密码学

#

1.

```
from Crypto.Util.number import getPrime, bytes_to_long
flag = open("flag", "rb").read()
# 第一次生成 p, q 和 n
assert(e<100000)
p = getPrime(1024)
q = getPrime(1024)
n = p * q
m = bytes_to_long(flag)
c = pow(m, e, n)
print(c, n)
# 输出另一个已知的明文加密结果，用于攻击
print(pow(294, e, n))
# 第二次生成新的 p 和 n，使用原来的 q
p = getPrime(1024)
n = p * q  # 注意这里使用的是新的 p 和原来的 q
m = bytes_to_long("BJD" * 32)
c = pow(m, e, n)
print(c, n)
'''
output:
c1=126416356178037461503322326463545962927078614802002075371991411836244383037571205700967412480202366669657557980096565477386163990253001230437662555185961493489304445998206752300464233730530516319325572308490834268594901837323037517440048741830625948568703186142899916759800635483164994869089232096275638715548756127020791005670186989929358182061090875681660973923141057175554829261410305056395717088762131671121879625844840653215457275941351753692339259225077949996073235369768241831629233850056699304034488534651414058468359198429084697875473417523654718924952043076445861613932287760420155341479138883383162441691201n1=135087741044602097433067140345467041372476273449811334618019534797360170214017258188084628983759947673756277494948396719445438224030599780738131224414076125306581689429878202567865830069470017117492301935423705709507055301679217028356271224014752510390007750173816339002224747273968237086950631362461156526222597696345913094217612695482609844261488246412850107309832153775092550112987378276216111580329764200116625478545156105979556288980735696841582256783334745439203265328934468498081128374766843900309764720539050698555222978506880269607011865434281398437839076243172747969262488295434134647541272088430703310630373816312688258064695181663703873520354757756771636157307594543439135636159708819673324077099012356377189361841989302263037618765171012086771073110060657280142204779660006209640566160586769998789769433190638366490850853775772732147923715487752045940978870788985984638924401415779745449392682478189379366070131008081697586750422645685477640316284314147279221685809984946958004030433124066435276376674663184736695423261692186653664230435790033884866341676426634958966072821558083319023511885001979609056722070465796470527645794118143056891375198608809164672720567786414427589401350164008087403871445081563580679552150 18
```

```
c2=979153370552535153498477459720877329811204688208387543826122582132404214848454954722487086658061408795223805022202997613522014736983452121073860054851302343517756732701026667062765906277626879215457936330799698812755973057557620930172778859116538571207100424990838508255127616637334499680058645411786925302368790414768248611809358160197554369255458675450109457987698749584630551177577492043403656419968285163536823819817573531356497236154342689914525321673807925458651854768512396355389740863270148775362744448115581639629326362342160548500035000156097215446881251055505465713854173913142040976382500435185442521721
n2=128062109030613683690543095751593603740223447745474593452169071281939575929380718158659540732875325459473706718383721448065397538294843560649193572856233052096006805709752246392143968051243508627721592723627787680368446347609176127087217873201593184324560508062277844350911611199826139873032559955431653954266580594621100564313925175487174478980849151676611723629842512016886394696522834523077128213988570164875907949965444688267056003322085352014433222672987471175288829859553752464248126164783271823994617099788934640932451355301354300078422233893602128034398508676151211480500348877675846936087763232522332542610471'''
```

学过信安的小朋友都知道
可以通过素数分解可以得到p，q值
分解出p，q，然后看print(pow(294, e, n))的值求e
通过得到的e解出
flag：BJD{p_is_common_divisor}
#

2. 看不懂啊，上网搜索一下，发现这个是一种题型
   找到解密代码

```
from gmpy2 import *
from Crypto.Util.number import *
from Crypto.PublicKey import RSA


# 公钥提取
with open("pub.key","r",encoding="utf-8") as file:
    text=file.read()
key=RSA.import_key(text)
e=key.e
n=key.n
print(e)
print(n)
q=
p=
with open("flag.enc","rb") as file:
    c=file.read()
d=invert(e,(p-1)*(q-1))
flag=pow(int.from_bytes(c,byteorder="big"),d,n)
print(long_to_bytes(flag))
```

