**《网络空间安全设计与实践Ⅲ》**

课程设计报告

|  |  |
| --- | --- |
| 学 号： |  |
| 姓 名： |  |
| 日 期： |  |

1. 概要设计：说明设计中用到的所有抽象数据类型的定义,主程序的流程以及各程序模 块之间的层次(调用)关系.技术开发思路
2. 详细设计：实现概要设计中定义所有数据类型，绘制流程图及关键技术实现伪码。
3. 调试分析：调试过程中遇到的问题并且是如何解决的以及对设计实现的回顾讨论和 分析;经验和体会及改进设想。
4. 测试结果：列出测试结果,包括输入的数据和相应的输出数据图示.
5. 附录：应附上带详细注释的源程序.

1.概要设计：

第一阶段，双方先建立TCP连接，通过Diffle-Hellman协议确定加解密算法所需要的密钥，通过aes-256-gcm进行加解密，大数部分的运算通过tommath库实现。

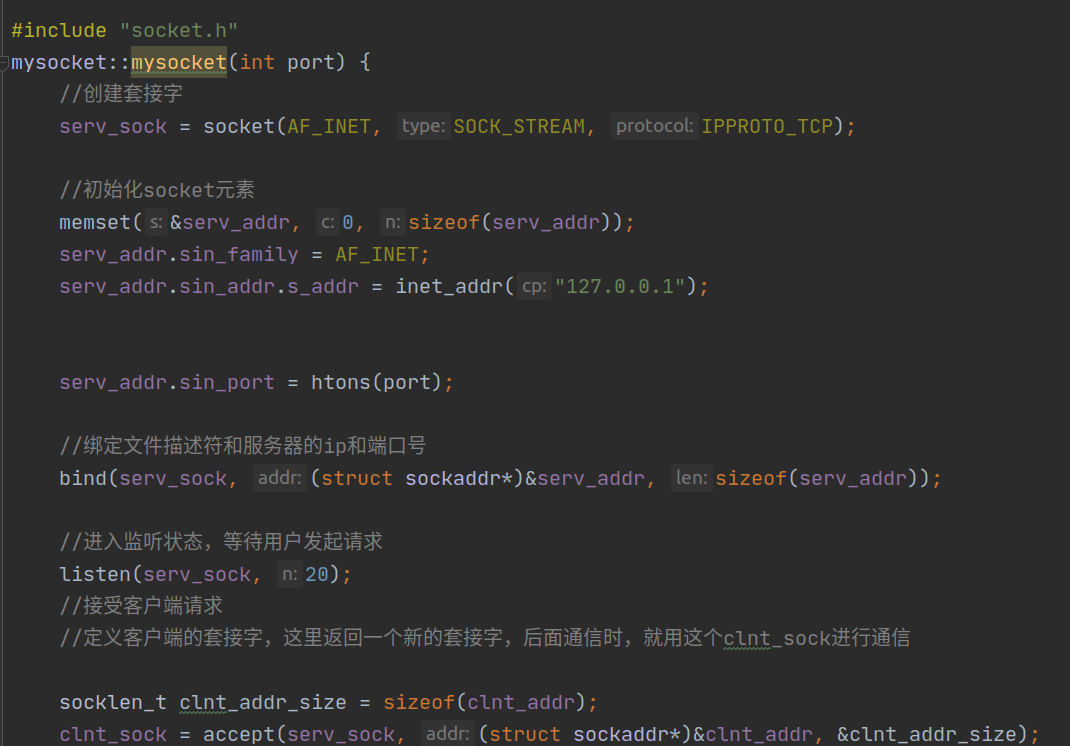
第二阶段，进行中间人攻击。通过ARP欺骗，中间人从中截获通信，与客户端和服务端分别商量密钥，使客户端误以为自己与服务端通信，服务端以为自己在跟客户端通信。

第三阶段，使用预共享密钥防止中间人攻击。通信双方提前利用安全信道协商预共享密钥，改进Diffle-Hellman协议计算方式，中间人如果仍然按照之前算法进行计算，会产生错误密钥，是通信内容出现错误，通信双方发现自己受到了中间人攻击 。

