



南开大学
Nankai University

南 开 大 学

计 算 机 学 院

网络安全技术作业报告

基于 RSA 算法自动分配密钥的加密聊天程序

朱浩泽 1911530

年级：2019 级

专业：计算机科学与技术

班级：计算机科学与技术 2 班

2022 年 4 月 30 日

目录

一、 实验目的	1
二、 实验内容	1
三、 实验步骤及实验结果	1
(一) 实验环境	1
(二) 代码结构	1
(三) 代码实现	2
1. 封装 DES 加密解密算法的类	2
2. 封装 RSA 加密解密算法的类，并检验其正确性。	2
3. 通讯部分	7
四、 实验结果	11
五、 实验遇到的问题及其解决方法	13
六、 实验结论	14

一、 实验目的

1. 加深对 RSA 算法基本工作原理的理解。
2. 掌握基于 RSA 算法的保密通信系统的基本设计方法。
3. 掌握在 Linux 操作系统实现 RSA 算法的基本编程方法。
4. 了解 Linux 操作系统异步 IO 接口的基本工作原理。

二、 实验内容

1. 要求在 Linux 操作系统中完成基于 RSA 算法的自动分配密钥加密聊天程序的编写。
2. 应用程序保持第三章“基于 DES 加密的 TCP 通信”中示例程序的全部功能，并在此基础上进行扩展，实现密钥自动生成，并基于 RSA 算法进行密钥共享。
3. 要求程序实现全双工通信，并且加密过程对用户完全透明。

