

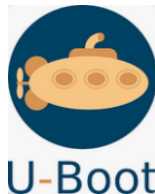
A Journey To the Secure World of OPTEE From the Non-secure World of Linux

Date: 2025-08-27

Manorit Chawdhry

J Keerthy

About us: TI Processors and Open Source



Upstream FIRST!



Open
Source

- ✓ Upstream FIRST mentality!
- ✓ Decades of contribution and collaboration
- ✓ Focus on long term, sustainable and quality products
- ✓ Upstream & opensource ecosystem in device architecture
- ✓ Ingrained culture to give back to the community

Introduction to the Speakers

Manorith Chawdhry, Senior Software Engineer at Texas Instruments, Bengaluru.

- Security in Jacinto SoCs.
- U-boot, TF-A, OP-TEE, Linux device driver development.



Keerthy Jagadish, Jacinto Apps Manager at Texas Instruments, Bengaluru.

- Linux, Thermal Power, Early bootup.

Overview

- Basics of Arm TrustZone
- What is OP-TEE
- How do we interact with OP-TEE
 - Pseudo Trusted application
 - User Mode Trusted application
 - How to choose a trusted application
- Control flow of OP-TEE
- Extended OTP trusted application

Disclaimers

- This is a technology presentation, not product-readiness or roadmap commitment
- Opinions presented here are that of the speakers and may not reflect that of Texas Instruments Inc.

Why need a secure world



Foundation of secure services (DRM, biometrics, payments)



Protects sensitive data (keys, credentials)

