RX ファミリ

Renesas Secure IP コンパチビリティモード 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 Compatibility Mode ドライバ(RSIP CM ドライバ)と呼びます

RSIP CM 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/ja/software-tool/fit

RSIP CM ドライバは 表 1 にまとめた暗号機能に行うための API を持ちます。

動作確認デバイス

RX261 グループ

表 1 RX RSIP CM ドライバ暗号アルゴリズム

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

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 5
1.1 用語 5
1.2 概要 6
1.3 製品構成 8
1.4 関連文書 10
1.5 開発環境 10
1.6 コードサイズ 11
1.7 性能情報 12
1.7.1 RX261 12

2. API 情報 16
2.1 ハードウェアの要求 16
2.2 ソフトウェアの要求 16
2.3 サポートされているツールチェイン 16
2.4 ヘッダファイル 16
2.5 整数型 16
2.6 コンフィグレーション 17
2.6.1 Platform の設定 18
2.6.2 乱数生成の設定 18
2.6.3 Wrapped Key の設定 18
2.6.4 AES の設定 18
2.6.5 ECC の設定 19
2.6.6 SHA224/256 の設定 19
2.7 構造体 20
2.8 戻り値 20
2.9 FIT モジュールの追加方法 21

3. RSIP CM ドライバの使用方法 22
3.1 初期化 23
3.2 メモリ使用 23
3.3 制限事項 23
3.3.1 動作エンディアン 23
3.3.2 MBEDTLS_PLATFORM_SETBUF_MACRO の定義 23
3.4 シングルパート演算とマルチパート演算 23
3.5 鍵の管理 24
3.5.1 鍵の注入 25
3.5.1.1 Encrypted Key を使用した鍵の注入 25
3.5.1.2 平文鍵を使用した鍵の注入 27
3.5.2 鍵の生成 27
3.6 乱数生成 28
3.7 対称鍵暗号 29
3.8 非対称鍵暗号 29
3.9 HASH 関数 29

4. API 関数 30
4.1 API 一覧と詳細 30
4.1.1 バージョン情報 34
4.1.1.1 R_RSIP_CM_GetVersion 34
4.1.2 Key Injection 35
4.1.2.1 R_RSIP_CM_AESxxx_InitialKeyWrap 35
4.1.2.2 R_RSIP_CM_ECC_xxx_InitialPrivateKeyWrap 37

5. 鍵の注入 39
5.1 鍵の注入 39
5.2 Security Key Management Tool を使用した Encrypted Key の生成方法 40
5.2.1 鍵の注入手順 40
5.2.1.1 CLI 版を使用する場合の手順 40
5.2.1.2 GUI 版を使用する場合の手順 42

6. サンプルプログラム 45
6.1 鍵注入と暗号の使用方法 45
6.1.1 デモプロジェクトのセットアップ 45
6.1.2 デモプロジェクトの概要 47
6.1.2.1 鍵とデモプロジェクトでの確認方法 50
6.1.3 デモプロジェクトの実行例 51

7. 付録 53
7.1 動作確認環境 53
7.2 トラブルシューティング 54
7.3 ユーザ鍵フォーマット 55
7.3.1 AES 55
7.3.2 ECC 55
7.3.3 HMAC 56

8. 参考ドキュメント 57

ホームページとサポート窓口 58

改訂記録 59

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。
ユーザ鍵、 User Key	ユーザが使用する平文状態の暗号鍵。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 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 Renesas Key Wrap Service 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。
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。
Renesas Key Wrap Service	UFPK から W-UFPK の生成に使用するサイト。 https://dlm.renesas.com/keywrap/

1.2 概要

RX RSIP CM ドライバは、PSA Certified Crypto API(以下 PSA Crypto API)の実装に RSIP を使用して、ハードウェアアクセラレーションを行うドライバです。RX RSIP CM ドライバは従来のシステムやサードパーティ製ソフトウェアおよびソリューションとの容易な統合を可能にしながら、最適化されたパフォーマンスと無制限のセキュアキー保存を提供します。

RX RSIP CM ドライバは PSA Crypto API 1.0 仕様に準拠した mbedTLS バージョン 3.6.2 と組み合わせています。PSA Crypto API 仕様の詳細については、https://armmbed.github.io/mbed-crypto/psa/#application-programming-interface の Arm ドキュメントを参照してください。

- ハッシュ
  - SHA224 計算
  - SHA256 計算
- MAC
  - HMAC
  - CMAC
- AES
  - 鍵ビット - 128、256
  - 平文鍵(Plaintext key)生成
  - ラップ鍵(Wrapped key)生成
  - パディングなし、PKCS7 パディングありの暗号化と復号化。
  - CBC、CTR、CCM、GCM モード
- ECC
  - カーブ
    - secp256r1
    - secp256k1
    - brainpoolP256r1
  - ラップ鍵(Wrapped key)生成
  - 署名と検証
- 乱数生成

RX RSIP CM ドライバの特徴は以下になります。
• 平文鍵（Plaintext key）の使用が可能
これにより、レガシーシステムとの互換性が確保され、ソフトウェア開発が簡素化されます。また、既存のソフトウェアやインフラと統合する際にも必要となる場合があります。多くの既存のプログラミングシステムでは、平文鍵のインストールをサポートしており、アプリケーションコードを用いてチップ上に安全に保存します。
• ラップ鍵（Wrapped key）のサポート
セキュアなキー保存のためのラップ鍵を使用することが可能ですが、必須ではありません。また、ラップドキーの生成もサポートされています。

ただし、以下点で注意事項があります。
・ 単純電力解析（Simple Power Analysis）、差分電力解析（Differential Power Analysis）の保護機能が必要な場合は、RX RSIP Protected Mode ドライバのご使用をご検討ください。

・ 平文鍵を使用する場合、ユーザ鍵の漏洩リスクがある
平文鍵を使用すると、鍵が外部に露出する可能性があります。ユーザはこのリスクを十分に評価し、適切な対策を講じる必要があります。

・ セキュアな鍵の更新に制限がある
RSIP の外部で鍵が露出する可能性があるため、セキュアな鍵の更新が制限されます。

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
r01an7445jj0100-rx-rsip-cm-security.pdf	RSIP CM ドライバ アプリケーションノート(日本語)
r01an7445ej0100-rx-rsip-cm-security.pdf	RSIP CM 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	e2 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	e2 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 ユーザガイド(英語)
FITModules	FIT モジュールフォルダ
r_rsip_cm_rx_v1.00.zip	RSIP CM ドライバ FIT Module
r_rsip_cm_rx_v1.00.xml	RSIP CM ドライバ FIT Module e2 studio FIT プラグイン用 XML ファイル
r_rsip_cm_rx_v1.00_extend.mdf	RSIP CM 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_cm_sample	鍵注入と暗号の使用方法のデモプロジェクト

r_rsip_cm_rx_v.1.00.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 CM ドライバコンフィグファイルフォルダ
r_rsip_cm_rx_config.h	RSIP CM ドライバコンフィグファイル(デフォルト設定)
r_rsip_cm_rx	RSIP PM ドライバ FIT Module フォルダ
src	ソースコードフォルダ
mbedtls	OSS Mbed TLS フォルダ
rm_psa_crypto	PSA Crypto API 実装のハードウェアアクセラレーション
r_rsip_cm_rx	RSIP CM ドライバ FIT Module フォルダ
src	ソースコードフォルダ
adaptors	psa crypto アダプターフォルダ
r_sce_adapt.c	psa crypto アダプターソースコード
primitive	RSIP アクセス用ソースコードフォルダ
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
DomainParams.c	DomainParam データ情報ファイル
r_rsip_rx_finctionxxx.c	RSIP アクセス用ソースコード ファイル名の xx には数値が入ります。
r_rsip_rx_pxx.c	RSIP アクセス用ソースコード ファイル名の xxx には数値が入ります。
s_flash.c	鍵情報ファイル
private/inc	RSIP CM ドライバ内部関数のプログラムコード格納用フォルダ
hw_sce_rx_private.h	内部関数用ヘッダファイル
r_rsip_rx261_iodefine.h	RSIP アクセス用ヘッダファイル
public/inc	RSIP CM ドライバ API のプログラム格納用フォルダ
hw_sce_rx_public.h	外部関数用ヘッダファイル
commom	共通ソースコードフォルダ
hw_sce_common.h	共通ヘッダファイル
hw_sce_private.h	RSIP CM ドライバヘッダファイル
hw_sce_aes_private.h	RSIP CM ドライバ AES API 用ヘッダファイル
hw_sce_ecc_private.h	RSIP CM ドライバ ECC API 用ヘッダファイル
hw_sce_hash_private.h	RSIP CM ドライバ HASH API 用ヘッダファイル
hw_sce_trng_private.h	RSIP CM ドライバ乱数生成 API 用ヘッダファイル
hw_sce_rsa_private.h	RSIP CM ドライバ RSA API 用ヘッダファイル
r_rsip_cm_key_injection	RSIP CM ドライバ Key Injection フォルダ
r_rsip_cm_key_injection.h	Key Injection API ヘッダファイル
r_rsip_cm_key_injection.c	Key Injection API ソースコード
doc	ドキュメント格納フォルダ
ja	r_rsip_cm_key_injection.c
r01an7445jj0100-rx-rsip-security.pdf	RSIP CM 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)
r01an7445ej0100-rx-rsip-security.pdf	RSIP CM ドライバ アプリケーションノート(英語)
r_rsip_cm_rx_if.h	RSIP CM ドライバヘッダファイル
readme.txt	Readme

1.4 関連文書

表 1-4 に本書の関連文書を示します。

表 1-4 関連文書

ドキュメント名称	資料リンク
Mbed TLS 3.6.2	https://github.com/Mbed-TLS/mbedtls/releases/tag/mbedtls-3.6.2
Mbed TLS documentation hub	https://mbed-tls.readthedocs.io/en/latest/
PSA Certified Crypto API 1.0	https://arm-software.github.io/psa-api/crypto/1.0/
Mbed TLS (Renesas)	https://github.com/renesas/mbedtls

注: Mbed TLS 3.6.2 は PSA Certified Crypto API 1.0 仕様を使用しています。
RX RSIP CM ドライバでは、公式の Mbed TLS ソースコードに変更をして、Mbed TLS (Renesas)を使用しています。

1.5 開発環境

RSIP CM 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。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。

1.6 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_cm_rx rev1.00

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 8.3.0.202411
(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	**156,990 バイト**	**185,376 バイト**	**134,280 バイト**
RAM	**16,731 バイト**	**19,836 バイト**	**22,876 バイト**
スタック	**3,592 バイト**	**2,108 バイト**	**3,968 バイト**

1.7 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 CM の動作クロック PCLKB は ICLK : PCLKB = 2 : 1 の設定をしています。ドライバは CC-RX、最適化レベル 2 でビルドしています。バージョンは「7.1 動作確認環境」をご参照ください。コンフィグレーションオプションはデフォルト設定です。

1.7.1 RX261

表 1-5 共通機能 API の性能

API	性能 (単位：サイクル)
mbedtls_platform_setup	460,000
mbedtls_platform_teardown	600

表 1-6 鍵管理 API の性能

API	鍵類	性能 (単位：サイクル)
R_RSIP_CM_AES128_InitialKeyWrap	Encrypted	10,000
	Plaintext	7,000
R_RSIP_CM_AES256_InitialKeyWrap	Encrypted	10,000
	Plaintext	7,000
R_RSIP_CM_ECC_secp256r1_InitialPrivateKeyWrap	Encrypted	10,000
	Plaintext	7,000
R_RSIP_CM_ECC_secp256k1_InitialPrivateKeyWrap	Encrypted	10,000
	Plaintext	7,000
R_RSIP_CM_ECC_brainpoolP256r1_InitialPrivateKeyWrap	Encrypted	10,000
	Plaintext	7,000

