SM2 数字签名安全性分析实验报告：k 值误用导致的私钥泄露

实验目的

本实验旨在验证 SM2 数字签名算法中 k 值管理不当可能导致的安全风险，具体包括：

k 值直接泄露导致私钥被破解

同一用户重复使用 k 值导致私钥泄露

不同用户使用相同 k 值导致私钥互推

通过数学推导和代码验证，展示 k 值在数字签名中的关键作用及其安全影响。

k 值泄露导致私钥破解

已知签名(r, s)和 k 值：

3.1 SM2 signature: leaking k

- Precompute：
  - $Z_A = H_{256}(ENTL_A||ID_A||a||b||x_G||y_G||x_A||y_A)$
- Key Generation： $P_A = d_A \cdot G$， order is n
- Sign(Z_A, M): $Sign_{d_A}(M, Z_A) \rightarrow (r, s)$
  - Set $\overline{M} = Z_A||M$,
  - $e = H_v(\overline{M})$
  - $k \leftarrow Z_n^*$， $kG = (x_1, y_1)$
  - $r = (e + x_1) \bmod n$,
  - $s = \left((1 + d_A)^{-1} \cdot (k - r \cdot d_A)\right) mod\ n$
  - Signature is (r, s)
- Verify (r, s) of M with P_A
  - $Z_A = H_{256}(ENTL_A||ID_A||a||b||x_G||y_G||x_A||y_A)$
  - Set $\overline{M} = Z_A||M$， $e = H_v(\overline{M})$
  - $t = (r + s) \bmod n$
  - $(x_1, y_1) = sG + tP_A$
  - $R = (e + x_1) \bmod n$, $Verify\ R = r$

- Compute d_A with $\sigma = (r, s)$ and k：
  - $s = \left((1 + d_A)^{-1} \cdot (k - r \cdot d_A)\right) mod\ n$
  - $s(1 + d_A) = (k - r \cdot d_A) \bmod n$
  - $d_A = (s + r)^{-1} \cdot (k - s) \bmod n$

重复使用 k 值导致私钥泄露

3.1 SM2 signature: reusing k

PART3 Application

- Precompute：
  - $Z_A = H_{256}(ENTL_A||ID_A||a||b||x_G||y_G||x_A||y_A)$
- Key Generation： $P_A = d_A \cdot G$， order is n
- Sign(Z_A, M): $Sign_{d_A}(M, Z_A) \rightarrow (r, s)$
  - Set $\overline{M} = Z_A||M$,
  - $e = H_v(\overline{M})$
  - $k \leftarrow Z_n^*$， $kG = (x_1, y_1)$
  - $r = (e + x_1) \bmod n$,
  - $s = \left((1 + d_A)^{-1} \cdot (k - r \cdot d_A)\right) mod\ n$
  - Signature is (r, s)
- Verify (r, s) of M with P_A
  - $Z_A = H_{256}(ENTL_A||ID_A||a||b||x_G||y_G||x_A||y_A)$
  - Set $\overline{M} = Z_A||M$， $e = H_v(\overline{M})$
  - $t = (r + s) \bmod n$
  - $(x_1, y_1) = sG + tP_A$
  - $R = (e + x_1) \bmod n$, $Verify\ R = r$

- Signing message M_1 with d_A
  - Randomly select $k \in [1, n-1]$， $kG = (x, y)$
  - $r_1 = (Hash(Z_A||M_1) + x) \bmod n$
  - $s_1 = \left((1 + d_A)^{-1} \cdot (k - r_1 \cdot d_A)\right) mod\ n$
- Signing message M_2 with d_A
  - Reuse the same k, $kG = (x, y)$
  - $r_2 = (Hash(Z_A||M_2) + x) \bmod n$
  - $s_2 = \left((1 + d_A)^{-1} \cdot (k - r_2 \cdot d_A)\right) mod\ n$
- Recovering d_A with 2 signatures $(r_1, s_1), (r_2, s_2)$
  - $s_1(1 + d_A) = (k - r_1 \cdot d_A) \bmod n$
  - $s_2(1 + d_A) = (k - r_2 \cdot d_A) \bmod n$
  - $d_A = \frac{s_2 - s_1}{s_1 - s_2 + r_1 - r_2} \bmod n$

使用相同k值对两个不同消息M_1, M_2签名得到两个签名(r_1,s_1)和(r_2,s_2)：

$$\begin{cases} s_1 = (1+d)^{-1} \cdot (k - r_1 \cdot d) \\ s_2 = (1+d)^{-1} \cdot (k - r_2 \cdot d) \end{cases}$$

