Understanding and Using eMMC Replay Protected Memory Block (RPMB)

Author: Zhi Feng
Associated Part Family:S40FCxxx

1. Introduction

Modern embedded systems demand secure, tamper-resistant storage for sensitive information such as cryptographic keys, counters, and authentication tokens. The Replay Protected Memory Block (RPMB) feature of eMMC devices provides a secure partition with replay protection, integrity verification, and access control.

RPMB was introduced in JEDEC eMMC version 4.4. In eMMC devices that comply with JEDEC 4.4 or higher versions, the RPMB partition should be readily available.

This application note introduces the RPMB concept, describes the basic setup process, outlines common use cases, and provides guidelines for system integration. The document assumes that the readers have basic knowledge of eMMC operations, as well as security algorithms such as symmetric keys, SHA (Secure Hash Algorithm), and MAC (Message Authentication Code). This document also uses Linux OS as an example to show how the host interacts with the eMMC device for accessing the RPMB partition.

2. RPMB Overview

Typically, an eMMC device has boot partitions, user data areas and one RPMB partition. The RPMB is a special partition in the eMMC device designed for secure data storage.

Figure 1: eMMC partitions

To access the RPMB partition, the host must first install a secret key (typically a 32-byte random number) to the RPMB, then make use of the key to perform authenticated transactions to access the RPMB. Figure 2 shows a general flow of the authenticated transactions.

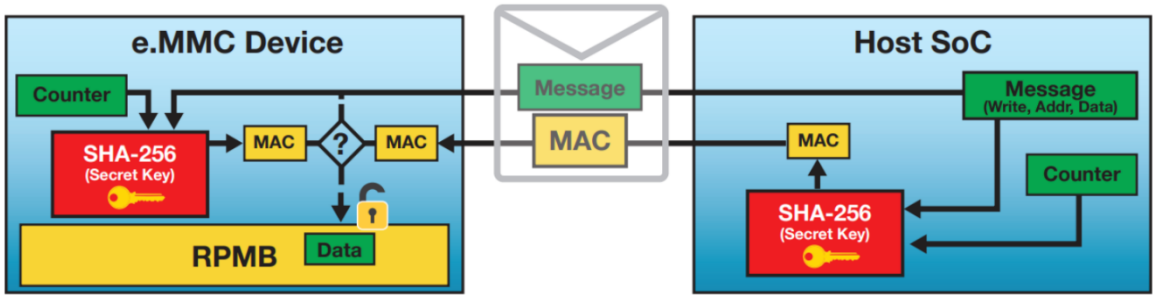

Figure 2: RPMB Authenticated Transactions

When initiating a transaction, the host uses SHA-256 algorithm to generate a MAC from the message payload to be transmitted. The value of the monotonic counter, which should be synchronized with the eMMC device, is included in the MAC calculation. This MAC is verified at the device side using the same key to ensure authenticity. The details of the operations are discussed in the following sections of this document.

Unlike normal user data areas:

- RPMB is not directly addressable by standard eMMC read/write commands.
- Access is performed using a frame-based protocol via standard eMMC commands (CMD23, CMD25,

CMD18, etc.).

- Each RPMB operation uses a 512-byte frame, containing fields such as address, data, nonce, write counter, MAC, and result code.
- Data integrity and authenticity are ensured using a symmetric authentication key and HMAC (SHA-256).

