Renesas Microcomputer

AES Library: User's Manual

Introduction

AES Library (hereinafter referred to as the AES Crypto Library) is the software library incorporated in the Renesas Microcomputer and includes the data encryption/decryption functions that use the AES encryption technology. The supported AES functions are different according to the correspondence MCUs. Please refer the “Introduction Guide” with this document. The “Introduction Guide” explains the information that depends on microcomputer.

The following documents are for reference on the specifications and standards related to the AES Crypto Library.

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

This material explains the Library below.

- AES Library V.1.06
- GCM Library V.1.02

Target Device

Renesas Micro Computer

Contents

1. Outline

1.1 AES Algorithm

The AES algorithm is a symmetric block cipher that can encrypt and decrypt information. The input data that is used for this block cipher is called cipher key. This cipher key is used for encryption and decryption as same. Therefore, this algorithm is called as symmetric block cipher, common key algorithm. Encryption converts data to an unintelligible form called ciphertext; decrypting the ciphertext converts the data back into its original form, called plaintext.

The NIST initially announced this as a process for the selection of a new encryption algorithm to replace the DES(*1) algorithm. In the end, the Rijndael algorithm developed by Joan Daemen and Vincent Rijmen was adopted.

The Feistel and SPN(*2) structures are representative structures for block-cipher algorithms. A SPN structure has been adopted for the AES algorithm. In the SPN structure, substitution and permutation are repeatedly applied (round processing). Randomization by the SPN structure is typically more efficient than that by the Feistel structure, which means that fewer rounds of processing are required, and processing is faster overall.

*1) DES: The Data Encryption Standard (DES) algorithm is a symmetric block cipher using cipher keys. This standard was adopted as the US government standard in 1977.
*2) SPN: The Substitution Permutation Network (SPN) is a type of block cipher architecture that is adopted for the AES algorithm.

In the AES algorithm, 128-, 192-, or 256-bit cipher keys are used to generate a 128-bit ciphertext.

A different key called round key is used in each round, and is generated from the given 128-, 192-, or 256-bit cipher key. This is called key expansion.

Key expansion has to be performed at least once. When the same key is used for encryption and decryption, key expansion does not have to be performed again.

1.2 Block Cipher Modes of Operation

The NIST SP 800-38A includes several modes of block cipher to generate ciphertext blocks by encrypting plaintext blocks. The AES Crypto Library supports ECB and CBC modes.

- ECB (Electronic CodeBook) Mode

In ECB mode, cipher key and plaintext are the input data. When same plaintext data pattern will be input, same ciphertext data will be output at one block.

- CBC (Cipher Block Chaining) Mode

In CBC mode, previous block of ciphertext would be input data in addition of key and plaintext. Each block of plaintext to be encrypted is XORed with the previous block of ciphertext. The result is then encrypted with the cipher key to create the corresponding block of ciphertext. Since there is no previous block of ciphertext for the first block, an initialization vector (ivec) is used in place of the previous block of ciphertext block. Each output block data will be different even if using same data pattern input cause by this feature.

1.3 Method of Implementation

In the AES algorithm, there are two types of implementation. One is generating expanded key before processing of crypt. Another one is generating expanded key during processing of crypt. The second one is generally called “on-the-fly”.

1.4 AES-GCM

Galois/Counter Mode (GCM) is a mode of operation for symmetric key cryptographic block ciphers. This mode provides both data authenticity and confidentiality. The data encryption function generates authentication tag and encrypts plain text. The data decryption function verifies authentication tag and decrypts cipher text. And, in case that plaintext would be not input to the GCM encryption function, output authentication tag is Galois Message Authentication Code (GMAC) value. In case that ciphertext would be not input to the GCM decryption function, it is possible to verify the authentication tag. And encryption function and decryption function can be input the additional authentication data. The additional authentication data is input as secondary key. If user does not need to input the additional authentication data, input can be omitted.

GCM generally uses AES as symmetric block cipher algorithm.

