RX ファミリ

Renesas Secure IP モジュール Protected Mode Firmware Integration Technology

要旨

本資料は、RX ファミリ搭載の RSIP 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 RSIP Protected Mode ドライバ(RSIP PM ドライバ)と呼びます

RSIP PM 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

RSIP PM ドライバは 表 1 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

RX261 グループ

表 1　RSIP-E11A 暗号アルゴリズム

暗号種別		アルゴリズム
非対称鍵暗号	署名生成/検証	ECDSA(secp256r1, brainpoolP256r1, secp256k1)：RFC6979
	鍵生成	secp256r1, brainpoolP256r1, secp256k1
対称鍵暗号	AES	AES(128/256 bit) ECB/CBC/CTR：FIPS 197, SP800-38A
ハッシュ	SHA	SHA-224, SHA-256：FIPS 180-4
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 800-90B で検定済み
鍵更新機能		AES, ECC, HMAC
Key Wrap		RFC3394
KDF		SP800-56A, SP800-56C ECC(secp256r1)

注

RFC 2104: HMAC: Keyed-Hashing for Message Authentication (rfc-editor.org)
RFC 4543: The Use of Galois Message Authentication Code (GMAC) in IPsec ESP and AH (rfc-editor.org)
RFC 6979 - Deterministic Usage of the Digital Signature Algorithm (DSA) and Elliptic Curve Digital Signature Algorithm (ECDSA) (ietf.org)
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 SP800-90A: https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-90Ar1.pdf
NIST SP800-90B: https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-90B.pdf
RFC 3394: Advanced Encryption Standard (AES) Key Wrap Algorithm (rfc-editor.org)
FIPS 180-4: https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.180-4.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。

表 1-1　用語説明

用語	内容
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、HMAC の場合は対称鍵がユーザ鍵となり、ECC の場合は非対称鍵がそれぞれユーザ鍵となる。
Encrypted Key	ユーザ鍵に UFPK もしくは KUK 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。
Wrapped Key	Encrypted Key を鍵の注入または鍵更新により RSIP 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。
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。
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。RSIP 内部にて HRK で UFPK に復号されて使用される。
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)	RSIP 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)	RSIP 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。
Image Encryption Key	ファームウェアアップデート API で使用するファームウェアの暗号化で使用する 256bit の鍵
Key Encryption Key	Image Encryption Key のラップで使用する AES 128bit の鍵

1.2 RSIP の概要

RX ファミリ内の Renesas Secure IP（RSIP）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、RSIP 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。RSIP は、RSIP 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。

RSIP 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。

RSIP は、RSIP 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、デバイス固有値である HUK に紐付けられて生成されるため、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 RSIP ハードウェアにアクセスするためには、RSIP PM ドライバを使用する必要があります。

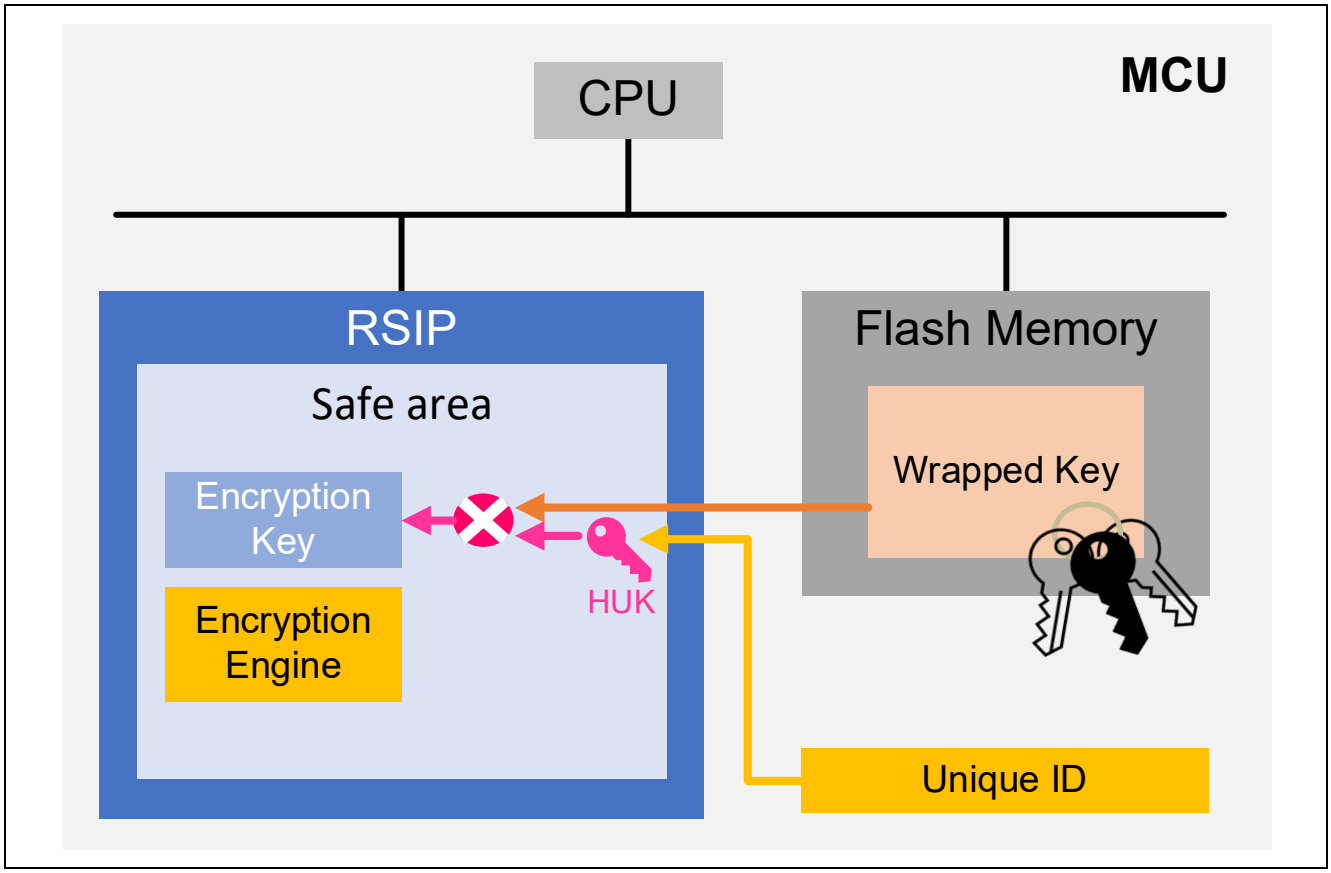

図 1-1 RSIP 搭載 MCU

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
r20an0748jj0200-rx-rsip-security.pdf	RSIP PM ドライバ アプリケーションノート(日本語)
r20an0748ej0200-rx-rsip-security.pdf	RSIP PM 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)
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)
FITModules	FIT モジュールフォルダ
r_rsip_protected_rx_v2.00.zip	RSIP PM ドライバ FIT Module
r_rsip_protected_rx_v2.00.xml	RSIP PM ドライバ FIT Module e^2 studio FIT プラグイン用 XML ファイル
r_rsip_protected_rx_v2.00_extend.mdf	RSIP PM 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
rx261_ek_rsip_sample	鍵注入と暗号の仕様方法のデモプロジェクト
rx261_ek_rsip_secure_update	セキュアブートとファームウェアアップデートのデモプロジェクト
rx261_ek_rsip_secure_boot	セキュアブートプロジェクト
rx261_ek_rsip_user_program	ファームウェアアップデートプロジェクト

r_rsip_protected_rx_v.2.00.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	RSIP PM ドライバコンフィグファイルフォルダ
r_rsip_protected_rx_config.h	RSIP PM ドライバコンフィグファイル(デフォルト設定)
r_rsip_protected_rx	RSIP PM ドライバ FIT Module フォルダ
src	ソースコードフォルダ
inc	ヘッダファイル格納フォルダ
rx261	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
r_rsip.h	common 部共通化のための API ヘッダ呼び出しファイル
r_rsip_cfg.h	common 部共通化のためのコンフィグ呼び出しファイル
primitive	RSIP アクセス用ソースコードフォルダ
common	マイコン機種非依存部分のプログラムコード格納用フォルダ
r_rsip_err.h	RSIP PM ドライバ内部関数の戻り値定義
r_rsip_util.h	RSIP PM ドライバ内部共通使用機能の定義
rx261	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
r_rsip_rx261_iodefine.h	RSIP アクセス用ヘッダファイル
r_rsip_primitive.h	RSIP アクセス用ソースコード用ヘッダファイル
r_rsip_rx_finctionxxx.c	RSIP アクセス用ソースコード(ファイル名の xxx は数値)
r_rsip_rx_pxx.c	RSIP アクセス用ソースコード(ファイル名の xx は数値)
s_flash.c	鍵情報ファイル
private	RSIP PM ドライバ内部関数のプログラムコード格納用フォルダ
common	マイコン機種非依存部分のプログラムコード格納用フォルダ
r_rsip_private.c	内部関数用ソースコード
r_rsip_private.h	内部関数用ヘッダファイル
rx261	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
r_rsip_hal.c	RSIP PM ドライバ HW 依存関数用ソースコード
r_rsip_wapper.c	RSIP PM ドライバラッパー関数用ソースコード
r_rsip_wrapper.h	RSIP PM ドライバラッパー関数用ヘッダファイル
public	RSIP PM ドライバ API のプログラム格納用フォルダ
common	マイコン機種非依存部分のプログラムコード格納用フォルダ
r_rsip,c	RSIP PM ドライバ共通機能 API 用ソースコード
r_rsip_aes.c	RSIP PM ドライバ AES API 用ソースコード
r_rsip_ecc.c	RSIP PM ドライバ ECC API 用ソースコード
r_rsip_kdf.c	RSIP PM ドライバ KDF API 用ソースコード
r_rsip_otf.c	RSIP PM ドライバ OTF API 用ソースコード
r_rsip_pki.c	RSIP PM ドライバ PKI API 用ソースコード
r_rsip_sha.c	RSIP PM ドライバ HASH API 用ソースコード
r_rsip_rsa.c	RSIP PM ドライバ RSA API 用ソースコード
r_rsip_public.h	RSIP PM ドライバ API 用ヘッダファイル
rx261	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
r_rsip_rx.c	RSIP PM ドライバ FIT モジュール関連 API 用ソースコード
r_rsip_fwup.c	RSIP PM ドライバ ファームウェアアップデート API 用ソースコード
r_rsip_key_injection.c	RSIP PM ドライバ 鍵注入 API 用ソースコード
doc	ドキュメント格納フォルダ
ja	ドキュメント格納フォルダ(日本語)
r20an0748jj0200-rx-rsip-security.pdf	RSIP PM ドライバ アプリケーションノート(日本語)
en	ドキュメント格納フォルダ(英語)
r20an0748ej0200-rx-rsip-security.pdf	RSIP PM ドライバ アプリケーションノート(英語)
r_rsip_protected_rx_if.h	RSIP PM ドライバヘッダファイル
readme.txt	Readme

1.4　開発環境

RSIP PM 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_rsip_protected_rx V2.00

コンパイラバージョン: Renesas Electronics C/C++ Compiler Package for RX Family V3.07.00
(最適化レベル 2、設定に"-lang = c99"オプションを追加)
GCC for Renesas RX 8.3.0.202311
(サイズ重視最適化、設定に" -std=gnu99"オプションを追加)
IAR C/C++ Compiler for Renesas RX version 5.10.01
(最適化レベル高/バランス)

コンフィグレーションオプション:
Renesas Electronics C/C++ Compiler Package for RX Family: -isa=rxv3, 最適化レベル 2
GCC for Renesas RX: RXv3, 最適化レベル-Os
IAR C/C++ Compiler for Renesas RX: --core rxv3 -Oh, 最適化レベル高(バランス)

ROM、RAM およびスタックのコードサイズ

分類	使用メモリ		
	Renesas Compiler	GCC	IAR Compiler
ROM	**99,884 バイト**	**100,437 バイト**	**98,378 バイト**
RAM	**12 バイト**	**12 バイト**	**12 バイト**
スタック	**288 バイト**	**296 バイト**	**264 バイト**

1.6 性能情報

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。RSIP 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。

1.6.1 RX261

表 1-4 共通機能 API の性能

API	性能 (単位：サイクル)
R_RSIP_Open	790,000
R_RSIP_Close	450
R_RSIP_GetVersion	30

表 1-5 鍵管理 API の性能

API	性能 (単位：サイクル)
R_RSIP_EncryptedKeyWrap	11,000
R_RSIP_KeyGenerate	5,700
R_RSIP_KeyPairGenerate	9,300,000
R_RSIP_PublicKeyExport	130
R_RSIP_InitialKeyWrap	9,200

表 1-6 乱数生成 API の性能

API	性能 (単位：サイクル)
R_RSIP_RandomNumberGenerate	1,000

表 1-7 AES API の性能

アルゴリズム	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
ECB モード暗号化	R_RSIP_AES_Cipher_Init	4,500	4,500	4,500
	R_RSIP_AES_Cipher_Update	840	950	1,100
	R_RSIP_AES_Cipher_Finish	460	460	460
ECB モード復号	R_RSIP_AES_Cipher_Init	4,500	4,500	4,500
	R_RSIP_AES_Cipher_Update	980	1,100	1,200
	R_RSIP_AES_Cipher_Finish	470	470	470
CBC モード暗号化	R_RSIP_AES_Cipher_Init	5,500	5,500	5,500
	R_RSIP_AES_Cipher_Update	880	990	1,100
	R_RSIP_AES_Cipher_Finish	480	480	480
CBC モード復号	R_RSIP_AES_Cipher_Init	5,500	5,500	5,500
	R_RSIP_AES_Cipher_Update	1,000	1,100	1,200
	R_RSIP_AES_Cipher_Finish	490	490	490
CTR モード	R_RSIP_AES_Cipher_Init	4,600	4,600	4,600
	R_RSIP_AES_Cipher_Update	930	1,000	1,100
	R_RSIP_AES_Cipher_Finish	500	500	500

表 1-8 AES AEAD API の性能

アルゴリズム	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
GCM モード暗号化	R_RSIP_AES_AEAD_Init	5,300	5,300	5,300
	R_RSIP_AES_AEAD_AADUpdate	430	430	430
	R_RSIP_AES_AEAD_Update	1,600	1,800	2,000
	R_RSIP_AES_AEAD_Finish	1,300	1,300	1,300
GCM モード復号	R_RSIP_AES_AEAD_Init	5,300	5,300	5,300
	R_RSIP_AES_AEAD_AADUpdate	430	430	430
	R_RSIP_AES_AEAD_Update	1,700	1,900	2,100
	R_RSIP_AES_AEAD_Verify	1,700	1,700	1,700
CCM モード暗号化	R_RSIP_AES_AEAD_Init	240	240	240
	R_RSIP_AES_AEAD_LengthsSet	50	50	50
	R_RSIP_AES_AEAD_AADUpdate	4,900	4,900	4,900
	R_RSIP_AES_AEAD_Update	1,700	1,900	2,100
	R_RSIP_AES_AEAD_Finish	1,100	1,100	1,100
CCM モード復号	R_RSIP_AES_AEAD_Init	240	240	260
	R_RSIP_AES_AEAD_LengthsSet	60	60	60
	R_RSIP_AES_AEAD_AADUpdate	4,900	4,900	4,900
	R_RSIP_AES_AEAD_Update	1,600	1,800	2,000
	R_RSIP_AES_AEAD_Verify	1,600	1,600	1,600

GCM の性能は、ivec を 128bit、追加認証データを 128bit、認証タグを 128bit に固定して計測しました。
CCM の性能は、ノンスを 56bit、追加認証データを 64bit、MAC を 32bit に固定して計測しました。

表 1-9 AES MAC API の性能

アルゴリズム	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
CMAC 生成	R_RSIP_AES_MAC_Init	4,100	4,100	4,100
	R_RSIP_AES_MAC_Update	770	880	990
	R_RSIP_AES_MAC_SignFinish	980	990	990
CMAC 検証	R_RSIP_AES_MAC_Init	4,100	4,100	4,100
	R_RSIP_AES_MAC_Update	770	880	990
	R_RSIP_AES_MAC_VerifyFinish	1,500	1,500	1,500

表 1-10 ECC API の性能

アルゴリズム	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
NIST secp256r1	R_RSIP_ECDSA_Sign	9,400,000	9,400,000	9,400,000
	R_RSIP_ECDSA_Verify	4,400,000	4,600,000	4,500,000
Koblitz secp256k1	R_RSIP_ECDSA_Sign	9,400,000	9,400,000	9,400,000
	R_RSIP_ECDSA_Verify	4,400,000	4,500,000	4,500,000
Brainpool p256r1	R_RSIP_ECDSA_Sign	9,400,000	9,400,000	9,400,000
	R_RSIP_ECDSA_Verify	4,400,000	4,400,000	4,400,000

表 1-11 KDF API の性能

API	性能 (単位：サイクル)
R_RSIP_PKI_ECDSA_CertVerify	5,200,000
R_RSIP_PKI_CertKeyImport	810,000
R_RSIP_PKI_VerifiedCertInfoExport	70
R_RSIP_PKI_VerifiedCertInfoImport	80
R_RSIP_ECDH_KeyAgree	9,100,000
R_RSIP_ECDH_PlainKeyAgree	9,100,000
R_RSIP_KDF_SHA_Init	70
R_RSIP_KDF_SHA_ECDHSecretUpdate	9,600
R_RSIP_KDF_SHA_Update	90
R_RSIP_KDF_SHA_Suspend	230
R_RSIP_KDF_SHA_Resume	210
R_RSIP_KDF_SHA_Finish	6,900
R_RSIP_DKMConcatenate	110
R_RSIP_KDF_DerivedKeyImport	30,000
R_RSIP_KDF_DerivedIVWrap	30,000

表 1-12 HASH API の性能

アルゴリズム	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
SHA224	R_RSIP_SHA_Compute	2,000	3,300	3,300
	R_RSIP_SHA_Init	50	50	50
	R_RSIP_SHA_Update	130	140	1,900
	R_RSIP_SHA_Finish	2,000	3,300	3,300
SHA256	R_RSIP_SHA_Compute	2,000	3,300	3,300
	R_RSIP_SHA_Init	50	50	50
	R_RSIP_SHA_Update	130	140	1,900
	R_RSIP_SHA_Finish	2,000	3,300	2,000
-	R_RSIP_SHA_Suspend	230		
	R_RSIP_SHA_Resume	210		

表 1-13 HMAC API の性能

アルゴリズム	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
HMAC-SHA224 生成	R_RSIP_HMAC_Compute	11,000	12,000	12,000
	R_RSIP_HMAC_Init	110	110	110
	R_RSIP_HMAC_Update	120	140	7,100
	R_RSIP_HMAC_SignFinish	11,000	12,000	5,300
HMAC-SHA224 検証	R_RSIP_HMAC_Verify	11,000	13,000	13,000
	R_RSIP_HMAC_Init	110	110	120
	R_RSIP_HMAC_Update	120	130	7,100
	R_RSIP_HMAC_VerifyFinish	11,000	13,000	6,100
HMAC-SHA256 生成	R_RSIP_HMAC_Compute	11,000	12,000	12,000
	R_RSIP_HMAC_Init	110	110	110
	R_RSIP_HMAC_Update	120	130	7,100
	R_RSIP_HMAC_SignFinish	11,000	12,000	5,100
HMAC-SHA256 検証	R_RSIP_HMAC_Verify	11,000	13,000	13,000
	R_RSIP_HMAC_Init	110	110	110
	R_RSIP_HMAC_Update	120	130	7,100
	R_RSIP_HMAC_VerifyFinish	11,000	13,000	6,100
-	R_RSIP_HMAC_Suspend	670		
	R_RSIP_HMAC_Resume	260		

表 1-14 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_RSIP_RFC3394_KeyWrap	16,000	23,000
R_RSIP_RFC3394_KeyUnwrap	18,000	25,000

表 1-15 セキュアブート/ファームウェアアップデート API 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_RSIP_SB_MAC_Verify_Init	2,100	2,100	2,100
R_RSIP_SB_MAC_Verify_Update	18,000	36,000	53,000
R_RSIP_SB_MAC_Verify_Finish	18,000	36,000	54,000

2. API 情報

2.1 ハードウェアの要求

RSIP PM ドライバは、RSIP 搭載デバイスでのみ使用可能です。RSIP 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

RSIP PM ドライバは、以下モジュールに依存します。

- r_bsp V7.51 以降をご使用ください。(BSP=Board Support Package)

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

```
/* Chip version.
   Character(s) = Value for macro =
   A          = 0xA           = Chip version A
                              = Encryption module not included, USB included, CAN
FD included (only CAN 2.0 protocol supported)
   B          = 0xB           = Chip version B
                              = Encryption module and USB included, CAN FD included
*/
#define BSP_CFG_MCU_PART_VERSION       (0xB)
```

2.3 サポートされているツールチェイン

RSIP PM 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_rsip_protected_rx_if.h に記載しています。

2.5 整数型

RSIP PM 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

本モジュールのコンフィグレーションオプションの設定は/r_config/r_rsip_protected_rx_config.h で行います。オプション名および設定値に関する説明を以下の表に示します。

設定値に関する説明を以下の表に示します。

表 2-1　/r_config/r_rsip_protected_rx_config.h　の定義

オプション名	デフォルト値	設定値の意味
RSIP_CFG_PARAM_CHECKING_ENABLE	BSP_CFG_PARAM_CHECKING_ENABLE	パラメータチェックを有効 1:有効 0:無効
RSIP_CFG_AES_128_ENABLE	0	AES 128bit を有効 1:有効 0:無効
RSIP_CFG_AES_256_ENABLE	0	AES 256bit を有効 1:有効 0:無効
RSIP_CFG_AES_ECB_CBC_CTR_ENABLE	0	AES ECB/CBC/CTR モードを有効 1:有効 0:無効
RSIP_CFG_AES_GCM_ENABLE	0	AES GCM モードを有効 1:有効 0:無効
RSIP_CFG_AES_CCM_ENABLE	0	AES CCM モードを有効 1:有効 0:無効
RSIP_CFG_AES_CMAC_ENABLE	0	AES CMAC を有効 1:有効 0:無効
RSIP_CFG_ECC_SECP256R1_ENABLE	0	ECC secp256r1 を有効 1:有効 0:無効
RSIP_CFG_ECC_BRAINPOOLP256R1_ENABLE	0	ECC braimpoolp256r1 を有効 1:有効 0:無効
RSIP_CFG_ECC_SECP256K1_ENABLE	0	ECC secp256k1 を有効 1:有効 0:無効
RSIP_CFG_SHA224_ENABLE	0	SHA224 を有効 1:有効 0:無効
RSIP_CFG_SHA256_ENABLE	0	SHA256 を有効 1:有効 0:無効
RSIP_CFG_HMAC_SHA224_ENABLE	0	HMAC-SHA224 を有効 1:有効 0:無効

オプション名	デフォルト値	設定値の意味
RSIP_CFG_HMAC_SHA256_ENABLE	0	HMAC-SHA256 を有効 1:有効 0:無効
RSIP_CFG_KDF_SHA256_ENABLE	0	KDF-SHA256 を有効 1:有効 0:無効
RSIP_CFG_SECURE_BOOT	0	セキュアブート機能を有効 1:有効 0:無効
RSIP_CFG_FIRMWARE_UPDATE	0	ファームウェアアップデート機能を有効 1:有効 0:無効

2.7　型定義

RSIP PM ドライバで使用している型の定義を示します。

表 2-2　RSIP PM ドライバ型の定義

定義	データ型	意味
rsip_ctrl_t	void	RSIP PM ドライバ管理構造体の API 引数用型定義 引数に使用するデータ型は rsip_instance_ctrl_t を使用してください。

2.8　構造体

RSIP PM ドライバで使用している構造体の定義を示します。

表 2-3　RSIP PM ドライバ構造体定義

定義	意味
rsip_instance_ctrl_t	RSIP PM ドライバ管理構造体のインスタンス
rsip_cfg_t	コンフィグレーション構造体 RX RSIP PM ドライバでは使用していません。
rsip_wrapped_key_t	Wrapped Key
rsip_sha_handle_t	SHA 演算のワーク領域
rsip_hmac_handle_t	HMAC 演算のワーク領域
rsip_wrapped_secret_t	ラップした共有秘密
rsip_kdf_sha_handle_t	KDF-SHA 演算のワーク領域
rsip_wrapped_dkm_t	ラップした DKM
rsip_verified_cert_info_t	検証された証明書の情報

2.9　列挙体

RSIP PM ドライバで使用している列挙体の定義を示します。

表 2-4 鍵の種類 enum rsip_key_type_t

列挙子	値	意味
RSIP_KEY_TYPE_INVALID	0x00000000	無効
RSIP_KEY_TYPE_AES_128	0x10000004	AES128bit 鍵
RSIP_KEY_TYPE_AES_256	0x10000008	AES256bit 鍵
RSIP_KEY_TYPE_ECC_SECP256R1_PUBLIC	0x04000008	secp256r1 公開鍵
RSIP_KEY_TYPE_ECC_SECP256K1_PUBLIC	0x04030008	secp256k1 公開鍵
RSIP_KEY_TYPE_ECC_BRAINPOOL_P256R1_PUBLIC	0x04030004	brainpoolP256r1 公開鍵
RSIP_KEY_TYPE_ECC_SECP256R1_PRIVATE	0x05000008	secp256r1 秘密鍵
RSIP_KEY_TYPE_ECC_SECP256K1_PRIVATE	0x05030008	secp256k1 秘密鍵
RSIP_KEY_TYPE_ECC_BRAINPOOL_P256R1_PRIVATE	0x05030004	brainpoolP256r1 秘密鍵
RSIP_KEY_TYPE_HMAC_SHA224	0x08010008	HMAC SHA224 鍵
RSIP_KEY_TYPE_HMAC_SHA256	0x08020008	HMAC SHA256 鍵
RSIP_KEY_TYPE_KUK	0xff000008	Key Update Key

表 2-5 ハッシュの種類 enum rsip_hash_type_t

列挙子	値	意味
RSIP_HASH_TYPE_SHA224	1	SHA224
RSIP_HASH_TYPE_SHA256	2	SHA256

表 2-6 AES 暗号モードの種類 enum rsip_aes_cipher_mode_t

列挙子	値	意味
RSIP_AES_CIPHER_MODE_ECB_ENC	0	AES ECB モード 暗号化
RSIP_AES_CIPHER_MODE_ECB_DEC	1	AES ECB モード 復号
RSIP_AES_CIPHER_MODE_CBC_ENC	2	AES CBC モード 暗号化
RSIP_AES_CIPHER_MODE_CBC_DEC	3	AES CBC モード 復号
RSIP_AES_CIPHER_MODE_CTR	4	AES CTR モード
RSIP_AES_CIPHER_MODE_CBC_ENC_WRAPPED_IV	7	ラップした初期化ベクタを使用した AES CBC モード 暗号化
RSIP_AES_CIPHER_MODE_CBC_DEC_WRAPPED_IV	8	ラップした初期化ベクタを使用した AES CBC モード 復号

表 2-7 AES AEAD モードの種類 enum rsip_aes_aead_type_t

列挙子	値	意味
RSIP_AES_AEAD_MODE_GCM_ENC	0	AES GCM モード 暗号化
RSIP_AES_AEAD_MODE_GCM_DEC	1	AES GCM モード 復号
RSIP_AES_AEAD_MODE_CCM_ENC	2	AES CCM モード 暗号化
RSIP_AES_AEAD_MODE_CCM_DEC	3	AES CCM モード 復号
RSIP_AES_AEAD_MODE_GCM_ENC_WRAPPED_IV	4	ラップした初期化ベクタを使用した AES GCM モード 暗号化
RSIP_AES_AEAD_MODE_GCM_DEC_WRAPPED_IV	5	ラップした初期化ベクタを使用した AES GCM モード 復号

表 2-8 AES MAC モードの種類 enum rsip_mac_type_t

列挙子	値	意味
RSIP_AES_MAC_MODE_CMAC	0	AES CMAC モード

表 2-9 初期化ベクタの種類 enum rsip_initial_vector_type_t

列挙子	値	意味
RSIP_INITIAL_VECTOR_TYPE_AES_16_BYTE	0	AES 用 16 バイト初期化ベクタ

表 2-10 wrappted_key_t 構造体のサイズ enum rsip_byte_size_wrapped_key_t

列挙子	値	意味
RSIP_BYTE_SIZE_WRAPPED_KEY_AES_128	36U	AES128 鍵のバイトサイズ
RSIP_BYTE_SIZE_WRAPPED_KEY_AES_256	52U	AES256 鍵のバイトサイズ
RSIP_BYTE_SIZE_WRAPPED_KEY_ECC_SECP256R1_PUBLIC	84U	secp256r1 公開鍵のバイトサイズ
RSIP_BYTE_SIZE_WRAPPED_KEY_ECC_SECP256K1_PUBLIC	84U	secp256k1 公開鍵のバイトサイズ
RSIP_BYTE_SIZE_WRAPPED_KEY_ECC_BRAINPOOL256R1_PUBLIC	84U	brainpoolP256r1 公開鍵のバイトサイズ
RSIP_BYTE_SIZE_WRAPPED_KEY_ECC_SECP256R1_PRIVATE	52U	secp256r1 秘密鍵のバイトサイズ
RSIP_BYTE_SIZE_WRAPPED_KEY_ECC_SECP256K1_PRIVATE	52U	secp256k1 秘密鍵のバイトサイズ
RSIP_BYTE_SIZE_WRAPPED_KEY_ECC_BRAINPOOL256R1_ PRIVATE	52U	brainpoolP256r1 秘密鍵のバイトサイズ
RSIP_BYTE_SIZE_WRAPPED_KEY_HMAC_SHA224	52U	HMAC_SHA224 鍵のバイトサイズ
RSIP_BYTE_SIZE_WRAPPED_KEY_HMAC_SHA256	52U	HMAC_SHA256 鍵のバイトサイズ
RSIP_BYTE_SIZE_WRAPPED_KEY_KUK	52U	Key Update Key のバイトサイズ

2.10 戻り値

以下に RSIP PM 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_bsp/mcu/all/fsp_common_api.h で fsp_err_t として定義されています。

表 2-11　戻り値 enum fsp_err_t

列挙子	値	意味
FSP_SUCCESS	0x00000	正常終了
FSP_ERR_ASSERTION	0x00001	必要な入力引数が NULL
FSP_ERR_INVALID_ARGUMENT	0x00003	入力引数が不正
FSP_ERR_UNSUPPORTED	0x00006	選択されたモードが非サポート
FSP_ERR_NOT_OPEN	0x00007	open されていない
FSP_ERR_ALREADY_OPEN	0x0000e	既に open されている
FSP_ERR_NOT_ENABLED	0x00013	指定した処理が実施できない
FSP_ERR_INVALID_SIZE	0x00017	入力サイズが不正
FSP_ERR_INVALID_STATE	0x0001e	呼び出し状態が不正
FSP_ERR_CRYPTO_RSIP_RESOURCE_CONFLICT	0x10100	HW のリソース衝突が発生
FSP_ERR_CRYPTO_RSIP_FATAL	0x10101	HW の致命的エラー
FSP_ERR_CRYPTO_RSIP_FAIL	0x10102	内部エラー
FSP_ERR_CRYPTO_RSIP_KEY_SET_FAIL	0x10103	入力した鍵の種類が不正
FSP_ERR_CRYPTO_RSIP_AUTHENTICATION	0x10104	検証失敗

2.11 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) e² studio 上で Smart Configurator を使用して FIT モジュールを追加する場合
e² 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 e² 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) e² studio 上で FIT Configurator を使用して FIT モジュールを追加する場合
e² 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 e² 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 e² 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。

3. RSIP PM ドライバの使用方法

RX ファミリ RSIP PM 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

RSIP PM ドライバが扱う鍵（入力する鍵、出力する鍵）は、RSIP 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 RSIP PM ドライバではこの不透明な鍵を Wrapped Key と呼びます。RSIP PM 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、RSIP の外部で鍵の秘匿と改ざん検知を実現します。

RSIP 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。

RSIP PM 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-Finish のよう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

RSIP 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。

RSIP PM 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_RSIP_Close()で RSIP PM ドライバを一度終了して、R_RSIP_Open()で RSIP 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 RSIP へのアクセス衝突回避

RSIP PM ドライバは、API 実行中に RSIP のハードウェア資源を占有します。マルチパート演算でも、提供する API は一連のマルチパート演算が終了するまで RSIP 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 RSIP PM ドライバを利用する際は、RSIP 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) RSIP PM ドライバ API 実行中に他の RSIP PM ドライバの API を実行することはできません。
2) マルチパート演算を提供する API では、現在処理中の一連の演算処理(Init~Finish または Verify)が完了するまでは他の RSIP PM ドライバ API を実行することができません。

RSIP PM ドライバの API でアクセス衝突が発生した場合、API は FSP_ERR_INVALID_STATE または FSP_ERR_CRYPTO_RSIP_RESOURCE_CONFLICT を返します。

3.3　BSP FIT モジュールの組込み

RSIP PM ドライバは、RSIP のモジュールストップ解除と設定のため、R_RSIP_Open()と R_RSIP_Close()で BSP FIT モジュールの以下の API を使用します。お客様のプログラムで BSP FIT を使用されない場合でも、BSP FIT を組み込んで使用してください。

・R_BSP_RegisterProtectEnable()

・R_BSP_RegisterProtectDisable()

R_BSP_RegisterProtectEnable()、R_BSP_RegisterProtectDisable()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。

また R_RSIP_Open()は呼び出される前に、BSP FIT 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 FIT のスタートアップを使用しない場合、R_RSIP_Open()を呼び出す前に R_BSP_StartupOpen()を呼び出してください。R_BSP_StartupOpen()は、R_RegisterProtectEnable()、R_RegisterProtectDisable()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

RSIP PM ドライバが提供する API には、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。

マルチパート演算では、1 つの暗号演算を Init-Update-Final 等のステップに分割して実行する API を提供しています。この構成により、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。

それぞれのマルチパート演算の API 仕様については、4.2章を参照してください。

マルチパート演算の API を呼び出した場合、RSIP PM ドライバの内部状態により、次に呼び出される API を管理します。適切な API が呼び出されなかった場合には、戻り値としてエラーが返されます。

3.5　初期化と終了、および状態遷移

本ドライバは、ドライバ管理のための共通機能 API を提供します。

No.	API	説明
1	R_RSIP_Open	RSIP PM ドライバのオープン処理を行います。 RSIP の初期化、RSIP 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_RSIP_Close	RSIP PM ドライバの終了処理を行います。
3	R_RSIP_GetVersion	RSIP PM 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では、最初に R_RSIP_Open() を呼び出して、RSIP ならびにドライバの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_RSIP_Close()を呼び出してください。

RSIP PM ドライバを使用中に何らかの問題が発生し、RSIP PM ドライバとその制御対象である RSIP をリセットしてから処理を再開したい場合は、R_RSIP_Close()を呼び出した後に R_RSIP_Open()を呼び出してください。

R_RSIP_Open()では、RSIP 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。

RSIP PM ドライバのバージョンを取得したい場合は、R_RSIP_GetVersion()を呼び出します。

RSIP PM ドライバでは、RSIP を使用可能な状態を管理するために 5 つの内部状態を保持しています。

状態名	説明
Close	RSIP PM ドライバ使用できない状態。R_RSIP_Open()を呼び出す前。 どの状態からでも R_RSIP_Close()を呼び出すと、Close 状態に遷移します。
Main	R_RSIP_Open()呼出し後の RSIP PM ドライバ使用可能状態。 マルチパート、シングルパート演算終了後は本状態に戻ります。
Each Algorithm	マルチパート演算の Init API 呼び出し後遷移します。 Init API で呼んだアルゴリズムのマルチパート演算が実行できる状態。
Firmware Update	R_RSIP_FWUP_StartUpdateFirmware()呼び出し後遷移します。 R_RSIP_FWUP_MAC_Sign_Init/Update/Finish()が呼び出せる状態。
Stop	R_RSIP_FWUP_MAC_Sign_Finish()の実行が終了すると遷移します。

図 3-1は RSIP PM ドライバの状態遷移図になります。

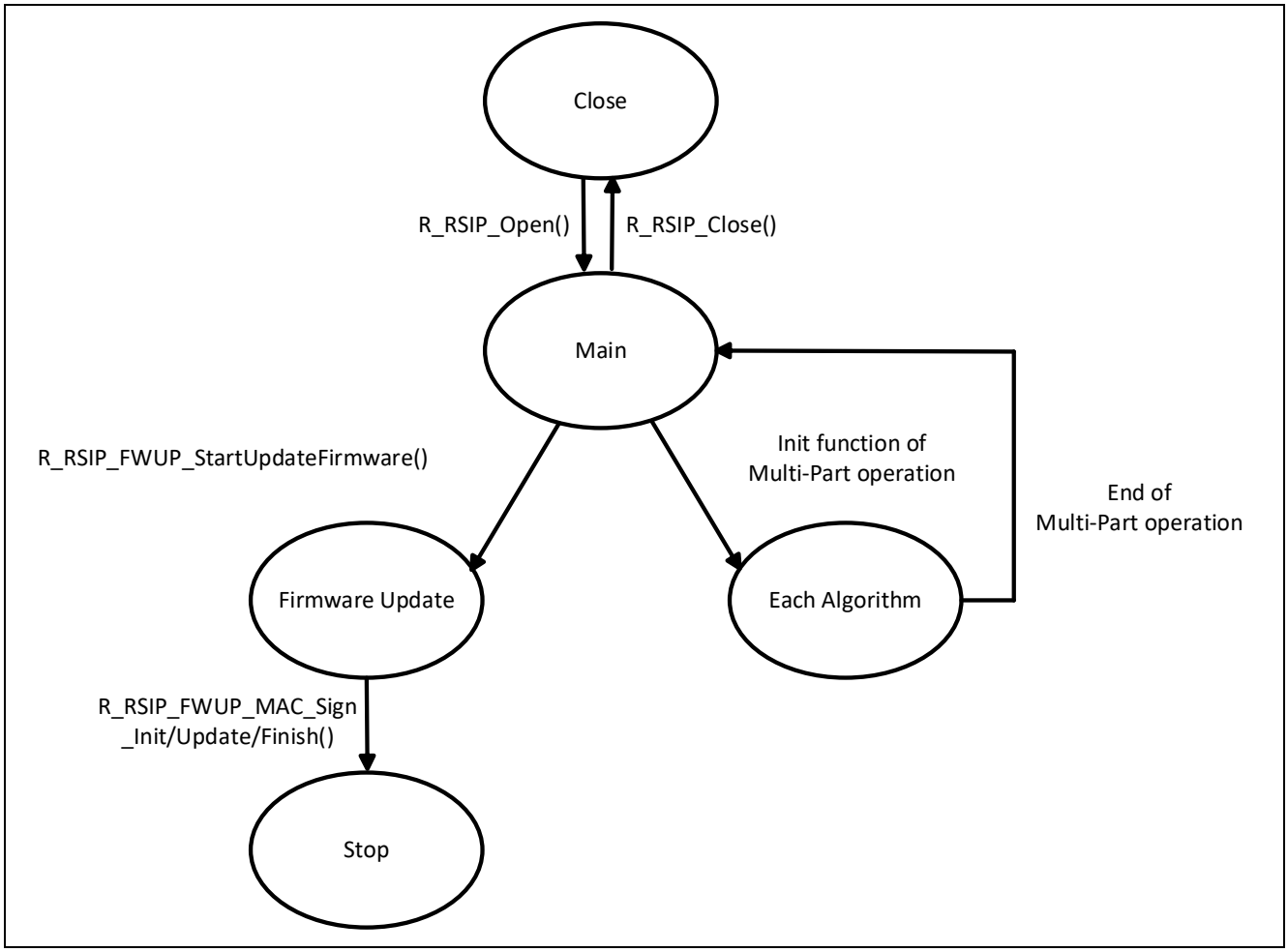

図 3-1 RSIP PM ドライバの状態遷移図

3.6 鍵の管理

本ドライバは、以下の鍵管理のための API を提供します、

No.	API	説明
1	R_RSIP_EncryptedKeyWrap R_RSIP_InitialKeyWrap	鍵の更新と注入
2	R_RSIP_KeyGenerate R_RSIP_KeyPairGenerate	鍵の生成
3	R_RSIP_PublicKeyExport	平文公開鍵の抽出

