

同济大学电信学院

密码学课程设计实验报告



目录

- 一、项目说明3
- 二、各部分介绍.....4
 - 1.DES 设计介绍.....4
 - 2.AES 设计介绍.....8
 - 3.RSA 设计说明.....10
 - 4.文件加密实现.....13
 - 5.额外功能介绍.....18
- 三、结果演示19
 - 1.对 txt 文本进行加密：19
 - 2.对其他格式的文件进行加密22
 - 3.对当前目录的所有文件进行加密并替换23
- 四、心得.....26

一、项目说明

本程序使用 ANSI X9.17 算法来生成随机数，其中包含了 DES 的实现，使用了 AES 对文件进行加密、解密，后面又实现了 RSA 对 AES 密钥的加密、解密。程序有如下的功能：



1 是产生新的 RSA 的公钥和私钥，可以选择产生 512 位的还是 1024 位的，如下：



2 是加密输入文件，对于密钥是通过读取文件获得，对于要加密的内容也是通过文件读取来完成，加密的结果都存放在另外的文件中，文件名都是在相应的内容前加上加密_。

3 是解密文件，这里模拟了通信过程，所以假设对方只有加密的文件和加密的 AES 密钥，所以需要输入对应的 AES 密钥加密后的文件名以及需要解密的文件的文件名。获得解密的后的 AES 密钥以及解密后的文件都存放在另外的文件中，文件名都是在相应的内容前加上解密_。

5 是一个衍生的内容，这里选择运行了之后**直接将当前目录下所有的文件都加密**，并将内容进行替换，当然对于程序运行时要进行读取密钥相关的文件（AES 密钥.txt、公钥.txt、加密_AES 密钥.txt 和私钥.txt、SRA 课设.exe），都没有进行加密了，能用的前提是当前目录下有 AES 密钥.txt 和公钥.txt 这 2 个文件，因为都是直接默认读取他们。**运行时请保证当前目录下没有关键的文件，不然怕解密不出来，虽然我这里测试过了都能正确解密，但还是小心为好。**

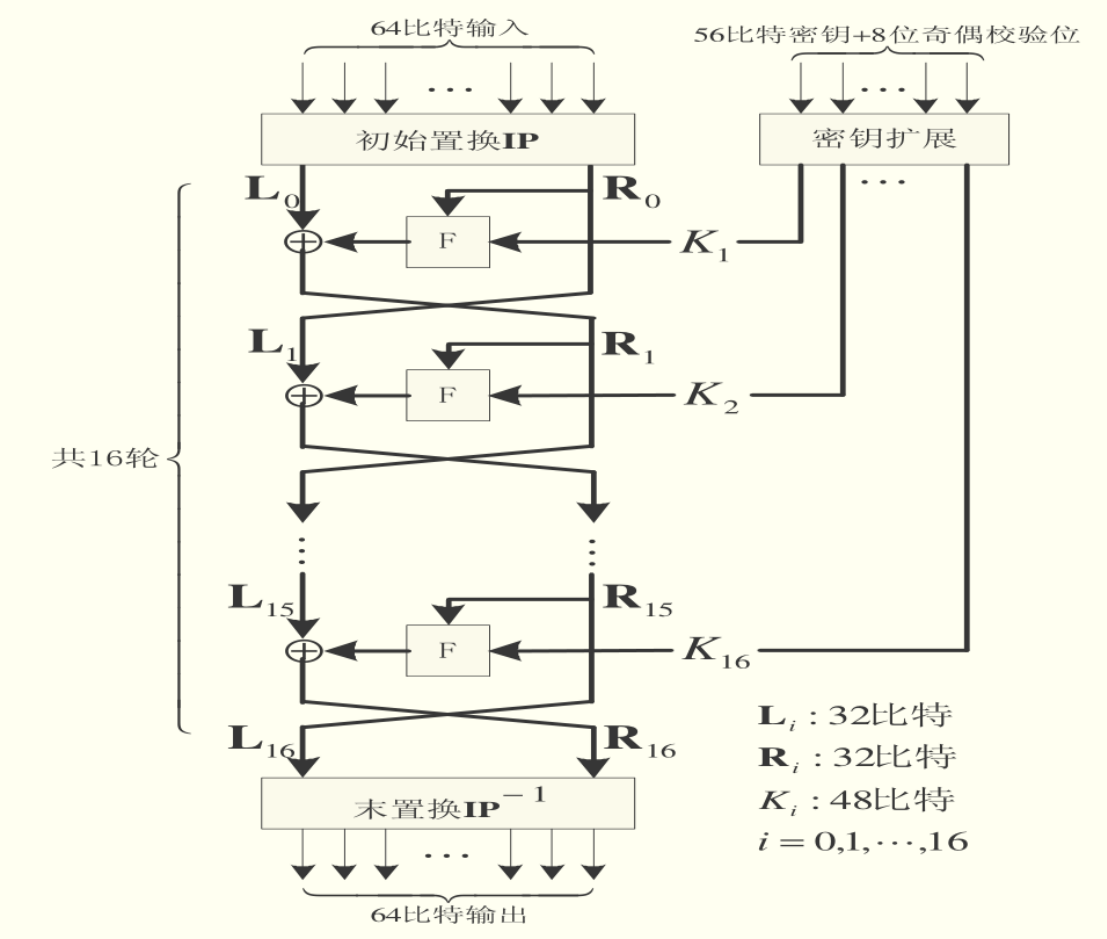
6 是 5 对应的内容，将当前目录下所有被 5 加密的文件都进行解密，能正确解密的前提是有对应的加密_AES 密钥.txt 和私钥.txt，这 2 个文件，因为这里也都是直接默认读取他们，没有进行额外的适配了。

tip: 这里所有的输入都设置了默认输入，如程序上所言，所以在放置好了文件之后，直接一路按回车就可以进行加密解密等等运算了，当然也可以自己手动输入文件名。注意使用之前需要生成公钥和私钥。

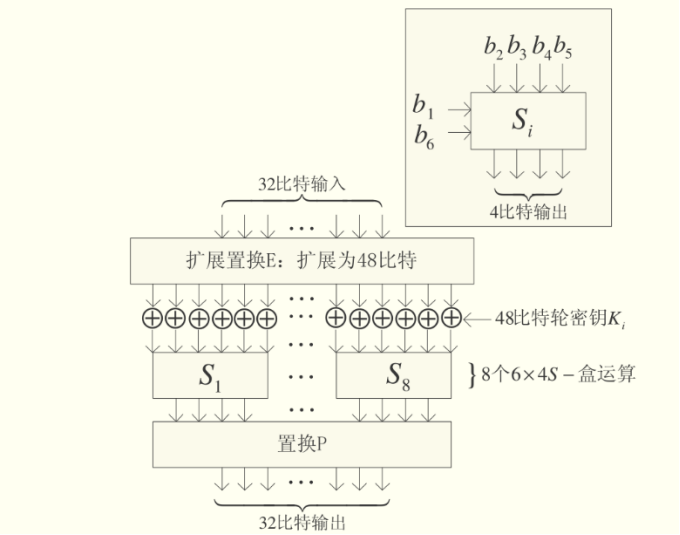
二、各部分介绍

1.DES 设计介绍

DES 的原理设计图如下：



F函数：



其中的F函数的实现是如下进行的：

将拓展、异或、S盒运算、IP盒置换通过输入输出合并起来

```
bitset<32> DES::f(bitset<32>a, bitset<48> key)//F函数，整合了上面的一些运算
{
    bitset<48>ex_48, xor_res;
    bitset<32>S_res, f_out;
    ex_48 = T0_48(a);
    xor_res = XOR(ex_48, key);
    S_res = S_change(xor_res);
    f_out = IP_change2(S_res);
    /*cout << "拓展到48位: ";
    for (int i = 0; i < 48; i++)
        cout << ex_48[i];
    cout << endl;
    cout << "拓展之后异或的结果: ";
    for (int i = 0; i < 48; i++)
        cout << xor_res[i];
    cout << endl;
    cout << "S盒操作完后: ";
    for (int i = 0; i < 32; i++)
        cout << S_res[i];
    cout << endl;
    cout << "再次IP置换后: ";
    for (int i = 0; i < 32; i++)
        cout << f_out[i];
    cout << endl;*/
    return f_out;
}
```

