西安电子科技大学

物联网安全实验课程 实验报告

实验名称 ECC 加密算法

物联网工程 1803041　班

姓名　魏红旭　学号 18030400014

同作者

实验日期　2021 年 5 月 25 日

成　绩

指导教师评语： 指导教师： 年月日
实验报告内容基本要求及参考格式 一、实验目的 二、实验所用仪器（或实验环境） 三、实验基本原理及步骤（或方案设计及理论计算） 四、实验数据记录（或仿真及软件设计） 五、实验结果分析及回答问题（或测试环境及测试结果）

一、实验目的：

编程实现 ECC，深入理解物联网数据安全中 ECC 加密解密原理

二、实验所用仪器（或实验环境）

计算机科学与技术学院实验中心，可接入 Internet 网台式机 44 台。

三、实验基本原理及要求

实验原理:

第一步：发送方选定一条 ECC $E_P(a,b)$，并取其上一点作为基点 G；

第二步：发送方选择一个私有密钥 k，并生成公开密钥 K=kG；

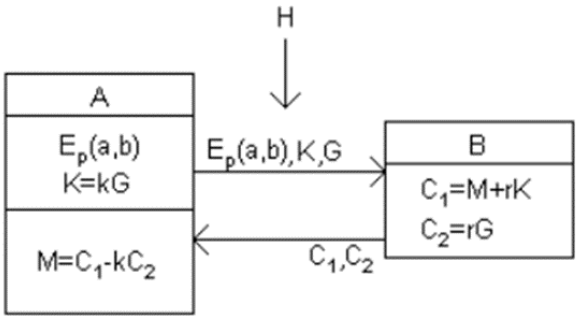
第三步：发送方将 Ep(a,b)和点 K，G 传给接收方；

第四步：接收方接收到信息后，将明文编码到 Ep(a,b)上一点 M，并产生一个随机整数 r（r<n，n 为 G 的阶数）；

第五步：接收方计算点 C1=M+rK 和 C2=rG，并将 C1、C2 传给发送方（容易理解，将加密信息传回给接收方，由接收方解密以知道发送方给他传的是啥）；

第六步：发送方收到信息后，计算 C1-kC2 得到的结果（解密过程）。

整个过程如图:



实验要求：

给出实验中的参数、过程以及结果。

四、实验步骤及实验数据记录：（要有文字描述和必要截图）

- **ECC 说明：**

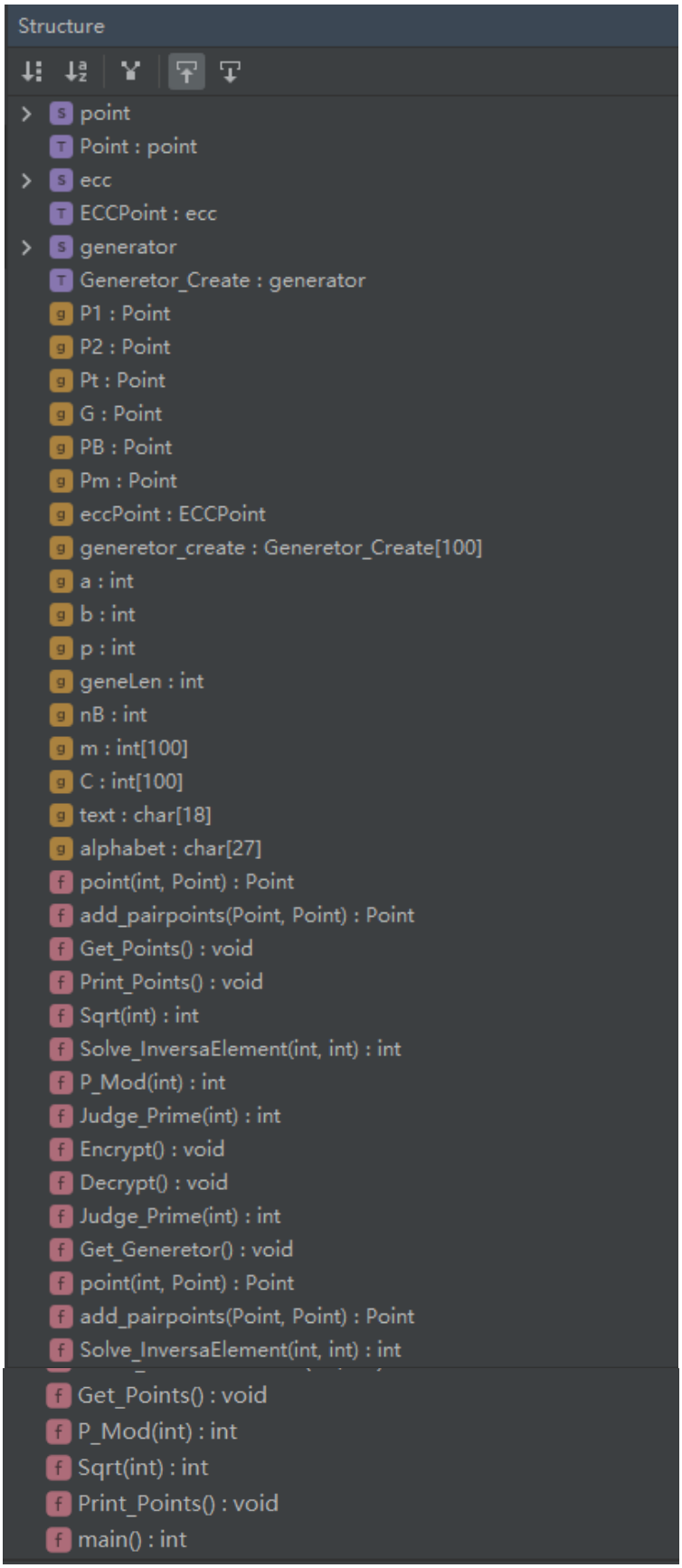
ECC 与 RSA 一样，也属于公开密钥算法。ECC 算法中存在着三个关键的变量：

