SM2的软件实现优化

SM2的基础实现以及优化

SM2实现

SM2 是一种椭圆曲线公钥密码算法，主要用于数字签名和密钥交换。SM2 的流程主要包括密钥生成、签名、验签和加解密四个部分。

1. 密钥生成:

- 用户选择一个素数域上的椭圆曲线和一个基点G。
- 用户随机选择一个私钥d，并计算公钥P = d * G (其中*表示椭圆曲线上的标量乘法)。
- 私钥d对用户保密，公钥P可以公开。

2. 签名:

- 用户对消息M进行摘要计算，得到消息摘要e。
- 用户选择一个随机数k，并计算椭圆曲线上的点C1 = k * G。
- 用户计算s = (k - r * dA) mod n，其中dA是用户的私钥，n是基点G的阶，r是C1的x坐标与e的运算结果。
- 用户将签名(r, s)与消息M一起发送给接收方。

3. 验签:

- 接收方使用发送方的公钥PA和消息M计算消息摘要e'。
- 接收方计算R = (s * G + r * PA)。
- 如果R的x坐标等于r，则验证通过，否则验证失败。

4. 加密:

- 发送方使用接收方的公钥PB对消息M进行加密，得到密文C = (C1, C2, C3)。
- C1 = k * G，其中k是随机数。
- C2 = M XOR t，其中t = KDF(x2 || y2, klen)，KDF是密钥派生函数，x2, y2是k * PB的坐标。
- C3 = Hash(x2 || M || y2)，Hash是密码杂凑函数。

5. 解密:

- 接收方使用自己的私钥dA对密文C进行解密。
- 计算S = dA * C1。
- 计算t' = KDF(x2 || y2, klen)，其中x2, y2是S的坐标。

- 计算M = C2 XOR t'。
- 验证C3是否等于Hash(x2 || M || y2)。

优化

1. 预计算表优化

在 `_build_precompute_table` 方法中，代码构建了一个预计算表来加速基点的标量乘法：

- 原理：预先计算并存储基点G的多个倍数（如G, 2G, 3G,...,15G），这样在实际计算k*G时可以直接查表获取部分结果，减少重复计算。
- 实现：表的大小由 `precompute_window_size` 决定（默认为4，即16个预计算点）。
- 优势：对于频繁使用基点G的运算（如密钥生成），可以显著减少计算量。

2. 滑动窗口法

在 `_sliding_window_scalar_mul` 方法中实现了滑动窗口法：

- 原理：将标量k表示为二进制形式，然后以固定大小的"窗口"扫描这些位，每次处理一个窗口而不是单个位。
- 步骤：
    a. 预计算奇数倍点（P, 3P, 5P,...）
    b. 从最高位开始扫描k的二进制表示
    c. 遇到1时，寻找最长的奇数窗口
    d. 通过加倍和加法操作组合结果
- 优势：减少了点加法操作的次数，窗口越大，性能提升越明显（但内存消耗也增加）。

3. 窗口法

在 `_windowed_scalar_mul` 方法中实现了固定窗口法：

- 原理：将标量k分成固定大小的窗口（如4位一组），每组对应一个预计算表中的值。
- 步骤：
    a. 将k分解为多个窗口
    b. 从最高位窗口开始处理
    c. 对每个窗口，先进行多次加倍操作，然后加上对应预计算点
- 优势：当使用预计算表时，效率更高，特别适合重复使用相同基点的情况。

4. 模运算优化

在 `_mod_inv` 和 `_mod_mul` 方法中实现了优化的模运算：

- 模逆计算：使用费马小定理（对于素数p，x^-1 ≡ x^(p-2) mod p），通过Python内置的pow函数高效计算。

- 模乘法：直接使用Python的模运算符优化。

5. 点运算优化

在 `point_add` 和 `point_double` 方法中实现了优化的点加法和点加倍：

- 点加法：优化了斜率计算，减少了模逆运算次数
- 点加倍：同样优化了斜率计算
- 检查特殊：处理无穷远点和相同点/相反点的情况

poc验证

1. 重用随机数k导致私钥泄露原理分析

当同一个用户对两个不同消息使用相同的随机数k进行签名时，攻击者可以通过两个签名推导出私钥。

推导过程：

- 第一个签名：`s1 = (1 + dA)^-1 * (k - r1 * dA) mod n`
- 第二个签名：`s2 = (1 + dA)^-1 * (k - r2 * dA) mod n`
- 解方程组可得：`dA = (s2 - s1) / (s1 - s2 + r1 - r2) mod n`

2. 不同用户使用相同k导致私钥泄露原理分析

当两个不同用户使用相同的随机数k进行签名时，他们可以互相推导出对方的私钥。

推导过程：

- Alice签名：`s1 = (1 + dA)^-1 * (k - r1 * dA) mod n`
- Bob签名(相同k)：`s2 = (1 + dB)^-1 * (k - r2 * dB) mod n`
- Alice可以计算：`dB = (k - s2) / (s2 + r2) mod n`
- Bob可以计算：`dA = (k - s1) / (s1 + r1) mod n`

3. 与ECDSA使用相同的d和k导致私钥泄露原理分析

当用户使用相同的私钥d和随机数k分别生成SM2和ECDSA签名时，攻击者可以通过两个签名推导出私钥。

推导过程：

- ECDSA签名：`s1 = (e1 + r1*d) * k^-1 mod n`
- SM2签名(相同d,k)：`s2 = (1 + d)^-1 * (k - r2*d) mod n`
- 解方程组可得：`d = (s1*s2 - e1) / (r1 - s1*s2 - s1*r2) mod n`

4. 签名可延展性问题原理分析

SM2签名(r,s)和(r,-s)都是有效的签名，这可能导致区块链网络分裂等问题。

验证原理：

1. 原始签名(r,s)验证：`t = r + s mod n`，`(x1,y1) = s*G + t*P`
2. 修改后签名(r,-s)验证：`t' = r + (-s) mod n = (r - s) mod n`，`(x1',y1') = (-s)*G + t'*P = s*G + t*P`（因为t'$P = (r\text{-}s)$d$G = r$dG - sd*G)

伪造签名

中本聪的签名基于ECDSA，其安全性依赖：

1. 私钥保密性：私钥 `d` 未知。
2. 随机数唯一性：每次签名的 `k` 必须不同。

量子计算攻击：

- 使用Shor算法破解椭圆曲线离散对数问题（ECDLP），从公钥 `P = d·G` 计算 `d`。
- 当前量子计算机未达到所需比特数（需百万级量子比特）。

签名延展性：

- ECDSA的 `(r, s)` 和 `(r, -s mod n)` 均为有效签名，但比特币网络已强制使用低 `s` 值（BIP 62）。

数学漏洞：

- 若签名时重用 `k`，可通过两次签名推导私钥（见POC 1）。

除非突破ECC数学安全性或获取中本聪的私钥，否则是无法实际完成伪造数字签名的。