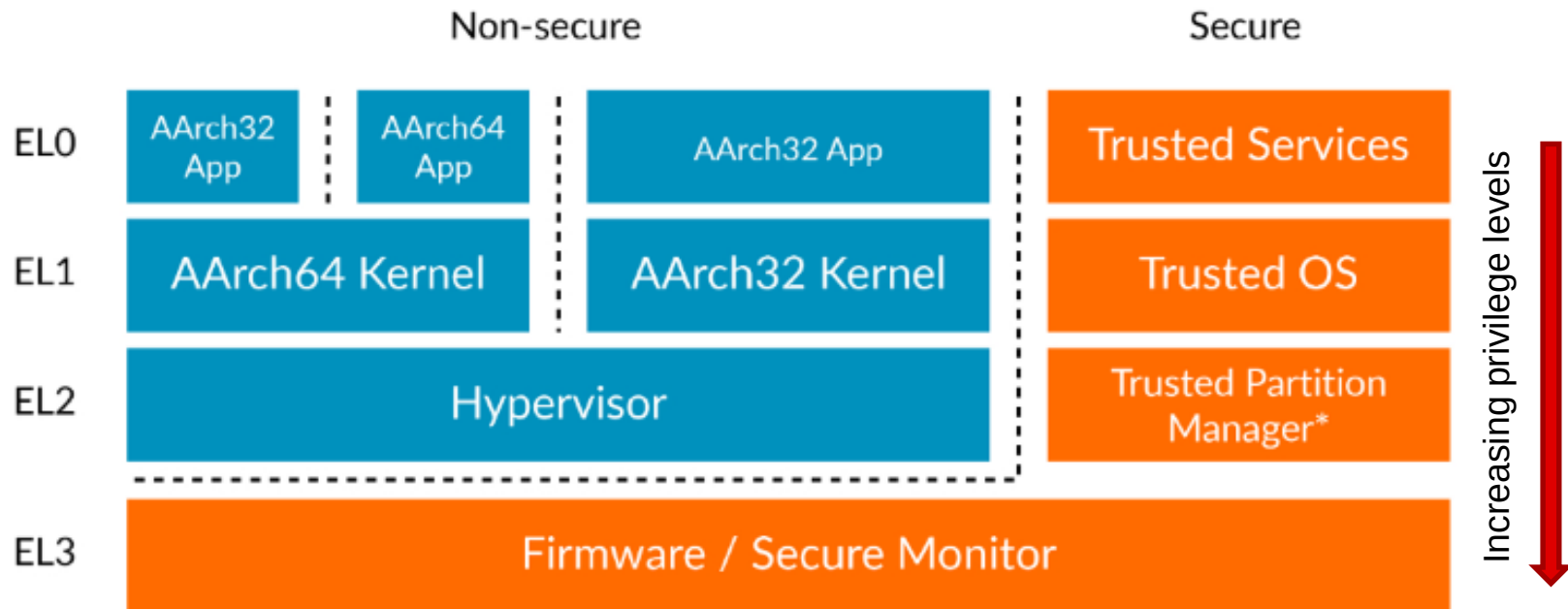


Shields these sensitive applications in case of compromise



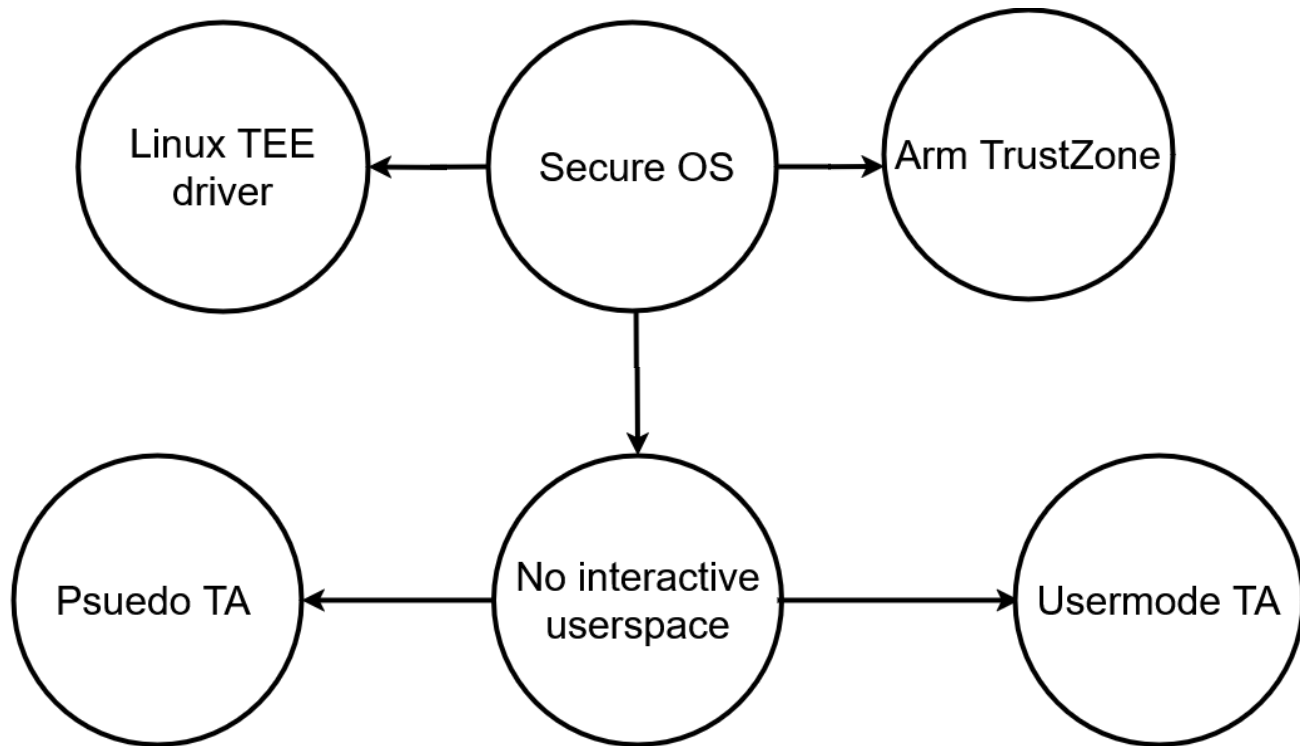
Used in phones, IoT devices, and payment systems

