LINUX SECURITY SUMMIT EUROPE

Using the ARM Trust-Zone to control Tamper Resistant Processors

U-boot/Linux Cryptographic Support

Jorge Ramirez-Ortiz
Device Security Lead

@Foundries.io

THE LINUX FOUNDATION

AGENDA

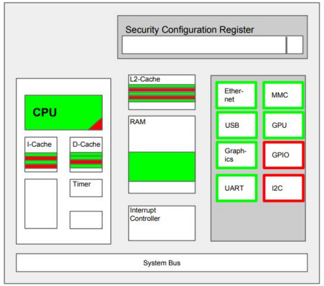
1. ARM TrustZone
2. Runtime Resident Firmware
3. OP-TEE: Root of Trust
4. OP-TEE: Secure Persistent Storage
5. OP-TEE: Trusted Application
6. OP-TEE: PKCS#11 Trusted Application
7. OP-TEE: Executing in the TrustZone
8. OP-TEE: Trusted Thread Exit
9. Yielding SMC: Working in the TrustZone
10. Use Case: NXP SE050X HSM (I2C)
11. Use Case: Versal ACAP Crypto
12. References

Maintainership in OP-TEE:
- Versal ACAP and drivers
- ZynqMP drivers
- SE05X crypto driver
- iMX I2C driver
- iMX RNGB driver

jorge@foundries.io

ARM TrustZone

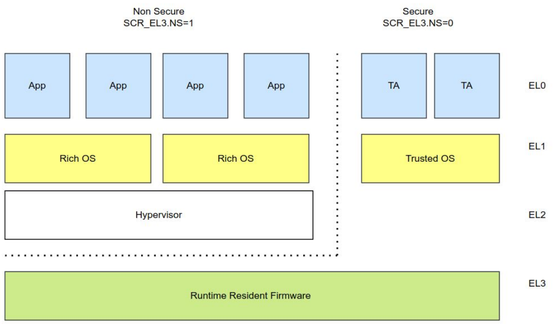
- Hardware feature on Cortex-A and since recently Cortex-M profiles
- Hardware enforced isolation built into the CPU
  - Additional Processor Mode (CPR15 NS bit)
  - MMU mode awareness (cache/TLB entries can coexist)
  - New Exception Level (EL3) to handle mode transition
    - Runtime Resident Firmware
      - Runtime Resident Firmware Services (std_svc or spd)
  - Vendor Specific Peripheral partitioning (i.e GPIO)
- TrustZone must be handled with care
  - ARM Trusted Firmware project
- QEMU emulation has been supported for <u>nearly 10 </u>years now



Runtime Resident Firmware

- Runtime Resident Firmware - executes at EL3
- Should be installed under the Root of Trust
- Entered via the SMC privileged call
  - smccc - follows calling convention
    - different calls:
      - Fast: Atomic - ARM, CPU, SiP(silicon), OEM ..
      - **Yielding**: Can be suspended.

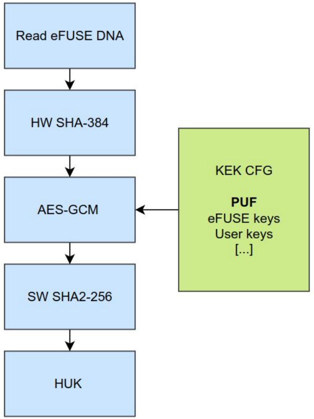
- OP-TEE Runtime Resident Firmware
- TrustedFirmware.org Runtime Resident Firmware
    - Provides Runtime Services:
      - Secure Payload Dispatcher
        - » switches between normal and secure world
        - » handles SMC functions targeting S-EL1
        - » manages S-EL1 context



- BL1 ROM
- BL2 EL3 RAM FIRMWARE (i.e: SPL)
- BL31 EL3 SECURE MONITOR
- BL31 EL1 SECURE PAYLOAD (ie:OP-TEE)
- BL32 EL2 BOOTLOADER (i.e: U-boot)

