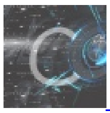
2021-07-23

原创

[无名函数](#) 于 2021-07-23 23:41:40 发布 47 收藏
分类专栏： [Buu-crypto](#) 文章标签： [算法](#)


[Buu-crypto 专栏收录该内容](#)
72 篇文章 1 订阅
订阅专栏

[ACTF新生赛2020]crypto-des

题目

encryptedkey

```
72143238992041641000000.000000,
77135357178006504000000000000000.000000,
1125868345616435400000000.000000,
67378029765916820000000.000000,
75553486092184703000000000000000.000000,
4397611913739958700000.000000,
76209378028621039000000000000000.000000
```

hint

```
To solve the key, Maybe you know some interesting data format about C language?
```

和一个加密压缩包
解题

为了解决这个问题，也许你知道一些关于C语言的有趣的数据格式？

那一串数字应该为double型(double:646f75626c65;01100100 01101111 01110101 01100010 01101100 01100101)

根据提示，我们需要解出压缩包密码，但还是尝试手贱一下不需要密码的方式，然后就真的没解出来，还是老老实实算密码吧

```python
from libnum import*
import struct
import binascii

s = [72143238992041641000000.000000,77135357178006504000000000000000.000000,1125868345616435400000000.000000,67378029765916820000000.000000,7555348609218470300000000000000.000000,4397611913739958700000.000000,762093780286210390000000000000000.000000]
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))
```

运行得到：

```
496e74657265737472696e67204964656120746f20656e6372797074
74707972747079727470797274707972747079727470797274707972
b'Interestring Idea to encrypt'
b'tpyrtpyrtpyrtpyrtpyrtpyrtpyr'
```

尝试解密发现密钥为：`Interestring Idea to encrypt`

解压得到：

```python
import pyDes
import base64
from FLAG import flag
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]
  ]
cipher_list = base64.b64encode(DES.encrypt(flag))
#b'vrkgBqeK7+h7mPyWujP8r5FqH5yyVlqv0CXudqoNHVAVdNO8ML4lM4zgez7weQXo'
```

DES全称为Data Encryption Standard，即数据加密标准，是一种使用密钥加密的块算法。
DES算法的入口参数有三个：Key、Data、Mode。其中Key为7个字节共56位，是DES算法的工作密钥；Data为8个字节64位，是要被加密或被解密的数据；Mode为DES的工作方式,有两种:加密或解密。
DES算法把64位的明文输入块变为64位的密文输出块,它所使用的密钥也是64位（实际用到了56位，第8、16、24、32、40、48、56、64位是校验位， 使得每个密钥都有奇数个1），其算法主要分为两步：初始置换和逆置换

这道题的话，直接输入解密代码就行：

```python
import pyDes
import base64
#from FLAG import flag
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]
  ]
#cipher_list = base64.b64encode(DES.encrypt(flag))

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

运行得到：b’actf{breaking_DES_is_just_a_small_piece_of_cake}’

答案

flag{breaking_DES_is_just_a_small_piece_of_cake}

[De1CTF2019]xorz

题目

```python
from itertools import *
from data import flag,plain

key=flag.strip("de1ctf{").strip("}")
assert(len(key)<38)
salt="WeAreDe1taTeam"
ki=cycle(key)
si=cycle(salt)
cipher = ''.join([hex(ord(p) ^ ord(next(ki)) ^ ord(next(si)))[2:].zfill(2) for p in plain])
print cipher
# output:
# 49380d773440222d1b421b3060380c3f403c3844791b202651306721135b6229294a3c3222357e766b2f15561b35305e3c3b670e49382c
295c6c170553577d3a2b791470406318315d753f03637f2b614a4f2e1c4f21027e227a4122757b446037786a7b0e37635024246d60136f78
02543e4d36265c3e035a725c6322700d626b345d1d6464283a016f35714d434124281b607d315f66212d671428026a4f4f79657e34153f34
67097e4e135f187a21767f02125b375563517a3742597b6c394e78742c4a725069606576777c314429264f6e330d7530453f22537f5e3034
560d22146831456b1b72725f30676d0d5c71617d48753e26667e2f7a334c731c22630a242c7140457a42324629064441036c7e646208630e
745531436b7c51743a36674c4f352a5575407b767a5c747176016c0676386e403a2b42356a727a04662b4446375f36265f3f124b724c6e34
6544706277641025063420016629225b43432428036f29341a2338627c47650b264c477c653a67043e6766152a485c7f3361726478065653
7e5468143f305f4537722352303c3d4379043d69797e6f3922527b24536e310d653d4c33696c635474637d0326516f745e610d7733403066
21105a7361654e3e392970687c2e335f3015677d4b3a724a4659767c2f5b7c16055a126820306c14315d6b59224a27311f747f336f4d5974
321a22507b22705a226c6d446a37375761423a2b5c29247163046d7e47032244377508300751727126326f117f7a38670c2b23203d4f2704
6a5c5e1532601126292f577776606f0c6d0126474b2a73737a41316362146e581d7c1228717664091c
```

解题

官方wp