Basics of Arm TrustZone



* Secure EL2 from Armv8.4-A

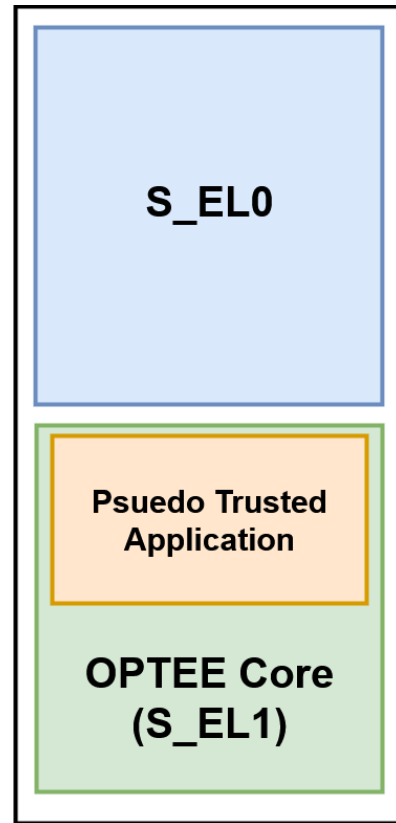
What is OP-TEE



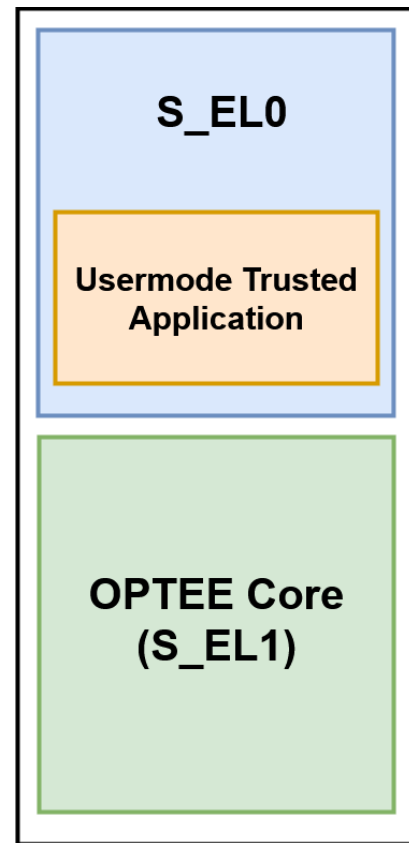
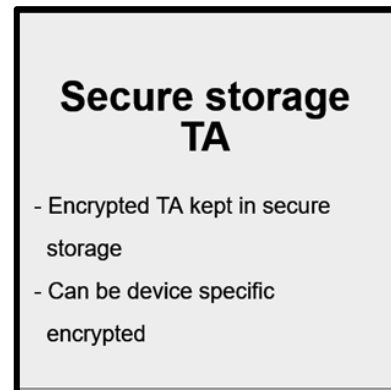
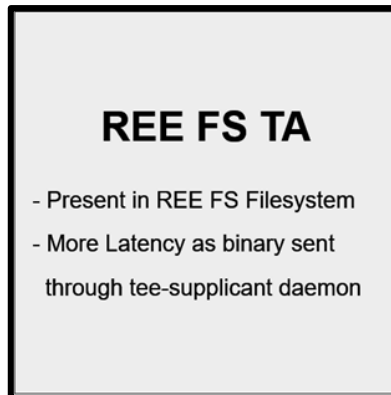
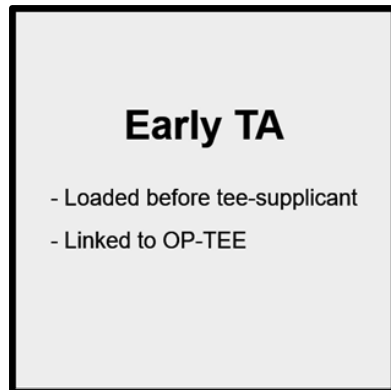
Pseudo Trusted application

- Size of OP-TEE ↑
- Faster loading
- Bugs can crash the OP-TEE kernel.
- SCMI, HWRNG, etc.