(1) 基点：基点 G，G 为椭圆曲线 Ep(a,b)上的点；

(2) 阶：如果椭圆曲线上一点 P，存在最小的正整数 n 使得数乘 nP = O ∞ ,则将 n 称为 P 的阶，若 n 不存在，则 P 是无限阶的；

(3) 公钥 K：则给定私钥 k 和基点 G，根据加法法则，计算 K 很容易但反过来，给定 K 和 G，求 k 就非常困难。因为实际使用中的 ECC 原则上把 p 取得相当大，n 也相当大，要把 n 个解点逐一算出来列成上表是不可能的；

- 算法代码结构：


Structure
point
Point : point
ecc
ECCPoint : ecc
generator
Generetor_Create : generator
P1 : Point
P2 : Point
Pt : Point
G : Point
PB : Point
Pm : Point
eccPoint : ECCPoint
generetor_create : Generetor_Create[100]
a : int
b : int
p : int
geneLen : int
nB : int
m : int[100]
C : int[100]
text : char[18]
alphabet : char[27]
point(int, Point) : Point
add_pairpoints(Point, Point) : Point
Get_Points() : void
Print_Points() : void
Sqrt(int) : int
Solve_InversaElement(int, int) : int
P_Mod(int) : int
Judge_Prime(int) : int
Encrypt() : void
Decrypt() : void
Judge_Prime(int) : int
Get_Generetor() : void
point(int, Point) : Point
add_pairpoints(Point, Point) : Point
Solve_InversaElement(int, int) : int
Get_Points() : void
P_Mod(int) : int
Sqrt(int) : int
Print_Points() : void
main() : int

- **首先程序的整体的过程为：**

1) 编程计算该椭圆曲线上所有在有限域 GF 上的点；
2) 编程实现椭圆曲线上任意一个点 P(例如 P=(12,5))的倍点运算的递归算法，即计算 k*P(k=2,3,…)；
3) 利用此递归算法找出椭圆曲线上的所有生成元 G 以及它们的阶 n，即满足 n*G=O；
4) 设计实现某一用户 B 的公钥、私钥算法，即得到 public key=(n, G, PB, Ep(a, b)) secure key=nB(小于 n)
5) 假如用户 A 发送明文消息“xidian university”并加密传输给用户 B，用户 B 接收消息后要能解密为明文。

- **关于离散对数的问题：**

离散对数问题，我们熟悉的 RSA 算法，是基于大数的质因数分解，即对两个质数相乘容易，而将其合数分解很难的这个特点进行加密。而 ECC 算法是在有限域 Fp 定义公式：Q=kP，已知大数 k 和点 P 的情况下，很容易求点 Q，但是已知的点 P、点 Q，却很难求得 k，这就是经典的离散对数问题，ECC 算法正是利用该特点进行加密，点 Q 为公钥，大数 k 为私钥，点 P 为基点，和 RSA 最大的实际区别主要是密钥长度。

