

**可信计算实验报告**

|  |  |
| --- | --- |
| 姓 名： |  |
| 学 院： | 网络空间安全学院 |
| 专 业： | 网络空间安全 |
| 班 级： |  |
| 学 号： |  |
| 指导教师： |  |

|  |  |
| --- | --- |
| 分数 |  |
| 教师签名 |  |

2024年6月15日

目 录

[1、实验目的 - 1 -](#_Toc735300122)

[2、实验内容以及流程说明 - 2 -](#_Toc1572133282)

[3、实验心得及建议 - 11 -](#_Toc157741735)

# 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tmpm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，这个工作主要在秘钥迁移和秘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、实验内容以及流程说明

* 1. 实验环境

Ubuntu 12.04 LTS

TPM Emulator

Trousers

* 1. 编译

在没有物理TPM的机器上，可以使用TPM Emulator完成TPM的相关实验。解压并编译安装TPM Emulator：

tar -zxvf 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

* 1. 初始化

编译本次实验源码：

tar -zxvf trusted-computing-projectv0.3.tar.gz

cd trusted-computing-projectv0.3

make clean

make

进行初始化操作，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 tcsd。tcsd 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。

​ 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。

sudo modprobe tpmd\_dev

sudo tpmd -f -d clear

# 再打开一个终端(注意：前面tpmd那个终端不要关)，运行

sudo tcsd

cd trusted-computing-projectv0.3/init

./Tspi\_TPM\_TakeOwnership01 -v 1.2

./create\_mig\_key -v 1.2 # 输入pin

如果以上所有安装流程正常，在init阶段的输出结果如下图所示：

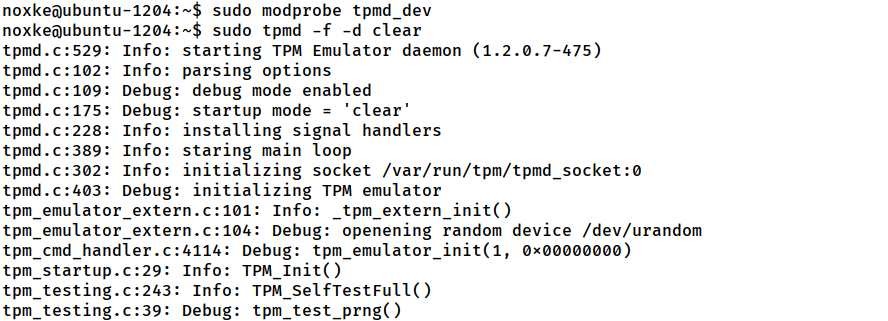


图2-1 终端1 tpmd

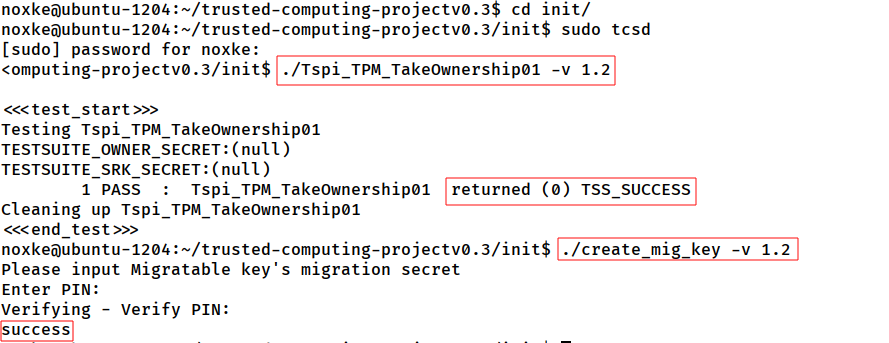


图2-2 初始化

* 1. 秘钥层次（KeyHierarchy）

​ 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，是最高权限的存储密钥，一个TPM仅存在一个，所有其他密钥都在存储根密钥的保护之下。

