Author: Zijun Ping

Created by: Typora

Owned by: Zijun Ping

11.Crypto-格密码

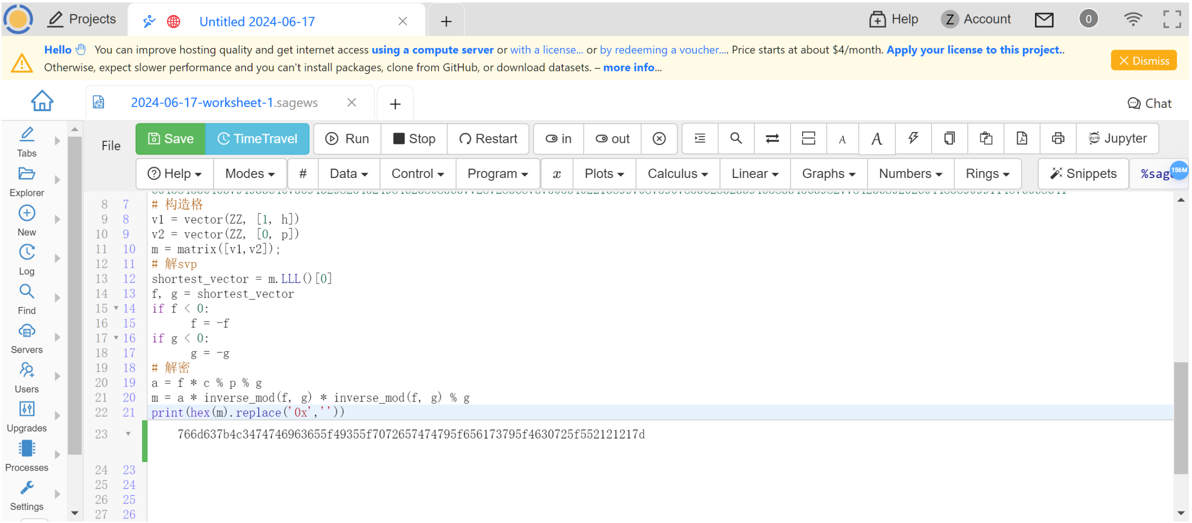
用python脚本构造格lattice：，然后求解

$$[k \quad a] \cdot \begin{bmatrix} P & 0 \\ h & 1 \end{bmatrix} = [g \quad a]$$

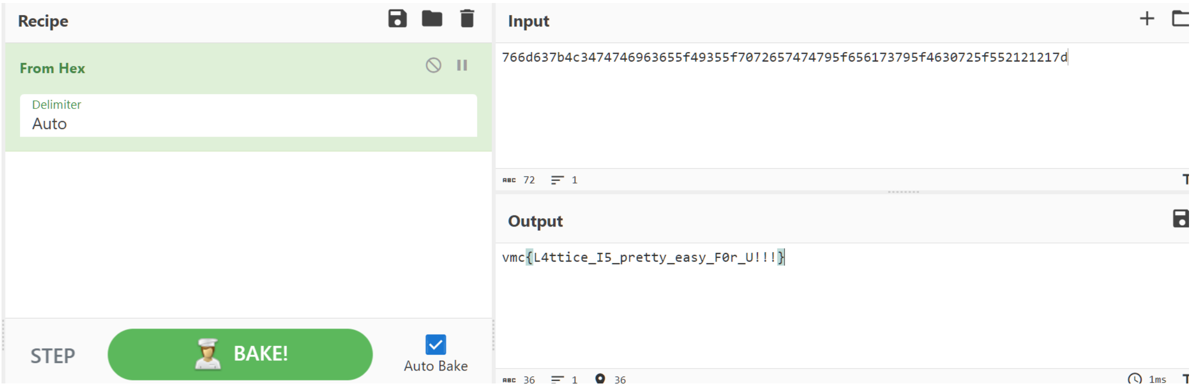
```python
from Crypto.Util.number import long_to_bytes
from Crypto.Util.number import *
from gmpy2 import invert
h=3119703644257993669984897588163519646148981645970159623905228587600893312102695077694083302457458885417050598823967223407746819354436158884034949014152266512952011325566132912895909002469925535025136645610582307138406160523107508501440775527011425638728376637896363049684239194647075454820086470873120639490247441953181603356520744078746347422734931081122121101236919093468466121154790239501192867162436046306642356338317840833004311612251318830651507785011784586018852639439079646128518413561284922837594011293496552633604964186065346378310655640546271245452661301465850125689480180834200268677914836249879518763299683154106127489894526284103318135401434258734813512204996818159795694792859056512702083083967192822475860088167115275989157978856188626541686230020127466841022457337574288285172779526754412902630516189871219566466719013573705857844096336971653391824063417643202936309836804160555849477723822822599660445176835384334098362569959808049010934051085812431434137033396570716222791402363694651542203639939230855830497205123833845771599751349104494659032275554195524926064391491530156746030476953457003098702060347433173617863337320709623625724274115972662527073974514231955124491033671202357375506503544276902435249004496759675
```

```
p=958396606448120961344481821891302529131234571519205540072533929007120447482644938027839069855225984909598017115601687321093037138370623689366302399472755221666720998204960391688686871213784218139146955110040572010513513710257471193690475085340782434134757767894370195789546551282060123540167285748706110004061656181247087025993389812671658906912145384679566372037915179523706023309666826191629634851598509448675340758156737530185937223948687001934035933966720541257653562031308634592003639145311246147664928094381997264652669457904886666262851691565920076364034433600062253014377522558972668765617631