SWG updates since HKG18

YVR18-117

OP-TEE: TA dynamic link

Jerome Forissier

TA Dynamic Link

OP-TEE 3.2.0+ supports shared libraries in Secure World (32- and 64-bits)

```
# Link and sign a shared library
ld -shared --soname=UUID1 -o libfoo.so *.o …
sign.py --in libfoo.so --out UUID1.ta …

# Link and sign a Trusted Application
ld -o UUID2.elf *.o … -lfoo
sign.py --in UUID2.elf --out UUID2.ta …
```

TA Dynamic Link - Usage Example

```
# Library Makefile
SHLIBNAME = libos_test
SHLIBUUID = ffd2bded-ab7d-4988-95ee-e4962fff7154
include $(TA_DEV_KIT)/mk/ta_dev_kit.mk

# TA Makefile
BINARY = 5b9e0e40-2636-11e1-ad9e-0002a5d5c51b
LDADD = -L... -los_test
include $(TA_DEV_KIT)/mk/ta_dev_kit.mk
```

https://github.com/OP-TEE/optee_test/tree/3.2.0/ta/os_test_lib
https://github.com/OP-TEE/optee_test/blob/3.2.0/ta/os_test

TA Dynamic Link - Implementation

The ELF loader in OP-TEE core has been modified to:

1. Locate the PT_DYNAMIC entry in the program headers
2. Parse the DT_NEEDED entries (= UUIDs of required libraries)
3. Load and verify each library (note: TAs do **not** share memory pages)
4. Process dynamic relocations of type R_ARM_GLOB_DAT / R_ARM_JUMP_SLOT (Armv7 and Armv8 AArch32) or R_AARCH64_GLOB_DAT / R_AARCH64_JUMP_SLOT (Armv8 Aarch64) by resolving symbols across binaries

The stack unwinding (crash dump) code has also been updated, as well as the `symbolize.py` tool.

TA Dynamic Link - Crash Dump

```
E/TC:? 0 Status of TA 5b9e0e40-2636-11e1-ad9e-0002a5d5c51b (0xe181f18) (active)
E/TC:? 0  arch: arm  load address: 0x105000 ctx-idr: 2
E/TC:? 0  stack: 0x102000 10240
E/TC:? 0  region 0: va 0x100000 pa 0xe100000 size 0x1000 flags ---R-X
E/TC:? 0  region 1: va 0x102000 pa 0xe41b000 size 0x3000 flags rw-RW-
E/TC:? 0  region 2: va 0x105000 pa 0xe300000 size 0x37000 flags r-x--- [0]
E/TC:? 0  region 3: va 0x13c000 pa 0xe337000 size 0xe4000 flags rw---- [0]
E/TC:? 0  region 4: va 0x220000 pa 0xe41e000 size 0x1000 flags r-x--- [1]
E/TC:? 0  region 5: va 0x230000 pa 0xe42e000 size 0x1000 flags rw---- [1]
E/TC:? 0  region 6: va 0x231000 pa 0xe42f000 size 0x1000 flags r-----
E/TC:? 0  [0] 5b9e0e40-2636-11e1-ad9e-0002a5d5c51b @ 0x105000
E/TC:? 0  [1] ffd2bded-ab7d-4988-95ee-e4962fff7154 @ 0x220000
E/TC:? 0 Call stack:
E/TC:? 0  0x00126934
...
```

