

课程实验报告

专业班级:

学 号:

姓 名:

报告日期: . .

目 录

1. 实验目的.....	1
2. 实验环境.....	1
3. 实验内容.....	1
3.1 安装、编译和初始化.....	1
3.2 密钥层次 (Key Hierarchy).....	3
3.3 密钥迁移 Seal、Unseal 和 extend.....	5
3.4 密钥迁移 (Key Migration)	10
3.5 远程证明 (Remote Attestation)	12
4. 实验心得.....	15

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 tpm 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。

2. 实验环境

- (1) Seed Ubuntu 12.04 LTS 32 位的 VMware 虚拟机
- (2) TPM Emulator
- (3) Trousers

3. 实验内容

3.1 安装、编译和初始化

为了模拟 TPM 的功能,我们在 ubuntu12.04 虚拟机中,安装 TPM Emulator,进而完成 TPM 的相关实验。首先为旧版本的虚拟机切换可用源,将实验工具包中的源码 tpm-emulator.tar.gz,置于工作目录下,解压并编译安装 TPM Emulator。

按照实验指导书的要求,一步步安装依赖,编译安装即可。最后安装 TSS 软件栈。安装 TPM 过程中使用的命令如下:

```
tar xvzf 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 TSS 软件栈
sudo apt-get install libtspi-dev trousers
```

将实验的源码在 Windows 下解压然后拷贝到虚拟机中，编译本次实验源码。首先进入到 `trusted-computing-projectv0.3` 工作目录中，执行 `make clean` 并 `make`，编译实验代码。接着开始初始化操作，使用 `modprobe` 载入 `tpm` 模块 `tpmd_dev`。其中 `modprobe` 命令用于自动载入模块，即根据 `depmod` 所产生的相依关系决定要载入的模块，当载入过程中发生错误 `modprobe` 会卸载所有模块。

使用 `tpmd -f -d clear` 的命令启动 `tpm` 模拟器，并在另一个终端中启动 Trousers 软件栈，命令为 `sudo tcscd`。`tcscd` 是一个用户空间下的守护进程，提供了与 TPM 交互的 API，负责管理 TPM 资源并处理来自 TSP 的本地和远程请求，如图 1 所示。

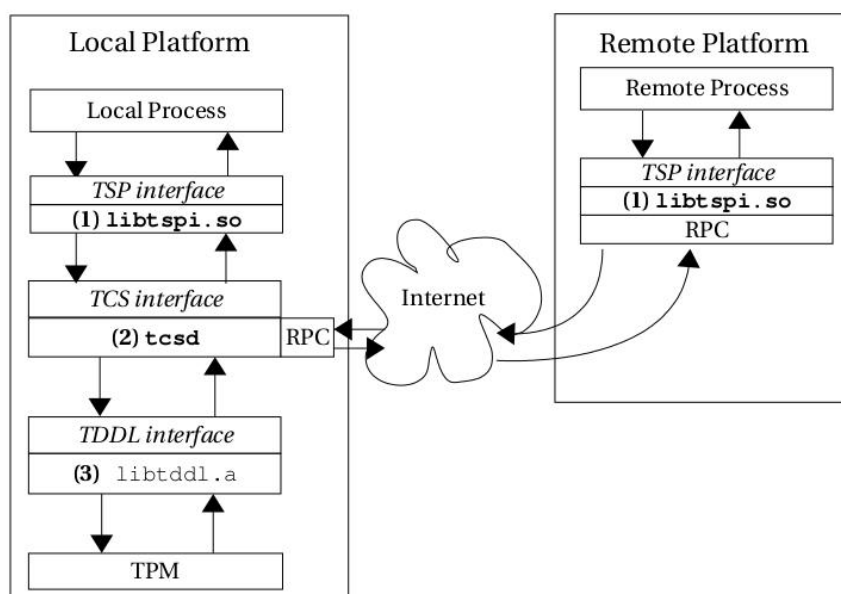


图 1 tcscd 进程的功能

进入到 `init` 目录中，通过 `take ownership` 的操作获得用户身份，生成 `AIK`。使用自定义的 `pin` 生成存储根密钥 `SRK`，后续就可以生成其他下层密钥。命令为：

```
./Tspi_TPM_TakeOwnership01 -v 1.2
./create_mig_key -v 1.2
```

完成初始化操作后的虚拟机状态如图 2 所示。

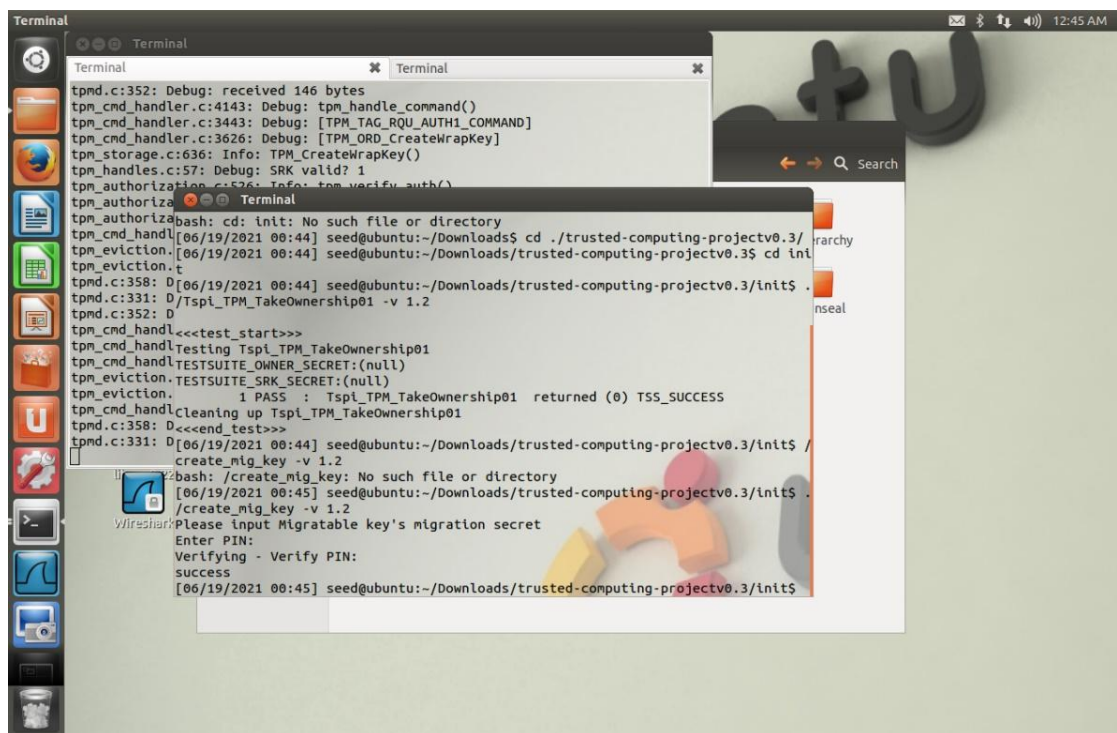


图 2 初始化成功

3.2 密钥层次 (Key Hierarchy)