表 1-7 乱数生成 API の性能

API	性能 (単位：サイクル)
psa_generate_random	17,000

表 1-8 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	psa_cipher_encrypt_setup	2,600	2,600	2,600
	psa_cipher_update	22,000	29,000	36,000
	psa_cipher_finish	900	900	900
ECB モード復号	psa_cipher_decrypt_setup	2,600	2,600	2,600
	psa_cipher_update	22,000	29,000	36,000
	psa_cipher_finish	1,300	1,300	1,300
CBC モード暗号化	psa_cipher_encrypt_setup	2,800	2,800	2,800
	psa_cipher_update	8,000	8,200	8,300
	psa_cipher_finish	1,400	1,400	1,400
CBC モード復号	psa_cipher_decrypt_setup	2,800	2,800	2,800
	psa_cipher_update	8,100	8,300	8,500
	psa_cipher_finish	1,900	1,900	1,900
CTR モード暗号化	psa_cipher_encrypt_setup	2,900	2,900	2,900
	psa_cipher_update	8,100	8,200	8,400
	psa_cipher_finish	1,400	1,400	1,400
CTR モード復号	psa_cipher_decrypt_setup	3,000	3,000	3,000
	psa_cipher_update	8,100	8,200	8,400
	psa_cipher_finish	1,900	1,900	1,900

表 1-9 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	psa_aead_encrypt_setup	15,000	15,000	15,000
	psa_aead_update_ad	840	840	840
	psa_aead_update	75,000	95,000	100,000
	psa_aead_finish	16,000	16,000	16,000
GCM モード復号	psa_aead_decrypt_setup	15,000	15,000	15,000
	psa_aead_update_ad	840	840	840
	psa_aead_update	75,000	95,000	100,000
	psa_aead_verify	16,000	16,000	16,000
CCM モード暗号化	psa_aead_encrypt_setup	2,700	2,700	2,700
	psa_aead_update_ad	7,600	7,600	7,600
	psa_aead_update	44,000	59,000	73,000
	psa_aead_finish	9,400	9,400	9,400
CCM モード復号	psa_aead_decrypt_setup	2,700	2,700	2,700
	psa_aead_update_ad	7,600	7,600	7,600
	psa_aead_update	44,000	59,000	73,000
	psa_aead_verify	9,300	9,300	9,400

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

表 1-10 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	psa_mac_sign_setup	8,000	8,000	8,000
	psa_mac_update	1,700	1,800	1,900
	psa_mac_sign_finish	2,600	2,600	2,600
CMAC 検証	psa_mac_verify_setup	8,000	8,000	8,000
	psa_mac_update	1,700	1,800	1,900
	psa_mac_verify_finish	2,800	2,800	2,800

表 1-11 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	psa_sign_hash	4,900,000	4,900,000	4,900,000
	psa_verify_hash	9,700,000	9,600,000	9,600,000
Koblitz secp256k1	psa_sign_hash	4,900,000	4,900,000	4,900,000
	psa_verify_hash	9,700,000	9,700,000	9,700,000
Brainpool p256r1	psa_sign_hash	4,900,000	4,900,000	4,900,000
	psa_verify_hash	9,600,000	9,600,000	9,600,000

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 生成	psa_hash_setup	390	390	390
	psa_hash_update	700	2,800	2,900
	psa_hash_finish	3,400	3,400	3,400
SHA224 検証	psa_hash_setup	390	390	390
	psa_hash_update	700	2,800	2,900
	psa_hash_verify	4,300	4,300	4,300
SHA256 生成	psa_hash_setup	380	380	380
	psa_hash_update	700	2,800	2,900
	psa_hash_finish	3,400	3,400	3,400
SHA256 検証	psa_hash_setup	380	380	380
	psa_hash_update	700	2,800	2,900
	psa_hash_verify	4,400	4,400	4,400

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	psa_mac_sign_setup	5,400	5,400	5,400
	psa_mac_update	1,200	3,300	3,400
	psa_mac_sign_finish	11,000	11,000	11,000
HMAC-SHA224 検証	psa_mac_verify_setup	5,500	5,500	5,500
	psa_mac_update	1,200	3,300	3,400
	psa_mac_verify_finish	12,000	12,000	12,000
HMAC-SHA256 生成	psa_mac_sign_setup	5,500	5,500	5,500
	psa_mac_update	1,200	3,300	3,400
	psa_mac_sign_finish	11,000	11,000	11,000
HMAC-SHA256 検証	psa_mac_verify_setup	5,700	5,700	5,700
	psa_mac_update	1,200	3,300	3,400
	psa_mac_verify_finish	12,000	12,000	12,000

2. API 情報

2.1 ハードウェアの要求

RX RSIP CM 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 CM ドライバは、以下モジュールに依存します。

- r_bsp V7.52 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 CM 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_cm_rx_if.h に記載しています。

2.5 整数型

RSIP CM 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

Mbed TLS コンフィグレーションオプションの設定は mbedtls_config.h で行います。オプション名および設定値は以下の表 2-1 に示します。

表 2-1 mbedtls_config.h の定義

オプション名	デフォルト値
MBEDTLS_PLATFORM_SETUP_TEARDOWN_ALT	Defined
MBEDTLS_AES_ALT	Defined
MBEDTLS_CCM_ALT	Defined
MBEDTLS_GCM_ALT	Defined
MBEDTLS_SHA256_ALT	Defined
MBEDTLS_SHA256_PROCESS_ALT	Defined
MBEDTLS_AES_SETKEY_ENC_ALT	Defined
MBEDTLS_AES_SETKEY_DEC_ALT	Defined
MBEDTLS_AES_ENCRYPT_ALT	Defined
MBEDTLS_AES_DECRYPT_ALT	Defined
MBEDTLS_ECDSA_VERIFY_ALT	Defined
MBEDTLS_ECDSA_SIGN_ALT	Defined
MBEDTLS_ENTROPY_HARDWARE_ALT	Defined
MBEDTLS_CIPHER_MODE_CBC	Defined
MBEDTLS_CIPHER_MODE_CTR	Defined
MBEDTLS_CIPHER_PADDING_PKCS7	Defined
MBEDTLS_CIPHER_PADDING_ONE_AND_ZEROS	Defined
MBEDTLS_CIPHER_PADDING_ZEROS_AND_LEN	Defined
MBEDTLS_CIPHER_PADDING_ZEROS	Defined
MBEDTLS_ECP_DP_SECP256R1_ENABLED	Defined
MBEDTLS_ECP_DP_SECP256K1_ENABLED	Defined
MBEDTLS_ECP_DP_BP256R1_ENABLED	Defined
MBEDTLS_ERROR_STRERROR_DUMMY	Defined
MBEDTLS_GENPRIME	Defined
MBEDTLS_FS_IO	Defined
MBEDTLS_NO_PLATFORM_ENTROPY	Defined
MBEDTLS_PKCS1_V15	Defined
MBEDTLS_PKCS1_V21	Defined
MBEDTLS_VERSION_FEATURES	Defined
MBEDTLS_AES_C	Defined
MBEDTLS_ASN1_PARSE_C	Defined
MBEDTLS_ASN1_WRITE_C	Defined
MBEDTLS_BASE64_C	Defined
MBEDTLS_BIGNUM_C	Defined
MBEDTLS_CCM_C	Defined
MBEDTLS_CIPHER_C	Defined
MBEDTLS_CMAC_C	Defined
MBEDTLS_CTR_DRBG_C	Defined
MBEDTLS_CTR_DRBG_C_ALT	Defined
MBEDTLS_ECDSA_C	Defined
MBEDTLS_ECP_C	Defined
MBEDTLS_ENTROPY_C	Defined
MBEDTLS_ERROR_C	Defined
MBEDTLS_GCM_C	Defined
MBEDTLS_HKDF_C	Defined

オプション名	デフォルト値
MBEDTLS_LMS_C	Defined
MBEDTLS_MD_C	Defined
MBEDTLS_MD5_C	Defined
MBEDTLS_OID_C	Defined
MBEDTLS_PEM_PARSE_C	Defined
MBEDTLS_PEM_WRITE_C	Defined
MBEDTLS_PKCS5_C	Defined
MBEDTLS_PKCS12_C	Defined
MBEDTLS_PLATFORM_C	Defined
MBEDTLS_PSA_CRYPTO_C	Defined
MBEDTLS_PSA_CRYPTO_ACCEL_DRV_C	Defined
MBEDTLS_PSA_CRYPTO_STORAGE_C	Defined
MBEDTLS_PSA_ITS_FILE_C	Defined
MBEDTLS_RSA_C	Defined
MBEDTLS_SHA224_C	Defined
MBEDTLS_SHA256_C	Defined
MBEDTLS_VERSION_C	Defined
MBEDTLS_MPI_WINDOW_SIZE	6
MBEDTLS_MPI_MAX_SIZE	1024
MBEDTLS_ECP_WINDOW_SIZE	6
MBEDTLS_ECP_FIXED_POINT_OPTIM	1
MBEDTLS_CHECK_RETURN	Enabled

2.6.1 Platform の設定

RSIP CM を使用して HW アクセラレーションするには、マクロ MBEDTLS_PLATFORM_SETUP_TEARDOWN_ALT をコンフィグファイル mbedtls_config.h で定義してください。これにより、RSIP を初期化するコードが有効になります。

2.6.2 乱数生成の設定

マクロ MBEDTLS_ENTROPY_HARDWARE_ALT をコンフィグファイル mbedtls_config.h で定義する必要があります。これにより、RSIP の TRNG をエントロピーソースとして使用できるようになります。この機能がないと、他の暗号化操作は(ソフトウェアのみのモードであっても)動作しません。

2.6.3 Wrapped Key の設定

RSIP を使用して Wrapped Key 生成する時は、キータイプ属性に PSA_KEY_TYPE_AES_WRAPPED または PSA_KEY_TYPE_ECC_KEY_PAIR_WRAPPED(curve)を設定します。

2.6.4 AES の設定

AES128/256 操作のハードウェアアクセラレーションを有効にするには、マクロ MBEDTLS_AES_SETKEY_ENC_ALT、MBEDTLS_AES_SETKEY_DEC_ALT、MBEDTLS_AES_ENCRYPT_ALT、および MBEDTLS_AES_DECRYPT_ALT をコンフィグファイル mbedtls_config.h で定義する必要があります。デフォルトでは AES が有効になっています。

2.6.5 ECC の設定

ECC キー生成操作のハードウェアアクセラレーションを有効にするには、マクロ MBEDTLS_ECP_ALT を構成ファイルで定義する必要があります。ECDSA の場合は、マクロ MBEDTLS_ECDSA_SIGN_ALT および MBEDTLS_ECDSA_VERIFY_ALT を定義する必要があります。デフォルトでは、ECC および ECDSA が有効になっています。ECC を無効にするには、MBEDTLS_ECP_C、MBEDTLS_ECDSA_C、および MBEDTLS_ECDH_C を定義を削除します。

2.6.6 SHA224/256 の設定

SHA224/256 計算のハードウェアアクセラレーションを有効にするには、マクロ MBEDTLS_SHA256_ALT および MBEDTLS_SHA256_PROCESS_ALT を構成ファイルで定義する必要があります。デフォルトでは SHA256 が有効になっています。

2.7 構造体

RSIP CM ドライバで Key Injection 使用している構造体の定義を示します。

表 2-2 RSIP CM ドライバで Key Injection 構造体定義

定義	意味
rsip_key_injection_type_t	Key injection タイプ構造体
rsip_aes_wrapped_key_t	AES Wrapped Key 構造体
rsip_ecc_private_wrapped_key_t	ECC Private Wrapped Key 構造体

2.8 戻り値

以下に RSIP CM ドライバの Key Injection 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 2-3 戻り値 enum fsp_err_t

列挙子	値	意味
FSP_SUCCESS	0x00000	正常終了
FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	0x10001	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_SCE_FAIL	0x10002	入力パラメータが不正

2.9 FIT モジュールの追加方法

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

(1) 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 CM ドライバの使用方法

RX ファミリ RSIP CM ドライバは以下の機能を提供します。API 詳細は 4 章で説明します。

- 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 CM 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。鍵の注入フローについては 3.5.1 を参照してください。詳細な鍵の注入方法は 5 章で説明します。

