cryptohack RSA 部分wp by crumbling

目录

RSA

STARTER

RSA Starter 1

```
print(pow(101,17,22663))
```

RSA Starter 2

用小数实现加密

```
e=65537
q=23
p=17
N=p*q
m=12
print(pow(m,e,N))
```

RSA Starter 3

求欧拉函数。

关于公式，直接贴个wiki吧

[欧拉函数 - 维基百科，自由的百科全书 (wikipedia.org)]

```
p = 857504083339712752489993810777
q = 1029224947942998075080348647219
print((p-1)*(q-1))
```

RSA Starter 4

在已知q，p，e的情况下求私钥d

```
import gmpy2
p = 857504083339712752489993810777
q = 1029224947942998075080348647219
e = 65537
r=(q-1)*(p-1)
d=gmpy2.invert(e,r)
print(d)
```

RSA Starter 5

在已知q，p，e的情况下解密RSA

```
N = 882564595536224140639625987659416029426239230804614613279163
e = 65537
c = 77578995801157823671636298847186723593814843845525223303932
p=857504083339712752489993810777
q=1029224947942998075080348647219
r=(q-1)*(p-1)
d=gmpy2.invert(e,r)
m=pow(c,d,N)
print(m)
```

RSA Starter 6

利用RSA对flag进行签名

```
from hashlib import sha256
flag="crypto{Immut4ble_m3ssag1ng}"
N = 
1521658365483673132763998122413391885589594837407238405084847990898228689073176948660908591885766404607537525316895505874318566439027305807445039023677432490330566347904656623296729776573162532802981405563531600259122757027127144522609491986447540788445998048963800109278857481155414977402895031069511268872385376374323875334978250812198533874675523781937317869934313509178399229956182738974513288002225987338752427329885034064877989790938197971402683717200395322105243121794063255293088000091943650724515072654304071472155336106331195428528985758207988029519963275782952572387475330637199045249130556406105105988580
d = 
1117590121064301426254822247344953309137884826949051885047439968169054728166505931715583169230045319733573572845925939236682330240568538958688367004374468399370912318080515463108851352145697931762801272188153715410723938946606313600733712059991545665975855930067344468926385492133218556270670757366065816499109845787449505485449147406503962192297267158829931584630606984516995945125082104441788663034622902130541034010040153014613541880654434090835510658208908298053365109559419203141167986613425641829224959213544114538446626127942879540872199056465870390378795695816844984149166769049158555016045789335053633424268
s=sha256(flag.encode())
b=s.hexdigest()
b=int(b,16)
enc=pow(b,d,N)
print(hex(enc))
```

PRIMES PART 1

Factoring

http://factordb.com/index.php?id=

对于不大的n可以用这个网站分解。

不过大部分时候都没法直接分解，所以只能用来做一些非常基础的RSA分解题目，或是其他用途。

Inferius Prime

n200位，同样网站分解。

```
from Crypto.Util.number import*
n = 742449129124467073921545687640895127535705902454369756401331
e = 3
ct = 39207274348578481322317340648475596807303160111338236677373
p=752708788837165590355094155871
q=986369682585281993933185289261
r=(p-1)*(q-1)
d=inverse(e,r)
flag=pow(ct,d,n)
flag=long_to_bytes(flag)
print(flag)
```

Monoprime

n不由p、q两个素数相乘得到，而直接是一个素数。

这是一个非常危险的做法，虽然“无法分解”，但因为欧拉函数就是n-1，所以反而更加容易求解。

```python
from Crypto.Util.number import inverse,  long_to_bytes
from gmpy2 import mpz
n = 
171731371218065444125482536302245915415603318380280392385291836472299752747934607246477508507827284075763910264995326010251268493630501989810855418416643352631102434317900028697993224868629935657273062472544675693365930943308086634291936846505861203914449338007760990051788980485462592823446469606824421932591
e = 65537
ct = 
161367550346730604451454756189028938964941280347662098798775466019463375610700074840105776873791605070092554650190486030367121011578171525759600774739890458414593857709994072516290998135846956596662071379067305011746842247628316996977338024343628757374524136260758515864509435302781735938531030576289086798942
r=n-1
d=inverse(e,r)
flag=pow(ct,d,n)
flag=long_to_bytes(flag)
print(flag)
```

