Misc

Happy 1024!

AI搜索关键词

> 根据搜索结果，同时包含“star”、“boat”、“dream”、“water”、“sky”这些元素的诗词是元代诗人唐珙的《题龙阳县青草湖》中的名句：“醉后不知天在水，满船清梦压星河。”
>
> 这句诗描绘了诗人醉卧扁舟，仰望星空，感受着梦境与现实交融的奇妙体验。诗中的“天在水”指的是天上的银河映照在水中，而“满船清梦压星河”则形容了诗人的梦境如同满载的船只，沉沉地压在了璀璨的星河之上。这句诗通过对自然景象的描绘，表达了诗人对梦境的留恋以及对现实的失意与失望。

Crypto

ECC-DH

共享密钥等于其中一方的私钥乘以另一方的公钥

```
a = 10809567548006703521
b = 9981694937346749887
p = 25321837821840919771
E = Curve(a, b, p)

findProof.main()
G_x = int(input("[+] Give me the G_x\n> "))
G_y = int(input("[+] Give me the G_y\n> "))
G = Point(G_x, G_y, E)
print(f"[+] Share_G : {G}")

b = randint(1, E.p)  # Bob's private key
B = b * G  # Bob's public key
print(f"[+] Bob_PubKey : {B}")

A_x = int(input("[+] Give me the Alice_PubKey.x\n> "))
A_y = int(input("[+] Give me the Alice_PubKey.y\n> "))
A = Point(A_x, A_y, E)  # Alice's Public Key
print(f"[+] Alice_PubKey : {A}")

Share_Key = b * A
Cipher = AES.new(MD5(Share_Key.x), AES.MODE_ECB)
pt = Cipher.decrypt(bytes.fromhex(input("[+] Give me the ciphertext\n> ")))
print(f"[+] Bob get the flag : {pt.decode()}")
```

```
┌──(root㉿Spreng)-[~]
└─# nc 118.195.138.159 10004
[+] sha256(XXXX+htC8hwrveKHEj447) ==
ffd95f4724211f4be013a17cad0d77c7914cd1cbc0f5200aa57f3bbc1d3de00c
[+] Plz tell me XXXX: DLrg
[+] Share G : (22565311448306005908,1397916078140553774)
[+] Alice_PubKey : (9250488053827849960,15122360881016255980)
[+] Give me the Bob_PubKey.x
> 10333552860092078375
[+] Bob_PubKey : (10333552860092078375,11655164729158644779)
[+] Alice tell Bob :
c06e8014b5e95178c688f177d0c28194a742fb67c042a34a9c08ece27e2670780ed81630a3cecaf3d
f343eeea32d2116
```

```
请输入proof: htC8hwrveKHEj447
请输入target_hash:
ffd95f4724211f4be013a17cad0d77c7914cd1cbc0f5200aa57f3bbc1d3de00c[+]
sha256(XXXX+htC8hwrveKHEj447) ==
ffd95f4724211f4be013a17cad0d77c7914cd1cbc0f5200aa57f3bbc1d3de00c
找到的XXXX是: DLrg
[+] Give me the G_x
> 22565311448306005908
[+] Give me the G_y
> 1397916078140553774
[+] Share_G : (22565311448306005908,1397916078140553774)
[+] Bob_PubKey : (10333552860092078375,11655164729158644779)
[+] Give me the Alice_PubKey.x
> 9250488053827849960
[+] Give me the Alice_PubKey.y
> 15122360881016255980
[+] Alice_PubKey : (9250488053827849960,15122360881016255980)
[+] Give me the ciphertext
>
c06e8014b5e95178c688f177d0c28194a742fb67c042a34a9c08ece27e2670780ed81630a3cecaf3d
f343eeea32d2116
[+] Bob get the flag : 0xGame{71234da9-baf8-406e-9cc7-d08ceedea945}
```

