everyday genius

MTK RPMB External User Guide

Version:	0.62
Release date:	2018-12-13

Specifications are subject to change without notice.

Document Revision History

Revision	Date	Author	Description
0.1	2015-01-12	Cool Lee	Initial draft
0.2	2015-02-02	Cool Lee	Initial draft. Add more chapters
0.3	2015-02-09	Cool Lee	Describe detail in CH1, CH3, CH7 and CH11.
0.4	2016/2/28	Cool Lee	Add GP API. Introduce in chapter 14, 15, 16.
0.5	2016/3/4	Cool Lee	GP API describe in detail.
0.6	2017/6/23	Light Hsieh	Update to reflect path change for Android N; Add chapter 19
0.61	2017/7/17	Light Hsieh	Update to reflect path change for hal GP Android N; Update chapter 19 for Android N
0.62	2018/1/24	Light Hsieh	Update chapter 14 to add description for 2 APIs
0.63	2018/12/13	Chun-Hung Wu	Update chapter 19 for Android P change

Table of Contents

Lists of Tables and Figures

1 INTRODUCTION

This chapter will introduce software architecture and the design guide including kernel driver and secure driver.

1.1 Concept

The concepts show as below indicates the flow of RPMB access with TEE environment. There is a normal world RPMB driver created as a thread which waiting a request from TEE RPMB driver and then return result.
We will talk about their missions of each block later.

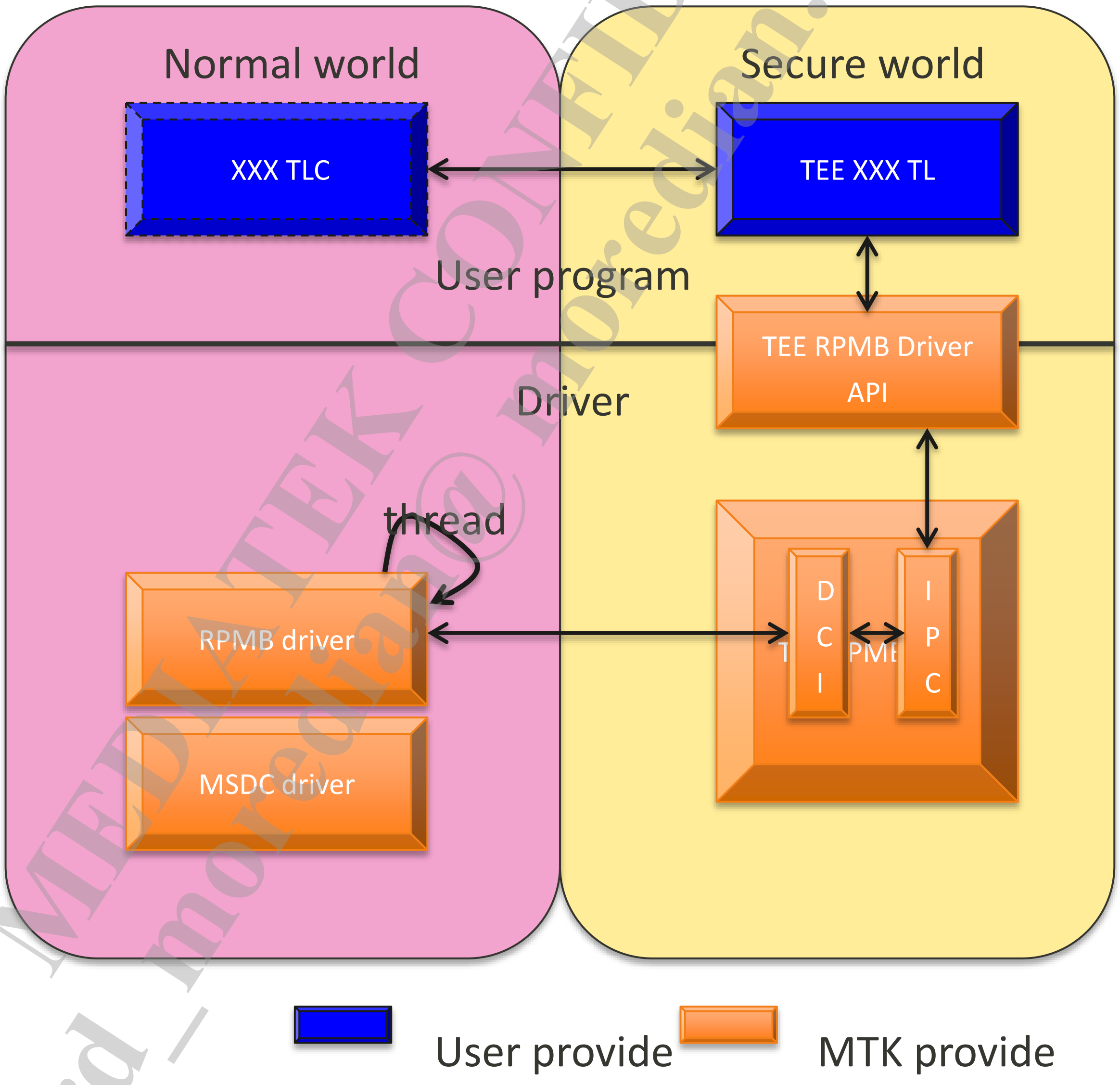


Figure 1-1. Block diagram

1.2 Program a key

RPMB Key is generated by CHIP ID and several hardware IDs within preloader period. In preloader, it pass key to TEE for later use. Before programming a key to emmc, the result error code 0x02 or 0x82 (MAC authentication fail) of getting write counter can let us know whether we should stop program. To avoid error retry to keep the key secret. Although we only program key at the first boot, it also has a risk of losing key by capturing emmc bus signals.

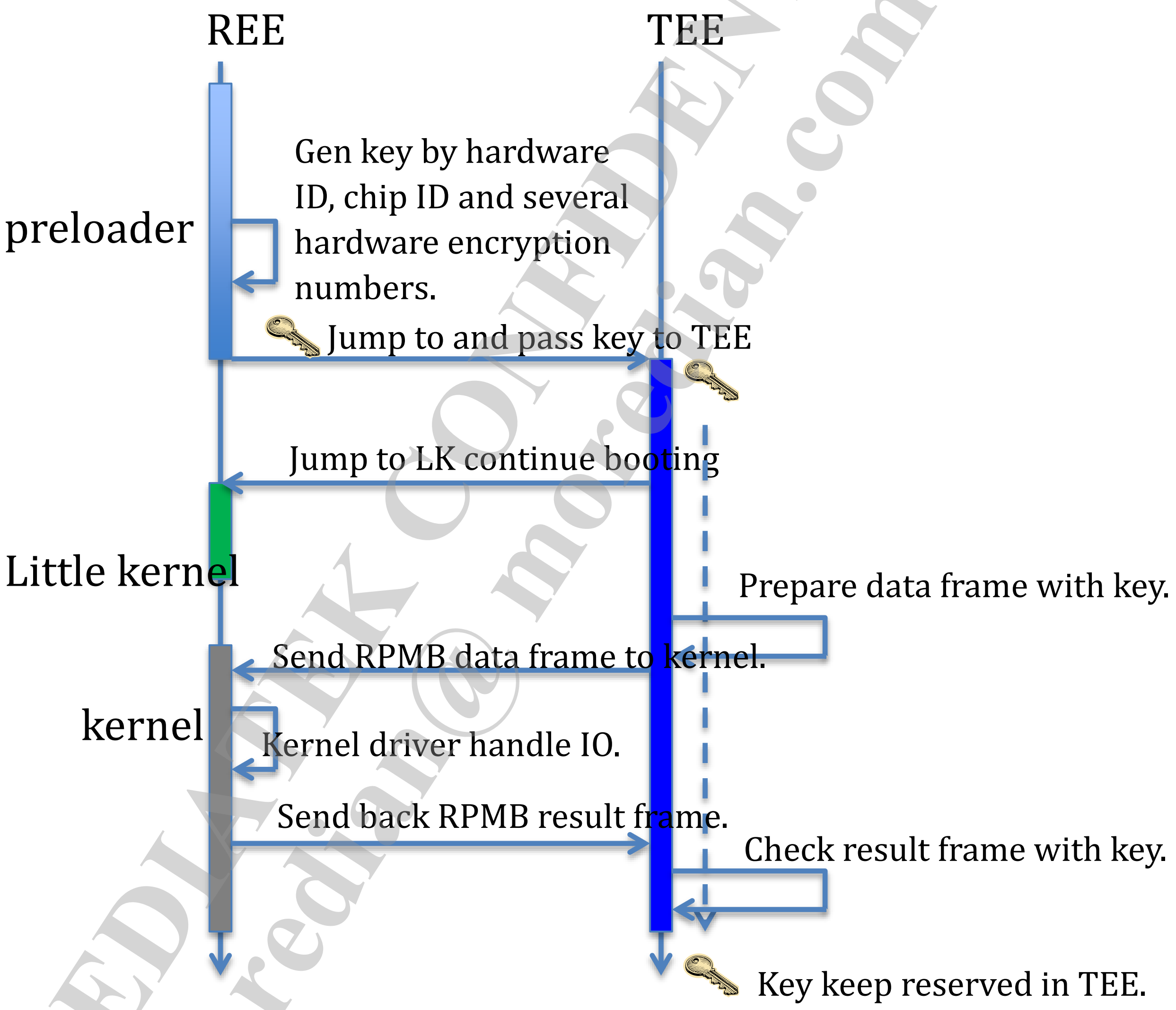

REE
TEE
preloader
Gen key by hardware ID, chip ID and several hardware encryption numbers.
Jump to and pass key to TEE
Jump to LK continue booting
Little kernel
Prepare data frame with key.
Send RPMB data frame to kernel.
kernel
Kernel driver handle IO.
Send back RPMB result frame.
Check result frame with key.
Key keep reserved in TEE.

Figure 1-2. Key usage flow

The key is not saved in the storage; it was generated by the computation according to the fixed numbers. Once this emmc was erased, preloader also can generate a same Key again when booting.

