

同济大学计算机系
现代密码学课程设计
说明书



学 号 2252552

姓 名 胡译文

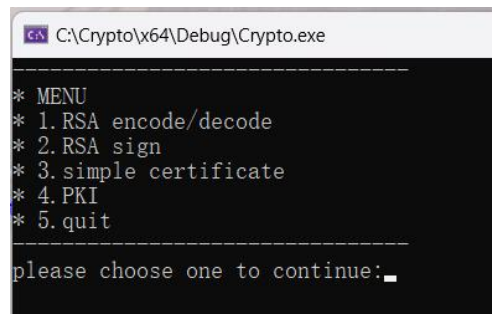
专 业 信息安全

目录

| | |
|--------------------|----|
| 一、 设计思路 | 1 |
| 二、 RSA 加密解密 | 1 |
| 三、 RSA 签名方案 | 5 |
| 四、 简单证书方案 | 7 |
| 五、 证书库 (PKI) | 11 |
| 六、 心得体会 | 15 |

一、设计思路

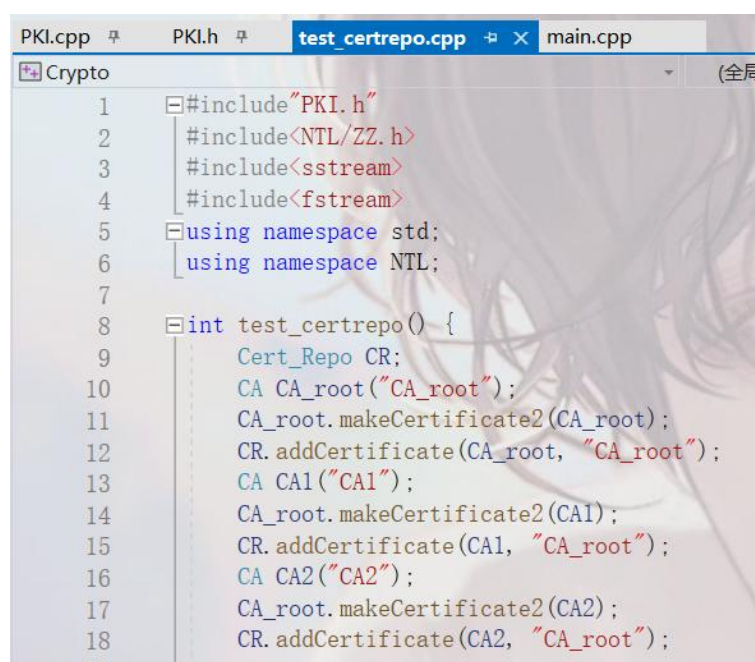
在主程序中设计一个菜单栏，可以选择四个任务项的测试模块。在光标处输入对应的数字即可转到相应的测试模块或退出，菜单项按顺序分别是：RSA 加/解密，RSA 签名，简单证书方案，PKI（或证书库），退出。



```
C:\Crypto\x64\Debug\Crypto.exe

* MENU
* 1.RSA encode/decode
* 2.RSA sign
* 3.simple certificate
* 4.PKI
* 5.quit
-----
please choose one to continue: _
```

每个测试模块函数单独写在 test_XXX.cpp 文件中，通过包含头文件的方式关联对应的功能模块。每个任务项都由 xxx.h 以及同名的 xxx.cpp 文件实现完整的功能。如下图为证书库任务的功能模块与测试模块。



```
PKI.cpp  PKI.h  test_certrepo.cpp  main.cpp
Crypto
1  #include "PKI.h"
2  #include <NTL/ZZ.h>
3  #include <sstream>
4  #include <fstream>
5  using namespace std;
6  using namespace NTL;
7
8  int test_certrepo() {
9      Cert_Repo CR;
10     CA CA_root("CA_root");
11     CA_root.makeCertificate2(CA_root);
12     CR.addCertificate(CA_root, "CA_root");
13     CA CA1("CA1");
14     CA_root.makeCertificate2(CA1);
15     CR.addCertificate(CA1, "CA_root");
16     CA CA2("CA2");
17     CA_root.makeCertificate2(CA2);
18     CR.addCertificate(CA2, "CA_root");
19 }
```