ECC-baby

用sage计算

```
from sage.all import *

p = 4559252311
a = 1750153947
b = 3464736227
E = EllipticCurve(GF(p),[a,b])
G = E(2909007728, 1842489211)
P = E(1923527223, 2181389961)
G_= E(1349689070,1217312018)
C = E(662346568,2640798701)

n = E.order()
```

```
factors = factor(n)
print(factors)  # 2^2 * 1139828071

result = []
factors = [4, 1139828071] #
for f1 in factors:
    t = n //f1
    res = discrete_log(t*P,t*G,operation='+')
    result  += [res]

print(result)  # [3, 530591416]
k = crt(result,factors)
print(k)  # 1670419487

k = 1670419487
M = C - G_ * k
print(M)  # (944662661 : 635214115 : 1)
```

```
from Crypto.Cipher import AES
from hashlib import md5


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


k = 1670419487
M_x = 944662661  # M = C - G_ * k = (944662661 : 635214115 : 1)
Cipher = AES.new(MD5(M_x), AES.MODE_ECB)
enc =
"29bb47e013bd91760b9750f90630d8ef82130596d56121dc101c631dd5d88201a41eb3baa5aa958a
6cd082298fc18418"
enc = bytes.fromhex(enc)
flag = Cipher.decrypt(enc)
print(flag.decode())  # 0xGame{0b0e28c2-b36d-d745-c0be-fcf0986f316a}
```

EzLogin-I

使用CBC字节翻转攻击：规定密文为C（已知），解密后的密文为B（未知），原来的明文为P（已知），他们满足 $IV \oplus B = P$，现在可以篡改IV，$IV' = P \oplus IV \oplus P'$ 使得 $IV' \oplus B = P \oplus IV \oplus P' \oplus B = P'$，这样就能篡改cookie。

```
#!/usr/local/bin/python
from base64 import b64encode, b64decode
from datetime import datetime
import json


def pad(data: bytes):
    l = 16 - len(data) % 16  # PCKS7 padding length
    return data + bytes([l] * l)
```

```python
def main():
    data = input("[+] Enter your cookie:\n>").strip()
    data = b64decode(data.encode())
    IV, C = data[:16], data[16:]  # 已知密文
    # {"username": "A
    # mqP7rMoHp/NewIwl
AOhBvwHRXLTEl9MYRwo4sDjhYSTIB/xRv835m+T4Jc19BYU5SgZRvtronSEn4BA+rYCPRA==

    now = datetime.now()
    t = int(datetime.timestamp(now))
    for i in range(5, 15):
        time = t - i
        cookie = {}
        cookie["username"] = "admin"
        cookie["time"] = time
        P_ = pad(json.dumps(cookie).encode())  # 目标明文

        cookie = {}
        cookie["username"] = "Admin"
        cookie["time"] = time
        P = pad(json.dumps(cookie).encode())  # 已知明文
        print(forge(P, P_, C, IV))


def forge(P, P_, C, IV):
    P = bytearray(P)
    P_ = bytearray(P_)
    IV = bytearray(IV)
    IV[14] = P_[14] ^ IV[14] ^ P[14]
    IV = bytes(IV)
    return b64encode(IV + C).decode()


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

```
┌──(root㉿Spreng)-[~]
└─# nc 118.195.138.159 10005
+--------------+
| [R] Regist   |
| [L] Login    |
| [F] Getflag  |
+--------------+

[+] Tell me your choice:
>R
[+] username:
>Admin
[+] cookie :
O1yW3bpyePzOQVVAlHCrL7hO3jLlKV/BxQa6j5F36V8X1r6Ugng7FvRinuy2km8xGGanhUQLLm/2/AW6s
hCFjg==
[+] Tell me your choice:
>L
```

```
[+] cookie:
>O1yW3bpyePzOQVVAlHCLL7hO3jLlKV/BxQa6j5F36V8X1r6Ugng7FvRinuy2km8xGGanhUQLLm/2/AW6
shCFjg==
[+] Here is flag1 : b'0xGame{ad34acff-a813-4bc3-a44a-c270edf244b7}'
[+] Tell me your choice:
>
```

