Misc

我叫曼波

没什么难度，对照着写出解密算法即可

1. 将曼波转为base3编码
2. base3编码每5位化为一组，一组对应一个字符
3. 带着密钥直接RC4解密

```
import base64


def manbo_decode(manbo_text: str):
    t = manbo_text.replace("曼波", "0")
    t = t.replace("哦耶", "1")
    t = t.replace("哇嗷", "2")
    return t


def base3_to_str(base3_s: str):
    s = [base3_s[i : i + 5] for i in range(0, len(base3_s), 5)]
    cipher = ""
    for i in range(len(s)):
        cipher += chr(int(s[i], 3))

    return cipher


def RC4_decrypt(cipher, K):
    S = [0] * 256
    T = [0] * 256
    for i in range(0, 256):
        S[i] = i
        T[i] = K[i % len(K)]

    j = 0
    for i in range(0, 256):
        j = (j + S[i] + ord(T[i])) % 256
        S[i], S[j] = S[j], S[i]

    i = 0
    j = 0

    plain = []
    cipher = base64.b64decode(cipher.encode()).decode()
    for s in cipher:
        i = (i + 1) % 256
        j = (j + S[i]) % 256
        S[i], S[j] = S[j], S[i]
        t = (S[i] + S[j]) % 256
        k = S[t]
        plain.append(chr(ord(s) ^ k))
```

```
    return "".join(plain)


def decode(key, manbo_text):
    base3_s = manbo_decode(manbo_text)
    cipher = base3_to_str(base3_s)
    plain_text = RC4_decrypt(cipher, key)
    return plain_text
```

然而，这是一场漫长的等待，硬控了我一个晚上，如果我会用脚本就好了。。。

```
Please input your choice and MANBO will make a series of corres
ponding behavioral responses.
1.generate the next part of flag
2.show current key
3.show current ciphertext
> 3
哦耶哦耶哦耶曼波哇嗷曼波哇嗷曼波曼波曼波哦耶曼波曼波曼波曼波曼
波哇嗷曼波哇嗷哦耶



Please input your choice and MANBO will make a series of corres
ponding behavioral responses.
1.generate the next part of flag
2.show current key
3.show current ciphertext
> 1
You've reached the end of flag.Good Luck!----MANBO
```

总之，一个个手动输入得到flag：

`OxGame{OH_yEah_Wow_Duang_HajiMi_u_MADE_it!_and_MaY_5e_YOu_hAv4_HeArD_7he_STOry_Of_Gu_Gao_MaN_BO}`

Crypto

Diffie-Hellman

分析代码，先验证，再交换密钥，再发密文

写出解密的代码，先爆破验证码，再生成私钥、公钥，交换后用共享密钥解密即可

```
from string import ascii_letters, digits
from hashlib import sha256
from random import choice
from itertools import product
from Crypto.Util.number import isPrime, getPrime
from Crypto.Cipher import AES
from hashlib import md5
from Crypto.Util.Padding import unpad


def find_proof(proof, target_hash):
```

```
    dict_ = ascii_letters + digits
    for word in ("".join(combination) for combination in product(dict_,
repeat=4)):
        if sha256((word + proof).encode()).hexdigest() == target_hash:
            return word
    return None


# 从用户那里获取proof和target_hash
proof = input("请输入proof: ")
target_hash = input("请输入target_hash: ")

# 打印出需要破解的哈希值
print(f"[+] sha256(XXXX+{proof}) == {target_hash}")

# 找到正确的XXXX
xxxx = find_proof(proof, target_hash)
if xxxx:
    print(f"找到的XXXX是: {xxxx}")
else:
    print("没有找到正确的XXXX")


q = int(input("请输入q: "))
g = int(input("请输入g: "))
Alice_PubKey = int(input("请输入Alice的公钥: "))

Bob_PriKey = 1234567890  # Bob私钥
Bob_PubKey = pow(g, Bob_PriKey, q)  # Bob的公钥
print(f"Bob的公钥是: {Bob_PubKey}")

ct_hex = input("把你的密文的十六进制字符串放这里")  # 加密后的密文的十六进制字符串
ct = bytes.fromhex(ct_hex)  # 将十六进制字符串转换为字节串

# 计算共享密钥
Share_Key = pow(Alice_PubKey, Bob_PriKey, q)

# 使用共享密钥的MD5哈希值作为AES解密的密钥
cipher = AES.new(md5(str(Share_Key).encode()).digest(), AES.MODE_ECB)

# 解密密文
pt = cipher.decrypt(ct)
print("解密的消息:", pt.decode())

# 去除填充
try:
    pt = unpad(pt, AES.block_size)
    print("解密的消息:", pt.decode())
except ValueError:
    print("解密失败，可能密钥错误或填充不正确。")
```

