

目 录

1 项目简介	1
1.1 项目概述	1
1.2 文件框架	1
2. RSA 加密解密	2
2.1 NTL 库的配置和使用	2
2.1.1 NTL 安装和配置	2
2.2 RSA 算法	2
2.2.1 RSA 加密解密算法	2
2.2.2 素数生成	2
2.2.3 密钥对生成	4
2.2.4 string 和 ZZ 的转换	4
2.2 使用说明	6
2.3.1 菜单 1: RSA 加密解密	6
2.2.3 菜单 2: RSA 解密(使用者提供私钥)	6
3 RSA 签名验证	8
3.1 RSA 签名方案:	8
3.1.1 RSA 签名方案的定义	8
3.2 设计说明	8
3.3 使用说明	8
3.3.1 菜单 3: RSA 签名算法	8
3.3.2 菜单 4: 验证 RSA 签名(使用者提供公钥)	9
4 证书方案	10
4.1 数字证书原理	10
4.2 设计说明	10
4.2.1 类封装	10
4.2.2 构造函数(签名生成)	11
4.2.3 验证签名函数	12
4.2.4 生成证书函数 void GenerateCertFile() const	12
4.2.5 主函数中的使用	12
4.3 使用说明	13
4.3.1 示例:	14
5 PKI 系统	16
5.1 PKI 系统原理	16
5.2 设计说明	16
5.2.1 整体框架	16
5.2.2 类设计	17
5.3.3 消息发送	18
5.3 使用说明	18
5.3.1 功能 1: 证书颁发	20
5.3.2 功能 2: 验证用户	21
5.3.3 功能 3: 证书显示	22
5.3.4 功能 4: 消息发送	23
5.3.5 功能 5: 消息验证	23
5.3.6 功能 6: PPT 案例演示	24

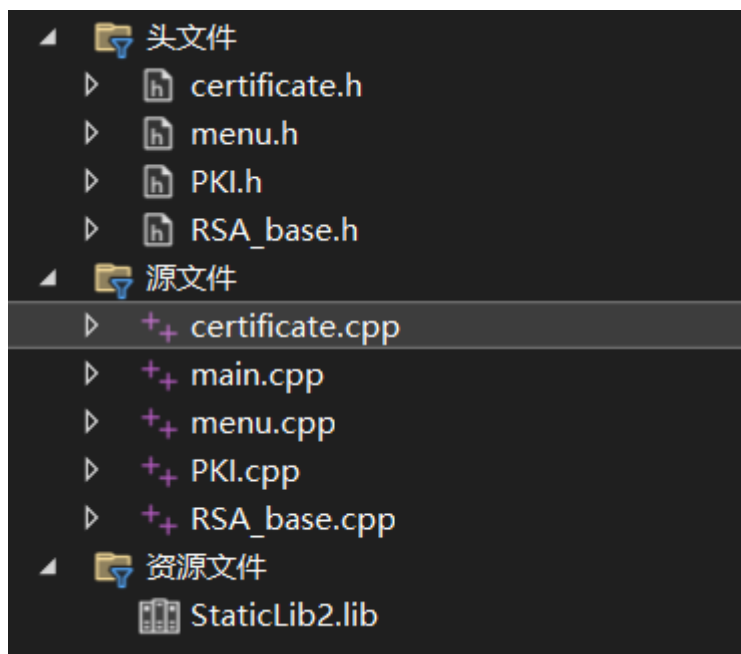
1 项目简介

1.1 项目概述

本项目是一个模拟的公钥基础设施（PKI）系统，提供了一系列基于 RSA 加密算法的功能，包括加密、解密、签名和验证签名等。项目通过一个主函数中的循环菜单与用户交互，允许用户选择不同的操作。以下是各个选项的功能概述：

1. RSA 加密解密：应用 RSA 加密算法对信息进行加密和解密。
 2. RSA 解密（使用者提供私钥）：用户可以提供自己的私钥来解密信息。
 3. RSA 签名算法：使用 RSA 算法进行数字签名。
 4. 验证 RSA 签名（使用者提供公钥）：用户可以提供公钥来验证数字签名的有效性。
 5. 一个简单的证书方案：实现一个基础的证书生成和验证流程，允许用户生成证书并验证签名。
 6. 严格层次 PKI 系统的简易版本：完整化模拟 PKI 并能模拟发送消息及其签名
- 整个系统设计为模块化，每个功能都是独立的模块，通过菜单选项触发相应的功能实现。项目提供了一个交互式的命令行界面，使用户能够方便地访问和使用 PKI 系统的各种功能。各个功能逐步递进，因此代码设计上也运用了继承的方法。

1.2 文件框架



前 4 个功能（PPT 中的前两个）主要函数再 RSA_base 中，此文件完成基本的签名验证加密解密等，这些函数也会在后续被多次使用，certificate 文件中构造证书类，完成一个简单的证书系统相关功能，PKI 文件中完成 pki 中需要使用的类。

2. RSA 加密解密

2.1 NTL 库的配置和使用

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。对于本项目而言，此库可提供素数生成，模运算等较复杂运算。

2.1.1 NTL 安装和配置

本次实验中采用的 NTL 版本为：WinNTL-11_5_1

将 NTL 源码从官网下载后，在 vs 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。

然后在项目上右键-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，并将 lib 添加到资源文件中。

参考教程为：[\(271 条消息\) NTL 库编译和使用配置_zfy1996 的博客-CSDN 博客_ntl 库编译。](#)

2.2 RSA 算法

2.2.1 RSA 加密解密算法

密码体制如下：

设 $n = pq$ ，其中 p, q 是素数

$$ab \equiv 1 \pmod{\varphi(n)}$$

则加密函数 $e(x) = x^b \pmod{n}$

解密函数 $d(y) = y^a \pmod{n}$

公钥为 (n, b) ，私钥为 (p, q, a) 。

分成如下模块：

1. 密钥生成；

1. 生成大素数

2. 生成密钥对 a, b ，满足 $\gcd(b, \varphi(n)) = 1$ ， $a = b^{-1} \pmod{\varphi(n)}$

2. 加密信息： $e(x) = x^b \pmod{n}$

3. 解密信息： $d(y) = y^a \pmod{n}$

4. string 和 ZZ（大整数）之间转换

2.2.2 素数生成

采用 Miller-Rabin 算法对随机生成的数进行检测是否为素数
算法流程

1. 输入：一个大整数 n ，要测试的次数 $iterations$ 。
2. 步骤：

边界条件检测:

如果 $n \leq 1$, 则 n 不是素数。

如果 $n = 2$, 直接判定为素数。

如果 n 是偶数且不等于 2, 则 n 不是素数。

将 $n-1$ 表示为: $n-1 = d \times 2^s$ 其中 d 是奇数, s 是正整数。具体步骤是不断将 $n-1$ 除以 2 直到无法再除, 得到奇数 d 和指数 s 。

计算: 计算 $x = a^d \bmod n$ 。

如果 $x \equiv 1 \pmod{n}$ 或 $x \equiv -1 \pmod{n}$, 则 n 可能是素数, 算法结束。

重复平方检测: 在接下来的 $s-1$ 步中, 计算 $x = x^2 \bmod n$ 。

如果在某次计算中, $x \equiv -1 \pmod{n}$, 则 n 可能是素数, 算法结束。

如果经过所有的平方检测, x 都不等于 $-1 \pmod{n}$, 则 n 是合数。

重复上述步骤: 对多个随机选择的基数 a 进行重复检测。如果所有的 a 都通过了测试, 则 n 是素数的可能性非常高; 否则, n 是合数。

输出: 根据以上测试结果, 返回 bool 值。

3. 代码:

```
bool Miller_Rabin(ZZ n, int iterations) {
    // 边界条件
    if (n <= 1) return false;
    if (n == 2) return true;
    if (n % 2 == 0) return false;

    // 将 n-1 分解为 d * 2^s
    ZZ d = n - 1;
    int s = 0;
    while (d % 2 == 0) {
        d /= 2;
        s++;
    }

    // 进行多次测试
    for (int i = 0; i < iterations; i++) {
        ZZ a = RandomBnd( n - 2); // 随机选择 a
        ZZ x = PowerMod(a, d, n); // 计算 x = a^d % n

        if (x == 1 || x == n - 1) continue; // 如果 x ≡ 1 或 x ≡ -1, 跳过本次测试

        bool passed = false;
        for (int r = 1; r < s; r++) {
            x = PowerMod(x, 2, n); // 计算 x = x^2 % n
            if (x == n - 1) {
                passed = true;
                break;
            }
        }
    }
}
```

```

    }
}

if (!passed) return false; // 如果所有平方检测都没有通过, n 是合数
}

return true; // 经过所有测试, n 可能是素数
}