二、RSA 加密解密

1. NTL 库的配置和使用

1.1. NTL 简介

NTL 是一种高性能的可移植 C++ 库，提供用于处理带符号的任意长度整数以及整数和有限域上的矢量，矩阵和多项式的数据结构和算法。

1.2. NTL 的安装和配置

本次实验中采用的 NTL 版本为：WinNTL-11_5_1

将 NTL 安装好后，在 vs2019 中创建一个静态库新项目。将 NTL 源码包中 src 目

录的文件全部添加到工程中。

然后在工程上右键-属性-c++配置-常规，在附加包含目录中添加 NTL 源码包中 include 目录的路径，修改 SDL 检查为否。在 c++配置中选择预编译头，设置为不使用预编译头。然后正常编译即可，编译完成后会生成 .lib 文件。

使用该静态库时，需要和上述步骤一样设置附加包含目录，修改 SDL 检查。然后选择链接器-常规，在附加库目录中将 lib 的文件加入其中。最后选择链接器-输入，将生成的 lib 文件名加入到附加依赖项中。

2. RSA 算法的实现

2.1. 类的声明

```
#include<NTL/ZZ.h>
using namespace NTL;

const int PRIME_LEN1 = 512;
const int PRIME_LEN2 = 1024;
struct Public_key{
    ZZ n,b;
};
struct Private_key{
    ZZ p,q,a;
};
class RSA {
public:
    Public_key pub;//用于加密别人的公钥
public:
    RSA() {} ;
    RSA(const RSA&);
    void operator=(const RSA&);
    void setPublicKey(Public_key);
    bool GenerateKey(int l=PRIME_LEN1);
    ZZ encrypt(ZZ) const;
    ZZ decrypt(ZZ) const;
    Public_key GetPublicKey() const { return this->key.pub; }
    Private_key GetPrivateKey() const { return this->key.pri; }
private:
    struct Key{
        Public_key pub;
        Private_key pri;
    };
    Key key;
};
```

Pub : 其他 RSA 类传递来的公钥，setPublicKey()用于设置它

Key : RSA 类自己的密钥，包括公钥、私钥，GenerateKey()用于生成密钥

encrypt(): 用于 RSA 加密

decrypt(): 用于 RSA 解密

2.2. 成员函数的实现

2.2.1.生成密钥

```
bool RSA::GenerateKey(int key_len){
    ZZ p, q;
    int l=0;
    if (key_len != PRIME_LEN1 && key_len != PRIME_LEN2) { //检测密钥长度是否合理
        std::cout << "key length is not support" << std::endl;
        return false;
    }
    else
        l = key_len;
    //使用NTL库中的素数生成函数
    GenPrime(p, l);
    GenPrime(q, l);
    //计算 $\varphi(n)$ 
    ZZ phi;
    mul(phi, p - 1, q - 1);
    ZZ a, b, d;
    //获得随机素数b
    do {
        RandomBnd(b, phi);
        GCD(d, b, phi);
    } while (b <= 1 || d != 1);
    //b mod  $\varphi(n)$ 的逆元
    InvMod(a, b, phi);
    ZZ n; //求n
    mul(n, p, q);
    Public_key pub = { n, b };
    Private_key pri = { p, q, a };
    this->key.pub = pub;
    this->key.pri = pri;
    return true;
}
```

2.2.2.加密与解密

调用 NTL 库中的快速求幂函数计算

$$e_k(x) = x^b \bmod n \quad d_k(y) = y^a \bmod n$$

```
ZZ RSA::encrypt(ZZ x) const
{
    return PowerMod(x, this->pub.b, this->pub.n);
}

ZZ RSA::decrypt(ZZ y) const
{
    return PowerMod(y, this->key.pri.a, this->key.pub.n);
}
```






