对称密码

概念了解

对称密码又叫单钥密码体制，使用单一密钥进行加解密，并且加解密都是统一算法，优点是处理速度快，适合大量数据加密，但是也有安全缺陷， 对多个用户时每个用户都要分发密钥，密钥管理难度成指数级上升。 流密码和块密码都是对称密码 流密码:LCG,LFSR,PRNG 块密码：AES，DES

LCG

LCG又叫线性同余生成器，是一种产生伪随机数的方法。

$$X_{n+1} = (a * X_n + b) \bmod m$$

Xn为代表第n个生成的随机数，X0被称为种子值。这里还定义了三个整数：a乘数，b增量，m模数，是产生器设定的常数。 m：是随机序列的模数，必须一个大于0的正整数。一般是比较大的素数或者是2的幂，以便提供较长的周期长度。 a：乘数，必须是一个与m互素的正整数 b：增量，必须与m互素的正整数

公式一：由Xn+1反推Xn

$$X_n = ((X_{n+1} - b) * a^{-1}) \bmod m\text{，这里}a^{-1}\text{是模逆元}$$

模逆元：若有ab和1对n同余，则称a和b关于n互为模倒数，也叫模逆元

$$b \equiv \frac{1}{a} \pmod{n} \quad \text{或} \quad b \equiv a^{-1} \pmod{n}$$

公式二：求a

$$\begin{cases} X_{n+1} & = & (a*X_n+b) & \bmod & m \\ X_n & = & (a*X_{n-1}) & \bmod & m \end{cases} \Rightarrow a=((X_{n+1}-X_n)*(X_n-X_{n-1})^{-1})\bmod m$$

公式三：求b

$$b=(X_{n+1}-a*X_n)\bmod m$$

公式四：求m

$$t_n=X_{n+1}-X_n, t_n=(a*X_n+b)-(a*X_{n-1}+b)=a(X_n-X_{n-1})=a*t_{n-1}\bmod m.$$

$$\therefore t_{n+1}t_{n-1}-t_nt_n=a*a*t_{n-1}*t_{n-1}-a*t_{n-1}*a*t_{n-1}=0\bmod m$$

$$\text{即}T_n=t_{n+1}t_{n-1}-t_nt_n\text{是m的倍数，故}T_n\text{和}T_{n-1}\text{的最大公因数即为m}$$

题型：1.a,b,m已知，用Xn+1反推Xn，由于不知道要推多少项，需要知道flag格式，来反推找到合适的flag格式

```python
import gmpy2
import libnum

a = 113439939100914101419354202285461590291215238896870692949311811932229780896397
b = 72690056717043801599061138120661051737492950240498432137862769084012701248181
m = 72097313349570386649549374079845053721904511050364850556329251464748004927777
c=9772191239287471628073298955242262680551177666345371468122081567252276480156

# c=(a*c0+b)%m
a_1=gmpy2.invert(a,m)

for i in range(2**16):
    c = (c - b) * a_1 % m
    #print(c)
    flag=libnum.n2s(int(c))
    if b'NSSCTF{' in flag:
        print(flag)
        break
```

2.不知道b，先求b，再用第一种方法求

```
import gmpy2
import libnum
from Crypto.Util.number import isPrime

a = 83968440254358975953360088805517488739689448515913931281582194839594954862517
m = 77161425490597512806099499399561161959645895427463118872087051902811605680317
c1=43959768681328408257423567932475057408934775157371406900460140947365416240650
c2=80520433362388643558721028892547812814667280727981604482607525950385529448088

b=(c2-a*c1) % m
#print(b)
#print(gmpy2.gcd(b,m))
a_1 = gmpy2.invert(a,m)
c = c1

for i in range(2**16):
    c = (c-b) * a_1 % m
    flag = libnum.n2s(int(c))

    if b'NSSCTF' in flag:
        print(flag)
        break
```

3.a,b都不知道，先求出a，再操作同第2种

```
import gmpy2
import libnum

m = 96343920769213509183566159649645883498232615147408833719260458991750774595569
c1 = 10252710164251491500439276567353270040858009893278574805365710282130751735178
c2 = 45921408119394697679791444870712342819994277665465694974769614615154688489325
c3 = 27580830484789044454303424960338587428190874764114011948712258959481449527087

a = (c3-c2) * gmpy2.invert(c2-c1,m) % m
# print(gmpy2.gcd(a,m))
a_1 = gmpy2.invert(a,m)
b = (c2-a*c1) % m
# print(gmpy2.gcd(b,m))
c = c1
for i in range(2**16):
    c = (c-b) * a_1 % m
    flag = libnum.n2s(int(c))

    if b'NSSCTF{' in flag:
        print(flag)
        break
```

