



华中科技大学

可信计算实验报告

姓 名： 邬雪菲
学 院： 网络空间安全学院
专 业： 网络空间安全
班 级： 网安 2104 班
学 号： U202112131
指导教师： 代炜琦

分数	
教师签名	

2024 年 5 月 30 日

目 录

1、实验目的	- 1 -
2、实验内容以及流程说明	- 2 -
2.1 环境配置	- 2 -
2.2 编译	- 2 -
2.3 初始化	- 4 -
2.4 创建密钥层次（KeyHierarchy）	- 4 -
2.5 Seal、Unseal 和 extend	- 7 -
2.6 密钥迁移（KeyMigration）	- 11 -
2.7 远程证明（RemoteAttestation）	- 12 -
3、实验心得及建议	- 16 -

1、实验目的

本实验的目的是让学生将从书本中学到的可信计算相关知识应用到实践中。在 linux 中使用 `tmppm` 模拟器，通过 TSS 软件栈调用相关硬件来完成远程证明、密钥迁移、密钥结构、数据密封等相关功能，了解 TPM 的安全性，学会调用 TSS 的各种接口来完成应用程序。

本实验的任务主要是在随文档提供的代码的基础下，填补代码中缺失的部分，这个工作主要在密钥迁移和密钥结构相关功能代码中，还有根据功能需要，补全数据密封功能所需要的代码文件。

实验具体包括以下四个任务：创建密钥层次、Seal 和 Unseal 实验、密钥迁移实验、远程证明实验。

2、实验内容以及流程说明

2.1 环境配置

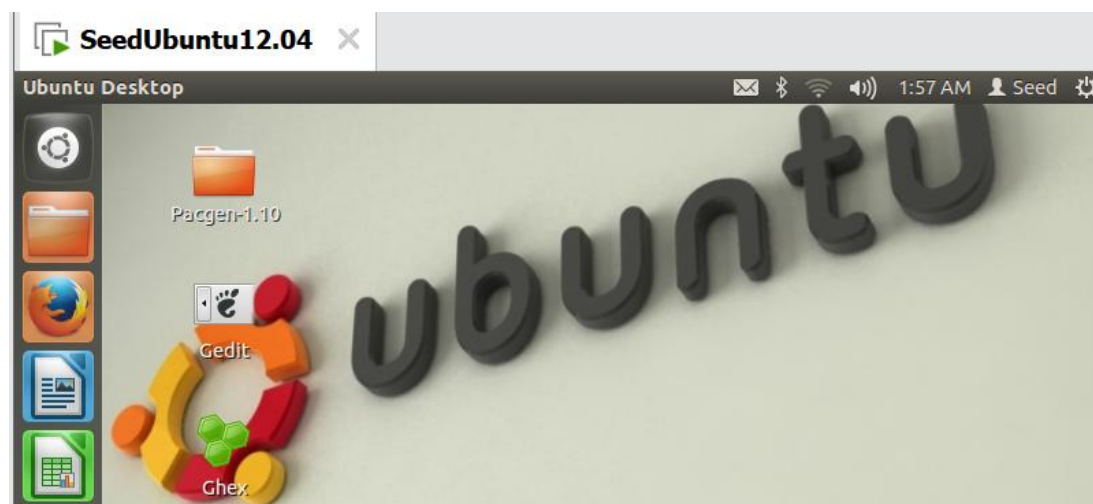
本实验在 linux 虚拟机上进行，因为没有物理 TPM，需要使用 TPM Emulator 完成 TPM 的相关实验。环境配置列表如下：

VMware Workstation 17 Pro SEEDUbuntu-16.04-32bit TPM Emulator Trousers

其中下载的虚拟机是 vmdk 文件，参考教程链接新建虚拟机：

<https://zhuanlan.zhihu.com/p/676485346>

安装完成后进入虚拟机的界面如下图：



2.2 编译

①首先需要换源。在线安装过程中某些国外网站没法访问导致安装命令无法执行，故需要换源，原实验手册中的参考链接给出的镜像源太久远无法使用，网络查找并更换了新的镜像源，参考链接如下：

https://blog.csdn.net/qq_44667165/article/details/108560272

注意，需要先把源文件内容删除再添加镜像源，否则更换源不成功。换源后执行 `sudo apt-get update`，可以看到执行成功如下图：

```
Get:88 http://mirrors.aliyun.com trusty-backports/multiverse Translation-en [1,215 B]
Get:89 http://mirrors.aliyun.com trusty-backports/restricted Translation-en [28 B]
Get:90 http://mirrors.aliyun.com trusty-backports/universe Translation-en [36.8 kB]
Fetched 26.0 MB in 35s (741 kB/s)
Reading package lists... Done
[05/23/2024 02:55] seed@ubuntu:~/Desktop$
```

②接下来解压并编译安装 TPM Emulator，执行如下命令：

```
tar xvfz tpm-emulator.tar.gz
cd tpm-emulator
sudo apt-get install libgmp-dev cmake
./build.sh
cd build
sudo make install
sudo depmod -a
```

安装过程截图如下所示，可见安装成功：

```
Setting up libgmp10 (2:5.1.3+dfsg-1ubuntu1) ...
Setting up libgmpxx4ldbl (2:5.1.3+dfsg-1ubuntu1) ...
Setting up cmake-data (2.8.12.2-0ubuntu3) ...
Setting up cmake (2.8.12.2-0ubuntu3) ...
Setting up libgmp-dev (2:5.1.3+dfsg-1ubuntu1) ...
Processing triggers for libc-bin ...
ldconfig deferred processing now taking place

[100%] Building C object tpm/unix/CMakeFiles/tpmd.dir/tpmd.o
Linking C executable tpmd
[100%] Built target tpmd
[05/23/2024 02:59] seed@ubuntu:~/Desktop/tpm-emulator$ cd build
[05/23/2024 03:00] seed@ubuntu:~/Desktop/tpm-emulator/build$ sudo make install
[ 58%] Built target tpm
[ 80%] Built target mtm
[ 90%] Built target tpm_crypto
[ 92%] Built target tddl
[ 94%] Built target tddl_static
[ 96%] Built target test_tddl
[ 98%] Built target tpmd_dev
[100%] Built target tpmd
Install the project...
-- Install configuration: ""
-- Installing: /usr/local/lib/libtddl.so.1.2.0.7
-- Installing: /usr/local/lib/libtddl.so.1.2
-- Installing: /usr/local/lib/libtddl.a
-- Installing: /usr/local/lib/libtddl.a
-- Installing: /usr/local/include/tddl.h
-- Installing: /usr/local/bin/tpmd
-- Removed runtime path from "/usr/local/bin/tpmd"
[05/23/2024 03:00] seed@ubuntu:~/Desktop/tpm-emulator/build$ sudo depmod -a
```

