

# 网络安全技术实验报告—— 基于 RSA 算法自动分配密钥的加密聊天程序

---

学院 网安学院

年级 大三

班级 信息安全单学位班

学号 1811494

姓名 刘旭萌

手机号 17320205058

## 一、实验目的

---

在讨论了传统的对称加密算法 DES 原理与实现技术的基础上，本章将以典型的非对称密码体系中 RSA 算法为例，以基于 TCP 协议的聊天程序加密为任务，系统地进行非对称密码体系 RSA 算法原理与应用编程技术的讨论和训练。通过练习达到以下的训练目的：

- 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。

本章编程训练的要求如下。

- 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

## 二、实验内容

---

本章训练要求读者在第三章“基于 DES 加密的 TCP 通信”的基础上进行二次开发，使原有的程序可以实现全自动生成 DES 密钥以及基于 RSA 算法的密钥分配。

- (1) 要求在 Linux 操作系统中完成基于 RSA 算法的保密通信程序的编写。
- (2) 程序必须包含 DES 密钥自动生成、RSA 密钥分配以及 DES 加密通讯三个部分。
- (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。
- (4) 用能力的同学可以使用select模型或者异步IO模型对“基于DES加密的TCP通信”一章中 socket 通讯部分代码进行优化

## 三、实验步骤及实验结果

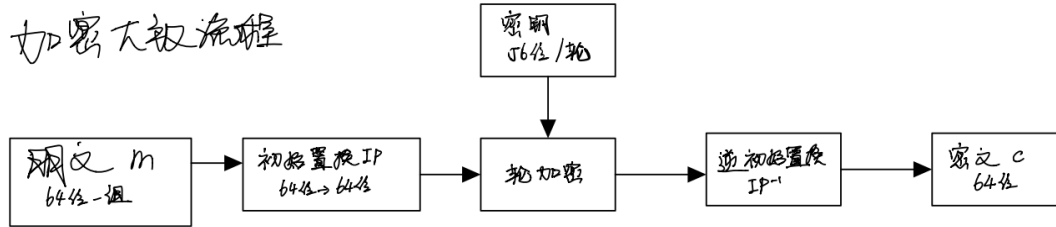
---

# DES加密解密

复用作业“基于DES加密的TCP聊天”中的代码，加解密过程在上一次报告中已经详述。这里只给出大致流程。

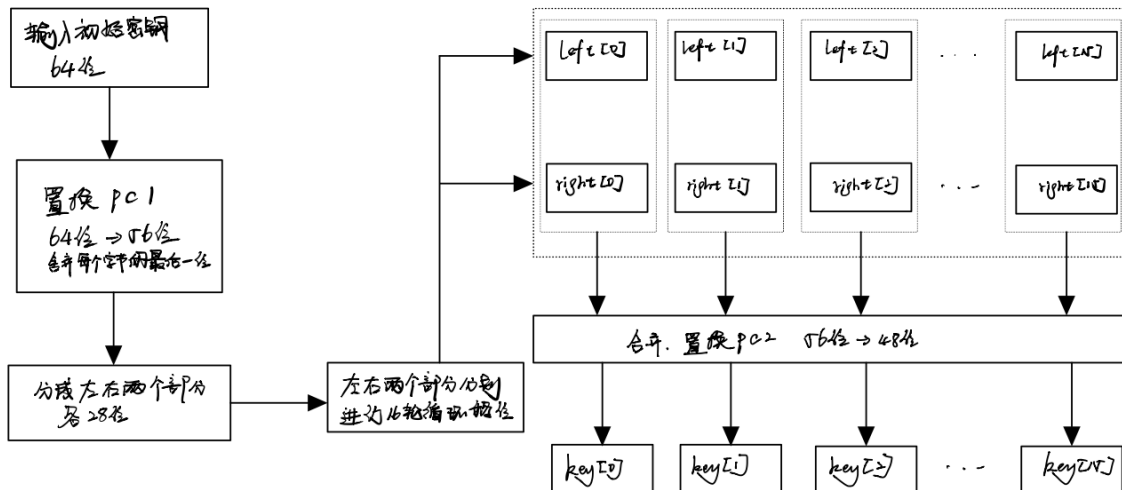
## 加密大致流程

(解密流程相似，只是密钥使用顺序相反)

