1 OP-TEE

1.1 Introduction

1.1.1 About OP-TEE

OP-TEE is a Trusted Execution Environment (TEE) designed as companion to a non-secure Linux kernel. OP-TEE implements TEE Internal Core API which is the API exposed to Trusted Applications and the TEE Client API, which is the API describing how to communicate with a TEE. Those APIs are defined in the GlobalPlatform API specifications. The non-secure OS is referred to as the Rich Execution Environment (REE) in TEE specifications. It is typically a Linux OS flavor as a GNU/Linux distribution or the AOSP. OP-TEE is designed primarily to rely on the Arm TrustZone technology as the underlying hardware isolation mechanism. However, it has been structured to be compatible with any isolation technology suitable for the TEE concept and goals, such as running as a virtual machine or on a dedicated CPU.

1.1.2 Goals

- Isolation - the TEE provides isolation from the non-secure OS and protects the loaded Trusted Applications (TAs) from each other using underlying hardware support.
- Small footprint - the TEE should remain small enough to reside in a reasonable amount of on-chip memory as found on Arm based systems.
- Portability - the TEE aims at being easily pluggable to different architectures and available hardware and has to support various setups such as multiple client OSes or multiple TEEs.

1.2 Interrupt Handling

1.2.1 Cases

Here all the cases are listed where OP-TEE OS is involved in world context switches. Optee_os executes in the secure world. World switch is done by the core's secure monitor level/mode, referred below as the Monitor.

1. When the normal world invokes the secure world, the normal world executes a SMC instruction. The SMC exception is always trapped by the Monitor. If the related service targets the trusted OS, the Monitor will switch to OP-TEE OS world execution. When the secure world returns to the normal world, OP-TEE OS executes a SMC that is caught by the Monitor which switches back to the normal world.
2. When a secure interrupt is signaled by the Arm GIC, it shall reach the OP-TEE OS interrupt exception vector. If the secure world is executing, OP-TEE OS will handle interrupt straight from its exception vector. If

the normal world is executing when the secure interrupt raises, the Monitor vector must handle the exception and invoke OP-TEE OS to serve the interrupt.

3. When a non-secure interrupt is signaled by the Arm GIC, it shall reach the normal world interrupt exception vector. If the normal world is executing, it will handle straight the exception from its exception vector. If the secure world is executing when the non-secure interrupt raises, OP-TEE OS will temporarily return back to normal world via the Monitor to let normal world serve the interrupt.

1.2.2 Core Exceptions

All SMC exceptions are trapped in the Monitor vector. IRQ/FIQ exceptions can be trapped either in the Monitor vector or in the state vector of the executing world. When the normal world is executing, the system is configured to route:

- secure interrupts to the Monitor that will forward to OP-TEE OS
- non-secure interrupts to the executing world exception vector.

When the secure world is executing, the system is configured to route:

- secure and non-secure interrupts to the executing OP-TEE OS exception vector. OP-TEE OS shall forward the non-secure interrupts to the normal world.

1.2.3 Scheduling

OP-TEE yielding services are carried through standard SMC. Execution of these services can be interrupted by foreign interrupts. To suspend and restore the service execution, optee_os assigns a trusted thread at yielding SMC entry. The trusted thread terminates when optee_os returns to the normal world with a service completion status. A trusted thread execution can be interrupted by multiple events

- Native Interrupt: In this case the native interrupt is handled by the interrupt exception handlers and once served, optee_os returns to the execution trusted thread.
- Foreign Interrupt: In this case, optee_os suspends the trusted thread and invokes the normal world through the Monitor (optee_os so-called RPC services). The trusted threads will resume only once normal world invokes the optee_os with the RPC service status.
- A trusted thread execution can lead optee_os to invoke a service in normal world: access a file, get the REE current time, etc. The trusted thread is first suspended then resumed during remote service execution.

Optee_os does not implement any thread scheduling. Each trusted thread is expected to track a service that is invoked from the normal world and should return to it with an execution status.

The OP-TEE Linux driver is designed so that the Linux thread invoking OP-TEE gets assigned a trusted thread on TEE side. The execution of the trusted thread is tied to the execution of the caller Linux thread which is under the Linux kernel scheduling decision. This means trusted threads are scheduled by the Linux kernel.

