ルネサスマイクロコンピュータ
AES ライブラリ: ユーザーズマニュアル

要旨

AES ライブラリは、ルネサスマイコンに対応したソフトウェアライブラリです。本資料は AES ライブラリの関数リファレンスを示します。対応マイコンによってサポートする AES 関数が異なります。マイコンに依存した情報については対応マイコン毎に用意している「導入ガイド」を合わせて参照してください。以下に示すドキュメントは、AES 暗号アルゴリズムに関するリファレンス仕様書です。

FIPS PUB 197, Announcing the ADVANCED ENCRYPTION STANDARD (AES)
http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf
NIST SP-800 38A, Recommendation for Block Cipher Modes of Operation
http://csrc.nist.gov/publications/nistpubs/800-38a/sp800-38a.pdf
NIST SP-800 38D, Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC
http://csrc.nist.gov/publications/nistpubs/800-38D/SP-800-38D.pdf

本資料は以下のライブラリについて記述しています

・AES ライブラリ V.1.06

・GCM ライブラリ V.1.02

動作確認デバイス

ルネサスマイクロコンピュータ

目次

1. 概要

1.1 AES アルゴリズム

AES アルゴリズムは、データに対して演算を行い変化させ(暗号化)、そのデータを元に戻す(復号)することができるブロック暗号です。また、この演算に用いられる入力値は暗号鍵と呼ばれ、暗号化でも復号でも同じ暗号鍵が使われるため、左右対称の暗号、共通鍵暗号と呼ばれます。暗号化は、平文(Plaintext)を暗号文(Ciphertext)と呼ばれるデータに変換します。復号は、暗号文を平文に変換します。

AES アルゴリズムが登場するまでは DES(*1)アルゴリズムが広く使われていましたが、DES アルゴリズムに代わり、AES アルゴリズムがより強固な暗号アルゴリズムとして NIST により公募されました。最終的に研究者 ヨアン・ダーメン (Joan Daemen) と フィンセント・ライメン (Vincent Rijmen) が設計した Rijndael （ラインダール）が AES アルゴリズムとして採用されています。

Feistel 構造と SPN(*2)構造は、ブロック暗号アルゴリズムの代表的な構造です。SPN 構造は、AES アルゴリズムで採用されました。SPN 構造において、置換と転置は繰り返し処理されます（ラウンド処理）。SPN 構造によるランダム化は、Feistel 構造での場合より効率的です。このことは、ラウンド処理が少なく済み、全体的な高速化を意味します。

(*1) DES(The Data Encryption Standard):
　暗号鍵使った左右対称のブロック暗号です。標準規格は、1977 年にアメリカ合衆国の暗号規格として採用されました。
(*2) SPN(The Substitution Permutation Network) :
　AES アルゴリズム用として採用されたブロック暗号構造の 1 種です。

AES アルゴリズムでは、一般的に 128、256 bit の暗号鍵がデータの暗号化/復号に使われます。

暗号/復号処理は内部で決められた演算を複数回(このうちの 1 回のことをラウンドと呼びます)行います。このそれぞれのラウンドで使うための鍵は拡大鍵と呼ばれ、暗号鍵から生成することができます。同じ鍵を使い続ける間は拡大鍵の生成処理は再度実行する必要はありません。

1.2 ブロック暗号モード

NIST SP 800-38A には、平文を暗号化して暗号文を生成するために、複数のモードが規定されています。AES ライブラリではこの内、ECB モードと CBC モードをサポートしています。

- ECB (Electronic CodeBook) Mode

ECB モードでは暗号鍵と平文が入力値となります。この場合、同じパターンが続く平文を入力するとブロック毎に必ず決まった暗号文が出力されます。

- CBC (Cipher Block Chaining) Mode

CBC モードでは暗号鍵と平文に加え、1 ブロック前に暗号化したデータブロックも入力値となります。暗号処理の前に平文と 1 ブロック前に暗号化したデータブロックを XOR した値を暗号鍵で暗号化します。最初のブロックでは 1 ブロック前に暗号化したデータブロックが無いので特別に「IVEC(初期化ベクタ)」と呼ばれるデータが使われます。これにより、同じパターンが続く平文を入力してもブロック毎に異なった暗号文が出力されます。

1.3 実装方式

AES 暗号には、鍵データに対し暗号処理前に拡大鍵を生成する方式と、拡大鍵を生成しながら暗号処理を行う方式の 2 通りの方式があります。後者を特に On-the-fly 実装と呼びます。

1.4　AES-GCM

Galois/Counter Mode(GCM)は共通鍵ブロック暗号を用いた暗号モードです。このモードはデータの信憑性と秘匿性を提供します。このモードを用いる暗号機能では、暗号データに対し認証タグと呼ばれるハッシュ値を生成します。また、このモードを用いる復号機能では、認証タグから暗号データが正しいものであるかどうかを検証します。また、GCM 暗号化関数に平文データを入力しない場合、出力される認証タグは Galois Message Authentication Code (GMAC)値となります。GCM 復号関数に暗号文データを入力しない場合、認証タグを検証することができます。また、暗号化、復号共に追加認証データを入力することができます。追加認証データは第 2 の鍵として入力することができます。追加認証データが不要な場合は入力を省略することができます。

