西安电子科技大学网络与信息安全学院

信息安全基础与密码学
综合实验

实 验 报 告（五）

SM2 椭圆曲线公钥密码算法

班级：

姓名：

学号

日期：

一、实验目的

1．实验环境

- ✧ Windows11，Python3.10，Visual Studio Code。

2．实验目标

- ✧ 通过编程实现 SM2 椭圆曲线公钥密码算法，加深理解；
- ✧ 体会密码学与数论的紧密联系，将数论的知识运用于密码学的方案设计中；
- ✧ 提高逻辑思维能力与实践能力。

二、方案设计

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．原理

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：

$$y^2 + \mathrm{axy} + \mathrm{by} = x^3 + \mathrm{c}x^2 + \mathrm{dx} + \mathrm{e}$$

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$\boldsymbol{O}$，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（椭圆曲线上的是有限的）。椭圆曲线算法那公钥密码所基于的曲线性质：椭圆曲线多倍点运算构成一个单向函数。在多倍点运算中，已知多倍点与基点，求解倍数的问题称为椭圆曲线离散对数问题。对于一般椭圆曲线的离散对数问题，目前只存在指数级计算复杂度的求解方法。与大数分解问题及有限域上离散对数问题相比，椭圆曲线离散对数问题的求解难度要大得多。因此，在相同安全程度要求下，椭圆曲线密码较其他公钥密码所需的秘钥规模要小得多。

3．算法步骤

✧ 加密算法：

设需要发送的消息为比特串$\boldsymbol{M}$，klen 为$\boldsymbol{M}$的比特长度。为了对明文$\boldsymbol{M}$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$\boldsymbol{k}\in[\boldsymbol{1},\boldsymbol{n-1}]$；
- A2：计算椭圆曲线点$\boldsymbol{C1}=[\boldsymbol{k}]\boldsymbol{G}=(\boldsymbol{x1},\boldsymbol{y1})$，按本文本第 1 部分 4.2.8 和 4.2.4 给出的细节，将$\boldsymbol{C1}$的数据类型转换为比特串；
- A3：计算椭圆曲线点$\boldsymbol{S}=[\boldsymbol{h}]\boldsymbol{PB}$，若$\boldsymbol{S}$是无穷远点，则报错并退出；
- A4：计算椭圆曲线点$[\boldsymbol{k}]\boldsymbol{PB}=(\boldsymbol{x2},\boldsymbol{y2})$，按本文本第 1 部分 4.2.5 和 4.2.4 给出的细节，将坐标$\boldsymbol{x2},\boldsymbol{y2}$的数据类型转换为比特串；
- A5：计算$\boldsymbol{t}=\boldsymbol{KDF}(\boldsymbol{x2}\mathbin{/\!/}\boldsymbol{y2},\boldsymbol{klen})$，若$\boldsymbol{t}$为全 0 比特串，则返回 A1；
- A6：计算$\boldsymbol{C2}=\boldsymbol{M}\oplus\boldsymbol{t}$；
- A7：计算$\boldsymbol{C3}=\boldsymbol{Hash}(\boldsymbol{x2}\mathbin{/\!/}\boldsymbol{M}\mathbin{/\!/}\boldsymbol{y2})$；
- A8：输出密文$\boldsymbol{C}=\boldsymbol{C1}\mathbin{/\!/}\boldsymbol{C2}\mathbin{/\!/}\boldsymbol{C3}$。

✧ 解密算法：

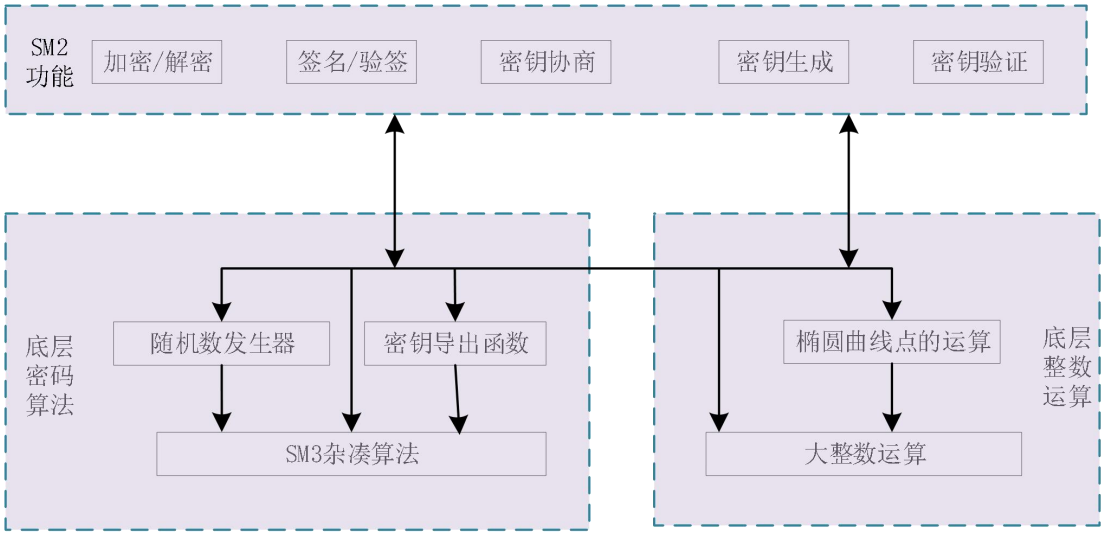
设 klen 为密文中$\boldsymbol{C2}$的比特长度。为了对密文$\boldsymbol{C}=\boldsymbol{C1}\mathbin{/\!/}\boldsymbol{C2}\mathbin{/\!/}\boldsymbol{C3}$进行解密，作为解密者的用户 B 应实现以下运算步骤：

