CTF 学习笔记三——Pohlig-Hellman 算法

算法由 Pohlig 和 Hellman 发明，这是一种为解决离散对数问题而提出的攻击方法，早在 1978 年就被提出。主要思想是对阶数进行分解，比如整数域中$y = g^x (modp)$里的 x 以及椭圆曲线离散对数问题中 Gk=Q 的 G 的阶 n，这样就把对应的离散对数问题转移到了每个因子条件下对应的离散对数，然后可以利用中国剩余定理进行求解。

假设需要求解的式子为　Q=lP，其中 P 为选取的一个基点，　l 为选定的随机数，相当于要求解的私钥。

首先求得 P 的阶 n，即可使得 nP 不存在的最小正整数，将 n 进行分解，设$n = {p_1}^{e_1} {p_2}^{e_2} \ldots\ldots {p_r}^{e_r}$，

将因子取出，计算$l_i \equiv l(mod{p_i}^{e_i})$，$i \in [1,r]$，即：

$$l \equiv l_1(mod{p_1}^{e_1})$$
$$l \equiv l_2(mod{p_2}^{e_2})$$
$$\ldots\ldots$$
$$l \equiv l_r(mod{p_r}^{e_r})$$

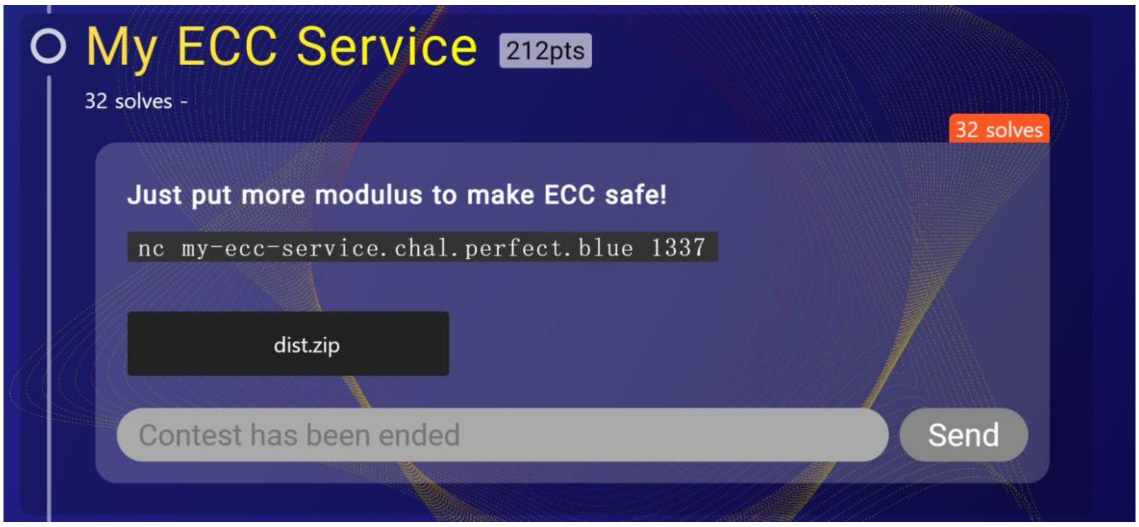
其中$l_1, l_2 \ldots\ldots l_r$是一些确定的数值，计算规则会在后文提出。于是就可以用中国剩余定理求解出 l 的数值。于是有$l = k \times {p_i}^{e_i} + l_i$，$i \in [1,r]$，$k \in Z$

对于$l_1, l_2 \ldots\ldots l_r$，其计算规则如下：

先定义点$P_i = \frac{n}{{p_i}^{e_i}} P$，$Q_i = \frac{n}{{p_i}^{e_i}} Q$，$i \in [1,r]$，那么就有${p_i}^{e_i} P_i = nP = O\infty$，因此$P_i$的阶为${p_i}^{e_i}$。对$Q = l \times P$两边同时乘以$\frac{n}{{p_i}^{e_i}}$，那么有$Q_i = l \times P_i$，将 l 代入，就有$Q_i = (k \times {p_i}^{e_i} + l_i) \times P_i = l_i P_i$

由于P_i的阶为${p_i}^{e_i}$，比 n 小，所以更适合运算。对所有的P_i，Q_i求离散对数，即可得到l_i对应的值，那么最后 l 的值也可以计算出来。

例题：Challenges - pbctf (perfect.blue)