使用一个密钥进行一次变化的安排如下：

```
bitset<64> DES::once(bitset<64> in, bitset<48>key, int n)//对in使用key进行一轮的完整加密，加密结果返回
{
    bitset<32> left_txt;
    bitset<32> right_txt;
    bitset<32>f_out, xor_res;
    bitset<64>res;
    separate(in, left_txt, right_txt);
    if (!is_decode || n == 15)//注意最后一轮有所不同
    {
        f_out = f(right_txt, key);
        xor_res = xor_32(left_txt, f_out);
        res = merge_once_res(right_txt, xor_res, n);
        return res;
    }
    else
    {
        f_out = f(left_txt, key);
        xor_res = xor_32(right_txt, f_out);
        res = merge_once_res(xor_res, left_txt, n);
        return res;
    }
}
```

进行一轮完整的DES加密，即进行16轮小的加密如下：

```
void DES::oper_16()//操作16次，进行完整加密
{
    bitset<64>temp;
    temp = IP_change(txt);
    show_bit(temp);
    for (int i = 0; i < 16; i++)
    {
        temp = once(temp, r_son_key[i], i);
    }
    des_res = IP_1_change(temp);
}
```

进行 DES 解密如下：

```
void DES::decode()//解密
{
    is_decode = 1;
    bitset<64> temp;
    temp = IP_1_change(c_txt);
    for (int i = 15; i >= 0; i--)
    {
        temp = once(temp, r_son_key[i], i);
    }
    decode_res = IP_change(temp);
}
```

按照 ANSI X9.17 伪随机数生成的算法，生成 m 长度的随机数如下进行：
先是 3 轮的 DES 交替加密解密：

```
bitset<64> three_des(bitset<64> in, DES des1, DES des2)//3轮交替的des加密、解密
{
    des1.get_txt(in);
    des1.oper_16();

    des2.get_ctxt(des1.des_res);
    des2.decode();

    des1.get_txt(des2.decode_res);
    des1.oper_16();

    return des1.des_res;
}
```

然后是组合在一起，生成随机的 bit 组合（其中密钥是直接写死了，依靠时间产生随机 bit，结果存放在 x 数组中）

```
bitset<64> key_time = StrToBit(local_time);

DES des1;
des1.get_key(or_key1);
des1.get_all_key();
DES des2;
des2.get_key(or_key2);
des2.get_all_key();

bitset<64> I = three_des(key_time, des1, des2);
bitset<64> I_xor_s;
for (int k = 0; k < m; k++)
{
    I_xor_s = axor(I, s);
    bitset<64> xt = three_des(I_xor_s, des1, des2);
    x[k] = xt;
    bitset<64> x_xor_I = axor(xt, I);
    s = three_des(x_xor_I, des1, des2);
}
```


最后就得到我们需要的 nbit 的随机数

```
ZZ random_ZZ(int n)//获得对应n个bit大小的随机数
{
    ZZ res = ZZ(0), t = ZZ(1);
    int m = n / 64;
    bitset<64>* x = new bitset<64>[m];
    des_opera(x, m);
    for (int i = 0; i < m; i++)
    {
        for (int j = 0; j < 64; j++)
        {
            if (x[i][j] == 1)
            {
                res += t;
            }
            t = t << 1;
        }
    }
    return res;
}
```

这是 DES 中用到的所有函数及功能说明：

```
class DES
{
private:
    bitset<64> txt;
    bitset<64> s_key;
    bitset<48> r_son_key[16];
    bitset<32> left_txt;
    bitset<32> right_txt;
    bitset<48> ex_txt;
    bitset<64> once_res;
    bitset<64> c_txt;
    bool is_decode = 0;

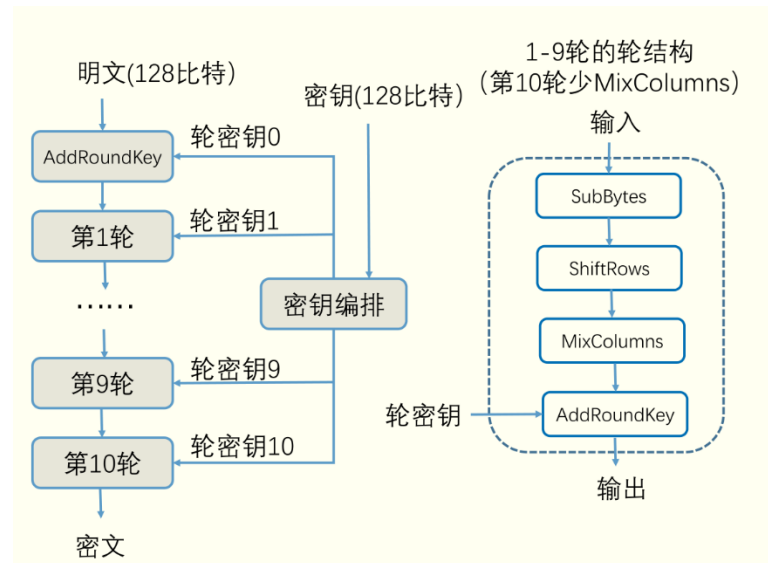
    bitset<56> key_IP_change(bitset<64>in);//密钥经过IP盒变化
    void get_key_CD(bitset<56> son_key, bitset<28> key_C[17], bitset<28> key_D[17]);//获得密钥的分组CD
    void get_16_son_key_48(bitset<28> key_C[17], bitset<28> key_D[17]);//得到16个48长度的密钥
    bitset<64> IP_change(bitset<64> in);//对输入进行IP盒变换
    bitset<64> IP_1_change(bitset<64> in);//进行另一种IP变换
    void separate(bitset<64>in, bitset<32>& l_txt, bitset<32>& r_txt);//将in对半分成分l_txt和r_txt
    bitset<48> TO_48(bitset<32> right_txt);//将32位的输入变为48位
    bitset<48> XOR(bitset<48>ex_48, bitset<48> key);//48位的进行异或
    bitset<32> S_change(bitset<48>xor_res);//S盒变换
    bitset<32> IP_change2(bitset<32> S_res);//另一种IP变换
    bitset<32> f(bitset<32>R, bitset<48> K);//F函数，整合了一些运算
    bitset<32> xor_32(bitset<32> a, bitset<32> b);//32位bit的数据进行异或
    bitset<64> merge_once_res(bitset<32>right_txt, bitset<32>&f_out, int n);//将输入进行拼接
    bitset<64> once(bitset<64> in, bitset<48>key, int n);//对in使用key进行一轮的完整加密，加密结果返回

public:
    bitset<64> des_res;
    bitset<64> decode_res;

    void get_key(bitset<64> in);//获得密钥
    void get_all_key();//得到所有的密钥
    void get_txt(bitset<64> in);//输入加密的txt
    void oper_16();//操作16次，进行完整加密
    void get_ctxt(bitset<64> in);//输入密文
    void decode();//解密
};
```

2.AES 设计介绍

AES 的设计原理图如下：



其中一轮的设计如下：

```
void AES::once(byte in[16], byte out[16], int w_n)
{
    byte Sub_res[16], Shift_res[16], Mix_res[16];
    SubBytes(in, Sub_res);

    ShiftRows(Sub_res, Shift_res);

    MixColumns(Shift_res, Mix_res);

    AddRoundKey(Mix_res, out, w_n);
}
```

AES 加密的设计如下：

进行 9 轮上述的操作，然后加上最后一轮的操作，即少一个 MixColumns。


```

void AES::aes(byte in[16], byte key[16], byte out[16])
{
    get_txt(in);
    get_key(key);
    byte temp[16], temp_in[16], temp_out[16];
    AddRoundKey(txt, temp, 0);
    for (int i = 1; i <= 9; i++)
    {
        once(temp, temp, i);
    }
    byte Sub_res[16], Shift_res[16];
    SubBytes(temp, Sub_res);

    ShiftRows(Sub_res, Shift_res);

    AddRoundKey(Shift_res, out, 10);
}