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 CM ドライバの暗号演算はオープンソース PSA Crypto API を通じて実現します。RX RSIP CM ドライバの rm_psa_crypto と RSIP CM 部分は、PSA Crypto API と連携し、暗号関連の操作を担う主要なモジュールです。RX RSIP CM ドライバの RSIP Key Injection 部分は鍵の注入をサポートします。RSIP CM ドライバの構成を図 3-1 RSIP CM 構成図に示します。

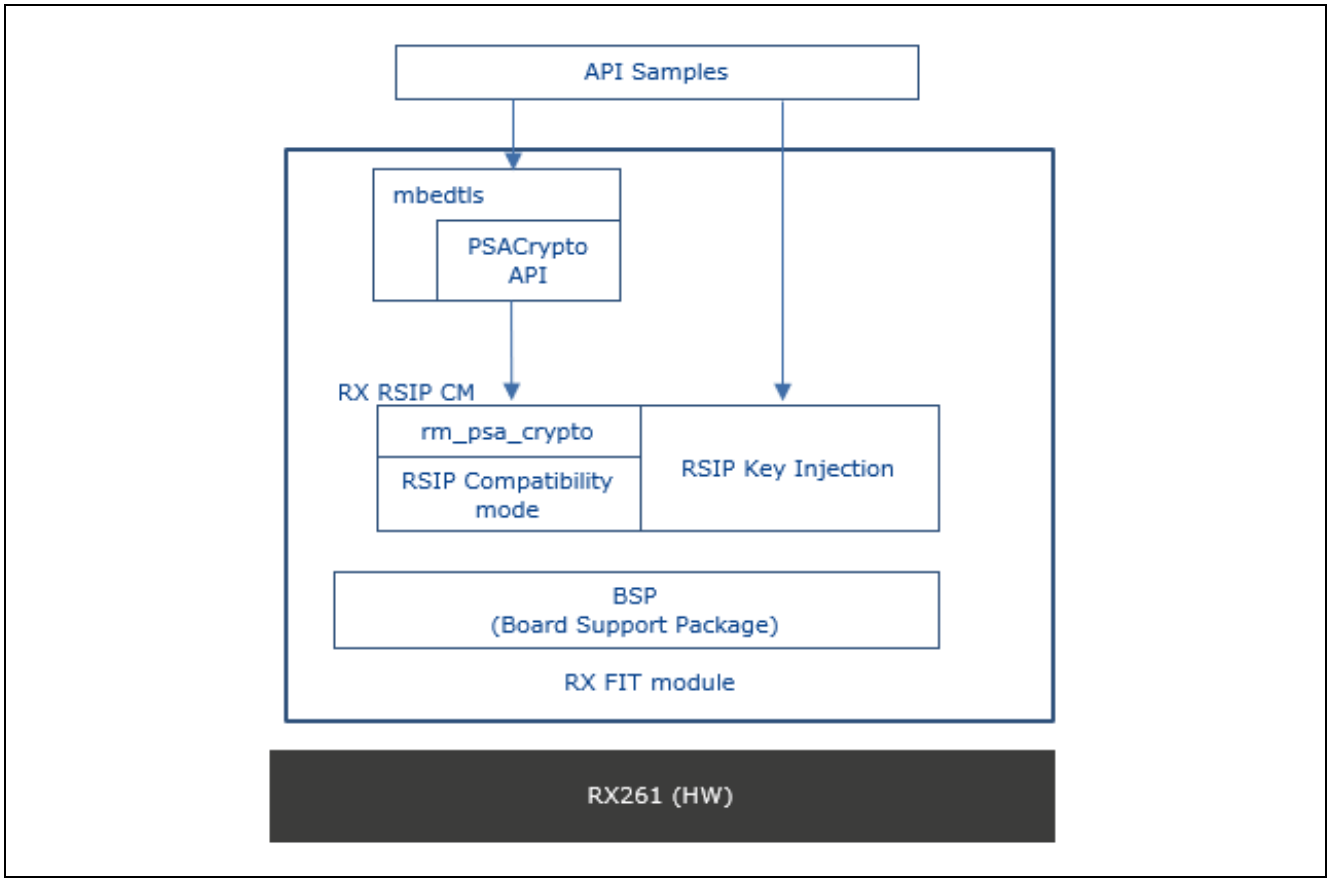

図 3-1 RSIP CM 構成図

3.1 初期化

RSIP の初期化処理には、PSA Crypto API を使用する前に mbedtls_platform_setup() を呼び出す必要があります。

No	PSA Crypto API	説明
1	mbedtls_platform_setup()	初期化

3.2 メモリ使用

使用される PSA Crypto 機能に応じて、ヒープ サイズの設定が必要です。割り当てられるヒープの合計は、個々のアルゴリズムのヒープ要件の合計になります。

アルゴリズム	ヒープ サイズ(bytes)
SHA224/256	0x500
AES	0x2000
ECC	0x2000

RSIP ドライバモジュールが使用される場合は、最小スタック 0x2000 が必要です。

3.3 制限事項

3.3.1 動作エンディアン

リトルエンディアンのみがサポートされています。

3.3.2 MBEDTLS_PLATFORM_SETBUF_MACRO の定義

RX RSIP CM ドライバでは、ビルドエラーを防ぐために MBEDTLS_PLATFORM_SETBUF_MACRO に、dummy_setbuf()というダミー関数を定義しています。mbedtls_config.h ファイルに定義されている MBEDTLS_PLATFORM_SETBUF_MACRO にユーザ定義関数を定義することで、ユーザが使用する関数に置き換えることができます。

3.4 シングルパート演算とマルチパート演算

PSA Crypto API は暗号演算を一つの API で提供するものと複数の API で提供するものがあります。本書では、前者をシングルパート演算、後者をマルチパート演算と呼びます。

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

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

シングルパート演算の詳細は PSA Certified Crypto API 1.0 で「3.3.1. Single-part Functions」に参照ください。マルチパート演算の詳細は PSA Certified Crypto API 1.0 で「3.3.2. Multi-part operations」に参照ください。

3.5 鍵の管理

No	PSA Crypto API	RSIP CM API	説明
1	-	R_RSIP_CM_AES128_InitialKeyWrap() R_RSIP_CM_AES256_InitialKeyWrap() R_RSIP_CM_ECC_secp256r1_InitialPrivateKeyWrap() R_RSIP_CM_ECC_secp256k1_InitialPrivateKeyWrap() R_RSIP_CM_ECC_brainpoolP256r1_InitialPrivateKeyWrap()	鍵の注入
2	psa_generate_key()	-	鍵の生成

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

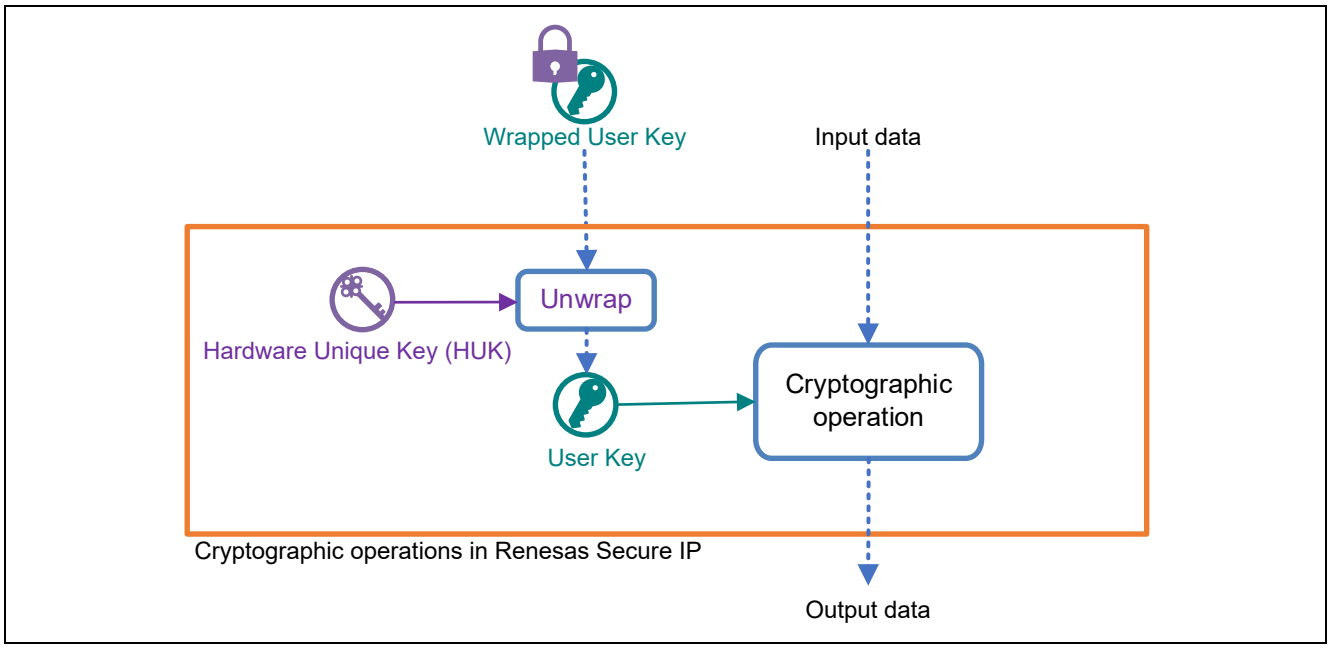

図 3-2 RSIP CM ドライバ暗号操作における鍵の取り扱い

RSIP CM 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 CM 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は、平文のまま使用します。

RSIP CM 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。

3.5.1 鍵の注入

RX RSIP CM ドライバの鍵の注入は Wrapped Key のみを対応しています。平文鍵を直接暗号演算 API に入力することはできません。平文鍵を使用する場合、Key Injection API を使用して、Wrapped Key に変換をして使用します。ただし、非対称暗号の公開鍵は、Wrapped Key に変換せずに、平文鍵のまま直接暗号演算 API で使用できます。

3.5.1.1 Encrypted Key を使用した鍵の注入

鍵注入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 Encrypted Key の注入のフローを示します。

