赛题名称：Crypto1

解题步骤（WriteUp）

第一步：task:

已知 p,q 高位的 coppersmith 攻击，不过这次要同时猜测两个变量，使用格密码即可解决。

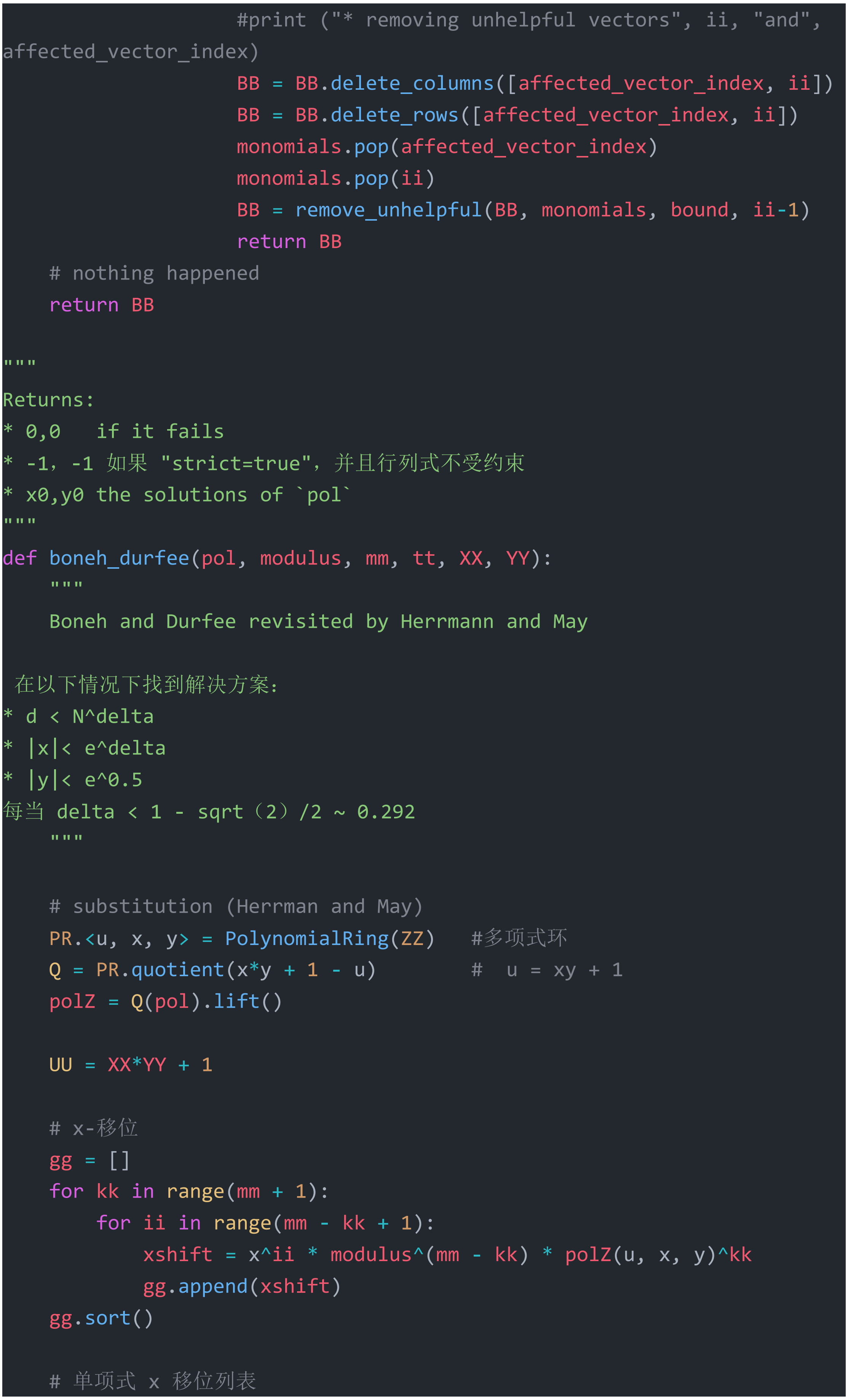
```
from Crypto.Util.number import *
from secret import flag

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 =
10298606334382818169101706132296175223148265097911761459232854033631955
99994199874177028119723234187421135201518886294725676039554819925149272
85801019993715247868027388036294100323295206260750653997980051233409135
84485256733800028438299225958729434485834767597199005886965860374215006
7210112531948312675289517
e =
94332227188033251470419190704216678578924281824166571884737945076375866
82424937635515990965447871322300310152561999033686699870566720437766171
32029489521716551431920759435789465738885764847462092614699701493818723
89631389369537155026693263975338398261567274837717090694055171425503933
824240291370948820767571
c =
84437879482958388121051989985943610317985560730924629180079819055930253
31381583535295916359347698581870048246223755270224784320490949831769051
27631857772671256470664666042958152919295054896113650305545593765467055
41333232100362213541469056985011640358767366350305910694542127597286950
765375388740496062563517
```

```
hint1 = 7371328422265637311129
hint2 = 1083219649182192077965
```

