利用生日悖论求解离散对数

生日悖论

离散对数$y = g^x \bmod p(p为素数)$

若已知y，g，p求x就十分困难，最多需要穷举p次。

生日悖论是指，23人的群体中，两个或更多人拥有相同生日的概率大于0.5.

其含义是，看起来概率很小发生的事情，事实上它发生的概率却很大。

用生日悖论攻击离散对数的基本原理：

Step 2：随机生成一个 1～p-1 之间的整数 a，计算$A = g^a \bmod p$，并将 a 和 A 填入 Tab_A;

Step 3：随机生成一个 1～p-1 之间的整数 b，计算$B = yg^b \bmod p$，并将 b 和 B 填入 Tab_B;

Step 4：重复 Step 2 和 Step 3，直到$n = \lfloor\sqrt{p}\rfloor$次；

Step 5：在表 Tab_A 和 Tab_B 中找重。如果找到 Tab_A 中的某个 A 与 Tab_B 中的某个 B 相等，则记下相对应的 a 和 b；如果没有找到，则回到 Step1；

根据上述步骤，如果出现了$A = B$，即$g^a \bmod p = [(g^x \bmod p)g^b]\bmod p$
由模数乘法的性质,$[(g^x \bmod p)g^b]\bmod p = g^{b+x} \bmod p$
p是素数，在循环群Z_p^*中，如果g是Z_p^*的发生元，g的阶数就是Z_p^*的基数，也即$\Phi(p) = p-1$
上述等式就可以推出$a = b + x(\bmod p-1)$
$x = a - b(\bmod p-1)$

python代码

```
import random

def attack(y, g, p):
    while True:
        A = Create_Table_A(g=g, p=p)
        B = Create_Table_B(y=y, g=g, p=p)
        for i in A.keys():
            if i in B:#如果B的keys里面有i
                a, b = A[i], B[i]
```

```
                x = (a-b) % (p-1)
                print('x =', x)
                return None
        print('失败一次')



def Create_Table_A(g, p):
    A = dict()
    for counter in range(0, int(p**0.5)):
        a = random.randint(0, p-1)
        ga = (g**a) % p
        A[ga] = a#键是ga,值是a,方便查询
    return A #用字典是因为字典是底层是用哈希表实现的，查找速度非常快。



def Create_Table_B(y, g, p):
    B = dict()
    for counter in range(0, int(p**0.5)):
        b = random.randint(0, p-1)
        gb = (g**b) % p
        ygb = (y * gb) % p
        B[ygb] = b
    return B


if __name__ == '__main__':
    y = 5125495
    g = 3
    p = 5767169

    attack(y, g, p)
```

Z257*下测试

3是$\mathbb{Z}_{257}^*$的生成元，取g=3

得到$178 = 3^{150} \mod 257$

```
if __name__ == '__main__':
    y = 178
    g = 3
    p = 257
    attack(y, g, p)

程序运行结果为:

失败一次
x = 150
```

用了两次就破解出来。

Z40961*下测试

3是 $\mathbb{Z}_{40961}^{*}$ 的生成元，仍然取g=3

提前算出 $35788 = 3^{20000} \mod 40961$

```
if __name__ == '__main__':
    y = 35788
    g = 3
    p = 40961
    attack(y, g, p)

程序运行结果为:

失败一次
失败一次
x = 20000
```

三次成功

速度也很快，几乎是一秒之内出结果

在Z104857601*下测试

3是 $\mathbb{Z}_{104857601}^{*}$ 的一个生成元，提前算出

$$66132073 = 3^{12345678} \mod 104857601$$

```
if __name__ == '__main__':
    y = 66132073
    g = 3
    p = 104857601
    attack(y, g, p)
```

根本算不出来。根据加进去的进度条显示，20秒才能算出一个离散对数，而总共要算大概70000次，还要进行比对。这辈子都算不完

换Z5767169*

3是 $\mathbb{Z}_{5767169}^{*}$ 的生成元，提前算出

$$5125495 = 3^{1234567} \mod 5767169$$

```python
if __name__ == '__main__':
    y = 5125495
    g = 3
    p = 5767169
    attack(y, g, p)



结果:
x = 1234567
```

22:45-23:35

计算用时50min