

---

# 同济大学计算机系

## 现在密码学课程设计说明书



学 号 \_\_\_\_\_

姓 名 \_\_\_\_\_

专 业 \_\_\_\_\_ 信息安全 \_\_\_\_\_

授课老师 \_\_\_\_\_ 杨礼珍 \_\_\_\_\_

---

## 目录

|    |                         |    |
|----|-------------------------|----|
| 一、 | RSA 加密和解密算法 .....       | 3  |
| 二、 | RSA 签名算法.....           | 7  |
| 三、 | 实现一个简单的证书方案.....        | 11 |
| 四、 | 实现严格层次 PKI 系统的简易版本..... | 15 |

# 一、RSA 加密和解密算法

## 0. NTL 库的配置和使用

官方网站: [NTL: 用于进行数论的库](#)

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。

本次实验中采用的 NTL 版本为: WinNTL-11\_5\_1

将 NTL 安装好后, 在 vs2022 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项目源文件中。

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 文件的路径加入其中。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。

详细教程参考了 [《NTL 库》使用教程 \(C++ 现代密码学\) | isSeymour](#)

## 1. 程序设计说明

### 1.1 原理

RSA 参数生成算法:

- 1) 生成两个随机大素数  $p, q$  (使用素性检测算法)
- 2)  $n \leftarrow pq, \phi(n) \leftarrow (p-1)(q-1)$
- 3) 选择一个随机数  $b$  ( $1 < b < \phi(n)$ ), 使  $\gcd(b, \phi(n))=1$  (使用 Euclidean 算法判断)
- 4)  $a \leftarrow b^{-1} \bmod \phi(n)$  (使用扩展 Euclidean 算法计算)
- 5) 公钥为  $(n, b)$ , 私钥为  $(p, q, a)$

RSA 的加密和解密:

- 1) 加密:  $y \leftarrow x^b \bmod n$ 。(使用平方乘算法)
- 2) 解密:  $x \leftarrow y^a \bmod n$ 。(使用平方乘算法)

### 1.2 文件结构

#### 1.RSA 加密解密

|—RSA.h //RSA 类的定义及函数声明  
|—RSA.cpp //RSA 类函数的实现  
|—main.cpp //具体加密解密流程的实现  
|—message.txt //被加密的明文, 在 main 函数中文件输入

### 1.3 算法实现

#### ①生成密钥

使用了 NTL 库中的以下函数:

void RandomPrime(ZZ& n, long l, long NumTrials=10) 生成大素数 pq

`void RandomBnd(ZZ& x, const ZZ& n)` 生成随机数  $b$ ，需要注意生成的随机数范围是 $[0 \sim n-1]$ ，而  $b$  的范围是 $(1, \varphi(n))$

`void InvMod(ZZ& x, const ZZ& a, const ZZ& n)` 求  $b$  的模  $\varphi(n)$  逆元  $a$

②加密&解密：

`inline ZZ PowerMod(const ZZ& a, const ZZ& e, const ZZ& n)` 计算模  $n$  指数运算

为了避免数值过大导致溢出错误，在传入参数时对  $a$  进行模  $n$  处理。

③主函数运行流程

在控制台窗口选取密钥长度：512/1024

声明 RSA 变量，自动生成 RSA 密钥

从文件中读取明文

运行加密算法，加密结果输出文件

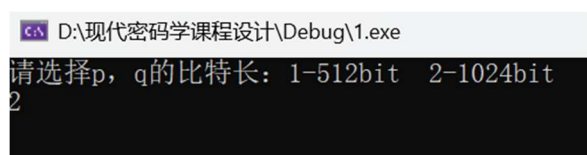
运行解密算法，解密结果输出文件

从加密结果文件中读取密文，运行解密算法，解密结果输出文件

查看目录下的 txt 文件，比对结果

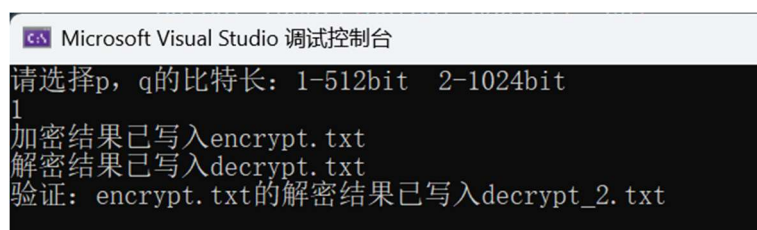
## 2. 程序使用说明

①选择密钥长度，根据提示语句进行选择：



选择 512bit 约在 10s 内出现下一行提示文字，1024bit 则更长。

②自动读取文件目录下的 message.txt 作为输入，加密结果写入 encrypt.txt，解密结果写入 decrypt.txt，decrypt\_2.txt