```

AES 的解密设计如下，解密时，每一轮的函数顺序与加密有些不同，函数也是重新写的，但是差别不大：

```

void AES::decode_once(byte in[16], byte out[16], int w_n)
{
    byte Sub_res[16], Shift_res[16], Add_res[16];
    InvShiftRows(in, Shift_res);

    InvSubBytes(Shift_res, Sub_res);

    AddRoundKey(Sub_res, Add_res, w_n);

    InvMixColumns(Add_res, out);
}

```

```

void AES::aes_decode(byte in[16], byte key[16], byte out[16])
{
    get_txt(in);
    get_key(key);
    byte temp[16], temp_in[16], temp_out[16];
    AddRoundKey(txt, temp, 10);
    for (int i = 9; i >= 1; i--)
    {
        decode_once(temp, temp, i);
    }
    byte Sub_res[16], Shift_res[16];
    InvShiftRows(temp, Shift_res);

    InvSubBytes(Shift_res, Sub_res);

    AddRoundKey(Sub_res, out, 0);
}

```

AES 整体所用运用到的函数：

```
class AES
{
private:
    byte or_key[16];
    byte w[11][4][4];
    byte txt[16];
    byte S_change(byte in, bool is_encry); //具体的S盒变换
    void get_txt(byte in[16]); //获得需要加密txt
    void get_key(byte in[16]); //得到密钥
    void AddRoundKey(byte in[16], byte out[16], int w_n); //轮密钥异或
    void SubBytes(byte in[16], byte out[16]); //S盒操作
    void ShiftRows(byte in[16], byte out[16]); //行移位
    byte GFmul(byte a, byte b); //
    void MixColumns(byte in[16], byte out[16]); //列混合
    void once(byte in[16], byte out[16], int w_n); //一轮操作
    void InvShiftRows(byte in[16], byte out[16]); //解密用的行移位
    void InvSubBytes(byte in[16], byte out[16]); //解密用的S盒操作
    void InvMixColumns(byte in[16], byte out[16]); //解密用的列混合
    void decode_once(byte in[16], byte out[16], int w_n); //解密的一轮操作
public:
    void aes(byte in[16], byte key[16], byte out[16]); //aes加密
    void aes_decode(byte in[16], byte key[16], byte out[16]); //解密
};
```

3.RSA 设计说明

RSA 的设计原理如下：

RSA密码体制		
		备注
公开密钥(n, b)	$n = pq$	$\phi(n)$
加密指数 b	b 与 $(p-1)(q-1)$ 互素	$= (p-1)(q-1)$
私钥(a, p, q)	p, q 是素数	$ab \equiv 1 \pmod{\phi(n)}$
解密指数 a	$a = b^{-1} \pmod{(p-1)(q-1)}$	要求 $b, \phi(n)$ 互素
加密 $e_K(x)$	$y = x^b \pmod n$	明文 $x \in \mathbb{Z}_n$
解密 $d_K(y)$	$x = y^a \pmod n$	密文 $y \in \mathbb{Z}_n$

获取素数 p 、 q ，需要进行随机数进行检测，这里我选择的是 Solovay-Strassen 素性检测算法，用到了之前的随机数生成函数，一个奇合数通过 Solovay-Strassen 的概率是 $1/2$ ，为了保证可靠性，这里对于每一个数都检测它对应长度的次数，512 位就检测 512 次，1024 位就检测 1024 次，如下：

```

ZZ get_prim(int Z_n)
{
    cout << "查找素数中..." << endl;
    ZZ n;
    int find_sum = 0;
    while (1)
    {
        bool is_find = true;
        n = random_ZZ(Z_n);
        if (n % 2 == 0 || n % 5 == 0)
            continue;
        find_sum++;
        if (find_sum % 10 == 0 && find_sum != 0)
            cout << "已查找随机数" << find_sum << "次" << endl;
        for (int i = 0; i < Z_n; i++)
        {
            ZZ b;
            while (1)
            {
                b = (random_ZZ(Z_n)) % n;
                if (b > 0)
                    break;
            }

            ZZ j = get_j(b, (n - 1) / 2, n);
            if (j != 1 && j != n - 1)
            {
                is_find = false;
                break;
            }
            int jac_res = jacobi(b, n);
            if ((n + jac_res) % n != j)
            {
                is_find = false;
                break;
            }
        }
        if (is_find)
        {
            cout << "找到一个素数!" << endl;
            return n;
        }
    }
}

```

获得 a , b 需要 b 和 $\phi(n)$ 互素, 然后还需要 $a*b=1 \pmod{\phi(n)}$, 所以这里直接使用拓展欧几里得算法, 如果生成 b 和 $\phi(n)$ 的公因数是 1, 就可以直接得出它的逆元 a 。如下:

```

void Euclidean(ZZ a, ZZ b, ZZ &gcd, ZZ &t) //gcd是a和b的公因数，如果gcd=1, t就是b的逆元
{
    ZZ a0 = a;
    if (b > a)
    {
        ZZ temp = a;
        a = b;
        b = temp;
    }
    ZZ t0 = ZZ(0);
    t = ZZ(1);
    ZZ q = a / b;
    gcd = a - q * b;
    ZZ temp;
    while (gcd > ZZ(0))
    {
        temp = (t0 - q * t) % a0;
        t0 = t;
        t = temp;
        a = b;
        b = gcd;
        q = a / b;
        gcd = a - q * b;
    }
    gcd = b;
}

```

对幂运算需要用平方乘算法, 将计算复杂度从 $O(n)$ 降低到 $O(\log n)$, 快了太多了!

```

ZZ Z_mul(ZZ b, ZZ c, ZZ n) //返回b的c次方模n
{
    if (c == 0)
        return ZZ(1);
    ZZ flag = ZZ(1), temp;
    int Z_c = get_Z_n(c);
    //cout << "Z_n:" << Z_n << endl;
    ZZ* t = new ZZ[Z_c];
    ZZ k = b;
    t[0] = k;
    for (int i = 1; i < Z_c; i++)
    {
        t[i] = (t[i - 1] * t[i - 1]) % n;
    }
    ZZ sum = ZZ(1);
    temp = c;
    for (int i = 0; i < Z_c; i++)
    {
        //cout << i << ":" << temp << endl;
        ZZ f = temp % ZZ(2);
        if (f == ZZ(1))
        {
            sum = (sum * t[i]) % n;
        }
        temp = temp >> 1;
    }

    return sum;
}

```


最后将其组合在一起就可以获得一组我们需要的 RSA 对应的公钥和私钥，如下：

```
void fileEncry::get_aRSA(int Z_n)//获得一组RSA密钥
{
    ZZ p = get_prim(Z_n);
    ZZ q = get_prim(Z_n);
    ZZ n = p * q;
    ZZ f_n = (p - 1) * (q - 1);
    ZZ gcd = ZZ(0), a, b;
    while (gcd != ZZ(1))
    {
        b = (random_ZZ(Z_n)) % f_n;
        Euclidean(f_n, b, gcd, a);
    }
    ofstream ou("公钥.txt", ios::out);
    ou << "n=" << endl << n << endl << "b=" << endl << b;
    ou.close();
    ofstream fou("私钥.txt", ios::out);
    fou << "n=" << endl << n << endl << "a=" << endl << a << endl << "p=" << endl << p << endl << "q=" << endl << q;
    fou.close();
}
```

4.文件加密实现

在以上的程序都准备并且测试无误之后就可以进行组合，得到我们需要的功能，实现的任务如下：

Bob加密文件 m 并发给Alice:

- ① 输入文件 m ， m 为任意长。如何把文件转成所需要的格式，请自行决定。CBC模式中，如果 m 的长度不是分组的倍数时，需要填充，如何填充请自行决定，但解密时需要把填充去掉。
- ② 生成128比特的随机数 k 作为临时的会话密钥。
- ③ 用Alice的公钥加密 k 得到 $c_1 \leftarrow k^b \bmod n$ 。
- ④ 用会话密钥 k 加密文件 m 得到 c_2 ，其中加密算法为AES，使用CBC模式。

