Writeup

Crypto

NSA

直接背包格，再调一下平衡系数即可：

$$[x_0,x_1,x_2,x_3,x_4,-1,k]\begin{bmatrix}C_0&0&0&0&0&0&a_0\\0&C_1&0&0&0&0&a_1\\0&0&C_2&0&0&0&a_2\\0&0&0&C_3&0&0&a_3\\0&0&0&0&C_4&0&a_4\\0&0&0&0&0&1&s\\0&0&0&0&0&0&n\end{bmatrix}=$$
$$[C_0x_0,C_1x_1,C_2x_2,C_3x_3,C_4x_4,-1,0]$$

```python
from secrets import randbits
from Crypto.Util.number import *
from hashlib import sha256

nbits = 2048
xbits = 32
'''
n = randbits(nbits)

s = 0
a = []
x = []
for i in range(5):
    ai = randbits(nbits)
    xi = randbits(xbits * i + 128)
    a.append(ai)
    x.append(xi)
    s += ai * xi

s %= n

ans = sha256("".join([str(i) for i in x]).encode()).hexdigest()
flag = f"ZJUCTF{{{ans}}}"
print(flag)
'''

```

```
27
28 n =
   19732624204389519673208573571820091520912596850609767229646060604157629412129634961737878038181324960943830838096171063821865840910241928807220749079924913674503765440130309590221378711193059731249502481420472630356357047121839402956439315825218701944366018002721041552132996353427585524853167908325798672971506764494803773017115500921698211393127775450879311900762238608066672063684450383740400093032166958430745525654722242595133474710483897119823871879293580801342591181168039049260377032266594362880178032628241500533238372811237081475146978394764724934440013595372634700351478225155332362133068711877898611378641
29 s =
   25968545790292770927664439022039737990740822274703473845844930805491973210053546757650612526671426169052862599487679523347480169973953855077248798990354540626902441004340668118824647635652733202266038565469480686593639141386160897997041025319530312635817177211940935304195338733597108954253489362890509953008579795465614477856365008523630015461376982331314525043834068281293242866988185504701331075706296655857609737273305897051800777391003779592347329297992465517975455425494933095449106283240537673407217335317072671448690092359553841225872584507479466178655269973803840975113771278026055850135141677845388049546
30 a =
   [10408175307348227837634927663895071529476747502502185137513974921146145423682039533172426778936636358324323912788471069026035168778419976679080184413445271720845102618347963190848191065559646770751655277489531629085722243398071815514153232748524230263971369717467799896204740816910663354935597598214850905156314156887989538373959310497944747453576532357871371863075761014213384743918508858141163976218524522607305196842635916581744439316075721739180965470299311350393451063367829414121337156065063861518282124855164605728530156876267682012313606474074632758901489079434948149811390226852506471410549293073583094154314,2826956038496019812945946482416890445409353379942737371910195815283014930407953560984050304541881757727133950655970882673250509676504411797139563741479803347728166434909245239520169836699624813956579708192528833011075716373130694486440413179246254805070876775939194011975071311950812218166317804413926814958108259717819968727607935380843933172829365840863170047563542992134283035471587832366000245927906492818175078740706094413165935662430005392989700582452460992968823735599098042761355907055043981216201918780001823362138579793645399017073624740587199777102060208687465901480275883322098554302763493132096511893678,1062574178893100785830388883065914842235401337854298592769659577715234041485401527997184201088342105372035364181218786865321323531235389255597056317755898859910793363334073586448622236411027055736731138984607683677108341528184616399865438075262328225377809455391164380657896819688393471343598725746086361696244738389203215750022930768128013143091276465680872626092823472203862695587378836490255825097232838202085681035254386219122004688438710763010199255687200880598186686280298265895537864898343746074023433404223554747268293243177331602484594119281270588198289285702752514234937996428572156846030362982375296734288746655131313906366456439002698912149464440477062456508241352531036142659786129534047250193815009938938451456170534015851619909583747969431990995384358613514962585570998968613854019699833870623452797126578061923294117627488817953427909692369
```

```
    19471207268114200207207268082845344335732801668784422886161379429856037920579327407784170093716092581196485791058543233924882406459833813410169931297399261627847976866518069425052174316121436628336278159439641483796992170756761511641359869153129085992655015909955296863413697799254057321046624395199748451674781067236038930175399205138285501898400444105867599673894320537167237330013758663076066877713479338691268354899117645445901521662513922052773749395722214090463347248196981868060239786749190266829133470588046958650363148592192357743453783999383200266694850576232013366482175580400126504924153112985830671791571729150276552617782355703038489520393441340713706112576543244470787929038348393779800947179266079330697886117577995873112270474905451612688195298847844226775663235117316316233884464539170131705812430540592713821720299545018169679141879516593359119078495699996765358693970850371204734231497643516531729414382001386014762945378114870937450813766645634470171843375886718590614087224]

M = matrix(QQ, 7, 7)
for i in range(5):
    M[i, i] = 1/2^(128+xbits*i)
    M[i, -1] = a[i]
M[-2, -2] = 1
M[-2, -1] = s
M[-1,-1] = n

ML= M.LLL()
v = ML[0]
X = []
for i in range(5):
    tmp = v[i]*2^(128+xbits*i)
    X.append(abs(tmp))

ans = sha256("".join([str(i) for i in X]).encode()).hexdigest()
flag = f"ZJUCTF{{{ans}}}"
print(flag)
```

FIB I

一开始因为虚拟机连的WebSocketReflector很慢所以只能在Windows下搞，没有sagemath就用python写了...