図 3-2に RSIP PM ドライバの暗号操作における鍵の取り扱いを示します。

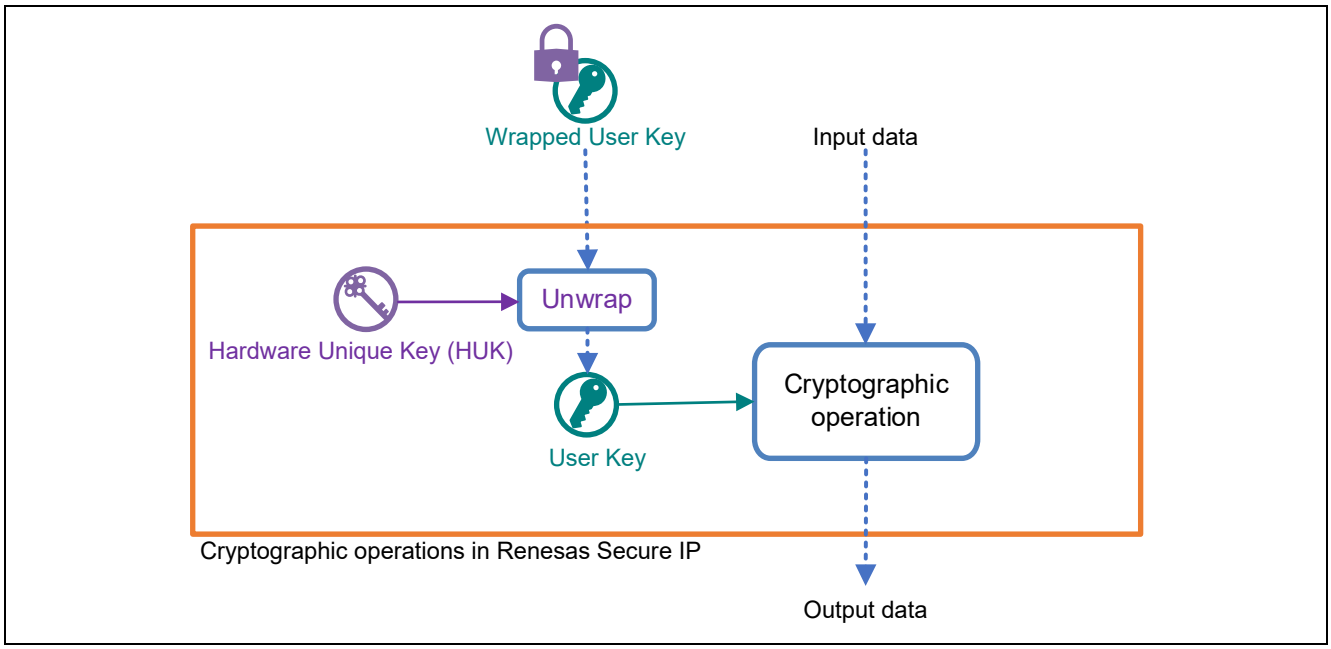

図 3-2 RSIP PM ドライバ暗号操作における鍵の取り扱い

RSIP PM ドライバの暗号演算で入出力する鍵は、RSIP 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 RSIP PM ドライバでは Wrapped Key と呼びます。ただし、非対称鍵暗号で使用する公開鍵の Wrapped Key は、平文の公開鍵に RSIP PM ドライバで使用する鍵管理情報が付加された形式となっています。

RSIP PM 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、RSIP 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 RSIP のみであり、ラッピングが解除された鍵は、暗号処理中に RSIP 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 のラッピングを解除することはできません。

3.6.1 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 3-3に Renesas Key Wrap Service を含む鍵の注入と更新のフローを示します。

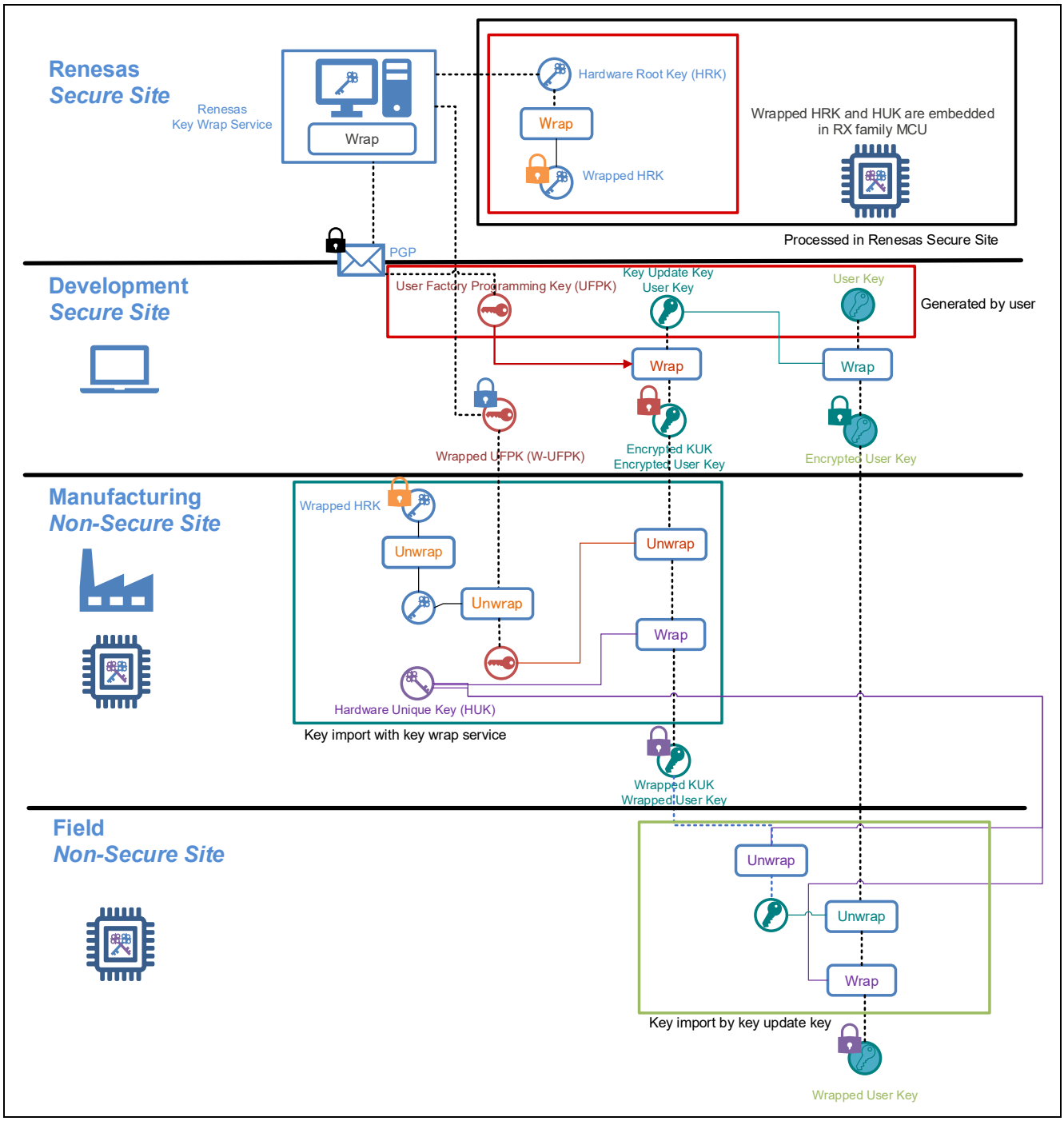

図 3-3 鍵の注入と更新のフロー

鍵注入時には、お客様のセキュアなサイト(図 3-3 Development Secure Site)で、ユーザ鍵、KUK、UFPK の生成を行い、ユーザ鍵、KUK を UFPK でラップしてください。また、ラップに使用した UFPK は、Renesas Key Wrap Service を使用して、W-UFPK を生成してください。鍵更新時には、お客様のセキュアなサイト(図 3-3 Development Secure Site)で、更新するユーザ鍵を生成し、更新するユーザ鍵を KUK でラップしてください。鍵注入と鍵更新時の鍵のラップには Security Key Management Tool を使用可能です。

3.6.1.1 鍵のラップアルゴリズム

鍵注入および鍵更新で使用する、ユーザ鍵を UFPK や KUK でラップする際のユーザ鍵ラップ方式を図 3-4 に示します。UFPK または KUK の上位 128bit を CBC Key、下位 128bit を CBC-MAC Key として、ユーザ鍵もしくは KUK のラップに使用します。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、5.3 3.6.1.1ユーザ鍵暗号化フォーマットを参照してください。

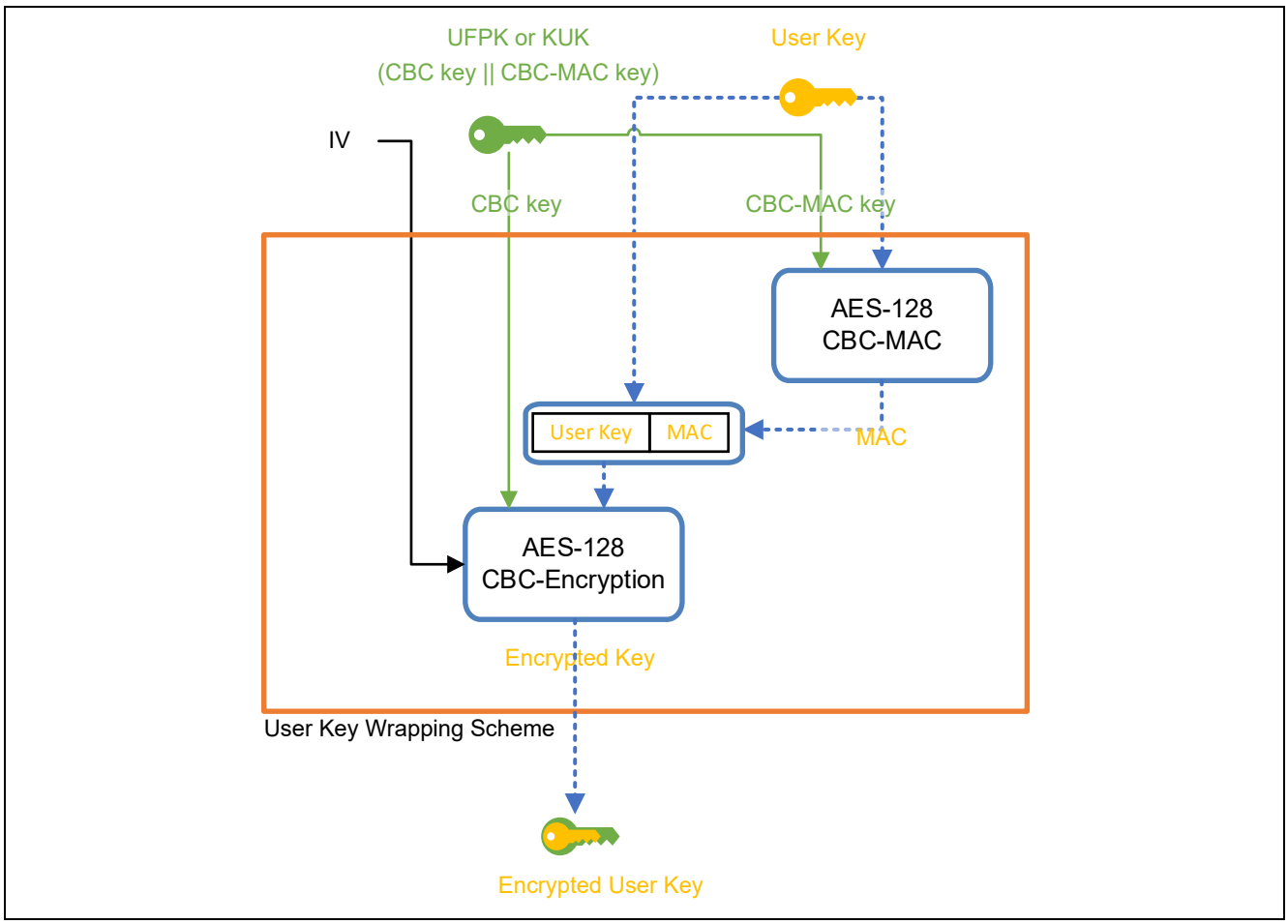

図 3-4 鍵注入および鍵更新時のユーザ鍵ラップ方式

User Key のラップの具体的な計算式は以下になります。

```
---------------------------------------------------------------------------------------------------------------------------------------
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。

3.6.2 鍵の生成

鍵の生成は、RSIP の乱数生成機能を用いて鍵を生成し、RSIP PM ドライバが利用できる Wrapped Key の形式で出力します。

対称鍵暗号に使用する鍵の Wrapped Key を生成する場合は、出力する鍵のタイプを指定して R_RSIP_KeyGenerate()を呼び出します。

非対称鍵暗号に使用する鍵ペアの Wrapped Key を生成する場合は、出力する鍵のタイプを指定して R_RSIP_KeyPairGenerate()を呼び出します。

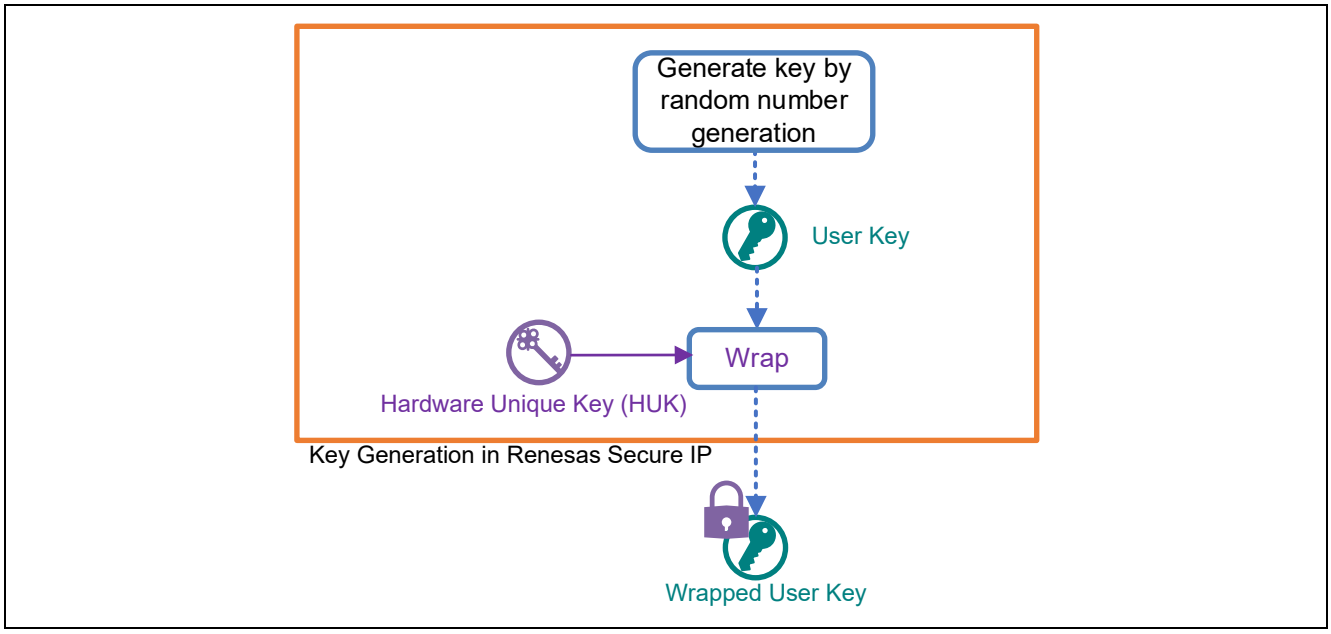

図 3-5 鍵生成のフロー

3.6.3 平文の公開鍵の抽出

公開鍵の Wrapped Key は、平文の公開鍵に RSIP PM ドライバで使用する鍵管理情報が付加された形式となっています。Wrapped Key から平文の公開鍵のみを抽出する場合は、R_RSIP_PublicKeyExport()を呼び出してください。抽出される鍵のフォーマットは4.2.2.5 R_RSIP_PublicKeyExportを参照してください。

3.7　乱数生成

本ドライバは、以下の乱数生成 API を提供します。

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

3.8　対称鍵暗号

本ドライバは、以下の対称鍵暗号演算 API を提供します。

No.	API	説明
1	R_RSIP_AES_Cipher_Init R_RSIP_AES_Cipher_Update R_RSIP_AES_Cipher_Finish	対称鍵暗号 AES 128/256bit: ECB, CBC, CTR 暗号/復号
2	R_RSIP_AES_AEAD_Init R_RSIP_AES_AEAD_LengthsSet R_RSIP_AES_AEAD_AADUpdate R_RSIP_AES_AEAD_Update R_RSIP_AES_AEAD_Finish R_RSIP_AES_AEAD_Verify	認証付き暗号 (AEAD) AES 128/256bit GCM, CCM 暗号/復号
3	R_RSIP_AES_MAC_Init R_RSIP_AES_MAC_Update R_RSIP_AES_MAC_SignFinish R_RSIP_AES_MAC_VerifyFinish	メッセージ認証コード (MAC) AES-CMAC 128/256bit MAC 生成/検証

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

対称鍵暗号の処理は、以下のように行います。

R_RSIP_AES_Cipher_Init()を呼び出して、暗号モードと必要な鍵と初期化ベクタを指定します。

R_RSIP_AES_Cipher_Update()関数を、連続したブロック単位の平文もしくは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。

暗号演算処理を完了するには、R_RSIP_AES_Cipher_Finish()を呼び出します。

3.8.2　認証付き暗号 (AEAD)

認証付き暗号の処理は、以下のように行います。

R_RSIP_AES_AEAD_Init()を呼び出して、暗号モードと必要な鍵と初期化ベクタを指定します。

暗号モードが CCM の場合には、R_RSIP_AES_AEAD_LengthsSet()を呼び出して、入力するデータのサイズを指定します。

R_RSIP_AES_AEAD_AADUpdate()を呼び出して、追加認証データを指定します。

R_RSIP_AES_AEAD_Update()関数を、連続したブロック単位のメッセージをまとめたデータに対して呼び出します。

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

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_RSIP_AES_AEAD_Verify()を呼び出します。

3.8.3 メッセージ認証コード (MAC)

メッセージ認証コードの処理は、以下のように行います。

R_RSIP_AES_MAC_Init()を呼び出して、MAC 演算モードと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_RSIP_AES_MAC_Update()関数を呼び出します。

MAC 生成演算の場合には、R_RSIP_AES_MAC_SignFinish()を呼び出して、MAC データを得て、暗号演算処理を終了します。

メッセージの MAC 検証を完了するには、R_RSIP_AES_MAC_VerifyFinish()を呼び出します。

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_RSIP_ECDSA_Sign R_RSIP_ECDSA_Verify	ECDSA の署名生成/検証を行います。

非対称鍵暗号の API は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_RSIP_SHA_Compute R_RSIP_SHA_Init R_RSIP_SHA_Update R_RSIP_SHA_Finish R_RSIP_SHA_Suspend R_RSIP_SHA_Resume	メッセージダイジェスト SHA-224/256
2	R_RSIP_HMAC_Compute R_RSIP_HMAC_Verify R_RSIP_HMAC_Init R_RSIP_HMAC_Update R_RSIP_HMAC_SignFinish R_RSIP_HMAC_VerifyFinish R_RSIP_HMAC_Suspend R_RSIP_HMAC_Resume	メッセージ認証コード (HMAC) HMAC-SHA224/256

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

メッセージダイジェスト生成の処理は、以下のように行います。

シングルパート演算を実行する場合は、R_RSIP_SHA_Compute()を呼び出してハッシュ演算を行います。

マルチパート演算を実行する場合は、R_RSIP_SHA_Init()を呼び出して、ハッシュ演算モードを指定します。

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

R_RSIP_SHA_Update()を呼び出した後で、R_RSIP_SHA_Suspend()を呼び出すことで、ハッシュ演算のマルチパート演算を中断することが可能です。ハッシュ演算を再開する場合、R_RSIP_SHA_Resume()を呼び出した後､再び R_RSIP_SHA_Update()もしくは R_RSIP_SHA_Finish()を呼び出して､ハッシュ演算を行ってください。

R_RSIP_SHA_Suspend()を呼び出した後、R_RSIP_SHA_Resume()を呼びださずに R_RSIP_SHA_Finish()を呼び出すことで､ハッシュ演算の途中データの計算結果を出力することが可能です。

3.10.2 メッセージ認証コード (HMAC)

メッセージ認証コードの処理は、以下のように行います。

シングルパート演算を実行する場合は、HMAC を生成するには R_RSIP_HMAC_Compute()を、HMAC を検証するには R_RSIP_HMAC_Verify()を呼び出します。

マルチパート演算を実行する場合は、R_RSIP_HMAC_Init()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_RSIP_AES_HMAC_Update()を呼び出します。

HMAC 生成を完了するには、R_RSIP_HMAC_SignFinish()を呼び出します。

メッセージの HMAC を検証するには、R_RSIP_HMAC_VerifyFinish()を呼び出して、検証する HMAC データ入力します。

R_RSIP_HMAC_Update()を呼び出した後で、R_RSIP_HMAC_Suspend()を呼び出すことで、HMAC 演算のマルチパート演算を中断することが可能です。HMAC 演算を再開する場合、R_RSIP_HMAC_Resume()を呼び出した後、再び R_RSIP_HMAC_Update()もしくは R_RSIP_HMAC_SignFinish()または R_RSIP_HMAC_VerifyFinish()を呼び出して、HMAC 演算を行ってください。

3.11 Key Wrap

本ドライバは、以下の Key Wrap のための API を提供します。

No.	API	説明
1	R_RSIP_RFC3394_KeyWrap	RFC3394 に準拠したアルゴリズムで鍵をラップします。
2	R_RSIP_RFC3394_KeyUnwrap	RFC3394 に準拠したアルゴリズムで鍵をアンラップします。

鍵をラップする場合は、R_RSIP_RFC3394_KeyWrap()を呼び出して、ラップに使用する鍵とラップする鍵を指定して、ラップされた鍵を得ます。

鍵をアンラップする場合は､R_RSIP_RFC3394_KeyWrap()を呼び出して､アンラップに使用する鍵とラップする鍵と出力する Wrapped Key の種類を指定して、アンラップされた鍵の Wrapped Key を得ます。

3.12 鍵交換・鍵派生

本ドライバは、以下の鍵交換・鍵派生のための API を提供します。

No.	API	説明
1	R_RSIP_PKI_ECDSA_CertVerify R_RSIP_PKI_CertKeyImport R_RSIP_PKI_VerifiedCertInfoExport R_RSIP_PKI_VerifiedCertInfoImport R_RSIP_ECDH_KeyAgree R_RSIP_ECDH_PlainKeyAgree	KDF-SHA256 で鍵交換を行います。
2	R_RSIP_KDF_SHA_Init R_RSIP_KDF_SHA_ECDHSecretUpdate R_RSIP_KDF_SHA_Update R_RSIP_KDF_SHA_Finish R_RSIP_KDF_SHA_Suspend R_RSIP_KDF_SHA_Resume R_RSIP_KDF_DKMConcatenate R_RSIP_KDF_DerivedKeyImport R_RSIP_KDF_DerivedIVWrap	KDF-SHA256 で鍵派生を行います。

公開鍵基盤(PKI)に対応し、鍵交換と鍵派生を実施します。まず鍵交換用の API により、共有秘密を生成します。そして共有秘密を使用して、鍵派生用の API により暗号鍵や初期化ベクトルを生成します。

3.12.1 公開鍵の受け取り

鍵交換相手(Peer)から、公開鍵を受け取る方法として、認証局(CA)などにより署名された公開鍵(検証された公開鍵)を入力する方法と、検証されていない公開鍵を受け取る方法があり、それぞれの方法で鍵交換を行うことができます。

3.12.1.1 検証された公開鍵の場合（証明書からの公開鍵の抽出）

検証された公開鍵を受け取るフローを図 3-6に示します。

1. 鍵共有の相手（Peer）は鍵ペアを生成し、その内の公開鍵を使用した証明書署名要求（CSR）を認証局（CA）に送り、CA が証明書を発行します。
2. RSIP を使用する製品において CA の公開鍵の Wrapped Key、CA が発行した証明書とその署名を受け取ります。
3. CA の公開鍵の Wrapped Key、証明書の署名、R_RSIP_SHA_Compute()で演算した証明書のハッシュを入力とし、R_RSIP_PKI_ECDSA_CertVerify()を呼び出し、署名の検証を行います。
   ※RSIP ドライバは、最新の検証された証明書情報のみを保持します。複数の証明書を取り扱う場合には、R_RSIP_PKI_VerifiedCertInfoExport()を呼び出して検証された証明書情報を取り出し、取り出しておいた検証された証明書情報を R_RSIP_PKI_VerifiedCertInfoImport()で取り込むことで使用してください。
4. 証明書を入力とし、R_RSIP_PKI_CertKeyImport()を呼び出し、証明書を検証して検証した証明書内の公開鍵の Wrapped Key を出力します。

※このフローで出力した公開鍵の Wrapped Key は、3.9章に記載されている非対称鍵暗号の API でも使用することができます。

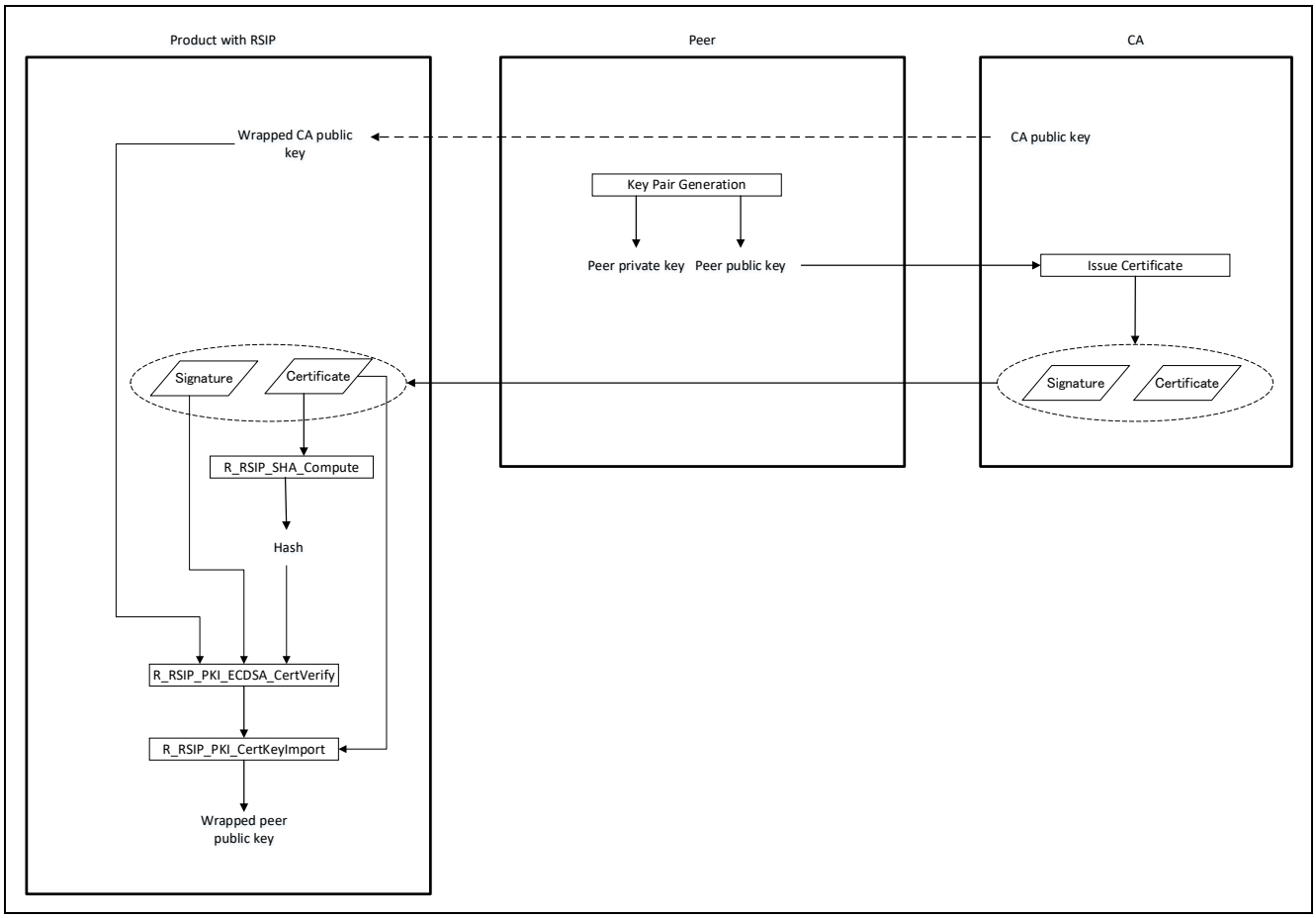

図 3-6 検証された公開鍵を受け取るフロー

3.12.1.2 検証されていない公開鍵の場合

検証されていない公開鍵を使用する場合は、Wrapped Key を出力する操作は必要ありません。検証されていない公開鍵を使用して共有秘密を生成する方法は3.12.2.2を参照してください。

ただし、検証されていない平文の公開鍵を入力とする方法は、中間者攻撃のリスクがあります。可能な場合は、検証された公開鍵を入力とする方法を使用してください。

3.12.2 共有秘密の生成

3.12.2.1 検証された公開鍵を入力とした共有秘密の生成

検証された公開鍵を入力とした共有秘密の生成のフローを図 3-7に示します。

1. 3.12.1で生成した公開鍵の Wrapped Key と、R_RSIP_KeyPiarGenerate()を呼び出して生成した秘密鍵の Wrapped Key を入力とし、R_RSIP_ECDH_KeyAgree()を呼び出してラップした共有秘密を生成します。
2. Peer に対して、R_RSIP_KeyPiarGenerate()を呼び出して生成した公開鍵の Wrapped Key を入力として R_RSIP_PublicKeyExport()を呼び出して平文の公開鍵を生成し、Peer 側での共有秘密の生成のために渡します。

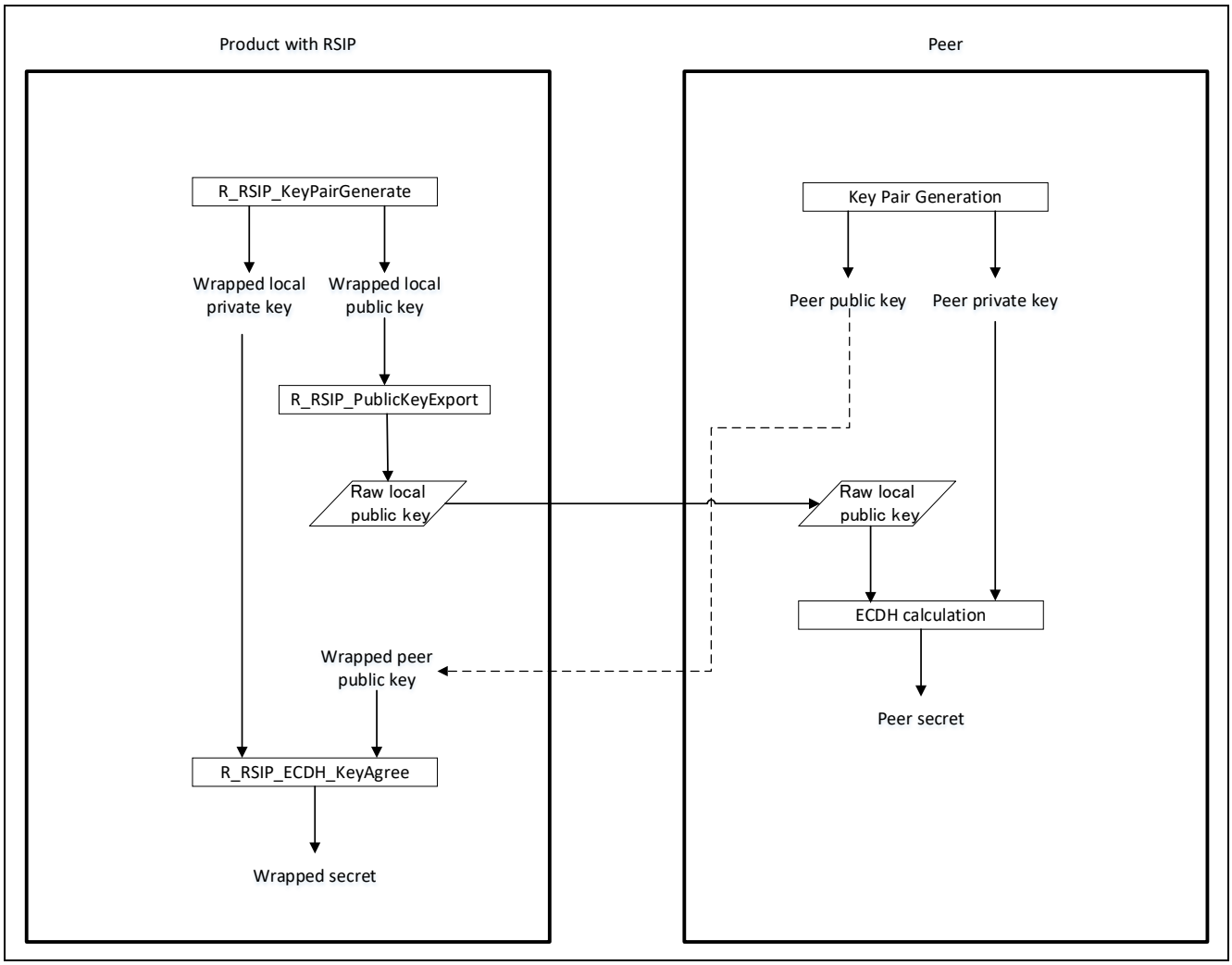

図 3-7 検証された公開鍵を入力とした共有秘密の生成のフロー

3.12.2.2 検証されていない公開鍵を入力とした共有秘密の生成

平文の公開鍵を入力とした共有秘密の生成のフローを図 3-8に示します。

1. Peer から受け取った公開鍵の Wrapped Key と、R_RSIP_KeyPiarGenerate()を呼び出して生成した秘密鍵の Wrapped Key を入力とし、R_RSIP_ECDH_PlainKeyAgree()を呼び出してラップした共有秘密を生成します。
2. Peer に対して、R_RSIP_KeyPiarGenerate()を呼び出して生成した公開鍵の Wrapped Key を入力として R_RSIP_PublicKeyExport()を呼び出して平文の公開鍵を生成し、Peer 側での共有秘密の生成のために渡します。

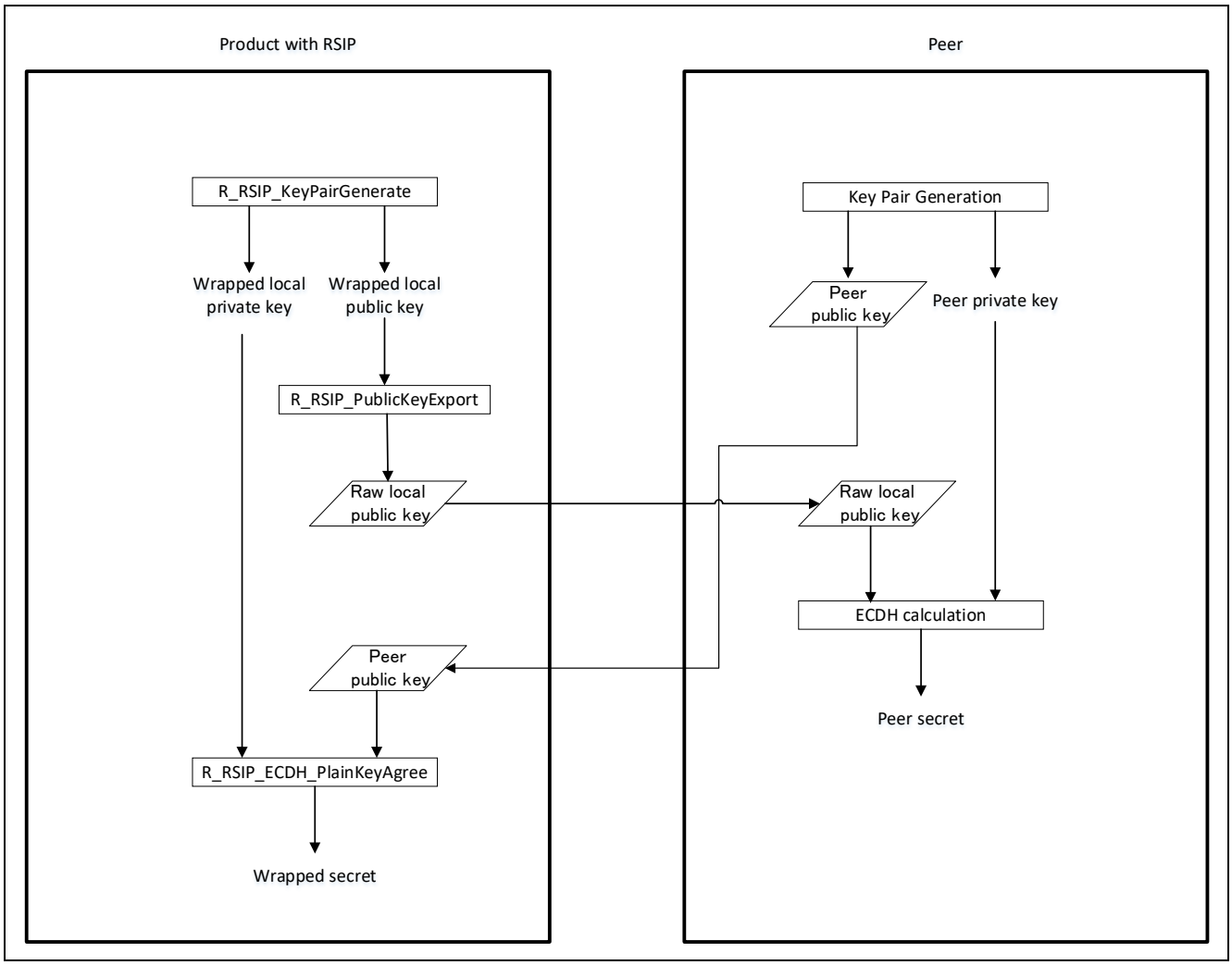

図 3-8 平文の公開鍵を入力とした共有秘密の生成のフロー

3.12.3 鍵派生

3.12.2.1で R_RSIP_ECDH_KeyAgree()から出力されたラップされた共有秘密、もしくは3.12.2.2で R_RSIP_ECDH_PlainKeyAgree()から出力されたラップされた共有秘密を入力として、R_RSIP_KDF_SHA_Init/ECDHSecretUpdate/Update/Finish()を呼び出して派生鍵材料(DKM)を生成します。DKM を連接する際は R_RSIP_KDF_DKMConcatenate()、Wrapped Key を出力する際は R_RSIP_KDF_DerivedKey_Import()、初期化ベクタを出力する際は R_RSIP_KDF_DerivedIVWrap()を呼び出します。

R_RSIP_KDF_SHA_ECDHSecretUpdate/Update()を呼び出した後で、R_RSIP_KDF_SHA_Suspend()を呼び出すことで、KDF 演算を中断することが可能です。KDF 演算を再開する場合、R_RSIP_KDF_SHA_Resume()を呼び出した後、R_RSIP_KDF_SHA_Update()もしくは R_RSIP_KDF_SHA_Finish()を呼び出して、KDF 演算を行ってください。

3.13 セキュアブート/ファームウェアアップデート

本ドライバは、以下のセキュアブートとファームウェアアップデートのための API を提供します。

No.	API	説明
1	R_RSIP_FWUP_StartUpdateFirmware R_RSIP_FWUP_MAC_Sign_Init R_RSIP_FWUP_MAC_Sign_Update R_RSIP_FWUP_MAC_Sign_Finish	暗号化されたファームウェアを復号して、MAC検証を行い、復号された平文ファームウェアのMACを生成します。
2	R_RSIP_SB_MAC_Verify_Init R_RSIP_SB_MAC_Verify_Update R_RSIP_SB_MAC_Verify_Finish	R_RSIP_FWUP_MAC_Sign_Update/Finish で復号された平文ファームウェアから生成された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に使用した鍵を安全に配送する機構を備えています。これをセキュアブートと併せて使用することで、ファームウェアアップデートを実現することができます。図 3-9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 検証フローを示します。

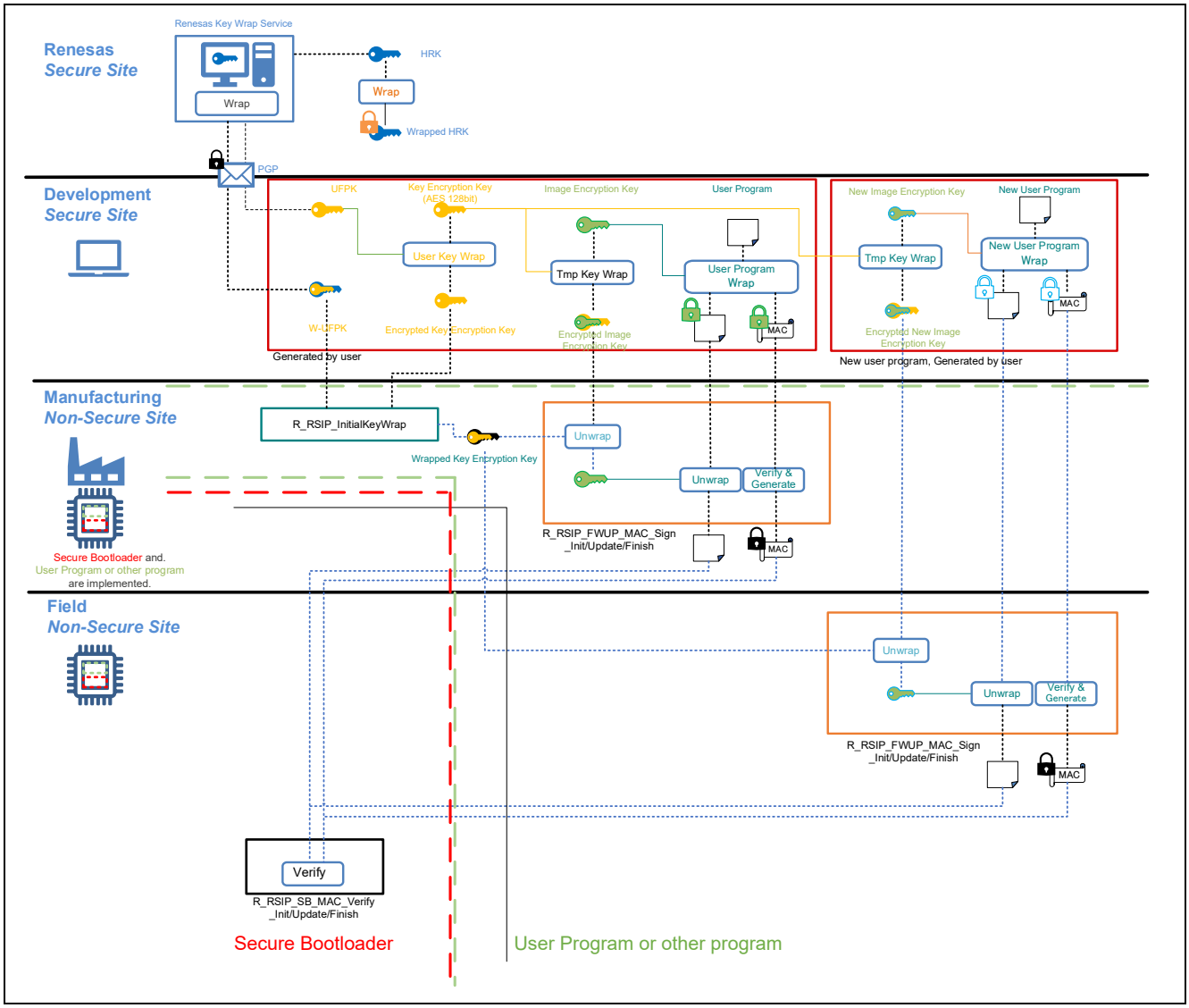

図 3-9 ファームウェアアップデートフロー

お客様のセキュアなサイト(図 3-9 Development Secure Site)で、UFPK、ユーザプログラムの暗号化で使用する鍵(Image Encryption Key)、Key Encryption Key を生成し、ユーザプログラムの暗号化と暗号化プログラムの MAC 生成を行ってください。また、Image Encryption Key を Key Encryption Key でラップしてください。ユーザプログラムの暗号化、MAC 生成、暗号化で使用した鍵のラップのアルゴリズムは3.13.3 ユーザプログラムの暗号化を参照してください。ユーザプログラムの暗号化、MAC 生成、暗号化で使用した鍵のラップには Security Key Management Tool を使用可能です。

3.13.1 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指します。リセット後にユーザプログラムを実行する前に、セキュアブートプログラムを実行し、セキュアブート後に実行されるプログラム検証します。

セキュアブートの実装には、R_RSIP_SB_MAC_VerifyInit/Update/Finish()を使用することが可能です。R_RSIP_SB_MAC_VerifyInit/Update/Finish()には、R_RSIP_FWUP_MAC_Sign_Update/Finish()から出力される平文ファームウェアと MAC 値を入力してください。

3.13.2 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とは、現在実行されているファームウェアを、新規機能追加や、不具合の改修を行うため、ファームウェアの更新を行う機能です。

ファームウェアアップデートの実装には、暗号化されたプログラムを復号して MAC 検証を行う R_RSIP_FWUP_MAC_Sign_Init/Update/Finish()を使用可能です。R_RSIP_FWUP_MAC_Sign_Finish()では、MAC 検証に成功すると、新たに HUK に紐づく鍵で、平文ファームウェアの MAC を新たに生成します。

R_RSIP_FWUP_MAC_Sign_Init/Update/Finish()は R_RSIP_StartUpdateFirmware()を呼び出して、RSIP を FrimwareUpdate 状態にした後で利用可能になります。

また、R_RSIP_FWUP_MAC_Sign_Init/Update/Finish()を使用するために、ファームウェアアップデートで使用する鍵として、Key Encryption Key(AES128bit の鍵)を注入しておく必要があります。Key Encryption Key の注入方法は、ユーザ鍵の注入と同じ手順です。3.6.1鍵の注入と更新を参照して、Key Encryption Key を注入してください。

3.13.3 ユーザプログラムの暗号化

図 3-10にユーザプログラムの暗号化方法を示します。

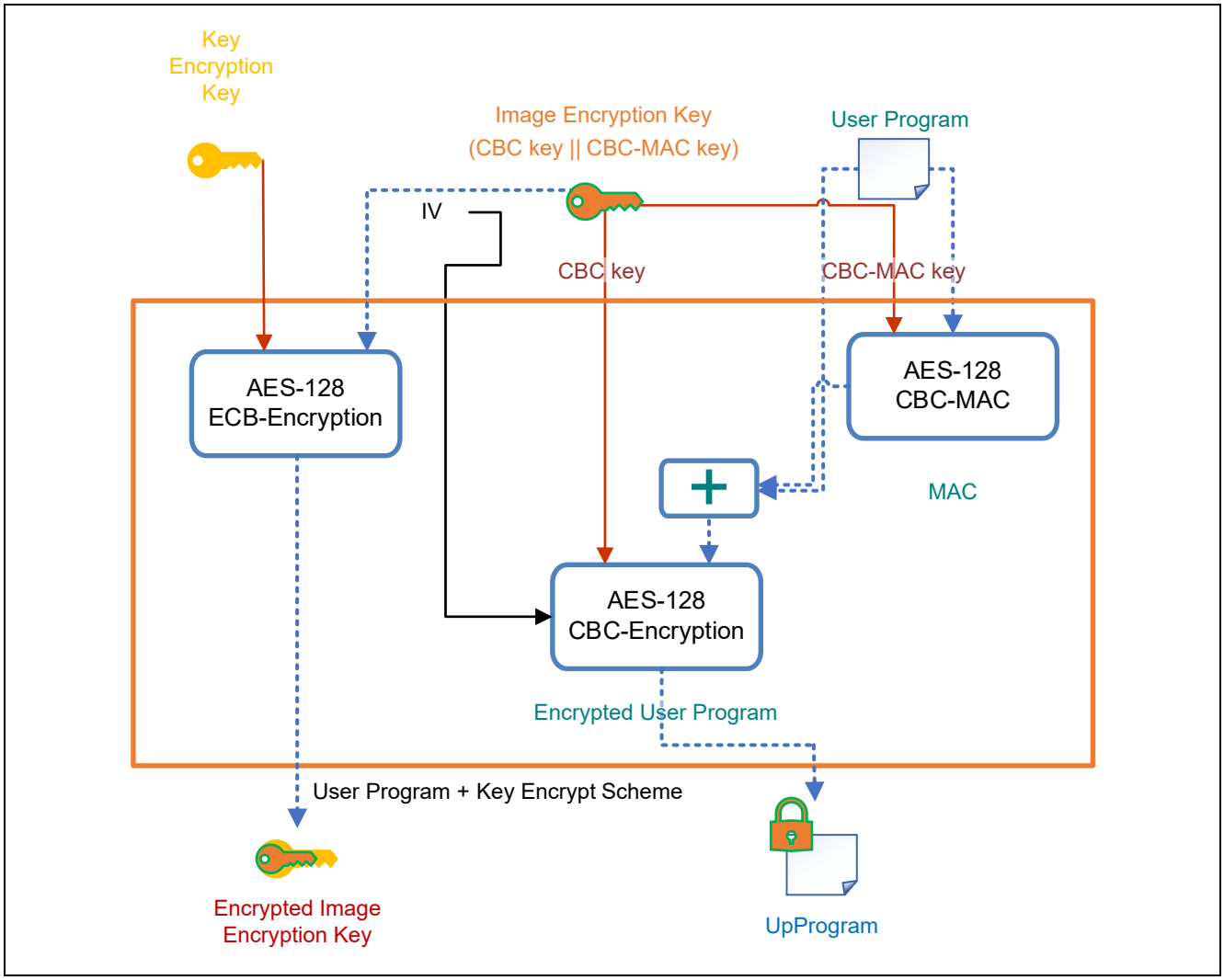

図 3-10 ファームウェアと Image Encryption Key の暗号化方法

ユーザプログラムを暗号化するための鍵(Image Encryption Key)、およびその鍵をラップするための鍵(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 をラップし、Encrypted Image Encryption Key を生成します。

4. API 関数

4.1 API 一覧

RSIP PM 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鍵管理 API

③乱数生成 API

④AES API

⑤ECC API

⑥HASH API

⑦KDF API

⑧Key Wrap API

⑨ファームウェアアップデート/セキュアブート API

実装している API を以下にまとめます。

表 4-1　共通機能 API

API	説明
R_RSIP_Open	RSIP をモジュールストップ状態から解除し、RSIP PM ドライバをオープンします。
R_RSIP_Close	RSIP をモジュールストップ状態にし、RSIP PM ドライバをクローズします。
R_RSIP_GetVersion	RSIP PM ドライバのバージョンを出力します。

表 4-2　鍵管理 API

API	説明
R_RSIP_InitialKeyWrap	鍵注入時に使用します。 UFPK でラップされたユーザ鍵から Wrapped Key のバイナリ値を生成します。
R_RSIP_EncryptedKeyWrap	鍵更新時に使用します。 KUK でラップされたユーザ鍵から Wrapped Key を生成します。
R_RSIP_KeyGenerate	対称鍵暗号の鍵を生成します。
R_RSIP_KeyPairGenerate	非対称鍵暗号の鍵を生成します。
R_RSIP_PublicKeyExport	公開鍵の Wrapped Key から平文の公開鍵を生成します。

表 4-3　乱数生成 API

API	説明
R_RSIP_RandomNumberGenerate	乱数を生成します。

表 4-4 AES API

API	説明
R_RSIP_AES_Cipher_Init	AES 暗号演算を実行する準備を行います。
R_RSIP_AES_Cipher_Update	AES 暗号演算を実行します。
R_RSIP_AES_Cipher_Finish	AES 暗号演算を終了します。
R_RSIP_AES_AEAD_Init	AES AEAD 演算を実行する準備を行います。
R_RSIP_AES_AEAD_LengthsSet	AES AEAD 演算で使用するデータのサイズを指定します。
R_RSIP_AES_AEAD_AADUpdate	AES AEAD 演算で使用する追加認証データを指定します。
R_RSIP_AES_AEAD_Update	AES AEAD 演算を実行します、
R_RSIP_AES_AEAD_Finish	AES AEAD の暗号化演算を終了します。
R_RSIP_AES_AEAD_Verify	AES AEAD の復号演算を終了します。
R_RSIP_AES_MAC_Init	AES MAC 演算を実行する準備を行います。
R_RSIP_AES_MAC_Update	AES MAC 演算を実行します。
R_RSIP_AES_MAC_SignFinish	AES MAC の生成演算を終了します。
R_RSIP_AES_MAC_VerifyFinish	AES MAC の検証演算を終了します。

表 4-5 ECC API

API	説明
R_RSIP_ECDSA_Sign	ECDSA 署名生成演算を実行します。
R_RSIP_ECDSA_Verify	ECDSA 署名検証演算を実行します。
R_RSIP_PKI_CertVerify	公開鍵の証明書に対して ECDSA 署名検証を行います。
R_RSIP_PKI_CertKeyImport	検証した証明書内の公開鍵をラップします。
R_RSIP_PKI_VerifiedCertInfoExport	検証した証明書の情報を出力します。
R_RSIP_PKI_VerifiedCertInfoImport	検証した証明書の情報を取り込みます。
R_RSIP_ECDH_KeyAgree	共有秘密を生成します。
R_RSIP_ECDH_PlainKeyAgree	共有秘密を生成します。

表 4-6 HASH API

API	説明
R_RSIP_SHA_Compute	SHA 演算を行います。
R_RSIP_SHA_Init	SHA 演算を実行する準備を行います。
R_RSIP_SHA_Update	SHA 演算を実行します。
R_RSIP_SHA_Finish	SHA 演算を終了します。
R_RSIP_SHA_Suspend	SHA 演算を中断します。
R_RSIP_SHA_Resume	SHA 演算を再開します。
R_RSIP_HMAC_Compute	HMAC 演算を行います。
R_RSIP_HMAC_Verify	HMAC 検証を行います。
R_RSIP_HMAC_Init	HMAC 演算を実行する準備を行います。
R_RSIP_HMAC_Update	HMAC 演算を実行します。
R_RSIP_HMAC_SignFinish	HMAC 生成演算を終了します。
R_RSIP_HMAC_VerifyFinish	HMAC 検証演算を終了します。
R_RSIP_HMAC_Suspend	HMAC 演算を中断します。
R_RSIP_HMAC_Resume	HMAC 演算を再開します。

表 4-7 KDF API

API	説明
R_RSIP_KDF_SHA_Init	KDF のためのハッシュ演算を実行する準備を行います。
R_RSIP_KDF_SHA_ECDHSecretUpdate	KDF のためのハッシュ演算に使用する ECDH 共有秘密を入力します。
R_RSIP_KDF_SHA_Update	KDF のためのハッシュ演算に使用するメッセージを入力します。
R_RSIP_KDF_SHA_Finish	ラップした DKM を生成します。
R_RSIP_KDF_SHA_Suspend	KDF のためのハッシュ演算を中断します。
R_RSIP_KDF_SHA_Resume	KDF のためのハッシュ演算を再開します。
R_RSIP_KDF_DKMConcatenate	2 つのラップした DKM を連接します。
R_RSIP_KDF_DerivedKeyImport	DKM から Wrapped Key を出力します。
R_RSIP_KDF_DerivedIVWrap	DKM からラップした初期化ベクタを出力します。

表 4-8 Key Wrap API

API	説明
R_RSIP_RFC3394_KeyWrap	RFC3394 に準拠したアルゴリズムで鍵をラップします。
R_RSIP_RFC3394_KeyUnwrap	RFC3394 に準拠したアルゴリズムで鍵をアンラップします。

表 4-9 ファームウェアアップデート/セキュアブート API

API	説明
R_RSIP_FWUP_StartUpdateFirmware	ファームウェアアップデート状態へ遷移します。
R_RSIP_FWUP_MAC_Sign_Init	暗号化されたファームウェアの復号と MAC 生成の準備を行います。
R_RSIP_FWUP_MAC_Sign_Update	暗号化されたファームウェアの復号し、平文ファームウェアを出力します。
R_RSIP_FWUP_MAC_Sign_Finish	暗号化されたファームウェアの復号と MAC の検証を行い、平文ファームウェアの MAC を生成します。
R_RSIP_SB_MAC_Verify_Init	平文ファームウェアの MAC を検証する準備を行います。
R_RSIP_SB_MAC_Verify_Update	平文ファームウェアの MAC 演算を実行します。
R_RSIP_SB_MAC_Verify_Finish	平文ファームウェアの MAC 検証を実行します。

4.2 API 詳細

4.2.1 共通機能

4.2.1.1 R_RSIP_Open

Format

