TrustZone

(TrustZone Secure World / Non-Secure World)

---

1. 개요

1.1. 목적

- ARM Cortex 프로세서에 내장된 하드웨어 기반 보안 기술로 시스템을 Secure world와 Non-Secure world로 분리하여 암복호화나 키 등을 안전한 영역에서 실행, 보관되도록 한다. Secure world는 외부에서 rwx 모두 불가능하고 정해진 handler만을 이용하여 접근을 할 수 있다.

1.2. 주요 기능

- TrustedFirmware-M에 종속된 기능으로 같이 사용
- FLASH 영역의 Secure world / Non-Secure world 분리로 인한 신뢰 컴퓨팅 환경 구성
- Secure world는 정해진 handler를 이용해야만 접근이 가능

1.3. Secure world / Non-Secure world

- MPU와 함께 사용하면 Secure world와 Non-Secure world 각각에 대해 privileged와 unprivileged 권한에 대한 읽기/쓰기 권한 설정 가능
- FLASH영역을 분리해서 사용하기에 메모리맵이 따로 있음
- 기본적인 TrustedFirmware-M의 api기능은 Secure world에서 동작함
- Config File을 통해 handler 지정과 접근 함수등을 설정할 수 있음.
- Secure world의 main함수는 따로 존재하며 별도로 빌드되고 추후에 Non-Secure world 빌드 펌웨어와 합쳐지는 구조를 가짐

---

2. High-Level

2.1. TrustZone 실행 흐름

1. Config 파일에서 handler 함수 설정
2. Non-Secure world에서 설정된 handler함수를 인자로 psa_call 함수 실행
3. psa_call 함수를 통하여 Secure world에 접근하여 함수 실행
4. 결과를 psa_call 함수를 통해서 받아옴

---

3. Module-Level

zephyr 설치 위치 기준

3.1. main 함수

`modules/tee/tf-m/trusted-firmware-m/secure_fw/spm/core/main.c`
인자 없음.
TrustedFirmware-M을 사용하면 기본적으로 실행되는 main 함수
Secure world에 포함되어 있음

3.2. psa_initial_attest_get_token

`modules/tee/tf-m/trusted-firmware-m/interface/src/tfm_attest_api.c`
att_get_iat 함수에서 호출되는 psa api 함수로 Non-Secure world에 존재하며 psa_call함수를 사용해서 Secure world의 PSA IAT 생성 함수에 접근하는 함수
Non-Secure world에 존재

Parameter	Value	설명
const uint8_t *auth_challenge	ch_buffer	nonce 값
size_t challenge_size	ch_sz	nonce 크기=64
uint8_t *token_buf	token_buffer	생성된 iat가 저장되는 buffer
size_t token_buf_size	token_buf_size	buf 크기 0x240
size_t *token_size	token_sz	iat 토큰 크기 저장

3.3. psa_status_t psa_call

`modules/tee/tf-m/trusted-firmware-m/interface/src/tfm_psa_call.c`
PSA API를 호출하는 함수로 Secure world의 handler를 이용해서 Secure world에 존재하는 함수를 실행
이 함수는 Non-Secure world에 존재하고 이후 동작은 Secure world에서 실행

Parameter	Value	설명
psa_handle_t handle	TFM_ATTESTATION_SERVICE_HANDLE	호출하려는 Secure world 함수 handle
int32_t type	TFM_ATTEST_GET_TOKEN	함수 타입
const psa_invec *in_vec	in_vec	인자
size_t in_len	IOVEC_LEN(in_vec)	인자 요소 개수
psa_outvec *out_vec	out_vec	출력

size_t out_len	IOVEC_LEN(out_vec)	출력 요소 개수

---

4. Source-Level

4.1. flash_layout.h

modules/tee/tf-m/trusted-firmware-m/platform/ext/target/stm/nucleo_l552ze_q/partition/flash_layout.h

