

# 课程设计说明书

2153689 黄泽华

## 1、程序设计说明

### a. Aes128.cpp, aes128.h

这一部分定义并实现了密钥长度为 128 位的 AES 算法。首先定义了几个 AES 算法中会使用到的量，包括  $Nk, Nr, Rcon$ ，以及  $sBox, sBox\_inverse$  的值，这些会在后续的加密中使用到。然后是一些内部函数，包括  $subBytes, shiftRows, mixColumns$  等。在 AES 的每一轮加密中，都需要使用到加轮密钥、字节代换、行移位、列混淆。然后是两个外部可以使用的函数，加密以及解密，都需要读入一个 128bit 的文本，以及一个 128bit 的密钥，然后输出 128bit 的明文或者密文。

以下时实现的代码：

Aes128.h

```
#pragma once
```

```
#include <iostream>
```

```
#include <vector>
```

```
using namespace std;
```

```
class AES_128
```

```
{
```

```
    const int Nk = 4, Nr = 10;
```

```
    const uint8_t Rcon[11] = {0x00, 0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40, 0x80, 0x1B, 0x36};
```

```
    const uint8_t sBox[256] =
```

```
    {
```

```
        0x63, 0x7C, 0x77, 0x7B, 0xF2, 0x6B, 0x6F, 0xC5, 0x30, 0x01, 0x67, 0x2B, 0xFE, 0xD7, 0xAB, 0x76,
```

```
        0xCA, 0x82, 0xC9, 0x7D, 0xFA, 0x59, 0x47, 0xF0, 0xAD, 0xD4, 0xA2, 0xAF, 0x9C, 0xA4, 0x72, 0xC0,
```

```
        0xB7, 0xFD, 0x93, 0x26, 0x36, 0x3F, 0xF7, 0xCC, 0x34, 0xA5, 0xE5, 0xF1, 0x71, 0xD8, 0x31, 0x15,
```

```
        0x04, 0xC7, 0x23, 0xC3, 0x18, 0x96, 0x05, 0x9A, 0x07, 0x12, 0x80, 0xE2, 0xEB, 0x27, 0xB2, 0x75,
```

```
        0x09, 0x83, 0x2C, 0x1A, 0x1B, 0x6E, 0x5A, 0xA0, 0x52, 0x3B, 0xD6, 0xB3, 0x29, 0xE3, 0x2F, 0x84,
```

```
        0x53, 0xD1, 0x00, 0xED, 0x20, 0xFC, 0xB1, 0x5B, 0x6A, 0xCB, 0xBE, 0x39, 0x4A, 0x4C, 0x58, 0xCF,
```

```
        0xD0, 0xEF, 0xAA, 0xFB, 0x43, 0x4D, 0x33, 0x85, 0x45, 0xF9, 0x02, 0x7F, 0x50, 0x3C, 0x9F, 0xA8,
```

```
        0x51, 0xA3, 0x40, 0x8F, 0x92, 0x9D, 0x38, 0xF5, 0xBC, 0xB6, 0xDA, 0x21, 0x10, 0xFF, 0xF3, 0xD2,
```

```
        0xCD, 0x0C, 0x13, 0xEC, 0x5F, 0x97, 0x44, 0x17, 0xC4, 0xA7, 0x7E, 0x3D, 0x64, 0x5D, 0x19, 0x73,
```

```

        0x60, 0x81, 0x4F, 0xDC, 0x22, 0x2A, 0x90, 0x88, 0x46, 0xEE, 0xB8,
0x14, 0xDE, 0x5E, 0x0B, 0xDB,
        0xE0, 0x32, 0x3A, 0x0A, 0x49, 0x06, 0x24, 0x5C, 0xC2, 0xD3, 0xAC,
0x62, 0x91, 0x95, 0xE4, 0x79,
        0xE7, 0xC8, 0x37, 0x6D, 0x8D, 0xD5, 0x4E, 0xA9, 0x6C, 0x56, 0xF4,
0xEA, 0x65, 0x7A, 0xAE, 0x08,
        0xBA, 0x78, 0x25, 0x2E, 0x1C, 0xA6, 0xB4, 0xC6, 0xE8, 0xDD, 0x74,
0x1F, 0x4B, 0xBD, 0x8B, 0x8A,
        0x70, 0x3E, 0xB5, 0x66, 0x48, 0x03, 0xF6, 0x0E, 0x61, 0x35, 0x57,
0xB9, 0x86, 0xC1, 0x1D, 0x9E,
        0xE1, 0xF8, 0x98, 0x11, 0x69, 0xD9, 0x8E, 0x94, 0x9B, 0x1E, 0x87,
0xE9, 0xCE, 0x55, 0x28, 0xDF,
        0x8C, 0xA1, 0x89, 0x0D, 0xBF, 0xE6, 0x42, 0x68, 0x41, 0x99, 0x2D,
0x0F, 0xB0, 0x54, 0xBB, 0x16};
    const uint8_t invSBox[256] =
    {
        0x52, 0x09, 0x6a, 0xd5, 0x30, 0x36, 0xa5, 0x38, 0xbf, 0x40, 0xa3,
0x9e, 0x81, 0xf3, 0xd7, 0xfb,
        0x7c, 0xe3, 0x39, 0x82, 0x9b, 0x2f, 0xff, 0x87, 0x34, 0x8e, 0x43,
0x44, 0xc4, 0xde, 0xe9, 0xcb,
        0x54, 0x7b, 0x94, 0x32, 0xa6, 0xc2, 0x23, 0x3d, 0xee, 0x4c, 0x95,
0x0b, 0x42, 0xfa, 0xc3, 0x4e,
        0x08, 0x2e, 0xa1, 0x66, 0x28, 0xd9, 0x24, 0xb2, 0x76, 0x5b, 0xa2,
0x49, 0x6d, 0x8b, 0xd1, 0x25,
        0x72, 0xf8, 0xf6, 0x64, 0x86, 0x68, 0x98, 0x16, 0xd4, 0xa4, 0x5c,
0xcc, 0x5d, 0x65, 0xb6, 0x92,
        0x6c, 0x70, 0x48, 0x50, 0xfd, 0xed, 0xb9, 0xda, 0x5e, 0x15, 0x46,
0x57, 0xa7, 0x8d, 0x9d, 0x84,
        0x90, 0xd8, 0xab, 0x00, 0x8c, 0xbc, 0xd3, 0x0a, 0xf7, 0xe4, 0x58,
0x05, 0xb8, 0xb3, 0x45, 0x06,
        0xd0, 0x2c, 0x1e, 0x8f, 0xca, 0x3f, 0x0f, 0x02, 0xc1, 0xaf, 0xbd,
0x03, 0x01, 0x13, 0x8a, 0x6b,
        0x3a, 0x91, 0x11, 0x41, 0x4f, 0x67, 0xdc, 0xea, 0x97, 0xf2, 0xcf,
0xce, 0xf0, 0xb4, 0xe6, 0x73,
        0x96, 0xac, 0x74, 0x22, 0xe7, 0xad, 0x35, 0x85, 0xe2, 0xf9, 0x37,
0xe8, 0x1c, 0x75, 0xdf, 0x6e,
        0x47, 0xf1, 0x1a, 0x71, 0x1d, 0x29, 0xc5, 0x89, 0x6f, 0xb7, 0x62,
0x0e, 0xaa, 0x18, 0xbe, 0x1b,
        0xfc, 0x56, 0x3e, 0x4b, 0xc6, 0xd2, 0x79, 0x20, 0x9a, 0xdb, 0xc0,
0xfe, 0x78, 0xcd, 0x5a, 0xf4,
        0x1f, 0xdd, 0xa8, 0x33, 0x88, 0x07, 0xc7, 0x31, 0xb1, 0x12, 0x10,
0x59, 0x27, 0x80, 0xec, 0x5f,
        0x60, 0x51, 0x7f, 0xa9, 0x19, 0xb5, 0x4a, 0x0d, 0x2d, 0xe5, 0x7a,
0x9f, 0x93, 0xc9, 0x9c, 0xef,
        0xa0, 0xe0, 0x3b, 0x4d, 0xae, 0x2a, 0xf5, 0xb0, 0xc8, 0xeb, 0xbb,
0x3c, 0x83, 0x53, 0x99, 0x61,
        0x17, 0x2b, 0x04, 0x7e, 0xba, 0x77, 0xd6, 0x26, 0xe1, 0x69, 0x14,
0x63, 0x55, 0x21, 0x0c, 0x7d};

```

```

    void subBytes(vector<uint8_t> &word);
    void invSubBytes(vector<uint8_t> &word);
    void rotWord(vector<uint8_t> &word);
    vector<uint8_t> keyExpansion(const vector<uint8_t> &key);
    void addRoundKey(const vector<uint8_t> expandedKey, const int round,
vector<uint8_t> &word);
    void shiftRows(vector<uint8_t> &word);
    void invShiftRows(vector<uint8_t> &word);
    uint8_t gmul(uint8_t a, uint8_t b);
    void mixColumns(vector<uint8_t> &word);
    void invMixColumns(vector<uint8_t> &word);

public:
    vector<uint8_t> encrypt(const vector<uint8_t> &plainText, const
vector<uint8_t> &key);
    vector<uint8_t> decrypt(const vector<uint8_t> &cipherText, const
vector<uint8_t> &key);
};

```

Aes128.cpp

```

#include "aes128.h"

void AES_128::subBytes(vector<uint8_t> &word)
{
    for (auto &byte : word)
        byte = sBox[byte];
}

void AES_128::invSubBytes(vector<uint8_t> &word)
{
    for (auto &byte : word)
        byte = invSBox[byte];
}

void AES_128::rotWord(vector<uint8_t> &word)
{
    swap(word[0], word[1]);
    swap(word[1], word[2]);
    swap(word[2], word[3]);
}

vector<uint8_t> AES_128::keyExpansion(const vector<uint8_t> &key)
{
    vector<uint8_t> w(176);
    // Copy the first 4 groups

