

ezrsa

题如其名，真的ez，可以看出是一个二次剩余，直接 `from sympy.ntheory.residue_ntheory import nthroot_mod` 调用函数就行了。(一开始没发现c和r给反了，卡挺久)

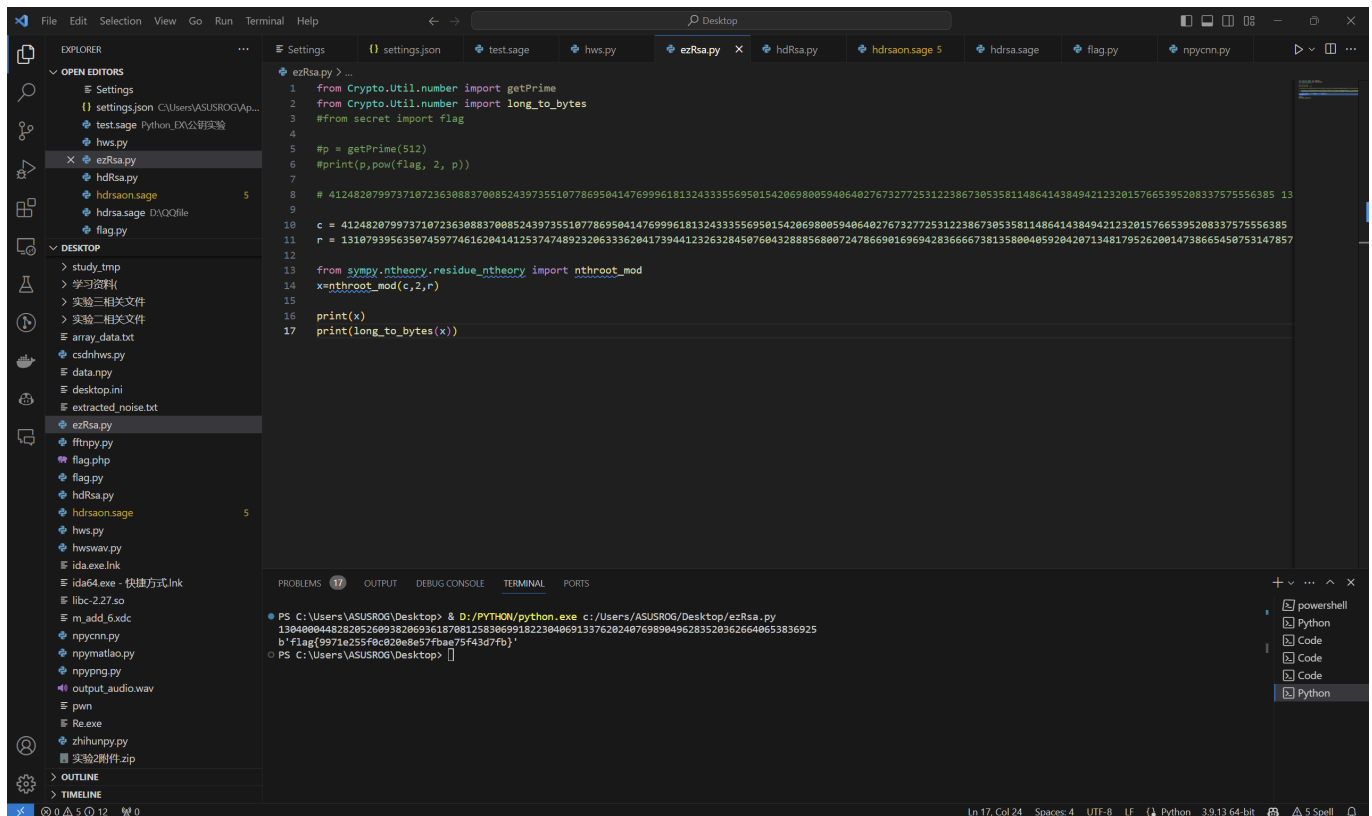
```
from Crypto.Util.number import long_to_bytes
c =
4124820799737107236308837008524397355107786950414769996181324333556950154206980059
406402767327725312238673053581148641438494212320157665395208337575556385
r =
1310793956350745977461620414125374748923206333620417394412326328450760432888568007
2478669016969428366667381358004059204207134817952620014738665450753147857

from sympy.ntheory.residue_ntheory import nthroot_mod
x=nthroot_mod(c,2,r)

print(x)
print(long_to_bytes(x))
```