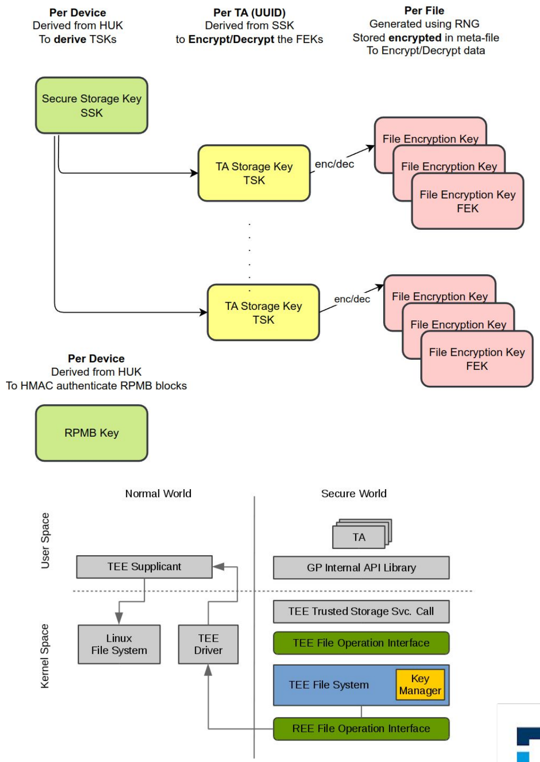
OP-TEE - Root of Trust

- **Hardware Unique Key** 128 bits
  - per device identifier
  - symmetric key used to derive (KDF) all other keys
    - secure storage keys
    - securing the linux keyring: sealing keys
    - external HSM protocol encryption keys (NXP SE05x)
  - must not be readable outside the TrustZone
    - risks secure storage duplication and its decryption
- Could be
  - Provided by SoC (iMX CAAM)
  - Generated by OP-TEE driver (AMD Versal ACAP)
    - uid + AES-GCM
- Serves as a device unique identifier
- Must only be available when booting signed firmware



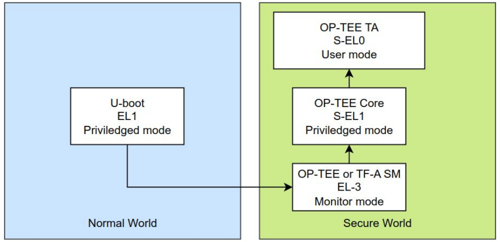
OP-TEE - Secure Persistent Storage

- Global Platform Trusted Storage Requirements
  - May be backed by NS resources
    - OP-TEE: REE-FS :
      - FEK **AES-GCM** auth enc/dec data
  - eMMC RPMB support
    - OP-TEE: RPMB-FS:
      - FEK **AES-CBC** enc/dec data
      - RPMB key: HMAC RPMB blocks
  - Bound to a device
  - Each TA can only access its own storage
  - Must protect against replay attacks

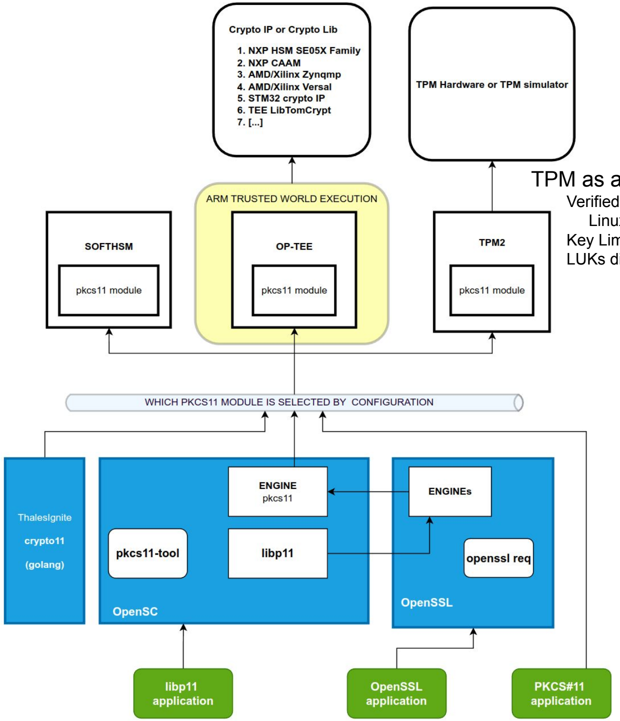


OP-TEE - Trusted Application

- The execution is split by the NS bit (TA is NS=0)
- TA are RSA signed
  - asymmetric public key built in the TEE core
- TA might be AES-GCM encrypted
  - symmetric key generation is platform dependent
- TA can be
  - stored in the REE filesystem (TA)
  - built-in the TEE image (Early TA)
- Upon load they are verified and decrypted
- TA are versioned (ta_ver.db) to prevent rollback attacks
  - However only effective when using RPMB_FS
- Built time TA configuration flags
  - multisession, keep alive, stack size, heap size …



