BUUCTF-CRYPTO-强网杯2019 Copperstudy

原创

大熊何在 于 2020-11-01 00:26:27 发布 1079 收藏 3

分类专栏： CRYPTO

本文链接：https://blog.csdn.net/zippo1234/article/details/109409929

BUUCTF-CRYPTO-强网杯2019 Copperstudy

每天一题，只能多不能少

[强网杯2019] Copperstudy

题目分析

RSA套娃，各种类型的RSA，对于我现在的水平来说真的是难于上青天。鉴于道奇时隔32年又夺冠军（28日的事了），今天就先记录下来。以后慢慢看

1. hash破解
2. e=3
3. 已知p高位攻击
4. 已知d低位攻击
5. 低加密指数广播攻击
6. m高位相同，高位相同的m，短填充攻击Coppersmith Shortpad Attack
7. Boneh and Durfee attack

开始

1.题目

给出一个远端环境，nc过去即可

2.第0层

```
[+]proof: skr=os.urandom(8)
[+]hashlib.sha256(skr).hexdigest()=b9f5a36134ba3b3b9a41c3ee519899f39fd85f231d9cb2d6c34415fcebe0aa8c
[+]skr[0:5].encode('hex')=13a03f1f32
[-]skr.encode('hex')=
```

破解一个原文8个字符的sha256，已知前5个字符。
使用hashcat爆破，记得加上–force参数，至少我这里是需要加上的，不然一直提示“No devices found/left.”

```
hashcat64.exe -a 3 --hex-salt -m 1420 b9f5a36134ba3b3b9a41c3ee519899f39fd85f231d9cb2d6c34415fcebe0aa8c:13a03f1f3
2 --potfile-disable ?b?b?b  -o res3.txt --outfile-format=2 --force
```

拼接即可

```
#!python3
# -*- coding: utf-8 -*-
# @Time : 2020/10/31 22:33
# @Author : A.James
# @FileName: tt0.py
import os
path="hashcat-5.1.0\\"
str1='b9f5a36134ba3b3b9a41c3ee519899f39fd85f231d9cb2d6c34415fcebe0aa8c'
str2='13a03f1f32'
with open(path+"res3.txt", 'r') as f:
    lines = f.readlines()
    last_line = lines[-1]
    #print(last_line)
    if(last_line[0:4]=="$HEX"):
        print(str2+last_line[5:11])
    else:
        print(str2+hex(last_line))
```

得到

```
13a03f1f321bdb17
```

输入，进入第1层

3.第1层

给出

```
[+]Generating challenge 1


  [+]n=131120618206856432396638311669283271195794258306324585688015444065067694612795909627723402491835694375593
942006355261836986045823857693811595637108236894172744795496275960953986211829958914545169537220250689262935125053831252275791697789466313699617535878563445822576836723132303786033240053377889139024340234318870614543685661007476185825902703859182046561560890535197095360019069640086357085106725502195468940060914835203554360910538663127184313184987836377127738784237774673166058655162481762487806371326158078862720298437701868334257920491081874873382378508062037282173748487992504198596468710570962970206709042 11

[+]e=3

[+]m=random.getrandbits(512)

[+]c=pow(m,e,n)=159875547240031002953260760364131636343986009476950968578039379989694417630147317203751961040107945558680690243936479660405932582678884637321844950207094575600430505771989883637547037416360880894724889710503246541621666576783765571104927037122863068684372846622488755082716244202626216334093533372170526743279026851 7

[+]((m>>72)<<72)=251918859427175920575786448609760554013540750157107862723884944356121905775184317054026184267723968190873 6

[-]long_to_bytes(m).encode('hex')=
```

e=3的套路直接上

```python
#!python3
# -*- coding: utf-8 -*-
# @Time : 2020/10/31 18:26
# @Author : A.James
# @FileName: tt1.py

from Crypto.Util.number import *
import gmpy2
import binascii
c=159875547240031002953260760364131636343986009476950968578039379989694417630147317203751961040107945558680690243936479660405932582678884637321844950207094575600430505771989883637547037416360880894724889710503246541621666576783765571104927037122863068684372846622488755082716244202626216334093533372170526743279026851 7
m = gmpy2.iroot(c,3)[0]
print(long_to_bytes(m))
mm ='FLAG{2^8rsa7589693fc689c77c5f5262d654272427}'
print(binascii.hexlify(long_to_bytes(m)))
```