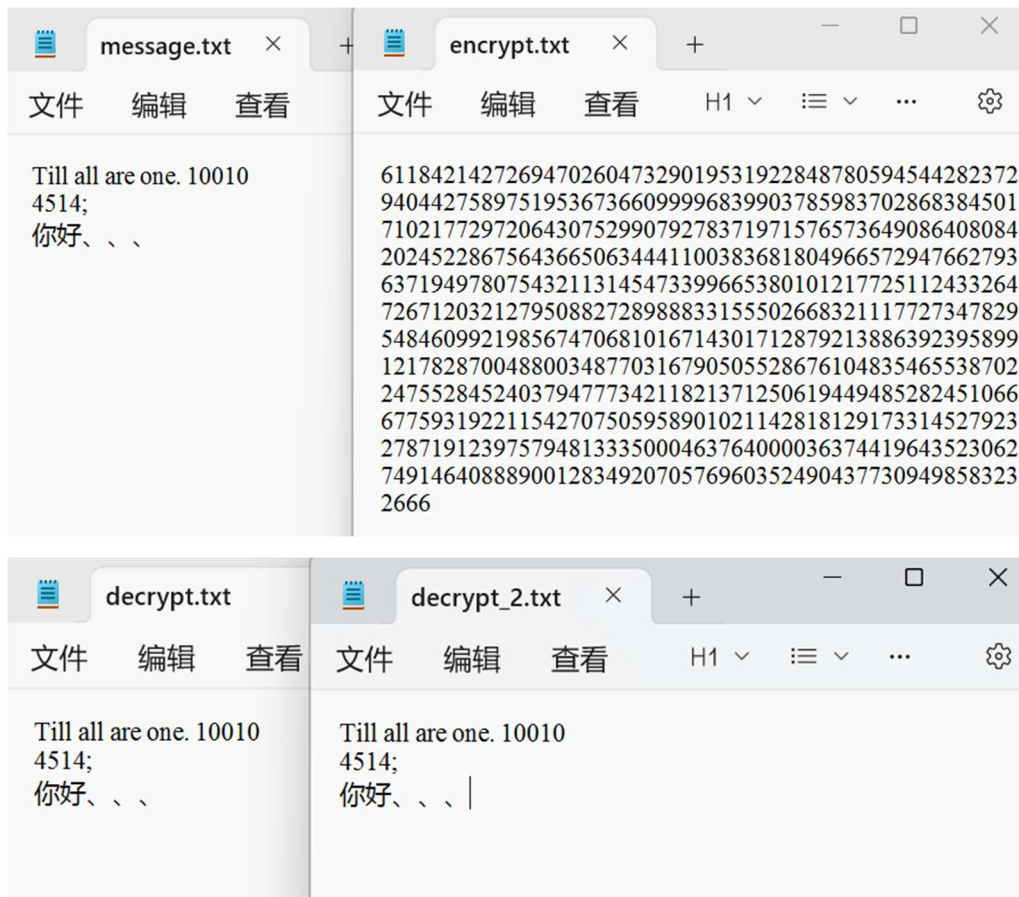
③查看 txt 文件

将文本转化为大整数的方法为将每个字符 (char) 转化为三位 int 整数并逐个拼接，即 $(\text{int})\text{ch1}||(\text{int})\text{ch2}||\dots$ 。测试文件中含有中文字符、中文标点、英文字符、英文标点、数字。

加密结果为大整数直接输出，因此 encrypted 中应当只有数字。

解密结果应当与明文完全一致，decrypt 与 decrypt\_2 都采用文本输出，不会出现中文字符导致的乱码。

|               |                 |      |      |
|---------------|-----------------|------|------|
| decrypt.txt   | 2025/9/6 21:36  | 文本文档 | 1 KB |
| decrypt_2.txt | 2025/9/6 21:36  | 文本文档 | 1 KB |
| encrypt.txt   | 2025/9/6 21:36  | 文本文档 | 1 KB |
| message.txt   | 2025/8/28 18:03 | 文本文档 | 1 KB |



### 3. 源代码

```
class RSA {
private:
    const int KEYLEN; //pq长度
    ZZ a, p, q;
    ZZ Euler_n; //  $\phi(n)$ 
    void GenerateKey();
public:
    ZZ n, b;
    RSA(int keylen = KEYLEN1) : KEYLEN(keylen) {this->GenerateKey();}
    void Encrypt(const ZZ& x, ZZ& y);
    void Decrypt(const ZZ& y, ZZ& x);
};

void RSA::GenerateKey() {
    const int keylen = KEYLEN;
    RandomPrime(p, keylen, 10); //测试20次
    RandomPrime(q, keylen, 10);
    n = p * q;
    Euler_n = (p - 1) * (q - 1);
}
```

---

```
do {
    b = RandomBnd(Euler_n - 3) + 2;
    //生成0-n之间的为随机数。0<=x<=n ; 2<=b<=φ(n)-1
} while (GCD(b, Euler_n) != 1);
InvMod(a, b, Euler_n); //a=b-1 mod φ(n)
}

void RSA::Encrypt(const ZZ& x, ZZ& y) {
    //y=xb mod n
    y = PowerMod(x % n, b, n);
}

void RSA::Decrypt(const ZZ& y, ZZ& x)
{
    //x=ya mod n;
    x = PowerMod(y % n, a, n);
}
```

## 二、RSA 签名算法

### 1. 程序设计说明

#### 1.1 原理

RSA 签名方案:

公钥:  $PK=(n,b)$ , 其中  $n=pq$ ,  $p$  和  $q$  为素数

私钥:  $SK=a$ , 满足:  $ab \equiv 1 \pmod{(p-1)(q-1)}$

签名算法: 对消息  $x \in \mathbb{Z}_n$ , 签名如下计算:

$$\text{sig}_{SK}(x) = x^a \pmod n$$

验证算法: 对签名消息  $(x,y)$  如下验证:

$$\text{Ver}_{PK}(x,y) = (x == y^b \pmod n)$$

#### 7.2.1 签名和 Hash 函数

签名方案:

① 计算消息  $x$  的 Hash 值  $z=h(x)$

② 计算  $z$  的签名  $\text{sig}_{SK}(z)$

③ 发送  $(x,y)$

验证方案: 计算  $z=h(x)$ ,  $\text{Ver}_{PK}(z,y)$

注: 本题及后续两题均采用 SHA-1 算法进行 Hash 值计算 (4.3.2)

### 1.2 文件结构

#### 2.RSA 签名

|               |                           |
|---------------|---------------------------|
| —RSA.h        | //RSA 类的定义及函数声明           |
| —RSA_Sign.cpp | //RSA 类函数的实现              |
| —SHA.h        | //SHA-1 算法                |
| —SHA.cpp      | //SHA-1 算法的具体实现           |
| —main.cpp     | //具体签名与验证流程的实现            |
| —message.txt  | //需要签名的消息, 在 main 函数中文件输入 |

### 1.3 算法实现

#### ① SHA-1 哈希算法

参考:

[CryptProject/RSA\\_Sign/SHA\\_1.h at master · Fcloodooud/CryptProject](#)  
[CryptProject/RSA\\_Sign/SHA\\_1.cpp at master · Fcloodooud/CryptProject](#)

#### ② 签名方案

由于要调用 SHA-1 算法, 所以输入消息  $x$  采用 `string` 型变量。

求消息  $x$  的 Hash 值, 然后用第一题的加密算法加密  $h(x)$

返回大整数  $\text{sig}$ , 为消息  $x$  的签名

#### ③ 验证方案

计算消息  $x$  的 Hash 值, 同时用解密算法还原签名。

返回两者是否相等。

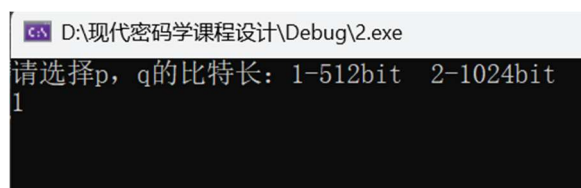
#### ④ 主函数运行流程

---

在控制台窗口选取密钥长度：512/1024  
声明 RSA 变量，自动生成 RSA 密钥  
从文件中读取明文  
对消息进行 RSA 数字签名，签名结果输出文件  
对签名和消息进行验证，验证结果在控制台输出  
对签名进行更改，再次进行验证，错误验证结果在控制台输出  
从签名文件中读取签名，运行验证算法，验证结果在控制台输出  
可由控制台的输出检验算法正确性

## 2. 程序使用说明

①选择密钥长度，根据提示语句进行选择：



```
C:\D:\现代密码学课程设计\Debug\2.exe
请选择p, q的比特长: 1-512bit 2-1024bit
1
```

选择 512bit 约在 10s 内出现下一行提示文字，1024bit 则更长。

②自动读取文件目录下的 message.txt 作为输入，运行三次验证，自动输出结果



```
Microsoft Visual Studio 调试控制台
请选择p, q的比特长: 1-512bit 2-1024bit
1
rsa签名已写入sign.txt
rsa签名验证结果为: true
错误签名验证结果为: false
验证: 对sign.txt与message.txt的验证结果为: true
```

③检验算法正确性

由于函数参数的不同，本题中 message.txt 直接输入一个 string 型变量，无需进行文本与整数类型的转化。测试文件中含有中文字符、中文标点、英文字符、英文标点、数字。

控制台会显示三次验证结果，如果运行正确，验证结果为：true false true。

## 3. 源代码

---

```
-----RSA-----
class RSA {
private:
    const int KEYLEN;//pq长度
    ZZ a, p, q;
    ZZ Euler_n;// $\phi(n)$ 
    void GenerateKey();
public:
    ZZ n, b;
    RSA(int keylen = KEYLEN1) : KEYLEN(keylen) {this->GenerateKey(); }
    void Encrypt(const ZZ& x, ZZ& y);
```



---

```

    void Decrypt(const ZZ& y, ZZ& x);
    void Sign(const string& x, ZZ& sig);
    bool Verify(const string& x, const ZZ& y);
};

```

```

void RSA::Sign(const string& x, ZZ& sig)
{
    /* sha加密 输出长度为160bit */
    SHA_1 sha;
    vector<DWORD> sha_x;
    ZZ hx = ZZ(0);
    sha_x = sha.SHA_Encrypt(x);
    hx = sha_x[0];
    for (int i = 1; i < sha_x.size(); i++)
        hx = (hx << (sizeofDWORD)) + sha_x[i];
    sig = PowerMod(hx % n, a, n);
}

```

```

bool RSA::Verify(const string& x, const ZZ& y)
{
    //verPK(x,y)=(x == y^b mod n)?
    SHA_1 sha;
    vector<DWORD> sha_x;
    ZZ hx = ZZ(0);
    sha_x = sha.SHA_Encrypt(x);
    hx = sha_x[0];
    for (int i = 1; i < sha_x.size(); i++)
        hx = (hx << (sizeofDWORD)) + sha_x[i];
    ZZ ver = PowerMod(y % n, b, n);
    return (ver == hx % n);
}

