher Modes of Operation: The CMAC Mode for Authentication (nist.gov)
NIST SP 800-38D, Recommendationfor Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC
NIST SP800-56A: Recommendation for Pair-Wise Key Establishment Schemes Using Discrete Lograrithm Cryptography (nist.gov)
NIST SP800-56C: Recommendation for Key-Derivation Methods in Key-Establishment Schemes (nist.gov)
NIST SP800-22: https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-22r1a.pdf
NIST SP800-90A: https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-90Ar1.pdf

目次

1. 概要

1.1 用語

本資料中で使用している用語の説明をいたします。MCU のユーザーズマニュアル ハードウェア編 Trusted Secure IP 章の「鍵インストール概念図」と鍵の名称が異なる用語を使用している箇所があります。「鍵インストール概念図」(本書の図 1-1)と合わせてご確認ください。

表 1-1　用語説明

用語	内容	鍵インストール概念図との対応
鍵注入	工場でデバイスに Wrapped Key を注入すること。	-
鍵更新	フィールドでデバイスに Wrapped Key を注入すること。	-
ユーザ鍵、 User Key	ユーザが使用する平文状態の暗号鍵。デバイス上では使用しない。 AES、DES、ARC4、HMAC の場合は共通鍵がユーザ鍵となり、RSA、ECC の場合は公開鍵、秘密鍵がそれぞれユーザ鍵となる。	Key-1
Encrypted Key	ユーザ鍵に UFPK もしくは鍵更新用鍵による MAC 値の付加および暗号化をして生成される鍵情報。同一のユーザ鍵に対する Encrypted Key は 各デバイスで共通の値となる。	eKey-1
Wrapped Key	Encrypted Key を鍵の注入または鍵更新により TSIP で使用できる形式に変換したデータ。Wrapped Key は HUK でラッピングされているため、同一の Encrypted Key に対する Wrapped Key でもデバイスごとに固有の値となる。	Index-1 もしくは Index-2
UFPK	User Factory Programming Key 鍵注入においてユーザ鍵から Encrypted Key を生成するために使用する、ユーザが設定する鍵束。 デバイス上では使用しない。	Key-2
W-UFPK	Wrapped UFPK UFPK を DLM サーバ上の HRK によりラッピングすることで生成される鍵情報。TSIP 内部にて HRK で UFPK に復号されて使用される。	Index-2
鍵更新用鍵、 Key Update Key (KUK)	鍵更新においてユーザ鍵から Encrypted Key を生成するために使用する、ユーザが設定する鍵。 デバイス上で鍵更新を行うにはあらかじめ鍵注入により Wrapped KUK を生成しておく必要がある。	-
Hardware Root Key (HRK)	TSIP 内部とルネサス内セキュアルームのみに存在する共通の暗号鍵。	-
Hardware Unique Key (HUK)	TSIP 内部で導出する、鍵の保護のために使用するデバイス固有の暗号鍵。	-
DLM (Device Lifecycle Management) サーバ	Renesas 鍵管理サーバ。UFPK のラッピングに使用する。	-

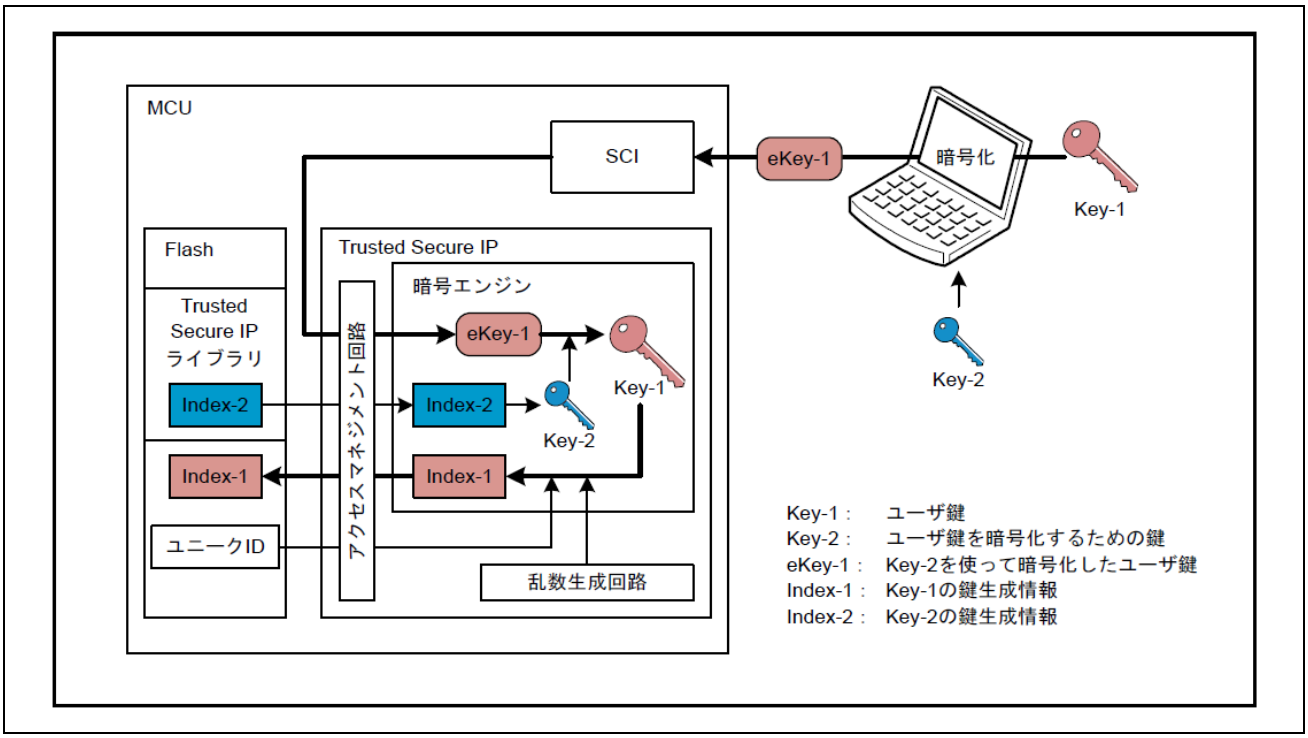

図 1-1　鍵インストール概念図 (RX65N グループ、RX651 グループ ユーザーズマニュアル　ハードウェア編　52. Trusted Secure IP 図 52.4 より抜粋)

1.2　TSIP 概要

RX ファミリ内の Trusted Secure IP（TSIP）ブロックは、不正アクセスを監視することで、MCU 内部に安全な領域を作成します。これにより、TSIP は暗号化エンジンおよびユーザ鍵（暗号鍵）を確実かつ安全に使用することが可能です。TSIP は、TSIP ブロックの外部において、暗号鍵を安全で解読不可能な Wrapped Key と呼ばれる形式で扱います。このため信頼できる安全な暗号処理において最も重要な要素である暗号鍵を、フラッシュメモリ内に保存することが可能です。

TSIP ブロックには安全領域があり、暗号化エンジン、平文の暗号鍵用のストレージが格納されています。

TSIP は、TSIP 内部で Wrapped Key から暗号演算に使用する暗号鍵を復元します。Wrapped Key は、Unique ID をもとに導出された HUK に紐付けられて生成されているため、デバイス固有の値になります。このため、あるデバイスの Wrapped Key を別のデバイスにコピーして使用することができません。アプリケーションから TSIP ハードウェアにアクセスするためには、TSIP ドライバを使用する必要があります。

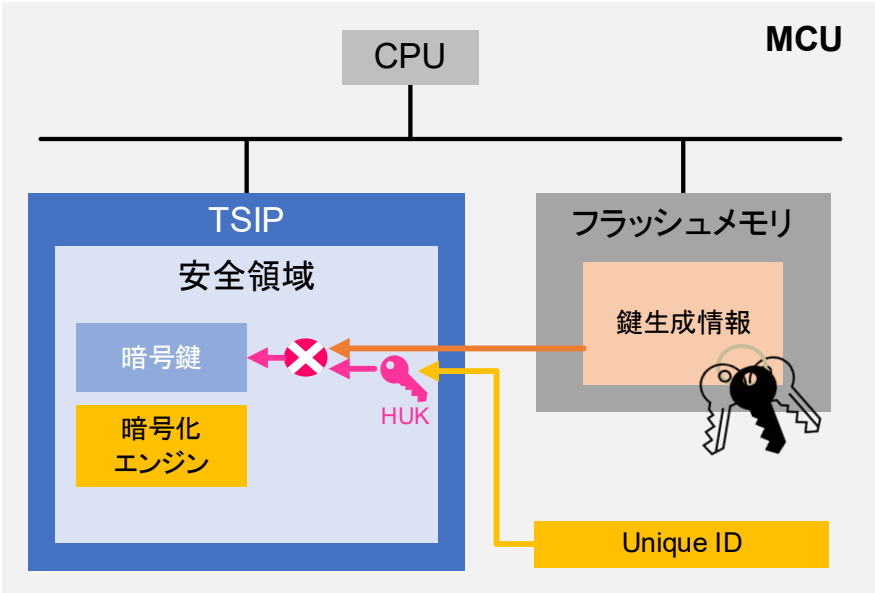

図 1-2　TSIP 搭載 MCU

1.3　製品構成

本製品は、以下の表 1-2 のファイルが含まれます。

表 1-2　製品構成

ファイル/ディレクトリ(**太字**)名	内容
Readme.txt	Readme
RX_TSIP_SoftwareLicenseAgreement_JPN.pdf	ソフトウェア利用許諾契約書(日本語)
RX_TSIP_SoftwareLicenseAgreement_ENG.pdf	ソフトウェア利用許諾契約書(英語)
r20an0371jj0121-rx-tsip-security.pdf	TSIP ドライバ アプリケーションノート(日本語)
r20an0371ej0121-rx-tsip-security.pdf	TSIP ドライバ アプリケーションノート(英語)
reference_documents	FIT モジュールを各種統合開発環境で使用する方法等を記したドキュメントを格納するフォルダ
ja	FIT モジュールを各種統合開発環境で使用する方法等を記したドキュメントを格納するフォルダ(日本語)
r01an1826jj0110-rx.pdf	CS+に組み込む方法(日本語)
r01an1723ju0121-rx.pdf	e^2 studio に組み込む方法(日本語)
r20an0451js0140-e2studio-sc.pdf	スマート・コンフィグレータ ユーザーガイド(日本語)
r01an5792jj0102-rx-tsip.pdf	AES 暗号プロジェクト アプリケーションノート(日本語)
r01an5880jj0103-rx-tsip.pdf	TLS 連携機能プロジェクト アプリケーションノート(日本語)
en	FIT モジュールを各種統合開発環境で使用する方法等を記したドキュメントを格納するフォルダ(英語)
r01an1826ej0110-rx.pdf	CS+に組み込む方法(英語)
r01an1723eu0121-rx.pdf	e^2 studio に組み込む方法(英語)
r20an0451es0140-e2studio-sc.pdf	スマート・コンフィグレータ ユーザーガイド(英語)
r01an5792ej0102-rx-tsip.pdf	AES 暗号プロジェクト アプリケーションノート(英語)
r01an5880ej0103-rx-tsip.pdf	TLS 連携機能プロジェクト アプリケーションノート(英語)
FITModules	FIT モジュールフォルダ
r_tsip_rx_v1.21.zip	TSIP ドライバ FIT Module
r_tsip_rx_v1.21.xml	TSIP ドライバ FIT Module e^2 studio FIT プラグイン用 XML ファイル
r_tsip_rx_v1.21_extend.mdf	TSIP ドライバ FIT Module スマート・コンフィグレータ用コンフィグレーション設定ファイル
FITDemos	デモプロジェクトフォルダ
rx*XXX_bbb*_tsip_sample *1 *2	鍵書き込み方法と暗号 API の利用方法を示すプロジェクト
rx*XXX_bbb*_tsip_secure_update *1 *2	セキュアブート/セキュアアップデートの実装例
rx*XXX_bbb*_tsip_secure_boot *1 *2	セキュアブートファームウェア
rx*XXX_bbb*_tsip_user_	セキュアアップデート後のユーザプログラム

program *1 *2	
rx65n_2mb_rsk_tsip_aes_sample	RX65N 用 AES 暗号プロジェクト
rx72n_ek_tsip_aes_sample	RX72N 用 AES 暗号プロジェクト
tool	ツールフォルダ
renesas_secure_flash_programmer \Renesas Secure Flash Programmer\bin\Debug	鍵とユーザプログラムに対し暗号化するツールとそのソースコード
Renesas Secure Flash Programmer.exe	鍵とユーザプログラムに対し暗号化するツール

*1：rx*XXX* には、サポートしている RX グループ名が入ります。

*2：*bbb* には、サポートしているボード名が入ります。通常 Update の IF として USB と UART を用意していますが、USB をサポートしていない MCU の場合、UART のみサポートします。UART のみサポートしているプロジェクトには、_secure_boot および_user_program の前に_sci も入ります。

RSK：rsk (UART のみサポート：rsk(_tsip)_sci)

MCB：mcb (UART のみサポート：mcb(_tsip)_sci)

Envision Kit：ek

r_tsip_rx_v.1.21.zip を解凍したフォルダには、以下の表 1-3 のファイルが含まれます。

表 1-3　ファイル構成

ファイル/ディレクトリ**(太字)**名	内容
r_config	TSIP ドライバコンフィグファイルフォルダ
r_tsip_rx_config.h	TSIP ドライバコンフィグファイル(デフォルト設定)
r_tsip_rx	TSIP ドライバ FIT Module フォルダ
src	TSIP ドライバソースコードフォルダ
targets	マイコン機種依存部分のプログラムコード格納用フォルダ
グループ名 *1	TSIP/TSIP-Lite ドライバフォルダ ファミリ毎にフォルダが存在します。
r_tsip_rx.c	TSIP システム部ドライバソースコード(必須)
r_tsip_*{アルゴリズム}*_rx.c	暗号アルゴリズム別 API ソースコード *{アルゴリズム}*には、aes,rsa などのアルゴリズム名が入ります。
r_tsip_rx_private.c	TSIP ドライバソースコード(オプションで ON/OFF 可能)
r_tsip_rx_private.h	TSIP ドライバヘッダファイル(オプションで ON/OFF 可能)
iodefine	TSIP アクセス用ヘッダファイル格納フォルダ
r_tsip_*{グループ名}*_iodefine.h	TSIP アクセス用ヘッダファイル グループ名にはフォルダ名と同じ名前が入ります。
ip	TSIP アクセス用ソースコード格納フォルダ
r_tsip_rx_pxx.c	TSIP アクセス用ソースコード ファイル名の xx には数値が入ります。
r_tsip_rx_subprcxx.c	TSIP アクセス用ソースコード ファイル名の xx には数値が入ります。
r_tsip_rx_functionxxx.c	TSIP ドライバソースコード(サブ API) ファイル名の xx には数値が入ります。
s_flash.c	鍵情報ファイル

	doc	
	ja	TSIP システム部ドライバソースコード(必須)
	r20an0371jj0121-rx-tsip-security.pdf	TSIP ドライバ アプリケーションノート(日本語)
	En	TSIP ドライバソースコード(オプションで ON/OFF 可能)
	r20an0371ej0121-rx-tsip-security.pdf	TSIP ドライバ アプリケーションノート(英語)
	r_tsip_rx_if.h	TSIP ドライバヘッダファイル
	readme.txt	Readme

1.4　開発環境

TSIP ドライバは以下の開発環境を用いて開発しました。ユーザアプリケーション開発時は以下のバージョン、またはより新しいものをご使用ください。

(1)統合開発環境

「7.1 動作確認環境」の項目「統合開発環境」を参照してください。

(2)C コンパイラ

「7.1 動作確認環境」の項目「C コンパイラ」を参照してください。

(3)エミュレータデバッガ

E2 Lite

(4)評価ボード

「7.1 動作確認環境」の項目「使用ボード」を参照してください。

いずれも、暗号機能付きの特別版の製品です。

製品型名をよくご確認の上、ご購入ください。

評価およびデモプロジェクト作成は、e^2 studio と CC-RX の組合せで実施しました。

プロジェクト変換機能で e^2 studio から CS+への変換が可能ですが、コンパイルエラー等問題が発生する場合はお問い合わせください。

1.5　コードサイズ

本モジュールの ROM サイズ、RAM サイズ、最大使用スタックサイズを下表に示します。

ROM (コードおよび定数) と RAM (グローバルデータ) のサイズは、ビルド時の「2.6 コンフィグレーション」のコンフィギュレーションオプションによって決まります。

下表の値は下記条件で確認しています。

モジュールリビジョン: r_tsip_rx rev1.21

コンパイラバージョン: Renesas Electronics C/C++ Compiler Package for RX Family V3.06.00
(統合開発環境のデフォルト設定に”-lang = c99”オプションを追加)
GCC for Renesas RX 8.3.0.202311
(統合開発環境のデフォルト設定に” -std=gnu99”オプションを追加)
IAR C/C++ Compiler for Renesas RX version 5.10.01
(統合開発環境のデフォルト設定)

コンフィグレーションオプション: デフォルト設定

ROM、RAM およびスタックのコードサイズ

デバイス	分類	使用メモリ		
		Renesas Compiler	GCC【注】	IAR Compiler
TSIP-Lite	ROM	**56,559 バイト**	**55,548 バイト**	**58,546 バイト**
	RAM	**1,364 バイト**	**1,364 バイト**	**1,364 バイト**
	スタック	**184 バイト**	**-**	**164 バイト**
TSIP	ROM	**453,129 バイト**	**435,069 バイト**	**451,349 バイト**
	RAM	**7,988 バイト**	**7,988 バイト**	**7,988s バイト**
	スタック	**1,648 バイト**	**-**	**1,376 バイト**

【注】スタック解析が行えない項目については、”-”を記載しています。

上記の表は、TSIP ドライバの全ての機能を有効にした際のコードサイズです。実際に使用する際のコードサイズの参考情報として、RX231 と RX65N で各機能のみを有効にした場合の ROM サイズ、RAM サイズ、最大使用スタックサイズを下表に示します。ここで、それぞれの API はいくつかの内部関数を重複して使用しているため、表記のサイズの合計とビルドサイズは異なります。

	ROM、RAM およびスタックのコードサイズ				
デバイス	機能	分類	使用メモリ		
			Renesas Compiler	GCC【注】	IAR Compiler【注】
RX231	共通機能 API	ROM	**5,201 バイト**	**5,617 バイト**	**5,382 バイト**
		RAM	**1,356 バイト**	**1,356 バイト**	**1,356 バイト**
		スタック	**80 バイト**	**-**	**-**
	乱数生成	ROM	**902 バイト**	**427 バイト**	**419 バイト**
		RAM	**0 バイト**	**0 バイト**	**0 バイト**
		スタック	**28 バイト**	**-**	**20 バイト**
	AES(128bit)	ROM	**23,700 バイト**	**23,402 バイト**	**25,034 バイト**
		RAM	**112 バイト**	**112 バイト**	**112 バイト**
		スタック	**184 バイト**	**-**	**164 バイト**
	AES(256bit)	ROM	**24,733 バイト**	**24,433 バイト**	**26,125 バイト**
		RAM	**112 バイト**	**192 バイト**	**112 バイト**
		スタック	**184 バイト**	**-**	**164 バイト**
	KeyWrap	ROM	**6,886 バイト**	**5,947 バイト**	**6,239 バイト**
		RAM	**0 バイト**	**0 バイト**	**0 バイト**
		スタック	**116 バイト**	**-**	**68 バイト**
	ファームウェアアップデート	ROM	**3,267 バイト**	**2,427 バイト**	**2,560 バイト**
		RAM	**8 バイト**	**8 バイト**	**8 バイト**
		スタック	**92 バイト**	**-**	**-**
RX65N	共通機能 API	ROM	**8,373 バイト**	**8,464 バイト**	**8,526 バイト**
		RAM	**1,452 バイト**	**1,400 バイト**	**1,400 バイト**
		スタック	**84 バイト**	**-**	**-**
	乱数生成	ROM	**4,310 バイト**	**437 バイト**	**433 バイト**
		RAM	**52 バイト**	**0 バイト**	**0 バイト**
		スタック	**28 バイト**	**-**	**20 バイト**
	AES(128bit)	ROM	**37,456 バイト**	**36,634 バイト**	**39,346 バイト**
		RAM	**244 バイト**	**192 バイト**	**192 バイト**
		スタック	**192 バイト**	**-**	**160 バイト**
	AES(256bit)	ROM	**30,455 バイト**	**24,433 バイト**	**31,653 バイト**
		RAM	**244 バイト**	**112 バイト**	**192 バイト**
		スタック	**192 バイト**	**-**	**160 バイト**
	DES	ROM	**10,261 バイト**	**10,030 バイト**	**10,383 バイト**

		RAM	**136 バイト**	**84 バイト**	**84 バイト**
		スタック	**68 バイト**	**-**	**48 バイト**
	ARC4	ROM	**8,932 バイト**	**8,829 バイト**	**9,088 バイト**
		RAM	**128 バイト**	**76 バイト**	**76 バイト**
		スタック	**68 バイト**	**-**	**40 バイト**
	RSA	ROM	**136,236 バイト**	**122,378 バイト**	**133,269 バイト**
		RAM	**6,724 バイト**	**6,724 バイト**	**6,724 バイト**
		スタック	**1,632 バイト**	**-**	**1,376 バイト**
	ECC	ROM	**41,867 バイト**	**45,974 バイト**	**43,496 バイト**
		RAM	**6,724 バイト**	**6,672 バイト**	**6,672 バイト**
		スタック	**360 バイト**	**-**	**340 バイト**
	HASH	ROM	**6,818 バイト**	**2,922 バイト**	**3,152 バイト**
		RAM	**52 バイト**	**0 バイト**	**0 バイト**
		スタック	**244 バイト**	**-**	**236 バイト**
	HMAC	ROM	**16,111 バイト**	**15,940 バイト**	**16,470 バイト**
		RAM	**216 バイト**	**164 バイト**	**164 バイト**
		スタック	**124 バイト**	**-**	**84 バイト**
	DH	ROM	**6,701 バイト**	**2,533 バイト**	**2,709 バイト**
		RAM	**52 バイト**	**0 バイト**	**0 バイト**
		スタック	**84 バイト**	**-**	**40 バイト**
	ECDH	ROM	**30,027 バイト**	**33,149 バイト**	**30,876 バイト**
		RAM	**6,668 バイト**	**6,616 バイト**	**6,616 バイト**
		スタック	**308 バイト**	**-**	**232 バイト**
	KeyWrap	ROM	**9,917 バイト**	**5,441 バイト**	**5,857 バイト**
		RAM	**52 バイト**	**0 バイト**	**0 バイト**
		スタック	**124 バイト**	**-**	**76 バイト**
	TLS(TLS1.2/1.3 共通)	ROM	**76,494 バイト**	**71,122 バイト**	**74,370 バイト**
		RAM	**7,844 バイト**	**7,792 バイト**	**7,792 バイト**
		スタック	**740 バイト**	**-**	**124 バイト**
	TLS(TLS1.2)	ROM	**136,486 バイト**	**131,468 バイト**	**135,669 バイト**
		RAM	**7,844 バイト**	**7,792 バイト**	**7,792 バイト**
		スタック	**812 バイト**	**-**	**184 バイト**
	TLS(TLS1.3)	ROM	**156,556 バイト**	**151,236 バイト**	**154,922 バイト**

		RAM	**6,680 バイト**	**6,628 バイト**	**6,628 バイト**
		スタック	**1,248 バイト**	**-**	**1,208 バイト**
	ファームウェアアップデート	ROM	**6,407 バイト**	**2,383 バイト**	**2,565 バイト**
		RAM	**60 バイト**	**8 バイト**	**8 バイト**
		スタック	**84 バイト**	**-**	**-**

【注】スタック解析が行えない項目については、”-”を記載しています。

1.6　セクション情報

TSIP ドライバはデフォルトセクションを使用します。

サンプルプログラムでは、C_FIRMWARE_UPDATE_CONTROL_BLOCK、C_FIRMWARE_UPDATE_CONTROL_BLOCK_MIRROR を使用します。コンパイラを CC-RX に設定し、Smart Configurator を使用して TSIP ドライバをプロジェクトに追加した際には、これらのセクションが設定されます。更に C_ENCRYPTED_KEY_BLOCK を使用します。変更が必要な場合は、セクション設定を編集して使用してください。

セキュアブート機能を使用する場合、BSECURE_BOOT*、PSECURE_BOOT、PSECURE_BOOT_ERASE、CSECURE_BOOT*、DSECURE_BOOT*、RSECURE_BOOT*を使用します。

1.7　性能情報

以下に各デバイスグループの TSIP-Lite ドライバ (RX231, RX23W, RX26T, RX66T, RX72T) および TSIP ドライバ (RX65N, RX671, RX72M, RX72N) の性能情報を示します。

性能はコアクロックである ICLK のサイクル単位での計測になります。TSIP-Lite および TSIP の動作クロック PCLKB は ICLK : PCLKB = 2 : 1 の設定をしています。ドライバは CC-RX、最適化レベル 2 でビルドしています。バージョンは「7.1 動作確認環境」をご参照ください。コンフィグレーションオプションはデフォルト設定です。

1.7.1　RX231

表 1-4　共通 API の性能

API	性能 (単位：サイクル)
R_TSIP_Open	7,400,000
R_TSIP_Close	460
R_TSIP_GetVersion	30
R_TSIP_GenerateAes128KeyIndex	4,000
R_TSIP_GenerateAes256KeyIndex	4,600
R_TSIP_GenerateAes128RandomKeyIndex	2,300
R_TSIP_GenerateAes256RandomKeyIndex	3,100
R_TSIP_GenerateRandomNumber	950
R_TSIP_GenerateUpdateKeyRingKeyIndex	4,600
R_TSIP_UpdateAes128KeyIndex	3,600
R_TSIP_UpdateAes256KeyIndex	4,200

表 1-5　Firmware 検証の性能

API	性能 (単位：サイクル)		
	2K バイト処理	4K バイト処理	6K バイト処理
R_TSIP_VerifyFirmwareMAC	13,000	24,000	35,000

表 1-6　AES の性能

API	性能 (単位：サイクル)		
	16 バイト処理	48 バイト処理	80 バイト処理
R_TSIP_Aes128EcbEncryptInit	1,400	1,400	1,400
R_TSIP_Aes128EcbEncryptUpdate	620	800	970
R_TSIP_Aes128EcbEncryptFinal	570	570	570
R_TSIP_Aes128EcbDecryptInit	1,400	1,400	1,400
R_TSIP_Aes128EcbDecryptUpdate	740	920	1,100
R_TSIP_Aes128EcbDecryptFinal	580	580	580
R_TSIP_Aes256EcbEncryptInit	1,700	1,700	1,700
R_TSIP_Aes256EcbEncryptUpdate	660	910	1,200
R_TSIP_Aes256EcbEncryptFinal	580	580	580
R_TSIP_Aes256EcbDecryptInit	1,700	1,700	1,700
R_TSIP_Aes256EcbDecryptUpdate	810	1,100	1,300
R_TSIP_Aes256EcbDecryptFinal	580	580	580
R_TSIP_Aes128CbcEncryptInit	1,400	1,400	1,400
R_TSIP_Aes128CbcEncryptUpdate	680	860	1,100
R_TSIP_Aes128CbcEncryptFinal	590	590	590
R_TSIP_Aes128CbcDecryptInit	1,400	1,400	1,400
R_TSIP_Aes128CbcDecryptUpdate	800	970	1,200
R_TSIP_Aes128CbcDecryptFinal	600	600	600
R_TSIP_Aes256CbcEncryptInit	1,700	1,700	1,700
R_TSIP_Aes256CbcEncryptUpdate	720	960	1,300
R_TSIP_Aes256CbcEncryptFinal	600	600	600
R_TSIP_Aes256CbcDecryptInit	1,700	1,700	1,700
R_TSIP_Aes256CbcDecryptUpdate	860	1,100	1,400
R_TSIP_Aes256CbcDecryptFinal	610	610	610

表 1-7　AES-GCM の性能

API	性能 (単位：サイクル)		
	48 バイト処理	64 バイト処理	80 バイト処理
R_TSIP_Aes128GcmEncryptInit	5,500	5,500	5,500
R_TSIP_Aes128GcmEncryptUpdate	2,900	3,400	3,900
R_TSIP_Aes128GcmEncryptFinal	1,300	1,300	1,300
R_TSIP_Aes128GcmDecryptInit	5,500	5,500	5,500
R_TSIP_Aes128GcmDecryptUpdate	2,500	2,600	2,700
R_TSIP_Aes128GcmDecryptFinal	2,100	2,100	2,100
R_TSIP_Aes256GcmEncryptInit	6,200	6,200	6,200
R_TSIP_Aes256GcmEncryptUpdate	3,000	3,500	4,100
R_TSIP_Aes256GcmEncryptFinal	1,400	1,400	1,400
R_TSIP_Aes256GcmDecryptInit	6,200	6,200	6,200
R_TSIP_Aes256GcmDecryptUpdate	2,600	2,700	2,800
R_TSIP_Aes256GcmDecryptFinal	2,200	2,200	2,200

GCM の性能は、ivec を 1024bit、追加認証データを 720bit、認証タグを 128bit に固定して計測しました。

表 1-8　AES-CCM の性能

API	性能 (単位：サイクル)		
	48 バイト処理	64 バイト処理	80 バイト処理
R_TSIP_Aes128CcmEncryptInit	2,700	2,700	2,700
R_TSIP_Aes128CcmEncryptUpdate	1,600	1,700	1,900
R_TSIP_Aes128CcmEncryptFinal	1,200	1,200	1,200
R_TSIP_Aes128CcmDecryptInit	2,500	2,500	2,500
R_TSIP_Aes128CcmDecryptUpdate	1,500	1,700	1,800
R_TSIP_Aes128CcmDecryptFinal	2,000	2,000	2,000
R_TSIP_Aes256CcmEncryptInit	3,000	3,000	3,000
R_TSIP_Aes256CcmEncryptUpdate	1,800	2,000	2,300
R_TSIP_Aes256CcmEncryptFinal	1,300	1,300	1,300
R_TSIP_Aes256CcmDecryptInit	3,000	3,000	3,000
R_TSIP_Aes256CcmDecryptUpdate	1,700	1,900	2,200
R_TSIP_Aes256CcmDecryptFinal	2,000	2,000	2,000

CCM の性能は、ノンスを 104bit、追加認証データを 880bit、MAC を 128bit に固定して計測しました。

表 1-9　AES-CMAC の性能

API	性能 (単位：サイクル)		
	48 バイト処理	64 バイト処理	80 バイト処理
R_TSIP_Aes128CmacGenerateInit	920	920	920
R_TSIP_Aes128CmacGenerateUpdate	820	900	990
R_TSIP_Aes128CmacGenerateFinal	1,100	1,100	1,100
R_TSIP_Aes128CmacVerifyInit	910	920	920
R_TSIP_Aes128CmacVerifyUpdate	820	910	1,000
R_TSIP_Aes128CmacVerifyFinal	1,800	1,800	1,800
R_TSIP_Aes256CmacGenerateInit	1,300	1,300	1,300
R_TSIP_Aes256CmacGenerateUpdate	880	1,100	1,200
R_TSIP_Aes256CmacGenerateFinal	1,200	1,200	1,200
R_TSIP_Aes256CmacVerifyInit	1,300	1,300	1,300
R_TSIP_Aes256CmacVerifyUpdate	890	1,100	1,200
R_TSIP_Aes256CmacVerifyFinal	1,900	1,900	1,900

表 1-10　AES Key Wrap の性能

API	性能 (単位：サイクル)	
	ラップ対象鍵 AES-128	ラップ対象鍵 AES-256
R_TSIP_Aes128KeyWrap	9,600	16,000
R_TSIP_Aes256KeyWrap	11,000	17,000
R_TSIP_Aes128KeyUnwrap	12,000	18,000
R_TSIP_Aes256KeyUnwrap	13,000	19,000

1.7.2　RX23W

表 1-11　共通 API の性能

API	性能 (単位：サイクル)
R_TSIP_Open	7,400,000
R_TSIP_Close	690
R_TSIP_GetVersion	40
R_TSIP_GenerateAes128KeyIndex	4,400
R_TSIP_GenerateAes256KeyIndex	5,000
R_TSIP_GenerateAes128RandomKeyIndex	2,500
R_TSIP_GenerateAes256RandomKeyIndex	3,400
R_TSIP_GenerateRandomNumber	1,100
R_TSIP_GenerateUpdateKeyRingKeyIndex	5,000
R_TSIP_UpdateAes128KeyIndex	3,900
R_TSIP_UpdateAes256KeyIndex	4,500

表 1-12　Firmware 検証の性能

API	性能 (単位：サイクル)		
	2K バイト処理	4K バイト処理	6K バイト処理
R_TSIP_VerifyFirmwareMAC	13,000	24,000	35,000

表 1-13　AES の性能

API	性能 (単位：サイクル)		
	16 バイト処理	48 バイト処理	80 バイト処理
R_TSIP_Aes128EcbEncryptInit	1,500	1,500	1,500
R_TSIP_Aes128EcbEncryptUpdate	750	930	1,200
R_TSIP_Aes128EcbEncryptFinal	660	660	660
R_TSIP_Aes128EcbDecryptInit	1,600	1,600	1,600
R_TSIP_Aes128EcbDecryptUpdate	860	1,100	1,300
R_TSIP_Aes128EcbDecryptFinal	670	670	670
R_TSIP_Aes256EcbEncryptInit	1,900	1,900	1,900
R_TSIP_Aes256EcbEncryptUpdate	780	1,100	1,300
R_TSIP_Aes256EcbEncryptFinal	670	670	670
R_TSIP_Aes256EcbDecryptInit	1,900	1,900	1,900
R_TSIP_Aes256EcbDecryptUpdate	930	1,200	1,500
R_TSIP_Aes256EcbDecryptFinal	690	690	690
R_TSIP_Aes128CbcEncryptInit	1,600	1,600	1,600
R_TSIP_Aes128CbcEncryptUpdate	820	1,100	1,200
R_TSIP_Aes128CbcEncryptFinal	690	690	690
R_TSIP_Aes128CbcDecryptInit	1,600	1,600	1,600
R_TSIP_Aes128CbcDecryptUpdate	930	1,200	1,300
R_TSIP_Aes128CbcDecryptFinal	700	700	700
R_TSIP_Aes256CbcEncryptInit	1,900	1,900	1,900
R_TSIP_Aes256CbcEncryptUpdate	860	1,100	1,400
R_TSIP_Aes256CbcEncryptFinal	700	700	700
R_TSIP_Aes256CbcDecryptInit	1,900	2,000	2,000
R_TSIP_Aes256CbcDecryptUpdate	1,000	1,300	1,500
R_TSIP_Aes256CbcDecryptFinal	720	720	720

表 1-14　AES-GCM の性能

API	性能 (単位：サイクル)		
	48 バイト処理	64 バイト処理	80 バイト処理
R_TSIP_Aes128GcmEncryptInit	6,300	6,300	6,300
R_TSIP_Aes128GcmEncryptUpdate	3,400	4,000	4,500
R_TSIP_Aes128GcmEncryptFinal	1,500	1,500	1,500
R_TSIP_Aes128GcmDecryptInit	6,300	6,300	6,300
R_TSIP_Aes128GcmDecryptUpdate	2,900	3,000	3,100
R_TSIP_Aes128GcmDecryptFinal	2,400	2,400	2,400
R_TSIP_Aes256GcmEncryptInit	7,000	7,000	7,000
R_TSIP_Aes256GcmEncryptUpdate	3,500	4,100	4,700
R_TSIP_Aes256GcmEncryptFinal	1,600	1,600	1,600
R_TSIP_Aes256GcmDecryptInit	7,000	7,000	7,000
R_TSIP_Aes256GcmDecryptUpdate	3,000	3,100	3,300
R_TSIP_Aes256GcmDecryptFinal	2,400	2,400	2,400

GCM の性能は、ivec を 1024bit、追加認証データを 720bit、認証タグを 128bit に固定して計測しました。

表 1-15　AES-CCM の性能

API	性能 (単位：サイクル)		
	48 バイト処理	64 バイト処理	80 バイト処理
R_TSIP_Aes128CcmEncryptInit	3,100	3,100	3,100
R_TSIP_Aes128CcmEncryptUpdate	1,800	2,000	2,200
R_TSIP_Aes128CcmEncryptFinal	1,500	1,500	1,500
R_TSIP_Aes128CcmDecryptInit	2,800	2,800	2,800
R_TSIP_Aes128CcmDecryptUpdate	1,700	1,900	2,100
R_TSIP_Aes128CcmDecryptFinal	2,300	2,300	2,300
R_TSIP_Aes256CcmEncryptInit	3,400	3,400	3,400
R_TSIP_Aes256CcmEncryptUpdate	2,000	2,300	2,500
R_TSIP_Aes256CcmEncryptFinal	1,500	1,500	1,500
R_TSIP_Aes256CcmDecryptInit	3,400	3,400	3,400
R_TSIP_Aes256CcmDecryptUpdate	1,900	2,200	2,400
R_TSIP_Aes256CcmDecryptFinal	2,300	2,300	2,300

CCM の性能は、ノンスを 104bit、追加認証データを 880bit、MAC を 128bit に固定して計測しました。

表 1-16　AES-CMAC の性能

API	性能 (単位：サイクル)		
	48 バイト処理	64 バイト処理	80 バイト処理
R_TSIP_Aes128CmacGenerateInit	1,100	1,100	1,100
R_TSIP_Aes128CmacGenerateUpdate	960	1,100	1,200
R_TSIP_Aes128CmacGenerateFinal	1,300	1,300	1,300
R_TSIP_Aes128CmacVerifyInit	1,100	1,100	1,100
R_TSIP_Aes128CmacVerifyUpdate	950	1,100	1,200
R_TSIP_Aes128CmacVerifyFinal	2,100	2,100	2,100
R_TSIP_Aes256CmacGenerateInit	1,400	1,400	1,400
R_TSIP_Aes256CmacGenerateUpdate	1,100	1,200	1,300
R_TSIP_Aes256CmacGenerateFinal	1,400	1,400	1,400
R_TSIP_Aes256CmacVerifyInit	1,400	1,400	1,400
R_TSIP_Aes256CmacVerifyUpdate	1,100	1,200	1,300
R_TSIP_Aes256CmacVerifyFinal	2,200	2,200	2,200

表 1-17　AES Key Wrap の性能

API	性能 (単位：サイクル)	
	ラップ対象鍵 AES-128	ラップ対象鍵 AES-256
R_TSIP_Aes128KeyWrap	11,000	17,000
R_TSIP_Aes256KeyWrap	12,000	18,000
R_TSIP_Aes128KeyUnwrap	14,000	20,000
R_TSIP_Aes256KeyUnwrap	15,000	21,000

1.7.3　RX26T

表 1-18　共通 API の性能

API	性能 (単位：サイクル)
R_TSIP_Open	7,400,000
R_TSIP_Close	280
R_TSIP_GetVersion	20
R_TSIP_GenerateAes128KeyIndex	4,000
R_TSIP_GenerateAes256KeyIndex	4,500
R_TSIP_GenerateAes128RandomKeyIndex	2,200
R_TSIP_GenerateAes256RandomKeyIndex	3,000
R_TSIP_GenerateRandomNumber	910
R_TSIP_GenerateUpdateKeyRingKeyIndex	4,500
R_TSIP_UpdateAes128KeyIndex	3,500
R_TSIP_UpdateAes256KeyIndex	4,100

表 1-19　Firmware 検証の性能

API	性能 (単位：サイクル)		
	2K バイト処理	4K バイト処理	6K バイト処理
R_TSIP_VerifyFirmwareMAC	12,000	23,000	34,000

表 1-20　AES の性能

API	性能 (単位：サイクル)		
	16 バイト処理	48 バイト処理	80 バイト処理
R_TSIP_Aes128EcbEncryptInit	1,300	1,300	1,300
R_TSIP_Aes128EcbEncryptUpdate	560	740	910
R_TSIP_Aes128EcbEncryptFinal	510	510	510
R_TSIP_Aes128EcbDecryptInit	1,300	1,300	1,300
R_TSIP_Aes128EcbDecryptUpdate	680	850	1,000
R_TSIP_Aes128EcbDecryptFinal	530	530	530
R_TSIP_Aes256EcbEncryptInit	1,600	1,600	1,600
R_TSIP_Aes256EcbEncryptUpdate	610	840	1,100
R_TSIP_Aes256EcbEncryptFinal	500	500	510
R_TSIP_Aes256EcbDecryptInit	1,600	1,600	1,600
R_TSIP_Aes256EcbDecryptUpdate	750	980	1,200
R_TSIP_Aes256EcbDecryptFinal	520	520	520
R_TSIP_Aes128CbcEncryptInit	1,300	1,300	1,300
R_TSIP_Aes128CbcEncryptUpdate	600	790	960
R_TSIP_Aes128CbcEncryptFinal	540	540	540
R_TSIP_Aes128CbcDecryptInit	1,300	1,300	1,300
R_TSIP_Aes128CbcDecryptUpdate	720	900	1,100
R_TSIP_Aes128CbcDecryptFinal	550	550	550
R_TSIP_Aes256CbcEncryptInit	1,600	1,600	1,600
R_TSIP_Aes256CbcEncryptUpdate	640	900	1,100
R_TSIP_Aes256CbcEncryptFinal	530	530	530
R_TSIP_Aes256CbcDecryptInit	1,600	1,600	1,600
R_TSIP_Aes256CbcDecryptUpdate	780	1,000	1,300
R_TSIP_Aes256CbcDecryptFinal	540	540	540

表 1-21　AES-GCM の性能

API	性能 (単位：サイクル)		
	48 バイト処理	64 バイト処理	80 バイト処理
R_TSIP_Aes128GcmEncryptInit	5,100	5,100	5,100
R_TSIP_Aes128GcmEncryptUpdate	2,600	3,100	3,600
R_TSIP_Aes128GcmEncryptFinal	1,200	1,200	1,200
R_TSIP_Aes128GcmDecryptInit	5,100	5,100	5,100
R_TSIP_Aes128GcmDecryptUpdate	2,200	2,300	2,400
R_TSIP_Aes128GcmDecryptFinal	2,000	2,000	2,000
R_TSIP_Aes256GcmEncryptInit	5,800	5,800	5,800
R_TSIP_Aes256GcmEncryptUpdate	2,700	3,200	3,700
R_TSIP_Aes256GcmEncryptFinal	1,300	1,300	1,300
R_TSIP_Aes256GcmDecryptInit	5,800	5,800	5,800
R_TSIP_Aes256GcmDecryptUpdate	2,300	2,400	2,500
R_TSIP_Aes256GcmDecryptFinal	2,000	2,000	2,000

GCM の性能は、ivec を 1024bit、追加認証データを 720bit、認証タグを 128bit に固定して計測しました。

表 1-22　AES-CCM の性能

API	性能 (単位：サイクル)		
	48 バイト処理	64 バイト処理	80 バイト処理
R_TSIP_Aes128CcmEncryptInit	2,500	2,500	2,500
R_TSIP_Aes128CcmEncryptUpdate	1,400	1,600	1,900
R_TSIP_Aes128CcmEncryptFinal	1,100	1,100	1,100
R_TSIP_Aes128CcmDecryptInit	2,200	2,200	2,200
R_TSIP_Aes128CcmDecryptUpdate	1,400	1,500	1,700
R_TSIP_Aes128CcmDecryptFinal	1,900	1,900	1,900
R_TSIP_Aes256CcmEncryptInit	2,800	2,800	2,800
R_TSIP_Aes256CcmEncryptUpdate	1,700	1,900	2,100
R_TSIP_Aes256CcmEncryptFinal	1,200	1,200	1,200
R_TSIP_Aes256CcmDecryptInit	2,800	2,800	2,800
R_TSIP_Aes256CcmDecryptUpdate	1,600	1,800	2,000
R_TSIP_Aes256CcmDecryptFinal	1,900	1,900	1,900

CCM の性能は、ノンスを 104bit、追加認証データを 880bit、MAC を 128bit に固定して計測しました。

表 1-23　AES-CMAC の性能

API	性能 (単位：サイクル)		
	48 バイト処理	64 バイト処理	80 バイト処理
R_TSIP_Aes128CmacGenerateInit	870	870	870
R_TSIP_Aes128CmacGenerateUpdate	720	810	900
R_TSIP_Aes128CmacGenerateFinal	1,000	1,000	1,000
R_TSIP_Aes128CmacVerifyInit	870	880	880
R_TSIP_Aes128CmacVerifyUpdate	720	810	900
R_TSIP_Aes128CmacVerifyFinal	1,700	1,700	1,700
R_TSIP_Aes256CmacGenerateInit	1,200	1,200	1,200
R_TSIP_Aes256CmacGenerateUpdate	800	920	1,000
R_TSIP_Aes256CmacGenerateFinal	1,100	1,100	1,100
R_TSIP_Aes256CmacVerifyInit	1,200	1,200	1,200
R_TSIP_Aes256CmacVerifyUpdate	790	910	1,000
R_TSIP_Aes256CmacVerifyFinal	1,800	1,800	1,800

表 1-24　AES Key Wrap の性能

API	性能 (単位：サイクル)	
	ラップ対象鍵 AES-128	ラップ対象鍵 AES-256
R_TSIP_Aes128KeyWrap	9,400	15,000
R_TSIP_Aes256KeyWrap	10,000	16,000
R_TSIP_Aes128KeyUnwrap	12,000	17,000
R_TSIP_Aes256KeyUnwrap	12,000	18,000

1.7.4　RX66T, RX72T

表 1-25　共通 API の性能

API	性能 (単位：サイクル)
R_TSIP_Open	7,400,000
R_TSIP_Close	290
R_TSIP_GetVersion	22
R_TSIP_GenerateAes128KeyIndex	4,000
R_TSIP_GenerateAes256KeyIndex	4,500
R_TSIP_GenerateAes128RandomKeyIndex	2,200
R_TSIP_GenerateAes256RandomKeyIndex	3,000
R_TSIP_GenerateRandomNumber	910
R_TSIP_GenerateUpdateKeyRingKeyIndex	4,500
R_TSIP_UpdateAes128KeyIndex	3,500
R_TSIP_UpdateAes256KeyIndex	4,100

表 1-26　Firmware 検証の性能

API	性能 (単位：サイクル)		
	2K バイト処理	4K バイト処理	6K バイト処理
R_TSIP_VerifyFirmwareMAC	12,000	24,000	35,000

表 1-27　AES の性能

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

表 1-28　AES-GCM の性能

API	性能 (単位：サイクル)		
	48 バイト処理	64 バイト処理	80 バイト処理
R_TSIP_Aes128GcmEncryptInit	5,200	5,200	5,200
R_TSIP_Aes128GcmEncryptUpdate	2,700	3,100	3,600
R_TSIP_Aes128GcmEncryptFinal	1,300	1,300	1,300
R_TSIP_Aes128GcmDecryptInit	5,200	5,200	5,200
R_TSIP_Aes128GcmDecryptUpdate	2,300	2,300	2,400
R_TSIP_Aes128GcmDecryptFinal	2,100	2,100	2,100
R_TSIP_Aes256GcmEncryptInit	5,900	5,900	5,900
R_TSIP_Aes256GcmEncryptUpdate	2,800	3,300	3,800
R_TSIP_Aes256GcmEncryptFinal	1,300	1,300	1,300
R_TSIP_Aes256GcmDecryptInit	5,900	5,900	5,900
R_TSIP_Aes256GcmDecryptUpdate	2,400	2,500	2,600
R_TSIP_Aes256GcmDecryptFinal	2,100	2,100	2,100

GCM の性能は、ivec を 1024bit、追加認証データを 720bit、認証タグを 128bit に固定して計測しました。

表 1-29　AES-CCM の性能

API	性能 (単位：サイクル)		
	48 バイト処理	64 バイト処理	80 バイト処理
R_TSIP_Aes128CcmEncryptInit	2,500	2,500	2,500
R_TSIP_Aes128CcmEncryptUpdate	1,500	1,700	1,900
R_TSIP_Aes128CcmEncryptFinal	1,200	1,200	1,200
R_TSIP_Aes128CcmDecryptInit	2,300	2,300	2,300
R_TSIP_Aes128CcmDecryptUpdate	1,400	1,600	1,800
R_TSIP_Aes128CcmDecryptFinal	1,900	1,900	1,900
R_TSIP_Aes256CcmEncryptInit	2,900	2,900	2,900
R_TSIP_Aes256CcmEncryptUpdate	1,700	2,000	2,200
R_TSIP_Aes256CcmEncryptFinal	1,200	1,200	1,200
R_TSIP_Aes256CcmDecryptInit	2,900	2,900	2,900
R_TSIP_Aes256CcmDecryptUpdate	1,600	1,900	2,100
R_TSIP_Aes256CcmDecryptFinal	2,000	2,000	2,000

CCM の性能は、ノンスを 104bit、追加認証データを 880bit、MAC を 128bit に固定して計測しました。

表 1-30　AES-CMAC の性能

API	性能 (単位：サイクル)		
	48 バイト処理	64 バイト処理	80 バイト処理
R_TSIP_Aes128CmacGenerateInit	890	880	880
R_TSIP_Aes128CmacGenerateUpdate	730	810	900
R_TSIP_Aes128CmacGenerateFinal	1,100	1,100	1,100
R_TSIP_Aes128CmacVerifyInit	880	880	880
R_TSIP_Aes128CmacVerifyUpdate	720	810	900
R_TSIP_Aes128CmacVerifyFinal	1,800	1,800	1,800
R_TSIP_Aes256CmacGenerateInit	1,200	1,200	1,200
R_TSIP_Aes256CmacGenerateUpdate	800	930	1,100
R_TSIP_Aes256CmacGenerateFinal	1,200	1,200	1,200
R_TSIP_Aes256CmacVerifyInit	1,200	1,200	1,200
R_TSIP_Aes256CmacVerifyUpdate	800	93s0	1,100
R_TSIP_Aes256CmacVerifyFinal	1,800	1,800	1,800

表 1-31　AES Key Wrap の性能

API	性能 (単位：サイクル)	
	ラップ対象鍵 AES-128	ラップ対象鍵 AES-256
R_TSIP_Aes128KeyWrap	9,400	16,000
R_TSIP_Aes256KeyWrap	11,000	17,000
R_TSIP_Aes128KeyUnwrap	12,000	18,000
R_TSIP_Aes256KeyUnwrap	13,000	19,000

1.7.5　RX65N

表 1-32　共通 API の性能

API	性能 (単位：サイクル)
R_TSIP_Open	5,800,000
R_TSIP_Close	460
R_TSIP_GetVersion	30
R_TSIP_GenerateAes128KeyIndex	2,700
R_TSIP_GenerateAes256KeyIndex	2,800
R_TSIP_GenerateAes128RandomKeyIndex	1,500
R_TSIP_GenerateAes256RandomKeyIndex	2,100
R_TSIP_GenerateRandomNumber	670
R_TSIP_GenerateUpdateKeyRingKeyIndex	2,800
R_TSIP_UpdateAes128KeyIndex	2,300
R_TSIP_UpdateAes256KeyIndex	2,400

表 1-33　Firmware 検証の性能

API	性能 (単位：サイクル)		
	8K バイト処理	16K バイト処理	24K バイト処理
R_TSIP_VerifyFirmwareMAC	22,000	42,000	63,000

表 1-34　AES の性能

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

表 1-35　AES-GCM の性能

API	性能 (単位：サイクル)		
	48 バイト処理	64 バイト処理	80 バイト処理
R_TSIP_Aes128GcmEncryptInit	5,600	5,600	5,600
R_TSIP_Aes128GcmEncryptUpdate	2,100	2,200	2,300
R_TSIP_Aes128GcmEncryptFinal	1,400	1,400	1,400
R_TSIP_Aes128GcmDecryptInit	5,500	5,500	5,500
R_TSIP_Aes128GcmDecryptUpdate	2,100	2,200	2,300
R_TSIP_Aes128GcmDecryptFinal	2,300	2,300	2,300
R_TSIP_Aes256GcmEncryptInit	5,500	5,500	5,500
R_TSIP_Aes256GcmEncryptUpdate	2,200	2,300	2,400
R_TSIP_Aes256GcmEncryptFinal	1,100	1,100	1,100
R_TSIP_Aes256GcmDecryptInit	5,500	5,500	5,500
R_TSIP_Aes256GcmDecryptUpdate	2,200	2,300	2,300
R_TSIP_Aes256GcmDecryptFinal	2,000	2,000	2,000

GCM の性能は、ivec を 1024bit、追加認証データを 720bit、認証タグを 128bit に固定して計測しました。

表 1-36　AES-CCM の性能

API	性能 (単位：サイクル)		
	48 バイト処理	64 バイト処理	80 バイト処理
R_TSIP_Aes128CcmEncryptInit	3,100	3,100	3,100
R_TSIP_Aes128CcmEncryptUpdate	1,200	1,300	1,400
R_TSIP_Aes128CcmEncryptFinal	940	940	940
R_TSIP_Aes128CcmDecryptInit	3,200	3,200	3,200
R_TSIP_Aes128CcmDecryptUpdate	1,100	1,200	1,300
R_TSIP_Aes128CcmDecryptFinal	2,000	2,000	2,000
R_TSIP_Aes256CcmEncryptInit	2,400	2,400	2,400
R_TSIP_Aes256CcmEncryptUpdate	1,200	1,300	1,400
R_TSIP_Aes256CcmEncryptFinal	990	990	990
R_TSIP_Aes256CcmDecryptInit	2,400	2,400	2,400
R_TSIP_Aes256CcmDecryptUpdate	1,100	1,200	1,300
R_TSIP_Aes256CcmDecryptFinal	2,100	2,100	2,100

CCM の性能は、ノンスを 104bit、追加認証データを 880bit、MAC を 128bit に固定して計測しました。

表 1-37　AES-CMAC の性能

API	性能 (単位：サイクル)		
	48 バイト処理	64 バイト処理	80 バイト処理
R_TSIP_Aes128CmacGenerateInit	1,200	1,200	1,200
R_TSIP_Aes128CmacGenerateUpdate	670	720	760
R_TSIP_Aes128CmacGenerateFinal	800	800	800
R_TSIP_Aes128CmacVerifyInit	1,200	1,200	1,200
R_TSIP_Aes128CmacVerifyUpdate	680	720	770
R_TSIP_Aes128CmacVerifyFinal	1,700	1,700	1,700
R_TSIP_Aes256CmacGenerateInit	1,300	1,300	1,300
R_TSIP_Aes256CmacGenerateUpdate	720	760	810
R_TSIP_Aes256CmacGenerateFinal	830	830	830
R_TSIP_Aes256CmacVerifyInit	1,300	1,300	1,300
R_TSIP_Aes256CmacVerifyUpdate	710	750	810
R_TSIP_Aes256CmacVerifyFinal	1,800	1,800	1,800

表 1-38　AES Key Wrap の性能

API	性能 (単位：サイクル)	
	ラップ対象鍵 AES-128	ラップ対象鍵 AES-256
R_TSIP_Aes128KeyWrap	8,300	13,000
R_TSIP_Aes256KeyWrap	8,400	14,000
R_TSIP_Aes128KeyUnwrap	9,400	14,000
R_TSIP_Aes256KeyUnwrap	9,500	15,000

表 1-39　共通 API(TDES Wrapped Key 生成)の性能

API	性能 (単位：サイクル)
R_TSIP_GenerateTdesKeyIndex	2,800
R_TSIP_GenerateTdesRandomKeyIndex	2,100
R_TSIP_UpdateTdesKeyIndex	2,400

表 1-40　TDES の性能

API	性能 (単位：サイクル)		
	16 バイト処理	48 バイト処理	80 バイト処理
R_TSIP_TdesEcbEncryptInit	1,100	1,100	1,100
R_TSIP_TdesEcbEncryptUpdate	560	800	1,100
R_TSIP_TdesEcbEncryptFinal	450	450	450
R_TSIP_TdesEcbDecryptInit	1,100	1,100	1,100
R_TSIP_TdesEcbDecryptUpdate	590	830	1,100
R_TSIP_TdesEcbDecryptFinal	470	470	470
R_TSIP_TdesCbcEncryptInit	1,200	1,200	1,200
R_TSIP_TdesCbcEncryptUpdate	630	870	1,200
R_TSIP_TdesCbcEncryptFinal	480	480	480
R_TSIP_TdesCbcDecryptInit	1,200	1,200	1,200
R_TSIP_TdesCbcDecryptUpdate	650	900	1,200
R_TSIP_TdesCbcDecryptFinal	490	490	490

表 1-41　共通 API(ARC4 Wrapped Key 生成)の性能

API	性能 (単位：サイクル)
R_TSIP_GenerateArc4KeyIndex	4,600
R_TSIP_GenerateArc4RandomKeyIndex	11,000
R_TSIP_UpdateArc4KeyIndex	4,200

表 1-42　ARC4 の性能

API	性能 (単位：サイクル)		
	16 バイト処理	48 バイト処理	80 バイト処理
R_TSIP_Arc4EncryptInit	2,100	2,100	2,100
R_TSIP_Arc4EncryptUpdate	500	630	810
R_TSIP_Arc4EncryptFinal	330	330	330
R_TSIP_Arc4DecryptInit	2,100	2,100	2,100
R_TSIP_Arc4DecryptUpdate	490	630	810
R_TSIP_Arc4DecryptFinal	320	330	330

表 1-43　共通 API(RSA Wrapped Key 生成)の性能

API	性能 (単位：サイクル)
R_TSIP_GenerateRsa1024PublicKeyIndex	38,000
R_TSIP_GenerateRsa1024PrivateKeyIndex	39,000
R_TSIP_GenerateRsa2048PublicKeyIndex	140,000
R_TSIP_GenerateRsa2048PrivateKeyIndex	140,000
R_TSIP_GenerateRsa1024RandomKeyIndex (注)	75,000,000
R_TSIP_GenerateRsa2048RandomKeyIndex (注)	540,000,000
R_TSIP_UpdateRsa1024PublicKeyIndex	38,000
R_TSIP_UpdateRsa1024PrivateKeyIndex	39,000
R_TSIP_UpdateRsa2048PublicKeyIndex	140,000
R_TSIP_UpdateRsa2048PrivateKeyIndex	140,000

【注】 10 回実行時の平均値です。

表 1-44　RSASSA-PKCS1-v1_5 署名生成/検証の性能(HASH=SHA1)

API	性能 (単位：サイクル)		
	Message size=1byte	Message size=128byte	Message size=256byte
R_TSIP_RsassaPkcs1024SignatureGenerate	1,300,000	1,300,000	1,300,000
R_TSIP_RsassaPkcs1024SignatureVerification	18,000	20,000	20,000
R_TSIP_RsassaPkcs2048SignatureGenerate	27,000,000	27,000,000	27,000,000
R_TSIP_RsassaPkcs2048SignatureVerification	140,000	140,000	140,000

表 1-45　RSASSA-PKCS1-v1_5 署名生成/検証の性能(HASH=SHA256)

API	性能 (単位：サイクル)		
	Message size=1byte	Message size=128byte	Message size=256byte
R_TSIP_RsassaPkcs1024SignatureGenerate	1,300,000	1,300,000	1,300,000
R_TSIP_RsassaPkcs1024SignatureVerification	18,000	19,000	20,000
R_TSIP_RsassaPkcs2048SignatureGenerate	27,000,000	27,000,000	27,000,000
R_TSIP_RsassaPkcs2048SignatureVerification	140,000	140,000	140,000

表 1-46　RSASSA-PKCS1-v1_5 署名生成/検証の性能(HASH=MD5)

API	性能 (単位：サイクル)		
	Message size=1byte	Message size=128byte	Message size=256byte
R_TSIP_RsassaPkcs1024SignatureGenerate	1,300,000	1,300,000	1,300,000
R_TSIP_RsassaPkcs1024SignatureVerification	18,000	19,000	19,000
R_TSIP_RsassaPkcs2048SignatureGenerate	27,000,000	27,000,000	27,000,000
R_TSIP_RsassaPkcs2048SignatureVerification	140,000	140,000	140,000

表 1-47　RSAES-PKCS1-v1_5 暗号化/復号の性能 鍵サイズ 1024bit

API	性能 (単位：サイクル)	
	Message size=1byte	Message size=117byte
R_TSIP_RsaesPkcs1024Encrypt	23,000	17,000
R_TSIP_RsaesPkcs1024Decrypt	1,300,000	1,300,000

表 1-48　RSAES-PKCS1-v1_5 暗号化/復号の性能 鍵サイズ 2048bit

API	性能 (単位：サイクル)	
	Message size=1byte	Message size=245byte
R_TSIP_RsaesPkcs2048Encrypt	150,000	140,000
R_TSIP_RsaesPkcs2048Decrypt	27,000,000	27,000,000

表 1-49　HASH(SHA1)の性能

API	性能 (単位：サイクル)		
	128 バイト処理	192 バイト処理	256 バイト処理
R_TSIP_Sha1Init	130	130	130
R_TSIP_Sha1Update	1,600	1,800	2,000
R_TSIP_Sha1Final	830	830	830

表 1-50　HASH(SHA256)の性能

API	性能 (単位：サイクル)		
	128 バイト処理	192 バイト処理	256 バイト処理
R_TSIP_Sha256Init	140]	140	140
R_TSIP_Sha256Update	1,600	1,800	2,000
R_TSIP_Sha256Final	840	840	840

表 1-51　HASH(MD5)の性能

API	性能 (単位：サイクル)		
	128 バイト処理	192 バイト処理	256 バイト処理
R_TSIP_Md5Init	120	120	120
R_TSIP_Md5Update	1,500	1,700	1,900
R_TSIP_Md5Final	790	790	790

表 1-52　共通 API(HMAC Wrapped Key 生成)の性能

API	性能 (単位：サイクル)
R_TSIP_GenerateSha1HmacKeyIndex	3,000
R_TSIP_GenerateSha256HmacKeyIndex	3,000
R_TSIP_UpdateSha1HmacKeyIndex	2,700
R_TSIP_UpdateSha256HmacKeyIndex	2,700

表 1-53　HMAC(SHA1)の性能

API	性能 (単位：サイクル)		
	128 バイト処理	192 バイト処理	256 バイト処理
R_TSIP_Sha1HmacGenerateInit	1,400	1,400	1,400
R_TSIP_Sha1HmacGenerateUpdate	980	1,300	1,500
R_TSIP_Sha1HmacGenerateFinal	2,000	2,000	2,000
R_TSIP_Sha1HmacVerifyInit	1,400	1,400	1,400
R_TSIP_Sha1HmacVerifyUpdate	980	1,300	1,500
R_TSIP_Sha1HmacVerifyFinal	3,700	3,700	3,700

表 1-54　HMAC(SHA256)の性能

API	性能 (単位：サイクル)		
	128 バイト処理	192 バイト処理	256 バイト処理
R_TSIP_Sha256HmacGenerateInit	1,900	1,900	1,900
R_TSIP_Sha256HmacGenerateUpdate	920	1,200	1,400
R_TSIP_Sha256HmacGenerateFinal	2,000	2,000	2,000
R_TSIP_Sha256HmacVerifyInit	1,900	1,900	1,900
R_TSIP_Sha256HmacVerifyUpdate	920	1,200	1,400
R_TSIP_Sha256HmacVerifyFinal	3,700	3,700	3,700

表 1-55　共通 API(ECC Wrapped Key 生成)の性能

API	性能 (単位：サイクル)
R_TSIP_GenerateEccP192PublicKeyIndex	3,300
R_TSIP_GenerateEccP224PublicKeyIndex	3,300
R_TSIP_GenerateEccP256PublicKeyIndex	3,300
R_TSIP_GenerateEccP384PublicKeyIndex	3,400
R_TSIP_GenerateEccP192PrivateKeyIndex	3,000
R_TSIP_GenerateEccP224PrivateKeyIndex	3,000
R_TSIP_GenerateEccP256PrivateKeyIndex	3,000
R_TSIP_GenerateEccP384PrivateKeyIndex	2,900
R_TSIP_GenerateEccP192RandomKeyIndex (注)	150,000
R_TSIP_GenerateEccP224RandomKeyIndex (注)	160,000
R_TSIP_GenerateEccP256RandomKeyIndex (注)	160,000
R_TSIP_GenerateEccP384RandomKeyIndex (注)	1,100,000
R_TSIP_UpdateEccP192PublicKeyIndex	3,000
R_TSIP_UpdateEccP224PublicKeyIndex	3,000
R_TSIP_UpdateEccP256PublicKeyIndex	3,000
R_TSIP_UpdateEccP384PublicKeyIndex	3,100
R_TSIP_UpdateEccP192PrivateKeyIndex	2,700
R_TSIP_UpdateEccP224PrivateKeyIndex	2,700
R_TSIP_UpdateEccP256PrivateKeyIndex	2,700
R_TSIP_UpdateEccP384PrivateKeyIndex	2,600

【注】　10 回実行時の平均値です。

表 1-56　ECDSA 署名生成/検証の性能

API	性能 (単位：サイクル)		
	Message size=1byte	Message size=128byte	Message size=256byte
R_TSIP_EcdsaP192SignatureGenerate	180,000	180,000	180,000
R_TSIP_EcdsaP224SignatureGenerate	180,000	190,000	180,000
R_TSIP_EcdsaP256SignatureGenerate	190,000	190,000	190,000
R_TSIP_EcdsaP384SignatureGenerate(注)	1,200,000		
R_TSIP_EcdsaP192SignatureVerification	340,000	340,000	340,000
R_TSIP_EcdsaP224SignatureVerification	360,000	360,000	360,000
R_TSIP_EcdsaP256SignatureVerification	360,000	360,000	360,000
R_TSIP_EcdsaP384SignatureVerification(注)	2,300,000		

【注】SHA384 計算は含まれません

表 1-57　鍵共有の性能

API	性能 (単位：サイクル)
R_TSIP_EcdhP256Init	60
R_TSIP_EcdhP256ReadPublicKey	360,000
R_TSIP_EcdhP256MakePublicKey	340,000
R_TSIP_EcdhP256CalculateSharedSecretIndex	380,000
R_TSIP_EcdhP256KeyDerivation	3,800
R_TSIP_EcdheP512KeyAgreement	3,400,000
R_TSIP_Rsa2048DhKeyAgreement	53,000,000

(KeyAgreement を除いた)鍵共有の性能は、鍵交換形式を ECDHE、派生させる鍵の種類を AES-128 に固定して計測しました。

1.7.6　RX671

表 1-58　共通 API の性能

API	性能 (単位：サイクル)
R_TSIP_Open	5,400,000
R_TSIP_Close	310
R_TSIP_GetVersion	24
R_TSIP_GenerateAes128KeyIndex	2,100
R_TSIP_GenerateAes256KeyIndex	2,200
R_TSIP_GenerateAes128RandomKeyIndex	1,200
R_TSIP_GenerateAes256RandomKeyIndex	1,700
R_TSIP_GenerateRandomNumber	540
R_TSIP_GenerateUpdateKeyRingKeyIndex	2,200
R_TSIP_UpdateAes128KeyIndex	1,800
R_TSIP_UpdateAes256KeyIndex	2,000

表 1-59　Firmware 検証の性能

API	性能 (単位：サイクル)		
	8K バイト処理	16K バイト処理	24K バイト処理
R_TSIP_VerifyFirmwareMAC	17,000	34,000	50,000

表 1-60　AES の性能

API	性能 (単位：サイクル)		
	16 バイト処理	48 バイト処理	80 バイト処理
R_TSIP_Aes128EcbEncryptInit	1,300	1,200	1,200
R_TSIP_Aes128EcbEncryptUpdate	390	500	630
R_TSIP_Aes128EcbEncryptFinal	330	320	320
R_TSIP_Aes128EcbDecryptInit	1,300	1,300	1,300
R_TSIP_Aes128EcbDecryptUpdate	460	560	700
R_TSIP_Aes128EcbDecryptFinal	340	340	340
R_TSIP_Aes256EcbEncryptInit	1,400	1,400	1,400
R_TSIP_Aes256EcbEncryptUpdate	410	530	660
R_TSIP_Aes256EcbEncryptFinal	330	320	320
R_TSIP_Aes256EcbDecryptInit	1,400	1,400	1,400
R_TSIP_Aes256EcbDecryptUpdate	470	580	720
R_TSIP_Aes256EcbDecryptFinal	340	330	330
R_TSIP_Aes128CbcEncryptInit	1,300	1,300	1,300
R_TSIP_Aes128CbcEncryptUpdate	430	540	670
R_TSIP_Aes128CbcEncryptFinal	340	340	340
R_TSIP_Aes128CbcDecryptInit	1,300	1,300	1,300
R_TSIP_Aes128CbcDecryptUpdate	490	600	730
R_TSIP_Aes128CbcDecryptFinal	350	350	350
R_TSIP_Aes256CbcEncryptInit	1,400	1,400	1,400
R_TSIP_Aes256CbcEncryptUpdate	460	580	710
R_TSIP_Aes256CbcEncryptFinal	340	340	340
R_TSIP_Aes256CbcDecryptInit	1,400	1,400	1,400
R_TSIP_Aes256CbcDecryptUpdate	530	640	770
R_TSIP_Aes256CbcDecryptFinal	350	350	350

表 1-61　AES-GCM の性能

API	性能 (単位：サイクル)		
	48 バイト処理	64 バイト処理	80 バイト処理
R_TSIP_Aes128GcmEncryptInit	4,200	4,100	4,100
R_TSIP_Aes128GcmEncryptUpdate	1,600	1,700	1,700
R_TSIP_Aes128GcmEncryptFinal	950	950	950
R_TSIP_Aes128GcmDecryptInit	4,100	4,100	4,100
R_TSIP_Aes128GcmDecryptUpdate	1,600	1,600	1,700
R_TSIP_Aes128GcmDecryptFinal	1,500	1,500	1,500
R_TSIP_Aes256GcmEncryptInit	4,200	4,100	4,100
R_TSIP_Aes256GcmEncryptUpdate	1,600	1,700	1,800
R_TSIP_Aes256GcmEncryptFinal	830	820	820
R_TSIP_Aes256GcmDecryptInit	4,200	4,100	4,100
R_TSIP_Aes256GcmDecryptUpdate	1,600	1,700	1,700
R_TSIP_Aes256GcmDecryptFinal	1,500	1,500	1,500

GCM の性能は、ivec を 1024bit、追加認証データを 720bit、認証タグを 128bit に固定して計測しました。

表 1-62　AES-CCM の性能

API	性能 (単位：サイクル)		
	48 バイト処理	64 バイト処理	80 バイト処理
R_TSIP_Aes128CcmEncryptInit	2,300	2,300	2,300
R_TSIP_Aes128CcmEncryptUpdate	890	960	1,100
R_TSIP_Aes128CcmEncryptFinal	760	750	750
R_TSIP_Aes128CcmDecryptInit	2,400	2,400	2,400
R_TSIP_Aes128CcmDecryptUpdate	810	880	960
R_TSIP_Aes128CcmDecryptFinal	1,500	1,500	1,500
R_TSIP_Aes256CcmEncryptInit	1,900	1,900	1,900
R_TSIP_Aes256CcmEncryptUpdate	940	1,100	1,200
R_TSIP_Aes256CcmEncryptFinal	770	770	770
R_TSIP_Aes256CcmDecryptInit	1,900	1,900	1,900
R_TSIP_Aes256CcmDecryptUpdate	850	930	1,100
R_TSIP_Aes256CcmDecryptFinal	1,500	1,500	1,500

CCM の性能は、ノンスを 104bit、追加認証データを 880bit、MAC を 128bit に固定して計測しました。

表 1-63　AES-CMAC の性能

API	性能 (単位：サイクル)		
	48 バイト処理	64 バイト処理	80 バイト処理
R_TSIP_Aes128CmacGenerateInit	880	870	870
R_TSIP_Aes128CmacGenerateUpdate	490	520	560
R_TSIP_Aes128CmacGenerateFinal	630	620	620
R_TSIP_Aes128CmacVerifyInit	870	870	870
R_TSIP_Aes128CmacVerifyUpdate	490	530	570
R_TSIP_Aes128CmacVerifyFinal	1,300	1,300	1,300
R_TSIP_Aes256CmacGenerateInit	1,000	1,000	1,000
R_TSIP_Aes256CmacGenerateUpdate	520	550	600
R_TSIP_Aes256CmacGenerateFinal	650	630	630
R_TSIP_Aes256CmacVerifyInit	990	990	990
R_TSIP_Aes256CmacVerifyUpdate	510	550	600
R_TSIP_Aes256CmacVerifyFinal	1,300	1,300	1,300

表 1-64　AES Key Wrap の性能

API	性能 (単位：サイクル)	
	ラップ対象鍵 AES-128	ラップ対象鍵 AES-256
R_TSIP_Aes128KeyWrap	6,400	10,000
R_TSIP_Aes256KeyWrap	6,600	11,000
R_TSIP_Aes128KeyUnwrap	7,200	11,000
R_TSIP_Aes256KeyUnwrap	7,400	12,000

表 1-65　共通 API(TDES Wrapped Key 生成)の性能

API	性能 (単位：サイクル)
R_TSIP_GenerateTdesKeyIndex	2,200
R_TSIP_GenerateTdesRandomKeyIndex	1,700
R_TSIP_UpdateTdesKeyIndex	2,000

表 1-66　TDES の性能

API	性能 (単位：サイクル)		
	16 バイト処理	48 バイト処理	80 バイト処理
R_TSIP_TdesEcbEncryptInit	800	790	790
R_TSIP_TdesEcbEncryptUpdate	430	620	820
R_TSIP_TdesEcbEncryptFinal	330	320	320
R_TSIP_TdesEcbDecryptInit	810	810	810
R_TSIP_TdesEcbDecryptUpdate	450	640	840
R_TSIP_TdesEcbDecryptFinal	330	330	330
R_TSIP_TdesCbcEncryptInit	850	840	840
R_TSIP_TdesCbcEncryptUpdate	490	680	880
R_TSIP_TdesCbcEncryptFinal	340	340	340
R_TSIP_TdesCbcDecryptInit	850	850	850
R_TSIP_TdesCbcDecryptUpdate	500	700	890
R_TSIP_TdesCbcDecryptFinal	350	350	350

表 1-67　共通 API(ARC4 Wrapped Key 生成)の性能

API	性能 (単位：サイクル)
R_TSIP_GenerateArc4KeyIndex	3,900
R_TSIP_GenerateArc4RandomKeyIndex	8,600
R_TSIP_UpdateArc4KeyIndex	3,700

表 1-68　ARC4 の性能

API	性能 (単位：サイクル)		
	16 バイト処理	48 バイト処理	80 バイト処理
R_TSIP_Arc4EncryptInit	1,800	1,800	1,800
R_TSIP_Arc4EncryptUpdate	360	480	610
R_TSIP_Arc4EncryptFinal	240	230	230
R_TSIP_Arc4DecryptInit	1,800	1,800	1,800
R_TSIP_Arc4DecryptUpdate	360	480	610
R_TSIP_Arc4DecryptFinal	230	230	230

表 1-69　共通 API(RSA Wrapped Key 生成)の性能

API	性能 (単位：サイクル)
R_TSIP_GenerateRsa1024PublicKeyIndex	37,000
R_TSIP_GenerateRsa1024PrivateKeyIndex	38,000
R_TSIP_GenerateRsa2048PublicKeyIndex	140,000
R_TSIP_GenerateRsa2048PrivateKeyIndex	140,000
R_TSIP_GenerateRsa1024RandomKeyIndex (注)	67,000,000
R_TSIP_GenerateRsa2048RandomKeyIndex (注)	460,000,000
R_TSIP_UpdateRsa1024PublicKeyIndex	37,000
R_TSIP_UpdateRsa1024PrivateKeyIndex	38,000
R_TSIP_UpdateRsa2048PublicKeyIndex	140,000
R_TSIP_UpdateRsa2048PrivateKeyIndex	140,000

【注】 10 回実行時の平均値です。

表 1-70　RSASSA-PKCS1-v1_5 署名生成/検証の性能(HASH=SHA1)

API	性能 (単位：サイクル)		
	Message size=1byte	Message size=128byte	Message size=256byte
R_TSIP_RsassaPkcs1024SignatureGenerate	1,300,000	1,300,000	1,300,000
R_TSIP_RsassaPkcs1024SignatureVerification	17,000	18,000	18,000
R_TSIP_RsassaPkcs2048SignatureGenerate	27,000,000	27,000,000	27,000,000
R_TSIP_RsassaPkcs2048SignatureVerification	140,000	140,000	140,000

表 1-71　RSASSA-PKCS1-v1_5 署名生成/検証の性能(HASH=SHA256)

API	性能 (単位：サイクル)		
	Message size=1byte	Message size=128byte	Message size=256byte
R_TSIP_RsassaPkcs1024SignatureGenerate	1,300,000	1,300,000	1,300,000
R_TSIP_RsassaPkcs1024SignatureVerification	17,000	18,000	18,000
R_TSIP_RsassaPkcs2048SignatureGenerate	27,000,000	27,000,000	27,000,000
R_TSIP_RsassaPkcs2048SignatureVerification	140,000	140,000	140,000

表 1-72　RSASSA-PKCS1-v1_5 署名生成/検証の性能(HASH=MD5)

API	性能 (単位：サイクル)		
	Message size=1byte	Message size=128byte	Message size=256byte
R_TSIP_RsassaPkcs1024SignatureGenerate	1,300,000	1,300,000	1,300,000
R_TSIP_RsassaPkcs1024SignatureVerification	17,000	18,000	18,000
R_TSIP_RsassaPkcs2048SignatureGenerate	27,000,000	27,000,000	27,000,000
R_TSIP_RsassaPkcs2048SignatureVerification	140,000	140,000	140,000

表 1-73　RSAES-PKCS1-v1_5 暗号化/復号の性能 鍵サイズ 1024bit

API	性能 (単位：サイクル)	
	Message size=1byte	Message size=117byte
R_TSIP_RsaesPkcs1024Encrypt	20,000	16,000
R_TSIP_RsaesPkcs1024Decrypt	1,300,000	1,300,000

表 1-74　RSAES-PKCS1-v1_5 暗号化/復号の性能 鍵サイズ 2048bit

API	性能 (単位：サイクル)	
	Message size=1byte	Message size=245byte
R_TSIP_RsaesPkcs2048Encrypt	150,000	140,000
R_TSIP_RsaesPkcs2048Decrypt	27,000,000	27,000,000

表 1-75　HASH(SHA1)の性能

API	性能 (単位：サイクル)		
	128 バイト処理	192 バイト処理	256 バイト処理
R_TSIP_Sha1Init	110	110	110
R_TSIP_Sha1Update	1,300	1,500	1,700
R_TSIP_Sha1Final	660	660	660

表 1-76　HASH(SHA256)の性能

API	性能 (単位：サイクル)		
	128 バイト処理	192 バイト処理	256 バイト処理
R_TSIP_Sha256Init	120	120	120
R_TSIP_Sha256Update	1,300	1,500	1,600
R_TSIP_Sha256Final	670	670	670

表 1-77　HASH(MD5)の性能

API	性能 (単位：サイクル)		
	128 バイト処理	192 バイト処理	256 バイト処理
R_TSIP_Md5Init	96	96	96
R_TSIP_Md5Update	1,200	1,300	1,500
R_TSIP_Md5Final	630	630	630

表 1-78　共通 API(HMAC Wrapped Key 生成)の性能

API	性能 (単位：サイクル)
R_TSIP_GenerateSha1HmacKeyIndex	2,300
R_TSIP_GenerateSha256HmacKeyIndex	2,300
R_TSIP_UpdateSha1HmacKeyIndex	2,100
R_TSIP_UpdateSha256HmacKeyIndex	2,000

表 1-79　HMAC(SHA1)の性能

API	性能 (単位：サイクル)		
	128 バイト処理	192 バイト処理	256 バイト処理
R_TSIP_Sha1HmacGenerateInit	1,100	1,100	1,100
R_TSIP_Sha1HmacGenerateUpdate	810	1,100	1,300
R_TSIP_Sha1HmacGenerateFinal	1,600	1,600	1,600
R_TSIP_Sha1HmacVerifyInit	1,100	1,100	1,100
R_TSIP_Sha1HmacVerifyUpdate	800	1,100	1,300
R_TSIP_Sha1HmacVerifyFinal	2,800	2,800	2,800

表 1-80　HMAC(SHA256)の性能

API	性能 (単位：サイクル)		
	128 バイト処理	192 バイト処理	256 バイト処理
R_TSIP_Sha256HmacGenerateInit	1,400	1,400	1,400
R_TSIP_Sha256HmacGenerateUpdate	750	910	1,100
R_TSIP_Sha256HmacGenerateFinal	1,600	1,600	1,600
R_TSIP_Sha256HmacVerifyInit	1,400	1,300	1,300
R_TSIP_Sha256HmacVerifyUpdate	730	910	1,100
R_TSIP_Sha256HmacVerifyFinal	2,700	2,700	2,700

表 1-81　共通 API(ECC Wrapped Key 生成)の性能

API	性能 (単位：サイクル)
R_TSIP_GenerateEccP192PublicKeyIndex	2,600
R_TSIP_GenerateEccP224PublicKeyIndex	2,600
R_TSIP_GenerateEccP256PublicKeyIndex	2,600
R_TSIP_GenerateEccP384PublicKeyIndex	2,800
R_TSIP_GenerateEccP192PrivateKeyIndex	2,300
R_TSIP_GenerateEccP224PrivateKeyIndex	2,300
R_TSIP_GenerateEccP256PrivateKeyIndex	2,300
R_TSIP_GenerateEccP384PrivateKeyIndex	2,300
R_TSIP_GenerateEccP192RandomKeyIndex (注)	140,000
R_TSIP_GenerateEccP224RandomKeyIndex (注)	150,000
R_TSIP_GenerateEccP256RandomKeyIndex (注)	150,000
R_TSIP_GenerateEccP384RandomKeyIndex (注)	1,100,000
R_TSIP_UpdateEccP192PublicKeyIndex	2,400
R_TSIP_UpdateEccP224PublicKeyIndex	2,300
R_TSIP_UpdateEccP256PublicKeyIndex	2,300
R_TSIP_UpdateEccP384PublicKeyIndex	2,500
R_TSIP_UpdateEccP192PrivateKeyIndex	2,100
R_TSIP_UpdateEccP224PrivateKeyIndex	2,100
R_TSIP_UpdateEccP256PrivateKeyIndex	2,100
R_TSIP_UpdateEccP384PrivateKeyIndex	2,100

【注】 10 回実行時の平均値です。

表 1-82　ECDSA 署名生成/検証の性能

API	性能 (単位：サイクル)		
	Message size=1byte	Message size=128byte	Message size=256byte
R_TSIP_EcdsaP192SignatureGenerate	170,000	170,000	170,000
R_TSIP_EcdsaP224SignatureGenerate	170,000	170,000	170,000
R_TSIP_EcdsaP256SignatureGenerate	170,000	180,000	170,000
R_TSIP_EcdsaP384SignatureGenerate(注)	1,200,000		
R_TSIP_EcdsaP192SignatureVerification	310,000	320,000	310,000
R_TSIP_EcdsaP224SignatureVerification	330,000	330,000	330,000
R_TSIP_EcdsaP256SignatureVerification	330,000	340,000	330,000
R_TSIP_EcdsaP384SignatureVerification(注)	2,200,000		

【注】SHA384 計算は含まれません

表 1-83　鍵共有の性能

API	性能 (単位：サイクル)
R_TSIP_EcdhP256Init	44
R_TSIP_EcdhP256ReadPublicKey	340,000
R_TSIP_EcdhP256MakePublicKey	320,000
R_TSIP_EcdhP256CalculateSharedSecretIndex	360,000
R_TSIP_EcdhP256KeyDerivation	3,000
R_TSIP_EcdheP512KeyAgreement	3,300,000
R_TSIP_Rsa2048DhKeyAgreement	53,000,000

(KeyAgreement を除いた)鍵共有の性能は、鍵交換形式を ECDHE、派生させる鍵の種類を AES-128 に固定して計測しました。

1.7.7　RX72M, RX72N

表 1-84　共通 API の性能

API	性能 (単位：サイクル)
R_TSIP_Open	6,300,000
R_TSIP_Close	310
R_TSIP_GetVersion	22
R_TSIP_GenerateAes128KeyIndex	2,200
R_TSIP_GenerateAes256KeyIndex	2,300
R_TSIP_GenerateAes128RandomKeyIndex	1,300
R_TSIP_GenerateAes256RandomKeyIndex	1,800
R_TSIP_GenerateRandomNumber	570
R_TSIP_GenerateUpdateKeyRingKeyIndex	2,300
R_TSIP_UpdateAes128KeyIndex	1,900
R_TSIP_UpdateAes256KeyIndex	2,100

表 1-85　Firmware 検証の性能

API	性能 (単位：サイクル)		
	8K バイト処理	16K バイト処理	24K バイト処理
R_TSIP_VerifyFirmwareMAC	19,000	38,000	56,000

表 1-86　AES の性能

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

表 1-87　AES-GCM の性能

API	性能 (単位：サイクル)		
	48 バイト処理	64 バイト処理	80 バイト処理
R_TSIP_Aes128GcmEncryptInit	4,400	4,400	4,400
R_TSIP_Aes128GcmEncryptUpdate	1,600	1,700	1,800
R_TSIP_Aes128GcmEncryptFinal	1,100	1,100	1,100
R_TSIP_Aes128GcmDecryptInit	4,300	4,300	4,300
R_TSIP_Aes128GcmDecryptUpdate	1,600	1,700	1,800
R_TSIP_Aes128GcmDecryptFinal	1,700	1,700	1,700
R_TSIP_Aes256GcmEncryptInit	4,400	4,400	4,400
R_TSIP_Aes256GcmEncryptUpdate	1,700	1,800	1,800
R_TSIP_Aes256GcmEncryptFinal	860	860	860
R_TSIP_Aes256GcmDecryptInit	1,700	1,700	1,700
R_TSIP_Aes256GcmDecryptUpdate	1,700	1,700	1,800
R_TSIP_Aes256GcmDecryptFinal	1,500	1,500	1,500

GCM の性能は、ivec を 1024bit、追加認証データを 720bit、認証タグを 128bit に固定して計測しました。

表 1-88　AES-CCM の性能

API	性能 (単位：サイクル)		
	48 バイト処理	64 バイト処理	80 バイト処理
R_TSIP_Aes128CcmEncryptInit	2,400	2,400	2,400
R_TSIP_Aes128CcmEncryptUpdate	910	980	1,100
R_TSIP_Aes128CcmEncryptFinal	760	760	760
R_TSIP_Aes128CcmDecryptInit	2,500	2,500	2,500
R_TSIP_Aes128CcmDecryptUpdate	830	900	980
R_TSIP_Aes128CcmDecryptFinal	1,500	1,500	1,500
R_TSIP_Aes256CcmEncryptInit	2,000	2,000	2,000
R_TSIP_Aes256CcmEncryptUpdate	960	1,100	1,200
R_TSIP_Aes256CcmEncryptFinal	800	800	800
R_TSIP_Aes256CcmDecryptInit	2,000	2,000	2,000
R_TSIP_Aes256CcmDecryptUpdate	860	960	1,100
R_TSIP_Aes256CcmDecryptFinal	1,600	1,600	1,600

CCM の性能は、ノンスを 104bit、追加認証データを 880bit、MAC を 128bit に固定して計測しました。

表 1-89　AES-CMAC の性能

API	性能 (単位：サイクル)		
	48 バイト処理	64 バイト処理	80 バイト処理
R_TSIP_Aes128CmacGenerateInit	920	910	920
R_TSIP_Aes128CmacGenerateUpdate	490	530	570
R_TSIP_Aes128CmacGenerateFinal	630	630	630
R_TSIP_Aes128CmacVerifyInit	910	920	920
R_TSIP_Aes128CmacVerifyUpdate	490	530	570
R_TSIP_Aes128CmacVerifyFinal	1,300	1,300	1,300
R_TSIP_Aes256CmacGenerateInit	1,100	1,100	1,100
R_TSIP_Aes256CmacGenerateUpdate	520	560	610
R_TSIP_Aes256CmacGenerateFinal	660	660	660
R_TSIP_Aes256CmacVerifyInit	1,100	1,100	1,100
R_TSIP_Aes256CmacVerifyUpdate	530	570	610
R_TSIP_Aes256CmacVerifyFinal	1,300	1,300	1,300

表 1-90　AES Key Wrap の性能

API	性能 (単位：サイクル)	
	ラップ対象鍵 AES-128	ラップ対象鍵 AES-256
R_TSIP_Aes128KeyWrap	6,500	11,000
R_TSIP_Aes256KeyWrap	6,800	11,000
R_TSIP_Aes128KeyUnwrap	7,400	12,000
R_TSIP_Aes256KeyUnwrap	7,600	12,000

表 1-91　共通 API(TDES Wrapped Key 生成)の性能

API	性能 (単位：サイクル)
R_TSIP_GenerateTdesKeyIndex	2,300
R_TSIP_GenerateTdesRandomKeyIndex	1,800
R_TSIP_UpdateTdesKeyIndex	2,100

表 1-92　TDES の性能

API	性能 (単位：サイクル)		
	16 バイト処理	48 バイト処理	80 バイト処理
R_TSIP_TdesEcbEncryptInit	830	830	830
R_TSIP_TdesEcbEncryptUpdate	440	640	840
R_TSIP_TdesEcbEncryptFinal	330	330	330
R_TSIP_TdesEcbDecryptInit	850	850	850
R_TSIP_TdesEcbDecryptUpdate	460	660	860
R_TSIP_TdesEcbDecryptFinal	340	340	350
R_TSIP_TdesCbcEncryptInit	880	890	890
R_TSIP_TdesCbcEncryptUpdate	490	690	890
R_TSIP_TdesCbcEncryptFinal	360	360	360
R_TSIP_TdesCbcDecryptInit	890	890	890
R_TSIP_TdesCbcDecryptUpdate	510	720	910
R_TSIP_TdesCbcDecryptFinal	370	370	370

表 1-93　共通 API(ARC4 Wrapped Key 生成)の性能

API	性能 (単位：サイクル)
R_TSIP_GenerateArc4KeyIndex	4,000
R_TSIP_GenerateArc4RandomKeyIndex	9,200
R_TSIP_UpdateArc4KeyIndex	3,800

表 1-94　ARC4 の性能

API	性能 (単位：サイクル)		
	16 バイト処理	48 バイト処理	80 バイト処理
R_TSIP_Arc4EncryptInit	1,900	1,900	1,900
R_TSIP_Arc4EncryptUpdate	370	490	620
R_TSIP_Arc4EncryptFinal	240	240	240
R_TSIP_Arc4DecryptInit	1,900	1,900	1,900
R_TSIP_Arc4DecryptUpdate	370	490	620
R_TSIP_Arc4DecryptFinal	240	240	240

表 1-95　共通 API(RSA Wrapped Key 生成)の性能

API	性能 (単位：サイクル)
R_TSIP_GenerateRsa1024PublicKeyIndex	37,000
R_TSIP_GenerateRsa1024PrivateKeyIndex	38,000
R_TSIP_GenerateRsa2048PublicKeyIndex	140,000
R_TSIP_GenerateRsa2048PrivateKeyIndex	140,000
R_TSIP_GenerateRsa1024RandomKeyIndex (注)	61,000,000
R_TSIP_GenerateRsa2048RandomKeyIndex (注)	450,000,000
R_TSIP_UpdateRsa1024PublicKeyIndex	37,000
R_TSIP_UpdateRsa1024PrivateKeyIndex	38,000
R_TSIP_UpdateRsa2048PublicKeyIndex	140,000
R_TSIP_UpdateRsa2048PrivateKeyIndex	140,000

【注】 10 回実行時の平均値です。

表 1-96　RSASSA-PKCS1-v1_5 署名生成/検証の性能(HASH=SHA1)

API	性能 (単位：サイクル)		
	Message size=1byte	Message size=128byte	Message size=256byte
R_TSIP_RsassaPkcs1024SignatureGenerate	1,300,000	1,300,000	1,300,000
R_TSIP_RsassaPkcs1024SignatureVerification	17,000	18,000	18,000
R_TSIP_RsassaPkcs2048SignatureGenerate	27,000,000	27,000,000	27,000,000
R_TSIP_RsassaPkcs2048SignatureVerification	140,000	140,000	140,000

表 1-97　RSASSA-PKCS1-v1_5 署名生成/検証の性能(HASH=SHA256)

API	性能 (単位：サイクル)		
	Message size=1byte	Message size=128byte	Message size=256byte
R_TSIP_RsassaPkcs1024SignatureGenerate	1,300,000	1,300,000	1,300,000
R_TSIP_RsassaPkcs1024SignatureVerification	17,000	18,000	18,000
R_TSIP_RsassaPkcs2048SignatureGenerate	27,000,000	27,000,000	27,000,000
R_TSIP_RsassaPkcs2048SignatureVerification	140,000	140,000	140,000

表 1-98　RSASSA-PKCS1-v1_5 署名生成/検証の性能(HASH=MD5)

API	性能 (単位：サイクル)		
	Message size=1byte	Message size=128byte	Message size=256byte
R_TSIP_RsassaPkcs1024SignatureGenerate	1,300,000	1,300,000	1,300,000
R_TSIP_RsassaPkcs1024SignatureVerification	17,000	18,000	18,000
R_TSIP_RsassaPkcs2048SignatureGenerate	27,000,000	27,000,000	27,000,000
R_TSIP_RsassaPkcs2048SignatureVerification	140,000	140,000	140,000

表 1-99　RSAES-PKCS1-v1_5 暗号化/復号の性能 鍵サイズ 1024bit

API	性能 (単位：サイクル)	
	Message size=1byte	Message size=117byte
R_TSIP_RsaesPkcs1024Encrypt	21,000	16,000
R_TSIP_RsaesPkcs1024Decrypt	1,300,000	1,300,000

表 1-100　RSAES-PKCS1-v1_5 暗号化/復号の性能 鍵サイズ 2048bit

API	性能 (単位：サイクル)	
	Message size=1byte	Message size=245byte
R_TSIP_RsaesPkcs2048Encrypt	150,000	140,000
R_TSIP_RsaesPkcs2048Decrypt	27,000,000	27,000,000

表 1-101　HASH(SHA1)の性能

API	性能 (単位：サイクル)		
	128 バイト処理	192 バイト処理	256 バイト処理
R_TSIP_Sha1Init	100	110	110
R_TSIP_Sha1Update	1,300	1,500	1,700
R_TSIP_Sha1Final	670	680	670

表 1-102　HASH(SHA256)の性能

API	性能 (単位：サイクル)		
	128 バイト処理	192 バイト処理	256 バイト処理
R_TSIP_Sha256Init	110	110	110
R_TSIP_Sha256Update	1,300	1,500	1,700
R_TSIP_Sha256Final	640	640	640

表 1-103　HASH(MD5)の性能

API	性能 (単位：サイクル)		
	128 バイト処理	192 バイト処理	256 バイト処理
R_TSIP_Md5Init	94	94	94
R_TSIP_Md5Update	1,200	1,400	1,500
R_TSIP_Md5Final	630	630	630

表 1-104　共通 API(HMAC Wrapped Key 生成)の性能

API	性能 (単位：サイクル)
R_TSIP_GenerateSha1HmacKeyIndex	2,400
R_TSIP_GenerateSha256HmacKeyIndex	2,400
R_TSIP_UpdateSha1HmacKeyIndex	2,200
R_TSIP_UpdateSha256HmacKeyIndex	2,200

表 1-105　HMAC(SHA1)の性能

API	性能 (単位：サイクル)		
	128 バイト処理	192 バイト処理	256 バイト処理
R_TSIP_Sha1HmacGenerateInit	1,100	1,100	1,100
R_TSIP_Sha1HmacGenerateUpdate	810	1,100	1,300
R_TSIP_Sha1HmacGenerateFinal	1,700	1,700	1,700
R_TSIP_Sha1HmacVerifyInit	1,100	1,100	1,100
R_TSIP_Sha1HmacVerifyUpdate	810	1,100	1,300
R_TSIP_Sha1HmacVerifyFinal	2,800	2,800	2,800

表 1-106　HMAC(SHA256)の性能

API	性能 (単位：サイクル)		
	128 バイト処理	192 バイト処理	256 バイト処理
R_TSIP_Sha256HmacGenerateInit	1,400	1,400	1,400
R_TSIP_Sha256HmacGenerateUpdate	740	910	1,100
R_TSIP_Sha256HmacGenerateFinal	1,600	1,600	1,600
R_TSIP_Sha256HmacVerifyInit	1,400	1,400	1,400
R_TSIP_Sha256HmacVerifyUpdate	730	910	1,100
R_TSIP_Sha256HmacVerifyFinal	2,800	2,800	2,800

表 1-107　共通 API(ECC Wrapped Key 生成)の性能

API	性能 (単位：サイクル)
R_TSIP_GenerateEccP192PublicKeyIndex	2,700
R_TSIP_GenerateEccP224PublicKeyIndex	2,700
R_TSIP_GenerateEccP256PublicKeyIndex	2,700
R_TSIP_GenerateEccP384PublicKeyIndex	2,900
R_TSIP_GenerateEccP192PrivateKeyIndex	2,400
R_TSIP_GenerateEccP224PrivateKeyIndex	2,400
R_TSIP_GenerateEccP256PrivateKeyIndex	2,400
R_TSIP_GenerateEccP384PrivateKeyIndex	2,400
R_TSIP_GenerateEccP192RandomKeyIndex (注)	140,000
R_TSIP_GenerateEccP224RandomKeyIndex (注)	150,000
R_TSIP_GenerateEccP256RandomKeyIndex (注)	150,000
R_TSIP_GenerateEccP384RandomKeyIndex (注)	1,100,000
R_TSIP_UpdateEccP192PublicKeyIndex	2,500
R_TSIP_UpdateEccP224PublicKeyIndex	2,500
R_TSIP_UpdateEccP256PublicKeyIndex	2,500
R_TSIP_UpdateEccP384PublicKeyIndex	2,600
R_TSIP_UpdateEccP192PrivateKeyIndex	2,200
R_TSIP_UpdateEccP224PrivateKeyIndex	2,200
R_TSIP_UpdateEccP256PrivateKeyIndex	2,200
R_TSIP_UpdateEccP384PrivateKeyIndex	2,200

【注】 10 回実行時の平均値です。

表 1-108　ECDSA 署名生成/検証の性能

API	性能 (単位：サイクル)		
	Message size=1byte	Message size=128byte	Message size=256byte
R_TSIP_EcdsaP192SignatureGenerate	170,000	170,000	170,000
R_TSIP_EcdsaP224SignatureGenerate	170,000	170,000	170,000
R_TSIP_EcdsaP256SignatureGenerate	180,000	170,000	170,000
R_TSIP_EcdsaP384SignatureGenerate(注)	1,200,000		
R_TSIP_EcdsaP192SignatureVerification	310,000	310,000	310,000
R_TSIP_EcdsaP224SignatureVerification	330,000	330,000	340,000
R_TSIP_EcdsaP256SignatureVerification	340,000	340,000	340,000
R_TSIP_EcdsaP384SignatureVerification(注)	2,100,000		

【注】SHA384 計算は含まれません

表 1-109　鍵共有の性能

API	性能 (単位：サイクル)
R_TSIP_EcdhP256Init	42
R_TSIP_EcdhP256ReadPublicKey	340,000
R_TSIP_EcdhP256MakePublicKey	320,000
R_TSIP_EcdhP256CalculateSharedSecretIndex	360,000
R_TSIP_EcdhP256KeyDerivation	3,200
R_TSIP_EcdheP512KeyAgreement	3,300,000
R_TSIP_Rsa2048DhKeyAgreement	53,000,000

(KeyAgreement を除いた)鍵共有の性能は、鍵交換形式を ECDHE、派生させる鍵の種類を AES-128 に固定して計測しました。

2. API 情報

2.1 ハードウェアの要求

TSIP ドライバは、TSIP 搭載デバイスでのみ使用可能です。TSIP を搭載している型名のデバイスをご使用ください。

2.2 ソフトウェアの要求

TSIP ドライバは、以下モジュールに依存します。

- r_bsp　V7.30 以降をご使用ください。(BSP=Board Support Package)

■RX231、RX23W を使用する場合(RX231 では、下記コメントの” = Chip”以降が一部異なります。)

r_config フォルダの r_bsp_config.h の以下マクロの値を 0xB、0xD(RX23W のみ)のいずれかに変更してください。

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

■RX26T を使用する場合

r_config フォルダの r_bsp_config.h の以下マクロの値を 0xB、0xD のいずれかに変更してください。

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

■RX66T、RX72T を使用する場合(RX72T では、下記コメントの"= PGA"以降が一部異なります。)

r_config フォルダの r_bsp_config.h の以下マクロの値を 0xE、0xF、0x10 のいずれかに変更してください。

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

■RX66N、RX671、RX72M、RX72N を使用する場合

r_config フォルダの r_bsp_config.h の以下マクロの値を 0x11 に変更してください。

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

■RX65N を使用する場合

r_config フォルダの r_bsp_config.h の以下マクロの値を true に変更してください。

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

2.3　サポートされているツールチェイン

TSIP ドライバは「7.1 動作確認環境」に示すツールチェインで動作を確認しています。

2.4　ヘッダファイル

すべての API 呼び出しとそれをサポートするインタフェース定義は r_tsip_rx_if.h に記載しています。

2.5　整数型

TSIP ドライバは ANSI C99 の stdint.h で定義されている整数型を使用しています。

2.6　コンフィグレーション

/r_config/r_tsip_rx_config.h で以下マクロの 1/0 を切り替えることで、各機能の ON/OFF を切り替えることができます。

Configuration options in r_config/r_tsip_rx_config.h		
定義	デフォルト値	備考
TSIP_AES_128_ECB_ENCRYPT	(1)	-
TSIP_AES_256_ECB_ENCRYPT	(1)	-
TSIP_AES_128_ECB_DECRYPT	(1)	-
TSIP_AES_256_ECB_DECRYPT	(1)	-
TSIP_AES_128_CBC_ENCRYPT	(1)	-
TSIP_AES_256_CBC_ENCRYPT	(1)	-
TSIP_AES_128_CBC_DECRYPT	(1)	-
TSIP_AES_256_CBC_DECRYPT	(1)	-
TSIP_AES_128_CTR	(1)	-
TSIP_AES_256_CTR	(1)	-
TSIP_AES_128_GCM_ENCRYPT	(1)	-
TSIP_AES_256_GCM_ENCRYPT	(1)	-
TSIP_AES_128_GCM_DECRYPT	(1)	-
TSIP_AES_256_GCM_DECRYPT	(1)	-
TSIP_AES_128_CMAC	(1)	-
TSIP_AES_256_CMAC	(1)	-
TSIP_AES_128_CCM_ENCRYPT	(1)	-
TSIP_AES_256_CCM_ENCRYPT	(1)	-
TSIP_AES_128_CCM_DECRYPT	(1)	-
TSIP_AES_256_CCM_DECRYPT	(1)	-
TSIP_AES_128_KEY_WRAP	(1)	
TSIP_AES_256_KEY_WRAP	(1)	
TSIP_TDES_ECB_ENCRYPT	(1)	-
TSIP_TDES_ECB_DECRYPT	(1)	-
TSIP_TDES_CBC_ENCRYPT	(1)	-
TSIP_TDES_CBC_DECRYPT	(1)	-
TSIP_ARC4_ENCRYPT	(1)	-
TSIP_ARC4_DECRYPT	(1)	-
TSIP_SHA_1	(1)	-
TSIP_SHA_256	(1)	-
TSIP_MD5	(1)	-
TSIP_SHA_1_HMAC	(1)	-
TSIP_SHA_256_HMAC	(1)	-
TSIP_RSAES_1024	(1)	-
TSIP_RSAES_2048	(1)	-
TSIP_RSAES_3072	(1)	-
TSIP_RSAES_4096	(1)	-
TSIP_RSASSA_1024	(1)	-
TSIP_RSASSA_2048	(1)	-
TSIP_RSASSA_3072	(1)	-

Configuration options in r_config/r_tsip_rx_config.h		
定義	デフォルト値	備考
TSIP_RSASSA_4096	(1)	-
TSIP_RSA_RETRY_COUNT_FOR_RSA_KEY_GENERATION	(5120*2)	RSA 鍵生成のリトライ回数。NIST FIPS186-4 では (5*(鍵長/2)が推奨値。デフォルト値は鍵長=2048 の時のものにマージン分として 2 倍している。
TSIP_ECDSA_P192	(1)	-
TSIP_ECDSA_P224	(1)	-
TSIP_ECDSA_P256	(1)	-
TSIP_ECDSA_P384	(1)	-
TSIP_ECDH_P256	(1)	-
TSIP_USER_SHA_384_ENABLED 【注】	(0)	-
TSIP_TLS	(1)	-
TSIP_SECURE_BOOT	(0)	-
TSIP_FIRMWARE_UPDATE	(1)	-
TSIP_MULTI_THREADING 【注】	(0)	-

注：ユーザ定義関数の定義方法は、4.3 ユーザ定義関数を参照してください。

2.7　構造体

TSIP ドライバで使用している構造体の定義は r_tsip_rx_if.h を参照してください。

2.8　戻り値

以下に TSIP ドライバの API 関数で使用している戻り値を示します。戻り値の列挙型は r_tsip_rx_if.h で定義されています。

```
typedef enum e_tsip_err
{
   TSIP_SUCCESS=0,
   TSIP_ERR_FAIL,                // 自己診断が異常終了
                                 // R_TSIP_VerifyFirmwareMAC による MAC 異常検出
                                 // または R_TSIP_各 API の内部エラー
   TSIP_ERR_RESOURCE_CONFLICT,  // 本処理に必要なリソースが他の処理で利用されている
                                 // ことによるリソース衝突が発生
   TSIP_ERR_RETRY,               // 自己診断が異常終了。本関数を再実行してください。
   TSIP_ERR_KEY_SET,             // 異常な Wrapped Key が入力された
   TSIP_ERR_AUTHENTICATION,      // 認証が失敗
                                 // または RSASSA-PKCS1-V.1.5 による署名文検証失敗
   TSIP_ERR_CALLBACK_UNREGIST,  // コールバック関数未登録
   TSIP_ERR_PARAMETER,           // 入力データが不正
   TSIP_ERR_PROHIBIT_FUNCTION,  // 不正な関数呼び出しが発生した
   TSIP_RESUME_FIRMWARE_GENERATE_MAC,//処理の続きがあります。API の再呼び出しが必要
   TSIP_ERR_VERIFICATION_FAIL,  // TLS1.3 のハンドシェイク検証が失敗
}e_tsip_err_t
```

2.9　FIT モジュールの追加方法

本モジュールは、使用するプロジェクトごとに追加する必要があります。ルネサスでは、Smart Configurator を使用した(1)、(3)の追加方法を推奨しています。ただし、Smart Configurator は、一部の RX デバイスのみサポートしています。サポートされていない RX デバイスについては(2)、(4)の方法を使用してください。

(1)　e^2 studio 上で Smart Configurator を使用して FIT モジュールを追加する場合
e^2 studio の Smart Configurator を使用して、自動的にユーザプロジェクトに FIT モジュールを追加します。詳細は、アプリケーションノート「Renesas e^2 studio スマート・コンフィグレータ ユーザーガイド (R20AN0451)」を参照してください。

(2)　e^2 studio 上で FIT Configurator を使用して FIT モジュールを追加する場合
e^2 studio の FIT Configurator を使用して、自動的にユーザプロジェクトに FIT モジュールを追加することができます。詳細は、アプリケーションノート「RX ファミリ e^2 studio に組み込む方法 Firmware Integration Technology (R01AN1723)」を参照してください。

(3)　CS+上で Smart Configurator を使用して FIT モジュールを追加する場合
CS+上で、スタンドアロン版 Smart Configurator を使用して、自動的にユーザプロジェクトに FIT モジュールを追加します。詳細は、アプリケーションノート「Renesas e^2 studio スマート・コンフィグレータ ユーザーガイド (R20AN0451)」を参照してください。

(4)　CS+上で FIT モジュールを追加する場合
CS+上で、手動でユーザプロジェクトに FIT モジュールを追加します。詳細は、アプリケーションノート「RX ファミリ CS+に組み込む方法 Firmware Integration Technology (R01AN1826)」を参照してください。

2.10 for 文、while 文、do while 文について

TSIP ドライバでは、レジスタの反映待ち処理等で for 文、while 文、do while 文（ループ処理）を使用しています。これらループ処理には、「WAIT_LOOP」をキーワードとしたコメントを記述しています。そのため、ループ処理にユーザがフェイルセーフの処理を組み込む場合は、「WAIT_LOOP」で該当の処理を検索できます。

「WAIT_LOOP」を記述している対象デバイス

・全てのデバイスグループ

以下に記述例を示します。

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

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

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

3.　TSIP ドライバの使用方法

RX ファミリ TSIP ドライバは、以下の機能を提供します。

- 乱数生成
- セキュアな鍵の管理
- 不正アクセス監視
- 暗号演算のアクセラレート
- TLS 処理のアクセラレート

TSIP ドライバが扱う鍵（入力する鍵、出力する鍵）は、TSIP のみがアクセス可能な Hardware Unique Key(HUK)と呼ばれるデバイス固有の鍵でラップされた不透明な鍵で、RX TSIP ドライバではこの不透明な鍵を Wrapped Key と呼びます。TSIP ドライバにおけるセキュアな鍵管理は、鍵を HUK でラップすることにより、TSIP の外部で鍵の秘匿と改ざん検知を実現します。

TSIP による不正アクセス監視は、本ドライバが提供するすべての暗号処理を対象とし、暗号処理中は常に有効です。本ドライバ使用中に暗号処理の改ざんを検出した場合、本ドライバは動作を停止します。

TSIP ドライバが暗号演算のアクセラレートのために提供する API には、暗号演算を一つの API で提供するものと複数の API で提供するものがあります。本書では、前者をシングルパート演算、後者をマルチパート演算と呼びます。

対称鍵暗号とハッシュは Init-Update-Final に分割されたマルチパート演算の API を提供しており、その他の暗号はシングルパート演算の API を提供しています。

3.1　不正アクセス検出からの復帰方法

TSIP による不正アクセス監視は、全ての暗号 API 実行時に常に有効です。本ドライバ使用中に暗号操作の改ざんを検出した場合、本ドライバは無限ループで動作を停止します。

TSIP ドライバの不正アクセスにより無限ループで動作停止しているかどうかは、ウォッチドッグタイマなどを使用してユーザアプリケーション側で検知する必要があります。

ユーザアプリケーションで不正アクセスを検知した場合は、ログ採取やシステムの再起動など、システムのセキュリティポリシーを満たす適切な処置を施してください。

不正アクセス検出からの復帰は、R_TSIP_Close()で TSIP ドライバを一度終了して、R_TSIP_Open()で TSIP を再度起動するか、デバイスをリセットしてください。

3.2　TSIP へのアクセス衝突回避

RX ファミリでは、すべての TSIP 搭載品において TSIP を 1 チャネルのみ利用することができます。TSIP ドライバは多くの周辺 IP のドライバと同じくドライバ API 実行中に TSIP のハードウェア資源を占有します。

マルチパート演算を提供する API のうち、対称鍵暗号および HMAC 関数は一連のマルチパート演算が終了するまで TSIP のハードウェア資源を占有し続けます。

このため、ユーザアプリケーションプログラムで TSIP ドライバを利用する際は、TSIP へのアクセス衝突を回避するため、以下の 2 点に注意してください。

1) TSIP ドライバ API 実行中に他の TSIP ドライバの API を実行することはできません。
2) 対称鍵暗号および HMAC 関数では、現在処理中の一連の演算処理(Init/Update/Final)が完了するまでは他の TSIP ドライバ API を実行することができません。

なお、メッセージダイジェスト生成関数はマルチパート演算の一連の演算処理の間に他の TSIP ドライバ API を実行することができます。

TSIP ドライバの API で TSIP のハードウェア資源のアクセス衝突が発生した場合、API は TSIP_ERR_RESOURCE_CONFLICT または TSIP_ERR_PROHIBIT_FUNCTION を返します。

TSIP ドライバを利用する際は、以下のいずれかの方法で TSIP へのアクセス衝突を回避してください。

- TSIP へのアクセス衝突が発生しないような順序で API を使用する
- TSIP ドライバが持つ TSIP のアクセス衝突回避機能を使用する
  リアルタイム OS の排他制御用のシステムコール（mutex や semaphore 等）を利用して、TSIP ドライバのユーザ定義関数である user_lock_function および user_unlock_function を実装してください。
  r_tsip_rx_config.h の TSIP_MULTI_THREADING を有効にし、TSIP ドライバが持つ TSIP のアクセス衝突回避機能を有効にします。
  リアルタイム OS 上の複数のスレッドで TSIP ドライバを同時に利用する場合等にご利用ください。

3.3　BSP FIT モジュールの組込み

TSIP ドライバは、2.2 章にあるように、内部で BSP FIT モジュールを使用しています。TSIP ドライバを使用する際には、以下の API をリンクしてください。詳細は、「ボードサポートパッケージモジュール

Firmware Integration Technology アプリケーションノート(R01AN1685xJxxxx)」を参照してください。

・R_BSP_RegisterProtectEnable()

・R_BSP_RegisterProtectDisable()

・R_BSP_InterruptsEnable()

・R_BSP_InterruptsDisable()

また、これらの API が呼び出される前に、BSP のスタートアップが完了していることを想定しています。BSP のスタートアップを使用しない場合、事前に R_BSP_StartupOpen()を呼び出してください。上記 API 内で使用する内部変数の初期化を行います。

3.4　シングルパート演算とマルチパート演算

TSIP ドライバが暗号演算のアクセラレートのために提供する API には、暗号演算を一つの API で提供するものと複数の API で提供するものがあります。本書では、前者をシングルパート演算、後者をマルチパート演算と呼びます。

対称鍵暗号とハッシュ（メッセージダイジェスト生成関数・HMAC 関数）はマルチパート演算の API を提供しており、その他の暗号はシングルパート演算の API を提供しています。

マルチパート演算とは、1 つの暗号演算を Init-Update-Final のステップに分割する API です。これにより、暗号演算を細かく制御することができ、メッセージデータを一度に処理するのではなく、断続的に処理することが可能になります。

すべてのマルチパート演算は以下のパターンに従っています。

Init: 演算を初期化し、開始します。

成功時、演算はアクティブになります。失敗すると、演算はエラー状態になります。

Update: 演算を更新します。

更新機能は、追加のパラメータを提供したり、処理のためのデータを供給したり、出力を生成したりする

ことができます。
成功した場合、操作はアクティブなままです。失敗すると、操作はエラー状態になります。

Final: 操作を終了するには、該当するファイナライズ関数を呼び出します。
この関数は、最終的な入力を受け取り、最終的な出力を生成し、操作に関連するすべてのリソースを解放します。
成功すると、操作は非アクティブ状態に戻ります。失敗した場合、操作はエラー状態になります。

3.5　初期化と終了

本ドライバは、以下のようなドライバ管理のための API を提供します。

No.	API	説明
1	R_TSIP_Open	TSIP ドライバの開始処理を行います。 TSIP の初期化、TSIP の故障検出回路と乱数発生回路のセルフテストを行います。
2	R_TSIP_Close	TSIP ドライバの終了処理を行います。
3	R_TSIP_SoftwareReset	TSIP をリセットします。
4	R_TSIP_GetVersion	TSIP ドライバのバージョンを取得します。

本ドライバを使用するアプリケーションは、他の関数を使用する前に R_TSIP_Open() を呼び出してドライバを初期化する必要があります。また、本ドライバの使用を終了する場合は、R_TSIP_Close()を呼び出す必要があります。

ドライバの使用中に何らかの問題が発生し、ドライバとその制御対象である TSIP をリセットしたい場合は、R_TSIP_Close()を呼び出した後に R_TSIP_SoftwareReset()または R_TSIP_Open()を呼び出す必要があります。TSIP ドライバの処理を再開しない場合は R_TSIP_SoftwareReset()、TSIP ドライバの処理を再開したい場合は R_TSIP_Open()を呼び出してください。

R_TSIP_Open()では、TSIP のハードウェア障害の検出と乱数生成回路に異常がないことを確認するセルフテストが行われます。乱数生成回路のセルフテストでは、物理乱数生成器が生成するデータに対して NIST SP800-90B に記載されているヘルステストを用いてエントロピーの評価を行い、乱数のシードを生成します。

3.6　乱数生成

本ドライバは、乱数生成のための API を提供します。

No.	API	説明
1	R_TSIP_GenerateRandomNumber	NIST SP800-90A に記載されている CTR-DRBG 法を用いて乱数を生成します。