得到

```
b'464c41477b325e3872736137353839363933666336383963373763356635323632643635343237323432377d'
```

输入hex部分，进入第2关

4.第2层

给出：

```
[+]Generating challenge 2


  [+]n=127846257290327895927666252030740181013549177514929526850838088255042218168473109104475321336169542622712
05877651255598995305639194329607493047941212754523879402744065076183778452640602625242851184095546100200565113011
66901610538089503844589968815742665739925263579545074913979782786041025247313930593034763501677382378226472464425
83648253315002592305154443133050252204383387258048314259457180218932159901672574126025417079339377729314501052566
86561904427613648184843619301241414264343057368192416551134404100386155751297424616254697041043851852081071306211
94629919698491236682483211303822317692508651902276300091817592l9

[+]e=65537

[+]m=random.getrandbits(512)

[+]c=pow(m,e,n)=62782408615711924505647887580059895955377425016167078750608325396078823073758876178738568612582876
76566561756788790422803083953531798758960876153450000312824716423377479478423151821280427005640456571042661393826
64302998015421153393879729263551292024543756422702956470022959537221269172084619081368498693930550456153543628170
30632420626621634838670700866112871743142623748651130976728617551823862023050720195286726128388098686875267654961
39587852889149894292245828492183954716722954100368588818363633648851642769832373122358315918580449083693768554841
27614933545955544787160352042318378588039587911741028067576722790778

[+]((p>>128)<<128)=97522826022187678545924975588711975512906538181361325096919121233043973599759518562689050415761
48571670561514964176898283825540359433129365122439559074713315212804295006210315656444015508888259264404606920840
53603243720571408903175188021300811980600935768415380089605603913803956970989644118217166645069086 72

[-]long_to_bytes(m).encode('hex')=
```

这里的运算时<< >>位移运算，也就是p右移128位再左移128位，右移直接吞左移不足位补零，也就是说这里给出p的最后128位去除补0并转换为10进制的值。

求p、q

```python
#!python3
# -*- coding: utf-8 -*-
# @Time : 2020/10/31 22:50
# @Author : A.James
# @FileName: tt2.py
p4 = 28659382766326598087551095496731621907344817003090006257991553498670648690645830788439163308197597488986880
80930841411966107892292623387422078161173619278949047765526765410366732440903341647984431629329143559667704508940
47111793505063044583029134192122352988382684883337
n = 12784625729032789592766625203074018101354917751492952685083808825504221816847310910447532133616954262271205877
76512555989953056391943296074930479412127545238794027440650761837784526406026252428511840955461002005651130166901
61610538089503844589968815742665739925263579545074913979782786041025247313930593034763501677382378226472464258364
82533150025923051544431330502522043833872580483142594571802189321599016725741260254170793393772931450105256865619
04427613648184843619301241414264343057368192416551134404100386155751297424616254697041043851852081071306219462991
96984912366824832113038223176925086519022763000918175921 9
pbits = 1024
kbits = pbits - p4.nbits()
print (p4.nbits())
p4 = p4 << kbits
PR.<x> = PolynomialRing(Zmod(n))
f = x + p4
x0 = f.small_roots(X=2^kbits, beta=0.4)[0]
print ("x:" ,hex(int(x0)))
p = p4+x0
print ("p: ", hex(int(p)))
assert n % p == 0
q = n/int(p)
print ("q: ", hex(int(q)))
```

