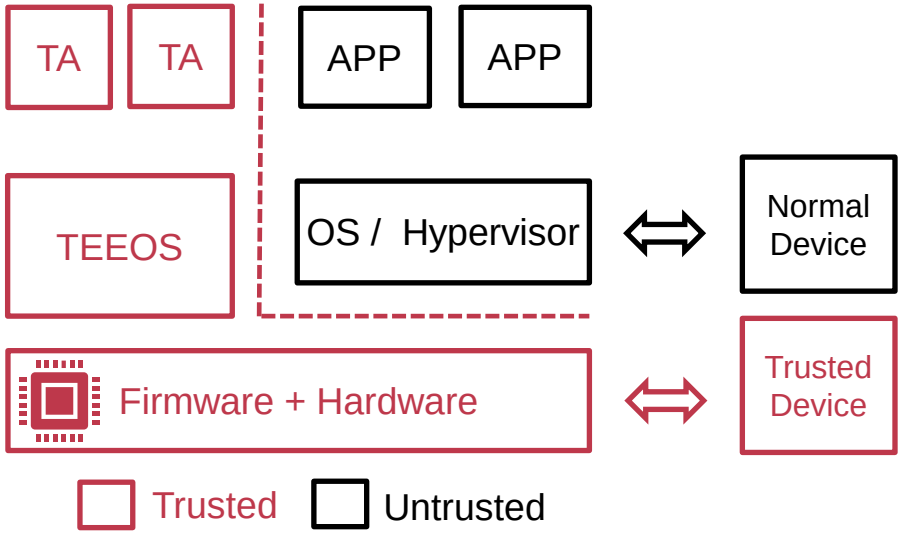
Openharmony-TEE

Erhu Feng

OH RISC-V SIG, Shanghai Jiao Tong University

2024.04.27

Trusted Execution Environment (TEE)



1. **TEE protects trusted app from untrusted software**
   - Hypervisor / OS
   - Other applications
2. **TEE contains secure hardware resources**
   - Secure CPU
   - Protected memory
   - Trusted Devices

- Intel SGX, TDX
- AMD SEV
- ARM TrustZone, CCA
- Penglai, Keystone

TEE is widely used in the mobile system

- **TEE protects the sensitive data and code for both users and developers**

Digital payment

Face recognition

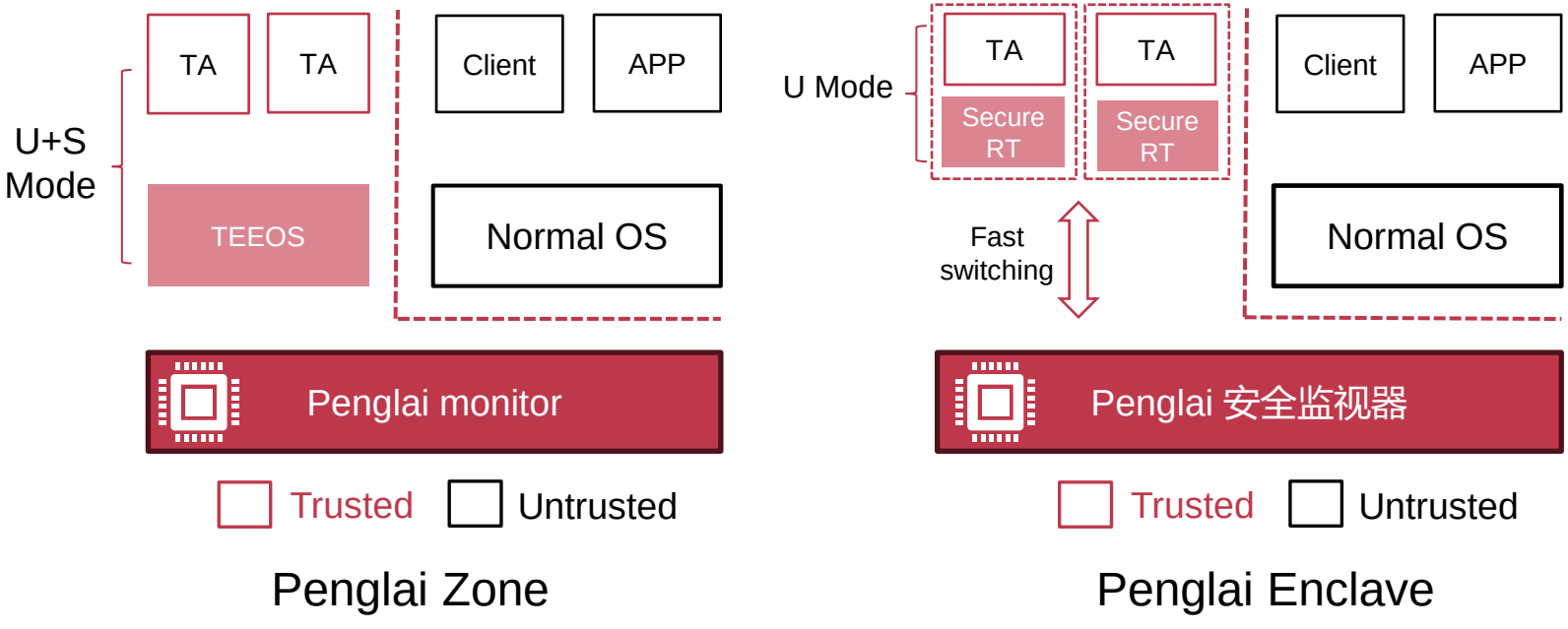
Digital Right Management

TEE in OpenHarmony

- **Provide a unified TEE architecture for both Arm and RISC-V**
  - Arm: TrustZone with OP-TEE OS
  - RISC-V: Penglai with OP-TEE OS / GP Runtime
- **Benefit: OpenHarmony+Penglai+RISC-V**
  - Open-sourced projects for both hardware and software stacks
  - Research platforms for OS, architecture and security
  - Easy to port the trusted applications from Arm ecology

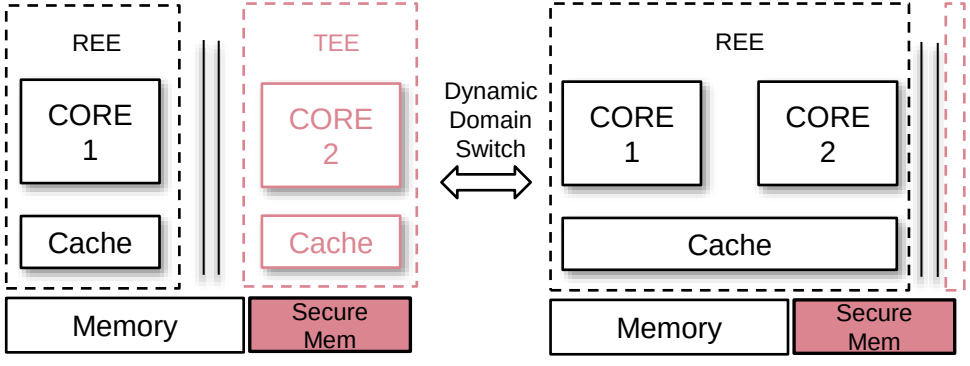
Penglai Architecture

- Provide two TEE abstractions：Enclave (U mode), Zone (U+S mode)
- Suitable for difference scenarios（Standard device and IoT）