得到flag：flag{decrypt_256}
#

3. 发现这引用了一个库叫binascii，这是个ascii的编码模块，可以在二进制和ASCII之间相互转化，然后还发现里面引用了个flag的文件使用如下代码进行解密

```
import gmpy2
```

```
from Crypto.Util.number import long_to_bytes

n =
[201296153524917654993401129431883171805487615978613008473058271415104656196705368446345582464392303716588369281030634328702457071803559071942848615109060712653524095794410481010849959239621485270973707054520705770987802462828200655737110156642919913720851570169012091141910685742086803977100428428359404284519495006076136346826841132087666940287892757485282542877057595284989863064942678171983406582418730248003360139462948916875910134149352378212918051232859053357627198237716478533788928688960784245722329343609406729624368495239155633287799421345044995688661352666280784852320982082370367241214818350357312013834233122165015562784996446641374941470061382384106014952445123490167716000909901401892658109487984009724854341198053306683197661702367622562506785400331701879404172361255600847157906042889811779058799105568138040826338276184162571441587908747807277196816038490991995801098366936836078850528885594612415951311884774799865642252141498029521264667585069093788376400057166757438141914437282421179801858680467482456460612259248328657580068523212827382008779181166387805782738637978788296276329006607223124881492046826474165408601107263821107544544784369104984726248575939329085311707286840686184079389581621595686952328923142 1 ,
299445375153979533615209227741241926055247113067538353037034788904141635107774605597983343130212163893562518749177920076382992258210188496485206738137867724528228095465711298163102072328832397713241228848049934189583094600094063428721731890084492379595774691141589912024334767105813562438157137628024784543902738083774306851571100954967279663080012541075179675593840197342798618409972391762542360690014535445597860639159700711300878111239120443122195355138806639138313587903766504390836606118311562051138737931068802558821144220257469864033550669965679095817106477464639942804447009228673977547486284259674882325303303,
257034378556001352151857784535839254469127316616040541841638832722655033230162957003572531053011467266678974974355325799749514783545704155542214017785361047372961543160563140394491163864943236684837498331478005574033684895422731694890802220093689039936584982639055675167986842114626070697966134346611481869018920162820659161909204433787561672508098724835017122257820043969699969830574239426071743141325984212691697225182244782488368810764846398373430793246369971451998350348333677430799353612761499909978759053136427752144860463813686196385518922927877831376222614335289152693334267689473585529197409018609826791807915]
```

```
c =
[191314326612179084702623384212996919985261577905835441567419812388221585639885202259869152345700373838881127244083929181139427219941255050147275459461333073297817476003028295882480429226357143910334319304111805450853164380843179273487052419275704327578929850913960449500854624295754400606529672538450413983996484423400429708144155719040576670281575129710793846017248163080786318444801102017873435830738151867717904777120400511571803188044221204720076367220639893153208635806313306471169938197777506841509504162980852614788411776816778672368656662073918470464839540292134953736134906906874730819301484618304257176145691534189843322663823516007202987573382695679998295810791025005595833492246020255492474314412217001835511745245947201713361464224241147984936906148286057027986369242562152605686280842513526760854485583335831407120068734044251285657527871298664157301245672940266059733960944377114534718126828505072892599351870489900541618725000330458123070144470515741279078702792681071099864619146713055071360076589823439235015396581159506065675371127830800519337093629612479077268943377341470364570391074219389847180008132146905521170933984639250070652367014525902426785836821690217648981478967947222734336303542854191511837816301203 1 ,
187150650716480400179672112972311065381399850876853585556505670577155505864648147636836882990378971828450075785714013590612137776451144146429030770035681555084658196285537471732442359365868124454400954507551543576467370870716058119841634165902783526054333623279490482437225562629799094882024425303075058193715947479362238352335869454235222569387010023706463820978461050149817633077292346757377022521551308371548768318858886691504188850880893245348925061997244867834462673367898727821378955525093535833058801449477141100098931341621853823099926044356647774361975873123172248627238135109744930874502817554524287461944 46,
228228456122485829313848044746331926247491884763014877011247270312854903259218779728996559261519970985787900827176643346203232849858034096887126018966970751855715783659242497325733436293163983107258482410312348652258253166615236387439648274456175813365540641036444217498322700550186092782087126071186100883012061705688351452579870960174408813599946559833863579427512314916549893358015994503236388061352492191302334120943965714596233221346857340286379692057181241820081481708623426228033822116162278951682936380508471565212173903618326402612086875652377019628414227184987900320219096615039006119546935171681953918379 7]