这是运行过程

```
┌──(root㉿Spreng)-[~]
└─# nc 118.195.138.159 10000
[+] sha256(XXXX+zDiTjZjxIV6pJH98) ==
360a6e4f92b235fce079f53aa2fdf87adcbcd92df7b1eb91a829f1466a3daec7
[+] Plz tell me XXXX: Q2kN
Share (q,g) : (26439189564619642815923329850569850543 3,
142708725683642572720700680917171427186)
Alice_PubKey : 10380922234684568182213999424407843 1430
[+] Give me the Bob_PubKey
> 154766045297383002137664386730844042405
Bob_PubKey : 154766045297383002137664386730844042405
Alice tell Bob :
0a1cb7f20be15c8ac26bd72e31955159c5098296cb930ff868e3b15c42755fe63544c0b88cbd7fa2e
e3f2ff2abbf2887
```

```
请输入proof: zDiTjZjxIV6pJH98
请输入target_hash:
360a6e4f92b235fce079f53aa2fdf87adcbcd92df7b1eb91a829f1466a3daec7
[+] sha256(XXXX+zDiTjZjxIV6pJH98) ==
360a6e4f92b235fce079f53aa2fdf87adcbcd92df7b1eb91a829f1466a3daec7
找到的XXXX是: Q2kN
请输入q: 26439189564619642815923329850569850543 3
请输入g: 142708725683642572720700680917171427186
请输入Alice的公钥: 10380922234684568182213999424407843 1430
Bob的公钥是: 154766045297383002137664386730844042405
把你的密文的十六进制字符串放这里
0a1cb7f20be15c8ac26bd72e31955159c5098296cb930ff868e3b15c42755fe63544c0b88cbd7fa2e
e3f2ff2abbf2887
解密的消息: 0xGame{107c7960-d339-48b5-92b9-d59ad5644cf6}
```

Elgamal

这道题是Elgamal已知签名伪造，算法基于以下结论：

$$
\begin{aligned}
u &= m'm^{-1} \mod \varphi(p) \\
s' &= su \mod \varphi(p) \\
r' &\equiv ru \mod \varphi(p) \\
r' &\equiv r \mod p
\end{aligned}
$$

其中 r' 可用CRT求

```python
#!/usr/local/bin/python

from hashlib import sha256


from Crypto.Util.number import getPrime, isPrime, inverse
from hashlib import sha256
from random import randint
import findProof


def Hash(msg):
```

```
    # 哈希函数
    return int(sha256(msg).hexdigest(), 16)

def forge_signature(q, m, m_, r, s):
    m = Hash(m)
    m_ = Hash(m_)
    phi = q - 1
    u = (m_ * inverse(m, phi)) % phi
    s_ = (s * u) % phi
    r_ = CRT(r * u, r, phi, q)
    return (r_, s_)


def CRT(r1, r2, p1, p2):
    """使用中国剩余定理求解"""
    M = p1 * p2
    M1 = p2
    M2 = p1

    r = (r1 * M1 * inverse(M1, p1) + r2 * M2 * inverse(M2, p2)) % M
    return r


if __name__ == "__main__":
    findProof.main()

    q = int(input("Enter q: "))
    msg = input("Enter message: ")
    msg = bytes.fromhex(msg)
    r = int(input("Enter r: "))
    s = int(input("Enter s: "))

    msg_ = (b"0xGame2024_Crypto_is_a_great_game").hex()
    print(f"msg_: {msg_}")

    msg_ = bytes.fromhex(msg_)
    r_, s_ = forge_signature(q, msg, msg_, r, s)
    print(f"r_: {r_}")
    print(f"s_: {s_}")
```