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。

参考K1、K2、K3的创建过程，以及TSS文档，完善create\_register\_key.c中创建K4的代码。

1. /\* =============================================================== \*/
2. initFlags = TSS\_KEY\_TYPE\_BIND | TSS\_KEY\_SIZE\_2048 |
3. TSS\_KEY\_VOLATILE | TSS\_KEY\_AUTHORIZATION |
4. TSS\_KEY\_MIGRATABLE;
5. result = my\_create\_load\_key(hContext, initFlags, hKey3, &hKey4, "K4");
6. **if** (result != TSS\_SUCCESS) {
7. print\_error("create\_key", result);
8. Tspi\_Context\_FreeMemory(hContext, NULL);
9. Tspi\_Context\_Close(hContext);
10. exit(result);
11. }
12. result = Tspi\_Context\_RegisterKey(hContext, hKey4,
13. TSS\_PS\_TYPE\_SYSTEM,
14. UUID\_K4,
15. TSS\_PS\_TYPE\_SYSTEM,
16. UUID\_K3);
17. **if** (result != TSS\_SUCCESS) {
18. print\_error("Tspi\_Context\_RegisterKey", result);
19. Tspi\_Context\_FreeMemory(hContext, NULL);
20. Tspi\_Context\_Close(hContext);
21. exit(result);
22. }
23. /\* =============================================================== \*/

在完成密钥创建后，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。

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。运行create\_register\_key，如图2-3所示

cd KeyHierarchy

make

./create\_register\_key -v 1.2

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。

1. /\* =============================================================== \*/
2. result = Tspi\_Context\_GetKeyByUUID( hContext, TSS\_PS\_TYPE\_SYSTEM,
3. UUID\_K4,
4. &hKey4 );
5. **if** (result != TSS\_SUCCESS) {
6. print\_error("Tspi\_Context\_LoadKeyByUUID", result);
7. print\_error\_exit(nameOfFunction, err\_string(result));
8. Tspi\_Context\_FreeMemory(hContext, NULL);
9. Tspi\_Context\_Close(hContext);
10. exit(result);
11. }
12. /\* =============================================================== \*/

