

Week3 Crypto official wp

[Week3] hard_rsa

首先flag是gpt生成的



ChatGPT

我明白了，您希望生成的flag字符串符合以下条件：

1. 总长度为255个字符。
2. 格式为"SBCTF{}"，其中大括号内的内容是有意义的，只包含大小写字母和数字。
3. "SBCTF{}"自身占用7个字符，所以大括号内的内容需要有248个字符。

由于生成一个有语义的、且长度恰好为248个字符的字符串比较困难，我建议使用一句有意义的话或者短语重复拼接，直到总长度达到所需的248个字符。我将为您生成这样一个flag。

现在我为您生成了一个符合要求的flag字符串。这个字符串由有意义的句子"TheQuickBrownFox123JumpsOverTheLazyDog987"重复拼接而成，总长度恰好为248个字符。完整的flag如下：

Copy code

```
SBCTF{TheQuickBrownFox123JumpsOverTheLazyDog987TheQuickBrownFox123Jumps
```

这个flag只包含大小写字母和数字，符合您的要求。如果需要进一步的调整或不同的内容，请告诉我。 [-]

看了一下选手（*Libr）写的wp，比我写的详细，放他的上来

网上查到的大部分e=3的题都是暴力开方但是这个看上去不行毕竟给了三个参数

于是可以开始推式子了（

$$ed \equiv 1 \pmod{\phi(n)} \quad (1)$$

$$ed = k_1 \cdot \phi(n) + 1 \quad (2)$$

因为e=3,d<n，有ed<3n,k_1\in\{1,2\}

又有

$$\begin{aligned} g_1 &\equiv d^3 \pmod{n} & (3) \\ e^3 g_1 &\equiv (ed)^3 \pmod{n} & (4) \\ &\equiv (1 + k_1 \cdot \phi(n))^3 \pmod{n} & (5) \\ &\equiv 1 + 3k_1 \cdot \phi(n) + 3(k_1 \cdot \phi(n))^2 + (k_1 \cdot \phi(n))^3 \pmod{n} & (6) \end{aligned}$$

可知

$$(k_1)^3 g_2 + 3(k_1 \cdot \phi(n))^2 + 3(k_1 \cdot \phi(n)) + 1 - 27g_1 \equiv 0 \pmod{n}$$

欧拉函数的定义如下

$$\phi(n) = (p-1)(q-1) \quad (7)$$

$$= n - p - q + 1 \quad (8)$$

$$\equiv -(p+q) + 1 \pmod{n} \quad (9)$$

设r=p+q，带入上面的式子有

$$k_1^3 g_2 + 3k_1^2(1-r)^2 + 3k_1(1-r) + 1 - 27g_1 \equiv 0 \pmod{n} \quad (10)$$

$$k_1^3 g_2 + 3k_1^2 - 6k_1^2 r + 3k_1^2 r^2 + 3k_1 - 3k_1 r + 1 - 27g_1 \equiv 0 \pmod{n} \quad (11)$$

$$3k_1^2 r^2 - (6k_1^2 + 3k_1)r + (3k_1^2 + 3k_1 + k_1^3 g_2 + 1 - 27g_1) \equiv 0 \pmod{n} \quad (12)$$

对r，假设p>q，有

$$r^2 = (p+q)^2 \quad (13)$$

$$= p^2 + q^2 + 2pq \quad (14)$$

$$= p^2 + q^2 + 2n \quad (15)$$

$$\leq 5q^2 + 2n \quad (16)$$

$$\leq 7n \quad (17)$$

$$3k_1^2 r^2 \leq 12r^2 \quad (18)$$

$$\leq 84n \quad (19)$$

所以

$$3k_1^2 r^2 - (6k_1^2 + 3k_1)r + (3k_1^2 + 3k_1 + k_1^3 g_2 + 1 - 27g_1) < 84n \quad (20)$$

$$3k_1^2 r^2 - (6k_1^2 + 3k_1)r + (3k_1^2 + 3k_1 + k_1^3 g_2 + 1 - 27g_1) = k_2 n, k_2 < 84. \quad (21)$$