我主要完成了第二阶段：

所有抽象数据类型：

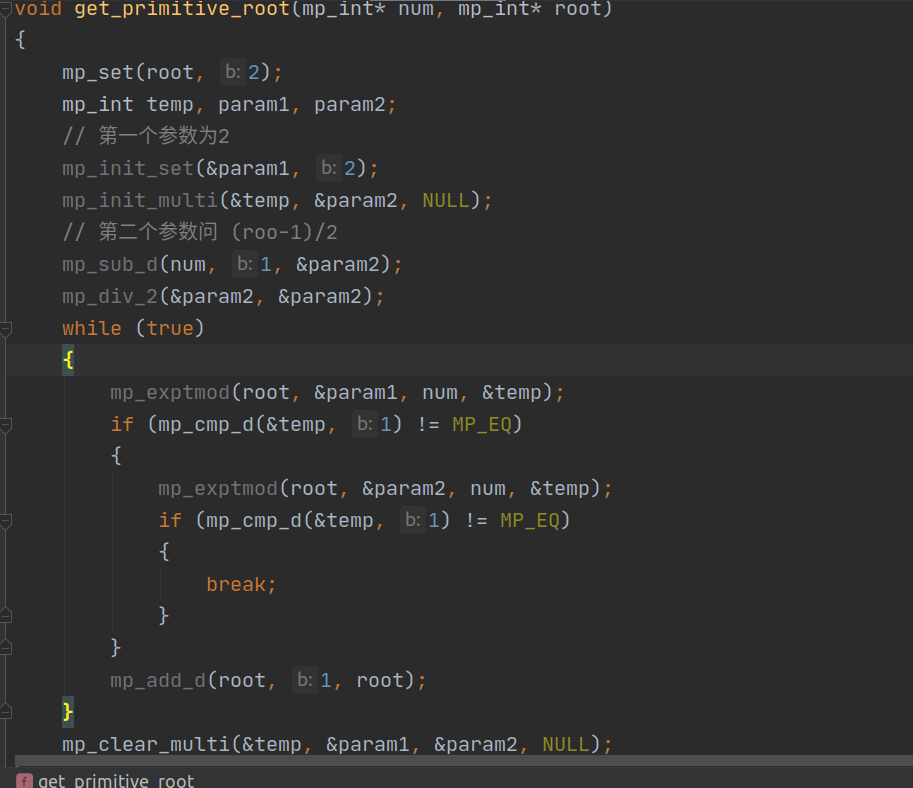
1. Ｓｏｃｋｅｔ通信模型：　ｓｏｃｋ结构体



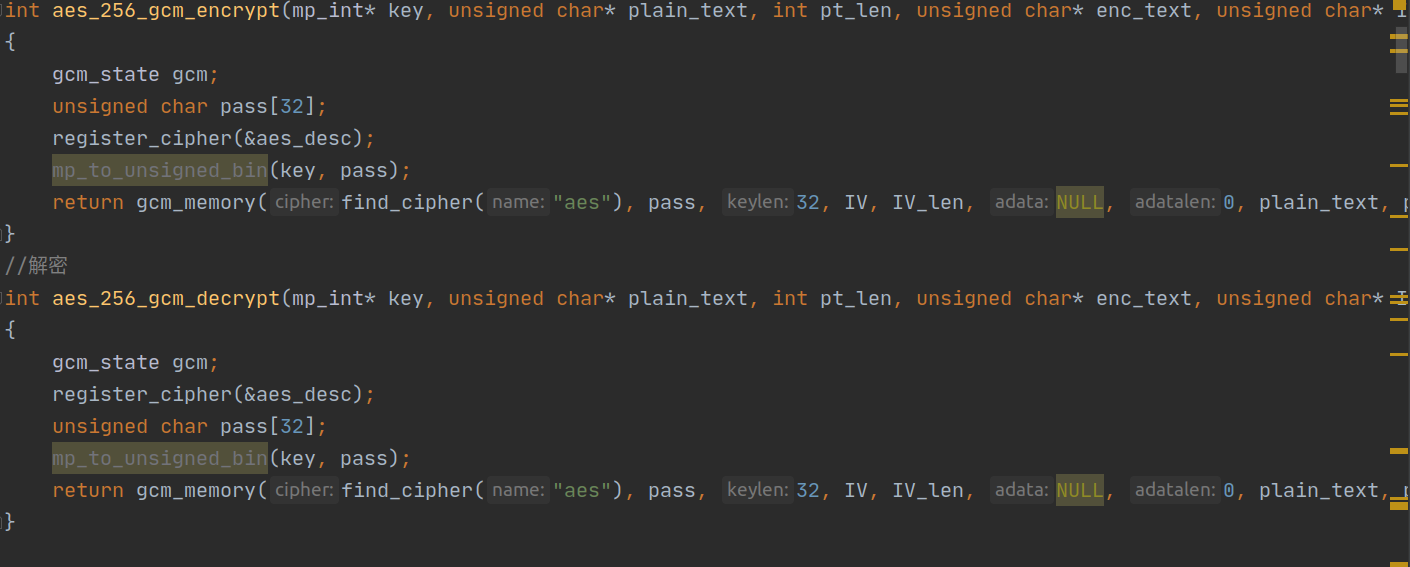
２．发送接受消息函数　ｓｅｎｄ（）　ｒｅｃｖ（）

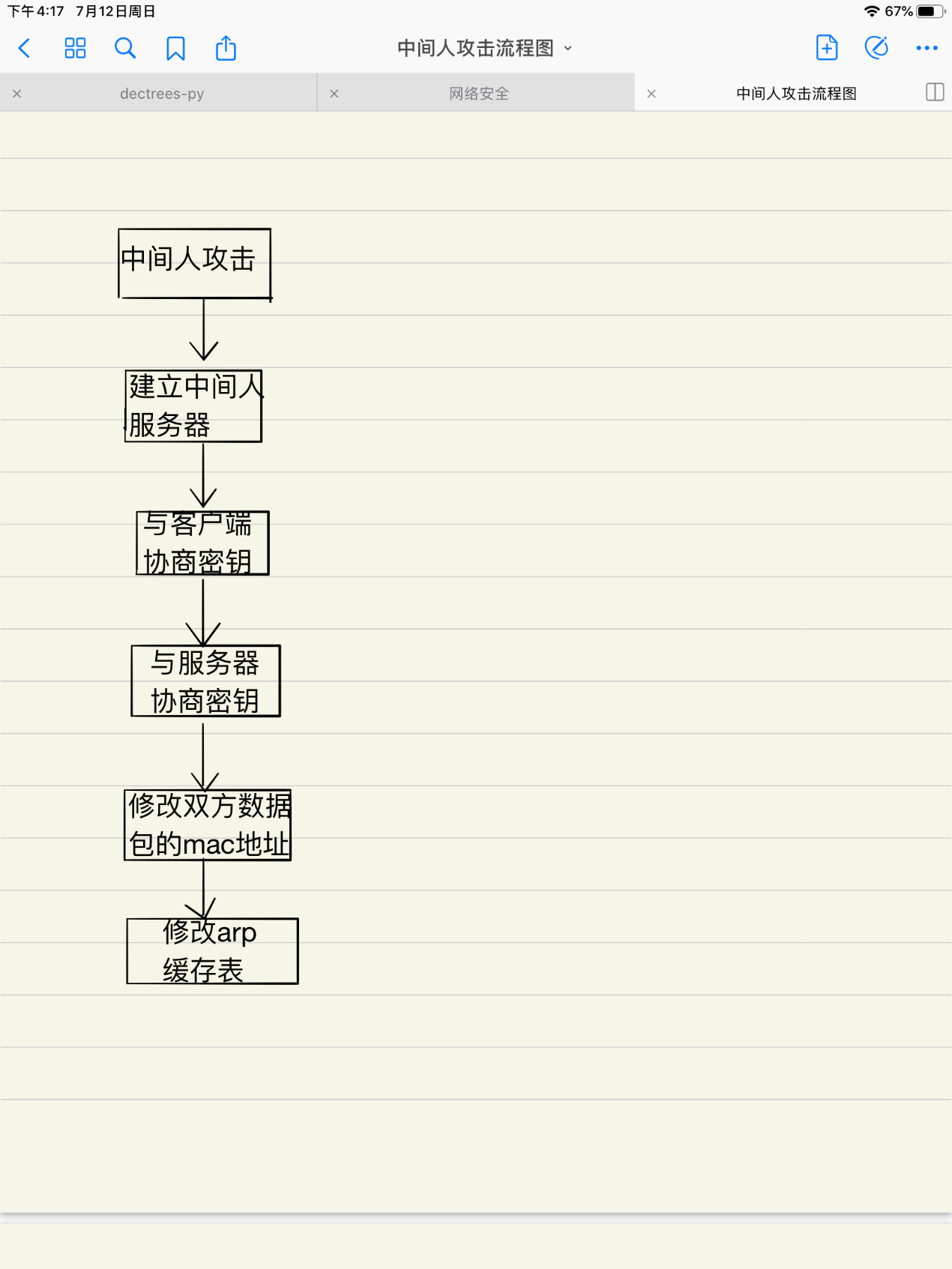


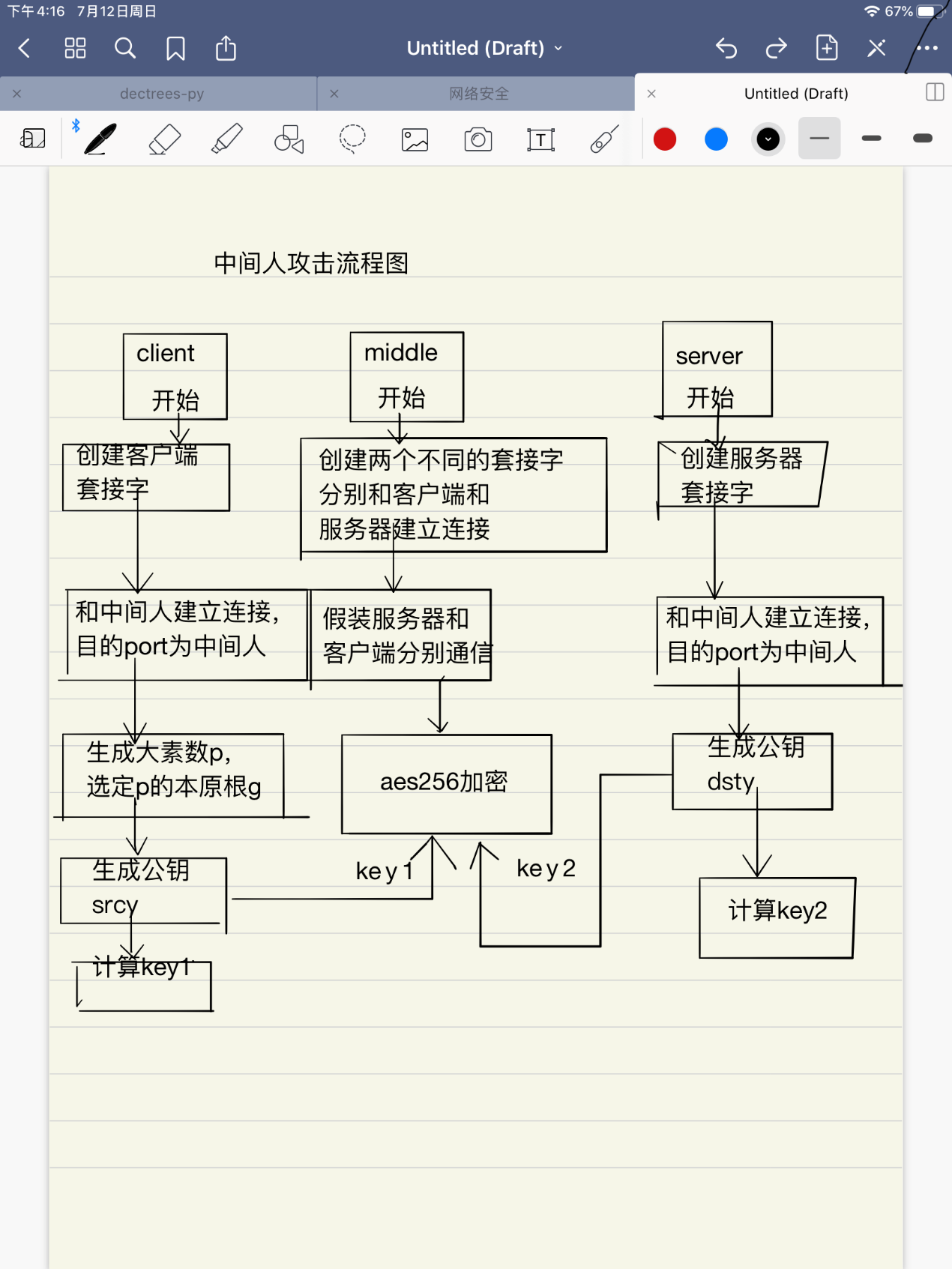
３．大数加密ｍｐ类型



４．Ａｅｓ２５６加密解密函数



主函数流程：、



２．技术思路：

针对第二阶段的中间人攻击，首先我了解了下原理。主要是中间人对客户端和服务器进行ＡＲＰ欺骗，这样就需要中间人修改ＡＲＰ缓存表中的目的ｍａｃ地址。中间人的目的是截获并且修改双方的通信数据，而第一步完成的正常通信模型利用了ａｅｓ２５６加密解密技术。这就需要密码学中所讲到的ｄｆ密钥交换协议。　利用原根的大数求模运算，通信双方可以协商好一套密钥，利用这个密钥加密解密进行通信　。中间人想要截获并且修改数据，就必须获得这个密钥　。　正确的做法是，分别假装服务器和客户端与对方进行交流。协商密钥，这样，在客户端和服务器不知请的情况下，就可以完成两个角色　。　这里协商的两个ｋｅｙ验证后是一致的　。数学上的证明，我记得是ｐ，ｑ，ｇ的求模运算后，利用费马小定理和欧拉公式，指数求模运算后ｋｅｙ恰好完全一样　。