1.5 Images of Block Cipher Modes of Operation

Electronic Code Book(ECB) mode

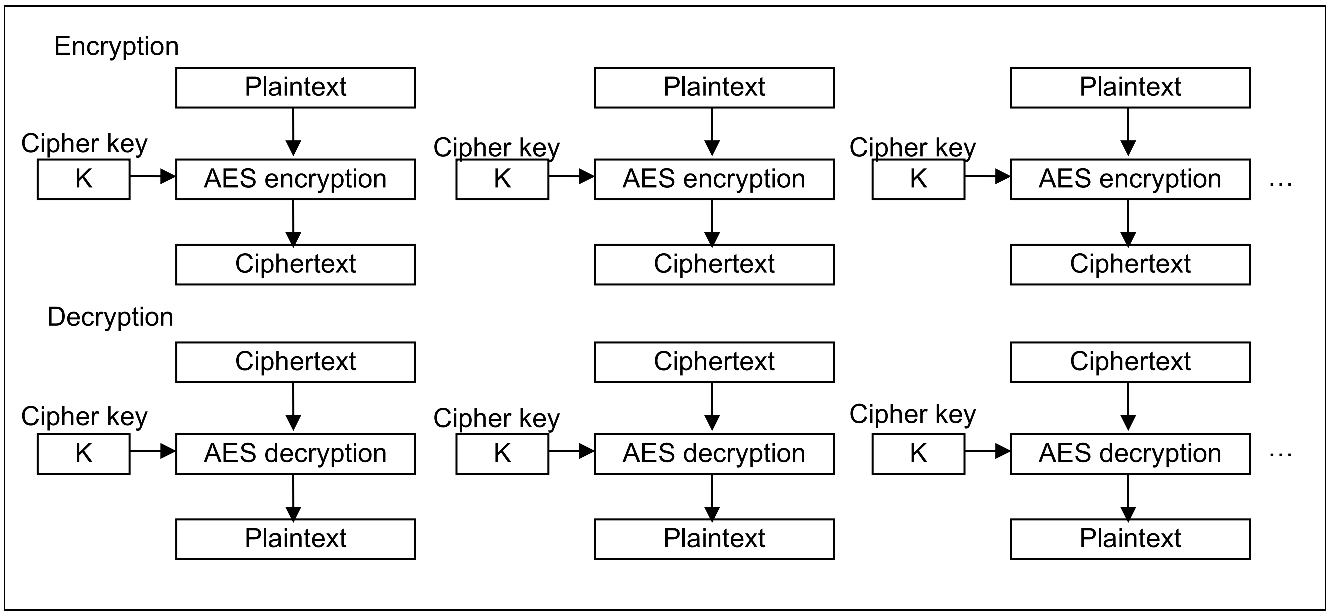

Figure 1 ECB mode

Cipher Block Chaining (CBC) mode

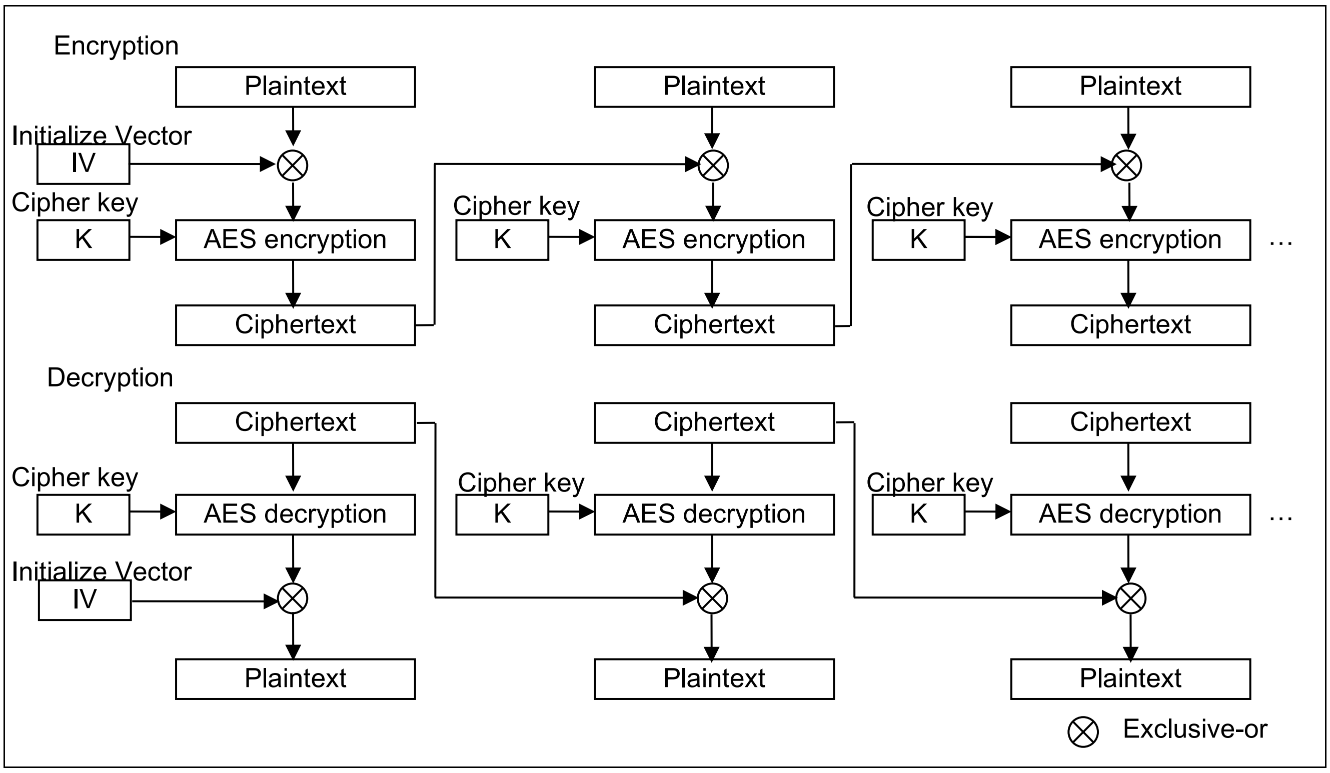

Figure 2 CBC mode

1.6 Validation Method for the Encryption/Decryption Data with this Library.

We use the information as reference for AES Encryption and Decryption and GCM Encryption and Decryption.

Test vectors for AES validation:

NIST (National Institute of Standards and Technology): This test pattern is standard pattern comes from American organization “NIST”.

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

Please search “AES Test Vectors” in this page.

User can download Test Vectors.

Test vectors for AES-GCM validation. CAVS 14.0

NIST (National Institute of Standards and Technology): This test pattern is standard pattern comes from American organization “NIST”.

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

Please search “GCM Test Vectors” in this page.

User can download Test Vectors.

1.7 Structure of Software Stack

AES Library has structure like below. If user uses GCM functions, please implement block cipher algorithm and key-setting functions. Sample program calls AES 128-bit key expansion function and AES 128 Encryption function (ECB mode).

