Normal ECC:

考点:

- 椭圆曲线
- DLP
- SmartAttack

脚本一把梭就可以了，因为题目生成的模数也比较难找，抽象，直接就抄了别人的参数了，根据题目的Hint很快就就可以搜出这个方法来，本周的送分题。[拓展阅读]

exp:

```
from hashlib import md5

def MD5(m):return md5(str(m).encode()).hexdigest()
p=11093300438765357787693823122068501933326829181518693650897090781749379503427651954028543076247583697669597230934286751428880673539155279232304301123931419
a=490963434153515882934487973185142842357175523008183292296815140698999054658777820556076794490414610737654365807063916602037816955706321036900113929329671
b=766854265479378498843649908673923944291517028734612164588409622294833827916530221344006007914196067967852601634802502955833597704271238261119799500231646
G=(4045939664332192284605924284905750194599514115248885617006435833400516258314135019849306107002566248677228498859069119557284134574413164612914441502516162,2847794627838984866808853730797794758944159239755903652092146137932959816137006954045318821531984715562135134681256836794735388745354065994745661832926404)
K=(9857925495630886472871072848615069766635115253576843197716242339068269151167072057478472997523547299286363591371734837904400286993818976404285783613138603,9981865329938877904579306200429599690480093951555010258809210740458120586507638100468722807717390033784290215217185921690103757911870933497240578867679716)
C1=(4349662787973529188741615503085571493571434812105745603868205005885464592782536198234863020839759214118594741734453731681116610298272107088387481605173124,1083570830235542579872939299345133716277325300044056633361161063323492929415974331661530877816894769756738610922343005600648987690000111563456782267433770)
C2=(5193866657417498376737132473732737330916570240569047910293144235752602489388092937375844109374780050061859498276712695321973801207620914447727053101524592,684299154840371832195648774293174908478389728255128448106858267664482339440737099810868633906297465450436417091302739473407943955874648486647511119341978)

E = EllipticCurve(GF(p), [0, 0, 0, a, b])
```

```
P = E([K[0],K[1]])
Q = E([G[0],G[1]])
c1 = E([C1[0],C1[1]])
c2 = E([C2[0],C2[1]])

def SmartAttack(P,Q,p):
    E = P.curve()
    Eqp = EllipticCurve(Qp(p, 2), [ ZZ(t) + randint(0,p)*p for t in
E.a_invariants() ])

    P_Qps = Eqp.lift_x(ZZ(P.xy()[0]), all=True)
    for P_Qp in P_Qps:
        if GF(p)(P_Qp.xy()[1]) == P.xy()[1]:
            break

    Q_Qps = Eqp.lift_x(ZZ(Q.xy()[0]), all=True)
    for Q_Qp in Q_Qps:
        if GF(p)(Q_Qp.xy()[1]) == Q.xy()[1]:
            break

    p_times_P = p*P_Qp
    p_times_Q = p*Q_Qp

    x_P,y_P = p_times_P.xy()
    x_Q,y_Q = p_times_Q.xy()

    phi_P = -(x_P/y_P)
    phi_Q = -(x_Q/y_Q)
    k = phi_Q/phi_P
    return ZZ(k)

r = SmartAttack(P, Q, p)
print(MD5((c1-k*c2).xy()[0]))
#裹上flag就可以了
```

Drange Leak:

考点

- Coppersmith的利用
- 阅读论文的能力
- 基本数学推导

这题其实比较简单的，看着论文长篇大论的蛮哈人，但其实我们知道一篇论文的顺序是：

- 引论(这篇论文的背景，解决了什么问题)
- 引理(用了哪些关键的定理、这些定理的内容是什么)
- 推导过程和结论(基本上追求速通的话就看这部分就可以)
- 实验数据
- 总结，引用文章……

