第17题、格密码分析与解决方案

通过构建特定矩阵并应用 `LLL 算法`，恢复密文 `c` 的明文，从而获取最终的标志字符串。

解题脚本

```
# 建议使用SageMath在线运行
# https://cocalc.com/features/sage

from Crypto.Util.number import *
from gmpy2 import invert

h =
311970364425799366998489758816351964614898164597015962390522858760089331210269507769408330245745888541705059882396722340774681935443615888403494901415226651295201132556613291289590900246992553502513664561058230713840616052310750850144077552701142563872837663789636304968423919464707545482008647087312063949024744195318160335652074407874634742273493108112212110123691909346846612115479023950119286716243604630664235633831784083300431161225131883065150778501178458601885263943907964612851841356128492283759401129349655263360496418606534637831065564054627124545266130146585012568948018083420026867791483624987951876329968315410612748989452628410331813540143425873481351220499681815979569479285905651270208308396719282247586008816711527598915797885618862654168623002012746684102245733757428828517277957675441290263051618987121956646671901357370585784409633697165339182406341764320293630983680416055584947772382282259966044517683538433409836256995980804901093405108581243143413703339657071622279140236369465154220363993923085583049720512383384577159975134910449465903227555419552492606439149153015674603047695345700309870206034743317361786333732070962362572427411597266252707397451423195512449103367120235737550650354427690243524900449675
p =
958396606448120961344481821891302529131234571519205540072533929007120447482644938027839069855225984909598017115601687321093037138370623689366302399472755221666720998204960391688686871213784218139146955110040572010513513710257471193690475085340782434134757767894370195789546551282060123540167285748706110004061656181247087025993398981267165890691214538467956637203791517952370602330966682619162963485159850944867534075815673753018593722394868700193403593396672054125765356203130863459200363914531124614764692809438199726465266945790488666626285169156592007636403443360006225301437752255897266876561763146766508532479031209192791719933364280473149045268623071494670752015337498338685674896828186550044711044706449912730457638945023144945899725372140139807478899719273567393929653142032920726449027277418648476280470840599530703214171595308025409850676858589945411985285200257825229131032920095947661866076555686111255840425468035670354014870389179342958912828532723039862073721465350955688340929352513243365563954638294876066618830813521617443475610064698193890094075786932957915036083650599261645906663293860533520556976700786359174781146828149075244841152143143030216821401471735118624215070906105894392159946202260400995019474190360 7
```

```
c =
767158168672362136291238223905378538545444315829681403660668665037917788547250510431680762324989504337886911129872811283399423182492713085621340850466620775416712732614020830060267436728824312319343522189998875569791210615658554329492772575522352284894778864900683043087475949382767603528807809596004213194946687583749277851731545121301799909443549367647188538291294493900118514585149301404415365107375148502163345229200555520367376148986792681776613941685900525114359918025093194374564154918465789098039688273260831196880453048787881692220588699313824407443049610470566611891830120455606865304514498141918370250200060333776115471607369861567714061571248634366226901360555240991881037749649258353302128601187117945261098036478588953880956746052307508140721683848770369771427200654010686745459084737649483182116223928908511586372587312845173805333061611628627699056444060210866689908925857413693914088384801590539118419817748706914585535977225375257595014121709225398146804929559026288825337416874187757876993164395155899845807880799091660701491902131855351054079661782867046531376623992742661604584530403794933640735943298254524984623808363772872836875750554022138997087590785362832339405833456398277842368920260448389099114876068641

mat = [[0, p], [1, h]]
M = Matrix(ZZ, mat)
a, g = M.LLL()[0]

m = (a*c % p*invert(a, g)*invert(a, g)) % g
flag = long_to_bytes(int(m))
print(flag)

# b'vmc{L4ttice_I5_pretty_easy_FOr_U!!!}'
```

