山东大学网络空间安全学院

网络空间安全创新创业实践

Project 4 SM2 impl

姓名：张麟康

学号：201900301107

1 原理分析

1.1 ECC加密算法

1.1.1 椭圆曲线离散对数问题（ECDLP）

椭圆曲线上的两个点P和Q，k为整数。

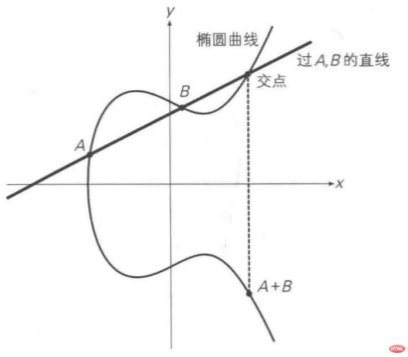
$$Q = kP$$

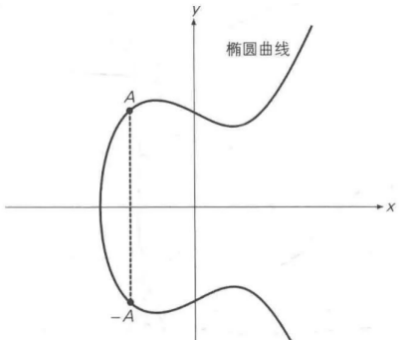
椭圆曲线加密的数学原理：点P称为基点；k为私钥；Q为公钥。

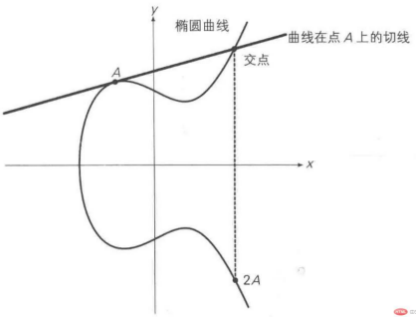
- 给定k和P，根据加法法则，计算Q很容易。
- 给定P和Q，求k非常困难。

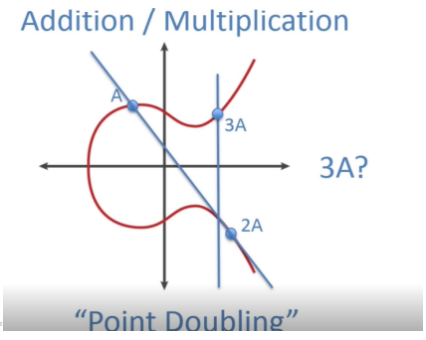
椭圆曲线：$y^2 = x^3 + ax + b(4a^3 + 27b^2 \neq 0)$**【描述方程与计算椭圆周长的方程类似】**

> 关于x轴对称，不存在奇点（处处有切线）。









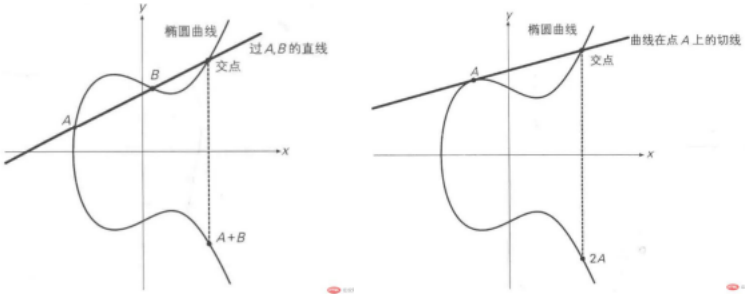
1.1.2 ECC过程

1. 选一条椭圆曲线$Ep(a,b)$，并取椭圆曲线上一点作为**基点**P
2. 选择一个大数k作为**私钥**，并生成公钥$Q=kP$
3. 加密：选择**随机数**r，将消息M生成密文C，密文是一个点对即$C=(rP,M+rQ)$
4. 解密：$M+rQ-k(rP)=M+r(kP)-k(rP)=M$

椭圆曲线是连续的，并不适合加密，要将椭圆曲线变成离散的点，因此把椭圆曲线定义在**有限域**上。

> 如果域F只包含有限个元素则称其为有限域。
>
> 有限域中元素的个数称为有限域的阶。
>
> 每个有限域的阶必为素数的幂，即有限域的阶可表示为p^n
>
> 该有限域通常称为Galois域，记作$GF(p^n)$。
>
> 在域的定义上，作如下修改：
>
> 1. 定义模p加法和模p乘法
> 2. 集合内的元素经过加法和乘法计算结果仍在集合内（封闭性）
> 3. 计算符合交换律、结合律、分配律
> 4. 加法和乘法有单位元

