Project 5

1、SM2 算法实现

1.1 算法实现过程

1.1.1 Point 数据类

- 用于表示椭圆曲线上的点(x, y)，或者特殊的“无穷远点”
- 当 $x = -1$ 且 $y = -1$ 时，标记为无穷远点 O

1.1.2 求模逆元

利用扩展欧几里得算法来求解模逆元：即给定整数 a 和质数 p ，找到一个整数 x 使得 `(a * x) % p == 1`

1.1.3 两点相加

- 如果其中一个点是无穷远点，则直接返回另一个点
- 若两点有相同的 x 但不同的 y，则它们的和应为无穷远点
- 当两点相同时，使用切线斜率公式
- 当两点不同时，使用割线斜率公式
- 新点的坐标计算：根据斜率推导出新的 x 和 y 值

1.1.4 标量乘法

初始结果设为无穷远点 O，然后逐步累加

1.1.5 生成非对称密钥对

- 随机选择一个私钥 `d`
- 计算对应的公钥 `Q = d·G`
- 确保公钥不是无穷远点后返回键对

1.1.6 密钥派生函数

- 基于 SHA-256 哈希迭代构造任意长度的秘密字节流
- 输入熵源 `Z` 和一个计数器 `ct` ，每次迭代都增加计数器以避免重复；
- 最终截取所需长度的前缀作为输出密钥

1.1.7 整数转字节数组

将一个大端序的大整数按指定长度拆分成字节数组

1.1.8 加密函数

- 随机选取一个临时私钥 `k`
- 计算临时公钥：`C1=k·G`
- 计算共享秘密点：`S = k·pub_key`
- 提取熵源：`Z = (S.x XOR pub_key.x) mod p`
- 通过 `KDF` 得到固定长度的密钥
- 将消息逐字节与密钥异或得到密文 `C2`
- 最终密文格式为 `[C1.x][C1.y][C2]`，均为定长32字节字段

1.1.9 主函数

- 调用 `generate_keypair()` 生成一对非对称密钥
- 循环迭代，每次迭代调用 `generate_random_string` 生成指定长度的随机消息
- 调用 `encrypt` 函数计算随机消息的 SM2 加密密文，在此过程中捕获异常避免程序中断
- 记录加密计算的开始时间和结束时间
- 输出密文并计算时间
- 计算并输出总用时和平均时间

1.2 运行结果

该算法实现了 SM2 加密功能，并通过对加密操作进行十次迭代以计算平均时间：（此处仅展示部分结果）

```
--------------------第10次加密--------------------
原始消息: kp2ZaBu8xXggtd0uUaJOOWxQpJ8M3BWSshQqkgjmEaqzkuUcReL8U5aCH8hTpllT
加密后的密文:
 1ba716e5c88c8f2b74d0d46f6fb5f6715b0a6a2ebface2e5aa85e54764b0149055a458928f189615a2ad2e85b0be74a8a36ff8c1222478bceadac277387bd3b7ce17916f0931bf501386c6335
 631253cf006e97a2724b2391b94991911174 21ad60ff2440314a0052ebfd02e4920402af702ef0d3d46ab2b23e6c9005239791d
============================================================
10次加密总用时：0.265536秒，平均加密一次用时：0.026554秒
```

2、poc 验证

下演示如何通过两个不同的用户使用相同的随机数 `k` 来推导出对方的私钥

2.1 原理分析

Alice 和 Bob 各自生成自己的密钥对，他们分别用自己的私钥对消息进行签名。签名过程中，首先选择一个随机数 k；计算点 $R = kG$，其中 G 是基点；计算哈希值 e；计算签名 $s = (k^{-1} \cdot (e + dR)) \mod n$，其中 d 是私钥。

给定签名 (R, s) 和消息 m，攻击者可以通过以下公式推导出私钥：$d = \frac{e - x_R}{k^{-1} \cdot s} \mod n$，其中 x_R 是点 R 的横坐标。

如果两个用户使用相同的随机数 k 来签名，那么攻击者可以利用这个信息来推导出双方的私钥：（假设 Alice 的签名为 (R_A, s_A)，Bob 的签名为 (R_B, s_B)）