- B1：从$\boldsymbol{C}$中取出比特串$\boldsymbol{C1}$，将$\boldsymbol{C1}$的数据类型转换为椭圆曲线上的点，验证$\boldsymbol{C1}$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$\boldsymbol{S}=[\boldsymbol{h}]\boldsymbol{C1}$，若$\boldsymbol{S}$是无穷远点，则报错并退出；
- B3：计算$[\boldsymbol{dB}]\boldsymbol{C1}=(\boldsymbol{x2},\boldsymbol{y2})$，按本文本第 1 部分 4.2.5 和 4.2.4 给出的细节，将坐标$\boldsymbol{x2},\boldsymbol{y2}$的数据类型转换为比特串；
- B4：计算$\boldsymbol{t}=\boldsymbol{KDF}(\boldsymbol{x2}\mathbin{/\!/}\boldsymbol{y2},\boldsymbol{klen})$，若$\boldsymbol{t}$为全 0 比特串，则报错并退出；
- B5：从$\boldsymbol{C}$中取出比特串$\boldsymbol{C2}$，计算$\boldsymbol{M}'=\boldsymbol{C2}\oplus\boldsymbol{t}$；
- B6：计算$\boldsymbol{u}=\boldsymbol{Hash}(\boldsymbol{x2}\mathbin{/\!/}\boldsymbol{M}'\mathbin{/\!/}\boldsymbol{y2})$，从$\boldsymbol{C}$中取出比特串$\boldsymbol{C3}$，若$\boldsymbol{u}\neq\boldsymbol{C3}$，则报错并退出；
- B7：输出明文 M′。