运行load\_key，如图2-4所示

./load\_key -v 1.2

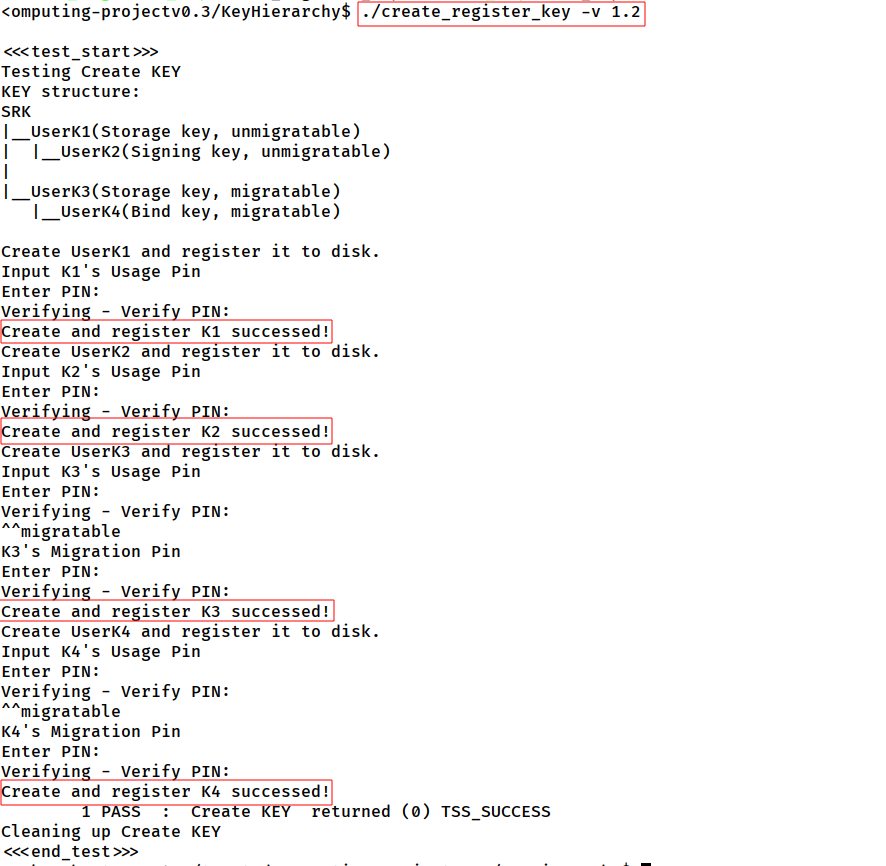


图2-3 创建register key

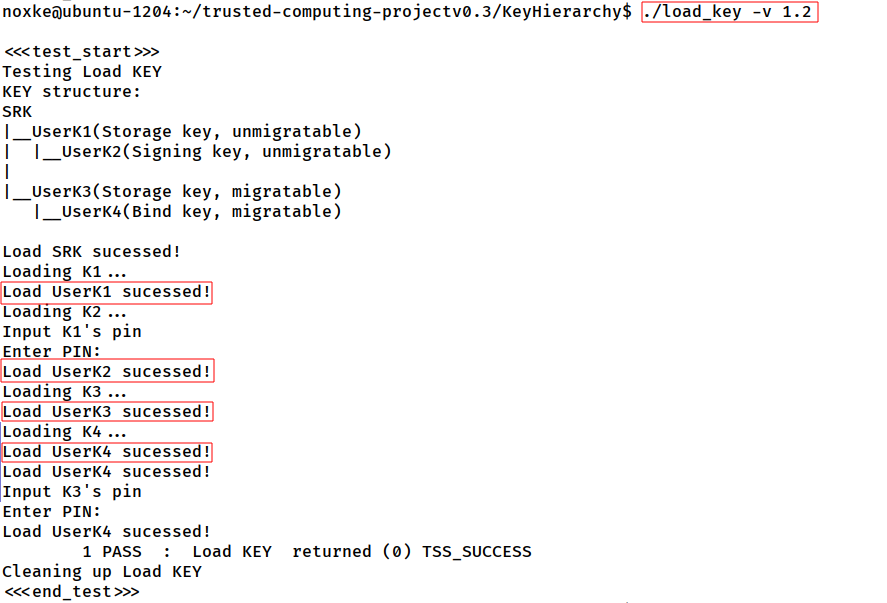


图2-4 load key

* 1. 秘钥迁移Seal、Unseal和extend

进入SealUnseal目录实验需要完成unseal\_file.c文件，完成后在trusted-computing-projectv0.3文件夹中make。

unseal\_file 根据seal\_file.c来还原即可，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。

