南开大学
Nankai University

南　开　大　学

网络空间安全学院

密码学大作业实验报告

RSA&AES&CBC

姓名：袁铭泽

学号：2012777

专业：信息安全

目录

一、 实验目的

设计一个保密通信的协议，具体要求为：利用 RSA 公钥密码算法，为双方分配一个 AES 算法的会话密钥，然后利用 AES 加密算法和分配的会话密钥，加解密传送的信息。

假设条件：假设通讯双方为 A 和 B，并假设发方拥有自己的 RSA 公钥 PK_A 和私钥 SK_A ，同时收方拥有自己的 RSA 公钥 PK_B 和私钥 SK_B ，同时收发双方已经通过某种方式知道了双方的公钥。

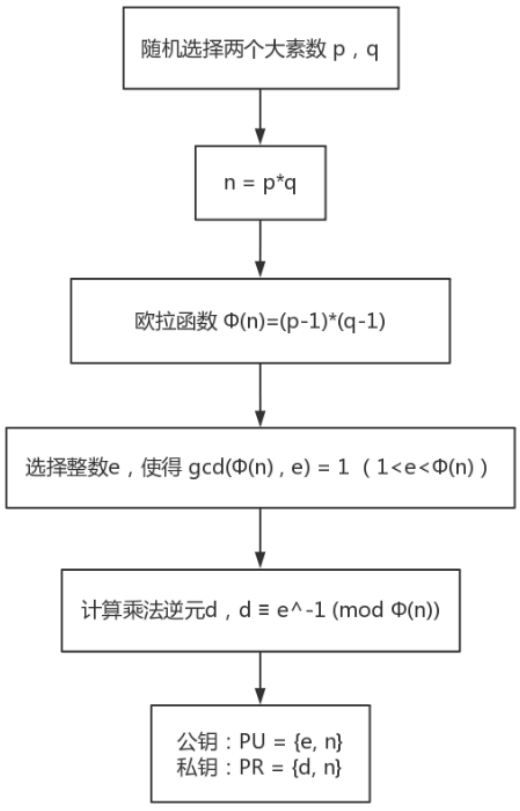
二、 实验原理

1.RSA 加密机制

属于非对称加密，公钥用于对数据进行加密，私钥对数据进行解密，两者不可逆。公钥和私钥是同时生成的，且一一对应。比如：A 拥有公钥，B 拥有公钥和私钥。A 将数据通过公钥进行加密后，发送密文给 B，B 可以通过私钥和公钥进行解密。

