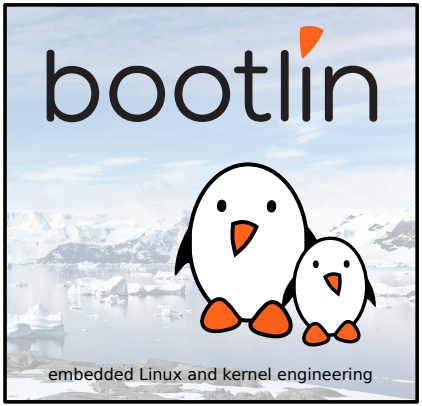
OP-TEE: When Linux Loses Control

Clément Léger
clement.leger@bootlin.com



Clément Léger

- Embedded Linux engineer at Bootlin
  - Embedded Linux **expertise**
  - Low-level development for Linux kernel, Bootloader, etc
  - **Development**, consulting and training
  - Strong open-source focus
- Ported Linux to a custom VLIW architecture
- Living in **Grenoble**, near the Alps in France

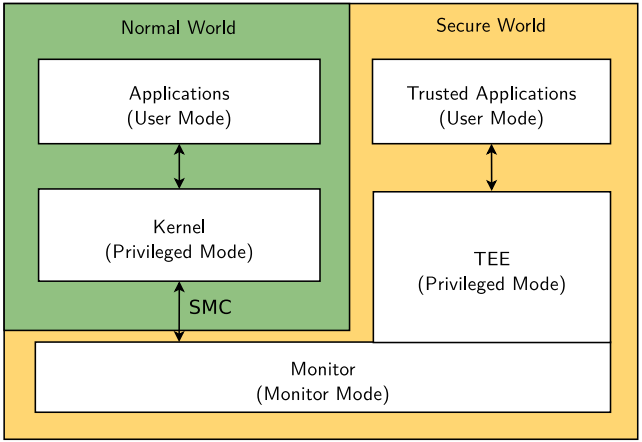
About this talk

- Explain what is OP-TEE
- Explain OP-TEE integration into an existing system which runs everything in secure mode
- Describe interactions between various components (SPL/OP-TEE/Bootloader/Linux)
- Present various protocols & methods to communicate between REE/TEE (SCMI, PSCI, SMC)
- Provide experience feedback on Microchip SAMA5D2 OP-TEE port (ARMv7-A)

What is OP-TEE ?

- **O**pen **P**ortable **T**rusted **E**xecution **E**nvironment
- Provides secure services to a Rich Execution Environment (**REE**) (Typically Linux).
- Based on hardware-enforced isolation technology (ARM TrustZone)
- Secure Monitor Calls (**SMC** instruction) allow to switch REE $\leftrightarrow$ TEE via the secure monitor
  - Much like a SVC but jump to a higher privileged mode (Monitor).

TrustZone

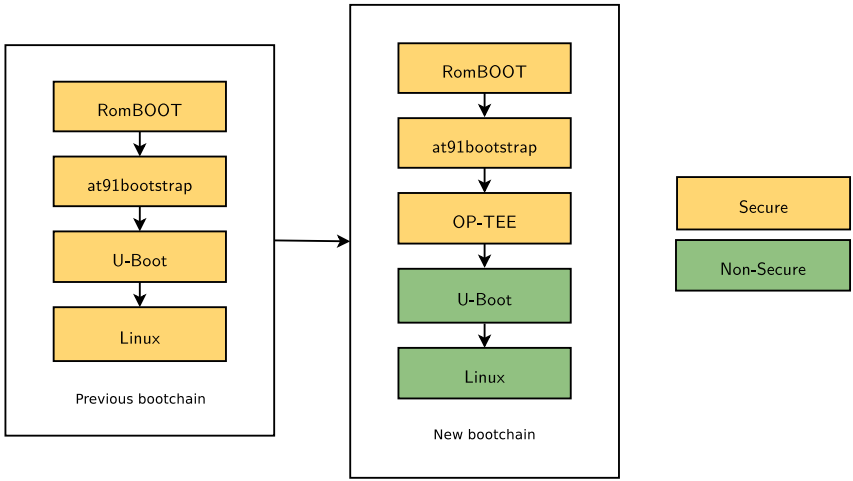

Normal World
Secure World
Applications
(User Mode)
Trusted Applications
(User Mode)
Kernel
(Privileged Mode)
TEE
(Privileged Mode)
SMC
Monitor
(Monitor Mode)