```

---

#### SHA

```

class SHA_1 {
    DWORD A, B, C, D, E;
    DWORD H0, H1, H2, H3, H4; //buffer
    DWORD W[80];
    vector<vector<DWORD>>> SHA_Pad(string x);
    DWORD Kt(int t);
    DWORD Ft(int t, DWORD B, DWORD C, DWORD D);
    DWORD ROTL(DWORD x, int s);
    void setW(vector<DWORD> m);
public:
    vector< DWORD> SHA_Encrypt(string x);
};

```

---

```

vector<DWORD> SHA_1::SHA_Encrypt(string x) {
    vector<vector<DWORD>> y = SHA_Pad(x);
    H0 = 0x67452301; H1 = 0xEFCDAB89; H2 = 0x98BADCFE; H3 = 0x10325476; H4 =
0xC3D2E1F0;
    int n = y.size();
    for (int i = 0; i < n; i++) {
        setW(y[i]);
        for (int t = 16; t <= 79; t++)
            W[t] = ROTL(W[t - 3] ^ W[t - 8] ^ W[t - 14] ^ W[t - 16], 1);
        A = H0; B = H1; C = H2; D = H3; E = H4;
        for (int t = 0; t <= 79; t++) {
            DWORD temp = ROTL(A, 5) + Ft(t, B, C, D) + E + W[t] + Kt(t);
            E = D; D = C; C = ROTL(B, 30); B = A; A = temp;
        }
        H0 = H0 + A; H1 = H1 + B; H2 = H2 + C; H3 = H3 + D; H4 = H4 + E;
    }
    vector<DWORD> res = { H0, H1, H2, H3, H4 };
    return res;
}

```

### 三、实现一个简单的证书方案

#### 1. 程序设计说明

##### 1.1 原理

协议 9.5 向 Alice 颁布证书

1. TA 通过出生证或护照等身份证明来确定 Alice 的身份。TA 形成一个串，记为  $ID(Alice)$ ， $ID(Alice)$  中包含 Alice 的身份识别信息。

2. Alice 的秘密签名密钥  $sig_{Alice}$  和相应的公开验证密钥  $ver_{Alice}$  被确定。

3. TA 产生对 Alice 身份标识和验证密钥的签名

$$s = sig_{TA}(ID(Alice) || ver_{Alice})^*$$

把证书

$$Cert(Alice) = (ID(Alice) || ver_{Alice} || s)$$

连同 Alice 的私钥  $sig_{Alice}$  一起传给 Alice。

本题中做如下约束：

① Alice 的证书  $Cert(Alice)$  格式如下：

$$Cert(Alice) = (ID(Alice) || ver_{Alice} || s || ID(TA) || flag)$$

其中签名  $s$  如下计算：

$$s = sig_{TA \text{ 的私钥}}(ID(Alice) || ver_{Alice})$$

$ver_{Alice}$  表示 Alice 的公钥， $flag$  标识 Alice 的公钥长度(素数  $p$  和  $q$  为 512 或 1024 位)

② 对应的验证算法不变，即

$$ver_{TA \text{ 的公钥}}(ID(Alice) || ver_{Alice}, s) = true$$

则说明证书可信，否则不可信。

##### 1.2 文件结构

###### 3. 简单证书方案

|                  |                               |
|------------------|-------------------------------|
| —RSA.h           | //RSA 类的定义及函数声明               |
| —RSA_Sign.cpp    | //RSA 类函数的实现                  |
| —SHA.h           | //SHA-1 算法                    |
| —SHA.cpp         | //SHA-1 算法的具体实现               |
| —Certificate.cpp | //具体证书发布与验证流程的实现              |
| —Alice_id.txt    | //Alice 的 id，在 main 函数中文件输入   |
| —TA_id.txt       | //验证机构 TA 的 id，在 main 函数中文件输入 |

##### 1.3 算法实现

###### ① 证书颁发

主要实现不同变量的拼接以及方便后续在整数中读取各类变量，因此对各个变量的长度进行规定，通过左移运算符进行拼接。

UserID: 规定最长为 64bit

Ver:  $n || b$  两者长度均为密钥长度\*2

Sig: 长度与  $n$  相同，密钥长度\*2

TAID: 规定最长为 64bit

Flag: 1 位, 0 表示 512bit, 1 表示 1024bit

注: 在最终的证书中 UserID 位于句首, 会截掉开头的数个 0, 因此最终证书的长度应当小于等于  $(64+64+keylen*4+keylen*2+1)$ 。另外, id 长度不可以超过 64bit, 否则会造成不同变量之间的重叠。

