Coppersmith攻击

一、引言

近期参加的各类CTF比赛中，常见到coppersmith攻击的考点，于是在此做一个简单的学习与总结，后续若还有新的coppersmith的考点，会在文章中作出补充。

二、Coppersmith攻击原理

真正能够用好一个方法，一定要搞清楚这个方法背后的本质原理，于是我在网上搜集资料，找到了Coppersmith攻击的原理。

对于$f(x) = a_1x^m + a_2x^{m-1} + \cdots + a_m + 1$,要找到这个多项式在$mod(N)$意义下的根，在不限制的条件下这是一个很困难的问题。

因为$f(x)$可以特别大只需要是N的倍数就可以

$mod(N)$这一步带来很多困难.

首先考虑一个特殊情况如果$f(x)$就等于0那么可以用牛顿迭代法求出来但是牛顿迭代法遇到$mod(N)$就失灵了.

怎么办呢？没有条件就创造条件！

大概的思路是如果x足够小使得$|f(x)| < N$那么$f(x) == 0$

这时候就满足了使用牛顿迭代法的条件

那么现在问题集中在怎么证明$|f(x)| < N$.答案是：无法证明！因为没有限制a_i的大小

Coppersmith想出了一个办法：能不能找一个系数的多项式$g(x)$当$x = 0$时$f(x) = 0$

也就是说希望找到一个系数的多项式系数小到能够满足牛顿迭代法的要求那么问题就解决了，就差把x求出来。

Coppersmith利用如下操作先得到一系列$h(x)$：$h_{ij}(x) = x^i * f^j(x)$

再将一系列的$h(x)$经过线性组合得到$g(x)$：$g(x) = \sum c_{ij} * h_{ij}(x) + d_{ij}$

为了让$g(x)$的系数尽量小就要用到LLL格基规约算法（关于LLL格基规约算法将在下一个博客中给出学习介绍）

三、Coppersmith攻击在CTF比赛中的运用

在CTF比赛中，使用Coppersmith攻击的主要原理是根据以下公式完成的：

$c = (m + t)^e (mod N)$

这里只需要将t看成上述的x，即可把这个公式转换为求解一个有关t的多项式的根，其中，t要求不能太大，否则无法使用Coppersmith攻击原理解决

具体来说，我们一般会在sagemath中利用$small roots$函数来解决

`small_roots()` 是 `SageMath` 中一个运用了 `coppersmith` 定理的函数。它只能解有限域内的方程。

small_roots(X = ,beta =) 有两个参数

X代表所需要求解根的上限；虽然是根的上限，并不是说上限越高越好，当上限超过某个值的时候就会计算失效，即使已知二进制位数满足条件，也无法用此函数求得结果；所以一般来说X取在给定情况下的最大求解上限

那么下面，我们给出几个题目来理解一下Coppersmith攻击的运用吧

1、极客：highlow

先看题目：

```
from Crypto.Util.number import *
e = 65537
p = getPrime(1024)
q = getPrime(1024)
n = p * q
pxor = p ^ (bytes_to_long(flag)<<400)
assert len(flag)== 44
m = bytes_to_long(flag)
c = pow(m, e, n)

print('c =',c)
print('n =',n)
print('pxor =',pxor)


'''
c =
110173361226910340532419922939631145908163198443842874486296636720492058928286003
964655057109229076855459399783763219273946554587274943618529528982809052209631636
254822952228561291641726195643446343655203288159722328256392926053117416559884271
668114060913296136279610702314570353032987936515464124969756622258571238058677566
519013745074478031986384663048624802020990768134715714953801325632526307892181730
072758906007467582854152744343933811257425260149860396526776056422265767414240537
495122808252312174202390891057940807073223576029410468226593354874206726990229693
720376629584978320657522720618537236533651717685 56
n =
140912063206225236748477201397615431548221908790353802454244812674825509322296119
659644249659583862550765939110628042992755817426651342073905328021097002251409998
126980208386836973758910356252552220018844772143618351014422887253830733343929951
860538672614976792343627949141080335746812926565228079286808127264621950778331840
181223695790027159004772903453960659125365292908119621178149004483197765907129462
595403824616324686348279599572869058064320056328646639850148723656726534768228339
218700718513134249034812823503423048191498946100898043214055894339806503406105216
590312348268233691148001508839886138778778810695 79
pxor =
124229245244085791439650934438639686782423445183921252684721764061493908790073948
877623812930339081158169421854801552819088679937157357924845248082716160727839419
054107753000815066526032809275137495740454967765165248115412626716101315676902716
808647904092798908601183908297141420793614426863816161203796966951
'''
```