4.a,b,m都不知道给出多组输出，让我们恢复原始种子 先求出m，然后进行3的操作

```
import gmpy2
import libnum
from Crypto.Util.number import GCD, isPrime, long_to_bytes

c=[47513456973995038401745402734715062697203139056061145149400619356555247755807,
   57250853157569177664354712595458385047274531304709190064872568447414717938749,
   30083421760501477670128918578491346192479634327952674530130693136467154794135,
   38739029019071698539301566649413274114468266283936163804522278316663267625091,
   42506270962409723585330663340839465445484970240895653869393419413017237427900]

t=[]
for i in range(1,len(c)):
    t.append(c[i]-c[i-1])
```

```
m = 0
for i in range(1,len(t)-1):
    m = GCD(t[i+1]*t[i-1]-t[i]**2, m)
# print(isPrime(m))       False

m//=2
# print(isPrime(m))


a = (c[3]-c[2])*gmpy2.invert(c[2]-c[1],m) % m
b = (c[2]-a*c[1]) % m
# print(gmpy2.gcd(a,m))
# print(gmpy2.gcd(b,m))
a_1=gmpy2.invert(a,m)

for i in range(2**16):
    c[1] = (c[1]-b) * a_1 % m
    flag = long_to_bytes(c[1])

    if b'NSSCTF{' in flag:
        print(flag)
        break
```

这里m//=2是因为我们数据太少，可能得到m的倍数，也就是km这时候需要遍历小数，手动除k

5.和4一样都是啥都不知道，只知道数据，但是用的公式不同，是用上述公式一进行反推

```
import gmpy2
from Crypto.Util.number import GCD, isPrime, long_to_bytes

c=[57648351648792284446777383544515312078150027665462203747924668509833442797796,
   90378879763416486117626477831653213918315023665514305359005153448529276829825,
   21826576702665114807208181233864324586557058567478767825970403161758214940301,
   47594460970742467761038407996122637655856234121180714918606854365482948918701,
   11871076497267630136796123094001159466754095580273018347962555675375123133730]

t=[]
for i in range(1,len(c)):
    t.append(c[i]-c[i-1])

m = 0
for i in range(1,len(t)-1):
    m = GCD(t[i+1]*t[i-1]-t[i]**2, m)
print(isPrime(m))   # False m的倍数
print(m)

for i in range(1,100):
    if isPrime(m//i):
        print(i)   # i是4
        m//=i
        break
```

```
    a = (c[3]-c[2])*gmpy2.invert(c[2]-c[1],m) % m
    b = (c[2]-a*c[1]) % m
    # print(gmpy2.gcd(a,m))
    # print(gmpy2.gcd(b,m))
    a_1=gmpy2.invert(a,m)

    for i in range(2**16):
        c[1] = (c[1]-b) * a_1 % m
        flag = long_to_bytes(c[1])

        if b'NSSCTF{' in flag:
            print(flag)
            break
```

6.同样是恢复参数，但是进行了两次加密

这里进行了两次加密，我们得到的并不是连续的输出，而是隔位输出，比如是 X2,X4,X6,X8,X10

```
self.seed = (self.a * self.seed + self.b) % self.m
self.seed = (self.a * self.seed + self.b) % self.m
```

所以我们应该先用公式二恢复模数m

$$\begin{cases} X_2 & = & (a * X_1 + b) & \mathrm{mod} & m \\ X_4 & = & (aX_3 + b) & \mathrm{mod} & m \end{cases} \Rightarrow (X_4 - X_2) = a(X_3 - X_1) \bmod m$$

再用得到的式子求a

$$(X_4 - X_2) = a(X_3 - X_1) \bmod m = a((aX_2 + b) - (aX_0 + b)) \bmod m = a^2(X_2 - X_0) \bmod m$$

最后得到一个很烦的平方，可以写出下列式子

这个方法我也还是一知半解(AMM 算法 也不是很明白)。这里，讲一下我的做法。前面已经说过，本题进行了两轮加密，所给输出是间隔的。

$$X_{n+1} = (aX_n + b) \bmod m = (a(aX_{n-1} + b) + b) \bmod m$$

这里把a平方看成a，(a+1)b看成b，又构成了一个LCG，以下是jio本:

```
import gmpy2
from Crypto.Util.number import GCD, isPrime, long_to_bytes

c =
[25445927935559969212648839062255651208014967526951331344342413906051118248013,
81572970970116732975667604095930675262596098540738447440566868976253289440293,
69567939256251108037791141501604764986761795428152073532189443862320514292 89,
88042506866508011592456777776490262927213783361334741921985316105965255450508,
56528321253217077264818468095361801768772635193272683611306054562555582850 92]

t=[]
for i in range(1,len(c)):
    t.append(c[i]-c[i-1])
```

```
m = 0
for i in range(1,len(t)-1):
    m = GCD(t[i+1]*t[i-1]-t[i]**2, m)
# print(isPrime(m))   # true

a = (c[3]-c[2])*gmpy2.invert(c[2]-c[1],m) % m
b = (c[2]-a*c[1]) % m             # 把(a+1)*b当成b就可以了
# print(gmpy2.gcd(a,m))
# print(gmpy2.gcd(b,m))
a_1=gmpy2.invert(a,m)

for i in range(2**16):
    c[1] = (c[1]-b) * a_1 % m
    flag = long_to_bytes(c[1])

    if b'NSSCTF{' in flag:
        print(flag)
        break
```