A way to implement virtualized RPMB support in OP-TEE

Volodymyr Babchuk

December, 2019

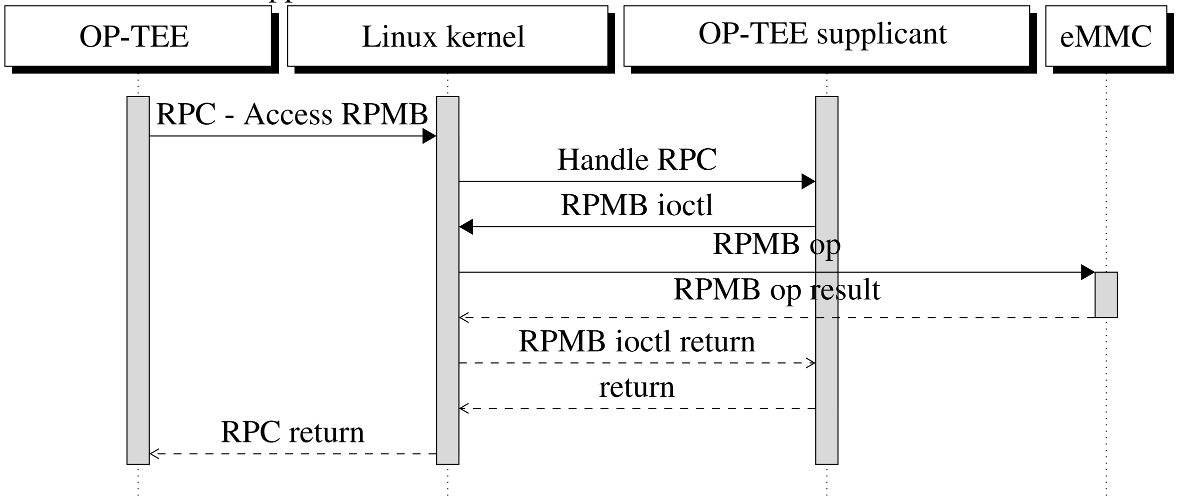
Abstract

Replay Protected Memory Block (RPMB) is the feature found in all eMMC and in some MMC cards. It guarantees that information written there can't be replaced with the old version. OP-TEE supports this to provide protection against replay attack for the Secure Storage services. But it does not support this feature in the virtualized systems as RPMB can't be easily shared between virtual machines. There are some ways to implement shared RPMB support, which are described in this paper.

Contents

1 RPMB without virtualization

OP-TEE does not have direct access to the RPMB device because it is the part of (e)MMC card and this card is used mostly by REE. Fortunately RPMB specification employs HMAC to ensure that only trusted code can read and write RPMB partition. So, there it is perfectly fine communicate with RPMB over Normal World. There is how it happens:



Linux kernel provide ioctls to communicate with RPMB partition on eMMC. OP-TEE supplicant receives RPMB requests RPC from OP-TEE and uses mentioned ioctls to access the RPMB partition.

It should be noted that during initialization OP-TEE reads the size of RPMB partition and write counter. Then, in runtime it maintains the write counter and returns an error if expected write counter does not correspond to one returned by RPMB. As OP-TEE is the only user of RPMB this is perfectly fine.