３．调试分析：调试过程中遇到的问题并且是如何解决的以及对设计实现的回顾讨论和 分析;经验和体会及改进设想。

　在具体实现过程中，我发现有两个问题没法解决：

1. 客户端，服务器，中间人，至少需要一个子网下３个不同的ｉｐ地址。
2. 因为需要中间人进行ａｒｐ欺骗，所以需要３个不同的ｍａｃ地址。

解决思路：

我只有一台笔记本，要独立完成，需要在本地网卡上绑定三个不同的ｉｐ，但这样得到的三个ｉｐ地址ｍａｃ地址是相同的，所以不可行。　接着我想到了，开三台虚拟机，是这样可以得到三个不同的ｉｐ和ｍａｃ，但实际运行过程中，笔记本内存只有８ｇ，开两台虚拟机就已经切换困难，开３台经常死机重启，虚拟机繁忙，实在影响进度　。　没办法，我只能还在想其他办法　。　最后，我决定使用不同的套接字，使用三个不同的端口先模拟这个过程　。客户端和中间人使用１２３４端口，中间人和服务器使用１２３４５端口　。这样我在ｃｐｐ文件定义了不同的套接字，将端口作为一个函数参数传入通信过程　。

改进设想：

1. 可以使用多线程，中间人可以同时和客户端和服务器进行通信，使用线程池，在不同的端口同时开启多个通信，这样可以避免单一的客户端发送消息给服务器，被中间人截获，篡改后发送给服务器，服务器再发送消息给客户端，中间人再次截获并篡改，发给客户端。多线程就可以使得中间人同时进行两边的通信，更真实。

