山东大学
网络空间安全学院

创新创业实践
Project5b

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 SM2 签名算法误用导致的私钥恢复攻击 **3**

1.1 攻击 1：已知 k 时的私钥恢复 3

1.2 攻击 2：同一用户两次签名复用 k 3

1.3 攻击 3：两个用户复用相同 k 4

1.4 实验结果 4

A scapy 函数 **5**

1 SM2 签名算法误用导致的私钥恢复攻击

SM2 签名算法的签名步骤如下：

$$
\begin{aligned}
&\text{已知：曲线参数}(p,a,b,G,n)\ \text{私钥}d_A \quad \}P\_A = d\_A \cdot G\\
&\text{消息}M \quad e \quad = \quad H(M)\ \text{随机数}k \quad \in \quad [1, n \ - \ 1]\\
&\text{计算}(x\_1, y\_1) = kG\\
&r = (e + x\_1) \bmod n\\
&s = \big((1 + d\_A) - 1(k - rd\_A)\big) \bmod n
\end{aligned}
$$

因此有：

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

整理得到：

$$s + sd_A \equiv k - rd_A \pmod{n}$$

$$s + d_A(s + r) \equiv k \pmod{n}$$

$$d_A(s + r) \equiv k - s \pmod{n}$$

$$\boxed{d_A \equiv (k - s)(s + r)^{-1} \bmod n}$$

1.1 攻击 1：已知 k 时的私钥恢复

若随机数 k 泄露，根据上式直接恢复：

$$\boxed{d_A = (k - s)(s + r)^{-1} \bmod n}$$

1.2 攻击 2：同一用户两次签名复用 k

用户对不同消息 M_1, M_2 使用相同 k，对应签名 (r_1, s_1) 和 (r_2, s_2)：

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

两式相减：

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

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

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

因此：

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

1.3 攻击 3：两个用户复用相同 k

若用户 A 和用户 B 使用相同 k 分别签名消息 M_A, M_B，则：

$$s_A(1+d_A) \equiv k - r_A d_A \pmod{n}$$
$$s_B(1+d_B) \equiv k - r_B d_B \pmod{n}$$

若 k 泄露或被一方知道，则另一方私钥恢复公式为：

$$\boxed{d_B \equiv (k - s_B)(s_B + r_B)^{-1} \bmod n}$$

1.4 实验结果

三个场景表明 SM2 的安全性高度依赖 k 的保密性和唯一性，一旦 k 泄露或重用，私钥 d_A 可被快速恢复。

```
=== 生成密钥 ===
用户A私钥 dA = 5085193342252927127597807602953344189929384969558544514119844426381848609340 0
用户B私钥 dB = 1578040872146172753300582369691365560933219483510296374677691647469504243133 7

=== 签名并演示漏洞 ===
A签名1: r1=576588356179461504413566885406530093704329127330944311624092771919507537668 43, s1=569074871343706262090850116344800703765842079452439914182660407692305114104 74
A签名2: r2=383493666124143783909896091329008444382702948285147289839319426709035861484 68, s2=590979360530222026861109818784724760445930587134302848039676700919582255926 90

[攻击1] 泄露k恢复私钥：
恢复dA = 5085193342252927127597807602953344189929384969558544514119844426381848609340 0, 是否匹配: True

[攻击2] 两次签名复用k恢复私钥：
恢复dA = 5085193342252927127597807602953344189929384969558544514119844426381848609340 0, 是否匹配: True

[攻击3] 两个用户复用k恢复对方私钥：
恢复dB = 1578040872146172753300582369691365560933219483510296374677691647469504243133 7, 是否匹配: True
+ Code + Markdown
```

