

## 云南大学数学与统计学院 上机实践报告

课程名称：近代密码学实验	年级：2015 级	上机实践成绩：
指导教师：陆正福	姓名：刘鹏	
上机实践名称：椭圆曲线离散对数问题实验	学号：20151910042	上机实践日期：2018-06-03
上机实践编号：No.06	组号：	上机实践时间：21:54

### 一、实验目的

1. 熟悉椭圆曲线离散对数问题（ECDLP）及其有关的密码体制；
2. 实现与 ECDLP 有关的基本算法；
3. 了解参数与参数规模

### 二、实验内容

1. 了解椭圆曲线离散对数问题（ECDLP）有关的算法
  2. 编程实现 Diffie-Hellman 密钥交换协议的椭圆曲线版本。
  3. 编程实现 ElGamal 加密体制的椭圆曲线版本。
- 说明：**基础有限域为素域 $GF(p)$ （ $p$ 为大素数）的情形为必做实验；基础有限域为 $GF(2^m)$ 的情形为选做实验

### 三、实验平台

Windows 10 Pro Workstation 1803;  
SageMath version 8.2, Release Date: 2018-05-05;

### 四、实验记录与实验结果分析

#### 1 题

了解与椭圆曲线离散对数（ECDLP）问题相关的算法。

**Solution:**

#### 背景材料

大多数使用公钥密码学进行加密和数字签名的产品和标准都是用 RSA 算法<sup>[1]</sup>。近年来，为了保证 RSA 使用安全性,密钥的位数一直在增加，这对于使用 RSA 体制的应用而言是一项巨大的负担，对进行大量安全交易的电子商务与银行系统而言更是如此。近你来出现的椭圆曲线密码学（ECC）对 RSA 提出了挑战。ECC 的主要优势在于，它可以使用比 RSA 短得多的密钥得到相同安全性，减少处理荷载。

椭圆曲线并不是椭圆，之所以称之椭圆曲线为这一类方程的样式，与计算椭圆周长的方程类似，也使用三次方程来表示的。一般，椭圆曲线的三次方程形式为

$$y^2 + axy + by = x^3 + cx^2 + dx + e$$

其中， $a$ ， $b$ ， $c$ ， $d$ 和 $e$ 是实数， $x$ 和 $y$ 是取值在实数集上的变量。在椭圆曲线加密中，并不需要这种普通形式，下述形式已经足够：

$$y^2 = x^3 + ax + b$$

这是一个三次方程。椭圆曲线的定义中，还需要一个称作无穷远点或者零点的元素，记作 $O$ 。

当方程满足 $4a^3 + 27b^2 \neq 0$ 时，以椭圆曲线上的所有点作为集合，可以定义一种加法，进而作出一个阿贝尔群，即一

个符合封闭性、加法结合律、加法单位元、逆元存在、加法交换律这 5 条性质的代数群。该加法是这样描述的：

- (1) 无穷远点  $O$  被称为该加法的单位元，在椭圆曲线上任取一点  $P$ ，都有  $P + O = O + P = P$ ；
- (2)  $\forall P = (x, y)$ ，其逆元为  $-P = (x, -y)$ ；
- (3) 若椭圆曲线上的三个点在共线，则认为这三个点的和为  $O$ ，即若  $P, Q, R$  三点共线，则  $P + Q + R = O$ ，进一步  $P + Q = -R$ 。也就是说，两个不在同一条竖直线上的点，和为与之共线且在椭圆曲线上的第三点相对于横轴的镜像对称点。
- (4) 对于同一个点，计算其 2 倍，只需做出该点的切线，并由此寻找该切线另外的与椭圆曲线相交的点的横轴景象。

很显然，这是一个阿贝尔群（在一些其他条件下），即交换群。因为任取两点，相加的顺序与第三点存在的位置无关。

以上是实数域上的椭圆曲线描述，在椭圆曲线密码体制中，使用的变元和系数均为有限域中元素的椭圆曲线。椭圆曲线密码体制使用两种椭圆曲线，分别是适合软件实现的定义在  $\mathbb{Z}_p$  上的素曲线（prime curve）和适合硬件实现的定义在  $\text{GF}(2^m)$  上的二元曲线。

