SM2

一、引言

SM2 是中国自主设计的公钥密码标准（依据 GBT 32918.1-2016），基于椭圆曲线离散对数问题（ECDLP）构建，主要应用于数字签名、密钥交换等安全场景。本文档将详细解析一个 SM2 算法的 Python 实现程序，阐述其核心原理、代码结构及功能实现。

二、SM2 原理

SM2 算法的安全性基于椭圆曲线离散对数问题的计算复杂性，其核心流程包括：

1. 椭圆曲线参数定义
2. 椭圆曲线点运算规则
3. 密钥对生成机制
4. 签名与验证算法流程

三、程序

1、椭圆曲线参数定义

SM2 算法基于特定的椭圆曲线方程：$y^2 \equiv x^3+ax+b \pmod p$，程序中严格遵循国家标准定义了以下关键参数：

```
p = 0xFFFFFFFEFFFFFFFFFFFFFFFFFFFFFFFF00000000FFFFFFFFFFFFFFFF
# 有限域素数模数
a = 0xFFFFFFFEFFFFFFFFFFFFFFFFFFFFFFFF00000000FFFFFFFFFFFFFFFC
# 曲线系数 a
b = 0x28E9FA9E9D9F5E344D5A9E4BCF6509A7F39789F515AB8F92DDBCBD414D940E93
# 曲线系数 b
n = 0xFFFFFFFEFFFFFFFFFFFFFFFFFFFFFFFF7203DF6B21C6052B53BBF40939D54123
# 基点 G 的阶
Gx = 0x32C4AE2C1F1981195F9904466A39C9948FE30BBFF2660BE1715A4589334C74C7
# 基点 G 的 x 坐标
Gy = 0xBC3736A2F4F6779C59BDCEE36B692153D0A9877CC62A474002DF32E52139F0A0
# 基点 G 的 y 坐标
```

参数说明：

p：定义有限域 GF (p)，所有运算均在该有限域内进行

a，b：椭圆曲线方程的系数

n：基点 G 的阶，即 n*G 为无穷远点

G：椭圆曲线上的生成元基点，所有公钥均为该点的倍数

2、数学基础模块

模运算实现

模逆运算作为有限域中的除法运算：

```
def mod_inverse(a, m):
    """模逆运算：计算 a 在模 m 下的逆元"""
    g, x, y = extended_gcd(a, m)
    if g != 1:
        return None  # 逆元不存在
    else:
        return x % m
def extended_gcd(a, b):
    """扩展欧几里得算法：用于计算模逆"""
    if a == 0:
        return (b, 0, 1)
    else:
        g, y, x = extended_gcd(b % a, a)
        return (g, x - (b // a) * y, y)
```

椭圆曲线点表示

程序定义了 Point 类表示椭圆曲线上的点：

```
class Point:
    """椭圆曲线上的点表示"""
    def __init__(self, x, y, is_infinity=False):
        self.x = x % p
        self.y = y % p
        self.is_infinity = is_infinity
     # 标记是否为无穷远点（加法零元）
    def __eq__(self, other):
        """点的相等性判断"""
        if self.is_infinity and other.is_infinity:
            return True
        if self.is_infinity or other.is_infinity:
            return False
        return self.x == other.x and self.y == other.y
```

3、椭圆曲线点运算

点加法（point_add 函数）

点加法实现了椭圆曲线上两点的加法运算，遵循以下规则：
若两点为 $P(x_1,y_1)$ 和 $Q(x_2,y_2)$ 且 $P \neq Q$：
斜率 $\lambda=(y_2-y_1)/(x_2-x_1) \bmod p$
新点 $R(x_3,y_3)$：$x_3=\lambda^2-x_1-x_2 \bmod p$，
$y_3=\lambda(x_1-x_3)-y_1 \bmod p$
若 $P=Q$（点加倍）：
斜率 $\lambda=(3x_1^2+a)/(2y_1) \bmod p$
新点计算规则同上

```
def point_add(p1, p2):
    """椭圆曲线点加法实现"""
    if p1.is_infinity:
        return p2
    if p2.is_infinity:
        return p1
    if p1.x == p2.x and p1.y != p2.y:
        return Point(0, 0, True)  # 无穷远点

    if p1 != p2:
        # 不同点相加
        dx = (p2.x - p1.x) % p
        dy = (p2.y - p1.y) % p
        inv_dx = mod_inverse(dx, p)
        if inv_dx is None:
            return Point(0, 0, True)
             # 逆元不存在，返回无穷远点
        lam = (dy * inv_dx) % p
    else:
        # 同一点加倍
        dy = (2 * p1.y) % p
        inv_dy = mod_inverse(dy, p)
        if inv_dy is None:
            return Point(0, 0, True)
             # 逆元不存在，返回无穷远点
        lam = ((3 * p1.x * p1.x + a) * inv_dy) % p

    x3 = (lam * lam - p1.x - p2.x) % p
```

```
    y3 = (lam * (p1.x - x3) - p1.y) % p
    return Point(x3, y3)
```