EL0/EL1 permissions

File #0 is the main executable

Files #1, #2... are libraries

Load address

TA Dynamic Link - Crash Dump

```
$ ./optee_os/scripts/symbolize.py -d optee_test/out/ta/*
<paste crash dump then ^D>
...
E/TC:? 0  [0] 5b9e0e40-2636-11e1-ad9e-0002a5d5c51b @ 0x105000
                                (optee_test/out/ta/os_test/5b9e0e40-2636-11e1-ad9e-0002a5d5c51b.elf)
E/TC:? 0  [1] ffd2bded-ab7d-4988-95ee-e4962fff7154 @ 0x220000
                                                    (optee_test/out/ta/os_test_lib/libos_test.so)
E/TC:? 0 Call stack:
E/TC:? 0  0x00126934 utee_panic at optee_os/lib/libutee/arch/arm/utee_syscalls_a32.S:52
E/TC:? 0  0x0012e623 TEE_Panic at optee_os/lib/libutee/tee_api_panic.c:13
E/TC:? 0  0x00220317 os_test_shlib_panic at optee_test/ta/os_test_lib/os_test_lib.c:17
E/TC:? 0  0x00107cdb ta_entry_call_lib_panic at optee_test/ta/os_test/os_test.c:1095
E/TC:? 0  0x00107e41 TA_InvokeCommandEntryPoint at optee_test/ta/os_test/ta_entry.c:116
E/TC:? 0  0x001268a7 entry_invoke_command at optee_os/lib/libutee/arch/arm/user_ta_entry.c:191
E/TC:? 0  0x00126903 __utee_entry at optee_os/lib/libutee/arch/arm/user_ta_entry.c:219
```