得到flag：

flag{9971e255f0c020e8e57fbae75f43d7fb}



hdrsa

题如其名，确实hard，观察发现破解的核心在于p的生成方式，搜索发现可以通过 Cheng's 4p - 1 elliptic curve complex multiplication based factorization algorithm破解。然后找到了github上写好的脚本：

https://github.com/pwang00/Cryptographic-Attacks/blob/master/Public%20Key/Factoring/cm_factor.sage

于是现在能分解n了，接着网上找到了一个类似的题目的脚本

<https://angmar2722.github.io/CTFwriteups/2021/idek2021/#destroyed-rsa>

稍作修改就能得到flag了

```
from sage.all import *
from random import choice
from math import gcd
import sys
from Crypto.Util.number import *
from tqdm import tqdm
import time

sys.setrecursionlimit(100000)

def polynomial_xgcd(a, b):
    """
    Computes the extended GCD of two polynomials using Euclid's algorithm.
    :param a: the first polynomial
    :param b: the second polynomial
    :return: a tuple containing r, s, and t
    """
    assert a.base_ring() == b.base_ring()

    r_prev, r = a, b
    s_prev, s = 1, 0
    t_prev, t = 0, 1

    while r:
        try:
            q = r_prev // r
            r_prev, r = r, r_prev - q * r
            s_prev, s = s, s_prev - q * s
            t_prev, t = t, t_prev - q * t
        except RuntimeError:
            raise ArithmeticError("r is not invertible", r)

    return r_prev, s_prev, t_prev

def polynomial_inverse(p, m):
    """
    Computes the inverse of a polynomial modulo a polynomial using the extended
    GCD.
    :param p: the polynomial
    :param m: the polynomial modulus
    :return: the inverse of p modulo m
    """
    g, s, t = polynomial_xgcd(p, m)
```

```

    return s * g.lc() ** -1

def factorize(N, D):
    """
    Recovers the prime factors from a modulus using Cheng's elliptic curve complex
    multiplication method.
    More information: Sedlacek V. et al., "I want to break square-free: The  $4p - 1$ 
    factorization method and its RSA backdoor viability"
    :param N: the modulus
    :param D: the discriminant to use to generate the Hilbert polynomial
    :return: a tuple containing the prime factors
    """
    assert D % 8 == 3, "D should be square-free"

    zmodn = Zmod(N)
    pr = zmodn["x"]

    H = pr(hilbert_class_polynomial(-D))
    Q = pr.quotient(H)
    j = Q.gen()

    try:
        k = j * polynomial_inverse((1728 - j).lift(), H)
    except ArithmeticError as err:
        # If some polynomial was not invertible during XGCD calculation, we can
        factor n.
        p = gcd(int(err.args[1].lc()), N)
        return int(p), int(N // p)

    E = EllipticCurve(Q, [3 * k, 2 * k])
    while True:
        x = zmodn.random_element()

        #print(f"Calculating division polynomial of  $Q\{x\}$ ...")
        z = E.division_polynomial(N, x=Q(x))

        try:
            d, _, _ = polynomial_xgcd(z.lift(), H)
        except ArithmeticError as err:
            # If some polynomial was not invertible during XGCD calculation, we
            can factor n.
            p = gcd(int(err.args[1].lc()), N)
            return int(p), int(N // p)

        p = gcd(int(d), N)
        if 1 < p < N:
            return int(p), int(N // p)

n=33096175288799617332885493596511258888440358453102256111974374065036495822068464
0754584393850796812833007843940336784599719224428969119533284286424077547165101460
4698479807993704196550820691790384976377613333270793745995065747238921438172267518
0680267601322518846740327465821156365587650099729691742190461412805684797779814685
5336939306463059440416150493262973269431000762285579221126342017624118238829230679
9530118973147228019937504549246270742643536920600027585214015443613852313543139818

