武汉大学国家网络安全学院
课程实验报告

公钥密码 RSA、ELGamal、ECC

专 业 、 班 ：	信安 9 班
课 程 名 称 ：	密码学实验
指 导 教 师 ：	王后珍
实 验 地 点 ：	C102
学 生 学 号 ：	2021302141097
学 生 姓 名 ：	李宇

2023 年 **12** 月 **29** 日

公钥密码 RSA、ELGamal、ECC 的实现

一、 实验目的

1. 掌握公钥密码的概念和基本工作方式；
2. 掌握 RSA 密码、ElGamal 密码和椭圆曲线密码的原理与算法；
3. 了解 RSA 密码、ElGamal 密码和椭圆曲线密码的安全性；
4. 了解 RSA 密码、ElGamal 密码和椭圆曲线密码的应用。

二、 实验内容及原理

1. 实现 RSA 密码的加解密
2. 实现 ElGamal 密码的加解密
3. 实现椭圆曲线密码的加解密

三、 实验环境

- 操作系统：Windows11 家庭版
- 基本硬件：CPU：AMD Ryzen 5 5600H, 16G 内存
- 所用软件：Pycharm 2023.2.1 专业版
- 所用语言：python3.10

四、 RSA 密码实现

RSA 公开密钥密码体制。所谓的公开密钥密码体制就是使用不同的加密密钥与解密密钥，是一种“由已知加密密钥推导出解密密钥在计算上是不可行的” 密码体制。该算法的实现过程如下所示：

* 密钥生成：

    1. 选择两个大素数 p 和 q：
        - 随机选择两个大素数，通常在几百位到几千位之间。
    2. 计算模数 n：
        - 计算模数 $n = pq$。
    3. 计算欧拉函数 $\phi(n)$：
        - 计算欧拉函数 $\phi(n) = (p-1)(q-1)$。
    4. 选择公钥 e：
        - 选择一个与 $\phi(n)$ 互质的整数 e，通常选择 65537。

5. 计算私钥 d:
   - 计算私钥 d，使得 $ed \equiv 1 \pmod{\phi(n)}$。
6. 公钥和私钥:
   - 公钥为 (n, e)。
   - 私钥为 (n, d)。

* 加密:

1. 将消息转换为数字:
   - 将要加密的消息转换为数字 m。
2. 计算密文 c:
   - 计算密文 $c \equiv m^e \pmod{n}$。

   密文为 c。

* 解密:

1. 计算明文 m:
   - 计算明文 $m \equiv c^d \pmod{n}$。

   明文为 m。

下面给出每部分的具体代码实现。

4.1 素数生成

RSA 首先需要指定两个大素数 PQ，然而此素数通常为 256bit、512bit 或者更高，因此需要我们使用代码来生成后续过程中所需的大素数 P,Q. 如下代码所示:

```
def generate_PQ(size):
    '''
    size: 产生的素数位数
    '''
    res=[0,0]
    for j in range(2):
        index = size
        print(index, "位质数: ", end="")
        num = 0
        for i in range(index):
            num = num * 2+ randint(0, 1)
        while is_prime(num) == False:
            num = num + 1
        res[j]=num
        print(num)
        print("---------------------------")
```

```python
    if res[0]==res[1]:
        raise ValueError('大素数产生失败，请调整素数位数重新生成')
    return res[0],res[1]
```

(1) 这段代码用于生成两个随机的大素数 P 和 Q，其位数由参数 size 指定。
(2) 通过循环两次，每次生成指定位数的随机二进制数，然后递增直到找到素数。
(3) 这里调用了 is_prime() 函数判断是否为素数

而 is_prime() 函数又是基于 miller_rabin 的素性判定算法，Miller-Rabin 是一种素性检测算法。首先，它处理了一些特殊情况，如 1、2 和偶数，然后将 p - 1 分解为 m * 2^k 的形式。接着，它随机选择一个整数 a，通过一系列计算检查 p 是否可能为合数。这个检测过程重复多次，通过多次随机选择不同的 a 值，以提高准确性。如果所有检测都通过，函数返回 True，表示 p 可能是素数；否则返回 False，表明 p 不是素数。

```python
def miller_rabin(p):
    if p == 1: return False
    if p == 2: return True
    if p % 2== 0: return False
    m, k, = p - 1, 0
    while m % 2== 0:
        m, k = m // 2, k + 1
    a = randint(2, p - 1)
    x = pow(a, m, p)
    if x == 1or x == p - 1: return True
    while k > 1:
        x = pow(x, 2, p)
        if x == 1: return False
        if x == p - 1: return True
        k = k - 1
    return False

def is_prime(p, r = 40):
    for i in range(r):
        if miller_rabin(p) == False:
            return False
    return True
```