得到

```
p=0x8ae08a8ccda172cc5768c98c935b06a185a5f86f1020ce864929dd61d0d6511141e94f589b4c10754fe4b278207414caedc5a0c47ca0
91ef3dad80c15b05776d4c574759b50106585973e7f7cda6d01db4bcfbb671151069287f276bb6c18d04cab2dfccf70a72a5edbc23fd636d
a989cb609b9f64429a11ce179a7e63951f07
q=0xbaaeffd0d50b8bbd0d9fcb6086388c65473593441b5551f0fdfa8d3a25e7bc7a3a905faeee4ef188c3f249aacbfa9f779efdc23a6154
2b66ad1ef884152ad7039a090617381793da84eda62f39959f1ed3c71e9fccfbf19c62d53b2a5a2e14d472b5dd10c7c9a0aa051ee14baff0
349b96caabdd03516aa7c8b7a692431f11b5
```

有了p和q就是正常的RSA了。

```python
#!python3
# -*- coding: utf-8 -*-
# @Time : 2020/10/31 23:02
# @Author : A.James
# @FileName: tt2-2.py
import gmpy2
import binascii

x=0xcb609b9f64429a11ce179a7e63951f07
p=0x8ae08a8ccda172cc5768c98c935b06a185a5f86f1020ce864929dd61d0d6511141e94f589b4c10754fe4b278207414caedc5a0c47ca0
91ef3dad80c15b05776d4c574759b50106585973e7f7cda6d01db4bcfbb671151069287f276bb6c18d04cab2dfccf70a72a5edbc23fd636d
a989cb609b9f64429a11ce179a7e63951f07
q=0xbaaeffd0d50b8bbd0d9fcb6086388c65473593441b5551f0fdfa8d3a25e7bc7a3a905faeee4ef188c3f249aacbfa9f779efdc23a6154
2b66ad1ef884152ad7039a090617381793da84eda62f39959f1ed3c71e9fccfbf19c62d53b2a5a2e14d472b5dd10c7c9a0aa051ee14baff0
349b96caabdd03516aa7c8b7a692431f11b5
n = 127846257290327895927666252030740181013549177514929526850838088255042218168473109104475321336169542622712058
776512555989953056391943296074930479412127545238794027440650761837784526406026252428511840955461002005651130166 9
016105380895038445899688157426657399252635795450749139797827860410252473139305930347635016773823782264724642583 6
482533150025923051544431330502522043833872580483142594571802189321599016725741260254170793393772931450105256865
619044276136481848436193012414142643430573681924165511344041003861557512974246162546970410438518520810713062194 6
299196984912366824832113038223176925086519022763000918175921 9
c = 627824086157119245056478875800598959553774250161670787506083253960788230737588761787385686125828765665617567
887904228030839535317987589608761534500003128247164233774794784231518212804270056404565710426613938264302998015 4
211533938797292635512920245437564227029564700229595372212691720846190813684986939305504561535436281703063242062 6
621634838670700866112871743142623748651130976728617551823862023050720195286726128388098686875267654961395878528 8
914989429224582849218395471672295410036858881836363364885164276983237312235831591858044908369376855484127614933 5
459555447871603520423183785880395879117410280675767227907 78
```

```
e = 65537
phi = (p-1)*(q-1)
d = gmpy2.invert(e,phi)
m = pow(c,d,n)
print(hex(m))
```

得到：

```
464c41477b325e3872736136653237376633353564626536646133656464366633353664326423664366 67d
```