第18题、DSA签名分析与解决方案

本题考查共享k攻击。本题实际上是给我们多组签名+消息，让我们进行共享k攻击，以此计算出密钥。

解题脚本

```
import socket  # 导入 socket 模块用于网络通信
import time  # 导入 time 模块用于时间控制
import math  # 导入 math 模块用于数学运算
from gmpy2 import invert  # 从 gmpy2 导入 invert 函数，用于求逆元
import hashlib  # 导入 hashlib 模块用于哈希计算

# 定义服务器的 IP 地址和端口
HOST = '10.12.153.8'
PORT = 31961
loop_times = 7  # 定义请求次数


def get_signatures():
    # 初始化签名和哈希值列表
    sign_s, hash_m = [], []
    for i in range(loop_times):
        print(f"\r获取到第{i + 1}组签名与消息", end='', flush=True)
        time.sleep(1.1 if i > 0 else 0)  # 等待一下，让服务端哈希值更新
        with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:  # 使用 with
语句管理 socket 连接
```

```
            try:
                s.connect((HOST, PORT))  # 连接到指定的主机和端口
                s.recv(1024)  # 接收初始响应
                s.send(b"1\n")  # 发送请求以获取签名和哈希值
                response = s.recv(1024).decode().split(",")  # 接收响应并解析
                sign_r = int(response[0])  # 提取签名 r 值
                sign_s.append(int(response[1]))  # 保存签名 s 值
                hash_m.append(int(response[2]))  # 保存消息哈希值
            except Exception as e:
                print("发生错误:", e)  # 捕获并打印异常
    return sign_r, sign_s, hash_m  # 返回最后一个 r 值和所有签名、哈希值


def compute_keys(sign_r, sign_s, hash_m):
    # 计算签名差值和哈希差值
    delta_s = [sign_s[i + 1] - sign_s[i] for i in range(len(sign_s) - 1)]
    delta_h = [hash_m[i + 1] - hash_m[i] for i in range(len(hash_m) - 1)]

    q = 0  # 初始化 q 值
    for i in range(2, len(delta_s)):
        # 计算 kq1 和 kq2，用于求解 q
        kq1 = abs(delta_s[i - 2] * delta_h[i - 1] -
                  delta_s[i - 1] * delta_h[i - 2])
        kq2 = abs(delta_s[i - 1] * delta_h[i] - delta_s[i] * delta_h[i - 1])
        kq = math.gcd(kq1, kq2)  # 计算 kq 的最大公约数
        q = math.gcd(q, kq) if q > 0 else kq  # 更新 q 值

    # 计算临时密钥 k 和密钥 x
    k = delta_h[0] * invert(delta_s[0], q) % q
    x = (k * sign_s[0] - hash_m[0]) * invert(sign_r, q) % q
    return q, k, x  # 返回计算出的 q, k, x


def send_final_signature(sign_r, sign_s):
    with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:  # 使用 with 语句
管理 socket 连接
        try:
            s.connect((HOST, PORT))  # 连接到指定的主机和端口
            s.recv(1024)  # 接收初始响应
            s.send(b"2\n")  # 发送请求以验证签名
            response = s.recv(1024)  # 接收响应
            signature_part = response.decode().split('"')[1]  # 提取签名部分
            to_check = int(hashlib.md5(signature_part.encode()
                                       ).hexdigest(), 16)  # 计算待检查的哈希值
            sign_s = (to_check + x * sign_r) * invert(k, q) % q  # 计算最终签名
            s.send(f"{sign_r}, {sign_s}\n".encode())  # 发送签名和 r 值
            final_response = s.recv(1024).decode()  # 接收最终响应
            final_response = s.recv(1024).decode().strip()  # 再次接收并去除空格
            print('\n\n', final_response, '\n\n')  # 打印最终响应
        except Exception as e:
            print("发生错误:", e)  # 捕获并打印异常


# 主程序
sign_r, sign_s, hash_m = get_signatures()  # 获取签名和哈希值
```

```
q, k, x = compute_keys(sign_r, sign_s, hash_m)  # 计算公钥和临时密钥
send_final_signature(sign_r, x)  # 发送最终签名

# flag是动态生成的，每个人flag不一样
# flag案例：vmc{x8Xr9WQdXWyTO8k91DwVHXOf8UTeuxh8}
```

第19题、简单RSA分析与解决方案

本题考查 **低加密指数广播攻击** 的基本原理。由于 RSA 加密的模数（n）只能分解为两个质数（p）和（q），当存在大量的不同模数（n）时，有较高概率会出现某些模数中的（p）或（q）是相同的。因此，不同的（n）之间可能存在相同的质因数（p）或（q）。通过计算不同（n）之间的最大公约数 `gcd()`，我们可以确定是否存在相同的质因数：**若最大公约数大于 1**，说明存在共享质因数，可以用来计算私钥。