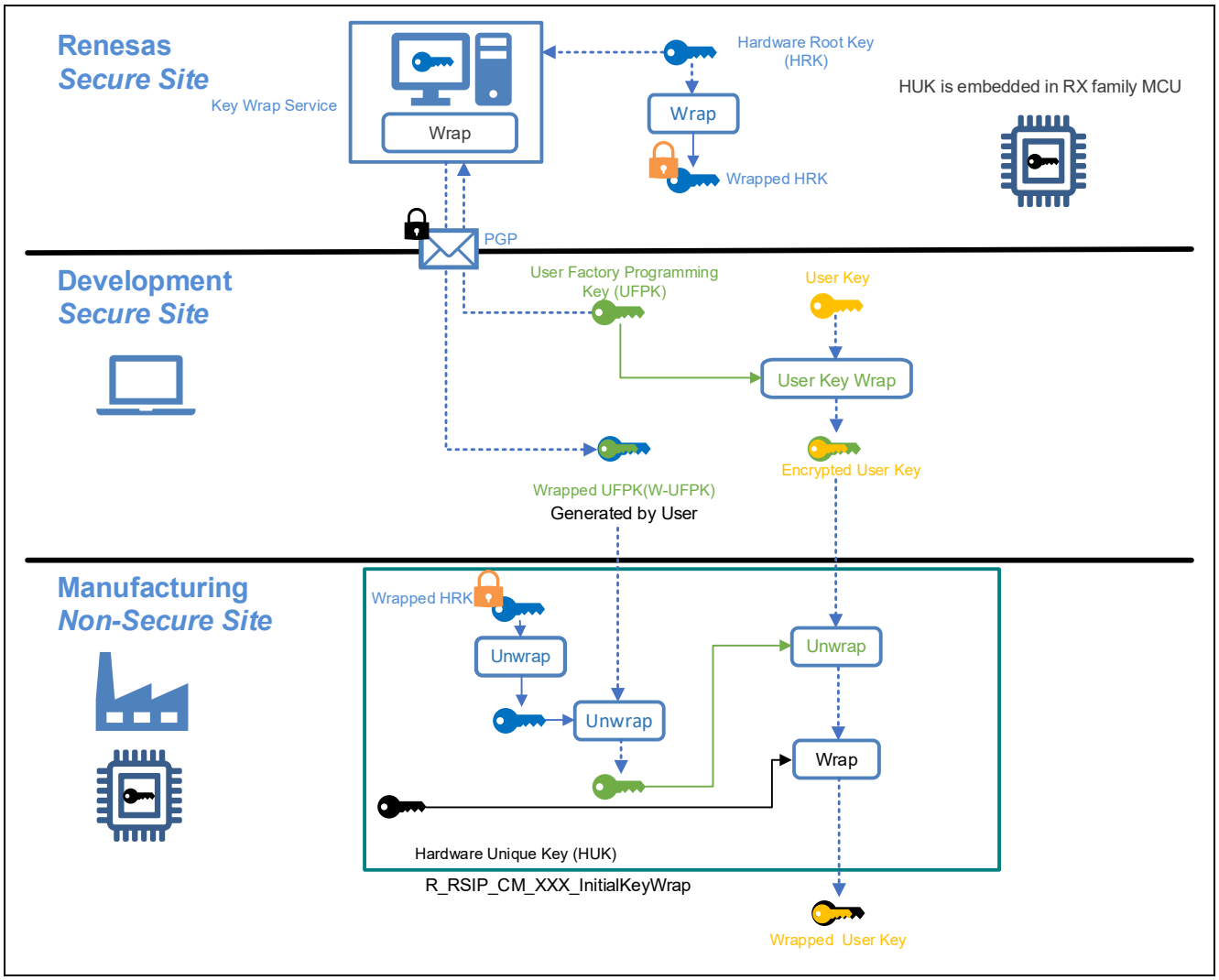

図 3-3 Encrypted Key を使用した鍵の注入

お客様のセキュアなサイト(図 3-3 Development Secure Site)で、ユーザ鍵、UFPK の生成を行います。UFPK は、お客様のアプリケーションで使用する User Key のラップに使用し、Encrypted User Key を生成します。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。鍵注入時には、Encrypted User Key を Key Injection API(R_RSIP_CM_XXX_InitialKeyWrap)に入力し、Wrapped key 形式の鍵を生成します。

ユーザ鍵を UFPK でラップして Encrypted User Key 生成する方法を図 3-4 に示します。ラップに使用する UFPK の先頭 128bit を鍵として、AES-128 CBC モードでユーザ鍵を暗号化します。ラップに使用する UFPK の後続 128bit を鍵として、AES-128 CBC-MAC でユーザ鍵の MAC を計算します。ユーザ鍵にユーザ鍵の MAC を連結し、それらを暗号化することで Encrypted User Key を生成します。

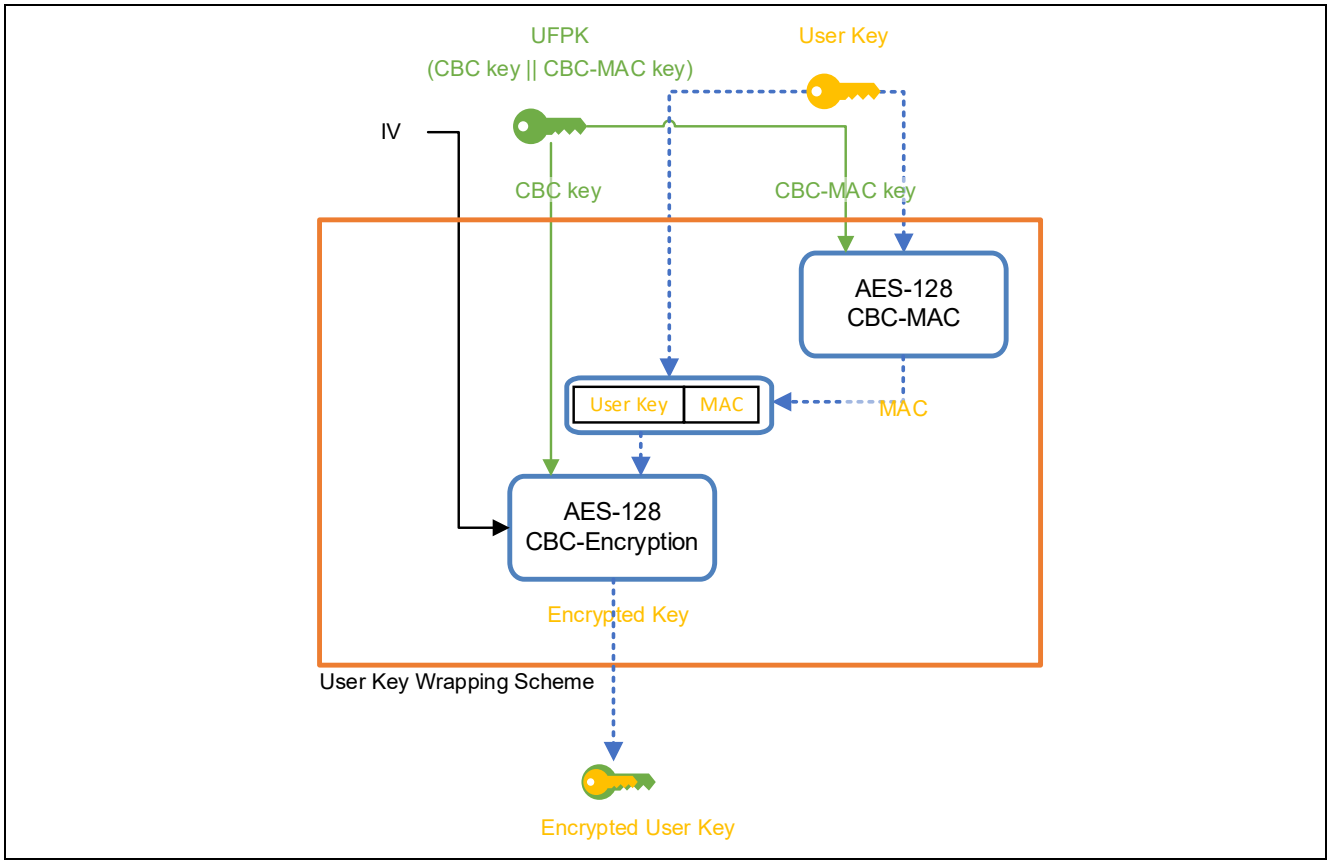

図 3-4 鍵注入時のユーザ鍵ラップ方式

Encrypted User Key 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。

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

3.5.1.2 平文鍵を使用した鍵の注入

平文鍵の注入は、PSA Crypto 鍵設定 API(psa_xxx_setup)で Key Injection API を使用して、ユーザ鍵を HUK で Wrapped Key に変換します。図 3-5 に平文鍵の注入のフローを示します。

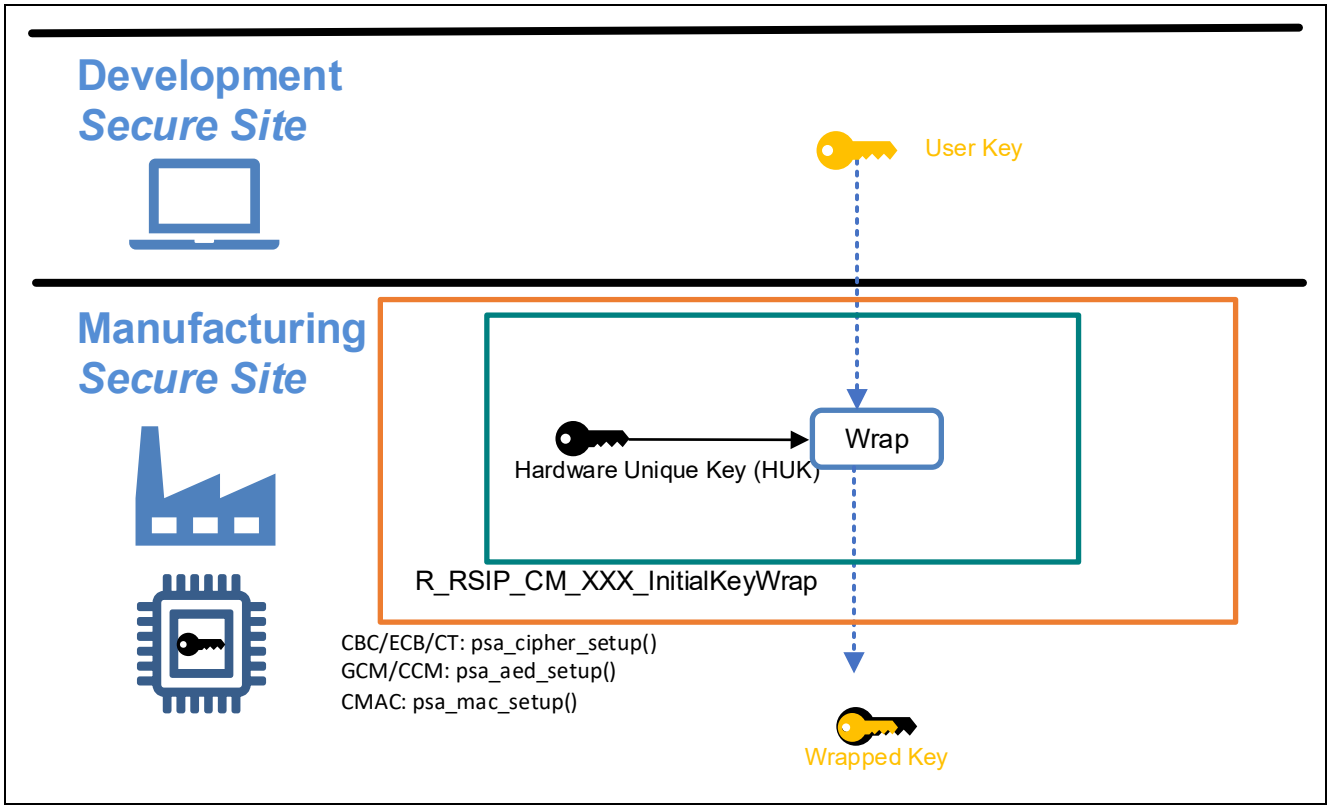

図 3-5 平文鍵を使用した鍵の注入

3.5.2 鍵の生成

鍵の生成は、PSA Crypto API の鍵生成機能を用いて乱数鍵を生成します。指定された鍵タイプが Wrapped Key の場合、乱数生成機能から鍵を生成し、Wrapped Key の形式で鍵を出力します。図 3-6 に Wrapped Key の生成のフローを示します。指定された鍵タイプが平文鍵の場合、乱数生成機能から鍵を生成し、平文のまま鍵を出力します。図 3-7 に平文鍵生成のフローを示します。ECC の平文鍵の生成はサポートしていません。

鍵の生成時に、鍵の属性のライフタイムを永続鍵(persistent keys)に設定することはサポートしていません。暫存鍵(volatile keys)のみ設定可能です。