Bootchain

Bootchain on ARMv7-A

- TF-A on ARMv7-A is supported on some SoC but bootchain is often different between them
- On SAMA5D2, there is no TF-A support, at91bootstrap is the "SPL" and loads the next software:
  - Can be U-Boot, Linux or another binary
- At some point in the bootchain, a switch to the normal world happens
- Our goal is to minimize the amount of code running as secure
  - Only ROMBoot, at91bootstrap and OP-TEE will run in secure mode

Bootchain on SAMA5D2



Booting OP-TEE (1)

- OP-TEE outputs a binary with a simple header (`tee.bin`, V1 header):

```
struct optee_header {
  uint32_t magic;
  uint8_t version;
  uint8_t arch;
  uint16_t flags;
  uint32_t init_size;
  uint32_t init_load_addr_hi;
  uint32_t init_load_addr_lo;
  uint32_t init_mem_usage;
  uint32_t paged_size;
};
```

- Binary should be copied at `init_load_addr` without header
- Another multi part header (V2) exists for loader that support separate binaries

Booting OP-TEE (2)

- Need to pass the following registers when jumping to OP-TEE:
  - **r0**: Page store area pointer (When using `CFG_WITH_PAGER`)
  - **r1**: Normal world first argument
  - **r2**: Normal world second argument (External device tree for OP-TEE)
  - **lr**: Normal world entry point (Can be hardcoded using `CFG_NS_ENTRY_ADDR`)
- After starting it, OP-TEE is *persistent* in memory an runs in secure world
  - Vector table is set for monitor mode (`MVBAR`) which allow catching SMC
- When returning to normal world (address in `lr`), `r0` will be set to 0
- at91bootstrap has been modified to support OP-TEE load
  1. Loads both OP-TEE and U-Boot from non volatile memory into RAM
  2. Set `lr` to U-Boot load address
  3. (When booting linux) Set `r1` to MACH type and `r2` with device tree
  4. Jump to OP-TEE entry point (`init_load_addr`)

OP-TEE Device Trees (1)

- OP-TEE manipulates 2 devices trees:
  - Embedded: Secure device tree for OP-TEE internal use (`CFG_EMBED_DT` selected by `CFG_EMBED_DTB_SOURCE_FILE`)
  - External: Non-Secure device tree meant for Normal World (Passed via `r2` or hardcoded using `CFG_DT_ADDR`)
- OP-TEE only works with flattened device tree
- External device tree can be modified or generated as a device tree overlay which can be merged by Normal World
  - `CFG_EXTERNAL_DTB_OVERLAY` will reuse an external device tree overlay or create it if needed
  - `CFG_GENERATE_DTB_OVERLAY` will always create a new device tree overlay

OP-TEE Device Trees (2)

- Device tree property `secure-status` allows to set the status of the node for secure world

```
/* Device should be set as accessible by both Secure & Non-Secure */
secure-status = "okay";
status = "okay";
```

```
/* Device should be set as accessible by Secure only */
secure-status = "okay";
status = "disabled";
```

- Nodes status in a device tree can be modified using `dt_enable_secure_status()`
  - Will set `status = "disabled"` and `secure-status = "okay"` for the node
  - Useful to change the state of a node after handling it in secure world

OP-TEE Device Trees (3)

- Some nodes can be generated by OP-TEE to pass information to Normal World via external device tree
  - `optee` node for Linux OP-TEE driver
  - `psci` node for PSCI informations
- Other changes to change a system state from Secure to Non-Secure
  - SCMI node is not generated and some modifications are needed for SCMI clocks
  - `assigned-clock-parent` handling fails due to missing re-parenting support in SCMI protocol

Securing peripherals

Why securing ?