## ②证书验证

本题中的验证算法为从函数参数输入 UserID、公钥、证书, 然后从证书中截取出签名 sig, 对 sig、ID||ver 进行签名验证。

也可以从证书中直接截取 UserID 和公钥, 然后与签名进行验证, 函数参数只需要证书即可。该验证算法在下一题中实现, 即输入证书自动进行验证并返回公钥。

截取出签名的过程为: 右移  $(64+1)$  bit 位, 模  $2^{keylen*2}$ 。

## ③主函数运行流程

在控制台窗口选取密钥长度: 512/1024

声明 RSA 变量 (用户 Alice 与 TA), 自动生成 RSA 密钥

从文件中读取 Alice 与 TA 的 id

将 Alice 的 id 与公钥传给 TA 中的函数, TA 颁发证书, 证书输出文件

将 Alice 的私钥输出文件

对证书进行验证, 验证结果在控制台输出

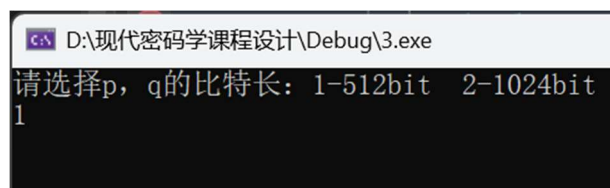
从证书文件中读取签名, 运行验证算法, 验证结果在控制台输出

对证书进行更改, 再次进行验证, 错误验证结果在控制台输出

查看目录下的 txt 文件与控制台输出结果

## 2. 程序使用说明

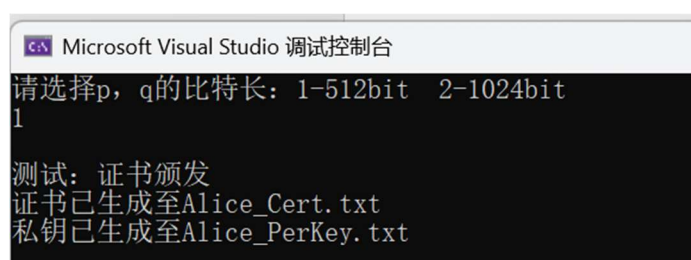
### ①选择密钥长度, 根据提示语句进行选择:



```
D:\现代密码学课程设计\Debug\3.exe
请选择p, q的比特长: 1-512bit 2-1024bit
1
```

选择 512bit 约在 10s 内出现下一行提示文字, 1024bit 则更长。

### ②自动读取文件目录下的 Alice\_id.txt 与 TA\_id.txt, 并令 TA 给 Alice 颁发证书, 证书与私钥会分别放入不同文件。



```
Microsoft Visual Studio 调试控制台
请选择p, q的比特长: 1-512bit 2-1024bit
1
测试: 证书颁发
证书已生成至Alice_Cert.txt
私钥已生成至Alice_PerKey.txt
```

### ③控制台显示三次证书验证结果, 如果运行正确, 验证结果为: true true false。





```
Microsoft Visual Studio 调试控制台
请选择p, q的比特长: 1-512bit 2-1024bit
1

测试: 证书颁发
证书已生成至Alice_Cert.txt
私钥已生成至Alice_PerKey.txt

测试: 证书验证
证书验证结果为: true
从Alice_Cert.txt读取的证书验证结果为: true
错误检测-证书验证结果为: false
```

#### ④查看 txt 文件

文件目录下可以查看 Alice 的证书和私钥, 模拟 TA 将证书和私钥一起传给 Alice。

|  |                |      |      |
|--|----------------|------|------|
|  Alice_Cert.txt   | 2025/9/6 23:07 | 文本文档 | 1 KB |
|  Alice_id.txt     | 2025/9/5 17:35 | 文本文档 | 1 KB |
|  Alice_PerKey.txt | 2025/9/6 23:07 | 文本文档 | 1 KB |
|  TA_id.txt        | 2025/9/5 17:35 | 文本文档 | 1 KB |

### 3. 源代码

```
class RSA {
private:
    const int KEYLEN;//pq长度
    ZZ a, p, q;
    ZZ Euler_n;// $\phi(n)$ 
    void GenerateKey();
public:
    ZZ n, b, ID;
    RSA(int keylen = KEYLEN1) : KEYLEN(keylen) {this->GenerateKey(); }
    void Encrypt(const ZZ& x, ZZ& y);
    void Decrypt(const ZZ& y, ZZ& x);
    void Sign(const string& x, ZZ& sig);
    bool Verify(const string& x, const ZZ& y);
    void printPerKey(ZZ& _p, ZZ& _q, ZZ& _a) {_p = p; _q = q; _a = a;}//打印私钥
    ZZ Certificate(const ZZ& User_ID, const ZZ& n, const ZZ& b);
    bool Cert_Verify(const ZZ& User_ID, const ZZ& n, const ZZ& b, const ZZ& Cert);
};

ZZ RSA::Certificate(const ZZ& User_ID, const ZZ& n, const ZZ& b) {
    ZZ Cert=User_ID;
    ZZ sig;
    Cert = (Cert << (KEYLEN * 2)) + n;//ver=n||b
    Cert = (Cert << (KEYLEN * 2)) + b;//Cert=ID||ver
    ostringstream out;
```

---

```

    out << Cert;
    string ID_ver = out.str();
    Sign(ID_ver, sig);
    Cert = (Cert << KEYLEN * 2) + sig;
    Cert = (Cert << TAID_LEN) + ID;
    Cert=(Cert<<1)+(KEYLEN == KEYLEN1) ? 0 : 1); //flag=0 -->512 1-->1024
    return Cert;
}

bool RSA::Cert_Verify(const ZZ& User_ID, const ZZ& n, const ZZ& b, const ZZ& Cert) {
    ZZ temp = power(ZZ(2), (KEYLEN * 2));
    ZZ sig;
    sig= Cert >> 1 + TAID_LEN;
    sig %= temp;
    temp = User_ID;
    temp = (temp << (KEYLEN * 2)) + n; //ver=n||b
    temp = (temp << (KEYLEN * 2)) + b; //Cert=ID||ver
    ostringstream out;
    out << temp;
    string ID_ver = out.str();
    bool ver=Verify(ID_ver, sig);
    return ver;
}

