学号 专业 信息安全 姓名

实验日期 **2024.10.15** 教师签字 成绩

实验报告

【实验名称】 可信计算第一次实验

【实验目的】

**模拟信任链构建**

假设有4个文件（按照顺序），要求实现以下功能  
**实现信任链构建**：先计算最后一个文件4哈希𝐻4*H*4​，然后把这个𝐻4*H*4​和前一个文件3链接后计算哈希𝐻3*H*3​，𝐻3*H*3​再和前一个文件2链接计算哈希𝐻2*H*2​， 𝐻2*H*2​再和前一个文件1链接计算哈希𝐻1*H*1​。则建立起了一个信任链𝐻1→H2→H3→H4*H*1​→*H*2​→*H*3​→*H*4​  
**实现信任链验证**：利用前面计算出来的信任链𝐻1→H2→H3→H4*H*1​→*H*2​→*H*3​→*H*4​，先读取文件1，拼接上𝐻2*H*2​，计算哈希值，判断和预存的𝐻1*H*1​是否相等，再判断文件2和𝐻3*H*3​的哈希值是否和预存𝐻2*H*2​相等，再判断文件3和𝐻4*H*4​的哈希值是否和预存的𝐻3*H*3​相等，最后再判断文件4的哈希值和预存的𝐻4*H*4​是否相等  
考虑最后一个文件（文件4），可以尝试替换成一个应用程序，即：随意语言编写一个2个纯输出程序，一个输出strA，一个输出strB。将其中一个作为正确的文件，即计算其𝐻4*H*4​。然后在验证信任链最后增加一个if else，如果验证正确，运行该程序，如果错误，中止程序。（每个语言都可以通过相关函数，直接运行其他应用程序）

【实验原理】

通过将各个文件的哈希值（Hash）按特定方式关联起来，形成一个链式结构，使得任何一个文件的篡改都会影响整个链条的完整性。

**构建步骤：**

1. **计算 H4（文件4的哈希值）：**
   * 从最后一个文件（文件4）开始，计算其哈希值，记为 **H4**。
   * 公式表示：H4 = Hash(文件4的内容)
2. **计算 H3（文件3的哈希值）：**
   * 将 **H4** 与文件3的内容连接（串联），然后计算其哈希值，得到 **H3**。
   * 公式表示：H3 = Hash(H4 || 文件3的内容)
   * 其中，|| 表示字符串或二进制数据的连接。
3. **计算 H2（文件2的哈希值）：**
   * 将 **H3** 与文件2的内容连接，计算哈希值，得到 **H2**。
   * 公式表示：H2 = Hash(H3 || 文件2的内容)
4. **计算 H1（文件1的哈希值）：**
   * 将 **H2** 与文件1的内容连接，计算哈希值，得到 **H1**。
   * 公式表示：H1 = Hash(H2 || 文件1的内容)

**验证步骤：**

1. **读取预存的哈希值：**
   * 从安全存储的位置读取预先计算并保存的哈希值：H1、H2、H3 和 H4。
   * 这些哈希值是在信任链构建阶段计算并安全保存的。
2. **验证 H1：**
   * **计算新的 H1'：**
     + 读取文件1的内容，记为 File1。
     + 将预存的 H2 与 File1 的内容连接（串联），得到 Data1 = H2 || File1，其中 || 表示连接操作。
     + 使用哈希函数计算 H1' = Hash(Data1)。
   * **比较 H1' 和预存的 H1：**
     + 如果 H1' == H1，则文件1和预存的 H2 均未被篡改。
     + 如果不相等，则验证失败，信任链被破坏，验证过程终止。
3. **验证 H2：**
   * **计算新的 H2'：**
     + 读取文件2的内容，记为 File2。
     + 将预存的 H3 与 File2 的内容连接，得到 Data2 = H3 || File2。
     + 计算 H2' = Hash(Data2)。
   * **比较 H2' 和预存的 H2：**
     + 如果 H2' == H2，则文件2和预存的 H3 均未被篡改。
     + 如果不相等，则验证失败，信任链被破坏，验证过程终止。
4. **验证 H3：**
   * **计算新的 H3'：**
     + 读取文件3的内容，记为 File3。
     + 将预存的 H4 与 File3 的内容连接，得到 Data3 = H4 || File3。
     + 计算 H3' = Hash(Data3)。
   * **比较 H3' 和预存的 H3：**
     + 如果 H3' == H3，则文件3和预存的 H4 均未被篡改。
     + 如果不相等，则验证失败，信任链被破坏，验证过程终止。
5. **验证 H4：**
   * **计算新的 H4'：**
     + 读取文件4的内容，记为 File4。
     + 计算 H4' = Hash(File4)。
   * **比较 H4' 和预存的 H4：**
     + 如果 H4' == H4，则文件4未被篡改。
     + 如果不相等，则验证失败，信任链被破坏，验证过程终止。