My ECC Service 212pts

32 solves -

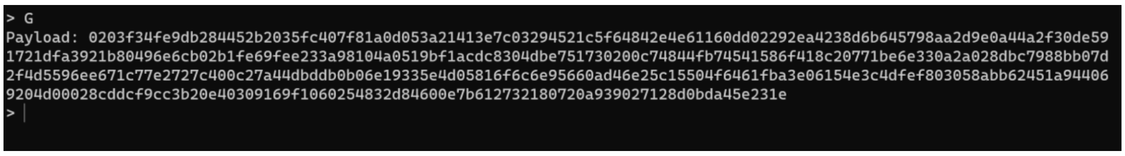
32 solves

Just put more modulus to make ECC safe!

```
nc my-ecc-service.chal.perfect.blue 1337
```

对于这道题目而言，根据代码可以轻松求出椭圆曲线的方程为$y^2 = x^3 - 3x + 7$，基点为(2,3)，p 的值一共给了 16 组。通过计算发现，给的 16 组 p 对应的阶 n 都满足 n=p-1，是一个合数，因而可以使用 Pohlig-Hellman 算法进行求解。当然，这边的 16 组 p 不用全部计算，选择阶 n 分解之后，n 的因数较小的进行计算即可。这边我们不妨取 16 组 p 当中的第 2 个 p 进行运算，即 p=7434077280325317871715778622377。

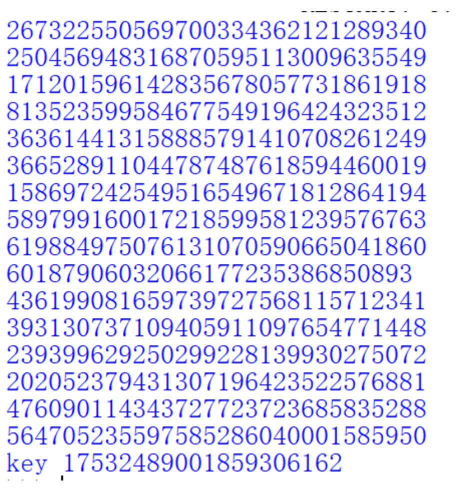
首先获取一下 payload 的值：



```
> G
Payload: 0203f34fe9db284452b2035fc407f81a0d053a21413e7c03294521c5f64842e4e61160dd02292ea4238d6b645798aa2d9e0a44a2f30de59
1721dfa3921b80496e6cb02b1fe69fee233a98104a0519bf1acdc8304dbe751730200c74844fb74541586f418c20771be6e330a2a028dbc7988bb07d
2f4d5596ee671c77e2727c400c27a44dbddb0b06e19335e4d05816f6c6e95660ad46e25c15504f6461fba3e06154e3c4dfef803058abb62451a94406
9204d00028cddcf9cc3b20e40309169f1060254832d84600e7b612732180720a939027128d0bda45e231e
> |
```

然后使用以下 python 代码对 payload 进行解码：

```python
payload="0203f34fe9db284452b2035fc407f81a0d053a21413e7c03294521c5f64842e4e61160dd02292ea4238d6b645798aa2d9e0a44a2f30de591721dfa3921b80496e6cb02b1fe69fee233a98104a0519bf1acdc8304dbe751730200c74844fb74541586f418c20771be6e330a2a028dbc7988bb07d2f4d5596ee671c77e2727c400c27a44dbddb0b06e19335e4d05816f6c6e95660ad46e25c15504f6461fba3e06154e3c4dfef803058abb62451a944069204d00028cddcf9cc3b20e40309169f1060254832d84600e7b612732180720a939027128d0bda45e231e"
keysec=payload[4:20]
num = payload[20:]
for i in range(0,len(num),26):
    print(int(num[i:i+26],16))
print("key",int(keysec,16))
```