在密钥层次中，我们需要理清实验中设计的密钥的属性和结构。TPM 对密钥进行分层管理，下层密钥由上层密钥进行加密保护。在初始化时已经生成了存储根密钥 SRK，它是最高权限的存储密钥，一个 TPM 仅存在唯一一个，所有其它的密钥都在存储根密钥的保护之下。

在整个 TPM 的密钥体系中，每个密钥在开始创建的时候都需要指定密钥属性，即可迁移密钥和不可迁移密钥。可迁移密钥并不局限于某个特定平台，可以由平台用户的控制在平台之间迁移。不可迁移密钥则永久地与某个指定平台关联。不可迁移密钥能够用来加密保护可迁移密钥，反之则不可。

实验需要构建的密钥层次结构如图 3 所示。其中 K1 为不可迁移的存储密钥，保护不可迁移的签名密钥 K2；K3 为可迁移的存储密钥，保护可迁移绑定密钥 K4。K1 和 K3 都直接以 SRK 作为父密钥。在参考代码中，已经完成了 K1、K2、K3 的生成和层次结构样例代码，我们只需完成 K4 的生成并使 K3 成为 K4 的父密钥。

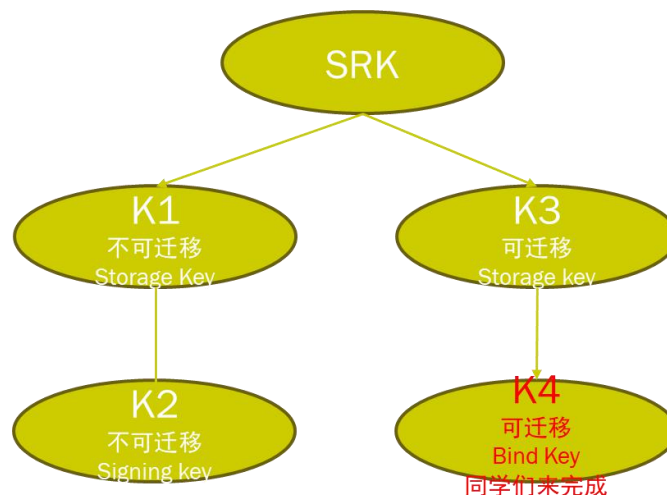


图 3 密钥的层次结构

在 `create_register_key.c` 中，需要我们完成 K4 密钥的生成，为其指定属性。其中 K4 的属性如图 4 所示。通过自己编写的函数 `my_create_key` 生成 TPM 密钥，函数 `my_create_load_key` 用于加载密钥。其中 `my_create_load_key` 函数具体功能是检查密钥句柄对应的密钥是否已经存在，若存在则调用 `Tspi_Key_LoadKey` 根据其父密钥加载；若不存在则调用 `my_create_key` 生成密钥。参数 `initFlags` 用于向 `my_create_load_key` 传递 K4 的相关信息。

```

printf("Create UserK4 and register it to disk.\n");
//K4:绑定密钥, 2048位, 可变密钥, 授权密钥, 可迁移密钥
initFlags = TSS_KEY_TYPE_BIND | TSS_KEY_SIZE_2048 |
            TSS_KEY_VOLATILE | TSS_KEY_AUTHORIZATION |
            TSS_KEY_MIGRATABLE;
result = my_create_load_key(hContext, initFlags, hKey3, &hKey4, "K4");//创建K4, 父密钥为K3

```

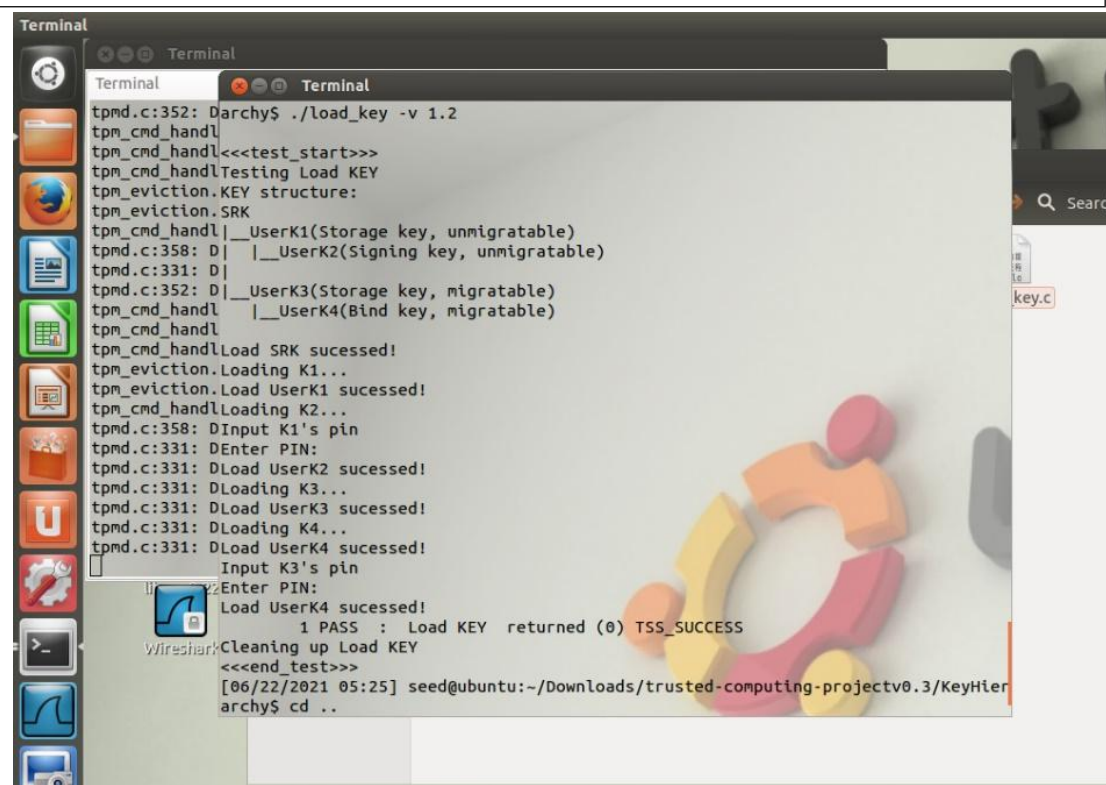
图 4 密钥 K4 的属性

在完成密钥创建后，需要将其注册到上下文中。函数 `Tspi_Context_Create` 用于创建上下文环境，上下文句柄由变量 `hContext` 保存。为该函数设置好注册密钥、UUID、父密钥、上下文和其他系统参数后，就可以完成密钥注册。同时在 `load_key.c` 中，需要借助 `Tspi_Context_Connect` 函数建立起上下文与 TPM 的连接，并在我们补充的部分，使用 `Tspi_Context_LoadKeyByUUID` 函数，在上下文环境中，通过密钥的 UUID 加载密钥，获得相应的密钥句柄，完成 K4 加载。