```

算法分析:

算法错误地将合数 n 判断为素数的概率至多 $1/4$, 因此对 n 进行多次测试, 每次使用不同的随机基数 a 。进行 $iterations$ 次独立的测试, 则将合数 n 判断为素数的概率为 $\left(\frac{1}{4}\right)^{iterations}$

因本次使用 NTL 库, 可直接使用库中函数 `ZZ GeneratePrime(int bitLength)`, 输入为素数比特值。

2.2.3 密钥对生成

随机生成 $b < \varphi(n)$ 的数, 并检验其与 $\varphi(n)$ 互素, 否则重新随机生成, 代码如下:

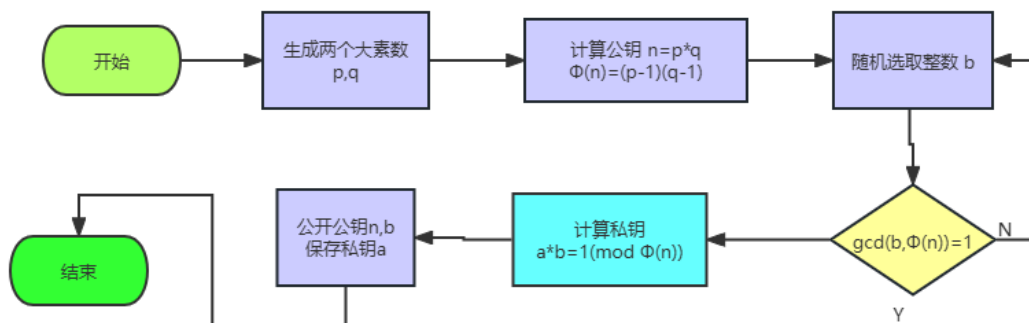
```

ZZ b_Generation(ZZ phi) {
    //srand(time(0));
    ZZ b;
    while (1) {
        RandomBnd(b, phi);
        if (GCD(b, phi) == 1)
            break;
    }
    return b;
}

```

生成对应的 a , 使用 `InvMod(a, b, phi)`; 含义为 $a = b^{-1} \bmod \phi$

整体流程图:



2.2.4 string 和 ZZ 的转换

考虑到需要加密的内容多为字符串形式, 因此需要将其转换为大整数, 以便加密运算, 处理思

路，同时也须将 `zz` 转换为 `string` 以便查看内容，思路为

1.初始化:

首先，复制输入的大整数数据，创建一个临时变量用于后续处理，并初始化一个空字符串用于存放最终结果。

2.循环处理:

只要临时变量的值不为零，继续执行以下操作:

取出临时变量的最低字节（即最后 8 位），并将这个字节转换成字符。

3.将这个字符添加到结果字符串的末尾。

将临时变量右移 8 位，以移除已经处理过的最低字节。

4.逆序字符串:

循环处理的过程中，字符是按照从最低有效字节到最高有效字节的顺序依次添加的，因此结果字符串需要进行逆序，以保证字节的排列顺序与原始大整数一致。

String 转为 zz:

1. 首先，定义一个结果变量，并将其初始值设为 0，这个变量将用于存储最终转换得到的大整数。

2. 接着，逐个遍历输入字符串中的每个字符。

3. 左移操作腾出空间:

在处理每个字符之前，将当前的结果左移 8 位（相当于乘以 256），为下一个字符的值腾出空间。这是因为每个字符在计算机中占 1 个字节（8 位）。

4.将字符的数值添加到结果中:

然后，将当前字符转换为无符号字符，获取其 ASCII 值，并将这个值加到结果中。这个步骤相当于把每个字符按顺序组合成一个大整数。

代码:

```
// 将字符串转换为 ZZ
ZZ StringToZZ(const string& data) {
    ZZ result(0);
    for (char ch : data) {

        result<<=8; // 乘以 256 以腾出空间
        result += conv<ZZ>(static_cast<unsigned char>(ch)); // 加入字符的
ASCII 值
    }
    return result;
}

// 将 ZZ 转换为字符串
string ZZToString(const ZZ& data) {
    ZZ temp = data;
    string result;
    while (temp !=0) {
```

```
        unsigned char ch = static_cast<unsigned
char>(conv<long>(temp %ZZ(256)));
        result += ch ; // 每次将当前字符加到结果字符串的最前面
        temp >>=8;
    }
    reverse(result.begin(), result.end());
    return result;
}
```