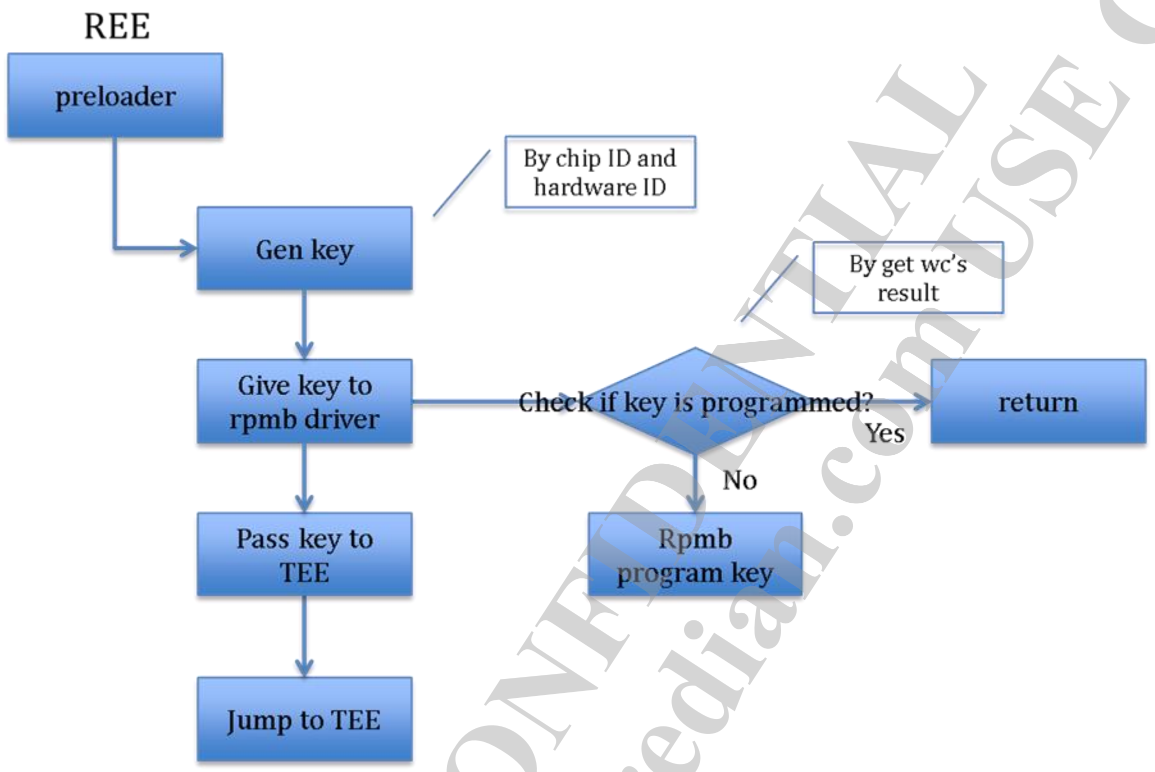


Figure 1-3. Preloader set key flow

1.3 Write Procedure

Now, I will present you the detail write procedure and how it works between TEE and kernel.

1. Kernel RPMB driver is registered as a character device. It contained a general ioctl interface that user space program can use. After this driver is called, it creates a thread and keeps listening signals from DCI channel. We'll talk about this driver detail later.
2. TLC sends requests to its TL through their APIs. Perhaps we don't need a TLC as well.
3. TL calls tlApiRpmbOpenSession/tlApiRpmbReadData/tlApiRpmbWriteData to notice driver we will start read or write.
4. Secure RPMB driver API prepare read or write data frame and send to IPC thread.
5. IPC thread copied data frame to dciMessage_t which is a share buffer allocated by kernel driver.
6. DCI notify kernel RPMB driver.
7. Kernel RPMB driver received a notification from DCI, and handled this IO then return a response data frame to DCI.
8. DCI do nothing and just return to IPC.
9. IPC thread copied write result response data frame to driver API.
10. Driver API authenticated write result response.

On the above steps, only step 2, 3 and 10 are user's responsibilities. Step 10 means you should do error handling by yourself.

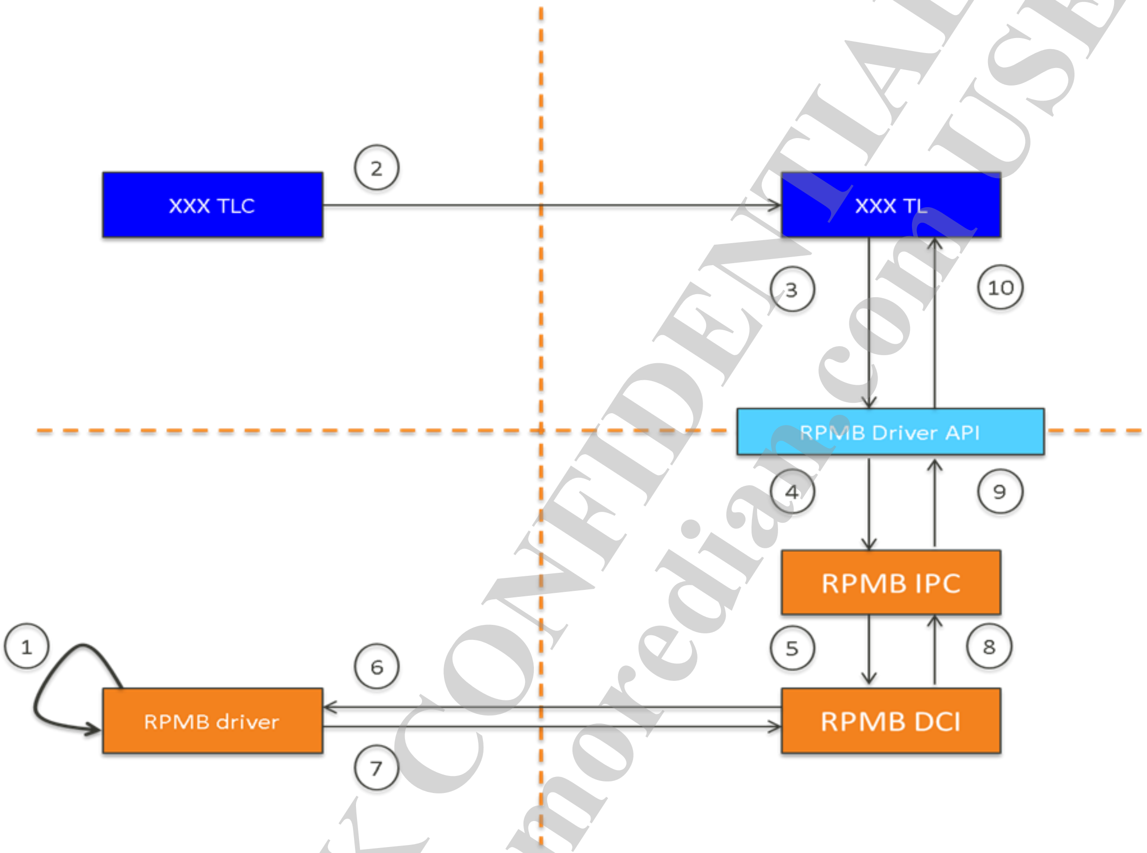


Figure 1-4. RPMB write procedure with TEE.

2 STRUCTURE

This chapter will show you RPMB driver package, including file structures and data structures.

2.1 File Structures

There are three RPMB packages in the TEE, containing drv, ta and tlc. tlc and ta were sample codes for the demonstration. They were put under the path alps/vendor/mediatek/proprietary/trustzone/common/hal/source/trustlets/common/. There is an independent kernel driver in normal world, but it will include drrpmb_Api.h which is copied from TEE to kernel path under alps/kernel-4.9/drivers/mmc/mediatek/rpmb/public/.

tz_mem.h will be included by rpmb_ops.c since we passed the RPMB key from preloader and we need retrieve the key in the secure driver with the same structure.

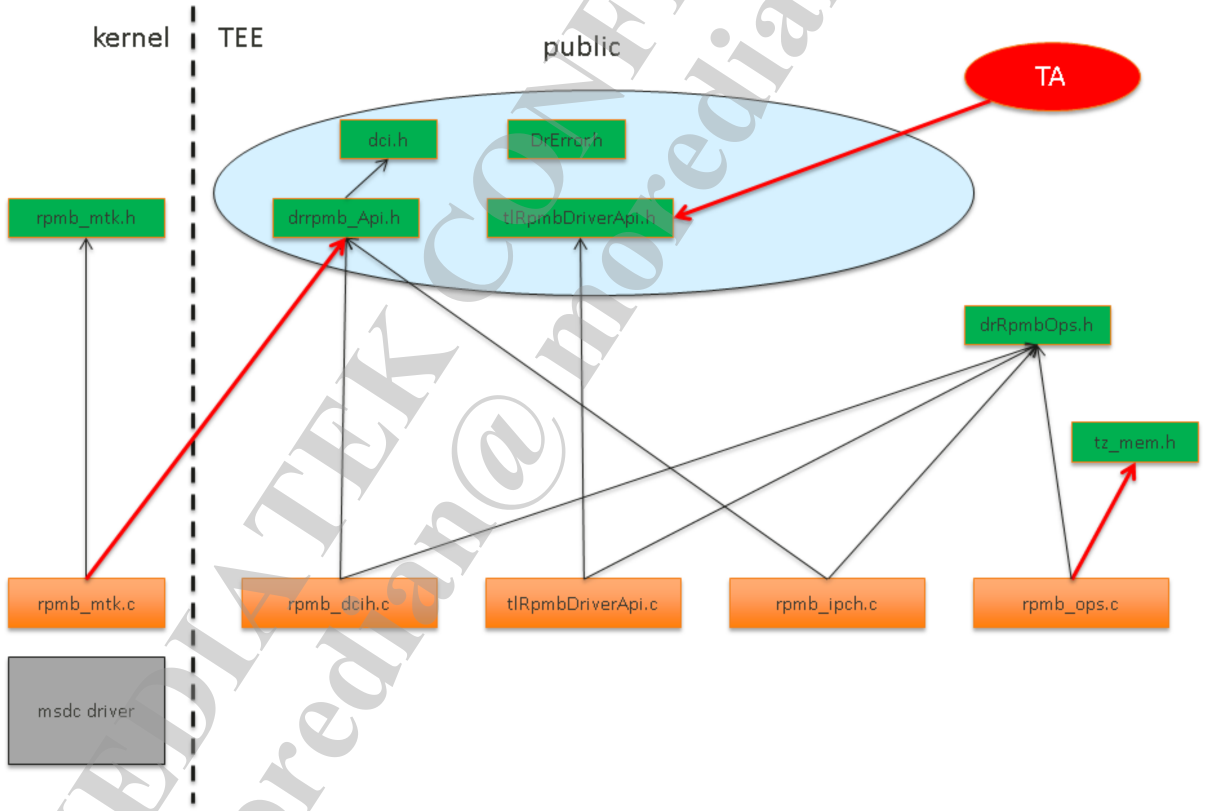


Figure 2-1. File structure

rpmb_mtk.c/rpmb_mtk.h: alps/kernel-4.9/drivers/char/rpmb

public: alps/kernel-4.9/drivers/mmc/mediatek/rpmb/public
alps/vendor/mediatek/proprietary/trustzone/common/hal/source/trustlets/common/drv/public

tz_mem.h: alps/bootable/bootloader/preloader/platform/mt67563/src/security/trustzone/inc alps/vendor/mediatek/proprietary/trustzone/common/hal/source/trustlets/common/drv/inc

2.2 Data Structures

In TEE RPMB driver packages, we had two interfaces, one is tlApiRpmb_t used for TL driver API talk to IPC thread and another for dciMessage_t which is communicated with kernel driver.