输入进入下一层

5.第3层

给出：

```
[+]Generating challenge 3


  [+]n=928965239796164317835697626459459187511623211851597903020857680957632483571461988826411606786230698570118
3292917998762349226785230417889446148629586409187134133949087068911027972028341597634220847612641493391402643666
6789270209690168581379143120688241413470569887426810705898518783625903350928784794371176183

[+]e=3

[+]m=random.getrandbits(512)

[+]c=pow(m,e,n)=561643781850494024042877639722806302954101741836490548059473295048929799211318523212813173263065
0644414569901278854771809137138969896971883076112007635963426288091241779703804951064723733725103707036927859619
1506725812511682495575589039521646062521091457438869068866365907962691742604895495670783101319608530

[+]d&((1<<512)-1)=787673996295376297668171075170955852109814939442242049800811601753001897317556022653997651874 8
97208487913321031340711138331360350633965420642045383644955

[-]long_to_bytes(m).encode('hex')=
```

已知d的低位，上脚本

```
#!python3
# -*- coding: utf-8 -*-
# @Time : 2020/10/31 23:12
# @Author : A.James
# @FileName: tt3.py
def partial_p(p0, kbits, n):
    PR.<x> = PolynomialRing(Zmod(n))
    nbits = n.nbits()

    f = 2^kbits*x + p0
    f = f.monic()
    roots = f.small_roots(X=2^(nbits//2-kbits), beta=0.3)  # find root < 2^(nbits//2-kbits) with factor >= n^0.3
    if roots:
        x0 = roots[0]
        p = gcd(2^kbits*x0 + p0, n)
        return ZZ(p)

def find_p(d0, kbits, e, n):
    X = var('X')

    for k in range(1, e+1):
        results = solve_mod([e*d0*X - k*X*(n-X+1) + k*n == X], 2^kbits)
        for x in results:
            p0 = ZZ(x[0])
            p = partial_p(p0, kbits, n)
            if p:
                return p


if __name__ == '__main__':
    n = 92896523979616431783569762645945918751162321185159790302085768095763248357146198882641160678623069857011
832929179987623492267852304178894461486295864091871341339490870689110279720283415976342208476126414933914026436 6
667892702096901685813791431206882414134705698874268107058985187836259033509287847943711761 83
    e = 3
    d = 78767399629537629766817107517095585210981493944224204980081160175300189731755602265399765187489720848791
332103134071113833136035063396542064204538364495 5

    nbits = n.nbits()
    kbits = d.nbits()

    print ("lower %d bits (of %d bits) is given" % (kbits, nbits))

    p = find_p(d, kbits, e, n)
    q = n//p
    print ("d0 = %d" % d)
    print ("d = %d" % inverse_mod(e, (p-1)*(q-1)))
```

得到d后就是正常的RSA解密了：

```python
#!python3
# -*- coding: utf-8 -*-
# @Time : 2020/10/31 23:14
# @Author : A.James
# @FileName: tt3-2.py
d = 619310159864109545223798417639639458341082141234398602013905120638421655714307992550941071190820465713412219
527866584156615119015361192629743241972427279013618535190600991760957183983415465217097531407150904237754135904631597159144976253463643630503169317797271549882695768084767963809412279563168024113702670
n = 928965239796164317835697626459459187511623211851597903020857680957632483571461988826411606786230698570118329291799876234922678523041788944614862958640918713413394908706891102797202834159763422084761264149339140264366667892702096901685813791431206882414134705698874268107058985187836259033509287847943711761830
e = 3
c = 561643781850494024042877639722806302954101741836490548059473295048929799211318523212813173263065064441456990127885477180913713896989697188307611200763596342628809124177970380495106472373372510370703692785961915067258125116824955755890395216460625210914574388690688663659079626917426048954956707831013196085300
m = pow(c,d,n)
print(hex(m))
```

得到

```
464c41477b325e3872736135616230383637343566366563373435363139613862363566653465633536307d
```

