实验目的与要求：

1、 理解公钥密码的基本思想
2、 理解 RSA 和数字签名的原理
3、 掌握 RSA 和数字签名的输入输出格式和密钥格式
4、 实现 RSA 算法公钥加密会话密钥，然后使用会话密钥和 3DES 加解密各种（word、txt、mp3、jpg）文件。

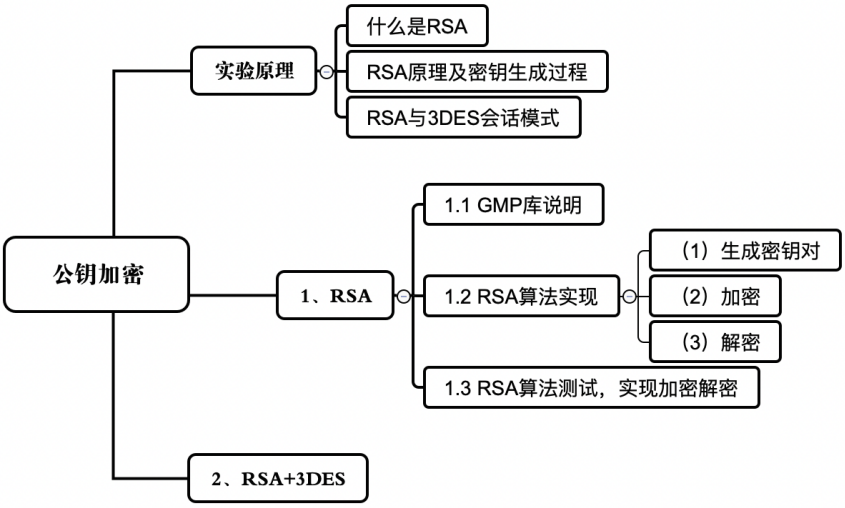
实验环境：

MacOS Visual Studio 2019；C++语言

实验原理：

简单描述 RSA 公钥加密会话密钥方法，然后应用分组密码 3DES 进行加密方案。

本次实验的思维导图如下所示：



1、什么是 RSA

RSA 是最流行的非对称加密算法之一，也被称为公钥加密，于 1977 年提出。

RSA 是非对称加密的一种，即用来加密的密钥和用来解密的密钥不是同一个。和 DES 一样也是分组加密算法，不同的是分组大小可以根据密钥的大小而改变。如果加密的数据不是分组大小的整数倍，则会根据具体的应用方式增加额外的填充位。RSA 很重要的一特点是当数据在网络中传输时，用来加密数据的密钥并不需要也和数据一起传送。因此，这就减少了密钥泄露的可能性。RSA 在不允许加密方解密数据时也很有用，加密的

一方使用一个密钥，称为公钥，解密的一方使用另一个密钥，称为私钥，私钥需要保持其私有性。

RSA 被认为是非常安全的，不过计算速度要比 DES 慢很多。同 DES 一样，其安全性也从未被证明过，但想攻破 RSA 算法涉及的大数（至少 200 位的大数）的因子分解是一个极其困难的问题。由于缺乏解决大数的因子分解的有效方法，因此，可以推测出目前没有有效的办法可以破解 RSA。

2、 RSA 原理及密钥生成过程

RSA 算法的原理是：根据数论，寻求两个大素数比较简单，而将它们的乘积进行因式分解却极其困难，因此可以将乘积公开作为加密密钥。

RSA 加密和解密数据围绕着模幂运算, 这是取模计算中的一种。取模计算是整数计算中的一种常见形式，比如 $x \bmod n$ 的结果就是 x/n 的余数。而模幂运算就是计算 $a^b \bmod n$ 的过程。

密钥的生成是 RSA 算法的核心，其密钥对生成过程如下：

1. 选择两个不相等的大素数p和q,计算出$n = pq$,n被称为 RSA 算法的公共模数;
2. 计算n的欧拉数$\phi(n)$, $\phi(n) = (p-1)(q-1)$;
3. 随机选择一个整数e作为公钥加密密钥指数,$1 < e < \phi(\mathrm{n})$,且e与$\phi(\mathrm{n})$互质;
4. 利用同余方程$ed \equiv 1(\mathrm{mod}\phi(n))$计算$e$对应的私钥解密指数 d。由于$\mathrm{GCD}\ (e, \phi(\mathrm{n})) = 1$,因此同余方程有唯一解,$d$就是$e$对于模$\phi(\mathrm{n})$的乘法逆元;
5. 将(e,n)封装成公钥,(d,n)封装成私钥，同时销毁p和q。