根据递推式可以写出：

$$f_a = \begin{bmatrix} x_a \\ y_a \end{bmatrix} = M^a \begin{bmatrix} 0 \\ 1 \end{bmatrix}, M = \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix}$$

假设 $M^a = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$ 可以得到 $x_a = b, y_a = d$，因此 b, d 已知，而 M 为对称矩阵，因此 M^a 也为对称矩阵，所以 $c = b$，然后 $det(M) = -1$，因此 $det(M^a) = ad - bc = (-1)^a$，因此可以得到 a 的两种情况。

同理对 b 也是这样，假设 $M^a = \begin{bmatrix} a_1 & b_1 \\ c_1 & d_1 \end{bmatrix}, M^b = \begin{bmatrix} a_2 & b_2 \\ c_2 & d_2 \end{bmatrix}, M^{(a+b)} = \begin{bmatrix} a_3 & b_3 \\ c_3 & d_3 \end{bmatrix}$，那么有 $\begin{bmatrix} a_1 & b_1 \\ c_1 & d_1 \end{bmatrix} \begin{bmatrix} a_2 & b_2 \\ c_2 & d_2 \end{bmatrix} = \begin{bmatrix} a_3 & b_3 \\ c_3 & d_3 \end{bmatrix}$，$b_3 = a_1 b_2 + b_1 d_2, d_3 = c_1 b_2 + d_1 d_2$，我们先假定 a_1 的值得到 b_3, c_3, d_3，而 $c_3 = a_2 c_1 + c_2 d_1$，这样求出 a_2 的值判断行列式是不是1或-1即可，这样就可以求出 M^a，从而求出 $M^{(a+b)}), M^{ka}$，最后一个直接求逆就行

```python
from Crypto.Util.number import *
from gmpy2 import *
from pwn import *
import numpy as np

def matrix_multiply(A, B, mod):
    return np.dot(A, B) % mod

def matrix_power(matrix, power, mod):
    size = len(matrix)
    result = np.eye(size, dtype=int)

    while power:
        if power % 2 == 1:
            result = matrix_multiply(result, matrix, mod)
        matrix = matrix_multiply(matrix, matrix, mod)
        power //= 2

    return result

def matrix_inv(matrix, mod):
    a, b = matrix[0]
    c, d = matrix[1]
    det = (a * d - b * c) % mod
    det_inv = invert(det, mod)
    inv_matrix = [
        [d * det_inv % mod, -b * det_inv % mod],
        [-c * det_inv % mod, a * det_inv % mod]
    ]
    return inv_matrix

context.log_level = "debug"
io = remote("127.0.0.1", 12546)
for i in range(10):
```

```
    io.recvuntil(b'p = ')
    p = int(io.recvline()[:-1].decode())
    io.recvuntil(b'k = ')
    k = int(io.recvline()[:-1].decode())
    io.recvuntil(b'fib(a, p) = ')
    AA = eval(io.recvline()[:-1].decode())
    io.recvuntil(b'fib(b, p) = ')
    BB = eval(io.recvline()[:-1].decode())

    b1 = int(AA[0])
    c1 = b1
    d1 = int(AA[1])
    a1 = (1+b1*c1)*inverse(d1, p)%p
    b2 = int(BB[0])
    c2 = b2
    d2 = int(BB[1])
    b3 = (a1*b2+b1*d2)%p
    c3 = b3
    d3 = (c1*b2+d1*d2)%p

    a2 = (c3 - c2*d1)*inverse(c1, p)%p
    if (a2*d2-b2*c2)%p not in [1, p-1]:
        a1 = (-1+b1*c1)*inverse(d1, p)%p
        b2 = int(BB[0])
        c2 = b2
        d2 = int(BB[1])
        b3 = (a1*b2+b1*d2)%p
        c3 = b3
        d3 = (c1*b2+d1*d2)%p

    res1 = b3*p+d3
    io.recvuntil(b'fib(a + b, p) = ')
    io.sendline(str(res1).encode())

    A = [[a1, b1],[c1,d1]]
    v = [[0],[1]]
    tmp = matrix_power(A, k, p)
    tmp = matrix_multiply(tmp, v, p)
    res2 = int(tmp[0][0])*p+int(tmp[1][0])
    io.recvuntil(b'fib(k * a, p) = ')
    io.sendline(str(res2).encode())

    io.recvuntil(b'if fib(a + c, p) = (0, 1), fib(c, p) = ')
    C = matrix_inv(A, p)
    tmp = matrix_multiply(C, v, p)
    res3 = int(tmp[0][0])*p+int(tmp[1][0])
    io.sendline(str(res3).encode())
```

```
    io.recvline()

```

Seed?