OP-TEE PKCS#11 Trusted Application

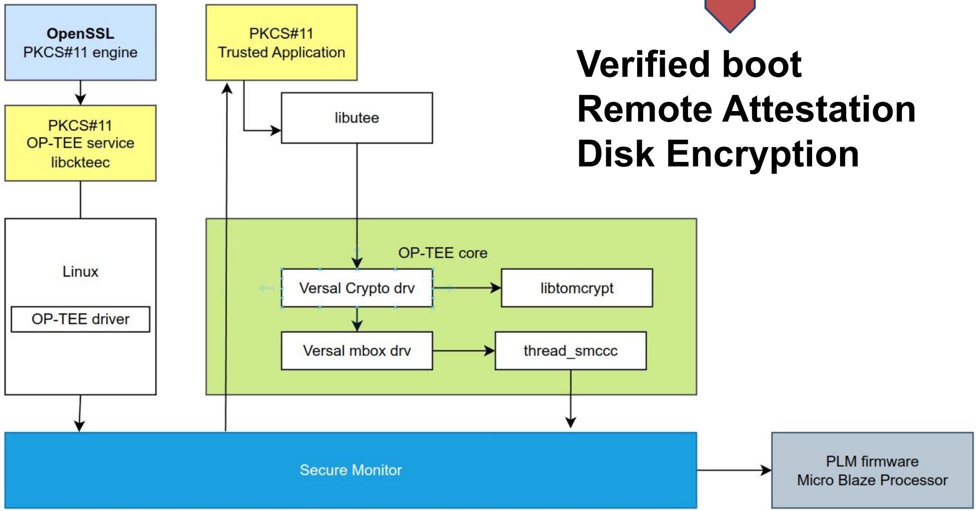


TPM as a PKCS#11 token
- Verified Boot
  - Linux Integrity Measure Architecture
- Key Lime Remote Attestation
- LUKs disk encryption

PKCS#11:

A generic secure interface for all things crypto

Key management, Crypto operations, Storage



Verified boot

Remote Attestation

Disk Encryption

AGENDA

1. ARM TrustZone
2. Runtime Resident Firmware
3. OP-TEE: Root of Trust
4. OP-TEE: Secure Persistent Storage
5. OP-TEE: Trusted Application
6. OP-TEE: PKCS#11 Trusted Application
7. OP-TEE: Executing in the TrustZone
8. OP-TEE: Trusted Thread Exit
9. Yielding SMC: Working in the TrustZone
10. Use Case: NXP SE050X HSM (I2C)
11. Use Case: Versal ACAP Crypto
12. References

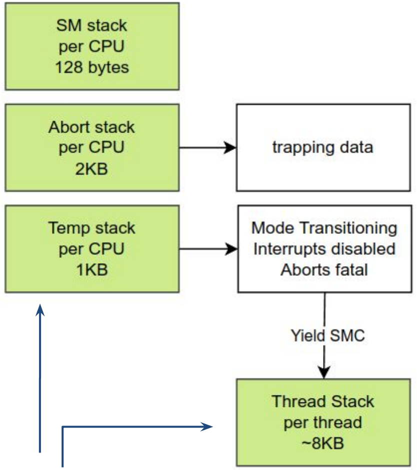
THE LINUX FOUNDATION

OP-TEE: Executing in the Trust Zone

- OP-TEE (optionally) implements an EL3 Runtime Resident Firmware (SM)
- A Secure Monitor manages all entries and exits of the secure world via SMC
- SMC
  - Fast: IRQ masked
  - Yielding: IRQ unmasked at some point
    - S-EL1 context can be suspended and resumed
    - TT scheduling controlled by REE

- OP-TEE does NOT implement a scheduler
  - Trusted Threads (shadow) are tied to the execution of the calling thread
    - All requests are always REE initiated
  - Number of TT is statically allocated

- OP-TEE supports multiple threads of execution
  - OP-TEE implements synchronization primitives
    - Native: spinlock - do not take with interrupts enabled
    - REE: mutex and condition variables
      - Do not take in interrupt context, do not leave the TA holding a mutex
      - In REE Linux it uses the Linux *completion* structures



