赛题解答

1(25 points)(done)

> N1=0xa4d80845630d3b332f74f667ec8a0e49aba15b6f0c4f4006161d62c91b78cf6811421cc76609d2d9dba2c43be9d8ecdc6a0dff64a8041dcde52c7f92820b0a38fc91419e8ec9a5c69d47edc6e347934b4d87f97c5759886dac6c1143ff55b8eb11acfaa6cc70956a8ec7796e1a063b123bc2e467e30937c5a69c7ab5f8ed17e1;
>
> e1=0x3458c2e97adef45f741c7db11ece6c0814aa5b6fad9144242cdaa16a6b4f3622477935f98a41765b92892b4de22a391cf08767447df113f5151c86edd109b97f9b045fd8ad5d7a51084684d4e2353db6c0e474d5d79f399a2bf4fd867ec85b7960845ab5497f705914912f797804c06dcff57139e040596d22b141e54835e0d3;
>
> c1=0x91b097a5b1f6b12accdbda15cd2247384e1b3ed8311085a0f3e0dbb5fffce650a355600a02674189d1b7f4075df079c70354a08646e85ecf31dd150220cd1d4ce22d55a946500f4bd8def74fb0acea3e8d2e7bb1d27ebf2ca2e80fc28c3f0d88a041d4a556a18147f66b88c65f19c99b4b94c3f78d468b8accb4da7e7ce31b29;
>
> 已知信息: 私钥d_1的取值范围为[2^{249}, 2^{250}]。

solution

首先，我们将三个已知的数都转化为十进制数，利用python我们得到：

```
n=11575730647648082389010148818863077672926481862595671119720918292331691945000456670513603713812470775724136423836220284833432623096586618288437290559981200141729579207600838215505150785455891775154099715203115815362751663860989166299202145552520887615251221942602273038843016898663481292599747743307400931529 7

e=36759119760107514342451522014504723756781932778854536007844886957566674786814219616184569606624829254075411466011531177462192834674042314582632169998869937630047822402849141636204405624465283618559148388023239618010023983515302829096233233543555639120782376815166070858869807741037590528029792278496614408403

c=10230686654816629728399057072871938074627435654734945631391113452551100988452759507761965831274615454091804060990479276626754093955423066125579044239871813916141413321893545539544000962665237747802204776291184052217098005138739126177489671042198397431885717655646736654516530679064136491719579044423584993156 1
```

我们观察到d小于n的$\frac{1}{4}$次，于是查阅资料，我们使用wiener attack来对其发起攻击。攻击脚本如下所示，在sagemath5.0环境下运行。

```
n=11575730647648082389010148818863077672926481862595671119720918292331691945000456670513603713812470775724136423836220284833432623096586618288437290559981200141729579207600838215505150785455891775154099715203115815362751663860989166299202145552520887615251221942602273038843016898663481292599747743307400931529 7
e=36759119760107514342451522014504723756781932778854536007844886957566674786814219616184569606624829254075411466011531177462192834674042314582632169998869937630047822402849141636204405624465283618559148388023239618010023983515302829096233233543555639120782376815166070858869807741037590528029792278496614408403
```

```
c=10230686654816629728399057072871938074627435654734945631391113452551100988452759507761965831274615454091804060990479276626754093955423066125579044239871813916141413321893545539544000962665237747802204776291184052217098005138739126177489671042198397431885717655646736654516530679064136491719579044423584993156 1

#Sage
def factor_rsa_wiener(N, e):
    N = Integer(N)
    e = Integer(e)
    cf = (e / N).continued_fraction().convergents()
    for f in cf:
        k = f.numer()
        d = f.denom()
        if k == 0:
            continue
        phi_N = ((e * d) - 1) / k
        b = -(N - phi_N + 1)
        dis = b ^ 2 - 4 * N
        if dis.sign() == 1:
            dis_sqrt = sqrt(dis)
            p = (-b + dis_sqrt) / 2
            q = (-b - dis_sqrt) / 2
            if p.is_integer() and q.is_integer() and (p * q) % N == 0:
                p = p % N
                q = q % N
                if p > q:
                    return (p, q)
                else:
                    return (q, p)

factor_rsa_wiener(n,e)
```

我们得到了p和q，分别是：

```
p=11731831938699772462127271873430115361544445093018344205508116263256543526314353604701640010896040499228777875913175294497034647460453248703129442880389173

q=98669421008864276080202536014862045622389251159471626544583577709359233654377182204113913919202457118314272975400864218592720506663004906761854089226993891
```

然后我们求得n的欧拉函数，即可获得d，我们往回代入，验证d的bit位是否在规定的范围内，发现d的bit位刚好是250位，符合题目条件。

再由给出的c来还原密文m即可，代码如下所示：

```
from gmpy2 import *
p=11731831938699772462127271873430115361544445093018344205508116263256543526314353604701640010896040499228777875913175294497034647460453248703129442880389173
q=98669421008864276080202536014862045622389251159471626544583577709359233654377182204113913919202457118314272975400864218592720506663004906761854089226993893
n=p*q
e=36759119760107514342451522014504723756781932778854536007844886957566674786814219616184569606624829254075411466011531177462192834674042314582632169998869937630047822402849141636204405624465283618559148388023239618010023983515302829096233233546355639120782376815166070858869807741037590528029792278496614408403
phi=(p-1)*(q-1)
d=invert(e,phi)
c=10230686654816629728399057072871938074627435654734945631391113452551100988452759507761965831274615454091804060990479276626754093955423066125579044239871813916141413321893545539544000962665237747802204776291184052217098005138739126177489671042198397431885717655646736654516530679064136491719579044423584993156 1
m=pow(c,d,n)
print(m)
```

得到结果，m的值为：

```
m=72115607506154761336411014109985685056395095747321491967349423247749565387913611968183729328564132391301926185639936696286792673324141050046544603161058262375131485972136960821908946097232689498056450728225835250747872903494693577037
```

首先将其转化为十六进制，然后根据赛题提示，还原原文即可。

```
m=72115607506154761336411014109985685056395095747321491967349423247749565387913611968183729328564132391301926185639936696286792673324141050046544603161058262375131485972136960821908946097232689498056450728225835250747872903494693577037
print(hex(m))
#0x2e73636974616d656874616d20666f206e6565757120656874207369205d79726f656874207265626d756e5b20636974656d687469726120646e61202c7365636e6569637320666f206e6565757120656874207369207363697461 6d656874614d
```

然后再还原为ASCII码就可以了，得出最后的结果：

```
Mathematics is the queen of sciences, and arithmetic [number theory] is the queen of mathematics.
```

翻译成中文就是：**数学是科学的女王，而算术（数论）是数学的女王。**

2(30 points)(done)

> N2=0xd231f2c194d3971821984dec9cf1ef58d538975f189045ef8a706f6165aab4929096f61a3eb7dd8021bf3fdc41fe3b3b0e4ecc579b4b5e7e035ffcc383436c9656533949881dca67c26d0e770e4bf62a09718dbabc2b40f2938f16327e347f187485aa48b044432e82f5371c08f6e0bbde46c713859aec715e2a2ca66574f3eb;

