

网络安全技术

实 验 报 告

学院：网安学院

年级：2020

班级：信安班

学号：2011395

姓名：魏伯繁

手机号：18611111192

2023 年 5 月 7 日

目录

1 实验目标	2
2 实验内容	2
2.1 公钥加密算法	2
2.1.1 公钥密码的特点	2
2.1.2 RSA 加密算法	3
2.2 DES 加密算法	4
2.3 S/C 通信流程	4
3 实验步骤	5
3.1 RSA 算法	5
3.2 交互模块	15
4 实验遇到的问题及其解决方法	20
4.1 求幂运算过慢	20
4.2 密钥管理	22
5 实验结论	24

1 实验目标

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。

2 实验内容

本次实验具体包括三个部分：第一个部分是 RSA 算法的实现，第二个是 DES 的实现，第三个则是双方通信模型的实现，其中 DES 加密的实现以及 C/S 通信模型已经在上一次实验中实现过，所以本次实验报告将首先对 RSA 加密算法进行介绍。

2.1 公钥加密算法

传统对称密码体制要求通信双方使用相同的密钥，因此应用系统的安全性完全依赖于密钥的保密。针对对称密码体系的缺陷，Diffe 和 Hellman 提出了新的密码体系—公钥密码体系，也称为非对称密码体系。在公钥加密系统中，加密和解密使用两把不同的密钥。加密的密钥（公钥）可以向公众公开，但是解密的密钥（私钥）必须是保密的，只有解密方知道。公钥密码体系要求算法要能够保证：任何企图获取私钥的人都无法从公钥中推算出来。公钥密码体制中最著名算法是 RSA，以及背包密码、McEliece 密码、Diffe_Hellman、Rabin、零知识证明、椭圆曲线、ElGamal 算法等。

2.1.1 公钥密码的特点

公钥密码体制如下部分组成：

1. 明文：作为算法的输入的消息或者数据。

2. 加密算法：加密算法对明文进行各种代换和变换。
3. 密文：作为算法的输出，看起来完全随机而杂乱的数据，依赖明文和密钥。对于给定的消息，不同的密钥将产生不同的密文，密文是随机的数据流，并且其意义是无法理解的。
4. 公钥和私钥：公钥和私钥成对出现，一个用来加密，另一个用来解密。
5. 解密算法：该算法用来接收密文，解密还原出明文。

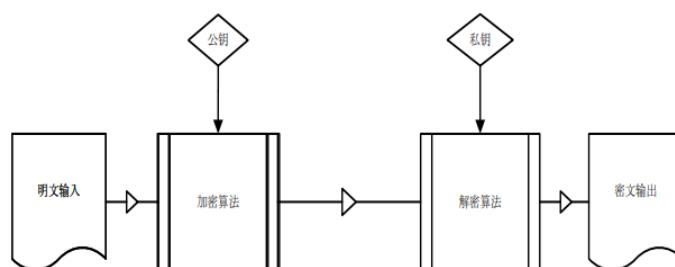


图 2.1: 公钥密码流程示意图

2.1.2 RSA 加密算法

RSA 加密算法是一种典型的公钥加密算法。RSA 算法的可靠性建立在分解大整数的困难性上。假如找到一种快速分解大整数算法的话，那么用 RSA 算法的安全性会极度下降。但是存在此类算法的可能性很小。目前只有使用短密钥进行加密的 RSA 加密结果才可能被穷举解破。只要其密钥的长度足够长，用 RSA 加密的信息的安全性就可以保证。RSA 密码体系使用了乘方运算。明文以分组为单位进行加密，每个分组的二进制值均小于 n ，也就是说分组的大小必须小于或者等于 $\text{Log}2N$

对于明文分组 M 和密文分组 C ，加密和解密的过程如下。

$$(1) \quad C = M^e \% n$$

$$(2) \quad M = C^d \% n = (M^e)^d \% n = M^{d \times e \% n} = M$$

其中 n 、 d 、 e 为三个整数，且 $d \times e \equiv 1 \% \phi(n)$ 。收发双方共享 n ，接受一方已知 d ，发送一方已知 e ，则此算法的公钥为 $\{e, n\}$ ，私钥是 $\{d, n\}$ 。

理解 RSA 基本工作原理需要数论的基础知识：

(1) 同余：两个整数 a ， b ，若它们除以整数 m 所得的余数相等，则称 a ， b 对于模 m 同余，记作 $a \equiv b \% m$ 。

(2) Euler 函数： $\phi(n)$ 是指所有小于 n 的正整数里，和 n 互质的整数的个数。其中 n 是一个正整数。假设整数 n 可以按照质因数分解写成如下形式： $n = P_1^{a_1} \times P_2^{a_2} \times \dots \times P_m^{a_m}$ ；其中， P_1, P_2, \dots, P_m 为质数。则 $\phi(n) = n \times \left(1 - \frac{1}{P_1}\right) \times \left(1 - \frac{1}{P_2}\right) \times \dots \times \left(1 - \frac{1}{P_m}\right)$ 。

图 2.2: 公钥密码流程示意图

RSA 密码体系公钥私钥生成方式如下。任意选取两个质数， p 和 q ，然后，设 $n = p \times q$ ；函数 $\phi(n)$ 为 Euler 函数，返回小于 n 且与 n 互质的正整数个数；选择一个任意正整数 e ，使其与 $\phi(n)$ 互质且小于 $\phi(n)$ ，公钥 e, n 已经确定；最后确定 d ，使得 $d \times e \equiv 1 \% \phi(n)$ ，即 $(d \times e - 1) \% \phi(n) = 0$ ，至此，私钥 d, n 也被确定。