推导私钥d：

$$\begin{aligned} (1+d)(s_1 - s_2) &= (k - r_1 d) - (k - r_2 d) \\ (1+d)(s_1 - s_2) &= d(r_2 - r_1) \\ s_1 - s_2 &= d\left[(r_2 - r_1) - (s_1 - s_2)\right] \\ d &= (s_1 - s_2) \cdot \left[(r_2 - r_1) - (s_1 - s_2)\right]^{-1} \mod n \end{aligned}$$

两个用户使用相同 k 值导致私钥互推

设用户A、B分别使用相同k值对各自消息签名：

$$\begin{cases} s_A = (1+d_A)^{-1} \cdot (k - r_A \cdot d_A) \\ s_B = (1+dB)^{-1} \cdot (k - r_B \cdot d_B) \end{cases}$$

用户A可推导用户B的私钥：

$$\begin{aligned} k &= s_A(1+d_A) + r_A d_A \quad (\text{A可计算k}) \\ s_B(1+dB) &= k - r_B d_B \\ s_B + s_B d_B + r_B d_B &= k \\ d_B(s_B + r_B) &= k - s_B \\ d_B &= (k - s_B) \cdot (s_B + r_B)^{-1} \mod n \end{aligned}$$

同理，用户B可推导用户A的私钥。

实验设计

k 值泄露实验

```
def test_k_leakage():
    priv_key, pub_key = create_key_pair()
    user_hash = compute_identity_hash(user_id, pub_key[0], pub_key[1])
    (r_val, s_val), k_val = generate_signature(priv_key, message, user_hash)

    # 数学推导私钥
    derived_priv = (k_val - s_val) * modular_inverse(r_val + s_val, ORDER_N) % ORDER_N
```

重复使用 k 值实验

```
def reusing_k_leakage():
```

```
    # 强制使用相同 k 值签名两个不同消息
    fixed_k = secrets.randbelow(ORDER_N)
    r1, s1 = sign_with_fixed_k(msg1, user_hash, fixed_k)
    r2, s2 = sign_with_fixed_k(msg2, user_hash, fixed_k)

    # 数学推导私钥
    numerator = (s2 - s1) % ORDER_N
    denominator = (s1 - s2 + r1 - r2) % ORDER_N
    derived_priv = numerator * modular_inverse(denominator, ORDER_N) % ORDER_N
```

两个用户相同 k 值实验

```
def two_users_same_k_leakage():
    fixed_k = secrets.randbelow(ORDER_N)
    # 用户 A 签名
    r1, s1 = sign_with_fixed_k(priv_key1, pub_key1, user_id1, msg, fixed_k)
    # 用户 B 签名
    r2, s2 = sign_with_fixed_k(priv_key2, pub_key2, user_id2, msg, fixed_k)

    # 用户 A 推导用户 B 私钥
    derived_priv2 = (fixed_k - s2) * modular_inverse(s2 + r2, ORDER_N) % ORDER_N
    # 用户 B 推导用户 A 私钥
    derived_priv1 = (fixed_k - s1) * modular_inverse(s1 + r1, ORDER_N) % ORDER_N
```

实验结果：

```
原始私钥: 0x214490853699db6820226a270f1e1e72eb1b1ccf1cfb8f9041a55974d701dfdf
推导私钥: 0x214490853699db6820226a270f1e1e72eb1b1ccf1cfb8f9041a55974d701dfdf
私钥匹配: True
原始私钥: 0x5e85a026c217f48325981dea40fc9bc4721eba2439476416574a9a4547f01480
推导私钥: 0x5e85a026c217f48325981dea40fc9bc4721eba2439476416574a9a4547f01480
私钥匹配: True
=== 用户1推导用户2的私钥 ===
用户2原始私钥: 0x1ce94923026791599efdf2fb6ec8c8e72ab9ead889e7994e4d21a6b146547579
用户1推导私钥: 0x1ce94923026791599efdf2fb6ec8c8e72ab9ead889e7994e4d21a6b146547579
匹配结果: True

=== 用户2推导用户1的私钥 ===
用户1原始私钥: 0x53fe71efb6ee96ab60fda37b09ec9716f27e0a2b40afa471e73c08f0cd145d83
用户2推导私钥: 0x53fe71efb6ee96ab60fda37b09ec9716f27e0a2b40afa471e73c08f0cd145d83
匹配结果: True
```