ECC

考点:

- ECC概念
- SageMath应用

ECC相关概念可以上网查查看，阿贝尔群下运算，具体概念这里不放了。

$$
\begin{gathered}
\text{我们设：} r\text{是加密方的生成随机数} \\
k\text{是私钥、}K\text{是公钥}(K = k * G) \\
\text{加密：} C_1 = M + r * K = M + r * k * G \\
\text{同时告诉解密方：} C_2 = r * G \\
\text{解密：} M = C_1 - r * k * G = C_1 - k * C_2
\end{gathered}
$$

SageMath自带DLP问题求解的函数，直接用就行，要注意的是，加密的时候信息一般要编码到曲线上面，但是这题并没有这样做，就导致了C1，C2都不是在曲线上的点，但这个不要紧，照着题目逆向求出来就行了

```
#sagemath
#part1:求私钥
q=1139075593950729137191297
a=930515656721155210883162
b=631258792856205568553568

G = (641322496020493855620384, 437819621961768591577606)
K = (781988559490437792081406, 76709224526706154630278)
E = EllipticCurve(GF(q),[0,0,0,a,b])
G = E.point(G)
K = E.point(K)
print(G.discrete_log(K))
#12515237792257199894
#part2:解密
from Crypto.Util.number import *
def add(P,Q):
    if P[0] != Q[0] and P[1] != Q[1]:
        t = ((Q[1]-P[1]) * inverse(Q[0]-P[0],q)) %q
    else:
        t = ((3*P[0]*P[0]+a) * inverse(2*P[1],q))%q

    x3 = t*t - P[0] - Q[0]
    y3 = t*(P[0] - x3) - P[1]
    return (x3%q, y3%q)

def mul(t, A, B=0):
    if not t: return B
    return mul(t//2, add(A,A), B if not t&1 else add(B,A) if B else A)


q=1139075593950729137191297
a=930515656721155210883162
b=631258792856205568553568

G = (641322496020493855620384, 437819621961768591577606)
```

```
K = (781988559490437792081406, 76709224526706154630278)
```

LLL-FirstBlood

直接规约就可以得到结果

详细的原理这里看

仙人指路1

怕误人子弟，这里就不详细地写出更多概念，原理了。

LLL算法的核心是施密特约化，输入一组向量基，得到一组约化基。

使得我们可以通过一定的构造，去规约出某个向量(一组数字)，那么在这道题中，因为A是正交矩阵，经过LLL算法之后就被“约掉”了，所以我们可以直接得到题设的结果。

exp:

```python
from Crypto.Util.number import *
C=
[[1528140902799730745476264672501768332416990282355490479242339131918301176698899635154781328839496210200676497333428,
2081687444435007467807250373278513114045272585243815458840083487459795021302180077490134099644993120009567147202772,
3080873409460299046339495750746632185307246572817534784703936044874106809413620470006445984962733721029566440253675,
3491734341995174183626991907292607070252197520631412767989879432598743851171175369180080355977574296558734415823458],
[2359409535809048127331244699867147546817134802610067329431135227991488324148374065940238308147500809599395748756798,
3191196199160821446351036460385791985682645040446022512790815348810555748825420237291839170774872264097466183208742,
46653465301553864572423453942842861983473362814515306708181113876767736288089400119492317775648206643242839430899283,
5369350746042850276067380638571565496087948799720968959426256192923852197959381101839484196445995828389461004495917],
[1641407111066265429602929560264443103285908072677065498760570514577412905392260182334706635555256537745902283191251,
2190536173399177167068153351271988931232272884028569669242062395087922275021628334797729266560930040116807133977244,
3127556759140845426132305699421707182108351516931881411928719802847628408656887897596425133523782526561471050447359,
3707239956529200159380870618471703921011276020439315706352183576289925263316580408968092016782483770373121972835410],
[9883814543195849013523934427451407019514807606993414569626142656857168165339,
13190422499129347541373922929251088892868361241120937213742340947017395215646,
18832738552342488056498211782604832513006649329982003661701684946590064734701,
22323329751908690611034666068697427811613727429398087082295754189068333861152]]
C = Matrix(ZZ,C).LLL()
flag = b''
for i in list(C[0]):
    flag +=(long_to_bytes(-i))
print(flag)
#b'0xGame{8e4d5924dc4cd78f11c1eeb99e991ab3}'
```

LLL-SecondBlood

先推公式:

$$\text{题设：} A*m+noise=c(mod p)$$
$$\text{展开：} A*m+noise=c+k*p$$
$$\text{构造：} A*m-c+k*p=-noise$$

