【CTF WriteUp】2020网鼎杯第一场Crypto题解

原创

零食商人 于 2020-05-10 17:23:53 发布

Crypto

boom

本地运行，过三关。第一关找个东西md5值等于给定值，破解一下得到en5oy；第二关解三元一次方程，得x=74, y=68, z=31；第三关解一元二次方程，找正数根x=89127561。全过得flag

you raise me up

本题中，模数n = 2 ** 512，所以此处可以使用Pohlig-Hellman算法逐渐升模求出flag。该算法的原理核心，在于已知x % 2 ** k的情况下，如何求出x % 2 ** (k+1)。

(i)当k = 0时，2 ** k = 1，显然有bytes_to_long(flag) = x % 2 ** k = 0
(ii)假设我们已经知道了x % 2 ** k = t，现在想要知道x % 2 ** (k+1)的值。设x = 2 ** k * x0 + t，根据代码有

```
c = m**x = m ** (2**k * x0 + t) = (m ** (2**k * x0)) * (m ** t)  (mod n)
c * (invert(m, n) ** t) = m ** (2**k * x0)  (mod n)
```

左边都是可求的，关键在于右边的处理。我们希望能够知道x0是奇数还是偶数，如果x0是偶数，那么x % 2 ** k+1依然为t；否则结果为t + 2 ** k。于是我们把等式两边都进行2 ** s次方，于是有

```
(c * (invert(m, n) ** t)) ** (2**s) = m ** (2**k * x0 * 2**s)  (mod n)
```

m是个奇数，所以与n互质。根据欧拉定理，m ** phi(n) = 1 (mod n)。phi(n)代表小于n中所有与n互质的正整数的个数，在本题中即为小于n的奇数个数，为2 ** 511。继续验证会发现：

```
pow(m, 2**511, n) = 1
pow(m, 2**510, n) = 1
pow(m, 2**509, n) > 1
```

如果我们取s = 509 - k，这样等式右边m的幂指数就变成(2 ** 509)*x0。如果x0是偶数，此处就可以被2 ** 510整除，等式右边就变成1；反之x0是奇数，等式右边就变成了m ** (2 ** 509) (mod n)，这个数不为1。而左边的所有内容都是可以算的，因此可以通过左边是否为1来判断x0的奇偶，进而求出x % 2 ** k+1的值。

完整代码如下：

```
#!/usr/bin/python
#-*- coding:utf-8 -*-
import gmpy2
from libnum import n2s

n = 2 ** 512
m = 391190709124527428959489662565274039318305952172936859403855079581402770986890308469084735451207885386318986
881041563704825943945069343345307381099559075
c = 666585139420321424585678945072365863252081679162179677590976689523300023402364287878602564495379799537321130
848560539702412318008592411761080248597258449 9

t = 0
invb = gmpy2.invert(m, n)
for k in range(509):
    tmp = (c * pow(invb, t, n)) % n
    tmp = pow(tmp, 2**(509 - k), n)
    if(tmp!=1):
       t += 2**k
print n2s(t)
```

其他解法

这题n = 2^512不是很大，因此可以用sage直接秒

```
n = 2 ** 512
m = 391190709124527428959489662565274039318305952172936859403855079581402770986890308469084735451207885386318986
881041563704825943945069343345307381099559075
c = 666585139420321424585678945072365863252081679162179677590976689523300023402364287878602564495379799537321130
848560539702412318008592411761080248597258449 9
ZmodN = Zmod(2^512)

m = ZmodN(m)
c = ZmodN(c)

print c.log(m)
```

