

## 《信息安全基础综合实验》课程实验报告

实验题目：SM2 椭圆曲线公钥密码算法

班级：1718039      学号：17180210027      姓名：李欣

### 一、实验目的

了解并实现 SM2 椭圆曲线公钥密码算法。

### 二、方案设计

#### 2.1 背景

N.Koblitz 和 V.Miller 在 1985 年各自独立地提出将椭圆曲线应用于公钥密码系统。椭圆曲线公钥密码所基于的曲线性质如下：

- 有限域上椭圆曲线在点加运算下构成有限交换群，且其阶与基域规模相近；
- 类似于有限域乘法群中的乘幂运算，椭圆曲线多倍点运算构成一个单向函数。

#### 2.2 椭圆曲线

椭圆曲线并非椭圆，之所以称为椭圆曲线是因为它的曲线方程与计算椭圆周长的方程类似。

椭圆曲线的方程：

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

其中  $a, b, c, d, e$  是满足某些简单条件的实数。

椭圆曲线有一个特殊的点，记为  $O$ ，它并不在椭圆曲线  $E$  上，此点称为无穷远点。

一条椭圆曲线是由全体解再加上一个无穷远点构成的集合（椭圆曲线上的点是有有限个）

$$E=\{(x, y) \mid Y^2+aXY+bY=X^3+cX^2+dX+e\} \cup \{O\}$$

#### 2.3 素数域 $F_p$

当  $p$  是奇素数时，素域  $F_p$  中的元素用整数  $0, 1, 2, \dots, p-1$  表示。

- a) 加法单位元是整数 0；
- b) 乘法单位元是整数 1；
- c) 域元素的加法是整数的模  $p$  加法，即若  $a, b \in F_p$ ，则

$$a+b=(a+b) \bmod p;$$

- d) 域元素的乘法是整数的模  $p$  乘法，即若  $a, b \in F_p$ ，则

$$a \cdot b=(a \cdot b) \bmod p。$$

#### 2.4 $F_p$ 上的椭圆曲线

定义在 $F_p$  ( $p$ 是大于3的素数)上的椭圆曲线方程为：

$$y^2 = x^3 + ax + b, \quad a, b \in F_p, \quad \text{且} (4a^3 + 27b^2) \bmod p \neq 0 \quad (1)$$

椭圆曲线 $E(F_p)$ 定义为：

$$E(F_p) = \{(x, y) | x, y \in F_p, \text{且满足方程(1)}\} \cup \{O\}$$

其中 $O$ 是无穷远点。

椭圆曲线 $E(F_p)$ 上的点的数目用 $\#E(F_p)$ 表示，称为椭圆曲线 $E(F_p)$ 的阶。

椭圆曲线 $E(F_p)$ 上的点按照下面的加法运算规则，构成一个交换群：

a)  $O + O = O$  ;

b)  $\forall P = (x, y) \in E(F_p) \setminus \{O\}, P + O = O + P = P$  ;

c)  $\forall P = (x, y) \in E(F_p) \setminus \{O\}, P$ 的逆元素 $-P = (x, -y), P + (-P) = O$  ;

d) 两个非互逆的不同点相加的规则：

设 $P_1 = (x_1, y_1) \in E(F_p) \setminus \{O\}, P_2 = (x_2, y_2) \in E(F_p) \setminus \{O\},$

且 $x_1 \neq x_2$ ，设 $P_3 = (x_3, y_3) = P_1 + P_2$ ，则

$$\begin{cases} x_3 = \lambda^2 - x_1 - x_2 \\ y_3 = \lambda(x_1 - x_3) - y_1 \end{cases}$$

其中 $\lambda = \frac{y_2 - y_1}{x_2 - x_1}$

e) 倍点规则：

设 $P_1 = (x_1, y_1) \in E(F_p) \setminus \{O\}$ ，且 $y_1 \neq 0$ ，

设 $P_3 = (x_3, y_3) = P_1 + P_2$ ，则

$$\begin{cases} x_3 = \lambda^2 - 2x_1 \\ y_3 = \lambda(x_1 - x_3) - y_1 \end{cases}$$