有限域上的椭圆曲线运算：



$$x_3 \equiv k^2 - x_1 - x_2 \pmod p$$
$$y_3 \equiv k(x_1 - x_3) - y_1 \pmod p$$
$$若P=Q,则k=\frac{3x_1^2+a}{2y_1} \pmod p$$
$$若P\neq Q,则k=\frac{y_2-y_1}{x_2-x_1} \pmod p$$

1.2 SM2加密算法及流程

1.2.1 加密算法

设需要发送的消息为比特串M，$klen$为M的比特长度。

1. 用随机数发生器产生随机数$k \in [1, n-1]$
2. 计算椭圆曲线点$C_1 = [k]G = (x_1, y_1)$，将C_1的**数据类型转换为比特串**
3. 计算椭圆曲线点$S = [h]P_B$，若S是无穷远点则报错并退出
4. 计算椭圆曲线点$[k]P_B = (x_2, y_2)$并将其**数据类型转换为比特串**
5. 计算$t = KDF(x_2||y_2, klen)$**【密钥派生函数】**，若t为全0比特串则返回第一步

> 密钥派生函数用于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算法。
> 设密码杂凑算法为 $H_v()$，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：
> 1) 计算 $Ha_i = H_v(Z \| ct)$；
> 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$(klen - (v \times \lfloor klen/v \rfloor))$比特；
> d) 令 $K = Ha_1 \| Ha_2 \| \cdots \| Ha_{\lceil klen/v \rceil - 1} \| Ha!_{\lceil klen/v \rceil}$。

6. 计算$C_2 = M \oplus t$
7. 计算$C_3 = Hash(x_2||M||y_2)$
8. 输出密文$C = C_1||C_2||C_3$

1.2.2 解密算法

1. 从密文C中取出比特串C_1并将其转换为椭圆曲线上的点坐标
2. 计算椭圆曲线上的点$S = [h]PB$，判断S是否为无穷远点，若是则错误退出。
3. 计算$[dB]C_1 = (x_2, y_2)$
4. 计算$t = KDF(x_2||y_2, klen)$，若t为全0比特串则错误退出
5. 从C中取出比特串C_2，计算$M' = C_2 \oplus t$

6. 计算$u = Hash(x_2||M'||y_2)$，从C中取出比特串C_3，比较u和C_3，若u不等于C_3则错误退出
7. 得到明文M'

1.2.3 正确性验证

由于$[dB]C_1 = [dB][k]G = [k]PB = (x_2, y_2)$，其中$PB = [dB]G$，易知$M' = M$。

因此，若$M = M'$则必有$u = C_3$，从而解密过程中的第6步用于验证解密是否成功。

1.3 SM2数字签名算法及流程

1.3.1 签名算法

设待签名的消息为M，为了获取消息的数字签名(r, s)，作为签名者的用户应实现一下运算步骤：

1. 置$\bar{M} = Z_A||M$
2. 计算$e = H_v(\bar{M})$，将e的数据类型转换为整数
3. 用随机数发生器产生随机数$k \in [1, n-1]$
4. 计算椭圆曲线点$(x_1, y_1) = [k]G$，将x_1的数据类型转换为整数
5. 计算$r = (e + x_1) \mod n$，若$r = 0$或$r + k = n$则返回步骤三。
6. 计算$s = ((1 + d_A)^{-1} \cdot (k - r \cdot d_A)) \mod n$，若$s = 0$则返回步骤3
7. 将r, s的数据类型转换为字符串，消息M的签名为(r, s)

1.3.2 验签算法

为了检验接收到的消息M'及其数字签名(r', s')，作为验证者的用户应实现以下运算步骤：

1. 检验$r' \in [1, n-1]$是否成立，若不成立则验证不通过
2. 检验$s' \in [1, n-1]$是否成立，若不成立则验证不通过
3. 置$\bar{M'} = Z_A||M'$
4. 计算$e' = H_v(\bar{M'})$，将其数据类型转换为整数
5. 将r', s'的数据类型转换为整数，计算$t = (r' + s') \mod n$，若$t = 0$则验证不通过
6. 计算椭圆曲线点$(x_1', y_1') = [s']G + [t]P_A$
7. 将x_1'的数据类型转换为整数，计算$R = (e' + x_1') \mod n$，检验$R = r'$是否成立，若成立则验证通过，否则验证不通过。