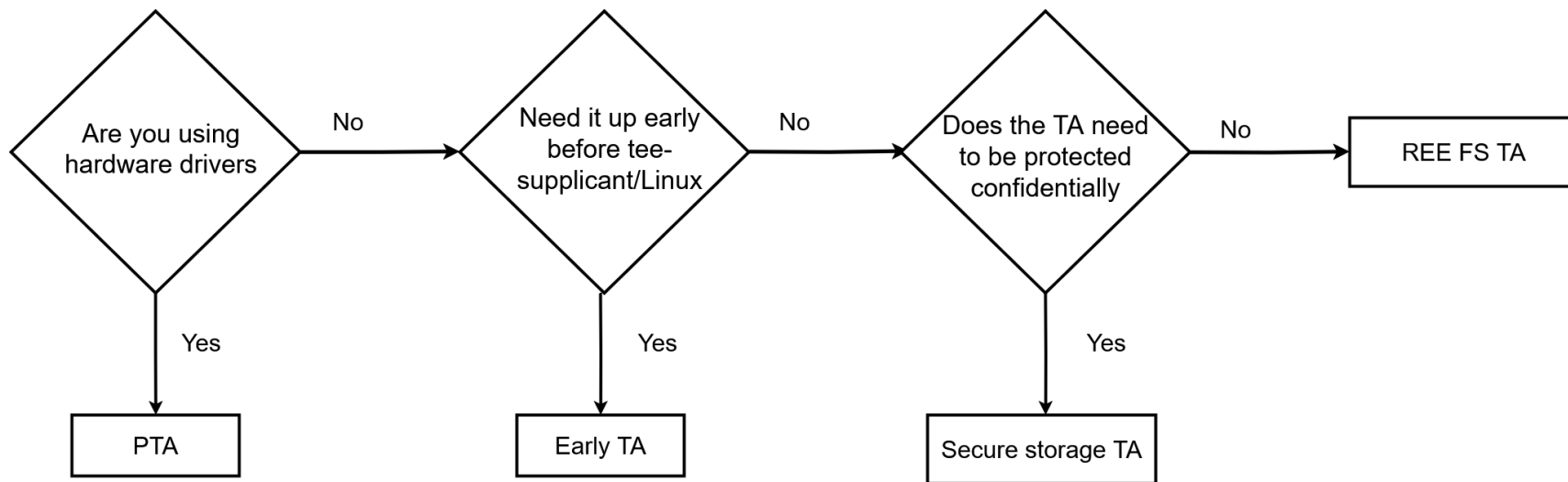
(https://github.com/OP-TEE/optee_os/tree/master/core/pta)



User Mode Trusted application



How to choose a trusted application



Control flow of OP-TEE

There are 3 core components of OP-TEE

- OP-TEE OS
- OP-TEE Client
- OP-TEE Kernel Driver

Control flow of OP-TEE (EL0)

Step-1: Initializing the context

Host Application (EL0)

```
/* Initialize a context connecting us to the TEE */  
res = TEEC_InitializeContext(NULL, &ctx->ctx);  
if (res != TEEC_SUCCESS)  
    errx(1, "TEEC_InitializeContext failed with code 0x%x", res);
```

OP-TEE Client (EL0)

libtee/src/tee_client_api.c

```
TEEC_Result TEEC_InitializeContext(const char *name,  
                                   TEEC_Context *ctx)
```

- Opens the first */dev/tee** device
- Stores it in *TEEC_Context *ctx*

Control flow of OP-TEE

Step-2: Opening the Session

Host Application (EL0)

```
/* Open a session with the TA */  
res = TEEC_OpenSession(&ctx->ctx, &ctx->sess, &uuid,  
    TEEC_LOGIN_PUBLIC, NULL, NULL,  
    &origin);
```

- Calls TEEC_OpenSession()

OP-TEE Client (EL0)

libtee/src/tee_client_api.c

```
TEEC_Result TEEC_OpenSession(TEEC_Context *ctx,  
    TEEC_Session *session, const TEEC_UUID *destination,  
    uint32_t connection_method, const void *connection_data,  
    TEEC_Operation *operation, uint32_t *ret_origin)
```

- Creates shared memory to pass to Kernel
- Passes struct tee_ioctl_open_session_arg through those buffers

Control flow of OP-TEE

```
switch (cmd) {  
case TEE_IOC_VERSION:  
    return tee_ioctl_version(ctx, uarg);  
case TEE_IOC_SHM_ALLOC:  
    return tee_ioctl_shm_alloc(ctx, uarg);  
case TEE_IOC_SHM_REGISTER:  
    return tee_ioctl_shm_register(ctx, uarg);  
case TEE_IOC_OPEN_SESSION:  
    return tee_ioctl_open_session(ctx, uarg);  
case TEE_IOC_INVOKE:  
    return tee_ioctl_invoke(ctx, uarg);  
case TEE_IOC_CANCEL:  
    return tee_ioctl_cancel(ctx, uarg);  
case TEE_IOC_CLOSE_SESSION:  
    return tee_ioctl_close_session(ctx, uarg);  
case TEE_IOC_SUPPL_RECV:  
    return tee_ioctl_supp_recv(ctx, uarg);  
case TEE_IOC_SUPPL_SEND:  
    return tee_ioctl_supp_send(ctx, uarg);  
default:  
    return -EINVAL;  
}
```

TEE Kernel Driver (EL1)

drivers/tee/tee_core.c: tee_ioctl(..)