③下一步是安装 TSS 软件栈：

```
sudo apt-get install libtspi-dev trousers
```

④最后将实验源码在 Windows 下解压然后拷贝到虚拟机中，编译

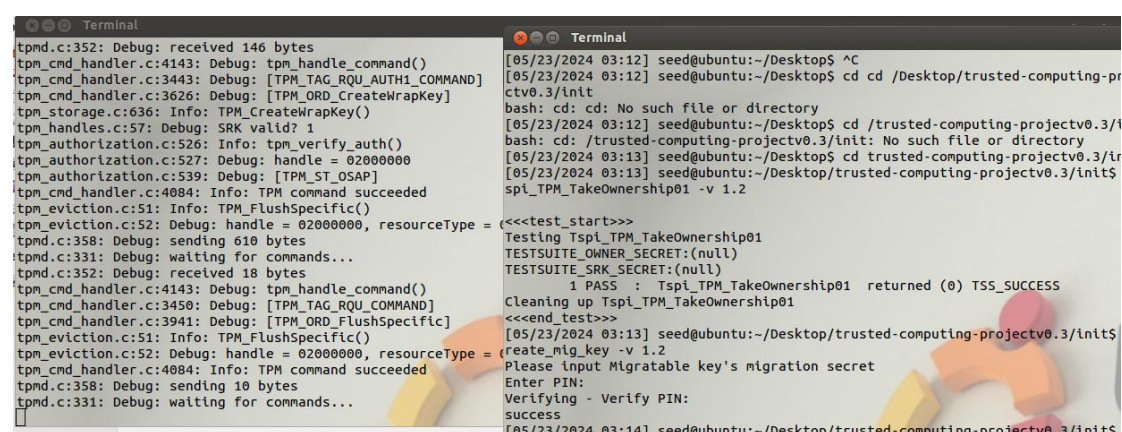
```
cd trusted-computing-projectv0.3
make clean
make
```

2.3 初始化

正式开始实验任务时首先需要进行环境初始化操作，具体执行命令和相关说明如下：

```
sudo modprobe tpmdev #如果遇到报错，先执行 sudo depmod -a
sudo tpmdev -f -d clear
#再打开一个终端(注意：前面 tpmdev 那个终端不要关)，运行
sudo tcscd
# 确保 TPM 没有被 TakeOwnership，否则会出错
#进入 init 目录，运行
cd ~/Desktop/trusted-computing-projectv0.3/init
./Tspi_TPM_TakeOwnership01 -v 1.2
#运行 输入的 pin: 123456
./create_mig_key -v 1.2
```

运行上述命令，如果所有安装流程正常，可以看到左侧 tpmdev 启动成功，右侧新终端初始化成功。

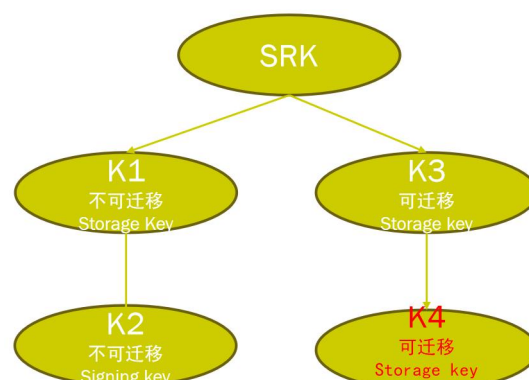


```
tpmdev.c:352: Debug: received 146 bytes
tpmdev_handler.c:4143: Debug: tpmdev_handle_command()
tpmdev_handler.c:3443: Debug: [TPM_TAG_RQU_AUTH1_COMMAND]
tpmdev_handler.c:3626: Debug: [TPM_ORD_CreateWrapKey]
tpmdev_storage.c:636: Info: TPM_CreateWrapKey()
tpmdev_handles.c:57: Debug: SRK valid? 1
tpmdev_authorization.c:526: Info: tpmdev_verify_auth()
tpmdev_authorization.c:527: Debug: handle = 02000000
tpmdev_authorization.c:539: Debug: [TPM_ST_OSAP]
tpmdev_handler.c:4084: Info: TPM command succeeded
tpmdev_eviction.c:51: Info: TPM_FlushSpecific()
tpmdev_eviction.c:52: Debug: handle = 02000000, resourceType =
tpmdev.c:358: Debug: sending 610 bytes
tpmdev.c:331: Debug: waiting for commands...
tpmdev.c:352: Debug: received 18 bytes
tpmdev_handler.c:4143: Debug: tpmdev_handle_command()
tpmdev_handler.c:3450: Debug: [TPM_TAG_RQU_COMMAND]
tpmdev_handler.c:3941: Debug: [TPM_ORD_FlushSpecific]
tpmdev_eviction.c:51: Info: TPM_FlushSpecific()
tpmdev_eviction.c:52: Debug: handle = 02000000, resourceType =
tpmdev_handler.c:4084: Info: TPM command succeeded
tpmdev.c:358: Debug: sending 10 bytes
tpmdev.c:331: Debug: waiting for commands...
```

```
[05/23/2024 03:12] seedubuntu:~/Desktop$ ^C
[05/23/2024 03:12] seedubuntu:~/Desktop$ cd ~/Desktop/trusted-computing-projectv0.3/init
bash: cd: No such file or directory
[05/23/2024 03:12] seedubuntu:~/Desktop$ cd ~/Desktop/trusted-computing-projectv0.3/init
bash: cd: /trusted-computing-projectv0.3/init: No such file or directory
[05/23/2024 03:13] seedubuntu:~/Desktop$ cd ~/Desktop/trusted-computing-projectv0.3/init
[05/23/2024 03:13] seedubuntu:~/Desktop/trusted-computing-projectv0.3/init$ ./Tspi_TPM_TakeOwnership01 -v 1.2
<<<test_start>>>
Testing Tspi_TPM_TakeOwnership01
TESTSUITE_OWNER_SECRET:(null)
TESTSUITE_SRK_SECRET:(null)
1 PASS : Tspi_TPM_TakeOwnership01 returned (0) TSS_SUCCESS
Cleaning up Tspi_TPM_TakeOwnership01
<<<end_test>>>
[05/23/2024 03:13] seedubuntu:~/Desktop/trusted-computing-projectv0.3/init$ ./create_mig_key -v 1.2
Please input Migratable key's migration secret
Enter PIN:
Verifying - Verify PIN:
success
[05/23/2024 03:14] seedubuntu:~/Desktop/trusted-computing-projectv0.3/init$
```

2.4 创建密钥层次（KeyHierarchy）

本次实验需要创建如下图所示的密钥层次。



首先进入 Key hierarchy 目录

```
cd ~/Desktop/trusted-computing-projectv0.3/KeyHierarchy
```