1.3 Notifications

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

- Synchronous notifications
- Asynchronous notifications

Secure world can wait for a notification to arrive. This allows the calling thread to sleep instead of spinning when waiting for something. This happens for instance when a thread waits for a mutex to become available. Synchronous notifications are limited by depending on RPC for delivery, this is only usable from a normal thread context. Secure interrupt handler or other atomic context cannot use synchronous notifications due to this.

Asynchrononous notifications uses a platform specific way of triggering a non-secure interrupt. This is done in a way suitable for a secure interrupt handler or another atomic context. This is useful when using a top half and bottom half kind of design in a device driver. The top half is done in the secure interrupt handler which then triggers normal world to make a yielding call into secure world to do the bottom half processing.

1.4 Memory Objects

A memory object, MOBJ, describes a piece of memory. The interface provided is mostly abstract when it comes to using the MOBJ to populate translation tables etc. There are different kinds of MOBJs describing:

- Physically contiguous memory
- Virtual memory
- Physically contiguous memory allocated from a pool
- Paged memory
- Secure copy paged shared memory

1.5 Memory Management Unit

1.5.1 Translation Tables

OP-TEE supports two translation table formats:

- Short-descriptor translation table format, available on ARMv7-A and ARMv8-A AArch32.
- Long-descriptor translation format, available on ARMv7-A with LPAE and ARMv8-A.

ARMv8-A AArch64 must use the long-descriptor translation format only. Translation table format is a static build time configuration option. The design around the translation table handling has been centered around these factors:

- Share translation tables between CPUs when possible to save memory and simplify paging.
- Support non-global CPU specific mappings to allow executing different TAs in parallel.

1.6 Pager

OP-TEE currently requires ¿256 KB RAM for OP-TEE kernel memory. This is not a problem if OP-TEE uses TrustZone protected DDR, but for security reasons OP-TEE may need to use TrustZone protected SRAM instead. The amount of available SRAM varies between platforms, from just a few KB up to over 512 KB. Platforms with just a few KB of SRAM cannot be expected to be able to run a complete TEE solution in SRAM. But those with 128 to 256 KB of SRAM can be expected to have a capable TEE solution in SRAM. The pager provides a solution to this by demand paging parts of OP-TEE using virtual memory.

1.6.1 Secure Memory

TrustZone protected SRAM is generally considered more secure than TrustZone protected DRAM as there is usually more attack vectors on DRAM. The attack vectors are hardware dependent and can be different for different platforms.

1.6.2 Paging Shared Secured Memory

Shared secure memory is achieved by letting several tee_pager_area using the same backing fobj. When a tee_pager_area is allocated and assigned a fobj it's also added to a list for tee_pager_areas using this fobj. This helps when a physical page is released. When a fault occurs first a matching tee_pager_area is located. Then tee_pager_pmem_head is searched to see if a physical page already holds the page of the fobj needed. If so the pagetable is updated to

map the physical page at the appropriate location. If no physical page was holding the page a new physical page is allocated, initialized and finally mapped. In order to make as few updates to mappings as possible changes to less restricted, no access to read-only or read-only to read-write, is done only if the virtual address was used when the page fault occurred. Changes in the other direction have to be done in all translation tables used to map the physical page.

1.7 Stacks

- Secure monitor stack (128 bytes), bound to the CPU. Only available if OP-TEE is compiled with a secure monitor always the case if the target is Armv7-A but never for Armv8-A.
- Temp stack (small 1KB), bound to the CPU. Used when transitioning from one state to another. Interrupts are always disabled when using this stack, aborts are fatal when using the temp stack.
- Abort stack (medium 2KB), bound to the CPU. Used when trapping a data or pre-fetch abort. Aborts from user space are never fatal the TA is only killed. Aborts from kernel mode are used by the pager to do the demand paging, if pager is disabled all kernel mode aborts are fatal.
- Thread stack (large 8KB), not bound to the CPU instead used by the current thread/task. Interrupts are usually enabled when using this stack.

