实验项目名称： 椭圆曲线密码算法 实验学时： 6

同组学生姓名： 无 实验地点： 科技楼4-1204

实验日期： 2023.12.13 实验成绩：

批改教师： 黄丹丹 批改时间： 2023.12.22

实验7 椭圆曲线密码算法

一、基于OBE模式的实验目的和要求

1、熟悉ECC加解密算法的构造和运行过程。

2、使用C/C++语言编写实现ECC算法程序，加深对ECC点加法的理解。

二、实验仪器和设备

Visual C/C++

三、实验过程

1、ECC密码体制是IEEE公钥密码标准P1363确定的公钥密码算法之一。有限域*Fp*上的椭圆曲线是指满足方程

*y*2 ≡ (*x*3+*ax*+*b*) mod *p*

的所有点(*x*, *y*)再加上一个无穷远点*O*构成的集合，其中*a*, *b*, *x*和 *y* 均在有限域*Fp*上取值，*p*是素数。对于椭圆曲线上的两点*P*, *Q*, 如果*P*=(*x, y*)，*Q*=(*x, -y*)，那么*S* = *P*+*Q* = *O*，即点(*x*, -*y*)是*P*的加法逆元，表示为 *-P*。

如果*P*, *Q*不互为逆元，*P*=(*x*1, *y*1)，*Q*=(*x*2, *y*2)， *S*=*P*+*Q*= (*x*3, *y*3)由以下公式确定：



其中，



倍点运算定义为重复加法，例如4*P* = *P*+*P*+*P*+*P*。

椭圆曲线密码算法的安全性基于椭圆曲线离散对数问题：设*P*为椭圆曲线上的点，*Q* = *xP*（倍点），*x*为正整数，已知*P*和*Q*很难确定出*x*的值。在本实验中，假设p为len比特素数，具体描述如下：

**系统参数**

素数p：len比特素数（即 且p为素数）

椭圆曲线*Ep(a, b)*，及阶为素数*n*的点*P*。

**密钥的生成**

用户随机选取整数*d*，1<*d*<*n*，计算*Q* = *dP*。则公钥为点*Q*，私钥为整数*d*。

**加密（明文无嵌入，随机化加密）**

对明文



随机选取整数*k* (*1<k<n*)，计算倍点*kQ* = (*x, y*), 计算



得到密文*c=*(*C0, c1, c2*) 。

**解密**

对于密文*c=*(*C0, c1, c2*)，先计算*dC0* = (*x, y*)，再计算



得到解密后的明文为 *m=*(*m1, m2*)。

2、编写程序，实现ECC密码并输出结果，编程的关键是实现点加法运算。

3、编写主函数，主要步骤有

生成并输出素数p和椭圆曲线*Ep(a, b)*，及阶为素数*n*的点*P*；（此步可选择完成）

选取私钥*d*；

输出公钥*Q*；

随机生成并输出明文；（或输入明文）

加密并输出密文；

解密并输出解密后的明文。

参考测试用例：例如，系统参数选为素数p = 7，椭圆曲线为*E*：y2 = x3 + 3，椭圆曲线加法群为*E7*(0, 3)（循环群），生成元取其中点*P* = (1, 2)，*P*的阶为13。

4、程序代码：

#include **<**stdio**.**h**>**#include **<**stdlib**.**h**>**   
#include **<**time**.**h**>  
typedef** **struct** **{**    **int** x**;**    **int** y**;  
}** Point**;  
  
int** mod\_inverse**(int** a**,** **int** m**)** **{**    **for** **(int** i **=** 1**;** i **<** m**;** i**++)** **{**        **if** **((**a **\*** i**)** **%** m **==** 1**)** **{**            **return** i**;**        **}**    **}**    **return** 0**;  
}  
  
int** get\_gcd**(int** x**,** **int** y**)** **{**    **if** **(**y **==** 0**)** **{**        **return** x**;**    **}** **else** **{**        **return** get\_gcd**(**y**,** x **%** y**);**    **}  
}**Point point\_addition**(int** a**,** **int** b**,** **int** p**,** Point Q**,** Point P**)** **{**    Point R**;**    **int** x1 **=** Q**.**x**,** y1 **=** Q**.**y**,** x2 **=** P**.**x**,** y2 **=** P**.**y**;**    **int** member**,** denominator**,** gcd\_value**,** inverse\_value**,** k**;**      
    **if** **(**x1 **==** 0 **&&** y1 **==** 0**)** **{**        **return** P**;**    **}**    **if** **(**x2 **==** 0 **&&** y2 **==** 0**)** **{**        **return** Q**;**    **}**      
      