三、 实验步骤及实验结果

(一) 实验环境

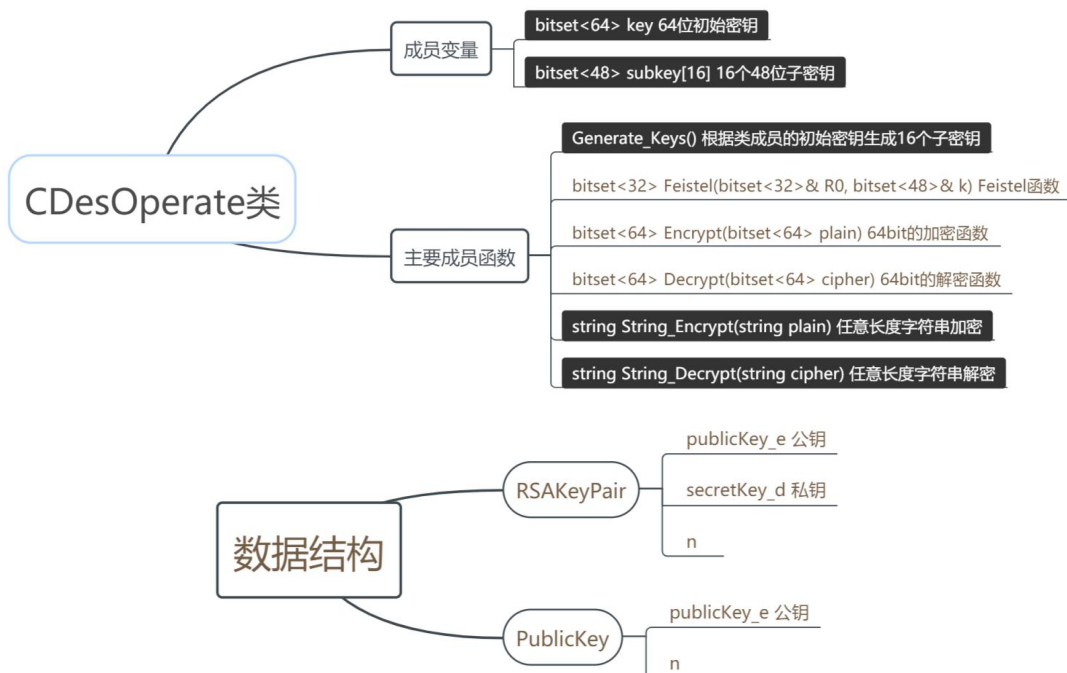
macOS 12.3(基于 unix), C++11, Cmake

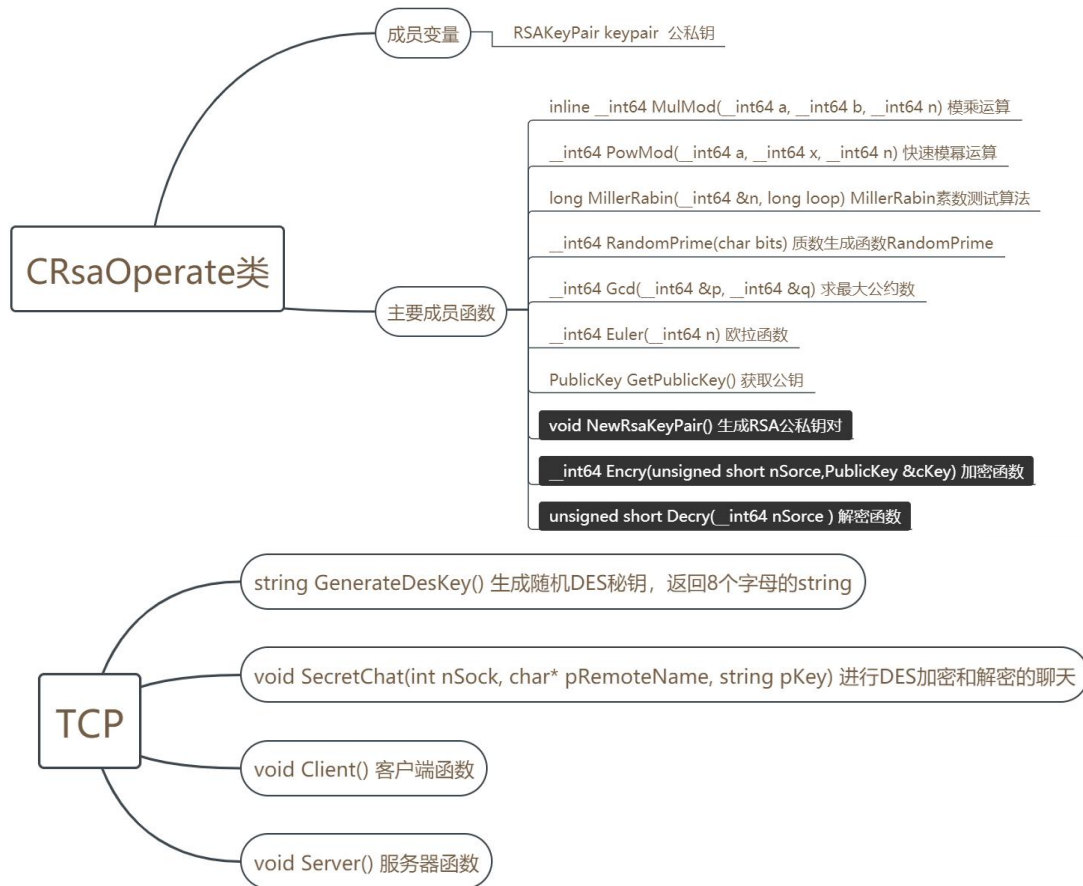
(二) 代码结构

CDesOperate.h CDesOperate.cpp 封装了 DES 加密解密的 CDesOperate 类。

CRsaOperate.h CRsaOperate.cpp 封装了 RSA 加密解密的 CRsaOperate 类。

head.h main.cpp 服务器和客户端的处理函数。





(三) 代码实现

1. 封装 DES 加密解密算法的类

CDesOperate 类较实验一没有发生变化, 后文对此类不加赘述。

2. 封装 RSA 加密解密算法的类, 并检验其正确性。

建立两个数据结构, 存储 RSA 公私钥对的全部数据

```

1 struct PublicKey {
2     __int64 nE;
3     __int64 nN;
4 };
5
6 struct RSAKeyPair {
7     __int64 publicKey_e;
8     __int64 secretKey_d;
9     __int64 n;
10 };

```

m_cParament, 该变量是存储 RSA 的基本参数, 定义如下:

```

1 struct RsaParam{
2     __int64 e;
3     __int64 n;
4     __int64 d;

```

```
5  __int64 f;  
6  __int64 p;  
7  __int64 q;  
8  __int64 s;  
9  };
```

编写 RSA.h 和 RSA.cpp 两个文件，在 RSA.h 文件中进行函数与类的声明，在 RSA.cpp 文件中进行定义。

CRsaOperate 类中的内容

- MulMod()
模乘运算即计算两个数的乘积然后取模
- PowMod()
快速模幂运算
- MillerRabin()
MillerRabin 素数测试算法
- RandomPrime()
质数生成函数 RandomPrime
- Gcd()
求最大公约数
- Euler()
欧拉函数：返回小于 n 且与 n 互质的正整数个数
- GetPublicKey()
获取公钥
- NewRsaKeyPair()
生成 RSA 公私钥对
- Encry()
加密函数
- Decry()
解密函数

```
1  class CRsaOperate {  
2  public:  
3      RsaParam m_cParament;  
4      CRsaOperate();  
5      static inline __int64 MulMod(__int64 a, unsigned long b, unsigned long n);  
6      static __int64 PowMod(__int64 base, __int64 pow, __int64 n);  
7      static long RabinMillerKn1(__int64 &n);  
8      static long RabinMiller(__int64 &n, long loop);  
9      static __int64 RandPrime(char bit);  
10     static __int64 Gcd(__int64 &p, __int64 &q);  
11     static __int64 Euclid(__int64 e, __int64 t_n);  
12     static __int64 Encry(unsigned short nScore, PublicKey &cKey);  
13     unsigned short Decry(__int64 nScore);  
14     PublicKey GetPublicKey();  
15 };
```

- 模乘运算

模乘运算即计算两个数的乘积然后取模

```
1 inline __int64 CRsaOperate::MulMod(__int64 a, unsigned long b, unsigned long n) {  
2     return (a % n) * (b % n) % n;  
3 }
```

- 快速模幂运算

利用快速幂算法进行快速模幂计算，可以在保证数据准确的情况下，快速得到高次幂模运算后的结果。

```
1 __int64 CRsaOperate::PowMod(__int64 base, __int64 pow, __int64 n) {  
2     __int64 a = base, b = pow, c = 1;  
3     while(b){  
4         while(!(b & 1)){  
5             b >>= 1;  
6             a = MulMod(a, a, n);  
7         }  
8         b--;  
9         c = MulMod(a, c, n);  
10    }  
11    return c;  
12 }
```