3.7　鍵の管理

本ドライバは、以下の種類の鍵管理操作のための API を提供します。

No.	API	説明
1	R_TSIP_GenerateUpdateKeyRingKeyIndex R_TSIP_GenerateAes*XXX*KeyIndex R_TSIP_GenerateTdesKeyIndex R_TSIP_GenerateArc4KeyIndex R_TSIP_GenerateSha*XXX*HmacKeyIndex R_TSIP_GenerateRsa*XXX*PublicKeyIndex R_TSIP_GenerateRsa*XXX*PrivateKeyIndex R_TSIP_GenerateEccP*XXX*PublicKeyIndex R_TSIP_GenerateEccP*XXX*PrivateKeyIndex R_TSIP_GenerateTlsRsaPublicKeyIndex	Renesas Key Wrap service を使って、ユーザ鍵を HUK でラップされた Wrapped Key に変換する鍵注入 API です。工場出荷時の鍵の注入に利用することができます。 [Aes] XXX = 128, 256 [Hmac] XXX = 1, 256 [Rsa] XXX = 1024, 2048, 3072 注, 4096 注 [Ecc] XXX = 192, 224, 256, 384
2	R_TSIP_UpdateAes*XXX*KeyIndex R_TSIP_UpdateTdesKeyIndex R_TSIP_UpdateArc4KeyIndex R_TSIP_UpdateRsa*XXX*PublicKeyIndex R_TSIP_UpdateRsa*XXX*PrivateKeyIndex R_TSIP_UpdateEccP*XXX*PublicKeyIndex R_TSIP_UpdateEccP*XXX*PrivateKeyIndex	鍵更新用鍵束を使って、ユーザ鍵を HUK でラップされた Wrapped Key に変換する鍵更新 API です。フィールドでの鍵の更新に使用することができます。 [Aes] XXX = 128, 256 [Hmac] XXX = 1, 256 [Rsa] XXX = 1024, 2048, 3072 注, 4096 注 [Ecc] XXX = 192, 224, 256, 384
3	R_TSIP_GenerateAes*XXX*RandomKeyIndex R_TSIP_GenerateTdesRandomKeyIndex R_TSIP_GenerateArc4RandomKeyIndex R_TSIP_GenerateRsa*XXX*RandomKeyIndex R_TSIP_GenerateEccP*XXX*RandomKeyIndex	ランダムな鍵を生成し Wrapped Key に変換します。 [Aes] XXX = 128, 256 [Rsa] XXX = 1024, 2048 [Ecc] XXX = 192, 224, 256, 384

注　　これらの鍵長は公開鍵のみ提供します。

図 3-1 に TSIP ドライバの暗号操作における鍵の取り扱いを示します。

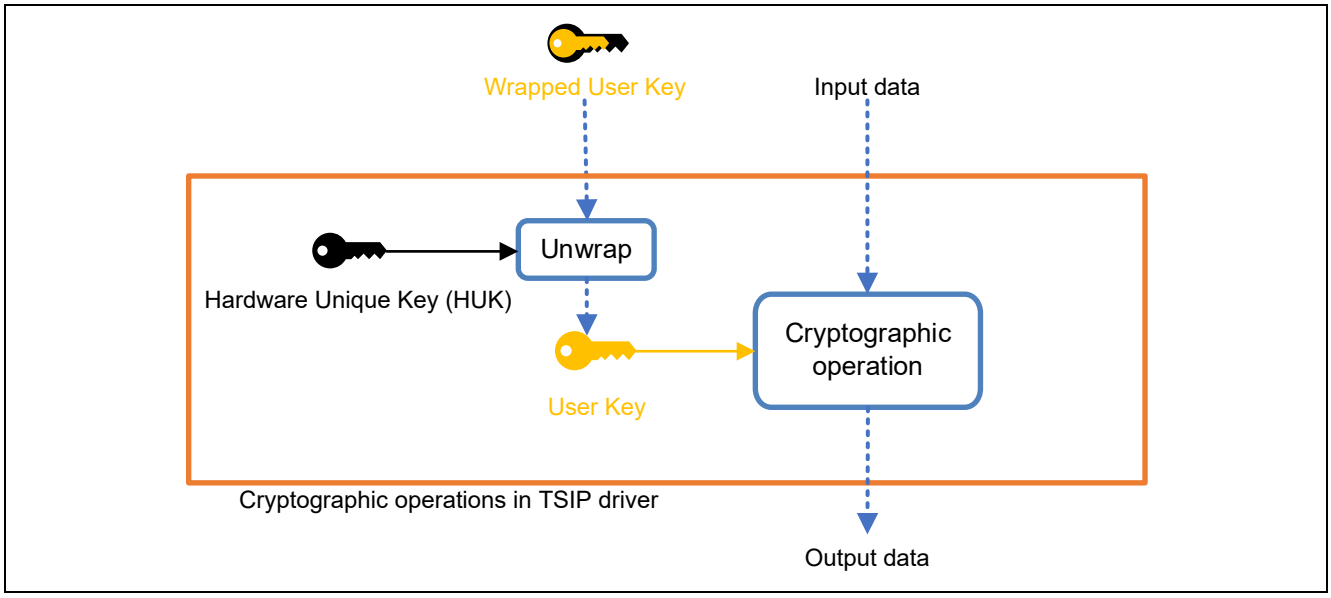

図 3-1　TSIP ドライバ暗号操作における鍵の取り扱い

TSIP ドライバの暗号演算で扱う鍵（入出力する鍵）は、TSIP だけがアクセス可能な HUK と呼ばれるデバイス固有の鍵でラップされた不透明な鍵です。これを RX TSIP ドライバでは Wrapped Key と呼びます。TSIP ドライバにおけるセキュアな鍵管理は、ユーザ鍵をデバイス固有の鍵でラップすることにより、TSIP

の外部で鍵の秘匿と改ざん検知を実現します。Wrapped Key のラッピングを解除できるのは TSIP のみであり、ラッピング解除された鍵は、暗号処理中に TSIP の内部のみに存在します。Wrapped Key はデバイス固有の鍵でラップされているため、不揮発メモリ上の Wrapped Key を別のデバイスにコピーしても、別のデバイス固有の鍵で Wrapped Key をアンラップすることはできません。

3.7.1　鍵の注入と更新

鍵注入および鍵更新はユーザ鍵のセキュアな配送を可能にする機構を備えており、ユーザ鍵を HUK でラップした Wrapped Key に変換します。図 3.2 に Renesas Key Wrap service を含む鍵の注入と更新のフローを示します。

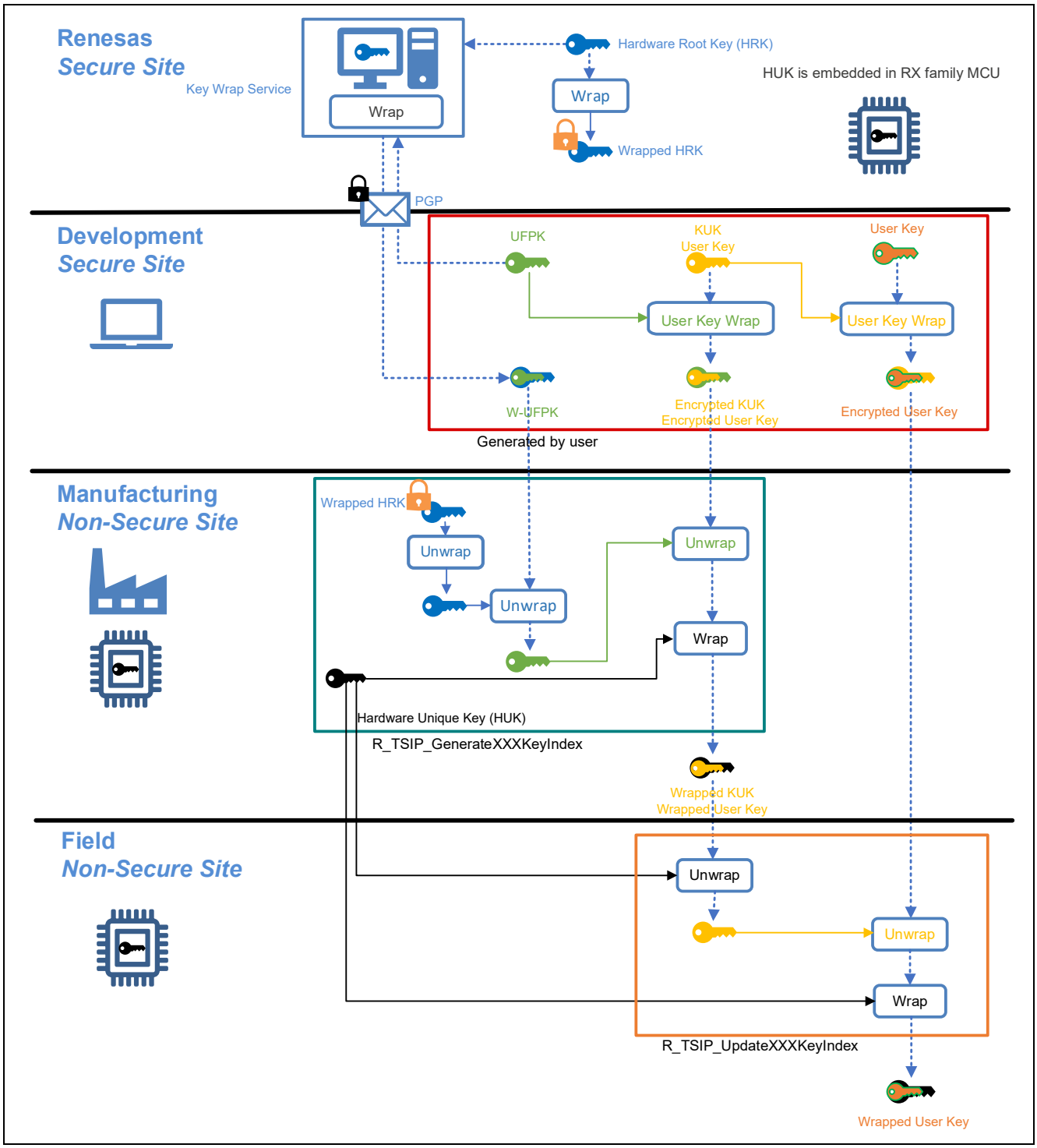

図 3-2 鍵の注入と更新のフロー

HUK による秘密鍵のラップは暗号化と MAC の付与、公開鍵のラップは MAC の付与のみを行います。公開鍵の Wrapped Key は暗号化されていないため、公開鍵の Wrapped Key から平文の公開鍵を取り出すことが可能です。詳細は 3.7.3 平文公開鍵の抽出を参照してください。

鍵注入および鍵更新において、ユーザ鍵を UFPK や鍵更新用鍵（Key Update Key, 以下 KUK）でラップする際のユーザ鍵ラップ方式を図 3-3 に示します。ラップに使用する UFPK または KUK の先頭 128bit を鍵として、AES-128 CBC モードでユーザ鍵を暗号化します。ラップに使用する UFPK または KUK の後続 128bit を鍵として、AES-128 CBC-MAC でユーザ鍵の MAC を計算します。ユーザ鍵にユーザ鍵の MAC を連結し、それらを暗号化することで Encrypted User Key を生成します。

暗号化するユーザ鍵（User Key）のデータフォーマットと、ラップされた鍵(Encrypted User Key)のデータフォーマットは 7.3 ユーザ鍵暗号化フォーマットを参照してください。

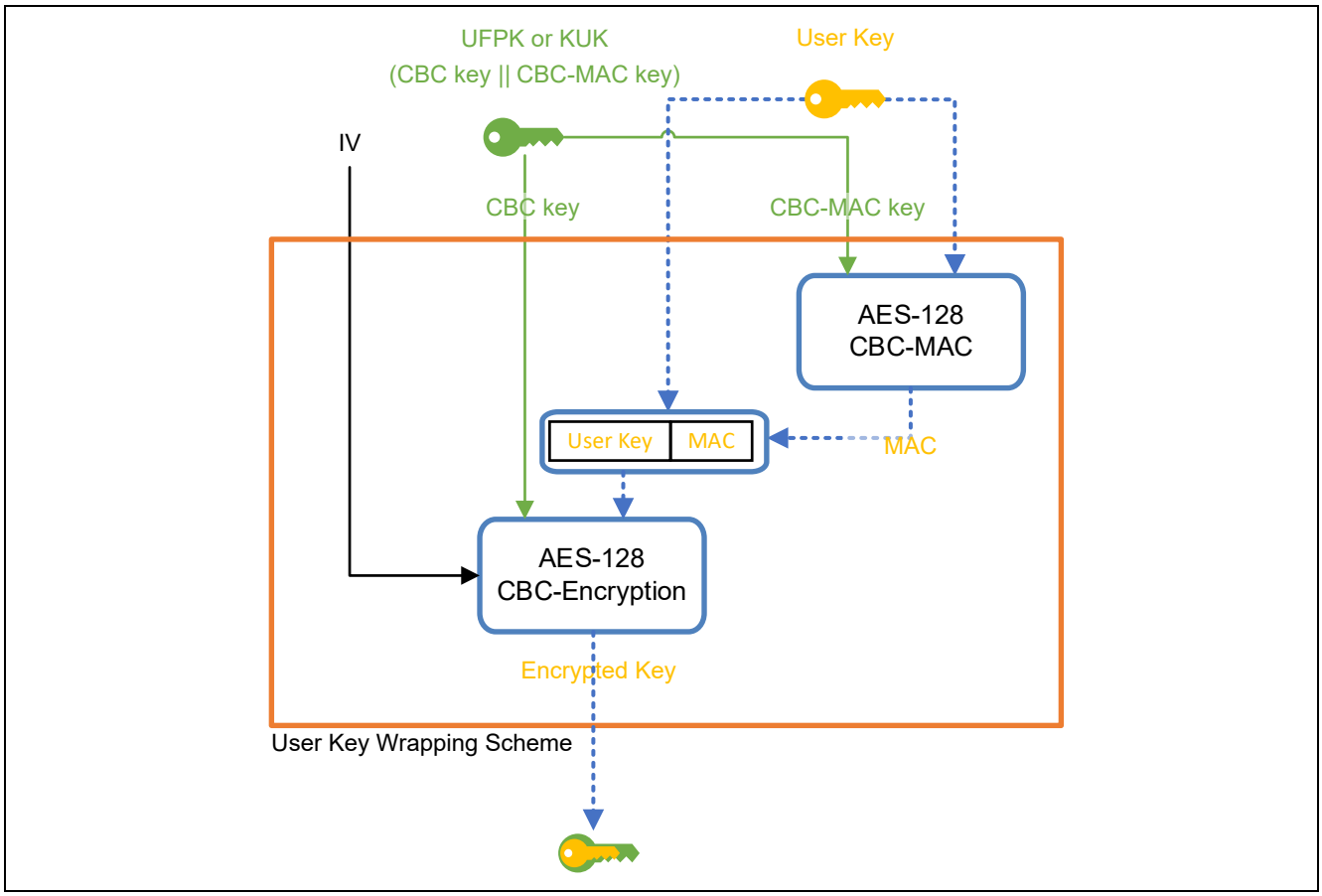

図 3-3 鍵注入および鍵更新時のユーザ鍵ラップ方式

ユーザ鍵の暗号化方法を C 言語の形式で表すと、以下のようになります。

```
uint32_t user_key[len];
uint32_t MAC[4] = 0;
uint32_t iv[4] = IV;
for (i = 0; i < len; i += 4)
{
        MAC  = AES_128_ENCRYPT(CBCMACkey[0: 3], xor_16byte(user_key[i: i+3], MAC[0: 3]));
        encrypted_key[i: i+3]
                = AES_128_ENCRYPT(CBCkey[0: 3], xor_16byte(user_key[i: i+3], iv[0: 3]));
        iv[0: 3] = encrypted_key [i: i+3];
}
encrypted_key[i: i+3] = AES_128_ENCRYPT(CBCkey[0: 3], xor_16byte(MAC[0: 3], iv[0: 3]));
```

ここで、使用している関数は以下の処理を意味します。
・AES_128_ENCRYPT(Key, Data)：暗号鍵 Key を用いた Data の AES128 ECB モードでの暗号化
・xor_16byte(data1, data2)：16 バイトの data1 と data2 の XOR 演算
また、各配列(CBCkey[]、CBCMACkey[]、MAC[]、iv[]、user_key[]、encrypted_key[])の要素 1 個分の大きさは 4 バイトです。

鍵更新に使用する KUK を含むユーザ鍵は、ユーザが作成し、製品製造時にデバイスに注入する必要があります。詳細な手順は 5.1 鍵の注入および 5.2 鍵の更新を参照してください。

3.7.2　鍵の生成

鍵の生成は、TSIP の乱数生成機能を用いて新しい鍵を生成し、TSIP ドライバが利用できる不透明な鍵の形式（Wrapped User Key）で出力します。

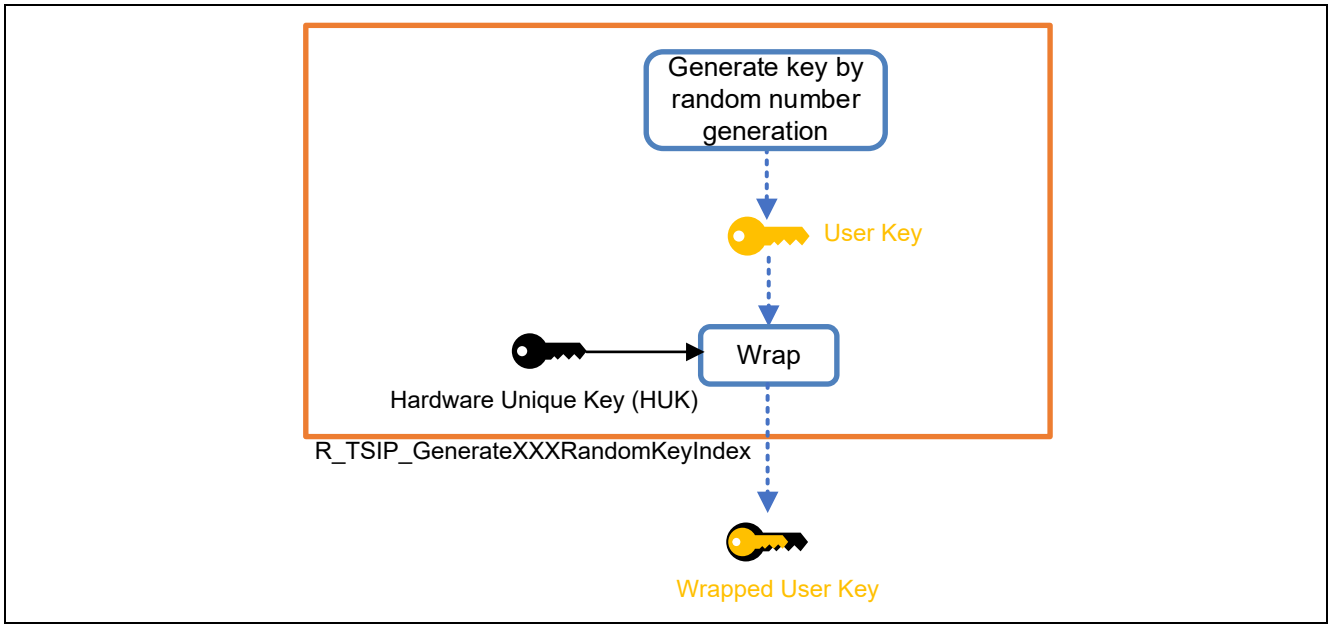

図 3-4 鍵生成のフロー

3.7.3　平文公開鍵の抽出

公開鍵の Wrapped Key は暗号化されていないため、公開鍵の Wrapped Key から平文の公開鍵を取り出すことが可能です。

非対称鍵暗号の各アルゴリズムにおける公開鍵の Wrapped Key のフォーマットを、7.4 非対称鍵暗号 公開鍵の Wrapped Key フォーマットに記載していますので、公開鍵の Wrapped Key から平文の公開鍵を抽出する際に参照してください。

3.8　対称鍵暗号

本ドライバは，以下の種類の共通暗号演算のための API を提供します。

No.	API	説明
1	R_TSIP_Aes*XXX[Mode]*Encrypt* R_TSIP_Aes*XXX[Mode]*Decrypt* R_TSIP_Aes*XXX*Ctr* R_TSIP_Tdes*[Mode]*Encrypt* R_TSIP_Tdes*[Mode]*Decrypt* R_TSIP_Arc4Encrypt* R_TSIP_Arc4Decrypt*	Symmetric ciphers AES 128/256bit: ECB, CBC, CTR encryption and decryption TDES: ECB, CBC encryption and decryption ARC4 *XXX*=128, 256 *Mode*=Ecb, Cbc
2	R_TSIP_Aes*XXX*GcmEncrypt* R_TSIP_Aes*XXX*GcmDecrypt* R_TSIP_Aes*XXX*CcmEncrypt* R_TSIP_Aes*XXX*CcmDecrypt*	Authenticated encryption with associated data (AEAD) AES-GCM, AES-CCM 128/256bit encryption and decryption *XXX*=128, 256
3	R_TSIP_Aes*XXX*CmacGenerate* R_TSIP_Aes*XXX*CmacVerify*	Message authentication codes (MAC) AES-CMAC 128/256bit MAC operation *XXX*=128, 256
4	R_TSIP_AES*XXX*KeyWrap R_TSIP_AES*XXX*KeyUnwrap	AES Key Wrap/ Unwrap *XXX*=128, 256

*= Init, Update, Final

対称鍵暗号演算の種類ごとに、マルチパート演算を可能にする一連の関数を API として提供します。マルチパート演算の詳細については、「3.4 シングルパート演算とマルチパート演算」を参照してください。

3.8.1　対称鍵暗号（Symmetric ciphers）

各 AES モードの暗号化処理は、以下のように行います。

R_TSIP_AesXXX[Mode]EncryptInit() を呼び出して、必要な鍵と初期ベクトルを指定します。

R_TSIP_AesXXX[Mode]EncryptUpdate() 関数を、連続したブロック単位の平文メッセージをまとめたデータに対して呼び出します。

暗号化処理を完了するには、R_TSIP_AesXXX[Mode]EncryptFinal() を呼び出します。

各 AES モードの復号処理は、以下のように行います。

R_TSIP_AesXXX[Mode]DecryptInit() を呼び出して、必要な鍵と初期ベクトルを指定します。

R_TSIP_AesXXX[Mode]DecryptUpdate() 関数を連続したブロック単位の暗号文メッセージをまとめたデータに対して呼び出します。

復号処理を完了させるには、 R_TSIP_AesXXX[Mode]DecryptFinal() を呼び出します。

TDES および ARC4 の暗号 API の使用方法は AES と同じです。

3.8.2　認証付き暗号 (AEAD)

AES-GCM 暗号化処理は、以下のように行います:

R_TSIP_AesXXXGcmEncryptInit() をコールして、必要な鍵と初期ベクトルを指定します。

R_TSIP_AesXXXGcmEncryptUpdate() 関数を、連続したブロック単位の平文メッセージをまとめたデータと追加データに対して呼び出します。

暗号化処理を完了し、認証タグを計算するために、R_TSIP_AesXXXGcmEncryptFinal() を呼び出します。

AES-GCM の復号処理は、以下のように行います:

R_TSIP_AesXXXGCMDecryptInit() をコールして、必要な鍵と初期ベクトルを指定します。

R_TSIP_AesXXXGCMDecryptUpdate() 関数を、連続したブロック単位の暗号文メッセージをまとめたデータと追加データに対して呼び出します。

復号処理を完了し、認証タグを計算し、参照値と照合するために、 R_TSIP_AesXXXGcmDecryptFinal() を呼び出します。

AES-CCM の暗号 API の使用方法は AES-GCM と同じです。

3.8.3　メッセージ認証コード (MAC)

AES-CMAC による MAC 生成は、以下のように行います:

R_TSIP_AesXXXCmacGenerateInit() を呼び出して、必要な鍵を指定します。

連続したメッセージをまとめたデータに対して R_TSIP_AesXXXCmacGenerateUpdate() 関数を呼び出します。

メッセージの MAC 生成を完了するには、R_TSIP_AesXXXCmacGenerateFinal() を呼び出します。

AES-CMAC 検証は、以下のように行います:

R_TSIP_AesXXXCmacVerifyInit() を呼び出して、必要な鍵を指定します。

メッセージをまとめたデータに対して R_TSIP_AesXXXCmacVerifyUpdate() 関数を呼び出します。

メッセージの MAC を検証するには、R_TSIP_AesXXXCmacVerifyFinal()を呼び出して、検証に必要な MAC を指定します。

3.9　非対称鍵暗号

本ドライバは、以下の非対称暗号操作のための API を提供します。

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

非対称鍵暗号の暗号化と復号および署名生成と検証の各 API はいずれもシングルパート演算のみ提供します。

3.10 HASH 関数

本ドライバは、以下のハッシュ演算のための API を提供します。

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

*= Init, Update, Final

ハッシュ演算の種類ごとに、マルチパート演算を可能にする一連の API を提供します。マルチパート演算の詳細については、「3.4 シングルパート演算とマルチパート演算」を参照してください。

3.10.1 メッセージダイジェスト (hash functions)

ハッシュ演算 API は次のように使用します:

R_TSIP_ShaXXXInit() を呼び出して、演算用に新しく割り当てたワーク領域を指定します。

連続したメッセージをまとめたデータに対して R_TSIP_ShaXXXUpdate() を呼び出します。

メッセージのダイジェストを計算するには、R_TSIP_ShaXXXFinal() を呼び出します。

R_TSIP_ShaXXXUpdate()の後で、R_TSIP_GetCurrentHashDigestValue()を呼び出すことで、ハッシュ演算の途中経過データを取り出すことができます。

MD5 API の使用方法は SHA と同じです。

3.10.2 メッセージ認証コード (HMAC)

HMAC 生成 API は以下のように使用します:

R_TSIP_ShaXXXHmacGenerateInit() を呼び出して、必要な鍵と演算用に新しく割り当てたワーク領域を指定します。

連続したメッセージをまとめたデータに対して R_TSIP_ShaXXXHmacGenerateUpdate()を呼び出します。

メッセージの MAC 生成を完了するには、R_TSIP_ShaXXXHmacGenerateFinal() を呼び出します。

HMAC 検証 API は以下のように使用します:

R_TSIP_ShaXXXHmacVerifyInit() を呼び出して、必要な鍵と演算用に新しく割り当てたワーク領域を指定します。

連続したメッセージをまとめたデータに対して R_TSIP_ShaXXXHmacVerifyUpdate()を呼び出します。

メッセージの MAC を検証するには、R_TSIP_ShaXXXHmacVerifyFinal() を呼び出して、検証に必要な MAC を指定します。

3.11 ECDH 鍵交換

本ドライバは、以下の ECDH 鍵交換のための API を提供します。

No.	API	説明
1	R_TSIP_EcdhP256Init	ECDH 鍵交換の初期化
2	R_TSIP_EcdhP256ReadPublicKey	ECDH 鍵交換相手の公開鍵と署名を検証し、公開鍵を取り出す
3	R_TSIP_EcdhP256MakePublicKey	乱数から一時的に鍵(Ephemeral Key)を生成と署名を生成する
4	R_TSIP_EcdhP256CalculateSharedSecretIndex	ECDH 鍵交換アルゴリズムで共有秘密”Z”を計算する
5	R_TSIP_EcdhP256KeyDerivation	共有秘密”Z”から鍵を導出する

ECDH 鍵交換 API は、NIST SP800-56A に準拠しています。

鍵交換において、あらかじめ注入した鍵(Static Key)を用いるか、乱数から一時的に生成した鍵(Ephemeral Key)を用いるかによって API の使用法が異なります。NIST SP800-56A で定義された鍵交換スキーム Ephemeral Unified Model, One-Pass Diffie-Hellman, Static Unified Model の場合の本 API の使用法を以下で説明します。

3.11.1 Ephemeral Unified Model (2e, 0s)

Ephemeral Unified Model では、鍵交換相手(Party U)と自身(Party V)の間で一時的な鍵ペアを生成し、ECDH 鍵交換を行います。

1. 鍵交換スキームの初期化のために、R_TSIP_EcdhP256Init()を実行してください。
   R_TSIP_EcdhP256Init()の引数 key_type には 0 (ECDHE)を指定してください。
2. Party U から受け取った公開鍵(QeU)と署名(sig)を R_TSIP_EcdhP256ReadPublicKey()で検証します。署名が正しければ、QeU_index を出力します。
3. R_TSIP_EcdhP256MakePublicKey()を実行して、Ephemeral 鍵ペア(QeV, deV_index)を生成します。
4. R_TSIP_EcdhP256CalculateSharedSecretIndex()に QeU_index と deV_index を入力して shared_secret_index を生成します。
5. R_TSIP_EcdhP256KeyDerivation()に shared_secret_index と other information を入力して、鍵を導出します。other information は Party U と Party V 間で同一の値を利用してください。

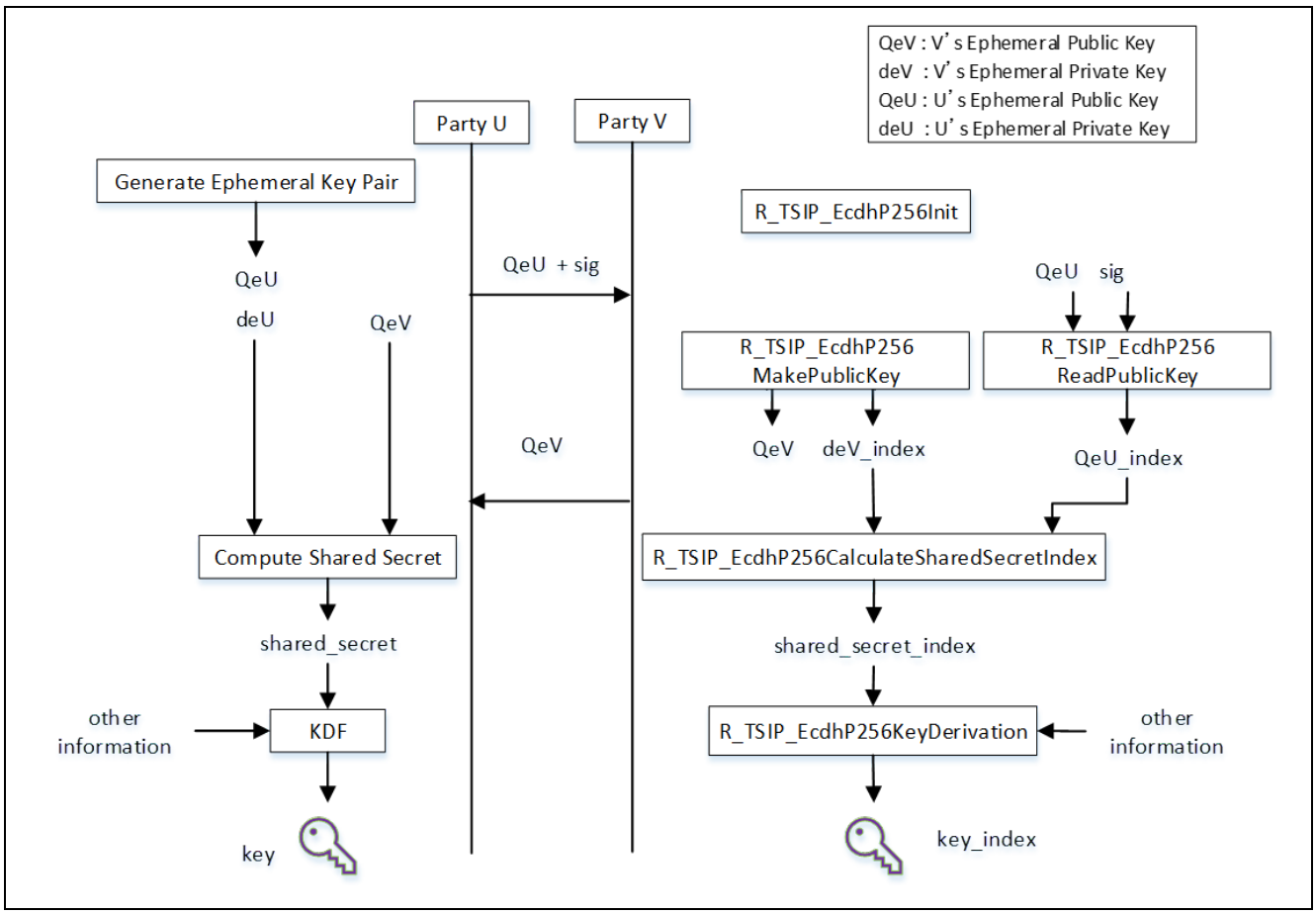

図 3-5 Ephemeral Unified Model を使用した鍵交換スキーム

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

One-Pass Diffie-Hellman では、鍵交換相手(Party U)が生成した一時的な鍵と自身(Party V)の静的な鍵を使い ECDH 鍵交換を行います。鍵交換スキームを開始する前に、Party U は Party V の公開鍵(QsV)を入手しておく必要があります。

1. 鍵交換スキームの初期化のために、R_TSIP_EcdhP256Init()を実行してください。
   R_TSIP_EcdhP256Init()の引数 key_type には 1 (ECDH)を指定してください。
2. Party U から受け取った公開鍵(QeU)と署名(sig)を R_TSIP_EcdhP256ReadPublicKey()で検証します。
   署名が正しければ、QeU_index を出力します。
3. Party U が所持する Party V の公開鍵(QsV)と対になる秘密鍵(dsV)の Wrapped Key(dsV_index)を生成してください。
4. R_TSIP_EcdhP256CalculateSharedSecretIndex()に QeU_index と dsV_index を入力して shared_secret_index を生成します。
5. R_TSIP_EcdhP256KeyDerivation()に shared_secret_index と other information を入力して、鍵を導出します。other information は Party U と Party V 間で同一の値を利用してください。

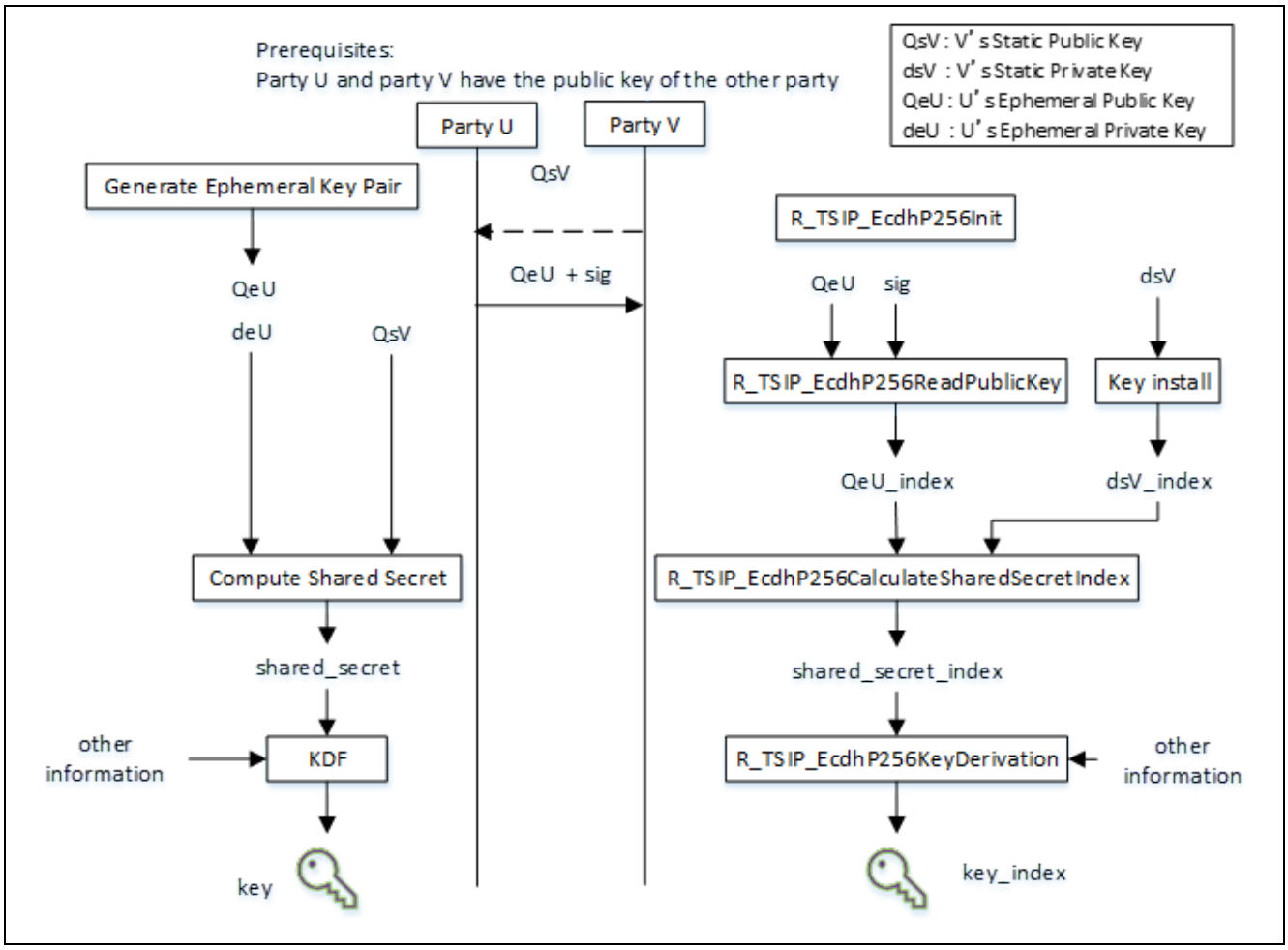

図 3-6 One-Pass Diffie-Hellman を使用した鍵交換スキーム

3.11.3 Static Unified Model (0e, 2s)

Static Unified Model では、鍵交換相手(Party U)の静的な鍵と自身(Party V)の静的な鍵を使い ECDH 鍵交換を行います。鍵交換スキームを開始する前に、お互いの公開鍵を入手しておく必要があります。

1. 鍵交換スキームの初期化のために、R_TSIP_EcdhP256Init()を実行してください。
   R_TSIP_EcdhP256Init()の引数 key_type には 1 (ECDH)を指定してください。
2. Party U の公開鍵(QsU)と署名(sig)を R_TSIP_EcdhP256ReadPublicKey()で検証します。署名が正しければ、QsU_index を出力します。
3. Party U が所持する Party V の公開鍵(QsV)と対になる秘密鍵(dsV)の Wrapped Key(dsV_index)を生成してください。
4. R_TSIP_EcdhP256CalculateSharedSecretIndex()に QsU_index と dsV_index を入力して shared_secret_index を生成します。
5. R_TSIP_EcdhP256KeyDerivation()に shared_secret_index と other information を入力して、鍵を導出します。other information は Party U と Party V 間で同一の値を利用してください。また、other information には Party U が生成した Nonce を含めてください。これは、静的な鍵を利用した ECDH 鍵交換によって毎回同じ鍵が導出されることを防ぐためです。

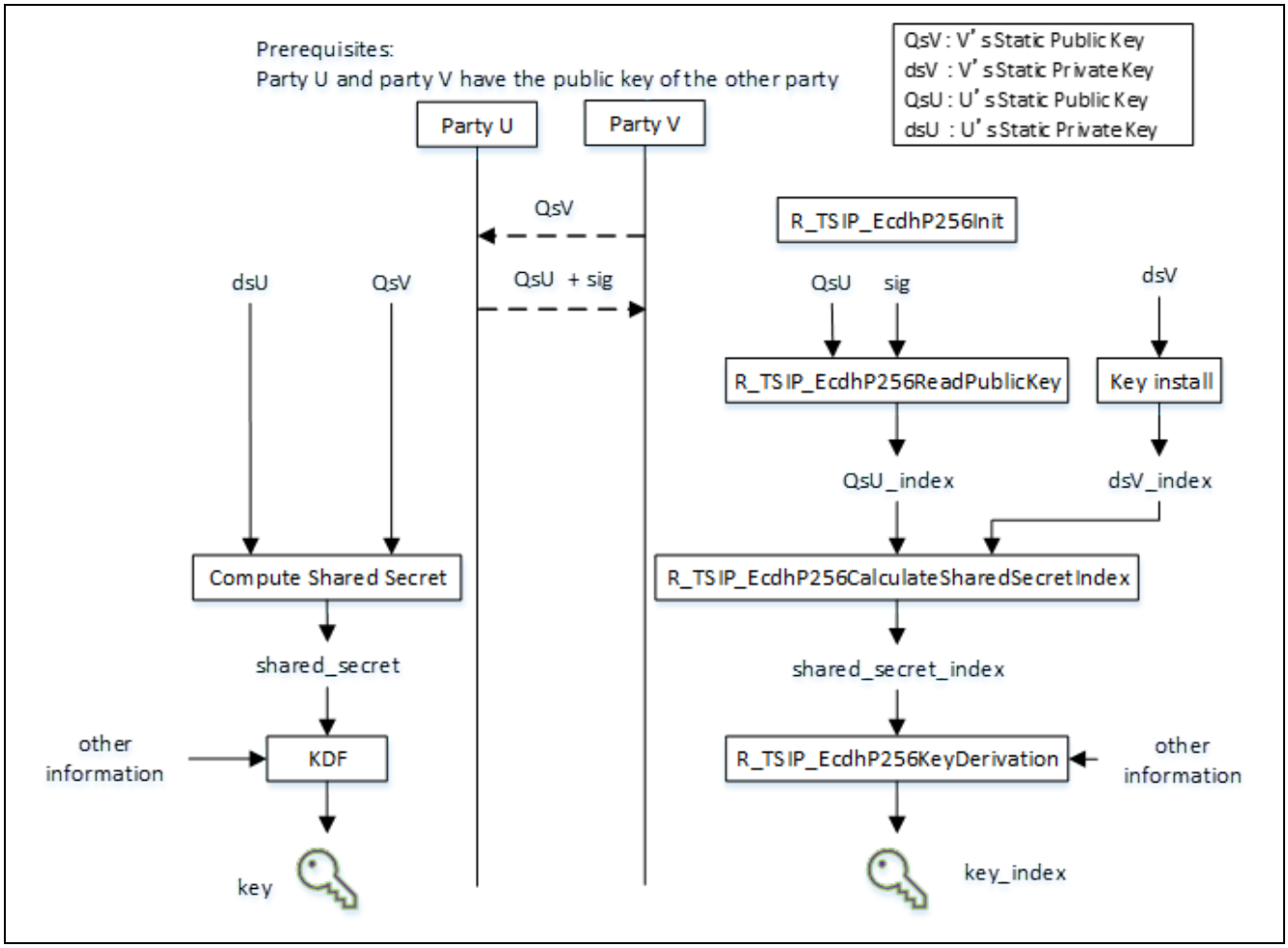

図 3-7 Static Unified Model を使用した鍵交換スキーム

3.12 TLS 連携機能(TLS1.2)

TLS 連携機能(TLS1.2)の API は、RFC5246 に準拠しています。

TLS 連携機能の内、TLS1.2 連携機能として、TLS1.2 クライアント機能および TLS1.2 サーバ機能をサポートしています。それぞれの機能の使用方法について示します。

3.12.1 TLS1.2 クライアント機能

No.	API	説明
1	R_TSIP_TlsRegisterCaCertificationPublicKeyIndex	TLS 連携機能で共通に使用
2	R_TSIP_GenerateTlsRsaPublicKeyIndex R_TSIP_UpdateTlsRsaPublicKeyIndex R_TSIP_TlsRootCertificateVerification R_TSIP_TlsCertificateVerification R_TSIP_TlsCertificateVerificationExtension	TLS1.2/TLS1.3 クライアント機能で共通に使用
3	R_TSIP_TlsGenerateServerRandom R_TSIP_TlsEncryptPreMasterSecretWithRsa2048PrivateKey R_TSIP_TlsGenerateMasterSecret R_TSIP_TlsGenerateSessionKey R_TSIP_TlsGenerateVerifyData R_TSIP_TlsServersEphemeralEcdhPublicKeyRetrieves R_TSIP_GenerateTlsP256EccKeyIndex R_TSIP_TlsGeneratePreMasterSecretWithEccP256Key R_TSIP_TlsGenerateExtendedMasterSecret R_TSIP_TlsCertificateVerifyVerification	TLS1.2 クライアント機能で使用
4	R_TSIP_Aes128CbcEncryptInit/Update/Final R_TSIP_Aes256CbcEncryptInit/Update/Final R_TSIP_Aes128CbcDecryptInit/Update/Final R_TSIP_Aes256CbcDecryptInit/Update/Final R_TSIP_Aes128GcmEncryptInit/Update/Final R_TSIP_Aes256GcmEncryptInit/Update/Final R_TSIP_Aes128GcmDecryptInit/Update/Final R_TSIP_Aes256GcmDecryptInit/Update/Final R_TSIP_Sha1HmacGenerateInit/Update/Final R_TSIP_Sha1HmacVerifyInit/Update/Final R_TSIP_Sha256HmacGenerateInit/Update/Final R_TSIP_Sha256HmacVerifyInit/Update/Final R_TSIP_EcdsaP256SignatureGenerate R_TSIP_RsassaPkcs2048SignatureGenerate R_TSIP_RsassaPss2048SignatureGenerate	汎用 API を TLS1.2 連携機能で使用

TLS1.2 クライアント機能を使用した TLS 通信の実施方法の概要を図 **3-8** に示します。

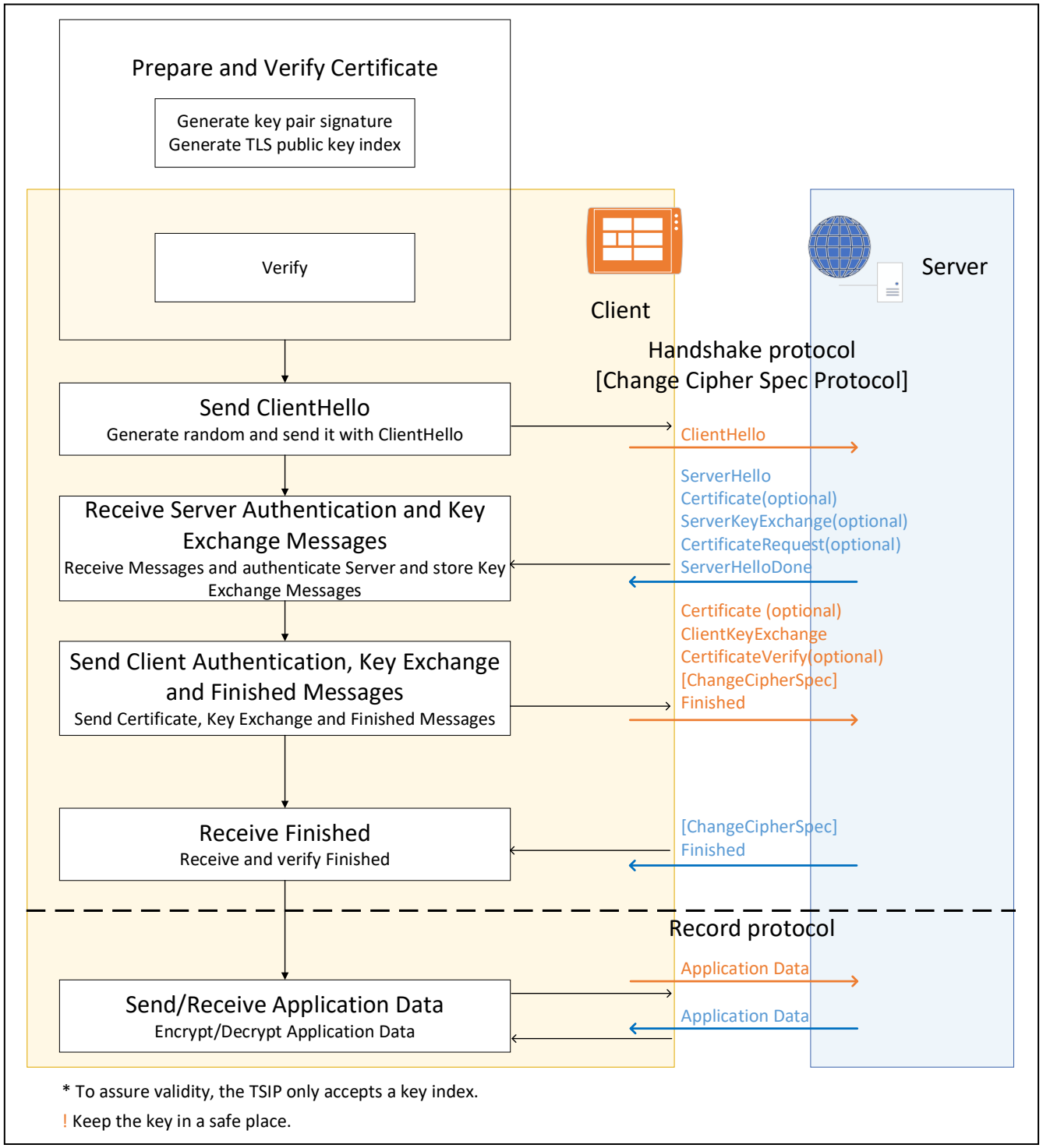

図 3-8 TLS1.2 クライアント機能の実施方法

1. 事前準備と証明書の検証（Prepare and verify certificate）

1.1 事前に以下のものを準備します。

- 接続するサーバのルート CA 証明書
- RSA2048bit の鍵ペア
- クライアント証明書

1.2 用意した RSA 鍵ペアの秘密鍵を使って、用意したルート CA 証明書の署名値を生成(署名アルゴルズムは RSA2048 PSS with SHA256)します。また用意した RSA 鍵ペアの公開鍵は、UFPK でラップし、Encrypted Key にします。これらの作業は安全なサイトで行ってください。

接続するサーバが複数あり、ルート CA 証明書を複数登録したい場合は、束としてルート CA 証明書をまとめたデータに対して、署名を付けてください。ルート CA 証明書に署名することで、意図しないサーバへの接続を防ぐことができます。

1.3 RSA 鍵ペアの公開鍵の Encrypted Key を入力として、R_TSIP_TlsRsaPublicKeyIndex を使用して、公開鍵の Wrapped Key を生成します。詳細な手順については、3.7.1 鍵の注入と更新を参照してください。

1.4 R_TSIP_TlsRegisterCaCertificationPublicKeyIndex()もしくは R_TSIP_Open()を使用して、公開鍵の Wrapped Key を TSIP ドライバへ登録します。R_TSIP_Open()を使用して登録する場合は、R_TSIP_Close()を呼んだ後、R_TSIP_Open()を呼んでください。

1.5 R_TSIP_TlsRootCertificateVerification()を使用して、ルート CA 証明書の束を検証し、検証に成功したらルート CA 証明書から公開鍵を抽出し、暗号化して出力します。

1.6 暗号化されたルート CA 証明書の公開鍵、およびクライアント証明書を入力として、R_TSIP_TlsCertificateVerification()または R_TSIP_TlsCertificateVerificationExtension()を使用してクライアント証明書を検証し、検証に成功したらクライアント証明書から公開鍵を抽出し、暗号化して出力します。

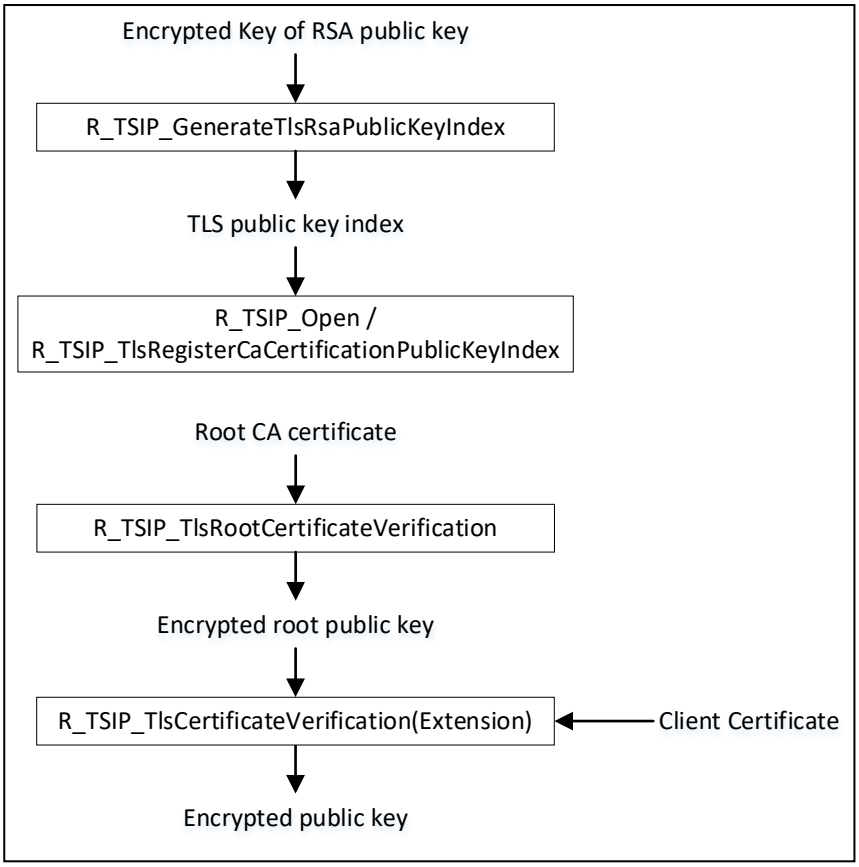

図 3-9 事前準備と証明書の検証

2. ClientHello の送信（Send ClientHello）

2.1 ClientHello を送信します。

3. 認証/鍵交換メッセージの受信（Receive Server Authentication and Key Exchange Messages）

3.1 ServerHello と ServerHello の random フィールドを受信します。

3.2 (Server) Certificate を受信した場合、R_TSIP_TlsCertificateVerification()または R_TSIP_TlsCertificateVerificationExtension()を使用して、暗号化したサーバ証明書の公開鍵を生成します。

3.3-a 鍵交換方式が ECDHE の場合、ServerKeyExchange から署名と ECDH 公開鍵を取り出して Pre Master Secret を生成します。

3.3-a.1 R_TSIP_GenerateTlsP256EccKeyIndex()を使用して、ECC 鍵ペアを生成します。

3.3-a.2 ServerKeyExchange の署名と ECDH 公開鍵、(Server) Certificate の公開鍵、及びハンドシェイクメッセージを入力として、R_TSIP_TlsServersEphemeralEcdhPublicKeyRetrieves()を使用して、暗号化した ephemeral ECDH 公開鍵を生成します。

3.3-a.3 ephemeral ECDH 公開鍵と 3.3-a.1 で生成した ECC 秘密鍵の Wrapped Key を入力として、R_TSIP_TlsGeneratePreMasterSecretWithEccP256Key()を使用して、暗号化した ephemeral Pre Master Secret を生成します。

3.3-b 鍵交換方式が RSA の場合、ここで実施する処理はありません。

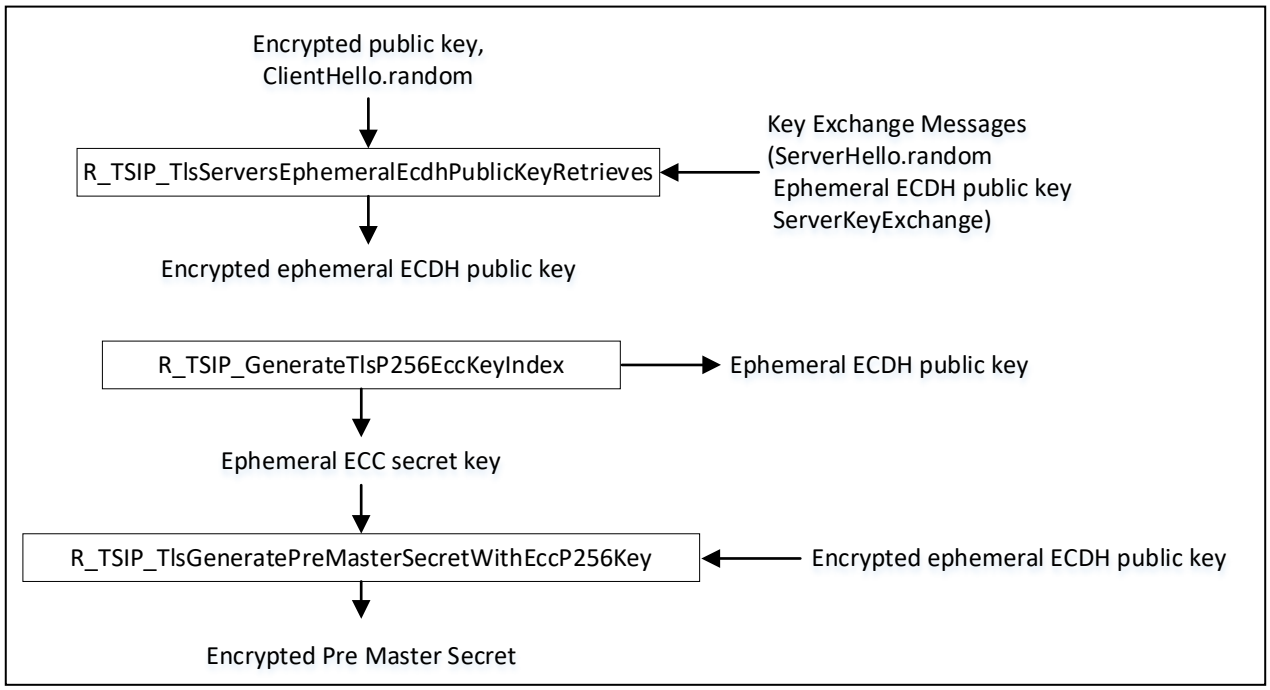

図 3-10 認証/鍵交換メッセージの受信

4. 認証/鍵交換メッセージの送信（Send Client Authentication and Key Exchange Messages）

4.1 CertificateRequest を受信した場合、クライアント証明書を(Client) Certificate としてサーバに送信します。

4.2-a 鍵交換方式が ECDHE の場合、ECDH 公開鍵を使用して ClientKeyExchange を生成します。

4.2-a.1 前述の 3.3.1 で生成した ECDH 公開鍵を、ClientKeyExchange の ClientECDiffieHellmanPublic フィールドとして使用します。

4.2-b 鍵交換方式が RSA の場合、Pre Master Secret を生成して ClientKeyExchange を生成します。

4.2-b.1 R_TSIP_TlsGeneratePreMasterSecret()を使用して、暗号化した ephemeral Pre Master Secret を生成します。

4.2-b.2 暗号化した ephemeral PreMasterSecret と(Server) Certificate の公開鍵を入力として、R_TSIP_TlsEncryptPreMasterSecretWithRsa2048PublicKey()を使用して、ClientKeyExchange に使用するために暗号化した ephemeral Pre Master Secret を生成します。

4.3 必要な場合、R_TSIP_EcdsaP256SignatureGenerate()、R_TSIP_RsassaPkcs2048SignatureGenerate()、または R_TSIP_RsassaPss2048SignatureGenerate()を使用して、CertificateVerify を生成します。

4.4 暗号化した ephemeral Pre Master Secret を入力として、R_TSIP_TlsGenerateMasterSecret()を使用して、暗号化した ephemeral Master Secret を生成します。

4.5 Extended Master Secret を生成する場合は、暗号化した ephemeral Pre Master Secret を入力として、R_TSIP_TlsGenerateExtendedMasterSecret()を使用して、暗号化した ephemeral Extended Master Secret を生成します。

4.6 暗号化した ephemeral Master Secret または Extended Master Secret と各ハンドシェイクメッセージを入力として、R_TSIP_TlsGenerateSessionKey()を使用して、TLS 通信の各種鍵の Wrapped Key を生成します。

4.7 (Client) Finished を生成します。

4.7.1 暗号化した ephemeral Master Secret または ephemeral Extended Master Secret とハンドシェイクメッセージのハッシュ値を入力として、R_TSIP_TlsGenerateVerifyData()を使用して、(Client) Finished に使用するための(Client) VerifyData を生成します。

4.7.2 暗号スイート AES CBC モードを使用する場合は、4.6 で生成したクライアント→サーバ通信時の AES 鍵の Wrapped Key と MAC 鍵の Wrapped Key、及び 4.7.1 で生成した(Client) VerifyData を入力として、AES CBC 暗号化 API と HMAC 生成 API を使用して、(Client) VerifyData を暗号化し(Client) Finished を生成します。

4.7.3 暗号スイート AES GCM モードを使用する場合は、4.6 で生成したクライアント→サーバ通信時の AES 鍵の Wrapped Key と 4.7.1 で生成した(Client) VerifyData を入力として、AES GCM 暗号化 API を使用して、(Client) VerifyData を暗号化し(Client) Finished を生成します。

4.8 各認証/鍵交換メッセージを送信します。

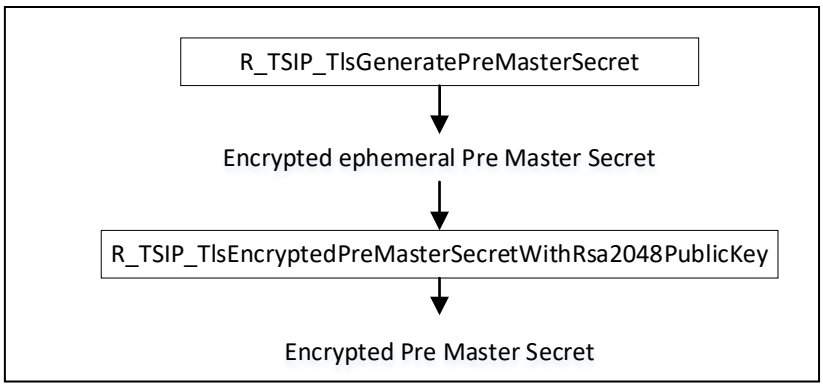

図 3-11 認証/鍵交換メッセージの送信（RSA の場合の Pre Master Secret 生成）

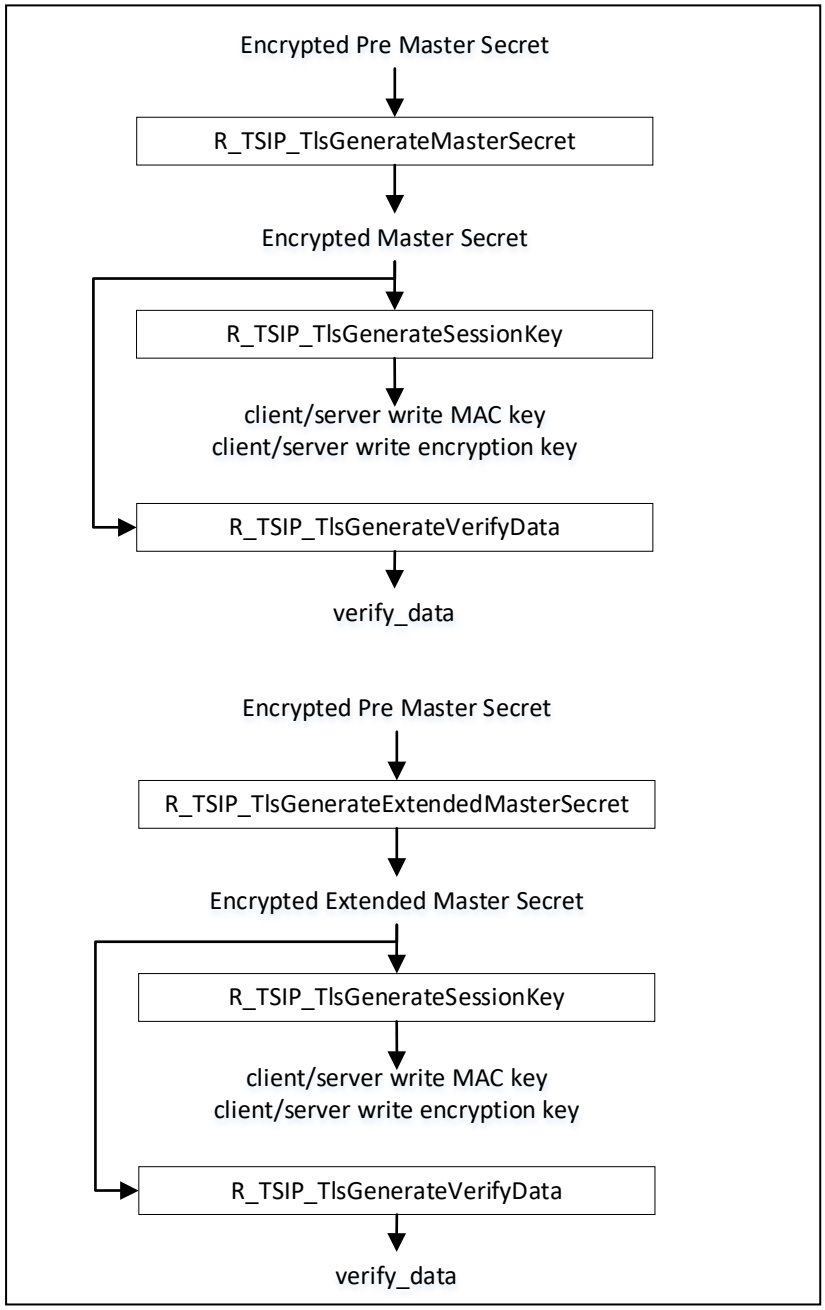

図 3-12 認証/鍵交換メッセージの送信（verify_data の生成）

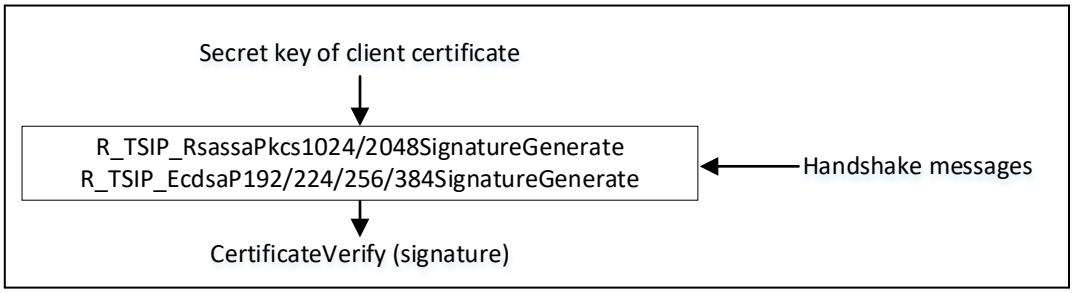

図 3-13 認証/鍵交換メッセージの送信（CertificateVerify の生成）

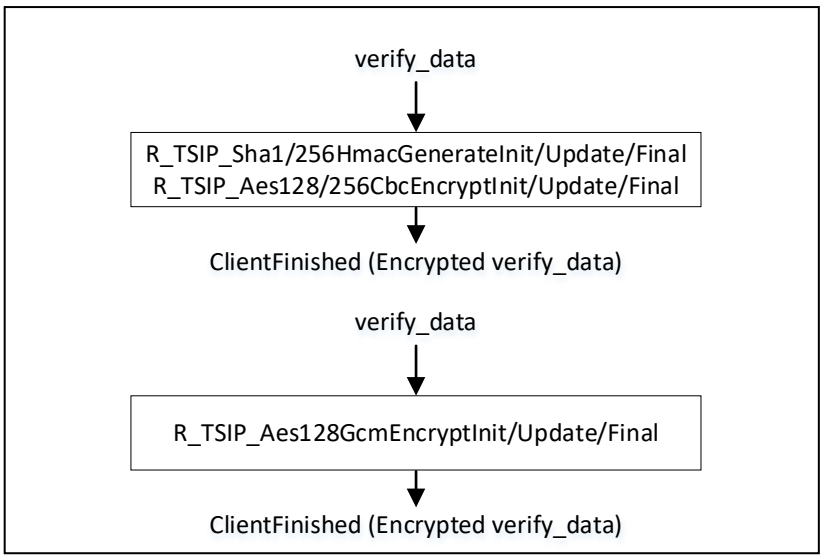

図 3-14 認証/鍵交換メッセージの送信（ClientFinished の生成）

5. (Server) Finished の受信（Receive Finished）

5.1 暗号化した ephemeral Master Secret または Extended Master Secret とハンドシェイクメッセージのハッシュ値を入力として、R_TSIP_TlsGenerateVerifyData()を使用して、(Server) VerifyData を生成します。

5.2 (Server) Finished を検証します。

5.2.1-a 暗号スイート AES CBC モードを使用する場合は、4.6 で生成したサーバ→クライアント通信時の AES 鍵の Wrapped Key と MAC 鍵の Wrapped Key、及び(Server) Finished を入力として、CBC 復号 API と HMAC 検証 API を使用して、VerifyData を復号します。

5.2.1-b 暗号スイート AES GCM モードを使用する場合は、4.6 で生成したサーバ→クライアント通信時の AES 鍵の Wrapped Key と(Server) Finished を入力として、GCM 復号 API を使用して、VerifyData を復号します。

5.2.2 上記の 5.1 で生成した(Server) VerifyData と、5.2.1-a または 5.2.1-b で復号した VerifyData が一致することを確認します。

6. アプリケーションデータの送受信（Send/Receive Application Data）

6.1-a 暗号スイート AES CBC モードを使用する場合は、4.6 で生成した AES 鍵の Wrapped Key と MAC 鍵の Wrapped Key を入力として、AES CBC 暗号化/復号 API と HMAC 生成/検証 API を使用して、アプリケーションデータを暗号化/復号します。

6.1-b 暗号スイート AES GCM モードを使用する場合は、4.6 で生成した AES 鍵の Wrapped Key を入力として、AES GCM 暗号化/復号 API を使用して、アプリケーションデータを暗号化/復号します。

3.12.2 TLS1.2 サーバ機能

No.	API	説明
1	R_TSIP_TlsRegisterCaCertificationPublicKeyIndex	TLS 連携機能で共通に使用
2	R_TSIP_GenerateTlsSVRsaPublicKeyIndex R_TSIP_UpdateTlsSVRsaPublicKeyIndex R_TSIP_TlsSVRootCertificateVerification R_TSIP_TlsSVCertificateVerification R_TSIP_TlsSVCertificateVerificationExtension	TLS1.2/TLS1.3 サーバ機能で共通に使用
3	R_TSIP_TlsSVGenerateServerRandom R_TSIP_TlsSVDecryptPreMasterSecretWithRsa2048PrivateKey R_TSIP_TlsSVGenerateMasterSecret R_TSIP_TlsSVGenerateSessionKey R_TSIP_TlsSVGenerateVerifyData R_TSIP_GenerateTlsSVP256EccKeyIndex R_TSIP_TlsSVGeneratePreMasterSecretWithEccP256Key R_TSIP_TlsSVGenerateExtendedMasterSecret R_TSIP_TlsSVCertificateVerifyVerification	TLS1.2 サーバ機能で使用
4	R_TSIP_Aes128CbcEncryptInit/Update/Final R_TSIP_Aes256CbcEncryptInit/Update/Final R_TSIP_Aes128CbcDecryptInit/Update/Final R_TSIP_Aes256CbcDecryptInit/Update/Final R_TSIP_Aes128GcmEncryptInit/Update/Final R_TSIP_Aes256GcmEncryptInit/Update/Final R_TSIP_Aes128GcmDecryptInit/Update/Final R_TSIP_Aes256GcmDecryptInit/Update/Final R_TSIP_Sha1HmacGenerateInit/Update/Final R_TSIP_Sha1HmacVerifyInit/Update/Final R_TSIP_Sha256HmacGenerateInit/Update/Final R_TSIP_Sha256HmacVerifyInit/Update/Final	汎用 API を TLS1.2 連携機能で使用

TLS1.2 サーバ機能を使用した TLS 通信の実施方法の概要を図 **3-15** にします。

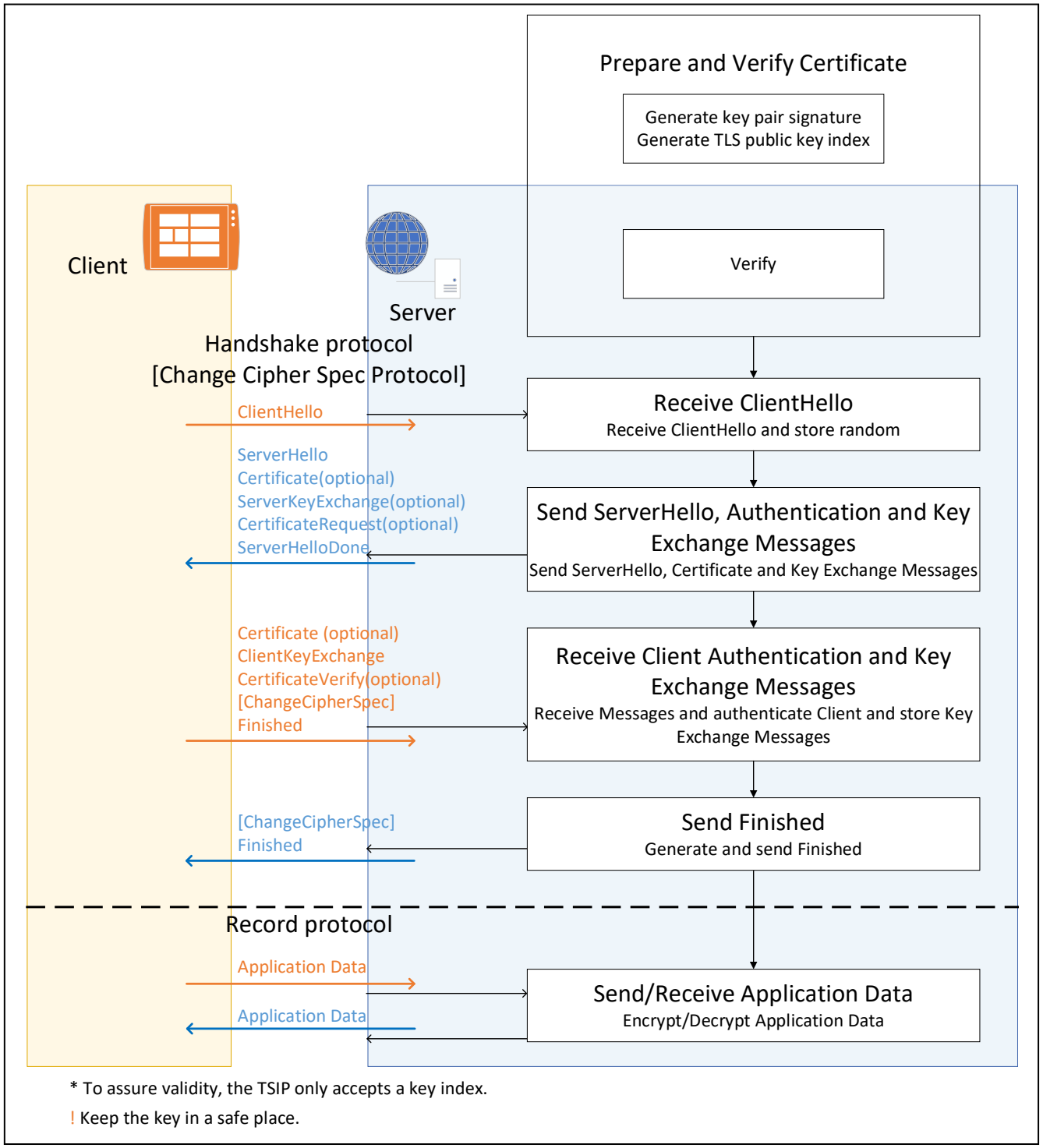

図 3-15 TLS1.2 サーバ機能の実施方法

1. 事前準備と証明書の検証（Prepare and verify certificate）

1.1 事前に以下のものを準備します。

- 接続するサーバのルート CA 証明書
- RSA2048bit の鍵ペア
- サーバ証明書

1.2 用意した RSA 鍵ペアの秘密鍵を使って、用意したルート CA 証明書の署名値を生成(署名アルゴルズムは RSA2048 PSS with SHA256)します。また用意した RSA 鍵ペアの公開鍵は、UFPK でラップし、Encrypted Key にします。これらの作業は安全なサイトで行ってください。

ルート CA 証明書を複数登録したい場合は、束としてルート CA 証明書をまとめたデータに対して、署名を付けてください。ルート CA 証明書に署名することで、不正な証明書チェーンを使用した TLS セッションの実施を防ぐことができます。

1.3 RSA 鍵ペアの公開鍵の Encrypted Key を入力として、R_TSIP_TlsSVRsaPublicKeyIndex を使用して、公開鍵の Wrapped Key を生成します。詳細な手順については、3.7.1 鍵の注入と更新を参照してください。

1.4 R_TSIP_TlsRegisterCaCertificationPublicKeyIndex()を使用して、公開鍵の Wrapped Key を TSIP ドライバに登録します。

1.5 R_TSIP_TlsSVRootCertificateVerification()を使用して、ルート CA 証明書の束を検証し、検証に成功したらルート CA 証明書から公開鍵を抽出し、暗号化して出力します。

1.6 暗号化されたルート CA 証明書の公開鍵、およびサーバ証明書を入力として、R_TSIP_TlsSVCertificateVerification()または R_TSIP_TlsSVCertificateVerificationExtension()を使用してサーバ証明書を検証し、検証に成功したらサーバ証明書から公開鍵を抽出し、暗号化して出力します。

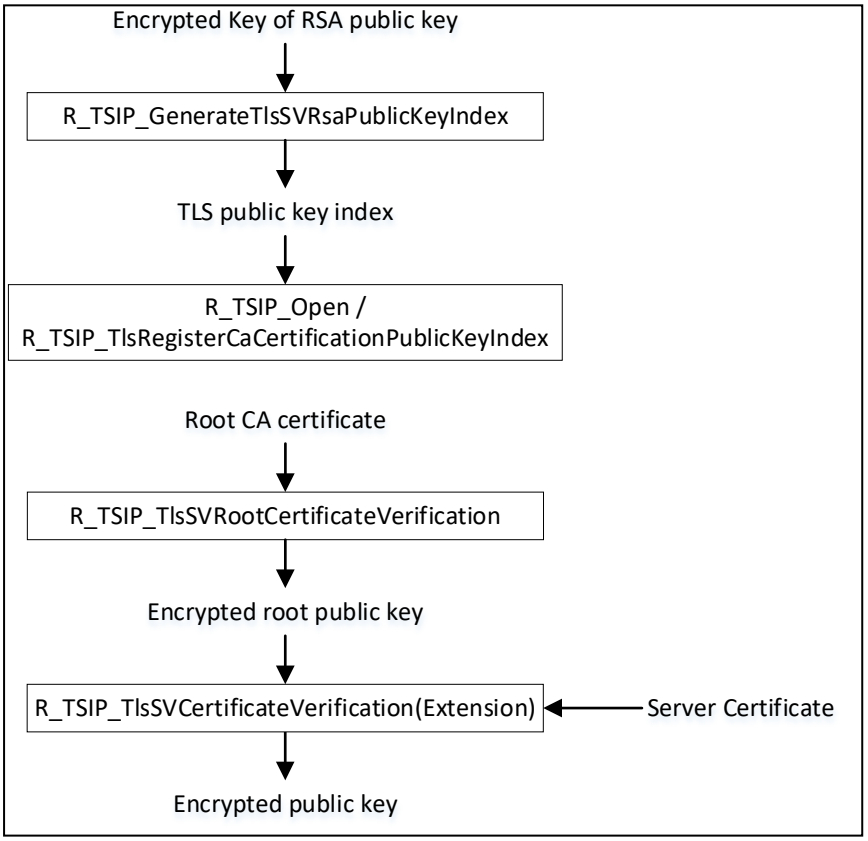

図 3-16 事前準備と証明書の検証

2. ClientHello の受信（Receive ClientHello）

2.1 ClientHello と ClientHello の random フィールドを受信します。

3. ServerHello と認証/鍵交換メッセージの送信（Send ServerHello, Authentication and Key Exchange Messages）

3.1 R_TSIP_TlsSVGenerateServerRandom()を使用して、ServerHello の random フィールドを生成します。

3.2-a 鍵交換方式が ECDHE の場合、ServerKeyExchange を生成します。

3.2-a.1 R_TSIP_GenerateTlsSVP256EccKeyIndex()を使用して、ECC 鍵ペアを生成します。ECC 鍵ペアの ECDH 公開鍵は、ServerKeyExchange の params の public フィールドとして使用します。

3.2-a.2 (Server) Certificate の秘密鍵の Wrapped Key とハンドシェイクメッセージのハッシュ値を入力として、R_TSIP_EcdsaP256SignatureGenerate()を使用して、署名を生成します、生成した署名は、ServerKeyExchange の signed_params フィールドとして使用します。

3.2-b 鍵交換方式が RSA の場合、ここで実施する処理はありません。

3.3 ServerHello と認証/鍵交換メッセージを送信します。

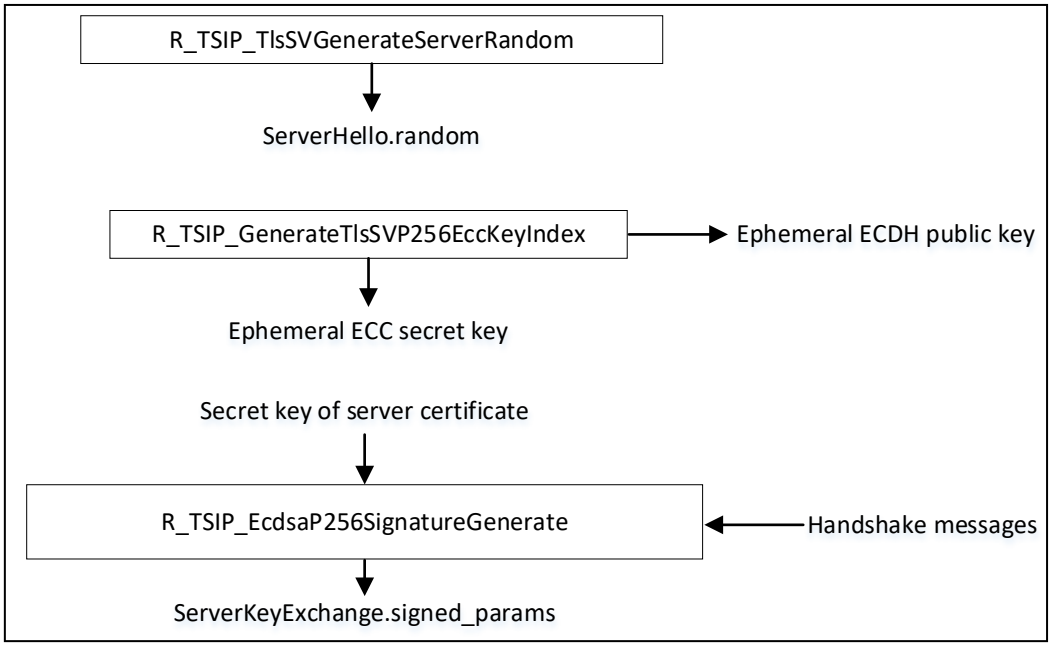

図 3-17 認証/鍵交換メッセージの送信

4. 認証/鍵交換メッセージの受信（Receive Client Authentication and Key Exchange Messages）

4.1 (Client) Certificate を受信した場合、R_TSIP_TlsSVCertificateVerification()または R_TSIP_TlsSVCertificateVerificationExtension()を使用して、暗号化したクライアント証明書の公開鍵を生成します。

4.2-a 鍵交換方式が ECDHE の場合、ClientKeyExchange から ECDH 公開鍵を取り出して Pre Master Secret を生成します。

4.2-a.1 ClientKeyExchange の ECDH 公開鍵と 3.2.1 で生成した ECC 秘密鍵の Wrapped Key を入力として、R_TSIP_TlsSVGeneratePreMasterSecretWithEccP256Key()を使用して、暗号化した ephemeral Pre Master Secret を生成します。

4.2-b 鍵交換方式が RSA の場合、ClientKeyExchange から Pre Master Secret を取り出します。

4.2b.1 ClientKeyExchange の暗号化された Pre Master Secret と(Server) Certificate の RSA 2048bit 秘密鍵を入力として、R_TSIP_TlsSVDecryptPreMasterSecretWithRsa2048PrvateKey()を使用して、暗号化した ephemeral Pre Master Secret を生成します。

4.3 暗号化した ephemeral Pre Master Secret を入力として、R_TSIP_TlsSVGenerateMasterSecret()を使用して、暗号化した ephemeral Master Secret を生成します。

4.4 Extended Master Secret を生成する場合は、暗号化した ephemeral Pre Master Secret を入力として、R_TSIP_TlsSVGenerateExtendedMasterSecret()を使用して、暗号化した ephemeral Extended Master Secret を生成します。

4.5 CertificateVerify を受信した場合、R_TSIP_TlsSVCertificateVerifyVerification()を使用して、CertificateVerify を検証します。

4.6 暗号化した ephemeral Master Secret または ephemeral Extended Master Secret と各ハンドシェイクメッセージを入力として、R_TSIP_TlsSVGenerateSessionKey()を使用して、TLS 通信の各種鍵の Wrapped Key を生成します。

4.7 (Client) Finished を検証します。

4.7.1 暗号化した ephemeral Master Secret または ephemeral Extended Master Secret とハンドシェイクメッセージのハッシュ値を入力として、R_TSIP_TlsSVGenerateVerifyData()を使用して、(Client) Finished と比較するための(Client) VerifyData を生成します。

4.7.2-a 暗号スイート AES CBC モードを使用する場合は、4.6 で生成したクライアント→サーバ通信時の AES 鍵の Wrapped Key と MAC 鍵の Wrapped Key、及び(Client) Finished を入力として、CBC 復号 API と HMAC 検証 API を使用して、VerifyData を復号します。

4.7.2-b 暗号スイート AES GCM モードを使用する場合は、4.6 で生成したクライアント→サーバ通信時の AES 鍵の Wrapped Key 生成情報と(Client) Finished を入力として、GCM 復号 API を使用して、VerifyData を復号します。

4.7.3 上記の 4.7.1 で生成した(Client) VerifyData と、4.7.2-a または 4.7.2-b で復号した VerifyData が一致することを確認します。

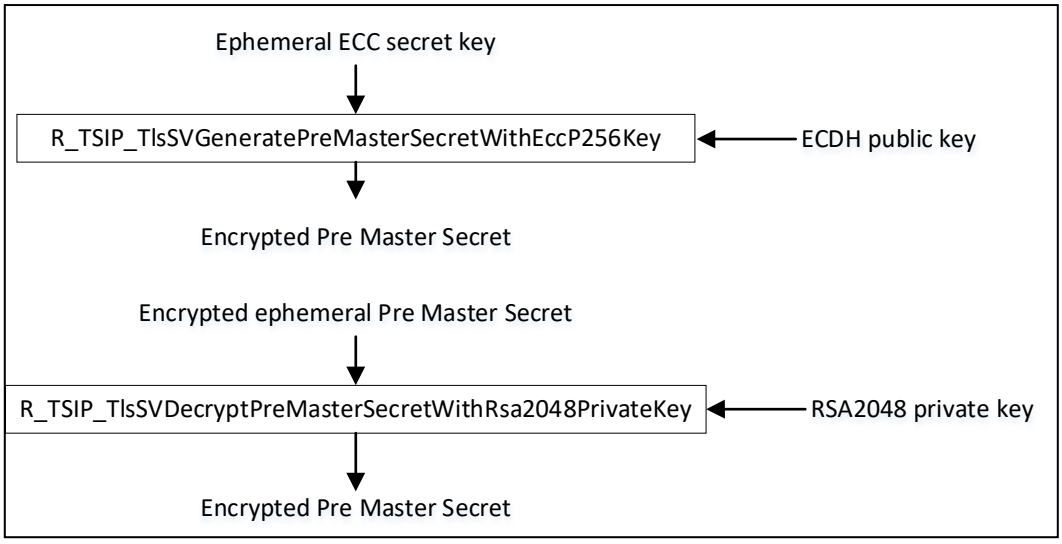

図 3-18 認証/鍵交換メッセージの受信（Pre Master Secret の生成）

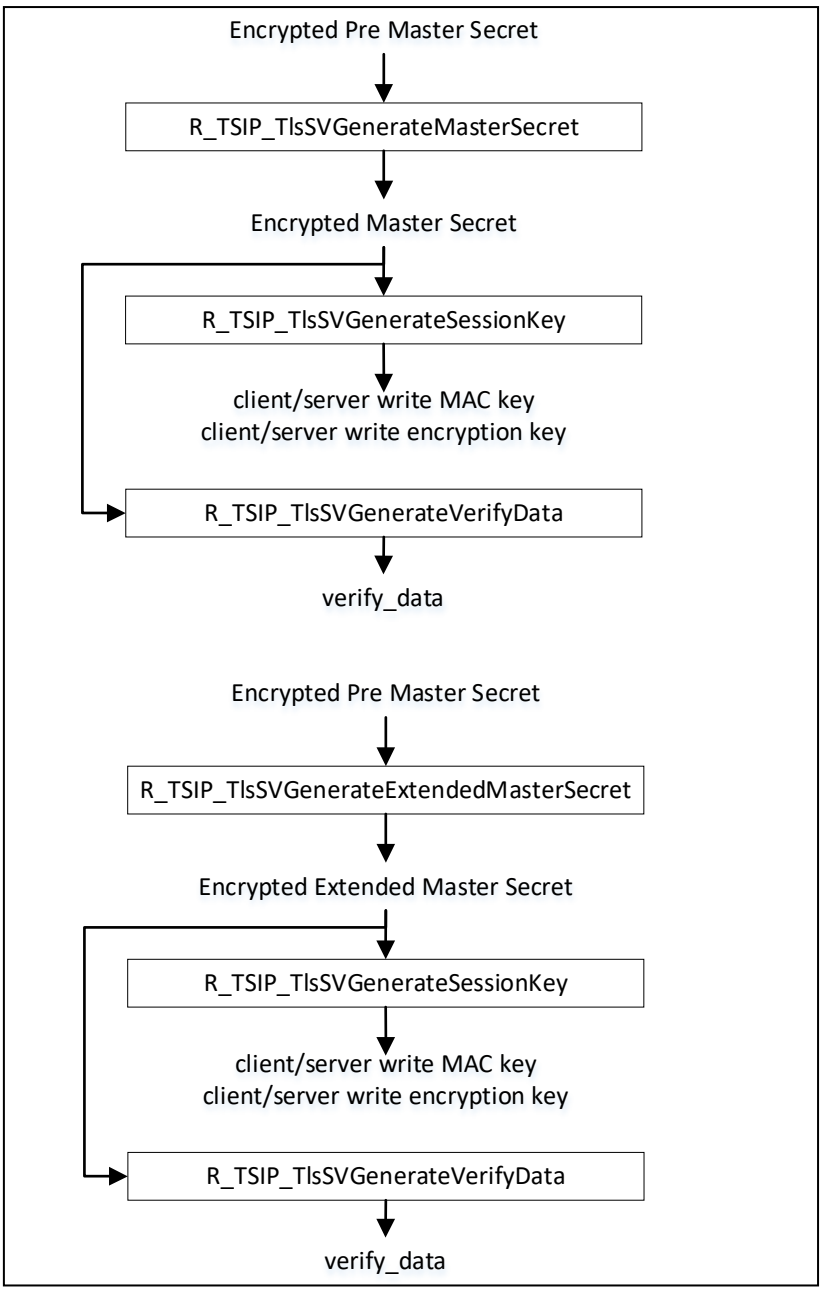

図 3-19 認証/鍵交換メッセージの受信（verify_data の生成）

5. (Server) Finished の送信（Send Finished）

5.1 暗号化した ephemeral Master Secret または ephemeral Extended Master Secret とハンドシェイクメッセージのハッシュ値を入力として、R_TSIP_TlsSVGenerateVerifyData()を使用して、(Server) VerifyData を生成します。

5.2-a 暗号スイート AES CBC モードを使用する場合は、4.6 で生成したサーバ→クライアント通信時の AES 鍵の Wrapped Key と MAC 鍵の Wrapped Key、及び 5.1 で生成した(Server) VerifyData を入力として、AES CBC 暗号化 API と HMAC 生成 API を使用して、(Server) VerifyData を暗号化し(Server) Finished を生成します。

5.2-b 暗号スイート AES GCM モードを使用する場合は、4.6 で生成したサーバ→クライアント通信時の AES 鍵の Wrapped Key と 5.1 で生成した(Server) VerifyData を入力として、AES GCM 暗号化 API を使用して、(Server) VerifyData を暗号化し(Server) Finished を生成します。

5.3 (Server) Finished を送信します。

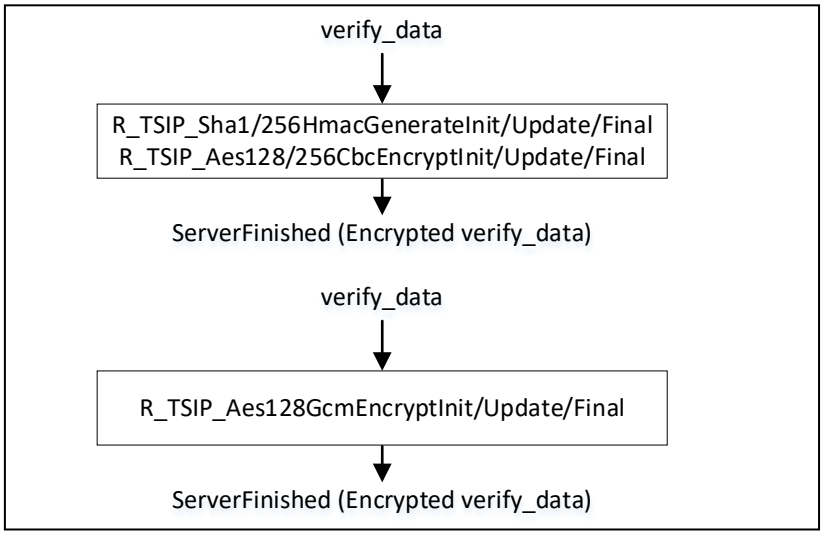

図 3-20 (Server) Finished の送信

6. アプリケーションデータの送受信（Send/Receive Application Data）

6.1-a 暗号スイート AES CBC モードを使用する場合は、4.6 で生成した AES 鍵の Wrapped Key と MAC 鍵の Wrapped Key を入力として、AES CBC 暗号化/復号 API と HMAC 生成/検証 API を使用して、アプリケーションデータを暗号化/復号します。

6.1-b 暗号スイート AES GCM モードを使用する場合は、4.6 で生成した AES 鍵の Wrapped Key を入力として、AES GCM 暗号化/復号 API を使用して、アプリケーションデータを暗号化/復号します。

3.13 TLS 連携機能(TLS1.3)

TLS 連携機能(TLS1.3)の API は、RFC8446 に準拠しています。

TLS 連携機能の内、TLS1.3 連携機能として、TLS1.3 クライアント機能および TLS1.3 サーバ機能をサポートしています。それぞれの機能の、Full Handshake、Resumption、0-RTT 使用時の TSIP ドライバの使用方法について示します。

3.13.1 TLS1.3 クライアント機能

No.	API	説明
1	R_TSIP_GenerateTlsRsaPublicKeyIndex R_TSIP_UpdateTlsRsaPublicKeyIndex R_TSIP_TlsRootCertificateVerification R_TSIP_TlsCertificateVerification R_TSIP_TlsCertificateVerificationExtension R_TSIP_Tls13EncryptInit R_TSIP_Tls13EncryptUpdate R_TSIP_Tls13EncryptFinal R_TSIP_Tls13DecryptInit R_TSIP_Tls13DecryptUpdate R_TSIP_Tls13DecryptFinal R_TSIP_Tls13CertificateVerifyGenerate R_TSIP_Tls13CertificateVerifyVerification	各機能で共通に使用
2	R_TSIP_GenerateTls13P256EccKeyIndex R_TSIP_Tls13GenerateEcdheSharedSecret R_TSIP_Tls13GenerateHandshakeSecret R_TSIP_Tls13GenerateServerHandshakeTrafficKey R_TSIP_Tls13GenerateClientHandshakeTrafficKey R_TSIP_Tls13ServerHandshakeVerification R_TSIP_Tls13GenerateMasterSecret R_TSIP_Tls13GenerateApplicationTrafficKey R_TSIP_Tls13UpdateApplicationTrafficKey	主に Full Handshake で使用
3	R_TSIP_Tls13GenerateResumptionMasterSecret R_TSIP_Tls13GeneratePreSharedKey R_TSIP_Tls13GeneratePskBinderKey R_TSIP_Tls13GenerateResumptionHandshakeSecret	Resumption で使用
4	R_TSIP_Tls13Generate0RttApplicationWriteKey	0-RTT で使用

3.13.1.1 Full Handshake

TLS1.3 クライアント機能を使用した Full Handshake 実施方法の概要を図 3-21 に示します。

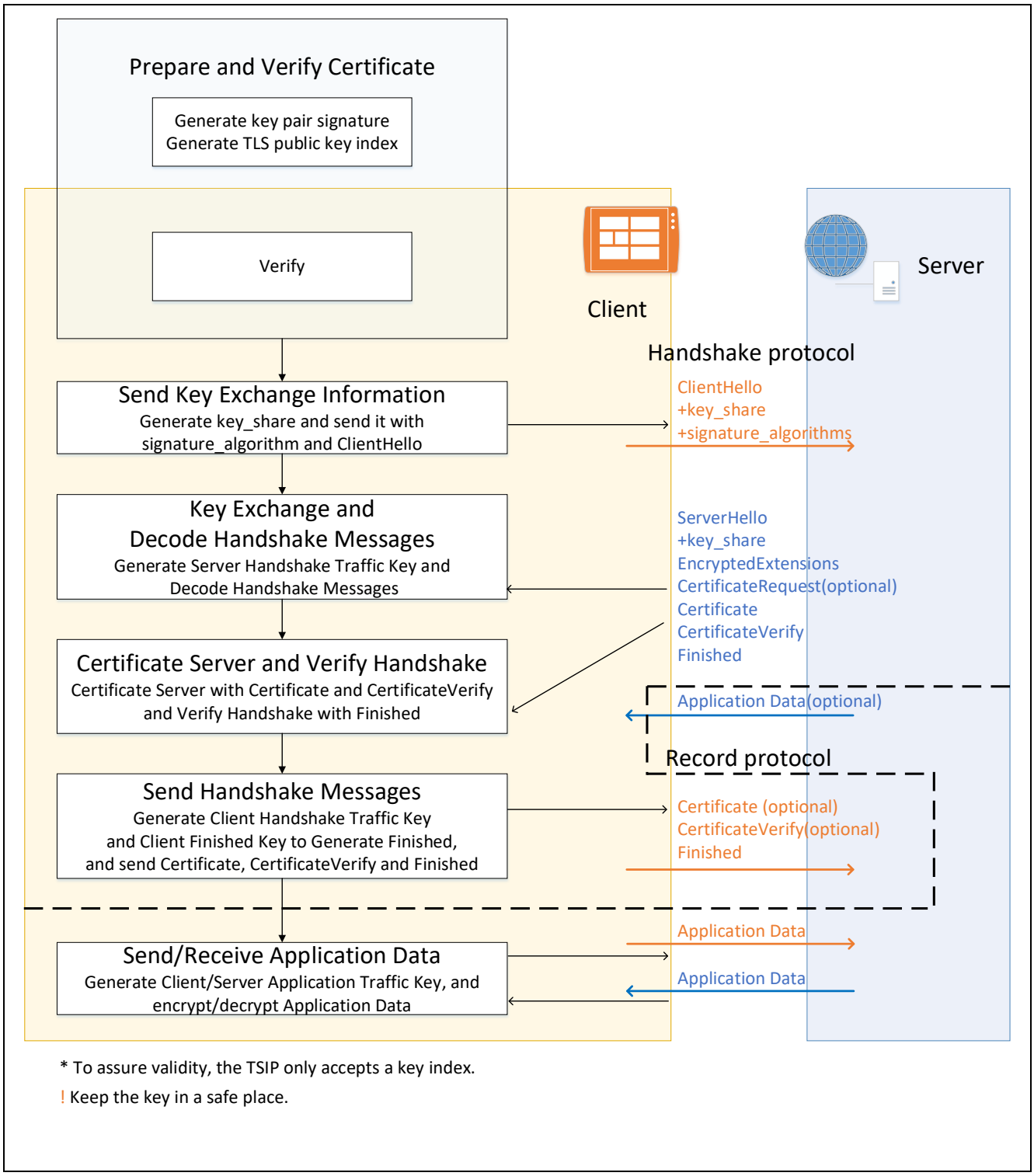

図 3-21 TLS1.3 クライアント機能 Full Handshake の実施方法

1. 事前準備と証明書の検証（Prepare and verify certificate）

1.1 事前に以下のものを準備します。

- 接続するサーバのルート CA 証明書
- RSA2048bit の鍵ペア
- クライアント証明書

1.2 用意した RSA 鍵ペアの秘密鍵を使って、用意したルート CA 証明書の署名値を生成(署名アルゴルズムは RSA2048 PSS with SHA256)します。また用意した RSA 鍵ペアの公開鍵は、UFPK でラップし、Encrypted Key にします。これらの作業は安全なサイトで行ってください。

接続するサーバが複数あり、ルート CA 証明書を複数登録したい場合は、束としてルート CA 証明書をまとめたデータに対して、署名を付けてください。ルート CA 証明書に署名することで、意図しないサーバへの接続を防ぐことができます。

1.3 RSA 鍵ペアの公開鍵の Encrypted Key を入力として、R_TSIP_TlsRsaPublicKeyIndex()を使用して、公開鍵の Wrapped Key を生成します。詳細な手順については、3.7.1 鍵の注入と更新を参照してください。

1.4 R_TSIP_TlsRegisterCaCertificationPublicKeyIndex()もしくは R_TSIP_Open()を使用して、公開鍵の Wrapped Key を TSIP ドライバへ登録します。R_TSIP_Open()を使用して登録する場合は、R_TSIP_Close()を呼んだ後、R_TSIP_Open()を呼んでください。

1.5 R_TSIP_TlsRootCertificateVerification()を使用して、ルート CA 証明書の束を検証し、暗号化したルート CA 証明書の公開鍵を生成します。

1.6 暗号化したルート CA 証明書の公開鍵、およびクライアント証明書の公開鍵を入力として、R_TSIP_TlsCertificateVerification()または R_TSIP_TlsCertificateVerificationExtension()を使用して、暗号化したクライアント証明書の公開鍵を生成します。

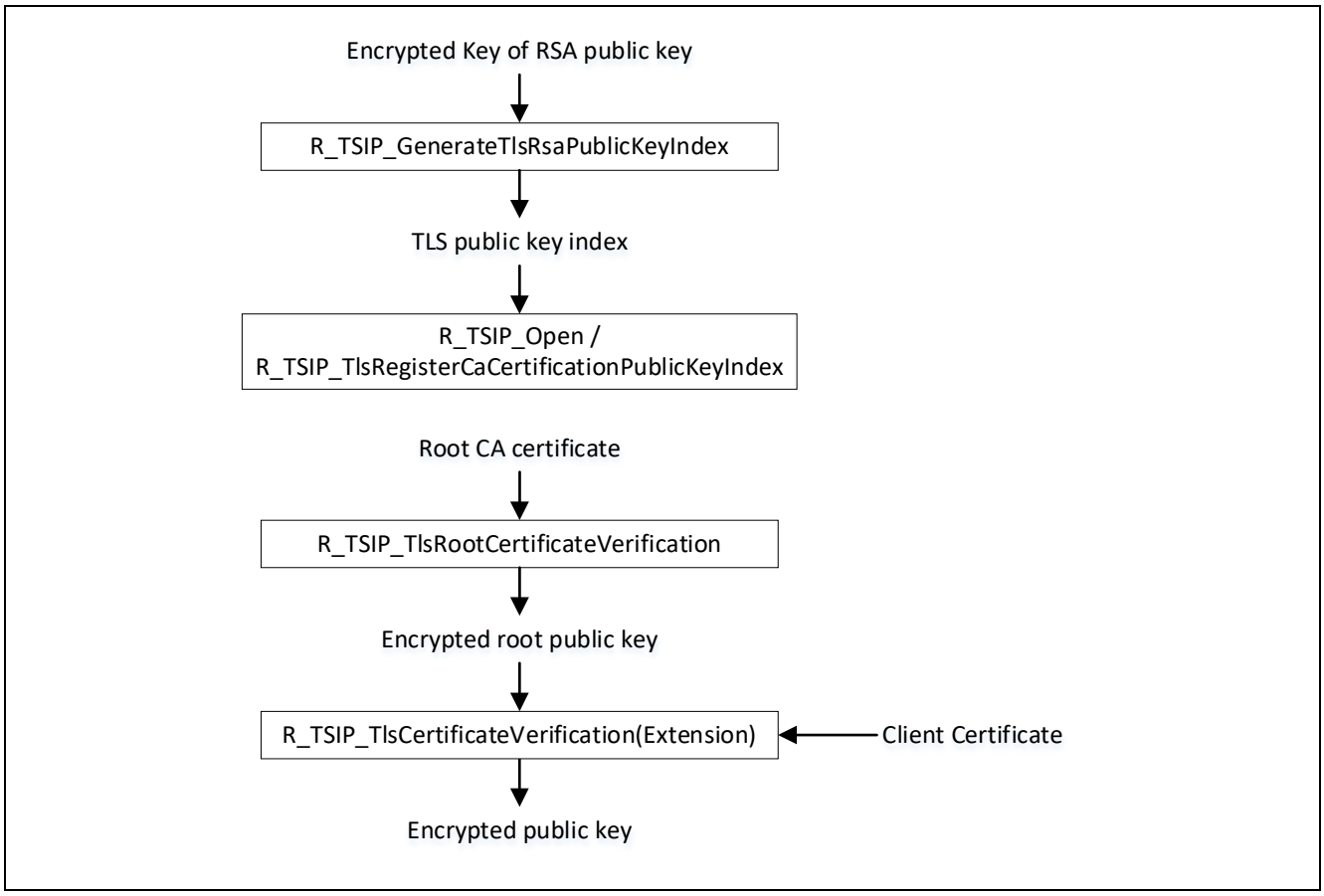

図 3-22 事前準備と証明書の検証

2. 鍵交換用データの送信（Send Key Exchange Information）
   2.1 R_TSIP_GenerateTls13P256KeyIndex()を使用して Ephemeral ECDH 公開鍵を生成します。
   2.2 ClientHello 送信時に、signature_algorithm フィールドと共に、key_share フィールドとして ECDH 公開鍵をサーバに送信します。

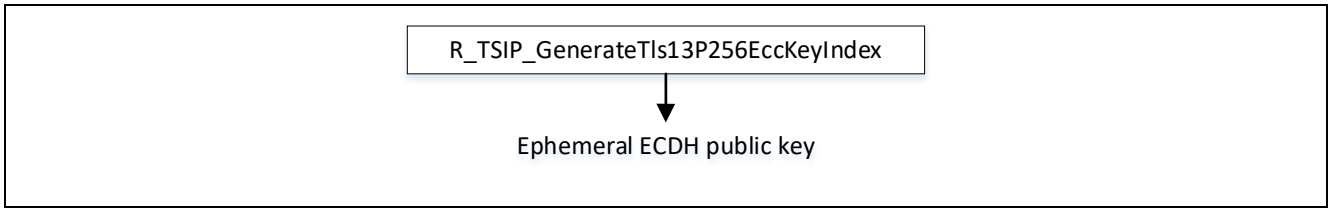

図 3-23 鍵交換用データの送信

3. 鍵交換とハンドシェイクメッセージの復号（Key Exchange and Decode Handshake Messages）
   3.1 サーバから受信した公開鍵を入力として、R_TSIP_Tls13GenerateEcdheSharedSecret()を使用して Shared Secret の Wrapped Key を生成します。
   3.2 Shared Secret の Wrapped Key を入力として、R_TSIP_Tls13GenerateHandshakeSecret()を使用して Handshake Secret の Wrapped Key を生成します。
   3.3 Handshake Secret の Wrapped Key を入力として、R_TSIP_Tls13GenerateServerHandshakeTrafficKey()を使用して、Handshake プロトコルで使用する Server Write Key と Server Finished Key の Wrapped Key を生成します。
   3.4 Server Write Key の Wrapped Key を入力として、R_TSIP_Tls13DecryptInit/Update/Final()を使用して、サーバから受信した暗号化されているハンドシェイクメッセージを復号します。

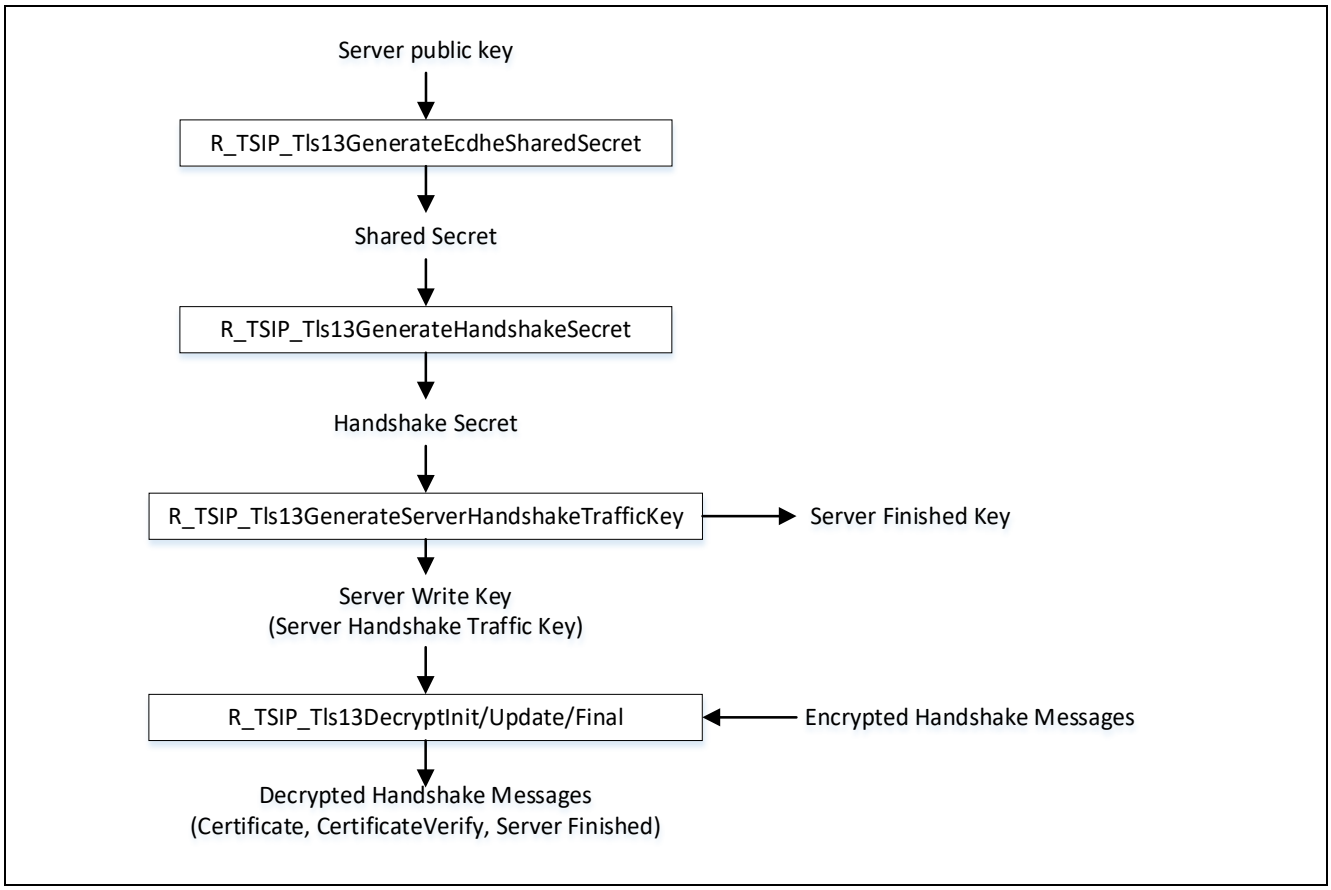

図 3-24 鍵交換とハンドシェイクメッセージの復号

4. サーバの認証とハンドシェイクの検証（Certificate Server and Verify Handshake）

4.1 ハンドシェイクメッセージ内の(Server) Certificate フィールドを入力として、R_TSIP_TlsCertificateVerification()を使用して、証明書の署名を検証します。

4.2 ハンドシェイクッセージ内の(Server) CertificateVerify フィールドを入力として、R_TSIP_Tls13CertificateVerifyVerification()を使用して、(Server) CertificateVerify フィールドを検証します。

4.3 ハンドシェイクメッセージ内の(Server) Finished フィールドと Server Finished Key の Wrapped Key を入力として、R_TSIP_Tls13ServerHandshakeVerification()を使用して、ハンドシェイクを検証します。TSIP においては、ハンドシェイクの検証結果は verify_data_index として出力されます。

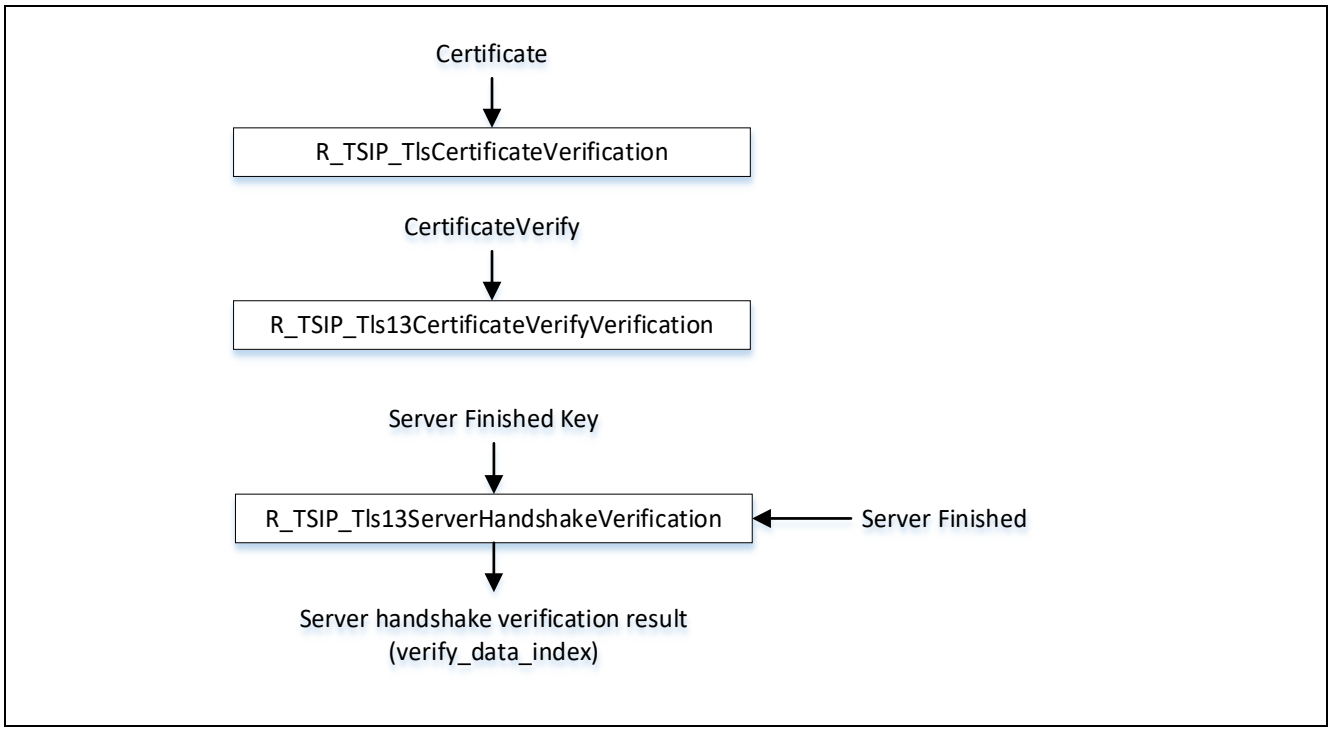

図 3-25 サーバの認証とハンドシェイクの検証

5. ハンドシェイクメッセージの送信（Send Handshake Messages）

5.1 サーバから CertificateRequest を受信している場合には、(Client) Certificate フィールドと(Client) CertificateVerify フィールドを作成します。(Client) CertificateVerify フィールドの署名を生成する際には、証明書の秘密鍵の Wrapped Key を入力として、R_TSIP_Tls13CertificateVerifyGenerate()を使用して生成します。

5.2 Handshake Secret の Wrapped Key を入力として、R_TSIP_Tls13GenerateClientHandshakeTrafficKey()を使用して、ハンドシェイクフェーズで使用する Client Write Key と Client Finished Key の Wrapped Key を生成します。

5.3 Client Finished Key の Wrapped Key を入力として、R_TSIP_Sha256HmacGenerateIniti/Update/Final()を使用して、(Client) Finished フィールドを作成します。