Size configurable at build time
(i.e: NXP SE05x requires 16KB)

OP-TEE: Trusted Thread exit

- Every execution request initiates in the Normal World
- <u>Yielding SMC</u> calls context can be <u>restored</u> → Sync/Async TT exits
- Asynchronous Exits
  - Foreign interrupts will suspend TT and return to NS
- Synchronous Exits
  - Callback the monitor software at EL3
    - thread_smccc
  - Callback the REE at EL1via monitor software at EL3
    - thread_rpc(<u>OPTEE_RPC_CMD_</u> …)
    - the REE can handle the request at EL1
      - U-boot can access RPMB
      - Linux a EL-1 can provide I2C access to the TEE
    - or notify an EL0 daemon to handle the request
      - the EL0 daemon is known as the *tee_supplicant*
        - » *tee_supplicant* handles RPMB requests in Linux

Normal Exit - tee_entry_std() returns

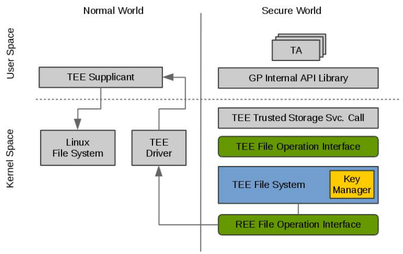
Foreign Interrupt Exit
Interrupt targeting Normal World
Interrupt targeting Secure Monitor

Thread RPC Exit

Thread SMC Exit

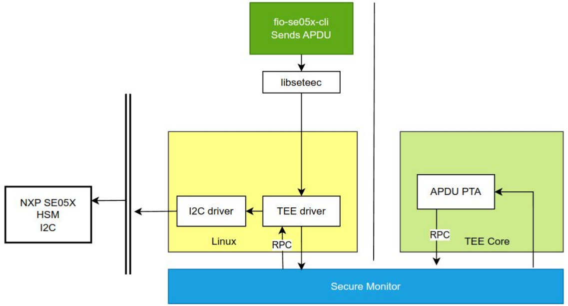
Yielding SMC: Working in the TrustZone

- EL0: User Linux, TEE supplicant



Example: Write to eMMC RPMB

- EL0: User Linux, NO TEE supplicant

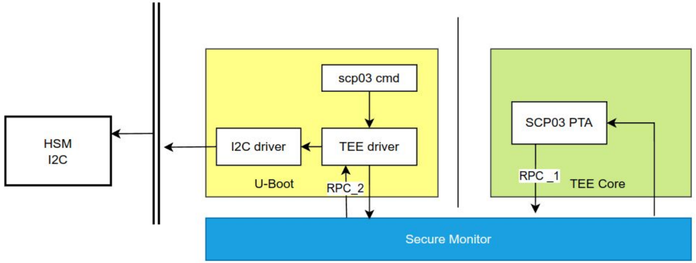


Example: Access to Secure Element

https://github.com/foundriesio/fio-se05x-cli

Not all RPCs on Linux need to have support from the TEE supplicant

- EL1: Single threaded Bootloader, U-boot



Example: HSM Key Rotation, scp_03 cmd

https://u-boot.readthedocs.io/en/latest/usage/cmd/scp03.html

SPL can load and boot OP-TEE so it is ready for U-boot

drivers/tee/optee/rpc.c

```
void optee_rpc_cmd(struct tee_context *ctx, struct optee *optee,
                   struct optee_msg_arg *arg)
{
        switch (arg->cmd) {
        case OPTEE_RPC_CMD_GET_TIME:
                handle_rpc_func_cmd_get_time(arg);
                break;
        case OPTEE_RPC_CMD_NOTIFICATION:
                handle_rpc_func_cmd_wq(optee, arg);
                break;
        case OPTEE_RPC_CMD_SUSPEND:
                handle_rpc_func_cmd_wait(arg);
                break;
        case OPTEE_RPC_CMD_I2C_TRANSFER:
                handle_rpc_func_cmd_i2c_transfer(ctx, arg);
                break;
        default:
                handle_rpc_supp_cmd(ctx, optee, arg);
        }
}
```