```python
def getKeyPool(cipher, stepSet, plainSet, keySet):
    ''' 传入的密文串、明文字符集、密钥字符集、密钥长度范围均作为数字列表处理.形如[0x11,0x22,0x33]
        返回一个字典，以可能的密钥长度为键，以对应的每一字节的密钥字符集构成的列表为值，密钥字符集为数字列表。
            形如{
                    1:[[0x11]],
                    3:[
                        [0x11,0x33,0x46],
                        [0x22,0x58],
                        [0x33]
                       ]
                }
    '''
    keyPool = dict()
    for step in stepSet:
        maybe = [None] * step
        for pos in xrange(step):
            maybe[pos] = []
            for k in keySet:
                flag = 1
                for c in cipher[pos::step]:
                    if c ^ k not in plainSet:
                        flag = 0
                if flag:
                    maybe[pos].append(k)
        for posPool in maybe:
            if len(posPool) == 0:
                maybe = []
                break
        if len(maybe) != 0:
            keyPool[step] = maybe
    return keyPool
```

```
def calCorrelation(cpool):
    '''传入字典，形如{'e':2,'p':3}
        返回可能性，0~1,值越大可能性越大
        (correlation between the decrypted column letter frequencies and
        the relative letter frequencies for normal English text)
    '''
    frequencies = {"e": 0.12702, "t": 0.09056, "a": 0.08167, "o": 0.07507, "i": 0.06966,
                   "n": 0.06749, "s": 0.06327, "h": 0.06094, "r": 0.05987, "d": 0.04253,
                   "l": 0.04025, "c": 0.02782, "u": 0.02758, "m": 0.02406, "w": 0.02360,
                   "f": 0.02228, "g": 0.02015, "y": 0.01974, "p": 0.01929, "b": 0.01492,
                   "v": 0.00978, "k": 0.00772, "j": 0.00153, "x": 0.00150, "q": 0.00095,
                   "z": 0.00074}
    relative = 0.0
    total = 0
    fpool = 'etaoinshrdlcumwfgypbvkjxqz'
    total = sum(cpool.values())  # 总和应包括字母和其他可见字符
    for i in cpool.keys():
        if i in fpool:
            relative += frequencies[i] * cpool[i] / total
    return relative


def analyseFrequency(cfreq):
    key = []
    for posFreq in cfreq:
        mostRelative = 0
        for keyChr in posFreq.keys():
            r = calCorrelation(posFreq[keyChr])
            if r > mostRelative:
                mostRelative = r
                keychar = keyChr
        key.append(keychar)

    return key


def getFrequency(cipher, keyPoolList):
    ''' 传入的密文作为数字列表处理
        传入密钥的字符集应为列表，依次包含各字节字符集。
            形如[[0x11,0x12],[0x22]]
        返回字频列表，依次为各字节字符集中每一字符作为密钥组成部分时对应的明文字频
            形如[{
                    0x11:{'a':2,'b':3},
                    0x12:{'e':6}
                 },
                 {
                    0x22:{'g':1}
                 }]
    '''
    freqList = []
    keyLen = len(keyPoolList)
    for i in xrange(keyLen):
        posFreq = dict()
        for k in keyPoolList[i]:
            posFreq[k] = dict()
            for c in cipher[i::keyLen]:
                p = chr(k ^ c)
                posFreq[k][p] = posFreq[k][p] + 1 if p in posFreq[k] else 1
        freqList.append(posFreq)
    return freqList
```