```
#include “r_rsip_protected_rx_if.h”
fsp_err_t R_RSIP_Open(
        rsip_ctrl_t * const p_ctrl,
        rsip_cfg_t const * const p_cfg
)
```

Parameters

p_ctrl	入力/出力	RSIP PM ドライバ管理構造体
p_cfg	入力	コンフィグレーション構造体

Return Values

FSP_SUCCESS	正常終了
FSP_ERR_ASSERTION	ポインタ引数が NULL
FSP_ERR_ALREADY_OPEN	ドライバはすでに Open している
FSP_ERR_CRYPTO_RSIP_FAIL	入力パラメータが不正
FSP_ERR_CRYPTO_RSIP_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
FSP_ERR_CRYPTO_RSIP_FATAL	ソフトウェアの破損またはハードウェアの障害を検出

Description

RSIP をモジュールストップ状態から解除し、RSIP の初期化ならびに RSIP PM ドライバをオープンします。

p_ctrl には、RSIP PM ドライバ共通で使用する管理構造体を入力します。p_ctrl は R_RSIP_Close()をするまで保持してください。また p_ctrl は、rsip_instance_ctrl_t 型の変数として宣言した RSIP ドライバ管理構造体を 0 クリアしたものを入力として呼び出してください。

p_cfg には、デバイスの端子設定構造体を入力します。RX RSIP PM ドライバでは使用しませんが、NULL を指定することはできません。任意の rsip_cfg_t 構造体のポインタを入力してください。

Reentrant

非対応

4.2.1.2 R_RSIP_Close

Format

```
#include “r_rsip_protected_rx_if.h”
fsp_err_t R_RSIP_Close (
        rsip_ctrl_t * const p_ctrl
)
```

Parameters

p_ctrl	入力/出力	RSIP PM ドライバ管理構造体

Return Values

FSP_SUCCESS	正常終了
FSP_ERR_ASSERTION	ポインタ引数が NULL
FSP_ERR_NOT_OPEN	RSIP PM ドライバが Open されていない
FSP_ERR_CRYPTO_RSIP_FATAL	ソフトウェアの破損を検出

Description

RSIP をモジュールストップ状態にし、RSIP PM ドライバをクローズします。

p_ctrl には、R_RSIP_Open()関数で使用した RSIP PM ドライバ管理構造体を指定します。

Reentrant

非対応

4.2.1.3 R_RSIP_GetVersion

Format

```
#include “r_rsip_protected_rx_if.h”
uint32_t R_RSIP_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

RSIP PM 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

4.2.2 鍵管理

4.2.2.1 R_RSIP_InitialKeyWrap

Format

```
#include “r_rsip_protected_rx_if.h”
  fsp_err_t R_RSIP_InitialKeyWrap(
        rsip_ctrl_t * const p_ctrl,
        void const * const p_wrapped_user_factory_programming_key,
        void const * const p_initial_vector,
        void const * const p_encrypted_key,
        rsip_wrapped_key_t * const p_wrapped_key
)
```

Parameters

p_ctrl	入力/出力	RSIP PM ドライバ管理構造体
p_wrapped_user_factory_programming_key	入力	W-UFPK
p_initial_vector	入力	初期化ベクタ (16 byte)
p_encrypted_key	入力	Encrypted Key
p_wrapped_key	入力/出力	Wrapped Key (p_value に表 2-10で定義されている各鍵の種類に対応したサイズが出力されます。)

Return Values

FSP_SUCCESS	正常終了
FSP_ERR_ASSERTION	ポインタ引数が NULL
FSP_ERR_CRYPTO_RSIP_KEY_SET_FAIL	p_encrypted_key に指定された鍵の検証でエラーが発生
FSP_ERR_CRYPTO_RSIP_FAIL	入力パラメータが不正
FSP_ERR_CRYPTO_RSIP_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
FSP_ERR_CRYPTO_RSIP_FATAL	ソフトウェアの破損またはハードウェアの障害を検出

Description

UFPK でラップされたユーザ鍵から、ユーザ鍵の Wrapped Key のバイナリデータを生成します。
p_wrapped_user_factory_programming_key にユーザ鍵のラップで使用した UFPK の W-UFPK、p_initial_vector にユーザ鍵のラップで使用した初期化ベクタ、p_encrypted_key に UFPK でラップされたユーザ鍵を指定します。p_wrapped_key の p_value フィールドに指定されたアドレスに Wrapped Key のバイナリデータを出力します。
p_wrapped_key の type フィールドには表 2-4に定義される値を入力してください。p_wrapped_key に出力される Wrapped Key のバイトサイズは、表 2-10で定義されている各鍵の種類に対応したサイズになります。type フィールドで指定したアルゴリズムに合わせて、必要なサイズの領域を確保し、アドレスを p_value フィールドに指定してください。

p_ctrl には、R_RSIP_Open()関数で使用した管理構造体のポインタを指定します。

Reentrant

非対応

4.2.2.2 R_RSIP_EncryptedKeyWrap

Format

```
#include “r_rsip_protected_rx_if.h”
 fsp_err_t R_RSIP_EncryptedKeyWrap(
        rsip_ctrl_t * const p_ctrl,
        rsip_wrapped_key_t const * const p_key_update_key,
        void const * const p_initial_vector,
        void const * const p_encrypted_key,
        rsip_wrapped_key_t * const p_wrapped_key
)
```

Parameters

p_ctrl	入力/出力	RSIP PM ドライバ管理構造体
p_key_update_key	入力	Key Update Key
p_initial_vector	入力	初期化ベクタ (16 byte)
p_encrypted_key	入力	Encrypted Key
p_wrapped_key	入力/出力	Wrapped Key (p_value に表 2-10で定義されている各鍵の種類に対応したサイズが出力されます。)

Return Values

FSP_SUCCESS	正常終了
FSP_ERR_ASSERTION	ポインタ引数が NULL
FSP_ERR_NOT_OPEN	RSIP PM ドライバが Open されていない
FSP_ERR_INVALID_STATE	RSIP PM ドライバの内部状態が不正
FSP_ERR_NOT_ENABLED	key_type にサポートしていない鍵の種類が指定された
FSP_ERR_CRYPTO_RSIP_FAIL	入力パラメータが不正
FSP_ERR_CRYPTO_RSIP_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
FSP_ERR_CRYPTO_RSIP_FATAL	ソフトウェアの破損またはハードウェアの障害を検出

Description

KUK でラップされたユーザ鍵から、Wrapped Key を生成します。
p_encrypted_key に入力された KUK でラップされたユーザ鍵を、p_key_update_key に入力された KUK の Wrapped Key と KUK でユーザ鍵をラップする時に使用した初期化ベクタ p_initial_vector を使用してアンラップします。その後 p_wrapped_key の p_value フィールドに指定されたアドレスに Wrapped Key のバイナリデータを出力します。
p_wrapped_key の type フィールドには表 2-4に定義される値を入力してください。p_wrapped_key に出力される Wrapped Key のバイトサイズは、表 2-10で定義されている各鍵の種類に対応したサイズになります。type フィールドで指定したアルゴリズムに合わせて、必要なサイズ以上の領域を確保し、アドレスを p_value フィールドに指定してください。

p_ctrl には、R_RSIP_Open()関数で使用した管理構造体のポインタを指定します。

Reentrant

非対応

4.2.2.3 R_RSIP_KeyGenerate

Format

```
#include “r_rsip_protected_rx_if.h”
fsp_err_t R_RSIP_KeyGenerate(
        rsip_ctrl_t * const p_ctrl,
        rsip_wrapped_key_t * const p_wrapped_key
)
```

Parameters

p_ctrl	入力/出力	RSIP PM ドライバ管理構造体
p_wrapped_key	入力/出力	Wrapped Key (p_value に表 2-10で定義されている各鍵の種類に対応したサイズが出力されます。)

Return Values

FSP_SUCCESS	正常終了
FSP_ERR_ASSERTION	ポインタ引数が NULL
FSP_ERR_NOT_OPEN	RSIP PM ドライバが Open されていない
FSP_ERR_INVALID_STATE	RSIP PM ドライバの内部状態が不正
FSP_ERR_NOT_ENABLED	key_type にサポートしていない鍵の種類が指定された
FSP_ERR_CRYPTO_RSIP_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
FSP_ERR_CRYPTO_RSIP_FATAL	ソフトウェアの破損またはハードウェアの障害を検出

Description

p_wrapped_key の type フィールドで指定した対称鍵の Wrapped Key を p_wrapped_key に出力します。p_wrapped_key の type フィールドには表 2-4に定義される値を入力してください。p_wrapped_key に出力される Wrapped Key のバイトサイズは、表 2-10で定義されている各鍵の種類に対応したサイズになります。type フィールドで指定したアルゴリズムに合わせて、必要なサイズ以上の領域を確保し、アドレスを p_value フィールドに指定してください。

p_ctrl には、R_RSIP_Open()関数で使用した管理構造体のポインタを指定します。

Reentrant

非対応

4.2.2.4 R_RSIP_KeyPairGenerate

Format

```
#include “r_rsip_protected_rx_if.h”
fsp_err_t R_RSIP_KeyPairGenerate(
        rsip_ctrl_t * const p_ctrl,
        rsip_wrapped_key_t *const p_wrapped_public_key,
        rsip_wrapped_key_t * const p_wrapped_private_key
)
```

Parameters

p_ctrl	入力/出力	RSIP PM ドライバ管理構造体
p_wrapped_public_key	入力/出力	公開鍵の Wrapped Key (p_value に表 2-10で定義されている各鍵の種類に対応したサイズが出力されます。)
p_wrapped_private_key	入力/出力	秘密鍵の Wrapped Key (p_value に表 2-10で定義されている各鍵の種類に対応したサイズが出力されます。)

Return Values

FSP_SUCCESS	正常終了
FSP_ERR_ASSERTION	ポインタ引数が NULL
FSP_ERR_NOT_OPEN	RSIP PM ドライバが Open されていない
FSP_ERR_INVALID_STATE	RSIP PM ドライバの内部状態が不正
FSP_ERR_NOT_ENABLED	key_type にサポートしていない鍵の種類が指定された
FSP_ERR_CRYPTO_RSIP_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
FSP_ERR_CRYPTO_RSIP_FATAL	ソフトウェアの破損またはハードウェアの障害を検出

Description

p_wrapped_public_key と p_wrapped_private_key の type フィールドで指定した非対称鍵の Wrapped Key を p_wrapped_public_key と p_wrapped_public_key に出力します。
p_wrapped_key の type フィールドには表 2-4に定義される値を入力してください。p_wrapped_key に出力される Wrapped Key のバイトサイズは、表 2-10で定義されている各鍵の種類に対応したサイズになります。type フィールドで指定したアルゴリズムに合わせて、必要なサイズ以上の領域を確保し、アドレスを p_value フィールドに指定してください。

p_ctrl には、R_RSIP_Open()関数で使用した管理構造体のポインタを指定します。

Reentrant

非対応

4.2.2.5 R_RSIP_PublicKeyExport

Format

```
#include “r_rsip_protected_rx_if.h”
fsp_err_t R_RSIP_PublicKeyExport(
        rsip_wrapped_key_t const * const p_wrapped_public_key,
        uint8_t * const p_raw_public_key
)
```

Parameters

p_wrapped_public_key	入力	公開鍵の Wrapped Key
p_raw_public_key	出力	平文の公開鍵 (64 byte)

Return Values

FSP_SUCCESS	正常終了
FSP_ERR_ASSERTION	ポインタ引数が NULL
FSP_ERR_CRYPTO_RSIP_KEY_SET_FAIL	p_wrapped_public_key に サポートされていない鍵の種類入力された

Description

公開鍵の Wrapped Key から平文の公開鍵を抽出します。

p_wrapped_public_key で指定された Wrapped Key から平文の公開鍵を p_raw_public_key に出力します。

p_raw_public_key に出力される公開鍵は以下のフォーマットで出力されます。

secp256r1, brainpoolP256r1,secp256k1 :
  p_raw_public_key[0:31] = 公開鍵 Qx
  p_raw_public_key[32:63] = 公開鍵 Qy

Reentrant

非対応

4.2.3 乱数生成

4.2.3.1 R_RSIP_RandomNumberGenerate

Format

```
#include “r_rsip_protected_rx_if.h”
fsp_err_t R_RSIP_RandomNumberGenerate(
        rsip_ctrl_t * const p_ctrl,
        uint8_t * const p_random
)
```

Parameters

p_ctrl	入力/出力	RSIP PM ドライバ管理構造体
p_random	出力	16 byte の乱数値

Return Values

FSP_SUCCESS	正常終了
FSP_ERR_ASSERTION	ポインタ引数が NULL
FSP_ERR_NOT_OPEN	RSIP PM ドライバが Open されていない
FSP_ERR_INVALID_STATE	RSIP PM ドライバの内部状態が不正
FSP_ERR_CRYPTO_RSIP_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
FSP_ERR_CRYPTO_RSIP_FATAL	ソフトウェアの破損またはハードウェアの障害を検出

Description

NIST SP800-90A に準拠した 16 バイトの乱数値を p_random に出力します。

p_ctrl には、R_RSIP_Open()関数で使用した管理構造体のポインタを指定します。

Reentrant

非対応

4.2.4 AES

4.2.4.1 R_RSIP_AES_Cipher_Init

Format

```
#include “r_rsip_protected_rx_if.h”
 fsp_err_t R_RSIP_AES_Cipher_Init(
        rsip_ctrl_t * const p_ctrl,
        rsip_aes_cipher_mode_t const mode,
        rsip_wrapped_key_t const * const p_wrapped_key,
        uint8_t const * const p_initial_vector
)
```

Parameters

p_ctrl	入力/出力	RSIP PM ドライバ管理構造体
mode	入力	実行するブロック暗号モード
p_wrapped_key	入力	Wrapped Key
p_initial_vector	入力	初期化ベクタ（16 byte）

Return Values

FSP_SUCCESS	正常終了
FSP_ERR_ASSERTION	ポインタ引数が NULL
FSP_ERR_NOT_OPEN	RSIP PM ドライバが Open されていない
FSP_ERR_INVALID_STATE	RSIP PM ドライバの内部状態が不正
FSP_ERR_NOT_ENABLED	p_wrapped_key にサポートしていない鍵の種類が指定された
FSP_ERR_INVALID_ARGUMENT	入力キーのタイプまたはモードが不正
FSP_ERR_CRYPTO_RSIP_KEY_SET_FAIL	p_wrapped_key で指定された鍵の検証でエラーが発生
FSP_ERR_CRYPTO_RSIP_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
FSP_ERR_CRYPTO_RSIP_FATAL	ソフトウェアの破損を検出

Description

AES 暗号演算を実行する準備を行います。

mode に実行する AES ブロック暗号モード、p_wrapped_key に暗号復号で使用する鍵の Wrapped Key を指定します。p_initial_vector は、mode によって指定するデータの意味が異なります。CBC の場合は初期化ベクタ、CTR の場合はノンスとなります。ECB の場合は使用しないため、NULL を指定してください。

p_ctrl には、R_RSIP_Open()関数で使用した管理構造体のポインタを指定します。p_ctrl に、R_RSIP_AES_Cipher_Init()関数の実行結果を格納します。p_ctrl に出力された値を、R_RSIP_AES_Cipher_Update()関数および R_RSIP_AES_Cipher_Finish()関数で使用します。

Reentrant

非対応

4.2.4.2 R_RSIP_AES_Cipher_Update

Format

```
#include “r_rsip_protected_rx_if.h”
 fsp_err_t R_RSIP_AES_Cipher_Update(
        rsip_ctrl_t * const p_ctrl,
        uint8_t const * const p_input,
        uint8_t * const p_output,
        uint32_t const length
)
```

Parameters

p_ctrl	入力/出力	RSIP PM ドライバ管理構造体
p_input	入力	入力データ領域
p_output	出力	出力データ領域 (length と同じサイズが出力されます)
length	入力	入力データのバイト長(0~任意 byte、16 の倍数である必要があります)

Return Values

FSP_SUCCESS	正常終了
FSP_ERR_ASSERTION	ポインタ引数が NULL
FSP_ERR_NOT_OPEN	RSIP PM ドライバが Open されていない
FSP_ERR_INVALID_STATE	RSIP PM ドライバの内部状態が不正
FSP_ERR_INVALID_SIZE	入力サイズが不正

Description

AES 暗号演算を行います。
R_RSIP_AES_Cipher_Init()の mode で暗号処理指定時は、p_input に入力された平文を length で指定されたサイズ分暗号化し、p_output に出力します。
R_RSIP_AES_Cipher_Init()の mode で復号処理指定時は、p_input に入力された暗号文を length で指定されたサイズ分復号し、p_output に出力します。

p_ctrl には、R_RSIP_AES_Cipher_Init()関数で使用した p_ctrl を指定します。

Reentrant

非対応

4.2.4.3 R_RSIP_AES_Cipher_Finish

Format

```
#include “r_rsip_protected_rx_if.h”
 fsp_err_t R_RSIP_AES_Cipher_Finish(
        rsip_ctrl_t *const p_ctrl
)
```

Parameters

p_ctrl	入力/出力	RSIP PM ドライバ管理構造体

Return Values

FSP_SUCCESS	正常終了
FSP_ERR_ASSERTION	ポインタ引数が NULL
FSP_ERR_NOT_OPEN	RSIP PM ドライバが Open されていない
FSP_ERR_INVALID_STATE	RSIP PM ドライバの内部状態が不正
FSP_ERR_CRYPTO_RSIP_FATAL	ソフトウェアの破損を検出

Description

AES 暗号演算を終了します。

p_ctrl には、R_RSIP_AES_Cipher_Init()関数および R_RSIP_AES_Cipher_Update()関数で使用した p_ctrl を指定します。

Reentrant

非対応

4.2.4.4 R_RSIP_AES_AEAD_Init

Format

```
#include “r_rsip_protected_rx_if.h”
 fsp_err_t R_RSIP_AES_AEAD_Init(
        rsip_ctrl_t * const p_ctrl,
        rsip_aes_aead_mode_t const mode,
        rsip_wrapped_key_t const * const p_wrapped_key,
        uint8_t const * const p_nonce,
        uint32_t const nonce_length
)
```

Parameters

p_ctrl	入力/出力	RSIP PM ドライバ管理構造体
mode	入力	実行する AEAD モード
p_wrapped_key	入力	Wrapped Key
p_nonce	入力	ノンス
nonce_length	入力	ノンスのバイト長(GCM の場合は 1byte 以上、CCM の場合は 7～13byte の値を指定してください)

Return Values

FSP_SUCCESS	正常終了
FSP_ERR_ASSERTION	ポインタ引数が NULL
FSP_ERR_NOT_OPEN	RSIP PM ドライバが Open されていない
FSP_ERR_INVALID_STATE	RSIP PM ドライバの内部状態が不正
FSP_ERR_NOT_ENABLED	p_wrapped_keyにサポートしていない鍵の種類が指定された
FSP_ERR_INVALID_ARGUMENT	入力キーのタイプまたはモードが不正
FSP_ERR_CRYPTO_RSIP_KEY_SET_FAIL	p_wrapped_key で指定された鍵の検証でエラーが発生
FSP_ERR_CRYPTO_RSIP_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
FSP_ERR_CRYPTO_RSIP_FATAL	ソフトウェアの破損を検出