①参考 K1、K2、K3 的创建过程以及 TSS 文档，可以完善 create_register_key.c 中创建 K4 的代码，只需要在前面的代码基础上修改部分参数即可。具体来说，先设置初始化密钥标志，然后调用 my_create_load_key 函数创建 k4，如下所示：

```
// K4 , migratable , parent key is K3
// TODO:
// 这部分由同学们来完成
//
printf("Create UserK4 and register it to disk.\n");
////////////////////////////////////
initFlags = TSS_KEY_TYPE_STORAGE | TSS_KEY_SIZE_2048 | TSS_KEY_VOLATILE | TSS_KEY_AUTHORIZATION | TSS_KEY_MIGRATABLE;
result = my_create_load_key(hContext,initFlags,hKey3,&hKey4,"K4");
if(result!=TSS_SUCCESS){
    print_error("create_key",result);
    Tspi_Context_FreeMemory(hContext,NULL);
    Tspi_Context_Close(hContext);
    exit(result);
}
result=Tspi_Context_RegisterKey(hContext,hKey4,TSS_PS_TYPE_SYSTEM,UUID_K4,TSS_PS_TYPE_SYSTEM,UUID_K3);
if(result!=TSS_SUCCESS){
    print_error("Tspi_Context_RegisterKey",result);
    Tspi_Context_FreeMemory(hContext,NULL);
    Tspi_Context_Close(hContext);
    exit(result);
}
////////////////////////////////////
printf("Create and register K4 succeeded!\n");
```

然后利用已经完成的 Makefile 文件可以一键完成编译生成对应的可执行程序，然后运行程序完成密钥层次的建立。

```
sudo make
./create_register_key -v 1.2
```

执行程序后进入创建密钥过程，可以看到打印出来的密钥层次。之后为每个密钥设置密码。执行过程如下两图所示：

```
chy$ ./create_register_key -v 1.2

<<<test_start>>>
Testing Create KEY
KEY structure:
SRK
|__UserK1(Storage key, unmigratable)
|  |__UserK2(Signing key, unmigratable)
|
|__UserK3(Storage key, migratable)
   |__UserK4(Bind key, migratable)

Create UserK1 and register it to disk.
Input K1's Usage Pin
Enter PIN:
Verifying - Verify PIN:
Create and register K1 succeeded!
Create UserK2 and register it to disk.
Input K2's Usage Pin
Enter PIN:
Verifying - Verify PIN:
12Create and register K2 succeeded!
Create UserK3 and register it to disk.
Input K3's Usage Pin
Enter PIN:
Verifying - Verify PIN:
^^migratable
```

```

^^migratable
K3's Migration Pin
Enter PIN:
Verifying - Verify PIN:
Create and register K3 succeeded!
Create UserK4 and register it to disk.
Input K4's Usage Pin
Enter PIN:
Verifying - Verify PIN:
^^migratable
K4's Migration Pin
Enter PIN:
Verifying - Verify PIN:
Create and register K4 succeeded!
1 PASS : Create KEY returned (0) TSS_SUCCESS
Cleaning up Create KEY
<<<end_test>>>
[05/23/2024 03:21] seed@ubuntu:~/Desktop/trusted-computing-projectv0.3/KeyHierarchy
hvs

```

该程序的作用是在加载 SRK 后，创建并加载四个 register UserKey，将创建的密钥层次结构注册到系统中的永久存储区。其中 UserK1 是不可迁移的 Storage Key, UserK2 是不可迁移的 Signing Key, UserK3 是可迁移的 Storage Key, UserK4 是可迁移的 Bind Key。

②参考 K1、K2、K3 的加载过程以及 TSS 文档，完善 load_key.c 中加载 K4 的代码，具体如下所示：

gedit ./load_key.c

```

// ----- load k4 -----
printf("Loading K4...\n");
// TODO:
// 这部分代码由同学们自己完成
////////////////////////////////////
result = Tspi_Context_GetKeyByUUID(hContext, TSS_PS_TYPE_SYSTEM, UUID_K4, &hKey4);
if (result != TSS_SUCCESS) {
    print_error("Tspi_Context_GetKeyByUUID", result);
    print_error_exit(nameOfFunction, err_string(result));
    Tspi_Context_FreeMemory(hContext, NULL);
    Tspi_Context_Close(hContext);
    exit(result);
}

result = set_popup_secret(hContext, hKey3, TSS_POLICY_USAGE, "Input K3's pin", 0);
if (TSS_SUCCESS != result) {
    print_error("set_popup_secret", result);
    Tspi_Context_Close(hContext);
    return result;
}

result = Tspi_Key_LoadKey(hKey4, hKey3);
if (result != TSS_SUCCESS) {
    print_error("Tspi_Key_LoadKey", result);
    Tspi_Context_FreeMemory(hContext, NULL);
    Tspi_Context_Close(hContext);
    exit(result);
}
////////////////////////////////////
printf("Load UserK4 succeded!\n");

```

修改完成之后编译运行


```
sudo make
./load_key -v 1.2
```

可以看到按照提示输入在 `create_register_key.c` 文件中创建输入的各个密钥，即可完成对这些密钥的加载。由此完成密钥层次的创建过程。

```
[05/23/2024 03:52] seed@ubuntu:~/Desktop/trusted-computing-projectv0.3/KeyHierarchy$ ./load_key -v 1.2
<<<test_start>>>
Testing Load KEY
KEY structure:
SRK
|__UserK1(Storage key, unmigratable)
|  |__UserK2(Signing key, unmigratable)
|  |
|  |__UserK3(Storage key, migratable)
|  |__UserK4(Bind key, migratable)

Load SRK succeeded!
Loading K1...
Load UserK1 succeeded!
Loading K2...
Input K1's pin
Enter PIN:
Load UserK2 succeeded!
Loading K3...
Load UserK3 succeeded!
Loading K4...
Input K3's pin
Enter PIN:
Load UserK4 succeeded!
Input K3's pin
Enter PIN:
Load UserK4 succeeded!
1 PASS : Load KEY returned (0) TSS_SUCCESS
Cleaning up Load KEY
<<<end_test>>>
```

2.5 Seal、Unseal 和 extend

配置可信平台模块（TPM）的计算机能够生成一个与特定硬件或软件环境绑定的密钥，这个密钥被称为封装密钥。在封装密钥首次生成时，TPM 会捕获并记录系统配置状态和文件哈希值的快照。封装密钥的解封或解密仅在当前系统状态与快照中记录的状态完全一致时才会发生。简而言之，封装过程涉及创建一个基于当前软硬件配置派生的密钥，利用这个密钥对敏感数据进行加密，以此确保数据的安全。