可以发现左边大部分是已知参数，右边是较小的未知质数

那么我们可以构造这样一个矩阵:

$$[x, k_1, k_2, k_3, k_4, 1]\begin{bmatrix} -A_1 & -A_2 & -A_3 & -A_4 & 1 & 0 \\ P_1 & 0 & 0 & 0 & 0 & 0 \\ 0 & P_2 & 0 & 0 & 0 & 0 \\ 0 & 0 & P_3 & 0 & 0 & 0 \\ 0 & 0 & 0 & P_4 & 0 & 0 \\ C_1 & C_2 & C_3 & C_4 & 0 & 2^{341} \end{bmatrix}$$

$$= [noise_1, noise_2, noise_3, noise_4, X, 2^{341}]$$

直接对右边这个构造的矩阵进行规约就好，得到的结果就是下面的矩阵，详细原理在上边。

exp:

```python
from Crypto.Util.number import *
q = 93424266017836508610201195685656564047152360599030090419771497782441539304359080246966668872698904795584736223553468162369727677365777373321732137220122253
mask = [62371284452369929205772256448586626775759511264678888587824613340579700694689258338442311166474068339991426597513746202802132907361145760890693963312267 47, 63680313892139538894175452567501692337259752291974468038850291597677014794455768607045615932009074823726908511521267823911264625475245266319344089810708 41, 51064734609827911885782853974206421376303472892528520450440211979886070827772318398397301696821585078220784124498279766633852820219161208374081925063414 43, 63180908429503312280333495175428101235963168503536374215872648864138771426126861777960230493049086964133862189925111127527886407324108455896798200030476 67]
c_ = [38235396647200290275869331524784927804385950044534892518441338309471653428393938788319148793346602506214228773330223211171203985284305197941096241862044 92, 17216596457502248199532449954605896911206726497325607684352146081678612467901362172193492346047241480399106565734366633793750481450454435272677903798164 25, 66863352007934483964895050238005931191610846880100938613881032425914652332370401449154714897383577491733133358147592080467739594985544118945567052385788 96, 49786058637998107649913028185198601088935625337119226626722033471341578240293931848392641821387734151199691818975059575537256034508589910930534433894406 6]

c = [i for i in c_]
mask.append(1)
mask.append(0)
tmp = [[0 for i in range(len(c)+2)] for _ in range(len(c))]
tmp.append(mask)

for i in range(len(c)):
    tmp[i][i]=q
c.append(0)
c.append(pow(2,341))
tmp.append(c)
```

```
tmp =matrix(ZZ,tmp)
tmp = tmp.LLL()
print(long_to_bytes(-tmp[0][4]))
#b'0xGame{19255b5c7b19c790e28d87c8a8bb1d33}'
```

以上是正解

下面是邪道速通:

Coppersmith

因为未知量的位数都比较小，直接考虑使用多元Coppersmith，这里解释一下多元Coppersmith的思想和作用:

我们想要在有限域中解方程，可以通过展开和一定的方式换到整数域上，把问题变成简单的解方程问题（利用牛顿迭代法），然后再利用LLL算法去对构造的数学式子进行求解。

总而言之就是，实现有限域求根的算法（前提是这个根必须要比模数小很多）。因为求根的时候用到了LLL算法去规约基向量，得到的结果也是一组基（性质更好的），这种问题就被称之为SVP（最短向量）问题。

同理的还有CVP,HNP这些，在将来的格密码学习中会经常打交道，这里就不误人子弟了。

```
import itertools
from Crypto.Util.number import *

def small_roots(f, bounds, m=1, d=None):
    if not d:
        d = f.degree()

    R = f.base_ring()
    N = R.cardinality()

    f /= f.coefficients().pop(0)
    f = f.change_ring(ZZ)

    G = Sequence([], f.parent())
    for i in range(m+1):
        base = N^(m-i) * f^i
        for shifts in itertools.product(range(d), repeat=f.nvariables()):
            g = base * prod(map(power, f.variables(), shifts))
            G.append(g)

    B, monomials = G.coefficient_matrix()
    monomials = vector(monomials)

    factors = [monomial(*bounds) for monomial in monomials]
    for i, factor in enumerate(factors):
        B.rescale_col(i, factor)

    B = B.dense_matrix().LLL()

    B = B.change_ring(QQ)
    for i, factor in enumerate(factors):
        B.rescale_col(i, 1/factor)

    H = Sequence([], f.parent().change_ring(QQ))
    for h in filter(None, B*monomials):
        H.append(h)
```

```
        I = H.ideal()
        if I.dimension() == -1:
            H.pop()
        elif I.dimension() == 0:
            roots = []
            for root in I.variety(ring=ZZ):
                root = tuple(R(root[var]) for var in f.variables())
                roots.append(root)
            return roots
    return []
q = 9342426601783650861020119568565656404715236059903009041977149778244153930435908024696666887269890479558473622355346816236972767736577737332173213722012253
mask = [623712844523699292057722564485866267757595112646788885878246133405797006946892583384423111647406833999142659751374620280213290736114576089069396331226747, 6368031389213953889417545256750169233725975229197446803885029159767701479445576860704561593200907482372690851152126782391126462547524526631934408981070841, 5106473460982791188578285397420642137630347289252852045044021197988607082777231839839730169682158507822078412449827976663385282021916120837408192506341443, 6318090842950331228033349517542810123596316850353637421587264886413877142612686177796023049304908696413386218992511112752788640732410845589679820003047667]
c_ = [3823539664720029027586933152478492780438595004453489251844133830947165342839393878831914879334660250621422877333022321117120398528430519794109624186204492, 1721659645750224819953244995460589691120672649732560768435214608167861246790136217219349234604724148039910656573436663379375048145045443527267790379816425, 668633520079344839648950502380059311916108468801009386138810324259146523323704014491547148973835774917331333581475920804677395949854411894556705238578896, 497860586379981076499130281851986010889356253371192266267220334713415782402939318483926418213877341511996918189750595755372560345085899109305344338944066]


A = mask[0]
c = c_[0]
PR.<x,noise> = PolynomialRing(Zmod(q))
f = A*x - c + noise
roots = small_roots(f,(2^320,2^50),2,3)
print(roots)
# [(404417766109752774365993311026206252937822359426120081323087457724287886115277329019989616964477, 585427539127961)]