- When running under a TEE, the system must remain secure
  - The REE MUST not be able to alter the TEE or system state
- Memories containing the TEE data must also remain out of reach of REE
- Need to reduce Linux critical peripheral access
- Any peripheral that can compromise the system state or the TEE integrity should be secure
  - Reset/shutdown, watchdog controllers → For obvious reason !
  - CPU Online/Offline control → Obvious reason
  - Clock controllers → Can disable clock of critical devices
  - Random number generator → Could leak random data used for secure purpose
  - Cryptographic engines → Can access secret asset
  - Bus controllers → Can change bus settings (QoS, etc)
  - Timers → Can alter the perception of time for TEE
  - OTP, Fuses → Can access sensitive data (secure assets, key, etc)

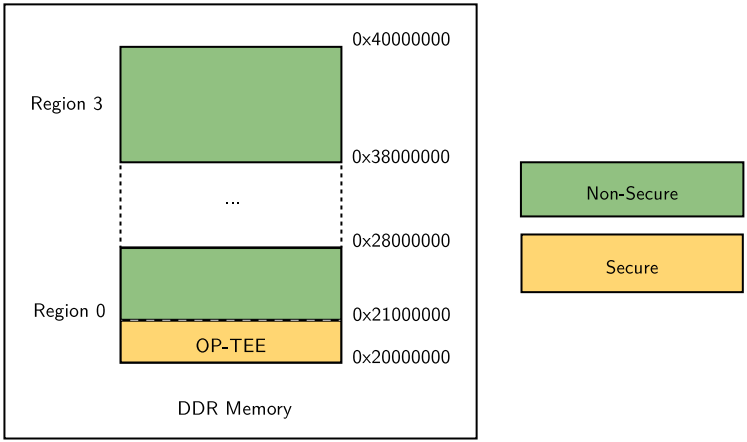
Hardware for security

- Depending on hardware, bus controllers often provide security settings if ARM TrustZone is supported.
  - An additional security bit is conveyed with accesses over the bus
  - Set peripherals access for secure masters only
  - Set memories secure layout by splitting them with more or less flexibility
- On SAMA5D2, bus controllers (Bus matrix) allow to secure devices and memories:
  - Memories can be split according to *regions*
  - Peripherals state can set as Secure or Non-secure

Securing memories

- TEE memories **MUST** be set as secure (text, data, etc)
- Memories (SRAM/DRAM/etc) can be split in 2 areas for each port according to predefined sizes
  - Either the first part is secure and the second part is non-secure or the reverse
- When setting a secure memory layout and using DDR, consider all the available boards !
  - Some boards have less DDR than others, so try to put OP-TEE in the beginning of the DDR rather than at the end
- SAMA5D2 initial memory map for OP-TEE was at 256MiB (`0x30000000`)
  - Memory map modified to put OP-TEE at start of DDR (`0x20000000`)

Secure DDR on SAMA5D2



Securing peripherals

- Security of peripherals is often a combination of peripheral and bus controller settings
  - Some registers have different secure behavior for read vs write
- SAMA5D2 Matrix buses can set peripheral security world.
  - Peripherals are either "Always Secure" or "Peripheral Securable" (can be configured as secure or non-secure)
  - Peripherals might have additional write protection features for safety
  - Interrupt are also affected by these settings and will target the secure interrupt controller
  - Some devices include functions that are used by both world (Special Function Register for instance)

OP-TEE peripheral security

- OP-TEE platform code is in charge of setting up hardware security
- There is no common framework to handle this, each platform implement it its own way
- Can be done using `secure-status`/`status` property in device tree with `_fdt_get_status` which returns a mask:
  - `DT_STATUS_OK_SEC`: `secure-status` is set to `"okay"`
  - `DT_STATUS_OK_NSEC`: `status` is set to `"okay"`
  - `DT_STATUS_DISABLED`: node is completely disabled
- Based on this information, peripherals can be set as secure with bus controllers