先讨论素曲线的情形。此时变量和系数均在  $\mathbb{Z}_p$  里面取值

$$y^2 \bmod p = (x^3 + ax + b) \bmod p$$

这样的代数系统可以记为  $E_p(a, b)$ 。作为有限域，必然包含加法与乘法这两种运算。把 ECC 中的加法运算与 RSA 中的模乘运算相对应，将 ECC 中的乘法运算与 RSA 中的模幂运算对应。如果建立基于椭圆曲线的密码体制，需要类似因子分解两个素数之积或求离散对数这样的难题。考虑方程  $Q = k \cdot P$ ，其中  $Q, P \in E_p(a, b)$  且  $k < p$ ，对于给定的  $k$  和  $P$  计算  $Q$  比较容易，而对给定的  $Q$  和  $P$  计算  $k$  比较困难。这就是椭圆曲线的离散对数问题。

可以考虑，若给定了  $k$  和  $P$ ，则可以通过类似快速幂取模算法的步骤，迅速得出  $Q$ ，但是通过碰撞的方式去解  $k$ ，就不得不一次一次累加在  $\mathbb{Z}_p$  上计算一次椭圆曲线加法是比较消耗时间的。若  $P = (x_P, y_P)$ ， $Q = (x_Q, y_Q)$ ，且  $P \neq Q$ ，则  $R = P + Q = (x_R, y_R)$  由下列规则确定：

$$\begin{aligned} x_R &= (\lambda^2 - x_P - x_Q) \bmod p \\ y_R &= (\lambda(x_P - x_R) - y_P) \bmod p \end{aligned}$$

其中

$$\lambda = \begin{cases} \left( \frac{y_Q - y_P}{x_Q - x_P} \right) \bmod p & P \neq Q \\ \left( \frac{3x_P^2 + a}{2y_P} \right) \bmod p & P = Q \end{cases}$$

椭圆曲线离散对数为什么难以计算？这是因为在  $P = n \times G$  中，如果  $n$  可以取得很大，那么计算就很困难，而这里的  $n$  是密码体制中的私钥，取得很大没有问题，因为并不进行私钥的传输。而模数  $p$  仅是确定有限阿贝尔群中点的个数，在  $E_p(a, b)$  中，点的个数  $N$  的范围是

$$p + 1 - 2\sqrt{p} \leq N \leq p + 1 + 2\sqrt{p}$$

大体上看， $E_p(a, b)$  中点的个数约等于  $\mathbb{Z}_p$  中元素的个数，即  $p$  个。在私钥生成公钥的公式中，乘子  $n$  不能比  $\text{order}(G)$  还要大，因为若  $n = \text{order}(G)$ ，那么  $P$  就是无穷远点；若  $n \geq \text{order}(G)$ ，就会造成重复计算浪费计算力。

## 2 题

编程实现 Diffie-Hellman 密钥交换协议的椭圆曲线版本。

程序代码

```
1  q = next_prime(2^256)      # A public integer, should be very big
2  R = IntegerModRing(q)
```

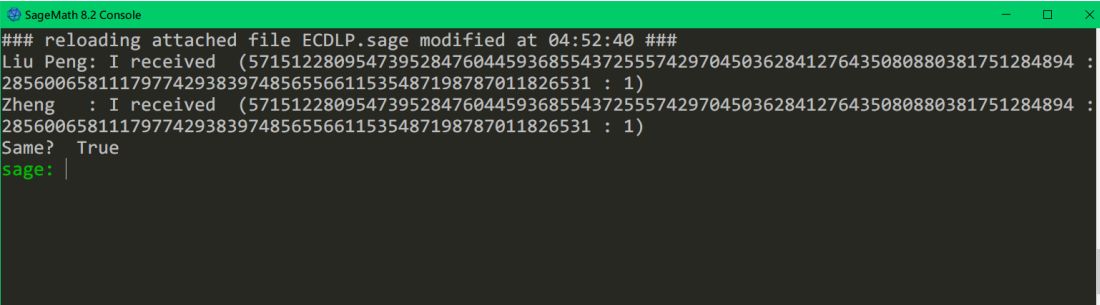
```

3  a = R(2)
4  b = R(9)
5
6  # check
7  if R(4*a^3 + 27*b^2) == 0:
8      raise ValueError("bad")
9
10 E = EllipticCurve(GF(q), [a,b])
11 G = E([0,3]) # base point, order(G) = 2373
12
13 Private_Key_Liupeng = 19961019
14 Public_Key_Liupeng = Private_Key_Liupeng * G
15
16 Private_Key_Zheng = 19970323
17 Public_Key_Zheng = Private_Key_Zheng * G
18
19 print "Liu Peng: I received ", Private_Key_Liupeng * Public_Key_Zheng
20 print "Zheng : I received ", Private_Key_Zheng * Public_Key_Liupeng
21
22 a = Private_Key_Liupeng * Public_Key_Zheng == Private_Key_Zheng * Public_Key_Liupeng
23 print "Same? ", a