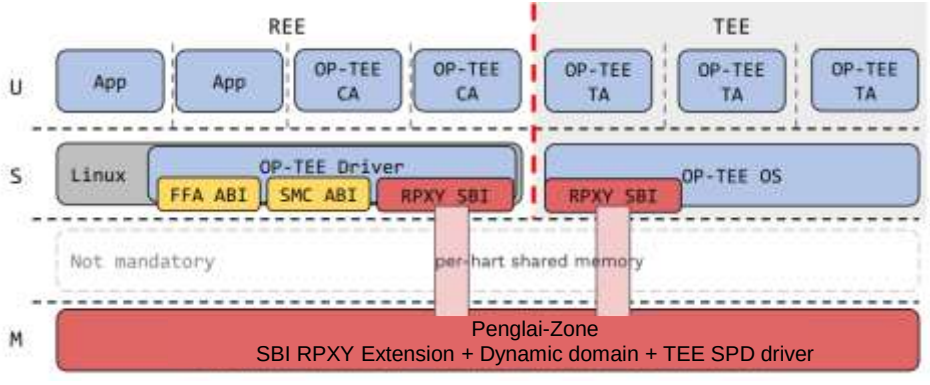
1. Penglai-Zone architecture

- **Underlying mechanizes**
  - Provide the TEE model which is similar to the TrustZone: TEE (Trusted execution environment) and REE (Rich execution environment)
  - Strong isolation between CPU, memory and I/O device
    - A presentation in Main program: Session 10D - sIOPMP
  - Dynamic domain switch between REE and TEE



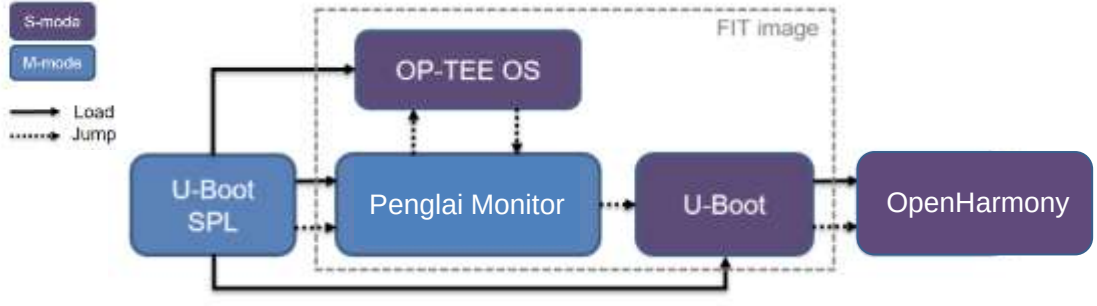
Penglai-Zone architecture

- **Components**
  - (secure) Penglai Monitor: Running as secure firmware in the M mode
  - (secure) OP-TEE OS: Trusted TEE OS running in the secure S mode
  - (Non-secure) OP-TEE Driver: Linux kernel driver installed in the REE
  - (secure) OP-TEE TA: Trusted application running in the TEE
  - (Non-secure) OP-TEE CA: Client application running in the REE



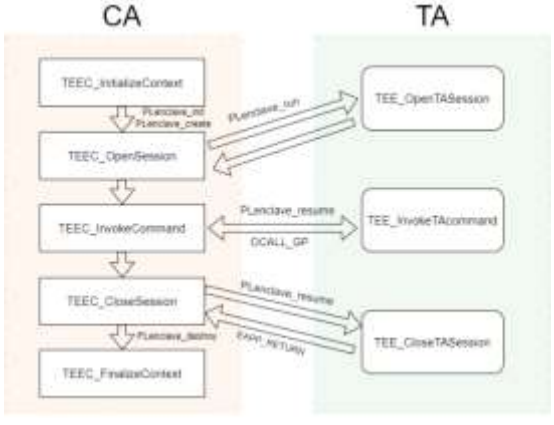
Penglai-Zone architecture

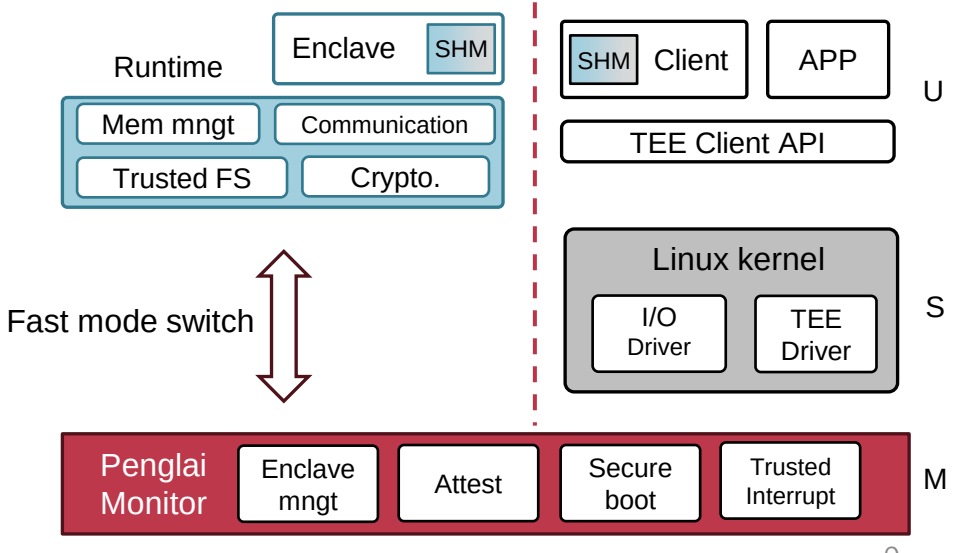
- **Secure Boot flow**
  - U-boot SPL loads and verifies the Penglai Monitor
  - Penglai monitor verifies the OP-TEE OS, and jumps to the OPTEE OS in the secure domain for initialization
  - After returning from OPTEE-OS, Penglai monitor jumps to the non-secure domain for loading U-Boot and OpenHarmony



2. Penglai-Enclave Architecture

- **Provide a more lightweight TEE abstraction: Enclave (U mode)**
  - Support various enclave runtimes
  - Automatically generate ecall/ocall function
  - TLS / Trusted FS supported
- **GP-based programming**





Distributed TEE in OpenHarmony

- **Offload the TEE task to a remote device**
  - Not all devices have the TEE support (IoT, legacy device)
  - Distributed TEE allows developers to offload TEE tasks to a remote TEE-enabled device
- **Aggregate the TEE hardware resource**
  - Different devices have the different TEE resources
  - Distributed TEE can aggregate all TEE resources to provide a unified TEE abstraction
- **Developer agnostic**
  - The developer does not need to care whether the underlying hardware supports TEE or not

