crypto 7.16

原创

PUTAOAO 于 2021-07-16 14:58:16 发布 37 收藏

分类专栏： 刷题记录

本文链接：https://blog.csdn.net/PUTAOAO/article/details/118784640

刷题记录 专栏收录该内容

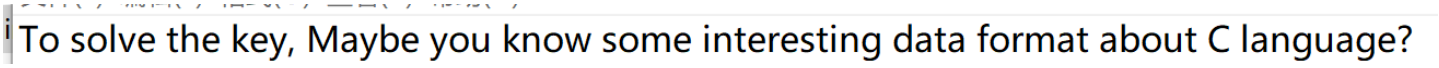
22 篇文章 0 订阅

订阅专栏

[ACTF新生赛2020]crypto-des

这道考点是des 和 数据在内存中的存储

刚开始就卡住了



c语言中有趣的数据结构

查了一下好像之前考到过数据在内存中的存储

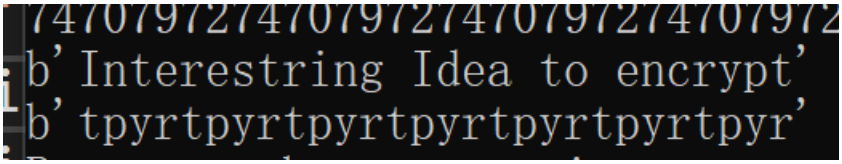
把数据转为内存中的存储 大佬脚本

```
from libnum import*
import struct
import binascii

s = [72143238992041641000000.000000,771353571780065040000000000000000.000000,112586834561643540000000.000000,673
78029765916820000000.000000,755534860921847030000000000.000000,439761191373995870000.000000,76209378028621039
000000000000000.000000]
a = ''
b = ''
for i in s:
    i = float(i)
    a += struct.pack('<f',i).hex()        #小端
print(a)

for j in s:
    i = float(i)
    b += struct.pack('>f',i).hex()        #小端
print(b)

a = 0x496e74657265737472696e67204964656120746f20656e6372797074
b = 0x74707972747079727470797274707972747079727470797274707972
print(n2s(a))
print(n2s(b))
```



解开压缩包

得到des加密脚本 而且已经有密钥了

直接base64 解密后再aes解密

```
import pyDes
import base64
deskey = "********"
DES = pyDes.des(deskey)
DES.setMode('ECB')
DES.Kn = [
   [1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 1, 0, 1, 1, 0, 0, 0, 0,
0, 1, 1, 1, 1, 0, 0, 1, 1, 0, 0, 0],
   [1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 1,
0, 1, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0],
   [0, 1, 1, 0, 0, 1, 0, 0, 1, 1, 0, 1, 0, 1, 1, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0,
1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0],
   [1, 1, 0, 0, 0, 1, 1, 0, 1, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 0,
1, 0, 0, 0, 1, 1, 0, 1, 0, 0, 1, 1],
   [0, 0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 1, 1, 0, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 0,
0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1],
   [0, 0, 1, 0, 1, 1, 1, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 0, 0, 1, 0,
0, 1, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0],
   [0, 0, 1, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 0, 1,
0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 1, 0],
   [0, 0, 0, 1, 1, 1, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0,
0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 1, 0],
   [1, 0, 0, 1, 1, 1, 0, 1, 0, 1, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1,
1, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0],
   [0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 1, 0,
0, 1, 0, 0, 1, 0, 1, 1, 1, 0, 0, 0],
   [0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 1, 1, 1, 0, 1, 0, 0, 1, 0, 0, 1, 1,
1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1],
   [0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 1, 1, 1, 0, 0, 1, 0, 1,
0, 0, 1, 0, 0, 0, 1, 1, 1, 1, 1, 0],
   [1, 1, 0, 1, 0, 0, 0, 1, 1, 0, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1,
1, 0, 0, 1, 1, 1, 1, 0, 0, 1, 0, 0],
   [1, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0,
1, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1],
   [1, 1, 1, 1, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0, 1, 1, 0, 0, 0, 0, 1, 1, 1, 0, 1, 0,
1, 1, 1, 0, 0, 0, 0, 1, 0, 1, 0, 1],
   [1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 1,
0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 1]
  ]

c=b'vrkgBqeK7+h7mPyWujP8r5FqH5yyVlqv0CXudqoNHVAVdNO8ML4lM4zgez7weQXo'
c=base64.b64decode(c)
flag=DES.decrypt(c)
print(flag)
```

数据在内存中的存储 要记下来

羊城杯 RRRRRRSA

e很大 就想到winner attack
但是用了一下不行
winner attack 需要满足e和n很接近
但是这道题不满足
但是n1和n2 却符合

```
n1/n2=(p1/p2)**2*(q1/q2)
n1/n2<q1/q2
```