其中

$$\lambda = \frac{3x_1^2 + a}{2y_1}$$

## 2.5 二元扩域

当 $q$ 是2的方幂 $2^m$ 时，二元扩域 $F_{2^m}$ 可以看成 $F_2$ 上的 $m$ 维向量空间，其元素可用长度为 $m$ 的比特串表示。

$F_{2^m}$ 中的任意一个元素 $a(x) = a_{m-1}x^{m-1} + a_{m-2}x^{m-2} + \cdots + a_1x + a_0$ 在 $F_2$ 上的系数恰好构成了长度为 $m$ 的比特串，用 $a = (a_{m-1}, a_{m-2}, \cdots, a_1, a_0)$ 表示。

a) 零元0用全0比特串表示；

b) 乘法单位元1用比特串(00...001)表示；

c) 两个域元素的加法为比特串的按比特异或运算；

d) 域元素 $a$ 和 $b$ 的乘法定义如下：设 $a$ 和 $b$ 对应的 $F_2$ 上多项式为 $a(x)$ 和 $b(x)$ ，则 $a \cdot b$ 定义为多项式 $(a(x)b(x)) \bmod f(x)$ 对应的比特串。

## 2.6 加密过程

设需要发送的消息为比特串 $M$ ， $klen$ 为 $M$ 的比特长度。

为了对明文 $M$ 进行加密，作为加密者的用户A应实现以下运算步骤：

A1：用随机数发生器产生随机数 $k \in [1, n-1]$ ；

A2：计算椭圆曲线点 $C1 = [k]G = (x1, y1)$ ，按本文第1部分4.2.8和4.2.4给出的细节，将 $C1$ 的数据类型转换为比特串；

A3：计算椭圆曲线点 $S = [h]PB$ ，若 $S$ 是无穷远点，则报错并退出；

A4：计算椭圆曲线点 $[k]PB = (x2, y2)$ ，按本文第1部分4.2.5和4.2.4给出的细节，将坐标 $x2, y2$ 的数据类型转换为比特串；

A5：计算 $t = KDF(x2 // y2, klen)$ ，若 $t$ 为全0比特串，则返回A1；

A6：计算 $C2 = M \oplus t$ ；

A7：计算 $C3 = Hash(x2 // M // y2)$ ；

A8：输出密文 $C = C1 // C2 // C3$ 。

## 2.7 解密过程

设 $klen$ 为密文中 $C2$ 的比特长度。为了对密文 $C = C1 // C2 // C3$ 进行解密，作为解密者的用户B应实现以下运算步骤：

B1：从 $C$ 中取出比特串 $C1$ ，按本文第1部分4.2.3和4.2.9给出的细节，将 $C1$ 的数据类型转换为椭圆曲线上的点，验证 $C1$ 是否满足椭圆曲线方程，若不满足则报错并退出；

B2：计算椭圆曲线点 $S = [h]C1$ ，若 $S$ 是无穷远点，则报错并退出；

B3：计算 $[dB]C1 = (x2, y2)$ ，按本文第1部分4.2.5和4.2.4给出的细节，将坐标 $x2, y2$ 的数据类型转换为比特串；

B4：计算 $t = KDF(x2 // y2, klen)$ ，若 $t$ 为全0比特串，则报错并退出；

B5：从 $C$ 中取出比特串 $C2$ ，计算 $M' = C2 \oplus t$ ；

B6：计算 $u = Hash(x2 // M' // y2)$ ，从 $C$ 中取出比特串 $C3$ ，若 $u \neq C3$ ，则报错

并退出；  
B7：输出明文  $M'$  。

## 三、方案实现

### 3.1 参数初始化

定义 FPECC 结构体，并初始化 Ecc256 变量：

```
10 struct FPECC {
11     char *p;
12     char *a;
13     char *b;
14     char *n;
15     char *x;
16     char *y;
17 };
18 ///SM2初始化各种参数*/
19 struct FPECC Ecc256 = {
20     "FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF00000000FFFFFFFFFFFFFFFF", //p
21     "FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF00000000FFFFFFFFFFFFFFFFC", //a
22     "28E9FA9E9D9F5E344D5A9E4BCF6509A7F39789F515AB8F92DDBCBD414D940E93", //b
23     "FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF7203DF6B21C6052B53BBF40939D54123", //n
24     "32C4AE2C1F1981195F9904466A39C9948FE30BBFF2660BE1715A4589334C74C7", //Gx
25     "BC3736A2F4F6779C59BDCEE36B692153D0A9877CC62A474002DF32E52139F0A0", //Gy
26 };
```