4.2 计算 n,m,e

根据 RSA 的算法流程，计算 n，m 并寻找与欧拉函数 n 互素的数 e。代码如下所示：

```python
def caculate_n(P,Q):
    return P*Q
```

```python

def Euler(P,Q):
    return (P-1)*(Q-1)

def find_e(Euler_n):
    for e in range(2,Euler_n):
        if gcd(e,Euler_n)==1:
            return e
    raise ValueError("不能找到合适条件的e，请调整素数位数")
```

在找 e 的过程中，调用 gcd() 函数，使满足 gcd(e,Euler_n) = 1（互素）。具体的 gcd 函数如下：采用了辗转相除法也就是欧几里得算法，公式如下：

$$\begin{aligned} d &= \gcd(a,b) \\ d &= \gcd(b, a \bmod b) \end{aligned} \Rightarrow \gcd(a,b) = \gcd(b, a \bmod b)$$

$$\gcd(a,b) = \gcd(b, a \bmod b) = \cdots = \gcd(m,0)$$

关于辗转相除法的代码实现如下，通过递归的思路进行处理.

```python
def gcd(a,b):
    a,b=adjust(a,b)
    if b==0:
        return a
    else :
        return gcd(b,a%b)

def adjust(a,b):
    if a>=b:
        return a,b
    else :
        return b,a
```

4.3 计算 d

get_d 使用的是扩展的欧几里得算法，而该算法实际上就是欧几里得算法的逆推过程，在公约数为 1 的情况下（互素）找到互素的数.

```python
def get_d(a, m):
    if gcd(a, m) != 1:
        return None
```

```python
    u1, u2, u3 = 1, 0, a
    v1, v2, v3 = 0, 1, m
    while v3 != 0:
        q = u3 // v3
        v1, v2, v3, u1, u2, u3 = (u1 - q * v1), (u2 - q * v2), (u3 - q * v3),
            v1, v2, v3
    return u1 % m
```

至此 RSA 算法最核心的公私钥对已经生成，计算所得的 e（私钥）需要好好保存，而公钥 d 可以公开，n 也可以公开。现在就能实现文件公钥加密，私钥解密的效果。

4.4 加解密部分

首先实现加解密中都要用到的模幂算法。

```python
def fast_modular_exponentiation(base, exponent, modulus):
    result = 1

    base = base % modulus # 将底数取模以避免溢出

    while exponent > 0:
        # 如果指数为奇数，则乘以当前底数
        if exponent % 2== 1:
            result = (result * base) % modulus

        # 将指数除以2，底数平方
        exponent //= 2
        base = (base * base) % modulus
```

在 RSA 的真正加解密过程时，是很简单的，只需要通过模乘运算即可得到密文/明文，加解密函数如下所示：

```python
def RSA_encode(message,e,n):
    encode_list = []
    for i in range(len(message)):
        encode_list.append(fast_modular_exponentiation(ord(message[i]), e, n))
    return encode_list

def RSA_decode(encode_list,d,n):
    message=''
    for v in encode_list:
        message+=chr(fast_modular_exponentiation(v, d, n))
    return message
```

五、 ElGamal 密码实现

* 密钥生成:

  1. 选择大素数 p 和本原元 g:
     - 随机选择一个大素数 p。
     - 选择一个模 p 的本原元 g，确保 g 的阶等于 $\phi(p) = p - 1$。
  2. 选择私钥 x:
     - 随机选择一个私钥 x，属于区间 $[1, p - 2]$。
  3. 计算公钥 y:
     - 计算公钥 $y \equiv g^x \pmod p$。

     公钥为 (p, g, y)，私钥为 x。

* 加密:

  1. 选择随机数 k:
     - 随机选择一个整数 k，属于区间 $[1, p - 2]$。
  2. 计算临时公钥 a 和共享密钥 s:
     - 计算临时公钥 $a \equiv g^k \pmod p$。
     - 计算共享密钥 $s \equiv y^k \pmod p$。
  3. 将消息 m 加密:
     - 将消息 m 乘以共享密钥 s，得到密文 c_1。
     - 将临时公钥 a 的离散对数加密得到密文 c_2。

