TEE subsystem

This document describes the TEE subsystem in Linux.

A TEE (Trusted Execution Environment) is a trusted OS running in some secure environment, for example, TrustZone on ARM CPUs, or a separate secure co-processor etc. A TEE driver handles the details needed to communicate with the TEE.

This subsystem deals with:

- Registration of TEE drivers
- Managing shared memory between Linux and the TEE
- Providing a generic API to the TEE

The TEE interface

include/uapi/linux/tee.h defines the generic interface to a TEE.

User space (the client) connects to the driver by opening /dev/tee[0-9]* or /dev/teepriv[0-9]*.

- TEE_IOC_SHM_ALLOC allocates shared memory and returns a file descriptor which user space can mmap. When user space doesn't need the file descriptor any more, it should be closed. When shared memory isn't needed any longer it should be unmapped with munmap() to allow the reuse of memory.
- TEE_IOC_VERSION lets user space know which TEE this driver handles and its capabilities.
- TEE_IOC_OPEN_SESSION opens a new session to a Trusted Application.
- TEE_IOC_INVOKE invokes a function in a Trusted Application.
- TEE_IOC_CANCEL may cancel an ongoing TEE_IOC_OPEN_SESSION or TEE_IOC_INVOKE.
- TEE_IOC_CLOSE_SESSION closes a session to a Trusted Application.

There are two classes of clients, normal clients and supplicants. The latter is a helper process for the TEE to access resources in Linux, for example file system access. A normal client opens /dev/tee[0-9]* and a supplicant opens /dev/teepriv[0-9].

Much of the communication between clients and the TEE is opaque to the driver. The main job for the driver is to receive requests from the clients, forward them to the TEE and send back the results. In the case of supplicants the communication goes in the other direction, the TEE sends requests to the supplicant which then sends back the result.

The TEE kernel interface

Kernel provides a TEE bus infrastructure where a Trusted Application is represented as a device identified via Universally Unique Identifier (UUID) and client drivers register a table of supported device UUIDs.

TEE bus infrastructure registers following APIs:

match():
: iterates over the client driver UUID table to find a corresponding match for device UUID. If a match is found, then this particular device is probed via corresponding probe API registered by the client driver. This process happens whenever a device or a client driver is registered with TEE bus.

uevent():
: notifies user-space (udev) whenever a new device is registered on TEE bus for auto-loading of modularized client drivers.

TEE bus device enumeration is specific to underlying TEE implementation, so it is left open for TEE drivers to provide corresponding implementation.

Then TEE client driver can talk to a matched Trusted Application using APIs listed in include/linux/tee_drv.h.

TEE client driver example

Suppose a TEE client driver needs to communicate with a Trusted Application having UUID: `ac6a4085-0e82-4c33-bf98-8eb8e118b6c2`, so driver registration snippet would look like:

```
static const struct tee_client_device_id client_id_table[] = {
        {UUID_INIT(0xac6a4085, 0x0e82, 0x4c33,
                   0xbf, 0x98, 0x8e, 0xb8, 0xe1, 0x18, 0xb6, 0xc2)},
        {}
};

MODULE_DEVICE_TABLE(tee, client_id_table);

static struct tee_client_driver client_driver = {
        .id_table       = client_id_table,
        .driver         = {
                .name           = DRIVER_NAME,
                .bus            = &tee_bus_type,
                .probe          = client_probe,
```

```
                .remove         = client_remove,
        },
};

static int __init client_init(void)
{
        return driver_register(&client_driver.driver);
}

static void __exit client_exit(void)
{
        driver_unregister(&client_driver.driver);
}

module_init(client_init);
module_exit(client_exit);
```

OP-TEE driver

The OP-TEE driver handles OP-TEE [1] based TEEs. Currently it is only the ARM TrustZone based OP-TEE solution that is supported.