运行过程：

```
┌──(root㉿Spreng)-[~]
└─# nc 118.195.138.159 10002
[+] sha256(XXXX+0ERfrggK1aHpr73u) ==
5d0826ebbc35e1156bba21721cce62d16332b2e23420af28182b80c61dc4f8b4
[+] Plz tell me XXXX: O8Wo
My Public Key (q,g,y):
(886798469271216258939475359268446289384972138380835927087059194630959142090150998734188848754080085338981170199816629242718554364890543200895344255684400 3,
567368802476024489446065429790865574942559626876832015787106089947040835160933669381028918617059177382590397662187587687059855933100778373848420453092288 8,
519233711605317306889507662782994479881867061574573399866371080503733316029438362679737818955554309376966296098849752499325845039155213207226028662882515 4)
```

```
The input msg : 57656c636f6d655f746f5f307847616d65323032345f43727970746f
And the msg signatue (r,s):
(352530056367503643041220054951514689361141657095929789988603559671795759510942506476850127260924070528619389222599570954638125837023614541345648397192742,
132843254408237794232442474996109984425668211319572247804449885675496774749900976015326444855071234288184325397858579536033018724597552338265247453596477 3)
Now, it's your turn to help me sign something
[+] Give me your message:
>307847616d65323032345f43727970746f5f69735f615f67726561745f67616d65
[+] Give me your r:
>135574106469941959853055032412816065300227356723485192603781636275815193518678785816589683684567815404698650653739178021706207002017746992564527183137109925130610337385004968079867847069232253083882138061344357782948547459815948815082297157362171541074444871781833298775294466975880785789947501521612816440 42
[+] Give me your s:
>264449526761687823824880704095087998070875076337640456069221303902170420624590562200851687380471401291862594895192576198136596420145862107728280880130225
flag : 0xGame{93e9adb8-8a6d-4517-9c61-13081c413e41}
```

```
请输入proof: OERfrggK1aHpr73u
请输入target_hash:
5d0826ebbc35e1156bba21721cce62d16332b2e23420af28182b80c61dc4f8b4
[+] sha256(XXXX+OERfrggK1aHpr73u) ==
5d0826ebbc35e1156bba21721cce62d16332b2e23420af28182b80c61dc4f8b4
找到的XXXX是: O8Wo
Enter q:
886798469271216258939475359268446289384972138380835927087059194630959142090150998734188848754080085338981170199816629242718554364890543200895344255684400 3
Enter message: 57656c636f6d655f746f5f307847616d65323032345f43727970746f
Enter r:
352530056367503643041220054951514689361141657095929789988603559671795759510942506476850127260924070528619389222599570954638125837023614541345648397192742
Enter s:
132843254408237794232442474996109984425668211319572247804449885675496774749900976015326444855071234288184325397858579536033018724597552338265247453596477 3
msg_: 307847616d65323032345f43727970746f5f69735f615f67726561745f67616d65
r_:
135574106469941959853055032412816065300227356723485192603781636275815193518678785816589683684567815404698650653739178021706207002017746992564527183137109925130610337385004968079867847069232253083882138061344357782948547459815948815082297157362171541074444871781833298775294466975880785789947501521612816440 42
s_:
264449526761687823824880704095087998070875076337640456069221303902170420624590562200851687380471401291862594895192576198136596420145862107728280880130225
```

LFSR-baby

这道题就是逆推原始状态，先描述一下顺推一次的过程，output^=mask & state

```
output ^= mask[i] & state[i]
# mask:     [1, 0, 0, 1]
# state:    [1, 1, 0, 1]
# result:   [1, 0, 0, 1]
# output: 0>>1>>1>>1>>0
# output
# state: [1 |1, 0, 1] 0
#         >>[1, 0, 1, 0]
```

由于提供的状态是顺推了128次的，等于state长度，所以我们知道一个完整的状态，下面描述逆推：

```
# state:       [1, 1, 0, 1]
# o_state:  [?, 1, 1, 0]
# mask:     [1, 0, 0, 1]
# result:   [?, 0, 0, 0]
#          0 1<<1<<1<<1<<1
# ? == 0^1 == 1
```

