

Pitfalls

姜舜天

2023 年 7 月 30 日

目录

1	延展性导致选择性伪造	2
2	泄露 k 导致泄露 d	2
2.1	ECDSA	2
2.2	Schnorr	2
2.3	SM2	3
3	重用 k 导致泄露 d	3
3.1	ECDSA	3
3.2	Schnorr	4
3.3	SM2	4
4	相同 k 导致泄露 d	4
4.1	ECDSA	4
4.2	Schnorr	5
4.3	SM2	5
5	k, d 相同导致泄露 d	6
5.1	ECDSA	6
5.2	Schnorr	6
5.3	SM2	7
6	具体测试	7
6.1	ECDSA	7

6.1.1	测试代码	7
6.1.2	测试结果	8
6.2	Schnorr	8
6.2.1	测试代码	8
6.2.2	测试结果	9
6.3	SM2	10
6.3.1	测试代码	10
6.3.2	测试结果	11

1 延展性导致选择性伪造

Listing 1: split

```

1 def ecdsa_split(r, s):
2     return r, -s

```

2 泄露 k 导致泄露 d

2.1 ECDSA

在得知 k 的情况下，可以计算 $d = (r)^{-1} \cdot (s * k - e) \bmod n$ ，其中 n 是生成元的阶，代码如下：

Listing 2: ecdsa_leaking_k

```

1 def ecdsa_leaking_k(k, r, s, m):
2     e = hashlib.sha256(m.encode())
3     d = (ecdsa.numbertheory.inverse_mod(r, G1.order()) * (s * k -
4         int(e.hexdigest(), 16))) % G1.order()
5     return d

```

2.2 Schnorr

在得知 k 的情况下，可以计算 $d = (e)^{-1}(s - k) \bmod n$ ，其中 n 是生成元的阶，代码如下：

Listing 3: Schnorr_leaking_k

```

1 def Schnorr_leaking_k(M, R, s, k):
2     e = hashlib.sha256((str(R.x()) + str(R.y()) + M).encode())
3     e = int(e.hexdigest(), 16)
4     d = (ecdsa.numbertheory.inverse_mod(e, G.order()) * (s - k)) %
        G.order()
5     return d

```

2.3 SM2

泄漏 k 的情况下，可以还原出私钥 $d = (k - s)(r + s)^{-1} \bmod n$ ，代码如下：

Listing 4: SM2_leaking_k

```

1 def sm2_leaking_k(r, s, k):
2     d = ((k - s) * ecdsa.numbertheory.inverse_mod(r + s, G.order())
        ) % G.order()
3     return d

```

3 重用 k 导致泄露 d

3.1 ECDSA

在反复使用相同的 k 的情况下，给定对于 m_1, m_2 的签名 $(r_1, s_1), (r_2, s_2)$ 可以计算 $d = (s_2 * r_1 - s_1 * r_2)^{-1} * (s_1 * e_2 - s_2 * e_1) \bmod n$ ，其中 n 是生成元的阶，代码如下：

Listing 5: ecdsa_reusing_k

```

1 def ecdsa_reusing_k(r1, r2, s1, s2, m1, m2):
2     e1 = hashlib.sha256(m1.encode())
3     e2 = hashlib.sha256(m2.encode())
4     e1 = int(e1.hexdigest(), 16)
5     e2 = int(e2.hexdigest(), 16)
6     d = (ecdsa.numbertheory.inverse_mod(s2 * r1 - s1 * r2, G1.order
        ())) * (s1 * e2 - s2 * e1) % G1.order()
7     return d

```

3.2 Schnorr

在反复使用相同的 k 的情况下, 给定对于 m_1, m_2 的签名 $(R_1, s_1), (R_2, s_2)$ 可以计算 $d = (e_1 - e_2)^{-1}(s_1 - s_2) \bmod n$, 其中 n 是生成元的阶, 代码如下:

Listing 6: Schnorr_reusing_k

```
1 def Schnorr_reusing_k(M1, M2, R1, R2, s1, s2):
2     e1 = hashlib.sha256((str(R1.x()) + str(R1.y()) + M1).encode())
3     e1 = int(e1.hexdigest(), 16)
4     e2 = hashlib.sha256((str(R2.x()) + str(R2.y()) + M2).encode())
5     e2 = int(e2.hexdigest(), 16)
6     d = ((s1 - s2) * ecdsa.numbertheory.inverse_mod((e1 - e2), G.
7         order())) % G.order()
8     return d
```

3.3 SM2

在反复使用相同的 k 的情况下, 给定对于 m_1, m_2 的签名 $(r_1, s_1), (r_2, s_2)$ 可以计算 $d = (r_1 - r_2 + s_1 - s_2)^{-1} * (s_2 - s_1) \bmod n$, 其中 n 是生成元的阶, 代码如下:

Listing 7: SM2_reusing_k

```
1 def sm2_reusing_k(r1, r2, s1, s2):
2     d = (ecdsa.numbertheory.inverse_mod((r1 - r2 + s1 - s2), G.
3         order()) * (s2 - s1)) % G.order()
4     return d
```

4 相同 k 导致泄露 d

4.1 ECDSA

在得知另一个人使用了同样的 k 的情况下, 可以计算出另一个人的私钥, 给定对于 (m, m') 分别签名 $(r, s), (r', s')$, 其中 $Sig(d, m) = (r, s)$, 可以计算 $d = (((s' * e - s * e') + s' * r * d) * (s * r')^{-1}) \bmod n$, 其中 n 是生成元的阶, 代码如下:

Listing 8: ecdsa_same_k

```

1 def ecdsa_using_same_k(r, r_, s, s_, m, m_, d): # sk对应的是r, s,
    m
2     e = hashlib.sha256(m.encode())
3     e_ = hashlib.sha256(m_.encode())
4     e = int(e.hexdigest(), 16)
5     e_ = int(e_.hexdigest(), 16)
6     d_ = (((s_ * e - s * e_) + s_ * r * d) * (ecdsa.numbertheory.
        inverse_mod(s * r_, G1.order())) % G1.order())
7     return d_

```

4.2 Schnorr

在得知另一个人使用了同样的 k 的情况下, 可以计算出另一个人的私钥, 给定对于 (m, m') 分别签名 $(R1, s1), (R2, s2)$, 可以计算 $d2 = (e_2)^{-1}(e_1 * d_1 - (s_1 - s_2)) \bmod n$, 其中 n 是生成元的阶, 代码如下:

Listing 9: Schnorr_same_k

```

1 def Schnorr_same_k(M1, M2, R1, R2, s1, s2, d1):
2     e1 = hashlib.sha256((str(R1.x()) + str(R1.y()) + M1).encode())
3     e1 = int(e1.hexdigest(), 16)
4     e2 = hashlib.sha256((str(R2.x()) + str(R2.y()) + M2).encode())
5     e2 = int(e2.hexdigest(), 16)
6     d2 = (ecdsa.numbertheory.inverse_mod(e2, G.order()) * (e1 * d1
        - (s1 - s2))) % G.order()
7     return d2

```

4.3 SM2

在得知另一个人使用了同样的 k 的情况下, 可以计算出另一个人的私钥, 给定对于 (m, m') 分别签名 $(r1, s1), (r2, s2)$, 可以计算 $d2 = (((r1 + s1) * d + (s1 - s2)) * (r2 + s2)^{-1}) \bmod n$, 其中 n 是生成元的阶, 代码如下:

Listing 10: SM2_same_k

```

1 def sm2_same_k(r1, r2, s1, s2, d1):

```

```

2     d2 = (((r1 + s1) * d1 + (s1 - s2)) * ecdsa.numbertheory.
3         inverse_mod((r2 + s2), G.order())) % G.order()
    return d2

```

5 k,d 相同导致泄露 d

5.1 ECDSA

在计算签名时 $k = d$, 给定 $m, (r, s)$ 我们可以计算出私钥 $d = (e * (s - r)^{-1})$, 其中 n 是生成元的阶, 代码如下:

Listing 11: ecdsa_same_kd

```

1 def ecdsa_same_d_k(r, s, m):
2     e = hashlib.sha256(m.encode())
3     e = int(e.hexdigest(), 16)
4     d = (e * ecdsa.numbertheory.inverse_mod(s - r, G1.order())) %
        G1.order()
5     return d

```

5.2 Schnorr

在计算签名时 $k = d$, 给定 $m, (r, s)$ 我们可以计算出私钥 $d = (e + 1)^{-1} * s$, 其中 n 是生成元的阶, 代码如下:

Listing 12: Schnorr_same_kd

```

1 def Schnorr_same_dk(M, R, s):
2     e = hashlib.sha256((str(R.x()) + str(R.y()) + M).encode())
3     e = int(e.hexdigest(), 16)
4     d = (ecdsa.numbertheory.inverse_mod((e+1), G.order()) * s) % G.order()
5     return d

```

5.3 SM2

在计算签名时 $k = d$, 给定 $m, (r, s)$ 我们可以计算出私钥 $d = (-s1) * (r + s - 1)^{-1}$, 其中 n 是生成元的阶, 代码如下:

Listing 13: SM2_same_kd

```
1 def sm2_same_d_k(r, s):
2     d = ((-s) * ecdsa.numbertheory.inverse_mod(r + s - 1, G.order())
3         )%G.order()
4     return d
```

6 具体测试

6.1 ECDSA

6.1.1 测试代码

Listing 14: ECDSA test

```
1 k = random.randint(2 ** 160, 2 ** 161)
2 print('k is:', k)
3
4 d, P = key_generate(G1)
5 d_, P_ = key_generate(G1)
6 print('d is:', d)
7 print('d_ is:', d_)
8
9 m = 'ecdsa'
10 m2 = 'sunshine'
11 k, r, s = ecdsa_sign(m, d, k)
12 d2 = ecdsa_leaking_k(k, r, s, m)
13 print('leaking_k_forged_d is:', d2)
14
15 k2, r2, s2 = ecdsa_sign(m2, d, k)
16 d3 = ecdsa_reusing_k(r, r2, s, s2, m, m2)
17 print('reusing_k_forged_d is:', d3)
```

```

18
19 k_, r_, s_ = ecdsa_sign(m2, d_, k)
20 d_ = ecdsa_using_same_k(r, r_, s, s_, m, m2, d)
21 print('using_same_k_forged_d_is:', d_)
22
23 rr, ss = ecdsa_split(r, s)
24 x = ecdsa_verify(m, r, s, P)
25 y = ecdsa_verify(m, rr, ss, P)
26 print(x == y)
27
28 _k, _r, _s = ecdsa_sign(m, d, d)
29 d4 = ecdsa_same_d_k(_r, _s, m)
30 print(d4)

```

6.1.2 测试结果

```

ecdsa_pitfalls
"/Users/mac/Documents/Py Programs/Pitfalls/venv/bin/python" /Users/mac/Documents/Py Programs/Pitfalls/ecdsa_pitfalls.py
k is: 2867472110602523273084740182847918797535621384721
d is: 2132379562141767522816485099038903718468395720920
d_ is: 610014594073097939838916340897847365488392283083
leaking k forged d is: 2132379562141767522816485099038903718468395720920
reusing k forged d is: 2132379562141767522816485099038903718468395720920
using same k forged d is: 610014594073097939838916340897847365488392283083
True
2132379562141767522816485099038903718468395720920

进程已结束, 退出代码0

```

图 1: ECDSA 执行结果

6.2 Schnorr

6.2.1 测试代码

Listing 15: Schnorr test

```

1 M = 'test'

```

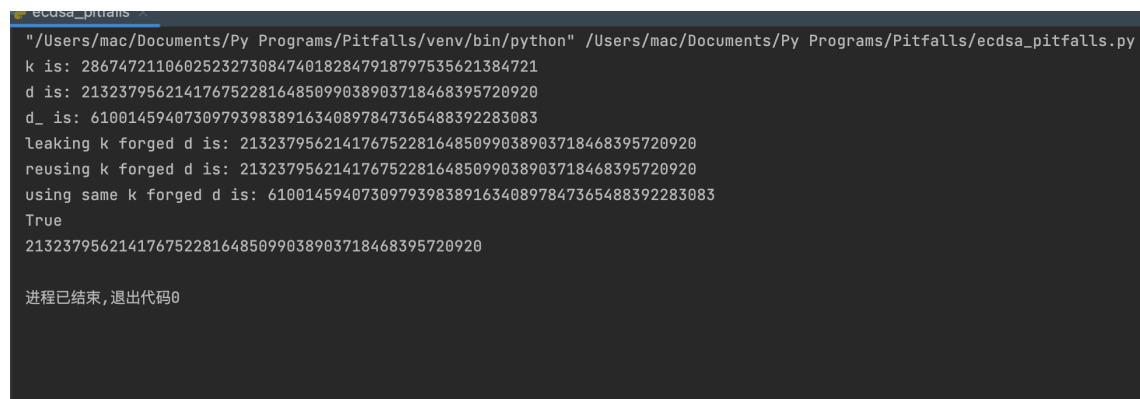


```

2 M2 = 'sunshine_rainbow_pony'
3 k = random.randint(2 ** 160, 2 ** 161)
4 d, P = Schnorr_KeyGen()
5 dd, PP = Schnorr_KeyGen()
6 print('d is: ', d)
7 print('d2 is ', dd)
8 R, s = Schnorr_Sign(M, d, k)
9 RR, ss = Schnorr_Sign(M2, dd, k)
10 SR, Ss = Schnorr_Sign(M, d, d)
11 R2, s2 = Schnorr_Sign(M2, d, k)
12 print(Schnorr_Verify(M, R, s, P))
13 d_ = Schnorr_leaking_k(M, R, s, k)
14 print('leaking_k: ', d_)
15 d2 = Schnorr_reusing_k(M, M2, R, R2, s, s2)
16 print('reusing_k: ', d2)
17 dd = Schnorr_same_k(M, M2, R, RR, s, ss, d)
18 print('same_k: ', dd)
19 Sd = Schnorr_same_dk(M, SR, Ss)
20 print('same_dk: ', Sd)

