Crypto01

题目：

```
from Crypto.Util.number import *
from secret import flag

p = getPrime(512)
q = getPrime(512)
n = p * q
d = getPrime(299)
e = inverse(d,(p-1)*(q-1))
m = bytes_to_long(flag)
c = pow(m,e,n)
hint1 = p >> (512-70)
hint2 = q >> (512-70)

print(f"n = {n}")
print(f"e = {e}")
print(f"c = {c}")
print(f"hint1 = {hint1}")
print(f"hint2 = {hint2}")

n =
65318835751656706270462803918372182811096669561139853833192009681234356986381524661604904085035483519298788284835759796179173585004238691057332447589167439506386243352011548441011828732868691543989256629925692290088144403935880664585931943707422938295860559274669263630591393175387389387981929391774213395003
e =
40738963799387627677248718814260921636491770341278750841486515130562842134876396915106681433734276005332410415984785584884091334455816402584507178231273998519376915363193650533215442952274343814099450187672503465016280527554101223321817109358581483535333734940968961773897326303002987203525415266163296607215
c =
28858301095788879003830568949705095027466057921892765321643055383309726369907606901866332954652632036004551285284813812187678314978562231120323470819722044516158810710408496414616381570397632550468546302235826400628265458069027801013266037490695637948933487646553291637002155559023268928639502489285322508063
hint1 = 624859718207126357681
hint2 = 810475217500119132950
```