Description

R_RSIP_AES_AEAD_Init()関数は AES AEAD 演算を実行する準備を行います。

mode に実行する AEAD モード、p_wrapped_key に AEAD で使用する鍵の Wrapped Key を指定します。p_nonce は、mode によって指定するデータの意味が異なります。GCM モードを指定した場合初期化ベクタ、CCM モードを指定した場合ノンスを入力します。nonce_length には p_nonce で指定した初期化ベクタもしくはノンスのバイト長を入力します。

p_ctrl には、R_RSIP_OPEN()関数で使用した管理構造体のポインタを指定します。p_ctrl に、R_RSIP_AES_AEAD_Init()関数の結果を格納します。p_ctrl に出力された値を、R_RSIP_AES_AEAD_LengthsSet()関数、R_RSIP_AES_AEAD_AADUpdate()関数、R_RSIP_AES_AEAD_Update()関数、R_RSIP_AES_AEAD_Finish()関数および R_RSIP_AES_AEAD_Verify()関数で使用します。

Reentrant

非対応

4.2.4.5 R_RSIP_AES_AEAD_LengthsSet

Format

```
#include “r_rsip_protected_rx_if.h”
 fsp_err_t R_RSIP_AES_AEAD_LengthsSet(
        rsip_ctrl_t * const p_ctrl,
        uint32_t const total_aad_length,
        uint32_t const total_text_length,
        uint32_t const tag_length
)
```

Parameters

p_ctrl	入力/出力	RSIP PM ドライバ管理構造体
total_aad_length	入力	追加認証データのバイト長(110 byte 以下)
total_text_length	入力	入出力データのバイト長(0~任意 byte)
tag_length	入力	認証タグのバイト長(4,6,8,10,12,14,16 byte)

Return Values

FSP_SUCCESS	正常終了
FSP_ERR_NOT_OPEN	RSIP PM ドライバが Open されていない
FSP_ERR_INVALID_SIZE	入力サイズが不正
FSP_ERR_INVALID_STATE	RSIP PM ドライバの内部状態が不正

Description

AES CCM 演算を行う場合に使用する API です。追加認証データ、入出力データ、および認証タグのバイト長を指定します。

p_ctrl には、R_RSIP_AES_AEAD_Init()関数で使用した p_ctrl を指定します。

Reentrant

非対応

4.2.4.6 R_RSIP_AES_AEAD_AADUpdate

Format

```
#include “r_rsip_protected_rx_if.h”
 fsp_err_t R_RSIP_AES_AEAD_AAdUpdate(
        rsip_ctrl_t * const p_ctrl,
        uint8_t const * const p_aad,
        uint32_t const aad_length
)
```

Parameters

p_ctrl	入力/出力	RSIP PM ドライバ管理構造体
p_aad	入力	追加認証データ領域
aad_length	入力	追加認証データのバイト長(0～任意 byte)

Return Values

FSP_SUCCESS	正常終了
FSP_ERR_ASSERTION	ポインタ引数が NULL
FSP_ERR_NOT_OPEN	RSIP PM ドライバが Open されていない
FSP_ERR_INVALID_STATE	RSIP PM ドライバの内部状態が不正
FSP_ERR_CRYPTO_RSIP_KEY_SET_FAIL	R_RSIP_AES_AEAD_Init で入力された p_wrapped_key の検証でエラーが発生
FSP_ERR_CRYPTO_RSIP_FAIL	内部異常
FSP_ERR_CRYPTO_RSIP_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
FSP_ERR_CRYPTO_RSIP_FATAL	ソフトウェアの破損を検出

Description

AES AEAD 演算で使用する追加認証データを指定します。
p_aad には追加認証データ、aad_length には追加認証データのバイトサイズを入力します。

p_ctrl には、R_RSIP_AES_AEAD_Init()関数および R_RSIP_AES_AEAD_LengthsSet()関数で使用した p_ctrl を指定します。

R_RSIP_AES_AEAD_Update()関数を呼び出す前に、本関数で追加認証データを入力してください。

Reentrant

非対応

4.2.4.7 R_RSIP_AES_AEAD_Update

Format

```
#include “r_rsip_protected_rx_if.h”
 fsp_err_t R_RSIP_AES_AEAD_Update(
        rsip_ctrl_t * const p_ctrl,
        uint8_t const * const p_input,
        uint32_t const input_length,
        uint8_t * const p_output,
        uint32_t * const p_output_length
)
```

Parameters

p_ctrl	入力/出力	RSIP PM ドライバ管理構造体
p_input	入力	入力データ領域
input_length	入力	入力データのバイト長(0~任意 byte)
p_output	出力	出力データ領域 (バッファリングされている分を含めて p_input への入力に対して演算が完了したサイズが出力されます)
p_output_length	出力	出力データのバイト長

Return Values

FSP_SUCCESS	正常終了
FSP_ERR_ASSERTION	ポインタ引数が NULL
FSP_ERR_NOT_OPEN	RSIP PM ドライバが Open されていない
FSP_ERR_INVALID_STATE	RSIP PM ドライバの内部状態が不正
FSP_ERR_INVALID_SIZE	Input_length で指定されたバイトサイズが不正

Description

AES AEAD 演算を実行します。
R_RSIP_AES_AEAD_Init()の mode で暗号処理指定時は、p_input に入力された平文を input_length で指定されたサイズ分暗号化し、p_output に暗号化データに出力します。input_length で指定したサイズが 16 バイトアラインを取れていないサイズの場合、16 バイトの端数分は次に呼ばれる R_RSIP_AES_AEAD_Update()、R_RSIP_AES_AEAD_Finish()、R_RSIP_AES_AEAD_Verify()で暗号化します。暗号化して p_output に出力したサイズは p_output_length に出力されます。
R_RSIP_AES_AEAD_Init()の mode で復号処理指定時は、p_input に入力された暗号文を length で指定されたサイズ分復号し、p_output に復号データを出力します。input_length で指定したサイズが 16 バイトアラインを取れていないサイズの場合、16 バイトの端数分は次に呼ばれる R_RSIP_AES_AEAD_Update()、R_RSIP_AES_AEAD_Finish()、R_RSIP_AES_AEAD_Verify()で復号します。復号して p_output に出力したサイズは p_output_length に出力されます。
p_input と p_output 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。

p_ctrl には、R_RSIP_AES_AEAD_Init()関数、R_RSIP_AES_AEAD_LengthsSet()関数、および R_RSIP_AES_AEAD_AADUpdate()関数で使用した p_ctrl を指定します。

Reentrant

非対応

4.2.4.8 R_RSIP_AES_AEAD_Finish

Format

```
#include “r_rsip_protected_rx_if.h”
 fsp_err_t R_RSIP_AES_AEAD_Finish(
        rsip_ctrl_t * const p_ctrl,
        uint8_t * const p_output,
        uint32_t * const p_output_length,
        uint8_t * const p_tag
)
```

Parameters

p_ctrl	入力/出力	RSIP PM ドライバ管理構造体
p_output	出力	出力データ領域 (バッファリングされている分を含めて演算が完了したサイズが出力されます)
p_output_length	出力	出力データのバイト長
p_tag	出力	認証タグ領域(16 バイト)

Return Values

FSP_SUCCESS	正常終了
FSP_ERR_ASSERTION	ポインタ引数が NULL
FSP_ERR_NOT_OPEN	RSIP PM ドライバが Open されていない
FSP_ERR_INVALID_STATE	RSIP PM ドライバの内部状態が不正
FSP_ERR_CRYPTO_RSIP_FAIL	内部エラーが発生
FSP_ERR_CRYPTO_RSIP_FATAL	ソフトウェアの破損を検出

Description

AES AEAD 演算の暗号化ならびに認証タグの生成を行います。
R_RSIP_AES_AEAD_Update()で入力された全データサイズが 16 バイトアラインを取れていない場合、R_RSIP_AES_AEAD_Finish()関数の p_output に 16 バイト端数分の暗号データ、p_output_length には本 API から出力された暗号データのサイズを出力します。p_tag には、認証タグが出力されます。
復号ならびにタグの検証は R_RSIP_AES_AEAD_Verify()を使用してください。

p_ctrl には、R_RSIP_AES_AEAD_Init()関数、R_RSIP_AES_AEAD_LengthsSet()関数、R_RSIP_AES_AEAD_AADUpdate()関数、および R_RSIP_AES_AEAD_Update()関数で使用した p_ctrl を指定します。

Reentrant

非対応

4.2.4.9 R_RSIP_AES_AEAD_Verify

Format

```
#include “r_rsip_protected_rx_if.h”
 fsp_err_t R_RSIP_AES_AEAD_Verify(
        rsip_ctrl_t * const p_ctrl,
        uint8_t * const p_output,
        uint32_t * const p_output_length,
        uint8_t const * const p_tag,
        uint32_t const tag_length
)
```

Parameters

p_ctrl	入力/出力	RSIP PM ドライバ管理構造体
p_output	出力	出力データ領域 (バッファリングされている分を含めて演算が完了したサイズが出力されます)
p_output_length	出力	出力データのバイト長
p_tag	入力	認証タグ領域
tag_length	入力	認証タグのバイト長(1～16 byte)

Return Values

FSP_SUCCESS	正常終了
FSP_ERR_ASSERTION	ポインタ引数が NULL
FSP_ERR_NOT_OPEN	RSIP PM ドライバが Open されていない
FSP_ERR_INVALID_STATE	RSIP PM ドライバの内部状態が不正
FSP_ERR_INVALID_SIZE	tag_length が不正な値
FSP_ERR_CRYPTO_RSIP_FAIL	内部エラーが発生
FSP_ERR_CRYPTO_RSIP_AUTHENTICATION	認証エラー
FSP_ERR_CRYPTO_RSIP_FATAL	ソフトウェアの破損を検出

Description

AES AEAD 演算の復号ならびにタグの検証を行います。
R_RSIP_AES_AEAD_Update()で入力された全データサイズが 16 バイトアラインを取れていない場合、R_RSIP_AES_AEAD_Verify()関数の p_output に端数分の平文データ、p_output_length には本 API から出力された平文データのサイズを出力します。p_tag には、認証で使用する認証タグ値を、tag_length にはタグの認証で使用するタグのバイト長を入力します。
暗号ならびにタグの生成は R_RSIP_AES_AEAD_Finish()を使用してください。

p_ctrl には、R_RSIP_AES_AEAD_Init()関数、R_RSIP_AES_AEAD_LengthsSet()関数、R_RSIP_AES_AEAD_AADUpdate()関数、および R_RSIP_AES_AEAD_Update()関数で使用した p_ctrl を指定します。

Reentrant

非対応

4.2.4.10 R_RSIP_AES_MAC_Init

Format

```
#include “r_rsip_protected_rx_if.h”
 fsp_err_t R_RSIP_AES_MAC_Init(
        rsip_ctrl_t * const p_ctrl,
        rsip_aes_mac_mode_t const mode,
        rsip_wrapped_key_t const * const p_wrapped_key
)
```

Parameters

p_ctrl	入力/出力	RSIP PM ドライバ管理構造体
mode	入力	実行する MAC 演算モード
p_wrapped_key	入力	Wrapped Key

Return Values

FSP_SUCCESS	正常終了
FSP_ERR_ASSERTION	ポインタ引数が NULL
FSP_ERR_NOT_OPEN	RSIP PM ドライバが Open されていない
FSP_ERR_INVALID_STATE	RSIP PM ドライバの内部状態が不正
FSP_ERR_NOT_ENABLED	p_wrapped_keyにサポートしていない鍵の種類が指定された
FSP_ERR_CRYPTO_RSIP_KEY_SET_FAIL	p_wrapped_keyの検証でエラーが発生
FSP_ERR_CRYPTO_RSIP_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
FSP_ERR_CRYPTO_RSIP_FATAL	ソフトウェアの破損を検出

Description

AES MAC 演算を実行する準備を行います。
mode に実行する MAC モード、p_wrapped_key に MAC 演算で使用する鍵の Wrapped Key を指定します。

p_ctrl には、R_RSIP_Open()関数で使用した管理構造体のポインタを指定します。p_ctrl に AES MAC 演算を実行する準備結果を格納します。p_ctrl に出力された値を、R_RSIP_AES_MAC_Update()関数、R_RSIP_AES_MAC_Finish()関数および R_RSIP_AES_MAC_Verify()関数で使用します。

Reentrant

非対応

4.2.4.11 R_RSIP_AES_MAC_Update

Format

```
#include “r_rsip_protected_rx_if.h”
 fsp_err_t R_RSIP_AES_MAC_Update(
        rsip_ctrl_t * const p_ctrl,
        uint8_t const * const p_message,
        uint32_t const message_length
)
```

Parameters

p_ctrl	入力/出力	RSIP PM ドライバ管理構造体
p_message	入力	メッセージデータ領域
message_length	入力	メッセージデータのバイト長(0～任意 byte)

Return Values

FSP_SUCCESS	正常終了
FSP_ERR_ASSERTION	ポインタ引数が NULL
FSP_ERR_NOT_OPEN	RSIP PM ドライバが Open されていない
FSP_ERR_INVALID_STATE	RSIP PM ドライバの内部状態が不正

Description

AES MAC 演算を実行します。演算が完了した後は、R_RSIP_AES_MAC_SignFinish()または R_RSIP_AES_MAC_VerifyFinish()を呼び出します。
p_message には MAC 検証もしくは生成するメッセージを、message_length には入力するメッセージ長を指定します。

p_ctrl には、R_RSIP_AES_MAC_Init()関数で使用した p_ctrl を指定します。

Reentrant

非対応

4.2.4.12 R_RSIP_AES_MAC_SignFinish

Format

```
#include “r_rsip_protected_rx_if.h”
 fsp_err_t R_RSIP_AES_MAC_SignFinish(
        rsip_ctrl_t * const p_ctrl,
        uint8_t * const p_mac
)
```

Parameters

p_ctrl	入力/出力	RSIP PM ドライバ管理構造体
p_mac	出力	MAC データ領域(16 byte)

Return Values

FSP_SUCCESS	正常終了
FSP_ERR_ASSERTION	ポインタ引数が NULL
FSP_ERR_NOT_OPEN	RSIP PM ドライバが Open されていない
FSP_ERR_INVALID_STATE	RSIP PM ドライバの内部状態が不正
FSP_ERR_CRYPTO_RSIP_FAIL	内部異常
FSP_ERR_CRYPTO_RSIP_FATAL	ソフトウェアの破損を検出

Description

AES MAC 演算による MAC 生成を行います。
R_RSIP_AES_MAC_Update()関数で入力されたメッセージの MAC 値を p_mac に出力します。
MAC 検証を行う場合 R_RSIP_AES_MAC_VerifyFinish()を使用してください。

p_ctrl には、R_RSIP_AES_MAC_Init()関数および R_RSIP_AES_MAC_Update()関数で使用した p_ctrl を指定します。

Reentrant

非対応

4.2.4.13 R_RSIP_AES_MAC_VerifyFinish

Format

```
#include “r_rsip_protected_rx_if.h”
 fsp_err_t R_RSIP_AES_MAC_VerifyFinish(
        rsip_ctrl_t * const p_ctrl,
        uint8_t const * const p_mac,
        uint32_t const mac_length
)
```

Parameters

p_ctrl	入力/出力	RSIP PM ドライバ管理構造体
p_mac	入力	MAC データ領域
mac_length	入力	MAC データのバイト長(2～16 byte)

Return Values

FSP_SUCCESS	正常終了
FSP_ERR_ASSERTION	ポインタ引数が NULL
FSP_ERR_NOT_OPEN	RSIP PM ドライバが Open されていない
FSP_ERR_INVALID_STATE	RSIP PM ドライバの内部状態が不正
FSP_ERR_INVALID_SIZE	mac_length で指定されたサイズが不正な値
FSP_ERR_CRYPTO_RSIP_FAIL	内部異常
FSP_ERR_CRYPTO_RSIP_AUTHENTICATION	MAC 検証エラー
FSP_ERR_CRYPTO_RSIP_FATAL	ソフトウェアの破損を検出

Description

AES MAC 演算による MAC 検証を行います。
R_RSIP_AES_MAC_Update()関数で入力されたメッセージの MAC 値と p_mac で入力された MAC の検証を行います。p_mac で入力された MAC 値のうち mac_length で指定されたバイト数分有効な MAC 値として検証を行います。
MAC 生成を行う場合 R_RSIP_AES_MAC_SignFinish()を使用してください。

p_ctrl には、R_RSIP_AES_MAC_Init()関数および R_RSIP_AES_MAC_Update()関数で使用した p_ctrl を指定します。

Reentrant

非対応

4.2.5 ECC

4.2.5.1 R_RSIP_ECDSA_Sign

Format

```
#include “r_rsip_protected_rx_if.h”
  fsp_err_t R_RSIP_ECDSA_Sign(
        rsip_ctrl_t * const p_ctrl,
        rsip_wrapped_key_t const * const p_wrapped_private_key,
        uint8_t const * const p_hash,
        uint8_t *const p_signature
)
```

Parameters

p_ctrl	入力/出力	RSIP PM ドライバ管理構造体
p_wrapped_private_key	入力	Wrapped Key
p_hash	入力	ハッシュ値 鍵長と同じサイズのハッシュ値を入力
p_signature	出力	署名 鍵長の 2 倍のサイズの署名を出力 (64 byte)

Return Values

FSP_SUCCESS	正常終了
FSP_ERR_ASSERTION	ポインタ引数が NULL
FSP_ERR_NOT_OPEN	RSIP PM ドライバが Open されていない
FSP_ERR_INVALID_STATE	RSIP PM ドライバの内部状態が不正
FSP_ERR_NOT_ENABLED	p_wrapped_key にサポートしていない鍵の種類が指定された
FSP_ERR_CRYPTO_RSIP_KEY_SET_FAIL	p_wrapped_key の検証でエラーが発生
FSP_ERR_CRYPTO_RSIP_FAIL	入力パラメータが不正または署名生成でエラーが発生
FSP_ERR_CRYPTO_RSIP_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
FSP_ERR_CRYPTO_RSIP_FATAL	ソフトウェアの破損を検出

Description

ECDSA 署名生成を行います。

p_wrapped_private_key に入力された秘密鍵を使用して、p_hash に入力されたハッシュ値の ECDSA 署名を p_signature に出力します。

p_ctrl には、R_RSIP_Open()関数で使用した管理構造体のポインタを指定します。

Reentrant

非対応

4.2.5.2 R_RSIP_ECDSA_Verify

Format

```
#include “r_rsip_protected_rx_if.h”
 fsp_err_t R_RSIP_ECDSA_Verify(
        rsip_ctrl_t *const p_ctrl,
        rsip_wrapped_key_t const * const p_wrapped_public_key,
        uint8_t const * const p_hash,
        uint8_t const * const p_signature
)
```

Parameters

p_ctrl	入力/出力	RSIP PM ドライバ管理構造体
p_wrapped_public_key	入力	Wrapped Key
p_hash	入力	検証するハッシュ値 ハッシュのバイト長は鍵長と同じサイズのハッシュ値を入力
p_signature	入力	検証する署名 鍵長の 2 倍のサイズの署名を入力

Return Values

FSP_SUCCESS	正常終了
FSP_ERR_ASSERTION	ポインタ引数が NULL
FSP_ERR_NOT_OPEN	RSIP PM ドライバが Open されていない
FSP_ERR_INVALID_STATE	RSIP PM ドライバの内部状態が不正
FSP_ERR_NOT_ENABLED	p_wrapped_keyにサポートしていない鍵の種類が指定された
FSP_ERR_CRYPTO_RSIP_KEY_SET_FAIL	p_wrapped_keyの検証でエラーが発生
FSP_ERR_CRYPTO_RSIP_FAIL	入力パラメータが不正または署名検証がエラー
FSP_ERR_CRYPTO_RSIP_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
FSP_ERR_CRYPTO_RSIP_FATAL	ソフトウェアの破損を検出

Description

ECDSA 署名検証を行います。
p_wrapped_public_key に入力された公開鍵を使用して、p_hash に入力されたハッシュ値と p_signature に入力された ECDSA 署名値の検証を行います。

p_ctrl には、R_RSIP_Open()関数で使用した管理構造体のポインタを指定します。

Reentrant

非対応

4.2.5.3 R_RSIP_PKI_ECDSA_CertVerify

Format

```
#include “r_rsip_protected_rx_if.h”
 fsp_err_t R_RSIP_PKI_ECDSA_CertVerify(
        rsip_ctrl_t *const p_ctrl,
        rsip_wrapped_key_t const * const p_wrapped_public_key,
        uint8_t const * const p_hash,
        uint8_t const * const p_signature
)
```

Parameters

p_ctrl	入力/出力	RSIP PM ドライバ管理構造体
p_wrapped_public_key	入力	Wrapped Key
p_hash	入力	検証するハッシュ値 ハッシュのバイト長は鍵長と同じサイズのハッシュ値を入力
p_signature	入力	検証する証明書の署名 鍵長の 2 倍のサイズの署名を入力

Return Values

FSP_SUCCESS	正常終了
FSP_ERR_ASSERTION	ポインタ引数が NULL
FSP_ERR_NOT_OPEN	RSIP PM ドライバが Open されていない
FSP_ERR_INVALID_STATE	RSIP PM ドライバの内部状態が不正
FSP_ERR_NOT_ENABLED	p_wrapped_keyにサポートしていない鍵の種類が指定された
FSP_ERR_CRYPTO_RSIP_KEY_SET_FAIL	p_wrapped_keyの検証でエラーが発生
FSP_ERR_CRYPTO_RSIP_FAIL	入力パラメータが不正または署名検証がエラー
FSP_ERR_CRYPTO_RSIP_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
FSP_ERR_CRYPTO_RSIP_FATAL	ソフトウェアの破損を検出

Description

KDF に使用するため、証明書に対して ECDSA 署名検証を行い、証明書情報をドライバに取り込みます。p_wrapped_public_key に入力された公開鍵を使って、p_hash に入力されたハッシュ値と p_signature に入力された ECDSA 署名値の検証を行います。

p_ctrl には、R_RSIP_Open()関数で使用した管理構造体のポインタを指定します。

Reentrant

非対応

4.2.5.4 R_RSIP_PKI_CertKeyImport

Format

```
#include “r_rsip_protected_rx_if.h”
 fsp_err_t R_RSIP_PKI_CertKeyImport(
        rsip_ctrl_t *const p_ctrl,
        uint8_t const * const p_cert,
        uint32_t const cert_length,
        uint8_t const * const p_key_param1,
        uint32_t const key_param1_length,
        uint8_t const * const p_key_param2,
        uint32_t const key_param2_length,
        rsip_hash_type_t const hash_function,
        rsip_wrapped_key_t * const p_wrapped_public_key
)
```

Parameters

p_ctrl	入力/出力	RSIP PM ドライバ管理構造体
p_cert	入力	証明書
cert_length	入力	証明書のバイト長(0~任意 byte)
p_key_param1	入力	証明書内の公開鍵パラメータ Qx
key_param1_length	入力	Qx のバイト長
p_key_param2	入力	証明書内の公開鍵パラメータ Qy
key_param2_length	入力	Qy のバイト長
hash_function	入力	証明書の署名の検証に使用するハッシュ関数の種類
p_wrapped_public_key	入力/出力	Wrapped Key (p_value に表 2-10で定義されている各鍵の種類に対応したサイズが出力されます。)

Return Values

FSP_SUCCESS	正常終了
FSP_ERR_ASSERTION	ポインタ引数が NULL
FSP_ERR_NOT_OPEN	RSIP PM ドライバが Open されていない
FSP_ERR_INVALID_STATE	RSIP PM ドライバの内部状態が不正
FSP_ERR_NOT_ENABLED	p_wrapped_keyにサポートしていない鍵の種類が指定された
FSP_ERR_CRYPTO_RSIP_FAIL	入力パラメータが不正または署名検証がエラー
FSP_ERR_CRYPTO_RSIP_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
FSP_ERR_CRYPTO_RSIP_FATAL	ソフトウェアの破損を検出

Description

証明書を検証し、検証した証明書内の公開鍵をラップして出力します。
p_cert に証明書、cert_length に証明書のバイト長、p_key_param1 に証明書内に含まれている公開鍵パラメータ Qx のアドレス、key_param1_length に Qx のバイト長、p_key_param2 に証明書内に含まれている公開鍵パラメータ Qy のアドレス、key_param2_length に Qy のバイト長、hash_function に証明書の署名の検証に使用するハッシュ関数の種類を指定し、p_wrapped_public_key に公開鍵の Wrapped Key を出力します。
p_wrapped_public_key の type フィールドには表 2-4に定義される値を入力してください。
p_wrapped_public_key に出力される Wrapped Key のバイトサイズは、表 2-10で定義されている各鍵の種類に対応したサイズになります。type フィールドで指定したアルゴリズムに合わせて、必要なサイズ以上の領域を確保し、アドレスを p_value フィールドに指定してください。

p_ctrl には、R_RSIP_Open()関数で使用した管理構造体のポインタを指定します。

Reentrant

非対応

4.2.5.5 R_RSIP_PKI_VerifiedCertInfoExport

Format

```
#include “r_rsip_protected_rx_if.h”
 fsp_err_t R_RSIP_PKI_VerifiedCertInfoExport(
        rsip_ctrl_t *const p_ctrl,
        rsip_verified_cert_info_t * const p_verified_cert_info
)
```

Parameters

p_ctrl	入力/出力	RSIP PM ドライバ管理構造体
p_verified_cert_info	出力	検証された証明書の情報 (52 byte)

Return Values

FSP_SUCCESS	正常終了
FSP_ERR_ASSERTION	ポインタ引数が NULL

Description

ドライバに格納されている検証された証明書の情報を出力します。出力される情報は、R_RSIP_PKI_ECDSA_CertVerify()関数で最後に検証された情報です。複数の検証された証明書の情報を切り替えて使用する場合に、本 API を使用して実現することができます。
p_verified_cert_info に検証された証明書の情報を出力します。

p_ctrl には、R_RSIP_Open()関数で使用した管理構造体のポインタを指定します。

Reentrant

非対応

4.2.5.6 R_RSIP_PKI_VerifiedCertInfoImport

Format

```
#include “r_rsip_protected_rx_if.h”
 fsp_err_t R_RSIP_PKI_VerifiedCertInfoImport(
        rsip_ctrl_t *const p_ctrl,
        rsip_verified_cert_info_t const * const p_verified_cert_info
```

Parameters

p_ctrl	入力/出力	RSIP PM ドライバ管理構造体
p_verified_cert_info	入力	検証された証明書の情報

Return Values

FSP_SUCCESS	正常終了
FSP_ERR_ASSERTION	ポインタ引数が NULL

Description

検証された証明書の情報をドライバに取り込みます。この情報は、R_RSIP_PKI_VerifiedCertInfoExport()関数により出力されたものです。複数の検証された証明書の情報を切り替えて使用する場合に、本 API を使用して実現することができます。

p_verified_cert_info に検証された証明書の情報を指定します。

p_ctrl には、R_RSIP_Open()関数で使用した管理構造体のポインタを指定します。

Reentrant

非対応

4.2.5.7 R_RSIP_ECDH_KeyAgree

Format

```
#include “r_rsip_protected_rx_if.h”
 fsp_err_t R_RSIP_ECDH_KeyAgree(
        rsip_ctrl_t *const p_ctrl,
        rsip_wrapped_key_t const * const p_wrapped_private_key,
        rsip_wrapped_key_t const * const p_wrapped_public_key,
        rsip_wrapped_secret_t * const p_wrapped_secret
)
```

Parameters

p_ctrl	入力/出力	RSIP PM ドライバ管理構造体
p_wrapped_private_key	入力	秘密鍵の Wrapped Key
p_wrapped_public_key	入力	公開鍵の Wrapped Key
p_wrapped_secret	出力	ラップした共有秘密 (56 byte)

Return Values

FSP_SUCCESS	正常終了
FSP_ERR_ASSERTION	ポインタ引数が NULL
FSP_ERR_NOT_OPEN	RSIP PM ドライバが Open されていない
FSP_ERR_INVALID_STATE	RSIP PM ドライバの内部状態が不正
FSP_ERR_NOT_ENABLED	p_wrapped_keyにサポートしていない鍵の種類が指定された
FSP_ERR_CRYPTO_RSIP_KEY_SET_FAIL	p_wrapped_keyの検証でエラーが発生
FSP_ERR_CRYPTO_RSIP_FAIL	入力パラメータが不正または署名検証がエラー
FSP_ERR_CRYPTO_RSIP_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
FSP_ERR_CRYPTO_RSIP_FATAL	ソフトウェアの破損を検出

Description

RSIP 使用デバイス側の保有している秘密鍵の Wrapped Key と、鍵共有を行う相手の公開鍵の Wrapped Key から共有秘密を生成し、ラップして出力します。
p_wrapped_private_key に RSIP 使用デバイス側の保有している秘密鍵の Wrapped Key を、p_wrapped_public_key に鍵共有を行う相手から受け取った公開鍵の Wrapped Key を指定し、p_wrapped_secret にラップされた共有秘密を出力します。

p_ctrl には、R_RSIP_PKI_ECDSA_CertVerify()関数および R_RSIP_PKI_CertKeyImport()関数で使用した管理構造体のポインタを指定します。

Reentrant

非対応

4.2.5.8 R_RSIP_ECDH_PlainKeyAgree

Format

```
#include “r_rsip_protected_rx_if.h”
 fsp_err_t R_RSIP_ECDH_PlainKeyAgree(
        rsip_ctrl_t *const p_ctrl,
        rsip_wrapped_key_t const * const p_wrapped_private_key,
        uint8_t const * const p_plain_public_key,
        rsip_wrapped_secret_t * const p_wrapped_secret
)
```

Parameters

p_ctrl	入力/出力	RSIP PM ドライバ管理構造体
p_wrapped_private_key	入力	秘密鍵の Wrapped Key
p_plain_public_key	入力	平文の公開鍵
p_wrapped_secret	出力	ラップした共有秘密 (56 byte)

Return Values

FSP_SUCCESS	正常終了
FSP_ERR_ASSERTION	ポインタ引数が NULL
FSP_ERR_NOT_OPEN	RSIP PM ドライバが Open されていない
FSP_ERR_INVALID_STATE	RSIP PM ドライバの内部状態が不正
FSP_ERR_NOT_ENABLED	p_wrapped_keyにサポートしていない鍵の種類が指定された
FSP_ERR_CRYPTO_RSIP_KEY_SET_FAIL	p_wrapped_keyの検証でエラーが発生
FSP_ERR_CRYPTO_RSIP_FAIL	入力パラメータが不正または署名検証がエラー
FSP_ERR_CRYPTO_RSIP_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
FSP_ERR_CRYPTO_RSIP_FATAL	ソフトウェアの破損を検出

Description

RSIP 使用デバイス側の保有している秘密鍵の Wrapped Key と、鍵共有を行う相手の公開鍵から共有秘密を生成し、ラップして出力します。
p_wrapped_private_key に RSIP 使用デバイス側の保有している秘密鍵の Wrapped Key を、p_wrapped_public_key に鍵共有を行う相手から受け取った平文の公開鍵を指定し、p_wrapped_secret にラップされた共有秘密を出力します。

p_ctrl には、R_RSIP_KeyPairGenerate()関数で使用した管理構造体のポインタを指定します。

証明書を用いた検証が行われていない平文鍵を使用した鍵交換は、中間者攻撃のリスクがあります。本 API ではなく R_RSIP_ECDH_KeyAgree()関数の使用を推奨します。

Reentrant
非対応

4.2.6 HASH

4.2.6.1 R_RSIP_SHA_Compute

Format

```
#include “r_rsip_protected_rx_if.h”
  fsp_err_t R_RSIP_SHA_Compute(
        rsip_ctrl_t * const p_ctrl,
        rsip_hash_type_t const hash_type,
        uint8_t const * const p_message,
        uint32_t const message_length,
        uint8_t * const p_digest
 )
```

Parameters

p_ctrl	入力/出力	RSIP PM ドライバ管理構造体
hash_type	入力	実行するハッシュ演算モード
p_message	入力	メッセージデータ領域
message_length	入力	メッセージデータのバイト長(0～任意 byte)
p_digest	出力	ハッシュデータ領域 SHA224:28 byte SHA256:32 byte

Return Values

FSP_SUCCESS	正常終了
FSP_ERR_ASSERTION	ポインタ引数が NULL
FSP_ERR_NOT_OPEN	RSIP PM ドライバが Open されていない
FSP_ERR_INVALID_STATE	RSIP PM ドライバの内部状態が不正
FSP_ERR_NOT_ENABLED	hash_type にサポートされていないハッシュ演算モードが指定された
FSP_ERR_INVALID_ARGUMENT	入力キーのタイプまたはモードが不正

Description

ハッシュ演算を行います。
hash_type に実行するハッシュ演算モードを指定します。

p_ctrl には､R_RSIP_Open()関数で使用した管理構造体のポインタを指定します｡p_message にはハッシュ演算を行うメッセージを入力し､message_length には p_message で入力するメッセージ長を指定します。

p_digest に、入力したメッセージのハッシュ値を出力します。

Reentrant

非対応

4.2.6.2 R_RSIP_SHA_Init

Format

```
#include “r_rsip_protected_rx_if.h”
 fsp_err_t R_RSIP_SHA_Init(
        rsip_ctrl_t * const p_ctrl,
        rsip_hash_type_t const hash_type
)
```

Parameters

p_ctrl	入力/出力	RSIP PM ドライバ管理構造体
hash_type	入力	実行するハッシュ演算モード

Return Values

FSP_SUCCESS	正常終了
FSP_ERR_ASSERTION	ポインタ引数が NULL
FSP_ERR_NOT_OPEN	RSIP PM ドライバが Open されていない
FSP_ERR_INVALID_STATE	RSIP PM ドライバの内部状態が不正
FSP_ERR_NOT_ENABLED	hash_type にサポートされていないハッシュ演算モードが指定された
FSP_ERR_INVALID_ARGUMENT	入力キーのタイプまたはモードが不正

Description

ハッシュ演算を実行する準備を行います。
hash_type に実行するハッシュ演算モードを指定します。

p_ctrl には、R_RSIP_Open()関数で使用した管理構造体のポインタを指定します。p_ctrl に、ハッシュ演算を実行する準備結果を格納します。p_ctrl に出力された値を、R_RSIP_SHA_Update()関数および R_RSIP_SHA_Finish()関数で使用します。

Reentrant

非対応

4.2.6.3 R_RSIP_SHA_Update

Format

```
#include “r_rsip_protected_rx_if.h”
 fsp_err_t R_RSIP_SHA_Update(
        rsip_ctrl_t * const p_ctrl,
        uint8_t const * const p_message,
        uint32_t const message_length
)
```

Parameters

p_ctrl	入力/出力	RSIP PM ドライバ管理構造体
p_message	入力	メッセージデータ領域
message_length	入力	メッセージデータのバイト長(0～任意 byte)

Return Values

FSP_SUCCESS	正常終了
FSP_ERR_ASSERTION	ポインタ引数が NULL
FSP_ERR_NOT_OPEN	RSIP PM ドライバが Open されていない
FSP_ERR_INVALID_STATE	RSIP PM ドライバの内部状態が不正
FSP_ERR_CRYPTO_RSIP_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
FSP_ERR_CRYPTO_RSIP_FATAL	ソフトウェアの破損を検出

Description

ハッシュ演算を行います。
p_message にはハッシュ演算を行うメッセージを入力し、message_length には p_message で入力するメッセージ長を指定します。

p_ctrl には、R_RSIP_SHA_Init()関数で使用した p_ctrl を指定します。

Reentrant

非対応

4.2.6.4 R_RSIP_SHA_Finish

Format

```
#include “r_rsip_protected_rx_if.h”
 fsp_err_t R_RSIP_SHA_Finish(
        rsip_ctrl_t * const p_ctrl,
        uint8_t * const p_digest
)
```

Parameters

p_ctrl	入力/出力	RSIP PM ドライバ管理構造体
p_digest	出力	ハッシュデータ領域 SHA224:28 byte SHA256:32 byte

Return Values

FSP_SUCCESS	正常終了
FSP_ERR_ASSERTION	ポインタ引数が NULL
FSP_ERR_NOT_OPEN	RSIP PM ドライバが Open されていない
FSP_ERR_INVALID_STATE	RSIP PM ドライバの内部状態が不正
FSP_ERR_CRYPTO_RSIP_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
FSP_ERR_CRYPTO_RSIP_FATAL	ソフトウェアの破損を検出

Description

ハッシュ演算結果を出力します。

p_digest に、R_RSIP_SHA_Update()関数で入力したメッセージのハッシュ値を出力します。

p_ctrl には、R_RSIP_SHA_Init()関数および R_RSIP_SHA_Update()関数で使用した p_ctrl を指定します。

Reentrant

非対応

4.2.6.5 R_RSIP_SHA_Suspend

Format

```
#include “r_rsip_protected_rx_if.h”
 fsp_err_t R_RSIP_SHA_Suspend(
        rsip_ctrl_t * const p_ctrl,
        rsip_sha_handle_t * const p_handle
)
```

Parameters

p_ctrl	入力/出力	RSIP PM ドライバ管理構造体
p_handle	出力	ハッシュ演算の制御情報（128 byte）

Return Values

FSP_SUCCESS	正常終了
FSP_ERR_ASSERTION	ポインタ引数が NULL
FSP_ERR_NOT_OPEN	RSIP PM ドライバが Open されていない
FSP_ERR_INVALID_STATE	RSIP PM ドライバの内部状態が不正

Description

R_RSIP_SHA_Init()関数および R_RSIP_SHA_Update()関数で実行後、他のマルチパート、シングルパート演算を実施できるように、ハッシュ演算を途中で停止します。それまでのハッシュ演算の中間値を p_handle に出力します。
R_RSIP_SHA_Suspend()関数呼び出し後、R_RSIP_SHA_Finish()関数を呼んだ場合、そこまでの R_RSIP_SHA_Update()関数に入力したメッセージのハッシュ値を出力します。
ハッシュ演算を再開したい場合は、R_RSIP_SHA_Resume()関数を呼び出してください。

p_ctrl には、R_RSIP_SHA_Init()関数および R_RSIP_SHA_Update()関数で使用した p_ctrl を指定します。

Reentrant

非対応

4.2.6.6 R_RSIP_SHA_Resume

Format

```
#include “r_rsip_protected_rx_if.h”
 fsp_err_t R_RSIP_SHA_Suspend(
        rsip_ctrl_t * const p_ctrl,
        rsip_sha_handle_t * const p_handle
)
```

Parameters

p_ctrl	入力/出力	RSIP PM ドライバ管理構造体
p_handle	入力	ハッシュ演算の制御情報

Return Values

FSP_SUCCESS	正常終了
FSP_ERR_ASSERTION	ポインタ引数が NULL
FSP_ERR_NOT_OPEN	RSIP PM ドライバが Open されていない
FSP_ERR_INVALID_STATE	RSIP PM ドライバの内部状態が不正