```

```

    for (int i = 0; i < 16; ++i)
        w[i] = key[i];
    for (int i = 4; i < 44; ++i)
    {
        vector<uint8_t> temp = {w[4 * i - 4], w[4 * i - 3], w[4 * i - 2], w[4
* i - 1]};
        if (i % Nk == 0)
        {
            rotWord(temp);
            subBytes(temp);
            temp[0] ^= Rcon[i / Nk];
        }
        w[4 * i + 0] = w[4 * (i - Nk) + 0] ^ temp[0];
        w[4 * i + 1] = w[4 * (i - Nk) + 1] ^ temp[1];
        w[4 * i + 2] = w[4 * (i - Nk) + 2] ^ temp[2];
        w[4 * i + 3] = w[4 * (i - Nk) + 3] ^ temp[3];
    }
    return w;
}

```

```

void AES_128::addRoundKey(const vector<uint8_t> expandedKey, const int round,
vector<uint8_t> &word)

```

```

{
    for (int i = 0; i < 16; ++i)
        word[i] ^= expandedKey[(round * 16) + i];
}

```

```

void AES_128::shiftRows(vector<uint8_t> &word)

```

```

{
    vector<uint8_t> tmp = {word[0], word[5], word[10], word[15],
        word[4], word[9], word[14], word[3],
        word[8], word[13], word[2], word[7],
        word[12], word[1], word[6], word[11]};

    word = tmp;
}

```

```

void AES_128::invShiftRows(vector<uint8_t> &word)

```

```

{
    vector<uint8_t> tmp = {word[0], word[13], word[10], word[7],
        word[4], word[1], word[14], word[11],
        word[8], word[5], word[2], word[15],
        word[12], word[9], word[6], word[3]};

    word = tmp;
}

```

```

uint8_t AES_128::gmul(uint8_t a, uint8_t b)

```

```

{
    uint8_t p = 0;
    for (int counter = 0; counter < 8; counter++)

```

```

{
    if (b & 1)
        p ^= a;
    bool hi_bit_set = a & 0x80;
    a <<= 1;
    if (hi_bit_set)
        a ^= 0x1B;
    b >>= 1;
}
return p;
}
// The quick algorithm showed in ppt
void AES_128::mixColumns(vector<uint8_t> &word)
{
    vector<uint8_t> tmp(4);
    for (int i = 0; i < 4; ++i)
    {
        for (int j = 0; j < 4; ++j)
            tmp[j] = word[i * 4 + j];
        // s(i,j) xor s(i,j) is zeros
        for (int j = 0; j < 4; ++j)
            for (int k = 0; k < 4; ++k)
                word[i * 4 + j] ^= tmp[k];
        for (int j = 0; j < 4; ++j)
            tmp[j] = gmul(tmp[j], 2);
        for (int j = 0; j < 4; ++j)
            for (int k = 0; k < 2; ++k)
                word[i * 4 + j] ^= tmp[(j + k) % 4];
    }
}

void AES_128::invMixColumns(vector<uint8_t> &word)
{
    vector<uint8_t> tmp(4);
    for (int i = 0; i < 4; ++i)
    {
        for (int j = 0; j < 4; ++j)
            tmp[j] = word[i * 4 + j];
        word[i * 4 + 0] = gmul(tmp[0], 14) ^ gmul(tmp[1], 11) ^ gmul(tmp[2],
13) ^ gmul(tmp[3], 9);
        word[i * 4 + 1] = gmul(tmp[0], 9) ^ gmul(tmp[1], 14) ^ gmul(tmp[2],
11) ^ gmul(tmp[3], 13);
        word[i * 4 + 2] = gmul(tmp[0], 13) ^ gmul(tmp[1], 9) ^ gmul(tmp[2],
14) ^ gmul(tmp[3], 11);
        word[i * 4 + 3] = gmul(tmp[0], 11) ^ gmul(tmp[1], 13) ^ gmul(tmp[2],
9) ^ gmul(tmp[3], 14);
    }
}

```

```

vector<uint8_t> AES_128::encrypt(const vector<uint8_t> &plainText, const
vector<uint8_t> &key)
{
    vector<uint8_t> cipherText(plainText);
    vector<uint8_t> expandedKey = keyExpansion(key);
    addRoundKey(expandedKey, 0, cipherText);
    for (int i = 1; i <= 10; ++i)
    {
        subBytes(cipherText);
        shiftRows(cipherText);
        if (i != 10)
            mixColumns(cipherText);
        addRoundKey(expandedKey, i, cipherText);
    }
    return cipherText;
}

vector<uint8_t> AES_128::decrypt(const vector<uint8_t> &cipherText, const
vector<uint8_t> &key)
{
    vector<uint8_t> plainText(cipherText);
    vector<uint8_t> expandedKey = keyExpansion(key);
    addRoundKey(expandedKey, 10, plainText);
    for (int i = 9; i >= 0; --i)
    {
        invShiftRows(plainText);
        invSubBytes(plainText);
        addRoundKey(expandedKey, i, plainText);
        if (i != 0)
            invMixColumns(plainText);
    }
    return plainText;
}

```

#### b. Aescbc.cpp, aescbc.h

在这一部分，利用已经实现的 128bit 的 AES 算法，使用 CBC 模式，并在输入文件的最后添加 PKCS7 填充，就可以实现对一个长的字符串的加密。在本函数中，我分别实现了两种不同的加密以及解密方式，其中一种为直接读入一个字符串，然后输出一个字符串；另一种则是先分别读入输入与输出文件名，然后从文件中进行输入、输出。后者基于前者，但是在使用的过程中会更加方便使用以及检验。

以下是实现的代码。为了应对文件名输入错误导致的异常情况，我加入了一些基本的错误处理机制：

Aescbc.h

```
#pragma once
```

```

#include <fstream>
#include <vector>
#include <random>
#include <string>

#include "aes128.h"

using namespace std;

class AES_CBC : public AES_128
{
private:
    vector<uint8_t> IV = {0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07,
0x08, 0x09, 0x0a, 0x0b, 0x0c, 0x0d, 0x0e, 0x0f};

public:
    void encryptFile(const string &inputFilename, const string
&outputFilename, const vector<uint8_t> &key);
    void decryptFile(const string &inputFilename, const string
&outputFilename, const vector<uint8_t> &key);
    string encryptString(const string &plaintext, const vector<uint8_t> &key);
    string decryptString(const string &ciphertext, const vector<uint8_t>
&key);
};

```

Aescbc.cpp

```

#include "aescbc.h"

string AES_CBC::encryptString(const string &plaintext, const vector<uint8_t>
&key)
{
    string ciphertext;
    vector<uint8_t> block(16);
    vector<uint8_t> lastCipherBlock = IV;
    for (int blocki = 0; blocki < (int)plaintext.size() / 16; ++blocki)
    {
        for (int i = 0; i < 16; ++i)
            block[i] = lastCipherBlock[i] ^ plaintext[i + blocki * 16];
        lastCipherBlock = encrypt(block, key);
        ciphertext.append(reinterpret_cast<const char *>(&lastCipherBlock[0]),
16);
    }
    int k = plaintext.size() % 16;
    for (int i = 0; i < 16; ++i)
        block[i] = lastCipherBlock[i] ^ (i < k ? plaintext[plaintext.size() -
k + i] : 16 - k);
    lastCipherBlock = encrypt(block, key);
}

```

```

        ciphertext.append(reinterpret_cast<const char *>(&lastCipherBlock[0]),
16);
        return ciphertext;
    }

string AES_CBC::decryptString(const string &ciphertext, const vector<uint8_t>
&key)
{
    string plaintext;
    vector<uint8_t> block(16);
    vector<uint8_t> lastCipherBlock = IV;
    for (int blocki = 0; blocki < (int)(ciphertext.size() + 15) / 16;
++blocki)
    {
        for (int i = 0; i < 16; ++i)
            block[i] = ciphertext[i + blocki * 16];
        vector<uint8_t> plainBlock = decrypt(block, key);
        for (int i = 0; i < 16; ++i)
            plainBlock[i] ^= lastCipherBlock[i];
        if (blocki < (int)(ciphertext.size() - 1) / 16)
            plaintext.append(reinterpret_cast<const char *>(&plainBlock[0]),
16);
        else if (plainBlock[15] < 16)
            plaintext.append(reinterpret_cast<const char *>(&plainBlock[0]),
16 - plainBlock[15]);
        lastCipherBlock = block;
    }
    return plaintext;
}

void AES_CBC::encryptFile(const string &inputFilename, const string
&outputFilename, const vector<uint8_t> &key)
{
    ifstream inputFile(inputFilename, ios::binary);
    if (!inputFile.is_open())
    {
        cerr << "aesCBC.cpp AESCBC::encryptfile: infile error" << endl;
        return;
    }
    istreambuf_iterator<char> beg(inputFile), end;
    string message(beg, end);
    inputFile.close();
    string cipher = encryptString(message, key);
    ofstream outputFile(outputFilename, ios::binary);
    outputFile.write(cipher.c_str(), cipher.size());
    outputFile.close();
}

```



```

void AES_CBC::decryptFile(const string &inputFilename, const string
&outputFilename, const vector<uint8_t> &key)
{
    ifstream inputFile(inputFilename, ios::binary);
    if (!inputFile.is_open())
    {
        cerr << "aesCBC.cpp AESCBC::decryptfile: infile error" << endl;
        return;
    }
    istreambuf_iterator<char> beg(inputFile), end;
    string message(beg, end);
    inputFile.close();
    string plain = decryptString(message, key);
    ofstream outputFile(outputFilename, ios::binary);
    outputFile.write(plain.c_str(), plain.size());
    outputFile.close();
}

```

#### c. Common.cpp, common.h

在这一部分，我实现了一些在其他函数中可能会使用到的一些工具函数，例如将一个字符转换为 16 进制，以及将一段字符串以 16 进制显示出来，含有读入一段 16 进制的文本到一个字符串中。以下是实现的代码：

##### Common.h

```

#pragma once
#include <sstream>
#include <vector>

using namespace std;

char toHex(const uint8_t i);

stringstream printHex(const vector<uint8_t> &text);

bool readHex(istream &in, uint8_t &t);