- MillerRabin 素数测试算法

概率算法：MillerRabin 素数测试算法

缺省参数 loop 为默认重复测试参数为 100

Rabin-Miller 算法具有较低的时间复杂度，但是无法保证所得结果一定是正确的，而是以一定的概率得到一个素数，所以需要设定一个安全参数来保证 Rabin-Miller 算法以高概率得到一个素数。

```
1 long CRsaOperate::RabinMillerKn1(__int64 &n) {  
2     __int64 a, q, k, v;  
3     q = n - 1;  
4     k = 0;  
5     while(!(q & 1)) {  
6         ++k;  
7         q >>= 1;  
8     }  
9     a = 2 + rand() % (n - 3);  
10    v = PowMod(a, q, n);  
11    if(v == 1) {  
12        return 1;  
13    }  
14    for(int j = 0; j < k; j++) {  
15        unsigned int z = 1;  
16        for(int w = 0; w < j; w++) {  
17            z *= 2;  
18        }  
19        if(PowMod(a, z*q, n) == n - 1)  
20            return 1;  
21    }  
22    return 0;  
23 }
```

```

24
25
26 long CRsaOperate::RabinMiller(__int64 &n, long loop=100) {
27     for(long i = 0; i < loop; i++){
28         if(!RabinMillerKnl(n)){
29             return 0;
30         }
31     }
32     return 1;
33 }

```

- 质数生成函数 RandomPrime

保证最高位是 1，再加上一个随机数，保证最低位是 1，即保证是奇数，进行拉宾 - 米勒测试 30 次，全部通过认为是质数

```

1  __int64 CRsaOperate::RandPrime(char bit) {
2      __int64 base;
3      do{
4          base = (unsigned long)1 << (bit - 1);
5          base += rand() % (base);
6          base |= 1;
7      }
8      while(!RabinMiller(base, 30));
9      return base;
10 }

```

- 求最大公约数

两数相等，最大公约数就是本身；辗转相除法， $\text{gcd}(a,b)=\text{gcd}(b, a-qb)$ 。求解方式这个方程组的方法是拓展的欧几里得算法。

```

1  __int64 CRsaOperate::Gcd(__int64 &p, __int64 &q) {
2      unsigned long long a = p > q ? p : q;
3      unsigned long long b = p < q ? p : q;
4      unsigned long long t;
5      if( p == q ){
6          return p;
7      }else{
8          while(b){
9              a = a % b;
10             t = a;
11             a = b;
12             b = t;
13         }
14         return a;
15     }
16 }

```

- 欧拉函数: 返回小于 n 且与 n 互质的正整数个数

```

1  __int64 CRsaOperate::Euclid(__int64 e, __int64 t_n) {
2      unsigned long long Max = 0xffffffffffffffff - t_n;
3      unsigned long long i = 1;
4      while(1){
5          if(((i*t_n)+1)%e == 0){

```

```

6         return ((i*t_n)+1)/e;
7     }
8     i++;
9     unsigned long long Tmp = (i+1)*t_n;
10    if(Tmp > Max){
11        return 0;
12    }
13 }
14 return 0;
15 }

```

- 公私钥对生成

除了加解密函数外，另外一个重要的内容是 RSA 大素数生成问题。在 CRSASection 类中 RsaGetParam 函数是用来生成 RSA 所有必要的参数，其中主要内容都是生成两个大素数。

```

1 RsaParam RsaGetParam() {
2     RsaParam Rsa = { 0 };
3     unsigned long long t;
4     Rsa.p = CRsaOperate::RandPrime(16);
5     Rsa.q = CRsaOperate::RandPrime(16);
6     Rsa.n = Rsa.p * Rsa.q;
7     Rsa.f = (Rsa.p - 1) * (Rsa.q - 1);
8     do {
9         Rsa.e = rand() % Rsa.f;
10        Rsa.e != 1;
11    }
12    while(CRsaOperate::Gcd(Rsa.e, Rsa.f) != 1);
13    Rsa.d = CRsaOperate::Euclid(Rsa.e, Rsa.f);
14    Rsa.s = 0;
15    t = Rsa.n >> 1;
16    while(t) {
17        Rsa.s++;
18        t >>= 1;
19    }
20    return Rsa;
21 }

```

- 获取公钥对

```

1 PublicKey CRsaOperate::GetPublicKey() {
2     PublicKey cTmp;
3     cTmp.nE = this -> m_cParament.e;
4     cTmp.nN = this -> m_cParament.n;
5     return cTmp;
6 }

```

- RSA 解密函数

```

1 unsigned short CRsaOperate::Decry(__int64 nScore) {
2     unsigned long long nRes = PowMod(nScore, m_cParament.d, m_cParament.n);
3     unsigned short *pRes = (unsigned short *)&(nRes);
4     if(pRes[1] != 0 || pRes[3] != 0 || pRes[2] != 0) {
5         return 0;
6     }
7     else {

```



```

8         return pRes[0];
9     }
10 }

```

3. 通讯部分

服务端将公钥发送给希望与其通话的客户端，然后客户端就可以通过公钥加密生成的 DES 密钥发送给服务端，服务端收到消息后利用自己的私钥进行解密便成功与客户端得到了相同的会话密钥。

- 客户端函数

与服务器建立连接后，客户端接收到服务器发送的 RSA 公钥对后，随机生成 DES 密钥并用公钥对加密发送给服务器；此后，用 DES 密钥进行聊天。

```

1  else {
2      std::cout << "Please input the server address:" << std::endl;
3      char strIPAddr[16];
4      std::cin >> strIPAddr;
5      int nConnectSocket, nLength;
6      struct sockaddr_in sDestAddr;
7      if((nConnectSocket = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
8          perror("Socket");
9          exit(errno);
10     }
11     int SEVERPORT = 6000;
12
13     sDestAddr.sin_family = AF_INET;
14     sDestAddr.sin_port = htons(SEVERPORT);
15     sDestAddr.sin_addr.s_addr = inet_addr(strIPAddr);
16     if(connect(nConnectSocket, (struct sockaddr *) &sDestAddr, sizeof(sDestAddr))
17        != 0) {
18         perror("Connect");
19         exit(errno);
20     }
21     else {
22         printf("Connect Success! \n");
23         char *strDesKey = new char [8];
24         GerenateDesKey(strDesKey);
25         printf("Create DES key success\n");
26         PublicKey cRsaPublicKey;
27         if(sizeof(cRsaPublicKey) == TotalRecv(nConnectSocket, (char *)&cRsaPublicKey
28            , sizeof(cRsaPublicKey), 0)) {
29             printf("Successful get the RSA public Key\n");
30         }
31         else {
32             perror("Get RSA public key ");
33             exit(0);
34         }
35     }
36     unsigned long long nEncryptDesKey[4];
37     unsigned short *pDesKey = (unsigned short *)strDesKey;
38     for(int i = 0; i < 4; i++) {
39         nEncryptDesKey[i] = CRsaOperate::Encry(pDesKey[i], cRsaPublicKey);
40     }
41     if(sizeof(unsigned long long)*4 != send(nConnectSocket, (char *)
42        nEncryptDesKey, sizeof(unsigned long long)*4, 0)) {