```
[+] Enter your cookie:
>
O1yW3bpyePzOQVVAlHCrL7hO3jLlKV/BxQa6j5F36V8X1r6Ugng7FvRinuy2km8xGGanhUQLLm/2/AW6s
hCFjg==
O1yW3bpyePzOQVVAlHCLL7hO3jLlKV/BxQa6j5F36V8X1r6Ugng7FvRinuy2km8xGGanhUQLLm/2/AW6s
hCFjg==
```

EzLogin-II

PaddingOracleAttack：与CBC翻转攻击类似，依然是$IV \oplus B = P$，只不过这次的P是未知的，由于unpad过程有报错提示，可以据此判断P的情况。对B的对应字节进行枚举，例如枚举最后一个字节，当$P' = 0x00$或$0x01$时，提示JSON Wrong，否则提示Unkown Wrong，一般希望得到的是$P' = 0x01$，这样就可以反推出B和P。枚举其他字节时，利用前面字节确定的B就可以定向控制P。

```python
#!/usr/local/bin/python
from base64 import b64encode, b64decode
from Crypto.Cipher import AES
from datetime import datetime
from os import urandom
import json
from time import sleep
from pwn import *


KEY = urandom(16)
flag2 = "0xGame{cOngr4t1ng_w1th_cOd3_4r3_1n_th3_wOrld}"  # 32 <= length < 48
ip = "118.195.138.159"
port = 10005
io = remote(ip, port)


def pad(data: bytes):
    l = 16 - len(data) % 16  # PCKS7 padding length
    return data + bytes([l] * l)


def unpad(data: bytes):
    for i in range(1, data[-1] + 1):
        if data[-1] != data[-i]:
            print("Unpad error")
    return data[: -data[-1]]


def encrypt(data):
    IV = urandom(16)
    ENC = AES.new(KEY, AES.MODE_CBC, IV)
    result = ENC.encrypt(pad(data.encode()))
```

```
    return b64encode(IV + result).decode()


def decrypt(data):
    data = b64decode(data)
    IV, C = data[:16], data[16:]
    DEC = AES.new(KEY, AES.MODE_CBC, IV)
    result = DEC.decrypt(C)
    return unpad(result).decode()


def login(data):
    data = data.encode()
    data = b64decode(data)
    IV, C = data[:16], data[16:]
    DEC = AES.new(KEY, AES.MODE_CBC, IV)
    data = DEC.decrypt(C)
    if data[-1] > 16:
        return False
    for i in range(1, data[-1] + 1):
        if data[-1] != data[-i]:
            return False
    return True


def main():
    # data = input("[+] Enter your cookie:\n>").strip()

    # 输出F
    io.sendlineafter(b"Tell me your choice:\n>", b"F")
    # 接收flag2
    data = io.recvuntil(b"\n", drop=True).decode().strip()
    data = data.split("Here is flag2 :")[1].strip()
    print(f"[+] Here is flag2: {data}")
    data = b64decode(data.encode())
    # 输出L
    io.sendlineafter(b"Tell me your choice:\n>", b"L")

    P = bytearray(0)
    for i in range(1, 3):
        P += explode(data, i)
        print(P)
    P = bytes(P).decode()
    print(P)  # 0xGame{6e02937e-634d-4f6f-8ef6-e5f3870O6cde}


def explode(data: bytes, block_num: int) -> bytearray:
    P = bytearray(16)  # 记录明文
    B = bytearray(16)  # 记录初步解密的块
    for i in range(1, 17):
        print("            ", end="\r")
        print(round(100 * (i - 1 + block_num * 16) / 48, 2), "%", end="\r")
        for byte in range(256):
            IV_ = bytearray(16)
            if block_num == 0:
                IV = data[:16]
```

