The Angel 👼 & Demon 😈 's Messages Write-up ✍️

Astrageldon

2024-02-11

HAPPY SPRING FESTIVAL 🥳

and fuck the school :(

1 The Angel's Message (SBCTF 2024 Week 4 Crypto)

- 题目描述：（题目名称与题目本身无关）
- 难度：**INSANE** （Easy < Normal < Hard < Harder < Insane < Easy Demon < Medium Demon < Hard Demon < Insane Demon < Extreme Demon）
- 解出人数：2

题目的逻辑是

$$\begin{cases} \boldsymbol{h} = \boldsymbol{x\alpha} - s\boldsymbol{e} \mod p \\ \boldsymbol{g} = \boldsymbol{x\alpha} - s\boldsymbol{e} \mod q \end{cases}$$

其中 $\boldsymbol{x}$ 是分量全为 0 或 1 的 $m \times n$ 型矩阵。给出 $N = pq$，$\boldsymbol{h}$，$\boldsymbol{g}$，$\boldsymbol{e}$，要求出 s。

Step 1.
根据 `magic_e` 函数的逻辑，$\boldsymbol{e}$ 中会出现一些比较特殊的元素 e，可以看作是由较小的私钥 $d < \frac{1}{3}N^{\frac{1}{4}}$ 生成的公钥 e，由于 p，q 相近（$q < p < 2q$），我们可以使用维纳攻击恢复出 $\varphi(N)$，进而得到 p，q。

Step 2. 根据开头的代码

```
assert flag.startswith(b'SBCTF{') and flag.endswith(b'}') and len(flag) < 200
```

可以知道 s 的大小可能超过了 p，q，但是比 N 小，因此应用中国剩余定理得到 $\boldsymbol{c}$，其中

$$\begin{cases} \boldsymbol{c} \equiv \boldsymbol{h} \mod p \\ \boldsymbol{c} \equiv \boldsymbol{g} \mod q \end{cases}$$