5.4 Client Write Key の Wrapped Key を入力として、R_TSIP_Tls13EncryptInit/Update/Final()を使用して、サーバへ送信するハンドシェイクメッセージを暗号化します。

5.5 暗号化したハンドシェイクメッセージを送信します。

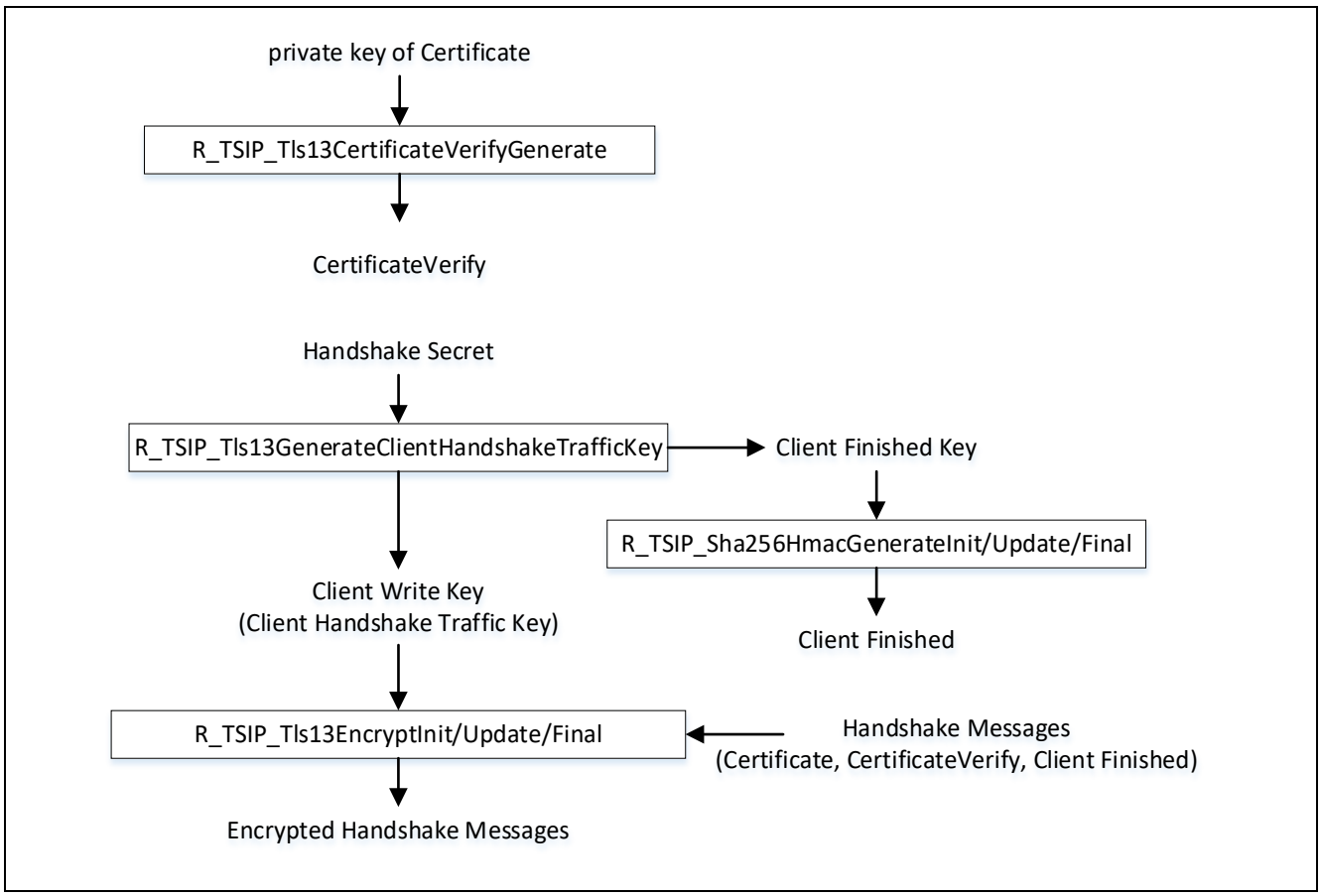

図 3-26 ハンドシェイクメッセージの送信

6. アプリケーションデータの送受信（Send/Receive Application Data）

6.1 Handshake Secret の Wrapped Key と verify_data_index を入力として、R_TSIP_Tls13GenerateMasterSecret()を使用して、Master Secret の Wrapped Key を生成します。
6.2 Master Secret の Wrapped Key を入力として、R_TSIP_Tls13GenerateApplicationTrafficKey()を使用して、Record プロトコルで使用する Server Write Key と Client Write Key の Wrapped Key および Application Secret の Wrapped Key を生成します。
6.3 Server Write Key または Client Write Key を更新する際には、Application Secret の Wrapped Key を入力として、R_TSIP_Tls13UpdateApplicationTrafficKey()を使用します。
6.4 サーバへの送信データを暗号化する際には、Cleint Write Key の Wrapped Key を入力として R_TSIP_Tls13EncryptInit/Update/Final()を使用します。
6.5 サーバからの受信データを復号する際には、Server Write Key の Wrapped Key を入力として R_TSIP_Tls13DecryptInit/Update/Final()を使用します。

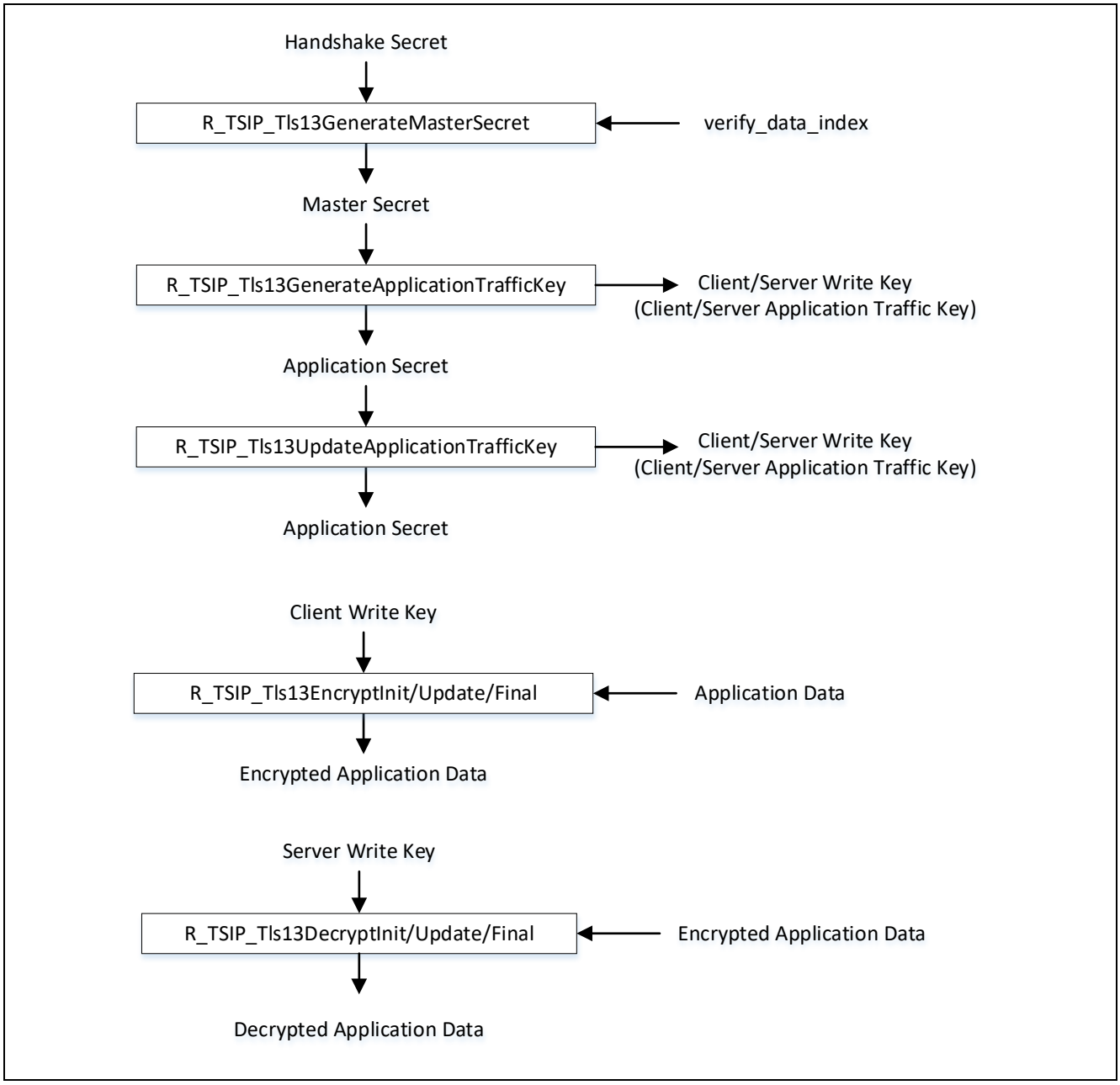

図 3-27 アプリケーションデータの送受信

3.13.1.2 Resumption

TLS1.3 クライアント機能を使用した Resumption 実施方法の概要を図 3-28 に示します。

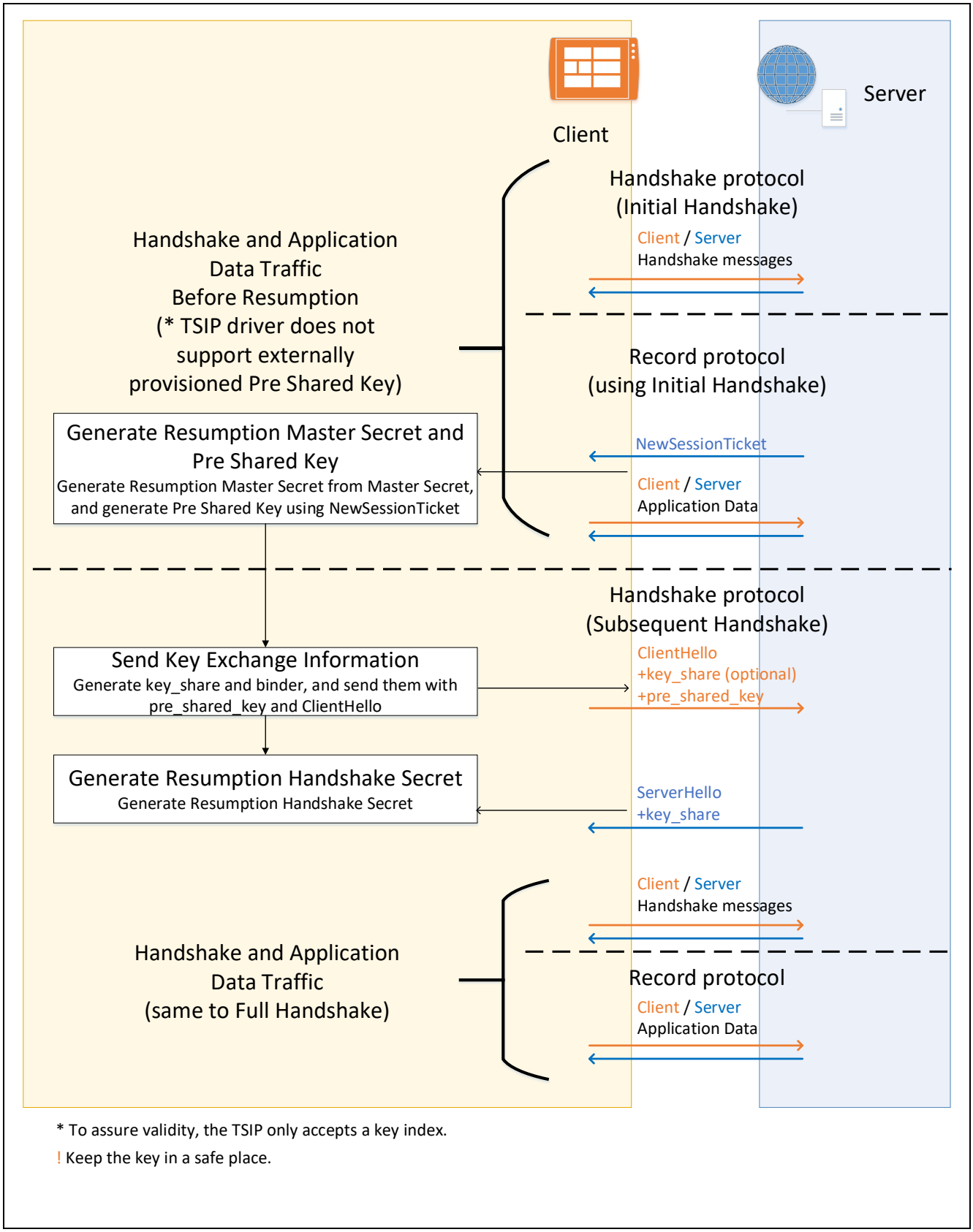

図 3-28 TLS1.3 クライアント機能 Resumption の実施方法

1. Resumption 前のハンドシェイクとアプリケーションデータ通信

1.1 Full Handshake により TLS1.3 通信を確立します。

1.2 アプリケーションデータとして、サーバから New Session Ticket を受信します。

※TSIP ドライバにおいては、Handshake を確立して生成した Pre Shared Key 以外の使用を許可していません。

2. Resumption Master Secret と Pre Shared Key の生成（Generate Resumption Master Secret and Pre Shared Key）

2.1 Full Handshake 時に使用した Master Secret の Wrapped Key を入力として、R_TSIP_Tls13GenerateResumptionMasterSecret()を使用して、Resumption Master Secret の Wrapped Key を生成します。

2.2 Resumption Master Secret の Wrapped Key とサーバから受信した New Session Ticket 内の Ticket Nonce を入力として、R_TSIP_Tls13GeneratePreSharedKey()を使用して、Pre Shared Key の Wrapped Key を生成します。

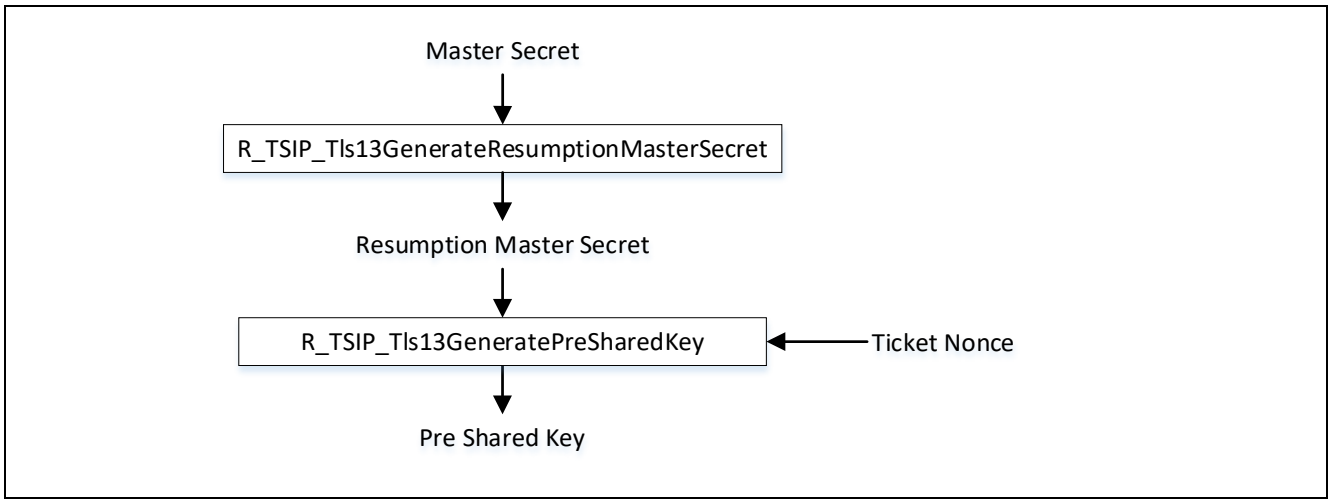

図 3-29 Resumption Master Secret と Pre Shared Key の生成

3. 鍵交換用データの送信（Send Key Exchange Information）
   3.1 Pre Shared Key の Wrapped Key を入力として、R_TSIP_Tls13GeneratePskBinderKey()を使用して、Binder Key の Wrapped Key を生成します。
   3.2 Binder Key の Wrapped Key を入力として、R_TSIP_Sha256HmacGenerateInit/Update/Finish()を使用して、Binder を生成します。
   3.3 3.13.1.1 の 2.1 項と同様に、R_TSIP_GenerateTls13P256KeyIndex()を使用して、Ephemeral ECDH 公開鍵を生成します。
   3.4 ClientHello 送信時に、pre_shared_key フィールドとして Binder を含めた Pre Shared Key の情報、また key_share フィールドとして ECDH 公開鍵も付加してサーバに送信します。

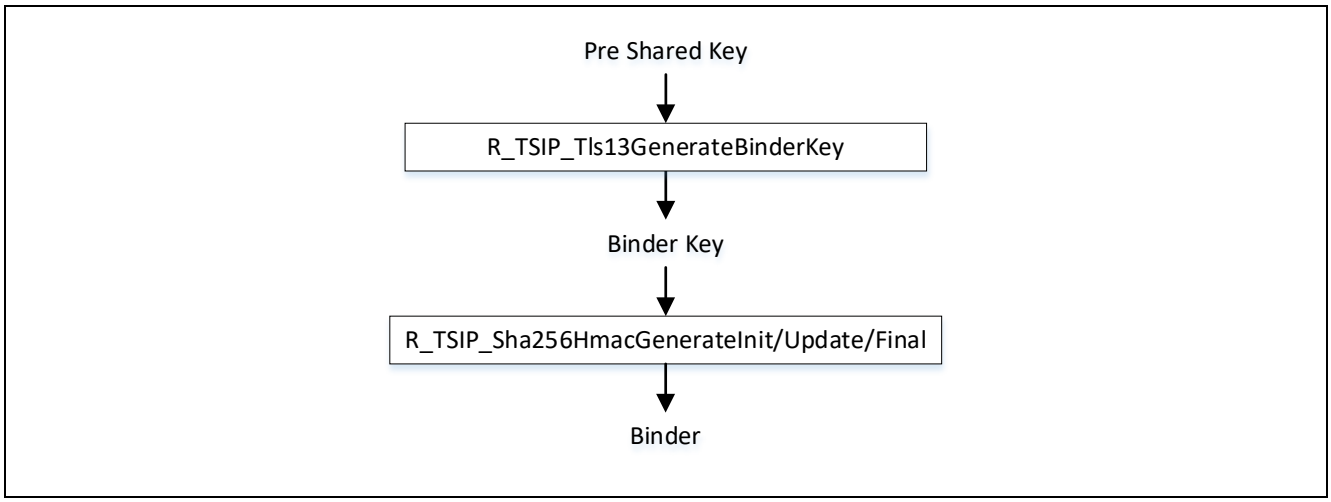

図 3-30 鍵交換用データの送信

4. Resumption 用 Handshake Secret の生成（Generate Resumption Handshake Secret）
   4.13.13.1.1 の 3.1 項と同様に、サーバから受信した公開鍵を入力として、R_TSIP_Tls13GenerateEcdheSharedSecret()を使用して Shared Secret の Wrapped Key を生成します。
   4.2 Pre Shared Key と Shared Secret の Wrapped Key を入力として、R_TSIP_Tls13GenerateResumptionHandshakeSecret()を使用して Resumption 用の Handshake Secret の Wrapped Key を生成します。

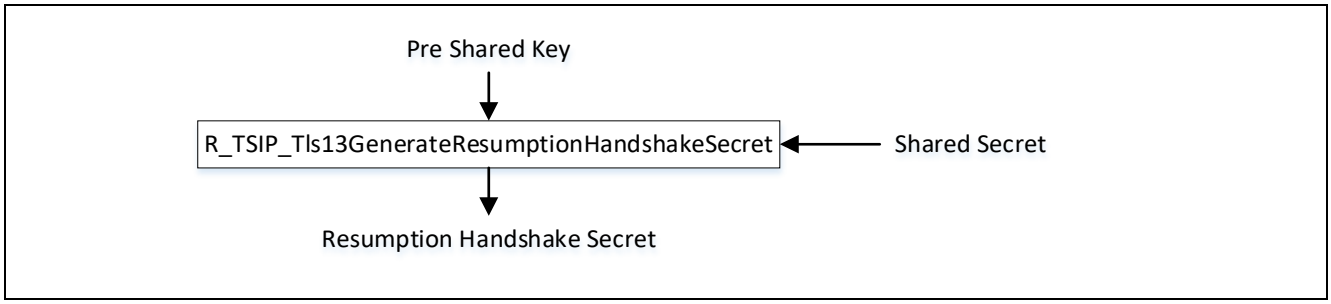

図 3-31 Resumption 用 Handshake Secret の生成

5. ハンドシェイクとアプリケーションデータ通信
   5.1 Resumption 用の Handshake Secret 生成以降は、Full Handshake と同様の通信シーケンスにより TLS1.3 通信を確立します。
   5.2 ハンドシェイクフェーズ終了後に、アプリケーションデータ通信を行います。

3.13.1.3 0-RTT

TLS1.3 クライアント機能を使用した 0-RTT 実施方法の概要を図 3-32 に示します。

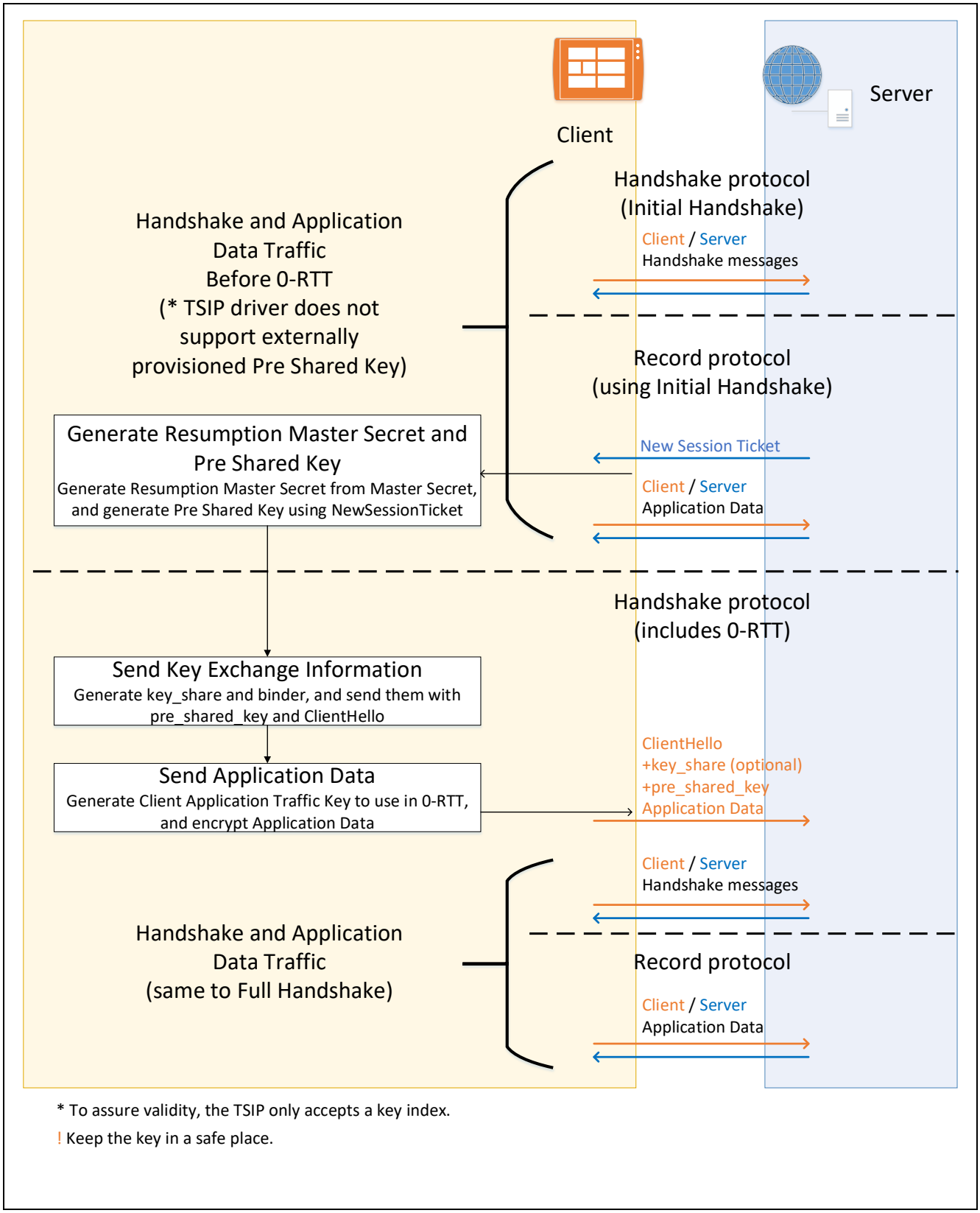

図 3-32 TLS1.3 クライアント機能 0-RTT の実施方法

1. Resumption 前のハンドシェイクとアプリケーションデータ通信

1.1 Full Handshake により TLS1.3 通信を確立します。

1.2 アプリケーションデータとして、サーバから New Session Ticket を受信します。

※TSIP ドライバにおいては、Handshake を確立して生成した Pre Shared Key 以外の使用を許可していません。

2. Resumption Master Secret と Pre Shared Key の生成（Generate Resumption Master Secret and Pre Shared Key）

2.1 以降の処理も含めて 3.13.1.2 の 2. 項と同様に、Full Handshake 時に使用した Master Secret の Wrapped Key を入力として、R_TSIP_Tls13GenerateResumptionMasterSecret()を使用して、Resumption Master Secret の Wrapped Key を生成します。

2.23.13.1.2 の 2.2 項と同様に、Resumption Master Secret の Wrapped Key とサーバから受信した New Session Ticket 内の Ticket Nonce を入力として、R_TSIP_Tls13GeneratePreSharedKey()を使用して、Pre Shared Key の Wrapped Key を生成します。

3. 鍵交換用データの送信（Send Key Exchange Information）

3.1 以降の処理も含めて 3.13.1.2 の 3. 項と同様に、Pre Shared Key の Wrapped Key を入力として、R_TSIP_Tls13GeneratePskBinderKey()を使用して、Binder Key の Wrapped Key を生成します。

3.2 Binder Key の Wrapped Key を入力として、R_TSIP_Sha256HmacGenerateInit/Update/Finish()を使用して、Binder を生成します。

3.33.13.1.1 の 2.1 項と同様に、R_TSIP_GenerateTls13P256KeyIndex()を使用して、Ephemeral ECDH 公開鍵を生成します。

3.4 ClientHello 送信時に、pre_shared_key フィールドとして Binder を含めた Pre Shared Key の情報、また key_share フィールドとして ECDH 公開鍵も付加してサーバに送信します。

4. アプリケーションデータの送信（Send Application Data）

4.1 Pre Shared Key の Wrapped Key を入力として、R_TSIP_Tls13Generate0RttApplicationWriteKey()を使用して、0-RTT で使用する Client Write Key の Wrapped Key を生成します。

4.2 Client Write Key の Wrapped Key を入力として、R_TSIP_Tls13EncryptInit/Update/Final()を使用して、アプリケーションデータを暗号化します。

4.3 ClientHello の送信後に、暗号化したアプリケーションデータをサーバに送信します。

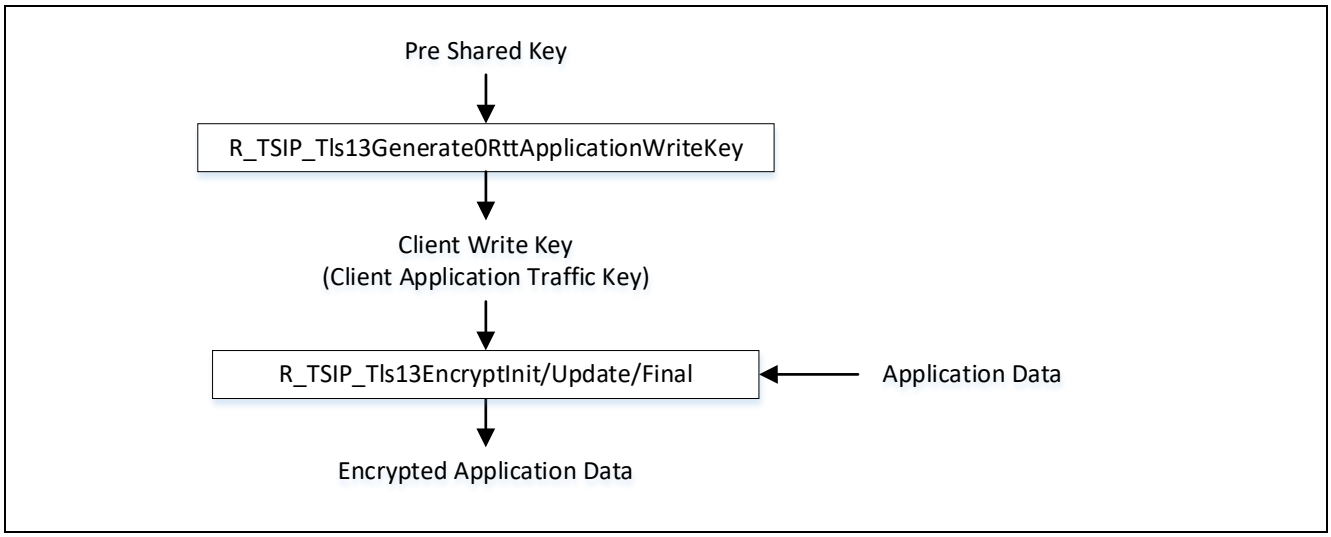

図 3-33 アプリケーションデータの送信

5. ハンドシェイクとアプリケーションデータ通信
   5.1 0-RTT としてのアプリケーションデータ送信以降は、Full Handshake と同様の通信シーケンスにより TLS1.3 通信を確立します。
   5.2 ハンドシェイクフェーズ終了後に、アプリケーションデータ通信を行います。

※0-RTT を使用する場合について、RFC8446 2.3 章に記載されているように、前方秘匿性が無いこと、リプレイ攻撃耐性が無いことがリスクとなります。この機能の使用については、本リスクを踏まえて判断してください。

3.13.2 TLS1.3 サーバ機能

No.	API	説明
1	R_TSIP_GenerateTlsRsaPublicKeyIndex R_TSIP_UpdateTlsRsaPublicKeyIndex R_TSIP_TlsRootCertificateVerification R_TSIP_TlsCertificateVerification R_TSIP_TlsCertificateVerificationExtension R_TSIP_Tls13EncryptInit R_TSIP_Tls13EncryptUpdate R_TSIP_Tls13EncryptFinal R_TSIP_Tls13DecryptInit R_TSIP_Tls13DecryptUpdate R_TSIP_Tls13DecryptFinal R_TSIP_Tls13SVCertificateVerifyGenerate R_TSIP_Tls13SVCertificateVerifyVerification	各機能で共通に使用
2	R_TSIP_GenerateTls13SVP256EccKeyIndex R_TSIP_Tls13SVGenerateEcdheSharedSecret R_TSIP_Tls13SVGenerateHandshakeSecret R_TSIP_Tls13SVGenerateServerHandshakeTrafficKey R_TSIP_Tls13SVGenerateClientHandshakeTrafficKey R_TSIP_Tls13SVServerHandshakeVerification R_TSIP_Tls13SVGenerateMasterSecret R_TSIP_Tls13SVGenerateApplicationTrafficKey R_TSIP_Tls13SVUpdateApplicationTrafficKey	主に Full Handshake で使用
3	R_TSIP_Tls13SVGenerateResumptionMasterSecret R_TSIP_Tls13SVGeneratePreSharedKey R_TSIP_Tls13SVGeneratePskBinderKey R_TSIP_Tls13SVGenerateResumptionHandshakeSecret	Resumption で使用
4	R_TSIP_Tls13SVGenerate0RttApplicationWriteKey	0-RTT で使用

3.13.2.1 Full Handshake

TLS1.3 サーバ機能を使用した Full Handshake 実施方法の概要を図 3-34 に示します。

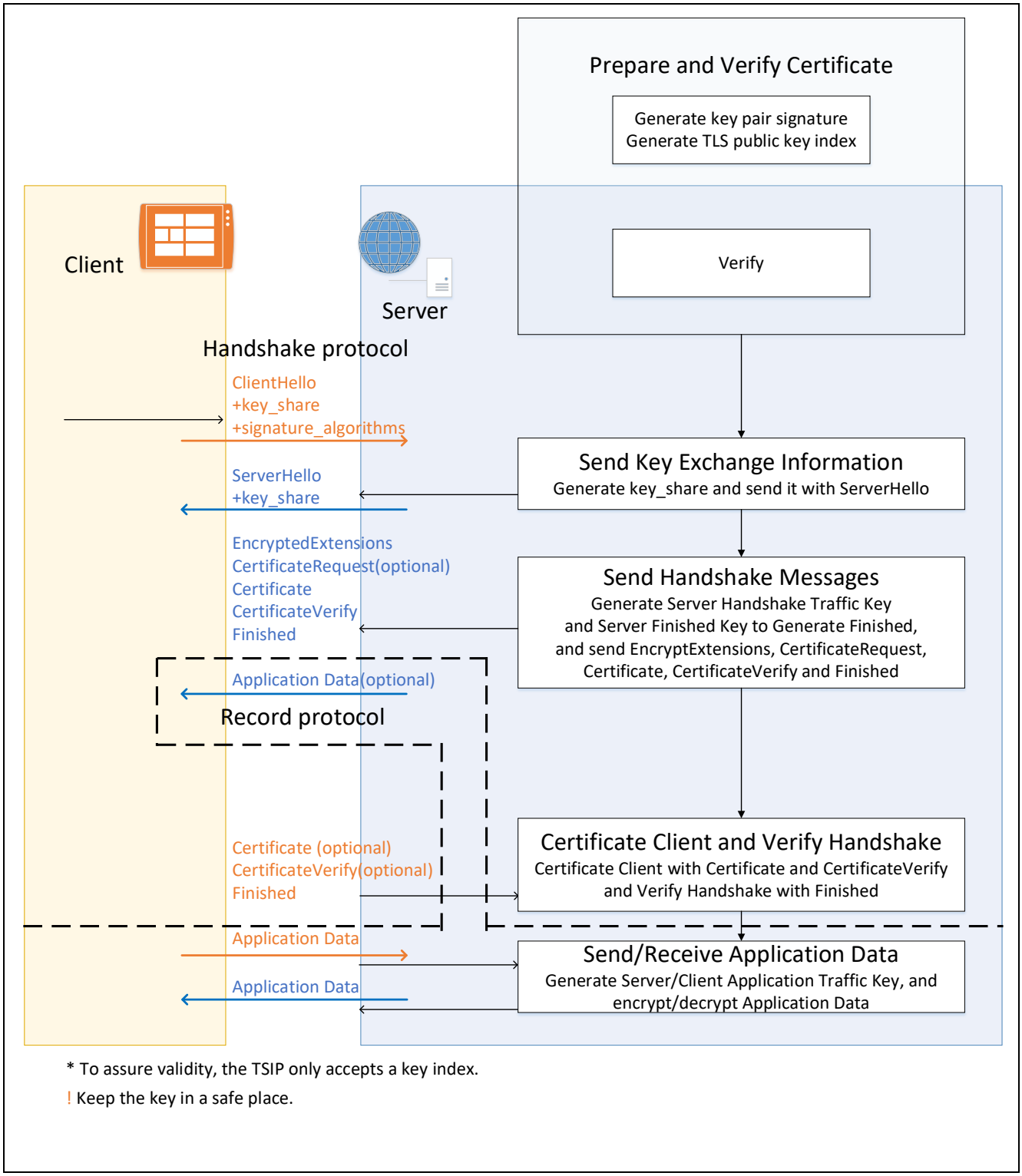

図 3-34 TLS1.3 サーバ機能 Full Handshake の実施方法

1. 事前準備と証明書の検証（Prepare and verify certificate）

1.1 事前に以下のものを準備します。

- 接続するサーバのルート CA 証明書
- RSA2048bit の鍵ペア
- サーバ証明書

1.2 用意した RSA 鍵ペアの秘密鍵を使って、用意したルート CA 証明書の署名値を生成(署名アルゴルズムは RSA2048 PSS with SHA256)します。また用意した RSA 鍵ペアの公開鍵は、UFPK でラップし、Encrypted Key にします。これらの作業は安全なサイトで行ってください。

ルート CA 証明書を複数登録したい場合は、束としてルート CA 証明書をまとめたデータに対して、署名を付けてください。ルート CA 証明書に署名することで、不正な証明書チェーンを使用した TLS セッションの実施を防ぐことができます。

1.3 RSA 鍵ペアの公開鍵の Encrypted Key を入力として、R_TSIP_TlsSVRsaPublicKeyIndex を使用して、公開鍵の Wrapped Key を生成します。詳細な手順については、3.7.1 鍵の注入と更新を参照してください。

1.4 R_TSIP_TlsRegisterCaCertificationPublicKeyIndex()を使用して、公開鍵の Wrapped Key を TSIP ドライバに登録します。

1.5 R_TSIP_TlsSVRootCertificateVerify()を使用して、ルート CA 証明書の束を検証し、暗号化したルート CA 証明書の公開鍵を生成します。

1.6 暗号化したルート CA 証明書の公開鍵、およびサーバ証明書の公開鍵を入力として、R_TSIP_TlsSVCertificateVerification()または R_TSIP_TlsSVCertificateVerificationExtension()を使用して、暗号化したサーバ証明書の公開鍵を生成します。

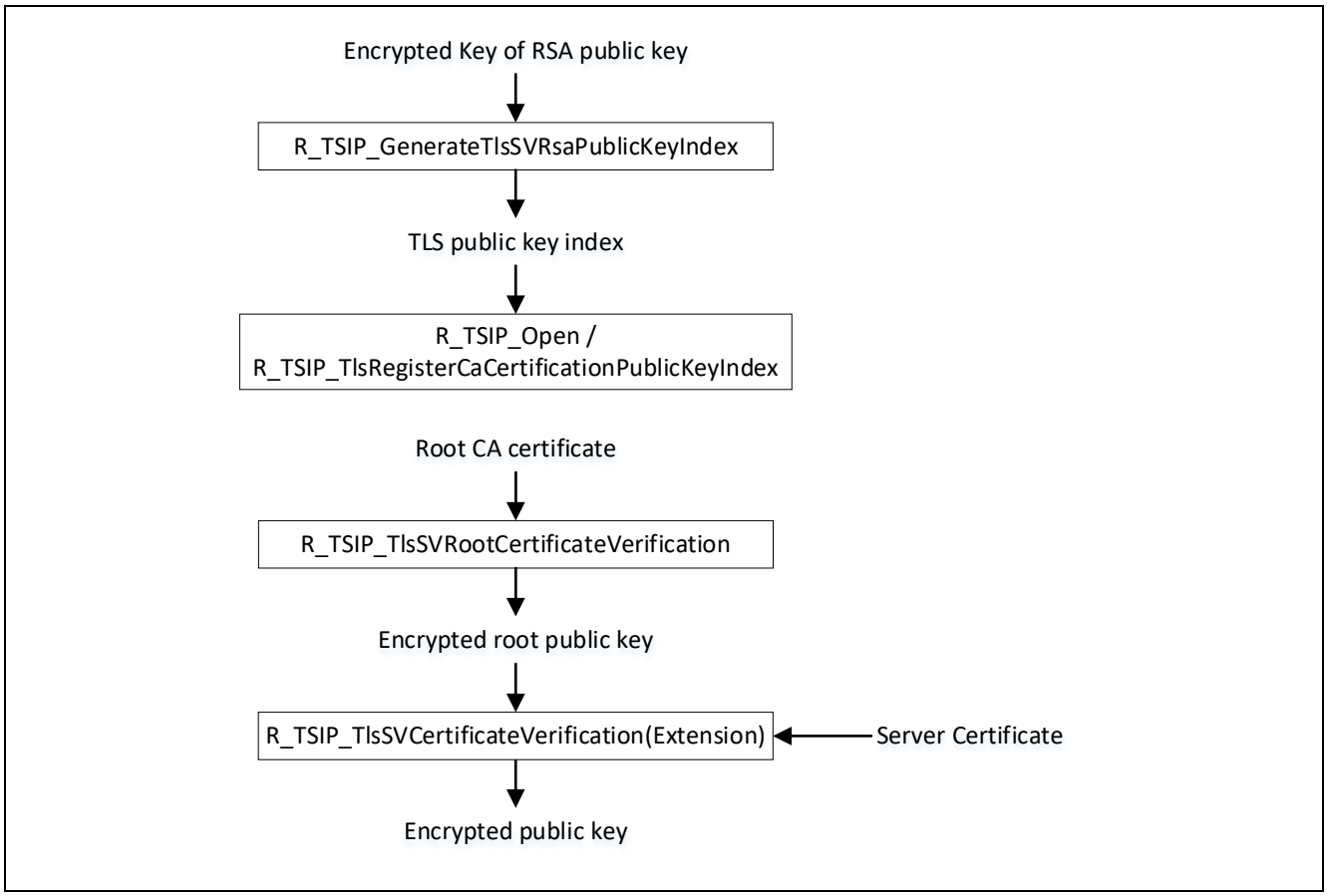

図 3-35 事前準備と証明書の検証

2. 鍵交換用データの送信（Send Key Exchange Information）

2.1 R_TSIP_GenerateTls13SVP256KeyIndex()を使用して Ephemeral ECDH 公開鍵を生成します。

2.2 ServerHello 送信時に、key_share フィールドとして ECDH 公開鍵をサーバに送信します。

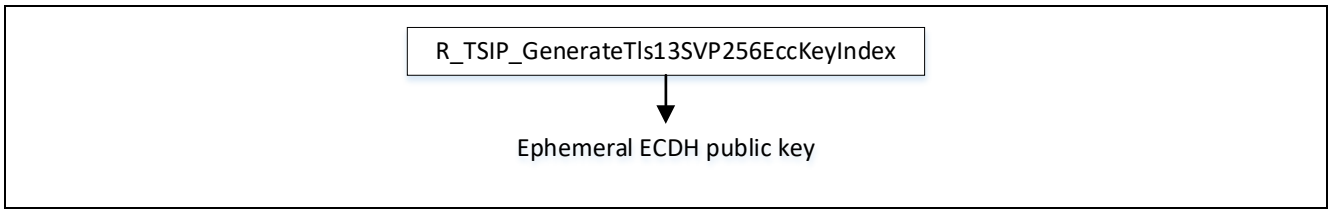

図 3-36 鍵交換用データの送信

3. ハンドシェイクメッセージの送信（Send Handshake Messages）

3.1 証明書の秘密鍵の Wrapped Key を入力として、R_TSIP_Tls13SVCertificateVerifyGenerate()を使用して、CertificateVerify フィールドを生成します。

3.2 クライアントから受信した公開鍵を入力として、R_TSIP_Tls13SVGenerateEcdheSharedSecret()を使用して Shared Secret の Wrapped Key を生成します。

3.3 Shared Secret の Wrapped Key を入力として、R_TSIP_Tls13SVGenerateHandshakeSecret()を使用して Handshake Secret の Wrapped Key を生成します。

3.4 Handshake Secret の Wrapped Key を入力として、R_TSIP_Tls13SVGenerateServerHandshakeTrafficKey()を使用して、Handshake プロトコルで使用する Server Write Key と Server Finished Key の Wrapped Key を生成します。

3.5 Server Write Key の Wrapped Key を入力として、R_TSIP_Tls13EncryptInit/Update/Final()を使用して、クライアントへ送信するハンドシェイクメッセージを暗号化します。

3.6 暗号化したハンドシェイクメッセージを送信します。

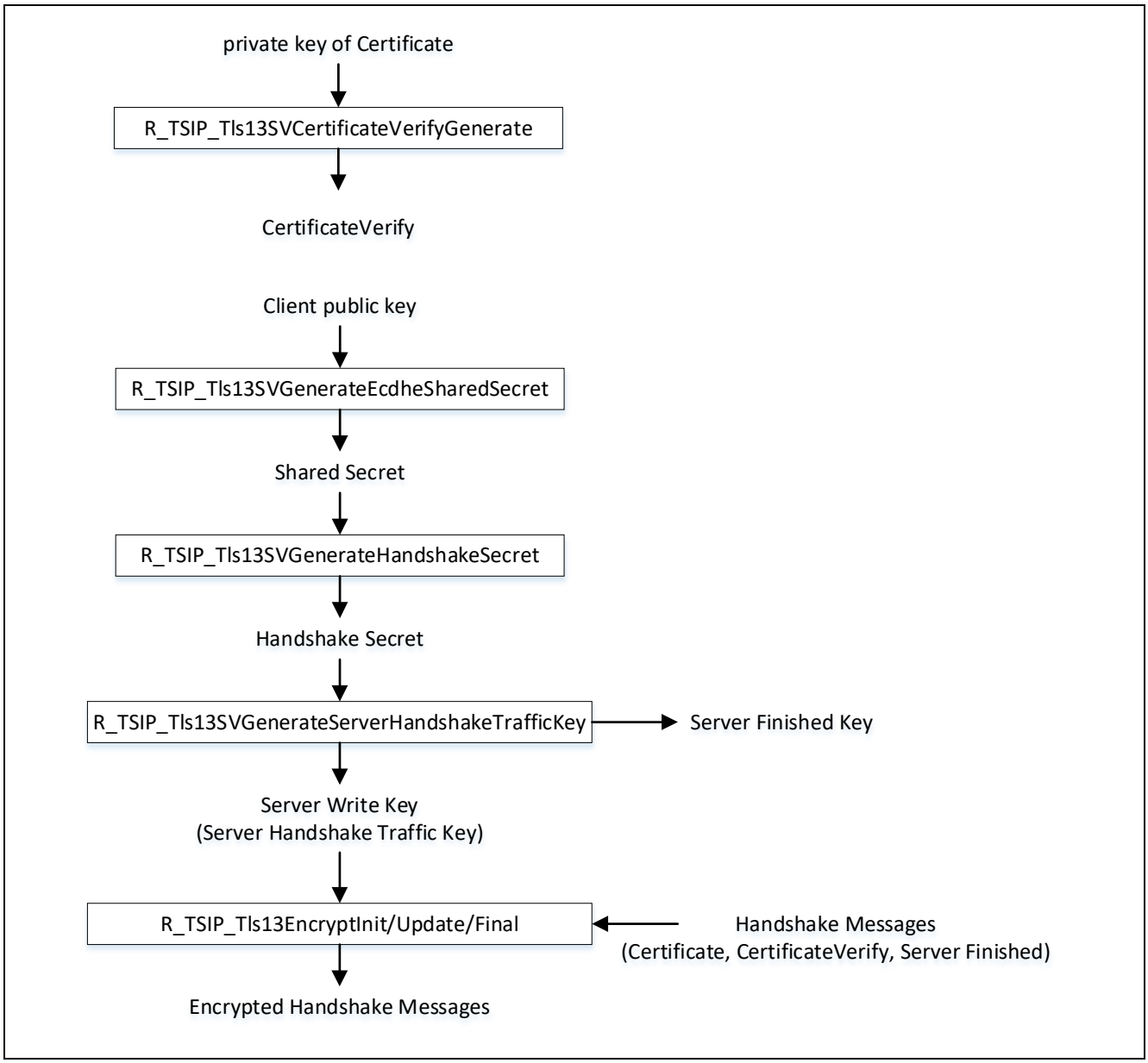

図 3-37 ハンドシェイクメッセージの送信

4. クライアントの認証とハンドシェイクの検証（Certificate Client and Verify Handshake）

4.1 Handshake Secret の Wrapped Key を入力として、
R_TSIP_Tls13SVGenerateClientHandshakeTrafficKey()を使用して、ハンドシェイクフェーズで使用する Client Write Key と Client Finished Key の Wrapped Key を生成します。

4.2 Client Write Key の Wrapped Key を入力として、R_TSIP_Tls13DecryptInit/Update/Final()を使用して、クライアントから受信した暗号化されているハンドシェイクメッセージを復号します。

4.3 ハンドシェイクメッセージ内の(Client) Certificate フィールドを入力として、
R_TSIP_TlsCertificateVerification()を使用して、証明書の署名を検証します。

4.4 ハンドシェイクッセージ内の(Client) CertificateVerify フィールドを入力として、
R_TSIP_Tls13SVCertificateVerifyVerification()を使用して、(Client) CertificateVerify フィールドを検証します。

4.5 ハンドシェイクメッセージ内の(Client) Finished フィールドと Client Finished Key の Wrapped Key を入力として、R_TSIP_Tls13SVClientHandshakeVerification()を使用して、ハンドシェイクを検証します。

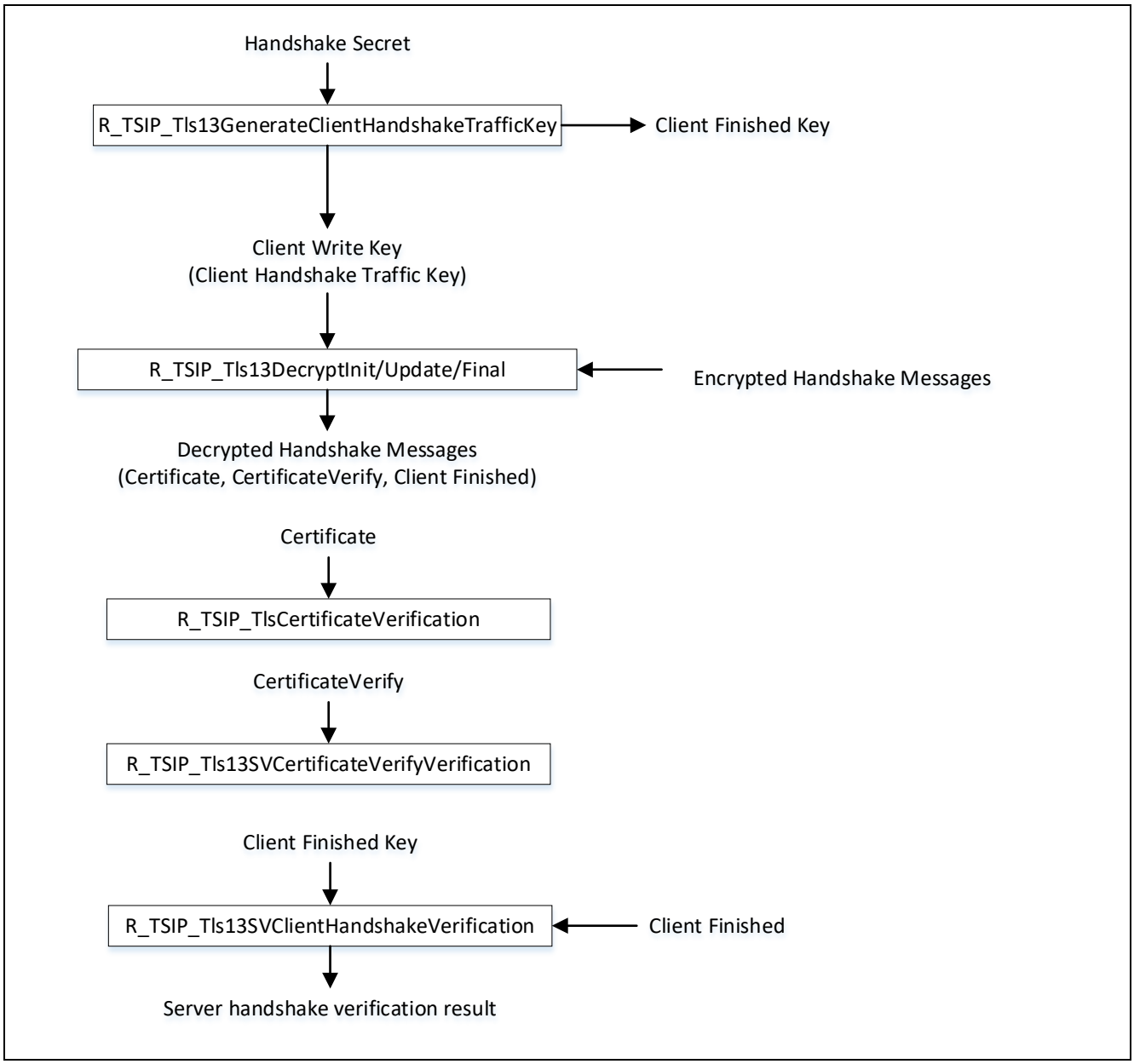

図 3-38 クライアントの認証とハンドシェイクの検証

5. アプリケーションデータの送受信（Send/Receive Application Data）

5.1 Handshake Secret の Wrapped Key を入力として、R_TSIP_Tls13SVGenerateMasterSecret()を使用して、Master Secret の Wrapped Key を生成します。
5.2 Master Secret の Wrapped Key を入力として、R_TSIP_Tls13SVGenerateApplicationTrafficKey()を使用して、Record プロトコルで使用する Server Write Key と Client Write Key の Wrapped Key および Application Secret の Wrapped Key を生成します。
5.3 Server Write Key または Client Write Key を更新する際には、Application Secret の Wrapped Key を入力として、R_TSIP_Tls13SVUpdateApplicationTrafficKey()を使用します。
5.4 サーバへの送信データを暗号化する際には、Server Write Key の Wrapped Key を入力として R_TSIP_Tls13EncryptInit/Update/Final()を使用します。
5.5 サーバからの受信データを復号する際には、Client Write Key の Wrapped Key を入力として R_TSIP_Tls13DecryptInit/Update/Final()を使用します。

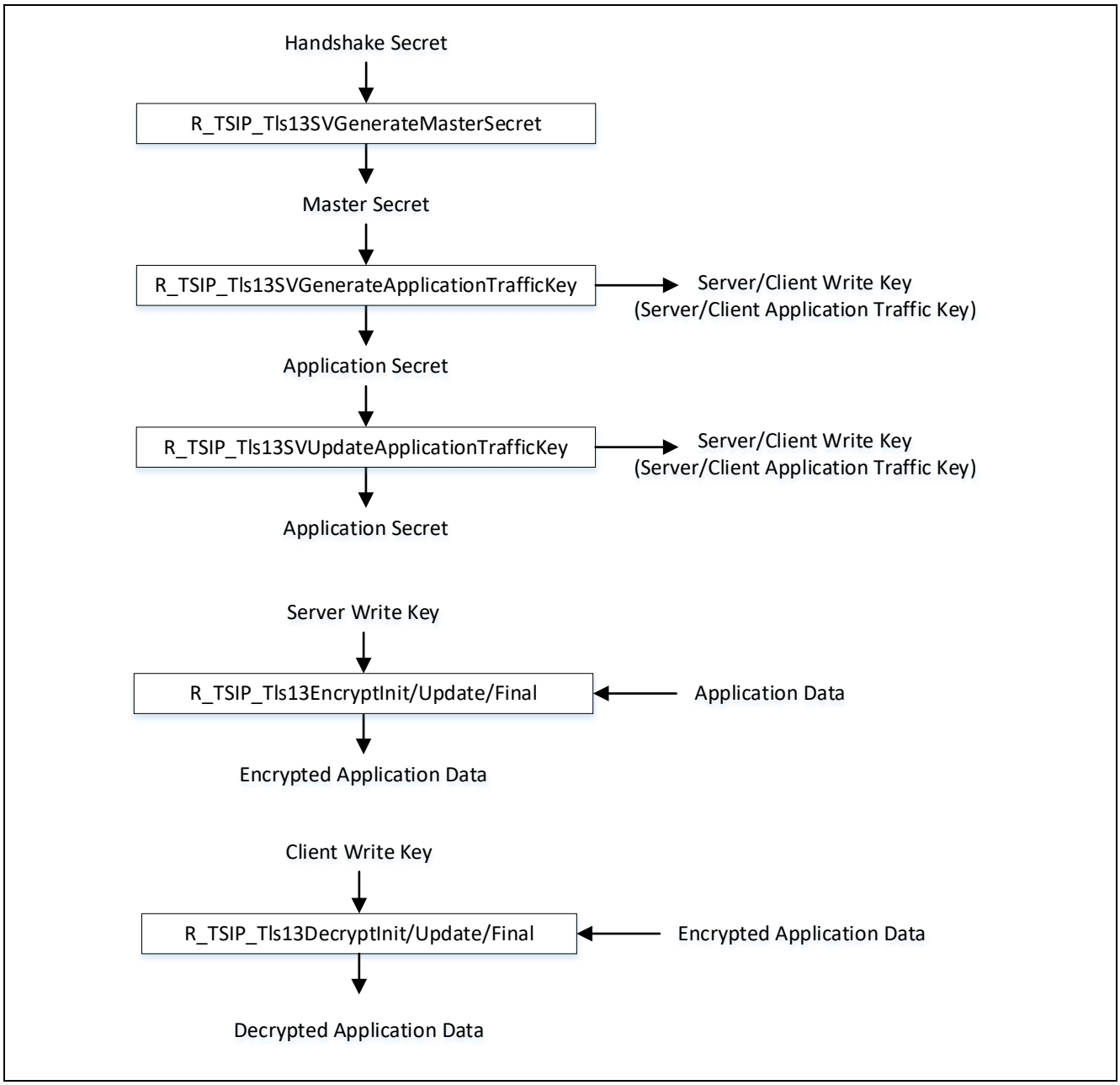

図 3-39 アプリケーションデータの送受信

※ClientFinished の受信を待たずにサーバから Application Data を送信する場合について、ClientFinished の検証でエラーが発生した場合、サーバプログラムが改ざんされていないという条件下でのみエラーを検出することができます。この機能の使用については、本リスクを踏まえて判断してください。

3.13.2.2 Resumption

TLS1.3 サーバ機能を使用した Resumption 実施方法の概要を図 3-40 に示します。

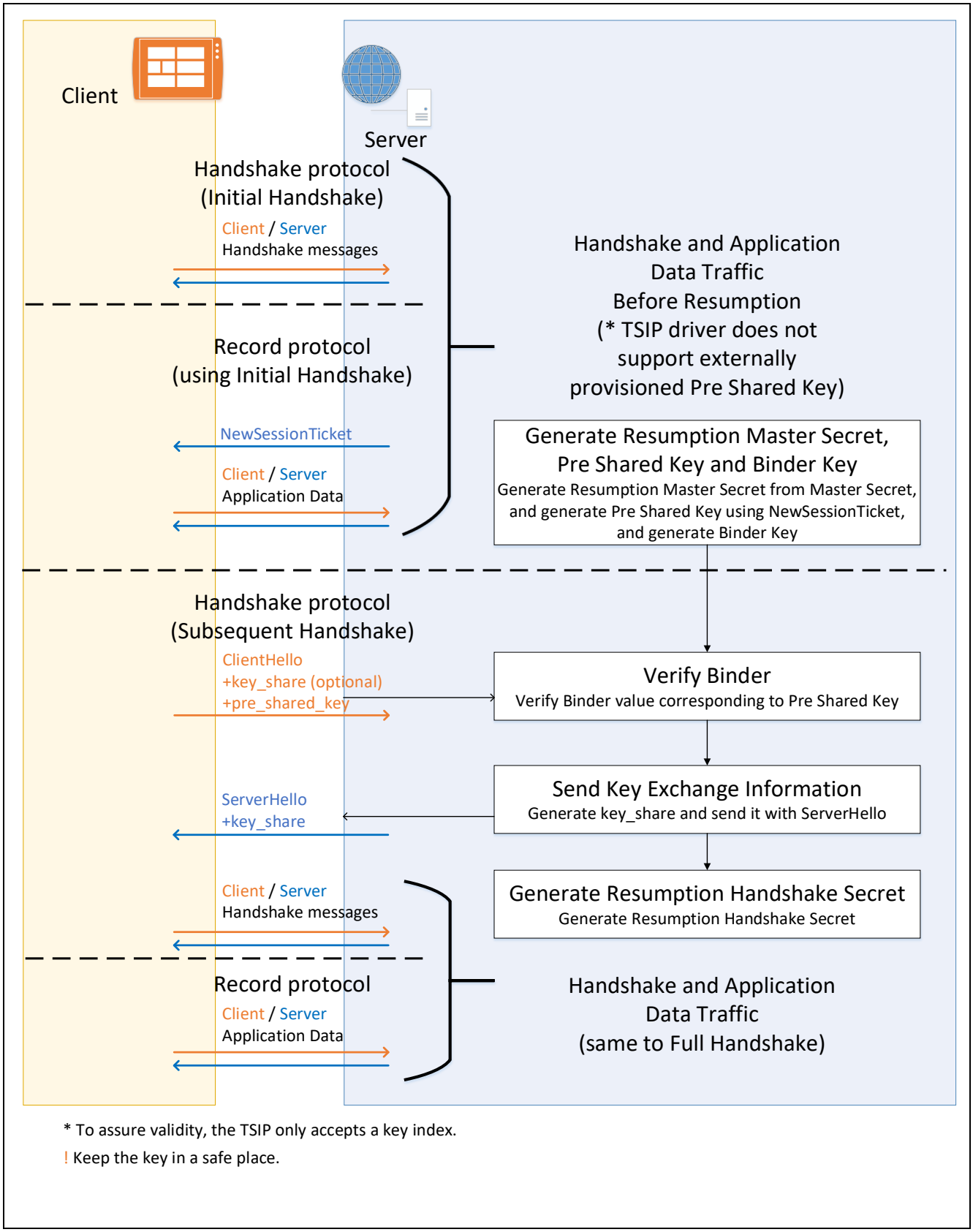

図 3-40 TLS1.3 サーバ機能 Resumption の実施方法

1. Resumption 前のハンドシェイクとアプリケーションデータ通信

1.1 Full Handshake により TLS1.3 通信を確立します。

※TSIP ドライバにおいては、Handshake を確立して生成した Pre Shared Key 以外の使用を許可していません。

2. Resumption Master Secret、Pre Shared Key および Binder Key の生成（Generate Resumption Master Secret, Pre Shared Key and Binder Key）

2.1 Full Handshake 時に使用した Master Secret の Wrapped Key を入力として、R_TSIP_Tls13SVGenerateResumptionMasterSecret()を使用して、Resumption Master Secret の Wrapped Key を生成します。
2.2 Resumption Master Secret の Wrapped Key とクライアントへ送信した New Session Ticket 内の Ticket Nonce を入力として、R_TSIP_Tls13SVGeneratePreSharedKey()を使用して、Pre Shared Key の Wrapped Key を生成します。
2.3 Pre Shared Key の Wrapped Key を入力として、R_TSIP_Tls13SVGeneratePskBinderKey()を使用して、Binder Key の Wrapped Key を生成します。

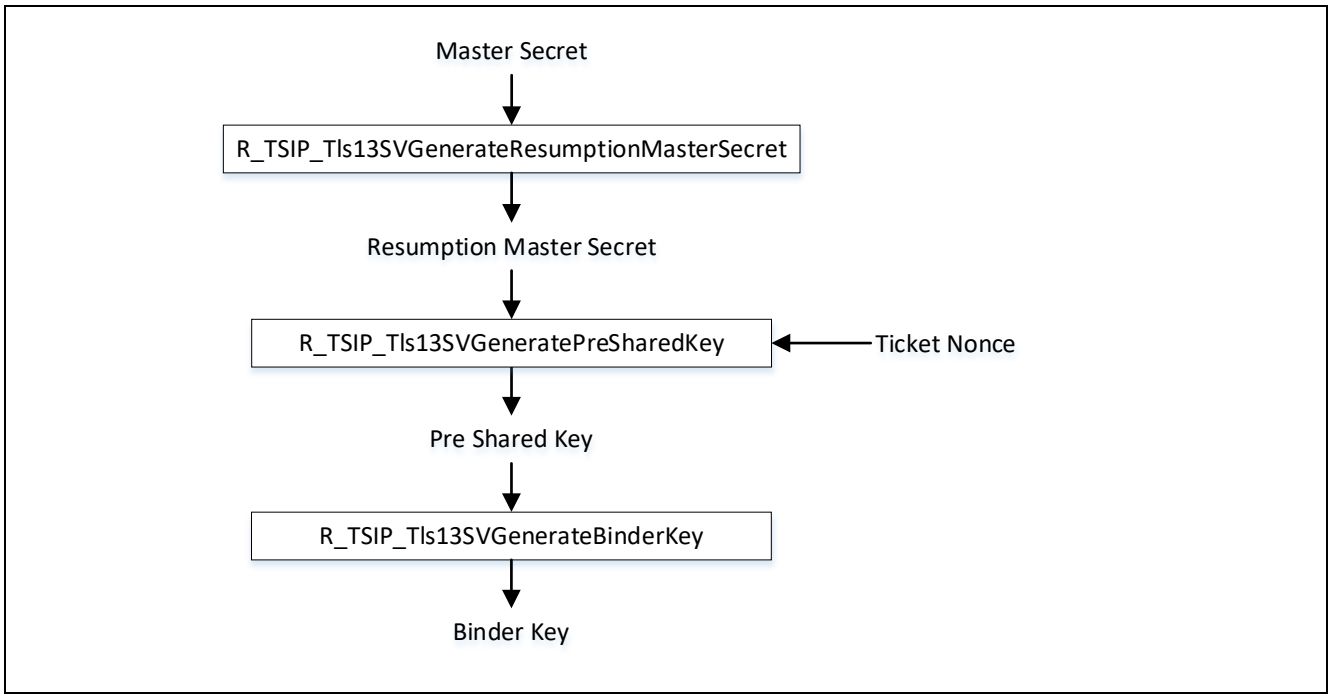

図 3-41 Resumption Master Secret、Pre Shared Key および Binder Key の生成

3. Binder の検証（Verify Binder）
3.1 ClientHello と共に受信した Binder と Pre Shared Key の情報に対応した Binder Key を入力して、R_TSIP_Sha256HmacVerifyInit/Update/Final()を使用して、Binder を検証します。

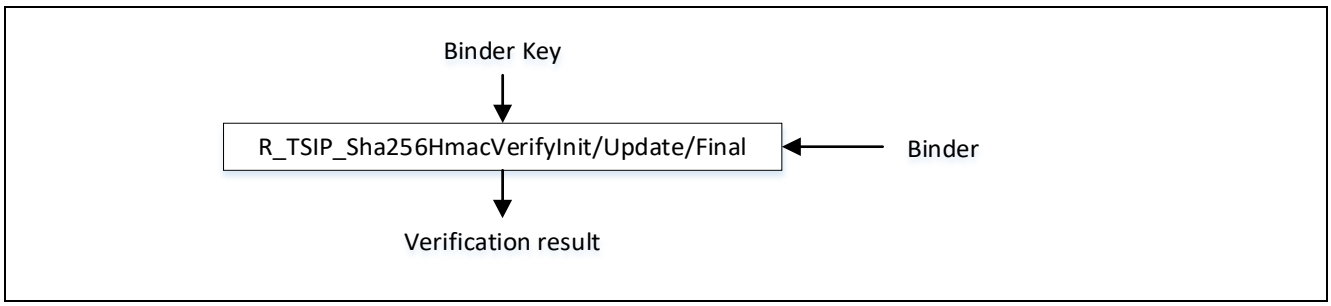

図 3-42 Binder の検証

4. Resumption 用 Handshake Secret の生成（Generate Resumption Handshake Secret）
   4.13.13.2.1 の 3.2 項と同様に、サーバから受信した公開鍵を入力として、R_TSIP_Tls13SVGenerateEcdheSharedSecret()を使用して Shared Secret の Wrapped Key を生成します。
   4.2 Pre Shared Key と Shared Secret の Wrapped Key を入力として、R_TSIP_Tls13SVGenerateResumptionHandshakeSecret()を使用して Resumption 用の Handshake Secret の Wrapped Key を生成します。

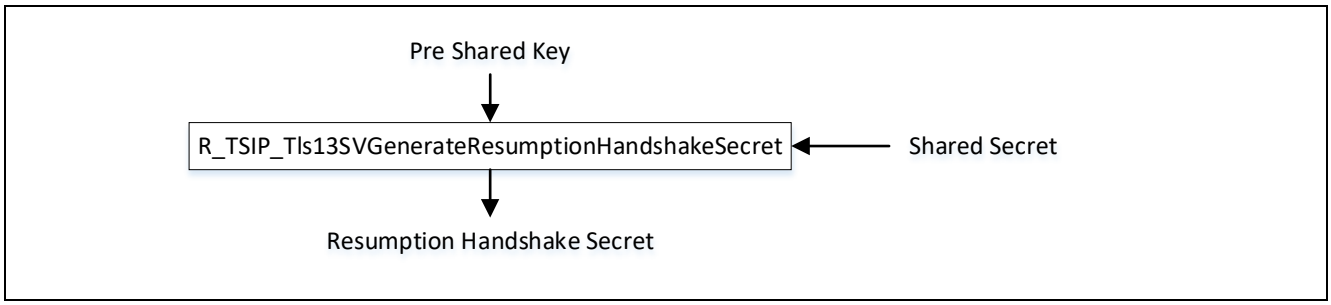

図 3-43 Resumption 用 Handshake Secret の生成

5. ハンドシェイクとアプリケーションデータ通信
   5.1 Resumption 用の Handshake Secret 生成以降は、Full Handshake と同様の通信シーケンスにより TLS1.3 通信を確立します。
   5.2 ハンドシェイクフェーズ終了後に、アプリケーションデータ通信を行います。

3.13.2.3 0-RTT

TLS1.3 サーバ機能を使用した 0-RTT 実施方法の概要を図 3-40 に示します。

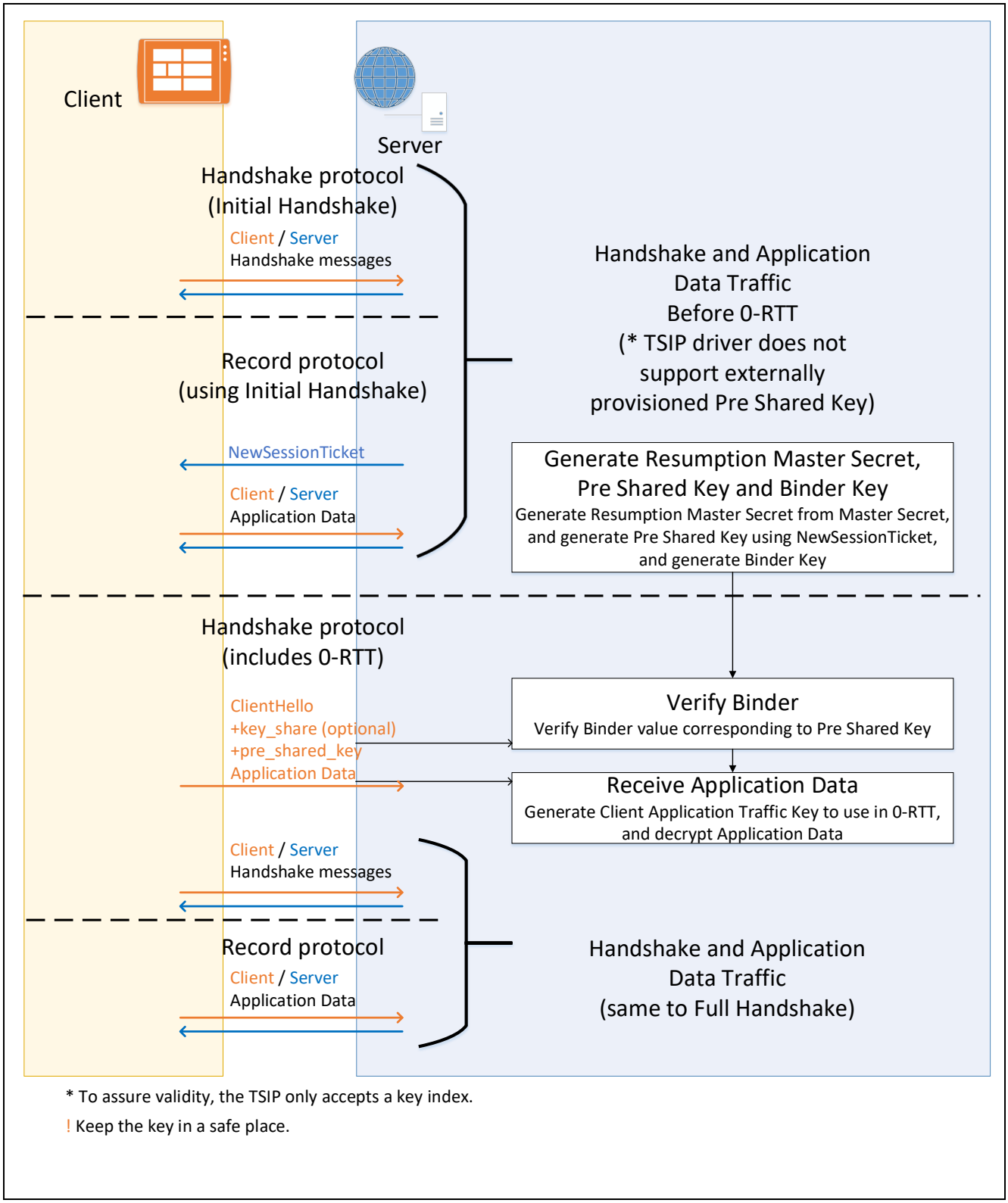

図 3-44 TLS1.3 サーバ機能 0-RTT の実施方法

1. Resumption 前のハンドシェイクとアプリケーションデータ通信

1.1 Full Handshake により TLS1.3 通信を確立します。

※TSIP ドライバにおいては、Handshake を確立して生成した Pre Shared Key 以外の使用を許可していません。

2. Resumption Master Secret、Pre Shared Key および Binder Key の生成（Generate Resumption Master Secret, Pre Shared Key and Binder Key）

2.1 以降の処理も含めて 3.13.2.2 の 2. 項と同様に、Full Handshake 時に使用した Master Secret の Wrapped Key を入力として、R_TSIP_Tls13SVGenerateResumptionMasterSecret()を使用して、Resumption Master Secret の Wrapped Key を生成します。
2.2 Resumption Master Secret の Wrapped Key とクライアントへ送信した New Session Ticket 内の Ticket Nonce を入力として、R_TSIP_Tls13SVGeneratePreSharedKey()を使用して、Pre Shared Key の Wrapped Key を生成します。
2.3 Pre Shared Key の Wrapped Key を入力として、R_TSIP_Tls13SVGeneratePskBinderKey()を使用して、Binder Key の Wrapped Key を生成します。

3. Binder の検証（Verify Binder）
3.13.13.2.2 の 3.1 項と同様に、ClientHello と共に受信した Binder と Pre Shared Key の情報に対応した Binder Key を入力して、R_TSIP_Sha256HmacVerifyInit/Update/Final()を使用して、Binder を検証します。

4. アプリケーションデータの受信（Receive Application Data）
4.1 Pre Shared Key の Wrapped Key を入力として、R_TSIP_Tls13SVGenerate0RttApplicationWriteKey()を使用して、0-RTT で使用する Client Write Key の Wrapped Key を生成します。
4.2 Client Write Key の Wrapped Key を入力として、R_TSIP_Tls13DecryptInit/Update/Final()を使用して、アプリケーションデータを復号します。

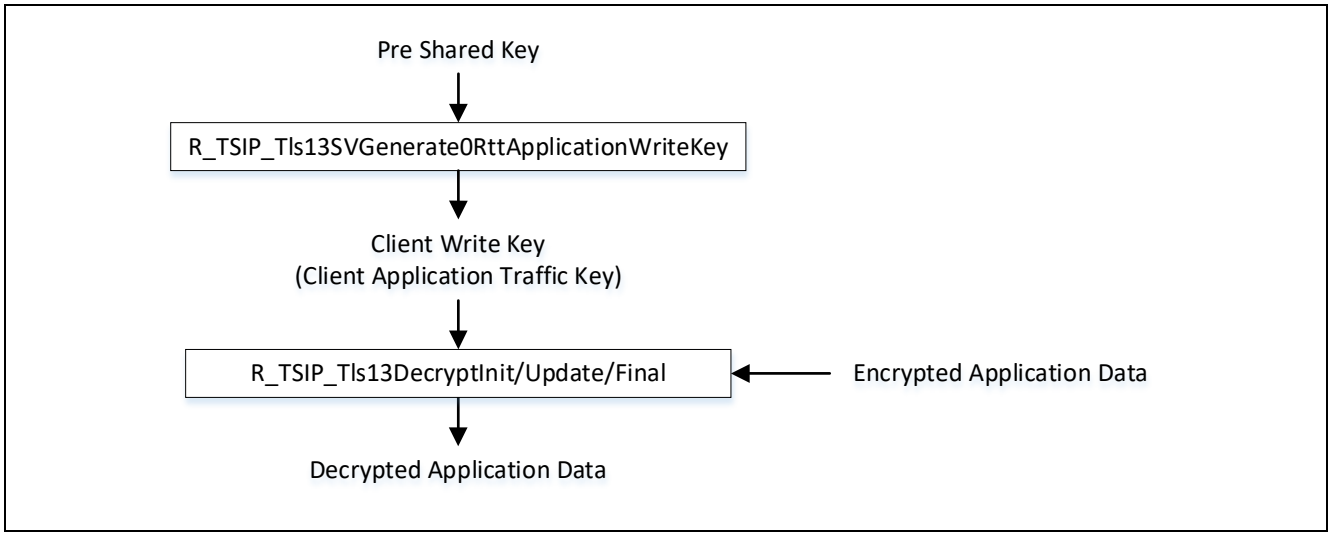

図 3-45 アプリケーションデータの受信

5. ハンドシェイクとアプリケーションデータ通信
5.1 0-RTT としてのアプリケーションデータ送信以降は、Full Handshake と同様の通信シーケンスにより TLS1.3 通信を確立します。
5.2 ハンドシェイクフェーズ終了後に、アプリケーションデータ通信を行います。

※0-RTT を使用する場合について、RFC8446 2.3 章に記載されているように、前方秘匿性が無いこと、リプレイ攻撃耐性が無いことがリスクとなります。この機能の使用については、本リスクを踏まえて判断してください。

3.14 ファームウェアアップデート

TSIP ドライバは、暗号化されたプログラムの復号と MAC 検証を行うファームウェアアップデート機能をサポートしています。
　本ドライバは、以下のファームアップデートのための API を提供します。

No.	API	説明
1	R_TSIP_StartUpdateFirmware	TSIP をファームウェアアップデート機能が使用できる状態に遷移させます。
2	R_TSIP_GenerateFirmwareMAC	暗号化されたプログラムを復号して、MAC 検証を行い、MAC を生成します。
3	R_TSIP_VerifyFirmwareMAC	指定されたエリアに対して、R_TSIP_GenerateFirmwareMAC で生成された MAC に対する検証を行います。

　ファームウェアアップデートでは、暗号化したプログラムならびに、プログラムの暗号化した使用した鍵を安全に配送する機構を備えております。図 3.2 に Renesas Key Wrap service を含むプログラムの暗号化、暗号鍵の注入と MAC 検証のフローを示します。

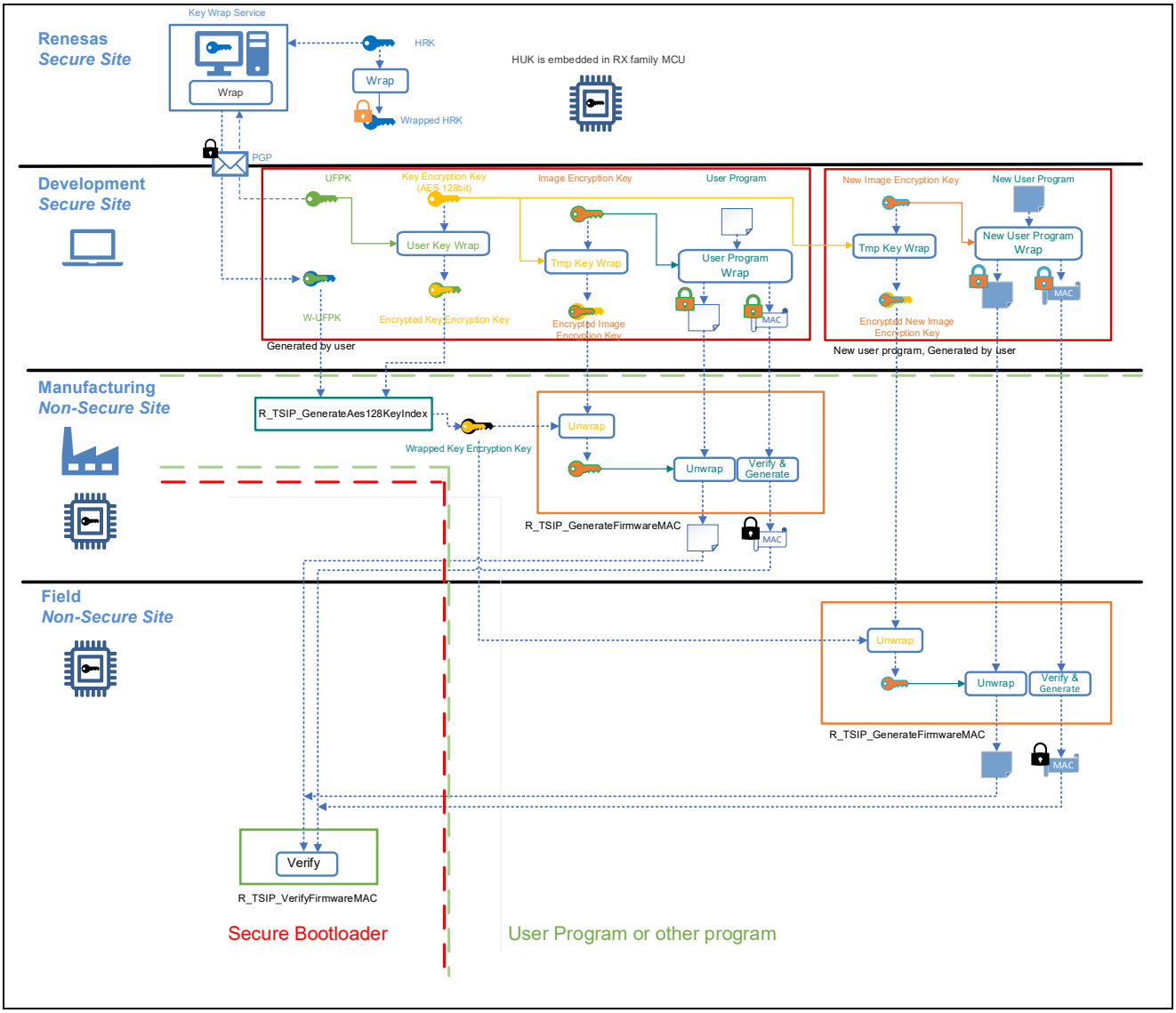

図 3-46　ファームウェアアップデートフロー

3.14.1 セキュアブート

セキュアブートとは、ユーザプログラムの改ざん検出機能をもつ、リセット後にユーザプログラムを実行する前に実行されるプログラムを指します。

セキュアブートとして、R_TSIP_VerifyFirmwareMAC を使用することが可能です。R_TSIP_VerifyFirmwareMAC には、R_TSIP_GenerateFirmwareMAC から出力される MAC 値を入力してください。

3.14.2 ファームウェアアップデート

ファームウェアアップデート動作では、暗号化されたプログラムを復号して、MAC 検証を行う R_TSIP_GenerateFirmwareMAC を使用可能です。R_TSIP_GenerateFirmwareMAC では、MAC 検証に成功すると、新たに HRK に紐づく MAC を生成します。

R_TSIP_GenerateFirmwareMAC を使用する場合、R_TSIP_StartUpdateFirmware を呼び出して、TSIP を FrimwareUpdate 状態にした後、実行可能です。

3.14.3 ユーザプログラムの暗号化

以下にユーザプログラムの暗号化方法を示します。ファームウェアの暗号化アルゴリズムは AES-CBC, MAC は AES-CBCMAC 使用しています。

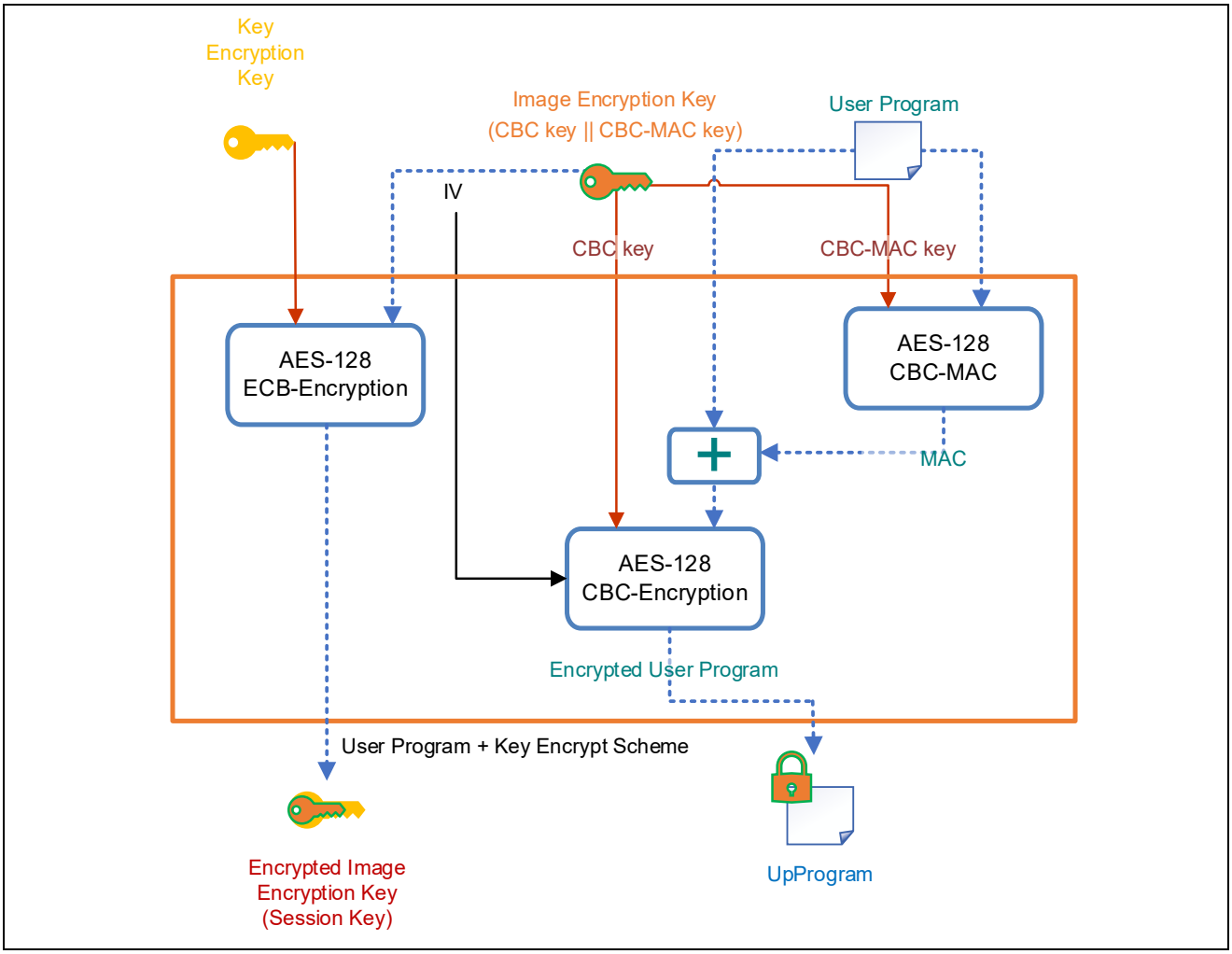

図 3-47　ファームウェアと Image Encryption Key の暗号化方法

ユーザプログラムを暗号化するための鍵(Image Encryption Key)、およびその鍵を暗号化するためのユーザプログラム鍵(Key Encryption Key)を用意します。Image Encryption key でユーザプログラムの MAC 生成および暗号化(UpProgram)を実施します。Key Encryption Key で Image Encryption key を暗号化し、Encrypted Image Encryption Key(Session Key)を生成します。

4. API 関数

4.1 API 一覧

TSIP ドライバでは、以下の API を実装しています。

① 共通機能 API
② 乱数生成 API
③ AES 暗号/復号 API
④ DES 暗号/復号 API
⑤ ARC4 暗号/復号 API
⑥ RSA 演算 API
⑦ ECC 署名生成検証 API
⑧ HASH 演算 API
⑨ HMAC 生成/検証 API
⑩ DH 演算 API
⑪ ECDH 鍵交換 API
⑫ Key Wrap API
⑬ TLS 機能 API
⑭ ファームウェアアップデート/セキュアブート API

以下表に実装している API を以下にまとめます。API 中の *XXX* は各アルゴリズムのビット長もしくは、SHA のモードを示します。

表 4-1　共通機能 API

API	説明	TSIP-Lite	TSIP
R_TSIP_Open	TSIP 機能を有効にします	✔	✔
R_TSIP_Close	TSIP 機能を無効にします	✔	✔
R_TSIP_SoftwareReset	TSIP モジュールをリセットします。	✔	✔
R_TSIP_GetVersion	TSIP ドライバのバージョンを出力します。	✔	✔
R_TSIP_GenerateUpdateKeyRingKeyIndex	鍵更新用鍵用の Wrapped Key を生成します。	✔	✔

表 4-2　乱数生成 API

API	説明	TSIP-Lite	TSIP
R_TSIP_GenrateRandomNumber	乱数を生成します。	✔	✔

表 4-3　AES 暗号/復号 API

API	説明	TSIP-Lite	TSIP
R_TSIP_GenerateAes*XXX*KeyIndex	AES の Wrapped Key を生成します。	✔	✔
R_TSIP_UpdateAes*XXX*KeyIndex	AES の Wrapped Key を更新します。	✔	✔
R_TSIP_GenerateAes*XXX*RandomKeyIndex	乱数から AES 鍵を生成し、Wrapped Key で出力します。	✔	✔
R_TSIP_Aes*XXX*EcbEncryptInit R_TSIP_Aes*XXX*EcbEncryptUpdate R_TSIP_Aes*XXX*EcbEncryptFinal	AES-ECB モードで暗号化を行います。	✔	✔
R_TSIP_Aes*XXX*EcbDecryptInit R_TSIP_Aes*XXX*EcbDecryptUpdate R_TSIP_Aes*XXX*EcbDecryptFinal	AES-ECB モードで復号を行います。	✔	✔
R_TSIP_Aes*XXX*CbcEncryptInit R_TSIP_Aes*XXX*CbcEncryptUpdate R_TSIP_Aes*XXX*CbcEncryptFinal	AES-CBC モードで暗号化を行います。	✔	✔
R_TSIP_Aes*XXX*CbcDecryptInit R_TSIP_Aes*XXX*CbcDecryptUpdate R_TSIP_Aes*XXX*CbcDecryptFinal	AES-CBC モードで復号を行います。	✔	✔
R_TSIP_Aes*XXX*CtrInit R_TSIP_Aes*XXX*CtrUpdate R_TSIP_Aes*XXX*CtrFinal	AES-CTR モードで暗号化または復号を行います。	✔	✔
R_TSIP_Aes*XXX*GcmEncryptInit R_TSIP_Aes*XXX*GcmEncryptUpdate R_TSIP_Aes*XXX*GcmEncryptFinal	AES-GCM モードで暗号化を行います。	✔	✔
R_TSIP_Aes*XXX*GcmDecryptInit R_TSIP_Aes*XXX*GcmDecryptUpdate R_TSIP_Aes*XXX*GcmDecryptFinal	AES-GCM モードで復号を行います。	✔	✔
R_TSIP_Aes*XXX*CcmEncryptInit R_TSIP_Aes*XXX*CcmEncryptUpdate R_TSIP_Aes*XXX*CcmEncryptFinal	AES-CCM モードで暗号化を行います。	✔	✔
R_TSIP_Aes*XXX*CcmDecryptInit R_TSIP_Aes*XXX*CcmDecryptUpdate R_TSIP_Aes*XXX*CcmDecryptFinal	AES-CCM モードで復号を行います。	✔	✔
R_TSIP_Aes*XXX*CmacGenerateInit R_TSIP_Aes*XXX*CmacGenerateUpdate R_TSIP_Aes*XXX*CmacGenerateFinal	AES-CMAC モードの MAC 生成を行います。	✔	✔
R_TSIP_Aes*XXX*CmacVerifyInit R_TSIP_Aes*XXX*CmacVerifyUpdate R_TSIP_Aes*XXX*CmacVerifyFinal	AES-CMAC モードで、MAC の検証を行います。	✔	✔

表 4-4　DES 暗号/復号 API

API	説明	TSIP-Lite	TSIP
R_TSIP_GenerateTdesKeyIndex	TDES の Wrapped Key を生成します。	-	✔
R_TSIP_UpdateTdesKeyIndex	TDES の Wrapped Key を更新します。	-	✔
R_TSIP_GenerateTdesRandomKeyIndex	乱数から TDES の鍵を生成し、Wrapped Key で出力します。	-	✔
R_TSIP_TdesEcbEncryptInit R_TSIP_TdesEcbEncryptUpdate R_TSIP_TdesEcbEncryptFinal	TDES-ECB モードで暗号化を行います。	-	✔
R_TSIP_TdesEcbDecryptInit R_TSIP_TdesEcbDecryptUpdate R_TSIP_TdesEcbDecryptFinal	TDES-ECB モードで復号を行います。	-	✔
R_TSIP_TdesCbcEncryptInit R_TSIP_TdesCbcEncryptUpdate R_TSIP_TdesCbcEncryptFinal	TDES-CBC モードで暗号化を行います。	-	✔
R_TSIP_TdesCbcDecryptInit R_TSIP_TdesCbcDecryptUpdate R_TSIP_TdesCbcDecryptFinal	TDES-CBC モードで復号を行います。	-	✔

表 4-5　ARC4 暗号/復号 API

API	説明	TSIP-Lite	TSIP
R_TSIP_GenerateArc4KeyIndex	ARC4 の Wrapped Key を生成します。	-	✔
R_TSIP_UpdateArc4KeyIndex	ARC4 の Wrapped Key を更新します。	-	✔
R_TSIP_GenerateArc4RandomKeyIndex	乱数から ARC4 の鍵を生成し、Wrapped Key で出力します。	-	✔
R_TSIP_Arc4EncryptInit R_TSIP_Arc4EncryptUpdate R_TSIP_Arc4EncryptFinal	ARC4 暗号化を行います。	-	✔
R_TSIP_Arc4DecryptInit R_TSIP_Arc4DecryptUpdate R_TSIP_Arc4DecryptFinal	ARC4 復号を行います。	-	✔

表 4-6　RSA 演算 API

API	説明	TSIP-Lite	TSIP
R_TSIP_GenerateRsa*XXX*PrivateKeyIndex	RSA 秘密鍵の Wrapped Key を生成します。	-	✔
R_TSIP_GenerateRsa*XXX*PublicKeyIndex	RSA 公開鍵の Wrapped Key を生成します。	-	✔
R_TSIP_UpdateRsa*XXX*PrivateKeyIndex	RSA 秘密鍵の Wrapped Key を更新します。	-	✔
R_TSIP_UpdateRsa*XXX*PublicKeyIndex	RSA 公開鍵の Wrapped Key を更新します。	-	✔
R_TSIP_GenerateRsa*XXX*RandomKeyIndex	乱数から RSA 鍵ペアの Wrapped Key を生成します。Exponent は 0x10001 固定です。	-	✔
R_TSIP_RsaesPkcs*XXX*Encrypt	RSAES-PKCS1-V1_5 による RSA 暗号化をします。	-	✔
R_TSIP_RsaesPkcs*XXX*Decrypt	RSAES-PKCS1-V1_5 による RSA 復号をします。	-	✔
R_TSIP_RsassaPkcs*XXX*SignatureGenerate	RSASSA-PKCS1-V1_5 による電子署名を生成します。	-	✔
R_TSIP_RsassaPkcs*XXX*SignatureVerification	RSASSA-PKCS1-V1_5 による電子署名の検証をします。	-	✔
R_TSIP_RsassaPss*XXX*SignatureGenerate	RSASSA-PSS による電子署名を生成します。	-	✔
R_TSIP_RsassaPss*XXX*SignatureVerification	RSASSA-PSS による電子署名の検証をします。	-	✔

表 4-7　ECC 署名生成/検証 API

API	説明	TSIP-Lite	TSIP
R_TSIP_GenerateEccP*XXX*PublicKeyIndex	ECC 公開鍵の Wrapped Key を生成します。	-	✔
R_TSIP_GenerateEccP*XXX*PrivateKeyIndex	ECC 秘密鍵の Wrapped Key を生成します。	-	✔
R_TSIP_UpdateEccP*XXX*PublicKeyIndex	ECC 公開鍵の Wrapped Key を更新します。	-	✔
R_TSIP_UpdateEccP*XXX*PrivateKeyIndex	ECC 秘密鍵の Wrapped Key を更新します。	-	✔
R_TSIP_GenerateEccP*XXX*RandomKeyIndex	乱数から ECC 鍵ペアの Wrapped Key を生成します。	-	✔
R_TSIP_EcdsaP*XXX*SignatureGenerate	ECDSA による電子署名を生成します。	-	✔
R_TSIP_EcdsaP*XXX*SignatureVerification	ECDSA による電子署名の検証をします。	-	✔

表 4-8　HASH 演算 API

API	説明	TSIP-Lite	TSIP
R_TSIP_Sha*XXX*Init R_TSIP_Sha*XXX*Update R_TSIP_Sha*XXX*Final	SHA によるハッシュ値演算を行います。	-	✔
R_TSIP_Md5Init R_TSIP_Md5Update R_TSIP_Md5Final	MD5 によるハッシュ値演算を行います。	-	✔
R_TSIP_GetCurrentHashDigestValue	入力済みデータに対するハッシュ値を取得します。	-	✔

表 4-9　HMAC 生成/検証 API

API	説明	TSIP-Lite	TSIP
R_TSIP_GenerateSha*XXX*HmacKeyIndex	SHA-HMAC の Wrapped Key を生成します。	-	✔
R_TSIP_UpdateSha*XXX*HmacKeyIndex	SHA-HMAC の Wrapped Key を更新します。	-	✔
R_TSIP_Sha*XXX*HmacGenerateInit R_TSIP_Sha*XXX*HmacGenerateUpdate R_TSIP_Sha*XXX*HmacGenerateFinal	SHA-HMAC 生成します。	-	✔
R_TSIP_Sha*XXX*HmacVerifyInit R_TSIP_Sha*XXX*HmacVerifyUpdate R_TSIP_Sha*XXX*HmacVerifyFinal	SHA-HMAC 検証します。	-	✔

表 4-10　DH 演算 API

API	説明	TSIP-Lite	TSIP
R_TSIP_Rsa2048DhKeyAgreement	RSA-2048 による DH 演算を実施します。	-	✔

表 4-11　ECDH 鍵交換 API

API	説明	TSIP-Lite	TSIP
R_TSIP_EcdhP256Init	ECDH P-256 鍵交換演算の準備をします	-	✔
R_TSIP_EcdhP256ReadPublicKey	鍵共有相手の ECC P-256 公開鍵の署名を検証します。	-	✔
R_TSIP_EcdhP256MakePublicKey	ECC P-256 秘密鍵に署名をつけます。	-	✔
R_TSIP_EcdhP256CalculateSharedSecretIndex	鍵共有相手の公開鍵と自分の秘密鍵から、共有秘密 Z を計算します。	-	✔
R_TSIP_EcdhP256KeyDerivation	Z から共有鍵を導出します。	-	✔
R_TSIP_EcdheP512KeyAgreement	Brainpool P512r1 を用いて ECDHE 演算を行います。	-	✔

表 4-12　KeyWrapAPI

API	説明	TSIP-Lite	TSIP
R_TSIP_Aes*XXX*KeyWrap	AES 鍵で、鍵をラップします。	✔	✔
R_TSIP_Aes*XXX*KeyUnwrap	AES 鍵で、鍵をアンラップします。	✔	✔

表 4-13　TLS 機能 API

API	説明	TSIP-Lite	TSIP
R_TSIP_GenerateTls*XX*RsaPublicKeyIndex	TLS 連携で使用する RSA 公開鍵の Wrapped Key を生成します。	-	✔
R_TSIP_UpdateTls*XX*RsaPublicKeyIndex	TLS 連携で使用する RSA 公開鍵の Wrapped Key を更新します。	-	✔
R_TSIP_RegisterCaCertificationPublicKeyIndex	ルート CA 証明書の束を検証するための公開鍵を登録します。	-	✔
R_TSIP_Tls*XX*RootCertificateVerification	ルート CA 証明書の束を検証します。	-	✔
R_TSIP_Tls*XX*CertificateVerification	サーバ証明書、中間証明書の署名を検証します。	-	✔
R_TSIP_Tls*XX*CertificateVerificationExtension	サーバ証明書、中間証明書の署名を検証します。	-	✔
R_TSIP_TlsGeneratePreMasterSecret	暗号化された PreMasterSecret を生成します。	-	✔
R_TSIP_TlsEncryptPreMasterSecretWithRsa2048PublicKey	PreMasterSecret を RSA2048 で暗号化します。	-	✔
R_TSIP_TlsSVGenerateServerRandom	ServerHello で使用する乱数値を生成します。	-	✔
R_TSIP_TlsSVDecryptPreMasterSecretWithRsa2048PrivateKey	PreMasterSecret を RSA2048 で復号します。	-	✔
R_TSIP_Tls*XX*GenerateMasterSecret	暗号化された MasterSecret を生成します。	-	✔
R_TSIP_Tls*XX*GenerateSessionKey	TLS 通信の各種鍵を出力します。	-	✔
R_TSIP_Tls*XX*GenerateVerifyData	Verify データを生成します。	-	✔
R_TSIP_TlsServersEphemeralEcdhPublicKeyRetrieves	ServerKeyExchange の署名を検証します。	-	✔
R_TSIP_GenerateTls*XX*P256EccKeyIndex	TLS 連携機能で使用する乱数から 256bit 素体上の楕円曲線暗号のための鍵ペアを生成します。	-	✔
R_TSIP_Tls*XX*GeneratePreMasterSecretWithEccP256Key	ECC で暗号化された PreMasterSecret を生成します。	-	✔
R_TSIP_Tls*XX*GenerateExtendedMasterSecret	暗号化された ExtendedMasterSecret を生成します。	-	✔
R_TSIP_TlsSVCertificateVerifyVerification	クライアントから受信した CertificateVerify を検証します。	-	✔
R_TSIP_GenerateTls13P256EccKeyIndex	TLS1.3 連携機能で使用する乱数から 256bit 素体上の楕円曲線暗号のための鍵ペアを生成します。	-	✔
R_TSIP_Tls13GenerateEcdheSharedSecret	Shared Secret を生成し、Wrapped Key 形式で出力します。	-	✔
R_TSIP_Tls13GenerateHandshakeSecret	Handshake Secret を生成し、Wrapped Key 形式で出力します。	-	✔

API	説明	TSIP -Lite	TSIP
R_TSIP_Tls13GenerateServerHandshakeTrafficKey	Server Write Key および Server Finished Key の Wrapped Key を生成します。	-	✔
R_TSIP_Tls13ServerHandshakeVerification	サーバから提供される Finished の情報を検証します。	-	✔
R_TSIP_Tls13GenerateClientHandshakeTrafficKey	Client Write Key および Client Finished Key の Wrapped Key を生成します。	-	✔
R_TSIP_Tls13GenerateMasterSecret	Master Secret を生成し、Wrapped Key 形式で出力します。	-	✔
R_TSIP_Tls13GenerateApplicationTrafficKey	Application Traffic Secret と Application Traffic Key の Wrapped Key を生成します。	-	✔
R_TSIP_Tls13UpdateApplicationTrafficKey	Application Traffic Secret と Application Traffic Key の Wrapped Key を更新します。	-	✔
R_TSIP_Tls13GenerateResumptionMasterSecret	Resumption Master Secret の Wrapped Key を生成します。	-	✔
R_TSIP_Tls13GeneratePreSharedKey	Pre Shared Key の Wrapped Key を生成します。	-	✔
R_TSIP_Tls13GeneratePskBinderKey	Binder Key の Wrapped Key を生成します。	-	✔
R_TSIP_Tls13GenerateResumptionHandshakeSecret	Resumption 用の Handshake Secret の Wrapped Key を生成します。	-	✔
R_TSIP_Tls13Generate0RttApplicationWriteKey	0-RTT 用の Client Write Key の Wrapped Key を生成します。	-	✔
R_TSIP_Tls13CertificateVerifyGenerate	サーバに送信する CertificateVerify を生成します。	-	✔
R_TSIP_Tls13CertificateVerifyVerification	サーバから受信した CertificateVerify を検証します。	-	✔
R_TSIP_GenerateTls13SVP256EccKeyIndex	TLS1.3 連携機能で使用する乱数から 256bit 素体上の楕円曲線暗号のための鍵ペアを生成します。	-	✔
R_TSIP_Tls13SVGenerateEcdheSharedSecret	Shared Secret を生成し、Wrapped Key 形式で出力します。	-	✔
R_TSIP_Tls13SVGenerateHandshakeSecret	Handshake Secret を生成し、Wrapped Key 形式で出力します。	-	✔
R_TSIP_Tls13SVGenerateServerHandshakeTrafficKey	Server Write Key および Server Finished Key の Wrapped Key を生成します。	-	✔
R_TSIP_Tls13SVGenerateClientHandshakeTrafficKey	Client Write Key および Client Finished Key の Wrapped Key を生成します。	-	✔
R_TSIP_Tls13SVClientHandshakeVerification	クライアントから提供される Finished の情報を検証します。	-	✔
R_TSIP_Tls13SVGenerateMasterSecret	Master Secret を生成し、Wrapped Key 形式で出力します。	-	✔
R_TSIP_Tls13SVGenerateApplicationTrafficKey	Application Traffic Secret と Application Traffic Key の Wrapped Key を生成します。	-	✔

API	説明	TSIP-Lite	TSIP
R_TSIP_Tls13SVUpdateApplicationTrafficKey	Application Traffic Secret と Application Traffic Key の Wrapped Key を更新します。	-	✔
R_TSIP_Tls13SVGenerateResumptionMasterSecret	Resumption Master Secret を生成し、Wrapped Key 形式で出力します。	-	✔
R_TSIP_Tls13SVGeneratePreSharedKey	Pre Shared Key の Wrapped Key を生成します。	-	✔
R_TSIP_Tls13SVGeneratePskBinderKey	Binder Key の Wrapped Key を生成します。	-	✔
R_TSIP_Tls13SVGenerateResumptionHandshakeSecret	Resumption 用の Handshake Secret の生成し、Wrapped Key 形式で出力します。	-	✔
R_TSIP_Tls13SVGenerate0RttApplicationWriteKey	0-RTT 用の Client Write Key の Wrapped Key を生成します。	-	✔
R_TSIP_Tls13SVCertificateVerifyGenerate	クライアントに送信する CertificateVerify を生成します。	-	✔
R_TSIP_Tls13SVCertificateVerifyVerification	クライアントから受信した CertificateVerify を検証します。	-	✔
R_TSIP_Tls13EncryptInit R_TSIP_Tls13EncryptUpdate R_TSIP_Tls13EncryptFinal	TLS1.3 通信データの暗号化を行います。	-	✔
R_TSIP_Tls13DecryptInit R_TSIP_Tls13DecryptUpdate R_TSIP_Tls13DecryptFinal	TLS1.3 通信データの復号を行います。	-	✔

表 4-14　ファームウェアアップデート API

API	説明	TSIP-Lite	TSIP
R_TSIP_StartUpdateFirmware	ファームウェアアップデートモードに遷移します。	✔	✔
R_TSIP_GenerateFirmwareMAC	暗号化されたファームウェアの復号と MAC 生成を行います。	✔	✔
R_TSIP_VerifyFirmwareMAC	ファームウェアの MAC チェックを行います。	✔	✔

4.2　API 詳細

4.2.1　共通機能 API

4.2.1.1　R_TSIP_Open

Format

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

Parameters

key_index_1	入力	TLS 連携 RSA 公開鍵の Wrapped Key
key_index_2	入力	Wrapped KUK

Return Values

TSIP_SUCCESS	正常終了
TSIP_ERR_FAIL	自己診断が異常終了
TSIP_ERR_RESOURCE_CONFLICT	本処理に必要なハードウェアリソースが他の処理で使用されていることによるリソース衝突が発生
TSIP_ERR_RETRY	自己診断が異常終了 本関数を再実行してください

Description

TSIP 機能を使用可能にします。

key_index_1 には R_TSIP_GenerateTlsRsaPublicKeyIndex()または R_TSIP_UpdateTlsRsaPublicKeyIndex()で生成した「TLS 連携 RSA 公開鍵の Wrapped Key」を入力してください。TLS 連携機能を使用しない場合は NULL ポインタを入力してください。

key_index_2 には R_TSIP_GenerateUpdateKeyRingKeyIndex()で生成した「Wrapped KUK」を入力してください。鍵更新機能を使用しない場合は NULL ポインタを入力してください。

【注】R_TSIP_Open()の実行中に RX がスタンバイモードに遷移することを防ぐため、R_TSIP_Open()内部で割り込み禁止 API の R_BSP_InterruptsDisable()と割り込み許可 API の R_BSP_InterruptsEnable()を呼び出しています。

Reentrant

非対応

4.2.1.2 R_TSIP_Close

Format

#include “r_tsip_rx_if.h”

e_tsip_err_t R_TSIP_Close(void)

Parameters

なし

Return Values

TSIP_SUCCESS	正常終了

Description

TSIP 機能を停止します

Reentrant

非対応

4.2.1.3　R_TSIP_SoftwareReset

Format

```
#include “r_tsip_rx_if.h”
void R_TSIP_SoftwareReset(void)
```

Parameters

なし

Return Values

なし

Description

TSIP を初期状態に戻します。

Reentrant

非対応

4.2.1.4　R_TSIP_GetVersion

Format

```
#include “r_tsip_rx_if.h”
uint32_t R_TSIP_GetVersion(void)
```

Parameters

なし

Return Values

上位 2 バイト:	メジャーバージョン（10 進表示）
下位 2 バイト:	マイナーバージョン（10 進表示）

Description

TSIP ドライバのバージョン情報を取得することができます。

Reentrant

非対応

4.2.1.5 R_TSIP_GenerateUpdateKeyRingKeyIndex

Format

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

Parameters

encrypted_provisioning_key	入力	W-UFPK
iv	入力	encrypted_key 生成時に使用した初期ベクタ
encrypted_key	入力	Encrypted KUK
key_index	出力	Wrapped KUK

Return Values

TSIP_SUCCESS :	正常終了
TSIP_ERR_FAIL:	内部エラーが発生
TSIP_ERR_RESOURCE_CONFLICT:	本処理に必要なハードウェアリソースが他の処理で使用されていることによるリソース衝突が発生

Description

KUK の Wrapped Key を出力するための API です。

encrypted_key には 7.3.7 KUK で示すデータを UFPK で暗号化したデータを入力してください。

encrypted_provisioning_key, iv, encrypted_key の説明、および key_index の使用方法については 3.7.1 鍵の注入と更新 を参照してください。

Reentrant

非対応

4.2.2　乱数生成

4.2.2.1　R_TSIP_GenerateRandomNumber

Format

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

Parameters

random	出力	4 ワード(16 バイト)の乱数値

Return Values

TSIP_SUCCESS :	正常終了
TSIP_ERR_RESOURCE_CONFLICT:	本処理に必要なハードウェアリソースが他の処理で使用されていることによるリソース衝突が発生

Description

NIST SP800-90A に準拠した 4 ワードの乱数値を生成することができます。

Reentrant

非対応

4.2.3 AES

4.2.3.1 R_TSIP_GenerateAes*XXX*KeyIndex

Format

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

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

Parameters

encrypted_provisioning_key	入力	W-UFPK
iv	入力	encrypted_key 生成時に使用した初期ベクタ
encrypted_key	入力	UFPK で暗号化された Encrypted Key
key_index	出力	Wrapped Key

Return Values

TSIP_SUCCESS :	正常終了
TSIP_ERR_FAIL:	内部エラーが発生
TSIP_ERR_RESOURCE_CONFLICT:	本処理に必要なハードウェアリソースが他の処理で使用されていることによるリソース衝突が発生

Description

R_TSIP_GenerateAes128KeyIndex は AES128bit の Wrapped Key を出力するための API です。

R_TSIP_GenerateAes256KeyIndex は AES256bit の Wrapped Key を出力するための API です。

encrypted_key には 7.3.1AES で示すデータを UFPK で暗号化したデータを入力してください。

encrypted_provisioning_key, iv, encrypted_key の説明、および key_index の使用方法については 3.7.1 鍵の注入と更新 を参照してください。

Reentrant

非対応

4.2.3.2 R_TSIP_UpdateAes*XXX*KeyIndex

Format

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

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

Parameters

iv	入力	encrypted_key 生成時に使用した初期ベクタ
encrypted_key	入力	KUK で暗号化された Encrypted Key
key_index	出力	Wrapped Key

Return Values

TSIP_SUCCESS :	正常終了
TSIP_ERR_FAIL:	内部エラーが発生
TSIP_ERR_RESOURCE_CONFLICT:	本処理に必要なハードウェアリソースが他の処理で使用されていることによるリソース衝突が発生

Description

R_TSIP_UpdateAes128KeyIndex は AES128 鍵の Wrapped Key を更新するための API です。

R_TSIP_UpdateAes256KeyIndex は AES256 鍵の Wrapped Key を更新するための API です。

encrypted_key には 7.3.1AES で示すデータを KUK で暗号化したデータを入力してください。

iv, encrypted_key の説明、および key_index の使用方法については 3.7.1 鍵の注入と更新を参照してください。

Reentrant

非対応

4.2.3.3 R_TSIP_GenerateAes*XXX*RandomKeyIndex

Format

(1)
```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_GenerateAes128RandomKeyIndex(
        tsip_aes_key_index_t *key_index
        )
```

(2)
```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_GenerateAes256RandomKeyIndex(
        tsip_aes_key_index_t *key_index
        )
```

Parameters

key_index	出力	(1) AES128 bit の AES の Wrapped Key (2) AES256 bit の AES の Wrapped Key

Return Values

TSIP_SUCCESS :	正常終了
TSIP_ERR_RESOURCE_CONFLICT:	本処理に必要なハードウェアリソースが他の処理で使用されていることによるリソース衝突が発生

Description

R_TSIP_GenerateAes128RandomKeyIndex は AES128 鍵の Wrapped Key を出力するための API です。

R_TSIP_GenerateAes256RandomKeyIndex は AES256 鍵の Wrapped Key を出力するための API です。

本 API は TSIP 内部にて乱数値からユーザ鍵を生成します。従ってユーザ鍵の入力は不要です。API が出力する Wrapped Key を使用しデータを暗号化することにより、データのデッドコピーを防ぐことができます。

key_index の使用方法については 3.7.1 鍵の注入と更新を参照してください。

Reentrant

非対応

4.2.3.4 R_TSIP_Aes*XXX*EcbEncryptInit

Format

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

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

Parameters

handle	出力	AES 用ハンドラ(ワーク領域)
key_index	入力	Wrapped Key

Return Values

TSIP_SUCCESS :	正常終了
TSIP_ERR_FAIL:	内部エラーが発生（TSIP のみ）
TSIP_ERR_RESOURCE_CONFLICT:	本処理に必要なハードウェアリソースが他の処理で使用されていることによるリソース衝突が発生
TSIP_ERR_KEY_SET:	異常な Wrapped Key が入力された

Description

R_TSIP_AesXXXEcbEncryptInit()関数は、AES 演算を実行する準備を行い、その結果を第一引数”handle”に書き出します。handle は、続く R_TSIP_AesXXXEcbEncryptUpdate()関数および R_TSIP_AesXXXEcbEncryptFinal()関数で引数として使用されます。

key_index の生成方法については、3.7.1 鍵の注入と更新を参照してください

Reentrant

非対応

4.2.3.5 R_TSIP_Aes*XXX*EcbEncryptUpdate

Format

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

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

Parameters

handle	入力/出力	AES 用ハンドラ(ワーク領域)
plain	入力	平文データ領域
cipher	出力	暗号文データ領域
plain_length	入力	平文データのバイト長(16 の倍数である必要があります)

Return Values

TSIP_SUCCESS :	正常終了
TSIP_ERR_PARAMETER:	不正なハンドルが入力された
TSIP_ERR_PROHIBIT_FUNCTION:	不正な関数が呼び出された

Description

R_TSIP_AesXXXEcbEncryptUpdate()関数は、第一引数”handle”で指定されたハンドルを使用し、第二引数の”plain”を R_TSIP_AesXXXEcbEncryptInit()関数で指定した key_index を用いて暗号化し、結果を第三引数”cipher”に書き出します。平文入力が完了した後は、R_TSIP_AesXXXEcbEncryptFinal()を呼び出してください。

plain と cipher は、同一アドレスの場合を除き、必ず領域が重ならないように配置してください。

Reentrant

非対応

4.2.3.6 R_TSIP_Aes*XXX*EcbEncrypFinal

Format

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

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

Parameters

handle	入力	AES 用ハンドラ(ワーク領域)
cipher	出力	暗号文データ領域(常に何も書き込まれません)
cipher_length	出力	暗号文データ長(常に 0 が書き込まれます)

Return Values

TSIP_SUCCESS :	正常終了
TSIP_ERR_FAIL:	内部エラーが発生
TSIP_ERR_PARAMETER:	不正なハンドルが入力された
TSIP_ERR_PROHIBIT_FUNCTION:	不正な関数が呼び出された