于是可以解出(r=p+q)，得出\phi(n)然后得出结果。

```
1 from Crypto.Util.number import long_to_bytes
2 n =
3 g1 =
```

```

4 g2 =
5 e = 3
6 for k in range(1, 3):
7     a = 3 * (k ^ 2)
8     b = -(6 * (k ^ 2) + 3 * k)
9     c = 3 * (k ^ 2) + 3 * k + (k ^ 3) * g2 - 27 * int(g1) + 1
10    det = b ^ 2 - 4 * a * c
11    for i in range(85):
12        c -= n
13        det = b ^ 2 - 4 * a * c
14        if det.is_square():
15            break
16    if det.is_square():
17        break
18 r = (-b + sqrt(b ^ 2 - 4 * a * c)) / (2 * a)
19 c =
20 phi = n - r + 1
21 d = pow(e, -1, phi)
22 assert pow(d, e, n) == g1
23 flag = pow(c, d, n)
24 print(long_to_bytes(flag))

```



App1e_Tree Feb 5

rsa那个的official wp抄你写的了



Astrageldon Feb 5

再次力荐SageMath内置的groebner_basis方法，让CTFer免受推柿子的苦恼



2



下面是我的exp

```

1 n =
1358215830479847592082136878159952147738127439491395985928232738440202833645515
5070191033091618001600779177213825182285336369521886009921268868513838711681881
1848003161829901100401906148511351889985854301200170827173131031943705808707388
6751231902569865495950861415659958828548463109058901253580021080363239267463565
9098864123371037174003836572602880877457857851274128161485297956518193774520508
2950937411014332947232970696524373739177041161753044945304908428190772820296748
4923441340258693579573883276845845024197652237795112043451002641953423688009420
6402934202760751447567171103761352928211450854185202396847054559
2 enc_flag =
3661548680203135717465118261606647815957963207743832438859391518331275928866092
8692083525564607202556632038907392388265359384638663852681340025732092669090705

```

```

7646296468158836470474036042147583948224544525575561104171696155795927900529640
6600623666312838550335290104524929338311598737092108500249468399384382625585362
9971193180754546493611632029410172575993828186037490173770493753243788617758075
3350734116935563920873362413346831361893190759884997180416624037329528727901820
0068962599404220423990138659626907797102310498532054432947535807516147433032600
558441821788124540260259662740089919934726120896864375394273657

3 enc_d =
2067448728669985634328202288426291556151139221426076544069690674040317518936698
9690287473768795209393317354134199368291381902866314128925745236629209103984978
4602590835742595231788662373821411609235767951302765069849639505199534357927484
4734699962227911426224691005713554990380111589204101300567264593509810146795995
3808048853412777049895417806958597753797163837386208985915022648605664788281092
1142431217282316045257828187414380777016449697758500490017956193511208971611735
5472151547260468460998782080733989529112163300560960983303468842358833434502034
679489965127244633545517106253617496279747948496168385505631973

4 enc_phi =
1356079881748078551855313304750791404295193993619629128304960420206848217252863
6237561886831940581702111112201978264161855517598353198771969285438223194979743
6878553202887428146270620262157703894078173257223007671519219524415007590044546
6314137609236889706276349702436810630025113055119839078967139922384983168150971
1092356093344645404416488083219929262363766915540748799187301522983936925775386
5115001834657605059163719155007394469516904948577601260010519360268881434761553
5189440993789830994051495113244559822781670608244280484457590100842168442019449
5339554708022893443679915498176969300535739611886005965899015126

5 e = 3
6 for k in range(3):
7     for m in range(200):
8         PR = PolynomialRing(ZZ, names=["p_q"])
9         p_q = PR.gens()[0]
10        f = (k^3*enc_phi+3*k^2*(1-p_q)^2+3*k*(1-p_q)+1)-enc_d*e^3-m*n
11        try:
12            roots = f.roots()
13            if len(roots) > 0:
14                #print(k, m, roots)
15                PR = PolynomialRing(ZZ, names=["z"])
16                z = PR.gens()[0]
17                g = z ** 2 - int(roots[0][0]) * z + n
18                p, q = g.roots(multiplicities=False)
19                phi = (p - 1) * (q - 1)
20                d = int(pow(e, -1, phi))
21                print(int(pow(enc_flag, d, n)))
22        except:
23            continue

```

都叫baby了，这个题肯定是本次比赛rsa最难的一个了，同时也结束了本人在本次比赛的命题任务

(hard_pic_encode / SuperBag / baby_pic_encode / broken_pem / a bit limit / ez_rsa / strange_rsa / hard_DSA / hard_rsa / baby_rsa)，希望选手们有所收获

回到本题，其实打法应该不少，也完全没有卡界，根据关键内容简单搜索大概找到的东西都能打，比如

A Polynomial Time Attack on RSA with Private CRT-Exponents Smaller Than $N^{0.073}$