2.2 DES 加密算法

由于上次已经实现了 DES 加密算法，所以不再对 DES 加密算法做特别详细的介绍，只对齐做简要的介绍，DES(Data Encryption Standard) 是目前最为流行的加密算法之一。DES 是对称的，也就是说它使用同一个密钥来加密和解密数据。

DES 还是一种分组加密算法，该算法每次处理固定长度的数据段，称之为分组。DES 分组的大小是 64 位，如果加密的数据长度不是 64 位的倍数，可以按照某种具体的规则来填充位。

从本质上来说，DES 的安全性依赖于虚假表象，从密码学的术语来讲就是依赖于“混乱和扩散”的原则。混乱的目的是为隐藏任何明文同密文、或者密钥之间的关系，而扩散的目的是使明文中的有效位和密钥一起组成尽可能多的密文。两者结合到一起就使得安全性变得相对较高。

DES 算法具体通过对明文进行一系列的排列和替换操作来将其加密。过程的关键就是从给定的初始密钥中得到 16 个子密钥的函数。要加密一组明文，每个子密钥按照顺序 (1-16) 以一系列的位操作施加于数据上，每个子密钥一次，一共重复 16 次。每一次迭代称之为轮。要对密文进行解密可以采用同样的步骤，只是子密钥是按照逆向的顺序 (16-1) 对密文进行处理。

2.3 S/C 通信流程

基于 TCP/IP 的服务-客户端通信模型在计网课程以及上次网络安全技术实验中都已经进行了实验，所以在这里不再做过多赘述，在这里简要复述其工作流程：服务器端和客户端的工作过程为：

1. 服务器首先启动监听程序，对指定的端口进行监听，等待接收客户端的连接请求。
2. 客户端启动程序，请求连接服务器的指定端口。
3. 服务器收到客户端的连接请求后，与客户端建立套接字连接。
4. 连接建立成功，客户端与服务器分别打开两个流，其中客户端的输入流连接到服务器的输出流，服务器的输入流连接到客户端的输出流，两边的流连接成功后进行双向通信。
5. 当通信完毕后，客户端和服务器两边各自断开连接。

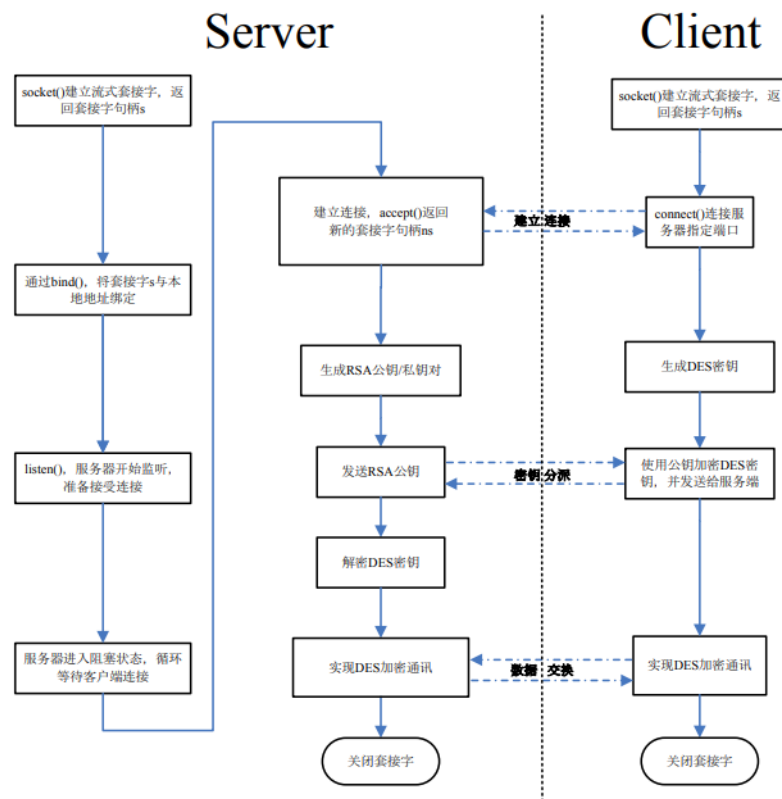


图 2.3: client-server 模型示意图

3 实验步骤

在实验步骤这个模块中我将着重介绍 RSA 算法的实现和交互模型的实现, 因为 DES 在上次实验中已经介绍的很详细了, 所以在此处不再做过多介绍。

3.1 RSA 算法

首先为应对 RSA 可能出现的一系列关于大整数的问题, 我们应该自定义大整数作为其中运算的中间代码:

```

1  class BigInt {
2      public:
3          //类内属性
4          vector<int> num; //用来存储真正的数, 在这里我们默认 0 是最高位
5          bool minus = false; //增加负数表示
6          //重载运算符
7          friend BigInt operator+(const BigInt& a, const BigInt& b);

```

```

8      friend BigInt operator-(const BigInt& a, const BigInt& b);
9      friend BigInt operator*(const BigInt& a, const BigInt& b);
10     friend BigInt operator/(const BigInt& a, const BigInt& b);
11     friend BigInt operator%(const BigInt& a, const BigInt& b);
12     //这些比较目前只比较 num, 不比较正负位
13     friend bool operator==(const BigInt& a, const BigInt& b);
14     friend bool operator!=(const BigInt& a, const BigInt& b) { return !(a == b); }
15     friend bool operator>=(const BigInt& a, const BigInt& b);
16     friend bool operator<(const BigInt& a, const BigInt& b) {
17         bool ret = a >= b;
18         return !ret;
19     }
20     //其他功能性函数
21     BigInt();
22     BigInt(int size, bool minus = false);
23     BigInt(int size, vector<int>& v, bool minus = false);
24     BigInt(const BigInt& bi);
25     bool isLegal(); //判断 vector 中的数是否合法
26     bool strictLegal(); //判断长度、数字是否合法
27     void generateNum(int size); //随机生成一个数
28     void generateOdd(int size);
29     void randGenerate(BigInt& n);
30     void equal(BigInt& b);
31 };