$44 * 8 + 400 = 752$
$1024 - 752 = 272$所以p高272已知$phigh = (pxor >> 752) << 752$
$plow = pxor$
已知672位，1024-672=352 未知

$352 \div 1024 \leq 0.44$

可用small roots()

```
from Crypto.Util.number import *
e = 65537
c =
110173361226910340532419922939631145908163198443842874486296636720492058928286003
964655057109229076855459399783763219273946554587274943618529528982809052209631636
254822952228561291641726195643446343655203288159722328256392926053117416559884271
668114060913296136279610702314570353032987936515464124969756622258571238058677566
519013745074478031986384663048624802020990768134715714953801325632526307892181730
072758906007467582854152744343933811257425260149860396526776056422265767414240537
495122808252312174202390891057940807073223576029410468226593354874206726990229693
720376629584978320657522720618537236533651717685 56
n =
140912063206225236748477201397615431548221908790353802454244812674825509322296119
659644249659583862550765939110628042992755817426651342073905328021097002251409998
126980208386836973758910356252552220018844772143618351014422887253830733343929951
860538672614976792343627949141080335746812926565228079286808127264621950778331840
181223695790027159004772903453960659125365292908119621178149004483197765907129462
595403824616324686348279599572869058064320056328646639850148723656726534768228339
218700718513134249034812823503423048191498946100898043214055894339806503406105216
590312348268233691148001508839886138778778810695 79
pxor =
124229245244085791439650934438639686782423445183921252684721764061493908790073948
877623812930339081158169421854801552819088679937157357924845248082716160727839419
054107753000815066526032809275137495740454967765165248115412626716101315676902716
808647904092798908601183908297141420793614426863816161203796966951
phigh=(pxor>>752)<<752
plow = pxor%2**400
print(phigh)
print(plow)
R.<x>= PolynomialRing(zmod(n))
f= phigh + x*2**400 + plow
f= f.monic()
root =f.small_roots(x=2^352,beta=0.4)
print(root)
if root:
p=int(phigh + root[0]*2**400 + plow)
q = n//p
d=inverse(e,(p-1)*(q-1))
m= pow(c,d,n)
print(long_to_bytes(int(m)))
else:
print(0)
```

2、i春秋冬季赛 RSA1

这是一道我比较喜欢的题目🤭考察Coppersmith以及简单的数论知识。

题目代码如下：

```
from Crypto.Util.number import *
import uuid
p, q = [getPrime(512) for _ in range(2)]
```

```
N = p * q
flag = b'flag{' + str(uuid.uuid4()).encode() + b'}'
flag += bin(getPrime((1024 - bytes_to_long(flag).bit_length()) / 8)).encode()
m1 = bytes_to_long(flag)
m2 = bytes_to_long(''.join(chr((ord(i) + 3) % 128) for i in
flag.decode()).encode())
e = getPrime(128)
c1 = pow(m1 * e, 2835, N)
c2 = pow(m2, 2025, N)
c3 = pow(m2, 2835, N) + e
print(f'{N = }')
print(f'{c1 = }')
print(f'{c2 = }')
print(f'{c3 = }')
'''
N =
176871561120476589165761750300633332586877708342448994506175624203633860119621512
31832
117292787638963191830018422108231774138036544719777702625640531221271663061772160
69180
660489956838996160593881736294376730183865900430531467128705723007994792699471182
51011
967950970286626852935438101046112260915112568392601
c1 =
472803750068170825211148855781321044276873844579639202637786615425522598608900753
21953
563867658233347930121507835612417278438979006705016537596357679038471176957659834
15569
428436468275967584180820981231609496539355050991398488884994542109246384254663122
86402
937947450053387735743436761001210007640212070440l9
c2 =
176231410933979134585886078013933649498379873444851943224935010972452769899603364
68615
827926919789119064372500815181215042880855031058770900868333943659011280275676714
01021
363043460015994016702919383690144361706938640340991387671670554566357601968885786
42643
971920733784690410395944410255241615897032471127315
c3 =
135594807884016971356816423169128168727346102408490289623885211179619571354105102
39365
824929233317934649741512978518465400829972561766865564085731806399270326540716208
51788
857331345905245779960933668193289604625001242014028162441044770182796731833680743
74836
717994805448310223434099196774685324616523478136309
'''
```