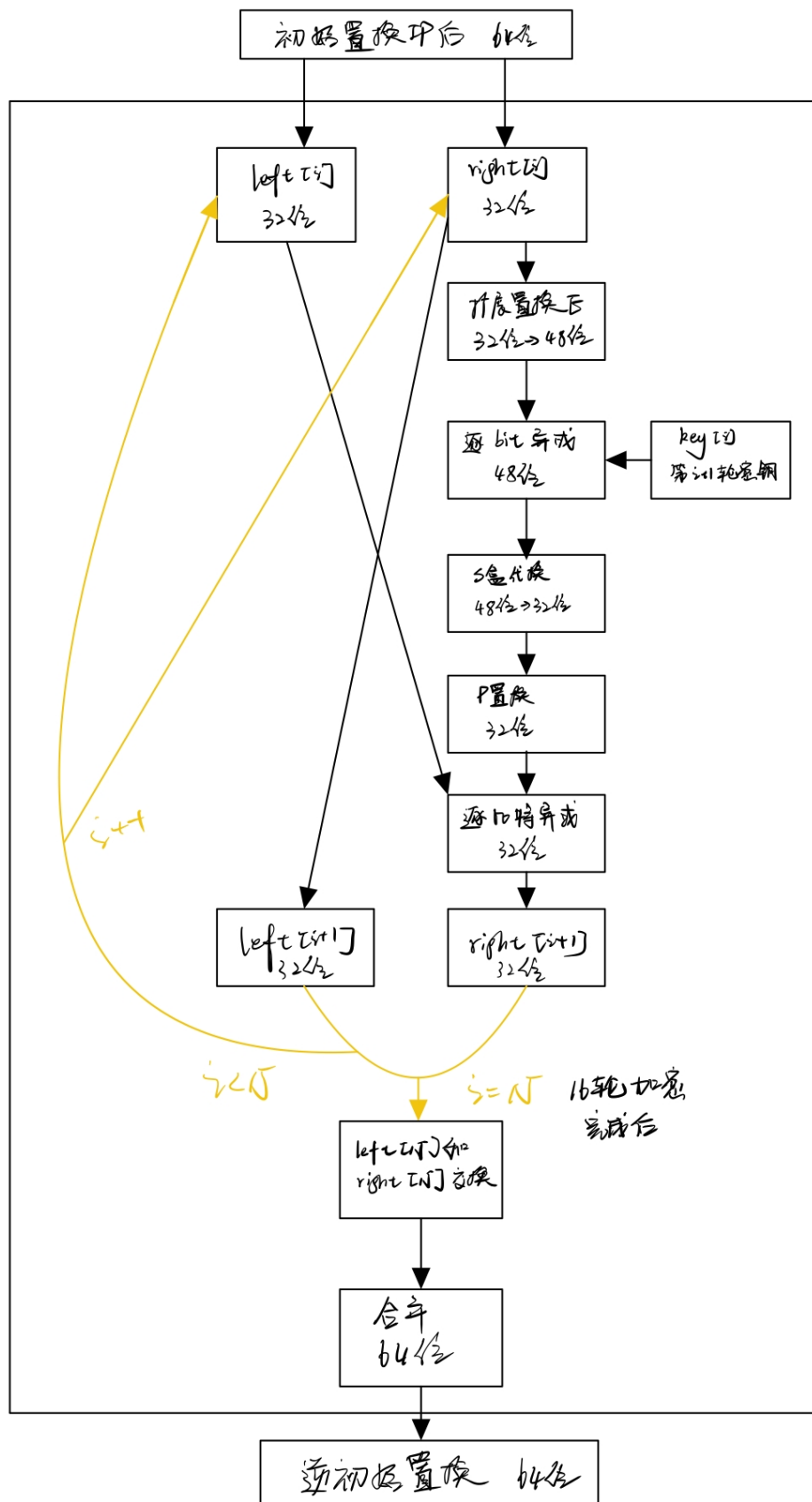


## 密钥生成

密钥生成



## 轮加密



## RSA加密解密

大素数生成 (512位)

原理:

### 1. 线性同余算法生成随机数

- 该算法产生的是伪随机数，只具有统计意义上的随机性，**易被攻破**，不宜在现实情况中使用

- 参数：
  - 模数：  $m(m > 0)$ ，为使随机数的周期尽可能大，  $m$  应尽量大，本次实验取  $m = 2^{31} - 1$
  - 乘数：  $a(0 \leq a < m)$ ，是  $m$  的原根，  $eg. a = 7^5 = 16807$
  - 增量：  $c$  (本次实验中取0)
  - 初值种子：  $X_0$ ，随机选取一32位整数，
- 随机数序列：  $X_{n+1} = (aX_n + c) \bmod n$

## 2. Rabin-Miller素数概率检测算法

- 定理1：如果  $p$  为大于2的素数，则方程  $x^2 \equiv 1 \pmod{p}$  的解只有  $x \equiv 1$  和  $x \equiv -1$
- 定理2：若  $n$  为素数，则  $a^{n-1} \equiv 1 \pmod{n}$
- 每轮检测的伪代码：

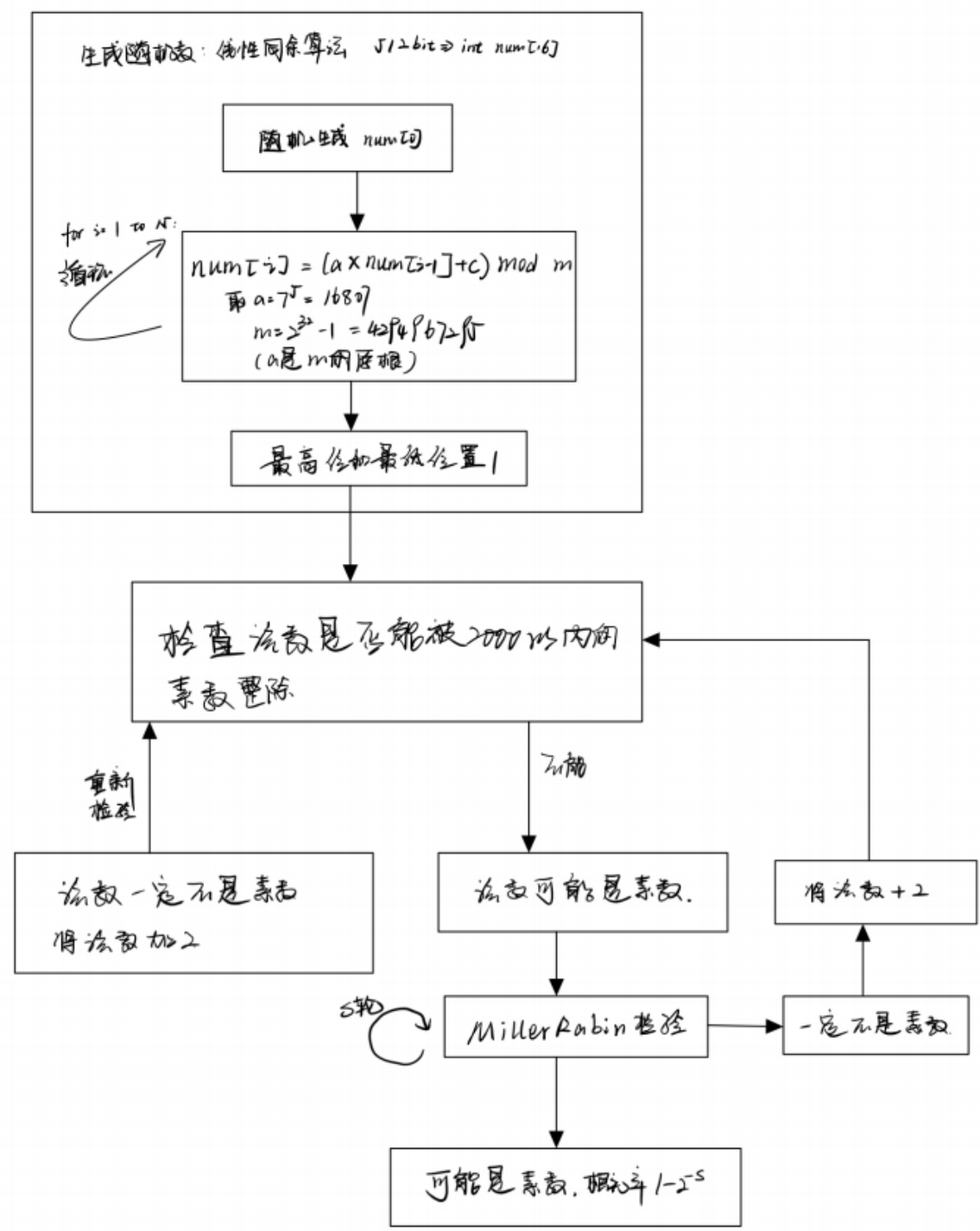
```

1  witness(a,n){
2      d=1;//d初值为1
3      for i=k downto 0 do{
4          x=d;
5          d=(d^2) % n;
6          if(d==1&&x!=1&&x!=n-1)//若为素数，x不可能不是1或n-1
7              return FALSE;
8          if (n-1的2^i位为1)//
9              d=(d*a) % n;
10     }
11     if(d!=1)return FALSE;//定理2
12     return TRUE;
13 }
```