```

在 BigInt 类中其实比较困哪的就是乘除法和取模运算，加减法、比较大小都是思路比较简单的，下面的代码就实现了基于 BigInt 的乘除法以及模运算。

```

1  BigInt operator*(const BigInt& a, const BigInt& b) {
2      BigInt Myb(b);
3      reverse(Myb.num.begin(), Myb.num.end());
4      BigInt ret; //默认初始化为 0
5      for (int i = 0; i < Myb.num.size(); i++) {
6          if (Myb.num[i] == 1) { //为 1 才有意义
7              //添加零

```

```
8     vector<int>v = AddZero(i, a.num);
9     //拿着向量新初始化一个 BigInt
10    BigInt b(v.size(), v);
11    //加到 ret 上
12    ret = ret + b;
13    }
14    }
15    if ((a.minus && b.minus) || (!a.minus && !b.minus)) { ret.minus = false; }
16    else { ret.minus = true; }
17    return ret;
18 }
19
20 BigInt operator/(const BigInt& a, const BigInt& b) {
21     if (a < b) {
22         //cout << " 非法的除法，被除数小于除数" << endl;
23         BigInt bi;
24         return bi;
25     }
26     BigInt Mya(a);
27     vector<int>vi; //存储结果的地方
28     for (int i = 0; i < a.num.size(); i++) {
29         BigInt myb = b;
30         for (int j = 1; j <= ((a.num.size() - 1) - i); j++) {
31             myb.num.push_back(0);
32         }
33         //cout << i << " ";
34         //cout << Mya << " " << myb << endl;
35         if (Mya >= myb) {
36             vi.push_back(1);
37             Mya = Mya - myb;
38             continue;
39         }
40         if (Mya < myb) {
41             vi.push_back(0);
42             continue;
```



```

43     }
44     //cout << Mya << " " << myb << " " << vi[i]<<endl;
45 }
46 //cout << endl;
47 //去除前导零
48 reverse(vi.begin(), vi.end());
49 for (int i = vi.size() - 1; i >= 0; i--) {
50     if (vi[vi.size() - 1] == 0) { vi.pop_back(); }
51     else { break; }
52 }
53 reverse(vi.begin(), vi.end());
54 BigInt ret(vi.size(), vi);
55 if ((a.minus && b.minus) || (!a.minus && !b.minus)) { ret.minus = false; }
56 else { ret.minus = true; }
57 return ret;
58 }
59
60 BigInt operator%(const BigInt& a, const BigInt& b) {
61     if (a < b) {
62         return a;
63     }
64     BigInt Mya(a);
65     vector<int>vi; //存储结果的地方
66     for (int i = 0; i < a.num.size(); i++) {
67         BigInt myb = b;
68         for (int j = 1; j <= ((a.num.size() - 1) - i); j++) {
69             myb.num.push_back(0);
70         }
71         //cout << i << " ";
72         //cout << Mya<<" "<< myb << endl;
73         if (Mya >= myb) {
74             vi.push_back(1);
75             Mya = Mya - myb;
76             continue;
77         }

```

```

78     if (Mya < myb) {
79         vi.push_back(0);
80         continue;
81     }
82     //cout << Mya << " " << myb << " " << vi[i]<<endl;
83 }
84 return Mya;
85 }
86

```

根据 RSA 加密算法的要素，我们不难看出，其中还需要很多在信安数基础以及密码学学到的算法，然而这些算法还需要应用到 BigInt 中，所以我们还需要针对 BigInt 编写代码实现这些数学公式的计算，下面的代码就实现了扩展的欧几里得算法：

```

1  BigInt extendEuclid(BigInt a, BigInt b, BigInt& x, BigInt& y) {
2      vector<int>vi1; vi1.push_back(1);
3      BigInt ONE(1, vi1);
4      vector<int>vi0; vi0.push_back(0);
5      BigInt ZERO(1, vi0);
6      if (b.num.size() == 0) {
7          x = ONE;
8          y = ZERO;
9          return a;
10     }
11     BigInt r = extendEuclid(b, a % b, x, y);
12     BigInt temp = x;
13     x = y;
14     y = (temp - (a / b) * y);
15     return r;
16 }

```

我们在进行 RSA 算法时可能还会随机生成素数，与此同时伴随的就是 gcd 的计算，接下来的代码就生成了一个随机的奇数，但是并不保证是一个随机素数，由于检验一个整数是否是素数会用到 gcd 算法，所以下面的代码也实现了关于 BigInt 的最大公约数的计算。

```
1  BigInt generatePrime(int i) {
2  BigInt b(i);
3  b.generateOdd(i);
4  vector<int>vi;
5  vi.push_back(1); vi.push_back(0);
6  BigInt TWO(2, vi);
7  while (true) {
8      if (Miller_Rabin(b, 5)) { return b; }
9      else {
10         b = b + TWO;
11     }
12 }
13 }
14 BigInt generategcd_1(BigInt& bi) {
15 BigInt b(bi.num.size() / 2);
16 b.generateOdd(bi.num.size() / 2);
17 vector<int>vi2; vi2.push_back(1);
18 BigInt ONE(1, vi2);
19 while (true) {
20     if (b >= bi) {
21         cout << " 计算互素值失败" << endl;
22         return ONE;
23     }
24     if (Euclid(b, bi) == ONE) {
25         return b;
26     }
27     else {
28         b = b + ONE;
29     }
30 }
31 }
```