由于之前的实验涉及到 RSA,这里只对其进行简单介绍：

RSA 算法流程图：



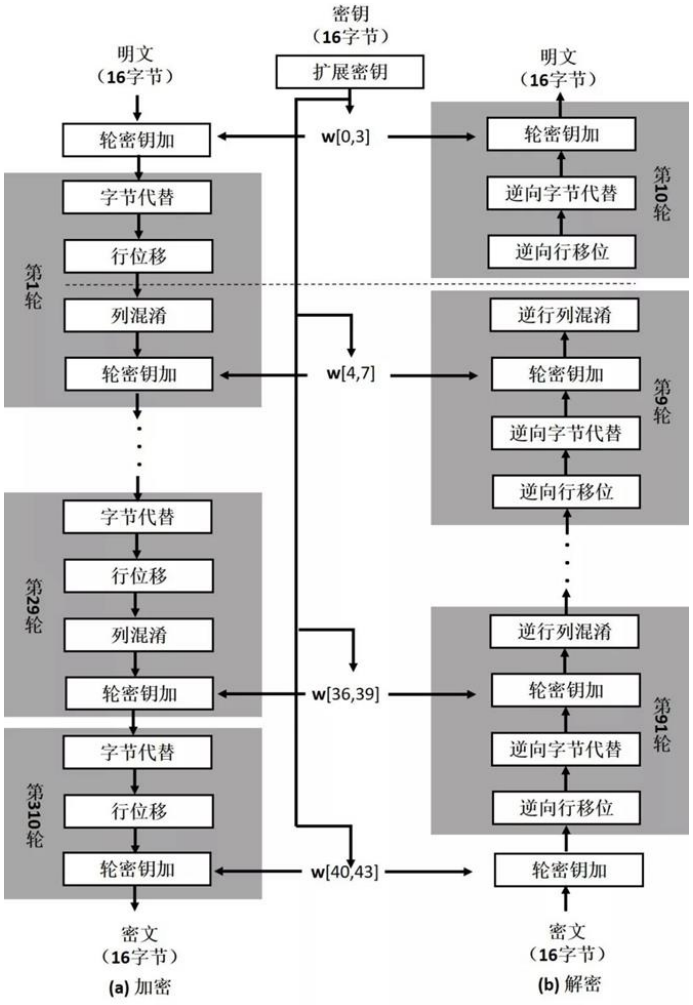
2.AES 加密机制

属于对称加密，就是说，A 用密码对数据进行 AES 加密后，B 用同样的密码对密文进行 AES 解密。

由于之前的实验中涉及 AES,这里只对其进行简单介绍:

AES 加密的主要过程包括：对明文状态的一次密钥加，N_r-1轮轮加密和末尾轮轮加密，最后得到密文。其中N_r-1轮轮加密每一轮有四个部件，包括字节代换部件、行移位变换、列混合变换和一个密钥加部件，末尾轮加密和前面轮加密类似，只是少了一个列混合变换部件。

AES 加解密流程图：



3.RSA&AES 加密

利用 RSA 来加密传输 AES 的密钥，用 AES 的密钥来加密数据。这样做既利用了 RSA 的灵活性，可以随时改动 AES 的密钥；又利用了 AES 的高效性，可以高效传输数据：

单纯使用 RSA 非对称加密的话，效率会很低，因为非对称加密方式虽然很保险，但是过程复杂，需要时间长；但，RSA 优势在于数据传输安全，用它来加密字节数不大的 AES 密钥就非常合适。

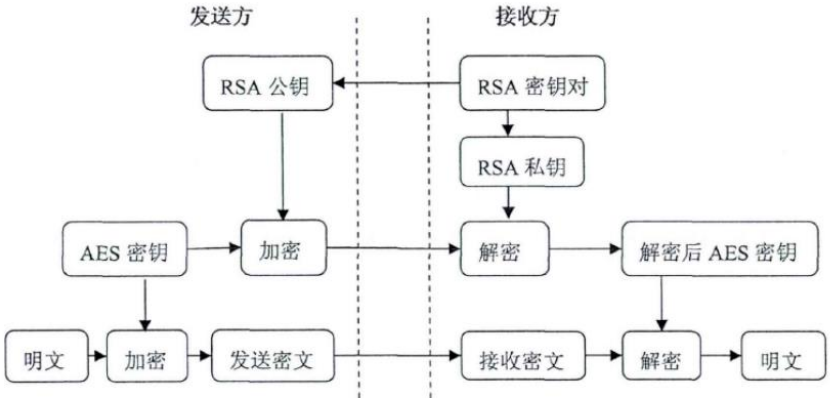
单纯利用 AES 对称加密方式的话，客户端和服务端的密钥是固定一样的，一旦密钥被人获取，那么我们发送的数据会被对方破解。但 AES 具有一个很大的优点，即，加密解密效率高，这种方式就适合传输量大的数据内容。

基于以上特点，就形成了 RSA,AES 混合加密的思路。

具体流程如下：

接收方利用 RSA 算法生成一对公钥和私钥，将公钥发送至发送方，自己保留私钥。接收方收到 RSA 公钥后，用其对 AES 密钥进行加密，将加密后的 AES 密钥发给接收方。接收方使用 RSA 私钥进行解密，获得解密后的 AES 密钥。当发送方向接收方发送信息，将明文用 AES 密钥进行加密，发送密文，接收方收到密文后，用解密后的 AES 密钥对其解密，获取发送信息的明文。

双向传输中，则是双方均为发送方和接收方，对称的进行上述操作。由于本实验的假设中说明，收发双方已经通过某种方式知道了双方的公钥，因此可以省去 RSA 公钥的传输过程，只传输加密后的 AES 密钥和加密后的信息。