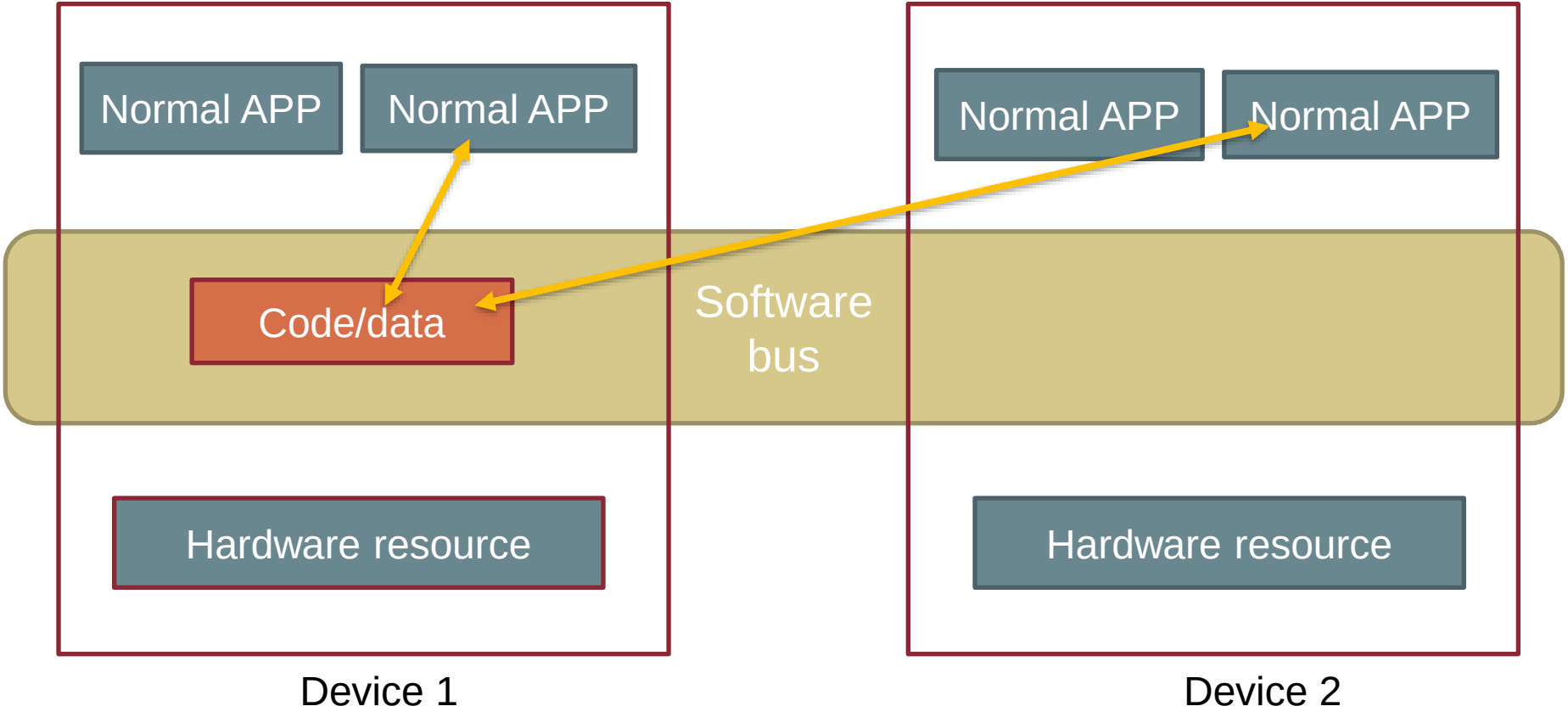
Demo1: Smart door lock with face recognition

- **Re-use the camera in the mobile phone**

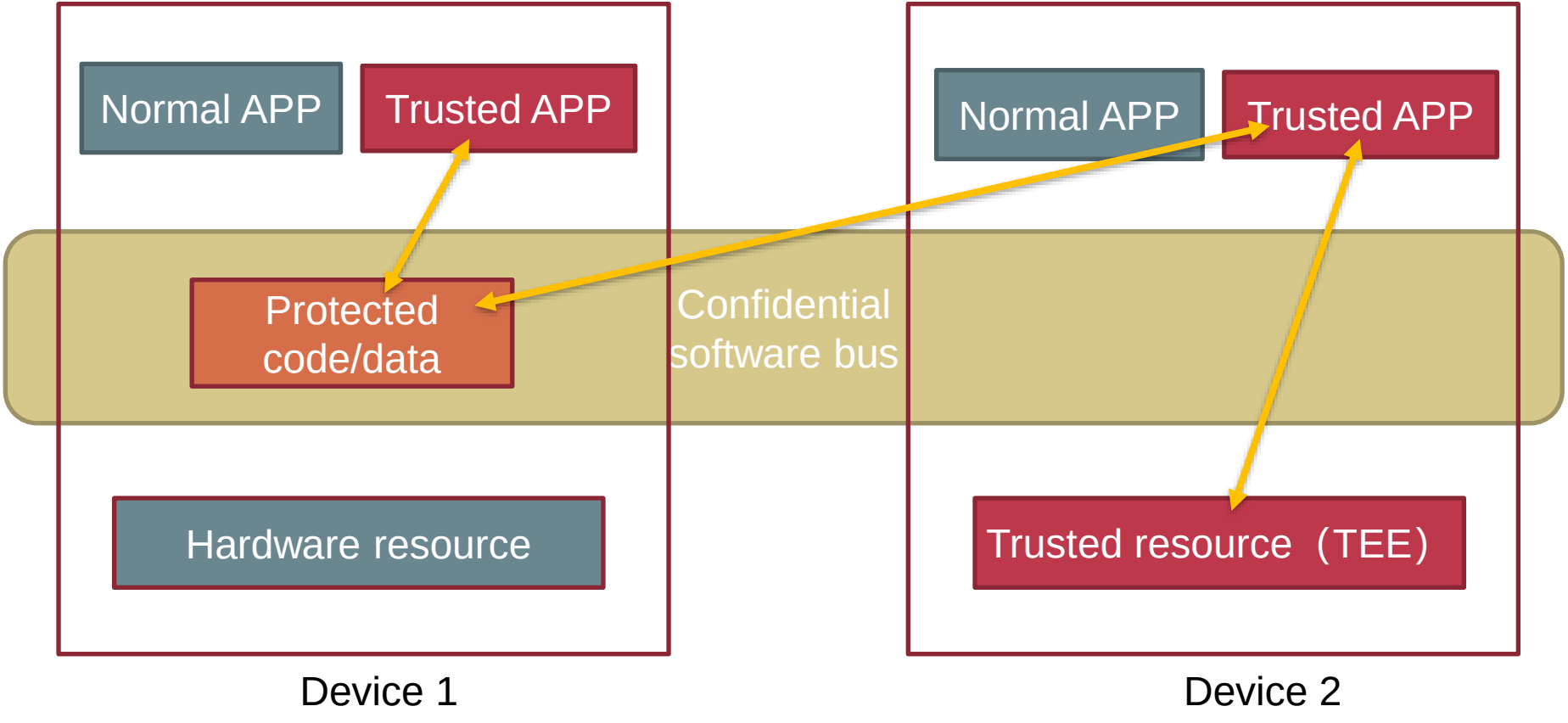
Demo2: Smart watch for personal health analysis