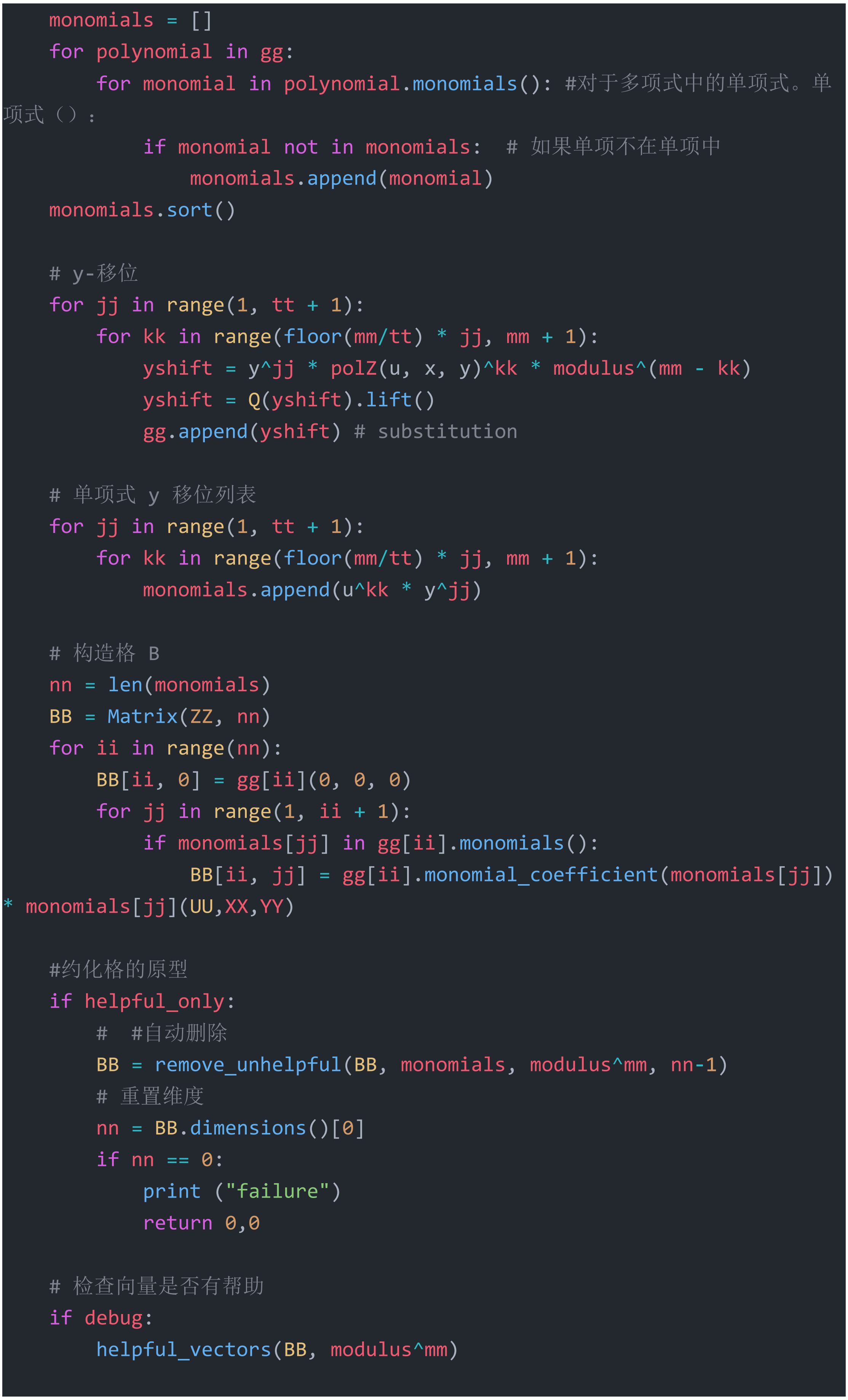
第二步：网络上有相似题目，直接使用 exp 修改后即可：
https://www.cnblogs.com/mumuhhh/p/17789591.html

