# Ellipse Curve Cryptography (ECC)

## 椭圆曲线

- 定义
    - 椭圆曲线是在射影平面上满足Weierstrass方程所有点的集合：
        - $Y^2Z+a_1XYZ+a_3YZ^2=X^3+a_2X^2Z+a_4XZ^2+a_6Z^3$
    - 曲线上的每个点都必须是非奇异的（光滑的），即各变量的偏导不同为0。
- 普通方程Ep(a,b)
    - $y^2=x^3+ax+b$
- 椭圆曲线 Abel 群(交换幺环)
    - 满足封闭性、结合性、有单位元、逆元，可交换
- 有限域椭圆曲线
    - $y^2=x^3+ax+b(mod p)$
    - 加乘除都需要 mod p
    - 约束条件：$4a^3+27b^2!= 0(mod p)$
- 取圆上两点P(x1,y1),Q(x2,y2)计算斜率：
    - $P=Q,k=(3x_2+a)/2y_1(mod p)$
    - $P!=Q,k=(y_2-y_1)/(x_2-x_1)(mod p)$
- 计算$P+Q$
    - $P+Q=k^2-x_1-x_2(mod p)$
- 阶 rank
    - nP=0,则n为P的阶，P为曲线一点。

## 椭圆曲线加密

这里取我已写好的代码的注释部分解释。

```
# ECC-Simple Example

# ECC key's generator steps
# 1.Ep(a,b)-->y^2=x^3+a*x+b(mod p),set the value of a,b and prime p.
# 2.check the validity ---> 4*a^3+27*b^2 != 0 (mod p)
# 3.choose a point G(x,y) of Ep(a,b),then calculate it's rank n.
# 4.consider K=kG with k<n.
# 5.finally, (E,K,G) is the public key and k is the private key of ECC.
#
```

```
# ECC enc2dec steps
# --------------enc
# 1.coding the message M into a point M of Ep(a,b)
# 2.randomly choose a r satisfy r < n.
# 3.calculate C1 = M + rK,C2 = rG,then get the (C1,C2)
# --------------dec
# 1.calculate M = C1 - k*C2 because of C1 - k*C2 = M+rK-krG=M+rkG-krG=M
```