467665085324790312091927917199333642804731490452686230714946707520153374983386856748968281865500447110447064499127304576389450231449458997253721401398074788997192735673939296531420329207264490272774186484762804708405995307032141715953080254098506768585899454119852852002578252291310329200959476618660765556861112558404254680356703540148703891793429589128285327230398620737214653509556883409293525132433655639546382948760666188308135216174434756100646981938900940757869329579150360836505992616459066632938605335205569767007863591747811468281490752448411521431430302168214014717351186242150709061058943921599462022604009950194741903607
c=767158168672362136291238223905378538545444315829681403660668665037917788547250510431680762324989504337886911129872811283399423182492713085621340850466620775416712732614020830060267436728824312319343522189998875569791210615658554329492772575522352284894778864900683043087475949382767603528807809596004213194946687583749277851731545121301799909443549367647188538291294493900118514585149301404415365107375148502163345229200555520367376148986792681776613941685900525114359918025093194374564154918465789098039688273260831196880453048787881692220588699313824407443049610470566611891830120455606865304514498141918370250200060333776115471607369861567714061571248634366226901360555240991881037749649258353302128601187117945261098036478588953880956746052307508140721683848770369771427200654010686745459084737649483182116223928908511586372587312845173805333061611628627699056444060210866689908925857413693914088384801590539118419817748706914585535977225375257595014121709225398146804929559026288825337416874187757876993164395155899845807880799091660701491902131855351054079661782867046531376623992742661604584530403794933640735943298254524984623808363772872836875750554022138997087590785362832339405833456398277842368920260448389099114876068641
# 构造格
v1 = vector(ZZ, [1, h])
v2 = vector(ZZ, [0, p])
m = matrix([v1,v2]);
# 解svp
shortest_vector = m.LLL()[0]
f, g = shortest_vector
if f < 0:
    f = -f
if g < 0:
    g = -g
# 解密
a = f * c % p % g
m = a * inverse_mod(f, g) * inverse_mod(f, g) % g
print(hex(m).replace('0x',''))  # 766d637b4c3474746963655f49355f7072657474795f656173795f4630725f552121217d
```