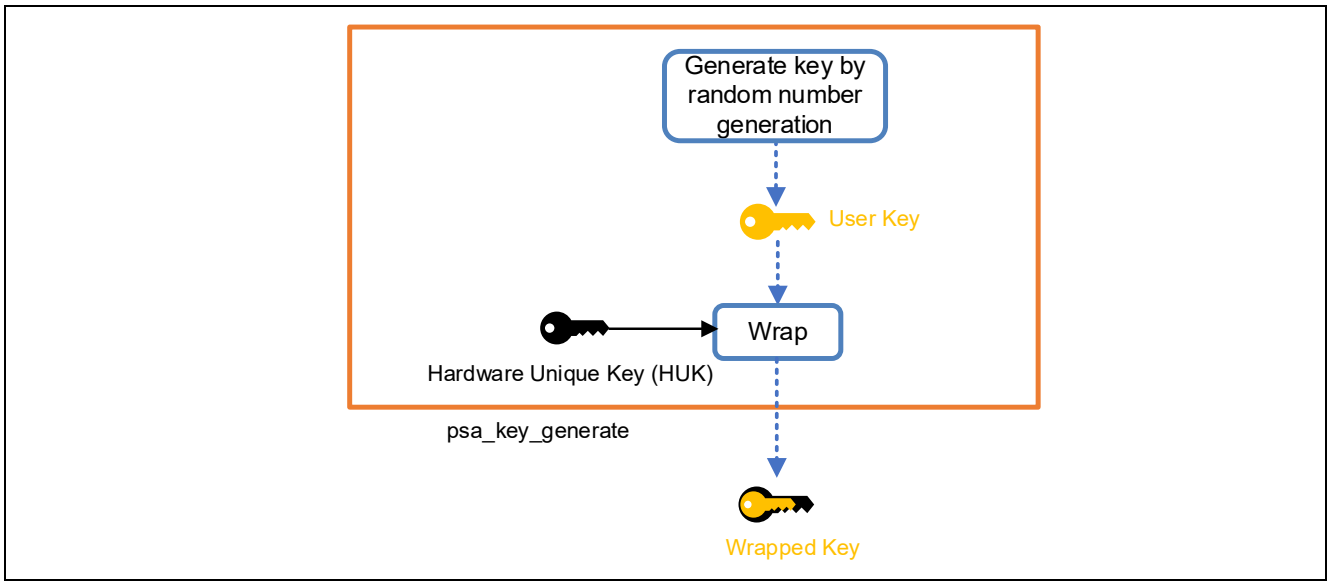

図 3-6 Wrapped Key 生成のフロー

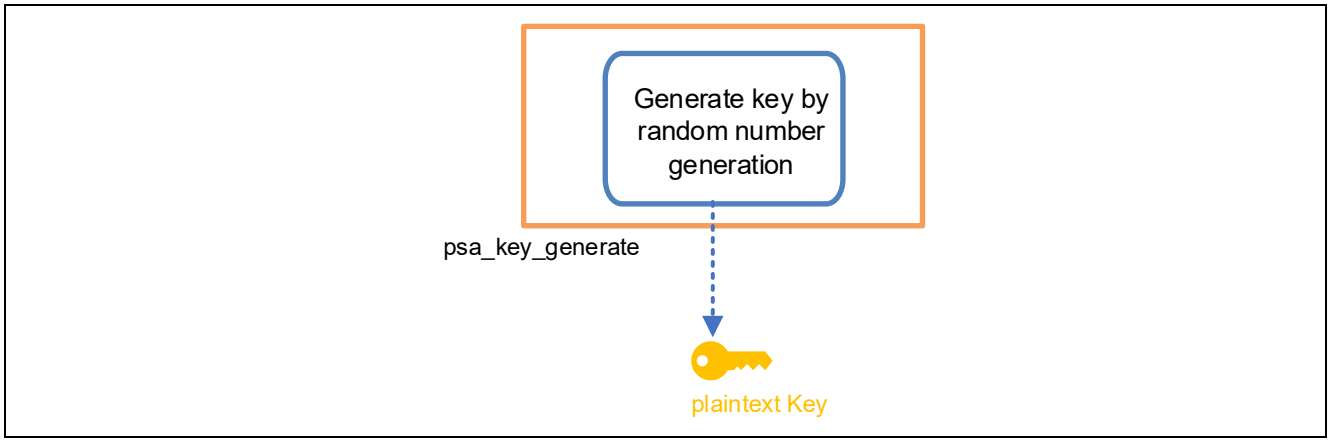

図 3-7 平文鍵を使用した鍵の注入

3.6 乱数生成

No	PSA Crypto API	説明
1	psa_generate_random	乱数の生成

3.7 対称鍵暗号

No	PSA Crypto API	説明
1	**AES-ECB/CBC/CTR** psa_cipher_encrypt psa_cipher_decrypt psa_cipher_encrypt_setup psa_cipher_decrypt_setup psa_cipher_update psa_cipher_finish	AES-ECB/CBC/CTR 暗号演算 AES-ECB/CBC/CTR 複号演算
2	**AES-CCM/GCM** psa_aead_encrypt psa_aead_decrypt psa_aead_encrypt_setup psa_aead_decrypt_setup psa_aead_set_nonce psa_aead_update_ad psa_aead_update psa_aead_finish psa_aead_verify	AES-CCM/GCM 暗号演算 AES-CCM/GCM 複号演算
3	**AES-CMAC** psa_mac_compute psa_mac_verify psa_mac_sign_setup psa_mac_verify_setup psa_mac_update psa_mac_sign_finish psa_mac_verify_finish	AES-CMAC 署名演算 AES-CMAC 検証演算

3.8 非対称鍵暗号

No	PSA Crypto API	説明
1	psa_sign_message psa_verify_message psa_sign_hash psa_verify_hash mbedtls_ecp_mul	ECC 署名演算 ECC 署名検証 ECC スカラ倍演算

3.9 HASH 関数

No	PSA Crypto API	説明
1	psa_hash_compute psa_hash_setup psa_hash_update psa_hash_finish psa_hash_verify	HASH 演算 HASH 検証

4. API 関数

4.1 API 一覧と詳細

RX RSIP CM ドライバの API を以下の表にまとめます。

初期化 API は Mbed TLS の platform 設定関数を利用して、RX RSIP CM ドライバをオープンします。API の詳細は 1.4 関連文書の Mbed TLS documentation hub を参照ください。

鍵管理、乱数生成、AES、ECC、HASH API は Mbed TLS の PSA Crypto API をサポートしています。API の詳細は 1.4 関連文書の PSA Certified Crypto API 1.0 を参照ください。

バージョン情報と Key Injection API は RX RSIP CM ドライバ独自の API を提供します。

表 4-1 初期化 API

API	説明	詳細
mbedtls_platform_setup	RSIP をモジュールストップ状態から解除し、RSIP CM ドライバをオープンします。	Mbed TLS documentation hub の以下を参照「platform setup」

表 4-2 鍵管理 API

API	説明	詳細
psa_key_attributes_init	鍵属性オブジェクトの初期値を取得します。	PSA Certified Crypto API 1.0 の以下を参照「9. Key management」
psa_get_key_attributes	鍵属性を取得します。	
psa_reset_key_attributes	鍵属性オブジェクトの状態を初期化します。	
psa_set_key_type	鍵タイプを設定します。	
psa_get_key_type	鍵タイプを取得します。	
psa_get_key_bits	鍵ビット長を取得します。	
psa_set_key_bits	鍵ビット長を設定します。	
psa_set_key_lifetime	Persistent key のライフタイムを設定します。	
psa_get_key_lifetime	鍵のライフタイムを取取得します。	
psa_set_key_id	鍵 ID を設定します。	
psa_get_key_id	鍵 ID を取得します。	
psa_set_key_algorithm	鍵の暗号アルゴリズムを設定します。	
psa_get_key_algorithm	鍵の暗号アルゴリズムを取得します。	
psa_set_key_usage_flag	鍵の使用用途を設定します。	
psa_get_key_usage_flag	鍵の使用用途を取得します。	
psa_import_key	鍵をインポートします。	
psa_generate_key	鍵もしくはペア鍵を生成します。	
psa_copy_key	鍵をコピーします。	
psa_destoroy_key	鍵を破棄します。	
psa_purge_key	不要な鍵のコピーを削除します。	
psa_export_key	鍵をエクスポートします。	
psa_export_public_key	公開鍵もしくはペア鍵の公開鍵をエクスポートします。	

表 4-3 乱数生成 API

API	説明	詳細
psa_generate_random	乱数を生成します。	PSA Certified Crypto API 1.0 の以下を参照「10.10.1. Random number generation」

表 4-4 AES-ECB/CBC/CTR API

API	説明	詳細
psa_cipher_encrypt	AES 暗号演算を実行します。	PSA Certified Crypto API 1.0 の以下を参照「10.4. Unauthenticated ciphers」
psa_cipher_decrypt	AES 複号演算を実行します。	
psa_cipher_operation_init	AES 暗号演算を実行する準備を行います。	
psa_cipher_encrypt_setup	AES 暗号演算の鍵を設定します。	
psa_cipher_decrypt_setup	AES 複号演算の鍵を設定します。	
psa_cipher_set_iv	AES 暗号演算の IV を設定します。	
psa_cipher_generate_iv	AES 暗号演算の IV を生成します。	
psa_cipher_update	AES 暗号演算を実行します。	
psa_cipher_finish	AES 暗号演算を終了します。	

表 4-5 AES-CCM/GCM API

API	説明	詳細
psa_aead_encrypt	AEAD 暗号化演算を実行します。	PSA Certified Crypto API 1.0 の以下を参照「10.5. Authenticated encryption with associated data (AEAD)」
psa_aead_decrypt	AEAD 復号演算を実行します。	
psa_aead_operation_init	AEAD 演算オブジェクトの初期値を返します。	
psa_aead_encrypt_setup	AEAD 暗号化演算用の鍵を設定します。	
psa_aead_decrypt_setup	AEAD 復号演算用の鍵を設定します。	
psa_aead_set_lengths	AEAD 演算の追加認証データのサイズを設定します。	
psa_aead_generate_nonce	AEAD 演算のノンスを生成します。	
psa_aead_set_nonce	AEAD 演算のノンスを設定します。	
psa_aead_update_ad	AEAD 演算に使用する追加認証データを渡します。	
psa_aead_update	AEAD 演算を実行します。	
psa_aead_finish	AEAD 暗号化演算の終了処理を行います。	
psa_aead_verify	AEAD 復号演算を終了し検証を行います。	
psa_aead_abort	AEAD 演算を中断します。	

表 4-6 AES-CMAC / HMAC-SHA API

API	説明	詳細
psa_mac_compute	メッセージの MAC 演算を実行します。	PSA Certified Crypto API 1.0 の以下を参照「10.3. Message authentication codes (MAC)」
psa_mac_verify	メッセージの MAC 演算を行い、与えられた MAC と比較を行います。	
psa_mac_operation_init	MAC 演算オブジェクトの初期値を返します。	
psa_mac_sign_setup	MAC 署名演算用の鍵を設定します。	
psa_mac_verify_setup	MAC 検証演算用の鍵を設定します。	
psa_mac_update	MAC 演算を実行します。	
psa_mac_sign_finish	MAC 署名演算の終了処理を行います。	
psa_mac_verify_finish	MAC 検証処理の終了処理を行います。	
psa_mac_abort	MAC 演算を中断します。	

表 4-7 ECC API

API	説明	詳細
psa_sign_message	秘密鍵を使用してメッセージの署名を行います。	PSA Certified Crypto API 1.0 の以下を参照「10.7. Asymmetric signature」 Mbed TLS documentation hub の以下を参照「スカラ倍演算 API」
psa_verify_message	公開鍵を使用してメッセージの検証を行います。	
psa_sign_hash	秘密鍵で HASH の署名を行います。	
psa_verify_hash	秘密鍵で HASH の検証を行います。	
mbedtls_ecp_mul	スカラ倍演算を行います。	

表 4-8 SHA-224/256 API

API	説明	詳細
psa_hash_compute	メッセージの HASH 演算を行います。	PSA Certified Crypto API 1.0 の以下を参照「10.2. Message digests (Hashes)」
psa_hash_compare	メッセージの HASH 演算を行い、与えられた HASH と比較を行います。	
psa_hash_operation_init	HASH 演算オブジェクトの初期値を返します。	
psa_hash_setup	HASH 演算オブジェクトの初期設定を行います。	
psa_hash_update	HASH 演算を行います。	
psa_hash_finish	HASH 演算の終了処理を行います。	
psa_hash_verify	HASH 検証処理の終了処理を行います。	
psa_hash_abort	HASH 演算を中断します。	
psa_hash_clone	ハッシュ演算処理をクローンします。	

