密码学
大作业报告

宣恩允

1911501

2022 年 6 月 14 日

目录

1 题目简述

在保密通信过程中，对消息完整性的检验和对消息来源的认证，是一个非常重要的问题，本次大作业的内容为：设计一个协议，利用 rsa 公钥加密算法和 MD5 哈希算法，实现对消息的完整性检验和发送者身份验证的功能，并编程实现这个协议。程序的要求是假设通讯双方为 A 合 B，并假设发方拥有自己的 RSA 公钥 PKA 和私钥 SKA ，同时收方 B 已经通过某种方式知道了发方的公钥 PKA。协议要求对发方 A 发来的消息，收方 B 通过检验，能够确定：

1. B 收到的消息是完整的，即消息在传送过程中没有遭到非法修改；
2. B 收到的消息来源是真实的，即该消息的确是由 A 发来的，而不是由其他人伪造的。

2 实现思路

实验中需要完成的两个功能是：确保接受的消息的完成性和确保消息来源的可靠性。对于这两个功能，我分别使用了 MD5 和 RSA 加密算法来实现，具体思路如下：

- **确保消息完整性**

  这一部分利用 MD5 加密算法，利用其无碰撞性，对需要发送给 B 的消息计算其 MD5 的哈希值，在 B 接受到该条消息时，B 再进行一次 MD5 哈希值的计算，若该次计算的结果与原先的哈希值一致，则可以说明消息在传输的过程中没有被恶意修改。此假设是基于 MD5 的无碰撞性，即两个不同的消息值不会产生相同的 MD5 哈希值。尽管 MD5 算法已经被证明可以在较短时间内找到冲突，但是在本次实验中不考虑可能会冲突的情况。

- **确保消息可靠性**

  这部分利用了 RSA 加密算法，对将要发送的信息在末尾加上发送方的签名，例如，在消息后加上 “i’m Alice.”，然后将该加上了签名的信息使用私钥进行加密，再发送给接收方 B。B 收到消息后，由于题目中假设 B 已经通过某种途径获取了发送方的公钥，B 使用该公钥解密接收到的消息，若可以看到消息末尾的附加信息是 “i’m Alice.” 则可以确定该信息是由 Alice 发出的，即信息是可靠的。

在此前的实验中，已经分别完成了 RSA 加密算法和 MD5 加密算法，因此本次实验中，把这两次实验的代码直接当作头文件引入。

实验包含的文件有：

```
大作业
        chat.exe        // 可执行程序
        Makefile        // 编译文件
        md5.cpp         // MD5加密算法
        md5.h
        rsa.cpp         // RSA加密算法
        rsa.h
        sc.cpp          // 主程序

      BigInt
            binaryHelpFunc.cpp        // 大数类
            binaryHelpFunc.h
```

由于 MD5 和 RSA 文件沿用前两次实验的代码，在此处不作讲解。这次主要编写的程序是 sc.cpp。

2.1 cs.cpp 代码结构

作为本次实验主要编写的部分，其代码结构如下：

```
// 开头是一些头文件链接、宏定义以及用到的全局变量

struct Message...          // 消息结构体

void initializaion()...           // socket初始化

void serverRecv(void *param)...           // 服务端接受线程

void serverSend(void *param)...           // 服务器发送线程

void clientRecv(void *param)...           // 客户端接收线程

void clientSend(void *param)...           // 客户端发送线程

void server()...          // 服务端主程序

void client()...          // 客户端主程序

int main()...         // 主函数
```

2.2 消息结构的定义

实验中发送时不仅仅只需要发送加密后的信息，还需要附带发送信息的 MD5 摘要和发送方加密后的签名。因此定义了如下的消息结构，在发送时整体发送，整体接受。

```
struct Message
{
    char msg[N_LENGTH + 1];
```

```
    char md5text[33];
    char signature[N_LENGTH + 1];
};
```

MD5 算法会生成 128 位的消息摘要，因此存储 MD5 的数据容量设置为 33（多一位结束符号，不然会出错，这个 bug 也是调试了好几天才发现，结果仅仅是把 32 改成 33 就解决了，以后一定要注意）。RSA 加密算法最多产生 1024 位的加密结果，因此将宏 “N_LENGTH” 置为 1024。