用https://cocalc.com/在线运行sagemath解密上面脚本

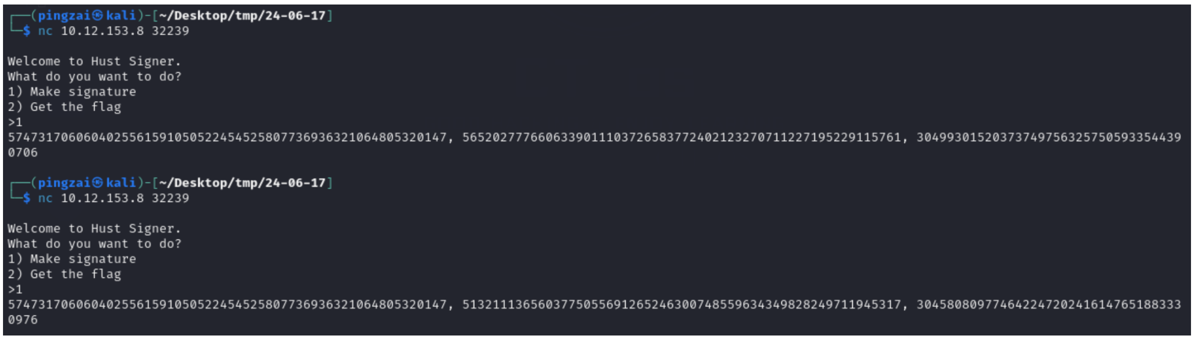


再用cyberchef解码得flag `vmc{L4ttice_I5_pretty_easy_F0r_U!!!}`



12.Crypto-签名算法

发现r不变，是共享k



```
┌──(pingzai㉿kali)-[~/Desktop/tmp/24-06-17]
└─$ nc 10.12.153.8 32239

Welcome to Hust Signer.
What do you want to do?
1) Make signature
2) Get the flag
>1
5747317060604025561591050522454525807736936321064805320147, 5652027776606339011103726583772402123270711227195229115761, 304993015203737497563257505933544390706

┌──(pingzai㉿kali)-[~/Desktop/tmp/24-06-17]
└─$ nc 10.12.153.8 32239

Welcome to Hust Signer.
What do you want to do?
1) Make signature
2) Get the flag
>1
5747317060604025561591050522454525807736936321064805320147, 5132111365603775055691265246300748559634349828249711945317, 304580809774642247202416147651883330976
```

如果在两次签名的过程中共享了 k，我们就可以进行攻击。

假设签名的消息为 m1,m2，显然，两者的 r 的值一样，此外

$$s_1 \equiv (H(m_1) + xr)k^{-1} \bmod q$$

$$s_2 \equiv (H(m_2) + xr)k^{-1} \bmod q$$

这里我们除了 x 和 k 不知道剩下的均知道，那么

$$s_1 k \equiv H(m_1) + xr$$

$$s_2 k \equiv H(m_2) + xr$$

两式相减

$$k(s_1 - s_2) \equiv H(m_1) - H(m_2) \bmod q$$

此时 即可解出 k，进一步我们可以解出 x。

若k被泄露，DSA签名算法也会被完全破解。可以计算出私钥x：

$$x = (sk - h(m)) * r^{-1}\ mod\ q$$

但是`p`和`q`还是未知的，查看代码中使用给的`ecsda.enerator_192`，猜测使用的NIST P-192椭圆曲线，其参数固定。故有解密脚本：

```python
import random
from ecdsa import ecdsa as ec
from datetime import datetime
import hashlib
import gmpy2
p = 6277101735386680763835789423207666416083908700390324961279
q = 6277101735386680763835789423176059013767194773182842284081
str1 = "3470545216309636255008853373556493008548980240643905225, 5575093540335095931762919420534746169532440392768986557355, 1053674520760038124359502110783630886"
str2 = "3470545216309636255008853373556493008548980240643905225, 1543509685086703115590790680094992130950572761005237727533, 2374781241738991141103267058089737994222"
msg = "18:28:48:get_flag"
r1, s1, m1 = (int(k) for k in str1.split(", "))
r2, s2, m2 = (int(k) for k in str2.split(", "))
r = r1
ds = s2 - s1
dm = m2 - m1
k = gmpy2.mul(dm, gmpy2.invert(ds, q))
k = gmpy2.f_mod(k, q)
tmp = gmpy2.mul(k, s1) - m1
x = tmp * gmpy2.invert(r, q)
x = gmpy2.f_mod(x, q)
RNG = random.Random()
g = ec.generator_192
N = g.order()
secret = x
PUBKEY = ec.Public_key(g, g * secret)
PRIVKEY = ec.Private_key(PUBKEY, secret)
hash = int(hashlib.md5(msg.encode()).hexdigest(), 16)
nonce = RNG.randrange(1, N)
```

```
signature = PRIVKEY.sign(hash, nonce)
print(f"{signature.r}, {signature.s}")
```

```
┌──(pingzai㉿kali)-[~/Desktop/tmp/24-06-17]
└─$ nc 10.12.153.8 32239

Welcome to Hust Signer.
What do you want to do?
1) Make signature
2) Get the flag
>2
Get signature for md5("06:16:51:get_flag")
731068159329242351098641310447018125047823951069208437045, 3239895669199212000782886318118283300127840373857664056524
Congratulation! Here is your flag:vmc{0OyEUykgFRYedHQdYnleQVcet4IK07xM}

┌──(pingzai㉿kali)-[~/Desktop/tmp/24-06-17]
└─$ 
```

13.Crypto-RSA

正常RS的加密过程，p1和p2是相邻素数，且差值小于1000，secret.txt中的N1和N2也很接近

```
p1 = getStrongPrime(1152)
p2 = nextprime(p1)
try:
    if p2 - p1 > 1000:
        raise Exception("Error")
except:
    exit()
```

参考https://blog.csdn.net/jcbx_/article/details/109306542利用连分数破解比例接近的p1和p2

那怎么通过$\frac{k}{dg}$得到p和q呢?