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

debug = True

strict = False

helpful_only = True
dimension_min = 7 # 如果晶格达到该尺寸，则停止移除
# 显示有用矢量的统计数据
def helpful_vectors(BB, modulus):
    nothelpful = 0
    for ii in range(BB.dimensions()[0]):
        if BB[ii,ii] >= modulus:
            nothelpful += 1

    print(nothelpful, "/", BB.dimensions()[0], " vectors are not
helpful")

# 显示带有 0 和 X 的矩阵
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 += '~'
        #print (a)
```

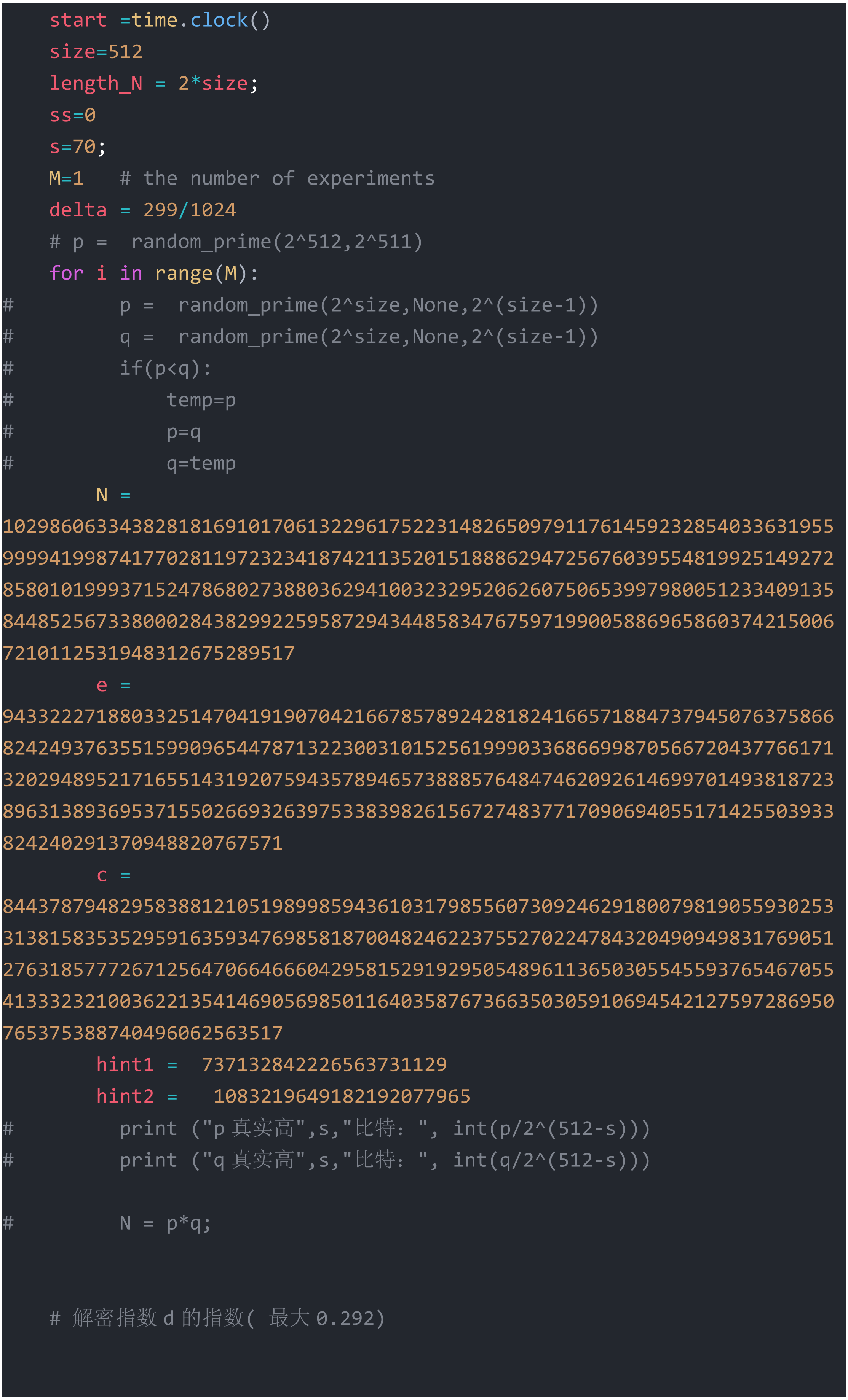
```
# 尝试删除无用的向量
# 从当前 = n-1（最后一个向量）开始
def remove_unhelpful(BB, monomials, bound, current):
    # 我们从当前 = n-1（最后一个向量）开始
    if current == -1 or BB.dimensions()[0] <= dimension_min:
        return BB