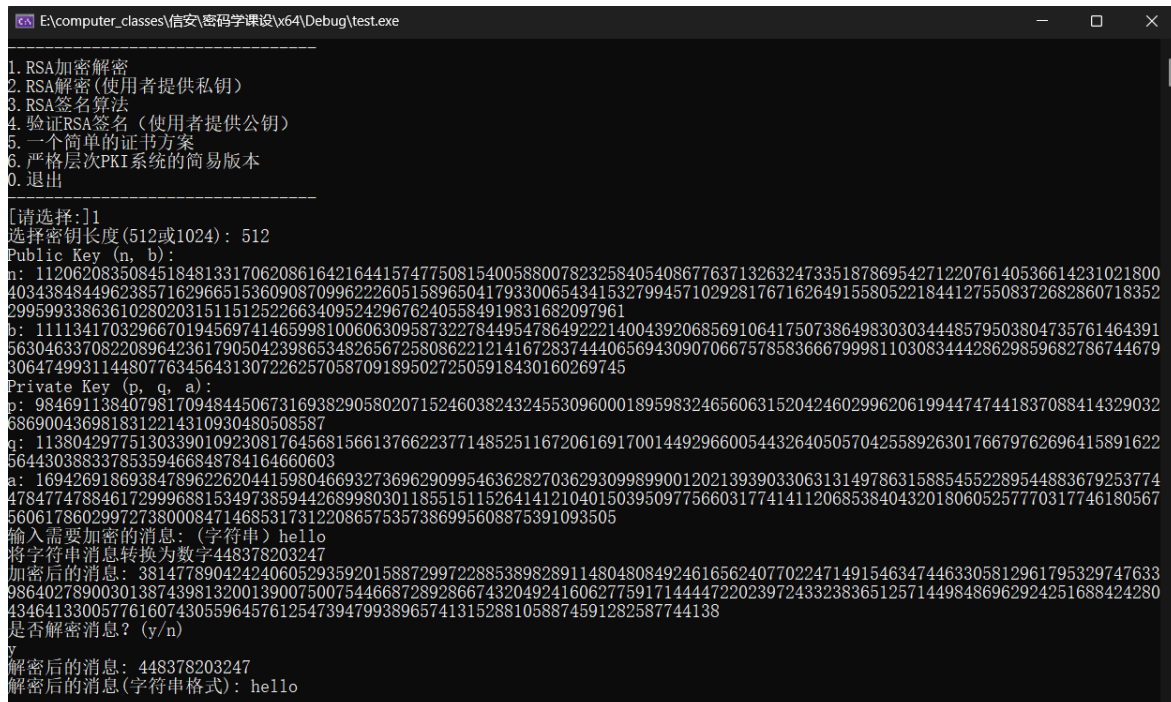
2.2 使用说明

2.3.1 菜单 1: RSA 加密解密

输入: 密钥长度 (512 或 1024), 生成密钥对

输入: 需加密信息, 生成加密后的大整数, 并可选择是否解密上面的大整数

演示界面:



```
E:\computer_classes\信安\密码学课设\64\Debug\test.exe
1. RSA加密解密
2. RSA解密(使用者提供私钥)
3. RSA签名算法
4. 验证RSA签名 (使用者提供公钥)
5. 一个简单的证书方案
6. 严格层次PKI系统的简易版本
0. 退出

[请选择:]1
选择密钥长度(512或1024): 512
Public Key (n, b):
n: 112062083508451848133170620861642164415747750815400588007823258405408677637132632473351878695427122076140536614231021800
403438484496238571629665153609087099622260515896504179330065434153279945710292817671626491558052218441275508372682860718352
299599338636102802031511512522663409524296762405584919831682097961
b: 111134170329667019456974146599810060630958732278449547864922214004392068569106417507386498303034448579503804735761464391
563046337082208964236179050423986534826567258086221214167283744406569430907066757858366679998110308344428629859682786744679
306474993114480776345643130722625705870918950272505918430160269745
Private Key (p, q, a):
p: 984691138407981709484450673169382905802071524603824324553096000189598324656063152042460299620619944747441837088414329032
6869004369818312214310930480508587
q: 113804297751303390109230817645681566137662237714852511672061691700144929660054432640505704255892630176679762696415891622
56443038833785359466848784164660603
a: 169426918693847896226204415980466932736962909954636282703629309989900120213939033063131497863158854552289544883679253774
478477478846172999688153497385944268998030118551511526414121040150395097756603177414112068538404320180605257770317746180567
56061786029972738000847146853173122086575357386995608875391093505
输入需要加密的消息: (字符串) hello
将字符串消息转换为数字448378203247
加密后的消息: 3814778904242406052935920158872997228853898289114804808492461656240770224714915463474463305812961795329747633
986402789003013874398132001390075007544668728928667432049241606277591714444722023972433238365125714498486962924251688424280
4346413300577616074305596457612547394799389657413152881058874591282587744138
是否解密消息? (y/n)
y
解密后的消息: 448378203247
解密后的消息(字符串格式): hello
```

2.2.3 菜单 2: RSA 解密(使用者提供私钥)

输入: 加密后的消息, 私钥 a, 公钥 n; 可得解密后的消息

```
-----
1. RSA加密解密
2. RSA解密(使用者提供私钥)
3. RSA签名算法
4. 验证RSA签名(使用者提供公钥)
5. 一个简单的证书方案
6. 严格层次PKI系统的简易版本
0. 退出
-----
[请选择:]2
请输入加密后的消息:
881477890424240605293592015887299722885389828911480480849246165624077022471491546347446330581296179532974763398640278900301
887439813200139007500754466872892866743204924160627759171444472202397243323836512571449848696292425168842428043464133005776
16074305596457612547394799389657413152881058874591282587744138
请输入私钥a:
169426918693847896226204415980466932736962909954636282703629309989900120213939033063131497863158854552289544883679253774478
477478846172999688153497385944268998030118551511526414121040150395097756603177414112068538404320180605257770317746180567560
51786029972738000847146853173122086575357386995608875391093505
请输入公钥n:
112062083508451848133170620861642164415747750815400588007823258405408677637132632473351878695427122076140536614231021800403
438484496238571629665153609087099622260515896504179330065434153279945710292817671626491558052218441275508372682860718352299
599338636102802031511512522663409524296762405584919831682097961
解密后的消息: 448378203247
解密后的消息(字符串格式): hello
-----
```

装

订

线

3 RSA 签名验证

3.1 RSA 签名方案：

3.1.1 RSA 签名方案的定义

一个签名方案是一个满足下列条件的五元组 (P, A, K, S, V)

P 是由所有可能的消息组成的一个集合

A 是由所有可能的签名组成的一个集合

K 为密钥空间，它是由所有可能的密钥组成的一个有限集合

对每一个 $k \in K$ ，有一个签名算法 $sig(k) \in S$ 和一个相应的验证算法 $ver(k) \in V$ 。其中对每一个消息 $x \in P$ 和每一个签名 $y \in A$ ，每个 $sig(k): P \rightarrow A$ 和 $ver: P \times A \rightarrow \{true, false\}$

由 $x \in P$ 和 $y \in A$ 组成的对 (x, y) 称为签名消息。

3.2 设计说明

在 RSA 加密解密的基础上，完成签名与验证，具体而言，签名即对需要签名的信息解密（使用公钥），验证即对签名加密后（使用私钥）与原始信息比对，具体代码如下：

```
// RSA 签名：使用私钥对消息进行签名
ZZ RSASign(const string& message, const ZZ& n, const ZZ& a) {
    ZZ m = StringToZZ(message); // 将消息转换为大整数
    return RSADecrypt(m, n, a); // 使用私钥对消息进行加密，生成签名
}

// RSA 签名验证：使用公钥验证签名
bool RSAVerify(const string& message, const ZZ& signature, const ZZ& n, const ZZ& b) {
    ZZ m = RSAEncrypt(signature, n, b);
    return (m == StringToZZ(message)); // 比较解密结果与原始消息是否一致
}
```

3.3 使用说明

3.3.1 菜单 3：RSA 签名算法

首先选择密钥长度，程序生成密钥对并显示：

输入需要签名的信息，输出生成的签名

选择是否验证签名，选择是则对刚才的签名进行验证，否则退出此选项

```
1. RSA加密解密
2. RSA解密(使用者提供私钥)
3. RSA签名算法
4. 验证RSA签名(使用者提供公钥)
5. 一个简单的证书方案
6. 严格层次PKI系统的简易版本
0. 退出

[请选择:]3
选择密钥长度(512或1024): 512
Public Key (n, b):
n: 135640968729837249637492670471411900976117133035305345091245013913098734496509260079101849648393528662282694510775832123
647096020168621716268198114145431652976198796651058663513206579672746291594287754088509467008827433450770711306714061803043
258411933720213684044824766737517851400657743692944267557467387493
b: 615887015931259128715355819171499793230017592352671598220590610551332975147930013417881081589466075970433381458819189035
489013808421385746792166900553542507661996445252761695746766622426400900214390383368331341560382342532762295839965837209698
1878308265172100476653356435687306425041826443806145619077286491
Private Key (p, q, a):
p: 10201582161773916672322461178140938599904019261911946312375272333026419657650065794980112356093078605550851237611408226
30472638463348680301721264803994367
q: 132960717836586169516341380651529485210122593930731272011606084662949511621168856647009399737816698519024066952399963437
63924978196551487305486970238247579
a: 689728616571465482870669697647042557771471045234942121587793057900890868334907398457993736541758924187938113681633714632
520989251652171087103026534428826594592791955493700451960444762186672251183586168451389907032527069465958131364111986928510
42803282681402674899304596460230533214080618406006404329175933359
输入需要签名的消息(字符串): hello world
将字符串消息转换为数字 126207244316550804821666916
生成的签名: 539698885825884766631690868719700109846004692449094667880233216955753158306475895209997825486155869684162230074191286310989
191286310989096672339151818926014877257178124626171809738863284854328824345270276768196549440674848249758203779945569469350281509586651
2815095866514388992077279778322952683745612938110911073124250905742255004
是否验证签名?(y/n)
y
签名验证结果: 有效
```

3.3.2 菜单 4: 验证 RSA 签名(使用者提供公钥)

输入待验证消息, 签名, 公钥 b, 公钥 n, 输出签名是否有效

```
1. RSA加密解密
2. RSA解密(使用者提供私钥)
3. RSA签名算法
4. 验证RSA签名(使用者提供公钥)
5. 一个简单的证书方案
6. 严格层次PKI系统的简易版本
0. 退出

[请选择:]4
请输入待验证消息:
hello world
请输入签名:
539698885825884766631690868719700109846004692449094667880233216955753158306475895209997825486155869684162230074191286310989
096672339151818926014877257178124626171809738863284854328824345270276768196549440674848249758203779945569469350281509586651
4388992077279778322952683745612938110911073124250905742255004
请输入公钥b:
615887015931259128715355819171499793230017592352671598220590610551332975147930013417881081589466075970433381458819189035489
013808421385746792166900553542507661996445252761695746766622426400900214390383368331341560382342532762295839965837209698187
8308265172100476653356435687306425041826443806145619077286491
请输入公钥n:
135640968729837249637492670471411900976117133035305345091245013913098734496509260079101849648393528662282694510775832123647
096020168621716268198114145431652976198796651058663513206579672746291594287754088509467008827433450770711306714061803043258
411933720213684044824766737517851400657743692944267557467387493
签名验证结果: 有效
```