1.3.3 正确性验证

由于$R = (e' + x_1') \mod n$且$r = (e + x) \mod n$，若为合法签名消息对则必然满足$R = r'$。

2 具体实现

2.1 代码结构

本项目采用Python实现，哈希函数采用hashlib库中的sha256算法，项目主要包含两个文件即utils.py和sm2_new.py，其中utils.py中主要包含一些与SM2加密算法耦合度较低的工具类函数如计算哈希值、模逆运算等，而sm2_new.py文件中主要编写SM2类用来具体实现SM2算法的加密和数字签名功能。

2.2 SM2加密算法主体部分具体实现

首先编写SM2类，定义椭圆曲线的具体参数，其中size变量表示明文分组大小，v表示哈希函数的输出比特长度，h为余因子。

```
class SM2:
    # 椭圆曲线参数设置
    p = 0x8542D69E4C044F18E8B92435BF6FF7DE457283915C45517D722EDB8B08F1DFC3
    a = 0x787968B4FA32C3FD2417842E73BBFEFF2F3C848B6831D7E0EC65228B3937E498
    b = 0x63E4C6D3B23B0C849CF84241484BFE48F61D59A5B16BA06E6E12D1DA27C5249A
    Gx = 0x421DEBD61B62EAB6746434EBC3CC315E32220B3BADD50BDC4C4E6C147FEDD43D
    Gy = 0x0680512BCBB42C07D47349D2153B70C4E5D7FDFCBFA36EA1A85841B9E46E09A2
    n = 0x8542D69E4C044F18E8B92435BF6FF7DD297720630485628D5AE74EE7C32E79B7
    v = 256  # 哈希函数输出比特长度
    h = 1
    size = 2048  # 明文分组大小
```

对于加密函数encrypt()，其参数为字符串形式的明文和椭圆曲线点形式公钥，返回十六进制字符串形式的明文。

首先将明文编码，按照分组大小进行分组，构建明文分块列表，最后一个分块不足一整个分块时候作为单个分块处理。

```
cnt = self.size // 4
plain_block_list = []
i = -1
for i in range(len(plain) // cnt):
    plain_block_list.append(int(plain[cnt * i:cnt * (i + 1)], 16))
if math.ceil(len(plain) / cnt) != len(plain) // cnt:
    plain_block_list.append(int(plain[cnt * (i + 1):], 16))
```

接下来调用块加密函数encrypt_block()，干函数首先将分块转换为二进制串形式并补足为8的整数倍比特长，随后根据公钥，进入循环。

1. 用随机数发生器产生随机数$k \in [1, n-1]$

```
x, y = pk
while True:
    k = random.randint(1,self.n-1)
    # 生成随机数
```

2. 计算椭圆曲线点$C_1 = [k]G = (x_1, y_1)$，将C_1的**数据类型转换为比特串**

```
C1 = self.ecc_multiply(k, (self.Gx, self.Gy))
C1_binary = point2bit(C1)
```

3. 计算椭圆曲线点$S = [h]P_B$，若S是无穷远点则报错并退出

```
if not self.test_point(C1_point):
    exit("error")
```

4. 计算椭圆曲线点$[k]P_B = (x_2, y_2)$并将其**数据类型转换为比特串**

```
C = self.ecc_multiply(k, (x, y))
C_binary = point2bit(C, False)
```

5. 计算$t = KDF(x_2||y_2, klen)$**【密钥派生函数】**，若t为全0比特串则返回第一步

```
t = self.KDF(C_binary[0], plain_block_bin_len)
if t.count('0') != plain_block_bin_len:
    break
```

> KDF按如下方案编写：
>
> 密钥派生函数需要调用密码杂凑算法。
> 设密码杂凑算法为 $H_v()$，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：
> 1) 计算 $Ha_i = H_v(Z \| ct)$；
> 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 $(klen - (v \times \lfloor klen/v \rfloor))$ 比特；
> d) 令 $K = Ha_1 \| Ha_2 \| \cdots \| Ha_{\lceil klen/v \rceil - 1} \| Ha!_{\lceil klen/v \rceil}$。

```
def KDF(self, Z, klen):
    """
    密钥派生函数
    :param Z:密钥二进制串
    :param klen:明文二进制长度
    :return: 扩展后的密钥二进制串
    """
    ct = 1  # 计数器
    K = ''  # 十六进制串
    for i in range(klen // self.v):
        K += hash(Z + bin(ct)[2:].zfill(32))
        ct += 1
    # 取剩余部分
    f_length = klen / self.v
    if math.ceil(f_length) != int(f_length):
        K += hash(Z + bin(ct)[2:].zfill(32))[:(klen - self.v * int(f_length)) // 4]
    # 返回二进制串
    K_binary = bin(int(K, 16))[2:].zfill(klen)
    return K_binary
```