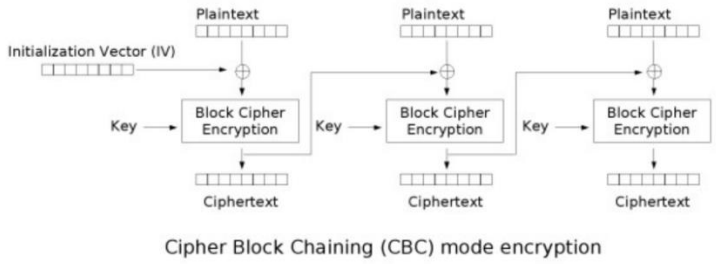
4.CBC 模式

AES-CBC 是一种分组对称加密算法，以 AES-128-CBC 为例进行说明：用同一组 key 进行明文和密文的转换，以 128bit 为一组，128bit==16Byte，即明文的 16 字节为一组对应加密后的 16 字节的密文。

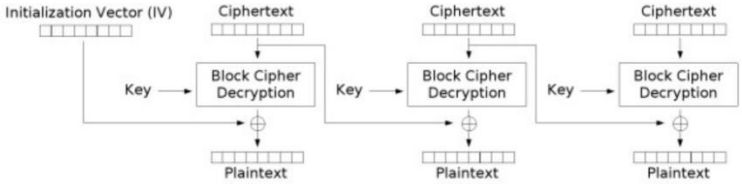
若最后剩余的明文不够 16 字节，需要进行填充，通常采用 PKCS7 进行填充。比如最后缺 3 个字节，则填充 3 个字节的 0x03;若最后缺 10 个字节，则填

充 10 个字节的 0x0a; 若明文正好是 16 个字节的整数倍，最后要再加入一个 16 字节 0x10 的组再进行加密。

CBC 加解密原理如下所示：



Cipher Block Chaining (CBC) mode encryption



CBC 加密原理：明文跟向量异或，再用 KEY 进行加密，结果作为下个 BLOCK 的初始化向量。CBC 解密原理：使用密钥先对密文解密，解密后再同初始向量异或得到明文。CBC 需要对明文块大小进行补位，由于前后加密的相关性，只能实施串行化动作，无法并行运算。另外，CBC 需要参量：密钥和初始化向量。

三、 基本思路实现

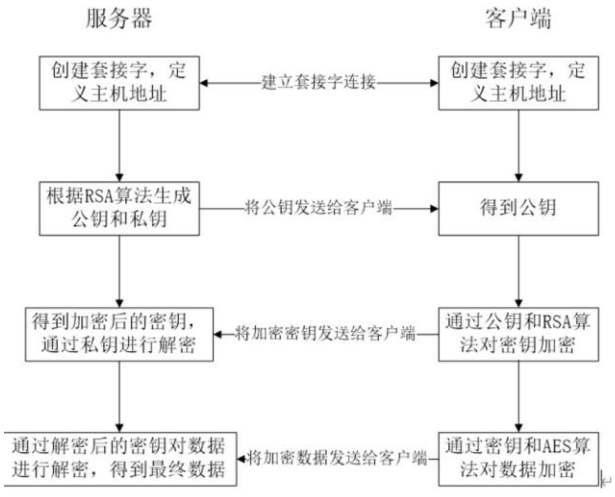
通过编写客户端程序和服务器程序，利用 windows 下的 socket 套接字来实现同一台主机上客户端与服务器进程之间数据的网络传输，用来实现密钥的分发和数据的加密传输。

其中聊天信息带有时间标签（通过时间戳获得），并创建了多线程，创建接收消息的线程，避免因为 recv 函数是阻塞函数造成的主线程等待问题，因此支持随时发送和接收消息，一方可连续发送多条信息。

密钥的传输基于 RSA 算法实现，由服务器根据 RSA 算法生成公钥和私钥，然后将公钥传输给客户端，由客户端根据公钥负责将 AES 加密算法的密钥进行加密传给服务器，服务器根据自己的私钥和 RSA 算法进行解密处理，得到 AES 的密钥。由于 RSA 是非对称加密算法，因此即使攻击者截取到了加密的数据和公钥，也会因为没有私钥而无法解密，保证了密钥的安全传输。

