BMZCTF-crypto- writeup

原创

rang# 于 2021-08-14 13:13:00 发布 1485 收藏 1

分类专栏： BMZCTF CRYPTO 文章标签： Crypto CTF BMCTF

版权声明：本文为博主原创文章，遵循 CC 4.0 BY-SA 版权协议，转载请附上原文出处链接和本声明。

本文链接：https://blog.csdn.net/weixin_45859850/article/details/119699182

版权

BMZCTF 同时被 2 个专栏收录

1 篇文章 0 订阅

订阅专栏

CRYPTO

22 篇文章 1 订阅

订阅专栏

BMZCTF -crypto- writeup

目录

2018 HEBTUCTF Simple Caesar!

2018 AFCTF 你能看出这是什么加密么

2018 HEBTUCTF easy_crypto （morse、baconian）

看的出来吗

easy_CRC

键盘之争

2018 AFCTF Single

HEBTUCTF WIFNVDSFFDYUBPAMEMS

Crypto_easy_crypto 拆分模数得到好求的phi

Crypto_xor DefCamp CTF 2020 – why-xor

rsass

2018 AFCTF One Secret, Two encryption

2018 AFCTF Vigenère

2018 AFCTF 你听过一次一密么？ Many-Time-Pad流密码加密

2018 QCTF Xman-RSA

火眼金睛

1837

Caesar

rsa

2020sdnisc-ezRSA

#####baby_dsa 上海市大学生网络安全大赛

技协杯-Crypto1

2018 AFCTF BASE

python2、python3byte和str间的转换

#######2018 AFCTF MyOwnCBC

2018 AFCTF 花开藏宝地 门限方案

2018 AFCTF 一道有趣的题目

[2021-红明谷]RSA at

tack 低加密指数

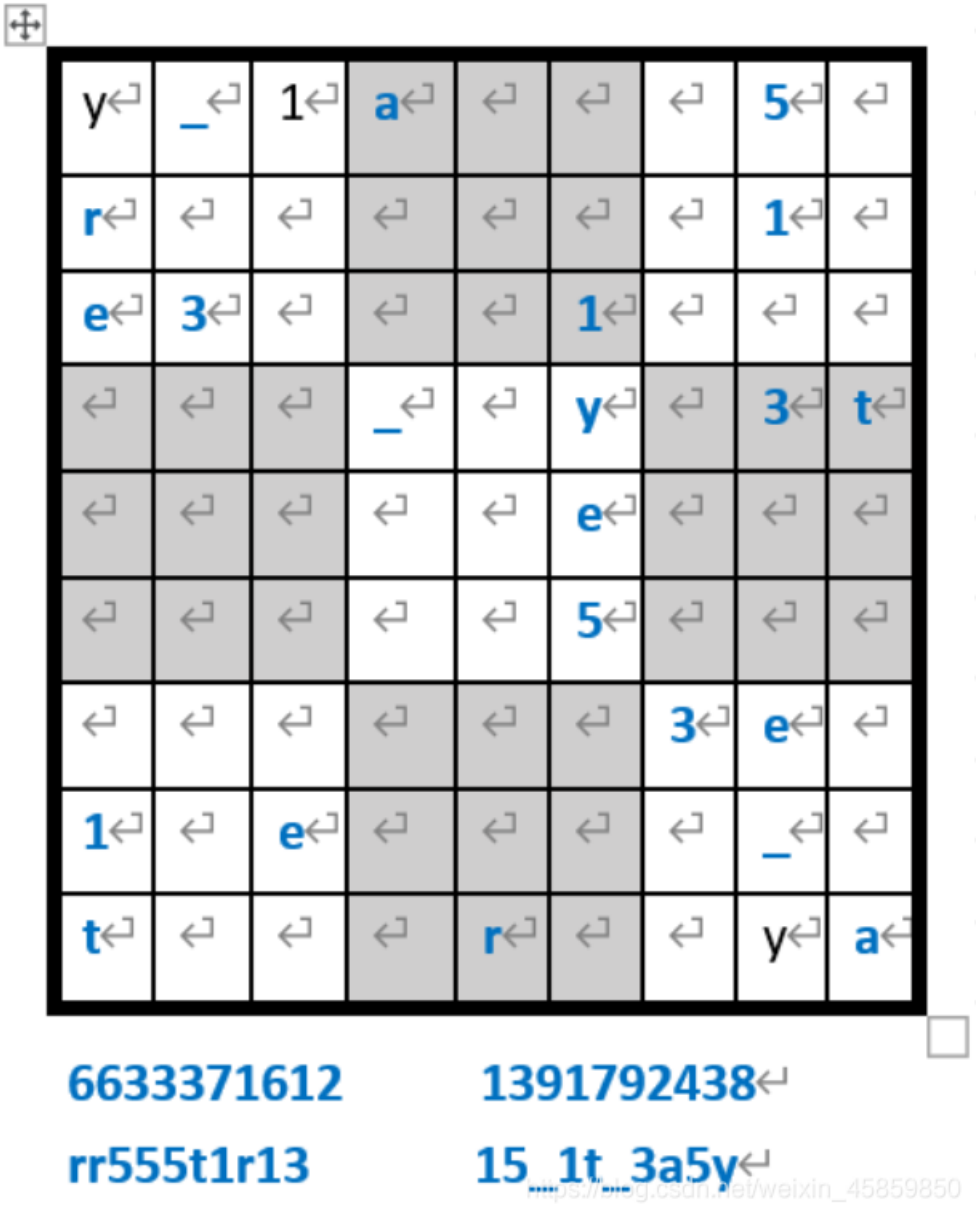
经典与现代的碰撞

LRX

Sudoku&Viginere

数独，每个块内可填：**1,3,5,a,e,r,t,y,_**

After solving Sudoku, you will find Viginere's secret:45 34 57 74 15 35 26 86 47 39



6633371612 1391792438
rr555t1r13 15_1t_3a5y

http://www.ab126.com/shuxue/10667.html

在线数独求解计算器

Ook

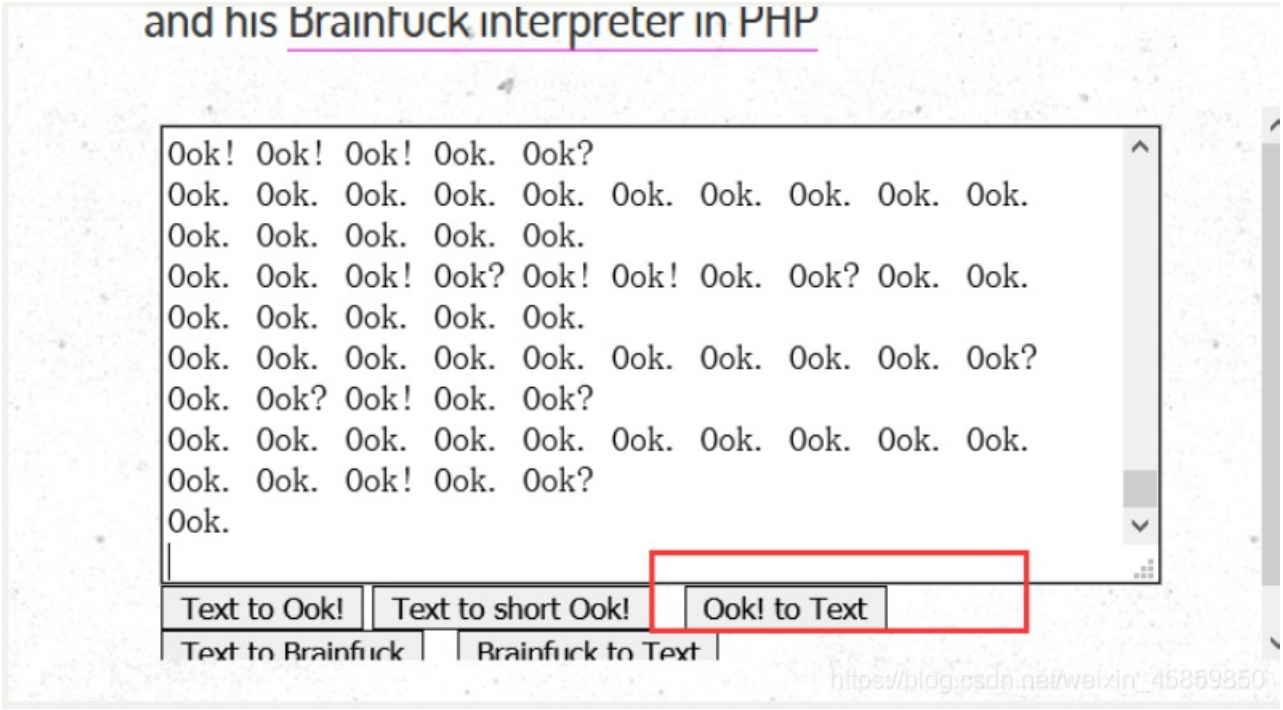
描述：

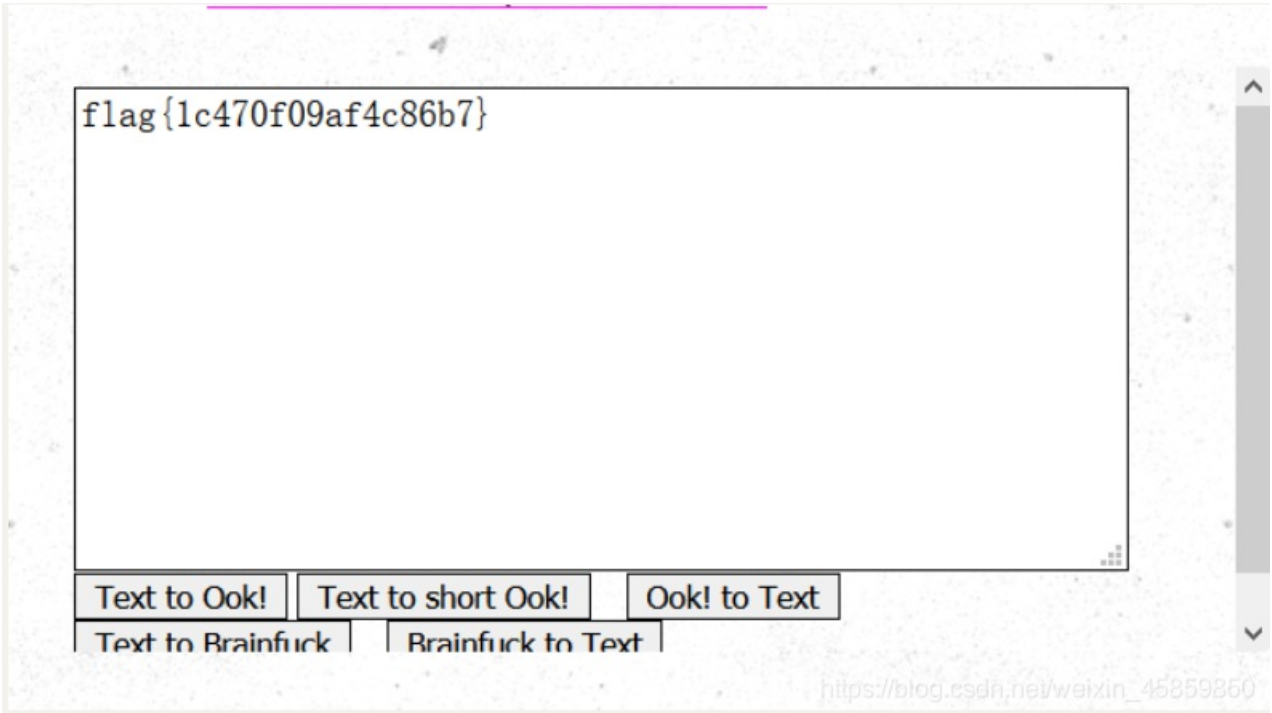
Brainfuck是一种极小化的计算机语言，它是由Urban Müller在1993年创建的。由于fuck在英语中是脏话，这种语言有时被称为brainf*ck或brainf**k，甚至被简称为BF。
Ook与Brainfuck类似，也是用替换法。
特征：

brainfuck语言用> < + - . , [ ]八种符号来替换C语言的各种语法和命令：
例如： +++++++++++++++++.>++

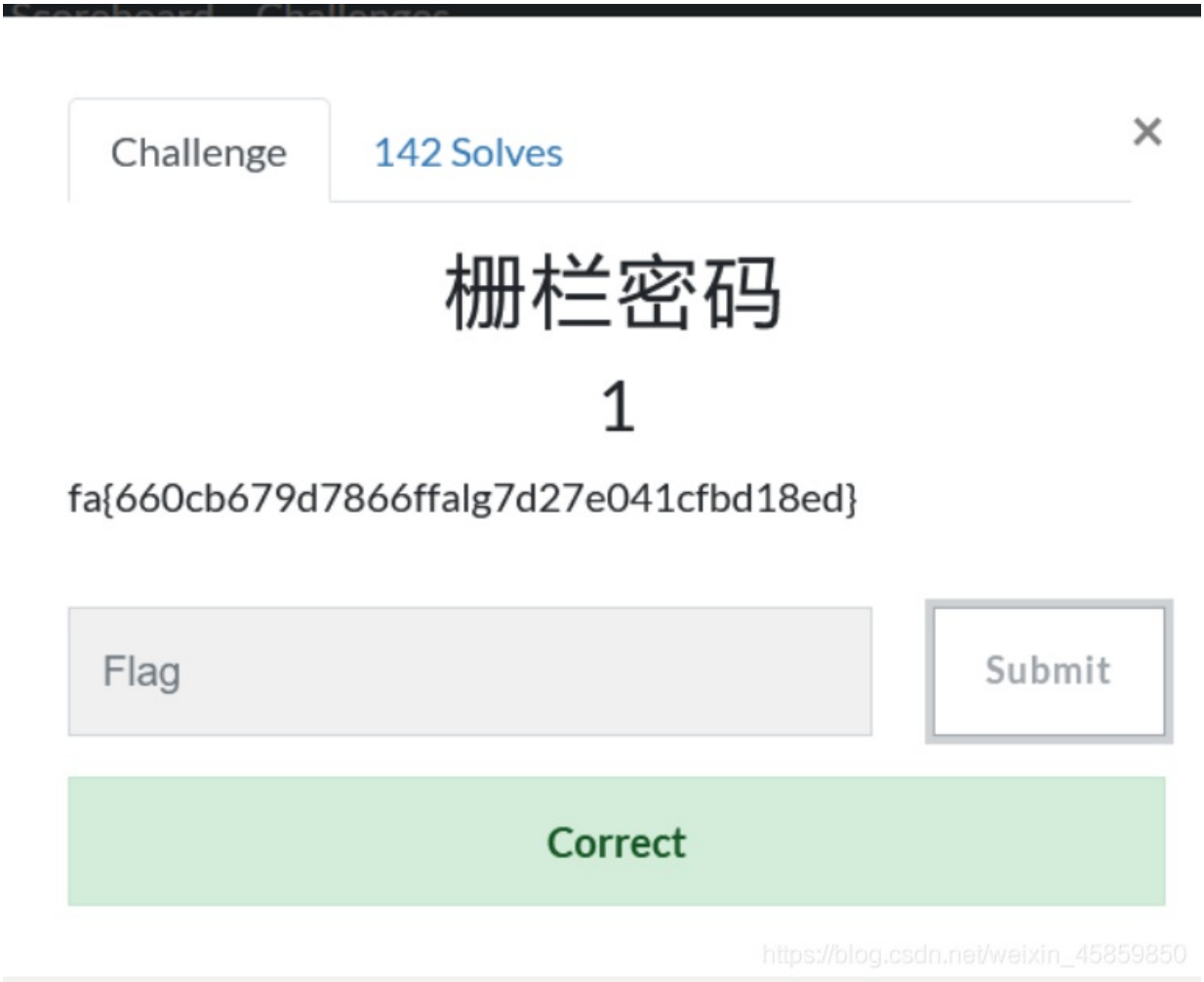
ook密码中有大量ook，加上一些符号；
Ook! has only three distinct syntax elements:
Ook.
Ook?
Ook!
这种就是ook密码
加密解密