GCM は共通鍵ブロック暗号として一般的に AES を用います。

1.5　ブロック暗号モードのイメージ

Electronic Code Book(ECB) モード

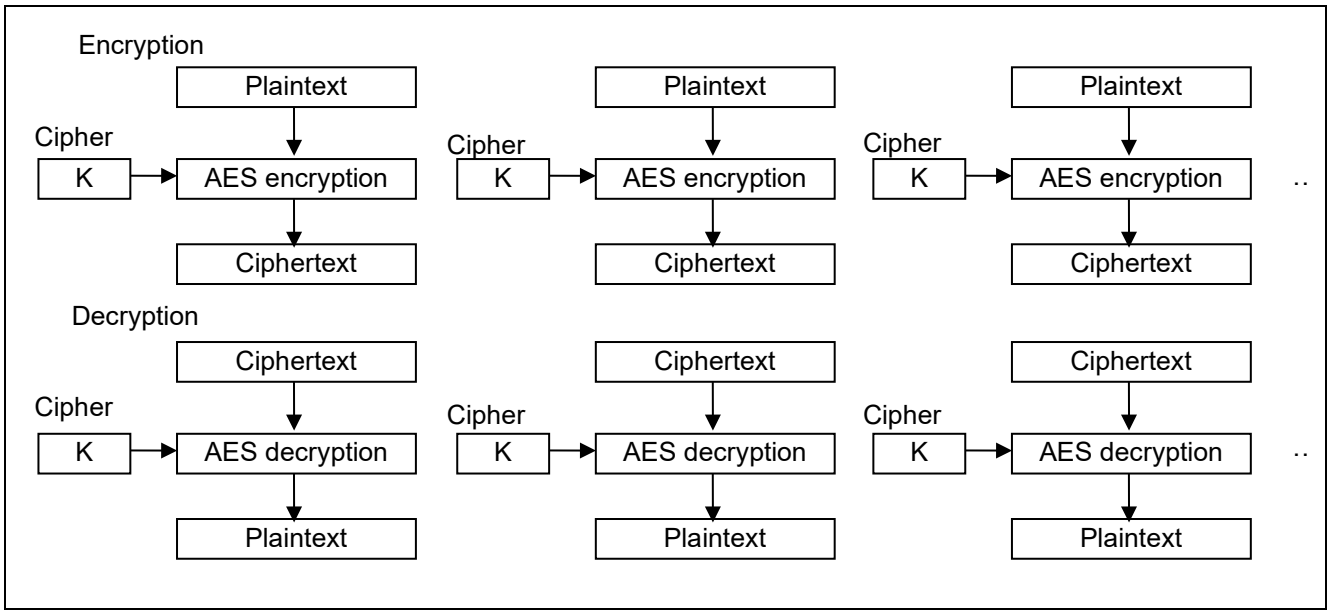

図 1　ECB モード

Cipher Block Chaining (CBC) モード

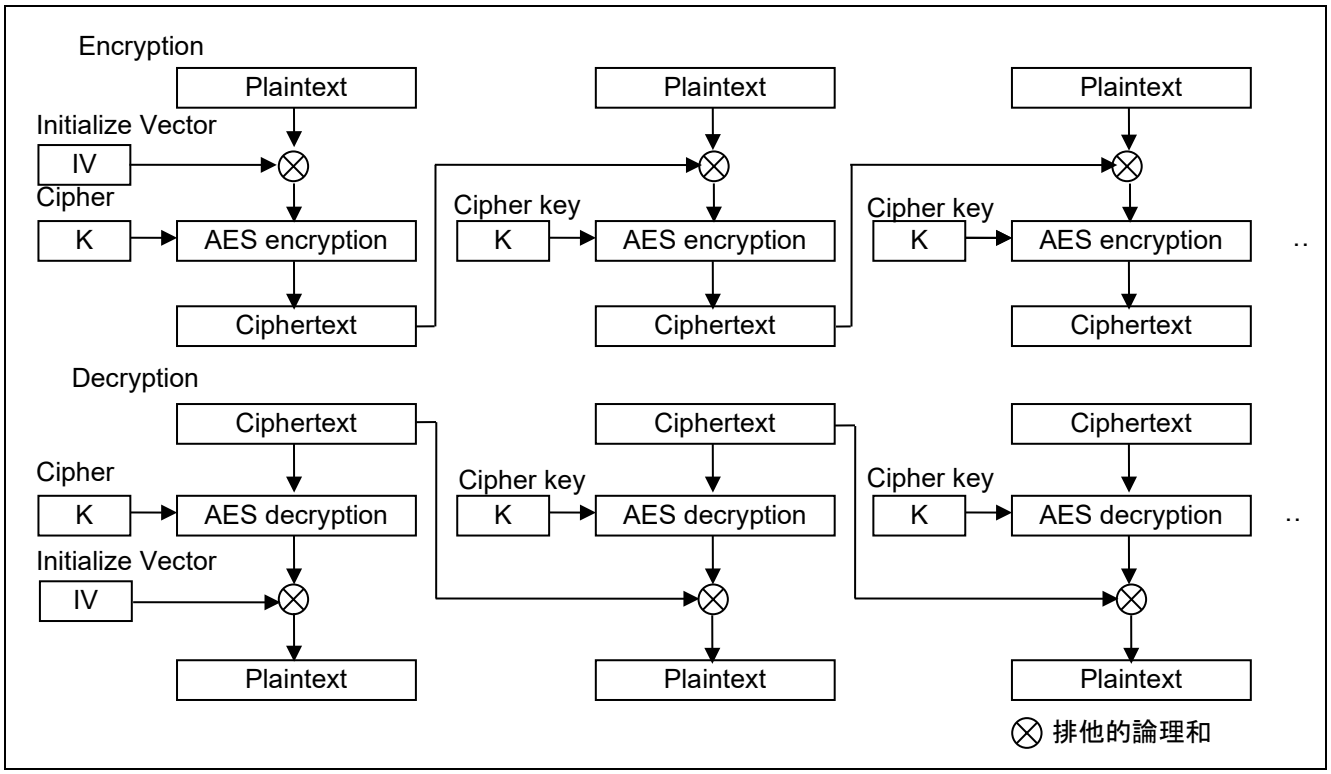

図 2　CBC モード

1.6　暗号データ検証方法

弊社では AES 暗号・復号および GCM 暗号・復号について以下情報をリファレンスとして使用しています。

AES 検証に用いたテストベクタ：

NIST (National Institute of Standards and Technology：アメリカの国立標準技術研究所)が公開している試験パターンです。

http://csrc.nist.gov/groups/STM/cavp/

上記のページで「AES Test Vectors」検索してください。

Test Vectors をダウンロードできます。

AES-GCM 検証に用いたテストベクタ： CAVS 14.0

NIST (National Institute of Standards and Technology：アメリカの国立標準技術研究所)が公開している試験パターンです。

http://csrc.nist.gov/groups/STM/cavp/

上記のページで「GCM Test Vectors」検索してください。

Test Vectors をダウンロードできます。

1.7　ソフトウェアスタック構成

AES ライブラリは以下のソフトウェアスタック構成になっています。GCM 関連の関数を使用する場合、GCM で使用するアルゴリズムの鍵設定関数と、ブロック暗号関数をユーザが実装してください。サンプルプログラムでは、ユーザ定義関数から AES 128-bit 拡大鍵生成関数と AES 128-bit 暗号化関数 (ECB モード)を呼び出しています。

