ez_Signature

关键在于求d，我们有$nonce * s = z + r * d \equiv order$

设$d = x_1 * 2^{128} + x_2$，则$nonce = x_2 * 2^{128} + x_1$，于是

$z + (2^{128} * r - s) * x_1 = (2^{128} * s - r) * x_2 \ mod \ order$

即$z + a_1 * x_1 = a_2 * x_2 \ mod \ order$

$$\begin{bmatrix} 1 & x_1 & k \end{bmatrix} \begin{bmatrix} z * a^{-1} & 0 & 2^{128} \\ a_1 * a_2^{-1} & 1 & 0 \\ order & 0 & 0 \end{bmatrix} = \begin{bmatrix} x_2 & x_1 & 2^{128} \end{bmatrix}$$

order为256bits，x_1，x_2均是128bits，目标向量有点大，还需优化。

我们自己写个demo，发现当d为254bits时，有50%几率可以规约出来，而当d为252bits时，有90%几率可以规约出来，接近百分之百。

```
from Crypto.Util.number import *
p = 0xfffffffffffffffffffffffffffffffffffffffffffffffffffffffefffffc2f
a = 0
b = 7
E = EllipticCurve(GF(p), [a, b])
G = E.gens()[0]
order = E.order()

#output
iv = '6f80a4db411283cbfc8c2f7520e65d28'
c =
'3c9efa092a2f8479f2d0ca43f5033a8374f737bef2335bcb65a8425ae1a4c5dd7fd0e35
f99c2c0c09d69d6935776f479b19345e9ef9ac466048e98a897d7153e08e4a4ba631b107
ae5e637db74c7e6f2'
public =
E(54937303865575887402131298022599122443206876645668389089331396952974854
50984,
11289033291602595648776467086884499036201249583033100575610195766915659
3225707)
s =
56275471718384662997548252167133434300653095757824127552821000258065614
147953
r =
88763903751156737929684129280869308730236267295398468396864060901397400
254458
msg = '79e3587b06b1caec3323a5f3b944b4946e06aedca38e2b0d6b231c4577a192bf'

z=bytes_to_long(bytes.fromhex(msg))
```

```
a1=2^128*r-s
a2=2^128*s-r
a2_=inverse_mod(a2,order)
z=z*a2_
for u in range(4):
    for v in range(4):
        Z=z+a1*a2_*u*2^126-v*2^126
        m=matrix(ZZ,[[Z,0,2^126],
                     [a1*a2_,1,0],
                     [order,0,0]])
        l=m.LLL()
        for i in l:
            if abs(i[-1])==2^126:
                dl=abs(i[0])
                dh=abs(i[1])
                d=(u*2^126+dh)*2^128+v*2^126+dl
                if d*G==public:
                    print('d=',d)
                    D=d
                    print(u,v)

hd = hex(D)[2:]
if len(hd) % 2 == 1:
    hd = '0' + hd
key = bytes.fromhex(hd)
from Crypto.Cipher import AES
cipher = AES.new(key, AES.MODE_CBC, iv=bytes.fromhex(iv))
ct = bytes.fromhex(c)
flag = cipher.decrypt(ct)
print(flag)
```

ez_pack

题目灵感来源jqctf2023的pack的前半部分，觉得思路很新奇，故与师傅们分享。

考察的是高纬度下格基的优化（模数、映射等等），原来还可以通过对规约后符合要求的向量线性组合找到更短的向量。

```
from Crypto.Util.number import *
from gmpy2 import *
import random
from hashlib import *

def SSP_solver(a, res):
    dim = 128
    w = 2^32
    L = matrix(ZZ, dim+1, dim+1)
```

```
    for _ in range(dim):
        L[_, _] = 2
        L[_, dim] = w*a[_]

    L[dim, dim] = -w*res
    for _ in range(dim):
        L[dim, _] = -1

    basis = L.LLL()[:-1]
    basis = basis.BKZ(block_size=24)
    for row in basis:
        if all([ele in [-1, 1] for ele in row[:-1]]):
            s = [i+1 if i<0 else i for i in row]
            return s

a1, res1 = ...
c1 = SSP_solver(a1, res1)
ls = [i+1 if i==0 else i-1 for i in c1[:128]]
assert (sum(a1[_]*ls[_] for _ in range(128))==res1)
c =
N =
e =

def mixture(d):
    return sha512(hex(d)[2:].encode()).hexdigest()

def decrypt(d,c,N):
    d_=int(mixture(d),16)
    D=gmpy2.next_prime(d_)
    m=pow(c,D,N)
    return long_to_bytes(int(m))

d_=int(mixture(d),16)
print(d_)
D=gmpy2.next_prime(d_)
print(D)
print(decrypt(d,c,N))
```