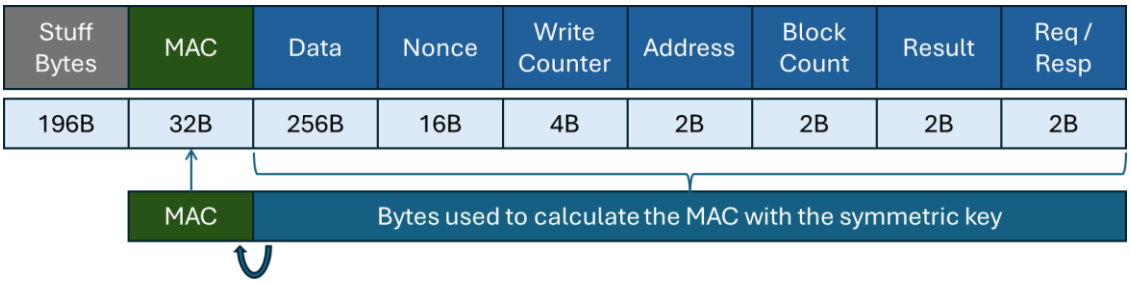

Figure 3: Example of an RPMB data frame

The last 2B in the RPMB frame indicate the Request or the Response type of the RPMB operations. JEDEC84-B51 defines all request and response types in Table 18.

Table 18 — RPMB Request/Response Message Types

Request Message Types	
0x0001	Authentication key programming request
0x0002	Reading of the Write Counter value -request
0x0003	Authenticated data write request
0x0004	Authenticated data read request
0x0005*	Result read request
0x0006	Authenticated Device Configuration write request
0x0007	Authenticated Device Configuration read request
Response Message Types	
0x0100	Authentication key programming response
0x0200	Reading of the Write Counter value -response
0x0300	Authenticated data write response
0x0400	Authenticated data read response
0x0500	Reserved
0x0600	Authenticated Device Configuration write response
0x0700	Authenticated Device Configuration read response
NOTE There is no corresponding response type for the Result read request because the reading of the result with this request is always relative to previous programming access.	

To initiate or respond to an RPMB operation, the host or the eMMC device must construct the frame according to the definition. The MAC field is calculated as follows, with all data fields in the frame:
MAC = HMAC(Key, data+nonce+write_counter+address+block_count+result+request/response)

To simplify the user software implementation, latest embedded Linux systems have included mmc drivers that support RPMB operations. Users can just invoke standard driver APIs to access the RPMB partition. The open source code, mmc_utils, that exists in standard Linux distributions, is used in this document to show how basic RPMB operations are done.

3. RPMB Operations

3.1 Initial Provisioning

Users may verify the RPMB partition size by reading the eMMC EXT_CSD[168]. If the field value is non-zero, RPMB partition is available. The value ranges from 0x01(1x128KB) to 0x80 (128x128KB=16MB). Current RPMB size is limited to maximum 16MB due to the address field inside the RPMB frame is limited to 2 bytes. All SkyHighMemory eMMC devices have an RPMB partition with 4MB size.

On Linux systems, once the eMMC is initialized, the RPMB partition is seen under /dev/. For example:

```
Linux >ls /dev/mmcblk0*
/dev/mmcblk0  /dev/mmcblk0boot0  /dev/mmcblk0boot1  /dev/mmcblk0p1  /dev/mmcblk0rpmb
```

The RPMB partition requires an initial provisioning of the authentication key for normal operations. This key must be created with a cryptographically strong random generator, unique per device, and securely programmed into the eMMC and host at manufacturing time, never exposed in plaintext thereafter. Typically, this step is done inside a secure facility.

The RPMB key is 32-byte in length. After preparing the key, the user can use the following command to install the key into the RPMB partition. Note that this is a one-time operation. No reversal is allowed. The example below uses openssl to generate the key and shows how the key is installed.

```
Linux > openssl rand 32 > key_file
```

```
Linux > mmc rpmb write-key /dev/mmcblk0rpmb key_file
```

If the key injection is successful, the initial provisioning step is done. The RPMB is ready for authenticated read and write operations.