3、RSA 与 3DES 会话模式

实现 RSA 算法后，可以使用 RSA 对 3DES 中的会话密钥进行加密，再用 3DES 对文件进行加密和解密。

RSA 加密，实现了公开密钥。B 可以给所有人发送公钥，其他人把要加密的信息用这把公钥加密后发送给 B，B 用自己的私钥就可以获得加密的信息了。

如果 A 想要将会话秘钥发给 B，并且只有 B 可以解开，那么就可以用 B 的公钥加密会话秘钥，然后将加密后的会话秘钥传给 B，B 就可以用自己的私钥解得会话秘钥。之后 A 如果想给 B 发送私密信息，就可以用会话秘钥并且采用 3DES 算法加密发送，B 使用之前解得的会话秘钥就可以查看 A 所加密的信息。具体流程如图 1 所示。

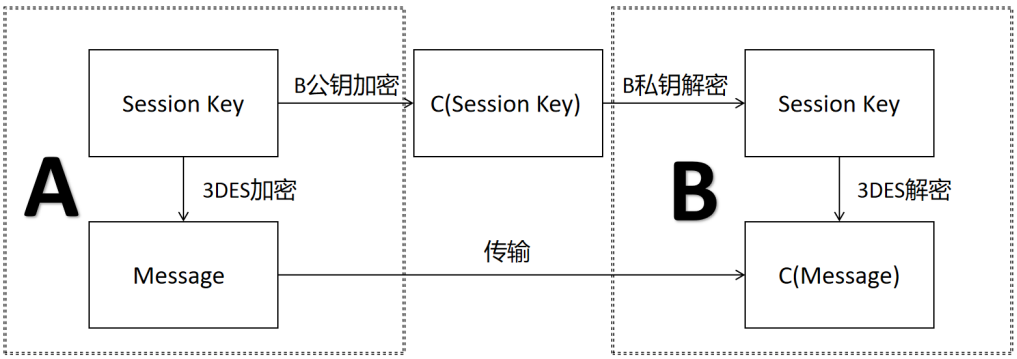


图 1. RSA 与 3DES 会话模式

实验内容：

1、 编程实现 RSA，并展示 RSA 加解密过程。
2、 编程实现 RSA 公钥加密会话密钥，并应用分组密码 3DES 加密过程。

实验步骤与结果：

1、编程实现 RSA，并展示 RSA 加解密过程。

1.1 GMP 库说明

RSA 中的密钥长度指的是公钥的长度，目前主流的公钥长度为 1024、2048 和 4096 位。由于已经有 768 位公钥被成功分解的先例，所以低于 1024 位的公钥都被认为是不安全的。而 C++自带的基本类型远远无法满足 RSA 的运算需求，RSA 算法的实现必须依赖于高精度整型运算。

所以在对 RSA 代码编写的过程之中就必须使用到 GMP 大数库，此库是一个基于 C 语言的开源库，调用前需要进行安装[1]，其中包含了数种自定义数据类型，包括：

```
1.  mpz_t   //多精度整型
2.  mpq_t   //多精度有理数
3.  mpf_t   //多精度浮点型
```

GMP 中的算术函数通常将保存输出结果的变量作为第一个参数，其后的参数为操作数[2]。调用时需要包含<gmp.h>头文件。需要主题的是，根据官方说明：“GMP C++ functions are in a separate libgmpxx library.”所以笔者在 VScode2019 环境中需要使用 *g++ RSA.cpp -lgmpxx -lgmp* 命令进行编译。

1.2 RSA 算法实现

RSA 算法的实现可以分为三个部分：（1）生成密钥对（2）加密（3）解密

（1）生成密钥对

[1] 参考：https://blog.csdn.net/zha_ojunchen/article/details/89818011
[2] 参考：https://gmplib.org/manual/index#Top