```

程序代码 1

## 运行结果



```

SageMath 8.2 Console
### reloading attached file ECDLP.sage modified at 04:52:40 ###
Liu Peng: I received (57151228095473952847604459368554372555742970450362841276435080880381751284894 :
2856006581117977429383974856556611535487198787011826531 : 1)
Zheng : I received (57151228095473952847604459368554372555742970450362841276435080880381751284894 :
2856006581117977429383974856556611535487198787011826531 : 1)
Same? True
sage:

```

运行结果 1

## 3 题

编程实现 ElGamal 加密体制的椭圆曲线版本。

## 背景材料

如果可以将明文 $m$ 进行编码，使之转换为 $E_p(a, b)$ 上的点的形式，就可以进行椭圆曲线加密。如下，A 用户要发送加密消息给 B 用户，假设消息已经通过某种公开方法完成了编码，那么可以通过如下方式生成公钥，加密明文以及解密密文。

$$\begin{aligned}
 P_B &= n_B \times G \\
 C_m &= (kG, P_m + kP_B) = (C_1, C_2) \\
 d_{n_B}(C_m) &= C_2 - n_B C_1
 \end{aligned}$$

可见，编码是进行加密的前提。有一种 ECC-ElGamal 的变种，叫做 MV-ElGamal，可以避免这个需要编码的问题。在这个加密变种里面，明文消息 $M$ 变为了两块： $m_1, m_2$ 。这也意味着，明文变成了 $\mathbb{Z}_p \times \mathbb{Z}_p$ 中的元素。加密算法如下：

$$e_{P_B}(M = (kG, c_1, c_2)) \in E_p(a, b) \times \mathbb{Z}_p \times \mathbb{Z}_p$$

其中,  $c_1 \equiv xm_1 \pmod{p}$ ,  $c_2 \equiv ym_2 \pmod{p}$ , 这里  $(x, y) = kP_B$ 。这里假设  $x, y \neq 0$ , 否则就重新选一个  $k$ 。可以看出, 这种编码确实解决了复杂编码的问题, 通过  $kP_B$  的两个坐标, 也可以迅速加密分组的信息, 同时, 信息是可以还原的, 因为  $(x, y) = kP_B = kn_B G$ , 即最后只要用私钥乘指数式  $kG$  即可。所以相应的解密函数为

$$d_{n_B}(C_0, c_1, c_2) = (c_1 x^{-1}, c_2 y^{-1})$$

这里的  $(x, y) = kC_0$ , 之后合并输出  $c_1$  与  $c_2$ , 就可以输出一组解密的消息。这个算法仍旧保留了 ElGamal 的分组密码本质。

#### 程序代码

```

1  #
2  # Choose the elliptic curve modulo p = large prime
3  # and G a point of high order
4  #
5  p = next_prime(10^10)
6  E = EllipticCurve(GF(p), [2011 ,1])
7  G = E([0, 1])
8  print G.additive_order()
9
10 #
11 # Encryption and decryption functions
12 #
13 def encrypt_mv_eg(Kpub ,m1 ,m2 ):
14     x, y = 0, 0      # temp value
15     while ((x == 0) or (y == 0)):
16         r = floor(p * random())
17         x = (r * Kpub)[0]
18         y = (r * Kpub)[1]
19     return r*G, m1*x, m2*y
20 def decrypt_mv_eg (kpri ,enc ):
21     x = (kpri * enc[0])[0]
22     y = (kpri * enc[0])[1]
23     return enc[1] * x^-1, enc [2] * y^-1
24
25 #
26 # Example
27 #
28 private_key = 19961019
29 public_key = private_key * G
30 decrypt_mv_eg(private_key, encrypt_mv_eg(public_key, 10101, 33333))
31
32 # text to number
33 def encoding(text):
34     result = 0
35     for c in text :
36         result = 256 * result + ord(c)      # bit operation
37     return result