3. 测试方法

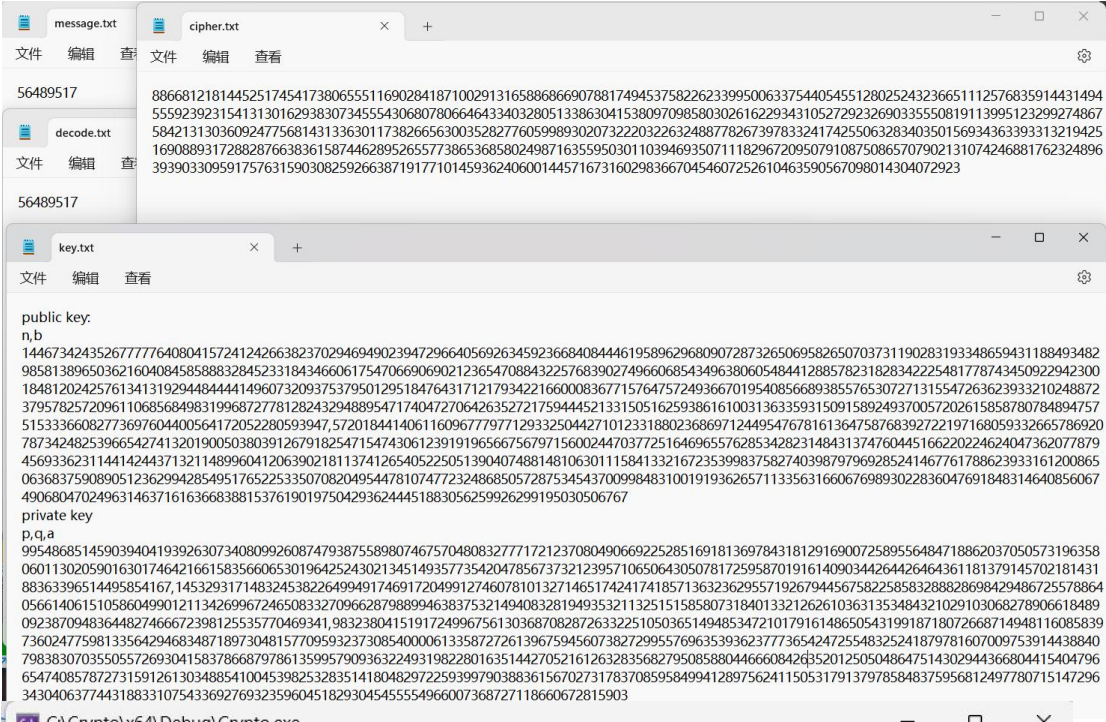
测试文件: test_rsa.cpp

测试函数: test_RSA()

输入格式: 在输入文件 message.txt 中写入一个大整数作为明文

输出格式：生成密文文件 cipher.txt，将生成的密文解密输出到文件 decode.txt，密钥储
存到 key.txt 中。

| | | | |
|---|-----------------|------------|------|
|  cipher.txt | 2024/8/20 23:47 | 文本文档 | 1 KB |
|  decode.txt | 2024/8/20 23:47 | 文本文档 | 1 KB |
|  key.txt | 2024/8/20 23:47 | 文本文档 | 3 KB |
|  main.cpp | 2024/8/20 20:59 | C++ Source | 1 KB |
|  message.txt | 2024/8/20 19:39 | 文本文档 | 1 KB |



```
* MENU
* 1. RSA encode/decode
* 2. RSA sign
* 3. simple certificate
* 4. PKI
* 5. quit

please choose one to continue:1
enter the length of key (512/1024):512
private key
p=9590120540025825879493331729995861995084483143300617996816349640943563725833138379359935
656759055884960379876479101406535302303945895505001481785199424267
q=1157943452828002288621345387585215093042095941718978553850896662103859680868696970861156
4643370482213831054802867443341773933478955911635601323069709058523
a=4187758454940421693975273948037184488462352936948148326728361771011931025670064422284037
557641902886156558261505768085518175145542813857098834886365465218355531714229181784597893
918622356029362256097169807460210521169093239774850929878671474966902742523027971290421789
2706472315476667099258637701433967549247
public key:
n=1110481729115425074334740112565391039268783404254558667572390486243773647912121669523238
150134850363975646806383213067960411590247199306452224003696398497879841556410173989893510
21304216731197096384776831102680231877544104560481385961758579078578685884088994829102777
73218284780693439052909690422454209377641
b=4563624298042948229578882215746115223000356043427405113752421603118047543766445353047379
974970505651882594422831766809809672685548725384541688747002672432263543827735658876270584
374660216955670972930704809235726557521957885820153537071696035415249212843117651098328097
3211455707498738160545017581132585683723
please choose one to continue:1
```



```

please choose one to continue:1
enter the length of key (512/1024):1024
private key
p=1055316183528801951884948281492077092355818491604127952193750933772934983699598582714655
886613633555915922976654413481333714384103582718311447525491118811010166240546057548112515
205361785977796717067845068645305535089383595070397179106109731301880526195747479339844033
15802698815287606853358970411067775870517
q=1643660975377654597988644317962743871066585491358486980289162341773406323865727041604093
130202107795811876133477376146631913007738309251081944607985602364576255140886255584959197
823218874631661874255445437307956784074346785374712851213152305019037103352801837419967400
83374190492985514417481962656021540539517
a=1694790992809392359761048261185699784803784254145891089312859443376885438791122644051463
350635983067177442873042721398904420882484375064545812619347457670849717016599826661209619
330353654614733021405656035585258982335288455325295177839836963429792348142911321120701770
803471452565920068902762167940382700736891865453895678742518923301415161234623004429260033
397791591576323286413721650463049286185574366590868351105268155961637296873521000332932160
773919187060784627199419663448017957404339139311760304854114446959047088172133037431651524
7514667642928847789091580001459601164853344050449006282381419597620706735048097
public key:
n=1734582027550774565858597267713643665288758135944596332512924233890206342433311231414240
526193484036865956122896161527429823745987795176583275098911410740371049185525523979994499
86418502224084646224952424494306979255344345544477485803147233162794276307770791275757874
841371637815527511302986448042337257849878079755662367745050636518600378926608545956978929
098241503503061123957009089906297089551667197651247624643901505933586688639776127911496996
621766000724593313253708792582657499251507325817360347224357226131736686892375723780234739
8973939151609812187670725889915457614963837688772004724482124573456170513720289
b=5258726926777356611555566350347431630537136834156455524048690355172115487813168843271337
852888289835552755934918138817467049160376555251862684612740778029761211990361610422235421
594804465924698130782950779932427652842500359246138489825686984393924311724388976903060478
447488359999725126946079332655542613722435681192123235887054445460706818706778596258949146
785514962811585725178725338285465911561699440783175768663964366153180732268153236884018933
421549218669160640807087449426220666909484436310863190242178478408986667463368004880701141
878710754865948043496964641037411948095545163126978221994449232261734389440417
please choose one to continue:

```

三、RSA 签名方案

1. RSA 签名算法

1.1. 类的声明

在 RSA 类的基础上，增加了 sig()签名函数和 ver()验证函数。

```

#ifndef RSA_SIG_H
#define RSA_SIG_H
#include "RSA.h"
#include <NTL/ZZ.h>
using namespace NTL;

class RsaSig{
public:
    bool ver(ZZ, ZZ) const;
    ZZ sig(ZZ) const;
public:
    RSA rsa;
};
#endif // !Sig

```

1.2. 成员函数的实现

调用 RSA 类中的加密函数进行签名, 调用解密函数进行验证。

```
#include "RsaSig.h"

bool RsaSig::ver(ZZ x, ZZ y) const
{
    return x == this->rsa.encrypt(y);
}

ZZ RsaSig::sig(ZZ x) const
{
    return this->rsa.decrypt(x);
}
```

2. 测试方法

测试文件: test_sig.cpp

测试函数: test_sig()

输入格式: 在输入文件 message.txt 中写入一个大整数作为明文

输出格式:

1. 对明文直接签名, 并验证该签名, 结果为正确, 签名写入 sign.txt, 验证结果写入 ver.txt (1 为验证通过, 0 为验证失败), 如图 1;

2. 修改第一步得到的签名, 并验证该签名, 结果为失败, 签名写入 sign2.txt, 验证结果写入 ver2.txt (1 为验证通过, 0 为验证失败), 如图 2;

