Trusted Execution Environment (TEE)

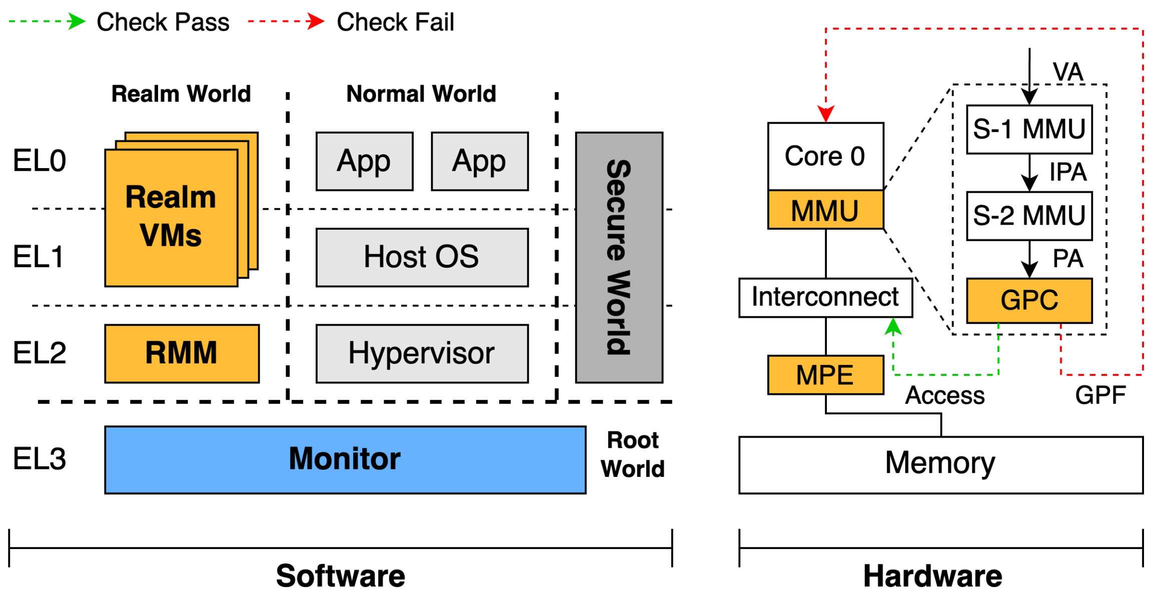
1 Lab Overview

The objective of this lab is for student to gains the hands-on experience on Arm CCA. We choose the Fixed Virtual Platform (FVP) as the experiment platform. The FVP is a complete simulation of an Arm system, including processor, memory and peripherals. As we introduced on lecture, Arm CCA is the next-generation security architecture for Armv9-A processors. After completing this lab, you should be able to understand the basic concepts of Arm CCA and how to use it to safeguard data and code.