进入到源码的 `KeyHierarchy` 目录，样例代码中创建加载 SRK、K1、K2、K3 的代码已经实现，按上述分析，指定其父密钥为 K3 和其他属性，补充完创建加载 K4 的代码。重新编译运行，完成 4 个密钥的生成与加载，命令如下，在

每次创建密钥时需要输入 pin，最后的加载结果如图 5 所示。

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

A screenshot of a Linux terminal window titled 'Terminal'. The terminal shows the execution of a TPM key loading process. The user runs './load_key -v 1.2'. The output includes TPM command handling logs, SRK loading success, and sequential loading of UserK1 through UserK4. Each key loading step prompts for a PIN. After loading all keys, it shows '1 PASS : Load KEY returned (0) TSS_SUCCESS' and 'Cleaning up Load KEY'. The terminal also shows the current directory as '~/Downloads/trusted-computing-projectv0.3/KeyHier' and the user as 'archy\$'.

```
Terminal  
tpmd.c:352: Darchy$ ./load_key -v 1.2  
tpm_cmd_handl  
tpm_cmd_handl<<<test_start>>>  
tpm_cmd_handlTesting Load KEY  
tpm_eviction.KEY structure:  
tpm_eviction.SRK  
tpm_cmd_handl|__UserK1(Storage key, unmigratable)  
tpmd.c:358: D| |__UserK2(Signing key, unmigratable)  
tpmd.c:331: D|  
tpmd.c:352: D|__UserK3(Storage key, migratable)  
tpm_cmd_handl|__UserK4(Bind key, migratable)  
tpm_cmd_handl  
tpm_cmd_handlLoad SRK sucessed!  
tpm_eviction.Loading K1...  
tpm_eviction.Load UserK1 sucessed!  
tpm_cmd_handlLoading K2...  
tpmd.c:358: DInput K1's pin  
tpmd.c:331: DEnter PIN:  
tpmd.c:331: DLoad UserK2 sucessed!  
tpmd.c:331: DLoading K3...  
tpmd.c:331: DLoad UserK3 sucessed!  
tpmd.c:331: DLoading K4...  
tpmd.c:331: DLoad UserK4 sucessed!  
Input K3's pin  
22Enter PIN:  
Load UserK4 sucessed!  
1 PASS : Load KEY returned (0) TSS_SUCCESS  
Cleaning up Load KEY  
<<<end_test>>>  
[06/22/2021 05:25] seed@ubuntu:~/Downloads/trusted-computing-projectv0.3/KeyHier  
archy$ cd ..
```

图 5 密钥层次实验结果

3.3 密钥迁移 Seal、Unseal 和 extend

为保护秘密信息，TPM 提供了 seal/unseal 的操作机制。具体来说，内置了 TPM 芯片的计算机可以创建一个密钥，该密钥被关联到特定的硬件或软件条件，这被称为密封密钥。首次创建密封密钥时，TPM 将记录配置值和文件哈希的快照。仅当前系统值与快照中的值相匹配时才解密封或释放密封密钥。加密数据时则是将文件加密的对称密钥 k 做 seal 操作，将密钥 K 与一个或一组指定的 PCR 值绑定，如果本机不安全，则无法 unseal 获得密钥 K，就无法解密数据，从而保护了数据的安全。反应在这次实验中，就是当 PCR 寄存器进行 extend 操作后，unseal 功能会失败。TCG 软件栈提供了 Seal/Unseal 的编程接口，应用层可调用这些 API 实现秘密信息的封装和解封。

样例代码中已经完成了 seal.c、unseal.c 代码，实现了秘密数据封装和解封的

功能。`seal_file.c` 代码则是将对称密钥 Seal，并加密文件保存在密文文件中，需要我们自行编写 `unseal_file.c`，完成从密文文件中提取对称密钥，并解密文件获得明文的过程。

首先尝试分析 `seal_file.c` 的结构。`seal_file.c` 的具体流程大致如下：在加载完密钥之后，调用 `Tspi_TPM_GetRandom` 生成随机数 `random` 并作为对称加密密钥用于对文件加密。后续调用 `Tspi_TPM_PcrRead` 获得 PCR 值，保存在变量中，然后调用 `Tspi_PcrComposite_SetPcrValue` 将 PCR 值与 PCR 句柄 `hPcrs` 绑定。调用 `Tspi_Data_Seal` 进行 Seal 操作，用 K1 加密对称密钥 `random`，与 PCR 值绑定后封装，获得句柄 `hEncData`，并调用 `Tspi_GetAttribData` 将秘密数据的长度和秘密数据的值分别获得并保存。

在完成密钥封装之后，开始对文件加密。使用的是 AES 的加密方式，以前面的随机数为密钥，把读取的文件内容加密后写入指针 `pBufOut`。最后向加密文件中写入 `u32EncDataLen`、`rgbEncData`、`size`、`pBufOut`，分别代表封装信息的长度，封装的秘密信息，文件长度和文件密文，完成一次 `seal_file` 的过程。

在此基础上，`unseal_file` 只需要根据其逆过程来还原即可，实现的思路如图 6 所示。首先构造 AES 解密函数 `aes_decrypt`，并对 TPM 做初始化。依次从文件中读取 `u32EncDataLen`、`rgbEncData`、`size`、`pBufOut` 的内容，调用 `Tspi_SetAttribData` 将封装秘密数据读取出来，获得秘密数据句柄 `hEncData`；调用 `Tspi_Data_Unseal` 将 `hEncData` 所指的秘密数据解封，若 PCR 满足相应状态，则能成功解封，获取到对称密钥 K；反之则无法解密。在取得密钥后，调用 `aes_decrypt` 即可解密数据获得明文，并保存到文件中。

