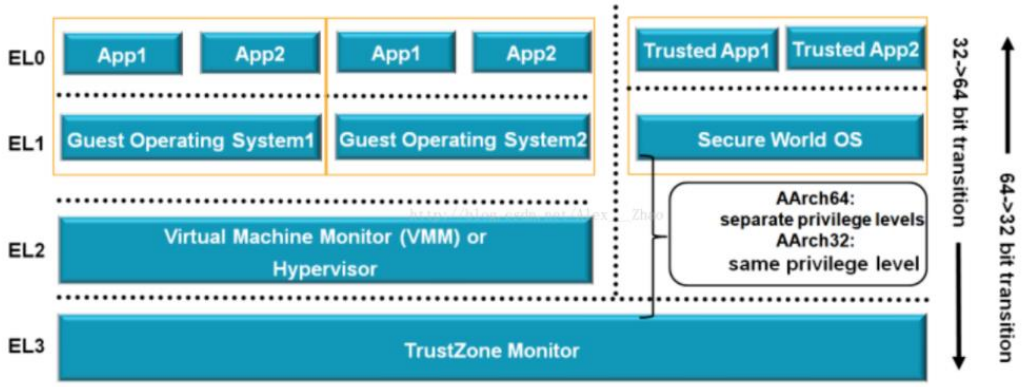
# OP-TEE Trusted Application on QorIQ ARM Based Processors

Trusted Execution Environment(TEE) is for ARM-based chips supporting TrustZone technology. NXP released LSDK integrates Open Portable TEE(OP-TEE), which is an open source project which contains a full implementation to make up a complete Trusted Execution Environment. This document will introduce OP-TEE architecture, OP-TEE OS loading and initialization, TA and CA communication in OPTEE runtime workflow, how to develop OP-TEE Trusted Application in LSDK environment.

## 1. OP-TEE Architecture

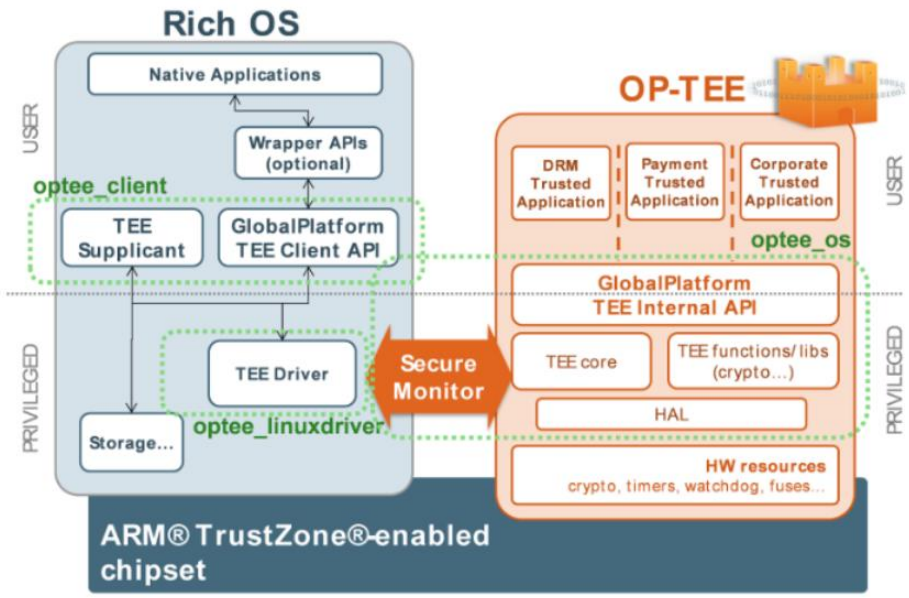
OP-TEE is based on ARM TrustZone to provide isolation of the TEE from the rich OS.
Please refer to the following Trustzone architecture in software perspective.




ARMV8 processor has four execution levels(EL0-EL3), EL3 divides a physical processor into two logical processors

OP-TEE consists of three components. OP-TEE Client, which is the client API running in normal world user space. OP-TEE Linux Kernel driver, which is the driver that handles the communication between normal world user space and secure world. OP-TEE Trusted OS, which is the Trusted OS running in secure world.

OP-TEE OS is made of 2 main components: the OP-TEE core and a collection of libraries designed for being used by Trusted Applications. While OP-TEE core executes in the ARM CPU privileged level (also referred to as 'kernel land'), the Trusted Applications execute in the non-privileged level (also referred to as the 'userland').