在线加密解密：https://www.splitbrain.org/services/ook


and his Brainfuck interpreter in PHP
Ook! Ook! Ook! Ook. Ook?
Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook.
Ook. Ook. Ook. Ook. Ook.
Ook. Ook. Ook! Ook? Ook! Ook! Ook. Ook? Ook. Ook.
Ook. Ook. Ook. Ook. Ook.
Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook?
Ook. Ook? Ook! Ook. Ook?
Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook. Ook.
Ook. Ook. Ook! Ook. Ook?
Ook.
Text to Ook!
Text to short Ook!
Ook! to Text
Text to Brainfuck
Brainfuck to Text


flag{1c470f09af4c86b7}
Text to Ook!
Text to short Ook!
Ook! to Text
Text to Brainfuck
Brainfuck to Text

栅栏密码



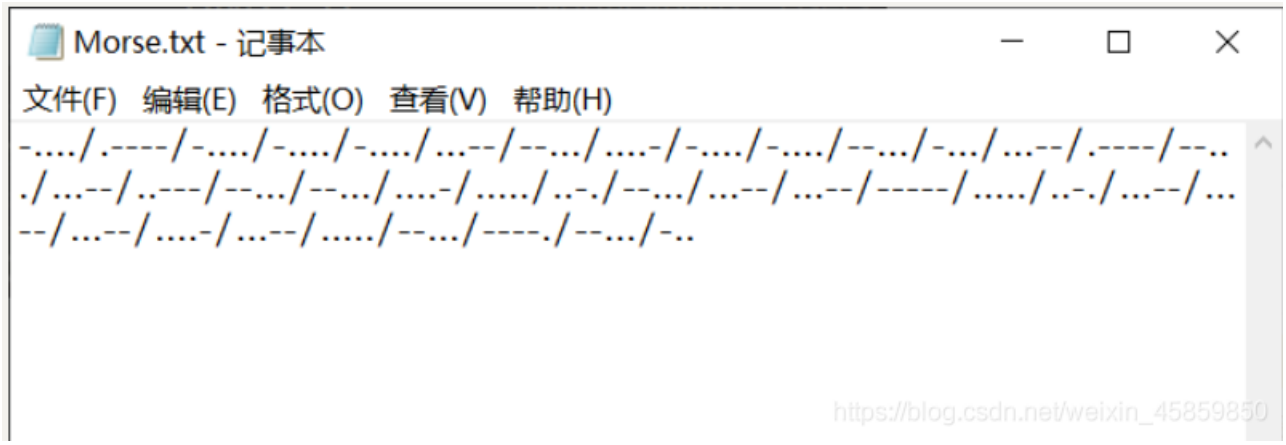

```
'''
遍历所有可能的栏数，并得到加/解密结果
'''
s = input()
factors = [fac for fac in range(2, len(s)) if len(s)%fac == 0] #取得密文长度的所有因数
for fac in factors:
    flag = ''
    for i in range(fac): #按一定的步长取几组字符，并连接起来，这里组数就等于步长数
        flag += s[i::fac]

    print(str(fac)+'栏：'+flag)
```

```
fa{660cb679d7866ffalg7d27e041cfbd18ed}
2栏：f{6c6976fagd701fd8da60b7d86f172e4cb1e}
19栏：flag{76d6207ceb064719cdf7b8d6168fefda}
```

4进制

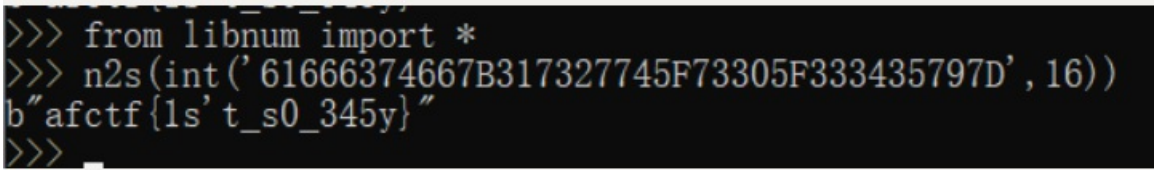
```
s = '1212 1230 1201 1213 1323 1012 1233 1311 1302 1202 1201 1303 1211 301 302 303 1331'.split(' ')
flag = ''
for i in s:
    flag = flag + chr(int(i,4))
 print(flag)
#flag{Fourbase123}
```

2018 AFCTF Morse



https://tool.lu/morse/

morse解码得到16进制数，转字符串即可



2018 HEBTUCTF 社会主义接班人

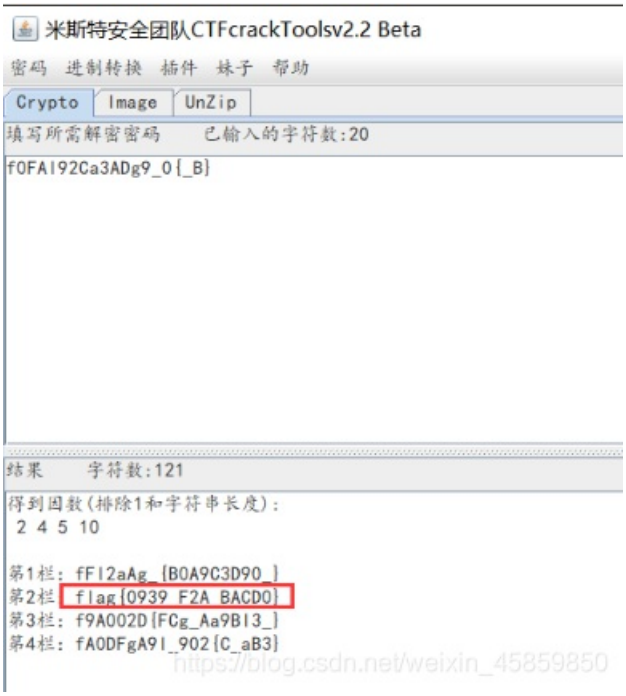
http://ctf.ssleye.com/cvencode.html

栅栏中的base

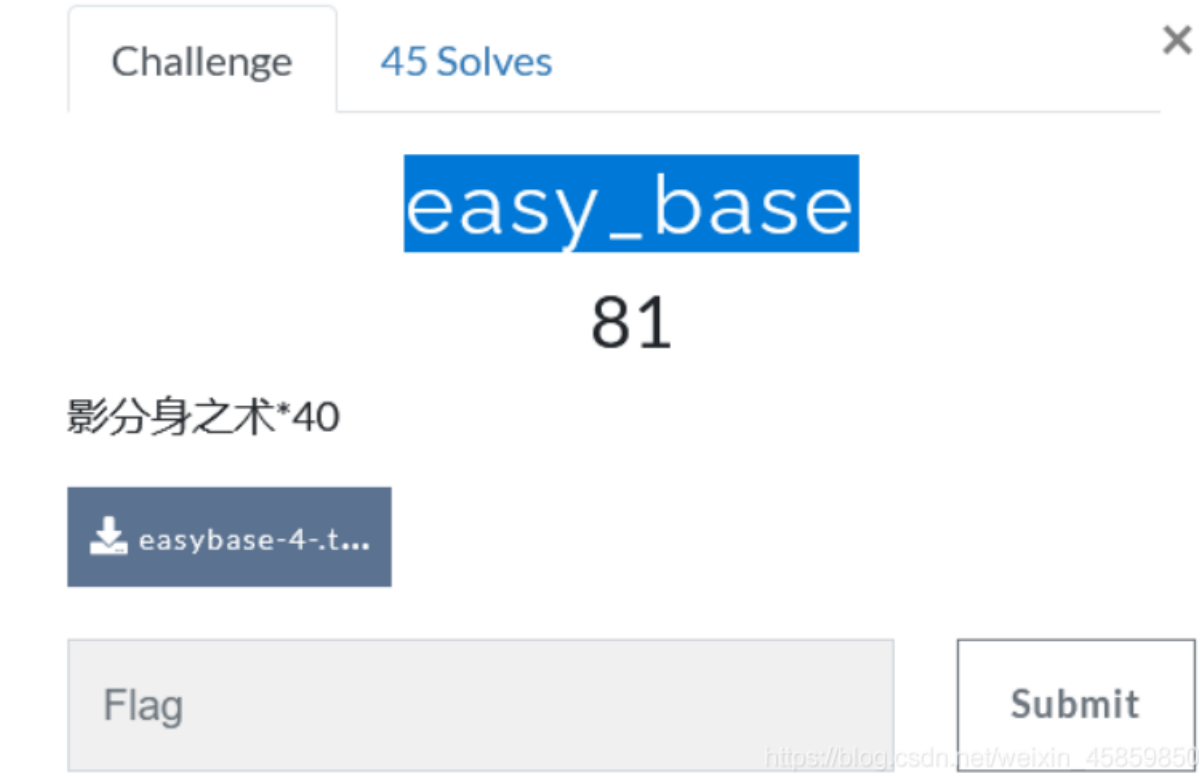
http://ctf.ssleye.com/base64.html

经过base16、32、64解码得到

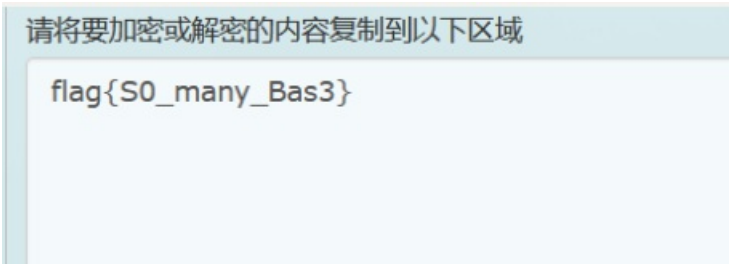
```
f0FAl92Ca3ADg9_0{_B}
```


米斯特安全团队CTFcrackToolsv2.2 Beta
密码 进制转换 插件 妹子 帮助
Crypto
Image
UnZip
填写所需解密密码 已输入的字符数:20
f0FAI92Ca3ADg9_0{_B}
结果 字符数:121
得到因数(排除1和字符串长度):
2 4 5 10
第1栏: fFI2aAg_{B0A9C3D90_}
第2栏: flag{0939_F2A_BACD0}
第3栏: f9A002D{FCg_Aa9BI3_}
第4栏: fA0DFgA9I_902{C_aB3}

easy_base


Challenge
45 Solves
easy_base
81
影分身之术*40
easybase-4-.t...
Flag
Submit

base64解码40次



CRC32 BOOM!

查看压缩包发现1.txt和2.txt文件很小，对其进行crc32爆破

crc32爆破脚本下载地址：`https://github.com/theonlypwner/crc32`

```
D:\Program Files\crc32-master>python crc32.py reverse 0x8e234ae0
4 bytes: {0x0e, 0xf5, 0x08, 0x69}
verification checksum: 0x8e234ae0 (OK)
6 bytes: 17LE1V (OK)
6 bytes: 53QD05 (OK)
6 bytes: 5Cm55e (OK)
6 bytes: 6cfFvu (OK)
6 bytes: 918GQb (OK)
6 bytes: BzTRNZ (OK)
6 bytes: EcS1dq (OK)
6 bytes: HqVcs7 (OK)
6 bytes: JpB1Br (OK)
6 bytes: LuKbrT (OK)
6 bytes: TRyOcs (OK)
6 bytes: _5IbAC (OK)
6 bytes: bugku_ (OK)
6 bytes: dpn8Ey (OK)
6 bytes: mz9jRH (OK)
6 bytes: q5ekFH (OK)
6 bytes: w018vn (OK)
6 bytes: yOOyOO (OK)
6 bytes: zRUFdx (OK)
```

https://blog.csdn.net/weixin_45859850

```
D:\Program Files\crc32-master>python crc32.py reverse 0x8115a277
4 bytes: {0x76, 0xa2, 0x0b, 0xe1}
verification checksum: 0x8115a277 (OK)
6 bytes: 1x5NQ8 (OK)
6 bytes: 9O2a5H (OK)
6 bytes: G1qi4N (OK)
6 bytes: IsCdeH (OK)
6 bytes: NjDZOc (OK)
6 bytes: OvJ7Un (OK)
6 bytes: P8CUk2 (OK)
6 bytes: PTOxov (OK)
6 bytes: QTqIto (OK)
6 bytes: Rh5f6k (OK)
6 bytes: XBiGbJ (OK)
6 bytes: Z3AdV_ (OK)
6 bytes: gsombC (OK)
6 bytes: mY3L6b (OK)
6 bytes: newctf (OK)
6 bytes: p7plP7 (OK)
6 bytes: t3mmQT (OK)
6 bytes: vBENeA (OK)
```

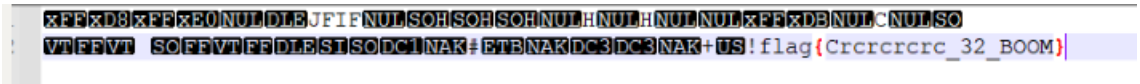
https://blog.csdn.net/weixin_45859850

生成密码字典：

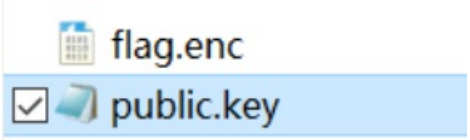
```
s1=["1x5NQ8","9O2a5H","G1qi4N","IsCdeH","NjDZOc","OvJ7Un","P8CUk2","PT0xov","QTqIto","Rh5f6k","XBiGbJ","Z3AdV_",
"gsombC","mY3L6b","newctf","p7plP7","t3mmQT","vBENeA"]

s2=["17LE1V","53QD05","5Cm55e","6cfFvu","9l8GQb","BzTRNZ","EcSldq","HqVcs7","JpB1Br","LuKbrT","TRyOcs","_5IbAC",
"bugku_","dpn8Ey","mz9jRH","q5ekFH","w0l8vn","yOOyO0","zRUFdx"]
l1=[]
l2=[]
for i in s2:
    for j in s1:
        l1.append(i+j)
for j in s2:
    for i in s1:
        l2.append(i+j)
with open ('pw.txt', 'a+') as fo:
    for k in l1:
        fo.write(k+'\n')
    for k in l2:
        fo.write(k+'\n')
```

用zip破解工具字典爆破得到压缩包密码

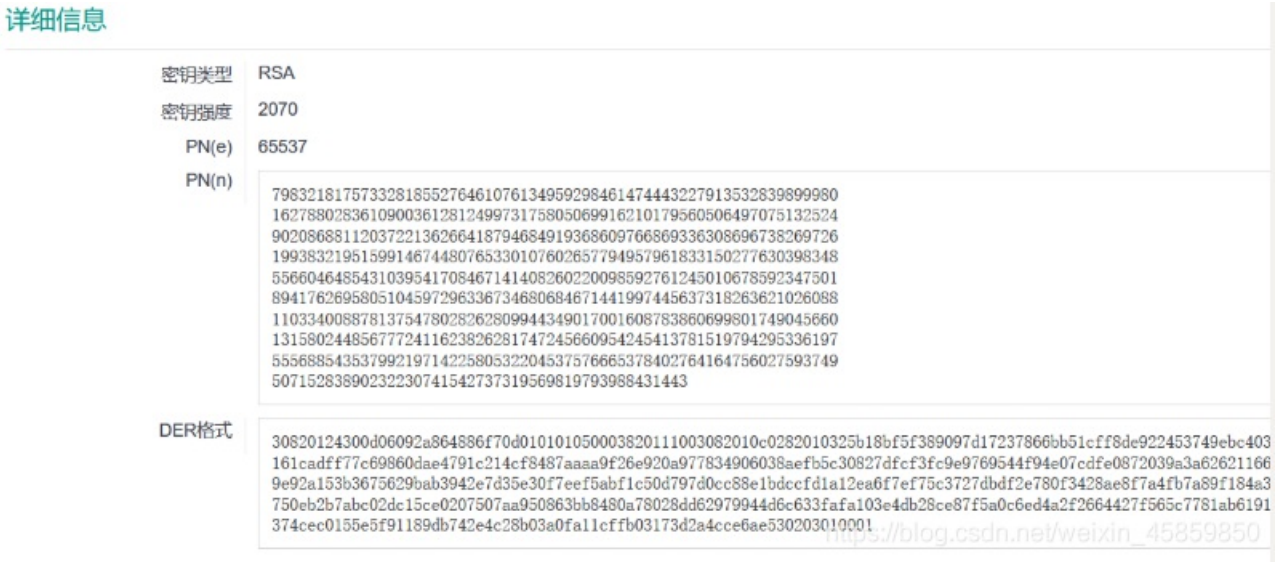
打开flag.jpg报错，用notepad++打开，得到flag



2018 AFCTF 可怜的RSA（公钥、密文.enc）



公钥解析：`http://ctf.ssleye.com/pub_asys.html`




```
from Crypto.PublicKey import RSA
f = open('D:\\Download\\public.key', 'rb').read()
pub = RSA.importKey(f)
n = pub.n
e = pub.e
print(n, '\n', e)
# n = 798321817573328185527646107613495929846147444322791353283989998016278802836109003612812499731758050699162101795605064970751325249020868811203722136266418794684919368609766869336308696738269726199383219515991467448076533010760265779495796183315027763039834855660464854310395417084671414082602200985927612450106785923475018941762695805104597296336734680684671441997445637318263621026088110334008878137547802826280994434901700160878386069980174904566013158024485677724116238262817472456609542454137815197942953361975556885435379921971422580532204537576665378402764164756027593749507152838902322307415427373195698197939884314431
# e = 65537
```