生成密钥对包括以下 5 个步骤：（1）随机生成两个足够大的素数（2）计算公共模数 n（3）计算欧拉函数（4）选取一较小的与 φ(n)互质的正整数 e 作为公共指数，数对(n, e)则为密钥对中的公钥（5）计算数论倒数 d，数对(n, d)则为密钥对中的私钥。

第一步，随机生成两个足够大的素数

根据著名的素数定理，随机选取一个正整数n，它是素数的概率为$1/ln\ (n)$，这个概率并不算小，所以笔者在这里这样选取素数：随机选取一个正整数，检测它是否为素数，如果它不是素数，就测试它邻近的正整数，直到找到一个素数为止。

比如，需要生成一个长度为 1024 位的素数，就先随机选取一个长度为 1024 位的正整数，它是素数的概率约为$1/ln\ (2^{1024}) \approx 1/710$，将偶数排除掉，即进行 305 次测试就可找到一个素数。这样，问题就转移到如何测试一个正整数是否为素数上了。目前最常用的素性检测方法是米勒-拉宾素性检测法,这里笔者直接使用 GMP 中的素数生成函数：*void mpz_nextprime (mpz_t rop, mpz_t op);*（表示将 rop 设置为大于 op 的下一个素数）。

```
//生成两个大素数
mpz_t *gen_primes()
{
    gmp_randstate_t grt;              //随机数生成
    gmp_randinit_default(grt);        //设置随机数生成算法为默认
    gmp_randseed_ui(grt, time(NULL)); //设置随机化种子为当前时间

    mpz_t key_p, key_q; //定义mpz_t类型变量
    mpz_init(key_p);
    mpz_init(key_q); //初始化，一个mpz_t类型的变量必须在初始化后才能被使用

    mpz_urandomb(key_p, grt, KEY_LENGTH / 2);
    mpz_urandomb(key_q, grt, KEY_LENGTH / 2); //随机生成两个大整数

    mpz_t *result = new mpz_t[2]; // new存储空间
    mpz_init(result[0]);
    mpz_init(result[1]);

    mpz_nextprime(result[0], key_p); //使用GMP自带的素数生成函数
    mpz_nextprime(result[1], key_q);

    mpz_clear(key_p); //释放占用的内存空间
    mpz_clear(key_q);

    return result; //返回生成的两个大素数
}
```

图 2：随机生成两个足够大的素数

第二步：计算公共模数 n

这一步为简单的乘法运算，首先定义并初始化变量，然后直接调用函数 *void mpz_mul(rop, op1, op2)*。

```
//生成密钥对
key_pair *gen_key_pair()
{
    //第一步：随机生成两个足够大的素数，p,q
    mpz_t *primes = gen_primes(); //调用自己定义的函数生成两个大素数

    //第二步：计算公共模数n，n=p*q
    mpz_t key_n, key_f; //定义并初始化变量
    mpz_init(key_n);
    mpz_init(key_f);
    mpz_mul(key_n, primes[0], primes[1]); //计算n，储存在key_n
```

图 3：计算公共模数 n

第三步：计算欧拉函数

计算欧拉函数值也只是简单的减法和乘法运算。

```
//第三步：计算欧拉函数：φ(n)=(p-1)*(q-1)
mpz_sub_ui(primes[0], primes[0], 1);  // p=p-1
mpz_sub_ui(primes[1], primes[1], 1);  // q=q-1
mpz_mul(key_f, primes[0], primes[1]); //计算欧拉函数，储存在key_f
```

图 4：计算欧拉函数

第四步：选取一较小的与 φ(n)互质的正整数 e 作为公共指数

这一步需要选取一个正整数 e，并输出公钥(n, e)。公共指数常取 3, 17 和 65537 三个值，这里笔者直接取 *e=65537*。

```
//第四步：选取一较小的与φ(n)互质的正整数e作为公共指数。
//数对(n, e)则为密钥对中的公钥
mpz_t key_e;
mpz_init_set_ui(key_e, 65537); //初始化并设置e为65537
// gmp_printf("%s (%ZX, %ZX)\n", "public key is:", key_n, key_e); //输出公钥(n, e)
```