解题脚本

```
import gmpy2
from Crypto.Util.number import long_to_bytes

n1 =
14311136148050672419989799427104593072360126727996357997737771323416025835284720
89349126877189321764554103129394131734892369539357882212706475370557570994389895
40147109234991903063070178183183908704264314942406845588026059583201065739138761
64986666561830970380943541512973969654875418388029963883263795540100294000652035
42172115303315860688622504771703538069426634062124105878883673906516864062919539
56538045619428269381098921993749954913784151561476934806784750248108281399517479
50220497130345509778576687918051304112321531099375758183212756982987812354817112
83148909728991523450039346686413637517629207741792168759 1
c1 =
82028527858762701578630520027565700904953935736719100055968935747110327407993979
36438752152316778125546684561981338463208326745896180102802370978316384870311169
50338190754875560024018194429229310040489695989832117662154636016719050133572425
72382404037130948843056335230486392141293948239850109995613917817437075533247937
55021757051461973597832982432346639642869313748849063526909019855981543807729556
42048805005272315224483433612028279459365749045116129242813403110800718943560588
40113551367679685843501230973814766333636366071624092209158710830474326736765119
90516075293548074046637736562426611866570077500698539680
n2 =
22460630266363520029013021830190111925329835544313855662313212281658578787408425
12174665494755562527807707085795049156185424418435883159369340172117825508001128
34831669908487929253820626874617080642973718527470264919164796262044750665560674
61737173240576845698707955971525234482303159455512217128800846697492645909822799
70618990114354121810953644167791737499142878562251189585475538501700357799804891
43269302602995480559026689552639233296998646915664631080302093367654605447101368
57719832988428394394769888465359508988507140048564837670939119403564949331948498
51641957273236162014718957226092345958379586698743147208 1
c2 =
19432526669205632933281999861455073126026491878458861462621671904525045303797730
34450466489498912173834941282650003880521498836488242269781052570255584626692704
90565735504558092250192966628136966155769767755573793223539776317012195767372119
86654717188414587448855563253088294339676132929423451927193840172170831630957970
11924504688419811672967799568272499325296405826213046693744936431377898508560042
24551171324980060370556506353359699655523192871955007746659111084949495714380111
60394509085343201107394718958934368493483937533921994429464995605815197936150743
85787551825274837378574513489443809502473030145074128245 3
```

```
n3 =
19992746611686250573671647760828360992790652451036608485720507826226043955859605
40869969945101941186315377018693218771736194124162223125558026025438219026236758
40160915520878349530994788419116847564357398873995973011845208350654076804782227
56923581533037301452594035962041816064218938028684243323353238297033311148235732
89308563408000914987629811120827170574433046017634534323302147564965163386552382
82181122244305154953329701261978196001378194955853343618035769946757763348453879
76082014911523417917110951173363569720887345722135810314736716071201425604128184
78472180418780724887209609702032566281676251061564543657 1
c3 =
16761239931080052583119467254883044613843428027350731469077889755297092262423557
56178094989477473815346679213150816442461153879504956463523631552553019868662367
17281153672828847401685496892176861222883330948256445879373165204696621548898106
17075602455599097628405412912960405528988078488648702010584562028792787672783864
51472232312221879762044595208024091347140241001562321951418951184978052185919070
81362936425089679735826512625444680810410642729401968097016330560647251461137425
96400325295624752267310145921728201937132472951530120140732967263171552147743333
23074684053360885766416233156773357099757155698049450 5239
n4 =
14921019826845713229726484877726270689241984653233268169688581852222590026842979
17223258220140235149709699863575145161378080791911082977825046231481929413284079
33614856891318843239054769903453448003545009998587208356041099552996491888242632
62669855070670628208634166768950607071709626319152810191066883413750048659163843
43644966917011275759876351453487059738908242467899944893030373346873197792564972
84479522961686068753816639555878783445780941432592733542247253000906719487288978
63727697421927341322881916785053396241076618270994707195235790442124786541590007
80759557644177573453034209389536619042112736318910325443
c4 =
12104686397227931738546574591380357622718178963781492437624997870522237479459381
81067581575868578972482431357378402303914528457296806280953630428776146079704221
74244590472216381300364819920309119378931144660615163064563457481337689879535274
29987096596623215114426732278065630014608792807941320949416675074089844010060080
92158122906960489655837758184715561574407723709757153622296005136792948341511450
38628790525226876385682714354934960762302287468634864260623856922705492415365593
84169613474232448181941652396294943544455326954005366751933697950468829610423401
26753065246325355204279902702276410404044661637678793126 2
n5 =
15272117973129901152469934169722584048788617147757632365791658800750926957768612
28688865511691821597407804029451507888531806865496015052744769259836508883181131
89951177642309640443140386537162037178599697037380410080286102601620328522301911
29230669874776941136044240565959807731829332386524847728126521323260017315161990
23821264576757413594986949603339017743020579139490963744906399217308507978724890
50550730679883147067204503923144076327593451793810553391643163929602592034960452
65143504969702593896637060523554936600159946787412996100488023723444073375164457
09384175081574963072765063172883596180243928865233460369 9
c5 =
53229776429337443134531913553517124094304509511821936660366973401404020622024293
19093295156685557666038667144031573087283411323579118551022111465343730249209962
22861666636331003591302316949272206703801068406427074138926340739839088705439896
69503418706826679796151806903840930878021044995465740106900682507071268494541175
87489480539877887273971258748795254086254991230380218491391522968395928667038744
03912296503961242361567822221589253466316231297080976876879308748352796893535008
78046195762371608413862452694337913668004592566130230890467721085888380692696013
60508046641806768766210684891031655538702026049109953 76
```

```
n6 =