> e2=0x5b5961921a49e3089262761e89629ab6dff2da1504a0e5eba1bb7b20d63c785a013fd6d9e021c01baf1b23830954d488041b92bca2fe2c92e3373dedd7e625da11275f6f18ee4aef336d0637505545f70f805902ddbacb21bb8276d34a0f6dfe37ede87dd95bb1494dbb5763639ba3984240f1178e32aa36ee3c5fcc8115dde5;
>
> c2=0x6a88a8fa2b8f28d96284298bab2061efeb35e3a086370e19523c15c429f5d783b9d4f32e31a402916f45ad4f2760ab30e77177335af44756bfbeef0f168b5e0dc8c3ddf75d141c358969cca0e7c2b8ab99ef8e33b031be1cbccd95b687682ac7b0dcc0d56f5651ee671d6358128d2e0801f247a6af4fe0dc5e8fb199eba0780f;
>
> 已知信息: 私钥d_2的取值范围为$[2^{285}, 2^{286}]$。

solution

我们根据d和n的大小关系，确定出该题我们应该使用$Boneh\ and\ Durfee\ attack$来进行攻击。我们使用下面的脚本，先求出

```
from __future__ import print_function
import time

############################################
# Config
############################################

"""
Setting debug to true will display more informations
about the lattice, the bounds, the vectors...
"""
debug = True

"""
Setting strict to true will stop the algorithm (and
return (-1, -1)) if we don't have a correct
upperbound on the determinant. Note that this
doesn't necesseraly mean that no solutions
will be found since the theoretical upperbound is
usualy far away from actual results. That is why
you should probably use `strict = False`
"""
strict = False

"""
This is experimental, but has provided remarkable results
so far. It tries to reduce the lattice as much as it can
while keeping its efficiency. I see no reason not to use
this option, but if things don't work, you should try
disabling it
"""
helpful_only = True
dimension_min = 7 # stop removing if lattice reaches that dimension

# display stats on helpful vectors
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")

# display matrix picture with 0 and X
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)

# tries to remove unhelpful vectors
# we start at current = n-1 (last vector)
def remove_unhelpful(BB, monomials, bound, current):
    # end of our recursive function
    if current == -1 or BB.dimensions()[0] <= dimension_min:
        return BB

    # we start by checking from the end
    for ii in range(current, -1, -1):
        # if it is unhelpful:
        if BB[ii, ii] >= bound:
            affected_vectors = 0
            affected_vector_index = 0
            # let's check if it affects other vectors
            for jj in range(ii + 1, BB.dimensions()[0]):
                # if another vector is affected:
                # we increase the count
                if BB[jj, ii] != 0:
                    affected_vectors += 1
                    affected_vector_index = jj

            # level:0
            # if no other vectors end up affected
            # we remove it
            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

            # level:1
            # if just one was affected we check
            # if it is affecting someone else
            elif affected_vectors == 1:
                affected_deeper = True
                for kk in range(affected_vector_index + 1, BB.dimensions()[0]):
                    # if it is affecting even one vector
```

```
                    # we give up on this one
                    if BB[kk, affected_vector_index] != 0:
                        affected_deeper = False
                # remove both it if no other vector was affected and
                # this helpful vector is not helpful enough
                # compared to our unhelpful one
                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 if `strict=true`, and determinant doesn't bound
* x0,y0 the solutions of `pol`
"""
def boneh_durfee(pol, modulus, mm, tt, XX, YY):
    """
    Boneh and Durfee revisited by Herrmann and May

    finds a solution if:
    * d < N^delta
    * |x| < e^delta
    * |y| < e^0.5
    whenever delta < 1 - sqrt(2)/2 ~ 0.292
    """

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

    UU = XX*YY + 1

    # x-shifts
    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()

    # x-shifts list of monomials
    monomials = []
    for polynomial in gg:
        for monomial in polynomial.monomials():
            if monomial not in monomials:
```

```
                monomials.append(monomial)
    monomials.sort()

    # y-shifts (selected by Herrman and May)
    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

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

    # construct lattice 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)

    # Prototype to reduce the lattice
    if helpful_only:
        # automatically remove
        BB = remove_unhelpful(BB, monomials, modulus^mm, nn-1)
        # reset dimension
        nn = BB.dimensions()[0]
        if nn == 0:
            print("failure")
            return 0,0

    # check if vectors are helpful
    if debug:
        helpful_vectors(BB, modulus^mm)

    # check if determinant is correctly bounded
    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.LLL()

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

    # transform vector i & j -> polynomials 1 & 2
    if debug:
        print("looking for independent vectors in the lattice")
    found_polynomials = False

    for pol1_idx in range(nn - 1):
        for pol2_idx in range(pol1_idx + 1, nn):
            # for i and j, create the two polynomials
            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)

            # resultant
            PR.<q> = PolynomialRing(ZZ)
            rr = pol1.resultant(pol2)

            # are these good polynomials?
            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():
    # the modulus
    N =
0xd231f2c194d3971821984dec9cf1ef58d538975f189045ef8a706f6165aab4929096f61a3eb7dd8
021bf3fdc41fe3b3b0e4ecc579b4b5e7e035ffcc383436c9656533949881dca67c26d0e770e4bf62a
09718dbabc2b40f2938f16327e347f187485aa48b044432e82f5371c08f6e0bbde46c713859aec715
e2a2ca66574f3eb
    # the public exponent
    e =
0x5b5961921a49e3089262761e89629ab6dff2da1504a0e5eba1bb7b20d63c785a013fd6d9e021c01
baf1b23830954d488041b92bca2fe2c92e3373dedd7e625da11275f6f18ee4aef336d0637505545f7
0f805902ddbacb21bb8276d34a0f6dfe37ede87dd95bb1494dbb5763639ba3984240f1178e32aa36e
e3c5fcc8115dde5

    # the hypothesis on the private exponent (the theoretical maximum is 0.292)
    delta = .28 # this means that d < N^delta

    m = 13 # size of the lattice (bigger the better/slower)

    # you need to be a lattice master to tweak these
    t = int((1-2*delta) * m)  # optimization from Herrmann and May
    X = 2*floor(N^delta)  # this _might_ be too much
    Y = floor(N^(1/2))    # correct if p, q are ~ same size
    # Problem put in equation
    P.<x,y> = PolynomialRing(ZZ)
    A = int((N+1)/2)
    pol = 1 + x * (A + y)

    #
    # Find the solutions!
    #

    # Checking bounds
    if debug:
        print("=== checking values ===")
        print("* delta:", delta)
        print("* delta < 0.292", delta < 0.292)
        print("* size of e:", int(log(e)/log(2)))
        print("* size of N:", int(log(N)/log(2)))
        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)

    # found a solution?
    if solx > 0:
        print("=== solution found ===")
        if False:
            print("x:", solx)
```

```
            print("y:", soly)

        d = int(pol(solx, soly) / e)
        print("private key found:", d)
    else:
        print("=== no solution was found ===")

    if debug:
        print(("=== %s seconds ===" % (time.time() - start_time)))

if __name__ == "__main__":
    example()
```

使用该脚本，攻击出的结果是：

```
=== checking values ===
* delta: 0.280000000000000
* delta < 0.292 True
* size of e: 1022
* size of N: 1023
* m: 13 , t: 5
=== running algorithm ===
* removing unhelpful vector 142
* removing unhelpful vector 141
* removing unhelpful vector 136
* removing unhelpful vector 135
* removing unhelpful vector 127
* removing unhelpful vector 117
* removing unhelpful vector 105
* removing unhelpful vectors 1 and 2
* removing unhelpful vector 0
57 / 135  vectors are not helpful
det(L) < e^(m*n) (good! If a solution exists < N^delta, it will be found)
00
X000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000~
01
XX000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000~
02
XXX000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
03
000X000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000~
04
000XX000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000~
05
000XXX000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
06
000XXXX000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
```