有现成的模板，直接打就行，注意那边传的是int，这里传的是bytes，翻源码知道bytes会加上sha512的值然后转成int，因此把长度改改就行：

```
from z3 import *
import random
from contextlib import contextmanager
from time import perf_counter

# ------------------
# Start of utility functions
# credits: @y011d4
# ------------------

N = 624
M = 397
MATRIX_A = 0x9908B0DF
UPPER_MASK = 0x80000000
LOWER_MASK = 0x7FFFFFFF


def random_seed(seed):
    init_key = []
    if isinstance(seed, int):
        while seed != 0:
            init_key.append(seed % 2**32)
            seed //= 2**32
    else:
        init_key = seed
    key = init_key if len(init_key) > 0 else [0]
    keyused = len(init_key) if len(init_key) > 0 else 1
    return init_by_array(key, keyused)


def init_by_array(init_key, key_length):
    s = 19650218
    mt = [0] * N
    mt[0] = s
    for mti in range(1, N):
        if isinstance(mt[mti - 1], int):
```

```
            mt[mti] = (1812433253 * (mt[mti - 1] ^ (mt[mti - 1] >> 30)) + mti)
% 2**32
        else:
            mt[mti] = 1812433253 * (mt[mti - 1] ^ LShR(mt[mti - 1], 30)) + mti
    i = 1
    j = 0
    k = N if N > key_length else key_length
    while k > 0:
        if isinstance(mt[i - 1], int):
            mt[i] = (
                (mt[i] ^ ((mt[i - 1] ^ (mt[i - 1] >> 30)) * 1664525)) +
init_key[j] + j
            ) % 2**32
        else:
            mt[i] = (
                (mt[i] ^ ((mt[i - 1] ^ LShR(mt[i - 1], 30)) * 1664525))
                + init_key[j]
                + j
            )
        i += 1
        j += 1
        if i >= N:
            mt[0] = mt[N - 1]
            i = 1
        if j >= key_length:
            j = 0
        k -= 1
    for k in range(1, N)[::-1]:
        if isinstance(mt[i - 1], int):
            mt[i] = (
                (mt[i] ^ ((mt[i - 1] ^ (mt[i - 1] >> 30)) * 1566083941)) - i
            ) % 2**32
        else:
            mt[i] = (mt[i] ^ ((mt[i - 1] ^ LShR(mt[i - 1], 30)) * 1566083941))
- i
        i += 1
        if i >= N:
            mt[0] = mt[N - 1]
            i = 1
    mt[0] = 0x80000000
    return mt


def update_mt(mt):
    for kk in range(N - M):
        y = (mt[kk] & UPPER_MASK) | (mt[kk + 1] & LOWER_MASK)
        if isinstance(y, int):
```

```
            mt[kk] = mt[kk + M] ^ (y >> 1) ^ (y % 2) * MATRIX_A
        else:
            mt[kk] = mt[kk + M] ^ LShR(y, 1) ^ (y % 2) * MATRIX_A
    for kk in range(N - M, N - 1):
        y = (mt[kk] & UPPER_MASK) | (mt[kk + 1] & LOWER_MASK)
        if isinstance(y, int):
            mt[kk] = mt[kk + (M - N)] ^ (y >> 1) ^ (y % 2) * MATRIX_A
        else:
            mt[kk] = mt[kk + (M - N)] ^ LShR(y, 1) ^ (y % 2) * MATRIX_A
    y = (mt[N - 1] & UPPER_MASK) | (mt[0] & LOWER_MASK)
    if isinstance(y, int):
        mt[N - 1] = mt[M - 1] ^ (y >> 1) ^ (y % 2) * MATRIX_A
    else:
        mt[N - 1] = mt[M - 1] ^ LShR(y, 1) ^ (y % 2) * MATRIX_A


def temper(state):
    y = state
    if isinstance(y, int):
        y ^= y >> 11
    else:
        y ^= LShR(y, 11)
    y ^= (y << 7) & 0x9D2C5680
    y ^= (y << 15) & 0xEFC60000
    if isinstance(y, int):
        y ^= y >> 18
    else:
        y ^= LShR(y, 18)
    return y


def mt_gen(init_state, *, index=N):
    state = init_state[:]  # copy
    while True:
        index += 1
        if index >= N:
            update_mt(state)
            index = 0
        yield temper(state[index])


def mt_gen_sol(sol, init_state, *, index=N):
    state = init_state[:]  # copy
    twist = 0
    while True:
        index += 1
        if index >= N:
```

```python
            # replace the new state with new symbolic variables
            # this somehow improve the performance of z3 a lot
            update_mt(state)
            next_state = [BitVec(f"__{twist}_state_{i}", 32) for i in range(N)]
            for x, y in zip(state, next_state):
                sol.add(x == y)
            state = next_state
            twist += 1
            index = 0
        yield temper(state[index])


# ------------------
# Start of testing realted things
# ------------------


@contextmanager
def timeit(task_name):
    print(f"[-] Start - {task_name}")
    start = perf_counter()
    try:
        yield
    finally:
        end = perf_counter()
        print(f"[-] End - {task_name}")
        print(f"[-] Elapsed time: {end - start:.2f} seconds")


def test_exact_recovery(nbits):
    print(f"[-] Testing exact recovery with {nbits} bits")
    random.seed(12345)
    outputs = [random.getrandbits(nbits) for _ in range(N * 32 // nbits)]

    state = [BitVec(f"state_{i}", 32) for i in range(N)]
    sol = Solver()
    for s, o in zip(mt_gen_sol(sol, state), outputs):
        sol.add(LShR(s, 32 - nbits) == o)
    with timeit("z3 solving"):
        assert sol.check() == sat

    m = sol.model()
    state = [m.evaluate(s).as_long() for s in state]