     密文为 (c_1, c_2)。

* 解密:

  1. 计算共享密钥 s:
     - 计算共享密钥 $s \equiv c_2^x \pmod p$。
  2. 计算共享密钥的逆元 s^{-1}:
     - 计算共享密钥的逆元 $s^{-1} \pmod p$。
  3. 解密消息 m:
     - 计算原始消息 $m \equiv c_1 \cdot s^{-1} \pmod p$。

ElGamal 算法的安全性基于离散对数问题的困难性。由于计算 g^x 的离散对数是一个困难的问题，因此除非私钥 x 被泄漏，否则攻击者很难还原出共享密钥 s，从而无法解密消息 m。

5.1 ELGamal 加解密实现

得益于 RSA 算法各个组件间的独立性，在实现 ELGamal 时需要改动的内容并不多，例如大素数的生成、模幂运算、欧几里得算法等均可直接使用之前的函数进行，只需要编写相应的加解密函数即可。这也说明了我们实现的算法就有高耦合低内聚的特性，具有很好的可以移植性。

ELGamal 加密函数如下：

```python
def ELGamal_encode(message, p, alpha, y):
    encode_list = []
    k = randint(2, p - 2)
    U = fast_modular_exponentiation(y, k, p)
    C1 = fast_modular_exponentiation(alpha, k, p)
    for i in range(len(message)):
        M = ord(message[i])
        C2 = U * M % p
        encode_list.append((C1, C2))
    return encode_list
```

ELGamal 解密函数如下：

```python
def ELGamal_decode(encode_list, d, p):
    res = ''
    for (c1, c2) in (encode_list):
        V = fast_modular_exponentiation(c1, d, p)
        V2 = get_d(V, p) # 求V在模p下的逆
        M = c2 * V2 % p
        res += chr(M)
    return res
```

六、 椭圆曲线密码实现

6.1 ECC 原理说明

在实现 ECC 的过程中所需的求模逆函数、欧几里得算法等均可继续沿用 RSA 中实现好的函数，需要重点添加的是 GF(p) 上的椭圆曲线上特殊的加法运算。关于此类特殊的加法运算定义如下所示：

设 E 是有限域 GF(p) 上的椭圆曲线，椭圆曲线点的加法运算定义如下：

1. **点的相加：** 设 P 和 Q 是曲线上的两个点，它们的坐标为 (x_P, y_P) 和 (x_Q, y_Q)。点 P 和 Q 的加法运算 $P+Q$ 是另一点，坐标为 (x_R, y_R)。
2. **加法规则：**
   - 如果 $P \neq Q$，则通过以下公式计算 $P+Q$ 的坐标：

$$x_R = \left(\frac{y_Q - y_P}{x_Q - x_P}\right)^2 - x_P - x_Q \pmod{p}$$

$$y_R = \frac{y_Q - y_P}{x_Q - x_P}(x_P - x_R) - y_P \pmod p$$

- 如果 $P = Q$，则通过以下公式计算 $P + P$ 的坐标：

$$x_R = \left(\frac{3x_P^2 + a}{2y_P}\right)^2 - 2x_P \pmod p$$

$$y_R = \frac{3x_P^2 + a}{2y_P}(x_P - x_R) - y_P \pmod p$$

其中，a 是椭圆曲线的参数。

3. **点的加法逆元：** 对于给定的点 P，它的加法逆元是点 $-P$，其坐标为 $(x_P, -y_P \mod p)$。

6.2 关键运算函数实现

对于点加运算，分为 4 种情况：

- 其中有一个点为无穷大点
- 两个点相同
- 两个点互逆
- 其他情况

代码的实现如下：

```python
def add(p1, p2, a, p):
    # 其中一个点为0元素的情况
    if p1 == 0:
        return p2
    if p2 == 0:
        return p1

    x1, y1 = p1[0], p1[1]
    x2, y2 = p2[0], p2[1]

    # 两个点互逆
    if (x1 == x2) & (y1 + y2 == 0):
        return 0
    # 两个点相同
    elif (x1 == x2) & (y1 == y2):
        l = ((3 * x1 * x1 + a) * get_d(2 * y1, p)) % p
        x3 = (l * l - 2* x1) % p
        y3 = (l * (x1 - x3) - y1) % p
        return (x3, y3)
    # 其他情况
    else:
```

```python
        l = (((y2 - y1) % p) * get_d((x2 - x1) % p, p)) % p
        x3 = (l * l - x1 - x2) % p
        y3 = (l * (x1 - x3) - y1) % p
        return (x3, y3)
```