下测试伪造签名的结果:

```
1. RSA加密解密
2. RSA解密(使用者提供私钥)
3. RSA签名算法
4. 验证RSA签名(使用者提供公钥)
5. 一个简单的证书方案
6. 严格层次PKI系统的简易版本
0. 退出

[请选择:]4
请输入待验证消息:
hello world
请输入签名:
52341087662347238478812308374627873333
请输入公钥b:
615887015931259128715355819171499793230017592352671598220590610551332975147930013417881081589466075970433381458819189035489
013808421385746792166900553542507661996445252761695746766622426400900214390383368331341560382342532762295839965837209698187
8308265172100476653356435687306425041826443806145619077286491
请输入公钥n:
135640968729837249637492670471411900976117133035305345091245013913098734496509260079101849648393528662282694510775832123647
096020168621716268198114145431652976198796651058663513206579672746291594287754088509467008827433450770711306714061803043258
411933720213684044824766737517851400657743692944267557467387493
签名验证结果: 无效
```

4 证书方案

4.1 数字证书原理

网络用户的证书包含用户的身份信息，公钥以及 TA 对这些消息的签名（本方案还包含 TA 的 ID 和公钥长度）。证书允许网络用户验证彼此公钥的真实性。一般地，我们假定有一个可信的权威机构 TA，为网络中的用户签署公钥，其实证书就是权威机构分发的签名。

简单证书方案如下：（向 Alice 颁布证书）

1. 生成证书：（有私钥的人才可）

- TA 生成公钥私钥对，私钥不写入证书但 Alice 知道，在证书系统中 Alice 需要私钥对传输的消息签名
- 对 Alice 的 ID 和公钥（包含 n,b）由生成的私钥进行签名，即 $s = \text{sig}_{\text{TA}}$ 的私钥 $(\text{ID}(\text{Alice}) \parallel \text{verAlice})$
- 将 Alice 的 ID，公钥，签名，TA 的 ID，公钥长度写入 TXT 文档形成证书

2. 验证证书：（任何人都可）

- 读取证书中的所有信息
- 使用证书中的公钥验证签名正确与否： verTA 的公钥 $(\text{ID}(\text{Alice}) \parallel \text{verAlice}, s) = \text{true}$

4.2 设计说明

4.2.1 类封装

为实现证书方案，创建了类 certification，具体如下：

```
class Certification {
protected:
    int flag; // 标识公钥长度
    string ID;
    ZZ n, b, p, q, a; // RSA 公私钥对
    string ID_ta;
    ZZ s; // s = sigTA 的私钥(ID || 公钥)
    string filepath;
public:
    Certification();
    Certification(int bitLength, const string& id);
    Certification(int bitLength, const string& id, const Certification& id_ta); // bitLength 为素数 p, q 长度 512 或 1024
    Certification(const string& filename);
    // 验证签名函数
    bool VerifySignature() const;
    // 生成 txt 文件函数
    void GenerateCertFile() const;
    void Display() const; // 显示证书信息
};
```

1. 类成员变量:

设置属性 `protected`: 表示这些成员变量在类及其派生类中是可访问的, 但在类外部不可访问。

`int flag`: 用于标识公钥长度。

`string ID`: 存储证书持有者的身份标识符。

`ZZ n, b, p, q, a`: 这些变量用于存储 RSA 加密算法中的公私钥对

`string ID_ta`: 存储签发者的身份标识符。

`ZZ s`: 存储签名。

`string filepath`: 存储证书文件的路径。

2. 构造函数:

`Certification(int bitLength, const string& id)`: 带参数的构造函数, 用于初始化证书的公钥长度、身份标识符。`bitLength` 是素数长度 512 或 1024 位。用于构造 TA。

`Certification(int bitLength, const string& id, const Certification& id_ta)` 用与构造普通用户

`Certification(const string& filename)`: 从文件中读取证书信息并初始化对象。

3. 成员函数:

`bool VerifySignature() const`: 用于验证数字签名的有效性。返回签名是否验证通过。

`void GenerateCertFile() const`: 生成一个包含证书信息的文本文件。

`void Display() const`: 显示证书的详细信息。

4.2.2 构造函数（签名生成）

其中, 构造函数有两种, 分别可通过设定信息和证书文件构造, 通过公钥长度、身份标识符和签发者标识符构造流程为:

1. 创建文件夹:

- 尝试创建文件夹 `folderPath=“certification”`。
- 如果创建成功或文件夹已存在则继续; 否则, 打印错误信息并结束流程。

2. 设置文件路径: 拼接文件路径 `filepath`, 格式为 `folderPath/id.txt`。

3. 设置标志: 将 `flag` 设置为 `bitLength`。

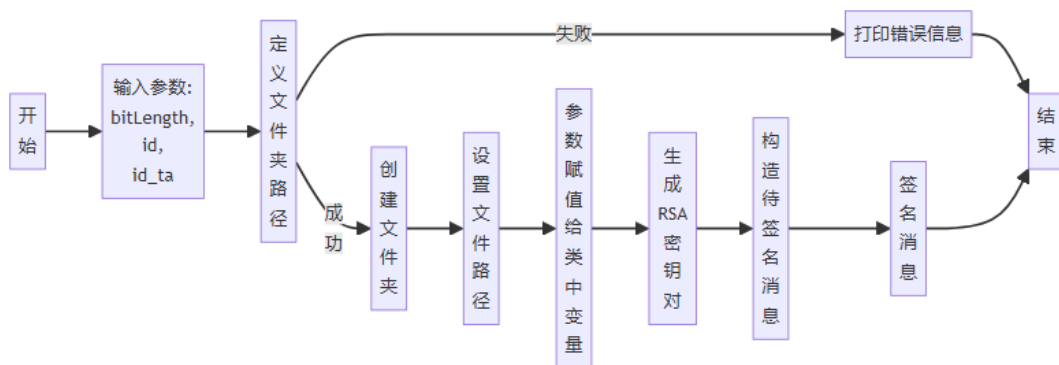
4. 设置身份标识符: 将输入的 `id` 和 `id_ta` 分别赋值给 `ID` 和 `ID_ta`。

5. 生成 RSA 密钥对: 调用 `GenerateRSAKeyPair` 函数, 传入 `n, b, p, q, a` 和 `bitLength`。

6. 构造待签名消息: 将 `ID` 与公钥参数 `n` 和 `b` 拼接成消息。

7. 签名消息: 调用 `RSASign` 函数, 传入构造的消息、TA 公钥 `n` 和 `ta` 私钥 `a`。将返回值赋值 `s`。

流程图:



4.2.3 验证签名函数

```

bool Certification::VerifySignature() const {
    string message = ID + ZZToString(n)+ ZZToString(b);
    Certification TA("Certification/" + ID_ta + ".txt");
    //TA.Display();
    return RSAVerify(message, s,TA.Getn(), TA.Getb());}
  