sum_n = 1
rev_n = []
ni = []
m = 0
for i in n:
    sum_n = sum_n * i
for i in n:
    temp = sum_n//i
    ni.append(temp)
    rev_n.append(gmpy2.invert(temp,i))
for i in range(len(n)):
    m += c[i]*rev_n[i]*ni[i] % sum_n
p = gmpy2.iroot(m%sum_n,4)[0]
print(p)



ee1 = 42
ee2 = 3
```

```
ce1 =
45722651786340123946960815003059322528810481841378247280642868553607692149509126
96287258303714246139880668948914174149497483688234150523425532568321909216305284
34616323384425290115023789311403561117569327128225168140231660689025694582999333
91973504078898958921809723346229893913662577294963528318424676803942288386430172
43088030761974818686389005011393457382050557092810901784264759826663434444718234
78493677145646863418710075058867283937511470335568892176046473556285575022083644
12269944908011305064122941446516990168924709684092200183860653173856272384
ce2 =
13908468332333567158469136439932325992349696889129103935400760239319454409539725
38974705921383523837304789919821112868937404972957814687530923196293655440328788
29999678403462166952084245827397770342610795503959180484210868439270094524799360
45850799096750074359160775182238980989229190157551197830879877097703347301072427
14947499180386832576996733235695086351850496548656546405977045145855774494973528
21317279560562792928006942038661672702689884373899457031170706044889992477501395
68614939965885211276821987586882908159585863514561191905040244967655444219603287
21440501488799423825927071635537806972676095332002582815 8
tmp =
86407877807860983516777956598254075768407045069785430900517174281341496344746255
49990127189609250816215714874447255289824240374190521948407209498098911348548712
22612682162490991065015935449289960707882463387
n1  =
15911581555796798614711625288508309704791837516232122410440958830726078821069050
40401282089626007175138043699271063836429465817357110159693160579750971283962247
93688502512064197480900597524273036117600046213782264312269836657468377790562715
30181865648115862947527212787824629516204832313026456390047768174765687040950636
53048054901440127905434609803039510038700411157427881374963098672470626365516628
95862304539759537737919454085894846793718541134577581574922412251809070902351163
25034822993748409011554673180494306003272836905082473475046277554085737627846557
24036769621408127634507105557816929906070679419277 6825039
def doit(ee,n,ce):
    k=0
    while True:
        x=ce+k*n
        if gmpy2.iroot(x,ee)[1]:
            return gmpy2.iroot(x,ee)[0]
        k=k+1
e1=doit(ee1,n1,ce1)
e2=doit(ee2,n1,ce2)-tmp
print(e1)
print(e2)


q1 =
12758731925343664356931214205855970681549721166108386659253421707931049726036530
74260956612811037100423927754538661746574049855390667416841960201378404729501023
80232067786400322600902938984916355631714439668326671310160916766472897536055371
47407608977947237291303704015335643752880892291148404946034208883 4871
q2 =
11440118822747958468088404615129970465692053616876713291658918235758346105333638
69961237832949325665677736954266894474103119694564585747311875129748682970926386
77515283584994416382872450167046416573472658841627690987228528798356894803559278
30870263528853765319209851496608916812371085467963867142497822 1959513
```

```
c1 =
26273997575393028169094278432125233903590619684634071323751038236455768537954349
87650744488257993421943326811811297700460750181220334219832278877196101120282306
03166527303021036386350781414447347150383783816869784006598225583375458609586450
85460286256902257167204915880987476381283404425741919963121752736704662488883775
53112150811733865238060867832661983902890972311681726923266536573935225617419479
51887577156666663584249108899327053951891486355179939770150550995812478327735917
00619457441251881929930378324388696245539978360122922771878708178539101042403050
99374036003514141761381247051680022886206648092700461 24
c2 =
73955911292288766490308196166858218992048326849957577249244508129774707878222663
87122334722132760470911599176362617225218345404468270014548817267727669872896838
10645152039280649746657690706329560374666000318844017091949015725082930817331071
53189257716431050648826207461712664998590490380169021625992614090509071408233529
90750298239508355767238575709803167676810456559665476121149766947851911064706646
50670539709162664871368451178045695545355202046090963801613412459043842573882682
86947739605142219101094739414514714316379031822057387381094297364250256213083008
95473186381826756650667842656050416299166317372707709596

x1=gmpy2.gcd(e1,(p-1)*(q1-1))
x2=gmpy2.gcd(e2,(p-1)*(q2-1))
d1=gmpy2.invert(e1//x1,(p-1)*(q1-1))
d2=gmpy2.invert(e2//x2,(p-1)*(q2-1))
m1 = pow(c1,d1,p*q1)
m2 = pow(c2,d2,p*q2)

d1_ = gmpy2.invert(7,(q1-1))
d2_ = gmpy2.invert(7,(q2-1))
b1 = pow(m1,d1_,q1)
b2 = pow(m2,d2_,q2)
m=(b1*q2*gmpy2.invert(q2,q1)+b2*q1*gmpy2.invert(q1,q2))%(q1*q2)
print(long_to_bytes(gmpy2.iroot(m,2)[0]))
```