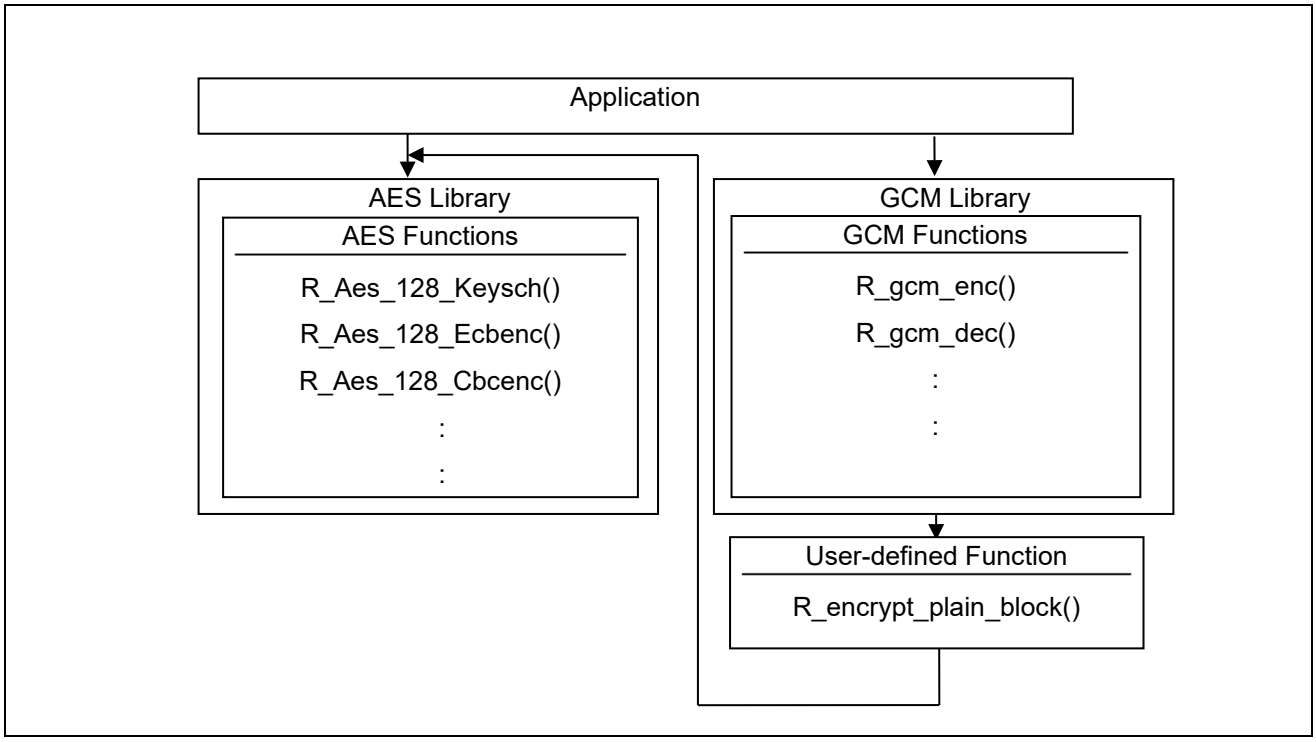

図 3　ソフトウェア構成

1.8　スタックの取り扱い

AES ライブラリと GCM ライブラリは、演算を行うときスタック領域に演算途中のデータを一時的に保存します。演算途中のデータをそのまま残すと、その情報から鍵情報などが推測される可能性があるため、ライブラリの関数から抜けるとき使用したローカル変数をクリアします。

クリアに要する時間は、暗号の演算に比べて極僅か(1%未満)です。

1.9　メモリ配置について

RL78 ファミリなど一部の機種用の AES ライブラリではメモリ配置に制限があります。

処理速度の関係でデータポインタを 16bit 長で扱うため、AES ライブラリが扱うデータは near 領域に配置されている必要があります。ライブラリ関数のポインタ型の引数には"near"や"__near"の修飾子が必要になりますが、本ドキュメントの関数リファレンスには記載しません。

詳細は各機種の導入ガイドを参照ください。

2. AES ライブラリ仕様

2.1 データタイプ

ここでは、ヘッダファイル”r_stdint.h”で定義されているデータタイプのうち、AES ライブラリが使用するものをリストにまとめます。

表 1 使用するデータタイプリスト

Datatype	Typedef	Number of bytes
unsigned char	uint8_t	1
unsigned short	uint16_t	2
unsigned long	uint32_t	4
signed char	int8_t	1
signed short	int16_t	2
signed long	int32_t	4

2.2 AES 関数リファレンス

2.2.1 AES 128-bit 拡大鍵生成関数

Usage

#include “r_aes.h”

void R_Aes_128_Keysch (uint8_t *key, uint32_t *ekey);

Parameters

key	入力	鍵データ領域 (16 byte)
ekey	出力	拡大鍵領域 (176 byte)

Return Value

無し

Description

R_Aes_128_Keysch() 関数は、第一引数”key”で指定されたデータから拡大鍵を第二引数で指定された”ekey”に書き出します。この処理のために、176 byte のメモリが ekey で指定した領域に必要です。

Remark

不正なポインタ(ex. Null Pointer)を指定した場合の動作は不定です。

2.2.2　AES 128-bit 暗号化関数 (ECB モード)

Usage

#include “r_aes.h”

void R_Aes_128_Ecbenc (uint8_t *pdat, uint8_t *cdat, uint32_t *ekey, uint16_t block);

Parameters

pdat	入力	平文データ領域 (block * 16 byte)
cdat	出力	暗号文データ領域 (block * 16 byte)
ekey	入力	拡大鍵領域 (176 byte)
block	入力	暗号化するブロック数 (1～任意 block)

Return Value

無し

Description

R_Aes_128_Ecbenc() 関数は、第一引数”pdat”で指定された平文から第三引数で指定された”ekey”を用いて ECB モードで暗号化します。暗号化データは第二引数で指定された”cdat”に出力します。暗号化するブロック数は第四引数の”block”で指定できます。1 ブロックは 16byte です。

Remark

不正なポインタ(ex. Null Pointer)を指定した場合の動作は不定です。

平文データ領域と暗号文データ領域に同一領域を指定可能です。

(ただし、完全に同じ領域でなく領域がオーバラップしている場合の動作は不定です。)

ekey に、R_Aes_128_Keysch() 関数で生成したデータ以外を指定した場合の動作は不定です。

2.2.3　AES 128-bit 復号関数 (ECB モード)

Usage

#include “r_aes.h”

void R_Aes_128_Ecbdec (uint8_t *cdat, uint8_t *pdat, uint32_t *ekey, uint16_t block);

Parameters

cdat	入力	暗号文データ領域 (block * 16 byte)
pdat	出力	平文データ領域 (block * 16 byte)
ekey	入力	拡大鍵領域 (176 byte)
block	入力	暗号化するブロック数 (1～任意 block)

Return Value

無し

Description

R_Aes_128_Ecbdec() 関数は、第一引数”cdat”で指定された暗号文から第三引数で指定された”ekey”を用いて ECB モードで復号します。平文データは第二引数で指定された”pdat”に出力します。復号するブロック数は第四引数の”block”で指定できます。1 ブロックは 16byte です。

Remark

不正なポインタ(ex. Null Pointer)を指定した場合の動作は不定です。

平文データ領域と暗号文データ領域に同一領域を指定可能です。

(ただし、完全に同じ領域でなく領域がオーバラップしている場合の動作は不定です。)

ekey に、R_Aes_128_Keysch() 関数で生成したデータ以外を指定した場合の動作は不定です。

2.2.4　AES 128-bit 暗号化関数 (CBC モード)

Usage

#include “r_aes.h”

void R_Aes_128_Cbcenc (uint8_t *pdat, uint8_t *cdat, uint8_t *ivec, uint32_t *ekey, uint16_t block);

Parameters

pdat	入力	平文データ領域 (block * 16 byte)
cdat	出力	暗号文データ領域 (block * 16 byte)
ivec	入力/出力	初期化ベクタ (16 byte)
ekey	入力	拡大鍵領域 (176 byte)
block	入力	暗号化するブロック数 (1～任意 block)