Description

R_TSIP_AesXXXEcbEncryptFinal()関数は、第一引数”handle”で指定されたハンドルを使用し、第二引数の”cipher”に演算結果、第三引数”cipher_length”に演算結果の長さを書き出します。第二引数は、本来は 16 バイトの倍数に満たない分の端数について暗号化した結果が書き出されますが、R_TSIP_AesXXXEcbEncryptUpdate()関数には 16 バイトの倍数でしか入力できない制限があるため、cipher には常に何も書き込まれず、cipher_length には常に 0 が書き込まれます。cipher, cipher_length は将来この制限が解除された際の互換性のための引数です。

Reentrant

非対応

4.2.3.7 R_TSIP_Aes*XXX*EcbDecryptInit

Format

(1)
```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_Aes128EcbDecryptInit(
        tsip_aes_handle_t *handle,
        tsip_aes_key_index_t *key_index
)
```

(2)
```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_Aes256EcbDecryptInit(
        tsip_aes_handle_t *handle,
        tsip_aes_key_index_t *key_index
)
```

Parameters

handle	出力	AES 用ハンドラ(ワーク領域)
key_index	入力	Wrapped Key

Return Values

TSIP_SUCCESS :	正常終了
TSIP_ERR_FAIL:	内部エラーが発生（TSIP ドライバのみ）
TSIP_ERR_RESOURCE_CONFLICT:	本処理に必要なハードウェアリソースが他の処理で使用されていることによるリソース衝突が発生
TSIP_ERR_KEY_SET:	異常な Wrapped Key が入力された

Description

R_TSIP_AesXXXEcbDecryptInit()関数は、AES 演算を実行する準備を行い、その結果を第一引数”handle”に書き出します。handle は、続く R_TSIP_AesXXXEcbDecryptUpdate()関数および R_TSIP_AesXXXEcbDecryptFinal()関数で引数として使用されます。

key_index の生成方法については、3.7.1 鍵の注入と更新を参照してください。

Reentrant

非対応

4.2.3.8 R_TSIP_Aes*XXX*EcbDecryptUpdate

Format

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

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

Parameters

handle	入力/出力	AES 用ハンドラ(ワーク領域)
cipher	入力	暗号文データ領域
plain	出力	平文データ領域
cipher_length	入力	暗号文データのバイト長(16 の倍数である必要があります)

Return Values

TSIP_SUCCESS :	正常終了
TSIP_ERR_PARAMETER:	不正なハンドルが入力された
TSIP_ERR_PROHIBIT_FUNCTION:	不正な関数が呼び出された

Description

R_TSIP_AesXXXEcbDecryptUpdate()関数は、第一引数“handle“で指定されたハンドルを使用し、第二引数の“cipher“を R_TSIP_AesXXXEcbDecryptInit()関数で指定した key_index を用いて復号し、結果を第三引数“plain“に書き出します。暗号文入力が完了した後は、R_TSIP_AesXXXEcbDecryptFinal()を呼び出してください。

plain と cipher は、同一アドレスの場合を除き、必ず領域が重ならないように配置してください。

Reentrant

非対応

4.2.3.9 R_TSIP_Aes*XXX*EcbDecryptFinal

Format

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

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

Parameters

handle	入力	AES 用ハンドラ(ワーク領域)
plain	出力	平文データ領域(常に何も書き込まれません)
plain_length	出力	平文データ長(常に 0 が書き込まれます)

Return Values

TSIP_SUCCESS :	正常終了
TSIP_ERR_FAIL:	内部エラーが発生
TSIP_ERR_PARAMETER:	不正なハンドルが入力された
TSIP_ERR_PROHIBIT_FUNCTION:	不正な関数が呼び出された

Description

R_TSIP_AesXXXEcbDecryptFinal()関数は、第一引数”handle”で指定されたハンドルを使用し、第二引数の”plain”に演算結果、第三引数”plain_length”に演算結果の長さを書き出します。第二引数は、本来は 16 バイトの倍数に満たない分の端数について復号した結果が書き出されますが、R_TSIP_AesXXXEcbDecryptUpdate()関数には 16 バイトの倍数でしか入力できない制限があるため、plain には常に何も書き込まれず、plain_length には常に 0 が書き込まれます。plain, plain_length は将来この制限が解除された際の互換性のための引数です。

Reentrant

非対応

4.2.3.10 R_TSIP_Aes*XXX*CbcEncryptInit

Format

(1)
```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_Aes128CbcEncryptInit(
        tsip_aes_handle_t *handle,
        tsip_aes_key_index_t *key_index,
        uint8_t *ivec
)
```

(2)
```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_Aes256CbcEncryptInit(
        tsip_aes_handle_t *handle,
        tsip_aes_key_index_t *key_index,
        uint8_t *ivec
)
```

Parameters

handle	出力	AES 用ハンドラ(ワーク領域)
key_index	入力	Wrapped Key
ivec	入力	初期化ベクタ(16 バイト)

Return Values

TSIP_SUCCESS :	正常終了
TSIP_ERR_FAIL:	内部エラーが発生（TSIP ドライバのみ）
TSIP_ERR_RESOURCE_CONFLICT:	本処理に必要なハードウェアリソースが他の処理で使用されていることによるリソース衝突が発生
TSIP_ERR_KEY_SET:	異常な Wrapped Key が入力された

Description

R_TSIP_AesXXXCbcEncryptInit()関数は、AES 演算を実行する準備を行い、その結果を第一引数”handle”に書き出します。handle は、続く R_TSIP_AesXXXCbcEncryptUpdate()関数および R_TSIP_AesXXXCbcEncryptFinal()関数で引数として使用されます。

TLS 連携機能で使用する場合、key_index には R_TSIP_TlsGenerateSessionKey()で生成された client_crypto_key_index もしくは server_crypto_key_index を入力してください。

key_index の生成方法については、3.7.1 鍵の注入と更新を参照してください。

Reentrant

非対応

4.2.3.11 R_TSIP_Aes*XXX*CbcEncryptUpdate

Format

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

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

Parameters

handle	入力/出力	AES 用ハンドラ(ワーク領域)
plain	入力	平文データ領域
cipher	出力	暗号文データ領域
plain_length	入力	平文データのバイト長(16 の倍数である必要があります)

Return Values

TSIP_SUCCESS :	正常終了
TSIP_ERR_PARAMETER:	不正なハンドルが入力された
TSIP_ERR_PROHIBIT_FUNCTION:	不正な関数が呼び出された

Description

R_TSIP_AesXXXCbcEncryptUpdate()関数は、第一引数”handle”で指定されたハンドルを使用し、第二引数の”plain”を R_TSIP_AesXXXCbcEncryptInit()関数で指定した key_index を用いて暗号化し、結果を第三引数”cipher”に書き出します。平文入力が完了した後は、R_TSIP_AesXXXCbcEncryptFinal()を呼び出してください。

plain と cipher は、同一アドレスの場合を除き、必ず領域が重ならないように配置してください。

Reentrant

非対応

4.2.3.12 R_TSIP_Aes*XXX*CbcEncryptFinal

Format

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

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

Parameters

handle	入力	AES 用ハンドラ(ワーク領域)
cipher	出力	暗号文データ領域(常に何も書き込まれません)
cipher_length	出力	暗号文データ長(常に 0 が書き込まれます)

Return Values

TSIP_SUCCESS :	正常終了
TSIP_ERR_FAIL:	内部エラーが発生
TSIP_ERR_PARAMETER:	不正なハンドルが入力された
TSIP_ERR_PROHIBIT_FUNCTION:	不正な関数が呼び出された

Description

R_TSIP_AesXXXCbcEncryptFinal()関数は、第一引数”handle”で指定されたハンドルを使用し、第二引数の”cipher”に演算結果、第三引数”cipher_length”に演算結果の長さを書き出します。第二引数は、本来は 16 バイトの倍数に満たない分の端数について暗号化した結果が書き出されますが、R_TSIP_AesXXXCbcEncryptUpdate()関数には 16 バイトの倍数でしか入力できない制限があるため、cipher には常に何も書き込まれず、cipher_length には常に 0 が書き込まれます。cipher, cipher_length は将来この制限が解除された際の互換性のための引数です。

Reentrant

非対応

4.2.3.13 R_TSIP_Aes*XXX*CbcDecryptInit

Format

(1)
```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_Aes128CbcDecryptInit(
        tsip_aes_handle_t *handle,
        tsip_aes_key_index_t *key_index,
        uint8_t *ivec
)
```

(2)
```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_Aes256CbcDecryptInit(
        tsip_aes_handle_t *handle,
        tsip_aes_key_index_t *key_index,
        uint8_t *ivec
)
```

Parameters

handle	出力	AES 用ハンドラ(ワーク領域)
key_index	入力	Wrapped Key
ivec	入力	初期化ベクタ(16 バイト)

Return Values

TSIP_SUCCESS :	正常終了
TSIP_ERR_FAIL:	内部エラーが発生（TSIP ドライバのみ）
TSIP_ERR_RESOURCE_CONFLICT:	本処理に必要なハードウェアリソースが他の処理で使用されていることによるリソース衝突が発生
TSIP_ERR_KEY_SET:	異常な Wrapped Key が入力された

Description

R_TSIP_AesXXXCbcDecryptInit()関数は、AES 演算を実行する準備を行い、その結果を第一引数”handle”に書き出します。handle は、続く R_TSIP_AesXXXCbcDecryptUpdate()関数および R_TSIP_AesXXXCbcDecryptFinal()関数で引数として使用されます。

TLS 連携機能で使用する場合、key_index には R_TSIP_TlsGenerateSessionKey()で生成された client_crypto_key_index もしくは server_crypto_key_index を入力してください。

key_index の生成方法については、3.7.1 鍵の注入と更新を参照してください

Reentrant

非対応

4.2.3.14 R_TSIP_Aes*XXX*CbcDecryptUpdate

Format

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

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

Parameters

handle	入力/出力	AES 用ハンドラ(ワーク領域)
cipher	入力	暗号文データ領域
plain	出力	平文データ領域
cipher_length	入力	暗号文データのバイト長(16 の倍数である必要があります)

Return Values

TSIP_SUCCESS :	正常終了
TSIP_ERR_PARAMETER:	不正なハンドルが入力された
TSIP_ERR_PROHIBIT_FUNCTION:	不正な関数が呼び出された

Description

R_TSIP_AesXXXCbcDecryptUpdate()関数は、第一引数”handle”で指定されたハンドルを使用し、第二引数の”cipher”を R_TSIP_AesXXXCbcDecryptInit()関数で指定した key_index を用いて復号し、結果を第三引数”plain”に書き出します。暗号文入力が完了した後は、R_TSIP_AesXXXCbcDecryptFinal()を呼び出してください。

plain と cipher は、同一アドレスの場合を除き、必ず領域が重ならないように配置してください。

Reentrant

非対応

4.2.3.15 R_TSIP_Aes*XXX*CbcDecryptFinal

Format

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

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

Parameters

handle	入力	AES 用ハンドラ(ワーク領域)
plain	出力	平文データ領域(常に何も書き込まれません)
plain_length	出力	平文データ長(常に 0 が書き込まれます)

Return Values

TSIP_SUCCESS :	正常終了
TSIP_ERR_FAIL:	内部エラーが発生
TSIP_ERR_PARAMETER:	不正なハンドルが入力された
TSIP_ERR_PROHIBIT_FUNCTION:	不正な関数が呼び出された

Description

R_TSIP_AesXXXCbcDecryptFinal()関数は、第一引数”handle”で指定されたハンドルを使用し、第二引数の”plain”に演算結果、第三引数”plain_length”に演算結果の長さを書き出します。第二引数は、本来は 16 バイトの倍数に満たない分の端数について復号した結果が書き出されますが、R_TSIP_AesXXXCbcDecryptUpdate()関数には 16 バイトの倍数でしか入力できない制限があるため、plain には常に何も書き込まれず、plain_length には常に 0 が書き込まれます。plain, plain_length は将来この制限が解除された際の互換性のための引数です。

Reentrant

非対応

4.2.3.16 R_TSIP_Aes*XXX*CtrInit

Format

(1)
```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_Aes128CtrInit(
        tsip_aes_handle_t *handle,
        tsip_aes_key_index_t *key_index,
        uint8_t *ictr
)
```

(2)
```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_Aes256CtrInit(
        tsip_aes_handle_t *handle,
        tsip_aes_key_index_t *key_index,
        uint8_t *ictr
)
```

Parameters

handle	出力	AES 用ハンドラ(ワーク領域)
key_index	入力	Wrapped Key
ictr	入力	初期カウンタ(16 バイト)

Return Values

TSIP_SUCCESS :	正常終了
TSIP_ERR_FAIL:	内部エラーが発生（TSIP ドライバのみ）
TSIP_ERR_RESOURCE_CONFLICT:	本処理に必要なハードウェアリソースが他の処理で使用されていることによるリソース衝突が発生
TSIP_ERR_KEY_SET:	異常な Wrapped Key が入力された

Description

本関数は、AES 演算を実行する準備を行い、その結果を引数”handle”に書き出します。handle は、続く R_TSIP_AesXXXCtrUpdate()関数および R_TSIP_AesXXXCtrFinal()関数で引数として使用されます。

key_index の生成方法については、3.7.1 鍵の注入と更新を参照してください

Reentrant

非対応

4.2.3.17 R_TSIP_Aes*XXX*CtrUpdate

Format

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

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

Parameters

handle	入力/出力	AES 用ハンドラ(ワーク領域)
itext	入力	入力文(平文または暗号文)データ領域
otext	出力	出力文(暗号文または平文)データ領域
itext_length	入力	入力文データのバイト長(16 の倍数である必要があります)

Return Values

TSIP_SUCCESS :	正常終了
TSIP_ERR_PARAMETER:	不正なハンドルが入力された
TSIP_ERR_PROHIBIT_FUNCTION:	不正な関数が呼び出された

Description

本関数は、引数”handle”で指定されたハンドルを使用し、引数”itext”を R_TSIP_AesXXXCtrInit()関数で指定した key_index を用いて暗号化し、結果を引数”otext”に書き出します。最終ブロックの入力完了後に、R_TSIP_AesXXXCtrFinal()を呼び出してください。最終ブロック長が 1～127 ビットの場合でも、itext および otext は 16 バイト単位の領域を確保し、itext の端数領域には任意の値を設定してください。またその場合、otext の端数領域に格納された値は読み捨ててください。

itext と otext は、同一アドレスの場合を除き、必ず領域が重ならないように配置してください。

Reentrant

非対応

4.2.3.18 R_TSIP_Aes*XXX*CtrFinal

Format

(1)
```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_Aes128CtrFinal(
        tsip_aes_handle_t *handle
        )
```

(2)
```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_Aes256CtrFinal(
        tsip_aes_handle_t *handle
        )
```

Parameters

handle	入力	AES 用ハンドラ(ワーク領域)

Return Values

TSIP_SUCCESS :	正常終了
TSIP_ERR_FAIL:	内部エラーが発生
TSIP_ERR_PARAMETER:	不正なハンドルが入力された
TSIP_ERR_PROHIBIT_FUNCTION:	不正な関数が呼び出された

Description

本関数は、引数”handle”で指定されたハンドルを使用し、演算を終了します。

Reentrant

非対応

4.2.3.19 R_TSIP_Aes*XXX*GcmEncryptInit

Format

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

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

Parameters

handle	出力	AES-GCM 用ハンドラ(ワーク領域)
key_index	入力	Wrapped Key
ivec	入力	初期化ベクタ領域(iv_len byte) 【注】
ivec_len	入力	初期化ベクタ長(1～任意 byte)

Return Values

TSIP_SUCCESS :	正常終了
TSIP_ERR_FAIL:	内部エラーが発生
TSIP_ERR_RESOURCE_CONFLICT:	本処理に必要なハードウェアリソースが他の処理で使用されていることによるリソース衝突が発生
TSIP_ERR_KEY_SET:	異常な Wrapped Key が入力された
TSIP_ERR_PARAMETER:	入力データが不正

Description

R_TSIP_AesXXXGcmEncryptInit()関数は、GCM 演算を実行する準備を行い、その結果を第一引数”handle”に書き出します。handle は、続く R_TSIP_AesXXX 1 GcmEncryptUpdate()関数および R_TSIP_AesXXXGcmEncryptFinal()関数で引数として使用されます。

【注】

key_index->type が”TSIP_KEY_INDEX_TYPE_AES128_FOR_TLS“の場合

R_TSIP_TlsGenerateSessionKey ()関数で select_cipher:6, 7 を指定して生成した key_index は、96bit の IV を含んでいます。第三引数の ivec には NULL ポインタを入力してください。第四引数の ivec_len に 0 を指定してください。

key_index の生成方法については、3.7.1 鍵の注入と更新を参照してください

Reentrant

非対応

4.2.3.20 R_TSIP_Aes*XXX*GcmEncryptUpdate

Format

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

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

Parameters

handle	入力/出力	AES 用ハンドラ(ワーク領域)
plain	入力	平文データ領域
cipher	出力	暗号文データ領域
plain_data_len	入力	平文データのバイト長(16 の倍数である必要があります)
aad	入力	追加認証データ (aad_len byte)
aad_len	入力	追加認証データ長(0～任意 byte)

Return Values

TSIP_SUCCESS :	正常終了
TSIP_ERR_PARAMETER:	不正なハンドルが入力された
TSIP_ERR_PROHIBIT_FUNCTION:	不正な関数が呼び出された

Description

R_TSIP_Aes128GcmEncryptUpdate()関数は、第二引数”plain”で指定された平文から R_TSIP_Aes128GcmEncryptInit()で指定された”key_index”と”ivec”、第五引数で指定された”aad”を用いて GCM で暗号化します。本関数内部で、aad, plain の入力値が 16byte を超えるまでユーザが入力したデータをバッファリングします。暗号化結果は”plain”入力データが 16byte 以上になってから、第三引数で指定された”cipher”に出力します。入力する”plain”, “aad”データ長はそれぞれ第四引数の”plain_data_len”, 第六引数の”aad_len”で指定します。ここでは、”aad”, “plain”入力データの総バイト数ではなく、ユーザが本関数を呼ぶ際に入力するデータ長を指定してください。入力値の”plain”および”aad”は 16byte で割り切れない場合、パディング処理は関数内部で実施します。データの入力は”aad”, “plain”の順で処理してください。”plain”データ入力開始後、”aad”データを入力するとエラーとなります。”aad”データと”plain”データが同時に本関数に入力された場合、”aad”データ処理後、”plain”データ入力状態に移行します。plain と cipher は、同一アドレスの場合を除き、必ず領域が重ならないように配置してください。

Reentrant

非対応

4.2.3.21 R_TSIP_Aes*XXX*GcmEncryptFinal

Format

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

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

Parameters

handle	入力	AES 用ハンドラ(ワーク領域)
cipher	出力	暗号文データ領域(常に何も書き込まれません)
cipher_data_len	出力	暗号文データ長(常に 0 が書き込まれます)
atag	出力	認証タグ領域(16byte)

Return Values

TSIP_SUCCESS :	正常終了
TSIP_ERR_FAIL:	内部エラーが発生
TSIP_ERR_PARAMETER:	不正なハンドルが入力された
TSIP_ERR_PROHIBIT_FUNCTION:	不正な関数が呼び出された

Description

R_TSIP_AesXXXGcmEncryptFinal()関数は、R_TSIP_AesXXXGcmEncryptUpdate()で入力した plain の総データ長に 16byte の端数データがある場合、第二引数で指定された”cipher”に端数分の暗号化したデータを出力します。このとき、16byte に満たない部分は 0 padding されています。認証タグは第四引数の”atag”に出力します。

Reentrant

非対応

4.2.3.22 R_TSIP_Aes*XXX*GcmDecryptInit

Format

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

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

Parameters

handle	出力	AES 用ハンドラ(ワーク領域)
key_index	入力	Wrapped Key
ivec	入力	初期化ベクタ領域(iv_len byte) 【注】
ivec_len	入力	初期化ベクタ長(1～任意 byte)

Return Values

TSIP_SUCCESS :	正常終了
TSIP_ERR_FAIL:	内部エラーが発生
TSIP_ERR_RESOURCE_CONFLICT:	本処理に必要なハードウェアリソースが他の処理で使用されていることによるリソース衝突が発生
TSIP_ERR_KEY_SET:	異常な Wrapped Key が入力された
TSIP_ERR_PARAMETER:	入力データが不正

Description

R_TSIP_AesXXXGcmDecryptInit()関数は、GCM 演算を実行する準備を行い、その結果を第一引数”handle”に書き出します。handle は、続く R_TSIP_AesXXXGcmDecryptUpdate()関数および R_TSIP_AesXXXGcmDecryptFinal()関数で引数として使用されます。

【注】

key_index->type が”TSIP_KEY_INDEX_TYPE_AES128_FOR_TLS“の場合

R_TSIP_TlsGenerateSessionKey ()関数で select_cipher:6, 7 を指定して生成した key_index は、96bit の IV を含んでいます。第三引数の ivec には NULL ポインタを入力してください。第四引数の ivec_len に 0 を指定してください。

key_index の生成方法については、3.7.1 鍵の注入と更新を参照してください

Reentrant

非対応

4.2.3.23 R_TSIP_Aes*XXX*GcmDecryptUpdate

Format

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

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

Parameters

handle	入力/出力	AES-GCM 用ハンドラ(ワーク領域)
cipher	入力	暗号文データ領域
plain	出力	平文データ領域
cipher_data_len	入力	暗号文データ長(0～任意 byte)
aad	入力	追加認証データ (aad_len byte)
aad_len	入力	追加認証データ長(0～任意 byte)

Return Values

TSIP_SUCCESS :	正常終了
TSIP_ERR_PARAMETER:	不正なハンドルが入力された
TSIP_ERR_PROHIBIT_FUNCTION:	不正な関数が呼び出された

Description

R_TSIP_AesXXXGcmDecryptUpdate()関数は、第二引数”cipher”で指定された暗号文から R_TSIP_AesXXXGcmDecryptInit()で指定された”key_index”と”ivec”、第五引数で指定された”aad”を用いて GCM で復号します。本関数内部で、aad, cipher の入力値が 16byte を超えるまでユーザが入力したデータをバッファリングします。復号結果は”cipher”入力データが 16byte 以上になってから、第三引数で指定された”plain”に出力します。入力する”cipher”, “aad”データ長はそれぞれ第四引数の”cipher_data_len”, 第六引数の”aad_len”で指定します。ここでは、”aad”, “cipher”入力データの総バイト数ではなく、ユーザが本関数を呼ぶ際に入力するデータ長を指定してください。入力値の”cipher”および”aad”は 16byte で割り切れない場合、パディング処理は関数内部で実施します。本関数の内部では、“aad”データ入力状態の後に“cipher”データ入力状態に遷移します。”aad”データと”cipher”データを同時に本関数に入力することは可能ですが、“cipher”データを入力する際には”aad”データを最後まで入力する必要があります。plain と cipher は、同一アドレスの場合を除き、必ず領域が重ならないように配置してください。

Reentrant

非対応

4.2.3.24 R_TSIP_Aes*XXX*GcmDecryptFinal

Format

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

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

Parameters

handle	入力	AES-GCM 用ハンドラ(ワーク領域)
plain	出力	平文データ領域(data_len byte)
plain_data_len	出力	平文データ長(0～任意 byte)
atag	入力	認証タグ領域(atag_len byte)
atag_len	入力	認証タグ長(4,8,12,13,14,15,16byte)

Return Values

TSIP_SUCCESS :	正常終了
TSIP_ERR_FAIL:	内部エラーが発生
TSIP_ERR_AUTHENTICATION :	認証エラー
TSIP_ERR_PROHIBIT_FUNCTION:	不正な関数が呼び出された
TSIP_ERR_PARAMETER:	入力データが不正

Description

R_TSIP_AesXXXGcmDecryptFinal()関数は、R_TSIP_AesXXXGcmDecryptUpdate()で指定された 16byte に満たない端数の暗号文を GCM で復号し、GCM 復号機能を終了させます。復号データ、認証タグはそれぞれ第二引数で指定された”plain”および、第四引数の”atag”に出力します。復号された総データ長は第三引数の”plain_data_len”に出力します。認証に失敗した場合は、戻り値 TSIP_ERR_AUTHENTICATION が返ります。第四引数で指定する”atag”は 16byte 以下で入力してください。16byte に満たない場合は、本関数内で 0padding を実施します。

Reentrant

非対応

4.2.3.25 R_TSIP_Aes*XXX*CcmEncryptInit

Format

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

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

Parameters

handle	出力	AES-CCM 用ハンドラ(ワーク領域)
key_index	入力	Wrapped Key
nonce	入力	ノンス
nonce_len	入力	ノンスデータ長(7~13 byte)
adata	入力	追加認証データ
a_len	入力	追加認証データ長(0~110byte)
payload_len	入力	ペイロード長(任意 byte)
mac_len	入力	MAC 長(4, 6, 8, 10, 12, 14, 16 byte)

Return Values

TSIP_SUCCESS :	正常終了
TSIP_ERR_FAIL:	内部エラーが発生
TSIP_ERR_RESOURCE_CONFLICT:	本処理に必要なハードウェアリソースが他の処理で使用されていることによるリソース衝突が発生
TSIP_ERR_KEY_SET:	異常な Wrapped Key が入力された

Description

R_TSIP_AesXXXCcmEncryptInit()関数は、CCM 演算を実行する準備を行い、その結果を第一引数“handle“に書き出します。handle は、続く R_TSIP_AesXXXCcmEncryptUpdate()関数および R_TSIP_AesXXXCcmEncryptFinal()関数で引数として使用されます。

key_index の生成方法については、3.7.1 鍵の注入と更新を参照してください。

Reentrant

非対応

4.2.3.26 R_TSIP_Aes*XXX*CcmEncryptUpdate

Format

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

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

Parameters

handle	入力/出力	AES 用ハンドラ(ワーク領域)
plain	入力	平文データ領域
cipher	出力	暗号文データ領域
plain_length	入力	平文データ長

Return Values

TSIP_SUCCESS :	正常終了
TSIP_ERR_PARAMETER:	不正なハンドルが入力された
TSIP_ERR_PROHIBIT_FUNCTION:	不正な関数が呼び出された

Description

R_TSIP_AesXXXCcmEncryptUpdate()関数は、第二引数“plain“で指定された平文から R_TSIP_AesXXXCcmEncryptInit()で指定された“key_index“, “nonce“, “adata“を用いて CCM を用いて暗号化します。本関数内部で plain の入力値が 16byte を超えるまでユーザが入力したデータをバッファリングします。暗号化結果は“plain“入力データが 16byte 以上になってから、第三引数で指定された“cipher“に出力します。入力する plain の総データ長は R_TSIP_AesXXXCcmEncryptInit()の payload_len で指定してください。本関数の plain_length には、ユーザが本関数を呼ぶ際に入力するデータ長を指定してください。入力値の plain は 16byte で割り切れない場合、パディング処理は関数内部で実施します。plain と cipher は、同一アドレスの場合を除き、必ず領域が重ならないように配置してください。

Reentrant

非対応

4.2.3.27 R_TSIP_Aes*XXX*CcmEncryptFinal

Format

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

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

Parameters

handle	入力	AES 用ハンドラ(ワーク領域)
cipher	出力	暗号文データ領域(常に何も書き込まれません)
cipher_length	出力	暗号文データ長(常に 0 が書き込まれます)
mac	出力	MAC 領域
mac_length	入力	MAC 長(4, 6, 8, 10, 12, 14, 16 byte)

Return Values

TSIP_SUCCESS :	正常終了
TSIP_ERR_FAIL:	内部エラーが発生
TSIP_ERR_PARAMETER:	不正なハンドルが入力された
TSIP_ERR_PROHIBIT_FUNCTION:	不正な関数が呼び出された

Description

R_TSIP_Aes*XXX*CcmEncryptFinal()関数は、R_TSIP_Aes*XXX*CcmEncryptUpdate()で入力した plain のデータ長に 16byte の端数データがある場合、第二引数で指定された“cipher"に端数分の暗号化したデータを出力します。MAC 値は第四引数の“mac“に出力します。第五引数の“mac_length“には、R_TSIP_Aes*XXX*CcmEncryptInit()の引数“mac_length“と同じ値を指定してください。

Reentrant

非対応

4.2.3.28 R_TSIP_Aes*XXX*CcmDecryptInit

Format

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

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

Parameters

handle	入力	AES-CCM 用ハンドラ(ワーク領域)
key_index	入力	Wrapped Key
nonce	入力	ノンス
nonce_len	入力	ノンスデータ長(7~13byte)
adata	入力	追加認証データ
a_len	入力	追加認証データ長(0~110byte)
payload_len	入力	ペイロード長(任意 byte)
mac_len	入力	MAC 長(4, 6, 8, 10, 12, 14, 16 byte)

Return Values

TSIP_SUCCESS :	正常終了
TSIP_ERR_FAIL:	内部エラーが発生
TSIP_ERR_RESOURCE_CONFLICT:	本処理に必要なハードウェアリソースが他の処理で使用されていることによるリソース衝突が発生
TSIP_ERR_KEY_SET:	異常な Wrapped Key が入力された

Description

R_TSIP_Aes*XXX*CcmDecryptInit()関数は、CCM 演算を実行する準備を行い、その結果を第一引数“handle“に書き出します。handle は、続く R_TSIP_Aes*XXX*CcmDecryptUpdate()関数および R_TSIP_Aes*XXX*CcmDecryptFinal()関数で引数として使用されます。

key_index の生成方法については、3.7.1 鍵の注入と更新を参照してください

Reentrant

非対応

4.2.3.29 R_TSIP_Aes*XXX*CcmDecryptUpdate

Format

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

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

Parameters

handle	入力/出力	AES-CCM 用ハンドラ(ワーク領域)
cipher	入力	暗号文データ領域
plain	出力	平文データ領域
cipher_length	入力	暗号文データ長

Return Values

TSIP_SUCCESS :	正常終了
TSIP_ERR_PARAMETER:	不正なハンドルが入力された
TSIP_ERR_PROHIBIT_FUNCTION:	不正な関数が呼び出された

Description

R_TSIP_AesXXXCcmDecryptUpdate()関数は、第二引数“cipher“で指定された暗号文から R_TSIP_AesXXXCcmDecryptInit()で指定された“key_index“, “nonce“, “adata“を用いて CCM を用いて復号します。本関数内部で cipher の入力値が 16byte を超えるまでユーザが入力したデータをバッファリングします。復号結果は“cipher“入力データが 16byte 以上になってから、第三引数で指定された“plain“に出力します。入力する cipher の総データ長は R_TSIP_AesXXXCcmDecryptInit()の payload_len で指定してください。本関数の cipher_length には、ユーザが本関数を呼ぶ際に入力するデータ長を指定してください。入力値の cipher は 16byte で割り切れない場合、パディング処理は関数内部で実施します。plain と cipher は、同一アドレスの場合を除き、必ず領域が重ならないように配置してください。

Reentrant

非対応

4.2.3.30 R_TSIP_Aes*XXX*CcmDecryptFinal

Format

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

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

Parameters

handle	入力	AES-GCM 用ハンドラ(ワーク領域)
plain	出力	平文データ領域
plain_length	出力	平文データ長
mac	入力	MAC 領域
mac_length	入力	MAC 長(4, 6, 8, 10, 12, 14, 16 byte)

Return Values

TSIP_SUCCESS :	正常終了
TSIP_ERR_FAIL:	内部エラーが発生
TSIP_ERR_PARAMETER:	不正なハンドルが入力された
TSIP_ERR_PROHIBIT_FUNCTION:	不正な関数が呼び出された

Description

R_TSIP_AesXXXCcmDecryptFinal()関数は、R_TSIP_AesXXXCcmDecryptUpdate()で入力した cipher のデータ長に 16byte の端数データがある場合、第二引数で指定された“cipher“に端数分の復号したデータを出力します。また、第四引数の“mac“を検証します。第五引数の“mac_length“には、R_TSIP_AesXXXCcmDecryptInit()の引数“mac_length“と同じ値を指定してください。

Reentrant

非対応

4.2.3.31 R_TSIP_Aes*XXX*CmacGenerateInit

Format

(1) #include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_Aes128CmacGenerateInit(
tsip_cmac_handle_t *handle,
tsip_aes_key_index_t *key_index
)

(2) #include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_Aes256CmacGenerateInit(
tsip_cmac_handle_t *handle,
tsip_aes_key_index_t *key_index
)

Parameters

handle	出力	AES-CMAC 用ハンドラ(ワーク領域)
key_index	入力	Wrapped Key

Return Values

TSIP_SUCCESS :	正常終了
TSIP_ERR_FAIL:	内部エラーが発生
TSIP_ERR_RESOURCE_CONFLICT:	本処理に必要なハードウェアリソースが他の処理で使用されていることによるリソース衝突が発生
TSIP_ERR_KEY_SET:	異常な Wrapped Key が入力された

Description

R_TSIP_AesXXXCmacGenerateInit()関数は、CMAC 演算を実行する準備を行い、その結果を第一引数”handle”に書き出します。handle は続く R_TSIP_AesXXXCmacGenerateUpdate()関数や、R_TSIP_AesXXXCmacGenerateFinal()関数の引数で使用します。

key_index の生成方法については、3.7.1 鍵の注入と更新を参照してください

Reentrant

非対応

4.2.3.32 R_TSIP_Aes*XXX*CmacGenerateUpdate

Format

(1)
```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_Aes128CmacGenerateUpdate(
        tsip_cmac_handle_t *handle,
        uint8_t *message,
        uint32_t message_length
)
```

(2)
```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_Aes256CmacGenerateUpdate(
        tsip_cmac_handle_t *handle,
        uint8_t *message,
        uint32_t message_length
)
```

Parameters

handle	入力/出力	AES-CMAC 用ハンドラ(ワーク領域)
message	入力	メッセージデータ領域(message_length byte)
message_length	入力	メッセージデータ長(0～任意 byte)

Return Values

TSIP_SUCCESS :	正常終了
TSIP_ERR_PARAMETER:	不正なハンドルが入力された
TSIP_ERR_PROHIBIT_FUNCTION:	不正な関数が呼び出された

Description

R_TSIP_AesXXXCmacGenerateUpdate()関数は、第二引数”message”で指定された message から R_TSIP_AesXXXCmacGenerateInit()で指定された”key_index”を用いて MAC 値を生成します。本関数内部で、”message”の入力値が 16byte を超えるまでユーザが入力したデータをバッファリングします。入力する”message”データ長は第三引数の”message_len”で指定します。ここでは、”message”入力データの総バイト数ではなく、ユーザが本関数を呼ぶ際に入力するメッセージのデータ長を入力してください。入力値の”message”は 16byte で割り切れない場合、パディング処理は関数内部で実施します。

Reentrant

非対応

4.2.3.33 R_TSIP_Aes*XXX*CmacGenerateFinal

Format

(1)
```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_Aes128CmacGenerateFinal(
        tsip_cmac_handle_t *handle,
        uint8_t *mac
)
```

(2)
```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_Aes256CmacGenerateFinal(
        tsip_cmac_handle_t *handle,
        uint8_t *mac
)
```

Parameters

handle	入力	AES-CMAC 用ハンドラ(ワーク領域)
mac	出力	MAC データ領域(16byte)

Return Values

TSIP_SUCCESS :	正常終了
TSIP_ERR_FAIL:	内部エラーが発生
TSIP_ERR_PARAMETER:	不正なハンドルが入力された
TSIP_ERR_PROHIBIT_FUNCTION:	不正な関数が呼び出された

Description

R_TSIP_AesXXXCmacGenerateFinal()関数は、第二引数で指定された”mac”に Mac 値を出力し、CMAC の動作を終了させます。

Reentrant

非対応

4.2.3.34 R_TSIP_Aes*XXX*CmacVerifyInit

Format

(1)
```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_Aes128CmacVerifyInit(
        tsip_cmac_handle_t *handle,
        tsip_aes_key_index_t *key_index
)
```

(2)
```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_Aes256CmacVerifyInit(
        tsip_cmac_handle_t *handle,
        tsip_aes_key_index_t *key_index
)
```

Parameters

handle	出力	AES-CMAC 用ハンドラ(ワーク領域)
key_index	入力	Wrapped Key

Return Values

TSIP_SUCCESS :	正常終了
TSIP_ERR_FAIL:	内部エラーが発生
TSIP_ERR_RESOURCE_CONFLICT:	本処理に必要なハードウェアリソースが他の処理で使用されていることによるリソース衝突が発生
TSIP_ERR_KEY_SET:	異常な Wrapped Key が入力された

Description

R_TSIP_AesXXXCmacVerifyInit()関数は、CMAC 演算を実行する準備を行い、その結果を第一引数”handle”に書き出します。handle は続く R_TSIP_AesXXXCmacVerifyUpdate()関数や、R_TSIP_AesXXXCmacVerifyFinal()関数の引数で使用します。

key_index の生成方法については、3.7.1 鍵の注入と更新を参照してください。

Reentrant

非対応

4.2.3.35 R_TSIP_Aes*XXX*CmacVerifyUpdate

Format

(1)
```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_Aes128CmacVerifyUpdate(
        tsip_cmac_handle_t *handle,
        uint8_t *message,
        uint32_t message_length
)
```

(2)
```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_Aes256CmacVerifyUpdate(
        tsip_cmac_handle_t *handle,
        uint8_t *message,
        uint32_t message_length
)
```

Parameters

handle	入力/出力	AES-CMAC 用ハンドラ(ワーク領域)
message	入力	メッセージデータ領域(message_length byte)
message_length	入力	メッセージデータ長(0～任意 byte)

Return Values

TSIP_SUCCESS :	正常終了
TSIP_ERR_PARAMETER:	不正なハンドルが入力された
TSIP_ERR_PROHIBIT_FUNCTION:	不正な関数が呼び出された

Description

R_TSIP_AesXXXCmacVerifyUpdate()関数は、第二引数”message”で指定された message から R_TSIP_AesXXXCmacVerifyInit()で指定された”key_index”を用いて MAC 値を生成します。本関数内部で、”message”の入力値が 16byte を超えるまでユーザが入力したデータをバッファリングします。入力する”message”データ長はそれぞれ第三引数の”message_len”で指定します。ここでは、”message”入力データの総バイト数ではなく、ユーザが本関数を呼ぶ際に入力するメッセージのデータ長を入力してください。入力値の”message”は 16byte で割り切れない場合、パディング処理は関数内部で実施します。

Reentrant

非対応

4.2.3.36 R_TSIP_Aes*XXX*CmacVerifyFinal

Format

(1)
```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_Aes128CmacVerifyFinal(
        tsip_cmac_handle_t *handle,
        uint8_t *mac,
        uint32_t mac_length
)
```

(2)
```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_Aes256CmacVerifyFinal(
        tsip_cmac_handle_t *handle,
        uint8_t *mac,
        uint32_t mac_length
)
```

Parameters

handle	入力	AES-CMAC 用ハンドラ(ワーク領域)
mac	入力	MAC データ領域(16byte)
mac_length	入力	MAC データ長(2～16byte)

Return Values

TSIP_SUCCESS :	正常終了
TSIP_ERR_FAIL:	内部エラーが発生
TSIP_ERR_AUTHENTICATION:	認証が失敗
TSIP_ERR_PARAMETER:	不正なハンドルが入力された
TSIP_ERR_PROHIBIT_FUNCTION:	不正な関数が呼び出された

Description

R_TSIP_AesXXXCmacVerifyFinal()関数は、第二引数で指定された”mac”に Mac 値を入力し、Mac 値を検証します。認証が失敗した場合は、戻り値 TSIP_ERR_AUTHENTICATION が返ります。Mac 値が 16byte 以下の場合は、本関数内で 0padding をします。

Reentrant

非対応

4.2.4 DES

4.2.4.1 R_TSIP_GenerateTdesKeyIndex

Format

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

Parameters

encrypted_provisioning_key	入力	W-UFPK
iv	入力	encrypted_key 生成時に使用した初期ベクタ
encrypted_key	入力	UFPK で暗号化された Encrypted Key
key_index	出力	Wrapped Key

Return Values

TSIP_SUCCESS :	正常終了
TSIP_ERR_RESOURCE_CONFLICT:	本処理に必要なハードウェアリソースが他の処理で使用されていることによるリソース衝突が発生
TSIP_ERR_FAIL:	内部エラーが発生

Description

TDES の鍵の Wrapped Key を出力するための API です。

encrypted_key には 7.3.2DES で示すデータを UFPK で暗号化したデータを入力してください。
encrypted_key, iv および encrypted_provisioning_key の説明、および key_index の使用方法は、3.7.1 鍵の注入と更新を参照してください。

Reentrant

非対応

4.2.4.2 R_TSIP_UpdateTdesKeyIndex

Format

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

Parameters

iv	入力	encrypted_key 生成時に使用した初期ベクタ
encrypted_key	入力	KUK で暗号化された Encrypted Key
key_index	出力	Wrapped Key

Return Values

TSIP_SUCCESS :	正常終了
TSIP_ERR_RESOURCE_CONFLICT:	本処理に必要なハードウェアリソースが他の処理で使用されていることによるリソース衝突が発生
TSIP_ERR_FAIL:	内部エラーが発生

Description

TDES の鍵の Wrapped Key を出力するための API です。

encrypted_key には 7.3.2 DES で示すデータを KUK で暗号化したデータを入力してください。iv, encrypted_key の説明、および key_index の使用方法については、3.7.1 鍵の注入と更新を参照してください。

Reentrant

非対応

4.2.4.3 R_TSIP_GenerateTdesRandomKeyIndex

Format

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

Parameters

key_index	出力	Wrapped Key

Return Values

TSIP_SUCCESS :	正常終了
TSIP_ERR_RESOURCE_CONFLICT:	本処理に必要なハードウェアリソースが他の処理で使用されていることによるリソース衝突が発生

Description

TDES の鍵の Wrapped Key を出力するための API です。

本 API は TSIP 内部にて乱数値からユーザ鍵を生成します。従ってユーザ鍵の入力は不要です。本 API が出力する Wrapped Key を使用しデータを暗号化することにより、データのデッドコピーを防ぐことができます。

key_index の使用方法については、3.7.1 鍵の注入と更新を参照してください。

Reentrant

非対応

4.2.4.4 R_TSIP_TdesEcbEncryptInit

Format

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

Parameters

handle	出力	TDES 用ハンドラ(ワーク領域)
key_index	入力	Wrapped Key

Return Values

TSIP_SUCCESS :	正常終了
TSIP_ERR_RESOURCE_CONFLICT:	本処理に必要なハードウェアリソースが他の処理で使用されていることによるリソース衝突が発生
TSIP_ERR_KEY_SET:	異常な Wrapped Key が入力された

Description

R_TSIP_TdesEcbEncryptInit()関数は、DES 演算を実行する準備を行い、その結果を第一引数”handle”に書き出します。handle は、続く R_TSIP_TdesEcbEncryptUpdate()関数および R_TSIP_TdesEcbEncryptFinal()関数で引数として使用されます。

key_index の生成方法については、3.7.1 鍵の注入と更新を参照してください。

Reentrant

非対応

4.2.4.5 R_TSIP_TdesEcbEncryptUpdate

Format

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

Parameters

handle	入力/出力	TDES 用ハンドラ(ワーク領域)
plain	入力	平文データ領域
cipher	出力	暗号文データ領域
plain_length	入力	平文データのバイト長(8 の倍数である必要があります)

Return Values

TSIP_SUCCESS :	正常終了
TSIP_ERR_PARAMETER:	不正なハンドルが入力された
TSIP_ERR_PROHIBIT_FUNCTION:	不正な関数が呼び出された

Description

R_TSIP_TdesEcbEncryptUpdate()関数は、第一引数”handle”で指定されたハンドルを使用し、第二引数の”plain”を R_TSIP_TdesEcbEncryptInit()関数で指定した key_index を用いて暗号化し、結果を第三引数”cipher”に書き出します。平文入力が完了した後は、R_TSIP_TdesEcbEncryptFinal()を呼び出してください。

plain と cipher は、同一アドレスの場合を除き、必ず領域が重ならないように配置してください。

Reentrant

非対応

4.2.4.6 R_TSIP_TdesEcbEncryptFinal

Format

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

Parameters

handle	入力	TDES 用ハンドラ(ワーク領域)
cipher	出力	暗号文データ領域(常に何も書き込まれません)
cipher_length	出力	暗号文データ長(常に 0 が書き込まれます)

Return Values

TSIP_SUCCESS :	正常終了
TSIP_ERR_FAIL:	内部エラーが発生
TSIP_ERR_PARAMETER:	不正なハンドルが入力された
TSIP_ERR_PROHIBIT_FUNCTION:	不正な関数が呼び出された

Description

R_TSIP_TdesEcbEncryptFinal()関数は、第一引数”handle”で指定されたハンドルを使用し、第二引数の”cipher”に演算結果、第三引数”cipher_length”に演算結果の長さを書き出します。第二引数は、本来は 8 バイトの倍数に満たない分の端数について暗号化した結果が書き出されますが、R_TSIP_TdesEcbEncryptUpdate()関数には 8 バイトの倍数でしか入力できない制限があるため、cipher には常に何も書き込まれず、cipher_length には常に 0 が書き込まれます。cipher, cipher_length は将来この制限が解除された際の互換性のための引数です。

Reentrant

非対応

4.2.4.7　R_TSIP_TdesEcbDecryptInit

Format

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

Parameters

handle	出力	TDES 用ハンドラ(ワーク領域)
key_index	入力	Wrapped Key

Return Values

TSIP_SUCCESS :	正常終了
TSIP_ERR_KEY_SET:	異常な Wrapped Key が入力された
TSIP_ERR_RESOURCE_CONFLICT:	本処理に必要なハードウェアリソースが他の処理で使用されていることによるリソース衝突が発生

Description

R_TSIP_TdesEcbDecryptInit()関数は、DES 演算を実行する準備を行い、その結果を第一引数”handle”に書き出します。handle は、続く R_TSIP_TdesEcbDecryptUpdate()関数および R_TSIP_TdesEcbDecryptFinal()関数で引数として使用されます。

key_index の生成方法については、3.7.1 鍵の注入と更新を参照してください。

Reentrant

非対応

4.2.4.8 R_TSIP_TdesEcbDecryptUpdate

Format

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

Parameters

handle	入力/出力	TDES 用ハンドラ(ワーク領域)
cipher	入力	暗号文データ領域
plain	出力	平文データ領域
cipher_length	入力	暗号文データのバイト長(8 の倍数である必要があります)

Return Values

TSIP_SUCCESS :	正常終了
TSIP_ERR_PARAMETER:	不正なハンドルが入力された
TSIP_ERR_PROHIBIT_FUNCTION:	不正な関数が呼び出された

Description

R_TSIP_TdesEcbDecryptUpdate()関数は、第一引数”handle”で指定されたハンドルを使用し、第二引数の”cipher”を R_TSIP_TdesEcbDecryptInit()関数で指定した key_index を用いて復号し、結果を第三引数”plain”に書き出します。暗号文入力が完了した後は、R_TSIP_TdesEcbDecryptFinal()を呼び出してください。

plain と cipher は、同一アドレスの場合を除き、必ず領域が重ならないように配置してください。

Reentrant

非対応

4.2.4.9 R_TSIP_TdesEcbDecryptFinal

Format

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

Parameters

handle	入力	TDES 用ハンドラ(ワーク領域)
plain	出力	平文データ領域(常に何も書き込まれません)
plain_length	出力	平文データ長(常に 0 が書き込まれます)

Return Values

TSIP_SUCCESS :	正常終了
TSIP_ERR_FAIL:	内部エラーが発生
TSIP_ERR_PARAMETER:	不正なハンドルが入力された
TSIP_ERR_PROHIBIT_FUNCTION:	不正な関数が呼び出された

Description

R_TSIP_TdesEcbDecryptFinal()関数は、第一引数”handle”で指定されたハンドルを使用し、第二引数の”plain”に演算結果、第三引数”plain_length”に演算結果の長さを書き出します。第二引数は、本来は 8 バイトの倍数に満たない分の端数について復号した結果が書き出されますが、R_TSIP_TdesEcbDecryptUpdate()関数には 8 バイトの倍数でしか入力できない制限があるため、plain には常に何も書き込まれず、plain_length には常に 0 が書き込まれます。plain, plain_length は将来この制限が解除された際の互換性のための引数です。

Reentrant

非対応

4.2.4.10 R_TSIP_TdesCbcEncryptInit

Format

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

Parameters

handle	出力	TDES 用ハンドラ(ワーク領域)
key_index	入力	Wrapped Key
ivec	入力	初期化ベクタ(8 バイト)

Return Values

TSIP_SUCCESS :	正常終了
TSIP_ERR_KEY_SET:	異常な Wrapped Key が入力された
TSIP_ERR_RESOURCE_CONFLICT:	本処理に必要なハードウェアリソースが他の処理で使用されていることによるリソース衝突が発生

Description

R_TSIP_TdesCbcEncryptInit()関数は、DES 演算を実行する準備を行い、その結果を第一引数”handle”に書き出します。handle は、続く R_TSIP_TdesCbcEncryptUpdate()関数および R_TSIP_TdesCbcEncryptFinal()関数で引数として使用されます。

key_index の生成方法については、3.7.1 鍵の注入と更新を参照してください。

Reentrant

非対応

4.2.4.11 R_TSIP_TdesCbcEncryptUpdate

Format

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

Parameters

handle	入力/出力	TDES 用ハンドラ(ワーク領域)
plain	入力	平文データ領域
cipher	出力	暗号文データ領域
plain_length	入力	平文データのバイト長(8 の倍数である必要があります)

Return Values

TSIP_SUCCESS :	正常終了
TSIP_ERR_PARAMETER:	不正なハンドルが入力された
TSIP_ERR_PROHIBIT_FUNCTION:	不正な関数が呼び出された

Description

R_TSIP_TdesCbcEncryptUpdate()関数は、第一引数”handle”で指定されたハンドルを使用し、第二引数の”plain”を R_TSIP_TdesCbcEncryptInit()関数で指定した key_index を用いて暗号化し、結果を第三引数”cipher”に書き出します。平文入力が完了した後は、R_TSIP_TdesCbcEncryptFinal()を呼び出してください。

plain と cipher は、同一アドレスの場合を除き、必ず領域が重ならないように配置してください。

Reentrant

非対応

4.2.4.12 R_TSIP_TdesCbcEncryptFinal

Format

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

Parameters

handle	入力	TDES 用ハンドラ(ワーク領域)
cipher	出力	暗号文データ領域(常に何も書き込まれません)
cipher_length	出力	暗号文データ長(常に 0 が書き込まれます)

Return Values

TSIP_SUCCESS :	正常終了
TSIP_ERR_FAIL:	内部エラーが発生
TSIP_ERR_PARAMETER:	不正なハンドルが入力された
TSIP_ERR_PROHIBIT_FUNCTION:	不正な関数が呼び出された

Description

R_TSIP_TdesCbcEncryptFinal()関数は、第一引数”handle”で指定されたハンドルを使用し、第二引数の”cipher”に演算結果、第三引数”cipher_length”に演算結果の長さを書き出します。第二引数は、本来は 8 バイトの倍数に満たない分の端数について暗号化した結果が書き出されますが、R_TSIP_TdesCbcEncryptUpdate()関数には 8 バイトの倍数でしか入力できない制限があるため、cipher には常に何も書き込まれず、cipher_length には常に 0 が書き込まれます。cipher, cipher_length は将来この制限が解除された際の互換性のための引数です。

Reentrant

非対応

4.2.4.13 R_TSIP_TdesCbcDecryptInit

Format

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

Parameters

handle	出力	TDES 用ハンドラ(ワーク領域)
key_index	入力	Wrapped Key
ivec	入力	初期化ベクタ(8 バイト)

Return Values

TSIP_SUCCESS :	正常終了
TSIP_ERR_KEY_SET:	異常な Wrapped Key が入力された
TSIP_ERR_RESOURCE_CONFLICT:	本処理に必要なハードウェアリソースが他の処理で使用されていることによるリソース衝突が発生

Description

R_TSIP_TdesCbcDecryptInit()関数は、DES 演算を実行する準備を行い、その結果を第一引数”handle”に書き出します。handle は、続く R_TSIP_TdesCbcDecryptUpdate()関数および R_TSIP_TdesCbcDecryptFinal()関数で引数として使用されます。

key_index の生成方法については、3.7.1 鍵の注入と更新を参照してください。

Reentrant

非対応

4.2.4.14 R_TSIP_TdesCbcDecryptUpdate

Format

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

Parameters

handle	入力/出力	TDES 用ハンドラ(ワーク領域)
cipher	入力	暗号文データ領域
plain	出力	平文データ領域
cipher_length	入力	暗号文データのバイト長(8 の倍数である必要があります)

Return Values

TSIP_SUCCESS :	正常終了
TSIP_ERR_PARAMETER:	不正なハンドルが入力された
TSIP_ERR_PROHIBIT_FUNCTION:	不正な関数が呼び出された

Description

R_TSIP_TdesCbcDecryptUpdate()関数は、第一引数”handle”で指定されたハンドルを使用し、第二引数の”cipher”を R_TSIP_TdesCbcDecryptInit()関数で指定した key_index を用いて復号し、結果を第三引数”plain”に書き出します。暗号文入力が完了した後は、R_TSIP_TdesCbcDecryptFinal()を呼び出してください。

plain と cipher は、同一アドレスの場合を除き、必ず領域が重ならないように配置してください。

Reentrant

非対応

4.2.4.15 R_TSIP_TdesCbcDecryptFinal

Format

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

Parameters

handle	入力	TDES 用ハンドラ(ワーク領域)
plain	出力	平文データ領域(常に何も書き込まれません)
plain_length	出力	平文データ長(常に 0 が書き込まれます)

Return Values

TSIP_SUCCESS :	正常終了
TSIP_ERR_FAIL:	内部エラーが発生
TSIP_ERR_PARAMETER:	不正なハンドルが入力された
TSIP_ERR_PROHIBIT_FUNCTION:	不正な関数が呼び出された

Description

R_TSIP_TdesCbcDecryptFinal()関数は、第一引数”handle”で指定されたハンドルを使用し、第二引数の”plain”に演算結果、第三引数”plain_length”に演算結果の長さを書き出します。第二引数は、本来は 8 バイトの倍数に満たない分の端数について復号した結果が書き出されますが、R_TSIP_TdesCbcDecryptUpdate()関数には 8 バイトの倍数でしか入力できない制限があるため、plain には常に何も書き込まれず、plain_length には常に 0 が書き込まれます。plain, plain_length は将来この制限が解除された際の互換性のための引数です。

Reentrant

非対応

4.2.5　ARC4

4.2.5.1　R_TSIP_GenerateArc4KeyIndex

Format

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

Parameters

encrypted_provisioning_key	入力	W-UFPK
iv	入力	encrypted_key 生成時に使用した初期ベクタ
encrypted_key	入力	UFPK で暗号化された Encrypted Key
key_index	出力	Wrapped Key

Return Values

TSIP_SUCCESS :	正常終了
TSIP_ERR_FAIL:	内部エラーが発生
TSIP_ERR_RESOURCE_CONFLICT:	本処理に必要なハードウェアリソースが他の処理で使用されていることによるリソース衝突が発生

Description

ARC4 の鍵の Wrapped Key を出力するための API です。

encrypted_key に入力する UFPK で暗号化するデータのフォーマットは、7.3.3 ARC4 を参照してください。

encrypted_key, iv, encrypted_provisioning_key の説明、および key_index の使用方法については、3.7.1 鍵の注入と更新を参照してください。

Reentrant

非対応

4.2.5.2 R_TSIP_UpdateArc4KeyIndex

Format

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

Parameters

iv	入力	encrypted_key 生成時に使用した初期ベクタ
encrypted_key	入力	KUK で暗号化された Encrypted Key
key_index	出力	Wrapped Key

Return Values

TSIP_SUCCESS :	正常終了
TSIP_ERR_FAIL:	内部エラーが発生
TSIP_ERR_RESOURCE_CONFLICT:	本処理に必要なハードウェアリソースが他の処理で使用されていることによるリソース衝突が発生

Description

ARC4 鍵の Wrapped Key を更新するための API です。

encrypted_key に入力する KUK で暗号化するデータのフォーマットは、7.3.3 ARC4 を参照してください。

iv, encrypted_key の説明、および key_index の使用方法については、3.7.1 鍵の注入と更新を参照してください。

Reentrant

非対応

4.2.5.3 R_TSIP_GenerateArc4RandomKeyIndex

Format

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

Parameters

key_index	出力	Wrapped Key

Return Values

TSIP_SUCCESS :	正常終了
TSIP_ERR_RESOURCE_CONFLICT:	本処理に必要なハードウェアリソースが他の処理で使用されていることによるリソース衝突が発生

Description

ARC4 の鍵の Wrapped Key を出力するための API です。

本 API は TSIP 内部にて乱数値からユーザ鍵を生成します。従ってユーザ鍵の入力は不要です。本 API が出力する Wrapped Key を使用しデータを暗号化することにより、データのデッドコピーを防ぐことができます。

key_index の使用方法については、3.7.1 鍵の注入と更新を参照してください。

Reentrant

非対応

4.2.5.4　R_TSIP_Arc4EncryptInit

Format

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

Parameters

handle	出力	ARC4 用ハンドラ(ワーク領域)
key_index	入力	Wrapped Key

Return Values

TSIP_SUCCESS :	正常終了
TSIP_ERR_FAIL:	内部エラーが発生
TSIP_ERR_RESOURCE_CONFLICT:	本処理に必要なハードウェアリソースが他の処理で使用されていることによるリソース衝突が発生
TSIP_ERR_KEY_SET:	異常な Wrapped Key が入力された

Description

R_TSIP_Arc4EncryptInit()関数は、ARC4 演算を実行する準備を行い、その結果を第一引数”handle”に書き出します。handle は、続く R_TSIP_Arc4EncryptUpdate()関数および R_TSIP_Arc4EncryptFinal()関数で引数として使用されます。

key_index の生成方法については、3.7.1 鍵の注入と更新を参照してください

Reentrant

非対応

4.2.5.5 R_TSIP_Arc4EncryptUpdate

Format

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

Parameters

handle	入力/出力	ARC4 用ハンドラ(ワーク領域)
plain	入力	平文データ領域
cipher	出力	暗号文データ領域
plain_length	入力	平文データのバイト長(16 の倍数である必要があります)

Return Values

TSIP_SUCCESS :	正常終了
TSIP_ERR_PARAMETER:	不正なハンドルが入力された
TSIP_ERR_PROHIBIT_FUNCTION:	不正な関数が呼び出された

Description

R_TSIP_Arc4EncryptUpdate()関数は、第一引数”handle”で指定されたハンドルを使用し、第二引数の”plain”を R_TSIP_Arc4EncryptInit()関数で指定した key_index を用いて暗号化し、結果を第三引数”cipher”に書き出します。平文入力が完了した後は、R_TSIP_Arc4EncryptFinal()を呼び出してください。

plain と cipher は、同一アドレスの場合を除き、必ず領域が重ならないように配置してください。

Reentrant

非対応

4.2.5.6 R_TSIP_Arc4EncryptFinal

Format

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

Parameters

handle	入力	ARC4 用ハンドラ(ワーク領域)
cipher	出力	暗号文データ領域(常に何も書き込まれません)
cipher_length	出力	暗号文データ長(常に 0 が書き込まれます)

Return Values

TSIP_SUCCESS :	正常終了
TSIP_ERR_FAIL:	内部エラーが発生
TSIP_ERR_PARAMETER:	不正なハンドルが入力された
TSIP_ERR_PROHIBIT_FUNCTION:	不正な関数が呼び出された

Description

R_TSIP_Arc4EncryptFinal()関数は、第一引数”handle”で指定されたハンドルを使用し、第二引数の”cipher”に演算結果、第三引数”cipher_length”に演算結果の長さを書き出します。第二引数は、本来は 16 バイトの倍数に満たない分の端数について暗号化した結果が書き出されますが、R_TSIP_Arc4EncryptUpdate()関数には 16 バイトの倍数でしか入力できない制限があるため、cipher には常に何も書き込まれず、cipher_length には常に 0 が書き込まれます。cipher, cipher_length は将来この制限が解除された際の互換性のための引数です。

Reentrant

非対応

4.2.5.7　R_TSIP_Arc4DecryptInit

Format

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

Parameters

handle	出力	ARC4 用ハンドラ(ワーク領域)
key_index	入力	Wrapped Key

Return Values

TSIP_SUCCESS :	正常終了
TSIP_ERR_RESOURCE_CONFLICT:	本処理に必要なハードウェアリソースが他の処理で使用されていることによるリソース衝突が発生
TSIP_ERR_KEY_SET:	異常な Wrapped Key が入力された

Description

R_TSIP_Arc4DecryptInit()関数は、ARC4 演算を実行する準備を行い、その結果を第一引数”handle”に書き出します。handle は、続く R_TSIP_Arc4DecryptUpdate()関数および R_TSIP_Arc4DecryptFinal()関数で引数として使用されます。

key_index の生成方法については、3.7.1 鍵の注入と更新を参照してください

Reentrant

非対応

4.2.5.8 R_TSIP_Arc4DecryptUpdate

Format

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

Parameters

handle	入力/出力	ARC4 用ハンドラ(ワーク領域)
cipher	入力	暗号文データ領域
plain	出力	平文データ領域
cipher_length	入力	暗号文データのバイト長(16 の倍数である必要があります)

Return Values

TSIP_SUCCESS :	正常終了
TSIP_ERR_PARAMETER:	不正なハンドルが入力された
TSIP_ERR_PROHIBIT_FUNCTION:	不正な関数が呼び出された

Description

R_TSIP_Arc4DecryptUpdate()関数は、第一引数”handle”で指定されたハンドルを使用し、第二引数の”cipher”を R_TSIP_Arc4DecryptInit()関数で指定した key_index を用いて復号し、結果を第三引数”plain”に書き出します。暗号文入力が完了した後は、R_TSIP_Arc4DecryptFinal()を呼び出してください。

plain と cipher は、同一アドレスの場合を除き、必ず領域が重ならないように配置してください。

Reentrant

非対応

4.2.5.9 R_TSIP_Arc4DecryptFinal

Format

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

Parameters

handle	入力	ARC4 用ハンドラ(ワーク領域)
plain	出力	平文データ領域(常に何も書き込まれません)
plain_length	出力	平文データ長(常に 0 が書き込まれます)

Return Values

TSIP_SUCCESS :	正常終了
TSIP_ERR_FAIL:	内部エラーが発生
TSIP_ERR_PARAMETER:	不正なハンドルが入力された
TSIP_ERR_PROHIBIT_FUNCTION:	不正な関数が呼び出された

Description

R_TSIP_Arc4DecryptFinal()関数は、第一引数”handle”で指定されたハンドルを使用し、第二引数の”plain”に演算結果、第三引数”plain_length”に演算結果の長さを書き出します。第二引数は、本来は 16 バイトの倍数に満たない分の端数について復号した結果が書き出されますが、R_TSIP_Arc4DecryptUpdate()関数には 16 バイトの倍数でしか入力できない制限があるため、plain には常に何も書き込まれず、plain_length には常に 0 が書き込まれます。plain, plain_length は将来この制限が解除された際の互換性のための引数です。

Reentrant

非対応

4.2.6　RSA

4.2.6.1　R_TSIP_GenerateRsa*XXX*PublicKeyIndex

Format

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

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

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

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

Parameters

encrypted_provisioning_key	入力	W-UFPK
iv	入力	encrypted_key 生成時に使用した初期ベクタ
encrypted_key	入力	UFPK で暗号化された Encrypted Key
key_index	出力	Wrapped Key
value		公開鍵値
key_management_info1		鍵管理情報
key_n		Modulus n (平文) (1) RSA 1024bit (2) RSA 2048bit (3) RSA 3072bit (4) RSA 4096bit
key_e		Exponent e(平文) (1) RSA 1024bit

	(2) RSA 2048bit
	(3) RSA 3072bit
	(4) RSA 4096bit
dummy	ダミー
key_management_info2	鍵管理情報

Return Values

TSIP_SUCCESS :	正常終了
TSIP_ERR_RESOURCE_CONFLICT:	本処理に必要なハードウェアリソースが他の処理で使用されていることによるリソース衝突が発生
TSIP_ERR_FAIL	内部エラーが発生

Description

1024 bit、2048bit、3072bit、4096bit の RSA 公開鍵 Wrapped Key を出力するための API です。

encrypted_key に入力する UFPK で暗号化するデータのフォーマットは、7.3.4RSA を参照してください。

encrypted_key と key_index は領域が重ならないように配置してください。

encrypted_provisioning_key, iv, encrypted_key の説明、および key_index の使用方法については、3.7.1 鍵の注入と更新を参照してください。

Reentrant

非対応

4.2.6.2 R_TSIP_GenerateRsa*XXX*PrivateKeyIndex

Format

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

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

Parameters

encrypted_provisioning_key	入力	W-UFPK
iv	入力	encrypted_key 生成時に使用した初期ベクタ
encrypted_key	入力	UFPK で暗号化された Encrypted Key
key_index	出力	RSA 秘密鍵 Wrapped Key (1) RSA 1024bit (2) RSA 2048bit

Return Values

TSIP_SUCCESS :	正常終了
TSIP_ERR_RESOURCE_CONFLICT:	本処理に必要なハードウェアリソースが他の処理で使用されていることによるリソース衝突が発生
TSIP_ERR_FAIL	内部エラーが発生

Description

1024 bit、2048bit の RSA 秘密鍵の Wrapped Key を出力するための API です。

encrypted_key に入力する UFPK で暗号化するデータのフォーマットは、7.3.4 RSA を参照してください。

encrypted_key と key_index は領域が重ならないように配置してください。

encrypted_provisioning_key, iv, encrypted_key の説明、および key_index の使用方法については 3.7.1 鍵の注入と更新を参照してください。

Reentrant

非対応

4.2.6.3 R_TSIP_UpdateRsa*XXX*PublicKeyIndex

Format

(1)
```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_UpdateRsa1024PublicKeyIndex(
        uint8_t *iv,
        uint8_t *encrypted_key,
        tsip_rsa1024_public_key_index_t *key_index
)
```

(2)
```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_UpdateRsa2048PublicKeyIndex(
        uint8_t *iv,
        uint8_t *encrypted_key,
        tsip_rsa2048_public_key_index_t *key_index
)
```

(3)
```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_UpdateRsa3072PublicKeyIndex(
        uint8_t *iv,
        uint8_t *encrypted_key,
        tsip_rsa3072_public_key_index_t *key_index
)
```

(4)
```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_UpdateRsa4096PublicKeyIndex(
        uint8_t *iv,
        uint8_t *encrypted_key,
        tsip_rsa4096_public_key_index_t *key_index
)
```

Parameters

iv	入力	encrypted_key 生成時に使用した初期ベクタ
encrypted_key	入力	KUK で暗号化された Encrypted Key
key_index	出力	Wrapped Key
value		公開鍵値
key_management_info1		鍵管理情報
key_n		Modulus n (平文) (1) RSA 1024bit (2) RSA 2048bit (3) RSA 3072bit (4) RSA 4096bit
key_e		Exponent e(平文) (1) RSA 1024bit (2) RSA 2048bit (3) RSA 3072bit (4) RSA 4096bit
dummy		ダミー
key_management_info2		鍵管理情報

Return Values

TSIP_SUCCESS :	正常終了
TSIP_ERR_RESOURCE_CONFLICT:	本処理に必要なハードウェアリソースが他の処理で使用されていることによるリソース衝突が発生
TSIP_ERR_FAIL	内部エラーが発生

Description

RSA 1024bit、2048bit、3072bit、4096bit 公開鍵の Wrapped Key を更新するための API です。

encrypted_key に入力する KUK で暗号化するデータのフォーマットは、7.3.4 RSA を参照してください。

iv, encrypted_key の説明、および key_index の使用方法については、3.7.1 鍵の注入と更新を参照してください。

Reentrant

非対応

4.2.6.4 R_TSIP_UpdateRsa*XXX*PrivateKeyIndex

Format

(1)
```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_UpdateRsa1024PrivateKeyIndex(
        uint8_t *iv,
        uint8_t *encrypted_key,
        tsip_rsa1024_private_key_index_t *key_index
)
```

(2)
```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_UpdateRsa2048PrivateKeyIndex(
        uint8_t *iv,
        uint8_t *encrypted_key,
        tsip_rsa2048_private_key_index_t *key_index
)
```

Parameters

iv	入力	encrypted_key 生成時に使用した初期ベクタ
encrypted_key	入力	KUK で暗号化された Encrypted Key
key_index	出力	Wrapped Key (1) RSA 1024bit (2) RSA 2048bit

Return Values

TSIP_SUCCESS :	正常終了
TSIP_ERR_RESOURCE_CONFLICT:	本処理に必要なハードウェアリソースが他の処理で使用されていることによるリソース衝突が発生
TSIP_ERR_FAIL	内部エラーが発生

Description

RSA 1024bit、2048bit 秘密鍵の Wrapped Key を更新するための API です。encrypted_key に入力する UFPK で暗号化するデータのフォーマットは、7.3.4 RSA を参照してください。

iv, encrypted_key の説明、および key_index の使用方法については、3.7.1 鍵の注入と更新を参照してください。

Reentrant

非対応

4.2.6.5 R_TSIP_GenerateRsa*XXX*RandomKeyIndex

Format

(1)
```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_GenerateRsa1024RandomKeyIndex(
        tsip_rsa1024_key_pair_index_t *key_pair_index
)
```

(2)
```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_GenerateRsa2048RandomKeyIndex(
        tsip_rsa2048_key_pair_index_t *key_pair_index
)
```

Parameters

key_pair_index	出力	RSA 鍵ペアの Wrapped Key
public		RSA 公開鍵の Wrapped Key
value		公開鍵値
key_management_info1		鍵管理情報
key_n		Modulus n (平文)
		(1) RSA 1024bit
		(2) RSA 2048bit
key_e		Exponent e (平文)
		(1) RSA 1024bit
		(2) RSA 2048bit
dummy		ダミー
key_management_info2		鍵管理情報
private		RSA 秘密鍵の Wrapped Key

Return Values

TSIP_SUCCESS :	正常終了
TSIP_ERR_RESOURCE_CONFLICT:	本処理に必要なハードウェアリソースが他の処理で使用されていることによるリソース衝突が発生
TSIP_ERR_FAIL	内部エラーが発生

Description

1024 bit、2048bit の RSA 公開鍵、秘密鍵ペアの Wrapped Key を出力するための API です。本 API は TSIP 内部にて乱数値からユーザ鍵を生成します。従ってユーザ鍵の入力は不要です。本 API が出力する Wrapped Key を使用しデータを暗号化することにより、データのデッドコピーを防ぐことができます。key_pair_index->public に公開鍵の Wrapped Key、key_pair_index->private に秘密鍵の Wrapped Key を生成します。公開鍵の exponent は 0x00010001 のみを生成しています。

key_pair_index->public ならびに key_pair_index->private 使用方法については 3.7.1 鍵の注入と更新を参照してください。key_pair_index->public は R_TSIP_GenerateRsa*XXX*PublicKeyIndex()から出力される公開鍵の Wrapped Key、key_pair_index->private は R_TSIP_GenerateRsa*XXX*PrivateKeyIndex()から出力される秘密鍵の Wrapped Key と同様の運用になります。

Reentrant

非対応

4.2.6.6 R_TSIP_RsaesPkcs*XXX*Encrypt

Format

(1)
```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_RsaesPkcs1024Encrypt(
        tsip_rsa_byte_data_t *plain,
        tsip_rsa_byte_data_t *cipher,
        tsip_rsa1024_public_key_index_t *key_index
)
```

(2)
```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_RsaesPkcs2048Encrypt(
        tsip_rsa_byte_data_t *plain,
        tsip_rsa_byte_data_t *cipher,
        tsip_rsa2048_public_key_index_t *key_index
)
```

(3)
```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_RsaesPkcs3072Encrypt(
        tsip_rsa_byte_data_t *plain,
        tsip_rsa_byte_data_t *cipher,
        tsip_rsa3072_public_key_index_t *key_index
)
```

(4)
```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_RsaesPkcs4096Encrypt(
        tsip_rsa_byte_data_t *plain,
        tsip_rsa_byte_data_t *cipher,
        tsip_rsa4096_public_key_index_t *key_index
)
```

Parameters

plain	入力	暗号化する平文データ
pdata		平文を格納している配列のポインタを指定
data_length		平文配列の有効データ長を指定 データサイズ <= Modulus n サイズ-11
cipher	出力	暗号化されたデータ
pdata		暗号文を格納する配列のポインタを指定
data_length		暗号文のバッファサイズを入力 暗号化後、有効データ長を出力(Modulus n サイズ)
key_index	入力	Wrapped Key (1) RSA 1024bit (2) RSA 2048bit (3) RSA 3072bit (4) RSA 4096bit

Return Values

TSIP_SUCCESS :	正常終了
TSIP_ERR_RESOURCE_CONFLICT:	本処理に必要なハードウェアリソースが他の処理で使用されていることによるリソース衝突が発生
TSIP_ERR_KEY_SET	異常な Wrapped Key が入力された
TSIP_ERR_PARAMETER	入力データが不正
TSIP_ERR_FAIL	内部エラーが発生（*XXX* = 3072, 4096 のみ）

Description

R_TSIP_RsaesPkcs*XXX*Encrypt()関数は、第一引数”plain”に入力された平文を RSAES-PKCS1-V1_5 に従って、RSA 暗号化をします。暗号化結果を第二引数”cipher”に書き出します。

key_index の生成方法については、3.7.1 鍵の注入と更新を参照してください。

Reentrant

非対応

4.2.6.7 R_TSIP_RsaesPkcs*XXX*Decrypt

Format

(1)
```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_RsaesPkcs1024Decrypt(
        tsip_rsa_byte_data_t *cipher,
        tsip_rsa_byte_data_t *plain,
        tsip_rsa1024_private_key_index_t *key_index
)
```

(2)
```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_RsaesPkcs2048Decrypt(
        tsip_rsa_byte_data_t *cipher,
        tsip_rsa_byte_data_t *plain,
        tsip_rsa2048_private_key_index_t *key_index
)
```

Parameters

cipher	入力	復号する暗号データ
pdata		暗号文を格納する配列のポインタを指定
data_length		暗号文配列の有効データ長を指定 (Modulus n サイズ)
plain	出力	復号された平文データ
pdata		平文を格納している配列のポインタを指定
data_length		平文配列の有効データ長を指定 データサイズ <= Modulus n サイズ-11
key_index	入力	Wrapped Key (1) RSA 1024bit (2) RSA 2048bit

Return Values

TSIP_SUCCESS :	正常終了
TSIP_ERR_RESOURCE_CONFLICT:	本処理に必要なハードウェアリソースが他の処理で使用されていることによるリソース衝突が発生
TSIP_ERR_KEY_SET	異常な Wrapped Key が入力された
TSIP_ERR_PARAMETER	入力データが不正

Description

R_TSIP_RsaesPkcs*XXX*Decrypt()関数は、第一引数”cipher”に入力された暗号文を RSAES-PKCS1-V1_5 に従って、RSA 復号を行います。復号結果を第二引数”plain”に出力します。

key_index の生成方法については、3.7.1 鍵の注入と更新を参照してください。

Reentrant

非対応

4.2.6.8 R_TSIP_RsassaPkcs*XXX*SignatureGenerate

Format

(1)
```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_RsassaPkcs1024SignatureGenerate(
        tsip_rsa_byte_data_t *message_hash,
        tsip_rsa_byte_data_t *signature,
        tsip_rsa1024_private_key_index_t *key_index,
        uint8_t hash_type
)
```

(2)
```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_RsassaPkcs2048SignatureGenerate(
        tsip_rsa_byte_data_t *message_hash,
        tsip_rsa_byte_data_t *signature,
        tsip_rsa2048_private_key_index_t *key_index,
        uint8_t hash_type
)
```

Parameters

message_hash	入力	署名を付けるメッセージまたはハッシュ値情報
pdata		メッセージまたはハッシュ値を格納している配列のポインタを指定
data_length		配列の有効データ長(メッセージの場合のみ指定)
data_type		message_hash のデータ種別を選択
		0 : メッセージ(ハッシュ計算を本関数で実行)
		1 : ハッシュ値(ハッシュ計算結果を入力)
signature	出力	署名文格納先情報
pdata		署名文を格納する配列のポインタを指定
data_length		データ長(バイト単位)
key_index	入力	Wrapped Key
		(1) RSA 1024bit
		(2) RSA 2048bit
hash_type	入力	署名を付ける hash の種類
		R_TSIP_RSA_HASH_MD5
		R_TSIP_RSA_HASH_SHA1
		R_TSIP_RSA_HASH_SHA256

Return Values

TSIP_SUCCESS :	正常終了
TSIP_ERR_RESOURCE_CONFLICT:	本処理に必要なハードウェアリソースが他の処理で使用されていることによるリソース衝突が発生
TSIP_ERR_KEY_SET	異常な Wrapped Key が入力された
TSIP_ERR_PARAMETER	入力データが不正

Description

R_TSIP_RsassaPkcs*XXX*SignatureGenerate()関数は、RSASSA-PKCS1-V1_5 に従って、第一引数”message_hash”に入力されたメッセージ文またはハッシュ値から、第三引数”key_index”に入力された秘密鍵の Wrapped Key を使って署名文を計算し、第二引数”signature”に書き出します。第一引数”message_hash->data_type”でメッセージを指定した場合、メッセージに対して第四引数”hash_type”で指定された HASH 計算を行います。第一引数”message_hash->data_type”でハッシュ値を指定した場合、第四引数”hash_type”で指定したハッシュアルゴリズムで計算したハッシュ値を”message_hash->pdata”へ入力してください。

key_index の生成方法については、3.7.1 鍵の注入と更新を参照してください。

Reentrant

非対応

4.2.6.9 R_TSIP_RsassaPkcs*XXX*SignatureVerification

Format

(1)
```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_RsassaPkcs1024SignatureVerification(
        tsip_rsa_byte_data_t *signature,
        tsip_rsa_byte_data_t *message_hash,
        tsip_rsa1024_public_key_index_t *key_index,
        uint8_t hash_type
)
```

(2)
```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_RsassaPkcs2048SignatureVerification(
        tsip_rsa_byte_data_t *signature,
        tsip_rsa_byte_data_t *message_hash,
        tsip_rsa2048_public_key_index_t *key_index,
        uint8_t hash_type
)
```

(3)
```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_RsassaPkcs3072SignatureVerification(
        tsip_rsa_byte_data_t *signature,
        tsip_rsa_byte_data_t *message_hash,
        tsip_rsa3072_public_key_index_t *key_index,
        uint8_t hash_type
)
```

(4)
```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_RsassaPkcs4096SignatureVerification(
        tsip_rsa_byte_data_t *signature,
        tsip_rsa_byte_data_t *message_hash,
        tsip_rsa4096_public_key_index_t *key_index,
        uint8_t hash_type
)
```

Parameters

signature	入力	検証する署名文情報
pdata		署名文を格納している配列のポインタを指定
message_hash	入力	検証するメッセージ文またはハッシュ値情報
pdata		メッセージまたはハッシュ値を格納している配列のポインタを指定
data_length		配列の有効データ長(メッセージの場合のみ指定)
data_type		message_hash のデータ種別を選択
		0 : メッセージ(ハッシュ計算を本関数で実行)
		1 : ハッシュ値(ハッシュ計算結果を入力)
key_index	入力	Wrapped Key
		(1) RSA 1024bit
		(2) RSA 2048bit
		(3) RSA 3072bit
		(4) RSA 4096bit
hash_type	入力	hash の種類

R_TSIP_RSA_HASH_MD5
R_TSIP_RSA_HASH_SHA1
R_TSIP_RSA_HASH_SHA256

Return Values

TSIP_SUCCESS :	正常終了
TSIP_ERR_RESOURCE_CONFLICT:	本処理に必要なハードウェアリソースが他の処理で使用されていることによるリソース衝突が発生
TSIP_ERR_KEY_SET	異常な Wrapped Key が入力された
TSIP_ERR_AUTHENTICATION	署名検証失敗
TSIP_ERR_PARAMETER	入力データが不正
TSIP_ERR_FAIL	内部エラーが発生（*XXX* = 3072, 4096 のみ）

Description

R_TSIP_RsassaPkcs*XXX*SignatureVerification()関数は、RSASSA-PKCS1-V1_5 に従って、第三引数”key_index”に入力された公開鍵の Wrapped Key を使い第一引数”signature”に入力された署名文と第二引数”message_hash”に入力されたメッセージ文またはハッシュ値の検証をします。第二引数”message_hash->data_type”でメッセージを指定した場合、第三引数”key_index”に入力された公開鍵の Wrapped Key と第四引数”hash_type”で指定された HASH 計算を行います。第二引数”message_hash->data_type”でハッシュ値を指定した場合、第四引数”hash_type”で指定したハッシュアルゴリズムで計算したハッシュ値を”message_hash->pdata”へ入力してください。

key_index の生成方法については 3.7.1 鍵の注入と更新を参照してください

Reentrant

非対応

4.2.6.10 R_TSIP_RsassaPss*XXX*SignatureGenerate

Format

(1)
```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_RsassaPss1024SignatureGenerate(
        tsip_rsa_byte_data_t *message_hash,
        tsip_rsa_byte_data_t *signature,
        tsip_rsa1024_private_key_index_t *key_index,
        uint8_t hash_type
)
```

(2)
```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_RsassaPss2048SignatureGenerate(
        tsip_rsa_byte_data_t *message_hash,
        tsip_rsa_byte_data_t *signature,
        tsip_rsa2048_private_key_index_t *key_index,
        uint8_t hash_type
)
```

Parameters

message_hash	入力	署名を付けるメッセージまたはハッシュ値情報
pdata		メッセージまたはハッシュ値を格納している配列のポインタを指定
data_length		配列の有効データ長(メッセージの場合のみ指定)
data_type		message_hash のデータ種別を選択
		0 : メッセージ(ハッシュ計算を本関数で実行)
		1 : ハッシュ値(ハッシュ計算結果を入力)
signature	出力	署名文格納先情報
pdata		署名文を格納する配列のポインタを指定
data_length		データ長(バイト単位)
key_index	入力	Wrapped Key
		(1) RSA 1024bit
		(2) RSA 2048bit
hash_type	入力	署名を付ける hash の種類
		R_TSIP_RSA_HASH_SHA1
		R_TSIP_RSA_HASH_SHA256

Return Values

TSIP_SUCCESS :	正常終了
TSIP_ERR_RESOURCE_CONFLICT:	本処理に必要なハードウェアリソースが他の処理で使用されていることによるリソース衝突が発生
TSIP_ERR_KEY_SET	異常な Wrapped Key が入力された
TSIP_ERR_PARAMETER	入力データが不正

Description

R_TSIP_RsassaPss*XXX*SignatureGenerate()関数は、RFC8017 8.1 章の RSASSA-PSS に従って、第一引数”message_hash”に入力されたメッセージ文またはハッシュ値から、第三引数”key_index”に入力された秘密鍵の Wrapped Key を使って署名文を計算し、第二引数”signature”に書き出します。ソルト長は 32 バイト固定です。

第一引数”message_hash->data_type”でメッセージを指定した場合、メッセージに対して第四引数”hash_type”で指定された HASH 計算を行います。

第一引数”message_hash->data_type”でハッシュ値を指定した場合、第四引数”hash_type”で指定したハッシュアルゴリズムで計算したハッシュ値を”message_hash->pdata”へ入力してください。

key_index の生成方法については、3.7.1 鍵の注入と更新を参照してください。

Reentrant

非対応

4.2.6.11 R_TSIP_RsassaPss*XXX*SignatureVerification

Format

(1)
```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_RsassaPss1024SignatureVerification(
        tsip_rsa_byte_data_t *signature,
        tsip_rsa_byte_data_t *message_hash,
        tsip_rsa1024_public_key_index_t *key_index,
        uint8_t hash_type
)
```

(2)
```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_RsassaPss2048SignatureVerification(
        tsip_rsa_byte_data_t *signature,
        tsip_rsa_byte_data_t *message_hash,
        tsip_rsa2048_public_key_index_t *key_index,
        uint8_t hash_type
)
```

Parameters

signature	入力	検証する署名文情報
pdata		署名文を格納している配列のポインタを指定
message_hash	入力	検証するメッセージ文またはハッシュ値情報
pdata		メッセージまたはハッシュ値を格納している配列のポインタを指定
data_length		配列の有効データ長(メッセージの場合のみ指定)
data_type		message_hash のデータ種別を選択
		0 : メッセージ(ハッシュ計算を本関数で実行)
		1 : ハッシュ値(ハッシュ計算結果を入力)
key_index	入力	Wrapped Key
		(1) RSA 1024bit
		(2) RSA 2048bit
hash_type	入力	hash の種類
		R_TSIP_RSA_HASH_SHA1
		R_TSIP_RSA_HASH_SHA256

Return Values

TSIP_SUCCESS :	正常終了
TSIP_ERR_RESOURCE_CONFLICT:	本処理に必要なハードウェアリソースが他の処理で使用されていることによるリソース衝突が発生
TSIP_ERR_KEY_SET	異常な Wrapped Key が入力された
TSIP_ERR_AUTHENTICATION	署名検証失敗
TSIP_ERR_PARAMETER	入力データが不正

Description

R_TSIP_RsassaPss*XXX*SignatureVerification()関数は、RFC8017 8.1 章の RSASSA-PSS に従って、第三引数”key_index”に入力された公開鍵の Wrapped Key を使い第一引数”signature”に入力された署名文と第二引数”message_hash”に入力されたメッセージ文またはハッシュ値の検証をします。ソルト長は 32 バイト固定です。

第二引数”message_hash->data_type”でメッセージを指定した場合、第三引数”key_index”に入力された公開鍵の Wrapped Key と第四引数”hash_type”で指定された HASH 計算を行います。

第二引数”message_hash->data_type”でハッシュ値を指定した場合、第四引数”hash_type”で指定したハッシュアルゴリズムで計算したハッシュ値を”message_hash->pdata”へ入力してください。

key_index の生成方法については、3.7.1 鍵の注入と更新を参照してください。

Reentrant

非対応

4.2.7　ECC

4.2.7.1　R_TSIP_GenerateEccP*XXX*PublicKeyIndex

Format

(1)
```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_GenerateEccP192PublicKeyIndex(
        uint8_t *encrypted_provisioning_key,
        uint8_t *iv,
        uint8_t *encrypted_key,
        tsip_ecc_public_key_index_t *key_index
)
```

(2)
```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_GenerateEccP224PublicKeyIndex(
        uint8_t *encrypted_provisioning_key,
        uint8_t *iv,
        uint8_t *encrypted_key,
        tsip_ecc_public_key_index_t *key_index
)
```

(3)
```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_GenerateEccP256PublicKeyIndex(
        uint8_t *encrypted_provisioning_key,
        uint8_t *iv,
        uint8_t *encrypted_key,
        tsip_ecc_public_key_index_t *key_index
)
```

(4)
```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_GenerateEccP384PublicKeyIndex(
        uint8_t *encrypted_provisioning_key,
        uint8_t *iv,
        uint8_t *encrypted_key,
        tsip_ecc_public_key_index_t *key_index
)
```

Parameters

encrypted_provisioning_key	入力	W-UFPK
iv	入力	encrypted_key 生成時に使用した初期ベクタ
encrypted_key	入力	UPFK で暗号化された Encrypted Key
key_index	出力	Wrapped Key
value		公開鍵値
key_management_info		鍵管理情報
key_q		(1) ECC P-192 公開鍵 Q(平文) (2) ECC P-224 公開鍵 Q(平文) (3) ECC P-256 公開鍵 Q(平文) (4) ECC P-384 公開鍵 Q(平文)

Return Values

TSIP_SUCCESS :	正常終了
TSIP_ERR_RESOURCE_CONFLICT:	本処理に必要なハードウェアリソースが他の処理で使用されていることによるリソース衝突が発生
TSIP_ERR_FAIL	内部エラーが発生

Description

ECC P-192、P-224、P-256、P-384 の公開鍵の Wrapped Key を出力するための API です。

encrypted_key に入力する Provisioning Key で暗号化する公開鍵の暗号化方式は、7.3.5 ECC を参照してください。

encrypted_key と key_index は領域が重ならないように配置してください。

key_index-> value.key_q には公開鍵の平文データを含む構造体が出力されます。そのフォーマットは、7.4.2 ECC を参照してください。

encrypted_provisioning_key, iv, encrypted_key の説明、および key_index の使用方法については、3.7.1 鍵の注入と更新を参照してください。

Reentrant

非対応

4.2.7.2 R_TSIP_GenerateEccP*XXX*PrivateKeyIndex

Format

(1)
```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_GenerateEccP192PrivateKeyIndex(
        uint8_t *encrypted_provisioning_key,
        uint8_t *iv,
        uint8_t *encrypted_key,
        tsip_ecc_private_key_index_t *key_index
)
```

(2)
```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_GenerateEccP224PrivateKeyIndex(
        uint8_t *encrypted_provisioning_key,
        uint8_t *iv,
        uint8_t *encrypted_key,
        tsip_ecc_private_key_index_t *key_index
)
```

(3)
```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_GenerateEccP256PrivateKeyIndex(
        uint8_t *encrypted_provisioning_key,
        uint8_t *iv,
        uint8_t *encrypted_key,
        tsip_ecc_private_key_index_t *key_index
)
```

(4)
```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_GenerateEccP384PrivateKeyIndex(
        uint8_t *encrypted_provisioning_key,
        uint8_t *iv,
        uint8_t *encrypted_key,
        tsip_ecc_private_key_index_t *key_index
)
```

Parameters

encrypted_provisioning_key	入力	W-UFPK
iv	入力	encrypted_key 生成時に使用した初期ベクタ
encrypted_key	入力	UFPK で暗号化された Encrypted Key
key_index	出力	Wrapped Key (1) ECC P-192 秘密鍵 (2) ECC P-224 秘密鍵 (3) ECC P-256 秘密鍵 (4) ECC P-384 秘密鍵

Return Values

TSIP_SUCCESS :	正常終了
TSIP_ERR_RESOURCE_CONFLICT:	本処理に必要なハードウェアリソースが他の処理で使用されていることによるリソース衝突が発生
TSIP_ERR_FAIL	内部エラーが発生

Description

ECC P-192、P-224、P-256、P-384 の秘密鍵の Wrapped Key を出力するための API です。

encrypted_key に入力する UFPK で暗号化するデータのフォーマットは、7.4 非対称鍵暗号 公開鍵の Wrapped Key フォーマットを参照してください。

encrypted_key と key_index は領域が重ならないように配置してください。

encrypted_provisioning_key, iv, encrypted_key の説明、および key_index の使用方法については、3.7.1 鍵の注入と更新を参照してください。

Reentrant

非対応

4.2.7.3 R_TSIP_UpdateEccP*XXX*PublicKeyIndex

Format

(1)
```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_UpdateEccP192PublicKeyIndex(
        uint8_t *iv,
        uint8_t *encrypted_key,
        tsip_ecc_public_key_index_t *key_index
)
```

(2)
```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_UpdateEccP224PublicKeyIndex(
        uint8_t *iv,
        uint8_t *encrypted_key,
        tsip_ecc_public_key_index_t *key_index
)
```

(3)
```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_UpdateEccP256PublicKeyIndex(
        uint8_t *iv,
        uint8_t *encrypted_key,
        tsip_ecc_public_key_index_t *key_index
)
```

(4)
```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_UpdateEccP384PublicKeyIndex(
        uint8_t *iv,
        uint8_t *encrypted_key,
        tsip_ecc_public_key_index_t *key_index
)
```

Parameters

iv	入力	encrypted_key 生成時に使用した初期ベクタ
encrypted_key	入力	KUK で暗号化された Encrypted Key
key_index	出力	Wrappd Key
value		公開鍵値
key_management_info		鍵管理情報
key_q		公開鍵(Qx || Qy) (平文) (1) ECC P-192 (2) ECC P-224 (3) ECC P-256 (4) ECC P-384

Return Values

TSIP_SUCCESS :	正常終了
TSIP_ERR_RESOURCE_CONFLICT:	本処理に必要なハードウェアリソースが他の処理で使用されていることによるリソース衝突が発生
TSIP_ERR_FAIL	内部エラーが発生

Description

ECC P-192、P-224、P-256、P-384 公開鍵の Wrappd Key を更新するための API です。

encrypted_key に入力する KUK で暗号化する公開鍵の暗号化方式ならびにフォーマットは、7.3.5ECC を参照してください。

key_index->value.key_q で出力される公開鍵 Q の平文データのフォーマットは、7.4.2 ECC を参照してください。

iv, encrypted_key の説明、および key_index の使用方法については、3.7.1 鍵の注入と更新を参照してください。

Reentrant

非対応

4.2.7.4 R_TSIP_UpdateEccP*XXX*PrivateKeyIndex

Format

(1)
```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_UpdateEccP192PrivateKeyIndex(
        uint8_t *iv,
        uint8_t *encrypted_key,
        tsip_ecc_private_key_index_t *key_index
)
```

(2)
```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_UpdateEccP224PrivateKeyIndex(
        uint8_t *iv,
        uint8_t *encrypted_key,
        tsip_ecc_private_key_index_t *key_index
)
```

(3)
```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_UpdateEccP256PrivateKeyIndex(
        uint8_t *iv,
        uint8_t *encrypted_key,
        tsip_ecc_private_key_index_t *key_index
)
```

(4)
```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_UpdateEccP384PrivateKeyIndex(
        uint8_t *iv,
        uint8_t *encrypted_key,
        tsip_ecc_private_key_index_t *key_index
)
```

Parameters

iv	入力	encrypted_key 生成時に使用した初期ベクタ
encrypted_key	入力	KUK で暗号化された Encypted Key
key_index	出力	Wrappd Key (1) ECC P-192 秘密鍵 (2) ECC P-224 秘密鍵 (3) ECC P-256 秘密鍵 (4) ECC P-384 秘密鍵

Return Values

TSIP_SUCCESS :	正常終了
TSIP_ERR_RESOURCE_CONFLICT:	本処理に必要なハードウェアリソースが他の処理で使用されていることによるリソース衝突が発生
TSIP_ERR_FAIL	内部エラーが発生

Description

ECC P-192、P-224、P-256、P-384 の秘密鍵の Wrappd Key を更新するための API です。

encrypted_key に入力する KUK で暗号化する秘密鍵の暗号化方式ならびにフォーマットは、7.3.5ECC を参照してください。

iv, encrypted_key の説明、および key_index の使用方法については、3.7.1 鍵の注入と更新を参照してください。

Reentrant

非対応

4.2.7.5 R_TSIP_GenerateEccP*XXX*RandomKeyIndex

Format

(1)
```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_GenerateEccP192RandomKeyIndex(
        tsip_ecc_key_pair_index_t *key_pair_index
)
```

(2)
```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_GenerateEccP224RandomKeyIndex(
        tsip_ecc_key_pair_index_t *key_pair_index
)
```

(3)
```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_GenerateEccP256RandomKeyIndex(
        tsip_ecc_key_pair_index_t *key_pair_index
)
```

(4)
```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_GenerateEccP384RandomKeyIndex(
        tsip_ecc_key_pair_index_t *key_pair_index
)
```

Parameters

key_pair_index	出力	ECC 公開鍵、秘密鍵ペアの Wrappd Key (1) ECC P-192 (2) ECC P-224 (3) ECC P-256 (4) ECC P-384
->public		公開鍵の Wrappd Key
value		公開鍵値
key_management_info		鍵管理情報
key_q		公開鍵(Qx || Qy) (平文)
->private		秘密鍵の Wrappd Key

Return Values

TSIP_SUCCESS :	正常終了
TSIP_ERR_RESOURCE_CONFLICT:	本処理に必要なハードウェアリソースが他の処理で使用されていることによるリソース衝突が発生
TSIP_ERR_FAIL	内部エラーが発生

Description

ECC P-192、P-224、P-256、P384 の公開鍵、秘密鍵ペアの Wrappd Key を出力するための API です。本 API は TSIP 内部にて乱数値からユーザ鍵を生成します。ユーザ鍵の入力は不要です。本 API が出力する Wrappd Key を使用しデータを暗号処理することにより、データのデッドコピーを防ぐことができます。key_pair_index->public に公開鍵の Wrappd Key、key_pair_index->private に秘密鍵の Wrappd Key を生成します。key_pair_index->public.value.key_q には、平文の公開鍵が出力されます。データのフォーマットは、7.4 非対称鍵暗号 公開鍵の Wrapped Key フォーマットを参照してください。

key_pair_index->public ならびに key_pair_index->private 使用方法については 3.7.1 鍵の注入と更新を参照してください。key_pair_index->public は R_TSIP_GenerateEccP*XXX*PublicKeyIndex()から出力される

公開鍵の Wrappd Key、key_pair_index->private は R_TSIP_GenerateEccP*XXX*PrivateKeyIndex()から出力される秘密鍵の Wrappd Key と同様の運用になります。

Reentrant

非対応

4.2.7.6 R_TSIP_EcdsaP*XXX*SignatureGenerate

Format

(1)
```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_EcdsaP192SignatureGenerate(
        tsip_ecdsa_byte_data_t *message_hash,
        tsip_ecdsa_byte_data_t *signature,
        tsip_ecc_private_key_index_t *key_index
)
```

(2)
```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_EcdsaP224SignatureGenerate(
        tsip_ecdsa_byte_data_t *message_hash,
        tsip_ecdsa_byte_data_t *signature,
        tsip_ecc_private_key_index_t *key_index
)
```

(3)
```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_EcdsaP256SignatureGenerate(
        tsip_ecdsa_byte_data_t *message_hash,
        tsip_ecdsa_byte_data_t *signature,
        tsip_ecc_private_key_index_t *key_index
)
```

(4)
```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_EcdsaP384SignatureGenerate(
        tsip_ecdsa_byte_data_t *message_hash,
        tsip_ecdsa_byte_data_t *signature,
        tsip_ecc_private_key_index_t *key_index
)
```

Parameters

message_hash	入力	署名を付けるメッセージまたはハッシュ値情報
pdata		メッセージまたはハッシュ値を格納している配列のポインタを指定
data_length		配列の有効データ長(メッセージの場合のみ指定)
data_type		message_hash のデータ種別を選択 0 : メッセージ(ハッシュ計算を本関数で実行) 1 : ハッシュ値(ハッシュ計算結果を入力)
signature	出力	署名文格納先情報
pdata		署名文を格納する配列のポインタを指定 署名形式は以下の通りです。 (1) ”0 padding(64bit) || 署名 r(192bit) || 0 padding(64bit) || 署名 s(192bit)” (2) ”0 padding(32bit) || 署名 r(224bit) || 0 padding(32bit) || 署名 s(224bit)” (3) ”署名 r(256bit) || 署名 s(256bit)” (4) ”署名 r(384bit) || 署名 s(384bit)”
data_length		データ長(バイト単位)
key_index	入力	Wrappd Key (1) ECC P-192 秘密鍵 (2) ECC P-224 秘密鍵

(3) ECC P-256 秘密鍵

(4) ECC P-384 秘密鍵

Return Values

TSIP_SUCCESS :	正常終了
TSIP_ERR_RESOURCE_CONFLICT:	本処理に必要なハードウェアリソースが他の処理で使用されていることによるリソース衝突が発生
TSIP_ERR_KEY_SET	異常な Wrapped Key が入力された
TSIP_ERR_FAIL	内部エラーが発生
TSIP_ERR_PARAMETER	入力データが不正

Description

(1)R_TSIP_EcdsaP192SignatureGenerate、(2) R_TSIP_EcdsaP224SignatureGenerate、
(3)R_TSIP_EcdsaP256SignatureGenerate を使用する場合

第一引数”message_hash->data_type”でメッセージを指定した場合、第一引数”message_hash->pdata”に入力されたメッセージ文を SHA-256 ハッシュ計算し、第三引数”key_index”に入力された秘密鍵の Wrappd Key から、ECDSA P-192、P-224、P-256 に従い署名文を第二引数”signature”に書き出します。

第一引数”message_hash->data_type”でハッシュ値を指定した場合、第一引数”message_hash->pdata”に入力された SHA-256 ハッシュ値の先頭 XXX ビット(=XXX/8 バイト)に対して、第三引数”key_index”に入力された秘密鍵の Wrappd Key から、ECDSA P-192、P-224、P-256 に従い署名文を第二引数”signature”に書き出します。

(4) R_TSIP_EcdsaP384SignatureGenerate を使用する場合

第一引数”message_hash->data_type”でメッセージを指定した場合、第一引数”message_hash->pdata”に入力されたメッセージ文を SHA-384 ハッシュ計算し、第三引数”key_index”に入力された秘密鍵の Wrappd Key から、ECDSA P-384 に従い署名文を第二引数”signature”に書き出します。

メッセージ入力を使用する場合、4.3 を参照して SHA384 のユーザ定義関数を用意してください。

第一引数”message_hash->data_type”でハッシュ値を指定した場合、第一引数”message_hash->pdata”に入力された SHA-384 ハッシュ値の 48 バイト全てに対して、第三引数”key_index”に入力された秘密鍵の Wrappd Key から、ECDSA P-384 に従い署名文を第二引数”signature”に書き出します。

key_index の生成方法については 3.7.1 鍵の注入と更新を参照してください。

Reentrant

非対応

4.2.7.7 R_TSIP_EcdsaP*XXX*SignatureVerification

Format

(1)
```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_EcdsaP192SignatureVerification(
        tsip_ecdsa_byte_data_t *signature,
        tsip_ecdsa_byte_data_t *message_hash,
        tsip_ecc_public_key_index_t *key_index
)
```

(2)
```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_EcdsaP224SignatureVerification(
        tsip_ecdsa_byte_data_t *signature,
        tsip_ecdsa_byte_data_t *message_hash,
        tsip_ecc_public_key_index_t *key_index
)
```

(3)
```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_EcdsaP256SignatureVerification(
        tsip_ecdsa_byte_data_t *signature,
        tsip_ecdsa_byte_data_t *message_hash,
        tsip_ecc_public_key_index_t *key_index
)
```

(4)
```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_EcdsaP384SignatureVerification(
        tsip_ecdsa_byte_data_t *signature,
        tsip_ecdsa_byte_data_t *message_hash,
        tsip_ecc_public_key_index_t *key_index
)
```

Parameters

signature	入力	検証する署名文情報
pdata		署名文を格納する配列のポインタを指定 署名形式は以下の通りです。 (1) ”0 padding(64bit) || 署名 r(192bit) || 0 padding(64bit) || 署名 s(192bit)” (2) ”0 padding(32bit) || 署名 r(224bit) || 0 padding(32bit) || 署名 s(224bit)” (3) ”署名 r(256bit) || 署名 s(256bit)” (4) ”署名 r(384bit) || 署名 s(384bit)”
message_hash	入力	検証するメッセージ文またはハッシュ値情報
pdata		メッセージまたはハッシュ値を格納している配列のポインタを指定
data_length		配列の有効データ長(メッセージの場合のみ指定)
data_type		message_hash のデータ種別を選択 0：メッセージ(ハッシュ計算を本関数で実行) 1：ハッシュ値(ハッシュ計算結果を入力)
key_index	入力	Wrappd Key (1) ECC P-192 公開鍵 (2) ECC P-224 公開鍵 (3) ECC P-256 公開鍵 (4) ECC P-384 公開鍵

Return Values

TSIP_SUCCESS :	正常終了
TSIP_ERR_RESOURCE_CONFLICT:	本処理に必要なハードウェアリソースが他の処理で使用されていることによるリソース衝突が発生
TSIP_ERR_KEY_SET	異常な Wrapped Key が入力された
TSIP_ERR_FAIL	内部エラーが発生、もしくは署名検証失敗
TSIP_ERR_PARAMETER	入力データが不正

Description

(1) R_TSIP_EcdsaP192SignatureVerification、(2) R_TSIP_EcdsaP224SignatureVerification、
(3) R_TSIP_EcdsaP256SignatureVerification を使用する場合

第二引数”message_hash->data_type”でメッセージを指定した場合、第二引数”message_hash->pdata”に入力されたメッセージ文を SHA-256 ハッシュ計算し、第三引数”key_index”に入力された公開鍵の Wrapped Key から、ECDSA P-192、P-224、P-256 に従い第一引数”signature”に入力された署名文との検証をします。

第二引数”message_hash->data_type”でハッシュ値を指定した場合、第二引数”message_hash->pdata”に入力された SHA-256 ハッシュ値の先頭 XXX ビット(=XXX/8 バイト)に対して、第三引数”key_index”に入力された公開鍵の Wrapped Key から、ECDSA P-192、P-224、P-256 に従い第一引数”signature”に入力された署名文との検証をします。

(4) R_TSIP_EcdsaP384SignatureVerification を使用する場合

第二引数”message_hash->data_type”でメッセージを指定した場合、第二引数”message_hash->pdata”に入力されたメッセージ文を SHA-384 ハッシュ計算し、第三引数”key_index”に入力された公開鍵の Wrapped Key から、ECDSA P-384 に従い第一引数”signature”に入力された署名文との検証をします。

メッセージ入力を使用する場合、4.3 を参照して SHA384 のユーザ定義関数を用意してください。

第二引数”message_hash->data_type”でハッシュ値を指定した場合、第二引数”message_hash->pdata”に入力された SHA-384 ハッシュ値の 48 バイト全てに対して、第三引数”key_index”に入力された公開鍵の Wrapped Key から、ECDSA P-384 に従い第一引数”signature”に入力された署名文との検証をします。

key_index の生成方法については、3.7.1 鍵の注入と更新を参照してください。

Reentrant

非対応

4.2.8　HASH

4.2.8.1　R_TSIP_Sha*XXX*Init

Format

(1)
```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_Sha1Init(
        tsip_sha_md5_handle_t *handle
)
```

(2)
```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_Sha256Init(
        tsip_sha_md5_handle_t *handle
)
```

Parameters

handle	出力	SHA 用ハンドラ(ワーク領域)

Return Values

TSIP_SUCCESS :	正常終了

Description

R_TSIP_Sha*XXX*Init()関数は、SHA1 もしくは SHA256 ハッシュ演算を実行する準備を行い、その結果を第一引数”handle”に書き出します。”handle”は、続く R_TSIP_Sha*XXX*Update()関数および R_TSIP_Sha*XXX*Final()関数で引数として使用されます。

Reentrant

非対応

4.2.8.2 R_TSIP_Sha*XXX*Update

Format

(1)
```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_Sha1Update(
        tsip_sha_md5_handle_t *handle,
        uint8_t *message,
        uint32_t message_length
)
```

(2)
```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_Sha256Update(
        tsip_sha_md5_handle_t *handle,
        uint8_t *message,
        uint32_t message_length
)
```

Parameters

handle	入力/出力	SHA 用ハンドラ(ワーク領域)
message	入力	メッセージ領域
message_length	入力	メッセージバイト長

Return Values

TSIP_SUCCESS :	正常終了
TSIP_ERR_RESOURCE_CONFLICT:	本処理に必要なハードウェアリソースが他の処理で使用されていることによるリソース衝突が発生
TSIP_ERR_PARAMETER:	不正なハンドルが入力された
TSIP_ERR_PROHIBIT_FUNCTION:	不正な関数が呼び出された

Description

R_TSIP_Sha*XXX*Update()関数は、第一引数”handle”で指定されたハンドルを使用し、第二引数の”message”と第三引数の”message_length”からハッシュ値を演算し、途中経過を第一引数”handle”に書き出します(R_TSIP_GetCurrentHashDigestValue()関数で取得可能)。メッセージ入力が完了した後は、R_TSIP_Sha*XXX*Final()を呼び出してください。

Reentrant

非対応

4.2.8.3 R_TSIP_Sha*XXX*Final

Format

(1)
```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_Sha1Final(
        tsip_sha_md5_handle_t *handle,
        uint8_t *digest,
        uint32_t *digest_length
)
```

(2)
```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_Sha256Final(
        tsip_sha_md5_handle_t *handle,
        uint8_t *digest,
        uint32_t *digest_length
)
```

Parameters

handle	入力	SHA 用ハンドル(ワーク領域)
digest	出力	hash データ領域
digest_length	出力	hash データバイト長 (1) SHA1 : 20byte (2) SHA256 : 32byte

Return Values

TSIP_SUCCESS :	正常終了
TSIP_ERR_RESOURCE_CONFLICT:	本処理に必要なハードウェアリソースが他の処理で使用されていることによるリソース衝突が発生
TSIP_ERR_PARAMETER:	不正なハンドルが入力された
TSIP_ERR_PROHIBIT_FUNCTION:	不正な関数が呼び出された

Description

R_TSIP_Sha*XXX*Final()関数は、第一引数”handle”で指定されたハンドルを使用し、第二引数の”digest”に演算結果、第三引数”digest_length”に演算結果の長さを書き出します。

Reentrant

非対応

4.2.8.4 R_TSIP_Md5Init

Format

```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_Md5Init(
        tsip_sha_md5_handle_t *handle
)
```

Parameters

handle	出力	MD5 用ハンドラ(ワーク領域)

Return Values

TSIP_SUCCESS :	正常終了

Description

R_TSIP_Md5Init()関数は、MD5 ハッシュ演算を実行する準備を行い、その結果を第一引数”handle”に書き出します。handle は、続く R_TSIP_Md5Update()関数および R_TSIP_Md5Final()関数で引数として使用されます。

Reentrant

非対応

4.2.8.5 R_TSIP_Md5Update

Format

```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_Md5Update(
        tsip_sha_md5_handle_t *handle,
        uint8_t *message,
        uint32_t message_length
)
```

Parameters

handle	入力/出力	MD5 用ハンドラ(ワーク領域)
message	入力	メッセージ領域
message_length	入力	メッセージバイト長

Return Values

TSIP_SUCCESS :	正常終了
TSIP_ERR_RESOURCE_CONFLICT:	本処理に必要なハードウェアリソースが他の処理で使用されていることによるリソース衝突が発生
TSIP_ERR_PARAMETER:	不正なハンドルが入力された
TSIP_ERR_PROHIBIT_FUNCTION:	不正な関数が呼び出された

Description

R_TSIP_Md5Update()関数は、第一引数”handle”で指定されたハンドルを使用し、第二引数の”message”と第三引数の”message_length”からハッシュ値を演算し、途中経過を第一引数”handle”に書き出します(R_TSIP_GetCurrentHashDigestValue()関数で取得可能)。メッセージ入力が完了した後は、R_TSIP_Md5Final()を呼び出してください。

Reentrant

非対応

4.2.8.6　R_TSIP_Md5Final

Format

```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_Sha1Final(
        tsip_sha_md5_handle_t *handle,
        uint8_t *digest,
        uint32_t *digest_length
)
```

Parameters

handle	入力	MD5 用ハンドル(ワーク領域)
digest	出力	hash データ領域
digest_length	出力	hash データバイト長(16byte)

Return Values

TSIP_SUCCESS :	正常終了
TSIP_ERR_RESOURCE_CONFLICT:	本処理に必要なハードウェアリソースが他の処理で使用されていることによるリソース衝突が発生
TSIP_ERR_PARAMETER:	不正なハンドルが入力された
TSIP_ERR_PROHIBIT_FUNCTION:	不正な関数が呼び出された

Description

R_TSIP_Md5Final()関数は、第一引数”handle”で指定されたハンドルを使用し、第二引数の”digest”に演算結果、第三引数”digest_length”に演算結果の長さを書き出します。

Reentrant

非対応

4.2.8.7　R_TSIP_GetCurrentHashDigestValue

Format

```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_GetCurrentHashDigestValue(
        tsip_sha_md5_handle_t *handle,
        uint8_t *digest,
        uint32_t *digest_length
)
```

Parameters

handle	入力	SHA,MD5 用ハンドラ(ワーク領域)
digest	出力	入力済みデータに対するハッシュ値データ領域
digest_length	出力	入力済みデータに対するハッシュ値データ長 (16, 20, 32 byte)

Return Values

TSIP_SUCCESS :	正常終了
TSIP_ERR_PARAMETER:	不正なハンドルが入力された
TSIP_ERR_PROHIBIT_FUNCTION:	不正な関数が呼び出された

Description

本関数は、引数”handle”で指定されたハンドルを使用し、引数”digest”に各 Update()関数【注】実行後の入力済みデータに対するハッシュ値データ、引数”digest_length”にデータ長を出力します。

【注】R_TSIP_Sha1Update()、R_TSIP_Sha256Update()、または R_TSIP_Md5Update()関数

Reentrant

非対応

4.2.9 HMAC

4.2.9.1 R_TSIP_GenerateSha*XXX*HmacKeyIndex

Format

(1)
```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_GenerateSha1HmacKeyIndex (
        uint8_t *encrypted_provisioning_key,
        uint8_t *iv,
        uint8_t *encrypted_key,
        tsip_hmac_sha_key_index_t *key_index
)
```

(2)
```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_GenerateSha256HmacKeyIndex (
        uint8_t *encrypted_provisioning_key,
        uint8_t *iv,
        uint8_t *encrypted_key,
        tsip_hmac_sha_key_index_t *key_index
)
```

Parameters

encrypted_provisioning_key	入力	W-UFPK
iv	入力	encrypted_key 生成時に使用した初期ベクタ
encrypted_key	入力	UFPK で暗号化された Encrypted Key
key_index	出力	Wrapped Key

Return Values

TSIP_SUCCESS :	正常終了
TSIP_ERR_FAIL:	内部エラーが発生
TSIP_ERR_RESOURCE_CONFLICT:	本処理に必要なハードウェアリソースが他の処理で使用されていることによるリソース衝突が発生

Description

SHA1-HMAC もしくは SHA256-HMAC の鍵の Wrapped Key を出力するための API です。

encrypted_key に入力する UPFK で暗号化するデータのフォーマットは、7.3.6 HMAC を参照してください。

encrypted_provisioning_key, iv, encrypted_key の説明、および key_index の使用方法については、3.7.1 鍵の注入と更新を参照してください。

Reentrant

非対応

4.2.9.2 R_TSIP_UpdateSha*XXX*HmacKeyIndex

Format

(1)
```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_UpdateSha1HmacKeyIndex (
        uint8_t *iv,
        uint8_t *encrypted_key,
        tsip_hmac_sha_key_index_t *key_index
)
```

(2)
```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_UpdateSha256HmacKeyIndex (
        uint8_t *iv,
        uint8_t *encrypted_key,
        tsip_hmac_sha_key_index_t *key_index
)
```

Parameters

iv	入力	encrypted_key 生成時に使用した初期ベクタ
encrypted_key	入力	KUK で暗号化された Encrypted Key
key_index	出力	Wrapped Key

Return Values

TSIP_SUCCESS :	正常終了
TSIP_ERR_FAIL:	内部エラーが発生
TSIP_ERR_RESOURCE_CONFLICT:	本処理に必要なハードウェアリソースが他の処理で使用されていることによるリソース衝突が発生

Description

SHA-HMAC の鍵の Wrapped Key を更新するための API です。

encrypted_key に入力する KUK で暗号化するデータのフォーマットは、7.3.6 HMAC を参照してください。

encrypted_provisioning_key, iv, encrypted_key の説明、および key_index の使用方法については、3.7.1 鍵の注入と更新を参照してください。

Reentrant

非対応

4.2.9.3 R_TSIP_Sha*XXX*HmacGenerateInit

Format

(1) #include “r_tsip_rx_if.h”

```
e_tsip_err_t R_TSIP_Sha1HmacGenerateInit(
        tsip_hmac_sha_handle_t *handle,
        tsip_hmac_sha_key_index_t *key_index
)
```

(2) #include “r_tsip_rx_if.h”

```
e_tsip_err_t R_TSIP_Sha256HmacGenerateInit(
        tsip_hmac_sha_handle_t *handle,
        tsip_hmac_sha_key_index_t *key_index
)
```

Parameters

handle	出力	SHA-HMAC 用ハンドラ(ワーク領域)
key_index	入力	Wrapped Key

Return Values

TSIP_SUCCESS :	正常終了
TSIP_ERR_RESOURCE_CONFLICT:	本処理に必要なハードウェアリソースが他の処理で使用されていることによるリソース衝突が発生
TSIP_ERR_KEY_SET:	異常な Wrapped Key が入力された

Description

R_TSIP_Sha*XXX*HmacGenerateInit()関数は、第二引数の”key_index”を用い SHA1-HMAC もしくは SHA256-HMAC 演算を実行する準備を行い、その結果を第一引数”handle”に書き出します。”key_index”には、TLS 連携機能の場合、R_TSIP_TlsGenerateSessionKey()関数で生成された Wrapped Key を使用してください。”handle”は続く R_TSIP_Sha*XXX*HmacGenerateUpdate()関数や、R_TSIP_Sha*XXX*HmacGenerateFinal()関数の引数で使用します。

key_index の生成方法については、3.7.1 鍵の注入と更新を参照してください。

Reentrant

非対応

4.2.9.4 R_TSIP_Sha*XXX*HmacGenerateUpdate

Format

(1) #include “r_tsip_rx_if.h”

```
e_tsip_err_t R_TSIP_Sha1HmacGenerateUpdate(
        tsip_hmac_sha_handle_t *handle,
        uint8_t *message,
        uint32_t message_length
)
```

(2) #include “r_tsip_rx_if.h”

```
e_tsip_err_t R_TSIP_Sha256HmacGenerateUpdate(
        tsip_hmac_sha_handle_t *handle,
        uint8_t *message,
        uint32_t message_length
)
```