Step 3. 最后剩下的问题被称作 AHSSP（Affine Hidden Subset-Sum Problem），解法与这里 完全一致。

Remark. 本题的名称来自 Laur 的同名专辑。

`sol.sage`（Link ）

```python
#sage

from Crypto.Util.number import *
from typing import Tuple, Iterator, Iterable, Optional

# Step 1: Perform Wiener's Attack in order to recover p and q
# Modified a little from: https://github.com/orisano/owiener/blob/master/
    owiener.py

def isqrt(n: int) -> int:
    if n == 0:
        return 0
    x = 2 ** ((int(n).bit_length() + 1) // 2)
    while True:
        y = (x + n // x) // 2
        if y >= x:
            return x
        x = y

def is_perfect_square(n: int) -> bool:
    return isqrt(n) ** 2 == n

def rational_to_contfrac(x: int, y: int) -> Iterator[int]:
    while y:
        a = x // y
        yield a
        x, y = y, x - a * y

def contfrac_to_rational_iter(contfrac: Iterable[int]) -> Iterator[Tuple[int,
    int]]:
    n0, d0 = 0, 1
    n1, d1 = 1, 0
    for q in contfrac:
        n = q * n1 + n0
        d = q * d1 + d0
        yield n, d
        n0, d0 = n1, d1
        n1, d1 = n, d

def convergents_from_contfrac(contfrac: Iterable[int]) -> Iterator[Tuple[int,
    int]]:
    n_, d_ = 1, 0
    for i, (n, d) in enumerate(contfrac_to_rational_iter(contfrac)):
        if i % 2 == 0:
            yield n + n_, d + d_
        else:
            yield n, d
        n_, d_ = n, d

def attack1(e: int, n: int) -> Optional[int]:
    f_ = rational_to_contfrac(e, n)
```

```
    for k, dg in convergents_from_contfrac(f_):
        edg = e * dg
        phi = edg // k
        x = n - phi + 1
        if x % 2 == 0 and is_perfect_square((x // 2) ** 2 - n):
            p = (x + isqrt(x ** 2 - 4 * n)) // 2
            assert n % p == 0
            return p, n // p
    return None


print("Step 1")
with open("output.txt", "r") as f: exec(f.read())

p = 0
i = 0
for E in e:
    i += 1
    print('\r%3d/%3d'%(i,len(e)),end='')
    res = attack1(E, N)
    if res:
        p, q = res
        print()
        break
assert p

# Step 2: Apply Chinese Remainder Theorem

print("Step 2")

hg = vector(crt([x, y], [p, q]) for x, y in zip(h, g))



# Step 3: Perform an attack against AHSSP
# Refer to: https://blog.maple3142.net/2023/04/30/d3ctf-2023-writeups/#d3pack

print("Step 3")

h, g, e = map(vector, [h, g, e])

def find_ortho_fp(*vecs, M = p):
    assert len(set(len(v) for v in vecs)) == 1
    L = block_matrix(ZZ, [[matrix(vecs).T, matrix.identity(len(vecs[0]))], [ZZ(
        M), 0]])
    print("LLL", L.dimensions())
    nv = len(vecs)
    L[:, :nv] *= M
    L = L.LLL()
    ret = []
    for row in L:
        if row[:nv] == 0:
```

```
            ret.append(row[nv:])
    return matrix(ret)

def find_ortho_zz(*vecs, M = p):
    assert len(set(len(v) for v in vecs)) == 1
    L = block_matrix(ZZ, [[matrix(vecs).T, matrix.identity(len(vecs[0]))]])
    print("LLL", L.dimensions())
    nv = len(vecs)
    L[:, :nv] *= M
    L = L.LLL()
    ret = []
    for row in L:
        if row[:nv] == 0:
            ret.append(row[nv:])
    return matrix(ret)

def attack2(v, e, p0):
    F = Zmod(p0)
    v = v.change_ring(F)
    e = e.change_ring(F)
    Mhe = find_ortho_fp(v, e, M = p0)
    assert Mhe * v % p0 == 0
    assert Mhe * e % p0 == 0
    Lx = find_ortho_zz(*Mhe[: m - n], M = p0).T
    Me = find_ortho_fp(e, M = p0)
    assert Me * e % p0 == 0
    alpha = (Me * matrix(F, Lx)).solve_right(Me * v)
    xa = Lx * alpha
    s = (xa - v)[0] / e[0]
    return s

s = attack2(hg, e, N)

unpad = lambda s: s[:s.index(b'\x00')] if b'\x00' in s else s

print(unpad(long_to_bytes(int(s))))
```

Flag: SBCTF{W1en3r_&_AHSSP_a10ng_w1th_CRT_1s_jus7_s0_d5mn_b0r1ng_4nd_sup3r_ez_:p_6y_@Astrageldon_#1145141919810}

2 The Demon's Message (SBCTF 2024 Week 4 Crypto)

- 题目描述：密码学天才 Alice 与 Bob 最近又双叒叕提出了一个新的密钥交换协议，你是 Eve，你需要找到他们协商后的公共密钥并解密对话的内容。（题目名称与题目本身无关）
- 提示：Anshel-Anshel-Goldfeld 密钥交换协议（🔗），可以用暴力穷举或者 Length Based Attack。
- 难度：**MEDIUM DEMON** （Easy < Normal < Hard < Harder < Insane < Easy Demon < Medium Demon < Hard Demon < Insane Demon < Extreme Demon）
- 解出人数：1

SBCTF Crypto 方向魔王关（确信）。本题~~(有点像阅读理解)~~是基于 Heisenberg 群（🔗）对 Anshel-Anshel-Goldfeld 密钥交换协议的实现，这种协议据说可以抵抗量子计算机的破解。本来的想法是使用 Length Based Attack，难度是 **INSANE DEMON** ，但是考虑到那样的话估计就真的没人写出来了，就把 M，L 给降了下来，所以暴力穷举就可以得到公共密钥 $K = K_A = K_B$。（或许改日可以针对此出一个 Revenge 🔥）

基本原理是

Alice's public/private information:

- *Alice's public key* is a tuple of elements $\mathbf{a} = (a_1, \ldots, a_n)$ in G.
- *Alice's private key* is a sequence of elements from $\mathbf{a}$ and their inverses: $a_{i_1}^{\varepsilon_1}, \ldots, a_{i_L}^{\varepsilon_L}$, where $a_{i_k} \in \mathbf{a}$ and $\varepsilon_k = \pm 1$. Based on that sequence she computes the product $A = a_{i_1}^{\varepsilon_1} \ldots a_{i_L}^{\varepsilon_L}$.

