网络安全技术

实
验
报
告

学　　院　网安学院

年　　级　21 级

班　　级　信安一班

学　　号　2113662

姓　　名　张丛

手机号　18086215842

2022 年 4 月 12 日

目录

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

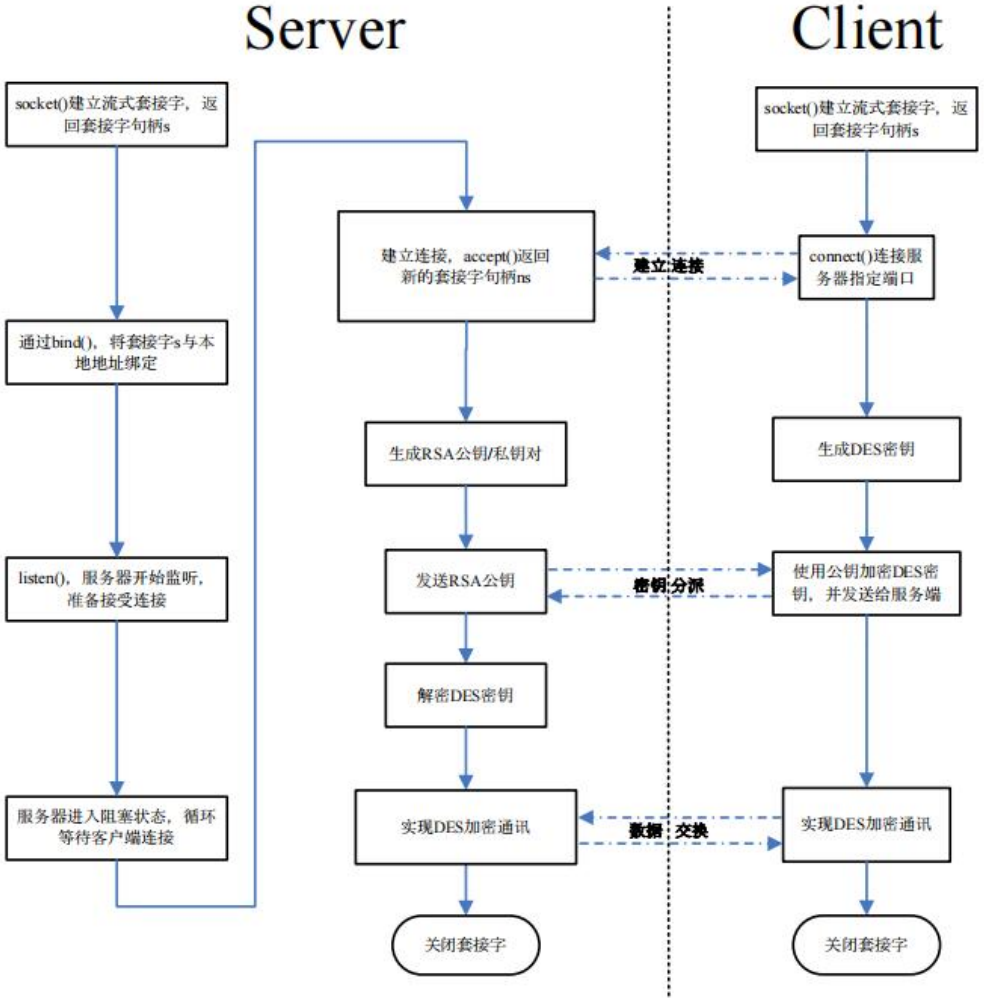
1. 要求在 Linux 操作系统中完成基于 RSA 算法的自动分配密钥加密聊天程序的编写。