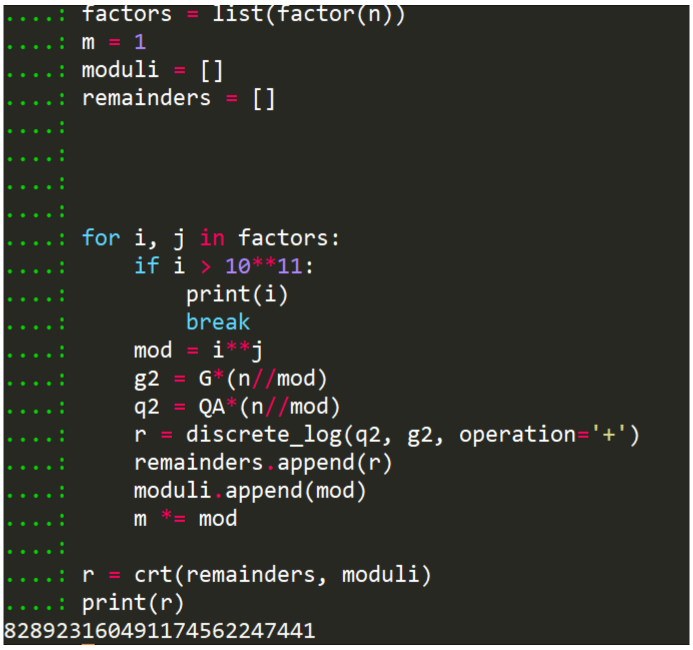
```
267322550569700334362121289340
250456948316870595113009635549
171201596142835678057731861918
813523599584677549196424323512
363614413158885791410708261249
366528911044787487618594460019
158697242549516549671812864194
589799160017218599581239576763
619884975076131070590665041860
60187906032066177235386850893
436199081659739727568115712341
393130737109405911097654771448
239399629250299228139930275072
202052379431307196423522576881
476090114343727723723685835288
564705235597585286040001585950
key 17532489001859306162
```

得到了 16 组基点 G(2,3)经过乘以 nonce 的操作后对应的横坐标的值，由于我们的 p 取得是第 2 个，因而此处 16 个横坐标的值也只需要取第 2 个即可。

再经过 Pohlig-Hellman 算法，即可求得 nonce 的值，sage 代码如下：

```
p = 743407728032531787171577862237
a = -3
b = 7
gx = 2
gy = 3
px = 250456948316870595113009635549
E = EllipticCurve(GF(p), [a, b])
G = E(gx, gy)
n = E.order()
py = E.lift_x(px)[1]
QA = E(px, py)
factors = list(factor(n))
m = 1
moduli = []
remainders = []
for i, j in factors:
    if i > 10**11:
        print(i)
        break
    mod = i**j
    g2 = G*(n//mod)
    q2 = QA*(n//mod)
    r = discrete_log(q2, g2, operation='+')
    remainders.append(r)
    moduli.append(mod)
    m *= mod

r = crt(remainders, moduli)
```

```python
print(r)
```



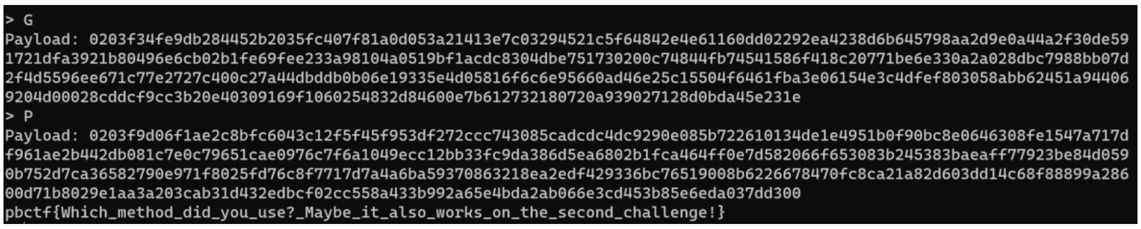
```
....: factors = list(factor(n))
....: m = 1
....: moduli = []
....: remainders = []
....:
....:
....:
....:
....: for i, j in factors:
....:     if i > 10**11:
....:         print(i)
....:         break
....:     mod = i**j
....:     g2 = G*(n//mod)
....:     q2 = QA*(n//mod)
....:     r = discrete_log(q2, g2, operation='+')
....:     remainders.append(r)
....:     moduli.append(mod)
....:     m *= mod
....:
....: r = crt(remainders, moduli)
....: print(r)
828923160491174562247441
```