需要注意的是，在上述过程中涉及到两个数相减的情况，这时也要确保结果是在 0 p-1 之内，也就是对相减结果也要即使进行取模运算，否则在寻找逆元时可能会出错。

而在加解密，以及公钥 Q 的生成上均需要使用特殊的倍乘算法，即将一个坐标点乘以一个常数，这种特殊乘法可使用类似模反复平方的思想来实现，内部借助上面定义好的 add 函数，具体代码如下：

```python
def mul(point, x, a, p):
    if x == 1:
        return point
    mod_point = point
    r = 0

    bit_length = x.bit_length()
    for _ in range(bit_length):
        if x & 1:
            r = add(r, mod_point, a, p)

        mod_point = add(mod_point, mod_point, a, p)
        x >>= 1

    return r
```

6.3 ECC 密钥生成

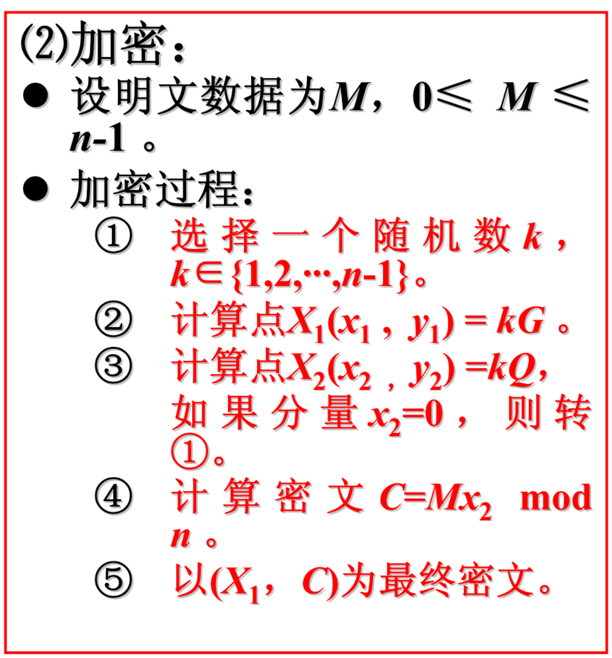
在密钥生成方面只需要随机选择一个整数 d，并对开始选择的生成元 G，利用之前定义的 mul 函数进行倍乘运算即可：

代码如下：

```python
def Gkey(d, G, a, p):
    return mul(G, d, a, p)
```

6.4 ECC 加密过程

加密过程过程如下图所示：

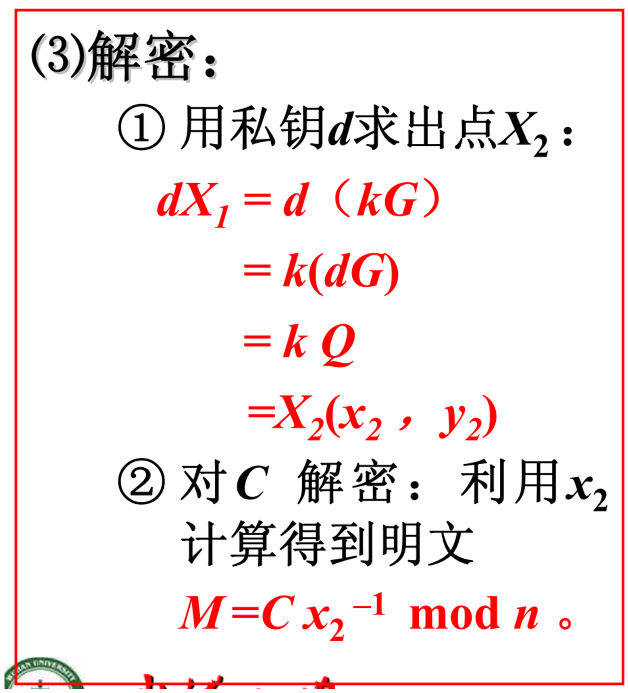


代码如下：

```python
# M是一个数值
def ECC_encode(M, n, G, Q, a, p):
    k = randint(1, n - 1)
    X2 = (0, 0)
    while (X2[0] == 0):
        X1 = mul(G, k, a, p)
        X2 = mul(Q, k, a, p)
    C = M * X2[0] % n
    return (X1, C)
```