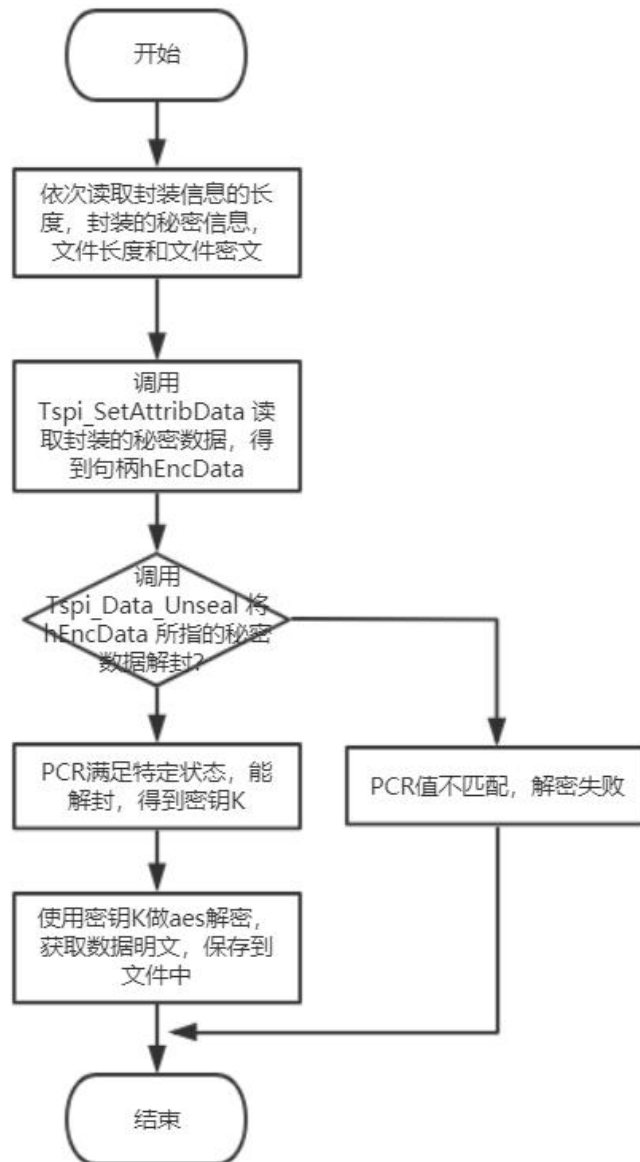


图 6 unseal_file 流程图

进入到源码的 SealUnseal 目录中，编译执行以上代码，按指导书的要求运行，得到的结果如下所示。

首先执行 Seal 操作，结果如图 7 所示。

```
[06/22/2021 05:25] seed@ubuntu:~/Downloads/trusted-computing-projectv0.3/KeyHier
archy$ cd ..
[06/22/2021 05:26] seed@ubuntu:~/Downloads/trusted-computing-projectv0.3$ cd ./S
ealUnseal/
[06/22/2021 05:26] seed@ubuntu:~/Downloads/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 71 d3 5a 63 11 11 f5 fa D.C....q.ZC....
00000040| b0 8d 6d 69 69 32 c9 3a 72 c1 80 e6 b0 d3 dd 68 ..mi2.:r.....h
00000050| e8 3e 32 76 bf c6 2e 06 b3 48 d7 86 44 13 c5 87 .>2v....H..D...
00000060| 06 86 3b 52 f2 8f 70 c7 bc bd 9d f3 87 ef 40 12 ...R..p.....@.
00000070| 2e d4 8c 24 5c 4e ad 69 70 8c b8 b4 ce e4 df f1 ...$(N..lp.....
00000080| 06 36 51 5b 73 04 d2 10 c0 8b 96 3b 22 58 14 bc .6Q[s.....;X..
00000090| d6 65 c1 97 69 c7 69 a5 a4 ac c0 f2 63 10 92 ff .e..i.i.....C...
000000a0| d8 74 2e 84 d3 26 f9 a7 53 a5 f2 e0 ab a1 71 b7 .t...&..S....q.
000000b0| 6c 1b 18 b6 39 25 cc af 67 23 fb be 0d 97 f2 31 l...9%..g#.....1
000000c0| c7 7a 33 a8 0a 93 b0 53 2b b7 5f e4 ae 21 d1 bd .z3....S+...!...
000000d0| 12 30 7f dd fe 88 dc f1 63 83 d0 e2 cc 00 96 63 .0.....C.....c
000000e0| f4 e5 e5 13 e7 3d a6 65 5c 13 ad 1e 6f e9 78 85 .....=e\...O.X.
000000f0| 6b 92 84 0c ec 5a 31 f6 5f 2d 71 42 72 40 4d 7c k....Z1...-qBr@M|
00000100| f4 c9 7d 91 1c 40 d9 e8 a2 e2 c7 1f dd 2b 29 4f ..}..@.....+0
00000110| fa 87 b5 1a fb 80 1c 23 ce 2a d5 36 4f c8 0b 45 .....#.*.60..E
00000120| 64 bd 4f 8e 25 be d7 ab af bb 84 25 2f 1b 2a c4 d.O.%.....%/*..
00000130| 90 46 69 a3 4d 46 96 76 .Fi.MF.v
Success
```

图 7 seal 成功

接着 Unseal 得到对应数据，如图 8 所示。

```
[06/22/2021 05:29] seed@ubuntu:~/Downloads/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 71 d3 5a 63 11 11 f5 fa D.C....q.ZC....
00000040| b0 8d 6d 69 69 32 c9 3a 72 c1 80 e6 b0 d3 dd 68 ..mi2.:r.....h
00000050| e8 3e 32 76 bf c6 2e 06 b3 48 d7 86 44 13 c5 87 .>2v....H..D...
00000060| 06 86 3b 52 f2 8f 70 c7 bc bd 9d f3 87 ef 40 12 ...R..p.....@.
00000070| 2e d4 8c 24 5c 4e ad 69 70 8c b8 b4 ce e4 df f1 ...$(N..lp.....
00000080| 06 36 51 5b 73 04 d2 10 c0 8b 96 3b 22 58 14 bc .6Q[s.....;X..
00000090| d6 65 c1 97 69 c7 69 a5 a4 ac c0 f2 63 10 92 ff .e..i.i.....C...
000000a0| d8 74 2e 84 d3 26 f9 a7 53 a5 f2 e0 ab a1 71 b7 .t...&..S....q.
000000b0| 6c 1b 18 b6 39 25 cc af 67 23 fb be 0d 97 f2 31 l...9%..g#.....1
000000c0| c7 7a 33 a8 0a 93 b0 53 2b b7 5f e4 ae 21 d1 bd .z3....S+...!...
000000d0| 12 30 7f dd fe 88 dc f1 63 83 d0 e2 cc 00 96 63 .0.....C.....c
000000e0| f4 e5 e5 13 e7 3d a6 65 5c 13 ad 1e 6f e9 78 85 .....=e\...O.X.
000000f0| 6b 92 84 0c ec 5a 31 f6 5f 2d 71 42 72 40 4d 7c k....Z1...-qBr@M|
00000100| f4 c9 7d 91 1c 40 d9 e8 a2 e2 c7 1f dd 2b 29 4f ..}..@.....+0
00000110| fa 87 b5 1a fb 80 1c 23 ce 2a d5 36 4f c8 0b 45 .....#.*.60..E
00000120| 64 bd 4f 8e 25 be d7 ab af bb 84 25 2f 1b 2a c4 d.O.%.....%/*..
00000130| 90 46 69 a3 4d 46 96 76 .Fi.MF.v
Unsealed Data:
00000000| 30 31 32 33 34 35 36 37 38 39 41 42 43 44 45 46 0123456789ABCDEF
00000010| 30 31 32 33 34 35 36 37 38 39 41 42 43 44 45 46 0123456789ABCDEF
Success
[06/22/2021 05:30] seed@ubuntu:~/Downloads/trusted-computing-projectv0.3/SealUnseal$
```