在 RSA 算法中一个常用的算法就是米勒拉宾算法，我们还需要针对 BigInt 来编写米勒拉宾算法，具体的算法如下面的代码所示：米勒兰斌算法会循环五次，这样可以获得一个置信度相当高的结果，对

于一般场景下是足够使用的。通过使用费马小定理做二次探测，可以在一个相对较快的时间内判断一个奇数是否为质数。

```

1
2  bool Miller_Rabin(BigInt b, int k) {
3      if (b.num[b.num.size() - 1] == 0) { return false; }
4      int s = 0; //这是 2 的整数次幂的表示
5      BigInt myb(b);
6      BigInt t; //另外一部分
7      BigInt r; //每一轮迭代都要用到的
8      //构造一个代表 2 的 BigInt 数
9      vector<int> vi; vi.push_back(1); vi.push_back(0);
10     BigInt TWO(2, vi);
11     vector<int> vi2; vi2.push_back(1);
12     BigInt ONE(1, vi2);
13     myb = myb - ONE;
14     //首先需要确定 s 和 t
15     while (true) {
16         BigInt bi;
17         bi = myb % TWO;
18         if (bi.num.size() == 0) {
19             s++;
20             myb = myb / TWO;
21             continue;
22         }
23         else {
24             t = myb;
25             break;
26         }
27     }
28     //cout << "s = " << s << " " << "t = " << t << endl;
29     for (int j = 1; j <= k; j++) {
30         BigInt bb;
31         Sleep(1000);
32         bb.randGenerate(b);

```

```

33     //cout << "bb=" << bb << endl;
34     for (int i = 0; i < s; i++) {
35         if (i == 0) {
36             r = MyquickPow(bb, t, b);
37             if (r == ONE) { break; }
38             BigInt check = b - ONE;
39             if (r == check) { break; }
40             if (s == 1) { cout << "fail to pass Miller_Rabin" << endl; return false; }
41             continue;
42         }
43         else {
44             r = MyquickPow(r, TWO, b);
45             BigInt check = b - ONE;
46             if (r == check) { break; }
47             if (i == s - 1) { cout << "fail to pass Miller_Rabin" << endl; return false; }
48             continue;
49         }
50     }
51 }
52 cout << "pass Miller_Rabin successfully" << endl;
53 return true;
54 }

```

由于随机选取素数的过程确实需要一定时间, 这个时间根据宿主机性能不同也会有较大的浮动。所以我在代码中也预设了一个 512/256 位的密钥, 可以由用户选择使用。

```

1  vector<BigInt>Prime512;
2  void InitialPrime512() {
3      string s1 = "B49700D49696726058F2506BAF6860981E4F097B89CF5540FB07A3CDEB9BDCC3";
4      string s2 = "DE9AE262BBFA557E9EF55E9D165B43E6F859803F4AC929FAE9680325BA295513";
5      string s3 = "A14A46D62943712797385A7259AD68FAE4110C83DBFD75AFAED7AC3B52F4308B";
6      string s4 = "A7BBD5BD2F177E7BAEB89DB47196F7640C8ED51F62ED6961BF5726877FF06A53";
7      string s5 = "B87E37FE315B0F4CC722BB0F815A184EA27FAB85119BFDD8A45DD98643926D1B";
8      string s6 = "8233A8BA28A6F782A7A70AFD7AE05808AE0F812515DBD75CEE2EB0967E7FA063";
9      Prime512.push_back(string2BigInt(s1));

```

```

10     Prime512.push_back(string2BigInt(s2));
11     Prime512.push_back(string2BigInt(s3));
12     Prime512.push_back(string2BigInt(s4));
13     Prime512.push_back(string2BigInt(s5));
14     Prime512.push_back(string2BigInt(s6));
15 }
16 vector<BigInt>Prime256;
17 void InitPrime256() {
18     string s1 = "DB92BF308AF4895A7EF3E3FCD7290853";
19     string s2 = "F4D7A6FF616582E7A18FDA48A5B6A303";
20     string s3 = "AD49678F3DBA34AA8789350E9FA3FE5B";
21     string s4 = "D711D47AEF9F9471FF9C7DFCD1DCC18B";
22     string s5 = "C502F81D0180027EE578EC5AF5C67EE3";
23     string s6 = "C2C899D5EA7E21517C42C9AB5104BA83";
24     Prime256.push_back(string2BigInt(s1));
25     Prime256.push_back(string2BigInt(s2));
26     Prime256.push_back(string2BigInt(s3));
27     Prime256.push_back(string2BigInt(s4));
28     Prime256.push_back(string2BigInt(s5));
29     Prime256.push_back(string2BigInt(s6));
30 }

```

上面的内容其实就是绝大部分 RSA 算法的核心内容，但是为了完成整个算法，还需要一些辅助函数，比如十进制转二进制、二进制转 16 进制等等，由于辅助代码很多，就不一一列举，在下面列举了一些我认为十分重要的辅助函数，他们在整个项目中屡次出现。其中包括随机生成一个 BigInt、已经将一个字符串类型转换为 BigInt 类型，这个功能是很重要的，因为我们经常会给定一些常数，这些常数没办法直接参与运算，他们参与运算的唯一方法就是变成 BigInt 类型，而 string 类型是我们给出一个常数最简单的方式，所以这个函数非常的重要

```

1 BigInt string2BigInt2(string s) {
2     vector<int>vi;
3     for (int i = 0; i < s.size(); i++) {
4         if (s[i] == '0') { vi.push_back(0); }
5         else { vi.push_back(1); }
6     }