```
                for j in range(16):
                    IV_[j] = IV[j]
                for j in range(1, i):
                    IV_[-j] = B[-j] ^ i
                IV_[-i] = byte
                IV_ = bytes(IV_)
                data_ = IV_ + data[16:32]
            elif block_num == 1:
                IV = data[16:32]
                for j in range(16):
                    IV_[j] = IV[j]
                for j in range(1, i):
                    IV_[-j] = B[-j] ^ i
                IV_[-i] = byte
                IV_ = bytes(IV_)
                data_ = IV_ + data[32:48]
            elif block_num == 2:
                IV = data[32:48]
                for j in range(16):
                    IV_[j] = IV[j]
                for j in range(1, i):
                    IV_[-j] = B[-j] ^ i
                IV_[-i] = byte
                IV_ = bytes(IV_)
                data_ = IV_ + data[48:64]
            else:
                return None
            if put(b64encode(data_)):
                B[-i] = IV_[-i] ^ i
                P[-i] = IV[-i] ^ B[-i]
                break
    return P


def put(try_data):
    # 输出try_data
    io.sendlineafter(b"cookie:\n>", try_data)
    # 接受信息
    response = io.recvline()
    if b"Unkown Wrong" in response:
        return False
    else:
        print(response)
        return True

    # if login(try_data):
    #     return True
    # else:
    #     return False


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

LLL-I

先用LLL还原，正交矩阵会被约掉

```
mt = matrix(ZZ, 0, 4)
mt = mt.stack(
    vector(
        [
            184978470348295101286515226402567457 5,
            266484808595592575435011776767362793 2,
            209978352739652015161027418059085416 6,
            102055859557730161710811192054580452 7,
        ]
    )
)
mt = mt.stack(
    vector(
        [
            120744956681112161402033402019580237 2,
            195462197699911287866115090367354323 2,
            132605040673153420157494369068823733 8,
            136181320809422744576811159195901196 3,
        ]
    )
)
mt = mt.stack(
    vector(
        [
            888810907577479776819993141014777624,
            121630273680792824087587442776534064 5,
            102735943742159906959932771287371956 7,
            238961447144792739830554790892164336,
        ]
    )
)
mt = mt.stack(
    vector(
        [
            606221645179409430372743869122 82,
            829585081387551685768360127174 68,
            700721180668268565643296276508 28,
            162967408621425077453222422353 26,
        ]
    )
)
print(mt.LLL()[0])
```

还原flag：

```
# 假设 Length 是 8
Length = 8

# 还原 flag
flag = b""
for noise in Noise[0]:
    flag += long_to_bytes(noise)

# 将字节字符串转换为 ASCII 字符串
flag = flag.decode("latin-1")  # 或者使用其他适当的编码

print(flag)  # 0xGame{04679c42-2bc1-42b2-b836-1b0ca542f36b}
```

LLL-II

先推公式：

$$\begin{aligned} C_i &= aC_{i-1} + b \mod m \\ km - aC_{i-1} + C_i &= b \mod m \end{aligned}$$

构造矩阵：

$$\begin{bmatrix} k_1 & k_2 & k_3 & k_4 & -a & 1 \end{bmatrix} \begin{bmatrix} m & 0 & 0 & 0 & 0 & 0 \\ 0 & m & 0 & 0 & 0 & 0 \\ 0 & 0 & m & 0 & 0 & 0 \\ 0 & 0 & 0 & m & 0 & 0 \\ C_0 & C_1 & C_2 & C_3 & -K/m & 0 \\ C_1 & C_2 & C_3 & C_4 & 0 & K \end{bmatrix} = \begin{bmatrix} b_1 & b_2 & b_3 & b_4 & aK/m & K \end{bmatrix}$$