图 8 unseal 成功

Extend 拓展 PCR 寄存器后再次 unseal，失败，如图 9 所示。

```

Success
[06/22/2021 05:30] seed@ubuntu:~/Downloads/trusted-computing-projectv0.3/SealUnseal$ ./extend -v 1.2
ulPcrValLen:20

Success
[06/22/2021 05:30] seed@ubuntu:~/Downloads/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 71 d3 5a 63 11 11 f5 fa D.c.....q.ZC...
00000040| b0 8d 6d 69 69 32 c9 3a 72 c1 80 e6 b0 d3 dd 68 ..mli2.:r.....h
00000050| e8 3e 32 76 bf c6 2e 06 b3 48 d7 86 44 13 c5 87 .>2v.....H..D...
00000060| 06 86 3b 52 f2 8f 70 c7 bc bd 9d f3 87 ef 40 12 ..;R..p.....@.
00000070| 2e d4 8c 24 5c 4e ad 69 70 8c b8 b4 ce e4 df f1 ...$\N.ip.....
00000080| 06 36 51 5b 73 04 d2 10 c0 8b 96 3b 22 58 14 bc .6Q[s.....;"X..
00000090| d6 65 c1 97 69 c7 69 a5 a4 ac c0 f2 63 10 92 ff .e..i.l.....c...
000000a0| d8 74 2e 84 d3 26 f9 a7 53 a5 f2 e0 ab a1 71 b7 .t...&..S.....q.
000000b0| 6c 1b 18 b6 39 25 cc af 67 23 fb be 0d 97 f2 31 l...9%..g#.....1
000000c0| c7 7a 33 a8 0a 93 b0 53 2b b7 5f e4 ae 21 d1 bd .z3...S+...!..
000000d0| 12 30 7f dd fe 88 dc f1 63 83 d0 e2 cc 60 96 63 .0.....c.....c
000000e0| f4 e5 e5 13 e7 3d a6 65 5c 13 ad 1e 6f e9 78 85 .....=e\...o.X.
000000f0| 6b 92 84 0c ec 5a 31 f6 5f 2d 71 42 72 40 4d 7c k....Z1._-qBr@M|
00000100| f4 c9 7d 91 1c 40 d9 e8 a2 e2 c7 1f dd 2b 29 4f ..}..@.....+ )0
00000110| fa 87 b5 1a fb 80 1c 23 ce 2a d5 36 4f c8 0b 45 .....#.*.60..E
00000120| 64 bd 4f 8e 25 be d7 ab af bb 84 25 2f 1b 2a c4 d.0.%.....%/.*.
00000130| 90 46 69 a3 4d 46 96 76 .Fi.MF.v
0 FAIL : Tspi_Data_Unseal returned (24) TPM_E_WRONGPCRVAL
unseal.c 0 FAIL : Tspi_Data_Unseal returned (24) TPM_E_WRONGPCRVAL
[06/22/2021 05:30] seed@ubuntu:~/Downloads/trusted-computing-projectv0.3/SealUnseal$

```

图 9 unseal 失败

Seal_test.en 文件内容，再 unseal 后查看内容，如图 10 所示。

```

[06/22/2021 05:30] seed@ubuntu:~/Downloads/trusted-computing-projectv0.3/SealUnseal$ ./seal_file test.c test.en
Input K1's Pin

Enter PIN:
[06/22/2021 05:30] seed@ubuntu:~/Downloads/trusted-computing-projectv0.3/SealUnseal$ ./unseal_file test.en test.de
bash: ./unseal_file: No such file or directory
[06/22/2021 05:31] seed@ubuntu:~/Downloads/trusted-computing-projectv0.3/SealUnseal$ sudo make
gcc -g -I../include -o unseal_file unseal_file.c ../common/common.o -ltspi -lcrypto
[06/22/2021 05:31] seed@ubuntu:~/Downloads/trusted-computing-projectv0.3/SealUnseal$ ./unseal_file test.en test.de
Input K1's Pin

Enter PIN:
Unsealed Data:
aa 36 52 ce e7 1a 84 c3 7d 82 28 02 92 58 f7 56
67 08 c5 8f 60 d5 31 b4 1b 3f 92 8d 0c 7a d4 6d
98 26 e1 ec eb df 99 60 5d 41 58 9d ba d6 12 3c
47 c4 de ba ca c7 a7 82 77 2b 12 94 5f 25 11 53
[06/22/2021 05:31] seed@ubuntu:~/Downloads/trusted-computing-projectv0.3/SealUnseal$

```

图 10 查看 seal 文件内容成功

PCR 拓展后再次 unseal 查看文件内容，失败，因为无法 unseal 得到加密的密钥，如图 11 所示。



图 11 查看 seal 文件内容失败

3.4 密钥迁移 (Key Migration)

密钥迁移操作用于将本地机器 TPM 上的可迁移密钥迁移到其他机器上。在本次实验中则是重新制作了另一个配置好环境的 ubuntu12.04 虚拟机，在两台机器间完成密钥迁移的过程。

密钥迁移的过程如图 12 所示。假定密钥从机器二迁移到机器一，机器一首先将自己的公钥封装成文件发送给机器二，机器二将需要移植的密钥的私钥部分利用 PKstorage 进行重新加密，用一个随机串打包，与其公钥和随机串一起封装成 blob 文件，发送给机器一，机器一解密后就得到了迁移密钥。