Return Value

無し

Description

R_Aes_128_Cbcenc() 関数は、第一引数”pdat”で指定された平文から第三引数で指定された”ivec”と第四引数で指定された”ekey”を用いて CBC モードで暗号化します。暗号化データは第二引数で指定された”cdat”に出力します。暗号化するブロック数は第五引数の”block”で指定できます。1 ブロックは 16byte です。"ivec"の領域は 16 byte のワークエリアとして使用され、演算後の"ivec"には暗号文の最終ブロックと同じ値が出力されます。

Remark

不正なポインタ(ex. Null Pointer)を指定した場合の動作は不定です。

平文データ領域と暗号文データ領域に同一領域を指定可能です。

(ただし、完全に同じ領域でなく領域がオーバラップしている場合の動作は不定です。)

ekey に、R_Aes_128_Keysch() 関数で生成したデータ以外を指定した場合の動作は不定です。

2.2.5　AES 128-bit 復号関数 (CBC モード)

Usage

#include “r_aes.h”

void R_Aes_128_Cbcdec (uint8_t *cdat, uint8_t *pdat, uint8_t *ivec, uint32_t *ekey, uint16_t block);

Parameters

cdat	入力	暗号文データ領域 (block * 16 byte)
pdat	出力	平文データ領域 (block * 16 byte)
ivec	入力/出力	初期化ベクタ (16 byte)
ekey	入力	拡大鍵領域 (176 byte)
block	入力	暗号化するブロック数 (1～任意 block)

Return Value

無し

Description

R_Aes_128_Cbcdec() 関数は、第一引数”cdat”で指定された暗号文から第三引数で指定された”ivec”と第四引数で指定された”ekey”を用いて CBC モードで復号します。平文データは第二引数で指定された”pdat”に出力します。復号するブロック数は第五引数の”block”で指定できます。1 ブロックは 16byte です。"ivec"の領域は 16 byte のワークエリアとして使用し、演算後の"ivec"には平文の最終ブロックと同じ値が出力されます。

Remark

不正なポインタ(ex. Null Pointer)を指定した場合の動作は不定です。

平文データ領域と暗号文データ領域に同一領域を指定可能です。

(ただし、完全に同じ領域でなく領域がオーバラップしている場合の動作は不定です。)

ekey に、R_Aes_128_Keysch() 関数で生成したデータ以外を指定した場合の動作は不定です。

2.2.6 AES 256-bit 拡大鍵生成関数

Usage

#include “r_aes.h”

void R_Aes_256_Keysch (uint8_t *key, uint32_t *ekey);

Parameters

key	入力	鍵データ領域 (32 byte)
ekey	出力	拡大鍵領域 (240 byte)

Return Value

無し

Description

R_Aes_256_Keysch() 関数は、第一引数”key”で指定されたデータから拡大鍵を第二引数で指定された”ekey”に書き出します。この処理のために、240 byte のメモリが ekey で指定した領域に必要です。

Remark

不正なポインタ(ex. Null Pointer)を指定した場合の動作は不定です。

2.2.7　AES 256-bit 暗号化関数 (ECB モード)

Usage

```
#include “r_aes.h”
void R_Aes_256_Ecbenc (uint8_t *pdat, uint8_t *cdat, uint32_t *ekey, uint16_t block);
```

Parameters

pdat	入力	平文データ領域 (block * 16 byte)
cdat	出力	暗号文データ領域 (block * 16 byte)
ekey	入力	拡大鍵領域 (240 byte)
block	入力	暗号化するブロック数 (1～任意 block)

Return Value

無し

Description

R_Aes_256_Ecbenc() 関数は、第一引数”pdat”で指定された平文から第三引数で指定された”ekey”を用いて ECB モードで暗号化します。暗号化データは第二引数で指定された”cdat”に出力します。暗号化するブロック数は第四引数の”block”で指定できます。1 ブロックは 16byte です。

Remark

不正なポインタ(ex. Null Pointer)を指定した場合の動作は不定です。

平文データ領域と暗号文データ領域に同一領域を指定可能です。

(ただし、完全に同じ領域でなく領域がオーバラップしている場合の動作は不定です。)

ekey に、R_Aes_256_Keysch() 関数で生成したデータ以外を指定した場合の動作は不定です。

2.2.8　AES 256-bit 復号関数 (ECB モード)

Usage

#include “r_aes.h”

void R_Aes_256_Ecbdec (uint8_t *cdat, uint8_t *pdat, uint32_t *ekey, uint16_t block);

Parameters

cdat	入力	暗号文データ領域 (block * 16 byte)
pdat	出力	平文データ領域 (block * 16 byte)
ekey	入力	拡大鍵領域 (240 byte)
block	入力	暗号化するブロック数 (1～任意 block)

Return Value

無し

Description

R_Aes_256_Ecbdec() 関数は、第一引数”cdat”で指定された暗号文から第三引数で指定された”ekey”を用いて ECB モードで復号します。平文データは第二引数で指定された”pdat”に出力します。復号するブロック数は第四引数の”block”で指定できます。1 ブロックは 16byte です。

Remark

不正なポインタ(ex. Null Pointer)を指定した場合の動作は不定です。

平文データ領域と暗号文データ領域に同一領域を指定可能です。

(ただし、完全に同じ領域でなく領域がオーバラップしている場合の動作は不定です。)

ekey に、R_Aes_256_Keysch() 関数で生成したデータ以外を指定した場合の動作は不定です。

2.2.9　AES 256-bit 暗号化関数 (CBC モード)

Usage

#include “r_aes.h”

void R_Aes_256_Cbcenc (uint8_t *pdat, uint8_t *cdat, uint8_t *ivec, uint32_t *ekey, uint16_t block);

Parameters

pdat	入力	平文データ領域 (block * 16 byte)
cdat	出力	暗号文データ領域 (block * 16 byte)
ivec	入力/出力	初期化ベクタ (16 byte)
ekey	入力	拡大鍵領域 (240 byte)
block	入力	暗号化するブロック数 (1～任意 block)

Return Value

無し

Description

R_Aes_256_Cbcenc() 関数は、第一引数”pdat”で指定された平文から第三引数で指定された”ivec”と第四引数で指定された”ekey”を用いて CBC モードで暗号化します。暗号化データは第二引数で指定された”cdat”に出力します。暗号化するブロック数は第五引数の”block”で指定できます。1 ブロックは 16byte です。"ivec"の領域は 16 byte のワークエリアとして使用し、演算後の"ivec"には暗号文の最終ブロックと同じ値が出力されます。

Remark

不正なポインタ(ex. Null Pointer)を指定した場合の動作は不定です。

平文データ領域と暗号文データ領域に同一領域を指定可能です。

(ただし、完全に同じ領域でなく領域がオーバラップしている場合の動作は不定です。)

ekey に、R_Aes_256_Keysch() 関数で生成したデータ以外を指定した場合の動作は不定です。

2.2.10 AES 256-bit 復号関数 (CBC モード)

Usage

#include “r_aes.h”

void R_Aes_256_Cbcdec (uint8_t *cdat, uint8_t *pdat, uint8_t *ivec, uint32_t *ekey, uint16_t block);