Parameters

handle	入力/出力	SHA-HMAC 用ハンドラ(ワーク領域)
message	入力	メッセージ領域
message_length	入力	メッセージバイト長

Return Values

TSIP_SUCCESS :	正常終了
TSIP_ERR_PARAMETER:	不正なハンドルが入力された
TSIP_ERR_PROHIBIT_FUNCTION:	不正な関数が呼び出された

Description

R_TSIP_Sha*XXX*HmacGenerateUpdate()関数は、第一引数”handle”で指定されたハンドルを使用し、第二引数の”message”と第三引数の”message_length”からハッシュ値を演算し、途中経過を第一引数”handle”に書き出します。メッセージ入力が完了した後は、R_TSIP_Sha*XXX*HmacGenerateFinal()を呼び出してください。

Reentrant

非対応

4.2.9.5 R_TSIP_Sha*XXX*HmacGenerateFinal

Format

(1) #include “r_tsip_rx_if.h”

```
e_tsip_err_t R_TSIP_Sha1HmacGenerateFinal(
        tsip_hmac_sha_handle_t *handle,
        uint8_t *mac
)
```

(2) #include “r_tsip_rx_if.h”

```
e_tsip_err_t R_TSIP_Sha256HmacGenerateFinal(
        tsip_hmac_sha_handle_t *handle,
        uint8_t *mac
)
```

Parameters

handle	入力	SHA-HMAC 用ハンドル(ワーク領域)
mac	出力	HMAC 領域 (1) SHA1-HMAC : 20byte (2) SHA256-HMAC : 32byte

Return Values

TSIP_SUCCESS :	正常終了
TSIP_ERR_FAIL:	内部エラーが発生
TSIP_ERR_PARAMETER:	不正なハンドルが入力された
TSIP_ERR_PROHIBIT_FUNCTION:	不正な関数が呼び出された

Description

R_TSIP_Sha*XXX*HmacGenerateFinal()関数は、第一引数”handle”で指定されたハンドルを使用し、第二引数の”mac”に演算結果を書き出します。

Reentrant

非対応

4.2.9.6 R_TSIP_Sha*XXX*HmacVerifyInit

Format

(1) #include “r_tsip_rx_if.h”

```
e_tsip_err_t R_TSIP_Sha1HmacVerifyInit(
        tsip_hmac_sha_handle_t *handle,
        tsip_hmac_sha_key_index_t *key_index
    )
```

(2) #include “r_tsip_rx_if.h”

```
e_tsip_err_t R_TSIP_Sha256HmacVerifyInit(
        tsip_hmac_sha_handle_t *handle,
        tsip_hmac_sha_key_index_t *key_index
    )
```

Parameters

handle	出力	SHA-HMAC 用ハンドラ(ワーク領域)
key_index	入力	Wrapped Key

Return Values

TSIP_SUCCESS :	正常終了
TSIP_ERR_RESOURCE_CONFLICT:	本処理に必要なハードウェアリソースが他の処理で使用されていることによるリソース衝突が発生
TSIP_ERR_KEY_SET:	異常な Wrapped Key が入力された

Description

R_TSIP_Sha*XXX*HmacVerifyInit()関数は、第一引数の”key_index”を用い SHA1-HMAC もしくは SHA256-HMAC 演算を実行する準備を行い、その結果を第一引数”handle”に書き出します。”key_index”には、TLS 連携機能で使用する場合、R_TSIP_TlsGenerateSessionKey()関数で生成された Wrapped Key を使用してください。”handle”は続く R_TSIP_Sha*XXX*HmacVerifyUpdate()関数や、R_TSIP_Sha*XXX*HmacVerifyFinal()関数の引数で使用します。

key_index の生成方法については、3.7.1 鍵の注入と更新を参照してください

Reentrant

非対応

4.2.9.7 R_TSIP_Sha*XXX*HmacVerifyUpdate

Format

(1) #include “r_tsip_rx_if.h”

```
e_tsip_err_t R_TSIP_Sha1HmacVerifyUpdate(
        tsip_hmac_sha_handle_t *handle,
        uint8_t *message,
        uint32_t message_length
)
```

(2) #include “r_tsip_rx_if.h”

```
e_tsip_err_t R_TSIP_Sha256HmacVerifyUpdate(
        tsip_hmac_sha_handle_t *handle,
        uint8_t *message,
        uint32_t message_length
)
```

Parameters

handle	入力/出力	SHA-HMAC 用ハンドラ(ワーク領域)
message	入力	メッセージ領域
message_length	入力	メッセージバイト長

Return Values

TSIP_SUCCESS :	正常終了
TSIP_ERR_PARAMETER:	不正なハンドルが入力された
TSIP_ERR_PROHIBIT_FUNCTION:	不正な関数が呼び出された

Description

R_TSIP_Sha*XXX*HmacVerifyUpdate()関数は、第一引数”handle”で指定されたハンドルを使用し、第二引数の”message”と第三引数の”message_length”からハッシュ値を演算し、途中経過を第一引数”handle”に書き出します。メッセージ入力が完了した後は、R_TSIP_Sha*XXX*HmacVerifyFinal()を呼び出してください。

Reentrant

非対応

4.2.9.8 R_TSIP_Sha*XXX*HmacVerifyFinal

Format

(1) #include “r_tsip_rx_if.h”

```
e_tsip_err_t R_TSIP_Sha1HmacVerifyFinal(
        tsip_hmac_sha_handle_t *handle,
        uint8_t *mac,
        uint32_t mac_length
)
```

(2) #include “r_tsip_rx_if.h”

```
e_tsip_err_t R_TSIP_Sha256HmacVerifyFinal(
        tsip_hmac_sha_handle_t *handle,
        uint8_t *mac,
        uint32_t mac_length
)
```

Parameters

handle	入力	SHA-HMAC 用ハンドル(ワーク領域)
mac	入力	HMAC 領域
mac_length	入力	HMAC 長

Return Values

TSIP_SUCCESS :	正常終了
TSIP_ERR_FAIL:	内部エラーが発生
TSIP_ERR_PARAMETER:	不正なハンドルが入力された
TSIP_ERR_PROHIBIT_FUNCTION:	不正な関数が呼び出された

Description

R_TSIP_Sha*XXX*HmacVerifyFinal()関数は、第一引数”handle”で指定されたハンドルを使用し、第二引数の”mac”と第三引数の”mac_length”から mac 値の検証を行います。“mac_length”の単位は byte で SHA1-HMAC の場合は 4 以上 20 以下、SHA256-HMAC の場合は 4 以上 32 以下の値を入力してください。

Reentrant

非対応

4.2.10 DH

4.2.10.1 R_TSIP_Rsa2048DhKeyAgreement

Format

```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_Rsa2048DhKeyAgreement(
        tsip_aes_key_index_t *key_index,
        tsip_rsa2048_private_key_index_t *sender_private_key_index,
        uint8_t *message,
        uint8_t *receiver_modulus,
        uint8_t *sender_modulus
)
```

Parameters

key_index	入力	AES-128 CMAC 演算 Wrapped Key
sender_private_key_index	入力	DH 演算で使用する秘密鍵の Wrapped Key 秘密鍵の Wrapped Key に含まれる秘密鍵 d を TSIP 内部で復号し、利用します
message	入力	メッセージ(2048bit) sender_private_key_index に含まれる素数(d)より小さい値を設定してください
receiver_modulus	入力	Receiver が計算したべき乗剰余演算結果 + MAC 2048bit べき乗剰余演算 || 128bit
sender_modulus	出力	Sender が計算したべき乗剰余演算結果 + MAC 2048bit べき乗剰余演算 || 128bit

Return Values

TSIP_SUCCESS :	正常終了
TSIP_ERR_KEY_SET:	異常な Wrapped Key が入力された
TSIP_ERR_RESOURCE_CONFLICT:	本処理に必要なハードウェアリソースが他の処理で使用されていることによるリソース衝突が発生
TSIP_ERR_FAIL:	内部エラーが発生

Description

RSA-2048 による DH 演算を実施します。

Sender は TSIP、Receiver は鍵交換相手を示します。

Reentrant

非対応

4.2.11 ECDH

4.2.11.1 R_TSIP_EcdhP256Init

Format

```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_EcdhP256Init (
        tsip_ecdh_handle_t *handle,
        uint32_t key_type,
        uint32_t use_key_id
  )
```

Parameters

handle	出力	ECDH 用ハンドラ(ワーク領域)
key_type	入力	鍵交換の種類　0：ECDHE 1：ECDH
user_key_id	入力	0：key_id 不使用 1：key_id 使用

Return Values

TSIP_SUCCESS :	正常終了
TSIP_ERR_PARAMETER:	入力データが不正

Description

R_TSIP_EcdhP256Init()関数は、ECDH 鍵交換を演算する準備を行い、その結果を第一引数”handle”に書き出します。”handle”は、続く R_TSIP_EcdhP256ReadPublicKey()、R_TSIP_EcdhP256MakePublicKey()、R_TSIP_EcdhP256CalculateSharedSecretIndex()、R_TSIP_EcdhP256KeyDerivation()関数で引数として使用されます。

第二引数の”key_type”では ECDH 鍵交換の種類を選択してください。ECDHE では、R_TSIP_EcdhP256MakePublicKey()関数で TSIP の乱数生成機能を使い ECC P-256 の鍵ペアを生成します。ECDH では、鍵交換ではあらかじめ注入した鍵を使用します。

第三引数の”use_key_id”は、鍵交換の際に key_id を使用する場合”1”を入力してください。key_id はスマートメータ向け規格の DLMS/COSEM 用途です。

key_index の生成方法については 3.7.1 鍵の注入と更新を参照してください。

Reentrant

非対応

4.2.11.2 R_TSIP_EcdhP256ReadPublicKey

Format

```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_EcdhP256ReadPublicKey (
        tsip_ecdh_handle_t *handle,
        tsip_ecc_public_key_index_t *public_key_index,
        uint8_t *public_key_data,
        tsip_ecdsa_byte_data_t *signature,
        tsip_ecc_public_key_index_t *key_index
)
```

Parameters

handle	入力/出力	ECDH 用ハンドラ(ワーク領域)	
public_key_index	入力	署名検証向けの公開鍵の Wrapped Key	
public_key_data	入力	key_id を使用しない場合	ECC P-256 公開鍵 (512bit)
		key_id を使用する場合	key_id (8bit) || 公開鍵 s(512bit)
signature	入力	public_key_data の ECDSA P-256 署名	
key_index	出力	public_key_data の Wrapped Key	

Return Values

TSIP_SUCCESS :	正常終了
TSIP_ERR_RESOURCE_CONFLICT:	本処理に必要なハードウェアリソースが他の処理で使用されていることによるリソース衝突が発生
TSIP_ERR_KEY_SET:	異常な Wrapped Key が入力された
TSIP_ERR_FAIL:	内部エラーが発生、もしくは署名検証失敗
TSIP_ERR_PARAMETER:	不正なハンドルが入力された
TSIP_ERR_PROHIBIT_FUNCTION:	不正な関数が呼び出された

Description

R_TSIP_EcdhP256ReadPublicKey()関数は ECDH 鍵交換相手の ECC P-256 public key の署名を検証し、署名が正しければ第 5 引数に public_key_data の Wrapped Key を出力します。

第一引数”handle”は続く R_TSIP_EcdhP256CalculateSharedSecretIndex()関数の引数で使用します。

key_index は R_TSIP_EcdhP256CalculateSharedSecretIndex()関数で Z を計算するための入力として使用します。

key_index の生成方法については 3.7.1 鍵の注入と更新を参照してください。

Reentrant

非対応

4.2.11.3 R_TSIP_EcdhP256MakePublicKey

Format

```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_EcdhP256MakePublicKey (
        tsip_ecdh_handle_t *handle,
        tsip_ecc_public_key_index_t *public_key_index,
        tsip_ecc_private_key_index_t *private_key_index,
        uint8_t *public_key,
        tsip_ecdsa_byte_data_t *signature,
        tsip_ecc_private_key_index_t *key_index
)
```

Parameters

handle	入力/出力	ECDH 用ハンドラ(ワーク領域) key_id を使用する場合は、 R_TSIP_EcdhP256Init()の実行後、handle->key_id に入力してください。
public_key_index	入力	ECDHE の場合は NULL ポインタを入力してください。 ECDH の場合は、ECC P-256 公開鍵の Wrapped Key を入力してください
private_key_index	入力	署名生成向けの ECC P-256 秘密鍵
public_key	出力	鍵交換用ユーザ公開鍵(512bit) key_id を使用する場合 key_id (8bit) || 公開鍵 (512bit) || 0 padding(24bit)
signature	出力	署名文格納先情報
->pdata		：署名文を格納する配列のポインタを指定 署名形式は”署名 r(256bit) || 署名 s(256bit)”
->data_length		：データ長(バイト単位)
key_index	出力	ECDHE の場合は乱数から生成された秘密鍵の Wrapped Key ECDH の場合は出力されません。

Return Values

TSIP_SUCCESS :	正常終了
TSIP_ERR_RESOURCE_CONFLICT:	本処理に必要なハードウェアリソースが他の処理で使用されていることによるリソース衝突が発生
TSIP_ERR_KEY_SET:	異常な Wrapped Key が入力された
TSIP_ERR_FAIL:	内部エラーが発生
TSIP_ERR_PARAMETER:	不正なハンドルが入力された
TSIP_ERR_PROHIBIT_FUNCTION:	不正な関数が呼び出された

Description

R_TSIP_EcdhP256MakePublicKey()関数は、一時的な鍵ペア(Ephemeral Key)の生成をおこない、生成した鍵もしくは入力した鍵を使用して署名を生成します。生成された署名はスマートメータ向け規格の DLMS/COSEM 用途です。

R_TSIP_EcdhP256Init()関数の key_type で ECDHE を指定した場合、TSIP の乱数生成機能を使い ECC P-256 の鍵ペアを生成します。公開鍵は public_key へ出力し、秘密鍵は key_index に出力されます。

R_TSIP_EcdhP256Init()関数の key_type で ECDH を指定した場合、public_key には public_key_index で入力した公開鍵を出力します。key_index には何も出力されません。

第一引数”handle”は続く R_TSIP_EcdhP256CalculateSharedSecretIndex()関数の引数で使用します。

key_index は R_TSIP_EcdhP256CalculateSharedSecretIndex()関数で Z を計算するための入力として使用します。

key_index の生成方法については 3.7.1 鍵の注入と更新を参照してください。

Reentrant

非対応

4.2.11.4 R_TSIP_EcdhP256CalculateSharedSecretIndex

Format

```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_EcdhP256CalculateSharedSecretIndex (
        tsip_ecdh_handle_t *handle,
        tsip_ecc_public_key_index_t *public_key_index,
        tsip_ecc_private_key_index_t *private_key_index,
        tsip_ecdh_key_index_t *shared_secret_index
)
```

Parameters

handle	入力/出力	ECDH 用ハンドラ(ワーク領域)
public_key_index	入力	R_TSIP_EcdhP256ReadPublicKey()で署名検証した公開鍵の Wrapped Key
private_key_index	入力	秘密鍵の Wrapped Key
shared_secret_index	出力	ECDH 鍵共有で計算した共有秘密 ”Z”の Wrapped Key

Return Values

TSIP_SUCCESS :	正常終了
TSIP_ERR_RESOURCE_CONFLICT:	本処理に必要なハードウェアリソースが他の処理で使用されていることによるリソース衝突が発生
TSIP_ERR_KEY_SET:	異常な Wrapped Key が入力された
TSIP_ERR_FAIL:	内部エラーが発生
TSIP_ERR_PARAMETER:	不正なハンドルが入力された
TSIP_ERR_PROHIBIT_FUNCTION:	不正な関数が呼び出された

Description

R_TSIP_EcdhP256CalculateSharedSecretIndex()関数は、鍵交換相手の公開鍵と自身の秘密鍵から ECDH 鍵交換アルゴリズムで共有秘密”Z”の Wrapped Key を出力します。

第二引数の public_key_index には、R_TSIP_EcdhP256ReadPublicKey()関数で署名検証した公開鍵の Wrapped Key である key_index を入力してください。

第三引数の private_key_index には、R_TSIP_EcdhP256Init()の key_type が 0 の場合には、R_TSIP_EcdhP256MakePublicKey()関数の出力の乱数から生成された秘密鍵の Wrapped Key である key_index、key_type が 0 以外の場合には、R_TSIP_EcdhP256MakePublicKey()関数の第二引数と対になる秘密鍵の Wrapped Key を入力してください。

shared_secret_index は、続く R_TSIP_EcdhP256KeyDerivation()関数で Wrapped Key を出力するための鍵材料として使用します。

key_index の生成方法については 3.7.1 鍵の注入と更新を参照してください。

Reentrant

非対応

4.2.11.5 R_TSIP_EcdhP256KeyDerivation

Format

```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_EcdhP256KeyDerivation (
        tsip_ecdh_handle_t *handle,
        tsip_ecdh_key_index_t *shared_secret_index,
        uint32_t key_type,
        uint32_t kdf_type,
        uint8_t *other_info,
        uint32_t other_info_length,
        tsip_hmac_sha_key_index_t *salt_key_index,
        tsip_aes_key_index_t *key_index
)
```

Parameters

handle	入力/出力	ECDH 用ハンドラ(ワーク領域)	
shared_secret_index	入力	R_TSIP_EcdhP256CalculateSharedSecretIndex で計算した”Z”の Wrapped Key	
key_type	入力	派生させる鍵の種類	0: AES-128
			1: AES-256
			2: SHA256-HMAC
kdf_type	入力	鍵導出の計算で使用するアルゴリズム	
			0: SHA256
			1: SHA256-HMAC
other_info	入力	鍵導出の計算で使用する追加データ AlgorithmID || PartyUInfo || PartyVInfo	
other_info_length	入力	other_info のバイト長(147 以下のバイト単位)	
salt_key_index	入力	Salt の Wrapped Key (kdf_type が 0 の場合は NULL を入力)	
key_index	出力	key_type に対応した Wrapped Key key_type:2 の場合、SHA256-HMAC の Wrapped Key を出力します。tsip_hmac_sha_key_index_t 型で事前に確保された領域の先頭アドレスを、(tsip_aes_key_index_t*)型でキャストして指定することが可能です。	

Return Values

TSIP_SUCCESS :	正常終了
TSIP_ERR_RESOURCE_CONFLICT:	本処理に必要なハードウェアリソースが他の処理で使用されていることによるリソース衝突が発生
TSIP_ERR_KEY_SET:	異常な Wrapped Key が入力された
TSIP_ERR_PARAMETER:	不正なハンドルが入力された
TSIP_ERR_PROHIBIT_FUNCTION:	不正な関数が呼び出された

Description

R_TSIP_EcdhP256KeyDerivation()関数は、R_TSIP_EcdhP256CalculateSharedSecretIndex()関数で計算した共有秘密”Z(shared_secret_index)”を鍵材料として、第三引数の key_type で指定した Wrapped Key

を導出します。鍵導出のアルゴリズムは、NIST SP800-56C の One-Step Key Derivation です。第四引数 kdf_type で、AES-128、AES-256 または SHA-256 HMAC を指定します。SHA-256 HMAC を指定する場合、第七引数 salt_key_index に、R_TSIP_GenerateSha256HmacKeyIndex()関数または R_TSIP_UpdateSha256HmacKeyIndex()関数で出力した Wrapped Key を指定します。

第五引数の other_info には鍵交換相手と共有している鍵導出のための固定値を入力してください。

第八引数の key_index は key_type に対応した Wrapped Key が出力されます。導出する Wrapped Key と、使用可能な関数の組合せを以下に示します。

導出する Wrapped Key	使用可能な関数
AES-128	AES128 全ての Init 関数、R_TSIP_Aes128KeyUnwrap()
AES-256	AES256 全ての Init 関数、R_TSIP_Aes256KeyUnwrap()
SHA256-HMAC	R_TSIP_Sha256HmacGenerateInit()、R_TSIP_Sha256HmacVerifyInit()

Reentrant

非対応

4.2.11.6 R_TSIP_EcdheP512KeyAgreement

Format

```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_EcdheP512KeyAgreement(
        tsip_aes_key_index_t *key_index,
        uint8_t *receiver_public_key,
        uint8_t *sender_public_key
)
```

Parameters

key_index	入力	AES-128 CMAC 演算用 Wrapped Key
receiver_public_key	入力	Receiver の Brainpool P512r1 公開鍵 Q(1024bit) || MAC(128bit)
sender_public_key	出力	Sender の Brainpool P512r1 公開鍵 Q(1024bit) || MAC(128bit)

Return Values

TSIP_SUCCESS :	正常終了
TSIP_ERR_RESOURCE_CONFLICT:	本処理に必要なハードウェアリソースが他の処理で使用されていることによるリソース衝突が発生
TSIP_ERR_KEY_SET:	異常な Wrapped Key が入力された
TSIP_ERR_FAIL:	内部エラーが発生

Description

Brainpool P512r1 を用いて鍵ペア生成の後、ECDHE 演算を行います。

Sender は TSIP、Receiver は鍵交換相手を示します。

Reentrant

非対応

4.2.12 KeyWrap

4.2.12.1 R_TSIP_Aes*XXX*KeyWrap

Format

(1)
```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_Aes128KeyWrap (
        tsip_aes_key_index_t *wrap_key_index,
        uint32_t target_key_type,
        tsip_aes_key_index_t *target_key_index,
        uint32_t *wrapped_key
)
```

(2)
```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_Aes256KeyWrap (
        tsip_aes_key_index_t *wrap_key_index,
        uint32_t target_key_type,
        tsip_aes_key_index_t *target_key_index,
        uint32_t *wrapped_key
)
```

Parameters

wrap_key_index	入力	(1) ラップに使用する AES128 鍵の Wrapped Key (2) ラップに使用する AES256 鍵の Wrapped Key
target_key_type	入力	ラップする対象の鍵の選択 0(R_TSIP_KEYWRAP_AES128): AES-128 2(R_TSIP_KEYWRAP_AES256): AES-256
target_key_index	入力	ラップする対象の Wrapped Key target_key_type 0 : 13 word size target_key_type 2 : 17 word size
wrapped_key	出力	ラップされた鍵 target_key_type 0 : 6 word size target_key_type 2 : 10 word size

Return Values

TSIP_SUCCESS :	正常終了
TSIP_ERR_RESOURCE_CONFLICT:	本処理に必要なハードウェアリソースが他の処理で使用されていることによるリソース衝突が発生
TSIP_ERR_KEY_SET	異常な Wrapped Key が入力された
TSIP_ERR_FAIL	内部エラーが発生

Description

R_TSIP_Aes*XXX*KeyWrap()関数は、第三引数に入力した target_key_index を第一引数の wrap_key_index を使いラップします。ラップされた鍵は第四引数の wrapped_key に書き出します。ラップのアルゴリズム RFC3394 に準拠します。ラップする対象の鍵は、第二引数の target_key_type で選択してください。

ラップに使用する鍵長が 128bit の場合は、R_TSIP_Aes128KeyWrap()、ラップに使用する鍵長が 256bit の場合は R_TSIP_Aes256KeyWrap()を使用します。

Reentrant

非対応

4.2.12.2 R_TSIP_Aes*XXX*KeyUnwrap

Format

(1)
```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_Aes128KeyUnwrap (
        tsip_aes_key_index_t *wrap_key_index,
        uint32_t target_key_type,
        uint32_t *wrapped_key,
        tsip_aes_key_index_t *target_key_index
)
```

(2)
```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_Aes256KeyUnwrap (
        tsip_aes_key_index_t *wrap_key_index,
        uint32_t target_key_type,
        uint32_t *wrapped_key,
        tsip_aes_key_index_t *target_key_index
)
```

Parameters

wrap_key_index	入力	(1) アンラップに使用する AES128 鍵の Wrapped Key (2) アンラップに使用する AES256 鍵の Wrapped Key
target_key_type	入力	アンラップする対象の鍵の選択 0(R_TSIP_KEYWRAP_AES128): AES-128 2(R_TSIP_KEYWRAP_AES256): AES-256
wrapped_key	入力	ラップされた鍵 target_key_type 0 : 6 word size target_key_type 2 : 10 word size
target_key_index	出力	Wrapped Key target_key_type 0 : 13word size target_key_type 2 : 17 word size

Return Values

TSIP_SUCCESS :	正常終了
TSIP_ERR_RESOURCE_CONFLICT:	本処理に必要なハードウェアリソースが他の処理で使用されていることによるリソース衝突が発生
TSIP_ERR_KEY_SET	異常な Wrapped Key が入力された
TSIP_ERR_FAIL	内部エラーが発生

Description

R_TSIP_Aes*XXX*KeyUnwrap 関数は、第三引数に入力した wrapped_key を第一引数の wrap_key_index を使いアンラップします。アンラップされた鍵は第四引数の target_key_index に書き出します。アンラップのアルゴリズム RFC3394 に準拠します。アンラップする対象の鍵は、第二引数の target_key_type で選択してください。

アンラップに使用する鍵長が 128bit の場合は、R_TSIP_Aes128KeyUnwrap()、アンラップに使用する鍵長が 256bit の場合は R_TSIP_Aes256KeyUnwrap()を使用します。

Reentrant

非対応

4.2.13 TLS（TLS1.2/1.3 共通）

Format 等で(1)(2)があるものについては、(1)はクライアント用、(2)はサーバ用です。

4.2.13.1 R_TSIP_GenerateTlsXXRsaPublicKeyIndex

Format

(1)
```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_GenerateTlsRsaPublicKeyIndex(
        uint8_t *encrypted_provisioning_key,
        uint8_t *iv,
        uint8_t *encrypted_key,
        tsip_tls_ca_certification_public_key_index_t *key_index
)
```
(2)
```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_GenerateTlsSVRsaPublicKeyIndex(
        uint8_t *encrypted_provisioning_key,
        uint8_t *iv,
        uint8_t *encrypted_key,
        tsip_tls_ca_certification_public_key_index_t *key_index
)
```

Parameters

encrypted_provisioning_key	入力	W-UFPK
iv	入力	encrypted_key 生成時に使用した初期化ベクタ
encrypted_key	入力	UFPK で暗号化された Encrypted Key
key_index	出力	TLS 連携機能で使用する 2048bit 長の RSA 公開鍵の Wrapped Key R_TSIP_TlsRegisterCaCertificationPublicKeyIndex の入力 key_index に使用してください。 (1)でクライアント用の Wrapped Key を作成した場合は、R_TSIP_Open の入力 key_index_1 に使用することも可能です。

Return Values

TSIP_SUCCESS	正常終了
TSIP_ERR_FAIL	内部エラーが発生
TSIP_ERR_RESOURCE_CONFLICT	本処理に必要なハードウェアリソースが他の処理で使用されていることによるリソース衝突が発生

Description

TLS 連携機能で使用する、RSA 2048bit の公開鍵の Wrapped Key を出力するための API です。

encrypted_key に入力する UFPK で暗号化するデータのフォーマットは、7.3.4.2 を参照してください。

encrypted_key と key_index は領域が重ならないように配置してください。

encrypted_provisioning_key, iv, encrypted_key の説明、および key_index の使用方法については、3.7.1 鍵の注入と更新を参照してください。

Reentrant

非対応

4.2.13.2 R_TSIP_UpdateTlsXXRsaPublicKeyIndex

Format

(1)
```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_UpdateTlsRsaPublicKeyKeyIndex(
        uint8_t *iv,
        uint8_t *encrypted_key,
        tsip_tls_ca_public_key_index_t *key_index
)
```
(2)
```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_UpdateTlsSVRsaPublicKeyKeyIndex(
        uint8_t *iv,
        uint8_t *encrypted_key,
        tsip_tls_ca_public_key_index_t *key_index
)
```

Parameters

iv	入力	encrypted_key 生成時に使用した初期ベクタ
encrypted_key	入力	KUK で暗号化された Encrypted Key
key_index	出力	TLS 連携機能で使用する RSA 2048bit 公開鍵の Wrapped Key R_TSIP_TlsRegisterCaCertificationPublicKeyIndex の入力 key_index に使用してください。 (1)でクライアント用の Wrapped Key を作成した場合は、R_TSIP_Open の入力 key_index_1 に使用することも可能です。

Return Values

TSIP_SUCCESS	正常終了
TSIP_ERR_FAIL	内部エラーが発生
TSIP_ERR_RESOURCE_CONFLICT	本処理に必要なハードウェアリソースが他の処理で使用されていることによるリソース衝突が発生

Description

TLS 連携機能で使用する、RSA 2048bit 公開鍵の Wrapped Key を更新するための API です。

encrypted_key に入力する KUK で暗号化するデータのフォーマットは、7.3.4.2 を参照してください。

encrypted_key と key_index は領域が重ならないように配置してください。

encrypted_provisioning_key, iv, encrypted_key の説明、および key_index の使用方法については、3.7.1 鍵の注入と更新を参照してください。

Reentrant

非対応

4.2.13.3 R_TSIP_TlsRegisterCaCertificationPublicKeyIndex

Format

```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_TlsRegisterCaCertificationPublicKeyIndex(
        e_tsip_tls_mode_t mode,
        tsip_tls_ca_certification_public_key_index_t *key_index
)
```

Parameters

mode	入力	登録する公開鍵を使用するエンティティーの種類 TSIP_TLS_MODE_CLIENT : クライアント TSIP_TLS_MODE_SERVER : サーバ
key_index	入力	TLS 連携機能で使用する RSA 2048bit 公開鍵の Wrapped Key R_TSIP_GenerateTlsRsaPublicKeyIndex、 R_TSIP_UpdateTlsRsaPublicKeyIndex、 R_TSIP_GenerateTlsSVRsaPublicKeyIndex または R_TSIP_UpdateTlsSVRsaPublicKeyIndex の出力 key_index を使用してください。

Return Values

TSIP_SUCCESS	正常終了
TSIP_ERR_KEY_SET	異常な Wrapped Key が入力された

Description

TLS 連携機能で使用する、ルート CA 証明書の束を検証するための公開鍵を登録するための API です。

クライアント用の公開鍵の登録については、R_TSIP_Open を使用して行うことも可能です。

Reentrant

非対応

4.2.13.4 R_TSIP_TlsXXRootCertificateVerification

Format

(1)
```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_TlsRootCertificateVerification(
        uint32_t *public_key_type,
        uint8_t *certificate,
        uint32_t certificate_length,
        uint32_t public_key_n_start_position,
        uint32_t public_key_n_end_position,
        uint32_t public_key_e_start_position,
        uint32_t public_key_e_end_position,
        uint8_t *signature,
        uint32_t *encrypted_root_public_key
)
```

(2)
```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_TlsSVRootCertificateVerification(
        uint32_t *public_key_type,
        uint8_t *certificate,
        uint32_t certificate_length,
        uint32_t public_key_n_start_position,
        uint32_t public_key_n_end_position,
        uint32_t public_key_e_start_position,
        uint32_t public_key_e_end_position,
        uint8_t *signature,
        uint32_t *encrypted_root_public_key
)
```

Parameters

public_key_type	入力	証明書に含まれている公開鍵の種類 0 : RSA 2048bit, 2: ECC P-256
certificate	入力	ルート CA 証明書の束(DER 形式)
certificate_length	入力	ルート CA 証明書の束のバイト長
public_key_n_start_position	入力	引数 certificate のアドレスを起点とした公開鍵の開始バイト位置 public_key_type 0 : n, 2 : Qx
public_key_n_end_position	入力	引数 certificate のアドレスを起点とした公開鍵の終了バイト位置 public_key_type 0 : n, 2 : Qx
public_key_e_start_position	入力	引数 certificate のアドレスを起点とした公開鍵の開始バイト位置 public_key_type 0 : e, 2 : Qy
public_key_e_end_position	入力	引数 certificate のアドレスを起点とした公開鍵の終了バイト位置 public_key_type 0 : e, 2 : Qy
signature	入力	ルート CA 証明書の束に対する署名データ 署名データサイズは 256 バイト 署名方式は「RSA2048 PSS with SHA256」
encrypted_root_public_key	出力	暗号化された ECDSA P256 もしくは RSA2048 公開鍵 R_TSIP_TlsXXCertificateVerification または R_TSIP_TlsXXCertificateVerificationExtension の入力 encrypted_input_public_key に使用してください。 public_key_type が 0 の場合 560 バイト、2 の場合 96 バイトが出力されます。

Return Values

TSIP_SUCCESS :	正常終了
TSIP_ERR_FAIL:	内部エラーが発生
TSIP_ERR_RESOURCE_CONFLICT:	本処理に必要なハードウェアリソースが他の処理で使用されていることによるリソース衝突が発生

Description

TLS 連携機能で使用する、ルート CA 証明書の束を検証するための API です。

Reentrant

非対応

4.2.13.5 R_TSIP_TlsXXCertificateVerification

Format

(1)
```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_TlsCertificateVerification(
        uint32_t *public_key_type,
        uint32_t *encrypted_input_public_key,
        uint8_t *certificate,
        uint32_t certificate_length,
        uint8_t *signature,
        uint32_t public_key_n_start_position,
        uint32_t public_key_n_end_position,
        uint32_t public_key_e_start_position,
        uint32_t public_key_e_end_position,
        uint32_t *encrypted_output_public_key
)
```

(2)
```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_TlsSVCertificateVerification(
        uint32_t *public_key_type,
        uint32_t *encrypted_input_public_key,
        uint8_t *certificate,
        uint32_t certificate_length,
        uint8_t *signature,
        uint32_t public_key_n_start_position,
        uint32_t public_key_n_end_position,
        uint32_t public_key_e_start_position,
        uint32_t public_key_e_end_position,
        uint32_t *encrypted_output_public_key
)
```

Parameters

public_key_type	入力	証明書に含まれている公開鍵の種類 0 : RSA 2048bit(sha256WithRsaEncryption 用) 1 : RSA 4096bit(sha256WithRsaEncryption 用) 2 : ECC P-256(ecdsa-with-SHA256 用) 3 : RSA 2048bit(RSASSA-PSS 用)
encrypted_input_public_key	入力	暗号化された公開鍵 R_TSIP_TlsXXRootCertificateVerification の出力 encrypted_root_public_key、または、 R_TSIP_TlsXXCertificateVerification または R_TSIP_TlsXXCertificateVerificationExtension の 出力 encrypted_output_public_key を使用してください。 データサイズ public_key_type 0,1,3:140 ワード(560 バイト), 2:24 ワード(96 バイト)
certificate	入力	証明書の束(DER 形式)
certificate_length	入力	証明書の束のバイト長
signature	入力	証明書の束に対する署名データ public_key_type:0 データサイズ 256 バイト 署名アルゴリズム sha256WithRSAEncryption public_key_type:1 データサイズ 512 バイト 署名アルゴリズム sha256WithRSAEncryption public_key_type:2

		データサイズ 64 バイト”r(256bit) || s(256bit)” 署名アルゴリズム ecdsa-with-SHA256 public_key_type:3 データサイズ 256 バイト 署名アルゴリズム RSASSA-PSS {sha256, mgf1SHA256, 0x20, trailerFieldBC}
public_key_n_start_position	入力	引数 certificate のアドレスを起点とした公開鍵の開始バイト位置 public_key_type 0,1,3 : n, 2 : Qx
public_key_n_end_position	入力	引数 certificate のアドレスを起点とした公開鍵の終了バイト位置 public_key_type 0,1,3 : n, 2 : Qx
public_key_e_start_position	入力	引数 certificate のアドレスを起点とした公開鍵の開始バイト位置 public_key_type 0,1,3 : e, 2 : Qy
public_key_e_end_position	入力	引数 certificate のアドレスを起点とした公開鍵の終了バイト位置 public_key_type 0,1,3 : e, 2 : Qy
encrypted_output_public_key	出力	暗号化された公開鍵 R_TSIP_TlsXXCertificateVerification または R_TSIP_TlsXXCertificateVerificationExtension の入力 encrypted_input_public_key、または R_TSIP_TlsEncryptPreMasterSecretWithRsa2048PublicKey または R_TSIP_TlsServersEphemeralEcdhPublicKeyRetrives の入力 encrypted_public_key に使用してください。 ただし public_key_type=1 選択時は R_TSIP_TlsXXCertificateVerification、R_TSIP_TlsXXCertificateVerificationExtension でのみ使用可能。 データサイズ public_key_type 0,1,3:140 ワード(560 バイト), 2:24 ワード(96 バイト)

Return Values

TSIP_SUCCESS :	正常終了
TSIP_ERR_FAIL:	内部エラーが発生
TSIP_ERR_RESOURCE_CONFLICT:	本処理に必要なハードウェアリソースが他の処理で使用されていることによるリソース衝突が発生

Description

TLS 連携機能で使用する、サーバ証明書、中間証明書の署名を検証するための API です。

R_TSIP_TlsXXCertificateVerificationExtension()関数と同じ用途で使用しますが、署名検証をする鍵のアルゴリズムと certificate から取り出す鍵のアルゴリズムが同一の場合には、こちらの関数を使用してください。

Reentrant

非対応

4.2.13.6 R_TSIP_TlsXXCertificateVerificationExtension

Format

(1)
```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_TlsCertificateVerificationExtension(
        uint32_t *public_key_type,
        uint32_t *public_key_output_type,
        uint32_t *encrypted_input_public_key,
        uint8_t *certificate,
        uint32_t certificate_length,
        uint8_t *signature,
        uint32_t public_key_n_start_position,
        uint32_t public_key_n_end_position,
        uint32_t public_key_e_start_position,
        uint32_t public_key_e_end_position,
        uint32_t *encrypted_output_public_key
)
```

(2)
```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_TlsSVCertificateVerificationExtension(
        uint32_t *public_key_type,
        uint32_t *public_key_output_type,
        uint32_t *encrypted_input_public_key,
        uint8_t *certificate,
        uint32_t certificate_length,
        uint8_t *signature,
        uint32_t public_key_n_start_position,
        uint32_t public_key_n_end_position,
        uint32_t public_key_e_start_position,
        uint32_t public_key_e_end_position,
        uint32_t *encrypted_output_public_key
)
```

Parameters

public_key_type	入力	入力する証明書に含まれている公開鍵の種類 0 : RSA 2048bit(sha256WithRsaEncryption 用) 1 : RSA 4096bit(sha256WithRsaEncryption 用) 2 : ECC P-256(ecdsa-with-SHA256 用) 3 : RSA 2048bit(RSASSA-PSS 用)
public_key_output_type	入力	certificate から出力する鍵の種類 0 : RSA 2048bit(sha256WithRsaEncryption 用) 1 : RSA 4096bit(sha256WithRsaEncryption 用) 2 : ECC P-256(ecdsa-with-SHA256 用) 3 : RSA 2048bit(RSASSA-PSS 用)
encrypted_input_public_key	入力	暗号化された公開鍵 R_TSIP_TlsXXRootCertificateVerification の出力 encrypted_root_public_key、または、 R_TSIP_TlsXXCertificateVerification または R_TSIP_TlsXXCertificateVerificationExtension の 出力 encrypted_output_public_key を使用してください。 データサイズ public_key_type 0,1,3:140 ワード(560 バイト), 2:24 ワード(96 バイト)
certificate	入力	証明書の束(DER 形式)
certificate_length	入力	証明書の束のバイト長
signature	入力	証明書の束に対する署名データ

		public_key_type:0 データサイズ 256 バイト 署名アルゴリズム sha256WithRSAEncryption public_key_type:1 データサイズ 512 バイト 署名アルゴリズム sha256WithRSAEncryption public_key_type:2 データサイズ 64 バイト”r(256bit) || s(256bit)” 署名アルゴリズム ecdsa-with-SHA256 public_key_type:3 データサイズ 256 バイト 署名アルゴリズム RSASSA-PSS {sha256, mgf1SHA256, 0x20, trailerFieldBC}
public_key_n_start_position	入力	引数 certificate のアドレスを起点とした公開鍵の開始バイト位置 public_key_type 0,1,3 : n, 2 : Qx
public_key_n_end_position	入力	引数 certificate のアドレスを起点とした公開鍵の終了バイト位置 public_key_type 0,1,3 : n, 2 : Qx
public_key_e_start_position	入力	引数 certificate のアドレスを起点とした公開鍵の開始バイト位置 public_key_type 0,1,3 : e, 2 : Qy
public_key_e_end_position	入力	引数 certificate のアドレスを起点とした公開鍵の終了バイト位置 public_key_type 0,1,3 : e, 2 : Qy
encrypted_output_public_key	出力	暗号化された公開鍵 R_TSIP_TlsXXCertificateVerification または R_TSIP_TlsXXCertificateVerificationExtension の入力 encrypted_input_public_key、または R_TSIP_TlsEncryptPreMasterSecretWithRsa2048PublicKey または R_TSIP_TlsServersEphemeralEcdhPublicKeyRetrives の入力 encrypted_public_key に使用してください。 ただし public_key_type=1 選択時は R_TSIP_TlsXXCertificateVerification、R_TSIP_TlsXXCertificateVerificationExtension でのみ使用可能。 データサイズ public_key_type 0,1,3:140 ワード(560 バイト), 2:24 ワード(96 バイト)

Return Values

TSIP_SUCCESS :	正常終了
TSIP_ERR_FAIL:	内部エラーが発生
TSIP_ERR_RESOURCE_CONFLICT:	本処理に必要なハードウェアリソースが他の処理で使用されていることによるリソース衝突が発生

Description

TLS 連携機能で使用する、サーバ証明書、中間証明書の署名を検証するための API です。

R_TSIP_TlsXXCertificateVerification()関数と同じ用途で使用しますが、署名検証をする鍵のアルゴリズムと certificate から取り出す鍵のアルゴリズムが異なる場合には、こちらの関数を使用してください。

Reentrant

非対応

4.2.14 TLS（TLS1.2）

Format 等で(1)(2)があるものについては、(1)はクライアント用、(2)はサーバ用です。

4.2.14.1 R_TSIP_TlsGeneratePreMasterSecret

Format

```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_TlsGeneratePreMasterSecret(
        uint32_t *tsip_pre_master_secret
)
```

Parameters

tsip_pre_master_secret	出力	暗号化した ephemeral Pre Master Secret データ R_TSIP_TlsGenerateMasterSecret、R_TSIP_TlsEncryptPreMasterSecretWithRsa2048PublicKey または R_TSIP_TlsGenerateExtendedMasterSecret の入力 tsip_pre_master_secret に使用してください。20 ワード(80 バイト)出力されます。

Return Values

TSIP_SUCCESS :	正常終了
TSIP_ERR_RESOURCE_CONFLICT:	本処理に必要なハードウェアリソースが他の処理で使用されていることによるリソース衝突が発生

Description

TLS 連携機能で使用する、暗号化された Pre Master Secret を生成するための API です。

Reentrant

非対応

4.2.14.2 R_TSIP_TlsEncryptPreMasterSecretWithRsa2048PublicKey

Format

```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_TlsEncryptPreMasterSecretWithRsa2048PublicKey(
        uint32_t *encryted_public_key,
        uint32_t *tsip_pre_master_secret,
        uint8_t *encrypted_pre_master_secret
)
```

Parameters

encrypted_public_key	入力	暗号化された公開鍵データ R_TSIP_TlsCertificateVerification または R_TSIP_TlsCertificateVerificationExtension の出力 encrypted_output_public_key を使用してください。 140 ワード(560 バイト)サイズ
tsip_pre_master_secret	入力	R_TSIP_TlsGeneratePreMasterSecret が出力する、暗号化した ephemeral Pre Master Secret データ
encrypted_pre_master_secret	出力	public_key を用いて RSA2048 で暗号化した Pre Master Secret データ

Return Values

TSIP_SUCCESS :	正常終了
TSIP_ERR_FAIL:	内部エラーが発生
TSIP_ERR_RESOURCE_CONFLICT:	本処理に必要なハードウェアリソースが他の処理で使用されていることによるリソース衝突が発生

Description

TLS 連携機能で使用する、入力データの公開鍵を用いて、Pre Master Secret を RSA2048 で暗号化するための API です。

Reentrant

非対応

4.2.14.3 R_TSIP_TlsSVGenerateServerRandom

Format

```
#include “r_tsip_rx_if.h”
 e_tsip_err_t R_TSIP_TlsSVGenerateServerRandom(
        e_tsip_tls_version_t client_tls_version,
        e_tsip_tls_version_t server_tls_version,
        uint32_t gmt_unix_time
        tsip_tls_sv_random_t *server_random
)
```

Parameters

client_tls_version	入力	TLS クライアントのプロトコルバージョン TSIP_TLS_VERSION_12 : TLS1.2 TSIP_TLS_VERSION_13 : TLS1.3
server_tls_version	入力	TLS サーバーのプロトコルバージョン TSIP_TLS_VERSION_12 : TLS1.2 TSIP_TLS_VERSION_13 : TLS1.3
gmt_unix_time	入力	標準の UNIX 32 ビット形式の現在の日時 client_tls_version=TSIP_TLS_VERSION_12 の時に使用されます。
server_random	出力	ServerHello で通知する乱数値と、その MAC 値
random		ServerHello で通知する乱数値 8 ワード(32 バイト)サイズ
mac		ServerHello で通知する乱数値の MAC 値 4 ワード(12 バイト)サイズ TLS1.3 通信を使用時、本データは使用されません。

Return Values

TSIP_SUCCESS :	正常終了
TSIP_ERR_RESOURCE_CONFLICT:	本処理に必要なハードウェアリソースが他の処理で使用されていることによるリソース衝突が発生
TSIP_ERR_PARAMETER	入力データが不正

Description

TLS 連携機能のサーバ機能で使用する、ServerHello で通知する乱数値と、その MAC 値を出力するための API です。
ServerHello で使用する乱数値を固定して使用した場合、同一の Master Secret を使用した TLS 通信を行い続けることが可能となります。このような攻撃を防止しサーバシステムの安全性を確保するため、安全に使用可能な乱数値を自動生成する機能を提供します。また、自動生成された乱数値を意図的に変更してその後の処理で使用するといった攻撃を防ぐために MAC 値も付加しています。ドライバの呼び出し側では、ServerHello 生成のたびに毎回必ず本 API を呼び出し、最新の出力を使用してください。

Reentrant

非対応

4.2.14.4 R_TSIP_TlsSVDecryptPreMasterSecretWithRsa2048PrivateKey

Format

```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_TlsSVDecryptPreMasterSecretWithRsa2048PrivateKey(
        tsip_rsa2048_private_key_index_t *private_key_index,
        uint8_t *encrypted_pre_master_secret,
        uint32_t *tsip_pre_master_secret
)
```

Parameters

private_key_index	入力	RSA 秘密鍵の Wrapped Key R_TSIP_GenerateRsa2048PrivateKeyIndex の出力 key_index を使用してください。
encrypted_pre_master_secret	入力	RSA2048 で暗号化された Pre Master Secret データ
tsip_pre_master_secret	出力	暗号化した ephemeral Pre Master Secret データ 16 ワード(64 バイト)で出力されます。

Return Values

TSIP_SUCCESS :	正常終了
TSIP_ERR_KEY_SET:	異常な Wrapped Key が入力された
TSIP_ERR_RESOURCE_CONFLICT:	本処理に必要なハードウェアリソースが他の処理で使用されていることによるリソース衝突が発生

Description

TLS 連携機能のサーバ機能で使用する、RSA2048 秘密鍵を用いて、クライアントから提供される暗号化された Pre Master Secret を RSA2048 で復号するための API です。

Reentrant

非対応

4.2.14.5 R_TSIP_TlsXXGenerateMasterSecret

Format

(1)
```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_TlsGenerateMasterSecret(
        uint32_t select_cipher_suite,
        uint32_t *tsip_pre_master_secret,
        uint8_t *client_random,
        uint8_t *server_random,
        uint32_t *tsip_master_secret
)
```

(2)
```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_TlsSVGenerateMasterSecret(
        uint32_t select_cipher_suite,
        uint32_t *tsip_pre_master_secret,
        uint8_t *client_random,
        tsip_tls_sv_random_t *server_random,
        uint32_t *tsip_master_secret
)
```

Parameters

select_cipher_suite	入力	選択する cipher suite 0:R_TSIP_TLS_RSA_WITH_AES_128_CBC_SHA 1:R_TSIP_TLS_RSA_WITH_AES_256_CBC_SHA 2:R_TSIP_TLS_RSA_WITH_AES_128_CBC_SHA256 3:R_TSIP_TLS_RSA_WITH_AES_256_CBC_SHA256 4:R_TSIP_TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256 5:R_TSIP_TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256 6:R_TSIP_TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256 7:R_TSIP_TLS_ECDHE_RSSA_WITH_AES_128_GCM_SHA256
tsip_pre_master_secret	入力	暗号化した ephemeral Pre Master Secret データ (1) R_TSIP_TlsGeneratePreMasterSecret の出力 tsip_pre_master_secret または R_TSIP_TlsGeneratePreMasterSecretWithEccP256Key の出力 encrypted_pre_master_secret を使用してください。 (2) R_TSIP_TlsSVDecryptPreMasterSecretWithRsa2048PrivateKey または R_TSIP_TlsSVGeneratePreMasterSecretWithEccP256Key の出力 tsip_pre_master_secret を使用してください。
client_random	入力	(1)ClientHello で通知した乱数値 32 バイト (2)ClientHello で通知された乱数値 32 バイト
server_random	入力	(1)ServerHello で通知された乱数値 32 バイト (2)ServerHello で通知した乱数値と、その MAC 値
tsip_master_secret	出力	暗号化した ephemeral Master Secret データ R_TSIP_TlsXXGenerateSessionKey または R_TSIP_TlsXXGenerateVerifyData の入力 tsip_master_secret に使用してください。 20 ワード(80 バイト)で出力されます。

Return Values

TSIP_SUCCESS :	正常終了
TSIP_ERR_FAIL:	内部エラーが発生

TSIP_ERR_RESOURCE_CONFLICT:	本処理に必要なハードウェアリソースが他の処理で使用されていることによるリソース衝突が発生

Description

TLS 連携機能で使用する、暗号化された Master Secret を生成するための API です。

Reentrant

非対応

4.2.14.6 R_TSIP_TlsXXGenerateSessionKey

Format

(1)
```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_TlsGenerateSessionKey(
        uint32_t select_cipher_suite,
        uint32_t *tsip_master_secret,
        uint8_t *client_random,
        uint8_t *server_random,
        uint8_t *nonce_explicit,
        tsip_hmac_sha_key_index_t *client_mac_key_index,
        tsip_hmac_sha_key_index_t *server_mac_key_index,
        tsip_aes_key_index_t *client_crypt_key_index,
        tsip_aes_key_index_t *server_crypt_key_index,
        uint8_t *client_iv,
        uint8_t *server_iv
)
```

(2)
```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_TlsSVGenerateSessionKey(
        uint32_t select_cipher_suite,
        uint32_t *tsip_master_secret,
        uint8_t *client_random,
        tsip_tls_sv_random_t *server_random,
        uint8_t *nonce_explicit,
        tsip_hmac_sha_key_index_t *client_mac_key_index,
        tsip_hmac_sha_key_index_t *server_mac_key_index,
        tsip_aes_key_index_t *client_crypt_key_index,
        tsip_aes_key_index_t *server_crypt_key_index,
        uint8_t *client_iv,
        uint8_t *server_iv
)
```

Parameters

select_cipher_suite	入力	選択する cipher suite 0:R_TSIP_TLS_RSA_WITH_AES_128_CBC_SHA 1:R_TSIP_TLS_RSA_WITH_AES_256_CBC_SHA 2:R_TSIP_TLS_RSA_WITH_AES_128_CBC_SHA256 3:R_TSIP_TLS_RSA_WITH_AES_256_CBC_SHA256 4:R_TSIP_TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256 5:R_TSIP_TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256 6:R_TSIP_TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256 7:R_TSIP_TLS_ECDHE_RSSA_WITH_AES_128_GCM_SHA256
tsip_master_secret	入力	暗号化した ephemeral Master Secret データ R_TSIP_TlsXXGenerateMasterSecret の出力 tsip_master_secret を使用してください。
client_random	入力	(1)ClientHello で通知した乱数値 32 バイト (2)ClientHello で通知された乱数値 32 バイト
server_random	入力	(1)ServerHello で通知された乱数値 32 バイト (2)ServerHello で通知した乱数値と、その MAC 値
nonce_explicit	入力	cipher suite AES128GCM で使用するノンス select_cipher_suite=6-7: 8 バイト
client_mac_key_index	出力	クライアント→サーバ通信時の MAC の Wrapped Key 形式データ
server_mac_key_index	出力	サーバ→クライアント通信時の MAC の Wrapped Key 形式データ
client_crypt_key_index	出力	クライアント→サーバ通信時の AES 共通鍵の Wrapped Key

server_crypt_key_index	出力	サーバ→クライアント通信時の AES 共通鍵の Wrapped Key
client_iv	出力	Client から Sever へ送信時に使用する IV select_cipher_suite が 0~5 の時に出力されます。 ((1)の場合、RX651,RX65N で NetX Duo を使用する場合に使用します) それ以外の場合には、何も出力されません。
server_iv	出力	Server から受信時に使用する IV select_cipher_suite が 0~5 の時に出力されます。 ((1)の場合、RX651,RX65N で NetX Duo を使用する場合に使用します) それ以外の場合には、何も出力されません。

Return Values

TSIP_SUCCESS :	正常終了
TSIP_ERR_FAIL:	内部エラーが発生
TSIP_ERR_RESOURCE_CONFLICT:	本処理に必要なハードウェアリソースが他の処理で使用されていることによるリソース衝突が発生

Description

TLS 連携機能で使用する、TLS 通信の各種鍵を出力するための API です。

client_iv、server_iv 引数には、引数の説明にある場合以外には何も出力されません。

Reentrant

非対応

4.2.14.7 R_TSIP_TlsXXGenerateVerifyData

Format

(1)
```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_TlsGenerateVerifyData(
        uint32_t select_verify_data,
        uint32_t *tsip_master_secret,
        uint8_t *hand_shake_hash,
        uint8_t *verify_data
)
```

(2)
```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_TlsSVGenerateVerifyData(
        uint32_t select_verify_data,
        uint32_t *tsip_master_secret,
        uint8_t *hand_shake_hash,
        uint8_t *verify_data
)
```

Parameters

select_verify_data	入力	選択する Client/Server の種別 R_TSIP_TLS_GENERATE_CLIENT_VERIFY : ClientVerifyData の生成 R_TSIP_TLS_GENERATE_SERVER_VERIFY : ServerVerifyData の生成
tsip_master_secret	入力	暗号化した ephemeral Master Secret データ R_TSIP_TlsXXGenerateMasterSecret の出力 tsip_master_secret を使用してください。
hand_shake_hash	入力	TLS ハンドシェイクメッセージ全体の SHA256 ハッシュ値
verify_data	出力	Finished メッセージ用の VerifyData

Return Values

TSIP_SUCCESS :	正常終了
TSIP_ERR_FAIL:	内部エラーが発生
TSIP_ERR_RESOURCE_CONFLICT:	本処理に必要なハードウェアリソースが他の処理で使用されていることによるリソース衝突が発生

Description

TLS 連携機能で使用する、VerifyData を生成するための API です。

Reentrant

非対応

4.2.14.8 R_TSIP_TlsServersEphemeralEcdhPublicKeyRetrieves

Format

```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_TlsServersEphemeralEcdhPublicKeyRetrieves(
        uint32_t public_key_type,
        uint8_t *client_random,
        uint8_t *server_random,
        uint8_t *server_ephemeral_ecdh_public_key,
        uint8_t *server_key_exchange_signature,
        uint32_t *encrypted_public_key,
        uint32_t *encrypted_ephemeral_ecdh_public_key
)
```

Parameters

public_key_type	入力	公開鍵の種類 0 : RSA 2048bit(rsa_pkcs1_sha256), 1 : RSA 4096bit(rsa_pkcs1_sha256), 2 : ECDSA P-256、 3 : RSA 2048bit(rsa_pss_rsae_sha256)
client_random	入力	ClientHello で通知した乱数値 32 バイト
server_random	入力	ServerHello で通知された乱数値 32 バイト
server_ephemeral_ecdh_public_key	入力	サーバから受け取った ephemeral ECDH 公開鍵 (非圧縮形式) 0padding(24bit) || 04(8bit) || Qx(256bit) || Qy(256bit)
server_key_exchange_signature	入力	ServerKeyExchange の署名データ public_key_type 0:256 バイト,2:64 バイト
encrypted_public_key	入力	署名検証のための暗号化された公開鍵 R_TSIP_TlsCertificateVerification または R_TSIP_TlsCertificateVerificationExtension の出力 encrypted_output_public_key を使用してください。 public_key_type 0:140 ワード(560 バイト), 2:24 ワード (96 バイト)
encrypted_ephemeral_ecdh_public_key	出力	R_TSIP_TlsGeneratePreMasterSecretWithEccP256Key で使用する、暗号化された ephemeral ECDH 公開鍵 24 ワード(96 バイト)サイズ

Return Values

TSIP_SUCCESS :	正常終了
TSIP_ERR_FAIL:	内部エラーが発生
TSIP_ERR_RESOURCE_CONFLICT:	本処理に必要なハードウェアリソースが他の処理で使用されていることによるリソース衝突が発生

Description

TLS 連携機能で使用する、入力された公開鍵データを用いて、ServerKeyExchange の署名を検証する API です。署名に成功した場合、R_TSIP_TlsGeneratePreMasterSecretWithEccP256Key で使用する ephemeral ECDH 公開鍵を暗号化して出力します。

該当暗号スイート: 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

Reentrant

非対応

4.2.14.9 R_TSIP_GenerateTlsXXP256EccKeyIndex

Format

(1)
```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_GenerateTlsP256EccKeyIndex(
        tsip_tls_p256_ecc_key_index_t *tls_p256_ecc_key_index,
        uint8_t *ephemeral_ecdh_public_key
)
```

(2)
```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_GenerateTlsSVP256EccKeyIndex(
        tsip_tls_p256_ecc_key_index_t *tls_p256_ecc_key_index,
        uint8_t *ephemeral_ecdh_public_key
)
```

Parameters

tls_p256_ecc_key_index	出力	Ephemeral ECC 秘密鍵の Wrapped Key R_TSIP_TlsXXGeneratePreMasterSecretWithEccP256Key の入力 tls_p256_ecc_key_index に使用してください。
ephemeral_ecdh_public_key	出力	(1)サーバへ送信する ephemeral ECDH 公開鍵 (2)クライアントへ送信する ephemeral ECDH 公開鍵 いずれも Qx(256bit) || Qy(256bit)

Return Values

TSIP_SUCCESS :	正常終了
TSIP_ERR_FAIL:	内部エラーが発生
TSIP_ERR_RESOURCE_CONFLICT:	本処理に必要なハードウェアリソースが他の処理で使用されていることによるリソース衝突が発生

Description

TLS 連携機能で使用する、乱数から 256bit 素体上の楕円曲線暗号のための鍵ペアを生成する API です。

該当暗号スイート: 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

Reentrant

非対応

4.2.14.10　R_TSIP_TlsXXGeneratePreMasterSecretWithEccP256Key

Format

(1)
```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_TlsGeneratePreMasterSecretWithEccP256Key(
        uint32_t *encrypted_public_key,
        tsip_tls_p256_ecc_key_index_t *tls_p256_ecc_key_index,
        uint32_t *tsip_pre_master_secret
)
```
(2)
```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_TlsSVGeneratePreMasterSecretWithEccP256Key(
        uint32_t *ecdh_public_key,
        tsip_tls_p256_ecc_key_index_t *tls_p256_ecc_key_index,
        uint32_t *tsip_pre_master_secret
)
```

Parameters

(1) encrypted_public_key (2) ecdh_public_key	入力	ephemeral ECDH 公開鍵 (1) R_TSIP_TlsServersEphemeralEcdhPublicKeyRetrieves の出力 encrypted_ephemeral_ecdh_public_key を使用してください。 (2) クライアントから受信した ephemeral ECDH 公開鍵を Qx(256bit) || Qy(256bit)の形式で使用してください。
tls_p256_ecc_key_index	入力	Ephemeral ECC 秘密鍵の Wrapped Key R_TSIP_GenerateTlsXXP256EccKeyIndex の出力 tls_p256_ecc_key_index を使用してください。
tsip_pre_master_secret	出力	暗号化した ephemeral Pre Master Secret データ 16 ワード(64 バイト)で出力されます。

Return Values

TSIP_SUCCESS :	正常終了
TSIP_ERR_FAIL:	内部エラーが発生
TSIP_ERR_RESOURCE_CONFLICT:	本処理に必要なハードウェアリソースが他の処理で使用されていることによるリソース衝突が発生
TSIP_ERR_KEY_SET:	異常な Wrapped Key が入力された

Description

TLS 連携機能で使用する、入力された鍵データを用いて、暗号化された Pre Master Secret を生成するための API です。

該当暗号スイート: 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

Reentrant

非対応

4.2.14.11　R_TSIP_TlsXXGenerateExtendedMasterSecret

Format

```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_TlsGenerateExtendedMasterSecret(
        uint32_t select_cipher_suite,
        uint32_t *tsip_pre_master_secret,
        uint8_t *digest,
        uint32_t *extended_master_secret
)
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_TlsSVGenerateExtendedMasterSecret(
        uint32_t select_cipher_suite,
        uint32_t *tsip_pre_master_secret,
        uint8_t *digest,
        uint32_t *extended_master_secret
)
```

Parameters

select_cipher_suite	入力	選択する cipher suite 2:R_TSIP_TLS_RSA_WITH_AES_128_CBC_SHA256 3:R_TSIP_TLS_RSA_WITH_AES_256_CBC_SHA256 4:R_TSIP_TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256 5:R_TSIP_TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256 6:R_TSIP_TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256 7:R_TSIP_TLS_ECDHE_RSSA_WITH_AES_128_GCM_SHA256
tsip_pre_master_secret	入力	暗号化した ephemeral Pre Master Secret データ (1) R_TSIP_TlsGeneratePreMasterSecret または R_TSIP_TlsGeneratePreMasterSecretWithEccP256Key の出力 tsip_pre_master_secret を使用してください。 (2) R_TSIP_TlsSVDecryptPreMasterSecretWithRsa2048PrivateKey または R_TSIP_TlsSVGeneratePreMasterSecretWithEccP256Key の出力 tsip_pre_master_secret を使用してください。
digest	入力	SHA256 で演算したメッセージハッシュ (ClientHello||ServerHello||Certificate||ServerKeyExchange ||CertificateRequest||ServerHelloDone||Certificate ||ClientKeyExchange)のように、ハンドシェイクメッセージを 連結した値のハッシュ値を演算して入力してください。 ハッシュ値の演算には R_TSIP_Sha256Init/Update/Final を使用し、 R_TSIP_Sha256Final の出力 digest を入力に使用してくださ い。
extended_master_secret	出力	暗号化した ephemeral Extended Master Secret データ 20 ワード(80 バイト)で出力されます。 R_TSIP_TlsXXGenerateSessionKey または R_TSIP_TlsXXGenerateVerifyData の入力 tsip_master_secret に使用してください。

Return Values

TSIP_SUCCESS :	正常終了
TSIP_ERR_FAIL:	内部エラーが発生

TSIP_ERR_RESOURCE_CONFLICT:	本処理に必要なハードウェアリソースが他の処理で使用されていることによるリソース衝突が発生

Description

TLS 連携機能で使用する、暗号化された Pre Master Secret データを用いて、暗号化された Extended Master Secret データを生成するための API です。

Reentrant

非対応

4.2.14.12　R_TSIP_TlsSVCertificateVerifyVerification

Format

```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_TlsSVCertificateVerifyVerification (
        uint32_t *key_index,
        e_tsip_tls_signature_scheme_type_t signature_scheme,
        uint8_t *handshake_message,
        uint32_t handshake_message_len,
        uint8_t *certificate_verify,
        uint32_t certificate_verify_len
)
```

Parameters

key_index	入力	暗号化された公開鍵 R_TSIP_TlsSVCertificateVerification または R_TSIP_TlsSVCertificateVerificationExtension の出力 encrypted_output_public_key を使用してください。
signature_scheme	入力	使用する署名アルゴリズム
handshake_message	入力	ハンドシェイクメッセージ (ClientHello||ServerHello||Certificate||ServerKeyExchange ||CertificateRequest||ServerHelloDone||Certificate ||ClientKeyExchange) のように、ハンドシェイクメッセージを連結した値を入力してください。
handshake_message_len	入力	handshake_message のバイト長
certificate_verify	入力	CertificateVerify CertificateVerify のデータを格納しているバッファの先頭アドレスを入力してください。
certificate_verify_len	入力	certificate_verify のバイト長

Return Values

TSIP_SUCCESS :	正常終了
TSIP_ERR_FAIL	内部エラーが発生、もしくは署名検証失敗
TSIP_ERR_RESOURCE_CONFLICT:	本処理に必要なハードウェアリソースが他の処理で使用されていることによるリソース衝突が発生
TSIP_ERR_PARAMETER	入力データが不正

Description

TLS 連携機能のサーバ機能で使用する、クライアントから受信した CertificateVerify を検証するための API です。アルゴリズムは rsa_pkcs1_sha256、ecdsa_secp256r1_sha256 および rsa_pss_rsae_sha256 を使用します。

Reentrant

非対応

4.2.15 TLS（TLS1.3）

4.2.15.1 R_TSIP_GenerateTls13P256EccKeyIndex

Format

```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_ TlsGenerateTls13P256EccKeyIndex(
        tsip_tls13_handle_t *handle,
        e_tsip_tls13_mode_t mode,
        tsip_tls_p256_ecc_key_index_t *key_index,
        uint8_t *ephemeral_ecdh_public_key
)
```

Parameters

handle	入力	同一セッションを示すハンドル番号(ワーク領域)
mode	入力	実施するハンドシェイクプロトコル TSIP_TLS13_MODE_FULL_HANDSHAKE : Full Handshake TSIP_TLS13_MODE_RESUMPTION : Resumption TSIP_TLS13_MODE_0_RTT : 0-RTT
key_index	出力	Ephemeral ECC 秘密鍵の Wrapped Key R_TSIP_Tls13GenerateEcdheSharedSecret の入力 key_index に使用してください。
ephemeral_ecdh_public_key	出力	サーバへ送信する Ephemeral ECDH 公開鍵 Qx(256bit) || Qy(256bit)

Return Values

TSIP_SUCCESS :	正常終了
TSIP_ERR_FAIL:	内部エラーが発生
TSIP_ERR_RESOURCE_CONFLICT:	本処理に必要なハードウェアリソースが他の処理で使用されていることによるリソース衝突が発生

Description

TLS1.3 連携機能で使用する、乱数から 256bit 素体上の楕円曲線暗号のための鍵ペアを生成するための API です。

Reentrant

非対応

4.2.15.2 R_TSIP_Tls13GenerateEcdheSharedSecret

Format

```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_ Tls13GenerateEcdheSharedSecret(
        e_tsip_tls13_mode_t mode,
        uint8_t *server_public_key,
        tsip_tls_p256_ecc_key_index_t *key_index,
        tsip_tls13_ephemeral_shared_secret_key_index_t *shared_secret_key_index
)
```

Parameters

mode	入力	実施するハンドシェイクプロトコル TSIP_TLS13_MODE_FULL_HANDSHAKE : Full Handshake TSIP_TLS13_MODE_RESUMPTION : Resumption TSIP_TLS13_MODE_0_RTT : 0-RTT
server_public_key	入力	サーバから提供される公開鍵 Qx(256bit) || Qy(256bit)
key_index	入力	Ephemeral ECC 秘密鍵の Wrapped Key R_TSIP_GenerateTls13P256EccKeyIndex の出力 key_index を使用してください。
shared_secret_key_index	出力	Shared Secret の Ephemeral 鍵の Wrapped Key R_TSIP_Tls13GenerateHandshakeSecret および R_TSIP_Tls13GenerateResumptionHandshakeSecret の入力 shared_secret_key_index に使用してください。

Return Values

TSIP_SUCCESS :	正常終了
TSIP_ERR_FAIL:	内部エラーが発生
TSIP_ERR_RESOURCE_CONFLICT:	本処理に必要なハードウェアリソースが他の処理で使用されていることによるリソース衝突が発生
TSIP_ERR_KEY_SET	異常な Wrapped Key が入力された

Description

TLS1.3 連携機能で使用する、サーバから提供される公開鍵とあらかじめ演算した秘密鍵を用いて、256bit 素体上の共有鍵である Shared Secret を計算し、Wrapped Key を生成するための API です。

該当暗号スイート: TLS_AES_128_GCM_SHA256, TLS_AES_128_CCM_SHA256

鍵交換の方式: ECDHE NIST P-256

Reentrant

非対応

4.2.15.3 R_TSIP_Tls13GenerateHandshakeSecret

Format

```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_ Tls13GenerateHandshakeSecret(
        tsip_tls13_ephemeral_shared_secret_key_index_t *shared_secret_key_index,
        tsip_tls13_ephemeral_handshake_secret_key_index_t *handshake_secret_key_index
)
```

Parameters

shared_secret_key_index	入力	Shared Secret の Ephemeral の Wrapped Key R_TSIP_Tls13GenerateEcdheSharedSecret の出力 shared_secret_key_index を使用してください。
handshake_secret_key_index	出力	Handshake Secret の Ephemeral の Wrapped Key R_TSIP_Tls13GenerateServerHandshakeTrafficKey、R_TSIP_Tls13GenerateClientHandshakeTrafficKey および R_TSIP_Tls13GenerateMasterSecret の入力 handshake_secret_key_index に使用してください。

Return Values

TSIP_SUCCESS :	正常終了
TSIP_ERR_FAIL:	内部エラーが発生
TSIP_ERR_RESOURCE_CONFLICT:	本処理に必要なハードウェアリソースが他の処理で使用されていることによるリソース衝突が発生
TSIP_ERR_KEY_SET	異常な Wrapped Key が入力された

Description

TLS1.3 連携機能で使用する、Shared Secret の Ephemeral 鍵を用いて、Handshake Secret の Wrapped Key を生成するための API です。

Reentrant

非対応

4.2.15.4 R_TSIP_Tls13GenerateServerHandshakeTrafficKey

Format

```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_Tls13GenerateServerHandshakeTrafficKey(
        tsip_tls13_handle_t *handle,
        e_tsip_tls13_mode_t mode,
        tsip_tls13_ephemeral_handshake_secret_key_index_t *handshake_secret_key_index,
        uint8_t *digest,
        tsip_aes_key_index_t *server_write_key_index,
        tsip_tls13_ephemeral_server_finished_key_index_t *server_finished_key_index
)
```

Parameters

handle	出力	同一セッションを示すハンドル番号(ワーク領域)
mode	入力	実施するハンドシェイクプロトコル TSIP_TLS13_MODE_FULL_HANDSHAKE : Full Handshake TSIP_TLS13_MODE_RESUMPTION : Resumption TSIP_TLS13_MODE_0_RTT : 0-RTT
handshake_secret_key_index	入力	Handshake Secret の Ephemeral の Wrapped Key R_TSIP_Tls13GenerateHandshakeSecret または R_TSIP_Tls13GenerateResumptionHandshakeSecret の出力 handshake_secret_key_index を使用してください。
digest	入力	SHA256 で演算したメッセージハッシュ(ClientHello||ServerHello)のハッシュ値を演算して入力してください。 ハッシュ値の演算には R_TSIP_Sha256Init/Update/Final を使用し、 R_TSIP_Sha256Final の出力 digest を入力に使用してください。
server_write_key_index	出力	Server Write Key の Ephemeral の Wrapped Key R_TSIP_Tls13DecryptInit の入力 key_index に使用してください。
server_finished_key_index	出力	Server Finished Key の Ephemeral の Wrapped Key R_TSIP_Tls13ServerHandshakeVerification の入力 server_finished_key_index に使用してください。

Return Values

TSIP_SUCCESS :	正常終了
TSIP_ERR_RESOURCE_CONFLICT:	本処理に必要なハードウェアリソースが他の処理で使用されていることによるリソース衝突が発生
TSIP_ERR_KEY_SET	異常な Wrapped Key が入力された

Description

TLS1.3 連携機能で使用する、R_TSIP_Tls13GenerateHandshakeSecret で出力された Handshake Secret を用いて Server Write Key および Server Finished Key の Wrapped Key を生成するための API です。

Reentrant

非対応

4.2.15.5 R_TSIP_Tls13GenerateClientHandshakeTrafficKey

Format

```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_Tls13GenerateClientHandshakeTrafficKey(
        tsip_tls13_handle_t *handle,
        e_tsip_tls13_mode_t mode,
        tsip_tls13_ephemeral_handshake_secret_key_index_t *handshake_secret_key_index,
        uint8_t *digest,
        tsip_aes_key_index_t *client_write_key_index,
        tsip_hmac_sha_key_index_t *client_finished_key_index
)
```

Parameters

handle	出力	同一セッションを示すハンドル番号(ワーク領域)
mode	入力	実施するハンドシェイクプロトコル TSIP_TLS13_MODE_FULL_HANDSHAKE : Full Handshake TSIP_TLS13_MODE_RESUMPTION : Resumption TSIP_TLS13_MODE_0_RTT : 0-RTT
handshake_secret_key_index	入力	Handshake Secret の Ephemeral の Wrapped Key R_TSIP_Tls13GenerateHandshakeSecret ま た は R_TSIP_Tls13GenerateResumptionHandshakeSecret の出力 handshake_secret_key_index を使用してください。
digest	入力	SHA256 で演算したメッセージハッシュ(ClientHello||ServerHello)のハッシュ値を演算して入力してください。 ハッシュ値の演算には R_TSIP_Sha256Init/Update/Final を使用し、 R_TSIP_Sha256Final の出力 digest を入力に使用してください。
client_write_key_index	出力	Client Write Key の Ephemeral の Wrapped Key R_TSIP_Tls13EncryptInit の入力 key_index に使用してください。
client_finished_key_index	出力	Client Finished Key の Ephemeral の Wrapped Key R_TSIP_Sha256HmacGenerateInit の入力 key_index に使用してください。

Return Values

TSIP_SUCCESS :	正常終了
TSIP_ERR_RESOURCE_CONFLICT:	本処理に必要なハードウェアリソースが他の処理で使用されていることによるリソース衝突が発生
TSIP_ERR_KEY_SET	異常な Wrapped Key が入力された

Description

TLS1.3 連携機能で使用する、R_TSIP_Tls13GenerateHandshakeSecret で出力された Handshake Secret を用いて Client Write Key および Client Finished Key の Wrapped Key を生成するための API です。

Reentrant

非対応

4.2.15.6 R_TSIP_Tls13ServerHandshakeVerification

Format

```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_ Tls13ServerHandshakeVerification(
        e_tsip_tls13_mode_t mode,
        tsip_tls13_ephemeral_server_finished_key_index_t *server_finished_key_index,
        uint8_t *digest,
        uint8_t *server_finished,
        uint32_t *verify_data_index
)
```

Parameters

mode	入力	実施するハンドシェイクプロトコル TSIP_TLS13_MODE_FULL_HANDSHAKE : Full Handshake TSIP_TLS13_MODE_RESUMPTION : Resumption TSIP_TLS13_MODE_0_RTT : 0-RTT
server_finished_key_index	入力	Server Finished Key の Ephemeral の Wrapped Key R_TSIP_Tls13GenerateServerHandshakeTrafficKey の出力 server_finished_key_index を使用してください。
digest	入力	SHA256 で演算したメッセージハッシュ (ClientHello||ServerHello||EncryptedExtensions ||CertificateRequest||Certificate||CertificateVerify) のように、ハンドシェイクメッセージを連結した値のハッシュ値を演算して入力してください。 ハッシュ値の演算には R_TSIP_Sha256Init/Update/Final を使用し、 R_TSIP_Sha256Final の出力 digest を入力に使用してください。
server_finished	入力	サーバから提供される Finished 情報 R_TSIP_Tls13DecryptUpdate/Final により取得した Server Finished のデータを格納しているバッファの先頭アドレスを入力してください。
verify_data_index	出力	TSIP 固有の仕様の Server Handshake 検証結果 R_TSIP_Tls13GenerateMasterSecret の入力 verify_data_index に使用してください。 データを出力するバッファの先頭アドレスを入力してください。必要サイズは 8 ワード(32 バイト)です。

Return Values

TSIP_SUCCESS :	正常終了
TSIP_ERR_FAIL	内部エラーが発生
TSIP_ERR_RESOURCE_CONFLICT:	本処理に必要なハードウェアリソースが他の処理で使用されていることによるリソース衝突が発生
TSIP_ERR_KEY_SET	異常な Wrapped Key が入力された
TSIP_ERR_VERIFICATION_FAIL	検証で合格しなかった

Description

TLS1.3 連携機能で使用する、サーバから提供された Finished の情報を用いて Handshake を検証するための API です。

Reentrant

非対応

4.2.15.7 R_TSIP_Tls13GenerateMasterSecret

Format

```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_ Tls13GenerateMasterSecret(
        tsip_tls13_handle_t *handle,
        e_tsip_tls13_mode_t mode,
        tsip_tls13_ephemeral_handshake_secret_key_index_t *handshake_secret_key_index,
        uint32_t *verify_data_index,
        tsip_tls13_ephemeral_master_secret_key_index_t *master_secret_key_index
)
```

Parameters

handle	出力	同一セッションを示すハンドル番号(ワーク領域)
mode	入力	実施するハンドシェイクプロトコル TSIP_TLS13_MODE_FULL_HANDSHAKE : Full Handshake TSIP_TLS13_MODE_RESUMPTION : Resumption TSIP_TLS13_MODE_0_RTT : 0-RTT
handshake_secret_key_index	入力	Handshake Secret の Ephemeral の Wrapped Key R_TSIP_Tls13GenerateHandshakeSecret の出力 handshake_secret_key_index を使用してください。
verify_data_index	入力	TSIP 固有の仕様の Server Handshake 検証結果 R_TSIP_Tls13ServerHandshakeVerification の出力 verify_data_index を使用してください。
master_secret_key_index	出力	Master Secret の Ephemeral の Wrapped Key R_TSIP_Tls13GenerateApplicationTrafficKey および R_TSIP_Tls13GenerateResumptionMasterSecret の入力 master_secret_key_index に使用してください。

Return Values

TSIP_SUCCESS :	正常終了
TSIP_ERR_FAIL	内部エラーが発生
TSIP_ERR_RESOURCE_CONFLICT:	本処理に必要なハードウェアリソースが他の処理で使用されていることによるリソース衝突が発生
TSIP_ERR_KEY_SET	異常な Wrapped Key が入力された

Description

TLS1.3 連携機能で使用する、Handshake Secret の Ephemeral 鍵を用いて、Master Secret の Ephemeral の Wrapped Key を生成するための API です。

Reentrant

非対応

4.2.15.8 R_TSIP_Tls13GenerateApplicationTrafficKey

Format

```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_Tls13GenerateApplicationTrafficKey(
        tsip_tls13_handle_t *handle,
        e_tsip_tls13_mode_t mode,
        tsip_tls13_ephemeral_master_secret_key_index_t *master_secret_key_index,
        uint8_t *digest,
        tsip_tls13_ephemeral_app_secret_key_index_t *server_app_secret_key_index,
        tsip_tls13_ephemeral_app_secret_key_index_t *client_app_secret_key_index,
        tsip_aes_key_index_t *server_write_key_index,
        tsip_aes_key_index_t *client_write_key_index
)
```

Parameters

handle	入力/出力	同一セッションを示すハンドル番号(ワーク領域)
mode	入力	実施するハンドシェイクプロトコル TSIP_TLS13_MODE_FULL_HANDSHAKE : Full Handshake TSIP_TLS13_MODE_RESUMPTION : Resumption TSIP_TLS13_MODE_0_RTT : 0-RTT
master_secret_key_index	入力	Master Secret の Ephemeral の Wrapped Key R_TSIP_Tls13GenerateMasterSecret の出力 master_secret_key_index を使用してください。
digest	入力	SHA256 で演算したメッセージハッシュ (ClientHello||ServerHello||EncryptedExtensions ||CertificateRequest||Certificate||CertificateVerify ||ServerFinished) のように、ハンドシェイクメッセージを連結した値のハッシュ値を演算して入力してください。 ハッシュ値の演算には R_TSIP_Sha256Init/Update/Final を使用し、R_TSIP_Sha256Final の出力 digest を入力に使用してください。
server_app_secret_key_index	出力	Server Application Traffic Secret の Ephemeral の Wrapped Key R_TSIP_Tls13UpdateApplicationTrafficKey の入力 input_app_secret_key_index に使用してください。
client_app_secret_key_index	出力	ClientApplicationTrafficSecret の Ephemeral の Wrapped Key R_TSIP_Tls13UpdateApplicationTrafficKey の入力 input_app_secret_key_index に使用してください。
server_write_key_index	出力	Server Write Key の Ephemeral の Wrapped Key R_TSIP_Tls13DecryptInit の入力 key_index に使用してください。
client_write_key_index	出力	Client Write Key の Ephemeral の Wrapped Key R_TSIP_Tls13EncryptInit の入力 key_index に使用してください。

Return Values

TSIP_SUCCESS :	正常終了
TSIP_ERR_RESOURCE_CONFLICT:	本処理に必要なハードウェアリソースが他の処理で使用されていることによるリソース衝突が発生
TSIP_ERR_KEY_SET	異常な Wrapped Key が入力された

Description

TLS1.3 連携機能で使用する、Master Secret の Ephemeral 鍵を用いて、Application Traffic Secret の Wrapped Key を生成するための API です。併せて、Server Write Key および Client Write Key の Ephemeral の Wrapped Key を生成します。

Reentrant

非対応

4.2.15.9 R_TSIP_Tls13UpdateApplicationTrafficKey

Format

```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_Tls13UpdateApplicationTrafficKey(
        tsip_tls13_handle_t *handle,
        e_tsip_tls13_mode_t mode,
        e_tsip_tls13_update_key_type_t key_type,
        tsip_tls13_ephemeral_app_secret_key_index_t *input_app_secret_key_index,
        tsip_tls13_ephemeral_app_secret_key_index_t *output_app_secret_key_index,
        tsip_aes_key_index_t *app_write_key_index
)
```

Parameters

handle	入力/出力	同一セッションを示すハンドル番号(ワーク領域)
mode	入力	実施するハンドシェイクプロトコル TSIP_TLS13_MODE_FULL_HANDSHAKE : Full Handshake TSIP_TLS13_MODE_RESUMPTION : Resumption TSIP_TLS13_MODE_0_RTT : 0-RTT
key_type	入力	更新する鍵の種類 TSIP_TLS13_UPDATE_SERVER_KEY : Server Application Traffic Secret TSIP_TLS13_UPDATE_CLIENT_KEY : Client Application Traffic Secret
input_app_secret_key_index	入力	入力する Server/Client Application Traffic Secret の Ephemeral の Wrapped Key R_TSIP_Tls13GenerateApplicationTrafficKey の出力 server/client_app_secret_key_index または R_TSIP_Tls13UpdateApplicationTrafficKey の出力 output_app_secret_key_index のうち、key_type に指定した鍵の種類に適合した入力を使用してください。
output_app_secret_key_index	出力	出力する Server/Client Application Traffic Secret の Ephemeral の Wrapped Key key_type に指定した鍵の種類に対応した出力が得られます。 R_TSIP_Tls13UpdateApplicationTrafficKey の入力 input_app_secret_key_index に使用してください。
app_write_key_index	出力	Server/Client Write Key の Ephemeral の Wrapped Key key_type に指定した鍵の種類に対応した出力が得られます。 Server Write Key は R_TSIP_Tls13DecryptInit の入力 key_index に使用してください。 Client Write Key は R_TSIP_Tls13EncryptInit の入力 key_index に使用してください。

Return Values

TSIP_SUCCESS :	正常終了
TSIP_ERR_FAIL	内部エラーが発生
TSIP_ERR_RESOURCE_CONFLICT:	本処理に必要なハードウェアリソースが他の処理で使用されていることによるリソース衝突が発生
TSIP_ERR_KEY_SET	異常な Wrapped Key が入力された
TSIP_ERR_PARAMETER	入力データが不正

Description

TLS1.3 連携機能で使用する、Application Traffic Secret を用いて、Application Traffic Secret の Wrapped Key と対応する暗号鍵の Wrapped Key を更新するための API です。

Reentrant

非対応

4.2.15.10　R_TSIP_Tls13GenerateResumptionMasterSecret

Format

```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_ Tls13GenerateResumptionMasterSecret(
        tsip_tls13_handle_t *handle,
        e_tsip_tls13_mode_t mode,
        tsip_tls13_ephemeral_master_secret_key_index_t *master_secret_key_index,
        uint8_t *digest,
        tsip_tls13_ephemeral_res_master_secret_key_index_t *res_master_secret_key_index
)
```

Parameters

handle	入力	同一セッションを示すハンドル番号(ワーク領域)
mode	入力	実施するハンドシェイクプロトコル TSIP_TLS13_MODE_FULL_HANDSHAKE : Full Handshake TSIP_TLS13_MODE_RESUMPTION : Resumption TSIP_TLS13_MODE_0_RTT : 0-RTT
master_secret_key_index	入力	Handshake Secret の Ephemeral の Wrapped Key R_TSIP_Tls13GenerateHandshakeSecret の出力 handshake_secret_key_index を使用してください。
digest	入力	SHA256 で演算したメッセージハッシュ (ClientHello||ServerHello||EncryptedExtensions ||CertificateRequest||Certificate||CertificateVerify ||ServerFinished||Certificate||CertificateVerify ||ClientFinished) のように、ハンドシェイクメッセージを連結した値のハッシュ値を演算して入力してください。 ハッシュ値の演算には R_TSIP_Sha256Init/Update/Final を使用し、 R_TSIP_Sha256Final の出力 digest を入力に使用してください。
res_master_secret_key_index	出力	Resumption Master Secret の Ephemeral の Wrapped Key R_TSIP_Tls13GeneratePreSharedKey の入力 res_master_secret_key_index に使用してください。

Return Values

TSIP_SUCCESS :	正常終了
TSIP_ERR_RESOURCE_CONFLICT:	本処理に必要なハードウェアリソースが他の処理で使用されていることによるリソース衝突が発生
TSIP_ERR_KEY_SET	異常な Wrapped Key が入力された

Description

TLS1.3 連携機能で使用する、Master Secret の Ephemeral 鍵を用いて、Resumption Master Secret の Wrapped Key を生成するための API です。

RFC8446 より、Master Secret の Ephemeral の Wrapped Key master_secret_key_index は、本 API によって Resumption Master Secret の Wrapped Key を生成した後に削除してください。

Reentrant

非対応

4.2.15.11　R_TSIP_Tls13GeneratePreSharedKey

Format

```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_ Tls13GeneratePreSharedKey(
        tsip_tls13_handle_t *handle,
        e_tsip_tls13_mode_t mode,
        tsip_tls13_ephemeral_res_master_secret_key_index_t *res_master_secret_key_index,
        uint8_t *ticket_nonce,
        uint32_t ticket_nonce_len,
        tsip_tls13_ephemeral_pre_shared_key_index_t *pre_shared_key_index
)
```

Parameters

handle	入力	同一セッションを示すハンドル番号(ワーク領域)
mode	入力	実施するハンドシェイクプロトコル TSIP_TLS13_MODE_FULL_HANDSHAKE : Full Handshake TSIP_TLS13_MODE_RESUMPTION : Resumption TSIP_TLS13_MODE_0_RTT : 0-RTT
res_master_secret_key_index	入力	Resupmtion Master Secret の Ephemeral の Wrapped Key R_TSIP_Tls13GenerateResumptionMasterSecret の出力 res_master_secret_key_index を使用してください。
ticket_nonce	入力	サーバから提供された Ticket Nonce TicketNonce のサイズが 16 バイトの倍数でない場合は、16 バイトの倍数となるように 0 padding して入力してください。
ticket_nonce_len	入力	Ticket Nonce のバイト長
pre_shared_key_index	出力	Pre Shared Key の Ephemeral の Wrapped Key R_TSIP_Tls13GeneratePskBinderKey、R_TSIP_Tls13GenerateResumptionHandshakeSecret および R_TSIP_Tls13Generate0RttApplicationWriteKey の入力 pre_shared_key_index に使用してください。

Return Values

TSIP_SUCCESS :	正常終了
TSIP_FAIL	内部エラーが発生
TSIP_ERR_RESOURCE_CONFLICT:	本処理に必要なハードウェアリソースが他の処理で使用されていることによるリソース衝突が発生
TSIP_ERR_KEY_SET	異常な Wrapped Key が入力された

Description

TLS1.3 連携機能で使用する、Resumption Master Secret の Ephemeral 鍵を用いて、New Session Ticket の情報から Pre Shared Key の Wrapped Key を生成するための API です。

Reentrant

非対応

4.2.15.12　R_TSIP_Tls13GeneratePskBinderKey

Format

```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_ Tls13GeneratePskBinderKey(
        tsip_tls13_handle_t *handle,
        tsip_tls13_ephemeral_pre_shared_key_index_t *pre_shared_key_index,
        tsip_hmac_sha_key_index_t *psk_binder_key_index
)
```

Parameters

handle	入力	同一セッションを示すハンドル番号(ワーク領域)
pre_shared_key_index	入力	Pre Shared Key の Ephemeral の Wrapped Key R_TSIP_Tls13GeneratePreSharedKey の出力 pre_shared_key_index を使用してください。
psk_binder_key_index	出力	Psk Binder Key の Ephemeral の Wrapped Key Psk Binder の生成に使用してください。 R_TSIP_Sha256HmacGenerateInit の入力 key_index に使用してください。

Return Values

TSIP_SUCCESS :	正常終了
TSIP_ERR_FAIL	内部エラーが発生
TSIP_ERR_RESOURCE_CONFLICT:	本処理に必要なハードウェアリソースが他の処理で使用されていることによるリソース衝突が発生
TSIP_ERR_KEY_SET	異常な Wrapped Key が入力された

Description

TLS1.3 連携機能で使用する、Binder Key の Wrapped Key を生成するための API です。

Reentrant

非対応

4.2.15.13　R_TSIP_Tls13GenerateResumptionHandshakeSecret

Format

```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_ Tls13GenerateResumptionHandshakeSecret(
        tsip_tls13_handle_t *handle,
        e_tsip_tls13_mode_t mode,
        tsip_tls13_ephemeral_pre_shared_key_index_t *pre_shared_key_index,
        tsip_tls13_ephemeral_shared_secret_key_index_t *shared_secret_key_index,
        tsip_tls13_ephemeral_handshake_secret_key_index_t *handshake_secret_key_index
)
```

Parameters

handle	入力	同一セッションを示すハンドル番号(ワーク領域)
mode	入力	実施するハンドシェイクプロトコル TSIP_TLS13_MODE_FULL_HANDSHAKE : Full Handshake TSIP_TLS13_MODE_RESUMPTION : Resumption TSIP_TLS13_MODE_0_RTT : 0-RTT
pre_shared_key_index	入力	Pre Shared Key の Ephemeral の Wrapped Key R_TSIP_Tls13GeneratePreSharedKey の出力 pre_shared_key_index を使用してください。
shared_secret_key_index	入力	Shared Secret の Ephemeral の Wrapped Key R_TSIP_Tls13GenerateEcdheSharedSecret の出力 shared_secret_key_index を使用してください。
handshake_secret_key_index	出力	Handshake Secret の Ephemeral の Wrapped Key R_TSIP_Tls13GenerateServerHandshakeTrafficKey、R_TSIP_Tls13GenerateClientHandshakeTrafficKey および R_TSIP_Tls13GenerateMasterSecret の入力 handshake_secret_key_index に使用してください。

Return Values

TSIP_SUCCESS :	正常終了
TSIP_FAIL	内部エラーが発生
TSIP_ERR_RESOURCE_CONFLICT:	本処理に必要なハードウェアリソースが他の処理で使用されていることによるリソース衝突が発生
TSIP_ERR_KEY_SET	異常な Wrapped Key が入力された

Description

TLS1.3 連携機能で使用する、R_TSIP_Tls13GeneratePreSharedKey で生成した Pre Shared Key の Wrapped Key を使用し、Handshake Secret の Wrapped Key を生成するための API です。

Pre Shared Key については、TSIP により生成したもののみを使用し、それ以外の Pre Shared Key についてはサポートしていません。

Reentrant

非対応

4.2.15.14　R_TSIP_Tls13Generate0RttApplicationWriteKey

Format

```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_ Tls13Generate0RttApplicationWriteKey(
        tsip_tls13_handle_t *handle,
        tsip_tls13_ephemeral_pre_shared_key_index_t *pre_shared_key_index,
        uint8_t *digest,
        tsip_aes_key_index_t *client_write_key_index
)
```

Parameters

handle	入力/出力	同一セッションを示すハンドル番号(ワーク領域)
pre_shared_key_index	入力	Pre Shared Key の Ephemeral の Wrapped Key R_TSIP_Tls13GeneratePreSharedKey の出力 pre_shared_key_index を使用してください。
digest	入力	SHA256 で演算したメッセージハッシュ ClientHello のハッシュ値を演算して入力してください。 ハッシュ値の演算には R_TSIP_Sha256Init/Update/Final を使用し、 R_TSIP_Sha256Final の出力 digest を入力に使用してください。
client_write_key_index	出力	Client Write Key の Ephemeral の Wrapped Key R_TSIP_Tls13EncryptInit の入力 key_index に使用してください。

Return Values

TSIP_SUCCESS :	正常終了
TSIP_ERR_FAIL	内部エラーが発生
TSIP_ERR_RESOURCE_CONFLICT:	本処理に必要なハードウェアリソースが他の処理で使用されていることによるリソース衝突が発生
TSIP_ERR_KEY_SET	異常な Wrapped Key が入力された

Description

TLS1.3 連携機能で使用する、R_TSIP_Tls13GeneratePreSharedKey で生成した Pre Shared Key を用いて、0-RTT で使用するための Client Write Key の Wrapped Key を生成するための API です。

0-RTT を使用する場合について、RFC8446 2.3 章に記載されているように、前方秘匿性が無いこと、リプレイ攻撃耐性が無いことがリスクとなります。この機能の使用については、本リスクを踏まえて判断してください。

Reentrant

非対応

4.2.15.15　R_TSIP_Tls13CertificateVerifyGenerate

Format

```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_ Tls13CertificateVerifyGenerate(
        uint32_t *key_index,
        e_tsip_tls13_signature_scheme_type_t signature_scheme,
        uint8_t *digest,
        uint8_t *certificate_verify,
        uint32_t *certificate_verify_len
)
```

Parameters

key_index	入力	署名生成用の秘密鍵の Wrapped Key R_TSIP_GenerateEccP256PrivateKeyIndex、 R_TSIP_GenerateEccP256RandomKeyIndex、 R_TSIP_UpdateEccP256PrivateKeyIndex、 R_TSIP_GenerateRsa2048PrivateKeyIndex、 R_TSIP_GenerateRsa2048RandomKeyIndex または R_TSIP_UpdateRsa2048PrivateKeyIndex の出力 key_pair_index または key_index を使用してください。 引数は uint32_t *でキャストしてから入力してください。
signature_scheme	入力	使用する署名アルゴリズム TSIP_TLS13_SIGNATURE_SCHEME_ECDSA_SECP256R1_SHA256 : ecdsa_secp256r1_sha256 TSIP_TLS13_SIGNATURE_SCHEME_RSA_PSS_RSAE_SHA256 : rsa_pss_rsae_sha256
digest	入力	SHA256 で演算したメッセージハッシュ (ClientHello||ServerHello||EncryptedExtensions ||CertificateRequest||Certificate||CertificateVerify ||ServerFinished||Certificate) のように、ハンドシェイクメッセージを連結した値のハッシュ値を演算して入力してください。 ハッシュ値の演算には R_TSIP_Sha256Init/Update/Final を使用し、 R_TSIP_Sha256Final の出力 digest を入力に使用してください。
certificate_verify	出力	CertificateVerify データは RFC8446 4.4.3 章の CertificateVerify の形式で出力されます。
certificate_verify_len	出力	certificate_verify のバイト長

Return Values

TSIP_SUCCESS :	正常終了
TSIP_ERR_FAIL	内部エラーが発生
TSIP_ERR_RESOURCE_CONFLICT:	本処理に必要なハードウェアリソースが他の処理で使用されていることによるリソース衝突が発生
TSIP_ERR_KEY_SET	異常な Wrapped Key が入力された
TSIP_ERR_PARAMETER	入力データが不正

Description

TLS1.3 連携機能で使用する、サーバに送信する CertificateVerify を生成するための API です。アルゴリズムは ecdsa_secp256r1_sha256 および rsa_pss_rsae_sha256 を使用します。

Reentrant

非対応

4.2.15.16　R_TSIP_Tls13CertificateVerifyVerification

Format

```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_ Tls13CertificateVerifyVerification(
        uint32_t *key_index,
        e_tsip_tls13_signature_scheme_type_t signature_scheme,
        uint8_t *digest,
        uint8_t *certificate_verify,
        uint32_t certificate_verify_len
)
```

Parameters

key_index	入力	暗号化された公開鍵 R_TSIP_TlsCertificateVerification または R_TSIP_TlsCertificateVerificationExtension の出力 encrypted_output_public_key を使用してください。
signature_scheme	入力	使用する署名アルゴリズム TSIP_TLS13_SIGNATURE_SCHEME_ECDSA_SECP256R1_SHA256 : ecdsa_secp256r1_sha256 TSIP_TLS13_SIGNATURE_SCHEME_RSA_PSS_RSAE_SHA256 : rsa_pss_rsae_sha256
digest	入力	SHA256 で演算したメッセージハッシュ (ClientHello||ServerHello||EncryptedExtensions ||CertificateRequest||Certificate) のように、ハンドシェイクメッセージを連結した値のハッシュ値を演算して入力してください。 ハッシュ値の演算には R_TSIP_Sha256Init/Update/Final を使用し、R_TSIP_Sha256Final の出力 digest を入力に使用してください。
certificate_verify	入力	CertificateVerify RFC8446 4.4.3 章の CertificateVerify の形式のデータを格納しているバッファの先頭アドレスを入力してください。
certificate_verify_len	入力	certificate_verify のバイト長

Return Values

TSIP_SUCCESS :	正常終了
TSIP_ERR_FAIL	内部エラーが発生、もしくは署名検証失敗
TSIP_ERR_RESOURCE_CONFLICT:	本処理に必要なハードウェアリソースが他の処理で使用されていることによるリソース衝突が発生
TSIP_ERR_KEY_SET	異常な Wrapped Key が入力された
TSIP_ERR_PARAMETER	入力データが不正

Description

TLS1.3 連携機能で使用する、サーバから受信した CertificateVerify を検証するための API です。アルゴリズムは ecdsa_secp256r1_sha256 および rsa_pss_rsae_sha256 を使用します。

Reentrant

非対応

4.2.15.17　R_TSIP_GenerateTls13SVP256EccKeyIndex

Format

```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_ TlsGenerateTls13SVP256EccKeyIndex(
        tsip_tls13_handle_t *handle,
        e_tsip_tls13_mode_t mode,
        tsip_tls_p256_ecc_key_index_t *key_index,
        uint8_t *ephemeral_ecdh_public_key
)
```

Parameters

handle	入力	同一セッションを示すハンドル番号(ワーク領域)
mode	入力	実施するハンドシェイクプロトコル TSIP_TLS13_MODE_FULL_HANDSHAKE : Full Handshake TSIP_TLS13_MODE_RESUMPTION : Resumption TSIP_TLS13_MODE_0_RTT : 0-RTT
key_index	出力	Ephemeral ECC の Wrapped Key R_TSIP_Tls13SVGenerateEcdheSharedSecret の入力 key_index に使用してください。
ephemeral_ecdh_public_key	出力	サーバへ送信する Ephemeral ECDH 公開鍵 Qx(256bit) || Qy(256bit)

Return Values

TSIP_SUCCESS :	正常終了
TSIP_ERR_FAIL:	内部エラーが発生
TSIP_ERR_RESOURCE_CONFLICT:	本処理に必要なハードウェアリソースが他の処理で使用されていることによるリソース衝突が発生

Description

TLS1.3 連携機能のサーバ機能で使用する、乱数から 256bit 素体上の楕円曲線暗号のための鍵ペアを生成するための API です。

Reentrant

非対応

4.2.15.18　R_TSIP_Tls13SVGenerateEcdheSharedSecret

Format

```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_ Tls13SVGenerateEcdheSharedSecret(
        e_tsip_tls13_mode_t mode,
        uint8_t *client_public_key,
        tsip_tls_p256_ecc_key_index_t *key_index,
        tsip_tls13_ephemeral_shared_secret_key_index_t *shared_secret_key_index
)
```

Parameters

mode	入力	実施するハンドシェイクプロトコル TSIP_TLS13_MODE_FULL_HANDSHAKE : Full Handshake TSIP_TLS13_MODE_RESUMPTION : Resumption TSIP_TLS13_MODE_0_RTT : 0-RTT
client_public_key	入力	クライアントから提供される公開鍵 Qx(256bit) || Qy(256bit)
key_index	入力	Ephemeral ECC 秘密鍵の Wrapped Key R_TSIP_GenerateTls13SVP256EccKeyIndex の出力 key_index を使用してください。
shared_secret_key_index	出力	Shared Secret の Ephemeral の Wrapped Key R_TSIP_Tls13SVGenerateHandshakeSecret および R_TSIP_Tls13SVGenerateResumptionHandshakeSecret の入力 shared_secret_key_index に使用してください。

Return Values

TSIP_SUCCESS :	正常終了
TSIP_ERR_FAIL:	内部エラーが発生
TSIP_ERR_RESOURCE_CONFLICT:	本処理に必要なハードウェアリソースが他の処理で使用されていることによるリソース衝突が発生
TSIP_ERR_KEY_SET	異常な Wrapped Key が入力された

Description

TLS1.3 連携機能のサーバ機能で使用する、サーバから提供される公開鍵とあらかじめ演算した秘密鍵を用いて、256bit 素体上の共有鍵である Shared Secret を計算し、Wrapped Key を生成するための API です。

該当暗号スイート: TLS_AES_128_GCM_SHA256, TLS_AES_128_CCM_SHA256

鍵交換の方式: ECDHE NIST P-256

Reentrant

非対応

4.2.15.19　R_TSIP_Tls13SVGenerateHandshakeSecret

Format

```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_ Tls13SVGenerateHandshakeSecret(
        tsip_tls13_ephemeral_shared_secret_key_index_t *shared_secret_key_index,
        tsip_tls13_ephemeral_handshake_secret_key_index_t *handshake_secret_key_index
)
```

Parameters

shared_secret_key_index	入力	Shared Secret の Ephemeral の Wrapped Key R_TSIP_Tls13SVGenerateEcdheSharedSecret の出力 shared_secret_key_index を使用してください。
handshake_secret_key_index	出力	Handshake Secret の Ephemeral の Wrapped Key R_TSIP_Tls13SVGenerateServerHandshakeTrafficKey、R_TSIP_Tls13SVGenerateClientHandshakeTrafficKey および R_TSIP_Tls13SVGenerateMasterSecret の入力 handshake_secret_key_index に使用してください。

Return Values

TSIP_SUCCESS :	正常終了
TSIP_ERR_FAIL:	内部エラーが発生
TSIP_ERR_RESOURCE_CONFLICT:	本処理に必要なハードウェアリソースが他の処理で使用されていることによるリソース衝突が発生
TSIP_ERR_KEY_SET	異常な Wrapped Key が入力された

Description

TLS1.3 連携機能のサーバ機能で使用する、Shared Secret の Ephemeral 鍵を用いて、Handshake Secret の Wrapped Key を生成するための API です。

Reentrant

非対応

4.2.15.20　R_TSIP_Tls13SVGenerateServerHandshakeTrafficKey

Format

```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_Tls13SVGenerateServerHandshakeTrafficKey(
        tsip_tls13_handle_t *handle,
        e_tsip_tls13_mode_t mode,
        tsip_tls13_ephemeral_handshake_secret_key_index_t *handshake_secret_key_index,
        uint8_t *digest,
        tsip_aes_key_index_t *server_write_key_index,
        tsip_hmac_sha_key_index_t *server_finished_key_index
)
```

Parameters

handle	出力	同一セッションを示すハンドル番号(ワーク領域)
mode	入力	実施するハンドシェイクプロトコル TSIP_TLS13_MODE_FULL_HANDSHAKE : Full Handshake TSIP_TLS13_MODE_RESUMPTION : Resumption TSIP_TLS13_MODE_0_RTT : 0-RTT
handshake_secret_key_index	入力	Handshake Secret の Ephemeral の Wrapped Key R_TSIP_Tls13SVGenerateHandshakeSecret または R_TSIP_Tls13SVGenerateResumptionHandshakeSecret の出力 handshake_secret_key_index を使用してください。
digest	入力	SHA256 で演算したメッセージハッシュ (ClientHello||ServerHello)のハッシュ値を演算して入力してください。 ハッシュ値の演算には R_TSIP_Sha256Init/Update/Final を使用し、 R_TSIP_Sha256Final の出力 digest を入力に使用してください。
server_write_key_index	出力	Server Write Key の Ephemeral の Wrapped Key R_TSIP_Tls13EncryptInit の入力 key_index に使用してください。
server_finished_key_index	出力	Server Finished Key の Ephemeral の Wrapped Key R_TSIP_Sha256HmacGenerateInit の入力 key_index に使用してください。

Return Values

TSIP_SUCCESS :	正常終了
TSIP_ERR_RESOURCE_CONFLICT:	本処理に必要なハードウェアリソースが他の処理で使用されていることによるリソース衝突が発生
TSIP_ERR_KEY_SET	異常な Wrapped Key が入力された

Description

TLS1.3 連携機能のサーバ機能で使用する、R_TSIP_Tls13SVGenerateHandshakeSecret で出力された Handshake Secret を用いて Server Write Key および Server Finished Key の Wrapped Key を生成するための API です。

Reentrant

非対応

4.2.15.21　R_TSIP_Tls13SVGenerateClientHandshakeTrafficKey

Format

```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_Tls13SVGenerateClientHandshakeTrafficKey(
        tsip_tls13_handle_t *handle,
        e_tsip_tls13_mode_t mode,
        tsip_tls13_ephemeral_handshake_secret_key_index_t *handshake_secret_key_index,
        uint8_t *digest,
        tsip_aes_key_index_t *client_write_key_index,
        tsip_tls13_ephemeral_client_finished_key_index_t *client_finished_key_index
)
```

Parameters

handle	出力	同一セッションを示すハンドル番号(ワーク領域)
mode	入力	実施するハンドシェイクプロトコル TSIP_TLS13_MODE_FULL_HANDSHAKE : Full Handshake TSIP_TLS13_MODE_RESUMPTION : Resumption TSIP_TLS13_MODE_0_RTT : 0-RTT
handshake_secret_key_index	入力	Handshake Secret の Ephemeral の Wrapped Key R_TSIP_Tls13SVGenerateHandshakeSecret または R_TSIP_Tls13SVGenerateResumptionHandshakeSecret の出力 handshake_secret_key_index を使用してください。
digest	入力	SHA256 で演算したメッセージハッシュ (ClientHello||ServerHello)のハッシュ値を演算して入力してください。 ハッシュ値の演算には R_TSIP_Sha256Init/Update/Final を使用し、 R_TSIP_Sha256Final の出力 digest を入力に使用してください。
client_write_key_index	出力	Client Write Key の Ephemeral の Wrapped Key R_TSIP_Tls13DecryptInit の入力 key_index に使用してください。
client_finished_key_index	出力	Client Finished Key の Ephemeral の Wrapped Key R_TSIP_Tls13SVClientHandshakeVerification の入力 client_finished_key_index に使用してください。

Return Values

TSIP_SUCCESS :	正常終了
TSIP_ERR_RESOURCE_CONFLICT:	本処理に必要なハードウェアリソースが他の処理で使用されていることによるリソース衝突が発生
TSIP_ERR_KEY_SET	異常な Wrapped Key が入力された

Description

TLS1.3 連携機能のサーバ機能で使用する、R_TSIP_Tls13SVGenerateHandshakeSecret で出力された Handshake Secret を用いて Client Write Key および Client Finished Key の Wrapped Key を生成するための API です。

Reentrant

非対応

4.2.15.22　R_TSIP_Tls13SVClientHandshakeVerification

Format

```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_ Tls13SVClientHandshakeVerification(
        e_tsip_tls13_mode_t mode,
        tsip_tls13_ephemeral_client_finished_key_index_t *client_finished_key_index,
        uint8_t *digest,
        uint8_t *client_finished
)
```

Parameters

mode	入力	実施するハンドシェイクプロトコル TSIP_TLS13_MODE_FULL_HANDSHAKE : Full Handshake TSIP_TLS13_MODE_RESUMPTION : Resumption TSIP_TLS13_MODE_0_RTT : 0-RTT
client_finished_key_index	入力	Client Finished Key の Ephemeral の Wrapped Key R_TSIP_Tls13SVGenerateClientHandshakeTrafficKey の出力 client_finished_key_index を使用してください。
digest	入力	SHA256 で演算したメッセージハッシュ (ClientHello||ServerHello||EncryptedExtensions ||CertificateRequest||Certificate||CertificateVerify ||ServerFinished||Certificate||CertificateVerify) のように、ハンドシェイクメッセージを連結した値のハッシュ値を演算して入力してください。 ハッシュ値の演算には R_TSIP_Sha256Init/Update/Final を使用し、R_TSIP_Sha256Final の出力 digest を入力に使用してください。
client_finished	入力	クライアントから提供される Finished 情報 R_TSIP_Tls13DecryptUpdate/Final により取得した Client Finished のデータを格納しているバッファの先頭アドレスを入力してください。

Return Values

TSIP_SUCCESS :	正常終了
TSIP_ERR_FAIL	内部エラーが発生
TSIP_ERR_RESOURCE_CONFLICT:	本処理に必要なハードウェアリソースが他の処理で使用されていることによるリソース衝突が発生
TSIP_ERR_KEY_SET	異常な Wrapped Key が入力された
TSIP_ERR_VERIFICATION_FAIL	検証で合格しなかった

Description

TLS1.3 連携機能のサーバ機能で使用する、クライアントから提供された Finished の情報を用いて Handshake を検証するための API です。

本 API からの戻り値が TSIP_ERR_VERIFICATION_FAIL であった場合、検証した Handshake を含む TLS 通信を中止してください。

Reentrant

非対応

4.2.15.23　R_TSIP_Tls13SVGenerateMasterSecret

Format

```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_ Tls13SVGenerateMasterSecret(
        tsip_tls13_handle_t *handle,
        e_tsip_tls13_mode_t mode,
        tsip_tls13_ephemeral_handshake_secret_key_index_t *handshake_secret_key_index,
        tsip_tls13_ephemeral_master_secret_key_index_t *master_secret_key_index
)
```

Parameters

handle	出力	同一セッションを示すハンドル番号(ワーク領域)
mode	入力	実施するハンドシェイクプロトコル TSIP_TLS13_MODE_FULL_HANDSHAKE : Full Handshake TSIP_TLS13_MODE_RESUMPTION : Resumption TSIP_TLS13_MODE_0_RTT : 0-RTT
handshake_secret_key_index	入力	Handshake Secret の Ephemeral の Wrapped Key R_TSIP_Tls13SVGenerateHandshakeSecret の出力 handshake_secret_key_index を使用してください。
master_secret_key_index	出力	Master Secret の Ephemeral の Wrapped Key R_TSIP_Tls13SVGenerateApplicationTrafficKey および R_TSIP_Tls13SVGenerateResumptionMasterSecret の入力 master_secret_key_index に使用してください。

Return Values

TSIP_SUCCESS :	正常終了
TSIP_ERR_FAIL	内部エラーが発生
TSIP_ERR_RESOURCE_CONFLICT:	本処理に必要なハードウェアリソースが他の処理で使用されていることによるリソース衝突が発生
TSIP_ERR_KEY_SET	異常な Wrapped Key が入力された

Description

TLS1.3 連携機能のサーバ機能で使用する、Handshake Secret の Ephemeral 鍵を用いて、Master Secret の Ephemeral の Wrapped Key を生成するための API です。

Reentrant

非対応

4.2.15.24　R_TSIP_Tls13SVGenerateApplicationTrafficKey

Format

```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_Tls13SVGenerateApplicationTrafficKey(
        tsip_tls13_handle_t *handle,
        e_tsip_tls13_mode_t mode,
        tsip_tls13_ephemeral_master_secret_key_index_t *master_secret_key_index,
        uint8_t *digest,
        tsip_tls13_ephemeral_app_secret_key_index_t *server_app_secret_key_index,
        tsip_tls13_ephemeral_app_secret_key_index_t *client_app_secret_key_index,
        tsip_aes_key_index_t *server_write_key_index,
        tsip_aes_key_index_t *client_write_key_index
)
```

Parameters

handle	入力/出力	同一セッションを示すハンドル番号(ワーク領域)
mode	入力	実施するハンドシェイクプロトコル TSIP_TLS13_MODE_FULL_HANDSHAKE : Full Handshake TSIP_TLS13_MODE_RESUMPTION : Resumption TSIP_TLS13_MODE_0_RTT : 0-RTT
master_secret_key_index	入力	Master Secret の Ephemeral の Wrapped Key R_TSIP_Tls13SVGenerateMasterSecret の出力 master_secret_key_index を使用してください。
digest	入力	SHA256 で演算したメッセージハッシュ (ClientHello||ServerHello||EncryptedExtensions ||CertificateRequest||Certificate||CertificateVerify ||ServerFinished) のように、ハンドシェイクメッセージを連結した値のハッシュ値を演算して入力してください。 ハッシュ値の演算には R_TSIP_Sha256Init/Update/Final を使用し、R_TSIP_Sha256Final の出力 digest を入力に使用してください。
server_app_secret_key_index	出力	Server Application Traffic Secret の Ephemeral の Wrapped Key R_TSIP_Tls13SVUpdateApplicationTrafficKey の入力 input_app_secret_key_index に使用してください。
client_app_secret_key_index	出力	Client Application Traffic Secret の Ephemeral の Wrapped Key R_TSIP_Tls13SVUpdateApplicationTrafficKey の入力 input_app_secret_key_index に使用してください。
server_write_key_index	出力	Server Write Key の Ephemeral の Wrapped Key R_TSIP_Tls13EncryptInit の入力 key_index に使用してください。
client_write_key_index	出力	Client Write Key の Ephemeral の Wrapped Key R_TSIP_Tls13DecryptInit の入力 key_index に使用してください。

Return Values

TSIP_SUCCESS :	正常終了
TSIP_ERR_RESOURCE_CONFLICT:	本処理に必要なハードウェアリソースが他の処理で使用されていることによるリソース衝突が発生
TSIP_ERR_KEY_SET	異常な Wrapped Key が入力された

Description

TLS1.3 連携機能のサーバ機能で使用する、Master Secret の Ephemeral 鍵を用いて、Application Traffic Secret の Wrapped Key を生成するための API です。併せて、Server Write Key および Client Write Key の Ephemeral の Wrapped Key を生成します。

Client Finished の受信を待たずにサーバから Application Data を送信する場合について、Client Finished の検証でエラーが発生した場合、サーバプログラムが改ざんされていないという条件下でのみエラーを検出することができます。この機能の使用については、本リスクを踏まえて判断してください。

Reentrant

非対応

4.2.15.25　R_TSIP_Tls13SVUpdateApplicationTrafficKey

Format

```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_Tls13SVUpdateApplicationTrafficKey(
        tsip_tls13_handle_t *handle,
        e_tsip_tls13_mode_t mode,
        e_tsip_tls13_update_key_type_t key_type,
        tsip_tls13_ephemeral_app_secret_key_index_t *input_app_secret_key_index,
        tsip_tls13_ephemeral_app_secret_key_index_t *output_app_secret_key_index,
        tsip_aes_key_index_t *app_write_key_index
)
```

Parameters

handle	入力/出力	同一セッションを示すハンドル番号(ワーク領域)
mode	入力	実施するハンドシェイクプロトコル TSIP_TLS13_MODE_FULL_HANDSHAKE : Full Handshake TSIP_TLS13_MODE_RESUMPTION : Resumption TSIP_TLS13_MODE_0_RTT : 0-RTT
key_type	入力	更新する鍵の種類 TSIP_TLS13_UPDATE_SERVER_KEY : Server Application Traffic Secret TSIP_TLS13_UPDATE_CLIENT_KEY : Client Application Traffic Secret
input_app_secret_key_index	入力	入力する Server/Client Application Traffic Secret の Ephemeral の Wrapped Key R_TSIP_Tls13SVGenerateApplicationTrafficKey の出力 server/client_app_secret_key_index または R_TSIP_Tls13SVUpdateApplicationTrafficKey の出力 output_app_secret_key_index のうち、key_type に指定した鍵の種類に適合した入力を使用してください。
output_app_secret_key_index	出力	出力する Server/Client Application Traffic Secret の Ephemeral の Wrapped Key key_type に指定した鍵の種類に対応した出力が得られます。 R_TSIP_Tls13SVUpdateApplicationTrafficKey の入力 input_app_secret_key_index に使用してください。
app_write_key_index	出力	Server/Client Write Key の Ephemeral の Wrapped Key key_type に指定した鍵の種類に対応した出力が得られます。 Server Write Key は R_TSIP_Tls13EncryptInit の入力 key_index に使用してください。 Client Write Key は R_TSIP_Tls13DecryptInit の入力 key_index に使用してください。

