武汉大学国家网络安全学院

密码学实验报告

学　　号　2021302181156

姓　　名　赵伯侯

实验名称　数字签名

指导教师　何琨

一、 实验名称: 数字签名

二、 实验目的及要求:

2.1 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

2.2 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

三、 实验设备环境及要求:

Windows 操作系统，python 高级语言开发环境

四、 实验内容与步骤:

4.1 编程实现 RSA 数字签名方案

4.1.1 原理

设 M 为明文，$K_{eA} =< e, n >$ 是 A 的公钥，$K_{dA} =< d, p, q, \varphi(n) >$ 是 A 的私钥，密钥的计算过程在之前的实验中已经实现，在此不过多赘述，可以得到用户 A 对明文 M 进行签名的过程是

$$S_A = D(M, K_{dA}) = (M^d) mod\, n$$

计算得到的 S_A 便是 A 对 M 的签名验证签名的过程是

$$E(S_A, K_{eA}) = (M^d)^e mod\ n = M$$

如果等式成立则签名能够成功验证。

4.1.2 实现

因为该实验是在上次实验 RSA 的基础上实现，所以直接调用上次实验的 RSA 代码。实现代码如下所示

```
from RSA import *

if __name__ == "__main__":

    msg = read_file('message.txt')
    messages = divide(msg)
    print("明文为:" + str(msg))

    e, n, d = init()
    # e=79
    # n=3337
    # d=1019

    # 加密
    secrets = []
    for message in messages:
        secret = RSA_encode(int(message), e, n)
        secrets.append(secret)
    C = ''.join(map(str, secrets))
    print("RSA签名得到的结果为:" + C)

```

```
    # 解密
    Message = []
    for secret in secrets:
        M = RSA_decode(int(secret), d, n)
        Message.append(M)
    M = ''.join(map(str, Message))
    print("RSA验证签名的结果为:" + M)
```

代码 1: RSA 签名

4.1.3 思考

（1）RSA 数字签名方案的几种攻击方法

数学攻击:

分解法：RSA 的安全性基于大数分解的困难。如果攻击者能够分解公钥中的大数（通常是两个大质数的乘积），他们就能破解私钥。这种攻击难度随着所使用的数位长度增加而增加。

选择密文攻击：攻击者可能会尝试用不同的密文来解密，从而推断出私钥。

实现缺陷攻击：

侧信道攻击：通过分析硬件实现 RSA 算法时的物理侧信道信息（如电力消耗、电磁辐射、处理时间等），攻击者可以获得关于私钥的信息。

错误消息攻击：如果加密系统在处理无效密文时返回的错误消息包含关于密钥的信息，攻击者可能会利用这些信息来推断私钥。

网络攻击：

中间人攻击：攻击者拦截通信双方之间的消息，伪装成通信的一方向另一方发送消息。在某些情况下，攻击者可能能够替换或篡改数字签名。

重放攻击：攻击者重新发送或延迟发送合法的数据包，这可能会在没有正确时间戳或序列号的情况下导致安全问题。

协议层面攻击：

Bleichenbacher 攻击: 这是对 PKCS#1 v1.5 填充的 RSA 加密的攻击，它利用了错误格式的 RSA 密文的错误处理方式来逐渐解密密文。

（2）基于 RSA 数字签名的盲签名方案的实现

基于 RSA 数字签名的盲签名方案旨在签名者对信息的内容一无所知。

密钥准备阶段的工作与一般 RSA 相同，在将所有参数初始化之后进行盲签名过程:

- 盲化消息：
  - 用户生成随机数 r（与 n 互质），计算盲因子 $b = r^e \mod n$。
  - 用户有消息 m，计算盲化消息 $m' = m \times b \mod n$。
- 请求签名：
  - 用户将盲化后的消息 m' 发送给签名者。
- 签名盲化消息：
  - 签名者使用私钥 d 对盲化消息进行签名，计算 $s' = (m')^d \mod n$。
  - 签名者将签名 s' 返回给用户。
- 去盲化：
  - 用户接收到盲签名 s'，计算真实签名 $s = s' \times r^{-1} \mod n$。

最后的验证过程可以使用公钥 (e, n) 来计算 $S^e mod\ n$ 如果结果等于原始消息 m 则签名有效实现 RSA 数字签名的盲签名伪代码如下所示

算法 1 RSA 盲签名算法

procedure GENERATEKEYS
    生成 RSA 密钥对 (e, d, n)
    return (e, d, n)
end procedure
procedure BLIND(m, e, n)
    选择随机数 r，且 $gcd(r, n) = 1$
    计算盲因子 $b = r^e \mod n$
    计算盲化消息 $m' = (m \times b) \mod n$
    return m', r
end procedure
procedure SIGN(m', d, n)
    计算盲签名 $s' = (m')^d \mod n$
    return s'
end procedure
procedure UNBLIND(s', r, n)
    计算 r 的逆 $r^{-1} \mod n$
    计算签名 $s = (s' \times r^{-1}) \mod n$
    return s
end procedure
procedure VERIFY(m, s, e, n)
    计算 $v = s^e \mod n$
    if $v = m$ **then**
        return True
    else
        return False
    end if
end procedure

