AN13975

i.MX RT eMMC RPMB Enablement

Rev. 1 — 15 June 2023 Application note

Document Information

Information	Content
Keywords	uSDHC, eMMC, Replay Protected Memory Block (RPMB)
Abstract	This document aims to introduce how to read and write the Replay Protected Memory Block (RPMB) partition of eMMC through uSDHC on the RT chips platform.

1 Introduction

This section describes the purpose of this application note and gives general information to the users who want to enable eMMC Replay Protected Memory Block (RPMB) partition in RT platforms.

1.1 Purpose

In addition to NOR flash, eMMC is commonly used in MCU development. For RT series chips, eMMC can be accessed through the uSDHC IP. In eMMC, the RPMB partition is a focus of customer attention because it can protect OEM or customer privacy information. This document aims to introduce how to read and write the RPMB partition of eMMC through uSDHC on the RT chips platform.

Note: *The platform used is RT1170, some modifications may be required for other RT platforms.*

1.2 Audience and scope

This document is targeted for i.MX RT users who want to understand:

- The RPMB read and write functionality available on the SoC
- The software enablement for this feature
- How to start developing an application that utilizes this feature

The reader must be familiar with basic secure provisioning and eMMC concepts.

1.3 Acronyms and abbreviations

Table 1. Acronyms and abbreviations

Acronym	Definition
AES	Advanced Encryption Standard
CRC	Cyclic Redundancy Check
CAAM	Cryptographic Acceleration and Assurance Module
eMMC	Embedded Multimedia Card
HMAC SHA-256	Hash Message Authentication Code using Secure Hash Algorithms 256bits
uSDHC	Ultra Secured Digital Host Controller
RPMB	Replay Protected Memory Block
RNG	Random Number Generator
MAC	Message Authentication Code

2 Overview

eMMC (Embedded Multimedia Card) is an embedded storage solution widely used in mobile devices, such as smartphones, tablets, and cameras. It integrates flash memory and a flash memory controller in a small BGA package, offering an excellent balance between performance, capacity, and cost.

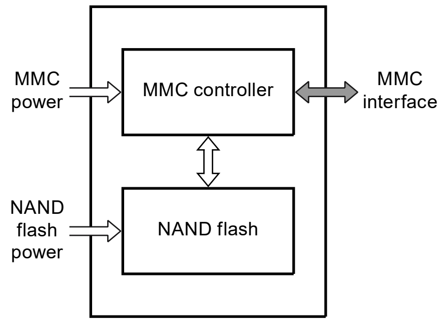

Figure 1. eMMC Device

In RT series chips, eMMC is connected by The ultra Secured Digital Host Controller (uSDHC) that provides the interface between the host system and the MMC cards. The module acts as a bridge, passing host bus transactions to the MMC cards by sending commands and performing data accesses to/from the cards. It handles the MMC protocols at the transmission level.

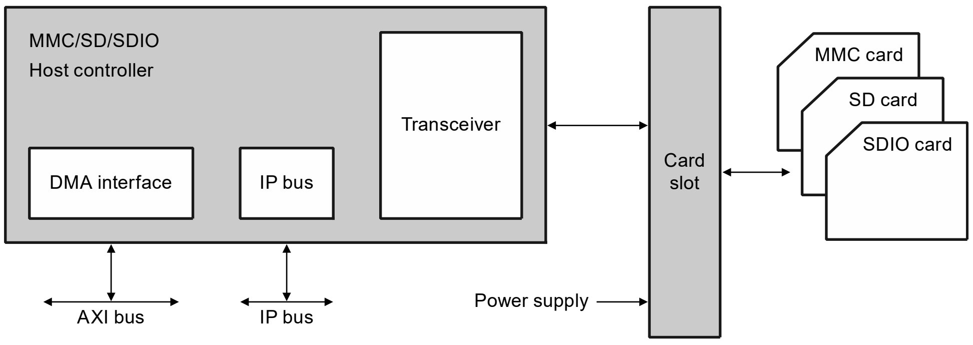

Figure 2. System connection of uSDHC

One of the essential components of eMMC is the RPMB partition that stands for Replay Protected Memory Block. The RPMB partition is a small portion of the eMMC memory dedicated to storing critical security-related information, such as encryption keys, device IDs, and certificates. It is designed to provide a tamper-proof mechanism for secure boot, secure firmware updates, digital signatures, content protection, and other security functions.

