



# Protected DMAbufs and its dynamic memory assignment woes

Sumit Garg

Senior Staff Engineer, QUALCOMM India Private Limited

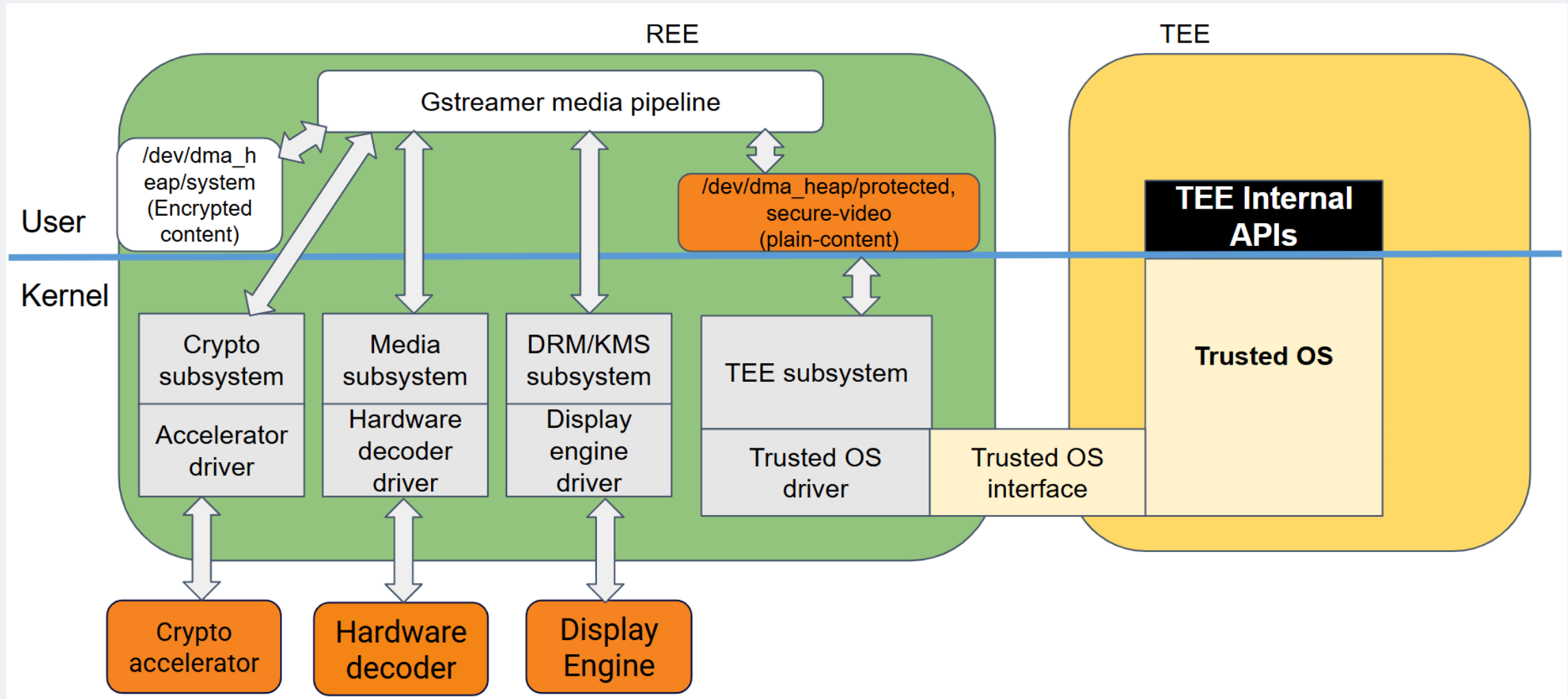
Snapdragon and Qualcomm branded products are products of Qualcomm Technologies, Inc. and/or its subsidiaries.



# Who am I?

- A member of the Qualcomm open-source team
- Have a keen interest in open-source boot & security
  - Linux kernel: TEE subsystem reviewer, author for TEE bus driver framework
  - edk2, TF-A and OP-TEE: Qualcomm platform maintainer
  - TF-A and OP-TEE: Firmware encryption framework maintainer
  - OP-TEE: Author for Rust no-std Trusted Applications support
- Other areas of interest
  - U-Boot: OF\_UPSTREAM devicetree maintainer
  - ftrace for OP-TEE
  - meta-arm-toolchain

# Protected DMAbufs use-case - Secure media pipeline



## Protected DMAbufs use-case

Secure DMA heaps have been discussed upstream for many years but haven't found it's way into the mainline kernel, the major reasons being:

- A missing end-to-end open-source use-case
- The difficulty to encode vendor specific metadata leading to vendor specific secure heaps being exposed to user-space

The Digital Rights Management (DRM) secure media pipeline use-case is the major driver here but with binding that the complete user-space and TEE stack not being fully open.

Kernel TEE subsystem came to the rescue here:

- Abstract out the vendor specific metadata to the backend TEE implementation
- Provide a common infrastructure to enable wider use-cases like protected DMAbufs based key/crypto operations etc.

## Protected DMAbufs: Static memory carveouts

Static memory carveouts for protected DMAbufs are based on reserved memory regions which are marked as “no-map” in devicetree.