```

```

39     perror("Send DES key Error");
40     exit(0);
41 }
42 else {
43     printf("Successful send the encrypted DES Key\n");
44 }
45 printf("Begin to chat...\n");
46 SecretChat(nConnectSocket, strIPAddr, strDesKey);
47 }
48 close(nConnectSocket);
49 }

```

• 服务器函数

与客户端建立连接后，生成 RSA 公私钥对，并将公钥对发送给客户端，然后接收客户端发来的加过密的 DES 密钥，用自己的私钥对解密；此后，用 DES 密钥进行通信。

```

1  if(temp == 's') {
2      int nListenSocket, nAcceptSocket;
3      struct sockaddr_in sLocalAddr, sRemoteAddr;
4      bzero(&sLocalAddr, sizeof(sLocalAddr));
5      sLocalAddr.sin_family = PF_INET;
6      sLocalAddr.sin_port = htons(6000);
7      sLocalAddr.sin_addr.s_addr = INADDR_ANY;
8      if ((nListenSocket = socket(PF_INET, SOCK_STREAM, 0)) == -1) {
9          perror("socket");
10         exit(1);
11     }
12     if(bind(nListenSocket, (struct sockaddr *) &sLocalAddr, sizeof(struct
13         sockaddr)) == -1) {
14         perror("bind");
15         exit(1);
16     }
17     if(listen(nListenSocket, 5) == -1) {
18         perror("listen");
19         exit(1);
20     }
21     printf("Listening...\n");
22     socklen_t nLength = 0;
23     nAcceptSocket = accept(nListenSocket, (struct sockaddr*)&sRemoteAddr, &
24         nLength);
25     close(nListenSocket);
26     printf("server: got connection from %s, port %d, socket %d\n", inet_ntoa(
27         sRemoteAddr.sin_addr), ntohs(sRemoteAddr.sin_port), nAcceptSocket);
28
29     //negotiate key
30     PublicKey cRsaPublicKey;
31     CRsaOperate cRsaOperate;
32     //CRsaOperate.show_par();
33     cRsaPublicKey = CRsaOperate.GetPublicKey();
34     if(send(nAcceptSocket, (char *)&cRsaPublicKey, sizeof(cRsaPublicKey), 0) !=
35         sizeof(cRsaPublicKey)){
36         perror("send");
37         exit(0);
38     }else{
39         printf("successful send the RSA public key. \n");

```

```

36     }
37     unsigned long long nEncryptDesKey[4];
38     char *strDesKey = new char[8];
39     if(4*sizeof(unsigned long long) != TotalRecv(nAcceptSocket,(char *)
        nEncryptDesKey, 4*sizeof(unsigned long long),0)) {
40         perror("TotalRecv DES key error");
41         exit(0);
42     }
43     else {
44         printf("successful get the DES key\n");
45         unsigned short * pDesKey = (unsigned short *)strDesKey;
46         for(int i = 0; i < 4; i++) {
47             pDesKey[i] = CRsaOperate.Decry(nEncryptDesKey[i]);
48             //cout << pDesKey[i] << endl;
49         }
50     }
51
52     printf("Begin to chat...\n");
53     SecretChat(nAcceptSocket, inet_ntoa(sRemoteAddr.sin_addr), strDesKey);
54     close(nAcceptSocket);
55 }

```

• SecretChat 函数

用 select 模型重写通信函数。传统的网络通信 I/O 是阻塞 I/O 或者非阻塞 I/O，阻塞 I/O 会使得进程必须暂停下来等到 I/O 事件的完全，非阻塞 I/O 则需要轮询，大量消耗 CPU 资源。因此出现了一种新的网络 I/O 模型——select 模型。select 模型的基本特点是可以一次性监听多个 I/O 请求，虽然本质上还是阻塞 I/O，但是由于其具备了一次性处理多个 I/O 的能力，所以可以大大加大网络 I/O 的效率。利用 select 模型重写 SecretChat 函数，首先初始化文件列表、等待时间、返回值。然后开始一个循环进行监听，每次监听之前先对于 select 模型进行询问，查看其中完成的 I/O 请求，以便多次处理。然后对于 sock 套接字和标准输入的文件描述符进行监听。对于 sock 套接字的监听。一旦存在 sock 套接字的请求便进行处理。对于标准输入的处理。一旦用户进行输入，便进行处理。

```

1 void SecretChat(int nSock, char *pRemoteName, char *pKey) {
2
3     std::cout << pRemoteName << std::endl;
4     CDesOperate cDes;
5     std::cout << pKey << std::endl;
6     int klength = strlen(pKey);
7     if(klength != 8){
8         printf("%s\n", pKey);
9         printf("Key length error\n");
10        return;
11    }
12
13    //select model
14    fd_set cHandleSet;
15    struct timeval tv;
16    int nRet;
17    while(1){
18        FD_ZERO(&cHandleSet);
19        FD_SET(nSock, &cHandleSet);
20        FD_SET(0, &cHandleSet);