Description

R_RSIP_SHA_Suspend()関数で中断されたハッシュ演算を再開します。
p_handle に R_RSIP_SHA_Suspend()関数から出力された p_handle を入力してください。

p_ctrl には、R_RSIP_Open()関数で使用した管理構造体のポインタを指定します。
R_RSIP_SHA_Suspend()関数で保持されたハッシュ演算の中間値を p_ctrl に設定し、続く R_RSIP_SHA_Update()関数および R_RSIP_SHA_Finish()関数で使用します。

Reentrant

非対応

4.2.6.7 R_RSIP_HMAC_Compute

Format

```
#include “r_rsip_protected_rx_if.h”
 fsp_err_t R_RSIP_HMAC_Compute(
        rsip_ctrl_t * const p_ctrl,
        rsip_wrapped_key_t const * const p_wrapped_key,
        uint8_t const * const p_message,
        uint32_t const message_length,
        uint8_t * const p_mac
)
```

Parameters

p_ctrl	入力/出力	RSIP PM ドライバ管理構造体
p_wrapped_key	入力	Wrapped Key
p_message	入力	メッセージデータ領域
message_length	入力	メッセージデータのバイト長(0～任意 byte)
p_mac	出力	HMAC データ領域 HMAC-SHA224:28 byte HMAC-SHA256:32 byte

Return Values

FSP_SUCCESS	正常終了
FSP_ERR_ASSERTION	ポインタ引数が NULL
FSP_ERR_NOT_OPEN	RSIP PM ドライバが Open されていない
FSP_ERR_INVALID_STATE	RSIP PM ドライバの内部状態が不正
FSP_ERR_NOT_ENABLED	p_wrapped_keyにサポートしていない鍵の種類が指定された
FSP_ERR_CRYPTO_RSIP_KEY_SET_FAIL	p_wrapped_keyの検証でエラーが発生
FSP_ERR_CRYPTO_RSIP_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
FSP_ERR_CRYPTO_RSIP_FATAL	ソフトウェアの破損を検出

Description

HMAC 演算を行います。

p_wrapped_key に HMAC 生成、検証で使用する鍵の Wrapped Key を指定します。

p_ctrl には､R_RSIP_Open()関数で使用した管理構造体のポインタを指定します｡p_messageにはハッシュ演算を行うメッセージを入力し､message_length には p_message で入力するメッセージ長を指定します。

p_mac に、HMAC 演算結果を出力します。

Reentrant

非対応

4.2.6.8 R_RSIP_HMAC_Verify

Format

```
#include “r_rsip_protected_rx_if.h”
 fsp_err_t R_RSIP_HMAC_Verify(
        rsip_ctrl_t * const p_ctrl,
        rsip_wrapped_key_t const * const p_wrapped_key,
        uint8_t const * const p_message,
        uint32_t const message_length,
        uint8_t const * const p_mac,
        uint32_t const mac_length
)
```

Parameters

p_ctrl	入力/出力	RSIP PM ドライバ管理構造体
p_wrapped_key	入力	Wrapped Key
p_message	入力	メッセージデータ領域
message_length	入力	メッセージデータのバイト長(0～任意 byte)
p_mac	入力	検証する HMAC データ領域
mac_length	入力	検証する HMAC データのバイト長

Return Values

FSP_SUCCESS	正常終了
FSP_ERR_ASSERTION	ポインタ引数が NULL
FSP_ERR_NOT_OPEN	RSIP PM ドライバが Open されていない
FSP_ERR_INVALID_STATE	RSIP PM ドライバの内部状態が不正
FSP_ERR_NOT_ENABLED	p_wrapped_keyにサポートしていない鍵の種類が指定された
FSP_ERR_INVALID_ARGUMENT	入力キーのタイプまたはモードが不正
FSP_ERR_CRYPTO_RSIP_KEY_SET_FAIL	p_wrapped_keyの検証でエラーが発生
FSP_ERR_CRYPTO_RSIP_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
FSP_ERR_CRYPTO_RSIP_FATAL	ソフトウェアの破損を検出

Description

HMAC 検証を行います。
p_wrapped_key に HMAC 生成、検証で使用する鍵の Wrapped Key を指定します。

p_ctrlには､R_RSIP_Open()関数で使用した管理構造体のポインタを指定します｡p_messageにはハッシュ演算を行うメッセージを入力し､message_length には p_message で入力するメッセージ長を指定します。入力されたメッセージの HMAC 値と p_mac で入力された HMAC 値の検証を行います｡p_mac で入力された MAC 値のうち mac_length で指定されたバイト数分有効な MAC 値として検証を行います。

Reentrant
非対応

4.2.6.9 R_RSIP_HMAC_Init

Format

```
#include “r_rsip_protected_rx_if.h”
 fsp_err_t R_RSIP_HMAC_Init(
        rsip_ctrl_t * const p_ctrl,
        rsip_wrapped_key_t const * const p_wrapped_key
)
```

Parameters

p_ctrl	入力/出力	RSIP PM ドライバ管理構造体
p_wrapped_key	入力	Wrapped Key

Return Values

FSP_SUCCESS	正常終了
FSP_ERR_ASSERTION	ポインタ引数が NULL
FSP_ERR_NOT_OPEN	RSIP PM ドライバが Open されていない
FSP_ERR_INVALID_STATE	RSIP PM ドライバの内部状態が不正
FSP_ERR_NOT_ENABLED	p_wrapped_keyにサポートしていない鍵の種類が指定された
FSP_ERR_CRYPTO_RSIP_KEY_SET_FAIL	p_wrapped_keyの検証でエラーが発生
FSP_ERR_CRYPTO_RSIP_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
FSP_ERR_CRYPTO_RSIP_FATAL	ソフトウェアの破損を検出

Description

HMAC 演算を実行する準備を行います。
p_wrapped_key に HMAC 生成、検証で使用する鍵の Wrapped Key を指定します。

p_ctrl には、R_RSIP_Open()関数で使用した管理構造体のポインタを指定します。p_ctrl に、HMAC 演算を実行する準備結果を格納します。p_ctrl に出力された値を、R_RSIP_HMAC_Update()関数、R_RSIP_HMAC_SignFinish()関数および R_RSIP_HMAC_VerifyFinish()関数で使用します。

Reentrant

非対応

4.2.6.10 R_RSIP_HMAC_Update

Format

```
#include “r_rsip_protected_rx_if.h”
 fsp_err_t R_RSIP_HMAC_Update(
        rsip_ctrl_t * const p_ctrl,
        uint8_t const * const p_message,
        uint32_t const message_length
)
```

Parameters

p_ctrl	入力/出力	RSIP PM ドライバ管理構造体
p_message	入力	メッセージデータ領域
message_length	入力	メッセージデータのバイト長(0～任意 byte)

Return Values

FSP_SUCCESS	正常終了
FSP_ERR_ASSERTION	ポインタ引数が NULL
FSP_ERR_NOT_OPEN	RSIP PM ドライバが Open されていない
FSP_ERR_INVALID_STATE	RSIP PM ドライバの内部状態が不正
FSP_ERR_NOT_ENABLED	R_RSIP_HMAC_Init で入力された p_wrapped_key にサポートしていない鍵の種類が指定された
FSP_ERR_CRYPTO_RSIP_KEY_SET_FAIL	R_RSIP_HMAC_Init で入力された p_wrapped_key の検証でエラーが発生
FSP_ERR_CRYPTO_RSIP_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
FSP_ERR_CRYPTO_RSIP_FATAL	ソフトウェアの破損を検出

Description

HMAC 演算を行います。
p_message にはハッシュ演算を行うメッセージを入力し、message_length には p_message で入力するメッセージ長を指定します。

p_ctrl には、R_RSIP_HMAC_Init()関数で使用した p_ctrl を指定します。

Reentrant

非対応

4.2.6.11 R_RSIP_HMAC_SignFinish

Format

```
#include “r_rsip_protected_rx_if.h”
 fsp_err_t R_RSIP_HMAC_SignFinish(
        rsip_ctrl_t *const p_ctrl,
        uint8_t * const p_mac
)
```

Parameters

p_ctrl	入力/出力	RSIP PM ドライバ管理構造体
p_mac	出力	HMAC データ領域 HMAC-SHA224:28 byte HMAC-SHA256:32 byte

Return Values

FSP_SUCCESS	正常終了
FSP_ERR_ASSERTION	ポインタ引数が NULL
FSP_ERR_NOT_OPEN	RSIP PM ドライバが Open されていない
FSP_ERR_INVALID_STATE	RSIP PM ドライバの内部状態が不正
FSP_ERR_NOT_ENABLED	R_RSIP_HMAC_Init で入力された p_wrapped_key にサポートしていない鍵の種類が指定された
FSP_ERR_CRYPTO_RSIP_KEY_SET_FAIL	R_RSIP_HMAC_Init で入力された p_wrapped_key の検証でエラーが発生
FSP_ERR_CRYPTO_RSIP_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
FSP_ERR_CRYPTO_RSIP_FATAL	ソフトウェアの破損を検出

Description

HMAC 生成を行います。
R_RSIP_HMAC_Update()関数で入力されたメッセージの HMAC 値を p_mac に出力します。
HMAC 検証は R_RSIP_HMAC_VerifyFinish()を使用してください。

p_ctrl には、R_RSIP_HMAC_Init()関数および R_RSIP_HMAC_Update()関数で使用した p_ctrl を指定します。

Reentrant

非対応

4.2.6.12 R_RSIP_HMAC_VerifyFinish

Format

```
#include “r_rsip_protected_rx_if.h”
 fsp_err_t R_RSIP_HMAC_VerifyFinish(
        rsip_ctrl_t * const p_ctrl,
        uint8_t const * const p_mac,
        uint32_t const mac_length
)
```

Parameters

p_ctrl	入力/出力	RSIP PM ドライバ管理構造体
p_mac	入力	検証する HMAC データ領域
mac_length	入力	検証する HMAC データのバイト長

Return Values

FSP_SUCCESS	正常終了
FSP_ERR_ASSERTION	ポインタ引数が NULL
FSP_ERR_NOT_OPEN	RSIP PM ドライバが Open されていない
FSP_ERR_INVALID_STATE	RSIP PM ドライバの内部状態が不正
FSP_ERR_INVALID_SIZE	mac_length が不正
FSP_ERR_NOT_ENABLED	R_RSIP_HMAC_Init で入力された p_wrapped_key にサポートしていない鍵の種類が指定された
FSP_ERR_CRYPTO_RSIP_KEY_SET_FAIL	R_RSIP_HMAC_Init で入力された p_wrapped_key の検証でエラーが発生
FSP_ERR_CRYPTO_RSIP_FAIL	MAC 検証エラー
FSP_ERR_CRYPTO_RSIP_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
FSP_ERR_CRYPTO_RSIP_FATAL	ソフトウェアの破損を検出

Description

HMAC 検証を行います。
R_RSIP_HMAC_Update()関数で入力されたメッセージの HMAC 値と p_mac で入力された HMAC 値の検証を行います。p_mac で入力された MAC 値のうち mac_length で指定されたバイト数分有効な MAC 値として検証を行います。
HMAC 生成は R_RSIP_HMAC_SignFinish()を使用します。

p_ctrl には、R_RSIP_HMAC_Init()関数および R_RSIP_HMAC_Update()関数で使用した p_ctrl を指定します。

Reentrant

非対応

4.2.6.13 R_RSIP_HMAC_Suspend

Format

```
#include “r_rsip_protected_rx_if.h”
 fsp_err_t R_RSIP_HMAC_Suspend(
        rsip_ctrl_t * const p_ctrl,
        rsip_hmac_handle_t * const p_handle
)
```

Parameters

p_ctrl	入力/出力	RSIP PM ドライバ管理構造体
p_handle	入力	HMAC 演算の制御情報 (288 byte)

Return Values

FSP_SUCCESS	正常終了
FSP_ERR_ASSERTION	ポインタ引数が NULL
FSP_ERR_NOT_OPEN	RSIP PM ドライバが Open されていない
FSP_ERR_INVALID_STATE	RSIP PM ドライバの内部状態が不正
FSP_ERR_CRYPTO_RSIP_FATAL	ソフトウェアの破損を検出

Description

R_RSIP_HMAC_Init()関数および R_RSIP_HMAC_Update()関数で実行後、他のマルチパート、シングルパート演算を実施できるように、HMAC 演算を途中で停止します。それまでの HMAC 演算の中間値を p_handle に出力します。
R_RSIP_HMAC_Suspend()関数を呼び出した後、R_RSIP_HMAC_SignFinish()関数または R_RSIP_HMAC_VerifyFinish()関数を呼んだ場合、そこまでの R_RSIP_HMAC_Update()関数に入力したメッセージの HMAC の出力または HMAC の検証を行います。
HMAC 演算を再開したい場合は、R_RSIP_HMAC_Resume()関数を呼び出してください。

p_ctrl には、R_RSIP_HMAC_Init()関数および R_RSIP_HMAC_Update()関数で使用した p_ctrl を指定します。

Reentrant

非対応

4.2.6.14 R_RSIP_HMAC_Resume

Format

```
#include “r_rsip_protected_rx_if.h”
 fsp_err_t R_RSIP_HMAC_Resume(
        rsip_ctrl_t * const p_ctrl,
        rsip_hmac_handle_t const * const p_handle
)
```

Parameters

p_ctrl	入力/出力	RSIP PM ドライバ管理構造体
p_handle	入力	HMAC 演算の制御情報

Return Values

FSP_SUCCESS	正常終了
FSP_ERR_ASSERTION	ポインタ引数が NULL
FSP_ERR_NOT_OPEN	RSIP PM ドライバが Open されていない
FSP_ERR_INVALID_STATE	RSIP PM ドライバの内部状態が不正

Description

R_RSIP_HMAC_Suspend()関数で中断された HMAC 演算を再開します。
p_handle に R_RSIP_HMAC_Suspend()関数から出力された p_handle を入力してください。

p_ctrl には、R_RSIP_Open()関数で使用した管理構造体のポインタを指定します。
R_RSIP_HMAC_Suspend()関数で保持された HMAC 演算の中間値を p_ctrl に指定し、続く R_RSIP_HMAC_Update()関数および R_RSIP_HMAC_SignFinish()または R_RSIP_HMAC_VerifyFinish()関数で使用します。

Reentrant

非対応

4.2.7 Key Derivation Function (KDF)

4.2.7.1 R_RSIP_KDF_SHA_Init

Format

```
#include “r_rsip_protected_rx_if.h”
 fsp_err_t R_RSIP_KDF_SHA_Init(
        rsip_ctrl_t * const p_ctrl,
        rsip_hash_type_t const hash_type
)
```

Parameters

p_ctrl	入力/出力	RSIP PM ドライバ管理構造体
hash_type	入力	実行するハッシュ演算モード

Return Values

FSP_SUCCESS	正常終了
FSP_ERR_ASSERTION	ポインタ引数が NULL
FSP_ERR_NOT_OPEN	RSIP PM ドライバが Open されていない
FSP_ERR_INVALID_STATE	RSIP PM ドライバの内部状態が不正
FSP_ERR_NOT_ENABLED	この機能では設定により入力キータイプがサポートしない

Description

KDF のためのハッシュ演算を実行する準備を行います。
hash_type に実行するハッシュ演算モードを指定します。

p_ctrl には、R_RSIP_Open()関数で使用した管理構造体のポインタを指定します。p_ctrl に、ハッシュ演算を実行する準備結果を格納します。p_ctrl に出力された値を、R_RSIP_KDF_SHA_ECDHSecretUpdate()関数、R_RSIP_KDF_SHA_Update()関数および R_RSIP_KDF_SHA_Finish()関数で使用します。

Reentrant

非対応

4.2.7.2 R_RSIP_KDF_SHA_ECDHSecretUpdate

Format

```
#include “r_rsip_protected_rx_if.h”
 fsp_err_t R_RSIP_KDF_SHA_ECDHSecretUpdate(
        rsip_ctrl_t * const p_ctrl,
        rsip_wrapped_secret_t const * const p_wrapped_secret
)
```

Parameters

p_ctrl	入力/出力	RSIP PM ドライバ管理構造体
p_wrapped_secret	入力	ラップした共有秘密

Return Values

FSP_SUCCESS	正常終了
FSP_ERR_ASSERTION	ポインタ引数が NULL
FSP_ERR_NOT_OPEN	RSIP PM ドライバが Open されていない
FSP_ERR_INVALID_STATE	RSIP PM ドライバの内部状態が不正
FSP_ERR_NOT_ENABLED	この機能では設定により入力キータイプがサポートしない
FSP_ERR_CRYPTO_RSIP_FAIL	入力パラメータが不正
FSP_ERR_CRYPTO_RSIP_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
FSP_ERR_CRYPTO_RSIP_FATAL	ソフトウェアの破損またはハードウェアの障害を検出

Description

KDF のためのハッシュ演算に使用する、ラップした ECDH 共有秘密を入力します。

p_wrapped_secret にラップした共有秘密を指定します。

p_ctrl には、R_RSIP_KDF_SHA_Init()関数で使用した管理構造体のポインタを指定します。

Reentrant

非対応

4.2.7.3 R_RSIP_KDF_SHA_Update

Format

```
#include “r_rsip_protected_rx_if.h”
 fsp_err_t R_RSIP_KDF_SHA_Update(
        rsip_ctrl_t * const p_ctrl,
        uint8_t const * const p_message,
        uint32_t const message_length
)
```

Parameters

p_ctrl	入力/出力	RSIP PM ドライバ管理構造体
p_message	入力	メッセージデータ領域
message_length	入力	メッセージデータのバイト長(0～任意 byte)

Return Values

FSP_SUCCESS	正常終了
FSP_ERR_ASSERTION	ポインタ引数が NULL
FSP_ERR_NOT_OPEN	RSIP PM ドライバが Open されていない
FSP_ERR_INVALID_STATE	RSIP PM ドライバの内部状態が不正
FSP_ERR_INVALID_SIZE	入力サイズが不正
FSP_ERR_CRYPTO_RSIP_FATAL	ソフトウェアの破損またはハードウェアの障害を検出

Description

KDF のためのハッシュ演算に使用するメッセージの入力を行います。
p_message にはハッシュ演算を行うメッセージを入力し、message_length には p_message で入力するメッセージ長を指定します。

p_ctrl には、R_RSIP_KDF_SHA_Init()関数で使用した p_ctrl を指定します。

Reentrant

非対応

4.2.7.4 R_RSIP_KDF_SHA_Finish

Format

```
#include “r_rsip_protected_rx_if.h”
 fsp_err_t R_RSIP_KDF_SHA_Finish(
        rsip_ctrl_t * const p_ctrl,
        rsip_wrapped_dkm_t * const p_wrapped_dkm
)
```

Parameters

p_ctrl	入力/出力	RSIP PM ドライバ管理構造体
p_wrapped_dkm	出力	ラップした DKM (Derived Keying Material) (64 byte)

Return Values

FSP_SUCCESS	正常終了
FSP_ERR_ASSERTION	ポインタ引数が NULL
FSP_ERR_NOT_OPEN	RSIP PM ドライバが Open されていない
FSP_ERR_INVALID_STATE	RSIP PM ドライバの内部状態が不正
FSP_ERR_CRYPTO_RSIP_FAIL	入力パラメータが不正
FSP_ERR_CRYPTO_RSIP_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
FSP_ERR_CRYPTO_RSIP_FATAL	ソフトウェアの破損またはハードウェアの障害を検出

Description

ラップした DKM を生成します。

p_wrapped_dkm にラップした DKM を出力します。

p_ctrl には、R_RSIP_KDF_SHA_Init()関数、R_RSIP_KDF_SHA_ECDHSecretUpdate()関数および R_RSIP_KDF_SHA_Update()関数で使用した管理構造体のポインタを指定します。

Reentrant

非対応

4.2.7.5 R_RSIP_KDF_SHA_Suspend

Format

```
#include “r_rsip_protected_rx_if.h”
 fsp_err_t R_RSIP_KDF_SHA_Suspend(
        rsip_ctrl_t * const p_ctrl,
        rsip_kdf_sha_handle_t * const p_handle
)
```

Parameters

p_ctrl	入力/出力	RSIP PM ドライバ管理構造体
p_handle	出力	KDF SHA 演算の制御情報 (232 byte)

Return Values

FSP_SUCCESS	正常終了
FSP_ERR_ASSERTION	ポインタ引数が NULL
FSP_ERR_NOT_OPEN	RSIP PM ドライバが Open されていない
FSP_ERR_INVALID_STATE	RSIP PM ドライバの内部状態が不正

Description

R_RSIP_KDF_SHA_Init()関数、R_RSIP_KDF_SHA_ECDHSecretUpdate()関数および R_RSIP_KDF_SHA_Update()関数で実行後、他のマルチパート、シングルパート演算を実施できるように、KDF のためのハッシュ演算を途中で停止します。それまでのハッシュ演算の中間値を p_handle に出力します。
KDF のためのハッシュ演算を再開したい場合は、R_RSIP_KDF_SHA_Resume()関数を呼び出してください。

p_ctrl には、R_RSIP_KDF_SHA_Init()関数、R_RSIP_KDF_SHA_ECDHSecretUpdate()関数および R_RSIP_KDF_SHA_Update()関数で使用した p_ctrl を指定します。

Reentrant

非対応

4.2.7.6 R_RSIP_KDF_SHA_Resume

Format

```
#include “r_rsip_protected_rx_if.h”
 fsp_err_t R_RSIP_KDF_SHA_Resume(
        rsip_ctrl_t * const p_ctrl,
        rsip_kdf_sha_handle_t const * const p_handle
)
```

Parameters

p_ctrl	入力/出力	RSIP PM ドライバ管理構造体
p_handle	入力	KDF SHA 演算の制御情報

Return Values

FSP_SUCCESS	正常終了
FSP_ERR_ASSERTION	ポインタ引数が NULL
FSP_ERR_NOT_OPEN	RSIP PM ドライバが Open されていない
FSP_ERR_INVALID_STATE	RSIP PM ドライバの内部状態が不正

Description

R_RSIP_KDF_SHA_Suspend()関数で中断された KDF のためのハッシュ演算を再開します。
p_handle に R_RSIP_KDF_SHA_Suspend()関数から出力された p_handle を入力してください。

p_ctrl には、R_RSIP_Open()関数で使用した管理構造体のポインタを指定します。
R_RSIP_KDF_SHA_Suspend()関数で保持されたハッシュ演算の中間値を p_ctrl に設定し、続く R_RSIP_KDF_SHA_ECDHSecretUpdate()関数、R_RSIP_KDF_SHA_Update()関数および R_RSIP_KDF_SHA_Finish()関数で使用します。

Reentrant

非対応

4.2.7.7 R_RSIP_KDF_DKMConcatenate

Format

```
#include “r_rsip_protected_rx_if.h”
 fsp_err_t R_RSIP_KDF_DKMConcatenate(
        rsip_wrapped_dkm_t * const p_wrapped_dkm1,
        rsip_wrapped_dkm_t const * const p_wrapped_dkm2,
        uint32_t const wrapped_dkm1_buffer_length
)
```

Parameters

p_wrapped_dkm1	入力/出力	1 つ目のラップした DKM
p_wrapped_dkm2	入力	2 つ目のラップした DKM
wrapped_dkm1_buffer_length	入力	1 つ目のラップした DKM を格納しているバッファのバイト長

Return Values

FSP_SUCCESS	正常終了
FSP_ERR_ASSERTION	ポインタ引数が NULL
FSP_ERR_INVALID_SIZE	入力サイズが不正
FSP_ERR_CRYPTO_RSIP_FAIL	入力パラメータが不正

Description

2 つのラップした DKM を連接します。

p_wrapped_dkm1 に 1 つ目のラップした DKM、p_wrapped_dkm2 に 2 つ目のラップした DKM、wrapped_dkm_buffer_length に 1 つ目のラップした DKM を格納しているバッファのサイズを指定します。連接した DKM は p_wrapped_dkm1 に出力されます。

Reentrant

非対応

4.2.7.8 R_RSIP_KDF_DerivedKeyImport

Format

```
#include “r_rsip_protected_rx_if.h”
 fsp_err_t R_RSIP_KDF_DerivedKeyImport(
        rsip_ctrl_t * const p_ctrl,
        rsip_wrapped_dkm_t const * const p_wrapped_dkm,
        uint32_t const position,
        rsip_wrapped_key_t * const p_wrapped_key
)
```

Parameters

p_ctrl	入力/出力	RSIP PM ドライバ管理構造体
p_wrapped_dkm	入力	ラップした DKM
position	入力	DKM から出力する鍵のデータのバイト位置
p_wrapped_key	出力	Wrapped Key (p_value に表 2-10で定義されている各鍵の種類に対応したサイズが出力されます。)

Return Values

FSP_SUCCESS	正常終了
FSP_ERR_ASSERTION	ポインタ引数が NULL
FSP_ERR_NOT_OPEN	RSIP PM ドライバが Open されていない
FSP_ERR_INVALID_STATE	RSIP PM ドライバの内部状態が不正
FSP_ERR_NOT_ENABLED	この機能では設定により入力キータイプがサポートしない
FSP_ERR_CRYPTO_RSIP_FAIL	入力パラメータが不正
FSP_ERR_CRYPTO_RSIP_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
FSP_ERR_CRYPTO_RSIP_FATAL	ソフトウェアの破損またはハードウェアの障害を検出

Description

DKM から Wrapped Key を出力します。

p_wrapped_dkm にラップした DKM、position に復号した DKM から出力する鍵のデータのバイト位置を指定してください。p_wrapped_key の p_value フィールドに指定されたアドレスに Wrapped Key のバイナリデータを出力します。
p_wrapped_key の type フィールドには表 2-4に定義される値を入力してください。p_wrapped_key に出力される Wrapped Key のバイトサイズは、表 2-10で定義されている各鍵の種類に対応したサイズになります。type フィールドで指定したアルゴリズムに合わせて、必要なサイズ以上の領域を確保し、アドレスを p_value フィールドに指定してください。

p_ctrl には、R_RSIP_KDF_SHA_Init()関数、R_RSIP_KDF_SHA_ECDHSecretUpdate()関数、R_RSIP_KDF_SHA_Update()関数および R_RSIP_KDF_SHA_Finish()関数で使用した管理構造体のポインタを指定します。

Reentrant
非対応

4.2.7.9 R_RSIP_KDF_DerivedIVWrap

Format

```
#include “r_rsip_protected_rx_if.h”
 fsp_err_t R_RSIP_KDF_DerivedIVWrap(
        rsip_ctrl_t * const p_ctrl,
        rsip_wrapped_dkm_t const * const p_wrapped_dkm,
        rsip_initial_vector_type_t const initial_vector_type,
        uint32_t const position,
        uint8_t const * const p_tls_sequence_num,
        uint8_t * const p_wrapped_initial_vector
)
```

Parameters

p_ctrl	入力/出力	RSIP PM ドライバ管理構造体
p_wrapped_dkm	入力	ラップした DKM
initial_vector_type	入力	出力する初期化ベクタの種類
position	入力	DKM から出力する鍵のデータのバイト位置
p_tls_sequence_num	入力	TLS 使用時に設定するシーケンス番号 NULL を指定してください。
p_wrapped_initial_vector	出力	ラップした初期化ベクタ（36 byte）

Return Values

FSP_SUCCESS	正常終了
FSP_ERR_ASSERTION	ポインタ引数が NULL
FSP_ERR_NOT_OPEN	RSIP PM ドライバが Open されていない
FSP_ERR_INVALID_STATE	RSIP PM ドライバの内部状態が不正
FSP_ERR_NOT_ENABLED	この機能では設定により入力キータイプがサポートしない
FSP_ERR_CRYPTO_RSIP_FAIL	入力パラメータが不正
FSP_ERR_CRYPTO_RSIP_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
FSP_ERR_CRYPTO_RSIP_FATAL	ソフトウェアの破損またはハードウェアの障害を検出

Description

DKM からラップした初期化ベクタを出力します。

p_wrapped_dkm にラップした DKM、initial_vector_type に出力する初期化ベクタの種類、position に復号した DKM から出力する鍵のデータのバイト位置、p_tls_sequence_num に NULL を指定してください。
p_wrapped_initial_vector にラップした初期化ベクタを出力します。

p_ctrl には、R_RSIP_KDF_SHA_Init()関数、R_RSIP_KDF_SHA_ECDHSecretUpdate()関数、R_RSIP_KDF_SHA_Update()関数および R_RSIP_KDF_SHA_Finish()関数で使用した管理構造体のポインタを指定します。

Reentrant
非対応

4.2.8 Key Wrap

4.2.8.1 R_RSIP_RFC3394_KeyWrap

Format

```
#include “r_rsip_protected_rx_if.h”
 fsp_err_t R_RSIP_RFC3394_KeyWrap(
        rsip_ctrl_t * const p_ctrl,
        rsip_wrapped_key_t const * const p_wrapped_kek,
        rsip_wrapped_key_t const * const p_wrapped_target_key,
        uint8_t * const p_rfc3394_wrapped_target_key
)
```

Parameters

p_ctrl	入力/出力	RSIP PM ドライバ管理構造体
p_wrapped_kek	入力	ラップに使用する鍵
p_wrapped_target_key	入力	ラップする鍵
p_rfc3394_wrapped_target_key	出力	ラップされた鍵 AES128 鍵：24 byte AES256 鍵：40 byte

Return Values

FSP_SUCCESS	正常終了
FSP_ERR_ASSERTION	ポインタ引数が NULL
FSP_ERR_NOT_OPEN	RSIP PM ドライバが Open されていない
FSP_ERR_INVALID_STATE	RSIP PM ドライバの内部状態が不正
FSP_ERR_NOT_ENABLED	この機能では設定により入力キータイプがサポートしない
FSP_ERR_INVALID_ARGUMENT	入力キーのタイプまたはモードが不正
FSP_ERR_CRYPTO_RSIP_FAIL	入力パラメータが不正
FSP_ERR_CRYPTO_RSIP_KEY_SET_FAIL	異常な Key が入力
FSP_ERR_CRYPTO_RSIP_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
FSP_ERR_CRYPTO_RSIP_FATAL	ソフトウェアの破損またはハードウェアの障害を検出

Description

RFC3394 に準拠したアルゴリズムで鍵をラップします。
p_wrapped_kek で指定した鍵で、p_wrapped_target_key をラップします。ラップされた鍵は p_rfc3394_wrapped_target_key に出力されます。

p_ctrl には、R_RSIP_Open()関数で使用した管理構造体のポインタを指定します。

Reentrant

非対応

4.2.8.2 R_RSIP_RFC3394_KeyUnwrap

Format

```
#include “r_rsip_protected_rx_if.h”
 fsp_err_t R_RSIP_RFC3394_KeyUnwrap(
        rsip_ctrl_t * const p_ctrl,
        rsip_wrapped_key_t const * const p_wrapped_kek,
        rsip_key_type_t const key_type,
        uint8_t const * const p_rfc3394_wrapped_target_key,
        rsip_wrapped_key_t * const p_wrapped_target_key
)
```

Parameters

p_ctrl	入力/出力	RSIP PM ドライバ管理構造体
p_wrapped_kek	入力	アンラップに使用する鍵
key_type	入力	出力する Wrapped Key の種類
p_rfc3394_wrapped_target_key	入力	ラップされた鍵
p_wrapped_target_key	入力/出力	アンラップした鍵 (p_value に表 2-10で定義されている各鍵の種類に対応したサイズが出力されます。)

Return Values

FSP_SUCCESS	正常終了
FSP_ERR_ASSERTION	ポインタ引数が NULL
FSP_ERR_NOT_OPEN	RSIP PM ドライバが Open されていない
FSP_ERR_INVALID_STATE	RSIP PM ドライバの内部状態が不正
FSP_ERR_NOT_ENABLED	この機能では設定により入力キータイプがサポートしない
FSP_ERR_INVALID_ARGUMENT	入力キーのタイプまたはモードが不正
FSP_ERR_CRYPTO_RSIP_FAIL	入力パラメータが不正
FSP_ERR_CRYPTO_RSIP_KEY_SET_FAIL	異常な Key が入力
FSP_ERR_CRYPTO_RSIP_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
FSP_ERR_CRYPTO_RSIP_FATAL	ソフトウェアの破損またはハードウェアの障害を検出

Description

RFC3394 に準拠したアルゴリズムで鍵をアンラップします。

p_wrapped_kek で指定した鍵を使用して、p_rfc3394_wrapped_target_key をアンラップします。アンラップされた鍵は Wrapped Key で p_wrapped_target_key に出力されます。key_type にアンラップされる鍵の種類を指定してください。

p_ctrl には、R_RSIP_Open()関数で使用した管理構造体のポインタを指定します。

Reentrant
非対応

4.2.9　セキュアブート/ファームウェアアップデート

4.2.9.1　R_RSIP_FWUP_StartUpdateFirmware

Format

```
#include “r_rsip_protected_rx_if.h”
  fsp_err_t R_RSIP_FWUP_StartUpdateFimware(
        rsip_ctrl_t * const p_ctrl
)
```

Parameters

p_ctrl	入力/出力	RSIP PM ドライバ管理構造体

Return Values

FSP_SUCCESS	正常終了
FSP_ERR_ASSERTION	ポインタ引数が NULL
FSP_ERR_NOT_OPEN	RSIP PM ドライバが Open されていない
FSP_ERR_INVALID_STATE	RSIP PM ドライバの内部状態が不正
FSP_ERR_CRYPTO_RSIP_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遷移します。

p_ctrl には、R_RSIP_Open()関数で使用した管理構造体のポインタを指定します。

Reentrant

非対応

4.2.9.2 R_RSIP_FWUP_MAC_Sign_Init

Format

```
#include “r_rsip_protected_rx_if.h”
 fsp_err_t R_RSIP_FWUP_MAC_Sign_Init(
        rsip_ctrl_t * const p_ctrl,
        rsip_wrapped_key_t const * const p_wrapped_key_encryption_key,
        uint8_t const * const p_encrypted_image_encryption_key,
        uint8_t const * const p_initial_vector,
        uint32_t const firmware_size
)
```

Parameters

p_ctrl	入力/出力	RSIP PM ドライバ管理構造体
p_wrapped_key_encryption_key	入力	Image Encryption key をラップするのに使用した鍵の Wrapped Key
p_encrypted_image_encryption_key	入力	Image Encryption key を Key Encryption Key でラップした鍵
p_initial_vector	入力	ファームウェアの暗号化で使用した初期化ベクタ
firmware_size	入力	暗号化されたファームウェアの全バイト長 (0～任意 byte)

Return Values

FSP_SUCCESS	正常終了
FSP_ERR_ASSERTION	ポインタ引数が NULL
FSP_ERR_NOT_OPEN	RSIP PM ドライバが Open されていない
FSP_ERR_INVALID_STATE	RSIP PM ドライバの内部状態が不正
FSP_ERR_CRYPTO_RSIP_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
FSP_ERR_CRYPTO_RSIP_KEY_SET_FAIL	異常な鍵が入力されている

Description

暗号化されたファームウェアを復号し、平文ファームウェアの MAC を生成する準備を行います。

p_wrapped_key_encryption_key に、ファームウェアを暗号化した鍵(Image Encryption Key)をラップするのに使用した鍵(Key Encryption Key)の Wrapped Key を、p_encrypted_image_encryption_key には、Key Encryption Key でラップされた Image Encryption key を入力します。p_initial_vector には、ファームウェアの暗号化で使用した初期化ベクタを入力します。firmware_size には、復号する暗号化されたファームウェアのバイトサイズを入力します。

p_ctrl には、R_RSIP_Open()関数で使用した管理構造体のポインタを指定します。p_ctrl には、R_RSIP_FWUP_MAC_Sign_Init()関数の実行結果が格納されます。p_ctrl に出力された値を、R_RSIP_FWUP_MAC_Sign_Update()関数および R_RSIP_FWUP_MAC_Sign_Finish()関数で使用します。

Reentrant

非対応

4.2.9.3 R_RSIP_FWUP_MAC_Sign_Update

Format

```
#include “r_rsip_protected_rx_if.h”
fsp_err_t R_RSIP_FWUP_MAC_Sign_Update(
        rsip_ctrl_t * const p_ctrl,
        uint8_t const * const p_input,
        uint8_t * const p_output,
        uint32_t const length
)
```

Parameters

p_ctrl	入力/出力	RSIP PM ドライバ管理構造体
p_input	入力	暗号化されたファームウェア
p_output	出力	平文ファームウェア (length と同じサイズが出力されます)
length	入力	入力する暗号化されたファームウェアのバイト長 (0~任意 byte、16 の倍数である必要があります)

Return Values

FSP_SUCCESS	正常終了
FSP_ERR_ASSERTION	ポインタ引数が NULL
FSP_ERR_NOT_OPEN	RSIP PM ドライバが Open されていない
FSP_ERR_INVALID_STATE	RSIP PM ドライバの内部状態が不正
FSP_ERR_INVALID_ARGUMENT	入力データが不正

Description

p_input に入力された暗号化されたファームウェアを、length で指定されたサイズ分復号し、平文ファームウェアを p_output に出力します。

暗号化されたファームウェアが複数のエリアに存在しているなど、復号処理を複数回に分けて行う必要がある場合、R_RSIP_FWUP_MAC_Sign_Update()使用して復号します。ただし最終 16 バイトを含む暗号化されたファームウェアは R_RSIP_FWUP_MAC_Sign_Finish()を使用して復号します。複数回に分けて復号処理を行う必要がない場合、R_RSIP_FWUP_MAC_Sign_Update()は使用せず、R_RSIP_FWUP_MAC_Sign_Finish()で復号してください。

p_ctrl には、R_RSIP_FWUP_MAC_Sign_Init()関数で使用した p_ctrl を指定します。

Reentrant

非対応

4.2.9.4 R_RSIP_FWUP_MAC_Sign_Finish

Format

```
#include “r_rsip_protected_rx_if.h”
fsp_err_t R_RSIP_FWUP_MAC_Sign_Finish(
        rsip_ctrl_t * const p_ctrl,
        uint8_t const * const p_input,
        uint8_t const * const p_input_mac,
        uint32_t const length,
        uint8_t * const p_output,
        uint8_t * const p_output_mac
)
```

Parameters

p_ctrl	入力/出力	RSIP PM ドライバ管理構造体
p_input	入力	入力ファームウェア領域
p_input_mac	入力	入力ファームウェアの MAC 値(16 バイト)
length	入力	入力ファームウェアのバイト長 (0~任意 byte、16 の倍数である必要があります)
p_output	出力	出力ファームウェア領域 (length と同じサイズが出力されます)
p_output_mac	出力	出力ファームウェアの MAC 値(16 バイト)

Return Values

FSP_SUCCESS	正常終了
FSP_ERR_ASSERTION	ポインタ引数が NULL
FSP_ERR_NOT_OPEN	RSIP PM ドライバが Open されていない
FSP_ERR_INVALID_STATE	RSIP PM ドライバの内部状態が不正
FSP_ERR_CRYPTO_RSIP_FAIL	異常終了