07
0000000x00~
08
0000000xx00~
09
0000000xxx00~
10
0000000xxxx00
11
0000000xxxxx00
12
00000000000x00~
13
00000000000xx00~
14
00000000000xxx00~
15
00000000000xxxx00
16
00000000000xxxxx00
17
00000000000xxxxxx00
18
000000000000000000x00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000~
19
000000000000000000xx00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000~
20
000000000000000000xxx00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000~
21
000000000000000000xxxx00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000~
22
000000000000000000xxxxx00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
23
000000000000000000xxxxxx00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
24
000000000000000000xxxxxxx00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000

25
000000000000000000000000x00000000000000000000000000000000000000000000000000000000000000000000000000000000~
26
000000000000000000000000xx00000000000000000000000000000000000000000000000000000000000000000000000000000000~
27
000000000000000000000000xxx00000000000000000000000000000000000000000000000000000000000000000000000000000000~
28
000000000000000000000000xxxx00000000000000000000000000000000000000000000000000000000000000000000000000000000~
29
000000000000000000000000xxxxx00000000000000000000000000000000000000000000000000000000000000000000000000000000
30
000000000000000000000000xxxxxx00000000000000000000000000000000000000000000000000000000000000000000000000000000
31
000000000000000000000000xxxxxxx00000000000000000000000000000000000000000000000000000000000000000000000000000000
32
000000000000000000000000xxxxxxxx00000000000000000000000000000000000000000000000000000000000000000000000000000000
33
0000000000000000000000000000000000x00000000000000000000000000000000000000000000000000000000000000000000000000000000~
34
0000000000000000000000000000000000xx00000000000000000000000000000000000000000000000000000000000000000000000000000000~
35
0000000000000000000000000000000000xxx00000000000000000000000000000000000000000000000000000000000000000000000000000000~
36
0000000000000000000000000000000000xxxx00000000000000000000000000000000000000000000000000000000000000000000000000000000~
37
0000000000000000000000000000000000xxxxx00000000000000000000000000000000000000000000000000000000000000000000000000000000~
38
0000000000000000000000000000000000xxxxxx00000000000000000000000000000000000000000000000000000000000000000000000000000000
39
0000000000000000000000000000000000xxxxxxx00000000000000000000000000000000000000000000000000000000000000000000000000000000
40
0000000000000000000000000000000000xxxxxxxx00000000000000000000000000000000000000000000000000000000000000000000000000000000
41
0000000000000000000000000000000000xxxxxxxxx00000000000000000000000000000000000000000000000000000000000000000000000000000000
42
0000000000000000000000000000000000000000000x00000000000000000000000000000000000000000000000000000000000000000000000000000000~