1812744556519026684182729975640807766504309577765045203503383158876399525062453022644966557886989911075957797269562388050347713338590762627948876188538298563363915808473858651492177827886555524504691468062951874945880440371556988697391854181208056218117265015250253737581289151904877158855808290120433415843525242051564390421179947284730172278489857267815089751579840509871237908981824512964880347414192712577180050524442708209394494518571868690028900090522078148436768981220940084475894138638768790638124119016286312541910456443273447884954181086209161765227322800923049978389289011505838519885961173440358180222624 9
c6 =
2625551961520404822614319606750443223876109845986978449151552936165052835749320005616623196600993458605095692554096855097708637686395841605169785299974081565211071562756383216779434333914364702548682103868142599355338105838477768408407890186681959208882692900392416470186936290224911113828252994437329629224782258411200582637781334626739579364350628562723437287922536352184401820922080861455065326765600711933303305357435210407951675455959745201471975055113425613549118728398786287459590835602915250133309913998661270053053432537991582471485775363429365803053660902542066262325929134649526894392488264686353472291858
n7 =
2523766099200044178317254092484790194980625547631465412449438304599414874335694094287444088113454266702556095133948156924842370191467273809871492103891965601306929886677835499193984629608954755988579444890262400397172916213716777978383144012424262753424985706102539505702275819219517767458908012015042037643682121772460383844518544357247481274662020578635322104520930707928167035529360268202341598256824534683225118682130283158075025917816673613320980517975056181205405164914760389117622168718301728545048742246053096879743029752976591295097124026498000548152482977850144660945086109878404919849596903574097431364089 9
c7 =
9758510052958978102948228334244869415165545109949510690189969202323542658592737853436921895896043005877526302279472466168422359921314039734929206526881778897067215647448266071512675649702774441944204282736857844920408229618077379117600268298609365142116955969115516459293315364107943215961013155993930752669360414805051578461860895483051303295323035671380094695239371154702653128168720030747483907996025437103350598863926938687075304844605508496789347812093047555407980728165302731410858261186919424953215143824645606262477453433776500347616509302859390460838136875303489356615422369128295995697637647427439868512713
n8 =
1995553317045866298576180445293244939059398751834656296989863129935220154598983189328146014841257851736952665872091948208870799532016610942091663989124130806422194470187000580020476217911968091810685102269251828336174432859229474724188131432984870051191179399762447401628785099941408834486396854371722903732441797259030673889354944523722552001632159198307140854593489701615733279844923645831536169952935050661555634831283944592202511864837286597163973237904940652458940497592273670323270035170147339523625759201155352564443062425771657527959492309827525057292019223762428232823251949010479895328575344465259589932595 7
c8 =
502243375354558457411714303126562751990870001784185290113834797139194553132393873855435220806050740385295311194766024837836737664116691293513210656047517744774817166430994409641435157277402369245001813937285807384929184298415660350764450346587311342662411821678344669613909185140250451164951950024187416956005620129288868857662958389340883969339550478535860727775628305983665715763402129632363213158651604726698853130088976299854782743486064045534664523726113758393323204289217604516039571323813836932300327843626503218010744068578918548546221031683378103495665823438469496171007950556614417755976638315101685533533 9
```

```
n9 =
206088591996349181941306010320885395774152845727778506536074101271322070051922650193090191648964223320291451729967833809593212980593199957744018804662792502041406492547965343910700433421813362000025038799046157360048280089266073727324414206074205843671909786269240204801513142842842718537758420203809950719166922208030635857092576343155625778251886232913176104744091905813464715798513841736078549920481151053586431363939896308795083964090641307769698460933220287356109225464236543229496115875219463931951186304491972522033491811584008406178399253381110625350722054404123148970708698868667644279443930706934261072940 61
c9 =
100501923444106089980689353754292396656707715191873878606053724436070326003292681465138242549472765536869225233858451867371948468028365261388695695813855083154704558670353851422391833988897104561902353515350421765530765105182976655884094754790553165886123978414455013934548403252798044115116159243693410322697012982762250670678041995148967326894381774679006006118019041324862400380718822395728101722426992992138985967494245287823866059141620762138535125496521776883277886570656769371534342800623934071354323631974730145928493436235645486123425630863190953833306152023414974077586555565398320501973690025260504703695991 0
n10 =
208058062851840496324949031820378986567966267771304886550044079702212717881653318822372577086039253148004203548956622174948642269596456250700752392154738617157225417416305533842599777135134341009267366960355846819398419218022476902204510661395532233510549428287027909149906311632951869430111645312355777678670261587643895958941948814424389532944267833218981014632083264656393588332212438601320357031359613141862347244417664192533518614652894747240090121327766736569689187357230712334528198154730944632916608538324773268916393737775076621619157775254964908323246597729263737871829482101594954509289912142056528153308691 0
c10 =
901588485967162255293200538901264608355557200369300441220318500930229190072248187406375925321971148964793772453717421173661754780544684994017831847773239764757765227817207167959108716372580334865271965514294712352538984676415148679080318564492200186995778605026902573402392252129229000029591988698410733858036585245384330203994610513481342765008508698708246744811783368314010882533900522609878773176859749919592970286877545921479814172431906082707504673496510271590855659018837819464864298105570193152298748220192028719030743533832435738266779004010636016620678768943552808013803769508977675662405020601347549475802

n = [n1, n2, n3, n4, n5, n6, n7, n8, n9, n10]
c = [c1, c2, c3, c4, c5, c6, c7, c8, c9, c10]

for i in range(len(n)):
    for j in range(i + 1, len(n)):
        gcd = gmpy2.gcd(n[i], n[j])
        if gcd != 1:
            p = gcd
            q = n[i] // p
            phi = (p - 1) * (q - 1)
            d = gmpy2.invert(65537, phi)
            m = gmpy2.powmod(c[i], d, n[i])
            flag = long_to_bytes(int(m))
            print(flag)
            break

# b'vmc{This_is_really_easy_rsa_HAHAAAA}'
```