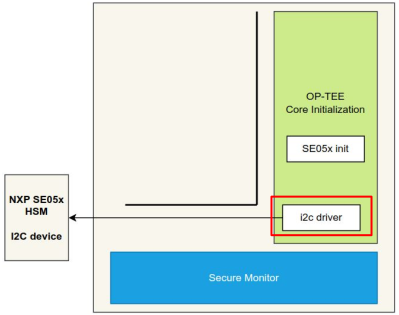
Use Case: NXP SE05X HSM (I2C)

- THREAD_RPC: delegate to EL1

- NXP SE050x Root of Trust at IC level
- FIPS 140-2 Certified Security device
- Keys can be pre-provisioned in memory
- Pre-provisioned keys can be imported and handled by OP-TEE's PKCS#11

- Provisioning (SCP03)
  - Data on i2c bus is **AES-GCM** with session keys.
  - AES-GCM in SW (LibTomCrypt)
  - Keys derived from programmable NVM values
  - KDF uses the HUK
    - HUK must be stable

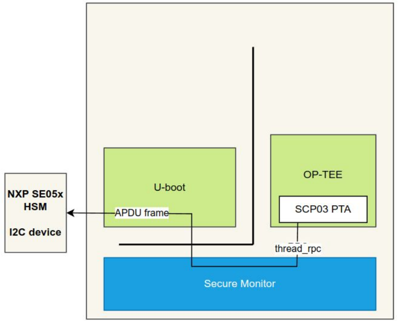
Early boot



Device Provisioning
Unique identifier
GP SCP03 enablement
Key rotation

Controlling access to the device during the boot sequence

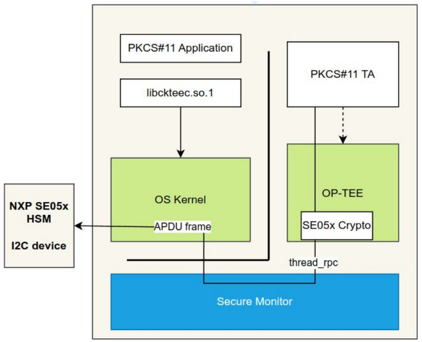
Normal World boot loader



Device Provisioning
Might update SCP03 keys

```c
int glue_i2c_init(void)
{
        if (transfer == rpc_io_i2c_transfer)
                return 0;

        transfer = native_i2c_transfer;

        return native_i2c_init();
}

static TEE_Result load_trampoline(void)
{
        if (IS_ENABLED(CFG_CORE_SE05X_I2C_TRAMPOLINE))
                transfer = rpc_io_i2c_transfer;

        return TEE_SUCCESS;
}

boot_final(load_trampoline);
```