图 5：选取公共指数 e

第五步：计算数论倒数 d

求e在模ϕ(n)下的乘法逆元（也被称为数论倒数）d，也就是求解未知数为d的模线性方程：$d \equiv e-1(mod\phi(n)) \Leftrightarrow ed \equiv 1(mod\phi(n))$。转化为普通二元一次不定方程，即为$ed+k\phi(n)=1$。笔者这里使用 GMP 中的求数论倒数的函数：*int mpz_invert(mpz_t rop, const mpz_t op1, const mpz_t op2)*

```
//第五步：计算数论倒数 d=e-1(modφ(n))
mpz_t key_d;
mpz_init(key_d);
mpz_invert(key_d, key_e, key_f); //求e的数论倒数d
// gmp_printf("%s (%ZX, %ZX)\n", "private key is:", key_n, key_e); //输出私钥(n, d)
```

图 6：计算数论倒数 d

（2）加密

加密的原理为：$C=f_e(M)=M^e \bmod n$，其中 M 为明文，(n, e)为公钥，C 为密文。主要是对 $M^e \bmod n$ 函数进行求值。

这种形如 $a^b \bmod n$ 的运算，称之为模幂运算。模幂运算在密码学中具有十分重要的意义，除了 RSA 加密外，离散对数加密等常用的加密方法里都有模幂运算。通过查找一些资料，笔者发现可以实现快速的模幂运算，下面是快速模幂算法的代码实现：

```
//加密函数
char *encrypt(const char *plain_text, const char *key_n, int key_e)
{
    mpz_t M, C, n;                          //定义并初始化变量
    mpz_init_set_str(M, plain_text, BASE); // M为明文
    mpz_init_set_str(n, key_n, BASE);      //(n, e)为公钥
    mpz_init_set_ui(C, 0);                 // C为密文

    mpz_powm_ui(C, M, key_e, n);              //使用GMP中模幂计算函数
    char *result = new char[KEY_LENGTH + 10]; // new一个空间
    mpz_get_str(result, BASE, C);             //把密文C转化为十六进制并储存到字符数组result中
    return result;                            //返回结果
}
```

图 7：加密函数

（3）解密

解密的原理为：$M=f_d(C)=C^d \bmod n$，其中 C 为密文，(n, d)为私钥，M 为明文。同理也是进行模幂运算，这里不再赘述。

```
//解密函数
char *decrypt(const char *cipher_text, const char *key_n, const char *key_d)
{
    mpz_t M, C, n, d;                       //定义并初始化变量
    mpz_init_set_str(C, cipher_text, BASE); // C为密文
    mpz_init_set_str(n, key_n, BASE);       //
    mpz_init_set_str(d, key_d, BASE);       //(n, d)为私钥
    mpz_init(M);                            // M为明文

    mpz_powm(M, C, d, n);                    //使用GMP中的模幂计算函数
    char *result = new char[KEY_LENGTH + 10]; // new一个空间
    mpz_get_str(result, BASE, M);             //把明文M转化为十六进制并储存到字符数组result中
    return result;                            //返回结果
}
```

图 8：解密函数

1.3 RSA 算法测试

接下来笔者以一个简单例子对 RSA 算法进行测试。编写主函数如下图 9 所示：

```
int main()
{
    key_pair *p = gen_key_pair(); //生成密钥对

    cout << "n = " << p->n << endl; //输出公共模数n
    cout << "d = " << p->d << endl; //输出数论倒数d
    cout << "e = " << p->e << endl; //输出公共指数e

    char buf[KEY_LENGTH + 10];
    cout << "请输入要加密的数字，二进制长度不超过" << KEY_LENGTH << endl;
    cin >> buf; //以数字加密为例进行测试

    char *cipher_text = encrypt(buf, p->n, p->e);          //进行加密
    cout << "密文为：" << cipher_text << endl;
    char *plain_text = decrypt(cipher_text, p->n, p->d);  //进行解密
    cout << "明文为：" << plain_text << endl;

    if (strcmp(buf, plain_text) != 0)
        cout << "无法解密" << endl;
    else
        cout << "解密成功" << endl;

    return 0;
}
```

