学号 专业 信息安全 姓名

实验日期 **2024.10.29** 教师签字 成绩

实验报告

【实验名称】 可信计算第二次实验

【实验目的】

在上一次的实验中，我们实际上已经掌握了信任链构建的核心，那么如何在真实场景中构建信任链呢？我们上一次的实验，其实模拟的就是静态信任链的构建和验证，如果验证不通过，就退出程序了，那么，本次实验内容：

在上次实验的基础上，增加PCR寄存器的功能，假设每个PCR寄存器可以存储256比特数据，共4个PCR寄存器，当信任链验证成功部分结果后，更新对应的寄存器数值：PCR1存储文件1（模拟BIOS）的哈希，PCR2存储文件2（模拟OSLoader）的哈希，PCR3存储文件3（模拟OS）的哈希，PCR4存储文件4（模拟应用）的哈希，据此实现动态信任链构建功能：

设置多个待启动的应用（EXE1和EXE2）

程序启动后，首先验证文件1-文件3的正确性（信任链），并据此初始化PCR1-PCR3

程序命令行输出提示，让用户选择要启动的应用

程序根据选择，首先清空PCR4，然后验证对应应用的信任链，根据验证结果初始化PCR4值、运行程序，或者不运行程序、提醒用户程序被修改

上述结束后，重新输出提示，让用户选择要启动的应用，循环往复，或用户选择关闭程序（模拟关机）

建议：在等待输入时，可选择替换程序为错误的程序，以让验证失败，以展示信任链验证效果

【实验原理】

可信计算技术旨在通过硬件和软件相结合的方式，确保系统的完整性和安全性。在可信计算中，信任链（Trust Chain）和平台配置寄存器（PCR，Platform Configuration Register）是关键概念。本次实验的目的在于：

1. **模拟动态信任链的构建与验证**：在上一次实验中，我们实现了静态信任链的构建与验证，但仅在验证失败时退出程序。本次实验则进一步模拟真实场景中的动态信任链，通过模拟PCR寄存器，实现对系统组件和应用程序的可信验证。
2. **模拟TPM芯片的工作机制**：通过在程序中模拟PCR寄存器的初始化、更新和清空过程，体验TPM芯片在系统启动和应用程序加载时的安全保障作用。

具体实现方法如下：

 **PCR寄存器的模拟：**

* 使用string PCR[4];数组来模拟4个256位的PCR寄存器。

 **哈希计算函数：**

* computeFileHash：计算单个文件的SHA-256哈希值。
* computeHash：计算字符串和文件内容连接后的SHA-256哈希值。
* toHexString：将二进制哈希值转换为十六进制字符串，方便显示。

 **信任链验证函数：**

* verifyTrustChain：验证文件1-3的信任链，成功后更新PCR1-PCR3。
* 在每一步验证成功后，使用computeFileHash计算对应文件的哈希值，并存入对应的PCR寄存器。

 **清空PCR4的函数：**

* clearPCR4：将PCR[3]置为空字符串，表示清空PCR4。

 **运行应用程序的函数：**

* runApplication：根据用户选择的应用程序，清空PCR4，验证应用程序的信任链，更新PCR4，并运行程序。
* 应用程序的哈希值存储在EXE1\_hashes.txt或EXE2\_hashes.txt中，格式与之前的hashes.txt相同。

 **主程序逻辑：**

* 启动程序后，首先读取hashes.txt，获取文件1-4的预存哈希值。
* 调用verifyTrustChain验证文件1-3的信任链，并初始化PCR1-PCR3。
* 进入循环，提示用户选择要运行的应用程序或退出程序。
* 根据用户选择，调用runApplication函数。

【实验内容】

环境：Win11

语言：C++11

密码学库：cryptopp

#include <iostream>

#include <fstream>

#include <string>

#include <sha.h>

#include <hex.h>

#ifdef \_WIN32

#include <windows.h>

#else

#include <unistd.h>

#endif

using namespace CryptoPP;

using namespace std;