2.1 Introduction of RPMB partition

The Replay Protected Memory Block (RPMB) partition serves as a critical component within eMMC storage systems, providing a secure area for storing sensitive security-related information. The RPMB partition's primary purpose is to provide a tamper-proof mechanism for security functions performed by mobile devices, such as smartphones, tablets, and cameras.

One of the most critical aspects of the RPMB partition is its ability to maintain confidentiality, availability, and integrity of stored data. The RPMB's ability to preserve the integrity of the stored data makes it an essential part of a security system since it helps prevent tampering and unauthorized access.

In addition to storing security-critical data, RPMB partition ensures that software updates to firmware and bootloaders are carried out securely. Software updates to these components of mobile devices must be secure since they can affect the device's authenticity and trustworthiness. With RPMB partition providing a secure area to implement such updates, mobile devices can ensure a higher level of security and trustworthiness.

The RPMB partition is protected through a replay protection protocol that prevents it from replay attacks and ensures that only authorized parties can access and modify stored data. The replay protection protocol is based on the Advanced Encryption Standard (AES) algorithm, which provides robust encryption, and makes it very challenging for attackers to tamper with stored data.

In summary, the RPMB partition serves a vital purpose, providing a tamper-proof mechanism for secure boot, firmware updates, digital signatures, content protection, and other security functions. The partition helps guarantee the confidentiality, availability, and integrity of stored data by using robust encryption and replay protection protocol. Its use provides a way for mobile devices to preserve and protect their trustworthiness, which is essential in our increasingly interconnected world.

2.2 Replay protection protocol

The replay protection protocol used by the RPMB partition provides a tamper-proof mechanism to ensure the authenticity and integrity of stored data. This protocol begins with an initialization step that establishes the encryption keys for the RPMB partition. These encryption keys are unique to each device and generate a set of secret keys for encryption and decryption of data stored in the partition.

When data is written to the partition, the replay protection protocol generates a Message Authentication Code (MAC) using the Advanced Encryption Standard (AES) algorithm and the set of encryption keys. This MAC is appended to the data before it is written to the RPMB partition. By doing this, the replay protection protocol ensures that the data in the partition is authentic and has not been tampered with during the write operation.

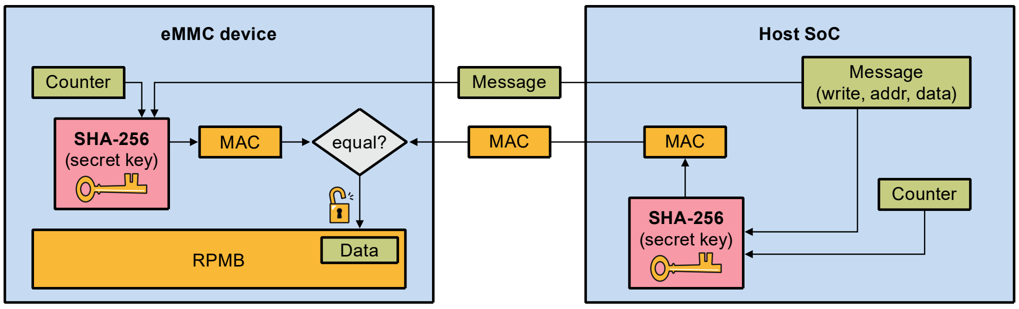

Figure 3. RPMB protect protocol

When data is read from the partition, the replay protection protocol checks the counter to ensure that the data being retrieved is not being replayed. If the counter is valid, the replay protection protocol generates a MAC using the same encryption keys and the AES algorithm that were used for the write operation. The replay protection protocol then compares this MAC with the MAC generated during the write operation to ensure that the data's integrity and authenticity are maintained throughout the read process.