Bob's public/private information:

- *Bob's public key* is a tuple of elements $\mathbf{b} = (b_1, \ldots, b_n)$ in G.
- *Bob's private key* is a sequence of elements from $\mathbf{b}$ and their inverses: $b_{j_1}^{\delta_1}, \ldots, b_{j_L}^{\delta_L}$, where $b_{j_k} \in \mathbf{b}$ and $\delta_k = \pm 1$. Based on that sequence he computes the product $B = b_{j_1}^{\delta_1} \ldots b_{j_L}^{\delta_L}$.

Transitions:

- Alice sends a tuple $\overline{\mathbf{a}} = (A^{-1} b_1 A, \ldots, A^{-1} b_n A)$ to Bob.
- Bob sends a tuple $\overline{\mathbf{b}} = (B^{-1} a_1 B, \ldots, B^{-1} a_n B)$ to Alice.

Shared key:

The key shared by Alice and Bob is the group element $K = A^{-1} B^{-1} A B \in G$ called the commutator of A and B.

- Alice computes K as a product $A^{-1} \cdot \left(B^{-1} a_{i_1}^{\varepsilon_1} B\right) \cdots \left(B^{-1} a_{i_L}^{\varepsilon_L} B\right) = A^{-1} B^{-1} A B$.
- Bob computes K as a product $\left(A^{-1} b_{j_L}^{-\delta_L} A\right) \cdots \left(A^{-1} b_{j_1}^{-\delta_1} A\right) \cdot B = A^{-1} B^{-1} A B$.

根据 Heisenberg 群 H 的群乘法运算：

Group structure [edit]

This is indeed a group, as is shown by the multiplication:

$$\begin{bmatrix} 1 & \mathbf{a} & c \\ 0 & I_n & \mathbf{b} \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & \mathbf{a}' & c' \\ 0 & I_n & \mathbf{b}' \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & \mathbf{a}+\mathbf{a}' & c+c'+\mathbf{a}\cdot\mathbf{b}' \\ 0 & I_n & \mathbf{b}+\mathbf{b}' \\ 0 & 0 & 1 \end{bmatrix}$$

and

$$\begin{bmatrix} 1 & \mathbf{a} & c \\ 0 & I_n & \mathbf{b} \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & -\mathbf{a} & -c+\mathbf{a}\cdot\mathbf{b} \\ 0 & I_n & -\mathbf{b} \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & I_n & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

可以将 n 阶 Heisenberg 群 H_n 的 $2n+1$ 个生成元素分成三个部分：

$$H_n = \langle a_1, \cdots, a_n, b_1, \cdots, b_n, c \rangle$$

它们满足：

$$[a_i, b_i] = c,\ [a_i, c] = [b_i, c] = 1,\ [a_i, a_j] = [b_i, b_j] = 1,\ i \neq j$$

其中 $[\cdot, \cdot]$ 是交换子（commutator）：$[g, h] = g^{-1}h^{-1}gh$。
因此最后的 K 一定具有如下的形式：

$$K = c^k, \qquad k \in \mathbb{Z}$$

这里将 K 对应矩阵的右上角分量取出作为公共的密钥。

但是实际解题的时候可以完全不用知道上面这些，根据 `Alice.py` 的内容进行暴力穷举即可。穷举的次数大致是 $\frac{M!}{(M-L)!} \cdot 2^L = 967680$。

Remark. 对于 Length Based Attack 的方法，参见 Heisenberg Groups as Platform for the AAG key-exchange protocol（🔗），该方法的关键在于对 Length Function 的计算，参见 An Efficient Implementation of Braid Groups（🔗）。