对第二个矩阵进行LLL即可

```
Cs = [
11804527453299586684489593808016317337345238230165321056832279785591503368758306671170625597063579251464905729051049524014502008954170088604924368057540940,
49309228843064865707596612886025574286083155588049505374701002630192288888174816170654547058431648095068595740538842061333445498958530647353613364865609 81,
53802638564461654495316471112600105946204167309325390977823995576034206583504070803661324901740604205307082935642528526684319235608826486913924465211884 65,
10746696290782998433216934286282230556131938525513632178308443345441147075710552571129957873399395862207656161609046567289600084193860244770966610161184627,
21950329575118309925589610215669048502787967373162385665138379952973942156382599169440876239236367893121347349494528395617651714462175200814027699625171 10
]
m =
12813864523019740432913161815051292412705285817864701047922722497269479288096574264414061282833203433542813637861620032851255308640850882149603687035724753

# A = Matrix([k1, k2, k3, k4, -a, 1])
# B = Matrix([b1, b2, b3, b4, a*K/m, K])
K = 2**128
M = Matrix([[m, 0, 0, 0, 0, 0],
            [0, m, 0, 0, 0, 0],
            [0, 0, m, 0, 0, 0],
            [0, 0, 0, m, 0, 0],
            [Cs[0], Cs[1], Cs[2], Cs[3], -K/m, 0],
```

```
            [Cs[1], Cs[2], Cs[3], Cs[4], 0, K]])
L = M.LLL()
print(L[1,-2] / K * m % m)
```

```
from Crypto.Util.number import getPrime, inverse
from hashlib import md5


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


m = 
128138645230197404329131618150512924127052858178647010479227224972694792880965742
644140612828332034335428136378616200328512553086408508821496036870357247 53
cur = 
118045274532995866844895938080163173373452382301653210568322797855915033687583066
711706255970635792514649057290510495240145020089541700886049243680575409 40
a = 
882286556556438929937814021762877259066481571322848505325476923637252821175658481
318279121105678856729786660242195409798 56
a = 
105335058376849464581926358411480063090049833236632732289586293246093519574939
print(a.bit_length())  # 256
seed = (cur * inverse(a, m)) % m
print(seed.bit_length())  # 510
print("0xGame{" + MD5(seed) + "}")  # 0xGame{2db84757dd4197f9b9441be25f35bfd5}
```

LLL-III

```
import itertools
def small_roots(f, bounds, m=1, d=None):
    if not d:
        d = f.degree()
    R = f.base_ring()
    N = R.cardinality()
    f /= f.coefficients().pop(0)
    f = f.change_ring(ZZ)
    G = Sequence([], f.parent())
    for i in range(m+1):
        base = N^(m-i) * f^i
        for shifts in itertools.product(range(d), repeat=f.nvariables()):
            g = base * prod(map(power, f.variables(), shifts))
            G.append(g)
    B, monomials = G.coefficient_matrix()
    monomials = vector(monomials)
    factors = [monomial(*bounds) for monomial in monomials]
    for i, factor in enumerate(factors):
        B.rescale_col(i, factor)
    B = B.dense_matrix().LLL()
    B = B.change_ring(QQ)
    for i, factor in enumerate(factors):
        B.rescale_col(i, 1/factor)
    H = Sequence([], f.parent().change_ring(QQ))
```

```
    for h in filter(None, B*monomials):
        H.append(h)
        I = H.ideal()
        if I.dimension() == -1:
            H.pop()
        elif I.dimension() == 0:
            roots = []
            for root in I.variety(ring=ZZ):
                root = tuple(R(root[var]) for var in f.variables())
                roots.append(root)
            return roots
    return []
```

```
# sage

n =
18126197502749523725363749082196797483810742900167355566427847172100838628174 3
a = 80470362380817459255864867107210711412685230469402969278321951982944620399953
b =
10831975937023678381462643300076672111133457058687360770832279051224010419035 1
output = [
    24661921912602137757626239650679579442410 15,
    18898927854396545717421213359957986329919 77,
    19965044065636422404539713590311300599822 31,
    13683011212558300772015891285705287352297 41,
    39993158550359852690592825183655814281616 59,
    34903289208895541197809449520823094970519 42,
    27027347063054396816727023360418793919210 64,
    23262045811090896463364784710736935772065 07,
    34289949642897082227512941057262310923939 19,
    13235080228330046399969546426845212661849 99,
    22085337700638299894019557570647841651786 29,
    14777505881643117377824309294244167354364 45,
    97345909871249550543027002059743782912631 3,
    18490381403021902873896645318135959447253 51,
    11727970632620267991635739553157389646052 14,
    17541021366348635870481915049982763609273 39,
    11348830105288048737084048636193370257970 4,
    28627689388588873044616163624624480559406 70,
    36259579060563117125944399631347394239337 12,
    39220856958882263898563459596344716083106 38,
]

#
PR.<x,y> = PolynomialRing(Zmod(n))
f = ((output[0]<<115)+ x) * a + b - ((output[1]<<115) + y)
roots = small_roots(f,(2^115, 2^115), m=4, d=4)
s1 = (output[0]<<115) + roots[0][0]
m = (s1 - b) * inverse_mod(a, n) % n
print(m)  #
10163961305054487229219262951527356203502269978844534485845515766884082897336 1
```

```
from hashlib import md5


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


seed = 10163961305054487229219262951527356203502269978844534485845515766884082897336 1
print("0xGame{" + MD5(seed) + "}")  # 0xGame{459049e068d93f6d70f1ea0da705264a}
```