- Generic IOCTL handler for any TEE
- tee_ioctl_open_session redirects to the implementation specific tee driver (amdtee, optee, etc.)

```
rc = ctx->teedev->desc->ops->open_session(ctx, &arg, params);
```

OP-TEE Kernel Driver (EL1)

tee/optee/call.c

```
int optee_open_session(struct tee_context *ctx, struct  
tee_ioctl_open_session_arg *arg, struct tee_param *param)
```

- Takes the arguments from the userspace passed struct
- Figures out the client UUID who called it and populate it's own message buffer with cmd = `OPTEE_MSG_CMD_OPEN_SESSION`

```
static int optee_smc_do_call_with_arg(struct tee_context *ctx,  
struct tee_shm *shm, u_int offs, bool system_thread)
```

- Passes the buffer pointers through the registers to SMC call
(For simplicity, the HVC calls are being ignored)

Control flow of OP-TEE

TF-A OP-TEE Driver (EL3)

services/spd/opteed/opteed_main.c

```
static uintptr_t opteed_smc_handler(uint32_t smc_fid, u_register_t x1,  
u_register_t x2, u_register_t x3, u_register_t x4, void *cookie, void *handle,  
u_register_t flags)
```

This is the generic handler that handles the communication and context switches to/from the secure world to the non-secure world, including saving and restoring the context between each world

Control flow of OP-TEE

Non-secure World Caller (Linux calling to TF-A)

A. OP-TEE Image Loading (if `OPTEE_ALLOW_SMC_LOAD` is enabled)

- If `smc_fid == NSSMC_OPTEED_CALL_UID`:
Returns the UUID of the image loading service
- If `smc_fid == NSSMC_OPTEED_CALL_LOAD_IMAGE`:
Handles loading the OP-TEE image via SMC

B. Regular SMC Handling

- Saves non-secure CPU context
- Checks if OP-TEE is loaded
- Initialises CPU context
- Sets up entry point based on SMC type (fast or yielding)
- Transfers control to OP-TEE
- Passes parameters from the non-secure world to OP-TEE

Control flow of OP-TEE

Secure World Caller (OP-TEE returning to TF-A)

A. Initialization Complete

- TEESMC_OPTEE_RETURN_ENTRY_DONE: OP-TEE has finished initialization
 - Stores vector table, registers power management hooks, and interrupt handlers
 - Returns to original C runtime context

B. Power Management Returns

- Handles the PSCI return states
- TEESMC_OPTEE_RETURN_{ON,RESUME,OFF,SUSPEND}_DONE
- TEESMC_OPTEE_RETURN_{SYSTEM_OFF,SYSTEM_RESET}_DONE

C. Normal Operation Returns

- TEESMC_OPTEE_RETURN_CALL_DONE: OP-TEE completed a request
 - Saves secure context, restores non-secure context
 - Returns results to non-secure world

D. Interrupt Handling

- TEESMC_OPTEE_RETURN_FIQ_DONE: OP-TEE finished handling an interrupt
 - Restores non-secure context and returns to non-secure world



Control flow of OP-TEE

optee_os (S_EL1)

core/arch/arm/kernel/thread_optee_smc.c

```
void thread_handle_fast_smc(struct thread_smc_args *args)
```

core/arch/arm/kernel/thread_optee_smc.c

```
uint32_t thread_handle_std_smc(uint32_t a0, uint32_t a1, uint32_t a2,  
uint32_t a3, uint32_t a4, uint32_t a5, uint32_t a6 __unused, uint32_t a7  
__maybe_unused)
```

Yielding SMC

- If non-secure world initiated a call - Spawn a new thread
- If OP-TEE thread initiated a call - Resume that thread

