Crypto Lab 3

1. BabyDLP

1.1. 题目分析

题目给出了一个根据**ElGamal加密算法**进行加密的脚本，给出的p，c，g的量级都在150余位。给出了flag的格式，但只对{}内的内容进行加密。

从量级来看，这道题即便是使用**BSGS**算法也有着很高的复杂度，故尝试采用**pohlig-hellman算法**来求解这道题，这个算法将一个大的离散对数问题分解成多个小的离散对数问题，在针对群阶因子不是很大的情况下有着优异的时间复杂度，是做这个题目的最佳手段。

1.2. 思路

此题要求解出g^x = c (mod p)中的x。

p是一个质数，所以modp的循环群的阶是p-1，先试着对p-1尝试做质因数分解：

```
sage: p = 2287423676558251281834658094770866774518877828888410121975136169939214
....: 99894585107737978246109443216862577834268294746592989575105135789786204953
....: 92070614563
sage: factor(p-1)
2 * 17 * 2509983517 * 2544123481 * 2569813369 * 2894510363 * 2958235517 * 346815
4477 * 4159802197 * 4166439437 * 79654994629813981763985900545160276768439280673
152129871668427706005929871047
```

可以看出p-1有多个因数，除了最后一个77位的数，其他几个因数都比较小，使用pohlig-hellman算法能有较好的时间复杂度。想要避免过于繁杂的运算，我们可以试着抛弃最后一个数，只用前面的数来实现算法。不可否认的是，这么做有一定的风险性，有可能得不到正确的结果。

题目中已经给出了g，不过这个g不是严格意义上的生成元，因为它只能生成这个群的一个子群，因此它的阶不是n-1，需要先找到实际的阶来简化运算。

利用子群的阶能被原来的群整除的性质，遍历 `p-1` 的所有质因子并检查 `g` 的幂取模后是否为 1 来找到真正的阶，然后用真正的阶进行质因数分解，对每一个质因子（除了最后一个）都使用Sagemath中的discrete_log函数来计算离散对数，得到系列的子结果。

最后用一次中国剩余定理来合并这些结果，得到最终的x，这个x值就是我们需要的离散对数，加上格式就能组合出Flag了。

1.3. 实现

将以上思路转换成sagemath的脚本：

```
p =
228742367655825128183465809477086677451887782888841012197513616993921499894585107737978246109443216862577834268294746592989575105135789786204953920706145 63
c =
400694870688129810359308484164498632493037771343698029167037852456466299951531369348988534378049063111531418159343533120971270985782583634834572399867536 1
g =
129929668910865560580436178601069527365988163425860141494833722029008573794411877221939979761487959915268445811495481234845192044400526761747855457863202 97
from Crypto.Util.number import *
def reduce_order(g, p, factors, order):
    for fac in factors:
        if pow(g, (p-1) // fac, p) == 1:
            order //= fac
    return order

order = p-1
dlogs=[]
factors,exponents = zip(*factor(order))
factors,exponents = zip(*factor(reduce_order(g, p, factors, order)))
primes = [factors[i] ^ exponents[i] for i in range(len(factors))][:-1]
for base in primes:
    t = order//int(base)
    ci=pow(c,t,p)
    gi=pow(g,t,p)
    dlogs += [discrete_log(ci,gi)]

x = crt(dlogs,primes)
print(b'ZJUCTF{' + long_to_bytes(x) + b'}')
```

上面的代码定义了reduce_order函数，用给定的质因数列表来减小生成元 `g` 的阶。它遍历每个质因数，检查 `g` 的 `(p-1) // fac` 次幂是否等于 1（模 `p`），如果是，就通过该质因数减小阶。

主要的计算部分对每个质因数（除了最后一个）计算 `c` 和 `g` 的 `t` 次幂（模 `p`），其中 `t` 是 `order` 除以质因数的值。然后，使用 `discrete_log` 函数计算这些幂的离散对数，并将结果添加到 `dlogs` 列表中。

使用 `crt` 函数将所有的离散对数结果合并成一个整数 `x`，这个 `x` 就是满足 `g^x ≡ c (mod p)` 的解。将 `x` 转换为字节串，并附加到字符串 `'ZJUCTF{'` 和 `'}'` 之间，以形成最终的Flag。

这个代码运行大约需要一分钟时间，在运行完成后可以直接输出正确的flag，即：

```

x = crt(dlogs,primes)
print(b'ZJUCTF{' + long_to_bytes(x) + b'}')
```

```
b'ZJUCTF{P0hlig_He1lman_sm00th_d1p}'
```

到校巴上验证通过：