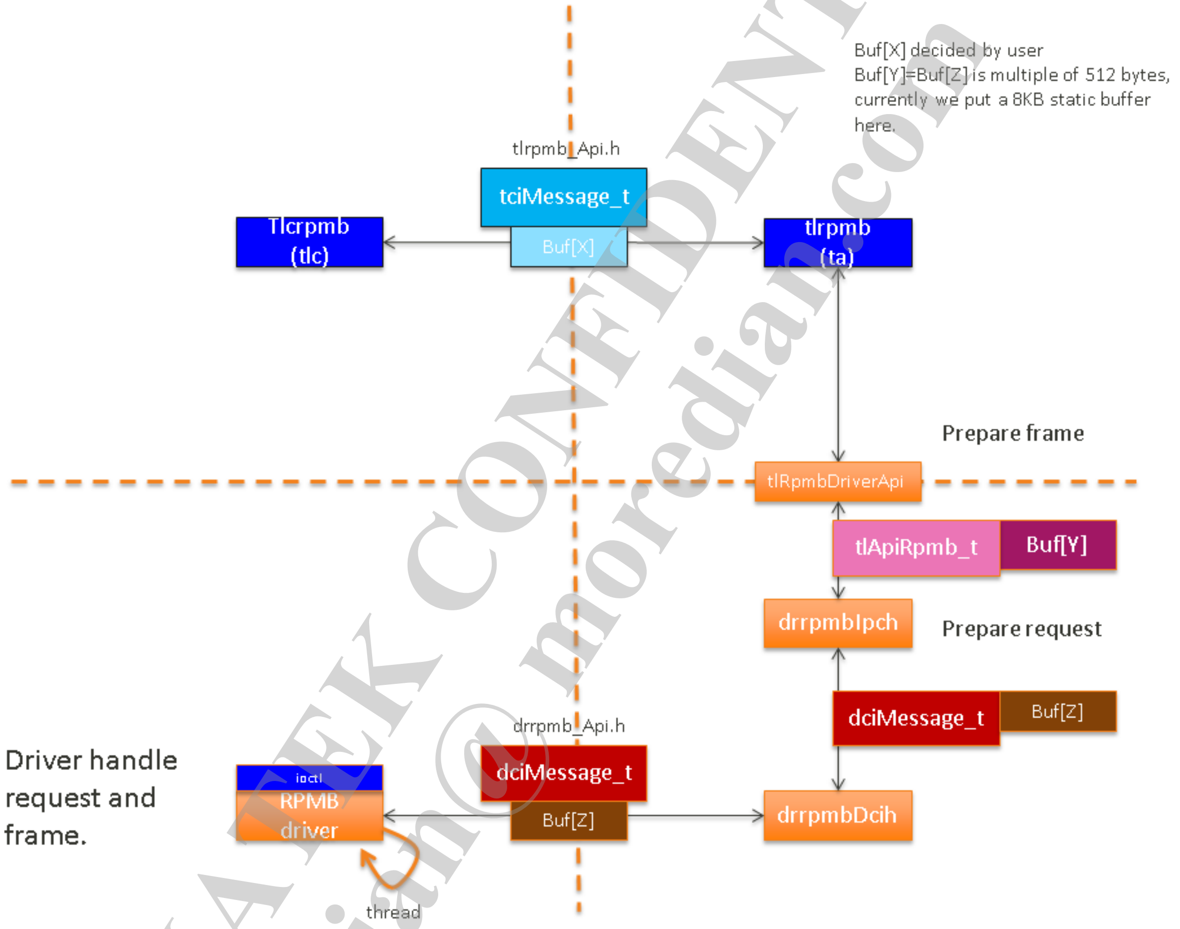


Figure 2-2. Data structure

3 DRIVER ARCHITECTURE

There are two parts in this chapter. Secure world TEE and normal world RPMB driver.

3.1 TEE RPMB Driver Introduction

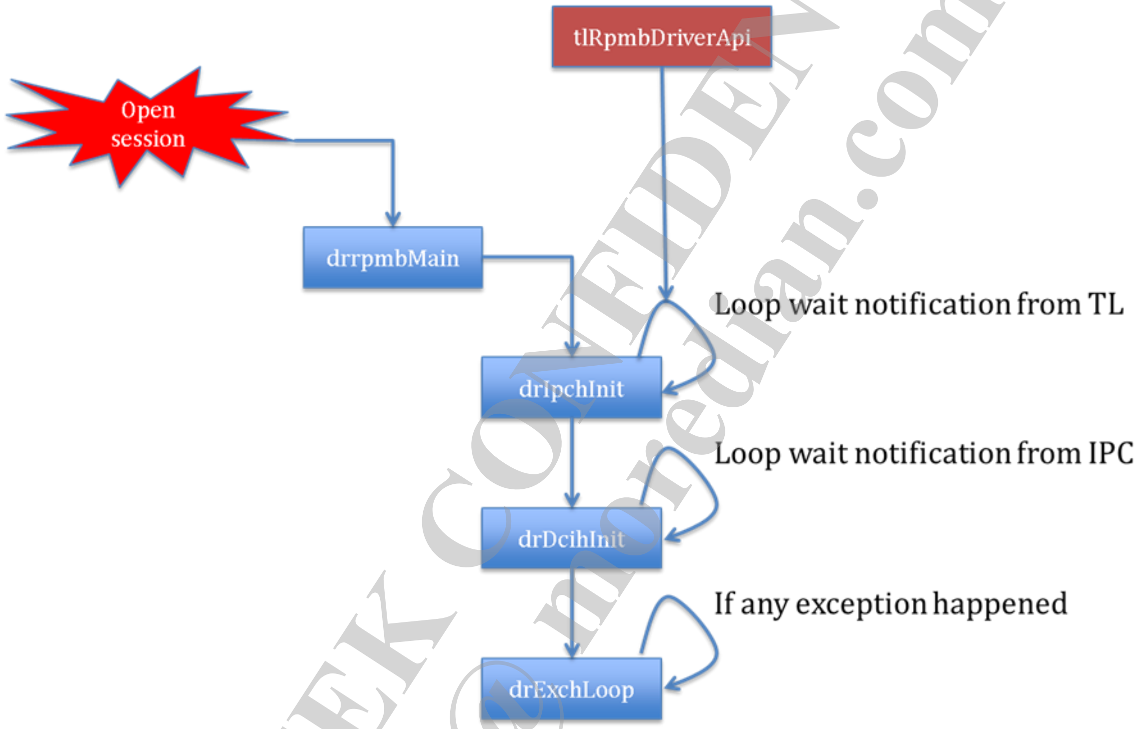

tlRpmbDriverApi
Open session
drrpmbMain
drIpchInit
Loop wait notification from TL
drDcihInit
Loop wait notification from IPC
drExchLoop
If any exception happened

Figure 3-1. TEE RPMB driver concept

3.1.1 TL RPMB driver API

Firstly, users require permission by calling **tlApiRpmbOpenSession** to get a session ID. With session ID, you can perform RPMB read/write by calling **tlApiRpmbReadData** and **tlApiRpmbWriteData**.

Users perform read and write behavior through this layer. The objective of this layer is to compose a request and call to driver IPC or use trustonic trustlets API to perform SHA256 HMAC computation. There is an internal function named tlApiHmacSha256 handle this.

For example, tlApiRpmbReadData steps as below.

1. Call to Driver for requiring a key for MAC calculation.
2. Call to Driver for requiring a block address since the partition management handled in the driver. We also need an address to consist the data frame and compute the HMAC.
3. According to user's buffer size, we separate to each 512 bytes data frame.

4. Call to Driver for handling READ DATA request. Before that, we construct a data frame of read data request.
5. According to the block count in the request data frame, we will receive the number of block count data block. Each block has its own HMAC and we need to authenticate it one by one. (This is the behavior when RPMB_MULTI_BLOCK_ACCESS option is disabled)

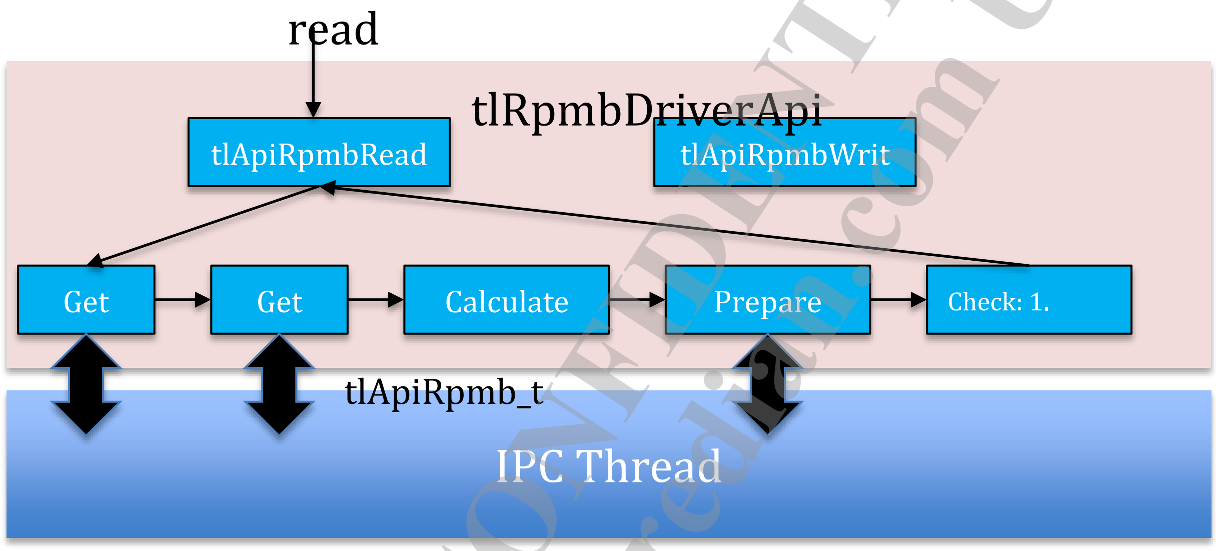


Figure 3-2. Flow of tlApiRpmbReadData

Following is an example of write procedure.

1. Same as read, call to driver to get a key first.
2. Same as read, call to driver to get a block address.
3. For write, getting a write counter is the first step of this protocol. Prepare the request data frame consist of a nonce generated by calling trustlet API and call to driver.
4. When returned, authenticate its MAC, nonce and result. Result must be no error.
5. If getting write counter is successful, continue to prepare write data frame. Now, we need address, write counter, block count and finally we compute HMAC from above then attach to data frame. Once ready, call to driver.
6. When returned, write result response is come back. Authenticate its HMAC, result and check write counter is same as above or not. Check write counter to avoid hacker replace an old legal data frame to cheat us.