实验需要补充完成 `unseal.c` 文件。整个任务流程主要如下命令所示：

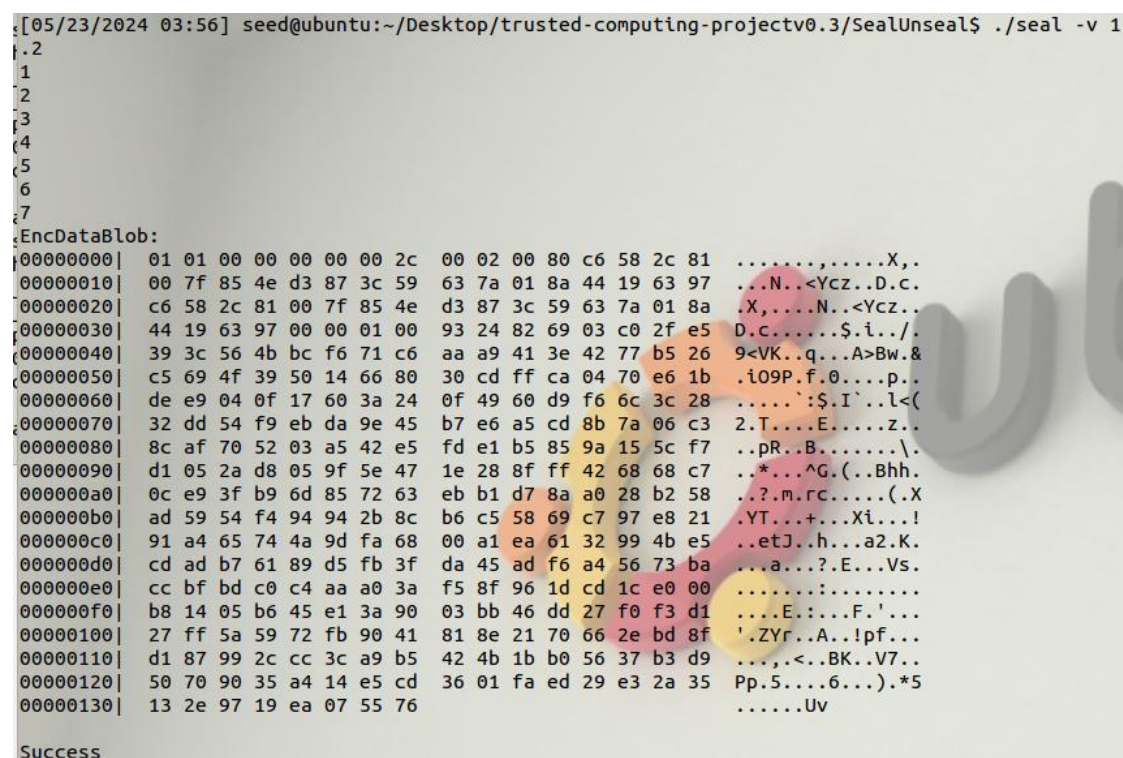
```
./seal -v 1.2 （成功）
./unseal -v 1.2 （成功）
#扩展 PCR 寄存器
./extend -v 1.2 （成功）
./unseal -v 1.2 （失败）
./seal_file test.c test.en （查看文件 test.en 的内容）
```

```
# unseal_file.c 由同学们自己完成。
./unseal_file test.en test.de #（查看文件 test.de 的内容）
./extend -v 1.2
./unseal_file test.en test.de #（失败）
./seal -v 1.2 （成功）
```

①首先对当前系统状态进行封装，即 Seal 操作：

```
./seal -v 1.2
```

如下图，可见执行成功：



```
[05/23/2024 03:56] seed@ubuntu:~/Desktop/trusted-computing-projectv0.3/SealUnseal$ ./seal -v 1
.2
1
2
3
4
5
6
7
EncDataBlob:
00000000| 01 01 00 00 00 00 00 2c 00 02 00 80 c6 58 2c 81 .....X,..
00000010| 00 7f 85 4e d3 87 3c 59 63 7a 01 8a 44 19 63 97 ...N..<Ycz..D.c.
00000020| c6 58 2c 81 00 7f 85 4e d3 87 3c 59 63 7a 01 8a .X,...N..<Ycz..
00000030| 44 19 63 97 00 00 01 00 93 24 82 69 03 c0 2f e5 D.c.....$.i../.
00000040| 39 3c 56 4b bc f6 71 c6 aa a9 41 3e 42 77 b5 26 9<VK...q...A>Bw.&
00000050| c5 69 4f 39 50 14 66 80 30 cd ff ca 04 70 e6 1b .i09P.f.0....p..
00000060| de e9 04 0f 17 60 3a 24 0f 49 60 d9 f6 6c 3c 28 .....$:I`.l<
00000070| 32 dd 54 f9 eb da 9e 45 b7 e6 a5 cd 8b 7a 06 c3 2.T....E.....Z..
00000080| 8c af 70 52 03 a5 42 e5 fd e1 b5 85 9a 15 5c f7 ..pR...B.....\..
00000090| d1 05 2a d8 05 9f 5e 47 1e 28 8f ff 42 68 68 c7 ..*...^G.(..Bhh.
000000a0| 0c e9 3f b9 6d 85 72 63 eb b1 d7 8a a0 28 b2 58 ..?.m.rc....(.X
000000b0| ad 59 54 f4 94 94 2b 8c b6 c5 58 69 c7 97 e8 21 .YT...+...Xi...!
000000c0| 91 a4 65 74 4a 9d fa 68 00 a1 ea 61 32 99 4b e5 ..etJ..h...a2.K.
000000d0| cd ad b7 61 89 d5 fb 3f da 45 ad f6 a4 56 73 ba ...a...?.E...Vs.
000000e0| cc bf bd c0 c4 aa a0 3a f5 8f 96 1d cd 1c e0 00 .....:.....
000000f0| b8 14 05 b6 45 e1 3a 90 03 bb 46 dd 27 f0 f3 d1 ....E.....F.'...
00000100| 27 ff 5a 59 72 fb 90 41 81 8e 21 70 66 2e bd 8f '.ZYr..A..!pf...
00000110| d1 87 99 2c cc 3c a9 b5 42 4b 1b b0 56 37 b3 d9 ...,<...BK...V7..
00000120| 50 70 90 35 a4 14 e5 cd 36 01 fa ed 29 e3 2a 35 Pp.5....6...)*5
00000130| 13 2e 97 19 ea 07 55 76 .....Uv
Success
```

封装密钥（Sealing Key）是配置了可信平台模块（TPM）的计算机生成的密钥，它不仅与特定的硬件或软件环境相绑定，而且与之紧密关联。在首次生成封装密钥的过程中，TPM 会捕捉并存储系统配置参数和文件哈希值的即时状态。封装密钥的解密或解封仅在当前系统配置与存储的快照完全一致时才被允许。这种密钥设计为不可移植，确保了其与特定系统环境的绑定性。