    random.setstate((3, tuple(state + [624]), None))
    for v in outputs:
        assert random.getrandbits(nbits) == v
```

```
175
176
177 def test_inexact_recovery(nitems):
178     print(f"[-] Testing inexact recovery with {nitems} items")
179     random.seed(12345)
180     outputs = [(random.randrange(3) + 1) % 3 for _ in range(nitems)]
181
182     state = [BitVec(f"state_{i}", 32) for i in range(N)]
183     sol = Solver()
184     for s, o in zip(mt_gen_sol(sol, state), outputs):
185         sol.add(LShR(s, 30) == o)
186     with timeit("z3 solving"):
187         assert sol.check() == sat
188
189     m = sol.model()
190     state = [m.evaluate(s).as_long() for s in state]
191
192     random.setstate((3, tuple(state + [624]), None))
193     for v in outputs:
194         assert random.randrange(3) == v
195
196
197 def test_exact_seed_recovery():
198     with open("./leak","r") as f:
199         data = f.readlines()
200     outputs = [int(i[2:-1],16) for i in data]
201     print(f"[-] Testing exact seed recovery")
202
    #random.seed(0x87638763DEADBEEF87638763DEADBEEF87638763DEADBEEF87638763DEADBEEF
    DEADBEEF)
203     #outputs = [random.getrandbits(32) for _ in range(N)]
204
205     nseeds = 36
206     seeds = [BitVec(f"seed_{i}", 32) for i in range(nseeds)]
207     state = init_by_array(seeds, len(seeds))
208     sol = Solver()
209     for s, o in zip(mt_gen_sol(sol, state), outputs):
210         sol.add(s == o)
211     with timeit("z3 solving"):
212         assert sol.check() == sat
213
214     m = sol.model()
215     seeds = [m.evaluate(s).as_long() for s in seeds]
216
217     seed = 0
218     for s in seeds[::-1]:
219         seed <<= 32
```

```
220         seed += s
221     print(f"[-] Recovered seed: {seed:x}")
222 
223     random.seed(seed)
224     for v in outputs:
225         assert random.getrandbits(32) == v
226 
227 
228 def test_inexact_seed_recovery_slow(nitems):
229     # ref: https://blog.maple3142.net/2022/11/13/seccon-ctf-2022-
    writeups/#janken-vs-kurenaif
230     print(f"[-] Testing inexact seed recovery (slow) with {nitems} items")
231     random.seed(12345)
232     outputs = [(random.randrange(3) + 1) % 3 for _ in range(nitems)]
233 
234     nseeds = N
235     seeds = [BitVec(f"seed_{i}", 32) for i in range(nseeds)]
236     state = random_seed(seeds)
237     sol = Solver()
238     for s, o in zip(mt_gen_sol(sol, state), outputs):
239         sol.add(LShR(s, 30) == o)
240     with timeit("z3 solving"):
241         assert sol.check() == sat
242 
243     m = sol.model()
244     seeds = [m.evaluate(s).as_long() for s in seeds]
245 
246     seed = 0
247     for s in seeds[::-1]:
248         seed <<= 32
249         seed += s
250     print(f"[-] Recovered seed: {seed}")
251 
252     random.seed(seed)
253     for v in outputs:
254         assert random.randrange(3) == v
255 
256 
257 def test_inexact_seed_recovery_fast(nitems):
258     # ref: https://blog.y011d4.com/20221113-seccon-ctf-writeup#janken-vs-
    kurenaif
259     print(f"[-] Testing inexact seed recovery (fast) with {nitems} items")
260     random.seed(12345)
261     outputs = [(random.randrange(3) + 1) % 3 for _ in range(nitems)]
262 
263     state = [BitVec(f"seed_{i}", 32) for i in range(N)]
264     sol = Solver()
```

```
    sol.add(state[0] == 0x80000000)  # this is really important!!!
    for s, o in zip(mt_gen_sol(sol, state), outputs):
        sol.add(LShR(s, 30) == o)
    with timeit("z3 solving first phase"):
        assert sol.check() == sat

    m = sol.model()
    state = [m.evaluate(s).as_long() for s in state]

    sol = Solver()
    nseeds = N
    seeds = [BitVec(f"seed_{i}", 32) for i in range(nseeds)]
    MT = random_seed(seeds)
    for s, o in zip(MT, state):
        sol.add(s == o)
    with timeit("z3 solving second phase"):
        assert sol.check() == sat

    m = sol.model()
    seeds = [m.evaluate(s).as_long() for s in seeds]

    seed = 0
    for s in seeds[::-1]:
        seed <<= 32
        seed += s
    print(f"[-] Recovered seed: {seed:x}")

    random.seed(seed)
    for v in outputs:
        assert random.randrange(3) == v


if __name__ == "__main__":
    # test_exact_recovery(32)
    # test_exact_recovery(16)
    # test_inexact_recovery(1337)
    test_exact_seed_recovery()
    # test_inexact_seed_recovery_fast(666)

    # too slow, but it should works
    # test_inexact_seed_recovery_slow(666)  # about 10 minutes
    # test_exact_recovery(8)  # idk, maybe forever?
#5a4a554354467b68616861686168612c44306e745f7930755f7468696e6b5f744831355f69735f
5245565f3f495f7468696e6b5f746869735f63616e5f62655f6c6f6f6f6f6f6f6f6f6f6f6e67657
27dc9850e4957bd0b35035956f527f00a03dd952745199de0c82bfa0f399dc48aca6c22947d3157
ae6d8239a276dea3d6829e691926b3f5db15e623989c3dabbbe3
```