脚本如下：

```python
from hashlib import md5


def MD5(m):
    return md5(str(m).encode()).hexdigest()


def recover_state(o_state: list[int], mask: list[int], length):
    # 计算state的最后一个元素
    o = o_state[-1]
    for i in range(length - 1, 0, -1):
        o ^= o_state[i - 1] & mask[i]
    o ^= 0
    state = [o] + o_state[:-1]

    return state


def decode(mask: list[int], outputs: list[int]):
    current_state = outputs
    for _ in range(128):
        current_state = recover_state(current_state, mask, 128)
    seed = int("".join(str(bit) for bit in current_state), 2)  # state转为seed
    return seed


if __name__ == "__main__":
    # 已知mask_seed,两次LFSR运算得到state,逆推seed
    Mask_seed = 245818399386224174743537177607796459213
    mask = [int(i) for i in bin(Mask_seed)[2:]]
    outputs1 = 103763907686833223776774671653901476306
    outputs1 = [0] + [int(bit) for bit in bin(outputs1)[2:]]
    outputs2 = 136523407741230013545146835206624093442
    outputs2 = [0] + [int(bit) for bit in bin(outputs2)[2:]]
```

```
    seed = decode(mask, outputs1)
    print(f"0xGame{{{MD5(seed)}}}")
    # 0xGame{030ec00de18ceb4ddea5f6612d28bf39}
```

说几个点：

1. 这里mask的第一位必须是1，不然逆推不了
2. 两组输出只有127位，需要在前面补个0
3. 虽然有两组输出，但只要用第一个逆推就可以了

LFSR-eazy

这道题就是根据给出的数据算出mask，需要对这种布尔运算有一定敏感度（我也是现学的），关键就是构造线性方程组：

$$k_{n+t} = c_1 k_t \oplus c_2 k_{t+1} \oplus \cdots \oplus c_n k_{t+n-1}$$

这里n=128，c表示mask的各个位，k表示state的各个位，k长度为2n即可求解，虽然有两组输出加初始状态，只要用两个就够了。

这里的线性方程组因为是二进制的，没法用numpy，就自己写了（学线代学的）

脚本如下：

```
from hashlib import md5


def MD5(m):
    return md5(str(m).encode()).hexdigest()


def func(A: list[list[int]], B: list[int], n: int):
    # A n行n+1列的二进制矩阵(增广矩阵)
    A = [A[i] + [B[i]] for i in range(n)]  # 增广矩阵
    for i in range(n):
        # 找出第i行的主元
        for j in range(i, n):
            if A[j][i]:
                # 交换第i行和第j行
                A[i], A[j] = A[j], A[i]
                break
        else:
            continue
        # 消元
        for j in range(n):
            if i != j and A[j][i]:
                A[j] = [A[j][k] ^ A[i][k] for k in range(n + 1)]
    X = [A[i][n] for i in range(n)]
    return X


if __name__ == "__main__":

    """
    165943427582675380464843619836793254673
```

```
    299913606793279087601607783679841106505
    192457791072277356149547266972735354901
    """

    outputs1 = 299913606793279087601607783679841106505
    outputs2 = 192457791072277356149547266972735354901
    outputs1 = [int(bit) for bit in bin(outputs1)[2:]]
    outputs2 = [int(bit) for bit in bin(outputs2)[2:]]

    # 定义长度为 2n 的二进制列表 K
    n = 128
    K = outputs1 + outputs2

    # 构造系数矩阵 A 和常数向量 B
    A = [K[i : i + n] for i in range(n)]
    B = K[n : 2 * n]

    # 求解线性方程组
    X = func(A, B, n)
    mask = int("".join(str(bit) for bit in X), 2)

    print(f"0xGame{{{MD5(mask)}}}")
    # 0xGame{d56821feacab64cdb87c754ad06823a2}
```