要求 AES的列混合运算及其逆运算使用我的课件中提供的快速算法。S-Box使用查表方式实现。AES的测试数据可参考AES的flash动画。

- ⑤ 把 (c_1, c_2) 发给Alice。

Alice解密恢复文件 m :

- ① 用Alice的RSA私钥解密 c_1 得到 k 。
- ② 用 k 解密 c_2 得到 m 。
- ③ 输出 m 。

对于文件，我们直接用二进制打开，获取文件的大小之后，直接整块进行读取，这样减少了 10 次数，使得文件读取更快，而且对于任意的文件都可以进行加密。填充的时候，需要先计算离形成 16 的倍数还差几个数，若差 x 个，则在后面添加 x 个 char (x)，如果 $x=0$ ，则在后面加入 16 个 char (0)。程序如下：

```
void fileEncry::get_txt(string file_name)
{
    len = 0;
    fstream in(file_name, ios::in | ios::binary);
    while (!in)
    {
        cout << "没有找到要加密的文件，请重新输入" << endl;
        cin >> file_name;
        in.open(file_name, ios::in | ios::binary);
    }
    in.seekg(0, std::ios::end);
    len = in.tellg();
    in.seekg(0, std::ios::beg);
    txt = new char[len + 20];
    in.read(txt, len);
    in.close();
    int temp = 16 - len % 16;
    if (len % 16 == 0)
    {
        for (int i = 0; i < 16; i++)
            txt[len++] = char(0);
    }
    else
    {
        while (len % 16 != 0)
        {
            txt[len++] = char(temp);
        }
    }
    txt[len] = '\0';
}
```

生成随机数 K 当做 AES 的密钥，同时用公钥 b 来对其进行加密：


```

ZZ fileEncry::encry_key(string key_file_name)//读取文件，获取密钥，加密密钥
{
    get_fkey(key_file_name);
    fstream in("公钥.txt", ios::in);
    if (!in)
    {
        cout << "没有找到公钥文件，请确认有无生成公钥.txt" << endl;
        char t = _getch();
        exit(0);
    }

    string t;
    in >> t;
    in >> n;
    in >> t;
    in >> b;
    ZZ res = ZZ(0), temp = ZZ(1);
    for (int i = 0; i < 16; i++)
    {
        res += temp * fkey[i];
        temp <<= 8;
    }
    ZZ y = Z_mul(res, b, n);
    string name = "加密_";
    name = name + key_file_name;
    fstream out(name, ios::out);
    out << y;
    return y;
}

```

在用前面生成的 AES 密钥来对文件进行 AES 加密，采用 CBC 模式

```

void fileEncry::encrye_txt(string file_name, string key_file_name)
{
    get_txt(file_name);
    encry_key(key_file_name);
    string name = "加密_";
    name = name + file_name;
    fstream fout(name, ios::out | ios::binary);
    AES aes;
    byte p[16], out[16];
    byte y[16];
    for (int i = 0; i < 16; i++)
        y[i] = byte(0);
    for (int i = 0; i < len / 16; i++)
    {
        for (int j = 0; j < 16; j++)
        {
            p[j] = txt[i * 16 + j] ^ y[j];
        }
        aes.aes(p, fkey, out);
        char s[16] = { '0', '1', '2', '3', '4', '5', '6', '7', '8', '9', 'a', 'b', 'c', 'd', 'e', 'f' };
        for (int i = 0; i < 16 * 2; i++)
        {
            if (i % 2 == 0)
            {
                int temp = unsigned int(out[i / 2]) >> 4;
                fout << s[temp];
            }
            else
            {
                int temp = unsigned int(out[i / 2]) % 16;
                fout << s[temp];
            }
        }
        for (int i = 0; i < 16; i++)//CBC
            y[i] = out[i];
    }
    fout.close();
}

```

解密时，先用私钥 a 来解密 AES 的密钥，如下

```
void fileEncry::decry_key(string key_file_name)//读取私钥，解密出AES密钥
{
    ZZ y;
    fstream yin(key_file_name, ios::in);
    while (!yin)
    {
        cout << "没有找到要解密的AES密钥文件，请重新输入" << endl;
        cin >> key_file_name;
        yin.open(key_file_name, ios::in);
    }
    yin >> y;
    yin.close();
    fstream in("私钥.txt", ios::in);
    if (!in)
    {
        cout << "没有找到私钥文件，请确认有无生成私钥.txt" << endl;

        char t = _getch();
        exit(0);
    }
    string t;
    in >> t;
    in >> n;
    in >> t;
    in >> a;
    ZZ x;
    x = rsa_jiemi(y, a, n);
    //cout << endl << x << endl;
    int temp = 256;
    for (int i = 0; i < 16; i++)
    {
        fkey[i] = byte(x % temp);
        x = x >> 8;
    }
}
```



用解密出来的 AES 密钥来对文件进行解密，如下：

```
void fileEncry::decry_txt(string file_name, string key_file_name)//使用AES密钥对文件进行解密
{
    decry_get_txt(file_name);
    decry_key(key_file_name);
    string name = "解密_";
    name = name + file_name;
    fstream fout(name, ios::out | ios::binary);
    AES aes;
    byte y[16];
    for (int i = 0; i < 16; i++)
        y[i] = byte(0);
    for (int j = 0; j < len / 32; j++)
    {
        byte temp[16], res[16];
        for (int k = 0; k < 16; k++)
        {
            temp[k] = encry_txt[j * 16 + k];
        }
        aes.aes_decode(temp, fkey, res);
        for (int i = 0; i < 16; i++)
        {
            res[i] = res[i] ^ y[i];
            y[i] = temp[i];
        }
        if (j == len / 32 - 1)
        {
            int last_i = res[15];
            if (last_i == 0)
                break;
            else
            {
                for (int k = 0; k < 16 - last_i; k++)
                {
                    fout.put(res[k]);
                }
            }
        }
        else
        {
            for (int k = 0; k < 16; k++)
            {
                fout.put(res[k]);
            }
        }
    }
    fout.close();
}
```

所有的函数如下：

```
class fileEncry
{
private:
    char* txt;
    byte* encry_txt;
    int len;
    ZZ p, q, n, f_n, a, b;
    byte fkey[16];
    void get_txt(string file_name);//得到需要加密的文件
    void decry_get_txt(string file_name);//得到需要解密的文件
    void get_fkey(string key_file_name)//得到使用的AES密钥
    void decry_key(string key_file_name)//用私钥解密AES
    ZZ encry_key(string key_file_name)//加密AES的密钥
public:
    void get_aRSA(int Z_n = 512);//得到一组RSA密钥
    void encrye_txt(string file_name, string key_file_name);//加密文件，加密的文件在另一个文件中
    void decry_txt(string file_name, string key_file_name);//解密文件，解密的文件在另一个文件中
    void re_encrye_txt(string file_name, string key_file_name);//加密文件，并将原文件覆盖
    void re_decry_txt(string file_name, string key_file_name);//解密文件，并将原文件覆盖
};
```

5.额外功能介绍

还实现了一个额外的功能，即遍历得到当前的目录的所有文件名，然后对所有文件进行加密，并替代这些内容，这样就可以完成对其文件内容的破坏，然后又实现了对这些加密的文件的解密复原，以此来模拟那些勒索病毒，对文件进行加密后，需要有私钥才可以进行解密，但是因为加密解密都是用的同一个 AES 密钥，而这一个密钥就也会被对方掌握，所以还是能够自己解密出来的，只是会比较麻烦，理想的应该是将所有文件进行 RSA 加密，就用公钥 b, n 对其加密，私钥 a 自己拿着，不过也没有专门写这个程序了，就用这个 AES 加密来模拟一下了。