分解模数n得到p、q `http://www.factordb.com`

[外链图片转存失败,源站可能有防盗链机制,建议将图片保存下来直接上传(img-2zJOcOoW-1628914224349)(C:\Users\Rang\AppData\Roaming\Typora\typora-user-images\image-20210522200951841.png)]

通过欧拉定理、p、q求摸反得到d

```
from gmpy2 import *
d = int(invert(e, (p-1)*(q-1)))
print(d)
# d = 406853230956379689450620815713768871010712825839536410687962650677800895818003893712259622281477453292088146173840036827322518131453630576229976208523593618949818777897059256426591560532784635697190752924923710375949616954069804342573867253630978123632384795587951365482103468722384133084798614863870775897915929475258974188300927376911833763105616386167881813301748585233563049693794370642976326692672223638908164822104832415788577945314264232531947860576966629150456995512932232264881080618006698700677529111454508900582785420549466798020451488168615035256292977390692401388790460066327347700109341639992159475755036449
```

打包密钥并对flag.enc解密（flag.enc文件是base64编码的密文）

```
from Crypto.PublicKey import RSA
from Crypto.Cipher import PKCS1_OAEP
from base64 import b64decode
d = 40685323095637968945062081571376887101071282583953641068796265067780089581800389371225962228147745329208814617384003682732251813145363057622997620852359361894981877897059256426591560532784635697190752924923710375949616954069804342573867253630978123632384795587951365482103468722384133084798614863870775897915929475258974188300927376911833763105616386167881813301748585233563049693794370642976326692672223638908164822104832415788577945314264232531947860576966629150456995512932232264881080618006698700677529111454508900582785420549466798020451488168615035256292977390692401388790460066327347700109341639992159475755036449
n = 79832181757332818552764610761349592984614744432279135328398999801627880283610900361281249973175805069916210179560506497075132524902086881120372213626641879468491936860976686933630869673826972619938321951599146744807653301076026577949579618331502776303983485566046485431039541708467141408260220098592761245010678592347501894176269580510459729633673468068467144199744563731826362102608811033400887813754780282628099443490170016087838606998017490456601315802448567772411623826281747245660954245413781519794295336197555688543537992197142258053220453757666537840276416475602759374950715283890232230741542737319569819793988431443
e = 65537
p = 3133337
q = n//p
key_info = RSA.construct((n, e, d, p, q))
key = RSA.importKey(key_info.exportKey())  #打包密钥
key = PKCS1_OAEP.new(key)                  #创建解密
f = open('E:\\Ctf\\BMZCTF\\crypto\\2018 AFCTF 可怜的RSA\\flag.enc', 'r').read()
c = b64decode(f)
flag = key.decrypt(c)                      #通过密钥解密密文
print(flag)
#b'afctf{R54_|5_$0_B0rin9}'
```

kali下操作：

第一步：在kali中使用命令：openssl rsa -pubin -text -modulus -in warmup -in pub.pem得到E和N的值。

第二步：使用python把16进制的N转换成10进制。

第三步：分解N得出P和Q以便求出私钥(d,n)。

第四步：在kali中使用python脚本生成私钥D。

第五步：在kali终端输入openssl进入openssl输入rsautl -decrypt -in flag.enc(密文名称) -inkey private.pem(我们所求得密钥名称)得到flag。

看键盘

ujn njkm ijnmhjk fgtrdcv tgvgy njkm hjuygbn ijnmhjk

flag{internet}

i改为大写即可

山东省大学生网络技术大赛-baby（低加密指数攻击）

```python
# -*- coding: utf-8 -*-

from Crypto.PublicKey import RSA
import libnum
import uuid

flag = "flag{***************}"
rsa = RSA.generate(4096,e=3)
p = rsa.p
d = rsa.d
e = rsa.e
N = rsa.n
m = libnum.s2n(flag)
c = pow(m, e, N)
print "[+]c:",c
print "[+]N:",N
'''
[+]c: 344246784248256132370323757453790755403533762276297110321055748005034935987304162433626178273150906891000336054704994248241503686290484460048497667442360486171016603355857692143806855595194896609965890260672529255195234519313297399628856624613870875481051164681136201776906304142511571230562974834120779230569459074206697120252340530156123334199103737410126562326533207078744933299179209709004476197370590921713711964909131345720658980347979789492440201727354371992484959207032839627676038150161293403965369113166771094366231134224937826657958579219178937599421230874628798840627205579064291831558595977568908961920440032408219063325752924761600720395057717945312555422441235169296712773063614670745457208237358063080030919834276783002877094695822824665722300665801952272782147762802137222159530977474534372897344694547124261079671881095489669072378778403160098284762003883273291447838770334912387099544738099911527273336160224065174431305427131672064217870385963129751531658486257219110805612426460922990168026629130170716857405486991638360074742247154265876095493722891819778300926771283688061131314598311823905209428926706964471286314856065799438858122606408057560353775841551357701559157821200251164860615401051393396557229047212946291490250330668235998239644446207792591061769134788393701008912130721000631012326351836365523609527628386563073006211952480592536147451188521635693884180862917488051001750086583878038782000348402155065167156406211656616421773718638745860695240222586429151006155960324431450348470315643566715591792127054661456096984755462109947489491213598530942479905330750043935345654217764687858212612913094632053140578820162660663656360180844991588067170369725908484588910191715832689201806912211684536120296985102 71
'''
```

低加密指数攻击

```
# -*- coding: utf-8 -*-
#低加密指数攻击
import gmpy2
import libnum
e = 3
c = 344246784248256132370323757453790755403533762276297110321055748005034935987304162433626178273150906891000336
054704994248241503686290484460048497667442360486171016603355857692143806855595194896609965890260672529255195234 5
193132973996288566246138708754810511646811362017769063041425115712305629748341207792305694590742066971202523405 3
015612333419910373741012656232653320707874493329917920970900447619737059092171371196490913134572065898034797978 9
492440201727354371992484959207032839627676038150161293403965 3
n = 691316677109436623113422493782665795857921917893759942123087462879884062720557906429183155859597756890896192
044003240821906332575292476160072039505771794531255542244123516929671277306361467074545720823735806308003091983 4
276783002877094695822824665722300665801952272782147762802137222159530977474534372897344694547124261079671881095 4
896690723787784031600982847620038832732914478387703349123870995447380999115272733361602240651744313054271316720 6
421787038596312975153165848625721911080561242646092299016802662913017071685740548699163836007474224715426587609 5
493722891819778300926771283688061131314598311823905209428926706964471286314856065799438858122606408057560353775 8
415513577015591578212002511648606154010513933965572290472129462914902503306682359982396444462077925910617691347 8
839370100891213072100063101232635183636552360952762838656307300621195248059253614745118852163569388418086291748 8
051001750086583878038782000348402155065167156406211656616421773718638745860695240222586429151006155960324431450 3
484703156435667155917921270546614560969847554621099474894912135985309424799053307500439353456542177646878582126 1
291309463205314057882016266066365636018084499158806717036972590848458891019171583268920180691221168453612029698 5
10271
k = 0
while 1:
    result = gmpy2.iroot(c + k*n,e)
    if(result[1] == True):
        print(libnum.n2s(int(result[0])))
        break
    k = k + 1
#b'flag{4c466c3d0949118a3ca3319b43fe792bef9e94a19c8f666d2ec6c890034d88ba}'
```

2018 AFCTF MagicNum

加密代码：

```
#include <stdio.h>
char flag[]="afctf{sec_is_everywhere}";

int main()
{
 for(int i=0;i<6;++i){
  printf("%20f\n",*(float*)(flag+i*4));
 }
 return 0;
}2018 AFCTF MagicNum
```

题目给出一些浮点型数据，本题考察浮点数在内存中的存储问题

我们可以用c语言，将参数的地址（机器内存储的都是二进制）给字符型指针，再用指针取值（此时指针取地址中的值已经是二进制），这样就跳过了手动将浮点数转二进制的过程

```c
#include <stdio.h>
int main()
{   float i;
    while(1){
  scanf("%f",&i);
    char *p = &i;
    printf("%c%c%c%c\n",*(p),*(p+1),*(p+2),*(p+3));
    }
    return 0;
}
```

```
7206591051017713800000000000000000.000000
afct
7186320967081137100000000.000000
f{se
7186320967081137100000000.000000
f{se
184896826254127600000000000000000.000000
c_is
7272325758805068700000000.000000
_eve
467465916746976620000000.000000
rywh
190616988374992920000000000000000000000.000000
ere}
```

https://blog.csdn.net/weixin_45859850

afctf{sec_is_everywhere}

easy_rsa 分解模数n

已知：

```
n =
20499421483319837632829005665244953604816631094131482091599739242452461959670789327098587429656441009883765163931516947567316643569963621519243386576155541991650610105070387440479691299670503655019032377026089584152047162143622592606512093871068907193787013919967475201572411584456318069752118161110853731611597336602111728937901380008855876406951363681839727114631417566905375167058609392654378267988132283758536576123045237315624774544667706040426027925497245266590365080287798629911056879889563806490213919247917120199512548392006107613124668838850719777385822083736801474373012496703900585089950184532462833403107 e = 65537 c =
20032571908334556518706996350628353762857932090373933681400888944312785947661616694094701195862850761910097168558675875627215052773260832659893374127927598227236026530897152252167201800939030104214536327304912430809760264428512032937328302608592176040184547483423711571171744055968148657366655074788139832543212184924191715976191905198701170693941252097895416146372351355392205316913304672023350664554170422353392751177640905416228502429781714418077926868264124293539349931619123485098501005249806829896270239835682082818035333364506995823621010867979820662847712684534266471691343787498430152750115780889521627965005
```

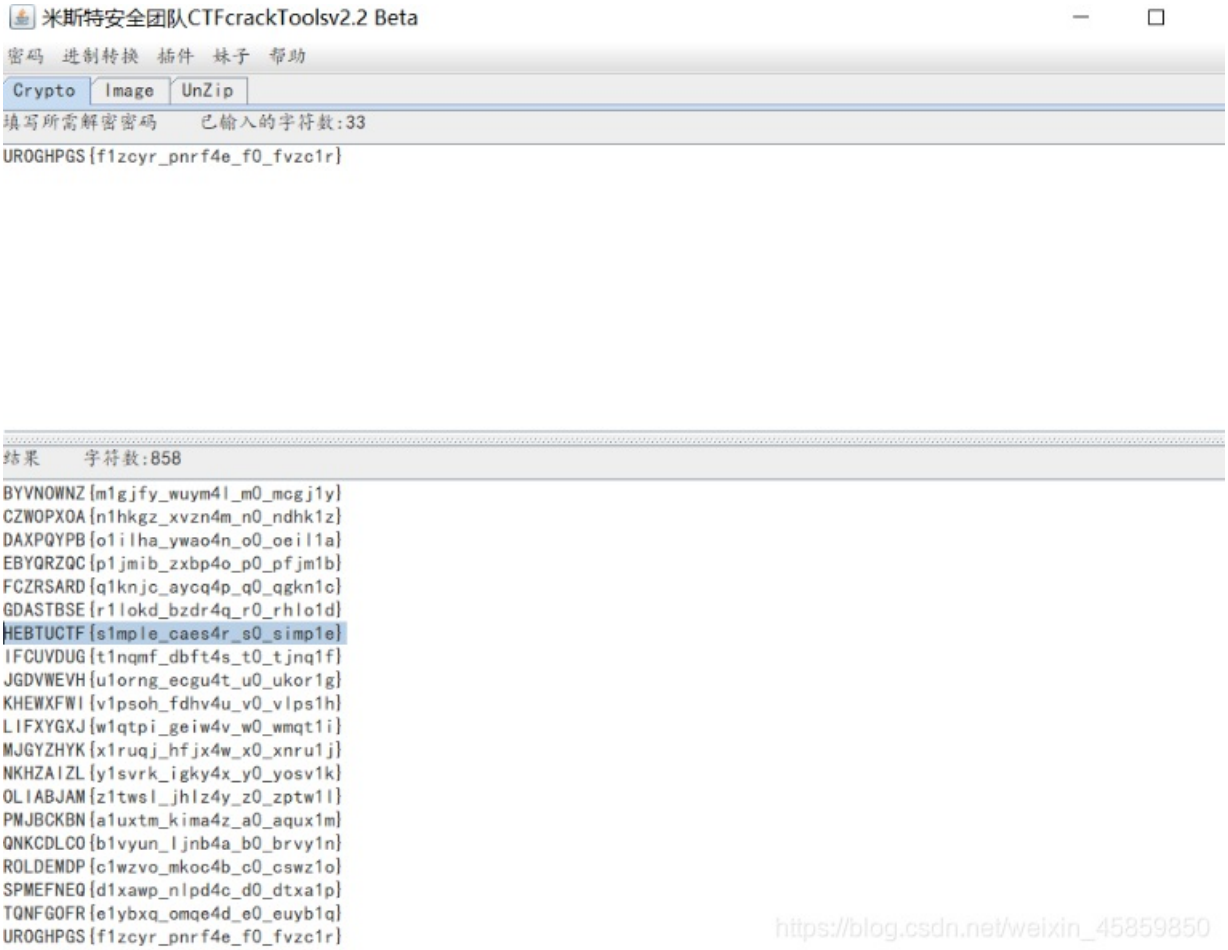
```
n = 20499421483319837632829005665244953604816631094131482091599739242452461959670789327098587429656441009883765163931516947567316643569963621519243386576155541991650610105070387440479691299670503655019032377026089584152047162143622592606512093871068907193787013919967475201572411584456318069752118161110853731611597336602111728937901380008855876406951363681839727114631417566905375167058609392654378267988132283758536576123045237315624774544667706040426027925497245266590365080287798629911056879889563806490213919247917120199512548392006107613124668838850719777385822083736801474373012496703900585089950184532462833403107
e = 65537
c = 20032571908334556518706996350628353762857932090373933681400888944312785947661616694094701195862850761910097168558675875627215052773260832659893374127927598227236026530897152252167201800939030104214536327304912430809760264428512032937328302608592176040184547483423711571171744055968148657366655074788139832543212184924191715976191905198701170693941252097895416146372351355392205316913304672023350664554170422353392751177640905416228502429781714418077926868264124293539349931619123485098501005249806829896270239835682082818035333364506995823621010867979820662847712684534266471691343787498430152750115780889521627965005
#分解n
p = 13814955814913694672370285369321779886226731666618994281652088616535726019491665403496522624661362048290501130699646565954445645187095816210781948579998714499751427835823481698626651809230358675305067121014907529617331950367792931369649905797713461724444938870656661175640192570290682002658424827844623758051 7
q = 14838571876712080829457706251985018463949561479328105289534614421625011408710288822236906556905903763624935854762835933332075497604618881756233534375247410198587969785411124659709063321435413562080841994568837407527676739117430250727922742918243680773926876937801544783445898154810996826280817970780244879927 1
import gmpy2
import libnum
d = gmpy2.invert(e,(p-1)*(q-1))
m = gmpy2.powmod(c,d,n)
print(libnum.n2s(int(m)))
#b'flag{Eeeasy_RSA!}'
```

small_rsa 同 山东省大学生网络技术大赛-baby（低加密指数攻击）

flag{4c466c3d0949118a3ca3319b43fe792bef9e94a19c8f666d2ec6c890034d88ba}

2018 HEBTUCTF Simple Caesar!

凯撒密码（rot13）



2018 AFCTF 你能看出这是什么加密么

p=0x928fb6aa9d813b6c3270131818a7c54edb18e3806942b88670106c1821e0326364194a8c49392849432b37632f0abe3f3c52e909b939c91c50e41a7b8cd00c67d6743b4f

q=0xec301417ccdffa679a8dcc4027dd0d75baf9d441625ed8930472165717f4732884c33f25d4ee6a6c9ae6c44aedad039b0b72cf42cab7f80d32b74061

e=0x10001

c=0x70c9133e1647e95c3cb99bd998a9028b5bf492929725a9e8e6d2e277fa0f37205580b196e5f121a2e83bc80a8204c99f5036a07c8cf6f96c420369b4161d2654a7eccbdaf583204b645e137b3bd15c5ce865298416fd5831cba0d947113ed5be5426b708b89451934d11f9aed9085b48b729449e461ff0863552149b965e22b6