`sol.py`（Link🔗）

```python
import re, random, functools, base64, os, itertools
from say_my_name import HeisenbergMatrix, HeisenbergGroup
from __params import N, M, L

from Crypto.Util.number import *
from Crypto.Cipher import AES
from random import randint as ri

pad = lambda s, l: s + bytes([ri(0,31) * bool(i) for i in range(l - len(s))])
unpad = lambda s: s[:s.index(b'\x00')] if b'\x00' in s else s
prod = lambda arr: functools.reduce(lambda a, b: a * b, arr)


def conjugate(w1, w2):
    return w2.inverse() * w1 * w2

def commutator(w1, w2):
    return w1.inverse() * w2.inverse() * w1 * w2

def encode(s):
    return base64.b64encode(s.encode()).decode()

def decode(s):
    return base64.b64decode(s.encode()).decode()

def parsearr(s):
    return [HeisenbergMatrix.fromstr(x) for x in decode(s).split('|')]
```

```python
def parse(s):
    return HeisenbergMatrix.fromstr(decode(s))

def group(s):
    return encode('|'.join(map(repr, s)))


def load_data():
    global set_A, set_B, pub_A, pub_B, enc
    directory = os.path.join(".", "intercepted_data")
    with open(os.path.join(directory, "set_A.txt"), 'r') as f:
        set_A = parsearr(f.read())
    with open(os.path.join(directory, "set_B.txt"), 'r') as f:
        set_B= parsearr(f.read())
    with open(os.path.join(directory, "pub_A.txt"), 'r') as f:
        pub_A = parsearr(f.read())
    with open(os.path.join(directory, "pub_B.txt"), 'r') as f:
        pub_B = parsearr(f.read())
    with open(os.path.join(".", "encrypted"), 'rb') as f:
        enc = f.read()

def oracle(key):
    key = key.c&(2**128 - 1)
    aes = AES.new(long_to_bytes(key, 16), AES.MODE_ECB)
    dec = unpad(aes.decrypt(enc))
    return b'Dear Bob,' in dec, dec

def test(priv_set_A, k_A):
    L_A = []
    priv_A = prod([word ** k for word, k in zip(priv_set_A, k_A)])
    for word, k in zip(priv_set_A, k_A):
        i = set_A.index(word)
        w = pub_B[i] ** k
        L_A.append(w)
    K_A = priv_A * prod(L_A).inverse()
    return K_A, oracle(K_A)

def bruteforce():
    for idx_A in itertools.permutations(range(M), L):
        for k_A in itertools.product([1, -1], repeat = L):
            print('\rProgress: %s %s        ' % (idx_A, k_A), end = '')
            priv_set_A = list(map(set_A.__getitem__, idx_A))
            result, (succ, dec) = test(priv_set_A, k_A)
            if succ:
                return result, dec

load_data()

key, dec = bruteforce()
print('\n\n')
if key:
```

```python
    print("Common Key Found: \033[32m%s\033[0m" % key)
    print()
    print("Message:\n\033[32m%s\033[0m" % dec.decode(encoding = 'utf-8', errors
        = 'ignore'))
else:
    print("\033[31mFailed\033[0m")
```

最后得到 Alice 向 Bob 发送的小秘密：

```
Progress: (0, 1, 2, 5, 8) (1, -1, 1, -1, -1)


Common Key Found: 60;0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
,0,0,0,0,0,0,0,0,0,0,-20733873232394119251999988446841

Message:
Dear Bob,
I am quite confident about our ingenious key exchange scheme so nobody can intercept or decipher our conversation. Here
is the most precious thing I possess, a flag:

SBCTF{WE4K_AAG_1s_Ez_t0_cr5ck_wh1l3_gr0up_th3ory_bas3d_cRypt0l0gy_1s_qu1t3_hardc0r3_:o_!__a_weak3ned_dem0n_l3v3l_6y_Astr
ageldon}

Thanks for participating in SBCTF :)
```

Flag: `SBCTF{WE4K_AAG_1s_Ez_t0_cr5ck_wh1l3_gr0up_th3ory_bas3d_cRypt0l0gy_1s_qu1t3_hardc0r3_:o_!__a_weak3ned_dem0n_l3v3l_6y_Astrageldon}`