$EzLCG$

[题目考点]

```
对Truncated LCG的格攻击(给出a,b,m)
```

[flag]

```
npuctf{7ruc4t3d-L(G-4nd-LLL-4r3-1nt3r3st1ng}
```

[题目分析]

Truncated LCG可以表示如下:

$x_i = 2^{\beta \cdot size(m)} y_i + z_i$，$\beta$为discarded bits的比例因子，使用的随机数流仅为$y_i s$，在给出部分连续y_i和(a, b, m)的情况下，我们能有效恢复出z_i，从而预测接下来的随机数流.

首先讨论一类求解模等式组的问题，可以表示为

$$\sum_{j=1}^{k} a_{ij} x_j = c_i (mod\ M), i \in \{1, \dots, k\}$$

如果此时我们对系数矩阵A进行格基约化，即AL=LLL(A)，则

$$\sum_{j=1}^{k} a'_{ij} x_j = c'_i (mod\ M)$$
$$C' = A.solve\_left(AL) \cdot C$$

因为AL为约简基，所以也同时有效减小$k_i (Mk_i + c_i = \sum_{j=1}^{k} a_{ij} x_j)$.（约化后可视作$k \to k_{min}$）

令$delta_X = [x_1 - x_0, x_2 - x_1, \dots, x_n - x_{n-1}]^T$

$delta\_Y = [2^{\beta \cdot size(m)}(y_1 - y_0), 2^{\beta \cdot size(m)}(y_2 - y_1), \dots, 2^{\beta \cdot size(m)}(y_n - y_{n-1})]^T$

$delta\_Z = [z_1 - z_0, z_2 - z_1, \dots, z_n - z_{n-1}]$

$\because x_{i+1} - x_i = a^i (x_1 - x_0) \quad (mod\ m)$

$\therefore$ 构造矩阵A如下

$$\begin{bmatrix} m & 0 & 0 & \dots & 0 \\ a & -1 & 0 & \dots & 0 \\ a^2 & 0 & -1 & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots \\ a^n & 0 & 0 & \dots & -1 \end{bmatrix}$$

$A \cdot delta\_X = 0 (mod\ m), AL = LLL(A)$

$AL \cdot delta\_X = 0 (mod\ m), AL \cdot delta\_X = m \cdot [k_0, \dots, k_{n-1}]^T$

$AL \cdot (delta\_Y + delta\_Z) = m \cdot [k_0, \dots, k_{n-1}]^T$

则此时满足：$k \to k_{min}$，因为delta_Z未知，我们只能利用delta_Y进行估值，可以做个粗略估计

AL中每个元大小近似取作$det(AL)^{\frac{1}{n}} = m^{\frac{1}{n}}$，而$size(delta_Z[i]) \leq \beta \cdot size(m)$，因此

$nm^{\frac{1}{n}} 2^{\beta \cdot size(m)} < m \Rightarrow$ 在m足够大时，n可忽略不计（一般取10即可），即$\beta < \frac{n-1}{n}$

在上述条件满足时，可视作$AL \cdot delta_Z < |m|$（但只是大致估计）

因此$k_i = round((AL \cdot delta_Y)_i / m)$，求得$k_i$后，$delta\_Z = AL.solve\_right(mk - AL \cdot delta\_Y)$，delta_X获知，即可推出种子破解整个truncated LCG.

本题破解prng后，即可知AES-CBC加密的key和iv，解密得到flag

[exp]

```
from Crypto.Cipher import AES
from Crypto.Util.number import *


def lcg(seed, a, b, m):
    x = seed % m
    while True:
        x = (a * x + b) % m
        yield x


def get_key():
    key = eval(open("key", "r").read().strip())
    return key


def get_data():
    with open("old","r") as f:
        leak_data = [int(line.strip(), 10) for line in f]
    return leak_data


def decrypt(key, leak_data):
    a, b, m = key['a'], key['b'], key['m']
    A = Matrix(ZZ, 10, 10)
    A[0, 0] = m
    for i in range(1, 10):
        A[i, 0] = a^i
        A[i, i] = -1
    AL = A.LLL()
    leak_data = [leak_data[i] << 64 for i in range(20)]
    delta_Y = vector([leak_data[i + 1] - leak_data[i] for i in range(10)])
    W1 = AL * delta_Y
    W2 = vector([round(RR(w) / m) * m - w for w in W1])
    delta_Z = AL.solve_right(W2)
    delta_X = delta_Y + delta_Z
    x0 = (inverse(a - 1, m) * (delta_X[0] - b)) % m
    predict_iter = lcg(x0, a, b, m)
    for i in range(20):
        key1 = next(predict_iter)
    key2 = next(predict_iter)
    key1 >>= 64
    key3 = (key1 << 64) + (key2 >> 64)
    key3 = long_to_bytes(key3).ljust(16, b'\x00')
    iv = long_to_bytes(next(predict_iter)).ljust(16, b'\x00')
    cipher = AES.new(key3, AES.MODE_CBC, iv)
    ct = open("ct", "rb").read()
    pt = cipher.decrypt(ct)
```

```python
    return pt


def main():
    key = get_key()
    leak_data = get_data()
    flag = decrypt(key, leak_data)
    print(flag)


if __name__ == "__main__":
    main()
```