队伍名称：瑞典厨师长

队伍ID：0x000023

CRYPTO

1.ancient recall

emmmm,自己看不太懂，但题目应该比较简单，丢给ai直接解出来（

```python
Major_Arcana = ["The Fool", "The Magician", "The High Priestess","The Empress", "The Emperor", "The Hierophant","The Lovers", "The Chariot", "Strength","The Hermit", "Wheel of Fortune", "Justice","The Hanged Man", "Death", "Temperance","The Devil", "The Tower", "The Star","The Moon", "The Sun", "Judgement","The World"]
wands = ["Ace of Wands", "Two of Wands", "Three of Wands", "Four of Wands", "Five of Wands", "Six of Wands", "Seven of Wands", "Eight of Wands", "Nine of Wands", "Ten of Wands", "Page of Wands", "Knight of Wands", "Queen of Wands", "King of Wands"]
cups = ["Ace of Cups", "Two of Cups", "Three of Cups", "Four of Cups", "Five of Cups", "Six of Cups", "Seven of Cups", "Eight of Cups", "Nine of Cups", "Ten of Cups", "Page of Cups", "Knight of Cups", "Queen of Cups", "King of Cups"]
swords = ["Ace of Swords", "Two of Swords", "Three of Swords", "Four of Swords", "Five of Swords", "Six of Swords", "Seven of Swords", "Eight of Swords", "Nine of Swords", "Ten of Swords", "Page of Swords", "Knight of Swords", "Queen of Swords", "King of Swords"]
pentacles = ["Ace of Pentacles", "Two of Pentacles", "Three of Pentacles", "Four of Pentacles", "Five of Pentacles", "Six of Pentacles", "Seven of Pentacles", "Eight of Pentacles", "Nine of Pentacles", "Ten of Pentacles", "Page of Pentacles", "Knight of Pentacles", "Queen of Pentacles", "King of Pentacles"]
Minor_Arcana = wands + cups + swords + pentacles
tarot = Major_Arcana + Minor_Arcana
def reverse_step(B):
    B0, B1, B2, B3, B4 = B
    numerator = B0 + B1 + B3 - B2 - B4
    if numerator % 2 != 0:
        raise ValueError("奇数无法整除")
    A1 = numerator // 2
    A0 = B0 - A1
    A2 = B1 - A1
    A3 = B2 - A2
    A4 = B3 - A3
    if A4 + A0 != B4:
        raise ValueError("验证失败")
    return [A0, A1, A2, A3, A4]
final_values = [
    2532951952066291774890498369114195917240794704918210520571067085311474675019,
    2532951952066291774890327666074100357898023013105443178881294700381509795270,
    2532951952066291774890554459287276604903130315859258544173068376967072335730,
    2532951952066291774890865328241532885391510162611534514014409174284299139015,
    2532951952066291774890830662608134156017946376309989934175833913921142609334
]
current = final_values.copy()
for _ in range(250):
    current = reverse_step(current)
def get_card(v):
    for k in range(22):
        if k ^ -1 == v:
            return f"re-{Major_Arcana[k]}"
    index = v % 78
    card = tarot[index]
    if card in Major_Arcana and v == index:
        return card
    return card
cards = [get_card(v) for v in current]
flag = "hgame{" + "&".join(cards).replace(" ", "_") + "}"
print(flag)
#hgame{re-The_Moon&re-The_Sun&Judgement&re-Temperance&Six_of_Cups}
```

2.Intergalactic Bound

add_THcurve部分符合符合https://www.hyperelliptic.org/EFD/g1p/auto-twistedhessian.html 的定义。 所以按照文章里套换元 x′=X/Z y′=Y/Z 得到 ax$^{3+y'}$3+z′^3=dx′y′z'这样构造出了齐次式子之后就可以构造椭圆曲线了。所以现在只需要求a的值即可代入脚本求解。因为

```
d = (a*G[0]^3+G[1]^3+1)%p*inverse(G[0]*G[1],p)%p
```