## 2. OP-TEE OS Loading and Initialization

OP-TEE binary is part of ppa.itb image as “loadables” node in ppa.itb image. In LSDK, please execute the following commands to generate ppa with OPTEE-OS enabled for ls1046ardb in the folder flexbuild/build/firmware/ppa/soc-ls1046/ppa.itb.

*$ flex-builder -c ppa-optee -m ls1046ardb*

OPTEE-OS tee_ls1046ardb.bin is generated with package optee_os, please execute the following commands to get tee_ls1046ardb.bin in the folder flexbuild/packages/apps/optee_os/out/arm-plat-ls/core/tee_ls1046ardb.bin.

*$ flex-builder -c optee_os -m ls1046ardb*

Please refer to the following ppa.its configuration for LS1046ARDB.

```
/{
        description = "PPA Firmware";
        #address-cells = <1>;
        images {
                firmware@1 {
                        description = "PPA Firmware: Version LSDK-18.03";
                        data = /incbin/("../obj/monitor.bin");
                        type = "firmware";
                        arch = "arm64";
                        compression = "none";
                };
                trustedOS@1 {
                        description = "Trusted OS";
                        data = /incbin/("../../tee.bin");
                        type = "OS";
```

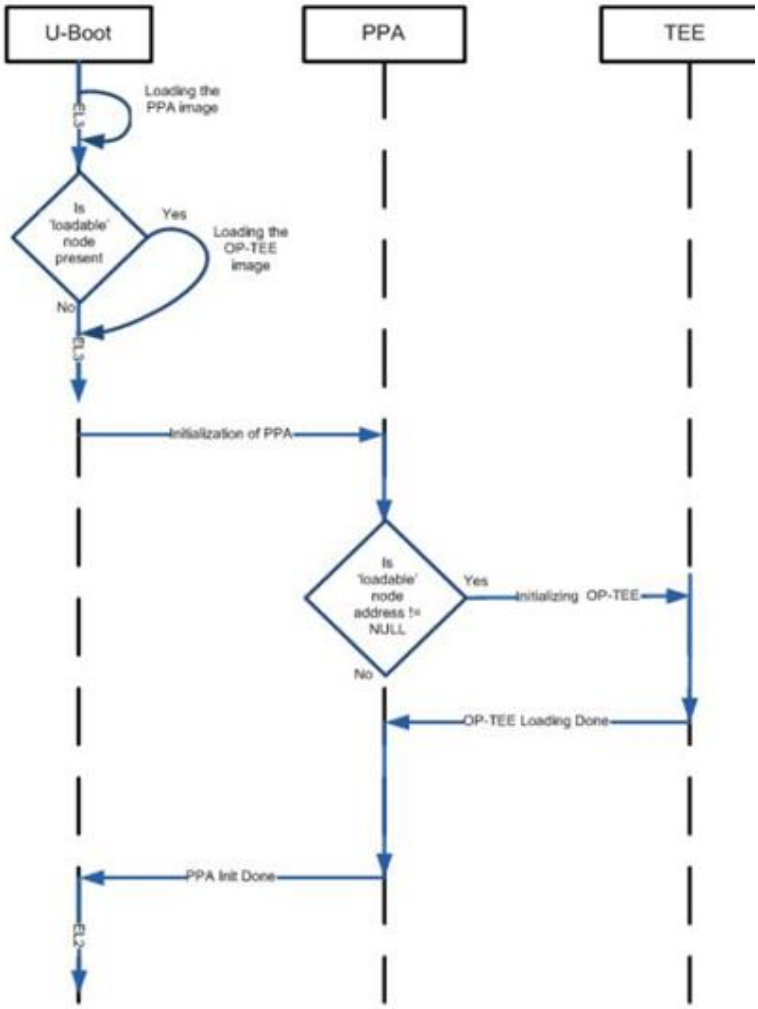
```
                arch = "arm64";
                compression = "none";
                load = <0x00200000>;
        };
};

configurations {
        default = "config@1";
        config@1 {
                description = "PPA Secure firmware";
                firmware = "firmware@1";
                loadables = "trustedOS@1";
        };
```

u-boot parses the ppa.itb image to check whether any loadables node is present in ppa.itb image, if it is present, u-boot loads the OP-TEE OS tee.bin, copy the binary tee.bin to DDR. After loading OP-TEE, u-boot passes the address where OP-TEE is loaded via SCRATCHRW registers to PPA.

After the binary tee.bin is loaded to the DDR, u-boot initiate the PPA initialization.

As part of OP-TEE initialization, PPA checks if loadables load address is not null, it initializes that OP-TEE.