The kernel trusted OS (OP-TEE) driver creates a protected memory pool based on the carveout configuration probed from OP-TEE.

Shortcomings:

- Lack of efficient memory reuse especially when buffer allocations reaches 100s of MBs
- Lack of support for scatter gather buffer allocations

## Protected DMAbufs: Dynamic memory assignment

The dynamic assignment of protected DMAbufs are based on regular DMAbufs allocated from the kernel CMA or system heap region.

The kernel trusted OS (OP-TEE) driver lends the regular DMAbuf region to the devices requiring exclusive access as per the use-case. Firmware Framework – A (FF-A) [memory management protocol](#) specifies how memory lending framework works.

Shortcomings:

- Missing framework in the kernel to enable exclusive device access once the memory lend happens

# Protected DMAbufs: Dynamic memory assignment - woes

drivers/tee/optee/protmem.c +35

```
static int init_dyn_protmem(struct optee_protmem_dyn_pool *rp)
{
    int rc;

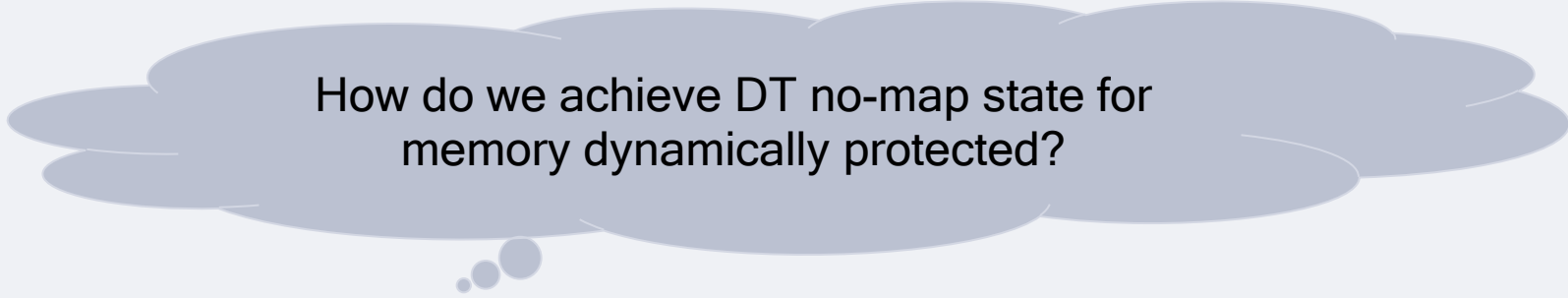
    rp->protmem = tee_shm_alloc_dma_mem(rp->optee->ctx, rp->page_count);
    if (IS_ERR(rp->protmem)) {
        rc = PTR_ERR(rp->protmem);
        goto err_null_protmem;
    }

    /*
     * TODO unmap the memory range since the physical memory will
     * become inaccessible after the lend_protmem() call.
     *
     * If the platform supports a hypervisor at EL2, it will unmap the
     * intermediate physical memory for us and stop cache pre-fetch of
     * the memory.
     */
    rc = rp->optee->ops->lend_protmem(rp->optee, rp->protmem,
                                     rp->mem_attrs,
                                     rp->mem_attr_count, rp->use_case);

    if (rc)
        goto err_put_shm;
    rp->protmem->flags |= TEE_SHM_DYNAMIC;
    ...
}
```

```
tee_shm_alloc_dma_mem()
-> dma_alloc_pages()
```

# Protected DMAbufs: Dynamic memory assignment – **solution?**



How do we achieve DT no-map state for memory dynamically protected?

- Heterogeneous Memory Management (HMM) subsystem can enable this use-case
  - [Exclusive access memory](#): enable support for CMA and system heap memory regions?
- Pages un-map call chain of interest

```
memunmap_pages()  
-> pageunmap_range()  
-> arch_remove_memory()
```
- Open to ideas?



# Thank you

Nothing in these materials is an offer to sell any of the components or devices referenced herein.

© Qualcomm Technologies, Inc. and/or its affiliated companies. All Rights Reserved.

Qualcomm and Snapdragon are trademarks or registered trademarks of Qualcomm Incorporated.  
Other products and brand names may be trademarks or registered trademarks of their respective owners.

References in this presentation to “Qualcomm” may mean Qualcomm Incorporated, Qualcomm Technologies, Inc., and/or other subsidiaries or business units within the Qualcomm corporate structure, as applicable. Qualcomm Incorporated includes our licensing business, QTL, and the vast majority of our patent portfolio. Qualcomm Technologies, Inc., a subsidiary of Qualcomm Incorporated, operates, along with its subsidiaries, substantially all of our engineering, research and development functions, and substantially all of our products and services businesses, including our QCT semiconductor business.

Snapdragon and Qualcomm branded products are products of Qualcomm Technologies, Inc. and/or its subsidiaries. Qualcomm patented technologies are licensed by Qualcomm Incorporated.

Follow us on: [in](#) [X](#) [@](#) [v](#) [f](#)

For more information, visit us at [qualcomm.com](https://www.qualcomm.com) & [qualcomm.com/blog](https://www.qualcomm.com/blog)