6. 计算$C_2 = M \oplus t$

```
            break
C2_binary = bin_xor(plain_block_bin, t)
```

7. 计算$C_3 = Hash(x_2||M||y_2)$

```
C3_hex = hash(C_binary[1] + plain_block_bin + C_binary[2])
```

8. 输出密文$C = C_1||C_2||C_3$

```
C1_hex = hex(int(C1_binary, 2))[2:].zfill(130)
C2_hex = hex(int(C2_binary, 2))[2:].zfill(plain_block_bin_len // 4)
return C1_hex + C2_hex + C3_hex
```

对于解密函数decrypt()，其参数为十六进制字符串形式的密文和私钥，返回明文字符串。

首先将密文划分为若干个密文分组。

```
cnt = (self.size + 256 + 520) // 4
cipher_block_list = []
i = -1
# 分块
for i in range(len(cipher) // cnt):
    cipher_block_list.append(cipher[cnt * i + cnt * (i + 1)])
if math.ceil(len(cipher) / cnt) != len(cipher) // cnt:
    cipher_block_list.append(cipher[cnt * (i + 1):])
```

接下来对于每个密文分组调用块解密函数decrypt_block()对每个块进行单独解密。

1. 从密文C中取出比特串C_1并将其转换为椭圆曲线上的点坐标

```
# 拆分密文各个部分
C1_hex = cipher_block[:130]
C2_hex = cipher_block[130:cipher_block_len - self.v // 4]
C3_hex = cipher_block[-self.v // 4:]
# C1转换为点
C1_point = hex2point(C1_hex)
```

2. 计算椭圆曲线上的点$S = [h]PB$，判断S是否为无穷远点，若是则错误退出。

```
if not self.test_point(C1_point):
    exit("error")
```

3. 计算$[dB]C_1 = (x_2, y_2)$

```
C1_point = hex2point(C1_hex)
dB_mul_C1 = self.ecc_multiply(sk, C1_point)
```

4. 计算$t = KDF(x_2||y_2, klen)$，若t为全0比特串则错误退出

```
# C2
C2_binary_len = len(C2_hex) * 4
C2_binary = bin(int(C2_hex, 16))[2:].zfill(C2_binary_len)

# 密钥派生
t = self.KDF(C1_binary[0], C2_binary_len)
```

5. 从C中取出比特串C_2，计算$M' = C_2 \oplus t$

```
plain_binary = bin_xor(C2_binary, t)
u = hash(C1_binary[1] + plain_binary + C1_binary[2])
```

6. 计算$u = Hash(x_2||M'||y_2)$，从C中取出比特串C_3，比较u和C_3，若u不等于C_3则错误退出

```
if u != C3_hex:
    exit(-1)
plain_hex = hex(int(plain_binary, 2))[2:].zfill(len(plain_binary) // 4)
return plain_hex
```

7. 得到明文M'

2.3 SM2椭圆曲线运算实现

根据有限域上椭圆曲线上点的运算法则，设计如下函数。

1. 两个相同点相加

```python
def ecc_add_same(self, G):
    """
    椭圆曲线上的相同坐标的两个点相加
    :param G: 生成元，基点
    :return: 相加之后的点
    """
    x1, y1 = G
    # 计算 斜率 k ，k 已不具备明确的几何意义
    tmp1 = 3 * x1 * x1 + self.a
    tmp2 = mod_inverse(2 * y1, self.p)
    k = tmp1 * tmp2 % self.p
    # 求 x3
    x3 = (k * k - x1 - x1) % self.p
    # 求 y3
    y3 = (k * (x1 - x3) - y1) % self.p

    return x3, y3
```

2. 两个不同点相加

```python
def ecc_add_diff(self, G1, G2):
    """
    椭圆曲线上两个不同点相加
    :param G1: 第一个点
    :param G2: 第二个点
    :return: 相加得到的点
    """
    x1, y1 = G1
    x2, y2 = G2
    # 计算 斜率 k
    tmp1 = y2 - y1
    tmp2 = mod_inverse((x2 - x1) % self.p, self.p)
    k = tmp1 * tmp2 % self.p
    # 求 x3
    x3 = (k * k - x1 - x2) % self.p
    # 求 y3
    y3 = (k * (x1 - x3) - y1) % self.p
    return x3, y3
```