RC4

脚本：

一般来说RC4一次一密安全，但这里的keystream并没有变化，只要逆推出密钥流即可求解。

一个注意的点是明文的长度要足够长，我的正好和flag一样长

```
import findProof

def find_keystream(m, c):
    # 将密文从十六进制转换为字节
    c_bytes = bytes.fromhex(c)

    # 将明文转换为字节
    m_bytes = bytes.fromhex(m)
    # 提取密钥流
    keystream = bytes(a ^ b for a, b in zip(m_bytes, c_bytes))
    return keystream


# 使用提取的密钥流来解密其他密文
def decrypt(ciphertext, keystream):
    if type(ciphertext) == bytes:
        ct = ciphertext
    else:
        ct = bytes.fromhex(ciphertext)

    result = b""
    for i in range(len(ct)):
        result += bytes([ct[i] ^ keystream[i % len(keystream)]])
```

```
    return result.decode("utf-8")


if __name__ == "__main__":
    findProof.main()
    m = "0xGame{Hello-World-This-Is-A-Very-long-flag}".encode("utf-8").hex()
    print(f"m={m}")
    c = input("请输入加密文本")
    keystream = find_keystream(m, c)

    c = input("请输入加密文本")
    # 解密操作
    decrypted_text = decrypt(c, keystream)
    print("解密后的文本:", decrypted_text)
```

运行过程：

```
┌──(root㉿Spreng)-[~]
└─# nc 118.195.138.159 10001
[+] sha256(XXXX+A2GqA4fgP7eMaAIK) ==
f21dca8a1c7818c93204236b5589b81a5e865a9010d2f2e0669431b3a61b3309
[+] Plz tell me XXXX: gCTR
[+] Give me the text you want to encrypt:
>307847616d657b48656c6c6f2d576f726c642d546869732d49732d412d566572792d6c6f6e672d666c61677d
[+] Here are the encrypt result:
c =
d38ae1aa11d23a7e64ca5ffa13c88cc6b8aaa16b265cd118a2c43fb4c8f5486d9606aea60d22c30a419496f6
[+] Give you the encrypted flag:
c =
d38ae1aa11d23a0e30900ba70dacd499e2f9bf0e630c9356dc9a76c5d393002c8a4da1ad0520df0e4cc097f6
```

```
请输入proof: A2GqA4fgP7eMaAIK
请输入target_hash:
f21dca8a1c7818c93204236b5589b81a5e865a9010d2f2e0669431b3a61b3309
[+] sha256(XXXX+A2GqA4fgP7eMaAIK) ==
f21dca8a1c7818c93204236b5589b81a5e865a9010d2f2e0669431b3a61b3309
找到的XXXX是: gCTR
m=307847616d657b48656c6c6f2d576f726c642d546869732d49732d412d566572792d6c6f6e672d666c61677d
请输入加密文本
d38ae1aa11d23a7e64ca5ffa13c88cc6b8aaa16b265cd118a2c43fb4c8f5486d9606aea60d22c30a419496f6
请输入加密文本
d38ae1aa11d23a0e30900ba70dacd499e2f9bf0e630c9356dc9a76c5d393002c8a4da1ad0520df0e4cc097f6
解密后的文本: 0xGame{81682337-6731-91c7-d060-3efcdfe1ba5f}
```

RSA-IV

总之就是写脚本算

```python
import gmpy2
from Crypto.Util.number import *
from sympy.ntheory import factorint
import findProof


def rsa0(n, e, c):
    def de(c, e, n):
        k = 0
        while True:
            m = c + n * k
            result, flag = gmpy2.iroot(m, e)
            if True == flag:
                return result
            k += 1

    m = de(c, e, n)
    print(m)


def rsa1(n, e, c, dp):
    p, q = factorint(n).keys()
    d = gmpy2.invert(e, (p - 1) * (q - 1))

    m = pow(c, d, n)
    print(m)


def rsa2(n, e, c):
    p, q = factorint(n).keys()
    d = gmpy2.invert(e, (p - 1) * (q - 1))

    m = pow(c, d, n)
    print(m)


def rsa3(n, e, c, e_, c_):

    p, q = factorint(n).keys()
    d = gmpy2.invert(e, (p - 1) * (q - 1))

    m = pow(c, d, n)
    print(m)


if __name__ == "__main__":
    findProof.main()
    n, e, c = input("Enter n, e, c: ").split(",")
    n, e, c = int(n), int(e), int(c)
    rsa0(n, e, c)
    n, e, c, dp = input("Enter n, e, c, dp: ").split(",")
    n, e, c, dp = int(n), int(e), int(c), int(dp)
    rsa1(n, e, c, dp)
    n, e, c = input("Enter n, e, c: ").split(",")
    n, e, c = int(n), int(e), int(c)
```

```
    rsa2(n, e, c)
    n, e, c, e_, c_ = input("Enter n, e, c, e_, c_: ").split(",")
    n, e, c, e_, c_ = int(n), int(e), int(c), int(e_), int(c_)
    rsa3(n, e, c, e_, c_)
```