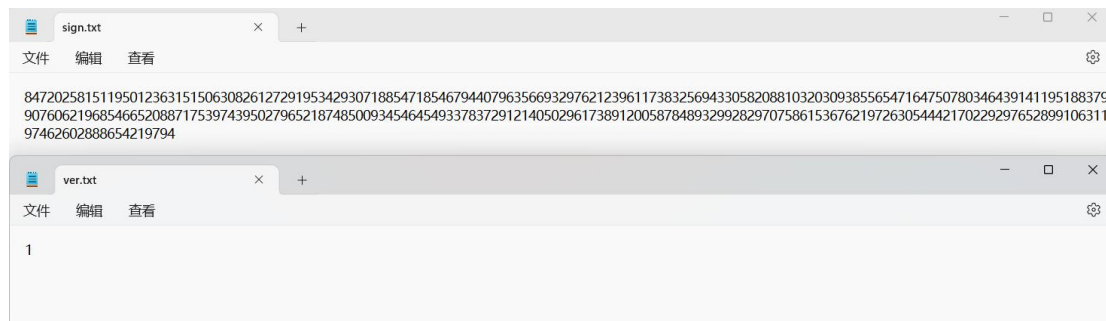


图 1

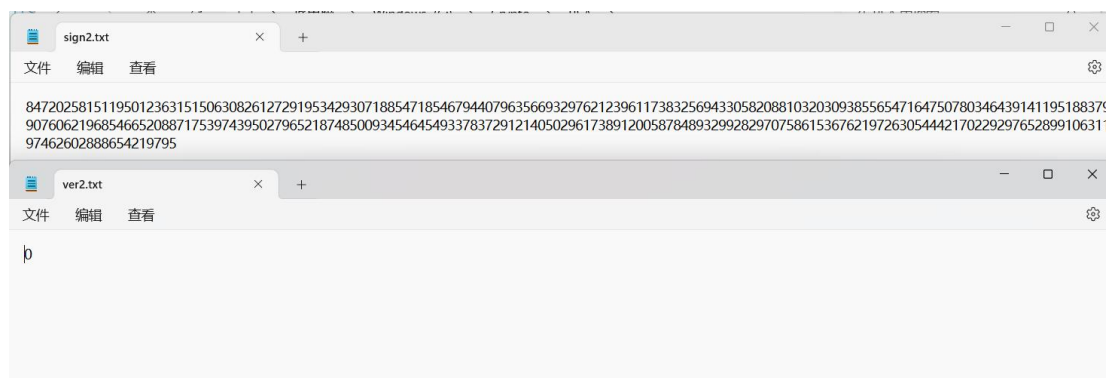


图 2

四、简单证书方案

1. 算法实现

1.1. TA 类的声明

```
class TA {
public:
    TA() {} ;
    TA(const string&);
    TA(int, const string&);
    void operator=(const TA&);
    string makeCertificate(Client&) const;
    string getID() const { return ID; }
    Public_key getPubTA() const { return this->sig.rsa.GetPublicKey(); }
private:
    RsaSig sig;
    string ID;
    int primeLen;
};
```

内置 RSA 签名类和 TA 的 ID，primeLen 用于记录密钥长度

makeCertificate(): 为 Client 对象生成证书

getID(): 类外访问 TA 的 ID

getPubTA(): 类外访问公钥

1.2. TA 类成员函数的实现

1.2.1. 构造函数

在构造函数中调用了 RSA 类中的密钥生成函数，使 TA 对象初始化时就能够被分配密钥，同时设定密钥长度为默认值（512）或选择的长度。

```
TA::TA(const string& id)
{
    primeLen = PRIME_LEN1;
    this->sig.rsa.GenerateKey();
    this->ID = id;
}

TA::TA(int len, const string& id)
{
    try {
        if (len == PRIME_LEN1 || len == PRIME_LEN2)
            primeLen = len;
        else
            throw string("prime length not allowed");
    }
    catch (string e)
    {
        cerr << e << endl;
    }
    this->sig.rsa.GenerateKey(len);
    this->ID = id;
}
```

1.2.2. 生成证书函数

首先需要给用户分配密钥，用自定义的格式转换函数获取密钥长度，接着按照证书算法的格式计算证书，并输出证书内容到文件。证书格式如 cert 所示。

```
string TA::makeCertificate(Client& user) const
{
    int pubKeyLen[2];
    user.sig.rsa.GenerateKey(primeLen);
    Public_key pub = user.sig.rsa.GetPublicKey();
    pubKeyLen[0] = ZZ2str(pub.b).length();
    pubKeyLen[1] = ZZ2str(pub.n).length();

    ZZ sigobj;
    sigobj = str2ZZ(user.getID() + ZZ2str(pub.b) + ZZ2str(pub.n));
    Public_key pubTA = this->sig.rsa.GetPublicKey();
    ZZ s = this->sig.sig(sigobj % pubTA.n);

    string cert =
        "Client:\n"
        + user.getID() + "\n"
        + "b:\n"
        + ZZ2str(pub.b) + "\n"
        + "n:\n"
        + ZZ2str(pub.n) + "\n"
        + "s:\n"
        + ZZ2str(s) + "\n"
        + "TA_ID:\n"
        + this->ID + "\n"
        + "b_len:\n"
        + int2str(pubKeyLen[0]) + "\n"
        + "n_len\n"
        + int2str(pubKeyLen[1]) + "\n";

    string dir = ".\\" + user.getID();
    if (_access(dir.c_str(), 0) == -1)
        mkdir(dir.c_str());
    string filename = ".\\" + user.getID() + "\\" + this->ID + "_cert.txt";
    ofstream fcert(filename, ios::out);
    try {
        if (!fcert.is_open())
            throw string("cert open error");
    }
    catch (string e)
    {
        cerr << e << endl;
    }
    fcert << cert;
    fcert.close();

    return cert;
}
```