②执行解封指令，由于系统状态未改变，与之前一致，执行成功：

```
./unseal -v 1.2
```

```
[05/23/2024 03:56] seed@ubuntu:~/Desktop/trusted-computing-projectv0.3/SealUnseal$ ./unseal -v 1.2
Sealed data:
00000000| 01 01 00 00 00 00 00 2c 00 02 00 80 c6 58 2c 81 .....X,.
00000010| 00 7f 85 4e d3 87 3c 59 63 7a 01 8a 44 19 63 97 ...N...<Ycz..D.c.
00000020| c6 58 2c 81 00 7f 85 4e d3 87 3c 59 63 7a 01 8a .X,...N...<Ycz..
00000030| 44 19 63 97 00 00 01 00 93 24 82 69 03 c0 2f e5 D.c.....$.i../.
00000040| 39 3c 56 4b bc f6 71 c6 aa a9 41 3e 42 77 b5 26 9<VK...q...A>Bw.&
00000050| c5 69 4f 39 50 14 66 80 30 cd ff ca 04 70 e6 1b .i09P.f.0....p..
00000060| de e9 04 0f 17 60 3a 24 0f 49 60 d9 f6 6c 3c 28 .....`$.I`.l<
00000070| 32 dd 54 f9 eb da 9e 45 b7 e6 a5 cd 8b 7a 06 c3 2.T....E....Z..
00000080| 8c af 70 52 03 a5 42 e5 fd e1 b5 85 9a 15 5c f7 ..pR..B.....\..
00000090| d1 05 2a d8 05 9f 5e 47 1e 28 8f ff 42 68 68 c7 ..*...^G.(..Bhh.
000000a0| 0c e9 3f b9 6d 85 72 63 eb b1 d7 8a a0 28 b2 58 ..?.m.rc.....(X
000000b0| ad 59 54 f4 94 94 2b 8c b6 c5 58 69 c7 97 e8 21 .YT...+...Xi...!
000000c0| 91 a4 65 74 4a 9d fa 68 00 a1 ea 61 32 99 4b e5 ..etJ..h...a2.K.
000000d0| cd ad b7 61 89 d5 fb 3f da 45 ad f6 a4 56 73 ba ...a...?.E...Vs.
000000e0| cc bf bd c0 c4 aa a0 3a f5 8f 96 1d cd 1c e0 00 .....:.....
000000f0| b8 14 05 b6 45 e1 3a 90 03 bb 46 dd 27 f0 f3 d1 ....E:....F.'...
00000100| 27 ff 5a 59 72 fb 90 41 81 8e 21 70 66 2e bd 8f '.ZYr..A..!pf...
00000110| d1 87 99 2c cc 3c a9 b5 42 4b 1b b0 56 37 b3 d9 ...,<..BK..V7..
00000120| 50 70 90 35 a4 14 e5 cd 36 01 fa ed 29 e3 2a 35 Pp.5....6...)*5
00000130| 13 2e 97 19 ea 07 55 76 .....Uv
Unsealed Data:
00000000| 30 31 32 33 34 35 36 37 38 39 41 42 43 44 45 46 0123456789ABCDEF
00000010| 30 31 32 33 34 35 36 37 38 39 41 42 43 44 45 46 0123456789ABCDEF
Success
```

③执行扩展指令，实现对 PCR 寄存器的扩展，执行成功：

```
./extend -v 1.2
```

```
[05/23/2024 03:56] seed@ubuntu:~/Desktop/trusted-computing-projectv0.3/SealUnseal$ ./extend -v 1.2
ulPcrValLen:20
Success
```

④再次执行解封指令，执行失败，可以看到运行失败的错误码提示，指示 TPM_E_WRONGPCRVAL 错误。因为 PCR 扩展之后系统状态改变，与封装时不一致，故无法解封。

```
./unseal -v 1.2
```

```
[05/23/2024 03:59] seed@ubuntu:~/Desktop/trusted-computing-projectv0.3/SealUnseal$ ./unseal -v 1.2
Sealed data:
00000000| 01 01 00 00 00 00 00 2c 00 02 00 80 c6 58 2c 81 .....X,.
00000010| 00 7f 85 4e d3 87 3c 59 63 7a 01 8a 44 19 63 97 ...N...<Ycz..D.c.
00000020| c6 58 2c 81 00 7f 85 4e d3 87 3c 59 63 7a 01 8a .X,...N...<Ycz..
00000030| 44 19 63 97 00 00 01 00 93 24 82 69 03 c0 2f e5 D.c.....$.i../.
00000040| 39 3c 56 4b bc f6 71 c6 aa a9 41 3e 42 77 b5 26 9<VK...q...A>Bw.&
00000050| c5 69 4f 39 50 14 66 80 30 cd ff ca 04 70 e6 1b .i09P.f.0....p..
00000060| de e9 04 0f 17 60 3a 24 0f 49 60 d9 f6 6c 3c 28 .....`$.I`.l<
00000070| 32 dd 54 f9 eb da 9e 45 b7 e6 a5 cd 8b 7a 06 c3 2.T....E....Z..
00000080| 8c af 70 52 03 a5 42 e5 fd e1 b5 85 9a 15 5c f7 ..pR..B.....\..
00000090| d1 05 2a d8 05 9f 5e 47 1e 28 8f ff 42 68 68 c7 ..*...^G.(..Bhh.
000000a0| 0c e9 3f b9 6d 85 72 63 eb b1 d7 8a a0 28 b2 58 ..?.m.rc.....(X
000000b0| ad 59 54 f4 94 94 2b 8c b6 c5 58 69 c7 97 e8 21 .YT...+...Xi...!
000000c0| 91 a4 65 74 4a 9d fa 68 00 a1 ea 61 32 99 4b e5 ..etJ..h...a2.K.
000000d0| cd ad b7 61 89 d5 fb 3f da 45 ad f6 a4 56 73 ba ...a...?.E...Vs.
000000e0| cc bf bd c0 c4 aa a0 3a f5 8f 96 1d cd 1c e0 00 .....:.....
000000f0| b8 14 05 b6 45 e1 3a 90 03 bb 46 dd 27 f0 f3 d1 ....E:....F.'...
00000100| 27 ff 5a 59 72 fb 90 41 81 8e 21 70 66 2e bd 8f '.ZYr..A..!pf...
00000110| d1 87 99 2c cc 3c a9 b5 42 4b 1b b0 56 37 b3 d9 ...,<..BK..V7..
00000120| 50 70 90 35 a4 14 e5 cd 36 01 fa ed 29 e3 2a 35 Pp.5....6...)*5
00000130| 13 2e 97 19 ea 07 55 76 .....Uv
0 FAIL : Tspi_Data_Unseal returned (24) TPM_E_WRONGPCRVAL
unseal.c 0 FAIL : Tspi_Data_Unseal returned (24) TPM_E_WRONGPCRVAL
[05/23/2024 03:59] seed@ubuntu:~/Desktop/trusted-computing-projectv0.3/SealUnseal$
```