下面对程序中对部分函数的实现进行说明：

- **判断是否为素数：**

```
//判断是否为素数
int Judge_Prime(int n) {
    int i, k;
    k = sqrt(n);
    for (i = 2; i <= k; i++)
        if (n % i == 0) break;
    if (i <= k) return -1;
    else return 0;
}
```

- **取模函数：**

```
//取模函数
int P_Mod(int s) {
    int i,result;    //i保存s/p的倍数,result保存模运算的结果
    i = s / p;
    result = s - i * p;
    if (result >= 0) return result;
    else return result + p;
}
```

- **判断平方根是否为整数**

```
//判断平方根是否为整数
int Sqrt(int s) {
    int temp;
    temp = (int) sqrt(s);//转为整型
    if (temp * temp == s) return temp;
    else return -1;
}
```

- **求 b 关于 n 的逆元**

```
//求b关于n的逆元
int Solve_InversaElement(int n, int b) {
    int q, r, r1 = n, r2 = b, t, t1 = 0, t2 = 1, i = 1;
    while (r2 > 0) {
        q = r1 / r2;
        r = r1 % r2;
        r1 = r2;
        r2 = r;
        t = t1 - q * t2;
        t1 = t2;
        t2 = t;
    }
    if (t1 >= 0)
        return t1 % n;
    else {
        while ((t1 + i * n) < 0)
            i++;
        return t1 + i * n;
    }
}
```

- 两点的加法运算：

```
//两点的加法运算
Point add_pairpoints(Point p1, Point p2) {
    long t;
    int x1 = p1.point_x, y1 = p1.point_y;
    int x2 = p2.point_x, y2 = p2.point_y;
    int tx, ty, x3, y3, flag = 0;
    //求
    if ((x2 == x1) && (y2 == y1)) {
        //相同点相加
        if (y1 == 0) flag = 1;
        else t = (3 * x1 * x1 + a) * Solve_InversaElement(p, b: 2 * y1) % p;
    } else {
        //不同点相加
        ty = y2 - y1;
        tx = x2 - x1;
        while (ty < 0)
            ty += p;
        while (tx < 0)
            tx += p;
```

- t 倍点运算的递归算法

```
//t倍点运算的递归算法
Point point(int k, Point p0) {
    if (k == 1) return p0;
    else if (k == 2) return add_pairpoints(p0, p0);
    else return add_pairpoints(p0, p2: point( k: k - 1, p0));
}
```

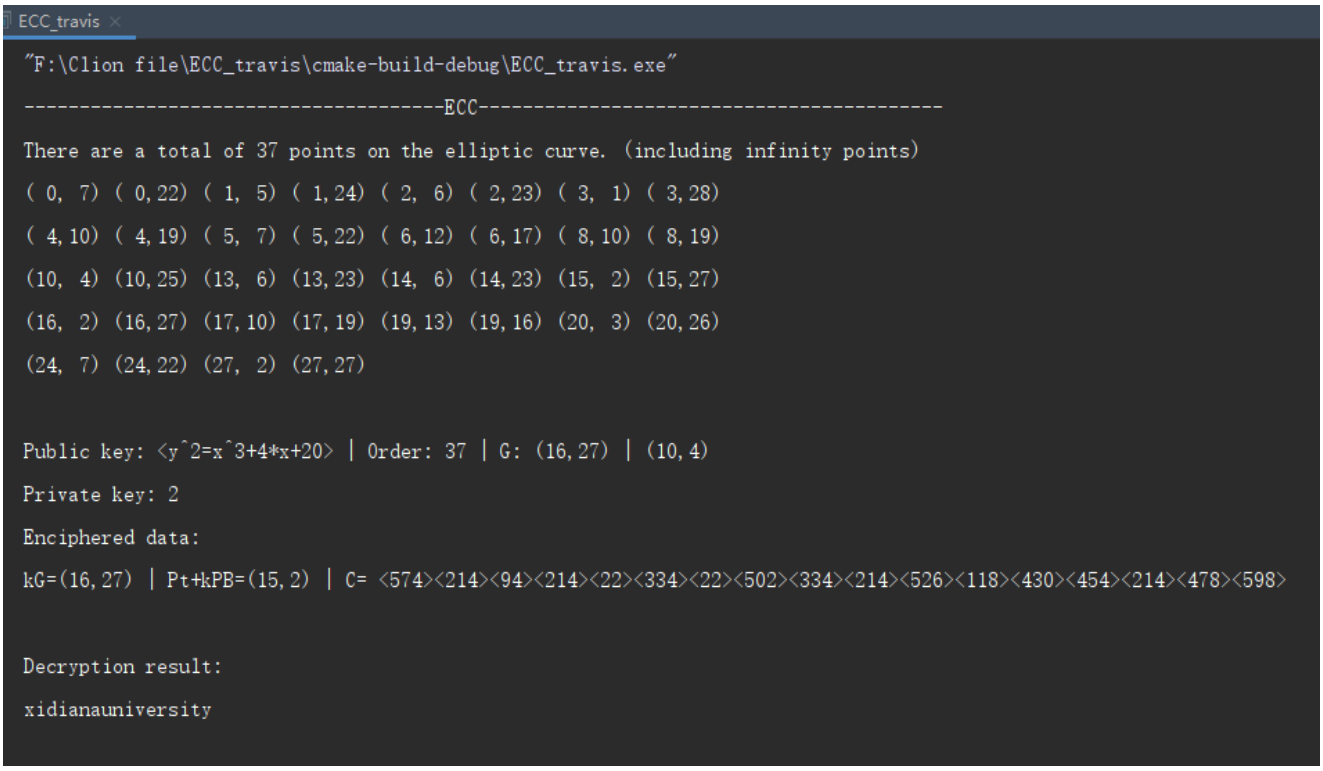
- 求生成元以及阶

```
//求生成元以及阶
void Get_Generetor() {
    int i, j = 0;
    int count = 1;
    Point p1, p2;
    Get_Points();
    for (i = 0; i < eccPoint.len; i++) {
        count = 1;
        p1.point_x = p2.point_x = eccPoint.p[i].point_x;
        p1.point_y = p2.point_y = eccPoint.p[i].point_y;
        while (1) {
            p2 = add_pairpoints(p1, p2);
            if (p2.point_x == -1 && p2.point_y == -1)
                break;
            count++;
            if (p2.point_x == p1.point_x) {
                break;
            }
        }
```