    # 开始从后面检查
    for ii in range(current, -1, -1):
        #  如果它没有用
        if BB[ii, ii] >= bound:
            affected_vectors = 0
            affected_vector_index = 0
             # 让我们检查它是否影响其他向量
            for jj in range(ii + 1, BB.dimensions()[0]):
                # 如果另一个向量受到影响：
                # 我们增加计数
                if BB[jj, ii] != 0:
                    affected_vectors += 1
                    affected_vector_index = jj

            # 等级：0
            # 如果没有其他载体最终受到影响
            # 我们删除它
            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

           # 等级：1
            #如果只有一个受到影响，我们会检查
            # 如果它正在影响别的向量
            elif affected_vectors == 1:
                affected_deeper = True
                for kk in range(affected_vector_index + 1,
BB.dimensions()[0]):
                    # 如果它影响哪怕一个向量
                    # 我们放弃这个
                    if BB[kk, affected_vector_index] != 0:
                        affected_deeper = False
                # 如果没有其他向量受到影响，则将其删除，并且
                # 这个有用的向量不够有用
                #与我们无用的相比
                if affected_deeper and abs(bound -
BB[affected_vector_index, affected_vector_index]) < abs(bound - BB[ii,
ii]):
```



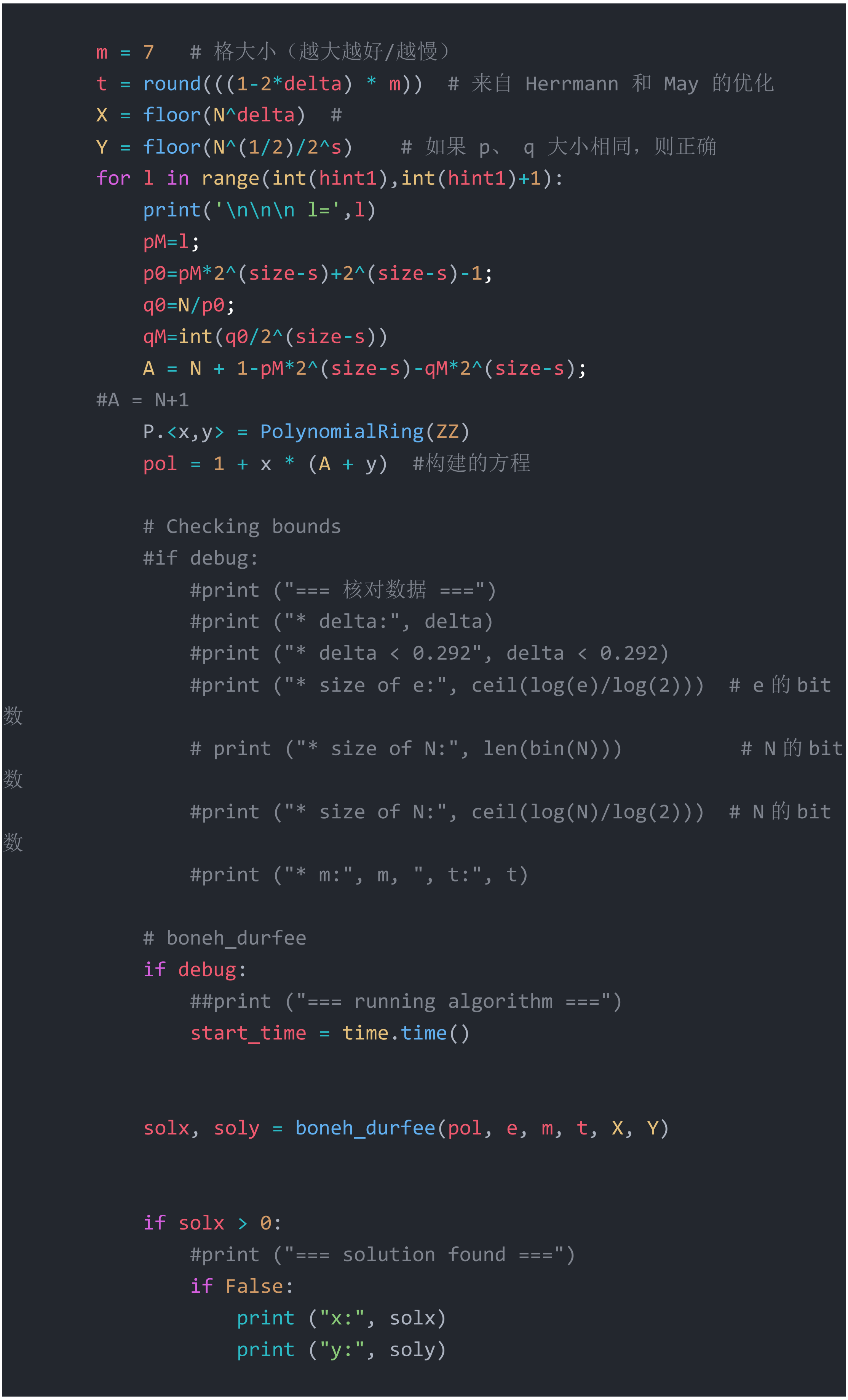
```
                    #print ("* removing unhelpful vectors", ii, "and",
affected_vector_index)
                    BB = BB.delete_columns([affected_vector_index, ii])
                    BB = BB.delete_rows([affected_vector_index, ii])
                    monomials.pop(affected_vector_index)
                    monomials.pop(ii)
                    BB = remove_unhelpful(BB, monomials, bound, ii-1)
                    return BB
    # nothing happened
    return BB