- **Send the personal health data to the TEE in mobile phone**

Distributed APP in OpenHarmony

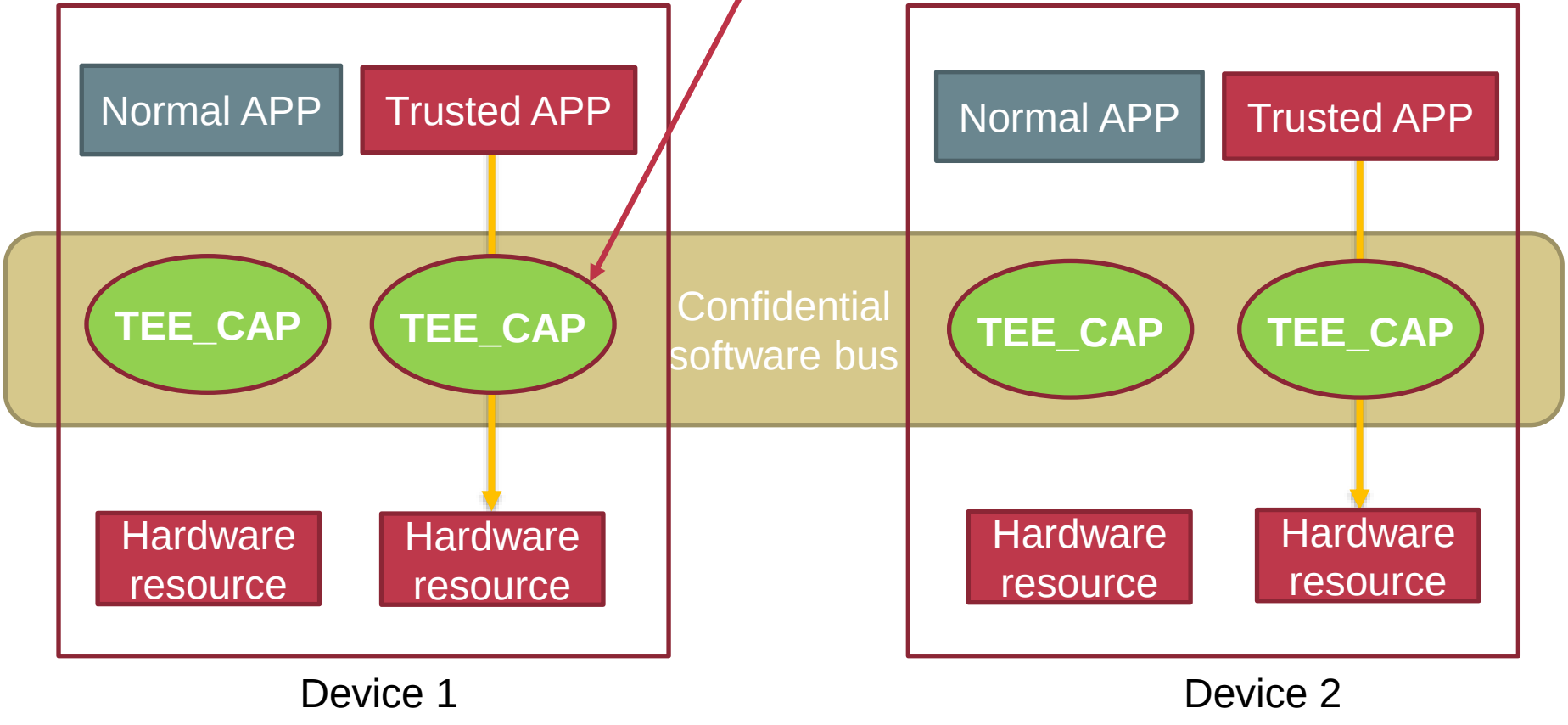


Distributed TEE design in OpenHarmony



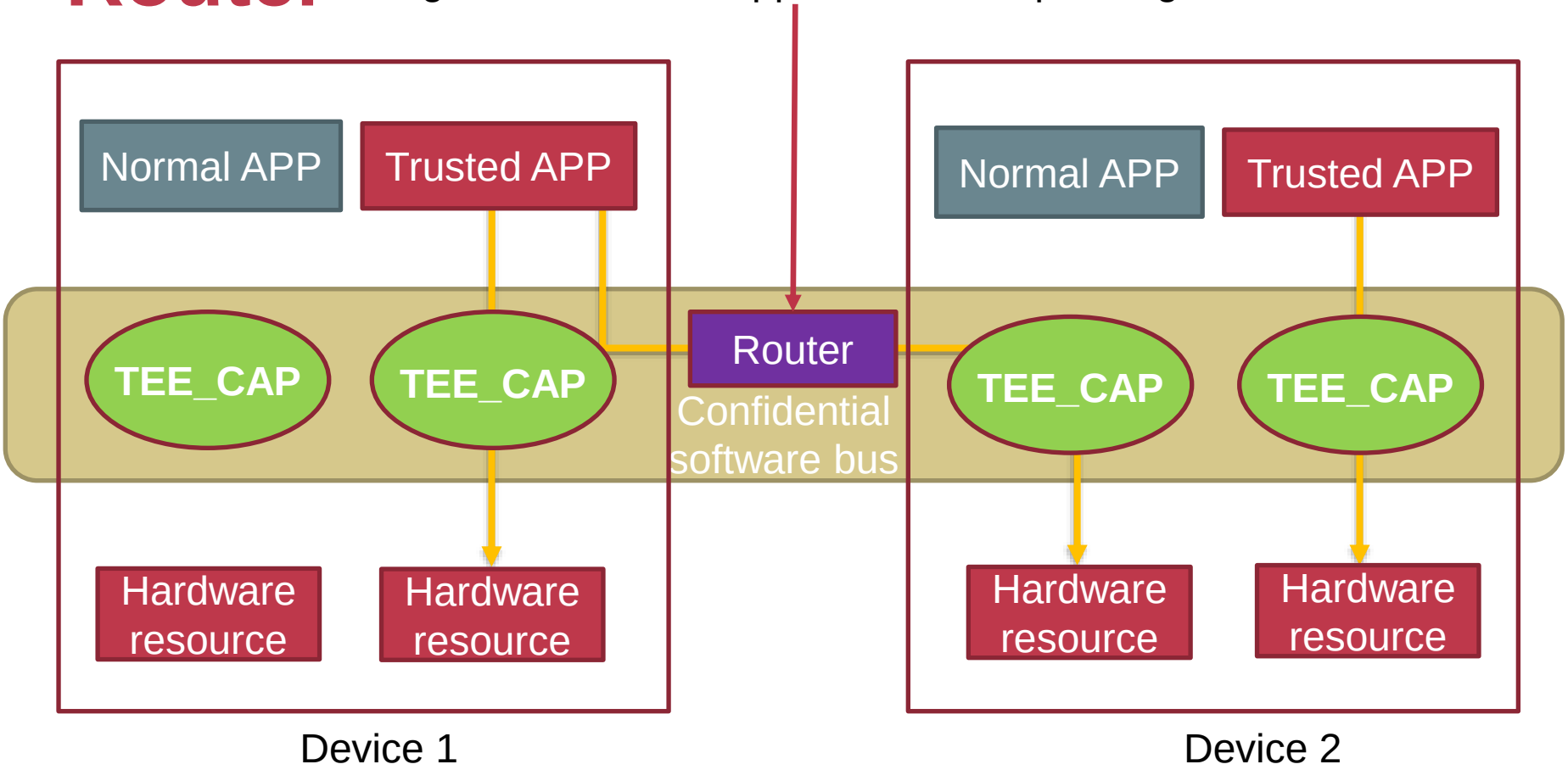
TEE Capability