- 该算法为概率性检测，若进行  $s$  轮检测，则是素数的概率至少为  $1 - 2^{-s}$

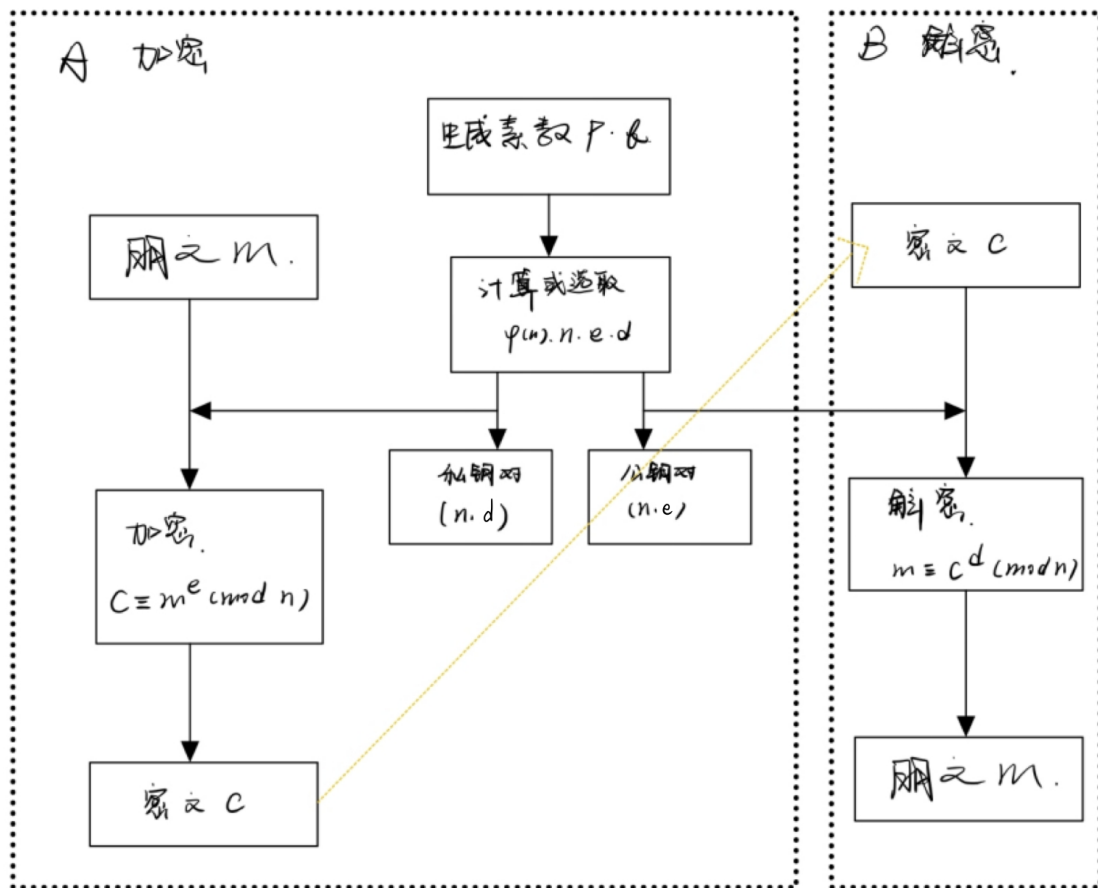
## 3. Eratosthenes筛素数

程序框图：



构建n的长度为1024比特的RSA算法，并利用该算法实现对明文的加密和解密。

程序框图



## 1. 生成密钥对

```

1  RSA::RSA(big p, big q, big e)
2  {
3      big v1;
4      v1.set(1);
5
6      this->p = p;
7      this->q = q;
8      this->e = e;
9
10     n = mul(p, q); // n=pq
11     big p1, q1;
12     p1 = sub(p, v1); // p-1
13     q1 = sub(q, v1); // q-1
14
15     phi = mul(p1, q1); // phi=(p-1)(q-1)
16     d = getinv(phi, e); // 逆元
17 }

```

## 2. 加密

```

1  RSAen::RSAen(RSA_ a, big m)
2  {
3      this->n = a.n;
4      this->e = a.e;
5      this->m = m;
6
7      c = pow(m, e, n); // m^e mod n
8  }

```

### 3. 解密

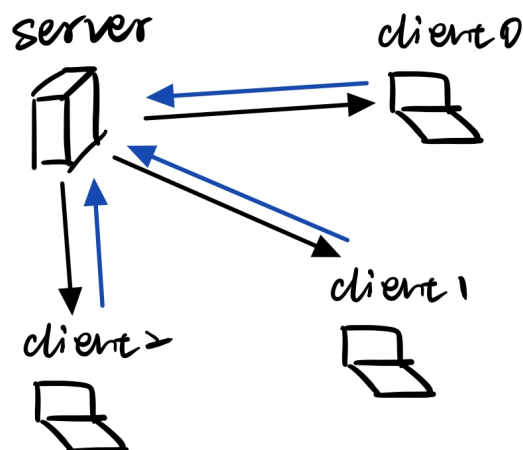
```
1 RSAdc::RSAdc(RSA_ a, big c)
2 {
3     this->n = a.n;
4     this->d = a.d;
5     this->c = c;
6
7     m = pow(c, d, n); // c^d mod n
8 }
```