./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）

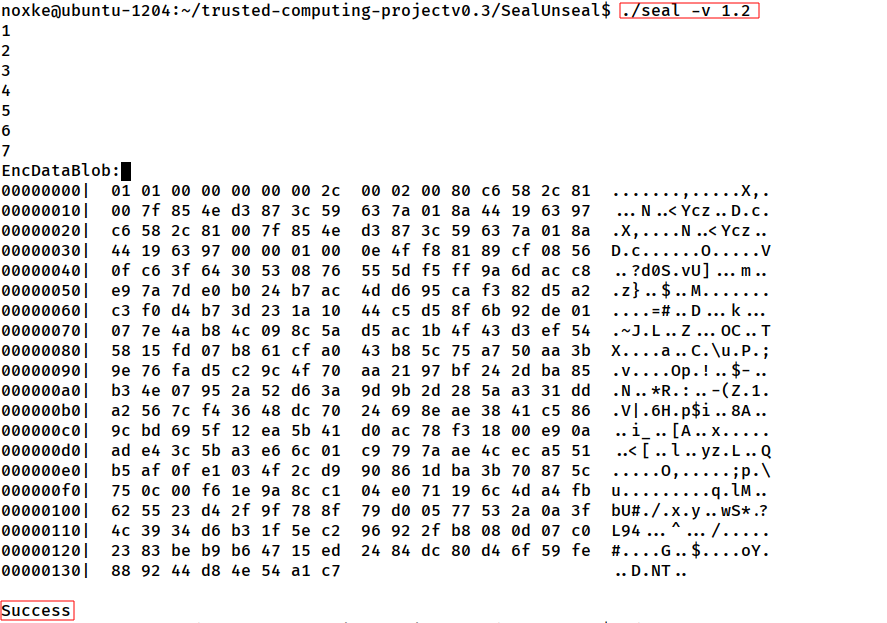


图2-5 seal 成功

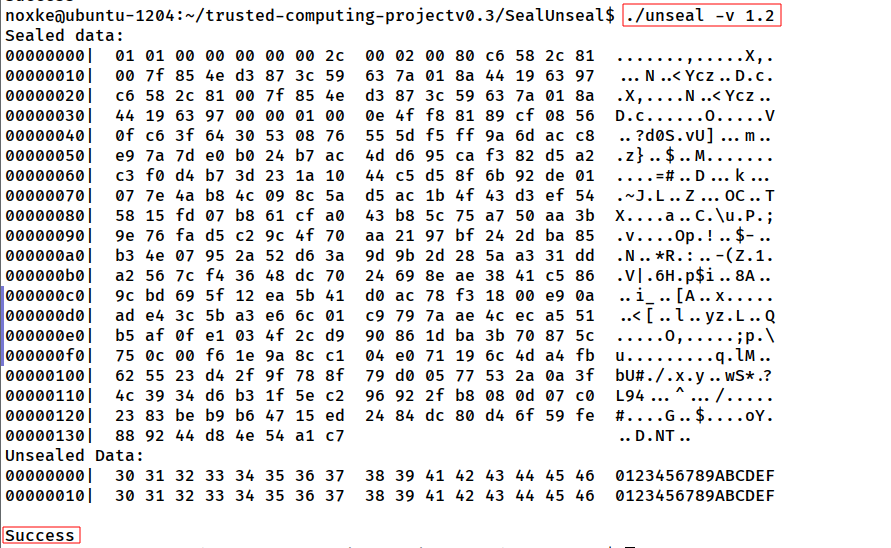


图2-6 unseal 成功

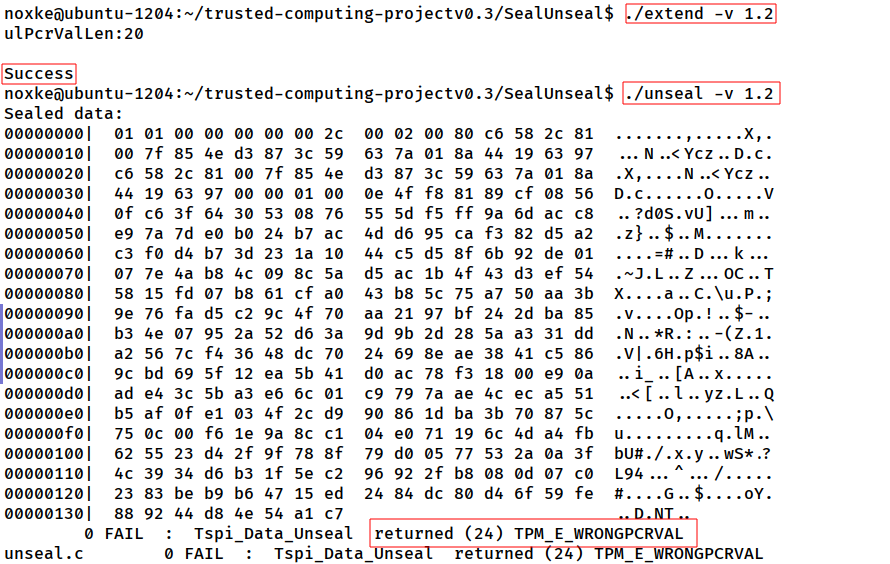


图2-7 extend 成功 unseal失败

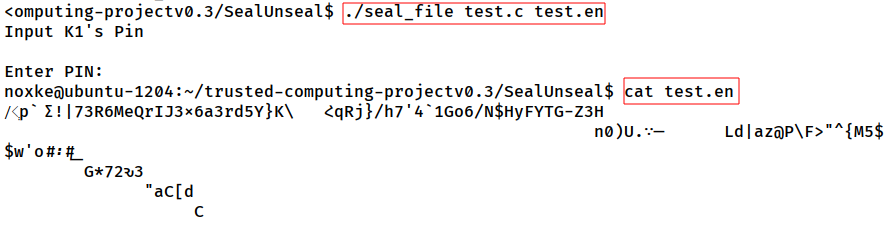


图2-8 seal\_file



图2-9 unseal\_file 成功

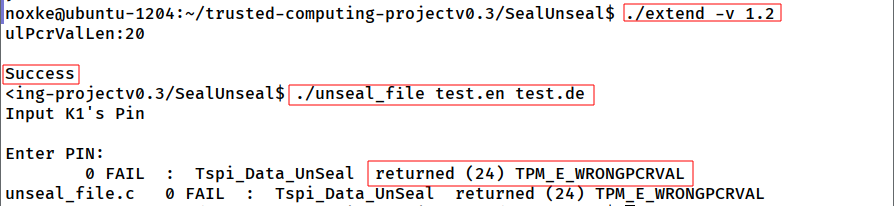


图2-10 extend unseal\_file 失败

* 1. 秘钥迁移（KeyMigration）

进入Key Migration，实验需要完成platform\_dst.c中的部分代码。

密钥从机器2迁移到机器1，机器1首先将自己的公钥封装成文件发送给机器2，机器2将需要移植的秘钥的私钥部分利用 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1，机器1解密后就得到了迁移密钥。

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。

1. /\* =============================================================== \*/
2. result = Tspi\_Key\_ConvertMigrationBlob(hNewMigKey,
3. hSRK,
4. u32RandomLen,
5. pRandom,
6. u32MigBlobLen,
7. pMigBlob);
8. /\* =============================================================== \*/