Square Eyes

n很大，但是因数是2个p。开根后求欧拉函数即可。

```
from Crypto.Util.number import long_to_bytes
import gmpy2
n =
5358608080440095500291771357081680162014513431473135653710144590277434917394228854430847057207314097137755279937196825836691648738068420432884398280717899706947590808421622539552595905522830477287828129468451603348017820880681544530219367217102690509858050546920967387773217961533840248976155944934530681383412036737495140945460002536319029916171978475845196941521227654069821335265949286852323819347421521958613802212243708581287369759591768616510443703785390939901983362985729445127385708393965885900968132177911918959413804648033776027792406631338349523293168623995819505905880063712213341282154091976032369425976747567282122321340565627163991550801088811059527681891937288274846673493780911000682244046847016747823992003731924330627676228412640554260353497690181172996205548039024904323396005664322467958181674609161806473941691576472456035556927356308621487154287912427647994698969247534705398570807671700527839182731803048353183881770896742316409103377437897509792162025732267942403327978928682763094002539259322238955307141696481165690135816431923419318007852547150832945263259802472192183641188778648920681859055874109771527379363107347122769566631921824876724746511032400041733810412379068494374906096523957488684342967534491
e = 65537
ct =
2225028859741824295009483898405634152915347268913545739073295125564396328109219279052204867278074366680359293024427542259527866024922504480203412177336464729822862223388605660761619777860956759445522323914812787820193462839009596771670266368302520670487597202516718110586475697244955479409668850256298070791712183716445280535622323966742837453101322424923672741846678451745144668341325899713880670769805631885133336611658194624288372105753421010363569741893933900974036144344915076724592549696380327768974176745774877757555399649150357319884999837264350050078508760002322924585545774377394273134536714929566681882196006333259309817481624559650932226481731347775715276815913661647113073555108893160520641460896467728696107266716966992211579858343256636614000348314424312091234787780782558468305222263909641198187849033302004887052127655691634955718514593555203989282142062850808839548818888668509262455490889283862560453598662919522224935145694435885396500780651530829377030371611921181207362217397805303962112100190783763061909945889717878397740711340114311597934724670601992737526668932871436226135393872881664511222789565256059138002651403875484920711316522536260604255269532161594824301047729082877262812899724246757871448545439896
p=gmpy2.iroot(n,2)[0]
p=231486675219980977208571688277907713376624837163484354773605674093550261691659344469498096645955237708538972031037591069839851132640490574169081911667200085032759516257389756660190291723776531706024403735795932925765306677739514076472227577564378672160951931742013232788960272945177926078818618552646005257724607452594403011569309432552409156857185523341922302647803557991790378160263307054224840000865423620840069581585503463959418623839259420337300300046063603083797762554362064405294417118592468115866527460284184960201454415130375354753809625621089206999290229006779019885089365093543856607356945682166313826531 07
r=p*(p-1)
d=gmpy2.invert(e,r)
m=pow(ct,d,n)
print(long_to_bytes(m))
```