## TCP通信

### 通信模型图

本次实验为了满足**多对多通信**，采用**C/S**模型进行设计，利用**server**对消息进行**转发**以实现**多个client**端之间的通信

C/S 模型

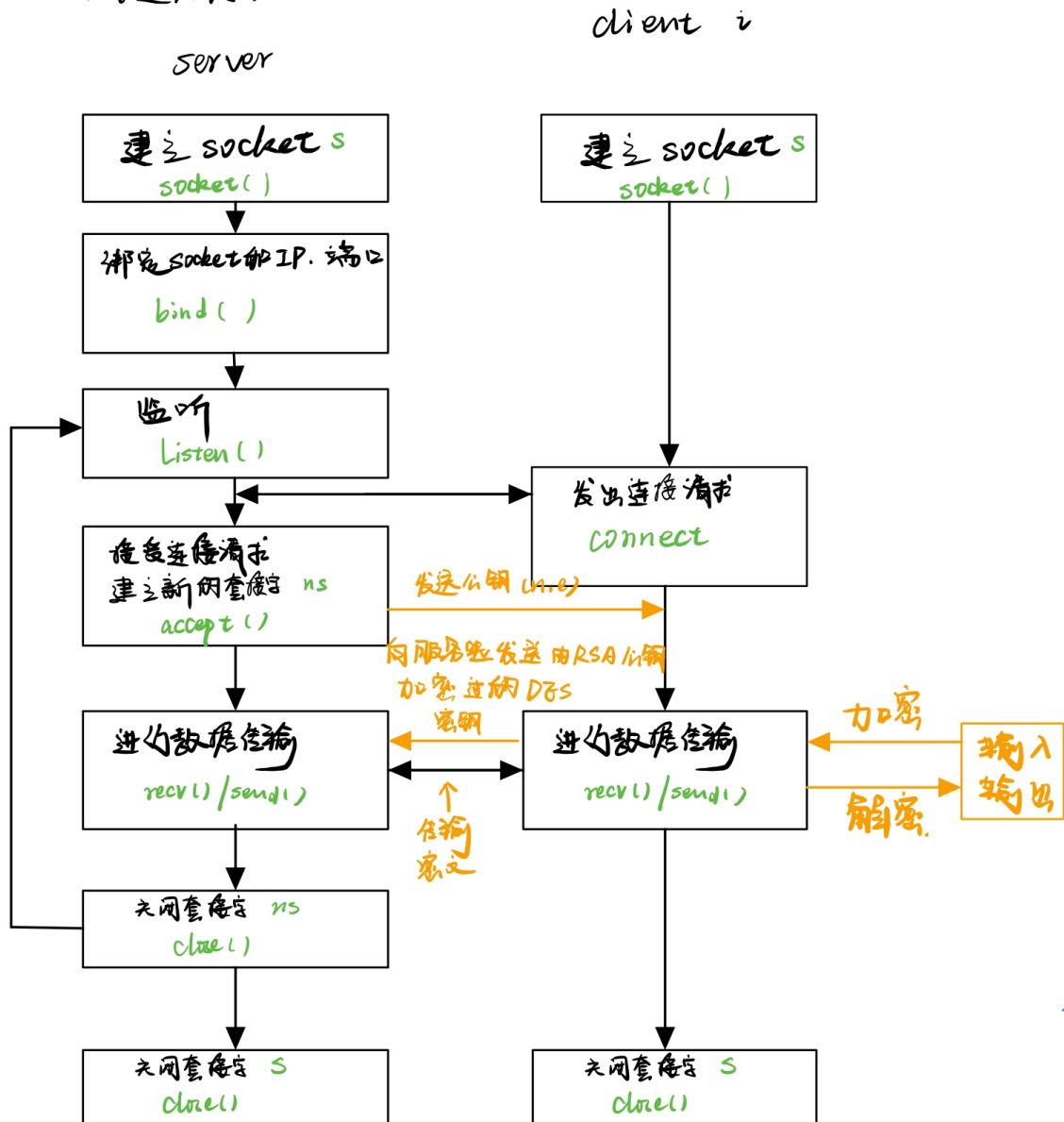


服务器当接收到client[i]想要转发给client[j]的加密消息时，先用deskey[i]进行解密，在用deskey[j]进行加密，并将密文转发至client[j]

### TCP通信原理

与上一个作业大致相似，但略有不同，这里将省略相同部分

# TCP通信原理



## 代码

server生成公钥和私钥

```

1  cout << "是否使用默认密钥对? 1是 2否" << endl;
2  int flag;
3  cin >> flag;
4  prime p, q;
5  if (flag == 1)
6  { // 事先生成好的素数
7  p.set("d0af256c8c72facbc0051054813505a340f899bcb05f4dfb83f2a9a4d14eabd312da
8  a80c24bd772b37fe7cf4a39b8803c37a905cab66365b1721ca4400005a49");
9  }
10 else if (flag == 2)
11 { // 也可以重新生成
12     p.getprime();
13     q.getprime();
14 }
    
```



```

15     cout << "p= "; p.number.print();
16     cout << "q= "; q.number.print();
17     cout << "开始生成密钥对" << endl;
18     RSA_ rsa(p.number, q.number, e);
19     cout << "n= "; rsa.n.print();
20     cout << "d= "; rsa.d.print();
21     cout << "e= "; rsa.e.print();
22     cout << "密钥对已生成" << endl;

```

client接收server发来的**公钥**

```

1 //接收n
2 memset(recvBuf, 0, sizeof(recvBuf));
3 recv(sockClient, recvBuf, 1024, 0);
4 n.stringtonum(recvBuf);
5 //接收e
6 memset(recvBuf, 0, sizeof(recvBuf));
7 recv(sockClient, recvBuf, 1024, 0);
8 e.stringtonum(recvBuf);

```

client对生成的**des密钥进行rsa加密**，并进行发送

```

1 RSA_ secretkey(n, e);
2 rsa_en_text(mtext, ciphertext, secretkey);
3 send(sockClient, ciphertext, 512, 0);

```

server接收client发来的连接请求，并解密**获得对应的DES密钥**

```

1 for (int i = 0; i < CLIENTNUM; i++)
2 {
3     listen(sockSer, 5);
4     sockConn[i] = accept(sockSer, (SOCKADDR*)&addrCli, &len); //失败
    sockConn=INVALID_SOCKET
5     if (sockConn[i] != INVALID_SOCKET)
6     {
7         cond++; //人数加一
8         string buf = "你的id是: ";
9
10        buf += 48 + i; //简化，最多九个人
11
12        send(sockConn[i], buf.data(), 50, 0);
13
14        char buf_[1024] = {};
15        rsa.n.numtostring(buf_);
16        send(sockConn[i], buf_, 1024, 0); //发送n
17        memset(buf_, 1024, 0);
18        e.numtostring(buf_);
19        send(sockConn[i], buf_, 512, 0);
20
21        //接收加密后的密钥
22        memset(buf_, 0, sizeof(buf_));
23        while (1)
24        {
25            recv(sockConn[i], buf_, 1024, 0);
26            //对密钥进行解密
27            if (buf_[0])