数据之间的传输基于 AES 算法实现，由客户端根据先前的密钥和 AES 算法对数据实现加密传输，再在服务器端根据计算得到的 AES 密钥对数据进行解密处理，从而得到最终的数据。

总体流程图：



双向传输则两端均作如上操作。

1. 协议设计

计算机网络中，各个实体之间的数据交换必须遵守事先约定好的规则，这些规则就称为协议。

a. 消息类型

共三种消息类型：

```
enum Type
{   chat,
    csexit,
    aes
};
```

chat: 服务器与客户端的普通聊天信息，只需正常打印出发送信息的时间、发送者和信息内容；

csexit: 服务器下线或客户端断开与服务器的连接，打印出下线的提示信息。

aes:发送加密后的 aes 密钥，打印出接收到的 aes 密钥信息。

b. 消息结构体

```
struct message {
    Type type;
    string time;
    string msg;
};
```

发送消息时，首先发送 RSA 公钥加密后的 AES 密钥，将消息类型设为 aes，将 RSA 公钥加密后的 aes 密钥（大整数类型）转换为 string 类型，放入消息的 msg 中进行发送。

```
/**
* 函数功能:将大整数转换为十六进制字符串并返回
*/
std::string BigInt::toString() const {
    std::string ans;
    base_t t = 0xffffffff;
    t <<= 32 - 4;    // 用于截取高4位
    for (int i = data1.size() - 1; i >= 0; --i) {
        base_t temp = data1[i];
        for (int j = 0; j < 8; ++j) {
            base_t num = t & temp;// 每次截取高4位
            num >>= 32 - 4;    // 将高4位移到低4位
            temp <<= 4;
            if (num < 10)
                ans.push_back((char)('0' + num));
            else
                ans.push_back((char)('A' + num - 10));
        }
    }
    while (ans.size() > 0 && ans.at(0) == '0')// 去掉高位无用的0
        ans = ans.substr(1);
    if (ans.empty())    // 空串说明为0
        ans.push_back('0');
    if (sign)    // 为负数加上负号
        ans = "-" + ans;
    return ans;
}
```

接收方收到加密后的 aes 密钥（string 类型）后，将其转化为大整数类型与 rsa 私钥进行解密，获得解密后的 aes 密钥。

```
/**
* 函数功能:根据给定的十六进制字符串数据构造一个大整数
*/
BigInt::BigInt(const std::string& str) : sign(false) {
    std::string t(str);
    if (t.size() && t.at(0) == '-') {
        if (t.size() > 1)
            sign = true;
        t = t.substr(1);
    }
    int cnt = (8 - (t.size() % 8)) % 8;// 数的长度不是8的倍数,补足0
    std::string temp;

    for (int i = 0; i < cnt; ++i)
        temp.push_back('0');

    t = temp + t;

    for (size_t i = 0; i < t.size(); i += 8) {
        base_t sum = 0;
        for (int j = 0; j < 8; ++j) {    // 8位十六进制组成大整数的一位
            char ch = t[i + j];
            int num = hexToNum(ch);
            sum = ((sum << 4) | (num));
        }
        data1.push_back(sum);
    }
    reverse(data1.begin(), data1.end());// 高位在后
    *this = trim();// 去除高位的0
}
```

接下来，发送正常聊天信息，窗口接收输入的聊天信息，判断消息类型，通过时间戳计算出发送消息的时间，将消息存入 message 结构体。再将 message 转为字符串类型，将聊天信息通过 AES-CBC 加密后发送，若消息类型为 csexit，命令行打印关闭的提示信息，关闭 socket。

```
// message类型转换为string类型，字段间以'\n'为分隔符
string mtos(message m) {
    string s;
    if (m.type == 0) {
        s = '0';
    }
    else if (m.type == 1) {
        s = '1';
    }
    else if (m.type == 2) {
        s = '2';
    }
    s.append("\n");
    s.append(m.time);
    s.append("\n");
    s.append(m.msg);
    s.append("\n");
    return s;
}
```