Ellen Jochemsz^{1**} and Alexander May²

¹ Department of Mathematics and Computer Science,
TU Eindhoven, 5600 MB Eindhoven, the Netherlands
`e.jochemsz@tue.nl`

² Faculty of Computer Science
TU Darmstadt, 64289 Darmstadt, Germany
`may@informatik.tu-darmstadt.de`

比如这篇里提到的

3 The Bleichenbacher-May Attack

In [2], Bleichenbacher and May describe two new attacks on RSA-CRT. One of them is meant for the case that both e and d_p and d_q are chosen to be smaller than in standard RSA-CRT. For notation, we use $e = N^\alpha$, $d_p < N^\delta$, and $d_q < N^\delta$ for some $\alpha \in [0, 1]$ and $\delta \in [0, \frac{1}{2}]$. Clearly, if an attack on this so called 'balanced' RSA works in the case $\alpha = 1$, then it threatens the security of standard RSA with small private CRT-exponents.

The attack of Bleichenbacher and May uses a lattice of dimension 3. The attack works whenever $\delta < \min\{\frac{1}{4}, \frac{2}{5} - \frac{2}{5}\alpha\}$, and therefore gives no result in the case $\alpha = 1$. However, we present a generalization of the attack for higher dimensional lattices that is applicable also for $\alpha = 1$. To explain our new attack, we first describe the basics of the BM-attack [2].

Bleichenbacher and May start with the two RSA-CRT equations $ed_p = 1 + k(p - 1)$ and $ed_q = 1 + l(q - 1)$, and rewrite these as

$$ed_p + k - 1 = kp \quad \text{and} \quad ed_q + l - 1 = lq.$$

Multiplying the two equations yields

$$e^2 d_p d_q + ed_p(l - 1) + ed_q(k - 1) - (N - 1)kl - (k + l - 1) = 0.$$

This can be transformed into the linear equation $e^2 x_1 + ex_2 - (N - 1)x_3 - x_4 = 0$, if we substitute $x_1 = d_p d_q$, $x_2 = d_p(l - 1) + d_q(k - 1)$, $x_3 = kl$, $x_4 = k + l - 1$.

The given linear equation leads directly to a lattice attack with a lattice of dimension 3. This attack works provided that $\delta < \min\{\frac{1}{4}, \frac{2}{5} - \frac{2}{5}\alpha\}$.

Although linearization of an equation makes the analysis easier and keeps the lattice dimension small, better results can sometimes be obtained by using a non-linear polynomial equation directly. In the next section, we will pursue this approach and use a polynomial with the variables x_1, \dots, x_4 corresponding to d_p , d_q , k , and l , respectively.

1 N,e,dp,dq,p,q,mod1,mod2=

(126177665667219526880362739245161907469717721994093849266781209812000134417755
3477871400910944442135904094776207555157321478988877066626627159863899964028937
6995592728380318044033759067166363712690981669814300866961844495774613030244233
1329146719751783765058131698796752643481154051277835987325341565423571998584478
9382242516134038450285846152465490469628232195624592657237450600980594383982290
4147942971729648587230595620457603370830927797374626294735767794931399618822598
9626622321276760868646565763877756803900174475577610440712159126476297825616272
29024260466953957487569475759224643022546394315271,
4276463639834167164549025720413739189651696351302645824086360701005123978504031
9358530821698440707266217223611829155438905911544029619335177001645867664832414
7220904317832032463055323339374307060498884717319694172761032282067389473298184
9176651369295876281620201986820360699053809934080127501504480181,
735410450085599741, 927684899266340059,
3297216511812791043165190883448275215825143893478805130676551758402853719379091
7213691777134934834585209448585560154724888394249124178891477648551348602083242

```

9510297764005681215213209353875262136332423042520928066611705772480985766743933
9492787890318657232191646643636157400773217234099417386062173777,
3826793454878331841480939908284159983672992325011083249496794741538564030064499
9440696574468355877435188787509298181357767282641695909241013355224836306779779
0313332151505939843047620037908354852363820786586064890661461517125324926322200
9906570535708578599468622546400525307616417360650123287221419223,
953821515474598870, 1036693196984120449)
2 M = Matrix(ZZ, [[ e^2, 1, 0, 0],
3                 [   e, 0, 1, 0],
4                 [-(N-1), 0, 0, 1],
5                 ])
6
7 weights = [2^60, 1, 1, 1]
8 MM = M * diagonal_matrix(weights)
9
10 v = MM.LLL()[0]
11 print(v[0] / weights[0])
12 print(v[1])
13 print(v[2])
14 print(v[3])