```python
def vigenereDecrypt(cipher, key):
    plain = ''
    cur = 0
    ll = len(key)
    for c in cipher:
        plain += chr(c ^ key[cur])
        cur = (cur + 1) % ll
    return plain


def main():
    ps = []
    ks = []
    ss = []
    ps.extend(xrange(32, 127))
    ks.extend(xrange(0xff + 1))
    ss.extend(xrange(38))
    cipher = getCipher(c)

    keyPool = getKeyPool(cipher=cipher, stepSet=ss, plainSet=ps, keySet=ks)
    for i in keyPool:
        freq = getFrequency(cipher, keyPool[i])
        key = analyseFrequency(freq)
        plain = vigenereDecrypt(cipher, key)
        print plain,"\n"
        print ''.join(map(chr,key))


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

但是上面程序适用于python2环境，
来看某位大佬的程序

```python
import string
from binascii import unhexlify, hexlify
from itertools import *

def bxor(a, b):     # xor two byte strings of different lengths
    if len(a) > len(b):
        return bytes([x ^ y for x, y in zip(a[:len(b)], b)])
    else:
        return bytes([x ^ y for x, y in zip(a, b[:len(a)])])

def hamming_distance(b1, b2):
    differing_bits = 0
    for byte in bxor(b1, b2):
        differing_bits += bin(byte).count("1")
    return differing_bits

def break_single_key_xor(text):
    key = 0
    possible_space = 0
    max_possible = 0
    letters = string.ascii_letters.encode('ascii')
    for a in range(0, len(text)):
        maxpossible = 0
        for b in range(0, len(text)):
            if(a == b):
                continue
            c = text[a] ^ text[b]
            if c not in letters and c != 0:
                continue
            maxpossible += 1
        if maxpossible > max_possible:
            max_possible = maxpossible
            possible_space = a
    key = text[possible_space] ^ 0x20
    return chr(key)

salt = "WeAreDe1taTeam"
si = cycle(salt)
b = unhexlify(b'49380d773440222d1b421b3060380c3f403c3844791b202651306721135b6229294a3c3222357e766b2f15561b35305e
3c3b670e49382c295c6c170553577d3a2b791470406318315d753f03637f2b614a4f2e1c4f21027e227a4122757b446037786a7b0e376350
24246d60136f7802543e4d36265c3e035a725c6322700d626b345d1d6464283a016f35714d434124281b607d315f66212d671428026a4f4f
79657e34153f3467097e4e135f187a21767f02125b375563517a3742597b6c394e78742c4a725069606576777c314429264f6e330d753045
3f22537f5e3034560d22146831456b1b72725f30676d0d5c71617d48753e26667e2f7a334c731c22630a242c7140457a4232462906444103
6c7e646208630e745531436b7c51743a36674c4f352a5575407b767a5c747176016c0676386e403a2b42356a727a04662b4446375f36265f
3f124b724c6e346544706277641025063420016629225b43432428036f29341a2338627c47650b264c477c653a67043e6766152a485c7f33
617264780656537e5468143f305f4537722352303c3d4379043d69797e6f3922527b24536e310d653d4c33696c635474637d0326516f745e
610d773340306621105a7361654e3e392970687c2e335f3015677d4b3a724a4659767c2f5b7c16055a126820306c14315d6b59224a27311f
747f336f4d5974321a22507b22705a226c6d446a37375761423a2b5c29247163046d7e47032244377508300751727126326f117f7a38670c
2b23203d4f27046a5c5e1532601126292f577776606f0c6d0126474b2a73737a41316362146e581d7c1228717664091c')
plain = ''.join([hex(ord(c) ^ ord(next(si)))[2:].zfill(2) for c in b.decode()])
b = unhexlify(plain)
print(plain)

normalized_distances = []

for KEYSIZE in range(2, 40):
    # 我们取其中前6段计算平局汉明距离
    b1 = b[: KEYSIZE]
    b2 = b[KEYSIZE: KEYSIZE * 2]
    b3 = b[KEYSIZE * 2: KEYSIZE * 3]
    b4 = b[KEYSIZE * 3: KEYSIZE * 4]
    b5 = b[KEYSIZE * 4: KEYSIZE * 5]
    b6 = b[KEYSIZE * 5: KEYSIZE * 6]

    normalized_distance = float(
        hamming_distance(b1, b2) +
        hamming_distance(b2, b3) +
        hamming_distance(b3, b4) +
        hamming_distance(b4, b5) +
        hamming_distance(b5, b6)
    ) / (KEYSIZE * 5)
    normalized_distances.append(
        (KEYSIZE, normalized_distance)
    )
normalized_distances = sorted(normalized_distances, key=lambda x: x[1])

for KEYSIZE, _ in normalized_distances[:5]:
    block_bytes = [[] for _ in range(KEYSIZE)]
    for i, byte in enumerate(b):
        block_bytes[i % KEYSIZE].append(byte)
    keys = ''
    try:
        for bbytes in block_bytes:
```

```
        for bbytes in block_bytes:
            keys += break_single_key_xor(bbytes)
        key = bytearray(keys * len(b), "utf-8")
        plaintext = bxor(b, key)
        print("keysize:", KEYSIZE)
        print("key is:", keys, "n")
        s = bytes.decode(plaintext)
        print(s)
    except Exception:
        continue
```