- 解密：

```
//解密
void Decrypt() {
    Point temp, temp1;
    int m, i;
    temp = point(nB, P1);
    temp.point_y = 0 - temp.point_y;
    temp1 = add_pairpoints(Pm, temp);//求解Pt
    printf( _Format: "\nDecryption result:\n");
    for (i = 0; i < strlen(text); i++) {
        m = (C[i] - temp1.point_y) / temp1.point_x;
        printf( _Format: "%c", alphabet[m]);//输出密文
    }
    printf( _Format: "\n");
}
```

五、实验结果分析及实验总结与体会



在进行测试时选择的 a：4，b：20，p：29，生成的 G 为(16，27)，其他结果如上图的输出所示；

输出的生成元和阶如下图所示：

(0,7)-[37]	(0,22)-[37]	(1,5)-[37]	(1,24)-[37]	(2,6)-[37]	(2,23)-[37]
(3,1)-[37]	(3,28)-[37]	(4,10)-[37]	(4,19)-[37]	(5,7)-[37]	(5,22)-[37]
(6,12)-[37]	(6,17)-[37]	(8,10)-[37]	(8,19)-[37]	(10,4)-[37]	(10,25)-[37]
(13,6)-[37]	(13,23)-[37]	(14,6)-[37]	(14,23)-[37]	(15,2)-[37]	(15,27)-[37]
(16,2)-[37]	(16,27)-[37]	(17,10)-[37]	(17,19)-[37]	(19,13)-[37]	(19,16)-[37]
(20,3)-[37]	(20,26)-[37]	(24,7)-[37]	(24,22)-[37]	(27,2)-[37]	(27,27)-[37]

- 实验总结：

通过此次实验，我对于 ECC 算法的原理以及整个实现过程有了较为深刻的了解，实际上，整个 ECC 算法的实现过程还是比较负责的，整个代码的实现过程重点的包括有限域 GF 上点的计算、找出椭圆曲线上的所有生成