3. 相邻的两个点相加

```python
def ecc_add_neighbor(self, point, pointBase):
    """
    相邻的两个点相加 ，pointBase 为基点，如：23P 点 + 24P 点
    :return: 新的点
    """
    return self.ecc_add_diff(pointBase, self.ecc_add_same(point))
```

4. 倍点

```python
def ecc_multiply(self, k, G):
    """
    椭圆曲线上的点乘以常数 k
    :param k: int
    :param G: 生成元 基点
    :return: 相乘之后的点
    """
    if k == 1:
        return G
    if k == 2:
        return self.ecc_add_same(G)
    if k == 3:
        return self.ecc_add_diff(G, self.ecc_add_same(G))
    if k % 2 == 0:
        return self.ecc_add_same(
            self.ecc_multiply(k // 2, G))
    if k % 2 == 1:
        return self.ecc_add_neighbor(self.ecc_multiply(k // 2, G), G)
```

2.4 SM2数字签名算法的主体实现

首先编写签名算法signify()，其参数为待签名消息M、用户标识IDA以及签名私钥和验签公钥。

依据国标按步骤实现，首先通过get_Z()方法得到ZA。

设待签名的消息为M，为了获取消息的数字签名(r, s)，作为签名者的用户应实现一下运算步骤：

1. 置$\bar{M} = Z_A || M$

```python
"""
ZA = get_Z(IDA, PA)
# A1
M_ = ZA + M
```

2. 计算$e = H_v(\bar{M})$，将e的数据类型转换为整数

```python
# A2
e = hash(M_)  # 返回十六进制字符串
e = int(e, 16)  # 十六进制字符串转换为整数
```

3. 用随机数发生器产生随机数$k \in [1, n-1]$

```python
# A3
k = random.randint(1, self.n - 1)
```

4. 计算椭圆曲线点$(x_1, y_1) = [k]G$，将x_1的数据类型转换为整数

```
# A4
x1 = self.ecc_multiply(k, (self.Gx, self.Gy))[0]
```

5. 计算$r = (e + x_1) \mod n$，若$r = 0$或$r + k = n$则返回步骤3

```
K = 0
while r == 0 or r + k == self.n:
```

6. 计算$s = ((1 + d_A)^{-1} \cdot (k - r \cdot d_A)) \mod n$，若$s = 0$则返回步骤3

```
K = 0
while r == 0 or r + k == self.n:
```

7. 将r, s的数据类型转换为字符串，消息M的签名为(r, s)

```
s = mod_inverse(self.n, 1 + dA)
# A7
r = int2bytes(r, math.ceil(math.log(self.n, 2) / 8))
s = int2bytes(s, math.ceil(math.log(self.n, 2) / 8))
return r, s
```

\

对于验签算法编写函数verify()，其参数为消息M及其签名sig，用户A的标识IDA以及用户A的公钥PA。

依据国标按步骤实现如下。

```
def verify(self, M, sig, IDA, PA):
    ZA = get_Z(IDA, PA)
    r = sig[0]
    s = sig[1]
    r = bytes2int(r)
    s = bytes2int(s)
```

1. 检验$r' \in [1, n-1]$是否成立，若不成立则验证不通过

```
if r < 1 or r > self.n - 1 or s < 1 or s > self.n - 1:
    return False
```

2. 检验$s' \in [1, n-1]$是否成立，若不成立则验证不通过

```
if r < 1 or r > self.n - 1 or s < 1 or s > self.n - 1:
    return False
```

3. 置$\bar{M}' = Z_A || M'$

```
M_ = ZA + M
```

4. 计算$e' = H_v(\bar{M}')$，将其数据类型转换为整数

```
e = hash(M_)
e = int(e, 16)
```

5. 将r', s'的数据类型转换为整数，计算$t = (r' + s') \mod n$，若$t = 0$则验证不通过

```
e = int(e, 16)
t = (r + s) % self.n
if t == 0:
    return False
```

6. 计算椭圆曲线点$(x_1', y_1') = [s']G + [t]P_A$

```
if t == 0:
    return False
x1 = self.ecc_add_diff(self.ecc_multiply(s, (self.Gx, self.Gy)), self.ecc_multiply(t, PA))[0]
```