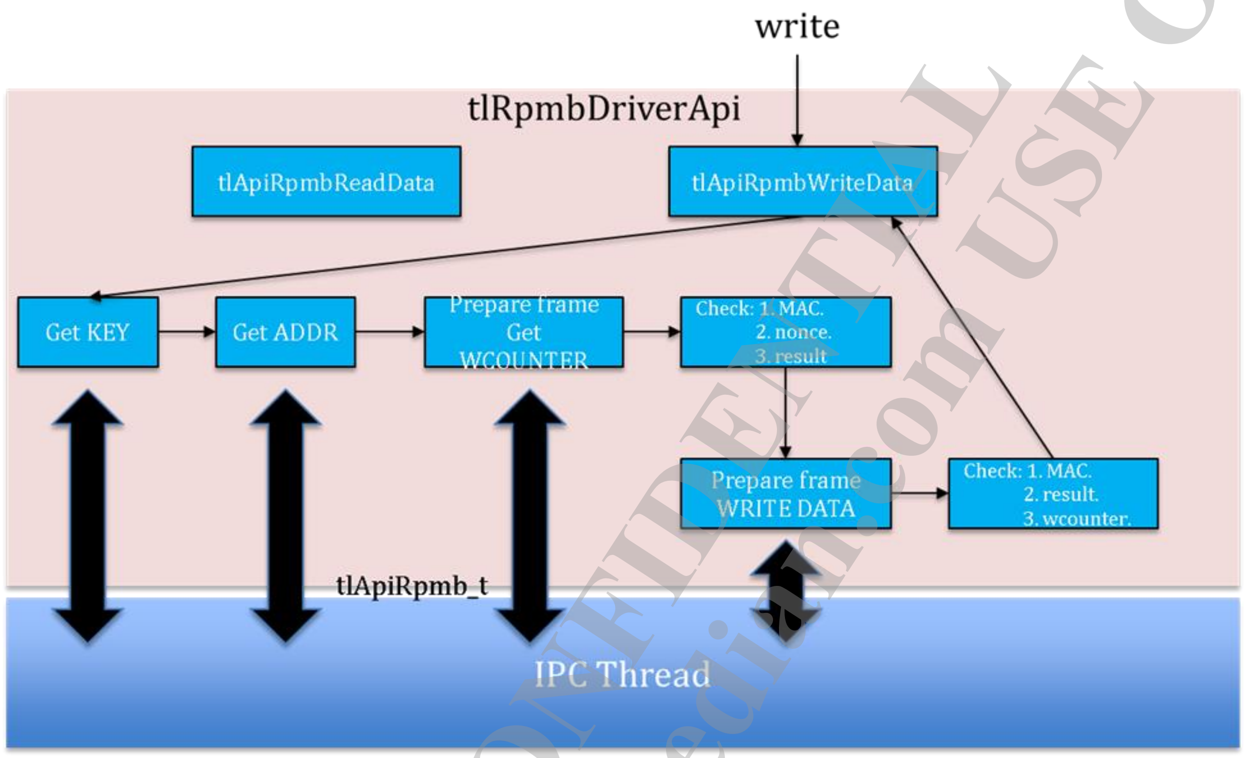


Figure 3-3. Flow of tlApiRpmbWriteData

3.1.2 Multiple Block Access

Here, we provide an alternative way to perform RPMB access. An option named **RPMB_MULTI_BLOCK_ACCESS** defined in drrpmb_Api.h used to control whether to enable multiple block access. To enable it, driver declares an internal 8KB buffer to improve transfer throughput. According to spec, MAC has to append to the last data frame so that we need to aggregate each frame's data then send to SHA256 computation. This is more efficiency, but cost memory.
If this option is enabled, pay attention to enlarge your TL's heap and stack size to more than 8KB. The suggestion is 16KB.

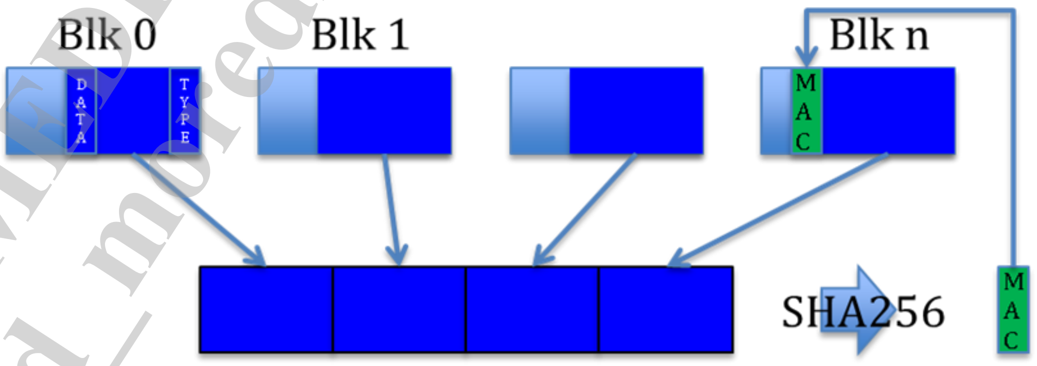


Figure 3-4. MAC on multiple block RPMB access.

The following diagram shows transferring 32KB data with **RPMB_MULTI_BLOCK_ACCESS** enabled. Since we have an internal 8KB buffer, the total transaction is reduced to 4 times. Each time just perform one CMD23 and CMD25. One block only contains 256 bytes data so that 32 blocks indicate to 8KB. We will loop 4 times until total 32KB data were transferred.

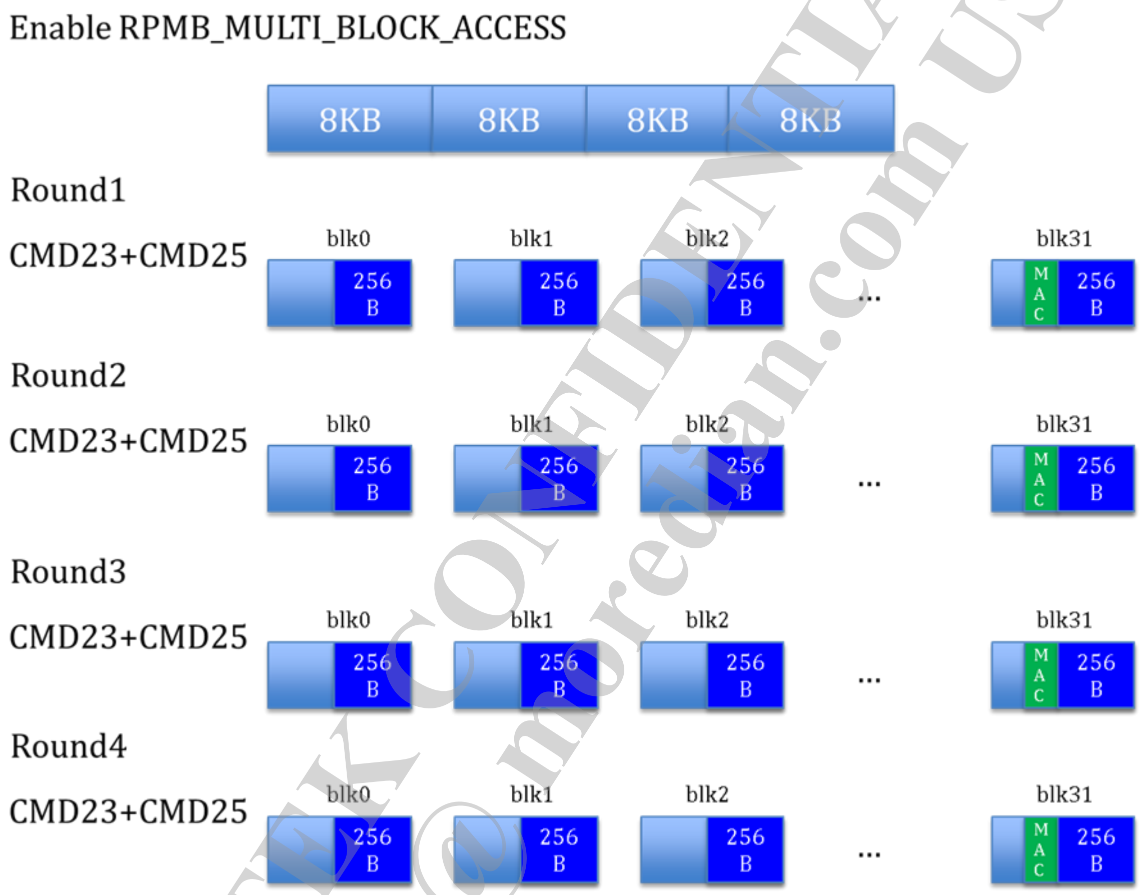


Figure 3-5. Example when enable RPMB_MULTI_BLOCK_ACCESS

On the other hand, transferring 32KB data with **RPMB_MULTI_BLOCK_ACCESS** disabled requires loop 32 times. In each iteration, only one blocks is transferred and each block require its own MAC calculation as shown in the following diagram.

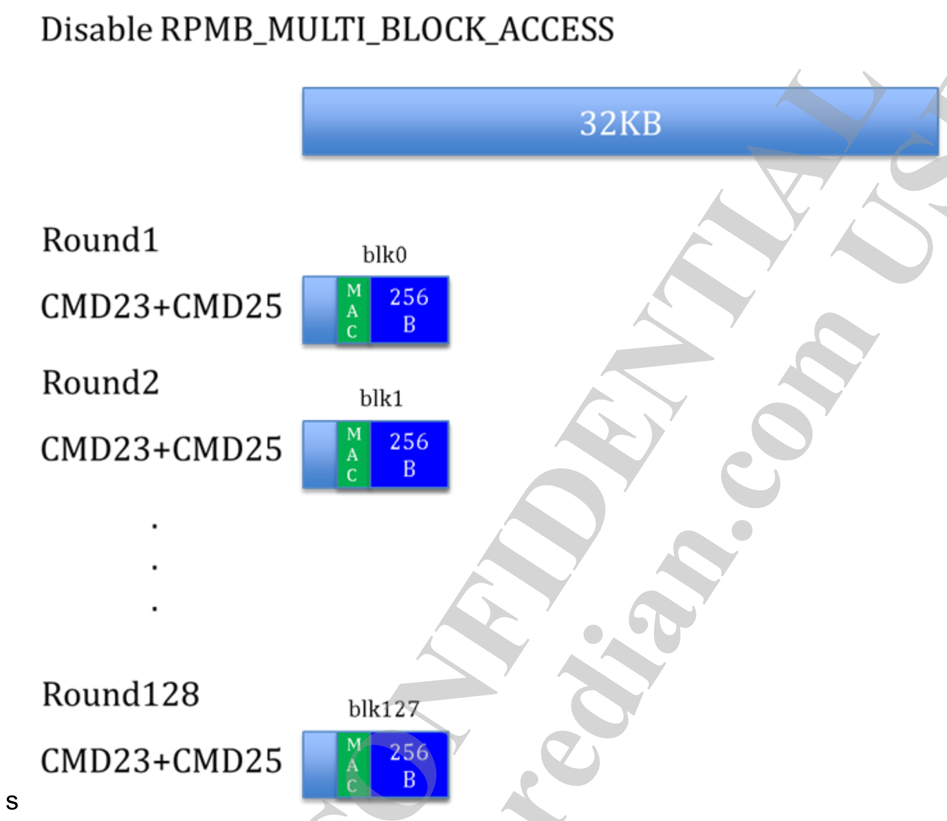


Figure 3-6. Example when disable RPMB_MULTI_BLOCK_ACCESS

3.1.3 TEE RPMB driver IPC

In this layer, we receive a data frame from Driver API. The objective of this layer is to define a command ID to handshake with kernel driver. The transition message structure named dciMessage_t is alloced by kernel driver when it called mc_malloc_wsm. Since IPC thread and Driver API used a different communication message structure, it has to copy data frame to its inside. When kernel driver returned, response data frame is also back. Finally, IPC has to copy response data frame to Driver API.