```

```
7     BigInt bi(vi.size(), vi);
8     return bi;
9 }
10 string Dec2Bin(int i) {
11     vector<int>vi;
12     unsigned long long int myi = i;
13     while (myi != 0) {
14         vi.push_back(myi % 2);
15         myi = myi / 2;
16     }
17     while (vi.size() < 8) {
18         vi.push_back(0);
19     }
20     reverse(vi.begin(), vi.end());
21     string s;
22     for (int i = 0; i < vi.size(); i++) {
23         if (vi[i] == 0) { s.append(string(1, '0')); }
24         else { s.append(string(1, '1')); }
25     }
26
27     return s;
28 }
29 void BigInt::randGenerate(BigInt& n) {
30     //范围是 2 到 n-2
31     vector<int>vi; vi.push_back(1); vi.push_back(0);
32     BigInt TWO(2, vi);
33     BigInt end = n - TWO;
34     //size 不会比 n 的 size 大
35     int size;
36     while (true) {
37         size = rand() % (n.num.size() + 1);
38         if (size > 0 && size != 1) { break; }
39     }
40     L:
41     BigInt ret(size);
```

```

42     ret.generateNum(size);
43     if (end < ret) { goto L; }
44     num.clear();
45     for (int i = 0; i < size; i++) {
46         num.push_back(ret.num[i]);
47     }
48 }
49

```

看到这里如果是对密码学比较熟悉的朋友可能就发现，缺少了快速模幂运算，这是 RSA 中必不可少的一部分，正确的，这个地方是我 debug 最多、破防最多的模块，于是我将他放在了实验中遇到的问题及解决办法模块进行介绍。

3.2 交互模块

其中交互框架如下面的代码所示，基本的代码仍然包括：所需的参数初始化、接收和发送线程的创建

```

1  void initailNeeded() {
2      WSASStartup(MAKEWORD(2, 2), &wsadata);
3      //指定 clintA 的性质
4      my_addr.sin_family = AF_INET;//使用 IPV4
5      my_addr.sin_port = htons(8888);//server 的端口号
6      my_addr.sin_addr.s_addr = htonl(2130706433);//主机 127.0.0.1
7      //指定 clientB 的性质
8      oppo_addr.sin_family = AF_INET;//使用 IPV4
9      oppo_addr.sin_port = htons(8887);//server 的端口号
10     oppo_addr.sin_addr.s_addr = htonl(2130706433);//主机 127.0.0.1
11
12     //绑定服务端
13     mySocket = socket(AF_INET, SOCK_DGRAM, 0);
14     bind(mySocket, (SOCKADDR*)&my_addr, sizeof(my_addr));
15 }
16
17 //就是一个框架而已
18 DWORD WINAPI myreceive(LPVOID p) {

```



```

19  char* Recvbuffer = new char[100000];
20  while (true) {
21      while (recvfrom(mySocket, Recvbuffer, 100000, 0, (sockaddr*)&oppo_addr, &olen) > 0) {
22          string s;
23          int i = 0;
24          while (Recvbuffer[i] != 0) {
25              s.append(string(1, Recvbuffer[i]));
26          }
27          cout << "[接受消息]" << s << endl;
28      }
29  }
30  }
31
32  //只是代表一个框架
33  DWORD WINAPI mysend(LPVOID p) {
34      char* sendbuffer = new char[100000];
35      while (true) {
36          cin.getline(sendbuffer, 100000);
37          sendto(mySocket, sendbuffer, 100000, 0, (sockaddr*)&oppo_addr, olen);
38      }
39  }
40

```

本次实验中国最重要的模块就是密钥的分配在密钥的分配模块中，通信双方需要根据事先已经约定好的 RSA 公钥对想要使用的 DES 密钥进行加密后发送，对方在接收到对方由己方公钥加密的 DES 密钥后再使用自己的私钥解密就可以得到正确的 DES 密钥。

```

1  int allocKey() {
2      cin >> check;
3      if (check != 1) { return -1; }
4      //发送身份识别
5      string t = N1 + IDA;
6      t = encrypt_B_public(t);
7      t = "[IDEN]" + t;
8      char* sendbuffer = new char[10000];