2 Before the Task

2.1 CCA Software Stack

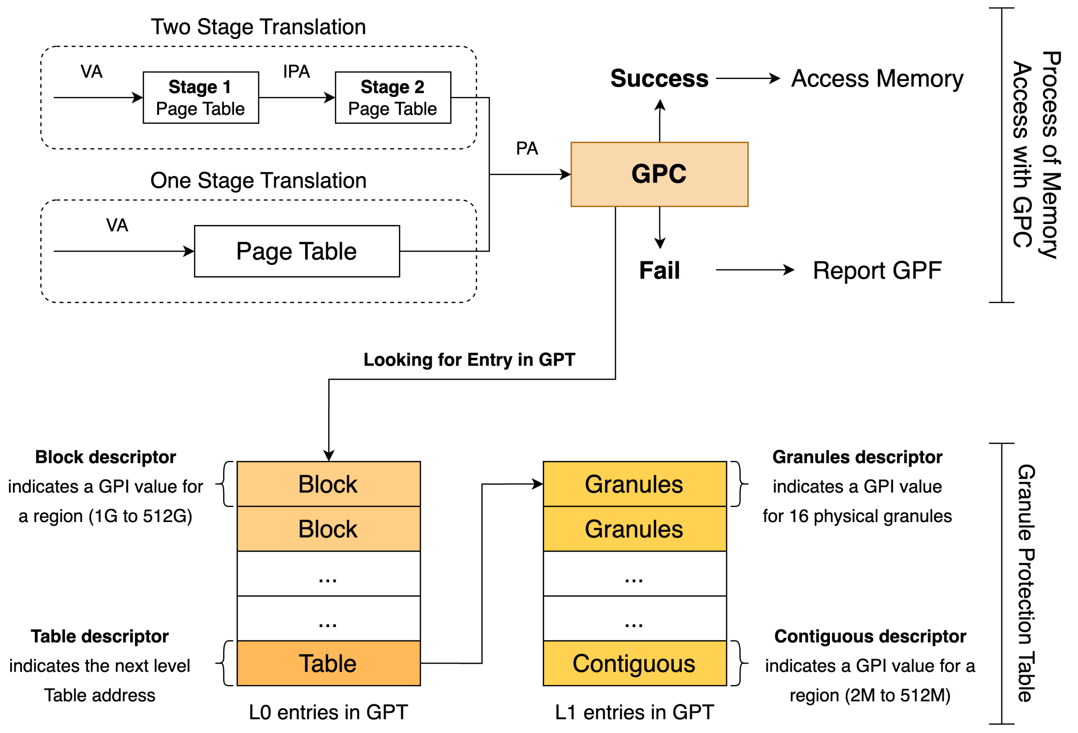
In this lab, we focus on the software stack related to Realm world. Realm world is a new isolated environment introduced by CCA. It is designed to run third-party applications and is isolated from Normal world and Secure world. The software running in the Realm world includes Realm VMs and Realm Management Monitor (RMM).



The Realm VM is a confidential virtual machine (VM) that can be dynamically created by the host in the Normal world. The RMM serves a dual purpose, enabling effective management the execution and memory of Realm VMs while restricting the host's access to memory. RMM exposes a set of Realm Management Interfaces (RMI) to the host, allowing the host to manage the Realm VMs. Through communication with the RMM, the host can make decisions for the Realm VMs, determining their execution schedule and memory allocation. Although the host can manage the Realm VMs, it cannot directly access their content as access to the Realm world from the Normal world is blocked. The RMM is also responsible for the isolation between different Realm VMs and ensures the isolation through the stage 2 translation table.

2.2 Granule Protection Check

CCA uses Granule Protection Check (GPC) to isolate worlds from each other and block invalid access. When translating VA to PA, the Memory Management Unit (MMU) will check whether the current Security state can access the target physical address. GPC performs physical address checks based on the Granule Protection Table (GPT). The GPT is stored in the Root world and has two levels to look up. Each entry in the tables has a Granule Protection Information (GPI). The GPI records information about one granule, such as its associated world. The entries in the GPT at different levels have different descriptors.


Two Stage Translation
VA
Stage 1 Page Table
IPA
Stage 2 Page Table
One Stage Translation
VA
Page Table
PA
GPC
Success
Access Memory
Fail
Report GPF
Process of Memory Access with GPC
Looking for Entry in GPT
Block descriptor indicates a GPI value for a region (1G to 512G)
Block
Block
...
...
Table
Table descriptor indicates the next level Table address
L0 entries in GPT
Granules
Granules
...
...
Contiguous
L1 entries in GPT
Granules descriptor indicates a GPI value for 16 physical granules
Contiguous descriptor indicates a GPI value for a region (2M to 512M)
Granule Protection Table

As shown in Figure 2.2, the GPT is a two-level table. In this lab, we focus on the GPT Table descriptor and the GPT Granule descriptor. A GPT Table descriptor contains a pointer to the base address of a next-level table, and the format is shown in Table 1. In CCA, granule is the minimum unit of memory protection. An 8-byte GPT Granules descriptor contains the GPI values for 16 physical granules. The GPI value to use is bits [(4*i) + 3 : (4*i)] of the descriptor, and the meaning of different encoding values is shown in Table 2.