把结果转成字符串就行。

Line

用GPT逆向了一下，然后感觉是线性的，因此直接采样线性结构然后解矩阵方程就行：

假设加密的线性矩阵是 A，明文和密钥向量为 x，采样多个明密文对产生的明文和密钥矩阵为 X，密文矩阵为 Y，那么有 $AX^T = Y$，对于一组密文来说有 $Ax = y$，因此首先利用 X, Y 求出 A 然后再求 x 即可。

```
from Crypto.Util.number import *
from tqdm import tqdm
from pwn import *

def encrypt(msg, key):
    io = process("./line")
    io.recvuntil(b'Give me the flag: ')
    io.send(msg)
    io.recvuntil(b'Give me the key: ')
    io.send(key)
    data = io.recvall()
    return data

know1 = bin(bytes_to_long(b'ZJUCTF{'))[2:].zfill(56)
know2 = bin(bytes_to_long(b'}'))[2:].zfill(8)

A = []
B = []
for i in tqdm(range(16*8)):
    bits = '0'*i+'1'*(16*8-i)
    u = know1+bits[:64]+know2+bits[64:]
    u_ = long_to_bytes(int(u, 2),blocksize=24)
    msg = u_[:16]
    key = u_[16:]
    enc = encrypt(msg, key)
    A.append(list(bin(bytes_to_long(bytes(enc)))[2:].zfill(128)))
    B.append([int(i) for i in u])

Am = matrix(GF(2), A).T
Bm = matrix(GF(2), B).T

C = vector(GF(2),
bin(bytes_to_long(bytes.fromhex("D5E398C694C22802B7D68FD5274B87E0")))
[2:].zfill(128))
sol = Am.solve_right(C)

```

```
res = Bm*sol
print(long_to_bytes(int(''.join(list(map(str, list(res)))), 2)))
```

Pseudo

根据next函数可知：

$$K_2 = aK_1 + b \pmod c$$

并且已知两个输出状态 K_{2h}, K_{1h}：

$$K_2 = dK_{2h} + K_{2l}, K_1 = dK_{1h} + K_{1l}$$

因此：

$$dK_{2h} + K_{2l} = a(dK_{1h} + K_{1l}) + b \pmod c$$

这里只有 K_{2l}, K_{1l} 不知道，并且这两个比较小，所以直接二元coppersmith解就行

```
import itertools
from pwn import *

context.log_level = "debug"

def small_roots(f, bounds, m=2, d=None):
    if not d:
        d = f.degree()

    R = f.base_ring()
    N = R.cardinality()

    f /= f.coefficients().pop(0)
    f = f.change_ring(ZZ)

    G = Sequence([], f.parent())
    for i in range(m+1):
        base = N^(m-i) * f^i
        for shifts in itertools.product(range(d), repeat=f.nvariables()):
            g = base * prod(map(power, f.variables(), shifts))
            G.append(g)

    B, monomials = G.coefficient_matrix()
    monomials = vector(monomials)

    factors = [monomial(*bounds) for monomial in monomials]
    for i, factor in enumerate(factors):
        B.rescale_col(i, factor)