```

```

28         {
29             char* mtext = new char[512];
30             rsa_de_text(mtext, buf_, rsa);
31             cout << "DES密钥为"; //为了方便观察, 打印
32             memset(deskey[i], 0, 8);
33             for (int j = 0; j < 8; j++)
34             {
35                 deskey[i][j] += mtext[j * 2];
36                 deskey[i][j] += mtext[j * 2 + 1] * 16;
37                 cout << hex << (int)deskey[i][j] << " ";
38             }
39             cout << endl;
40             //事先生成加密和解密每一轮所需的密钥, 防止重复计算
41             getkeys((u_char*)deskey[i], deskey_final[i].key_final);
42             for (int k = 0; k < 16; k++)
43                 for (int j = 0; j < 6; j++)
44                     deskey_final[i].key_final_[15 - k][j] =
deskey_final[i].key_final[k][j];
45             break;
46         }
47     }
48     //处理完成标志
49     cout << "clients " << i << " have connected" << endl;
50 }

```

server转发接收到的消息

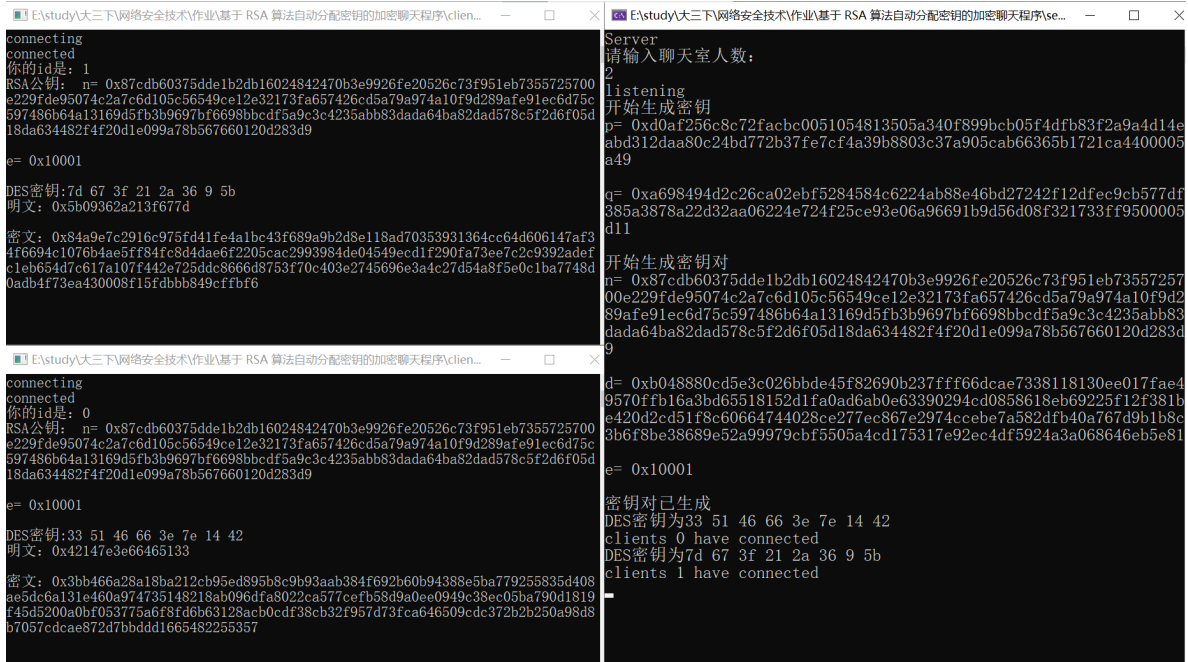
```

1  msg_form m = string_to_msg(recvBuf); //解析报文格式
2  int id = m.to_name - 48;
3
4  msg_form tmp;
5  char buf[BUF_SIZE] = {};
6  //使用source方的密钥解密
7  msg_de(m.msg, deskey_final[(int)lparam].key_final_, buf);
8  cout << buf << endl;
9  //使用destination方的密钥加密
10 msg_en(buf, deskey_final[id].key_final, tmp.msg);
11 //封装报文
12 tmp.from_name = m.from_name;
13 tmp.to_name = m.to_name;
14 //发送消息
15 send(sockConn[id], (const char*)&tmp, 2048, 0);