Bits	Name
[63:52]	Reserved, RES0
[51:12]	Next-level Table Address
[11:4]	Reserved, RES0
[3:0]	0b0011 Table descriptor

Table 1: GPT Table descriptor format

Value	Meaning
0b0000	No accesses permitted
0b1000	Accesses permitted to Secure physical address space only
0b1001	Accesses permitted to Non-secure physical address space only
0b1010	Accesses permitted to Root physical address space only
0b1011	Accesses permitted to Realm physical address space only
0b1111	All accesses permitted
Otherwise	Reserved

Table 2: GPT Granule descriptor encoding

2.3 Docker Image

If you have already installed the necessary tools on your machine, you can skip this section.

To make it easier for you to complete the lab tasks, we have prepared a necessary environment for you. We build a docker image that contains all the necessary tools and libraries. You can use the Docker image with the following steps:

- First, you should have Docker installed on your machine. You can follow the instructions on the official website to install it: `https://docs.docker.com/engine/install/ubuntu/`.
- Then, download the Docker image from our lab website (`https://compass.sustech.edu.cn/cs315/2024Fall/tee-lab/aemfvp-builder.tar.gz`).
- Once downloaded, load the Docker image by running the following command in your terminal:

  ```
  $ docker load -i aemfvp-builder.tar.gz
  ```

To verify that the Docker image was loaded successfully, you can use the command to see whether a newly loaded image in the list:

```
$ docker images
```

2.4 Source Code

We have already uploaded the source code of the lab to our lab website. Please download the source code from the lab website and extract it to your working directory. They are the source code of the TF-RMM, TF-A, Linux Kernel, and kvmtool.

3 Lab Tasks

Note that all the scripts can be found in the lab directory `https://compass.sustech.edu.cn/cs315/2024Fall/tee-lab/scripts.tar.xz`. Other source code can also be found in the lab directory `https://compass.sustech.edu.cn/cs315/2024Fall/tee-lab/`. You can download and extract them to your working directory.

3.1 Task 1: Launch FVP

If you use docker as your compilation environment, you can use following scripts to mount your source code path to docker:

```
$  ./container.sh -v </absolute/path/to/workspace> run
```

After enter docker, you should run:

```
$ export PATH=/opt/aarch64-none-elf/bin:$PATH
```

In this way, we can use aarch64-none-elf for cross compilation.

Then, you can use toolchains in docker env to compile source code for FVP.

TF-RMM is an official implementation of RMM. We have upload the source code, and you can download it from our lab website. You can use following command compile TF-RMM:

```
$  export CROSS_COMPILE=aarch64-none-elf-

$ cd tf-rmm
$ rm -rf build
$ cmake -DRMM_CONFIG=fvp_defcfg -S . -B build -DLOG_LEVEL=40
$ cmake --build build
```

After compilation, you can find a image file in your output path.

As mentioned above, CCA requires system software in Normal world to cooperate with RMM for Realm VM management. Therefore, we need a modified hypervisor. In this lab, we choose KVM as the hypervisor, which is a virtualization module in the Linux kernel that allows the kernel to function as a hypervisor. We have upload the source code, and you can download it from our lab website. You can use following command compile Linux Kernel:

```
$ export CROSS_COMPILE=aarch64-linux-gnu-
$ export ARCH=arm64

$ cd linux-cca
$ make defconfig
$ make all -j16
```