所以q1/q2在(n1/n2,1)的区间中

```
import gmpy2
import sympy
from Crypto.Util.number import long_to_bytes
def transform(x,y):       #使用辗转相处将分数 x/y 转为连分数的形式
    res=[]
    while y:
        res.append(x//y)
        x,y=y,x%y
    return res

def continued_fraction(sub_res):
    numerator,denominator=1,0
    for i in sub_res[::-1]:      #从sublist的后面往前循环
        denominator,numerator=numerator,i*numerator+denominator
    return denominator,numerator   #得到渐进分数的分母和分子，并返回


#求解每个渐进分数
def sub_fraction(x,y):
    res=transform(x,y)
    res=list(map(continued_fraction,(res[0:i] for i in range(1,len(res)))))  #将连分数的结果逐一截取以求渐进分数
    return res

def get_pq(a,b,c):      #由p+q和pq的值通过维达定理来求解p和q
    par=gmpy2.isqrt(b*b-4*a*c)   #由上述可得，开根号一定是整数，因为有解
    x1,x2=(-b+par)//(2*a),(-b-par)//(2*a)
    return x1,x2

def wienerAttack(e,n):
    for (d,k) in sub_fraction(e,n):  #用一个for循环来注意试探e/n的连续函数的渐进分数，直到找到一个满足条件的渐进分数
        if k==0:                     #可能会出现连分数的第一个为0的情况，排除
            continue
        if (e*d-1)%k!=0:             #ed=1 (mod φ(n)) 因此如果找到了d的话，(ed-1)会整除φ(n),也就是存在k使得(e*d-1)//k
=φ(n)
            continue

        phi=(e*d-1)//k               #这个结果就是 φ(n)
        px,qy=get_pq(1,n-phi+1,n)
        if px*qy==n:
            p,q=abs(int(px)),abs(int(qy))      #可能会得到两个负数，负负得正未尝不会出现
            d=gmpy2.invert(e,(p-1)*(q-1))      #求ed=1 (mod  φ(n))的结果，也就是e关于 φ(n)的乘法逆元d
            return d
    print("该方法不适用")
```

```python
import gmpy2
def transform(x,y):       #使用辗转相处将分数 x/y 转为连分数的形式
    res=[]
    while y:
        res.append(x//y)
        x,y=y,x%y
    return res

def continued_fraction(sub_res):
    numerator,denominator=1,0
    for i in sub_res[::-1]:      #从sublist的后面往前循环
        denominator,numerator=numerator,i*numerator+denominator
    return denominator,numerator   #得到渐进分数的分母和分子，并返回


#求解每个渐进分数
def sub_fraction(x,y):
    res=transform(x,y)
    res=list(map(continued_fraction,(res[0:i] for i in range(1,len(res)))))  #将连分数的结果逐一截取以求渐进分数
    return res

def get_pq(a,b,c):      #由p+q和pq的值通过维达定理来求解p和q
    par=gmpy2.isqrt(b*b-4*a*c)   #由上述可得，开根号一定是整数，因为有解
    x1,x2=(-b+par)//(2*a),(-b-par)//(2*a)
    return x1,x2

def wienerAttack(e,n):
    for (d,k) in sub_fraction(e,n):  #用一个for循环来注意试探e/n的连续函数的渐进分数，直到找到一个满足条件的渐进分数
        if k==0:                     #可能会出现连分数的第一个为0的情况，排除
            continue
        if (n1%k)==0 and k!=1:             #ed=1 (mod φ(n)) 因此如果找到了d的话，(ed-1)会整除φ(n),也就是存在k使得(e*d
-1)//k=φ(n)
            return k
    print("该方法不适用")


n1=60143104944034567859993561862949071559877219267755259679749062284763163484947626697494729046430386559610613113754453726683312513915610558734802079868190554644983911078936369464590301246394586190666760362763580192139772729890492729488892169933099057105842090125200369295070365451134781912223048179092058016446222199742919885472867511334714233086339832790286482634562102936600597781342756061479024744312357407750731307860842457299116947352106025529309727703385914891200109853084742321655388368371397596144557614128458065859276522963419738435137978069417053712567764148183279165963454266011754149684758060746773409666706463583389316772088889398359242197165140562147489286818190852679930372669254697353483887004105934649944725189954685412228899457155711301864163839538810653626724347n1=60143104944034567859993561862949071559877219267755259679749062284763163484947626697494729046430386559610613113754453726683312513915610558734802079868190554644983911078936369464590301246394586190666760362763580192139772729890492729488892169933099057105842090125200369295070365451134781912223048179092058016446222199742919885472867511334714233086339832790286482634562102936600597781342756061479024744312357407750731307860842457299116947352106025529309727703385914891200109853084742321655388368371397596144557614128458065859276522963419738435137978069417053712567764148183279165963454266011754149684758060746773409666706463583389316772088889398359242197165140562147489286818190852679930372669254697353483887004105934649944725189954685412228899457155711301864163839538810653626724347c1=55