2 RPMB with virtualization

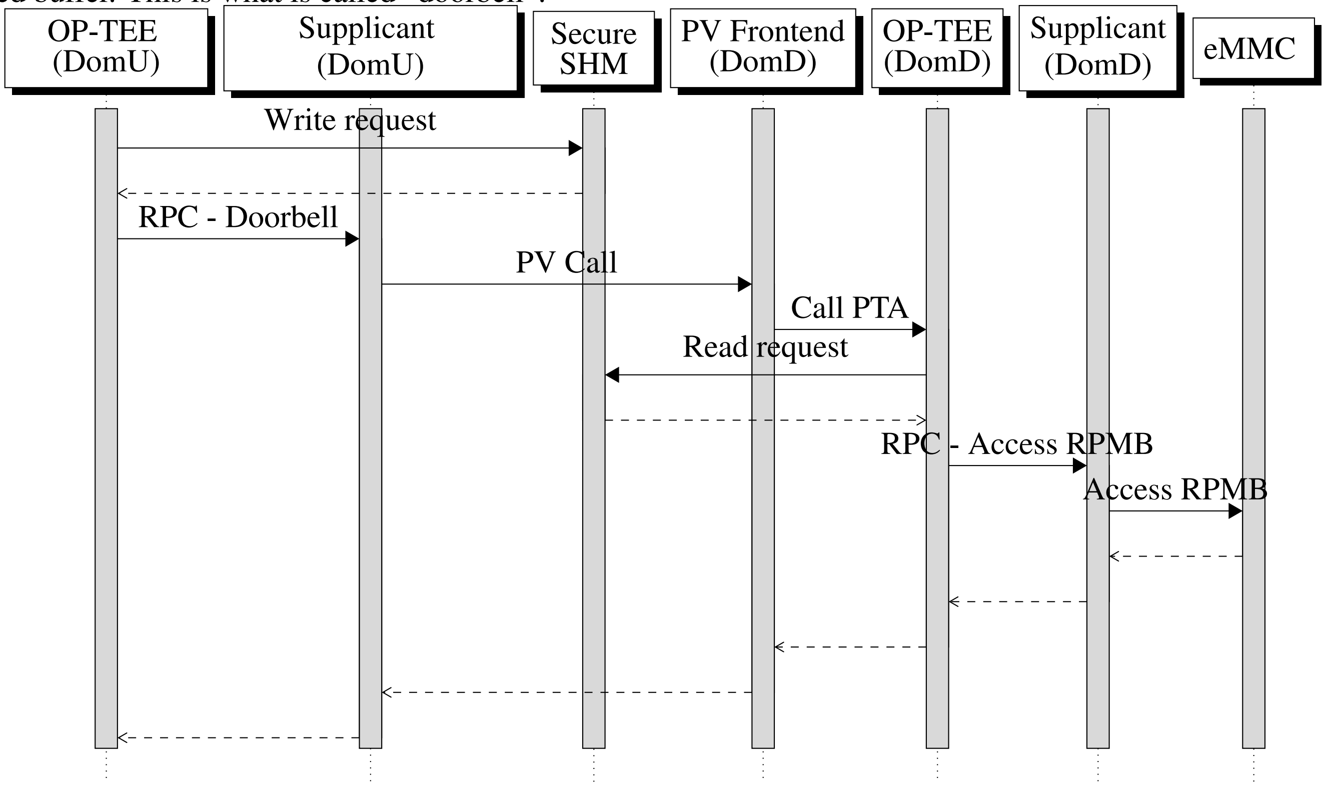
2.1 Sharing a single RPMB partition

Now suppose we want to share the single RPMB partition between the multiple guests. The simplest solution is to provide each guest with its own RPMB-capable device. But in the most cases platform will have only one eMMC device, so all guests should share it in some way. Also, this device will physically available to only one guest, let's call it "Driver Domain" or "DomD". Writes to RPMB should be atomic, but nature or RPMB driver in OP-TEE requires multiple writes if RPMB is not capable of writing big buffer at once. This leads to series of writes, with increasing write counter for every write request. That will lead to de-synchronization of write counter value stored in different TEE instances. Also, that could lead to race condition, when two or mode TEE instances submit batch of write/read requests in the same time.

2.2 Communication between OP-TEE instances

As other domains can't work with the eMMC directly we will require PV protocol to access foreign RPMB partition. The problem is how to divide RPMB partition between the multiple guests. We want each OP-TEE instance to operate with the physical RPMB partition, but on other hand we want to ensure maximum isolation between TEE instances. Clearly, we need one designated TEE instance that is capable to accessing RPMB. This is the instance that is associated with DomD. Also we need some communication method between TEE instances. As OP-TEE is scheduled by Normal World, we can't make direct communication between TEE instances, because in this way one guest can lock up some another guest.

So we need a secure way to exchange messages between TEE instances, but which is scheduled by NW. This involves inter-guest communication in the NW using mechanisms provided by the hypervisor. We are not required to send actual data, only signal to another TEE instance that there is a some request in some shared buffer. This is what is called "doorbell".



This diagram does not include Linux kernels for simplicity.

As you can see, client TEE instance writes the request to a shared buffer inside the secure memory. Then it issues RPC request back to own supplicant. Supplicant uses hypervisor-provided way to signal frontend server in the DomD. In case of Xen, inter-domain events can be used. This is a common mechanism to implement PV drivers in the Xen. Virtio also provides similar methods of signaling. PV server in DomD opens session to pTA in OP-TEE that belongs to DomD. This pTA accesses the previously written shared memory and reads the request from DomU. Then it can handle request - i.e. issue RPMB command. This provides the secure way to transmit requests from one TEE instance to another without adding inter-domain locking mechanisms. Obviously the same mechanism can be used not only for RPMB requests but for any inter-domain communication.