【实验内容】

环境：Win11

语言：C++11

密码学库：cryptopp

1. **构建信任链**

// 信任链构建

#include <iostream>

#include <fstream>

#include <string>

#include <sha.h>

#include <hex.h>

using namespace CryptoPP;

using namespace std;

// 计算文件的哈希值

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;

}

int main() {

// 计算H4

string H4 = computeFileHash("EXE1.exe");

// 计算H3

string H3 = computeHash(H4, "file3.txt");

// 计算H2

string H2 = computeHash(H3, "file2.txt");

// 计算H1

string H1 = computeHash(H2, "file1.txt");

// 将哈希值保存到文件

ofstream hashFile("hashes.txt");

if (!hashFile) {

cerr << "无法创建哈希文件" << endl;

exit(1);

}

hashFile << H1 << endl;

hashFile << H2 << endl;

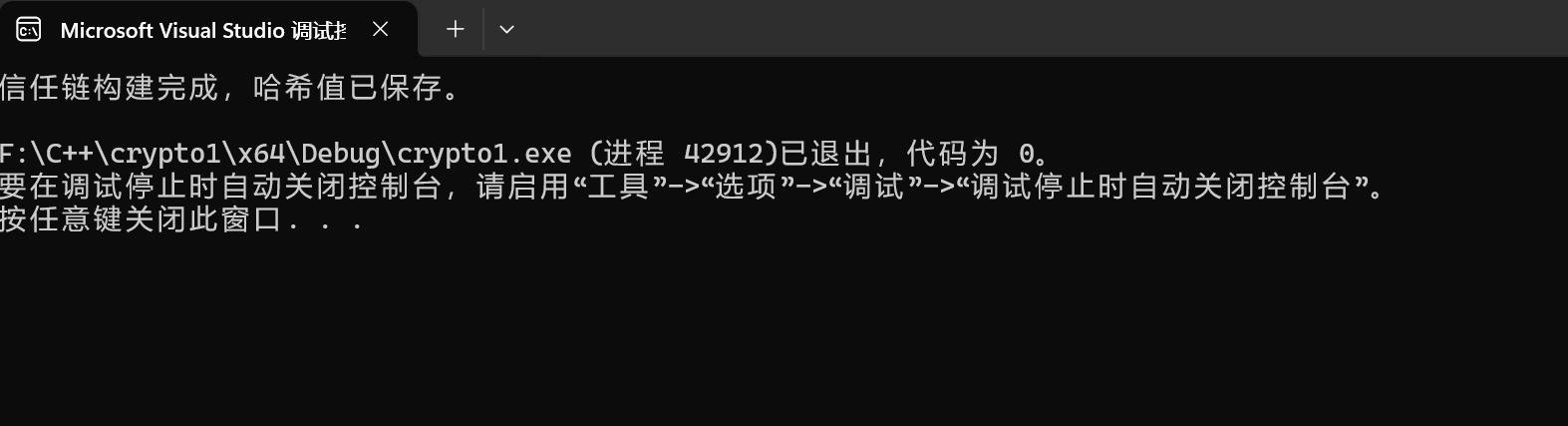
hashFile << H3 << endl;