```

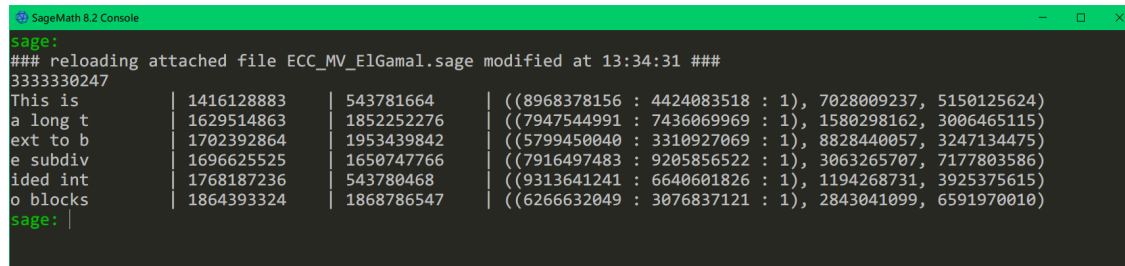
```

38
39 # number to text
40 def decoding(number):
41     number = Integer (number)
42     result = ''
43     for i in number.digits(256):
44         result = chr(i) + result
45     return result
46
47 #
48 # TABLE for a long text
49 # 1st column : Decoded and decrypted text ( original message )
50 # 2nd , 3rd: encoded blocks
51 # rest : encrypted blocks
52 #
53 text = 'This is a long text to be subdivided into blocks'
54 k = floor(log(p, 256))      # log_256(p), 8 bit
55 key = 19961019
56 for i in range(0, len(text), 2*k):
57     m1 = encoding(text[i:i+k])
58     m2 = encoding(text[i+k:i+2*k])
59     enc = encrypt_mv_eg (key*G, m1, m2)
60     d1 = decoding(decrypt_mv_eg(key, enc)[0])
61     d2 = decoding(decrypt_mv_eg(key, enc)[1])
62     print d1 + d2, "\t|", m1, "\t|", m2, "\t|", enc

```

程序代码 2

运行结果



```

sage:
### reloading attached file ECC_MV_ElGamal.sage modified at 13:34:31 ###
3333330247
This is      | 1416128883      | 543781664      | ((8968378156 : 4424083518 : 1), 7028009237, 5150125624)
a long t     | 1629514863      | 1852252276     | ((7947544991 : 7436069969 : 1), 1580298162, 3006465115)
ext to b     | 1702392864      | 1953439842     | ((5799450040 : 3310927069 : 1), 8828440057, 3247134475)
e subdiv     | 1696625525      | 1650747766     | ((7916497483 : 9205856522 : 1), 3063265707, 7177803586)
ided int     | 1768187236      | 543780468      | ((9313641241 : 6640601826 : 1), 1194268731, 3925375615)
o blocks     | 1864393324      | 1868786547     | ((6266632049 : 3076837121 : 1), 2843041099, 6591970010)
sage:

```

运行结果 2

## 六、实验体会

在由椭圆曲线构造出来的素域里进行数乘运算是需要进行算法优化的，就像快速幂取模算法一样，如果直接进行连加会非常耗费时间，在生成素域中的初始元 $G$ 之后，我本想用循环做（当时还不清楚 SageMath 已经做了乘号运算符重载），结果很久都出不来公钥，最后只好作罢。

## 七、参考文献

- [1] STALLINGS W. 密码编码学与网络安全：原理与实践 [M]. 6th ed. 北京：机械工业出版社, 2015.
- [2] [https://doc.sagemath.org/html/en/reference/curves/sage/schemes/elliptic\\_curves/ell\\_point.html](https://doc.sagemath.org/html/en/reference/curves/sage/schemes/elliptic_curves/ell_point.html)