```

##### common.cpp

```

#include "common.h"

char toHex(const uint8_t i)
{
    if (i % 16 < 10)
        return '0' + i % 16;
    return i % 16 - 10 + 'A';
}

```

```

stringstream printHex(const vector<uint8_t> &text)
{
    stringstream ss;
    for (uint32_t count = 0; count < text.size(); ++count)
    {
        if (count % 16 == 8)
            ss << ' ';
        else if (count != 0 && count % 16 == 0)
            ss << endl;
        ss << toHex(text[count] / 16) << toHex(text[count] % 16);
    }
    return ss;
}

bool readHex(istream &in, uint8_t &t)
{
    int k;
    t = 0;
    for (int times = 0; times < 2; ++times)
    {
        while (1)
        {
            k = in.get();
            if (k == EOF)
                return false;
            else if (k <= '9' && k >= '0')
                t = t * 16 + k - '0';
            else if (k <= 'f' && k >= 'a')
                t = t * 16 + k - 'a' + 10;
            else if (k <= 'F' && k >= 'A')
                t = t * 16 + k - 'A' + 10;
            else
                continue;
            break;
        }
    }
    return true;
}

```

#### d. RSA.cpp, RSA.h

在这一部分，我使用了 NTL 库中的函数，实现了 RSA 的密钥生成、加密、解密。具体分别是：

给定密钥的 bit 长度，默认为 512bit，生成一对 RSA 的公钥和私钥；

存储这一对 RSA 密钥的公钥、私钥到一个文件中；

通过打开一个文件来加载一对 RSA 的公钥和私钥；

获取这个 RSA 的公钥;

给定一组公钥和一段消息, 然后使用公钥对这个消息进行加密;

给定一个密文, 然后直接使用存储的私钥对这个消息进行解密;

给定一段消息, 然后使用存储的私钥对其进行签名;

给定一组公钥和一段消息以及其签名, 然后对这个签名进行验证。

以下是实现的代码:

RSA.h

```
#pragma once
#include <string>
#include <NTL/ZZ.h>
#include <NTL/ZZ_p.h>
#include <NTL/ZZ_pXFactoring.h>
using namespace std;

NTL_CLIENT

string toString(ZZ z);

class RSA
{
private:
    /* stores private and public key */
    ZZ a, b, n;
    /* default = 512 bit */
    int size;

public:
    RSA(){};
    RSA(string name);
    void keyGenreate(int keysize = 512);
    void getKey(string &B, string &N);           // get public key
    void getKey(ZZ &B, ZZ &N) { B = b, N = n; }; // get public key
    string encrypt(string plaintext, string B, string N);
    string decrypt(string ciphertext);
    string sign(string message);
    bool verify(string message, string signature, string B, string N);
    void store(string filename);
};
```

RSA.cpp

```
#include <iostream>
#include <sstream>
```

```

#include <string>
#include <fstream>
#include "RSA.h"

string toString(ZZ z)
{
    std::ostringstream oss;
    oss << z;
    return oss.str();
}

NTL::ZZ ZZFromStr(const std::string &str)
{
    NTL::ZZ number;
    std::stringstream ss(str);
    ss >> number;
    return number;
}

RSA::RSA(string name)
{
    ifstream i = ifstream(name.c_str(), ios::in);
    if (!i.is_open())
    {
        cerr << "RSA.cpp RSA::RSA(): input error" << endl;
        return;
    }
    i >> a >> b >> n >> size;
    i.close();
}

void RSA::keyGenreate(int key_size)
{
    if (key_size != 512 && key_size != 1024)
    {
        cerr << "RSA.cpp RSA::keyGenerate: key_size should be 512 or 1024" <<
endl;
        return;
    }
    ZZ p, q, phi;
    size = key_size;
    GenGermainPrime(p, size);
    GenGermainPrime(q, size);
    while (p == q)
        GenGermainPrime(q, size);
    n = p * q;
    phi = (p - 1) * (q - 1);
    do

```

```

    {
        RandomBnd(b, phi);
    } while (GCD(b, phi) != 1);
    InvMod(a, b, phi);
}

void RSA::getKey(string &B, string &N)
{
    B = toString(b);
    N = toString(n);
}

string RSA::encrypt(string plaintext, string B, string N)
{
    ZZ pt = conv<ZZ>(plaintext.c_str());
    ZZ eb = conv<ZZ>(B.c_str());
    ZZ en = conv<ZZ>(N.c_str());
    ZZ ciphertext;
    PowerMod(ciphertext, pt, eb, en);
    return toString(ciphertext);
}

string RSA::decrypt(string ciphertext)
{
    ZZ ct = conv<ZZ>(ciphertext.c_str());
    ZZ plaintext;
    PowerMod(plaintext, ct, a, n);
    return toString(plaintext);
}

string RSA::sign(string message)
{
    ZZ m, s, mm;
    m = ZZFromStr(message);
    rem(mm, m, n);
    s = PowerMod(m, a, n);
    string signature = toString(s);
    return signature;
}

bool RSA::verify(string message, string signature, string B, string N)
{
    ZZ m, s;
    m = ZZFromStr(message);
    s = ZZFromStr(signature);
    ZZ e = ZZFromStr(B);
    ZZ mod = ZZFromStr(N);
    ZZ computedSignature = PowerMod(s, e, mod);

```

```

        return m == computedSignature;
    }

    void RSA::store(string filename)
    {
        ofstream o(filename, ios::out | ios::binary);
        o << a << endl;
        o << b << endl;
        o << n << endl;
        o << size << endl;
        o.close();
    }

```

#### e. Sha1.cpp, sha1.h

在这里实现了 SHA-1 算法的加密，分别有两种输出方式，一种输出的为一段字符串，其中表示的是 16 进制；另一种是直接输出 ZZ 类，其中保存的也是 sha1 值。实现的代码如下：

Sha1.h

```

#pragma once
#include <iostream>
#include <string>
#include <vector>
#include <NTL/ZZ_p.h>
#define SHA1ROTATELEFT(value, bits) (((value) << (bits)) | ((value) >> (32 - (bits))))

NTL_CLIENT

using namespace std;

class SHA_1
{
    vector<uint32_t> h{0x67452301, 0xefcdab89, 0x98badcfe, 0x10325476,
0xc3d2e1f0};
    void processChunk(vector<uint32_t> &h, const string &chunk);

public:
    string sha1(const string &input);
    ZZ sha1zz(const string &input);
};

```

Sha1.cpp

```

#include "sha1.h"
#include "common.h"

```

```

void SHA_1::processChunk(vector<uint32_t> &h, const string &chunk)
{
    vector<uint32_t> w(80, 0);
    for (int i = 0; i < 16; ++i)
    {
        w[i] = (chunk[i * 4] & 0xff) << 24 |
                (chunk[i * 4 + 1] & 0xff) << 16 |
                (chunk[i * 4 + 2] & 0xff) << 8 |
                (chunk[i * 4 + 3] & 0xff);
    }
    for (int i = 16; i < 80; ++i)
        w[i] = SHA1ROTATELEFT((w[i - 3] ^ w[i - 8] ^ w[i - 14] ^ w[i - 16]),
1);
    uint32_t a = h[0];
    uint32_t b = h[1];
    uint32_t c = h[2];
    uint32_t d = h[3];
    uint32_t e = h[4];

    for (int i = 0; i < 80; ++i)
    {
        uint32_t f, k;
        if (i < 20)
        {
            f = (b & c) | ((~b) & d);
            k = 0x5a827999;
        }
        else if (i < 40)
        {
            f = b ^ c ^ d;
            k = 0x6ed9eba1;
        }
        else if (i < 60)
        {
            f = (b & c) | (b & d) | (c & d);
            k = 0x8f1bbcdc;
        }
        else
        {
            f = b ^ c ^ d;
            k = 0xca62c1d6;
        }
        uint32_t temp = SHA1ROTATELEFT(a, 5) + f + e + k + w[i];
        e = d;
        d = c;
        c = SHA1ROTATELEFT(b, 30);
        b = a;
        a = temp;
    }
}

```

```

    }
    h[0] += a;
    h[1] += b;
    h[2] += c;
    h[3] += d;
    h[4] += e;
}

string SHA_1::sha1(const string &input)
{
    string padded_input = input;
    uint64_t orig_length = input.length() * 8;
    padded_input += (char)0x80;
    while (padded_input.length() % 64 != 56)
        padded_input += (char)0x00;
    for (int i = 7; i >= 0; --i)
        padded_input += (char)((orig_length >> (i * 8)) & 0xff);
    for (size_t i = 0; i < padded_input.length(); i += 64)
        processChunk(h, padded_input.substr(i, 64));
    string result;
    for (size_t i = 0; i < h.size(); ++i)
        for (int j = 7; j >= 0; --j)
            result += toHex(((h[i] >> (j * 4)) & 0xf));
    return result;
}

ZZ SHA_1::sha1zz(const string &input)
{
    string padded_input = input;
    uint64_t orig_length = input.length() * 8;
    padded_input += (char)0x80;
    while (padded_input.length() % 64 != 56)
        padded_input += (char)0x00;
    for (int i = 7; i >= 0; --i)
        padded_input += (char)((orig_length >> (i * 8)) & 0xff);
    for (size_t i = 0; i < padded_input.length(); i += 64)
        processChunk(h, padded_input.substr(i, 64));
    ZZ result = conv<ZZ>("0");
    for (size_t i = h.size() - 1; i < h.size(); --i)
        for (int j = 0; j < 8; ++j)
            result = result * 16 + ((h[i] >> (j * 4)) & 0xf);
    return result;
}

```

#### f. Certificate.cpp, certificate.h

这一部分实现的是一个简单的证书方案。在每一次使用证书时，先尝试能否打开一个预先设定好的文件名，如歌可以打开，就从其中读取公钥、私钥，否则就生成一组公钥、私钥



并存储在其中。这样做是为了保证在每一次程序的运行过程中，都可以验证之前颁发的证书而不出现错误。在证书颁发时，需要签名者的 ID，公钥，并返回一个证书。验证证书时，会读入证书，并返回证书是否真实 (bool)，以及证书中存储的 ID、公钥。

