《密码学》课程设计实验报告

***本次实验源文件和实验报告均已上传至 Signature 文件夹下：**

[sterzhang/Cryptology: Jianshu Zhang's cryptological experiments (github.com)](https://github.com/sterzhang/Cryptology)

实验序号：06　　　　　　　　　　　　　　　　　　实验项目名称：数字签名

学号	2021302181216	姓名	张鉴殊	专业、班	21 信安
实验地点	C202	指导教师	余荣威	时间	2023.12.29

一、实验目的及要求

实验目的：

(1) 掌握数字签名的概念；

(2) 掌握基于 RSA 密码、ElGamal 密码和椭圆曲线密码的数字签名方法；

(3) 了解基于 RSA 密码、ElGamal 密码和椭圆曲线密码的数字签名的安全性；

(4) 熟悉盲签名的原理，了解盲签名的应用。

实验要求：

(1) 掌握 RSA 数字签名的实现方案；

(2) 掌握 ElGamal 数字签名的实现方案；

(3) 掌握 SM2 椭圆曲线数字签名的实现方案；

(4) 了解数字签名实现中的相关优化算法。

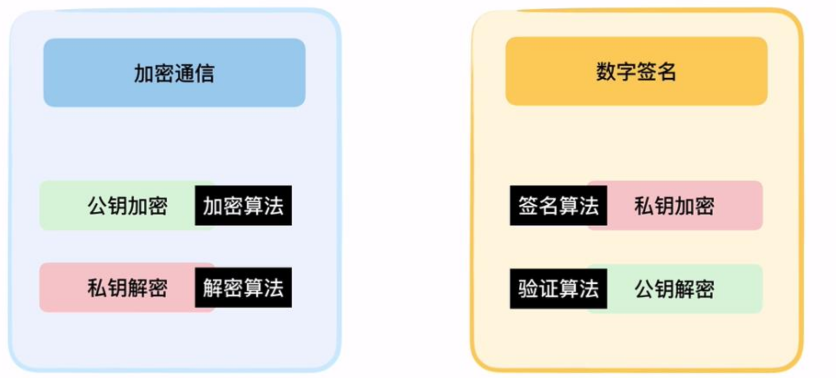
二、实验设备（环境）及要求

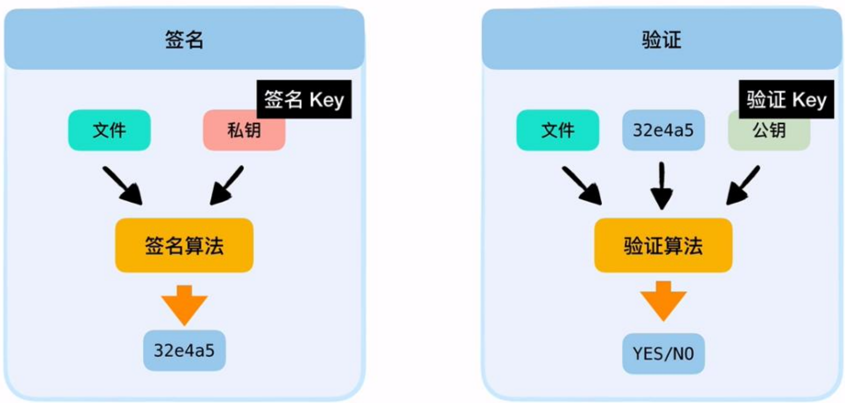
Windows 操作系统，高级语言开发环境

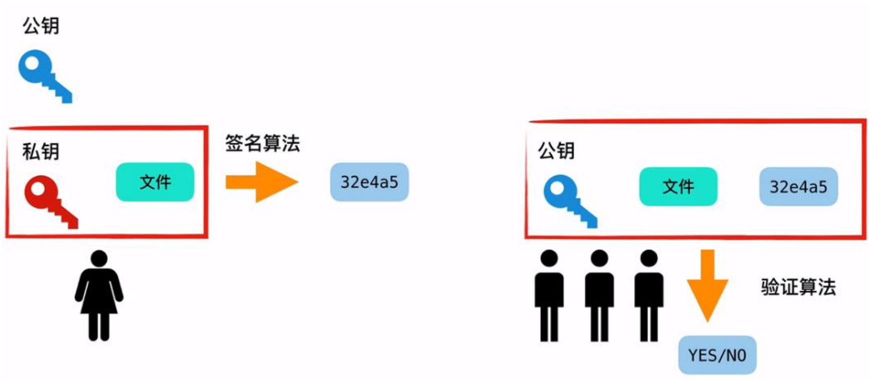
三、实验内容与步骤

加密通信和数字签名都具有对称性，在加密通信中用的是公钥进行加密，用私钥进行解密；而在数字签名中，签名时用的是私钥加密，验证时用的是公钥。

以下是整个签名和验证的流程图：







1. 编程实现 RSA 数字签名方案

参数准备阶段：同实验 07 中的 RSA 密码算法

以下是 RSA 数字签名算法：

签名运算：

$$S = M^d \mod n \tag{8-5}$$

验证签名运算（判断式）：

$$M = S^e \mod n \qquad (8\text{-}6)$$

以下是 RSA 加解密算法，RSA 加解密算法的整体流程如下：

①随机地选择两个大素数 p 和 q，而且保密；

②计算 n=pq，将 n 公开；

③计算 φ(n)=(p-1)(q-1)，对 φ(n)保密；

④随机地选取一个正整数 e，1<e<φ(n)且（e， φ(n)）=1，将 e 公开；

⑤根据 ed＝1 mod φ(n)，求出 d，并对 d 保密；

⑥加密运算：

$$C = Me \mod n \qquad (7\text{-}4)$$