图 9：RSA 算法测试主函数

首先对 RSA.cpp 文件进行编译，然后运行编译程序，从下图 10 可以看到，RSA 算法实现了成功解密和加密。

```
~/Desktop/大三下/网络安全/谢晓锋_2018031275_EXP2_公钥密码 g++ RSA.cpp -lgmpxx -lgmp   进行编译
~/Desktop/大三下/网络安全/谢晓锋_2018031275_EXP2_公钥密码 ./a.out
n = 231723c191f3ef15f62f42787a25e597f3b1d7aab903c22abf32d9d78fe00bbe0428a9a109f29dc9a199d5b82aa987214c74b495d9e94acdb074df7a3091a3edb2b9465822767bb619326c87895a004b78411d91a68bd5080c14c4cae0c6aa588e5f4295100cb4a631b7f03798bb2f754d04f298dcdb22695ecfa48659dcfd01a87559b23a72833d633c6e2715bcc6daf3d84a9b8f83ba68b3141376f6afa790555d509cc3cdb48316138c4ed7844e0f3ef9f0ee9bec99fd12e4465914454123486 3d03d0f6dfd557207a7c0cb87a8e3cd312cd186e08da85b0c1b73392ff3aa7992beb423db465ab34cde6f5ab7ed9ab696a252a58fbaa2c9bdf0a947b5dc7
d = 1d0df3d53afc37627fccf1c4bf9ed153d7bbd14ed292821ad2e54fea647ea63a574733ecdb47bfce4bb80c75c91d457e14c82d1e46a81bffe3b361b6d2b0448374106466656 2a55c49d0ae7c2b0612686ad1bfa1f1bca64f5b7176e064ed19eee93386f5d5ac2f87cc72f8bb9002ad70faacb6124c0c4599e99f14230a152880335e223ebbda49e93884db6d5b66c6ede88813e0ee239d09cdc119d55881f1e71e639cf0a6def33cf77dd02d69c2578f9487b92f8b56589c62ba300d8360c9cacfad34468935fb7061312d0571c71d9d0bee88d7428e9695b1203dfdf9258a63c74193f63ca6b677fa347954999685f95f37f709f181cc6d251d834a061b681   输出n,d,e的信息
e = 65537
请输入要加密的数字，二进制长度不超过2048   输入要加密的数字进行测试
37826763684638
密文为：27abccaf9e9ffeaa8c9b600e5d71ec867ed9ee5084280f54f851561de00e0a8b4848f55bdf0d12a9eb56288f4987851fc8cc9ef6c4d706c035a118ad471599f7b1bee79cbd5406fbefd6ebeed389505e9b35745866aa269cd7fbd914c28e6f9d994c86ac0b5a2dba5a27ca06934c474b6b0e24b25c090263292d6013ece7b9a29659146e644fdee45aded0558a1786f1530dd56f4a459954d79fcb5212c3e3bd282d3fb20d081963e4ee5b132b53817a76ec425bed7ca1029e3eac1798b4e25f68f9dbf73d32f0f7e72eaecbb3474efd3996b625451d2f7bb4add94d86aeb309508cd5015a993e48836d6d7d40bf33d0ce4bd02497230 1402fecf117292ac 加密成功，输出密文
明文为：37826763684638 解密成功，输出明文
解密成功
```

图 10：RSA 算法测试结果

2、编程实现 RSA 公钥加密会话密钥，并应用分组密码 3DES 加密过程。

实现 RSA 算法后，可以使用 RSA 对 3DES 中的会话密钥进行加密，再用 3DES 对文件进行加密和解密。关于 RSA 和 3DES 的会话模式，笔者已经在实验原理部分的第 2 点中进行了详细阐述，具体流程见图 1，此处不再赘述。

笔者已经在第一题中实现了 RSA 加解密，且在实验一也实现了 3DES 算法的加解密操作，因此，本部分只需要对 RSA 和 3DES 的代码进行组合。笔者编写了一个主函数模拟实现 A 和 B 之间的 RSA 与 3DES 会话模式，逻辑步骤如下：