## 代码

代码中注释我已详尽，这里阐述总体思路：

1. 选取a,b,p生成Ep(a,b)
2. 选择小密钥基点G，选取小密钥k，生成公钥K。
3. 至此，公密钥生成完成。
4. 加密：选择随机数r，将密文与rK相加（代码简化为密文与rK的x坐标相乘）为C1，rG为C2，得到(C1,C2)。
5. 解密：M=C1-k*C2（代码简化为M=C1/r_K_x）

```
# ECC-Simple Example
# author:LiuDingyi 19307130247
#
# ECC key's generator steps
# 1.Ep(a,b)-->y^2=x^3+a*x+b(mod p),set the value of a,b and prime p.
# 2.check the validity ---> 4*a^3+27*b^2 != 0 (mod p)
# 3.choose a point G(x,y) of Ep(a,b),then calculate it's rank n.
# 4.consider K=kG with k<n.
# 5.finally, (E,K,G) is the public key and k is the private key of ECC.
#
# ECC enc2dec steps
# --------------enc
# 1.coding the message M into a point M of Ep(a,b)
# 2.randomly choose a r satisfy r < n.
# 3.calculate C1 = M + rK,C2 = rG,then get the (C1,C2)
# --------------dec
# 1.calculate M = C1 - k*C2 because of C1 - k*C2 = M+rK-krG=M+rkG-krG=M

def get_inverse(mu, p):
    """
    get mu's inverse element
    """
    for i in range(1, p):
        if (i*mu)%p == 1:
            return i
    return -1


def get_gcd(zi, mu):
    """
    Gets the greatest common divisor of zi and mu
    """
    if mu:
        return get_gcd(mu, zi%mu)
    else:
        return zi

def get_np(x1, y1, x2, y2, a, p):
    """
    P(x1,y1),Q(x2,y2)--->(P+Q)(x3,y3)
    """

    flag = 1  # consider +/-
    # get the gradient k
    # case1:if p=q  k=(3x2+a)/2y1 (mod p)
    if x1 == x2 and y1 == y2:
        zi = 3 * (x1 ** 2) + a
        mu = 2 * y1

    # case2:if p!=q  k=(y2-y1)/(x2-x1) (mod p)
    else:
        zi = y2 - y1
        mu = x2 - x1
        if zi * mu < 0:
            flag = 0
            zi = abs(zi)
            mu = abs(mu)

    gcd_value = get_gcd(zi, mu)
    zi = zi // gcd_value
    mu = mu // gcd_value

    # k = zi * mu^(-1) (mod p)
    inverse_value = get_inverse(mu, p)
    k = (zi * inverse_value)

    if flag == 0:
        k = -k
    k = k % p

    # calculate (P+Q)-->(x3,y3)

    x3 = (k ** 2 - x1 - x2) % p
    y3 = (k * (x1 - x3) - y1) % p
    return x3,y3

def get_param(x0, a, b, p):
    """
    calculate p and -p
    """
    y0 = -1
    for i in range(p):
        # find the i which satisfy condition
        if i**2%p == (x0**3 + a*x0 + b)%p:
            y0 = i
            break

    if y0 == -1:
        return False

    x1 = x0
    y1 = (-1*y0) % p
    return x0,y0,x1,y1

def get_points(a,b,p):
    """
    calculate points in range(0,1-p)
    """
    list = []
    index = 1
    for i in range(p):
        val =get_param(i, a, b, p)  # get the point of Ep(a,b)
        if(val != False):
            x0,y0,x1,y1 = val
            list.append([x0,y0])
            list.append([x1,y1])
            print("point {}: ({},{})".format(index,x0,y0))
            index+=1
            print("point {}: ({},{})".format(index,x1,y1))
            index+=1
    return list


def get_rank(x0, y0, a, b, p):
    """
    get the rank
    """
    x1 = x0
    y1 = (-1*y0)%p
    tempX = x0
    tempY = y0
    n = 1
    while True:
        n += 1
        p_x,p_y = get_np(tempX, tempY, x0, y0, a, p)
        if p_x == x1 and p_y == y1:
            return n+1
        tempX = p_x
        tempY = p_y

def get_public(G_x, G_y, key, a, p):
    """
    calculate nG
    """
    temp_x = G_x
    temp_y = G_y
    while key != 1:
        temp_x,temp_y = get_np(temp_x,temp_y, G_x, G_y, a, p)
        key -= 1
    return temp_x,temp_y

def ECC():
    err = Exception('parameters error!please input again!')
    a = int(input("set the value a(>0): "))
    b = int(input("set the value b(>0): "))
    p = int(input("set the prime value p: "))
    # check the validity of a and b.
    if (4*(a**3)+27*(b**2))%p == 0:
        raise err

    # get the points of Ep(a,b)(mod p).
    list = get_points(a,b,p)
    index = int(input('Choose a points as G(input the order):'))
    G_x = list[index][0]
    G_y = list[index][1]

    # get the rank of G(x,y).
    n = get_rank(G_x,G_y,a,b,p)

    # generate the k and K
    k = int(input("choose k (<{}) : ".format(n)))
    K_x,K_y = get_public(G_x,G_y,k,a,p)

    # enc:(E,K,G) and n and message ---> (C1= M + rK,C2= rG)
    # Simplify:C1= M + rK ---> C1 = Ascii(m)*r_K_x
    # then dec ---> k*C2=r*k*G=r*K=rK,m=C1/r_K_x
    r = int(input("choose r (<{}) : ".format(n)))
    r_G_x,r_G_y = get_public(G_x, G_y, r, a, p)     # rG
    r_K_x,r_K_y = get_public(K_x, K_y, r, a, p)     # rK

    message = input("input the string need to encrypt:")
    message = message.strip()
    c = []
    print("enc-message: ",end="")
    for char in message:
        ascii_char = ord(char)
        cipher_text = ascii_char*r_K_x
        c.append([r_G_x, r_G_y, cipher_text])
        print("({},{}),{}".format(r_G_x, r_G_y, cipher_text),end="-")

    print("\ndec-message: ",end="")
    for charArr in c:
        decrypto_text_x,decrypto_text_y = get_public(charArr[0], charArr[1], k, a,
p)
        print(chr(charArr[2]//decrypto_text_x),end="")


if __name__ == "__main__":
    print("************ECC start************")
    ECC()
```