我们已知:

$$edg = k(p-1)(q-1) + g, \quad k > g$$

所以

$$\lfloor \frac{edg}{k} \rfloor = (p-1)(q-1)$$

又因为

$$\frac{pq - (p-1)(q-1) + 1}{2} = \frac{p+q}{2}$$

$$(\frac{p+q}{2})^2 - pq = (\frac{p-q}{2})^2$$

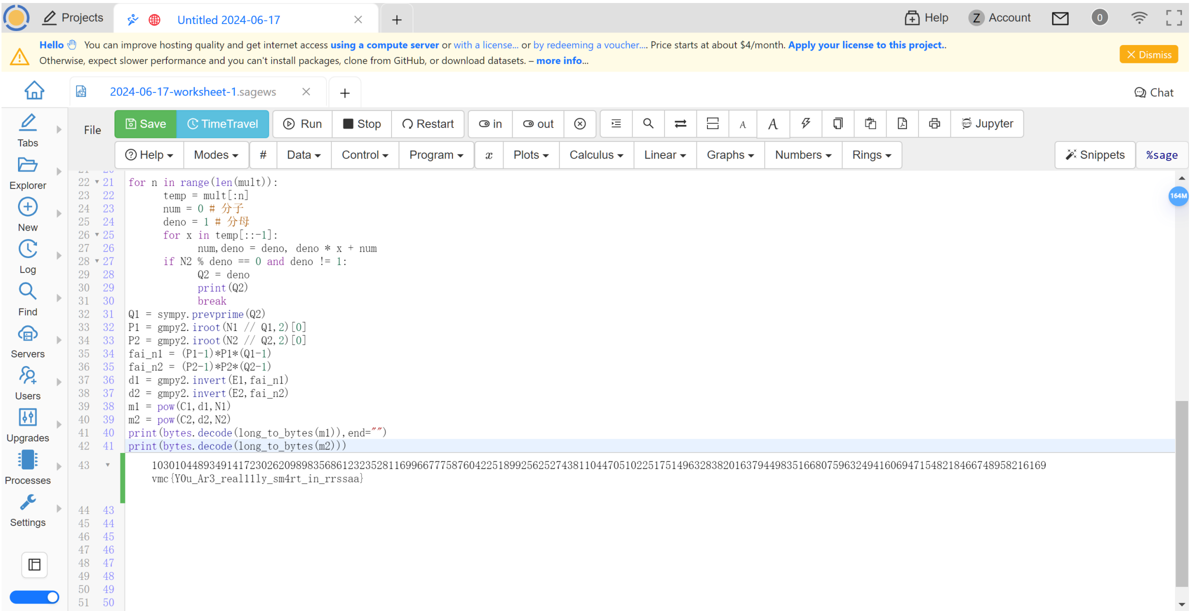
所以我们可以通过判断$(\frac{p-q}{2})^2$是否为平方数来确定我们找的连分数是否正确，然后再算出p和q就行了。

```
from Crypto.Util.number import *
import gmpy2,sympy
N1=27682578737141139764880192910976946263355689816882797515059917479242862799083599745594956880258244112867559722435850732812023189662581052511287867553308318268020022386306820424829898858029986193412922645944359409248568131057377380697236238480724883073062491532254626363468032145049953168789073328812076794158602028961853986034378144749656228541552641207393473830715156452473432130040360471566096165146087202836036783304640579183082301858529818598032339821237841219774124710789761912675044056265735587753304064079484844965820681168729776560497921764083742448045654891113500035063474318442078036531813957551086231747079155691690001433127187382636049871228279519466735719768798574776353687049667125384146566107739705553580693984918816215940308884007192621418304753551998125658993859095063641090798574130161651257890916914325076137436869018454577522833
```

```python
C1=14360977893873474578201937159000122429359790977572665232657843468076201963407780015131857192621550737338805880514393357390576423731328871867241029260294051045710144482989801857054158816998897546124709802730198690244128545073634486145786763294634081834588146373913232490890078533918320777534358739106486350300547206365723045306767038214923412032633833255742963954701475401704385045019069883734625251436409851588044241336835452728962860280865504000103361559688861149086469939940113748174610019620309023214292662384279070127090992947332945432141695583191136521301940116585610033790348125114471980285332011918355578839128892075058698885319243593345096734776497817461251643381989958326810478500026684389358920342021836572511688796450072700142033952403561408907486022094802237175920044147084170050294965826258250618675638343726352907476393474128674488943
E1=13890651822147152152440433003961663329775276553417657086890003923713341985748541563942319663606839723729622444208321376863048810071797788441534210423928095042473512914798605311533592878319037769524825092637473498810897213634962596575364999214632281035276824604157539672166114224672974757283201751024174908243 1