得到结果：
de1ctf{9b10a98b-71bb-4bdf-a6ff-f319943de21f}
#

4. 先搜索，发现这个sage是一种算法，使用如下代码解密

```
#sage
P=43753
R.<y> = PolynomialRing(GF(P))
```

```
N=34036*y^177 + 23068*y^176 + 13147*y^175 + 36344*y^174 + 10045*y^173 +
41049*y^172 + 17786*y^171 + 16601*y^170 + 7929*y^169 + 37570*y^168 + 990*y^167 +
9622*y^166 + 39273*y^165 + 35284*y^164 + 15632*y^163 + 18850*y^162 + 8800*y^161
+ 33148*y^160 + 12147*y^159 + 40487*y^158 + 6407*y^157 + 34111*y^156 +
8446*y^155 + 21908*y^154 + 16812*y^153 + 40624*y^152 + 43506*y^151 + 39116*y^150
+ 33011*y^149 + 23914*y^148 + 2210*y^147 + 23196*y^146 + 43359*y^145 +
34455*y^144 + 17684*y^143 + 25262*y^142 + 982*y^141 + 24015*y^140 + 27968*y^139
+ 37463*y^138 + 10667*y^137 + 39519*y^136 + 31176*y^135 + 27520*y^134 +
32118*y^133 + 8333*y^132 + 38945*y^131 + 34713*y^130 + 1107*y^129 + 43604*y^128
+ 4433*y^127 + 18110*y^126 + 17658*y^125 + 32354*y^124 + 3219*y^123 +
40238*y^122 + 10439*y^121 + 3669*y^120 + 8713*y^119 + 21027*y^118 + 29480*y^117
+ 5477*y^116 + 24332*y^115 + 43480*y^114 + 33406*y^113 + 43121*y^112 +
1114*y^111 + 17198*y^110 + 22829*y^109 + 24424*y^108 + 16523*y^107 + 20424*y^106
+ 36206*y^105 + 41849*y^104 + 3584*y^103 + 26500*y^102 + 31897*y^101 +
34640*y^100 + 27449*y^99 + 30962*y^98 + 41434*y^97 + 22125*y^96 + 24314*y^95 +
3944*y^94 + 18400*y^93 + 38476*y^92 + 28904*y^91 + 27936*y^90 + 41867*y^89 +
25573*y^88 + 25659*y^87 + 33443*y^86 + 18435*y^85 + 5934*y^84 + 38030*y^83 +
17563*y^82 + 24086*y^81 + 36782*y^80 + 20922*y^79 + 38933*y^78 + 23448*y^77 +
10599*y^76 + 7156*y^75 + 29044*y^74 + 23605*y^73 + 7657*y^72 + 28200*y^71 +
2431*y^70 + 3860*y^69 + 23259*y^68 + 14590*y^67 + 33631*y^66 + 15673*y^65 +
36049*y^64 + 29728*y^63 + 22413*y^62 + 18602*y^61 + 18557*y^60 + 23505*y^59 +
17642*y^58 + 12595*y^57 + 17255*y^56 + 15316*y^55 + 8948*y^54 + 38*y^53 +
40329*y^52 + 9823*y^51 + 5798*y^50 + 6379*y^49 + 8662*y^48 + 34640*y^47 +
38321*y^46 + 18760*y^45 + 13135*y^44 + 15926*y^43 + 34952*y^42 + 28940*y^41 +
13558*y^40 + 42579*y^39 + 38015*y^38 + 33788*y^37 + 12381*y^36 + 195*y^35 +
13709*y^34 + 31500*y^33 + 32994*y^32 + 30486*y^31 + 40414*y^30 + 2578*y^29 +
30525*y^28 + 43067*y^27 + 6195*y^26 + 36288*y^25 + 23236*y^24 + 21493*y^23 +
15808*y^22 + 34500*y^21 + 6390*y^20 + 42994*y^19 + 42151*y^18 + 19248*y^17 +
19291*y^16 + 8124*y^15 + 40161*y^14 + 24726*y^13 + 31874*y^12 + 30272*y^11 +
30761*y^10 + 2296*y^9 + 11017*y^8 + 16559*y^7 + 28949*y^6 + 40499*y^5 +
22377*y^4 + 33628*y^3 + 30598*y^2 + 4386*y + 23814
S.<x> = R.quotient(N)
```

```