实现的代码如下：

Certificate.h

```
#pragma once
#include "RSA.h"
#include <fstream>
#include <algorithm>
#include <sstream>

class Certificate
{
private:
    ZZ a, b, n, size;
    string TA;

public:
    Certificate(string ta = "1002153689");
    void issue(const string &id, const string &ib, const string &in,
stringstream &o);
    bool verify(stringstream &i, string &id, string &vb, string &vn);
};
```

Certificate.cpp

```
#include "Certificate.h"
#include "sha1.h"
using namespace std;

Certificate::Certificate(string ta)
{
    ifstream in("cert.cert", ios::in | ios::binary);
    if (!in.is_open())
    {
        RSA rsa;
        rsa.keyGenreate(1024);
        rsa.store("cert.cert");
        std::ofstream outputFile("cert.cert", ios::out | ios_base::app);
        TA = ta;
        outputFile << TA << endl;
        outputFile.close();
        in = ifstream("cert.cert", ios::in | ios::binary);
    }
    in >> a >> b >> n >> size >> TA;
    in.close();
}
```

```

}

void Certificate::issue(const string &id, const string &ib, const string &in,
stringstream &o)
{
    string plain = ib + in + id;
    SHA_1 sha;
    ZZ plainh = sha.sha1zz(plain);
    ZZ cert = PowerMod(plainh, a, n);
    o << id << endl;
    o << ib << endl;
    o << in << endl;
    o << ib.length() << endl;
    o << in.length() << endl;
    o << cert << endl;
    o << TA << endl;
}

bool Certificate::verify(stringstream &i, string &id, string &vb, string &vn)
{
    int flagb, flagn;
    string ta;
    ZZ cert, cb, cn;
    i >> id >> vb >> vn >> flagb >> flagn >> cert >> ta;
    if (ta != "1002153689")
        return false;
    if ((int)vb.length() != flagb || (int)vn.length() != flagn)
        return false;
    string plain = vb + vn + id;
    SHA_1 sha;
    ZZ plainh = sha.sha1zz(plain);
    ZZ certd = PowerMod(cert, b, n);
    return (bool)(certd == plainh);
}

```

g. File.cpp, file.h

这一部分实现的是一个文件传输的功能。无论是发送还是接受，都需要本人的私钥、双方的证书以及输入输出文件的名称。具体的代码实现如下：

File.h

```

#pragma once
#include "aescbc.h"
#include "Certificate.h"
#include "RSA.h"
#include "sha1.h"

```

NTL\_CLIENT

```
using namespace std;
```

```
class fileEncrypt : Certificate
{
public:
    void send(RSA &sender, const string &inFileName, const string
&outFileName, const string &sSign, const string &rSign);
    void receive(RSA &receiver, const string &inFileName, const string
&outFileName, const string &sSign, const string &rSign);
};
```

File.cpp

```
#include "file.h"
#include "sha1.h"
#include <cstdlib>
#include <Windows.h>

void fileEncrypt::send(RSA &sender, const string &inFileName, const string
&outFilename, const string &sSign, const string &rSign)
{
    // verify receiver's signature
    string c1, c2, rid, rb, rn;
    stringstream srSign(rSign);
    if (!verify(srSign, rid, rb, rn))
    {
        cout << "Send receiver certificate not verified." << endl;
        return;
    }
    cout << "Send receiver certificate verified." << endl;
```

以上的这一部分，实现的是对接收者证书的验证。只有当验证成功时，才会进行下一步操作。在验证的同时，从证书中也获得了接收者的公钥。

```
    // encrypt key to c2
    vector<uint8_t> key(16);
    for (auto &i : key)
        i = rand() % 256;
    ZZ zkey = conv<ZZ>(0);
    for (auto &i : key)
        zkey = (zkey * 256) + (unsigned int)i;
    c2 = sender.encrypt(toString(zkey), rb, rn);
```

在以上的这一部分中，实现的时对 AES 的密钥的 RSA 加密，并将加密后的内容储存在 c2 中。

```
    // read message
    ifstream in(inFileName, ios::in | ios::binary);
```

```

istreambuf_iterator<char> beg(in), end;
string message(beg, end);
in.close();

// encrypt message
ZZ b, n;
sender.getKey(b, n);
stringstream o;
issue(rid, toString(b), toString(n), o);
SHA_1 sha;
o << sSign << endl;
ZZ hashm = sha.sha1zz(message);
o << hashm << endl;
o << message;
AES_CBC aes;
srand((unsigned)time(NULL));
c1 = aes.encryptString(o.str(), key);

```

这一部分时生成 c1, c1 中都包括发送者的证书、消息的 hash 值以及具体的消息。在生成 c1 后, 就可以将 c1 和 c2 一起发送了。

```

string cMessage = c2 + '\n' + c1;
ofstream outFile(outFileName.c_str(), ios::out | ios::binary);
outFile.write(cMessage.c_str(), cMessage.size());
outFile.close();
}

void fileEncrypt::receive(RSA &receiver, const string &inFileName, const
string &outFileName, const string &sSign, const string &rSign)
{
    ifstream inFile(inFileName, ios::in | ios::binary);
    if (!inFile.is_open())
    {
        cout << "file not opened. From fileEncrypt::receive()" << endl;
        return;
    }
    string c1, c2;
    int c;
    while ((c = inFile.get()) != '\n')
        c2 += c;
    while ((c = inFile.get()) != EOF)
        c1 += c;
    inFile.close();

    ZZ zkey = conv<ZZ>(receiver.decrypt(c2).c_str());
    vector<uint8_t> key(16);
    for (int i = 15; i >= 0; --i, zkey /= 256)
        key[i] = (uint8_t)(zkey % 256);

```

这一部分首先是读入消息，然后解析出 c1 和 c2，然后再把 c2 中的 key 通过私钥解密出来，得到 AES 的密钥。

```
AES_CBC aes;
stringstream i(aes.decryptString(c1, key));
string sid, sb, sn, tmp[8];
if (verify(i, sid, sb, sn))
    cout << "sender verified" << endl;
else
    cout << "sender not verified" << endl;
ZZ hash;
```

这一部分将消息进行解密，然后验证证书，并查看证书是否正确，然后输出。最后将 c1 中的消息有 hash 值进行验证，然后输出验证结果以及消息。

```
// get message and save it to file
i >> tmp[0] >> tmp[1] >> tmp[2] >> tmp[3] >> tmp[4] >> tmp[5] >> tmp[6] >>
tmp[7];
i >> hash;
i.get();
string message = i.str().substr(i.tellg());
ofstream ofile(outFileName, ios::out | ios::binary);
ofile.write(message.c_str(), message.length());
ofile.close();
}
```

#### h. Main.cpp

在这一部分，主要是控制输入输出以及部分的文件流的打开关闭，同时进行一些进行一些必要且人性化的提示，然后调用各个其他的函数来实现具体的功能。在每一部份的函数中，都有着详细的提示，可以告诉用户当前进行到了那一步，这样可以及时地对用户进行反馈。具体代码如下：

```
// 2153689 HZH
#include "aes128.h"
#include "aescbc.h"
#include "sha1.h"
#include "common.h"
#include "RSA.h"
#include "Certificate.h"
#include "file.h"
#include <cstdlib>
#include <Windows.h>

using namespace std;
```

```

void aesEncrypt()
{
    cout << "main.cpp aesEncrypt: AES encryption starts" << endl;
    string infile, outfile;
    cin >> infile >> outfile;
    ifstream i(infile.c_str(), ios::binary | ios::in);
    if (!i.is_open())
    {
        cerr << "main.cpp aesEncrypt: infile error" << endl;
        return;
    }
    cout << "main.cpp aesEncrypt: reading " << infile << endl;
    vector<uint8_t> key;
    vector<uint8_t> plainText(16, 0);
    cout << "main.cpp aesEncrypt: reading key" << endl;
    for (auto &k : plainText)
        if (!readHex(i, k))
        {
            cerr << "main.cpp aesEncrypt: not 128 bit" << endl;
            return;
        }
    if (key.size() != 16)
    {
        cerr << "main.cpp aesEncrypt: regenerating key" << endl;
        key.resize(16);
        for (auto &i : key)
            i = rand() % 256;
    }
    cout << "main.cpp aesEncrypt: encrypting" << endl;
    AES_128 aes;
    vector<uint8_t> cipherText = aes.encrypt(plainText, key);
    ofstream o(outfile.c_str(), ios::binary | ios::out);
    o << printHex(cipherText).str() << endl;
    o << printHex(key).str() << endl;
    i.close();
    o.close();
    cout << "main.cpp aesEncrypt: success!" << endl;
    cout << endl;
}

```

```

void aesDecrypt()
{
    cout << "main.cpp aesDecrypt: AES decryption starts" << endl;
    string infile, outfile;
    cin >> infile >> outfile;
    cout << "main.cpp aesDecrypt: reading " << infile << endl;
    ifstream i(infile.c_str(), ios::binary | ios::in);
    if (!i.is_open())

```

```

{
    cerr << "main.cpp aesDecrypt: infile error" << endl;
    return;
}
vector<uint8_t> cipherText(16, 0);
vector<uint8_t> key(16);
cout << "main.cpp aesDecrypt: reading key" << endl;
for (auto &k : cipherText)
    if (!readHex(i, k))
    {
        cerr << "main.cpp aesDecrypt: not 128 bit" << endl;
        return;
    }
for (auto &k : key)
    if (!readHex(i, k))
    {
        cerr << "main.cpp aesDecrypt: key inavailable" << endl;
        return;
    }
cout << "main.cpp aesDecrypt: decrypting" << endl;
AES_128 aes;
vector<uint8_t> plainText = aes.decrypt(cipherText, key);
ofstream o(outfile.c_str(), ios::binary | ios::out);
o << printHex(plainText).str() << endl;
i.close();
o.close();
cout << "main.cpp aesDecrypt: success!" << endl;
cout << endl;
}

void CBCencrypt()
{
    cout << "main.cpp CBCencrypt: CBC encryption starts" << endl;
    string infile, outfile, keyfile;
    cin >> infile >> outfile >> keyfile;
    cout << "main.cpp CBCdecrypt: generating key" << endl;
    vector<uint8_t> key(16);
    for (auto &i : key)
        i = rand() % 256;
    AES_CBC aes;
    cout << "main.cpp CBCencrypt: encrypting" << endl;
    aes.encryptFile(infile, outfile, key);
    cout << "main.cpp CBCencrypt: saving key" << endl;
    ofstream keyout(keyfile, ios::binary | ios::out);
    keyout << printHex(key).str() << endl;
    keyout.close();
    cout << "main.cpp CBCencrypt: success!" << endl;
    cout << endl;
}