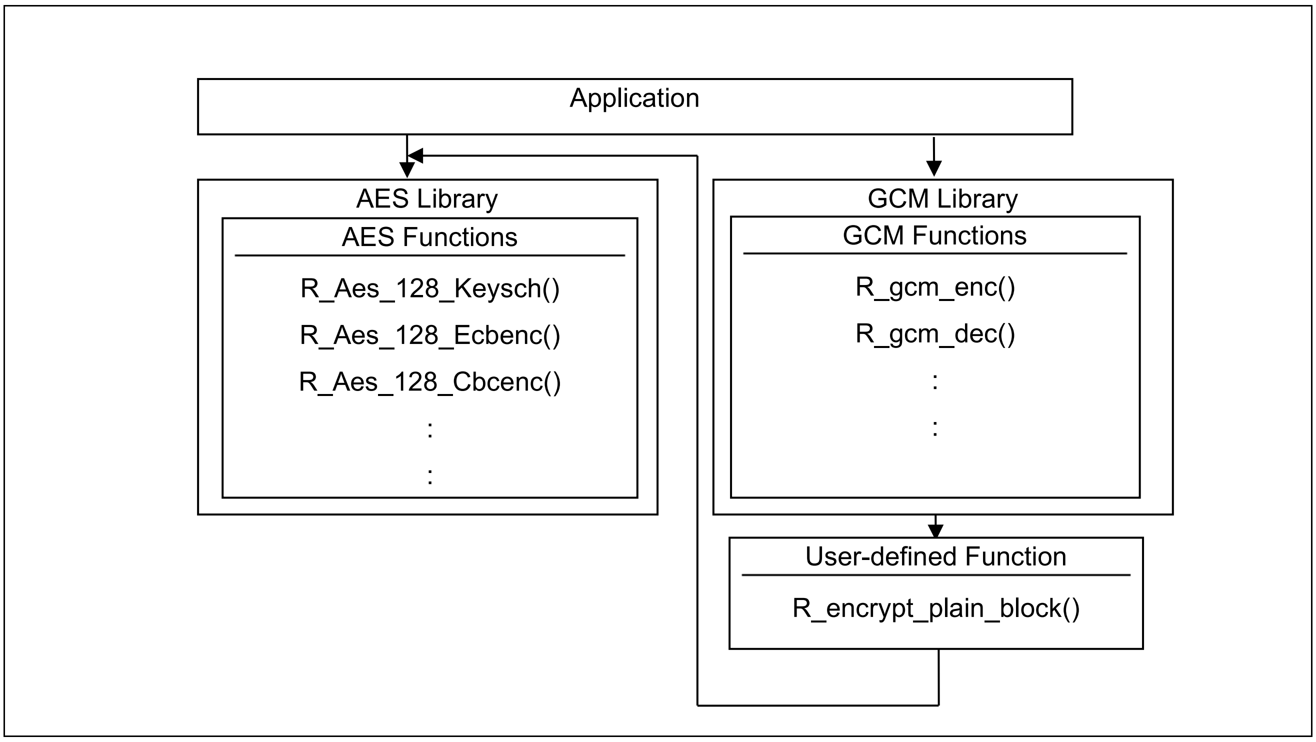

Figure 3 Structure of Software

1.8 Notice for Stack

AES Library and GCM Library store temporary data to stack. If this stack data continues existing, attacker may be able to expect key data. So, these Library will clear all stack data when returning. The time for clearing is very few.

1.9 About Memory Allocation

There is the memory allocation limitation for RL78 Family and other 16bit-MCUs to use AES Library.

These MCUs use the integer data as 16bit to make better performance. So, AES Library usually uses the data that is allocated in near area. This material has no explanation for “near”, “__near” modifier for pointer argument of library functions.

Please refer to the each MCUs “Introduction Guide” to get details.

2. AES Library Specification

2.1 Library Type Definitions

This section gives the details about the type definitions used in the library defined in a header file “r_stdint.h”.

Table 1 the type definitions list

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 Functions Reference

2.2.1 AES 128-bit Key Schedule

Usage

```
#include “r_aes.h”

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

Parameters

key	input	key data area. (16 byte)
ekey	output	expanded key area. (176 byte)

Return Value

none

Description

The R_Aes_128_Keysch() function executes the key schedule of AES. The R_Aes_128_Keysch() function expands the given key specified by the first argument “key” and writes the expanded key to an “ekey”. Because 176 bytes memory area is required to store the expansion key, “ekey” has to point to the memory area of at least 176 bytes.

Remark

When an invalid pointer (ex. NULL) is passed as a parameter, the functions behavior is undefined.

2.2.2 AES 128-bit Encryption Function (ECB Mode)

Usage

#include “r_aes.h”

void R_Aes_128_Ecbenc (uint8_t *ptext, uint8_t *ctext, uint32_t *ekey, uint16_t block);

Parameters

ptext	input	plain text area. (block * 16 byte)
ctext	output	cipher text area. (block * 16 byte)
ekey	input	expanded key area. (176 byte)
block	input	number of encryption block. (1 ~ any block)

Return Value

none

Description

The R_Aes_128_Ecbenc() function encrypts the plaintext at a specified address (the first argument “ptext”) by using the expanded key (the third argument “ekey”) in ECB mode and outputs ciphertext to a designated address (the second argument “ctext”). The amount specified by the number of blocks (the fourth argument “block” and each block is 16 byte in length) is encrypted.

Remark

When an invalid pointer (ex. NULL) is passed as a parameter, the functions behavior is undefined.

It is possible to allocate plain text area and cipher text area in same address.

(If these areas are not completely overlap, the function behavior is not defined.)

If store the data excluding output from R_Aes_128_Keysch() to the ekey, the function behavior is not defined.

2.2.3 AES 128-bit Decryption Function (ECB Mode)

Usage

#include “r_aes.h”

void R_Aes_128_Ecbdec (uint8_t *ctext, uint8_t *ptext, uint32_t *ekey, uint16_t block);

Parameter

ctext	input	cipher text area. (block * 16 byte)
ptext	output	plain text area. (block * 16 byte)
ekey	input	expanded key area. (176 byte)
block	input	number of decryption block. (1~any block)

Return Value

none

Description

The R_Aes_128_Ecbdec() function decrypts the ciphertext at a specified address (the first argument “ctext”) by using the expanded key (the third argument “ekey”) in ECB mode and outputs plaintext to a designated address (the second argument “ptext”). The amount specified by the number of blocks (the fourth argument “block” and each block is 16 byte in length) is encrypted.

Remark

When an invalid pointer (ex. NULL) is passed as a parameter, the functions behavior is undefined.

It is possible to allocate plain text area and cipher text area in same address.

(If these areas are not completely overlap, the function behavior is not defined.)

If store the data excluding output from R_Aes_128_Keysch() to the ekey, the function behavior is not defined.

2.2.4 AES 128-bit Encryption Function (CBC Mode)

Usage

```
#include “r_aes.h”
void R_Aes_128_Cbcenc (uint8_t *ptext, uint8_t *ctext, uint8_t *ivec, uint32_t *ekey, uint16_t block);
```

Parameters

ptext	input	plain text area. (block * 16 byte)
ctext	output	cipher text area. (block * 16 byte)
ivec	input/output	initialization vector area. (16 byte)
ekey	input	expanded key area. (176 byte)
block	input	number of encryption block. (1~any block)

Return Value

none

Description

The R_Aes_128_Cbcenc() function encrypts the plaintext at a specified address (the first argument “ptext”) by using the expanded key (the forth argument “ekey”) with initialization vector (the third argument “ivec”) in CBC mode and outputs ciphertext to a designated address (the second argument “ctext”). The amount specified by the number of blocks (the fifth argument “block” and each block are 16 byte in length) is encrypted. The "ivec" is used as 16-byte work area. When the block encryption has finished, the last encrypted block data would be output to the "ivec".

Remark

When an invalid pointer (ex. NULL) is passed as a parameter, the functions behavior is undefined.

It is possible to allocate plain text area and cipher text area in same address.

(If these areas are not completely overlap, the function behavior is not defined.)

If store the data excluding output from R_Aes_128_Keysch() to the ekey, the function behavior is not defined.

2.2.5 AES 128-bit Decryption Function (CBC Mode)

Usage

#include “r_aes.h”

void R_Aes_128_Cbcdec (uint8_t *ctext, uint8_t *ptext, uint32_t *ekey, uint16_t block);

Parameter

ctext	input	cipher text area. (block * 16 byte)
ptext	output	plain text area. (block * 16 byte)
ivec	input/output	initialization vector area. (16 byte)
ekey	input	expanded key area. (176 byte)
block	input	number of decryption block. (1~any block)

Return Value

none

Description

The R_Aes_128_Cbcdec() function decrypts the ciphertext at a specified address (the first argument “ctext”) by using the expanded key (the forth argument “key”) with initialization vector (the third argument “ivec”) in CBC mode and outputs plaintext to a designated address (the second argument “ptext”). The amount specified by the number of blocks (the fifth argument “block” and each block are 16 byte in length) is encrypted. The "ivec" is used as 16-byte work area. When the block encryption has finished, the last decrypted block data would be output to the "ivec".

Remark

When an invalid pointer (ex. NULL) is passed as a parameter, the functions behavior is undefined.

It is possible to allocate plain text area and cipher text area in same address.

(If these areas are not completely overlap, the function behavior is not defined.)

If store the data excluding output from R_Aes_128_Keysch() to the ekey, the function behavior is not defined.

2.2.6 AES 256-bit Key Schedule

Usage