元 G 以及它们的阶 n、以及加解密过程，通过对算法的实现，也真正理解了其在保密通信中的实际作用。

六、源代码

```
#include <stdio.h>
#include<stdlib.h>
#include<string.h>
#include<math.h>
#include<time.h>
typedef struct point {
    int point_x;
    int point_y;
} Point;
typedef struct ecc {
    struct point p[100];
    int len;
} ECCPoint;
typedef struct generator {
    Point p;
    int p_class;
} Generetor_Create;

Point P1, P2, Pt, G, PB, Pm;
ECCPoint eccPoint;
Generetor_Create generetor_create[100];

int a = 4, b = 20, p = 29, geneLen, nB;   //nB 为私钥
int m[100], C[100];
char text[] = "xidian university";
char alphabet[27] = "abcdefghijklmnopqrstuvwxyz";

Point point(int k, Point p);
Point add_pairpoints(Point p1, Point p2);
void Get_Points();
void Print_Points();
int Sqrt(int s);
int Solve_InversaElement(int n, int b);
int P_Mod(int s);
int Judge_Prime(int n);

//加密
void Encrypt() {
```

```
    int num, i, j, gene_class, num_t, k;
    srand(time(NULL));
    //明文转换过程
    for (i = 0; i < strlen(text); i++) {
        for (j = 0; j < 26; j++) {
            if (text[i] == alphabet[j]) {
                m[i] = j;//将字符串明文换成数字，并存到整型数组 m 里面
            }
        }
    }
    //选择生成元
    num = rand() % geneLen;
    gene_class = generetor_create[num].p_class;
    while (Judge_Prime(gene_class) == -1) {//不是素数
        num = rand() % (geneLen - 3) + 3;
        gene_class = generetor_create[num].p_class;
    }
    G = generetor_create[num].p;
    nB = rand() % (gene_class - 1) + 1;//选择私钥
    PB = point(nB, G);
    printf("\nPublic key: ");
    printf("<y^2=x^3+%d*x+%d> | Order: %d | G: (%d,%d) | (%d,%d)\n", a, b, g
ene_class, G.point_x, G.point_y, PB.point_x, PB.point_y);
    printf("Private key: %d\n",nB);
    //加密
    k = rand() % (gene_class - 2) + 1;
    P1 = point(k, G);
    num_t = rand() % eccPoint.len; //选择映射点
    Pt = eccPoint.p[num_t];
    P2 = point(k, PB);
    Pm = add_pairpoints(Pt, P2);
    printf("Enciphered data:\n");
    printf("kG=(%d,%d) | Pt+kPB=(%d,%d) | C= ", P1.point_x, P1.point_y, Pm.p
oint_x, Pm.point_y);
    for (i = 0; i < strlen(text); i++) {
        C[i] = m[i] * Pt.point_x + Pt.point_y;
        printf("<%d>", C[i]);
    }
    printf(" \n");
}

//解密
void Decrypt() {
    Point temp, temp1;
```

```
    int m, i;
    temp = point(nB, P1);
    temp.point_y = 0 - temp.point_y;
    temp1 = add_pairpoints(Pm, temp);//求解 Pt
    printf("\nDecryption result:\n");
    for (i = 0; i < strlen(text); i++) {
        m = (C[i] - temp1.point_y) / temp1.point_x;
        printf("%c", alphabet[m]);//输出密文
    }
    printf("\n");
}

//判断是否为素数
int Judge_Prime(int n) {
    int i, k;
    k = sqrt(n);
    for (i = 2; i <= k; i++)
        if (n % i == 0) break;
    if (i <= k) return -1;
     else return 0;
}

//求生成元以及阶
void Get_Generetor() {
    int i, j = 0;
    int count = 1;
    Point p1, p2;
    Get_Points();
    printf("\n\n");
    for (i = 0; i < eccPoint.len; i++) {
        count = 1;
        p1.point_x = p2.point_x = eccPoint.p[i].point_x;
        p1.point_y = p2.point_y = eccPoint.p[i].point_y;
        while (1) {
            p2 = add_pairpoints(p1, p2);
            if (p2.point_x == -1 && p2.point_y == -1)
                break;
            count++;
            if (p2.point_x == p1.point_x) {
                break;
            }
        }
        count++;
        if (count <= eccPoint.len + 1) {
```

```
125.             generetor_create[j].p.point_x = p1.point_x;
126.             generetor_create[j].p.point_y = p1.point_y;
127.             generetor_create[j].p_class = count;
128.             printf("(%d,%d)-
    [%d]\t",generetor_create[j].p.point_x,generetor_create[j].p.point_y,genereto
    r_create[j].p_class);
129.             j++;
130.             if(j % 6 == 0) printf("\n");
131.         }
132.         geneLen = j;
133.     }
134. }
135.
136. //t 倍点运算的递归算法
137. Point point(int k, Point p0) {
138.     if (k == 1) return p0;
139.     else if (k == 2) return add_pairpoints(p0, p0);
140.     else return add_pairpoints(p0, point(k - 1, p0));