easy_ya

本题为环境题，连过去之后首先是一个Proof Of Work，内容由4个可打印字符组成，要求其hash值（注意hash方法会变化）的前20个字符为特定内容。写程序爆破过即可。好在本题无后续交互，只需要做一遍拿到数据即可。过了POW以后要提交队伍token，交互内容如下：

```
λ nc 39.96.90.217 17497
POW(printable)
x[:20] = 0c0a899b9049339fe1fd
<built-in function openssl_sha512>
> #8nu
Please input your token: icq6778fc9a63ebc19ac0d5c4799f686
0x65d4ce3b0b1b3f48bb9fdL
0xf0230f43414a9c9ac0488L
0xbd592ebe04025b783fb5bL
0x28d194dcd1c79b4bb8074L
0x7c493be8f0fdbb740ec29L
523302236767994199900474363344509016779069116440336509402579663927023212822016176967224920257215697515649639915194389392664393435704126520500342935380422629852958893595198501678886910437288107490865842806189885864227955305292596726620737250011643028592306930279653650874633693082819617471411085797465171317473575199984137570425840763157847215554005503026976635376775672263338018024842152016782550728415532850496820668169596712488535626476321971915772540785626943870257914929277013026283202706480202573253694027965561121582254805059775814137542861056288774182091519461752761702279929984980399296157122817271523455097901334302748985253698791419583126820263035649222528798493821648538568572195005358978375666451378629463679501572550304224399031869931606326980885244883575380615433423693907673692911481956161922234589747374211957098461625670478121184734721478951324090304946630772999533103517416951906651316589244984076827527352185687877958699658011472950078992791104307370649334689901210147765149584862868749574640976109508247204313701721196275177445231013673265642224568068643851710967790673640935447579098048865343477978938147696617434878198467680874826616766904653374028721826187308127128623535689512750830887981058826202565202400511498116479126828060721351777093052860074248131123861633687756928866328288114135612849582257727497809922257024131348207980532057240060103798568227977564521497134127080257805342446192248133124844549442334197110567598270256443362563074473166503647429611051333058467042150637666479765740465760837773368463067714839472516227071241189038004851446423799255357523722335018334006275302424683874359612832738469924300750750078529793617519763670882091967677827342326944222743651042579235831828367996040140715066967834876505105847352068608085060284716208922782057542541292881761158407247706056426656079240060590899310402168208571288608455268870805780135942398347873755131825844602660879197979908250552518202142180574849619997871142575611493295891685289794234943270549985371377356916876993228441993132176346052582480841840804748103391892367360829385946406970449784716173650039815882209635063166095538900798262007788490773894562344153159899172908210604994390998911915844110196707427658085882905282469152575391386020890630965576157859422503998168371754442071688959425226316542691412155479949609490654136752136268169657071799792559095529184991835794530321488538421681769778618794781798023171785137030552612046467752539222563124502252249216633629456799053699058869997259517104469625277661903217941478712515410225036472670310527285546698901400933609424770210891844094217108058732611592907771742194956029486947118236668497133789513359969711219377800792445039910707973247184794559823234910383917863274595901409809925947810801523788484660537782346172645868448689924998004174305607989627751083896823072974502051002938622805773194416922082008505978954197109276983203418146359602343859753100633270064891076975093660747639031114030191274267700106589347034095041936064299870030348871682676606636326281904215806439865695618534813588022526789402941981479793172040559465677731109438967699833830980999036786292613931775140998490830680179915200517972413799762602382639278024563069864802394511961767962794541539804610687171547518389073841731192574743022204072539434702273448993006971209219762916364731777233035588738754300808515095910442739194202398529899728836835763806245406663610000054847380989129973234306982579329370640780578258715012334208183998360919905940150292578303976045073051960307051913608670382543880705878768804499117273757543755924966847135204608561434518616159515225619367299472254003992021705845121588986263894651018831158204686629942399765951539593487116752246370608518696224242185916440211151441687048015192744910734916382008453358070938399392930118756933885249862207310667864949685811759353894772074064033966815303362040947174470018370959127628070432398543733045254194926285253171017624356188839737468397275917718168456089152912356937124198060235882313880801448847128485540145601102010993704690543496782861646681282105222292469109080887070097079594270576616386279040641714854021033972227390282546946009952076906152202404212376653662055442668361209921979092157340327824176034441892563082380636694903990127953262418797621161152886046168707309688845069417545932119357192548094963597707933928490545247616116568407397514317936282177960612717137877101769801870833011719585733647656773893067209953623172640181863923665519993686574221421639192629684497967944071191999961782848918674720606882936028961473980287178745425671436350716455950767670663619893451972701108420620445090384684061202710614653087751269275797414606844202906199835329300172012235041439111403882025273749304932486209933063210329615378763233804435230938147302042342906594493918571987967435872658940694933762917131980730231789770708346145846509135977637294146556803807514074087692893439056375014346378299974475429016205283231333324476672587316837005423476551
```

根据代码可以看到，后边的四个长数是n1、c1、n2和c2，其中n1和n2有共同的质因数p，所以可以简单求公约数后得到n1的质因数分解，用RSA的方法逆推回ek

```
import gmpy2
from libnum import n2s,s2n
import string

n1 = ...
c1 = ...
n2 = ...
c2 = ...
p = gmpy2.gcd(n1, n2)
q1 = n1 // p
q2 = n2 // p
e = 0x10001
phi1 = (p-1)*(q1-1)
d1 = gmpy2.invert(e, phi1)
m = pow(c1, d1, n1)
ek = n2s(m)
```