Normal World user

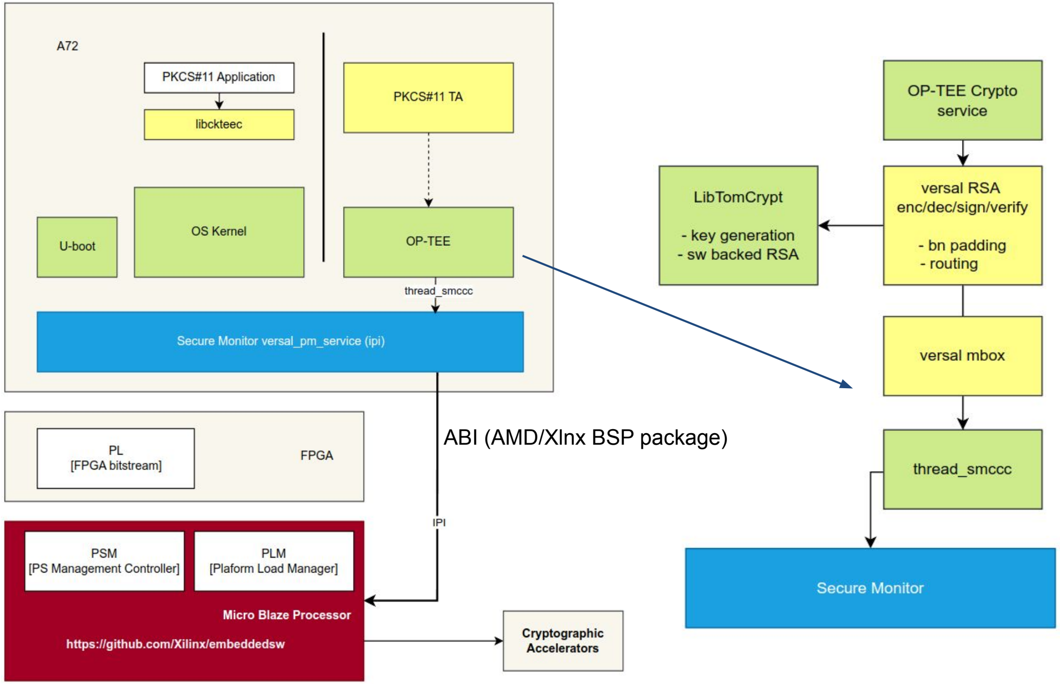


Secure Storage
Crypto Operations
RSA
ECC
TRNG
AES-CTR

THE LINUX FOUNDATION

Use Case: Versal ACAP Crypto

- THREAD_SMCCC delegate to EL3
- Most Secure Services are implemented in the **PLM firmware** executing in a remote processor
  - Cryptography
    - AES-GCM
    - RSA/ECC
    - SHA3-384
    - Physical Unclonable Device
  - eFuse
  - FPGA configuration
- Others are implemented natively in OP-TEE
  - TRNG, GPIO
- The Mailbox driver is implemented in the **Secure Monitor** firmware routing to the PLM
  - Defines ABI with PLM services
- Device tree: critical services in use by the Secure World must be reserved and its **secure-status** property set accordingly.[1]



[1] https://www.kernel.org/doc/Documentation/devicetree/bindings/arm/secure.txt

References

- SMC Calling Convention : https://developer.arm.com/documentation/den0028/latest
- OP-TEE: https://optee.readthedocs.io/en/latest/
  - QEMU ARMv8 https://optee.readthedocs.io/en/latest/building/devices/qemu.html#qemu-v8
- AMD Versal ACAP : https://optee.readthedocs.io/en/latest/building/devices/versal.html
  - Crypto Driver: https://github.com/OP-TEE/optee_os/tree/master/core/drivers/crypto/versal
  - **AMD Versal ACAP, Technical Overview, Embedded World 2023**, https://shorturl.at/gLOQ0
- SE05X: https://optee.readthedocs.io/en/latest/architecture/crypto.html#nxp-se05x-family-of-secure-elements
  - Crypto Driver: https://github.com/OP-TEE/optee_os/tree/master/core/drivers/crypto/se050
  - **Secure Elements in a TEE, Embedded Recipes 2022,** https://shorturl.at/txFW3

LINUX SECURITY SUMMIT

EUROPE

Secure Boot on TEE

SoC fuses/OTP registers store the Root of Trust

- The public key hash to save register space.
- The first bootloader MUST be signed with the RoT key
    - The public key is then embedded in the image for ROM verification

The first bootloader then boots a FIT image [multiple signed binaries in a single file]