```

对应数据求完剩下的就不写了=.=

[week3] ezLCG

很经典的一个格子了，hnp问题，LLL打一下就好

```

1 a = 700422662418845693420459228419976525190677154239520053190899519534362891
2 b = 676480746474236007375202937993822423100599311126708017580109897281421063
3 m = 844119564578779233006299744321394282273047303209637385219366850426130687
4 h = [0,12348955370613140461688478695491701240600980908866901,
      28017806374538330606757310617318498653460398186490944,
      10673614611905813029405446449393243291024607022481969,
      260609204914994428874455738524755200218180374665478,
      28676953586474855586464705518607638671525932889080989,
      32181413272741919304315353206081126098176979133402871,
      17075587087877158375108221588026648473003610109807021,
      23433678215530643676572947814576447334594716158303548,
      36568487875936468852654147678033030300536559749164043,
      21929172319902779563837790346553743989109542463810048]
5
6 for i in range(len(h)):
7     h[i] <= 64
8 A = [1]
9 B = [0]

```

```

10 for i in range(1, len(h)-1):
11     A.append(a*A[i-1] % m)
12     B.append((a*B[i-1]+a*h[i]+b-h[i+1]) % m)
13 A = A[1:]
14 B = B[1:]
15
16 M = matrix(ZZ, 11, 11)
17
18 for i in range(9):
19     M[i, i] = m
20     M[9, i] = A[i]
21     M[10, i] = B[i]
22     M[i, 9] = M[i, 10] = 0
23 M[9, 9] = 1
24 M[10, 10] = 2^64
25 M[9, 10] = 0
26
27 #print(B)
28 vl = M.LLL()[0]
29 l1 = vl[-2]
30 h1 = h[1]
31 s1 = l1+h1
32 #s1 = a*seed+b %m
33 seed = ((s1 - b)*inverse_mod(a,m))%m
34 print(seed)

```

[week3] ezDH

拆解一下过程需要解个DLP就ok了，可以参考去年hgame的wp（我不知道J1an出了这个问题，发现之后立刻去关掉了我去年写的hgame2023 crypto wp😅）

```

1 from Crypto.Util.number import *
2 N=0x2be227c3c0e997310bc6dad4ccfeec793dca4359aef966217a88a27da31ffbcd6bb271780d8
   ba89e3cf202904efde03c59fef3e362b12e5af5afe8431cde31888211d72cc1a00f7c92cb6adb17
   ca909c3b84fcad66ac3be724fbcbe13d83bbd3ad50c41a79fcd04c251be61c0749ea497e65e408
   dac4bbcb3148db4ad9ca0aa4ee032f2a4d6e6482093aa7133e5b1800001
3 g=2
4 B=0x17e10a2083de26078d8b12793d5fbe442e9887e34185ad931da8768993c8aed1021c3518590
   4bc6983d460bcb387aeb382678ed1e93b778250642b3a9562278f2ef6e7711725babfdc03a079b5
   3f371320428de4dca7e1f68cff86f19aa1f2dda26e73f8e0134e52aa0e60ffef196a515e5b69c17
   9e0a9c3e8e387eb99a8f1ff22dfcd015f47567e772138b45d51b8c60e35
5 #Bob_secret=sympy.discrete_log(N,B,g)
6 #print(Bob_secret)
7

```



```

8 Bob_secret=11602779770217499822215719679451821031762640907179667170567949793457
1678761815772207892112249430038560233514115753081518545170230525854529272204524
58480453
9 A=0x23541ab31d368ed9f6f408fe5d88943ad932fcc18a0ee5dcd6626ff3a587e93fd0905023614
07c97681e2ede42ddac63a2d1ae90d69a7d89018aaaa0a20a72ec5e3b1c32ad9b52f045c152dc24
1cda3e1d3c5113d12b38b2a51c192a46c9ea596feecd48f014ba7ed79ec25013e792d6b34084df0
fbfdfffdd8c9be25f17137b962f0573a4accc6c054fbcfc840a3263de0d
10 p=10308557587164022706514635928683596042430744257161357917401709178525251949052
862844157076675265640736433114332060941186746356284152357428536662634011811471
11 a=-3
12 b=12025546603409903147849155748251759419142010234655737862009645598188095634720
748182992483657892210501454931311258190525269096993560769402996969969639642661
13 E = EllipticCurve(GF(p), [a, b])
14 shared_secret = pow(A, Bob_secret, N)
15
16 P1=E(70809900703286597169794287820413445810564745307504068010892741429267629075
6075896028267789874443773217228350468519594472439387297210453647558365792279061
3,
2832485697489487710275277424364437932500878908252148038877331553735414600413988
507867556321216633534775373232089463497194466132076392296803176716939083279)
17 c=E(979522334457133660374936534934015969226308364944635870980350982507427121493
6211743181681939343755265689293712730369079161405717970886618739746371443816105
,
5273993583802060289486447020494650280753482410237600353959483870090126674407588
299767257409341089774043265420102255566569099412592022804273930818968458978)
18 P2=shared_secret*P1
19 m=(c-P2)[0]
20 print(long_to_bytes(m))