The following diagram shows the RPMB frame flows for a key injection operation:

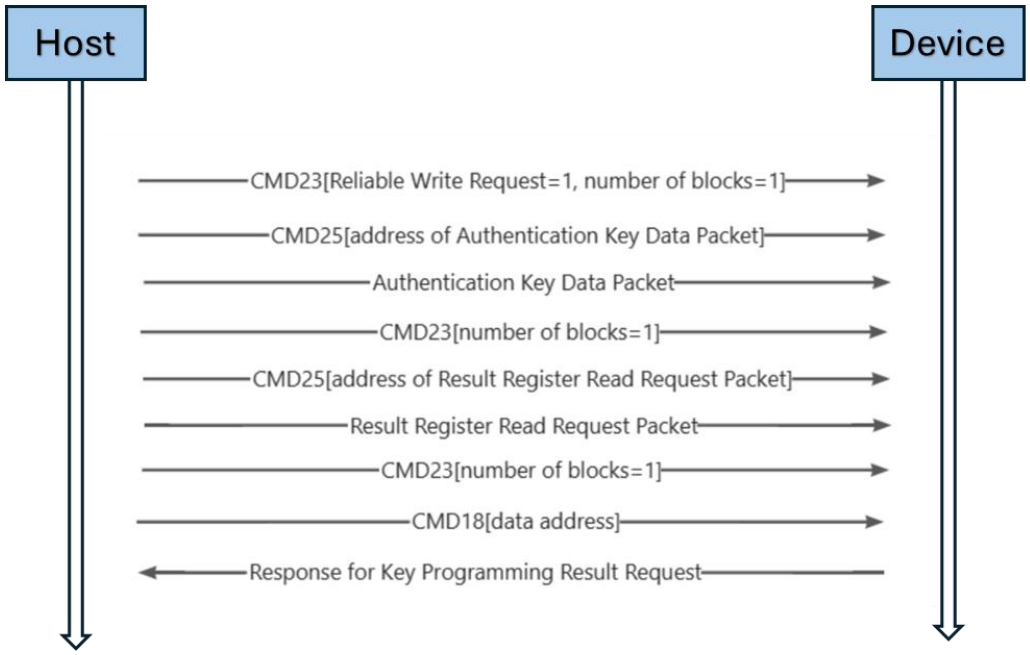

Figure 4: Example of RPMB Key Programming

3.2 Authenticated Write Operations

Within the RPMB partition, memory is organized into blocks, 256B each in size. Users can select any block to write into. No erase function is provided, but authenticated users can modify the data by writing new data into the same block.

To do a write operation, the host must possess the correct key, obtain the current monotonic counter value from the eMMC, then prepare the data frame to write to the RPMB partition. If the data are to be written to multiple blocks, the host can send multiple frames (CMD25) in one write operation. In this case, only the last frame should contain the valid MAC, which is calculated from data concatenated from all frames. Prior frames should have 0 values in the MAC field. All frames should have the same write counter value. The entire multi-block write operation is treated as a single write operation. That means the counter will be incremented by 1 if the write is successful.

The following command sequence on Linux shows an example of write operations.

```
Linux > mmc rpmb read-counter /dev/mmcblk0rpmb
Counter value: 0x00000003
Linux > mmc rpmb write-block /dev/mmcblk0rpmb 0 payload_data key
Linux > mmc rpmb read-counter /dev/mmcblk0rpmb
Counter value: 0x00000004
```

Because the value of the monotonic counter increases by 1 after each successful write operation, the same data frame cannot be recorded and replayed at a later time; therefore, the RPMB contents are protected from replay attacks.

Reading the monotonic counter value is not a secure operation. The host can synchronize the counter with the eMMC device by retrieving the counter value before a write operation. The following diagrams show the RPMB frame flows for a read counter operation and an authenticated write operation:

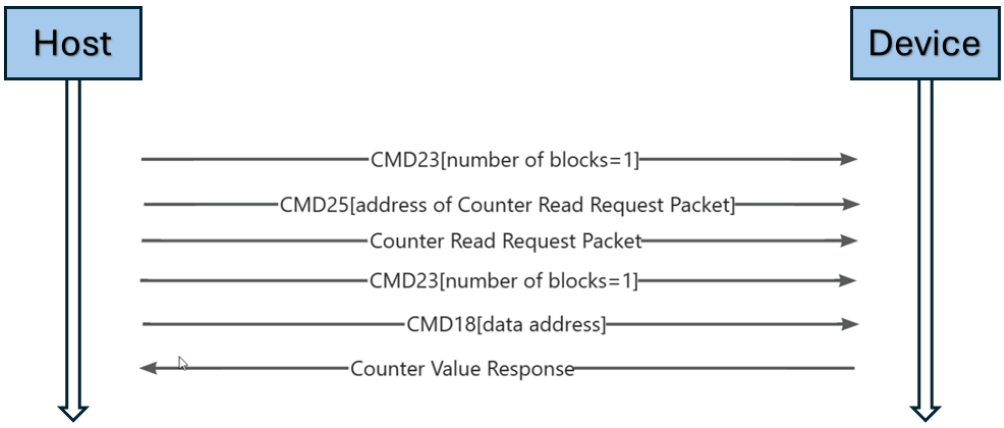

Figure 5: Example of RPMB Read Counter

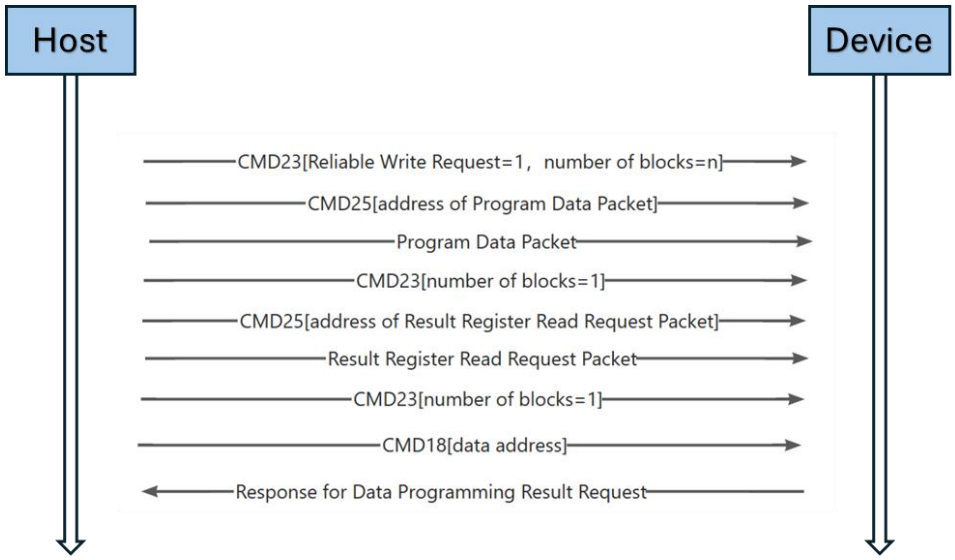

Figure 6: Example of RPMB Authenticated Write Operation

3.3 Read Operations (Authenticated or not)

The RPMB partition supports authenticated and non-authenticated read operations. That means the host can choose to read out data from the RPMB with or without the key. When using the key, the host can validate the authenticity, as well as the integrity of the data. When not using the key, the RPMB still outputs the data, but the authenticity is not guaranteed.

```
Linux > mmc rpmb read-counter /dev/mmcblk0rpmb
Counter value: 0x00000005
Linux > mmc rpmb read-block /dev/mmcblk0rpmb 5 1 - key
This is to test RPMB.
Linux > mmc rpmb read-block /dev/mmcblk0rpmb 5 1 -
This is to test RPMB.
Linux > mmc rpmb read-counter /dev/mmcblk0rpmb
Counter value: 0x00000005
```

Note that the value of the monotonic counter does not change after a read operation. The host can choose to provide the key or not when reading.