6.5 ECC 解密过程

解密过程过程如下图所示：



代码如下：

```python
def ECC_decode(X1, C, n, a, p, d):
    X2 = mul(X1, d, a, p)
    M = C * get_d(X2[0], n) % n
    return M
```

七、 实验结果与总结

在验证部分选择明文："-say my name -heisenberg"，作为起始信息，对其使用不同的算法进行加密解密，观察解密结果是否与初始明文相同。实验所用代码已上传至 GitHub：公开密钥密码算法 python 复现代码

7.1 RSA 加解密效果

选定密钥位数为 128bit，运行下述代码：

```python
if __name__ == '__main__':
    message = "-say my name -heisenberg"
    P, Q = generate_PQ(128)
    n=caculate_n(P,Q)
    Euler_n=Euler(P,Q)
    e=find_e(Euler_n)
    d=get_d(e,Euler_n)
    cipher_text=RSA_encode(message, e, n)
    print("RSA加密所得结果",cipher_text)
    plain_text = RSA_decode(cipher_text,d,n)
    print("RSA解密结果",plain_text)
```

最终结果如下图所示，可以看到成功实现了 RSA 加解密。

```
D:\Anaconda\python.exe D:\python\密码学\RSA&ELGamal\main.py
128 位质数: 147436337982980336091416952249214045631
----------------------------
128 位质数: 198928054718965670877438285513149232281
----------------------------
RSA加密所得结果 [91125, 1520875, 912673, 1771561, 32768, 1295029, 1771561, 32768, 1331000, 912673, 1295029,
RSA解密结果 -say my name -heisenberg
```

7.2 ELGamal 加解密效果

在初始素数的选择上，我选择 p=10243, alpha= 2 ,d=666; 修改主函数代码如下：

```python
if __name__ == '__main__':
    message = "-say my name -heisenberg"
    # ELGamal
    p, alpha, d = 10243, 2, 666
    y = fast_modular_exponentiation(alpha, d, p)
    print("私钥d: {} ;  公钥y: {} ".format(d, y))
    # 公钥加密
    cipher_text = ELGamal_encode(message, p, alpha, y)
    print("ELGamal加密所得结果", cipher_text)
    # 私钥解密
```

```python
    plain_text= ELGamal_decode(cipher_text, d, p)
    print("ELGamal解密结果", plain_text)
```

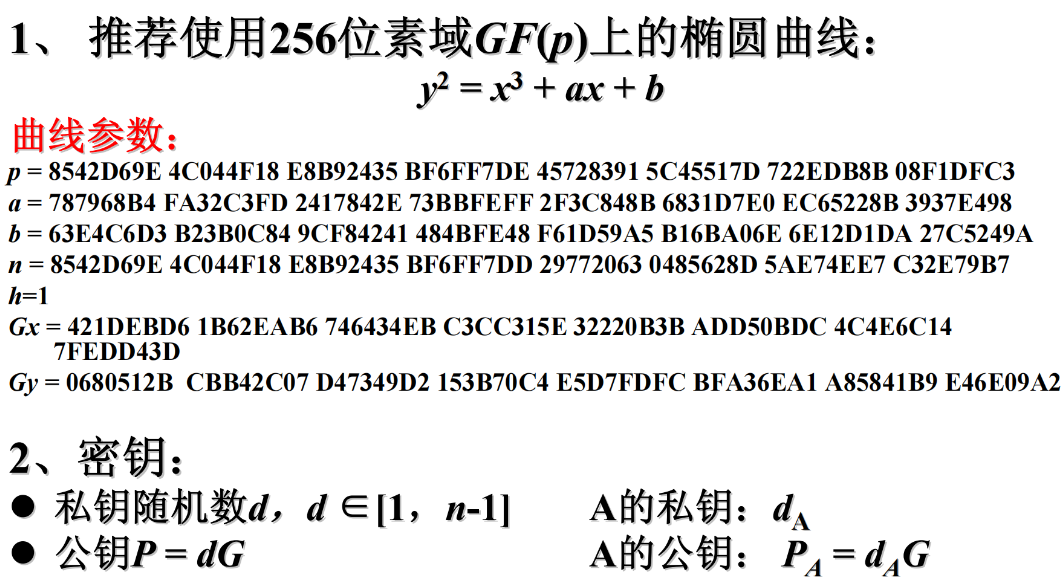
最终结果如下图所示，可以看到成功实现了 ELGamal 加解密。

```
D:\Anaconda\python.exe D:\python\密码学\RSA&ELGamal\main.py
私钥d: 666 ;  公钥y: 1370
ELGamal加密所得结果 [(2562, 500), (2562, 3554), (2562, 3354), (2562, 7035), (2562, 4908), (2562, 73), (
ELGamal解密结果 -say my name -heisenberg

进程已结束，退出代码为 0
```