```

```
9      memcpy(sendbuffer, t.c_str(), t.size());
10      int ii = sendto(mySocket, sendbuffer, t.size(), 0, (sockaddr*)&oppo_addr, olen);
11      //接收反馈
12      char* recvbuffer = new char[10000];
13      memset(recvbuffer, 0, 10000);
14      recvfrom(mySocket, recvbuffer, 10000, 0, (sockaddr*)&oppo_addr, &olen);
15      string s = recvbuffer;
16      s = cleanString(s);
17      if (s.substr(0, 6) == "[IDEN]") {
18          s = s.substr(6, 10000);
19      }
20      outTime();
21      s = decrypt_A_private(s);
22      if (s == N1 + N2) {
23      }
24      else {
25          cout << "[TIPS] 身份认证失败" << endl;
26          return -1;
27      }
28      cout << endl;
29      //再次准备发送
30      t.clear();
31      t = encrypt_B_public(N2);
32      t = "[IDEN]" + t;
33      memset(sendbuffer, 0, 10000);
34      memcpy(sendbuffer, t.c_str(), t.size());
35      sendto(mySocket, sendbuffer, 10000, 0, (sockaddr*)&oppo_addr, olen);
36
37      Sleep(30 * 1000);
38      //准备发送 AES 密钥的第一部分
39      string key1 = AESKey.substr(0, 16);
40      key1 = encrypt_A_private(key1);
41      int i = 0; vector<string>vs;
42      while (i < key1.size()) {
43          string temp = key1.substr(i, 16); i += 16;
```

```
44         vs.push_back(temp);
45     }
46     for (int i = 0; i < vs.size(); i++) {
47         vs[i] = encrypt_B_public(vs[i]);
48         memset(sendbuffer, 0, 10000);
49         vs[i] = "[key1]" + vs[i];
50         memcpy(sendbuffer, vs[i].c_str(), vs[i].size());
51         sendto(mySocket, sendbuffer, 10000, 0, (sockaddr*)&oppo_addr, olen);
52     }
53
54     //准备发送 AES 密钥的第二部分
55     string key2 = AESKey.substr(16, 16);
56     key2 = encrypt_A_private(key2);
57     i = 0; vs.clear();
58     while (i < key2.size()) {
59         string temp = key2.substr(i, 16); i += 16;
60         vs.push_back(temp);
61     }
62     for (int i = 0; i < vs.size(); i++) {
63         vs[i] = encrypt_B_public(vs[i]);
64         memset(sendbuffer, 0, 10000);
65         vs[i] = "[key2]" + vs[i];
66         memcpy(sendbuffer, vs[i].c_str(), vs[i].size());
67         sendto(mySocket, sendbuffer, 10000, 0, (sockaddr*)&oppo_addr, olen);
68     }
69
70     string last = "[OVER]";
71     memset(sendbuffer, 0, 10000);
72     memcpy(sendbuffer, last.c_str(), last.size());
73     sendto(mySocket, sendbuffer, 10000, 0, (sockaddr*)&oppo_addr, olen);
74     outTime();
75     cout << endl;
76 }
```

在这个过程中需要使用到的辅助函数包括能够将字符串类型的实现约定好的对方的公钥以及自己

的私钥编程 RSA 算法能够识别的类别的算法，具体的算法如下：

```
1  string Dec2Hex(int n) {
2      string s;
3      string t = "0123456789ABCDEF";
4      int x;
5      while (n != 0) {
6          x = n % 16;
7          // 将 n % 16 转换为字符逆序存入 s
8          s = t[x] + s;
9          n = n / 16;
10     }
11
12     if (s == "") {
13         cout << " 错误的十进制转 16 进制输入" << endl;
14         return s;
15     }
16     else {
17         if (s.size() == 1) {
18             s.push_back('0');
19             reverse(s.begin(), s.end());
20         }
21         return s;
22     }
23 }
24 string Text2Hex(string s) {
25     string ret;
26     for (int i = 0; i < s.size(); i++) {
27         int t = s[i];
28         string ss = Dec2Hex(t);
29         ret.append(ss);
30     }
31     return ret;
32 }
```

4 实验遇到的问题及其解决方法

4.1 求幂运算过慢

其实这应该是编写 RSA 算法必然会遇到的一个问题，就是我们的求幂运算如果按照常规算法来做的话会做到天荒地老，对于一个 512bit 的 RSA 算法来说，如果不使用快速模幂运算，可能要计算个几年都不一定会有结果，于是我们需要使用快速模幂运算来超大幅度的加速这个计算过程。具体的快速模幂算法的实现如下：其中最上面的算法是常规的模式算法，他会直接调用之前我们在介绍 BigInt 时编写的模算法，这种算法的效率显然是很低的。

```
1  BigInt pow(BigInt& a, BigInt& b, BigInt& m) {
2      vector<int>vi;
3      vi.push_back(1);
4      BigInt ONE(1, vi);
5      BigInt Myb(b);
6      BigInt Mya(a);
7      Myb = Myb - ONE;
8      if (Myb.num.size() == 0) { Mya = Mya % m; return Mya; }
9      while (true) {
10         if (Myb.num.size() != 0) {
11             Mya = Mya * a;
12             Mya = Mya % m;
13             Myb = Myb - ONE;
14         }
15         else {
16             break;
17         }
18     }
19     return Mya;
20 }
21 //用来存储已经计算好的阶
22 //一定要注意，在外部调用 quickPow 之前一定要清空 store 一次
23 BigInt quickPow(BigInt& a, BigInt& b, BigInt& m) {
24     //cout << "b=" << b << endl;
25     //他其实需要做的就只是调度
26     vector<int>vi2; vi2.push_back(1); vi2.push_back(0);
```

```
27     BigInt TWO(2, vi2);
28     vector<int>vi1; vi1.push_back(1);
29     BigInt ONE(1, vi1);
30     //如果是 2 次方直接计算
31     if (b == TWO) {
32         BigInt ret;
33         ret = pow(a, b, m);
34         return ret;
35     }
36     //一次方直接返回
37     if (b == ONE) {
38         return a;
39     }
40     //偶数次方
41     if (b.num[b.num.size() - 1] == 0) {
42         BigInt ret;
43         BigInt t = b / TWO;
44         BigInt Mya = (a * a) % m;
45         ret = quickPow(Mya, t, m); //转而计算  $a^2$  的  $b/2$  次方
46         return ret;
47     }
48     //奇数次方
49     else {
50         BigInt ret;
51         BigInt t = b / TWO;
52         BigInt Mya = (a * a) % m;
53         ret = quickPow(Mya, t, m); //转而计算  $a^2$  的  $b/2$  次方
54         ret = (ret * a) % m; //因为是奇数所以还要再乘一个  $a$ 
55         return ret;
56     }
57 }
58 }
59 BigInt MyquickPow(BigInt& a, BigInt& b, BigInt& m) {
60     return quickPow(a, b, m);
61 }
```