where to find <UUID>.elf

U-Boot: OP-TEE driver

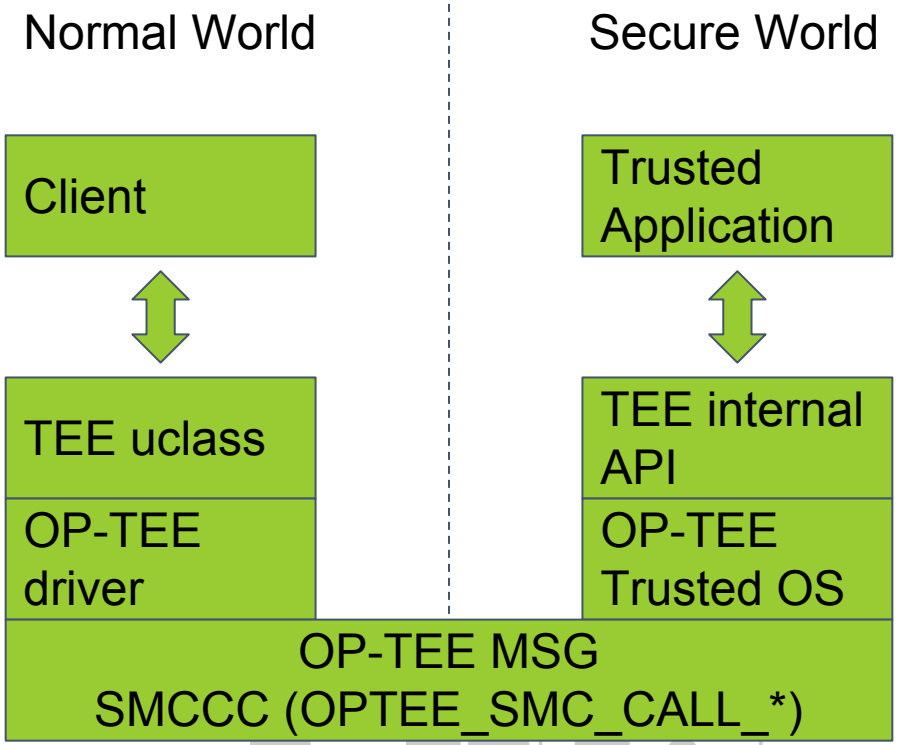
Jens Wiklander

Introduction

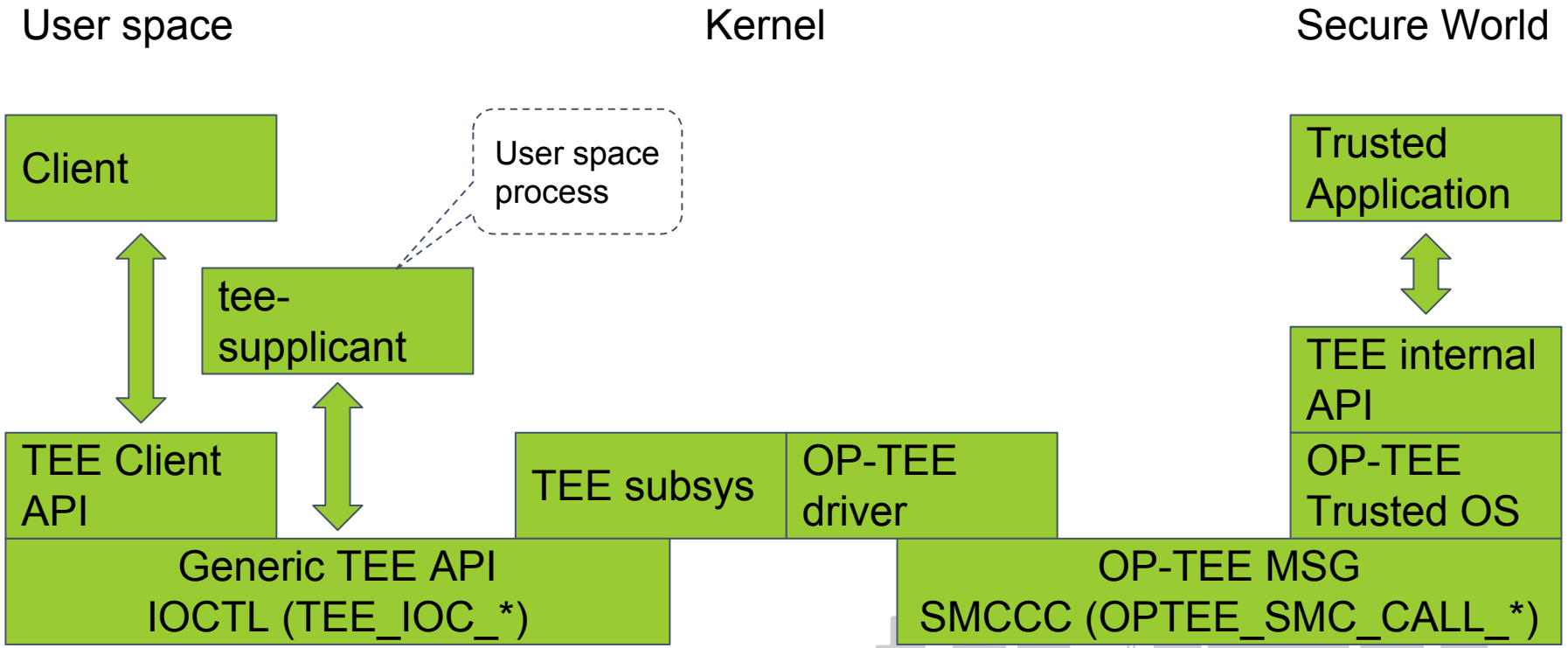
- The Linux kernel has an OP-TEE driver
- It has only been a matter of time before U-Boot would need the same
- The focus is OP-TEE based on ARM TrustZone
- Currently being upstreamed, latest posted version is v3

Software components

- Client
  - Privileged mode
  - Other drivers/code in U-Boot
- TEE uclass
  - Generic interface used by clients
- OP-TEE driver
  - Registers with the uclass
- Trusted OS
  - The TEE itself, running in secure world