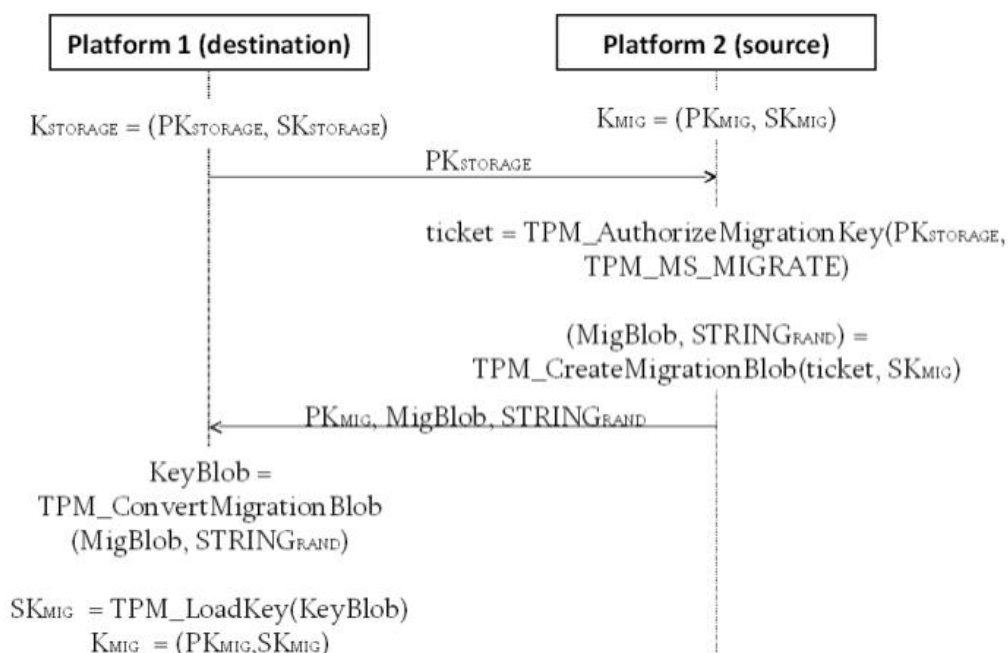


图 12 密钥迁移的过程

在样例代码中已经实现了前半部分，实验中需要完成的部分就是从 blob 文件获取迁移密钥的过程。首先，从 `mig.blob` 中可以得到一些随机串相关的信息、迁移块长度 `u32MigBlobLen` 和迁移块数据 `pMigBlob`。结合 `Tspi_Key_ConvertMigrationBlob` 函数，从迁移块中获取迁移密钥句柄 `hNewMigKey`，同时确定迁移密钥的父密钥为 `SRK`。调用 `Tspi_Key_LoadKey` 加载迁移密钥；调用 `sign_and_verify` 在上下文中对迁移密钥进行签名和验证，就可以完成密钥的迁移。

将完成修改的代码分别拷贝到两个虚拟机中编译运行。在机器一中运行 `./platform_dst -g`，产生公钥文件 `srk.pub`，将其移动到机器二中，运行 `./platform_src`。产生名为 `mig.blob` 的文件，再拷回到机器一中，并在机器一中运行 `./platform_dst -m`。最终的结果如图 13、14 所示，密钥迁移成功。

```

[06/22/2021 05:49] seed@ubuntu:~/Downloads/trusted-computing-projectv0.3/KeyMigration$ ./platform_src
Input Migration Key's Pin

Enter PIN:
OK
[06/22/2021 05:49] seed@ubuntu:~/Downloads/trusted-computing-projectv0.3/KeyMigration$
  
```

图 13 机器 2 生成 `mig.blob`

```

[06/22/2021 05:31] seed@ubuntu:~/Downloads/trusted-computing-projectv0.3/SealUnseal$ cd ..
[06/22/2021 05:42] seed@ubuntu:~/Downloads/trusted-computing-projectv0.3$ cd ./KeyMigration/
[06/22/2021 05:42] seed@ubuntu:~/Downloads/trusted-computing-projectv0.3/KeyMigration$ sudo make
gcc -g -I../include -o platform_dst platform_dst.c ../common/common.o -ltspi
[06/22/2021 05:42] seed@ubuntu:~/Downloads/trusted-computing-projectv0.3/KeyMigration$ ./platform_dst -g
Generating Pub Key Success
[06/22/2021 05:42] seed@ubuntu:~/Downloads/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:
9d 9d 85 89 aa 21 79 72 7f 7c 38 4d 61 e7 1d ca
0c 08 73 fa 37 f4 05 20 33 3d a8 e3 62 a0 48 ed
c9 b6 02 03 c9 31 6f b4 01 99 01 ed 19 00 5d 5c
50 44 1d 51 50 98 17 32 0e 2a a0 64 a0 1f ef d9
76 f4 1e 89 e8 7e 74 c6 b7 c4 12 d7 28 18 b5 c6
93 8b 7a 89 12 17 2c f0 93 14 0a 10 c9 d6 b0 0b
cb 55 fd 16 7b b2 6d 38 78 84 17 7f 4a 0b d6 75
54 19 97 67 15 50 67 55 03 e2 3f bb 12 8e fc 38
d8 8c e5 59 5b 38 72 3b 15 6d 13 24 7d 14 b2 11
9c 74 f4 bd 2a e8 72 b4 90 69 26 7c bb 8d 36 f3
56 5c 05 e4 22 6f 3d 85 86 c9 69 2f 4f ea 49 3c
4e c0 2f e4 c2 dd 2c e6 8a 41 ba 1b 43 98 d6 b7
c9 e4 07 1a 7a cd fe 75 d4 d3 a0 dd fb 85 69 1a
ff 32 fe 1a 80 68 cb 66 b0 f3 cb 40 17 5a fd ea
3c ec 35 f8 b7 30 5b a2 00 d1 08 76 71 24 50 9b
70 a8 3a 85 45 21 f9 68 10 49 00 ab 33 df ea ad
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
[06/22/2021 05:50] seed@ubuntu:~/Downloads/trusted-computing-projectv0.3/KeyMigration$

```

图 14 密钥迁移成功

3.5 远程证明 (Remote Attestation)