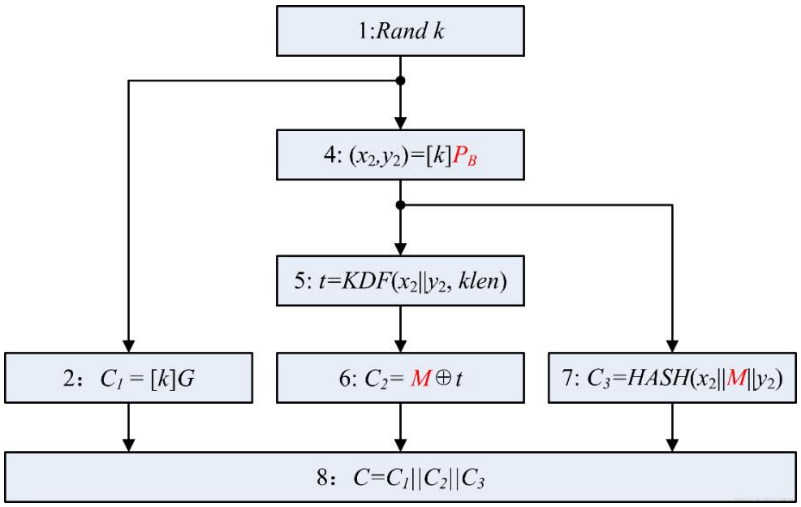
三、方案实现

1．算法流程图

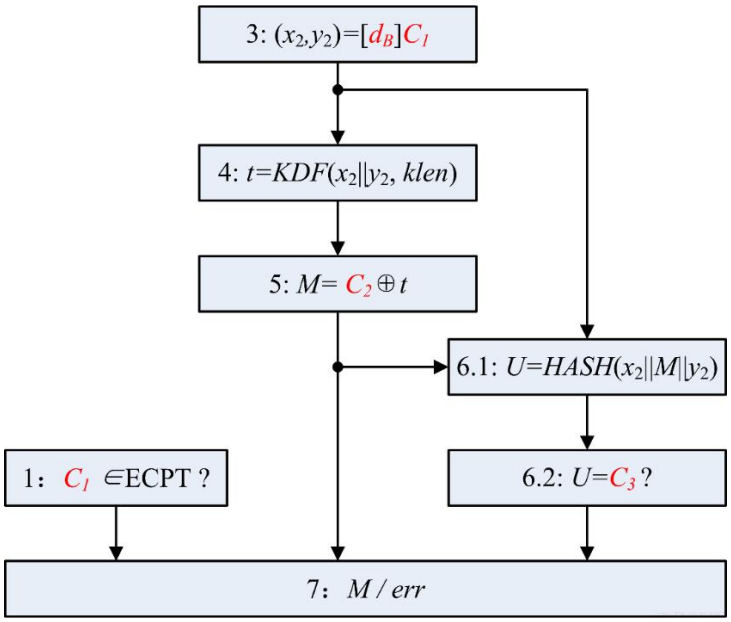
a．整体结构：



b．加密流程



c．解密流程



2. 主要函数介绍

a. _init_(self):初始化生成 Fp 域椭圆曲线的相关的安全参数

```
def _init_(self):
        self.p =
0x8542D69E4C044F18E8B92435BF6FF7DE457283915C45517D722EDB8
B08F1DFC3
        self.a =
0x787968B4FA32C3FD2417842E73BBFEFF2F3C848B6831D7E0EC65228
B3937E498
        self.b =
0x63E4C6D3B23B0C849CF84241484BFE48F61D59A5B16BA06E6E12D1D
A27C5249A
        self.h = 1
        self.Gx =
0x421DEBD61B62EAB6746434EBC3CC315E32220B3BADD50BDC4C4E6C1
47FEDD43D
        self.Gy =
0x0680512BCBB42C07D47349D2153B70C4E5D7FDFCBFA36EA1A85841B
9E46E09A2
        self.n =
0x8542D69E4C044F18E8B92435BF6FF7DD297720630485628D5AE74EE
7C32E79B7
        print("椭圆曲线参数如下:")
        print("p:%d" % self.p)
        print("a:%d" % self.a)
        print("b:%d" % self.b)
        print("Gx:%d" % self.Gx)
```

```
        print("Gy:%d" % self.Gy)
```

b. pro_private(self)：生成私钥

```
def pro_private(self):
        self.private_key = random.randint(1, self.n - 2)
        return self.private_key
```

c. PP(self, x, y)：二倍点计算

```
def PP(self, x, y):
        lumda = ((3 * x**2 + self.a) * inverse(2 * y, self.p)) %
self.p
        x1 = (lumda**2 - 2 * x) % self.p
        y1 = (lumda * (x - x1) - y) % self.p
        return x1, y1
```

d. add(self, x1, y1, x2, y2)：加法运算

```
def add(self, x1, y1, x2, y2):
        if x1 == 0 and y1 == 0:
            return x2, y2
        elif x1 == x2 and y1 == y2:
            x3, y3 = self.PP(x1, y1)
            return x3, y3
        elif x1 == x2 and y1 != y2:
            return (0, 0)
        elif x1 != x2:
            lumda = ((y2 - y1) * inverse(x2 - x1, self.p)) %
self.p
            x3 = (lumda**2 - x1 - x2) % self.p
            y3 = (lumda * (x1 - x3) - y1) % self.p
            return x3, y3
```

e. k_PP(self, x, y, k)：k 倍点计算

```
def k_PP(self, x, y, k):
        k = bin(k)[2:]
        x0 = 0
        y0 = 0
        coun = 0
        for i in k:
            if y0 != 0:
                x0, y0 = self.PP(x0, y0)
            if i == '1':
                x0, y0 = self.add(x0, y0, x, y)
            coun += 1
        return (x0, y0)
```

f. encrypt(self, M):加密

```
 def encrypt(self, M):
        k = random.randint(1, self.n - 1)
        # 消息为字符串，现转换成16进制
        m = binascii.b2a_hex(M.encode('utf-8'))
        m = hex(int(str(m)[2:-1], 16))[2:]
        C1 = self.k_PP(self.Gx, self.Gy, k)
        C1 = '04' + self.long_to_byte(C1[0]) +
self.long_to_byte(C1[1])
        public_key = self.k_PP(self.Gx, self.Gy,
self.private_key)
        if public_key[0] == 0 and public_key[1] == 0:
            print("error")
            exit(0)
        x2, y2 = self.k_PP(public_key[0], public_key[1], k)
        ml = len(m)
        print("\n输出公钥\nPx=%d" % public_key[0])
        print("Py=%d" % public_key[1])
        t = self.kdf(str(hex(x2)[2:]) + str(hex(y2)[2:]),
ml)
        if (int(t, 16) == 0):
            encrypt()
        C2 = hex(int(m, 16) ^ int(t, 16))[2:]
        s = (str(hex(x2)[2:]) + m +
str(hex(y2)[2:])).encode("utf8")
        C3 = hashlib.sha256(s).hexdigest()
        print("加密结果为:\n%s" % (C1 + C2 + C3))
        return C1 + C2 + C3
```