第一步：B 首先用 RSA 算法，得到公钥(n,e)和私钥(n,d)，并将公钥分发给 A（私钥只有 B 自己知道）。这样 A 把要加密的信息用这把公钥加密后发送给 B，B 用自己的私钥就可以获得加密的信息了。

```
//第一步：B首先用RSA，得到公钥(n,e)和私钥(n,d)，并将公钥分发给A（私钥只有B自己知道）
cout << "B开始使用RSA生成公钥(n,e)和私钥(n,d)" << endl;
key_pair *p = gen_key_pair(); //生成密钥对

cout << "n = " << p->n << endl; //输出公共模数n
cout << "d = " << p->d << endl; //输出数论倒数d
cout << "e = " << p->e << endl; //输出公共指数e
cout << "------------------------------------------------" << endl;
```

图 12：模拟会话模式第一步

使用 *g++ RSA_3DES.cpp -lgmpxx -lgmp* 命令编译程序，并执行编译文件，可以看到第一步的运行结果如下图 13 所示，输出了 RSA 的公钥和私钥信息。

```
~/Desktop/大二下/网络安全/谢晓锋_2018031275_EXP2_公钥密码 ./a.out
B开始使用RSA生成公钥(n,e)和私钥(n,d)
n = 7f13c70abff67eaeeebbd262e5b9b828577528d5755c2ffc8c189af85c7f152989dd78a3755d279cecc1dfc77b1e31daf37c7b5c5ca73c3a32dd2568
fb9a37edcb195f390afc8fc116ff01d364905afc0e88df16d0f4bd80afc436012a73f03a068d8f17f17a8eab9f30ce001d45a177caf090d99d423a0de518
e7864ae787b841e175b439edfff09813b7399179e8d4ea913c00d4b54e48f8f6d6f72a568a2feb44c1cdf72273844a7f081433eb0fdcbcfd362c496f227b
7406bc57ede13b5dc425049d04045b906540f4c0dc828fe7896ef12b311cf1fd57c7a48f03a3db8764de2b2031981eec98549bf22eee96f45c7d07b2e171
c0df8897ec11415a7e77
d = 4477f4839f4081ab6168038853bc8833fc435cf7fe1ee8a00b1feac7ac96b0614c2def252b4440e6b54e147397ab82333583b9b7df1820443f499631
8576e7711913f9f65cdbc9cfbed970275d2b650bf84594cd22f4dde98ea9b2e79dc54d17782365d860a0efdb60cb4b019b1abe5bef4b23f6970e8d0fcd196
64909fe535e34fe0fc864b935006952f11285ad20426002ebefa9338c8bb467ae298fbc9fd2c50815c097b316fc1f3b0fc038d2dd0ac11793898644e1539
c2f9e639ab07dff89aff3384969eafbb0f462c0620399d3309673cbf06cb9b4156eb5e51c38fa459fb5d3e46aee1b665c02fbed459dbd7d054fbacbb5987
c3c6ebb6d8c1dd100f89
e = 65537
------------------------------------------------
```

图 13：第一步运行结果

第二步：A,输入 3 个 3DES 的会话密钥，用来后面对文件进行加密。但是这个 3DES 加密后的文件发给 B，如果 B 要进行解密，也需要使用到这 3 个密钥。为了安全，下一步 A 可以用 RSA 加密这 3 个密钥，再传输给 B 进行解密。这样 A 和 B 就都拥有了 3DES 的密钥，可以进行加密文件的传输了。

```
//第二步：A,输入3个3DES的会话密钥，用来后面对文件进行加密
cout << "A收到B得公钥后，开始输入3个3DES的密钥" << endl;
type key_1[64], key_2[64], key_3[64];
cout << "请输入3DES密钥1" << endl;
cin >> key_1;
cout << "请输入3DES密钥2" << endl;
cin >> key_2;
cout << "请输入3DES密钥3" << endl;
cin >> key_3;
cout << "------------------------------------------------" << endl;
```