094296873556883585060020895253176070835143350249581136609315815308788255684072804968957510292559743192424646169207794748893753882418256401223641287546922358162629295622258913168323493447075410872354874300793298956869374606043622559405978242734950156459436487837698668489891733875650048466360950142617732135781244969524095348835624828008115829566644654403962285001724209210887446203934276651265377137788183939798543755386888532680013170540716736655670269251318800501517579803401154996881233025210176293554542024052540093890387437964747460765498713092018160196637928204190194154199389276666685436565665236397481709703644555328705818892269499380797044554054118656321389474821224725533693520856047736578402581854165941599254178019515615183102894716647680969742744705218868455450832e1=125932919717342481428108392434488550259190856475011752106073050593074410065655587870702051419898088541590032
```

```
0985404803264962526985633790104840606696833728949195140438430046654361657867953980821569875449107634038669751894
8419895268049696498272031094236309803803729823608854215226233796069683774155739820423103
n2=60143104944034567859993561862949071559877219267755259679749062284763163484947626697494729046430386559610613113
3754453726683312513915610558734802079868195633647431732875392121458684331843306730889424418620069322578265236351
4075910293385198095389952498969051376423424356595729177141835433052437156643807877975620110063987303209809947479
3979156188562294991269824670176932143032590291200304167877444070405659786209353098104069687252286892113904124736
2592257285423948870944137019745161211585845927019259709501237550818918272189606436413992759328318871765171844153
5274243479854627670281353765523024638613244081781838421393302449066067763590504829772567289102786879961061529710
2887865312353355976016771127026517144162305687390366991869425904358001708167134923205187071649355743451757912 1
c2=3932844614015625757148418471386131972290586419755672073085277305914790228312325276765143027835795087262677834
8596897711320942449693270603776870301102881405303651558719085454281142395652056217241751656631812580544180434349
8402369197654331223891168608275937115937323855623282557595093552986623615086115319723869952399085132732362398588
5458684584968686536078029035028713909214358703739680170435169273698595515293560198775885975942188667090773512013
7698039900161327397951758852875291442188850946273771733011504922325622240838288097946309825051094566685479503461
9385023735209836842966589717009220694267882364765752361890401028484185476342902141751677674314750032160567010942
7589921141997934080271168498971013021592652638713853881953119981084147521814260669115292823636253418162220134 7
e2=1259329197173424814281083924344885502591908564750117521060730505930744100656555878707020514198980885415900322
0985404803264962526985633790104840606696833728949195140438430046654361657867953980821569875449107634038669751894
8419895268049696498272031094236309803803729823608854215226233796069683774155739820425393

q1=wienerAttack(n1,n2)
p1=gmpy2.iroot(n1//q1,2)[0]
p2 =sympy.nextprime(p1)
q2=sympy.nextprime(q1)
phi1=p1*(p1-1)*(q1-1)
phi2=p2*(p2-1)*(q2-1)
d1=gmpy2.invert(e1,phi1)
d2=gmpy2.invert(e2,phi2)
print(long_to_bytes(pow(c1,d1,n1)))
print(long_to_bytes(pow(c2,d2,n2)))
```

这里phi1和phi2
因为窝太菜了 写了`（p1-1）*(p1-1)*(q1-1)`
就一直出不来
这里有一个性质

> 若m=m1*m2 m1与m有相同的素因数 则phi(m)=m2*phi(m1)

[CISCN2018]oldstreamgame

考点是lsfr
https://www.anquanke.com/post/id/181811
参考这篇文章

```python
#python3
from Crypto.Util.number import*

f = open('key','rb').read()
r = bytes_to_long(f)
bin_out = bin(r)[2:].zfill(100*8)
R = bin_out[:32]    #获取输出序列中与掩码msk长度相同的值
print(R)

mask = '10100100000010000000100010010100'  #顺序 c_n,c_n-1,。。。,c_1
key =  '00100000111111011110111011111000'

R = ''
for i in range(32):
    output = 'x'+key[:31]
    out = int(key[-1])^int(output[-3])^int(output[-5])^int(output[-8])^int(output[-12])^int(output[-20])^int(out
put[-27])^int(output[-30])
    R += str(out)
    key = str(out)+key[:31]
print('flag{'+hex(eval('0b'+R[::-1]))+'}')
```