```

```

}

void CBCdecrypt()
{
    cout << "main.cpp CBCdecrypt: CBC decryption starts" << endl;
    string infile, outfile, keyfile;
    cin >> infile >> outfile >> keyfile;
    vector<uint8_t> key(16);
    ifstream input(keyfile, ios::binary | ios::in);
    if (!input.is_open())
    {
        cerr << "main.cpp CBCdecrypt: infile error" << endl;
        return;
    }
    for (auto &k : key)
        if (!readHex(input, k))
        {
            cerr << "main.cpp CBCdecrypt: key inavailable" << endl;
            exit(-1);
        }
    input.close();
    cout << "main.cpp CBCdecrypt: decrypting" << endl;
    AES_CBC aes;
    aes.decryptFile(infile, outfile, key);
    cout << "main.cpp CBCdecrypt: success!" << endl;
    cout << endl;
}

void SHA1()
{
    string infile;
    cin >> infile;
    cout << "main.cpp SHA1: reading infile" << endl;
    ifstream inputFile(infile, ios::binary | ios::in);
    if (!inputFile.is_open())
    {
        cerr << "main.cpp SHA1: infile error" << endl;
        return;
    }
    istreambuf_iterator<char> beg(inputFile), end;
    string message(beg, end);
    inputFile.close();
    SHA_1 sha;
    cout << sha.sha1(message) << endl;
    cout << endl;
}

void RSAencrypt()

```



```

{
    cout << "main.cpp RSAencrypt: RSA encryption starts" << endl;
    string infile, outfile, keyfile, message, b, n;
    int keysize;
    cin >> infile >> outfile >> keyfile >> keysize;
    if (keysize != 512 && keysize != 1024)
    {
        cerr << "main.cpp RSAencrypt: keysize should be 512 or 1024" << endl;
        return;
    }
    cout << "main.cpp RSAencrypt: reading " << infile << endl;
    ifstream inputFile(infile, ios::binary | ios::in);
    if (!inputFile.is_open())
    {
        cerr << "main.cpp RSAencrypt: infile error" << endl;
        return;
    }
    inputFile >> message;
    inputFile.close();
    RSA rsa;
    cout << "main.cpp RSAencrypt: encrypting" << endl;
    rsa.keyGenreate(keysize);
    rsa.getKey(b, n);
    string ciphertext = rsa.encrypt(message, b, n);
    ofstream output(outfile, ios::binary | ios::out);
    output << ciphertext << endl;
    output.close();
    rsa.store(keyfile);
    cout << "main.cpp RSAencrypt: success" << endl;
    cout << endl;
}

void RSAdecrypt()
{
    cout << "main.cpp RSAdecrypt: RSA decryption starts" << endl;
    string infile, outfile, keyfile, message, b, n;
    cin >> infile >> outfile >> keyfile;
    cout << "main.cpp RSAdecrypt: reading " << infile << endl;
    ifstream inputFile(infile, ios::binary | ios::in);
    if (!inputFile.is_open())
    {
        cerr << "main.cpp RSAdecrypt: infile error" << endl;
        return;
    }
    inputFile >> message;
    inputFile.close();
    cout << "main.cpp RSAdecrypt: reading " << keyfile << endl;
    RSA rsa(keyfile);

```

```

    ofstream output(outfile, ios::binary | ios::out);
    cout << "main.cpp RSAdecrypt: decrypting" << endl;
    output << rsa.decrypt(message) << endl;
    output.close();
    cout << "success" << endl;
    cout << endl;
}

void RSAsign()
{
    cout << "main.cpp RSAsign: RSA signing starts" << endl;
    string infile, outfile, keyfile, message, b, n;
    int keysize;
    cin >> infile >> outfile >> keyfile >> keysize;
    if (keysize != 512 && keysize != 1024)
    {
        cerr << "main.cpp RSAsign: keysize should be 512 or 1024" << endl;
        return;
    }
    cout << "main.cpp RSAsign: reading " << infile << endl;
    ifstream inputFile(infile, ios::binary | ios::in);
    if (!inputFile.is_open())
    {
        cerr << "main.cpp RSAsign: infile error" << endl;
        return;
    }
    inputFile >> message;
    inputFile.close();
    RSA rsa;
    cout << "main.cpp RSAsign: signing" << endl;
    rsa.keyGenreate(keysize);
    rsa.getKey(b, n);
    string ciphertext = rsa.sign(message);
    ofstream output(outfile, ios::binary | ios::out);
    output << ciphertext << endl;
    output.close();
    ofstream keyout(keyfile, ios::binary | ios::out);
    keyout << b << endl;
    keyout << n << endl;
    keyout.close();
    cout << "main.cpp RSAsign: success" << endl;
    cout << endl;
}

void RSAverify()
{
    cout << "main.cpp RSAverify: RSA verifying starts" << endl;
    string infile, outfile, keyfile, message, b, n, sign;

```

```

cin >> infile >> outfile >> keyfile;
cout << "main.cpp RSAverify: reading" << infile << endl;
ifstream inputFile(infile, ios::binary | ios::in);
if (!inputFile.is_open())
{
    cerr << "main.cpp RSAverify: infile error" << endl;
    return;
}
inputFile >> message;
inputFile.close();
cout << "main.cpp RSAverify: reading " << keyfile << endl;
ifstream keyFile(keyfile, ios::binary | ios::in);
if (!keyFile.is_open())
{
    cerr << "main.cpp RSAverify: keyfile error" << endl;
    return;
}
keyFile >> b >> n;
keyFile.close();
cout << "main.cpp RSAverify: reading " << outfile << endl;
ifstream signFile(outfile, ios::binary | ios::in);
if (!signFile.is_open())
{
    cerr << "main.cpp RSAverify: signfile error" << endl;
    return;
}
signFile >> sign;
signFile.close();
RSA rsa;
if (rsa.verify(message, sign, b, n))
    cout << "Verified" << endl;
else
    cout << "Not verified" << endl;
cout << endl;
}

void generatesign()
{
    cout << "main.cpp generatesign: generating sign" << endl;
    string id, pubkeyfilename, prikeyfilename, signfilename, b, n;
    int keysize;
    cin >> id >> pubkeyfilename >> prikeyfilename >> signfilename >> keysize;
    RSA rsa;
    rsa.keyGenreate(keysize);
    rsa.store(prikeyfilename);
    rsa.getKey(b, n);
    ofstream pubkeyfile(pubkeyfilename, ios::binary | ios::out);
    pubkeyfile << b << endl;
}

```

```

    pubkeyfile << n << endl;
    pubkeyfile.close();
    stringstream ss;
    Certificate cert;
    cert.issue(id, b, n, ss);
    ofstream signfile(signfilename, ios::binary | ios::out);
    signfile << ss.str();
    signfile.close();
    cout << "main.cpp generatesign: success" << endl;
    cout << endl;
}

void sendfile()
{
    cout << "main.cpp sendfile: Sending file" << endl;
    string infile, outfile, prikeyfile, sendersign, recerversign, senderid,
    senderb, sendern;
    cin >> infile >> outfile >> prikeyfile >> sendersign >> recerversign;
    RSA sender(prikeyfile);
    ifstream sendersignfile(sendersign, ios::binary | ios::in);
    if (!sendersignfile.is_open())
    {
        cerr << "main.cpp sendfile: sendersign error" << endl;
        return;
    }
    istreambuf_iterator<char> beg1(sendersignfile), end1;
    string sendersignmessage(beg1, end1);
    sendersignfile.close();

    ifstream recerversignfile(recerversign, ios::binary | ios::in);
    if (!recerversignfile.is_open())
    {
        cerr << "main.cpp sendfile: recerversign error" << endl;
        return;
    }
    istreambuf_iterator<char> beg2(recerversignfile), end2;
    string receiversignmessage(beg2, end2);
    recerversignfile.close();

    fileEncrypt file;
    file.send(sender, infile, outfile, sendersignmessage,
    receiversignmessage);
    cout << "main.cpp sendfile: file send to " << outfile << " successfully"
    << endl;
    cout << endl;
}

void receivefile()

```

```

{
    cout << "main.cpp receivefile: receiving file" << endl;
    string infile, outfile, prikeyfile, sendersign, recerversign, senderid,
    senderb, sendern;
    cin >> infile >> outfile >> prikeyfile >> sendersign >> recerversign;
    RSA receiver(prikeyfile);

    ifstream sendersignfile(sendersign, ios::binary | ios::in);
    if (!sendersignfile.is_open())
    {
        cerr << "main.cpp sendfile: sendersign error" << endl;
        return;
    }
    istreambuf_iterator<char> beg1(sendersignfile), end1;
    string sendersignmessage(beg1, end1);
    sendersignfile.close();

    ifstream recerversignfile(recerversign, ios::binary | ios::in);
    if (!recerversignfile.is_open())
    {
        cerr << "main.cpp sendfile: recerversign error" << endl;
        return;
    }
    istreambuf_iterator<char> beg2(recerversignfile), end2;
    string receiversignmessage(beg2, end2);
    recerversignfile.close();

    fileEncrypt file;
    file.receive(receiver, infile, outfile, sendersignmessage,
    receiversignmessage);
    cout << "main.cpp receivefile: file received to " << outfile << "
    successfully!" << endl;
    cout << endl;
}

void usage()
{
    cout << "AEncrypt [infilename] [outfilename]: AES encrypt 128 bit" <<
    endl;
    cout << "AESdecrypt [infilename] [outfilename]: AES decrypt 128 bit" <<
    endl;
    cout << "CBCencrypt [infilename] [outfilename] [keyfilename]: AES
    encrypt CBC mode" << endl;
    cout << "CBCdecrypt [infilename] [outfilename] [keyfilename]: AES
    decrypt CBC mode" << endl;
    cout << "SHA1 [infilename]: SHA-1" << endl;
    cout << "RSAencrypt [infilename] [outfilename] [keyfile] [keysize]: RSA
    encrypt" << endl;
}