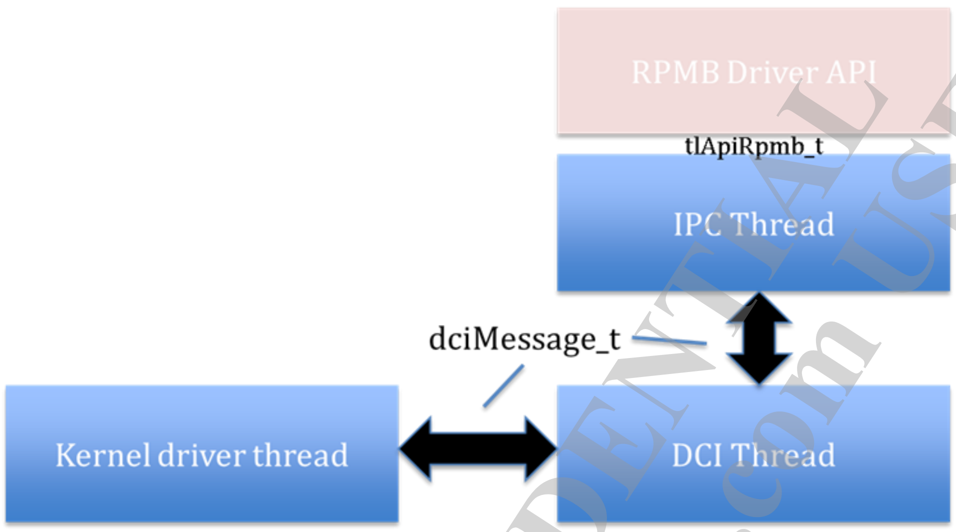


Figure 3-7. communication message

3.1.4 TEE RPMB driver DCI

There is nothing here. Only to handle notification between kernel driver and IPC thread.

3.2 Normal world RPMB Driver Introduction

The key words of kernel driver are character device and thread. Firstly, we registered RPMB driver as a character device driver so that we can re-use the functions with IOCTLs by users. Secondly, we create a thread which is waiting a notification from secure RPMB driver DCI thread. Once received a notification, the following coming is a request, and then we service it.
Figure 3-5 show you what we do in the driver initialization. Since driver initialization was called earlier than mcDaemon and services are not ready, that caused us open session fail. The first thread named “rpmb_open” used to retry open session until it success. If open success, create second thread named “rpmb_Dci” for listening DCI.

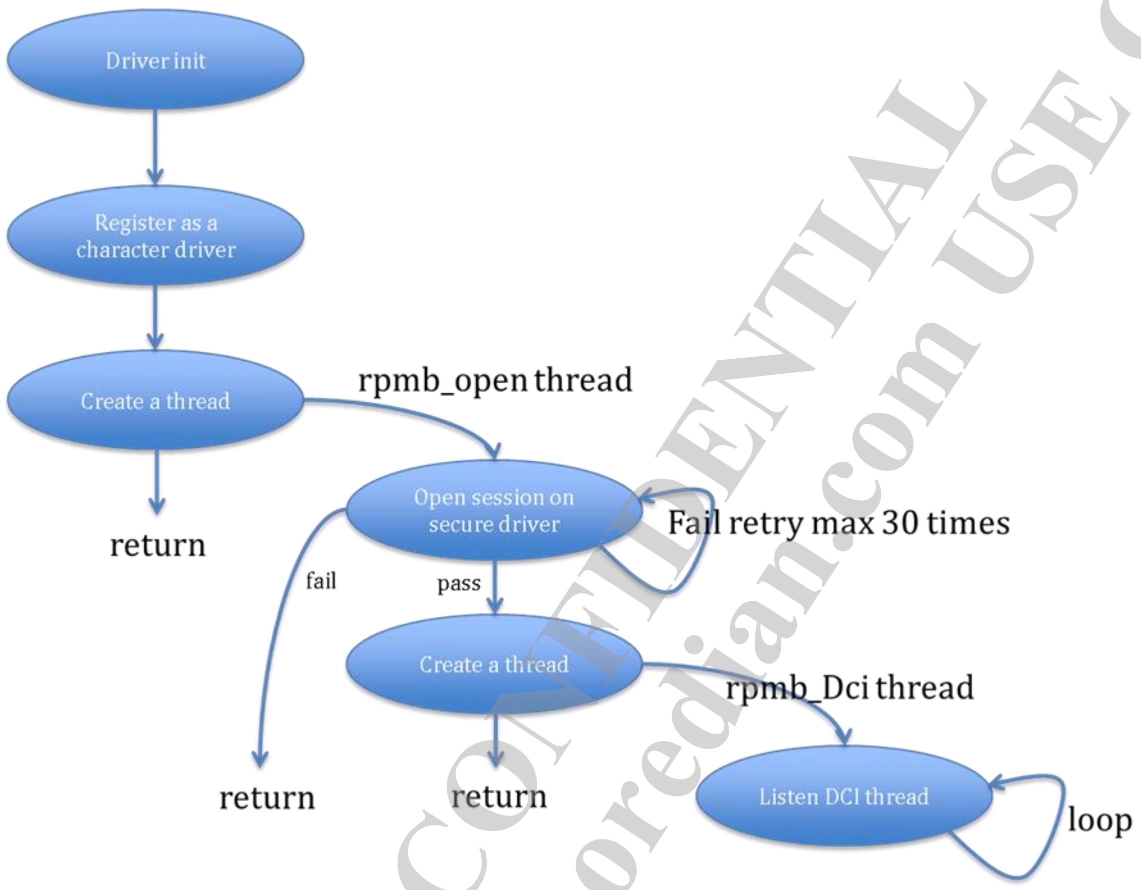


Figure 3-8. Flow of kernel driver init

4 INTERFACES

This chapter introduce interfaces exported by driver API that trusted agents can use. To use following APIs, you should include tlRpmbDriverApi.h.

4.1 tlApiRpmbOpenSession

```
_TLAPI_EXTERN_C uint32_t tlApiRpmbOpenSession(uint32_t uid)
```

Calling tlRpmbOpenSession to get a permission and it will return a valid ID. Any errors you will get an invalid ID 0xFFFFFFFF.

All TLs must open a session on the secure RPMB driver in the beginning. Driver supports max 5 users now, and it's defined by MAX_DR_SESSIONS in drSmgmt.h that means it is a customization.

4.2 tlApiRpmbCloseSession

```
_TLAPI_EXTERN_C uint32_t tlApiRpmbCloseSession()
```

Close session is paired with open session. Pay attention to close session if any returns.

4.3 tlApiRpmbReadData

```
_TLAPI_EXTERN_C tlApiResult_t tlApiRpmbReadData(
        uint32_t sid,
        uint8_t *buf,
        uint32_t bufSize,
        int *result)
```

Session id is used by driver to determine which logical partition (refer to chapter 6. Partition management) in RPMB will be read. RPMB driver will take care of the transfer size and copied to the buffer pointer. The output result is zero means no error. We will define the error code more detail. The read always begin from the first byte of the targeted logical partition.

4.4 __tlApiRpmbReadData

```
_TLAPI_EXTERN_C tlApiResult_t tlApiRpmbReadData(
        uint32_t sid,
        uint32_t offset,
        uint8_t *buf,
        IN  uint32_t bufSize,
        int *result)
```

Session id is used by driver to determine which logical partition in RPMB will be read. Offset is used by driver to determine the first byte in the targeted logical partition to be read. RPMB driver will take care of the transfer size and copied to the buffer pointer. The output result is zero means no error. We will define the error code more detail. With this API, TA can directly get targeted byte range in the targeted partition without prepare extra buffer space for byte preceding offset.

4.5 tlApiRpmbWriteData

```
_TLAPI_EXTERN_C tlApiResult_t tlApiRpmbWriteData (
        uint32_t sid,
        uint8_t *buf,
        uint32_t bufSize,
        int *result)
```

Similar to tlApiRpmbReadData, but this is for write.

4.6 __tlApiRpmbWriteData

```
_TLAPI_EXTERN_C tlApiResult_t tlApiRpmbWriteData (
        uint32_t sid,
        uint32_t offset,
        uint8_t *buf,
        uint32_t bufSize,
        int *result)
```

Similar to __tlApiRpmbReadData, but this is for write.

5 HOW TO USE

The TEE RPMB driver provided a simple interface that we talked about in chapter 4. Only you need care about is the session ID, offset (optional), input buffer, buffer size and the output result. Result zero means no problem.

Following show the sample codes how you need to do.

```
static tciReturnCode_t tlRpmbRead(tciMessage_t* message)
{
   tlApiCrSession_t  crSession;
   tlApiResult_t     tlRet;
   tciReturnCode_t   ret = RET_OK;
   int result;

   do
   {
      // Calling driver with IPC.
      // Call waits for the answer and blocks the trustlet.
      // The function is located inside drrpmb.lib
      // that is implemented and built within drrpmb

      crSession = tlApiRpmbOpenSession(0);
      if (crSession == 0xFFFFFFFF)
         return RET_ERROR;

      tlApiRpmbReadData(crSession, message->Buf, message->BufSize, &result);

      if (result) {
         tlDbgPrintLnf("[Trustlet Tlrpmb] tlApiReadData error. (%x)", result);
         ret = RET_ERROR;
      }

      tlApiRpmbCloseSession(crSession);

   }while(false);

   message->ResultData = result;

   return ret;
}
```

6 PARTITION MANAGEMENT

Like RPMB key was delivered to TEE driver from preloader, we deliver RPMB size too. According to RPMB size, TEE RPMB driver default separates each partition averagely. The default partition number was defined depend on MAX_DR_SESSIONS which used to limit user number in the TEE driver makefile.

We also provide customization to adjust each partition size for users.