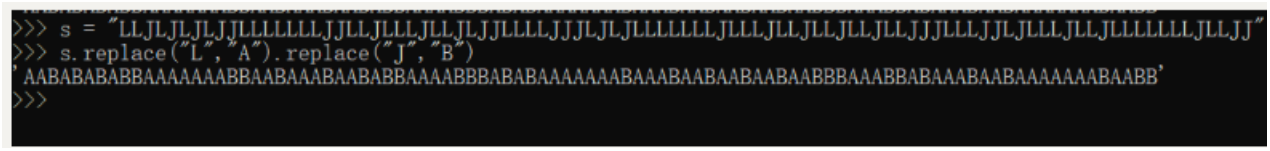
```
p=0x928fb6aa9d813b6c3270131818a7c54edb18e3806942b88670106c1821e0326364194a8c49392849432b37632f0abe3f3c52e909b939c91c50e41a7b8cd00c67d6743b4f

q=0xec301417ccdffa679a8dcc4027dd0d75baf9d441625ed8930472165717f4732884c33f25d4ee6a6c9ae6c44aedad039b0b72cf42cab7f80d32b74061

e=0x10001

c=0x70c9133e1647e95c3cb99bd998a9028b5bf492929725a9e8e6d2e277fa0f37205580b196e5f121a2e83bc80a8204c99f5036a07c8cf6f96c420369b4161d2654a7eccbdaf583204b645e137b3bd15c5ce865298416fd5831cba0d947113ed5be5426b708b89451934d11f9aed9085b48b729449e461ff0863552149b965e22b6
import gmpy2
import libnum
d = gmpy2.invert(e,(q-1)*(p-1))
m = pow(c,d,p*q)
print(libnum.n2s(int(m)))
```

b'\x02\xd3\xe4v\xea\x80r\x83\xda\x99\x88\xf5#\x08\xbbAT\x8b\xaf\xd2\xf4\xdc\x9f\xd3\xbf\xb7A\xc3\xcc\xc5`\xa1\x8b\x86\x18y\xd0&\x88\x10\xef\xbe\x83\xcer\xceC\x17\xec[\xb7%\x08\xef\x16\x1f\xab\x0c\x96\xa3\xdcN^\x8e,\xa3\x11{\x99U\xcd\x15o\xd7B\xf4L\x8f}&\xc5$\xca\xd5;\xf9\x02Y\xc1\xbbS\xfd4\x83M\x96\xa9\xbd;\x83/\xf7\x00**afctf{R54_|5_KaTeX parse error: Expected group after '_' at position 2: 0_imp13}**'

2018 HEBTUCTF easy_crypto （morse、baconian）

摩斯解密：`http://rumkin.com/tools/cipher/morse.php`

将L转为A、J转为B



培根解密：

```
http://rumkin.com/tools/cipher/baconian.php
```

When decoding, it will use "0", "A", and "a" as an "A"; "1", "B", and "b" are all equivalent as well. Other letters are ignored.

Decrypt

Distinct codes

Your message: (Swap A and B)

AABABABABBAAAAAAABBAABAAABAABABBAAAABBBABABAAAAAAABAAABAABAABAABAABBBAAABBABAAABAABAAAAAAABAABB

This is your encoded or decoded text:

FLAGISYOUARESOGREAT

https://blog.csdn.net/weixin_45859850

Bacon Encode_Decode System

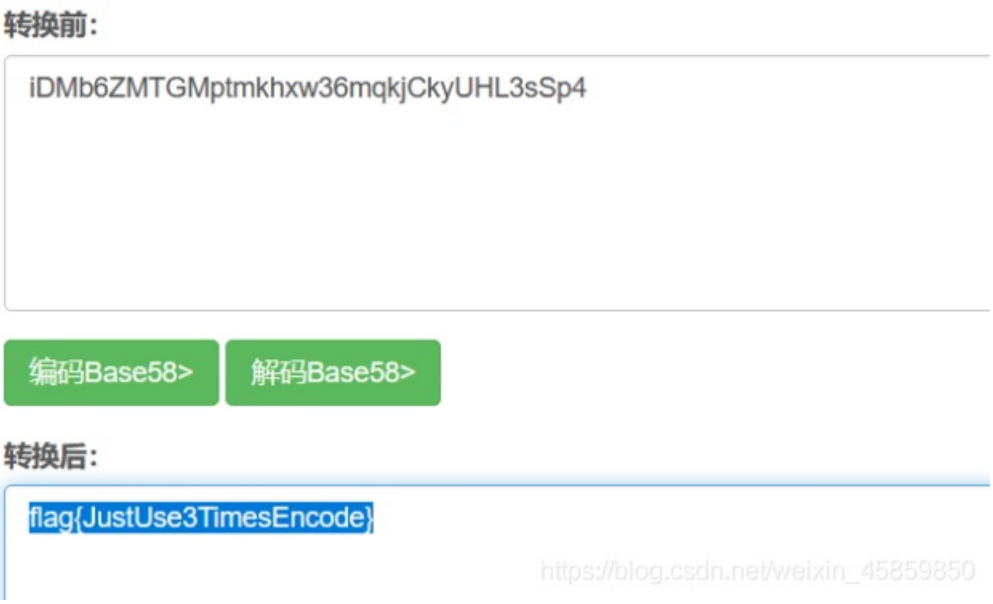
```
# coding:utf8

import re

alphabet = ['a','b','c','d','e','f','g','h','i','j','k','l','m','n','o','p','q','r','s','t','u','v','w','x','y',
'z']

first_cipher = ["aaaaa","aaaab","aaaba","aaabb","aabaa","aabab","aabba","aabbb","abaaa","abaab","ababa","ababb",
"abbaa","abbab","abbba","abbbb","baaaa","baaab","baaba","baabb","babaa","babab","babba","babbb","bbaaa","bbaab"]

second_cipher = ["aaaaa","aaaab","aaaba","aaabb","aabaa","aabab","aabba","aabbb","abaaa","abaaa","abaab","ababa"
,"ababb","abbaa","abbab","abbba","abbbb","baaaa","baaab","baaba","baabb","baabb","babaa","babab","babba","babbb"
]

def encode():
    upper_flag = False # 用于判断输入是否为大写
    string = input("please input string to encode:\n")
    if string.isupper():
        upper_flag = True
        string = string.lower()
    e_string1 = ""
    e_string2 = ""
    for index in string:
        for i in range(0,26):
            if index == alphabet[i]:
                e_string1 += first_cipher[i]
                e_string2 += second_cipher[i]
                break
    if upper_flag:
        e_string1 = e_string1.upper()
        e_string2 = e_string2.upper()
    print("first encode method result is:\n"+e_string1)
    print("second encode method result is:\n"+e_string2)
    return


def decode():
    upper_flag = False  # 用于判断输入是否为大写
    e_string = input("please input string to decode:\n")
    if e_string.isupper():
        upper_flag = True
        e_string = e_string.lower()
    e_array = re.findall(".{5}",e_string)
    d_string1 = ""
```

```
    d_string2 = ""
    for index in e_array:
        for i in range(0,26):
            if index == first_cipher[i]:
                d_string1 += alphabet[i]
            if index == second_cipher[i]:
                d_string2 += alphabet[i]
    if upper_flag:
        d_string1 = d_string1.upper()
        d_string2 = d_string2.upper()
    print("first decode method result is:\n"+d_string1)
    print("second decode method result is:\n"+d_string2)
    return


if __name__ == '__main__':
    print("\t\tcoding by qux")
    while True:
        print("\t*******Bacon Encode_Decode System*******")
        print("input should be only lowercase or uppercase,cipher just include a,b(or A,B)")
        print("1.encode\n2.decode\n3.exit")
        s_number = input("please input number to choose\n")
        if s_number == "1":
            encode()
            input()
        elif s_number == "2":
            decode()
            input()
        elif s_number == "3":
            break
        else:
            continue
```

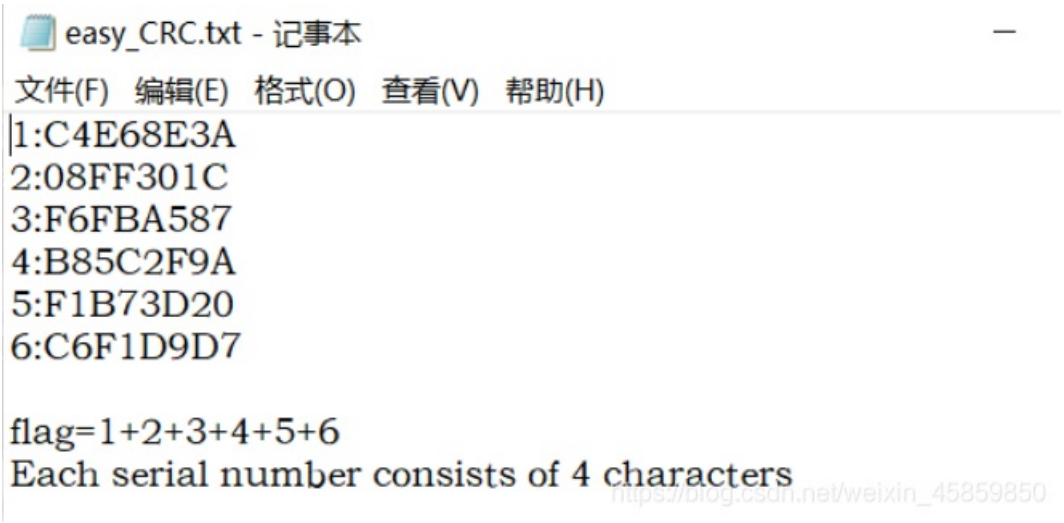
看的出来吗

经过base64解码、base91解码、base58解码得到

`https://base64.supfree.net/`
`http://ctf.ssleye.com/base91.html`
`http://ctf.ssleye.com/base58w.html`



easy_CRC



crc32爆破

十六进制转字符并整合：

```
s = "0x62, 0x33, 0x32, 0x33, 0x31, 0x65, 0x39, 0x34, 0x66, 0x30, 0x34, 0x35, 0x65, 0x37, 0x36, 0x65, 0x35, 0x39,
 0x33, 0x32, 0x36, 0x36, 0x66, 0x34"
s = s.split(', ')
flag = ''
for i in range(len(s)):
    flag += chr(int(s[i][2:],16))
    print(flag)
```

```
b3231e94f045e7
b3231e94f045e76
b3231e94f045e76e
b3231e94f045e76e5
b3231e94f045e76e59
b3231e94f045e76e593
b3231e94f045e76e5932
b3231e94f045e76e59326
b3231e94f045e76e593266
b3231e94f045e76e593266f
b3231e94f045e76e593266f4
```

https://blog.csdn.net/weixin_45859850

键盘之争

键盘布局有qwerty、dvorak、colemak

此题考察的是将qwerty布局转为dvorak

https://www.cnblogs.com/zhangshenjia/archive/2012/04/11/qwerty_dvorak_colemak.html


```
dic={r"'":"q",

r",":"w",

r".":"e",

"p":"4",

"y":"t",

"f":"y",
```

```
"g":"u",

"c":"i",

"r":"o",

"l":"p",

r"/":r"[",

r"=":r"]",

r'"':'Q',

r"<":"W",

r">":"E",

"P":"R",

"Y":"T",

"F":"Y",

"G":"U",

"C":"I",

"R":"O",

"L":"P",

r"?":r"{",

r"+":r"}",

"a":"a",

"A":"A",

"o":"s",

"O":"S",

"e":"d",

"E":"D",

"u":"f",

"U":"F",

"i":"g",

"I":"G",

"d":"h",

"D":"H",
```

```
"h":"j",
"H":"J",
"t":"k",
"T":"K",
"n":"l",
"N":"L",
"s":";",
"S":":",
r"-":r"'",
r'_':r'"',
r";":"z",
r":":"Z",
"q":"x",
"Q":"X",
"j":"c",
"J":"C",
"k":"v",
"K":"V",
"x":"b",
"X":"B",
"b":"n",
"B":"N",
"m":"m",
"M":"M",
"w":r",",
"W":r"<",
"v":r".",
"V":r">",
"z":r"/",
```

```
"Z":r"?",

r'!':"!",

r"@":r"@",

r"#":r"#",

r"$":r"$",

r"%":r"%",

r"^":r"^",

r"&":r"&",

r"*":r"*",

r"(":r"(",

r")":r")",

r"[":r"-",

r"]":r"=",

r"{":r"_",

r"}":r"+"}

s=r'ypau_kjg;"g;"ypau+'
for i in s:
 print (" ".join([key for key, value in dic.items() if value == i]),end='')
#flag{this_is_flag}
```

转md5

```
import hashlib

m = hashlib.md5()

m.update(*b*'this_is_flag')

print("flag{" + m.hexdigest() + "}")
```

flag{951c712ac2c3e57053c43d80c0a9e543}

2018 AFCTF Single

密文：

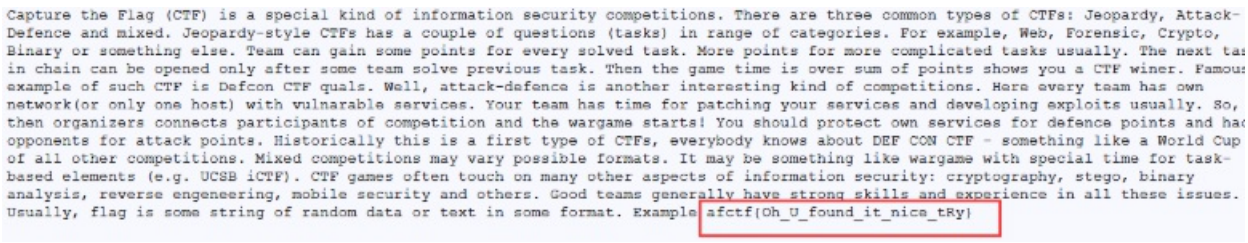
> Jmqrida rva Lfmz (JRL) eu m uqajemf seny xl enlxdomrexn uajiderc jxoqarerexnu. Rvada mda rvdaa jxooxn rcqau xl JRLu: Paxqmdyc, Mrrmjs-Yalanja mny oekay.
> Paxqmdyc-urcfa JRLu vmu m jxiqfa xl giaurexnu (rmusu) en dmnza xl jmrazxdeau. Lxd akmoqfa, Wab, Lxdanuej, Jdcqrx, Benmdc xd uxoarvenz afua. Ramo jmn zmen uxoa qxenru lxd atadc uxftay rmus. Oxda qxenru lxd oxda jxoqfejmray rmusu iuimffc. Rva nakr rmus en jvmen jmn ba xqanay xnfc mlrad uxoa ramo uxfta qdatexiu rmus. Rvan rva zmoa reoa eu xtad uio xl qxenru uvxwu cxi m JRL wenad. Lmoxiu akmoqfa xl uijv JRL eu Yaljxn JRL gimfu.
> Waff, mrrmjs-yalanja eu mnxrvad enradaurenz seny xl jxoqarerexnu. Vada atadc ramo vmu xwn narwxds(xd xnfc xna vxur) werv tifnmdmbfa uadtejau. Cxid ramo vmu reoa lxd qmrjvenz cxid uadtejau mny yatafxqenz akqfxeru iuimffc. Ux, rvan xdzmnehadu jxnnajru qmdrejeqmnru xl jxoqarerexn mny rva wmdzmoa urmdru! Cxi uvxify qdxrajr xwn uadtejau lxd yalanja qxenru mny vmjs xqqxnanru lxd mrrmjs qxenru. Veurxdejmffc rveu eu m ledur rcqa xl JRLu, atadcbxyc snxwu mbxir YAL JXN JRL - uxoarvenz fesa m Wxdfy Jiq xl mff xrvad jxoqarerexnu.
> Oekay jxoqarerexnu omc tmdc qxuuebfa lxdomru. Er omc ba uxoarvenz fesa wmdzmoa werv uqajemf reoa lxd rmus-bmuay afaoanru (a.z. IJUB eJRL).
> JRL zmoau xlran rxijv xn omnc xrvad muqajru xl enlxdomrexn uajiderc: jdcqrxzdmqvc, urazx, benmdc mnmfcueu, datadua anzanaadenz, oxbefa uajiderc mny xrvadu. Zxxy ramou zanadmffc vmta urdxnz useffu mny akqadeanja en mff rvaua euuiau.
> Iuimffc, lfmz eu uxoa urdenz xl dmnyxo ymrm xd rakr en uxoa lxdomr. Akmoqfa mljrl{Xv_l_lxiny_er_neja_rDc}