```

```

        cout << "RSAdecrypt  [infilename] [outfilename] [keyfile]: RSA decrypt"
<< endl;
        cout << "RSAsign      [messagefile] [signfile] [keyfile] [keysize]: RSA
sign" << endl;
        cout << "RSAverify   [messagefile] [signfile] [keyfile]: RSA signature
verify" << endl;
        cout << "generatesign [id] [pubkeyfile] [prikeyfile] [signfile] [keysize]:
RSA signature verify" << endl;
        cout << "sendfile     [infile] [outfile] [prikeyfile] [sendersign]
[receiversign]: RSA signature verify" << endl;
        cout << "receivefile  [infile] [outfile] [prikeyfile] [sendersign]
[receiversign]: RSA signature verify" << endl;
        cout << "quit : exit" << endl;
        cout << "help : show help" << endl;
        cout << "E.g.   AESencrypt infile.txt outfile.txt" << endl;
        cout << "        RSAencrypt infile.txt outfile.txt keyfile.key 512" <<
endl;
        cout << endl;
    }
}

```

```

int main(int argv, char **argc)
{
    usage();
    while (1)
    {
        string type;
        cin >> type;
        if (type == "help")
            usage();
        else if (type == "quit")
            break;
        else if (type == "AESencrypt")
            aesEncrypt();
        else if (type == "AESdecrypt")
            aesDecrypt();
        else if (type == "CBCencrypt")
            CBCencrypt();
        else if (type == "CBCdecrypt")
            CBCdecrypt();
        else if (type == "SHA1")
            SHA1();
        else if (type == "RSAencrypt")
            RSAencrypt();
        else if (type == "RSAdecrypt")
            RSAdecrypt();
        else if (type == "RSAsign")
            RSAsign();
        else if (type == "RSAverify")

```

```

        RSAverify();
    else if (type == "generatesign")
        generatesign();
    else if (type == "sendfile")
        sendfile();
    else if (type == "receivefile")
        receivefile();
}
return 0;
}

```

## 2、程序使用说明

a. 预先准备的测试文本如下：

01\_0

0011223344556677 8899aabbccddeeff

02\_0

```

#include "aes128.h"

void AES_128::subBytes(vector<uint8_t> &word)
{
    for (auto &byte : word)
        byte = sBox[byte];
}

void AES_128::invSubBytes(vector<uint8_t> &word)
{
    for (auto &byte : word)
        byte = invSBox[byte];
}

void AES_128::rotWord(vector<uint8_t> &word)
{
    swap(word[0], word[1]);
    swap(word[1], word[2]);
    swap(word[2], word[3]);
}

vector<uint8_t> AES_128::keyExpansion(const vector<uint8_t> &key)
{
    vector<uint8_t> w(176);
    // Copy the first 4 groups
    for (int i = 0; i < 16; ++i)
        w[i] = key[i];
}

```

```

        for (int i = 4; i < 44; ++i)
        {
            vector<uint8_t> temp = {w[4 * i - 4], w[4 * i - 3], w[4 * i - 2], w[4
* i - 1]};
            if (i % Nk == 0)
            {
                rotWord(temp);
                subBytes(temp);
                temp[0] ^= Rcon[i / Nk];
            }
            w[4 * i + 0] = w[4 * (i - Nk) + 0] ^ temp[0];
            w[4 * i + 1] = w[4 * (i - Nk) + 1] ^ temp[1];
            w[4 * i + 2] = w[4 * (i - Nk) + 2] ^ temp[2];
            w[4 * i + 3] = w[4 * (i - Nk) + 3] ^ temp[3];
        }
        return w;
    }
}

```

```

void AES_128::addRoundKey(const vector<uint8_t> expandedKey, const int round,
vector<uint8_t> &state)
{
    for (int i = 0; i < 16; ++i)
        state[i] ^= expandedKey[(round * 16) + i];
}

```

```

void AES_128::shiftRows(vector<uint8_t> &state)
{
    vector<uint8_t> tmp = {state[0], state[5], state[10], state[15],
                           state[4], state[9], state[14], state[3],
                           state[8], state[13], state[2], state[7],
                           state[12], state[1], state[6], state[11]};

    state = tmp;
}

```

```

void AES_128::invShiftRows(vector<uint8_t> &state)
{
    vector<uint8_t> tmp = {state[0], state[13], state[10], state[7],
                           state[4], state[1], state[14], state[11],
                           state[8], state[5], state[2], state[15],
                           state[12], state[9], state[6], state[3]};

    state = tmp;
}

```

```

uint8_t AES_128::gmul(uint8_t a, uint8_t b)
{
    uint8_t p = 0;
    for (int counter = 0; counter < 8; counter++)
    {
        if (b & 1)

```



```

        p ^= a;
        bool hi_bit_set = a & 0x80;
        a <<= 1;
        if (hi_bit_set)
            a ^= 0x1B;
        b >>= 1;
    }
    return p;
}
// The quick algorithm showed in ppt
void AES_128::mixColumns(vector<uint8_t> &state)
{
    vector<uint8_t> tmp(4);
    for (int i = 0; i < 4; ++i)
    {
        for (int j = 0; j < 4; ++j)
            tmp[j] = state[i * 4 + j];
        // s(i,j) xor s(i,j) is zeros
        for (int j = 0; j < 4; ++j)
            for (int k = 0; k < 4; ++k)
                state[i * 4 + j] ^= tmp[k];
        for (int j = 0; j < 4; ++j)
            tmp[j] = gmul(tmp[j], 2);
        for (int j = 0; j < 4; ++j)
            for (int k = 0; k < 2; ++k)
                state[i * 4 + j] ^= tmp[(j + k) % 4];
    }
}

void AES_128::invMixColumns(vector<uint8_t> &state)
{
    vector<uint8_t> tmp(4);
    for (int i = 0; i < 4; ++i)
    {
        for (int j = 0; j < 4; ++j)
            tmp[j] = state[i * 4 + j];
        state[i * 4 + 0] = gmul(tmp[0], 14) ^ gmul(tmp[1], 11) ^ gmul(tmp[2],
13) ^ gmul(tmp[3], 9);
        state[i * 4 + 1] = gmul(tmp[0], 9) ^ gmul(tmp[1], 14) ^ gmul(tmp[2],
11) ^ gmul(tmp[3], 13);
        state[i * 4 + 2] = gmul(tmp[0], 13) ^ gmul(tmp[1], 9) ^ gmul(tmp[2],
14) ^ gmul(tmp[3], 11);
        state[i * 4 + 3] = gmul(tmp[0], 11) ^ gmul(tmp[1], 13) ^ gmul(tmp[2],
9) ^ gmul(tmp[3], 14);
    }
}
/*****

```

\* Since the algorithm written in ppt seems to be incorrect, I use the above function instead.

```
void AES_128::invMixColumns(vector<uint8_t> &state)
{
    vector<uint8_t> tmp(4);
    for (int i = 0; i < 4; ++i)
    {
        for (int j = 0; j < 4; ++j)
            tmp[j] = state[i * 4 + j];
        for (int j = 0; j < 4; ++j)
            for (int k = 0; k < 4; ++k)
                state[i * 4 + j] ^= tmp[k];
        for (int j = 0; j < 4; ++j)
            for (int k = 0; k < 2; ++k)
                state[i * 4 + j] ^= tmp[(j + k) % 4];
        tmp[0] = gmul(tmp[0] ^ tmp[2], 2);
        tmp[1] = gmul(tmp[1] ^ tmp[3], 2);
        for (int j = 0; j < 4; ++j)
            state[i * 4 + j] ^= tmp[j % 2];
        tmp[0] = gmul(tmp[0] ^ tmp[1], 2);
        for (int j = 0; j < 4; ++j)
            state[i * 4 + j] ^= tmp[0];
    }
}

*****/
```

```
vector<uint8_t> AES_128::encrypt(const vector<uint8_t> &plainText, const
vector<uint8_t> &key)
{
    vector<uint8_t> cipherText(plainText);
    vector<uint8_t> expandedKey = keyExpansion(key);
    addRoundKey(expandedKey, 0, cipherText);
    for (int i = 1; i <= 10; ++i)
    {
        subBytes(cipherText);
        shiftRows(cipherText);
        if (i != 10)
            mixColumns(cipherText);
        addRoundKey(expandedKey, i, cipherText);
    }
    return cipherText;
}
```

```
vector<uint8_t> AES_128::decrypt(const vector<uint8_t> &cipherText, const
vector<uint8_t> &key)
{
    vector<uint8_t> plainText(cipherText);
    vector<uint8_t> expandedKey = keyExpansion(key);
```

```

    addRoundKey(expandedKey, 10, plainText);
    for (int i = 9; i >= 0; --i)
    {
        invShiftRows(plainText);
        invSubBytes(plainText);
        addRoundKey(expandedKey, i, plainText);
        if (i != 0)
            invMixColumns(plainText);
    }
    return plainText;
}123456

```

03\_0 (空)

03\_1

a

03\_2

abc

03\_3

abcdefghijklmnopqrstuvwxyz

04\_0

123456789

05\_0 (同上)

06\_0 (同 02\_0)

b. 编译运行的标本如下:

AESencrypt 01\_0 01\_1

AESdecrypt 01\_1 01\_2

CBCencrypt 02\_0 02\_1 02\_2

CBCdecrypt 02\_1 02\_3 02\_2

SHA1 03\_0

SHA1 03\_1

SHA1 03\_2

SHA1 03\_3

RSACencrypt 04\_0 04\_1 04\_2 512

RSAddecrypt 04\_1 04\_3 04\_2

RSASign 05\_0 05\_1 05\_2 512

```

RSAverify 05_0 05_1 05_2

generatesign 100001 06_1_0 06_1_1 06_1_2 512

generatesign 100002 06_2_0 06_2_1 06_2_2 512

sendfile 06_0 06_3 06_1_1 06_1_2 06_2_2

receivefile 06_3 06_4 06_2_1 06_1_2 06_2_2

quit