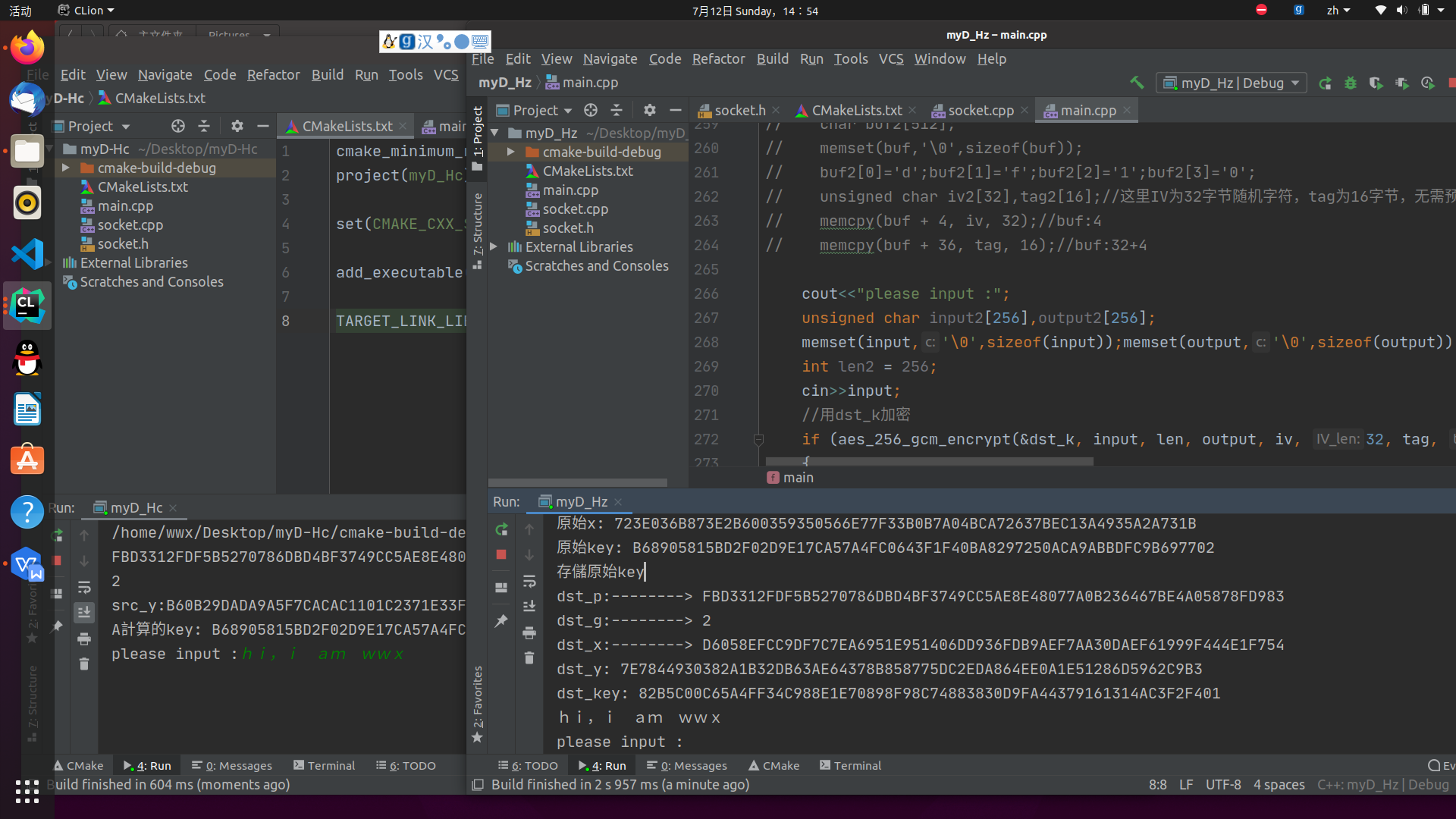
２．此次课设只是简单模拟了利用arp欺骗进行中间人攻击的过程，环境局限于客户端服务端和中间人都在同一局域网中，并未对其他情况例如客户端和服务端不在同一局域网中进行模拟，以下简单讨论其他的中间人攻击方式，即外网环境下的ｑｒｐ欺骗，技术难度更高。

会话劫持：TCP会话劫持的攻击方式可以对基于TCP的任何应用发起攻击，如HTTP、FTP、Telnet等。对于攻击者来说，所必须要做的就是窥探到正在进行TCP通信的两台[主机](https://baike.baidu.com/item/%E4%B8%BB%E6%9C%BA)之间传送的报文，这样攻击者就可以得知该报文的源IP、源TCP[端口号](https://baike.baidu.com/item/%E7%AB%AF%E5%8F%A3%E5%8F%B7)、目的IP、目的TCP端号，从而可以得知其中一台主机对将要收到的下一个TCP[报文段](https://baike.baidu.com/item/%E6%8A%A5%E6%96%87%E6%AE%B5)中seq和ackseq值的要求。这样，在该合法主机收到另一台合法主机发送的TCP[报文](https://baike.baidu.com/item/%E6%8A%A5%E6%96%87)前，攻击者根据所截获的信息向该主机发出一个带有净荷的TCP报文，如果该主机先收到攻击报文，就可以把合法的TCP会话建立在攻击主机与被攻击主机之间。带有净荷的攻击报文能够使被攻击主机对下一个要收到的TCP报文中的确认序号（ackseq）的值的要求发生变化，从而使另一台合法的主机向被攻击主机发出的报文被被攻击主机拒绝。TCP会话劫持攻击方式的好处在于使攻击者避开了被攻击[主机](https://baike.baidu.com/item/%E4%B8%BB%E6%9C%BA)对访问者的身份验证和安全认证，从而使攻击者直接进入对被攻击主机的的访问状态。