FLASH영역을 나눌 때 Secure world의 크기와 Non-Secure world의 크기를 지정하는 부분
보드의 크기에 따라 나눠야하며 반드시 해당보드의 dts파일과 같이 수정해야한다.
만약 펌웨어의 크기가 커진다면 유동적으로 조절하며 빌드

```
#define FLASH_S_PARTITION_SIZE          (0x2D000) /* S partition */
// #define FLASH_NS_PARTITION_SIZE         (0x9000) /* NS partition */
#define FLASH_NS_PARTITION_SIZE         (0x20000) /* NS partition */
#define FLASH_PARTITION_SIZE
(FLASH_S_PARTITION_SIZE+FLASH_NS_PARTITION_SIZE)
```

4.2. nucleo_l552ze_q_stm32l552xx_ns.dts

zephyr/boards/st/nucleo_l552ze_q/nucleo_l552ze_q_stm32l552xx_ns.dts
보드의 메모리맵을 나타내는 파일로 4.1. flash_layout.h 파일과 같이 수정을 진행해야 한다.

```
&flash0 {
	partitions {
		compatible = "fixed-partitions";
		#address-cells = <1>;
		#size-cells = <1>;
		/*
		 * Following flash partition is compatible with requirements
		 * given in TFM configuration given for current board:
		 * multiple image boot, no tests.
		 * It might require adjustment depending on evolutions on
TFM.
		 */
		boot_partition: partition@0 {
			label = "mcuboot";
			reg = <0x00000000 DT_SIZE_K(80)>;
			read-only;
		};
		/* Secure image primary slot */
		slot0_partition: partition@14000 {
			label = "image-0";
```

```
                reg = <0x00014000 DT_SIZE_K(180)>;
            };
            /* Non-secure image primary slot */
            slot0_ns_partition: partition@41000 {
                label = "image-0-nonsecure";
                reg = <0x00041000 DT_SIZE_K(128)>;
            };
            /* Secure image secondary slot */
            slot1_partition: partition@4a000 {
                label = "image-1";
                reg = <0x0004a000 DT_SIZE_K(88)>;
            };
            /* Non-secure image secondary slot */
            slot1_ns_partition: partition@77000 {
                label = "image-1-nonsecure";
                reg = <0x00077000 DT_SIZE_K(36)>;
            };
            /* Applicative Non Volatile Storage */
            /* Not available in this config, use secure storage */
        };
};
```

4.3. psa_initial_attest_get_token

modules/tee/tf-m/trusted-firmware-m/interface/src/tfm_attest_api.c
psa_call 함수를 이용하여 PSA IAT 생성 함수를 호출
Non-Secure world에 존재

```
psa_status_t
psa_initial_attest_get_token(const uint8_t *auth_challenge,
                             size_t         challenge_size,
                             uint8_t       *token_buf,
                             size_t         token_buf_size,
                             size_t        *token_size)
{
    psa_status_t status;

    psa_invec in_vec[] = {
        {auth_challenge, challenge_size}
    };
    psa_outvec out_vec[] = {
        {token_buf, token_buf_size}
    };

    status = psa_call(TFM_ATTESTATION_SERVICE_HANDLE, TFM_ATTEST_GET_TOKEN,
                      in_vec, IOVEC_LEN(in_vec),
                      out_vec, IOVEC_LEN(out_vec));
```

```
    if (status == PSA_SUCCESS) {
        *token_size = out_vec[0].len;
    }

    return status;
}
```

4.4. psa_call

modules/tee/tf-m/trusted-firmware-m/interface/src/tfm_psa_call.c
인자로 받은 handle에 따라 Secure world에 존재하는 함수를 실행시키는 함수
Non-Secure world에 존재하며 이후 동작은 Secure world에서 동작
PSA IAT를 발급하는 모든 함수는 Secure world에서 동작

```
psa_status_t psa_call(psa_handle_t handle,
                      int32_t type,
                      const psa_invec *in_vec,
                      size_t in_len,
                      psa_outvec *out_vec,
                      size_t out_len)
{
    if ((type    > PSA_CALL_TYPE_MAX) ||
        (type    < PSA_CALL_TYPE_MIN) ||
        (in_len  > PSA_MAX_IOVEC)     ||
        (out_len > PSA_MAX_IOVEC)) {
        psa_panic();
    }

    return tfm_psa_call_pack(handle, PARAM_PACK(type, in_len, out_len),
                             in_vec, out_vec);
}
```