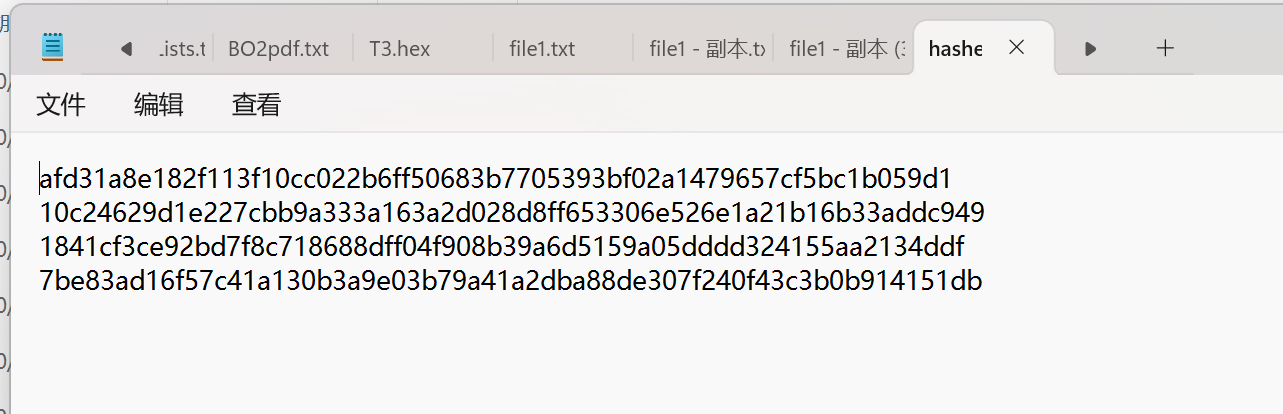
hashFile << H4 << endl;

cout << "信任链构建完成，哈希值已保存。" << endl;

return 0;

}

运行截图：

信任链哈希：

1. **验证信任链**

// 信任链验证

#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;

// 计算文件的哈希值

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;

}

int main() {

// 读取预存的哈希值

ifstream hashFile("hashes.txt");

if (!hashFile) {

cerr << "无法打开哈希文件" << endl;

exit(1);

}

string storedH1, storedH2, storedH3, storedH4;

getline(hashFile, storedH1);

getline(hashFile, storedH2);

getline(hashFile, storedH3);

getline(hashFile, storedH4);

// 验证H1

string computedH2 = computeHash(storedH2, "file1.txt");

if (computedH2 != storedH1) {

cerr << "H1验证失败！" << endl;

exit(1);

}

// 验证H2

string computedH3 = computeHash(storedH3, "file2.txt");

if (computedH3 != storedH2) {

cerr << "H2验证失败！" << endl;

exit(1);

}

// 验证H3

string computedH4 = computeHash(storedH4, "file3.txt");

if (computedH4 != storedH3) {

cerr << "H3验证失败！" << endl;

exit(1);

}

// 验证H4

string computedH4Final = computeFileHash("EXE1.exe");

if (computedH4Final != storedH4) {

cerr << "E1验证失败！" << endl;

//system("EXE2.exe");

exit(1);

}

cout << "信任链验证成功，正在运行应用程序..." << endl;

// 运行应用程序（file4.txt作为可执行文件）

#ifdef \_WIN32

system("EXE1.exe");

#else

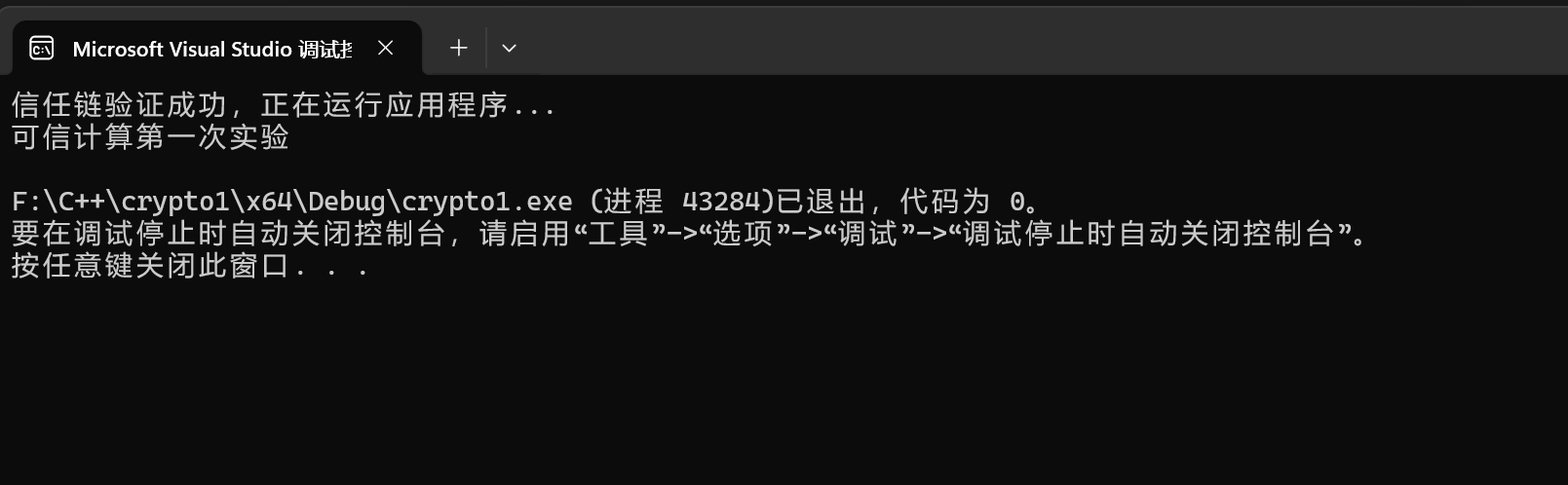
system("./file4");