第20题、分组密码工作模式分析与解决方案

本题涉及到 **CBC 字节翻转攻击** 的应用。只需对 `user_key` 进行合适的位修改，再用修改后的 `user_key` 解密，即可让明文变成我们想要的 `AdminAdmin!_____`。

解题脚本

```
import socket

HOST = '10.12.153.8'  # 服务器地址
PORT = 32014          # 端口号

# 创建 socket 连接
with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
    try:
        s.connect((HOST, PORT))  # 连接到服务器
        s.recv(1024)  # 接收初始消息

        s.send(b"1\n")  # 发送创建账户请求
        s.recv(1024)  # 接收中间响应

        # 接收token
        response = s.recv(1024).decode()
        start_index = response.find("token:") + len("token: ")
        end_index = response.find("\n", start_index)
        token = response[start_index:end_index].strip()

        # 计算用户密钥
        user_key = bytes.fromhex(token[:32])  # 前32个字节为user_key
        cur = b'HUSTCTFer!_______'
        tar = b'AdminAdmin!_____'
        user_key = bytes([user_key[i] ^ cur[i] ^ tar[i]
                          for i in range(len(cur))])

        s.send(b"3\n")  # 发送登录请求
        s.recv(1024)  # 接收中间响应

        # 发送计算后的用户密钥和密文
        s.send(f'{user_key.hex() + token[32:]} \n'.encode())
        s.recv(1024)  # 接收中间响应

        # 输出最终响应
        print("\n\n", s.recv(1024).decode(), "\n")
        s.send(b"4\n")  # 发送退出请求

    except Exception as e:
        print("发生错误:", e)

# flag是动态生成的，每个人flag不一样
# flag案例：vmc{U7pSxq1bodyGydI6gpc1amTCkYvkBnvF}
```

第21题、古典密码破译分析与解决方案

三阶Hill密码需要通过三组密文和明文来解出加密矩阵。根据题目提供的信息，我们已经知道了两组明文和一个字符，可以通过枚举未知的两个明文字符来填充明文矩阵。接着，利用这两组明文和密文，可以解出加密矩阵。然后，使用该加密矩阵对密文进行解密，得到解密后的明文。如果解密出来的明文都是可打印字符，则输出解密结果。

解题脚本

```
# 建议使用SageMath在线运行
# https://cocalc.com/features/sage

from Crypto.Util.number import *
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]

p = matrix(Zmod(127), 3, 3, [118, 109, 99, 123, 0, 0, 125, 32, 32])
c = matrix(Zmod(127), 3, 3, [62, 117, 16, 108, 57, 10, 62, 82, 126])

for i in range(0, 127):
    for j in range(0, 127):
        p[1, 1] = j
        p[1, 2] = i
        flag = True
        if p.is_invertible():
            K = p.solve_right(c)
            decode = ''
            for k in range(13):
                tmp = matrix(Zmod(127), 1, 3, rawi[k*3:(k+1)*3])
                A = K.solve_left(tmp)
                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}")

'''
56,60|vmc{<8e3EDJua&47mUIDe3ouIzWeRO1_dK9D}
56,78|vmc{N8eDEDhua^47nUI[e3(uI=WeLO1mdKhD}
56,94|vmc{^8eEED.ua-47}UI7e3vuI#WeqO1AdK/D}
56,100|vmc{d8euED8uaj47SUIie34uI9WeoO1pdKiD}
56,112|vmc{p8eVEDLuae47~UINe3/uIeWekO1OdK^D}
56,117|vmc{u8e~ED*ua.47[UI#e3wuIMWe?O1adKdD}
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/eIe2ekV1O`K^m}
69,117|vmc{uEe~.D*6a.$7[yI#(3weIM2e?V1a`Kdm}
108,60|vmc{<le3hDJwa&s7mfIDo3o5IzBeRI1_TK9i}
```

```
108,78|vmc{NleDhDhwa^s7nfI[o3(5I=BeLI1mTKhi}
108,94|vmc{^leEhD.wa-s7}fI7o3v5I#BeqI1ATK/i}
108,100|vmc{dleuhD8wajs7SfIio345I9BeoI1pTKii}
108,112|vmc{pleVhDLwaes7~fINo3/5IeBekI1OTK^i}
108,117|vmc{ule~hD*wa.s7[fI#o3w5IMBe?I1aTKdi}
'''
# 经测试flag是vmc{d8euED8uaj47SUIie34uI9WeoO1pdKiD}
```

第22题、RSA加密算法攻击分析与解决方案

对于这一题，我们有以下表达式：

$$
\frac{N1}{N2} = \left(\frac{P1}{P2}\right)^2 \cdot \frac{Q1}{Q2}
$$

显然可以推导出：

$$
\frac{N1}{N2} < \frac{Q1}{Q2}
$$

因此，我们可以确定 $\frac{Q1}{Q2}$ 的范围为：

$$
\frac{Q1}{Q2} \in \left(\frac{N1}{N2}, 1\right)
$$

这是解题的关键。

接下来我们对 $\displaystyle \frac{N1}{N2}$ 进行 **连分数展开**，并求出其各项的渐近分数。在这些渐近分数中，某个连分数的分母可能就是 $Q1$。我们可以通过验证条件 $N \% Q == 0$ 来确认这个猜测。