```

---

## 四、实现严格层次 PKI 系统的简易版本

### 1. 程序设计说明

#### 1.1 原理

PKI 系统组成:

- ①包含一个根 CA(CAroot), 根 CA 的证书由自己签名, 自己给自己颁发。
- ②根 CA 下有 2 个下级 CA (CA1,CA2), 它们的证书由根 CA 签名和颁发。
- ③CA1 和 CA2 下是用户。用户证书由 CA1 或 CA2 签名和颁发。
- ④各个 CA 生成证书后,把证书存储到证书库中。

证书库:

- ①只存储 CA 发来的证书。
- ②对任何人提出的查询申请(申请内容为证书所有者的 ID), 返回证书路径。  
如 Bob 发送申请为 ID (Alice), 而 Alice 的证书由 CA1 颁发, 则证书库返回路径为: CAroot 的证书, CA1 的证书, Alice 的证书

实现该证书系统的一个使用例子:

- ①Alice 向 CA1 或 CA2 申请证书, CA 生成 Alice 的证书, 并把 Alice 的证书存储到系统的证书库中供以后查询。
- ②Bob 向 CA1 或 CA2 申请证书, CA 生成 Bob 的证书, 并把 Bob 的证书存储到系统的证书库中供以后查询。
- ③Eve 向 CA1 或 CA2 申请证书, CA 生成 Eve 的证书, 并把 Eve 的证书存储到系统的证书库中供以后查询。
- ④Alice 向 Bob 发送消息和该消息的签名。
- ⑤Bob 在证书库中查询 Alice 的证书,证书库向 Bob 返回 Alice 的证书路径(或称为证书链)(包含根 CA 的证书, 给 Alice 颁发证书的 CA 的证书及 Alice 的证书)。
- ⑥Bob 验证 Alice 的证书路径是否正确, 如果正确, 则用 Alice 的证书中的公钥验证 Alice 发来的签名是否正确。

另: 要求不同对象(如 TA、CA、证书库、Alice 和 Bob) 完成的任务都用独立的模块完成。

#### 1.2 文件结构

##### 4.PKI 系统

|               |                 |
|---------------|-----------------|
| —RSA.h        | //RSA 类的定义及函数声明 |
| —RSA_Sign.cpp | //RSA 类函数的实现    |
| —SHA.h        | //SHA-1 算法      |
| —SHA.cpp      | //SHA-1 算法的具体实现 |
| —Library.h    | //证书库模块定义       |
| —Library.cpp  | //证书库相关函数的实现    |

---

```
|—User.h           //用户（AliceBobEve）分别要做的任务
|—PKI.cpp          //用户任务的调用
|—Alice_message.txt //例子④中 Alice 发送的消息
```

### 1.3 算法实现

#### ①证书库

存储空间：二维数组。为了查找方便，所有证书在加入数组时会读取 UserID 和 TAID，放入数组的同一行中。

加入证书库：在证书中截取两个 id，与证书本身一起放入数组。

查找证书路径：根据 UserID 查找证书路径。逐一比对 UserID，从对应用户证书中找到次级 CA 的 ID，再查找根 CA 颁发给次级 CA 的证书，根 CA 的证书为数组的第 0 行。

#### ②用户任务

在 User.h 中有三个类，分别为 \_Alice, \_Bob, \_Eve。为了模拟不同用户间的消息传递以及用户个人信息的不公开性，类中的 RSA 变量均为 private。每个人执行的任务对应一条 public 函数，用以模拟不同用户在 PKI 中的操作。

#### ③CA

类\_CA 位于 PKI.cpp（即主函数所在）中，其中包含三个 RSA 类变量，即三个 CA。

以上①②③为本题中对证书库、Alice/Bob/...、CA（TA）的模块划分，模拟各个终端访问信息的受限性。

#### ④使用例子的实现

申请证书：第三题中的证书颁发

添加到证书库：Library 相关函数

发送消息和签名：第二题中的签名方案

查询证书链：Library 相关函数

验证证书：第三题中证书验证方案的改进版（即从证书中读取公钥和 ID 和签名进行验证，如果验证成功返回公钥）

签名验证：第二题中的签名验证方案

#### ⑤主函数运行流程

生成并初始化三个 CA，证书加入证书库

CA1 颁发 Alice 的证书，证书输出至文件 ①

CA2 颁发 Bob 的证书，证书输出至文件 ②

CA1 颁发 Eve 的证书，证书输出至文件 ③

从文件中读取 Alice 的消息并签名，签名结果输出至文件 ④

证书库处理 Bob 查询请求，返回证书链 ⑤

Bob 在 CA 中验证证书链，并用公钥验证签名 ⑥

⑥的验证结果会在控制台输出

检查文件目录

## 2. 程序使用说明

①CA 初始化，即 CA 生成密钥以及相互颁发证书并加入证书库的过程。

大概需要 10s 左右，出现以下字样表示结束。



```
Microsoft Visual Studio 调试控制台  
CA初始化完毕  
实现该证书系统的一个使用例子：
```