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

实验二整体流程如下：



（一）DES

在实验一已经实现了 DES 的加解密，在实验二中对 DES 代码略有修改，对代码文件的层次结构进行了优化。

Cilent 端随机生成 DES 密钥代码如下：

```
void GerenateDesKey(char* x){
    int i;
    srand(time(NULL));
    for (i = 0; i < 8; i++)
    {
        switch ((rand() % 3))
        {
        case 1:
            x[i] = 'A' + rand() % 26;
            break;
        case 2:
            x[i] = 'a' + rand() % 26;
            break;
        default:
            x[i] = '0' + rand() % 10;
            break;
        }
    }
    x[i] = '\0';
}
```

（二）RAS

封装 RAS 的类：

```
class CRsaOperate {
public:
    RsaParam m_cParament;
    CRsaOperate();
    static inline __int64 MulMod(__int64 a, unsigned long b, unsigned long n);
    static __int64 PowMod(__int64 base, __int64 pow, __int64 n);
    static long RabinMillerKnl(__int64 &n);
    static long RabinMiller(__int64 &n, long loop);
    static __int64 RandPrime(char bit);
    static __int64 Gcd(__int64 &p, __int64 &q);
    static __int64 Euclid(__int64 e, __int64 t_n);
    static __int64 Encry(unsigned short nScore, PublicKey &cKey);
    unsigned short Decry(__int64 nScore);
    PublicKey GetPublicKey();
};
```

各个函数的功能以及代码实现：

1. MulMod():计算 `(a * b) % n`，用于 RSA 运算中的模乘操作。

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

2. PowMod()：计算 `(base ^ pow) % n`，用于 RSA 运算中的模幂操作。

```
__int64 CRsaOperate::PowMod(__int64 base, __int64 pow, __int64 n) {
    __int64 a = base, b = pow, c = 1;
    while(b){
        while(!(b & 1)){
            b >>= 1;
            a = MulMod(a, a, n);
        }
        b--;
        c = MulMod(a, c, n);
    }
    return c;
}
```

3. RabinMillerKnl()：Rabin-Miller 素性测试的内部核心部分，用于判断一个数是否为素数。

```
long CRsaOperate::RabinMillerKnl(__int64 &n) {
    __int64 a, q, k, v;
    q = n - 1;
    k = 0;
    while(!(q & 1)) {
        ++k;
        q >>= 1;
    }
    a = 2 + rand() % (n - 3);
    v = PowMod(a, q, n);
    if(v == 1) {
        return 1;
    }
    for(int j = 0; j < k; j++) {
        unsigned int z = 1;
        for(int w = 0; w < j; w++) {
            z *= 2;
        }
        if(PowMod(a, z*q, n) == n - 1)
            return 1;
    }
    return 0;
}
```

4. RabinMiller()：执行 Rabin-Miller 素性测试，重复进行测试以提高准确性。

```
long CRsaOperate::RabinMiller(__int64 &n, long loop=100) {
    for(long i = 0; i < loop ; i++){
        if(!RabinMillerKnl(n)){
            return 0;
        }
    }
    return 1;
}
```

5. RandPrime()：生成一个指定位数的随机素数，用于生成 RSA 密钥中的素数。

```
__int64 CRsaOperate::RandPrime(char bit) {
    __int64 base;
    do{
        base = (unsigned long)1 << (bit - 1);
        base += rand() % (base);
        base |= 1;
    }
    while(!RabinMiller(base, 30));
    return base;
}
```

6. Gcd()：计算两个数的最大公约数，用于 RSA 密钥生成中的参数选择。

```
__int64 CRsaOperate::Gcd(__int64 &p, __int64 &q) {
    unsigned long long a = p > q ? p : q;
    unsigned long long b = p < q ? p : q;
    unsigned long long t;
    if( p == q ){
        return p;
    }else{
        while(b){
            a = a % b;
            t = a;
            a = b;
            b = t;
        }
        return a;
    }
}
```

7. Euclid()：使用扩展欧几里得算法计算模反元素，用于 RSA 密钥生成中的私钥计算。

```
__int64 CRsaOperate::Euclid(__int64 e, __int64 t_n) {
    unsigned long long Max = 0xffffffffffffffff - t_n;
    unsigned long long i = 1;
    while(1){
        if(((i*t_n)+1)%e == 0){
            return ((i*t_n)+1)/e;
        }
        i++;
        unsigned long long Tmp = (i+1)*t_n;
        if(Tmp > Max){
            return 0;
        }
    }
    return 0;
}
```