Manyprime

n很大，但因为因数较多各个因数较小，所以可以网站可以直接分解。根据公式求出欧拉函数就可以成功求解。

```
from Crypto.Util.number import long_to_bytes
import gmpy2
n =
580642391898843192929563856870897799650883152718761762932292482252152591279871421569162037190419036435041797739880389529593674485555792234900969402019055601781662044515999210032698275981631376651117318677368742867687180140048715627160641771118040372573575479330830092989800730105573700557717146251860588802509310534792310748898504394966263819959963273509119791037525504422606634640173277598774814099540555569257179715908642917355365791447508751401889724095964924513196281345665480688029639999472649549163147599540142367575413885729653166517595719991872223011969856259344396899748662101941230745601719730556631637
e = 65537
ct =
320721490534624434149993723527322977960556510750628354856260732098109692581338409999983376131354918370047625150454728718467998870322344980985635149656977787964380651868131740312053755501594999166365821315043312308622388016666802478485476059625888033017198083472976011719998333985531756978678758897472845358167730221506573817798467100023754709109274265835201757369829744113233607359526441007577850111228850004361838028842815813724076511058179239339760639518034583306154826603816927757236549096339501503316601078891287408682099750164720032975016814187899399273719181407940397071512493967454225665490162619270814464
p=
[9282105380008121879,9303850685953812323,9389357739583927789,10336650220878499841,10638241655447339831,11282698189561966721,11328768673634243077,11403460639036243901,11473665579512371723,11492065299277279799,11530534813954192171,11665347949879312361,
12132158321859677597,12834461276877415051,12955403765595949597,12973972336777979701,13099895578757581201,13572286589428162097,14100640260554622013,14178869592193599187,14278240802299816541,14523070016044624039,14963354250199553339,15364597561881860737,15669758663523555763,15824122791679574573,15998365463074268941,16656402470578844539,16898740504023346457,17138336856793050757,17174065872156629921,17281246625998849649]
r=1
for i in p:
    r=r*(i-1)
d=gmpy2.invert(e,r)
m=pow(ct,d,n)
print(long_to_bytes(m))
```

PUBLIC EXPONENT

Salty

e为1，虽然加密时很快，但解密时更快。

```
from Crypto.Util.number import long_to_bytes,getPrime
n =
110581795715958566206600392161360212579669637391437097703685154237017351570464767725324182051199901920318211290404777259728923614917211291562555864753005179326101890427669819834642007924406862482343614488768256951616086287044725034412802176312273081322195866046098595306261781788276570920467840172004530873767
e = 1
ct =
44981230718212183604274785925793145442655465025264554046028251311164494127485
m=pow(ct,e,n)
print(long_to_bytes(m))
```

Modulus Inutilis

n很大，但e=3很小，有m^3=ct<n的可能

```
from Crypto.Util.number import long_to_bytes
import gmpy2
n =
17258212916191948536348548470938004244269544560039009244721959293554822498047075403658429865201816363311805874117705688359853941515579440852166618074161313773416434156467811969628473425365608002907061241714688204565170146117869742910273064909154666642642308154422770994836108669814632309362483307560217924183202838588431342622551598499747369771295105890359290073146330677383341121242366368309126850094371525078749496850520075015636716490087482193603562501577348571256210991732071282478547626856068209192987351212490642903450263288650415552403935705444809043563866466823492258216747445926536608548665086042098252335883
e = 3
ct =
243251053617903760309941844835411292373350655973075480264001352919865180151222189820473358411037759381328642957324889519192337152355302808400638052620580409813222660643570085177957
m=gmpy2.iroot(ct,3)[0]
print(long_to_bytes(m))
```