解题脚本

```
from Crypto.Util.number import *
import gmpy2
n1 =
27682578737141139764880192910976946263355689816882797515059917479242862799083599
74559495688025824411286755972243585073281202318966258105251128786755330831826802
00223863068204248298988580299861934129226459443594092485681310573773806972362384
80724883073062491532254626363468032145049953168789073328812076794158602028961853
98603437814474965622854155264120739347383071515645247343213004036047156609616514
60872028360367833046405791830823018585298185980323398212378412197741247107897619
12675044056265735587753304064079484844965820681168729776560497921764083742448045
65489111350003506347431844207803653181395755108623174707915569169000143312718738
26360498712282795194667357197687985747763536870496671253841465661077397055535806
93984918816215940308884007192621418304753551998125658993859095063641090798574130
16165125789091691432507613743686901845457752283 3
```

```
c1 =
14360977893873474578201937159000122429359790977572665232657843468076201963407780
01513185719262155073733880588051439335739057642373132887186724102926029405104571
01444829898018570541588169988975461247098027301986902441285450736344861457867632
94634081834588146373913232490890078533918320777534358739106486350300547206365723
04530676703821492341203263383325574296395470147540170438504501906988373462525143
64098515880442413368354527289628602808655040001033615596888611490864699399401137
48174610019620309023214292662384279070127090992947332945432141695583191136521301
94011658561003379034812511447198028533201191835557883912889207505869888531924359
33450967347764978174612516433819899583268104785000266843893589203420218365725116
88796450072700142033952403561408907486022094802237175920044147084170050294965826
25825061867563834372635290747639347412867448894 3
e1 =
13890651822147152152440433003961663329775276553417657086890003923713341985748541
56394231966360683972372962244420832137686304881007179778844153421042392809504247
35129147986053115335928783190377695248250926374734988108972136349625965753649992
14632281035276824604157539672166114224672974757283201751024174908243 1
n2 =
27682578737141139764880192910976946263355689816882797515059917479242862799083599
74559495688025824411286755972243585073281202318966258105251128786755330857625423
27069532905190599761592392055592959651101487344496502099772359531632554948080567
07188551192674128213090005439928085856216617642935948961573449294338310127166077
19526340293984886168621448511568679903290114731475934848106241810912041866130258
54138687826022824631651711290631979614557791936650419028229489630325800540670502
27612838335828201043413949164885293325493829570131849345344856137656453666135670
72497418474911555072082649755876332012721825197057614475031978212119448356354537
11573231669689831760131452678568988654371017999585883427412574574720363114904022
79982286349929050116350394664561659857216725849236910894778018502118673902399095
64648780846215520703476443234269954910908080876 9
c2 =
11293777290569693972360166961981727494638218221438571150393361751316389824613571
82022937091519150076661941059711767123244345269163473411265228552180682428495907
35580106612047309549288472609464038672979328626877704496325060878831879201077660
50673462588812979708792790888354008526054467620780053118019643408427959406056087
37096017099283404789008026966374787714327068306957531839714484448126238246346908
07554230975270071614494119339366694514764673520492644552036327299091646660066882
94056405955940041007137719228484035343153943155100892867641033645111253109951972
00379841352400350557400078439945870534726235315536341351334179786894208513697754
81166508153360006273539337089132374388419093249200700134981536277678916749695860
34872104569344923832761239959420543717354211689339868787331074579605476477152218
06881073208991302345624042572082104703022465991 8
e2 =
13890651822147152152440433003961663329775276553417657086890003923713341985748541
56394231966360683972372962244420832137686304881007179778844153421042392809504247
35129147986053115335928783190377695248250926374734988108972136349625965753649992
14632281035276824604157539672166114224672974757283201751024174908261 9
```

```
def continuedFra(x, y):
    cF = []
    while y:
        cF += [x // y]
        x, y = y, x % y
    return cF
```

```
def Simplify(ctnf):
    numerator = 1
    denominator = 0
    for x in ctnf[::-1]:
        numerator, denominator = x * numerator + denominator, numerator
    return (numerator, denominator)


def getit(c):
    cf = []
    for i in range(1, len(c)):
        cf.append(Simplify(c[:i]))
    return cf


def wienerAttack(e, n):
    cf = continuedFra(e, n)
    for (Q1, Q2) in getit(cf):
        if Q1 == 0:
            continue
        if n1 % Q1 == 0 and Q1 != 1:
            return (Q1, Q2)
    print('Not Found')


Q1, Q2 = wienerAttack(n1, n2)

P1 = gmpy2.iroot(n1//Q1, 2)[0]
P2 = gmpy2.next_prime(P1)

phi1 = P1*(P1-1)*(Q1-1)
phi2 = P2*(P2-1)*(Q2-1)
d1 = gmpy2.invert(e1, phi1)
d2 = gmpy2.invert(e2, phi2)
m1 = long_to_bytes(gmpy2.powmod(c1, d1, n1))
m2 = long_to_bytes(gmpy2.powmod(c2, d2, n2))
print((m1+m2))

# b'vmc{YOu_Ar3_real11ly_sm4rt_in_rrssaa}
```