Once the OP-TEE initialization is done, PPA initialization resumes. Before exiting to U-Boot after its initialization, PPA change the exception level from EL3 to EL2. Now, U-Boot run in EL2 mode.




## 3. OP-TEE Runtime Workflow

TEE(Trust Execution Envrionment) works together with REE(Rich Execution Envrionment), please refer to the following diagram for OP-TEE software work flow.

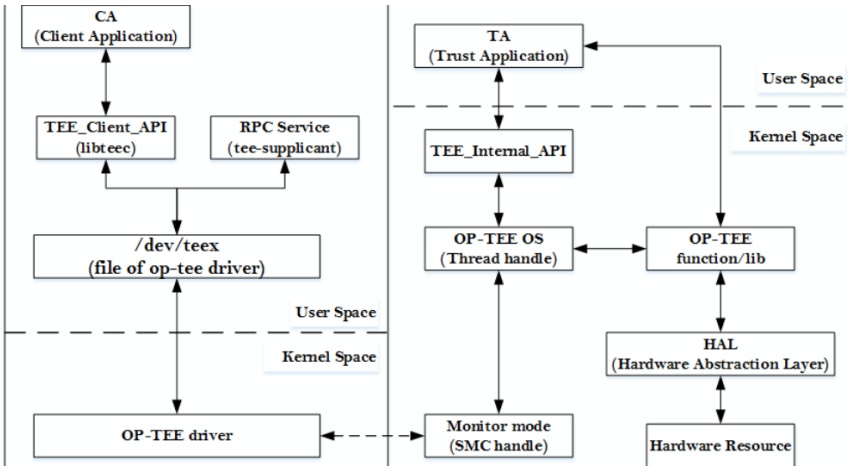
Tee-supplicant runs in user space, it will access the resource from the Rich filesystem through OP-TEE Linux driver ioctl calls.

OP-TEE provides the user a linux userspace APIs library libteec.

CA(Client Application) in userspace invokes system calls operation, the system will trap into Linux Kernel space to call the corresponding TEE driver.

In TEE driver, SMC instruction will be invoked to implement the communication with OP-TEE, SMC software handler will invoked to enter into Cortex monitor mode to control cortext to switch to secure world or non-secure world.

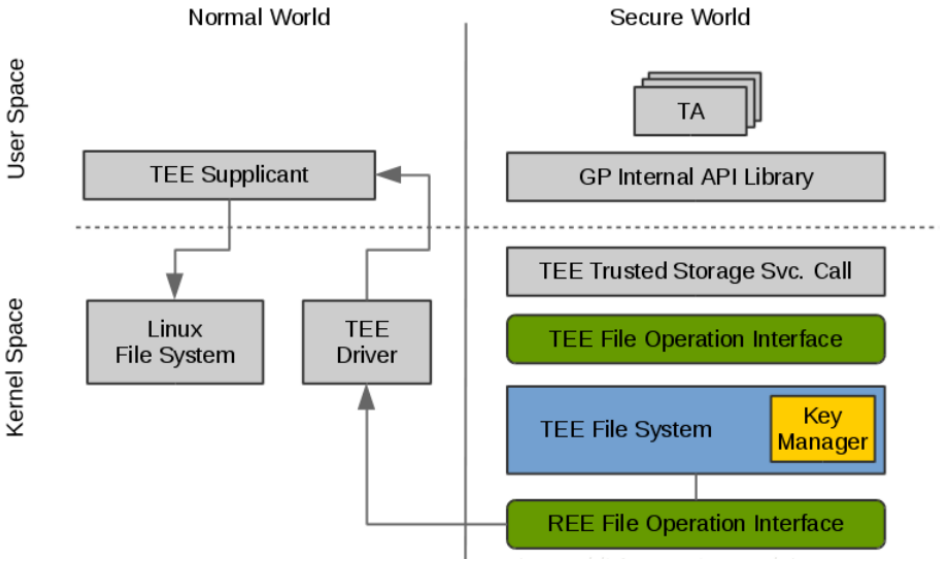
After SMC interrupt handler switching cortex to secure world and copy related parameters, TEE OS will take over the control. TEE OS will get the data passed from CA side, then parse UUID data related with TA, and looking for the corresponding TA image mounted in TEE OS. If the related TA image is not mounted, TEE OS will communicate with tee_supplicant, to get the corresponding TA image from the filesystem to load TA image to TEE OS. Then TEE OS will switch to TEE userspace, pass other parameters to PA process sent by CA. TA process parses the value of command ID in these parameters to perform the specific action.