N2=27682578737141139764880192910976946263355689816882797515059917479242862799083599745594956880258244112867559722435850732812023189662581052511287867553308576254232706953290519059976159239205559295965110148734449650209977235953163255494808056707188551192674128213090005439928085856216617642935948961573449294338310127166077195263402939848861686214485115686799032901147314759348481062418109120418661302585413868782602282463165171129063197961455779193665041902822948963032580054067050227612838335828201043413949164885293325493829570131849345344856137656453666135670724974184749115550720826497558763320127218251970576144750319782121194483563545371157323166968983176013145267856898865437101799958588342741257457472036311490402279982286349929050116350394664561659857216725849236910894778018502118673902399095646487808462155207034764432342699549109080808769
C2=11293777290569693972360166961981727494638218221438571150393361751316389824613571820229370915191500766619410597117671232443452691634734112652285521806824284959073558010661204730954928847260946403867297932862687770449632506087883187920107766050673462588812979708792790888354008526054467620780053118019643408427959406056087370960170992834047890080269663747877143270683069575318397144844481262382463469080755423097527007161449411933936669451476467352049264455203632729909164666006688294056405955940041007137719228484035343153943155100892867641033645111253109951972003798413524003505574000784399458705347262353155363413513341797868942085136977548116650815336000627353933708913237438841909324920070013498153627767891674969586034872104569344923832761239959420543717354211689339868787331074579605476477152218068810732089913023456240425720821047030224659918
E2=13890651822147152152440433003961663329775276553417657086890003923713341985748541563942319663606839723729622444208321376863048810071797788441534210423928095042473512914798605311533592878319037769524825092637473498810897213634962596575364999214632281035276824604157539672166114224672974757283201751024174908261 9
def exgcd(x,y):
    mult = []
    if y > x:
        x,y = y,x
    while y:
        mult.append(x//y)
        x,y = y,x % y
    return mult

mult = exgcd(N2,N1) # N1<N2

for n in range(len(mult)):
    temp = mult[:n]
    num = 0 # 分子
    deno = 1 # 分母
    for x in temp[::-1]:
        num,deno = deno, deno * x + num
    if N2 % deno == 0 and deno != 1:
        Q2 = deno
```

```
            print(Q2)
            break
Q1 = sympy.prevprime(Q2)
P1 = gmpy2.iroot(N1 // Q1,2)[0]
P2 = gmpy2.iroot(N2 // Q2,2)[0]
fai_n1 = (P1-1)*P1*(Q1-1)
fai_n2 = (P2-1)*P2*(Q2-1)
d1 = gmpy2.invert(E1,fai_n1)
d2 = gmpy2.invert(E2,fai_n2)
m1 = pow(C1,d1,N1)
m2 = pow(C2,d2,N2)
print(bytes.decode(long_to_bytes(m1)),end="")
print(bytes.decode(long_to_bytes(m2)))
```