7.3 ECC 加解密效果

在椭圆曲线的参数选择、大素数 p、生成元的选择上参考课程 PPT 上的推荐的 256 位素域 GF(p) 上的椭圆曲线：



同时将对字符串的处理融入加解密函数中（之前的加解密函数只是对单个数值进行加解密），如下所示：

```python
def ECC_en_message(message,n, G, Q, a, p):
    encode_list=[]
    for i in range(len(message)):
        t=ord(message[i])
        encode_list.append(ECC_encode(t,n, G, Q, a, p))
    return encode_list
def ECC_de_message(encode_list, n, a, p, d):
    r=''
    for (X1,C) in encode_list:
```

```python
        M=ECC_decode(X1,C,n, a, p, d)
        r+=chr(M)
    return r
if __name__ == '__main__':
    p = int("8542D69E4C044F18E8B92435BF6FF7DE457283915C45517D722EDB8B08F1DFC3", 16)
    a = int("787968B4FA32C3FD2417842E73BBFEFF2F3C848B6831D7E0EC65228B3937E498", 16)
    b = int("63E4C6D3B23B0C849CF84241484BFE48F61D59A5B16BA06E6E12D1DA27C5249A", 16)
    n = int("8542D69E4C044F18E8B92435BF6FF7DD297720630485628D5AE74EE7C32E79B7", 16)
    G = (int("421DEBD61B62EAB6746434EBC3CC315E32220B3BADD50BDC4C4E6C147FEDD43D", 16), \
        int("0680512BCBB42C07D47349D2153B70C4E5D7FDFCBFA36EA1A85841B9E46E09A2", 16))
    d = 100
    M = 5
    Q = Gkey(d, G, a, p)
    encode_list=ECC_en_message(message, n, G, Q, a, p)
    print("ECC加密所得结果,格式为(X1,C)")
    print(encode_list)
    plain_text=ECC_de_message(encode_list, n, a, p, d)
    print("ECC解密所得结果")
    print(plain_text)
```

运行代码结果如下，可以看到成功实现了 ECC 加解密。

```
D:\Anaconda\python.exe D:\python\密码学\RSA&ELGamal\main.py
ECC加密所得结果,格式为(X1,C)
[((7147102611013679219708830101710977563494925478897046761620202442464624072181, 2656915041373113701623451581
ECC解密所得结果
-say my name -heisenberg

进程已结束，退出代码为 0
```

7.4 个人收获

通过此次实验，我学习了公钥密码算法的三种主要代表性算法，分别是 RSA、ECC 和 ElGamal。这一学习过程深入探讨了密码学领域的关键概念和算法设计原理。

首先，RSA 算法展现了非对称密码体制的强大力量，通过合理选择大素数和巧妙构建数学问题，实现了安全的数据加密和数字签名。理解了 RSA 的密钥生成、加密和解密过程，对于公钥密码体制的基本工作原理有了更清晰的认识。

其次，ECC 算法引入了椭圆曲线的数学结构，通过有限域上的点的加法运算实现了高效的加密和

数字签名。掌握了椭圆曲线点的加法规则和密钥协商算法，对于 ECC 在资源受限环境中的广泛应用有了更深刻的理解。

最后，ElGamal 算法则突显了基于离散对数问题的密码学思想。通过学习 ElGamal 的密钥生成、加密和解密步骤，更好地理解了离散对数问题对密码学安全性的重要性，以及它在实际应用中的应用场景。

参考文献

[1] RSA ——经典的非对称加密算法（知乎）：https://zhuanlan.zhihu.com/p/450180396

[2] 超详细！ECC 椭圆曲线密码算法加密过程详解！（csdn）：https://blog.csdn.net/weixin_41754258/article/details/119595838

[3] 椭圆曲线加密算法（ECC）（知乎）:https://zhuanlan.zhihu.com/p/101907402

[4]《密码学导论》第三版

教师评语评分

评语：

评　　分：

评 阅 人：

评阅时间：