```

```
    B = B.dense_matrix().LLL()

    B = B.change_ring(QQ)
    for i, factor in enumerate(factors):
        B.rescale_col(i, 1/factor)

    H = Sequence([], f.parent().change_ring(QQ))
    for h in filter(None, B*monomials):
        H.append(h)
        I = H.ideal()
        if I.dimension() == -1:
            H.pop()
        elif I.dimension() == 0:
            roots = []
            for root in I.variety(ring=ZZ):
                root = tuple(R(root[var]) for var in f.variables())
                roots.append(root)
            return roots

    return []

io = remote("127.0.0.1", 44937)
io.recvuntil(b'a = ')
a = int(io.recvline()[:-1].decode(), 16)
io.recvuntil(b'b = ')
b = int(io.recvline()[:-1].decode(), 16)
io.recvuntil(b'c = ')
c = int(io.recvline()[:-1].decode(), 16)
io.recvuntil(b'd = ')
d = int(io.recvline()[:-1].decode(), 16)
io.recvuntil(b'R0 = ')
R0 = int(io.recvline()[:-1].decode(), 16)
io.recvuntil(b'R1 = ')
R1 = int(io.recvline()[:-1].decode(), 16)

t1 = R0
t3 = R1

P.<x, y> = PolynomialRing(Zmod(c))
f = a*(d*t1+x)+b-d*t3-y
t2, t4 = small_roots(f, (2^256, 2^256), m=3,d=3)[0]
k1 = d*t1+t2

key = k1
res = [t1]
for i in range(19):
    key = (a*key+b)%c
```

```
    res.append(int(key)//d)

for i in range(20):
    io.sendline(hex(res[i])[2:].encode())
io.recvall()

```

ezxor

把前一部分逆了之后得到这样的结果：

```
b'Rom\x02n{e!\xe9^ the mundane worl\xa4 \ts\x10like a starqy sky
admired\xe0from a&bustling city.
[\xcaUC\x94F{Tell_do>e!_Wel\xa3\xafme_to_YJUCTF_2<24!}'
```

后一部分flag直接可以猜出来是：

```
ZJUCTF{Well_done!_Welcome_to_ZJUCTF_2024!}
```

```
cipher =
'0110001110110101101101100000001110110100010100100100011000111111010110001100101
0000111111101001111011000001000110001111111011011001011001101101000100011110111
1100100101110111001110000000101101001001010010111000100100011000111110000000000
1110010111011111000000100100001001110010011011011100111000000010000011100000001O
1110110100111101111100101110001011110010100011100000001011101101100100101000111
0000000100000110111000010010011011000110100011101110011011100010111111101110111
0100011101101011011011000111111101111100011101110111100010110011010001001011000
0100100001001110010010111011101000111111101111011011000110100111101011100011010
0011011010111001110011001100000010111001111000010001010010011001111011100110110
1110110111100101011011100001001010001010111011100111000001100101011001101001000
1100100100011000010110010100100100110111001100101011010011111011010110010101100
0001100011000110011001111101100110000111101110010101110111000010100000100011110
1100000111111001010110'
from Crypto.Util.number import *

def check(x):
    for i in x:
        if ord(i) not in range(32, 128):
            return 0
    return 1

```

```
cipher = cipher[:-1]
cipher = cipher[::-1]
res = ''
for i in range(len(cipher)):
    tmp = int(cipher[i])
    for j in range(i):
        tmp ^= int(res[j])
    res += str(tmp)
    if not check(res):
        res = res[:-1] + ('1' if res[-1] == '0' else '0')
res = long_to_bytes(int(res[::-1],2))
print(res)

```

Shad0wTime

没有限制，所以直接重放就行

```
from pwn import *

context.log_level = "debug"

io = process(["python3", "shad0wtime.py"])
io.recvuntil(b"Enter your option: ")
io.sendline(b"sign_time")
data = eval(io.recvline()[:-1])

io.recvuntil(b"Enter your option: ")
io.sendline(b"verify")
io.recvuntil(b"Enter the message: ")
io.sendline(data["msg"].encode())
io.recvuntil(b"Enter r in hexadecimal form: ")
io.sendline(data["r"].encode())
io.recvuntil(b"Enter s in hexadecimal form: ")
io.sendline(data["s"].encode())
io.recvline()
```

easy pad

unpad函数的漏洞，利用这个可以打出明文（如果没有random），加了random之后先利用flag的字符范围进行筛选，然后重复多次即可消去random的影响

```
from Crypto.Util.number import *
```

```python
from pwn import *

def myxor(byte1, byte2, length):
    return long_to_bytes(bytes_to_long(byte1)^bytes_to_long(byte2), blocksize=length)

def repeat(io, value, times):
    global eee
    cnt = 0
    for i in range(times):
        io.recvuntil(b"1. Decrypt\n2. Quit")
        io.sendline(b'1')
        io.recvuntil(b'Give me ciphertext')
        io.sendline(value.hex().encode())
        res = io.recvline()
        assert b"True" in res or b"False" in res
        if b"False" in res:
            cnt += 1
        eee += 1
    return cnt >= times*2//3

context.log_level = "debug"

#io = process(["python3", "task.py"])
io = remote("127.0.0.1", 64928)

data = bytes.fromhex(io.recvline()[:-1].decode())
#test=io.recvline()
iv, c1, c2 = data[:16], data[16:32], data[32:]
table = b'0123456789abcdef'

eee = 0
length = 1
plain1 = b''
while True:
    char = 0
    while True:
        flag = 0
        if length >= 2:
            tmp = iv + myxor(c1, long_to_bytes(char)+myxor(plain1, long_to_bytes(length)*(length-1), length-1), 16) + c2
        else:
            tmp = iv + myxor(c1, long_to_bytes(char), 16) + c2
        if long_to_bytes(char^length) in table:
            if repeat(io, tmp, 16):
                plain1 = long_to_bytes(char^length) + plain1
                flag = 1
```

```
                break

        if flag == 0:
            char += 1
        if char == 257:
            break

    length += 1
    if length == 17:
        break


length = 1
plain2 = b''
while True:
    char = 0
    while True:
        flag = 0
        if length >= 2:
            tmp = myxor(iv, long_to_bytes(char)+myxor(plain2,
long_to_bytes(length)*(length-1), length-1), 16) + c1
        else:
            tmp = myxor(iv, long_to_bytes(char), 16) + c1
        if long_to_bytes(char^length) in table:
            if repeat(io, tmp, 16):
                plain2 = long_to_bytes(char^length) + plain2
                flag = 1
                break


        if flag == 0:
            char += 1
        if char == 257:
            break

    length += 1
    if length == 17:
        break

print(eee)
print(plain2+plain1)

msg = plain2+plain1
io.recvuntil(b"1. Decrypt\n2. Quit")
io.sendline(b'2')
io.sendline(msg)
io.recvline()
```

```
93
```