7. 将x_1'的数据类型转换为整数，计算$R = (e' + x_1') \mod n$，检验$R = r'$是否成立，若成立则验证通过，否则验证不通过。

```
x1 = self.ecc_add_diff(self.ecc_multiply(s, (self.Gx, self.Gy)), self.ecc_multiply(t, P
R = (e + x1) % self.n
if R != r:
    return False
return True
```

3 运行结果

3.1 公钥加密算法

通过多次测试可以看到成功实现SM2算法的基本功能。

```python
if __name__ == '__main__':
    start = time.time()
    sm2_obj = SM2()
    key = sm2_obj.key_produce()
    PB, dB = key
    print('pk:', PB)
    print('sk', dB)
    plain = "sdu网络空间安全创新创业实践课程项目4"
    # plain = '123456'
    print('original text:', plain)
    # 加密
    cipher = sm2_obj.encrypt(plain, PB)
    print("cipher text:", cipher)
    # 解密
    p = sm2_obj.decrypt(cipher, dB)
    print("plain text:", p)
    # 判断
    if plain == p:
        print("encrypted and decrypted successfully.")
    else:
        print("some errors occurred.")

    print("time cost:", time.time() - start)
```

```
D:\Python310\python.exe "F:/course-project-2022/Project 4 SM2 impl/sm2_new.py"
original text: encryption standard
cipher text: 047e45c9a14ea78434d29ee5d8c648e3d153b0302c6835c67b362bf1aae1584c05432fc1da30d58a25e171cb4eb2ef1d0ddec1a955b1e2de90d0df08172d536256c3e4d
plain text: encryption standard
encrypted and decrypted successfully.
time cost: 0.2220611572265625
```

```
D:\Python310\python.exe "F:/course-project-2022/Project 4 SM2 impl/sm2_new.py"
pk: (73426673214970007951688187887144206986730494442436641777813329979535560 41420, 582058657663277483988716326832413422139336021687946220086162349 80
sk 353190090307318241969672864729249464463319519515954581446363791448752873747 86
original text: 张麟康201900301107
cipher text: 040dcd7dae80dec4dcda0622bb1ffa3e53bd909cd79ae4edd6a3cd098aa202eed24fe4046a8167f272307b846672fab0db47d21ca5632b589d1c9d1cf69f33418f58a1a
plain text: 张麟康201900301107
encrypted and decrypted successfully.
time cost: 0.2915663719177246
```

```
D:\Python310\python.exe "F:/course-project-2022/Project 4 SM2 impl/sm2_new.py"
pk: (39453409813875809348336033001030257353691481795038981983018454890026595112322, 5106951883373985518460083202442435959805222459471243276020954577
sk 11783966013213672389906165908476051988306196545186874774671020095937324207044
original text: sdu网络空间安全创新创业实践课程项目4
cipher text: 040585cb9eff27509c14fe933c3328f3a51e266df8e3539e3736e43ad957d92dfd7b7b4438b5121f66be1c377240b75e5d7c5fd4308ebb2befe9be527d95770c96691c9
plain text: sdu网络空间安全创新创业实践课程项目4
encrypted and decrypted successfully.
time cost: 0.29775428771972656
```

3.2 数字签名算法

经验证，成功实现数字签名的签名和验签过程。

```
#
dA = dB
PA = PB
IDA = '201900301107@mail.sdu.edu.cn'
M = "sdu网络空间安全创新创业实践课程项目4"
Sig = sm2_obj.signify(M, IDA, dA, PA)
print('message:', M)
print('IDA:', IDA)
print('signature:')
print('r:', Sig[0])
print('s:', Sig[1])
print('verification(valid):')
sm2_obj.verify(M, Sig, IDA, PA)
print('verification(invalid):')
M='sdu网络空间安全创新创业实践课程项目5'
sm2_obj.verify(M, Sig, IDA, PA)
```

```
D:\Python310\python.exe C:/Users/Connor/Desktop/pySM2-master/SM2_Signature.py
message: sdu网络空间安全创新创业实践课程项目4
IDA: 201900301107@mail.sdu.edu.cn
signature:
r: [7, 136, 126, 225, 11, 23, 139, 94, 76, 194, 153, 168, 250, 213, 153, 233, 41, 222, 29, 166, 159, 15, 176, 183, 247, 225, 51, 66, 39, 68, 39, 11]
s: [130, 208, 216, 218, 112, 50, 57, 187, 88, 241, 58, 34, 8, 187, 157, 196, 101, 99, 96, 62, 178, 180, 32, 226, 23, 73, 39, 18, 98, 99, 168, 113]
verification(valid):
verified successfully.
verification(invalid):
some errors occurred.
```