利用G和Q构造方程解出a

```
p = 55099055368053948610276786301
Gx = 19663446762962927633037926740
Gy = 35074412430915656071777015320
Qx = 26805137673536635825884330180
Qy = 26376833112609309475951186883
# 计算 Gy^3 + 1 mod p
Gy_cubed = pow(Gy, 3, p)
Gy_cubed_plus_1 = (Gy_cubed + 1) % p
# 计算 Qy^3 + 1 mod p
Qy_cubed = pow(Qy, 3, p)
Qy_cubed_plus_1 = (Qy_cubed + 1) % p
# 计算分子：(Gy^3+1)*Qx*Qy - (Qy^3+1)*Gx*Gy mod p
term1 = (Gy_cubed_plus_1 * Qx) % p
term1 = (term1 * Qy) % p
term2 = (Qy_cubed_plus_1 * Gx) % p
term2 = (term2 * Gy) % p
numerator = (term1 - term2) % p
```

```
# 计算分母：Qx^3*Gx*Gy - Gx^3*Qx*Qy mod p
Qx_cubed = pow(Qx, 3, p)
term3 = (Qx_cubed * Gx) % p
term3 = (term3 * Gy) % p
Gx_cubed = pow(Gx, 3, p)
term4 = (Gx_cubed * Qx) % p
term4 = (term4 * Qy) % p
denominator = (term3 - term4) % p
# 计算逆元
inv_denominator = pow(denominator, -1, p)
a = (numerator * inv_denominator) % p
print(a)
#a=39081810733380615260725035189
```

求得a的值构建出椭圆曲线后使用 Pohlig Hellman 即可解出 Q = xG 中的 x

```
from Crypto.Util.number import *
a = 39081810733380615260725035189
p = 55099055368053948610276786301
P = (19663446762962927633037926740, 35074412430915656071777015320)
Q = (26805137673536635825884330180, 26376833112609309475951186883)
d = (a * Q[0] ** 3 + Q[1] ** 3 + 1) * inverse(Q[0] * Q[1], p) % p
# construct ECC to get a solution of aX^3+Y^3+Z^3=dXYZ
R.<x,y,z> = Zmod(p)[]
cubic = a * x^3 + y^3 + z^3 - d*x*y*z
E = EllipticCurve_from_cubic(cubic,morphism=True)
P = E(P)
Q = E(Q)
P_ord = P.order()
def Pohlig_Hellman(n, P, Q):
    return discrete_log(Q, P, ord=n, operation='+')
x = Pohlig_Hellman(P_ord,P,Q)
print(x)
#x=2633177798829352921583206736
```

```
import hashlib
from Crypto.Cipher import AES
from Crypto.Util.Padding import unpad
x = 2633177798829352921583206736
key = hashlib.sha256(str(x).encode()).digest()
cipher = AES.new(key, AES.MODE_ECB)
ciphertext = b"k\xe8\xbe\x94\x9e\xfc\xe2\x9e\x97\xe5\xf3\x04'\x8f\xb2\x01T\x06\x88\x04\xeb3Jl\xdd Pk$\x00:\xf5"
decrypted_flag = unpad(cipher.decrypt(ciphertext), 16)
print(f"解密后的数据: {decrypted_flag}")
#解密后的数据: b'hgame{N0th1ng_bu7_up_Up_UP!}'
```

