CSEC 659/559 Trusted Computing and Trusted Execution
Lab1: Getting Started with ARM TrustZone

Instructor: Ivan De Oliveira Nunes

1 Introduction

TrustZone is a hardware extension available in modern ARM CPUs that provides a mechanism to isolate security critical components in a system. The hardware separates a rich general-purpose operating system, from a much smaller (thus less likely to have vulnerabilities), secure operating system. The former is commonly referred to as "normal world", while the latter is referred to as "secure" or "trusted world". The main idea behind TrustZone is that bugs, vulnerabilities, or malicious code located in the normal world are (should be) unable to corrupt or tamper with processes running in the secure world. Future lectures will cover in details how TrustZone achieves this separation and why/how this can make systems more secure. However, this knowledge is not strictly required for completing this lab.

In this lab, you will study how to use ARM TrustZone. Instead of using a physical ARM TrustZone device, you will use the QEMU emulator[1]. In the normal world, we run a Linux distribution. In the secure world, we run the `OP-TEE` operating system. The environment is illustrated in Figure 1. One can setup this environment by following the directions at `https://optee.readthedocs.io/en/latest/building/gits/build.html`. `OP-TEE` is a small OS specifically designed to run within TrustZone's secure world and to support execution of multiple "secure apps" within the secure world. In principle, `OP-TEE` and its secure apps can not be compromised, even if all normal apps and the rich OS (e.g., Linux), running in the normal world, have been compromised.

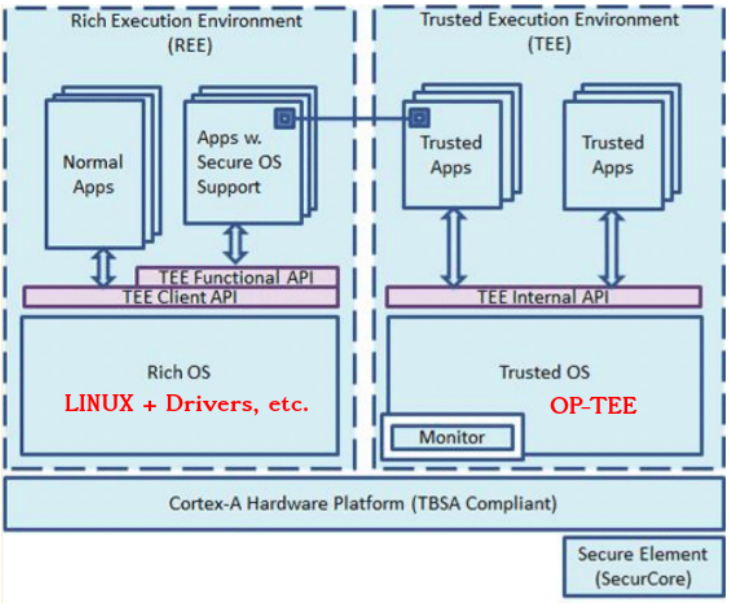

Figure 1: ARM Cortex A: Normal world runs Linux, Trusted world runs OP-TEE.

1.1 Deliverables

Submit a lab report on `MyCourses`. The report should have screenshots and clear answer to all questions.

[1]Most physical devices equipped with TrustZone (e.g., your phone) use secure boot to prevent any changes to TrustZone secure world. So we can't use them for prototyping.

2 Assignments

1. Follow the instructions available at `https://optee.readthedocs.io/en/latest/building/gits/build.html`. This includes the installation of `repo`, if you don't already have it, and pre-requisite packages. The installation process will setup the environment in Figure 1 (with Normal World running Linux and Trusted World running `OP-TEE`). Note that these instructions support builds targeting several platforms. In our case, the target platform is `QEMU` (v7), i.e., a simulator rather than a real device.

2. Open a terminal and change current directory to the build directory created within your `OP-TEE` folder, for example `cd ~/<optee folder path>/build` (this path varies depending on the location you chose for the `OP-TEE` folder in step 1).

   In this directory, type `make run`. The first time you type `make run` it will take a while. It may also tell you that your installation is missing some pre-requisite package. If this happens, go ahead, install that package, and try `make run` again. Repeat this until `make run` succeeds. (pre-requisites can be found here : `https://optee.readthedocs.io/en/latest/building/prerequisites.html#prerequisites`)

   OBS: You can compile faster by parallelizing the compilation process using the -j parameter. Template : `make -j<number_of_cores> run` . Example using 8 cores : `make -j8 run`

   Once `make run` succeeds, you will end up in the QEMU console and it will also spawn two UART consoles. One console containing the UART for secure world and one console containing the UART for normal world.

   You are supposed to see three terminals at this step. Take a screenshot that shows all three terminals.

3. You will see that it stops waiting for input on the QEMU console. To continue, type `c` in the QEMU console. After this step, the QEMU device will keep booting. You will notice that Linux boot is shown in one the terminals. When it stops, it will prompt you to login. Type `root` to login as the root; there is no password.

   Type `uname -a` to show the information of this Linux distribution. Take a screenshot.

4. `xtest` is a test suite built to test TrustZone. Type `xtest -l 0` in the normal world to start testing. **Answer the following question: what do you see in the normal world and secure world terminals?**

5. **Modifying TrustZone Apps:**

   `xtest` runs several sample tests. However, you can also run individual examples. `make run` preloads the compiled binaries of the examples in the directory `/usr/bin`. In the normal world Linux terminal, type `cd /usr/bin`, then type `ls` to see the list of binaries in that directory. `OP-TEE` example executable names start with `optee_example_`. A description of the examples is available at `https://optee.readthedocs.io/en/latest/building/gits/optee_examples/optee_examples.html`.