```

```

21     tv.tv_sec = 1;
22     tv.tv_usec = 0;
23     nRet = select(nSock>0? nSock+ 1:1, &cHandleSet, NULL, NULL, &tv);
24     if(nRet < 0){
25         printf("Select error!\n");
26         break;
27     }
28     if(nRet == 0){
29         continue;
30     }
31     if(FD_ISSET(nSock,&cHandleSet)){
32         bzero(&strSocketBuffer, BUFFERSIZE);
33         int nLength = 0;
34         nLength = TotalRecv(nSock, strSocketBuffer,BUFFERSIZE,0);
35         if(nLength !=BUFFERSIZE) break;
36         else{
37             int nLen = BUFFERSIZE;
38             cDes.Decry(strSocketBuffer,BUFFERSIZE,strDecryBuffer,nLen,pKey,8);
39             strDecryBuffer[BUFFERSIZE-1]=0;
40             if(strDecryBuffer[0]!=0&&strDecryBuffer[0]!='\n'){
41                 printf("Receive message form <%=>: %s",pRemoteName,strDecryBuffer)
42                     ;
43                 if(0==memcmp("quit",strDecryBuffer,4)){
44                     printf("Quit!\n");
45                     break;
46                 }
47             }
48         }
49     if(FD_ISSET(0,&cHandleSet)){
50         bzero(&strStdinBuffer, BUFFERSIZE);
51         while(strStdinBuffer[0]==0){
52             if (fgets(strStdinBuffer, BUFFERSIZE, stdin) == NULL){
53                 continue;
54             }
55         }
56         int nLen = BUFFERSIZE;
57         cDes.Encry(strStdinBuffer,BUFFERSIZE,strEncryBuffer,nLen,pKey,8);
58         if(send(nSock, strEncryBuffer, BUFFERSIZE,0)!=BUFFERSIZE){
59             perror("send");
60         }else{
61             if(0==memcmp("quit",strStdinBuffer,4)){
62                 printf("Quit!\n");
63                 break;
64             }
65         }
66     }
67 }
68 }

```

- GerenateDesKey 函数

随机生成 DES 密钥。

```

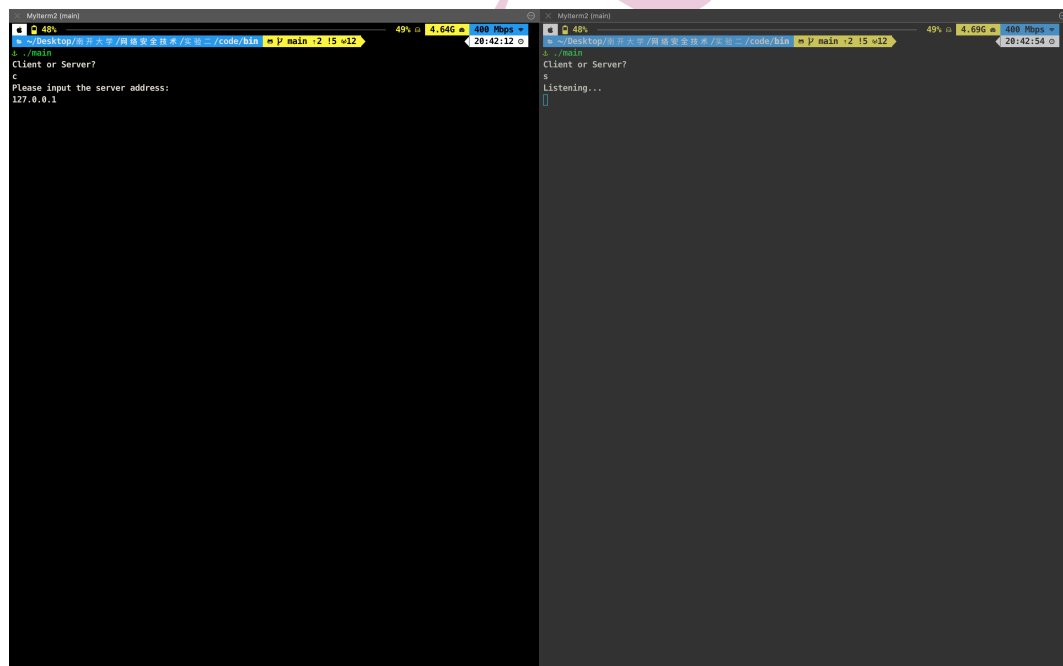
1 void GerenateDesKey(char* x){
2     int i;