用https://cocalc.com/在线运行sagemath解密上面脚本



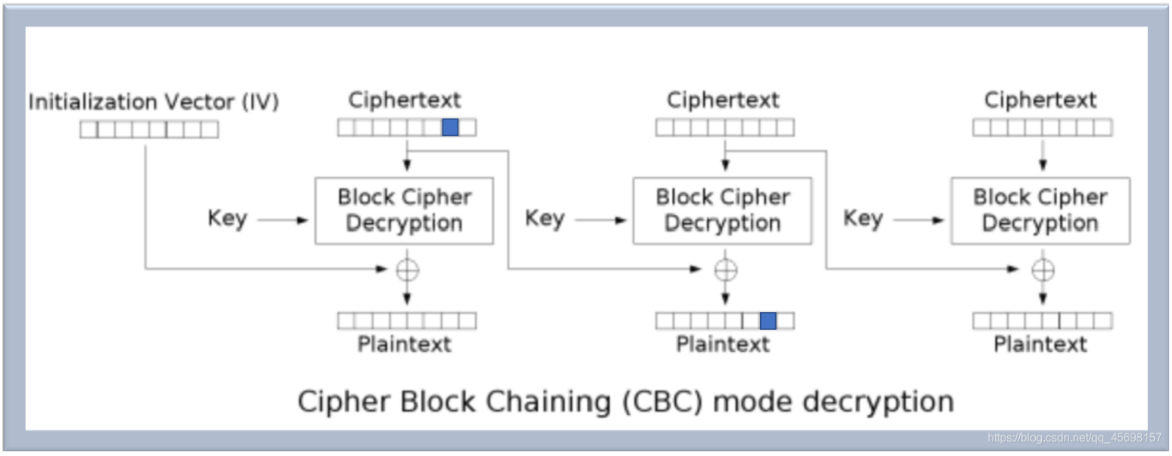
14.Crypto-分组密码工作模式

AES,CBC 16位明文，已知初始向量IV 和 密文，破解密钥。

```
    s.send("Enter your token > \n".encode())
    try:
        authcode = read_line(s).decode()
        user_key = bytes.fromhex(authcode)[:16]
        code = bytes.fromhex(authcode)[16:]
        cipher = AES.new(auth_key, AES.MODE_CBC, user_key) #
        token = cipher.decrypt(code) # AdminAdmin!_____
```

使用CBC反转攻击

三、CBC字节反转攻击原理



字节反转攻击原理文字描述:

我们从上述图片可以看出，如果猜测出明文分组P_n，那么可以通过修改密文分组C_{i-1}为C_{i-1} ***XOR*** P_n ***XOR*** A，那么解密出来的P_n的结果是A。当然我们一般是不可能直接猜出来整个明文分组P_n，我们更多遇到的是上述图片描述的那样，仅知道P_n中部分明文。同样地，我们依旧可以发动字节反转攻击，仅需要针对正确的位置进行修改密文内容即可。

这种攻击的精髓在于：通过CBC的加密特点改变明文内容，但是这种改变并不会引起其它明文块儿对应位的改变。这种攻击常用来绕过过滤器，提权（比如从guest变为admin）等。

```python
token = "8ed436b2c20f007a7b524ea36cb8cefb5797aeed0a922ed6f46276c941205c42 "
iv = bytes.fromhex(token[:32])
token1 = b'HUSTCTFer!______'
token2 = b'AdminAdmin!_____'
iv2 = b''
for i in range(16):
    k = token1[i]^token2[i]^iv[i] # 先异或原来的明文抵消掉原明文，再异或替换的明文
    iv2 += k.to_bytes()
iv = iv2
token = iv.hex() + token[32:]
print(token)
```

```
┌──(pingzai㉿kali)-[~/Desktop/tmp/24-06-17]
└─$ nc 10.12.153.8 32243
Only admin can get the flag !

Enter your choice:
1) Create HUSTCTFer Account
2) Create Admin Account
3) Login
4) Exit
1
here is your token: 8ed436b2c20f007a7b524ea36cb8cefb5797aeed0a922ed6f46276c941205c42

Enter your choice:
1) Create HUSTCTFer Account
2) Create Admin Account
3) Login
4) Exit
87e5088fef1a2272601d30a36cb8cefb5797aeed0a922ed6f46276c941205c42
WTF

Enter your choice:
1) Create HUSTCTFer Account
2) Create Admin Account
3) Login
4) Exit
3
Enter your token >
87e5088fef1a2272601d30a36cb8cefb5797aeed0a922ed6f46276c941205c42
Hello Admin! Here is your FLAG: vmc{P9bvWOOsBocAcJz7mRZ2dUOGPCTDsJlR}
```

15.Crypto-Hill密码

密文

```
>u\x10l9\npI,0\x04^J\x00ib\x03\x0c\x158d\x1f\x08Ixk\nF\x19fz\x14PT\x04\x03>R~
```