这是运行过程

```
┌──(root㉿Spreng)-[~]
└─# nc 118.195.138.159 10003
[+] sha256(XXXX+w9DO0LUBTJj5RCMl) ==
3df53a7ad0501dc4591a6cd9afad1cd193a2246ab99af41b96dfc5dc8560e143
[+] Plz tell me XXXX: hh8T
[+] input choice:
>0
(10369596058700265147804638930737239392452271429827176407602600037556620020485982
56163812253, 3,
40368981912258047629226558170434736364399371539930206195642868576827409930436379
38069)
[+] input answer:
>34304355600142096156705763789
[+] score:1
[+] input choice:
>1
(13772444429422763557581603919887263387 9, 65537,
45118616519967059354565601566228957486, 4177961855610225677)
[+] input answer:
>307553862897987837704818011 33
[+] score:2
[+] input choice:
>2
(18617192733940655118450657311872841155 3, 13056123510065050093202086694143409827,
24656933185203234815907939571590932040)
[+] input answer:
>10031827255389277718757460954
[+] score:3
[+] input choice:
>3
(26156154914745377069120430253476474006 1,
16568848140377285971479716073660193736 3, 15316048306105491836825914284113147973 3,
13287359868879464615406256672498401263 3, 21688961825736844253283585404535248758 4)
[+] input answer:
>10043149880954326724251344473
[+] score:4
[+] flag:b'0xGame{2b5e024a-3c62-4f4a-afe0-b81851d9efc8}'
```

Reverse

BabyUPX

先用exeinfope查壳，发现有upx加壳，用upx脱壳，使用IDA解析

```
int __cdecl main(int argc, const char **argv, const char **envp)
{
```

```
  int result; // eax
  char Str1; // [rsp+20h] [rbp-40h]
  int v5; // [rsp+5Ch] [rbp-4h]

  _main(argc, argv, envp);
  v5 = 0;
  puts("input flag here:");
  scanf("%s", &Str1);
  v5 = strncmp(&Str1, "0xGame", 6ui64);
  if ( v5 || strlen(&Str1) != 44 )
  {
    printf("invalid");
    result = 0;
  }
  else
  {
    encode(&Str1);
    if ( !strcmp(&Str1, &encdata) )
      printf("succeed");
    result = 0;
  }
  return result;
}
```

只要获取encdata并解码即可，这是python代码：

```python
def decode(encoded_bytes):
    decoded_bytes = bytearray()

    for byte in encoded_bytes:
        # 逆转编码过程：将字节右移4位，然后除以16，再与原始字节左移4位进行或操作
        original_byte = ((byte & 0x0F) << 4) | ((byte & 0xF0) >> 4)
        decoded_bytes.append(original_byte)

    return bytes(decoded_bytes)


# 提供的加密数据
encdata = (
    0x03,
    0x87,
    0x74,
    0x16,
    0xD6,
    0x56,
    0xB7,
    0x63,
    0x83,
    0x46,
    0x66,
    0x66,
    0x43,
    0x53,
    0x83,
    0xD2,
```

```
    0x23,
    0x93,
    0x56,
    0x53,
    0xD2,
    0x43,
    0x36,
    0x36,
    0x03,
    0xD2,
    0x16,
    0x93,
    0x36,
    0x26,
    0xD2,
    0x93,
    0x73,
    0x13,
    0x66,
    0x56,
    0x36,
    0x33,
    0x33,
    0x83,
    0x56,
    0x23,
    0x66,
    0xD7,
)

# 解密数据
decoded_data = decode(encdata)
print("Decoded data:", decoded_data)

# Decoded data: b'0xGame{68dff458-29e5-4cc0-a9cb-971fec338e2f}'
```