```

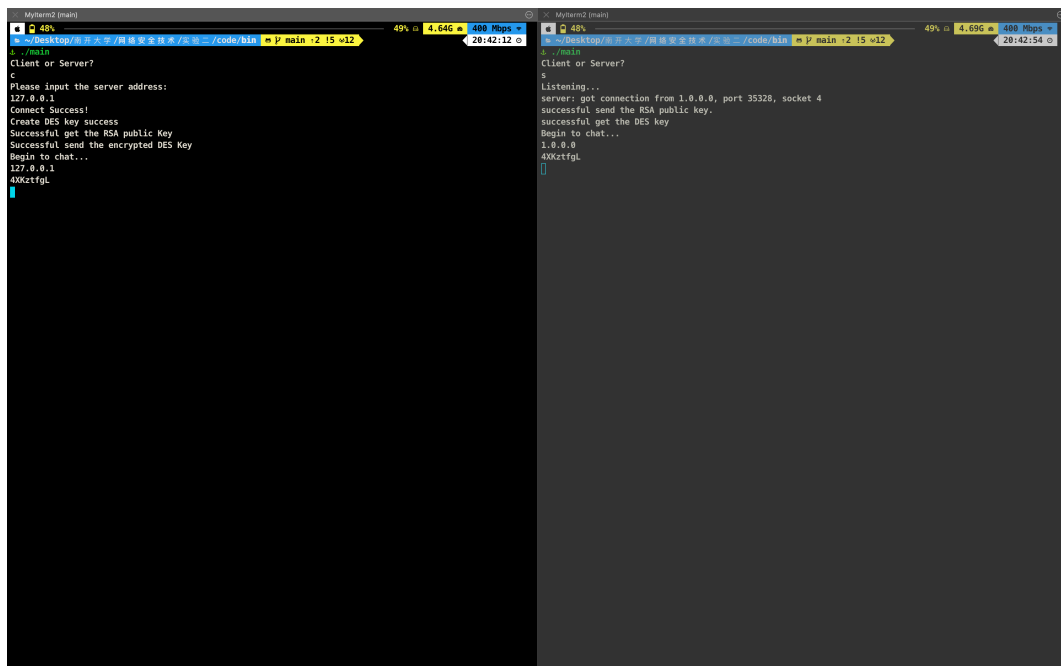
```
3  srand(time(NULL));
4  for (i = 0; i < 8; i++)
5  {
6      switch ((rand() % 3))
7      {
8          case 1:
9              x[i] = 'A' + rand() % 26;
10             break;
11          case 2:
12              x[i] = 'a' + rand() % 26;
13              break;
14          default:
15              x[i] = '0' + rand() % 10;
16              break;
17      }
18  }
19  x[i] = '\0';
20 }
```

四、 实验结果

客户端上线，连接成功



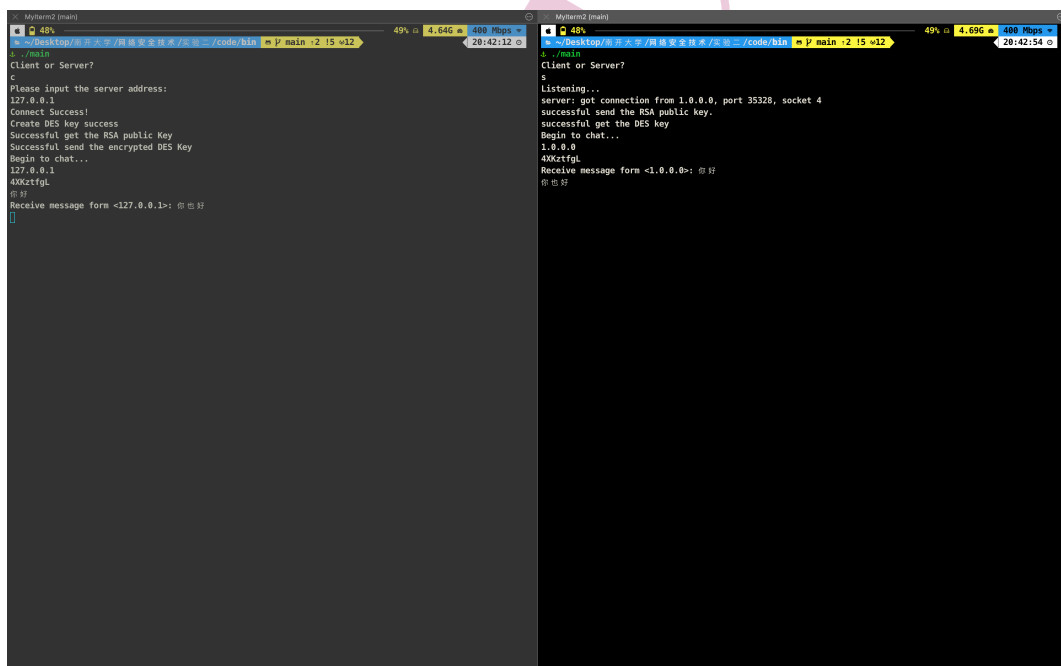
客户端生成 DES 密钥并传输到服务器，服务器解密成功



```
MyTerm2 (main)
49% 4.64G 388 Mbps
./main
Client or Server?
c
Please input the server address:
127.0.0.1
Connect Success!
Create DES Key success
Successful get the RSA public Key
Successful send the encrypted DES Key
Begin to chat...
127.0.0.1
40xztfgl

MyTerm2 (main)
49% 4.69G 488 Mbps
./main
Client or Server?
s
Listening...
server: got connection from 1.0.0.0, port 35328, socket 4
successful send the RSA public key.
successful get the DES key
Begin to chat...
1.0.0.0
40xztfgl
```