Everything is Big

e很大时可以考虑wiener's Attack

捡了个脚本

太久了稍微找了下，应该是来自[BugkuCTF RSA(wiener's attack)_bugkectf rsa_ProboxDu的博客-CSDN博客]()

```
from __future__ import print_function
import libnum


def continued_fractions_expansion(numerator, denominator):  # (e,N)
    result = []

    divident = numerator % denominator
    quotient = numerator // denominator
    result.append(quotient)
```

```python
    while divident != 0:
        numerator = numerator - quotient * denominator

        tmp = denominator
        denominator = numerator
        numerator = tmp

        divident = numerator % denominator
        quotient = numerator // denominator
        result.append(quotient)

    return result


def convergents(expansion):
    convergents = [(expansion[0], 1)]
    for i in range(1, len(expansion)):
        numerator = 1
        denominator = expansion[i]
        for j in range(i - 1, -1, -1):
            numerator += expansion[j] * denominator
            if j == 0:
                break
            tmp = denominator
            denominator = numerator
            numerator = tmp
        convergents.append((numerator, denominator))  # (k,d)
    return convergents


def newtonSqrt(n):
    approx = n // 2
    better = (approx + n // approx) // 2
    while better != approx:
        approx = better
        better = (approx + n // approx) // 2
    return approx


def wiener_attack(cons, e, N):
    for cs in cons:
        k, d = cs
        if k == 0:
            continue
        phi_N = (e * d - 1) // k
        # x**2 - ((N - phi_N) + 1) * x + N = 0
        a = 1
        b = -((N - phi_N) + 1)
        c = N
        delta = b * b - 4 * a * c
        if delta <= 0:
            continue
        x1 = (newtonSqrt(delta) - b) // (2 * a)
        x2 = -(newtonSqrt(delta) + b) // (2 * a)
```

```
        if x1 * x2 == N:
            return [x1, x2, k, d]


if __name__ == "__main__":
    n =
0xb8af3d3afb893a602de4afe2a29d7615075d1e570f8bad8ebbe9b5b9076594cf06b6e7b30905b6
420e950043380ea746f0a14dae34469aa723e946e484a58bcd92d1039105871ffd63ffe64534b7d7
f8d84b4a569723f7a833e6daf5e182d658655f739a4e37bd9f4a44aff6ca0255cda5313c3048f56e
ed5b21dc8d88bf5a8f8379eac83d8523e484fa6ae8dbcb239e65d3777829a6903d779cd2498b255f
cf275e5f49471f35992435ee7cade98c8e82a8beb5ce1749349caa16759afc4e799edb12d299374d
748a9e3c82e1cc983cdf9daec0a2739dadcc0982c1e7e492139cbff18c5d44529407edfd8e75743d
2f51ce2b58573fea6fbd4fe25154b9964d
    e =
0x9ab58dbc8049b574c361573955f08ea69f97ecf37400f9626d8f5ac55ca087165ce5e1f459ef6f
a5f158cc8e75cb400a7473e89dd38922ead221b33bc33d6d716fb0e4e127b0fc18a197daf856a706
2b49fba7a86e3a138956af04f481b7a7d481994aeebc2672e500f3f6d8c581268c2cfad484515 8f7
9c2ef28f242f4fa8f6e573b8723a752d96169c9d885ada59cdeb6dbe932de86a019a7e8fc8aeb077
48cfb272bd36d94fe83351252187c2e0bc58bb7a0a0af154b63397e6c68af4314601e29b07caed30
1b6831cf34caa579eb42a8c8bf69898d04b495174b5d7de0f20cf2b8fc55ed35c6ad157d3e7009f1
6d6b61786ee40583850e67af13e9d25be3
    c =
0x3f984ff5244f1836ed69361f29905ca1ae6b3dcf249133c398d7762f5e27791917469429398914
4c9d25e940d2f66058b2289c75d1b8d0729f9a7c4564404a5fd4313675f85f31b47156068878e236
c5635156b0fa21e24346c2041ae42423078577a1413f41375a4d49296ab17910ae214b45155c4570
f95ca874ccae9fa80433a1ab453cbb28d780c2f1f4dc7071c93aff3924d76c5b4068a0371dff8253
1313f281a8acadaa2bd5078d3ddcefcb981f37ff9b8b14c7d9bf1accffe7857160982a2c7d9ee01d
3e82265eec9c7401ecc7f02581fd0d912684f42d1b71df87a1ca51515aab4e58fab4da96e154ea6c
dfb573a71d81b2ea4a080a1066e1bc3474
    expansion = continued_fractions_expansion(e, n)
    cons = convergents(expansion)
    p, q, k, d = wiener_attack(cons, e, n)
    m = pow(c, d, n)
    print(libnum.n2s(m))
```