表 4-9 バージョン情報 API

API	説明
R_RSIP_CM_GetVersion	RX RSIP CM ドライバのバージョン情報を取得します。

表 4-10 Key Injection API

API	説明
R_RSIP_CM_AES128_InitialKeyWrap	AES 128bit の Wrapped Key を生成します。
R_RSIP_CM_AES256_InitialKeyWrap	AES 256bit の Wrapped Key を生成します。
R_RSIP_CM_ECC_secp256r1_InitialPrivateKeyWrap	ECC secp256r1 の秘密鍵の Wrapped Key を生成します。
R_RSIP_CM_ECC_secp256k1_InitialPrivateKeyWrap	ECC secp256k1 の秘密鍵の Wrapped Key を 生成します。
R_RSIP_CM_ECC_brainpoolP256r1_InitialPrivateKeyWrap	ECC brainpoolP256r1 秘密鍵の Wrapped Key を生成します。

4.1.1 バージョン情報

4.1.1.1 R_RSIP_CM_GetVersion

Format

(1) uint32_t R_RSIP_CM_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

R_RSIP_CM_GetVersion はドライバのバージョンを出力します。

4.1.2 Key Injection

4.1.2.1 R_RSIP_CM_AESxxx_InitialKeyWrap

Format

(1) fsp_err_t R_RSIP_CM_AES128_InitialKeyWrap(
rsip_key_injection_type_t const key_injection_type,
uint8_t const * const p_wrapped_user_factory_programming_key,
uint8_t const * const p_initial_vector,
uint8_t const * const p_user_key,
rsip_aes_wrapped_key_t * const p_wrapped_key)

(2) fsp_err_t R_RSIP_CM_AES256_InitialKeyWrap(
rsip_key_injection_type_t const key_injection_type,
uint8_t const * const p_wrapped_user_factory_programming_key,
uint8_t const * const p_initial_vector,
uint8_t const * const p_user_key,
rsip_aes_wrapped_key_t * const p_wrapped_key)

Parameters

key_injection_type	入力	p_user_key に入力する鍵のタイプ (1) RSIP_KEY_INJECTION_TYPE_ENCRYPTED(0): 暗号化鍵 (2) RSIP_KEY_INJECTION_TYPE_PLAIN (1): 平文鍵
p_wrapped_user_factory_programming_key	入力	W-UFPK key_injection_type が RSIP_KEY_INJECTION_TYPE_PLAIN の場合は、指定不要
p_initial_vector	入力	初期化ベクタ key_injection_type が RSIP_KEY_INJECTION_TYPE_PLAIN の場合は、指定不要
p_user_key	入力	ユーザ鍵 key_injection_type が RSIP_KEY_INJECTION_TYPE_ENCRYPTED の場合は、Encrypted User Key を入力 key_injection_type が RSIP_KEY_INJECTION_TYPE_PLAIN の場合は、Plain User Key を入力
p_wrapped_key	出力	AES の Wrapped Key

Return Values : fsp_err_t

FSP_SUCCESS	正常終了
FSP_ERR_CRYPTO_SCE_FAIL	入力パラメータが不正
FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	本処理に必要なハードウェアリソースが他の処理で使用されていることによるリソース衝突が発生

Description

R_RSIP_CM_AES128_InitialKeyWrap は AES128bit の Wrapped Key を出力するための API です。
R_RSIP_CM_AES256_InitialKeyWrap は AES256bit の Wrapped Key を出力するための API です。

key_injection_type が RSIP_KEY_INJECTION_TYPE_ENCRYPTED (0) の場合は、
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_user_key Encrypted User Key を指定します。
key_injection_type が RSIP_KEY_INJECTION_TYPE_PLAIN (1) の場合は、
p_wrapped_user_factory_programming_key と p_initial_vector が指定不要です。p_user_key は Plain User Key を指定します。
p_user_key ユーザ鍵には 7.3.1 ユーザ鍵フォーマット AES で示すデータを入力してください。

4.1.2.2 R_RSIP_CM_ECC_xxx_InitialPrivateKeyWrap

Format

(1) fsp_err_t R_RSIP_CM_ECC_secp256r1_InitialPrivateKeyWrap (
rsip_key_injection_type_t const key_injection_type,
uint8_t const * const p_wrapped_user_factory_programming_key,
uint8_t const * const p_initial_vector,
uint8_t const * const p_user_key,
rsip_aes_wrapped_key_t * const p_wrapped_key)

(2) fsp_err_t R_RSIP_CM_ECC_secp256k1_InitialPrivateKeyWrap (
rsip_key_injection_type_t const key_injection_type,
uint8_t const * const p_wrapped_user_factory_programming_key,
uint8_t const * const p_initial_vector,
uint8_t const * const p_user_key,
rsip_aes_wrapped_key_t * const p_wrapped_key)

(3) fsp_err_t R_RSIP_CM_ECC_brainpoolP256r1_InitialPrivateKeyWrap (
rsip_key_injection_type_t const key_injection_type,
uint8_t const * const p_wrapped_user_factory_programming_key,
uint8_t const * const p_initial_vector,
uint8_t const * const p_user_key,
rsip_aes_wrapped_key_t * const p_wrapped_key)

Parameters

key_injection_type	入力	p_user_key に入力する鍵のタイプ (1) RSIP_KEY_INJECTION_TYPE_ENCRYPTED(0): 暗号化鍵 (2) RSIP_KEY_INJECTION_TYPE_PLAIN (1): 平文鍵
p_wrapped_user_factory_programming_key	入力	W-UFPK key_injection_type が RSIP_KEY_INJECTION_TYPE_PLAIN の場合は、指定不要
p_initial_vector	入力	初期化ベクタ key_injection_type が RSIP_KEY_INJECTION_TYPE_PLAIN の場合は、指定不要
p_user_key	入力	ユーザ鍵 key_injection_type が RSIP_KEY_INJECTION_TYPE_ENCRYPTED の場合は、Encrypted User Key を入力 key_injection_type が RSIP_KEY_INJECTION_TYPE_PLAIN の場合は、Plain User Key を入力
p_wrapped_key	出力	256-bit ECC 秘密鍵の Wrapped Key

Return Values : fsp_err_t

FSP_SUCCESS	正常終了
FSP_ERR_CRYPTO_SCE_FAIL	入力パラメータが不正
FSP_ERR_CRYPTO_SCE_RESOURCE_CONFLICT	本処理に必要なハードウェアリソースが他の処理で使用されていることによるリソース衝突が発生

Description

R_RSIP_CM_ECC _secp256r1_InitialPrivateKeyWrap は ECC 256bit の secp256r1 秘密鍵の Wrapped Key を出力するための API です。
R_RSIP_CM_ECC _secp256k1_InitialPrivateKeyWrap は ECC 256bit の secp256k1 秘密鍵の Wrapped Key を出力するための API です。
R_RSIP_CM_ECC _brainpoolP256r1_InitialPrivateKeyWrap は ECC 256bit の brainpoolP256r1 秘密鍵の Wrapped Key を出力するための API です。
key_injection_type が RSIP_KEY_INJECTION_TYPE_ENCRYPTED(0) の場合は、
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_user_key は Encrypted User Key を指定します。
key_injection_type が RSIP_KEY_INJECTION_TYPE_PLAIN (1) の場合は、
p_wrapped_user_factory_programming_key と p_initial_vector が指定不要です。
p_user_key ユーザ鍵には 7.3.2 ユーザ鍵フォーマット ECC で示すデータを入力してください。

5. 鍵の注入

本章では、RSIP CM 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。鍵の注入もデフォルトで Wrapped Key として扱われます。現在のバージョンでは、ECC の公開鍵のみ平文鍵の使用をサポートしています。

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 CM ドライバの鍵注入のメカニズムは 3.5.1 鍵の注入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。

3.5.1 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 7.3 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 3-4 鍵注入時のユーザ鍵ラップ方式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、5.2 を参照してください。

5.2 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。

5.2.1 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。

5.2.1.1 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 "22222222222222222222222222222222111111111111111111111111111111"