```
#include "r_aes.h"
void R_Aes_256_Keysch (uint8_t *key, uint32_t *ekey);
```

Parameters

key	input	key data area. (32 byte)
ekey	output	expanded key area. (240 byte)

Return Value

none

Description

The R_Aes_256_Keysch() function executes the key schedule of AES. The R_Aes_256_Keysch() function expands the given key specified by the first argument “key” and writes the expanded key to an “ekey”. Because 240 bytes memory area is required to store the expansion key, “ekey” has to point to the memory area of at least 240 bytes.

Remark

When an invalid pointer (ex. NULL) is passed as a parameter, the functions behavior is undefined.

2.2.7 AES 256-bit Encryption Function (ECB Mode)

Usage

#include “r_aes.h”

void R_Aes_256_Ecbenc (uint8_t *ptext, uint8_t *ctext, uint32_t *ekey, uint16_t block);

Parameters

ptext	input	plain text area. (block * 16 byte)
ctext	output	cipher text area. (block * 16 byte)
ekey	input	expanded key area. (240 byte)
block	input	number of encryption block. (1～任意 block)

Return Value

none

Description

The R_Aes_256_Ecbenc() function encrypts the plaintext at a specified address (the first argument “ptext”) by using the expanded key (the third argument “ekey”) in ECB mode and outputs ciphertext to a designated address (the second argument “ctext”). The amount specified by the number of blocks (the fourth argument “block” and each block is 16 byte in length) is encrypted.

Remark

When an invalid pointer (ex. NULL) is passed as a parameter, the functions behavior is undefined.

It is possible to allocate plain text area and cipher text area in same address.

(If these areas are not completely overlap, the function behavior is not defined.)

If store the data excluding output from R_Aes_256_Keysch() to the ekey, the function behavior is not defined.

2.2.8 AES 256-bit Decryption Function (ECB Mode)

Usage

```
#include "r_aes.h"
void R_Aes_256_Ecbdec (uint8_t *ctext, uint8_t *ptext, uint32_t *ekey, uint16_t block);
```

Parameters

ctext	input	cipher text area. (block * 16 byte)
ptext	output	plain text area. (block * 16 byte)
ekey	input	expanded key area. (240 byte)
block	input	number of decryption block. (1~any block)

Return Value

none

Description

The R_Aes_256_Ecbdec() function decrypts the ciphertext at a specified address (the first argument “ctext”) by using the expanded key (the third argument “ekey”) in ECB mode and outputs plaintext to a designated address (the second argument “ptext”). The amount specified by the number of blocks (the fourth argument “block” and each block is 16 byte in length) is encrypted.

Remark

When an invalid pointer (ex. NULL) is passed as a parameter, the functions behavior is undefined.

It is possible to allocate plain text area and cipher text area in same address.

(If these areas are not completely overlap, the function behavior is not defined.)

If store the data excluding output from R_Aes_256_Keysch() to the ekey, the function behavior is not defined.

2.2.9 AES 256-bit Encryption Function (CBC Mode)

Usage

#include “r_aes.h”

void R_Aes_256_Cbcenc (uint8_t *ptext, uint8_t *ctext, uint8_t *ivec, uint32_t *ekey, uint16_t block);

Parameters

ptext	input	plain text area. (block * 16 byte)
ctext	output	cipher text area. (block * 16 byte)
ivec	input/output	initialization vector area. (16 byte)
ekey	input	expanded key area. (240 byte)
block	input	number of encryption block. (1~any block)

Return Value

none

Description

The R_Aes_256_Cbcenc() function encrypts the plaintext at a specified address (the first argument “ptext”) by using the expanded key (the forth argument “ekey”) with initialization vector (the third argument “ivec”) in CBC mode and outputs ciphertext to a designated address (the second argument “ctext”). The amount specified by the number of blocks (the fifth argument “block” and each block are 16 byte in length) is encrypted. The "ivec" is used as 16-byte work area. When the block encryption has finished, the last encrypted block data would be output to the "ivec".

Remark

When an invalid pointer (ex. NULL) is passed as a parameter, the functions behavior is undefined.

It is possible to allocate plain text area and cipher text area in same address.

(If these areas are not completely overlap, the function behavior is not defined.)

If store the data excluding output from R_Aes_256_Keysch() to the ekey, the function behavior is not defined.

2.2.10 AES 256-bit Decryption Function (CBC Mode)

Usage

#include “r_aes.h”

void R_Aes_256_Cbcdec (uint8_t *ctext, uint8_t *ptext, uint32_t *ekey, uint16_t block);

Parameters

ctext	input	cipher text area. (block * 16 byte)
ptext	output	plain text area. (block * 16 byte)
ivec	input/output	initialization vector area. (16 byte)
ekey	input	expanded key area. (240 byte)
block	input	number of decryption block. (1~any block)

Return Value

none

Description

The R_Aes_256_Cbcdec() function decrypts the ciphertext at a specified address (the first argument “ctext”) by using the expanded key (the forth argument “key”) with initialization vector (the third argument “ivec”) in CBC mode and outputs plaintext to a designated address (the second argument “ptext”). The amount specified by the number of blocks (the fifth argument “block” and each block are 16 byte in length) is encrypted. The "ivec" is used as 16-byte work area. When the block encryption has finished, the last decrypted block data would be output to the "ivec".

Remark

When an invalid pointer (ex. NULL) is passed as a parameter, the functions behavior is undefined.

It is possible to allocate plain text area and cipher text area in same address.

(If these areas are not completely overlap, the function behavior is not defined.)

If store the data excluding output from R_Aes_256_Keysch() to the ekey, the function behavior is not defined.

2.2.11 AES 128-bit on-the-fly Encryption Function (ECB Mode)

Usage