第23题、LFSR分析与解决方案

该LFSR的反馈函数表达如下：

$a_{i+n} = \sum_{j=1}^{n} c_j a_{i+n-j} (\text{其中} c_j \in \mathbb{F}_2)$

不妨设

$$l=\begin{pmatrix} a_1 & a_2 & \cdots & a_{n-1} & a_n \\ a_2 & a_3 & \cdots & a_n & a_{n+1} \\ a_3 & a_4 & \cdots & a_{n+1} & a_{n+2} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ a_n & a_{n+1} & \cdots & a_{2n-2} & a_{2n-1} \end{pmatrix} \quad r=\begin{pmatrix} a_{n+1} \\ a_{n+2} \\ a_{n+3} \\ \vdots \\ a_{2n} \end{pmatrix}$$

显然有

$$l\cdot\begin{pmatrix} c_1 \\ c_2 \\ c_3 \\ \vdots \\ c_n \end{pmatrix}=r$$

这里可以用`SageMath`的`solve_right`函数求解

解出`cn`的值后，构造递推矩阵`m`

$$m=\begin{pmatrix} 0 & 0 & \cdots & 0 & c_n \\ 1 & 0 & \cdots & 0 & c_{n-1} \\ 0 & 1 & \cdots & 0 & c_{n-2} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & 1 & c_1 \end{pmatrix}$$

根据题意

$$flag=\begin{pmatrix} f_1 & f_2 & \cdots & f_{399} & f_{400} \end{pmatrix}$$

而 $flag$ 先进行了2024轮递推，再开始输出，即

$$flag\cdot m^{2024}=output$$

所以用已知的输出与 m^{2024} 解出 $flag$

解题脚本

```
# 建议使用SageMath在线运行
# https://cocalc.com/features/sage

ct =
'1000010000111000011001000101011101111001010010101110010010000111101001100101010
1100101010011101010000011010011001100111110000100101011011101001111100010101001
1101111111100101011011101000011001111001010100100011111100001111010101010000101
1110101110100011011001101111110001101000100100011000010010100011010111110101100
0111010101011011100111001001110010101110011101010110011101101010010010101111110
0001101111111000111000000100011111100001110101000110011000000101000100001000101
0011010101101001010010111111001011000010000100000110001100111101101110111110001
1100100111101111011000000010111100010010010100101011110110010001100101111001101
0000010001001110111000010011101010011011111011000110000100010000101101001011100
0110011100010110111100011010101101110001011101101101011101011000101110100010000'
n = 400

c = list(map(int, list(ct)))
```

```
l = matrix(GF(2), [c[i:i+n] for i in range(n)])
r = matrix(GF(2), [c[i+n:i+n+1] for i in range(n)])
k = l.solve_right(r)

m = matrix(GF(2),n)
for i in range(n-1):
    m[i+1,i]=1
for i in range(n):
    m[i,-1]=k[i][0]

v = (m^2024).solve_left(matrix(GF(2), c[:n]))
sv = ''.join([str(i) for i in v[0]])
''.join([chr(int(sv[i:i+8],2)) for i in range(0,len(sv),8)])

# \x00mc{Brut!e_@r_so1^e_A_sy3teM_0f_eq&&aT10n3??}mQUMY
# 这个flag感觉有问题感觉
# flag应该是 vmc{Brut!e_@r_so1^e_A_sy3teM_0f_eq&&aT10n3??}
```

第24题、DH密钥交换算法分析与解决方案

这道题表面上考察 **DH 密钥交换**，实则核心在于 **离散对数问题** 的解法。题目给出 (A = 7^a \pmod{p})，我们需要计算私钥 (a) 来解出 flag。由于题目要求给出素数 (p)，可以选择构造一个 (p) 使得 (p - 1) 拥有较多小质因数，这样便于使用 **Pohlig-Hellman 算法** 来求解公钥 (A) 对应的私钥 (a)。幸运的是，在 SageMath 中有现成的函数可以直接使用，因此无需手动实现该算法。

解题脚本

第一步、构造一个至少1024位的素数p

这里构造一个至少有1024位的素数 p, 使得 p−1 的质因数仅有 2、3 和 5，这一步是为了构造适合 `Pohlig-Hellman算法` 的素数

```
from Crypto.Util.number import isPrime
import random


def generate_prime_with_factors_of_2_3_5(bits=1024):
    count = 0
    while True:
        count += 1
        print(f'[{count}]', end='\r')
        # 随机选择a, b, c的值来生成p-1，使得p有足够的位数
        a = random.randint(100, 1000)  # 控制2的指数
        b = random.randint(100, 700)   # 控制3的指数
        c = random.randint(100, 500)   # 控制5的指数
        p_minus_1 = (2 ** a) * (3 ** b) * (5**c)
        p = p_minus_1 + 1
        if p.bit_length() >= bits and isPrime(p):
            print(f"tried {count} times")
            return p, a, b, c


# 生成一个至少1024位的素数p，满足p-1的质因数仅为2, 3, 5
```

```
p, a, b, c = generate_prime_with_factors_of_2_3_5()
print(f"p = {p}")
print(f"p-1 = 2^{a} * 3^{b} * 5^{c}")
print(f"p.bit_length() = {p.bit_length()}")

'''
tried 241 times
p =
12556525073754241807711536615931754502400950993096179978303984794812514738062489
99027009487987474628937865240529484483880639448709701174539063275550960844784506
42207033743330852761605348022833811035156250000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000001
p-1 = 2^230 * 3^406 * 5^242
p.bit_length() = 1436
'''
```