39个字符，即我们可以得到13组 3*3 矩阵 乘 3个字节向量的式子

我们又已知开头是 `vmc{}` 末尾一点有空格

最后一位一定是空格，未知的只有4个数，sagemath爆破启动！

```python
from Crypto.Util.number import *
def is_printable(s):
    for char in s:
        if not char == ' ' and not char.isprintable()   :
            return False
    return True
raw = b'>u\x10l9\npI,0\x04^J\x00ib\x03\x0c\x158d\x1f\x08Ixk\nF\x19fz\x14PT\x04\x03>R~'
rawi = [int(x) for x in raw]
print(rawi)
print([ord('v'),ord('m'),ord('c'),ord('{'),ord('}')])
# print(rawi)
c = matrix(Zmod(127), 3, 3 , [118,109,99,123,0,0,125,32,32])
# 118 109 99
# 123 xx  xx
# 125 32  32
p = matrix(Zmod(127), 3, 3 , [62,117,16,108,57,10,62,82,126])
#  62 117 16
# 108  57 10
# 62  82 126
for i in range(0,127):
    print(i)
    for j in range(127):
#         for a in range(127):
#             c[0,1] = a
            # print(c[1,2])
        c[1,1] = j
        c[1,2] = i
        #
        flag = True
        if c.is_invertible() is True:
            K = c.solve_right(p) # 解AK = B
            # K = matrix(Zmod(127), 3, 3 , [1,0,1,7,1,2,2,2,3])
            # print(f'{i},{j},{a}| {K} ')
            decode = ''
            # tmp = matrix(Zmod(127), 3, 1 ,rawi[36:39])
            # A = K.solve_right(tmp)
            for k in range(13):
                tmp = matrix(Zmod(127), 1, 3 ,rawi[k*3:(k+1)*3])
                A = K.solve_left(tmp) # [1*3]
                for l in range(3):
                    char = chr(A[0,l])
                    if not char == ' ' and not char.isprintable() :
                        flag = False
                        break
                    decode += char
                if flag is False:
                    break
```

```
            if flag is True:
                print(f'{i},{j}|{decode}')
```

先测试倒数三位全是空格的情况，但是没有爆破出可打印的flag

```
60
60,56 | vmc{<8e3EDJua&47mUIDe3ouIzWeR01_dK9D}
60,69 | vmc{<Ee3.DJ6a&$7myID(3oeIz2eRV1_`K9m}
60,108 | vmc{<1e3hDJwa&s7mfIDo3o5IzBeRI1_TK9i}
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
78,56 | vmc{N8eDEDhua^47nUI[e3(uI=WeL01mdKhD}
78,69 | vmc{NEeD.Dh6a^$7nyI[(3(eI=2eLV1m`Khm}
78,108 | vmc{N1eDhDhwa^s7nfI[o3(5I=BeLI1mTKhi}
79
```

以上是倒数第二位是空格的情况，看起来不太乐观，应该没有flag

但其实是有的，原来这题的flag不是有语义的一句话了，这害得我纠结了半天...... `vmc{d8euED8uaj47SUIie34uI9Weo01pdKiD}`

Insert Cell Type Format Toolbars Run Validate Stop Kernel

SageMath 10.3 | Trusted   Kernel is idle (halt...)   CPU

```
                        break
                    decode += char
                if flag is False:
                    break
            if flag is True:
                print(f'{i},{j}|{decode}')
```

```
Out[4]: 56
56,60|vmc{<8e3EDJua&47mUIDe3ouIzWeR01_dK9D}
56,78|vmc{N8eDEDhua^47nUI[e3(uI=WeL01mdKhD}
56,94|vmc{^8eEED.ua-47}UI7e3vuI#Weq01AdK/D}
56,100|vmc{d8euED8uaj47SUIie34uI9Weo01pdKiD}
56,112|vmc{p8eVEDLuae47~UINe3/uIeWek010dK^D}
56,117|vmc{u8e~ED*ua.47[UI#e3wuIMWe?01adKdD}
57
58
59
60
61
62
63
64
65
66
67
68
69
69,60|vmc{<Ee3.DJ6a&$7myID(3oeIz2eRV1_`K9m}
69,78|vmc{NEeD.Dh6a^$7nyI[(3(eI=2eLV1m`Khm}
69,94|vmc{^EeE.D.6a-$7}yI7(3veI#2eqV1A`K/m}
69,100|vmc{dEeu.D86aj$7SyIi(34eI92eoV1p`Kim}
69,112|vmc{pEeV.DL6ae$7~yIN(3/eIe2ekV10`K^m}
```