8. Encry()：使用公钥对给定的数据进行加密。

```
__int64 CRsaOperate::Encry(unsigned short nScore, PublicKey &cKey) {
    return PowMod(nScore, cKey.nE, cKey.nN);
}
```

9. Decry()：使用私钥对给定的加密数据进行解密。

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

10. GetPublicKey()：获取当前对象的公钥。

```
PublicKey CRsaOperate::GetPublicKey() {
    PublicKey cTmp;
    cTmp.nE = this -> m_cParament.e;
    cTmp.nN = this -> m_cParament.n;
    return cTmp;
}
```

（三）全双工通信

相比于实验一，通信部分除了主客户端 socket 的绑定、监听、连接等等，主要增加的部分有：

1. 密钥分配
2. 使用 select 机制

1. 密钥分配

(1) Server 端生成 RSA 公钥私钥对

(2) Server 端将 RSA 公钥通过 socket 发送到 Client 端

(3) Cilent 端随机生成 DES 密钥

(4) Client 使用公钥加密 DES 密钥，然后将加密后 DES 密钥发回 Server 端

代码实现：

Server 部分：

```
// 密钥协商
PublicKey cRsaPublicKey;
CRsaOperate CRsaOperate;
cRsaPublicKey = CRsaOperate.GetPublicKey();
// 发送RSA公钥
if(send(nAcceptSocket, (char *)(&cRsaPublicKey), sizeof(cRsaPublicKey), 0) != sizeof(cRsaPublicKey)){
    perror("send");
    exit(0);
}else{
    printf("successful send the RSA public key. \n");
}
// 接收DES密钥
unsigned long long nEncryptDesKey[4];
char *strDesKey = new char[8];
if(4*sizeof(unsigned long long) != TotalRecv(nAcceptSocket,(char *)nEncryptDesKey, 4*sizeof(unsigned long l
    perror("TotalRecv DES key error");
    exit(0);
}
else {
    printf("successful get the DES key\n");
    // 解密DES密钥
    unsigned short * pDesKey = (unsigned short *)strDesKey;
    for(int i = 0;i < 4; i++) {
        pDesKey[i] = CRsaOperate.Decry(nEncryptDesKey[i]);
    }
}
```

Client 部分：

```
// 生成DES密钥
GerenateDesKey(strDesKey);
printf("Create DES key success\n");
PublicKey cRsaPublicKey;
// 接收RSA公钥
if(sizeof(cRsaPublicKey) == TotalRecv(nConnectSocket,(char *)&cRsaPublicKey,
    printf("Successful get the RSA public Key\n");
}
else {
    perror("Get RSA public key ");
    exit(0);
}
// 加密DES密钥并发送给服务器
unsigned long long nEncryptDesKey[4];
unsigned short *pDesKey = (unsigned short *)strDesKey;
for(int i = 0; i < 4; i++) {
    nEncryptDesKey[i] = CRsaOperate::Encry(pDesKey[i],cRsaPublicKey);
}
if(sizeof(unsigned long long)*4 != send(nConnectSocket, (char *)nEncryptDesKe
    perror("Send DES key Error");
    exit(0);
}
else {
    printf("Successful send the encrypted DES Key\n");
}
```

2. 使用 select 机制

为了提升程序效率，Linux 提供了 select 函数接口，用以同时管理若干个套接字或者句柄上的 IO 操作，通过该 API，程序可以同时监控多个 socket 或者句柄上的 IO 操作。