### 3.2 公私钥对的生成

#### 3.2.1 变量初始化

```
30 void sm2_keygen(unsigned char *wx, int *wxlen, unsigned char *wy, int *wylen, unsigned char *privkey, int *privkeylen)
31 {
32     struct FPECC *cfig = &Ecc256;
33     epoint *g;
34     big a, b, p, n, x, y, key1;
35     miracl *mip = mirsys(20, 0);
36
37     mip->IOBASE = 16;
38
39     p = mirvar(0);
40     a = mirvar(0);
41     b = mirvar(0);
42     n = mirvar(0);
43     x = mirvar(0);
44     y = mirvar(0);
45
46     key1 = mirvar(0);
47     ///将给定数据赋值给对应变量
48     cinstr(p, cfig->p);
49     cinstr(a, cfig->a);
50     cinstr(b, cfig->b);
51     cinstr(n, cfig->n);
52     cinstr(x, cfig->x);
53     cinstr(y, cfig->y);
```

#### 3.2.2 初始化椭圆曲线和点

使用 ecurve\_init()、epoint\_init()、epoint\_set() 函数初始化椭圆曲线

```
54 ///初始化椭圆曲线
55 ecurve_init(a, b, p, MR_PROJECTIVE);
56 g = epoint_init();
57 epoint_set(x, y, 0, g);
58
```

### 3.2.3 初始化随机的点 $PB = dB \cdot G$

```
58
59 //初始化随机种子
60 irand(time(NULL) + SEED_CONST);
61 bigrand(n, key1); //随机数dB
62 ecurve_mult(key1, g, g); //倍点运算后的g点即为PB
63 epoint_get(g, x, y); //得到PB的x和y坐标
64
```

### 3.3 密钥派生函数 KDF

密钥派生函数的作用是从一个共享的秘密比特串中派生出密钥数据。在密钥协商过程中,密钥派生函数作用在密钥交换所获共享的秘密比特串上,从中产生所需的会话密钥或进一步加密所需的密钥数据。

密钥派生函数需要调用密码杂凑函数。

设密码杂凑函数为  $H_v()$ ,其输出是长度恰为  $v$  比特的杂凑值。

密钥派生函数  $KDF(Z, klen)$ :

输入:比特串  $Z$ ,整数  $klen$ (表示要获得的密钥数据的比特长度,要求该值小于  $(2^{32}-1)v$ )。

输出:长度为  $klen$  的密钥数据比特串  $K$ 。

a) 初始化一个 32 比特构成的计数器  $ct=0x00000001$ ;

b) 对  $i$  从 1 到  $\lceil klen/v \rceil$  执行:

b.1) 计算  $Ha_i = H_v(Z \parallel ct)$ ;

b.2)  $ct^{++}$ ;

c) 若  $klen/v$  是整数,令  $Ha_{\lceil klen/v \rceil} = Ha_{\lceil klen/v \rceil}$ ,

否则令  $Ha_{\lceil klen/v \rceil}$  为  $Ha_{\lceil klen/v \rceil}$  最左边的  $(klen - (v \times \lceil klen/v \rceil))$  比特;