```
#include "r_aes.h"
void R_Aes_128_OtfEcbenc (uint8_t *ptext, uint8_t *ctext, uint16_t *key, uint16_t block);
```

Parameters

ptext	input	plain text area. (block * 16 byte)
ctext	output	cipher text area. (block * 16 byte)
key	input	key area. (16 byte)
block	input	number of encryption block. (1~any block)

Return Value

none

Description

R_Aes_128_OtfEcbenc() function encrypts the plaintext at a specified address (the first argument “ptext”) by using the key (the third argument “key”) in ECB mode and outputs ciphertext to a designated address (the second argument “ctext”). The amount specified by the number of blocks (the fourth argument “block” and each block is 16 byte in length) is encrypted.

Remark

When an invalid pointer (ex. NULL) is passed as a parameter, the functions behavior is undefined.

It is possible to allocate plain text area and cipher text area in same address.

(If these areas are not completely overlap, the function behavior is not defined.)

2.2.12 AES 128-bit on-the-fly Decryption Function (ECB Mode)

Usage

```
#include "r_aes.h"
void R_Aes_128_OtfEcbdec (uint8_t *ctext, uint8_t *ptext, uint16_t *key, uint16_t block);
```

Parameter

ctext	input	cipher text area. (block * 16 byte)
ptext	output	plain text area. (block * 16 byte)
key	input	key area. (16 byte)
block	input	number of decryption block. (1~any block)

Return Value

none

Description

R_Aes_128_OtfEcbdec() function decrypts the ciphertext at a specified address (the first argument "ctext") by using the key (the third argument "key") in ECB mode and outputs plaintext to a designated address (the second argument "ptext"). The amount specified by the number of blocks (the fourth argument "block" and each block is 16 byte in length) is encrypted.

Remark

When an invalid pointer (ex. NULL) is passed as a parameter, the functions behavior is undefined.

It is possible to allocate plain text area and cipher text area in same address.

(If these areas are not completely overlap, the function behavior is not defined.)

2.2.13 AES 128-bit on-the-fly Encryption Function (CBC Mode)

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	input	plain text area. (block * 16 byte)
ctext	output	cipher text area. (block * 16 byte)
ivec	input/output	initialization vector area. (16 byte)
key	input	key area. (16 byte)
block	input	number of encryption block. (1~any block)

Return Value

none

Description

The R_Aes_128_OtfCbcenc() function encrypts the plaintext at a specified address (the first argument “ptext”) by using the key (the forth argument “key”) with initialization vector (the third argument “ivec”) in CBC mode and outputs ciphertext to a designated address (the second argument “ctext”). The amount specified by the number of blocks (the fifth argument “block” and each block are 16 byte in length) is encrypted. The "ivec" is used as 16-byte work area. When the block encryption has finished, the last encrypted block data would be output to the "ivec".

Remark

When an invalid pointer (ex. NULL) is passed as a parameter, the functions behavior is undefined.

It is possible to allocate plain text area and cipher text area in same address.

(If these areas are not completely overlap, the function behavior is not defined.)

When encrypting continuation data and this function needs to be called two or more times, the right plain text result can be output by this function, without changing ivec.

2.2.14 AES 128-bit on-the-fly Decryption Function (CBC Mode)

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);

Parameter

ctext	input	cipher text area. (block * 16 byte)
ptext	output	plain text area. (block * 16 byte)
ivec	input/output	initialization vector area. (16 byte)
key	input	key area. (16 byte)
block	input	number of decryption block. (1~any block)

Return Value

none

Description

The R_Aes_128_OtfCbcdec() function decrypts the ciphertext at a specified address (the first argument “ctext”) by using the key (the forth argument “key”) with initialization vector (the third argument “ivec”) in CBC mode and outputs plaintext to a designated address (the second argument “ptext”). The amount specified by the number of blocks (the fifth argument “block” and each block are 16 byte in length) is encrypted. The "ivec" is used as 16-byte work area. When the block encryption has finished, the last decrypted block data would be output to the "ivec".

Remark

When an invalid pointer (ex. NULL) is passed as a parameter, the functions behavior is undefined.

It is possible to allocate plain text area and cipher text area in same address.

(If these areas are not completely overlap, the function behavior is not defined.)

2.2.15 AES Table Data Transmission Function (M16C only)

Usage

#include “r_aes.h”

void R_Aes_Table_Init(void);

Parameter

none

Return Value

none

Description

It is necessary for the AES library of M16C to transfer table data to the RAM area. Please call this function before working to AES encryption and decryption. This function transfers table data to _DATA_TBL_NEAR of the RAM area from _DATA_TBL_ROM of the ROM area.

Remark

none

3. GCM Library Specification

3.1 Library Type Definitions

GCM Library uses the data type that is defined in header file “r_stdint.h”. This is same as data type for AES Library. Please refer to the section “2.1 Library Type Definitions”

3.2 GCM Functions Reference (Standard functions)

3.2.1 GCM Encryption Function

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	input	plain text area.	(data_len bytes)