C=5209*x^176 + 10881*x^175 + 31096*x^174 + 23354*x^173 + 28337*x^172 +
15982*x^171 + 13515*x^170 + 21641*x^169 + 10254*x^168 + 34588*x^167 +
27434*x^166 + 29552*x^165 + 7105*x^164 + 22604*x^163 + 41253*x^162 + 42675*x^161
+ 21153*x^160 + 32838*x^159 + 34391*x^158 + 832*x^157 + 720*x^156 + 22883*x^155
+ 19236*x^154 + 33772*x^153 + 5020*x^152 + 17943*x^151 + 26967*x^150 +
30847*x^149 + 10306*x^148 + 33966*x^147 + 43255*x^146 + 20342*x^145 + 4474*x^144
+ 3490*x^143 + 38033*x^142 + 11224*x^141 + 30565*x^140 + 31967*x^139 +
32382*x^138 + 9759*x^137 + 1030*x^136 + 32122*x^135 + 42614*x^134 + 14280*x^133
+ 16533*x^132 + 32676*x^131 + 43070*x^130 + 36009*x^129 + 28497*x^128 +
2940*x^127 + 9747*x^126 + 22758*x^125 + 16615*x^124 + 14086*x^123 + 13038*x^122
+ 39603*x^121 + 36260*x^120 + 32502*x^119 + 17619*x^118 + 17700*x^117 +
15083*x^116 + 11311*x^115 + 36496*x^114 + 1300*x^113 + 13601*x^112 + 43425*x^111
+ 10376*x^110 + 11551*x^109 + 13684*x^108 + 14955*x^107 + 6661*x^106 +
12674*x^105 + 21534*x^104 + 32132*x^103 + 34135*x^102 + 43684*x^101 + 837*x^100
+ 29311*x^99 + 4849*x^98 + 26632*x^97 + 26662*x^96 + 10159*x^95 + 32657*x^94 +
12149*x^93 + 17858*x^92 + 35805*x^91 + 19391*x^90 + 30884*x^89 + 42039*x^88 +
17292*x^87 + 4694*x^86 + 1497*x^85 + 1744*x^84 + 31071*x^83 + 26246*x^82 +
24402*x^81 + 22068*x^80 + 39263*x^79 + 23703*x^78 + 21484*x^77 + 12241*x^76 +
28821*x^75 + 32886*x^74 + 43075*x^73 + 35741*x^72 + 19936*x^71 + 37219*x^70 +
33411*x^69 + 8301*x^68 + 12949*x^67 + 28611*x^66 + 42654*x^65 + 6910*x^64 +
18523*x^63 + 31144*x^62 + 21398*x^61 + 36298*x^60 + 27158*x^59 + 918*x^58 +
38601*x^57 + 4269*x^56 + 5699*x^55 + 36444*x^54 + 34791*x^53 + 37978*x^52 +
32481*x^51 + 8039*x^50 + 11012*x^49 + 11454*x^48 + 30450*x^47 + 1381*x^46 +
32403*x^45 + 8202*x^44 + 8404*x^43 + 37648*x^42 + 43696*x^41 + 34237*x^40 +
36490*x^39 + 41423*x^38 + 35792*x^37 + 36950*x^36 + 31086*x^35 + 38970*x^34 +
12439*x^33 + 7963*x^32 + 16150*x^31 + 11382*x^30 + 3038*x^29 + 20157*x^28 +
23531*x^27 + 32866*x^26 + 5428*x^25 + 21132*x^24 + 13443*x^23 + 28909*x^22 +
42716*x^21 + 6567*x^20 + 24744*x^19 + 8727*x^18 + 14895*x^17 + 28172*x^16 +
30903*x^15 + 26608*x^14 + 27314*x^13 + 42224*x^12 + 42551*x^11 + 37726*x^10 +
11203*x^9 + 36816*x^8 + 5537*x^7 + 20301*x^6 + 17591*x^5 + 41279*x^4 + 7999*x^3
+ 33753*x^2 + 34551*x + 9659
# factor(N)
p,q = N.factor()
p,q = p[0],q[0]
# print(p)
phi=(pow(P,65)-1)*(pow(P,112)-1)
e = 65537
d = inverse_mod(e,phi)
m = C^d
# print(m)
print("".join([chr(c) for c in m.list()]))
```