遍历获得当前目录的文件：

```
void get_dir(string path, string sdir[], int &len) //遍历得到path下的所有文件，将文件名和数量返回
{
    long hFile = 0;
    string copypath = path;
    struct _finddata_t fileInfo;
    string pathName, exdName;

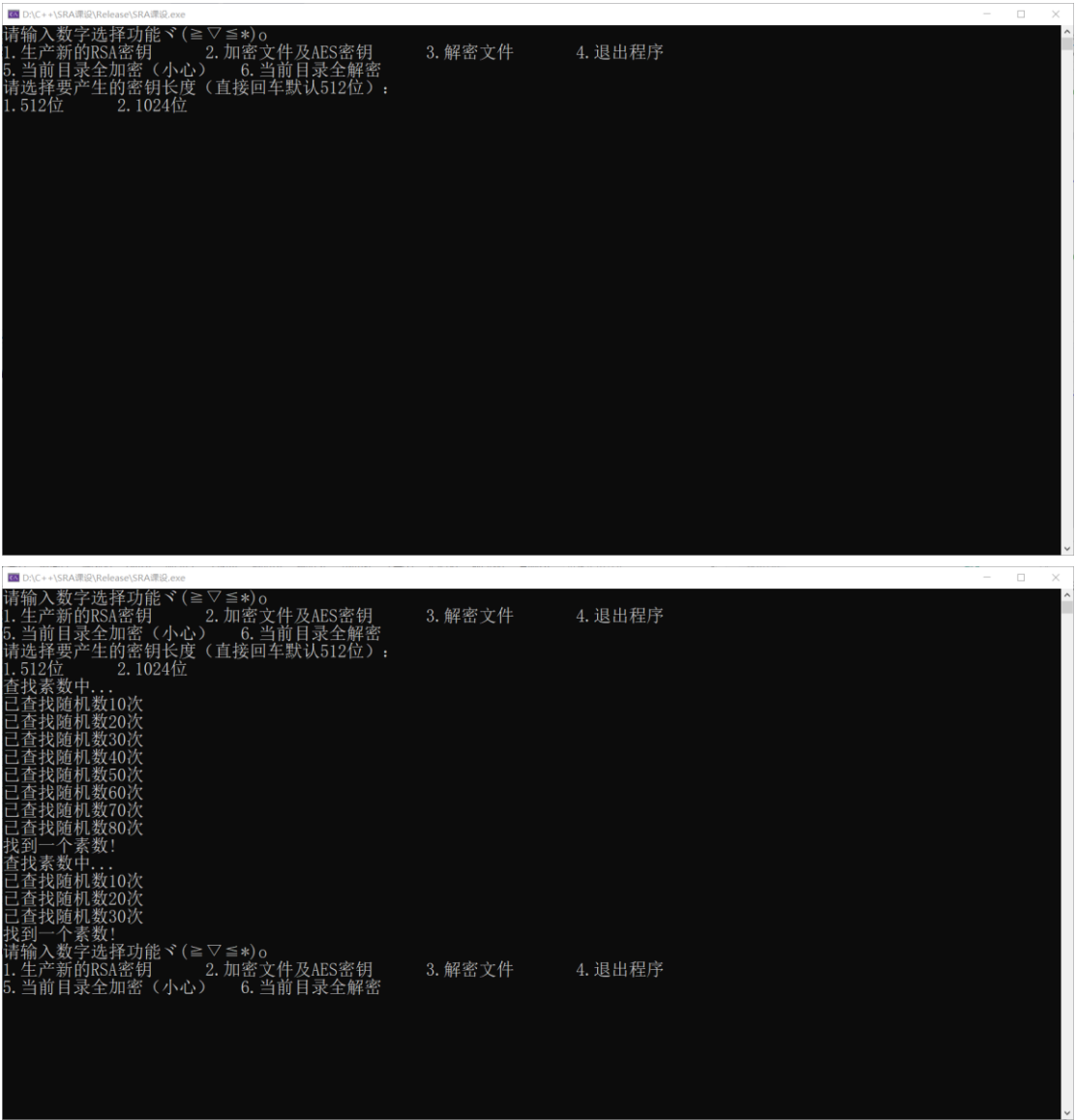
    if ((hFile = _findfirst(pathName.assign(path).append("\\*").c_str(), &fileInfo)) == -1) {
        return;
    }
    do
    {
        if (!(fileInfo.attrib & _A_SUBDIR)) //是文件属性的才加入数组
        {
            //cout << fileInfo.name << endl;
            sdir[len++] = copypath + "\\ " + fileInfo.name;
        }
    } while (_findnext(hFile, &fileInfo) == 0);
    _findclose(hFile);
    return;
}
```

加密后对原内容进行覆盖其实就是将文件打开后重新填入加密后的内容，解密也是同理，不再赘述。

三、结果演示

1.对 txt 文本进行加密：

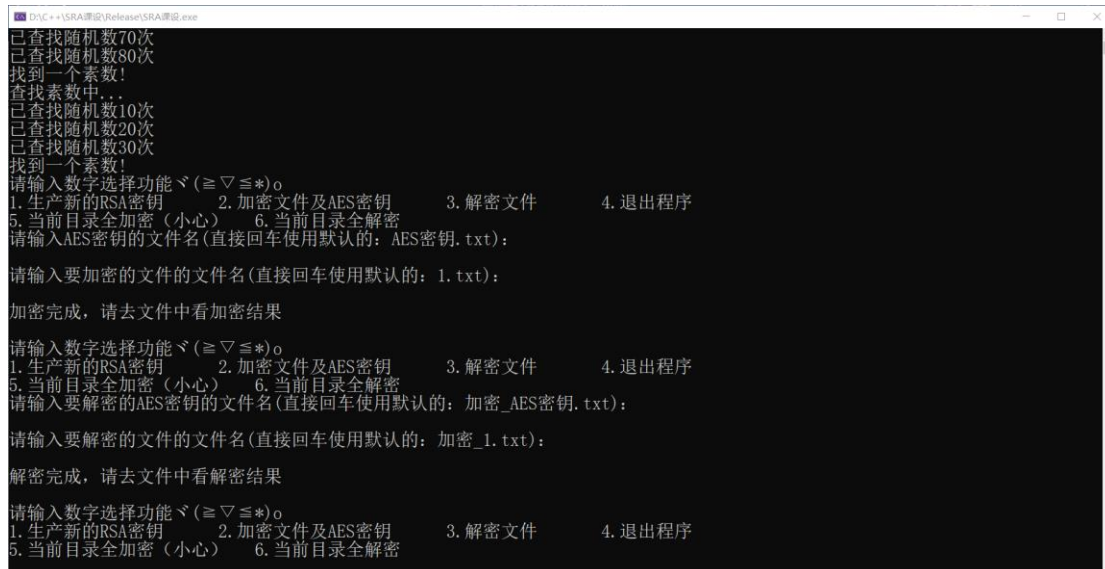
用程序获取 RSA 对应的私钥公钥存入文件中，并自己设置 AES 密钥，AES 密钥可以是任意数字或者字符，但是私钥公钥文件必须由 exe 生成或者按照其固定格式写入：