// 模拟PCR寄存器

string PCR[4];

string computeFileHash(const string& filename) {

    SHA256 hash;

    string digest;

    ifstream file(filename, ios::binary);

    if (!file) {

        cerr << "无法打开文件：" << filename << endl;

        exit(1);

    }

    string contents((istreambuf\_iterator<char>(file)), istreambuf\_iterator<char>());

    StringSource ss(contents, true,

        new HashFilter(hash,

            new HexEncoder(

                new StringSink(digest), false)));

    return digest;

}

// 计算字符串和文件的哈希值

string computeHash(const string& data, const string& filename) {

    SHA256 hash;

    string digest;

    ifstream file(filename, ios::binary);

    if (!file) {

        cerr << "无法打开文件：" << filename << endl;

        exit(1);

    }

    string contents((istreambuf\_iterator<char>(file)), istreambuf\_iterator<char>());

    string combined = data + contents;

    StringSource ss(combined, true,

        new HashFilter(hash,

            new HexEncoder(

                new StringSink(digest), false)));

    return digest;

}

// 将哈希值转换为十六进制字符串

string toHexString(const string& hash) {

    string hexDigest;

    HexEncoder encoder(new StringSink(hexDigest));

    encoder.Put((const byte\*)hash.data(), hash.size());

    encoder.MessageEnd();

    return hexDigest;

}

// 验证信任链并更新PCR寄存器

bool verifyTrustChain(const string& appName, const string& h1, const string& h2, const string& h3, const string& h4) {

  //   验证H1

    string computedH1 = computeHash(h2, "file1.txt");

    if (computedH1 != h1) {

        cerr << "文件1验证失败！" << endl;

        return false;

    }

   //  更新PCR1

    PCR[0] = computeFileHash("file1.txt");

    // 验证H2

    string computedH2 = computeHash(h3, "file2.txt");

    if (computedH2 != h2) {

        cerr << "文件2验证失败！" << endl;

        return false;

    }

   //  更新PCR2

    PCR[1] = computeFileHash("file2.txt");

 //    验证H3

    string computedH3 = computeHash(h4, "file3.txt");

    if (computedH3 != h3) {

        cerr << "文件3验证失败！" << endl;

        return false;

    }

    // 更新PCR3

    PCR[2] = computeFileHash("file3.txt");

    cout << "文件1-3验证成功，PCR1-PCR3已更新。" << endl;

    return true;

}

void clearPCR4() {

    PCR[3] = "";

    cout << "PCR4已清空。" << endl;

}

// 运行应用程序

void runApplication(const string& appName) {

    string appFile = appName + ".exe"; // Windows下使用.exe

#ifndef \_WIN32

    appFile = "./" + appName; // Linux下使用可执行文件

#endif

 //    假设应用程序的信任链哈希值已预先存储在对应的哈希文件中

    string hashFileName = appName + "\_hashes.txt";

    ifstream hashFile(hashFileName);

    if (!hashFile) {

        cerr << "无法打开哈希文件：" << hashFileName << endl;

        return;

    }

    string h1, h2, h3, h4;

    getline(hashFile, h1);

    // 清空PCR4

    clearPCR4();

  //   验证应用程序的信任链

  //   验证H4

    string computedH4 = computeFileHash(appFile);

    if (computedH4 != h1) {

        cerr << "应用程序验证失败！程序可能已被修改。" << endl;

        return;

    }

  //   更新PCR4

    PCR[3] = computedH4;

    cout << "应用程序验证成功，正在运行 " << appName << "..." << endl;

    cout << "当前PCR值（十六进制显示）：" << endl;

    for (int i = 0; i < 4; ++i) {

        cout << "PCR" << i + 1 << ": " << toHexString(PCR[i]) << endl;

    }

     //运行应用程序

#ifdef \_WIN32

    system(appFile.c\_str());