结果发现ek是中文：愿我所爱无忧恙岁长安，根据ek的生成规则，我们可以知道ek是\xe6\x84\xbf开头，加上padding与key交错生成的内容，所以可以得到key为

```
8891898088b197a0bfa78199b28195bfae89
```

因此可以算出题目中的a、b、c和d

```
key = '8891898088b197a0bfa78199b28195bfae89'.decode('hex')
limit = lambda n: n & 0xffffffff

Key  = [ord(i) for i in key]
a = limit((Key[0] << 24) | (Key[1] << 16) | (Key[2] << 8) | Key[3])
b = limit((Key[4] << 24) | (Key[5] << 16) | (Key[6] << 8) | Key[7])
c = limit((Key[8] << 24) | (Key[9] << 16) | (Key[10] << 8) | Key[11])
d = limit((Key[12] << 24) | (Key[13] << 16) | (Key[14] << 8) | Key[15])
```

接下来，我们发现题目在加密时，每8个字符作为一组，利用该组生成y和z然后对y和z进行运算，返回结果。注意到返回的结果为

```
hex((y << 52) ^ (pads << 20) ^ z)
```

所以我们可以知道，
1.返回结果（视作84位）的前32位是y；
2.返回结果的第33位至第54位为pads的前20位，第55位至第64位为pads的后12位和z的前12位异或的结果。
注意到y和z一共运算了32轮，所以这时pads应该是limit(32*pad)，其后5位是0，所以pads的情况相当于：

```
????? ******** ******** ******** ???00000
```

其中 * 代表已知，? 代表未知，0就是0。去掉后边5个0后，另外32位就是pad。这个pad中，未知的地方只有12位，总计4096种情况，每种情况对应唯一的z。所以可以通过爆破这4096种情况，逆推32轮得到初始的y和z，再观察其转为字符后是否在string.printable中来确定答案。完整代码如下：

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
import gmpy2
from libnum import n2s,s2n
import string
```

```
n1 = 523302236767994199900474363344509016779069116440336509402579663927023212822016176967224920257215697515649639915194389392664393435704126520500342935380422629852958939595198501678886910437288107490865842806189885864227955305292596726620737250011643028592306930279653650874633693082819617471411085797465171317473575199984137570425840763157847215554005503026976635376775672263338018024842152016782550728415532850496820668169596712488535626476321971915772540785626943870257914929277013026283202706480202573253694027965561121582254805059775814137542861056288774182091519461752761702279929984980399296157122817271523455097901334302748985253698791419583126820263035649222528798493821648538568572195005358978375666451378629463679501572550302243990318699316063269808852448835753806154334236939076736929114819561619222345897473742119570984616256704781211847347214789513240903049466307729995331035174169519066513165892449840768275273521856878779586996580114729500789927911043073706493346899012101477651495848628687495746409761095082472043137017211962751774452310136732656422456806864385171096779067364093544757909804886534347797893814769661743487819846768087482661676690465337402872182618730812712862353568951275083088798105882620256520240051 1
c1 = 498116479126828060721351777093052860074248131123861633687756928866328288114135612849582257727497809922257024131348207980532057240060103798568227977564521497134127080257805342446192248133124844549442334197110567598270256443362563074473166503647429611051333058467042150637666479765740465760837773368463067714839472516227071241189038004851446423799255357523722335018334006275302424683874359612832738469924300750750078529793617519763670882091967677827342326944222743651042579235831828367996040140715066967834876505105847352068608085060284716208922782057542541292881761158407247706056426656079240060590899310402168208571288608455268870805780135942398347873755131825844602660879197979082505525182021421805748496199978711425756114932958916852897423494327054998537137735691687699322844199313217634605258248084184080474810339189236736082938594640697044978471617365003981588220963506316609538900798262007788490773894562344153159899172908210604994390998911915844110196707427658085882905282469152575391386020890630965576157859422503998168371754442071688959425226316542691412155479949609490654136752136268169657071799792559095529184991835794530321488538421681769778618794781798023171785137030552612046467752539222563124502252249216633629456799 0
n2 = 536990588699972595171044696252776619032179414787125154102250364726703105272855466989014009336094247702108918440942171080587326115929077717421949560294869471182366684971337895133599697112193778007924450399107079732471847945598232349103839178632745959014098099259478108015237884846605377823461726458684486899249980041743056079896277510838968230729745020510029386228057731944169220820085059789541971092769832034181463596023438595731006332700648910769750936607476390311140301912742677001065893470340950419360642998700303488716826766066363262819042158064398656956185348135880225267894029419814797931720405594656777311094389676998338309809990367862926139317751409984908306801799152005179724137997626023826392780245630698648023945119617679627945415398046106871715475183890738417311925747430222040725394347022734489930069712092197629163647317772330355887387543008085150959104427391942023985298997288368357638062454066636100000548473809891299732343069825793293706407805782587150123342081839983609199059401502925783039760450730519603070519136086703825438807058787688044991172737575437559249668471352046085614345186161595152256193672994722540039920217058451215889862638946510188311582046866299423997659515395934871167522463706085186962242421859 2
c2 = 164402111514416870480151927449107349163820084533580709383993929301187569338852498622073106678649496858117593538947720740640339668153033620409471744700183709591276280704323985437330452541949262852531710176243561888397374683972759177181684560891529123569371241980602358823138808014488471284855401456011020109937046905434967828616466812821052222924691090808870700970795942705766163862790406417148540210339722273902825469460099520769061522024042123766536620554426683612099219790921573403278241760344418925630823806366949039901279532624187976211611528860461687073096888450694175459321193571925480949635977079339284905452476161156840739751431793628217796061271713787710176980180703301171958573364765773893067209953623172640181863923665519993686574221421639192629684497967944071191999961782848918674720606882936028961473980287178745425671436350716455950767670663619893451972701108420620445090384684061202710614653087751269275797414606844202906199835329300172012235041439111403882025273749304932486209933063210329615378763233804435230938147302042342906594493918571987967435872658940694933762917131980730231789770708346145846509135977637294146556803807514074087692893439056375014346378299974475429016205283231333324476672587316837005423476510 3
p = gmpy2.gcd(n1, n2)
q1 = n1 // p
q2 = n2 // p
e = 0x10001
phi1 = (p-1)*(q1-1)
d1 = gmpy2.invert(e, phi1)
m = pow(c1, d1, n1)
ek = n2s(m)