"""
Returns:
* 0,0   if it fails
* -1，-1 如果 "strict=true"，并且行列式不受约束
* x0,y0 the solutions of `pol`
"""
def boneh_durfee(pol, modulus, mm, tt, XX, YY):
    """
    Boneh and Durfee revisited by Herrmann and May

 在以下情况下找到解决方案:
* d < N^delta
* |x|< e^delta
* |y|< e^0.5
每当 delta < 1 - sqrt（2）/2 ~ 0.292
    """

    # substitution (Herrman and May)
    PR.<u, x, y> = PolynomialRing(ZZ)   #多项式环
    Q = PR.quotient(x*y + 1 - u)        #  u = xy + 1
    polZ = Q(pol).lift()

    UU = XX*YY + 1

    # x-移位
    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()

    # 单项式 x 移位列表
```



```
    monomials = []
    for polynomial in gg:
        for monomial in polynomial.monomials(): #对于多项式中的单项式。单项式（）：
            if monomial not in monomials:  # 如果单项不在单项中
                monomials.append(monomial)
    monomials.sort()

    # y-移位
    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) # substitution

    # 单项式 y 移位列表
    for jj in range(1, tt + 1):
        for kk in range(floor(mm/tt) * jj, mm + 1):
            monomials.append(u^kk * y^jj)

    # 构造格 B
    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)

    #约化格的原型
    if helpful_only:
        #  #自动删除
        BB = remove_unhelpful(BB, monomials, modulus^mm, nn-1)
        # 重置维度
        nn = BB.dimensions()[0]
        if nn == 0:
            print ("failure")
            return 0,0

    # 检查向量是否有帮助
    if debug:
        helpful_vectors(BB, modulus^mm)
    # 检查行列式是否正确界定
    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)")

    # display the lattice basis
    if debug:
        matrix_overview(BB, modulus^mm)

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

    #BB = BB.BKZ(block_size=25)
    BB = BB.LLL()