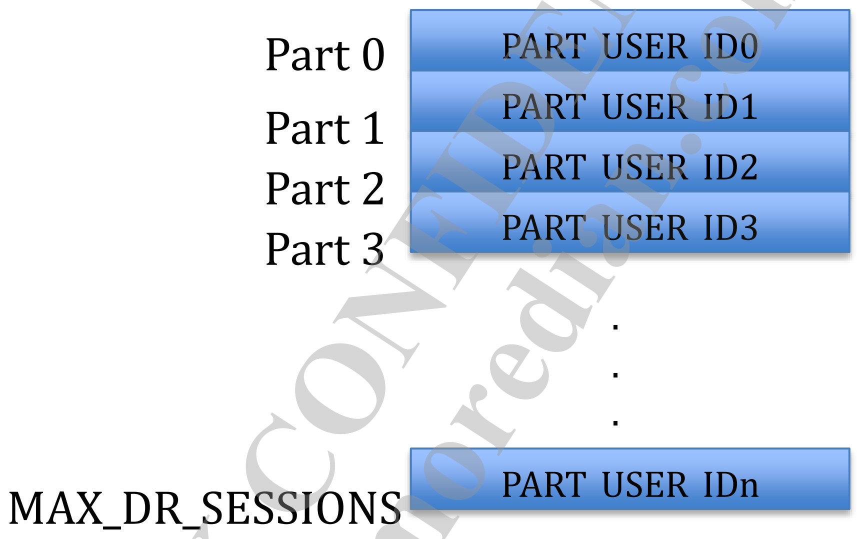


Figure 6-1. partition layout concept

7 ERROR HANDLING

7.1 IO error

Since hardware driver has retry error handling mechanism, it caused RPMB write return a fail. To resolve this problem, TL has to retry the whole process by calling tlApiRpmbWriteData again.

7.2 Open session error

TL calls driver API to open a session, it will return a session ID. When the return value is 0xFFFFFFFF, it means there is no session permitted or you give a wrong user ID. Most of time, user ID is unique, wrong user ID means you are duplicated with others.

7.3 Interface error

8 STAND-ALONE KERNEL RPMB DRIVER WITHOUT TEE

Make sure that enable CONFIG_CRYPTO_HMAC and CONFIG_CRYPTO_SHA256 can provide us HMAC-SHA256 service.
Without TEE, we don't guarantee the security. There is an easy way to re-use this driver to access RPMB, it is IOCTLs. There are three ioctl codes, RPMB_IOCTL_PROGRAM_KEY, RPMB_IOCTL_WRITE_DATA, RPMB_IOCTL_READ_DATA and an ioctl structure rpmb_ioc_param defined in emmc_rpmb.h. (It should be better to move to a public header.)

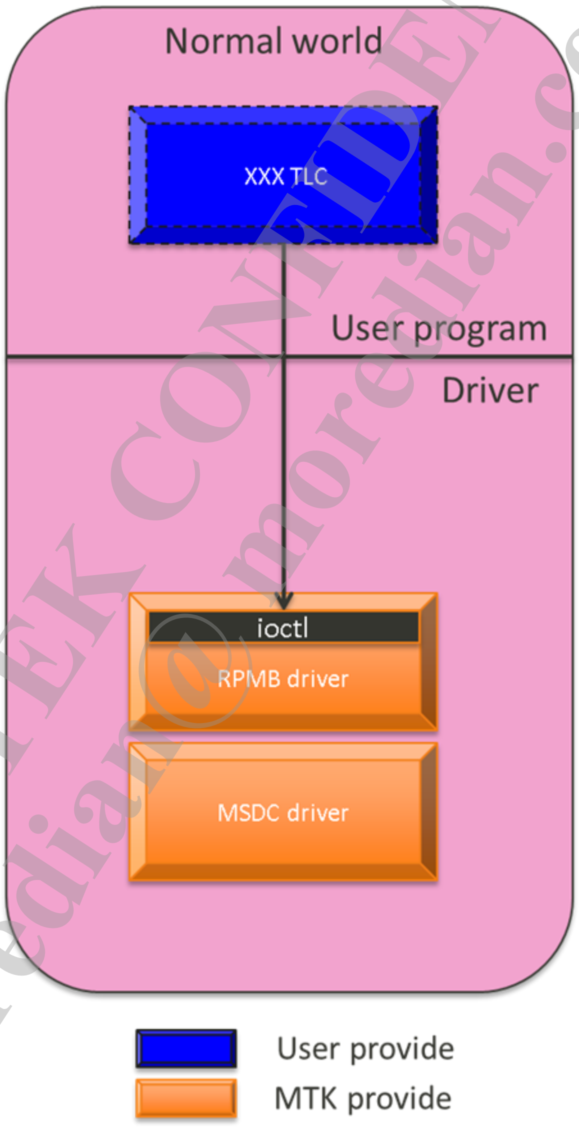


Figure 8-1. stand-alone driver architecture

8.1 Default Kernel Block IOCTL

Kernel also provided an interface to access RPMB. But this is just a simple interface that only performs CMD23, CMD25 and CMD18 procedure. All the RPMB data frame and MAC computation are user's responsibilities. The alternative way is composed data frame in the user space.

9 ATTACK SIMULATION

9.1 Cheating Attack

Due to RPMB key put in the TEE, hacker cannot create a new frame to access RPMB.

9.2 Denial-of-service Attack

We cannot prevent this attack. MSDC hardware relies on other modules such like clock and power. If you want, stop clock or cut power off can make MSDC IO fail. This imply to RPMB read/write function can be stopped.

9.3 Replay Attack

To consider read case, there is a nonce filed which is random number in the data frame used to prevent replay attack.
To consider write case, write counter also can be guaranteed by nonce. Write data result and write counter can prevent replay attack.

10 DRIVER PACKAGE

We are going to show you the driver package contents. There are two parts, one is common code and another is platform related. Here, following table show you summary of package files.

Table 10-1. Driver package

Binary	Source path	File
Preloader	vendor/mediatek/proprietary/bootable/bootloader/preloader/ platform/**[mtxxxx]**/src/drivers	mmc_rpmb.c mmc_*.c msdc_*.c
	vendor/mediatek/proprietary/bootable/bootloader/preloader/ platform/**[mtxxxx]**/src/drivers/inc	mmc_rpmb.h mmc_*.h msdc_*.h
	vendor/mediatek/proprietary/bootable/bootloader/preloader/ platform/**[mtxxxx]**/src/drivers	Makefile
	vendor/mediatek/proprietary/bootable/bootloader/preloader/ platform/**[mtxxxx]**/src/security/trustzone	tz_init.c tz_tbase.c
	vendor/mediatek/proprietary/bootable/bootloader/preloader/ platform/**[mtxxxx]**/src/security/trustzone/inc	tz_init.h tz_mem.h
	vendor/mediatek/proprietary/bootable/bootloader/preloader/ platform/**[mtxxxx]**/src	Seclib.a SecPlat.a
TEE driver	vendor/mediatek/proprietary/ /trustzone/common/hal/source/trustlets/rpmb/common	drv ta tlc
	vendor/mediatek/proprietary/ /trustzone/trustonic/source/trustlets/rpmb/common/drv	makefile.mk
	vendor/mediatek/proprietary/ /trustzone/trustonic/source/trustlets/rpmb/common/ta	makefile.mk
Kernel driver	kernel-4.9/drivers/char/rpmb	Makefile rpm-mtk.c
	kernel-3.18/drivers/mmc/host/mediatek/rpmb	drrpmb/public drrpmb_gp/p ublic

11 LIMITATION

1. TL stack and heap size at least 16KB. RPMB_MULTI_BLOCK_ACCESS is default enabled.

2. Currently, maximum transfer size is 8KB which defined by MAX_RPMB_REQUEST_SIZE in drrpmb_Api.h. If you have performance issue, try to enlarge this size, but remember to enlarge stack and heap size at the same time.

3. TAs have to provide error handling by themselves when RPMB driver API return an error which is not zero.

4. TL has to take responsibility of giving a single user id to prevent access in a same partition.

5. Partition number is default decided by MAX_DR_SESSIONS and each partition length is average. The start address and length of each partition could be customized on user's wish by editing rpmb_part_tbl. Currently the checking mechanism is not complete, so pay attention to your transfer length not cross over partition length.

12 UNIT TEST

1. Build a user program named tlcrpmb. Build command: mmm vendor/mediatek/proprietary/protect-bsp/trustzone/trustonic/trustlets/rpmb/Tlcrpmb/Locals/Code/; the output will be under system/bin.
2. adb push tlcrpmb /data
3. Execute program: tlcrpmb

Table 12-1. test case

Item	Command	Description
TEE read	./tlcrpmb read [length]	According to user ID to decide the start address to read.
TEE write	./tlcrpmb write [length] [value]	According to user ID to decide the start address to write and the write value.
IOCTL read	./tlcrpmb ioctl read [length] [addr]	Decide which address you prefer to read and the length through ioctl.
IOCTL write	./tlcrpmb ioctl write [length] [value] [addr]	Decide address, length and value you prefer to write through ioctl.
Multiple TAs access		Cooperate with DRM TA to use different user ID.
Multiple block access		Enable option RPMB_MULTI_BLOCK_ACCESS.
Single block access		Disable option RPMB_MULTI_BLOCK_ACCESS.
Large size transfer		Since internal buffer just set to 8KB, try larger size more than 8KB.
Small size transfer		Try small size transfer within a block.
Concurrent stress test		Playing mp3 or recording video and perform RPMB read/write at the same time.
CRC error test		Hardware driver give a retry when crc happened, it might cause RPMB access fail.

13 DEMO

14 GP API

Now we support Global Platform APIs. Include header file tlRpmbDriverApi.h.

14.1 TEE_RpmbOpenSession

```
_TLAPI_EXTERN_C uint32_t TEE_RpmbOpenSession(
            IN uint32_t uid)
```

Same usage as legacy api. Input a user id, it will return a session id to you. You can take this session id to do following things.

14.2 TEE_RpmbCloseSession

```
_TLAPI_EXTERN_C TEE_Result TEE_RpmbCloseSession(
            IN uint32_t sid)
```

Close this session by session id.

14.3 TEE_RpmbReadDatabyOffset

```
_TLAPI_EXTERN_C TEE_Result TEE_RpmbReadDatabyOffset(
            IN uint32_t sid,
            IN uint32_t offset,
            OUT uint8_t *buf,
            INT uint32_t bufSize,
            OUT int *result)
```

This is new added API. Read by offset can improve performance. You can decide where begin to read.

14.4 TEE_RpmbWriteDatabyOffset

```
_TLAPI_EXTERN_C TEE_Result TEE_RpmbWriteDatabyOffset(
            IN uint32_t sid,
            IN uint32_t offset,
            IN uint8_t *buf,
            IN uint32_t bufSize,
```

IN int *result)

This is new added API same as read by offset. You can decide where begin to write.

14.5 TEE_RpmbReadData

_TLAPI_EXTERN_C TEE_Result TEE_RpmbReadData(

IN uint32_t sid,

OUT uint8_t *buf,

IN uint32_t bufSize,

IN int *result)

Default read from your partition head.

14.6 TEE_RpmbWriteData

_TLAPI_EXTERN_C TEE_Result TEE_RpmbWriteData(

IN uint32_t sid,

IN uint8_t *buf,

IN uint32_t bufSize,

IN int *result)

Default write from your partition head. User must read all their data to update a part of data then write all back.