输入进入下一层。

6.第4层

给出：

```
[+]e=3

[+]m=random.getrandbits(512)



[+]n1=78642188663937191491235684351005990853149481644703243255021321296087539054265733392095095639539412823093600710316645130404423641473150336492175402885270861906530337207734106926328737198871118125840680572148601743121884788919989184318198417654263598170932154428514561079675550090698019678767738203477097731989


[+]c1=pow(m,e,n1)=23419685303892339080979695469481275906709035609088426118328601771163101123641599051556995351678670765521269546319724616458499631461037359417701720430452076029312714313804716888119910334476982840024696320503747736428099717113471541651211596481005191146454458591558743268791485623924245960696651150688621664860



[+]n2==98174485544103863705821086588292917749386955237408645745685476234349659452606822650329076955303471252833860010724515777826660887118742978051231030080666542833950748806944312437614585352818344599399156268450521239843157288915059003487783576003027303399985723834248634230998110618288843582573006048070816520647


[+]c2=pow(m,e,n2)=72080679612442543693944655041130370753964497034378634203383617624269927191363529233872659451561571441107920350406295389613006330637565645758727103723546610079332161151567096389071050158035757745766399510575237344950873632114050632573903701015749830874081198250578516967517980592506626547273178363503100507676



[+]n3=91638855323231795590642755267985988356764327384001022396221901964430032527111968159623063760057482761918901490239790230176524505469897183382928646349163030620342744192731246392941227433195249399795012672172947919435254998997253131826888070173526892674308708289629739522194864912899817994807268945141349669311


[+]c3=pow(m,e,n3)=22149989692509889061584875630258740744292355239822482581889060656197919681655781672277545701325284646570773490123892626601106871432216449814891757715588851851459306683123591338089745675044763551335899599807235257516935037356212345033087798267959242561085752109746935300735969972249665700075907145744305255616

[-]long_to_bytes(m).encode('hex')=
```

低加密指数广播攻击

```python
#!python3
# -*- coding: utf-8 -*-
# @Time : 2020/10/31 23:17
# @Author : A.James
# @FileName: tt4.py
import gmpy2
from functools import reduce
from Crypto.Util.number import *
import binascii

def chinese_remainder(n, a):
    sum = 0
    prod = reduce(lambda a, b: a * b, n)
    for n_i, a_i in zip(n, a):
        p = prod // n_i
        sum += a_i * gmpy2.invert(p, n_i) * p
    return int(sum % prod)
n1=78642188663937191491235684351005990853149481644703243255021321296087539054265733392095095639539412823093600710316645130404423641473150336492175402885270861906530337207734106926328737198871118125840680572148601743121884788919989184318198417654263598170932154428514561079675550090698019678767738203477097731989
c1=23419685303892339080979695469481275906709035609088426118328601771163101123641599051556995351678670765521269546319724616458499631461037359417701720430452076029312714313804716888119910334476982840024696320503747736428099717113471541651211596481005191146454458591558743268791485623924245960696651150688621664860
n2=98174485544103863705821086588292917749386955237408645745685476234349659452606822650329076955303471252833860010724515777826660887118742978051231030080666542833950748806944312437614585352818344599399156268450521239843157288915059003487783576003027303399985723834248634230998110618288843582573006048070816520647
c2=72080679612442543693944655041130370753964497034378634203383617624269927191363529233872659451561571441107920350406295389613006330637565645758727103723546610079332161151567096389071050158035757745766399510575237344950873632114050632573903701015749830874081198250578516967517980592506626547273178363503100507676
n3=91638855323231795590642755267985988356764327384001022396221901964430032527111968159623063760057482761918901490239790230176524505469897183382928646349163030620342744192731246392941227433195249399795012672172947919435254998997253131826888070173526892674308708289629739522194864912899817994807268945141349669311
c3=22149989692509889061584875630258740744292355239822482581889060656197919681655781672277545701325284646570773490123892626601106871432216449814891757715588851851459306683123591338089745675044763551335899599807235257516935037356212345033087798267959242561085752109746935300735969972249665700075907145744305255616

n=[n1,n2,n3]
c=[c1,c2,c3]
ans=chinese_remainder(n, c)
ans=gmpy2.iroot(ans,3)[0] # e = 3
print(binascii.hexlify(long_to_bytes(ans)))
```