Parameters

cdat	入力	暗号文データ領域 (block * 16 byte)
pdat	出力	平文データ領域 (block * 16 byte)
ivec	入力/出力	初期化ベクタ (16 byte)
ekey	入力	拡大鍵領域 (240 byte)
block	入力	暗号化するブロック数 (1～任意 block)

Return Value

無し

Description

R_Aes_256_Cbcdec() 関数は、第一引数”cdat”で指定された暗号文から第三引数で指定された”ivec”と第四引数で指定された”ekey”を用いて CBC モードで復号します。平文データは第二引数で指定された”pdat”に出力します。復号するブロック数は第五引数の”block”で指定できます。1 ブロックは 16byte です。"ivec"の領域は 16 byte のワークエリアとして使用し、演算後の"ivec"には平文の最終ブロックと同じ値が出力されます。

Remark

不正なポインタ(ex. Null Pointer)を指定した場合の動作は不定です。

平文データ領域と暗号文データ領域に同一領域を指定可能です。

(ただし、完全に同じ領域でなく領域がオーバラップしている場合の動作は不定です。)

ekey に、R_Aes_256_Keysch() 関数で生成したデータ以外を指定した場合の動作は不定です。

2.2.11 AES 128-bit on-the-fly 暗号化関数 (ECB モード)

Usage

#include “r_aes.h”

void R_Aes_128_OtfEcbenc (uint8_t *ptext, uint8_t *ctext, uint16_t *key, uint16_t block);

Parameters

ptext	入力	平文データ領域 (block * 16 byte)
ctext	出力	暗号文データ領域 (block * 16 byte)
key	入力	鍵領域 (16 byte)
block	入力	暗号化するブロック数 (1～任意 block)

Return Value

無し

Description

R_Aes_128_OtfEcbenc() 関数は、第一引数”ptext”で指定された平文から第三引数で指定された”key”を用いて ECB モードで暗号化します。暗号化データは第二引数で指定された”ctext”に出力します。暗号化するブロック数は第四引数の”block”で指定できます。1 ブロックは 16byte です。

Remark

不正なポインタ(ex. Null Pointer)を指定した場合の動作は不定です。

平文データ領域と暗号文データ領域に同一領域を指定可能です。

(ただし、完全に同じ領域でなく領域がオーバラップしている場合の動作は不定です。)

2.2.12 AES 128-bit on-the-fly 復号関数 (ECB モード)

Usage

#include “r_aes.h”

void R_Aes_128_OtfEcbdec (uint8_t *ctext, uint8_t *ptext, uint16_t *key, uint16_t block);

Parameters

ctext	入力	暗号文データ領域 (block * 16 byte)
ptext	出力	平文データ領域 (block * 16 byte)
key	入力	鍵領域 (16 byte)
block	入力	暗号化するブロック数 (1～任意 block)

Return Value

無し

Description

R_Aes_128_OtfEcbdec() 関数は、第一引数”ctext”で指定された暗号文から第三引数で指定された”key”を用いて ECB モードで復号します。平文データは第二引数で指定された”ptext”に出力します。復号するブロック数は第四引数の”block”で指定できます。1 ブロックは 16byte です。

Remark

不正なポインタ(ex. Null Pointer)を指定した場合の動作は不定です。

平文データ領域と暗号文データ領域に同一領域を指定可能です。

(ただし、完全に同じ領域でなく領域がオーバラップしている場合の動作は不定です。)

2.2.13 AES 128-bit on-the-fly 暗号化関数 (CBC モード)

Usage

#include “r_aes.h”

void R_Aes_128_OtfCbcenc (uint8_t *ptext, uint8_t *ctext, uint8_t *ivec, uint16_t *key, uint16_t block);

Parameters

ptext	入力	平文データ領域 (block * 16 byte)
ctext	出力	暗号文データ領域 (block * 16 byte)
ivec	入力/出力	初期化ベクタ (16 byte)
key	入力	鍵領域 (16 byte)
block	入力	暗号化するブロック数 (1～任意 block)

Return Value

無し

Description

R_Aes_128_OtfCbcenc () 関数は、第一引数”ptext”で指定された平文から第三引数で指定された”ivec”と第四引数で指定された”key”を用いて CBC モードで暗号化します。暗号化データは第二引数で指定された”ctext”に出力します。暗号化するブロック数は第五引数の”block”で指定できます。1 ブロックは 16byte です。"ivec"の領域は 16 byte のワークエリアとして使用し、演算後の"ivec"には暗号文の最終ブロックと同じ値が出力されます。

Remark

不正なポインタ(ex. Null Pointer)を指定した場合の動作は不定です。

平文データ領域と暗号文データ領域に同一領域を指定可能です。

(ただし、完全に同じ領域でなく領域がオーバラップしている場合の動作は不定です。)

2.2.14 AES 128-bit on-the-fly 復号関数 (CBC モード)

Usage

#include “r_aes.h”

void R_Aes_128_OtfCbcdec (uint8_t *ctext, uint8_t *ptext, uint8_t *ivec, uint16_t *key, uint16_t block);

Parameters

ctext	入力	暗号文データ領域 (block * 16 byte)
ptext	出力	平文データ領域 (block * 16 byte)
ivec	入力/出力	初期化ベクタ (16 byte)
key	入力	鍵領域 (16 byte)
block	入力	暗号化するブロック数 (1～任意 block)

Return Value

無し

Description

R_Aes_128_OtfCbcdec() 関数は、第一引数”ctext”で指定された暗号文から第三引数で指定された”ivec”と第四引数で指定された”key”を用いて CBC モードで復号します。平文データは第二引数で指定された”ptext”に出力します。復号するブロック数は第五引数の”block”で指定できます。1 ブロックは 16byte です。"ivec"の領域は 16 byte のワークエリアとして使用し、演算後の"ivec"には平文の最終ブロックと同じ値が出力されます。

Remark

不正なポインタ(ex. Null Pointer)を指定した場合の動作は不定です。

平文データ領域と暗号文データ領域に同一領域を指定可能です。

(ただし、完全に同じ領域でなく領域がオーバラップしている場合の動作は不定です。)

2.2.15 AES テーブルデータ転送関数 (M16C のみ)

Usage

```
#include “r_aes.h”
void R_Aes_Table_Init(void);
```

Parameters

無し

Return Value

無し

Description

M16C の AES ライブラリはテーブルデータを RAM 領域に転送する必要があります。暗号化/復号処理を行う前に、本関数を呼び出してください。ROM 領域の_DATA_TBL_ROM から RAM 領域の _DATA_TBL_NEAR にテーブルデータを転送します。

Remark

無し

3. GCM ライブラリ仕様

3.1 データタイプ

GCM ライブラリは、ヘッダファイル”r_stdint.h”で定義されているデータタイプを使用します。AES ライブラリのデータタイプと同様です。「2.1 データタイプ」を参照してください。