43
000000000000000000000000000000000000000000xx00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000~
44
000000000000000000000000000000000000000000xxx00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000~
45
000000000000000000000000000000000000000000xxxx00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000~
46
000000000000000000000000000000000000000000xxxxx00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000~
47
000000000000000000000000000000000000000000xxxxxx00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000~
48
000000000000000000000000000000000000000000xxxxxxx00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
49
000000000000000000000000000000000000000000xxxxxxxx00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
50
000000000000000000000000000000000000000000xxxxxxxxx00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
51
000000000000000000000000000000000000000000xxxxxxxxxx00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
52
000000000000000000000000000000000000000000000000000000000000x00000000000000000000000000000000000000000000000000000000000000000000000000000000000000~
53
000000000000000000000000000000000000000000000000000000000000xx00000000000000000000000000000000000000000000000000000000000000000000000000000000000000~
54
000000000000000000000000000000000000000000000000000000000000xxx00000000000000000000000000000000000000000000000000000000000000000000000000000000000000~
55
000000000000000000000000000000000000000000000000000000000000xxxx00000000000000000000000000000000000000000000000000000000000000000000000000000000000000~
56
000000000000000000000000000000000000000000000000000000000000xxxxx00000000000000000000000000000000000000000000000000000000000000000000000000000000000000~
57
000000000000000000000000000000000000000000000000000000000000xxxxxx00000000000000000000000000000000000000000000000000000000000000000000000000000000000000~
58
000000000000000000000000000000000000000000000000000000000000xxxxxxx00000000000000000000000000000000000000000000000000000000000000000000000000000000000000
59
000000000000000000000000000000000000000000000000000000000000xxxxxxxx00000000000000000000000000000000000000000000000000000000000000000000000000000000000000
60
000000000000000000000000000000000000000000000000000000000000xxxxxxxxx00000000000000000000000000000000000000000000000000000000000000000000000000000000000000

```
61
000000000000000000000000000000000000000000000000000xxxxxxxxxx0000000000000000000
000000000000000000000000000000000000000000000000000000000000
62
000000000000000000000000000000000000000000000000000xxxxxxxxxxx0000000000000000000
000000000000000000000000000000000000000000000000000000000000
63
000000000000000000000000000000000000000000000000000000000000000x0000000000000000
000000000000000000000000000000000000000000000000000000000000~
64
000000000000000000000000000000000000000000000000000000000000000xx0000000000000000
000000000000000000000000000000000000000000000000000000000000~
65
000000000000000000000000000000000000000000000000000000000000000xxx0000000000000000
000000000000000000000000000000000000000000000000000000000000~
66
000000000000000000000000000000000000000000000000000000000000000xxxx0000000000000000
000000000000000000000000000000000000000000000000000000000000~
67
000000000000000000000000000000000000000000000000000000000000000xxxxx0000000000000000
000000000000000000000000000000000000000000000000000000000000~
68
000000000000000000000000000000000000000000000000000000000000000xxxxxx0000000000000000
000000000000000000000000000000000000000000000000000000000000~
69
000000000000000000000000000000000000000000000000000000000000000xxxxxxx0000000000000000
000000000000000000000000000000000000000000000000000000000000~
70
000000000000000000000000000000000000000000000000000000000000000xxxxxxxx0000000000000000
000000000000000000000000000000000000000000000000000000000000
71
000000000000000000000000000000000000000000000000000000000000000xxxxxxxxx000000000000000
000000000000000000000000000000000000000000000000000000000000
72
000000000000000000000000000000000000000000000000000000000000000xxxxxxxxxx00000000000000
000000000000000000000000000000000000000000000000000000000000
73
000000000000000000000000000000000000000000000000000000000000000xxxxxxxxxxx0000000000000
000000000000000000000000000000000000000000000000000000000000
74
000000000000000000000000000000000000000000000000000000000000000xxxxxxxxxxxx000000000000
000000000000000000000000000000000000000000000000000000000000
75
000000000000000000000000000000000000000000000000000000000000000000000000000000000000000x00000
000000000000000000000000000000000000000000000000000000000000~
76
000000000000000000000000000000000000000000000000000000000000000000000000000000000000000xx0000
000000000000000000000000000000000000000000000000000000000000~
77
000000000000000000000000000000000000000000000000000000000000000000000000000000000000000xxx000
000000000000000000000000000000000000000000000000000000000000~
78
000000000000000000000000000000000000000000000000000000000000000000000000000000000000000xxxx00
000000000000000000000000000000000000000000000000000000000000~
```

```
000000000000000000000000000000000000000000000000000000000000000000000000000xxxxx0000000000000000000000000000000000000000000000000000000000~
000000000000000000000000000000000000000000000000000000000000000000000000000xxxxxx000000000000000000000000000000000000000000000000000000~
000000000000000000000000000000000000000000000000000000000000000000000000000xxxxxxx00000000000000000000000000000000000000000000000000000~
000000000000000000000000000000000000000000000000000000000000000000000000000xxxxxxxx0000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000xxxxxxxxx000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000xxxxxxxxxx00000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000xxxxxxxxxxx0000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000xxxxxxxxxxxx000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000xxxxxxxxxxxxx00000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000x0000000000000000000000000000000000000000000000~
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000xx000000000000000000000000000000000000000000000~
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000xxx00000000000000000000000000000000000000000000~
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000xxxx0000000000000000000000000000000000000000000~
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000xxxxx000000000000000000000000000000000000000000~
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000xxxxxx00000000000000000000000000000000000000000~
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000xxxxxxx0000000000000000000000000000000000000000~
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000xxxxxxxx000000000000000000000000000000000000000~
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000xxxxxxxxx00000000000000000000000000000000000000
```

```
97
000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000xxxxxxxxxx0000000000000000000000000000000000000
98
000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000xxxxxxxxxxx000000000000000000000000000000000000
99
000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000xxxxxxxxxxxx00000000000000000000000000000000000
100
000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000xxxxxxxxxxxxx0000000000000000000000000000000000
101
000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000xxxxxxxxxxxxxx000000000000000000000000000000000
102
xxx0xxx00000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000x00000000000000000000000000000000
103
000xxxx0xxxx000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000x0000000000000000000000000000000
104
0000000xxxxx0xxxxx000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000x000000000000000000000000000000
105
000000000000xxxxxx0xxxxxx00000000000000000000000000000000000000000000000000000000
000000000000000000000000x00000000000000000000000000000
106
000000000000000000xxxxxxx0xxxxxxx000000000000000000000000000000000000000000000000
0000000000000000000000000x0000000000000000000000000000
107
0000000000000000000000000xxxxxxxx0xxxxxxxx000000000000000000000000000000000000000
00000000000000000000000000x000000000000000000000000000
108
000000000000000000000000000000000xxxxxxxxx0xxxxxxxxx00000000000000000000000000000
000000000000000000000000000x00000000000000000000000000
109
000000000000000000000000000000000000000000xxxxxxxxxx0xxxxxxxxxx000000000000000000
0000000000000000000000000000x0000000000000000000000000
110
0000000000000000000000000000000000000000000000000000xxxxxxxxxxx0xxxxxxxxxxx000000
00000000000000000000000000000x000000000000000000000000
111
000000000000000000000000000000000000000000000000000000000000000xxxxxxxxxxxx0xxxxx
xxxxxxx00000000000000000000000x00000000000000000000000
112
000000000000000000000000000000000000000000000000000000000000000000000000000xxxxxx
xxxxxxx0xxxxxxxxxxxxx0000000000x0000000000000000000000
113
000xxxx0xxxx00xxxx000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000xx00000000x000000000000000000000
114
0000000xxxxx0xxxxx00xxxxx00000000000000000000000000000000000000000000000000000000
00000000000000000000000xx00000000x00000000000000000000
```

```
115
000000000000xxxxxx0xxxxxx00xxxxxx000000000000000000000000000000000000000000000000
000000000000000000000000xx00000000x0000000000000000000
116
000000000000000000xxxxxxx0xxxxxxx00xxxxxxx000000000000000000000000000000000000000
0000000000000000000000000xx00000000x000000000000000000
117
0000000000000000000000000xxxxxxxx0xxxxxxxx00xxxxxxxx00000000000000000000000000000
00000000000000000000000000xx00000000x00000000000000000
118
000000000000000000000000000000000xxxxxxxxx0xxxxxxxxx00xxxxxxxxx000000000000000000
000000000000000000000000000xx00000000x0000000000000000
119
000000000000000000000000000000000000000000xxxxxxxxxx0xxxxxxxxxx00xxxxxxxxxx000000
0000000000000000000000000000xx00000000x000000000000000
120
0000000000000000000000000000000000000000000000000000xxxxxxxxxxx0xxxxxxxxxxx00xxxx
xxxxxxx0000000000000000000000xx00000000x00000000000000
121
000000000000000000000000000000000000000000000000000000000000000xxxxxxxxxxxx0xxxxx
xxxxxxx00xxxxxxxxxxxx000000000xx00000000x0000000000000
122
0000000xxxxx0xxxxx00xxxxx000xxxxx000000000000000000000000000000000000000000000000
00000000000000000000000xxx0000000xx000000x000000000000
123
000000000000xxxxxx0xxxxxx00xxxxxx000xxxxxx000000000000000000000000000000000000000
000000000000000000000000xxx0000000xx000000x00000000000
124
000000000000000000xxxxxxx0xxxxxxx00xxxxxxx000xxxxxxx00000000000000000000000000000
0000000000000000000000000xxx0000000xx000000x0000000000
125
0000000000000000000000000xxxxxxxx0xxxxxxxx00xxxxxxxx000xxxxxxxx000000000000000000
00000000000000000000000000xxx0000000xx000000x000000000
126
000000000000000000000000000000000xxxxxxxxx0xxxxxxxxx00xxxxxxxxx000xxxxxxxxx000000
000000000000000000000000000xxx0000000xx000000x00000000
127
000000000000000000000000000000000000000000xxxxxxxxxx0xxxxxxxxxx00xxxxxxxxxx000xxx
xxxxxxx000000000000000000000xxx0000000xx000000x0000000
128
0000000000000000000000000000000000000000000000000000xxxxxxxxxxx0xxxxxxxxxxx00xxxx
xxxxxxx000xxxxxxxxxxx00000000xxx0000000xx000000x000000
129
000000000000000000xxxxxxx0xxxxxxx00xxxxxxx000xxxxxxx0000xxxxxxx000000000000000000
0000000000000000000000000xxxx000000xxx00000xx000x00000
130
0000000000000000000000000xxxxxxxx0xxxxxxxx00xxxxxxxx000xxxxxxxx0000xxxxxxxx000000
00000000000000000000000000xxxx000000xxx00000xx000x0000
131
000000000000000000000000000000000xxxxxxxxx0xxxxxxxxx00xxxxxxxxx000xxxxxxxxx0000xx
xxxxxxx00000000000000000000xxxx000000xxx00000xx000x000
132
000000000000000000000000000000000000000000xxxxxxxxxx0xxxxxxxxxx00xxxxxxxxxx000xxx
xxxxxxx0000xxxxxxxxxx0000000xxxx000000xxx00000xx000x00
```

```
133
000000000000000000000000xxxxxxxx0xxxxxxxx00xxxxxxxx000xxxxxxxx0000xxxxxxxx00000x
xxxxxxx0000000000000000000xxxxx00000xxxx0000xxx00xx0x0
134
000000000000000000000000000000000xxxxxxxxx0xxxxxxxxx00xxxxxxxxx000xxxxxxxxx0000xx
xxxxxxx00000xxxxxxxxx000000xxxxx00000xxxx0000xxx00xx0x
optimizing basis of the lattice via LLL, this can take a long time
LLL is done!
looking for independent vectors in the lattice
found them, using vectors 0 and 1
=== solution found ===
private key found:
968844854708094364290116428668387038621146739410171005860090983632734914915143562
69093
=== 6491.134232282639 seconds ===
```