TPM 进行远程证明的实验同样需要用到两个虚拟机，这里我们仍用上述的两个虚拟机，配置好网络，进行实验。其中远程证明的原理如图 15 所示。

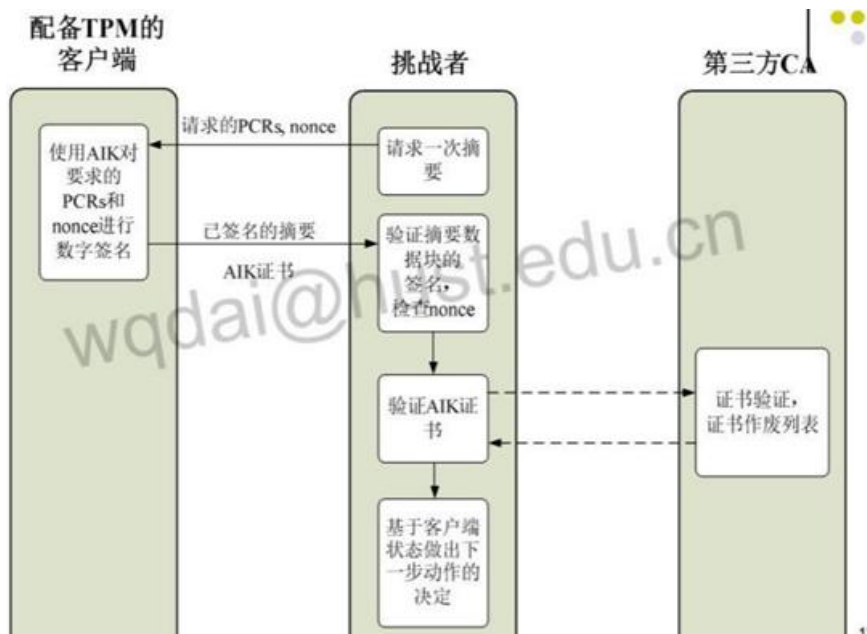


图 15 远程证明原理

首先在机器 1 中，进入 RemoteAttestation/init, 执行 Create_AIK 的操作，如图 16 所示。

```

[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_AIK.c:762 receive_msg_size is 44
Create_AIK.c:780 1 PASS : Create AIK returned (0) TSS_SUCCESS
786 Ending remote_attestation()
801 remote_data_size 44
803 keyPubSize 256
Create_AIK.c:805 ##The content of the keyPub is:##
d9 7b e9 42 c0 f8 be f2 39 5f 2f 71 ff 59 45 e7
62 03 89 5e bc d6 9c c4 08 4c a8 a2 74 09 38 1f
c1 41 4e d7 1f b8 bc 99 21 65 71 5f f5 c0 8a 7d
a5 41 e3 4f 66 41 5d e9 28 29 06 48 58 69 09 54
3f be a6 07 c9 36 4d 02 99 42 1a 59 b9 77 7a 6c
95 27 5c 4f 5e 91 5c 77 d7 66 4a 77 49 3d b6 36
7f 68 5f 26 4f 9a 07 28 ad 6e e5 00 f5 30 2c f1
ef e0 d1 67 2f 8e ad 81 9a 83 44 a9 30 df 67 77
ef f2 9f 48 7f 2b 47 47 f4 cf 7c 63 4a cb 11 b9
87 6a 85 b1 3c 15 4d eb df 59 af 8e b5 14 6c c7
44 7a 3e cf 07 55 e9 13 e7 55 10 99 06 24 30 56
d1 19 ae 7a a0 6f 40 4a 19 6a 80 c6 68 0d cb 9c
f7 0a 75 64 48 92 c9 63 5e 7f 96 d5 34 cc 0e 65
e6 11 16 34 5e 18 13 01 73 61 27 18 12 55 4d f9
29 3c e4 25 ca b9 2c b6 b3 f4 a7 53 11 c5 fc a0
08 59 26 64 e8 1a fd f4 3e 58 b7 a7 85 c9 f6 07
Create_AIK.c:807 ##The content of the keyPub is over##
809 credLen 104
Create_AIK.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 5a 5a 5a 5a 5a 5a 5a 5a
5a 5a 5a 5a 5a 5a 5a 5a 5a 5a 5a 5a 5a 5a 5a
Create_AIK.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 01 00 00 68 00 00 00
10 00 00 00 10 0f 6d 08 90 19 6d 08 98 1a 6d 08
00 00 00 00 00 00 00 00 30 00 00 00
819 -- receive msg --
[06/22/2021 05:53] seed@ubuntu:~/Downloads/trusted-computing-projectv0.3/RemoteAttestation/init$

```

图 16 ./Create_AIK

返回到 RemoteAttestation 目录下，创建 RAServer，如图 17。

```

5a 5a 5a 5a 5a 5a 5a 5a
Create_AIK.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 01 00 00 68 00 00 00
10 00 00 00 10 0f 6d 08 90 19 6d 08 98 1a 6d 08
00 00 00 00 00 00 00 00 30 00 00 00
819 -- receive msg --
[06/22/2021 05:53] seed@ubuntu:~/Downloads/trusted-computing-projectv0.3/RemoteAttestation/init$ cd ..
[06/22/2021 05:53] seed@ubuntu:~/Downloads/trusted-computing-projectv0.3/RemoteAttestation$ make clean
rm -f *.o RAClient RAServer
make -C init/ clean
make[1]: Entering directory '/home/seed/Downloads/trusted-computing-projectv0.3/RemoteAttestation/init'
rm -f *.o ../bin/Create_AIK *.bbg *.bb *.da
make[1]: Leaving directory '/home/seed/Downloads/trusted-computing-projectv0.3/RemoteAttestation/init'
[06/22/2021 05:54] seed@ubuntu:~/Downloads/trusted-computing-projectv0.3/RemoteAttestation$ make
gcc -c RAServer.c -I../include
gcc -c RACom.c
gcc -c remote_attestation.c -I../include
gcc -o RAServer RAServer.o RACom.o remote_attestation.o ../common/common.o -L/usr/local/lib -ltspi -lssl -lcrypto
gcc -c RAClient.c -I../include
RAClient.c: In function 'main':
RAClient.c:419:5: warning: format '%d' expects argument of type 'int', but argument 2 has type 'long unsigned int' [-Wformat]
RAClient.c:420:5: warning: format '%d' expects argument of type 'int', but argument 2 has type 'long unsigned int' [-Wformat]
RAClient.c:437:5: warning: format '%d' expects argument of type 'int', but argument 2 has type 'long unsigned int' [-Wformat]
RAClient.c:442:5: warning: format '%d' expects argument of type 'int', but argument 2 has type 'long unsigned int' [-Wformat]
RAClient.c:447:5: warning: format '%d' expects argument of type 'int', but argument 2 has type 'long unsigned int' [-Wformat]
RAClient.c:524:5: warning: format '%d' expects argument of type 'int', but argument 2 has type 'long unsigned int' [-Wformat]
gcc -o RAClient RAClient.o RACom.o ../common/common.o -lssl -L/usr/local/lib -ltspi -lssl -lcrypto
make -C init/
make[1]: Entering directory '/home/seed/Downloads/trusted-computing-projectv0.3/RemoteAttestation/init'
gcc -g -I../include -o Create_AIK Create_AIK.c ../common/common.o -ltspi
Create_AIK.c: In function 'main':
Create_AIK.c:803:3: warning: format '%d' expects argument of type 'int', but argument 3 has type 'long unsigned int' [-Wformat]
Create_AIK.c:809:3: warning: format '%d' expects argument of type 'int', but argument 3 has type 'long unsigned int' [-Wformat]
make[1]: Leaving directory '/home/seed/Downloads/trusted-computing-projectv0.3/RemoteAttestation/init'
[06/22/2021 05:54] seed@ubuntu:~/Downloads/trusted-computing-projectv0.3/RemoteAttestation$ ./RAServer
bash: ./RAServer: No such file or directory
[06/22/2021 05:54] seed@ubuntu:~/Downloads/trusted-computing-projectv0.3/RemoteAttestation$ ./RAServer

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