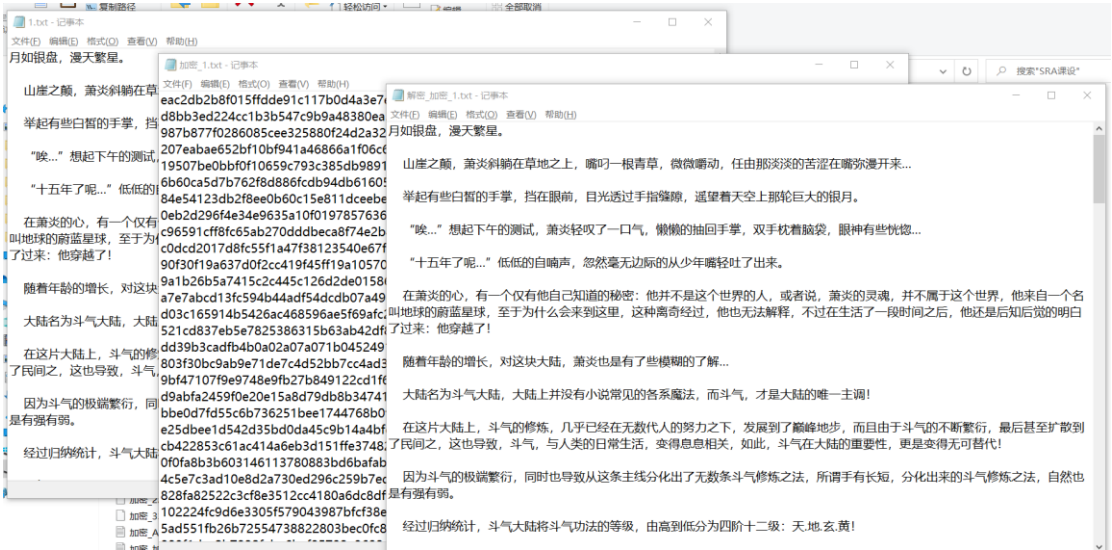
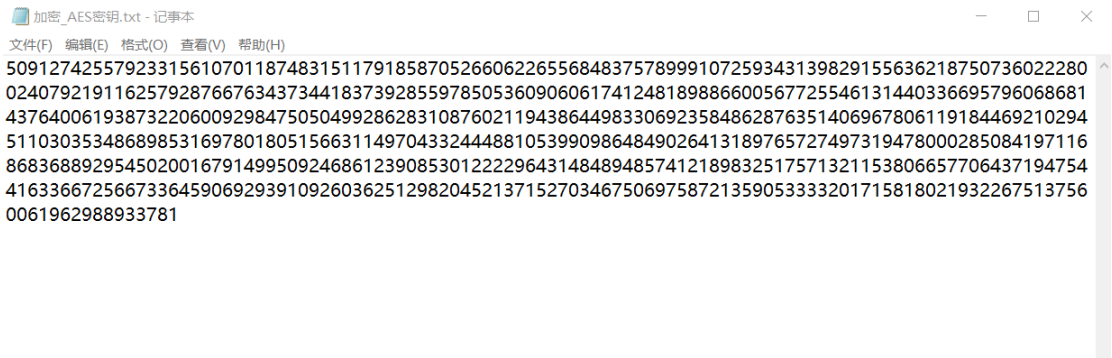


然后运用程序对其进行加密、解密：

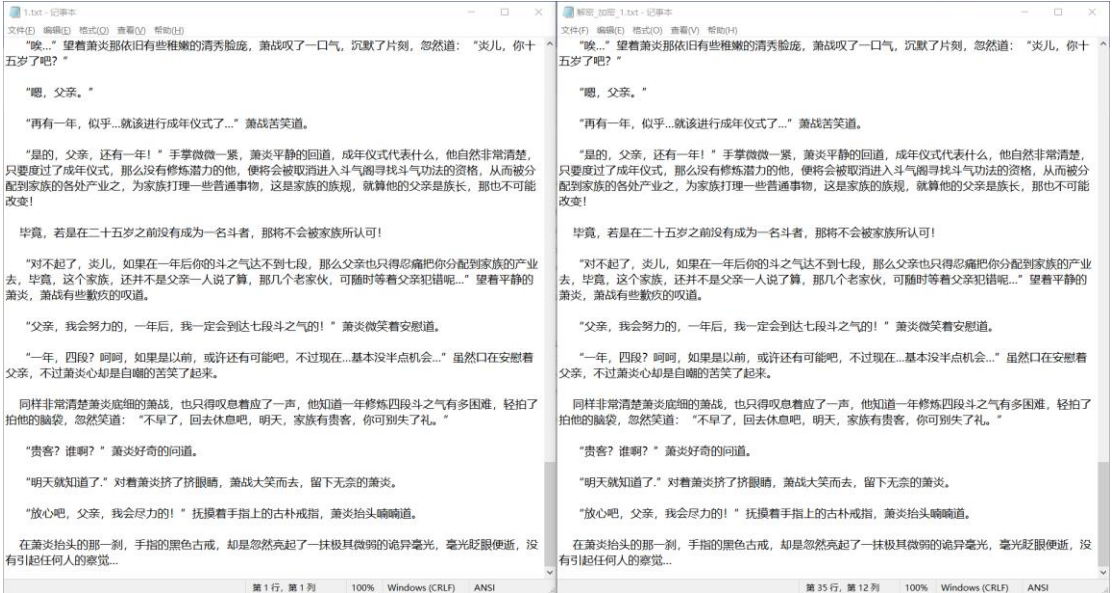
（因为已经设置了默认的输入，可以直接回车，更加快捷）



加密后获得了加密_AES 密钥.txt 文件和加密_1.txt 文件，解密后获得了解密_加密_1.txt 文件，AES 密钥的解密数据就没有保存下来了，直接在程序中使用。如下：