2.3 server 主程序

initializaion 函数用于初始化 socket 的相关参数，该部分比较简单，在此不予解释。

下面分析 server 向 client 发送消息的实现过程，反过来 client 向 server 发送消息的相同的实现原理。

服务端主程序 server 中，先生成了客户端使用的 RSA 密钥，然后与设置 socket 为监听状态等待连接。在开始聊天前，需要先将服务器的 RSA 公钥密钥发送给已经连接的客户端：

```
memset(server_Pk, 0, KEY_LENGTH);
strcpy(server_Pk, server_rsa.getPk().getbits().c_str());
send(remoteSocket, server_Pk, KEY_LENGTH, 0);
```

在客户端接受：

```
memset(server_Pk, 0, KEY_LENGTH);
recv(clientSocket, server_Pk, KEY_LENGTH, 0);
BigInt a(server_Pk);
server_rsa.setPk(server_Pk);
```

发送完毕后需要输入服务器的签名，用于消息传输过程中的身份认证，此后，可以开始进入消息传输。

使用了 C++ 多线程来实现此部分，这样可以实现全双工通信，否则仅仅是半双工通信。

```
_beginthread(serverSend, 0, &remoteSocket);
_beginthread(serverRecv, 0, &remoteSocket);
```

2.4 发送和接受的线程函数

- **发送函数**

  对输入的信息以及签名使用服务器的 RSA 私钥加密，计算加密后的消息的 MD5 信息摘要，包装成结构体后，一起发送给客户端。

```
        Message sendmsg;
        memset(&sendmsg, 0, sizeof(Message));

        string temp;

        getline(cin, temp);
        BigInt a = server_rsa.encry(temp);

        // 添加RSA加密后的密文
        temp = a.getbits();
        strcpy(sendmsg.msg, temp.c_str());

        // 添加签名
        BigInt b = server_rsa.encry(signature);
        temp = b.getbits();
        strcpy(sendmsg.signature, temp.c_str());

        // 添加MD5摘要
```

```
19          md5.Update(sendmsg.msg);
20          strcpy(sendmsg.md5text, md5.Tostring().c_str());
21
22          int len = sizeof(Message);
23          send(remoteSocket, (char *)&sendmsg, len, 0);
```

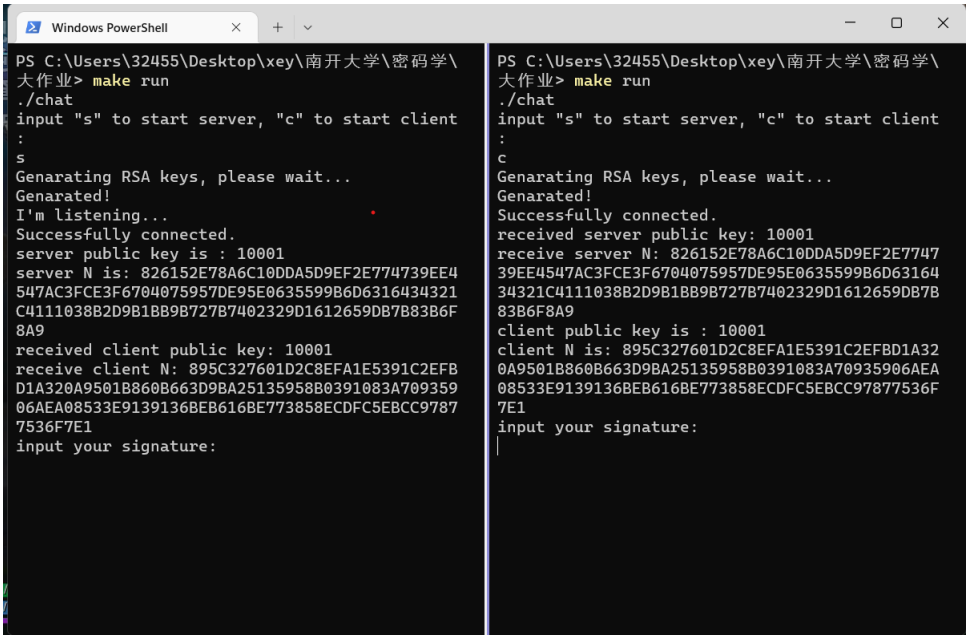
- **接收函数**

  对接受的信息以及签名使用客户端的 RSA 公钥解密，计算解密前消息的 MD5 信息摘要，对比其是否与接受的 MD5 一致，若一致则正确接受，若不一致，则说明传输过程中，消息被篡改。签名解密后，检验是否是服务器的签名，若不是，则说明该信息是伪造的，并不出自服务器。