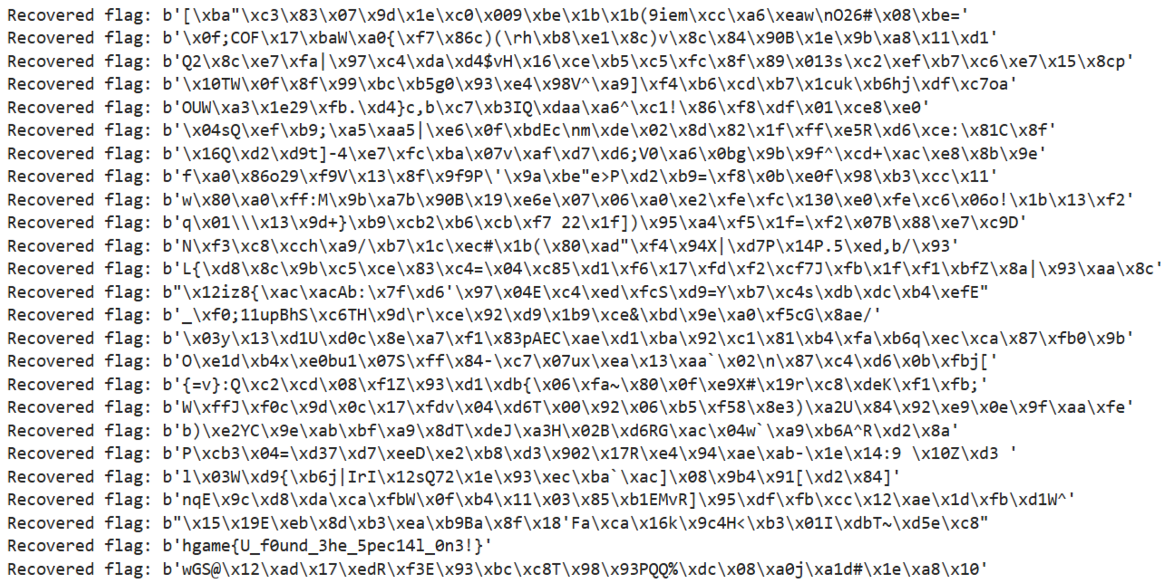
3.Spica

隐子集和问题（HSSP / Hidden Subset Sum Problem）。解题参考：https://yanmo312.github.io/2022/11/26/gemima_6/#%E4%B8%89%E3%80%81%E9%9A%90%E5%AD%90%E9%9B%86%E5%92%8C%E9%97%AE%E9%A2%98%EF%BC%88HSSP-Hidden-Subset-Sum-Problem%EF%BC%89

```
from Crypto.Util.number import *
from sage.all import *
import time
def read_data(filename):
    with open(filename, 'r') as f:
        m = int(f.readline().strip())
        n = 70
        p = int(f.readline().strip())
        h_line = f.readline().strip()
        w = list(map(int, h_line[1:-1].split(', ')))
    return m, n, p, w
# 生成 orthoLattice 的相关函数
def orthoLattice(b, x0):
    m = b.length()
    M = Matrix(ZZ, m, m)
    # 生成正交矩阵
    for i in range(1, m):
        M[i, i] = 1
    M[1:m, 0] = -b[1:m] * inverse_mod(b[0], x0)
    M[0, 0] = x0
    for i in range(1, m):
        M[i, 0] = mod(M[i, 0], x0)
    return M
def allpmones(v):
    return len([vj for vj in v if vj in [-1, 0, 1]]) == len(v)
def allones(v):
    if all(vj in (0, 1) for vj in v):
        return v
    if all(vj in (0, -1) for vj in v):
        return -v
    return None
# 恢复只包含 {0,1} 或 {-1,0,1} 的向量
def recoverBinary(M5):
    lv = [allones(vi) for vi in M5 if allones(vi)]
    n = M5.nrows()
    for v in lv:
        for i in range(n):
            nv = allones(M5[i] - v)
            if nv and nv not in lv:
                lv.append(nv)
            nv = allones(M5[i] + v)
            if nv and nv not in lv:
                lv.append(nv)
    return Matrix(lv)
def kernelLLL(M):
    n = M.nrows()
    m = M.ncols()
    if m < 2 * n:
        return M.right_kernel().matrix()
    K = 2 ^ (m // 2) * M.height()
    MB = Matrix(ZZ, m + n, m)
    MB[:n] = K * M
    MB[n:] = identity_matrix(m)
    MB2 = MB.T.LLL().T