根据上述题目代码，我们有：

$c_2 \equiv m_2^{2025} mod(N)$

$c_3 - e \equiv m_2^{2835} mod(N)$

观察2025与2835这两个数字，突然顿悟，发现他们的最大公约数为14175！

于是进行线性变化：

$$(c_3 - e)^5 - c_2^7 = 0 mod(N)$$

至此，我们便可以使用Coppersmith攻击原理，求出e了！

接下来就是其他知识点的考察了，不过我们能发现，运用Coppersmith我们对这道题的进度推进了很多！

我们注意到：

$$m_1 = \sum 256^i * flag_i$$

$$m2 = \sum 256^i * (flag_i + 3)$$

(这里利用了bytes_to_long函数的原理，将每个字符转换为八位二进制的数)

我们发现$m_2 - m_1$是一个定值

于是我们可以对以下两个式子做相关消息攻击：

$$c_1 \equiv (e * m_1)^{2835} mod(N)$$

$$c_2 \equiv (m_1 + b)^{2025} mod(N)$$

而m是大于n的，所以还需要做小范围爆破

```
from Crypto.Util.number import *
import sys
def HGCD(a, b):
if 2 * b.degree() < a.degree() or a.degree() = 1:
return 1, 0, 0, 1
m = a.degree() / 2
a_top, a_bot = a.quo_rem(x^m)
b_top, b_bot = b.quo_rem(x^m)
R00, R01, R10, R11 = HGCD(a_top, b_top)
c = R00 * a + R01 * b
d = R10 * a + R11 * b
q, e = c.quo_rem(d)
d_top, d_bot = d.quo_rem(x^(m / 2))
e_top, e_bot = e.quo_rem(x^(m / 2))
S00, S01, S10, S11 = HGCD(d_top, e_top)
RET00 = S01 * R00 + (S00 - q * S01) * R10
RET01 = S01 * R01 + (S00 - q * S01) * R11
RET10 = S11 * R00 + (S10 - q * S11) * R10
RET11 = S11 * R01 + (S10 - q * S11) * R11
return RET00, RET01, RET10, RET11
def GCD(a, b):
print(a.degree(), b.degree())
q, r = a.quo_rem(b)
if r = 0:
return b
R00, R01, R10, R11 = HGCD(a, b)
c = R00 * a + R01 * b
d = R10 * a + R11 * b
if d = 0:
return c.monic()
q, r = c.quo_rem(d)
if r = 0:
```

```
return d
return GCD(d, r)
sys.setrecursionlimit(500000)
N =
176871561120476589165761750300633332586877708342448994506175624203633860119621512
31832
117292787638963191830018422108231774138036544719777702625640531221271663061772160
69180
660489956838996160593881736294376730183865900430531467128705723007994792699471182
51011
967950970286626852935438101046112260915112568392601
c1 =
472803750068170825211148855781321044276873844579639202637786615425522598608900753
21953
563867658233347930121507835612417278438979006705016537596357679038471176957659834
15569
428436468275967584180820981231609496539355050991398488884994542109246384254663122
86402
937947450053387735743436761001210007640212070440l9
c2 =
176231410933979134585886078013933649498379873444851943224935010972452769899603364
68615
827926919789119064372500815181215042880855031058770900868333943659011280275676714
01021
363043460015994016702919383690144361706938640340991387671670554566357601968885786
42643
971920733784690410395944410255241615897032471127315
c3 =
135594807884016971356816423169128168727346102408490289623885211179619571354105102
39365
824929233317934649741512978518465400829972561766865564085731806399270326540716208
51788
857331345905245779960933668193289604625001242014028162441044770182796731833680743
74836
717994805448310223434099196774685324616523478136309
R.<x> = PolynomialRing(Zmod(N))
f = (c3 - x)^5 - c2^7
res = f.monic().small_roots(X=2^128,beta=0.4,epsilon=0.02)
e = int(res[0])
print(f"e = {e}")
b =
138604255630984394504644405862999441108691457990544710059664868220625513430462483
76311
979729177999252936082401988695875971773687666145304433574557360333076181743282892
46889
930263321025496073979013516194253249935830875007140615239459258573684989221027684
58574
857510324727265052999967460998294909713988129273348867
R.<x> = PolynomialRing(Zmod(N))
f = (e * x)^2835 - c1
g = (x + b)^2025 - c2
res = GCD(f,g)
from tqdm import *
m = int(-res.monic().coefficients()[0])
print(f"m mod N = {m}")
for k in trange(2^16):
```

```
mm = m + k*N
flag = long_to_bytes(int(mm))
if b"flag" in flag:
print(flag)
break
# flag{2404dcef-4223-417d-aee0-c236241f2320}
```