大致上都是这样，所以这里直接快进到中间，这里可以发现：

$$d = M * d_1 + d_0$$
$$ed = 1(mod phi)$$
$$\text{展开：} ed = 1 + k * phi$$
$$\text{再展：} e * (M * d_1 + d_0) - 1 - k * phi = 0$$
$$\text{展：} e * (M * d_1 + d_0) - 1 - k * [N - (q + p - 1)] = 0$$
$$\text{到这一步就是题目的关键了}$$

这里引入一个Coppersmith定理：用来求解有限域的小根问题。(给定一个模数下的数学式子，求解其中的较小未知数。)

在论文中可以看到这个Coppersmith实现的一定过程。具体核心思想在上周的WP中已经给出，所以这里直接拿来用：只要找对了关系式，找到较小的根，直接small_roots就完了。

那么在这里，我们先粗略地估算一下位数(不需要太准确，我们只要知道较小的未知数是否存在就行)：

$$\text{式子：} e * (M * d_1 + d_0) - 1 - k * [N - (q + p - 1)] = 0$$
$$e : 2048\text{位，} N : 2048\text{位，} k : 1024\text{位左右可能}$$
$$\text{利用已知信息构造有限域式子：}$$
$$e * d_0 - 1 - k * [N - (q + p - 1)] = 0(mod M * e)$$
$$\text{令} q + p - 1 = z$$
$$\text{得} e * d_0 - 1 - k * (N - z) = 0(mod M * e)$$
$$d_0 : 70\text{位，} k < 1024\text{位，} z\text{：} 512\text{位}$$
$$\text{之后利用} small\_roots \text{函数去解未知数就可以}$$

其实论文看不看都行，经验熟练了直接拿手推就好。

这里有一点小坑的地方在于，能够常规small_roots函数能求解的位数比较低，我卡的参数也比较极限，有的师傅构造出来了，但是解不出来，原因可能有几个：

- bounds:这个求解的界卡太准不一定好，卡个差不多就行了，太大、过小都不一定求得出来
- m:格基规约的维数?总之越大求解的小根就越准确，但是算法的速度也会越慢。
- d:多项式的次数。

经验之谈：找对办法的情况下，估算好位数之后就可以开始梭哈了。

exp:

```
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 []

n = 208906498070980985909883675045898841041698824611378227009154211388252430824010732856516883963651191770483143783423356300037588019184717700672567810324414087556002224431364428028346730337267502627925917137294543593210857762459015070248433510321813926211607093212357303771058589280384295615634512128315553620847998683968166209005308216499271436750425087541453002357071644805958671591830207304882445238903774942005519827326734204636104200464054962221438632937211278471963156990114804078592456028787591927633580277126664904368773099586949303008811541442620127863886781700418276034851035962587221518670336183461803142217 57
e = 184956246910043293454947397681391196548692947810014395032283756756567802055338320885519256034579133759652366662485601108245228164057845936224893920635696939803077112732620461785221551500579180046700626381332295114413788570674418088146639796563291185761743897732236720785703460565695687695861363338785851844959007696104856825237130353388151803552262966270238562186626778516912004008700866618253186627181723226972395971483044000502012019574910476543472229466934577849506941191289570109387084571946381643706899693959148665894680774474111605319951947404139509280858249853171143935919616982156677499378800239849671718671 49
```

```
c = 726874831148943099664958333429634223912097653596989015164052828126403734591956324774419834084762267133216554027392707903728846350158689567565239779121113003379756232085817724965762748556814734336898185229543535897087537560152501328825971723210625365604172417463730791502152490452684902597606217435136043108950589825667303506002087189255602042975484908444842839430741430137669998320326207204195183571307550940229130128133765856743707560914491390552662575937446501868409223681817428277721533697988649505361910595183528208748720159398116447712007386425964497894019235178127060970259576736273132095939765716138468145932 3
leak=1366079098401465558063611568736188922407158688855743696295229140368073931645429303081666091047350029458813882163620079412132988883075796922728657002116081264961050571135067568577934631972509091611731164227232466620946955867161069722984281649269939959485289412410372423671900421208861337 17
PR.<x,k,z> = PolynomialRing(Zmod(e*leak))
f = e*x - k*n + k*z - 1
roots = small_roots(f,(2^100,2^1024,2^1024),3,3)
print(roots)
#从这里得到的z = (p+q-1)，后续直接解方程就可以了。
```