   /output "C:¥work¥ufpk.key"

```
C:¥Renesas¥SecurityKeyMangementTool¥cli>skmt.exe /genufpk /ufpk "22222222222222222222222222222222
11111111111111111111111111111111" /output "C:¥work¥ufpk.key"
UFPK: 22222222222222222222222222222222111111111111111111111111111111
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

5.2.1.2 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**] タブに 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。

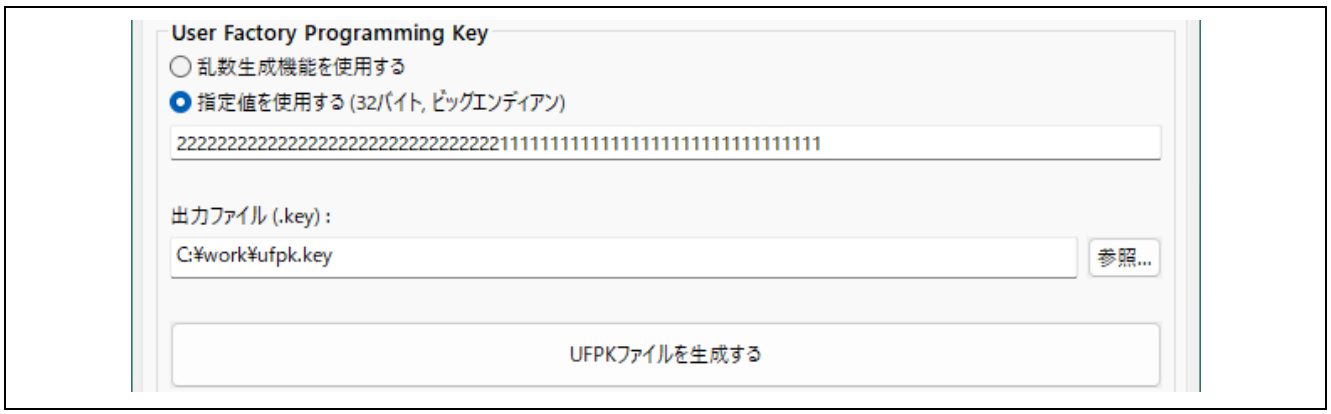

図 5-4 [**UFPK 生成**]タブ 指定値で UFPK を生成する場合の例

"**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。

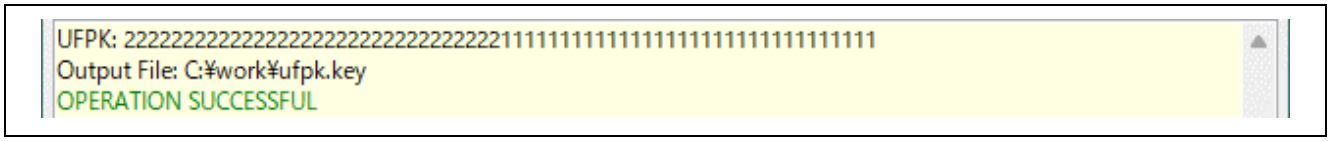

図 5-5 [**UFPK 生成**]タブ 実行結果

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

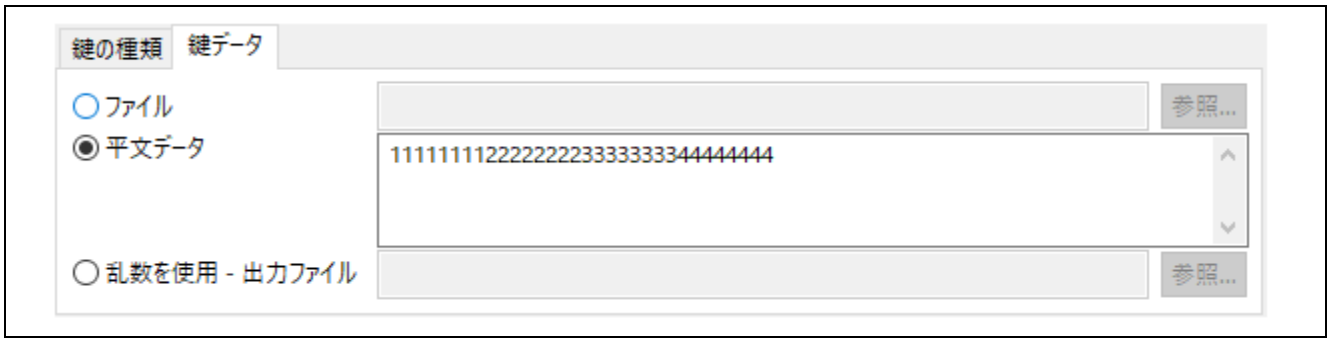

図 5-7 [**鍵のラッピング**] - [**鍵のデータ**]タブ AES128 鍵ファイルを C ソース出力設定例

正常に終了すると以下のように表示されます。

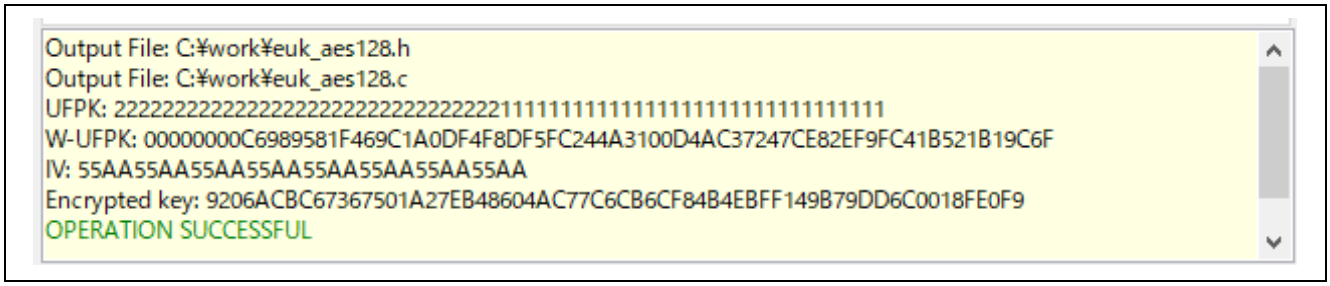

図 5-8 [**鍵のラッピング**]タブ 実行結果

出力された C ソースファイルのデータを、5.2.1.1 と同様の方法で組み込みます。

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_cm_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リトルエンディアンで動作します。

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。

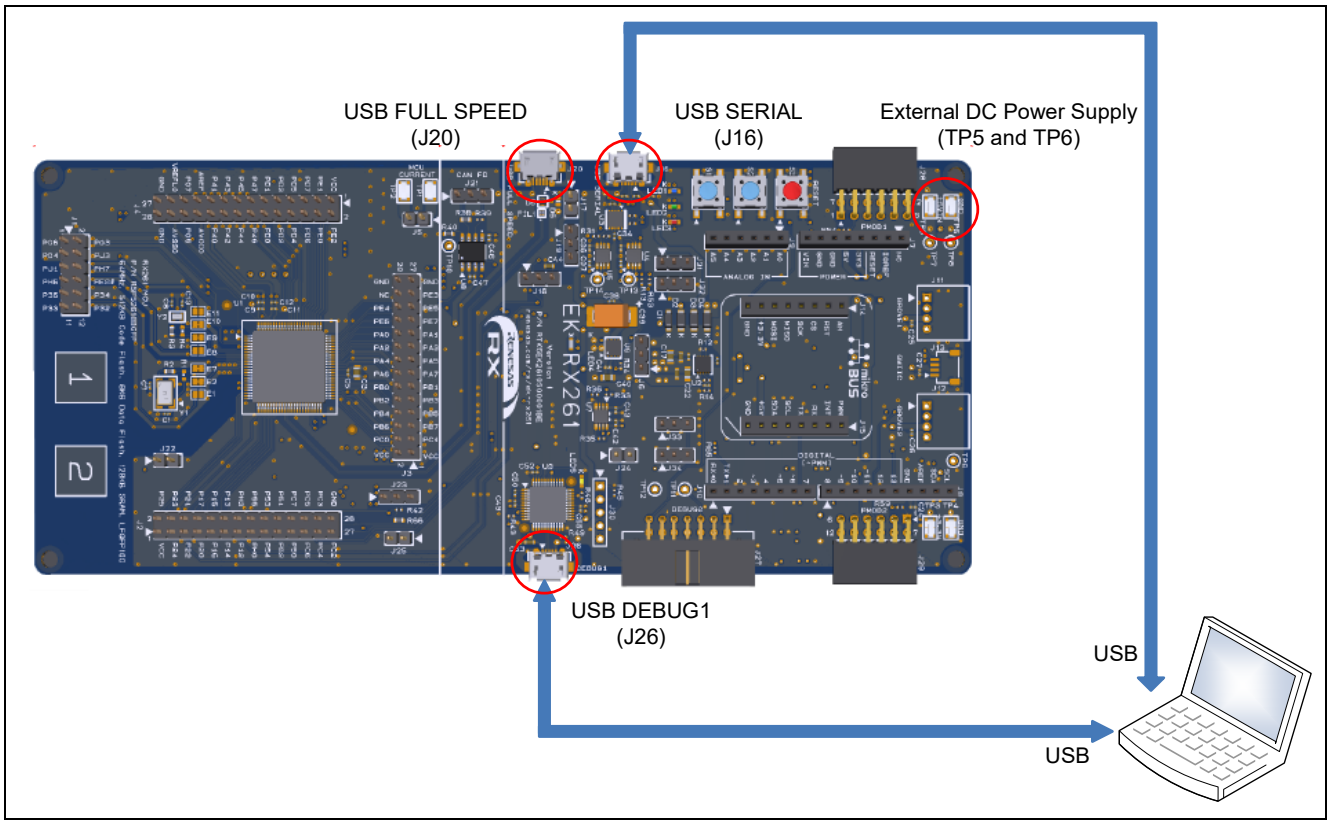

図 6-1 EK-RX261 と PC の接続

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。

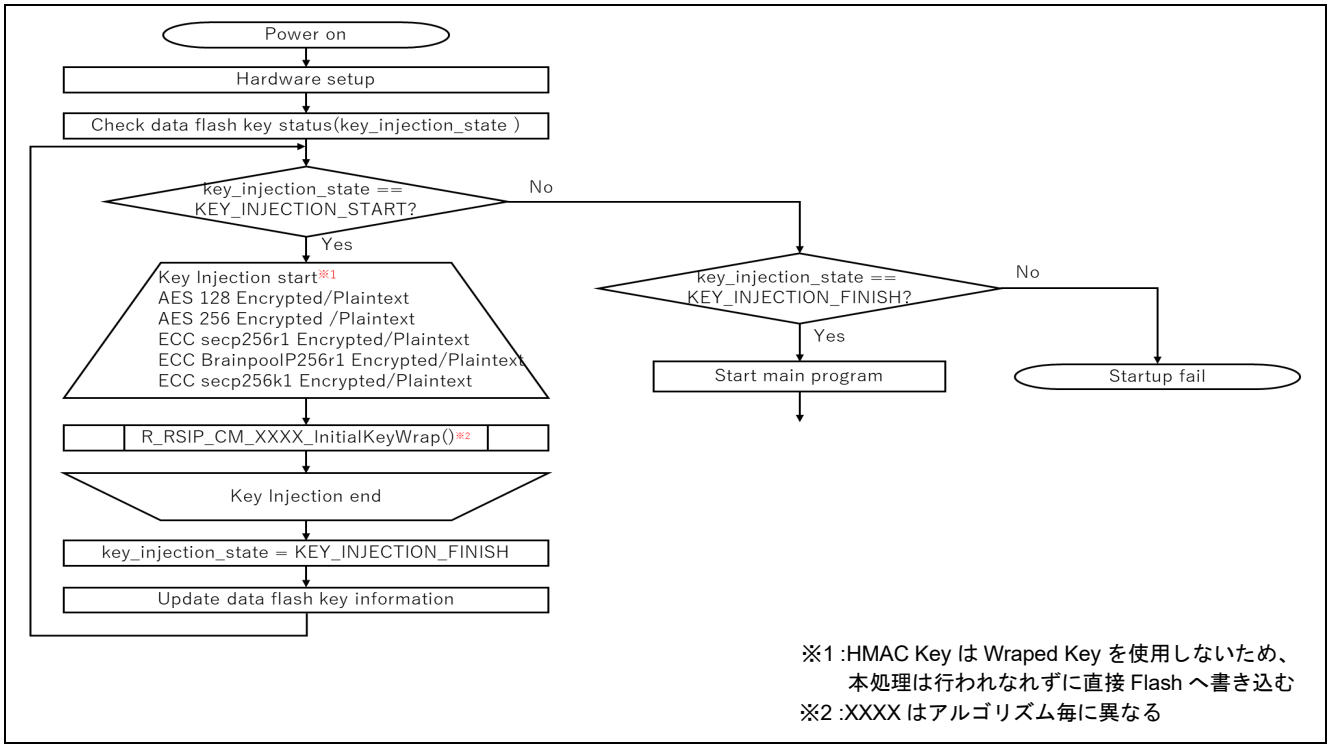

図 6-2 鍵注入と暗号の使用方法のデモプロジェクトの動作フロー

デモプロジェクトの状態管理、使用する鍵は、データフラッシュ(key_block_data)で管理しています。鍵のインジェクション時もしくは、データ更新途中の電断対策で、データフラッシュ内にメイン領域とミラー領域の 2 面にデータを保持して、データ管理をしています。

表 6-2 key_block_data 構造体

型	名称	説明
st_key_block_data_t	(key_block_data 構造体)	データフラッシュに置かれる鍵データ構造体
uint32_t	key_injection_status	以下の STATE を示す KEY_INJECTION_START KEY_INJECTION_FINISH 詳細は次表参照
(struct)	key_data	鍵データ格納構造体
rsip_aes_wrapped_key_t	user_aes128_key_index_encrypted	AES-128 演算で使用する Encrypted Key
rsip_aes_wrapped_key_t	user_aes128_key_index_plaintext	AES-128 演算で使用する Plaintext Key
rsip_aes_wrapped_key_t	user_aes256_key_index_encrypted	AES-256 演算で使用する Encrypted Key
rsip_aes_wrapped_key_t	user_aes256_key_index_plaintext	AES-256 演算で使用する Plaintext Key
uint8_t[]	user_sha224hmac_key_index_plaintext	HMAC SHA-224 演算で使用する Plaintext Key
uint8_t[]	user_sha256ac_key_index_plaintext	HMAC SHA-256 演算で使用する Plaintext Key

型	名称	説明
rsip_ecc_private_wrapped_key_t	user_ecc_secp256r1_private_key_index_encrypted	ECDSA secp256r1 署名検証で使用する Encrypted Key
rsip_ecc_private_wrapped_key_t	user_ecc_secp256r1_private_key_index_plaintext	ECDSA secp256r1 署名検証で使用する Plaintext Key
rsip_ecc_private_wrapped_key_t	user_ecc_brainpoolp256r1_private_key_index_encrypted	ECDSA Brainpoolp256r1 署名検証で使用する Encrypted Key
rsip_ecc_private_wrapped_key_t	user_ecc_brainpoolp256r1_private_key_index_plaintext	ECDSA Brainpoolp256r1 署名検証で使用する Plaintext Key
rsip_ecc_private_wrapped_key_t	user_ecc_secp256k1_private_key_index_encrypted	ECDSA secp256k1 署名検証で使用する Encrypted Key
rsip_ecc_private_wrapped_key_t	user_ecc_secp256k1_private_key_index_plaintext	ECDSA secp256k1 署名検証で使用する Plaintext Key

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 6-3 デモプロジェクトの状態

状態	動作内容
KEY_INJECTION_START	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、KEY_INJECTION_FINISH 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、KEY_INJECTION_FINISH からスタートします。
KEY_INJECTION_FINISH	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 6-4 デモプロジェクトのコマンド一覧

コマンド	動作
display	Wrapped Key / Plaintext Key　生成された鍵を表示します。
encdemo-encrypted [Arg1]	Arg1 の値を AES 128bit ECB モードで暗号化します(encrypted key を利用)。
encdemo-plaintext [Arg1]	Arg1 の値を AES 128bit ECB モードで暗号化します(plaintext key を利用)。
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。

デモプロジェクトでは、IV を”55aa55aa55aa55aa55aa55aa55aa55aa"、および以下の値を鍵のデフォルト値として使用しています。

表 6-5　デモプロジェクトの鍵値

key type	value
UFPK	22222222222222222222222222222221111111111111111111111111111111
AES-128bit	11111111222222223333333344444444
AES-256bit	ffffffffeeeeeeeeddddddddccccccccbbbbbbbbaaaaaaaa0000000099999999
HMAC SHA-224	000102030405060708090a0b0c0d0e0f10111213141516171819 1a1b
HMAC SHA-256	000102030405060708090a0b0c0d0e0f10111213141516171819 1a1b1c1d1e1f
ECC secp256r1 private	519b423d715f8b581f4fa8ee59f4771a5b44c8130b4e3eacca54a56dda72b464
ECC secp256r1 public	1ccbe91c075fc7f4f033bfa248db8fccd3565de94bbfb12f3c59ff46c271bf83 ce4014c68811f9a21a1fdb2c0e6113e06db7ca93b7404e78dc7ccd5ca89a4ca9
ECC BrainpoolP256 r1 private	01e98cf5a934e4f829f94a3fa4ec18ab421fbaa40e0da1f7a3cad1d7ac38cf4f
ECC BrainpoolP256 r1 public	5c568a2b6d3d9828734f9793507ce15629d37d357586e789faf07aeb69c50207 9a61f3fad6defedcb410a709055ca1a033422a9db29f554548438d6f5138d375