4.2 编程实现 ELGamal 数字签名方案

4.2.1 复习结论

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

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

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

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 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 $Y_A = \alpha^{X_A} mod\, p$

A 的私钥是 X_A；A 的公钥是 $\{p, \alpha, Y_A\}$

4.2.2 原理

（1）系统参数：

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 $y \equiv \alpha^x mod\, p$，取 y 为自己的公开的加密钥。例如选择 x=16,$y = \alpha^x mod\, p = 10^{16} mod\, 19 = 4$ 即私钥为 16，公钥为 4

（2）产生签名：

将明文消息 M（0≤M≤p-1）加密成密文的过程如下：

- 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
- 计算 $r = \alpha^k mod\, p = 10^5 mod\, 19 = 3$

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

- 取 (r,s)=(3,4) 作为 m=14 的签名

（3）验证签名：

对签名（r，s）验证的过程如下：

- 计算 $V_1 = \alpha^m mod\ p = 10^{14} mod\ 19 = 16$
- 计算 $V_2 = y^r r^s mod\ p = 4^3 * 3^4 mod\ 19 = 7 * 5 mod\ 19 = 16$

由于 $V_1 = V_2$，所以签名是合法的

4.2.3 实现

实现的 ELGamal 数字签名程序如下所示

```
from ELGamal import *



def elgamal_sign(message, p, g, d):
    """生成ElGamal数字签名"""
    M = message
    while True:
        k = randint(2, p - 2)
        if e_gcd(k, p - 1)[0] == 1:  # 确保k和p-1互质
            break
    r = fastExpMod(g, k, p)

    k_inv = e_gcd(k, p - 1)[1] % (p - 1)
    s = (k_inv * (M - d * r)) % (p - 1)
    return r, s



def elgamal_verify(message, r, s, p, g, y):
    """验证ElGamal数字签名"""
    M = message
    if r < 1 or r > p - 1:
        return False

```

```
    left = fastExpMod(g, M, p)
    right = (fastExpMod(y, r, p) * fastExpMod(r, s, p)) % p
    return left == right


# 示例使用
if __name__ == '__main__':
    message = read_file("message.txt")

    p, y, d, g = generate_key()  # 生成ElGamal密钥
    # 生成签名

    r, s = elgamal_sign(int(message), p, g, d)
    print("ELGamal签名后的结果为：")
    print("r=" + str(r))
    print("s=" + str(s))

    # 验证签名
    m = elgamal_verify(int(message), r, s, p, g, y)
    print("ELGamal签名验证的结果为：", m)
```

代码 2: ELGamal 签名

4.2.4 例子

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 α=10 最终得到的运行结果如下图所示

```
E:\Python_code\venvs\Scripts\python.exe E:\Python_code\codes\cryptography\lab_6\ELGamal_name\ELGamal_name.py
ELGamal签名后的结果为：
r=3
s=4
ELGamal签名验证的结果为： True

Process finished with exit code 0
```

图 1: ELGamal 例子处理结果

4.3 编程实现 SM2 椭圆曲线数字签名方案

4.3.1 原理

（1）系统参数产生

- 椭圆曲线方程：特定的椭圆曲线方程。
- 有限域：椭圆曲线定义在素数域或二元域上。
- 基点（G）：椭圆曲线上的一个点，其阶为大素数。
- 阶（n）：基点 G 的阶，即最小正整数 n，使得 nG=O（O 为无穷远点）。

（2）产生签名

1 密钥生成：

- 随机选择私钥 d（$1 < d < n$）。
- 计算公钥 $P = dG$。

2 生成签名：

- 随机选择 k（$1 < k < n$）。
- 计算椭圆曲线点 $(x_1, y_1) = kG$。
- 计算 $r = (e + x_1) \mod n$，其中 e 是消息 M 的哈希值。
- 如果 $r = 0$ 或 $r + k = n$，重新选择 k。
- 计算 $s = ((1/d) \cdot (k - r \cdot d)) \mod n$。如果 $s = 0$，重新选择 k。
- 签名为 (r, s)。