Reverse

BabyASM

先化简原文

```
data = [···]

def printFLAG():
    printf("%s\n", data)

main:
    index = 0
    goto    L3
L4:
    eax = data[index]
    eax += 28
    data[index] = eax
    index += 1

L3:
    if index<= 21 goto L4

L5:
    edx = data[index]
    eax = data[index-22]
    edx ^= eax
    data[index-22] = edx
    add index, 1
    if index <= 42 goto L5
    printFLAG
```

转为python代码就是:

```
data = [
    20,
    92,
    43,
    69,
```

```
    81,
    73,
    95,
    23,
    72,
    22,
    24,
    69,
    25,
    27,
    22,
    17,
    23,
    29,
    24,
    73,
    17,
    24,
    85,
    27,
    112,
    76,
    15,
    92,
    24,
    1,
    73,
    84,
    13,
    81,
    12,
    0,
    84,
    73,
    82,
    8,
    82,
    81,
    76,
    125,
]  # 44


def printf(format, *args):
    print(format % args)  # 简化的printf实现，实际printf更复杂


def printFLAG(flag_bytes):
    printf("%s", bytes(flag_bytes))


def main():
    # 第一个循环：将每个字节增加28
    for i in range(22):
        data[i] += 28
```

```
    printFLAG(data)  # 打印处理后的data数组

    # 第二个循环：对data数组进行异或操作
    for i in range(22, 43):
        data[i] ^= data[i - 22]

    printFLAG(data)  # 打印处理后的data数组
    # 0xGame{3d24a572-394e-aec7-b9c2-f9097fda1f4L}


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

LittlePuzzle

在线反编译jar，转python

```
def exit():
    print("解谜失败")
    exit(1)


def check(board, x, y):
    t = board[x][y]
    x_ = x - x % 3
    y_ = y - y % 3

    for i in range(9):
        if i != x and i != y and (t == board[x][i] or t == board[i][y]):
            return False

    for i in range(3):
        for j in range(3):
            if x_ + i != x and y_ + j != y and t == board[x_ + i][y_ + j]:
                return False

    return True


def flag(answer):
    s = []

    for i in range(0, len(answer), 6):
        var3 = int(answer[i : i + 6])
        s.append(hex(var3)[2:])

    return "".join(s)


def main():
    board = [
        [5, 7, 0, 9, 4, 0, 8, 0, 0],
        [0, 0, 8, 0, 3, 0, 0, 0, 5],
        [0, 1, 0, 2, 0, 0, 0, 3, 7],
```