```

Matrix

考研真题：大一新师傅的练习册上就有解法的。

$$\text{已知}A^{secret}=C$$

$$\text{通过相似矩阵得：}A=P^{-1}*B*P$$

$$\text{那么问题就变成：}A^{secret}=(P^{-1}*B*P)^{secret}=C$$

$$\text{对中间的式子展开得到：}P^{-1}*B^{secret}*P$$

$$\text{其中}B\text{是对角矩阵，问题就变成了求对角元素上面的离散对数问题}$$

$$\text{那么之后参照上周的解法就可以了。}$$

exp:

```

#sage
A=[[12143520799533590286, 1517884368, 12143520745929978443, 796545089340,
12143514553710344843, 28963398496032, 12143436449354407235, 158437186324560,
12143329129091084963, 144214939188320, 12143459416553205779, 11289521392968],
[12143520799533124067, 1552775781, 12143520745442171123, 796372987410, 12143514596803995443,
28617862048776, 12143437786643111987, 155426784993480, 12143333265382547123,
140792203111560, 12143460985399172467, 10983300063372],[12143520799533026603, 1545759072,
12143520746151921286, 781222462020, 12143514741528175043, 27856210942560,
12143440210529480891, 150563969013744, 12143339455702534403, 135941365971840,
12143463119774571623, 10579745342712],[4857408319806885466, 2428704161425648657,
12143520747462241175, 758851601758, 12143514933292307603, 7286139389566980165,
9714738936567334300, 144947557513044, 12143346444338047691, 130561054163540,
4857352974113333366, 2428714303424782417],[12143520799533339320, 1476842796,
12143520749060275613, 733281428880, 12143515144091549812, 25896324662208,
12143446129977471347, 139126289668080, 12143353609086952433, 125093278125816,
12143467808884068695, 9705993135696],[3469577371288079926, 5204366058378782250,
12143520750775862343, 706665985740, 12143515359139397843, 24876891455539,
12143449149385190675, 5204499435641729607, 1734628523990131469, 119757210113970,
12143470097256549947, 9282407958928],[10986995009101166671, 1734788687033207505,
12143520752514668698, 680173911560, 12143515570582515443, 23883386182656,
12143452072344092516, 10408859957710764174, 8673790006740000925, 4047954924507284041,
12143472277719610437, 8879790035168],[12143520799534210329, 8095680534365818753,
12143520754224346525, 6071761054204856029, 12143515774342357443, 22931775530664,
12143454859049102627, 122586336122081, 12143373761302849103, 109840689548590,
8095634066844843878, 8500892291801],[2428704159899526175, 7286112481016467893,
12143520755876491019, 629765964828, 12143515968446948123, 9714838668887734012,
4857345013259425502, 117630592711632, 12143379764863568374, 105318302849760,
2428659620509049335, 7286120625945355053],[7286112479717322389, 7286112480971640825,
12143520757456628435, 606320684970, 12143516152115449139, 4857429497934652454,
4857347490735050126, 112978994964264, 12143385390297217523, 101086824360217,
7286069740980100293, 7286120294834973633],[7727695054246476847, 1202487728,
12143520758958480293, 584144077140, 12143516325240923843, 20377952745696,
12143462294760579275, 108622249048560, 12143390651947217363, 97133513961120,
12143479741445599772, 8831658996900830432],[12143520799535388887, 1161628182,
12143520760380594623, 563225247585, 12143516488091679443, 19626876325056,
12143464472820678035, 104545135017180, 12143395570399006523, 93441517429260,
12143481309754543787, 7218375794633]]
```

```
enc=[[1128584799051509503, 7585413350741918021, 1165825451243641266, 477577914899276103,
2941386515764607825, 11283325421744133699, 4096971712575507616, 8118672870538606033,
2377937081025778041, 6576171711896495163, 6152554374963853172, 5022013484610428974],
[8354008012616001452, 7787447107046065118, 9504997911333967278, 1082773427768571094,
6015520658629219637, 11244285744740006951, 4493944053220750368, 3504246247470690014,