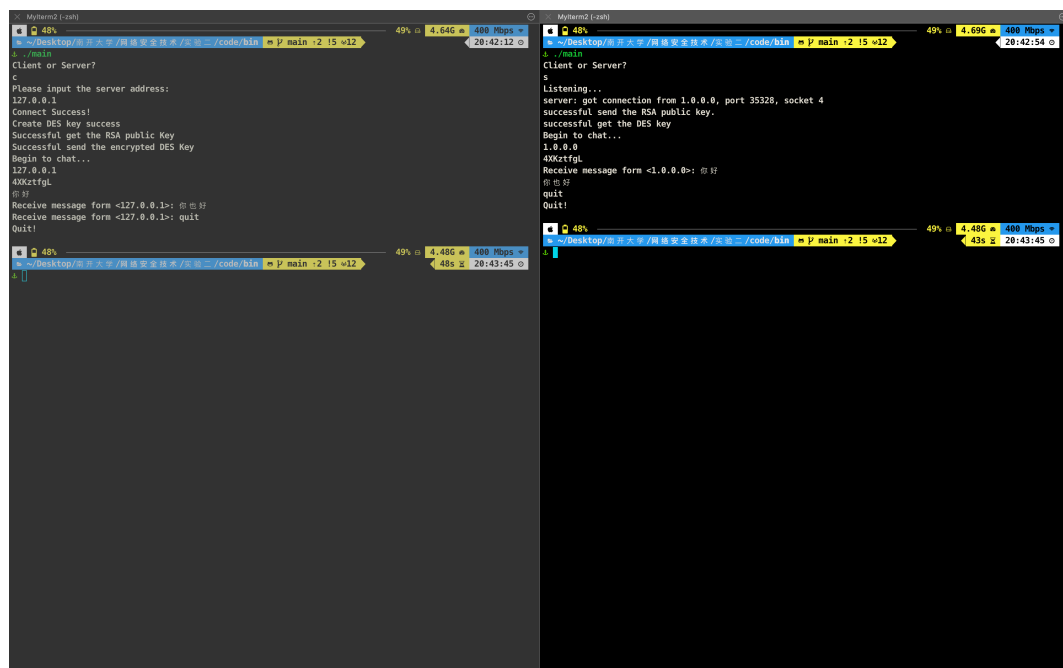
客户端和服务端之间相互通信



```
MyTerm2 (main)
49% 4.64G 388 Mbps
./main
Client or Server?
c
Please input the server address:
127.0.0.1
Connect Success!
Create DES Key success
Successful get the RSA public Key
Successful send the encrypted DES Key
Begin to chat...
127.0.0.1
40xztfgl
Receive message form <127.0.0.1>: 你好

MyTerm2 (main)
49% 4.69G 488 Mbps
./main
Client or Server?
s
Listening...
server: got connection from 1.0.0.0, port 35328, socket 4
successful send the RSA public key.
successful get the DES key
Begin to chat...
1.0.0.0
40xztfgl
Receive message form <1.0.0.0>: 你好
```

输入 Quit 后停止通信

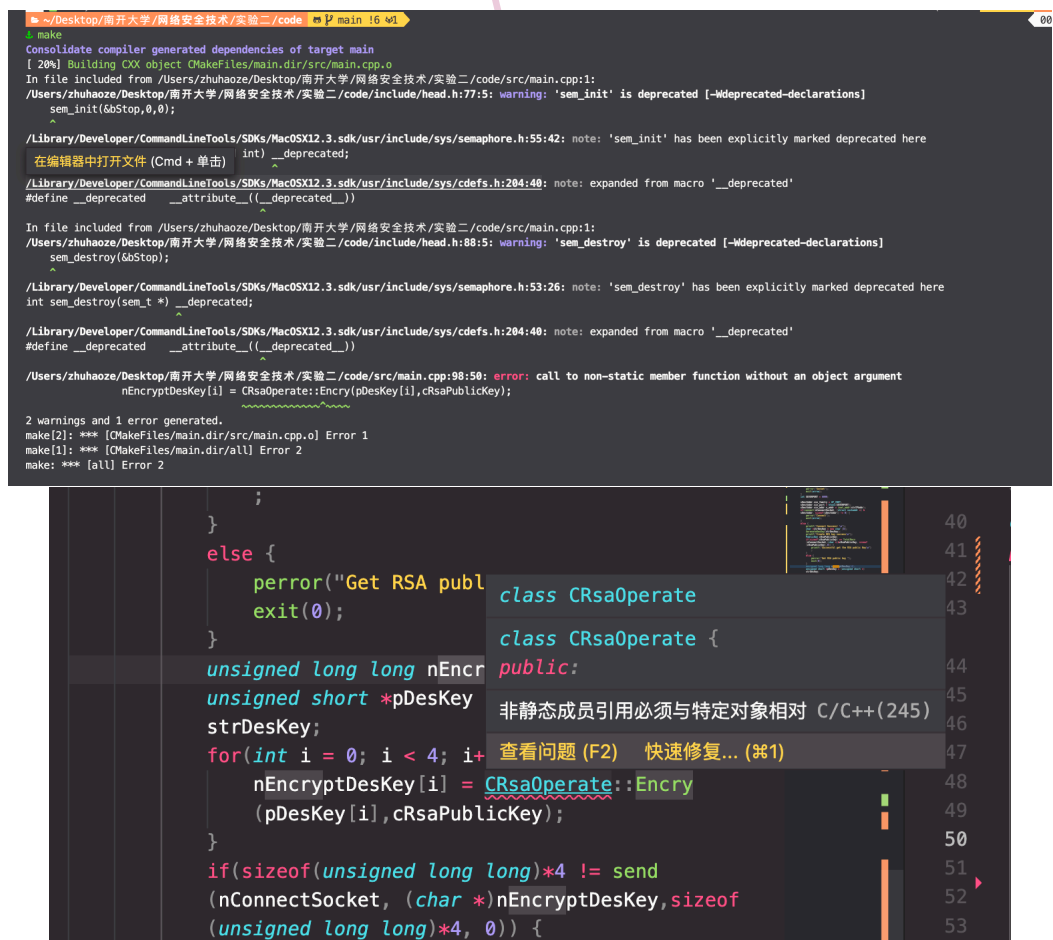


```
MyTerm2 (zsh)
~/Desktop/南开大学/网络安全技术/实验二/code/bin 49% ▣ 4.64G 488 Mbps
$ ./main
Client or Server?
c
Please input the server address:
127.0.0.1
Connect Success!
Create DES Key success
Successful get the RSA public Key
Successful send the encrypted DES Key
Begin to chat...
127.0.0.1
40xztfgl
你好
Receive message form <127.0.0.1>: 你好
Receive message form <127.0.0.1>: quit
Quit!

MyTerm2 (zsh)
~/Desktop/南开大学/网络安全技术/实验二/code/bin 49% ▣ 4.69G 488 Mbps
$ ./main
Client or Server?
s
Listening...
server: got connection from 1.0.0.0, port 35328, socket 4
successful send the RSA public key.
successful get the DES key
Begin to chat...
1.0.0.0
40xztfgl
Receive message form <1.0.0.0>: 你好
你也好
quit
Quit!
```