第二步、接收公钥A、B和密文m

```
import socket
from Crypto.Util.number import *
import gmpy2

# 这里设置你申请的环境的IP和端口号
HOST = '10.12.153.8'
PORT = 10615

p =
12556525073754241807711536615931754502400950993096179978303984794812514738062489
99027009487987474628937865240529484483880639448709701174539063275550960844784506
42207033743330852761605348022833811035156250000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000001


with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
    s.connect((HOST, PORT))
    msg = s.recv(1024).decode()
    s.send((str(p)+'\n').encode())
    msg = s.recv(1024).decode()
    A = gmpy2.mpz(msg.split(": ")[1].strip())
    msg = s.recv(1024).decode()
    B = gmpy2.mpz(msg.split("Bob公钥: ")[1].split("\n")[0].strip())
    m = gmpy2.mpz(msg.split("密文: ")[1].strip())
    print(f"A = {A}")
    print(f"B = {B}")
    print(f"m = {m}")
    print(f"p = {p}")

'''
```

```
A =
35566995292103426783116854139656733816636394736145021996353067014268715416427849
97119636897899316672079603039286441019733387216080090193785290022360013541636779
46386844933203453073205489158777257835345405122620355189693067109336813677425279
85327454075384734609011757051993657631982909351625663028722738083583443888404294
18426725910042542472654490597535467836242171880342934121702214759551270466199919
59632722679225589581470575106694
B =
49925512350744218713879721079683637714871362332761802392862463536891163599194107
25041440739723408220491150036255059188021684456127922797856578063653007508543762
03825291528731417788571076202982865612927233051355026003868177599592955887988064
90554207828558713801142086760733346230128991624250118700261502021710639609077831
43952078876819868496068228454956523392683249724797267889518106109240459463770234
19710372938035244494467456994 2
m =
52990511364810189197093624614458461902671222817048954410825030335197572649896744
74699287159936351973504655280082134126049545446421123229320910169359396644073270
69321404497425823323811343729108388418224967949337788356272012533097878402642053
78054670131223214553093759306391365609138384655565867794933825539203261425421458
12919114873668101371323487405154778048135565800969853779939098912288208872372530
11458755426767833584560979924277
p =
12556525073754241807711536615931754502400950993096179978303984794812514738062489
99027009487987474628937865240529484483880639448709701174539063275550960844784506
42207033743330852761605348022833811035156250000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000001
'''
```

第三步、用SageMath计算私钥a和flag

```
# 建议使用SageMath在线运行
# https://cocalc.com/features/sage

from Crypto.Util.number import long_to_bytes
from gmpy2 import invert

A =
35566995292103426783116854139656733816636394736145021996353067014268715416427849
97119636897899316672079603039286441019733387216080090193785290022360013541636779
46386844933203453073205489158777257835345405122620355189693067109336813677425279
85327454075384734609011757051993657631982909351625663028722738083583443888404294
18426725910042542472654490597535467836242171880342934121702214759551270466199919
59632722679225589581470575106694
B =
49925512350744218713879721079683637714871362332761802392862463536891163599194107
25041440739723408220491150036255059188021684456127922797856578063653007508543762
03825291528731417788571076202982865612927233051355026003868177599592955887988064
90554207828558713801142086760733346230128991624250118700261502021710639609077831
43952078876819868496068228454956523392683249724797267889518106109240459463770234
19710372938035244494467456994 2
```

```
m =
52990511364810189197093624614458461902671222817048954410825030335197572649896744
74699287159936351973504655280082134126049545446421123229320910169359396644073270
69321404497425823323811343729108388418224967949337788356272012533097878402642053
78054670131223214553093759306391365609138384655565867794933825539203261425421458
12919114873668101371323487405154778048135565800969853779939098912288208872372530
11458755426767833584560979924277
p =
12556525073754241807711536615931754502400950993096179978303984794812514738062489
99027009487987474628937865240529484483880639448709701174539063275550960844784506
42207033743330852761605348022833811035156250000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000001

R = GF(p)
a = R(A).log(7)

superkey = pow(B, a, p)
flag = m*invert(superkey, p) % p
print(long_to_bytes(flag))

# flag是动态生成的，每个人flag不一样
# flag案例：vmc{ccWkmguSPf2FN6XdChbSqByJEneFdsQr}
```

⭐ Mevinagrise ⭐

> **Status**: Always Learning
> **Motto**: Carpe Diem

"Turning code into magic, one line at a time."