    assert MB2[:n, : m - n] == 0
    Ke = MB2[n:, : m - n].T
    return Ke
def attack(m, n, p, w):
    print("n =", n, "m =", m)
    iota = 0.035
    nx0 = int(2 * iota * n^2 + n * log(n, 2))
    print("nx0 =", nx0)
    x0 = p
    b = vector(w)
    M = orthoLattice(b, x0)
    t = time.time()
    M2 = M.LLL()
    print("LLL step1: %.1f" % (time.time() - t))
    MOrtho = M2[: m - n]
    print("log(Height, 2) = ", int(log(MOrtho.height(), 2)))
    t2 = time.time()
    ke = kernelLLL(MOrtho)
    print("Kernel: %.1f" % (time.time() - t2))
    if n > 170:
        return
    beta = 2
    tbk = time.time()
    while beta < n:
        if beta == 2:
            M5 = ke.LLL()
        else:
            M5 = M5.BKZ(block_size=beta)
        if len([True for v in M5 if allpmones(v)]) == n:
            break
        if beta == 2:
            beta = 10
        else:
            beta += 10
    print("BKZ beta=%d: %.1f" % (beta, time.time() - tbk))
    t2 = time.time()
    MB = recoverBinary(M5)
    print("Recovery: %.1f" % (time.time() - t2))
    print("Number of recovered vector = ", MB.nrows())
    print("Number of recovered vector.T = ", MB.ncols())
    return MB
m, n, p, w = read_data('data.txt')
res = attack(m, n, p, w)
def bits_to_long(bits):
    return int(''.join(str(bit) for bit in bits), 2)
def extract_flags(MB):
    flags = []
    # 遍历 MB 的每一行，将每一行转换为一个二进制数字
    for row in MB:
        flag_bits = [int(element) for element in row]  # 获取每行的二进制位
        flag_long = bits_to_long(flag_bits)  # 转换为整数
        flag = long_to_bytes(flag_long)  # 转换为字节串
        flags.append(flag)
    return flags
flags = extract_flags(res)
for flag in flags:
    print(f"Recovered flag: {flag}")
```

```
#Recovered flag: b'hgame{U_f0und_3he_5pec14l_0n3!}'
```