LLL-ThirdBlood

考点：

- 格基的理解
- DSA算法
- 签名伪造

详细的文章在这里 [一类基于各种DSA的HNP问题求解]

其实仔细搜索一下hint的内容，网上应该也有大量的文章，通过大量地搜索发现：阴差阳错之下甚至发现和20年的学长出过的赛题撞了[demo]。（一周内共有十位师傅，包括一位校内新生做出来了，跪了）

part1:拖取数据

k的位数这里没有卡太死(甚至导致了非预期解)，直接拖四组就行，求稳多拖几组的结果都一样的。

```
#part1:拖取数据
from pwn import *
from hashlib import sha1
from Crypto.Util.number import *

context(os='linux', arch='amd64', log_level='debug')
h = bytes_to_long(sha1(b'test').digest())
s = []
r = []

io = remote('0.0.0.0',10002)

def Sign(target):
    target.sendafter(b'>',b'S')
    target.sendafter(b'>',b'test')
    target.recvuntil(b's = ')
    s_ = int(target.recvline())
```

```
    target.recvuntil(b'r = ')
    r_ = int(target.recvline())
    s.append(s_)
    r.append(r_)

io.recvuntil(b'q=')
q=int(io.recvline())
io.recvuntil(b'g=')
g=int(io.recvline())
io.recvuntil(b'y=')
y=int(io.recvline())

for i in range(10):
    Sign(io)
io.close()

print(f'q={q}')
print(f'h={h}')
print(f'r={r}')
print(f's={s}')
```

part2:求解私钥

仔细想想这一步，其实就是上周LLLSecond的拓展，原理是一个差不多的(甚至构造也能一样?)

以下是论文的构造解：

```
from Crypto.Util.number import inverse
q=
h=
r=
s=
#填入以上数据

A=[]
B=[]
M=[]

t = 120
#填入k的大概位数，相当于一个上界，比想要求解的向量大一点就行
for i in range(len(r)):
    tmp = [0 for i in range(len(r)+2)]
    tmp[i] = q
    A.append(mod(r[i]*s[0]*inverse_mod(r[0]*s[i],q),q))
    B.append(mod((r[0]*h-r[i]*h)*inverse_mod(r[0]*s[i],q),q))
    M.append(tmp)

A.extend([1,0])
B.extend([0,2^(t)])
M.append(A)
M.append(B)
M = matrix(ZZ, M)
#构造矩阵
T = M.LLL()

```

```
s0 = s[0]
s1 = s[1]
r0 =r[0]
k = T[0][0]
x = inverse(r0, q) * (s0 * k - h) % q
print(x)
#获得私钥
```

以下是学弟的非预期解：

```
q =
A =
B =
M = matrix(ZZ,22,22)
for i in range(20):
    M[i,i]=q
    M[20,i]=A[i]
    M[20,i]=B[i]
M[20,20]=1
M[21,21]=pow(2,160)
res=M.LLL()
assert pow(2,160)=res[-1][-1]
print(abs(res[-1][-1]))
```