```

compile.bat

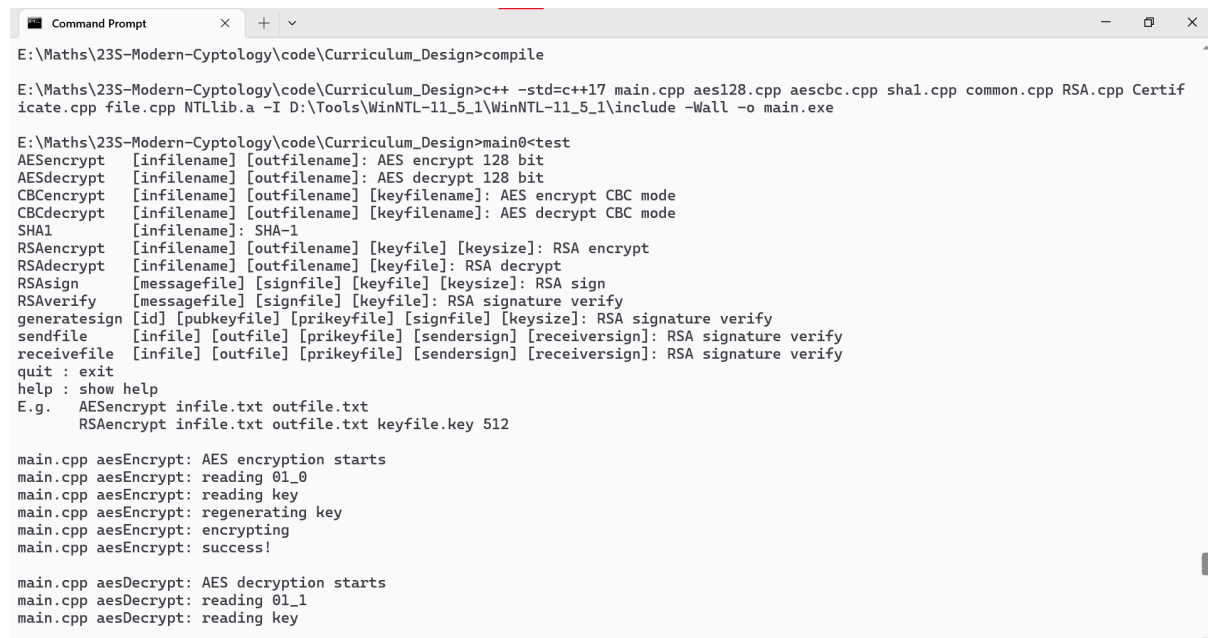
```

c++ -std=c++17 main.cpp aes128.cpp aescbc.cpp sha1.cpp common.cpp RSA.cpp
Certificate.cpp file.cpp NTLib.a -I D:\Tools\WinNTL-11_5_1\WinNTL-
11_5_1\include -Wall -o main.exe
main<test
fc 01_0 01_2 /c /w
echo n | comp 02_0 02_3
fc 04_0 04_3 /w
echo n | comp 06_4 06_0

```

c. 测试过程以及结果:

在命令行中输入 compile, 执行“compile.bat”, 然后观察命令提示符中的输出:



```

E:\Maths\23S-Modern-Cyptology\code\Curriculum_Design>compile

E:\Maths\23S-Modern-Cyptology\code\Curriculum_Design>c++ -std=c++17 main.cpp aes128.cpp aescbc.cpp sha1.cpp common.cpp RSA.cpp Certif
icate.cpp file.cpp NTLib.a -I D:\Tools\WinNTL-11_5_1\WinNTL-11_5_1\include -Wall -o main.exe

E:\Maths\23S-Modern-Cyptology\code\Curriculum_Design>main<test
AESencrypt [infilename] [outfilename]: AES encrypt 128 bit
AESdecrypt [infilename] [outfilename]: AES decrypt 128 bit
CBCencrypt [infilename] [outfilename] [keyfilename]: AES encrypt CBC mode
CBCdecrypt [infilename] [outfilename] [keyfilename]: AES decrypt CBC mode
SHA1 [infilename]: SHA-1
RSAencrypt [infilename] [outfilename] [keyfile] [keysize]: RSA encrypt
RSAdecrypt [infilename] [outfilename] [keyfile]: RSA decrypt
RSAsign [messagefile] [signfile] [keyfile] [keysize]: RSA sign
RSAverify [messagefile] [signfile] [keyfile]: RSA signature verify
generatesign [id] [pubkeyfile] [prikeyfile] [signfile] [keysize]: RSA signature verify
sendfile [infile] [outfile] [prikeyfile] [sendersign] [receiversign]: RSA signature verify
receivefile [infile] [outfile] [prikeyfile] [sendersign] [receiversign]: RSA signature verify
quit : exit
help : show help
E.g. AESencrypt infile.txt outfile.txt
RSAencrypt infile.txt outfile.txt keyfile.key 512

main.cpp aesEncrypt: AES encryption starts
main.cpp aesEncrypt: reading 01_0
main.cpp aesEncrypt: reading key
main.cpp aesEncrypt: regenerating key
main.cpp aesEncrypt: encrypting
main.cpp aesEncrypt: success!

main.cpp aesDecrypt: AES decryption starts
main.cpp aesDecrypt: reading 01_1
main.cpp aesDecrypt: reading key

```

```
Command Prompt
main.cpp aesDecrypt: AES decryption starts
main.cpp aesDecrypt: reading 01_1
main.cpp aesDecrypt: reading key
main.cpp aesDecrypt: decrypting
main.cpp aesDecrypt: success!

main.cpp CBCencrypt: CBC encryption starts
main.cpp CBCdecrypt: generating key
main.cpp CBCencrypt: encrypting
main.cpp CBCencrypt: saving key
main.cpp CBCencrypt: success!

main.cpp CBCdecrypt: CBC decryption starts
main.cpp CBCdecrypt: decrypting
main.cpp CBCdecrypt: success!

main.cpp SHA1: reading infile
DA39A3EE5E6B4B0D3255BFEF95601890AFD80709

main.cpp SHA1: reading infile
86F7E437FAA5A7FCE15D1DDCB9EAEAEA377667B8

main.cpp SHA1: reading infile
A9993E364706816ABA3E25717850C26C9CD0D89D

main.cpp SHA1: reading infile
32D10C7B8CF96570CA04CE37F2A19D084240D3A89

main.cpp RSAencrypt: RSA encryption starts
main.cpp RSAencrypt: reading 04_0
main.cpp RSAencrypt: encrypting
main.cpp RSAencrypt: success
```

```
Command Prompt
main.cpp RSAencrypt: encrypting
main.cpp RSAencrypt: success

main.cpp RSAdecrypt: RSA decryption starts
main.cpp RSAdecrypt: reading 04_1
main.cpp RSAdecrypt: reading 04_2
main.cpp RSAdecrypt: decrypting
success

main.cpp RSAsign: RSA signing starts
main.cpp RSAsign: reading 05_0
main.cpp RSAsign: signing
main.cpp RSAsign: success

main.cpp RSAverify: RSA verifying starts
main.cpp RSAverify: reading 05_0
main.cpp RSAverify: reading 05_2
main.cpp RSAverify: reading 05_1
Verified

main.cpp generatesign: generating sign
main.cpp generatesign: success

main.cpp generatesign: generating sign
main.cpp generatesign: success

main.cpp sendfile: Sending file
Send receiver certificate verified.
main.cpp sendfile: file sended to 06_3 successfully

main.cpp receivefile: receiving file
sender verified
message verified
```

```
Command Prompt
main.cpp generatesign: success

main.cpp sendfile: Sending file
Send receiver certificate verified.
main.cpp sendfile: file sent to 06_3 successfully

main.cpp receivefile: receiving file
sender verified
message verified
main.cpp receivefile: file received to 06_4 successfully!

E:\Maths\23S-Modern-Cyptology\code\Curriculum_Design>fc 01_0 01_2 /c /w
Comparing files 01_0 and 01_2
FC: no differences encountered

E:\Maths\23S-Modern-Cyptology\code\Curriculum_Design>echo n | comp 02_0 02_3
Comparing 02_0 and 02_3...
Files compare OK

Compare more files (Y/N) ?
E:\Maths\23S-Modern-Cyptology\code\Curriculum_Design>fc 04_0 04_3 /w
Comparing files 04_0 and 04_3
FC: no differences encountered

E:\Maths\23S-Modern-Cyptology\code\Curriculum_Design>echo n | comp 06_4 06_0
Comparing 06_4 and 06_0...
Files compare OK

Compare more files (Y/N) ?
E:\Maths\23S-Modern-Cyptology\code\Curriculum_Design>
```