    **if** **(**x1 **==** x2 **&&** y1 **==** y2**)** **{**        member **=** 3 **\*** **(**x1 **\*** x1**)** **+** a**;**        denominator **=** 2 **\*** y1**;**    **}** **else** **{**        member **=** y2 **-** y1**;**        denominator **=** x2 **-** x1**;**          
        **if** **(**member **\*** denominator **<** 0**)** **{**            member **=** abs**(**member**);**            denominator **=** abs**(**denominator**);**            member**=**denominator**\***p**-**member**;**        **}**    **}**      
    gcd\_value **=** get\_gcd**(**member**,** denominator**);**    member **=** member **/** gcd\_value**;**    denominator **=** denominator **/** gcd\_value**;**      
    inverse\_value **=** mod\_inverse**(**denominator**,** p**);**    k **=** **(**member **\*** inverse\_value**)** **%** p**;**      
    R**.**x **=** **(**k **\*** k **-** x1 **-** x2 **+** p**)** **%** p**;**    R**.**y **=** **(**k **\*** **(**x1 **-** R**.**x**)** **-** y1 **+** p**)** **%** p**;**      
    **return** R**;  
}**Point scalar\_multiply**(int** d**,** **int** a**,** **int** b**,** **int** p**,** Point P**)** **{**    Point Q **=** **{**0**,** 0**};**    Point temp **=** P**;**      
    **while** **(**d **!=** 1**)** **{**        Q **=** point\_addition**(**a**,** b**,** p**,** temp**,** P**);**      
        temp **=** Q**;**        d **-=** 1**;**    **}**      
    **if(**Q**.**x**<**0**){**        Q**.**x**=(**Q**.**x**+**p**)%**p**;**    **}**    **if(**Q**.**y**<**0**){**        Q**.**y**=(**Q**.**y**+**p**)%**p**;**    **}**      
    **return** Q**;  
}  
  
int** main**()** **{**    *// 系统参数*    **int** p **=** 7**;**    **int** a **=** 0**;**    **int** b **=** 3**;**    Point G **=** **{**1**,** 2**};**    **int** n **=** 13**;**      
    *// 生成私钥 d*    **int** private\_key\_d **=** 4**;**      
    *// 生成公钥 Q*    Point public\_key\_Q **=** scalar\_multiply**(**private\_key\_d**,** a**,** b**,** p**,** G**);**    *//printf("%d %d",public\_key\_Q.x,public\_key\_Q.y);*    *// 明文*    Point plaintext **=** **{**3**,** 4**};**      
    *// 加密*    srand**(**time**(**0**));**    **int** k **=** rand**()%**7**+**1**;**    Point C0 **=** scalar\_multiply**(**k**,** a**,** b**,** p**,** G**);**    *//printf("C0=(%d %d)\n",C0.x,C0.y);*    Point KP**=**scalar\_multiply**(**k**,**a**,**b**,**p**,**public\_key\_Q**);**    *//printf("KP=%d %d\n",KP.x,KP.y);*      
    Point ciphertext**;**    ciphertext**.**x **=** **(**KP**.**x**\***plaintext**.**x**)%**p**;**    ciphertext**.**y **=** **(**plaintext**.**y **\*** KP**.**y**)** **%** p**;**    *//printf("%d %d\n",ciphertext.x,ciphertext.y);*      
    *// 解密*    Point dC0 **=** scalar\_multiply**(**private\_key\_d**,** a**,** b**,** p**,** C0**);**    *//Point dC0 = {1,5};*    *//printf("dC0=(%d %d)\n",dC0.x,dC0.y);*    Point decrypted\_text**;**    *//decrypted\_text.x = ciphertext.x;*      
    *//printf("%d %d \n",dC0.x,mod\_inverse(dC0.x,p));*    decrypted\_text**.**x **=** **(**ciphertext**.**x **\*** mod\_inverse**(**dC0**.**x**,**p**))** **%** p**;**    **if** **(**decrypted\_text**.**x **<** 0**)** **{**        decrypted\_text**.**x **+=** p**;**    **}**      
    decrypted\_text**.**y **=** **(**ciphertext**.**y**\***mod\_inverse**(**dC0**.**y**,**p**))** **%** p**;**    **if** **(**decrypted\_text**.**y **<** 0**)** **{**        decrypted\_text**.**y **+=** p**;**    **}**      
    *// 输出结果*    printf**("素数域 (p): %d**\n**",** p**);**    printf**("曲线参数 (a, b): %d, %d**\n**",** a**,** b**);**    printf**("基点 (P): (%d, %d)**\n**",** G**.**x**,** G**.**y**);**    printf**("基点阶数 (n): %d**\n**",** n**);**    printf**("私钥 (d): %d**\n**",** private\_key\_d**);**    printf**("公钥 (Q): (%d, %d)**\n**",** public\_key\_Q**.**x**,** public\_key\_Q**.**y**);**    printf**("明文: (%d, %d)**\n**",** plaintext**.**x**,** plaintext**.**y**);**    printf**("随机数：%d**\n**",**k**);**    printf**("密文: ( (%d,%d), %d, %d )**\n**",**C0**.**x**,**C0**.**y**,** ciphertext**.**x**,** ciphertext**.**y**);**    printf**("解密后的文本: (%d, %d)**\n**",** decrypted\_text**.**x**,** decrypted\_text**.**y**);**      
    **return** 0**;  
}**

四、实验结果与分析

测试结果截图与分析如下：

文本

描述已自动生成

文本

中度可信度描述已自动生成

设定好椭圆的基本参数a，b，p，基点，然后计算出私钥，公钥，最后进行加密解密操作。

图片包含 文本

描述已自动生成

经过老师的提示，将随机数k也输出出来

五、基于OBE模式的学生自我评价与体会

（该实验对自身分析、设计、思辨、创新等个人综合能力与素质的影响）

本次实验是用C语言实现ecc算法，算法中最有特点的就是用随机数k进行随机加密，所以在回显中也要输出一下随机数k。既然是随机数，就需要随机生成，可以利用rand（）函数实现伪随机。本次实验还是有着一定难度的。我就在实验过程中遇到了程序中的倍点运算和点加运算的结果不一致，最后发现是λ的运算过程中可能会出现负数的情况，进行修改之后程序终于能正常输出内容了。本次实验不止锻炼了我的C语言能力，还让我对ecc机密算法有了更深刻的认识。