发现是论文题，利用coppersmith来进行约束求解

参考了该论文https://eprint.iacr.org/2023/367.pdf

在github中找到了类似的代码，加以修改得：

```
import time
time.clock = time.time

debug = True

strict = False
```

```python
helpful_only = True
dimension_min = 7

def helpful_vectors(BB, modulus):
    nothelpful = 0
    for ii in range(BB.dimensions()[0]):
        if BB[ii,ii] >= modulus:
            nothelpful += 1

def matrix_overview(BB, bound):
    for ii in range(BB.dimensions()[0]):
        a = ('%02d ' % ii)
        for jj in range(BB.dimensions()[1]):
            a += '0' if BB[ii,jj] == 0 else 'X'
            if BB.dimensions()[0] < 60:
                a += ' '
        if BB[ii, ii] >= bound:
            a += '~'

def remove_unhelpful(BB, monomials, bound, current):
    if current == -1 or BB.dimensions()[0] <= dimension_min:
        return BB

    for ii in range(current, -1, -1):

        if BB[ii, ii] >= bound:
            affected_vectors = 0
            affected_vector_index = 0

            for jj in range(ii + 1, BB.dimensions()[0]):

                if BB[jj, ii] != 0:
                    affected_vectors += 1
                    affected_vector_index = jj

            if affected_vectors == 0:
                #print ("* removing unhelpful vector", ii)
                BB = BB.delete_columns([ii])
                BB = BB.delete_rows([ii])
                monomials.pop(ii)
                BB = remove_unhelpful(BB, monomials, bound, ii-1)
                return BB

            elif affected_vectors == 1:
                affected_deeper = True
                for kk in range(affected_vector_index + 1, BB.dimensions()[0]):
                    if BB[kk, affected_vector_index] != 0:
                        affected_deeper = False

                if affected_deeper and abs(bound - BB[affected_vector_index,
affected_vector_index]) < abs(bound - BB[ii, ii]):
                    BB = BB.delete_columns([affected_vector_index, ii])
                    BB = BB.delete_rows([affected_vector_index, ii])
                    monomials.pop(affected_vector_index)
                    monomials.pop(ii)
                    BB = remove_unhelpful(BB, monomials, bound, ii-1)
```

```
                    return BB
    return BB


def boneh_durfee(pol, modulus, mm, tt, XX, YY):

    PR.<u, x, y> = PolynomialRing(ZZ)
    Q = PR.quotient(x*y + 1 - u)
    polZ = Q(pol).lift()

    UU = XX*YY + 1

    gg = []
    for kk in range(mm + 1):
        for ii in range(mm - kk + 1):
            xshift = x^ii * modulus^(mm - kk) * polZ(u, x, y)^kk
            gg.append(xshift)
    gg.sort()

    monomials = []
    for polynomial in gg:
        for monomial in polynomial.monomials():
            if monomial not in monomials:
                monomials.append(monomial)
    monomials.sort()

    for jj in range(1, tt + 1):
        for kk in range(floor(mm/tt) * jj, mm + 1):
            yshift = y^jj * polZ(u, x, y)^kk * modulus^(mm - kk)
            yshift = Q(yshift).lift()
            gg.append(yshift)

    for jj in range(1, tt + 1):
        for kk in range(floor(mm/tt) * jj, mm + 1):
            monomials.append(u^kk * y^jj)

    nn = len(monomials)
    BB = Matrix(ZZ, nn)
    for ii in range(nn):
        BB[ii, 0] = gg[ii](0, 0, 0)
        for jj in range(1, ii + 1):
            if monomials[jj] in gg[ii].monomials():
                BB[ii, jj] = gg[ii].monomial_coefficient(monomials[jj]) *
monomials[jj](UU,XX,YY)

    if helpful_only:
        BB = remove_unhelpful(BB, monomials, modulus^mm, nn-1)
        nn = BB.dimensions()[0]
        if nn == 0:
            print ("failure")
            return 0,0

    if debug:
        helpful_vectors(BB, modulus^mm)

    det = BB.det()
```

```
    bound = modulus^(mm*nn)
    if det >= bound:
        print ("We do not have det < bound. Solutions might not be found.")
        print ("Try with highers m and t.")
        if debug:
            diff = (log(det) - log(bound)) / log(2)
            print ("size det(L) - size e^(m*n) = ", floor(diff))
        if strict:
            return -1, -1
    else:
        print ("det(L) < e^(m*n) (good! If a solution exists < N^delta, it will be found)")

    if debug:
        matrix_overview(BB, modulus^mm)

    if debug:
        print ("optimizing basis of the lattice via LLL, this can take a long time")

    BB = BB.LLL()

    if debug:
        print ("LLL is done!")

    if debug:
        print ("在格中寻找线性无关向量")
    found_polynomials = False

    for pol1_idx in range(nn - 1):
        for pol2_idx in range(pol1_idx + 1, nn):
            PR.<w,z> = PolynomialRing(ZZ)
            pol1 = pol2 = 0
            for jj in range(nn):
                pol1 += monomials[jj](w*z+1,w,z) * BB[pol1_idx, jj] / monomials[jj](UU,XX,YY)
                pol2 += monomials[jj](w*z+1,w,z) * BB[pol2_idx, jj] / monomials[jj](UU,XX,YY)
            PR.<q> = PolynomialRing(ZZ)
            rr = pol1.resultant(pol2)
            if rr.is_zero() or rr.monomials() == [1]:
                continue
            else:
                print ("found them, using vectors", pol1_idx, "and", pol2_idx)
                found_polynomials = True
                break
        if found_polynomials:
            break

    if not found_polynomials:
        print ("no independant vectors could be found. This should very rarely happen...")
        return 0, 0

    rr = rr(q, q)
    soly = rr.roots()
```

```
    if len(soly) == 0:
        print ("Your prediction (delta) is too small")
        return 0, 0

    soly = soly[0][0]
    ss = pol1(q, soly)
    solx = ss.roots()[0][0]
    return solx, soly

def example():
    start =time.clock()
    size=512
    length_N = 2*size;
    ss=0
    s=70;
    M=1
    delta = 299/1024
    for i in range(M):
        N =
65318835751656706270462803918372182811096669561139853833192009681234356986381524661604904085035483519298788284835759796179173585004238691057332447589167439506386243352011548441011828732868691543989256629925692290088144403935880664585931943707422938295860559274669263630591393175387389387981929391774213395003
        e =
40738963799387627677248718814260921636491770341278750841486515130562842134876396915106681433734276005332410415984785584884091334455816402584507178231273998519376915363193650533215442952274343814099450187672503465016280527554101223321817109358581483535333734940968961773897326303002987203525415266163296607215
        c =
28858301095788879003830568949705095027466057921892765321643055383309726369907606901866332954652632036004551285284813812187678314978562231120323470819722044516158810710408496414616381570397632550468546302235826400628265458069027801013266037490695637948933487646553291637002155559023268928639502489285322508063
        hint1 =   624859718207126357681
        hint2 =   810475217500119132950

        m = 7
        t = round(((1-2*delta) * m))
        X = floor(N^delta)
        Y = floor(N^(1/2)/2^s)
        for l in range(int(hint1),int(hint1)+1):
            print('\n\n\n l=',l)
            pM=l;
            p0=pM*2^(size-s)+2^(size-s)-1;
            q0=N/p0;
            qM=int(q0/2^(size-s))
            A = N + 1-pM*2^(size-s)-qM*2^(size-s);
            P.<x,y> = PolynomialRing(ZZ)
            pol = 1 + x * (A + y)

            if debug:
                start_time = time.time()
            solx, soly = boneh_durfee(pol, e, m, t, X, Y)
```

```python
            if solx > 0:
                if False:
                    print ("x:", solx)
                    print ("y:", soly)

                d_sol = int(pol(solx, soly) / e)
                ss=ss+1

                print ("=== solution found ===")
                print ("p的高比特为：",l)
                print ("q的高比特为：",qM)
                print ("d=",d_sol)

            if debug:
                print("=== %s seconds ===" % (time.time() - start_time))
        print("ss=",ss)
        end=time.clock()
        print('Running time: %s Seconds'%(end-start))
if __name__ == "__main__":
    example()
```