```

注意，需要验证的信息也是需要拼接的

4.2.4 生成证书函数 void GenerateCertFile() const

根据类中信息生成证书文件，证书文件格式如下：

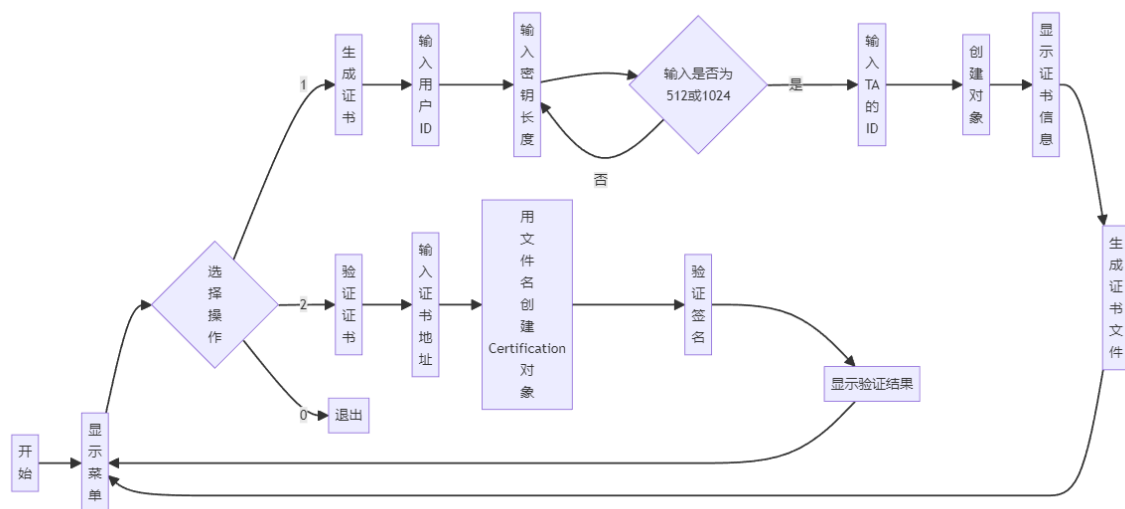
```

<ID>
<n>
<b>
<s(签名)>
<ID_ta>
<bitlength(密钥长度)>
  
```

4.2.5 主函数中的使用

分成两个功能生成证书和验证证书

整个流程为



4.3 使用说明

1. 选择要进行的操作，
2. 对于生成证书操作，需输入用户 ID，密钥长度，颁发机构 TA 的 ID，输出证书的相关信息并生成证书文件（两个证书）。
3. 对于验证证书操作，输入文件地址，输出验证结果

4.3.1 示例：

```
选择 E:\computer_classes\信安\密码学课设\64\Debug\test.exe
1. RSA加密解密
2. RSA解密(使用者提供私钥)
3. RSA签名算法
4. 验证RSA签名(使用者提供公钥)
5. 一个简单的证书方案
6. 严格层次PKI系统的简易版本
0. 退出

[请选择:]5

-----
1. 生成证书
2. 验证证书
0. 退出
-----

[请选择:]1
开始生成证书，请输入你的ID
alice
选择密钥长度(512或1024)：512
请输入证书颁发机构TA的ID
cal
TA证书
ID: cal
公钥 n: 5843059340870006130143669872711595817204307695238226318449209661705846950578928032843647751354658869361124148812093
777054020183924155972383270841877816034641815904571284694786947662295483640560396811300343800062685081323017505365952101541
0950988679977868627565549996424081090142074139999629552172194306387587
公钥 b: 9133916094475361298658188984220240760792076970573457695230717155420068420702799207213357246301902466826490609956729
455822770240262782637900227004344940150109538474934616867499208238402007128181658224085655161102437672401809863315467933531
210122677379728427239323858279817926453193904727613596308420148819249
签名 s: 5460153236820000984231406937917354122502866197879523525867536397309754809909810668441863439039583074976868016693227
05890682132814833962822112824462676526766646025687663216420961975078491935617548037434842200646491293966610544499165572568
2827624463633864814951313939179044181847414924145736409916827672361752
颁发机构 ID: cal
公钥长度标识 flag: 512
用户证书
ID: alice
公钥 n: 8949088298041271386061405787594276614396466412614529429670971124664042065628510039726400501895621163884974727385760
076140572930943347242400647412325167773311482445227399002831767442675615140322269461332190576832457388107938571687673010183
78738956135939236657222533166041141549769068879631041999725093876597
公钥 b: 1127309899736461437758125676221174633531779879385653293889140909370633180381435015759899982117812785579152607907047
893722986960758955899863543029936613396689969035544540408337308833314991259437127820540507687051802156840694024484438426615
3055632400506441216792065799715444442129494904428391941804103130356531
签名 s: 4965242738154179337428157515530683969064458790686577672604520142769987771857435487145103383432624555150224410480551
94594934014961118066643041072507696547218809238918706666122715634543154243025776555957317339286079455152827822717434999038
7981936702634644494997604270593154758753137591588832754879385079338032
颁发机构 ID: cal
公钥长度标识 flag: 512
证书已保存到 Certification/cal.txt 文件中。
证书已保存到 Certification/alice.txt 文件中。
```

Certification/alice..txt 如下图所示：

```
文件  编辑  查看
alice
894908829804127138606140578759427661439646641261452942967097112466404206562851003972640050189562116388
497472738576007614057293094334724240064741232516777331148244522739900283176744267561514032226946133219
05768324573881079385716876730101837873895613593392366572225331660411415497690688796310419997250938765
97
112730989973646143775812567622117463353177987938565329388914090937063318038143501575989998211781278557
915260790704789372298696075895589986354302993661339668996903554454040833730883331499125943712782054050
768705180215684069402448443842661530556324005064412167920657997154444421294949044283919418041031303565
31
496524273815417933742815751553068396906445879068657767260452014276998777185743548714510338343262455515
022441048055194594934014961111806664304107250769654721880923891870666612271563454315424302577655595731
733928607945515282782271743499903879819367026346444949976042705931547587531375915888327548793850793380
32
ca1
512
```

Certification/cal..txt 如下图所示

文件 编辑 查看



```
ca1
584305934087000613014366987271159581720430769523822631844920966170584695057892803284364775135465886936
112414881209377705402018392415597238327084187781603464181590457128469478694766229548364056039681130034
380006268508132301750536595210154109509886799778686275655499964240810901420741399996295521721943063875
87
913391609447536129865818898422024076079207697057345769523071715542006842070279920721335724630190246682
649060995672945582277024026278263790022700434494015010953847493461686749920823840200712818165822408565
516110243767240180986331546793353121012267737972842723932385827981792645319390472761359630842014881924
9
546015323682000098423140693791735412250286619787952352586753639730975480990981066844186343903958307497
686801669322705989068213281483396282211282446267652676664602568766321642096197507849193561754803743484
220064649129396661054449916557256828276244636338648149513139391790441818474149241457364099168276723617
52
ca1
512
```

验证证书:

```
1. 生成证书
2. 验证证书
0. 退出
```

```
[请选择:]2
开始验证证书, 请输入证书地址
Certification/alice.txt
证书正确
```

篡改证书内容后再验证 (篡改签名, 将签名删除部分内容)

篡改后证书 alice

```
alice
894908829804127138606140578759427661439646641261452942967097112466404206562851003972640050189562116388
497472738576007614057293094334724240064741232516777331148244522739900283176744267561514032226946133219
057683245738810793857168767301018378738956135933923665722225331660411415497690688796310419997250938765
97
112730989973646143775812567622117463353177987938565329388914090937063318038143501575989998211781278557
915260790704789372298696075895589986354302993661339668996903554454040833730883331499125943712782054050
768705180215684069402448443842661530556324005064412167920657997154444421294949044283919418041031303565
31
496524273815417933742815751553068396906445879068657767260452014276998777185743548714510338343262455515
022441048055194594934014961111806664304107250769654721880923891870666612271563454315424302577655595731
ca1
512
```

验证结果:

```
1. 生成证书
2. 验证证书
0. 退出
```

```
[请选择:]2
开始验证证书, 请输入证书地址
Certification/alice.txt
证书错误
```


5 PKI 系统

5.1 PKI 系统原理

PKI (Public Key Infrastructure, 公钥基础设施) 是一种用于管理和保护数字密钥的系统和协议的集合。它提供了一种安全的方式来创建、管理和分发数字证书。以下是 PKI 系统组成部分:

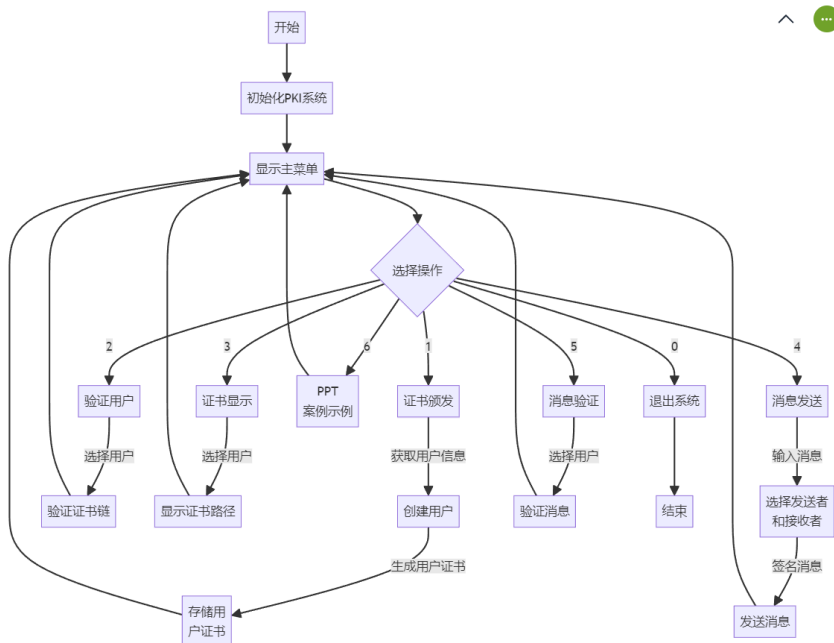
1. 密钥对生成:
 - 用户生成一对密钥, 包括一个公钥和一个私钥。
 - 公钥可以公开分享, 而私钥由用户保管。
2. 数字证书:
 - 数字证书是一个电子文档, 它将公钥与用户的身份信息绑定在一起。
 - 证书由可信的第三方, 即证书颁发机构 (CA) 签发。
3. 证书颁发机构 (CA):
 - CA 是负责签发和管理数字证书的机构。
 - 它验证用户的身份, 并使用自己的私钥对证书进行数字签名。
4. 证书库: 证书的集中存放地, 提供公众查询服务, 可查询公钥等。

5.2 设计说明

5.2.1 整体框架

因 pki 系统功能繁多, 因此用一个菜单将所有功能集合, 具体如下: 首先初始化 PKI 环境并创建根证书颁发机构 (CA), 然后允许用户输入中间 CA 的数量并为每个中间 CA 生成证书, 之后进入一个主菜单循环, 提供选项包括证书颁发、用户验证、证书显示、消息发送、消息验证以及 PPT 案例示例操作。

流程图如下:



5.2.2 类设计

User 类:

继承自 Certification 类，表示系统中的一个用户，可以是根 CA、中间 CA 或普通用户。

包含一个 UserType 枚举类型的成员变量 userType，用于标识用户类型。

包含一个 messageInfo 向量，存储消息文件名和发送用户 ID 的键值对。

多个构造函数，用于初始化不同类型的用户，包括根 CA、中间 CA 和普通用户，与 Certification 类类似。

DisplayUser，显示用户信息。

ChooseReceivedMessages，用于选择接收到的消息。

PKI 类:

私有成员 certStore，是一个 map，以用户 ID 为键，存储 User 对象，代表证书库。

InsertUser(string& your_ID, int& bitLength, User& CAid)，用于输入用户 ID 和密钥长度，并创建一个新的用户返回于 CAid 中。

StoreCertificate(const User& user)，将用户证书存储到证书库中。

DisplayCertificatePath(const string& userID)，显示指定用户的证书路径。

bool VerifyCertificateChain(const string& userID, bool is_display=true)，验证用户的证书链是否有效，并可选显示详细信息。

string ChooseUser(const string prmp) const;，提示用户选择一个用户。

void SendMessage(const string& senderID, const string& message, const string& receiverID);，模拟发送加密消息。

bool VerifyMessage(const string & receiverID);，验证消息的签名。

5.3.3 消息发送

在这个模块需要完成，创建消息文件夹，使用发送者私钥对消息进行签名，将消息和签名保存到文件中，代码如下：

```
void PKI::SendMessage(const string& senderID, const string& message, const
string& receiverID) {

    // 使用私钥 a 对消息进行签名
    cout << "请输入私钥" << endl;
    ZZ a_temp;
    cin >> a_temp;
    ZZ signature = RSASign(message, certStore[senderID].Getn(), a_temp);
    // 签名 = msgHash^a mod n

    // 将消息和签名保存到文件中
    string fileName;
    string folderPath = "Message";
    // 创建文件夹
    if (_mkdir(folderPath.c_str()) == 0 || errno == EEXIST) {
        // 拼接文件路径
        fileName = folderPath + "/" + certStore[senderID].GetID() + "_to_"
+ certStore[receiverID].GetID() ;
        while (FileExists(fileName + ".txt")) fileName += "()";
        fileName += ".txt";
    }
    else cerr << "创建文件夹失败" << endl;

    ofstream outFile(fileName,ios::binary);

    outFile << message << endl;
    outFile << signature << endl;
    outFile.close();

    // 将文件名和用户 ID 添加到 vector 中
    certStore[receiverID].messageInfo.push_back(make_pair(fileName,
certStore[senderID].GetID()));

    cout << "消息和消息的签名已保存到" << fileName << endl;
}
```

5.3 使用说明

进入 PKI 系统，首先初始化，自动生成 CAROOT 及其证书，随后可选需要几个中间 CA，并输入其 ID，程序生成相应证书存储。

示例:

```
-----
1. RSA加密解密
2. RSA解密(使用者提供私钥)
3. RSA签名算法
4. 验证RSA签名(使用者提供公钥)
5. 一个简单的证书方案
6. 严格层次PKI系统的简易版本
0. 退出
-----
[请选择:]6
正在初始化
证书已保存到 Certification/CAroot.txt 文件中。
选择中间CA个数
2
开始生成证书, 请输入CA的ID
cal
证书已保存到 Certification/cal.txt 文件中。
开始生成证书, 请输入CA的ID
ca2
证书已保存到 Certification/ca2.txt 文件中。
-----
```

证书如下;

1. CAROOT:

```
|CAroot
141564330937256547568040106616669537068173376693643603548591161461843134708969832399294174109873528664
330164982586842332417481351404438983748937980838477939993139100568160711821850684469245400829820895343
393181579851506073550520356278464183663895184367260408005384441851681999197483347327043482108340814660
699
929615311155617898651698821701004512426218879165409575271259271222566752890624331195809931063104102823
241651546164402640200364375360482042423054920960100337567652697805794021690920250840960777126644914378
621139755533140903843134165075086646026119181713231318616118997057920112906026635090698991902940334575
49
131049580457572061102398736384539732898152611531831664038662217243640375565181182815397089693098101171
954837915618432848749910097852642636745950982959038788069754286759047449172917243382667456672010571983
591051371349514237343957862626387879747739920006059643807915183183683209803312794768971017550090832984
440
CAroot
512
```

Cal

```
ca1
921938126154138052973433765575837333614023015351698610057228685253144193764734532896569994825469770985
570839122899990823336378797448999145219158482907575832909387672886994405313488213679760879944566828503
902621782726386229652758578198900054495195972555087185445711105477174122449987520322646309765075105628
31
326818126538132380613340936706326503017964405558232197198366684292889087690311967195768315602376362927
420060421478847414971112880342692725769313786224250298253311596223109176657234615871832892740773316443
374917241588742014426609140575329566160769523318612937046138877178010017054711894764430622451771463888
05
102272236498077941179467600401282678229337521942876532312496600821244486721700147612226245982869029500
522468432414485573902272831598368419234170759839047914473934757319566319886652893201191973389545062940
774405964583152060911589465297007799796169310769705470259001688716476291147371613664151849981030113715
116
CAroot
512
```