If the RPMB partition is erased, the replay protection protocol is responsible for ensuring that all stored data is correctly erased. The counter and authentication information must also be correctly erased to prevent unauthorized access in the future.

2.3 Message authentication code calculation

The MAC is calculated using HMAC SHA-256. The HMAC SHA-256 calculation takes as input a key and a message. The resulting MAC is 256 bits (32 Bytes) embedded in the data frame as part of the request or response.

The key used for the MAC calculation is always the 256-bit authentication key stored in the eMMC. The message used as input to the MAC calculation is the concatenation of the fields in the data frames excluding stuff bytes, the MAC itself, start bit, CRC16, and end bit. That is, the MAC is calculated over bytes [283:0] of the data frame in that order.

If several data frames are sent as part of one request or response, the input message to MAC is the concatenation of bytes [283:0] of each data frame in the order in which the data frames are sent. The MAC is added only to the last data frame.

Example: Assume that the write counter is 0x12345678. The host wants to write at address 0x0010 two sectors of where the first sector is 256 bytes of 0xAA and the second sector is 256 bytes of 0xBB. The host must then send to following two requests frames where the MAC in the second request frame is an HMAC:

Table 2. MAC example

Start	Stuff Bytes	Key/ (MAC)	Data	Nonce	Write Counter	Address	Block Count	Result	Req/ Resp	CRC16	End
1 bit	196 Bytes	32 Bytes (256b)	256 Bytes	16 Bytes	4 Bytes	2 Bytes	2 Bytes	2 Bytes	2 Bytes	2 Bytes	1 bit
	[511:316]	[315:284]	[283:28]	[27:12]	[11:8]	[7:6]	[5:4]	[3:2]	[1:0]		
Request frame 1											
	0x0000..	0x0000..	0xAA...	0x0000..	0x12345678	0x0010	0x0002	0x0000	0x0003		
Request frame 2											
	0x0000..	MAC	0xBB...	0x0000..	0x12345678	0x0010	0x0002	0x0000	0x0003		

SHA-256 is calculated over the following message:

0xAA... (total 256 times) ... 0xAA

0x0000 0x0000 0x0000 0x0000 0x0000 0x0000 0x0000 0x0000

0x1234 0x5678 0x0010 0x0002 0x0000 0x0003

0xBB... (total 256 times) ... 0xBB

0x0000 0x0000 0x0000 0x0000 0x0000 0x0000 0x0000 0x0000

0x1234 0x5678 0x0010 0x0002 0x0000 0x0003

The key used for the HMAC SHA-256 calculation is the authentication key stored in the eMMC.

3 Access to RPMB

Note: *The EVK boards do not have eMMC. This document uses EVB to enable this feature.*

Warning: *The RPMB authenticate key can only be programed once.*

In the RT chips, after initializing the MMC card, it will enter the user partition by default. The first step is to use the "**MMC_SelectPartition**" function to switch the partition to RPMB.

Before programing the authentication key to RPMB, it is recommended to use "**mmc_rpmb_read**" function to check if RPMB works right. As the key is not programmed, the response frame contains the error message code that indicates "**Authentication key not yet programmed**".

The example code includes two ways of programming key that are controlled by macro "**PLAIN_KEY**". The first is the plain key. Users can directly set the key in the "**rpmb_key**" buffer. After programming key to RPMB, store the key. The second way is more secure. The key is generated by the CAAM RNG driver. After programming the key to RPMB, the key is encapsulated to a blob by the "**BlobEnc**" function. Users must store the blob. To use the key to write or read the partition, decapsulate the blob by the "**Blobdec**" function.

After programming the key to RPMB, users can read and write RPMB partition by the “**mmc_rpmb_read**” and “**mmc_rpmb_write**” functions. The counter must be set properly during writing.