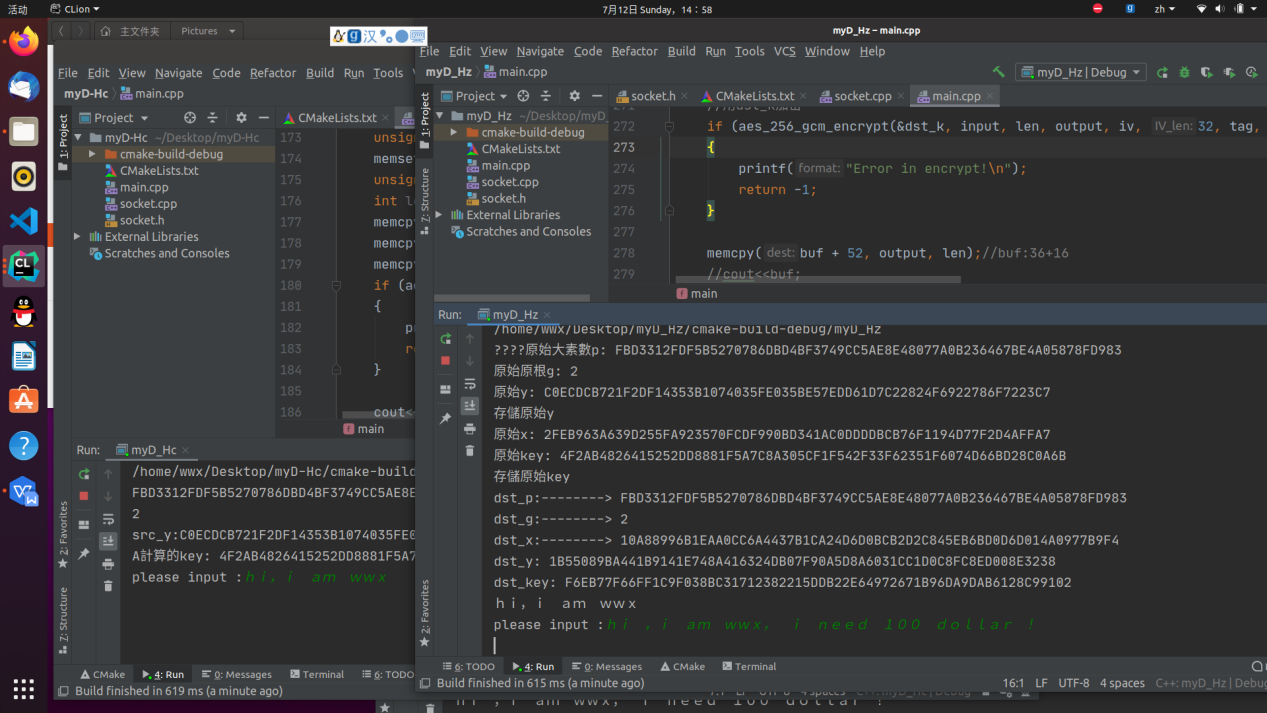
DNS欺骗：攻击者通过入侵DNS[服务器](https://baike.baidu.com/item/%E6%9C%8D%E5%8A%A1%E5%99%A8)、控制[路由器](https://baike.baidu.com/item/%E8%B7%AF%E7%94%B1%E5%99%A8)等方法把受害者要访问的目标机器域名对应的IP解析为攻击者所控制的机器，这样受害者原本要发送给目标机器的数据就发到了攻击者的机器上，这时攻击者就可以监听甚至修改数据，从而收集到大量的信息。如果攻击者只是想监听双方会话的数据，他会转发所有的数据到真正的目标机器上，让目标机器进行处理，再把处理结果发回到原来的受害者机器；如果攻击者要进行彻底的破坏，他会伪装目标机器返回数据，这样受害者接收处理的就不再是原来期望的数据，而是攻击者所期望的了。