Description

p_input に入力された暗号化されたファームウェアを、length で指定されたサイズ分復号し、平文ファームウェアを p_output に出力します。その後、p_input_mac に入力された暗号化されたファームウェアの MAC 検証を行います。暗号化されたファームウェアの MAC 検証に成功した場合、平文ファームウェアの MAC 値を生成し、p_output_mac に出力します。

p_ctrl には、R_RSIP_FWUP_MAC_Sign_Init()関数および R_RSIP_FWUP_MAC_Sign_Update()関数で使用した p_ctrl を指定します。

Reentrant

非対応

4.2.9.5 R_RSIP_SB_MAC_Verify_Init

Format

```
#include “r_rsip_protected_rx_if.h”
  fsp_err_t R_RSIP_SB_MAC_Verify_Init(
        rsip_ctrl_t * const p_ctrl
)
```

Parameters

p_ctrl	入力/出力	RSIP PM ドライバ管理構造体

Return Values

FSP_SUCCESS	正常終了
FSP_ERR_ASSERTION	ポインタ引数が NULL
FSP_ERR_NOT_OPEN	RSIP PM ドライバが Open されていない
FSP_ERR_INVALID_STATE	RSIP PM ドライバの内部状態が不正

Description

平文ファームウェアの MAC 検証の準備を行います。

p_ctrl には、R_RSIP_Open()関数で使用した管理構造体のポインタを指定します。p_ctrl には、R_RSIP_SB_MAC_Verify_Init()関数の実行結果が格納されます。p_ctrl に出力された値を、R_RSIP_SB_MAC_Verify_Update()関数および R_RSIP_SB_MAC_Verify_Finish()関数で使用します。

Reentrant

非対応

4.2.9.6 R_RSIP_SB_MAC_Verify_Update

Format

```
#include “r_rsip_protected_rx_if.h”
  fsp_err_t R_RSIP_SB_MAC_Verify_Update(
        rsip_ctrl_t * const p_ctrl,
        uint8_t const * const p_input,
        uint32_t const input_length
)
```

Parameters

p_ctrl	入力/出力	RSIP PM ドライバ管理構造体
p_input	入力	入力ファームウェア領域
input_length	入力	入力ファームウェアのバイト長 (0~任意 byte、16 の倍数である必要があります)

Return Values

FSP_SUCCESS	正常終了
FSP_ERR_ASSERTION	ポインタ引数が NULL
FSP_ERR_NOT_OPEN	RSIP PM ドライバが Open されていない
FSP_ERR_INVALID_STATE	RSIP PM ドライバの内部状態が不正
FSP_ERR_CRYPTO_RSIP_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
FSP_ERR_CRYPTO_RSIP_FAIL	異常終了

Description

p_input に入力された平文ファームウェアを、input_length で指定されたサイズ分入力し、MAC 演算を行います。

平文ファームウェアが複数のエリアに存在しているなど、平文ファームウェアの入力を複数回に分けて行う必要がある場合、R_RSIP_SB_MAC_Verify_Update()を使用します。最終 16 バイトを含む平文ファームウェアは R_RSIP_SB_MAC_Verify_Finish()で入力します。平文ファームウェア入力を分ける必要がない場合は、R_RSIP_SB_MAC_Verify_Update()を使用せず、R_RSIP_SB_MAC_Verify_Finish()を使用します。

p_ctrl には、R_RSIP_SB_MAC_Verify_Init()関数で使用した p_ctrl を指定します。

Reentrant

非対応

4.2.9.7 R_RSIP_SB_MAC_Verify_Finish

Format

```
#include “r_rsip_protected_rx_if.h”
fsp_err_t R_RSIP_SB_MAC_Verify_Finish(
        rsip_ctrl_t * const p_ctrl,
        uint8_t const * const p_input,
        uint8_t const * const p_mac,
        uint32_t const input_length
)
```

Parameters

p_ctrl	入力/出力	RSIP PM ドライバ管理構造体
p_input	入力	入力ファームウェア領域
p_mac	入力	入力ファームウェアの MAC 値(16 バイト)
input_length	入力	入力ファームウェアのバイト長 (0~任意 byte、16 の倍数である必要があります)

Return Values

FSP_SUCCESS	正常終了
FSP_ERR_ASSERTION	ポインタ引数が NULL
FSP_ERR_NOT_OPEN	RSIP PM ドライバが Open されていない
FSP_ERR_INVALID_STATE	RSIP PM ドライバの内部状態が不正
FSP_ERR_CRYPTO_RSIP_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
FSP_ERR_CRYPTO_RSIP_FAIL	異常終了

Description

入力された平文のファームウェアの MAC 検証を行います。

p_ctrl には、R_RSIP_SB_MAC_Verify_Init()関数および R_RSIP_SB_MAC_Verify_Update()関数で使用した引数を指定します。

Reentrant

非対応

5. 鍵の注入と更新

本章では、RSIP PM ドライバが扱う暗号鍵をフラッシュメモリなどの不揮発性メモリにプログラムする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。

RSIP PM ドライバの鍵注入のメカニズムは3.6.1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 を参照してください。

3.6.1.1 鍵のラップアルゴリズムに記載されている手順に従い、ユーザ鍵を UFPK（ufpk.bin）でラップした Encrypted Key を生成します。ユーザ鍵のフォーマットは5.3を参照してください。

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。

使用する API については、図 3-3 鍵の注入と更新のフローを参照してください。

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メモリに注入します。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、5.4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。

RSIP PM ドライバの鍵更新のメカニズムは3.6.1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で紹介する方法を用いてフラッシュメモリ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、その後フラッシュメモリに書き込む処理が必要です。

   使用する API については、図 3-3 鍵の注入と更新のフロー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メモリに注入します。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、フラッシュメモリ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

   3.6.1.1 鍵のラップアルゴリズムに記載されている手順に従い、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 のフラッシュメモリ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、5.4 を参照してください。

5.3　3.6.1.1ユーザ鍵暗号化フォーマット

ユーザ鍵/KUK を暗号化する時の入力の形式を以下に示します。3.6.1.1 鍵のラップアルゴリズムに示す方法で暗号化と MAC の連結を行うことで、Encrypted Key を生成します。

・AES 128bit 鍵

入力(User Key)

byte	16			
	4	4	4	4
0-15	128 bit AES 鍵			

・AES 256bit 鍵

入力(User Key)

byte	16			
	4	4	4	4
0-31	256 bit AES 鍵			

・ECC secp256r1, brainpoolP256r1, secp256k1 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			

・ECC secp256r1, brainpoolP256r1, secp256k1 秘密鍵

入力(User Key)

byte	16			
	4	4	4	4
0-31	ECC 256 bit 秘密鍵 d			

・HMAC-SHA256 鍵

入力(User Key)

byte	16			
	4	4	4	4
0-31	HMAC-SHA256 鍵			

・KUK

入力(User Key)

byte	16			
	4	4	4	4
0-15	AES 128bit CBC 鍵			
16-31	AES 128bit CBCMAC 鍵			

5.4　Security Key Management Tool を使用した Encrypted Key の生成方法

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。

5.4.1　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。

5.4.1.1　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 "2222222222222222222222222222222211111111111111111111111111111111"

/output "C:¥work¥ufpk.key"

```
C:¥Renesas¥SecurityKeyMangementTool¥cli>skmt.exe /genufpk /ufpk "22222222222222222222222222222221
1111111111111111111111111111111111" /output "C:¥work¥ufpk.key"
UFPK: 22222222222222222222222222222221111111111111111111111111111111111
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-RSIP-E11A" /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。

図 5-2 genkey コマンド実行例

5.4.1.2 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。

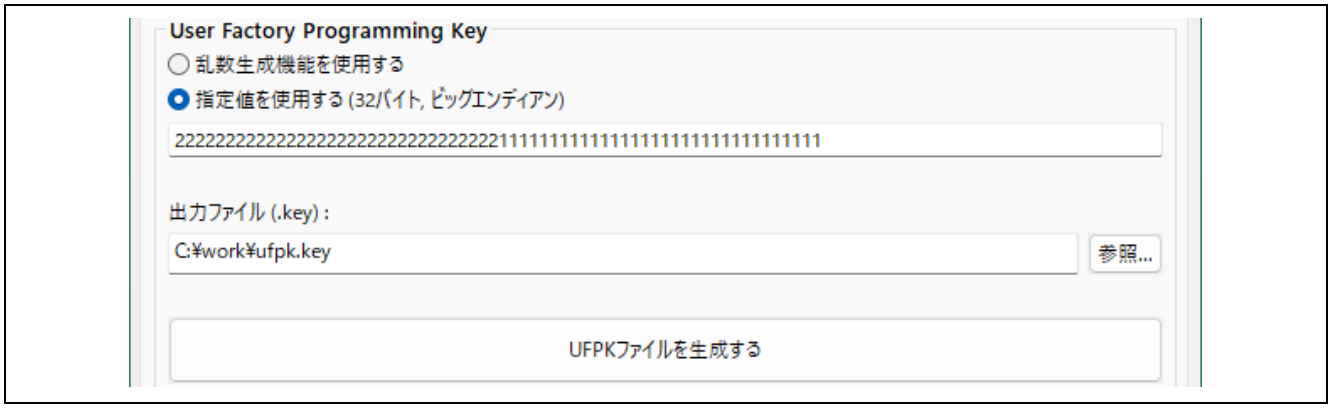

図 5-4 [UFPK 生成]タブ 指定値で UFPK を生成する場合の例

“**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。

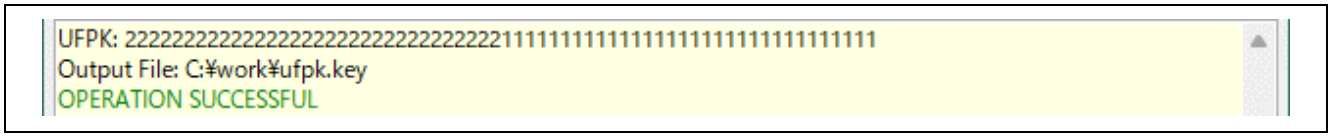

図 5-5 [UFPK 生成]タブ 実行結果

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

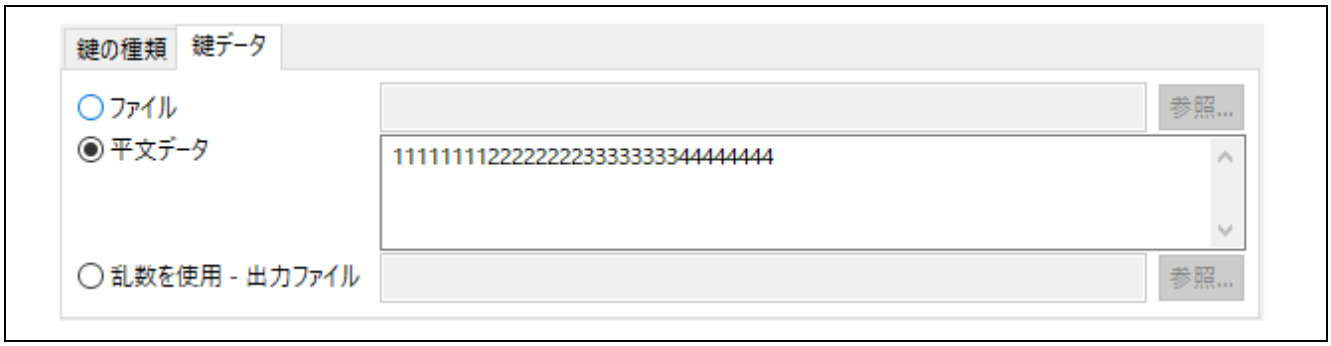

図 5-7 [鍵のラッピング] - [鍵のデータ]タブ AES128 鍵ファイルを C ソース出力設定例

正常に終了すると以下のように表示されます。

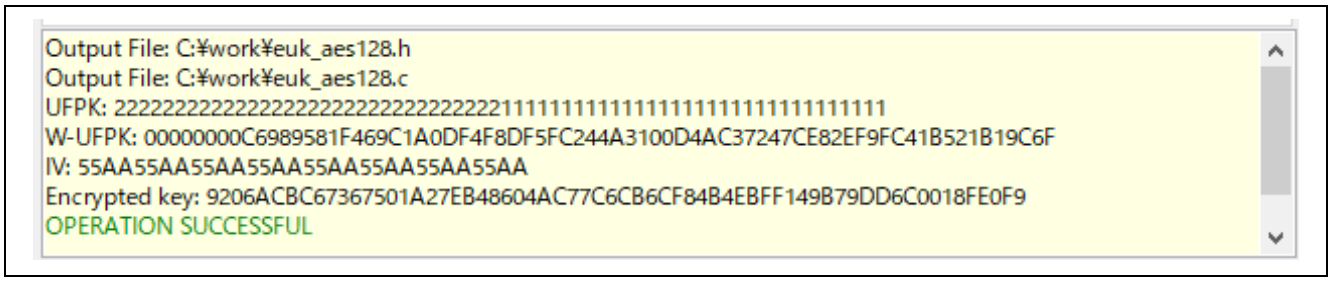

図 5-8 [鍵のラッピング]タブ 実行結果

出力された C ソースファイルのデータを、5.4.1.1と同様の方法で組み込みます。

5.4.2 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。

5.4.2.1 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 "22222222222222222222222222222222111111111111111111111111111111111"
        /output "C:¥work¥ufpk.key"
```

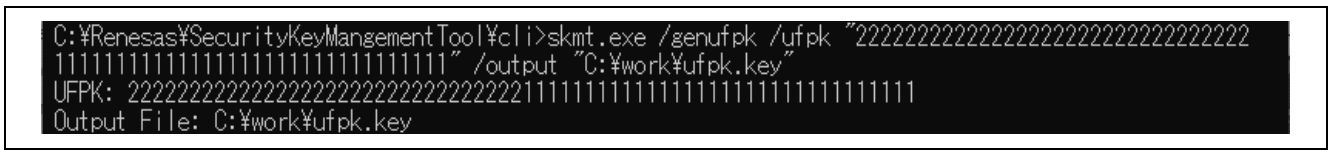

図 5-9 genufpk コマンド実行結果

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 5-10genkuk 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-RSIP-E11A" /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。

```
C:\Renesas\SecurityKeyManagementTool\cli>skmt.exe /genkey /ufpk file="c:\work\ufpk.key" /wuf
pk file="C:\work\ufpk.key_enc.key" /mcu "RX-RSIP-E11A" /keytype "key-update-key" /key file="
C:\work\kuk.key" /filetype "csource" /output "C:\work\kuk.c"
Output File: C:\work\kuk.h
Output File: C:\work\kuk.c
UFPK: 22222222222222222222222222222221111111111111111111111111111111
W-UFPK: 00000000C6989581F469C1A0DF4F8DF5FC244A3100D4AC37247CE82EF9FC41B521B19C6F
IV: 0DA41458F21567D8AB26503AA287EA0A
Encrypted key: CDB1E35254A026DC0898F5DE60A8639D7168A034E9454CC3D0D0B333997EE0571598182CD3771
AA893843B43C7F0858A
```