再将得到的 nonce 值，带入到下一轮 sha256 计算，即可得到最后的答案，

python 代码如下：

```python
from Crypto.Util.number import inverse
from hashlib import sha256
import os
class ECPoint:
    def __init__(self, point, mod):
        self.x = point[0]
        self.y = point[1]
        self.mod = mod

    def inf(self):
        return ECPoint((0, 0), self.mod)

    def _is_inf(self):
        return self.x == 0 and self.y == 0

    def __eq__(self, other):
        assert self.mod == other.mod
        return self.x == other.x and self.y == other.y

    def __add__(self, other):
        assert self.mod == other.mod
        P, Q = self, other
        if P._is_inf() and Q._is_inf():
            return self.inf()
        elif P._is_inf():
            return Q
        elif Q._is_inf():
            return P

        if P == Q:
            lam = (3 * P.x**2 - 3) * inverse(2 * P.y, self.mod) % self.mod
        elif P.x == Q.x:
            return self.inf()
        else:
            lam = (Q.y - P.y) * inverse(Q.x - P.x, self.mod) % self.mod

        x = (lam**2 - P.x - Q.x) % self.mod
        y = (lam * (P.x - x) - P.y) % self.mod

        return ECPoint((x, y), self.mod)

    def __rmul__(self, other: int):
        base, ret = self, self.inf()
        while other > 0:
            if other & 1:
                ret = ret + base
            other >>= 1
            base = base + base
        return ret
BASE_POINT = (2, 3)
MODS = [
        942340315817634793955564145941,
        743407728032531787171577862237,
        738544131228408810877899501401,
        1259364878519558726929217176601,
        1008010020840510185943345843979,
        1091751292145929362278703826843,
        793740294757729426365912710779,
        1150777367270126864511515229247,
        763179896322263629934390422709,
        636578605918784948191113787037,
        1026431693628541431558922383259,
        1017462942498845298161486906117,
        734931478529974629373494426499,
        934230128883556339260430101091,
        960517171253207745834255748181,
        746815232752302425332893938923,
]
```

```python
k = 82892316049117456224744l
k = k.to_bytes(10,"big")
k = sha256(k + b'wow').digest()[:10]
key = sha256(k).digest()[:8]
nonce = int.from_bytes(k, 'big')
ret = b""
for mod in MODS:
    p = ECPoint(BASE_POINT, mod)
    x = (nonce * p).x
    ret += x.to_bytes(13, "big")
ret = b"\x02\x03" + key + ret
print(ret.hex())
#0203f9d06f1ae2c8bfc6043c12f5f45f953df272ccc743085cadcdc4dc9290e085b722610134de1e49
51b0f90bc8e0646308fe1547a717df961ae2b442db081c7e0c79651cae0976c7f6a1049ecc12bb33fc9
da386d5ea6802b1fca464ff0e7d582066f653083b245383baeaff77923be84d0590b752d7ca36582790
e971f8025fd76c8f7717d7a4a6ba59370863218ea2edf429336bc76519008b6226678470fc8ca21a82d
603dd14c68f88899a28600d71b8029e1aa3a203cab31d432edbcf02cc558a433b992a65e4bda2ab066e
3cd453b85e6eda037dd300
```



```
> G
Payload: 0203f34fe9db284452b2035fc407f81a0d053a21413e7c03294521c5f64842e4e61160dd02292ea4238d6b645798aa2d9e0a44a2f30de59
1721dfa3921b80496e6cb02b1fe69fee233a98104a0519bf1acdc8304dbe751730200c74844fb74541586f418c20771be6e330a2a028dbc7988bb07d
2f4d5596ee671c77e2727c400c27a44dbddb0b06e19335e4d05816f6c6e95660ad46e25c15504f6461fba3e06154e3c4dfef803058abb62451a94406
9204d00028cddcf9cc3b20e40309169f1060254832d84600e7b612732180720a939027128d0bda45e231e
> P
Payload: 0203f9d06f1ae2c8bfc6043c12f5f45f953df272ccc743085cadcdc4dc9290e085b722610134de1e4951b0f90bc8e0646308fe1547a717d
f961ae2b442db081c7e0c79651cae0976c7f6a1049ecc12bb33fc9da386d5ea6802b1fca464ff0e7d582066f653083b245383baeaff77923be84d059
0b752d7ca36582790e971f8025fd76c8f7717d7a4a6ba59370863218ea2edf429336bc76519008b6226678470fc8ca21a82d603dd14c68f88899a286
00d71b8029e1aa3a203cab31d432edbcf02cc558a433b992a65e4bda2ab066e3cd453b85e6eda037dd300
pbctf{Which_method_did_you_use?_Maybe_it_also_works_on_the_second_challenge!}
```

得到 flag 的值：

pbctf{Which_method_did_you_use?_Maybe_it_also_works_on_the_second_challenge!}