它在一个进程中同时监视多个文件描述符的状态，包括是否可读、是否可写等。

它允许程序等待多个文件描述符中的任何一个变为就绪状态，从而实现同时处理多个 I/O 操作的能力，进而可以免去开启多个进程的系统开销。

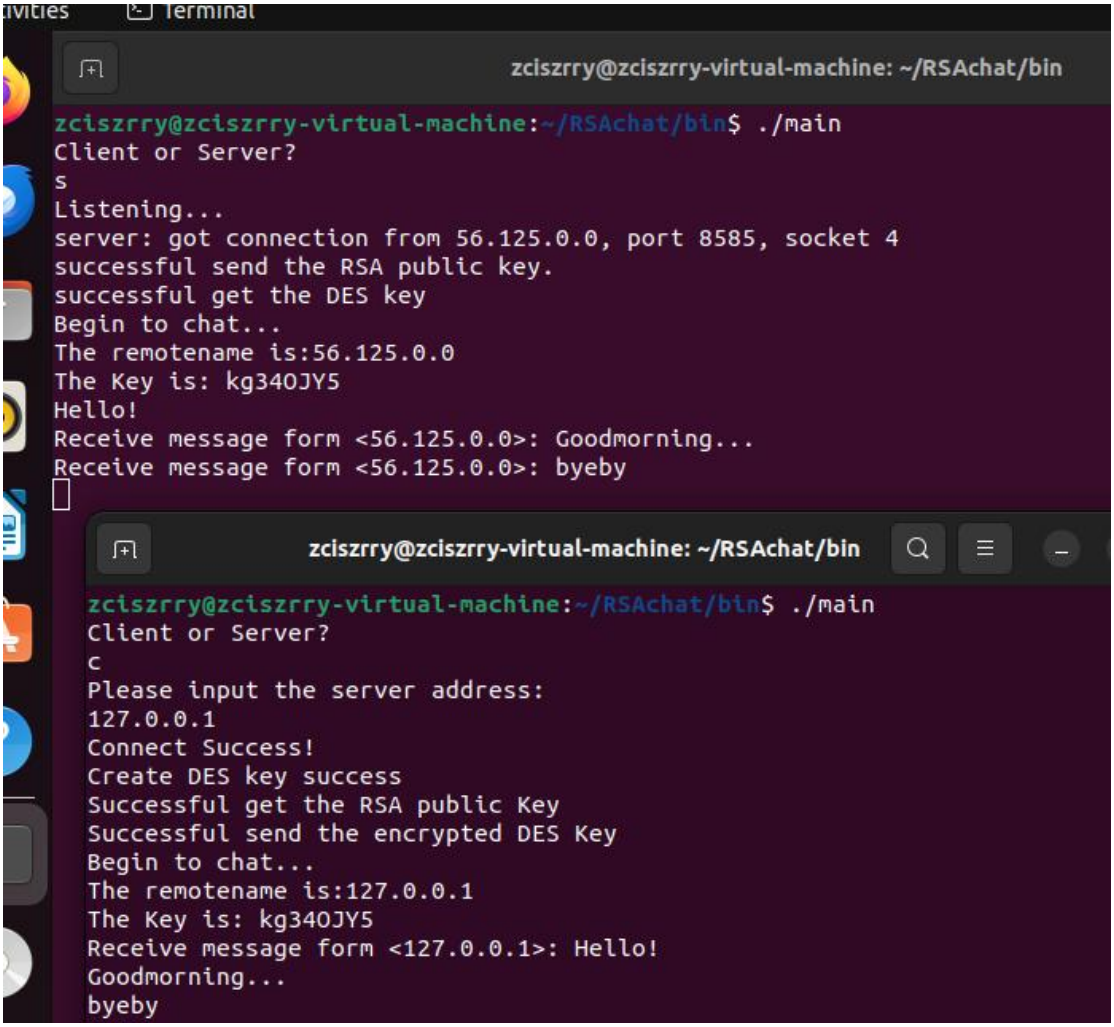
代码如下：

```
// 使用 select 模型重写通信函数
void SecretChat(int nSock, char *pRemoteName, char *pKey){

    std::cout << pRemoteName << std::endl;
    // 创建 DES 操作对象
    CDesOperate cDes;
    std::cout << pKey << std::endl;
    // 检查密钥长度
    int klength = strlen(pKey);
    if(klength != 8){
        printf("%s\n",pKey);
        printf("Key length error\n");
        return;
    }

    // 初始化 select 模型需要的变量
    fd_set cHandleSet;
    struct timeval tv;
    int nRet;
    while(1){
        // 清空文件描述符集合
        FD_ZERO(&cHandleSet);
        // 添加套接字和标准输入文件描述符到集合中
        FD_SET(nSock, &cHandleSet);
        FD_SET(0, &cHandleSet);
        // 设置超时时间为1 秒
        tv.tv_sec = 1;
```

```
28.         tv.tv_usec = 0;
29.         // 监听文件描述符的状态
30.         nRet = select(nSock>0? nSock+ 1:1, &cHandleSet, NULL, NULL, &tv);
31.         // 处理 select 函数的返回值
32.         if(nRet < 0){
33.             printf("Select error!\n");
34.             break;
35.         }
36.         if(nRet == 0){
37.             continue;
38.         }
39.         // 处理套接字上的数据
40.         if(FD_ISSET(nSock,&cHandleSet)){
41.             bzero(&strSocketBuffer, BUFFERSIZE);
42.             int nLength = 0;
43.             nLength = TotalRecv(nSock, strSocketBuffer,BUFFERSIZE,0);
44.             if(nLength !=BUFFERSIZE) break;
45.             else{
46.                 int nLen = BUFFERSIZE;
47.                 cDes.Decry(strSocketBuffer,BUFFERSIZE,strDecryBuffer,nLen,pKey,8);