3.2　GCM 関数リファレンス(標準仕様)

3.2.1　GCM 暗号化関数

Usage

```
#include “r_gcm.h”
int32_t R_gcm_enc(uint8_t *plain, uint8_t *cipher, uint32_t data_len, uint8_t *key,
                  uint8_t *atag, uint32_t atag_len, uint8_t *iv, uint32_t iv_len,
                  uint8_t *aad, uint32_t aad_len);
```

Parameters

plain	入力	平文データ領域 (data_len byte)
cipher	出力	暗号文データ領域 (data_len byte)
data_len	入力	平文データ長 (0～任意 byte)
key	入力	鍵領域(16 byte)
atag	出力	認証タグ領域 (atag_len byte)
atag_len	入力	認証タグ長 (1～16 byte)
iv	入力	初期化ベクタ領域 (iv_len byte)
iv_len	入力	初期化ベクタ長 (1～任意 byte)
aad	入力	追加認証データ (aad_len byte)
aad_len	入力	追加認証データ長 (0～任意 byte)

Return Value

0	正常終了
-1	異常終了 (atag_len=0, iv_len=0)

Description

R_gcm_enc() 関数は、第一引数”plain”で指定された平文から第四引数で指定された”key”と第七引数で指定された”iv”と第九引数で指定された”aad”を用いて GCM で暗号化します。暗号データは第二引数で指定された”cipher”に出力します。認証タグは第五引数で指定された”atag”に出力します。暗号化するデータ長は第三引数の”data_len”で指定できます。

Remark

入力値の plain 及び aad は 16byte で割り切れない場合、パディング処理は関数内部で実施します。

鍵のサイズは組み合わせるブロック暗号に依存します。

不正なポインタ(ex. Null Pointer)、奇数アドレスを指定した場合の動作は不定です。

data_len = 0 の場合は、plain と cipher を使用しません。この場合出力される認証タグは GMAC 値と呼ばれます。aad_len = 0 の場合は、aad を使用しません。

平文データ領域と暗号文データ領域に同一領域を指定可能です。

(ただし、完全に同じ領域でなく領域がオーバラップしている場合の動作は不定です。)

3.2.2　GCM 復号関数

Usage

```
#include “r_gcm.h”
int32_t R_gcm_dec(uint8_t *cipher, uint8_t *plain, uint32_t data_len, uint8_t *key,
                  uint8_t *atag, uint32_t atag_len, uint8_t *iv, uint32_t iv_len,
                  uint8_t *aad, uint32_t aad_len);
```

Parameters

cipher	入力	暗号文データ領域 (data_len byte)
plain	出力	平文データ領域 (data_len byte)
data_len	入力	暗号文データ長 (0～任意 byte)
key	入力	鍵領域(16 byte)
atag	入力	認証タグ領域 (atag_len byte)
atag_len	入力	認証タグ長 (1～16 byte)
iv	入力	初期化ベクタ領域 (iv_len byte)
iv_len	入力	初期化ベクタ長 (1～任意 byte)
aad	入力	追加認証データ (aad_len byte)
aad_len	入力	追加認証データ長 (0～任意 byte)

Return Value

0	認証成功
-1	認証失敗または、異常終了 (atag_len=0, iv_len=0)

Description

R_gcm_dec() 関数は、第一引数”cipher”で指定された暗号文から第四引数で指定された”key”と第五引数で指定された”atag”と第七引数で指定された”iv”と第九引数で指定された”aad”を用いて GCM で復号、及び認証タグの検証をします。認証タグの検証結果が OK の場合、平文データは第二引数で指定された”plain”に出力します。認証タグの検証結果が NG の場合、”plain”には 0 が出力されます。復号するデータ長は第三引数の”data_len”で指定できます。

Remark

入力値の plain 及び aad は 16byte で割り切れない場合、パディング処理は関数内部で実施します。

鍵のサイズは組み合わせるブロック暗号に依存します。

不正なポインタ(ex. Null Pointer)、奇数アドレスを指定した場合の動作は不定です。

data_len = 0 の場合は、plain と cipher を使用しません。この場合入力される認証タグは GMAC 値として検証されます。aad_len = 0 の場合は、aad を使用しません。

平文データ領域と暗号文データ領域に同一領域を指定可能です。

(ただし、完全に同じ領域でなく領域がオーバラップしている場合の動作は不定です。)

3.3　GCM 関数リファレンス(オプション仕様)

GCM 関数リファレンス(オプション仕様)の関数群は、組み込みシステムに適した処理をするための関数群です。

3.2.1 GCM 暗号化関数や、3.2.2 GCM 復号関数は以下デメリットがあります。

すべての引数を一度に入力するため関数呼び出し 1 回あたりの呼び出し時間がかかります。

平文/暗号文のデータ長で指定したすべてのデータの暗号化/復号が完了するまで処理が継続します。

3.3.1 GCM 暗号化開始関数、3.3.2 GCM 復号開始関数、3.3.3 GCM 処理継続関数を使用することでこのデメリットを解消できます。

GCM 暗号化開始関数または、GCM 復号開始関数を呼び出すと、GCM 処理継続関数を呼び出す必要のある回数を戻り値で返します。ユーザはこの戻り値の回数分 GCM 処理継続関数を呼び出してください。

GCM 暗号化の内部処理は STEP1～STEP7、GCM 復号の内部処理は STEP1～STEP8 まであります。GCM 関数リファレンス(オプション仕様)の関数群は指定された平文ブロック数(n)に従い GCM 暗号処理、GCM 復号処理を以下のように分割します。説明のため、分割ブロック数は 1 とします。

GCM 暗号化処理：

	n=0	n=1	n=2	n=3	n=4 …	n=100 …
STEP1～STEP2	1	1	1	1	1	1
STEP3(※)	2	2	2,3	2,3,4	2,3,4,5	2,…101
STEP4	3	3	4	5	6	102
STEP5(※)	4	4	5,6	6,7,8	7,8,9,10	103,…202
STEP6～STEP7	5	5	7	9	11	203

GCM 復号処理：

	n=0	n=1	n=2	n=3	n=4 …	n=100 …
STEP1～STEP3	1	1	1	1	1	1
STEP5	2	2	2	2	2	2
STEP6(※)	3	3	3,4	3,4,5	3,4,5,6	3,…102
STEP4(※)	4	4	5,6	6,7,8	7,8,9,10	103,…202
STEP7～STEP8	5	5	7	9	11	203

(処理の都合上、順序を入れ替えています)

処理時間が平文ブロック数に比例する個所(※)について、分割ブロック数を用いて何ブロック単位で処理するかを指定することができます。

平文ブロック数 100 を分割ブロック数 8 指定で GCM 暗号化する場合、分割ブロック数が 1 の場合 100 分割されるところ、13 分割(100/8 の 12 回と余りの 1 回分)となります。

従って、GCM 暗号化開始関数は 29(1+13+1+13+1)を戻り値とし、ユーザは GCM 処理継続関数を 29 回呼び出すことで全 100 ブロックの暗号化を完了することができます。