⑤执行 `/seal file test.c test.en` 指令后，gedit 查看文件 test.en 的内容：


```
[05/23/2024 03:59] seed@ubuntu:~/Desktop/trusted-computing-projectv0.3/SealUnseal$ ./seal_file test.c test.en
Input K1's Pin

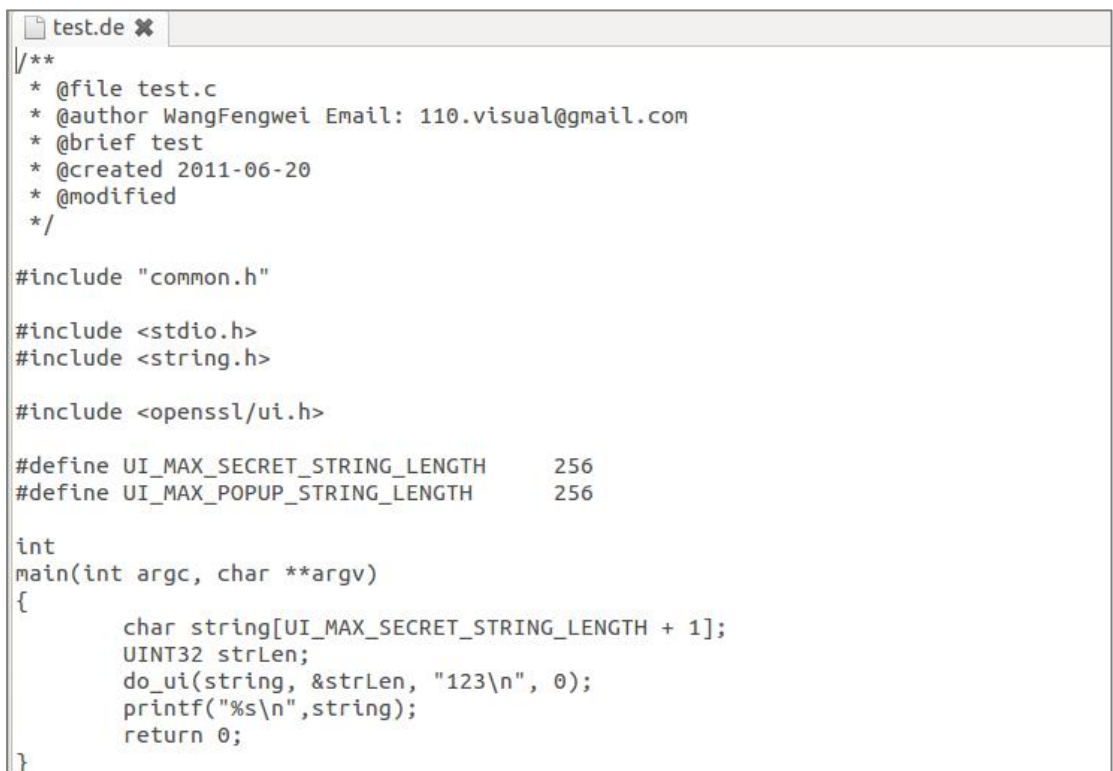
Enter PIN:
[05/23/2024 04:01] seed@ubuntu:~/Desktop/trusted-computing-projectv0.3/SealUnseal$ gedit ./test.en
```



test 本身是一个可读的程序，而在使用 seal_file 密封之后，test.en 的内容变成乱码无法查阅，这是加密后的结果。

⑥重新 make 编译生成 unseal_file 文件，使用 unseal 对 test.en 进行解封，得到的 test.de 文件是原始的明文文件

```
./unseal_file test.en test.de #（查看文件 test.de 的内容）
gedit test.de
```



⑦执行 `./extend -v 1.2` 再次完成对 PCR 寄存器的扩展：

```
[05/23/2024 04:06] seed@ubuntu:~/Desktop/trusted-computing-projectv0.3/SealUnseal$ ./extend -v 1.2
ulPcrValLen:20

Success
```

⑧执行 `/unseal_file test.en test.de`，可看到对 `test.en` 文件的解密封失败：

```
[05/23/2024 04:06] seed@ubuntu:~/Desktop/trusted-computing-projectv0.3/SealUnseal$ ./unseal_file test.en test.de
Input K1's Pin

Enter PIN:
1
0 FAIL : Tspi_Data_Unseal returned (24) TPM_E_WRONGPCRVAL
unseal file.c 0 FAIL : Tspi_Data_Unseal returned (24) TPM_E_WRONGPCRVAL
```

2.6 秘钥迁移（KeyMigration）

进入 Key Migration 目录

```
cd ~/Desktop/trusted-computing-projectv0.3/KeyMigration
```

①实验首先需要完成 `platform_dst.c` 中的部分代码。

在 `platform_dst.c` 文件中需要添加的代码部分就是密钥迁移的操作,具体如下所示代码。该过程首先利用 `Tspi_Key_ConvertMigrationBlob()`函数进行迁移，然后加载新的迁移密钥并验证是否加载成功，再验证新的迁移密钥是否有效。

```
// TODO: 完成以下代码, 参考Tspi_Key_ConvertMigrationBlob
////////////////////////////////////
result = Tspi_Key_ConvertMigrationBlob(hNewMigKey,
                                       hSRK,
                                       u32RandomLen,
                                       pRandom,
                                       u32MigBlobLen,
                                       pMigBlob);

////////////////////////////////////
free(pRandom);
free(pMigBlob);
if (TSS_SUCCESS != result) {
    print_error("Tspi_Key_ConvertMigrationBlob", result);
    Tspi_Context_Close(hContext);
    return result;
}
```

改完代码后执行 `sudo make` 重新编译。

②在虚拟机 1 中运行如下指令，即让密钥迁移源生成公钥文件

```
./platform_dst -g
```

查看路径下的文件列表，观察到产生名为 `srk.pub` 的文件，也就是公钥文件

```
[05/23/2024 04:12] seed@ubuntu:~/Desktop/trusted-computing-projectv0.3/KeyMigration$ ./platform_dst -g
Generating Pub Key Success
[05/23/2024 04:12] seed@ubuntu:~/Desktop/trusted-computing-projectv0.3/KeyMigration$ ls
Makefile platform_dst platform_dst.c~ platform_src platform_src.c_ srk.pub
```

③把文件 `srk.pub` 拷贝到虚拟机 2 中，也就是模拟密钥的迁移过程，运行生成迁移密钥

```
./platform_src
```



```
[05/23/2024 04:13] seed@ubuntu:~/Desktop/trusted-computing-projectv0.3/KeyMigration$ ./platform_src
Input Migration Key's Pin

Enter PIN:
OK
[05/23/2024 04:17] seed@ubuntu:~/Desktop/trusted-computing-projectv0.3/KeyMigration$ ls
Makefile  mig.blob  platform_dst  platform_dst.c  platform_dst.c~  platform_src  platform_src.c  srk.pub
```