1.3. Client 类的声明

```
class Client {
    friend class TA;
public:
    Client() {}
    Client(const string& id) { ID = id; }
    void operator=(const Client&);
    string getID() const { return ID; }

    void callCertificate(const TA&);
    bool verifyCertificate(const string&, const TA&);
    string getCertificate() const { return Certificate; }
    void writeLog(const string&, const string&, bool);
public:
    RsaSig sig;
private:
    string Certificate;
    string ID;
};
```

Client 包括证书的申请者或验证者，内置 RSA 签名类、Client 的 ID 和证书。

callCertificate(): 向 TA 申请证书

verifyCertificate(): 申请 TA 验证证书

writeLog(): 生成日志文件，用于保存验证结果

1.4. Client 类成员函数的实现

1.4.1. 验证证书函数

首先读证书文件，解析各项数据，获取用户 ID、公钥 b、n、签名 s，然后用 TA 的公钥对签名 s 进行验证。

```
bool Client::verifyCertificate(const string& cert, const TA& ta)
{
    string TName = ta.getID();
    stringstream stream(cert);
    string ID;
    getline(stream, ID);
    getline(stream, ID);

    string bstr, nstr;
    ZZ b, n;
    Public_key pub;
    getline(stream, bstr);
    getline(stream, bstr);
    getline(stream, nstr);
    getline(stream, nstr);
    b = str2ZZ(bstr);
    n = str2ZZ(nstr);
    pub.b = b;
    pub.n = n;

    string sstr;
    ZZ s;
    getline(stream, sstr);
    getline(stream, sstr);
    s = str2ZZ(sstr);
    ZZ sig_obj = str2ZZ(ID + bstr + nstr);
    Public_key pubTA = ta.getPubTA();
    this->sig.rsa.setPublicKey(pubTA);
    bool is_pass = this->sig.ver(sig_obj % pubTA.n, s % pubTA.n);
    if (is_pass)
        this->sig.rsa.setPublicKey(pub);
    writeLog(ID, TName, is_pass);
    return is_pass;
}
```

1.4.2. 申请证书函数

向引用的 TA 对象申请证书并保存在 Certificate 变量中。

```
void Client::callCertificate(const TA& ta)
{
    this->Certificate = ta.makeCertificate(*this);
}
```

1.5. 其他类型转换函数

```
string int2str(int n)
{
    stringstream s;
    s << n;
    return s.str();
}

int str2int(const string& str)
{
    int res(0);
    int len = str.length();
    for (int i = 0; i < len; i++)
        res = res * 10 + str[i] - '0';
    return res;
}
```



```

vector<std::string> stringSplit(const string& str, char delim) {
    stringstream ss(str);
    string item;
    vector<std::string> elems;
    while (getline(ss, item, delim)) {
        if (!item.empty()) {
            elems.push_back(item);
        }
    }
    return elems;
}

string ZZ2str(const ZZ& zz)
{
    stringstream s;
    s << zz;
    return s.str();
}

ZZ str2ZZ(const string& str)
{
    int len = str.length();
    ZZ res;
    res = 0;
    for (int i = 0; i < len; i++)
    {
        res = res * 10 + str[i] - '0';
    }
    return res;
}

```

2. 测试方法

测试文件：test_cert.cpp

测试函数：test_Certificate()

输入格式：无输入

输出格式：

- 1.控制台直接输出 Alice 验证 Bob 和 Oscar（伪造 Bob）的证书是否通过，如图 2；
- 2.将验证结果写入 ID 文件夹的 ver.log 中，如图 3；

```

Client Alice("Alice");
Client Bob("Bob");
Client Oscar("Bob");
//Alice.callCertificate(ta);
Bob.callCertificate(ta);
Oscar.callCertificate(ta2);

cout << "Alice verify Bob's Certificate" << endl;
if (Alice.verifyCertificate(Bob.getCertificate(), ta))
    cout << "pass" << endl;
else
    cout << "fail" << endl;

cout << "Alice verify Oscar's Certificate" << endl;
if (Alice.verifyCertificate(Oscar.getCertificate(), ta))
    cout << "pass" << endl;
else
    cout << "fail" << endl;