#endif

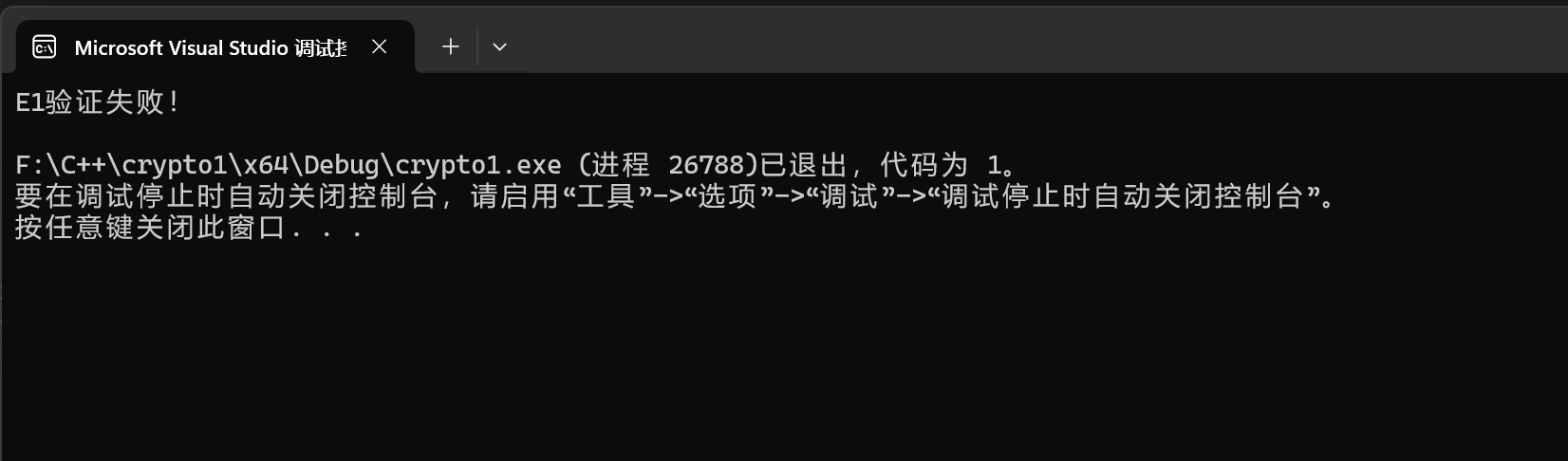
return 0;