```

## 运行截图

注: 为了方便观察, 将加密相关信息打印, 实际使用时应对用户透明

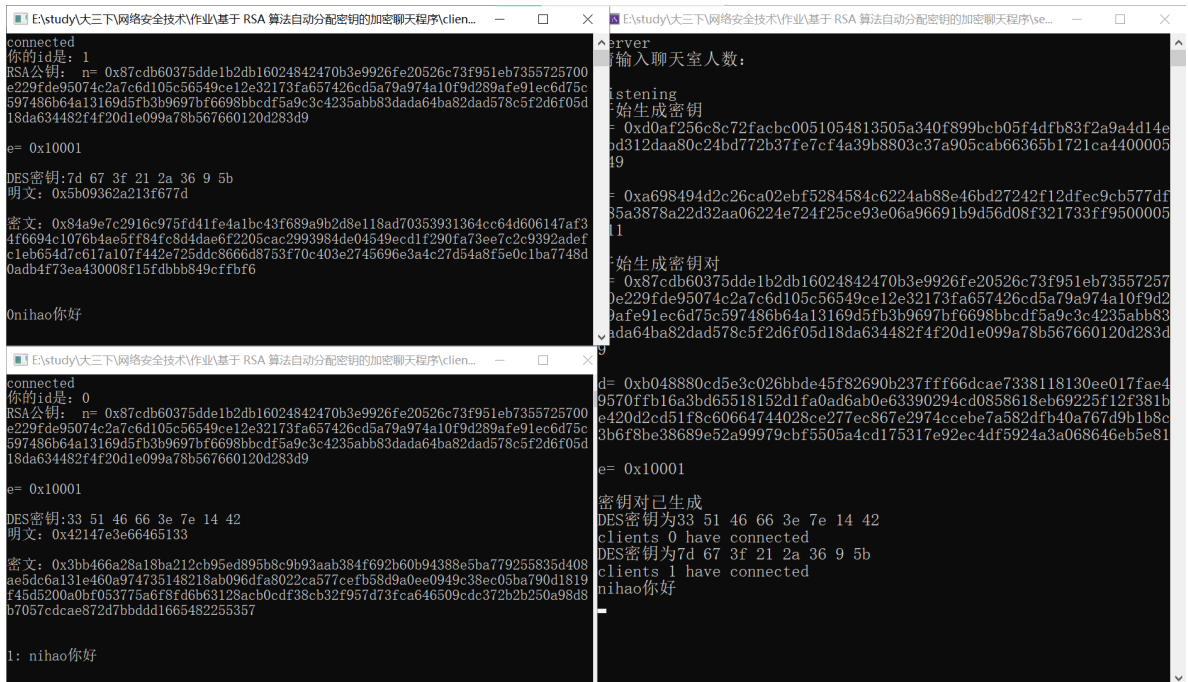


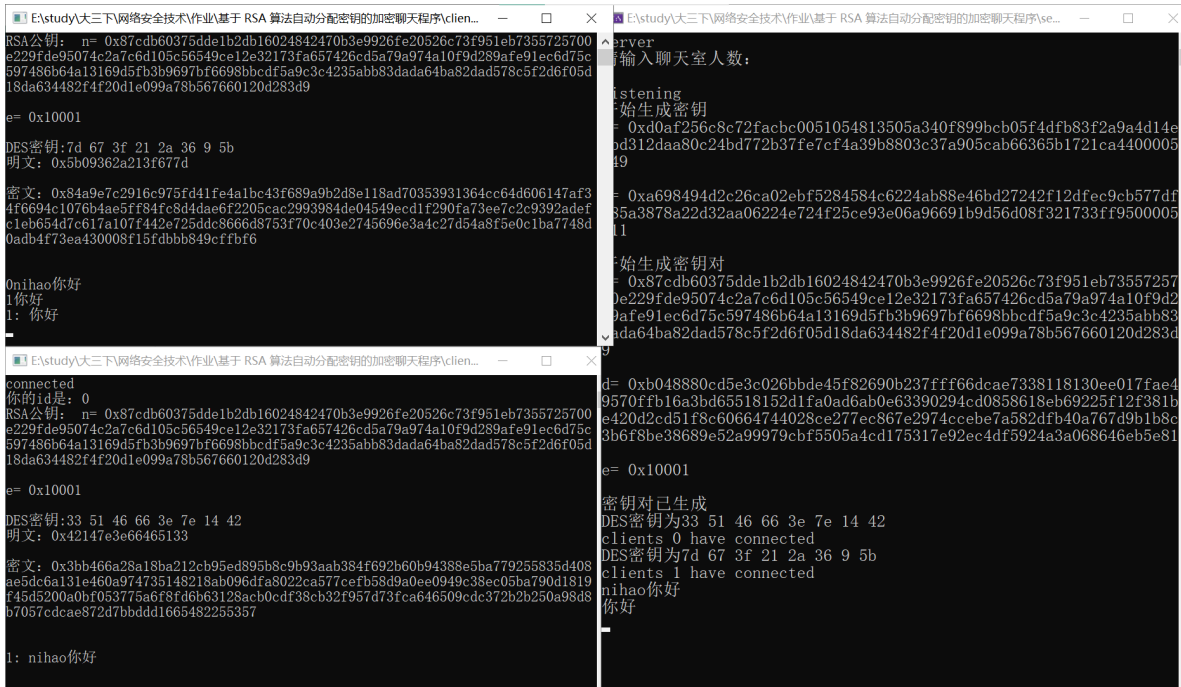
## 通信过程样例:

### 1. 生成大素数

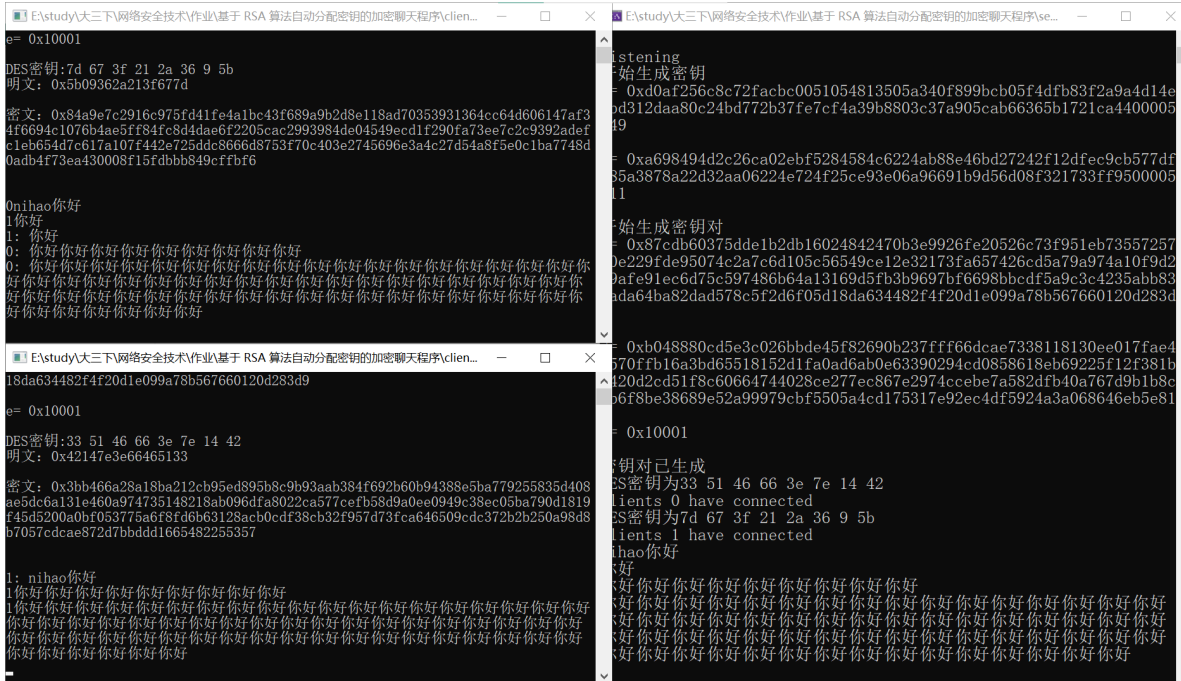
```
是否使用默认密钥对? 1是 2否
2
p= 0x48ab26154aa3cfff7f92018dfb7bd7c2ea37e48505028c0a048246dff25b4d2f5fe20f532df9a1a0157ea6dedff03e23b539b64536f3a2a0144abdd000005eb
q= 0xe0ab33bab8be77cd1333f782596d3ec5441f27caef66c53d81168c925e1c03358c906dda7a5d70aded9793a2a55305a52c1345ea66f43ad1018f4cb2000006e7
开始生成密钥对
n= 0x3fc65a542ae92f25500ef74bab90cfa5660e94dba16295b31a68d9c2b4125743983b0b7fe62b382112e48aff37c4f78f1ace13909869af501d5a376ce5ae76e274a1042bcf77c942351d8b81f302840f9210692da338dbacd8bd2c56cabc347643558fac39eb9002cef047ce6ec3c87492c7c9de467d40fc0923d10028d90d
d= 0x2de991ac187c6989963a4004850e2762d3feced73b49ab15a10353dd833252517afa78a5ed88872cf34b7bde80ebd7417a0db3fe1c0bb1bf9e720d86e34f978f8d69499c14b4e889ae367ef76eb7531b124797be1cc977a72e8d89c0d669a91f987e9804d8ba30e306ca53ef7da75a85160327bf590d42c8a21a1f3d1038e0d
e= 0x10001
```