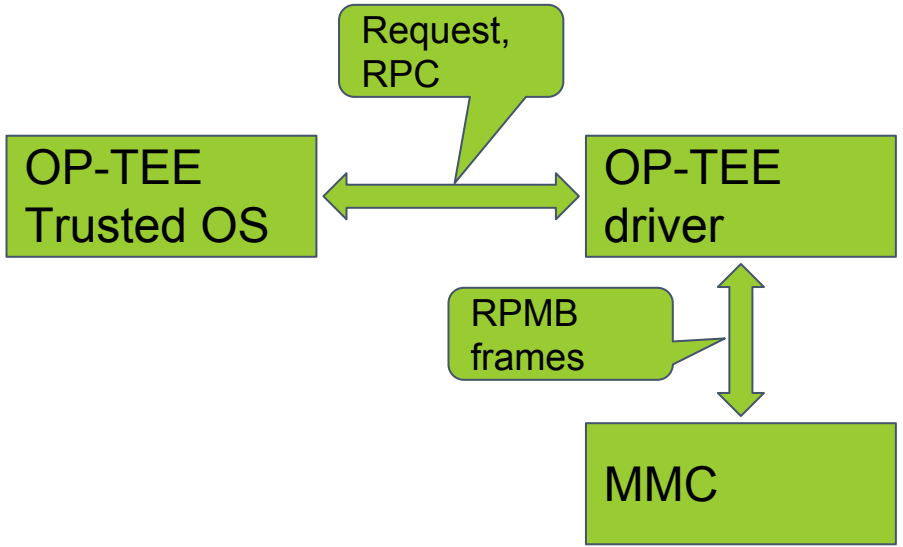


Comparison with Linux kernel



OP-TEE driver

- OP-TEE driver issues requests based client calls
- Some requests need to access RPMB
  - Remote Procedure Call (RPC) back with RPMB frames
- OP-TEE driver uses the infrastructure in U-Boot to interact with MMC



Client interface

This is an interface as capable as GlobalPlatform TEE Client API, but tailored for U-Boot.

The four most important functions are:

- tee_find_device() - to find the TEE device
- tee_open_session() - to open a session to a Trusted Application
- tee_invoke_func() - to invoke a function in a Trusted Application
- tee_close_session() - to close a session to a Trusted Application

U-Boot: Android Verified Boot 2.0

Igor Opaniuk

Introduction

- Android Verified Boot establishes a chain of trust from the bootloader to system image. Integrity checking of:
  - Boot: Linux kernel + ramdisk.
  - System/Vendor parts: verifying root hashes of dm-verity hash trees
- AVB 2.0 support was added to U-Boot and already upstreamed (included in U-Boot v2018.07 release), patchwork link
- HKG18 Presentation