以下に全 100 ブロックのデータを 8 ブロック単位で暗号化する際の処理フローを例示します。

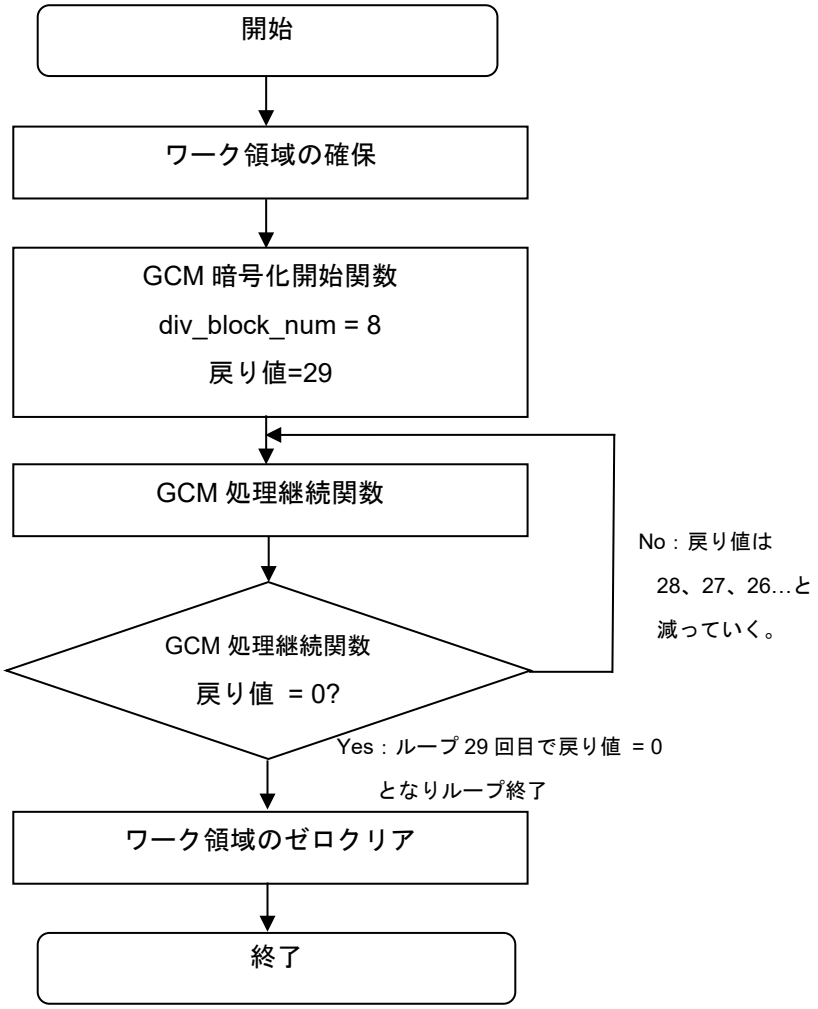

図 4　処理フロー

3.3.1　GCM 暗号化開始関数

Usage

```
#include “r_gcm.h”
int32_t R_gcm_enc_start(uint8_t *plain, uint8_t *cipher, uint32_t data_len, uint8_t *key,
                        uint8_t *atag, uint32_t atag_len, uint8_t *iv, uint32_t iv_len,
                        uint8_t *aad, uint32_t aad_len,
                        uint32_t div_block_num, uint32_t *work);
```

Parameters

plain	入力	平文データ領域 (data_len byte)
cipher	出力	暗号文データ領域 (data_len byte)
data_len	入力	平文データ長 (0～任意 byte)
key	入力	鍵領域(16 byte)
atag	出力	認証タグ領域 (atag_len byte)
atag_len	入力	認証タグ長 (1～16 byte)
iv	入力	初期化ベクタ領域 (iv_len byte)
iv_len	入力	初期化ベクタ長 (1～任意 byte)
aad	入力	追加認証データ (aad_len byte)
aad_len	入力	追加認証データ長 (0～任意 byte)
div_block_num	入力	分割ブロック数(1～任意 block)
work	出力	分割処理管理用ワーク領域(156 byte 使用します)

Description

R_gcm_enc_start() 関数は、第一引数”plain”で指定された平文から第四引数で指定された”key”と第七引数で指定された”iv”と第九引数で指定された”aad”を用いて GCM で暗号化します。暗号データは第二引数で指定された”cipher”に出力します。認証タグは第五引数で指定された”atag”に出力します。暗号化するデータ長は第三引数の”data_len”で指定できます。

本関数は各引数を第十二引数”work”に記録します。暗号処理は行いません。暗号処理を実行するためには 3.3.3 GCM 処理継続関数を繰り返し呼び出してください。3.3.3 GCM 処理継続関数は第十一引数” div_block_num”で指定した単位で平文を読み込み、暗号処理を実行します。1 ブロックは 16byte です。

Return Value

正数	正常終了、全ブロック暗号化するために必要な GCM 処理継続関数呼び出し回数
-1	異常終了 (atag_len=0, iv_len=0, div_block_num=0)

Remark

入力値の plain 及び aad は 16byte で割り切れない場合、パディング処理は関数内部で実施します。

鍵のサイズは組み合わせるブロック暗号に依存します。

不正なポインタ(ex. Null Pointer)、奇数アドレスを指定した場合の動作は不定です。

data_len = 0 の場合は、plain と cipher を使用しません。この場合出力される認証タグは GMAC 値と呼ばれます。aad_len = 0 の場合は、aad を使用しません。

平文データ領域と暗号文データ領域に同一領域を指定可能です。

(ただし、完全に同じ領域でなく領域がオーバラップしている場合の動作は不定です。)

work は演算途中のデータ等が格納されます。演算完了後、ユーザがゼロクリアしてください。

3.3.2　GCM 復号開始関数

Usage

```
#include “r_gcm.h”
int32_t R_gcm_dec_start (uint8_t *cipher, uint8_t *plain, uint32_t data_len, uint8_t *key,
                         uint8_t *atag, uint32_t atag_len, uint8_t *iv, uint32_t iv_len,
                         uint8_t *aad, uint32_t aad_len,
                         uint32_t div_block_num, uint32_t *work);
```

Parameters

cipher	入力	暗号文データ領域 (data_len byte)
plain	出力	平文データ領域 (data_len byte)
data_len	入力	暗号文データ長 (0～任意 byte)
key	入力	鍵領域(16 byte)
atag	入力	認証タグ領域 (atag_len byte)
atag_len	入力	認証タグ長 (1～16 byte)
iv	入力	初期化ベクタ領域 (iv_len byte)
iv_len	入力	初期化ベクタ長 (1～任意 byte)
aad	入力	追加認証データ (aad_len byte)
aad_len	入力	追加認証データ長 (0～任意 byte)
div_block_num	入力	分割ブロック数(1～任意 block)
work	出力	分割処理管理用ワーク領域(156 byte 使用します)