```
        [0, 0, 9, 7, 2, 0, 0, 0, 0],
        [7, 3, 4, 0, 0, 8, 0, 0, 0],
        [0, 0, 0, 0, 0, 0, 7, 5, 1],
        [3, 0, 0, 0, 1, 4, 2, 0, 0],
        [0, 6, 0, 0, 0, 2, 0, 4, 0],
        [0, 2, 7, 0, 0, 9, 5, 0, 0],
    ]

    # board = [
    #    [5, 7, 3| 9, 4, 1| 8, 6, 2],
    #          0        0     0  0
    #    [2, 4, 8| 6, 3, 7| 1, 9, 5],
    #     0  0     0     0  0  0
    #    [9, 1, 6| 2, 8, 5| 4, 3, 7],
    #     0     0     0  0  0
    #    [1, 5, 9| 7, 2, 6| 3, 8, 4],
    #     0  0           0  0  0  0
    #    [7, 3, 4| 1, 5, 8| 6, 2, 9],
    #              0  0     0  0  0
    #    [6, 8, 2| 4, 9, 3| 7, 5, 1],
    #     0  0  0  0  0  0
    #    [3, 9, 5| 8, 1, 4| 2, 7, 6],
    #        0  0  0           0  0
    #    [8, 6, 1| 5, 7, 2| 9, 4, 3],
    #     0     0  0  0     0     0
    #    [4, 2, 7| 3, 6, 9| 5, 1, 8]
    #     0        0  0        0  0
    #     3162 246719 96854 156384 15629 682493 95876 815793 43618
    #     4+6+5+6+5+6+5+6+5= 48
    # ]

    print("请输入你的解谜结果:")
    answer = "316224671996854156384156296824939587681579343618"
    if len(answer) != 48:
        exit()

    var3 = 0

    for i in range(9):
        for j in range(9):
            if board[i][j] == 0:
                var6 = int(answer[var3])
                if 1 <= var6 <= 9:  # ASCII调整，'0'字符的ASCII码是48，所以需要减去48
                    board[i][j] = var6
                    var3 += 1
                else:
                    exit()

    for i in range(9):
        for j in range(9):
            if not check(board, i, j):
                exit()

    print(f"0xGame{{{flag(answer)}}}")  #
0xGame{4d340a40fcd088c5dc9c48778e5643a666b53e42}
```

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

    # board = [
    #     [5, 7, 3, 9, 4, 1, 8, 6, 2],
    #     [2, 4, 8, 6, 3, 7, 1, 9, 5],
    #     [9, 1, 6, 2, 8, 5, 4, 3, 7],
    #     [1, 5, 9, 7, 2, 6, 3, 8, 4],
    #     [7, 3, 4, 1, 5, 8, 6, 2, 9],
    #     [6, 8, 2, 4, 9, 3, 7, 5, 1],
    #     [3, 9, 5, 8, 1, 4, 2, 7, 6],
    #     [8, 6, 1, 5, 7, 2, 9, 4, 3],
    #     [4, 2, 7, 3, 6, 9, 5, 1, 8],
    # ]
    # for i in range(9):
    #     for j in range(9):
    #         if not check(board, i, j):
    #             print("解谜失败")
```