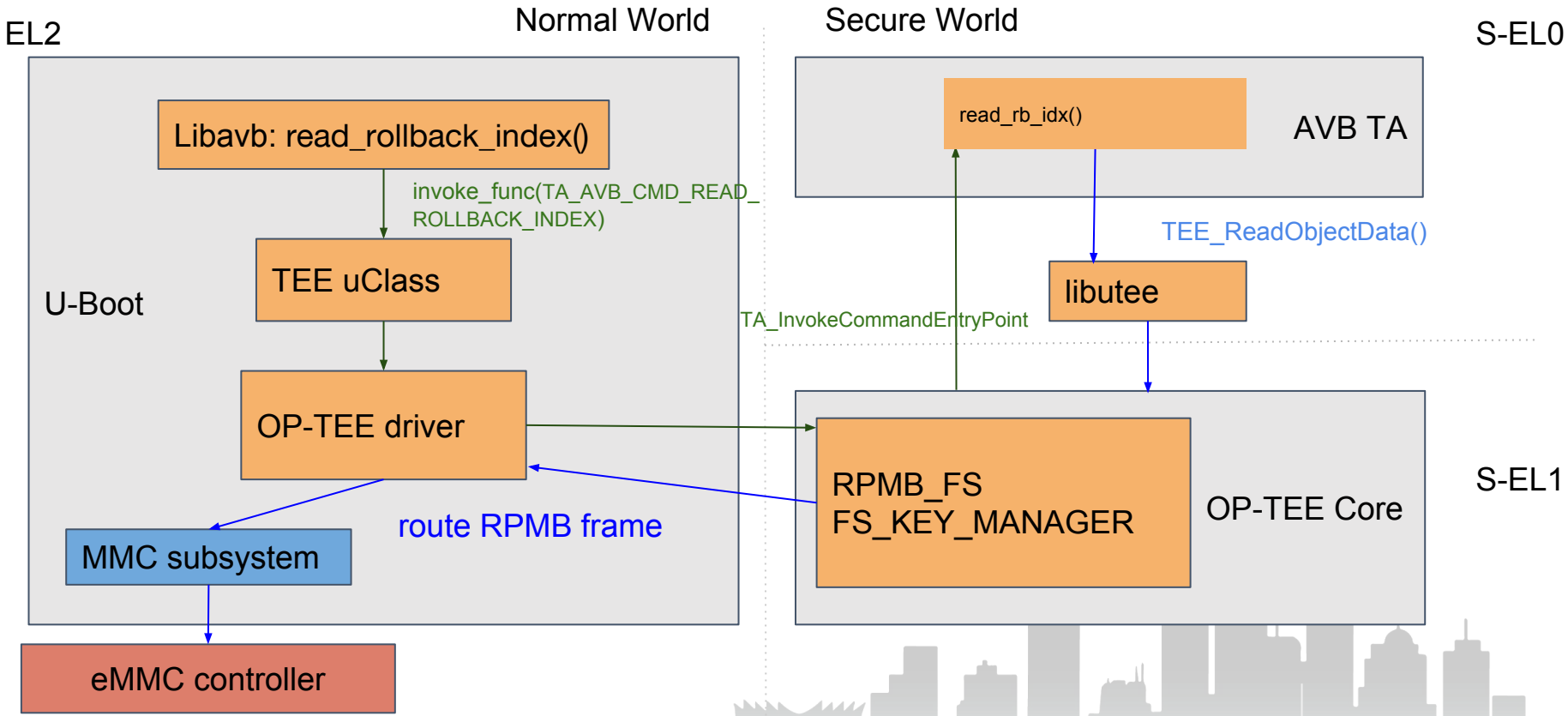
How to enable

- Kconfig symbols:
  - CONFIG_LIBAVB=y
  - CONFIG_AVB_VERIFY=y
  - CONFIG_CMD_AVB=y
- AVB verification process is invoked from U-Boot shell
  - avb init <mmc_id>
  - avb verify
- Additional details about integration on particular board: README.avb2

Tamper-evident storage

- AvbOps, that should work use tamper-evident storage:
  - validate_vbmeta_public_key()
  - read_rollback_index()
  - write_rollback_index()
  - read_is_device_unlocked()
  - write_persistent_value()
  - read_persistent_value()
- Leverage U-Boot OP-TEE driver introduced by Jens Wiklander

Tamper-evident storage



Next steps

- Implementation/refactoring of AvbOps:
  - validate_vbmeta_public_key()
  - write_persistent_value()
  - read_persistent_value()
- RPMB device ID in U-Boot vs Linux sync:
  - eMMC Device ID in U-Boot and Linux can sometime get out of sync, this has happened on the Poplar board
- A/B support
- Fastboot integration:
  - OEM command subset: oem lock/unlock etc.

OP-TEE on HiKey620 AOSP

Victor Chong

OP-TEE on HiKey620 AOSP

- Session about OP-TEE AOSP integration in Hong Kong
- Improvements
  - Daily CI build
  - Upgraded to P - fully Treble-ized
  - Added build scripts to simplify building

How to try it?