```

图 1

```
C:\Crypto\x64\Debug\Crypto.exe

* MENU
* 1.RSA encode/decode
* 2.RSA sign
* 3.simple certificate
* 4.PKI
* 5.quit
-----
please choose one to continue:3
Alice verify Bob's Certificate
pass
Alice verify Oscar's Certificate
fail
```

图 2

```
ver.log
文件 编辑 查看
date: 2024/8/20
time: 23:47:31
from:Bob
to: Alice
TA:Authority
status: fail
-----
date: 2024/8/26
time: 20:23:42
from:Bob
to: Alice
TA:Authority
status: pass
-----
date: 2024/8/26
time: 20:23:42
from:Bob
to: Alice
TA:Authority
status: fail
-----
行 48, 列 21 942 个字符
```

图 3

五、证书库（PKI）

1. 算法实现

1.1. CA 类与 USER 类

参照 TA 类与 Client 类编写，仅在 CA 类中加入了针对 CA 对象的生成证书函数 makeCert()，函数实现也与原生成函数相同，因此不在此赘述。

```

class CA {
    friend class Cert_Repo;
public:
    CA() {};
    CA(const string&);
    CA(int, const string&);
    void operator=(const CA&);
    string makeCert(USER&) const;
    void makeCertificate2(CA& ca);
    string getID() const { return ID; }
    Public_key getPubCA() const { return this->sig.rsa.GetPublicKey(); }
private:
    RsaSig sig;
    string ID;
    string Certificate;
    int primeLen;
};

```

1.2. Cert_Repo 类的声明

```

class Cert_Repo {
public:
    void addCertificate(const CA& ca, const string& pri_ca);
    void addCertificate(const USER& cl, const string& ca);
    bool verifyCertificate(vector<string>& path, const string& ownerID) const;
    vector<string> queryCertificatePath(const std::string& ownerID) const;
private:
    map<string, string> caCertMap; // Client/CA ID -> CA ID
    map<string, string> certificates; // ID -> certificate
};

```

包含两个 map，caCertMap 用于保存证书申请人的 ID 与权威机构 ID，certificates 用于保存 ID 与对应的证书。

addCertificate(): 添加证书与权威机构 ID 到两个 Map

verifyCertificate(): 验证证书路径

queryCertificatePath(): 查找证书路径

1.3. Cert_Repo 类成员函数的实现

1.3.1. addCertificate()

```

void Cert_Repo::addCertificate(const CA& ca, const string& pri_ca) {
    certificates[ca.ID] = ca.Certificate;
    if (ca.ID != pri_ca) {
        caCertMap[ca.ID] = pri_ca;
    }
}

void Cert_Repo::addCertificate(const USER& cl, const string& ca) {
    certificates[cl.ID] = cl.Certificate;
    caCertMap[cl.ID] = ca;
}

```


1.3.2. verifyCertificate()

按照最开始的证书 ID 在证书库中查找下一个 ID，同时与证书路径对比是否正确。

```
bool Cert_Repo::verifyCertificate(vector<string>& path, const string& ownerID) const {
    bool flag=true;
    string currentID = ownerID;
    string root = "CA_root";
    vector<string> temp=path;
    reverse(temp.begin(), temp.end());
    for (int i = 0; i < temp.size(); i++) {
        flag = temp[i] == certificates.at(currentID);
        if (currentID != root) {
            string caID = caCertMap.at(currentID);
            currentID = caID;
        }
    }
    return flag;
}
```

1.3.3. queryCertificatePath()

根据 ownerID 在证书库中查找下一个 ID。

```
vector<string> Cert_Repo::queryCertificatePath(const string& ownerID) const {
    vector<string> path;
    string currentID = ownerID;
    string root = "CA_root";
    if (certificates.find(currentID) == certificates.end())
        return path;
    while (currentID != root) {
        string caID = caCertMap.at(currentID);
        path.push_back(certificates.at(currentID));
        currentID = caID;
    }
    path.push_back(certificates.at(currentID));

    reverse(path.begin(), path.end());
    return path;
}
```