运行得到

```
1e5d4c055104471c6f234f5501555b5a014e5d001c2a54470555064c443e235b4c0e590356542a130a4242335a47551a590a136f1d5d4d44
0b0956773613180b5f184015210e4f541c075a47064e5f001e2a4f711844430c473e2413011a100556153d1e4f45061441151901470a196f
035b0c4443185b322e130806431d5a072a46385901555c5b550a541c1a2600564d5f054c453e32444c0a434d43182a0b1c540a55415a550a
5e1b0f613a5c1f10021e56773a5a0206100852063c4a18581a1d15411d17111b052113460850104c472239564c0755015a13271e0a55553b
5a47551a54010e2a06130b5506005a393013180c100f52072a4a1b5e1b165d50064e411d0521111f235f114c47362447094f10035c066f19
025402191915110b4206182a544702100109133e394505175509671b6f0b01484e06505b061b50034a2911521e44431b5a233f13180b5508
1315230501544037404155034 84f0c2602564d470a18407b775d031110004a54290319544e06505b060b424f092e1a770443101952333213
030d554d551b2006064206555d50141c454f0c3d1b5e4d43061e453e39544c17580856581802001102105443101d111a043c03521455074c
473f3213000a5b085d113c194f5e08555415180f5f433e270d131d420c1957773f560d11440d40543c060e470b55545b114e470e193c155f
4d47110947343f13180c100f565a000403484e184c15050250081f2a54470545104c5536251325435302461a3b4a02484e12545c1b426507
0b3b5440055543185b36231301025b084054220f4f42071b1554020f430b196f19564d4002055d79
keysize: 30
key is: W3lc0m3tOjo1nu55un1ojOt3m0cl3W n
In faith I do not love thee with mine eyes,For they in thee a thousand errors note;But `tis my heart that loves 
what they despise,Who in despite of view is pleased to dote.Nor are mine ears with thy tongue`s tune delighted;N
or tender feeling to base touches prone,Nor taste, nor smell, desire to be invitedTo any sensual feast with thee
 alone.But my five wits, nor my five senses canDissuade one foolish heart from serving thee,Who leaves unswayed 
the likeness of a man,Thy proud heart`s slave and vassal wretch to be.Only my plague thus far I count my gain,Th
at she that makes me sin awards me pain.
keysize: 9
key is: mulpelOut n
s( u4hiN:9q07t:0upZ1+J r!1RS> A,w;!Fco.
F.* v)o  h) 1gy3yflf*t0pMA: qr67c"ujG:h!/C2JIfmjui`Is:)vq-Zlu* uf7C17u.^^vdI6i7rF6]5N (6 f$yviu" *i< R}18g6!3}FD
i g -*0fn{O0ounQ#W/nvudsH'm4jxyhc|niQv*G%d!2NI3 H u7fKno9N.* v$dbesgf jp?U flaec"bFn*vc1 c"hqLdsS:}2BI2e?uostw8r