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

    # 替换向量 i 和 j ->多项式 1 和 2
    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):

            # 对于 i and j, 构造两个多项式

            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)

            # 结果
            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)

    # solutions
    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():
    ############################################
    # 随机生成数据
    ##########################################
    #start_time =time.perf_counter
```



```
    start =time.clock()
    size=512
    length_N = 2*size;
    ss=0
    s=70;
    M=1   # the number of experiments
    delta = 299/1024
    # p =  random_prime(2^512,2^511)
    for i in range(M):
#         p =  random_prime(2^size,None,2^(size-1))
#         q =  random_prime(2^size,None,2^(size-1))
#         if(p<q):
#             temp=p
#             p=q
#             q=temp
        N =
10298606334382818169101706132296175223148265097911761459232854033631955
99994199874177028119723234187421135201518886294725676039554819925149272
85801019993715247868027388036294100323295206260750653997980051233409135
84485256733800028438299225958729434485834767597199005886965860374215006
7210112531948312675289517
        e =
94332227188033251470419190704216678578924281824166571884737945076375866
82424937635515990965447871322300310152561999033686699870566720437766171
32029489521716551431920759435789465738885764847462092614699701493818723
89631389369537155026693263975338398261567274837717090694055171425503933
824240291370948820767571
        c =
84437879482958388121051989985943610317985560730924629180079819055930253
31381583535295916359347698581870048246223755270224784320490949831769051
27631857772671256470664666042958152919295054896113650305545593765467055
41333232100362213541469056985011640358767366350305910694542127597286950
765375388740496062563517
        hint1 =  737132842226563731129
        hint2 =   1083219649182192077965
#         print ("p 真实高",s,"比特: ", int(p/2^(512-s)))
#         print ("q 真实高",s,"比特: ", int(q/2^(512-s)))

#         N = p*q;


    # 解密指数 d 的指数( 最大 0.292)
```

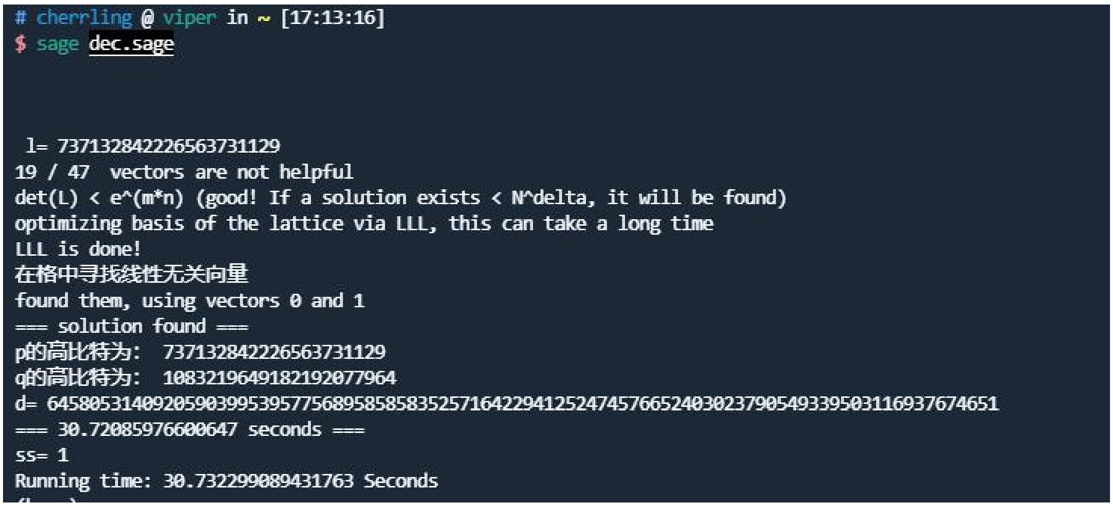


```
m = 7   # 格大小（越大越好/越慢）
t = round(((1-2*delta) * m))  # 来自 Herrmann 和 May 的优化
X = floor(N^delta)  #
Y = floor(N^(1/2)/2^s)    # 如果 p、 q 大小相同，则正确
for l in range(int(hint1),int(hint1)+1):
    print('\n\n\n l=',l)
    pM=l;
    p0=pM*2^(size-s)+2^(size-s)-1;
    q0=N/p0;
    qM=int(q0/2^(size-s))
    A = N + 1-pM*2^(size-s)-qM*2^(size-s);
#A = N+1
    P.<x,y> = PolynomialRing(ZZ)
    pol = 1 + x * (A + y)  #构建的方程

