Project 5: SM2 的软件实现优化

a). python 实现 sm2 的基础实现以及各种算法的改进尝试

b). 20250713-wen-sm2-public.pdf 中提到的关于签名算法的误用 分别基于做 poc 验证，给出推导文档以及验证代码

一. SM2 的基础实现和优化

```
=== 性能对比 ===
基础版 50次操作: 10.707秒
优化版 50次操作: 9.099秒
性能提升: 1.2x
```

计算层面优化：

预计算加速：滑动窗口法

采用 4-bit 窗口优化，预计算 1~15 倍基点，减少实时计算点乘的成本。

使用类变量 _precomputed 缓存窗口值，避免重复计算。

模逆缓存

采用 @lru_cache 修饰 _mod_inv() 方法，对模逆操作结果进行缓存，加速重复计算。

点加与点倍乘单独优化

point_add 和 point_double 独立重写，避免不必要的判断分支。

针对 x1 == x2 情况提前返回无穷远点。

内存与数据结构优化：

缓存与局部变量

模逆与预计算点的缓存显著降低重复内存读写。

预热预计算表通过 SM2UltraOpt._precompute() 方法在首次运行时构建。

延迟初始化

scalar_mult 中延迟窗口分解，仅在必要时调用窗口值，节省内存。

签名与验证流程优化：

消除重复计算

sign() 和 verify() 中共享哈希计算 e = Hash(ZA || msg)。

点乘使用滑动窗口方法，验证点加使用优化版本 _point_add。

二. 签名算法误用攻击

PoC1：ECDSA 中重用相同 kkk 导致私钥泄露

两条使用相同 kkk 的签名 (r1,s1)(r_1,s_1)(r1 ,s1) 与 (r2,s2)(r_2,s_2)(r2 ,s2)：

s1=k−1(e1+dr1),s2=k−1(e2+dr2)s_1 = k^{-1}(e_1 + d r_1),\quad s_2 = k^{-1}(e_2 + d r_2)s1 =k−1(e1 +dr1),s2 =k−1(e2 +dr2)

两式相减：
s1−s2=k−1(e1−e2)(modn)s_1 - s_2 = k^{-1}(e_1 - e_2) \pmod ns1 −s2 =k−1(e1 −e2)(modn)
因而
k=(e1−e2)⋅(s1−s2)−1(modn)k = (e_1 - e_2)\cdot (s_1 - s_2)^{-1} \pmod nk=(e1 −e2)⋅(s1 −s2)−1(modn)

得到 kkk 后，代入任一签名的等式可解 ddd：
d=(s1k−e1)r1−1(modn)d = (s_1 k - e_1) r_1^{-1} \pmod nd=(s1 k−e1)r1−1 (modn)

结论：若 kkk 被重复使用，且攻击者获得两条签名与对应消息，则可恢复私钥 ddd。

```
=== PoC1: ECDSA 重用 nonce k 导致私钥泄露

已知：两条签名 (r1,s1), (r2,s2) 使用同一 nonce k。
ECDSA 签名公式:
  s = k^{-1} (e + d*r)  （模 n）
于是对两条消息有:
  s1 = k^{-1}(e1 + d*r1)
  s2 = k^{-1}(e2 + d*r2)
减两式得:  s1 - s2 = k^{-1} (e1 - e2)  =>  k = (e1 - e2) * inv(s1 - s2)

计算中间量 (模 n=191):
  e1 = 188
  e2 = 142
  s1 = 185
  s2 = 2
  num = e1 - e2 = 46
  den = s1 - s2 = 183
  恢复的 k = 42
  恢复的 d = 74
  原始 d   = 74
验证:  成功
```

PoC2：同一私钥在 ECDSA 与 SM2 中重用同一 kkk

设 ECDSA 的签名参数为 (re,se)(r_e,s_e)(re ,se)、SM2 的签名为 (rs,ss)(r_s,s_s)(rs ,ss)（均使用相同私钥 ddd 与相同 nonce kkk）。

ECDSA 给出：
k=(ee+dre)· se−1(modn)(1)k = (e_e + d r_e)\cdot s_e^{-1} \pmod n \quad (1)k=(ee +dre)· se−1 (modn)(1)

SM2 给出（教学形式）：
k=ss(1+d)+rsd(modn)(2)k = s_s(1+d) + r_s d \pmod n \quad (2)k=ss (1+d)+rs d(modn)(2)

将 (1),(2) 相等并整理代数，得到关于 ddd 的一次线性同余方程：
(re−sess−sers)· d≡(sess−ee)(modn)(r_e - s_e s_s - s_e r_s)\cdot d \equiv (s_e s_s - e_e) \pmod n(re −se ss −se rs)· d≡(se ss −ee)(modn)
记 A=re−sess−sersA = r_e - s_e s_s - s_e r_sA=re −se ss −se rs 、B=sess−eeB = s_e s_s - e_eB=se ss −ee 。若 A≢0A \not\equiv 0A ≡0（模 nnn），则
d=B· A−1(modn)d = B \cdot A^{-1} \pmod nd=B· A−1(modn)

结论：跨算法（不同签名方案）复用同一 kkk 仍可能使私钥暴露，因为不同算法给出的是不同的关于 kkk 与 ddd 的代数表达，组合后可解出 ddd。

```
已知：
  ECDSA: k = (e_e + d * r_e) * inv(s_e)   （等式 1）
  SM2  : k = s_s*(1 + d) + r_s * d          （等式 2）
将两式相等并整理，得到线性方程（关于 d）：
  (re - se*ss - se*rs) * d = (se*ss - ee)   （模 n）

中间量 (模 n=191): A = 21, B = 102
  恢复的 d = 114
  原始 d   = 114
验证： 成功
=== PoC2 完成 ===
```