d) 令  $K = Ha_1 \parallel Ha_2 \parallel \dots \parallel Ha_{\lceil klen/v \rceil - 1} \parallel Ha_{\lceil klen/v \rceil}$ 。

```
80 int kdf(unsigned char *z1, unsigned char *zr, int klen, unsigned char *kbuf)
81 {
82     unsigned char buf[70];
83     unsigned char digest[32];
84     unsigned int ct = 0x00000001;
85     int i, m, n;
86     unsigned char *p;
87     memcpy(buf, z1, 32);
88     memcpy(buf + 32, zr, 32);
89     m = klen / 32;
90     n = klen % 32;
91     p = kbuf;
92     for (i = 0; i < m; i++)
93     {
94         buf[64] = (ct >> 24) & 0xFF;
95         buf[65] = (ct >> 16) & 0xFF;
96         buf[66] = (ct >> 8) & 0xFF;
97         buf[67] = ct & 0xFF;
98         sm3(buf, 68, p);
99         p += 32;
100        ct++;
101    }
102    if (n != 0)
103    {
104        buf[64] = (ct >> 24) & 0xFF;
105        buf[65] = (ct >> 16) & 0xFF;
106        buf[66] = (ct >> 8) & 0xFF;
107        buf[67] = ct & 0xFF;
108        sm3(buf, 68, digest);
109    }
110    memcpy(p, digest, n);
111    for (i = 0; i < klen; i++)
112    {
113        if (kbuf[i] != 0)
114            break;
115    }
116    if (i < klen)
117        return 1;
118    else
119        return 0;
120}
```

### 3.4 SM2 加密



```

166 sm2_encrypt_again:
167     do
168     {
169         bigrand(n, k); //产生随机大数k
170     } while (k->len == 0);
171     ecurve_mult(k, g, g); //倍点运算得到C1
172     epoint_get(g, c1, c2);
173     big_to_bytes(32, c1, (char *)outmsg, TRUE); //将C1的坐标表示为字节码
174     big_to_bytes(32, c2, (char *)outmsg + 32, TRUE);
175     //判断S是否为无穷远点
176     if (point_at_infinity(w))
177         goto exit_sm2_encrypt;
178     //计算kPb并将其表示为字节码
179     ecurve_mult(k, w, w);
180     epoint_get(w, x2, y2);
181     big_to_bytes(32, x2, (char *)z1, TRUE);
182     big_to_bytes(32, y2, (char *)zr, TRUE);
183     //计算t, 利用密钥派生函数
184     if (kdf(z1, zr, msglen, outmsg + 64) == 0)
185         goto sm2_encrypt_again;
186     //将M与t逐字节进行异或得到C2
187     for (i = 0; i < msglen; i++)
188     {
189         outmsg[64 + i] ^= msg[i];
190     }
191     //将x2, M, y2连接, 并计算其哈希值
192     memcpy(tmp, z1, 32);
193     memcpy(tmp + 32, msg, msglen);
194     memcpy(tmp + 32 + msglen, zr, 32);
195     sm3(tmp, 64 + msglen, &outmsg[64 + msglen]);
196     //得到的密文总长度为96加明文长度
197     ret = msglen + 64 + 32;

```

### 3.5 SM2 解密

```

253     ecurve_init(a, b, p, MR_PROJECTIVE);
254     g = epoint_init();
255     bytes_to_big(32, (char *)msg, x);
256     bytes_to_big(32, (char *)msg + 32, y);
257     if (!epoint_set(x, y, 0, g))
258         goto exit_sm2_decrypt;
259     if (point_at_infinity(g))
260         goto exit_sm2_decrypt;
261     ecurve_mult(key1, g, g);
262     epoint_get(g, x2, y2);
263     big_to_bytes(32, x2, (char *)z1, TRUE);
264     big_to_bytes(32, y2, (char *)zr, TRUE);
265
266     if (kdf(z1, zr, msglen, outmsg) == 0)
267         goto exit_sm2_decrypt;
268     for (i = 0; i < msglen; i++)
269     {
270         outmsg[i] ^= msg[i + 64];
271     }
272     memcpy(tmp, z1, 32);
273     memcpy(tmp + 32, outmsg, msglen);
274     memcpy(tmp + 32 + msglen, zr, 32);
275     sm3(tmp, 64 + msglen, c3);
276     if (memcmp(c3, msg + 64 + msglen, 32) != 0)
277         goto exit_sm2_decrypt;
278     ret = msglen;

```

### 3.6 测试 SM2 加密解密流程

#### 3.6.1 文件读取

```
9      FILE * fp;
10     int i = 0;
11     fopen_s(&fp, "1.txt", "r");
12     fread_s(M, MAX, sizeof(char), MAX, fp);
13     printf("***** Plaintext *****\n\n");
14     printf("%s", M);
15
```