加密程序：

```
#include <bits/stdc++.h>
using namespace std;
int main()
{
 freopen("Plain.txt","r",stdin);
 freopen("Cipher.txt","w",stdout);
 map<char, char> f;
 int arr[26];
 for(int i=0;i<26;++i){
  arr[i]=i;
 }
 random_shuffle(arr,arr+26);
 for(int i=0;i<26;++i){
  f['a'+i]='a'+arr[i];
  f['A'+i]='A'+arr[i];
 }
 char ch;
 while((ch=getchar())!=EOF){
  if(f.count(ch)){
   putchar(f[ch]);
  }else{
   putchar(ch);
  }
 }
 return 0;
}
```

发现随机加密，找不到规律

进行词频分析：`https://quipqiup.com/`

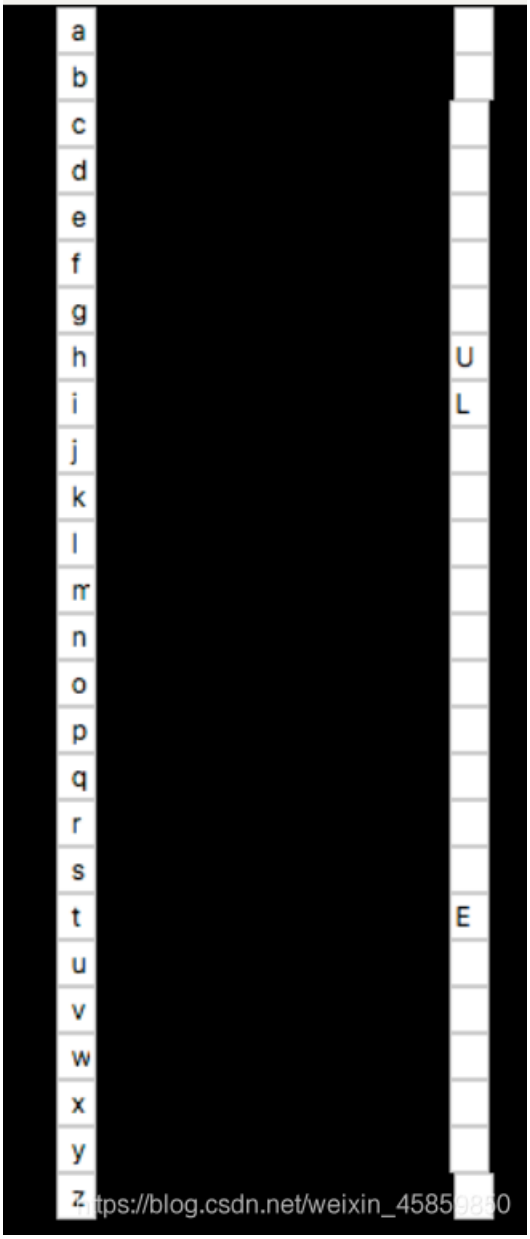


HEBTUCTF WIFNVDSFFDYUBPAMEMS

题目：

`Looks like I’m too lazy to complete it. >_< Can you help me a little?

EULS LS I XCISSLXIC SHNEHNXH: EUH VWLXF YDAKN OAR MWBJS AZHD EUH CIPT GAQ! OLQUE, ING TAW BIT GLH. DWN, ING TAW’CC CLZH IE CHISE I KULCH. ING GTLNQ LN TAWD YHGS BINT THIDS ODAB NAK. KAWCG TAW YH KLCCLNQ EA EDIGH? ICC EUH GITS ODAB EULS GIT EA EUIE, OAD ANH XUINXH, MWSE ANH XUINXH, EA XABH YIXF UHDH ING EHCC AWD HNHBLHS EUIE EUHT BIT EIFH AWD CLZHS, YWE EUHT’CC NHZHD EIFH AWD OCIQ!OCIQ LS UHYEWXEO KLONZGSOOGTWYJIBHBS.`



以为时图片隐写，试了下无果

词频分析拿到flag:https://quipqiup.com/

```
Iffws iawt A'j yff iovb yf lfjzityt ay. >_< Lon bfh etiz
jt o iayyit? THIS IS A CLASSICAL SENTENCE: THE QUICK BROWN
FOX JUMPS OVER THE LAZY DOG! FIGHT, AND YOU MAY DIE. RUN,
AND YOU'LL LIVE AT LEAST A WHILE. AND DYING IN YOUR BEDS
MANY YEARS FROM NOW. WOULD YOU BE WILLING TO TRADE? ALL
THE DAYS FROM THIS DAY TO THAT, FOR ONE CHANCE, JUST ONE
CHANCE, TO COME BACK HERE AND TELL OUR ENEMIES THAT THEY
MAY TAKE OUR LIVES, BUT THEY'LL NEVER TAKE OUR FLAG!FLAG
IS HEBTUCTF WIFNVDSFFDYUBPAMEMS
```

https://blog.csdn.net/weixin_45859850

Crypto_easy_crypto 拆分模数得到好求的phi

已知：

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

def keygen(nbit):
    while True:
        p, q, r = [getPrime(nbit) for _ in range(3)]
        if isPrime(p + q + r):
            pubkey = (p * q * r, p + q + r)
            privkey = (p, q, r)
            return pubkey, privkey

def encrypt(msg, pubkey):
    enc = pow(bytes_to_long(msg.encode('utf-8')), 0x10001, pubkey[0] * pubkey[1])
    return enc

nbit = 512

pubkey, _ = keygen(nbit)
print('pubkey =', pubkey)

enc = encrypt(flag, pubkey)
print('enc =', enc)

#pubkey = (763929224901239050077647342425144363318006219360210465113458622937882119766705458273788947718911994090187932947694326848868575500906975653763043489419853300731695950407389203582333794360159494405111004208058198680365172060235321945875374635278292661761319773173838900295933943437761251440819434675631450032782188481758975272071015244168751016874668742536151359822585793537758876078350987062972237271763834743266722655055439868971805843593929839097963319788083763,
27750416681837900468631425875797804482827864226099175414717825894823303299159277202974070903630276868248755642608884903122768656427165401563920443482151217)
#enc = 75792946030351358172345011312562823242401324242720009030358010738472229173060927926104068764516981219562320470316394487673234750881703578636534350967466406093255250353293286265621430496561242412635766499748205338001614326669463272264180368261461602882731755218646871227208367955966350677611465199404418177657321603820184344657767493722091812128172471874746852461561933818220447727466546901267467157281665272749336347504434786828126640220094443484489557202265794687819277761333360455251492033841378598681530778207081671460524290147906609215869586607599493998265681367558165634687391111237612883182763790250187088835 21
```

已知公钥分为俩部分，求私钥（d,n）即可

这里直接求（p-1)(q-1)(r-1)(p+q+r-1)不好求，求p+q+r-1即可

证明过程：`https://blog.cryptohack.org/cryptoctf2020`

decode.py:

```
#pubkey[0] = (763929224901239050077647342425144363318006219360210465113458622937882119766705458273788947718911994090187932947694326848868575500906975653763043489419853300731695950407389203582333794360159494405111004208058198680365172060235321945875374635278292661761319773173838900295933943437761251440819434675631450032782188481758975272071015244168751016874668742536151359822585793537758876078350987062972237271763834743266722655055439868971805843593929839097963319788083763,
#pubkey[1] = 27750416681837900468631425875797804482827864226099175414717825894823303299159277202974070903630276868248755642608884903122768656427165401563920443482151217)
enc = 757929460303513581723450113125628232424013242427200090303580107384722291730609279261040687645169812195623204703163944876732347508817035786365343509674664060932552503532932862656214304965612424126357664997482053380016143266694632722641803682614616028827317552186468712272083679559663506776114651994044181776573216038201843446577674937220918121281724718747468524615619338182204477274665469012674671572816652727493363475044347868281266402200944434844895572022657946878192777613333604552514920338413785986815307782070816714605242901479066092158695866075994939982656813675581656346873911112376128831827637902501870888352 1
#已知c、e、n = q*p*r、k=p+q+r
#
import gmpy2
import libnum
n = 763929224901239050077647342425144363318006219360210465113458622937882119766705458273788947718911994090187932947694326848868575500906975653763043489419853300731695950407389203582333794360159494405111004208058198680365172060235321945875374635278292661761319773173838900295933943437761251440819434675631450032782188481758975272071015244168751016874668742536151359822585793537758876078350987062972237271763834743266722655055439868971805843593929839097963319788083763
k = 27750416681837900468631425875797804482827864226099175414717825894823303299159277202974070903630276868248755642608884903122768656427165401563920443482151217
e = int(0x10001)
d = gmpy2.invert(e,k-1)
m = gmpy2.powmod(enc,d,k)
print(libnum.n2s(int(m)))
#b'BMZCTF{9d8ef46w59c4-4s5w-8d4a-w9d5-5dq78e9d}'
```

Crypto_xor DefCamp CTF 2020 – why-xor

题目：

```
xored = ['\x00', '\x00', '\x00', '\x18', 'C', '_', '\x05', 'E', 'V', 'T', 'F', 'U', 'R', 'B', '_', 'U', 'G', '_', 'V', '\x17', 'V', 'S', '@', '\x03', '[', 'C', '\x02', '\x07', 'C', 'Q', 'S', 'M', '\x02', 'P', 'M', '_', 'S', '\x12', 'V', '\x07', 'B', 'V', 'Q', '\x15', 'S', 'T', '\x11', '_', '\x05', 'A', 'P', '\x02', '\x17', 'R', 'Q', 'L', '\x04', 'P', 'E', 'W', 'P', 'L', '\x04', '\x07', '\x15', 'T', 'V', 'L', '\x1b']
s1 = ""
s2 = ""
# ['\x00', '\x00', '\x00'] at start of xored is the best hint you get
a_list = [chr(ord(a) ^ ord(b)) for a,b in zip(s1, s2)]
print(a_list)
print("".join(a_list))
```

提示前三个字符异或后为\x00 'ctf' ^ 'ctf' == '\x00\x00\x00'

解题：

```
ls = ['\x00', '\x00', '\x00', '\x18', 'C', '_', '\x05', 'E', 'V', 'T', 'F', 'U', 'R', 'B', '_', 'U', 'G', '_', 'V', '\x17', 'V', 'S', '@', '\x03', '[', 'C', '\x02', '\x07', 'C', 'Q', 'S', 'M', '\x02', 'P', 'M', '_', 'S', '\x12', 'V', '\x07', 'B', 'V', 'Q', '\x15', 'S', 'T', '\x11', '_', '\x05', 'A', 'P', '\x02', '\x17', 'R', 'Q', 'L', '\x04', 'P', 'E', 'W', 'P', 'L', '\x04', '\x07', '\x15', 'T', 'V', 'L', '\x1b']
secret = "ctf" * (len(ls)//3)
print(secret)
a_list = [chr(ord(a) ^ ord(b)) for a,b in zip(ls, secret)]
print(a_list)
print("".join(a_list))
#ctf{79f107231696395c004e87dd7709d3990f0d602a57e9f56ac428b31138bda258}
```

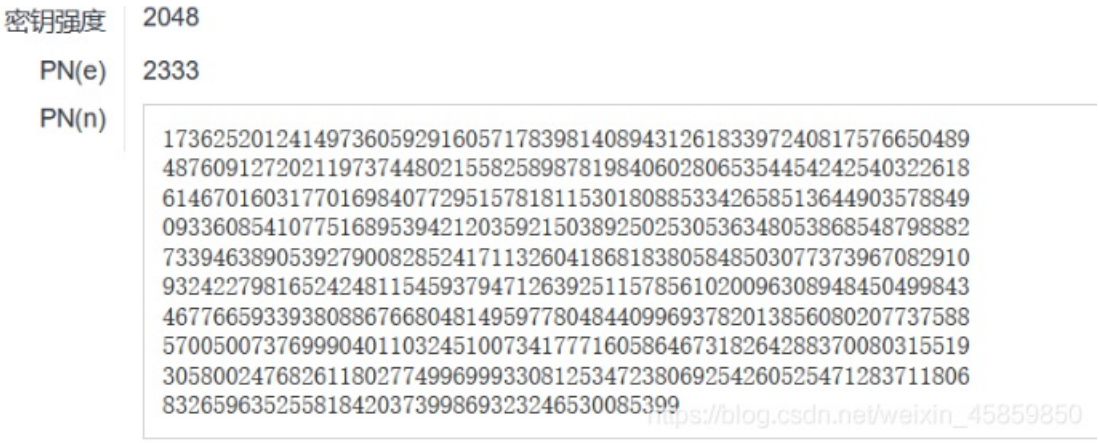
rsass

压缩包弱口令：password

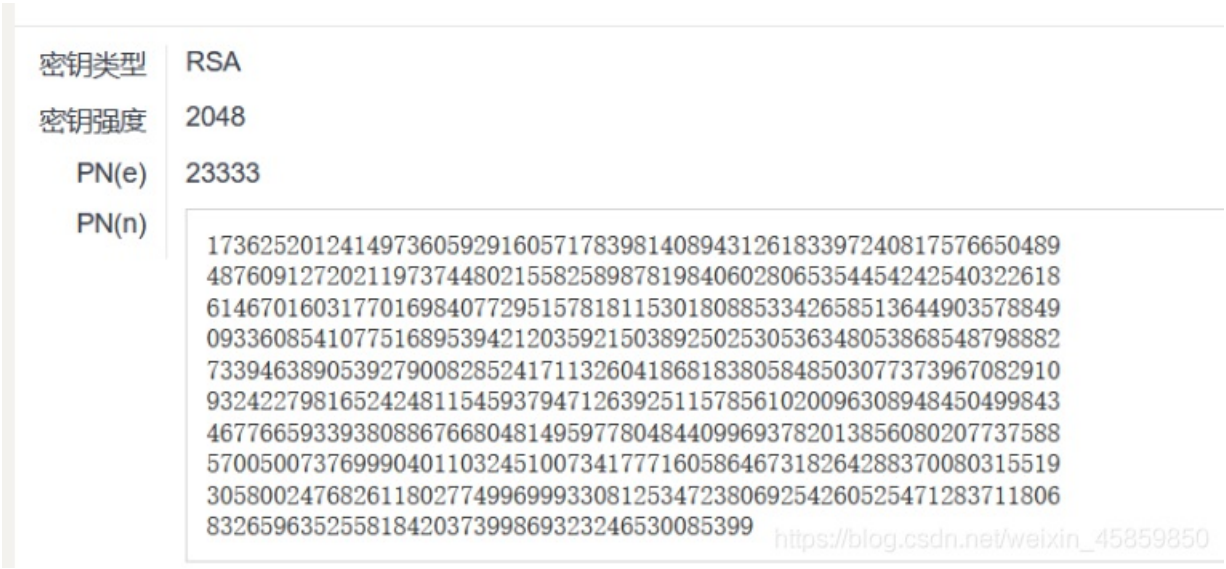
得到 俩个公钥和对应密文

公钥解析：http://ctf.ssleye.com/pub_asys.html

pubkey1



pubkey2



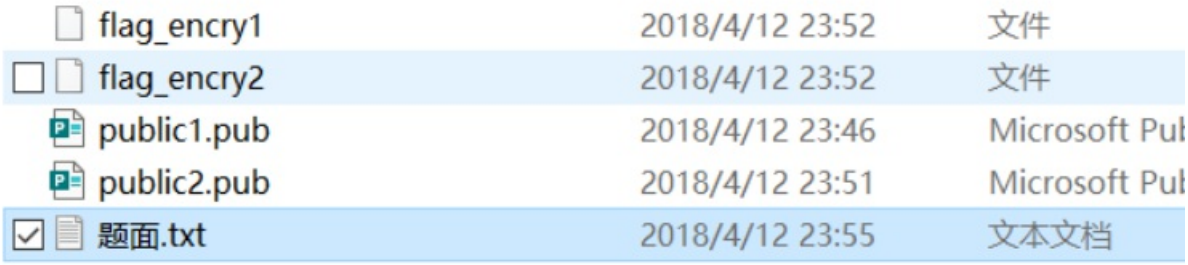
发现对公钥中的模数n相同，使用共模攻击

注意：密文用base64编码了，需要解密再转十进制