Communication

Doorbell

- Doorbell mechanism needed to send notifications from REE to TEE
- Hardware mailboxes can be used if available
- SMC instruction allows to switch from Normal World to Secure World via the Secure Monitor by generating a synchronous exception
  - SMC Calling Convention define ranges for function IDs and argument passing
  - Registers `r0` contains the function ID to call and `r1 - r7` are used for arguments
  - Registers `r0 - r7` are used to return values
- Allows to call specific OP-TEE services handled by `sm_platform_handler`
- SAMA5D2 does not have mailboxes → SMC are used for notifications

Protocols

- Some protocols allows to control the critical peripherals into Secure World
- **S**ystem **C**ontrol and **M**anagement **I**nterface allows to manage clocks, power and reset domain
  - SMCs can be used as the doorbell mechanism for SCMI.
- **P**ower **S**tate and **C**oordination **I**nterface allows to control power states of system and CPUs
- OP-TEE driver for specific communication via OP-TEE Message Protocol
  - Communication with Trusted Application (TA/Pseudo TA) from Non-Secure client applications
  - Handles RPC from OP-TEE (Services offered by Non-Secure World, I2C, REE timer for instance)

SCMI

- Set of standard interfaces for power, performance and system management
- OP-TEE integrates a SCMI server which can handle clocks, power domains & reset domains.
- Linux provides support for SCMI (with `CONFIG_ARM_SCMI_PROTOCOL`).
- Communication can be done using SMCs or mailboxes
  - At the time of this talk, SCMI driver needs `CONFIG_ARM_SCMI_PROTOCOL` which needs `CONFIG_MAILBOX`
  - SCMI support via SMC needs `CONFIG_HAVE_ARM_SMCCC_DISCOVERY` to be enabled which needs `CONFIG_ARM_PSCI_FW` to detect the SMC version supported.
  - Basic PSCI support is needed in OP-TEE platform to use SCMI driver with SMC.

SCMI Device tree (1)

- `arm,smc-id` defines the function ID to be used with SMCs to call OP-TEE platform specific SCMI handling
  - On OP-TEE side, this function ID must be handled in `sm_platform_handler()` to call SCMI channel handling (`scmi_smt_fastcall_smc_entry`)
- A shared memory area (Secure/Non-Secure) must be defined in the device tree for SCMI data exchange
- Supported protocols are defined as `scmi0` subnodes where `reg` describes the protocol identifier
- U-Boot also provides support for SCMI clocks and uses the same device tree bindings than Linux
  - If clock are needed early in U-Boot, `u-boot,dm-pre-reloc` property should be added to SCMI nodes for pre relocation probing

SCMI Device tree (2)

```
reserved-memory {
  scmi0_shmem: scmi0_shmem@21400000 {
    no-map;
    reg = <0x21400000 0x80>;
  };
};

firmware {
  scmi0 {
    compatible = "arm,scmi-smc";
    shmem = <&scmi0_shmem>;
    arm,smc-id = <0x2000200>;

    scmi0_clock: scmi0_clock@14 {
      #clock-cells = <0x01>;
      reg = <0x14>;
    };
  };
};
```

SCMI Clocks

- SCMI protocol for clocks allows to query clock count, enable/disable and set/get the rate for clocks
- `CONFIG_SCMI_CLK` in Linux
  - SCMI clocks are probed later that **clocksources** !
  - This can be a problem if you have a core **clocksource** needing an SCMI clock to work
- `CONFIG_CLK_SCMI` in U-Boot
- SCMI identifies clocks using a single integer
- Clocks identifier are expected to be contiguous and in the range of `[0 - clock_count[`

SCMI Clocks

- Existing device tree should be modified to use SCMI clocks instead of physical ones (This can be a tedious task when using a lot of clocks !)
- For instance, this clock description:

```
&tdes {
  clocks = <&pmc PMC_TYPE_PERIPHERAL 11>;
};
```

- Will become:

```
&tdes {
  clocks = <&scmi0_clock AT91_SCMI_CLK_PERIPH_TDES_CLK>;
};
```