|uZd /stZ1+MeudfRI iX }
n{d$s$6sw f&fd&s1/k?Opflf dcpOJt -B,10o:xKw:="fW5(oaul!]nl8>c<s /:e^ K6dt'_BvoB 98nLvc.I )=ap5*`rn* 6jn Rv!8z-d&
=tMueoe83uq^opQv>d0k2K_flz>ddHt:2x08Zm{26RWh R7xt"O3a^1y-!Pvk+D  6d"7kus`+ 2}y"Xpflaec&?lKv<#m e`n}kG!+u | BHfI3
6n oO'w$>w8n6rgK1,J 7u.ZSvmM.|-!N *.Hna9wc3nu l" 5nu8
keysize: 2
key is: tr n
j/8w%v3nQ;'u'/(u<)rhX 5q'r>0LW)8|-q"&^a~06A.5!h-xgi/96 {"Baly+j4gU|;&hu.5r<+rjX;l67~3LPauhdw"gIl;7rf5gms3xmw)x67
j/@Za|t7o.u^4L+u'()!x nnTt$9-q>1LF68x7?7j^yh&~'5(!x*i{N.kbvl"N(vtdz&tH8l*noa3ieeiqSg4|"d>3PM$8u!s.aSl~'!I.5!h sz
Xra 'rr.KDal~d}&u^8o,od)"r<5oqSemW-e>3DP5}=dq(tkv&vkmgey6tlX 5vbu{gLM7qe!{iyu::t$)ri$q>[e j67i.QKaly!zggWwu&4E43
!q<=xTv$95~j4 /wcdr>&]qm&:t$)ry6=}\np1dk&AFaw !?!iTtr0r')"`n1=xOo,91rl1LM&8e,z"*lptcvb 1doehpNw `'s>3MFatx/z)cHk
;,|' gl}+1JUyai0xk#K$yc0 4&Htz5 ' )e<3|mNa-95e{3FKal~d}"(tvw::j8gqp$zkX 5q7d>!DQaQ1'p2hO8v::` .o0u I 2q'7j/DWaup
/z4&V};0sia&v}7ymm$92vw)
keysize: 7
key is: zttpuvl n
d)8u$r+fW? w9!.u>(vpP 3u p >JW+9x5y" Zf|.8G.7 l5pgm(;(q}"Cetq+l0`Wb5 hw/1j4+tn_9b07|2HHiun`p yGj;5sb-omu7 oy/x46
n7HZgxs5q s^6M/m/(/% "p`Rt&8)i61JB1:f997h_}p.~!1/#f$o{O*sjvj&L6xrdx'pP0l,jhc-gcekpW <|$`91NC"8w w6iSjz #W 3!j!wb
Prg{ pl MDcmz|u&sZ?m2ab) s8-gqUajU3k83FQ1e5dw,sux vilc}q6rh_"+xduyfHU?qc%|w u8;p<!ro v<Ek&j46m6YKgh~#diaWut",M45
%v>#vRv&81fb4(u}jt>$\uu.:r .pg8;}^oh9dm"FD yy!= mL|r6v +<nh1?yKw$97vk3RC 8g-~:"lvpdt|.7dmdlhFw&d q =KFcu|7r)eLl9
2r! emy39JS}fk.vm#J ak0y0!Jjt3 %!-}43ziIc373ey2BSilx`z 6zpw8;n oqv }iF.3q5e:9LQgU6%n<nO:w>"h (k7kqO 0p#/b/BSfwn!
|4$Wy#8soe!tc9 ml !:vq-
keysize: 19
key is: h{ct>aoQuaougultjj{ n
v&/qoe(MB  f 7.k$&hgI yd:W9%QV<9b-i</Bhi6|R5 {6 tq)'.pa-B-ydm!zTi:8hm0<n5<t K  m%,y KHgkpkm-vI .*Wa zlf2fmi d? l
eSAB}g,h=rF2R3z='8!45sKSa988p 1TX?$q 9}yEZi5e &/9~4qt A?k.cq[5waed&lV1p#yi+ rFdzjTt3d$z&<JB5894nfFq 2 W.-?a<zm^8
rdsa5LWftxze)oQ)o`zy%g!4zpMeuI$y7$B&feb3sln hsb}jh68yE2c tnfRM/ol=roQjn;g?.an<w Cj:e'7%;Lnfyd ofyWok/(L#5kb'yGm#
*2fl*5xrd>+;xvx;;a%7ra(4aUy:" H'R]fdx99?q[n}!rk<?Ei$ yZn29)le-EZ rv7Y#9wwgdnd>)kujypb=E f#2XG t`1s5j_mq?g!twz86R
Sgyf*wz#I^9\d%b53Ijz-a.< r:yovm`>"2v|+@Uycdkl"dakR=/w9rpn$buQ<<f1.-:gPrJ64w*nQ y 5q bz-4rjT!'p97r1MKhbvei/Wn 7`n
y he8cbmh,/Sp<
```