図 5-11genkey 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_RSIP_InitialKeyWrap()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. ECC secp256r1 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-RSIP-E11A"
/keytype "secp256r1-public" /key
"19b3f37e35d0a5448983bfc91f69b8e167c135fa0f863d6d0efb99fce34f593823b8eb34f45ae0197aef66426a08459019d63b04bc5eccf3b428181a92f3ff9c"
/filetype "csource" /keyname "secp256r1public"
/output "C:¥work¥secp256r1public.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\SecurityKeyManagementTool\cli>skmt.exe /genkey /kuk file="C:\work\kuk.key" /mcu "
RX-RSIP-E11A" /keytype "secp256r1-public" /key "19b3f37e35d0a5448983bfc91f69b8e167c135fa0f86
3d6d0efb99fce34f593823b8eb34f45ae0197aef66426a08459019d63b04bc5eccf3b428181a92f3ff9c" /filet
ype "csource" /keyname "secp256r1public" /output "C:\work\secp256r1public.c"
Output File: C:\work\secp256r1public.h
Output File: C:\work\secp256r1public.c
KUK: D0AEC19726CBC0E2FB403866B9B465A6C0D05B7A60362D5F435F9A3E98C79084
IV: 86077570BC362CBFD707CA84C5D118C2
Encrypted key: 89FF8126D500B5C25ADB98A96552809F32E1B7B5177427FFC9A91A6935F7F5CBECE6E16636ADD
86E371A30A4D3D8355AC5669F0C18ED8CC8E6A4795D8A8C12B4C397302906A54064BE344698D723797F
```

図 5-12genkey 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_RSIP_EncryptedKeyWrap()の引数に渡してください。更新する secp256r1 公開鍵の Wrapped Key を出力します。

5.4.2.2 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。

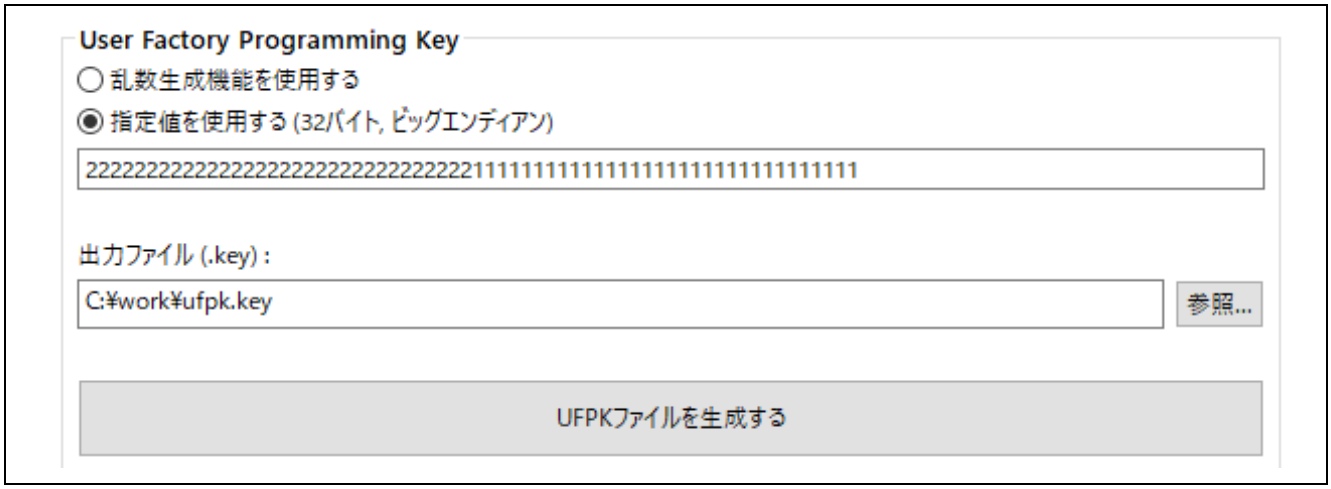

図 5-14[UFPK 生成]タブ UFPK 生成設定例

“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。

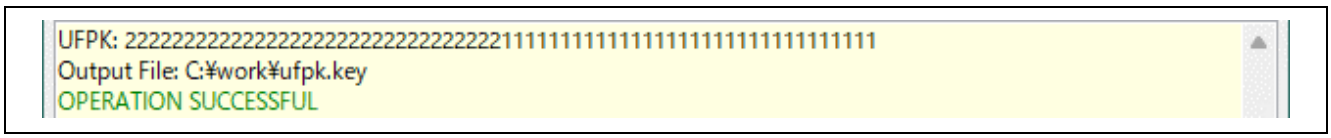

図 5-15[UFPK 生成]タブ 実行結果

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。

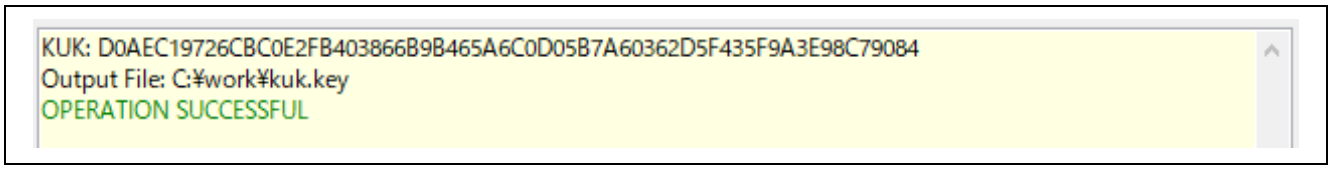

図 5-17[KUK 生成]タブ 実行結果

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。

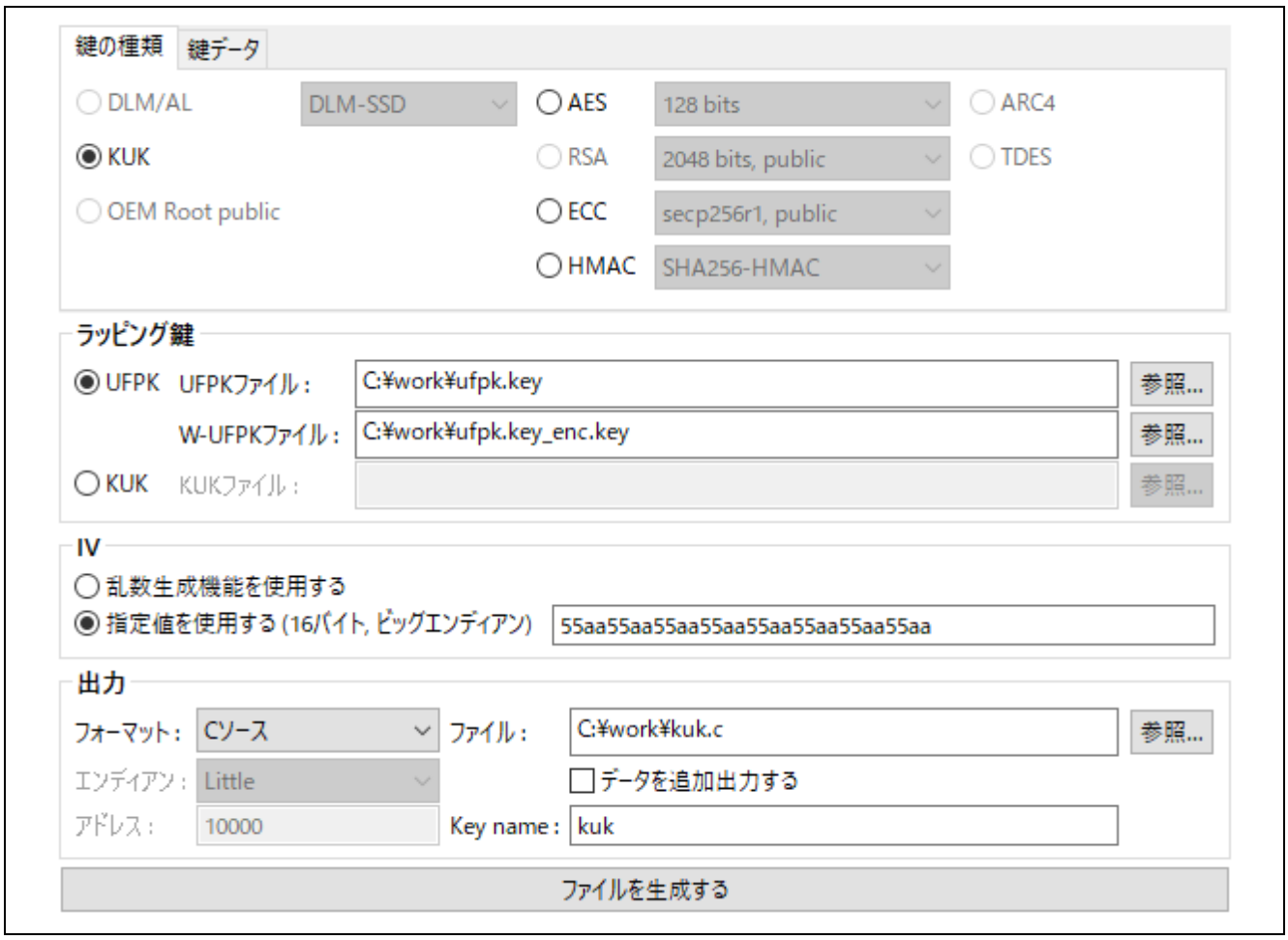

図 5-18[鍵のラッピング] – [鍵の種類]タブ KUK ファイルの C ソースファイル出力設定例

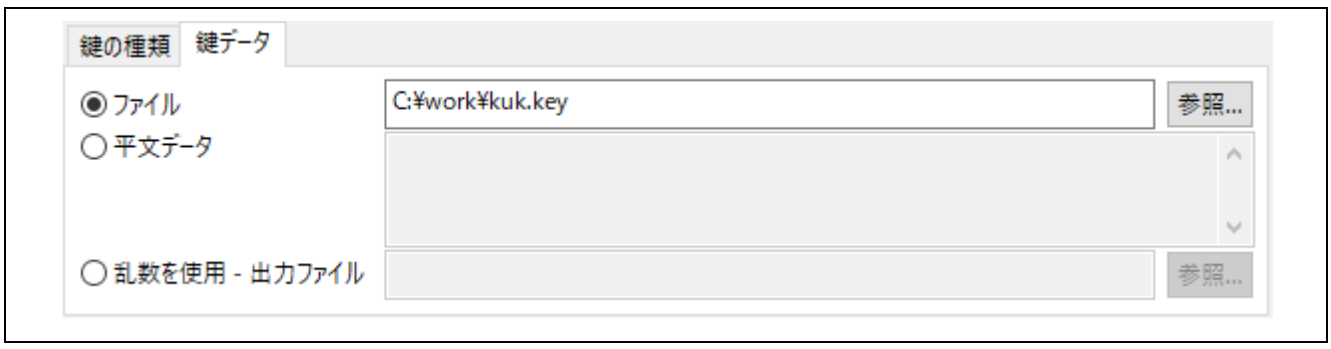

図 5-19[鍵のラッピング] – [鍵の種類]タブ KUK ファイルを RFP ファイル出力設定例

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_RSIP_InitialKeyWrap() 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 secp256r1 公開鍵ファイルの生成

[**鍵のラッピング**]タブを使用して secp256r1 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”ECC”と”secp256r1,public”を選択し、[**鍵データ**] タブで secp256r1 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 secp256r1 公開鍵 C ソースファイル生成例

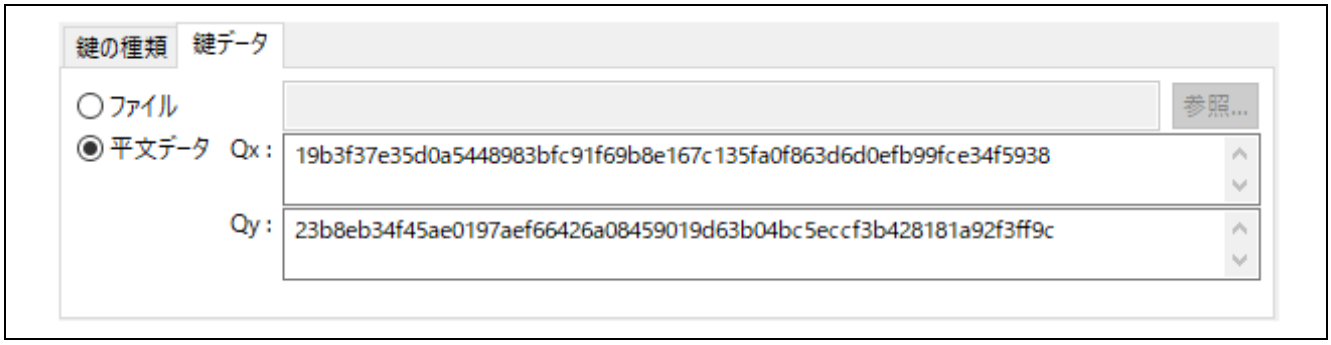

図 5-21[鍵のラッピング] – [鍵データ]タブ secp256r1 公開鍵 C ソースファイル生成例

正常終了すると以下のように出力されます。

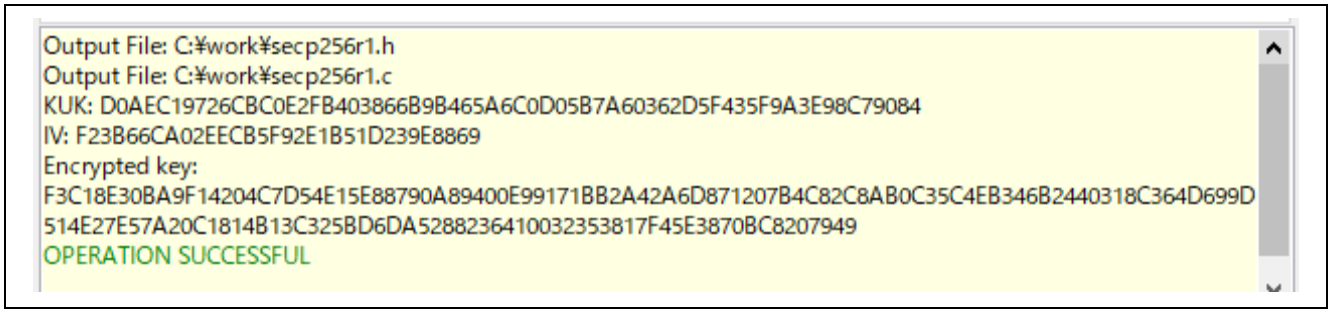

図 5-22[鍵のラッピング]タブ　実行結果

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 鍵注入と暗号の使用方法

表 6-1に示すデモプロジェクトで、RSIP ドライバが提供する暗号演算、乱数生成、および鍵注入用の API の使用方法を確認することができます。

表 6-1 鍵注入と暗号の使用方法のデモプロジェクト

MCU	デモプロジェクト
RX261	rx261_ek_rsip_sample

デモプロジェクトでは実行結果を UART で出力します。ターミナルソフトをインストールした PC とデモプロジェクト実行ボードを接続します。以降の説明では、PC 上の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 で動作します。

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。

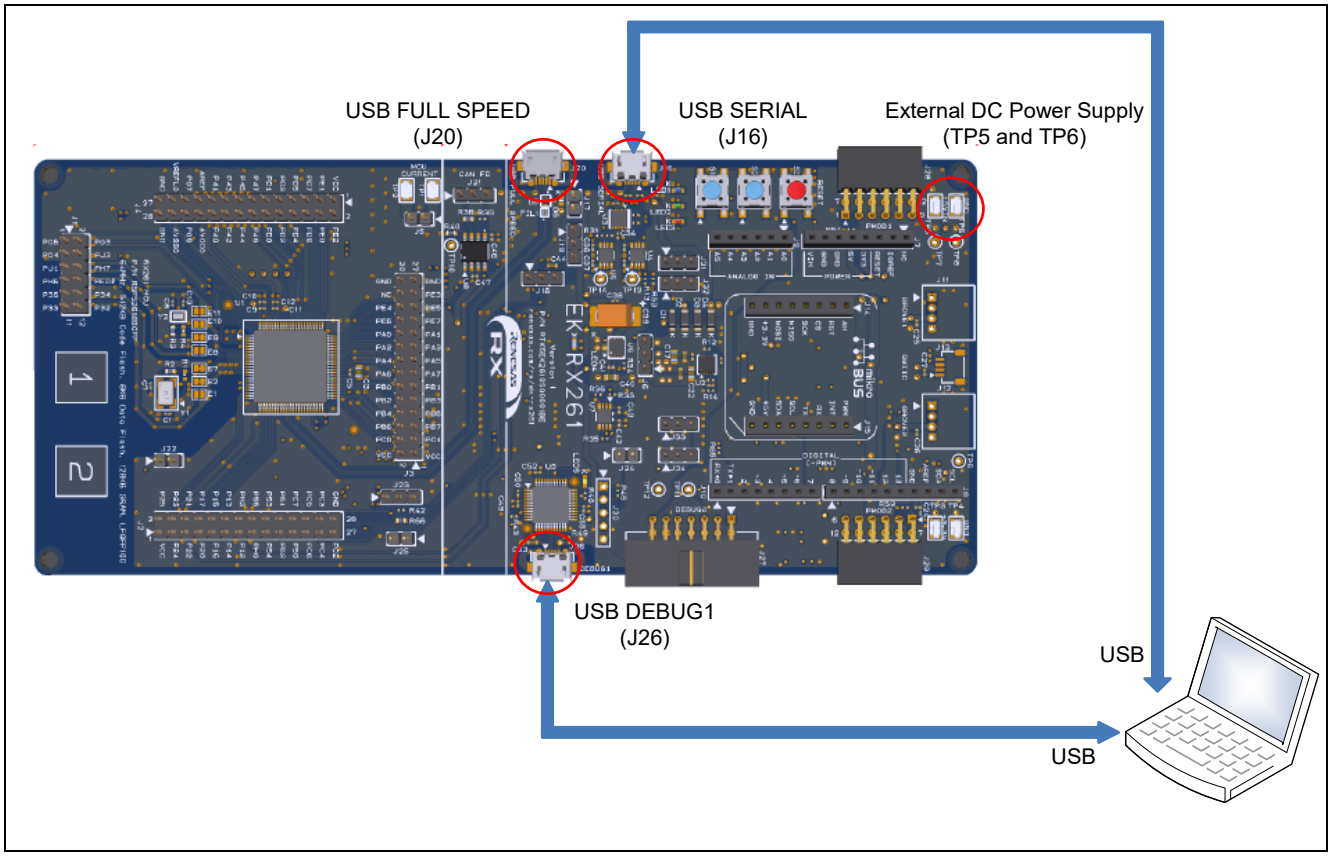

図 6-1 EK-RX261 と PC の接続

EK-RX261 は USB DEBUG1、USB FULL SPEED、USB SERIAL、および外部直流電源の端子から電源を供給することができます。いずれかの方法で電源を供給してください。

Tera Term のシリアルポート設定と端末の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

デモプロジェクトの動作フローを図 6-2に示します。

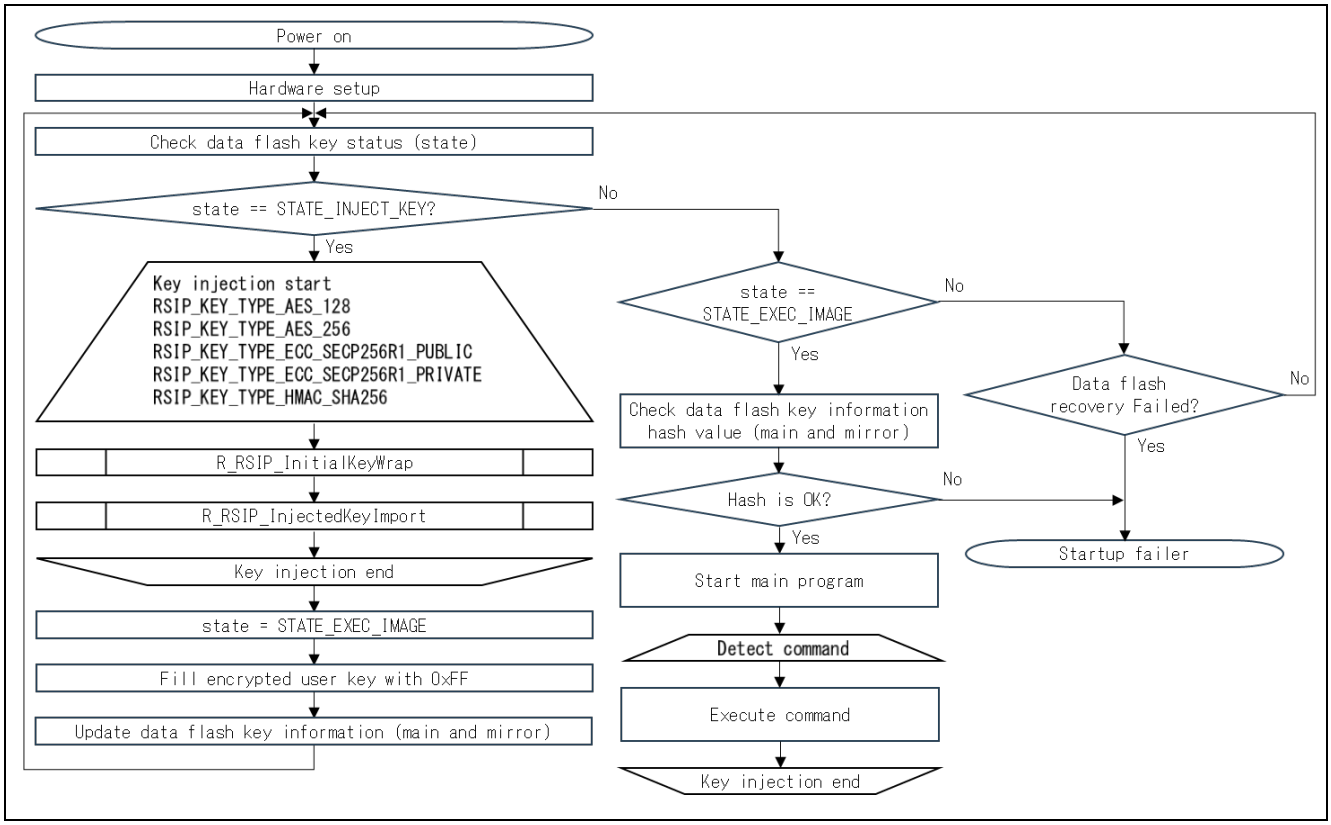

図 6-2 鍵注入と暗号の使用方法のデモプロジェクトの動作フロー

デモプロジェクトは状態遷移で動作を管理しています。状態はデータフラッシュ内のフラグで管理しています。

表 6-2 デモプロジェクトの状態

状態	動作内容
STATE_INJECT_KEY	データフラッシュに Wrapped Key の注入を行います。 デモプロジェクトでは、データフラッシュ内の鍵を格納するエリアをメイン面とミラー面に分けて管理しています。2 面に分けて鍵データを管理することで、鍵の書き込み時に電源遮断などが発生し失敗した場合でも鍵データを復旧することが可能です。 鍵の注入が完了したら、STATE_EXEC_IMAGE に遷移します。 鍵の注入は、デモプロジェクトダウンロード後初回起動時のみ実施し、2 回目以降の起動時は、STATE_EXEC_IMAGE 状態からスタートします。
STATE_EXEC_IMAGE	データフラッシュの鍵データが壊れていないかをハッシュ値を使用して確認します。鍵データが壊れていないことが確認できたら、コマンドを受け付けます。

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 デモプロジェクトのコマンド一覧

コマンド	動作
display	Wrapped Key を表示します。
encdemo [Arg1]	Arg1 の値を AES 128bit ECB モードで暗号化します。
function	下記の処理を実行し、API 関数の動作確認を行います。 ・AES128/256 ECB,CBC,CTR,CCM,GCM 暗号化/復号 ・AES128/256 CMAC 生成/検証 ・SHA-224/256 HMAC 生成/検証 ・ECDSA P256 署名生成/検証
random	疑似乱数を生成します。

6.1.2.1 Wrapped Key の特徴とデモプロジェクトでの確認方法

RSIP PM ドライバはユーザ鍵を平文のままでは使用しません。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、平文のユーザ鍵が流出しないことを意味します。デモプロジェクトでは、データフラッシュに注入されたユーザ鍵の Wrapped Key を使用しています。ユーザ鍵の Wrapped Key を置いているデータフラッシュ領域(0x00100000)をリードしても、ユーザ鍵の平文が書かれていないことを確認できます。

Wrapped Key は HUK に紐付いて生成されるため、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。これはユーザ鍵のデッドコピーを防ぐことを意味します。他のデバイスで生成した Wrapped Key をデモプロジェクトに組み込んで使用しようとしても、RSIP PM ドライバがエラーになることを確認できます。

また Wrapped Key には乱数が含まれているため、同じデバイスで同じユーザ鍵の Wrapped Key を生成した場合でも、違う値が生成されます。このことにより、Wrapped Key からユーザ鍵を推測することを困難にします。これらの Wrapped Key の特徴は、デモプロジェクトで確認することができます。デモプロジェクトをダウンロードしなおして、display コマンドを実行するたびに、Wrapped Key の値が変わることを確認できます。

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の注入が行われた場合の表示を図 6-3に、その後MCUに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 6-4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。

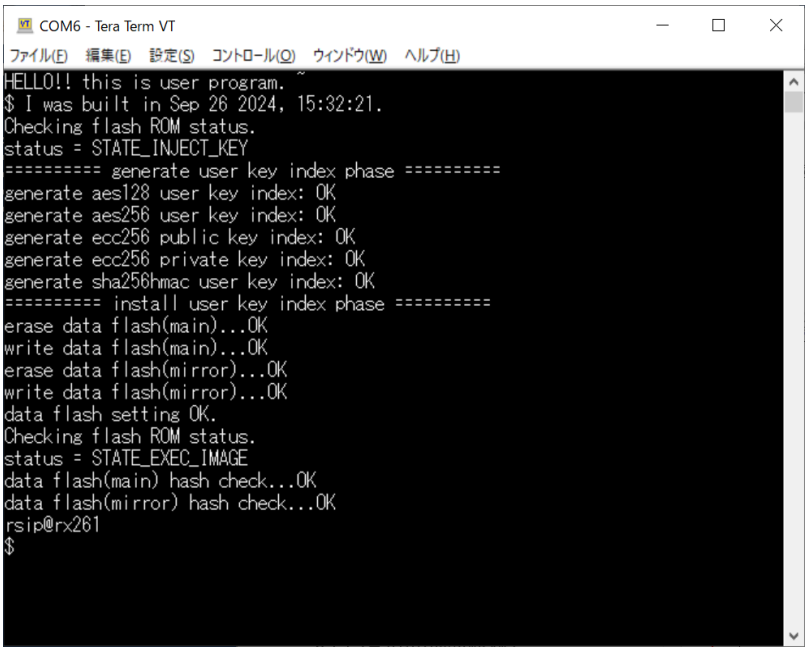

図 6-3 Tera Term の表示（初回起動時）

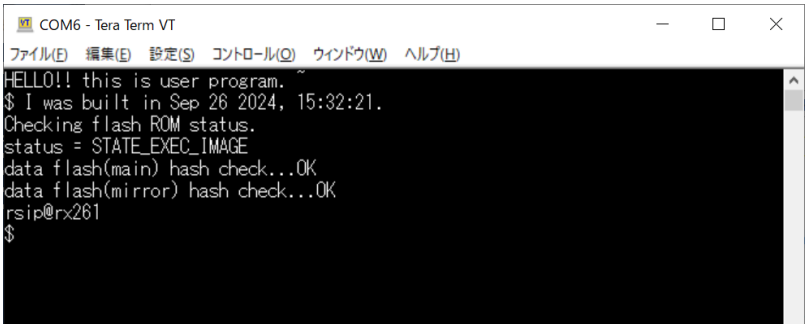

図 6-4 Tera Term の表示（2 回目以降の起動時）

コマンド使用例として、encdemo コマンドの使用例を示します。

encdemo コマンドでは、注入済みの AES128bit の鍵を使って、コマンドの引数で入力されたデータを AES-ECB で暗号化します。

サンプルではあらかじめ AES128bit の鍵として、”11111111222222223333333344444444”という鍵データが注入されています。

データ” aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa”を encdemo コマンドで入力した場合の実行例を以下の図 6-5に示します。

図 6-5 Tera Term の表示（encdemo コマンド実行時）

6.2　セキュアブートとファームウェアアップデート

セキュアブートとファームウェアアップデートのデモプロジェクトは、セキュアブートとファームウェアアップデートの実現方法を示しています。このデモプロジェクトでは、セキュアブートプログラムとファームウェアアップデートプログラムをデバイスにセットアップする手段として、以下の二つの方法を紹介します。

1. 最初にセキュアブートプログラムだけある状態にしておいて、セキュアブートプログラムの機能を使って後からファームウェアアップデートプログラムをフラッシュメモリに書き込む方法（二段階セットアップ）
2. セキュアブートプログラムとファームウェアアップデートプログラムを併せた Motorola S format ファイルを作成して、一度にフラッシュメモリに書き込む方法（一括セットアップ）

セキュアブートとファームウェアアップデートのデモプロジェクトでは、4.2.9 セキュアブート/ファームウェアアップデートの API を使用して、3.13.3 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ファームウェアアップデートプログラムをデバイス内で復号します。暗号化されたファームウェアアップデートプログラムを検証してからフラッシュメモリに書き込み起動することで、セキュアにファームウェアアップデートを実行することができます。また、実行前にファームウェアアップデートプログラムを検証することで、セキュアブートを実行することができます。

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、ファームウェア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ではセキュアブート動作に加えて、ファームウェアアップデートプログラムをデバイスに書き込むためのファームウェアアップデート動作も実行します。

セキュアブートプロジェクト　　：rx261_ek_rsip_secure_boot

ファームウェアアップデートプロジェクト：rx261_ek_rsip_user_program

本デモプロジェクトのファームウェアアップデート機能は、ファームウェアアップデート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インタフェースとして、UART もしくは USB を用意しています。また、Security Key Management Tool は、セキュアブートプログラム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(*1)を生成することができます。（Factory Programming 機能）

[*1] 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 RSIP 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。

本デモプロジェクトでは、実行結果の出力ならびに UART を使ったファームウェアアップデート動作で UART を使用します。ターミナルソフトをインストールした PC とデモプロジェクト実行ボードを接続してください。以降の説明では、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 で動作します。

本デモプロジェクトで使用している FIT モジュールのバージョンを以下に示します。セキュアブート処理で使用する API および内部関数は SECURE_BOOT セクションに配置しています。

表 6-4 デモプロジェクトで使用する FIT モジュール

FIT モジュール	バージョン
r_bsp【注】	7.53
r_byteq	2.11
r_cmt_rx	5.71
r_flash_rx	5.22
r_fwup	2.04
r_sci_rx	5.41
r_sha_rx	1.06
r_sys_time_rx	1.02
r_tfat_driver_rx	2.61
r_tfat_rx	4.14
r_usb_basic_mini	1.31
r_usb_hmsc_mini	1.31

【注】ブートローダプロジェクトにおいて USB メモリを使用したファームウェアアップデートを行うため、BSP FIT モジュールに変更を加えています。

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は、表 6-5に示すボード上で動作します。使用するインタフェースに合わせてボードの設定を行ってください。

表 6-5 デモプロジェクトで使用するボード、及びその設定

ボード	UART 使用時のジャンパ設定	USB 使用時のジャンパ設定
EK RX261	-	J17：開放、J18：開放、J19：1-2 短絡

ボードと PC の接続は以下の通りです。

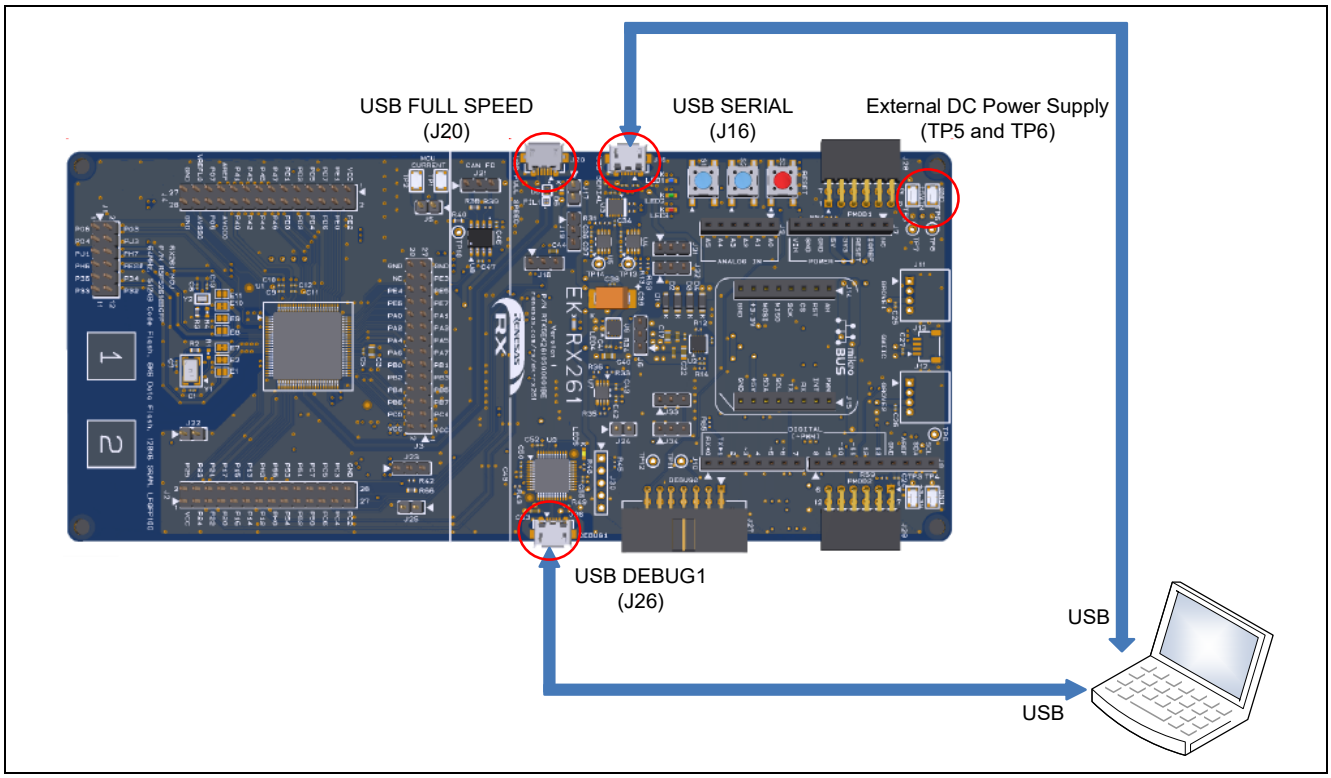

図 6-6 UART を使用したファームウェアアップデートを行うときの EK-RX261 と PC の接続図

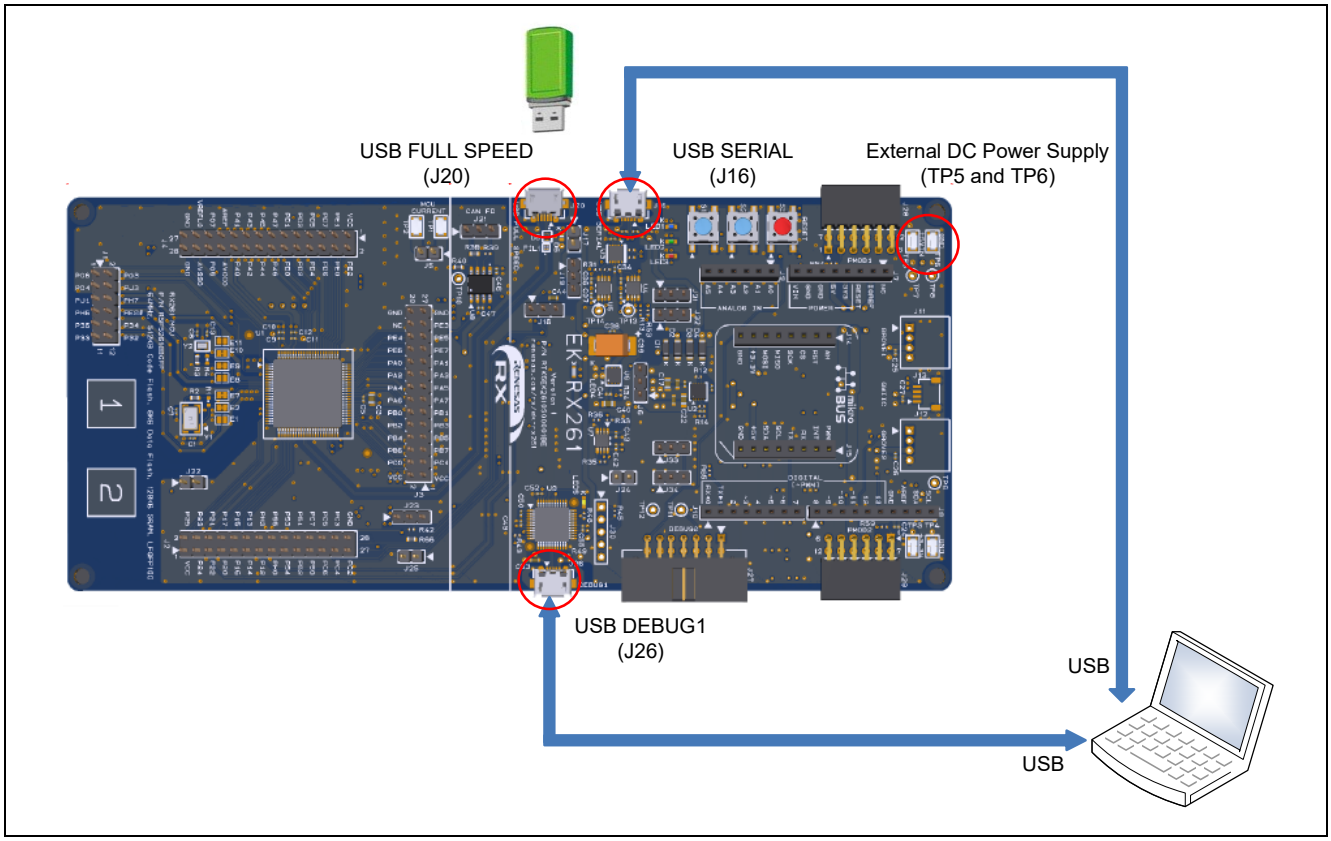

図 6-7 USB を使用したファームウェアアップデートを行うときの EK-RX261 と PC の接続図

Tera Term のシリアルポート設定と端末の設定は以下の通りです。

表 6-6 Tera Term の設定

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
フロー制御	RTS/CTS
改行コード（送信）	CR
ローカルエコー	OFF

6.2.2 デモプロジェクトの概要

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ディレクトリ構成を表 6-7に示します。

表 6-7 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ディレクトリ構成

ディレクトリ名	内容
rx261_ek_rsip_secure_update	セキュアブート/セキュアアップデートの実装例
rx261_ek_rsip_secure_boot	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
rx261_ek_rsip_user_program	ファームウェアアップデートプログラム
src	ファームウェアアップデートプログラムのソースコード

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では、フラッシュメモリ領域を 2 面に分けて、メイン面、バッファ面として扱います。

メイン面：実行対象のプログラムを格納するエリア

バッファ面：更新対象のプログラムを格納するエリア

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の動作フローを図 6-8に示します。

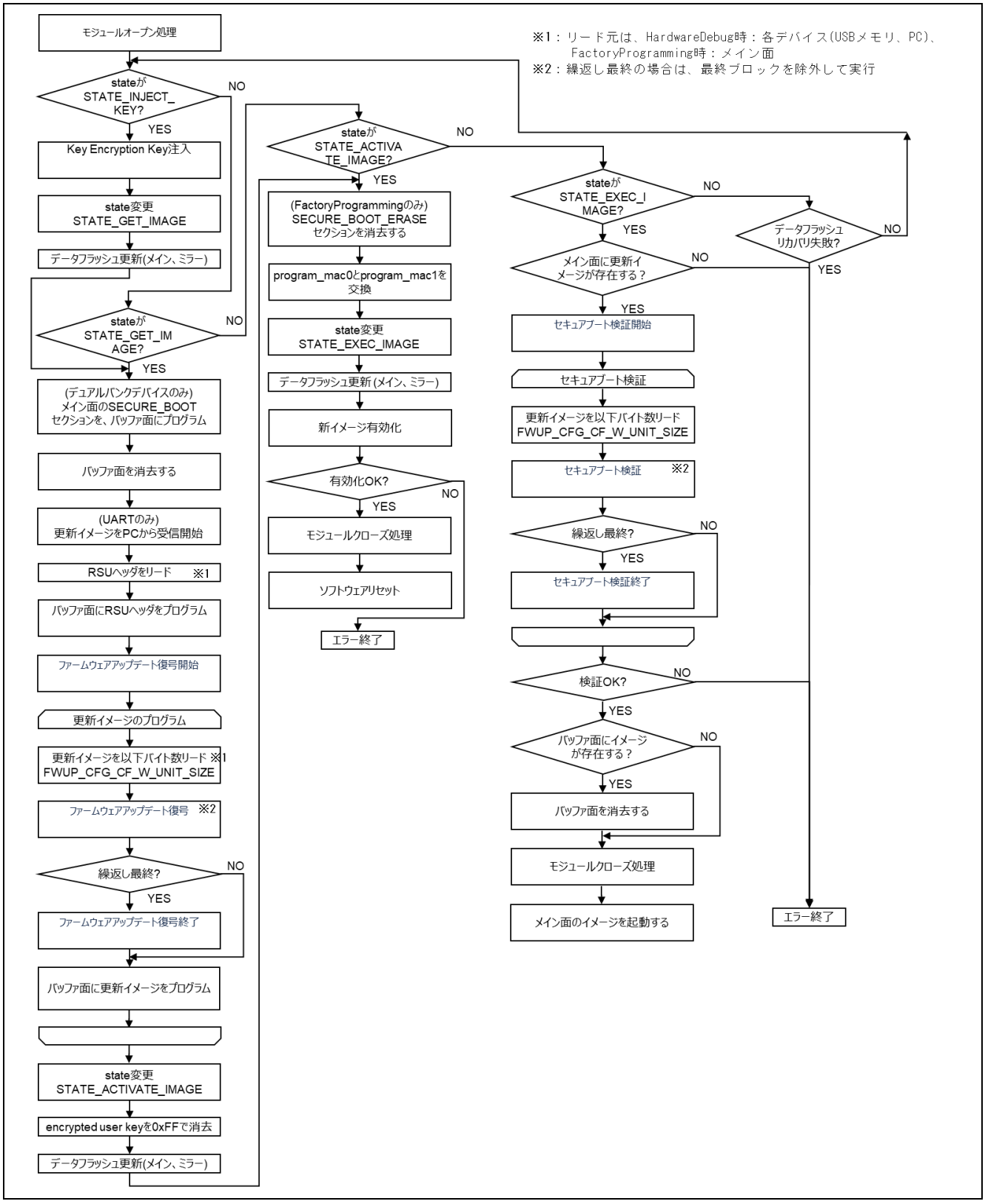

図 6-8 セキュアブートプロジェクトの動作フロー

セキュアブートプロジェクトは状態遷移で動作を管理しています。状態はデータフラッシュ内のフラグで管理しています。各状態と各状態と動作内容を表 6-8に示します。

表 6-8 セキュアブートプロジェクトの状態

状態	動作内容
STATE_INJECT_KEY	データフラッシュに Wrapped Key を格納します。この時、データフラッシュのメイン面とミラー面に Wrapped Key を格納することで、更新時の電源遮断などによる書き込み失敗時の Wrapped Key の消失を防いでいます。鍵の注入処理が完了したら、STATE_GET_IMAGE に遷移します。
STATE_GET_IMAGE	暗号化されたファームウェアアップデートプログラムを取得して復号します。最初にバッファ面の消去を行います。USB メモリもしくは UART 経由で暗号化されたファームウェアアップデートプログラムを取得しながら復号し、バッファ面に書き込みます。Encrypted Key を消去後、STATE_ACTIVATE_IMAGE に遷移します。
STATE_ACTIVATE_IMAGE	更新イメージを有効化します。一括セットアップの場合、先に鍵注入のための処理を消去します。メイン面にファームウェアアップデートプログラムをコピー後、STATE_EXEC_IMAGE に遷移し、ソフトウェアリセットを実行します。
STATE_EXEC_IMAGE	メイン面のプログラムの MAC 検証を行います。バッファ面にイメージが存在する場合、バッファ面のプログラムを消去後、メイン面のプログラムを起動します。

6.2.2.2 ファームウェアアップデートプロジェクト

ファームウェアアップデートの動作フローを図 6-9に示します。

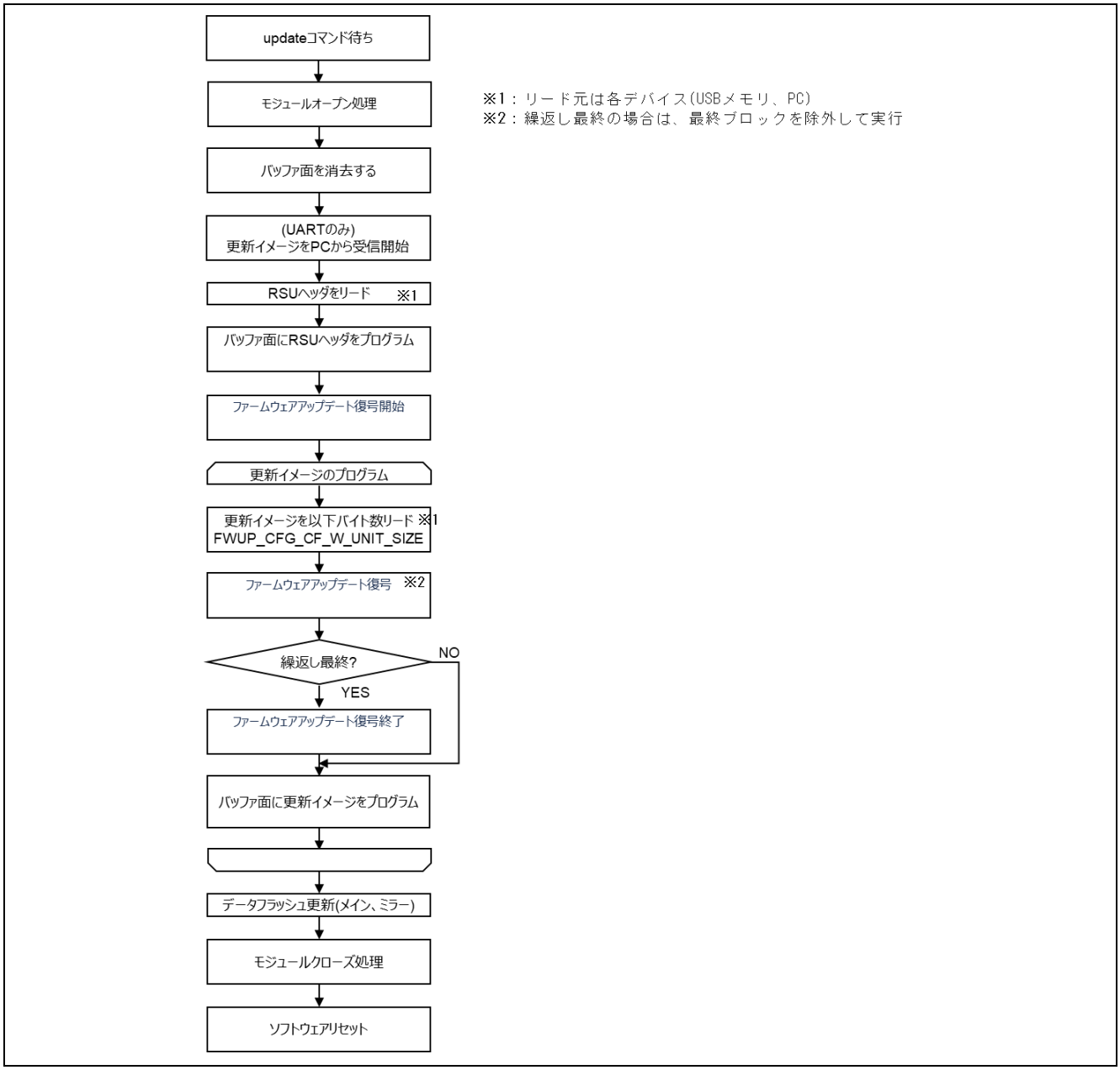

図 6-9 ファームウェアアップデートプログラムの動作フロー

ファームウェアアップデートプロジェクトでは以下のコマンドを実装しています。

表 6-9 ファームウェアアップデートプロジェクトのコマンド一覧

コマンド	動作
update	ファームウェアアップデートを実行します。 最初にバッファ面の消去を行います。USB メモリもしくは UART 経由で暗号化されたユーザプログラムを取得しながら復号し、バッファ面に書き込みます。 ファームウェアアップデートが正常に終了したら、状態を STATE_ACTIVATE_IMAGE に遷移後、ソフトウェアリセットを実行します。

6.2.3　デモプロジェクトの実行例

6.2.3.1　二段階セットアップの実行例

(1) ファームウェア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、ファームウェアアップデート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 = “22222222222222222222222222222222111111111111111111111111111111111”

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-RSIP-E11A" /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 rx261_SecureUpdate.skmt をロードして使用可能です。

rx261_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。

rx261_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。

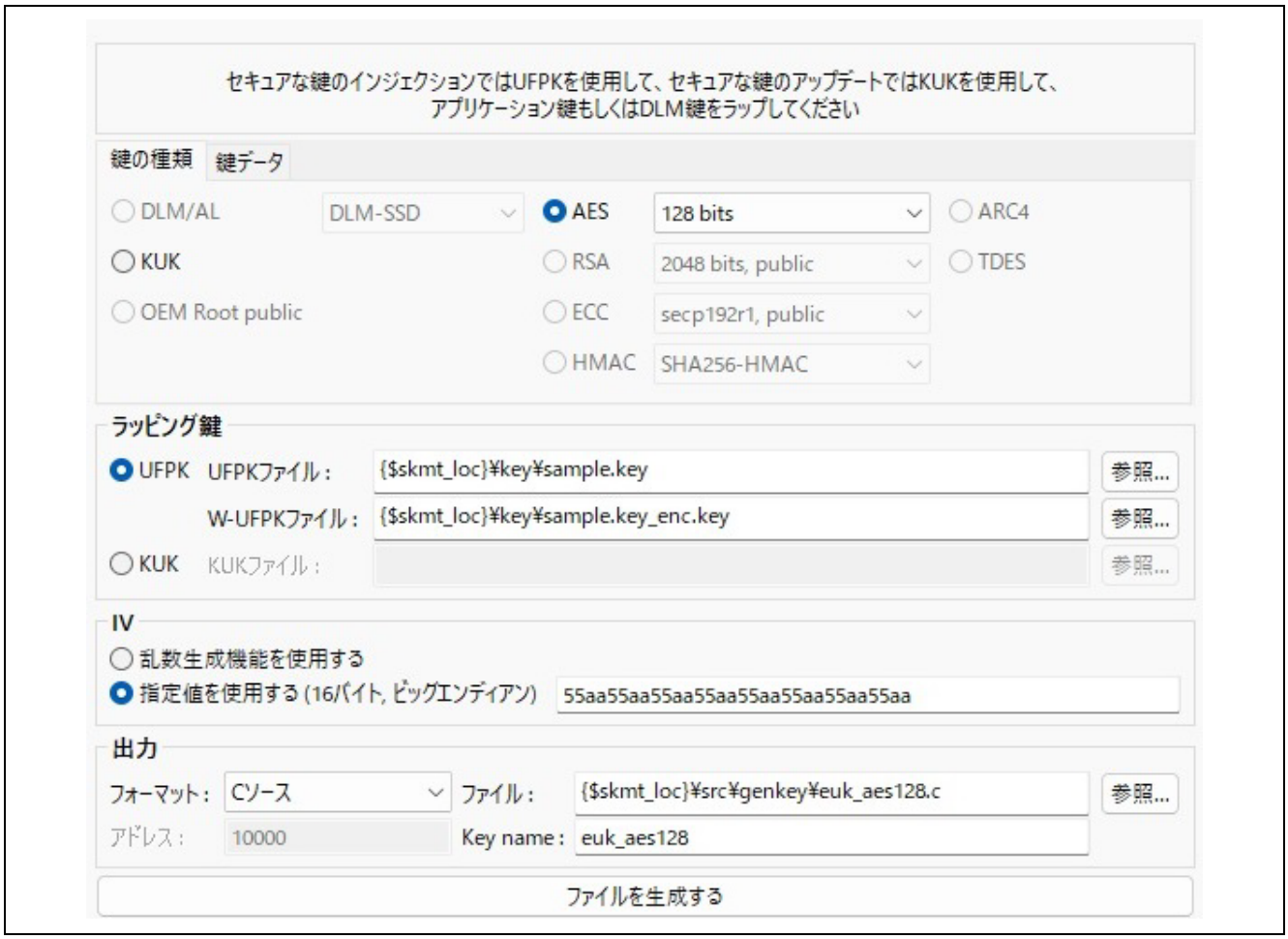

図 6-10 鍵データファイルの生成（[鍵のラッピング]タブ）

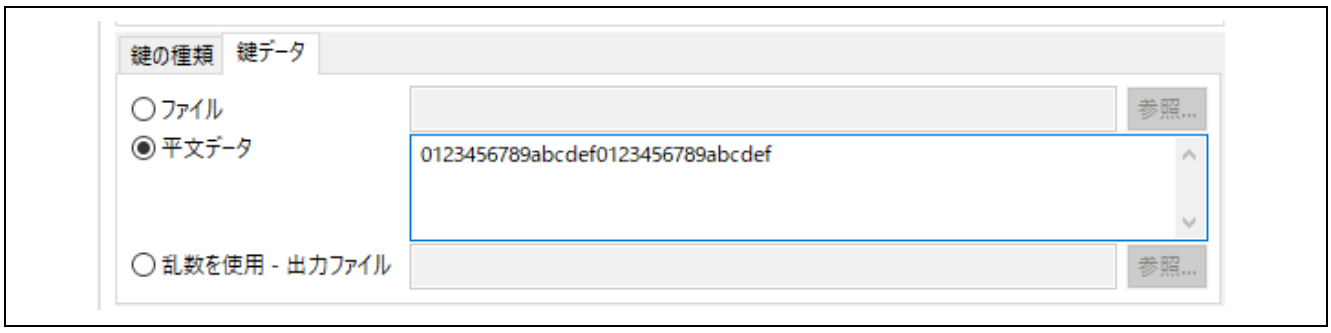

図 6-11 鍵データファイルの生成（[鍵のラッピング]タブ-[鍵データ]タブ）

(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(rx261_ek_rsip_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 rx261_ek_rsip_secure_boot プロジェクトの src フォルダの下に置きます。

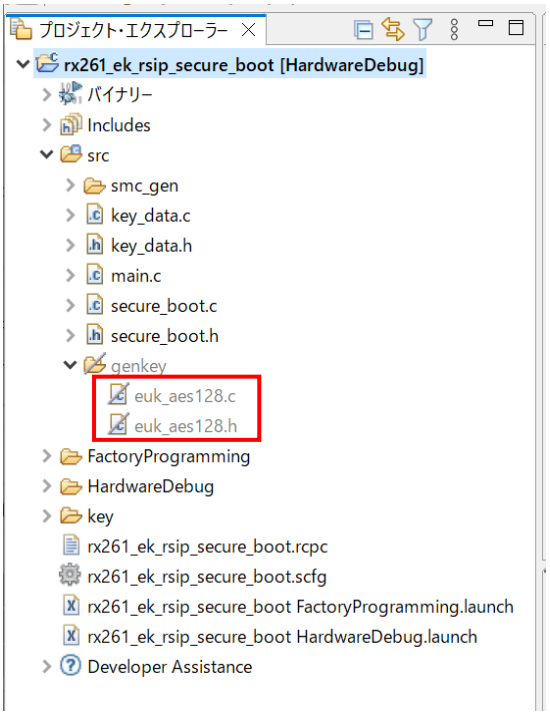

図 6-12 e²studio Project Explorer

セキュアブートプロジェクトをビルドする前に、プロジェクトで使用するライブラリをビルドします。プルダウンから「Library Key Injection」と「Library USB Memory」のそれぞれを選択してビルドしてください。

図 6-13 ライブラリのビルド

また、USB メモリを使用したファームウェアアップデートを実行する場合は、プリプロセッサ・マクロの定義において

ENABLE_USB=1

と設定してください。設定完了後 e²studio でビルドします。

(3) ファームウェアアップデートプロジェクトのビルド

ファームウェアアップデートプロジェクト(rx261_ek_tsip_user_program)を、e²studio のワークスペースにインポート後、USB メモリを使用したファームウェアアップデートを実行する場合は(2) 同様にプリプロセッサ・マクロの定義において

ENABLE_USB=1

と設定してください。設定完了後 e²studio でビルドします。

(4) ファームウェア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 "2" /prg
"{$skmt_loc}¥..¥rx261_ek_rsip_user_program¥Release
¥rx261_ek_rsip_user_program.mot"
/enckey "0123456789abcdef0123456789abcdef" /session_key
“fedcba9876543210fedcba98765432100123456789abcdef0123456789abcdef”
/iv_fw “55aa55aa55aa55aa55aa55aa55aa55aa”
/startaddr "FFEB8300" /endaddr "FFFEFFFF" /filetype "bin" /flash_wsize 128
/output ""{$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 6-10　MCU の指定アドレス

MCU	パラメータ		アドレス
	GUI	CLI	
RX261	開始アドレス	/startaddr	FFFB8300
	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
　ファームウェアアップデートプロジェクトから出力される Motorola S format ファイル
　(rx261_ek_rsip_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 6-10 MCUの指定アドレス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
　RSU ヘッダ Ver　：　2

-「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 rx261_SecureUpdate.skmt をロードして使用可能です。
rx261_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。
rx261_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。

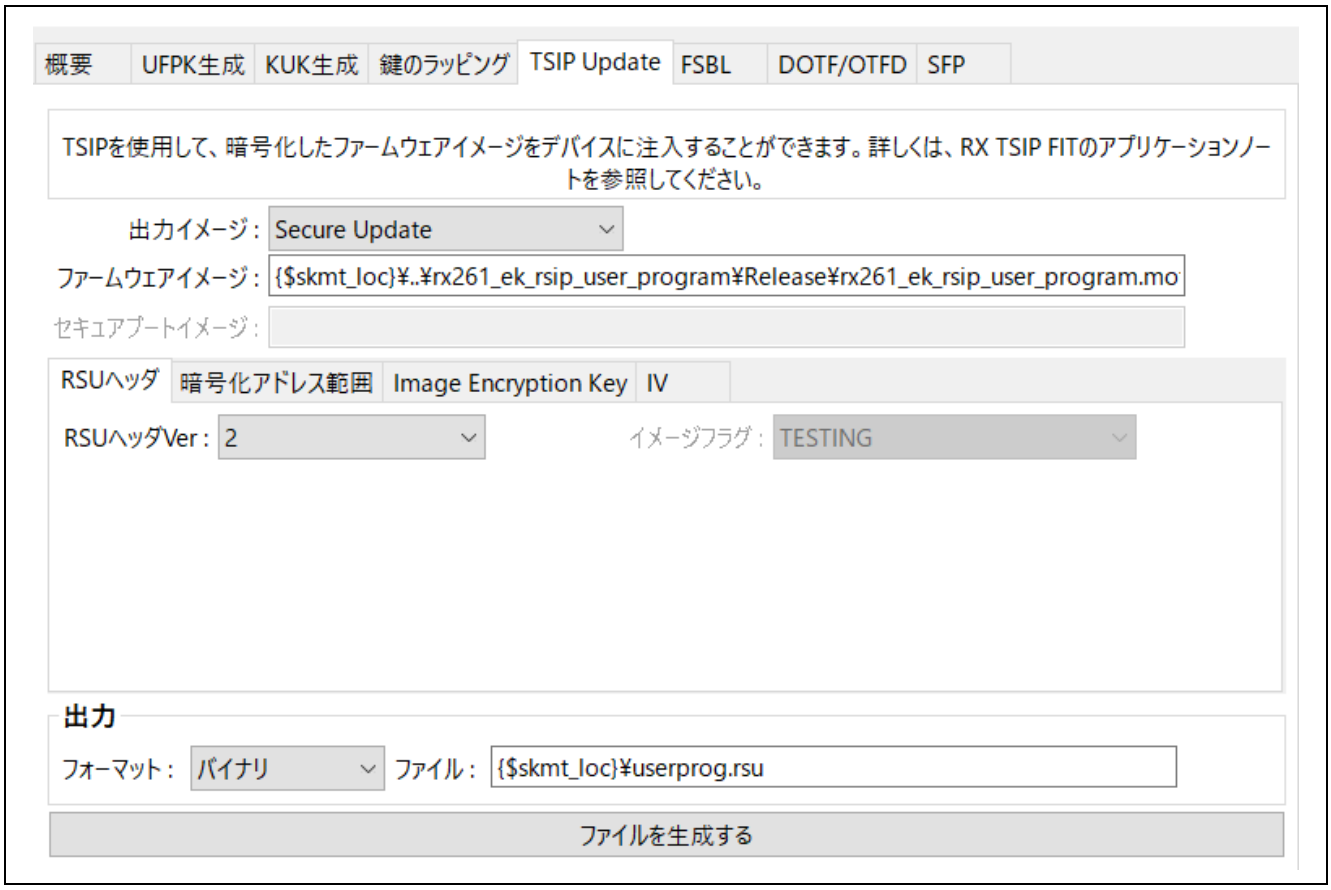

図 6-14 暗号化ファイルの生成（[TSIP UPDATE]タブ-[RSU ヘッダ]タブ）

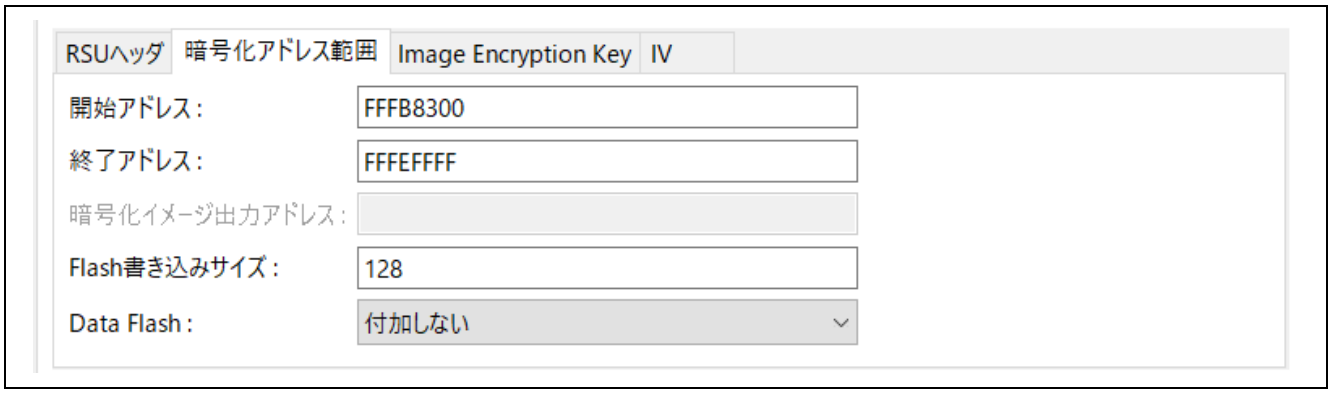

図 6-15 暗号化ファイルの生成（[TSIP UPDATE]タブ-[暗号化アドレス範囲]タブ）

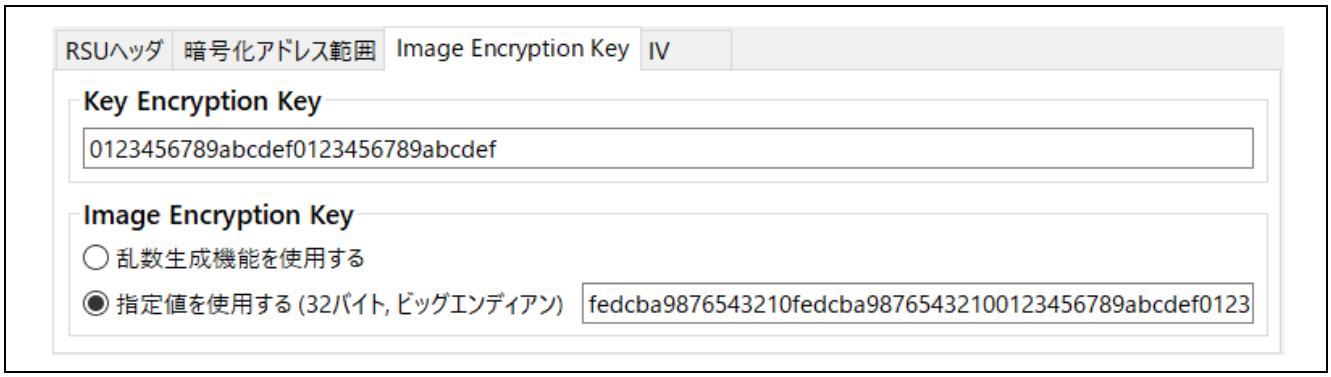

図 6-16 暗号化ファイルの生成（[TSIP UPDATE]タブ-[Image Encryption Key]タブ）

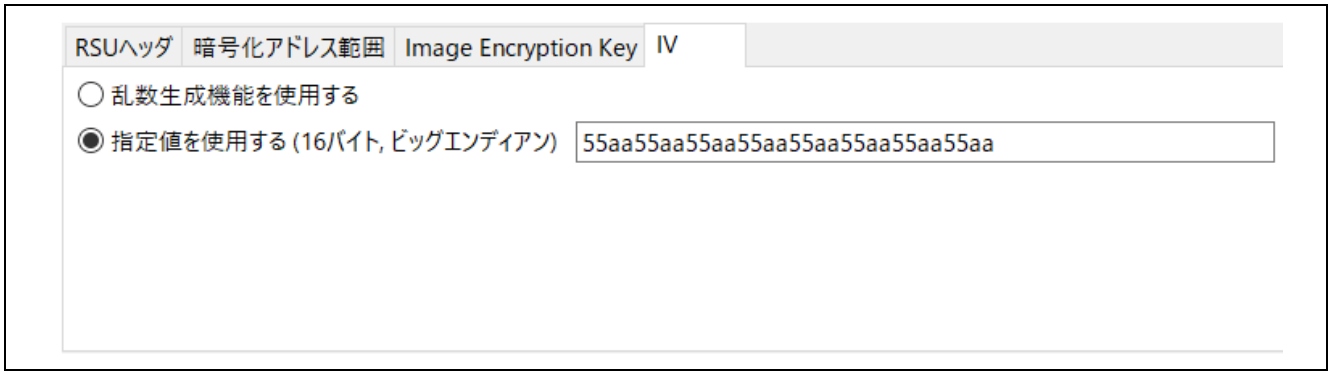

図 6-17 暗号化ファイルの生成（[TSIP UPDATE]タブ-[IV]タブ）

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 Tera Termの設定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 rx261_ek_rsip_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。

使用するボードのフラッシュメモリが全消去されていない場合、正常に動かないため、RFP でフラッシュメモリを、「チップ消去」で、全消去してください。

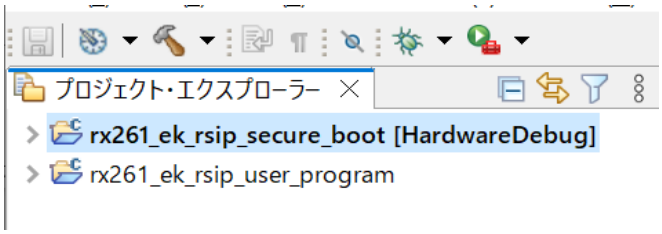

図 6-18 セキュアブートプロジェクトの実行

(7) ファームウェアアップデートプログラムの書き込み

セキュアブートプロジェクトを実行すると Tera Term に以下のようなログが出力され、更新イメージの受信待ち状態になります。