（3）签名验证：

1 验证 r 和 s 是否在区间 $[1, n-1]$ 内。

2 计算消息 M 的哈希值 e。

3 计算 $t = (r + s) \mod n$。如果 $t = 0$，则签名无效。

4 计算椭圆曲线点 $(x_1, y_1) = sG + tP$。

5 验证 r 是否等于 $(e + x_1) \mod n$。如果等于，则签名有效；否则，无效。

4.3.2 实现

实现的 SM2 签名算法的代码如下所示

```
from SM2 import *
from MATH import *



def sm2_sign(message, dA, a, b, p, Gx, Gy, n):
    """SM2数字签名"""
    e = int(Hash(message), 16)
    while True:
        k = randint(1, n - 1)
        x1, y1 = mul_point(Gx, Gy, k, a, p)
        r = (e + x1) % n
        if r == 0 or r + k == n:
            continue
        s = (e_gcd(1 + dA, n)[1] * (k - r * dA)) % n
        if s == 0:
            continue
        break
    return r, s



def sm2_verify(message, r, s, PAx, PAy, a, b, p, Gx, Gy, n):
    """SM2签名验证"""
    if not (1 <= r < n and 1 <= s < n):
```

```
        return False
    e = int(Hash(message), 16)
    t = (r + s) % n
    if t == 0:
        return False
    a1,a2=mul_point(Gx, Gy, s, a, p)
    b1,b2=mul_point(PAx, PAy, t, a, p)
    x1, y1 = add_points(a1,a2, b1,b2, a, p)
    R = (e + x1) % n
    return R == r


# 示例使用
if __name__ == '__main__':
    message = read_file('message.txt')
    m_bit=str_bit(message)
    m=hex(int(m_bit, 2))[2:]

    dA = randint(1, n - 1)  # 私钥
    PAx, PAy = mul_point(gx, gy, dA, a, p)  # 公钥

    # 生成签名
    r, s = sm2_sign(m, dA, a, b, p, gx, gy, n)
    print("SM2数字签名后的结果为:")
    print("r="+str(r))
    print("s="+str(s))

    # 验证签名
    is_valid = sm2_verify(m, r, s, PAx, PAy, a, b, p, gx, gy, n)
```

```
53      print("SM2数字签名验证结果为：", is_valid)
```

代码 3: SM2 签名

4.3.3 思考 1: 椭圆曲线加密、签名的快速实现

（1）提示 1：模参数、曲线参数的选取优化。选择具有高效运算特性的椭圆曲线，如 Koblitz 曲线或特定的 Weierstrass 曲线。使用特殊形式的模数，例如梅森素数（Mersenne primes），可以使模运算更快。

（2）提示 2：点加和倍点运算的快速实现。例如蒙哥马利梯形算法或双加倍算法。使用预计算和窗口方法，预先计算曲线点的多倍数，并在运算时重用这些值。

（3）端到端优化。在硬件层面，使用专门设计的处理器或协处理器来加速椭圆曲线运算。在软件层面，使用汇编语言或低级语言优化关键的数学运算，如模乘和模逆。

（4）并行计算。利用现代处理器的多核特性，将一些独立的运算过程并行化。对于一些算法步骤，如哈希运算，在多个处理单元上同时执行。

（5）使用特殊的椭圆曲线。例如，Barreto-Naehrig (BN) 曲线可以用于配对加密，这些曲线对于特定的密码学应用来说是特别高效的。

（6）减少带宽和存储需求。采用压缩形式的椭圆曲线点表示，只存储和传输 x 坐标和 y 坐标的一个位标志，从而减少数据的大小。

4.3.4 思考 2:k=15 时，kP 运算次数

反复平方乘 $31P = [11111]2P = 2(2(2(2P + P) + P) + P) + P$，共 4 次加法、4 次倍加

改进编码 $31P = (25 - 1)P = 2(2(2(2(2P)))) - P$，需要 5 次倍加，1 次加法（减法）

因此可以得出在反复平方乘算法中 $15P = [1111]2P = 2(2(2P + P) + P) + P$ 共 3 次加法、3 次倍加

改进编码 $15P = (16-1)P = 2(2(2(2P))) - P$，需要 4 次倍加，1 次加法（减法）

五、 实验结果与处理

5.1 RSA 数字签名

5.1.1 实验（1）

采用上次实验中 RSA 密码算法的相关参数，对于 M 进行签名及验证

签名程序运行结果如下图所示

```
E:\Python_code\venvs\Scripts\python.exe E:\Python_code\codes\cryptography\lab_6\RSA_name\RSA_name.py
明文为:6882326879666683
公钥<n,e>为:18742697 17569289
私钥<p,q,d,fn>为：2797 6701 10796009 18733200
RSA签名得到的结果为:129417991423533011918602816425024443722660832
RSA验证签名的结果为:6882326879666683

Process finished with exit code 0
```