Ca2

```
ca2
949903023596385034080471031145039422915116674173027349008567100203112430514290380196890125867226813791
410915526950229935922333665860366256456026922569296133818766585306177100704403630700870512106409815598
561358683566874392912250802251326638083713839297327377983723212420642980258353467871029899384836423089
31
537209700300614580220867848506536158504547176262577247235301945940174111624723963684411099117900230804
896467852453715646335200696019343003697541911159272958306889469383034505918936554409384418797520455051
439472428852598113481753985363171522363519752444744283921232112589936489319018360499608420521659560801
45
742318314172278774624413941935941535940857318881462824062624766858582969912320241041790779757885007356
131145668394199562142869043271144132738795457012526505731326601762306196177706945604394766998655398569
225936170790065038718201023370044564377700720567816854416395000372791070850060804618288443767527742297
39
CAroot
512
```

5.3.1 功能 1：证书颁发

输入需要颁发证书的 ID，选择密钥长度和颁发 CA，生成证书，并输出私钥 a，以便后续消息发送

```
1. 证书颁发
2. 验证用户
3. 证书显示
4. 消息发送
5. 消息验证
6. ppt案例示例
0. 退出

[请选择:]1
开始生成证书，请输入你的ID
alice
选择密钥长度(512或1024)：512

1:ca1
2:ca2
请选择为你颁发机构的CA
1
User Type: User
ID: alice
公钥 n: 917633772057336627020205664686335627703516695482140683089331651498700554337404325716032552871536832908635227839887
104648511565324969100625119442516384840779719302383920583035223769792077478278695576019370118410773047629022410057543788622
4176202663798585786591097559307794619985689716063879274326676609336539
公钥 b: 250168007880553057285263710104304443248934262568098941396866037829866667679674346743400694430826203466554203912284
563708790161227319970285449677558782785640037500262818624304481205230184453751403108209121643869316481887922245754000709234
9800763555423796664801461257491494773240307460089497370032524184784901
签名 s: 4768842433625990402086111700068735349684184238773143354551501307825504382812120245234729543323770963302927972009699
425316210056734694471303163659849109598864875523591642587980099428355100420743346280820882305489412476069121972509750803425
4116219715374646398996948543555554313648840982762850391022810344205007
颁发机构 ID: ca1
公钥长度标识 flag: 512
私钥 a: 7580915028778251320428214855016809127880329289103232717092809590230602448474444782584806464883440121174123791343058
20687817516530975320231356722436425485586700966719221335209081515831140564005071432166058047813884995795212884360488604956
7870250556814354503669152214846137523419266953200194765407429126609101
证书已保存到 Certification/alice.txt 文件中。
```

证书文件如下:

```
alice
917633772705733662702020566468633562770351669548214068308933165149870055433740432571603255287153683290
863522783988710464851156532496910062511944251638484077971930238392058303522376979207747827869557601937
011841077304762902241005754378862241762026637985857865910975593077946199856897160638792743266766093365
39
25016800788055305728526371010434044432489342625680989413968660378298666767967434674340069443082620346
655420391228456370879016122731997028544967755878278564003750026281862430448120523018445375140310820912
164386931648188792224575400070923498007635554237966648014612574914947732403074600894973700325241847849
01
476884243362599040208611170006873534968418423877314335455150130782550438281212024523472954332377096330
292797200969942531621005673469447130316365984910959886487552359164258798009942835510042074334628082088
230548941247606912197250975080342541162197153746463989969485435555543136488409827628503910228103442050
07
ca1
512
```

同理生成用户 bob

```
1. 证书颁发
2. 验证用户
3. 证书显示
4. 消息发送
5. 消息验证
6. ppt案例示例
0. 退出
-----
[请选择:]1
开始生成证书, 请输入你的ID
bob
选择密钥长度(512或1024): 512
-----
1:ca1
2:ca2
请选择为你颁发机构的CA
2
User Type: User
ID: bob
公钥 n: 1361171646621868015231028530298377775293546204525912631928816087570100616811889215239833442208465802899619501400581
747811070604985587005918050488876203470878557368741863980083645870784412474679886253707091237856833851242586177149983029870
53108423571456102036553102952435037163748361186929385920141382386072333
公钥 b: 5075258500508941251964382329914334489230834042976709833317975531897541626951613198128021523254535090378375100671706
363305332321604783630807666221376854243017594398184408956662976101370165381559172813134333997717140030232540761660161128941
4843570665133069601514080947128050475754530938966880423867350178629831
签名 s: 8080372322757763734754069915298396973574969354193624713139670361560476347870980171255111513458900101487248578732059
940663537107697820952114132566094654964728964403924456958688961898186535313849601275613874552644237062048573161725688324850
3639200537907435293957732058282300179261992725629289449478911783161824
颁发机构 ID: ca2
公钥长度标识 flag: 512
私钥 a: 6501459912682286253997044144848020073720813363921369887281277738793770387514554514866795322475387757792680306821224
604029526515133568656490045619387738258425935733556406970434349364351296065085155465431128305819381289645633861548538856042
3202674702628875864961209221008517872999870310558087470056469031409571
证书已保存到 Certification/bob.txt 文件中。
```

5.3.2 功能 2: 验证用户

选择需要验证的用户, 输出其证书链, 和对每个证书的验证结果和证书内容, 示例如下:


```

=====
[请选择:]2
请选择需要验证的用户
=====
1. alice
2. bob
0. 退出
=====
[请选择:]1
-->alice-->cal-->CAroot
ID: alice
公钥 n: 9176337727057336627020205664686335627703516695482140683089331651498700554337404325716032552871536832908
04648511565324969100625119442516384840779719302383920583035223769792077478278695576019370118410773047629022410
176202663798585786591097559307794619985689716063879274326676609336539
公钥 b: 250168007880553057285263710104340443248934262568098941396866037829866667679674346743400694430826203466
63708790161227319970285449677558782785640037500262818624304481205230184453751403108209121643869316481887922245
9800763555423796664801461257491494773240307460089497370032524184784901
签名 s: 4768842433625990402086111700068735349684184238773143354551501307825504382812120245234729543323770963302
25316210056734694471303163659849109598864875523591642587980099428355100420743346280820882305489412476069121972
11621971537464639899694854355554313648840982762850391022810344205007
颁发机构 ID: cal
公钥长度标识 flag: 512
验证成功
=====
ID: cal
公钥 n: 9219381261541380529734337655758373336140230153516986100572286852531441937647345328965699948254697709855
908233363787974489991452191584829075758329093876728869944053134882136797608799445668285039026217827263862296527
6449519597255508718544571110547717412244998752032264630976507510562831
公钥 b: 3268181265381323806133409367063265030179644055582321971983666842928890876903119671957683156023763629274
74149711128803426927257693137862242502982533115962231091766572346158718328927407733164433749172415887420144266
6616076952331861293704613887717801001705471189476443062245177146388805
签名 s: 1022722364980779411794676004012826782293375219428765323124966008212444867217001476122262459828690295005
855739022728315983684192341707598390479144739347573195663198866528932011919733895450629407744059645831520609115
9796169310769705470259001688716476291147371613664151849981030113715116
颁发机构 ID: CAroot
公钥长度标识 flag: 512
验证成功
=====
ID: CAroot
公钥 n: 14156433093725654756804010661666953706817337669364360354859116146184313470896983239929417410987352866433
4233241748135140443898374893798083847793999313910056816071182185068446924540082982089534339318157985150607355052
83663895184367260408005384441851681999197483347327043482108340814660699
公钥 b: 92961531115561789865169882170100451242621887916540957527125927122256675289062433119580993106310410282324
0264020036437536048204242305492096010033756765269780579402169092025084096077712664491437862113975553314090384313
4602611918171323131861611899705792011290602663509069899190294033457549
签名 s: 13104958045757206110239873638453973289815261153183166403866221724364037556518118281539708969309810117193
3284874991009785264263674595098295903878806975428675904744917291724338266745667201057198359105137134951423734393
79747739920006059643807915183183683209803312794768971017550090832984440
颁发机构 ID: CAroot
公钥长度标识 flag: 512
验证成功
=====