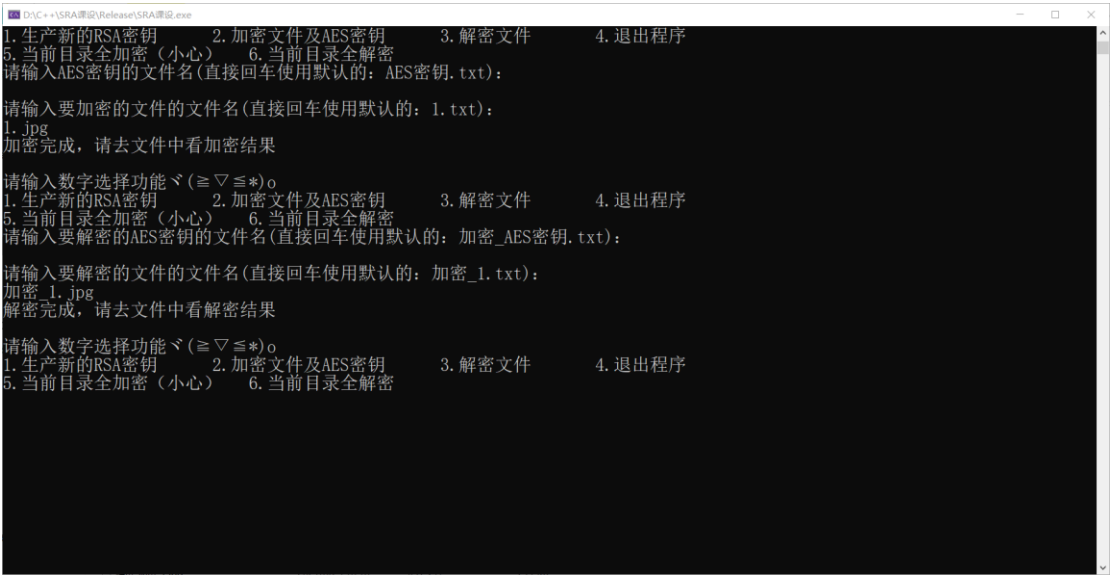


查看最后一行，可以发现是完全相同的：

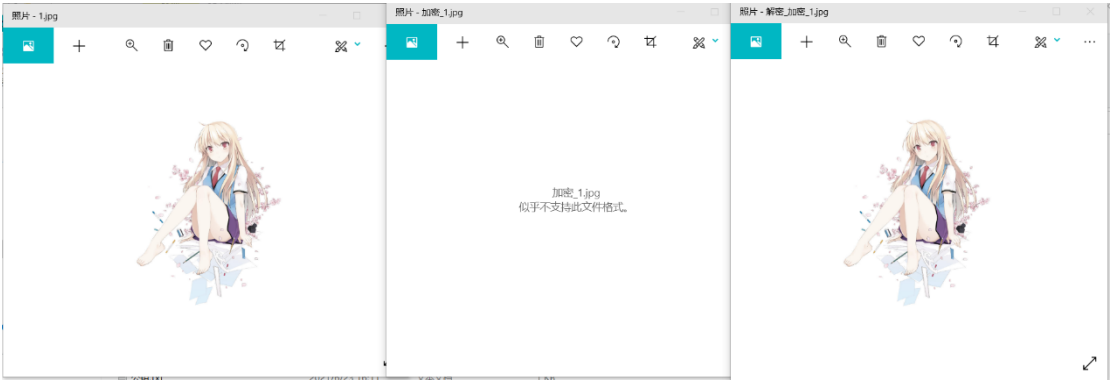


2.对其他格式的文件进行加密

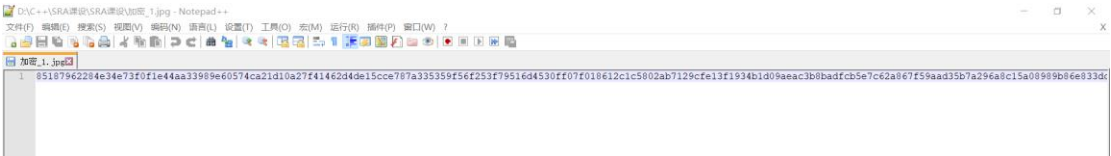
例如对图片 1.jpg 进行加密：



得到如下：













加密后是完全打不开的，解密后又是和原图一样的，将加密的图片强行打开，得到如下：



说明已经被加密成一连串的字符串了。
经过验证对于其他可以被二进制打开的文件都可以如此进行加密。

3.对当前目录的所有文件进行加密并替换

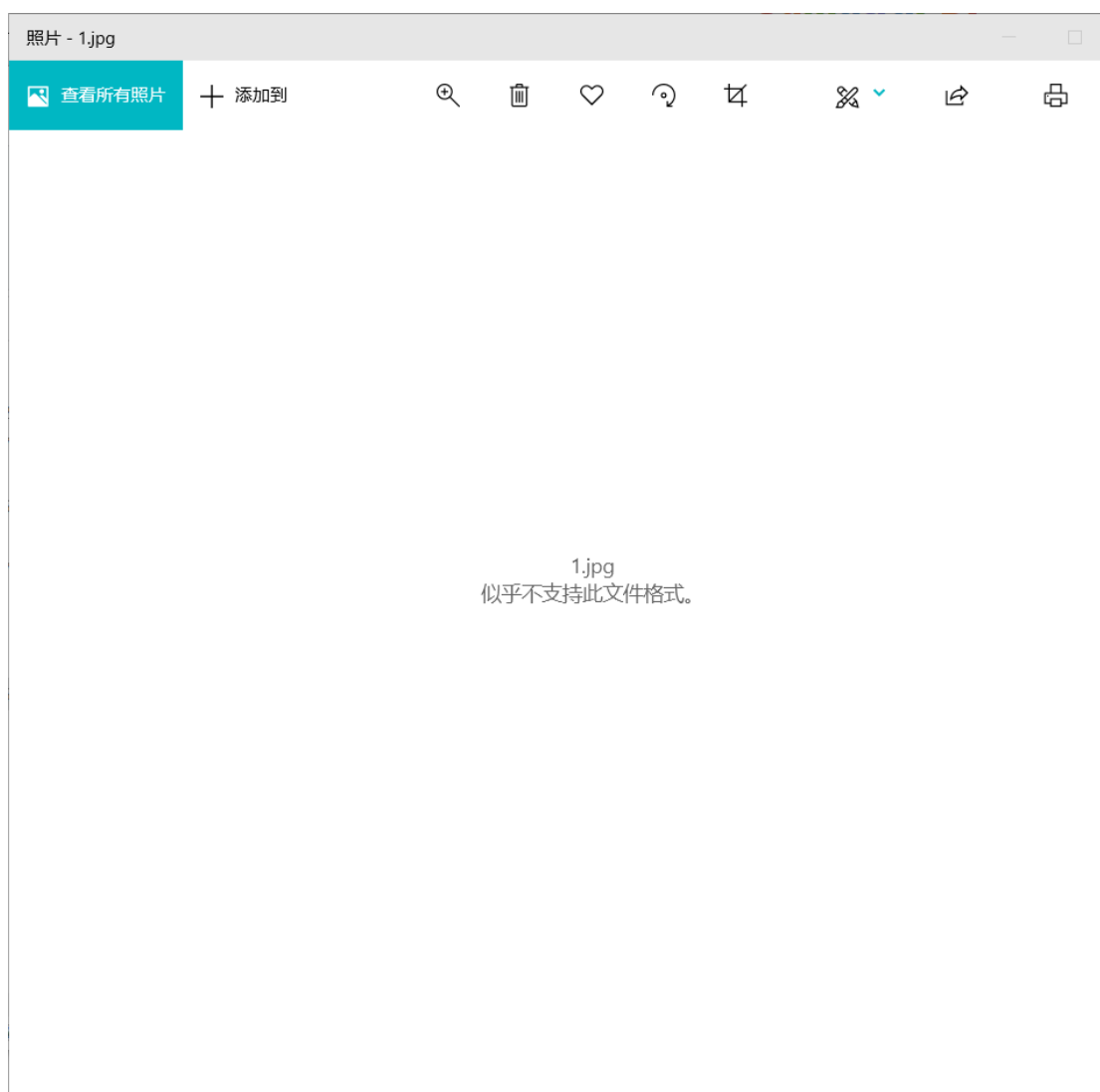
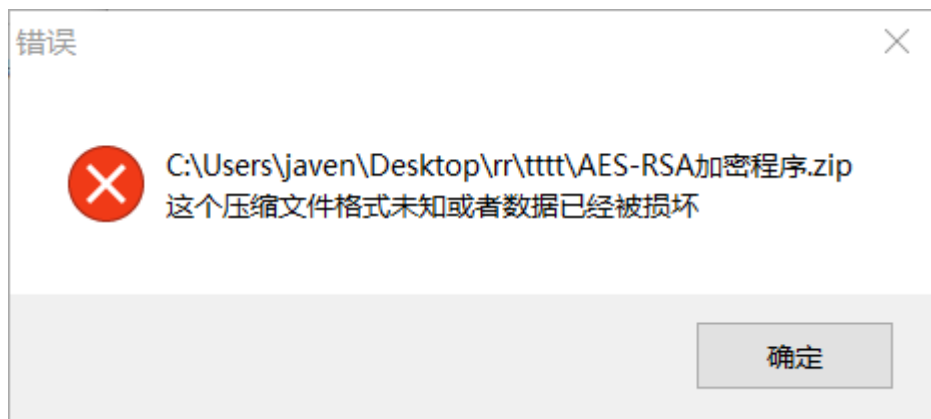
该目录下的文件

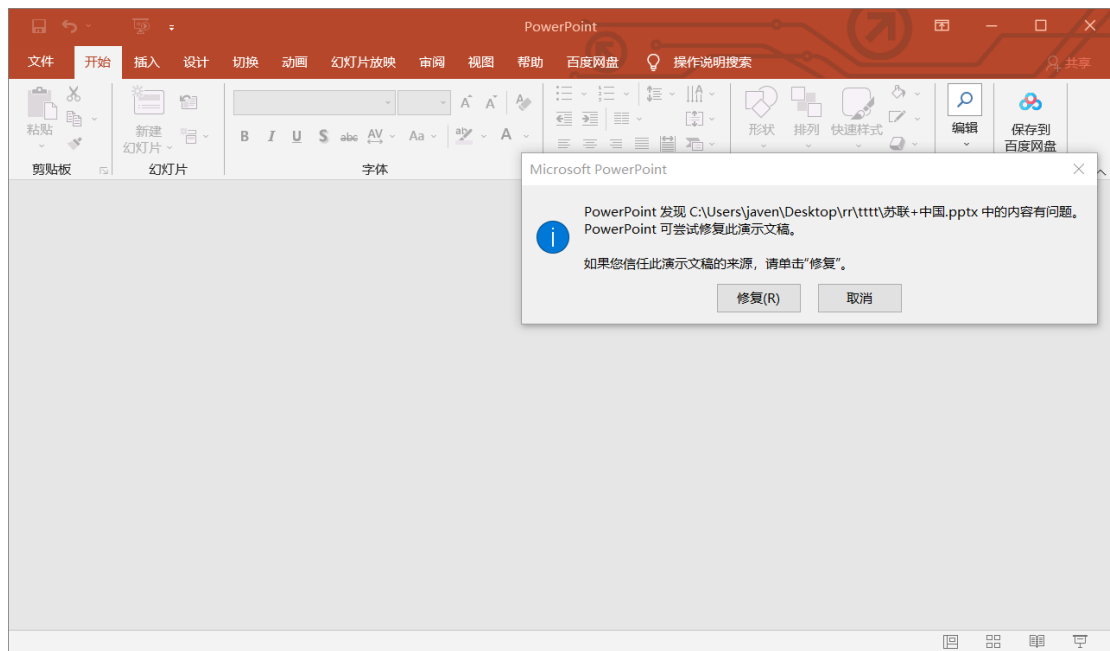
名称	修改日期	类型	大小
 1.jpg	2020/10/13 1:48	JPG 文件	211 Ki
 AES-RSA加密程序.zip	2021/6/22 22:22	WinRAR ZIP 压缩...	49 Ki
 AES密钥.txt	2021/6/3 20:21	文本文档	1 Ki
 DeepL	2021/3/17 13:54	快捷方式	3 Ki
 SRA课设.exe	2021/6/22 22:19	应用程序	101 Ki
 第1章 古典密码.pdf	2021/3/8 19:47	PDF 文件	1,999 Ki
 公钥.txt	2021/6/22 15:46	文本文档	1 Ki
 加密_AES密钥.txt	2021/6/22 22:19	文本文档	1 Ki
 私钥.txt	2021/6/22 15:46	文本文档	1 Ki
 苏联+中国.pptx	2021/6/1 11:17	Microsoft Power...	1,918 Ki