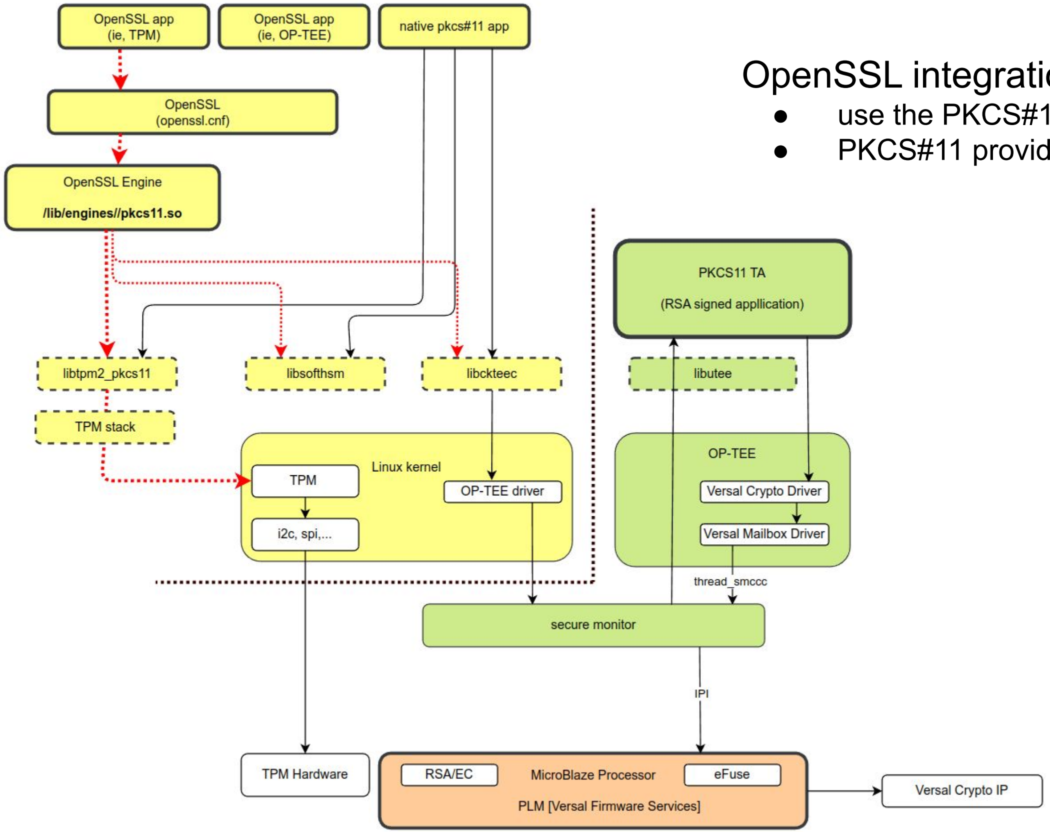
- Images in the FIT should be **signed with different keys** to reduce the attack surface **and enable key rotation**
    - Reduce RoT key exposure [unless the FIT is also being validated by the ROM]
- The First bootloader will validate and boot the subsequent firmwares.
    - Usually a software implementation of the cryptographic driver.

Once the TEE is up early in the boot process (**unless you are a ChromeOS user)** it can access the IP to perform cryptographic operations. For example

- AES-GCM (for secure storage)
- RSA verification of the TAs

U-Boot can then access OP-TEE for further validation and Secure Storage in RPMB.

BACKUP



OpenSSL integration

- use the PKCS#11 engine
- PKCS#11 provider not quite ready

ARM64 Code dive

Normal World EL1

```c
static u32 do_call_with_arg(struct udevice *dev, struct optee_msg_arg *arg)
{
	struct optee_pdata *pdata = dev_get_plat(dev);
	struct rpc_param param = { .a0 = OPTEE_SMC_CALL_WITH_ARG };
	void *page_list = NULL;

	reg_pair_from_64(&param.a1, &param.a2, virt_to_phys(arg));
	while (true) {
		struct arm_smccc_res res;

		/* If cache are off from U-Boot, sync the cache shared with OP-TEE */
		if (!dcache_status())
			flush_shm_dcache(dev, arg);

		pdata->invoke_fn(param.a0, param.a1, param.a2, param.a3,
				 param.a4, param.a5, param.a6, param.a7, &res);

		/* If cache are off from U-Boot, sync the cache shared with OP-TEE */
		if (!dcache_status())
			flush_shm_dcache(dev, arg);

		free(page_list);
		page_list = NULL;

		if (OPTEE_SMC_RETURN_IS_RPC(res.a0)) {
			param.a0 = res.a0;
			param.a1 = res.a1;
			param.a2 = res.a2;
			param.a3 = res.a3;
			handle_rpc(dev, &param, &page_list);
		} else {
			/*
			 * In case we've accessed RPMB to serve an RPC
			 * request we need to restore the previously
			 * selected partition as the caller may expect it
			 * to remain unchanged.
			 */
			optee_suppl_rpmb_release(dev);
			return call_err_to_res(res.a0);
		}
	}
}
```

Notifies the tee_supplicant (RPMB, I2C, …)

Then calls the monitor again

Monitor EL3

opteed_smc_handler

```c
/*
 * OPTEE is returning from a call or being preempted from a call, in
 * either case execution should resume in the normal world.
 */
case TEESMC_OPTEED_RETURN_CALL_DONE:
	/*
	 * This is the result from the secure client of an
	 * earlier request. The results are in x0-x3. Copy it
	 * into the non-secure context, save the secure state
	 * and return to the non-secure state.
	 */
	assert(handle == cm_get_context(SECURE));
	cm_el1_sysregs_context_save(SECURE);

	/* Get a reference to the non-secure context */
	ns_cpu_context = cm_get_context(NON_SECURE);
	assert(ns_cpu_context);

	/* Restore non-secure state */
	cm_el1_sysregs_context_restore(NON_SECURE);
	cm_set_next_eret_context(NON_SECURE);

	SMC_RET4(ns_cpu_context, x1, x2, x3, x4);
```