②生成三个证书，均可以在文件目录下找到。

```
实现该证书系统的一个使用例子：  
  
Alice向CA1或CA2申请证书（此处假定为CA1）  
证书已生成至Alice_Cert.txt  
  
Bob向CA1或CA2申请证书（此处假定为CA2）  
证书已生成至Bob_Cert.txt  
  
Eve向CA1或CA2申请证书（此处假定为CA1）  
证书已生成至Eve_Cert.txt
```

③读取 Alice 的消息（在 Alice\_message.txt 中），生成签名，可在文件目录下找到。

```
Alice向Bob发送消息和该消息的签名  
已成功读取Alice_message.txt  
Alice的签名已生成至Alice_sig.txt
```

④证书链，以及验证（“证书路径正确”）。验证正确后从文件中读取消息和签名进行验证，验证结果会在控制台输出，结果应当为 true。

```
Bob在证书库中查询Alice的证书  
证书链-根CA 已生成至Alice_Cert_CAroot.txt  
证书链-CA1 已生成至Alice_Cert_CA1.txt  
证书链-用户 已生成至Alice_Cert_User.txt  
  
Bob验证Alice的证书路径是否正确  
证书路径正确  
已成功读取Alice_message.txt  
已成功读取Alice_sig.txt  
签名验证: true
```

⑤完整控制台输出：

```
Microsoft Visual Studio 调试控制台

CA初始化完毕

实现该证书系统的一个使用例子：

Alice向CA1或CA2申请证书（此处假定为CA1）
证书已生成至Alice_Cert.txt

Bob向CA1或CA2申请证书（此处假定为CA2）
证书已生成至Bob_Cert.txt









Eve向CA1或CA2申请证书（此处假定为CA1）
证书已生成至Eve_Cert.txt

Alice向Bob发送消息和该消息的签名
已成功读取Alice_message.txt
Alice的签名已生成至Alice_sig.txt

Bob在证书库中查询Alice的证书
证书链-根CA 已生成至Alice_Cert_CAroot.txt
证书链-CA1 已生成至Alice_Cert_CA1.txt
证书链-用户 已生成至Alice_Cert_User.txt

Bob验证Alice的证书路径是否正确
证书路径正确
已成功读取Alice_message.txt
已成功读取Alice_sig.txt
签名验证：true
```

生成的 txt 目录：

|   |               |      |      |
|---|---------------|------|------|
|  Alice_Cert.txt        | 2025/9/7 0:01 | 文本文档 | 1 KB |
|  Alice_Cert_CA1.txt    | 2025/9/7 0:01 | 文本文档 | 1 KB |
|  Alice_Cert_CAroot.txt | 2025/9/7 0:01 | 文本文档 | 1 KB |
|  Alice_Cert_User.txt   | 2025/9/7 0:01 | 文本文档 | 1 KB |
|  Alice_message.txt     | 2025/9/6 1:33 | 文本文档 | 1 KB |
|  Alice_sig.txt         | 2025/9/7 0:01 | 文本文档 | 1 KB |
|  Bob_Cert.txt          | 2025/9/7 0:01 | 文本文档 | 1 KB |
|  Eve_Cert.txt          | 2025/9/7 0:01 | 文本文档 | 1 KB |

### 3. 源代码

-----RSA-----

```
class RSA {
private:
    const int KEYLEN;//pq长度
    ZZ a, p, q;
    ZZ Euler_n;// $\phi(n)$ 
    void GenerateKey();
public:
    ZZ n, b, ID;
    RSA(int keylen = KEYLEN1) : KEYLEN(keylen) {this->GenerateKey();}
    void Encrypt(const ZZ& x, ZZ& y);
```

---

```

void Decrypt(const ZZ& y, ZZ& x);
void Sign(const string& x, ZZ& sig);
bool Verify(const string& x, const ZZ& y);
void printPerKey(ZZ& _p, ZZ& _q, ZZ& _a); //打印私钥
ZZ Certificate(const ZZ& User_ID, const ZZ& n, const ZZ& b);
bool Cert_Verify(const ZZ& User_ID, const ZZ& n, const ZZ& b, const ZZ& Cert);
bool Cert_Verify(const ZZ& User_ID, const ZZ& Cert, ZZ& n_ret, ZZ& b_ret);
};

bool Verify(const string& x, const ZZ&n, const ZZ&b, const ZZ& y)
{
    //verPK(x,y)=(x == y^b mod n)?

    SHA_1 sha;
    vector<DWORD> sha_x;
    ZZ hx = ZZ(0);

    sha_x = sha.SHA_Encrypt(x);

    hx = sha_x[0];

    for (int i = 1; i < sha_x.size(); i++) {
        hx = (hx << (sizeofDWORD)) + sha_x[i];
    }

    ZZ ver = PowerMod(y % n, b, n);

    return (ver == hx % n);
}

bool RSA::Cert_Verify(const ZZ& User_ID, const ZZ& Cert, ZZ&n_ret, ZZ&b_ret) {
    //如果验证成功会返回公钥
    ZZ temp = power(ZZ(2), (KEYLEN * 2));
    ZZ sig;
    ZZ _n;
    ZZ _b;
    sig = Cert >> 1 + TAID_LEN;
    sig %= temp;

    _n = _b = Cert >> 1 + TAID_LEN + KEYLEN * 2;
    _n = _n >> KEYLEN * 2;
    _b %= temp;
    _n %= temp;

    temp = power(ZZ(2), (KEYLEN * 2 + 1 + TAID_LEN));