Clocks in OP-TEE

- Previously no framework to handle a clock tree (custom to each platform)
- Pull-Request ongoing to add a basic clock framework
  - Adds device tree parsing to query clocks from drivers
  - Use existing device tree bindings to assign clock parents and rates
- SCMI generic clock support will be added and use this generic clock framework
  - Proposal is to use device tree bindings to match physical clocks to SCMI clocks

```
tdes_clk@19 {
  reg = <AT91_SCMI_CLK_PERIPH_TDES_CLK>; /* SCMI identifier */
  clocks = <&pmc PMC_TYPE_PERIPHERAL 11>; /* Physical clock */
};
```

PSCI

- Standard interface for power management
  - System shutdown and reset
  - Core idle management
  - Dynamic addition and removal of cores, and secondary core boot
- PSCI support can be enabled in Linux with `CONFIG_ARM_PSCI_FW`
- In U-Boot, PSCI reset and shutdown are supported with `CONFIG_SYSRESET_PSCI`
- In OP-TEE platform specific code must be added (weak functions) to handle PSCI request and report supported features

PSCI Device tree

- PSCI node can be generated by OP-TEE in the external device tree
  - Allows to always have the correct PSCI SMC function ID

```
psci {
  sys_reset = <0x84000009>;
  sys_poweroff = <0x84000008>;
  cpu_on = <0x84000003>;
  cpu_off = <0x84000002>;
  cpu_suspend = <0x84000001>;
  method = "smc";
  compatible = "arm,psci-1.0", "arm,psci-0.2", "arm,psci";
};
```

PSCI Suspend, Reset, Shutdown

- PSCI provides a suspend method to enter system in suspend mode
- Semantic for suspend is to enter the deepest suspend mode available
- On SAMA5D2, there are multiple suspend modes which are selectable by a command line option (`atmel.pm_modes`)
  - Solution is to use a custom Silicon Provider SMC (SIP SMC) to set suspend mode at boot time
  - SMC is done according to the existing boot parameter `atmel.pm_modes`

PM Support in OP-TEE

- OP-TEE provides arch support for suspend support
- PM functions allows to register suspend/resume callback that will be called on PM state changes
  - `register_pm_driver_cb()` and `register_pm_core_service_cb()`
- platform code should call `pm_change_state()` to change PM state

```
pm_change_state(PM_OP_SUSPEND, 0);
```

CPU Idle

- CPU Idle via PSCI is supported in Linux when enabling `CONFIG_ARM_PSCI_CPUIDLE`
- PSCI idle states must be described in device tree
- `entry-method` property must be set to `"psci"`
- Multiple states can be describe and will be chosen according to their usage "cost"
- `arm,psci-suspend-param` value will be passed with PSCI call to allow TEE to identify the requested idle mode

Idle states

```
cpu@0 {
  enable-method = "psci";
  cpu-idle-states = <&psci_standby>;
};

idle-states {
  entry-method = "psci";

  psci_standby: psci-standby {
    compatible = "arm,idle-state";
    idle-state-name = "psci,standby";
    arm,psci-suspend-param = <0x0>;
    entry-latency-us = <1000>;
    exit-latency-us = <700>;
    min-residency-us = <2000>;
  };
};
```

OP-TEE driver

- An OP-TEE driver is present in Linux
- This driver is probed based on device tree and OP-TEE can also generate this node in the external device tree overlay

```
reserved-memory {
  optee_core@20000000 {
    reg = <0x20000000 0x1000000>;
  };

  optee_shm@21000000 {
    reg = <0x21000000 0x400000>;
  };
};

firmware {
  optee {
    compatible = "linaro,optee-tz";
    method = "smc";
  };
};
```