```

```

3605685558292967081125911301901297054082495113948914639318253241487821418208699929
8397377845534568556100933934481180701997394558264969597606662342898026915506749002
4913262507921073481766817959427999545260685014991002325986586501845658732435251768
3345166425491765570317847294474465862853419534697702341855076162025452817851697206
6618936960223660362493931786389085393392950207048675797593816271435700130995225483
3166258361048026081637453766338848405885753559367461730686553196455721001495155241
3188381377348691712215324849502237269091257254177594361462673394820625290047311824
0712831444072243770979419529210034883903111038448366933374841531126421441232024514
4861687426862974810630891619770548256210997686590975099394053150563253361209294928
3847930960995869695789057029544449427781906344342797264345978489445078701515171567
6537385237767990406742547664321563688829289809321534752244260529319454316532580416
1824387498499233540601252293280439613558940865762385191388682984992490237732377701
0305770791270972541703330906130888058398866646389282863329283996886695377698972231
0954204550783825704710017434214644199415756584929214239679433211393230307782953067
2465296261364463149412588774393560947753375413213316007880426986646320641128969568
9822239744549769598254692287154982824293836848677461735042079071109306991091413531
9635330786253331223459637232106417577225350441291

```

e=65537

```

c=18727536751318634510453486523999469989217090448972541333076711519217253025362539
3062151741036312498277557971553595091826062438445856091864605758318579599363539202
1546256839475689623587025458787609944348132229535034609104476621832003349608211106
1874689979816536338925534736319257625080436241385444582104675575945843944325329482
2553986237695607000569717855942461517564526611106601774100617668231506539201297550
3768340671187845489516999276598898157704926841062878016102610266747785092456495016
9534465221636774117139213904978528065404380450232999976061365869729867160278792919
9239524617160567336634126185042907593427921016129734757065504417112269027028799047
5794509650768358820202611621924756372784452558053393248936264001798187845749576695
7651636334210427318481370847520231353963402776434085824276493487280457081057576419
1987921655276520658100755510986290562980055133376750812535713567917823663134974180
4490024668331091128666812296262398719541250275010713832178163134400792941392549894
1305073151151649812722502097507174731476455226784593349460029529688580846629684409
1612401062566502515356974852161817112538289440970059783116540091633055220150093646
0694381132465187260178682583395122471753860526848616704311484844557654459604951303
0814715643699832755385438774101417742155958568338200337780315828360388931210783788
5491964835073892174406797445622388505256985237867456926792546588756970045576002345
3760353467279062646835966289034179325663832217549768041488780573100668851407763522
0251058446155698817936917756040392339952984287108753249573992190684924907243361454
5319458973155343802539527630971239359995893495205324483418191744545506744253222956
2325069808244579956629002644272659782395400898257337343063631534716062002288419979
2802146835964566122193384854585459609764055248940477792767970908947595403335028728
7833943519423030861868256961619722983499902810335

```

a = [35, 51, 91, 115, 123, 187, 235, 267, 403, 427]

b = [427, 403, 267, 235, 187, 123, 115, 91, 51, 35]

```

def roots_of_unity(e, phi, n, rounds=250):
    # Divide common factors of `phi` and `e` until they're coprime.
    phi_coprime = phi
    while gcd(phi_coprime, e) != 1:
        phi_coprime //= gcd(phi_coprime, e)

    # Don't know how many roots of unity there are, so just try and collect a
    bunch
    roots = set(pow(i, phi_coprime, n) for i in range(1, rounds))

