rsa

通过对pq生成的代码进行测试可以发现

q=base+x p=base+y

```python
def get_modulus(bits):
    base = getrandbits(bits)
    p,q = base,base

    while not isPrime(p):
        p+=getrandbits(bits//4)

    while not isPrime(q):
        q+=getrandbits(bits//8)
    print(len(bin(p-base))-2) # 520
    print(len(bin(base))-2) # 2047
    return p,q,p*q
```

通过观察p和base的位数可以发现x只影响p的低520位

又已知 n=p*q =(base+x)(base+y)

对n开方得到的值（base+z）的高位是和pq相等的z对base的影响也不会超过低520位

于是我们考虑用coppersmith的方法来得到p

求p使用的sagemath代码：

```

```

```
n = int(6491001379101188470619474595218807028644407069616577584074380696293122993321600303041472833555957945832536098520748385798067137421510743577978137774182628976294815928791265751078139869584196565823765590651814155402223905159578202066556752793549093193494351919415045985886904164582551789114057922241042158613074337214063536963557041285552410542372370507149885280778670319828096379474371543999352653232299147279464828022096795148554528525701600499786417328803831751290607421590794957076432003935406149487751790274865460732992039502426598097943501541531781504473727341873985640716279668779643926516781210716713798580339413092746276890306735990092803606838446529879807807416225540925134561617431700754244404305825138253268014519757379802140937940828688744638560992176817013407746841844920970636999727176622083816350246626729564250357121595003107383623178298405474857439232298469159175569445890145831753250234804742711253046654443070248738806622616389389695566088172147603319814848309210330216149639665372183936524229364336769822352679815070966389523040722092398005700153923578257053571413662152372745200371073473292238931696371793695497855549418213166578612933410988570880333507084020004211952459895040859438333944509891370867149601785 1)
p4 = 2547744370831027559081382209327784095959044962845658749599898283488707686247076394156616911811786622881075317133033376108015439094351492371803455420759033207994841175191748357042881631524305832372442921275905912309328469010270786601348802659688864244960709323996940034627344202644339739772494169014498850539065763931137238348908096638375158914299133711371781581241185704706584438957258367758671713555319861287744241011998373411979549242684478515448303455862368081960923915181229858276092372692856436791360453049982032902631451947440490712145518182925216515907524714399253185740056169720036830346136127001364499692072 5
p4 = p4-(p4%2^600)
p4 = p4>>600

pbits = 2048
kbits = pbits - p4.nbits()
print(p4.nbits())
p4 = p4 << kbits
PR.<x> = PolynomialRing(Zmod(n))
f = x + p4
roots = f.small_roots(X=2^kbits, beta=0.4)
print((roots[0]+p4))
#
```

```
# p =
2547744370831027559081382209327784095959044962845658749599898 2
```

再来看题目的要求

```
t=pow(2,N//999-20210708)
m=pow(2,t,N)
```

这里让求$2^{2^x}\ mod\ n$

根据费马小定理可知

$2^t = 2^{t\ mod\ \phi(n)}\ mod\ n$

前面我们得到了p和q，求出$\phi(n)$就可以轻松求t了

```
m=pow(2,t,N)phi = (p-1)*(q-1)
t=pow(2,N//999-20210708,phi)
m=pow(2,t,N)
```

完整解题脚本

```
#! /usr/bin/sage
import gmpy2
from hashlib import *
from Crypto.Util.number import *
from sage.all import *

```

```
n = int(64910013791011884706194745952188070286444070696165775840743806962931229933216003030414728335555957945832536098520748385798067137421510743577978137774182628976294815928791265751078139869584196565823765590651814155402223905159578202066556752793549093193494351919415045985886904164582551789114057922241042158613074337214063536963557041285552410542372370507149885280778670319828096379474371543999352653232299147279464828022096795148554528525701600499786417328803831751290607421590794957076432003935406149487751790274865460732992039502426598097943501541531781504473727341873985640716279668779643926516781210716713798580339413092746276890306735990092803606838446529879807807416225540925134561617431700754244404305825138253268014519757379802140937940828688744638560992176817013407746841844920970636999727176622083816350246626729564250357121595003107383623178298405474857439232298469159175569445890145831753250234804742711253046654443070248738806622616389389695566088172147603319814848309210330216149639665372183936524229364336769822352679815070966389523040722092398005700153923578257053571413662152372745200371073473292238931696371793695497855549418213166578612933410988570880333507084020004211952459895040859438333944509891370867149601785l)

p4 = 25477443708310275590813822093277840959590449628456587495998982834887076862470763941566169118117866228810753171330333761080154390943514923718034554207590332079948411751917483570428816315243058323724429212759059123093284690102707866013488026596888642449607093239969400346273442026443397397724941690144988505390657639311372383489080966383751589142991337113717815812411857047065844389572583677586717135553198612877442410119983734119795492426844785154483034558623680819609239151812298582760923726928564367913604530499820329026314519474404907121455181829252165159075247143992531857400561697200368303461361270013644996920725
p4 = p4-(p4%2^600)
p4 = p4>>600

pbits = 2048
kbits = pbits - p4.nbits()
print(p4.nbits())
p4 = p4 << kbits
PR.<x> = PolynomialRing(Zmod(n))
f = x + p4
roots = f.small_roots(X=2^kbits, beta=0.4)
print((roots[0]+p4))
#
```

```
# p =
2547744370831027559081382209327784095959044962845658749599898 2
```

```
phi = (p-1)*(q-1)
t=pow(2,N//999-20210708,phi)
m=pow(2,t,N)

flag="flag{"+md5(long_to_bytes(m)).hexdigest()+"}"
print(flag)
```

flag: `flag{b1939e1c9d198adf9df7341f149ae876}`