Dynamic and fine-grained management for TEE resources (resource splitting and aggregation)



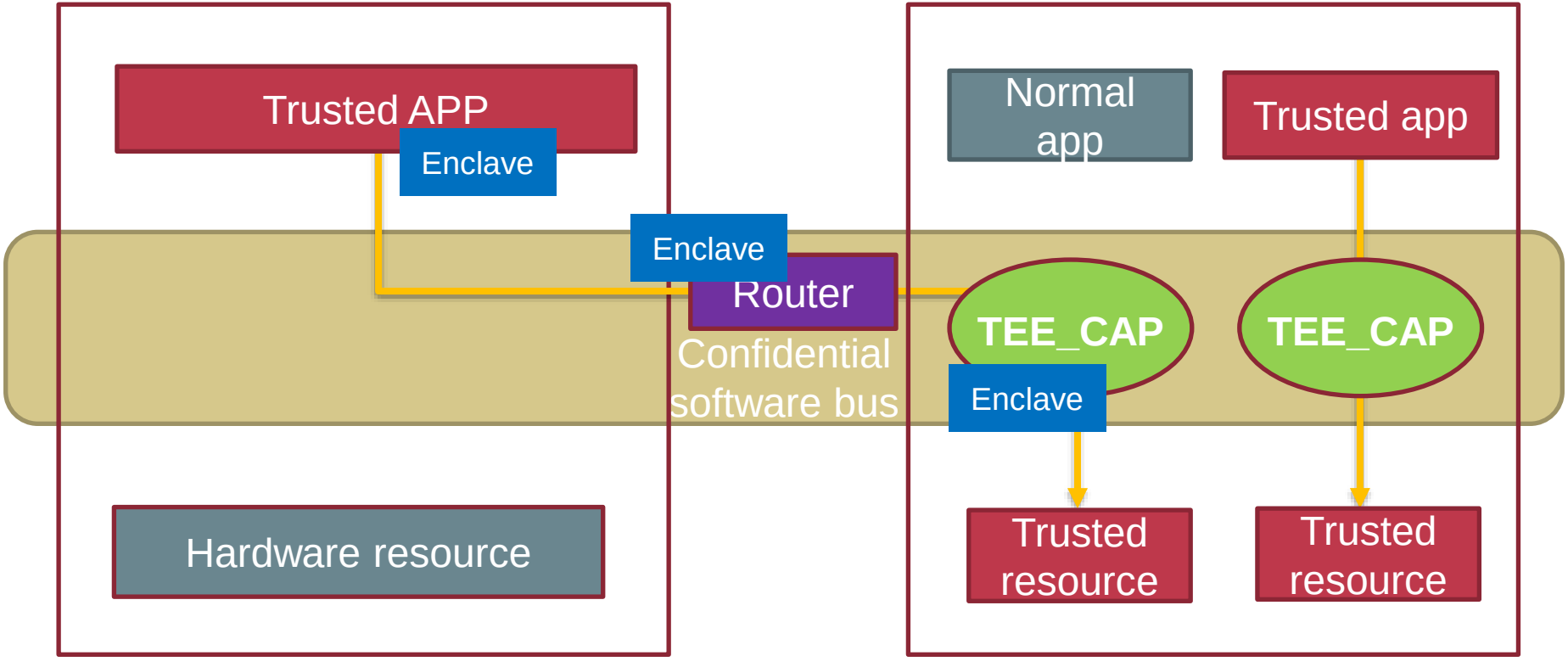
Router

Automatically select the idle TEE resource, and migrate the trusted app to the corresponding device