图 2: RSA 运算结果

5.2 ELGamal 数字签名

5.2.1 实验（2）

采用上次实验中 ELGamal 密码算法的相关参数对于 M 进行签名及验证

签名程序运行结果如下图所示

```
E:\Python_code\venvs\Scripts\python.exe E:\Python_code\codes\cryptography\lab_6\ELGamal_name\ELGamal_name.py
ELGamal签名后的结果为:
r=123664963837373114028042872414867518238221933606104526700461562916268236069563417678565983543089319783376836744540268112573207477224173828758642043213 6
s=67982763170417604203884866269034607318300924938597876377674798821893741774126075008432465914550463013554515389892122150136310691746163933915746929324 5
ELGamal签名验证的结果为： True

Process finished with exit code 0
```

图 3: ELGamal 运算结果

5.2.2 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。

选择签名算法为 $mx = sk + r mod\, p - 1$, 选择验证算法为 $y^m = r^s \alpha^r mod\, p$

编写验证程序如下所示

```
from ELGamal import *



def elgamal_sign(message, p, g, d):
    """生成ElGamal数字签名"""
    M = message
    while True:
        k = randint(2, p - 2)
        if e_gcd(k, p - 1)[0] == 1:  # 确保k和p-1互质
            break
    r = fastExpMod(g, k, p)

    k_inv = e_gcd(k, p - 1)[1] % (p - 1)
    s = (k_inv * (M * d - r)) % (p - 1)
    return r, s



def elgamal_verify(message, r, s, p, g, y):
    """验证ElGamal数字签名"""
    M = message
    if r < 1 or r > p - 1:
        return False

    left = fastExpMod(y, M, p)
    right = (fastExpMod(g, r, p) * fastExpMod(r, s, p)) % p
    return left == right



```

```python
# 示例使用
if __name__ == '__main__':
    message = read_file("message.txt")

    p, y, d, g = generate_key()  # 生成ElGamal密钥
    # 生成签名

    r, s = elgamal_sign(int(message), p, g, d)
    print("ELGamal签名后的结果为：")
    print("r=" + str(r))
    print("s=" + str(s))

    # 验证签名
    m = elgamal_verify(int(message), r, s, p, g, y)
    print("ELGamal签名验证的结果为：", m)
```

代码 4: ELGamal 变形签名

该程序的运行结果为

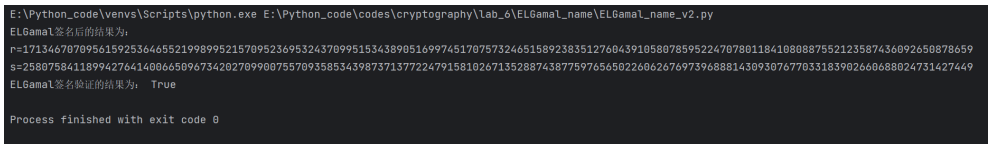

图 4: ELGamal 变形运算结果

5.2.3 实验（4）

设 $r = \alpha^k mod\, p$ 根据签名算法的一般形式 $Ak = B + Cxmod\, p - 1$, 以及对应的验证算法的一般形式 $r^A = \alpha^C y^B mod\, p$, 自己尝试设计新的基于离散对数的数字签名方案，并对于 M 进行签名及验证。

5.3 SM2 数字签名

SM2 数字签名程序运行结果如下图所示

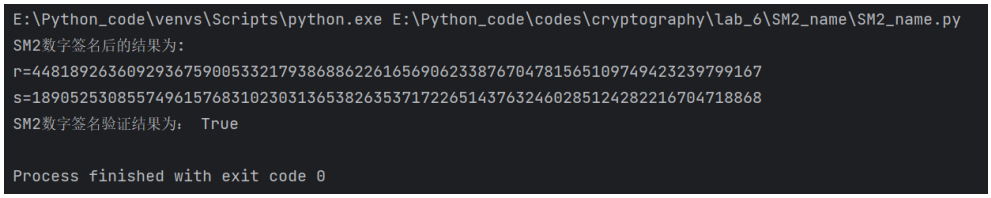

图 5: SM2 运算结果

六、 分析与讨论

通过实践 RSA、ElGamal 和 SM2 等不同类型的数字签名方案，我不仅掌握了它们的理论原理，而且对它们在实际应用中的重要性有了更深的理解。特别是，我学会了如何在 Python 环境中实现这些算法，这让我对编程实现密码学算法的过程有了更全面的认识。此外，我也对这些算法的安全性有了更深刻的认识，特别是在面对不同类型的攻击时。这次实验不仅增强了我的技术技能，也提高了我的安全意识。

七、 教师评语