得到：

```
464c41477b325e3872736138633566336366663462633039353334396665633635666332323633653837387d
```

输入进入下一层

7.第5层

给出：

```
[+]Generating challenge 5


  [+]n= 113604829563460357756722229849309932731534576966155520277171862442445354404910882358287832757024693652075211204635679309777620586814014894544893424988818766425089667672311645586528776360047956843961901352792631908859388801090108188344342619580661377758180391734771694803991493164412644148805229529911069578061


[+]e=7


[+]m=random.getrandbits(512)


[+]c=pow(m,e,n)=112992730284209629010217336632593897028023711212853788739137950706145189880318698604512926758021533447981943498594790549326550460216939216988828130624120379925895123186121819609415184887470233938291227816332249857236198616538782622327476603338806349004620909717360739157545735826670038169284252348037995399308


[+]x=pow(m+1,e,n)=112992730284209629010217336632593897028023711212853788739137950706145189880318698604512926758021552486915464025361447529153776277710423467951041523831865232164370127602772602643378592695459331174613894578701940837730590029577336924367384969935652616989527416027725713616493815764725131271563545176286794438175


[-]long_to_bytes(m).encode('hex')=
```

高位相同的m，短填充攻击Coppersmith Shortpad Attack

```python
#!python3
# -*- coding: utf-8 -*-
# @Time : 2020/10/31 23:24
# @Author : A.James
# @FileName: tt5.py
def short_pad_attack(c1, c2, e, n):
    PRxy.<x,y> = PolynomialRing(Zmod(n))
    PRx.<xn> = PolynomialRing(Zmod(n))
    PRZZ.<xz,yz> = PolynomialRing(Zmod(n))

    g1 = x^e - c1
    g2 = (x+y)^e - c2

    q1 = g1.change_ring(PRZZ)
    q2 = g2.change_ring(PRZZ)

    h = q2.resultant(q1)
    h = h.univariate_polynomial()
    h = h.change_ring(PRx).subs(y=xn)
    h = h.monic()

    kbits = n.nbits()//(2*e*e)
    diff = h.small_roots(X=2^kbits, beta=0.5)[0]  # find root < 2^kbits with factor >= n^0.5

    return diff

def related_message_attack(c1, c2, diff, e, n):
    PRx.<x> = PolynomialRing(Zmod(n))
    g1 = x^e - c1
    g2 = (x+diff)^e - c2

    def gcd(g1, g2):
```

```
    def gcd(g1, g2):
        while g2:
            g1, g2 = g2, g1 % g2
        return g1.monic()

    return -gcd(g1, g2)[0]


if __name__ == '__main__':
    n = 113604829563460357756722229849309932731534576966155520277171862442445354404910882358287832757024693652075211204635679309777620586814014894544893424988818766425089667672311645586528776360047956843961901352792631908859388801090108188344342619580661377758180391734771694803991493164412644148805229529911069578061
    e = 7

    # nbits = n.nbits()
    # kbits = nbits//(2*e*e)
    # print ("upper %d bits (of %d bits) is same" % (nbits-kbits, nbits))

    # ^^ = bit-wise XOR
    # http://doc.sagemath.org/html/en/faq/faq-usage.html#how-do-i-use-the-bitwise-xor-operator-in-sage
    # m1 = randrange(2^nbits)
    # m2 = m1 ^^ randrange(2^kbits)
    # c1 = pow(m1, e, n)
    # c2 = pow(m2, e, n)
    c1 = 164049851390841470947043007648504309649804857724005652660540753983805882970332014099145127242554403730950272988692590364500716177707553619384613221326938775905215756707180764803535659350287343632569198728798374555279481732378101195790782529098798684598482402295997081331538418016332802838476802558161233231960
    c2 = 924632688236283865268719563859347760434328330353496542527574527284055400220933495600586496916203535285696909829043530354709355431827846007716550974060075082183464174468083061976131682190685735634023159395765634524514870143813805164228292484704768874478275329131330238908862102950098119315738757212998172768030

    diff = short_pad_attack(c1, c2, e, n)
    print ("difference of two messages is %d" % diff)

    #print (m1)
    m1 = related_message_attack(c1, c2, diff, e, n)
    print (m1)
    #print (m2)
    print (m1 + diff)
```