```python
from gmpy2 import invert
import libnum
import base64
import os
import sys
#path = os.getcwd()#获取当前路径
path = sys.path[0] #获取脚本路径
def gongmo(n, c1, c2, e1, e2):
    def egcd(a, b):
        #扩展欧拉定理
        if b == 0:
            return a, 0
        else:
            x, y = egcd(b, a % b)
            return y, x - (a // b) * y
    s = egcd(e1, e2)
    s1 = s[0]
    s2 = s[1]

    # 求模反元素（逆元）
    if s1 < 0:
        s1 = - s1
        c1 = invert(c1, n)
    elif s2 < 0:
        s2 = - s2
        c2 = invert(c2, n)
    m = pow(c1, s1, n) * pow(c2, s2, n) % n
    return m
n = 17362520124149736059291605717839814089431261833972408175766504894876091272021197374480215582589878198406028065354454242540322618614670160317701698407729515781811530180885334265851364490357884909336085410775168953942120359215038925025305363480538685487988827339463890539279008285241711326041868183805848503077373967082910932422798165242481154593794712639251157856102009630894845049984346776659339380886766804814959778048440996937820138560802077375885700500737699904011032451007341777160586467318264288370080315519305800247682611802774996999330812534723806925426052547128371180683265963525581842037399869323246530085399
with open(path + '.\\rsa\\flag1.enc','rb') as f1:
    c1 = f1.read()
    c1 = base64.b64decode(c1)
    c1 = libnum.s2n(c1)
    print(c1)
with open(path + '.\\rsa\\flag2.enc','rb') as f2:
    c2 = f2.read()
    c2 = base64.b64decode(c2)
    c2 = libnum.s2n(c2)
e1 = 2333
e2 = 23333

result = gongmo(n, c1, c2, e1, e2)
print(result)
print(libnum.n2s(int(result)))
#b'flag{4b0b4c8a-82f3-4d80-902b-8e7a5706f8fe}'
```

去掉-即flag

2018 AFCTF One Secret, Two encryption



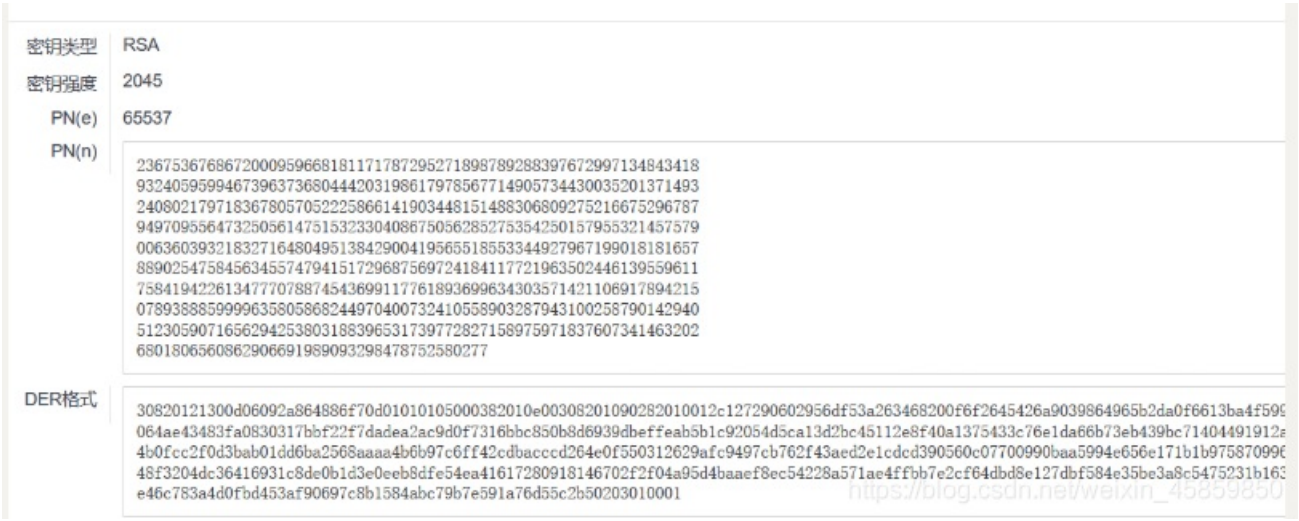
题目给出了俩组密文和公钥，提示了素数生成慢、偷懒、用相同密文（解一个即可）

猜测两个模数n公用了一个素数

分别进行公钥提取：`http://ctf.ssleye.com/pub_asys.html`

public1:

public2:



求俩个n的最大公约数得到其中一个素数即可解题

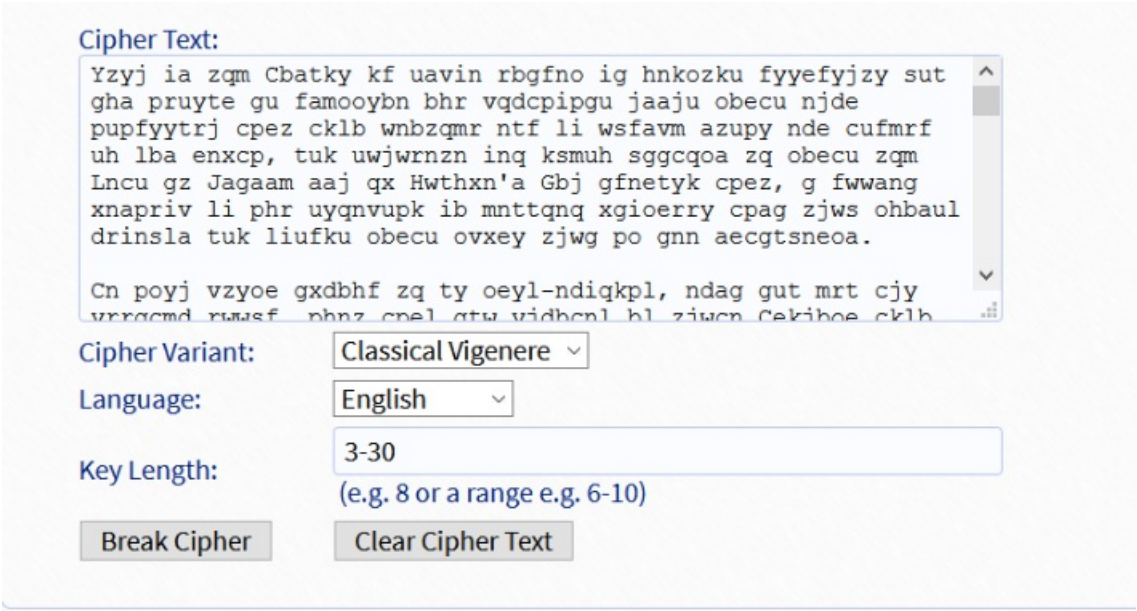
decode.py

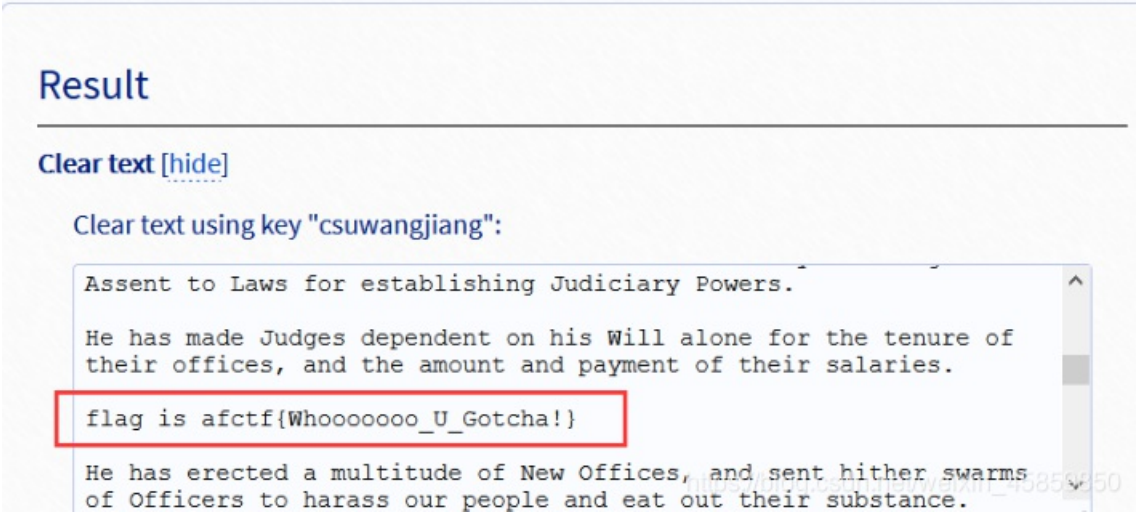
```
n1 = 485029713816222346882648162308244024957913687679831265273520469868961396900863254522097669917030845408239083474257071824780420206092949357164207467942856516840587711068151810566730178565351769768449098237507898988604045111508212092898258838091460927300815397790795053249860548622588397364314151602405831536057298874460713411025448942151602693724916349398268133662872603348912470565721776822905848715586526508042748802892187960833889893354082556488901216618134617727663982834637636216893420882246729567376187696586457316452933688525057735776731425658101947413065141210089783960649118942437395924402369566965321349832964
e1 = 166662663296036823900115940804776599127025004220624415744717118819565730293301950193210177799951000123573633884310770987178590674939300425761412980206108115586143372238014500153718114261351529013883576523600281168998647228076240815717643750302175306158874652043372073460895363911155855693049072151757999449308855101305083569001977260074431739821818388340219206048097997945646993786325778136252118457814212944122428832106721725409309113975986436241662107879085361014650716439042856013203440242834878648506244428367706708431121109714505981728529818874621868624754285069693368779495316600601299037277003994790396589299
n2 = 236753676867200095966818117178729527189878928839767299713484341893240595994673963736804442031986179785677149057344300352013714932408021797183678057052225866141903448151488306809275216675296787949709556473250561475153233040867505628527535425015795532145757900636039321832716480495138429004195655185533449279671990181816578890254758456345574794151729687569724184117721963502446139559611758419422613477707887454369911776189369963430357142110691789421507893888599996358058682449704007324105589032879431002587901429405123059071656294253803188396531739772827158975971837607341463202680180656086290669198909329847875258027 7
e2 = 65537
#n1 和 n2极有可能有公约数
import gmpy2
import libnum
p = gmpy2.gcd(n1,n2)
q1 = n1//p
q2 = n2//p
d1 = gmpy2.invert(e1,(p-1)*(q1-1))
d2 = gmpy2.invert(e2,(p-1)*(q2-1))
with open("..\\flag_encry1","rb") as f1:
    c1 = f1.read()
    c1 = libnum.s2n(c1)
with open("..\\flag_encry2","rb") as f2:
    c2 = f2.read()
    c2 = libnum.s2n(c2)
m1 = gmpy2.powmod(c1,d1,n1)
m2 = gmpy2.powmod(c2,d2,n2)
print(libnum.n2s(int(m1)))
print(libnum.n2s(int(m2)))
```

```
06\xae\x16\x98mqE4\x96\xf8Z\xd8\xef\xdb\xd6(\xb2Ws\xe7u\x95\xd5I\xddn\xdf\xb0\xe9\xa0\xef\xe8\x0c\x97B\xc7\xe1\x
nSSL is widely used\r\nflag is afctf{You_Know_0p3u55I}'
b'\x02~\x83\xa7\xed\xd5\xde\xf1\xb8\xc6\x17%5:\xf7wZn\xc2WbDn)\x070\xfc\x98}\xa5\x96@\x90?Y\xba\xe5\xca\xdam\xba
d\x94\x1c#\xecti\x8b\x89\x18hH\xc5\xbf\x10\xe983C}3Cz>HbX\xbe\x98<JG\x86??\xe2?\xfc\xaf\xb60\xbe\xec\xe3h\x07\xd
b1t\x9bp\x03y\x12\xf2<\x99\xae\xf6\xde\x9b\xdf\xdd\xb4\xf7\x88t\xe0\xff% \x11"B\xa7r\xf2}\xf3\xaf\xca\x9dfI.\x08
E\xed\xb38\xe5\xcc\xc3#\x1eT*\xec\xc4a\x95\xd9\xd9\xe7\xd4\x88O\xa3\'\xae9S\xa3\xd1\x7f~+\xefqa\xe1y\x82\x19l\x8
\xc1\xbd\t\x88\xbf+Th\xba\x8f\xfd\xb0\x89\n\x92\x00OpenSSL is widely used\r\nflag is afctf{You_Know_0p3u55I}'
```

2018 AFCTF Vigenère

vigenere暴力破解：`https://www.guballa.de/vigenere-solver`





afctf{Whooooooo_U_Gotcha!}

参考：`https://www.programmersought.com/article/62834774254/`

encode

```c
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

int main()
{
 freopen("flag.txt","r",stdin);
 freopen("flag_encode.txt","w",stdout);
 char key[] = /*SADLY SAYING! Key is eaten by Monster!*/;
 int len = strlen(key);
 char ch;
 int index = 0;
 while((ch = getchar()) != EOF){
  if(ch>='a'&&ch<='z'){
   putchar((ch-'a'+key[index%len]-'a')%26+'a');
   ++index;
  }else if(ch>='A'&&ch<='Z'){
   putchar((ch-'A'+key[index%len]-'a')%26+'A');
   ++index;
  }else{
   putchar(ch);
  }
 }
 return 0;
}
```

decode

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

int main()
{
 freopen("flag.txt", "w", stdout);
 freopen("flag_encode.txt", "r", stdin);

    //char encoded[] = "luigvycxwpf";
 //char flag[] = "flagisafctf";
 //char key[] = "enereahavige";
      //ursanuyigea   gjiangcsuwa qfimdqccyik gjiangcsuwa
 char key[] = "csuwangjiang";

 int len = strlen(key);
 char ch;
 int index = 0;
 while ((ch = getchar()) != EOF) {
  if (ch >= 'a' && ch <= 'z') {

   for (int i = 0; i < 50; i++)
   {
    char test = (ch - 'a') + 26 * i + 'a' - (key[index % len] - 'a');
    if (test >= 'a' && test <= 'z')
    {
     putchar(test);
     break;
    }
   }
   ++index;
  }
  else if (ch >= 'A' && ch <= 'Z') {
   for (int i = 0; i < 50; i++)
   {
    char test = (ch - 'A') + 26 * i + 'A' - (key[index % len] - 'a');
    if (test >= 'A' && test <= 'Z')
    {
     putchar(test);
     break;
    }
   }
   ++index;
  }
  else {
   putchar(ch);
  }
 }
 return 0;
}
```