1.8 Shared Memory

Shared Memory is a block of memory that is shared between the non-secure and the secure world. It is used to transfer data between both worlds. The shared memory is allocated and managed by the non-secure world, i.e. the Linux OP-TEE driver. Secure world only considers the individual shared buffers, not their pool. Shared memory buffer references manipulated must fit inside one of the shared memory areas known from the OP-TEE core. OP-TEE supports two kinds of shared memory areas: an area for contiguous buffers and an area for noncontiguous buffers. At least one has to be enabled.

1.9 Thread Handling

OP-TEE core uses a couple of threads to be able to support running jobs in parallel. There are handlers for different purposes. In thread.c you will find a function called thread_init_primary(...) which assigns init_handlers (functions) that should be called when OP-TEE core receives standard or fast calls, FIQ and PSCI calls. There are default handlers for these services, but the platform can decide if they want to implement their own platform specific handlers instead.

1.9.1 Synchronization Primitives

- Spin-lock
- Mutex
- Condvar

1.10 Secure Boot

There are no additional specific build options for the verification of OP-TEE. If the authentication framework is enabled and specified the BL32 build option when building TF-A, the BL32 related certificates will be created automatically by the cert_create tool, and then these certificates will be verified during booting up.

1.10.1 Authentication Framework & Chain of Trust

The Trusted Firmware-A (TF-A) framework fulfills the following requirements:

- Possiblity for a platform port to specify the Chain of Trust in terms of certificate hierarchy and the mechanisms used to verify a particular certificate.
- The framework should distinguish between:
  - The mechanism used to encode and transport information.
  - The mechanism used to verify the transported information.

The framework has been designed following a modular approach.

1.10.2 Authentication Modes

The AM supports the following authentication methods:

- Hash
- Digital Signature

The platform may specify these methods in the Chain of Trust (CoT) in case it decides to define a custom CoT instead of reusing a predefined one. If a data image uses multiple methods, then all the methods must be a part of the same CoT. The number and type of parameters are method specific. These parameters should be obtained from the parent image using the IPM.

1.10.3 Specifying a Chain of Trust

A CoT can be described as a set of image descriptors linked together in a particular order. The order dictates the sequence in which they must be verified. Each image has a set of properties which allow the AM to verify it.

1.11 Secure Storage

Secure Storage in OP-TEE is implemented according to what has been defined in GlobalPlatform's TEE Internal Core API. This specification mandates that it should be possible to store general-purpose data and key material that guarantees confidentiality and integrity of the data stored and the atomicity of the operations that modifies the storage. There are currently two secure storage implementations in OP-TEE:

- Relying on the normal world (REE) file system. It is enabled at compile time by CFG_REE_FS=y.
- Making use of the Replay Protected Memory Block (RPMB) partition of an eMMC device, and is enabled by setting CFG_RPMB_FS=y.

They can be used simultaneously.

1.11.1 Key Manager

Key manager is a component in TEE file system, and is responsible for handling data encryption and decryption and also management of the sensitive key materials. There are three types of keys used by the key manager: the Secure Storage Key (SSK), the TA Storage Key (TSK) and the File Encryption Key (FEK).

1.11.2 Hash Tree

The hash tree is responsible for handling data encryption and decryption of a secure storage file. The hash tree is implemented as a binary tree where each node in the tree protects its two child nodes and a data block. The meta data is stored in a header which also protects the top node. All fields (header, nodes, and blocks) are duplicated with two versions, 0 and 1, to ensure atomic updates.

1.12 Trusted Application

1.12.1 Pseudo Trusted Application

A Pseudo Trusted Application is not a Trusted Application. A Pseudo TA is not a specific entity, it is an interface exposed by the OP-TEE Core to its outer world: to secure client Trusted Applications and to non-secure client entities. These are implemented directly to the OP-TEE core tree and are built along with and statically built into the OP-TEE core blob.