所以我们找到了我们的私钥d，再进行进一步的代入求解，得到了密文m！

```
def rsa_decrypt(N, e, c, d):
    # Step 1: Decrypt the ciphertext to get the plaintext
    m = pow(c, d, N)

    # Step 2: Convert the plaintext to ASCII
    m_ascii = ''.join(chr((m >> 8 * i) & 0xff) for i in range((m.bit_length() +
7) // 8))

    return m, m_ascii
N =
0xd231f2c194d3971821984dec9cf1ef58d538975f189045ef8a706f6165aab4929096f61a3eb7dd8
021bf3fdc41fe3b3b0e4ecc579b4b5e7e035ffcc383436c9656533949881dca67c26d0e770e4bf62a
09718dbabc2b40f2938f16327e347f187485aa48b044432e82f5371c08f6e0bbde46c713859aec715
e2a2ca66574f3eb
e =
0x5b5961921a49e3089262761e89629ab6dff2da1504a0e5eba1bb7b20d63c785a013fd6d9e021c01
baf1b23830954d488041b92bca2fe2c92e3373dedd7e625da11275f6f18ee4aef336d0637505545f7
0f805902ddbacb21bb8276d34a0f6dfe37ede87dd95bb1494dbb5763639ba3984240f1178e32aa36e
e3c5fcc8115dde5
c =
0x6a88a8fa2b8f28d96284298bab2061efeb35e3a086370e19523c15c429f5d783b9d4f32e31a4029
16f45ad4f2760ab30e77177335af44756bfbeef0f168b5e0dc8c3ddf75d141c358969cca0e7c2b8ab
99ef8e33b031be1cbccd95b687682ac7b0dcc0d56f5651ee671d6358128d2e0801f247a6af4fe0dc5
e8fb199eba0780f
d =
968844854708094364290116428668387038621146739410171005860090983632734914915143562
69093
m, m_ascii = rsa_decrypt(N, e, c, d)
print("明文m:", m)
print("明文m的ASCII码:", m_ascii)
```

我们解得：

```
明文m:
467953840882317181499103662991214298422064486570574390714467505707261187314252509
296933072417732732809103029658160034371
明文m的ASCII码: Cryptography is typically bypassed,not penetrated.
```

所以最后，我们获得了明文的信息，英文是：

```
Cryptography is typically bypassed,not penetrated.
```

3(40 points)

> N3=0xf4c548636db62ffcc7ac4a0797952bea9a65bd426175af2435f72657e67ec8194667bfa94ce23c6f1e5baf3201867ab41701f6b8768e71009c41a3d5e9e7c109455341d549c7611f9f52851a2f017906aa9ccbedb95d238468e2c8577d30ecc4f158e3811fd5e2a6051443d468e3506bbc39bba710e34a604ac9e85d0feef8b3;
>
> e3=0x16f4b438ba14e05afa944f7da9904f8c78ea52e4ca0be7fa2b5f84e22ddd7b0578a3477b19b7bb4a7f825acc45da2dd10e62dbd94a3386b97d92ee817b0c66c1507514a7860b9139bc2ac3a4e0fe304199214da00a4ca82bfcb7b18253e7e6144828e584dac2dfb9a03fabaf2376ce7c269923fbb60fc68325b9f6443e1f896f;
>
> c3=0x26b1823cf836b226e2f5c90fdcd8420dbfcd02765b26e52ef3e5c0ab494c2f4650e475e280b0b5fff0d5016621186420b09e4706a5866e4a3319f23ef09d92c4e36acba39a0f6213fbe5ee1a736ce383e6e12351e6cbfd43f10a96b7fe34bdbaf948f2fb075d9063723c9f747fe6247ae9209e5d417faf2e37e6fee2eb863556;
>
> 已知信息: 私钥d3的取值范围为$[2^{299},2^{300}]$。

solution

4(40 points)

> N4=0xd46dd141810786e451320ca452b379024fd263501ae767760f3dcf34b79806b85e36b0fee538dac61a5872c37d051a8a026384d09f12b7e1adae7eb15c4d75878007ee0043c2186cf8999c59eb66f689f55baf190bd80e70bf47b553be76bd4efffc782a51b43314d54b83fc19461e1beb6021164f64723b505e5a619cb62335;
>
> e4=0x92fbeeef2d40eb125234cfe4c063c4607f12aec7e3014b32fb4600e58c4eac1ec485192a1b03745632f2966311ad68bd1e49dd9d08b2bff67f58e214c8d7bae0142559994c24e347ff7555c86aa30ccd03cf794e6f00eead7f15e24f33da61fae11ec81e4e09bcc76c1a0ed5ca8c2f512856cdb42470beee7111a2410188697d;
>
> c4=0x8c5e9db89f96d769f6514836407755caf71b7bc6f5db2246200b0f824dac7ea3be5ba022c0e191d76c69b7d20c7cad5c49e381479c7cbe7ba055ce8aec2cad1a19d42aa5c4b8c07c67e22c70289891d53c3d55dff50e506ec7fb480df44f9b3219f8c73e0702d8072e9f6aabed8bb5d35f583bea30ce850b154d4fd8c39e4fb8;
>
> 已知信息: 私钥d4的取值范围为$[2^{399},2^{400}]$,此外已知d4的汉明重量较轻，其最高310位比特（MSBs）汉明重量不超过5，剩余90位比特较为随机.

solution

5(30 points)

> N5=0x94eab94581f4931a5ea6aabcfe0598600fa3e0a06573887aed69e274f14484472dc3feaf50d4ef384e502f747f5605c1d2a4c8172b6ef134b7e96d6c383a9cb967ccbbd8b3647848d34928982a274999c2df00bd7dd11bf25acd61411e3395637e85dd84ecf785ff1027eed91f3976c8186e2e940edcb5fed8d759a5028b47a1;
>
> e5=0x124c552642ef2467aaecde51b0f3e1bee2ebe87bae39a956ad56cf7eec669cdc7b9664ea435b4c3492b8e610e0a182e1a76c7af443ca2962672b4e703c4f359cf8d88a67db77be2491b74bcdae58691b69e6ea06d067815b26fc0d669d8c06f11a728154dc8cdf983a056633fecadc417df4304625c3e6f91ec3d655a91a29e9;
>
> c5=0x63e09028c774513b5420236f8405f970c8d97c8347697c44f50b23e5cc964c921413b5e6742bb5ba7ef49f032e372f502babc0040f9c7cc2c9f4e27d18aefff0e764529ba70f6a7b22d525d0aaeb1d21432817b6b148b8143c80a6401a5c9adfecf0c033181bb076a2192a4866c5355c9e401fba78d5f22b9c1661c0065a1a28;
>
> 已知信息: 私钥d5的取值范围为[2^{511}, 2^{512}],此外d5的最高256位比特MSBs(记为dm)及最低176位比特LSBs(记为dl)取值均已知，但是中间80位比特(记为dx)取值未知，即满足如下关系：$d5 = dl + dx * 2^{176} + dm * 2^{256}$，具体取值如下：
>
> dl=0x2b26d177dc20ceea15de6e3c5a03207fb326a42d53a9；
>
> dm=0xacfad4bbb97a99b6bbc82c8b44a5260bcfe9c4a0acf437186ff4d5d1594cc5c1。

solution

6(40 points)

> N6=0x94e4c83c67c6d6e33d83cc2953df899e8c4b33894f653d5bbc84d7dd9058e6949221897f6e5b7b8bd9013f495c906862e401436e77be585474066f6c220751dd9b2b8be66f07ad7f090547a6e759e482ba263b941b32c27c62c4b558d96dda168b28c52e550b7d7ff145a5996c0b398714cf5ee8f0ea1a3d5b17c592f1c15275;
>
> e6=0x949b2e72766be1e83ee278a56bc86a2d3268b719507068ac62c6d249a810284edaac39335e8d699630887c13864f4cdf1c0c423b2f7ae88ccc60a827332e6c410800c7c7a1677918c28aa51086991d1290fc64b8e1b0f14b482f35d86139bb3491a59e2ad99dcd35bd129a44c3b8e2667e405dc2d307a5bb5a1504d7ded3bda3;