接收消息时，首先将字符串解析为 message 结构体类型，判断消息类型，若为 chat，打印聊天信息，并将字符串（加密后的聊天消息）由 AES-CBC 解密，打印解密后的消息明文。

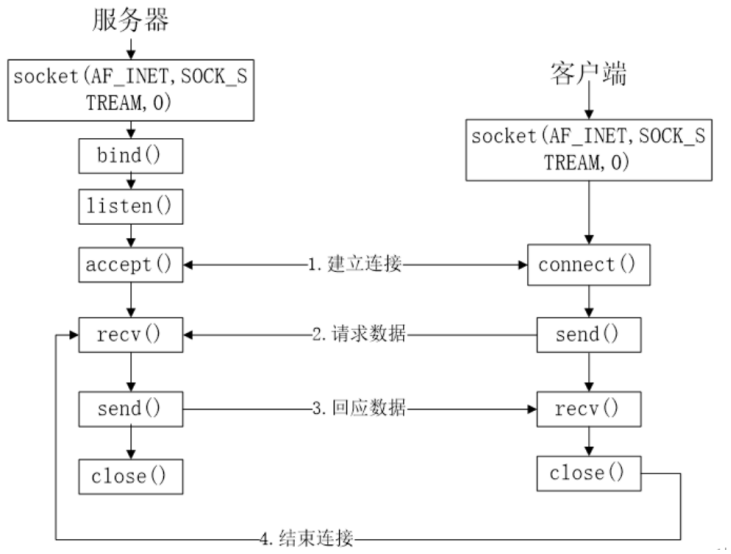
```
// string类型转换为message类型，字段间以'\n'为分隔符
message stom(string s) {
    message m;
    if (s[0] == '0') {
        m.type = chat;
    }
    else if (s[0] == '1') {
        m.type = csexit;
    }
    else if (s[0] == '2')
        m.type = aes;
    int i = 2;
    while (s[i] != '\n') {
        i++;
    }
    m.time = s.substr(2, i-2 );
    m.msg = s.substr(i + 1);
    return m;
}
```

2. 建立 socket 套接字连接

在网络中，要全局的标识一个参与通信的进程，需要三元组：协议，IP 地址以及端口号。要描述两个应用进程之间的端到端的通信关联需要五元组：协议，信源主机 IP，信源应用进程端口，信宿主机 IP，信宿应用进程端口。为了实现两个应用进程的通信连接，提出了套接字的概念。套接字可以理解为通信连接的一端，将两个套接字连接在一起，可以实现不同进程之间的通信。

在本程序中，首先通过 socket 函数分别为客户端和服务器创建套接字，然后定义家族协议、主机地址和主机端口等，得到 socket 嵌套字，通过 bind 函数在服务器端将套接字地址与所创建的套接字号连接起来，设定套接字为监听状态，准备接收由客户端进程发出的连接请求。客户端提出与服务器建立连接的请求，如果服务器进程接受请求，则服务器进程与客户机进城之间便建立了一条通信连接，之后便可以通过 recv 函数和 send 函数分别实现在已建立的套接字上接收和发送数据，实现同一台主机下数据之间的网络传输。