具体的文字输出如下：

```
E:\Maths\23S-Modern-Cyptology\code\Curriculum_Design>compile
```

```
E:\Maths\23S-Modern-Cyptology\code\Curriculum_Design>c++ -std=c++17 main.cpp aes128.cpp
aescbc.cpp sha1.cpp common.cpp RSA.cpp Certificate.cpp file.cpp NTLlib.a -I D:\Tools\WinNTL-
11_5_1\WinNTL-11_5_1\include -Wall -o main.exe
```

```
E:\Maths\23S-Modern-Cyptology\code\Curriculum_Design>main0<test
```

```
AESEncrypt [infilename] [outfilename]: AES encrypt 128 bit
```

```
AESDecrypt [infilename] [outfilename]: AES decrypt 128 bit
```

```
CBCEncrypt [infilename] [outfilename] [keyfilename]: AES encrypt CBC mode
```

```
CBCTransform [infilename] [outfilename] [keyfilename]: AES decrypt CBC mode
```

```
SHA1 [infilename]: SHA-1
```

```
RSASign [infilename] [outfilename] [keyfile] [keysize]: RSA encrypt
```

```
RSASign [infilename] [outfilename] [keyfile]: RSA decrypt
```

```
RSASign [messagefile] [signfile] [keyfile] [keysize]: RSA sign
```

```
RSASign [messagefile] [signfile] [keyfile]: RSA signature verify
```

```
generatesign [id] [pubkeyfile] [prikeyfile] [signfile] [keysize]: RSA signature verify
```

```
sendfile [infile] [outfile] [prikeyfile] [sendersign] [receiversign]: RSA signature verify
```

```
receivefile [infile] [outfile] [prikeyfile] [sendersign] [receiversign]: RSA signature verify
```

```
quit : exit
```

help : show help

E.g. AESencrypt infile.txt outfile.txt

RSAencrypt infile.txt outfile.txt keyfile.key 512

main.cpp aesEncrypt: AES encryption starts

main.cpp aesEncrypt: reading 01\_0

main.cpp aesEncrypt: reading key

main.cpp aesEncrypt: regenerating key

main.cpp aesEncrypt: encrypting

main.cpp aesEncrypt: success!

main.cpp aesDecrypt: AES decryption starts

main.cpp aesDecrypt: reading 01\_1

main.cpp aesDecrypt: reading key

main.cpp aesDecrypt: decrypting

main.cpp aesDecrypt: success!

main.cpp CBCencrypt: CBC encryption starts

main.cpp CBCdecrypt: generating key

main.cpp CBCencrypt: encrypting

main.cpp CBCencrypt: saving key

main.cpp CBCencrypt: success!

main.cpp CBCdecrypt: CBC decryption starts

main.cpp CBCdecrypt: decrypting

main.cpp CBCdecrypt: success!

main.cpp SHA1: reading infile

DA39A3EE5E6B4B0D3255BFEF95601890AFD80709

main.cpp SHA1: reading infile

86F7E437FAA5A7FCE15D1DDCB9EAEAEA377667B8

main.cpp SHA1: reading infile

A9993E364706816ABA3E25717850C26C9CD0D89D

main.cpp SHA1: reading infile

32D10C7B8CF96570CA04CE37F2A19D84240D3A89

main.cpp RSAencrypt: RSA encryption starts

main.cpp RSAencrypt: reading 04\_0

main.cpp RSAencrypt: encrypting

main.cpp RSAencrypt: success

main.cpp RSAdecrypt: RSA decryption starts

main.cpp RSAdecrypt: reading 04\_1

main.cpp RSAdecrypt: reading 04\_2

main.cpp RSAdecrypt: decrypting

success

main.cpp RSAsign: RSA signing starts

main.cpp RSAsign: reading 05\_0

main.cpp RSAsign: signing

main.cpp RSAsign: success

main.cpp RSAverify: RSA verifying starts

main.cpp RSAverify: reading 05\_0

main.cpp RSAverify: reading 05\_2

main.cpp RSAverify: reading 05\_1

Verified

main.cpp generatesign: generating sign



main.cpp generatesign: success

main.cpp generatesign: generating sign

main.cpp generatesign: success

main.cpp sendfile: Sending file

Send receiver certificate verified.

main.cpp sendfile: file sended to 06\_3 successfully

main.cpp receivefile: receiving file

sender verified

message verified

main.cpp receivefile: file received to 06\_4 successfully!

E:\Maths\23S-Modern-Cyptology\code\Curriculum\_Design>fc 01\_0 01\_2 /c /w

Comparing files 01\_0 and 01\_2

FC: no differences encountered

E:\Maths\23S-Modern-Cyptology\code\Curriculum\_Design>echo n | comp 02\_0 02\_3

Comparing 02\_0 and 02\_3...

Files compare OK

Compare more files (Y/N) ?

E:\Maths\23S-Modern-Cyptology\code\Curriculum\_Design>fc 04\_0 04\_3 /w

Comparing files 04\_0 and 04\_3

FC: no differences encountered

E:\Maths\23S-Modern-Cyptology\code\Curriculum\_Design>echo n | comp 06\_4 06\_0

Comparing 06\_4 and 06\_0...

Files compare OK

Compare more files (Y/N) ?

E:\Maths\23S-Modern-Cyptology\code\Curriculum\_Design>

经过测试不难发现，程序可以正确编译并通过 test 中的所有测试，其中包括 AES 加密解密、CBC 模式的 AES 加密解密、SHA-1 算法、RSA 的加密、解密和签名、验证，生成证书以及发送、接受消息。这样就初步地验证了程序的正确性。

#### d. 程序运行说明

直接运行程序，然后可以看到程序的输入提示：

此时，直接按照程序的提示进行输入，如果需要推出，既可以输入“quit”回车，也可以使用快捷键 Ctrl+C 直接退出而不会有什么不好的结果。

```
E:\Maths\23S-Modern-Cyptology\code\Curriculum_Design>main
AESencrypt [infile] [outfile]: AES encrypt 128 bit
AESdecrypt [infile] [outfile]: AES decrypt 128 bit
CBCencrypt [infile] [outfile] [keyfile]: AES encrypt CBC mode
CBCdecrypt [infile] [outfile] [keyfile]: AES decrypt CBC mode
SHA1 [infile]: SHA-1
RSAencrypt [infile] [outfile] [keyfile] [keysize]: RSA encrypt
RSAdecrypt [infile] [outfile] [keyfile]: RSA decrypt
RSAsign [messagefile] [signfile] [keyfile] [keysize]: RSA sign
RSAverify [messagefile] [signfile] [keyfile]: RSA signature verify
generatesign [id] [pubkeyfile] [prikeyfile] [signfile] [keysize]: RSA signature verify
sendfile [infile] [outfile] [prikeyfile] [sendersign] [receiversign]: RSA signature verify
receivefile [infile] [outfile] [prikeyfile] [sendersign] [receiversign]: RSA signature verify
quit : exit
help : show help
E.g. AESencrypt infile.txt outfile.txt
RSAencrypt infile.txt outfile.txt keyfile.key 512
```

假如希望使用 RSA 进行加密，加密内容位于文件“04\_0”，希望加密后的内容输出到“04\_1”，将密钥保存至“04\_2”，密钥的大小为 512bit 时，可以按照如图所示方式进行输入：

```
E:\Maths\23S-Modern-Cyptology\code\Curriculum_Design>main
AESencrypt [infile] [outfile]: AES encrypt 128 bit
AESdecrypt [infile] [outfile]: AES decrypt 128 bit
CBCencrypt [infile] [outfile] [keyfile]: AES encrypt CBC mode
CBCdecrypt [infile] [outfile] [keyfile]: AES decrypt CBC mode
SHA1 [infile]: SHA-1
RSAencrypt [infile] [outfile] [keyfile] [keysize]: RSA encrypt
RSAdecrypt [infile] [outfile] [keyfile]: RSA decrypt
RSAsign [messagefile] [signfile] [keyfile] [keysize]: RSA sign
RSAverify [messagefile] [signfile] [keyfile]: RSA signature verify
generatesign [id] [pubkeyfile] [prikeyfile] [signfile] [keysize]: RSA signature verify
sendfile [infile] [outfile] [prikeyfile] [sendersign] [receiversign]: RSA signature verify
receivefile [infile] [outfile] [prikeyfile] [sendersign] [receiversign]: RSA signature verify
quit : exit
help : show help
E.g. AESencrypt infile.txt outfile.txt
RSAencrypt infile.txt outfile.txt keyfile.key 512

RSAencrypt 04_0 04_1 04_2 512
main.cpp RSAencrypt: RSA encryption starts
main.cpp RSAencrypt: reading 04_0
main.cpp RSAencrypt: encrypting
main.cpp RSAencrypt: success

quit

E:\Maths\23S-Modern-Cyptology\code\Curriculum_Design>
```

其他指令也是类似，可以参考文件“test”中的指令进行输入输出。

每一个功能的输入输出格式如下表：

类型	输入	输出
AESencrypt	读入文件	输出文件
AESdecrypt	读入文件	输出文件
CBCencrypt	读入文件	输出密钥、密文两个文件
CBCdecrypt	读入密文、密钥文件	输出文件
SHA1	读入文件	控制台直接显示
RSAencrypt	读入文件、控制台输入密钥大小	输出一个密文文件，一个密钥文件
RSAdecrypt	读入一个密文文件、一个密钥文件	输出一个解密后的文件
RSAsign	读入文件、控制台输入密钥大小	输出一个签名文件和一个公钥文件
RSaverify	读入一个公钥文件、一个文件和一个文件签名文件	控制台直接输出是否通过验证
Generatesign	控制台读入 ID、密钥大小	将公钥、私钥、证书输出到 3 个文件中
Sendfile	从文件读入输入文件、发送者私钥、双方的签名文件	加密后的文件输出到一个文件中，证书是否验证通过输出到控制台中
receivefile	从文件中读入密文、接收者私钥、双方签名文件	解密后的文件输出到一个文件中，证书是否验证通过、文件的 hash 值是否与文件相符都输出到控制台中

具体每一个输入格式参考附件中的“test”，每一行都是一个指令，初始化的文件都以“\_0”结尾（hash 测试除外）。

附件文件说明：

本次的程序中一共有 15 个源文件，全部以“.h”“.cpp”结尾，分别为：

Certificate.h

common.cpp

common.h

file.cpp

file.h

main.cpp

RSA.cpp

RSA.h

sha1.cpp

sha1.h

aes128.cpp

aes128.h

aescbc.cpp

aescbc.h

Certificate.cpp

还有一个预先设置好的 NTL 静态编译库文件“NTLlib.a”。编译可以输入指令

```
c++ -std=c++17 main.cpp aes128.cpp aescbc.cpp sha1.cpp common.cpp RSA.cpp  
Certificate.cpp file.cpp NTLlib.a -I D:\Tools\WinNTL-11_5_1\WinNTL-  
11_5_1\include -Wall -o main.exe
```

其中“-I”需要改成 NTL 的 include 目录；此外还可以直接运行“compile.bat”进行编译和测试。

附件中以数字开头的文件均为测试示例所使用的文件。