> c6=0x6fd6fae8ab4e95e622e5dad2921c6f12e911df08768abf2d10d212ad9a26e4c5ec71640d7a6b3488064fd424224bc2c762b956af95a3212de37a57d74c0299936f48ae3d8b8803e644e8d1306ab735c94fd815fe8c77982b32d51e9b6f3b3d4f3753810b61fb528c3e9eb774dabd93a3c5c9919ae3fb90e8e998ed3e7f949738;
>
> 已知信息: 私钥d6的取值范围为[2^{559}, 2^{560}],此外d6的最高123位比特MSBs(记为dm)取值未知，但其剩余437位比特(记为dl)取值已知，即满足如下关系：$d6 = dl + dm * 2^{437}$，dl具体取值如下: dl=0x6da211f0d34b。

solution

7(30 points)

> N7=0xaeb75bb97217271bf312a7897da81a544fe469ba0f1cf75304f2a5629717e1e3d0a9a28e71135443cc19f78c60dd3f7ea4ea28ae64657d5ac3b46e9755020de73cb5c4f89a682e0193916221bc8f4abb595f2c058bbb99e199a66144a9a9b258a74db847b2460107233280c94e854394595043f62bf77cd96c9ed3eca71b726d;
>
> e7=0x42b63e1113b4a84d0b037006a9bb729b52db495fa6b475bb64129a855a4ed6511792d0df946c5d7e22085d0db07bce5e408454a61c0cea51cf6d25e2455a2c6dc092e4b09bf4efb2157ffc1d1db3e969499479d721330ec4ac864e656318bc7bb9831a0dccf582406c87ae5d3ab9ffec351271dbb5481a0b6ed75a760b4f7e0d;
>
> c7=0xe1f90d9f115f9ba0b65ea8826ffec785bbe1b195fbb6f93c6ea28940f0d9b571930addb3e2714999ba5a19d17af22f1bc8da49f8b515ab03b6d276140b69fedf980d1aef78d0f3c0f6effdf2e92ce9195866f85672037537021178f8c65989b57f29de2c4c9306fe3e13aef29f962f86b8d5216907e85f28260b9f41cfe2651;
>
> 已知信息: 私钥d7与$phi7 = (p7 - 1) * (q7 - 1)$很接近,据估计,phi7-d7的取值范围为[$2^{267}$, 2^{268}]。

solution

8(30 points)

> N8=0xf12eac2099c4190a6f586bea0b4fc3f9dff4f23f0cb8e42cbeff950aa1df8a373c49df7974fb33b4b6619eadb2d6c01f80da1b433295b199df11b323114c439884eb31fa568bd747ae37079e885e2490c3b5a56d61b9d10533983ff78fe85e07876fe2ae07ae7ea1c71f0f9c2d6beccdcd8baf046a58549aec19d45d48d7d92d;
>
> e8=0xb8906f5097658f27cc448d98974d9e7ccd4e8a8f25a80007826c341dcb2ac42420f899e5a89045fbefd9163bc94e6f98b4953546203be4bec249031587a27dbf;

> c8=0x162a6dee8bcbe24698b9249137c2a157890910fa74a56e7d2792b5b4f29112aba03448995ff32ed24bec5118f7433212196d3f99e1c794b61395d8183e4658c9dc05953a87c069c9390773c7f885907840ebd29676afac7bf3374d54c81c4e404f09716b9885d243c41dc48db561f8291b88826cae32bfd575a472e523f455c4;
>
> 已知信息: 私钥d8的最低900位比特LSBs(记为dl)取值已知，剩余约124位比特(记为dm)取值未知，即满足如下关系：
>
> $d8 = dl + dm * 2^{900}$，dl具体取值如下:
>
> dl=0x4cbec287edc86c5b2a9e1975d64d2a24d3930075f0d445163c7b1ceec9ee0319fe1166af348b49004d2420b83bcb82d4879e93dba01ee76c5ca1b7141490465e824bdb5e91d04016c6bbbaa41c4470747ee8163f710b2d8adb8ab2168dcc996b5ab5f85a2269dc459379fb68848cec487

solution

9(30 points)

> e9=65537
>
> N9=0xcc5b706f373a79c680cec9527aac573fd435129cf16c23334085bf97832e5a6c78b633c2f244b12a62f87ec5295dd89fcf3c808c39e45a9afdbda2f8d2d0b50d61b685c0fe9eb41a7018a40f98892f96d738e2a4e740d4e507bcbd07f68c1ecb2ca10bd780ce65265a7e4da00f1031a5db9d038878a29a5ffefcaf2119720005;
>
> c9=0x20bac8a7d73a74c9913377846c13c3d2bd9f47e6df118d1486a96ed184ca9910e0f250500065cfb44105a41dff655364cabc3067ef3cd3d7d983e75c9303b786ac97507cfe803b788b12e582232028ca9772d05004aef194076ec442e3ee55e17fbb4a57f332b4393ac056c024141cc2b82f9dbc6d3c77f6eff20cd0ecc9cbab;
>
> 已知信息:
>
> 私钥d9的最低530位比特LSBs取值(记为dl)已知，剩余高位比特(记为dm)取值未知，即满足如下关系：
>
> $d9 = dl + dm * 2^{530}$，dl具体取值如下:
>
> dl=0x20142ae2802b877eb4dfa8a462e7d017c4d348181c367fd1a661ec9b6bbcca9dcb6601ccb6c10416b7f3c20129527346bbc136ee60f9945125cba03a9bba3720f7411

solution

10(25 points)(done)

> e10=65537
>
> N10=0x8d0df1ce526c39f9b057de462778a61ceda2049c7e32ee99d40baa4b22b7fd438e9ca1dfd7467684625add252095ee97c698199f4c5991279f6d3e74d4c14d01d137d42722df0d4565ff2a5275f9cac66dc4dfdf3304f85cbdc3d18eda1e32ac5d03675141a722ceefe0ea0533b53d7e50ed7eda1a1bbce47ed0ecb966f8678d

> c10=0x3b42fa3dc9089a21e9dabfe18297df47272f7e0ff59bf9bf16bc55e7fa70504c03fed56ca5ae93ac028f60ce5da3c145c6d181c5bd3c267288ec4765a19ca6b957b4535a1a185bd1b87d2e39b30e2430ed648175c29fdc1fde3787c426783dd66ba17f98b42ba13a7b3532970d0aa31b5ffa5f3eae243337a1668bae456bfbfb
>
> 已知信息: 私钥d10的取值范围为[a,b],a和b的具体取值如下
>
> a=0x19ffe8024fcf0320b3107f380f2e7deff71d561c4266c0f439d1aca20cd43d2aa6aed8679a16b2e1d3ff4ba3fc4da69cf34e35ead6f7eb79923960b9c83d9923e591b07b65275bf67f0b3d424cd7e6e6dd88ea39a5cfa27ecee61caaacc93e751dbb2a4c196f0ce0c36d44c35d6658d71b6c48b7b29400ab9161a0000000000
>
> b=0x19ffe8024fcf0320b3107f380f2e7deff71d561c4266c0f439d1aca20cd43d2aa6aed8679a16b2e1d3ff4ba3fc4da69cf34e35ead6f7eb79923960b9c83d9923e591b07b65275bf67f0b3d424cd7e6e6dd88ea39a5cfa27ecee61caaacc93e751dbb2a4c196f0ce0c36d44c35d6658d71b6c48b7b29400ab9161affffffffff