}

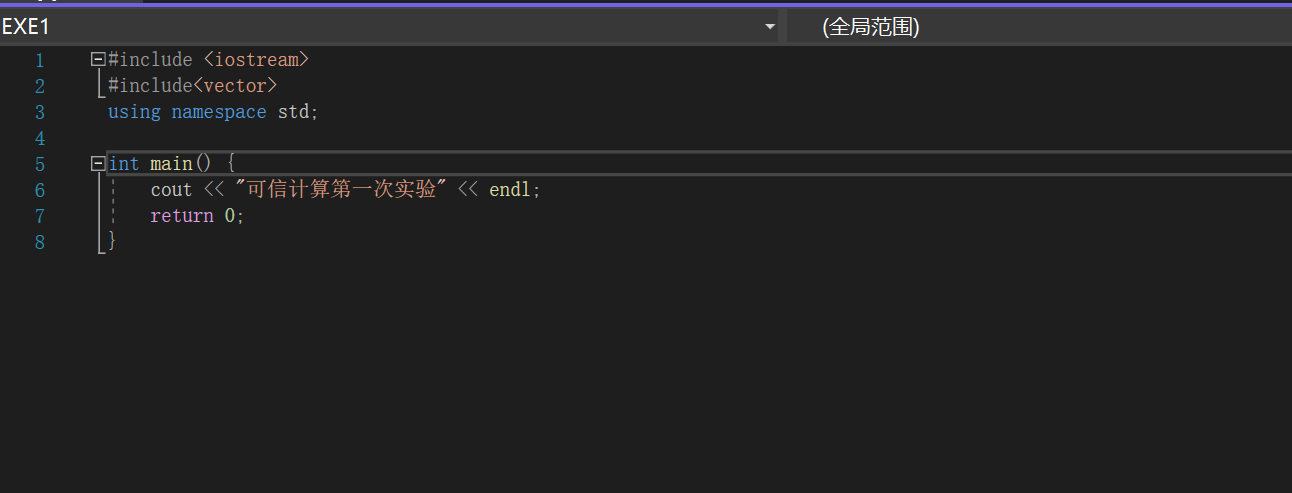
运行截图（成功）：



运行截图（失败）：

****

EXE1.exe源文件：

****

【小结或讨论】

本次实验成功实现了可信计算中信任链的构建和验证。通过计算各个文件的哈希值并依次连接形成信任链，保证了文件内容的完整性和不可篡改性。实验中，我们从最后一个文件开始计算哈希值，逐步链接并计算之前文件的哈希值，最终形成了一个完整的信任链。在验证过程中，成功读取预存的哈希值，并通过逐步比较计算出的哈希值与预存值相等，确保了链中各个文件的安全性。

实验表明，信任链的构建和验证在确保数据完整性方面具有显著效果，一旦某个文件被篡改，验证过程会立即失败，从而阻止后续操作的执行。这种机制可以有效地防止恶意篡改，尤其适用于需要保护数据完整性的场景，如安全存储和传输。

实验中使用了C++11和cryptopp库来计算哈希值，运行环境为Windows 11。在代码实现中，通过SHA256算法计算文件的哈希值，将其作为链条的一部分进行验证。实验还包括了在验证通过后运行指定应用程序的步骤，这进一步展示了信任链的实际应用场景，例如用于安全地执行应用程序。

**信任链的安全意义：**

* **完整性保证：** 任何一个文件的修改都会导致其哈希值改变，进而影响整个信任链的哈希值。这使得篡改难以被隐藏。
* **防篡改：** 攻击者无法仅修改某一个文件而不被检测到，因为这需要重新计算并替换所有相关的哈希值，而这些哈希值通常是安全存储的。
* **验证方便：** 在验证过程中，只需按照链式顺序验证哈希值是否匹配，即可确定所有文件的完整性。

**信任链的核心原理：**

* **链式依赖：** 每个哈希值的计算都依赖于后续的哈希值和当前文件的内容，形成了前后关联的链条。
* **防篡改性：** 任何对文件内容的修改都会破坏哈希值的匹配，从而被检测到。
* **安全性提升：** 增加了攻击者的篡改难度，无法在不被发现的情况下单独修改某个文件。

**类比与拓展：**

* **区块链技术：** 信任链的原理与区块链类似，区块链中每个区块包含前一个区块的哈希值，确保链条的完整性和数据的不可篡改性。
* **实际应用：** 信任链可用于软件更新、安全传输、多级文件验证等场景，提高系统的整体安全性。

**总结：**

通过构建和验证信任链，我们可以：

* **确保文件的完整性和真实性。**
* **防止未经授权的修改和篡改。**
* **在验证成功后，安全地执行受信任的应用程序。**

这种方法增强了系统的安全性，确保了关键操作只有在所有相关文件都可信的情况下才会执行。