Fast SMC

- Doesn't create any threads as such
- Doesn't enable the interrupts, non-preemptible

Control flow of OP-TEE

```
void __tee_entry_fast(struct thread_smc_args *args)
{
    switch (args->a0) {

        /* Generic functions */
        case OPTEE_SMC_CALLS_COUNT:
            tee_entry_get_api_call_count(args);
            break;
        case OPTEE_SMC_CALLS_UUID:
            tee_entry_get_api_uuid(args);
            break;
        case OPTEE_SMC_CALLS_REVISION:
            tee_entry_get_api_revision(args);
            break;
        case OPTEE_SMC_CALL_GET_OS_UUID:
            tee_entry_get_os_uuid(args);
            break;
        ...
    }
}
```

```
TEE_Result __tee_entry_std(struct optee_msg_arg *arg, uint32_t num_params)
{
    TEE_Result res = TEE_SUCCESS;

    /* Enable foreign interrupts for STD calls */
    thread_set_foreign_intr(true);
    switch (arg->cmd) {
        case OPTEE_MSG_CMD_OPEN_SESSION:
            entry_open_session(arg, num_params);
            break;
        case OPTEE_MSG_CMD_CLOSE_SESSION:
            entry_close_session(arg, num_params);
            break;
        case OPTEE_MSG_CMD_INVOKE_COMMAND:
            entry_invoke_command(arg, num_params);
            break;
        case OPTEE_MSG_CMD_CANCEL:
            entry_cancel(arg, num_params);
            break;
        ...
    }
}
```

OPTEE_MSG_CMD_OPEN_SESSION
under a yielding SMC call

Control flow of OP-TEE

optee_os (S_EL1)

core/tee/entry_std.c

```
static void entry_open_session(struct optee_msg_arg *arg, uint32_t num_params)|
```

core/kernel/tee_ta_manager.c

```
TEE_Result tee_ta_open_session(TEE_ErrorOrigin *err, struct tee_ta_session **sess,  
struct tee_ta_session_head *open_sessions, const TEE_UUID *uuid,  
const TEE_Identity *clnt_id, uint32_t cancel_req_to, struct tee_ta_param *param)
```

core/kernel/tee_ta_manager.c

```
static TEE_Result tee_ta_init_session(TEE_ErrorOrigin *err,  
struct tee_ta_session_head *open_sessions, const TEE_UUID *uuid,  
struct tee_ta_session **sess)
```

core/kernel/pseudo_ta.c

```
static TEE_Result pseudo_ta_enter_open_session(struct ts_session *s)
```

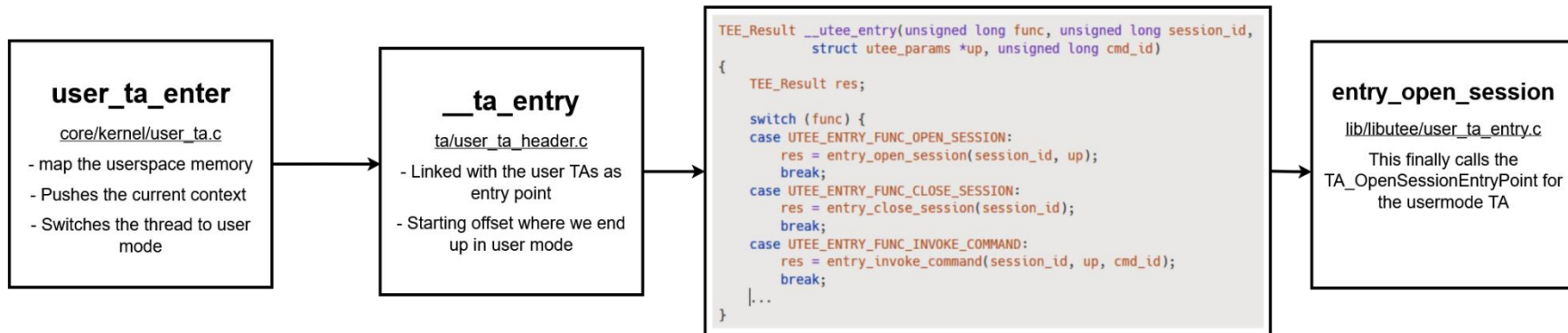
- struct ts_session has all the necessary info to redirect to the designated PTA
- This jumps to the OpenSessionEntryPoint of the PTA

core/kernel/user_ta.c

```
static TEE_Result user_ta_enter_open_session(struct ts_session *s)
```

- Here we diverge to go to user_ta_enter with UTEE_ENTRY_FUNC_OPEN_SESSION as the parameter

Control flow of OP-TEE



Extended OTP application

Why the need?