```

图 17 ./RAServer

接着在机器 2 中同样进入该目录下，执行 `./RAClient 192.168.94.156 192.168.94.155`，其中 192.168.94.156 和 192.168.94.155 分别是机器 2 和机器 1 的 IP，如图 18 所示。

```
[06/22/2021 05:55] seed@ubuntu:~/Downloads/trusted-computing-projectv0.3/RemoteA
ttestation$ ./RAClient 192.168.94.156 192.168.94.155
source IP: 192.168.94.155
###
--> --> Dealing with No.1: 192.168.94.155
===== Start to Communicate with RAServer =====
#####type: 2
===== Check Status DONE: CONNECTION Permitted! =====
sizeof_rc=====88

keyPubSize=====256

===== The length of the PCR response(buffer) is: 664 =====
RAClient.c:433 ##The content of the PCR response buffer is:##
30 00 00 00 00 01 00 00 00 01 00 00 58 00 00 00
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
5f c2 14 cd 04 f1 5a 64 32 24 c8 e3 b9 83 4f c5
92 1e c6 4e 62 9d e8 e8 77 16 6e 60 c3 38 57 5d
26 e3 84 6a 15 4c 2f 32 79 87 24 40 4d 5e 0c d9
c3 d9 6c d9 34 d4 ac b9 14 ba 97 9c b6 4b 4b 20
27 24 e5 7b e1 04 cc 8e 26 ff 73 9f 42 e2 aa bd
29 50 4c cc 21 a1 8d 61 3e 45 66 55 ac cf 89 fa
54 a4 5c 85 6e fa c6 88 15 06 51 18 7f c0 2d 3e
1b ec d6 79 30 72 13 27 3e f9 bc 48 3d 51 b8 4d
51 95 47 58 05 7a 1e b9 0c 90 2b 9f d5 fb 96 80
86 7e 66 fd 01 20 d3 91 a3 47 24 78 03 93 c2 46
9b 2f 11 18 7a cb bc e3 c6 fe 62 16 60 4f 58 ad
86 0a bc 37 19 73 b4 ac eb 46 0c 9b 28 43 3c 54
```

图 18 `./RAClient`

在机器一上，可以看到 `nonce match` 的提示，表示远程证明成功。

```
ef e0 d1 67 2f 8e ad 81 9a 83 44 a9 30 df 67 77
ef f2 9f 48 7f 2b 47 47 f4 cf 7c 63 4a cb 11 b9
87 6a 85 b1 3c 15 4d eb df 59 af 8e b5 14 6c c7
44 7a 3e cf 07 55 e9 13 e7 55 10 99 06 24 30 56
d1 19 ae 7a a0 6f 40 4a 19 6a 80 c6 68 0d cb 9c
f7 0a 75 64 48 92 c9 63 5e 7f 96 d5 34 cc 0e 65
e6 11 16 34 5e 18 13 01 73 61 27 18 12 55 4d f9
29 3c e4 25 ca b9 2c b6 b3 f4 a7 53 11 c5 fc a0
08 59 26 64 e8 1a fd f4 3e 58 b7 a7 85 c9 f6 07
RAClient.c:445 ##The content of the keyPub is over##

sizeof_rc: 88

RAClient.c:448 ##The content of the rc is:##
00 02 00 53 00 00 50 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 00 00 00
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! =====
[06/22/2021 05:56] seed@ubuntu:~/Downloads/trusted-computing-projectv0.3/RemoteAttestation$
[06/22/2021 06:14] seed@ubuntu:~/Downloads/trusted-computing-projectv0.3/RemoteAttestation$
```

图 19 远程证明成功

4. 实验心得

这次 TPM 可信模块的编程实验，加深了我对 TPM 的理解，帮助我对于可信计算课程上学到的东西有了更深入感悟。实验中借助 TPM Emulator 模拟真实的 TPM 硬件环境，通过编程完成了密钥的生成与加载，并组成了一定的密钥层次，在此基础上又完成了 Seal/Unseal，密钥迁移和远程证明的功能。这些功能实际上就是 TPM 的核心内容，TPM 芯片的核心功能正是提供可信的密钥，对数据做加密保护，以及提供本地证明和远程证明保证安全性。

在实验中，我加强了对 TSS 软件栈结构的学习，对相关的 API 有了更多的了解，能够在查阅资料的情况下，进行初步的 TPM 环境配置与利用这些 api 编程实现指定功能，阅读代码和编程的能力得到了一定锻炼。

实验中有不少的重点难点。密钥层次的实验相对简单，涉及到的 api 和补充的代码都不多，按照前面密钥生成的步骤改写一下即可。密钥生成与加载则要弄清楚整个的结构，包括创建上下文，连接上下文，加载密钥，获得策略，验证等等。这些是生成 TPM 可信环境的固定步骤，有些在样例代码中已经给出，有些则需要自己模仿着完成。后面在 Seal/Unseal 部分，就遇到了一些困难。这里涉及到的操作相对复杂，在阅读代码时花了很多的时间，一边查找涉及到的 api，一边思考意义。最终用流程图的方式把整个 Seal/Unseal 过程，尤其是对文件的 seal_file 的代码，相对清晰的还原了出来，才成功的弄清楚需要编写的 unseal_file.c 的结构。由于密文文件结构比较复杂，一开始在读文件获取信息时出现了很多错误，后面才弄清楚这些数据的类型并依次读取，最终完成了该部分实验。最后的密钥迁移和远程证明的学习过程也类似，都是在理解原理的基础上，才能动手完成。

总的来说，这次实验中遇到了不少困难，也有很多收获。TPM 的编程是以前没有接触过的，一方面在搜集资料的过程中，我得到了实践的锻炼；另一方面，在思考问题结构和编写代码时，我也对 TPM 中的相关问题有了更深的思考，一些之前有疑问的地方也豁然开朗。由于要查询的资料比较多，实验完成的过程中也要感谢老师、助教和同学们的指导、交流与帮助。