



山东大学 网络空间安全学院

创新创业实践 Project5c

学生姓名：滕怀源

学号：202200460054

学院：网络空间安全学院

专业班级：2022 级网安 2 班

完成时间：2025 年 7 月 26 日

目录

1 伪造中本聪的数字签名	3
1.1 实验代码说明	3
1.2 数学推导	3
1.3 实验结果	4
1.4 安全性分析	4
A scapy 函数	5

1 伪造中本聪的数字签名

本节通过在 Kaggle 平台运行的 Python 实验代码，验证了 SM2 签名算法在随机数 k 重用场景下的安全漏洞。实验模拟攻击者通过收集目标用户两条不同消息的签名，并利用数学推导恢复目标私钥，从而实现伪造任意消息签名。

1.1 实验代码说明

核心代码流程如下：

1. **生成密钥对**：调用 `sm2_keygen()` 随机生成用户 Alice 的私钥 d_A 与公钥 $P_A = d_A G$ 。
2. **签名过程**：使用固定随机数 k_{fixed} 对两条不同消息 M_1 、 M_2 签名，生成 (r_1, s_1) 和 (r_2, s_2) 。
3. **攻击步骤**：攻击者通过公式

$$d_A \equiv (s_2 - s_1) \cdot ((s_1 - s_2) + (r_1 - r_2))^{-1} \pmod{n}$$

恢复 Alice 的私钥。

4. **伪造签名**：攻击者使用恢复的 d_A 对一条新的消息 M_f 生成签名 (r_f, s_f) ，并用公钥 P_A 验证通过。

1.2 数学推导

SM2 签名过程如下：

$$\begin{aligned} e &= H(M), \\ (x_1, y_1) &= kG, \\ r &= (e + x_1) \pmod{n}, \\ s &= (1 + d_A)^{-1}(k - rd_A) \pmod{n}. \end{aligned}$$

整理 s 的定义式可得：

$$s(1 + d_A) \equiv k - rd_A \pmod{n}.$$

对于两次签名 (r_1, s_1) 和 (r_2, s_2) ，有

$$\begin{cases} s_1(1 + d_A) \equiv k - r_1 d_A \pmod{n}, \\ s_2(1 + d_A) \equiv k - r_2 d_A \pmod{n}. \end{cases}$$

两式相减，消去 k ：

$$(s_1 - s_2)(1 + d_A) \equiv (r_2 - r_1)d_A \pmod{n}.$$

移项得：

$$(s_1 - s_2) \equiv d_A((r_2 - r_1) - (s_1 - s_2)) \pmod{n}.$$

最终得到攻击公式：

$$d_A \equiv (s_1 - s_2) \cdot ((r_2 - r_1) - (s_1 - s_2))^{-1} \pmod{n}.$$

1.3 实验结果

Kaggle 运行结果显示：

```
=== 1. Alice 生成密钥 ===
Alice 公钥: (96182348615822097906329167833847028352674562863970427933536819529864515804010, 28495314961583272454618394348346193766539238364067201064786606742710001054779)
Alice 私钥 (仅Alice知道): 39740498730967353462529454594286702720299214613827093173805240373431192299443

=== 2. Alice 使用相同k签名两条消息 ===
签名1: r1=33184028689507590961614684689334421310288948716852851723351815883740495798186, s1=52415869675834309349379556937791648127504490868869987139738017148813056728545
签名2: r2=69599298667370672595909723771685394300663805493555760399949503838549899759742, s2=2132389044546292607315681150177433390523863199280412663351334152467103954344

=== 3. 攻击者利用两次签名恢复私钥 ===
恢复的私钥: 39740498730967353462529454594286702720299214613827093173805240373431192299443, 是否匹配: True

=== 4. 攻击者伪造签名 ===
伪造签名: r=57389017497857855863459954365547949119389426713671528339822643360222561906097, s=72667335261518464473567786836763345042951459373351988650575511094654016558435
验证伪造签名: True
```

- 原始私钥 d_A 与攻击恢复的私钥完全一致，证明攻击可行。
- 使用恢复的 d_A 对任意新消息 M_f 签名，公钥 P_A 验证通过，表明攻击者可完全伪造合法签名。

1.4 安全性分析

实验验证了 SM2 签名算法对随机数 k 的依赖性极强，一旦 k 重用或泄露，攻击者能够在多项式时间内恢复私钥，进而伪造任意签名。因此，必须使用 RFC6979 等规范生成确定性 k 或高强度随机数，避免复用。