④查看路径下的文件列表，发现产生名为 mig.blob 的文件，文件 mig.blob 拷贝到虚拟机 1 中，也就是进行传输迁移密钥的过程。接着回到虚拟机 1 中，终端运行 `./platform_dst -m` 命令，程序将利用 `Tspi_Key_ConvertMigrationBlob()` 函数进行密钥转换和迁移操作。可观察到返回 `Migration Success` 字符串，迁移成功！

```
[05/23/2024 04:17] seed@ubuntu:~/Desktop/trusted-computing-projectv0.3/KeyMigration$ ./platform_dst -m
Data before signing:
30 39 38 37 36 35 34 33 32 31 30 39 38 37 36 35
34 33 32 31
Data after signing:
35 cd ce 55 d7 9d df 05 ad 98 91 8f 42 22 46 51
47 03 e3 3d e4 5f 0c 6f c3 78 95 eb 47 55 67 4c
47 65 42 a9 49 1d 92 b6 e8 ad 86 b0 fd a6 dd 9d
f1 4d 88 a1 67 b5 8a 5f 48 65 7b 5a ab e4 45 0d
2d 5f 1c 35 03 b6 ad 9f c6 35 e7 e1 a0 79 53 11
83 32 37 1e 1b 10 c1 f4 fa 2d ee 5f 35 6f 6a f7
8f c0 93 e2 91 54 f6 ed 2b 61 ba 49 66 9c 1c 0e
74 ad 0f b1 7b b6 6c 18 49 29 e9 0d 5c cf bb 27
4d 2e c8 9a 22 0d 73 ac db 10 97 8e 5d 82 a2 4a
d1 47 55 e6 0c 17 d7 a8 18 4f ce fc 5a 88 5c cf
f2 51 55 78 c3 6e 56 85 00 ce ec af 39 b6 92 c4
da 85 58 82 07 f2 02 47 2d 95 64 29 b0 10 3d 06
0b 52 ef 2c 3a c1 38 e7 b9 e7 75 4e e1 18 2b 71
46 a3 6a e7 3d da f3 9e 3f d2 31 49 19 38 24 0f
6e 4e 5b e0 14 af 31 ba bf 13 6b 77 e9 e4 ed f0
f4 60 f2 d8 35 35 e7 bd 9d b8 ac 3d 51 39 b8 1d
Data after verifying:
30 39 38 37 36 35 34 33 32 31 30 39 38 37 36 35
34 33 32 31
Migration Success
```

2.7 远程证明（RemoteAttestation）

远程证明过程是一种确保远程计算机系统安全和完整性的技术。首先，在服务器端，通过执行特定的初始化命令，创建一个称为“认证身份密钥”（AIK）的密钥对，这是远程证明中用于验证的关键。然后，服务器端启动一个远程证明服务，等待客户端的连接。在客户端，首先获取客户端和服务器的 IP 地址，然后运行客户端程序，通过指定的 IP 地址连接到服务器。一旦连接建立，服务器端将使用 AIK 对当前的系统状态进行签名，并将这个签名信息发送给客户端。客户端接收到签名信息后，会进行验证，以确认服务器端的系统状态是否可信。这个过程可以有效地验证服务器端的硬件和软件配置，确保没有被篡改或存在安全漏洞。

本实验配置两台相同的虚拟机分别作为机器 1 和机器 2 完成远程证明过程。

机器 1 操作：

①首先在虚拟机 1 中使用 ifconfig 命令查看其 ip 地址为 192.168.117.142

```
[05/23/2024 04:27] seed@ubuntu:~$ ifconfig
eth0      Link encap:Ethernet  HWaddr 00:0c:29:77:ad:10
          inet addr:192.168.117.142  Bcast:192.168.117.255  Mask:255.255.255.0
          inet6 addr: fe80::20c:29ff:fe77:ad10/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:40401 errors:499 dropped:0 overruns:0 frame:0
          TX packets:25173 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:52791132 (52.7 MB)  TX bytes:1788507 (1.7 MB)
          Interrupt:19 Base address:0x2000

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:16436  Metric:1
          RX packets:734 errors:0 dropped:0 overruns:0 frame:0
          TX packets:734 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:104704 (104.7 KB)  TX bytes:104704 (104.7 KB)
```

②进入虚拟机 1 的 RemoteAttestation/init 目录，运行命令创建身份认证密钥 AIK，这是用于远程证明的关键对之一。

```
cd ~/Desktop/trusted-computing-projectv0.3/RemoteAttestation/init
./Create_AI
```

```
[05/23/2024 04:21] seed@ubuntu:~/Desktop/trusted-computing-projectv0.3/RemoteAttestation/init$ ./Create_AI
K
497 Beginning remote_attestation()
[Client:]Tspi_TPM_CollateIdentityRequest.....
[CA:]Decrypt symmetric key.....
[CA:]Decrypt symmetric blob
[CA:] Verify the identity binding.....
[CA:]Create a fake credential and encrypt it.....
[Client:]Activate the TPM identity, receiving back the decrypted credential.....
Create_AI.c:762 receive_msg_size is 44
Create_AI.c:780      1 PASS : Create AIK returned (0) TSS_SUCCESS
786 Ending remote_attestation()
801 remote_data_size 44
803 keyPubSize 256
Create_AI.c:805 ##The content of the keyPub is:##
95 dc 6e 95 3d b0 f8 d1 f3 27 dd b4 6d b9 bc a8
65 9b fa eb 5a 2e 81 d1 6c 6f 3c ef c3 ef 0b a2
a6 1a 5d c4 be 03 03 ff aa df ee c2 80 a8 51 3c
41 38 f9 04 df 41 ae 8c 4f 8e 56 91 75 c3 1e 24
f1 6a b6 9b e9 52 8b 27 09 97 e6 2c e8 4f 33 5c
2c ec bc d3 2c c6 e8 72 33 65 a0 92 ed 01 86 14
dc 0d 99 58 56 74 35 cc aa 7e 53 bd 41 07 35 3b
4d d9 cf 7a 1f b9 b1 b0 98 c1 85 90 3a 56 85 66
b9 49 e7 c1 61 91 b5 61 27 d0 c1 0a 12 34 10 16
75 25 f8 7b 10 4b a2 e7 e7 2c 5d 38 dd c8 a1 ed
ad 4b 56 84 5a f1 56 66 e1 69 a9 f1 a1 f6 4b 08
2a 5e 39 a4 fb 0b 7b 1f 6d 46 79 da 81 21 03 cb
ec d2 a7 ad f7 ce 54 eb b7 e7 69 c0 8e 9b de 2b
04 28 1c 8d f2 46 07 5b 89 1a 00 16 aa 40 b4 da
e9 10 7b ec a5 4f fe 57 7b 0c ee e4 44 9d 64 03
1b 0c a5 a2 7f e3 52 29 7e 0f 2d 27 af 68 b8 f1
Create_AI.c:807 ##The content of the keyPub is over##
809 credLen 104
Create_AI.c:811 ##The content of the cred is:##
5a 5a 5a 5a 5a 5a 5a 5a 5a 5a 5a 5a 5a 5a 5a
5a 5a 5a 5a 5a 5a 5a 5a 5a 5a 5a 5a 5a 5a 5a
5a 5a 5a 5a 5a 5a 5a 5a 5a 5a 5a 5a 5a 5a 5a
5a 5a 5a 5a 5a 5a 5a 5a 5a 5a 5a 5a 5a 5a 5a
5a 5a 5a 5a 5a 5a 5a 5a 5a 5a 5a 5a 5a 5a 5a
5a 5a 5a 5a 5a 5a 5a 5a 5a 5a 5a 5a 5a 5a 5a
5a 5a 5a 5a 5a 5a 5a
Create_AI.c:813 ##The content of the cred is over##
815 After calling remote_attestation()
817 -- remote data --
00 00 00 00 00 00 00 00 00 00 01 00 00 68 00 00 00
30 00 00 00 10 00 00 00 88 9d ca 08 90 9e ca 08
00 00 00 00 11 00 00 00 38 91 ca 08
819 -- receive msg --
```