S-EL1

```asm
/* void thread_smccc(struct thread_smc_args *arg_res) */

FUNC thread_smccc , :
	push	x0, xzr
	mov	x8, x0
	load_xregs x8, 0, 0, 7
#ifdef CFG_CORE_SEL2_SPMC
	hvc	#0
#else
	smc	#0
#endif
	pop	x8, xzr
	store_xregs x8, 0, 0, 7
	ret
END_FUNC thread_smccc
```

```asm
/* void thread_rpc_spsr(uint32_t rv[THREAD_RPC_NUM_ARGS], uint64_t spsr) */

FUNC thread_rpc_spsr , :
	/* Mask all maskable exceptions before switching to temporary stack
	msr	daifset, #DAIFBIT_ALL
	push	x0, xzr
	push	x1, x30
	bl	thread_get_ctx_regs
	ldr	x30, [sp, #8]
	store_xregs x0, THREAD_CTX_REGS_X19, 19, 30
	mov	x19, x0

#if defined(CFG_CORE_PAUTH)
	/* Save APIAKEY */
	read_apiakeyhi	x1
	read_apiakeylo	x2
	store_xregs x0, THREAD_CTX_REGS_APIAKEY_HI, 1, 2
#endif

	bl	thread_get_tmp_sp
	pop	x1, xzr		/* Match "push x1, x30" above */
	mov	x2, sp
	str	x2, [x19, #THREAD_CTX_REGS_SP]
	ldr	x20, [sp]	/* Get pointer to rv[] */
	mov	sp, x0		/* Switch to tmp stack */
	/*
	 * We need to read rv[] early, because thread_state_suspend
	 * can invoke virt_unset_guest() which will unmap pages,
	 * where rv[] resides
	 */
	load_wregs x20, 0, 21, 23	/* Load rv[] into w20-w22 */

	adr	x2, .thread_rpc_return
	mov	w0, #THREAD_FLAGS_COPY_ARGS_ON_RETURN
	bl	thread_state_suspend
	mov	x4, x0		/* Supply thread index */
	ldr	w0, =TEESMC_OPTEED_RETURN_CALL_DONE
	mov	x1, x21
	mov	x2, x22
	mov	x3, x23
	smc	#0
	/* SMC should not return */
	panic_at_smc_return
```

```c
int thread_state_suspend(uint32_t flags, uint32_t cpsr, vaddr_t pc)
{
	struct thread_core_local *l = thread_get_core_local();
	int ct = l->curr_thread;

	assert(ct != THREAD_ID_INVALID);

	if (core_mmu_user_mapping_is_active())
		ftrace_suspend();

	thread_check_canaries();

	release_unused_kernel_stack(threads + ct, cpsr);

	if (is_from_user(cpsr)) {
		thread_user_save_vfp();
		tee_ta_update_session_utime_suspend();
		tee_ta_gprof_sample_pc(pc);
	}
	thread_lazy_restore_ns_vfp();

	thread_lock_global();

	assert(threads[ct].state == THREAD_STATE_ACTIVE);
	threads[ct].flags |= flags;
	threads[ct].regs.cpsr = cpsr;
	threads[ct].regs.pc = pc;
	threads[ct].state = THREAD_STATE_SUSPENDED;

	threads[ct].have_user_map = core_mmu_user_mapping_is_active();
	if (threads[ct].have_user_map) {
		if (threads[ct].flags & THREAD_FLAGS_EXIT_ON_FOREIGN_INTR)
			tee_ta_ftrace_update_times_suspend();
		core_mmu_get_user_map(&threads[ct].user_map);
		core_mmu_set_user_map(NULL);
	}

	if (IS_ENABLED(CFG_SECURE_PARTITION)) {
		struct ts_session *ts_sess =
			TAILQ_FIRST(&threads[ct].tsd.sess_stack);

		spmc_sp_set_to_preempted(ts_sess);
	}

	l->curr_thread = THREAD_ID_INVALID;

	if (IS_ENABLED(CFG_NS_VIRTUALIZATION))
		virt_unset_guest();

	thread_unlock_global();

	return ct;
}
```

Zynqmp: U-boot/Linux