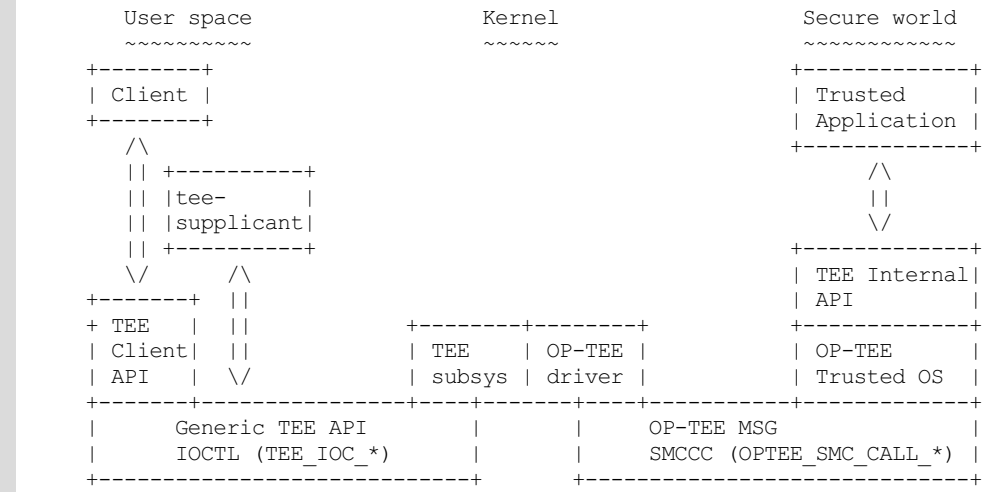
Lowest level of communication with OP-TEE builds on ARM SMC Calling Convention (SMCCC) [2], which is the foundation for OP-TEE's SMC interface [3] used internally by the driver. Stacked on top of that is OP-TEE Message Protocol [4].

OP-TEE SMC interface provides the basic functions required by SMCCC and some additional functions specific for OP-TEE. The most interesting functions are:

- OPTEE_SMC_FUNCID_CALLS_UID (part of SMCCC) returns the version information which is then returned by TEE_IOC_VERSION
- OPTEE_SMC_CALL_GET_OS_UUID returns the particular OP-TEE implementation, used to tell, for instance, a TrustZone OP-TEE apart from an OP-TEE running on a separate secure co-processor.
- OPTEE_SMC_CALL_WITH_ARG drives the OP-TEE message protocol
- OPTEE_SMC_GET_SHM_CONFIG lets the driver and OP-TEE agree on which memory range to used for shared memory between Linux and OP-TEE.

The GlobalPlatform TEE Client API [5] is implemented on top of the generic TEE API.

Picture of the relationship between the different components in the OP-TEE architecture:



RPC (Remote Procedure Call) are requests from secure world to kernel driver or tee-supplicant. An RPC is identified by a special range of SMCCC return values from OPTEE_SMC_CALL_WITH_ARG. RPC messages which are intended for the kernel are handled by the kernel driver. Other RPC messages will be forwarded to tee-supplicant without further involvement of the driver, except switching shared memory buffer representation.

OP-TEE device enumeration

OP-TEE provides a pseudo Trusted Application: drivers/tee/optee/device.c in order to support device enumeration. In other words, OP-TEE driver invokes this application to retrieve a list of Trusted Applications which can be registered as devices on the TEE bus.

OP-TEE notifications

There are two kinds of notifications that secure world can use to make normal world aware of some event.

1. Synchronous notifications delivered with `OPTEE_RPC_CMD_NOTIFICATION` using the `OPTEE_RPC_NOTIFICATION_SEND` parameter.
2. Asynchronous notifications delivered with a combination of a non-secure edge-triggered interrupt and a fast call from the

non-secure interrupt handler.

Synchronous notifications are limited by depending on RPC for delivery, this is only usable when secure world is entered with a yielding call via `OPTEE_SMC_CALL_WITH_ARG`. This excludes such notifications from secure world interrupt handlers.

An asynchronous notification is delivered via a non-secure edge-triggered interrupt to an interrupt handler registered in the OP-TEE driver. The actual notification value are retrieved with the fast call `OPTEE_SMC_GET_ASYNC_NOTIF_VALUE`. Note that one interrupt can represent multiple notifications.