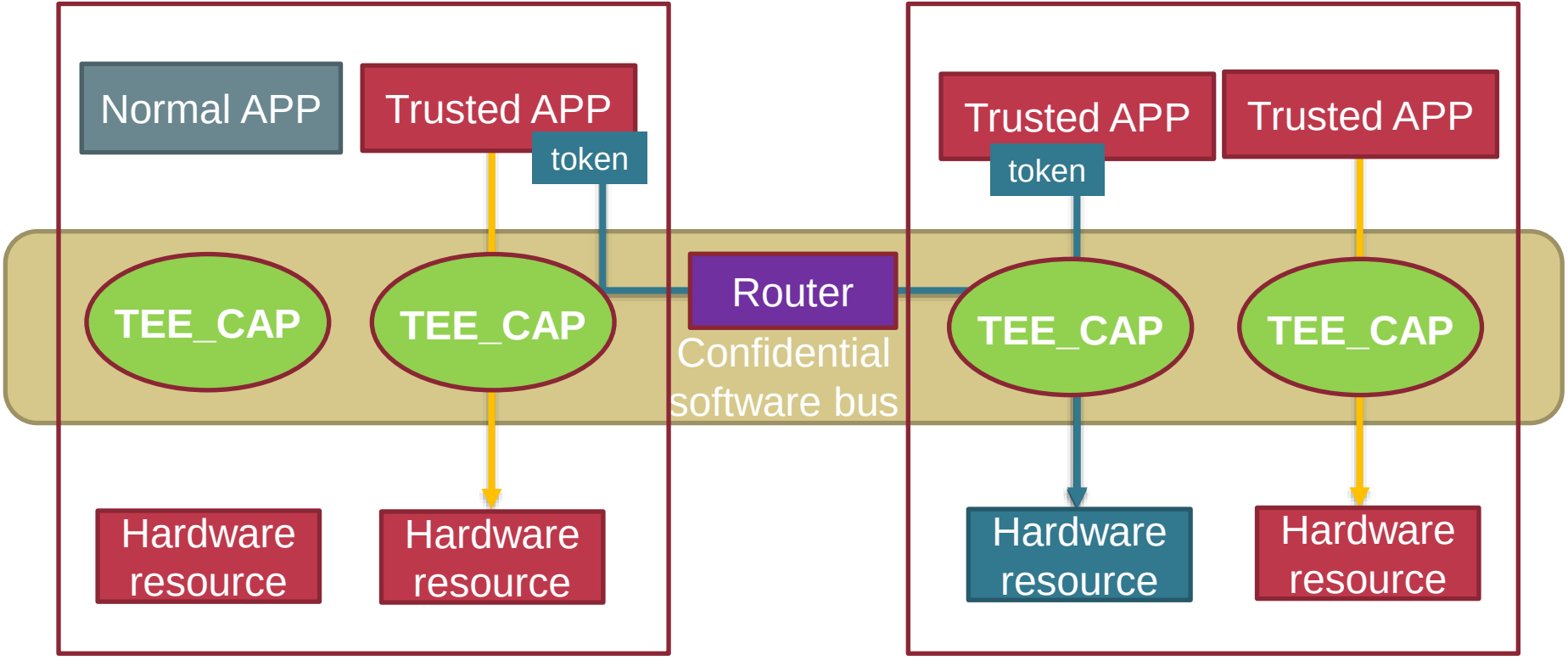
Use case 1: offload the trusted app

- **Deploy the trusted app to a remote device with TEE**



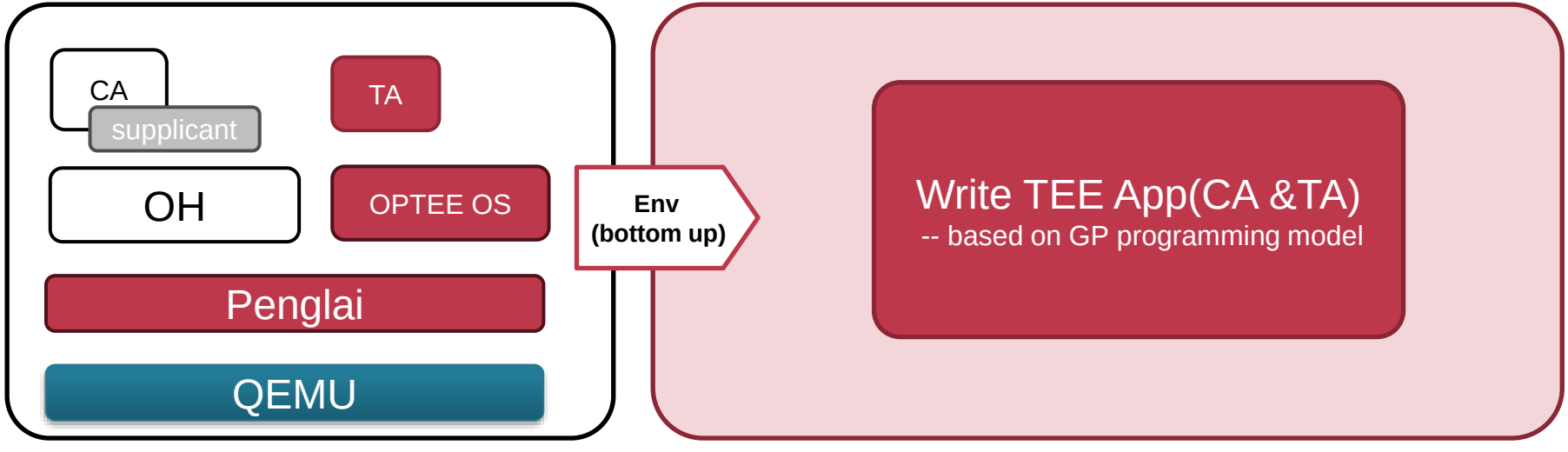
Use case 2: Sharing the TEE resource

- **Trusted app can share the same TEE_cap with token**



In tutorial 1, we will

- Prepare the OpenHarmony development environment for you
- Prepare the Penglai-Zone TEE development environment and go through the development workflows with you



How to run Penglai-Zone in OpenHarmony

- **Download Penglai-Zone project and OH image**

```
export WORKDIR=`pwd`
git clone https://github.com/openharmony-research/test_polyos_with_optee.git
wget -O images.zip https://ipads.se.sjtu.edu.cn:1313/f/e8b6021f2d674cbab710/?dl=1
unzip images.zip ## sudo apt install –y unzip
```

- **Prepare Device Tree Blob**

```
sudo apt-get update -y
sudo apt-get install -y device-tree-compiler

dtc -I dts -O dtb -o qemu-virt-new.dtb test_polyos_with_optee/qemu-virt-restrict.dts
```

- **Prepare toolchain**

```
cd $WORKDIR/test_polyos_with_optee && mkdir -p toolchain && cd toolchain/
wget https://github.com/riscv-collab/riscv-gnu-toolchain/releases/download/2023.07.07/riscv64-glibc-ubuntu-20.04-gcc-nightly-2023.07.07-nightly.tar.gz
tar zxvf riscv64-glibc-ubuntu-20.04-gcc-nightly-2023.07.07-nightly.tar.gz
export TOOLCHAIN=$WORKDIR/test_polyos_with_optee/toolchain
```

How to run Penglai-Zone in OpenHarmony

- **Compile Penglai-Zone monitor**

```
cd $WORKDIR/test_polyos_with_optee
git clone https://github.com/Penglai-Enclave/opensbi.git -b dev-rpxy-optee-v3
cd opensbi
# sudo apt install –y build-essential
CROSS_COMPILE=$TOOLCHAIN/riscv/bin/riscv64-unknown-linux-gnu- make PLATFORM=generic
cp build/platform/generic/firmware/fw_dynamic.elf $WORKDIR
```

- **Compile OPTEE OS, Client and examples**

```
sudo apt install -y pkg-config
sudo apt install -y uuid-dev
Sudo apt install -y python3-pyelftools
sudo apt install -y cmake
cd $WORKDIR/test_polyos_with_optee
./scripts/build_optee.sh              # for easily compilation all together
```

How to run Penglai-Zone in OpenHarmony

- **Copy CA, TA and startup script to OH image**

```
cd $WORKDIR
mkdir -p mnt
sudo mount images/system.img ./mnt

sudo cp -rf test_polyos_with_optee/optee_client/build/out/export/usr/sbin/tee-supplicant ./mnt/system/bin/

sudo mkdir -p ./mnt/system/lib/optee_armtz
sudo cp test_polyos_with_optee/optee_examples/hello_world/ta/8aaaf200-2450-11e4-abe2-0002a5d5c51b.ta ./mnt/system/lib/optee_armtz/
sudo cp test_polyos_with_optee/optee_examples/hello_world/host/optee_example_hello_world ./mnt/system/bin/

sudo umount ./mnt
```

```
cd $WORKDIR
sudo mount -o loop images/userdata.img ./mnt

cat > mnt/start_optee_supplicant.sh << EOF
if [ -e /bin/tee-supplicant -a -e /dev/teepriv0 ]; then
        echo "Starting tee-supplicant..."
        tee-supplicant&
        ifconfig lo up
        exit 0
else
        echo "tee-supplicant or TEE device not found"
        exit 1
fi
;;
EOF

sudo chmod a+x mnt/start_optee_supplicant.sh
sudo umount ./mnt
```