solution

使用coppersmith攻击即可获得结果。

```
import itertools
def small_roots(f, bounds, m=1, d=None):
    if not d:
        d = f.degree()

    R = f.base_ring()
    N = R.cardinality()

    k = ZZ(f.coefficients().pop(0))
    g = gcd(k, N)
    k = R(k/g)

    f *= 1/k
    f = f.change_ring(ZZ)

    vars = f.variables()
    G = Sequence([], f.parent())
    for k in range(m):
        for i in range(m-k+1):
            for subvars in itertools.combinations_with_replacement(vars[1:], i):
                g = f**k * prod(subvars) * N**(max(d-k, 0))
                G.append(g)

    B, monomials = G.coefficient_matrix()
    monomials = vector(monomials)

    factors = [monomial(*bounds) for monomial in monomials]
    for i, factor in enumerate(factors):
        B.rescale_col(i, factor)

    B = B.dense_matrix().LLL()
    B = B.change_ring(QQ)
    for i, factor in enumerate(factors):
        B.rescale_col(i, Integer(1)/factor)
```

```
    H = Sequence([], f.parent().change_ring(QQ))
    for h in filter(None, B*monomials):
        H.append(h)
        I = H.ideal()
        if I.dimension() == -1:
            H.pop()
        elif I.dimension() == 0:
            roots = []
            for root in I.variety(ring=ZZ):
                root = tuple(R(root[var]) for var in f.variables())
                roots.append(root)
            return roots

    return []


c1=0x3b42fa3dc9089a21e9dabfe18297df47272f7e0ff59bf9bf16bc55e7fa70504c03fed56ca5ae
93ac028f60ce5da3c145c6d181c5bd3c267288ec4765a19ca6b957b4535a1a185bd1b87d2e39b30e2
430ed648175c29fdc1fde3787c426783dd66ba17f98b42ba13a7b353297 0d0aa31b5ffa5f3eae2433
37a1668bae456bfbfb
leak1=0x19ffe8024fcf0320b3107f380f2e7deff71d561c4266c0f439d1aca20cd43d2aa6aed8679
a16b2e1d3ff4ba3fc4da69cf34e35ead6f7eb79923960b9c83d9923e591b07b65275bf67f0b3d424c
d7e6e6dd88ea39a5cfa27ecee61caaacc93e751dbb2a4c196f0ce0c36d44c35d6658d71b6c48b7b29
400ab9161a0000000000
n1=0x8d0df1ce526c39f9b057de462778a61ceda2049c7e32ee99d40baa4b22b7fd438e9ca1dfd746
7684625add252095ee97c698199f4c5991279f6d3e74d4c14d01d137d42722df0d4565ff2a5275f9c
ac66dc4dfdf3304f85cbdc3d18eda1e32ac5d03675141a722ceefe0ea0533b53d7e50ed7eda1a1bbc
e47ed0ecb966f8678d
e1=65537


k1 = e1*leak1 // n1 + 1
PR.<x,y> = PolynomialRing(Zmod(e1*leak1))
f = 1 + k1*((n1+1)-y) - e1*x
bounds = (2^160,2^513)

res = small_roots(f,bounds,m=2,d=2)
leak = int(res[0][1])

PR.<x> = PolynomialRing(RealField(1000))
f = x*(leak-x) - n1
ph = int(f.roots()[0][0])


PR.<x> = PolynomialRing(Zmod(n1))
f = ph + x
res = f.small_roots(X=2^(160+6), beta=0.499,epsilon=0.02)[0]
p1 = int(ph + res)
q1 = n1 // p1
print(p1)
print(q1)
```

攻击，求出p和q

```
p=84147670891338737694689450322242729025436170282367766217749984657082681139159059122107822462178302965531773032081674130167421032066049038127148395512045890
q=11771189496077447472415256585690278237732856915764247194903035840506904720003320569381951682128930227151469309640903353145013377646752106391473373652877601
```

然后套用RSA解密脚本即可，求出m的值

```
n=0x8d0df1ce526c39f9b057de462778a61ceda2049c7e32ee99d40baa4b22b7fd438e9ca1dfd7467684625add252095ee97c698199f4c5991279f6d3e74d4c14d01d137d42722df0d4565ff2a5275f9cac66dc4dfdf3304f85cbdc3d18eda1e32ac5d03675141a722ceefe0ea0533b53d7e50ed7eda1a1bbce47ed0ecb966f8678d
p=84147670891338737694689450322242729025436170282367766217749984657082681139159059122107822462178302965531773032081674130167421032066049038127148395512045890
q=11771189496077447472415256585690278237732856915764247194903035840506904720003320569381951682128930227151469309640903353145013377646752106391473373652877601
c=0x3b42fa3dc9089a21e9dabfe18297df47272f7e0ff59bf9bf16bc55e7fa70504c03fed56ca5ae93ac028f60ce5da3c145c6d181c5bd3c267288ec4765a19ca6b957b4535a1a185bd1b87d2e39b30e2430ed648175c29fdc1fde3787c426783dd66ba17f98b42ba13a7b3532970d0aa31b5ffa5f3eae243337a1668bae456bfbfb
e=65537
from gmpy2 import *
phi=(p-1)*(q-1)
d=invert(e,phi)
m=pow(c,d,n)
print(m)
```

m的值为`820307570753893249794408475962305073085942139245561017669`，转化为16进制就是：`21746920646e756f662065766168204920 2c616b65727545`

然后我们解密即可。

```
m=820307570753893249794408475962305073085942139245561017669
m_ascii = ''.join(chr((m >> 8 * i) & 0xff) for i in range((m.bit_length() + 7) // 8))
print(m_ascii)
```

解得：

```
Eureka, I have found it!
```

这就是我们要求的密文。

11(40 points)(done)

> e11=65537
>
> N11=0xcb5645c59c402b0edcf96cbd6a7308b64aac2f37a3c6f96be7c421c4b7f0a4adbdecd88cbea1128352fb21baae583fe4ceb3fc93c4905803ad3e9214ada050d5c0ff785a13a5c9157c3154ad8d7015a2d239fe13ef836d3279c5cd5dc96013ac40f372a9c9226d2f5fe73f312c56e11d9cdfbf9fb0db627ac1a752f5f0bd2b29

> c11=0x84e4aa0be481e9c4bbd4c71dba5235cccd8312759de35c326c7e4cdda494196d1c0cae298240942af3082fac215965999c908a79bf07e093ee0c402e727a09a1c1f13831875d66ebbc3f89507163de90339af055bcd7d778574775214accfbd8ae20001f27bc196b974cb3ac215fea3debb7b17a21a8ebb1a9880a671539ef21
>
> 已知信息: 私钥d11的取值范围为[a,b],a和b的具体取值如下
>
> a=0x4f77b72b04e6fb2d02e5a43edef4784a2e22df0d42bfc7c9093a58ec35eb21a11962103be960b0088d0cc2e0dfb473bc2ba0a22cea1c73997442c8fab5e4bad22cd131055b0382eb9264ad40ec8257abaff11b33b173ffd0168039bf40dc203eb325d884d2845fd2b5a37f41a0f64183db0c256c24450000000000000000000000
>
> b=0x4f77b72b04e6fb2d02e5a43edef4784a2e22df0d42bfc7c9093a58ec35eb21a11962103be960b0088d0cc2e0dfb473bc2ba0a22cea1c73997442c8fab5e4bad22cd131055b0382eb9264ad40ec8257abaff11b33b173ffd0168039bf40dc203eb325d884d2845fd2b5a37f41a0f64183db0c256c2445ffffffffffffffffffff