2.3 Partitioning RPMB Area

The provided mechanism answers the question "how to access RPMB from multiple domains". But there is another question: "how to share space on RPMB partition?". Now OP-TEE have a static number of domains configured at a compilation time (`CFG_VIRT_GUEST_COUNT` configuration option). So, we can divide RPMB partition into `CFG_VIRT_GUEST_COUNT` (equal) parts and grant every guest access to one of these. But how we can make sure that guest A will have access to partition for guest A, not for guest B? We can't rely on VM ID received by hypervisor at least because VMs can be restarted and each time it will get a new ID. Also, IDs will vary depending on VM creation order. Clearly we need persistent Virtual Machine ID (GUID will be fine). But I can't see a way how TEE instance can be sure that it communicates with the same VM every boot. As hypervisor could swap GUIDs between "good" and "rouge" VMs. Actually, the same thing possible with multiple CAs, accessing the same TA, as there is no mechanism to authenticate CA. This should be keep in mind while developing Trusted Applications.

So, as I'm seeing it, RPMB partition should have partition table somewhere. The very beginning of the partition should be fine.

This table should hold at least the following values:

- Magic value (might serve as the version number also)
- Total count of the partitions
- Table of partitions, each one having the next entries
  - GUID of the VM
  - Offset of the partition in RPMB write blocks
  - Size of the partition in RPMB write blocks

3 Extending TEE virtualization API

With features above, we need to extend virtualization-related APIs. The `OPTEE_SMC_VM_CREATED` call should be extended with the following information:

- GUID of the virtual machine
- Flag to indicate that this machine have access to real RPMB partition

In the future number of flags can be extended to denote ability to access hardware accelerators for example.

Also we need pTA for RPMB partition table management. So hypervisor (or trusted domain) can configure the table, e.g. - to assign GUIDs to table entries. It is debatable if it also should be able to wipe out partitions to re-assign GUIDs later.

Another pTA along with some shared memory mechanism is needed to enable inter-TEE instances communication as described in subsection 2.2.

So, only one TEE instance should enable this two pTAs.

4 Virtual RPMB partition

There is completely another approach to RPMB available. Next version of virtio interface specification provides virtio-rpmb interface which can be used to "forward" RPMB requests to another VM. The same mechanism can be used to enable software-emulated RPMB partitions inside of some "trusted" domain. This approach requires absolutely no changes in the OP-TEE code. We need to write software emulator for RPMB like the one used in OP-TEE supplicant, but with virtio-rpmb interface. Then every TEE instance will see own virtual RPMB device and work with it as usual, providing that kernel will have virtio-rpmb frontend code, which will be implemented anyways, as part of the virtio specification.

Actually, I believe that it is possible to use hardware RPMB to ensure replay protection in the emulated RPMB devices. For example, we can store hashes of virtual RPMB data in the hardware-backed RPMB devices.

5 Multiple hardware RPMB devices

If platform have more than one RPM partition (for example NVMe devices can support up to 8 independent RPMB partitions), then it is possible to assign each RPMB partition to a different VM. There are two ways possible.

5.1 Hardware partitioning

Some platforms allows to "give" access to a certain device to a certain VM. For example, if platform have 3 independent MMC controllers, every controller can be assigned to own VM and it will have exclusive access to that MMC. In this case no changes needed neither in OP-TEE, nor in linux driver. This is a matter of hypervisor configuration.

5.2 Virtio-rpmb assisted partitioning

It is possible that RPMB devices can't be exclusively assigned to VM. For example, all RPMBs are belonging to one NVMe drive. Or platform configuration have tightly bound MMC controllers, so there is no chance to assign them to a different VMs.

In this case virtio-rpmb protocol can be used. We can configure hypervisor to provide each VM with own RPMB physical partition via virtio-rpmb. This setup is similar to one described in section 4, but with real HW instead if emulation.

6 Comparison of proposed approaches

	Sharing using OP-TEE	Virtual RPMB	Hardware partitioning	Virtio-assisted partitioning
Support for only one hardware RPMB device	✓	✓	✗	✗
Support for none of hardware RPMBs	✗	✓	✗	✗
Support for multiple hardware RPMBs	✓(although design will be even more complex)	✓	✓	✓
Requires changes to OP-TEE	✓	Only to support HW-backed virtual RPMB	✗	✗
Requires changes hyper-visor/guests	✓	Being developed and up-streamed anyways	✗	Being developed and up-streamed anyways
Provides full hardware security	✓	Possibly, with reworking RPMB driver in OP-TEE	✓	✓
Can be used as generic solution	✓	✓	✗(because not so many platforms have multiple RPMBs)	✗(because not so many platforms have multiple RPMBs)