cipher	output	cipher text area.	(data_len bytes)
data_len	input	plain text length	(0-any bytes)
key	input	key area.	(16 byte)
atag	output	authentication tag	(atag_len bytes)
atag_len	input	authentication tag length	(1-16 bytes)
iv	input	initialize vector area	(iv_len bytes)
iv_len	input	initialize vector length	(1-any bytes)
aad	input	aad area	(aad_len bytes)
aad_len	input	aad length	(0-any bytes)

Return Value

0	Normal termination
-1	Abnormal termination (atag_len=0, iv_len=0)

Description

The R_gcm_enc() function encrypts the plaintext at a specified address (the 1st argument “plain”) by using the key (the 4th argument “key”) and the initialize vector (the 7th argument “iv”) and the additional authentication data (the 9th argument “aad”). Encrypted data will output on the 2nd argument “cipher”. Authentication Tag data will output to the 5th argument “atag”. Lengths to encrypt are specified 3rd argument “data_len”.

Remark

When input data, plain and aad are not aligned 16byte, the padding process is executed in this function.

The key size depends on block cipher algorithm.

When an invalid pointer (ex. NULL) or odd address is passed as a parameter, the functions behavior is undefined.

When data_len = 0, this function does not use plain and cipher. This case, output authentication tag is called as GMAC value. When aad_len = 0, this function does not use aad.

It is possible to allocate plain text area and cipher text area in same address.

(If these areas are not completely overlap, the function behavior is not defined.)

3.2.2 GCM Decryption Function

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	input	cipher text area.	(data_len bytes)
plain	output	plain text area.	(data_len bytes)
data_len	input	plain text length	(0-any bytes)
key	input	key area.	(16 byte)
atag	input	authentication tag	(atag_len bytes)
atag_len	input	authentication tag length	(1-16 bytes)
iv	input	initialize vector area	(iv_len bytes)
iv_len	input	initialize vector length	(1-any bytes)
aad	input	aad area	(aad_len bytes)
aad_len	input	aad length	(0-any bytes)

Return Value

0	Successful authentication
-1	Failure authentication or illegal termination (atag_len=0, iv_len=0)

Description

The R_gcm_dec() function decrypts and verifies the ciphertext at a specified address (the 1st argument “cipher”) by using the key (the 4th argument “key”) and the authentication tag (the 5th argument “atag”) and the initialize vector (the 7th argument “iv”) and the additional authentication data (the 9th argument “aad”). If result of verification is OK, this function outputs plaintext to the 2nd argument “plain”. If result of verification is NG, this function outputs 0 to “plain”. Lengths to decrypt are specified 3rd argument “data_len”.

Remark

When input data, plain and aad are not aligned 16byte, the padding process is executed in this function.

The key size depends on block cipher algorithm.

When an invalid pointer (ex. NULL) or odd address is passed as a parameter, the functions behavior is undefined.

When data_len = 0, this function does not use plain and cipher. This case, input authentication tag would be verified as GMAC value. When aad_len = 0, this function does not use aad.

It is possible to allocate plain text area and cipher text area in same address.

(If these areas are not completely overlap, the function behavior is not defined.)

3.3 GCM Functions Reference (Optional functions)

GCM functions reference (optional functions) functions are processing GCM with better use for embedded system. There are de-merit in 3.2.1 GCM Encryption Function and 3.2.2 GCM Decryption Function like below.

- More processing time is needed for calling function because all arguments in one time.

- Blocking in function until all plain/cipher data would be encrypted/decrypted.

Using 3.3.1 GCM Start Encryption Function, 3.3.2 GCM Start Decryption Function, 3.3.3 GCM Repeat Function solves this de-merit.

GCM Start Encryption Function or GCM Start Decryption Function return the value that how times GCM Repeat Function calling is needed. Please call GCM Repeat Function repeatedly the times that is returned from each GCM Start Encryption/Decryption Functions.

There are 8 steps in GCM decryption internal process, 7 steps in GCM encryption internal function. GCM function reference (Optional functions) functions will divide the Encryption/Decryption process as below. This example uses “1” as divided block number.

GCM encryption process:

	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 decryption process：

	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	

(STEP4 and STEP6 are exchanged for reasons about internal implement.)

User can specify the divided block number for processing time is proportional(*) to plain block number.

In case, plain block number = 100, divided block number = 8: process is divided into 13 pieces (100/8 = 12 + 1 (Fraction))

Therefore, GCM Encryption Function returns 29 (1+13+1+13+1), user calls GCM Repeat Function in 29 times and completes 100 block encryptions.

This is processing flow with all 100 block encryption with divided unit is 8block.

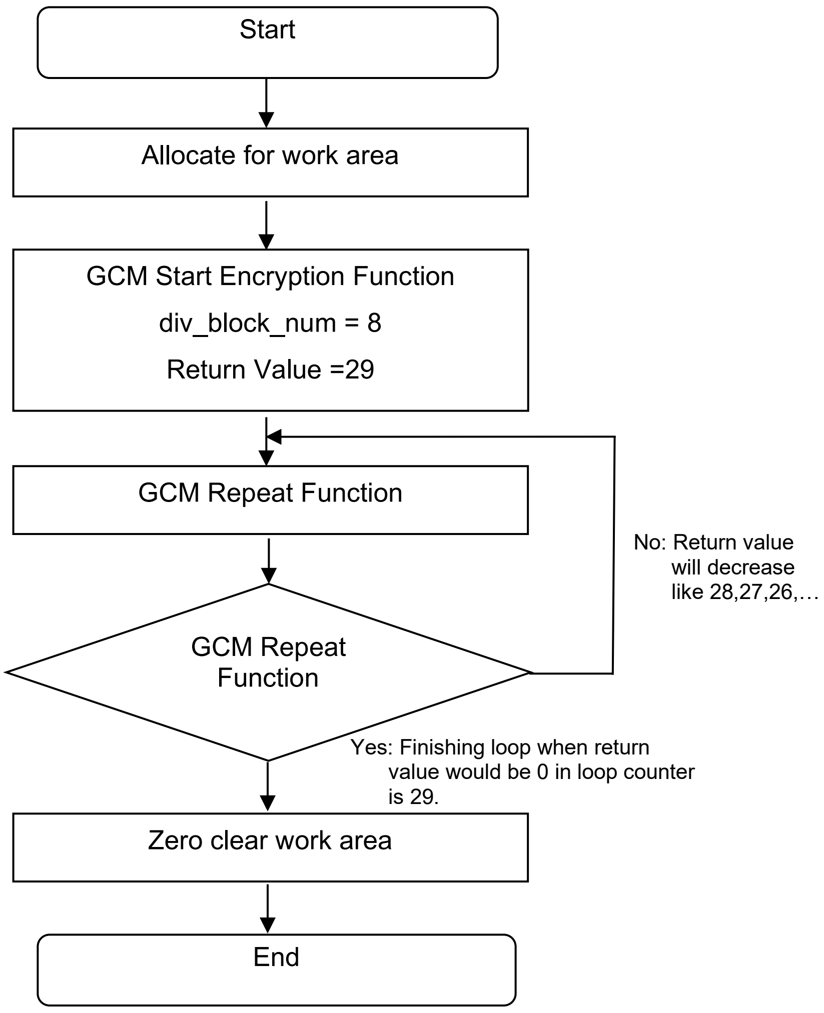
Start
Allocate for work area
GCM Start Encryption Function
div_block_num = 8
Return Value =29
GCM Repeat Function
No: Return value will decrease like 28,27,26,…
GCM Repeat Function
Yes: Finishing loop when return value would be 0 in loop counter is 29.
Zero clear work area
End

Figure 4 Processing flow

3.3.1 GCM Start Encryption Function