The following diagram shows the RPMB frame flow for a read operation:

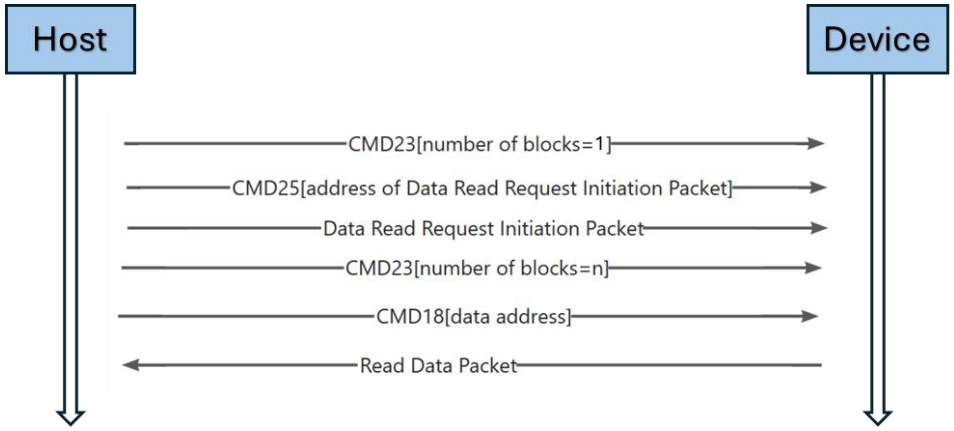

Figure 7: Example of RPMB Read Operation

Unlike the sequence flow from the write operation, the first CMD23 in a read sequence always sets the number of blocks to 1, because the host will only send one request frame to the eMMC device, even in multi-block read situations. For MAC calculation, the read operation follows the same principle. In case of multi-block reads, only the last frame from the device will contain the valid MAC, which is calculated over the concatenated data over all frames.

4. Practical Usage of RPMB

As the data in RPMB can be authenticated and protected from replay attacks, RPMB has many useful real-life applications. For example:

1. Secure key storage: RPMB can be used to store cryptographic keys, certificates, or license tokens, as the contents from RPMB can be authenticated by the host.
2. Replay-Protected counters: The monotonic counter of the RPMB can be used to keep track of critical applications that are not supposed to rollback, such as software/firmware versions, automotive mileage info, etc.
3. Trusted boot: RPMB can be used to store hash values of boot code so that the code can be validated before running. These hash values are protected from tampering inside the RPMB.
4. Sensitive logging data: RPMB provides tamper-resistant logging for sensitive data such as automotive mileage, accident logs, and maintenance information.

5. Implementation Considerations

1. Key management: The RPMB key must be securely provisioned, usually during manufacturing in a secure facility, and never exposed in plaintext.
2. Performance: RPMB access is slower than normal eMMC operations due to authentication and HMAC processing. It is designed for small data objects, not bulk storage.
3. Error recovery: Always verify the result code after each operation. Design systems to handle cases such as “key not programmed” or “authentication failure.”

6. Summary

The eMMC RPMB partition provides a secure and replay-protected storage area essential for modern embedded applications. With careful provisioning and proper integration, RPMB enables robust protection of keys, counters, and sensitive data, making it a useful tool of secure system design. SkyHighMemory eMMC devices, which comply with JEDEC eMMC 5.1 standard, provide RPMB functionality to satisfy customers’ requirements of secure storage.

7. References

JESD84-B51, Embedded Multi-Media Card (eMMC) Electrical Standard (5.1)

8. Revision History

Document Title: AN200010 - Understanding and using e.MMC Replay Protected Memory Block (RPMB) Document Number: 002-00010				
Rev.	**ECN No.**	**Orig. of Change**	**Submission Date**	**Description of Change**
**	–	–	09/08/2025	Initial version

Trademarks and Notice

The contents of this document are subject to change without notice. This document may contain information on a SkyHighMemory product under development by SkyHighMemory. SkyHighMemory reserves the right to change or discontinue work on any product without notice. The information in this document is provided as is without warranty or guarantee of any kind as to its accuracy, completeness, operability, fitness for particular purpose, merchantability, non-infringement of third-party rights, or any other warranty, express, implied, or statutory. SkyHighMemory assumes no liability for any damages of any kind arising out of the use of the information in this document.

Copyright © 2025 SkyHighMemory All rights reserved.