```

---

```

temp = Cert / temp;

ostringstream out;
out << temp;
string ID_ver = out.str();

bool ver = Verify(ID_ver, sig);
n_ret = _n; b_ret = _b;
return ver;
}

-----Library-----

class CERTLIB {
private:
    vector<vector<ZZ>>>lib;
    const int KEYLEN;
public:
    CERTLIB(int keylen = KEYLEN1) : KEYLEN(keylen) {};
    vector<ZZ> findroute(const ZZ&User_ID);
    void addtolib(const ZZ& Cert);
};

vector<ZZ> CERTLIB::findroute(const ZZ& User_ID) {
    vector<ZZ> route;
    route.push_back(lib[0][NO_CERT]); //CAROOT 是第一个加入证书库的证书
    ZZ TA_ID;
    for (int i = 1; i < lib.size(); i++)
        if (lib[i][NO_USERID] == User_ID) {
            TA_ID = lib[i][NO_TAID];
            for (int j = 1; j < lib.size(); j++)
                if (lib[j][NO_USERID] == TA_ID &&
                    lib[j][NO_TAID] == lib[0][NO_USERID]) { //次级CA证书
                    route.push_back(lib[j][NO_CERT]);
                    break;
                }
            route.push_back(lib[i][NO_CERT]);
            break; //默认每个id只会有一个证书，只会更新而不会重复
        }
    return route;
}

void CERTLIB::addtolib(const ZZ& Cert) {
    vector<ZZ> tmp(3);

```

---

```

tmp[NO_CERT]=(Cert);
ZZ User_ID = power(ZZ(2), 1 + TAID_LEN + KEYLEN * 2 * 3);
//flag|TAID|sig|ver ver=n|b
User_ID = Cert / User_ID;
tmp[NO_USERID] = (User_ID);
ZZ TA_ID = power(ZZ(2), TAID_LEN );
TA_ID = (Cert >> 1) % TA_ID;
tmp[NO_TAID] = (TA_ID);
lib.push_back(tmp);
}

```

```

-----User-----

class _Alice {
private: RSA Alice;
public:
    ZZ ApplyCertificate( RSA& CA, CERTLIB& lib) {
        Alice.ID = Alice_ID;
        ZZ Alice_Cert = CA.Certificate(Alice.ID, Alice.n, Alice.b);
        lib.addtolib(Alice_Cert);
        return Alice_Cert;
    }
    ZZ SignMessage(const string&filename) {
        ZZ sig;
        string message;
        ReadFile(filename, message);
        Alice.Sign(message, sig);
        return sig;
    }
};

class _Bob {
private: RSA Bob;
public:
    ZZ ApplyCertificate( RSA& CA, CERTLIB& lib) {
        Bob.ID = Bob_ID;
        ZZ Bob_Cert = CA.Certificate(Bob.ID, Bob.n, Bob.b);
        lib.addtolib(Bob_Cert);
        return Bob_Cert;
    }
    vector<ZZ> CheckCertificate(const ZZ& User_ID, CERTLIB& lib){
        return lib.findroute(User_ID);
    }
    bool VerifyCertRoute(RSA& CAROOT, RSA& CA1, RSA& CA2, vector<ZZ>&
route,ZZ&n,ZZ&b) {

```

---

```

    ZZ n, b;
    bool ver1 = CAROOT.Cert_Verify(CAROOT_ID, route[0], n, b);
    bool ver2 = CAROOT.Cert_Verify(CA1_ID, route[1], n, b);
    bool ver3 = CA1.Cert_Verify(Alice_ID, route[2], n, b);
    _n = n, _b = b;
    if (ver1 && ver2 && ver3) {
        cout << "证书路径正确" << endl;
        return 1;
    }
    else {
        cout << "证书路径验证失败" << endl;
        return 0;
    }
}

bool VerifySig(string &m_filename, string &sig_filename, ZZ&n, ZZ&b) {
    string message;
    ReadFile(m_filename, message);
    ZZ sig;
    ReadFile(sig_filename, sig);
    bool ver = Verify(message, n, b, sig);
    cout << "签名验证: " << ( ver? "true" : "false") << endl;
    return ver;
}

};

class _Eve {
private: RSA Eve;
public:
    ZZ ApplyCertificate(RSA& CA, CERTLIB& lib) {
        Eve.ID = Bob_ID;
        ZZ Eve_Cert = CA.Certificate(Eve.ID, Eve.n, Eve.b);
        lib.addtolib(Eve_Cert);
        return Eve_Cert;
    }
};

-----CA-----

class CA {
public:
    RSA CAROOT, CA1, CA2;
    void Init(){
        CAROOT.ID = CAROOT_ID;    CA1.ID = CA1_ID;    CA2.ID = CA2_ID;
        ZZ tmp;

```

---

```
    tmp = CAROOT.Certificate(CAROOT.ID, CAROOT.n, CAROOT.b);  
    certlib.addtolib(tmp);  
    tmp = CAROOT.Certificate(CA1.ID, CA1.n, CA1.b);  
    certlib.addtolib(tmp);  
    tmp = CAROOT.Certificate(CA2.ID, CA2.n, CA2.b);  
    certlib.addtolib(tmp);  
    cout << "CA初始化完毕" << endl;  
}  
};
```