A scapy 函数

```

1  import hashlib
2  import random
3
4  # =====
5  # SM2 椭圆曲线参数 (sm2p256v1)
6  # =====
7  p = int("FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF00000000FFFFFFFFFFFFFFFF", 16)
8  a = int("FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF00000000FFFFFFFFFFFFFFFC", 16)
9  b = int("28E9FA9E9D9F5E344D5AEF20E93E3FBD6B5F6F4C52C9A7A3EB5C3C4F9DCC73E", 16)
10 n = int("FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF7203DF6B21C6052B53BBF40939D54123", 16)
11 Gx = int("32C4AE2C1F1981195F9904466A39C9948FE30BBFF2660BE1715A4589334C74C7", 16)
12 Gy = int("BC3736A2F4F6779C59BDCEE36B692153D0A9877CC62A474002DF32E52139F0A0", 16)
13 G = (Gx, Gy)
14
15 # ===== 椭圆曲线运算 =====
16 def inverse_mod(x, m):
17     return pow(x, -1, m)
18
19 def point_add(P, Q):
20     if P == (0, 0): return Q
21     if Q == (0, 0): return P
22     if P == Q:
23         lam = (3 * P[0]*P[0] + a) * inverse_mod(2 * P[1], p) % p
24     else:
25         lam = (Q[1]-P[1]) * inverse_mod(Q[0]-P[0], p) % p
26     x = (lam*lam - P[0] - Q[0]) % p
27     y = (lam*(P[0]-x)-P[1]) % p
28     return (x, y)
29
30 def scalar_mul(k, P):
31     R = (0, 0)
32     while k > 0:
33         if k & 1:
34             R = point_add(R, P)
35             P = point_add(P, P)
36             k >>= 1
37     return R
38
39 # ===== 哈希函数 (模拟SM3) =====
40 def sm3_hash(msg):
41     return int(hashlib.sha256(msg.encode()).hexdigest(), 16)
42
43 # ===== SM2 KeyGen =====
44 def sm2_keygen():
45     d = random.randint(1, n-1)
46     P = scalar_mul(d, G)

```

```

47     return d, P
48
49 # ===== SM2 签名 =====
50 def sm2_sign(msg, d, k=None):
51     e = sm3_hash(msg)
52     if not k:
53         k = random.randint(1, n-1)
54     x1, y1 = scalar_mul(k, G)
55     r = (e + x1) % n
56     s = (inverse_mod(1+d, n) * (k - r*d)) % n
57     return (r, s, k)
58
59 # ===== SM2 验签 =====
60 def sm2_verify(msg, sig, P):
61     r, s = sig
62     e = sm3_hash(msg)
63     t = (r + s) % n
64     x1, y1 = point_add(scalar_mul(s, G), scalar_mul(t, P))
65     R = (e + x1) % n
66     return R == r
67
68 # ===== 攻击: 复用k恢复私钥 =====
69 def attack_reuse_k(r1, s1, r2, s2):
70     num = (s2 - s1) % n
71     den = (s1 - s2 + r1 - r2) % n
72     return (num * inverse_mod(den, n)) % n
73
74 # =====
75 # 模拟攻击场景
76 # =====
77 print("==== 1. Alice 生成密钥 ===")
78 dA, PA = sm2_keygen()
79 print(f"Alice 公钥: {PA}")
80 print(f"Alice 私钥 (仅 Alice 知道): {dA}")
81
82 # Alice 签两条消息, 误用相同k
83 print("\n==== 2. Alice 使用相同k签名两条消息 ===")
84 k_fixed = random.randint(1, n-1)
85 m1 = "Alice first message"
86 m2 = "Alice second message"
87
88 r1, s1, _ = sm2_sign(m1, dA, k_fixed)
89 r2, s2, _ = sm2_sign(m2, dA, k_fixed)
90 print(f"签名1: r1={r1}, s1={s1}")
91 print(f"签名2: r2={r2}, s2={s2}")
92
93 # 攻击者恢复私钥
94 print("\n==== 3. 攻击者利用两次签名恢复私钥 ===")

```

```
95 dA_recovered = attack_reuse_k(r1, s1, r2, s2)
96 print(f"恢复的私钥: {dA_recovered}, 是否匹配: {dA == dA_recovered}")
97
98 # 伪造新消息签名
99 print("\n==== 4. 攻击者伪造签名 ===")
100 fake_msg = "This is a fake signed message"
101 r_fake, s_fake, _ = sm2_sign(fake_msg, dA_recovered)
102 print(f"伪造签名: r={r_fake}, s={s_fake}")
103 print("验证伪造签名:", sm2_verify(fake_msg, (r_fake, s_fake), PA))
```