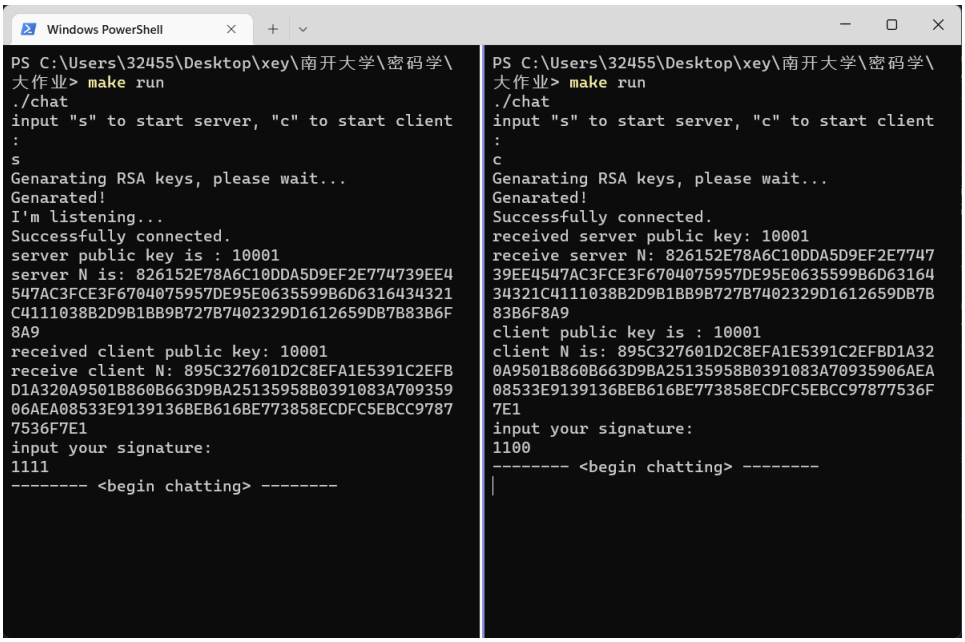
```
1           Message recvmsg;
2           memset(&recvmsg, 0, sizeof(Message));
3           int len = sizeof(Message);
4           recv(remoteSocket, (char *)&recvmsg, len, 0);
```

3 运行结果

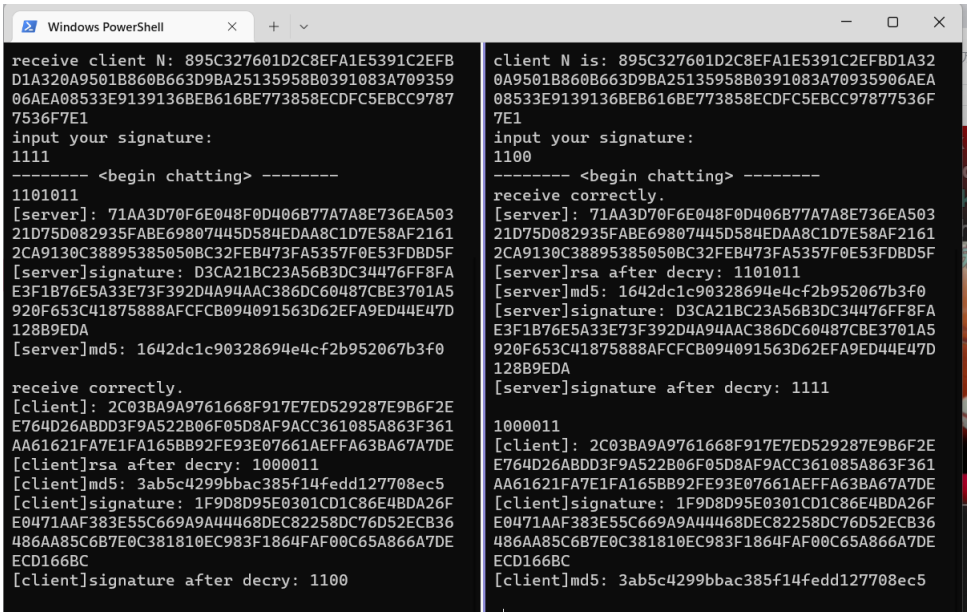
终端中输入 “make run” 运行程序：



可以看到双方先将 RSA 密钥发送，然后输入双方的签名：



这里服务器签名为 “1111”，客户端签名为 “1100”，接下来开始发送消息：



可以看到双方发送的原始信息，以及加密的信息，以及发送后解密的信息。