FIB II

主体逻辑和FIB I差不多，根据输出可以得到 $M^{g*x}, M = \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix}$ 的值,把n丢到factordb和yafu里面发现是可以完全分解的，其中有几个因子可以使得 M^{g*x} 可相似对角化，并且是smooth的，这样的因子对角化分解后即可把矩阵DLP变成DLP直接解掉，这里比较坑的地方是DLP的数不一定是生成元，所以模取的是乘法群的阶而不是群阶。然后对于不可相似对角化的因子，有些因子是比较小的，对于这些因子首先求出 $GL(n,p)$ 群下的阶，然后在阶里面搜索对应的解，对于不是很小的因子，先计算 $GL(n,p)$ 群阶，然后枚举得到它的阶，然后在阶里面找一些小因子做pohlig-hellman，这样搞有概率CRT之后得到差不多300bits的结果，最后枚举20bits即可得到结果。第二问就比较简单了，在本地生成smooth的素数，当可以相似对角化的时候可以很快求出来

```python
from random import getrandbits
from gmpy2 import invert
from Crypto.Util.number import *
from pwn import *

context.log_level = "debug"

def fib(n, p):
    a, b = 0, 1
    for i in range(n.bit_length() - 1, -1, -1):
        c = a * (2 * b - a)
        d = a * a + b * b
        if (n >> i) & 1:
            a, b = d % p, (c + d) % p
        else:
            a, b = c % p, d % p
    return (a, b)

def recover(b, d, p):
    c = b
    a = (1+b*c)*inverse(d, p)%p
    return matrix(Zmod(p), [[a, b], [c, d]])

ppp = 0x231d5fa471913e79facfd95e9b874e2d499def420e0914fab5c9f87e71c2418d1194066bd8376aa8f02ef35c1926f73a46477cd4a88beae89ba575bb3e1b04271426c6706356dd8cd9aa742d7ad0343f8939bfd2110d45122929d29dc022da26551e1ed7000
M = matrix(Zmod(ppp), [[0,1],[1,1]])
e = 0x10001
M = M^e
```

```
io = remote("127.0.0.1", 43641)
io.sendline(b'ZJUCTF{fibonacci_sequence_mod_n_is_a_kind_of_group}')
io.recvuntil(b'print(f2)')
io.recvline()
io.recvline()
io.recvline()
data = eval(io.recvline()[:-1].decode())
print(data)
b, d = data
Y = recover(b, d, ppp)
'''
MMM = matrix(Zmod(ppp), [[0,1],[1,1]])
a = getrandbits(32)
v = matrix(Zmod(ppp), [[0],[1]])
YYY = MMM^a*v
aaa1, aaa2 = fib(a, ppp)
assert YYY[0, 0] == aaa1 and YYY[1, 0] == aaa2
'''

p_lists = [2**12, 5**4, 15271784978279,
899889935029682511225429150065010811552017719005924136271659168643090431]
order1 = 6144
order2 = 2500
dlogs = []
modulus = [order1, order2]
for p in p_lists:
    MM = M.change_ring(Zmod(p))
    YY = Y.change_ring(Zmod(p))
    if p <=2**12:
        if p==2**12:
            order=order1
        else:
            order = order2
        for i in range(order):
            if MM^i==YY:
                print(i)
                dlog = i
                dlogs.append(i)
                break
    else:
        g, mat = MM.jordan_form(transformation=True)
        assert mat*g*mat.inverse() == MM
        h = mat.inverse() * YY * mat
        y = int(h[0, 0])
        x = int(g[0, 0])
        F = GF(p)
        dlog = discrete_log(F(y), F(x))
```

```
        dlogs.append(dlog)
        print(F(x).multiplicative_order()==p-1)
        modulus.append(F(x).multiplicative_order())
        #modulus.append(p-1)

tmp_prime = 10714146599832792643
order = 21428293199665585288
facs = [8, 11, 37, 8171]
for fac in facs:
    tmp_M = (M^(order//fac)).change_ring(Zmod(tmp_prime))
    tmp_Y = (Y^(order//fac)).change_ring(Zmod(tmp_prime))
    for i in range(fac):
        if tmp_M^i == tmp_Y:
            dlogs.append(i)
            modulus.append(fac)

flag = crt(dlogs, modulus)
print(len(bin(flag))-2)
module = lcm(modulus)

k = 0
while True:
    tmp = long_to_bytes(flag + k*module)
    if b'ZJUCTF' in tmp:
        plain1 = tmp
        io.sendline(str(bytes_to_long(tmp)).encode())
        break
    k += 1
    if k == 2**22:
        print("failed!")
        break

p = 233732552676035055204245032440089798453547968519823616885385341247176515491231670158461029580800000001
facs = [324, 418, 354, 428, 314, 286, 295, 426, 287, 300, 351, 267, 270, 297, 366, 280, 316, 274, 342, 390, 340, 294, 347, 336, 334, 351, 462, 457, 407, 464, 445, 294, 269, 379, 333, 304, 338, 341, 347, 448]
M = matrix(Zmod(p), [[0,1],[1,1]])
io.sendline(str(p).encode())
data = eval(io.recvline()[:-1].decode())
print(data)
b, d = data
Y = recover(b, d, p)
g, mat = M.jordan_form(transformation=True)
assert mat*g*mat.inverse() == M
h = mat.inverse() * Y * mat
```

```
y = int(h[0, 0])
x = int(g[0, 0])
F = GF(p)
dlog = discrete_log(F(y), F(x))
print(plain1+long_to_bytes(dlog))
```