得到

```
464c41477b325e387273613339386366386466376332363636316262376362363562326239666165323565 7d
```

输入进入最后一层

8.第6层

给出：

```
[+]Generating challenge 6


  [+]n=0xbadd260d14ea665b62e7d2e634f20a6382ac369cd44017305b69cf3a2694667ee651acded7085e0757d169b090f29f3f86fec25
5746674ffa8a6a3e1c9e1861003eb39f82cf74d84cc18e345f60865f998b33fc182a1a4ffa71f5ae48a1b5cb4c5f154b0997dc9b001e4418
15ce59c6c825f064fdca678858758dc2cebbc4d27L

[+]d=random.getrandbits(1024*0.270)

[+]e=invmod(d,phin)

[+]hex(e)=0x11722b54dd6f3ad9ce81da6f6ecb0acaf2cbc3885841d08b32abc0672d1a7293f9856db8f9407dc05f6f373a2d9246752a7c
c7b1b6923f1827adfaeefc811e6e5989cce9f00897cfc1fc57987cce4862b5343bc8e91ddf2bd9e23aea9316a69f28f407cfe324d546a7dd
e13eb0bd052f694aefe8ec0f5298800277dbab4a33bbL

[+]m=random.getrandbits(512)

[+]c=pow(m,e,n)=0xe3505f41ec936cf6bd8ae344bfec85746dc7d87a5943b3a7136482dd7b980f68f52c887585d1c7ca099310c4da2f70
d4d5345d3641428797030177da6cc0d41e7b28d0abce694157c611697df8d0add3d900c00f778ac3428f341f47ecc4d868c6c5de0724b0c3
403296d84f26736aa66f7905d498fa1862ca59e97f8f866cL

[-]long_to_bytes(m).encode('hex')=
```