solution

这题的脚本我们还是使用原来的第十题的脚本，调整参数后进行d的coppersmith攻击即可。

```
#from Crypto.Util.number import *
#from tqdm import *
import itertools

def small_roots(f, bounds, m=1, d=None):
    if not d:
        d = f.degree()

    R = f.base_ring()
    N = R.cardinality()

    k = ZZ(f.coefficients().pop(0))
    g = gcd(k, N)
    k = R(k/g)

    f *= 1/k
    f = f.change_ring(ZZ)

    vars = f.variables()
    G = Sequence([], f.parent())
    for k in range(m):
        for i in range(m-k+1):
            for subvars in itertools.combinations_with_replacement(vars[1:], i):
                g = f**k * prod(subvars) * N**(max(d-k, 0))
                G.append(g)

    B, monomials = G.coefficient_matrix()
    monomials = vector(monomials)

    factors = [monomial(*bounds) for monomial in monomials]
    for i, factor in enumerate(factors):
        B.rescale_col(i, factor)

    B = B.dense_matrix().LLL()
```

```
    B = B.change_ring(QQ)
    for i, factor in enumerate(factors):
        B.rescale_col(i, Integer(1)/factor)

    H = Sequence([], f.parent().change_ring(QQ))
    for h in filter(None, B*monomials):
        H.append(h)
        I = H.ideal()
        if I.dimension() == -1:
            H.pop()
        elif I.dimension() == 0:
            roots = []
            for root in I.variety(ring=ZZ):
                root = tuple(R(root[var]) for var in f.variables())
                roots.append(root)
            return roots

    return []


c1=0x84e4aa0be481e9c4bbd4c71dba5235cccd8312759de35c326c7e4cdda494196d1c0cae298240
942af3082fac215965999c908a79bf07e093ee0c402e727a09a1c1f13831875d66ebbc3f89507163d
e90339af055bcd7d778574775214accfbd8ae20001f27bc196b974cb3ac215fea3debb7b17a21a8eb
b1a9880a671539ef21
leak1=0x4f77b72b04e6fb2d02e5a43edef4784a2e22df0d42bfc7c9093a58ec35eb21a11962103be
960b0088d0cc2e0dfb473bc2ba0a22cea1c73997442c8fab5e4bad22cd131055b0382eb9264ad40ec
8257abaff11b33b173ffd0168039bf40dc203eb325d884d2845fd2b5a37f41a0f64183db0c256c244
500000000000000000000
n1=0xcb5645c59c402b0edcf96cbd6a7308b64aac2f37a3c6f96be7c421c4b7f0a4adbdecd88cbea1
128352fb21baae583fe4ceb3fc93c4905803ad3e9214ada050d5c0ff785a13a5c9157c3154ad8d701
5a2d239fe13ef836d3279c5cd5dc96013ac40f372a9c9226d2f5fe73f312c56e11d9cdfbf9fb0db62
7ac1a752f5f0bd2b29
e1=65537


k1 = e1*leak1 // n1 + 1
PR.<x,y> = PolynomialRing(Zmod(e1*leak1))
f = 1 + k1*((n1+1)-y) - e1*x
bounds = (2^320,2^963)

res = small_roots(f,bounds,m=2,d=2)
leak = int(res[0][1])

PR.<x> = PolynomialRing(RealField(1000))
f = x*(leak-x) - n1
ph = int(f.roots()[0][0])


PR.<x> = PolynomialRing(Zmod(n1))
f = ph + x
res = f.small_roots(X=2^(220+6), beta=0.499,epsilon=0.02)[0]
p1 = int(ph + res)
q1 = n1 // p1
print(p1)
print(q1)
```

解出p和q，得到：

```
p=11309138517753609601189729920172165631612370294087077281755734175142319827761966692830479699631439962525444124861722707903014820730766996645318082582620491
q=12625904132876024391053747255529472882205295003498735138849760876178996107401873371263341157195294988792537890500933341025075865608380995322266553378244827
```

然后我们就使用正常的解密脚本进行求解即可。

```
e=65537
n=0xcb5645c59c402b0edcf96cbd6a7308b64aac2f37a3c6f96be7c421c4b7f0a4adbdecd88cbea11
28352fb21baae583fe4ceb3fc93c4905803ad3e9214ada050d5c0ff785a13a5c9157c3154ad8d7015
a2d239fe13ef836d3279c5cd5dc96013ac40f372a9c9226d2f5fe73f312c56e11d9cdfbf9fb0db627
ac1a752f5f0bd2b29
c=0x84e4aa0be481e9c4bbd4c71dba5235cccd8312759de35c326c7e4cdda494196d1c0cae2982409
42af3082fac215965999c908a79bf07e093ee0c402e727a09a1c1f13831875d66ebbc3f89507163de
90339af055bcd7d778574775214accfbd8ae20001f27bc196b974cb3ac215fea3debb7b17a21a8ebb
1a9880a671539ef21
p=113091385177536096011897299201721656316123702940870772817557341751423198277619666
92830479699631439962525444124861722707903014820730766996645318082582620491
q=126259041328760243910537472555294728822052950034987351388497608761789961074018733
71263341157195294988792537890500933341025075865608380995322266553378244827
from gmpy2 import *
phi=(p-1)*(q-1)
d=invert(e,phi)
m=pow(c,d,n)
print(m)
m_ascii = ''.join(chr((m >> 8 * i) & 0xff) for i in range((m.bit_length() + 7) //
8))
print(m_ascii)
```

解得：

```
871459330528493685900356342244447392885109122774019102948898640050654591861594334
271960704663752901134803717028296430848471791111286785166702335793931444314976572
456442657548331883149982403838356072713277556885620603888164587804444614989534913
9015593369673
I have discovered a truly marvelous proof of this, which however the margin is
not large enough to contain.
```

所以我们本题的密文是：

```
I have discovered a truly marvelous proof of this, which however the margin is
not large enough to contain.
```

12(40 points)

> e12=65537

> N12=0x9fac422a93f6e486e3ddae088bb5f5d06dec183ab81290042a9c98c53352961a00db3e9def7adff842381a395cedf1d06294f0b63457133e4e44cabb7633c562dcbfffdffe541d66c46ddf6a28b686c478300bcf31945f2a6495f140e64f78fa5cd47d1885233f175f28e38f1bfc422a6853ca19a7dd47a291a9e7de78a67bf1
>
> c12=0x35476c9d0e5ad9d364ea31d8f6628b92a4f6307b1fef754e49286bc7f53ea8cd013a7ebf2a21b2327af44498d267e19526c2051a02f22cca9cab567f7ceefe5003137e396c23742370e14ec2c6a90943ca848908e87420f560d34eae4635475effa867722276710c6f4b6cb9b295777d62f3f03c57603ac815072864aadbf041
>
> 已知信息:素因子q12是模数$N12 = p12 * q12$与整数N的部分近似公因子，即$N = k * q12 + r$,其中$2^{511} < k < 2^{512}$ 和$2^{255} < r < 2^{256}$均为正整数且N的具体取值如下:
>
> N=0x8199f8d487909988daf7d692ce8b1ffb4c37aa8010c8ca337ae4398c521383dc51007645cb6a1743c9b52ec5808e9e0e6f54d5fbb143cf81651240beab342dfb4622f073c4f8ab968dd5c8d4be3b7dd55c2cb9ef9c06294cd87e5fa29e38279c850f03687dc8c83c68104dca88e3a5c8559a01c040e7d5107e4a9f2385429f90

solution