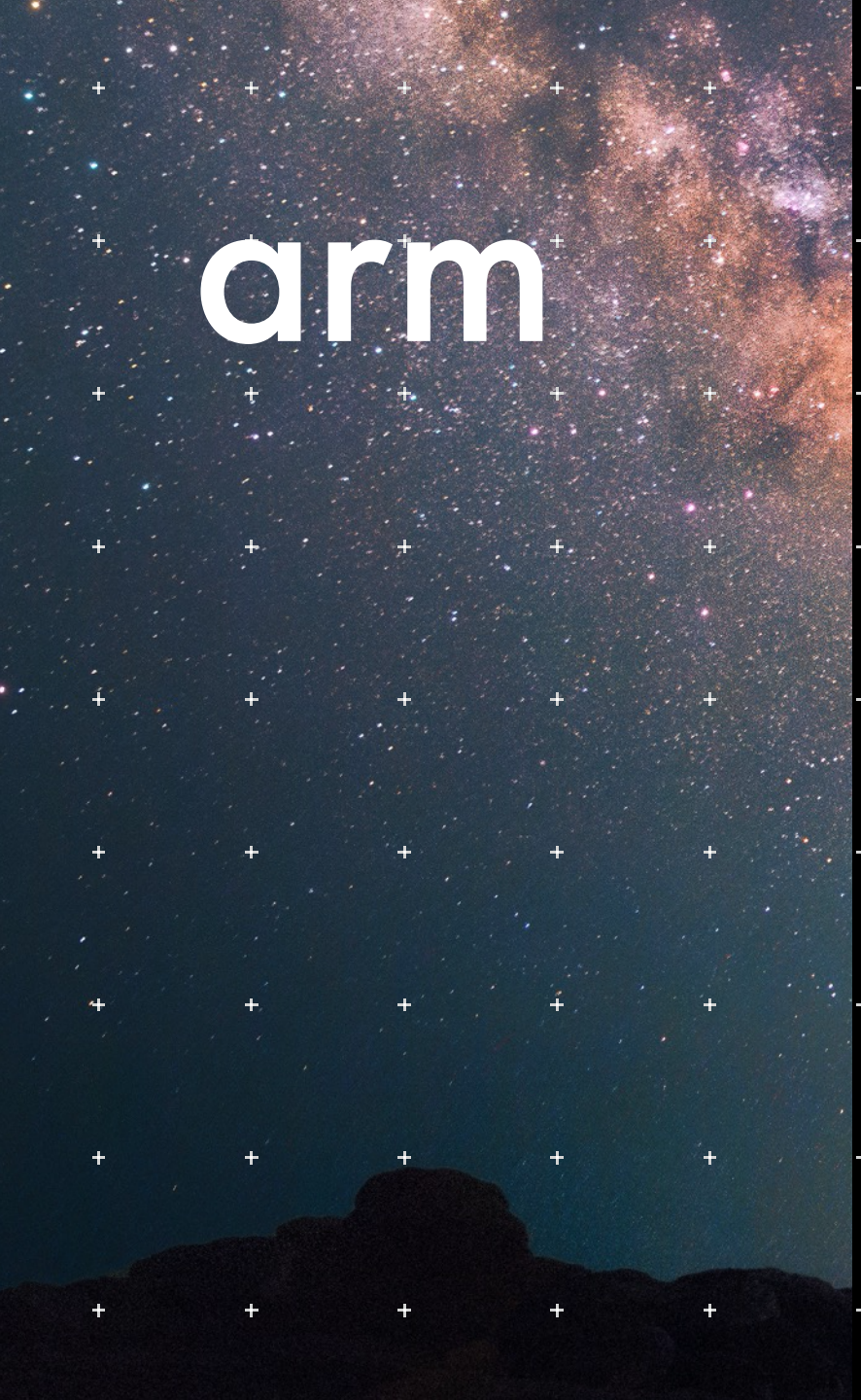


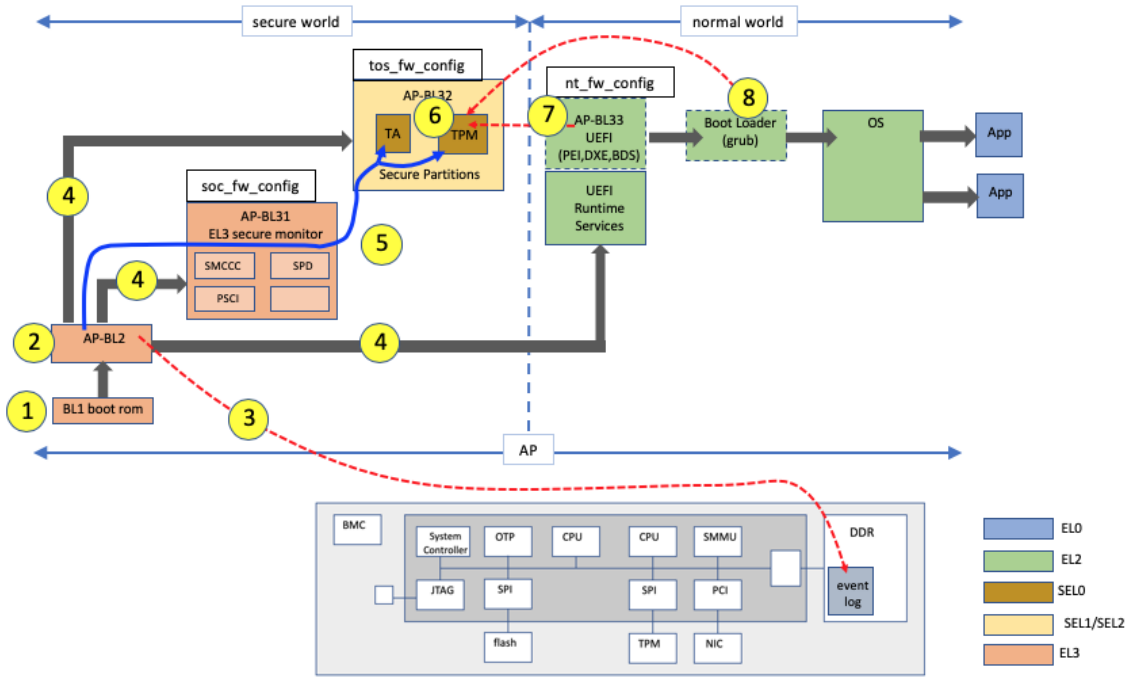
TF-A Technical Forum

TF-A and Measured Boot

October 2020

TF-A and Measured Boot

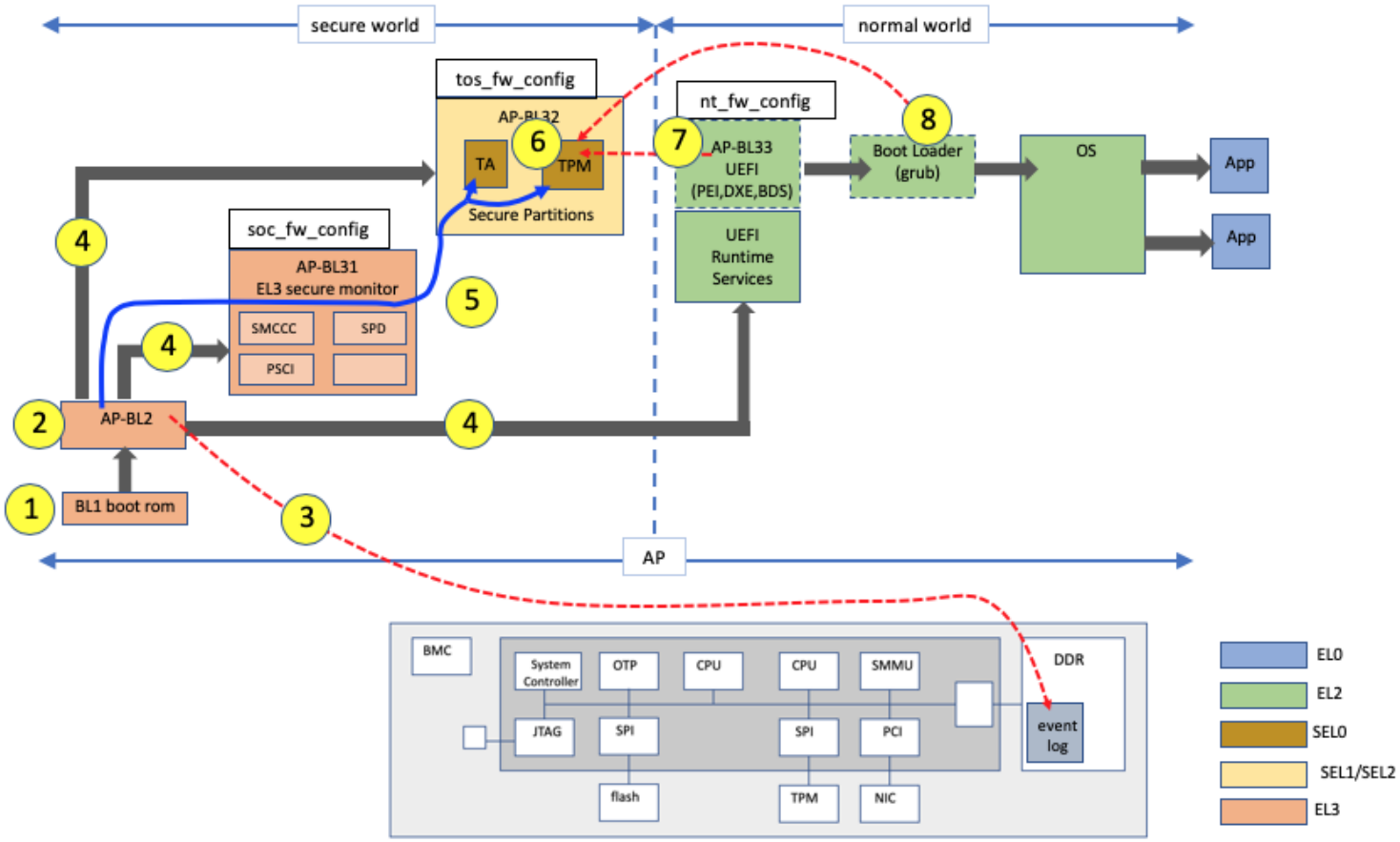
- **Goals**
  - Supply a Measured Boot capability to the existing BL1, BL2, BL31, BL32, BL33 boot flow.
  - Supply a secure Event Log and APIs that manipulate it to a platform provided TPM solution.
  - Supply a fTPM test service to validate the supplied Event Log and API's .
  - Provide guidance for platform providers to supply a full Measured Boot solution.



- **Challenges**
  - TF-A needs to measure BL image boot for a variety of undefined Platform TPM solutions.
  - An fTPM test service has been implemented as an OPTEE trusted app based on a number of external Open Source projects (more on this later by Javier)
  - Feedback on an initial TF-A implementation had identified a number of possible extensions that have attempted to be addressed in later patches.
    - ***Upgrading/changing the hash algorithm***
    - ***Requires new API for passing hash algorithm from platform configuration/eFuses***
    - ***Requires new API for passing platform specific 'image_data'***
    - ***SPM manifest for passing Event Log, CRB region***
    - ***ACPI TPM2 table extension – needs SPCI definition***
    - ***Normal world (BL33, OS) to SPCI-based TPM service***
    - ***Secure partition to SPCI-based TPM service***
    - ***…***

arm

TF-A Measured Boot Flow



Boot Flow Steps and Implementation Details

• SHA256/384/512 hash algorithm for Measured Boot calculations is passed as 'TPM_HASH_ALG' build parameter used by Mbed TLS.

