创新创业实验 project5

一、SM2 算法基础

SM2 是中国国家标准 GM/T 0003-2012 定义的椭圆曲线公钥密码算法，基于 **256 位椭圆曲线**，主要用于：

- 数字签名（SM2-Sign）
- 密钥交换（SM2-KeyExchange）
- 公钥加密（SM2-Encrypt）

核心参数（推荐曲线）：

- 素数域：$p = 0xFFFFFFFEFFFFFFFFFFFFFFFFFFFFFFFF00000000FFFFFFFFFFFFFFFF$
- 曲线方程：$y^2 = x^3 + ax + b$，其中 $a = 0xFFFFFFFEFFFFFFFFFFFFFFFFFFFFFFFF00000000FFFFFFFFFFFFFFFC$，$b = 0x28E9FA9E9D9F5E344D5A9E4BCF6509A7F39789F515AB8F92DDBCBD414D940E93$
- 阶数：$n = 0xFFFFFFFEFFFFFFFFFFFFFFFFFFFFFFF7203DF6B21C6052B53BBF40939D54123$
- 基点：$G = (x_G, y_G)$，其中 $x_G = 0x32C4AE2C1F1981195F9904466A39C9948FE30BBFF2660BE1715A4589334C74C7$，$y_G = 0xBC3736A2F4F6779C59BDCEE36B692153D0A9877CC62A474002DF32E52139F0A0$

二、核心实现与优化

1. 椭圆曲线点运算（基础但关键）

椭圆曲线点的加法、标量乘法是 SM2 的核心，直接影响性能。

```
class Point:
    def __init__(self, x: int, y: int, is_inf: bool = False):
        self.x = x
        self.y = y
        self.is_inf = is_inf  # 标记是否为无穷远点

    def __eq__(self, other) -> bool:
        if self.is_inf:
            return other.is_inf
        return self.x == other.x and self.y == other.y

    def __repr__(self) -> str:
        if self.is_inf:
            return "Point(inf)"
        return f"Point(0x{self.x:064x}, 0x{self.y:064x})"


# 预定义点
INF = Point(0, 0, True)  # 无穷远点
G = Point(x_G, y_G)      # 基点
```

2. 模运算优化

SM2 大量依赖模运算，通过 Python 内置函数和数学优化提升效率

```
        if k == 0:
            return INF
        naf = naf_representation(k)
        result = INF
        current = p

        for digit in naf:
            if digit == 1:
                result = point_add(result, current)
            elif digit == -1:
                # 加上当前点的逆元 (x, -y mod p)
                inv_current = Point(current.x, (-current.y) % p)
                result = point_add(result, inv_current)
            # 双倍当前点
            current = point_add(current, current)
        return result
```

-

标量乘法使用**二进制快速幂**（时间复杂度$O(\log k)$），替代朴素的循环累加（$O(k)$）。

- 模逆运算利用**费马小定理**（$a^{p-2} \mod p$），比扩展欧几里得算法更简洁。

3. 密钥生成

```
def generate_key_pair() -> Tuple[int, Point]:
    """生成SM2密钥对：(私钥d, 公钥Q=d*G)"""
    # 生成1 < d < n的随机私钥
    while True:
        d = int.from_bytes(os.urandom(32), byteorder='big')
        if 1 < d < n:
            break
    # 计算公钥Q = d * G
    Q = point_mul(d, G)
    return d, Q
```

- 私钥生成使用 os.urandom（密码学安全随机数生成器），避免伪随机数风险。
- 模 n 确保私钥在有效范围内（$1 < d < n$）。

4. 签名与验证（SM2-Sign）

```
def verify(Q: Point, message: bytes, signature: Tuple[int, int]) -> bool:
    """SM2签名验证：使用公钥Q验证签名"""
    r, s = signature

    # 验证参数范围
    if not (1 <= r < n and 1 <= s < n):
        return False

    e = int.from_bytes(sm3_hash(message), byteorder='big')
    t = (r + s) % n

    if t == 0:
        return False

    # 计算u1*G + u2*Q
    u1 = (s * t) % n
    u2 = (r * t) % n
    u1G = point_mul(u1, G)
    u2Q = point_mul(u2, Q)
    P = point_add(u1G, u2Q)

    if P.is_inf:
        return False
```

- 签名中随机数 k 的生成使用 os.urandom，避免重复（防止私钥泄露）。
- 验证时合并计算 u1 和 u2，减少一次点乘法

三、高级优化策略

1. 标量乘法进一步优化

NAF（非相邻形式）表示：将标量 k 转换为 NAF 形式，减少点加法次数（平均减少 50%）。

```
def naf_representation(k: int) -> list:
    """将整数k转换为非相邻形式(NAF)，减少点加法次数"""
    naf = []
    while k > 0:
        if k % 2 == 1:
            sign = 2 - (k % 4)
            naf.append(sign)
            k -= sign
        else:
            naf.append(0)
        k = k // 2
    return naf
```

2. 预计算基点倍数

- 对常用基点 G 的倍数（如\(2^i G\)）预计算并缓存，加速多次签名 / 加密中的点乘法。

3. 模运算优化

- 使用 gmpy2 库替代内置 int，提供更快的大数模运算（需安装 pip install gmpy2）。