FirstSight-Jar

先丢到在线jar反编译网站上，得到：

```
import java.util.Scanner;

public class EzJar {
   static String Alphabat = "0123456789abcdef";

   private static String encrypt(String var0) {
      StringBuilder var1 = new StringBuilder();

      for(int var2 = 0; var2 < var0.length(); ++var2) {
         int var3 = Alphabat.indexOf(var0.charAt(var2));
         if (var3 < 0) {
            var1.append(var0.charAt(var2));
         } else {
            var3 = (var3 * 5 + 3) % 16;
```

```
            var1.append(Alphabat.charAt(var3));
         }
      }

      return var1.toString();
   }

   public static void main(String[] var0) {
      System.out.println("请以uuid的格式输入你的flag内容:");
      Scanner var1 = new Scanner(System.in);
      String var2 = var1.nextLine();
      var1.close();
      if (encrypt(var2).equals("ab50e920-4a97-70d1-b646-cdac5c873376")) {
         System.out.println(String.format("验证成功: 0xGame{%s}", var2));
      } else {
         System.out.println("验证失败");
      }

   }
}
```

编写解密脚本

```python
def decrypt(encrypted_str):
    alphabat = "0123456789abcdef"
    decrypted_str = ""

    for char in encrypted_str:
        if char in alphabat:
            # 找到字符在alphabat中的索引
            index = alphabat.index(char)
            # 逆向计算原始索引
            original_index = (index - 3) * 13 % 16
            decrypted_str += alphabat[original_index]
        else:
            # 如果字符不是十六进制字符，直接追加
            decrypted_str += char

    return decrypted_str


# 示例加密字符串
encrypted_example = "ab50e920-4a97-70d1-b646-cdac5c873376"

# 解密示例
decrypted = decrypt(encrypted_example)
print(f"0xGame{{{decrypted}}}")
# 0xGame{b8a9fe39-dbe4-4926-87d7-52b5a5140047}
```

FisrtSight-Pyc

先丢到在线python反编译网站上，AI修复后，得到：

```
#!/usr/bin/env python
# visit https://tool.lu/pyc/  for more information
# Version: Python 3.8

import hashlib
user_input = input("请输入神秘代号：")
if user_input != "Ciallo~":
    print("代号不是这个哦")
    exit()

input_hash = hashlib.md5(user_input.encode()).hexdigest()
input_hash = list(input_hash)

# 修复后的循环
for i in range(len(input_hash)):
    if input_hash[i].isdigit():  # 检查是否是数字
        original_num = int(input_hash[i])
        new_num = (original_num + 5) % 10
        input_hash[i] = str(new_num)

# 将列表转换回字符串
input_hash = "".join(input_hash)

# 输出结果
print("0xGame{{{}}}".format(input_hash))
```

修改代码，

```
# user_input = input("请输入神秘代号：")
# if user_input != "Ciallo~":
#     print("代号不是这个哦")
#     exit()

user_input = "Ciallo~"
```

运行得到flag：`0xGame{2f0ef0217bf3a7c598d381b077672e09}`

Xor::Ramdom

首先读码可知flag长度为38，内容30，random应当取以0x77u为种子的第二个随机数123

```
#include <iostream>
#include <string>
#include <random>
#include <array>
using namespace std;

int main(int argc, char const *argv[])
{
    unsigned char a;
    srand(0x77u);
    a = rand();
    a = rand();
    printf("%d\n", a); // 输出随机数123
```

```
    srand(0x1919810); // 设定随机数种子
    a = rand();
    a = rand();
    printf("%d\n", a); // 输出随机数204
    return 0;
}
```