insane pad

zip函数的漏洞，只比较到短序列的长度，利用unpad函数的漏洞先求出flag的pad长度为4，然后依次处理每一块得到五块明文之前的异或关系，然后flag头（7字节）加上pad以及'}'一共12字节，因此只有四字节未知，并且范围是十六进制字符，因此根据哈希值爆破即可

```
from Crypto.Util.number import *
from hashlib import sha256
from pwn import *

flag = b'ZJUCTF{11111111111111111111111111111111111111111111111111111111111111}'

def xor(a, b):
    return long_to_bytes(bytes_to_long(a)^bytes_to_long(b))

def pad(msg):
    need = 16 - (len(msg) & 0xf)
    return msg + need * bytes([need])

def send(io, msg, iv):
    io.recvuntil(b'>')
    io.sendline(b'V')
    io.recvuntil(b'input your message >')
    io.sendline(msg.hex().encode())
    io.recvuntil(b"input your iv >")
    io.sendline(iv.hex().encode())

def i_chunk(data, i):
    return data[16*i:16*i+16]

# context.log_level = "debug"

block = 5

# io = process(["python3", "insane_pad.py"])
io = remote("127.0.0.1", 37671)
io.recvuntil(b'>')
io.sendline(b'G')
```

```
data = bytes.fromhex(io.recvline()[:-1].decode())

io.recvuntil(b'>')
io.sendline(b'H')
hash = bytes.fromhex(io.recvline()[:-1].decode())

print(data)
print(hash)
'''
tmp_chunk = 4
iv = list(b'\x00'*16)
last = 0
tmp_char = list(data)[tmp_chunk*16-1]
for i in range(256):
    check = 0
    iv_char = 0
    tmp_data = list(data)
    tmp_data[tmp_chunk*16-1] = i
    tmp_data = bytes(tmp_data)
    for j in range(256):
        tmp_iv = iv.copy()
        tmp_iv[0] = j
        tmp_iv = bytes(tmp_iv)
        send(io, tmp_data, tmp_iv)
        res = io.recvline()
        if check >= 2:
            break
        if b"Valid!" in res:
            iv_char = j
            check += 1
    if check == 1:
        iv[0] = iv_char
        last = i^79^tmp_char
        break

print(last)
'''
tmp_chunk = 4
tmp_char = list(data)[tmp_chunk*16-1]
last = 4

result = []
for chunk in range(1, 5):
    data1 = data[16*chunk:16*chunk+16]+data[16:]
    new_iv = list(b'\x00'*16)
    for k in range(16):
        tmp_data = list(data1)
```

```
        tmp_data[tmp_chunk*16-1] = last^(79-k)^tmp_char
        tmp_data = bytes(tmp_data)
        for i in range(256):
            tmp_iv = new_iv.copy()
            tmp_iv[k] = i
            tmp_iv = bytes(tmp_iv)
            send(io, tmp_data, tmp_iv)
            res = io.recvline()
            if b"Valid!" in res:
                iv_char = i
                new_iv[k] = iv_char
                break
    result.append(xor(bytes(new_iv),data[16*chunk-16:16*chunk]))

tmp = result[-1]
known = b'}' + bytes([last])*last
known1 = b'ZJUCTF{'
known2 = xor(known, tmp[-last-1:])

table = b"0123456789abcdef"

for a1 in table:
    for a2 in table:
        for a3 in table:
            for a4 in table:
                tmp = known1+bytes([a1,a2,a3,a4])+known2
                res = tmp
                for k in range(len(result)):
                    res+=xor(result[k], tmp)
                res = res[:-4]
                H = sha256(res).digest()
                if H == hash:
                    print(res)
                    exit()

```