2018 AFCTF 你听过一次一密么？ Many-Time-Pad流密码加密

https://github.com/Jwomers/many-time-pad-attack/blob/master/attack.py

decode

python2在线：http://www.dooccn.com/python/

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

```
## OTP - Recovering the private key from a set of messages that were encrypted w/ the same private key (Many tim
e pad attack) - crypto100-many_time_secret @ alexctf 2017
# Original code by jwomers: https://github.com/Jwomers/many-time-pad-attack/blob/master/attack.py)

import string
import collections
import sets, sys

# 11 unknown ciphertexts (in hex format), all encrpyted with the same key

c1='25030206463d3d393131555f7f1d061d4052111a19544e2e5d'
c2='0f020606150f203f307f5c0a7f24070747130e16545000035d'
c3='1203075429152a7020365c167f390f1013170b1006481e1314'
c4='0f4610170e1e2235787f7853372c0f065752111b15454e0e09'
c5='081543000e1e6f3f3a3348533a270d064a02111a1b5f4e0a18'
c6='0909075412132e247436425332281a1c561f04071d520f0b11'
c7='4116111b101e2170203011113a69001b475206011552050219'
c8='0410060646122970203754533342c17545a01451811411a470e'
c9='021311114a5b0335207f7c167f22001b44520c15544801125d'
c10='061406114 60c26243c7f5c167f3d015446010053005907145d'
c11='0f05110d160f263f3a7f4210372c03111313090415481d49'
ciphers = [c1, c2, c3, c4, c5, c6, c7, c8, c9, c10, c11]
# The target ciphertext we want to crack
#target_cipher = "0529242a631234122d2b36697f13272c207f2021283a6b0c7908"

# XORs two string
def strxor(a, b):     # xor two strings (trims the longer input)
    return "".join([chr(ord(x) ^ ord(y)) for (x, y) in zip(a, b)])

def target_fix(target_cipher):
 # To store the final key
 final_key = [None]*150
 # To store the positions we know are broken
 known_key_positions = set()

 # For each ciphertext
 for current_index, ciphertext in enumerate(ciphers):
  counter = collections.Counter()
  # for each other ciphertext
  for index, ciphertext2 in enumerate(ciphers):
   if current_index != index: # don't xor a ciphertext with itself
    for indexOfChar, char in enumerate(strxor(ciphertext.decode('hex'), ciphertext2.decode('hex'))): # Xor the t
wo ciphertexts
     # If a character in the xored result is a alphanumeric character, it means there was probably a space chara
cter in one of the plaintexts (we don't know which one)
     if char in string.printable and char.isalpha(): counter[indexOfChar] += 1 # Increment the counter at this i
ndex
  knownSpaceIndexes = []

  # Loop through all positions where a space character was possible in the current_index cipher
  for ind, val in counter.items():
   # If a space was found at least 7 times at this index out of the 9 possible XORS, then the space character wa
s likely from the current_index cipher!
   if val >= 7: knownSpaceIndexes.append(ind)
  #print knownSpaceIndexes # Shows all the positions where we now know the key!

  # Now Xor the current_index with spaces, and at the knownSpaceIndexes positions we get the key back!
  xor_with_spaces = strxor(ciphertext.decode('hex'),' '*150)
  for index in knownSpaceIndexes:
```

```
    # Store the key's value at the correct position
    final_key[index] = xor_with_spaces[index].encode('hex')
    # Record that we known the key at this position
    known_key_positions.add(index)

# Construct a hex key from the currently known key, adding in '00' hex chars where we do not know (to make a co
mplete hex string)
final_key_hex = ''.join([val if val is not None else '00' for val in final_key])
# Xor the currently known key with the target cipher
output = strxor(target_cipher.decode('hex'),final_key_hex.decode('hex'))

print "Fix this sentence:"
print ''.join([char if index in known_key_positions else '*' for index, char in enumerate(output)])+"\n"

# WAIT.. MANUAL STEP HERE
# This output are printing a * if that character is not known yet
# fix the missing characters like this: "Let*M**k*ow if *o{*a" = "cure, Let Me know if you a"
# if is too hard, change the target_cipher to another one and try again
# and we have our key to fix the entire text!

#sys.exit(0) #comment and continue if u got a good key

target_plaintext = "cure, Let Me know if you a"
print "Fixed:"
print target_plaintext+"\n"

key = strxor(target_cipher.decode('hex'),target_plaintext)

print "Decrypted msg:"
for cipher in ciphers:
 print strxor(cipher.decode('hex'),key)

print "\nPrivate key recovered: "+key+"\n"

for i in ciphers:
target_fix(i)
```

```
gree with me to use this
ncryption scheme always.

Private key recovered: afctf{OPT_1s_Int3rest1ng}

Fix this sentence:
```

https://blog.csdn.net/weixin_45859850

2018 QCTF Xman-RSA

火眼金睛

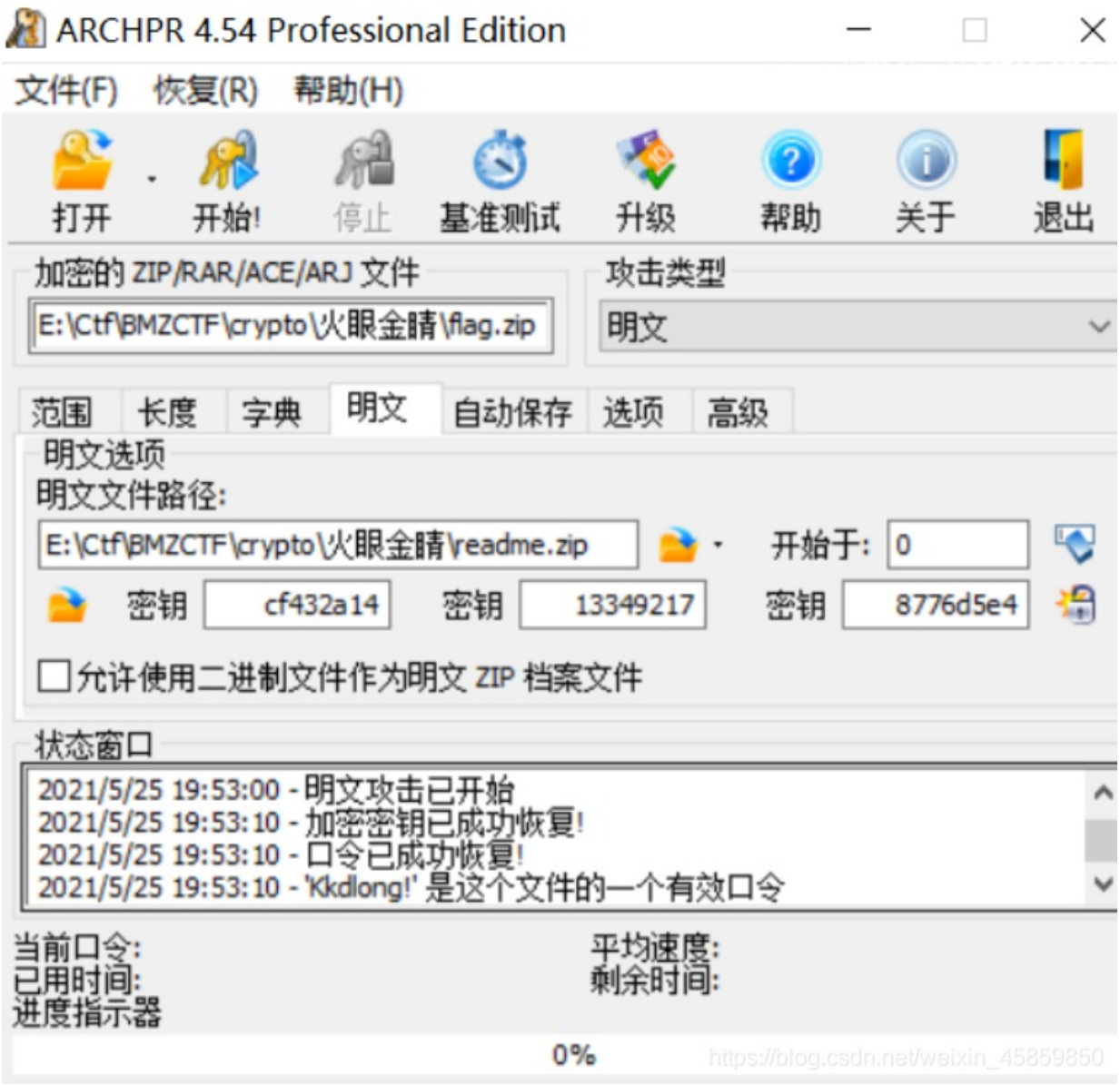
五位数字爆破得到flag.zip 和 readme.txt

利用明文攻击破解flag.zip

破解zip加密文件一般有五种：zip属性隐藏、zip伪加密、暴力破解、明文攻击、CRC32碰撞

```
https://www.cnblogs.com/islsy/p/10611488.html
```

将readme.txt压缩为压缩包再进行明文攻击



得到：

[外链图片转存失败,源站可能有防盗链机制,建议将图片保存下来直接上传(img-qWBHtvvb-1628914224362)(C:\Users\Rang\AppData\Roaming\Typora\typora-user-images\image-20210525195618000.png)]

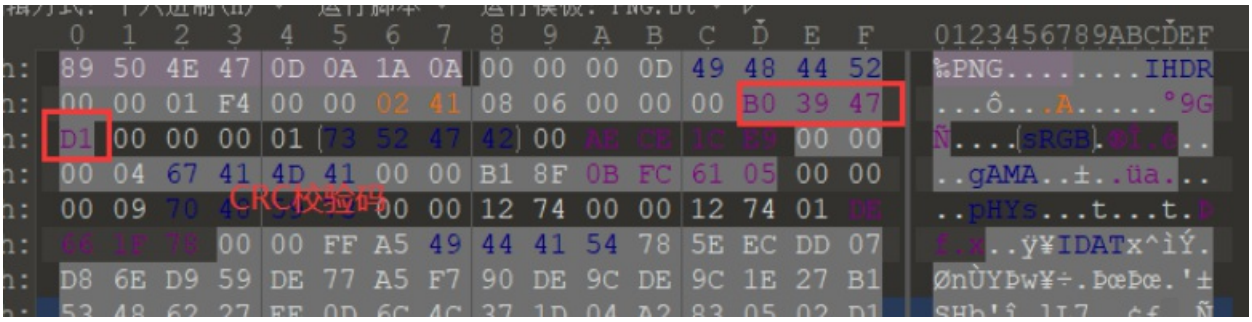
发现图片尾部有base64密文，解密得到

```
>>> base64.b64decode('ZmxhZ3tUaDFzXzFzX2Zha2VmMWFnfQ==')
b'flag{Th1s_1s_fakeflag}'
>>>
```

提交发现错误

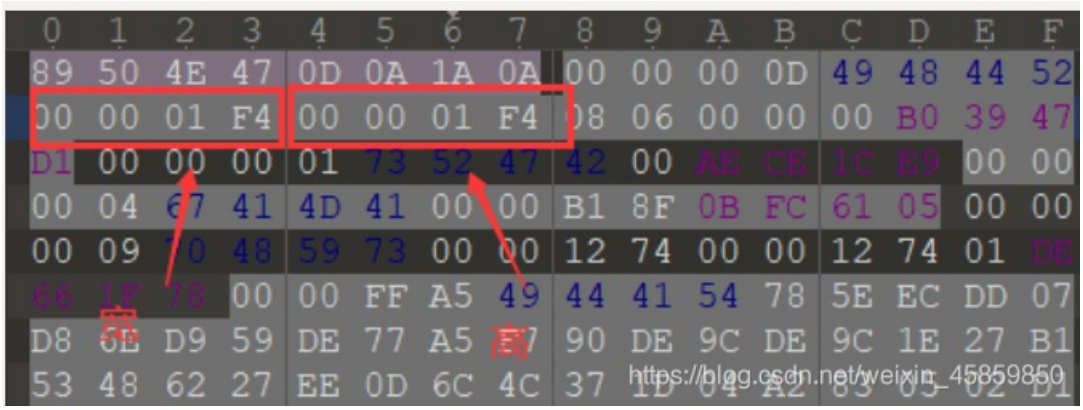
用脚本还原：

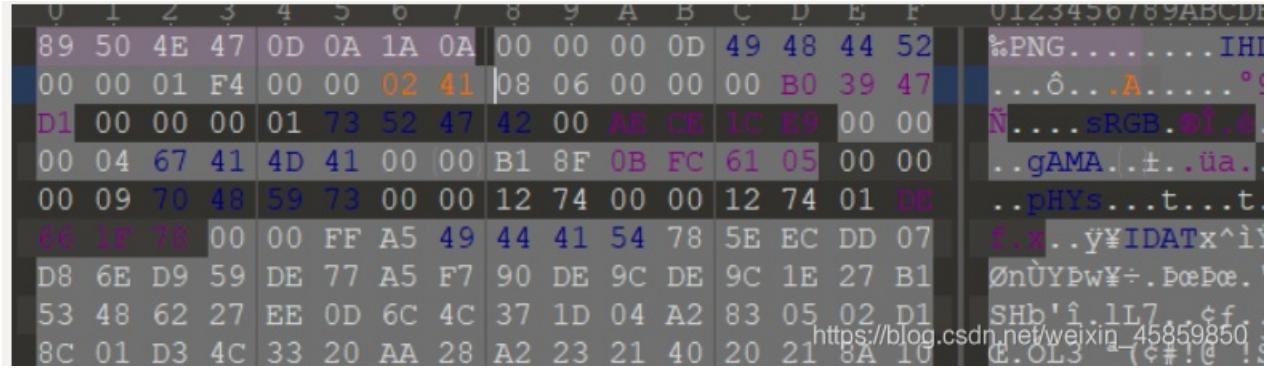
```python
# -*- coding: utf-8 -*-
import binascii
import struct

crc32key = 0xB03947D1
for i in range(0, 65535):
  height = struct.pack('>i', i)
  data = '\x49\x48\x44\x52\x00\x00\x01\xF4' + height + '\x08\x06\x00\x00\x00'  #校验位
  crc32result = binascii.crc32(data) & 0xffffffff

  if crc32result == crc32key:
    print(''.join(map(lambda c: "%02X" % ord(c), height)))
```



考虑修改图片宽高，png图片参数参考：

https://blog.csdn.net/weixin_42212792/article/details/112815529?utm_medium=distribute.pc_relevant.none-task-blog-baidujs_title-0&spm=1001.2101.3001.4242

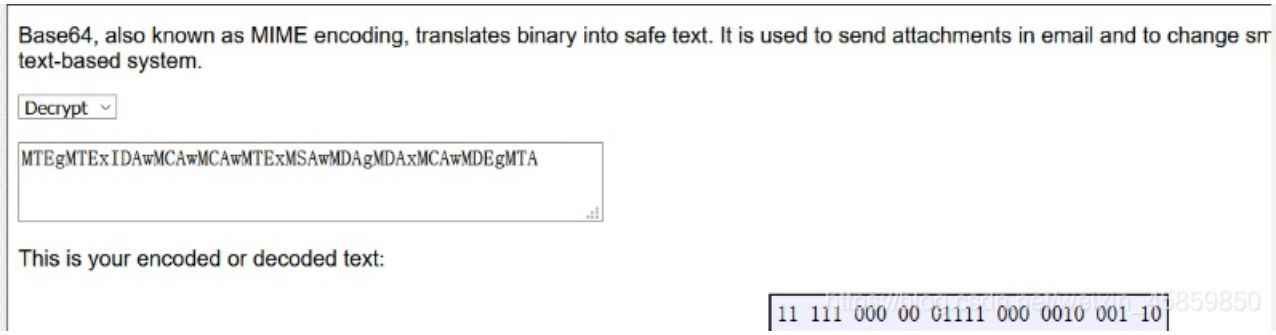




1837

`MTEgMTExIDAwMCAwMCAwMTExMSAwMDAgMDAxMCAwMDEgMTA`

摩斯密码发明于1837年，密文经过base64解码后得到01字符串，morse解密



转成.和-,morse解密得到flag

Decrypt

Your message: (Reverse - Swap)

```
-- --- ... .. .---- ... ..-. ..- -.
```

This is your encoded or decoded text:

https://blog.csdn.net/we

flag{MOSI1SFUN}

Caesar

变异的凯撒密码，前三位到flag分别移位345。。。

注意'\'需要再加个\转义一下

```
c = 'ch\\at]ZW+S$)6Q#k'
s = ''
for i in range(len(c)):
    s += chr(ord(c[i])+ 3 + i)
print(s)
#flag{eca6_17Ea4}
```