How to run Penglai-Zone in OpenHarmony

- **Run the openHarmony with Penglai and optee**

```
cd $WORKDIR
./test_polyos_with_optee/run_polyos.sh
```

- **After Login, execute**

```
cd data
./start_optee_supplicant.sh
optee_example_hello_world
```

```
D/TC:? 0 ldelf_syscall_open_bin:167 res=0
D/TC:? 0 read_fat:2140 fat_address 0
D/TC:? 0 tee_rpmb_read:1251 Read 8 blocks at index 2
D/TC:? 0 dump_fh:1885 fh->filename=/dirfile.db.hash
D/TC:? 0 dump_fh:1886 fh->rpmb_fat_address=512
D/TC:? 0 dump_fh:1887 fh->fat_entry.start_address=0
D/TC:? 0 dump_fh:1888 fh->fat_entry.data_size=0
D/TC:? 0 read_fat:2140 fat_address 512
D/TC:? 0 tee_rpmb_read:1251 Read 8 blocks at index 2
D/LD:  ldelf:176 ELF (8aaaf200-2450-11e4-abe2-0002a5d5c51b) at 0x40077000
D/TA:  TA_CreateEntryPoint:39 has been called
D/TA:  __GP11_TA_OpenSessionEntryPoint:68 has been called
I/TA: Hello World!
Invoking TA to increment 42
D/TA:  inc_value:105 has been called
I/TA: Got value: 42 from NW
I/TA: Increase value to: 43
TA incremented value to 43
D/TC:? 0 tee_ta_close_session:460 csess 0xf0c61f60 id 1
D/TC:? 0 tee_ta_close_session:479 Destroy session
I/TA: Goodbye!
D/TA:  TA_DestroyEntryPoint:50 has been called
D/TC:? 0 destroy_context:318 Destroy TA ctx (0xf0c61ef8)
```

HelloWorld example

- CA side logic

```
/*
 * Prepare the argument. Pass a value in the first parameter,
 * the remaining three parameters are unused.
 */
op.paramTypes = TEEC_PARAM_TYPES(TEEC_VALUE_INOUT, TEEC_NONE,
                 TEEC_NONE, TEEC_NONE);
op.params[0].value.a = 42;

/*
 * TA_HELLO_WORLD_CMD_INC_VALUE is the actual function in the TA to be
 * called.
 */
printf("Invoking TA to increment %d\n", op.params[0].value.a);
res = TEEC_InvokeCommand(&sess, TA_HELLO_WORLD_CMD_INC_VALUE, &op,
           &err_origin);
if (res != TEEC_SUCCESS)
    errx(1, "TEEC_InvokeCommand failed with code 0x%x origin 0x%x",
         res, err_origin);
printf("TA incremented value to %d\n", op.params[0].value.a);
```

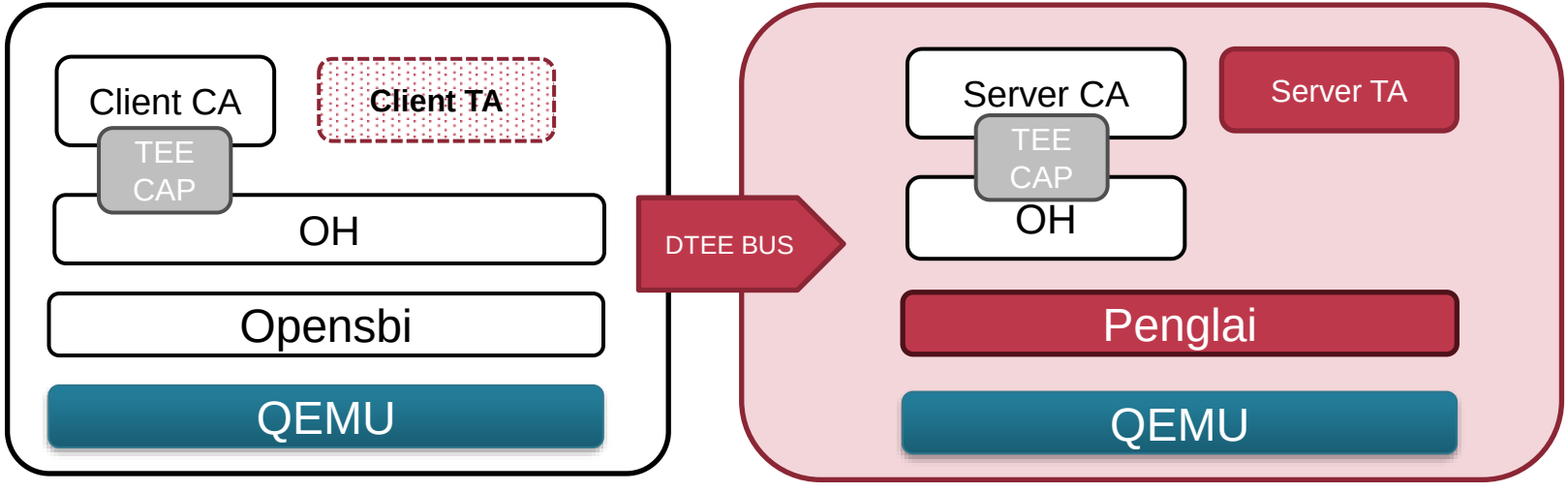
- TA side logic

```
/*
 * Called when a TA is invoked. sess_ctx hold that value that was
 * assigned by TA_OpenSessionEntryPoint(). The rest of the paramters
 * comes from normal world.
 */
TEE_Result TA_InvokeCommandEntryPoint(void __maybe_unused *sess_ctx,
            uint32_t cmd_id,
            uint32_t param_types, TEE_Param params[4])
{
    (void)&sess_ctx; /* Unused parameter */

    switch (cmd_id) {
    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;
    }
}
```

```
static TEE_Result inc_value(uint32_t param_types,
    TEE_Param params[4])
{
    ...
    params[0].value.a++;

    return TEE_SUCCESS;
}
```

In tutorial 2, we will

- Prepare the Distributed TEE development environment for you
- Offload the TEE application to another machine with TEE



How to run distributed TEE in OpenHarmony

- **Download the repo**

```
git clone https://github.com/openharmony-research/dteegen.git
cd dteegen
git submodule update --init --recursive
```