#else

    system(appFile.c\_str());

#endif

}

int main() {

     //假设信任链的哈希值已预先存储在hashes.txt中

    ifstream hashFile("hashes.txt");

    if (!hashFile) {

        cerr << "无法打开哈希文件：hashes.txt" << endl;

        exit(1);

    }

    string h1, h2, h3, h4;

    getline(hashFile, h1);

    getline(hashFile, h2);

    getline(hashFile, h3);

    getline(hashFile, h4);

     //首先验证文件1-3的信任链，初始化PCR1-PCR3

    if (!verifyTrustChain("file", h1, h2, h3, h4)) {

        cerr << "系统关键文件验证失败，程序即将退出。" << endl;

        exit(1);

    }

    while (true) {

        cout << endl;

        cout << "请选择要启动的应用程序：" << endl;

        cout << "1. EXE1" << endl;

        cout << "2. EXE2" << endl;

        cout << "0. 退出程序" << endl;

        cout << "请输入选项（0-2）：";

        int choice;

        cin >> choice;

        if (choice == 0) {

            cout << "程序已退出，模拟关机。" << endl;

            break;

        }

        else if (choice == 1) {

            runApplication("EXE1");

        }

        else if (choice == 2) {

            runApplication("EXE2");

        }

        else {

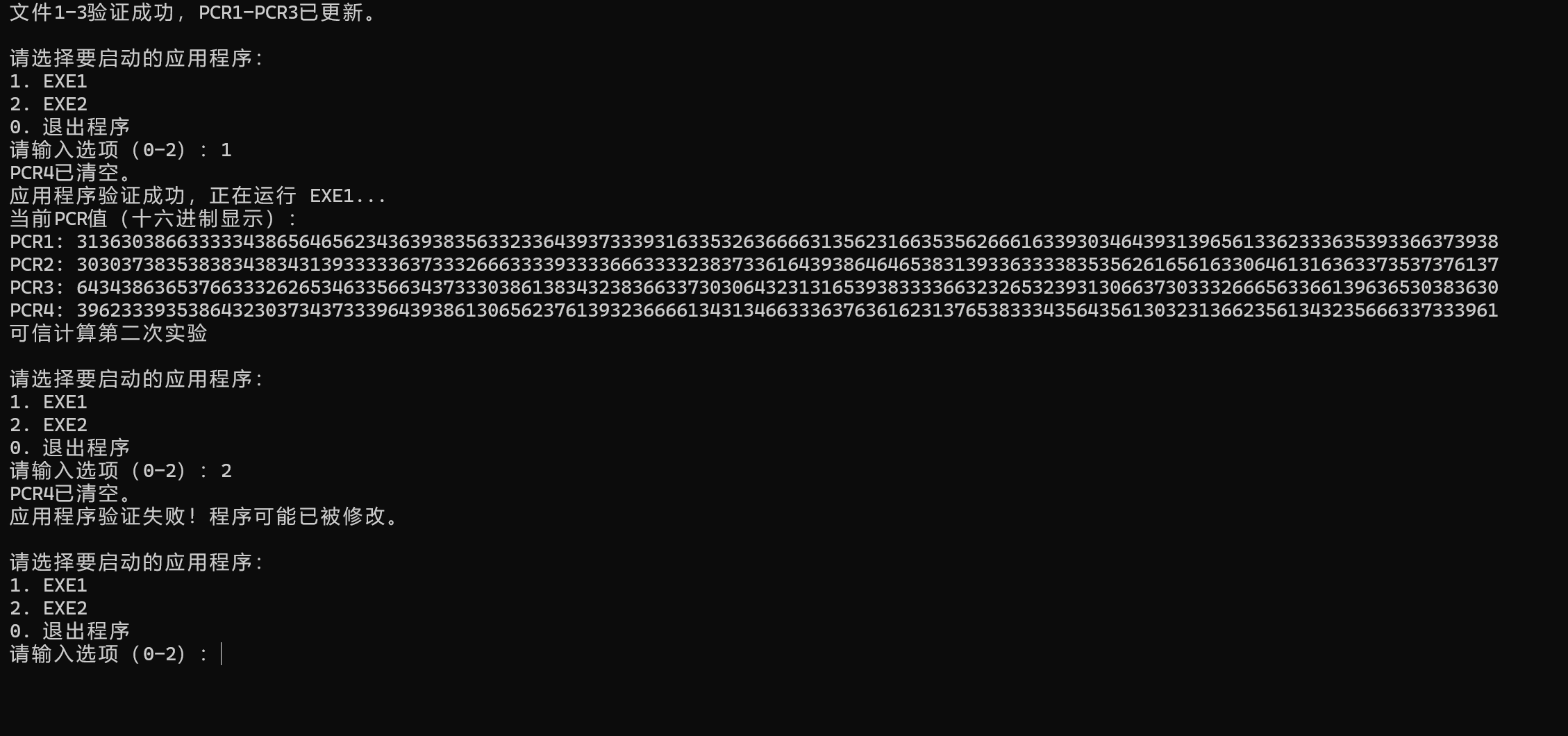
            cout << "无效的选项，请重新输入。" << endl;