3、西湖论剑 matrix RSA

先上题目：

```
import random
import string
from Crypto.Util.number import *
from secret import flag
ext_len = 9*23 - len(flag)
flag += ''.join(random.choice(string.printable) for _ in range(ext_len))
def my_rsa_encrypt():
    p = getPrime(512)
    q = getPrime(512)
    n = p * q
    data = []
    for i in range(9):
        data.append(bytes_to_long(flag[23*i:23*(i+1)].encode()))
    M = Matrix(Zmod(n), [data[i:i+3] for i in range(0, len(data), 3)])
    e = 65537
    C = M**e
    print("p =", p >> 100)
    print("n =", n)
    return C
C = my_rsa_encrypt()
print("C =", C)
'''
high_p =
12305755811288164655681709252717258015229295989302934566212712319314835335461946241491177972870130171728224502716603340551353785064416812665744581355110400
p =
97075296687215080948787543836368130587614075289501890137893157324470486317408493158942535764158436311073700029129493797572 75
n =
13229877767208554709651108726625506628550213502012409390045213826299315538176681642495584979616805920437932507556809443125987792335366492687598622302047258564591941482132288021329918815742762280414099689868556407548475491833967009980618687397459413918232488462001878094363019675473697280503603879894672641400 9
```

```
C =
[13070095298901431143443402809881041208929472827015670561832673332229746571449570
40721595306186553400967053837103046580449911496620606577459330904730827754258126 4
13009644725436054603606406759494478372084497948305781849685285473666081800857873 8
23765366221360353648153310374930982834625408498806745411384432719 41
71108771421281691064141020659106224750236412635914570166893031318860027728093402 4
53305986361330527563506168063047627979831630830003190075818824767924892107148560 0
48725155587353683119195901991465464478196049173060097561821877061015587704803006 4
99153902855903286456023726638247758665778434728734461065079337757
67999998657112350704927993584783146575182096185020115836188544590466205688442741 0
39622382576899587857972463337900200038021257164640987281308471100297698062626107 3
80871262596623736773815445544153508352926374272336154553916204320257697068627063 2
36060520725376727528604938949588845448940836430120015498687885615]
[
23893343854815011808020457237095285782125931083991537368666368653089096539223297 5
67339111502968295914745423286070638369517207554770793304994639155083818859208362 0
57394004419565231389473766857235749279110546079776040193183912062870294579472815 5
88333047561915280189529367474392709554971446978468118280633281993
97113238292698297515191777559151644026586936686318684993839452036271971715084413 3
22119072784732767130662752839738565135802058085179180960176991229544643055567953 0
08740056270014642977604138970740440806659418025886809264300307152997132414423133 0
09204631459033990541239679149688943454919589809459277644541596 01
44904507975955275578858125671789564568591470104141872573541481508697254621798834 9
10263012676346204850278744732796211742615531019931085695420000582627144871996018 8
50098958417750918177991375489106531511894991744745328626887250694950153424439172 6
67977623425955725695498585224383607063387876414273539268016177401]
[
67805732998935098446255672500407441801838056284635701147853683333480924477835278 0
30145327818330916280792499177503535618310624546400536573924729837478349680007368 7
81306805363621196573313903080315513952415535369016620873765493531188596985587834 4
08434835281527678166509365418905214174034794683785063802543354572
13486048723056269216825615499052563411132892702727634833280269923882908676944418 6
24902325737619945647093190397919828623788245644333036340084254490542292357044974 1
39884304715033710988658109160936809398722070125690919829906642273377982021120160 7
02344103998315875166038849942426382506293976662337161520494820727
95932690738697024519546289135992512776877884741458439242887603021792409575448192 5
08456813215486904392440772808083658410285088451086298418303987628634150431725794 9
04656250453314950126433260613949819432633322599879072805834951478466009343397728 7
11205498602927752917834774516505262381463414617797291857077444676]

'''
```

我们能发现这里也有Coppersmith的踪迹!