建立 socket 连接的流程图如下：



3. RSA 算法实现密钥加密分发

本程序中，RSA 算法步骤如下：

1.由服务器端生成两个不相等的质数 p 和 q；然后计算 p 和 q 的乘积 n。

2.计算 n 的欧拉函数 φ(n)=（p-1）*(q-1)，选择一个整数 e，条件是 1< e < φ(n)，且 e 与 φ(n)互质；

3. 计算 e 对于 φ(n)的模反元素 d，使得 ed ≡ 1 (mod φ(n))。通过扩展欧几里得算法求解得到 d 的值作为私钥的一部分。

4.将 n 和 e 封装成公钥，n 和 d 封装成私钥，客户端得到 RSA 公钥后，对 AES 的密钥进行加密传输给服务器端，即用公钥（n,e）算出 m^e≡c (mod n)中 c 的值；服务器利用私钥解密得到 AES 的密钥，同样也是利用 m^e≡c (mod n)算出 c 的值即为 AES 的密钥。

```
//RSA部分
//产生大素数
BigInt p = GeneratePrime();
//16进制形式显示
BigInt q = GeneratePrime();
BigInt n = p * q;
BigInt t = (p - 1) * (q - 1);
//e为公开钥,d为秘密钥，即e模t的乘法逆元,y用于参与扩展欧几里得运算，存储t模e的乘法逆元
BigInt e, d, y, temp;
while (1)
{
    //产生与t互质的e
    e.Random();
    while (!(Gcd(e, t) == 1))
    {
        e.Random();
    }
    //用扩展欧几里德算法试图求出e模t的乘法逆元
    temp = ExtendedGcd(e, t, d, y);
    //e*d模t结果为1，说明d确实是e模t的乘法逆元
    temp = (e * d) % t;
    if (temp == 1)
        break;
    //否则重新生成e
}
cout << "client端公钥密钥已生成" << endl;
BigInt ae;
ae.Random();//生成 作为aes密钥
cout << "生成AES密钥：" << endl;
ae.display();
```

RSA 部分代码详见 bigint.h，gen.h，client/server.cpp。

4. AES-CBC 算法实现数据加密传输

AES 的 CBC 模式加密：

```
string EncryptionAES(const string& strSrc) //AES加密
{
    size_t length = strSrc.length();
    int block_num = length / BLOCK_SIZE + 1;
    //明文
    char* szDataIn = new char[block_num * BLOCK_SIZE + 1];
    memset(szDataIn, 0x00, block_num * BLOCK_SIZE + 1);
    strcpy_s(szDataIn, strlen(strSrc.c_str()) + 1, strSrc.c_str());
    //进行PKCS7填充。
    int k = length % BLOCK_SIZE;
    int j = length / BLOCK_SIZE;
    int padding = BLOCK_SIZE - k;
    for (int i = 0; i < padding; i++)
        szDataIn[j * BLOCK_SIZE + k + i] = padding;
    szDataIn[block_num * BLOCK_SIZE] = '\0';
    //加密后的密文
    char* szDataOut = new char[block_num * BLOCK_SIZE + 1];
    memset(szDataOut, 0, block_num * BLOCK_SIZE + 1);
    //进行AES的CBC模式加密
    AES aes;
    aes.MakeKey(g_key, g_iv, 32, 16);
    aes.Encrypt(szDataIn, szDataOut, block_num * BLOCK_SIZE, AES::CBC);
    string str = base64_encode((unsigned char*)szDataOut,
        block_num * BLOCK_SIZE);
    delete[] szDataIn;
    delete[] szDataOut;
    return str;
}
```

AES 的 CBC 模式解密：

```
string DecryptionAES(const string& strSrc) //AES解密
{
    string strData = base64_decode(strSrc);
    size_t length = strData.length();
    //密文
    char* szDataIn = new char[length + 1];
    memcpy(szDataIn, strData.c_str(), length + 1);
    //明文
    char* szDataOut = new char[length + 1];
    memcpy(szDataOut, strData.c_str(), length + 1);
    //进行AES的CBC模式解密
    AES aes;
    aes.MakeKey(g_key, g_iv, 32, 16);
    aes.Decrypt(szDataIn, szDataOut, length, AES::CBC);
    //去PKCS7Padding填充
    if (0x00 < szDataOut[length - 1] && szDataOut[length - 1] <= 0x16)
    {
        int tmp = szDataOut[length - 1];
        for (int i = length - 1; i >= length - tmp; i--)
        {
            if (szDataOut[i] != tmp)
            {
                memset(szDataOut, 0, length);
                cout << "去填充失败！解密出错！！" << endl;
                break;
            }
            else
                szDataOut[i] = 0;
        }
    }
    string strDest(szDataOut);
    delete[] szDataIn;
    delete[] szDataOut;
    return strDest;
}
```