   Within the `/etc/bin` directory, type `./optee_example_hello_world`. This will run the example program.

   Answer the following question: what do you see in the normal world and secure world terminals?

   Now you are going to modify the behavior of the hello world program by modifying its source code. At this point you can close all three (QEMU, normal world, secure world) terminals.

   In your host machine, you will be able to find the source code of the Hello World example at:

   - Normal world: `~/<optee folder path>/optee_examples/hello_world/host/main.c`

- Secure world: `~/<optee folder path>/optee_examples/hello_world/ta/hello_world_ta.c`

Your task is to modify both of these files such that:

- Normal world prints your full name;
- Instead of 42, normal world APP gives the `ASCII` encoding of the first letter of your first name to the trusted world APP;
- Instead of incrementing the received number by 1, trusted world APP should increment it by 5.

Once you are done with your changes to these files, you can go into `~/<optee folder path>/build` directory and type `make run` again. It will build your changes and load the updated executable into `/usr/bin` folder of the normal world Linux terminal.

In the normal world terminal, type `/usr/bin/optee_example_hello_world` to run your modified TrustZone app. Take a screenshot of both normal world and secure world terminals. The result should reflect your changes to the behavior of the hello world APP (Normal world prints your name, the number corresponds to the first letter of your name, and it gets incremented by 5 by the secure world counterpart).

This exercise illustrates how to modify TrustZone apps in `OP-TEE`. You can use these general steps to modify any TrustZone example APP at will, by changing its untrusted and trusted counterparts. Feel free to explore other cases and available example APPs. This will be useful should you decide to use TrustZone in your final research project.

6. **Adding a new funtionality to hello world:**

   One of the essential features of TrustZone is the ability to isolate and protect the memory of the secure world from the normal world. The most common usage for this is to store secrets and do cryptography operations inside the secure world without leaking any information to the normal world. Your task is to create two new functionalities for the hello world application: encryption and decryption. You will use the RSA algorithm to encrypt the message msg. The encryption function of RSA is the following :

$$enc = msg^E \bmod N; \tag{1}$$

   And the decryption function:

$$msg = enc^D \bmod N; \tag{2}$$

   Where the public key is $K_{pub} = (D = 41, N = 133)$, and the private key is $K_{priv} = (E = 29, N = 133)$. **In this work, the value of msg is your RIT ID number**. To add the functionalities, you are expected to modify three files:

   1. File **hello_world_ta.h** : Define two new IDs as shown bellow:

```
//<optee_path>/out-br/build/optee_examples_ext-1.0/hello_world/ta/include/
    hello_world_ta.h
...
/* The function IDs implemented in this TA */
#define TA_HELLO_WORLD_CMD_INC_VALUE         0
#define TA_HELLO_WORLD_CMD_DEC_VALUE         1
// ADD THE TWO LINES BELLOW
#define TA_ENCRYPT                           2
#define TA_DECRYPT                           3
...
```

2. File **hello_world_ta.c** : implement two new functions `tee_encrypt()` and `tee_decrypt()`. The function `tee_encrypt()` must receive *msg* inside the struct **TEE_Param** and replace its value for its encrypted value *enc*. The function `tee_decrypt()` must receive *enc* inside the struct **TEE_Param** and replace its value for its decrypted value *msg*. **DELIVERY: For each function you will need to print its input and output values (*msg* and *enc*). All the printed messages must appear in the secure world terminal.**. The header template of each function are: (TIPS: You can follow the structure of the functions inc_value() and dec_value() to implement your functions. Try to understand what the hello world code do before hand and then "modify" these functions with the new functionalities)

```
// <optee_path>/optee_examples/hello_world/ta/hello_world_ta.c
...
static TEE_Result tee_encrypt(uint32_t param_types,TEE_Param params[4]){
// CODE HERE
}
...
static TEE_Result tee_decrypt(uint32_t param_types,TEE_Param params[4]){
// CODE HERE
}
...
```

You also need to modify the entry point of the secure world to be able to call the functions above.

```
// <optee_path>/optee_examples/hello_world/ta/hello_world_ta.c
...
TEE_Result TA_InvokeCommandEntryPoint(void __maybe_unused *sess_ctx,
            uint32_t cmd_id,
            uint32_t param_types, TEE_Param params[4])
{
    (void)&sess_ctx;

    switch (cmd_id) {
    // ADD THE TWO CASES BELLOW
    case TA_ENCRYPT :
        return tee_encrypt(param_types, params);
    case TA_DECRYPT :
        return tee_decrypt(param_types, params);
    case TA_HELLO_WORLD_CMD_INC_VALUE:
        return inc_value(param_types, params);
    case TA_HELLO_WORLD_CMD_DEC_VALUE:
        return dec_value(param_types, params);
    default:
        return TEE_ERROR_BAD_PARAMETERS;
    }
}
...
```

3. File **main.c** : The main file of this example runs in the normal world. You will have to modify the main file to call the encryption and decryption functions that are inside the secure world to encrypt and then decrypt your RIT ID. **DELIVERY: You will have to print the values of *msg* before encryption, after decryption, and the value of the encrypted message *enc*. The prints must appear in the normal world terminal**.

```
// <optee_path>/optee_examples/hello_world/host/main.c
```

You have to delivery ...

- All three modified files (add comments to your code please).
- Your RIT ID and the encrypted value of it.
- Screenshot of the secure world and normal world terminal with all the information that was asked (tagged with: **DELIVERY**).