1. At reset the boot ROM (BL1 in this example) executes, verifies the first mutable firmware image (BL2), and transfers control to BL2.
BL1 has the option of measuring BL2 and passing that measurement as a parameter to BL2.
• *BL1 calculates BL2 image hash and passes these data to BL2 via 'FW_CONFIG/TB_FW_CONFIG' device tree in new 'bl2_hash_data' byte array added in 'fvp_fw_config.dts for FVP'.*

2. BL2 performs its normal initialisation and receives its own hash from BL1 image.
• *BL2 has an option of measuring itself but by the time Measured Boot driver starts, BL2 image data are changed because of the initialisation of global variables and memory tables.*

3. BL2 writes the measurement of BL2 (i.e. itself) to the TPM Event Log in Secure memory.

4. BL2 loads, verifies, and measures each additional firmware image (e.g. BL31, BL32, BL33) along with any critical data (e.g. SOC_FW_CONFIG, TOS_FW_CONFIG, NT_FW_CNFIG) and writes the measurement into the TPM Event Log.

5. BL2 transfers control to BL31, which then initialises and transfers control to the BL32 Secure Partition code.

Boot Flow Steps and Implementation Details

6. The TPM Secure Partition service initialises. The initialisation process includes initialising the TPM, reading the Event Log and extending all measurements from the log into the TPM.

• Event Log address and size is passed by TOS_FW_CONFIG device tree in 2 new added properties:

- Property name: *'tpm_event_log_addr'*
  Value type is an unsigned 64-bit integer specifying the physical address of the event log.
- Property name: *'tpm_event_log_size'*
  Value type is an unsigned 32-bit integer specifying the size of the Event Log.

```
/* TPM Event Log Config */
tpm_event_log {
    compatible = "arm,tos_fw";
    tpm_event_log_addr = <0x0 0x0>;
    tpm_event_log_size = <0x0>;
};
```

7. BL2 transfers control to BL33 (UEFI), including passing the address of the Event Log. UEFI measures all drivers and programs (e.g. grub) loaded from external storage and writes them into the TPM (via the SP TPM service) and Event Log.

• Event Log is calculated by BL2 in Secure memory and copied to Non-secure memory.
Address in Non-secure memory is calculated as:

"nt_fw_config_addr + nt_fw_config_max_size"
with values obtained from 'tb_fw_config':
nt_fw_config_addr = <0x0 0x80000000>;
nt_fw_config_max_size = <0x200>;

arm

Boot Flow Steps and Implementation Details

TF-A provides Event Log to the BL33 (TFTF/UEFI/U-boot) in ‘NT_FW_CONFIG’ device tree,
which address is passed by BL31 as 'arg0’ parameter.

8. Bootloader (grub) measures the OS kernel and writes it into the TPM (via the SP TPM service) and Event Log.

Build Options

- **TPM_HASH_ALG**: hash algorithm for Measured Boot calculations used by MbedTLS.
  Corresponds to HASH_ALG values “sha256” (default), “sha384”, “sha512” used by Trusted Board Boot.

- **EVENT_LOG_SIZE**: Event Log length in bytes, default 1024.
  FVP booting TFTF with 8 recordered events: sha256: 486; sha384: 598; sha512: 710 bytes.