```

```
    assert all(pow(root, e, n) == 1 for root in roots)
    return roots, phi_coprime

start_time = time.time()
for D in b:
    p, q = factorize(n, D)

    assert p*q == n
    # n is prime
    # Problem: e and phi are not coprime - d does not exist
    phi = (p - 1) * (q-1)

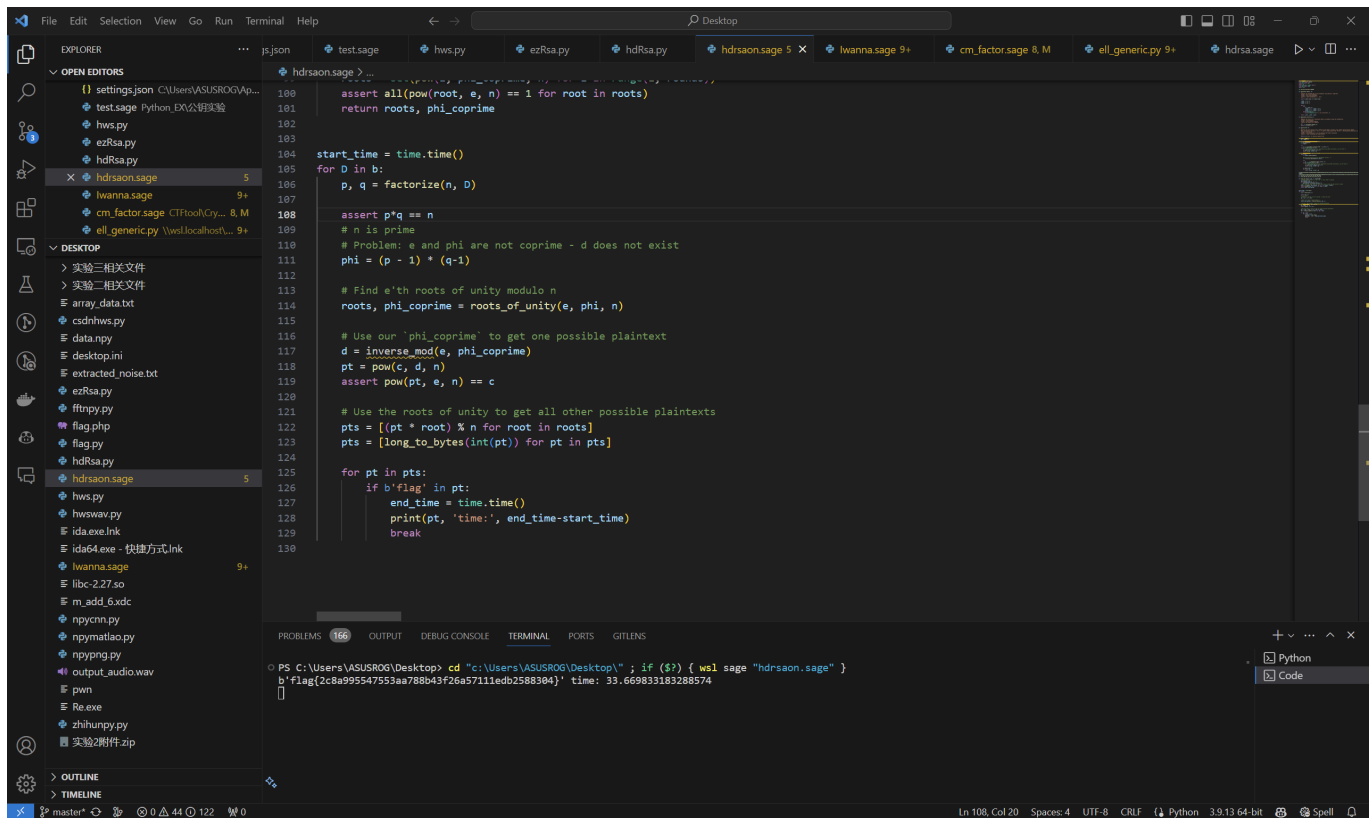
    # Find e'th roots of unity modulo n
    roots, phi_coprime = roots_of_unity(e, phi, n)

    # Use our `phi_coprime` to get one possible plaintext
    d = inverse_mod(e, phi_coprime)
    pt = pow(c, d, n)
    assert pow(pt, e, n) == c

    # Use the roots of unity to get all other possible plaintexts
    pts = [(pt * root) % n for root in roots]
    pts = [long_to_bytes(int(pt)) for pt in pts]

    for pt in pts:
        if b'flag' in pt:
            end_time = time.time()
            print(pt, 'time:', end_time-start_time)
            break
```

```
flag{2c8a995547553aa788b43f26a57111edb2588304}
```



PS：倒着遍历a有奇效