4.2 密钥管理

其实这个也没什么好办法，要么把密钥写死的程序里要么让程序去固定的位置去找，我使用的是两者结合，用户可以选择是使用在程序中已经写死的 RSA512 位或者 256 位密钥亦或是自己去指定的路径下寻找，不管怎么样都需要将字符串类型的数字转成 bigint 类型的数据，这个过程还是很麻烦的。

```
1  BigInt string2BigInt(string s) {
2      vector<int>vi;
3      for (int i = 0; i < s.size(); i++) {
4          if (i == 0) {
5              if (s[i] == '0') { continue; }
6              if (s[i] == '1') { vi.push_back(1); continue; }
7              if (s[i] == '2') { vi.push_back(1); vi.push_back(0); continue; }
8              if (s[i] == '3') { vi.push_back(1); vi.push_back(1); continue; }
9              if (s[i] == '4') { vi.push_back(1); vi.push_back(0); vi.push_back(0); continue; }
10             if (s[i] == '5') { vi.push_back(1); vi.push_back(0); vi.push_back(1); continue; }
11             if (s[i] == '6') { vi.push_back(1); vi.push_back(1); vi.push_back(0); continue; }
12             if (s[i] == '7') { vi.push_back(1); vi.push_back(1); vi.push_back(1); continue; }
13             if (s[i] == '8') { vi.push_back(1); vi.push_back(0); vi.push_back(0);
14                 vi.push_back(0); continue; }
15             if (s[i] == '9') { vi.push_back(1); vi.push_back(0); vi.push_back(0);
16                 vi.push_back(1); continue; }
17             if (s[i] == 'A' || s[i] == 'a') { vi.push_back(1); vi.push_back(0); vi.push_back(1);
18                 vi.push_back(0); continue; }
19             if (s[i] == 'B' || s[i] == 'b') { vi.push_back(1); vi.push_back(0); vi.push_back(1);
20                 vi.push_back(1); continue; }
21             if (s[i] == 'C' || s[i] == 'c') { vi.push_back(1); vi.push_back(1); vi.push_back(0);
22                 vi.push_back(0); continue; }
23             if (s[i] == 'D' || s[i] == 'd') { vi.push_back(1); vi.push_back(1); vi.push_back(0);
24                 vi.push_back(1); continue; }
25             if (s[i] == 'E' || s[i] == 'e') { vi.push_back(1); vi.push_back(1); vi.push_back(1);
26                 vi.push_back(0); continue; }
27             if (s[i] == 'F' || s[i] == 'f') { vi.push_back(1); vi.push_back(1); vi.push_back(1);
```

```
28     vi.push_back(1); continue; }
29 }
30 if (s[i] == '0') { vi.push_back(0); vi.push_back(0); vi.push_back(0); vi.push_back(0);
31 continue; }
32 if (s[i] == '1') { vi.push_back(0); vi.push_back(0); vi.push_back(0); vi.push_back(1);
33 continue; }
34 if (s[i] == '2') { vi.push_back(0); vi.push_back(0); vi.push_back(1); vi.push_back(0);
35 continue; }
36 if (s[i] == '3') { vi.push_back(0); vi.push_back(0); vi.push_back(1); vi.push_back(1);
37 continue; }
38 if (s[i] == '4') { vi.push_back(0); vi.push_back(1); vi.push_back(0); vi.push_back(0);
39 continue; }
40 if (s[i] == '5') { vi.push_back(0); vi.push_back(1); vi.push_back(0); vi.push_back(1);
41 continue; }
42 if (s[i] == '6') { vi.push_back(0); vi.push_back(1); vi.push_back(1); vi.push_back(0);
43 continue; }
44 if (s[i] == '7') { vi.push_back(0); vi.push_back(1); vi.push_back(1); vi.push_back(1);
45 continue; }
46 if (s[i] == '8') { vi.push_back(1); vi.push_back(0); vi.push_back(0); vi.push_back(0);
47 continue; }
48 if (s[i] == '9') { vi.push_back(1); vi.push_back(0); vi.push_back(0); vi.push_back(1);
49 continue; }
50 if (s[i] == 'A' || s[i] == 'a') { vi.push_back(1); vi.push_back(0); vi.push_back(1);
51 vi.push_back(0); continue; }
52 if (s[i] == 'B' || s[i] == 'b') { vi.push_back(1); vi.push_back(0); vi.push_back(1);
53 vi.push_back(1); continue; }
54 if (s[i] == 'C' || s[i] == 'c') { vi.push_back(1); vi.push_back(1); vi.push_back(0);
55 vi.push_back(0); continue; }
56 if (s[i] == 'D' || s[i] == 'd') { vi.push_back(1); vi.push_back(1); vi.push_back(0);
57 vi.push_back(1); continue; }
58 if (s[i] == 'E' || s[i] == 'e') { vi.push_back(1); vi.push_back(1); vi.push_back(1);
59 vi.push_back(0); continue; }
60 if (s[i] == 'F' || s[i] == 'f') { vi.push_back(1); vi.push_back(1); vi.push_back(1);
61 vi.push_back(1); continue; }
62 }
```