1.12.2 User Mode Trusted Applications

User Mode Trusted Applications are loaded (mapped into memory) by OP-TEE core in the Secure World when something in Rich Execution Environment (REE) wants to talk to that particular application UUID. They

run at a lower CPU privilege level than OP-TEE core code. In that respect, they are quite similar to regular applications running in the REE, except that they execute in Secure World.
Trusted Application benefit from the GlobalPlatform TEE Internal Core API as specified by the GlobalPlatform TEE specifications. There are several types of user mode TAs, which differ by the way they are stored.

Early TA The so-called early TAs are virtually identical to the REE FS TAs, but instead of being loaded from the Normal World file system, they are linked into a special data section in the TEE core blob. Therefore, they are available even before tee-supplicant and the REE's filesystems have come up. Please find more details in the early TA commit.

Secure Storage TA These are stored in secure storage. The meta data is stored in a database of all installed TAs and the actual binary is stored encrypted and integrity protected as a separate file in the untrusted REE filesystem (flash). Before these TAs can be loaded they have to be installed first, this is something that can be done during initial deployment or at a later stage.

1.12.3 Loading and Preparing for TA Execution

User mode TAs are loaded into final memory in the same way using the user mode ELF loader ldelf. The different TA locations have a common interface towards ldelf which makes the user mode operations identical regardless of how the TA is stored. After ldelf has returned with a TA prepared for execution it still remains in memory to serve the TA if dlopen() and friends are used. ldelf is also used to dump stack trace and detailed memory mappings if a TA is terminated via an abort.

1.13 Virtualization

OP-TEE has experimental virtualization support. This is when one OP-TEE instance can run TAs from multiple virtual machines. OP-TEE isolates all VM-related states, so one VM can't affect another in any way. With virtualization support enabled, OP-TEE will rely on a hypervisor, because only the hypervisor knows which VM is calling OP-TEE. Also, naturally the hypervisor should inform OP-TEE about creation and destruction of VMs. Besides, in almost all cases, hypervisor enables two-stage MMU translation, so VMs does not see real physical address of memory, instead they work with intermediate physical addresses (IPAs). On other hand OP-TEE can't translate IPA to PA, so this is a hypervisor's responsibility to do this kind of translation. So, hypervisor should include a component that knows about OP-TEE protocol internals and can do this translation. We call this component "TEE mediator" and right now only XEN hypervisor have OP-TEE mediator.

1.14 SPMC

The SPMC is a critical component in the FF-A flow. Some of its major responsibilities are:

- Initialisation and runtime management of the SPs. The SPMC component is responsible for initialisation of the Secure Partitions (loading the image, setting up the stack, heap, ...).
- Routing messages between endpoints. The SPMC is responsible for passing FF-A messages from normal world to SPs and back. It also responsible for passing FF-A messages between SPs.
- Memory management. The SPMC is responsible for the memory management of the SPs. Memory can be shared between SPs and between a SP to the normal world.

1.14.1 Secure Partitions

Secure Partitions (SPs) are the endpoints used in the FF-A protocol. When OP-TEE is used as a SPMC SPs run primarily inside S-EL0. OP-TEE will use FF-A for it transport layer when the OP-TEE CFG_CORE_FFA=y configuration flag is enabled. The SPMC will expose the OP-TEE core, privileged mode, as an secure endpoint itself. This is used to handle all GlobalPlatform programming mode operations. All GlobalPlatform messages are encapsulated inside FF-A messages. The OP-TEE endpoint will unpack the messages and afterwards handle them as standard OP-TEE calls. This is needed as TF-A (S-EL3) does only allow FF-A messages to be passed to the secure world when the SPMD is enabled. SPs run from the initial boot of the system until power down and don't have any built-in session management compared to GPD TEE TAs. The only means of communicating with the outside world is through messages defined in the FF-A specification. The context of a SP is saved between executions.

1.14.2 Running and exiting SPs

The SPMC resumes/starts the SP by calling the sp_enter(). This will set up the SP context and jump into S-EL0. Whenever the SP performs a system call it will end up in sp_handle_svc(). sp_handle_svc() stores the current context of the SP and makes sure that we don't return to S-EL0 but instead returns to S-EL1 back to sp_enter(). sp_enter() will pass the FF-A registers (x0-x7) to spmc_sp_msg_handler(). This will process the FF-A message.