③然后返回上级目录执行./RAServer 启动远程证明服务器，如下：

```
[05/23/2024 04:21] seed@ubuntu:~/Desktop/trusted-computing-projectv0.3/RemoteAttestation/init$ cd ..
[05/23/2024 04:22] seed@ubuntu:~/Desktop/trusted-computing-projectv0.3/RemoteAttestation$ ./RAServer

*****
***** Welcome to IBM Remote Attestation Server 111*****
*****
```

机器 2 操作：

④进入虚拟机 2 同样使用 ifconfig 命令查看 ip 为 192.168.117.144：

```
[05/23/2024 07:11] seed@ubuntu:~/Desktop$ ifconfig
eth0      Link encap:Ethernet  HWaddr 00:0c:29:f3:29:5a
          inet addr:192.168.117.144  Bcast:192.168.117.255  Mask:255.255.255.0
          inet6 addr: fe80::20c:29ff:fef3:295a/64  Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:163 errors:0 dropped:0 overruns:0 frame:0
          TX packets:175 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:44764 (44.7 KB)  TX bytes:18204 (18.2 KB)
          Interrupt:19 Base address:0x2000

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:16436  Metric:1
          RX packets:31 errors:0 dropped:0 overruns:0 frame:0
          TX packets:31 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:2464 (2.4 KB)  TX bytes:2464 (2.4 KB)
```

⑤然后进入虚拟机 2 的 RemoteAttestation 目录，执行./RAClient 启动远程证明客户端，注意命令参数 1 是虚拟机 2 的 ip 地址，参数 2 是虚拟机 1 的 ip 地址。

```
./RAClient 192.168.117.144 192.168.117.142)
```

执行完成后双方完成远程证明过程，虚拟机 2 的终端显示内容如下，可以看到 PCR signature verify OK 和 nonce match 的字符串。

```
RAClient.c:450 ##The content of the rc is over##

Verify SHA1(TCPA_QUOTE_INFO) signature...
===== PCR signature verify OK =====
Verify nonce...
pcrCompositeSize: 48

RAClient.c:526 ##The content of the pcrComposite is:##
01 01 00 00 51 55 4f 54 10 ea a7 90 83 cb 30 d3
d9 86 6a ad 27 52 f9 33 2d 82 a4 ec 00 01 02 03
04 05 06 07 08 09 0a 0b 0c 0d 0e 0f 10 11 12 13
RAClient.c:528 ##The content of the pcrComposite is over##

RAClient.c:540 ##The content of the pcrComposite(nonce) is:##
00 01 02 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f
10 11 12 13
RAClient.c:542 ##The content of the pcrComposite(nonce) is over##

===== nonce match! =====
source IP:
```

虚拟 1 的终端内容显示如下，可见最后打印出的 Send Encrypted PCR to RAClient 字符串。两个终端的内容说明远程证明成功！

```
remote_attestation.c:842 ##The content of the pcr12 Value is over##
remote_attestation.c:840 ##The content of the pcr14 Value is:##
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00
remote_attestation.c:842 ##The content of the pcr14 Value is over##
remote_attestation.c:850 selectpcrSize: 80  nr_select_PCR: 4
Tspi_TPM_Quote...
remote_attestation.c:956 pcr_tpmSize: 80
remote_attestation.c:969 requested malloc size for rc = 1032
remote_attestation.c:992 *****0*****
remote_attestation.c:992 *****0*****
remote_attestation.c:992 *****0*****
remote_attestation.c:992 *****0*****
remote_attestation.c:992 *****0*****
remote_attestation.c:992 *****0*****
remote_attestation.c:992 *****0*****
remote_attestation.c:992 *****0*****
remote_attestation.c:992 *****1*****
remote_attestation.c:992 *****2*****
remote_attestation.c:992 *****0*****
remote_attestation.c:992 *****0*****
remote_attestation.c:992 *****16*****
remote_attestation.c:992 *****0*****
remote_attestation.c:992 *****64*****
remote_attestation.c:992 *****0*****
remote_attestation.c:995 pcr_tpmSize = 80
pcr_tpm:
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
remote_attestation.c:1005 receive_msg_size is 664
remote_attestation.c:1181 1 PASS : Remote Attestation returned (0) TSS_SUCCESS
1187 remote_data_size: 24  receive_msg_size: 664
1188 Ending remote_attestation()
##The nonce received from the client is:##
00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
10 11 12 13
##The nonce received from the client is over##
===== Send Encrypted PCR to RAClient =====
```

3、实验心得及建议

本次实验的难度适合，主要是理解课堂上的 TPM 理论知识并进行简单应用，主要的代码和环境配置在实验资料中均已给出，只需要按着流程走，稍微补充代码即可。更重要的是对整个实验的实现原理和过程了解清楚。

尽管实验过程中遇到了一些挑战，但这些问题最终都得到了解决。首先，环境配置失败的问题是由于实验手册中提供的镜像源链接已经过时，导致某些扩展库无法下载。通过更换新的镜像源，这个问题便迎刃而解。其次，加载 K4 密钥失败的问题，经过仔细检查代码，我发现是由于函数调用不正确导致的。将 `Tspi_Context_LoadKeyBYUUID` 函数更改为 `Tspi_Context_GetKeyBYUUID` 后，程序便能顺利获取密钥参数并正常运行。

通过这次实验，我对 TPM 中的核心概念有了更加深入的理解。首先，密钥层次的概念让我认识到了在安全计算中，不同层次的密钥如何协同工作以提供不同级别的保护。我了解到，每个层次的密钥都有其特定的用途和权限，它们共同构成了一个安全的密钥管理架构。封装密钥的概念进一步加深了我对数据保护的理解。我明白了封装密钥是如何与特定的硬件或软件状态绑定，以及如何利用这种绑定来确保数据的安全性。这种密钥只有在特定的系统状态下才能被解封，这为敏感数据提供了额外的安全层。密钥迁移的概念则让我理解了在不同设备或环境中安全地传输密钥的重要性。远程证明则让我认识到了如何远程验证一个系统的完整性和安全性，这对于构建信任和安全至关重要。

总的来说，这次实验不仅让我动手实践，还让我对密钥层次、封装密钥、密钥迁移和远程证明等关键概念有了更深刻的认识。非常感谢老师的精心备课和助教在这次实验中的帮助。

此外，关于实验我想提出一点小小建议：由于实验操作性强，需要编写的代码量较少且难度不高，整体还是跟着指导书进行，更像是现象观察类实验。所以我建议老师考虑在课程平台上直接发布题目，让学生分别提交各个任务的实验结果。这样的做法利于老师进行更高效的检查和评分，也可减轻不必要的报告负担。