A scapy 函数

```python
import hashlib
import random

# ========================
# SM2 椭圆曲线参数 (sm2p256v1)
# ========================
p = int("FFFFFFFEFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF00000000FFFFFFFFFFFFFFFF", 16)
a = int("FFFFFFFEFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF00000000FFFFFFFFFFFFFFFC", 16)
b = int("28E9FA9E9D9F5E344D5AEF20E93E3FBD6B5F6F4C52C9A7A3EB5C3C4F9DCC73E", 16)
n = int("FFFFFFFEFFFFFFFFFFFFFFFFFFFFFFFF7203DF6B21C6052B53BBF40939D54123", 16)
Gx = int("32C4AE2C1F1981195F9904466A39C9948FE30BBFF2660BE1715A4589334C74C7", 16)
Gy = int("BC3736A2F4F6779C59BDCEE36B692153D0A9877CC62A474002DF32E52139F0A0", 16)
G = (Gx, Gy)

# ========== 椭圆曲线运算 ==========
def inverse_mod(x, m):
    return pow(x, -1, m)

def point_add(P, Q):
    if P == (0, 0): return Q
    if Q == (0, 0): return P
    if P == Q:
        lam = (3 * P[0]*P[0] + a) * inverse_mod(2 * P[1], p) % p
    else:
        lam = (Q[1]-P[1]) * inverse_mod(Q[0]-P[0], p) % p
    x = (lam*lam - P[0] - Q[0]) % p
    y = (lam*(P[0]-x)-P[1]) % p
    return (x, y)

def scalar_mul(k, P):
    R = (0, 0)
    while k > 0:
        if k & 1:
            R = point_add(R, P)
        P = point_add(P, P)
        k >>= 1
    return R

# ========== SM3 哈希 ==========
def sm3_hash(msg):
    return int(hashlib.sha256(msg.encode()).hexdigest(), 16)  # 用SHA256代替

# ========== SM2 签名 & 验签 ==========
def sm2_keygen():
    d = random.randint(1, n-1)
    P = scalar_mul(d, G)
```

```
    return d, P

def sm2_sign(msg, d, k=None):
    e = sm3_hash(msg)
    if not k:
        k = random.randint(1, n-1)
    x1, y1 = scalar_mul(k, G)
    r = (e + x1) % n
    s = (inverse_mod(1+d, n) * (k - r*d)) % n
    return (r, s, k)  # 返回k便于测试漏洞

def sm2_verify(msg, sig, P):
    r, s = sig
    e = sm3_hash(msg)
    t = (r + s) % n
    x1, y1 = point_add(scalar_mul(s, G), scalar_mul(t, P))
    R = (e + x1) % n
    return R == r

# ========================
# 场景1: k泄露
# ========================
def attack_leak_k(r, s, k):
    return ((k - s) * inverse_mod(s + r, n)) % n

# ========================
# 场景2: 同用户复用k
# ========================
def attack_reuse_k(r1, s1, r2, s2):
    num = (s2 - s1) % n
    den = (s1 - s2 + r1 - r2) % n
    return (num * inverse_mod(den, n)) % n

# ========================
# 场景3: 两用户复用k
# ========================
def attack_two_users(k, s2, r2):
    return ((k - s2) * inverse_mod(s2 + r2, n)) % n

# ========================
# 验证PoC
# ========================
print("=== 生成密钥 ===")
dA, PA = sm2_keygen()
dB, PB = sm2_keygen()
print(f"用户A私钥 dA = {dA}")
print(f"用户B私钥 dB = {dB}")

```

```
# 测试消息
m1 = "Message1"
m2 = "Message2"

# 签名 (A重用k)
print("\n==== 签名并演示漏洞 ====")
k_fixed = random.randint(1, n-1)
r1, s1, _ = sm2_sign(m1, dA, k_fixed)
r2, s2, _ = sm2_sign(m2, dA, k_fixed)

print(f"A签名1: r1={r1}, s1={s1}")
print(f"A签名2: r2={r2}, s2={s2}")

# ====== 攻击1: 已知k恢复dA ======
print("\n[攻击1] 泄露k恢复私钥: ")
dA_recovered = attack_leak_k(r1, s1, k_fixed)
print(f"恢复dA = {dA_recovered}, 是否匹配: {dA == dA_recovered}")

# ====== 攻击2: 同用户两次签名复用k恢复dA ======
print("\n[攻击2] 两次签名复用k恢复私钥: ")
dA_from_2sigs = attack_reuse_k(r1, s1, r2, s2)
print(f"恢复dA = {dA_from_2sigs}, 是否匹配: {dA == dA_from_2sigs}")

# ====== 攻击3: 两用户复用相同k ======
print("\n[攻击3] 两个用户复用k恢复对方私钥: ")
# B签名用相同k
rB, sB, _ = sm2_sign("OtherMsg", dB, k_fixed)
dB_recovered = attack_two_users(k_fixed, sB, rB)
print(f"恢复dB = {dB_recovered}, 是否匹配: {dB == dB_recovered}")
```