```

5.3.3 功能 3：证书显示

选择需要验证的用户，输出其证书链，和每个证书内容，示例如下：

```

=====
1. alice
2. bob
0. 退出
=====
[请选择:]2
bob-->ca2-->CAroot-->
ID: bob
公钥 n: 13611716466218680152310285302983777752935462045259126319288160875701006168118892152398334422084658028996
7478110706049855870059180504888762034708785573687418639800836458707844124746798862537070912378568338512425861773
53108423571456102036553102952435037163748361186929385920141382386072333
公钥 b: 50752585005089412519643823299143344892308340429767098333179755318975416269516131981280215232545350903783
3633053323216047836308076662213768542430175943981844089566629761013701653815591728131343339977171400302325407616
4843570665133069601514080947128050475754530938966880423867350178629831
签名 s: 80803723227577637347540699152983969735749693541936247131396703615604763478709801712551115134589001014877
9406635371076978209521141325660946549647289644039244569586889618981865353138496012756138745526442370620485731617
3639200537907435293957732058282300179261992725629289449478911783161824
颁发机构 ID: ca2
公钥长度标识 flag: 512
=====

```

```
ID: ca2
公钥 n: 94990302335963850340804710311450394229151166741730273490085671002031124305142903801968901258672268137914109155269502
299359223336658603662564560269225692961338187665853061771007044036307008705121064098155985613586835668743929122508022513266
3808371383929732737798372321242064298025835346787102989938483642308931
公钥 b: 5372097003006145802208678485065361585045471762625772472353019459401741116247239636844110991179002308048964678524537
156463352006960193430036975419111592729583068894693830345059189365544093844187975204550514394724288525981134817539853631715
2236351975244474428392123211258993648931901836049960842052165956080145
签名 s: 7423183141722787746244139419359415359408573188814628240626247668585829699123202410417907797578850073561311456683941
995621428690432711441327387954570125265057313266017623061961777069456043947669986553985692259361707900650387182010233700445
6437770072056781685441639500037279107085006080461828844376752774229739
颁发机构 ID: CAroot
公钥长度标识 flag: 512

ID: CAroot
公钥 n: 1415643309372565475680401066166695370681733766936436035485911614618431347089698323992941741098735286643301649825868
423324174813514044389837489379808384779399931391005681607118218506844692454008298208953433931815798515060735505203562784641
83663895184367260408005384441851681999197483347327043482108340814660699
公钥 b: 9296153111556178986516988217010045124262188791654095752712592712225667528906243311958099310631041028232416515461644
026402003643753604820424230549209601003375676526978057940216909202508409607771266449143786211397555331409038431341650750866
4602611918171323131861611899705792011290602663509069899190294033457549
签名 s: 1310495804575720611023987363845397328981526115318316640386622172436403755651811828153970896930981011719548379156184
828487499100978526426367459509829590387880697542867590474491729172433826674566720105719835910513713495142373439578626263878
79747739920006059643807915183183683209803312794768971017550090832984440
颁发机构 ID: CAroot
公钥长度标识 flag: 512
```

5.3.4 功能 4：消息发送

输入需要加密的消息，选择消息接收者和发出者，输入消息发出者的私钥，生成消息文件

```
. 证书颁发
. 验证用户
. 证书显示
. 消息发送
. 消息验证
. ppt案例示例
. 退出
-----
请选择:4
输入需要加密的消息: (字符串) hello world
llo world
CAroot
alice
bob
cal
ca2
请选择消息接收者

CAroot
alice
bob
cal
ca2
请选择你是谁 (消息发送者)

请输入私钥
58091502877825132042821485501680912788032928910323271709280959023060244847444478258480646488344012117412379134305820687817
16530975320323135672243642548558670096671922133520908151583114056400507143216605804781388499579521288436048860495678702505
6814354503669152214846137523419266953200194765407429126609101
消息和消息的签名已保存到Message/alice to bob.txt
```

5.3.5 功能 5：消息验证

选择查看谁的消息，显示其收到的消息，选择验证哪一条消息，显示内容和验证相关过程和结果。


```
-----
1. 证书颁发
2. 验证用户
3. 证书显示
4. 消息发送
5. 消息验证
6. ppt案例示例
0. 退出
-----
[请选择:]5
1 Croot
2 alice
3 bob
4 cal
5 ca2
你要查看谁的消息:
3
收到的消息:
1 消息文件名: Message/alice_to_bob.txt | 发送者: alice
请选择要详细查看的序号:
1
信息内容: ello world
签名: 679191550528466057102359223963587765106326763128609982138133430971476056599084540678037649418541783759037290899080387
516567917099236009069964178842170452710004549945402065065304017126729428419697697920814598990350901651331961039126545440518
48457249030431929561617197628118472816021568407968718549536938354354
现在开始验证签名...
查询 发送者alice 的证书路径并验证
-->alice-->cal-->Croot
ID: alice
公钥 n: 9176337727057336627020205664686335627703516695482140683089331651498700554337404325716032552871536832908635227839887
104648511565324969100625119442516384840779719302383920583035223769792077478278695576019370118410773047629022410057543788622
4176202663798585786591097559307794619985689716063879274326676609336539
公钥 b: 250168007880553057285263710104340443248934262568098941396866037829866667679674346743400694430826203466554203912284
563708790161227319970285449677558782785640037500262818624304481205230184453751403108209121643869316481887922245754000709234
9800763555423796664801461257491494773240307460089497370032524184784901
签名 s: 4768842433625990402086111700068735349684184238773143354551501307825504382812120245234729543323770963302927972009699
425316210056734694471303163659849109598864875523591642587980099428355100420743346280820882305489412476069121972509750803425
411621971537464639899694854355554313648840982762850391022810344205007
颁发机构 ID: cal
公钥长度标识 flag: 512
验证成功
-----
ID: cal
公钥 n: 9219381261541380529734337655758373336140230153516986100572286852531441937647345328965699948254697709855708391228999
908233363787974489991452191584829075758329093876728869944053134882136797608799445668285039026217827263862296527585781989000
5449519597255508718544571110547717412244998752032264630976507510562831
公钥 b: 3268181265381323806133409367063265030179644055582321971983666842928890876903119671957683156023763629274200604214788
474149711128803426927257693137862242502982533115962231091766572346158718328927407733164433749172415887420144266091405753295
6616076952331861293704613887717801001705471189476443062245177146388805
签名 s: 1022722364980779411794676004012826782293375219428765323124966008212444867217001476122262459828690295005224684324144
655739022728315983684192341707598390479144739347573195663198866528932011919733895450629407744059645831520609115894652970077
99796169310769705470259001688716476291147371613664151849981030113715116
颁发机构 ID: Croot
公钥长度标识 flag: 512
验证成功
-----
ID: Croot
公钥 n: 1415643309372565475680401066166695370681733766936436035485911614618431347089698323992941741098735286643301649825868
423324174813514044389837489379808384779399931391005681607118218506844692454008298208953433931815798515060735505203562784641
83663895184367260408005384441851681999197483347327043482108340814660699
公钥 b: 9296153111556178986516988217010045124262188791654095752712592712225667528906243311958099310631041028232416515461644
026402003643753604820424230549209601003375676526978057940216909202508409607771266449143786211397555331409038431341650750866
4602611918171323131861611899705792011290602663509069899190294033457549
签名 s: 1310495804575720611023987363845397328981526115318316640386622172436403755651811828153970896930981011719548379156184
328487499100978526426367459509829590387880697542867590474491729172433826674566720105719835910513713495142373439578626263878
79747739920006059643807915183183683209803312794768971017550090832984440
颁发机构 ID: Croot
公钥长度标识 flag: 512
验证成功
-----
证书链无误
消息验证无误
-----
```

5.3.6 功能 6: PPT 案例演示

此功能使将上述操作统一起来，省去各项输入，以便检查，具体内容与上述相似。