感觉代码最后加个对flag的处理，判断只有符合hgame{}格式的flag输出会好点（但数据不是很大，还是一眼就从输出里找到正确flag）。　输出部分截图还是很好找的是吧（



```
Recovered flag: b'[\xba"\xc3\x83\x07\x9d\x1e\xc0\x009\xbe\x1b\x1b(9iem\xcc\xa6\xeaw\nO26#\x08\xbe='
Recovered flag: b'\x0f;COF\x17\xbaW\xa0{\xf7\x86c)(\rh\xb8\xe1\x8c)v\x8c\x84\x90B\x1e\x9b\xa8\x11\xd1'
Recovered flag: b'Q2\x8c\xe7\xfa|\x97\xc4\xda\xd4$vH\x16\xce\xb5\xc5\xfc\x8f\x89\x013s\xc2\xef\xb7\xc6\xe7\x15\x8cp'
Recovered flag: b'\x10TW\x0f\x8f\x99\xbc\xb5g0\x93\xe4\x98V^\xa9]\xf4\xb6\xcd\xb7\x1cuk\xb6hj\xdf\xc7oa'
Recovered flag: b'OUW\xa3\x1e29\xfb.\xd4}c,b\xc7\xb3IQ\xdaa\xa6^\xc1!\x86\xf8\xdf\x01\xce8\xe0'
Recovered flag: b'\x04sQ\xef\xb9;\xa5\xaa5|\xe6\x0f\xbdEc\nm\xde\x02\x8d\x82\x1f\xff\xe5R\xd6\xce:\x81C\x8f'
Recovered flag: b'\x16Q\xd2\xd9t]-4\xe7\xfc\xba\x07v\xaf\xd7\xd6;V0\xa6\x0bg\x9b\x9f^\xcd+\xac\xe8\x8b\x9e'
Recovered flag: b'f\xa0\x86o29\xf9V\x13\x8f\x9f9P\'\x9a\xbe"e>P\xd2\xb9=\xf8\x0b\xe0f\x98\xb3\xcc\x11'
Recovered flag: b'w\x80\xa0\xff:M\x9b\xa7b\x90B\x19\xe6e\x07\x06\xa0\xe2\xfe\xfc\x130\xe0\xfe\xc6\x06o!\x1b\x13\xf2'
Recovered flag: b'q\x01\\\x13\x9d+}\xb9\xcb2\xb6\xcb\xf7 22\x1f])\x95\xa4\xf5\x1f=\xf2\x07B\x88\xe7\xc9D'
Recovered flag: b'N\xf3\xc8\xcch\xa9/\xb7\x1c\xec#\x1b(\x80\xad"\xf4\x94X|\xd7P\x14P.5\xed,b/\x93'
Recovered flag: b'L{\xd8\x8c\x9b\xc5\xce\x83\xc4=\x04\xc85\xd1\xf6\x17\xfd\xf2\xcf7J\xfb\x1f\xf1\xbfZ\x8a|\x93\xaa\x8c'
Recovered flag: b"\x12iz8{\xac\xacAb:\x7f\xd6'\x97\x04E\xc4\xed\xfcS\xd9=Y\xb7\xc4s\xdb\xdc\xb4\xefE"
Recovered flag: b'_\xf0;11upBhS\xc6TH\x9d\r\xce\x92\xd9\x1b9\xce&\xbd\x9e\xa0\xf5cG\x8ae/'
Recovered flag: b'\x03y\x13\xd1U\xd0c\x8e\xa7\xf1\x83pAEC\xae\xd1\xba\x92\xc1\x81\xb4\xfa\xb6q\xec\xca\x87\xfb0\x9b'
Recovered flag: b'O\xe1d\xb4x\xe0bu1\x07S\xff\x84-\xc7\x07ux\xea\x13\xaa`\x02\n\x87\xc4\xd6\x0b\xfbj['
Recovered flag: b'{=v}:Q\xc2\xcd\x08\xf1Z\x93\xd1\xdb{\x06\xfa~\x80\x0f\xe9X#\x19r\xc8\xdeK\xf1\xfb;'
Recovered flag: b'W\xffJ\xf0c\x9d\x0c\x17\xfdv\x04\xd6T\x00\x92\x06\xb5\xf58\x8e3)\xa2U\x84\x92\xe9\x0e\x9f\xaa\xfe'
Recovered flag: b'b)\xe2YC\x9e\xab\xbf\xa9\x8dT\xdeJ\xa3H\x02B\xd6RG\xac\x04w`\xa9\xb6A^R\xd2\x8a'
Recovered flag: b'P\xcb3\x04=\xd37\xd7\xeeD\xe2\xb8\xd3\x902\x17R\xe4\x94\xae\xab-\x1e\x14:9 \x10Z\xd3 '
Recovered flag: b'l\x03W\xd9{\xb6j|IrI\x12sQ72\x1e\x93\xec\xba`\xac]\x08\x9b4\x91[\xd2\x84]'
Recovered flag: b'nqE\x9c\xd8\xda\xca\xfbW\x0f\xb4\x11\x03\x85\xb1EMvR]\x95\xdf\xfb\xcc\x12\xae\x1d\xfb\xd1W^'
Recovered flag: b"\x15\x19E\xeb\x8d\xb3\xea\xb9Ba\x8f\x18'Fa\xca\x16k\x9c4H<\xb3\x01I\xdbT~\xd5e\xc8"
Recovered flag: b'hgame{U_f0und_3he_5pec14l_0n3!}'
Recovered flag: b'wGS@\x12\xad\x17\xedR\xf3E\x93\xbc\xc8T\x98\x93PQQ%\xdc\x08\xa0j\xa1d#\x1e\xa8\x10'
```