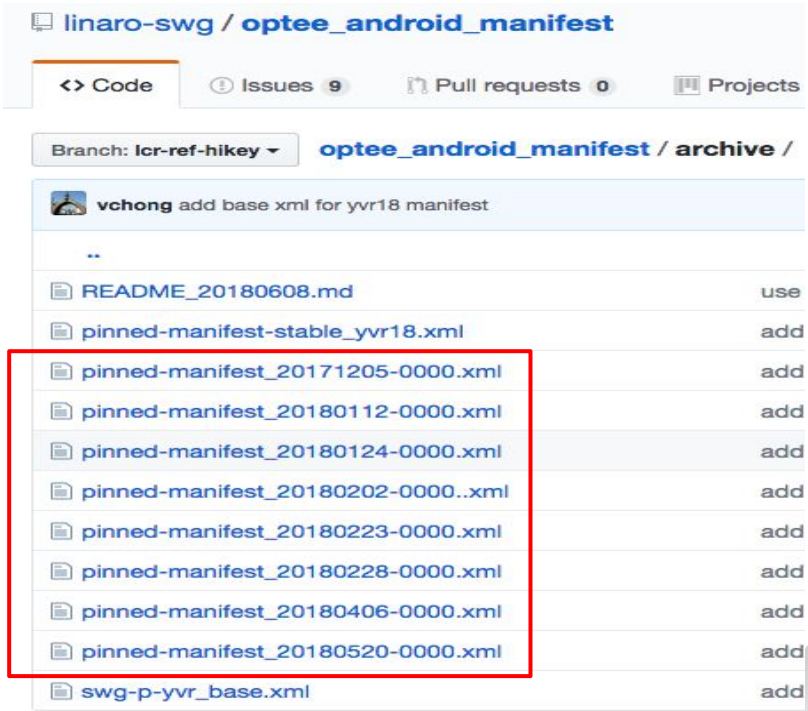
- Build

```
$ git clone https://github.com/linaro-swg/optee_android_manifest -b lcr_ref_hikey
$ cd optee_android_manifest
$ ./sync-p.sh
$ ./build-p.sh

# alternatively for relatively stable builds
$ ./sync.sh -v p -bm <name of a pinned manifest file in archive/>
# e.g.
$ ./sync.sh -v p -bm pinned-manifest_20180520-0000.xml
```

How to try it?

- Pinned manifest @ https://github.com/linaro-swg/optee_android_manifest/tree/lcr-ref-hikey/archive



How to try it?

- Flash (HiKey 6220)
  - Put board in recovery mode by connecting jumpers 1-2 and 3-4
  - Power on board
  - Run command

    $ cp -a out/target/product/hikey/*.img device/linaro/hikey/installer/hikey/

    $ sudo ./device/linaro/hikey/installer/hikey/flash-all.sh /dev/ttyUSB<x>

    x = device number that appears after rebooting with the 3-4 jumper connected

    e.g.

    $ sudo ./device/linaro/hikey/installer/hikey/flash-all.sh /dev/ttyUSB0
  - Power off board
  - Remove jumper 3-4
  - Power on board

How to try it?

- Test

```
$ adb root
$ adb shell xtest
```

PKCS#11 with OP-TEE

Etienne Carriere

PKCS#11 with OP-TEE

- Session about it in Hong Kong
- Refactored the TA interface
- AES operations,
  RSA (signature gen+ver)
  EC (ECDH) key generation
- Open pull requests
  - optee_client: Pkcs#11 services through OP-TEE SKS TA
  - optee_os: Secure key services: step1: TA basics and API
  - optee_test: xtest: regression 41xx target Secure Key Services tests
  - build: buildroot: optee_os services built as buildroot external package
- Main development branch for SKS TA
  - https://github.com/etienne-lms/optee_os/tree/sks/ta_services/secure_key_services

Misc

Misc other work

- FOSSology + ScanCode + SPDX
- Buildroot for OP-TEE developer setup
- mbed TLS for TA's
- Spectre & Meltdown: Improved branch predictor invalidation (ICIALL not recommended on Arm64)
- Automatic testing of GitHub pull requests using IBART
- Stepping up default gcc version (from 6.x to 8.x)
- Added support for Poplar at build.git
- RPi3 setup - updated and reworked (now uses a "proper" boot)
- Last but not least, answered tons of questions at GitHub