- **EVENT_LOG_LEVEL**: chooses the log level for Measured Boot driver. See LOG_LEVEL values, default is 40 (LOG_LEVEL_INFO)

Platform specific details

```
#define BL2_STRING                      "BL_2"
#define BL31_STRING                     "BL_31"
#define BL32_STRING                     "BL_32"
#define BL33_STRING                     "BL_33"
#define HW_CONFIG_STRING                "HW_CONFIG"
#define NT_FW_CONFIG_STRING             "NT_FW_CONFIG"
#define SOC_FW_CONFIG_STRING            "SOC_FW_CONFIG"
#define TOS_FW_CONFIG_STRING            "TOS_FW_CONFIG"

typedef struct {
        unsigned int id;
        const char *name;
        unsigned int pcr;
} image_data_t;

typedef struct {
        const image_data_t *images_data;
        int (*set_nt_fw_info)(uintptr_t config_base,
#ifdef SPD_opteed
                              uintptr_t log_addr,
#endif
                              size_t log_size, uintptr_t *ns_log_addr);
        int (*set_tos_fw_info)(uintptr_t config_base, uintptr_t log_addr,
                               size_t log_size);
} measured_boot_data_t;
```


arm

Platform specific details

```
/* Platform's table with platform specific image IDs, names and PCRs */
static const image_data_t plat_images_data[] = {
        { BL2_IMAGE_ID, BL2_STRING, PCR_0 },     /* Reserved for BL2 */
        { INVALID_ID, NULL, (unsigned int)(-1) }       /* Terminator */
};

static const measured_boot_data_t plat_measured_boot_data = {
        plat_images_data,
        NULL,      /* platform_set_nt_fw_info */
        NULL       /* platform_set_tos_fw_info */
};

/*
 * Function returns pointer to platform's measured_boot_data_t structure
 *
 * Must be overridden in the platform code
 */
#pragma weak plat_get_measured_boot_data

const measured_boot_data_t *plat_get_measured_boot_data(void)
{
        return &plat_measured_boot_data;
}
```

FVP example

```
/* FVP table with platform specific image IDs, names and PCRs */
static const image_data_t fvp_images_data[] = {
        { BL2_IMAGE_ID, BL2_STRING, PCR_0 },                    /* Reserved for BL2 */
        { BL31_IMAGE_ID, BL31_STRING, PCR_0 },
        { BL32_IMAGE_ID, BL32_STRING, PCR_0 },
        { BL33_IMAGE_ID, BL33_STRING, PCR_0 },
        { HW_CONFIG_ID, HW_CONFIG_STRING, PCR_0 },
        { NT_FW_CONFIG_ID, NT_FW_CONFIG_STRING, PCR_0 },
        { SOC_FW_CONFIG_ID, SOC_FW_CONFIG_STRING, PCR_0 },
        { TOS_FW_CONFIG_ID, TOS_FW_CONFIG_STRING, PCR_0 },
        { INVALID_ID, NULL, (unsigned int)(-1) } /* Terminator */
};
static const measured_boot_data_t fvp_measured_boot_data = {
        fvp_images_data,
        arm_set_nt_fw_info,
        arm_set_tos_fw_info
};
/*
 * Function retuns pointer to FVP plat_measured_boot_data_t structure
 */
const measured_boot_data_t *plat_get_measured_boot_data(void)
{
        return &fvp_measured_boot_data;
}
```


arm

Feedback and future work

"Would be good if the hash alg comes from the config file. This will make the implementation "crypto agile" from the very beginning. It is common to want to upgrade/change the hash algorithm and since BL1 is in ROM, you potentially break measured boot on old devices in case a hash algorithm is broken. The other option is to get the hash algorithm from the platform, perhaps a platform gets it from eFuses as opposed to config files."

- Upgrading/changing the hash algorithm in the run time requires building TF-A with SHA512 increasing the size of Mbed TLS internal buffers.
- Requires new API for passing hash algorithm from platform configuration/eFuses to Measured Boot driver.

"Need a provision to support platform specific image id's and image_data. A FIP can contain platform specific images, that may need to be measured too."

- Implemented.

- *When using NT_FW_CONFIG for passing Event Log, implement flag that specifies whether measurements are recorded in TPM.*
- Planned.