    # Checking bounds
    #if debug:
        #print ("=== 核对数据 ===")
        #print ("* delta:", delta)
        #print ("* delta < 0.292", delta < 0.292)
        #print ("* size of e:", ceil(log(e)/log(2)))  # e 的 bit
数

        # print ("* size of N:", len(bin(N)))          # N 的 bit
数

        #print ("* size of N:", ceil(log(N)/log(2)))  # N 的 bit
数

        #print ("* m:", m, ", t:", t)

    # boneh_durfee
    if debug:
        ##print ("=== running algorithm ===")
        start_time = time.time()


    solx, soly = boneh_durfee(pol, e, m, t, X, Y)


    if solx > 0:
        #print ("=== solution found ===")
        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))
            #break
        print("ss=",ss)
                            #end=time.process_time
        end=time.clock()
        print('Running time: %s Seconds'%(end-start))
if __name__ == "__main__":
    example()
```

第三步：输出 d 值

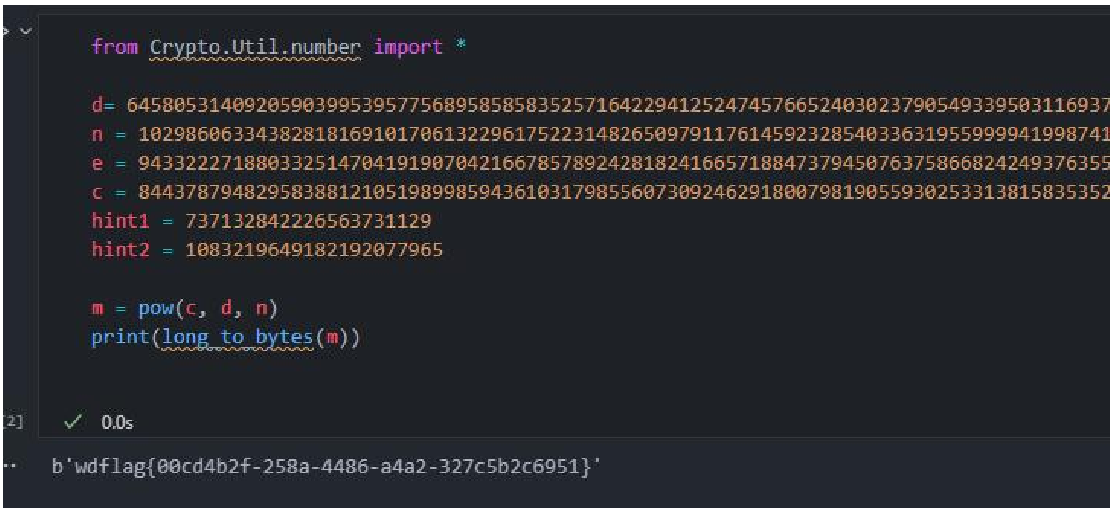


```
# cherrling @ viper in ~ [17:13:16]
$ sage dec.sage



 l= 737132842226563731129
19 / 47  vectors are not helpful
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:  737132842226563731129
q的高比特为:  1083219649182192077964
d= 645805314092059039953957756895858583525716422941252474576652403023790549339503116937674651
=== 30.72085976600647 seconds ===
ss= 1
Running time: 30.732299089431763 Seconds
```

……

第 N 步：计算输出 flag

b'wdflag{00cd4b2f-258a-4486-a4a2-327c5b2c6951}'



```
from Crypto.Util.number import *

d= 645805314092059039953957756895858583525716422941252474576652403023790549339503116937
n = 10298606334382818169101706132296175223148265097911761459232854033631955999941998741
e = 94332227188033251470419190704216678578924281824166571884737945076375866824249376355
c = 84437879482958388121051989985943610317985560730924629180079819055930253313815835352
hint1 = 737132842226563731129
hint2 = 1083219649182192077965

m = pow(c, d, n)
print(long_to_bytes(m))
```

[2] ✓ 0.0s

```
b'wdflag{00cd4b2f-258a-4486-a4a2-327c5b2c6951}'
```