⑦解密运算：

$$M = Cd \mod n \qquad (7\text{-}5)$$

实验 1：采用实验 06 中的 RSA 密码算法的相关参数，对于 M 进行签名及验证。

1）要对 M 进行签名及验证，首先需要用 RSA 解密算法（利用私钥 d,n，以及明文）

```
    # 签名
    signature = decrypt(private_key, plaintext)
```

这里的 decrypt 和 RSA 加密中的没有区别，可以来看一下具体的实现过程：

```
#RSA 解密
def decrypt(key, ciphertext):
    n, d = key
    plaintext = fast_expmod(ciphertext, d, n)
    return plaintext
```

这里用到了快速幂算法：

```
# 快速幂算法
def fast_expmod(a,b,n):
    d = 1
    while b != 0:
        if(b & 1) == 1:
            d = (d * a) % n
        b >>= 1
        a = a * a % n
    return d
```

2）对 M 的签名进行验证，需要用 RSA 加密算法（利用公钥 e，n，以及签名），得到的结果应该是 M：

```
    # 验证签名
    plaintext = encrypt(public_key, signature)
    print("验证签名：\n",hex(plaintext))
```

3）这里我的 main 函数中设置的参数的和课本上的 p218 一致，用来验证正确性：

```
if __name__ == '__main__':
    p = 
0xE86C7F16FD24818FFC502409D33A83C2A2A07FDFE971EB52DE97A3DE092980279EA29E32F
378F5E6B7AB1049BB9E8C5EAE84DBF2847EB94FF14C1E84CF568415
    q = 
0xD7D9D94071FCC67EDE82084BBEDEAE1AAF765917B6877F3193BBAEB5F9F36007127C9AA98
D436A80B3CCE3FCD56D57C4103FB18F1819D5C238A49B0985FE7B49
    e = 5
    # 获取数据
    plaintext = 
0xB503BE7137293906649E0AE436E29819EA2D06ABF31E10091A7383349DE84C5B
```