得到结果：

```
 l= 624859718207126357681
det(L) < e^(m*n) (good! If a solution exists < N^delta, it will be found)
optimizing basis of the lattice via LLL, this can take a long time
LLL is done!
在格中寻找线性无关向量
found them, using vectors 0 and 1
=== solution found ===
p的高比特为：  624859718207126357681
q的高比特为：  810475217500119132950
d= 514966421261428616864174659198108787824325455855002618826560538514908088230254475149863519
=== 33.83120322227478 seconds ===
ss= 1
Running time: 33.84715294837952 Seconds
```

```
d=514966421261428616864174659198108787824325455855002618826560538514908088230254475149863519
```

然后用正常的rsa解密即可

```python
d=514966421261428616864174659198108787824325455855002618826560538514908088230254475149863519
e=407389637993876276772487188142609216364917703412787508414865151305628421348763969151066814337342760053324104159847855848840913344558164025845071782312739985193769153631936505332154429522743438140994501876725034650162805275541012233218171093585814835353337349409689617738973263030029872035254152661632966 07215
n=653188357516567062704628039183721828110966695611398538331920096812343569863815246616049040850354835192987882848357597961791735850042386910573324475891674395063862433520115484410118287328686915439892566299256922900881444039358806645859319437074229382958605592746692636305913931753873893879819293917742133950 03
c=288583010957888790038305689497050950274660579218927653216430553833097263699076069018663329546526320360045512852848138121876783149785622311203234708197220445161588107104084964146163815703976325504685463022358264006282654580690278010132660374906956379489334876465532916370021555590232689286395024892853225080 63
from Crypto.Util.number import *
m=pow(c,d,n)
print(long_to_bytes(m))
```

```
[2]:
d=514966421261428616864174659198108787824325455855002618826560538514908088230254475149863519
e=407389637993876276772487188142609216364917703412787508414865151305628421348763969151066814337342760053324104159847855848840913344558164025845071782312739
n=653188357516567062704628039183721828110966695611398538331920096812343569863815246616049040850354835192987882848357597961791735850042386910573324475891674
c=288583010957888790038305689497050950274660579218927653216430553833097263699076069018663329546526320360045512852848138121876783149785622311203234708197220
from Crypto.Util.number import *
m=pow(c,d,n)
print(long_to_bytes(m))
b'wdflag{097e0b16-3991-488a-b512-c3dbfb86db4a}'
```

解得：`b'wdflag{097e0b16-3991-488a-b512-c3dbfb86db4a}'`

```
wdflag{097e0b16-3991-488a-b512-c3dbfb86db4a}
```