15 GP API Test Case

Table 15-1. test case

Item	Command	Description
TEE read	./tlcrpmb_gp read [user id] [offset] [length]	User id: make you choose which partition to read. Offset: In you partition, where begin to read. Length: Read length.
TEE write	./tlcrpmb_gp write [user id] [offset] [length] [value]	Same as above, except you could decide write value.
Multiple TAs access	Open two terminal and run tlcrpmb_gp read or write with different user id.	Expect they are not conflicted by each other.
Simulate same TA access	Open two terminal and run tlcrpmb with same user id at same time. Ex. ./tlcrpmb_gp read 0 0 256 ./tlcrpmb_gp read 0 0 512 [Expected result] Second execution with existed user id will return fail.	New comer user cannot open an existed session to access their partition.
Cross partition access	Ex. If a partition size is 32K, ./tlcrpmb_gp read 0 0 32768 or ./tlcrpmb_gp read 0 32766 2 [Expected result] Return error.	Anyone cannot access out of their partition.
Every size transfer	Full block test: ./tlcrpmb_gp write 0 0 256 1 ./tlcrpmb_gp read 0 0 256 Compare if read data is 1. ./tlcrpmb_gp write 0 0 512 ./tlcrpmb_gp read 0 0 512 Compare if read data is 2. ./tlcrpmb_gp write 0 0 4096 3 ./tlcrpmb_gp read 0 0 4096 Compare if read data is 3. ./tlcrpmb_gp write 0 0 32768 4	Test with different offset, different size(full block or partial block)

Item	Command	Description
	./tlcrpmb_gp read 0 0 32768 Compare if read data is 4. ./tlcrpmb_gp write 0 256 256 1 ./tlcrpmb_gp read 0 256 256 Compare if read data is 1. ./tlcrpmb_gp read 0 0 512 Compare if read data 0~256 is 4. Compare if read data 256~512 is 1 ./tlcrpmb_gp write 0 4096 4096 2 ./tlcrpmb_gp read 0 0 8192 Compare if read data 4096~8192 is 2. Partial block test: ./tlcrpmb_gp write 0 0 32 5 ./tlcrpmb_gp read 0 0 256 Compare if read data 0~32 is 5. Compare if read data 32~256 is 4. ./tlcrpmb_gp write 0 32 260 6 ./tlcrpmb_gp read 0 0 512 Compare if read data 0~32 is 5. Compare if read data 32~260 is 6. Compare if read data 260~512 is 1. Using same concepts as above to test different large size.	
Concurrent stress test		Playing mp3 or recording video and perform RPMB read/write at the same time.

16 How to Enable GP support (Not Need for MT6763 Android N)

Step 1. Find vendor\mediatek\proprietary\trustzone\trustonic\source\build\platform\[project]\tee_config.mk

Step 2. Edit tee_config.mk.

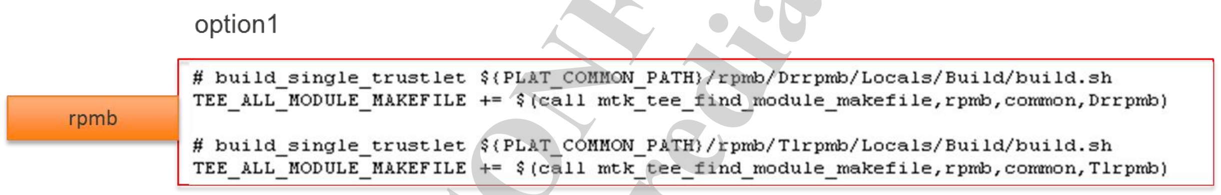
There are three groups named 'rpmb', 'rpmb_legacy' and 'rpmb_gp' here.

Option 1: rpmb

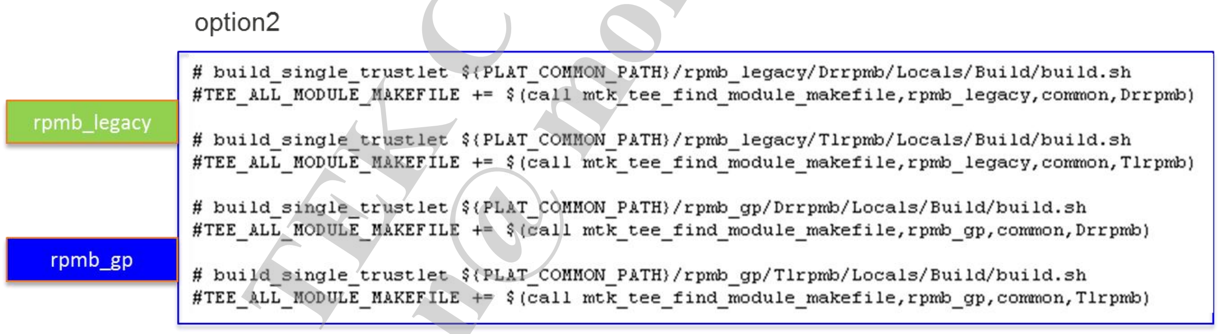
Option 2: rpmb_legacy and rpmb_gp.

You could choose one of the options to enable. For example, enable rpmb but disable rpmb_legacy and rpmb_gp or enable rpmb_legacy and rpmb_gp but disable rpmb.

You cannot choose option1 and one of option2 together.



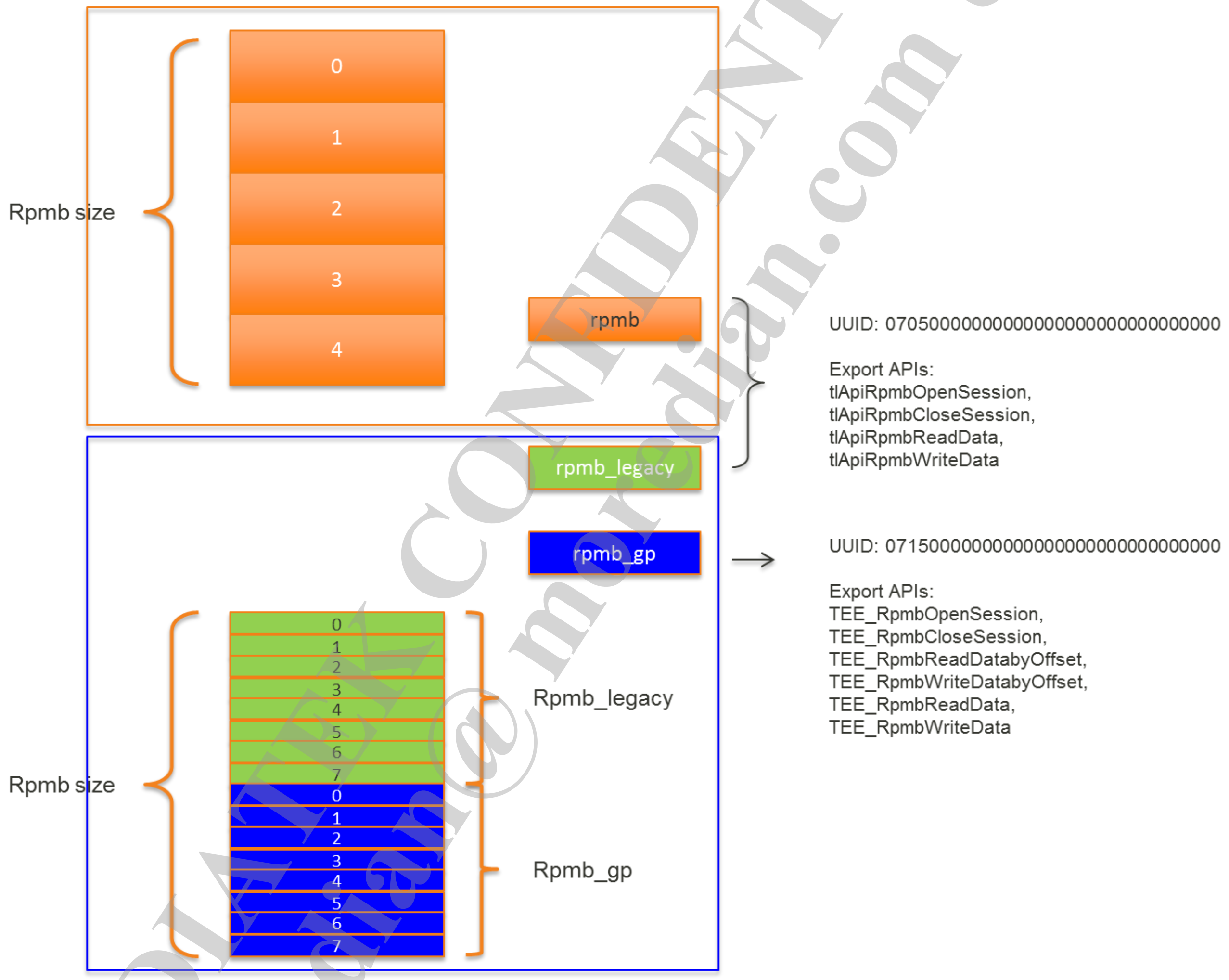
```
# build_single_trustlet ${PLAT_COMMON_PATH}/rpmb/Drrpmb/Locals/Build/build.sh
TEE_ALL_MODULE_MAKEFILE += $(call mtk_tee_find_module_makefile,rpmb,common,Drrpmb)

# build_single_trustlet ${PLAT_COMMON_PATH}/rpmb/Tlrpmb/Locals/Build/build.sh
TEE_ALL_MODULE_MAKEFILE += $(call mtk_tee_find_module_makefile,rpmb,common,Tlrpmb)
```



```
# build_single_trustlet ${PLAT_COMMON_PATH}/rpmb_legacy/Drrpmb/Locals/Build/build.sh
#TEE_ALL_MODULE_MAKEFILE += $(call mtk_tee_find_module_makefile,rpmb_legacy,common,Drrpmb)

# build_single_trustlet ${PLAT_COMMON_PATH}/rpmb_legacy/Tlrpmb/Locals/Build/build.sh
#TEE_ALL_MODULE_MAKEFILE += $(call mtk_tee_find_module_makefile,rpmb_legacy,common,Tlrpmb)

# build_single_trustlet ${PLAT_COMMON_PATH}/rpmb_gp/Drrpmb/Locals/Build/build.sh
#TEE_ALL_MODULE_MAKEFILE += $(call mtk_tee_find_module_makefile,rpmb_gp,common,Drrpmb)

# build_single_trustlet ${PLAT_COMMON_PATH}/rpmb_gp/Tlrpmb/Locals/Build/build.sh
#TEE_ALL_MODULE_MAKEFILE += $(call mtk_tee_find_module_makefile,rpmb_gp,common,Tlrpmb)
```

17 GP API Design Concept (Not Need for MT6763 Android N)

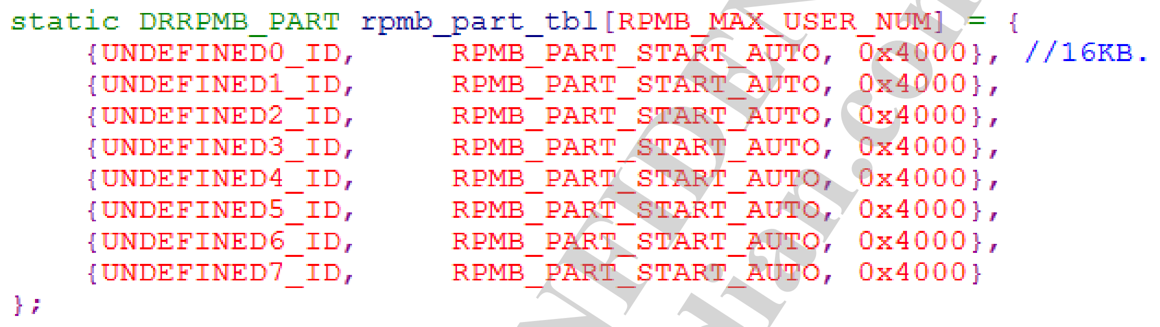
To backward compatible, create a new rpmb_legacy totally same as rpmb. Same UUID, same export APIs to make kernel driver and TA users work transparently. The only different is partition layout changed from original rpmb.