4）可以看到验证结果与明文一致，说明正确：

```
(py38) root@6dceeba50b41:~/project/test/Sign# python rsa_signature.py
明文:
 0xb503be7137293906649e0ae436e29819ea2d06abf31e10091a7383349de84c5b
phi:
 0xc3f8e7f2836de23254fc66235dd74398be4a82ec846c0667b7c15d5c9e2b81f9e433025aa6df1fb73b280ec7048aac42ffbaa1b66ab83107cb9fe11fc0
26d0aec065107c11beeac74c073e8ebb901278c5611f146c78646773afcd67f8b6fde00d12ad838d3de413f05960cda0623114be356c629a5b747a6062d02
251a2c1a0
d:
 0x4e638ffa9af8c0e0eecb5c0e25894e3d18ea9ac501c4cf5cafe6f2250c116730c1ae00f10f8c731617a99f82ce9dde81331773e291167a031e3ff3a64c
dc537919c2069807192ab61e694c3917d33a96b55a0c6e91c9c1c2fb131ef6637c658cd207789b054bf4d4c68a26b8a68dad3b7f4891c10a8afb64268decd
a870de70d
签名:
 0x4c1d55916936a1f6e948daf371fa667ac706c39e89dbf576a07d3e253a1ceb2fb06535d1873412790e08b398d7e6fda9c75ee06bdb3cb089e69147947f
de05d6ef6b5d2102caf59283e7c77a7bd2ac9ddc7c2e6fd491c00b6404739043b95b2ab72651e33dd3b19dc2a7e0862c4028f813098623450520ba85b61d0
af872fa14
验证签名:
 0xb503be7137293906649e0ae436e29819ea2d06abf31e10091a7383349de84c5b
```

思考 1：RSA 数字签名方案的几种攻击方法

在网上搜到相关的资料基本上分为中间人攻击，选择明文攻击，选择密文攻击，重放攻击，时序攻击，公钥替换攻击，分解大素数。分解大素数这一种暂时不讨论，其他的我都以一种好理解的方式去写下自己的理解。

中间人攻击类似在给朋友发短信的时候，有人在不被你们发现的情况下拦截

了这些消息，并可以随意更改它们，然后再发送给你的朋友。这个人就像一个隐形的“中间人”，他可以窃听、篡改你们的通信。这就是中间人攻击的本质，攻击者置身于通信双方之间，秘密地拦截和/或修改他们之间的信息。

选择明文攻击就是通过观察机器的输出（密文），试图理解甚至找到一种方法来还原或解密。选择密文攻击同理。重放攻击是攻击者拦截了一次有效的数据传输（如登录请求），然后重复发送这些数据来尝试获得未授权的访问或其他利益。在时序攻击中，攻击者通过精确地测量加密操作所需的时间来获取敏感信息，例如私钥。即使加密算法本身很安全，但其执行所需的时间可能泄露关键信息。公钥替换攻击类似于某人在你和朋友之间传递消息时，偷偷更换了你们用来加密通信的密码本。在这种攻击中，攻击者尝试替换公钥，使得加密的消息实际上是用攻击者的公钥加密的，从而让攻击者能够解密并读取这些消息。

思考 2：基于 RSA 数字签名的盲签名方案的实现

1）RSA 密钥生成

设 n=pq 是两个大素数 p 和 q 的乘积

计算私钥指数 d，使得 de≡1modφ(n)，其中φ(n)是欧拉函数φ(n)=(p−1)(q−1)

2）消息的盲化

发送者有一个消息 M

选择一个随机盲化因子 r，并且 r 与 n 互素

计算盲化的消息 M′ 如下：M′=M * r^e (mod n)

3）请求签名

发送者将盲化的消息 M′ 发送给签名者

签名者对 M′ 进行签名，得到盲签名 S′：S′=(M′)^d (mod n)

4）移除盲化

发送者接收到盲签名 S′，并计算原始签名 S = S′*r^−1 (modn)

5）验证签名

任何人都可以验证签名 S 是否有效：通过计算 S^e modn 并检查结果是否等于原始消息 M

2．编程实现 ELGamal 数字签名方案

复习数论的一个结论。对于素数q，如果α是q的原根，则有：$\alpha,\alpha^2,...,\alpha^{q-1}$ 取模$(\text{mod } q)$后各不相同。因此如果α是q的原根，进一步有：

1．对于任意整数m，$\alpha^m \equiv 1(\text{mod } q)$ 当且仅当 $m \equiv 0(\text{mod } q-1)$

2．对于任意整数i，j，$\alpha^i \equiv \alpha^j(\text{mod } q)$ 当且仅当 $i \equiv j(\text{mod } q-1)$