按 5 加密：

```
C:\Users\javen\Desktop\rr\TTTT\SRA课设.exe
请输入数字选择功能 (≡ ▽ ≐ *) o
1. 生产新的RSA密钥      2. 加密文件及AES密钥      3. 解密文件      4. 退出程序
5. 当前目录全加密 (小心)  6. 当前目录全解密
加密中...
. \\1.jpg
. \\AES-RSA加密程序.zip
. \\DeepL.lnk
. \\第1章 古典密码.pdf
. \\苏联+中国.pptx
请输入数字选择功能 (≡ ▽ ≐ *) o
1. 生产新的RSA密钥      2. 加密文件及AES密钥      3. 解密文件      4. 退出程序
5. 当前目录全加密 (小心)  6. 当前目录全解密
```

然后查看目录下的文件，发现都不能打开：

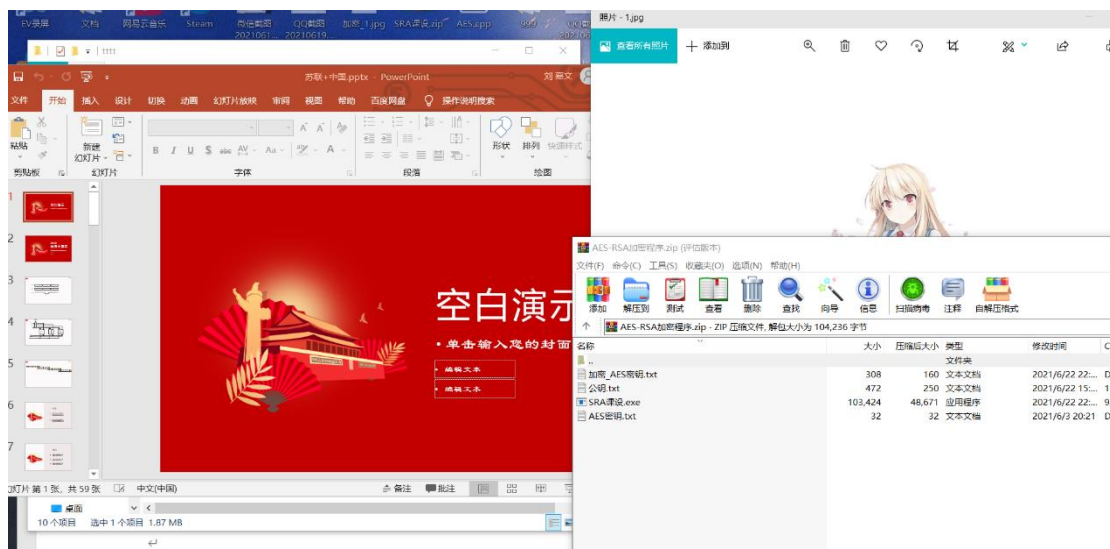




然后按 6，对其进行解密



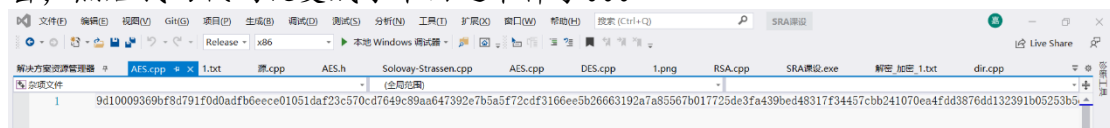
再次查看文件，发现可以正常打开：



四、心得

写完之后还是大舒了一口气，这一次课设真的是让我为不多的感觉十分繁琐的代码，断断续续地写了一个多星期，个人感觉 DES 是比较麻烦的一个程序，也可能是因为最开始写的是 DES 程序，所以有些不熟练，去看一些别人写的代码，发现可以用 `bitset` 数据变量，然后就感觉找到了写的路数了，后面就基本没有看过别人的代码了，就是 S 盒、IP 盒这种是直接复制过来的，中间也出了很多 bug，最后靠的是把所有中间变量全部输出出来，来和网上给的样例进行一一比较，最后找到问题并解决，只可惜当初 debug 的时候没有把相关的进行截图留存。写 AES 的时候一开始也是用 `bitset` 来进行写的，后面写了一些发现写起来很不顺利，再查看之后发现很多都是 8 位 8 位地进行操作的，所以就将其改为了 `byte` 作为主要的数据单位进行操作，写起来就顺利了挺多。写 RSA 主要是用了 ZZ 变量，所以一开始要配置链接，也还是搞了一会才明白网上的教程是是什么意思。RSA 整体写起来还是挺顺利的。后面将他们进行整合的时候，一开始是用的 EOF 进行判断是否读完，一个一个读取，然后发现对于一些文件不能完全读取，就改为了先获取文件的长度，然后再整块地读取，这样基本所有的文件类型都可以读取了，还可以开辟动态数组存放相应的数据。做到最后发现把所有的无关输出删除之后程序快了很多，后面把程序改为了 `release` 之后，发现又快了很多，基本都文件都可以进行秒加密了，这种速度其实是我一开始没有想到的，因为当时把 DES 所有的中间变量都输出，结果加密一次还需要好几秒。

中间还发生了一点大插曲，我在加上额外功能的时候，原本是开了个 `test` 文件夹，都是把 `exe` 放到这个文件夹，遍历文件然后替换加密的，后面发现程序有 bug，加密完了之后不能够解密出来，然后在调试的时候一不小心在当前代码的文件夹下面运行了这个程序，然后所有的代码都被加密了，关键是当时还不能解密，然后我的代码就变成了下面这个样子。。。



只有主函数那个 `cpp` 还有一个运行时留下来的备份，其他全没了，然后直接慌了，后面冷静下来之后，想了想，这些代码全部都被我用 AES 加密了，而还好我还有 `aes` 相关的关键密钥，所以只需要再用 AES 就进行解密就好了，后面又在看唯一剩下的主函数 `cpp` 的代码，然后终于发现了问题，如下



因为没有写 `}`，所以对于 `if` 语句中那些不应该加密的文件也会进行加密，这样我在解密的时候就不能获得相应的密钥了，然后我又仿造了我遍历文件的函数，来查看我当时加密的顺序，然后发现有一些是在我使用加密函数加密 AES 密钥之前加密的，对于这些我用我原来的程序，获得原本的 AES 密钥加密后的结果，然后使用程序，对 AES 密钥解密，用解密的 AES 密钥（即一开始加密使用的 AES 密

钥)对他们进行解密,成功解密了几个 `cpp`。然后后面的就不能这么解密了,因为后面对 `AES` 密钥这个文件加密了,也就是说后面的文件都是用 `AES` 密钥加密后的结果当做 `AES` 加密的密钥,然后我就用这个密钥再次对其进行解密,就得到了我当初备份的一个 `zip` 文件,然后所有的文件就都回来了,真的是幸好只有这里有问题,不然要是其他地方出了问题,就不一定能复原回来了,这一次也是长教训了,深刻理解了备份文件的重要性,之后一定要用 `git` 来进行备份编写程序!