## 4. Develop OP-TEE Trusted Application

The attached secure storage application demonstrates how to develop trusted application using OP-TEE.

The following is the structure of the storage demo application.




CA source code is in optee_storage/host folder, teec_uuid is defined as the following in storage_helpers.c, this value is used to open the specific TEEC session.

*TEEC_UUID teec_uuid = {*

*0x59e4d3d3, 0x0199, 0x4f74,*

*{ 0xb9, 0x4d, 0x53, 0xd3, 0xda, 0xa5, 0x7d, 0x73 }*

*};*

TA source code is located in optee_storage/ta, please refer to the following TEE_UUID definition in storage_ta_helper.h.

*#define TEE_UUID { 0x59e4d3d3, 0x0199, 0x4f74, *

*{ 0xb9, 0x4d, 0x53, 0xd3, 0xda, 0xa5, 0x7d, 0x73 } }*

Please refer to the following definition in TA Makefile.

*BINARY = 59e4d3d3-0199-4f74-b94d-53d3daa57d73*

Please refer to the following command IDs and related APIs definition on CA side.

```
/* Command ID definiton */
#define CMD_CREATE_OPERATION                1
#define CMD_READ_OPERATION                  2
#define CMD_WRITE_OPERATION                 3
#define CMD_TRUNCATE_OPERATION              4
#define CMD_RENAME_OPERATION                5
#define CMD_DELETE_OPERATION                6
#define CMD_TEST_OPERATION                  7

extern int storage_file_create(uint32_t len, char *name);
extern int storage_file_read(uint32_t file_len, char *file_name, uint32_t len, char *buf);
extern int storage_file_write(uint32_t file_len, char *file_name, uint32_t len, char *buf);
extern int storage_file_truncate(uint32_t file_len, char *file_name, uint32_t size);
extern int storage_file_rename(uint32_t old_len, char *old_name, uint32_t len, char *name);
extern int storage_file_delete(uint32_t len, char *name);
```

Please refer to the following command IDs and APIs definition on TA side.

```
/* Command ID definiton */
#define CMD_CREATE_OPERATION                1
#define CMD_READ_OPERATION                  2
#define CMD_WRITE_OPERATION                 3
#define CMD_TRUNCATE_OPERATION              4
#define CMD_RENAME_OPERATION                5
#define CMD_DELETE_OPERATION                6
#define CMD_TEST_OPERATION                  7

TEE_Result storage_ta_file_create(uint32_t paramTypes, TEE_Param params[4]);
TEE_Result storage_ta_file_read(uint32_t paramTypes, TEE_Param params[4]);
TEE_Result storage_ta_file_write(uint32_t paramTypes, TEE_Param params[4]);
TEE_Result storage_ta_file_truncate(uint32_t paramTypes, TEE_Param params[4]);
TEE_Result storage_ta_file_rename(uint32_t paramTypes, TEE_Param params[4]);
TEE_Result storage_ta_file_delete(uint32_t paramTypes, TEE_Param params[4]);

Please refer to CA and TA communication context setup in host/storage_helpers.c.
int storage_file_create(uint32_t len, char *name)
{
…
        /* Set communication context between CA and TA */
        memset(&operation, 0x0, sizeof(TEEC_Operation));
        operation.started = 1;
        operation.paramTypes = TEEC_PARAM_TYPES(TEEC_MEMREF_TEMP_INPUT,
TEEC_NONE, TEEC_NONE, TEEC_NONE);
        operation.params[0].tmpref.size = len;
        operation.params[0].tmpref.buffer = name;
```

```
        /* Send command to TA */
        ret = teec_command_invoke(&operation, &session, CMD_CREATE_OPERATION);
...
}
```

The following section discusses how to build OP-TEE trusted application into LSDK rootfs filesystem.
Please build optee_os and optee_client with flex-builder in LSDK.

*$ flex-builder -c optee_os  -a arm64*

*$ flex-builder -c optee_client  -a arm64*

*After set up Toolchain environment for ARM64.*

*$ make TA_DEV_KIT_DIR=<optee_os path>/out/arm-plat-ls/export-ta_arm64*
*TEEC_EXPORT=<optee_client path>/out/export*

Final CA image and TA image,
*CA: host/storage-test*
*TA: ta/59e4d3d3-0199-4f74-b94d-53d3daa57d73.ta*

Copy TA image to /lib/optee_libtz/ and CA to root file system.
$ tee-supplicant /dev/teepriv0 &
$ ./storage-test > log