同ELGamal加密方案一样，ELGamal数字签名方案的基本元素是素数p和随机数α，其中α是p的原根。用户A通过如下步骤产生**公钥/私钥对**：

1．生成随机整数X_A，使得$1<X_A<p-1$。

2．计算$Y_A=\alpha^{X_A} \text{ mod } p$。

A 的私钥是X_A；A 的公钥是$\{p,\alpha,Y_A\}$。

例：设 p=19，m=14,构造一个 ELGamal 数字签名方案，并用它对 m 签名。

对于 p=19，原根有{2,3,10,13,14,15}，任选其中之一作为模 19 的本原元（生成元），如选择$\alpha=10$

步骤 1：**密钥生成**

同ELGamal加密方案一样，ELGamal数字签名方案的基本元素是素数p和α，其中α是p的原根。用户A通过如下步骤产生公钥/私钥对：

① 生成随机整数X_A，使得$1<X_A<p-1$。

② 计算$Y_A=\alpha^{X_A} \text{ mod } p$。

A 的私钥是X_A；A 的公钥是$\{p,\alpha,Y_A\}$。

用户随机地选择一个整数 x 作为自己的秘密的解密钥，1<x<p-1，计算 y ≡α^x mod p，取 y 为自己的公开的加密钥。例如选择$x=16$，$y=\alpha^d \text{ mod } p=10^5 \text{ mod} 19=4$，即私钥为 16，公钥为 4.

步骤 2：**签名过程**

如果用户 A 要对明文消息 m 加签名，0≤m≤p-1，其具体签名如下：

① 随机选择一个整数 k，1<k<p-1，且(k，p-1)=1；

② 计算 r = α^k mod p

③ 计算 $s=(m-x_A r)k^{-1} \bmod p-1$
④ 取(r，s)作为 m 的签名，并以<m，r，s>的形式发给用户 B

将明文消息 M（0≤M≤p-1)加密成密文的过程如下：

① 随机地选取一个整数 k，k 与 p-1 互素且 1≤k≤p-1。例如随机选择 $k=5$

② 计算 $r=\alpha^k \bmod p=10^5 \bmod 19=3$

$$s=(m-xr)k^{-1} \bmod p-1=(14-16\times 3)\times 5^{-1} \bmod 18=2\times 11 \bmod 18=4$$

③ 取（r，s）=（3,4）作为 m=14 的签名。

步骤 3：**验证过程**

用户 B 验证签名的流程如下：

判断 $\alpha^m = y_A^r\ r^s \bmod p$ 是否成立，若成立，那么判定签名为真，否则判定签名为假。

对签名（r，s）验证的过程如下：

① 计算 $V_1=\alpha^m \bmod p=10^{14} \bmod 19=16$

② 计算 $V_2=y^r r^s \bmod p=4^3\times 3^4 \bmod 19=7\times 5 \bmod 19=16$

由于 $V_1=V_2$，所以签名是合法的。

实验 2：采用实验 07 中的 ELGamal 密码算法的相关参数，对于 M 进行签名及验证。

1）私钥 beta 生成，公钥是 $\{p,\alpha,Y_A\}$：

```
z=16
class Sign :
    def __init__(self,a,b,c,d):
        self.p=a
        self.alpha=b
        self.beta=int(pow(self.alpha,z)%self.p)
```

2）用户 A 要对明文消息 m 加签名，$0 \le m \le p-1$，随机选择一个整数 k，$1<k<p-1$，且$(k, p-1)=1$：

3）计算 $r = \alpha^k \bmod p$ 以及 $s = (m-x_A r)k^{-1} \bmod p-1$：

```
    def createR(self):
        a=int((self.alpha**self.k)%self.p)
        return a
    def createS(self):
        a=(self.invK()*(self.m-z*self.r))%(self.p-1)
        return a
   def invK(self):
        kc=self.k
        mc=self.p-1
        y=0
        x=1
        if (mc==1):
            return 0
        while (kc>1):
            quotient=kc//mc
            temp=mc
            mc=kc%mc
            kc=temp
            temp=y
            y=x-quotient*y
            x=temp
        if (x<0):
            x=x+self.p-1
        return x
```

4）取(r，s)作为 m 的签名，并以<m，r，s>的形式发给用户 B

```
        print("m的签名<r，s> :
("+str(self.m)+","+str(self.r)+","+str(self.s)+")")
        print("发送<m，r，s> :
("+str(self.m)+","+str(self.r)+","+str(self.s)+")")
```