key = '8891898088b197a0bfa78199b28195bfae89'.decode('hex')
limit = lambda n: n & 0xffffffff
```

```
Key  = [ord(i) for i in key]
a = limit((Key[0] << 24) | (Key[1] << 16) | (Key[2] << 8) | Key[3])
b = limit((Key[4] << 24) | (Key[5] << 16) | (Key[6] << 8) | Key[7])
c = limit((Key[8] << 24) | (Key[9] << 16) | (Key[10] << 8) | Key[11])
d = limit((Key[12] << 24) | (Key[13] << 16) | (Key[14] << 8) | Key[15])

outputs = [0x65d4ce3b0b1b3f48bb9fdL, 0xf0230f43414a9c9ac0488L, 0xbd592ebe04025b783fb5bL, 0x28d194dcd1c79b4bb8074
L, 0x7c493be8f0fdbb740ec29L]

def reversecalc(a, b, c, d, y, z, pad):
    for i in range(32, 0, -1):
        # print 'Round %d: %d, %d' % (i, y, z)
        pads = limit(pad * i)
        paramz = (y*16+c)^(y+pads)^((y>>5)+d)
        if(z < paramz):
            z = limit(z - paramz + 0x100000000)
        else:
            z = limit(z - paramz)
        paramy = (z*16+a)^(z+pads)^((z>>5)+b)
        if(y < paramy):
            y = limit(y - paramy + 0x100000000)
        else:
            y = limit(y - paramy)
        # print (y, z)
    return y, z

for output in outputs:
    print hex(output)
    binout = bin(output)[2:]
    binout = '0'*(84-len(binout))+binout
    y = int(binout[:32], 2)
    for i in range(4096):
        bini = bin(i)[2:]
        bini = '0'*(12-len(bini))+bini
        tmpbinpads = bini[0:5] + binout[32:52] + bini[5:12] + '00000'
        pad = int(tmpbinpads, 2) / 32
        pads = limit(int(tmpbinpads, 2))
        z = output ^ (y<<52) ^ (pads<<20)
        y0, z0 = reversecalc(a, b, c, d, y, z, pad)
        tmpstr = n2s(y0)+n2s(z0)
        valid = True
        for j in tmpstr:
            if (not j in string.printable)and(j != '\x00'):
                valid = False
                break
        if(valid):
            print tmpstr
```

输出结果如下：

```
0x65d4ce3b0b1b3f48bb9fdL
S60 '\M(
flag{5fe
0xf0230f43414a9c9ac0488L
86c73a3e
0xbd592ebe04025b783fb5bL
Us
NU\KNP-F
381ee168
0x28d194dcd1c79b4bb8074L
)^C04<b5
Z"N*'U_x000C_=
83c04451
GNM1!Uyl
0x7c493be8f0fdbb740ec29L
9o`YE
1356f}
QG_x000C_rVHZo
7ju-px/_x000C_
```

在其中找看起来像的，拼接得到flag