- 对于 Alice：$d_A = \frac{e_A - x_{R_A}}{k^{-1} \cdot s_A} \mod n$
- 对于 Bob：$d_B = \frac{e_B - x_{R_B}}{k^{-1} \cdot s_B} \mod n$
- 由于 k 相同，因此 k^{-1} 也相同，从而可以推导出：
  - $d_A = \frac{e_A - x_{R_A}}{k^{-1} \cdot s_A} \mod n$
  - $d_B = \frac{e_B - x_{R_B}}{k^{-1} \cdot s_B} \mod n$
- 通过这两个方程，攻击者可以计算出：

$$d_A - d_B = \frac{e_A - x_{R_A}}{k^{-1} \cdot s_A} - \frac{e_B - x_{R_B}}{k^{-1} \cdot s_B} = \frac{(e_A - e_B) - (x_{R_A} - x_{R_B})}{k^{-1}}$$

2.2 代码实现

2.2.1 签名函数

- 生成一个范围在 [1, n-1] 之间的随机整数 `k`
- 通过标量乘法计算点 `R = k*G`，其中 `G` 是基点。如果 `R` 是无穷远点，则返回 `None`
- 对消息进行哈希处理，取前64个字符，并将其转换为整数
- 使用私钥和 `e` 计算签名 `s`
- 返回 `(R, s)` 作为签名

2.2.2 验证函数

- 从签名中提取 `R` 和 `s`
- 检查 `R` 是否为无穷远点， `s` 是否在有效范围内
- 对消息进行哈希处理，取前64个字符，并将其转换为整数
- 计算 `w = s^-1 mod n`
- 根据公式计算 `u1` 和 `u2`
- 根据公式计算 `X` 和 `Y`
- 比较计算出的 `(X, Y)` 与 `R` 是否相等

2.2.3 生成密钥对并签名消息

- 调用 `generate_keypair` 函数生成 Alice 和 Bob 的密钥对
- 分别对 Alice 和 Bob 的消息进行签名
- 输出签名结果

2.2.4 推导私钥

- 对消息进行哈希处理，取前64个字符，并将其转换为整数
- 从签名中提取 `R` 和 `s`
- 使用公式计算 `k`
- 使用公式推导私钥
- 输出推导出的私钥

2.3 运行结果

上述代码通过两个不同的用户使用相同的随机数 `k` 来推导出对方的私钥，结果如下：

```
Alice 的签名： (Point(x=42945097421892471623833629430894317712054153196478408211200598160990567396566494362859
7, y=32286620591620541372074532942568892361290385734580424734088249442630538669927288016172, is_infinite=False),
 122919482609080896193182219379236116176761394084778703074790363472221749225129700505554)
Bob 的签名： (Point(x=30781077541205526765757135802140240899802303904946598526107588195866561537341216734980
9, y=21203744002639815138551455447742641137570157511057457638116659115218410247441506030929 4, is_infinite=False),
 79162772955919323009245147549111656664007625955094761138700931279534300790242624720 7)
推导出的 Alice 的私钥： 387962491415011835652845702039630176454031535416198786961828779221108952054842994492
推导出的 Bob 的私钥： 850053094417361123802664759951571402155638052296462601485736217506628260618660279197
```

3、伪造中本聪的数字签名

3.1 实现过程

3.1.1 签名过程

- 对消息进行 SHA-256 哈希得到整数 $e = H(m) \mod n$
- 随机选择一个秘密整数 $k \in [1, n-1]$
- 计算临时密钥对 `(R, r)`，其中 $R = k \cdot G$，$r = R.x \mod n$
- 如果 `r == 0`，重新选 `k`
- 计算签名分量 $s = \left(k^{-1} \cdot (e + d \cdot r)\right) \mod n$
- 返回二元组 `(r, s)` 作为签名

3.1.2 验证过程

给定签名 `(r, s)`，验证是否满足以下等式：$u_1 \cdot G + u_2 \cdot Q \equiv R \mod p$，其中 $u_1 = e'/s$，$u_2 = r/s$

3.2 运行结果

上述代码以 SM2 算法为基础实现伪造中本聪的数字签名，结果如下：

```
私钥（十六进制）: 0x1413ab9fcd03b42f4823f96d63d34092856a898c70035cb63229a8dcafe59b1e7885a
公钥坐标: X=360821611758195156071334894897361357151945422037318640756938459744331863768391055541737,
 Y=406081091717079571151965119864117187151555924675385499570639701147831613287121470242124

原始消息:
伪造中本聪的数字签名

签名结果（r, s）: r=a18cb9f4edc698984ad5044c5b3f713ca7d83f1f7de9d53a0e8583fbbf79e395d6ba03,
 s=4fae06916e0d73999fa5f9e1c0a30b116859f48a05724c1b4251894b032d27f8bcbeb5

签名验证结果: 无效
篡改后的消息验证结果: 正确拒绝
```