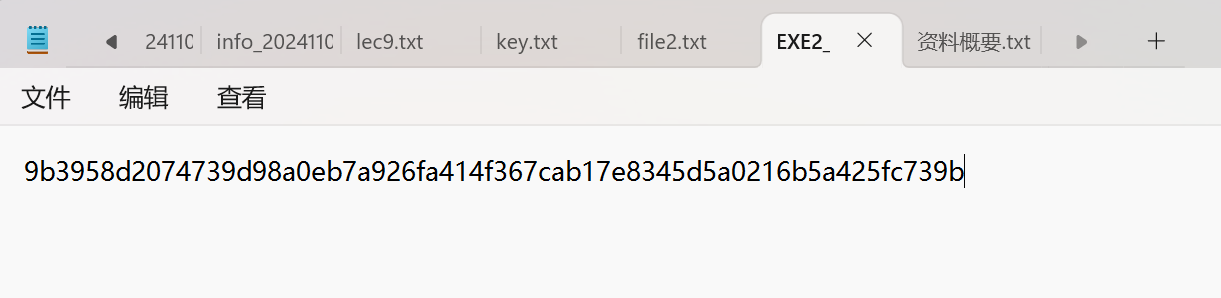
        }

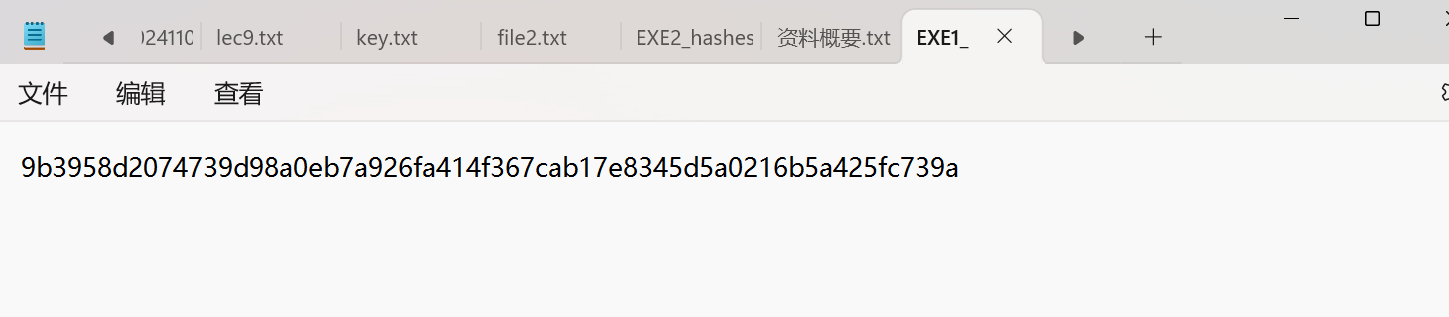
    }

    return 0;