4 Software enablement

The RPMB functionality can be enabled using an attachment that contains the needing C source files. Replace the file that contains SDK. Below is the basic RPMB request and the response code. See this AN’s software attachment for details.

```
status_t mmc_rpmb_request(mmc_card_t *card, rpmb_frame_t *s, unsigned int count,
 bool is_rel_write)
{
    assert(card != NULL);

    sdmmchost_cmd_t command      = {0};
    sdmmchost_transfer_t content = {0};
    sdmmchost_data_t data        = {0};
    status_t error               = kStatus_Success;

    /* Legacy mmc card , do not support the command */
    if ((card->csd.systemSpecificationVersion ==
 (uint32_t)kMMC_SpecificationVersion3) &&
        (card->csd.csdStructureVersion == (uint32_t)kMMC_CsdStrucureVersion12))
    {
        return kStatus_Success;
    }
    error = SDMMC_SetBlockCount(card -> host, count, is_rel_write);

    if(kStatus_Success != error)
    {
        PRINTF("mmc set block count fail!!\r\n");
        return error;
    }

    command.index        = (uint32_t)kSDMMC_WriteMultipleBlock;
    command.argument     = 0U;
    command.responseType = kCARD_ResponseTypeR1;

    data.txData = (const uint32_t *)s;
    data.blockCount = count;
    data.blockSize = MMC_MAX_BLOCK_LEN;

    content.command = &command;
    content.data    = &data;
    error           = SDMMCHOST_TransferFunction(card->host, &content);

    if(kStatus_Success != error)
    {
        PRINTF("mmc write multiple block fail!!\r\n");
        return error;
    }
    return kStatus_Success;
}

status_t mmc_rpmb_response(mmc_card_t *card, rpmb_frame_t *s, unsigned int
 count, unsigned short expected)
{
    assert(card != NULL);
    sdmmchost_cmd_t command      = {0};
    sdmmchost_transfer_t content = {0};
    sdmmchost_data_t data        = {0};
    status_t error               = kStatus_Success;

    /* Legacy mmc card , do not support the command */
    if ((card->csd.systemSpecificationVersion ==
  (uint32_t)kMMC_SpecificationVersion3) &&
        (card->csd.csdStructureVersion == (uint32_t)kMMC_CsdStrucureVersion12))
    {
        return kStatus_Success;
    }

    error = SDMMC_SetBlockCount(card -> host, count, false);

    command.index        = (uint32_t)kSDMMC_ReadMultipleBlock;
    command.argument     = 0U;
    command.responseType = kCARD_ResponseTypeR1;

    data.rxData = (uint32_t *)s;
    data.blockCount = count;
    data.blockSize = MMC_MAX_BLOCK_LEN;

    content.command = &command;
    content.data    = &data;
    error           = SDMMCHOST_TransferFunction(card->host, &content);

    if(kStatus_Success != error)
    {
        PRINTF("mmc read multiple block fail!!\r\n");
        return error;
    }

    if (expected && be16_to_cpu(s->request) != expected) {
        PRINTF("ERROR: request command not match!!\r\n");
        return kStatus_Fail;
    }

    /* Check the response and the status */
    if (be16_to_cpu(s->result)) {
            PRINTF("%s %s\n", rpmb_err_msg[be16_to_cpu(s->result) &
 RPMB_ERR_MSK],
                  (be16_to_cpu(s->result) & RPMB_ERR_CNT_EXPIRED) ?
                  "Write counter has expired" : "");
    }

    /* Return the status of the command */
    return kStatus_Success;
}
```

Note: *These functions only work correctly when the device is configured for the eMMC. Use the "**AccessCard**" function to check if host can access eMMC successfully before switching to RPMB.*

5 References

The following documents may offer further reference.

- *Embedded Multi-Media Card (eMMC) Electrical Standard (5.0)* (document JESD84-B50)

- *Reference Manual for the i.MX RT1170 Processor* (document i.MX RT1170 RM)

6 Revision history

Table 3 summarizes the changes since the initial release.

Table 3. Revision history

Revision number	Date	Substantive changes
1	15 June 2023	Initial release

7 Legal information

7.1 Definitions

7.2 Disclaimers

7.3 Trademarks

Contents

Date of release: 15 June 2023
Document identifier: AN_RPMB_MCU