g. decrypt(self, C):解密

```
def decrypt(self, C):
        C1 = C[2:130]
        C3 = C[-64:]
        lC2 = len(C) - len(C1) - len(C3) - 2
        C2 = C[130:130 + lC2]
        x2, y2 = self.k_PP(int(C1[0:64], 16), int(C1[64:],
16), self.private_key)
        t = self.kdf(str(hex(x2)[2:]) + str(hex(y2)[2:]),
lC2)
        m = hex(int(C2, 16) ^ int(t, 16))[2:]
        s = (str(hex(x2)[2:]) + m +
str(hex(y2)[2:])).encode("utf8")
```

```
        CC3 = hashlib.sha256(s).hexdigest()
        if CC3 == C3:
            print("\n明文没被篡改")
            print(binascii.unhexlify(m))
        else:
            print("\n明文已被篡改")
```

3. 算法实现主要代码

```
if __name__ == '__main__':
    print("                椭圆曲线加密:                ")
    m = ECC()
    m._init_()
    m.pro_private()
    f = open("1.txt", "r")
    M = f.read()
    C = m.encrypt(M)
    m.decrypt(C)
    f.close()
```

四、数据分析

输入为 1.txt

明文: Only the children know what they are looking for. They waste their time over a rag doll and it becomes very importantto them;

```
PS C:\Users[redacted]\实验5-SM2椭圆曲线公钥密码算法"
PS C:\Users[redacted]> python -u "c:\Users[redacted]"
                    SM2椭圆曲线加密：
椭圆曲线参数如下：
p:60275702009245096385686171515219896416297121499402250955537857683885541941187
a:5449205298558957408044368562985702748167184172631336258559797854591532557224 8
b:45183185393608134601425506985501881231876135519103376096391853873370470098074
Gx:29905514254078361236418469080477708234343499662916671209092838329800180225085
Gy:29405937379755419157903904478921572542806770830401260612308519640632340013 14

输出公钥
Px=36563179310954843168043034362614958650316715177452583977305217763458823873451
Py=15398338323349491151409171296368557292835635357787796141057337486794763748119
加密结果为：
04717f34be933514f55f7578add3c57f61df678172a477a2541a857afb88f70af066617dee2d5737bc019ae19befa235eae7e9f2aaf12
7283b12dfb55c737e40746a3f778934c4b2286b7ffbe009022c0899e491f19cdd2344a137c3f7bb8ae745571506f2185f87a8ceb8d3e1
84f2e87ed29afc9db3c7f8ab4ee2ad622449a4f433ff25cf83fd9409468472eecd10000cb8288501661048ca95fd8a7fdf8155dc4e66f
8e1ec0dadc57177788efcb33c66395d926d4d8e7fe6c6920ead5ccf779f4abe397e2b8219ba8bdf8d940d0691b2bf5a34cc566758328c
3169ffb6
解密结果为：
04717f34be933514f55f7578add3c57f61df678172a477a2541a857afb88f70af066617dee2d5737bc019ae19befa235eae7e9f2aaf12
7283b12dfb55c737e40746a3f778934c4b2286b7ffbe009022c0899e491f19cdd2344a137c3f7bb8ae745571506f2185f87a8ceb8d3e1
84f2e87ed29afc9db3c7f8ab4ee2ad622449a4f433ff25cf83fd9409468472eecd10000cb8288501661048ca95fd8a7fdf8155dc4e66f
8e1ec0dadc57177788efcb33c66395d926d4d8e7fe6c6920ead5ccf779f4abe397e2b8219ba8bdf8d940d0691b2bf5a34cc566758328c
3169ffb6

明文未被篡改
b'Only the children know what they are looking for. They waste their time over a rag doll and it becomes very
 importantto them;'
PS C:\Users[redacted]>
```

五、思考与总结

1. 实验过程中遇到了什么问题，如何解决的？

   K 倍点运算的实现可通过循环调用二倍点计算函数实现。在加密部分，t 是通过密钥派生函数生成的一个中间参数，需要判断其是否为全 0 串，如果是则要重新选取随机数 k。

2. 通过该实验有何收获和感想？

   看起来很复杂的算法分解为一个个函数其实就能在很大程度上降低复杂度，做起来并没有那么复杂，需要的是我们肯动手。