In this lab, we choose TF-A to run as Monitor. During boot stage, TF-A will load RMM to Realm world. Therefore, we need to specify the position of RMM image when compiling TF-A. You can use following command compile TF-A:

```
$ export CROSS_COMPILE=aarch64-none-elf-

$ cd trusted-firmware-a
$ make PLAT=fvp DEBUG=1 clean
$ make CROSS_COMPILE=aarch64-none-elf- \
  PLAT=fvp \
  CTX_INCLUDE_AARCH32_REGS=0 \
  ENABLE_RME=1 \
  RMM=../tf-rmm/build/rmm.img \
  FVP_HW_CONFIG_DTS=fdts/fvp-base-gicv3-psci-1t.dts \
  DEBUG=1 \
  ARM_LINUX_KERNEL_AS_BL33=1 \
  PRELOADED_BL33_BASE=0x84000000 \
  all fip
```

When all the files are compiled, you can get following images: Then, you can use following script to launch FVP: (If you use the docker image, you can find 'FVP_Base_RevC-2xAEMvA' in '~/models'. Otherwise, you need to download the FVP by reading: `https://gitlab.arm.com/`

arm-reference-solutions/arm-reference-solutions-docs/-/blob/master/docs/aemfvp-a-rme/install-fvp.rst).

```
FVP_Base="~/models/Linux64_GCC-9.3/FVP_Base_RevC-2xAEMvA"
BL1_BIN=/path/to/trusted-firmware-a/build/fvp/debug/bl1.bin
FIP_BIN=/path/to/trusted-firmware-a/build/fvp/debug/fip.bin
KERNEL_DIR=/path/to/linux-cca/arch/arm64/boot/Image
DISK_DIR=/path/to/rootfs.ext4

${FVP_Base}                                                     \
  -C bp.refcounter.non_arch_start_at_default=1                    \
  -C bp.secureflashloader.fname=${BL1_BIN}              \
  -C bp.flashloader0.fname=${FIP_BIN}                   \
  -C bp.refcounter.use_real_time=0                                \
  -C bp.ve_sysregs.exit_on_shutdown=1                             \
  -C cache_state_modelled=0                                       \
  -C bp.dram_size=2                                               \
  -C bp.secure_memory=1                                           \
  -C pci.pci_smmuv3.mmu.SMMU_ROOT_IDR0=3                          \
  -C pci.pci_smmuv3.mmu.SMMU_ROOT_IIDR=0x43B                      \
  -C pci.pci_smmuv3.mmu.root_register_page_offset=0x20000         \
  -C cluster0.NUM_CORES=4                                         \
  -C cluster0.PA_SIZE=48                                          \
  -C cluster0.ecv_support_level=2                                 \
  -C cluster0.gicv3.cpuintf-mmap-access-level=2                   \
  -C cluster0.gicv3.without-DS-support=1                          \
  -C cluster0.gicv4.mask-virtual-interrupt=1                      \
  -C cluster0.has_arm_v8-6=1                                      \
  -C cluster0.has_amu=1                                           \
  -C cluster0.has_branch_target_exception=1                       \
  -C cluster0.rme_support_level=2                                 \
  -C cluster0.has_rndr=1                                          \
  -C cluster0.has_v8_7_pmu_extension=2                            \
  -C cluster0.max_32bit_el=-1                                     \
  -C cluster0.stage12_tlb_size=1024                               \
  -C cluster0.check_memory_attributes=0                           \
  -C cluster0.ish_is_osh=1                                        \
  -C cluster0.restriction_on_speculative_execution=2              \
  -C cluster0.restriction_on_speculative_execution_aarch32=2      \
  -C cluster1.NUM_CORES=4                                         \
  -C cluster1.PA_SIZE=48                                          \
  -C cluster1.ecv_support_level=2                                 \
  -C cluster1.gicv3.cpuintf-mmap-access-level=2                   \
  -C cluster1.gicv3.without-DS-support=1                          \
  -C cluster1.gicv4.mask-virtual-interrupt=1                      \
  -C cluster1.has_arm_v8-6=1                                      \
  -C cluster1.has_amu=1                                           \
  -C cluster1.has_branch_target_exception=1                       \
  -C cluster1.rme_support_level=2                                 \
  -C cluster1.has_rndr=1                                          \
  -C cluster1.has_v8_7_pmu_extension=2                            \
  -C cluster1.max_32bit_el=-1                                     \
  -C cluster1.stage12_tlb_size=1024                               \
  -C cluster1.check_memory_attributes=0                          \
  -C cluster1.ish_is_osh=1                                       \
  -C cluster1.restriction_on_speculative_execution=2             \
  -C cluster1.restriction_on_speculative_execution_aarch32=2     \
  -C pctl.startup=0.0.0.0                                        \
  -C bp.smsc_91c111.enabled=1          \
  -C bp.hostbridge.userNetworking=1                      \
  --data cluster0.cpu0=${KERNEL_DIR}@0x84000000 \
  -C bp.virtioblockdevice.image_path=${DISK_DIR} \
  -C bp.pl011_uart0.out_file=uart0.log \
  -C bp.pl011_uart1.out_file=uart1.log \
  -C bp.pl011_uart2.out_file=uart2.log \
  -C bp.pl011_uart3.out_file=uart3.log
```