48.                 strDecryBuffer[BUFFERSIZE-1]=0;
49.                 if(strDecryBuffer[0]!=0&&strDecryBuffer[0]!='\n'){
50.                     printf("Receive message form <%s>: %s",pRemoteName,strDecryBuffer)
    ;
51.                     if(0==memcmp("quit",strDecryBuffer,4)){
52.                         printf("Quit!\n");
53.                         break;
54.                     }
55.                 }
56.             }
57.         }
58.         // 处理标准输入上的数据
59.         if(FD_ISSET(0,&cHandleSet)){
60.             bzero(&strStdinBuffer, BUFFERSIZE);
61.             while(strStdinBuffer[0]==0){
62.                 if (fgets(strStdinBuffer, BUFFERSIZE, stdin) == NULL){
63.                     continue;
64.                 }
65.             }
66.             int nLen = BUFFERSIZE;
67.             cDes.Encry(strStdinBuffer,BUFFERSIZE,strEncryBuffer,nLen,pKey,8);
68.             if(send(nSock, strEncryBuffer, BUFFERSIZE,0)!=BUFFERSIZE){
69.                 perror("send");
70.             }else{
```

```
                if(0==memcmp("quit",strStdinBuffer,4)){
                    printf("Quit!\n");
                    break;
                }
            }
        }
    }
}
```

实验结果：



可见，程序实现了基于 DES 加密的 TCP 通信，实现密钥自动生成，并基于 RSA 算法进行密钥共享，且加密过程对用户完全透明。

四、实验遇到的问题及其解决方法

(1) cmake 的学习使用，CMakeLists.txt 的语法比较复杂。解决方法：查找参考资料，借助大语言模型

(2) RSA 进行大素数生成和幂乘时，若不注意细节，很容易出现崩溃。需要仔细阅读参考书。

五、实验结论

通过本次实验，成功地完成了基于 RSA 算法的自动分配密钥加密聊天程序的编写。

在程序中，使用了 CMake 来管理项目的构建过程，确保项目可以在 linux 上顺利构建和运行。

进一步复习了课堂上的理论知识，包括 DES 和 RSA 的加解密。

对 linux 的 select 模型有了初步认识，还联想到了曾经在操作系统学习过的进程管理相关的知识。