e很大，d<N的0.292次方
使用Boneh and Durfee attack
直接上github上脚本就行了

```python
#!python3
# -*- coding: utf-8 -*-
# @Time : 2020/10/31 23:37
# @Author : A.James
# @FileName: tt6.py
# @Email : alexjames@sina.com
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


##########################################
# Functions
##########################################

# 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)
```

```
            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():
    ###########################################
    # How To Use This Script
    ###########################################

    #
    # The problem to solve (edit the following values)
    #

    # the modulus
    N = 0xbadd260d14ea665b62e7d2e634f20a6382ac369cd44017305b69cf3a2694667ee651acded7085e0757d169b090f29f3f86fec2
55746674ffa8a6a3e1c9e1861003eb39f82cf74d84cc18e345f60865f998b33fc182a1a4ffa71f5ae48a1b5cb4c5f154b0997dc9b001e441
815ce59c6c825f064fdca678858758dc2cebbc4d27L
    # the public exponent
    e = 0x11722b54dd6f3ad9ce81da6f6ecb0acaf2cbc3885841d08b32abc0672d1a7293f9856db8f9407dc05f6f373a2d9246752a7cc7
b1b6923f1827adfaeefc811e6e5989cce9f00897cfc1fc57987cce4862b5343bc8e91ddf2bd9e23aea9316a69f28f407cfe324d546a7dde1
3eb0bd052f694aefe8ec0f5298800277dbab4a33bbL
    # the cipher
    c = 0xe3505f41ec936cf6bd8ae344bfec85746dc7d87a5943b3a7136482dd7b980f68f52c887585d1c7ca099310c4da2f70d4d5345d
3641428797030177da6cc0d41e7b28d0abce694157c611697df8d0add3d900c00f778ac3428f341f47ecc4d868c6c5de0724b0c3403296d8
4f26736aa66f7905d498fa1862ca59e97f8f866cL
    # N = 1223860506325229217061311060769277932662809074575195569226664917788295923182258068254827980044327897948509224364580633710384108602315948278671275929116954163390193629085404406948620198903415888266127001730506434825480031875906292174474143221481891552753712400106399586592752703762527733011758841458650563595941144303946316846360823516592983134458628387511936370348028060251445171372366329706681012876990727824643474548384686948253636791281063727540594356673409962206314229342193673475035682871226838531921722580360244203396093041346917955033190754124441657364130994391338365845140921985293352610673558760588449970782 7
    # e = 1185055248150302025739280842474351085176354818493653618031770715584195978815186297644595781069156847560982100065359458471703752842982833076357155616498861963532028812598346335864888709003195790001154630084121171266447747476794140665197778417796900102595416744137791232680613223237549779823892846402546690520197718054105312969150112019701008000167726081431390684367065297201963199746735226439229689419299897154281608153480810679275800867603992976334540265757868181889177509114055597738286853120296448626112374866375249090945532486030296763614937956798941803701512680099398021640317868259975961261408500449965277690517
    # c = 9472193174575536616954091686751964873836697237500198884451530469300324470671555310791335185133679697207007374620225900775502162690848135615431624557389304657410880981454777737587420426091879654002644281066474715074536611611252677882396384453641127487515845176069574754606670518031472235144795376526854484442135299818868525539923568705203042265537204111153151119105287648912908771710419648445826883069030285651763726003413418764301988228077415599665616637501056116290476861280240577145515875430665394216054222788697052979429015400411487342877096677666406389711074591330476335174211990429870900468249946600544116793793

    # the hypothesis on the private exponent (the theoretical maximum is 0.292)
    delta = .18  # this means that d < N^delta

    #
    # Lattice (tweak those values)
    #

    # you should tweak this (after a first run), (e.g. increment it until a solution is found)
    m = 4  # size of the lattice (bigger the better/slower)

    # you need to be a lattice master to tweak these
```

```
    # 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

    #
    # Don't touch anything below
    #

    # 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)
        m = pow(c, d, N)
        print '[-]d is ' + str(d)
        print '[-]m is: ' + str(m)
        print '[-]hex(m) is: ' + '{:x}'.format(int(m))
        print '[-]str(m) is: ' + '{:x}'.format(int(m)).decode('hex')
    else:
        print "[!]no solution was found!"
        print '[!]All Done!'

    if debug:
        print("[!]Timer: %s s" % (time.time() - start_time))
        print '[!]All Done!'


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

得到：

```
looking for independent vectors in the lattice
found them, using vectors 0 and 1
=== solution found ===
[-]d is 776765455081795377117377680209510234887230129318575063382634593357724998207571
[-]m is: 561625664447464377732492715642529630820143635640479763522621585360875210937572855917766325763474674836799964854461239512729228476161083355216318822502685 6
[-]hex(m) is: 6b3bb0cdc72a7f2ce89902e19db0fb2c0514c76874b2ca4113b86e6dc128d44cc859283db4ca8b0b5d9ee35032aec8cc8bb96e8c11547915fc9ef05aa2d72b28
[-]str(m) is: k;▒▒▒*,▒ ▒,▒ht▒▒A▒nm▒(▒L▒Y(=▒υ
                                        ]▒▒P2▒▒ή▒Ty▒▒▒Z▒▒+(
[!]Timer: 0.682039022446 s
[!]All Done!
```

输入，得到flag

9.get flag

```
flag{a214781d-bb31-49dd-a66b-2139ce6b2e76}
```

结语

已经乱套了。。。勉强算今天完成指标？

参考： BUUCTF 强网杯2019 Copperstudy.