1. 测试结果

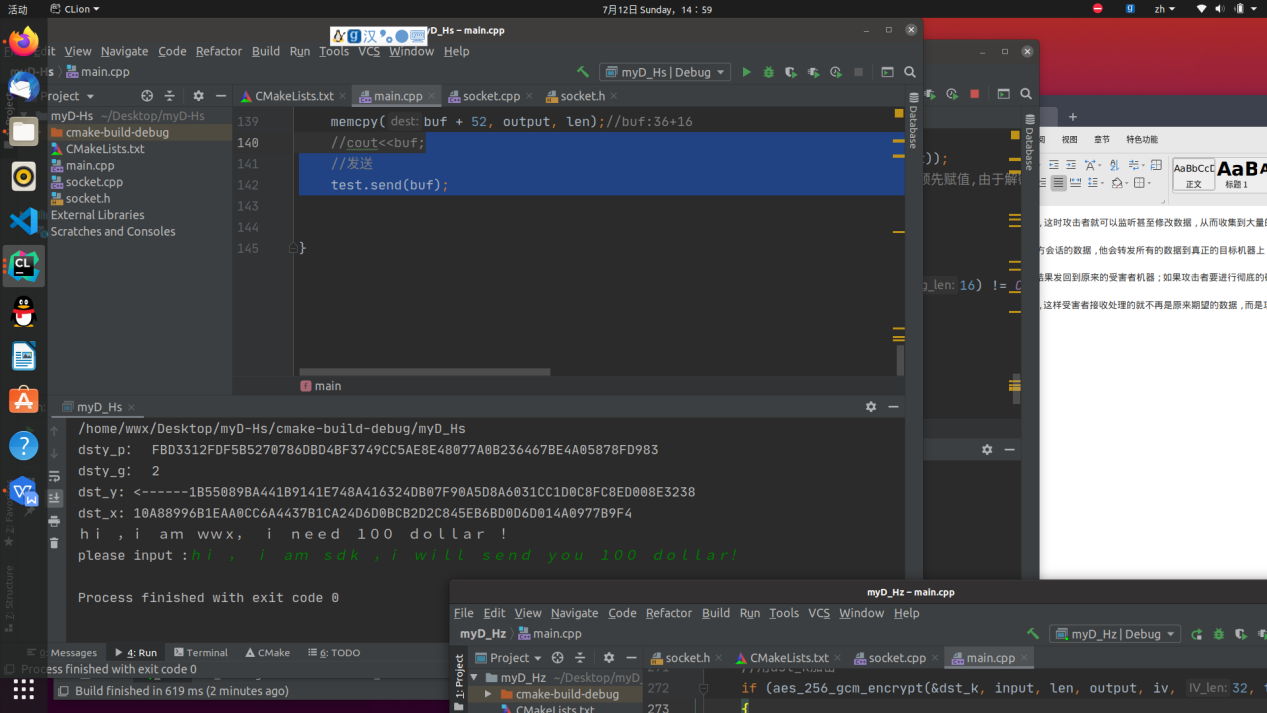
客户端向服务器发送消息，中间人进行截获：



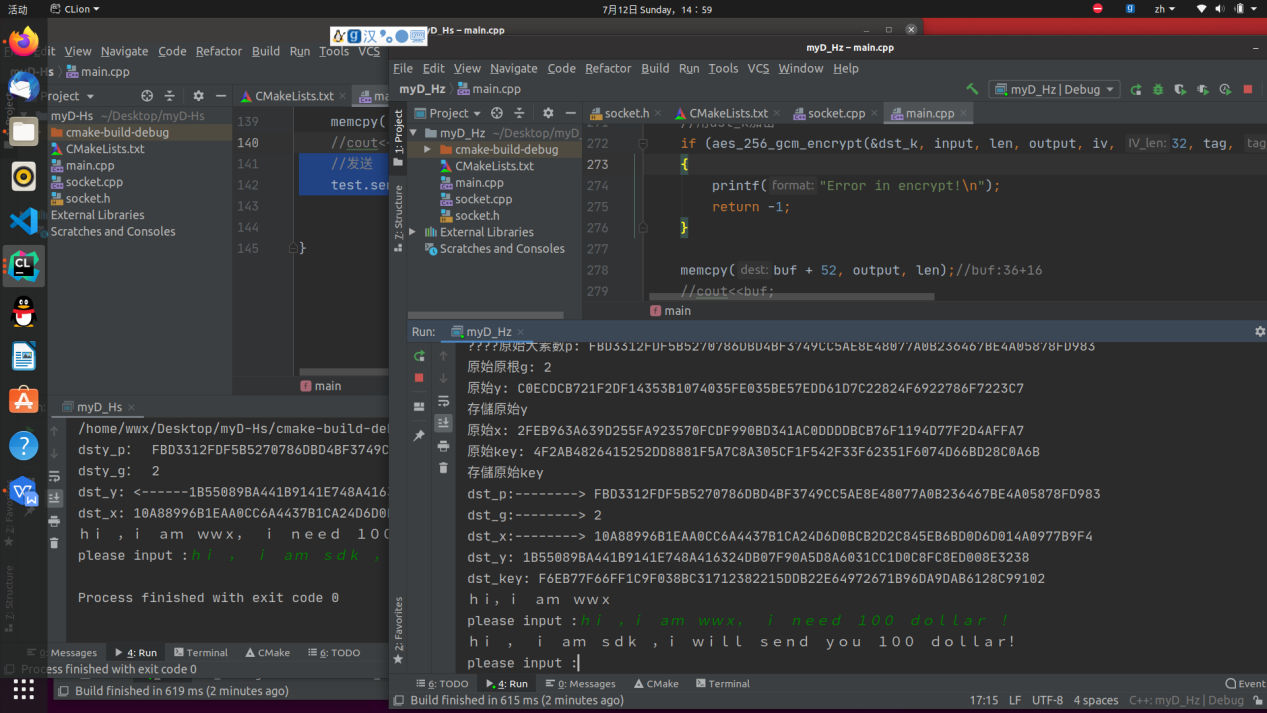
中间人截获消息，篡改消息发送给服务器



服务器接受到中间人发来的假消息，以为是客户端发来的，回复客户端。



中间人截获服务器发给客户端的消息



再次发送假消息给客户端。客户端误认为是服务器发来的消息。

Cmakelist.txt

cmake\_minimum\_required(VERSION 3.13)

project(myD\_Hz)

set(CMAKE\_CXX\_STANDARD 11)

add\_executable(myD\_Hz main.cpp socket.cpp socket.h)

TARGET\_LINK\_LIBRARIES(myD\_Hz pcap tommath tomcrypt)

Socket.h

#ifndef MYD\_HS\_SOCKET\_H

#define MYD\_HS\_SOCKET\_H

#include<stdio.h>

#include<sys/socket.h>

#include<netinet/in.h>

#include<stdlib.h>

#include<arpa/inet.h>

#include<unistd.h>

#include<string.h>

class mysocket\_client{

public:

mysocket\_client();

mysocket\_client(int port);

~mysocket\_client();

void send(char \* value);

void recv(char \* value);

private:

int len=512;

int sock;

struct sockaddr\_in serv\_addr;

};

class mysocket {

public:

mysocket();

mysocket(int port);

~mysocket();

void send(char \* value);

void recv(char \* value);

private:

int len=512;

int serv\_sock;

int clnt\_sock;

struct sockaddr\_in serv\_addr;

struct sockaddr\_in clnt\_addr;

};

#endif //MYD\_HS\_SOCKET\_H

Socket.cpp

#include "socket.h"

mysocket::mysocket(int port) {

//创建套接字

serv\_sock = socket(AF\_INET, SOCK\_STREAM, IPPROTO\_TCP);

//初始化socket元素

memset(&serv\_addr, 0, sizeof(serv\_addr));

serv\_addr.sin\_family = AF\_INET;

serv\_addr.sin\_addr.s\_addr = inet\_addr("127.0.0.1");

serv\_addr.sin\_port = htons(port);

//绑定文件描述符和服务器的ip和端口号

bind(serv\_sock, (struct sockaddr\*)&serv\_addr, sizeof(serv\_addr));

//进入监听状态，等待用户发起请求

listen(serv\_sock, 20);

//接受客户端请求

//定义客户端的套接字，这里返回一个新的套接字，后面通信时，就用这个clnt\_sock进行通信

socklen\_t clnt\_addr\_size = sizeof(clnt\_addr);

clnt\_sock = accept(serv\_sock, (struct sockaddr\*)&clnt\_addr, &clnt\_addr\_size);

}

mysocket\_client::mysocket\_client(int port) {

sock = socket(AF\_INET, SOCK\_STREAM, 0);

//服务器的ip为本地，端口号1234

memset(&serv\_addr, 0, sizeof(serv\_addr));

serv\_addr.sin\_family = AF\_INET;

serv\_addr.sin\_addr.s\_addr = inet\_addr("127.0.0.1");

serv\_addr.sin\_port = htons(port);

//向服务器发送连接请求

connect(sock, (struct sockaddr\*)&serv\_addr, sizeof(serv\_addr));

}

mysocket::~mysocket() {

close(clnt\_sock);

close(serv\_sock);

}

void mysocket::recv(char \* value) {

read(clnt\_sock, value, len);

}

void mysocket::send(char \* value) {

write(clnt\_sock, value, len);

}

mysocket\_client::~mysocket\_client() {

close(sock);

}

void mysocket\_client::send(char \*value) {

write(sock, value, len);

}

void mysocket\_client::recv(char \*value) {

read(sock, value, len);

}

Main.cpp

#include <iostream>

#include "socket.h"

#include <tommath.h>//sudo apt-get install libtommath-dev

#include <tomcrypt.h>//sudo apt-get install libtomcrypt-dev

#include "socket.h"

using namespace std;

//加密

int aes\_256\_gcm\_encrypt(mp\_int\* key, unsigned char\* plain\_text, int pt\_len, unsigned char\* enc\_text, unsigned char\* IV, int IV\_len, unsigned char\* tag, unsigned long tag\_len)