### 2. 输入目的clientid和消息内容，消息内容可以为中文和英文





## 发送长消息示例



## 程序的退出

client端输入quit，向server发送quit消息后主动退出

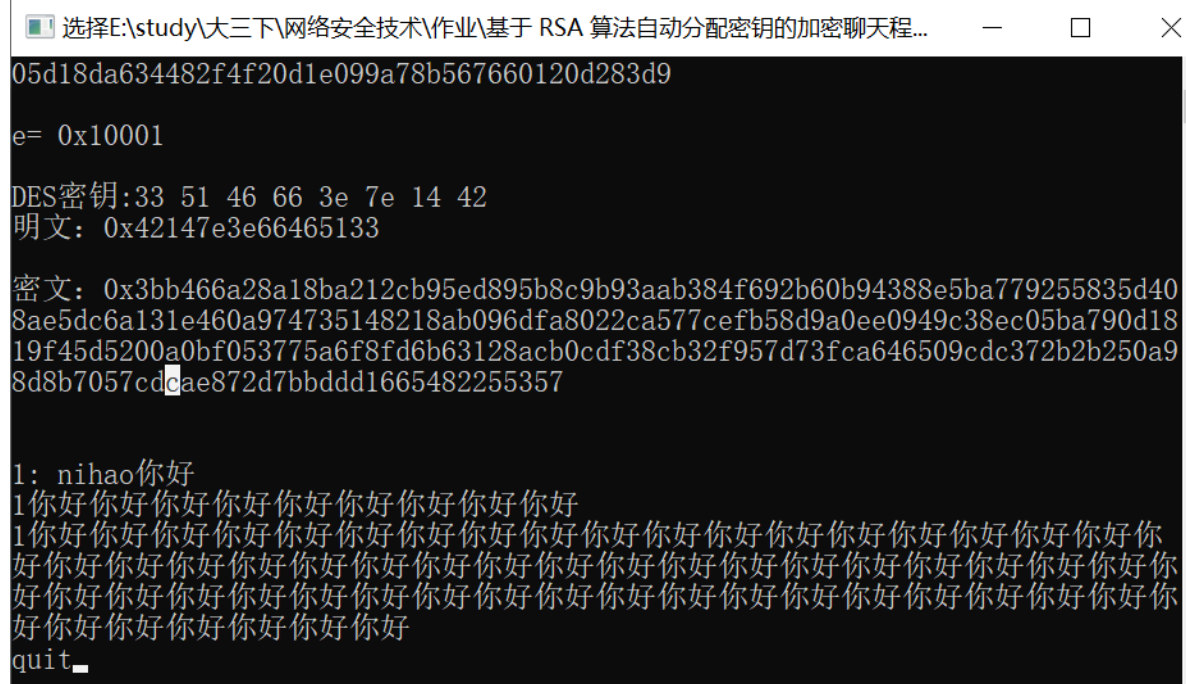
server端收到client发送的quit指令判断该client退出，当所有client都断开连接时退出程序，关闭socket

```
1 if (!strcmp(m.msg, "quit"))
2 {
3     cout << "用户" << (int)lparam << "退出聊天" << endl;
4     cond--; if (!cond) return 0;
5 }
```

关闭socket

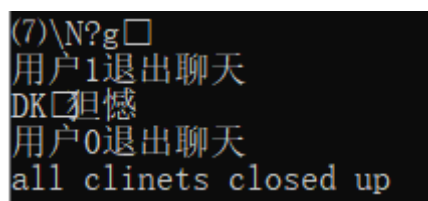
```
1 closesocket(sockSer);
2 WSACleanup();
```

例如,



The screenshot shows a terminal window titled "选择E:\study\大三下\网络安全技术\作业\基于 RSA 算法自动分配密钥的加密聊天程...". The terminal output includes a hexadecimal string "05d18da634482f4f20d1e099a78b567660120d283d9", the value "e= 0x10001", a "DES密钥" (DES key) of "33 51 46 66 3e 7e 14 42", and a "明文" (plaintext) of "0x42147e3e66465133". It then displays a long "密文" (ciphertext) string. Below this, there is a chat log starting with "1: nihao你好", followed by multiple "1你好" messages, and ending with "quit\_".

1. client1发送 quit, 发送后客户端退出, server显示该用户断开连接
2. client0发送 quit, 所有连接断开, server退出



The screenshot shows the following terminal output: "(7)\N?g", "用户1退出聊天", "DK", "用户0退出聊天", and "all clinets closed up".