点乘法（point_multiply 函数）

点乘法实现了点的标量乘法（k*P=P+P+...+P，共 k 次），采用二进制快速幂法优化运算效率：

```
def point_multiply(p, k):
    """椭圆曲线点乘法（倍点加法实现）"""
    result = Point(0, 0, True)  # 无穷远点
    current = p
    while k > 0:
        if k % 2 == 1:
            result = point_add(result, current)
        current = point_add(current, current)  # 加倍点
        k = k // 2  # 右移一位
    return result
```

4、密钥对生成

SM2 密钥对生成遵循以下规则：
私钥 d：1 到 n-2 之间的随机整数
公钥 Q：Q=d*G（基点 G 乘以私钥 d 的点积）

```
def generate_key_pair():
    """生成 SM2 密钥对"""
    d = random.randint(1, n-2)  # 私钥
    Q = point_multiply(Point(Gx, Gy), d)  # 公钥
    return d, Q
```

5、签名与验证算法

签名算法（sm2_sign 函数）

签名流程：
1. 对消息做哈希处理得到 e
2. 生成随机数 k，计算点 P=k*G
3. 计算 r=(e+P.x)mod n（需确保 r≠0 且 r+k≠n）
4. 计算 s=[(k-r*d)/(1+d)]mod n（需确保 s≠0）
5. 签名结果为(r,s)

```
def sm2_sign(d, msg):
    """SM2 签名算法实现"""
    e = sm3_hash(msg)  # 哈希计算
    while True:
        k = random.randint(1, n-1)
        P = point_multiply(Point(Gx, Gy), k)
        r = (e + P.x) % n
        if r == 0 or r + k == n:
            continue
        s = (mod_inverse(1 + d, n) * (k - r * d)) % n
        if s != 0:
            break
    return (r, s)
```

验证算法（sm2_verify 函数）

验证流程：
1. 检查 r 和 s 的有效性（$1 \le r,s \le n-1$）
2. 计算消息哈希 e 和 $t=(r+s) \bmod n$（$t \neq 0$）
3. 计算点 $P_1=sG$ 和 $P_2=tQ$，得到 $P=P_1+P_2$
4. 验证 $(e+P.x) \bmod n == r$ 是否成立

```
def sm2_verify(Q, msg, signature):
    """SM2 验证算法实现"""
    r, s = signature
    if r < 1 or r > n-1 or s < 1 or s > n-1:
        return False

    e = sm3_hash(msg)
    t = (r + s) % n
    if t == 0:
        return False

    P1 = point_multiply(Point(Gx, Gy), s)
    P2 = point_multiply(Q, t)
    P = point_add(P1, P2)

    if P.is_infinity:
        return False

    return (e + P.x) % n == r
```

四、结果

程序通过示例代码展示了完整的 SM2 算法应用流程:

```
if __name__ == "__main__":
    # 生成密钥对
    private_key, public_key = generate_key_pair()
    print(f"私钥: {hex(private_key)}")
    print(f"公钥: {public_key}")

    # 待签名消息
    message = b"Hello, SM2!"
    print(f"消息: {message.decode()}")

    # 签名
    signature = sm2_sign(private_key, message)
    print(f"签名: (r={hex(signature[0])},
s={hex(signature[1])})")

    # 验证
    valid = sm2_verify(public_key, message, signature)
    print(f"验证结果: {'成功' if valid else '失败'}")
```

示例运行结果可验证：

1. 合法消息的签名能通过验证
2. 篡改后的消息签名验证失败
3. 完整展示了从密钥生成到签名验证的全流程

```
============== RESTART: C:/Users/DELL/Desktop/project/sm2.py ==============
私钥: 0xef3a15e8acbbcb528b840ca12d3d56b37f23e5f03d9dc8657c096f83734c5cb0
公钥: Point(0xaa8b32f6978a2ae37b59d889cc12e7d68d689ffca975cf2291cd63ae3370e
6e1, 0x5870ce36d5fb94d1ce9fc123b2976d70b928a6eccad4c87e70a97852d653c8d3)
消息: Hello, SM2!
签名: (r=0x3f3e132ea7b28d21808cff1632b083a15a32ca387bdd8eca340a003f77f392f7
, s=0x9f24fa3f20636e692d3fc047b65c5bde4fda65ddd335fc18d3bc368f11ef068)
验证结果: 成功

============== RESTART: C:/Users/DELL/Desktop/project/sm2.py ==============
私钥: 0xafe8beadaf35f52c11d257d45a7f7699c90cf9ce1e3e971ca711739f3d84ee39
公钥: Point(0xdd4fd0df3d5fef347feee2d6e734f5f32ea24f47deeed0130d8533ad60aef
8b3, 0xc0e3bd14b481814b59ed10a295870f3d48bf32489f58559b0d9e88f67d194aa9)
消息: Hello, SM2!
签名: (r=0xd96945528e6542e8b803fa4168143a09560393422151e612921f7cf5e7d99acc
, s=0x8a7efc4bca6b4b48438e11394d134ed5ad8507b9062dfd28a73e016fe613d5de)
验证结果: 成功
```