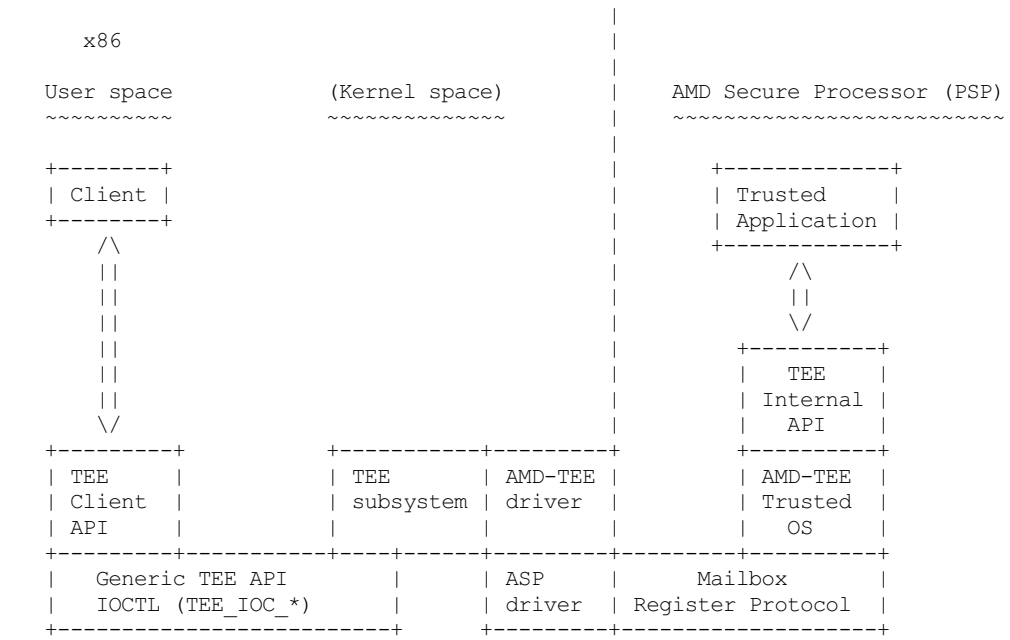
One notification value `OPTEE_SMC_ASYNC_NOTIF_VALUE_DO_BOTTOM_HALF` has a special meaning. When this value is received it means that normal world is supposed to make a yielding call `OPTEE_MSG_CMD_DO_BOTTOM_HALF`. This call is done from the thread assisting the interrupt handler. This is a building block for OP-TEE OS in secure world to implement the top half and bottom half style of device drivers.

AMD-TEE driver

The AMD-TEE driver handles the communication with AMD's TEE environment. The TEE environment is provided by AMD Secure Processor.

The AMD Secure Processor (formerly called Platform Security Processor or PSP) is a dedicated processor that features ARM TrustZone technology, along with a software-based Trusted Execution Environment (TEE) designed to enable third-party Trusted Applications. This feature is currently enabled only for APUs.

The following picture shows a high level overview of AMD-TEE:



At the lowest level (in x86), the AMD Secure Processor (ASP) driver uses the CPU to PSP mailbox register to submit commands to the PSP. The format of the command buffer is opaque to the ASP driver. It's role is to submit commands to the secure processor and return results to AMD-TEE driver. The interface between AMD-TEE driver and AMD Secure Processor driver can be found in [6].

The AMD-TEE driver packages the command buffer payload for processing in TEE. The command buffer format for the different TEE commands can be found in [7].

The TEE commands supported by AMD-TEE Trusted OS are:

- TEE_CMD_ID_LOAD_TA - loads a Trusted Application (TA) binary into TEE environment.
- TEE_CMD_ID_UNLOAD_TA - unloads TA binary from TEE environment.
- TEE_CMD_ID_OPEN_SESSION - opens a session with a loaded TA.
- TEE_CMD_ID_CLOSE_SESSION - closes session with loaded TA
- TEE_CMD_ID_INVOKE_CMD - invokes a command with loaded TA
- TEE_CMD_ID_MAP_SHARED_MEM - maps shared memory
- TEE_CMD_ID_UNMAP_SHARED_MEM - unmaps shared memory

AMD-TEE Trusted OS is the firmware running on AMD Secure Processor.

The AMD-TEE driver registers itself with TEE subsystem and implements the following driver function callbacks:

- get_version - returns the driver implementation id and capability.
- open - sets up the driver context data structure.
- release - frees up driver resources.
- open_session - loads the TA binary and opens session with loaded TA.

- close_session - closes session with loaded TA and unloads it.
- invoke_func - invokes a command with loaded TA.

cancel_req driver callback is not supported by AMD-TEE.

The GlobalPlatform TEE Client API [5] can be used by the user space (client) to talk to AMD's TEE. AMD's TEE provides a secure environment for loading, opening a session, invoking commands and closing session with TA.

References

[1] https://github.com/OP-TEE/optee_os

[2] http://infocenter.arm.com/help/topic/com.arm.doc.den0028a/index.html

[3] drivers/tee/optee/optee_smc.h

[4] drivers/tee/optee/optee_msg.h

[5] http://www.globalplatform.org/specificationsdevice.asp look for
"TEE Client API Specification v1.0" and click download.

[6] include/linux/psp-tee.h

[7] drivers/tee/amdtee/amdtee_if.h