Project5 SM2 的软件实现和优化

一．项目说明

sm2_0.cpp　　基础 sm2 算法实现

Sm2_1..cpp　　优化后的 sm2 算法

二. sm2 原理结构

椭圆曲线加密算法原理如下：

设私钥、公钥分别为 k、K，即 K = kG，其中 G 为 G 点，

公钥加密：选择随机数 r，将消息 M 生成密文 C，该密文是一个点对，即：

C = {rG, M+rK}，其中 K 为公钥

私钥解密：M + rK - k(rG) = M + r(kG) - k(rG) = M

其中 k、K 分别为私钥、公钥。

1、签名过程

设 G 是椭圆曲线上的参考点，dA 是私钥，PA 是公钥，PA=dA∗G

对 e 进行数字签名得到签名结果(r,s)，计算过程是：

首先选取随机数 k，　计算 r=e+x1，其中(x1,y1)=k∗G ,计算 s=(1+dA)的-1 次方∗(k−r∗dA)

该过程用私钥进行。

2、签名验证的过程

验证签名就是利用得到的签名、公钥、椭圆曲线参数等对签名进行验证，验证主要步骤是：

首先计算 t=r+s，如果 t=0 表明没有通过。 然后通过 t 与 s 计算曲线上的点(x1,y1)=s*G+t*PA

再计算 R=x1+e，然后验证 R 与 r 是不是相等，如果相等则表明验证通过。

3、验证的原理

(x1,y1)=s*G+t*PA

=s*G+(r+s)*PA

=s*G+(r+s)*dA*G

=(1+dA)*s*G+r*dA*G

=(1+dA)*(1+dA)的-1 次方*(k−r*dA)*G+r*dA*G

=(k−r*dA)*G+r*dA*G

=k*G

可知依据公钥得到的椭圆曲线上的这个点和签名时的点是一致的。，由这个 x1 和收到的信息相加，看是否与发送的签名 r 是否相符，相符则通过验证。

三.优化思路:

1.模逆计算缓存优化

```
@lru_cache(maxsize=1024)
def mod_inv(a: int, m: int) -> int:
    if a < 0:
        a = (a % m + m) % m

    # 使用扩展欧几里得算法
    old_r, r = a, m
    old_x, x = 1, 0
```

使用 @lru_cache 装饰器缓存常用的模逆运算结果，消除重复计算。直接在模逆函数内实现扩展欧几里得，避免函数调用开销

2. 椭圆曲线点运算优化

3. 标量乘法算法优化

```
def scalar_mul(k: int, point: Point) -> Point:
    if point is None or k % N == 0:
        return None

    if k < 0:
        return scalar_mul(-k, point_neg(point))

    result = None
    base = point

    while k > 0:
        if k & 1:
            result = point_add(result, base)
        base = point_add(base, base)
        k >>= 1
```

改进了二进制展开法的实现

4. SM3 哈希计算

```
def sm3_hash(message: bytes) -> bytes:
    # padding
    msg_len = len(message) * 8
    message += b'\x80'

    #  计算填充长度
    k = (448 - (len(message) * 8) % 512) % 512
    message += b'\x00' * (k // 8)
    message += msg_len.to_bytes(8, 'big')
```

直接计算，减少中间变量

优化效果分析:

优化项	优化前复杂度	优化后复杂度	提升幅度
模逆计算(缓存命中)	O(log n)	O(1)	显著提升
椭圆曲线点运算	O(1)	O(1)	常数优化
SM3 哈希计算	O(n)	O(n)	常数优化

四. 代码运行结果

优化前:

```
C:\Users\86139\AppData\Local\Programs\Python\Python39\python.exe D:\pyproject\sm2_0.py
私钥 d = 0x69e43994d266fa0966dd6a55d07df1794b4c71ca5472722f19284b709a2f867b
公钥 P = ('0x12201c3ec8f512030281022cb4dd2f41b93e1f95a21e020ff31aaaeac8c18f3f',
 '0xc0b77277b30dc079a640dad8d04f5eec726785c2c615ffed6edabfd81e088c82')
计算哈希值： be69cd9a2848b0b2d668f7a02327c3b724e9dbc71631990b749d34401f5e59fb
签名结果 r = 0x7219a56cc5a4110b38b23dc9cb0f72f0b2ebcb476578090a05483d9077986354
s = 0xe9c8f1a080821ecfd3bd83aa1eb3e74817c0c9798b4272d171175abd9d088076
验证 = True
所用时间： 2.860121726989746 秒
```

优化后:

```
C:\Users\86139\AppData\Local\Programs\Python\Python39\python.exe D:\pyproject\sm2_1.p
私钥 d = 0x51f50cd7e6faebfa729e751dcce94ff2891ae64ee72f0142764223afd51cdbe1
公钥 P = (0x93b64c63c67f927669663e036aea8aaa7e520795841beb8ebca9104949a1174a,
 0x11c87e9d50dcd36891601a822f653a623e2ce20a80c9b534fccb6c37adabbe2b)
ZA = 15e8828b21d457881bed7973316915019d5ec84ba14ac375639962f1a6ec8faf
签名 r = 0x74ee4ac16bdb873de9892b096d12ff235d1123a81d0cc64e9cb128b2405f455e
签名 s = 0x5db326cbc3ae4bf7362f5e2db0f142a743cd4c58116d6a2779e1f182905e0688
所用时间： 0.219221830368042 秒
验证结果: ✓ 通过
```

优化效果约为 13 倍。