```

[week3] ez_ecdlp

分两个部分，一个部分是发现 `P1.order()==N1` 之后打 `smartattack`，另一个部分是en算ecdlp，这不由得让我怀疑J1an是因为前几天春秋杯冬季赛那个**题被折磨之后出的这个来折磨选手



注：这里第二个部分的flag出小了，似乎并不需要利用CRT合并数据

```

1 def SmartAttack(P,Q,p):
2     E = P.curve()
3     Eqp = EllipticCurve(Qp(p, 2), [ ZZ(t) + randint(0,p)*p for t in
E.a_invariants() ])
4
5     P_Qps = Eqp.lift_x(ZZ(P.xy()[0]), all=True)
6     for P_Qp in P_Qps:
7         if GF(p)(P_Qp.xy()[1]) == P.xy()[1]:

```

```

8         break
9
10    Q_Qps = Eqp.lift_x(ZZ(Q.xy()[0]), all=True)
11    for Q_Qp in Q_Qps:
12        if GF(p)(Q_Qp.xy()[1]) == Q.xy()[1]:
13            break
14
15    p_times_P = p*P_Qp
16    p_times_Q = p*Q_Qp
17
18    x_P,y_P = p_times_P.xy()
19    x_Q,y_Q = p_times_Q.xy()
20
21    phi_P = -(x_P/y_P)
22    phi_Q = -(x_Q/y_Q)
23    k = phi_Q/phi_P
24    return ZZ(k)
25
26 #N1 = 92636417177965240871815246762704348071
27 N1 = 297342668339361548416629796745639177971
28 A1 =
4208715803791813173086894172778966025419787767340027559010619240548499823390
29 B1 =
11846440123913040489420209031751160809904311707943252241515965930654415480691
30 P1x =
479750084250968709343887919962436485997147832319843477221083468203689368148
31 P1y =
15452861783577624143044213767588871736433639621547613407582902947429567101675
32 P1 = (P1x,P1y)
33 E1 = EllipticCurve(Zmod(N1), [0, 0, 0, A1, B1])
34 P1 = E1(P1)
35 Q1 =
E1(2222272585170140628429982927710250032383314762510568687410283421473372012956
6,11457880282290536070683750562309262342859411139347565743387209585550769930346
)
36 print(P1.order()==N1)
37 print(SmartAttack(P1,Q1,N1))

```

```

1 #N2 = 6471339743593595797696002766822660599108196938080465998531085409467
2 p1 = 69857405335111415530599248077
3 p2 = 92636417177965240871815246762704348071
4 A2 = 3199218821393204771660095172457569312269694438403110131957204042314
5 B2 = 762889472027318213897694878260359911054972690369935049954326689904
6 P2x = 2557373437970770011124755960432555084678930336188254243278984381842
7 P2y = 4442763096366920105760404533052204677305995021662082361185473321644

```

```
8 P = (P2x,P2y)
9
10 E2 = EllipticCurve(Zmod(p1), [0, 0, 0, A2, B2])
11 P2 = E2(P)
12 Q2 =
    E2(850210212566920982015201804401545577446922087165778054847791802617,399954305
    5830043575258541123873518686116525200791513741952645059609)
13
14 E3 = EllipticCurve(Zmod(p2), [0, 0, 0, A2, B2])
15 P3 = E3(P)
16 Q3 =
    E3(850210212566920982015201804401545577446922087165778054847791802617,399954305
    5830043575258541123873518686116525200791513741952645059609)
17
18 print("1")
19 o1 = P2.order()
20 o2 = P3.order()
21 d1 = discrete_log(Q2,P2,operation='+')
22 print(d1)
23 d2 = discrete_log(Q3,P3,operation='+')
24 print(d2)
25 d = crt([d1,d2],[o1,o2])
26 print(d)
```