## 四、实验遇到的问题及其解决方法

1. 栈的使用超过限制  
调整VisualStudio中对于栈大小的限制
2. 使用int[16]代表512位整数导致计算非常复杂, 且经常由于溢出导致错误  
改为每个int代表4位整数, 但是这样可能导致空间占用大, 但是程序更加简便, 编程难度大大降低
3. 加解密结果不正确
  1. 可能是缓冲区接收数据前没有置零
  2. 数据传输补全
  3. 报文格式解析不正确

## 五、实验结论

- 细致地学习了RSA加解密的各个细节

## 附录：大整数运算

```

1  class big//十六进制
2  {
3  public:
4      int num[512];//使用数组保存大整数，每一个元素都代表原数的4个bit，即16进制
5      big();//初始化全零
6      void copy(big);//复制另一个大整数的值
7      void print();
8      void set(unsigned int a);
9      void set(int a);//赋值为a如 set(1)
10     void set(string a);//set(0xa)
11     void set(string a, int x);//set("Hi"),每个字母对应数组中的两个元素
12     void set(unsigned long long a);
13     int getbit(int i);//第2^i位是即，如i=0是零次项位
14 };
15 big add(big a, big b);
16 big sub(big a, big b);
17 big sub(big a, big b, int d);//偏移，a-b*16^4
18 int compare(big a, big b,int d);//
19 int compare(big a, big b);
20 big mul(big a, big b);
21 big mod(big a, big b);
22 big pow(big a, big b,big n);//a^b mod n
23 big getinv(big a, big b);
24 big div(big a, big b,big &c);//返回商，r是余数

```

- 加法减法和乘法类似，这里只详细解释较为复杂的乘法

```

1  big mul(big a, big b)
2  {
3      big c;
4      for (int i = 0; i < 256; i++)
5      {
6          for (int j = 0; j < 256; j++)
7          {
8              c.num[i + j] += a.num[i] * b.num[j];//每个元素都是int类型，
              a.num[i]和b.num[j]最大为15，不会溢出
9          }
10     }
11     for (int i = 0; i < 510; i++)
12     {
13         c.num[i + 1] += c.num[i] / 16;//进位运算
14         c.num[i] %= 16;
15     }
16     return c;
17 }

```

- 取模和除法类似，这里解释除法

```

1  big div(big a, big b, big& c)
2  {
3      big q;
4      int t = compare(a, b);
5      if (t == -1)return q;
6      if (t == 0)
7      {
8          big c;

```



```

9         c.set(1);
10        return c;//0
11    }
12    int x, y;//x,y是a、b的最高位
13    for (x = 511; x >= 0; x--)
14        if (a.num[x])
15            break;
16    for (y = 511; y >= 0; y--)
17        if (b.num[y])
18            break;
19    int d = x - y;
20
21    c.copy(a);//c=a
22    while (d >= 0)//c的位数大于等于b
23    {
24        while (c.num[x] > b.num[y])//如果c的最高位大于b的最高位，则c一定大于b
25        {
26            c = sub(c, b, d);//c-=b*16^d
27            big v1;
28            v1.num[d]=1;
29            q=add(q,v1);//商在对应为上加1
30        }
31        if (c.num[x] == b.num[y] && compare(c, b, d) == 1)//处理最高位相
等情况
32        {
33            c = sub(c, b, d);
34            big v1;
35            v1.num[d] = 1;
36            q = add(q, v1);
37        }
38        //此时c<b*16^d
39        d--;
40        int t;
41        while (compare(c, b) == 1 && c.num[x])//借高位余数
42        {
43            c = sub(c, b, d);
44            big v1;
45            v1.num[d] = 1;
46            q = add(q, v1);
47        }
48        if (c.num[x] == 0)x--;
49        t = compare(c, b);
50        if (t == -1)break;//c<b
51        if (t == 0)//相等，返回0
52        {
53            c = sub(c, b);
54            big v1;
55            v1.num[d] = 1;
56            q = add(q, v1);
57            return q;
58        }
59    }
60    return q;//商
61 }

```

- 快速幂，复杂度  $O(\lg N)$

由于数值过大，不能使用常规求幂方法

例如, 求  $0x123^{0x256} \bmod n$

$$0x256 = 2 \times 16^2 + 5 \times 16^1 + 6 \times 16^0$$

$$\text{即 } 0x123^{0x256} = 0x123^{2 \times 16^2 + 5 \times 16^1 + 6 \times 16^0} = 0x123^{2 \times 16^2} \times 0x123^{5 \times 16^1} \times 0x123^{6 \times 16^0}$$

所以我们可以先计算出  $0x123^{16^0}$ 、 $0x123^{16^1}$  和  $0x123^{16^2}$

而  $0x123^{16^{i+1}} = (0x123^{16^i})^{16}$ , 此处也应用快速幂计算, 即以此计算出2次幂、4次幂、8次幂、16次幂

```
1
2  big pow(big a, big b, big n)//a^b mod n
3  {
4      big buffer[512];
5      buffer[0].copy(a);
6      int x;
7      for (x = 511; x > 0; x--)
8          if (b.num[x])
9              break;
10     for (int i = 1; i <= x; i++)
11         { //双层快速幂算法求a**(b**i)
12             big tempbuf[5];
13             tempbuf[0].copy(buffer[i - 1]);
14             for (int j = 1; j < 5; j++)
15             {
16                 big temp = mul(tempbuf[j - 1], tempbuf[j - 1]);
17                 tempbuf[j] = mod(temp, n);
18             }
19             buffer[i].copy(tempbuf[4]);
20         }
21
22     big product;
23     product.num[0] = 1;
24
25     for (int i = 511; i >= 0; i--)
26         { //a**(k*(b**i))
27             big temp;
28             for(int j=0;j<b.num[i];j++)
29             {
30                 temp=mul(product, buffer[i]);
31                 product=mod(temp,n);
32             }
33         }
34
35     return product;
36 }
```

- 扩展欧几里得算法求逆元

```
1  big getinv(big a, big b) //a mod b
2  {
3      b = mod(b, a);
4      big q[100], t[100], r[100];
5      r[0].copy(a);
6      r[1].copy(b);
7      t[0].set(0);
8      t[1].set(1); //赋初值
```



```
9      int i;
10     big v1;
11     v1.set(1);
12     for(i=2;i<500;i++)
13     {
14         q[i - 1] = div(r[i - 2], r[i - 1], r[i]);
15         big temp = mul(q[i - 1], t[i - 1]);
16         while (compare(temp, t[i - 2]) == 1)//本次大整数没有实现负数
17             t[i - 2] = add(t[i - 2], r[0]);
18         t[i] = sub(t[i - 2], temp);
19         if (compare(r[i], v1) == 0)
20             break;
21     }
22     return t[i];
23 }
```