```

6.2.2 测试结果



```

ecdsa_pitfalls -
"/Users/mac/Documents/Py Programs/Pitfalls/venv/bin/python" /Users/mac/Documents/Py Programs/Pitfalls/ecdsa_pitfalls.py
k is: 2867472110602523273084740182847918797535621384721
d is: 2132379562141767522816485099038903718468395720920
d_ is: 610014594073097939838916340897847365488392283083
leaking k forged d is: 2132379562141767522816485099038903718468395720920
reusing k forged d is: 2132379562141767522816485099038903718468395720920
using same k forged d is: 610014594073097939838916340897847365488392283083
True
2132379562141767522816485099038903718468395720920

进程已结束, 退出代码0

```

图 2: Schnorr 执行结果

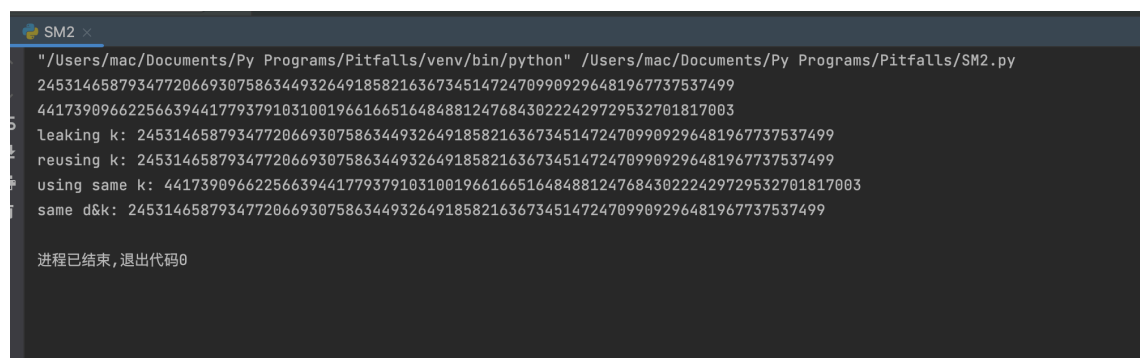
6.3 SM2

6.3.1 测试代码

Listing 16: SM2 test

```
1 message = 'sm2'
2 m = 'sm3'
3 k = random.randint(1, SM2_N - 1)
4
5 t0 = time.time()
6 d, P = SM2_Key_Generate()
7 d2, P2 = SM2_Key_Generate()
8 print(d)
9 print(d2)
10
11 r, s = sm2_sig(message, d, k)
12 r2, s2 = sm2_sig(m, d, k)
13 r22, s22 = sm2_sig(m, d2, k)
14 rr, ss = sm2_sig(m, d, d)
15
16 d_ = sm2_leaking_k(r, s, k)
17 print('leaking_k:', d_)
18 dd = sm2_reusing_k(r, r2, s, s2)
19 print('reusing_k:', dd)
20 d_2 = sm2_same_k(r, r22, s, s22, d)
21 print('using_same_k:', d_2)
22 # sm2 签名验证
23 d__ = sm2_same_d_k(rr, ss)
24 print('same_d&k:', d__)
```

6.3.2 测试结果



```
SM2 <
"/Users/mac/Documents/Py Programs/Pitfalls/venv/bin/python" /Users/mac/Documents/Py Programs/Pitfalls/SM2.py
24531465879347720669307586344932649185821636734514724709909296481967737537499
44173909662256639441779379103100196616651648488124768430222429729532701817003
leaking k: 24531465879347720669307586344932649185821636734514724709909296481967737537499
reusing k: 24531465879347720669307586344932649185821636734514724709909296481967737537499
using same k: 44173909662256639441779379103100196616651648488124768430222429729532701817003
same dk: 24531465879347720669307586344932649185821636734514724709909296481967737537499

进程已结束,退出代码0
```

图 3: SM2 执行结果

参考文献

- [1] https://en.wikipedia.org/wiki/Schnorr_signature
- [2] https://en.wikipedia.org/wiki/Elliptic_Curve_Digital_Signature_Algorithm
- [3] [https://en.wikipedia.org/wiki/SM9_\(cryptography_standard\)](https://en.wikipedia.org/wiki/SM9_(cryptography_standard))