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	input	plain text area.	(data_len bytes)
cipher	output	cipher text area.	(data_len bytes)
data_len	input	plain text length	(0-any bytes)
key	input	key area.	(16 byte)
atag	output	authentication tag	(atag_len bytes)
atag_len	input	authentication tag length	(1-16 bytes)
iv	input	initialize vector area	(iv_len bytes)
iv_len	input	initialize vector length	(1-any bytes)
aad	input	aad area	(aad_len bytes)
aad_len	input	aad length	(0-any bytes)
div_block_num	input	divided block number (1-any blocks)	
work	output	work area for dividing process (requires 156 byte)	

Description

The R_gcm_enc() function encrypts the plaintext at a specified address (the 1st argument "plain") by using the key (the 4th argument "key") and the initialize vector (the 7th argument "iv") and the additional authentication data (the 9th argument "aad"). Encrypted data will output on the 2nd argument "cipher". Authentication Tag data will output to the 5th argument "atag". Lengths to encrypt are specified 3rd argument "data_len".

This function will recode each argument data to the 12th argument "work". This function does not execute encryption. To execute encryption, please call 3.3.3 GCM Repeat Function repeatedly. 3.3.3 GCM Repeat Function reads plaintext unit that specified in 11th argument "div_block_num". The 1block size is 16 bytes.

Return Value

positive value	Normal termination. The times to call GCM Repeat Function for all block encryption completion.
-1	Abnormal termination (atag_len=0, iv_len=0, div_block_num=0)

Remark

When input data, plain and aad are not aligned 16byte, the padding process is executed in this function.

The key size depends on block cipher algorithm.

When an invalid pointer (ex. NULL) or odd address is passed as a parameter, the functions behavior is undefined.

When data_len = 0, this function does not use plain and cipher. This case, output authentication tag is called as GMAC value.When aad_len = 0, this function does not use aad.

It is possible to allocate plain text area and cipher text area in same address.

(If these areas are not completely overlap, the function behavior is not defined.)

The temporary data will store to work, please clear work area after all data encryptions.

3.3.2 GCM Start Decryption Function

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	input	cipher text area.	(data_len bytes)
plain	output	plain text area.	(data_len bytes)
data_len	input	plain text length	(0-any bytes)
key	input	key area.	(16 byte)
atag	input	authentication tag	(atag_len bytes)
atag_len	input	authentication tag length	(1-16 bytes)
iv	input	initialize vector area	(iv_len bytes)
iv_len	input	initialize vector length	(1-any bytes)
aad	input	aad area	(aad_len bytes)
aad_len	input	aad length	(0-any bytes)
div_block_num	input	divided block number (1-any blocks)	
work	output	work area for dividing process (requires 156 byte)	

Description

The R_gcm_dec() function decrypts and verifies the ciphertext at a specified address (the 1st argument “cipher”) by using the key (the 4th argument “key”) and the authentication tag (the 5th argument “atag”) and the initialize vector (the 7th argument “iv”) and the additional authentication data (the 9th argument “aad”). If result of verification is OK, this function outputs plaintext to the 2nd argument “plain”. If result of verification is NG, this function outputs 0 to “plain”. Lengths to decrypt are specified 3rd argument “data_len”.

This function will recode each argument data to the 12th argument “work”. This function does not execute decryption. To execute decryption, please call 3.3.3 GCM Repeat Function repeatedly. 3.3.3 GCM Repeat Function reads ciphertext unit that specified in 11th argument “div_block_num”. The block size is 16 bytes.

Return Value

positive value	Normal termination. The times to call GCM Repeat Function for all block encryption completion.
-1	Abnormal termination (atag_len=0, iv_len=0, div_block_num=0)

Remark

When input data, plain and aad are not aligned 16byte, the padding process is executed in this function.

The key size depends on block cipher algorithm.

When an invalid pointer (ex. NULL) or odd address is passed as a parameter, the functions behavior is undefined.

When data_len = 0, this function does not use plain and cipher. This case, input authentication tag would be verified as GMAC value. When aad_len = 0, this function does not use aad.

It is possible to allocate plain text area and cipher text area in same address.

(If these areas are not completely overlap, the function behavior is not defined.)

The temporary data will store to work, please clear work area after all data encryptions.

3.3.3 GCM Repeat Function

Usage

#include “r_gcm.h”

int32_t R_gcm_repeat(uint32_t *work);

Parameters

work	input/output	work area for dividing process (requires 156 byte)

Return Value

positive value	Normal termination. The times to call GCM Repeat Function for all block encryption completion.
0	Successful authentication and all encryption/decryption.
-1	Failure authentication or illegal termination. (No call GCM Start Encryption/Decryption Function)

Description

R_gcm_repeat() function is the function that restarts the pending GCM processing. This function returns the value that means remaining times for encryption/decryption completion. Please call this function until this function returns 0.

Remark

none

3.4 GCM User-defined Functions Reference

3.4.1 128bit Data Block Encryption Function for GCM

Usage