18 Customization

Each rpmb_legacy and rpmb_gp extend max user number to 8. Each user partition was pre-defined to 16KB. Users can resize partition size according to their requirements. The start address was calculated automatically so that user only need to care is the user partition size.

You can find partition table in drrpmbOps.c.



```
static DRRPMB_PART rpmb_part_tbl[RPMB_MAX_USER_NUM] = {
     {UNDEFINED0_ID,      RPMB_PART_START_AUTO, 0x4000}, //16KB.
     {UNDEFINED1_ID,      RPMB_PART_START_AUTO, 0x4000},
     {UNDEFINED2_ID,      RPMB_PART_START_AUTO, 0x4000},
     {UNDEFINED3_ID,      RPMB_PART_START_AUTO, 0x4000},
     {UNDEFINED4_ID,      RPMB_PART_START_AUTO, 0x4000},
     {UNDEFINED5_ID,      RPMB_PART_START_AUTO, 0x4000},
     {UNDEFINED6_ID,      RPMB_PART_START_AUTO, 0x4000},
     {UNDEFINED7_ID,      RPMB_PART_START_AUTO, 0x4000}
};
```

There is an option named ‘RPMB_GP_PART_SIZE_PERCENT’ defined as 100. It means the percentage of full rpmb size used to GP partition. For example, if RPMB size is 1MB, set this value 50 means 512KB for legacy partition and 512KB for GP partition. Once again, please pay attention to customize each user partition size not to oversize the boundary.

19 Programming RPMB key in Linux Kernel

For security consideration, the default design of programming RPMB key is in preloader. However, some phone vendor want to program RPMB key in Linux kernel (because of certain concern). Therefore, mechanism for programming RPMB key in Linux kernel is also implemented. Since programming key in Linux Kernel have the risk of revealing key in non-secure environment. This mechanism is default disabled. This chapter describe how to enable this mechanism. **Note: anyone who enable/use this mechanism shall accept the risk of revealing key. Mediatek won't take the responsibility for loss incurred by the key revealing.**

19.1 Enable RPMB key programming in Linux Kernel

To enable RPMB key programming in Linux Kernel, some modification shall be made:

1. Preloader

   File: vendor/mediatek/proprietary/bootable/bootloader/preloader/custom/$project/cust_bldr.mak

   Remove line: **CFG_RPMB_KEY_PROGRAMED_IN_KERNEL := 0**

   Enabled/Add line: **CFG_RPMB_KEY_PROGRAMED_IN_KERNEL := 1**

2. Linux Kernel

   File: kerne-4.9/drivers/char/rpmb/Makefile

   Enable/Add line: **ccflags-y += -DCFG_RPMB_KEY_PROGRAMED_IN_KERNEL**

3. Secure world

   File: vendor/mediatek/proprietary/trustzone/common/hal/source/trustlets/rpmb/common/drv/Android.mk

   Remove line: **CFG_RPMB_KEY_PROGRAMED_IN_KERNEL := 0**

   Enable/Add line: **CFG_RPMB_KEY_PROGRAMED_IN_KERNEL := 1**

   Note: There are 2 instance of CFG_RPMB_KEY_PROGRAMED_IN_KERNEL in this Android.mk, values of both instance shall be changed from 0 to 1

   File: vendor/mediatek/proprietary/trustzone/common/hal/source/trustlets/rpmb/common/ta/Android.mk

   Remove line: **CFG_RPMB_KEY_PROGRAMED_IN_KERNEL := 0**

   Enable/Add line: **CFG_RPMB_KEY_PROGRAMED_IN_KERNEL := 1**

File:

vendor/mediatek/proprietary/trustzone/common/hal/source/trustlets/rpmb/common/tlc/Android.mk

Enable/Add line: **LOCAL_CFLAGS += -DCFG_RPMB_KEY_PROGRAMED_IN_KERNEL**

Note: There are 2 lines containing

"#LOCAL_CFLAGS += -DCFG_RPMB_KEY_PROGRAMED_IN_KERNEL" in this Android.mk, please remove the leading # sign of these 2 lines

File:

vendor/mediatek/proprietary/trustzone/trustonic/source/trustlets/rpmb/common/ta/makefile.mk

Enable/Add line:

ARMCC_COMPILATION_FLAGS += -DCFG_RPMB_KEY_PROGRAMED_IN_KERNEL

Note: This file is not existed and no need this step for Android P.

19.2 Use per device RPMB key

Default rpmb key is common (RPMB_PER_DEVICE_KEY=no) for all devices, to use per device RPMB key, change below should be made:
vendor/mediatek/proprietary/bootable/bootloader/preloader/custom/[$project]/[$project].mk

Enabled/Add line: **RPMB_PER_DEVICE_KEY=yes**

Note: commn key location
vendor/mediatek/proprietary/bootable/bootloader/preloader/custom/[$project]/inc/rpmb_cust_key.h

19.3 UT for RPMB key programming in Linux Kernel

tlcrpmb_gp can be used to test RPMB key programming in Linux Kernel. Note that, tlcrpmb_gp is for testing/demonstration purpose during SW development. To perform key programming in Linux Kernel during mass production, phone vendor shall develop their RPMB key programming application/flow by referenceing tlcrpmb_gp.

1. Enable TA

   Modify

   vendor/mediatek/proprietary/trustzone/trustonic/source/build/platform/**[mtXXXX]**/tee_config.mk

   #TEE_ALL_MODULE_MAKEFILE += $(call mtk_tee_find_module_makefile,rpmb,common,ta)

   Please remove the # sign

   CAUTION: Enable TA may reveal the backdoor for others to trigger write from normal world.

2. Build tlcrpmb_gp

   Use the following build command:

   mmm vendor/mediatek/proprietary/trustzone/common/hal/source/trustlets/rpmb/common/tlc

   The built executable will be:

   out/target product/[$project]/obj/EXECUTABLES/tlcrpmb_gp_intermediates/tlcrpmb_gp

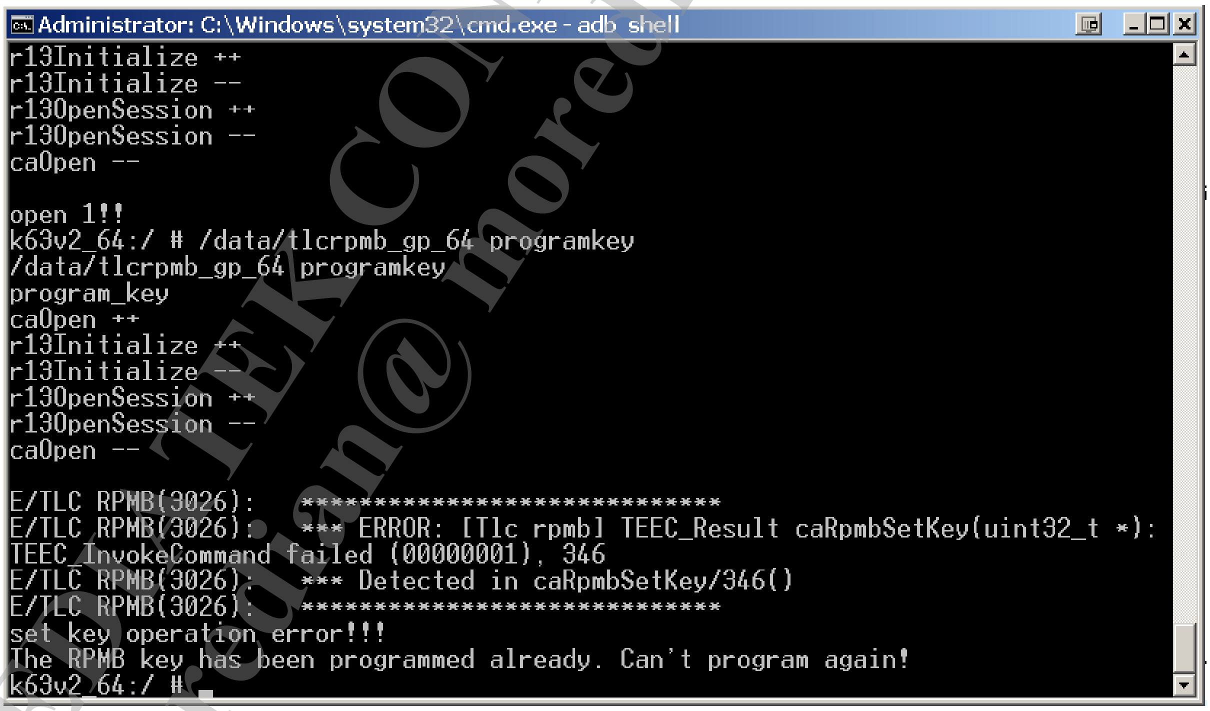
3. Run tlcrpmb for RPMB key programming UT

   Push the built tlrpmb_gp into phone, then issue the following command in adb shell:

   tlrpmb_gp programkey

   If key programming is programmed successfully in this execution, there shall be no error message.

   If key programming failed in this execution, there shall be error message. For example, if key had been programmed before, there will be error message as below:


Administrator: C:\Windows\system32\cmd.exe - adb shell

```
r13Initialize ++
r13Initialize --
r13OpenSession ++
r13OpenSession --
caOpen --

open 1!!
k63v2_64:/ # /data/tlcrpmb_gp_64 programkey
/data/tlcrpmb_gp_64 programkey
program_key
caOpen ++
r13Initialize ++
r13Initialize --
r13OpenSession ++
r13OpenSession --
caOpen --

E/TLC RPMB(3026):   ******************************
E/TLC RPMB(3026):   *** ERROR: [Tlc rpmb] TEEC_Result caRpmbSetKey(uint32_t *):
TEEC_InvokeCommand failed (00000001), 346
E/TLC RPMB(3026):   *** Detected in caRpmbSetKey/346()
E/TLC RPMB(3026):   ******************************
set key operation error!!!
The RPMB key has been programmed already. Can't program again!
k63v2_64:/ #
```