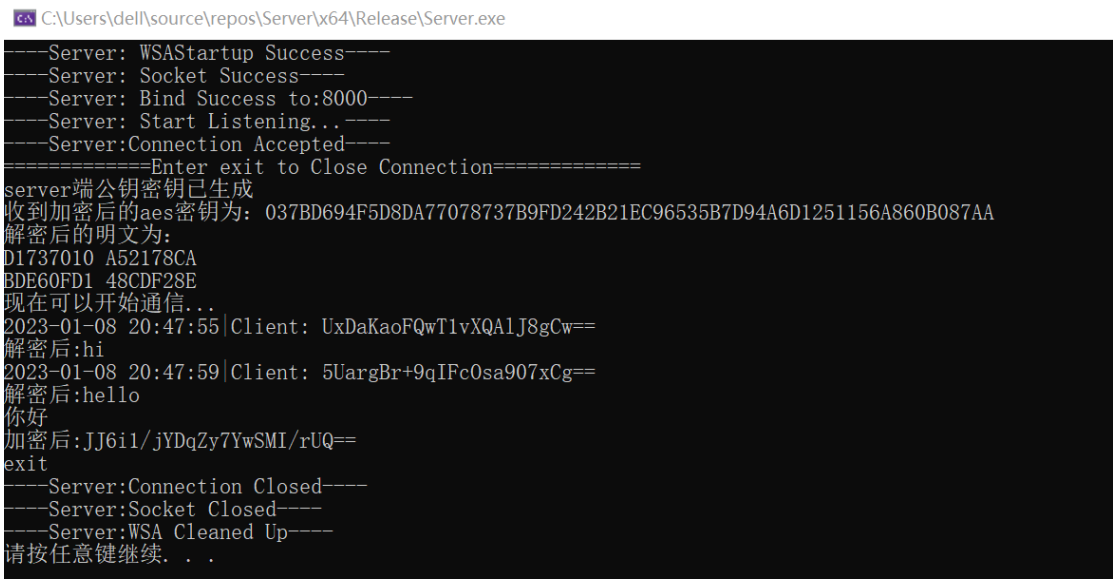
AES 算法在对明文加密时，并不是把整个明文加密成一整段密文，而是把明文拆分为一个个独立的明文块，每一个明文块长度 128bit，这些明文块经过 AES 加密器复杂处理，生成一个个独立的密文块，这些密文块拼接在一起，就是最终的 AES 加密的结果。如果明文块长度不足 128bit，就需要对明文块进行填充。填充方式有很多种，这里采用 PKCS7Padding。PKCS5Padding：如果明文块少于 16 个字节，在明文块末尾补足相应数量的字符，且每个字节的值等于缺少的字符数。PKCS7Padding 原理与 PKCS5Padding 相似，区别是 PKCS5Padding 的 blocksize 为 8 字节，而 PKCS7Padding 的 blocksize 可以为 1 到 255 字节。

AES 的工作模式，体现在把明文块加密成密文块的处理过程中。AES 加密算法提供了五种不同的工作模式：CBC,ECB,CTR,CFB,OFB。本实验采用 CBC 模式，它的原理如上文所述。它的优点有：不容易主动攻击；安全性好于 ECB；适合传输长度长的报文；是 SSL、IPSec 的标准。它的缺点有：不利于并行计算；误差传递；需要初始化向量 IV。

AES-CBC 的具体代码详见 AES.h, Base64.h, AES.cpp，Base64.cpp，server/client.cpp。

四、 实验结果

Server 端：可以看到 socket 建立成功，收到加密后的 aes 密钥，并成功解密，解密出的明文与 client 端生成的 aes 密钥明文一致。成功收到 client 端经过加密后发来的信息，解密后恢复消息明文。由于采用多线程，无阻塞问题，同一端可以连续发送多条信息。可以看到英文、中文均可正确发送接收。输入 exit 后 server 端关闭连接。



Client 端：可以看到 socket 建立成功，生成 aes 密钥，进行加密并发送。数据传输效果与 server 端相对应。Server 端输入 exit 之后，client 端提示对方已断连，再输入回车，client 端套接字也进行关闭。