五、 实验遇到的问题及其解决方法

首先我们在使用 Encry 函数时, 经历了如下报错



```
~/Desktop/南开大学/网络安全技术/实验二/code 49% ▣ 4.64G 488 Mbps
$ make
Consolidate compiler generated dependencies of target main
[ 28%] Building CXX object CMakeFiles/main.dir/src/main.cpp.o
In file included from /Users/zhuhaoye/Desktop/南开大学/网络安全技术/实验二/code/src/main.cpp:1:
/Users/zhuhaoye/Desktop/南开大学/网络安全技术/实验二/code/include/head.h:77:5: warning: 'sem_init' is deprecated [-Wdeprecated-declarations]
    sem_init(&stop,0,0);
    ^
/Library/Developer/CommandLineTools/SDKs/MacOSX12.3.sdk/usr/include/sys/semaphore.h:55:42: note: 'sem_init' has been explicitly marked deprecated here
    int) __deprecated;
    ^
/Library/Developer/CommandLineTools/SDKs/MacOSX12.3.sdk/usr/include/sys/cdefs.h:204:40: note: expanded from macro '__deprecated'
#define __deprecated __attribute__((__deprecated__))
^
In file included from /Users/zhuhaoye/Desktop/南开大学/网络安全技术/实验二/code/src/main.cpp:1:
/Users/zhuhaoye/Desktop/南开大学/网络安全技术/实验二/code/include/head.h:88:5: warning: 'sem_destroy' is deprecated [-Wdeprecated-declarations]
    sem_destroy(&stop);
    ^
/Library/Developer/CommandLineTools/SDKs/MacOSX12.3.sdk/usr/include/sys/semaphore.h:53:26: note: 'sem_destroy' has been explicitly marked deprecated here
int sem_destroy(sem_t *) __deprecated;
^
/Library/Developer/CommandLineTools/SDKs/MacOSX12.3.sdk/usr/include/sys/cdefs.h:204:40: note: expanded from macro '__deprecated'
#define __deprecated __attribute__((__deprecated__))
^
/Users/zhuhaoye/Desktop/南开大学/网络安全技术/实验二/code/src/main.cpp:98:50: error: call to non-static member function without an object argument
    nEncryptDesKey[i] = CRsaOperate::Encry(pDesKey[i],cRsaPublicKey);
                                ^
2 warnings and 1 error generated.
make[2]: *** [CMakeFiles/main.dir/src/main.cpp.o] Error 1
make[1]: *** [CMakeFiles/main.dir/all] Error 2
make: *** [all] Error 2

}
else {
    perror("Get RSA publ
    exit(0);
}
unsigned long long nEncr
unsigned short *pDesKey
strDesKey;
for(int i = 0; i < 4; i+
    nEncryptDesKey[i] = CRsaOperate::Encry
        (pDesKey[i],cRsaPublicKey);
}
if(sizeof(unsigned long long)*4 != send
    (nConnectSocket, (char *)nEncryptDesKey,sizeof
    (unsigned long long)*4, 0)) {
}
```

可以看出，其问题在于，我们不能直接调用类的函数。为了解决这个问题，我们查阅了资料得知，应当在其前面加上 `static` 声明其为静态函数，便可以在类外直接调用。

然后是如何将整型（按 ASCII 码值）转换成 `string` 类型的问题。服务器接收到 DES 的密文分别是四个短整型（2 个字大小）的值。如果使用 `to_string` 函数，则是将数字换成对应的字符串数字，如 65 到 “65”，并没有根据 ASCII 码转换成对应的字符 “A”。而如果仅仅在转换的短整型前面加上 `(char*)&`，则只能将短整型的第一个字节转换成字符，而少了一个。最终，我们使用 `sprintf` 函数解决了这个转换的问题。

六、 实验结论

本次实验在第一次实验，基于 TCP 的 DES 加密通讯的基础上实现了随机生成 DES 密钥、随机生成 RSA 公钥、密钥，并且通过 RSA 加密方法共享 DES 密钥。加深了对 RSA 非对称加密算法的步骤的理解。本实验只使用了 RSA 算法对 DES 密钥的传输进行加密解密，而不是完全使用 RSA 加密解密，是因为 RSA 算法执行的运算量比较大，系统消耗的资源会比较大。而只对 DES 密钥共享进行 RSA 加密，既保证了通信的安全，又节省了资源。此外值得注意的是，由于客户端加密 DES 密钥时，只能获取 RSA 的公钥对，而 RSA 类中的成员变量是公私钥对（对于客户端不可见），所以 RSA 加密函数中的参数既包括了明文分组，又包括公钥对。

此外，我们较实验一的基础上，进一步掌握了 Linux 下的通信机制。使用 `select` 函数替换了原来使用 `fork` 进程的方法，减少了进程开销，在实现上也更加简单。