Description

R_gcm_dec_start() 関数は、第一引数”cipher”で指定された暗号文から第四引数で指定された”key”と第五引数で指定された”atag”と第七引数で指定された”iv”と第九引数で指定された”aad”を用いて GCM で復号、及び認証タグの検証をします。認証タグの検証結果が OK の場合、平文データは第二引数で指定された”plain”に出力します。認証タグの検証結果が NG の場合、”plain”には 0 が出力されます。復号するデータ長は第三引数の”data_len”で指定できます。

本関数は各引数を第十二引数”work”に記録します。復号処理は行いません。復号処理を実行するためには 3.3.3 GCM 処理継続関数を繰り返し呼び出してください。3.3.3 GCM 処理継続関数は第十一引数” div_block_num”で指定した単位で暗号文を読み込み、復号処理を実行します。1 ブロックは 16byte です。

Return Value

正数	正常終了、全ブロック復号するために必要な GCM 処理継続関数呼び出し回数
-1	異常終了 (atag_len=0, iv_len=0, div_block_num=0)

Remark

入力値の plain 及び aad は 16byte で割り切れない場合、パディング処理は関数内部で実施します。

鍵のサイズは組み合わせるブロック暗号に依存します。

不正なポインタ(ex. Null Pointer)、奇数アドレスを指定した場合の動作は不定です。

data_len = 0 の場合は、plain と cipher を使用しません。この場合入力される認証タグは GMAC 値として検証されます。aad_len = 0 の場合は、aad を使用しません。

平文データ領域と暗号文データ領域に同一領域を指定可能です。

(ただし、完全に同じ領域でなく領域がオーバラップしている場合の動作は不定です。)

work は演算途中のデータ等が格納されます。演算完了後、ユーザがゼロクリアしてください。

3.3.3 GCM 処理継続関数

Usage

#include “r_gcm.h”

int32_t R_gcm_repeat(uint32_t *work);

Parameters

work	入力/出力	分割処理管理用ワーク領域(156 byte 使用します)

Return Value

正数	正常終了、全ブロック暗号化/復号するために必要な GCM 処理継続関数呼び出し回数
0	正常終了、全ブロック暗号化/復号完了
-1	認証失敗または、異常終了 (GCM 暗号化開始関数及び GCM 復号開始関数の未呼び出し)

Description

R_gcm_repeat()関数は、中断した GCM 処理を再開する関数です。本関数の戻り値は全ブロック暗号化/復号するために必要な GCM 処理継続関数呼び出し回数を戻り値として返します。戻り値が 0 になるまで繰り返し呼び出してください。

Remark

なし

3.4　GCM ユーザ定義関数リファレンス

3.4.1　GCM 用 128bit ブロック暗号関数

Usage

#include “r_gcm.h”

int32_t R_encrypt_plain_block(uint8_t *plain, uint8_t *key);

Parameters

plain	入力/出力	平文/暗号文データ領域 (16 byte)
key	入力	鍵データ領域

Return Value

0	正常終了
-1	異常終了

Description

本関数はユーザ定義関数です。以下仕様に従いユーザにて実装してください。

R_encrypt_plain_block()関数は、第一引数”plain”で指定された平文データを 128bit ブロック暗号化して、第一引数”plain”で指定された領域に書き戻します。

128bit ブロック暗号に使用する鍵は、第二引数”key”で指定します。

Remark

GCM を使用しない場合、本関数の定義は不要です。

鍵のサイズは組み合わせるブロック暗号に依存します。

改訂記録

Rev.	発行日	改訂内容	
		ページ	ポイント
1.00	2011.02.18	—	初版発行
1.01	2011.05.06	—	GCM に対応しました。
1.02	2011.09.07	—	GCM をサンプルプログラムに変更しました。 on-the-fly 実装の関数を追加しました。
1.03	2012.04.16	—	ivec に関する説明を追加しました。 誤記修正
1.04	2012.09.28	21,22	Remark の文章中に「、奇数アドレス」を追加しました。
1.05	2012.12.27	— 25 —	AES-192 を追加しました。 AES テーブルデータ転送関数を追加しました。 AES ライブラリ表記を M3S-AES-LIB から AES ライブラリに変更しました。
1.06	2013.03.29	—	■ドキュメント構成変更 1.6 暗号データ検証方法を追加しました 1.7 提供形態を追加しました 1.8 ソフトウェアスタック構成を追加しました 1.9 スタックの取り扱いを追加しました ■GCM のサンプルの評価を完了し、正式にライブラリ化しました。 3. GCM ライブラリ仕様を追加しました 3.2.GCM 関数リファレンス(標準仕様)と 3.3.GCM 関数リファレンス(オプション仕様)に分離しました。 4.サンプルプログラムを追加しました。 (GCM とサンプルの項目を分離しました) 3.2.1.GCM 暗号化関数の平文ブロック数に 0 を許容するようにしました。 3.2.1.GCM 暗号化関数の追加認証ブロック数に 0 を許容するようにしました。 3.2.2.GCM 復号関数の平文ブロック数に 0 を許容するようにしました。 3.2.2.GCM 復号関数の追加認証ブロック数に 0 を許容するようにしました。 ■表記ブレ修正 GCM モード→GCM に統一しました。
1.07	2013.04.19	—	GCM 関連の関数仕様を変更しました。 誤記修正。

Rev.	発行日	改訂内容	
		ページ	ポイント
1.08	2013.04.23	A-1	改定記録 Rev.1.07 の発効日を修正 ■Rev.1.07 での改定内容の詳細を記載します。
		30	- 3.2.1 GCM 暗号化関数 データサイズの指定方法をブロック数からデータ長に変更。 引数 block_num を data_len に変更。 引数 aad_block_num を aad_len に変更。 文章中の「ブロック数」を「データ長」に変更。 引数の範囲を「block」から「byte」表現に変更。 パディング処理を関数内部で実施するように変更。 data_len や aad_len が 0 の場合の挙動を追記。
		31	- 3.2.2 GCM 復号関数 データサイズの指定方法をブロック数からデータ長に変更。 引数 block_num を data_len に変更。 引数 aad_block_num を aad_len に変更。 文章中の「ブロック数」を「データ長」に変更。 引数の範囲を「block」から「byte」表現に変更。 パディング処理を関数内部で実施するように変更。 data_len や aad_len が 0 の場合の挙動を追記。
		32	- 3.3 GCM 関数リファレンス(オプション仕様) 説明文をブロックから byte に変更。
		34	- 3.3.1 GCM 暗号化開始関数 3.2.1 GCM 暗号化関数と同様の変更。
		35	- 3.3.2 GCM 復号開始関数 3.2.2 GCM 復号関数と同様の変更。
1.09	2013.07.04	4 8	AES-GCM 検証に用いたテストベクタを変更しました。 1.10 メモリ配置について を追加しました。
1.10	2015.12.01	—	AES ライブラリと GCM ライブラリのバージョン情報を更新しました。 GMAC 値の出力方法を追記しました。 1.7 提供形態、4.サンプルプログラムを削除
2.00	2021.4.23	— 5	AES 192-bit のサポートを終了しました。 AES の検証に用いたテストベクタを変更しました。

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。