图 14：模拟会话模式第二步

运行结果如下图 15 所示，笔者随便了输入了 3 个密钥。

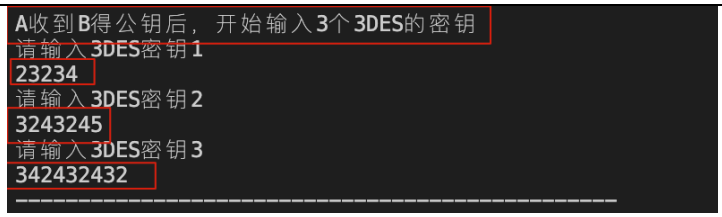


图 15：第二步运行结果

第三步：A 将该会话密钥(3DES 的 3 个密码)使用 RSA 公钥加密，并且将密文直接发送给 B。

```
//第三步：A将该会话密钥(3DES的3个密码)使用RSA公钥加密，并且将密文直接发送给B
cout << "A开始使用B的RSA公钥加密3DES的3个密码,并将密文发给B" << endl;
type *key_1_en = encrypt(key_1, p->n, p->e);
type *key_2_en = encrypt(key_2, p->n, p->e);
type *key_3_en = encrypt(key_3, p->n, p->e);
cout << "加密后的3DES密钥1" << key_1_en << endl;
cout << "加密后的3DES密钥2" << key_2_en << endl;
cout << "加密后的3DES密钥3" << key_3_en << endl;
cout << "----------------------------------------------" << endl;
```

图 16：模拟会话模式第三步

运行结果如下图 17 所示，使用 RSA 完成了对 3DES 的 3 个密码进行加密。

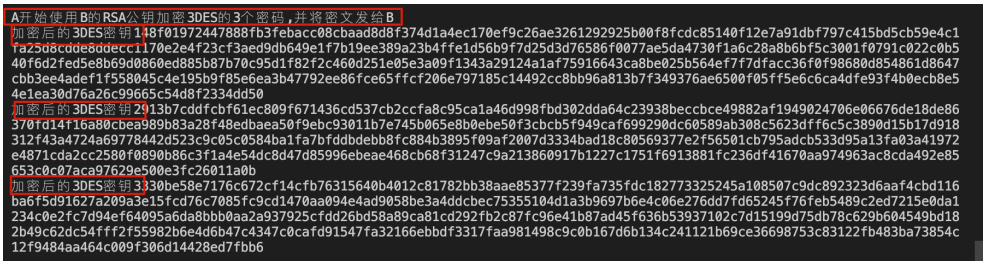


图 17：第三步运行结果

第四步：B 收到 A 发来的 3 个 3DES 密钥的密文后，使用自己的 RSA 私钥进行解密。

```
//第四步：B收到A发来的3个3DES密钥后，使用自己的RSA私钥进行解密
cout << "B收到A发来的3个3DES密钥后，使用自己的RSA私钥进行解密" << endl;
type *key_1_de = decrypt(key_1_en, p->n, p->d);
type *key_2_de = decrypt(key_2_en, p->n, p->d);
type *key_3_de = decrypt(key_3_en, p->n, p->d);
cout << "解密后的3DES密钥1" << key_1_de << endl;
cout << "解密后的3DES密钥2" << key_2_de << endl;
cout << "解密后的3DES密钥3" << key_3_de << endl;
cout << "----------------------------------------------" << endl;
```

图 18：模拟会话模式第四步

运行结果如下图 19 所示，可以看到 B 解密后的 3DES 密钥跟 A 刚开始设置的密钥是一致的。到这里 A 和 B 就都拥有了 3DES 的密钥串，都可以进行加密解密从而传输文件了。

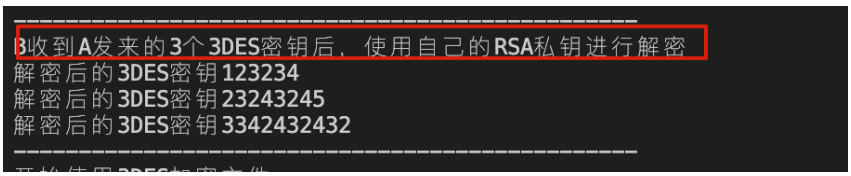


图 19：第四步运行结果

第五步：A 使用 3DES 加密一个图片文件（*testJPG_org.jpg*），并将加密后的文件（*testJPG_Encode.jpg*）发给 B。

```
//现在A、B都已经通过RSA机制安全拥有了3DES的3个密钥，都可以进行加密解密从而传输文件了
//第五步：A加密一个图片文件，并将加密后的文件发给B
Treble_DES d;
cout << "开始使用3DES加密文件" << endl;
cout << d.treble_DES_Encrypt("testJPG_org.jpg", key_1, key_2, key_3, "testJPG_Encode.jpg");
cout << endl;
cout << "加密结束，并将加密后的文件发送给对方" << endl;
```