```
#include “r_gcm.h”
int32_t R_encrypt_plain_block(uint8_t *plain, uint8_t *key);
```

Parameters

plain	input/output	plaintext/ciphertext area	(16 bytes)
key	input	key data area	

Return Value

0	Normal termination
-1	Abnormal termination

Description

This function is user definition function. Please implement using following specifications.

The R_encrypt_plain_block() encrypts the 128bits data specified the 1st argument “plain” and stores to the 1st argument.

The key data with 128 bit block encryption is specified in 2nd argument “key”

Remark

This function is not needed for no GCM system.

The key size depends on block cipher algorithm.

Revision History

Rev.	Date	Description	
		Page	Summary
1.00	Feb 18, 2011	—	First edition issued
1.01	May 06, 2011	—	Add GCM support.
1.02	Sep 07, 2011	—	Change GCM to sample program. Add on-the-fly functions
1.03	Apr 16, 2012	—	Add explanation about ivec. Other clerical error correction.
1.04	Sep 28, 2012	21,22	Add “or odd address” in Remark.
1.05	Dec 27, 2012	— 25 —	Add Function of AES-192. Add “AES Table Data Transmission Function” Changed AES library notation from M3S-AES-LIB to AES Library.
1.06	Mar 29, 2013	—	■ Changed document structure Added 1.6. Validation method for the Encryption/Decryption data with this Library. Added 1.7. Providing method Added 1.8. Structure of Software Stack Added 1.9. Notice for stack ■ Completed GCM sample source code test. Added GCM Library as formal version Added 3. GCM Library specification Separated 3.2 GCM Function Reference (Standard Functions) and 3.3 GCM Function Reference (Optional Functions). Added 4.Sample Program (Separated GCM section and Sample section) Changed forgiving 0 into plaintext block number in 3.2.1.GCM Encryption Function. Changed forgiving 0 into additional authentication block number in 3.2.1.GCM Encryption Function. Changed forgiving 0 into plaintext block number in 3.2.2.GCM Decryption Function. Changed forgiving 0 into additional authentication block number in 3.2.2.GCM Decryption Function.
1.07	Apr 19, 2013	—	Changed GCM functions specification. Other clerical error correction.

Rev.	Date	Description	
		Page	Summary
1.08	Apr 23, 2013		■ Shown details of Rev.1.07's Summary.
		30	- 3.2.1 GCM Encryption Function Changed: specification method of data size. "block number" to "data length (byte)". Changed: the argument "block_num" to "data_len". Changed: the argument "aad_block_num" to "aad_len". Changed explanation: explanation in material "block number" to "data length". Changed explanation: the meaning of argument "blocks" to "bytes". Changed: the executing of padding process in function. Added explanation: The behavior when data_len and aad_len are "0".
		31	- 3.2.2 GCM Decryption Function Changed: specification method of data size. "block number" to "data length (byte)". Changed: the argument "block_num" to "data_len". Changed: the argument "aad_block_num" to "aad_len". Changed explanation: explanation in material "block number" to "data length". Changed explanation: the meaning of argument "blocks" to "bytes". Changed: the executing of padding process in function. Added explanation: The behavior when data_len and aad_len are "0".
		32	- 3.3 GCM Functions Reference (Optional functions) Changed explanation: Block to Byte.
		34	- 3.3.1 GCM Start Encryption Function Same as 3.2.1 GCM Encryption Function.
		35	- 3.3.2 GCM Start Decryption Function Same as 3.2.2 GCM Decryption Function.
1.09	Jul 04, 2013	4 8	Changed Test vectors for AES-GCM validation. Added 1.10 About Memory Allocation
1.10	Dec 01, 2015	—	Changed AES library version and GCM library version information. Added output method of GMAC value Deleted 1.7. Providing method Deleted 4.Sample Program
2.00	Apr 23, 2021	— 6	AES 192-bit is no longer supported. Changed Test vectors for AES validation.

General Precautions in the Handling of Microprocessing Unit and Microcontroller Unit Products

The following usage notes are applicable to all Microprocessing unit and Microcontroller unit products from Renesas. For detailed usage notes on the products covered by this document, refer to the relevant sections of the document as well as any technical updates that have been issued for the products.

1. Precaution against Electrostatic Discharge (ESD)

   A strong electrical field, when exposed to a CMOS device, can cause destruction of the gate oxide and ultimately degrade the device operation. Steps must be taken to stop the generation of static electricity as much as possible, and quickly dissipate it when it occurs. Environmental control must be adequate. When it is dry, a humidifier should be used. This is recommended to avoid using insulators that can easily build up static electricity. Semiconductor devices must be stored and transported in an anti-static container, static shielding bag or conductive material. All test and measurement tools including work benches and floors must be grounded. The operator must also be grounded using a wrist strap. Semiconductor devices must not be touched with bare hands. Similar precautions must be taken for printed circuit boards with mounted semiconductor devices.
2. Processing at power-on

   The state of the product is undefined at the time when power is supplied. The states of internal circuits in the LSI are indeterminate and the states of register settings and pins are undefined at the time when power is supplied. In a finished product where the reset signal is applied to the external reset pin, the states of pins are not guaranteed from the time when power is supplied until the reset process is completed. In a similar way, the states of pins in a product that is reset by an on-chip power-on reset function are not guaranteed from the time when power is supplied until the power reaches the level at which resetting is specified.
3. Input of signal during power-off state

   Do not input signals or an I/O pull-up power supply while the device is powered off. The current injection that results from input of such a signal or I/O pull-up power supply may cause malfunction and the abnormal current that passes in the device at this time may cause degradation of internal elements. Follow the guideline for input signal during power-off state as described in your product documentation.
4. Handling of unused pins

   Handle unused pins in accordance with the directions given under handling of unused pins in the manual. The input pins of CMOS products are generally in the high-impedance state. In operation with an unused pin in the open-circuit state, extra electromagnetic noise is induced in the vicinity of the LSI, an associated shoot-through current flows internally, and malfunctions occur due to the false recognition of the pin state as an input signal become possible.
5. Clock signals

   After applying a reset, only release the reset line after the operating clock signal becomes stable. When switching the clock signal during program execution, wait until the target clock signal is stabilized. When the clock signal is generated with an external resonator or from an external oscillator during a reset, ensure that the reset line is only released after full stabilization of the clock signal. Additionally, when switching to a clock signal produced with an external resonator or by an external oscillator while program execution is in progress, wait until the target clock signal is stable.
6. Voltage application waveform at input pin

   Waveform distortion due to input noise or a reflected wave may cause malfunction. If the input of the CMOS device stays in the area between V_{IL} (Max.) and V_{IH} (Min.) due to noise, for example, the device may malfunction. Take care to prevent chattering noise from entering the device when the input level is fixed, and also in the transition period when the input level passes through the area between V_{IL} (Max.) and V_{IH} (Min.).
7. Prohibition of access to reserved addresses

   Access to reserved addresses is prohibited. The reserved addresses are provided for possible future expansion of functions. Do not access these addresses as the correct operation of the LSI is not guaranteed.
8. Differences between products

   Before changing from one product to another, for example to a product with a different part number, confirm that the change will not lead to problems. The characteristics of a microprocessing unit or microcontroller unit products in the same group but having a different part number might differ in terms of internal memory capacity, layout pattern, and other factors, which can affect the ranges of electrical characteristics, such as characteristic values, operating margins, immunity to noise, and amount of radiated noise. When changing to a product with a different part number, implement a system-evaluation test for the given product.