#### 3.6.2 生成私钥

```
24     sm2_keygen(xB, &wxlen, yB, &wylen, dB, &pklen);
25     printf("\n\n***** Privkey *****\n\n");
26     do {
27         printf("%02X", dB[i]);
28         i++;
29     } while (dB[i] != '\0' || dB[i + 1] != '\0');
```

#### 3.6.3 加密并输出加密后的密文

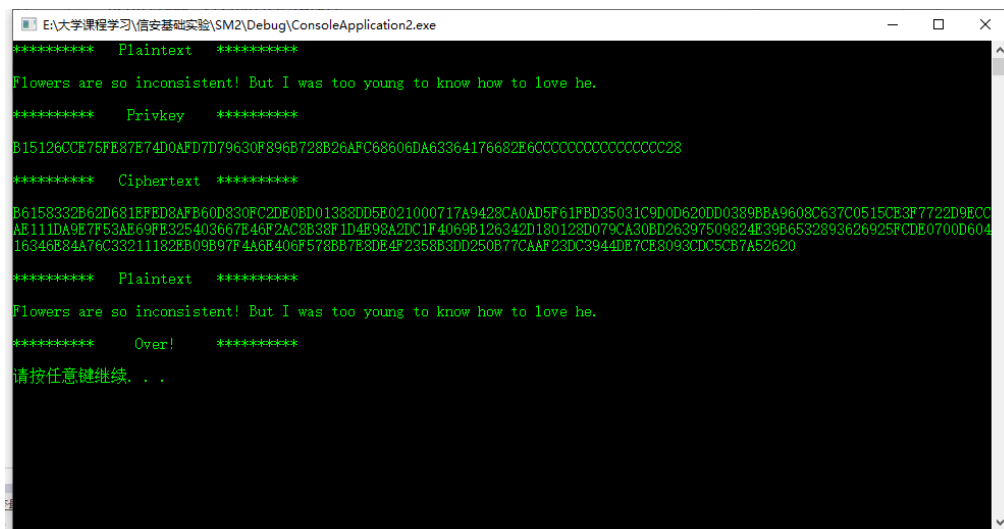
```
33     printf("\n\n***** Ciphertext *****\n\n");
34     sm2_encrypt(M, strlen((char*)M), xB, 32, yB, 32, MM);
35     //printf(MM, 64 + strlen((char*)M) + 32);
36     i = 0;
37     do {
38         printf("%02X", MM[i]);
39         i++;
40     } while (MM[i] != '\0' || MM[i + 1] != '\0' || MM[i+2] != 0);
```

#### 3.6.4 解密并输出明文

```
41     printf("\n\n***** Plaintext *****\n\n");
42     sm2_decrypt(MM, 64 + strlen((char*)M) + 32, dB, 32, MMM);
43     printf(MMM, strlen((char*)M));
```

## 四、数据分析

运行结果：



```
E:\大学课程学习\信息安全实验\SM2\Debug\ConsoleApplication2.exe
***** Plaintext *****
Flowers are so inconsistent! But I was too young to know how to love he.
***** Privkey *****
B15126CCE75FE87E74D0AFD7D79630F896B728B26AFC68606DA63364176682E6CCCCCCCCCCCCCCC28
***** Ciphertext *****
B6158332B62D681EFED8AFB60D830FC2DE0BD01388DD5E021000717A9428CA0AD5F61FBD35031C9D0D620DD0389BBA9608C637C0515CE3F7722D9ECC
AE111DA9E7F53AB69FE325403667E46F2ACBB38F1D4E98A2DC1F4069B126342D180128D079CA30BD26397509824E39B6532893626925FCDE0700D604
16346E84A76C33211182EB09B97F4A6E406F578BB7E8DE4F2358B3DD250B77CAAF23DC3944DE7CE8093CDC5CB7A52620
***** Plaintext *****
Flowers are so inconsistent! But I was too young to know how to love he.
***** Over! *****
请按任意键继续. . .
```

## 五、总结

通过本次实验了解了 SM2 公钥密码体系算法的加解密，中途也遇到了比如未作无穷远点判断等问题导致无法解密的情况，最后通过判断是否为无穷远点解决。SM3 算法的代码直接参考网上的代码，直接在编写的 SM2 加解密函数中调用即可。