After reading above instruction, please finish following tasks and questions.

Task 1.a: Please provide screenshots demonstrating that you have successfully booted the Linux Kernel on FVP. (20%)

Task 1.b: What are the exception level and the security state of RMM? (10%)

Task 1.c: Review the source code of 'linux-cca', identify at least two instances where RMIs are dispatched to RMM, and explain the functionality of these RMIs. (20%) (The definitions of RMIs can be found in the source code of 'kvm__realm__rmi.h'.)

3.2 Task 2: Launch Realm VM

To launch Realm VM, we also need a user-level software to interact with system software. In this lab, we choose kvmtool, a lightweight and simple tool for managing and running VMs using the interfaces provided by KVM. You can use following command to compile kvmtool.

```
$ export CROSS_COMPILE=aarch64-linux-gnu-

$ cd kvmtool-cca
$ make CROSS_COMPILE=$CROSS_COMPILE ARCH=arm64 clean
$ make lkvm-static CROSS_COMPILE=$CROSS_COMPILE ARCH=arm64 -j16
```

If you meet some problems during compilation, you can refer to the official document of kvmtool: https://github.com/clearlinux/kvmtool/blob/master/INSTALL

For example, you may need to install libdtc:arm64 to solve the dependency problem.

After compilation, you should copy the kvmtool binary to the FVP and lauch it again. You can use mount command to mount rootfs.ext4 to a specific path, and then copy the kvmtool binary to rootfs.ext4.

For example:

```
$ mkdir /mnt/rootfs
$ mount -o loop rootfs.ext4 /mnt/rootfs
$ cp lkvm-static /mnt/rootfs/test
```

In FVP, the rootfs is mounted as read-only, therefore, we need to remount it as read-write:

```
$ ./init.sh
```

Then, you can use following command to launch a Realm VM:

```
$ cd test
$ ./lkvm-static run --realm             \
  --measurement-algo=["sha256", "sha512"] \
  --disable-sve             \
  --disk test_vm.img --kernel Image
```

We have prepared a rootfs and guest kernel image in '/text' directory.

After reading above instruction, please finish following tasks and questions.

Task 2.a: Provide screenshots to show that you have already compiled kvmtool and copy it to FVP. (20%)

Task 2.b: Provide screenshots to show that you have already launched a Realm VM. (10%)

Task 2.c: Write a simpe application to print your Student ID and name, and execute it in the Realm VM. (10%) (You need to compile a static-linked binary and copy it to the Realm VM. You can use mount command to mount test_vm.img to a specific path as what you did for rootfs.ext4.)

Task 2.d: What is the exception level and the security state of kvmtool? (10%)