只要把数独给解了，把填入的数组合在一起运行一下就出来了

Tea

由于C和python的位运算符不太一样，而且程序本来就是C写的，脚本就用C了

tea的解密只要把程序倒过来抄一遍就可以了

```
#include <stdio.h>
#include <stdint.h>

void preProcess(char *a1, int len)
{
    char *i = &a1[len - 1];
    char v5;

    while (1)
    {
        if (a1 >= i)
            break;
        v5 = *a1;
        *a1 = *i;
        *i = v5;
        ++a1;
        --i;
    }
}

void tea_dec(uint32_t *data, uint32_t *k)
{
    uint32_t v3 = data[0];
    uint32_t v4 = data[1];
    int v5 = -1640531527 * 32;
    for (int i = 0; i < 32; ++i)
    {
```

```
        v4 -= (v3 + v5) ^ ((v3 >> 5) + k[3]) ^ (k[2] + 16 * v3);
        v3 -= (v4 + v5) ^ ((v4 >> 5) + k[1]) ^ (*k + 16 * v4);
        v5 += 1640531527;
    }
    data[0] = v3;
    data[1] = v4;
}

int main()
{
    char data[] = {
        0xC9, 0xB6, 0x5C, 0xCE, 0xF8, 0xEE, 0x8E, 0xA2, 0x33, 0x36,
        0x34, 0x63, 0x37, 0x32, 0x36, 0x64, 0x38, 0x37, 0x65, 0x33,
        0x62, 0x33, 0x63, 0x64, 0x36, 0x39, 0x64, 0x34, 0x64, 0x30,
        0x62, 0x38, 0x2A, 0x7A, 0x7C, 0x3B, 0x85, 0x33, 0x6D, 0xD3};

    char k[] = {
        0x02, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
        0x02, 0x00, 0x00, 0x00, 0x04, 0x00, 0x00, 0x00};

    preProcess(&data, 40);
    for (int i = 0; i < 5; i++)
    {
        tea_dec((uint32_t *)(__int64)&data[32 * i], (uint32_t *)(__int64)&k);
    }
    printf("%s\n", data);  // 0xGame{e8b0d4d96dc3b3e78d627c463c9953a1}

    return 0;
}
```

The Matrix

这道题读程比较麻烦，hello就是给矩阵赋值的，前几位保存行数列数，代码中只用到了3阶矩阵

matmul就是矩阵的乘法

加密过程就是用输入做出7个矩阵，其中存在数据的复用。然后再轮流左乘k对应的4个矩阵，与data对比。

解密过程也很简单，data左乘k的逆矩阵就是flag。

```
# 3阶矩阵赋值
def hello(source: list) -> list:
    return [source[3 * i : 3 * i + 3] for i in range(3)]


# 矩阵乘法
def matmul(a: list[list], b: list[list]) -> list[list]:
    c = [[0, 0, 0], [0, 0, 0], [0, 0, 0]]
    for i in range(3):
        for j in range(3):
            for k in range(3):
                c[i][j] += a[i][k] * b[k][j]
    return c
```

```
# 64
dst = [
    488,
    448,
    385,
    1367,
    1235,
    1054,
    317,
    273,
    258,
    620,
    320,
    325,
    1463,
    748,
    755,
    1513,
    797,
    822,
    333,
    266,
    402,
    189,
    159,
    245,
    189,
    161,
    257,
    354,
    327,
    547,
    251,
    192,
    294,
    401,
    291,
    439,
    240,
    253,
    269,
    670,
    704,
    753,
    145,
    159,
    164,
    553,
    315,
    302,
    1252,
    728,
    711,
    1469,
    805,
    740,
```

```
    455,
    337,
    213,
    254,
    229,
    110,
    300,
    160,
    158,
    0,
]

k_ = [
    [[7, -2, -2], [3, -1, 0], [-6, 2, 1]],
    [[-26, 8, 3], [24, -7, -3], [-7, 2, 1]],
    [[2, -2, -1], [-1, 1, 1], [2, -1, -2]],
    [[-1, 5, -2], [1, -6, 3], [-1, 8, -4]],
]


def main():

    for i in range(7):
        dst_ = hello(dst[9 * i : 9 * i + 9])
        memory_ = matmul(k_[i % 4], dst_)
        for j in range(3):
            for k in range(3):
                print(chr(memory_[j][k]), end="")
        print("\b\b\b", end="")


if __name__ == "__main__":
    main()
# 0xGame{78d51c59-6dc3-30d2-1276-18e13f80c478}
```