```
HELLO!! this is boot program. ~
$ I was built in Sep 27 2024, 09:29:33.
Checking flash ROM status.
status = STATE_INJECT_KEY
========== generate user key index phase ==========
generate aes128 user program mac key index: OK
========== install user key index phase ==========
erase data flash(main)...OK
write data flash(main)...OK
erase data flash(mirror)...OK
write data flash(mirror)...OK
data flash setting OK.
Checking flash ROM status.
status = STATE_GET_IMAGE

==== Image updater [with buffer] ====
Erase buffer area...OK
send image(*.rsu) via UART.
```

図 6-19 セキュアブート 更新イメージの受信待ち状態のログ

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ウェア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。

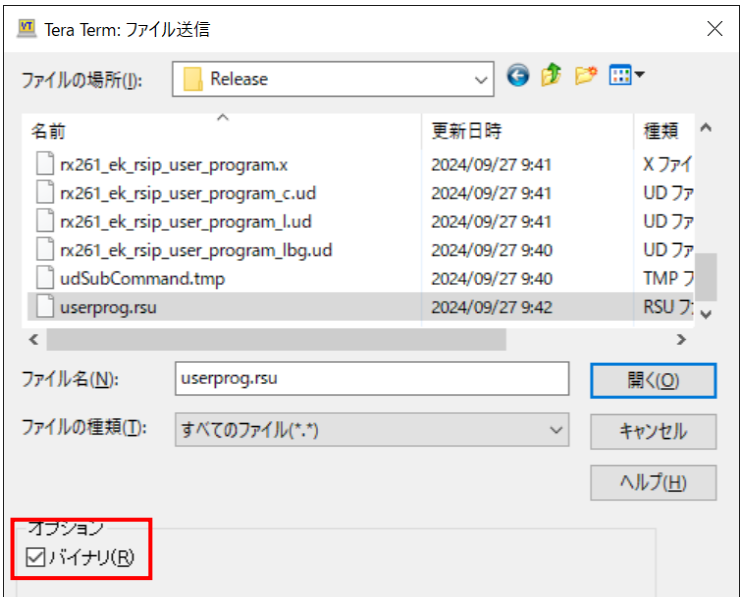

図 6-20 Tera Term ファイル送信ダイアログ

正常にファームウェアアップデート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。

```
erase data flash(main)...OK
write data flash(main)...OK
erase data flash(mirror)...OK
write data flash(mirror)...OK
data flash setting OK.
Checking flash ROM status.
status = STATE_ACTIVATE_IMAGE
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.
activating image ... OK
software reset...
HELLO!! this is boot program. ~
$ I was built in Sep 27 2024, 09:29:33.
Checking flash ROM status.
status = STATE_EXEC_IMAGE

==== BootLoader [with buffer] ====
secure boot sequence: success.
execute image ...
HELLO!! this is user program. ~
$ I was built in Sep 27 2024, 09:40:38.
Version ver 1.00.
rsip@rx261
$
```

図 6-21 セキュアブート ファームウェアアップデートプログラムを書き込み時のログ

(8) ファームウェアアップデート動作

続いて、内容を一部変更したファームウェアアップデートプロジェクトをビルドし、書き込み済みのファームウェアアップデートプログラムをその変更したプログラムに更新します。ファームウェアアップデートプロジェクト main.c にある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 Motorola S format 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。

```
/* Command prompt related */
#define PROMPT                          ("rsip@rx261\r\n$ ")
#define VERSION                         ("ver 1.00")
#define HELLO_MESSAGE                   ("HELLO!! this is user program. ~\r\n$ ")
```

図 6-22 main.c 変更箇所（rx261_ek_rsip_user_program）

update コマンドを実行します。(7)の手順同様で、Ver 1.01 に更新したファームウェアアップデートプログラムが書き込まれます。ファームウェアアップデート後にシステムがリブートします。表示されるログから、図 6-21の太字の部分が更新されることを確認することができます。

6.2.3.2　一括セットアップの書き込み実行例

本章では、セキュアブートと暗号化されたファームウェアアップデート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を説明します。

(1) ファームウェアアップデート用暗号鍵ファイルの生成

6.2.3.1 二段階セットアップの実行例の(1)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(rx261_ek_rsip_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 rx261_ek_rsip_secure_boot プロジェクトの src フォルダの下に置きます。

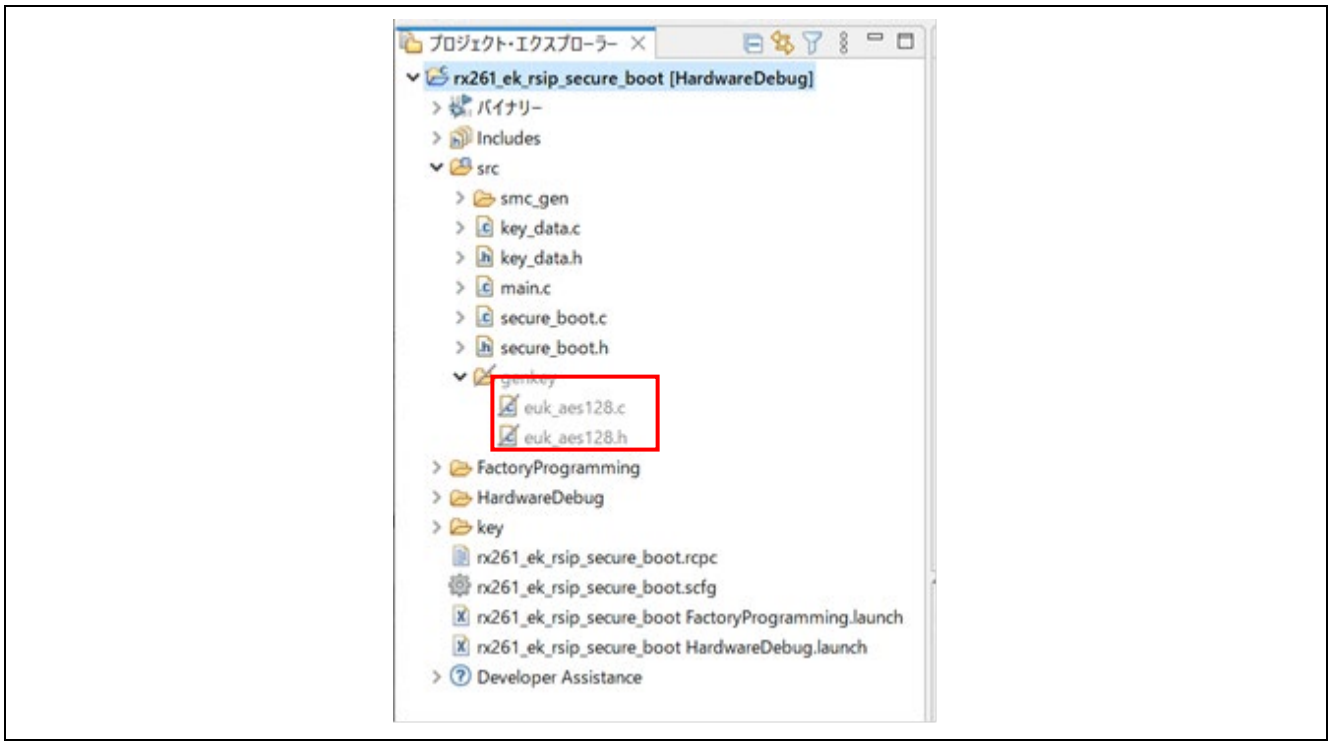

図 6-23 e²studio Project Explorer

「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ブ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。

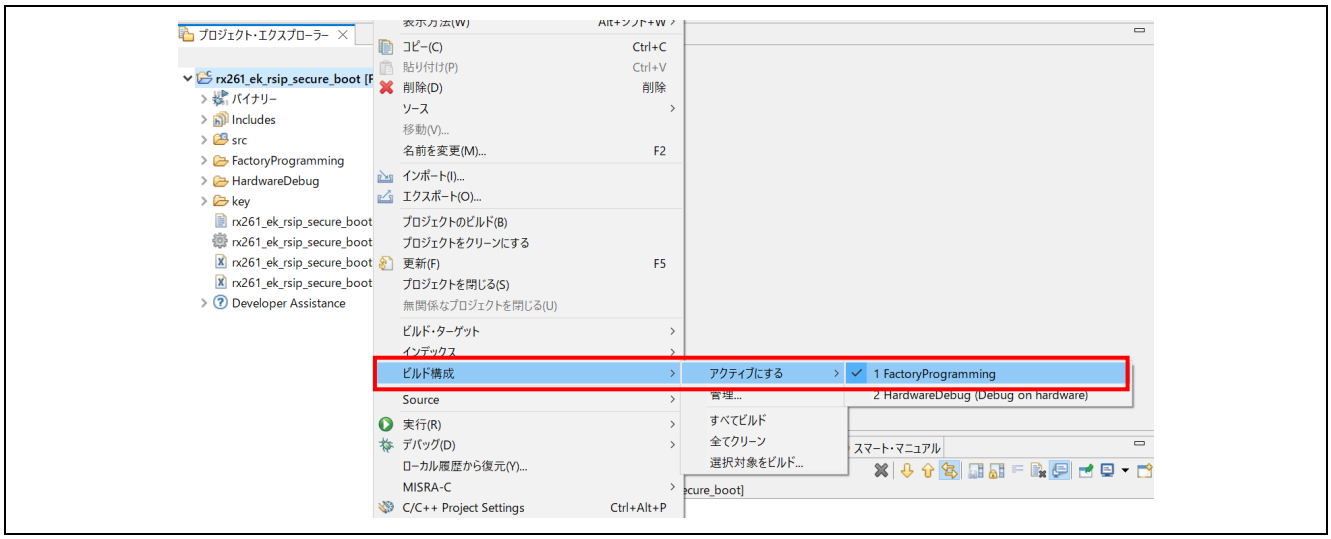

図 6-24 ビルド構成 Factory Programming への変更

「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) ファームウェアアップデートプロジェクトのビルド

6.2.3.1 二段階セットアップの実行例の(3)と同様の手順でビルドします。

(4) ファームウェア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 "2" /prg
"{$skmt_loc}¥..¥rx261_ek_rsip_user_program¥Release_rx261¥rx261_ek_rsip_user_program.mot"
/prg_sb "{$skmt_loc}¥FactoryProgramming¥rx261_ek_rsip_secure_boot.mot "
/enckey "0123456789abcdef0123456789abcdef" /session_key
“fedcba9876543210fedcba98765432100123456789abcdef0123456789abcdef”
/iv_fw “55aa55aa55aa55aa55aa55aa55aa55aa”
/startaddr "FFFB8300" /endaddr "FFFEFFFF" /destaddr "FFFB8300"
/filetype "mot" /flash_wsize 128 /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 6-11　MCU の指定アドレス

MCU	パラメータ		アドレス
	GUI	CLI	
RX261	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

・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
ファームウェアアップデートプロジェクトから出力される Motorola S format ファイル
(rx261_ek_rsip_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 6-11 MCUの指定アドレス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
RSU ヘッダ Ver ： 2

-「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 rx261_SecureUpdate.skmt をロードして使用可能です。
rx261_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。
rx261_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。

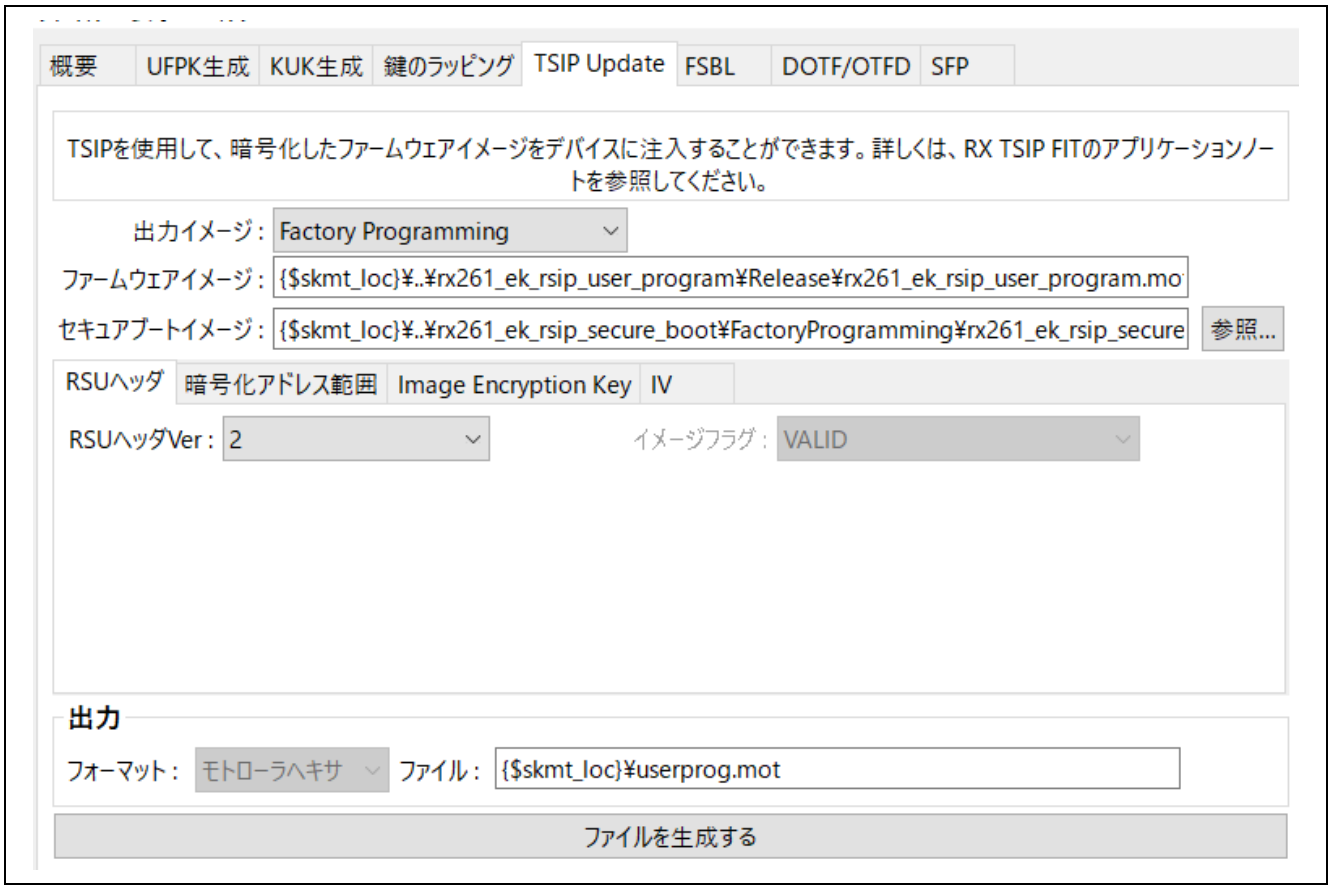

図 6-25 暗号化ファイルの生成（[TSIP UPDATE]タブ-[RSU ヘッダ]タブ）

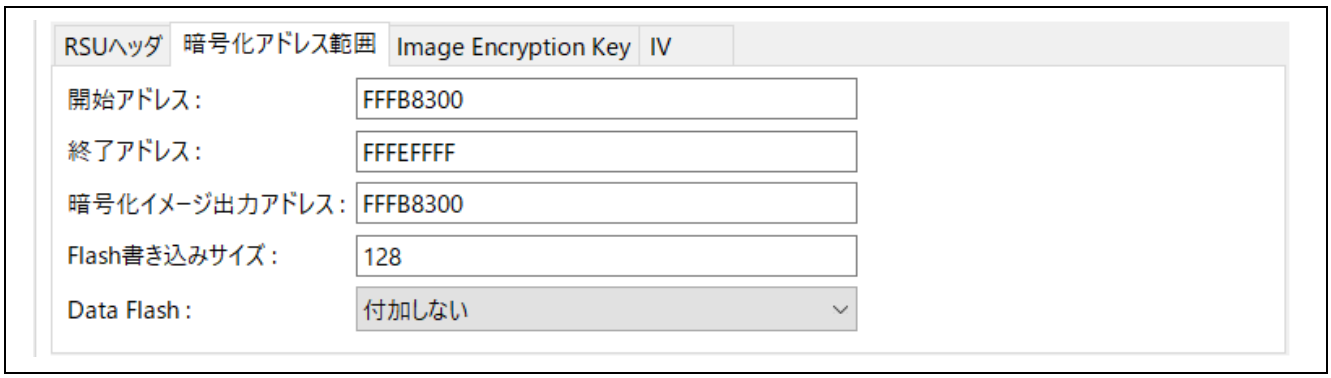

図 6-26 暗号化ファイルの生成（[TSIP UPDATE]タブ-[暗号化アドレス範囲]タブ）

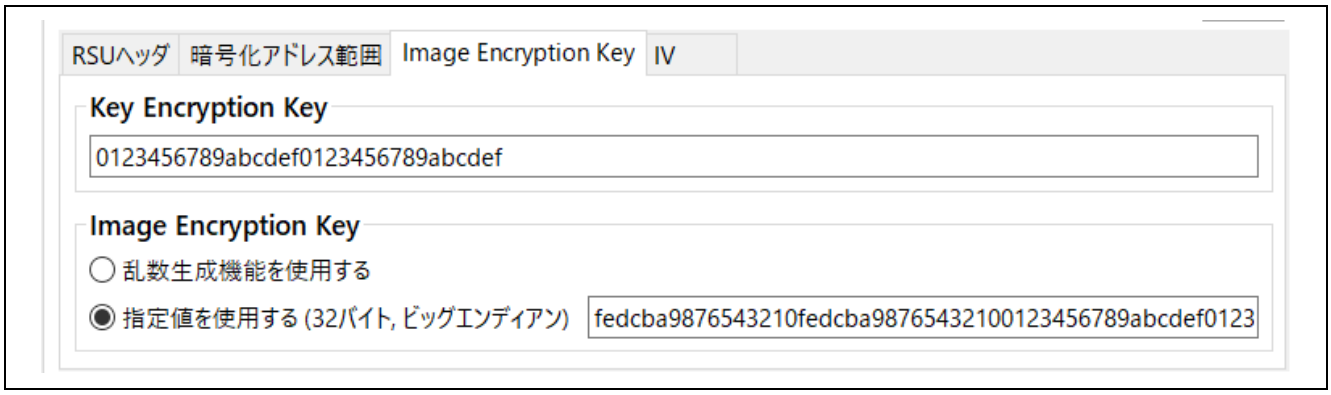

図 6-27 暗号化ファイルの生成（[TSIP UPDATE]タブ-[Image Encryption Key]タブ）

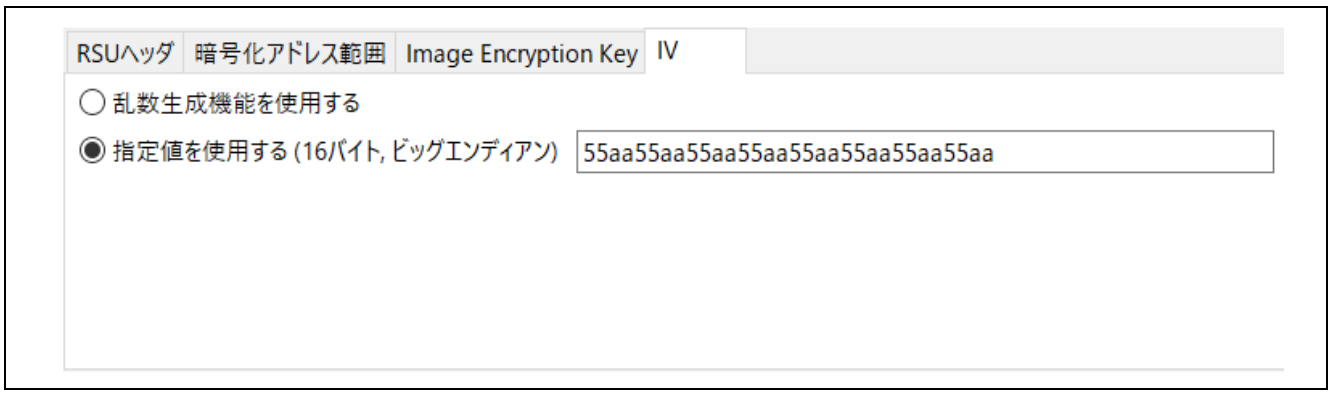

図 6-28 暗号化ファイルの生成（[TSIP UPDATE]タブ-[IV]タブ）

平文のセキュアブートプログラム＋暗号化したファームウェアアップデートプログラムの 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 を使用したフラッシュ書き込み

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。

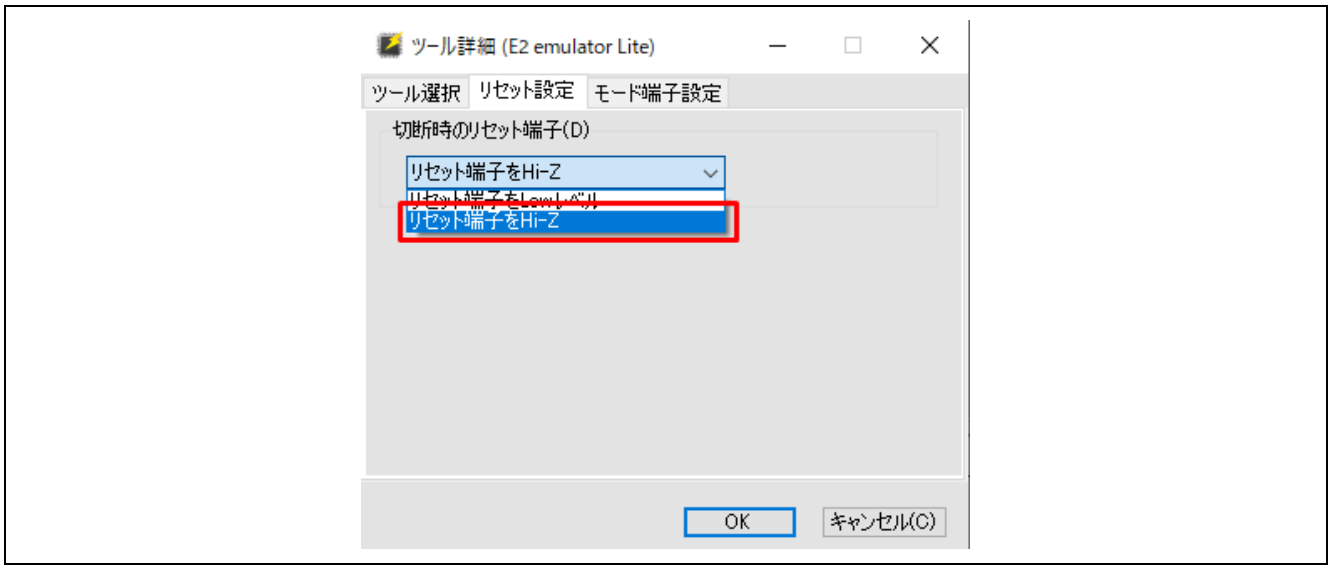

図 6-29 RFP リセット設定タブ

RFP を使用して、書き込みが正常に行われると、以下のようなログが出力されます。

```
HELLO!! this is boot program. ~
$ I was built in Oct  2 2024, 14:10:22.
Checking flash ROM status.
status = STATE_INJECT_KEY
========== generate user key index phase ==========
generate aes128 user program mac key index: OK
========== install user key index phase ==========
erase data flash(main)...OK
write data flash(main)...OK
erase data flash(mirror)...OK
write data flash(mirror)...OK
data flash setting OK.
Checking flash ROM status.
status = STATE_GET_IMAGE
```

図 6-30 RFP を使用した暗号ファームウェアアップデートプログラム書き込み時のログ

その後デバイスを起動すると、デバイスはセキュアブートプログラムを実行して、暗号化されたファームウェアアップデートプログラムを復号後、ファームウェアアップデートプログラムを実行します。なお、暗号化されたファームウェアアップデートプログラムの復号は、一括書き込み後、初回起動時のみ行われます。

正常に動作すると、以下のようなログが出力されます。

```
erase data flash(main)...OK
write data flash(main)...OK
erase data flash(mirror)...OK
write data flash(mirror)...OK
data flash setting OK.
Checking flash ROM status.
status = STATE_ACTIVATE_IMAGE
erase secure boot erase area...OK
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.
activating image ... OK
software reset...
HELLO!! this is boot program. ~
$ I was built in Oct  2 2024, 14:10:22.
Checking flash ROM status.
status = STATE_EXEC_IMAGE

==== BootLoader [with buffer] ====
secure boot sequence: success.
execute image ...
HELLO!! this is user program. ~
$ I was built in Oct  2 2024, 14:13:48.
Version ver 1.00.
rsip@rx261
$
```

図 6-31 一括書き込み後の初回起動時のログ

(7) ファームウェアアップデート動作

ファームウェアアップデート動作は、6.2.3.1 二段階セットアップの実行例の(8)をご参照ください。

6.2.4　ファームウェアアップデートプロジェクトのデバッグについて

ファームウェアアップデートプロジェクトは、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、ファームウェアアップデート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。ファームウェアアップデートプロジェクトをデバッグする場合は、デバッグ用の構成を HardwareDebug に切り替えて、デバッグを行ってください。

- 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 (Debug on hardware)]を選択すると、ビルドが開始します。

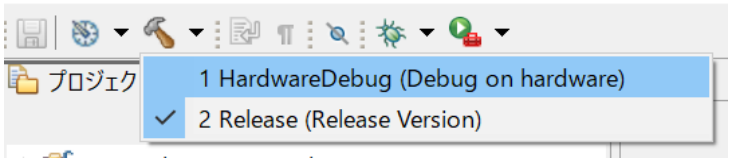

図 6-32 e²studio プロジェクトメニュー

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」とデバッグ用の構成「HardwareDebug」の違い
各ユーザプログラムプロジェクトの構成「Release」と「HardwareDebug」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 6-12　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	HardwareDebug
RESETVECT	0xFFFEFFFC	0xFFFFFFFC
EXCEPTVECT	0xFFFEFF80	0xFFFFFF80

6.2.5　セキュアブートプログラムからファームウェアアップデート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ファームウェアアップデート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ファームウェアアップデートプログラムに引き継がれることになります。そこで、本デモのセキュアブート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、RSIP、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。

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 2025-04 IAR Embedded Workbench for Renesas RX 5.10.1
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) V3.07.00 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 5.10.1 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.2.00
使用ボード	EK-RX261 (型名：RTK5EK2610SxxxxxBJ)

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 Encrypted Key の動的な生成方法

3.6.1 鍵の注入と更新の図 3-4 鍵注入および鍵更新時のユーザ鍵ラップ方式に示される、Encrypted Key を生成する処理は、RSIP ドライバを使用してデバイス上で動的に行うことも可能です。処理の概要を図 7-1 に示します。

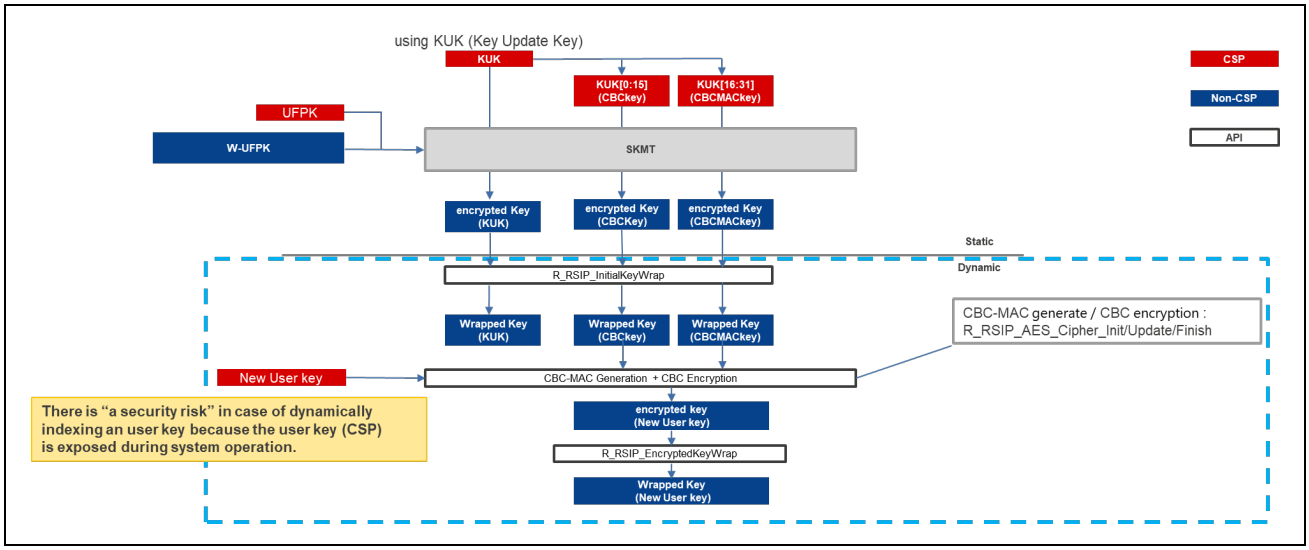

図 7-1 Encrypted Key の動的な生成方法

また、図 7-1内の水色の点線枠内の部分について具体的な処理を C 言語の形式で表すと、以下のようになります。

```
---------------------------------------------------------------------------------------------------------------------------------------
uint8_t new_user_key[LEN] = “New User key“; /* LEN is byte size of User Key */
uint8_t encrypted_key[LEN+16];
uint32_t MAC[4] = 0; /* Zero initialization */

r_instance_ctrl_t p_ctrl;
rsip_wrapped_key_t CBCMACkey;
rsip_wrapped_key_t CBCkey;
rsip_wrapped_key_t KUK;
rsip_wrapped_key_t wrapped_key;

uint32_t i, j;
uint8_t work_input[16];
uint8_t work_output[16];
uint8_t work_dummy[16];
uint32_t dummy;

/*Inject KUK as CBCMACkey and CBCkey*/
CBCkey.type = RSIP_KEY_TYPE_AES_128;
CBCkey.p_value = “Address to store wrapped key value”;
R_RSIP_InitialKeyWrap((rsip_ctrl_t *)p_ctrl, ‘W-UFPK’, ‘Initial vector’,
        ‘Encrypted key for 1st half of KUK as CBCkey’, &CBCkey);
CBCMACkey.type = RSIP_KEY_TYPE_AES_128;
CBCMACkey.p_value = “Address to store wrapped key value”;
R_RSIP_InitialKeyWrap((rsip_ctrl_t *)p_ctrl, ‘W-UFPK’, ‘Initial vector’,
        ‘Encrypted key for 2nd half of KUK as CBCMACkey’, &CBCMACkey);
```

```
/* Inject KUK */
KUK.type = RSIP_KEY_TYPE_KUK;
KUK.p_value = “Address to store wrapped key value”;
R_RSIP_InitialKeyWrap((rsip_ctrl_t *)p_ctrl, ‘W-UFPK’, ‘Initial vector’,
        ‘Encrypted key for KUK’, &KUK);

/* AES-128 CBC-MAC */
R_RSIP_AES_Cipher_Init((rsip_ctrl_t *)p_ctrl, RSIP_AES_CIPHER_MODE_ECB_ENC, &CBCMACkey,
NULL);
for (i = 0; i < LEN; i += 16)
{
        for (j=0; j<16; j++)
        {
                work_input[j] = new_user_key[i+j] ^ mac[j];
        }
        R_RSIP_AES_Cipher_Update((rsip_ctrl_t *)p_ctrl, work_input, work_output, 16);
        memcpy(mac, work_output, 16);
}
R_RSIP_AES_Cipher_Finish((rsip_ctrl_t *)p_ctrl);

/* AES-128 CBC-Encryption */
R_RSIP_AES_Cipher_tInit((rsip_ctrl_t *)p_ctrl, RSIP_AES_CIPHER_MODE_CBC_ENC, &CBCkey, iv);
R_RSIP_AES_Cipher_Update((rsip_ctrl_t *)p_ctrl, new_user_key, encrypted_key, len);
R_RSIP_AES_Cipher_Update((rsip_ctrl_t *)p_ctrl , mac, &encrypted_key[len], 16);
R_RSIP_AES_Cipher_Finish((rsip_ctrl_t *)p_ctrl);

/* Generate wrapped key */
wrapped_key.type = “Key type of the wrapped key”;
wrapped_key.p_value = “Address to store wrapped key value”;
R_TSIP_EncryptedKeyWrap((rsip_ctrl_t *)p_ctrl , &KUK, iv, encrypted_key, &wrapped_key);
```

ここで、‘W-UFPK’や‘Initial vector’のようにシングルクオーテーションで囲われている文言は SKMT で生成した鍵データファイルの値を代入、 “New User Key”のようにダブルクオーテーションで囲われている文言はユーザにおいて用意する値を代入することを意味しています。

ただし、本処理は、平文状態のユーザ鍵が非セキュア領域に露出するため、非推奨です。セキュリティ上のリスクを十分にご検討の上、当該リスクを許容される場合のみ、ご使用ください。また、ラップ処理が完了した後、即座に平文状態のユーザ鍵を廃棄するなど、適切なリスク対策の実施をご検討ください。

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	2024.10.15	—	初版発行
2.00	2025.07.31	—	・鍵管理の仕様変更 ・ECDH KDF 機能の追加 ・HMAC Suspend/Resume API の追加

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。