- **Download the prebuild OH image for distributed TEE**
  - It will take a few minutes

```
bash ./scripts/download_prebuilt.sh
export OH_HOME=`pwd`/polyos
export OH_IMAGES=$OH_HOME/out/riscv64_virt/packages/phone/images
```

How to run distributed TEE in OpenHarmony

- **Build Penglai monitor and driver**

```
cd Penglai-Enclave-sPMP
export PENGLAI_HOME=`pwd`

# build opensbi and download docker image
# sudo apt install docker.io
bash ./build_opensbi.sh (or sudo bash ./build_opensbi.sh )

# build the driver (optionally, we have prepared the Penglai driver in the OH image)
bash ./scripts/build_driver_for_oh.sh

cd ..
```

How to run distributed TEE in OpenHarmony

- **Create a quick demo**

```
# download dteegen tool
curl -o dteegen https://raw.githubusercontent.com/iku-iku-iku/dteegen/master/scripts/all_in_one.sh
chmod +x dteegen
sudo mv dteegen /usr/local/bin

# create new project
export PROJECT_NAME=new_project
export PROJECT_PATH=`pwd`/$PROJECT_NAME
dteegen create $PROJECT_NAME (or sudo dteegen create $PROJECT_NAME )
dteegen deploy $PROJECT_NAME (or sudo dteegen deploy $PROJECT_NAME )
```

How to run distributed TEE in OpenHarmony

- **Do some preparation for running OpenHarmony.**

```
# Copy opensbi to $OH_HOME
cp $PENGLAI_HOME/opensbi-1.2/build-oe/qemu-virt/platform/generic/firmware/fw_jump.bin $OH_HOME

# Copy scripts to $OH_HOME
cp $PENGLAI_HOME/scripts/start_server.sh $OH_HOME
cp $PENGLAI_HOME/scripts/start_client.sh $OH_HOME

export MOUNT_PATH=/tmp/mount (or sudo export MOUNT_PATH=/tmp/mount )
mkdir -p $MOUNT_PATH

# Inject depencies to OH images
./scripts/copy_penglai_dep.sh (or ./scripts/sudo_copy_penglai_dep.sh )

# Inject built files to OH images
./scripts/copy_penglai_app.sh

# Since instances can not share the same images, we need to copy them.
./scripts/create_images.sh
```

How to run distributed TEE in OpenHarmony

- **Create network bridge.**

```
sudo ip link add name br0 type bridge
sudo ip link set dev br0 up
sudo ip addr add 192.168.1.109/24 dev br0
sudo iptables -P FORWARD ACCEPT
```

How to run distributed TEE in OpenHarmony

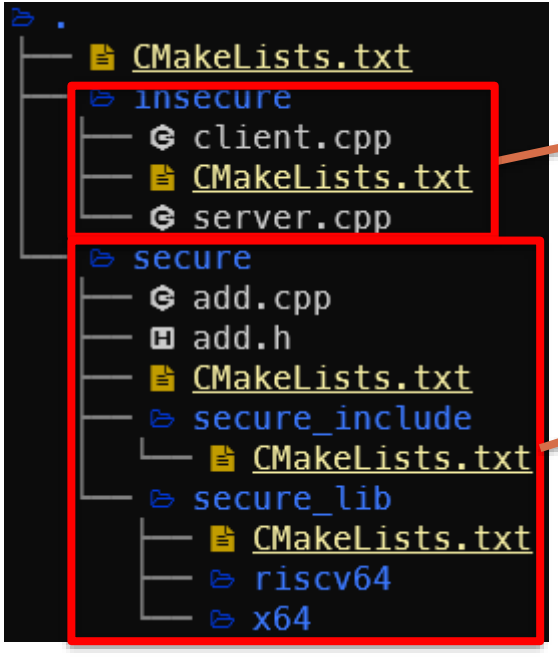
- **Run server and client.**

```
# run server in a machine with TEE
cd $OH_HOME
./start_server.sh

# in OH
cd data
insmod penglai.ko
./server

# waiting for log: Enter a number to stop
the server:
```

```
# run client in a machine without TEE

cd $OH_HOME
./start_client.sh
# in OH
cd data
./client
```

```
MATCHED 2
CALL SERVICE: _Z_task_handler
randomize_guid
BEGIN DUMMY WRITE
END DUMMY WRITE
WRITE (4538) ENCLAVE ID: 58
READ DONE: ENCLAVE_ID = 58
decrypting... origin_data_len = 16
RES: 0
mul(1, 2) == 2
HOOKED: FUNCTION ID: 1, INPUT BUFFER SIZE: 64, OUTPUT BUFFER SIZE: 16
sealed key len = 153
sealed key len = 153
CALL SERVICE: _Z_task_handler
randomize_guid
BEGIN DUMMY WRITE
END DUMMY WRITE
WRITE (4538) ENCLAVE ID: 58
READ DONE: ENCLAVE_ID = 58
decrypting... origin_data_len = 16
RES: 0
add(1, 2) == 3
```

How to develop distributed tee project

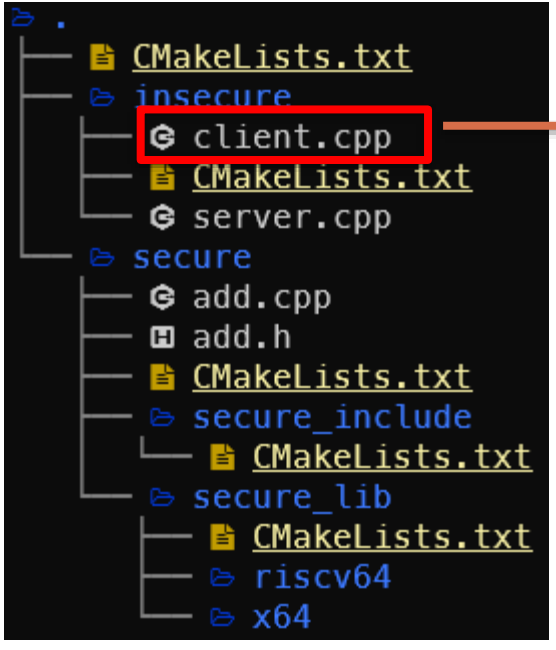


Write untrusted logic in the “insecure” folder

Write secure logic(run in TEE) in the “secure” folder

- Without distributed tee ability
- Debugging friendly
- Deployable with dteegen

How to develop distributed tee project




```cpp
#include "../secure/add.h"
#include "TEE-Capability/distributed_tee.h"

int main() {
  auto ctx = init_distributed_tee_context({.side = SIDE::Client,
                                           .mode = MODE::Transparent,
                                           .name = "template_client",
                                           .version = "1.0"});

  int res;
  int a = 1, b = 2;
  res = mul(a, b);
  printf("mul(%d, %d) == %d\n", a, b, res);
  res = add(a, b);
  printf("add(%d, %d) == %d\n", a, b, res);
  destroy_distributed_tee_context(ctx);
}
```

define and construct the context

call distributed tee func just like local func

destroy the context