图 20：模拟会话模式第五步

运行结果如下图 21 所示（输出 1 表示 3DES 加密程序运行成功），可以看到 jpg 文件成功进行了加密。

```
--------------------------------------------------
开始使用3DES加密文件
1
加密结束，并将加密后的文件发送给对方
```

图 21：第五步运行结果

第六步：B 收到加密后的文件后，同样使用 3DES 的密钥进行解密（*testJPG_Decode.jpg*）。

```
//第六步：B收到加密后的文件后，同样使用3DES的密钥进行解密
cout << "开始使用3DES解密文件" << endl;
cout << d.treble_DES_Decrypt("testJPG_Encode.jpg", key_1_de, key_2_de, key_3_de, "testJPG_Decode.jpg");
cout << endl;
cout << "解密完成，完成加密文件传输流程" << endl;
```

图 22：模拟会话模式第六步

运行结果如下图 23 所示（输出 1 表示 3DES 解密程序运行成功），可以看到 jpg 文件成功进行了解密。

```
开始使用3DES解密文件
1
解密完成，完成加密文件传输流程
~/Desktop/大三下/网络安全/谢晓锋_2018031275_EXP2_公钥密码
```

图 23：第六步运行结果

到这里就完成了 A 和 B 之间的 RSA 与 3DES 会话模式模拟。接下来查看刚刚加解密的图片文件，如下图 24、25、26 所示。说明对 jpg 文件的加密和解密都成功了，也说明了 RSA 与 3DES 会话模式的成功进行。

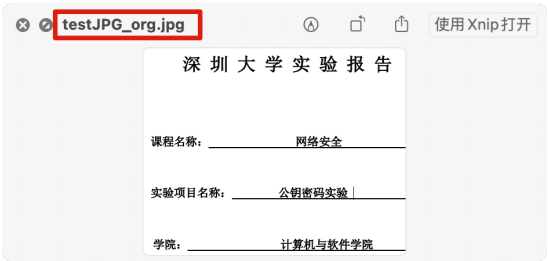


图 24: testJPG_org.jpg 文件

图 25: testJPG_Encode.jpg 文件（打不开）

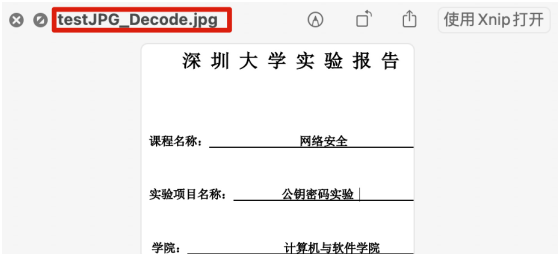


图 26: testJPG_Decode.jpg 文件

实验结论：

通过本次实验，笔者理解了公钥密码的基本思想以及理解 RSA 和数字签名的原理，掌握了 RSA 和数字签名的输入输出格式和密钥格式，同时，通过与之前实验相结合，实现了使用 RSA 算法公钥加密会话密钥，然后使用会话密钥和 3DES 加解密各种文件。除了巩固课堂中所学的知识的同时，也学到了一些编程技巧，比说，C++中 GMP 库函数的使用等。

RSA 算法基于分解大整数的时间消耗，从而保证安全。因此实现 RSA 算法需要用到很多的数学知识，例如拓展的欧拉公式、米勒-拉宾素数检验等，这样才有利于降低算法的复杂度。但即使如此，RSA 算法由于其大数运算，与 3DES 等对称加密算法相比，在加密信息时其加密时间仍然较高。因此 RSA 算法常常只是作为会话秘钥的加密，而不加密文本信息，本实验充分展示了这一点。

RSA 算法除了做加密还可以进行数字签名的认证，即加密者用自己的私钥加密文档，其他人可以用他的公钥对该文档进行认证。有了这层属性，RSA 在现代社会也应用的更加广泛。

指导教师批阅意见： 成绩评定： 指导教师签字： 年　　月　　日
备注：

注：1、报告内的项目或内容设置，可根据实际情况加以调整和补充。

2、教师批改学生实验报告时间应在学生提交实验报告时间后 10 日内。