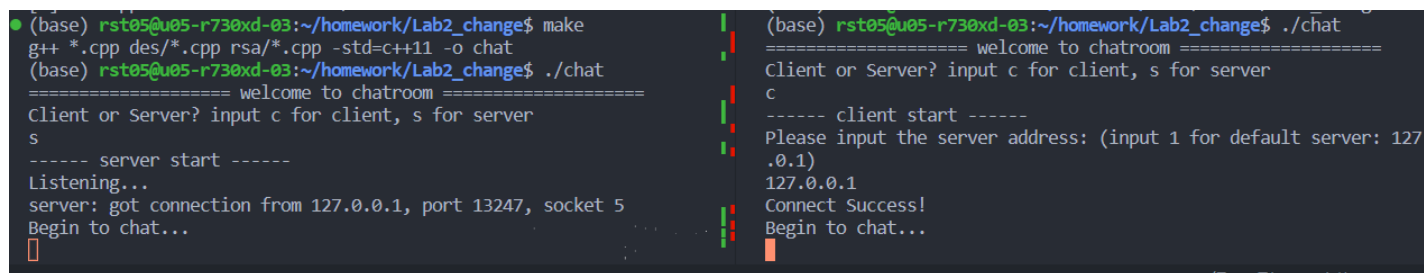
```

63  BigInt b(vi.size(), vi);
64  return b;
65  }
66

```

5 实验结论

首先我们在 linux 上运行对应的代码查看结果，首先是选择本次运行的类型是服务端还是客户端。这个选择和上一个实验中的基于 DES 加密的算法是非常像的



```

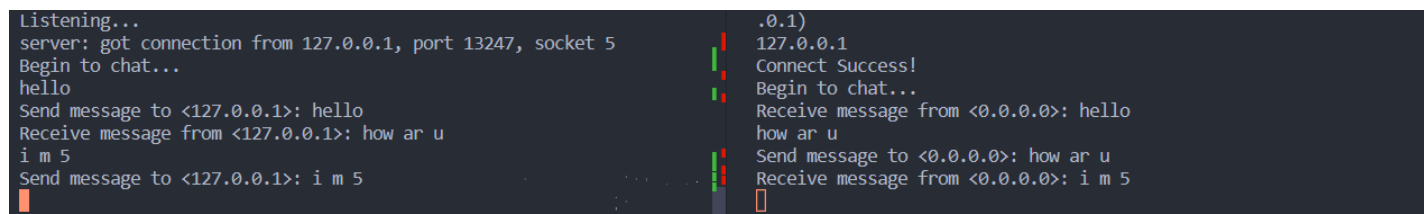
(base) rst05@u05-r730xd-03:~/homework/Lab2_change$ make
g++ *.cpp des/*.cpp rsa/*.cpp -std=c++11 -o chat
(base) rst05@u05-r730xd-03:~/homework/Lab2_change$ ./chat
===== welcome to chatroom =====
Client or Server? input c for client, s for server
c
----- client start -----
Please input the server address: (input 1 for default server: 127.0.0.1)
127.0.0.1
Connect Success!
Begin to chat...

(base) rst05@u05-r730xd-03:~/homework/Lab2_change$ ./chat
===== welcome to chatroom =====
Client or Server? input c for client, s for server
s
----- server start -----
Listening...
server: got connection from 127.0.0.1, port 13247, socket 5
Begin to chat...

```

图 5.4: 进入聊天室

接下来我们常数输入英文和数字，查看能否正常传输：



```

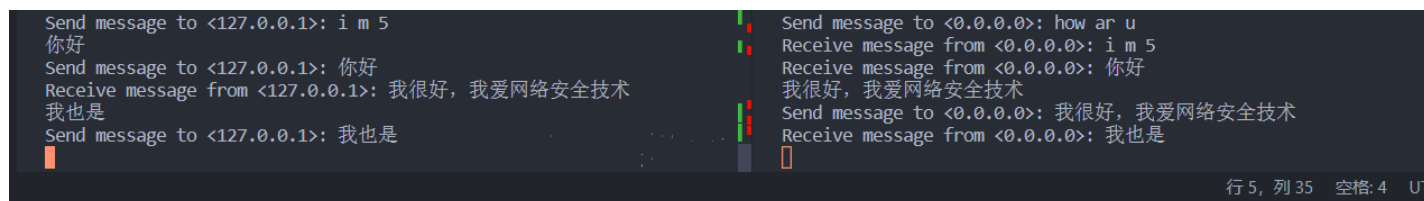
Listening...
server: got connection from 127.0.0.1, port 13247, socket 5
Begin to chat...
hello
Send message to <127.0.0.1>: hello
Receive message from <127.0.0.1>: how ar u
i m 5
Send message to <127.0.0.1>: i m 5

.0.1)
127.0.0.1
Connect Success!
Begin to chat...
Receive message from <0.0.0.0>: hello
how ar u
Send message to <0.0.0.0>: how ar u
Receive message from <0.0.0.0>: i m 5

```

图 5.5: 开始聊天

然后我们尝试使用中文交流，发现也可以正常传输数据。



```

Send message to <127.0.0.1>: i m 5
你好
Send message to <127.0.0.1>: 你好
Receive message from <127.0.0.1>: 我也很好，我爱网络安全技术
我也是
Send message to <127.0.0.1>: 我也是

Send message to <0.0.0.0>: how ar u
Receive message from <0.0.0.0>: i m 5
Receive message from <0.0.0.0>: 你好
我也很好，我爱网络安全技术
Send message to <0.0.0.0>: 我也很好，我爱网络安全技术
Receive message from <0.0.0.0>: 我也是

```

图 5.6: 开始聊天