1738582001618280397, 2330057776906622572, 3043456814665571080, 2981613454022714952],
[2508674373714509177, 3544963739532775937, 7952732753025175616, 11161786730565526285,
3397123486689639675, 6454135592624912854, 6613201018024296927, 9748485344986779929,
1819761609989340766, 1259944825407465767, 1596049024644778041, 7769939905324967788],
[4200851163596876950, 11960539098651202761, 3303721151143544462, 2532304102428121556,
11083895221097319129, 1171933471304558017, 1549099593543874478, 6088238862927163233,
6459553630361959801, 947358195425767572, 2090533922210134578, 9023030120605201052],
[2271102089902208138, 1614812525306266829, 1546249462332047661, 3168333397191737100,
7678980468150522028, 3128939172985153696, 1146041044751755224, 1187017322706514061 7,
8351303466095252790, 694704483676649448, 7944218023016968278, 583421745603756386],
[10309472503110333289, 11005982619907188 22, 10235859400888405310, 910925705831020921,
10771855884237562064, 9970830255165655653, 11678899608458971536, 4368822164222204233,
3104861419162339779, 4540709628196554222, 7851809145727500968, 12086896840826708824],
[10973051751637593366, 5039073157846327641, 4855314857834773443, 4416954195828423951,
8243966437000815560, 8250554263390748131, 8093181066366682440, 1145520354143718292,
2947290130236370 45, 10115389386419597159, 2767140395261835843, 6724257139233017485],
[6878768250003631244, 10834164422364241529, 6946589221005878489, 5397342184795218 33,
2691724062063066048, 3989403041446358401, 815244541494093987, 11168528286389981272,
2021358468726921955, 1123433019094267521, 524639025046508882, 5720273332497702547],
[6688451244183880831, 10892730373179989558, 6987453292894341174, 5572212176769878684,
11332149024403380575, 3944612864568504791, 6768594304071589280, 10526434024562201079,
10241323610053039912, 11204735584108657 53, 3061536351482262 48, 3606666063074222104],
[7556871914690327290, 11353594909211427742, 747771112781361153, 1245068803956910299,
2831489557155431404, 1800035620948876551, 1050411779595241927, 5665981688041778089,
2028968510484240787, 4386552235402890530, 10334391443650474796, 3883841302951550608],
[4485787817401669404, 1845011915009529 34, 3690661645276970957, 6263309802498749034,
6484490370652685031, 9743108369653588026, 3045941510087387269, 5870433915209047275,
4679598273992216016, 11839352681285251516, 4957980185504231911, 7925596893607015470],
[10004497128784667 19, 7022601702937838844, 10958499074827911 66, 11989051568709522226,
6768031250066783733, 185945517026191241, 4280928696740160411, 5633542561098902406,
10176177574499086410, 5782837249861240943, 7406530879613861823, 1971858224839520916]]
p=12143520799543738643
A = matrix(GF(p), A)
enc = matrix(GF(p), enc)
B,P = A.eigenmatrix_right()
P_inv = P.inverse()
assert P*B*P_inv == A
B_=((P_inv*enc*P)[0])[0]
b=(B[0])[0]
x=discrete_log(mod(B_,p),mod(b,p))
print(x)
#6208835615336459559
#md5后交一下flag就行
```

Overflow:

签名用的是ElGamal算法，这里这样放出这种题，理由如下：

- 了解一下ElGamal算法的特点
- 适应代码审计（有些时候题目代码很长，很容易让人感到害怕）

签名的时候没有对消息做校验，那么导致了我们的签名可以溢出被模数消掉，直接看WP吧，摆烂了。

核心的考点其实不难，只要细心就好了。

exp:

```python

from pwn import *
from Crypto.Util.number import *
io = remote('0.0.0.0',10002)
io.recvuntil(b'key:\n')
pub = eval(io.recvline())
io.recvuntil(b'>')
msg = long_to_bytes(bytes_to_long(b'0xGame')+pub[0]-1)
io.sendline(msg)
io.recvuntil(b'r=')
r = int(io.recvline())
io.recvuntil(b's=')
s = int(io.recvline())
io.recvuntil(b'flag.\n')
io.sendline(str(r).encode())
io.sendline(str(s).encode())
io.interactive()
io.close()
#b'0xGame{24b6edfdc07d71311774ed15248f434e}'
```