141. }
142.
143. //两点的加法运算
144. Point add_pairpoints(Point p1, Point p2) {
145.     long t;
146.     int x1 = p1.point_x, y1 = p1.point_y;
147.     int x2 = p2.point_x, y2 = p2.point_y;
148.     int tx, ty, x3, y3, flag = 0;
149.     //求
150.     if ((x2 == x1) && (y2 == y1)) {
151.         //相同点相加
152.         if (y1 == 0) flag = 1;
153.         else t = (3 * x1 * x1 + a) * Solve_InversaElement(p, 2 * y1) % p;
154.     } else {
155.         //不同点相加
156.         ty = y2 - y1;
157.         tx = x2 - x1;
158.         while (ty < 0)
159.             ty += p;
160.         while (tx < 0)
161.             tx += p;
162.         if (tx == 0 && ty != 0) flag = 1;
163.         else t = ty * Solve_InversaElement(p, tx) % p;
164.     }
165.     if (flag == 1) {
166.         p2.point_x = -1;
```

```
167.         p2.point_y = -1;
168.     } else {
169.         x3 = (t * t - x1 - x2) % p;
170.         y3 = (t * (x1 - x3) - y1) % p;
171.         //使结果在有限域 GF(P)上
172.         while (x3 < 0)
173.             x3 += p;
174.         while (y3 < 0)
175.             y3 += p;
176.         p2.point_x = x3;
177.         p2.point_y = y3;
178.     }
179.     return p2;
180. }
181.
182. //求 b 关于 n 的逆元
183. int Solve_InversaElement(int n, int b) {
184.     int q, r, r1 = n, r2 = b, t, t1 = 0, t2 = 1, i = 1;
185.     while (r2 > 0) {
186.         q = r1 / r2;
187.         r = r1 % r2;
188.         r1 = r2;
189.         r2 = r;
190.         t = t1 - q * t2;
191.         t1 = t2;
192.         t2 = t;
193.     }
194.     if (t1 >= 0)
195.         return t1 % n;
196.     else {
197.         while ((t1 + i * n) < 0)
198.             i++;
199.         return t1 + i * n;
200.     }
201. }
202.
203. //求出椭圆曲线上所有点
204. void Get_Points() {
205.     int i = 0, j = 0, s, y = 0, n = 0, q = 0;
206.     int modsqrt = 0, flag = 0;
207.     if (4 * a * a * a + 27 * b * b != 0) {
208.         for (i = 0; i <= p - 1; i++) {
209.             flag = 0;
210.             n = 1;
```

```
            y = 0;
            s = i * i * i + a * i + b;
            while (s < 0) {
                s += p;
            }
            s = P_Mod(s);
            modsqrt = Sqrt(s);
            if (modsqrt != -1) {
                flag = 1;
                y = modsqrt;
            } else {
                while (n <= p - 1) {
                    q = s + n * p;
                    modsqrt = Sqrt(q);
                    if (modsqrt != -1) {
                        y = modsqrt;
                        flag = 1;
                        break;
                    }
                    flag = 0;
                    n++;
                }
            }
            if (flag == 1) {
                eccPoint.p[j].point_x = i;
                eccPoint.p[j].point_y = y;
                j++;
                if (y != 0) {
                    eccPoint.p[j].point_x = i;
                    eccPoint.p[j].point_y = (p - y) % p;
                    j++;
                }
            }
        }
        eccPoint.len = j;//点集个数
        Print_Points(); //打印点集
    }
}

//取模函数
int P_Mod(int s) {
    int i,result;    //i 保存 s/p 的倍数,result 保存模运算的结果
    i = s / p;
    result = s - i * p;
```

```
255.     if (result >= 0) return result;
256.     else return result + p;
257. }
258. 
259. //判断平方根是否为整数
260. int Sqrt(int s) {
261.     int temp;
262.     temp = (int) sqrt(s);//转为整型
263.     if (temp * temp == s) return temp;
264.     else return -1;
265. }
266. 
267. //打印点集
268. void Print_Points() {
269.     int i, len = eccPoint.len;
270.     printf("---------------------------------------ECC-----------------------
    --------------");
271.     printf("\nThere are a total of %d points on the elliptic curve. (includ
    ing infinity points)", len + 1);
272.     for (i = 0; i < len; i++) {
273.         if (i % 8 == 0) {
274.             printf("\n");
275.         }
276.         printf("(%2d,%2d)\t", eccPoint.p[i].point_x, eccPoint.p[i].point_y)
    ;
277.     }
278.     printf("\n");
279. }
```