可以看出flag应当从v13, v14, v15, v16还原

不难发现这4个整数不算0的话恰好有30个字节，那就把他们转成字节解密

```
def int64_to_bytes_list(*ints):
    # 将每个64位整数转换为8字节，然后合并它们
    merged_bytes = b"".join(x.to_bytes(8, "big") for x in ints)
    # 转换为字节列表，并去除零字节
    byte_list = [byte for byte in merged_bytes if byte != 0]
    return byte_list


# 示例整数
v13 = 1306349891698577164
v14 = 1666090013777994827
v15 = 1885983640899768073
v16 = 99338668810000

# 转换并打印结果
# byte_list = int64_to_bytes_list(v13, v14, v15, v16)
byte_list = int64_to_bytes_list(v16, v15, v14, v13)
byte_list.reverse()
flag_list = []

# 解密过程
randomValue = 123  # 123, 204
for i in range(30):
    if i % 2 == 1:
        xorValue = byte_list[i] ^ randomValue
    else:
        xorValue = byte_list[i] ^ (randomValue + 3)
    flag_list.append(chr(xorValue))

flag_list = "".join(flag_list)
print(f"0xGame{{{flag_list}}}")
```

先是直接拼接v13, v14, v15, v16，再尝试反转，得到了有点奇怪的flag：

```
0xGame{ndom'!w4ys_'Ra5_n0t_alr4nd0m_i}
```

于是尝试拼接v16, v15, v14, v13，再反转，得到flag：

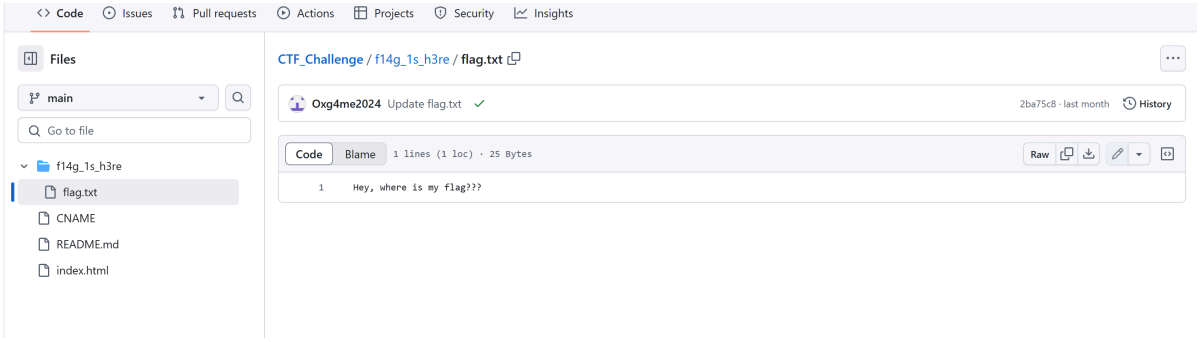
```
0xGame{r4nd0m_i5_n0t_alw4ys_'Random'!}
```

OSINT

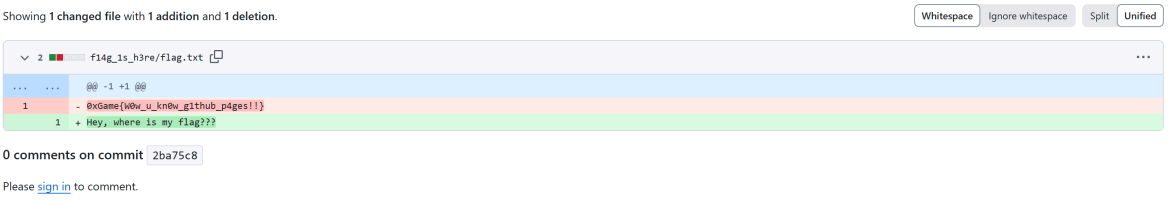
互联网的一角

提供的网页只有一条提示：这个网址好像指向了另一个网址.....

既然是OSINT，那就用搜索引擎查一下找到GitHub



哈哈，原来是上传了再改掉



给我干哪来了，这还是国内吗？？

注意到文字：SUPER ST-BERNARD

在Google地图上通过文字找到地址：Bourg-Saint-Bernard 2, 1946 Bourg-Saint-Pierre, 瑞士

搜索地址的 国家_州_区_城市 得到flag：`OxGame{Switzerland_Valais_Entremont_Bourg-Saint-Pierre}`