{

gcm\_state gcm;

unsigned char pass[32];

register\_cipher(&aes\_desc);

mp\_to\_unsigned\_bin(key, pass);

return gcm\_memory(find\_cipher("aes"), pass, 32, IV, IV\_len, NULL, 0, plain\_text, pt\_len, enc\_text, tag, &tag\_len, GCM\_ENCRYPT);

}

//解密

int aes\_256\_gcm\_decrypt(mp\_int\* key, unsigned char\* plain\_text, int pt\_len, unsigned char\* enc\_text, unsigned char\* IV, int IV\_len, unsigned char\* tag, unsigned long tag\_len)

{

gcm\_state gcm;

register\_cipher(&aes\_desc);

unsigned char pass[32];

mp\_to\_unsigned\_bin(key, pass);

return gcm\_memory(find\_cipher("aes"), pass, 32, IV, IV\_len, NULL, 0, plain\_text, pt\_len, enc\_text, tag, &tag\_len, GCM\_DECRYPT);

}

//求原根

void get\_primitive\_root(mp\_int\* num, mp\_int\* root)

{

mp\_set(root, 2);

mp\_int temp, param1, param2;

// 第一个参数为2

mp\_init\_set(&param1, 2);

mp\_init\_multi(&temp, &param2, NULL);

// 第二个参数问 (roo-1)/2

mp\_sub\_d(num, 1, &param2);

mp\_div\_2(&param2, &param2);

while (true)

{

mp\_exptmod(root, &param1, num, &temp);

if (mp\_cmp\_d(&temp, 1) != MP\_EQ)

{

mp\_exptmod(root, &param2, num, &temp);

if (mp\_cmp\_d(&temp, 1) != MP\_EQ)

{

break;

}

}

mp\_add\_d(root, 1, root);

}

mp\_clear\_multi(&temp, &param1, &param2, NULL);

}

//生成大素数

int rng(unsigned char \*dst, int len, void \*dat)

{

int x;

for (x = 0; x < len; x++) dst[x] = rand() & 0xFF;

return len;

}

int preme\_random(mp\_int \*p)

{

int err = mp\_prime\_random\_ex(p, 8, 256, LTM\_PRIME\_2MSB\_ON | LTM\_PRIME\_SAFE, rng, NULL);

if (err != MP\_OKAY) {

return -1;

}

return 0;

}

////----------------------------------------------------------

//int Mtim(char buf[]){

// if (buf[0]=='d'&&buf[1]=='f'&&buf[2]==1){

// if (buf[3]==1){

// //發起密鑰交換

//

// }

// if (buf[3]==2){

// //回應密鑰交換

// }

// if (buf[3]==0){

// //交換密鑰後的正常傳輸

//

// }

// }

// else{//三次握手數據包

//

// }

//}

int main() {

cout<<"????";

mysocket test(1234);

mp\_int p,g,x,b,y,k;

int options;

mp\_init(&p);mp\_init(&g);mp\_init(&x);mp\_init(&b);mp\_init(&y);mp\_init(&k);

//接收p,g

char value[512];memset(value,'\0',sizeof(value));

char src\_y[512];memset(src\_y,'\0',sizeof(src\_y));

char src\_key[512];memset(src\_key,'\0',sizeof(src\_key));

char dst\_key[512];memset(dst\_key,'\0',sizeof(dst\_key));

char value\_d[512];memset(value\_d,'\0',sizeof(value\_d));

//cout<<"Server Modern ronud 1 :input 1;"

// " Client Modern Round 1: input 2;"

// "Client Modern Round 2: input 3 "

// "Server Modern round 2: input 4;"

// <<endl;

//cin>>options;

//－－－－－－－－－－－－協商密鑰部分　ｒｏｕｎｄ　１－－－－－－－－－－－－－－－－－－－－－－－－

//中間人假裝ｓｅｒｖｅｒ，截取ｃｌｉｅｎｔ的src\_x src＿p,src\_g　並計算保存src\_y, src\_key

// if(options==1){

test.recv(value);

mp\_read\_radix(&p,value,0x10);//char[]->mp\_int

cout<<"原始大素數p: "<<value<<endl;

memset(value,'\0',sizeof(value));

test.recv(value);

mp\_read\_radix(&g,value,0x10);//char[]->mp\_int

cout<<"原始原根g: "<<value<<endl;

memset(value,'\0',sizeof(value));

/\*协商密钥\*/

//1.生成b

mp\_rand(&b, p.used);

//2.计算y=g^b mod p

mp\_exptmod(&g,&b,&p,&y);

mp\_toradix(&y,value,0x10);//mp\_int->char[]

cout<<"原始y: "<<value<<endl;

cout<<"存儲原始y"<<endl;

test.send(value);

//char\* src\_y=value;

strcpy(src\_y,value);

//cout<<"src\_y:"<<src\_y<<endl;

memset(value,'\0',sizeof(value));

// //3.发送y

// mp\_toradix(&y,value,0x10);//mp\_int->char[]

// test.send(value);

// memset(value,'\0',sizeof(value));

//4.接收x

test.recv(value);

cout<<"原始x: "<<value<<endl;

mp\_read\_radix(&x,value,0x10);//char[]->mp\_int

memset(value,'\0',sizeof(value));

//5.计算k= x^b mod p

mp\_exptmod(&x,&b,&p,&k);

mp\_toradix(&k,value,0x10);//mp\_int->char[]

cout<<"原始key: "<<value<<endl;

cout<<"存儲原始key"<<endl;

//char \*src\_key=value;

strcpy(src\_key,value);

//cout<<"src\_key"<<src\_key<<endl;

//// mp\_toradix(&k,value,0x10);//mp\_int->char[]

//// cout<<value<<endl;

// }

/\*协商密钥\*/

//1.生成大数a(与p大小相近)

mp\_int a;

mp\_init(&a);

mp\_rand(&a, p.used);

//--------中間人生成dst\_p,dst\_g,dst\_x,並假裝client，發送給ｓｅｒｖｅｒ--------------------

//if(options==2) {

mysocket\_client test2(12345);

//生成大素数p

mp\_int dst\_p;

mp\_init(&dst\_p);

preme\_random(&dst\_p);

//求原根g

mp\_int dst\_g;

mp\_init(&dst\_g);

get\_primitive\_root(&dst\_p, &dst\_g);

mp\_toradix(&dst\_p, value\_d, 0x10);//mp\_int->char[]

test2.send(value\_d);

cout << "dst\_p:--------> " << value\_d << endl;

memset(value\_d, '\0', sizeof(value\_d));

mp\_toradix(&dst\_g, value\_d, 0x10);//mp\_int->char[]

test2.send(value\_d);

cout << "dst\_g:--------> " << value\_d << endl;

memset(value\_d, '\0', sizeof(value\_d));