在机器1中运行

./platform\_dst -g

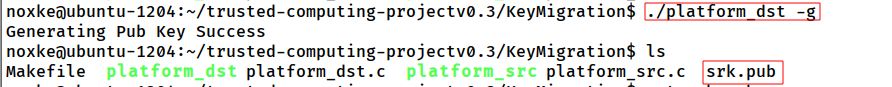


图2-11 机器1

产生名为srk.pub的文件，把文件srk.pub拷贝到机器2中，运行

./platform\_src

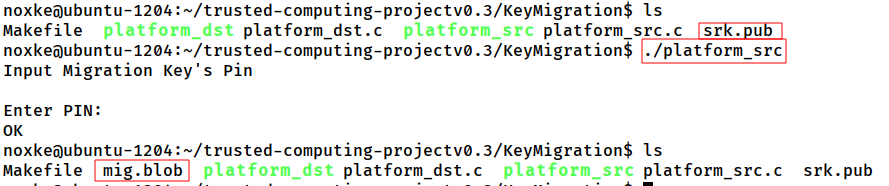


图2-12 机器2

产生名为mig.blob的文件，文件mig.blob拷回到机器1中，在机器1中运行

./platform\_dst -m



图2-13 机器1

* 1. 远程证明（RemoteAttestation）

机器1：192.168.56.19

1、进入Remote Attestation\init目录

2、运行./Create\_AIK

3、返回上级目录

4、运行./RAServer

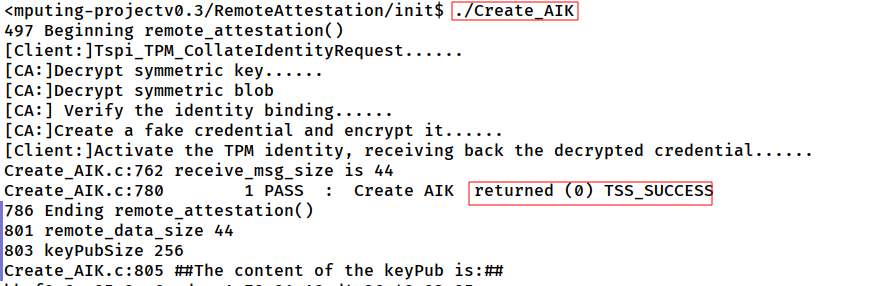


图2-14 Create\_AIK



图2-15 RAServer

机器2：192.168.56.20

1、进入Remote Attestation目录

2、运行./RAClient 机器2的ip 机器1的ip

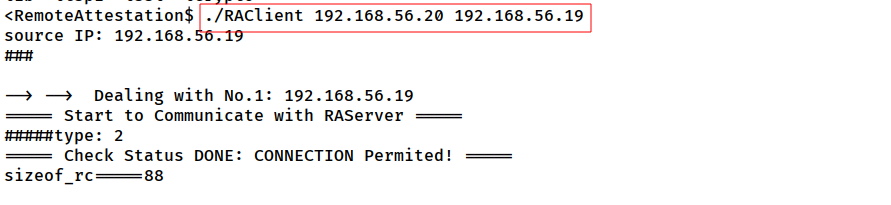


图2-16 RAClient

机器2 RAClient显示nonce match! 说明远程证明成功，如图2-17所示。

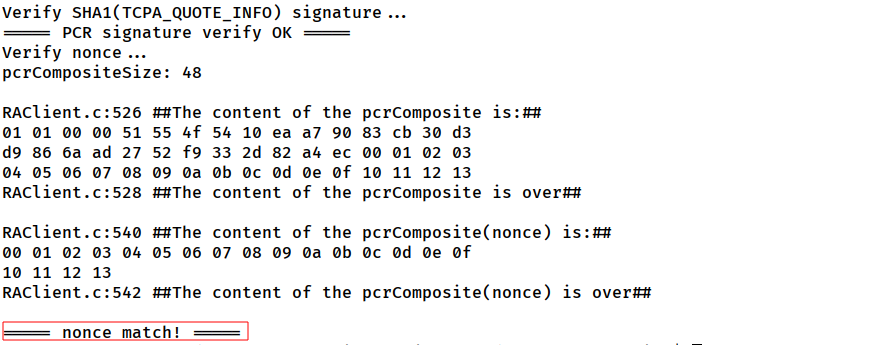


图2-17 远程证明

# 3、实验心得及建议

本次实验基于TPM Emulator，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。

实验文档较为详细，按照实验文档操作即可完成实验内容。在KeyHierarchy部分，需要补充create\_register\_key.c和load\_key.c代码中有关K4的部分，参考代码中关于K1、K2、K3的部分可以较容易地实现。在密钥迁移部分，需要完整编写unseal\_file代码，由于没有实例代码，实现时遇到部分困难，由于没有参考代码，只能通过研究seal\_file的实现，逆向分析出unseal\_file的实实现，反复调整测试后，终于实现了unseal\_file。

实验中遇到的另一个问题是实验环境的搭建，由于习惯使用较新的开发环境，一开始使用Ubuntu 24.04虚拟机，编译时tpm emulator和实验代码都无法通过编译，切换到Ubuntu 22.04后，仍然无法通过编译，继续降级使用Ubuntu 20.04测试，tpm emulator能够通过编译，但实验代码无法通过编译。最后按照文档切换到Ubuntu 12.04后，tpm emulator和实验代码都能够正常编译，继续进行实验时，发现unseal功能始终异常，无法成功解密，大量调试测试后无果，最终猜测可能是64位操作系统的原因，切换到32位Ubuntu 12.04后，实验终于顺利进行。

本实验的难度和内容都很合适，适合学习练习，但实验环境相对落后，实验仅能在32位Ubuntu 12.04环境运行，但Ubuntu 12.04已停止维护，可用软件源较少，相应的文档支持也较少，希望后续课程能够对实验进行调整，使实验能够在较新平台上进行。