ECC secp256k1 private	ef871c07ce463c95384f1f388ef0107b655998623365eab0debdf9a94a7d2303
ECC secp256k1 public	1f9dfb391adade2c8fcbfef52e3115982e19c79ff0571c988a06aec223fb58a6 13898b7f7123287906f43ba6bfbf920aa49377e0ad400d3b8bd94dbb7245c025

function コマンドで使用する入力値を以下に示します。

表 6-6　function コマンドの入力値

data type	value
AES ECB/CBC/CTR/ CCM plaintext	aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaabbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb ccccccccccccccccccccccccccccccccdddddddddddddddddddddddddddddddd
AES CBC IV	12345678fedcba0955555555aaaaaaaa
AES CTR Counter	12345678fedcba0955555555aaaaaaaa
AES CCM Nonce	101112131415161718191a1b
AES CCM AAD	000102030405060708090a0b0c0d0e0f10111213141516171819 1a1b1c1d1e1f 202122232425262728292a2b2c2d2e2f303132333435363738393a3b3c3d3e3f 404142434445464748494a4b4c4d4e4f505152535455565758595a5b5c5d5e5f 6061626364656667686 96a6b
AES GCM plaintext	aaaaaaaabbbbbbbbccccccccdddddddd
AES GCM IV	b3d8cc017cbb89b39e0f67e2
AES GCM AAD	24825602bd12a984e0092d3e448eda5f
HMAC SHA-224/SHA-256 plaintext	53616D706C65206D65737361676520666F72206B65796C656E3C626C6F636 B6C656E
SHA-224/ SHA-256 plaintext	6162636462636465636465666465666765666768666768696768696a68696a6b 696a6b6c6a6b6c6d6b6c6d6e6c6d6e6f6d6e6f706e6f7071
ECC secp256r1/Brain poolP256r1/secp256k1 plaintext	5905238877c77421f73e43ee3da6f2d9e2ccad5fc942dcec0cbd25482935faaf 416983fe165b1a045ee2bcd2e6dca3bdf46c4310a7461f9a37960ca672d3feb5 473e253605fb1ddfd28065b53cb5858a8ad28175bf9bd386a5e471ea7a65c17c c934a9d791e91491eb3754d03799790fe2d308d16146d5c9b0d0debd97d79ce8

6.1.2.1 鍵とデモプロジェクトでの確認方法

デモプロジェクトでは、ユーザ鍵の Wrapped Key を生成しています。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 CM 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不可能にします。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起動時の鍵の注入が行われた場合の表示を図 6-3 に示します。これらのように表示された後、表 6-4 のコマンドを使用することができるようになります。

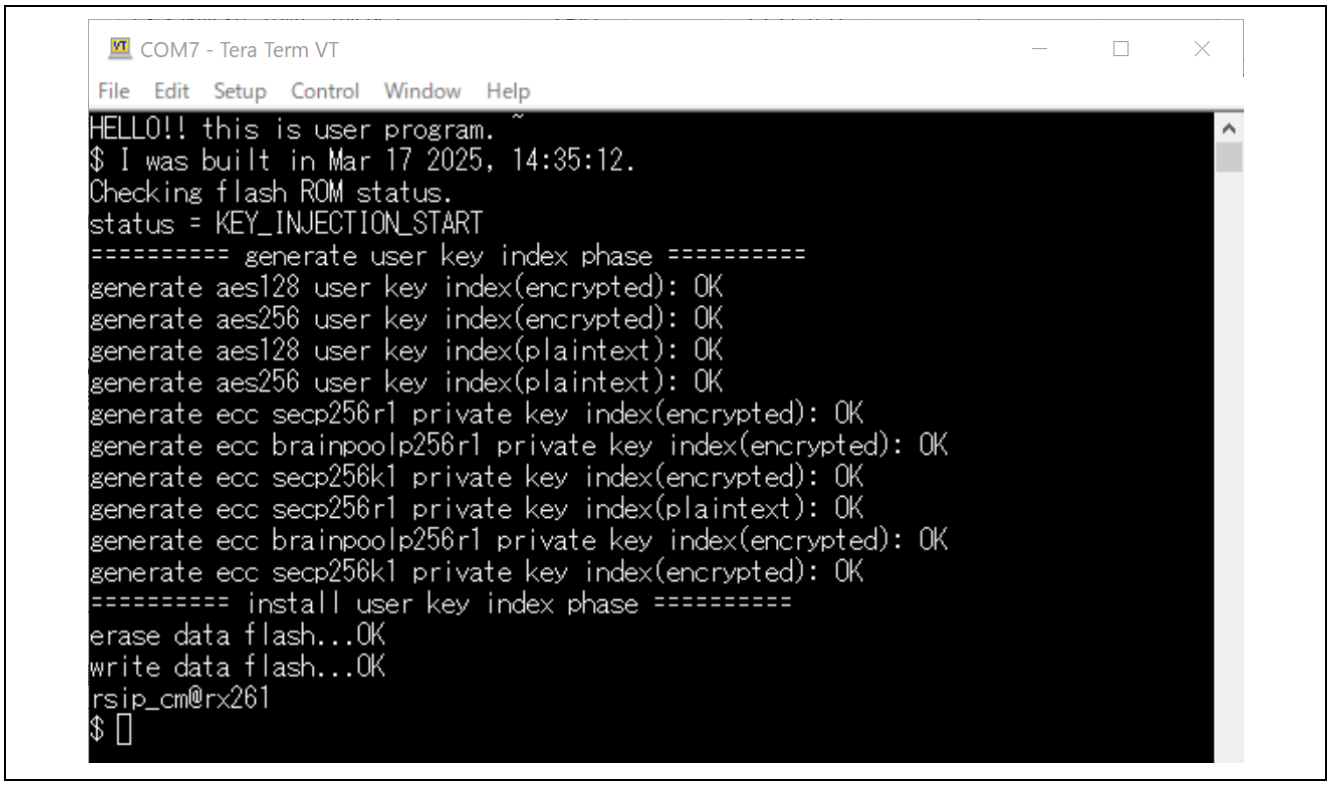

図 6-3 Tera Term の表示（起動時）

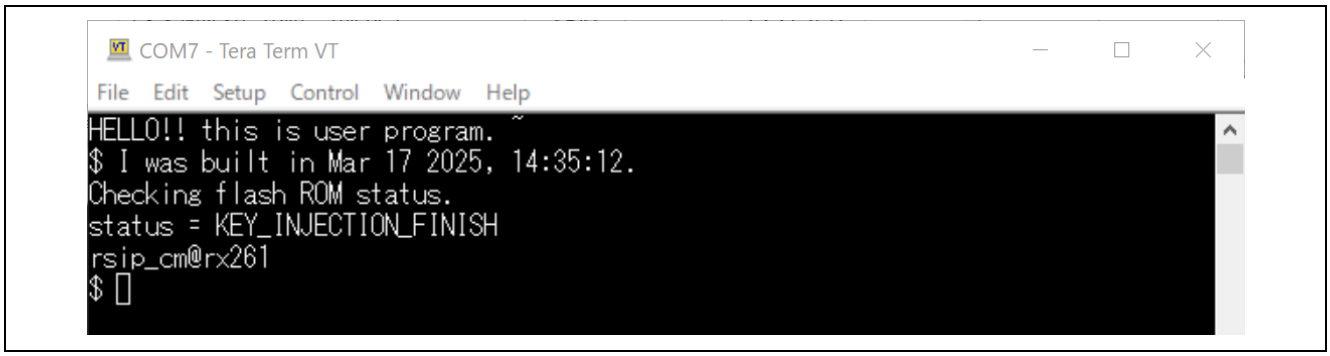

図 6-4 Tera Term の表示（2 回目以降の起動時）

コマンド使用例として、encdemo-encrypted と encdemo-plaintext コマンドの使用例を示します。

encdemo-encrypted コマンドでは、注入済みの AES128bit の encrypted wrapped 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。

encdemo-plaintext コマンドでは、注入済みの AES128bit の plaintext wrapped 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-encrypted 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-encrypted コマンド実行時）

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

図 6-6 Tera Term の表示（encdemo-plaintext コマンド実行時）

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 2025-01 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 8.3.0.202411 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	リトルエンディアン
モジュールのバージョン	Ver.1.00
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)」
- e² 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 e² 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 e²studio 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。
「e²studio から CS+への移行方法」
> 「既存のプロジェクトを変換して CS+の新規プロジェクトを作成」
https://www.renesas.com/jp/ja/products/software-tools/tools/migration-tools/migration-e2studio-to-csplus.html

【注意】：手順 5 で
「変換直後のプロジェクト構成ファイルをまとめてバックアップする(C)」
チェックが入っている場合に、[Q0268002]ダイアログが出る場合があります。
[Q0268002]ダイアログで [はい]ボタンを押した場合、コンパイラのインクルード・パスを設定しなおす必要があります。

7.3 ユーザ鍵フォーマット

本章では、ユーザ鍵のデータフォーマット Plain User Key と、Encrypted User Key のデータフォーマットを示します。

7.3.1 AES

AES 128bit 鍵

入力(Plaintext User Key)

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

入力(Encrypted User Key)

byte	16			
	4	4	4	4
0-15	encrypted_user_key(128bit AES 鍵)			
16-31	MAC			

AES 256bit 鍵

入力(Plaintext User Key)

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

入力(Encrypted User Key)

byte	16			
	4	4	4	4
0-31	encrypted_user_key(256bit AES 鍵)			
32-47	MAC			

7.3.2 ECC

ECC secp256r1/secp256k1/brainpoolP256r1 公開鍵

入力(Plaintext 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/secp256k1/brainpoolP256r1 秘密鍵

入力(Plaintext User Key)

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

入力(Encrypted User Key)

byte	16			
	4	4	4	4
0-31	encrypted_user_key(ECC P-256bit 秘密鍵 d)			
32-47	MAC			

7.3.3 HMAC

SHA224-HMAC 鍵
入力(Plaintext User Key)

byte	16			
	4	4	4	4
0-31	HMAC-SHA224 鍵			
				0 padding

SHA256-HMAC 鍵
入力(Plaintext User Key)

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

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	2025.4.25	-	初版発行

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。