5）用户 B 验证签名，判断 $\alpha^m = y_A{}^r \ r^s \mod p$ 是否成立，若成立，那么判定签名为真，否则判定签名为假。计算 $V_1 = \alpha^m \bmod p = 10^{14} \bmod 19 = 16$，计算 $V_2 = y^r r^s \bmod p = 4^3 \times 3^4 \bmod 19 = 7 \times 5 \bmod 19 = 16$，由于 $V_1 = V_2$，所以签名是合法的；

```python
class Verify:
    def __init__(self,a,b,c,d,e,f):
        self.p=a
        self.alpha=b
        self.beta=c
        self.m=d
        self.r=e
        self.s=f
    def v1(self,b,c,d):
        a=int(((b**c)*(c**d))%self.p)
        return a
    def v2(self,b,c):
        a=int((b**c)%self.p)
        return a
    def verified(self):
        if(self.v1(self.beta,self.r,self.s)==self.v2(self.alpha,self.m)):
            print("ElGamal 验证签名成功")
            print("V1: "+str(self.v1(self.beta,self.r,self.s)))
            print("V2: "+str(self.v2(self.alpha,self.m)))
        else:
            print("ElGamal 验证签名失败")
            print("V1: "+str(self.v1(self.beta,self.r,self.s)))
            print("V2: "+str(self.v2(self.alpha,self.m)))
```

6）运行可以看到结果与例子相同，验证成功：

```
(py38) root@6dceeba50b41:~/project/test/Sign# python elgamal_signature.py
p : 19
alpha : 10
m : 14
k : 5
<m, r, s> : (14,3,4)
开始验证签名...
ElGamal验证签名成功
V1: 16
V2: 16
```

实验 3：任意选作教材 p254 表 8-1 中的数字签名的变形算法，对于 M 进行签名及验证。

这里我选择变形算法的第一个，对 Elgamal 签名和验证算法进行相应的修改：

编号	签名算法	验证算法
1	$mx = rk + s \mod p-1$	$y^{m} = r^{r}\alpha^{s} \mod p$

```python
    def createR(self):
        a=int((self.alpha**self.k)%self.p)
        return a
    def createS(self):
        # a=(self.invK()*(self.m-z*self.r))%(self.p-1)
        a=((self.m*self.beta)-(self.r*self.k))%(self.p-1)
        return a
```

```python
    def v1(self,b,c,d):
        a=int(((b**b)*(c**d))%self.p)
        return a
    def v2(self,b,c):
        a=int((b**c)%self.p)
        return a

    def verified(self):
        if(self.v1(self.r,self.alpha,self.s)==self.v2(self.beta,self.m)):
            print("ElGamal 验证签名成功")
```

四、实验结果与数据处理

所有结果已经展示在上面步骤的相关位置

五、分析与讨论

1. 在完成数字签名的变形算法时书上写了在实际应用中，对数据的哈希值进行签名，而不是直接对数据本身签名，这是为什么呢？

通过查阅资料，我得知哈希函数可以将任意长度的数据转换为固定长度的哈希值。这个过程是单向的，从哈希值几乎不可能恢复原始数据。因此，哈希值可以代表原始数据，但不会泄露数据的具体内容。另外，哈希值对输入数据非常敏感，即使是微小的变化也会导致完全不同的哈希值。因此，验证签名的哈希值可

以确保数据未被篡改，提供了一种数据完整性的保障。再者，数字签名算法通常涉及计算密集型的数学操作，直接对大量数据进行签名会非常耗时。相比之下，哈希函数通常更快且计算资源需求更低。因此，先生成数据的哈希值，然后对这个较短的哈希值进行签名，可以显著提高签名的速度。

2. 除了哈希函数优化，数字签名实现中的相关优化算法还有哪些？

椭圆曲线密码学，相比于传统的基于大数因子分解的算法（如 RSA），ECC 可以在使用更短的密钥长度的同时提供同等的安全性。这意味着更快的计算速度、更小的存储需求和更低的能源消耗。还可以利用多线程和并行处理来加速签名和验证过程。或者缓存常用的计算结果，如在 RSA 中常见的模数指数来减少重复计算，提高性能。

六、教师评语 签名： 日期：	成绩