Return Values

TSIP_SUCCESS :	正常終了
TSIP_ERR_FAIL	内部エラーが発生
TSIP_ERR_RESOURCE_CONFLICT:	本処理に必要なハードウェアリソースが他の処理で使用されていることによるリソース衝突が発生
TSIP_ERR_KEY_SET	異常な Wrapped Key が入力された
TSIP_ERR_PARAMETER	入力データが不正

Description

TLS1.3 連携機能のサーバ機能で使用する、Application Traffic Secret を用いて、Application Traffic Secret の Wrapped Key と対応する暗号鍵の Wrapped Key を更新するための API です。

Reentrant

非対応

4.2.15.26　R_TSIP_Tls13SVGenerateResumptionMasterSecret

Format

```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_ Tls13SVGenerateResumptionMasterSecret(
        tsip_tls13_handle_t *handle,
        e_tsip_tls13_mode_t mode,
        tsip_tls13_ephemeral_master_secret_key_index_t *master_secret_key_index,
        uint8_t *digest,
        tsip_tls13_ephemeral_res_master_secret_key_index_t *res_master_secret_key_index
)
```

Parameters

handle	入力	同一セッションを示すハンドル番号(ワーク領域)
mode	入力	実施するハンドシェイクプロトコル TSIP_TLS13_MODE_FULL_HANDSHAKE : Full Handshake TSIP_TLS13_MODE_RESUMPTION : Resumption TSIP_TLS13_MODE_0_RTT : 0-RTT
master_secret_key_index	入力	Handshake Secret の Ephemeral の Wrapped Key R_TSIP_Tls13SVGenerateHandshakeSecret の出力 handshake_secret_key_index を使用してください。
digest	入力	SHA256 で演算したメッセージハッシュ (ClientHello||ServerHello||EncryptedExtensions ||CertificateRequest||Certificate||CertificateVerify ||ServerFinished||Certificate||CertificateVerify ||ClientFinished) のように、ハンドシェイクメッセージを連結した値のハッシュ値を演算して入力してください。 ハッシュ値の演算には R_TSIP_Sha256Init/Update/Final を使用し、 R_TSIP_Sha256Final の出力 digest を入力に使用してください。
res_master_secret_key_index	出力	Resumption Master Secret の Ephemeral の Wrapped Key R_TSIP_Tls13SVGeneratePreSharedKey の入力 res_master_secret_key_index に使用してください。

Return Values

TSIP_SUCCESS :	正常終了
TSIP_ERR_RESOURCE_CONFLICT:	本処理に必要なハードウェアリソースが他の処理で使用されていることによるリソース衝突が発生
TSIP_ERR_KEY_SET	異常な Wrapped Key が入力された

Description

TLS1.3 連携機能のサーバ機能で使用する、Master Secret の Ephemeral 鍵を用いて、Resumption Master Secret の Wrapped Key を生成するための API です。

RFC8446 より、Master Secret の Ephemeral の Wrapped Keymaster_secret_key_index は、本 API によって Resumption Master Secret の Wrapped Key を生成した後に削除してください。

Reentrant

非対応

4.2.15.27　R_TSIP_Tls13SVGeneratePreSharedKey

Format

```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_ Tls13SVGeneratePreSharedKey(
        tsip_tls13_handle_t *handle,
        e_tsip_tls13_mode_t mode,
        tsip_tls13_ephemeral_res_master_secret_key_index_t *res_master_secret_key_index,
        uint8_t *ticket_nonce,
        uint32_t ticket_nonce_len,
        tsip_tls13_ephemeral_pre_shared_key_index_t *pre_shared_key_index
)
```

Parameters

handle	入力	同一セッションを示すハンドル番号(ワーク領域)
mode	入力	実施するハンドシェイクプロトコル TSIP_TLS13_MODE_FULL_HANDSHAKE : Full Handshake TSIP_TLS13_MODE_RESUMPTION : Resumption TSIP_TLS13_MODE_0_RTT : 0-RTT
res_master_secret_key_index	入力	Resumption Master Secret の Ephemeral の Wrapped Key R_TSIP_Tls13SVGenerateResumptionMasterSecret の出力 res_master_secret_key_index を使用してください。
ticket_nonce	入力	サーバから提供された Ticket Nonce Ticket Nonce のサイズが 16 バイトの倍数でない場合は、16 バイトの倍数となるように 0 padding して入力してください。
ticket_nonce_len	入力	TicketNonce のバイト長
pre_shared_key_index	出力	Pre Shared Key の Ephemeral の Wrapped Key R_TSIP_Tls13SVGeneratePskBinderKey、 R_TSIP_Tls13SVGenerateResumptionHandshakeSecret および R_TSIP_Tls13SVGenerate0RttApplicationWriteKey の入力 pre_shared_key_index に使用してください。

Return Values

TSIP_SUCCESS :	正常終了
TSIP_FAIL	内部エラーが発生
TSIP_ERR_RESOURCE_CONFLICT:	本処理に必要なハードウェアリソースが他の処理で使用されていることによるリソース衝突が発生
TSIP_ERR_KEY_SET	異常な Wrapped Key が入力された

Description

TLS1.3 連携機能のサーバ機能で使用する、Resumption Master Secret の Ephemeral 鍵を用いて、New Session Ticket の情報から Pre Shared Key の Wrapped Key を生成するための API です。

Reentrant

非対応

4.2.15.28　R_TSIP_Tls13SVGeneratePskBinderKey

Format

```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_ Tls13SVGeneratePskBinderKey(
        tsip_tls13_handle_t *handle,
        tsip_tls13_ephemeral_pre_shared_key_index_t *pre_shared_key_index,
        tsip_hmac_sha_key_index_t *psk_binder_key_index
)
```

Parameters

handle	入力	同一セッションを示すハンドル番号(ワーク領域)
pre_shared_key_index	入力	Pre Shared Key の Ephemeral の Wrapped Key R_TSIP_Tls13SVGeneratePreSharedKey の出力 pre_shared_key_index を使用してください。
psk_binder_key_index	出力	Psk Binder Key の Ephemeral の Wrapped Key Psk Binder の生成に使用してください。 R_TSIP_Sha256HmacVerifyInit の入力 key_index に使用してください。

Return Values

TSIP_SUCCESS :	正常終了
TSIP_ERR_FAIL	内部エラーが発生
TSIP_ERR_RESOURCE_CONFLICT:	本処理に必要なハードウェアリソースが他の処理で使用されていることによるリソース衝突が発生
TSIP_ERR_KEY_SET	異常な Wrapped Key が入力された

Description

TLS1.3 連携機能のサーバ機能で使用する、Binder Key の Wrapped Key を生成するための API です。

Reentrant

非対応

4.2.15.29　R_TSIP_Tls13SVGenerateResumptionHandshakeSecret

Format

```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_ Tls13SVGenerateResumptionHandshakeSecret(
        tsip_tls13_handle_t *handle,
        e_tsip_tls13_mode_t mode,
        tsip_tls13_ephemeral_pre_shared_key_index_t *pre_shared_key_index,
        tsip_tls13_ephemeral_shared_secret_key_index_t *shared_secret_key_index,
        tsip_tls13_ephemeral_handshake_secret_key_index_t *handshake_secret_key_index
)
```

Parameters

handle	入力	同一セッションを示すハンドル番号(ワーク領域)
mode	入力	実施するハンドシェイクプロトコル TSIP_TLS13_MODE_FULL_HANDSHAKE : Full Handshake TSIP_TLS13_MODE_RESUMPTION : Resumption TSIP_TLS13_MODE_0_RTT : 0-RTT
pre_shared_key_index	入力	Pre Shared Key の Ephemeral の Wrapped Key R_TSIP_Tls13SVGeneratePreSharedKey の出力 pre_shared_key_index を使用してください。
shared_secret_key_index	入力	Shared Secret の Ephemeral の Wrapped Key R_TSIP_Tls13SVGenerateEcdheSharedSecret の出力 shared_secret_key_index を使用してください。
handshake_secret_key_index	出力	Handshake Secret の Ephemeral の Wrapped Key R_TSIP_Tls13SVGenerateServerHandshakeTrafficKey、R_TSIP_Tls13SVGenerateClientHandshakeTrafficKey および R_TSIP_Tls13SVGenerateMasterSecret の入力 handshake_secret_key_index に使用してください。

Return Values

TSIP_SUCCESS :	正常終了
TSIP_FAIL	内部エラーが発生
TSIP_ERR_RESOURCE_CONFLICT:	本処理に必要なハードウェアリソースが他の処理で使用されていることによるリソース衝突が発生
TSIP_ERR_KEY_SET	異常な Wrapped Key が入力された

Description

TLS1.3 連携機能のサーバ機能で使用する、R_TSIP_Tls13GeneratePreSharedKey で生成した Pre Shared Key の Wrapped Key を使用し、Handshake Secret の Wrapped Key を生成するための API です。

Pre Shared Key については、TSIP により生成したもののみを使用し、それ以外の Pre Shared Key についてはサポートしていません。

Reentrant

非対応

4.2.15.30　R_TSIP_Tls13SVGenerate0RttApplicationWriteKey

Format

```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_ Tls13SVGenerate0RttApplicationWriteKey(
        tsip_tls13_handle_t *handle,
        tsip_tls13_ephemeral_pre_shared_key_index_t *pre_shared_key_index,
        uint8_t *digest,
        tsip_aes_key_index_t *client_write_key_index
)
```

Parameters

handle	入力/出力	同一セッションを示すハンドル番号(ワーク領域)
pre_shared_key_index	入力	Pre Shared Key の Ephemeral の Wrapped Key R_TSIP_Tls13SVGeneratePreSharedKey の出力 pre_shared_key_index を使用してください。
digest	入力	SHA256 で演算したメッセージハッシュ ClientHello のハッシュ値を演算して入力してください。 ハッシュ値の演算には R_TSIP_Sha256Init/Update/Final を使用し、 R_TSIP_Sha256Final の出力 digest を入力に使用してください。
client_write_key_index	出力	Client Write Key の Ephemeral の Wrapped Key R_TSIP_Tls13DecryptInit の入力 key_index に使用してください。

Return Values

TSIP_SUCCESS :	正常終了
TSIP_ERR_FAIL	内部エラーが発生
TSIP_ERR_RESOURCE_CONFLICT:	本処理に必要なハードウェアリソースが他の処理で使用されていることによるリソース衝突が発生
TSIP_ERR_KEY_SET	異常な Wrapped Key が入力された

Description

TLS1.3 連携機能のサーバ機能で使用する、R_TSIP_Tls13GeneratePreSharedKey で生成した Pre Shared Key を用いて、0-RTT で使用するための Client Write Key の Wrapped Key を生成するための API です。

0-RTT を使用する場合について、RFC8446 2.3 章に記載されているように、前方秘匿性が無いこと、リプレイ攻撃耐性が無いことがリスクとなります。この機能の使用については、本リスクを踏まえて判断してください。

Reentrant

非対応

4.2.15.31　R_TSIP_Tls13SVCertificateVerifyGenerate

Format

```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_ Tls13SVCertificateVerifyGenerate(
        uint32_t *key_index,
        e_tsip_tls13_signature_scheme_type_t signature_scheme,
        uint8_t *digest,
        uint8_t *certificate_verify,
        uint32_t *certificate_verify_len
)
```

Parameters

key_index	入力	署名生成用の秘密鍵の Wrapped Key R_TSIP_GenerateEccP256PrivateKeyIndex、 R_TSIP_GenerateEccP256RandomKeyIndex、 R_TSIP_UpdateEccP256PrivateKeyIndex、 R_TSIP_GenerateRsa2048PrivateKeyIndex、 R_TSIP_GenerateRsa2048RandomKeyIndex または R_TSIP_UpdateRsa2048PrivateKeyIndex の出力 key_pair_index または key_index を使用してください。 引数は uint32_t *でキャストしてから入力してください。
signature_scheme	入力	使用する署名アルゴリズム TSIP_TLS13_SIGNATURE_SCHEME_ECDSA_SECP256R1_SHA256 : ecdsa_secp256r1_sha256 TSIP_TLS13_SIGNATURE_SCHEME_RSA_PSS_RSAE_SHA256 : rsa_pss_rsae_sha256
digest	入力	SHA256 で演算したメッセージハッシュ (ClientHello||ServerHello||EncryptedExtensions ||CertificateRequest||Certificate) のように、ハンドシェイクメッセージを連結した値のハッシュ値を演算して入力してください。 ハッシュ値の演算には R_TSIP_Sha256Init/Update/Final を使用し、 R_TSIP_Sha256Final の出力 digest を入力に使用してください。
certificate_verify	出力	CertificateVerify データは RFC8446 4.4.3 章の CertificateVerify の形式で出力されます。
certificate_verify_len	出力	certificate_verify のバイト長

Return Values

TSIP_SUCCESS :	正常終了
TSIP_ERR_FAIL	内部エラーが発生
TSIP_ERR_RESOURCE_CONFLICT:	本処理に必要なハードウェアリソースが他の処理で使用されていることによるリソース衝突が発生
TSIP_ERR_KEY_SET	異常な Wrapped Key が入力された
TSIP_ERR_PARAMETER	入力データが不正

Description

TLS1.3 連携機能のサーバ機能で使用する、クライアントに送信する CertificateVerify を生成するための API です。アルゴリズムは ecdsa_secp256r1_sha256 および rsa_pss_rsae_sha256 を使用します。

Reentrant

非対応

4.2.15.32　R_TSIP_Tls13SVCertificateVerifyVerification

Format

```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_ Tls13SVCertificateVerifyVerification(
        uint32_t *key_index,
        e_tsip_tls13_signature_scheme_type_t signature_scheme,
        uint8_t *digest,
        uint8_t *certificate_verify,
        uint32_t certificate_verify_len
)
```

Parameters

key_index	入力	暗号化された公開鍵 R_TSIP_TlsCertificateVerification または R_TSIP_TlsCertificateVerificationExtension の出力 encrypted_output_public_key を使用してください。
signature_scheme	入力	使用する署名アルゴリズム TSIP_TLS13_SIGNATURE_SCHEME_ECDSA_SECP256R1_SHA256 : ecdsa_secp256r1_sha256 TSIP_TLS13_SIGNATURE_SCHEME_RSA_PSS_RSAE_SHA256 : rsa_pss_rsae_sha256
digest	入力	SHA256 で演算したメッセージハッシュ (ClientHello||ServerHello||EncryptedExtensions ||CertificateRequest||Certificate||CertificateVerify ||ServerFinished||Certificate) のように、ハンドシェイクメッセージを連結した値のハッシュ値を演算して入力してください。 ハッシュ値の演算には R_TSIP_Sha256Init/Update/Final を使用し、R_TSIP_Sha256Final の出力 digest を入力に使用してください。
certificate_verify	入力	CertificateVerify RFC8446 4.4.3 章の CertificateVerify の形式のデータを格納しているバッファの先頭アドレスを入力してください。
certificate_verify_len	入力	certificate_verify のバイト長

Return Values

TSIP_SUCCESS :	正常終了
TSIP_ERR_FAIL	内部エラーが発生、もしくは署名検証失敗
TSIP_ERR_RESOURCE_CONFLICT:	本処理に必要なハードウェアリソースが他の処理で使用されていることによるリソース衝突が発生
TSIP_ERR_KEY_SET	異常な Wrapped Key が入力された
TSIP_ERR_PARAMETER	入力データが不正

Description

TLS1.3 連携機能のサーバ機能で使用する、クライアントから受信した CertificateVerify を検証するための API です。アルゴリズムは ecdsa_secp256r1_sha256 および rsa_pss_rsae_sha256 を使用します。

Reentrant

非対応

4.2.15.33　R_TSIP_Tls13EncryptInit

Format

```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_ Tls13EncryptInit(
        tsip_tls13_handle_t *handle,
        e_tsip_tls13_phase_t phase,
        e_tsip_tls13_mode_t mode,
        e_tsip_tls13_cipher_suite_t cipher_suite,
        tsip_aes_key_index_t *key_index,
        uint32_t payload_length
)
```

Parameters

handle	出力	TLS1.3 用ハンドラ(ワーク領域)
phase	入力	通信フェーズ TSIP_TLS13_PHASE_HANDSHAKE : ハンドシェイクフェーズ TSIP_TLS13_PHASE_APPLICATION : アプリケーションフェーズ
mode	入力	実施するハンドシェイクプロトコル TSIP_TLS13_MODE_FULL_HANDSHAKE : Full Handshake TSIP_TLS13_MODE_RESUMPTION : Resumption TSIP_TLS13_MODE_0_RTT : 0-RTT
cipher_suite	入力	暗号スイート TSIP_TLS13_CIPHER_SUITE_AES_128_GCM_SHA256 : TLS_AES_128_GCM_SHA256 TSIP_TLS13_CIPHER_SUITE_AES_128_CCM_SHA256 : TLS_AES_128_CCM_SHA256
key_index	入力	暗号化に使用する鍵の Ephemeral の Wrapped Key
payload_length	入力	暗号化するデータのバイト長

Return Values

TSIP_SUCCESS :	正常終了
TSIP_ERR_FAIL	内部エラーが発生
TSIP_ERR_RESOURCE_CONFLICT:	本処理に必要なハードウェアリソースが他の処理で使用されていることによるリソース衝突が発生
TSIP_ERR_KEY_SET	異常な Wrapped Key が入力された

Description

R_TSIP_Tls13EncryptInit()関数は、TLS1.3 通信データの暗号化を実行する準備を行い、その結果を第一引数“handle“に書き出します。handle は、続く R_TSIP_Tls13EncryptUpdate()関数および R_TSIP_Tls13EncryptFinal()関数で引数として使用されます。

Reentrant

非対応

4.2.15.34　R_TSIP_Tls13EncryptUpdate

Format

```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_ Tls13EncryptUpdate(
        tsip_tls13_handle_t *handle,
        uint8_t *plain,
        uint8_t *cipher,
        uint32_t plain_length
)
```

Parameters

handle	入力/出力	TLS1.3 用ハンドラ(ワーク領域)
plain	入力	平文データ領域
cipher	出力	暗号文データ領域
plain_length	入力	平文データ長

Return Values

TSIP_SUCCESS :	正常終了
TSIP_ERR_PARAMETER	入力データが不正
TSIP_ERR_PROHIBIT_FUNCTION	不正な関数が呼び出された

Description

R_TSIP_Tls13EncryptUpdate()関数は、第二引数“plain“で指定された平文から R_TSIP_Tls13EncryptInit()関数で指定された“key_index“を用いて暗号化します。本関数内部で plain の入力値が 16byte を超えるまでユーザが入力したデータをバッファリングします。暗号化結果は“plain“入力データが 16byte 以上になってから、第三引数で指定された“cipher“に出力します。入力する plain の総データ長は R_TSIP_Tls13EncryptInit()関数の payload_length で指定してください。本関数の plain_length には、ユーザが本関数を呼ぶ際に入力するデータ長を指定してください。入力値の plain は 16byte で割り切れない場合、パディング処理は関数内部で実施します。

plain と cipher は、同一アドレスの場合を除き、必ず領域が重ならないように配置してください。

Reentrant

非対応

4.2.15.35　R_TSIP_Tls13EncryptFinal

Format

```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_ Tls13EncryptFinal(
        tsip_tls13_handle_t *handle,
        uint8_t *cipher,
        uint32_t *cipher_length
)
```

Parameters

handle	入力/出力	TLS1.3 用ハンドラ(ワーク領域)
cipher	出力	暗号文データ領域
cipher_length	出力	暗号文データ長

Return Values

TSIP_SUCCESS :	正常終了
TSIP_ERR_FAIL	内部エラーが発生
TSIP_ERR_PARAMETER	入力データが不正
TSIP_ERR_PROHIBIT_FUNCTION	不正な関数が呼び出された

Description

R_TSIP_Tls13EncryptFinal()関数は、R_TSIP_Tls13EncryptUpdate()関数で入力した plain のデータ長に 16byte の端数データがある場合、第二引数で指定された“cipher"に端数分の暗号化したデータを出力します。このとき、16byte に満たない部分は 0padding されています。

Reentrant

非対応

4.2.15.36　R_TSIP_Tls13DecryptInit

Format

```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_ Tls13DecryptInit(
        tsip_tls13_handle_t *handle,
        e_tsip_tls13_phase_t phase,
        e_tsip_tls13_mode_t mode,
        e_tsip_tls13_cipher_suite_t cipher_suite,
        tsip_aes_key_index_t *key_index,
        uint32_t payload_length
)
```

Parameters

handle	出力	TLS1.3 用ハンドラ(ワーク領域)
phase	入力	通信フェーズ TSIP_TLS13_PHASE_HANDSHAKE : ハンドシェイクフェーズ TSIP_TLS13_PHASE_APPLICATION : アプリケーションフェーズ
mode	入力	実施するハンドシェイクプロトコル TSIP_TLS13_MODE_FULL_HANDSHAKE : Full Handshake TSIP_TLS13_MODE_RESUMPTION : Resumption TSIP_TLS13_MODE_0_RTT : 0-RTT
cipher_suite	入力	暗号スイート TSIP_TLS13_CIPHER_SUITE_AES_128_GCM_SHA256 : TLS_AES_128_GCM_SHA256 TSIP_TLS13_CIPHER_SUITE_AES_128_CCM_SHA256 : TLS_AES_128_CCM_SHA256
key_index	入力	復号に使用する鍵の Ephemeral の Wrapped Key
payload_length	入力	復号するデータのバイト長

Return Values

TSIP_SUCCESS :	正常終了
TSIP_ERR_FAIL	内部エラーが発生
TSIP_ERR_RESOURCE_CONFLICT:	本処理に必要なハードウェアリソースが他の処理で使用されていることによるリソース衝突が発生
TSIP_ERR_KEY_SET	異常な Wrapped Key が入力された

Description

R_TSIP_Tls13DecryptInit()関数は、TLS1.3 通信データの復号を実行する準備を行い、その結果を第一引数“handle“に書き出します。handle は、続く R_TSIP_Tls13DecryptUpdate()関数および R_TSIP_Tls13DecryptFinal()関数で引数として使用されます。

Reentrant

非対応

4.2.15.37　R_TSIP_Tls13DecryptUpdate

Format

```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_ Tls13DecryptUpdate(
        tsip_tls13_handle_t *handle,
        uint8_t *cipher,
        uint8_t *plain,
        uint32_t cipher_length
)
```

Parameters

handle	入力/出力	TLS1.3 用ハンドラ(ワーク領域)
cipher	入力	暗号文データ領域
plain	出力	平文データ領域
cipher_length	入力	暗号文データ長

Return Values

TSIP_SUCCESS :	正常終了
TSIP_ERR_PARAMETER	入力データが不正
TSIP_ERR_PROHIBIT_FUNCTION	不正な関数が呼び出された

Description

R_TSIP_Tls13DecryptUpdate()関数は、第二引数“cipher“で指定された暗号文から R_TSIP_Tls13DecryptInit()関数で指定された“key_index“を用いて復号します。本関数内部で cipher の入力値が 16byte を超えるまでユーザが入力したデータをバッファリングします。暗号化結果は“cipher“入力データが 16byte 以上になってから、第三引数で指定された“plain“に出力します。入力する cipher の総データ長は R_TSIP_Tls13DecryptInit()関数の payload_length で指定してください。本関数の cipher_length には、ユーザが本関数を呼ぶ際に入力するデータ長を指定してください。入力値の cipher は 16byte で割り切れない場合、パディング処理は関数内部で実施します。

plain と cipher は、同一アドレスの場合を除き、必ず領域が重ならないように配置してください。

Reentrant

非対応

4.2.15.38　R_TSIP_Tls13DecryptFinal

Format

```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_ Tls13DecryptFinal(
        tsip_tls13_handle_t *handle,
        uint8_t *plain,
        uint32_t *plain_length
)
```

Parameters

handle	入力/出力	TLS1.3 用ハンドラ(ワーク領域)
plain	出力	平文データ領域
plain_length	出力	平文データ長

Return Values

TSIP_SUCCESS :	正常終了
TSIP_ERR_FAIL	内部エラーが発生
TSIP_ERR_PARAMETER	入力データが不正
TSIP_ERR_PROHIBIT_FUNCTION	不正な関数が呼び出された

Description

R_TSIP_Tls13DecryptFinal()関数は、R_TSIP_Tls13DecryptUpdate()関数で入力した cipher のデータ長に 16byte の端数データがある場合、第二引数で指定された“plain“に端数分の復号したデータを出力します。このとき、16byte に満たない部分は 0padding されています。plain は 4 の倍数の RAM アドレスを指定してください。

Reentrant

非対応

4.2.16 ファームウェアアップデート

4.2.16.1 R_TSIP_StartUpdateFirmware

Format

e_tsip_err_t R_TSIP_StartUpdateFirmware(void)

Parameters

なし

Return Values

TSIP_SUCCESS :	正常終了
TSIP_ERR_RESOURCE_CONFLICT:	本処理に必要なハードウェアリソースが他の処理で使用されていることによるリソース衝突が発生

Description

ファームウェアアップデート状態へ移行します。

Reentrant

非対応

4.2.16.2 R_TSIP_GenerateFirmwareMAC

Format

```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_GenerateFirmwareMAC(
        uint32_t *InData_KeyIndex,
        uint32_t *InData_SessionKey,
        uint32_t *InData_UpProgram,
        uint32_t *InData_IV,
        uint32_t *OutData_Program,
        uint32_t MAX_CNT,
        TSIP_GEN_MAC_CB_FUNC_T p_callback,
        tsip_firmware_generate_mac_resume_handle_t *tsip_firmware_generate_mac_resume_handle
)
```

Parameters

InData_KeyIndex	入力	Wrapped Key Encryption Key
InData_SessionKey	入力	Encrypted Image Encryption Key
InData_UpProgram	入力	暗号化されたファームウェアを一時的に格納するための領域(デモプロジェクトでは、コードフラッシュメモリの 1 ブロックサイズ分確保)
InData_IV	入力	暗号化されたファームウェアを復号するための初期化ベクタ領域
OutData_Program	出力	復号されたファームウェアを一時的に格納するための領域(デモプロジェクトでは、コードフラッシュメモリの 1 ブロックサイズ分確保)
MAX_CNT	入力	暗号化されたファームウェアのワードサイズ+MAC サイズ ファームウェアのワードサイズは 4 の倍数である必要がある。MAC は 4 ワード（128bit）固定のため、ファームウェアのワードサイズ＋4 を入力。暗号化されたファームウェアは 16 ワードが最小であるため、MAX_CNT の最小値は 20
p_callback	入力	ユーザ側で対応が必要な場合に、複数回呼ばれる。対応内容は、列挙型 TSIP_FW_CB_REQ_TYPE で判別する。
tsip_firmware_generate_mac_resume_handle	入力	R_TSIP_GenerateFirmwareMAC 用ハンドラ(ワーク領域)

Return Values

TSIP_SUCCESS :	正常終了
TSIP_ERR_FAIL:	内部エラーが発生
TSIP_ERR_RESOURCE_CONFLICT:	本処理に必要なハードウェアリソースが他の処理で使用されていることによるリソース衝突が発生
TSIP_ERR_KEY_SET	異常な Wrapped Key が入力された
TSIP_ERR_CALLBACK_UNREGIST	p_callback の値が不正
TSIP_ERR_PARAMETER	入力データが不正
TSIP_RESUME_FIRMWARE_GENERATE_MAC	処理の続きがあります。API の再呼び出しが必要。

Description

暗号化されたファームウェアとファームウェアチェックサム値に対し、ファームウェアの復号と新たな MAC 値生成を行います。ユーザは復号されたファームウェアと新たな MAC 値をフラッシュ ROM に書き込むことでファームウェアアップデートを行うことができます。ファームウェアアップデートについては 3.14 ファームウェアアップデートを参照してください。

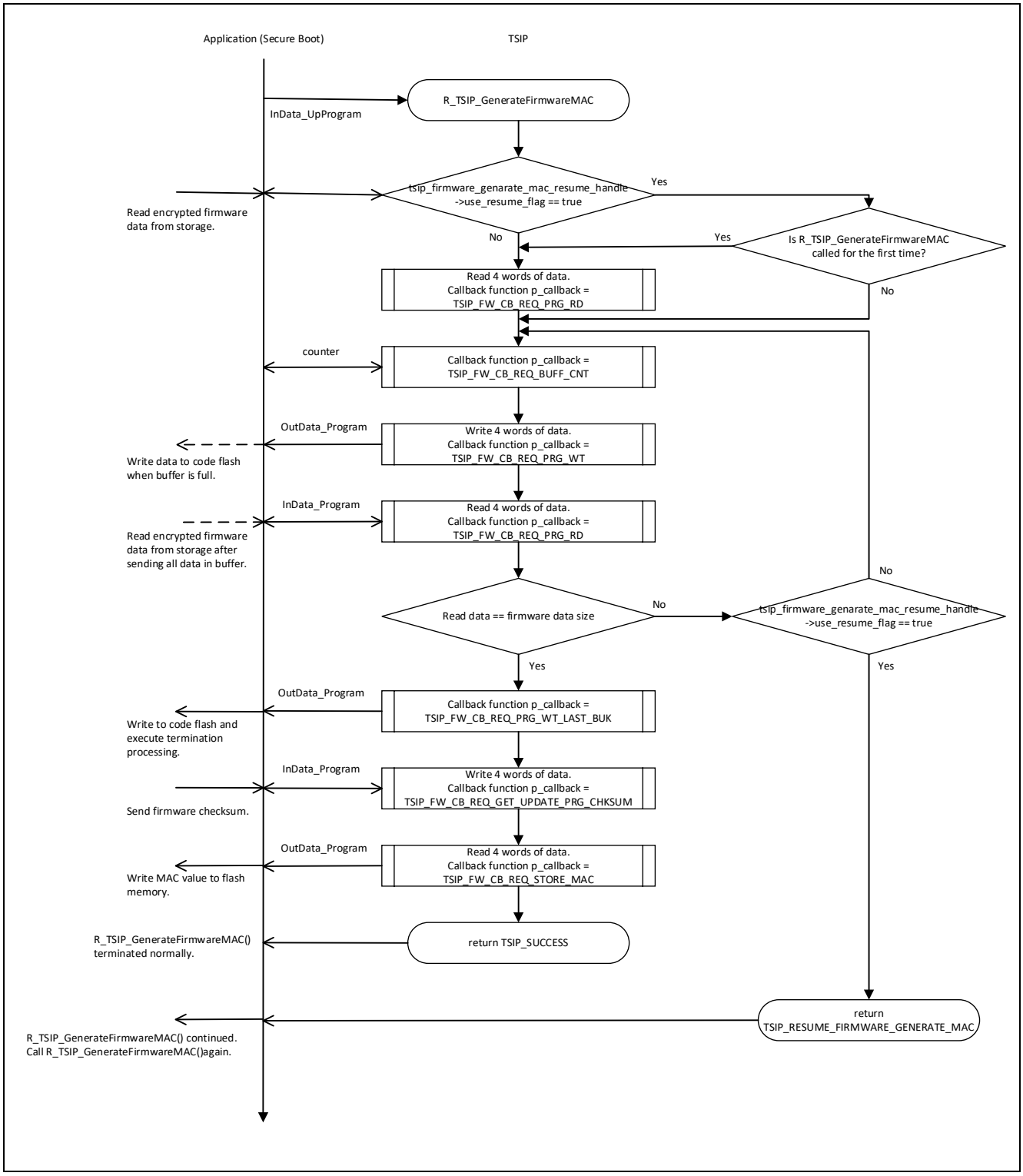

図 4-1　コールバック関数呼び出しフロー図

ファームウェアデータのリード、ライト処理を 4 ワード毎に行います。このため、第七引数 p_callback で登録されたコールバック関数を以下の順で呼び出します。()内はコールバック関数 p_callback 第一引数”req_type”の処理種別になります。

1. インクリメント調整(TSIP_FW_CB_REQ_BUFF_CNT)
2. 復号されたファームウェアを保存先へ書き込み(TSIP_FW_CB_REQ_PRG_WT)

3. 暗号化されたファームウェアの InData_UpProgram への格納(TSIP_FW_CB_REQ_PRG_RD)

コールバック関数内の処理は、毎回実施する必要はなく、確保した InData_Program /OutData_Program のサイズに応じて対応してください。

例えば、512 ワードのバッファを確保した場合は、512/4=128 回目にバッファ位置のインクリメント調整(TSIP_FW_CB_REQ_BUFF_CNT)、保存先への書き込み(TSIP_FW_CB_REQ_PRG_WT)、暗号化されたファームウェアを InData_UpProgram (TSIP_FW_CB_REQ_PRG_RD)に格納を実施します。

最後の保存先への書き込み要求は、TSIP_FW_CB_REQ_PRG_WT ではなく、req_type = TSIP_FW_CB_REQ_PRG_WT_LAST_BLK を指定します。

また、本 API は全ファームウェアの読み込み・書き込み完了後に、再度、コールバック関数 p_callback を呼び出します。ユーザはコールバック関数 p_callback の第一引数”req_type”が TSIP_FW_CB_REQ_GET_UPDATE_PRG_CHKSUM であることを確認後、チェックサム値を p_callback の第四引数”InData_UpProgram”に渡してください。また本 API はチェックサム値読み込み後、チェックサム値検証が正しければファームウェア MAC 値を生成します。その後、コールバック関数 p_callback の第一引数”req_type”が TSIP_FW_CB_REQ_STORE_MAC で第五引数”OutData_Program”で MAC 値をユーザに渡します。ユーザは MAC 値をフラッシュ領域に保存してください。

tsip_firmware_generate_mac_resume_handle->use_resume_flag=true に設定して呼び出した場合、ファームアップデート処理をすべて行わず、ファームアップデートの開始、更新関数として動作します。処理の続きがある場合、戻り値に TSIP_RESUME_FIRMWARE_GENERATE_MAC を返します。戻り値が TSIP_SUCCESS になるまで、R_TSIP_GenerateFirmwareMAC()を呼んでください。戻り値に TSIP_SUCCESS が返ったら、ファームアップデート処理は正常終了したことを示します。

Reentrant

非対応

4.2.16.3 R_TSIP_VerifyFirmwareMAC

Format

```
#include “r_tsip_rx_if.h”
e_tsip_err_t R_TSIP_VerifyFirmwareMAC(
        uint32_t *InData_Program,
        uint32_t MAX_CNT,
        uint32_t *InData_MAC
)
```

Parameters

InData_Program	入力	ファームウェア
MAX_CNT	入力	ファームウェアのワードサイズ+MAC サイズ 4 の倍数である必要がある。MAC は 4 ワード（16byte）固定のため、ファームウェアのワードサイズ＋4 を入力。 ファームウェアは 16 ワードが最小であるため、MAX_CNT の最小値は 20
InData_MAC	入力	比較する MAC 値(16byte)

Return Values

TSIP_SUCCESS :	正常終了
TSIP_ERR_FAIL:	内部エラーが発生
TSIP_ERR_RESOURCE_CONFLICT:	本処理に必要なハードウェアリソースが他の処理で使用されていることによるリソース衝突が発生
TSIP_ERR_PARAMETER	入力データが不正

Description

ファームウェアと MAC 値に対し、MAC 値の検証を行います。第三引数”InData_Mac"には R_TSIP_GenerateFirmwareMAC()で生成した MAC 値を渡してください。

MAC 検証アルゴリズムは AES-CMAC を使用しています。

Reentrant

非対応

4.2.16.4 TSIP_GEN_MAC_CB_FUNC_T 型

Format

```
#include “r_tsip_rx_if.h”
typedef void (*TSIP_GEN_MAC_CB_FUNC_T)(
        TSIP_FW_CB_REQ_TYPE req_type,
        uint32_t iLoop,
        uint32_t *counter,
        uint32_t *InData_UpProgram,
        uint32_t *OutData_Program,
        uint32_t MAX_CNT)
```

Parameters

req_type	入力	要求内容(TSIP_FW_CB_REQ_TYPE)
iLoop	入力	ループ回数(ワード単位)
counter	入力	領域参照用のオフセット
InData_UpProgram	入力	R_TSIP_GenerateFirmwareMAC()の第三引数”InData_UpProgram”と同アドレス
OutData_Program	入力/出力	R_TSIP_GenerateFirmwareMAC()の第五引数”OutData_Program”と同アドレス
MAX_CNT	入力	R_TSIP_GenerateFirmwareMAC()の第六引数”MAX_CNT”と同値

Return Values

なし

Description

R_TSIP_GenerateFirmwareMAC 関数で使用されます。同関数の第七引数で登録します。

復号されたファームウェアと MAC をユーザ側で保存するために使用します。

InData_UpProgram と OutData_Program の領域サイズは、4 の倍数であり、かつ、最低 4 ワード必要です。InData_UpProgram と OutData_Program は、同じサイズにしてください。デモプロジェクトは、コードフラッシュのブロックサイズにしています。

本コールバック関数では、R_TSIP_GenerateFirmwareMAC 関数の中で、複数の要求内容で呼び出されます。要求内容は、第一引数”req_type”に格納されます。

第一引数”req_type には、列挙型 TSIP_FW_CB_REQ_TYPE で定義された値が入ります。

```
typedef enum
{
    TSIP_FW_CB_REQ_PRG_WT = 0u,
    TSIP_FW_CB_REQ_PRG_RD,
    TSIP_FW_CB_REQ_BUFF_CNT,
    TSIP_FW_CB_REQ_PRG_WT_LAST_BLK,
    TSIP_FW_CB_REQ_GET_UPDATE_PRG_CHKSUM,
    TSIP_FW_CB_REQ_STORE_MAC,
}TSIP_FW_CB_REQ_TYPE;
```

この値によって、ユーザ側は必要な対応を行います。

<req_type = TSIP_FW_CB_REQ_PRG_WT>

復号されたファームウェアの保存要求です。

TSIP Module は、4 ワード単位で第五引数”OutData_Program"にデータを格納した後、その都度、本要求を出します。

要求のたびに処理する必要はありません。

ユーザ側で確保した領域に応じて、復号されたファームウェアを保存してください。例えば、8 ワード分領域を確保した場合は、2 回に 1 回復号されたファームウェアを保存してください。

復号されたサイズ数の合計は、 第二引数”iLoop”に格納されています。

本要求での”iLoop”最大値は、第六引数"MAX_CNT”から 4 ワード分引いた値です。最後の 4 ワードおよび保存できていないファームウェアは、<req_type = TSIP_FW_CB_REQ_PRG_WT_LAST_BLK>の要求で対応します。

<req_type = TSIP_FW_CB_REQ_PRG_RD>

更新する暗号化されたファームウェアの取得要求です。

TSIP Module は、4 ワード単位で復号処理をする前に、その都度、本要求を出します。

仕組みは、<req_type = TSIP_FW_CB_REQ_PRG_WT>と同じです。

ユーザ側で確保した領域に応じて、第四引数”InData_UpProgram”に格納してください。

<req_type = TSIP_FW_CB_REQ_BUFF_CNT>

第四引数”InData_UpProgram”と第五引数”OutData_Program”に参照するときのオフセット値要求です。

第三引数”counter”対して 4 ワードインクリメントした値を第三引数”counter”に戻してください。

第四引数”InData_UpProgram”と第五引数”OutData_Program”で確保したサイズを超える場合は、第三引数”counter”を初期値に戻してください。

<req_type = TSIP_FW_CB_REQ_PRG_WT_LAST_BLK>

暗号化されたファームウェアの最後のブロックに対して復号された時に要求を出します。復号されたファームウェアで保存できていない領域は、このタイミングで保存してください。

<req_type = TSIP_FW_CB_REQ_GET_UPDATE_PRG_CHKSUM>

更新するファームウェアのファームウェアチェックサム値の取得要求です。

チェックサム値を第四引数”InData_UpProgram”に格納してください。チェックサムのサイズは、16byte です。

<req_type = req_type = TSIP_FW_CB_REQ_STORE_MAC>

復号したファームウェアに対する MAC を出力します。

第五引数”OutData_Program”に MAC が格納されています。サイズは 16byte 分です。

第六引数”MAX_CNT”は、R_TSIP_GenerateFirmwareMAC()の第六引数”MAX_CNT”と同値です。

4.3　ユーザ定義関数

このセクションでは、TSIP ドライバが呼び出すユーザ定義関数について説明します。

4.3.1　user_sha384_fucntion

Format

```
#include “r_tsip_rx_config.h”
uint32_t user_sha384_function (
        uint8_t *message,
        uint8_t *digest,
        uint32_t message_length)
```

Parameters

message	入力	メッセージの先頭アドレス
digest	出力	ハッシュ計算結果格納アドレス(48 バイト)
message_length	入力	メッセージの有効バイト数

Return Values

0	ハッシュ値格納成功
0 以外	ハッシュ値格納失敗

Description

TSIP では SHA384 が HW でサポートされていないため、下記 API では署名生成/検証のために SHA384 関数をユーザが作成する必要があります。下記 API を使用するには、r_tsip_rx_config.h の TSIP_USER_SHA_384_ENABLED を有効にし、user_sha384_function 関数を用意してください。

・R_TSIP_EcdsaP384SignatureGenerate

・R_TSIP_EcdsaP384SignatureVerification

本関数はコンフィグレーション TSIP_USER_SHA_384_ENABLED を有効にすると定義可能となります。

引数 message で指定されたアドレスから引数 message_length バイトまでの領域について SHA384 ハッシュ計算を行います。

計算結果は、引数 digest に指定されたアドレスに格納してください。

4.3.2　user_lock_fucntion

Format

#include “r_tsip_rx_config.h”

void user_lock_function (

　　void)

Parameters

なし

Return Values

なし

Description

3.2 に示す、TSIP のアクセス衝突回避機能を使用するために、排他制御のための資源獲得関数をユーザが作成する必要があります。r_tsip_rx_config.h の TSIP_MULTI_THREADING を有効にし、user_lock_function 関数を実装してください。

アクセス衝突回避機能と併せてセキュアブート機能を使用する場合は、本関数をセキュアブート領域に配置してください。

4.3.3 user_unlock_fucntion

Format

#include “r_tsip_rx_config.h”

void user_unlock_function (

　　void)

Parameters

なし

Return Values

なし

Description

3.2 に示す、TSIP のアクセス衝突回避機能を使用するために、排他制御のための資源獲得関数をユーザが作成する必要があります。r_tsip_rx_config.h の TSIP_MULTI_THREADING を有効にし、user_unlock_function 関数を実装してください。

アクセス衝突回避機能と併せてセキュアブート機能を使用する場合は、本関数をセキュアブート領域に配置してください。

5.　鍵の注入と更新

本章では、TSIP ドライバが扱う暗号鍵を内蔵フラッシュメモリなどの不揮発性メモリに書き込む方法について説明します。

5.1　鍵の注入

お客様の製造工程でお客様の製品に安全に鍵を注入する手順を紹介します。

TSIP ドライバの鍵注入のメカニズムは 3.7.1 鍵の注入と更新を参照してください。

ユーザ鍵の注入には、ルネサスが提供する Renesas Key Wrap Service および RX ファミリ MCU 上で動作する鍵注入プログラムが必要です。また、Security Key Management Tool 等の補助ツールを利用して作業を簡略化することもできます。

本アプリケーションノート付属のデモプロジェクトに鍵注入プログラムの例が含まれていますので、参考にしてください。

ユーザ鍵の注入を実現する手順を以下に示します。

1. ユーザ鍵の注入に必要な鍵データを用意する
   任意のツールを利用し、注入するユーザ鍵のラップに使用する 256bit の UFPK および 128bit の IV を用意します。以下は OpenSSL を利用して UFPK と IV を生成する例です。

```
> openssl rand 32 > ufpk.bin
> openssl rand 16 > iv.bin
```

   Renesas Key Wrap Service(https://dlm.renesas.com/keywrap)を使用して、ufpk.bin を HRK でラップした W-UFPK を生成します。詳細な情報は、Renesas Key Wrap Service の FAQ をご参照ください。

   3.7.1 で説明するユーザ鍵ラップ方式（図 3-3 鍵注入および鍵更新時のユーザ鍵ラップ方式）に従い、ユーザ鍵を UFPK（ufpk.bin）でラップした Encrypted Key を生成します。

2. ユーザ鍵注入プログラムを作成する
   Step 1 で生成した Encrypted Key 、ufpk.bin 、および iv.bin を暗号アルゴリズム毎に用意されている鍵注入 API に入力して、ユーザ鍵の Wrapped Key を生成し、不揮発メモリに書き込むプログラムを作成します。
   鍵注入 API の使用方法は 3.7.1 鍵の注入と更新をご参照ください。

3. 鍵を注入する
   ユーザ鍵注入プログラムを RX ファミリ MCU 上で実行し、ユーザ鍵をフラッシュに注入します。ユーザ鍵注入プログラムに含まれる、鍵注入用のデータは鍵の注入完了後に消去することを推奨します。

Step 1、Step 2 の作業を補助するツールとして、Secure Key Management Tool があります。ツールの詳細は、5.3 を参照してください。

5.2　鍵の更新

フィールドでお客様の製品に安全に鍵を注入または更新する手順を紹介します。

TSIP ドライバの鍵更新のメカニズムは 3.7.1 鍵の注入と更新を参照してください。

フィールドでユーザ鍵の注入や更新を行うためには、あらかじめお客様製品のアプリケーションプログラムに鍵更新を実現する機能を持たせておく必要があります。

鍵更新機能は KUK と鍵更新 API を利用して実現します。KUK は製品製造工程において、5.1 鍵の注入で紹介する方法を用いてフラッシュに書き込んでおく必要があります。

ユーザ鍵の更新を実現する手順を以下に示します。

1. 鍵更新機能を持つユーザアプリケーションを作成する
   鍵更新 API を使用して鍵更新機能を持つユーザアプリケーションを作成します。鍵更新機能として、Encrypted Key を何らかの通信インタフェースを用いて受け取り、鍵更新 API を用いて Wrapped Key に変換、その後フラッシュに書き込む処理が必要です。
   鍵更新 API の使い方は 3.7.1 鍵の注入と更新を参照してください。

2. KUK とユーザ鍵をデバイスに注入する
   5.1 を参照して、KUK を含むユーザ鍵注入プログラムを作成し、KUK とユーザ鍵をフラッシュに注入します。ユーザ鍵注入プログラムに含まれる、鍵注入用のデータは鍵の注入完了後に消去することを推奨します。

3. 鍵更新機能を持つユーザアプリケーションプログラムをデバイスにプログラムする
   任意のプログラミング方法で、フラッシュに鍵更新機能を持つユーザアプリケーションプログラムを書き込みます。

4. アップデートするユーザ鍵データを作成する
   3.7.1 で説明するユーザ鍵ラップ方式（図 3-3 鍵注入および鍵更新時のユーザ鍵ラップ方式）に従い、ユーザ鍵を KUK でラップした Encrypted Key を生成します。

5. ユーザ鍵を更新する
   Encrypted Key を RX ファミリ MCU 上で動作する鍵更新機能を持つユーザアプリケーションプログラムに渡します。ユーザアプリケーションプログラムの鍵更新機能が動作することで、RX ファミリ MCU のフラッシュ上に格納された鍵の更新を実現します。ユーザアプリケーションプログラムに持たせる機能次第で、新たなユーザ鍵の注入を行うことも可能です。

Step 4 の作業を補助するツールとして、Secure Key Management Tool があります。ツールの詳細は、5.3 を参照してください。

5.3　Security Key Management Tool を使用した Encrypted Key 生成方法

暗号化されたユーザ鍵(Encrypted Key)を生成するために、Security Key Management Tool を使用することが可能です。

Security Key Management Tool はコマンドラインインタフェース版(CLI 版)も用意しているため、工場などの生産工程でも簡単に扱うことが可能です。

Security Key Management Tool

https://www.renesas.com/software-tool/security-key-management-tool

Security Key Management Tool の使用方法の詳細はユーザーズマニュアルをご確認ください。

5.3.1　鍵の注入手順

鍵を注入するときに使用する鍵のファイル生成手順を示します。

以下の例では、AES の C ソースの生成手順を示します。

5.3.1.1　CLI 版を使用する場合の手順

1. UFPK の生成

   ターミナルソフトで genufpk コマンドを実行します。

> skmt.exe /genufpk

/ufpk "22222222222222222222222222222222111111111111111111111111111111111"

/output "C:\work\ufpk.key"

```
C:¥Renesas¥SecurityKeyMangementTool¥cli>skmt.exe /genufpk /ufpk "22222222222222222222222222222222
11111111111111111111111111111111" /output "C:¥work¥ufpk.key"
UFPK: 22222222222222222222222222222222111111111111111111111111111111111
Output File: C:¥work¥ufpk.key
```

図 5-1　genufpk コマンド実行結果

2. W-UFPK の取得

   1 で生成した ufpk.key ファイルを Renesas Key Wrap service(https://dlm.renesas.com/keywrap)に送付して W-UFPK を取得します。

   詳細な取得情報は、Renesas Key Wrap Service の FAQ をご参照ください。

3. AES128 鍵ファイルを C ソースファイルで生成

   ターミナルソフトで genkey コマンドを実行します。

> skmt.exe /genkey /iv "55aa55aa55aa55aa55aa55aa55aa55aa" /ufpk file="C:\work\ufpk.key"
/wufpk file="C:\work\ufpk.key_enc.key" /mcu "RX-TSIP" /keytype "AES-128" /key
"11111111222222223333333344444444" /filetype "csource" /keyname "euk_aes128"
/output "C:\work\euk_aes128.c"

   UFPK ファイルは手順 1、W-UFPK ファイルは手順 2 で生成したものを使用します。

```
C:¥Renesas¥SecurityKeyMangementTool¥cli>skmt.exe /genkey /iv "55aa55aa55aa55aa55aa55aa55aa55aa" /
ufpk file="C:¥work¥ufpk.key" /wufpk file="C:¥work¥ufpk.key_enc.key" /mcu "RX-TSIP" /keytype "AES-
128" /key "11111111222222223333333344444444" /filetype "csource" /keyname "euk_aes128" /output "C
:¥work¥euk_aes128.c"
Output File: C:¥work¥euk_aes128.h
Output File: C:¥work¥euk_aes128.c
UFPK: 22222222222222222222222222222222111111111111111111111111111111111
W-UFPK: 00000001DF741B6058AFDFE2DD186EA0AF168453F2B1F311ABAF6133AC68926B0A475130
IV: 55AA55AA55AA55AA55AA55AA55AA55AA
Encrypted key: 9206ACBC87367501A27EB48604AC77C6CB6CF84B4EBFF149B79DD6C0018FE0F9
```

図 5-2　genkey コマンド実行例

5.3.1.2　GUI 版を使用する場合の手順

1. MCU/MPU と暗号エンジンの選択
   [**概要**]タブで MCU/MPU と暗号エンジンを選択します。

図 5-3　[**概要**]タブ

2. UFPK を生成
   [UFPK **生成**] tab に UFPK 値を設定して、拡張子*.key というファイル名で、UFPK ファイルを生成します。この例では ufpk.key というファイル名を使用しています。

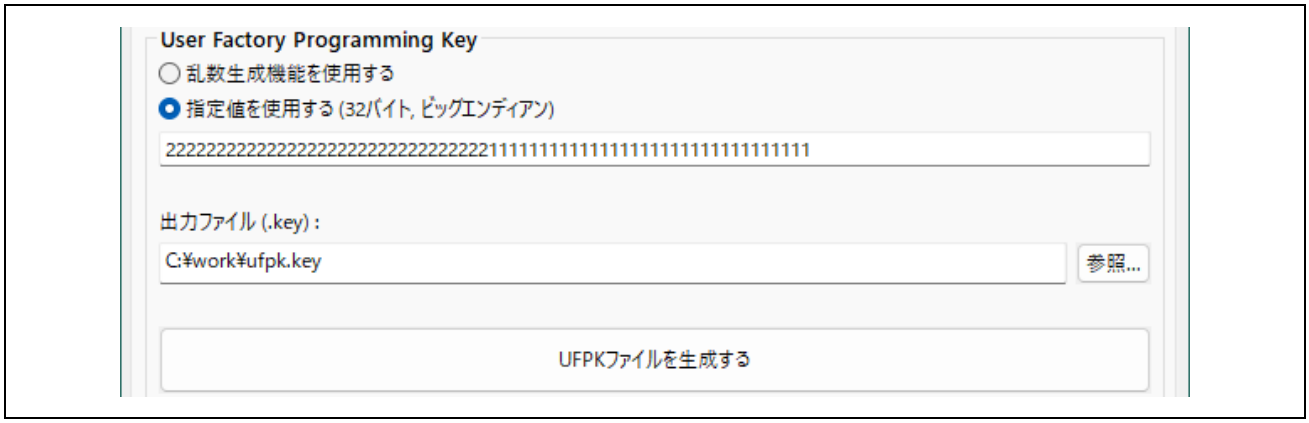

図 5-4　[UFPK **生成**]タブ 指定値で UFPK を生成する場合の例

“**UFPK ファイルを生成する**”ボタンを押すと UFPK ファイルが生成されます。正常にファイルが生成された場合、以下のような実行結果が出力されます。

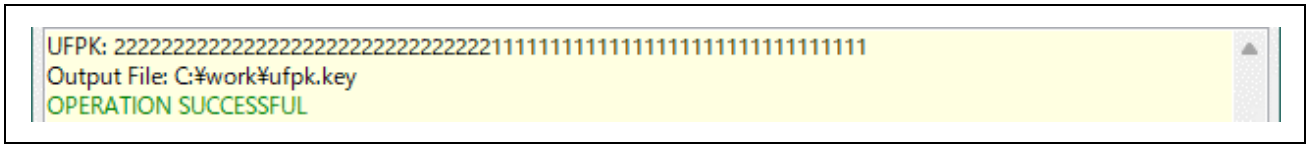

図 5-5　[UFPK **生成**]タブ 実行結果

3. W-UFPK の取得
   2 で生成した ufpk.key ファイルを Renesas Key Wrap service(https://dlm.renesas.com/keywrap)に送付して W-UFPK を取得します。
   詳細な取得情報は、Renesas Key Wrap Service の FAQ をご参照ください。

4. AES128 鍵ファイルを C ソースファイルで生成
   [**鍵のラッピング**]タブで AES 128 鍵ファイルを生成します。
   [**鍵の種類**]タブで”AES(128 bits)”を選択後、[**鍵データ**]タブで AES128 の鍵データを入力してください。
   “Wrapping key”には、手順 2 で生成した UFPK ファイルと手順 3 で取得した W-UFPK ファイルを設定してください。フォーマットに”**C ソース**”を選択します。

図 5-6　[**鍵のラッピング**] - [**鍵の種類**]タブ AES128 鍵ファイル C ソース出力設定例

図 5-7　[**鍵のラッピング**] - [**鍵のデータ**]タブ AES128 鍵ファイルを C ソース出力設定例

正常に終了すると以下のように表示されます。

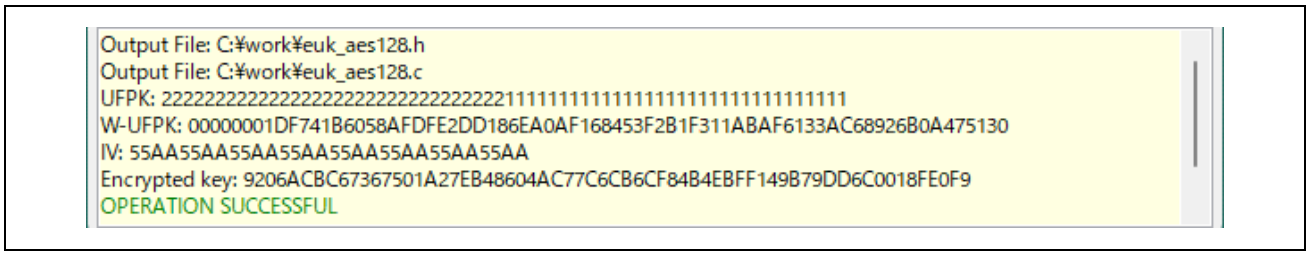

図 5-8　[**鍵のラッピング**]タブ 実行結果

出力された C ソースファイルのデータを、5.3.1.1 と同様の方法で組み込みます。

5.3.2　鍵更新時の操作手順

鍵を更新する場合、KUK をあらかじめ注入しておくことで、新たな UFPK を用意することなく鍵の更新を可能にします。

5.3.2.1　CLI 版を使用する場合の手順

1. UFPK の生成
   ターミナルソフトで genufpk コマンドを実行します。
   > skmt.exe /genufpk
   /ufpk "ec6b8fa5c0d5da5142ccaf3a31aebeae2346cfe7ef644b9b6b70523cba0f5c5c"
   /output "C:\work\ufpk.key"

```
C:¥Renesas¥SecurityKeyMangementTool¥cli>skmt.exe /genufpk /ufpk "ec6b8fa5c0d5da5142ccaf3a3
1aebeae2346cfe7ef644b9b6b70523cba0f5c5c" /output "C:¥work¥ufpk.key"
UFPK: EC6B8FA5C0D5DA5142CCAF3A31AEBEAE2346CFE7EF644B9B6B70523CBA0F5C5C
Output File: C:¥work¥ufpk.key
```

図 5-9　genufpk コマンド実行結果

2. W-UFPK の取得
   手順 1 で生成した ufpk.key ファイルを Renesas Key Wrap service(https://dlm.renesas.com/keywrap)に送って、W-UFPK ファイルを受け取ります。
   詳細は、Renesas Key Wrap Service の FAQ を参照してください。

3. KUK の生成
   ターミナルソフトで genkuk コマンドを実行します。
   > skmt.exe /genkuk /output "C:\work\kuk.key"
   /kuk "d0aec19726cbc0e2fb403866b9b465a6c0d05b7a60362d5f435f9a3e98c79084"

```
C:¥Renesas¥SecurityKeyMangementTool¥cli>skmt.exe /genkuk /output "C:¥work¥kuk.key" /kuk "d
0aec19726cbc0e2fb403866b9b465a6c0d05b7a60362d5f435f9a3e98c79084"
KUK: D0AEC19726CBC0E2FB403866B9B465A6C0D05B7A60362D5F435F9A3E98C79084
Output File: C:¥work¥kuk.key
```

図 5-10 genkuk コマンド実行結果

4. KUK を UFPK で暗号化した Encrypted User Key ファイルの生成
   ターミナルソフトで genkey コマンドを実行し、KUK を UFPK で暗号化します。

> skmt.exe /genkey /ufpk file="c:\work\ufpk.key" /wufpk file="C:\work\ufpk.key_enc.key" /mcu "RX-TSIP" /keytype "key-update-key" /key file="C:\work\kuk.key" /filetype "csource" /output "C:\work\kuk.c"

手順 1 で生成した UFPK ファイル、手順 2 で受け取った W-UFPK ファイルを使用します。

図 5-11 genkey コマンド実行例

ここで出力した C ソースファイルのデータを利用し、お客様のプログラム内で、R_TSIP_GenerateUpdateKeyRingKeyIndex()を実行して、KUK を注入してください。

続いて、市場で鍵を更新する場合の、KUK を使ったラッピング方法を示します。

5. RSA 2048 公開鍵の暗号化ファイルの生成
   ターミナルソフトで genkey コマンドを実行します。

```
> skmt.exe /genkey /kuk file="C:\work\kuk.key" /mcu "RX-TSIP" /keytype "RSA-2048-public"
        /key "bad47a84c1782e4dbdd913f2a261fc8b65838412c6e45a2068ed6d7f16e9cdf4
              462b39119563cafb74b9cbf25cfd544bdae23bff0ebe7f6441042b7e109b9a8a
              faa056821ef8efaab219d21d6763484785622d918d395a2a31f2ece8385a8131
              e5ff143314a82e21afd713bae817cc0ee3514d4839007ccb55d68409c97a18ab
              62fa6f9f89b3f94a2777c47d6136775a56a9a0127f682470bef831fbec4bcd7b
              5095a7823fd70745d37d1bf72b63c4b1b4a3d0581e74bf9ade93cc4614861755
              3931a79d92e9e488ef47223ee6f6c061884b13c9065b591139de13c1ea292749
              1ed00fb793cd68f463f5f64baa53916b46c818ab99706557a1c2d50d232577d1
              00010001"
              /filetype "csource" /keyname "rsa2048public"
              /output "C:\work\rsa2048public.c"
```

KUK ファイルは手順 3 で生成したファイルを使用します。

```
C:¥Renesas¥SecurityKeyMangementTool¥cli>skmt.exe /genkey /kuk file="C:¥work¥kuk.key" /mcu
"RX-TSIP" /keytype "RSA-2048-public" /key "bad47a84c1782e4dbdd913f2a261fc8b65838412c6e45a2
068ed6d7f16e9cdf4462b39119563cafb74b9cbf25cfd544bdae23bff0ebe7f6441042b7e109b9a8afaa056821
ef8efaab219d21d6763484785622d918d395a2a31f2ece8385a8131e5ff143314a82e21afd713bae817cc0ee35
14d4839007ccb55d68409c97a18ab62fa6f9f89b3f94a2777c47d6136775a56a9a0127f682470bef831fbec4bc
d7b5095a7823fd70745d37d1bf72b63c4b1b4a3d0581e74bf9ade93cc46148617553931a79d92e9e488ef47223
ee6f6c061884b13c9065b591139de13c1ea2927491ed00fb793cd68f463f5f64baa53916b46c818ab99706557a
1c2d50d232577d100010001" /filetype "csource" /keyname "rsa2048public" /output "C:¥work¥rsa
2048public.c"
Output File: C:¥work¥rsa2048public.h
Output File: C:¥work¥rsa2048public.c
KUK: D0AEC19726CBC0E2FB403866B9B465A6C0D05B7A60362D5F435F9A3E98C79084
IV: 9A537E8D13E62700DAF2DB0A44E770A9
Encrypted key: EFEB764914ACE4DA87737EC90A2B1B14520A7C18992DCDB7E945540A3220165E3E022CBC11F
B1A0C39563C8702ECF4C931B1BE4198CCA0DCA373D1395F7046D9AB02AF7D93C5512E9438370DB2E5027F4BFB8
606409F81D7CDB138F446474C9C04BA9613DC928EDD09B9C5D2002F54CF739A94FD8C5CD694A80AD12D9B00067
2EBBEE9D24298717D47FB05092A8178D173B45EA2BA47FA0C742084783FF718C593F8F6E2AC8E6013A9CA4EA25
73F8619CEF0B97DDC2F184665E42FF89009B9E8C89C5115D3733273E1432BD97EABB9E9190589F225E287F5C79
25597810F07E7A478BA0A464590C53A881739B8F89CC372B7055DD02051105113E2179D39ADE17D3420BB1F506
F96A345A4B6FB131A96E089C2ADAF5772D7F8F16A7F5B703FD3
```

図 5-12 genkey コマンド実行例

出力された C ソースファイルを取り込んだデータをデバイスの外部インタフェースから入力し、R_TSIP_Update2048PublicKeyIndex()の引数に渡してください。更新する RSA2048 公開鍵の Wrapped Key を出力します。

5.3.2.2　GUI 版を使用する場合の手順

1. MCU/MPU と暗号エンジンの選択
   [**概要**]タブで MCU/MPU と暗号エンジンを選択します。

図 5-13 [**概要**]タブ

2. UFPK の生成
   [**UFPK 生成**]タブで UFPK 値をセットして、拡張子*.key というファイル名の UFPK ファイルを生成します。この例では ufpk.key というファイル名を使用します。

図 5-14 [**UFPK 生成**]タブ UFPK 生成設定例

“UFPK **ファイルを生成する**”ボタンを押すと UFPK ファイルが生成されます。
正常終了すると以下の表示がされます。

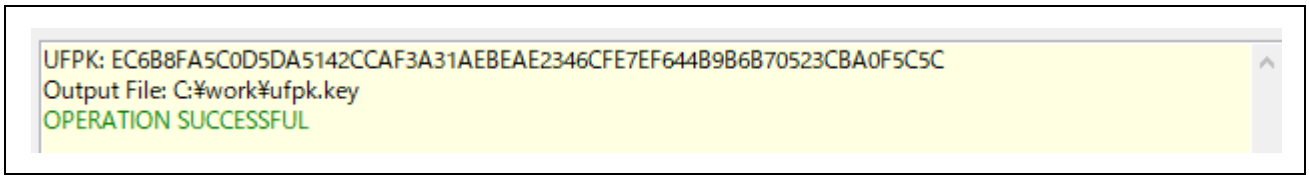

図 5-15 [**UFPK 生成**]タブ 実行結果

3. W-UFPK 取得
   2 で生成した ufpk.key ファイルを Renesas Key Wrap service(https://dlm.renesas.com/keywrap)に送付して W-UFPK を取得します。
   詳細な取得情報は、Renesas Key Wrap Service の FAQ をご参照ください。

4. KUK の生成

[**KUK 生成**]タブでツールを使って KUK 用のランダム値を生成するか、256bit の KUK を入力するか選択します。出力ファイル名は拡張子*.key という名前で設定します。この例では kuk.key というファイル名を使用します。

図 5-16 [**KUK 生成**]タブ KUK ファイル生成設定例

“KUK **ファイルを生成する**”ボタンを押すと KUK ファイルが生成されます。正常終了すると以下のように表示されます。

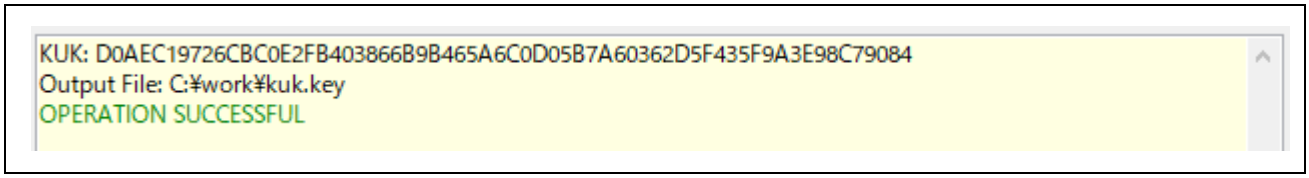

図 5-17 [**KUK 生成**]タブ 実行結果

5. C ソースファイルフォーマットの KUK ファイルの生成
   [**鍵のラッピング**] タブの [**鍵の種類**] タブで KUK を選択します。[**鍵のデータ**]タブで、前のステップで生成した KUK ファイル名(この例では kuk.key)を入力します。[**鍵のラッピング**]は "UFPK "を選択し、手順 2 で生成した UFPK ファイルと手順 3 で取得した W-UFPK ファイルを選択します。この例では簡略化のため、IV は"**乱数生成機能を使用する**"を選択します。"**出力**"パネルで、フォーマットとして「C ソース」を選択し、C ソースファイル名を入力します。

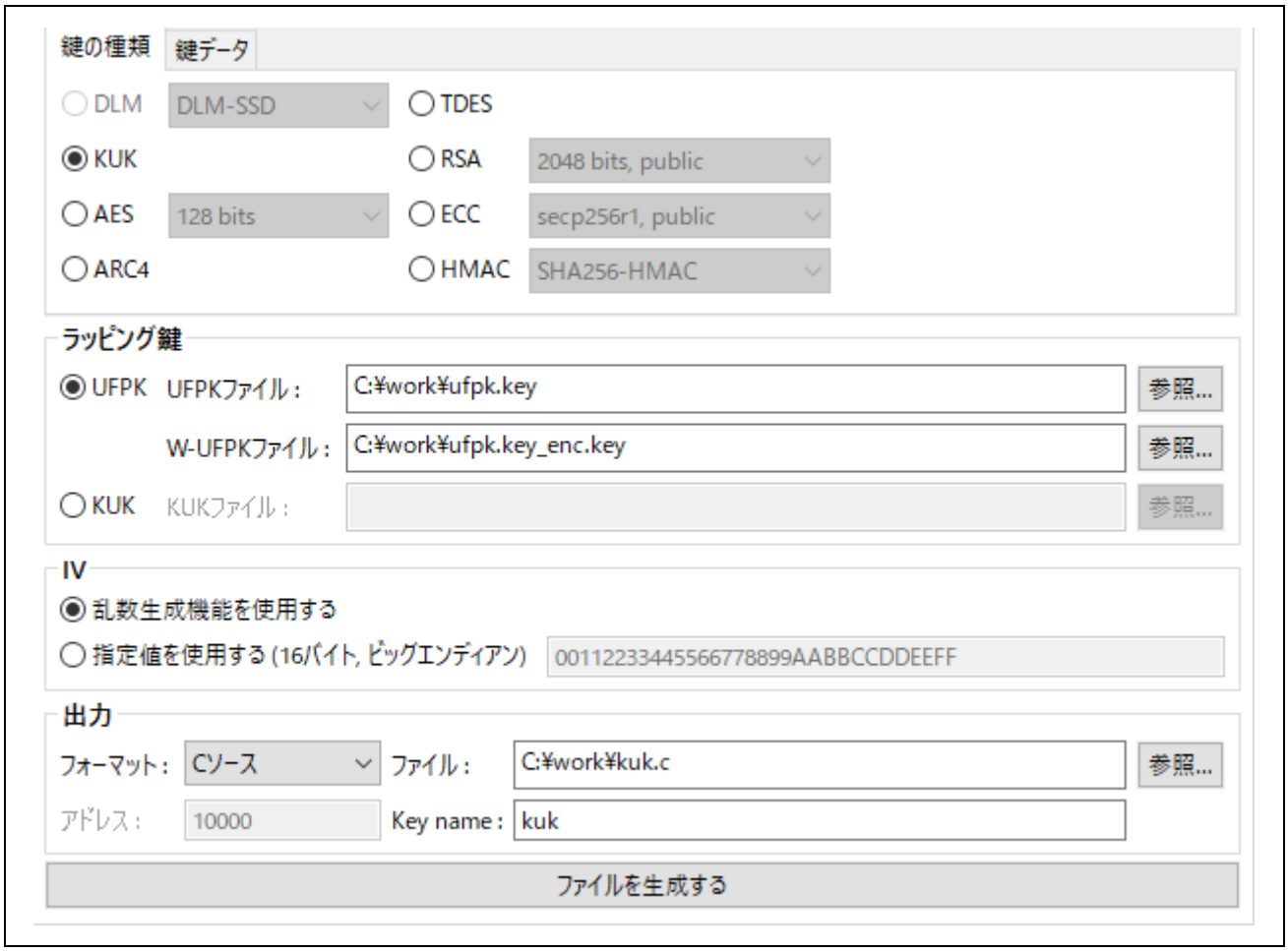

図 5-18 [**鍵のラッピング**] – [**鍵の種類**]タブ KUK ファイルの C ソースファイル出力設定例

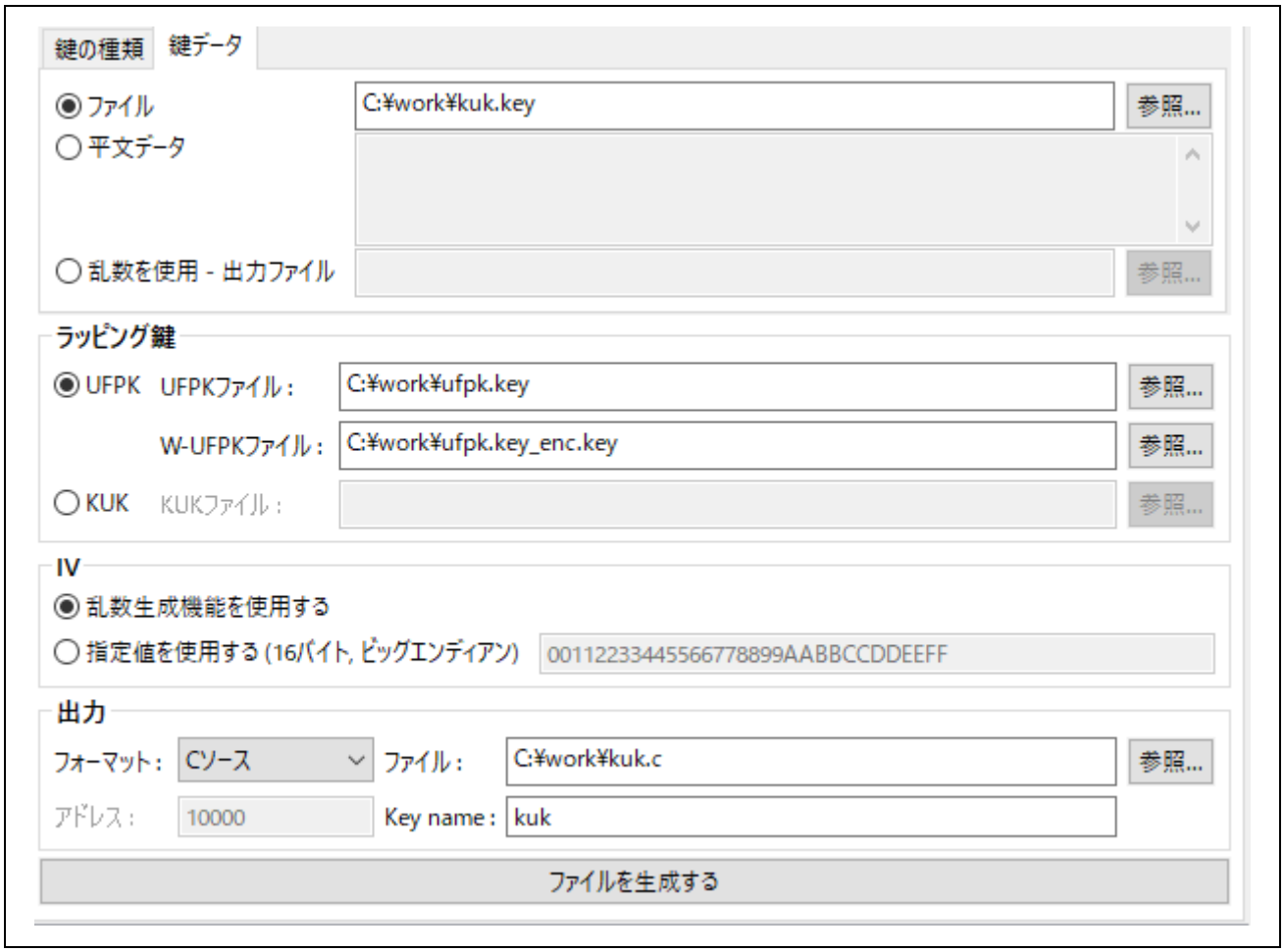

図 5-19 [**鍵のラッピング**] – [**鍵の種類**]タブ KUK ファイルを RFP ファイル出力設定例

ここで出力した C ソースファイルのデータを利用し、お客様のプログラム内で、R_TSIP_GenerateUpdateKeyRingKeyIndex()を実行して、KUK を注入してください。

続いて、市場で鍵を更新する場合の、KUK を使ったラッピング方法を示します。

6. C ソースファイルの RSA 2048 公開鍵ファイルの生成

[**鍵のラッピング**]タブを使用して RSA 2048 公開鍵ファイルを C ソースファイルとして生成します。[**鍵の種類**]タブで”RSA”と”2048 bits public”を選択し、[**鍵データ**] タブで RSA 2048 公開鍵データを入力します。

“**鍵のラッピング**”に 4 の手順で作成した KUK ファイルを設定します。簡略化のため、この例では IV に”**乱数生成機能を使用する**”を選択します。”**出力**”のフォーマットに”**C ソース**”を選択し、拡張子*.c のファイル名を指定します。

図 5-20 **[鍵のラッピング] – [鍵の種類]**タブ RSA 2048 公開鍵 C ソースファイル生成例

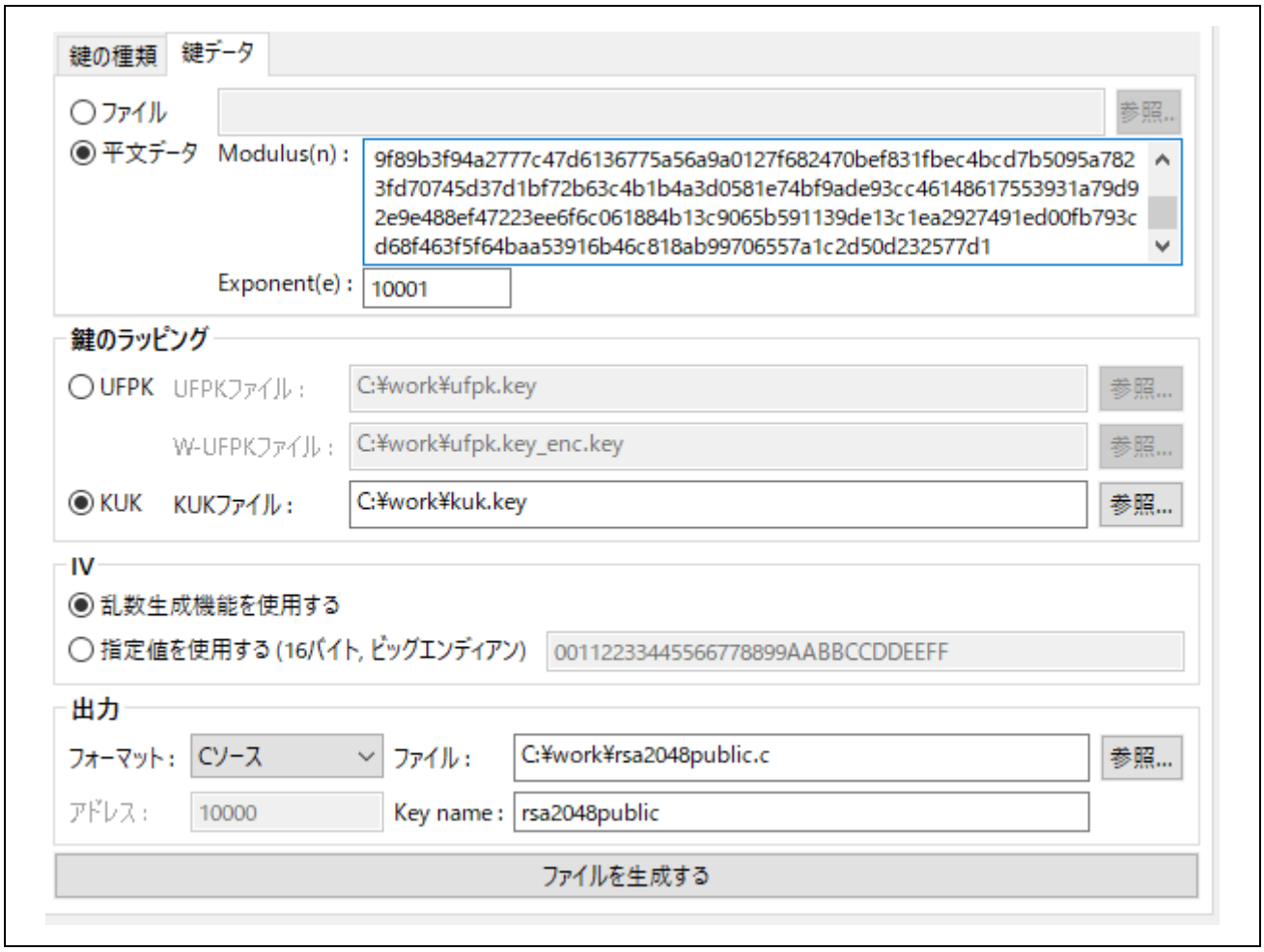

図 5-21 **[鍵のラッピング] – [鍵データ]**タブ RSA 2048 公開鍵 C ソースファイル生成例

正常終了すると以下のように出力されます。

図 5-22 [**鍵のラッピング**]タブ　実行結果

生成された C ソースファイルを更新するデータのプロジェクトに組み込んで更新するデータを作成することで、市場で鍵の更新を行うことが可能になります。

6. サンプルプログラム

6.1 デモプロジェクト動作確認方法

AES 暗号プロジェクトおよび TLS 連携機能プロジェクトを除くデモプロジェクトは「表 6-1　各 MCU で使用するデモプロジェクト」に示す MCU 上で動作します。MCU が搭載されたボードと PC は USB で接続します。PC では Tera Term を使用します。

表 6-1　各 MCU で使用するデモプロジェクト

MCU	デモプロジェクト
RX231	rx231_rsk_tsip_sample
RX26T	rx26t_mcb_tsip_sample
RX65N	rx65n_2mb_rsk_tsip_sample
RX66T	rx66t_rsk_tsip_sample
RX671	rx671_rsk_tsip_sample
RX72M	rx72m_rsk_tsip_sample
RX72N	rx72n_rsk_tsip_sample
RX72T	rx72t_rsk_tsip_sample

デュアルバンク機能搭載デバイス用のプロジェクトは、コードフラッシュをリニアモードで使用します。

デモプロジェクトは Little Endian で動作確認をしています。

本章のデモプロジェクトの実行結果などは RSK RX231 の動作結果で説明をしています。また、ターミナルソフトとしては Tera Term を使用しています。

※注意事項

TSIP 搭載デバイスのデモプロジェクトのリンクサイズは 128K バイトより大きくなります。使用するには製品版の CC-RX コンパイラを購入してください。

RX66N は RSK が販売されていないため、デモプロジェクトを用意していません。RX66N 上で動作確認を行うには、アプリケーションノート「RX スマート・コンフィグレータ ユーザーガイド: e^2 studio 編 (R20AN0451JSxxxx)」の 4 章を参照して、動作デバイスを RX66N に変換してください。
RX72N(R5F572NNHDBD)と RX66N(R5F566NNHDBD)はピン配置に互換があるため、デバイスの変換のみで同様に動作させることが可能です。TSIP ドライバも同じターゲットフォルダを利用するため、同一の UFPK で動作可能です。

6.1.1　デモプロジェクトのセットアップ

ボードと PC の接続は以下の通りです。

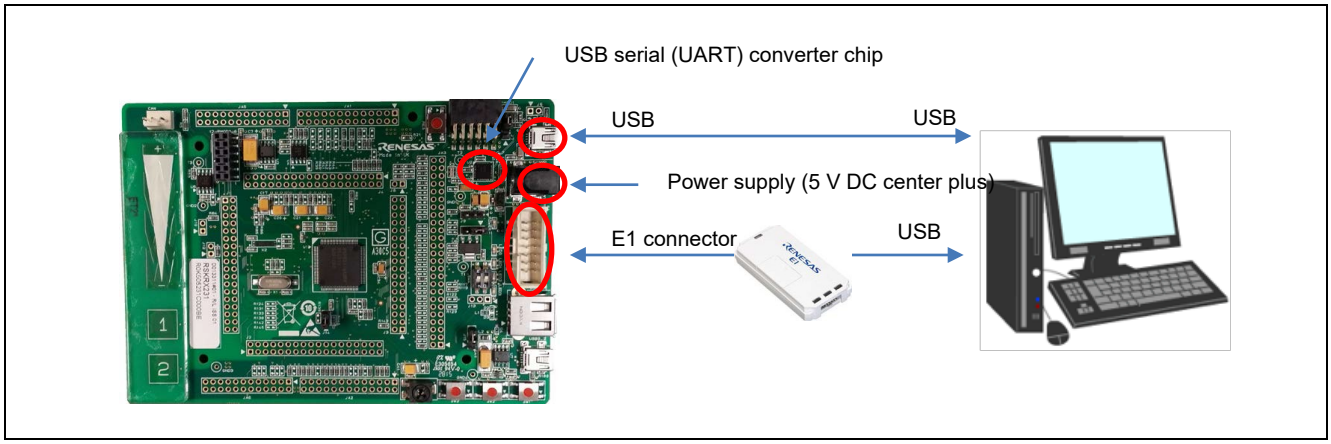

図 6-1 RSK RX231 と PC の接続

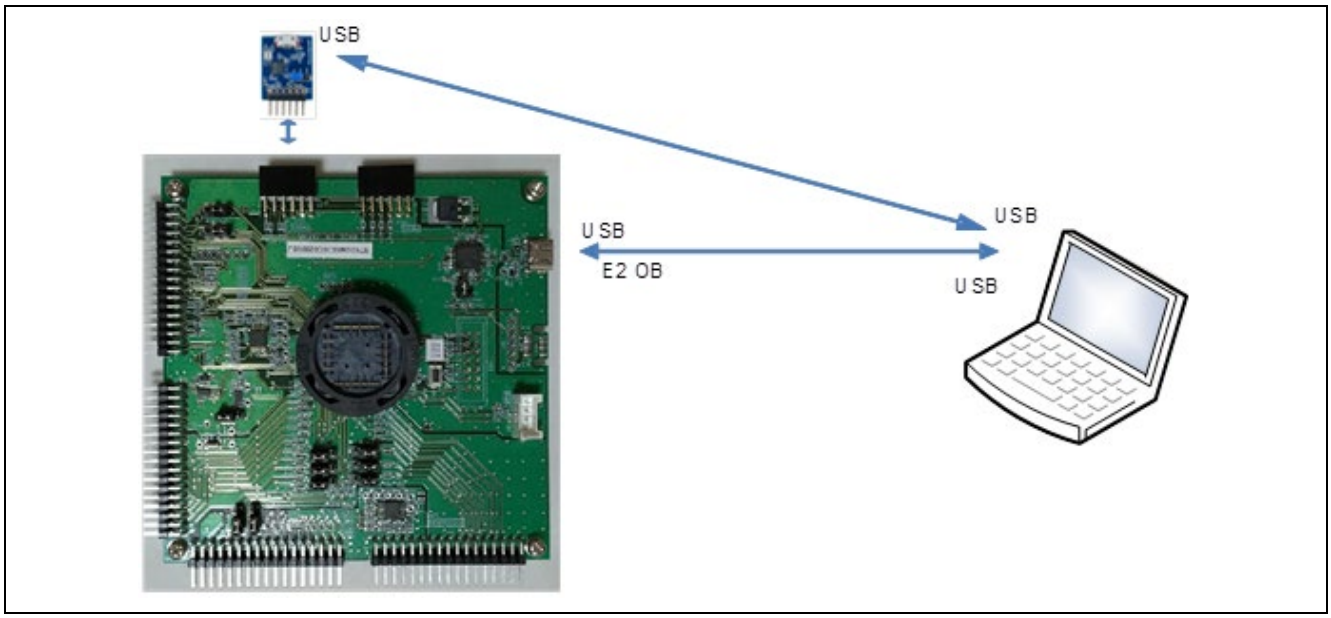

図 6-2 MCB RX26T と PC の接続

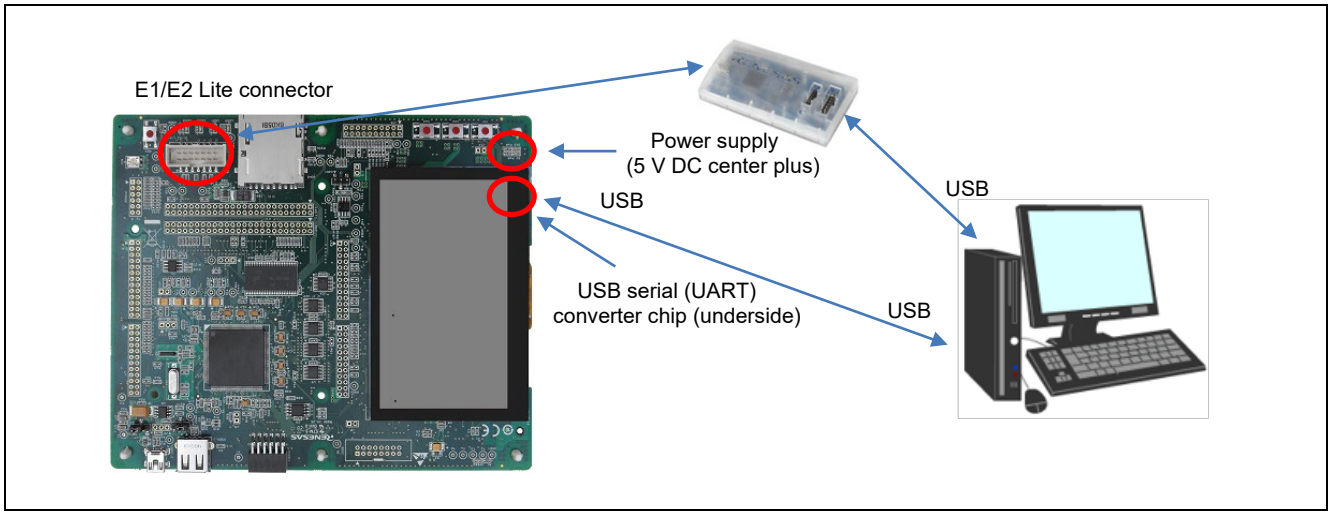

図 6-3 RSK RX65N-2MB と PC の接続

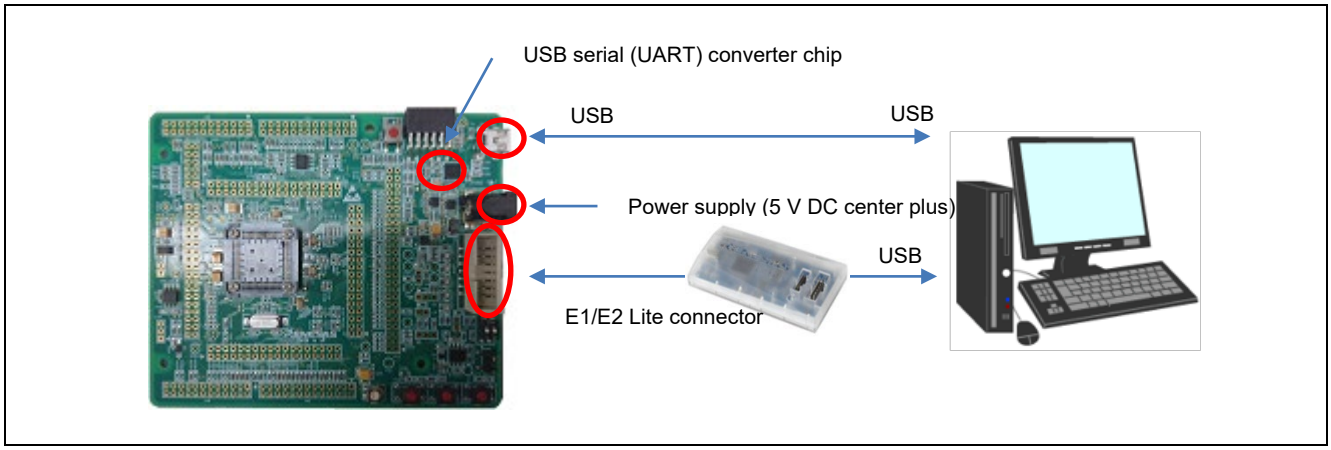

図 6-4 RSK RX66T と PC の接続

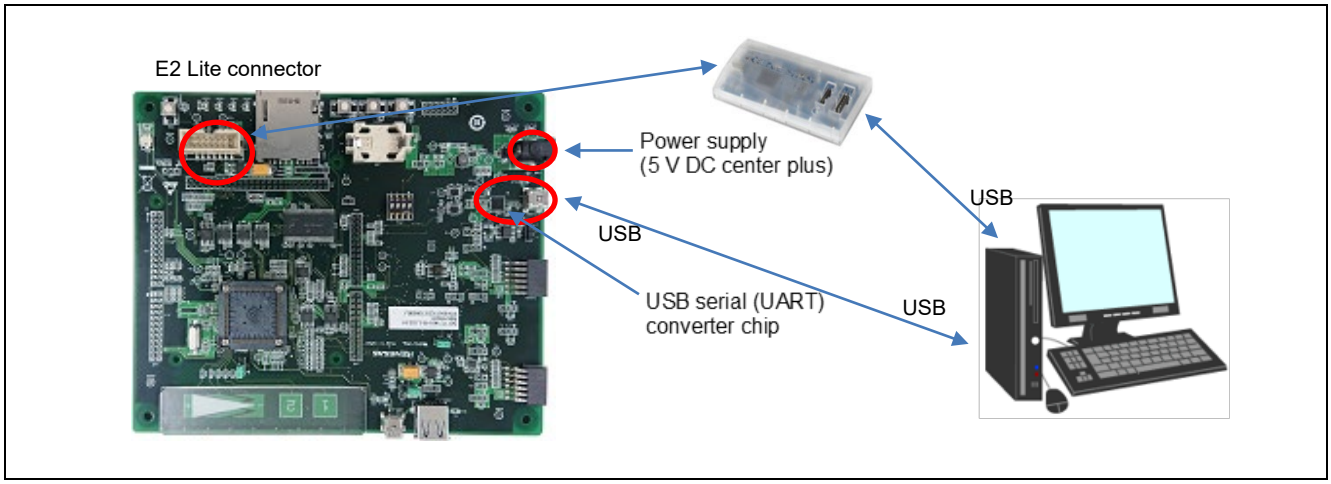

図 6-5 RSK RX671 と PC の接続

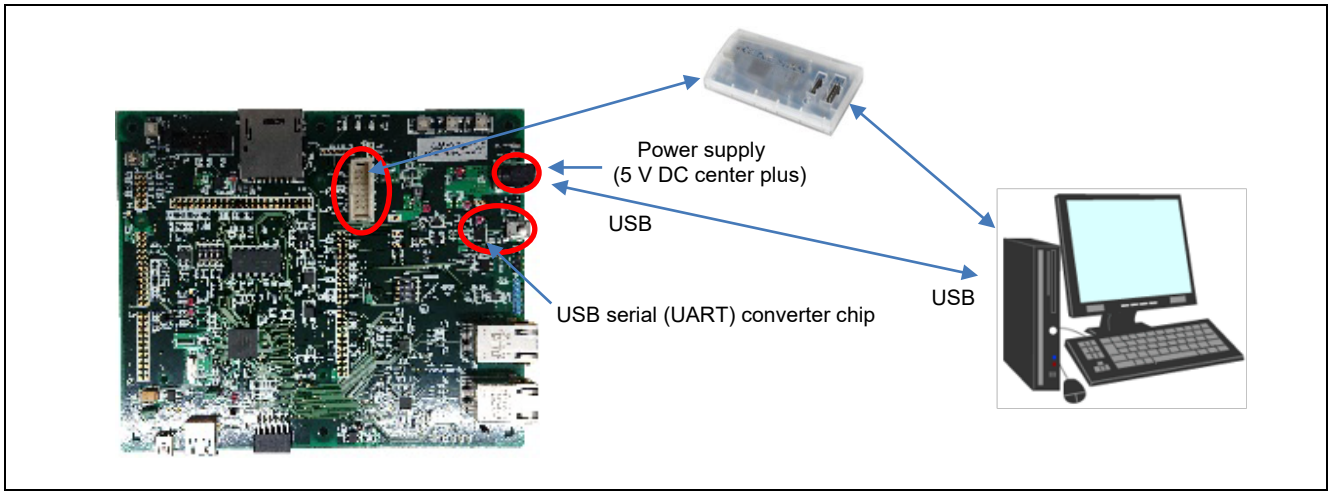

図 6-6 RSK RX72M と PC の接続

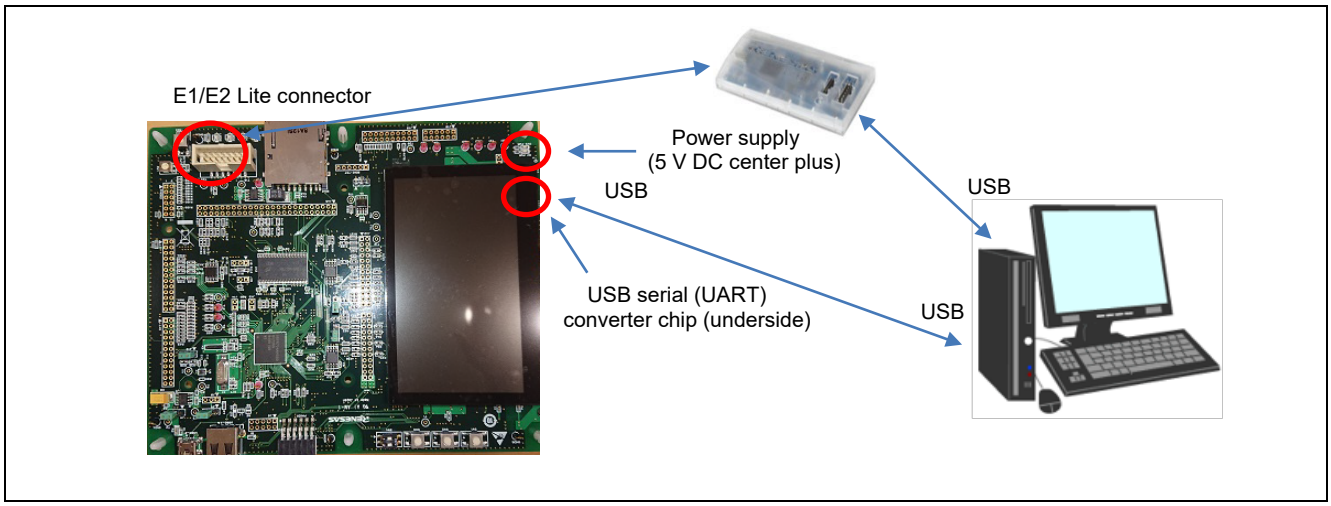

図 6-7 RSK RX72N と PC の接続

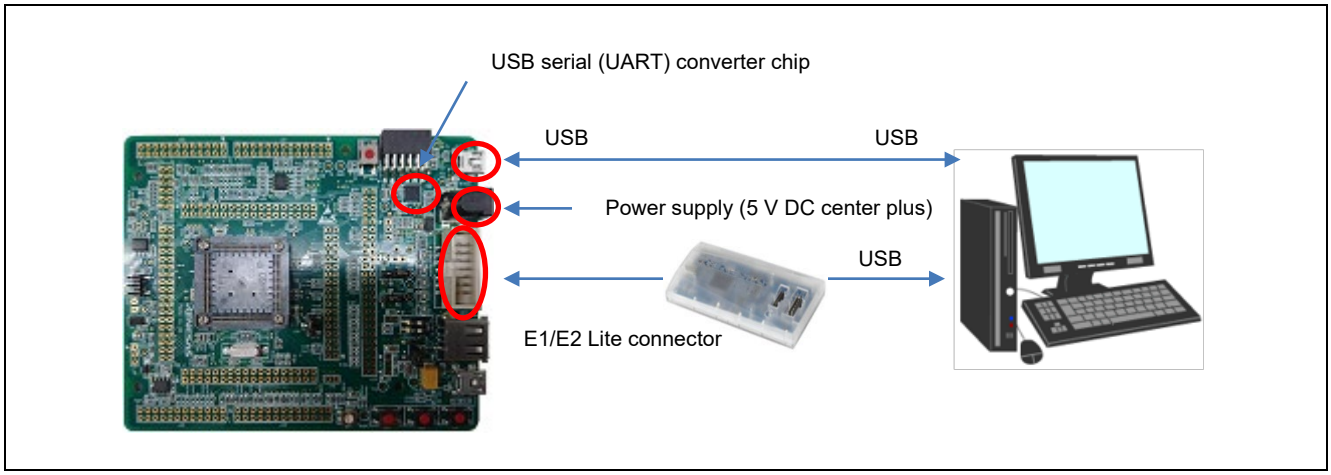

図 6-8 RSK RX72T と PC の接続

Tera Term のシリアル設定は以下の通りです。

・スピード：115200 bps

・データ：8 ビット

・パリティ：なし

・ストップビット：1 ビット

・改行コード（送信）：CR

6.1.2　デモプロジェクトの概要

デモプロジェクトで使用する鍵データ構造体は以下の通りです。

表 6-2　デモプロジェクトの鍵データ構造体

名称	型	説明
st_key_block_data_t	-	C_FIRMWARE_UPDATE_CONTROL_BLOCK, C_FIRMWARE_UPDATE_CONTROL_BLOCK_MIRROR に置かれる鍵データ構造体
firmware_update_control_data	-	firmware update 実施時の mac 情報格納構造体
user_program_max_cnt	uint32_t	firmware update 時の firmware バイトサイズ
lifecycle_state	uint32_t	鍵データとユーザプログラム*1 のステータス LIFECYCLE_STATE_BLANK、 LIFECYCLE_STATE_UPDATING、 LIFECYCLE_STATE_ON_THE_MARKET の 3 状態があります。
		1.LIFECYCLE_STATE_BLANK:
		鍵：注入未完了
		ユーザプログラム：注入未完了
		2.LIFECYCLE_STATE_UPDATING:
		鍵：注入完、鍵データ HASH 未設定
		ユーザプログラム：UPDATE_TEMPORARY_AREA: 更新プログラム書き込み完了 UPDATE_FIRMWARE_AREA: プログラムは更新前のプログラム
		3.LIFECYCLE_STATE_ON_THE_MARKET:
		鍵：注入完、鍵データ HASH 設定完了 HASH の検証正常終了
		ユーザプログラム：UPDATE_TEMPORARY_AREA: 更新プログラム書き込み完了 UPDATE_FIRMWARE_AREA: UPDATE_TEMPORARY_AREA からコピー完了。MAC 検証完了。 ユーザプログラム動作可能
program_mac0[]	uint32_t	UPDATE_FIRMWARE_AREA の MAC 値
program_mac1[]	uint32_t	UPDATE_TEMPORARY_AREA の MAC 値
key_data	-	鍵データ格納構造体
user_aes128_key_index	tsip_aes_key_index_t	AES 128 用 Key Index
user_aes256_key_index	tsip_aes_key_index_t	AES 256 用 Key Index
user_tdes_key_index *2	tsip_tdes_key_index_t	TDES 用 Key Index
user_arc4_key_index *2	tsip_arc4_key_index_t	ARC4 用 Key Index
user_sha1hmac_key_index *2	tsip_hmac_sha_key_index_t	SHA-1 HMAC 用 Key Index
user_sha256hmac_key_index *2	tsip_hmac_sha_key_index_t	SHA-256 HMAC 用 Key Index
user_rsa1024_ne_key_index *2	tsip_rsa1024_public_key_index_t	RSA 1024 用公開鍵 Key Index
user_rsa1024_nd_key_index *2	tsip_rsa1024_private_key_index_t	RSA 1024 用秘密鍵 Key Index
user_rsa2048_ne_key_index *2	tsip_rsa2048_public_key_index_t	RSA 2048 用公開鍵 Key Index
user_rsa2048_nd_key_index *2	tsip_rsa2048_private_key_index_t	RSA 2048 用秘密鍵 Key Index
user_ecc192_public_key_index *2	tsip_ecc_public_key_index_t	ECC 192 用公開鍵 Key Index
user_ecc192_private_key_index *2	tsip_ecc_private_key_index_t	ECC 192 用秘密鍵 Key Index
user_ecc224_public_key_index *2	tsip_ecc_public_key_index_t	ECC 224 用公開鍵 Key Index
user_ecc224_private_key_index *2	tsip_ecc_private_key_index_t	ECC 224 用秘密鍵 Key Index
user_ecc256_public_key_index *2	tsip_ecc_public_key_index_t	ECC 256 用公開鍵 Key Index
user_ecc256_private_key_index *2	tsip_ecc_private_key_index_t	ECC 256 用秘密鍵 Key Index
hash_sha1[]	uint8_t	firmware_update_control_data と key_data の SHA1 HASH 値

*1 デモプロジェクトでは鍵データのみのステータスになります。
ユーザプログラムのステータスはセキュアブートプロジェクト(6.2 セキュアブート・ファームウェアアップデートのデモプロジェクト動作確認方法)で使用します。

*2 TSIP-Lite 搭載デバイスの場合、TDES、ARC4、HMAC、RSA、ECC の鍵情報は使用できません。また、TSIP-Lite 搭載デバイスのデモプロジェクトに添付されている key_data.c,key_data.h にもデータは存在しません。RX671 のデモプロジェクトでは、key_data.c,key_data.h に ARC4 および RSA2048 のデータは存在しません。

デモプロジェクトの鍵情報構造体はデータフラッシュ上に、main データと mirror データを用意しています。(下図　main データ：C_FIRMWARE_UPDATE_CONTROL_BLOCK、

mirror データ:C_FIRMWARE_UPDATE_CONTROL_BLOCK_MIRROR)

これにより、鍵情報を更新時の電源遮断などによる書き込み失敗時の鍵情報消失を防ぎます。

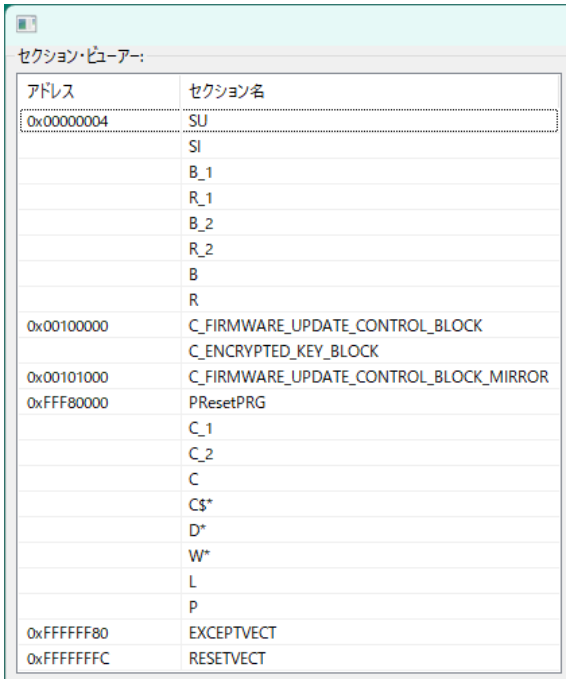

図 6-9　RX231 デモプロジェクトのセクション

key_data.c および key_data.h において、(デモプロジェクトフォルダ)\src\genkey にある Encrypted Key 定義ファイルをインクルードしています。生成方法は「5.3.1 鍵の注入手順」を参照してください。

デモプロジェクトでは以下のコマンドを実装しています。

表 6-3　デモプロジェクト　コマンド一覧

コマンド	動作
display	Wrapped Key を表示します。
encdemo [Arg1]	Arg1 の値を AES 128bit で暗号化します。
function	AES128/256-ECB, CBC, CTR, GCM, CCM 暗号化、復号 AES128/256-CMAC, SHA-HMAC 生成、検証 TDES 暗号化、復号 ARC4 暗号化、復号 (RSA1024, RSA2048 のみ)RSAES-PKCS1_v1_5 暗号化、復号 (RSA1024, RSA2048 のみ)RSASSA-PKCS1_v1_5 署名生成、検証 P-384 を除く ECDSA 署名生成、検証のテストが動作します。*1
random	R_TSIP_GenerateRandomNumber()によるランダム値の生成をします。

*1:TSIP-Lite 搭載デバイスでは SHA-HMAC 生成、検証、TDES 暗号化、復号、ARC4 暗号化、復号、RSAES-PKCS1_v1_5 暗号化、復号、RSASSA-PKCS1_v1_5 署名生成、検証、ECDSA 署名生成、検証は実装できません。また RX671 のデモプロジェクトでは、ARC4 および RSA2048 は非対応です。

デモプロジェクトでは、IV を”55aa55aa55aa55aa55aa55aa55aa55aa"、および以下の値を鍵のデフォルト値として使用しています。

表 6-4　デモプロジェクト　デフォルトの鍵の値

key type	value
UFPK	22222222222222222222222222222221111111111111111111111111111111
AES-128bit	11111111222222223333333344444444
AES-256bit	ffffffffeeeeeeeeddddddddccccccccbbbbbbbbaaaaaaaa0000000099999999
3 TDES *2	01010101010101010202020202020202020404040404040404
ARC4-2048 bit *2	29041972fb42ba5fc7127712f13829c929041972fb42ba5fc7127712f13829c9 29041972fb42ba5fc7127712f13829c929041972fb42ba5fc7127712f13829c9 29041972fb42ba5fc7127712f13829c929041972fb42ba5fc7127712f13829c9 29041972fb42ba5fc7127712f13829c929041972fb42ba5fc7127712f13829c9 29041972fb42ba5fc7127712f13829c929041972fb42ba5fc7127712f13829c9 29041972fb42ba5fc7127712f13829c929041972fb42ba5fc7127712f13829c9 29041972fb42ba5fc7127712f13829c929041972fb42ba5fc7127712f13829c9 29041972fb42ba5fc7127712f13829c929041972fb42ba5fc7127712f13829c9
SHA-1 HMAC *2	000102030405060708090a0b0c0d0e0f10111213
SHA-256 HMAC *2	000102030405060708090a0b0c0d0e0f101112131415161718191a1b1c1d1e1f
RSA-1024bit public *1*2	ccd6cb86f59ffea97c278b7cf395fa56f3709a958bce1e6e3ae196b471f4517f 64d32d81d10bcea5b55b9ea659c3b9db1854a696b801a8e72439265e85bc6138 cf874f45fe22a2477fe8337671357db67180b1ac9a0e84c098dbcdca8a3e6a72 dfcee7e6907ede2d53aa726e8e97ea26fcba3bbfa7bf7bfbbb70f1ec1d153125 00010001
RSA-1024bit private *1*2	ccd6cb86f59ffea97c278b7cf395fa56f3709a958bce1e6e3ae196b471f4517f 64d32d81d10bcea5b55b9ea659c3b9db1854a696b801a8e72439265e85bc6138 cf874f45fe22a2477fe8337671357db67180b1ac9a0e84c098dbcdca8a3e6a72 dfcee7e6907ede2d53aa726e8e97ea26fcba3bbfa7bf7bfbbb70f1ec1d153125 096ed2cc923f1df11c208e11e0fdc51b7ff66d87f97a32788d099a7110d65972 6e68332e493c2bf6019608864c97f0d52017b5dc36f90c982858e16574ef29e2 e8b6f4386bdbb1beae60390aaac0160ce787f98ba34c83d6612badbc99d17666 04e72b394d1991fc4ded87e11cf165caae0cc652771b8395ed5b9a99a9fa5781
RSA-2048bit public *1*2	bad47a84c1782e4dbdd913f2a261fc8b65838412c6e45a2068ed6d7f16e9cdf4 462b39119563cafb74b9cbf25cfd544bdae23bff0ebe7f6441042b7e109b9a8a faa056821ef8efaab219d21d6763484785622d918d395a2a31f2ece8385a8131 e5ff143314a82e21afd713bae817cc0ee3514d4839007ccb55d68409c97a18ab 62fa6f9f89b3f94a2777c47d6136775a56a9a0127f682470bef831fbec4bcd7b 5095a7823fd70745d37d1bf72b63c4b1b4a3d0581e74bf9ade93cc4614861755 3931a79d92e9e488ef47223ee6f6c061884b13c9065b591139de13c1ea292749 1ed00fb793cd68f463f5f64baa53916b46c818ab99706557a1c2d50d232577d1 00010001
RSA-2048bit private *1*2	bad47a84c1782e4dbdd913f2a261fc8b65838412c6e45a2068ed6d7f16e9cdf4 462b39119563cafb74b9cbf25cfd544bdae23bff0ebe7f6441042b7e109b9a8a faa056821ef8efaab219d21d6763484785622d918d395a2a31f2ece8385a8131 e5ff143314a82e21afd713bae817cc0ee3514d4839007ccb55d68409c97a18ab 62fa6f9f89b3f94a2777c47d6136775a56a9a0127f682470bef831fbec4bcd7b 5095a7823fd70745d37d1bf72b63c4b1b4a3d0581e74bf9ade93cc4614861755 3931a79d92e9e488ef47223ee6f6c061884b13c9065b591139de13c1ea292749 1ed00fb793cd68f463f5f64baa53916b46c818ab99706557a1c2d50d232577d1 40d60f24b61d76783d3bb1dc00b55f96a2a686f59b3750fdb15c40251c370c65 cada222673811bc6b305ed7c90ffcb3abdddc8336612ff13b42a75cb7c88fb93 6291b523d80acce5a0842c724ed85a1393faf3d470bda8083fa84dc5f3149984

	4f0c7c1e93fb1f734a5a29fb31a35c8a0822455f1c850a49e8629714ec6a2657 efe75ec1ca6e62f9a3756c9b20b4855bdc9a3ab58c43d8af85b837a7fd15aa11 49c119cfe960c05a9d4cea69c9fb6a897145674882bf57241d77c054dc4c94e8 349d376296137eb421686159cb878d15d171eda8692834afc871988f203fc822 c5dcee7f6c48df663ea3dc755e7dc06aebd41d05f1ca2891e2679783244d068f
ECC-192bit public *2 (上段:Qx、下段:Qy)	fba2aac647884b504eb8cd5a0a1287babcc62163f606a9a2 dae6d4cc05ef4f27d79ee38b71c9c8ef4865d98850d84aa5
ECC-192bit private *2	7891686032fd8057f636b44b1f47cce564d2509923a7465b
ECC-224bit public *2 (上段:Qx、下段:Qy)	4c741e4d20103670b7161ae72271082155838418084335338ac38fa4 db7919151ac28587b72bad7ab180ec8e95ab9e2c8d81d9b9d7e2e383
ECC-224bit private *2	888fc992893bdd8aa02c80768832605d020b81ae0b25474154ec89aa
ECC-256bit public *2 (上段:Qx、下段:Qy)	1ccbe91c075fc7f4f033bfa248db8fccd3565de94bbfb12f3c59ff46c271bf83 ce4014c68811f9a21a1fdb2c0e6113e06db7ca93b7404e78dc7ccd5ca89a4ca9
ECC-256bit private *2	519b423d715f8b581f4fa8ee59f4771a5b44c8130b4e3eacca54a56dda72b464

*1 : RSA 1024bit の鍵ペアは以下の値です。

Modulus n : ccd6cb86f59ffea97c278b7cf395fa56f3709a958bce1e6e3ae196b471f4517f
64d32d81d10bcea5b55b9ea659c3b9db1854a696b801a8e72439265e85bc6138
cf874f45fe22a2477fe8337671357db67180b1ac9a0e84c098dbcdca8a3e6a72
dfcee7e6907ede2d53aa726e8e97ea26fcba3bbfa7bf7bfbbb70f1ec1d153125

Exponent e : 00010001

Decryption Exponent d : 096ed2cc923f1df11c208e11e0fdc51b7ff66d87f97a32788d099a7110d65972
6e68332e493c2bf6019608864c97f0d52017b5dc36f90c982858e16574ef29e2
e8b6f4386bdbb1beae60390aaac0160ce787f98ba34c83d6612badbc99d17666
04e72b394d1991fc4ded87e11cf165caae0cc652771b8395ed5b9a99a9fa5781

RSA 2048bit の鍵ペアは以下の値です。

Modulus n : bad47a84c1782e4dbdd913f2a261fc8b65838412c6e45a2068ed6d7f16e9cdf4
462b39119563cafb74b9cbf25cfd544bdae23bff0ebe7f6441042b7e109b9a8a
faa056821ef8efaab219d21d6763484785622d918d395a2a31f2ece8385a8131
e5ff143314a82e21afd713bae817cc0ee3514d4839007ccb55d68409c97a18ab
62fa6f9f89b3f94a2777c47d6136775a56a9a0127f682470bef831fbec4bcd7b
5095a7823fd70745d37d1bf72b63c4b1b4a3d0581e74bf9ade93cc4614861755
3931a79d92e9e488ef47223ee6f6c061884b13c9065b591139de13c1ea292749
1ed00fb793cd68f463f5f64baa53916b46c818ab99706557a1c2d50d232577d1

Exponent e : 00010001

Decryption Exponent d : 40d60f24b61d76783d3bb1dc00b55f96a2a686f59b3750fdb15c40251c370c65
cada222673811bc6b305ed7c90ffcb3abdddc8336612ff13b42a75cb7c88fb93
6291b523d80acce5a0842c724ed85a1393faf3d470bda8083fa84dc5f3149984
4f0c7c1e93fb1f734a5a29fb31a35c8a0822455f1c850a49e8629714ec6a2657
efe75ec1ca6e62f9a3756c9b20b4855bdc9a3ab58c43d8af85b837a7fd15aa11
49c119cfe960c05a9d4cea69c9fb6a897145674882bf57241d77c054dc4c94e8
349d376296137eb421686159cb878d15d171eda8692834afc871988f203fc822
c5dcee7f6c48df663ea3dc755e7dc06aebd41d05f1ca2891e2679783244d068f

*2 :TSIP-Lite 搭載デバイスでは、TDES、ARC4、HMAC、RSA、ECC の鍵情報は使用できません。また、TSIP-Lite 搭載デバイスのデモプロジェクトに添付されている key_data.c,key_data.h にはデータが存在しません。RX671 のデモプロジェクトでは、key_data.c,key_data.h に ARC4 および RSA2048 のデータは存在しません。

MCU にプログラムをダウンロード後、初回起動時は鍵の注入を行います。鍵の注入はユーザ鍵から TSIP で使用する Wrapped Key に変換し、内蔵データフラッシュに書き込みます。起動時の鍵の注入フローチャートを図 6-10 に示します。鍵の注入が正常に行われた場合、コマンド・レスポンスが実行できるようになります。その際の表示については、6.1.3 に示しています。

デモプロジェクトでは、UFPK として以下の値を使用しています。

UFPK = “22222222222222222222222222222222111111111111111111111111111111111”