Customers use one-time programmable E-fuse bits (extended OTP) for storing versioning information, which is useful during OTA updates, custom stacks, etc.

For this, we needed a host application from Linux that could help securely write these values.

Why OP-TEE?

The TI foundational security (TIFS) exposes some APIs to write these bits in an access control manner to allow only secure hosts to write to this.

```
static TEE_Result invoke_command(void *session __unused,
                                uint32_t command, uint32_t param_types,
                                TEE_Param params[4])
{
    switch (command) {
        case TA_OTP_KEYWRITING_CMD_READ_MMR:
            return read_otp_mmr(param_types, params);
        case TA_OTP_KEYWRITING_CMD_WRITE_ROW:
            return write_otp_row(param_types, params);
        case TA_OTP_KEYWRITING_CMD_LOCK_ROW:
            return lock_otp_row(param_types, params);
        default:
            EMSG("Command ID 0x%"PRIx32" is not supported", command);
            return TEE_ERROR_NOT_SUPPORTED;
    }
}

pseudo_ta_register(.uuid = PTA_K3_OTP_KEYWRITING_UUID,
                   .name = PTA_K3_OTP_KEYWRITING_NAME,
                   .flags = PTA_DEFAULT_FLAGS,
                   .invoke_command_entry_point = invoke_command);
```



Extended OTP application

- The non-secure world invokes a command call, and that goes through the entire stack to reach our pseudo-trusted application to write only the bits exposed through the extended OTP

```
op.paramTypes = TEEC_PARAM_TYPES(TEEC_VALUE_INPUT, TEEC_VALUE_INPUT,  
                                TEEC_NONE, TEEC_NONE);  
  
op.params[0].value.a = index_row;  
op.params[1].value.a = data;  
op.params[1].value.b = mask;  
  
res = TEEC_InvokeCommand(&ctx->sess, TA_OTP_KEYWRITING_CMD_WRITE_ROW,  
                        &op, &origin);
```

Host side (EL0)

Extended OTP application

Secure side (S EL1)

Only OP-TEE has access to call the following APIs

OP-TEE exposes only the extended OTP related APIs so that the non-secure side (Linux) can write/read and lock it.

```
static TEE_Result write_otp_row(uint32_t param_types, TEE_Param params[4])
{
    TEE_Result ret = TEE_SUCCESS;
    const uint32_t exp_param_types =
        TEE_PARAM_TYPES(TEE_PARAM_TYPE_VALUE_INPUT,
                        TEE_PARAM_TYPE_VALUE_INPUT, TEE_PARAM_TYPE_NONE,
                        TEE_PARAM_TYPE_NONE);

    /*
     * Safely get the invocation parameters
     */
    if (param_types != exp_param_types)
        return TEE_ERROR_BAD_PARAMETERS;

    ret = ti_sci_write_otp_row(params[0].value.a, params[1].value.a,
                              params[1].value.b);
    if (ret)
        return ret;

    DMSG("Written the value: 0x%08"PRIx32, params[1].value.a);

    return TEE_SUCCESS;
}
```