2. 测试方法

测试文件: test_certrepo.cpp

测试函数: test_certrepo()

输入格式: 在输入文件 message.txt 中写入一个大整数作为明文

输出格式:

1.控制台输出测试流程以及签名验证结果, 如图 3;

2.将证书路径写入 path.txt 中, 如图 4;

```
ifstream file("message.txt", ios::in);
ZZ x;
file >> x;
Alice.sig.rsa.GenerateKey();
Bob.sig.rsa.setPublicKey(Alice.sig.rsa.GetPublicKey());
ZZ sign = Alice.sig.sig(x);
```

图 1

```

ofstream path_file("path.txt", ios::out);
if (!path_file.is_open())
    cout << "sign fail" << endl;
vector<string> path;
cout << "Bob search Alice's certificate path" << endl;
path = CR.queryCertificatePath("Alice");
for (int i = 0; i < path.size(); i++) {
    path_file << path[i]<<endl;
}
path_file.close();

cout << "Bob verify Alice's certificate path" << endl;
if (CR.verifyCertificate(path, Alice.ID)) {
    cout << "certificate path right" << endl;
    bool ver = Bob.sig.ver(x, sign);
    if (ver)
        cout << "pass" << endl;
    else
        cout << "fail" << endl;
}
return true;

```

图 2

```

C:\Crypto\x64\Debug\Crypto.exe

* MENU
* 1. RSA encode/decode
* 2. RSA sign
* 3. simple certificate
* 4. PKI
* 5. quit

-----
please choose one to continue:4
Bob search Alice's certificate path
Bob verify Alice's certificate path
certificate path right
pass

```

图 3

```

path.txt
文件 编辑 查看

CA:
CA_root
b:
5388497019099538902664846613858153716610773481782158381648159210862630159532677831782100896135619690991
7104261152493106949806103184764925499037622805184313340462300104680773939667504189158786514186684055637
515443937839857780833643603389620513916775586508274388876544666699552949574937890946367032687801195493
n:
788115151051922623864522397243472330056743477311631180691214223845565502532421206298994641043193027768
5905783450520340568825269236359656528411703111929482236472196593420953955315769512884305252953766835724
935409872562751076907689412980154261599986349247978038183510846559872132168569177056622286807680953191
s:
4435560706522648320151822685371927321853100851399650634317313990938838402465663159131490545742670032552
4096936658559631988855509965291205623224525734876557226872658065636509712382664986548664918257417280847
126796343626578945673259939879782345741020097397806175369502079692783571092123130988473081649241935857
PRI CA_ID:
CA_root
b len:
308
n len:
308

CA:
CA1
b:
3381743511600113994281249102570489161088334000487474578427670692761543950241565323383236625601047986569
9612552783166131711615207352737145287538510460673353594142251400199024197324318337755078330320247029974
970978737209162347873964602728946751492431987659257724287787475578479811453931589642348323738483799815
n:
1257286070441636523910190924376251347147111095111724237485009074339999907359018457786398444211702925298
9612552783166131711615207352737145287538510460673353594142251400199024197324318337755078330320247029974
9801266368066266749780099925871714665392106648521295021945381758334153047937532845955984028560255524581
s:
632633598218037187406853726590287913758435887806593453250788810555936118675915352456482257232883676783
0113809650389997220594988181796359691197312880245838211490519690397833610056363356693942180474142120409
27030978534508900772311760344258321542300217356379331889067111648608934958537210142771252975864769988
PRI_CA_ID:

```

图 4

六、心得体会

本次课程设计涵盖了 RSA 加密、签名以及证书机制，最终融合形成了一个综合的证书库系统。这一过程不仅激发了我学习密码学的热情，而且加深了我对于各类加密算法和安全协议的认识，让我能够全面评估它们的长处与不足。通过使用 C++ 实现这些密码学实验，我不仅提高了自己在 C++ 编程方面的技能，还学习到了如何使用标准模板库 (STL) 中的容器，特别是 `bitset` 容器，以及流对象如 `stringstream` 和 `fstream`，还有数论库 NTL。

通过对密码学算法的深入理解和实现，我深刻体会到了在密码学领域内数学知识的核心地位和其重要性，这激发了我对数学的极大兴趣。在课余时间，我也主动探索了密码学的基础知识，这次实验经历可能是对我以后对其他专业课程学习有很大帮助。