デモプロジェクトの key フォルダに Security Key Management Tool で使用可能な UFPK ファイル sample.key と W-UFPK ファイル sample.key_enc.key を用意しています。

MCU で新たにプログラムをダウンロードせずに起動した場合は、鍵情報のチェックだけを実施します。その際の表示については、6.1.3 に示しています。

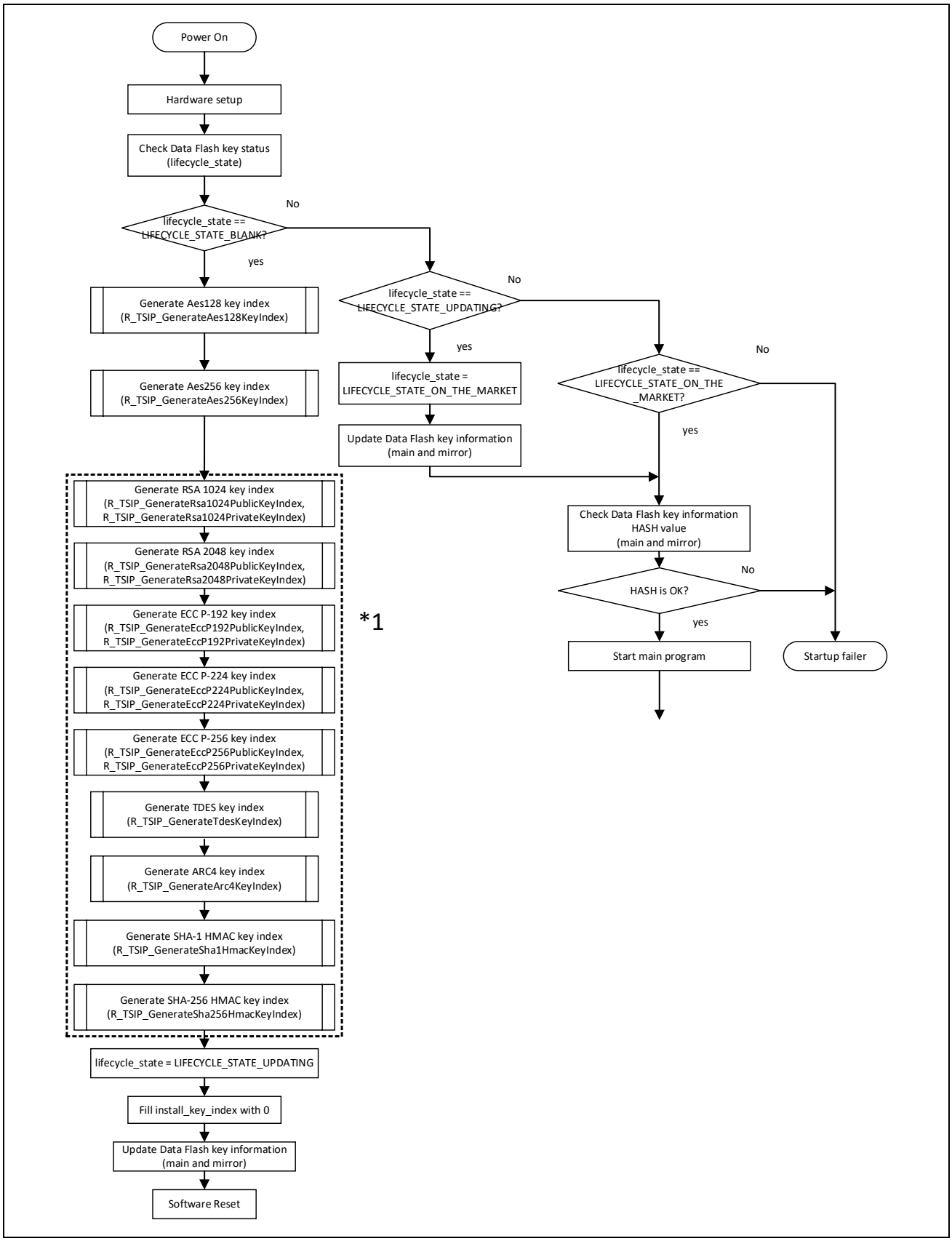

図 6-10　起動時の鍵の注入 フローチャート

*1: TSIP-Lite 搭載デバイスのデモプロジェクトでは実施しません。RX671 のデモプロジェクトでは ARC4 および RSA2048 を実施しません。

6.1.3　デモプロジェクトの実行例

MCU にプログラムをダウンロードし初回起動時の鍵インストールが正常に行われた場合の表示を図 6-11 に、MCU で新たにプログラムをダウンロードせずに起動した場合の表示を図 6-12 に示します。これらのように表示された後、表 6-3 のコマンドを使用することができるようになります。

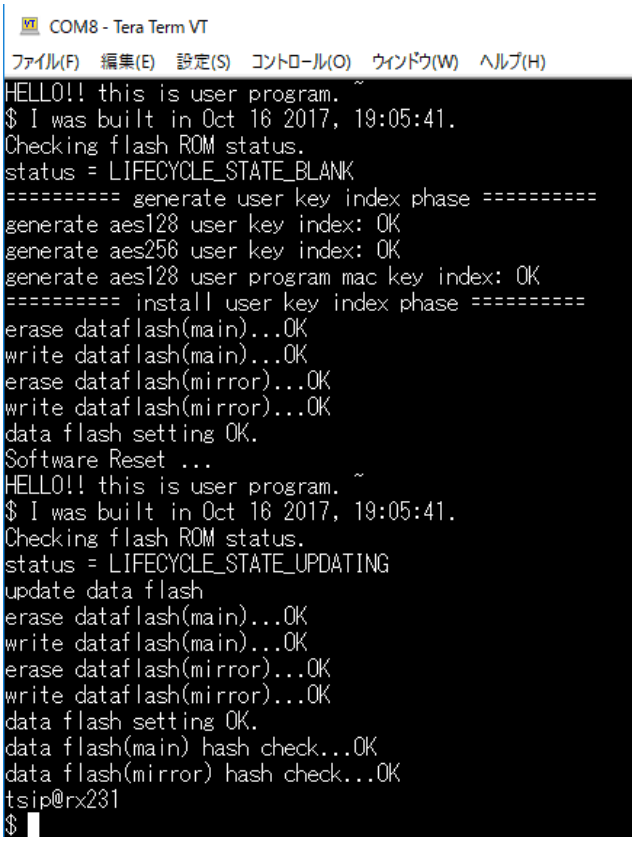

図 6-11　Tera Term 表示(RSK RX231 初回起動時)

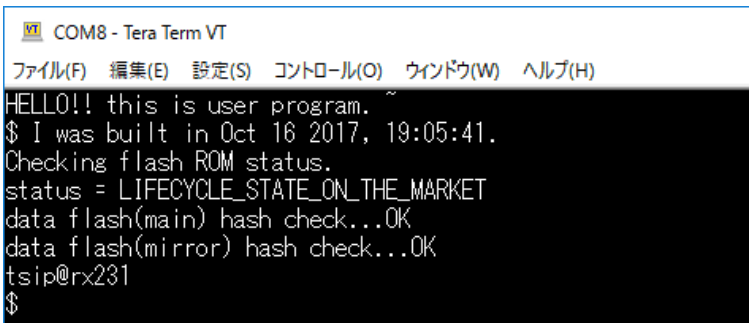

図 6-12　Tera Term 表示(RSK RX231 2 回目以降の起動時)

コマンドの使用例として、encdemo aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa コマンドで、AES128bit の Wrapped Key を使用し、encdemo コマンドの引数を図 6-13 に示します。暗号化に使用している鍵は Security Key Management Tool で入力したユーザ鍵(16 バイト(128bit))です。デフォルト状態の”0x11, 0x11, 0x11, 0x11, 0x22, 0x22, 0x22, 0x22, 0x33, 0x33, 0x33, 0x33, 0x44, 0x44, 0x44, 0x44”を使用した場合、結果は以下に示すスクリーンショットのようになります。

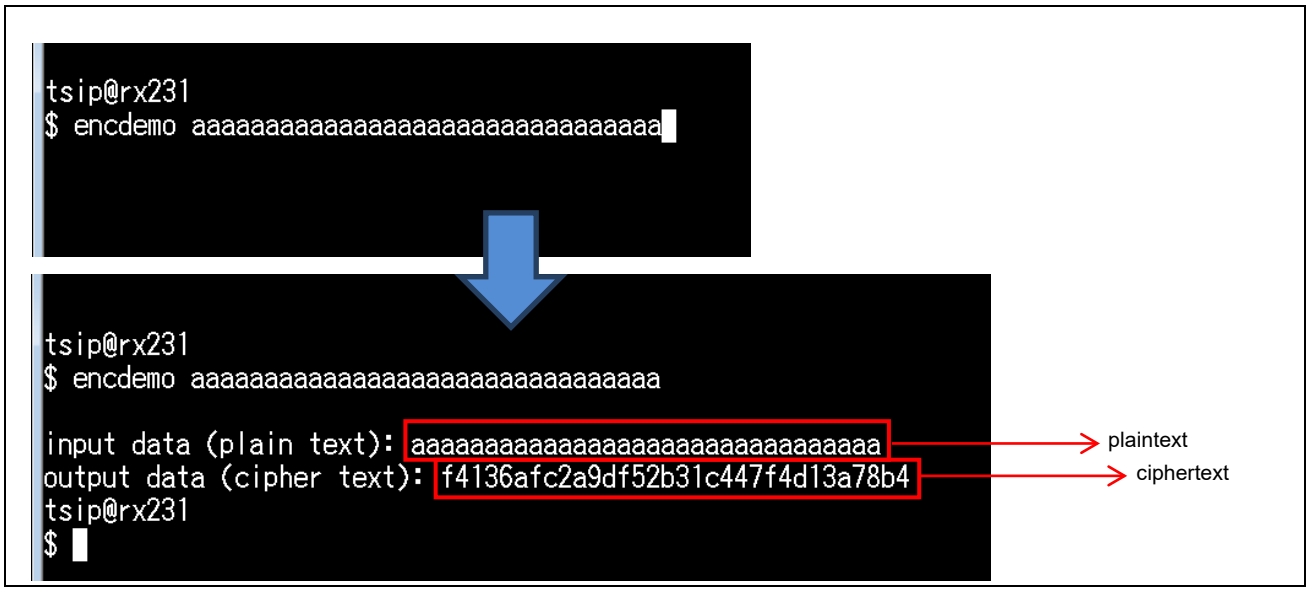

図 6-13　Tera Term 表示(encdemo コマンド)

■ポイント①

ユーザ鍵はデータフラッシュ領域(0x00100000)に格納されていますが、Security Key Management Tool で指定したユーザ鍵(デフォルト値"0x11, 0x11, 0x11, 0x11, 0x22, 0x22, 0x22, 0x22, 0x33, 0x33, 0x33, 0x33, 0x44, 0x44, 0x44, 0x44")が出現しないことを確認してください。

これはソフトウェアをリバースエンジニアリングされた際に、ユーザ鍵が流出しないことを意味します。

■ポイント②

Wrapped Key は HUK に紐付いています。あるチップで生成した Wrapped Key を他のチップにコピーして使用することはできません。TSIP ドライバが Wrapped Key を HUK と照合して検査するためです。

これはソフトウェアのデッドコピーを防ぐことを意味します。

■ポイント③

Wrapped Key は入力されるユーザ鍵が同じであっても毎回異なる値として出力されます。Wrapped Key を生成する際に乱数を用いているためです。

これは Wrapped Key からユーザ鍵を推測されるリスクを低減することを意味します。

6.2　セキュアブート・ファームウェアアップデートのデモプロジェクト動作確認方法

セキュアブート・ファームアップデートデモプロジェクトでは、4.2.16 ファームウェアアップデートの API を使用して、3.14.3 ユーザプログラムの暗号化で示した方法で暗号化されたユーザプログラムを、デバイス内で復号します。暗号化されたユーザプログラムを検証後、内蔵 Flash ROM に書き込ことを、ファームアップデートと呼びます。また、実行前にユーザプログラムを検証することをセキュアブートと呼びます。セキュアブート・ファームアップデートデモプロジェクトでは、ファームアップデートとセキュアブートを実現するためのサンプルプログラムです。

本デモプロジェクトの中にはセキュアブートプロジェクトと、ファームアップデートプロジェクトの 2 個のプロジェクトを用意しています。セキュアブートプロジェクトには、初期ファームをデバイスに書き込むためのファームアップデート動作も入っています。

セキュアブートプロジェクト　：rx*XXX*_*bbb*_tsip_secure_boot[*1]

ファームアップデートプロジェクト：rx*XXX*_*bbb*_tsip_user_program[*1]

本デモプロジェクトのファームアップデートは、初期プログラムもしくは更新するプログラムをデバイスに送信する IF として、UART(*2)もしくは USB(*3)を用意しています。また Security Key Management Tool は、工場出荷書き込みを想定し、セキュアブートと暗号化されたユーザプログラムを合わせた Motorola S format ファイルを生成(*4)することができます。(Factory Programming 機能)

本デモプロジェクトは、内蔵 Flash ROM が Dual Bank モードをサポートしているデバイスは、Dual Bank モードを使用したファームウェアアップデート動作を実現しています。

【*1】USB をサポートしていないデバイスのプロジェクトは UART のみのサポートになります。USB を未サポートのプロジェクトのプロジェクト名には、_sci とついています。また、*bbb* にはボード名が入ります。RSK の場合は、rsk, MCB の場合は mcb、Envision Kit は ek が入ります。

【*2】Dual Bank モードを使用できないデバイス(RX231,RX66T,RX72T)では、UART をフロー制御するため、PMOD USB-UART 変換基板を使用します。また、ボードに USB シリアル変換機能が搭載されていないボード(MCB RX26T)も PMOD USB-UART 変換基板を使用します。

【*3】RX66T、RX26T は USB I/F は未サポートのため、USB を使用したデモプロジェクトの用意はありません。

【*4】セキュアブートと暗号化されたユーザプログラムを合わせた Motorola S format ファイルは、Motorola S format ファイル自体を盗まれると、該当する RX TSIP 搭載デバイスで復号することが可能です。セキュアブートを含む暗号化されたユーザプログラムを書き込む場合は、セキュアな工場で書き込みを行ってください。

6.2.1　デモプロジェクトのセットアップ

本デモプロジェクトでは、表 6-5　デモプロジェクトで使用するボード、およびその設定 に示すボード上で動作します。USB を使用する場合、ボードを USB HOST として使うために、RSK 上のジャンパを USB-HOST 側に設定してください。工場出荷時は USB-FUNCTION 側となっています。また、Dual Bank モード未サポートのデバイスでは、ステータス表示ならびに更新プログラム送信のため、PMOD USB-UART 変換基板を使用します。

表 6-5　デモプロジェクトで使用するボード、およびその設定

ボード	ジャンパ設定	Flash Mode	その他
RSK RX231	J15 : 1-2 short	-	IF で UART 使用時は、RSK 上の 0Ω抵抗を付け替える必要があります。 PMOD USB-UART 変換基板が必要です。
MCB RX26T	(なし)	Dual Mode	PMOD USB-UART 変換基板が必要です。 UART のみサポート
RSK RX65N	J7 : 1-2 short J16 : 2-3 short	Dual Mode	-
RSK RX66T	(なし)	-	PMOD USB-UART 変換基板が必要です。 UART のみサポート
RSK RX671	J8 : 2-3 short J13 : 2-3 short	Dual Mode	-
RSK RX72M	J8 : 2-3 short J10 : 2-3 short	Dual Mode	-
RSK RX72N	J7 : 2-3 short J8 : 2-3 short	Dual Mode	-
RX72N Envision Kit	(なし)	Dual Mode	-
RSK RX72T	J13 : 1-2 short	-	PMOD USB-UART 変換基板が必要です。

実行結果の図は RX65N 実行時の結果を示しています。また、ターミナルソフトとして Tera Term を使用しています。

デモプロジェクトは Little Endian で動作確認をしています。また、RX Firmware Update FIT V.1.05 を一部変更して使用しています。変更箇所には /* modified from original fwup code */とコメントを入れてあります。

TSIP 搭載デバイスのデモプロジェクトのリンクサイズは 128K バイトより大きくなります。使用するには製品版の CC-RX コンパイラを購入してください。

ボードと PC の接続は以下の通りです。

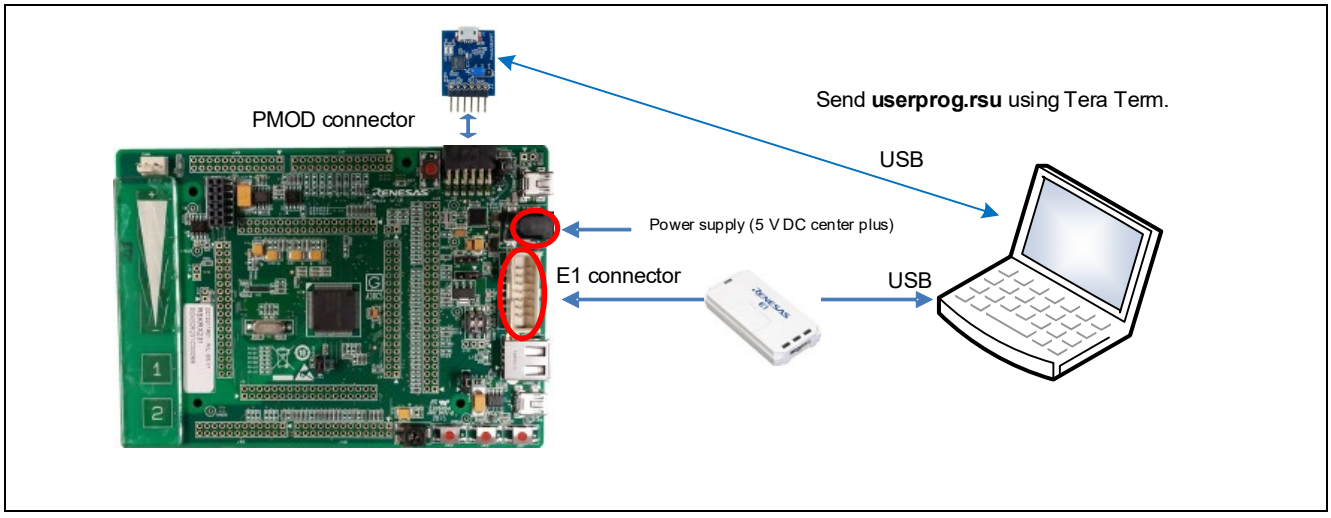

図 6-14 UART※1 を使用したファームアップデートを行うときの RSK RX231※2 と PC の接続

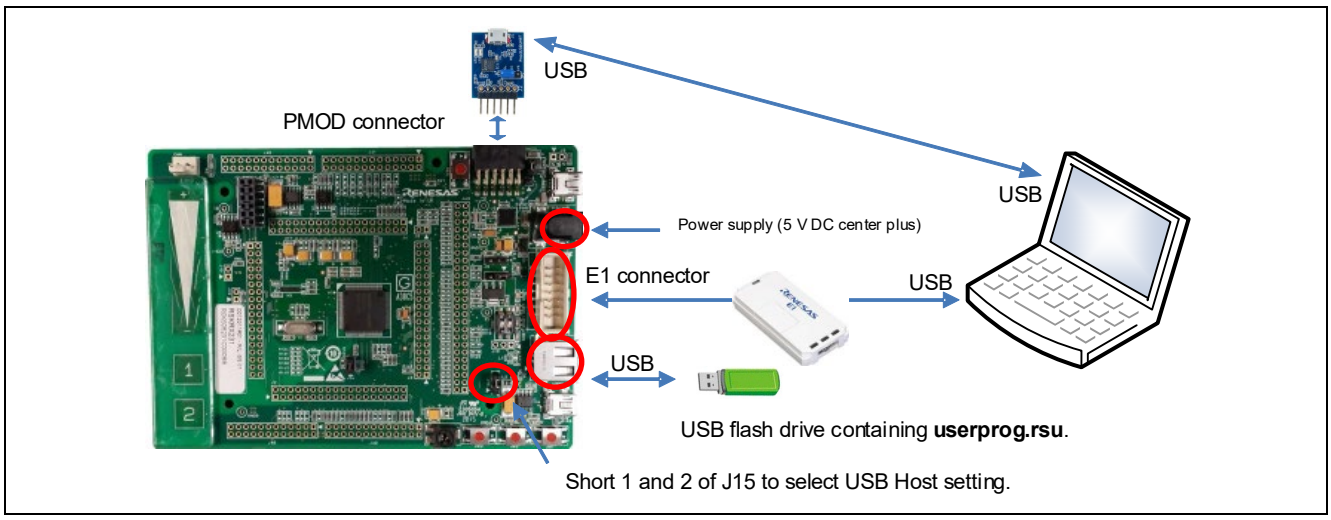

図 6-15 USB を使用したファームアップデートを行うときの RSK RX231 と PC の接続

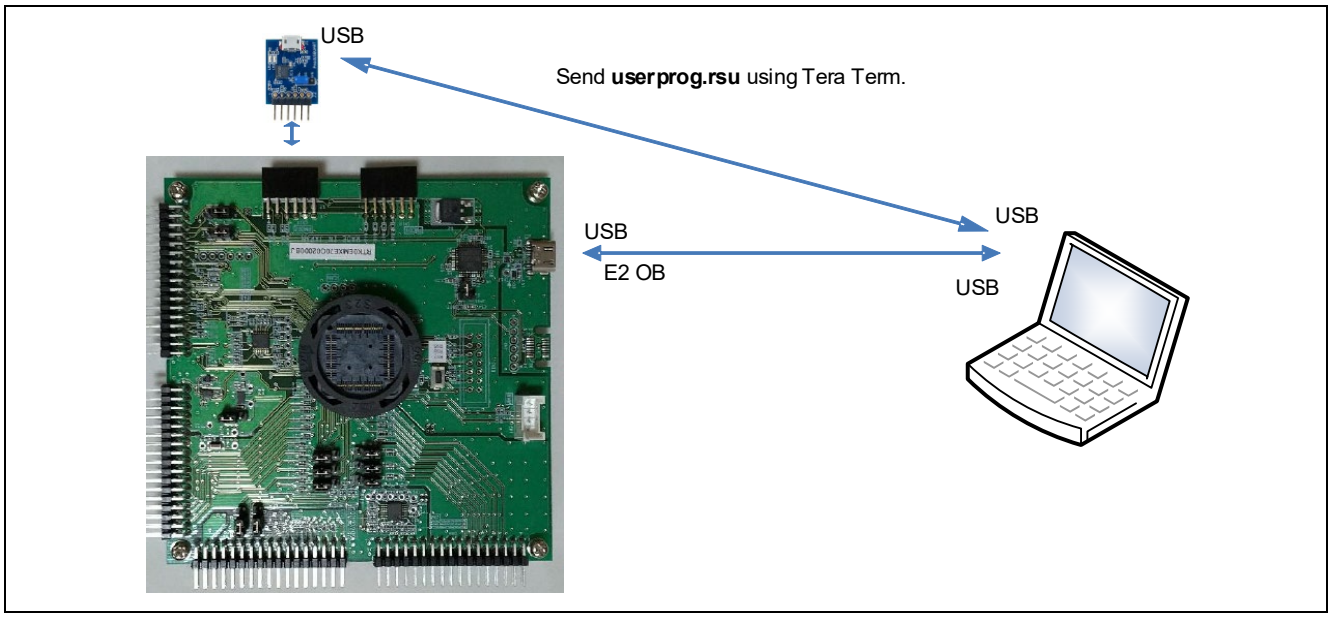

図 6-16 UART を使用したファームアップデートを行うときの MCB RX26T と PC の接続

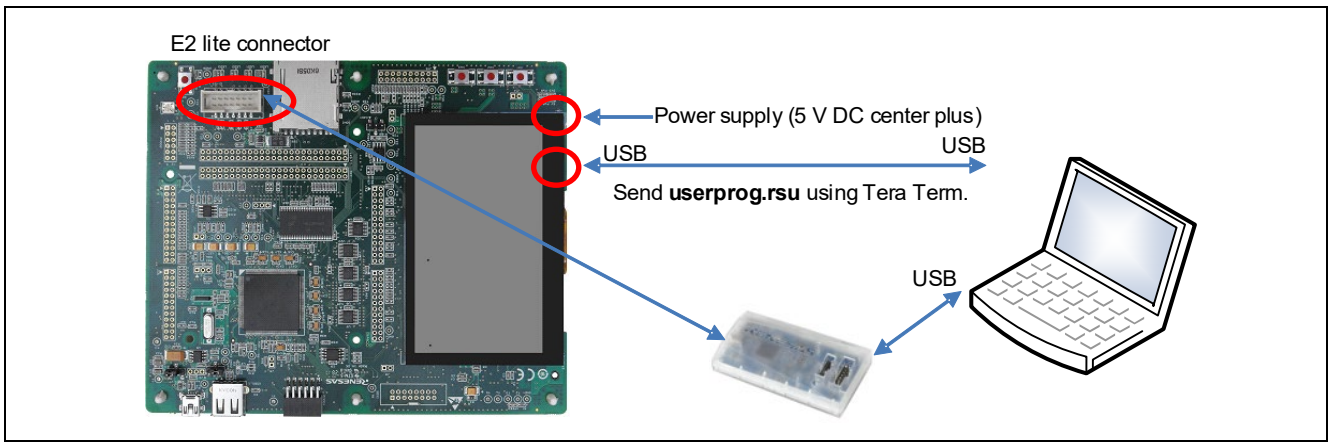

図 6-17 UART を使用したファームアップデートを行うときの RSK RX65N-2MB と PC の接続

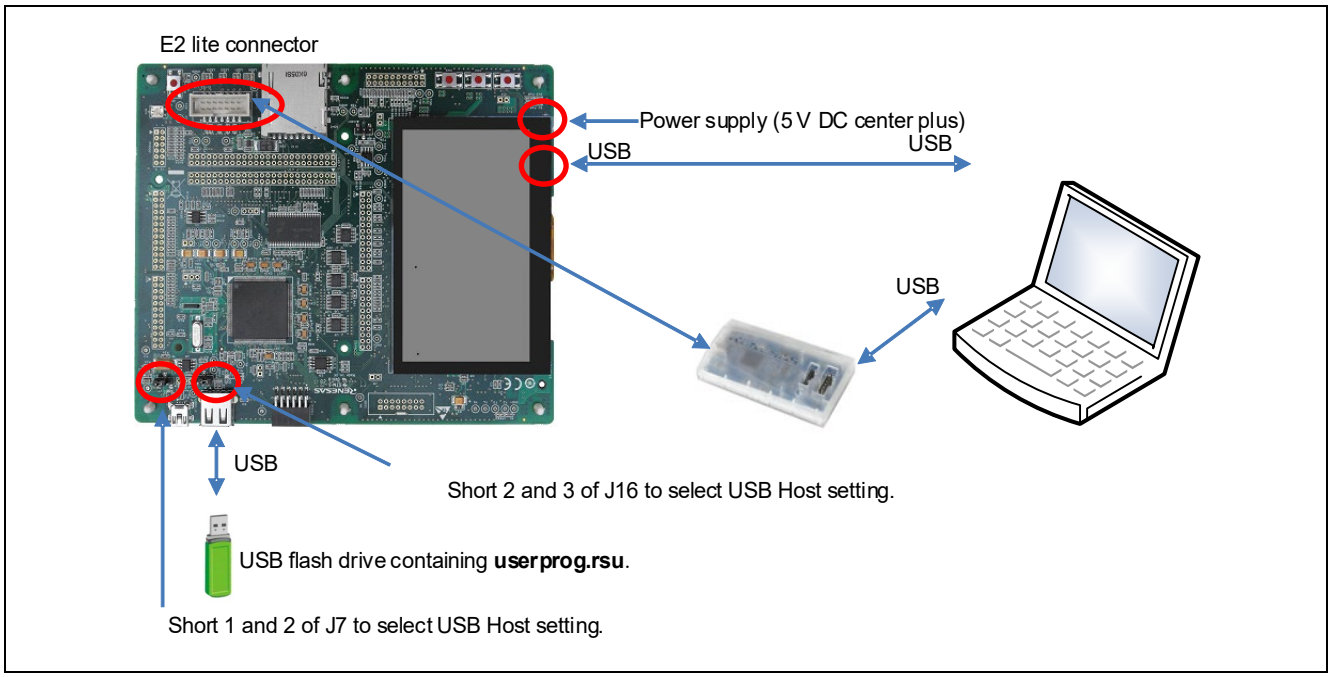

図 6-18 USB を使用したファームアップデートを行うときの RSK RX65N-2MB と PC の接続

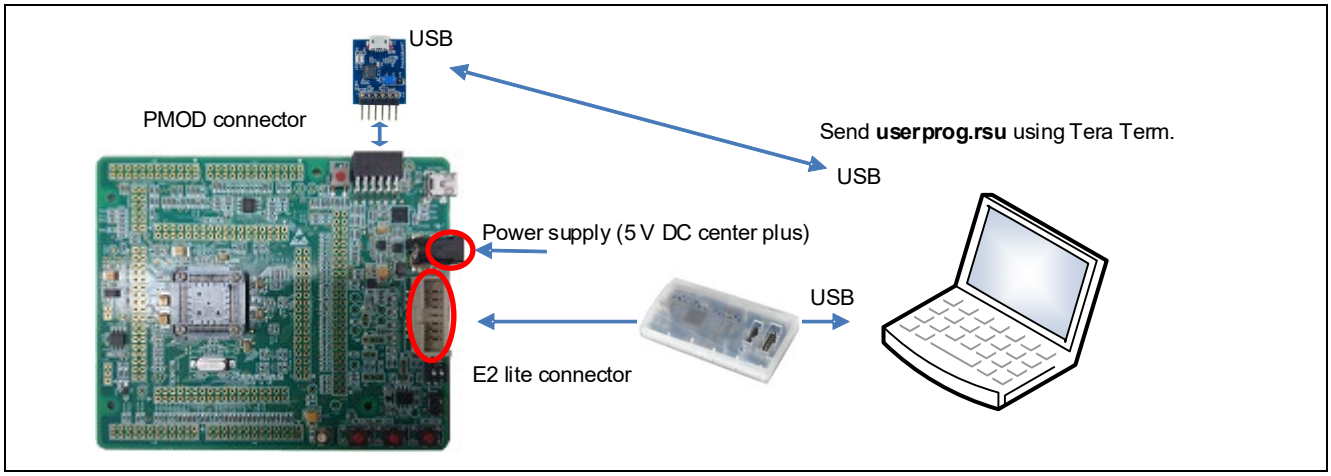

図 6-19 UART※1 を使用したファームアップデートを行うときの RSK RX66T と PC の接続

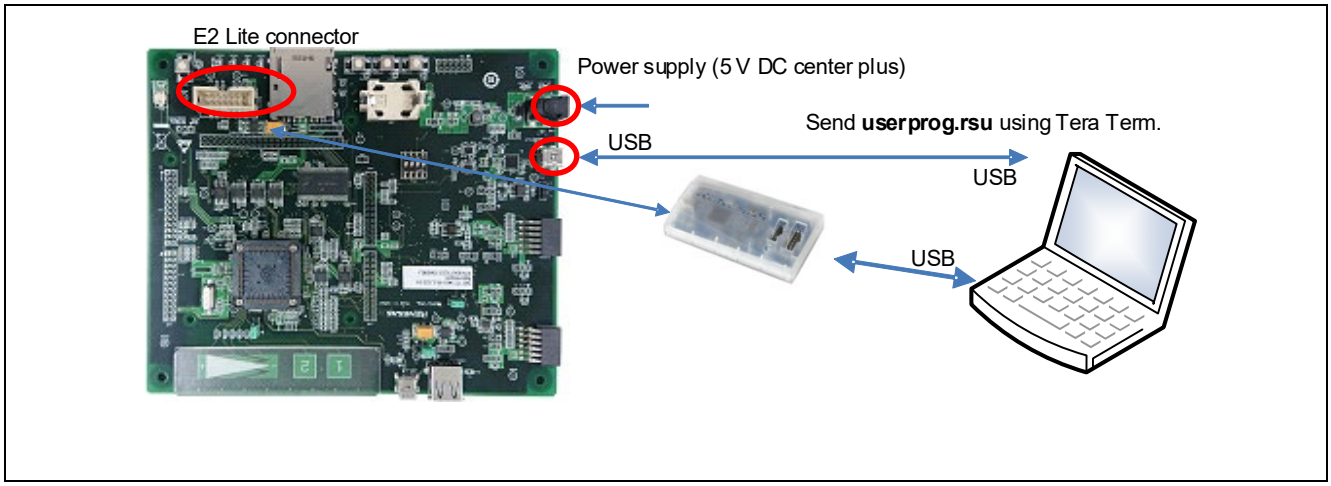

図 6-20 UART を使用したファームアップデートを行うときの RSK RX671 と PC の接続

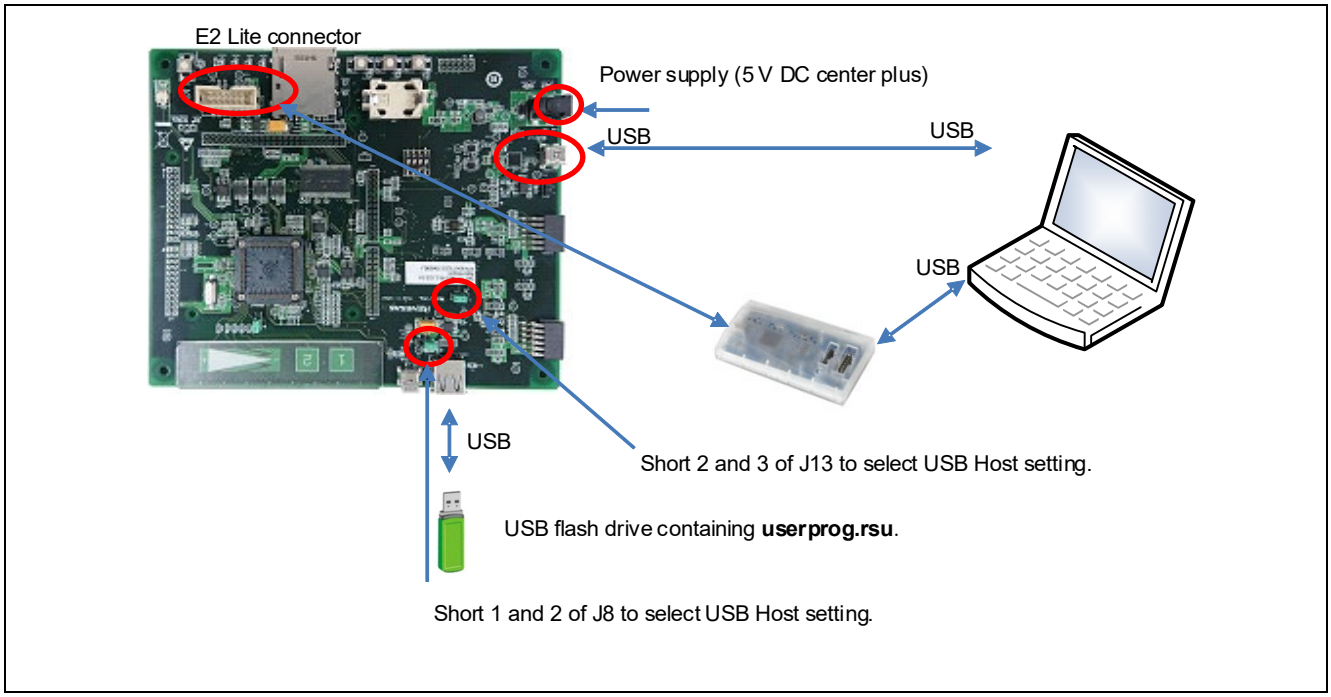

図 6-21 USB を使用したファームアップデートを行うときの RSK RX671 と PC の接続

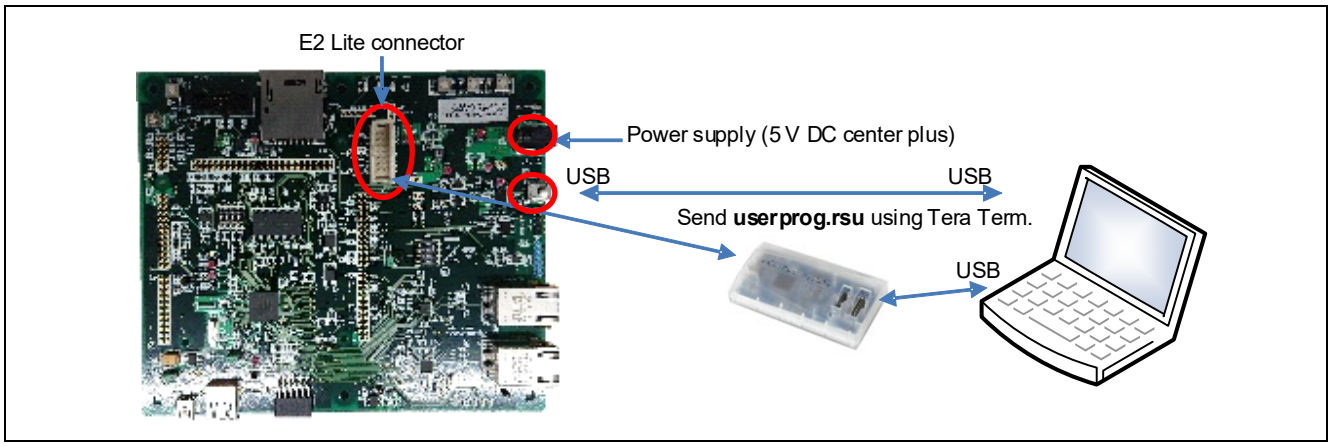

図 6-22 UART を使用したファームアップデートを行うときの RSK RX72M と PC の接続

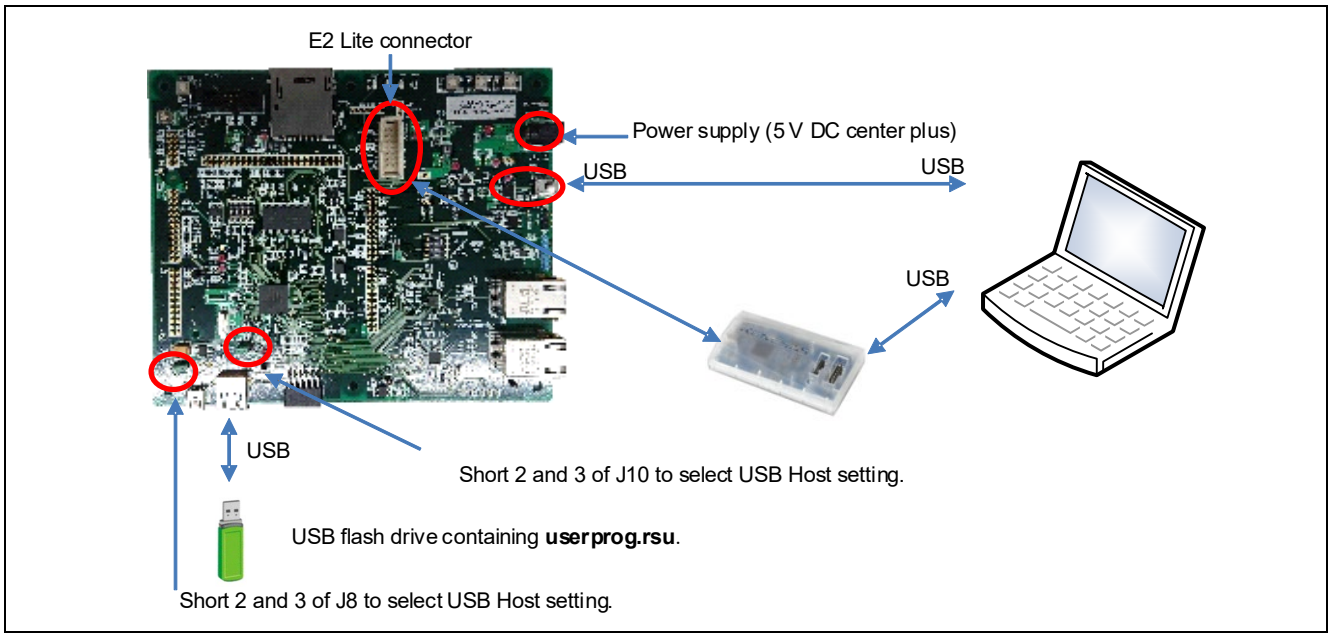

図 6-23 USB を使用したファームアップデートを行うときの RSK RX72M と PC の接続

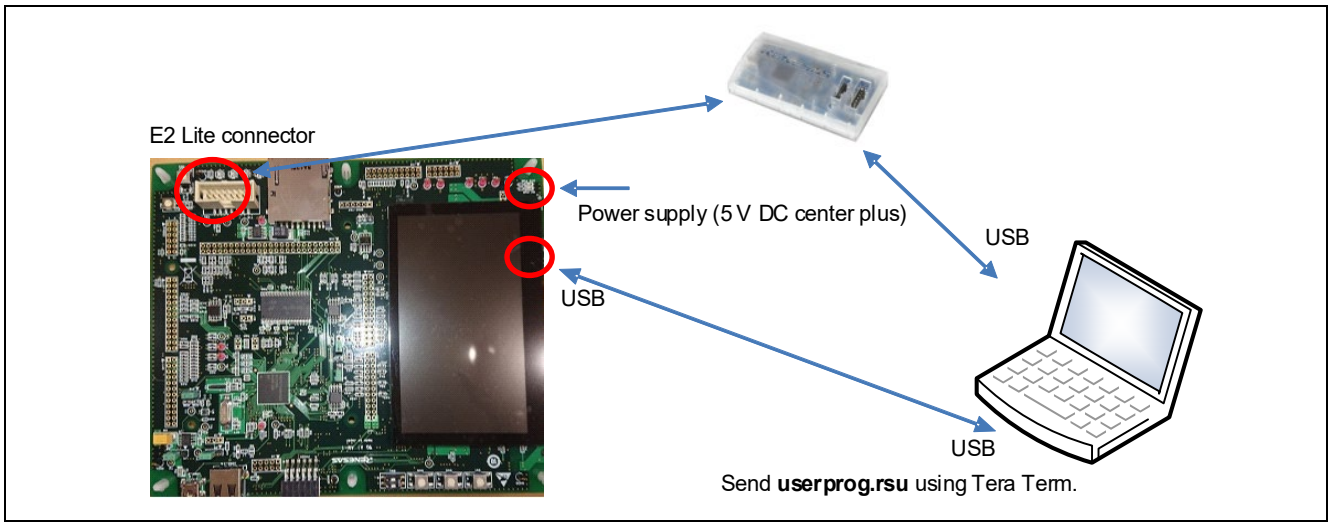

図 6-24 UART を使用したファームアップデートを行うときの RSK RX72N と PC の接続

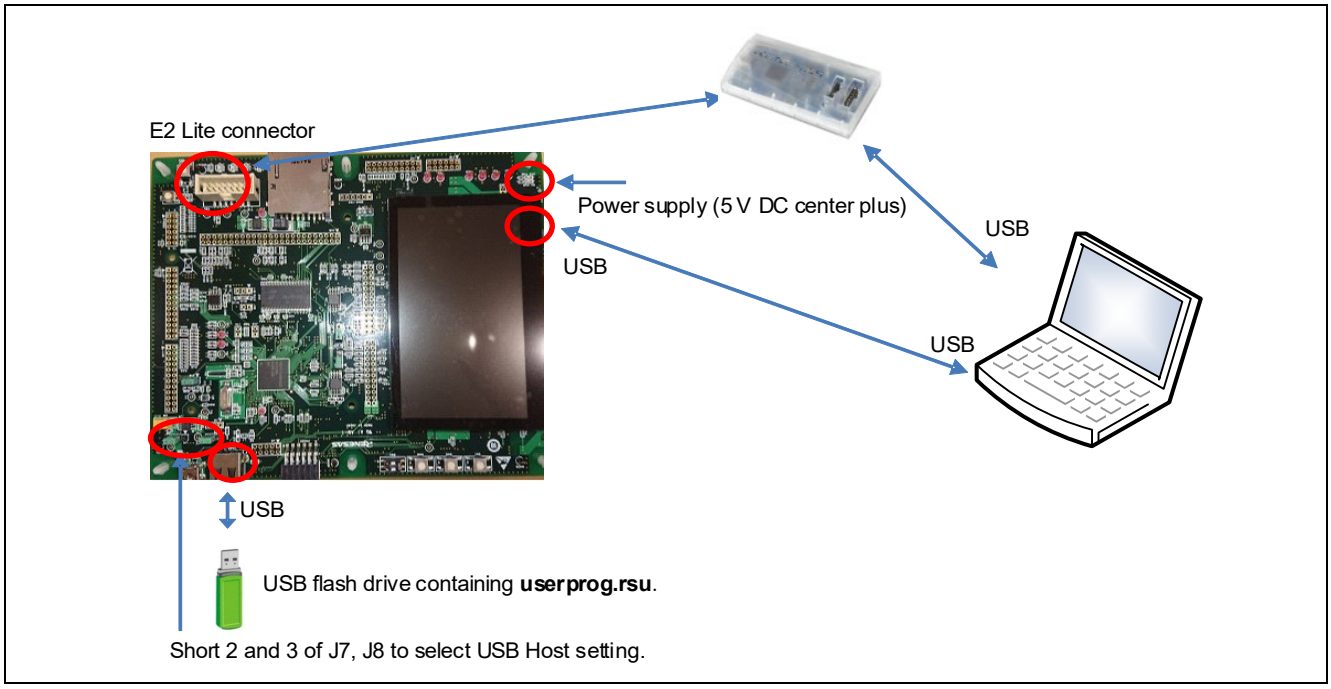

図 6-25 USB を使用したファームアップデートを行うときの RSK RX72N と PC の接続

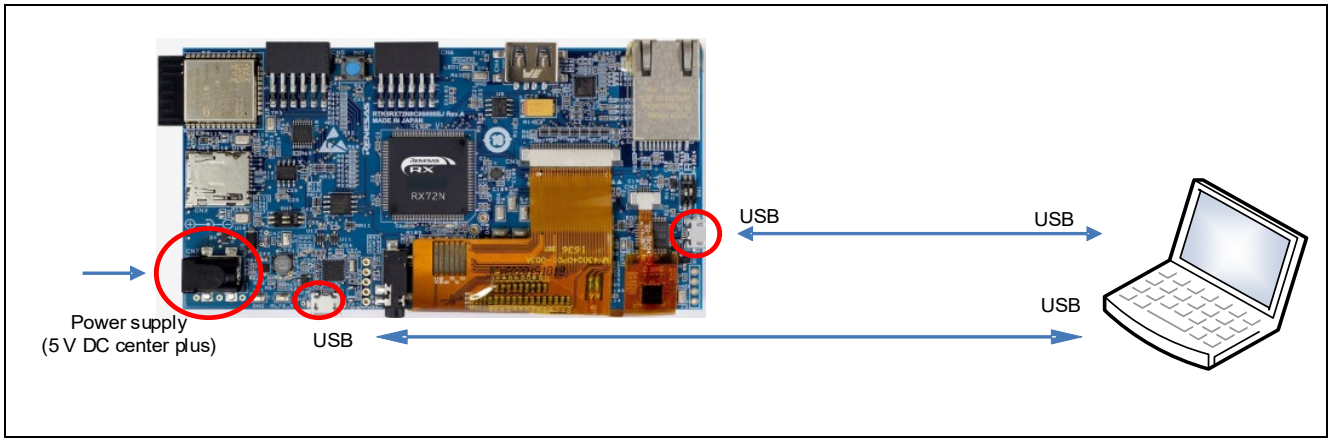

図 6-26 UART を使用したファームアップデートを行うときの RX72N Envision Kit と PC の接続

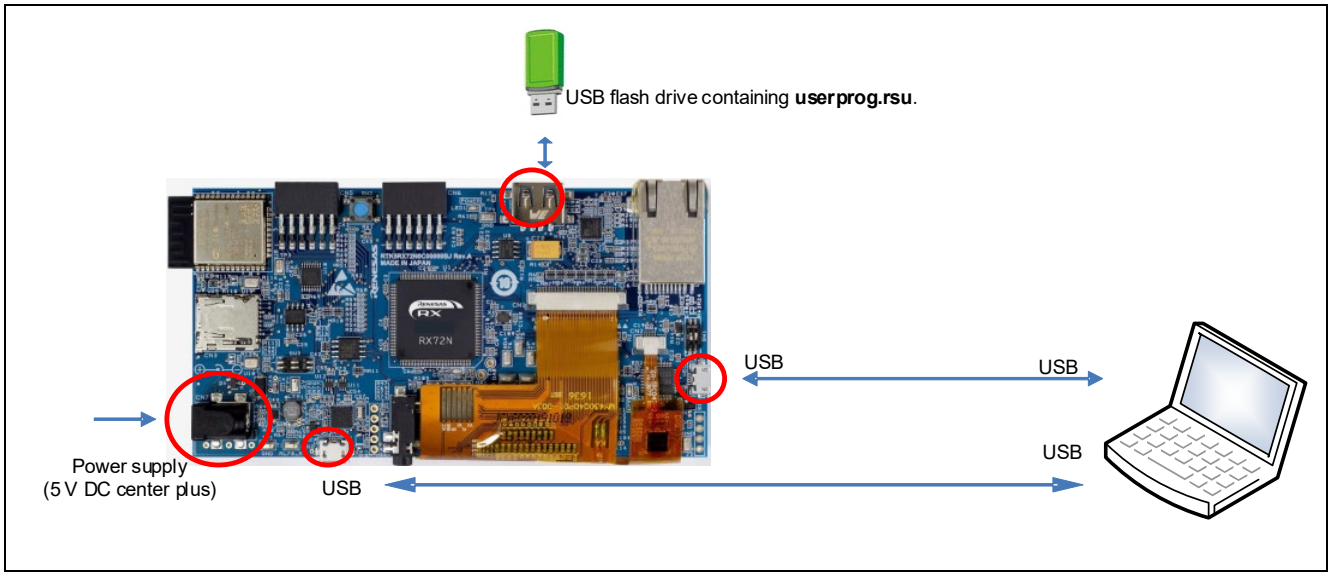

図 6-27 USB を使用したファームアップデートを行うときの RX72N Envision Kit と PC の接続

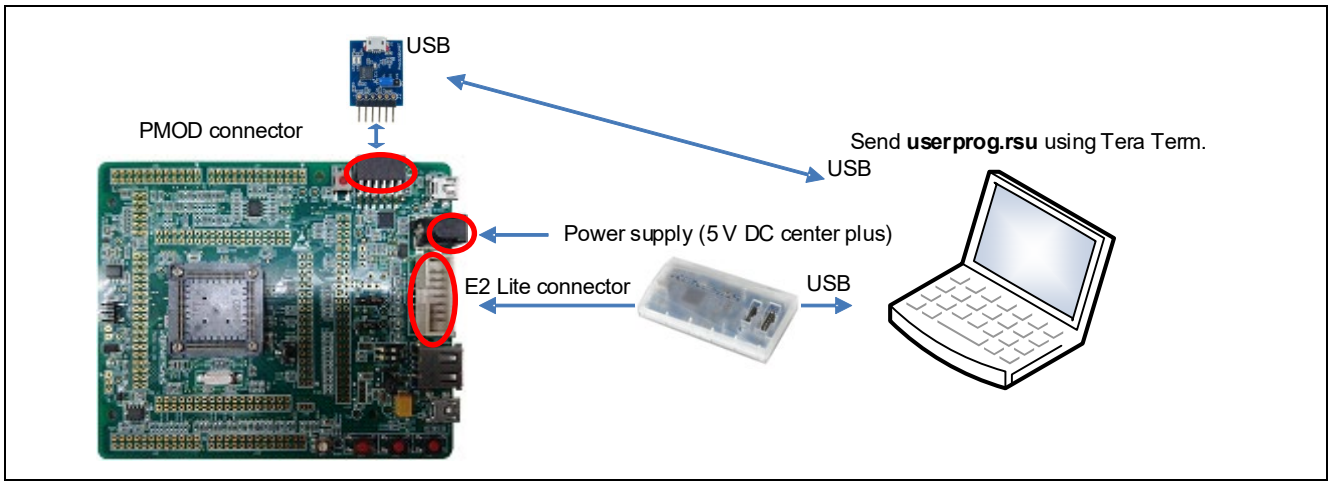

図 6-28 UART※1 を使用したファームアップデートを行うときの RSK RX72T と PC の接続

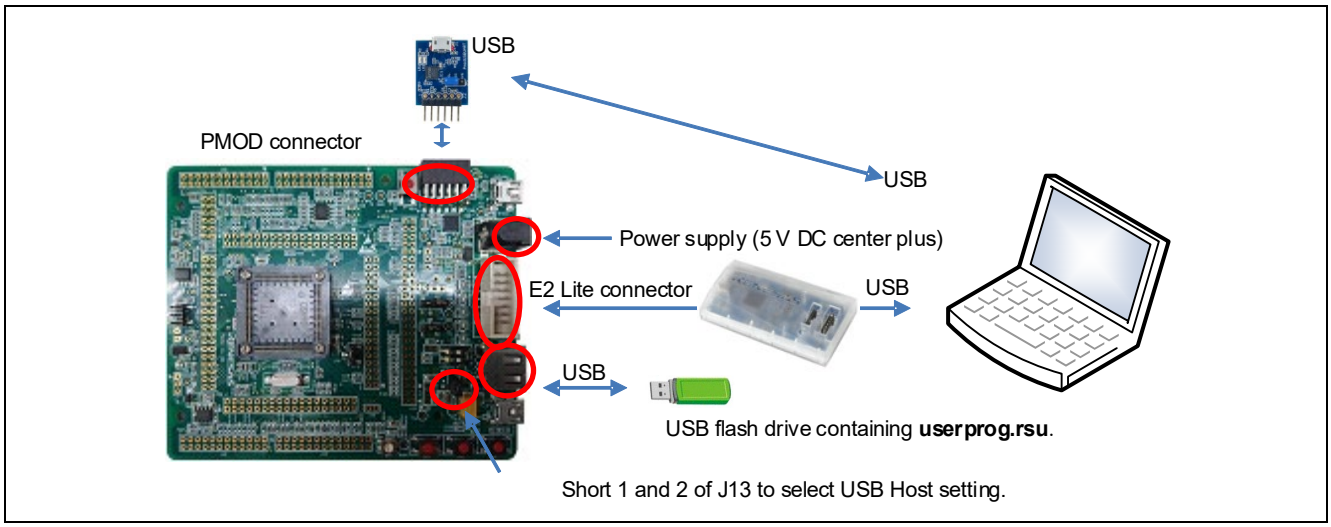

図 6-29 USB を使用したファームアップデートを行うときの RSK RX72T と PC の接続

【*1 】本デモプロジェクトでは、Digilent 社製 Pmod USBUART を使用して動作確認を行っています。RX RSK ボード上の PMOD コネクタとは以下のように接続を行う必要があります。

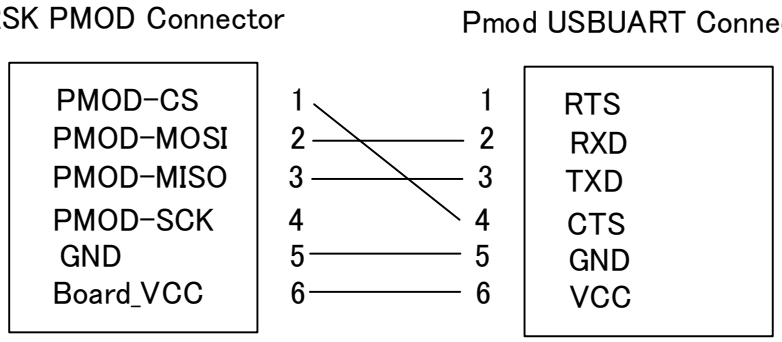

図 6-30　RX RSK と Pmod USBUART の接続図

【*2 】RSK RX231 で PMOD1 コネクタに接続された Pmod USBUART 基板のフロー制御を行うには、以下実装済の 0Ω抵抗を未実装になっているパターンへの載せ替えが必要になります。

実装済		未実装
R125	→	R126
R128	→	R127
R135	→	R134

Tera Term のシリアル設定は以下の通りです。

表 6-6　ボード RSK RX231, RSK RX66T, RSK RX72T の場合

項目	設定値
スピード	115200bps
データ	8 ビット
パリティ	なし
ストップビット	1 ビット
フロー制御	RTS/CTS
改行コード	CR
ローカルエコー	OFF

表 6-7　ボード MCB RX26T, RSK RX65N-2MB, RSK RX671, RSK RX72M, RSK RX72N, RX72N Envision Kit の場合

項目	設定値
スピード	57600bps
データ	8 ビット
パリティ	なし
ストップビット	1 ビット
フロー制御	なし
改行コード	CR
ローカルエコー	OFF

6.2.2　セキュアブート・ファームアップデートプロジェクトの概要

本章では、セキュアブートプロジェクト、ファームアップデートプロジェクトの概要について説明いたします。各セキュアブート、ファームウェアアップデートプロジェクトのディレクトリ構成を以下の表 **6-8** に示します。

表 6-8 セキュアブート・ファームウェアアップデートプロジェクトのディレクトリ構成

ディレクトリ名	内容
FITDemos	デモプロジェクトフォルダ
rx*XXX*_*bbb*_tsip_secure_update *1 *2	セキュアブート/セキュアアップデートの実装例
rx*XXX*_*bbb*_tsip_secure_boot *1 *2	セキュアブートファームウェア
key	動作確認に使用可能な UFPK と W-UFPK ファイル
skmt	Security Key Management Tool の設定ファイル
src	セキュアブートファームウェアのソースコード
rx*XXX*_*bbb*_tsip_user_program *1 *2	セキュアアップデート後のユーザプログラム
src	ユーザプログラムのソースコード

*1：rx*XXX* には、サポートしている RX グループ名が入ります。

*2：*bbb* には、サポートしているボード名が入ります。通常 Update の IF として USB と UART を用意していますが、USB をサポートしていない MCU の場合、UART のみサポートします。UART のみサポートしているプロジェクトには、_secure_boot および_user_program の前に_sci も入ります。

RSK：rsk　(UART のみサポート：rsk(_tsip)_sci)

MCB：mcb (UART のみサポート：mcb(_tsip)_sci)

Envision Kit：ek

RX TSIP FIT のセキュアブート、ファームアップデートプロジェクトでは、Flash ROM 領域を 2 面に分けて、メイン面、バッファ面として扱うサンプルプロジェクトです。

メイン面：実行対象のユーザプログラムを格納するエリア

バッファ面：更新対象のイメージを格納するエリア

Flash ROM に Dual Bank 機能を持つデバイスでは、Dual Bank 機能を活用するファームアップデート動作でファームアップデートを行います。

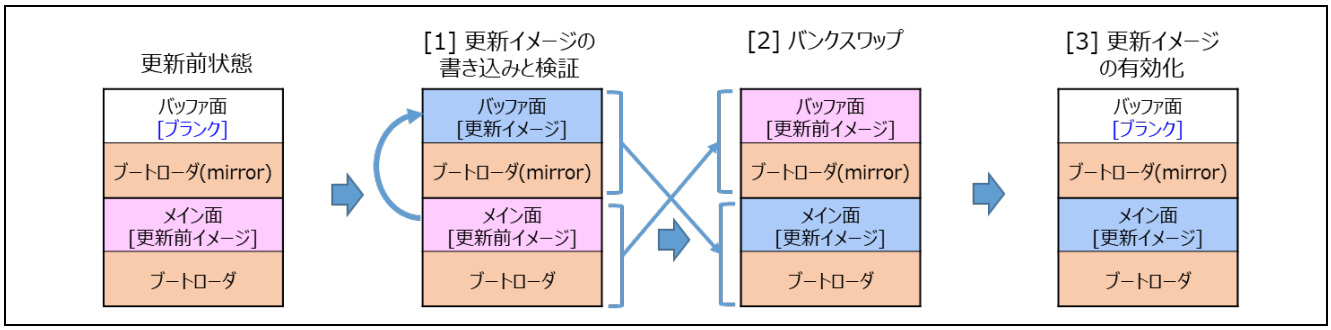

図 6-31 Dual Bank を活用するファームアップデート動作

Flash ROM に Dual Bank 機能を持たないデバイスでは、Flash ROM をメイン面とバッファ面の 2 面に分けて使用する半面更新方式でファームアップデートいたします。

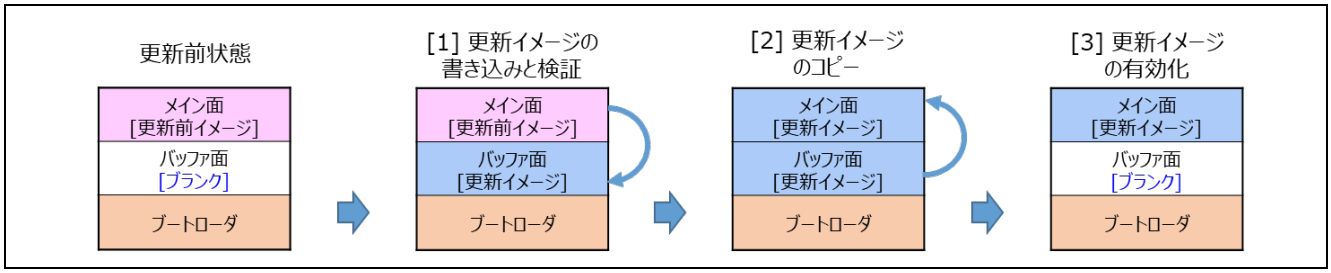

図 6-32 半面更新方式のファームアップデート動作

これらの状態管理と正当性検証で使用する MAC 値ならびに、復号処理で使用する鍵は、Data Flash に置かれている key_block_data によって管理しています。鍵のインジェクション時もしくは、データ更新途中の電断対策で、Data Flash 内に Main 領域と Mirror 領域の 2 面に同じデータを保持して、データ管理をしています。

表 6-9　key_block_data 構造体

名称	型	説明
st_key_block_data_t	-	C_FIRMWARE_UPDATE_CONTROL_BLOCK, C_FIRMWARE_UPDATE_CONTROL_BLOCK_MIRROR に置かれる鍵データ構造体
firmware_update_control_data	-	firmware update 実施時の mac 情報格納構造体
user_program_max_cnt	uint32_t	firmware update 時の firmware バイトサイズ
lifecycle_state	uint32_t	表 6-10　lifecycle_state 説明
program_mac0[]	uint32_t	バッファ面の MAC 値
program_mac1[]	uint32_t	メイン面の MAC 値
key_data	-	鍵データ格納構造体
user_aes128_key_index	tsip_aes_key_index_t	暗号化されたユーザプログラムの復号で使用する Key Index
hash_sha1[]	uint8_t	firmware_update_control_data と key_data の SHA1 HASH 値

表 6-10　lifecycle_state 説明

Status	説明	
BLANK	鍵	注入未完了
	プログラム	バッファ面：インストール未完了 メイン面　：インストール未完了
UPDATING	鍵	注入完了　鍵データ HASH 未設定
	プログラム	バッファ面：更新プログラム書き込み完了 メイン面　：更新前プログラム
ON_THE_MARKET	鍵	注入完了　鍵データ HASH 設定済
	プログラム	バッファ面：更新前プログラム メイン面　：プログラムは更新後のプログラム、プログラム実行可能

6.2.2.1　セキュアブートプロジェクト

セキュアブートプロジェクトは、ユーザプログラムの復号で使用する鍵注入とユーザプログラムの検証を行います。また、初期ユーザプログラム(本サンプルでは、ファームアップデートプログラム)をインストールするためのプログラムも含まれています。

セキュアブートプロジェクトのフローは以下になります。

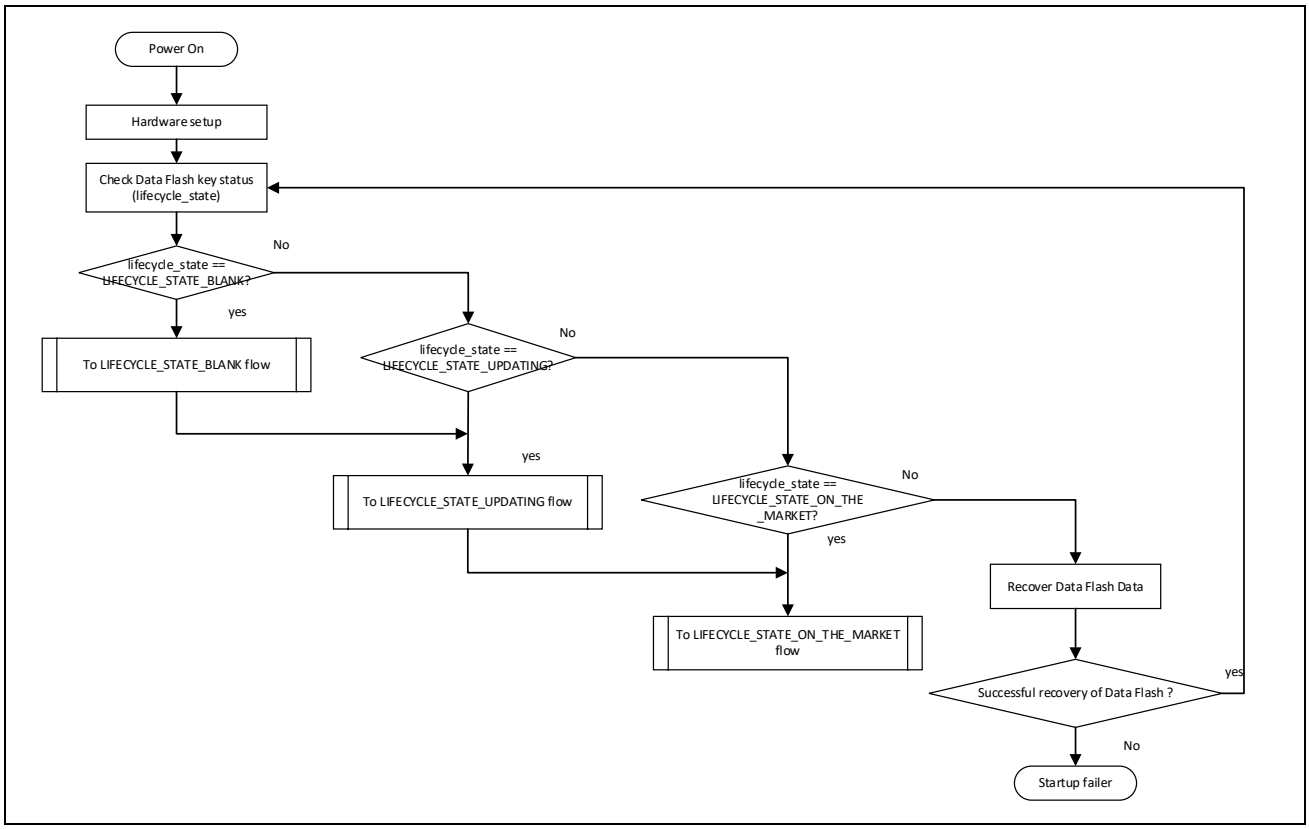

図 6-33　セキュアブート メインフロー

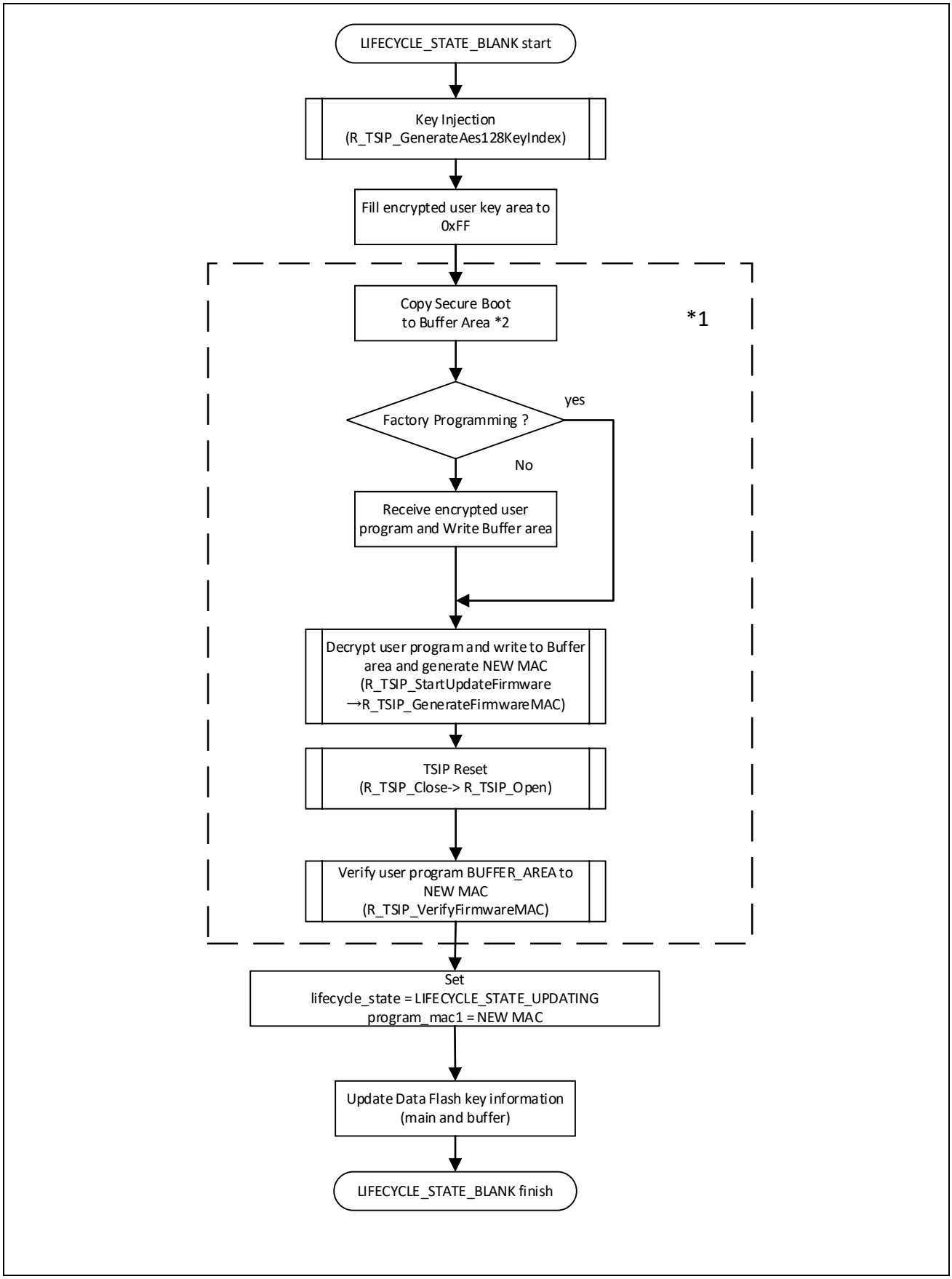

図 6-34　セキュアブート LIFECYCLE_STATE_BLANK フロー

*1 Factory Programming ではない場合、R_FWUP_SecureBoot 内で処理を行います。

*2 Dual Bank 搭載デバイスの場合実施します。

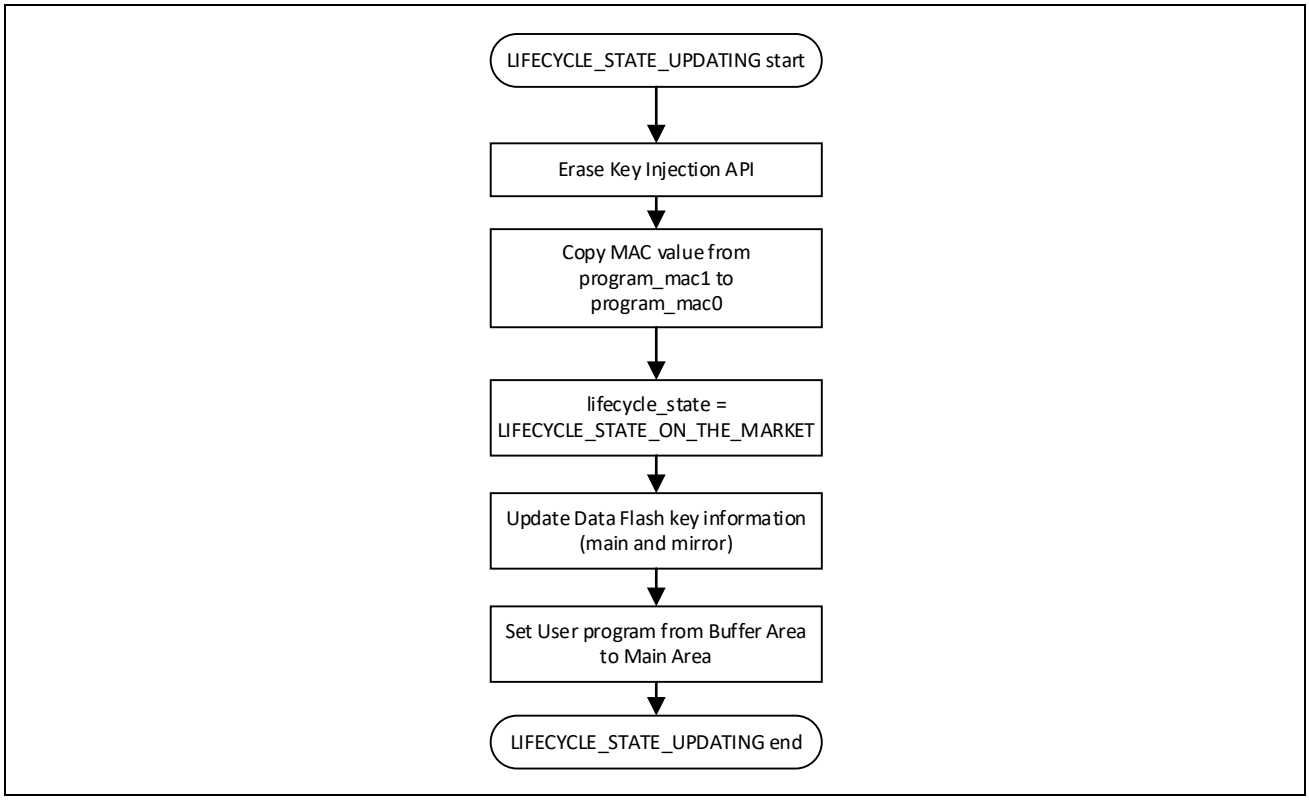

図 6-35　セキュアブート LIFECYCLE_STATE_UPDATING フロー

鍵の注入に使用した API は、鍵の注入完了後に消去することを推奨します。本デモプロジェクトにおいても消去していますが、消去が失敗した場合でも動作を継続する仕様となっています。

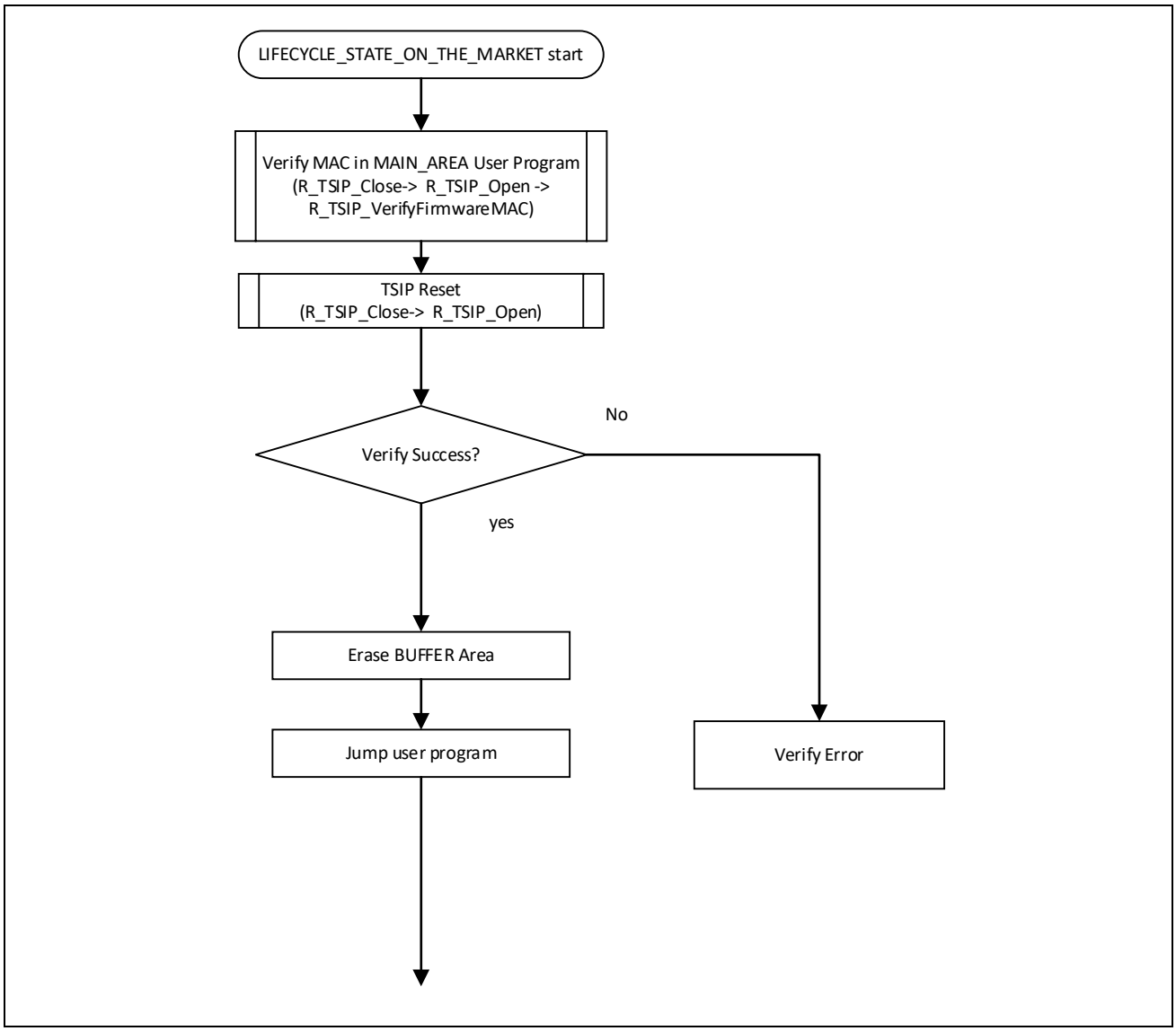

図 6-36　セキュアブート LIFECYCLE_STATE_ON_THE_MARKET フロー

「Jump user program」を実行するまでに、R_USB_Close、R_TSIP_Close、R_FLASH_Close および R_SCI_Close を実行します。

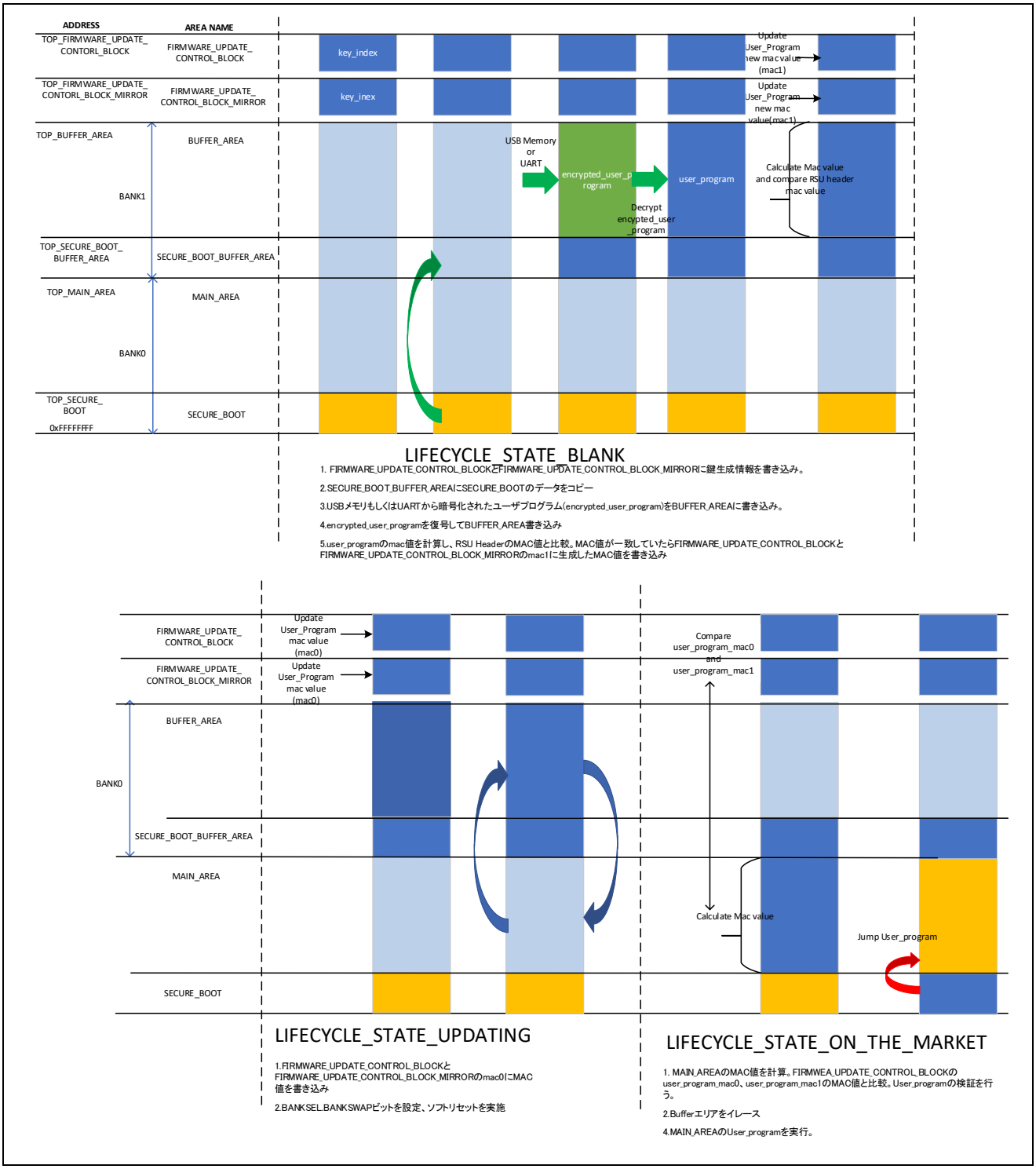

図 6-37　セキュアブート 各 lifecycle_state の内蔵 Flash メモリ状態 (Dual Bank 搭載デバイス)

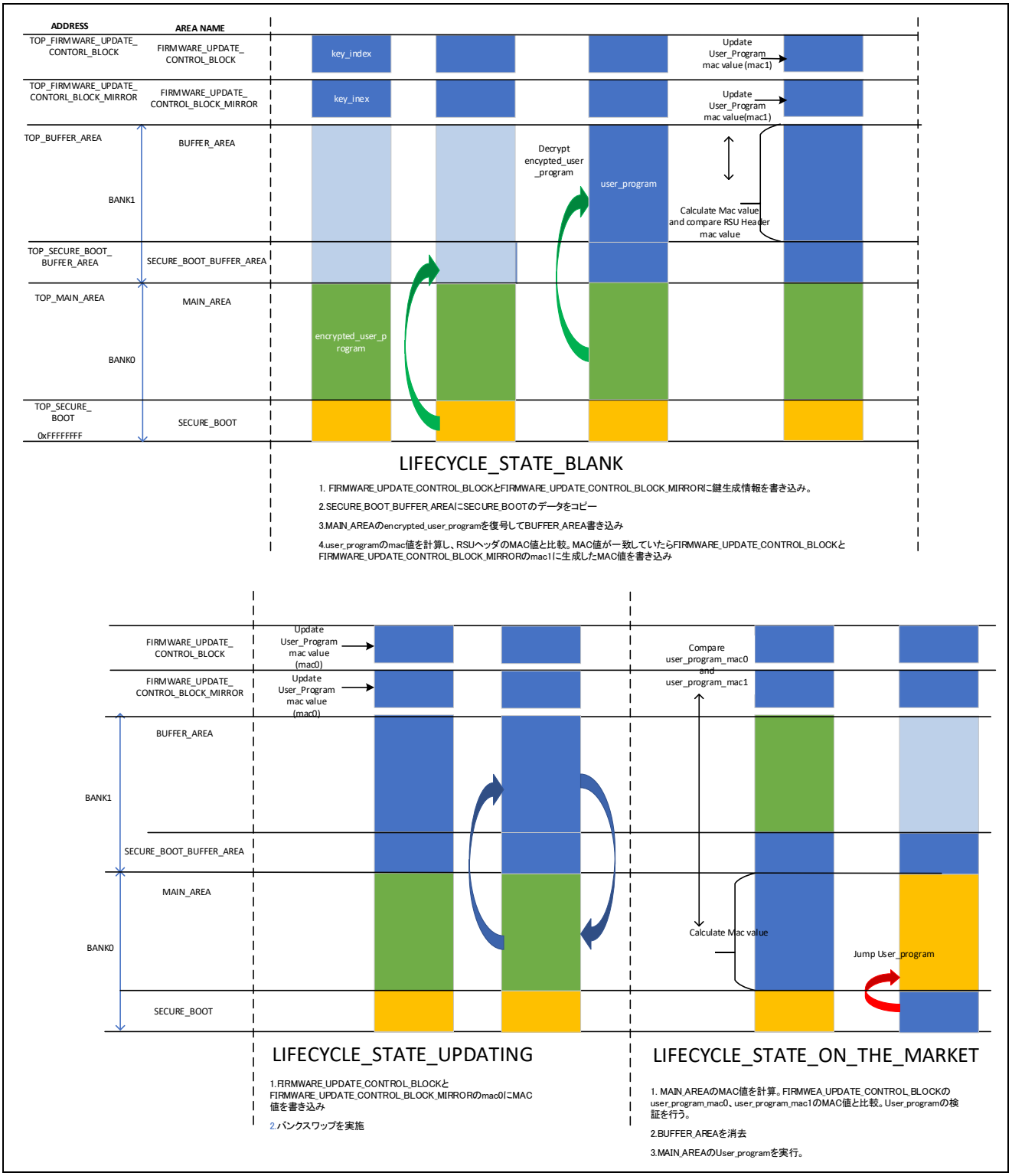

図 6-38　セキュアブート 各 lifecycle_state の内蔵 Flash メモリ状態
(Dual Bank 搭載デバイス　Factory Programming)

表 6-11　セキュアブートプロジェクトで設定している各エリアのアドレス(Dual Bank 対応デバイス)

	RX26T	RX65N	RX671	RX72M,RX72N
TOP_FIRMWARE_UPDATE_CONTORL_BLOCK	0x00100000	0x00100000	0x00100000	0x00100000
TOP_FIRMWARE_UPDATE_CONTORL_BLOCK_MIRROR	0x00102000	0x00104000	0x00101000	0x00104000
TOP_BUFFER_AREA	0xFFF80000	0xFFE00000	0xFFE00000	0xFFC00000
TOP_SECURE_BOOT_BUFFER_AREA	0xFFFB0000	0xFFEF0000	0xFFEF0000	0xFFDF0000
TOP_MAIN_AREA	0xFFFC0000	0xFFF00000	0xFFF00000	0xFFE00000
TOP_SECURE_BOOT_AREA	0xFFFF0000	0xFFFF0000	0xFFFF0000	0xFFFF0000

セキュアブートプロジェクトは、以下のモジュールを SECURE_BOOT セクションに置いています。このため、各 FIT モジュールのソースに変更をしています。

表 6-12　変更が必要な FIT モジュールとソース

モジュール	フォルダ	ファイル
bsp	r_bsp\board\generic_rx*XXX*	hwsetup.c
	r_bsp\mcu\all	dbsct.c
		lowlvl.c
		lowsrc.c
		mcu_locks.c
		r_bsp_common.c
		r_bsp_cpu.c
		r_bsp_interrupts.c
		r_bsp_locking.c
		r_bsp_mcu_startup.c
		r_bsp_software_interrupt.c
		sbrk.c
	r_bsp\mcu\rx*XXX*	mcu_clocks.c
		mcu_init.c
		mcu_interrupts.h (RX231, RX66T, RX72T のみ)
		mcu_interrupts.c
		mcu_mapped_interrupts.c (RX231 を除く)
		r_bsp_vbatt.c (RX231 のみ)
byteq	r_byteq\src	r_byteq.c
flash	r_flash\src	r_flash_fcu.c
		r_flash_group.c
		r_flash_nofcu.c
		r_flash_rx.c
r_fwup	r_fwup	r_fwup_if.h
	r_fwup\src	r_fwup_boot_loader.c
		r_fwup.c
		r_fwup_private.h
sci	r_sci_rx\src	r_sci_rx.c
	r_sci_rx\src\targets\rx*XXX*	r_sci_rx*XXX*.c
		r_sci_rx*XXX*_data.c
general	general	r_cg_hardware_setup.c
		r_smc_cgc.c
		r_smc_cgc_user.c
		r_smc_interrupt.c
pincfg	r_pincfg	r_sci_rx_pinset.c
		r_usb_basic_pinset.c (RX231, RX26T,RX66T を除く)
		r_usb_basic_mini_pinset.c (RX231 のみ)

* rx*XXX* には RX グループ名が入ります。

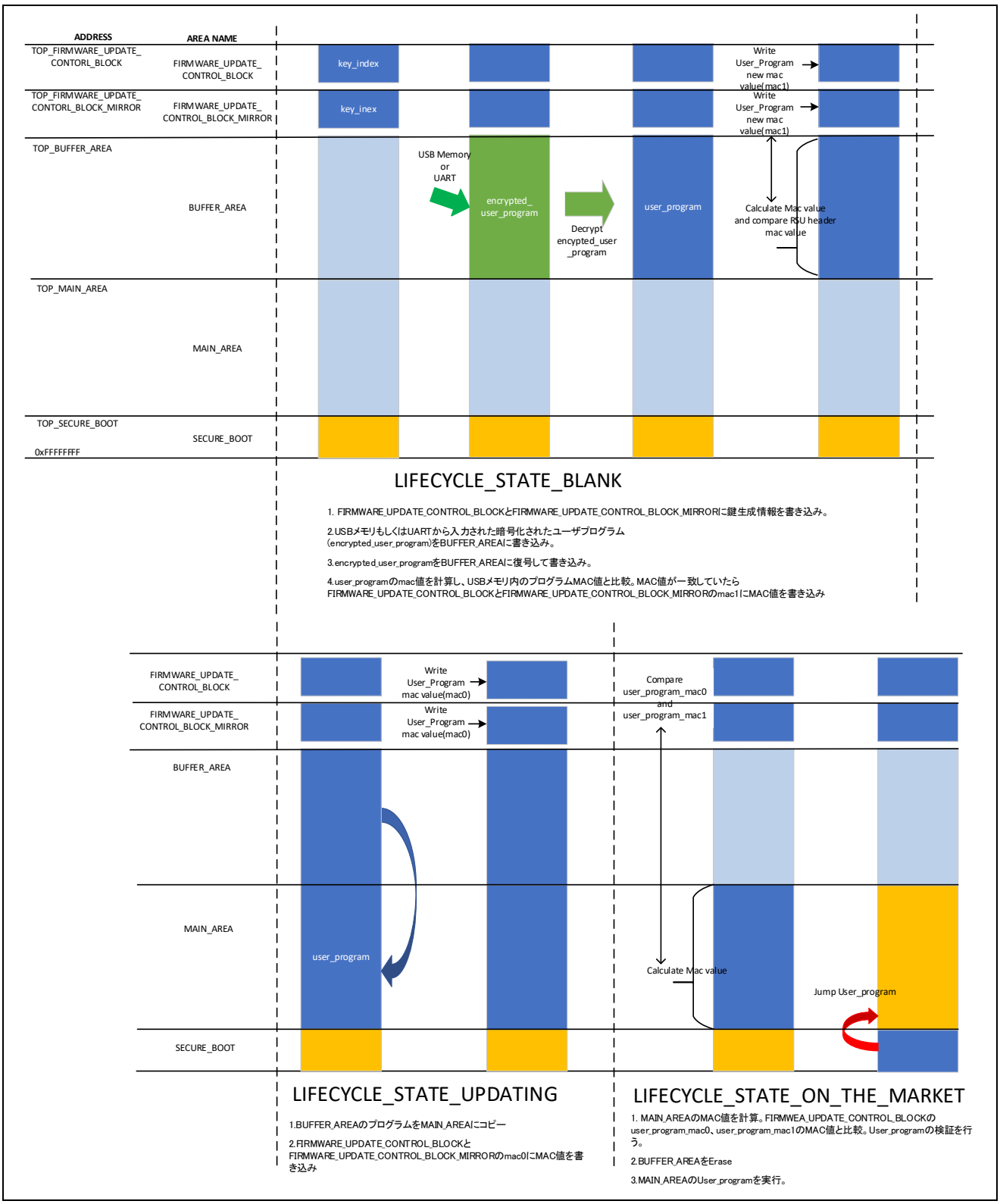

図 6-39　セキュアブート 各 lifecycle_state の内蔵 Flash メモリ状態(Dual Bank 非搭載デバイス)

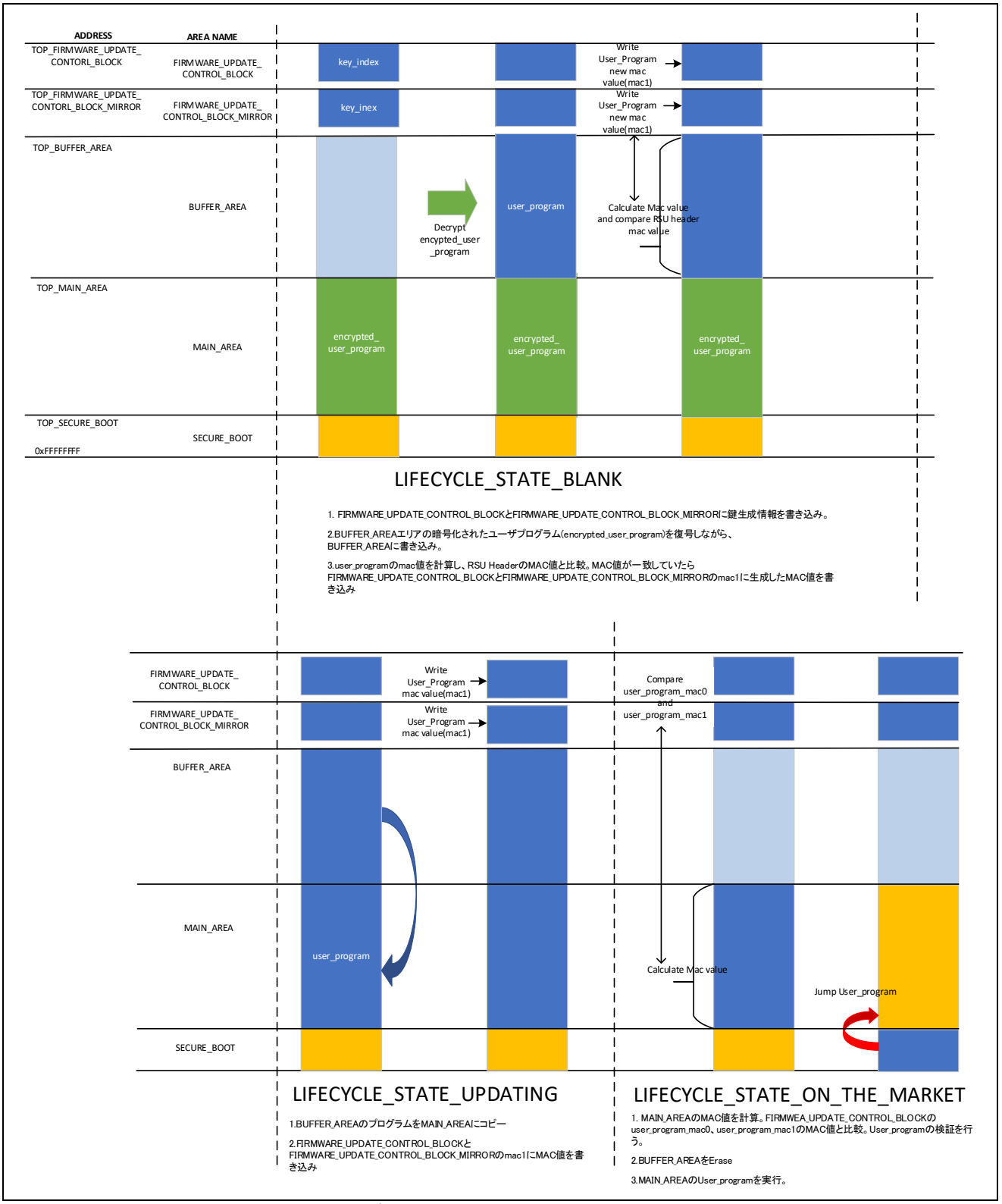

図 6-40　セキュアブート 各 lifecycle_state の内蔵 Flash メモリ状態
(Dual Bank 非搭載デバイス　Factory Programming 動作時)

表 6-13　セキュアブートプロジェクトで設定している各エリアのアドレス(Dual Bank 非搭載デバイス)

	RX231	RX66T	RX72T
TOP_FIRMWARE_UPDATE_ CONTORL_BLOCK	0x00100000	0x00100000	0x00100000
TOP_FIRMWARE_UPDATE_ CONTORL_BLOCK_MIRROR	0x00101000	0x00104000	0x00104000
TOP_BUFFER_AREA	0xFFF80000	0xFFF80000	0xFFF00000
TOP_MAIN_AREA	0xFFFB8000	0xFFFB8000	0xFFF78000
TOP_SECURE_BOOT_AREA	0xFFFF0000	0xFFFF0000	0xFFFF0000

6.2.2.2　ファームアップデートプロジェクト

ファームアップデートプロジェクトは、ユーザプログラムを復号し、ユーザプログラムの検証を行います。ファームアップデートプロジェクトの update コマンド受信後のフローは以下になります。

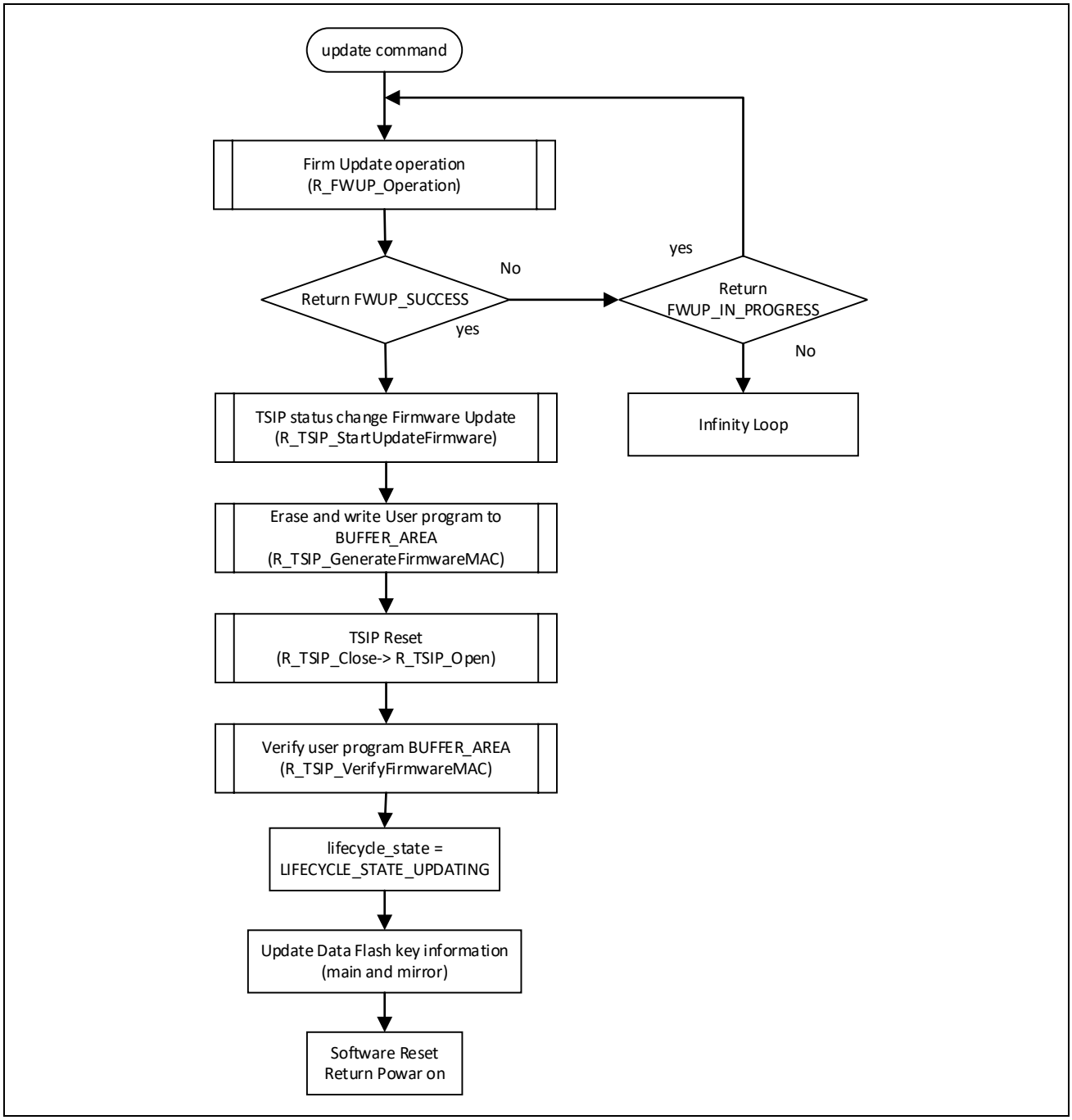

図 6-41　ファームアップデートフロー

6.2.3　セキュアブート・ファームアップデートの実行例

全てのボードで UART によるファームアップデート動作をサポートします。加えて、デバイスに USB HOST 機能を持ち、USB ホストソケットが搭載されたボードでは、USB メモリによるファームアップデート動作をサポートします。

(1) ファームアップデート用暗号鍵ファイルの生成

最初に Security Key Management Tool の「鍵のラッピング」で、ユーザプログラムの暗号化で使用する Image Encryption Key を暗号化するための鍵ファイルを生成します。

セキュアブートプロジェクトでは、UFPK、Key Encryption Key として以下の値を使用しています。

UFPK = “22222222222222222222222222222221111111111111111111111111111111”

Key Encryption Key = “0123456789abcdef0123456789abcdef”

・コマンドライン版を使用される場合
以下コマンドを実行してください。

```
> skmt.exe /genkey /iv "55aa55aa55aa55aa55aa55aa55aa55aa" /ufpk
  file="{$skmt_loc}\..\key\sample.key"
  /wufpk file="{$skmt_loc}\key\sample.key_enc.key"
  /mcu "RX-TSIP" /keytype "AES-128" /key "0123456789ABCDEF0123456789ABCDEF"
  /filetype "csource" /keyname "euk_aes128" /output "{$skmt_loc}\genkey\euk_aes128.c"
```

{$skmt_loc}には、実行する Secure Boot プロジェクトフォルダのパスを入力してください。

・スタンドアロン版を使用される場合
以下を入力してください。

-「鍵の種類」タブ
「AES」-「128bits」を選択

-「鍵のデータ」タブ
平文データに”0123456789abcdef0123456789abcdef”を入力

-「ラッピング鍵」
UFPK　: サンプルに付属している sample_key.key
W-UFPK : サンプルに付属している sample_key_enc.key

-「IV」
「指定値を使用する」を選択し、”55aa55aa55aa55aa55aa55aa55aa55aa”を入力

-「出力」
フォーマット : C ソース
ファイル名　: euk_aes128.c
Key name　: euk_aes128

これらの設定はサンプルに付属している Security Key Management Tool の設定保存ファイル rx*XXX*_SecureUpdate.skmt をロードして使用可能です。
rx*XXX*_SecureUpdate.skmt は xml ファイル形式のテキストで編集可能なファイルです。
rx*XXX*_SecureUpdate.skmt 内の{$skmt_loc}に、Secure Boot プロジェクトのフォルダパスを入力してください。

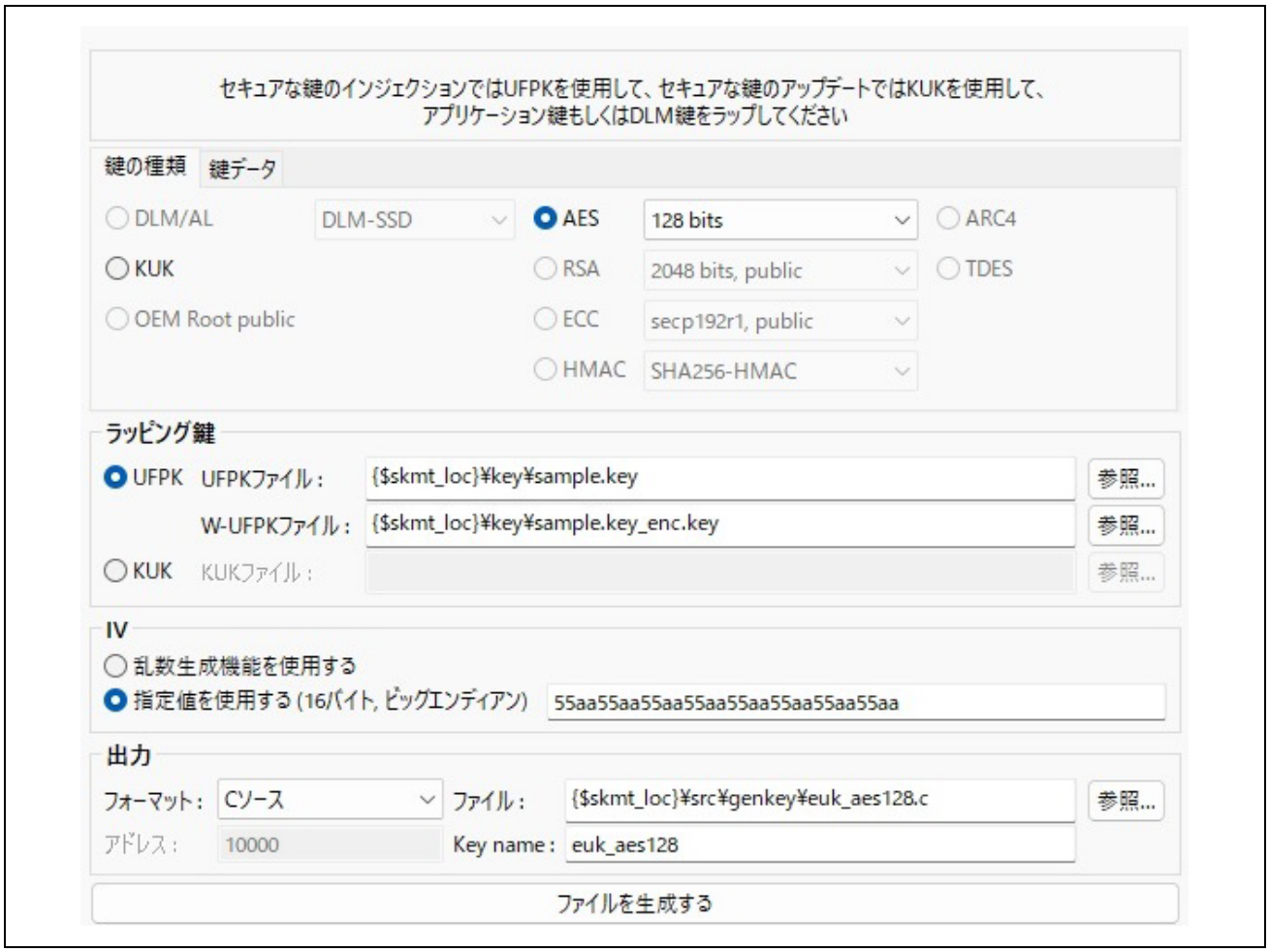

図 6-42 鍵データファイルの生成([鍵のラッピング]タブ)

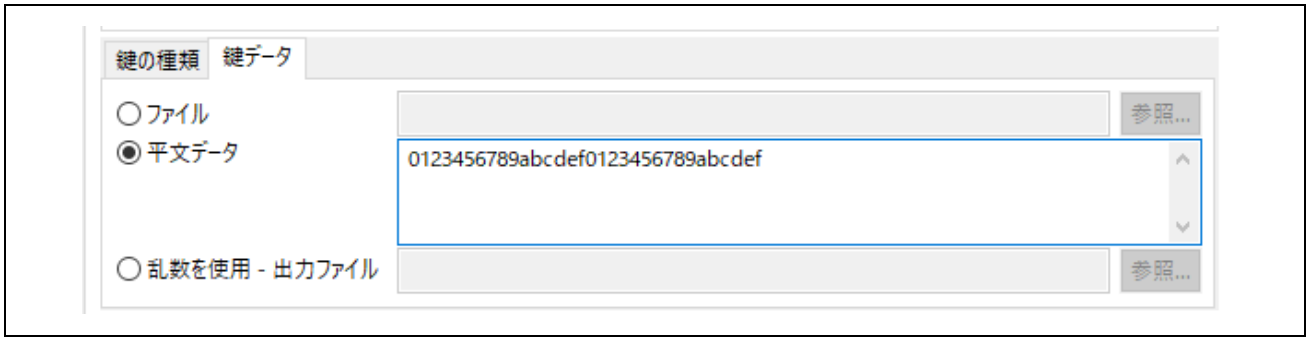

図 6-43 鍵データファイルの生成([鍵のラッピング]タブ-[鍵データ]タブ)

(2) セキュアブートプロジェクトのビルド

セキュアブートプロジェクト(rx*XXX*_bbb_tsip_secure_boot)を、e²studio のワークスペースにインポート後、(1)で生成した euk_aes128.c と euk_aes128.h を rx*XXX*_bbb_tsip_secure_boot プロジェクトの src フォルダの下に置きます。

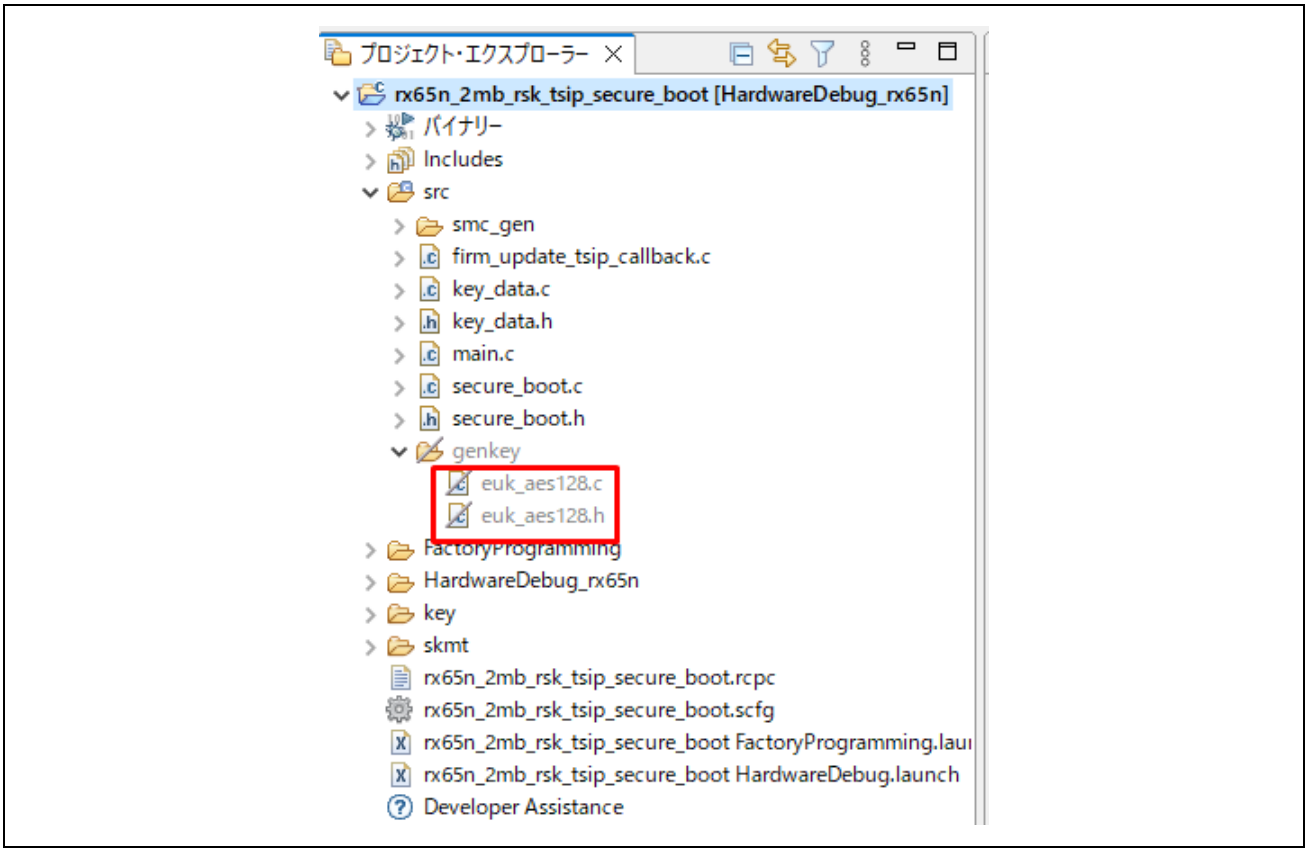

図 6-44 e²studio Project Explorer

また、USB メモリ対応プロジェクトについては、src\smc_gen\r_config 内の r_fwup_config.h 内の

#define FWUP_CFG_COMMUNICATION_FUNCTION

を(2) に設定してください。設定完了後 e²studio でビルドします。

(3) ファームアップデートプロジェクトのビルド

ファームアップデートプロジェクト(rx*XXX*_bbb_tsip_user_program)を、e²studio のワークスペースにインポート後、USB メモリ対応プロジェクトについては(2) 同様に src\smc_gen\r_config 内の r_fwup_config.h 内の

```
#define FWUP_CFG_COMMUNICATION_FUNCTION
```

を(2) に設定してください。設定完了後 e²studio でビルドします。

(4) ファームアップデートプログラムの暗号化

・コマンドライン版を使用される場合
以下コマンドを実行してください。

```
> skmt.exe /enctsip /mode "update" /ver "1" /prg
            "{$skmt_loc}\..\rx65n_2mb_rsk_tsip_user_program\Release_rx65n\rx65n_2mb_rsk_ts
            ip_user_program.mot"
            /enckey "0123456789abcdef0123456789abcdef"
            /startaddr "FFE00300" /endaddr "FFFEFFFF" /imgflg "testing" /filetype "bin"
            /output ""{$skmt_loc}\userprog.rsu"
```

{$skmt_loc}には、実行する Secure Boot プロジェクトフォルダパスを入力してください。

/stardaddr ならびに/endaddr は MCU によって設定値が異なります。下記表の値を入力してください。

表 6-14　各 MCU の選択内容

MCU	パラメータ		アドレス
	GUI	CLI	
RX231 RX66T	開始アドレス	/startaddr	FFFB8300
	終了アドレス	/endaddr	FFFEFFFF
RX72T	開始アドレス	/startaddr	FFF78300
	終了アドレス	/endaddr	FFFEFFFF
RX26T	開始アドレス	/startaddr	FFFC0300
	終了アドレス	/endaddr	FFFEFFFF
RX65N RX671	開始アドレス	/startaddr	FFF00300
	終了アドレス	/endaddr	FFFEFFFF
RX72M RX72N	開始アドレス	/startaddr	FFE00300
	終了アドレス	/endaddr	FFFEFFFF

・スタンドアロン版を使用される場合
以下を入力してください。

-出力イメージ
「Secure Update」を選択

-ファームウェアイメージ
アップデートプロジェクトから出力される mot ファイル(rx*XXX*_*bbb*_tsip_user_program.mot)

-「暗号化アドレス範囲」タブ
以下 MCU ごとに値が異なります。表 6-14　各 MCU の選択内容 の値を入力してください。

-「IV」タブ
「指定値を使用する」を選択し、”55aa55aa55aa55aa55aa55aa55aa55aa”を入力

-「RSU ヘッダ」タブ
イメージフラグ　：TESTING
RSU ヘッダ Ver　：　1

-「出力」
フォーマット : バイナリ
ファイル　　: userprog.rsu

これらの設定はサンプルに付属している Security Key Management Tool の設定保存ファイル rx*XXX*_SecureUpdate.skmt をロードして使用可能です。
rx*XXX*_SecureUpdate.skmt は xml ファイル形式のテキストで編集可能なファイルです。
rx*XXX*_SecureUpdate.skmt 内の{$skmt_loc}に、Secure Boot プロジェクトのフォルダパスを入力してください。