Extended OTP application

Writing to the extended
OTP bits

```
root@j7200-hs-evm:~# optee_example_otp_keywriting --write --val 0xdead --mask
0xffff --index 4
val: 0x0000dead
mask: 0x0000ffff
D/TC:? 0 tee_ta_init_session_with_context:605 Re-open TA acc50f40-0613-4abd-8dfe-
a964cb74eb69
Prepare session with the TA
F/TC:? 0 k3_sec_proxy_send:131 Verifying the thread
- Write the OTP Row
F/TC:? 0 k3_sec_proxy_verify_thread:71 Check for thread corruption
F/TC:? 0 k3_sec_proxy_verify_thread:88 Check for thread direction
F/TC:? 0 k3_sec_proxy_verify_thread:100 Check for thread queue
F/TC:? 0 k3_sec_proxy_verify_thread:113 Success
F/TC:? 0 k3_sec_proxy_recv:193 Verifying thread
F/TC:? 0 k3_sec_proxy_verify_thread:71 Check for thread corruption
F/TC:? 0 k3_sec_proxy_verify_thread:88 Check for thread direction
F/TC:? 0 k3_sec_proxy_verify_thread:100 Check for thread queue
F/TC:? 0 k3_sec_proxy_verify_thread:113 Success
D/TC:? 0 ti_sci_write_otp_row:429 resp.row_val: 0x0000dead
I/TC: Written the value: 0x0000dead

We're done, close and release TEE resources
F/TC:? 0 plat_rng_add_jitter_entropy:72 0x14
D/TC:? 0 tee_ta_close_session:510 csess 0x9e8815d0 id 2
D/TC:? 0 tee_ta_close_session:529 Destroy session
```

Extended OTP application

Reading from the
extended OTP bits

```
root@j7200-hs-evm:~# optee_example_otp_keywriting --read --index 4
val: 0x00000000
D/TC:? @ tee_ta_init_pseudo_ta_session:296 Lookup pseudo TA
acc50f40-0613-4abd-8dfe-a964cb74eb69
mask: 0xffffffff
Prepare session with the TA
D/TC:? @ tee_ta_init_pseudo_ta_session:309 Open pta_k3_otp.ta
D/TC:? @ tee_ta_init_pseudo_ta_session:326 pta_k3_otp.ta :
acc50f40-0613-4abd-8dfe-a964cb74eb69
- Read the OTP MMR
F/TC:? @ k3_sec_proxy_send:131 Verifying the thread
F/TC:? @ k3_sec_proxy_verify_thread:71 Check for thread corruption
F/TC:? @ k3_sec_proxy_verify_thread:88 Check for thread direction
F/TC:? @ k3_sec_proxy_verify_thread:100 Check for thread queue
F/TC:? @ k3_sec_proxy_verify_thread:113 Success
F/TC:? @ k3_sec_proxy_recv:193 Verifying thread
F/TC:? @ k3_sec_proxy_verify_thread:71 Check for thread corruption
F/TC:? @ k3_sec_proxy_verify_thread:88 Check for thread direction
F/TC:? @ k3_sec_proxy_verify_thread:100 Check for thread queue
F/TC:? @ k3_sec_proxy_verify_thread:113 Success
I/TC: Got the value: 0xdeadbeef // wrote deadbeef successfully
Value from TA: 0xdeadbeef
D/TC:? @ tee_ta_close_session:510 csess 0x9e881780 id 2

We're done, close and release TEE resources
D/TC:? @ tee_ta_close_session:529 Destroy session
```

References

- <https://optee.readthedocs.io/en/latest/general/about.html>
- <https://slava-moskvin.medium.com/arm-trustzone-a-crash-course-for-reversers-923e5812eccb>
- https://software-dl.ti.com/tisci/esd/latest/2_tisci_msgs/security/extended_opt.html
- <https://trustedfirmware-a.readthedocs.io/en/v2.12/components/spd/optee-dispatcher.html>
- <https://docs.kernel.org/tee/op-tee.html>

Credits and Acknowledgement

- Texas Instruments Inc.
- The Linux Foundation.

Q&A

- Contact Information:
 - Manorit Chawdhry <m-chawdhry@ti.com>
 - J Keerthy <j-keerthy@ti.com>
- Also on IRC @ libera.chat #linux-ti

Learn more about TI products

- <https://www.ti.com/linux>
- <https://www.ti.com/processors>
- <https://www.ti.com/edgeai>



Why choose TI MCUs and processors?

✓ Scalability

Our products offer scalable performance that can adapt and grow as the needs of your customers evolve.

✓ Efficiency

We design products that extend battery life, maximize performance for every watt expended, and unlock the highest levels of system efficiency.

✓ Affordability

We strive to make innovation accessible to all by creating cost-effective products that feature state-of-the-art technology and package designs.

✓ Availability

Our investment in internal manufacturing capacity provides greater assurance of supply, supporting your growth for decades to come.