运行代码得到
watevr{RSA_from_ikea_is_fun_but_insecure#k20944uehdjfnjd335uro}
#

5. 素数挺多啊，当rsa解吧

```
from gmpy2 import *
import math
def extended_gcd(a, b):
    """扩展欧几里得算法，返回gcd及其系数"""
    if a == 0:
        return b, 0, 1
    else:
        gcd, x1, y1 = extended_gcd(b % a, a)
        x = y1 - (b // a) * x1
```

```
        y = x1
        return gcd, x, y

def mod_inverse(x, m):
    """计算x对m的逆元"""
    gcd, inverse, _ = extended_gcd(x, m)
    if gcd != 1:
        raise ValueError(f"{x}没有在模{m}下的逆元")
    else:
        return inverse % m  # 确保结果为正数

x = 65537
m =
2217990918*2338725372*2370292206*2463878386*2706073948*2794985116*2804303068*292
3072266*2970591036*3207148518*3654864130*3831680818*3939901242*4093178560*427842
8892
try:
    inv = mod_inverse(x, m)
    print(f"{x}\n在模\n{m}\n下的逆元是：\n{inv}")
except ValueError as e:
    print(e)

def decrypt(ciphertext, d, n):
    plaintext = pow(ciphertext, d, n)
    return plaintext

# RSA参数
n =
172900660705949795710096633812142013204595698513585023686512455142135382299699156580649925581673235868950889339228353538040557726389802513282 61
e = 65537    # 公开指数e
d =
136192544098365970544162544550529407837527503830853233011864322612693775580814032050304765205677588092279876620086944474425442545817376043827 3  # 私有指数d，用于解密
ciphertext =
143220384337616554046783935681585378497835894814635210756948026546110488988786051446637504106557346754233282562131144229299940372407529953635 95
# 解密消息
plaintext = decrypt(ciphertext, d, n)

# 将解密后的数字转换为字符串
chars = []
for i in range((len(bin(plaintext)) - 1) // 4):
    byte = (plaintext >> (8 * i)) & 255
    chars.append(chr(byte))
message = ''.join(chars)

print(f"解密后的消息是: \n{message}")
```

运行得到galf，欸我flag怎么反了
运行rev(flag),直接执行反序让它立正
得到flag{us4_s1ge_t0_cal_phl}