OP-TEE architecture

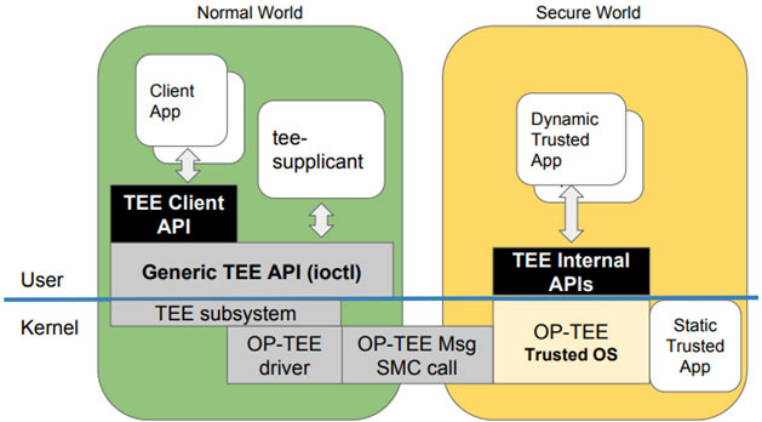


Figure: from Linaro

Other Peripherals

True Random Number Generator

- Linux driver for OP-TEE RNG must be enabled using `CONFIG_HW_RANDOM_OPTEE`
- On OP-TEE side, a Pseudo Trusted Application (**PTA**) provides access to the secure RNG when `CFG_HWRNG_PTA` is enabled
  - Linux driver will then communicate with this PTA to query random data
- If correctly detected by Linux driver, `optee-rng` hwrng will be available and selected as current hwrng:

```
# cat /sys/class/misc/hw_random/rng_available
optee-rng
# cat /sys/class/misc/hw_random/rng_current
optee-rng
```

- Secure random data are then available using `/dev/hwrng`

Timer

- OP-TEE provide a timer based on *ARM Generic Timer* extension and one using REE time
  - A secure timer should be added if SoC does not support *ARM Generic Timer* or to have a more secure timer
- OP-TEE provides an interface to register a secure timer

```
static const struct time_source atmel_tcb_time_source = {
  .name = "atmel_tcb",
  .protection_level = 1000,
  .get_sys_time = atmel_tcb_get_sys_time,
};

REGISTER_TIME_SOURCE(atmel_tcb_time_source)
```

- Counter should return relative time that has elapsed since a fixed point
  - 64bits counter is recommended to avoid wrapping
- `tee_time_get_sys_time()` can then be used to query the time using this clock

PL310 L2 cache

- PL310 cache setup is supported by OP-TEE
- If configuration is expected to be modified by Linux, `.write_sec` function should be set
  - Function defined in platform code and should override `outer_cache::write_sec()`
  - Often uses SMCs with custom function ID but no standard for that
  - Secure/Non-Secure state must be discovered by platform custom code (Device Tree parsing)

Interrupts

- Currently only GIC is supported by OP-TEE
  - SAMA5D2 Support for Secure Advanced Interrupt Controller will be added
- Basic interrupt framework allows to enable/disable an interrupt
  - Pull-Request ongoing for interrupt specifier (level, priority) support
- `itr_core_handler()` has to be redefined by platform code to handle irqs
- Interrupts used with the bus controller capabilities are **really** useful to find were the security violations are happening.
  - Can display the address of violation and the location where it happened:

```
Matrix 0 permission failure from master 0, address 0x200118b8, mon_lr = 0x3ffa4a5c
```

  - `mon_lr` points to the address where the violation was generated (read/write)

Contribution status

- OP-TEE:
  - Generic clock framework to handle clock tree (In progress: 4705#)
  - SCMI clock support based on device tree (TODO)
  - SAMA5D2 enhanced secure support (Clocks, SCMI, suspend, reset, shutdown, interrupts, TRNG, Timers) (TODO)
- at91bootstrap:
  - Support for OP-TEE loading (In progress: 128#)
- Linux:
  - SAMA5D2 PL310 L2 cache `write_sec()` support (TODO)
  - SAMA5D2 Secure suspend support (SMC to select) (TODO)

Questions? Suggestions? Comments?

Clément Léger
clement.leger@bootlin.com

https://bootlin.com/pub/conferences/2021

Going further and sources

- OP-TEE Documentation : https://optee.readthedocs.io/en/latest/
- SCMI specification: https://developer.arm.com/architectures/system-architectures/software-standards/scmi
- PSCI specification: https://developer.arm.com/architectures/system-architectures/software-standards/psci