嗯，确确实实的非预期了，而且和题目说的一样，和上周的预期构造是相同，我怎么能如此的粗心大意？

part3:伪造签名

```
from Crypto.Util.number import *
from random import getrandbits,randint
from hashlib import sha1
pri_key = 2746225058150767948б

class DSA:
    def __init__(self):
        self.q=
        self.p=
        self.g=
        self.y=
        self.x = pri_key
    def sign(self,m):
        H = bytes_to_long(sha1(m).digest())
        k = getrandbits(128)
        r = pow(self.g,k,self.p)%self.q
        s = (inverse(k,self.q)*(H+r*self.x))%self.q
        return (s,r)

    def verify(self,m,s_,r_):
        H = bytes_to_long(sha1(m).digest())
        u1 = (inverse(s_,self.q)*H)%self.q
        u2 = (inverse(s_,self.q)*r_)%self.q
        r = (pow(self.g,u1,self.p)*pow(self.y,u2,self.p))%self.p%self.q
        if r == r_:
```

```python
                return True
            else:
                return False

Test = DSA()
s,r = Test.sign(b'admin')
assert Test.verify(b'admin',s,r) == True
print(s,r)
```

Orac1e

第一周就学过的，CBC分组模式大致是什么流程。

详细文章这里看(看我看我)，下面的代码就不用看了，有点啰嗦的。

介绍视频这里看视频在这里。

这里简单阐述一下：因为CBC分组模式存在的原因，我们可以把解密的处理分为以下流程

- 密文分组
- 密文组解密->得到明文组
- 合并明文组
- 去填充
- 检验是否合法

我们将中间解密的这块算法视作一个“黑盒”，

其中因为CBC分组解密流程的原因：密文->黑盒->上组密文->明文，

又因为必须检验填充是否合法的原因：

当且仅当去填充位上的数字==去填充位数，解密才能成功

那么按照第一周的CBC思想，通过已知明文去猜解密钥(中间经过黑盒的向量)，这道题就结束了。

```python
from pwn import *
from base64 import b64encode,b64decode
#context(log_level='debug')
io = remote('0.0.0.0',10002)

size = 16

def Orac1e(payload,index):
    data = b64encode(payload)
    io.sendafter(b'>',data)
    tmp = io.recvline()
    if tmp == b' Data update\n':
        print(index)
        return 1
    else:
        return 0

def Oracle(BIV,BC):
    D = []
```

```python
    for index in range(15,-1,-1):
        I = [0 for _ in range(index)]
        for i in range(256):
            if D == []:
                CIV = bytes(I)+bytes([i])
            else:
                CIV = bytes(I)+bytes([i])+xor(D,16-index)
            assert len(CIV) == 16
            payload = CIV+BC
            if Orac1e(payload,index):
                D.insert(0,(i^(16-index)))
                break
    return xor(D,BIV)

def Attack(c):
    Block_count = len(c)//16
    print(f'Block_count={Block_count}')
    iv = c[0:16]
    m = b''
    for i in range(1,Block_count):
        m += Oracle(c[size*(i-1):size*i],c[size*i:size*(i+1)])
        #print(m)
    return m

io.recvline()
c = io.recvline()[:-1]
c = b64decode(c)
print(Attack(c))
```

写在最后：

希望喜欢研究密码学和数学的小伙伴们能坚持学下去，别**被很长的题目**和**看不懂的数学理论**"劝退"，也不要为了简简单单的"上分"这个理由，随随便便抄了代码解了这题就算过了(经验的教训)，不管啥方向都好，多复现多复现。

去年0xGame开始的时候，出题人还是全方向的零基础，甚至去问学长什么是异或？甚至四周密码爆零(偷偷写了一题)。而人生第一次用编程解题还是在那年的misc方向中......(zys师傅确实给了挺大的帮助)。所以喜欢就做吧，0基础也能学得挺好，不是很强才能开始，而是开始了才能变强。

事实上CTF赛事中的密码学可能对未来工作、就业的帮助不是那么大(甚至在今天有人劝我考公)，但密码学却是信息安全的基础建设之一，而且也能帮助自己快速入门算法、参与竞赛。在一些逆向破解的活动中，密码学中学到的技巧也确确实实地给了我挺大的帮助(一眼定算法，手撕密钥......)，总之学以致用，干就完了。