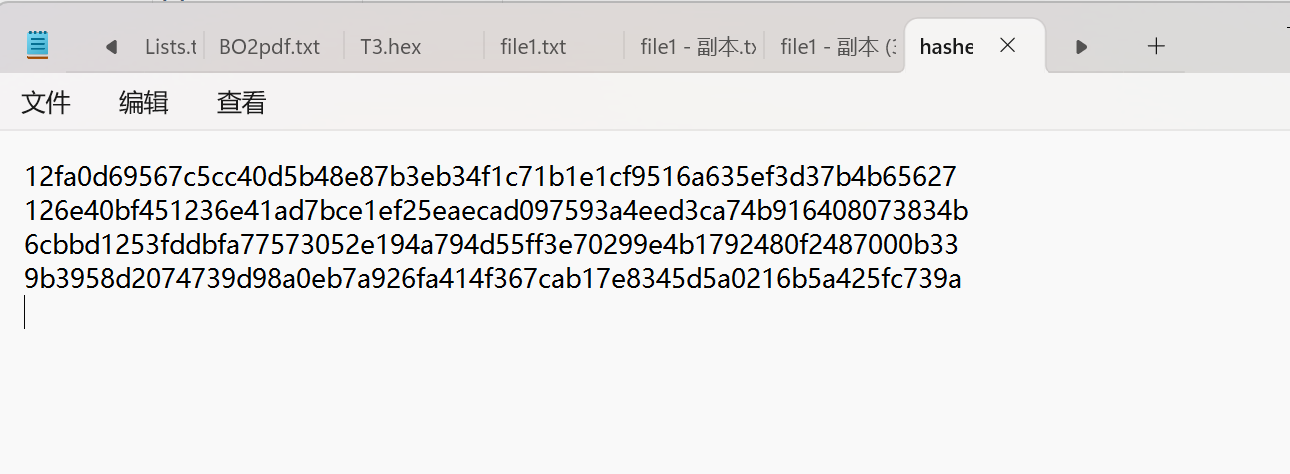
}

运行截图：

EXE2哈希：

EXE1哈希：

信任链哈希：



【小结或讨论】

本次实验成功地实现了动态信任链的构建与验证，模拟了TPM芯片的PCR寄存器功能，增强了对可信计算原理的理解。通过验证系统组件和应用程序的完整性，确保了只有可信的代码才能运行，提高了系统的安全性。

在实际应用中，本质上来说上面本次实验就是可信计算TPM芯片的工作，并且它只在进入选择程序阶段，可以清空PCR4，其他阶段均不可。在后续中，实际上这个选择选项进行执行功能的过程，就是TPM对外提供接口的过程。

TPM作为可信计算平台的核心，实际上是一块安装在主板上，含有密码运算部件和存储部件的系统级芯片。TPM技术最核心的功能在于对CPU处理的数据流进行加密，同时监测系统底层的状态。在此基础上，可以开发出唯一身份识别、系统登录加密、文件夹加密、网络通讯加密等各个环节的安全应用，它能够生成加密的密钥，还有密钥的存储和身份的验证，可以高速进行数据加密和还原，作为保护BIOS和OS不被修改的辅助处理器，通过TSS与TPM的结合来构建跨平台与软硬件系统的可信计算体系结构。即使用户硬盘被盗也不会造成数据泄漏。TPM的序号无法轻易被读出，其读取过程经过加密算法处理，与IC卡一样具有传输加密的安全特性，即TPM芯片就是一颗内嵌于计算机内的智能卡，该芯片的序号代表着该机、该装置、该硬件等信息。TPM上的数字就如同身份证号码，是唯一识别而不重复的一组数字。