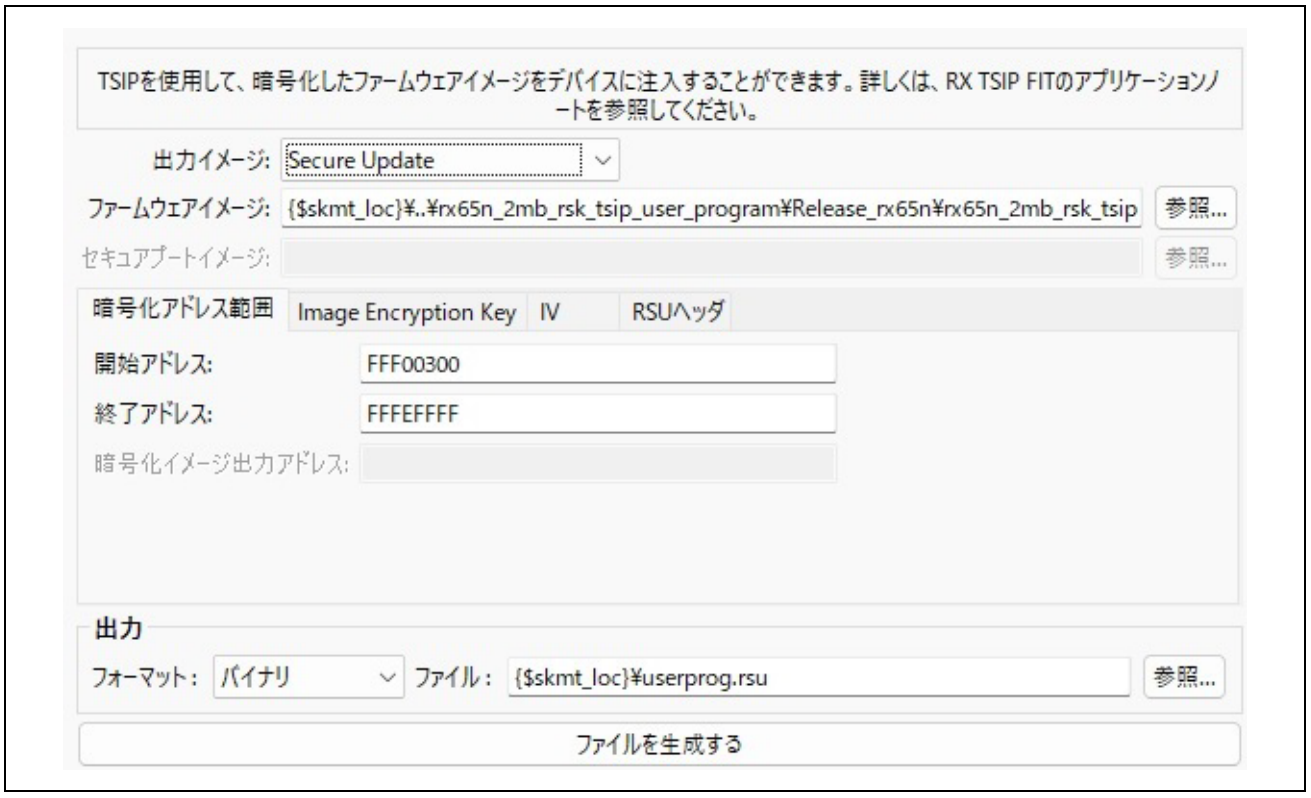

図 6-45 暗号化ファイルの生成([TSIP UPDATE]タブ)

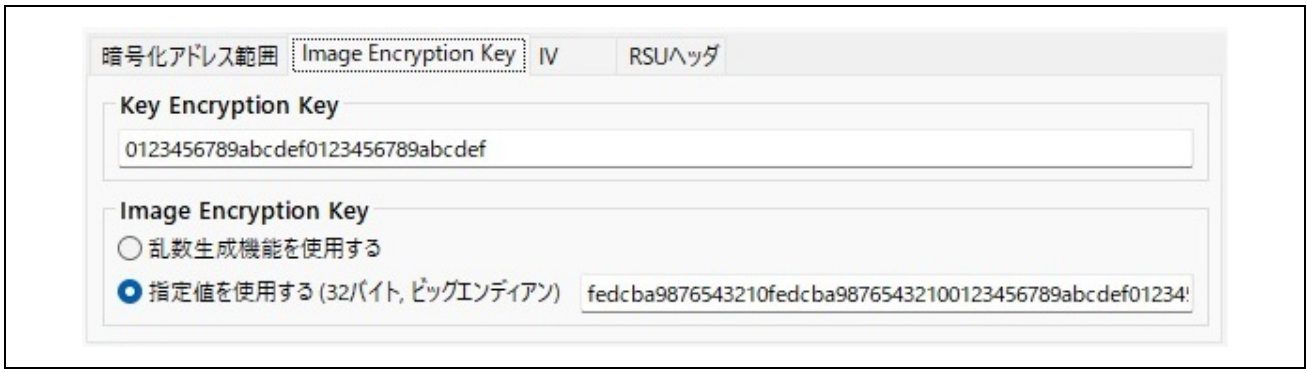

図 6-46 暗号化ファイルの生成([TSIP UPDATE]タブ-[Image Encryption Key]タブ)

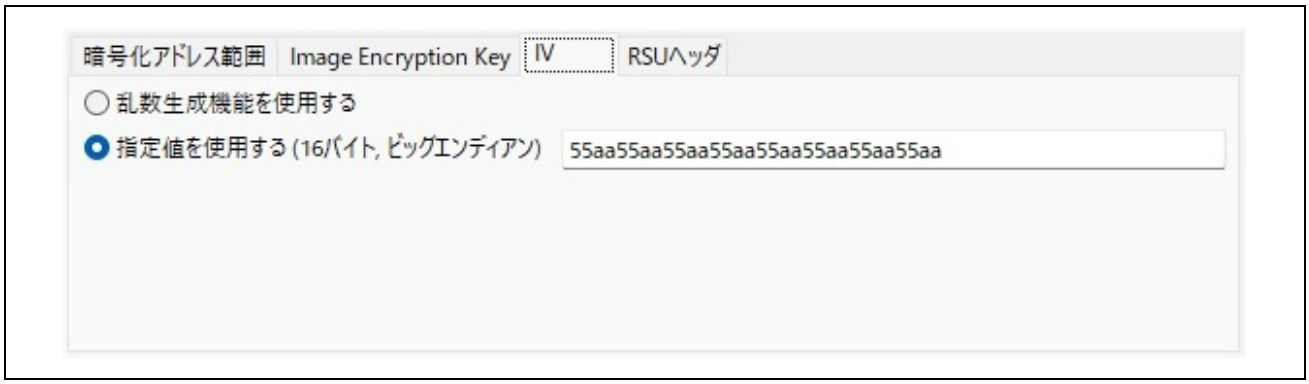

図 6-47 暗号化ファイルの生成([TSIP UPDATE]タブ-[IV]タブ)

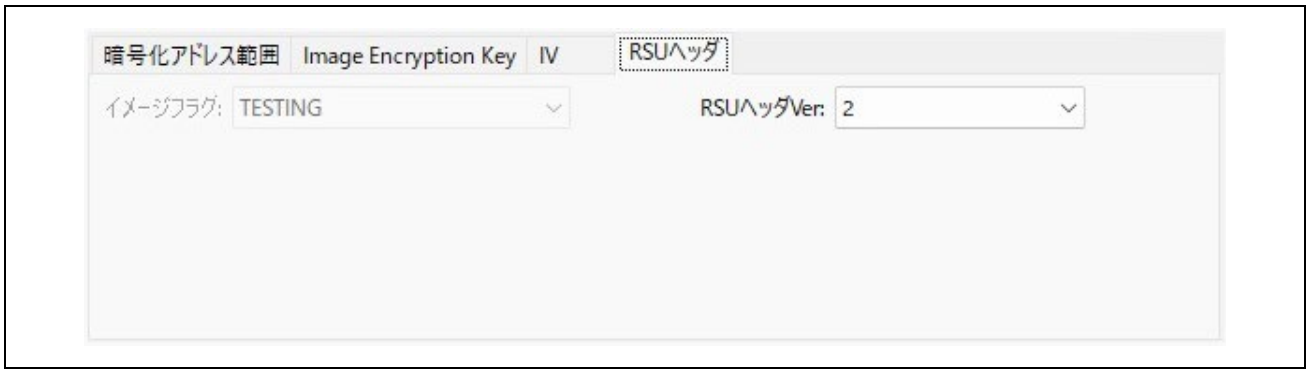

図 6-48 暗号化ファイルの生成([TSIP UPDATE]タブ-[RSU ヘッダ]タブ)

生成したファイル（デフォルトファイル名：userprog.rsu）を以下のように使用します。

・UART の場合、Tera Term のファイル送信を使用してデバイスに送信します。

・USB の場合、USB メモリに格納してボードに接続します。

(5) ターミナルソフト(Tera Term)の起動

6.2.1 デモプロジェクトのセットアップに示したボード接続図の通りに接続をし、Tera Term を起動します。シリアル設定は「表 6-6　ボード RSK RX231, RSK RX66T, RSK RX72T の場合」または「表 6-7 ボード MCB RX26T, RSK RX65N-2MB, RSK RX671, RSK RX72M, RSK RX72N, RX72N Envision Kit の場合」を参照してください。

(6) セキュアブートプロジェクトの実行

e²studio のプロジェクトエクスプローラーで rx*XXX*_*bbb*_tsip_secure_boot を選択後、  ボタンを押して、セキュアブートプロジェクトを実行します。

使用するボードの内蔵 Flash が全消去されていない場合、正常に動かないため、RFP で内蔵 Flash を、「チップ消去」で、全消去してください。

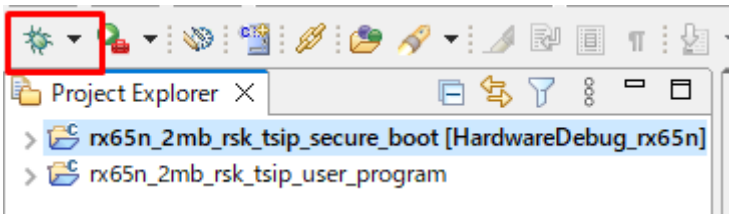

図 6-49　セキュアブートプロジェクトの実行(RX65N RSK 使用時)

(7) 初期ユーザプログラムのインストール

セキュアブートプロジェクトを実行すると暗号化されたユーザプログラムを復号後、ユーザプログラムを実行します。

正常に動作すると Tera Term に以下のようなログが出力されます。

```
HELLO!! this is boot program. ~
$ I was built in May 10 2023, 13:37:17.
Checking flash ROM status.
status = LIFECYCLE_STATE_BLANK
bank info = 1. (start bank = 0)
========== generate user key index phase ==========
generate aes128 user program mac key index: OK
------------------------------------------------
RX65N secure boot program
------------------------------------------------
Checking flash ROM status.
bank 0 status = 0xff [LIFECYCLE_STATE_BLANK]
bank 1 status = 0xff [LIFECYCLE_STATE_BLANK]
bank info = 1. (start bank = 0)
start installing user program.
erase bank1 secure boot mirror area...OK
copy secure boot from bank0 to bank1...OK
erase install area (code flash): OK
send "userprog.rsu" via UART.
```

図 6-50　セキュアブート 初期ユーザプログラムをインストール待ちまでのログ　(RSK RX65N 使用時)

USB の場合は、USB メモリをボードに接続します。

UART の場合は、ファイル > ファイルの送信 で、(3)で暗号化したファームアップデートプログラム(サンプルでは userprog.rsu)を選択します。この時にオプションでバイナリのチェックボックスに、チェックを入れて、”開く”ボタンを押してください。

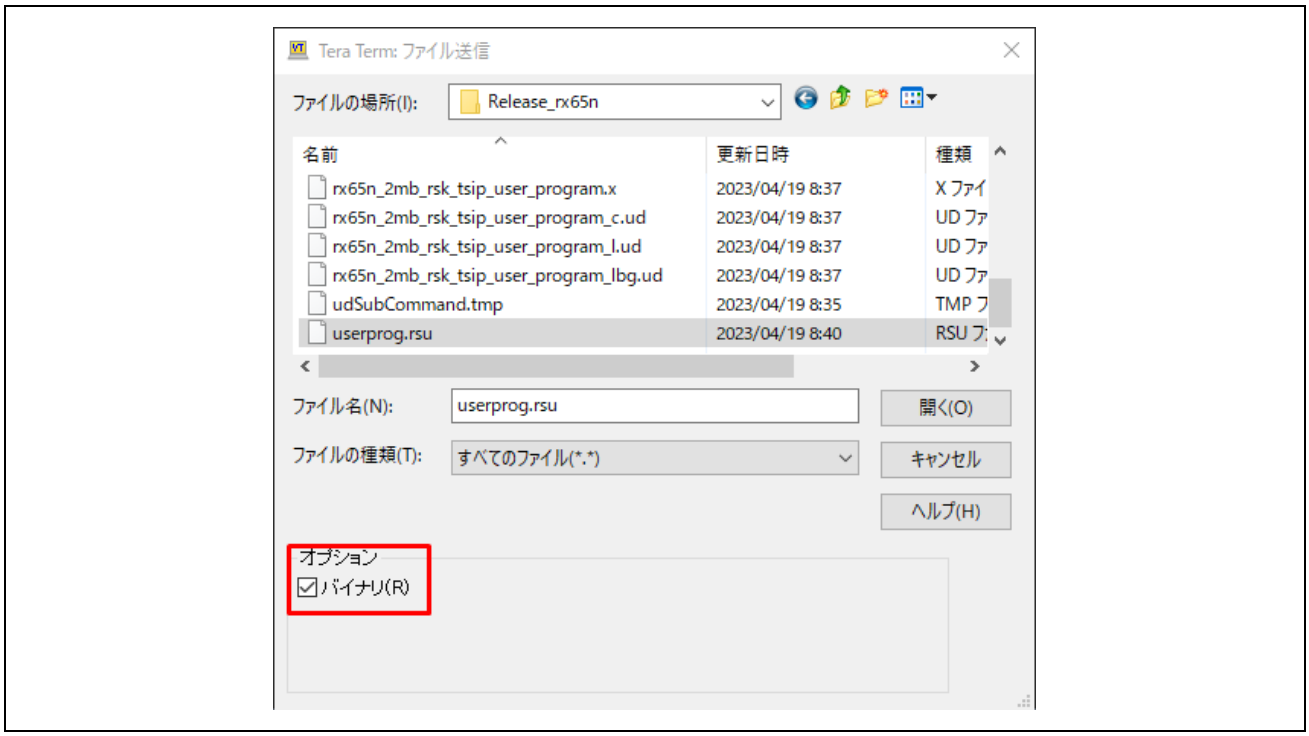

図 6-51　Tera Term ファイル送信ダイアログ

正常に初期ユーザプログラムの書き込みが終わると、Tera Term に以下のようなログが出力されます。

```
  installing firmware...100%(960/960KB).
  completed installing firmware.
  integrity check scheme = mac-aes128-cmac-with-tsip
  bank1(temporary area) on code flash integrity check...transit TSIP status to
UpdateFirmware: OK
  extract update file parameters: OK
  decrypt program:
  100%... OK
  code flash MAC check...OK
  ========== install user key index phase ==========
  erase data flash(main)...OK
  write data flash(main)...OK
  erase data flash(mirror)...OK
  write data flash(mirror)...OK
  data flash setting OK.
  software reset...
  HELLO!! this is boot program. ~
  $ I was built in May 10 2023, 13:37:17.
  Checking flash ROM status.
  status = LIFECYCLE_STATE_UPDATING
  bank info = 1. (start bank = 0)
  update data flash
  erase data flash(main)...OK
  write data flash(main)...OK
  erase data flash(mirror)...OK
  write data flash(mirror)...OK
  data flash setting OK.
  ------------------------------------------------
  RX65N secure boot program
  ------------------------------------------------
  Checking flash ROM status.
  bank 0 status = 0xff [LIFECYCLE_STATE_BLANK]
  bank 1 status = 0xfe [LIFECYCLE_STATE_TESTING]
  bank info = 1. (start bank = 0)
  integrity check scheme = mac-aes128-cmac-with-tsip
  bank1(temporary area) on code flash integrity check...OK
  update LIFECYCLE_STATE from [LIFECYCLE_STATE_TESTING] to [LIFECYCLE_STATE_VALID]
  bank1(temporary area) block0 erase (to update LIFECYCLE_STATE)...OK
  bank1(temporary area) block0 write (to update LIFECYCLE_STATE)...OK
  swap bank...[FWUP_STATE_BANK1_UPDATE_LIFECYCLE_WRITE_COMPLETE and
LIFECYCLE_STATE_VALID]
  HELLO!! this is boot program. ~
  $ I was built in May 10 2023, 13:37:17.
  Checking flash ROM status.
  status = LIFECYCLE_STATE_ON_THE_MARKET
  bank info = 0. (start bank = 1)
  ------------------------------------------------
  RX65N secure boot program
  ------------------------------------------------
  Checking flash ROM status.
  bank 0 status = 0xf8 [LIFECYCLE_STATE_VALID]
  bank 1 status = 0xff [LIFECYCLE_STATE_BLANK]
  bank info = 0. (start bank = 1)
  integrity check scheme = mac-aes128-cmac-with-tsip
  bank0(execute area) on code flash integrity check...OK
  erase install area (code flash): OK
  data flash(main) hash check...OK
  data flash(mirror) hash check...OK
  secure boot sequence: success.
  HELLO!! this is user program. ~
  $ I was built in May  9 2023, 17:19:52.
  Version ver 1.00.
  tsip@rx65n
  $
```

図 6-52　セキュアブート 初期ファームをインストールのログ　(RSK RX65N 使用時)

(8) ファームアップデート動作

　続いて、ユーザプログラムプロジェクトの内容を一部変更し、ファームウェアアップデートを行います。ユーザプログラムプロジェクト main.c にあるユーザプログラムのバージョン情報を、”ver 1.00”から”ver 1.01”にしてください。ビルド実行後出来上がった MOT ファイルを(3)の手順でビルド後、(4)の手順で暗号化してください。

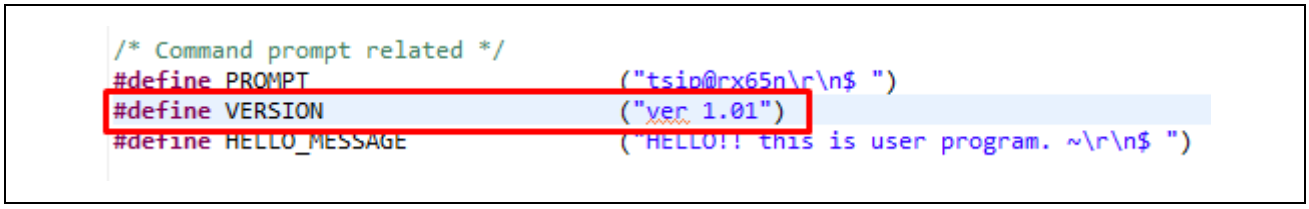

図 6-53　main.c 変更箇所(rx65n_2mb_rsk_tsip_user_program)

update コマンドを実行します。(7)の手順同様で、Ver 1.01 に更新したユーザプログラムをインストールします。アップロード後にシステムがリブートします。

```
tsip@rx65n
$
install start ...
[INFO] Receive file created.
[INFO] ------------------------------------------------
[INFO] Firmware update user program
[INFO] ------------------------------------------------
[INFO] Send Update firmware via UART.
[INFO] Flash Write: Address = 0xFFE00000, length = 1024byte ... OK
[INFO] Flash Write: Address = 0xFFE00400, length = 1024byte ... OK
 --- (中略) ---
[INFO] Flash Write: Address = 0xFFEEF400, length = 1024byte ... OK
[INFO] Flash Write: Address = 0xFFEEF800, length = 1024byte ... OK
[INFO] Flash Write: Address = 0xFFEEFC00, length = 1024byte ... OK
[INFO] Firmware update to Flash is complete.
[INFO] Firmware update to Flash is complete.
[INFO] Check signature of update firmware is complete.
[INFO] Testing.
[FWUP_main] Firmware update completely.
transit TSIP status to UpdateFirmware: OK
extract update file parameters: OK
decrypt program:
: 100%... OK
code flash MAC check...OK
 100 % 100 % ... install complete
HELLO!! this is boot program. ~
$ I was built in May 10 2023, 13:37:17.
Checking flash ROM status.
status = LIFECYCLE_STATE_UPDATING
bank info = 0. (start bank = 1)
update data flash
erase data flash(main)...OK
write data flash(main)...OK
erase data flash(mirror)...OK
write data flash(mirror)...OK
data flash setting OK.
------------------------------------------------
RX65N secure boot program
------------------------------------------------
Checking flash ROM status.
bank 0 status = 0xf8 [LIFECYCLE_STATE_VALID]
bank 1 status = 0xfe [LIFECYCLE_STATE_TESTING]
bank info = 0. (start bank = 1)
integrity check scheme = mac-aes128-cmac-with-tsip
bank1(temporary area) on code flash integrity check...OK
update LIFECYCLE_STATE from [LIFECYCLE_STATE_TESTING] to [LIFECYCLE_STATE_VALID]
bank1(temporary area) block0 erase (to update LIFECYCLE_STATE)...OK
bank1(temporary area) block0 write (to update LIFECYCLE_STATE)...OK
swap bank...[FWUP_STATE_BANK1_UPDATE_LIFECYCLE_WRITE_COMPLETE and
LIFECYCLE_STATE_VALID]
HELLO!! this is boot program. ~
$ I was built in May 10 2023, 13:37:17.
Checking flash ROM status.
status = LIFECYCLE_STATE_ON_THE_MARKET
bank info = 1. (start bank = 0)
------------------------------------------------
RX65N secure boot program
------------------------------------------------
Checking flash ROM status.
bank 0 status = 0xf8 [LIFECYCLE_STATE_VALID]
bank 1 status = 0xf8 [LIFECYCLE_STATE_VALID]
bank info = 1. (start bank = 0)
integrity check scheme = mac-aes128-cmac-with-tsip
bank0(execute area) on code flash integrity check...OK
erase install area (code flash): OK
data flash(main) hash check...OK
data flash(mirror) hash check...OK
secure boot sequence: success.
□HELLO!! this is user program. ~
$ I was built in May  9 2023, 17:19:52.
Version ver 1.01.
tsip@rx65n
$
```

図 6-54　ファームアップデート時のログ　(RSK RX65N 使用時)

6.2.4　セキュアブートと暗号化されたユーザプログラムを合わせた Motorola S format ファイルを使用した工場書き込み実行例

本章では、セキュアブートと暗号化されたユーザプログラムを合わせた Motorola S format ファイルの作成方法ならびに、Renesas Flash Programmer(RFP)を使った書き込み方法(Factory Programming)を説明します。

(1) ファームアップデート用暗号鍵ファイルの生成

「6.2.3 セキュアブート・ファームアップデートの実行例」と同様の手順で生成します。

(2) セキュアブートプロジェクトのビルド

セキュアブートプロジェクト(rx*XXX*_bbb_tsip_secure_boot)を、e²studio のワークスペースにインポート後、(1)で生成した euk_aes128.c と euk_aes128.h を rx*XXX*_bbb_tsip_secure_boot プロジェクトの src フォルダの下に置きます。

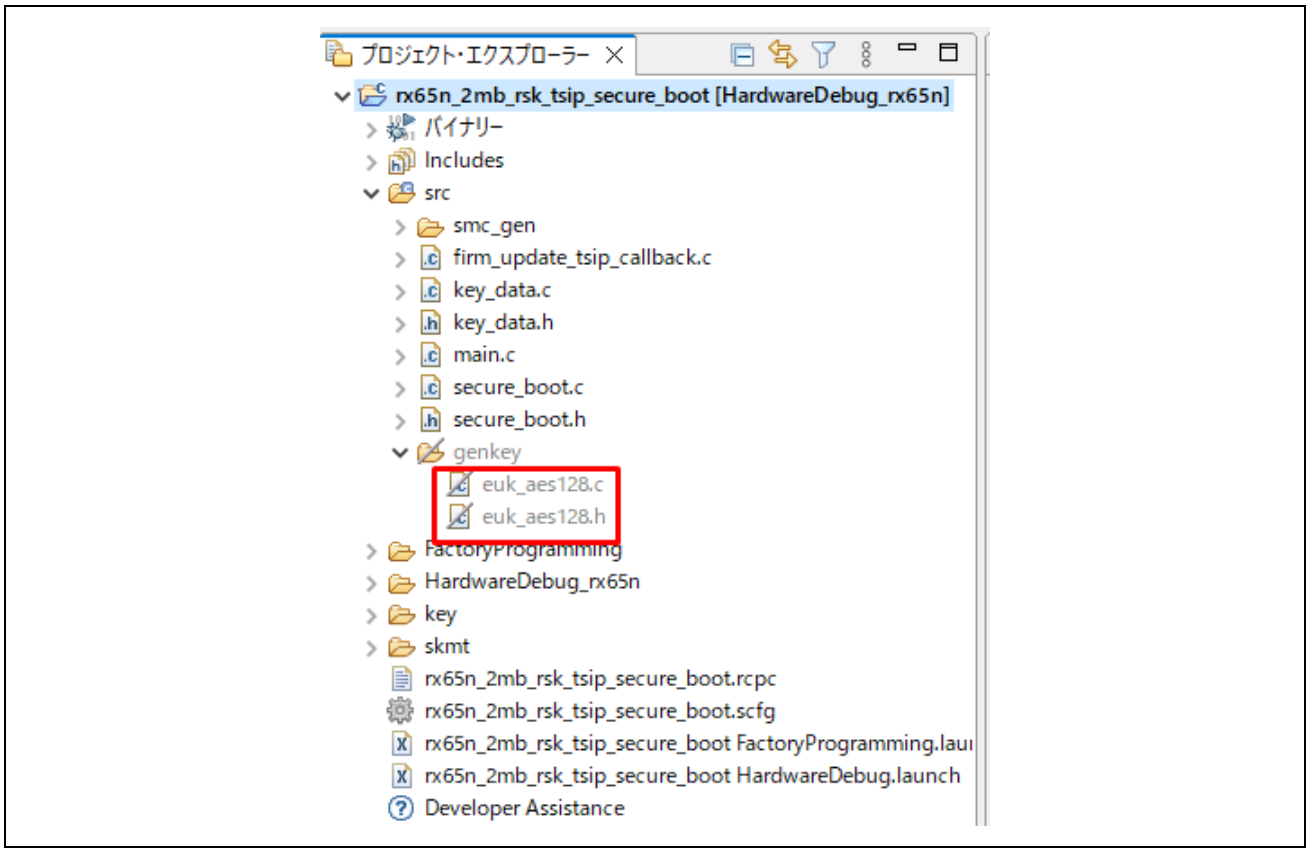

図 6-55 e²studio Project Explorer

「ビルド構成」を、”Factory Programming”の構成に変更します。e²studio のプロジェクトエクスプローラーからセキュアアップデートプロジェクトを選択し、右クリック -> ビルド構成 -> アクティブにする で “Factory Programming”を選択します。

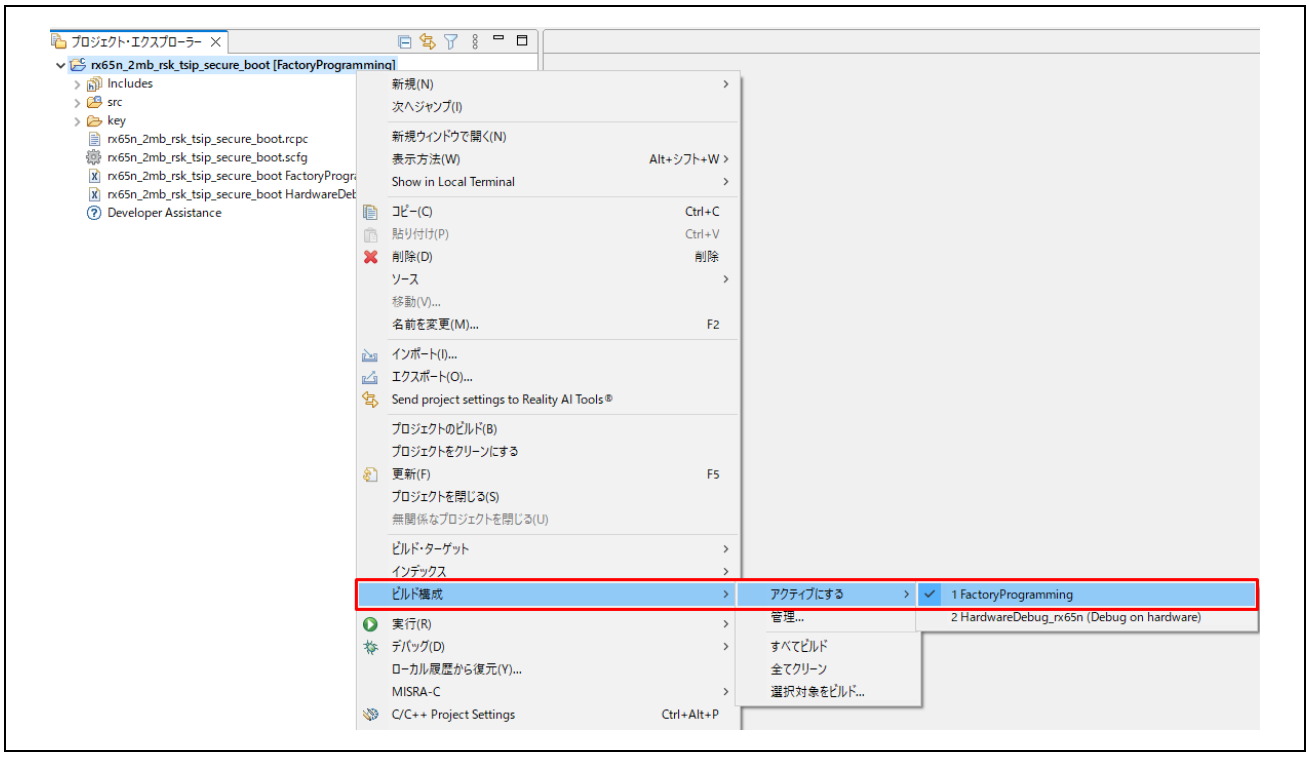

図 6-56　ビルド構成 Factory Programming への変更

「ビルド構成」を「Factory Programming」へ変更後、セキュアブートプロジェクトをビルドしてください。

ビルド構成：HardwareDebug からは、ビルドマクロ FACTORY_PROGRAMMING の定義が追加されています。またセクション情報が異なります。

(3) ファームアップデートプロジェクトのビルド

「6.2.3 セキュアブート・ファームアップデートの実行例」と同様の手順でビルドします。

(4) ファームアップデートプログラムの暗号化

・コマンドライン版を使用される場合
　以下コマンドを実行してください。

```
> skmt.exe /enctsip /mode "factory" /ver "1" /prg
          "{$skmt_loc}\..\rx65n_2mb_rsk_tsip_user_program\Release_rx65n\rx65n_2mb_rsk_ts
          ip_user_program.mot"
          /prg_sb "{$skmt_loc}\FactoryProgramming\rx65n_2mb_rsk_tsip_secure_boot.mot "
          /enckey "0123456789abcdef0123456789abcdef"
          /startaddr "FFE00300" /endaddr "FFFEFFFF" /destaddr "FFE00300"
          /imgflg "valid" /filetype "mot"
          /output ""{$skmt_loc}\userprog.mot"
```

{$skmt_loc}には、実行する Secure Boot プロジェクトフォルダパスを入力してください。

/stardaddr、/endaddr ならびに/destaddr は MCU によって設定値が異なります。下記表の値を入力してください。

表 6-15　各 MCU の選択内容

MCU	パラメータ		アドレス
	GUI	CLI	
RX231 RX66T	開始アドレス	/startaddr	FFFB8300
	終了アドレス	/endaddr	FFFEFFFF
	暗号化イメージ出力アドレス	/destaddr	FFFB8300
RX72T	開始アドレス	/startaddr	FFF78300
	終了アドレス	/endaddr	FFFEFFFF
	暗号化イメージ出力アドレス	/destaddr	FFF78300
RX26T	開始アドレス	/startaddr	FFFC0300
	終了アドレス	/endaddr	FFFEFFFF
	暗号化イメージ出力アドレス	/destaddr	FFFC0300
RX65N RX671	開始アドレス	/startaddr	FFF00300
	終了アドレス	/endaddr	FFFEFFFF
	暗号化イメージ出力アドレス	/destaddr	FFF00300
RX72M RX72N	開始アドレス	/startaddr	FFE00300
	終了アドレス	/endaddr	FFFEFFFF
	暗号化イメージ出力アドレス	/destaddr	FFE00300

・スタンドアロン版を使用される場合
　以下を入力してください。

-出力イメージ
　「Secure Update」を選択

-ファームウェアイメージ
　アップデートプロジェクトから出力される mot ファイル(rx*XXX*_*bbb*_tsip_user_program.mot)

-「暗号化アドレス範囲」タブ
　以下 MCU ごとに設定値が異なります。表 6-15　各 MCU の選択内容 の値を入力してください。

-「IV」タブ
　「指定値を使用する」を選択し、”55aa55aa55aa55aa55aa55aa55aa55aa”を入力

-「RSU ヘッダ」タブ
　イメージフラグ　：TESTING
　RSU ヘッダ Ver　：　1

-「出力」
　　フォーマット : バイナリ
　　ファイル　　: userprog.mot

これらの設定はサンプルに付属している Security Key Management Tool の設定保存ファイル rx*XXX*_SecureUpdate.skmt をロードして使用可能です。
rx*XXX*_SecureUpdate.skmt は xml ファイル形式のテキストで編集可能なファイルです。
rx*XXX*_SecureUpdate.skmt 内の{$skmt_loc}に、Secure Boot プロジェクトのフォルダパスを入力してください。

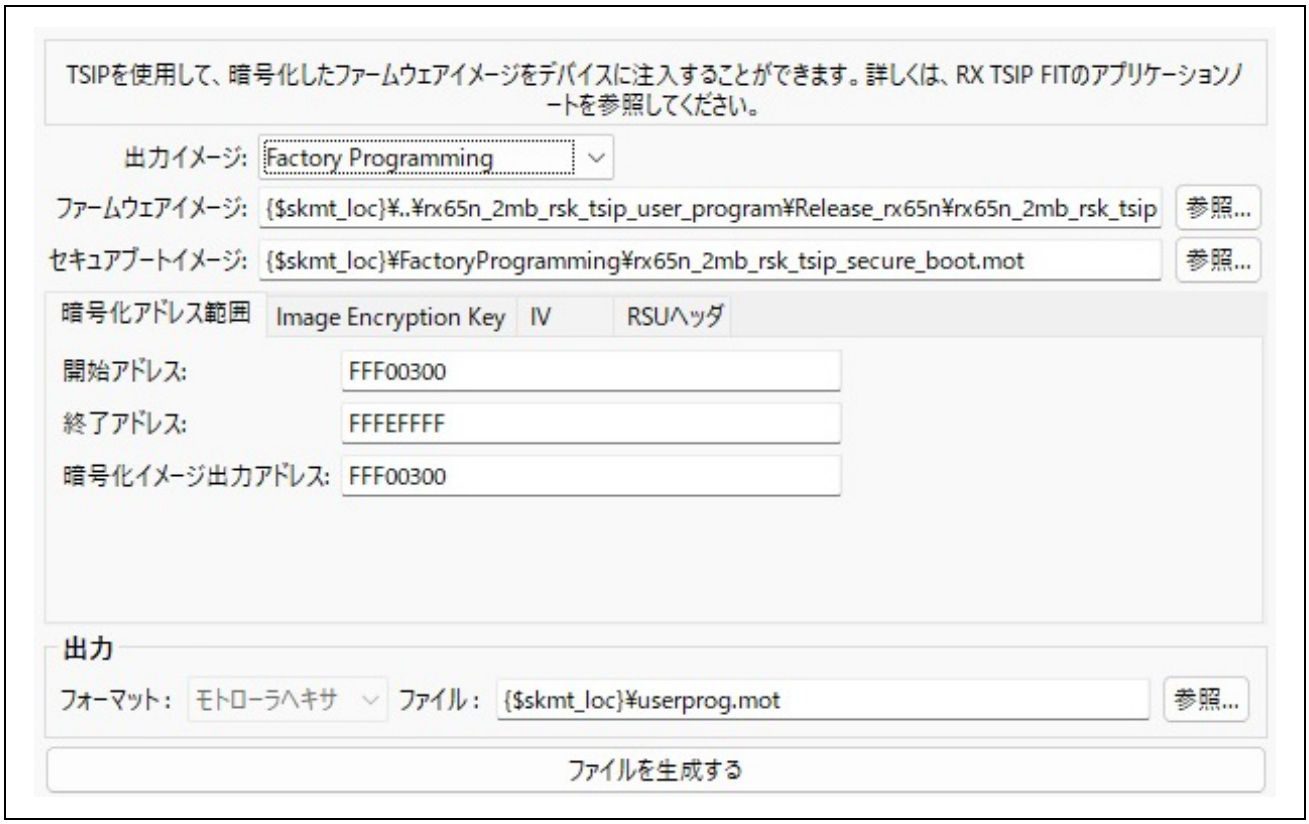

図 6-57 暗号化ファイルの生成([TSIP UPDATE]タブ)

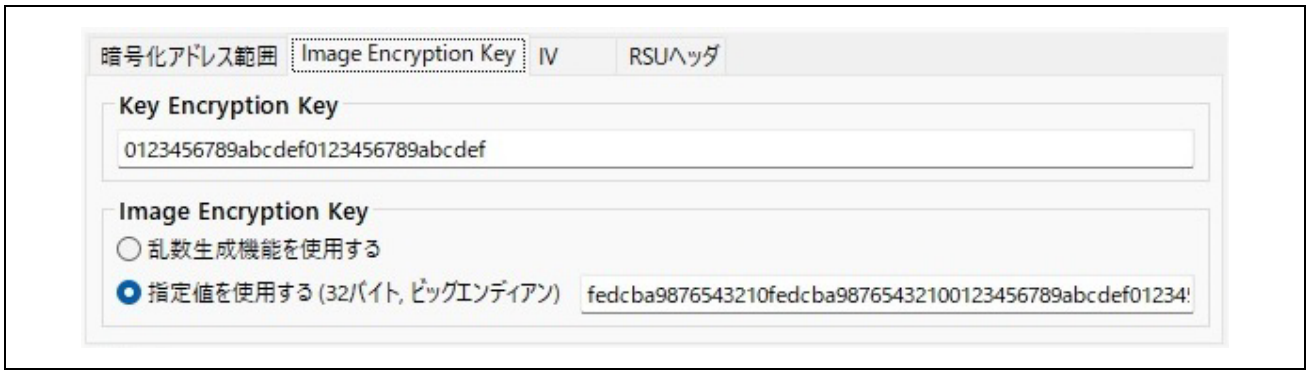

図 6-58 暗号化ファイルの生成([TSIP UPDATE]タブ-[Image Encryption Key]タブ)

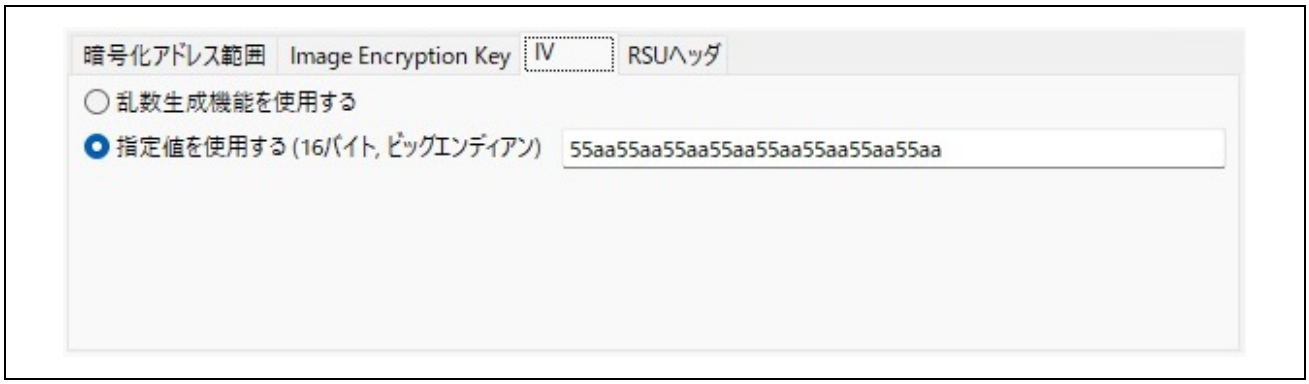

図 6-59 暗号化ファイルの生成([TSIP UPDATE]タブ-[IV]タブ)

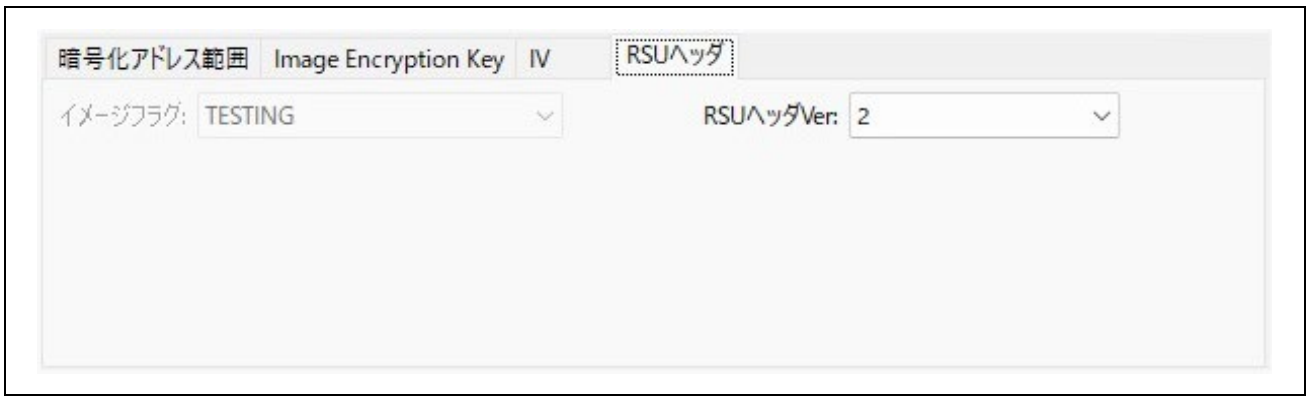

図 6-60 暗号化ファイルの生成([TSIP UPDATE]タブ-[RSU ヘッダ]タブ)

暗号化したユーザプログラム＋平文のセキュアブートの Motorola S format ファイル（デフォルトファイル名：userprog.mot）を、RFP を使ってデバイスに書き込みます。

(5) ターミナルソフト(Tera Term)の起動

RFP を使った書き込み前に、6.2.1 デモプロジェクトのセットアップに示したボード接続図の通りに接続をし、Tera Term を起動します。

(6) RFP を使用した書き込み

RFP を起動し、プロジェクト作成します。プロジェクト作成後、接続設定タブ > ツールの詳細ボタン > リセット設定タブ　切断時のリセット端子を Hi-Z に設定してください。

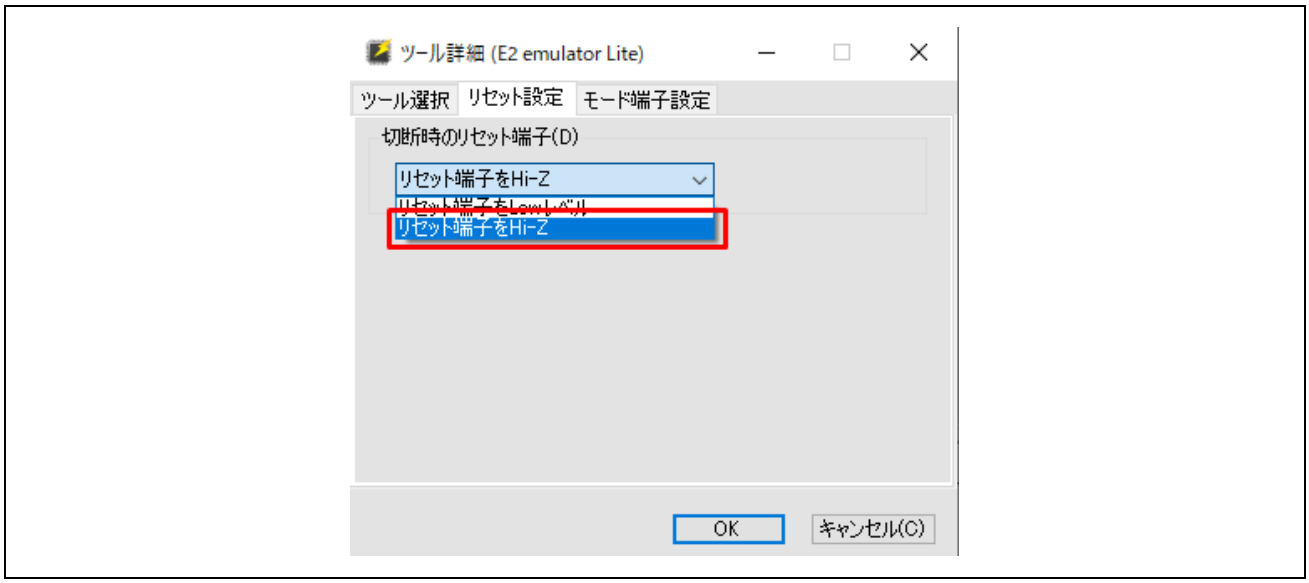

図 6-61　RFP リセット設定タブ

RFP を使用して、書き込みが正常に行われると、デバイスはセキュアブートプログラムを実行して、暗号化されたユーザプログラムを復号後、ユーザプログラムを実行します。

正常に動作すると以下のようなログが出力されます。

```
HELLO!! this is boot program. ~
$ I was built in May  9 2023, 17:15:53.
Checking flash ROM status.
status = LIFECYCLE_STATE_BLANK
bank info = 1. (start bank = 0)
========== generate user key index phase ==========
generate aes128 user program mac key index: OK
Copy Secure Boot...OK
Decrypt User Program.
transit TSIP status to UpdateFirmware: OK
extract update file parameters: OK
decrypt program:
100%... OK
code flash MAC check...OK
========== install user key index phase ==========
erase data flash(main)...OK
write data flash(main)...OK
erase data flash(mirror)...OK
write data flash(mirror)...OK
data flash setting OK.
software reset...
HELLO!! this is boot program. ~
$ I was built in May  9 2023, 17:15:53.
Checking flash ROM status.
status = LIFECYCLE_STATE_UPDATING
bank info = 1. (start bank = 0)
update data flash
erase data flash(main)...OK
write data flash(main)...OK
erase data flash(mirror)...OK
write data flash(mirror)...OK
data flash setting OK.
```

図 6-62　RFP を使用した暗号ユーザプログラム書き込み時のログ(1)　(RSK RX65N 使用時)

```
------------------------------------------------
RX65N secure boot program
------------------------------------------------
Checking flash ROM status.
bank 0 status = 0xf8 [LIFECYCLE_STATE_VALID]
bank 1 status = 0xfe [LIFECYCLE_STATE_TESTING]
bank info = 1. (start bank = 0)
integrity check scheme = mac-aes128-cmac-with-tsip
bank1(temporary area) on code flash integrity check...OK
update LIFECYCLE_STATE from [LIFECYCLE_STATE_TESTING] to [LIFECYCLE_STATE_VALID]
bank1(temporary area) block0 erase (to update LIFECYCLE_STATE)...OK
bank1(temporary area) block0 write (to update LIFECYCLE_STATE)...OK
swap bank...[FWUP_STATE_BANK1_UPDATE_LIFECYCLE_WRITE_COMPLETE and
LIFECYCLE_STATE_VALID]
HELLO!! this is boot program. ~
$ I was built in May  9 2023, 17:15:53.
Checking flash ROM status.
status = LIFECYCLE_STATE_ON_THE_MARKET
bank info = 0. (start bank = 1)
------------------------------------------------
RX65N secure boot program
------------------------------------------------
Checking flash ROM status.
bank 0 status = 0xf8 [LIFECYCLE_STATE_VALID]
bank 1 status = 0xf8 [LIFECYCLE_STATE_VALID]
bank info = 0. (start bank = 1)
integrity check scheme = mac-aes128-cmac-with-tsip
bank0(execute area) on code flash integrity check...OK
erase install area (code flash): OK
data flash(main) hash check...OK
data flash(mirror) hash check...OK
secure boot sequence: success.
□HELLO!! this is user program. ~
$ I was built in May 10 2023, 13:52:34.
Version ver 1.00.
tsip@rx65n
$
```

図 6-63　RFP を使用した暗号ユーザプログラム書き込み時のログ(2)　(RSK RX65N 使用時)

(7) ファームアップデート動作

ファームアップデート動作は、「6.2.3 セキュアブート・ファームアップデートの実行例」の(8) ファームアップデート動作をご参照ください。

6.2.5　ユーザプログラムプロジェクトのデバッグについて

ユーザプログラムのプロジェクトは、セキュアブート経由で起動するため、プロジェクトのリセットベクタ(RESETVECT)を 0xFFF7FFFC 番地に置いています。このため、ユーザプログラムプロジェクトをそのままデバッグすることができません。ユーザプログラムプロジェクトをデバッグする場合は、表 6-16　各ユーザプログラムのデバッグ用構成に示すデバッグ用の構成に切り替えて、デバッグを行ってください。

表 6-16　各ユーザプログラムのデバッグ用構成

ボード	ユーザプログラム	デバッグ用構成
RSK RX231	rx231_rsk_tsip_user_program	HardwareDebug_rx231
MCB RX26T	rx26t_mcb_tsip_sci_user_program	HardwareDebug_rx26t
RSK RX65N	rx65n_2mb_rsk_tsip_user_program	HardwareDebug_rx65n
RSK RX66T	rx66t_rsk_tsip_sci_user_program	HardwareDebug_rx66t
RSK RX671	rx671_rsk_tsip_user_program	HardwareDebug_rx671
RSK RX72M	rx72m_rsk_tsip_user_program	HardwareDebug_rx72m
RSK RX72N	rx72n_rsk_tsip_user_program	HardwareDebug_rx72n
RX72N Envision Kit	rx72n_ek_tsip_user_program	HardwareDebug_rx72n
RSK RX72T	rx72t_rsk_tsip_user_program	HardwareDebug_rx72t

- デバッガ接続構成の切り替え方法：

1.e²studio メニュービルドボタン  の横の▼を押してください。
[HardwareDebug_rx231(Debug on hardware)]を選択すると、ビルドが開始します。

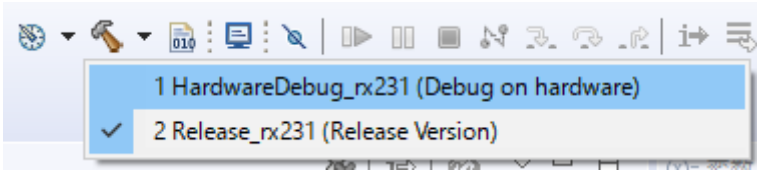

図 6-64　e²studio プロジェクトメニュー(RX231 の場合)

2.ビルドが完了するとデバッガによるデバッグが可能になります。

※リリース用の構成「Release_rx*XXX*」とデバッグ用の構成「HardwareDebug_rx*XXX*」の違い
各ユーザプログラムプロジェクトの構成「Release_rx*XXX*」と「HardwareDebug_rx*XXX*」ではビルド時のリセットベクタ(RESETVECT)と割り込みベクタ(EXCEPTVECT)のアドレスが異なります。

表 6-17　ユーザプログラムプロジェクト 各構成のベクタ設定アドレス

シンボル	Release_rx*XXX*	HardwareDebug_rx*XXX*
RESETVECT	0xFFFEFFFC	0xFFFFFFFC
EXCEPTVECT	0xFFFEFF80	0xFFFFFF80

6.2.6　セキュアブートからユーザプログラムへの遷移時の注意事項

セキュアブートプログラムからユーザプログラムへの遷移時には、セキュアブートプログラムの周辺機能の設定がアプリケーションに引き継がれることになります。そこで、ブートローダのサンプルプログラムは以下の通り実装しています。

セキュアブートプログラムで使用するFITモジュール（SCI、Flash、TSIP、USB）に関しては、ブートローダ終了時にAPI関数をcloseした状態にします。また、その他の設定に関してはスマート・コンフィグレータを使用した際の初期値となります。

お客様にて、セキュアブートプログラムのサンプルプログラムを改造して使用される場合は、セキュアブートプログラムにて設定した周辺機能の設定がアプリケーション側に引き継がれる事になりますので、セキュアブートプログラムからユーザプログラムに遷移する前に周辺機能の設定を初期化するか、アプリケーション側と周辺機能の設定を共通化されることを推奨します。

なお、アプリケーションを作成される際も、セキュアブートプログラムの実装を考慮して開発頂きますようお願いします。

6.2.7　デモプロジェクトの性能情報

セキュアブートのデモプロジェクトの性能情報を示します。
性能はコアクロックである ICLK のサイクル単位での計測になります。ドライバは CC-RX、最適化レベル 2 でビルド、その他の条件は下記の通りです。

モジュールリビジョン: r_tsip_rx rev1.21

コンパイラバージョン: Renesas Electronics C/C++ Compiler Package for RX Family V3.06.00

(統合開発環境のデフォルト設定に”-lang = c99”オプションを追加)

本アプリケーションノートのパッケージに含まれるセキュアブートのデモプロジェクトの処理性能として、リセットからファームウェアの MAC 検証までにかかるサイクル数を表 6-18 に示します。

表 6-18 セキュアブートのデモプロジェクトの処理性能（単位：cycle）

ボード	処理性能 （リセット～MAC 検証）
RSK RX231	300,000,000
MCB RX26T	280,000,000
RSK RX65N	280,000,000
RSK RX66T	380,000,000
RSK RX671	280,000,000
RSK RX72M	560,000,000
RSK RX72N	560,000,000
RX72N Envision Kit	560,000,000
RSK RX72T	450,000,000

7. 付録

7.1 動作確認環境

本ドライバの動作確認環境を以下に示します。

表 7-1　動作確認環境

項目	内容
統合開発環境	ルネサスエレクトロニクス製 e^2 studio 2024-04 IAR Embedded Workbench for Renesas RX 5.10.01
C コンパイラ	ルネサスエレクトロニクス製 C/C++ Compiler for RX Family(CC-RX) V3.06.00 コンパイルオプション：統合開発環境のデフォルト設定に以下のオプションを追加 -lang = c99
	GCC for Renesas RX 8.3.0.202311 コンパイルオプション：統合開発環境のデフォルト設定に以下のオプションを追加 -std = gnu99
	IAR C/C++ Compiler for Renesas RX version 5.10.01 コンパイルオプション：統合開発環境のデフォルト設定
エンディアン	ビッグエンディアン/リトルエンディアン
モジュールのバージョン	Ver.1.21
使用ボード	Renesas Starter Kit for RX231(B 版) (型名：R0K505231S020BE) Renesas Solution Starter Kit for RX23W(TSIP 搭載) (型名：RTK5523W8BC00001BJ) MCB-RX26T Type B (型名：RTK0EMXE70C02000BJ) Renesas Starter Kit+ for RX65N-2MB(TSIP 搭載) (型名：RTK50565N2S10010BE) Renesas Starter Kit for RX66T(TSIP 搭載) (型名：RTK50566T0S00010BE) Renesas Starter Kit+ for RX671 (型名：RTK55671xxxxxxxxxx) Renesas Starter Kit+ for RX72M(TSIP 搭載) (型名：RTK5572MNHSxxxxxxx) Renesas Starter Kit+ for RX72N(TSIP 搭載) (型名：RTK5572NNHCxxxxxxx) Renesas Starter Kit for RX72T(TSIP 搭載) (型名：RTK5572TKCS00010BE)

7.2　トラブルシューティング

(1) Q：本 FIT モジュールをプロジェクトに追加しましたが、ビルド実行すると「Could not open source file "platform.h"」エラーが発生します。

A：FIT モジュールがプロジェクトに正しく追加されていない可能性があります。プロジェクトへの追加方法をご確認ください。

- CS+を使用している場合
  アプリケーションノート「RX ファミリ CS+に組み込む方法 Firmware Integration Technology (R01AN1826)」
- e^2 studio を使用している場合
  アプリケーションノート「RX ファミリ e^2 studio に組み込む方法 Firmware Integration Technology (R01AN1723)」

また、本 FIT モジュールを使用する場合、ボードサポートパッケージ FIT モジュール(BSP モジュール)もプロジェクトに追加する必要があります。BSP モジュールの追加方法は、アプリケーションノート「ボードサポートパッケージモジュール(R01AN1685)」を参照してください。

(2) Q：FITDemos の e^{2}studio サンプルプロジェクトを CS+で使用したい。

A：以下の web サイトを参照してください。
「e^{2}studio から CS+への移行方法」
> 「既存のプロジェクトを変換して CS+の新規プロジェクトを作成」
https://www.renesas.com/jp/ja/products/software-tools/tools/migration-tools/migration-e2studio-to-csplus.html

【注意】：手順 5 で
「変換直後のプロジェクト構成ファイルをまとめてバックアップする(C)」
チェックが入っている場合に、[Q0268002]ダイアログが出る場合があります。
[Q0268002]ダイアログで [はい]ボタンを押した場合、コンパイラのインクルード・パスを設定しなおす必要があります。

7.3　ユーザ鍵暗号化フォーマット

ユーザ鍵を注入するときに UFPK と IV を使って、もしくは鍵の更新時に KUK と iv を使用してユーザ鍵をラップします。その時に、ラップする鍵データのフォーマットは暗号アルゴリズムによって異なります。本章では、暗号化するユーザ鍵のデータフォーマット(User Key)と、ラップされた鍵(Encrypted User Key)のデータフォーマットを示します。

暗号化方法に関しては 3.7.1 鍵の注入と更新をご参照ください。

7.3.1　AES

7.3.1.1　AES 128bit 鍵

入力(User Key)

byte	16			
	4	4	4	4
0-15	128 bit AES 鍵			

出力(Encrypted Key)

byte	16			
	4	4	4	4
0-15	encrypted_user_key(128bit AES 鍵)			
16-31	MAC			

7.3.1.2　AES 256bit 鍵

入力(User Key)

byte	16			
	4	4	4	4
0-31	256 bit AES 鍵			

出力(Encrypted Key)

byte	16			
	4	4	4	4
0-31	encrypted_user_key(256bit AES 鍵)			
32-47	MAC			

7.3.2　DES

入力(User Key)

byte	16			
	4	4	4	4
0-7	56bit DES key with odd parity1【注】			
8-15	56bit DES key with odd parity2【注】			
16-23	56bit DES key with odd parity3【注】			

出力(Encrypted Key)

byte	16			
	4	4	4	4
0-23	encrypted_user_key(56bit DES key with odd parity1 || 56bit DES key with odd parity2 || 56bit DES key with odd parity3)			
24-39	MAC			

注： 鍵データ 7bit 毎に奇数パリティをつけてください。
例 DES key data = 0x00000000000000 → 0x0101010101010101
DES key data = 0xFFFFFFFFFFFFFF → 0xFEFEFEFEFEFEFEFE

2-DES の場合は 56bit DES key with odd parity1 と 56bit DES key with odd parity3 に同じ鍵を入れてください。

DES の場合 56bit DES key with odd parity1, 56bit DES key with odd parity2, 56bit DES key with odd parity3 すべて同じ値を入れてください。

7.3.3　ARC4

入力(User Key)

byte	16			
	4	4	4	4
0-255	ARC4			

出力(Encrypted Key)

byte	16			
	4	4	4	4
0-255	encrypted_user_key(ARC4)			
256-272	MAC			

7.3.4　RSA

7.3.4.1　RSA 1024bit 鍵

(1) 公開鍵

入力(User Key)

byte	16			
	4	4	4	4
0-127	RSA 1024bit 公開鍵 n			
128-143	RSA 1024bit 公開鍵 e	0 padding		

出力(Encrypted Key)

byte	16			
	4	4	4	4
0-143	encrypted_user_key(RSA 1024bit 公開鍵 n || e || 0 padding)			
144-159	MAC			

(2) 秘密鍵

入力(User Key)

byte	16			
	4	4	4	4
0-127	RSA 1024bit 公開鍵 n			
128-255	RSA 1024bit 秘密鍵 d			

出力(Encrypted Key)

byte	16			
	4	4	4	4
0-255	encrypted_user_key(RSA 1024bit 公開鍵 n || 秘密鍵 d)			
256-271	MAC			

7.3.4.2　RSA 2048bit 鍵

(1) 公開鍵

入力(User Key)

byte	16			
	4	4	4	4
0-255	RSA 2048bit 公開鍵 n			
256-271	RSA 2048bit 公開鍵 e	0 padding		

出力(Encrypted Key)

byte	16			
	4	4	4	4
0-271	encrypted_user_key(RSA 2048bit 公開鍵 n || e || 0 padding)			
272-287	MAC			

(2) 秘密鍵

入力(User Key)

byte	16			
	4	4	4	4
0-255	RSA 2048bit 公開鍵 n			
256-511	RSA 2048bit 秘密鍵 d			

出力(Encrypted Key)

byte	16			
	4	4	4	4
0-511	encrypted_user_key(RSA 2048bit 公開鍵 n || 秘密鍵 d)			
512-527	MAC			

7.3.4.3　RSA 3072bit 鍵

(1) 公開鍵

入力(User Key)

byte	16			
	4	4	4	4
0-383	RSA 3072bit 公開鍵 n			
384-399	RSA 3072bit 公開鍵 e	0 padding		

出力(Encrypted Key)

byte	16			
	4	4	4	4
0-399	encrypted_user_key(RSA 3072bit 公開鍵 n || e || 0 padding)			
400-415	MAC			

7.3.4.4　RSA 4096bit 鍵

(1) 公開鍵

入力(User Key)

byte	16			
	4	4	4	4
0-511	RSA 4096bit 公開鍵 n			
512-527	RSA 4096bit 公開鍵 e	0 padding		

出力(Encrypted Key)

byte	16			
	4	4	4	4
0-527	encrypted_user_key(RSA 4096bit 公開鍵 n || e || 0 padding)			
528-543	MAC			

7.3.5　ECC

7.3.5.1　ECC P192

(1) 公開鍵

入力(User Key)

byte	16			
	4	4	4	4
0-31	0 padding			
	ECC P-192 bit 公開鍵 Qx			
32-63	0 padding			
	ECC P-192 bit 公開鍵 Qy			

出力(Encrypted Key)

byte	16			
	4	4	4	4
0-63	encrypted_user_key(0 padding || ECC P-192bit 公開鍵 Qx || 0 padding || ECC P-192bit 公開鍵 Qy)			
64-79	MAC			

(2) 秘密鍵

入力(User Key)

byte	16			
	4	4	4	4
0-31	0 padding			
	ECC P-192 bit 秘密鍵 d			

出力(Encrypted Key)

byte	16			
	4	4	4	4
0-31	encrypted_user_key(0 padding || ECC P-192bit 秘密鍵 d)			
32-47	MAC			

7.3.5.2　ECC P224

(1) 公開鍵

入力(User Key)

byte	16			
	4	4	4	4
0-31	0 padding	ECC P-224 bit 公開鍵 Qx		
32-63	0 padding	ECC P-224 bit 公開鍵 Qy		

出力(Encrypted Key)

byte	16			
	4	4	4	4
0-63	encrypted_user_key(0 padding || ECC P-224bit 公開鍵 Qx || 0 padding || ECC P-224bit 公開鍵 Qy)			
64-79	MAC			

(2) 秘密鍵

入力(User Key)

byte	16			
	4	4	4	4
0-31	0 padding	ECC P-224 bit 秘密鍵 d		

出力(Encrypted Key)

byte	16			
	4	4	4	4
0-31	encrypted_user_key(0 padding || ECC P-224bit 秘密鍵 d)			
32-47	MAC			

7.3.5.3　ECC P256

(1) 公開鍵

入力(User Key)

byte	16			
	4	4	4	4
0-31	ECC 256 bit 公開鍵 Qx			
32-63	ECC 256 bit 公開鍵 Qy			

出力(Encrypted Key)

byte	16			
	4	4	4	4
0-63	encrypted_user_key(ECC 256bit 公開鍵 Qx || ECC 256bit 公開鍵 Qy)			
64-79	MAC			

(2) 秘密鍵

入力(User Key)

byte	16			
	4	4	4	4
0-31	ECC 256 bit 秘密鍵 d			

出力(Encrypted Key)

byte	16			
	4	4	4	4
0-31	encrypted_user_key(ECC P-256bit 秘密鍵 d)			
32-47	MAC			

7.3.5.4　ECC P384

(1) 公開鍵

入力(User Key)

byte	16			
	4	4	4	4
0-47	ECC 384 bit 公開鍵 Qx			
48-95	ECC 384 bit 公開鍵 Qy			

出力(Encrypted Key)

byte	16			
	4	4	4	4
0-95	encrypted_user_key(ECC 384bit 公開鍵 Qx || ECC 384bit 公開鍵 Qy)			
96-111	MAC			

(2) 秘密鍵

入力(User Key)

byte	16			
	4	4	4	4
0-47	ECC 384 bit 秘密鍵 d			

出力(Encrypted Key)

byte	16			
	4	4	4	4
0-47	encrypted_user_key(ECC 384bit 秘密鍵 d)			
48-63	MAC			

7.3.6　HMAC

7.3.6.1　SHA1-HMAC 鍵

入力(User Key)

byte	16			
	4	4	4	4
0-31	HMAC-SHA1 鍵			
		0 padding		

出力(Encrypted Key)

byte	16			
	4	4	4	4
0-31	encrypted_user_key(HMAC-SHA1 || 0 padding)			
32-47	MAC			

7.3.6.2　SHA256-HMAC 鍵

入力(User Key)

byte	16			
	4	4	4	4
0-31	HMAC-SHA256 鍵			

出力(Encrypted Key)

byte	16			
	4	4	4	4
0-31	encrypted_user_key(HMAC-SHA256)			
32-47	MAC			

7.3.7　KUK

入力(User Key)

byte	16			
	4	4	4	4
0-15	AES 128bit CBC 鍵			
16-31	AES 128bit CBCMAC 鍵			

出力(Encrypted Key)

byte	16			
	4	4	4	4
0-31	encrypted_user_key(AES 128bit CBC 鍵 || CBCMAC 鍵)			
32-47	MAC			

7.4　非対称鍵暗号 公開鍵の Wrapped Key フォーマット

非対称鍵暗号の公開鍵は、Wrapped Key に平文情報が含まれています。このため、TSIP の鍵生成機能を使用した Wrapped Key から平文情報が抽出可能です。各暗号アルゴリズムのデータフォーマットは以下の通りです。

7.4.1　RSA

RSA の公開鍵の Wrapped Key 構造体 tsip_rsa*XXXX*_public_key_index_t のメンバー value.key_n および value_e に公開鍵の平文データが含まれています。

key_n には Modulus、key_e には Exponent の値がビッグエンディアン並びで出力されます。

7.4.2　ECC

ECC の公開鍵の Wrapped Key 構造体 tsip_ecc_public_key_index_t のメンバー value.key_q に公開鍵の平文データが含まれています。key_q のフォーマットは以下のようになります。

7.4.2.1　ECC P-192

byte	128 bit			
	32bit	32bit	32bit	32bit
0-15	0 padding		ECC P-192 公開鍵 Qx	
16-31	ECC P-192 公開鍵 Qx(続き)			
32-47	0 padding		ECC P-192 公開鍵 Qy	
48-63	ECC P-192 公開鍵 Qy(続き)			
64-79	Wrapped Key の鍵管理情報			

7.4.2.2　ECC P-224

byte	128 bit			
	32bit	32bit	32bit	32bit
0-15	0 padding	ECC P-224 公開鍵 Qx		
16-31	ECC P-224 公開鍵 Qx(続き)			
32-47	0 padding	ECC P-224 公開鍵 Qy		
48-63	ECC P-224 公開鍵 Qy(続き)			
64-79	Wrapped Key の鍵管理情報			

7.4.2.3　ECC P-256

byte	128 bit			
	32bit	32bit	32bit	32bit
0-31	ECC P-256 公開鍵 Qx			
32-63	ECC P-256 公開鍵 Qy			
64-79	Wrapped Key の鍵管理情報			

7.4.2.4　ECC P-384

byte	128 bit			
	32bit	32bit	32bit	32bit
0-47	ECC P-384 公開鍵 Qx			
48-95	ECC P-384 公開鍵 Qy			
96-111	Wrapped Key の鍵管理情報			

7.5　Renesas Secure Flash Programmer の使用方法

7.5.1　用語

Renesas Secure Flash Programmer は RX TSIP FIT に付属している、Encrypted Key および暗号化したユーザプログラムを生成できるツールです(【注】)。Security Key Management Tool とは一部用語が異なります。Renesas Secure Flash Programmer を使用される場合、以下表のように読み替えてご使用ください。

【注】 本ツールは V.1.19 で開発終了しました。

表 7-2　Security Key Management Tool と Renesas Secure Flash Programmer 用語対比表

Security Key Management Tool	Renesas Secure Flash Programmer
UFPK(User Factory Programming Key)	Provisioning Key
W-UFPK(Wrapped User Factory Programming Key)	Encrypted Provisioning Key
KUK(Key Update Key)	鍵更新用鍵束、Update Key Ring
Encrypted Key	Encrypted Key
Wrapped Key	鍵生成情報、Key Index
Key Encryption Key	Prog User Key、User Program Key

7.5.2　provisioning key タブ

図 7-1 に Renesas Secure Flash Programmer の provisioning key タブを示します。
provisioning key タブでは、DLM サーバに送るための Provisioning Key ファイルを作成します。

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。

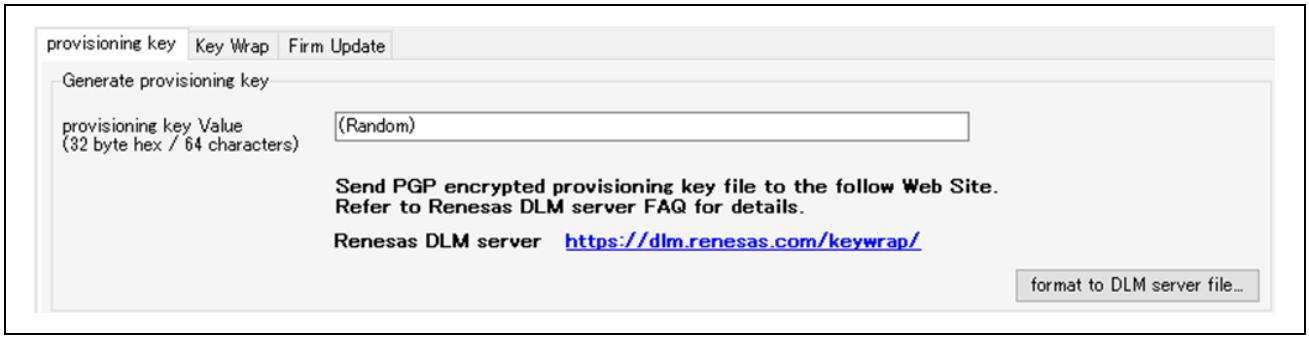

図 7-1 Renesas Secure Flash Programmer の provisioning key タブ

7.5.3　Key Wrap タブ

図 7-2 に Renesas Secure Flash Programmer の Key Wrap タブを、表 7-3 に Key Wrap タブの設定値の説明を示します。表 7-3 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 7-4 を参照してください。

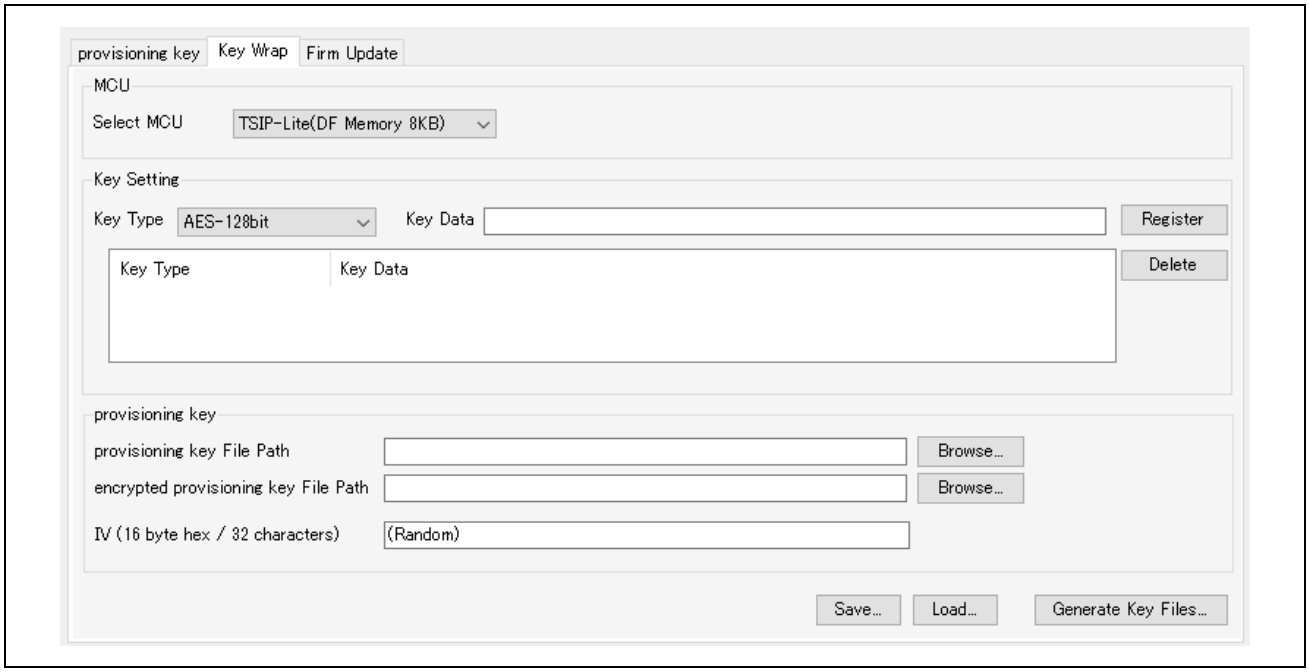

図 7-2 Renesas Secure Flash Programmer の Key Wrap タブ

表 7-3 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 16KB) 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。
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。
Key Data	ユーザ鍵 入力する鍵データフォーマットは、7.5.3.1 KeyData 入力フォーマットを参照してください。	生成するユーザ鍵データを指定してください。
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、7.5.2 provisioning key タブ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 ファイルの生成方法は 7.5.2 provisioning key タブを参照してください。
IV (16 byte hex / 32 characters)	IV 値	16 バイトの IV 値を入力してください。 入力値を”(Random)”のまま Generate Key File…ボタンを押した場合、Renesas Secure Flash Programmer 内で生成した乱数値を IV 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 言語の暗号鍵ファイルを出力します。

表 7-4 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。
Save…	Renesas Secure Flash Programmer の設定情報を*.ini ファイルに保存します。
Load…	「Save…」ボタンで保存した Renesas Secure Flash Programmer の設定情報を読み込みます。

7.5.3.1　KeyData 入力フォーマット

Key Wrap タブの Key Data に入力するデータは以下のデータを big-endian の並びで入力してください。

(1) AES-128bit データフォーマット

Byte	128bit
0-15	AES128 鍵データ

(2) AES-256bit データフォーマット

byte	256bit
0-31	AES256 鍵データ

(3) TDES データフォーマット

byte	DES ユーザ鍵 1	DES ユーザ鍵 2	DES ユーザ鍵 3
0-23	DES 鍵データ	DES 鍵データ	DES 鍵データ

(4) 2Key-TDES データフォーマット

byte	DES ユーザ鍵 1	DES ユーザ鍵 2
0-15	DES 鍵データ	DES 鍵データ

(5) DES データフォーマット

byte	DES ユーザ鍵 1
0-7	DES 鍵データ

DES 鍵データは鍵データ 7 ビットに対し、1 ビットの奇数パリティを付加したデータのため 8bit データになります。

DES 鍵データのフォーマットは以下になります。

DES ユーザ鍵 n							
バイト No	0		1		…	8	
ビット	7-1	0	7-1	0	…	7-1	0
データ	鍵データ	奇数パリティ	鍵データ	奇数パリティ	…	鍵データ	奇数パリティ

例:　パリティを付けた場合、DES ユーザ鍵 0x00000000000000 は 0x0101010101010101、0xFFFFFFFFFFFFFF は 0xFEFEFEFEFEFEFEFE、 0x01020304050607 は 0x018080614029190E になります。

(6) ARC4 データフォーマット

byte	2048bit
0-255	ARC4 鍵データ

(7) SHA1-HMAC データフォーマット

byte	160bit
0-19	SHA1-HMAC 鍵データ

(8) SHA256-HMAC データフォーマット

byte	256bit
0-31	SHA256-HMAC 鍵データ

(9) RSA-1024bit Public データフォーマット(132 バイト)

byte	RSA 1024bit Modulus n	RSA 1024bit Exponent e
0-131	128 バイト RSA Modulus n データ	4 バイト RSA Exponent e データ

【注】公開鍵

(10) RSA-1024bit Private データフォーマット(256 バイト)

byte	RSA 1024bit Modulus n	RSA 1024bit Decryption Exponent d
0-255	128 バイト RSA Modulus n データ	128 バイト RSA Decryption Exponent d データ

(11) RSA-1024bit All データフォーマット(260 バイト)

byte	RSA 1024bit Modulus n	RSA 1024bit Exponent e	RSA 1024bit Decryption Exponent d
0-259	128 バイト RSA Modulus n データ	4 バイト RSA Exponent e データ	128 バイト RSA Decryption Exponent d データ

(12) RSA-2048bit Public データフォーマット(260 バイト)

byte	RSA 2048bit Modulus n	RSA 2048bit Exponent e
0-259	256 バイト RSA Modulus n データ	4 バイト RSA Exponent e データ

(13) RSA-2048bit Private データフォーマット(512 バイト)

byte	RSA 2048bit Modulus n	RSA 2048bit Decryption Exponent d
0-511	256 バイト RSAModulus n データ	256 バイト RSA Decryption Exponent d データ

(14) RSA-2048bit All データフォーマット(516 バイト)

byte	RSA 2048bit Modulus n	RSA 2048bit Exponent e	RSA 2048bit Decryption Exponent d
0-515	256 バイト RSA Modulus n データ	4 バイト RSA Exponent e データ	256 バイト RSA Decryption Exponent d データ

(15) RSA-3072bit Public データフォーマット(388 バイト)

byte	RSA 3072bit Modulus n	RSA 3072bit Exponent e
0-387	384 バイト RSA Modulus n データ	4 バイト RSA Exponent e データ

(16) RSA-4096bit Public データフォーマット(516 バイト)

byte	RSA 4096bit Modulus n	RSA 4096bit Exponent e
0-515	512 バイト RSA Modulus n データ	4 バイト RSA Exponent e データ

(17) ECC-192bit Public データフォーマット(48 バイト)

Byte	ECC-192bit Public key Qx	ECC-192bit Public key Qy
0-47	24 バイト ECC 公開鍵 Qx データ	24 バイト ECC 公開鍵 Qy データ

(18) ECC-192bit Private データフォーマット(24 バイト)

Byte	ECC-192bit Private key
0-23	24 バイト ECC 秘密鍵データ

(19) ECC-192bit All データフォーマット(72 バイト)

byte	ECC-192bit Public key Qx	ECC-192bit Public key Qy	ECC-192bit Private key
0-71	24 バイト ECC 公開鍵 Qx データ	24 バイト ECC 公開鍵 Qy データ	24 バイト ECC 秘密鍵データ

(20) ECC-224bit Public データフォーマット(56 バイト)

byte	ECC-224bit Public key Qx	ECC-224bit Public key Qy
0-55	28 バイト ECC 公開鍵 Qx データ	28 バイト ECC 公開鍵 Qy データ

(21) ECC-224bit Private データフォーマット(28 バイト)

byte	ECC-224bit Private key
0-27	28 バイト ECC 秘密鍵データ

(22) ECC-224bit All データフォーマット(84 バイト)

byte	ECC-224bit Public key Qx	ECC-224bit Public key Qy	ECC-224bit Private key
0-83	28 バイト ECC 公開鍵 Qx データ	28 バイト ECC 公開鍵 Qy データ	28 バイト ECC 秘密鍵データ

(23) ECC-256bit Public データフォーマット(64 バイト)

byte	ECC-256bit Public key Qx	ECC-256bit Public key Qy
0-63	32 バイト ECC 公開鍵 Qx データ	32 バイト ECC 公開鍵 Qy データ

(24) ECC-256bit Private データフォーマット(32 バイト)

Byte	ECC-256bit Private key
0-31	32 バイト ECC 秘密鍵データ

(25) ECC-256bit All データフォーマット(96 バイト)

byte	ECC-256bit Public key Qx	ECC-256bit Public key Qy	ECC-256bit Private key
0-95	32 バイト ECC 公開鍵 Qx データ	32 バイト ECC 公開鍵 Qy データ	32 バイト ECC 秘密鍵データ

(26) ECC-384bit Public データフォーマット(96 バイト)

byte	ECC-384bit Public key Qx	ECC-384bit Public key Qy
0-95	48 バイト ECC 公開鍵 Qx データ	48 バイト ECC 公開鍵 Qy データ

(27) ECC-384bit Private データフォーマット(48 バイト)

Byte	ECC-384bit Private key
0-47	48 バイト ECC 秘密鍵データ

(28) ECC-384bit All データフォーマット(144 バイト)

byte	ECC-384bit Public key Qx	ECC-384bit Public key Qy	ECC-384bit Private key
0-143	48 バイト ECC 公開鍵 Qx データ	48 バイト ECC 公開鍵 Qy データ	48 バイト ECC 秘密鍵データ

(29) 鍵更新用鍵束データフォーマット

byte	256bit
0-31	鍵更新用鍵束データ

7.5.4　Firm Update タブ

図 7-3 に Renesas Secure Flash Programmer の Firm Update タブを、表 7-5 に Firm Update タブの設定値の説明を示します。表 7-5 の説明に沿って設定値を入力し、「Generate…」を押して暗号化したユーザプログラムファイル（拡張子 *.rsu もしくは拡張子*.mot）を生成してください。各ボタンの説明は、表 7-6 を参照してください。

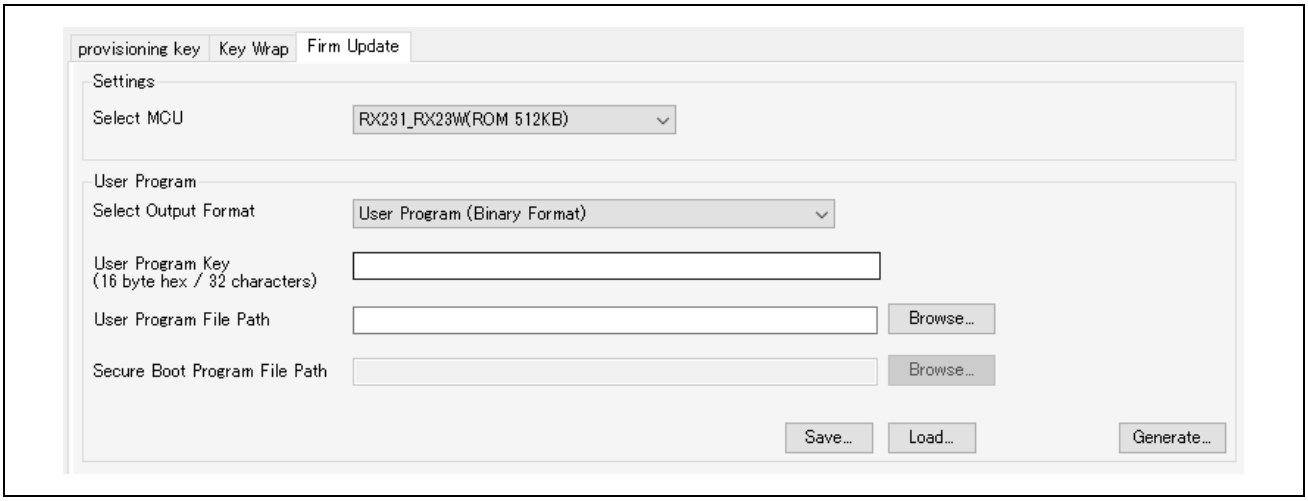

図 7-3 Renesas Secure Flash Programmer の Firm Update タブ

表 7-5 Firm Update タブ設定値の説明

パラメータ	設定値	説明
Select MCU	RX231_RX23W(ROM 512KB) RX231_RX23W(ROM 384KB) RX66T_RX72T(ROM 1MB) RX66T_RX72T(ROM 512KB) RX66T(ROM 256KB) RX26T(ROM 512KB) RX65N_RX651_RX671(ROM 2MB) RX65N_RX651_RX671(ROM 1.5MB) RX671(ROM 1MB) RX72M_RX72N_RX66N(ROM 4MB) RX72M_RX72N_RX66N(ROM 2MB)	使用するデバイスのグループ名と ROM サイズに合わせて選択してください。 詳細は 7.5.4.1 Select MCU を参照。
Select Output Format	User Program (Binary Format), User Program + Secure Boot Program(Motorola S Format)	出力するファイルフォーマットの選択 詳細は 7.5.4.2 Select Output Format を参照。
User Program Key	AES128 鍵の ASCII データ	ファームウェアアップデートで使用する AES128bit の鍵を入力 詳細は 7.5.4.3 User Program Key を参照。
User Program File Path	暗号化して MAC を付与するプログラムの Motorola S format ファイルパス	暗号化して MAC を付与する Motorola S format ファイルのファイルパスを指定してください。 詳細は 7.5.4.4 User Program File Path を参照。
Secure Boot Program File Path	セキュアブートプログラムの Motorola S format ファイルパス	暗号化するプログラムと合わせるセキュアブートプログラムの Motorola S format ファイルのファイルパスを指定してください。 詳細は 7.5.4.5 Secure Boot Program File Path を参照。

表 7-6 Firm Update タブのボタンの説明

ボタン	説明
Browse…	暗号化するユーザプログラム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…	暗号化したユーザプログラム（拡張子,rsu または拡張子 mot）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、暗号化したユーザプログラム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。
Save…	Renesas Secure Flash Programmer の設定情報を*.ini ファイルに保存します。
Load…	「Save…」ボタンで保存した Renesas Secure Flash Programmer の設定情報を読み込みます。

7.5.4.1　Select MCU

使用する MCU ファミリに合わせて選択してください。

セキュアブート・ファームアップデートサンプルの MAIN_AREA, BUFFER_AREA, SECURE_BOOT 領域を以下の領域に設定しています。

表 7-7 Select MCU による領域

設定値	SECURE_BOOT	MAIN_AREA	BUFFER_AREA
RX231_RX23W (ROM 512KB)	0xFFFF0000 – 0xFFFFFFFF	0xFFFB8000 - 0xFFFEFFFF	0xFFF80000 - 0xFFFB7FFF
RX231_RX23W (ROM 384KB)		0xFFFC8000 - 0xFFFEFFFF	0xFFFA0000 - 0xFFFC7FFF
RX66T_RX72T (ROM 1MB)		0xFFF78000 – 0xFFFEFFFF	0xFFF00000 - 0xFFF77FFF
RX66T_RX72T (ROM 512KB)		0xFFFB8000 - 0xFFFEFFFF	0xFFF80000 - 0xFFFB7FFF
RX66T (ROM 256KB)		0xFFFD8000 - 0xFFFEFFFF	0xFFFC0000 - 0xFFFD7FFF
RX26T (ROM 512KB)		0xFFFC0000 - 0xFFFEFFFF	0xFFF80000 - 0xFFFAFFFF
RX65N_RX651_RX671 (ROM 2MB)		0xFFF00000 – 0xFFFEFFFF	0xFFE00000 - 0xFFEEFFFF
RX65N_RX651_RX671 (ROM 1.5MB)		0xFFF40000 – 0xFFFEFFFF	0xFFE40000 - 0xFFEEFFFF
RX671 (ROM 1MB)		0xFFF80000 – 0xFFFEFFFF	0xFFE80000 - 0xFFEEFFFF
RX72M_RX72N_RX66N (ROM 4MB)		0xFFE00000 – 0xFFFEFFFF	0xFFC00000 - 0xFFDEFFFF
RX72M_RX72N_RX66N (ROM 2MB)		0xFFF00000 – 0xFFFEFFFF	0xFFD00000 - 0xFFDEFFFF

7.5.4.2　Select Output Format

Select Output Format では、生成するファイルのフォーマットを選択します。
各フォーマットには、RX Firmware Update FIT V.1.0x で定義される *.rsu ファイルヘッダを TSIP 用に一部変更した RSU ヘッダを付加したファイルを出力します。

表 7-8 TSIP 用 RX Firmware Update FIT V.1.0x RSU ヘッダフォーマット

offset	Component	Contents name	Length	Note
0x0000	Header	Magic Code	7	“Renesas”
0x0007		Image Flags	1	User Program (Binary Format) : 0xFE(TESTING) User Program + Secure Boot Program(Motorola S Format) : 0xf8(VALID)
0x0008	Signature	Firmware Verification Type	32	ASCII “mac-aes128-cmac-with-tsip”
0x0028		Signature size	4	使用していません(0x00)。
0x002C		Signature	256	使用していません(0x00)。
0x012C	Option	Dataflash Flag	4	使用していません(0x00)。
0x0130		Dataflash Start Address	4	使用していません(0x00)。
0x0134		Dataflash End Address	4	使用していません(0x00)。
0x0138		Image Size	4	使用していません(0x00)。
0x013C		Reserved	123	All 0x00
0x01B7		復号フラグ *1	1	0x00 : 復号未実施
0x01B8		Format Type	4	出力するファイルフォーマット ASCII で”rsu”(0x72, 0x73, 0x75, 0x00) もしくは “mot”(0x6D, 0x6F, 0x74, 0x00)
0x01BC		IV *1	16	ユーザプログラム暗号化時に使用した IV *2
0x01CC		SessionKey0 *1	16	ユーザプログラム暗号化時に使用した鍵を prog user key で暗号化した鍵 *2
0x01DC		SessionKey1 *1	16	ユーザプログラム暗号化時に使用した鍵を prog user key で暗号化した鍵 *2
0x01EC		暗号化されたユーザプログラム+MAC のワードサイズ *1	4	暗号化されたユーザプログラム+ユーザプログラム暗号化時に生成された MAC の合計ワードサイズ
0x01F0		MAC *1	16	ユーザプログラム暗号化時に生成された MAC *2
0x0200	Descriptor	Sequence Number	4	常に 1
0x0204		Start Address	4	ユーザプログラム領域の開始アドレス
0x0208		End Address	4	ユーザプログラム領域の終了アドレス
0x020C		Execution Address	4	ユーザプログラム実行開始アドレス格納アドレス
0x0210		Hardware ID	4	使用していません(0x00)。
0x0214		Reserved(0x00)	236	-
0x0300	Application Binary		N	暗号化されたユーザプログラム
0x0300 +N	Dataflash Binary		M	対応していません。

【*1】: RX Firmware Update FIT V.1.0x で定義される *.rsu ファイルフォーマットから TSIP 用に変更したパラメータ
【*2】: 図 3-47　ファームウェアと Image Encryption Key の暗号化方法 を参照ください。

(1) User Program (Binary Format)選択時：

User Program 入力されたファイルの MAIN_AREA 内のデータのみを暗号化し、RSU ヘッダを付加したバイナリファイルを出力します。

(2) User Program + Secure Boot Program(Motorola S Format)選択時：

User Program 入力されたファイルの MAIN_AREA 内のデータのみを暗号化し、メインエリアの先頭 0～0x300 に RSU ヘッダを付加したデータを、”Secure Boot Program File Path”で指定した Motorola S format ファイルを合わせたデータを出力します。OFS 領域ならびに Data Flash 領域のデータは”Secure Boot Program File Path”で指定したファイルのデータを使用します。

セキュアブートと暗号化されたユーザプログラムを合わせた Motorola S format ファイルは、Motorola S format ファイル自体を盗まれると、該当する RX TSIP 搭載デバイスに書きこみ、実行することで、復号することが可能です。セキュアブートを含む暗号化されたユーザプログラムを書き込む場合は、セキュアな工場で書き込みを行ってください。

7.5.4.3　User Program Key

図 3-47　ファームウェアと Image Encryption Key の暗号化方法 で示される Tmp Key を暗号化するための User Prog Key を指定します。

7.5.4.4　User Program File Path

暗号化するユーザプログラムを指定します。

7.5.4.5　Secure Boot Program File Path

Select Output Format で”User Program + Secure Boot Program(Motorola S Format)”選択時、暗号化したユーザプログラムと組み合わせるセキュアブートプログラムの Motorola S format ファイルを指定します。

7.5.5　鍵の注入手順

鍵を注入するときに使用する鍵のファイル生成手順を示します。

以下の例では、AES の C ソースファイルの生成手順を示します。

1. Provisioning Key を生成

   provisioning key タブに Provisioning Key 値を設定して、拡張子*.key というファイル名で、Provisioning Key ファイルを生成します。この例では sample.key というファイル名を使用しています。

図 7-4　provisioning key タブ 指定値で Provisioning Key を生成する場合の例

“format to DLM server file…”ボタンを押すと Provisioning Key ファイルが生成されます。

2. Encrypted Provisioning Key の取得

   1 で生成した sample.key ファイルを Renesas Key Wrap service(https://dlm.renesas.com/keywrap)に送付して Encrypted Provisioning Key を取得します。

   詳細な取得情報は、Renesas Key Wrap Service の FAQ もしくはアプリケーションノートをご参照ください。

3. AES128 鍵ファイルを生成

[Key Wrap]タブで AES 128 鍵ファイルを生成します。
使用するデバイスが TSIP 搭載デバイスか TSIP-Lite 搭載デバイスかと Data Flash のサイズで、Select MCU を選択してください。
Key Type に鍵の種類、Key Data に鍵データを入力後、”Register”ボタンを押してください。
“provisioning key File Path”には、手順 1 で生成した Provisioning Key ファイル、
“encryoted provisioning key File Path”には手順 2 で取得した Encrypted Provisioning Key ファイルを設定してください。この例では簡略化のため IV は”(Random)”を選択します。
すべての設定が完了したら、”Generate Key File…”ボタンを押してください。
鍵データファイル(key_data.c および key_data.h)を任意にフォルダに出力可能です。
本例では AES-128bit の鍵のみ設定していますが、一つの鍵を設定後、次の鍵を Key Type ならびに Key Data に設定して、”Register”ボタンを押すことで、1 つのファイルに複数の鍵の暗号化が可能です。

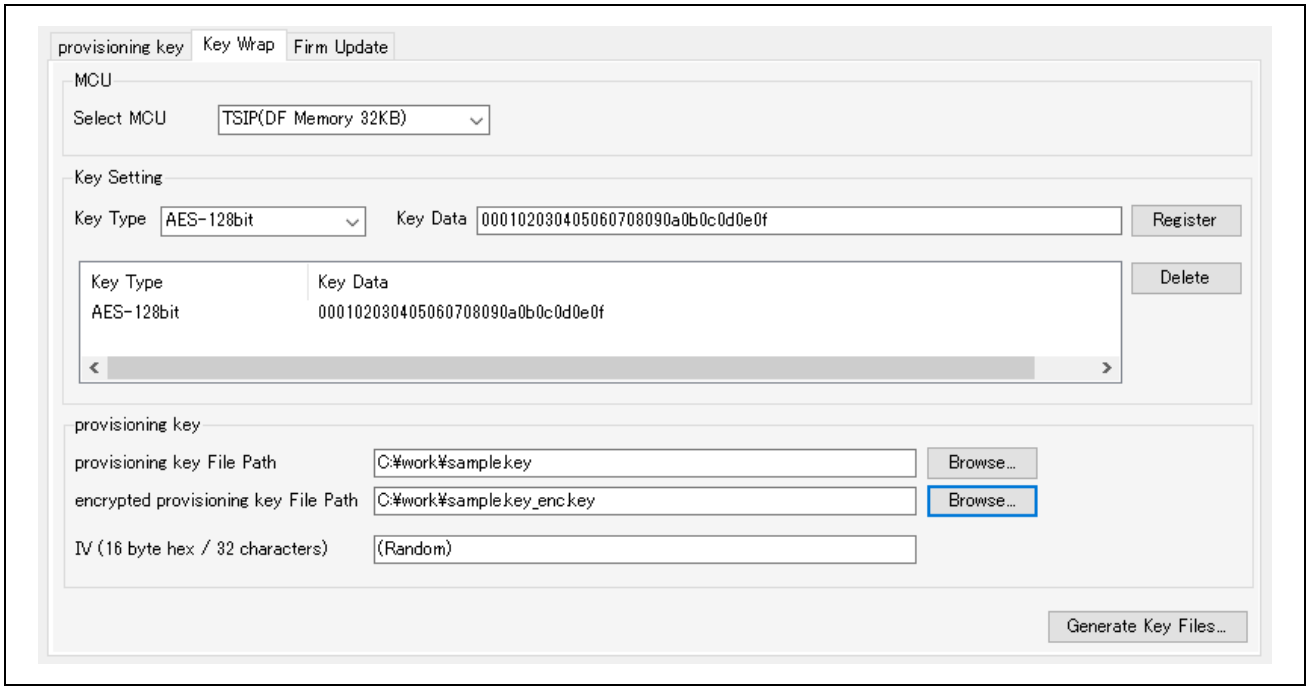

図 7-5　Key Wrap タブ AES128 鍵ファイル出力設定例

出力された C ソースファイルをご使用になっているプロジェクトに組み込みます。プログラム内で、R_TSIP_GenerateAes128KeyIndex()に鍵データを渡すことで、TSIP から鍵生成情報が出力されます。

7.5.6　鍵更新時の操作方法

鍵更新では、鍵更新用鍵束をあらかじめ注入しておくことで新たな Provisioning Key の用意が不要となります。

1. Provisioning Key を生成
   provisioning key タブに Provisioning Key 値を設定して、拡張子*.key というファイル名で、
   Provisioning Key ファイルを生成します。この例では sample.key というファイル名を使用しています。

図 7-6　provisioning key タブ 指定値で Provisioning Key を生成する場合の例

“format to DLM server file…”ボタンを押すと Provisioning Key ファイルが生成されます。

2. Encrypted Provisioning Key の取得
   1 で生成した sample.key ファイルを Renesas Key Wrap service(https://dlm.renesas.com/keywrap)に送付して Encrypted Provisioning Key を取得します。
   詳細な取得情報は、Renesas Key Wrap Service の FAQ もしくはアプリケーションノートをご参照ください。

3. 鍵更新用鍵束ファイルを生成

Key Wrap タブで鍵更新用鍵束ファイルを生成します。
使用するデバイスが TSIP 搭載デバイスか TSIP-Lite 搭載デバイスかと Data Flash のサイズで、Select MCU を選択してください。
Key Type に鍵の種類、Key Data に鍵データを入力後、”Register”ボタンを押してください。
“provisioning key File Path”には、手順 1 で生成した Provisioning Key ファイル、
“encryoted provisioning key File Path”には手順 2 で取得した Encrypted Provisioning Key ファイルを設定してください。この例では簡略化のため IV は”(Random)”を選択します。
すべての設定が完了したら、”Generate Key File…”ボタンを押してください。
鍵データファイル(key_data.c および key_data.h)を任意にフォルダに出力可能です。
本例では AES-128bit の鍵と合わせて鍵更新用鍵束を生成する例を示しています。

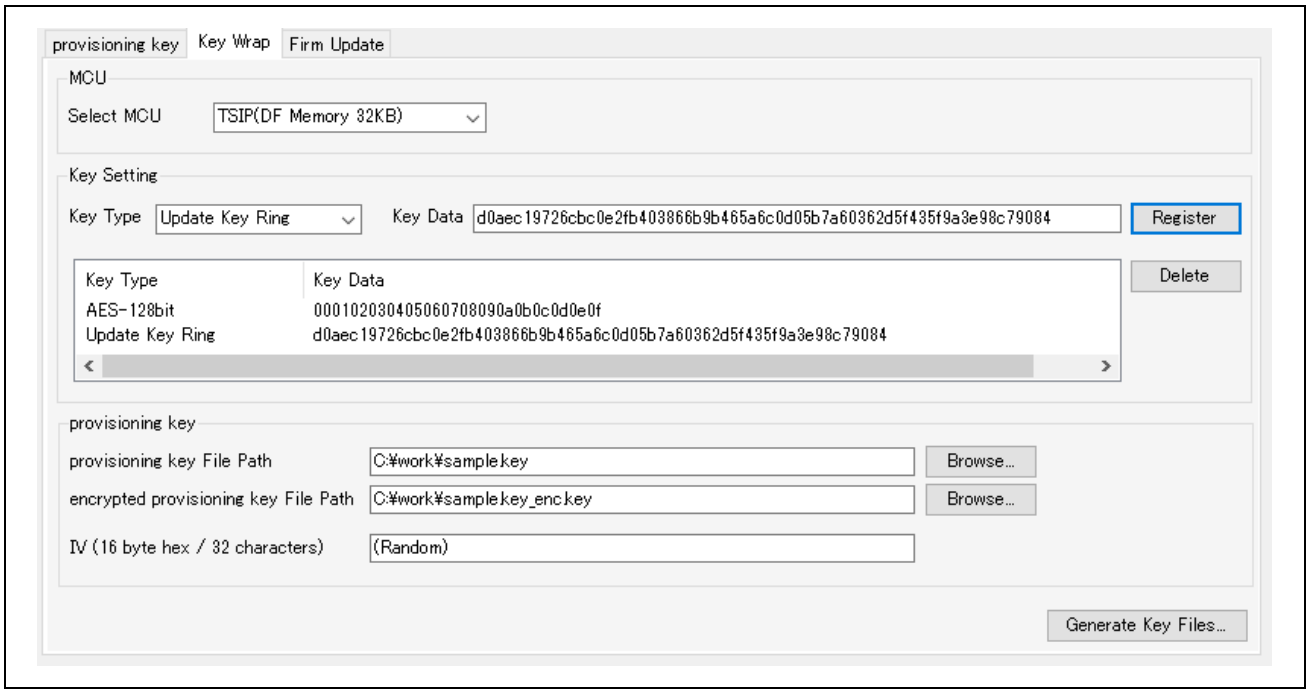

図 7-7　Key Wrap タブ AES128 鍵ファイルと鍵更新用鍵束出力設定例

出力された C ソースファイルをご使用になっているプロジェクトに組み込みます。プログラム内で、R_TSIP_GenerateUpdateKeyRingKeyIndex()に鍵データを渡すことで、TSIP から鍵更新用鍵束の鍵生成情報が出力されます。
出力された鍵更新用鍵束の鍵生成情報は、R_TSIP_Open()の第 2 引数に渡してください。

続いて、市場で鍵を更新する場合の、鍵更新用鍵束を使ったラッピング方法を示します。

4. 鍵更新用鍵束の.key ファイルを生成します。provisioning key タブの”provisioning key Value”に手順 3 で設定した Update Ker Ring の値を入力し、”format to DLM server file…”ボタンを押して、 鍵更新用鍵束の.key ファイルを生成します。

図 7-8 Renesas Secure Flash Programmer の provisioning key タブ

5. 更新する鍵データの鍵ファイルを生成

   Key Wrap タブで更新する鍵データの鍵ファイルを生成します。

   使用するデバイスが TSIP 搭載デバイスか TSIP-Lite 搭載デバイスかと Data Flash のサイズで、Select MCU を選択してください。

   Key Type に鍵の種類、Key Data に鍵データを入力後、”Register”ボタンを押してください。

   “provisioning key File Path”には、手順 4 で生成した鍵更新用鍵束の Provisioning Key ファイル、“encryoted provisioning key File Path”の値は使用しないので、任意の Encrypted Provisioning Key ファイルを設定してください。この例では、手順 2 で取得した Encrypted Provisioning Key ファイル(UpdateKeyRing.key)を指定しています。

   この例では簡略化のため IV は”(Random)”を選択します。

   すべての設定が完了したら、”Generate Key File…”ボタンを押してください。

   鍵データファイル(key_data.c および key_data.h)を任意にフォルダに出力可能です。

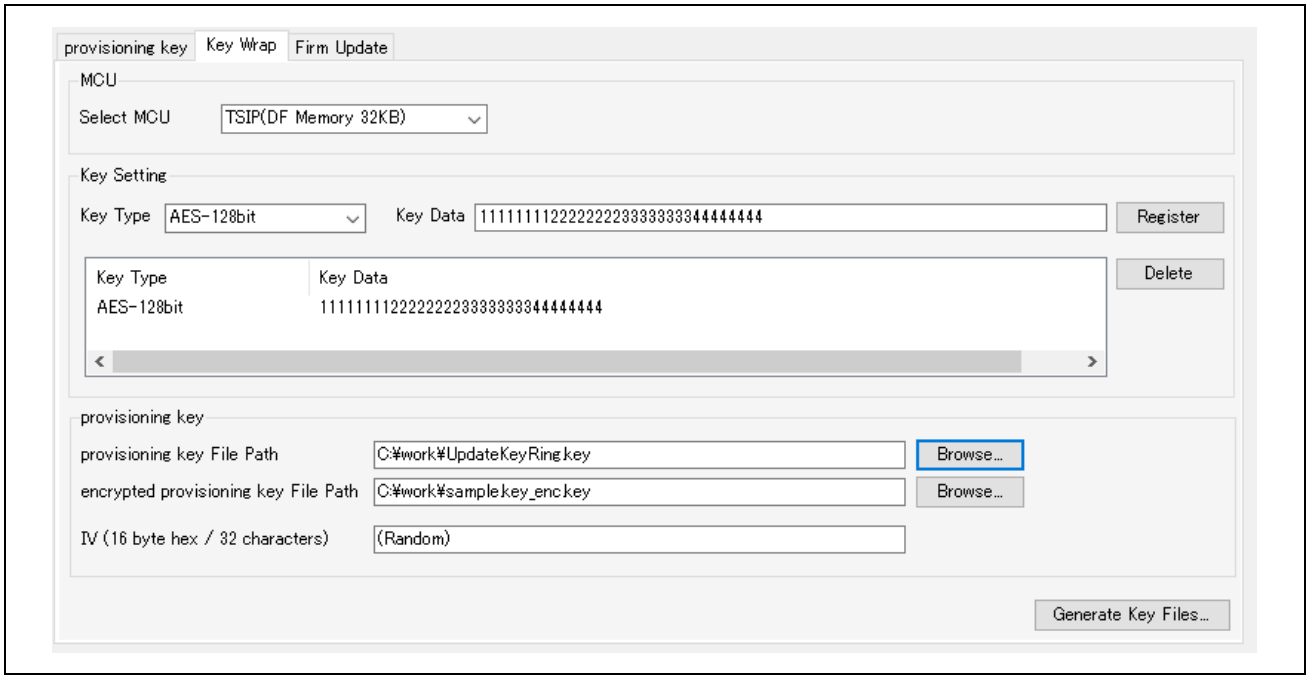

図 7-9 Renesas Secure Flash Programmer の Key Wrap タブ 鍵更新用鍵束による更新鍵の暗号化設定

6. 出力された key ファイル内の key_data.encrypted_user_xxx_key[]、key_data.iv[]を R_TSIP_UpdatexxxKeyIndex()の encrypted_key と iv に入力します。
(xxx には暗号アルゴリズムと鍵長が入ります。)
AES 128 の場合、key_data.encrypted_user_aes128_key[]を R_TSIP_UpdateAes128KeyIndex()の encrypted_key に、key_data.iv を R_TSIP_UpdateAes128KeyIndex()の iv にそれぞれ入力します。

7. R_TSIP_UpdatexxxKeyIndex()から出力される KeyIndex を新しい鍵の鍵生成生成として使用することが可能です。それまで使用していた鍵生成生成情報は消去してください。

7.5.7　使用時の注意事項

(1) 出力ファイルフォーマット

Renesas Secure Flash Programmer を使って鍵データの C ソースファイルを出力した場合、出力される構造体のメンバー名が Security Key Management Tool と異なります。

付属デモプロジェクト\src フォルダの key_data.c、key_data.h、\genkey フォルダの各ファイル、および以下表を参考に、置き換えを行ってください。

表 7-9　C ソースファイル　構造体比較表

Renesas Secure Flash Programmer *1	Security Key Management Tool *2	説明
encrypted_user_xxxx_key[]	encrypted_user_key[]	暗号化されたユーザ鍵
encrypted_provisioning_key[] *3	wufpk[] *3	暗号化された Provisioning Key
iv[] *3	initial_vector[] *3	初期化ベクタ
user_xxxx_key_index[]	-	鍵生成情報
-	keytype	RX TSIP ドライバでは使用しません。
-	shared_key_number	RX TSIP ドライバでは使用しません。
-	crc[]	RX TSIP ドライバでは使用しません。

*1 st_key_block_data_t.key_data の構造体メンバー名になります。

*2 g_<keyname>の構造体メンバー名になります。詳細は Security Key Management Tool のユーザーズマニュアル「4.5.4.2 filetype オプション csource 指定時」を参照してください。

*3 wufpk および initial_vector はそれぞれ、全ての鍵で同じ値を使用してください。

8. 参考ドキュメント

ユーザーズマニュアル：ハードウェア
（最新版をルネサス エレクトロニクスホームページから入手してください。）

テクニカルアップデート／テクニカルニュース
（最新の情報をルネサス エレクトロニクスホームページから入手してください。）

ユーザーズマニュアル：開発環境
RX ファミリ CC-RX コンパイラ ユーザーズマニュアル（R20UT3248）
（最新版をルネサス エレクトロニクスホームページから入手してください。）

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	2015.09.27	—	初版発行
1.01	2016.06.27	—	ファームウェアアップデート機能を追加
1.02	2017.05.31	—	・AES-CMAC 機能を追加 ・R_TSIP_SelfCheck1 ()と R_TSIP_SelfCheck2 ()と R_TSIP_SoftwareReset を R_TSIP_Open()に統合 ・R_TSIP_InstallAes128UserKey()と R_TSIP_InstallAes256UserKey()に引数を追加 ・USB メモリを用いたファームウェアアップデートの内容を変更
1.03	2017.09.30	—	・SHA ならびに RSA 用の下記フラグを追加 TSIP_SHA1, TSIP_SHA256, TSIP_RSA_1024,TSIP_RSA_2048 ・ R_TSIP_ERROR_PROHIBIT_FUNCTION エラーを追加
1.04	2018.02.28	—	・TLS ならびに SECURE BOOT 用の下記フラグを追加 TSIP_TLS, TSIP_SHA_1_HMAC,TSIP_SHA_256_HMAC, SECURE_BOOT ・戻り値の型を enum e_tsip_err_t 型に変更 ・R_TSIP_Rsa1024ModularExponent 削除 ・R_TSIP_Rsa2048ModularExponent 削除 ・TLS 用 API 追加
1.05	2018.04.30	—	・TDES、MD5 用の下記フラグを追加 TSIP_TDES_ECB_ENCRYPT,TSIP_TDES_ECB_DECRYPT, TSIP_TDES_CBC_ENCRYPT,TSIP_TDES_CBC_DECRYPT, TSIP_MD5 RSA 用フラグ変更 RSASSA_1024, RSASSA_2048, RSAES_1024, RSAES_2048, ・MD5 用 API 追加 ・TDES 用 API 追加 ・RSAES-PKCS1-v1_5 用 API 追加 ・TLS 用 API 追加
1.06	2018.09.28	—	・RX66T 対応追加 ・R_TSIP_TlsAes128CbcEncryptInit/Update/Final R_TSIP_TlsAes128CbcDecryptInit/Update/Final R_TSIP_TlsAes256CbcEncryptInit/Update/Final R_TSIP_TlsAes256CbcDecryptInit/Update/Final R_TSIP_TlsSha1HmacGenerateInit/Update/Final R_TSIP_TlsSha256HmacGenerateInit/Update/Final R_TSIP_TlsSha1HmacVerifyInit/Update/Final R_TSIP_TlsSha256HmacVerifyInit/Update/Final 削除 ・引数 各アルゴリズムの Init API 鍵生成情報の型を変更 ・ 鍵更新用 API 追加 ・R_TSIP_Aes128CbcEncryptInit/Update/Final R_TSIP_Aes128CbcDecryptInit/Update/Final R_TSIP_Aes256CbcEncryptInit/Update/Final R_TSIP_Aes256CbcDecryptInit/Update/Final TLS 対応の記述追加

			・引数 各アルゴリズムの Init API 鍵生成情報の型を変更 ・鍵更新用 API 追加 ・RSA 鍵生成 API 追加 ・SHA-HMAC API 修正
1.07	2019.02.28	—	RX72T 対応追加
1.08	2019.09.30	—	・RX23W、RX72M、および楕円曲線暗号対応追加 ・コンフィグレーション「SECURE_BOOT」を「TSIP_SECURE_BOOT」に変更し、「TSIP_INSTALL_KEY_RING_INDEX」を削除 ・R_TSIP_GenerateTdesUserKeyIndex を R_TSIP_GenerateTdesKeyIndex に変更
1.09	2020.03.31	—	・CCM、HMAC 鍵生成、ECDH および Key Wrap 機能追加 ・RX66N、RX72N 対応追加 ・R_TSIP_Open 引数および用語説明の s_flash を削除し、新たにファイル s_flash.c を追加 ・コンフィグレーション TSIP_USER_HASH_ENABLED、および RSA 署名生成/検証関数で使用するユーザ定義関数 R_TSIP_RSA_IF_HASH を追加 ・GCM 演算実行準備関数の Return Values において、TSIP_ERR_FAIL を追加、および記載順序を統一 ・Return Values の TSIP_ERR_PARAMETER において、ハンドル以外の要因があるものを「入力データが不正」に変更 ・RSA/ECC 公開鍵を生成する関数において、Parameters の鍵生成情報に各メンバの説明を追加 ・RSA 暗号化/復号関数において、Parameters の説明を修正 ・署名生成/検証関数において、データ種別としてメッセージとハッシュ値を選択する機能を追加 ・character、mirror のスペルを修正 ・s_inst1/2 を key_index_1/2 に型も含め変更、合わせて R_TSIP_UpdateTlsRsaPublicKeyIndex を key_index_1 の対象に追加
1.10	2020.06.30	—	・ECC P-384 鍵インストール、鍵生成、鍵更新機能を追加 ・ARC4、ECDSA P-384 機能追加 ・ECDH P-256 機能の RX72M、RX66N、RX72N 対応追加 ・Key Wrap 機能の RX72M、RX66N、RX72N 対応追加 ・デモプロジェクトに ARC4 機能および RX72N 対応を追加 ・コンフィグレーション TSIP_USER_HASH_ENABLED、および RSA 署名生成/検証関数で使用するユーザ定義関数 R_TSIP_RSA_IF_HASH を削除 ・コンフィグレーション TSIP_ECDSA_P384、TSIP_ECDH_P256、TSIP_USER_SHA_384_ENABLED を追加 ・ECDH 鍵交換関数 R_TSIP_EcdhXXX()の関数名を、R_TSIP_EcdhP256XXX()に変更 ・ECC 公開鍵の構造体 tsip_ecc_public_key_index_t を変更
1.11	2020.09.30	—	・DH2048bit および ECDHE512bit 機能を追加 ・R_TSIP_GenerateXXXKeyIndex()および R_TSIP_UpdateXXXKeyIndex()の Parameters において、iv の説明を統一 ・R_TSIP_EcdhP256Init()において、ECDH(AES GCM128 with IV)を機能削除 ・R_TSIP_AesXXXKeyWrap()と R_TSIP_AesXXXKeyUnwrap()を TSIP-Lite/TSIP 共通の API 関数に変更

1.12	2021.06.30	—	・AES 暗号プロジェクトおよび TLS 連携機能プロジェクトを追加
1.13	2021.08.31	—	RX671 対応追加
1.14	2021.10.22	—	TLS1.3 対応追加(RX65N のみ)
1.15	2022.03.31	—	・TLS1.3 対応追加(RX66N、RX72M、RX72N) ・TLS1.2 RSA 4096bit 対応追加 ・ハッシュ値演算途中経過取得関数追加 ・ref フォルダ、r_tsip_md5_rx.c および r_tsip_sha_rx.c を削除し、r_tsip_hash_rx.c を追加
1.16	2022.09.15	—	・TLS1.3 対応追加(Resumption/0-RTT) ・AES-CTR 対応追加 ・RSA3072、RSA4096 対応追加
1.17	2023.01.20	—	・アプリケーションノートの章構成見直し ・TLS1.3 サーバ対応追加(RX65N, RX72N)
1.18	2023.05.24	—	RX26T 対応追加
1.19	2023.11.30	—	セキュアブート・ファームウェアアップデートのデモプロジェクトの更新
1.20	2024.02.28	—	AES-CCM 復号機能に関する HW 不具合のためのソフトウェアワークアラウンドを適用(TSIP-Lite)
1.21	2024.06.28	—	・TLS1.2 サーバ対応追加(RX65N, RX66N、RX72M、RX72N) ・R_TSIP_TlsServersEphemeralEcdhPublicKeyRetrieves で対応する公開鍵の種類を追加

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。