于是先把p，q求出:

```
from sage.all import *
n =
13229877767208554709651108726625506628550213502012409390045213826299315538176681 6
42495584979616805920437932507556809443125987792335366492687598622302047258564591 9
41482132288021329918815742762280414099689868556407548475491833967009980618687397 4
59413918232488462001878094363019675473697280503603879894672641400 9
p4 =
97075296687215080948787543836368130587614075289501890137893157324470486317408493 1
58942535764158436311073700029129493797572 75
```

```
e =  65537
pbits = 512
kbits = pbits - p4.nbits()
print(p4.nbits())
p4 = p4 << kbits
PR.<x> = PolynomialRing(Zmod(n))
f = x + p4
roots = f.small_roots(X=2^kbits, beta=0.4)
if roots:
    p = p4+int(roots[0])
    print("n= "+str(n))
    print("p= "+str(p))
    print("q= "+str(n//p))
```

接下来就是一个非线性费马定理的使用了，这是以前从来没有接触过的东西捏(￣▽￣)*，也感谢学长的帮助，给到了文献参考！

[]: https://www.gcsu.edu/sites/files/page-assets/node-808/attachments/pangia.pdf

```
import random
import string
from Crypto.Util.number import *
n=
13229877767208554709651108726625506628550213502012409390045213826299315538176681642495584979616805920437932507556809443125987792335366492687598622302047258564591941482132288021329918815742762280414099689868556407548475491833967009980618687397459413918232488462001878094363019675473697280503603879894672641400 9
p=
12305755811288164655681709252717258015229295989302934566212712319314835335461946241491177972870130171728224502716603340551354171940107285908105124549960063
q=
10750967246621849802090386055921679114516122704252330881722100331526757637044067492444912824266860574267360247681890637480406758188129451052986858429875143
e=65537
```

```
C =
[(13070095298901431143443402809881041208929472827015670561832673332229746571449570407215953061865534009670538371030465804499114966206065774593309047308277542581264130096447254360546036064067594944783720844979483057818496852854736660818008578738237653662213603536481533103749309828346254084988067454113844327194 1,711087714212816910641410206591062247502364126359145701668930313188600277280934024533059863613305275635061680630476279798316308300031900758188247679248921071485600487251555873536831191959019914654644781960491730600975618218770610155877048030064991539028559032864560237266382477586657784347287344610650793377571,679999986571123507049279935847831465751820961850201158361885445904662056884427410396223825768995878579724633379002000380212571646409872813084711002976980626261073808712625966237367738154455441535083529263742723361545539162043202576970686270632360605207253767275286049389495888454489408364301200154986878856151),(23893343854815011808020457237095285782125931083991537368666368653089096539223297567339111502968295914745423286070638369517207554770793304994639155083818859208362057394004419565231389473766857235749279110546079776040193183912062870294579472815588333047561915280189529367474392709554971446978468118280633281993,971132382926982975151917775591516440265869366863186849938394520362719717150844133221190727847327671306627528397385651358020580851791809601769912295446430555679530087400562700146429776041389707404408066594180258868092643003071529971324144231330092046314590339905412396791496889434549195898094592776445415960 1,449045079759552755788581256717895645685914701041418725735414815086972546217988349102630126763462048502787447327962117426155310199310856954200005826271448719960188500989584177509181779913754891065315118949917447453286268872506949501534244391726679776234259557256954985852243836070633878764142735392680161774 01),(67805732998935098446255672500407441801838056284635701147853683333480924477835278030145327818330916280792499177503535618310624546400536573924729837478349680007368781306805363621196573313903080315513952415535369016620873765493531188596985587834408434835281527678166509365418905214174034794683785063802543354572,134860487230562692168256154990525634111328927027276348332802699238829086769444186249023257376199456470931903979198286237882456443330363400842544905422923570449741398843047150337109886581091609368093987220701256909198299066422733779820211201607023441039983158751660388499424263825062939766623371615204948207275932690738697024519546289135992512776877884741458439242887603021792409575448192508456813215486904392440772808083658410285088451086298418303987628634150431725794904656250453314950126433260613949819432633322599879072805834951478466009343397728711205498602927752917834774516505262381463414617797291857077444676)]
C = matrix(Zmod(n), C)
phi=(p**2-1)*(q**2-1)
d = inverse(e,phi)
M = C**d
flag = b""
for i in range(3):
    for j in range(3):
        m = int(M[i,j])
        flag += long_to_bytes(m)
print(flag)
```

四、总结

总的来说，Coppersmith攻击的考察是十分常见的，这一般用于解决 RSA 攻击中“已知某些二进制位，求剩余位”这一类问题

当然，Coppersmith攻击很少会直接考察，一般会与其他知识点结合起来形成综合题

因此，不断深入学习其他攻击方式也是十分重要的！