//2.x=g^a mod p

mp\_int dst\_x;

mp\_init(&dst\_x);

mp\_exptmod(&dst\_g, &a, &dst\_p, &dst\_x);

//3.发送dst\_x

mp\_toradix(&dst\_x, value\_d, 0x10);//mp\_int->char[]

test2.send(value\_d);

cout << "dst\_x:--------> " << value\_d << endl;

memset(value\_d, '\0', sizeof(value\_d));

//－－－－－－－－－－－－－－－－－－回應部分－－－－－－－－－－－－－－－－－－－－－－

//--------------中間人假裝ｃｌｉｅｎｔ，截獲ｓｅｒｖｅｒ發送的dst\_y，並計算dst\_key

//if(options==3){

//接收dst\_y

mp\_int dst\_y;

mp\_init(&dst\_y);

test2.recv(value\_d);

mp\_read\_radix(&dst\_y, value\_d, 0x10);//char[]->mp\_int

cout << "dst\_y: " << value\_d << endl;

memset(value\_d, '\0', sizeof(value\_d));

//5.计算dst\_key= y^a mod p

mp\_int dst\_k;

mp\_init(&dst\_k);

mp\_exptmod(&dst\_y, &a, &p, &dst\_k);

mp\_toradix(&dst\_k, value, 0x10);//mp\_int->char[]

strcpy(dst\_key,value);

cout << "dst\_key: " << value << endl;

// }

//}

//--------------中間人假裝ｓｅｒｖｅｒ，發送給ｃｌｉｅｎｔ　緩存好的src\_y

/\*正常通信\*/

/\*

我自定义的协议头：

开头两个字节为df，第三位为1（版本号），第四位为操作特征符

定义特征符为

1：发起密钥交换

2：回应密钥交换

0：交换后的正常数据传输

定义一个特征头部有助于抓包中进行判定和筛选数据包

\*/

char buf[512];

memset(buf,'\0',sizeof(buf));

test.recv(buf);

unsigned char input[256],output[256];

memset(input,'\0',sizeof(input));memset(output,'\0',sizeof(output));

unsigned char iv[32],tag[16];//这里IV为32字节随机字符，tag为16字节，无需预先赋值,由于解密也需要iv和tag,故这两个参数随着socket一起发过去

int len = 256;

memcpy(iv,buf + 4, 32);

memcpy(tag,buf + 36, 16);

memcpy(output, buf + 52, len);

//用src\_k解密

if (aes\_256\_gcm\_decrypt(&k, input, len, output, iv, 32, tag, 16) != CRYPT\_OK)

{

printf("Error in encrypt!\n");

return -1;

}

cout<<input<<endl;

//============================

/\*正常通信\*/

/\*

我自定义的协议头：

开头两个字节为df，第三位为1（版本号），第四位为操作特征符

定义特征符为

1：发起密钥交换

2：回应密钥交换

0：交换后的正常数据传输

定义一个特征头部有助于抓包中进行判定和筛选数据包

\*/

// char buf2[512];

// memset(buf,'\0',sizeof(buf));

// buf2[0]='d';buf2[1]='f';buf2[2]='1';buf2[3]='0';

// unsigned char iv2[32],tag2[16];//这里IV为32字节随机字符，tag为16字节，无需预先赋值,由于解密也需要iv和tag,故这两个参数随着socket一起发过去

// memcpy(buf + 4, iv, 32);//buf:4

// memcpy(buf + 36, tag, 16);//buf:32+4

cout<<"please input :";

unsigned char input2[256],output2[256];

memset(input,'\0',sizeof(input));memset(output,'\0',sizeof(output));

int len2 = 256;

cin>>input;

//用dst\_k加密

if (aes\_256\_gcm\_encrypt(&dst\_k, input, len, output, iv, 32, tag, 16) != CRYPT\_OK)

{

printf("Error in encrypt!\n");

return -1;

}

memcpy(buf + 52, output, len);//buf:36+16

//cout<<buf;

//发送

test2.send(buf);

//-----------------------------------

/\*正常通信\*/

/\*

我自定义的协议头：

开头两个字节为df，第三位为1（版本号），第四位为操作特征符

定义特征符为

1：发起密钥交换

2：回应密钥交换

0：交换后的正常数据传输

定义一个特征头部有助于抓包中进行判定和筛选数据包

\*/

char buf3[512];

memset(buf,'\0',sizeof(buf));

test2.recv(buf);

unsigned char input3[256],output3[256];

memset(input,'\0',sizeof(input));memset(output,'\0',sizeof(output));

unsigned char iv3[32],tag3[16];//这里IV为32字节随机字符，tag为16字节，无需预先赋值,由于解密也需要iv和tag,故这两个参数随着socket一起发过去

int len3 = 256;

memcpy(iv,buf + 4, 32);

memcpy(tag,buf + 36, 16);

memcpy(output, buf + 52, len);

//用dst\_k解密

if (aes\_256\_gcm\_decrypt(&dst\_k, input, len, output, iv, 32, tag, 16) != CRYPT\_OK)

{

printf("Error in encrypt!\n");

return -1;

}

cout<<input<<endl;

//============================

/\*正常通信\*/

/\*

我自定义的协议头：

开头两个字节为df，第三位为1（版本号），第四位为操作特征符

定义特征符为

1：发起密钥交换

2：回应密钥交换

0：交换后的正常数据传输

定义一个特征头部有助于抓包中进行判定和筛选数据包

// \*/

// char buf4[512];

// memset(buf,'\0',sizeof(buf));

// buf2[0]='d';buf2[1]='f';buf2[2]='1';buf2[3]='0';

// unsigned char iv4[32],tag4[16];//这里IV为32字节随机字符，tag为16字节，无需预先赋值,由于解密也需要iv和tag,故这两个参数随着socket一起发过去

memcpy(buf + 4, iv, 32);//buf:4

memcpy(buf + 36, tag, 16);//buf:32+4

cout<<"please input :";

unsigned char input4[256],output4[256];

memset(input,'\0',sizeof(input));memset(output,'\0',sizeof(output));

int len4 = 256;

cin>>input;

//用src\_k加密

if (aes\_256\_gcm\_encrypt(&k, input, len, output, iv, 32, tag, 16) != CRYPT\_OK)

{

printf("Error in encrypt!\n");

return -1;

}

memcpy(buf + 52, output, len);//buf:36+16

//cout<<buf;

//发送

test.send(buf);

}