可以看出当运行出来的是明文的时候密钥为flag

答案

flag{W3lc0m3tOjo1nu55un1ojOt3m0cl3W}

[NCTF2019]easyRSA

题目

```python
from flag import flag

e = 0x1337
p = 199138677823743837339927520157607820029746574557746549094921488292877226509198315016018919385259781238148402
8333160336349681632761989992793278279018794264296646743588440844918305432716251472809502739344058793414384291714
53002453838897458102128836690385604150324972907981960626767679153125735677417397078196059
q = 112213695905472142415221444515326532320352429478341683352811183503269676555434601229013679319423878238944956
8302443866536744134116586967511738444433946082467160530862269105814005281678483061191798791158097787930936113817
64939789057524575349501163689452810148280625226541609383166347879832134495444706697124741
n = p * q

assert(flag.startswith('NCTF'))
m = int.from_bytes(flag.encode(), 'big')
assert(m.bit_length() > 1337)

c = pow(m, e, n)
print(c)
# 10562302690541901187975815594605242014385201583329309191736952454310803387032252007244962585846519762051885640
8560821570605938290135725928129582614323279751385817843603025992654081343320941348807890132073822778495033440424
8738985037348765620065785686209690086079227320644755213245843098953482025615602112889129638741468969395204730260
4774923411425863612316726417214819110981605912408620996068520823370069362751149060142640529571400977787330956486
8494490054027502249920485628980043093195771926933156582759124491983657379655700352648417823999783073889206810686
46219895287752359564029778568376881425070363592696751183359
```

解题
已知n、p、q、c
但是算出来的phi与e不互素
所以要换一种方式来解

[NCTF2019]babyRSA

题目

```python
from Crypto.Util.number import *
from flag import flag

def nextPrime(n):
    n += 2 if n & 1 else 1
    while not isPrime(n):
        n += 2
    return n

p = getPrime(1024)
q = nextPrime(p)
n = p * q
e = 0x10001
d = inverse(e, (p-1) * (q-1))
c = pow(bytes_to_long(flag.encode()), e, n)

# d = 19275778946037899718035455438175509175723911466127462154506916564101519923603308900331427601983476886255849200332374081996442976307058597390881168155862238533018621944733299208108185814179466844504468163200369996564265921022888670062554504758512453217434777820468049494313818291727050400752551716550403647148197148884408264686846693842118387217753516963449753809860354047619256787869400297858568139700396567519469825398575103885487624463424429913017729585620877168171603444111464692841379661112075123399343270610272287865200880398193573260848268633461983435015031227070217852728240847398084414687146397303110709214913
# c = 53827231680738281106961685582942066817579911490227778211275633014134832238745272333007211808392986170767056850411742474158261570965830550693373939878922627642112252270358807544174570567239091355252449579359069026656797771011301113927802375029286562257052624314319530035200939329243759021112800772552051182174367441120640694296786329232598986279971458038927539892556152731403000210406545059014427878106536265243057063166631693417972057529387555900565689867382278034874672741143982571879621407965511362205328096876068673856393677437055275116807199553807463776311564686898441508783814605609907556528994493400453135218O4
```

解题

已知e、d、c还需要知道n就可以求解了

```python
from Crypto.Util.number import *
from gmpy2 import *

e = 0x10001
d = 19275778946037899718035455438175509175723911466127462154506916564101519923603308900331427601983476886255849200332374081996442976307058597390881168155862238533018621944733299208108185814179466844504468163200369996564265921022888670062554504758512453217434777820468049494313818291727050400752551716550403647148197148884408264686846693842118387217753516963449753809860354047619256787869400297858568139700396567519469825398575103885487624463424429913017729585620877168171603444111464692841379661112075123399343270610272287865200880398193573260848268633461983435015031227070217852728240847398084414687146397303110709214913
c = 53827231680738281106961685582942066817579911490227778211275633014134832238745272333007211808392986170767056850411742474158261570965830550693373939878922627642112252270358807544174570567239091355525244957935906902665679777101130111392780237502928656225705262431431953003520093932924375902111280077255205118217436744112064069429678632923259898627997145803892753989255615273140300021040654505901442787810653626524305706316663169341797205752938755590056568986738227803487467274114398257187962140796551136220532809687606867385639367743705527511680719955380746377631156468689844150878381460560990755652899449340045313521804

kphi = e*d - 1

for k in range(1, e):
    if kphi % k == 0:
        phi = kphi // k
        root = iroot(phi, 2)[0]
        for p in range(root - 2000, root + 2000):
            if phi % (p-1) == 0: break
        else: continue
        break

q = phi//(p-1) + 1
m = pow(c, d, p*q)
print(long_to_bytes(m))
```

运行得到`b'NCTF{70u2_nn47h_14_v3ry_gOO0000000d}'`
答案
flag{70u2_nn47h_14_v3ry_gOO0000000d}