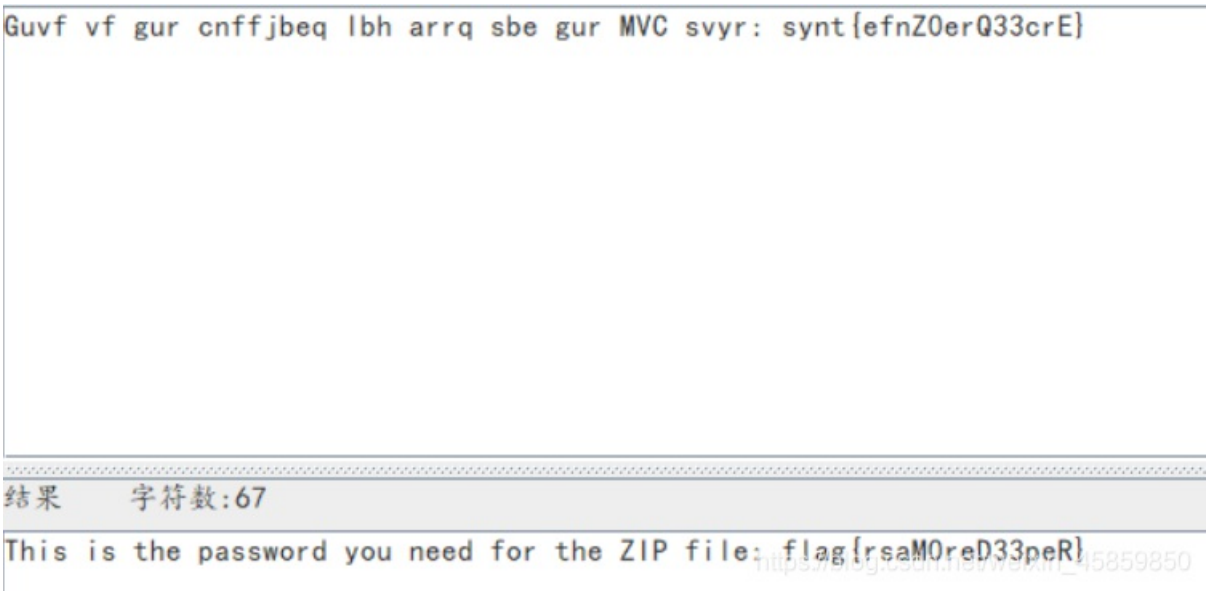
rsa

题目只给了密文c

将c开e次方，这里e=3

```
>>> c = 41533724213287870641685486418457081839214434469901585725061145597354419505017069841182359444419288897490
83563790293558028143373121367441549974248211570336802004821051820943229232421937298269855190652251294220483768084460779714162849925877879859830009443131489814222929347727356161133596952286154320203632402474746221329869391084573936183461000331479459596844437629766814547554821924339932862055270030292695747870264843896228169328351847545403125618907194079862964811868472929672702887526 16
>>> libnum.n2s(int(gmpy2.iroot(c,3)[0]))
b'Guvf vf gur cnffjbeq lbh arrq sbe gur MVC svyr: synt{efnZ0erQ33crE}\n'
```

rot13解密一下



2020sdnisc-ezRSA

共模攻击

```
#2020sdnisc-ezRSA
n = 0xa1d4d377001f1b8d5b2740514ce699b49dc8a02f12df9a960e80e2a6ee13b7a97d9f508721e3dd7a6842c24ab25ab87d1132358de7
c6c4cee3fb3ec9b7fd873626bd0251d16912de1f0f1a2bba52b082339113ad1a262121db31db9ee1bf9f26023182acce8f84612bfeb07580
3cf610f27b7b16147f7d29cc3fd463df7ea31ca860d59aae5506479c76206603de54044e7b778e21082c4c4da795d39dc2b9c0589e577a77
3133c89fa8e3a4bd047b8e7d6da0d9a0d8a3c1a3607ce983deb350e1c649725cccb0e9d756fc3107dd4352aa18c45a65bab7772a4c5aef70
20a1e67e6085cc125d9fc042d96489a08d885f448ece8f7f254067dfff0c4e72a63557
e1 = 0xf4c1158f
c1 = 120517963665240884892844451092955026863414984269652772300699152941591319762314737899772793642639650994222356
64772377527806056937807146913186636839939477289822416651808959334080391379832745196358999673432349794330181905171
87090807518655868569656941242449109980876397661605271517459716669684900920279597477446629607627693769738733623141
36931706967798518824049949236734839715283148061308924165098540170911373251952012256174079596457881458762028820242
72310620418325709475500292470800913856034743255209090548913213515493298752123929957850900829061439870079967092880
05692609756924823628055245227290288940649158862576448537833423
e2 = 0xf493f7d1
c2 = 16648382384980770705624348910895797622774711113202207693584907182552301186239613809347201161450012615995859
73841066145243849675635348553830561494921177666879386498442969679094475089469195779923426450853008402689461122851
36989633474023291098381096216097704069257005209833878114510748384703700446786340992020034809259032675087440061954
54552340253250608172238138589850747208721241681429439264676946767177135035590071128743817500054063714001099629435
08349497151148446064846096531445037416174913915923050332242843403926133165817310272633884358263778516770288515592
95983215176249952636313180194516350199933780820807438121279 5
def gongmo(n, c1, c2, e1, e2):
    def egcd(a, b):
        #扩展欧拉定理
        if b == 0:
            return a, 0
        else:
            x, y = egcd(b, a % b)
            return y, x - (a // b) * y
    s = egcd(e1, e2)
    s1 = s[0]
    s2 = s[1]

    # 求模反元素（逆元）
    if s1 < 0:
        s1 = - s1
        c1 = invert(c1, n)
    elif s2 < 0:
        s2 = - s2
        c2 = invert(c2, n)
    m = pow(c1, s1, n) * pow(c2, s2, n) % n
    return m
result = gongmo(n, c1, c2, e1, e2)
#print(result)
print(libnum.n2s(int(result)))
#flag{8c16c91be3f3287ff5a10167e922b33b}
```

#####baby_dsa 上海市大学生网络安全大赛

dsa加密

http://blog.iyzyi.com/index.php/archives/1973/

encode

```
#!/usr/bin/env python
from Crypto.Util.number import *
from hashlib import sha512,md5
from os import urandom
import random
```

```
def hash(message):
 return int(sha512(message).hexdigest(), 16)

def key_gen():
 q = getPrime(256)
 while True:
  p = random.getrandbits(2816)*q + 1
  if isPrime(p):
   print(p.bit_length())
   break
 while True:
  g = pow(random.randrange(1, p-1), (p-1)/q, p)
  if g != 1:
   break
 x = random.randrange(1, q)
 y = pow(g, x, p)
 pubkey = (p, q, g, y)
 privkey = x
 return pubkey, privkey

def sign(message, pubkey, privkey):
 p, q, g, y = pubkey
 x = privkey
 k = pow(y, x, g) * random.randrange(1, 512) % q
 r = pow(g, k, p) % q
 s = inverse(k, q) * (hash(message) + x * r) % q
 return r, s

def verify(message, signature, pubkey):
 p, q, g, y = pubkey
 r, s = signature
 if not (0 < r < q) or not (0 < s < q):
  return False
 w = inverse(s, q)
 u1 = (hash(message) * w) % q
 u2 = (r * w) % q
 v = ((pow(g, u1, p) * pow(y, u2, p)) % p) % q
 return v == r

pubkey, privkey = key_gen()
print(pubkey)

message1 = urandom(16).encode('hex')
signature1 = sign(message1, pubkey, privkey)
print(message1,signature1)
print(verify(message1, signature1, pubkey))

message2 = urandom(16).encode('hex')
signature2 = sign(message2, pubkey, privkey)
print(message2,signature2)
print(verify(message2, signature2, pubkey))

flag = 'flag{'+md5(long_to_bytes(privkey)).hexdigest()+'}'
```

decode.py

```
from Crypto.Util.number import *
from hashlib import sha512,md5
```

```python
def hash(message):
    return int(sha512(message).hexdigest(), 16)


def sign(message, pubkey, privkey, random1):
    p, q, g, y = pubkey
    x = privkey
    k = pow(y, x, g) * random1 % q
    r = pow(g, k, p) % q
    s = inverse(k, q) * (hash(message) + x * r) % q
    return r, s

public = (329722646303732445800883728449896337264903888939068505184968017501650564600176122010985892162426604403
513313413540256123563583342820688688830802777235303076740092107834686837729840121381205325031600203394169227219 2
644613252296579884516731560436501073253924457646558698855484781747029397755111633297587215976579633451933658235 3
853865395180065700696535751460600168119111406146064719303273413685829798360425854068113522363260652926364845508 0
721375648215308442771454969426468569597753153742568221215555356884866608857693288823465935521366490975378175391 7
401161977762663658097504411914908081677033980915039079517766159760522261279115347385813009437510156898969769563 6
878694957219772654447995856340198809515320802179604569017889184392657881699100628228958019936641745518659548997 3
000351770200485095008494228829300145039695936946379585625051402553034971207474762463147744467360158847593356030 7
451941432762549494636506982105155695 33, 823028354421121371258914033681512499102687068248547861266003904136223021 96443, 115623326429934097110649837149549569522588059235437403414219551847254052191169950639131132467659068536523 4888717034572213506000988500233474883647716980616616980618023179491896121452069836187483911045461026638834197798 4902756569838594616255112661600466818870137432772800368859461445854700956291885576855069405183771903076277144927 7690294337307106130587882776912116986752878291432721528351718594807810619185568400798577612030120545521425556730 7186531035533198628860642271152579087759137677083418061849279426536217860311123661549522561210125034442193258803 8244804199229449738675082560512062564365473035097263889257937140778993389305893378514344032352806521972367991027 4597211607448356887616577973988415231040744517935579245129923056406973440115205507238938281857076351414044452134 4593558696528945028202422206448896587876999156636711515361976158384356157953170505795593328800855616595206617330 4891391375100346312776539530448611005, 290999623787731812697719691852061290246619413463636312382146969900546384 5 1471078284315396270485191609160167902883086617633233151951515630140153717306990818150902846432264735225663242468 4809349121024262597006913707483811117644197481959053785475083406472583099140506505071300193356002443007750220524 9322191919329692022703433239550352913968084726866847876105591147020547846993654908603927370610562331603089432964 7854079835313487893708833667292816289433296176227755934586047991624808682111781199039102518712519307405900108644 1305977133252774698996653122297123447837449168657347308016270030881395674066421144002959751936839166935726200833 7851327362288597103518713525675115161421709560918853521785793022996343222548183839785857731366925889229760436173 3790454539614675560928416374347629777268654847517019760541284768958717152245322905593271271415486998945480856145 8852031769119489235598402066924082778376081494632258448434048562053)
p, q, g, y = public

tmp1 = (b'0234e7971889def7e60348f77db94b7a', (108592362699597657352363937799363052173055743318392345021902267089
29991582386, 137075573238956952604710531378285238377458956832183313433600273803109801088 19))
tmp2 = (b'16c5ac270b72f70319657b4410d985d4', (419606422463790676405247094160015360582928173191097643173697772244
26218746518, 746767253225155935023462754688434115637469821493736700210826863413690767190 88))
message1, r1, s1 = tmp1[0], tmp1[1][0], tmp1[1][1]
message2, r2, s2 = tmp2[0], tmp2[1][0], tmp2[1][1]
hm1, hm2 = hash(message1), hash(message2)
#print(hm1, hm2)


for random1 in range(1, 512):
    for random2 in range(1, 512):
        random1_inv = inverse(random1, q)
        random_mul = random1_inv * random2
        x = (s1 * hm2 - s2 * hm1 * random_mul) * inverse(s2 * r1 * random_mul - s1 * r2, q)
        x = x % q
        #print(x)
        #print(sign(message1, public, x, random1))
        #print(tmp1[1])
```

```
        #print(tmp1[1])
        if sign(message1, public, x, random1) == tmp1[1]:
            print(x)
            flag = 'flag{'+md5(long_to_bytes(x)).hexdigest()+'}'
            print(flag)
            print(random2)
            exit()
    print(random1)


# 成功拿到flag后可以知道random1 = 36, random2 = 58 ^_^
#flag{448018d725636e5ca1ed9675a8b23df4}
```

技协杯-Crypto1

已知：

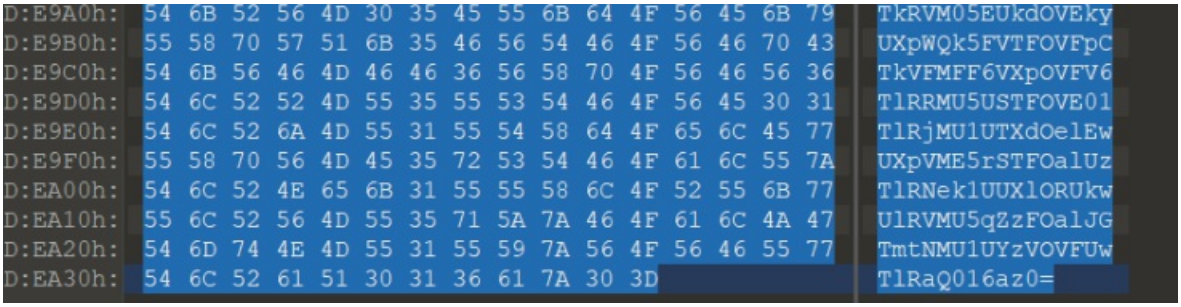
N=
27043815113145707550121063378240438939248782258483454049856453521584032220592887787562059443380038610290598575622062916485307184091409554538514854202498873547297773103239316954239641667336894313367845772845023065721439134501310426590019355844524756920078307005324929526063425487642209456355023230208467860774742785099317901398963867106582476740326082746953262216504020111095817613656347722083195076143847189035035063021757574382557663479442190050695927108198420191943271875798380684096592680930644200136436407472778630811000456073548836196890375350827356906695355960262739240052449868264201871744730229732543231817722155104619728560689810728302467665028384809892663986446651601308413555363224109072764136958626173559366461328524629930387585431098744710318579504049054110615479024389369915476786946136053237025832393660694888738856106252492880973063366479687255584014466230247361472260166921534634915730707929809348742977618533073406759188033895968722998372458726742790244324576861937027399498160467838205116976541368504923992681650515673706697972844901390862442952533559867450483754852300653372344992667809502664492109889 e = 5

发现加密指数e比较少，使用低加密指数攻击

```
#python3
## -*- coding: utf-8 -*-#
import gmpy2
import libnum
n=27043815113145707550121063378240438939248782258483454049856453521584032220592887787562059443380038610290598575622062916485307184091409554538514854202498873547297773103239316954239641667336894313367845772845023065721439134501310426590019355844524756920078307005324929526063425487642209456355023230208467860774742785099317901398963867106582476740326082746953262216504020111095817613656347722083195076143847189035035063021757574382557663479442190050695927108198420191943271875798380684096592680930644200136436407472778630811000456073548836196890375350827356906953559602627392400524498682642018717447302297325432318177221
c=55104619728560689810728302467665028384809892663986446651601308413555363224109072764136958626173559366461328524629930387585431098744710318579504049054110615479024389369915476786946136053237025832393660694888738856106252492880973063366479687255584014466230247361472260166921534634915730707929809348742977618533073406759188033895968722998372458726742790244324576861937027399498160467838205116976541368504923992681650515673706697972844901390862442952533559867450483754852300653372344992667809502664492109889
e = 5
k = 0
while 1:
    result = gmpy2.iroot(c + k*n,e)   #(mpz(.....),True)
    if(result[1] == True):
        print(libnum.n2s(int(result[0])))
      break
    k = k + 1
#b'flag{c783eb94-2595-11eb-a596-00e04c000975}'
```

2018 AFCTF BASE

题目给出一大串十六进制数据，用010Editor粘贴十六进制转字符串，发现末尾有等于号，猜测是base64

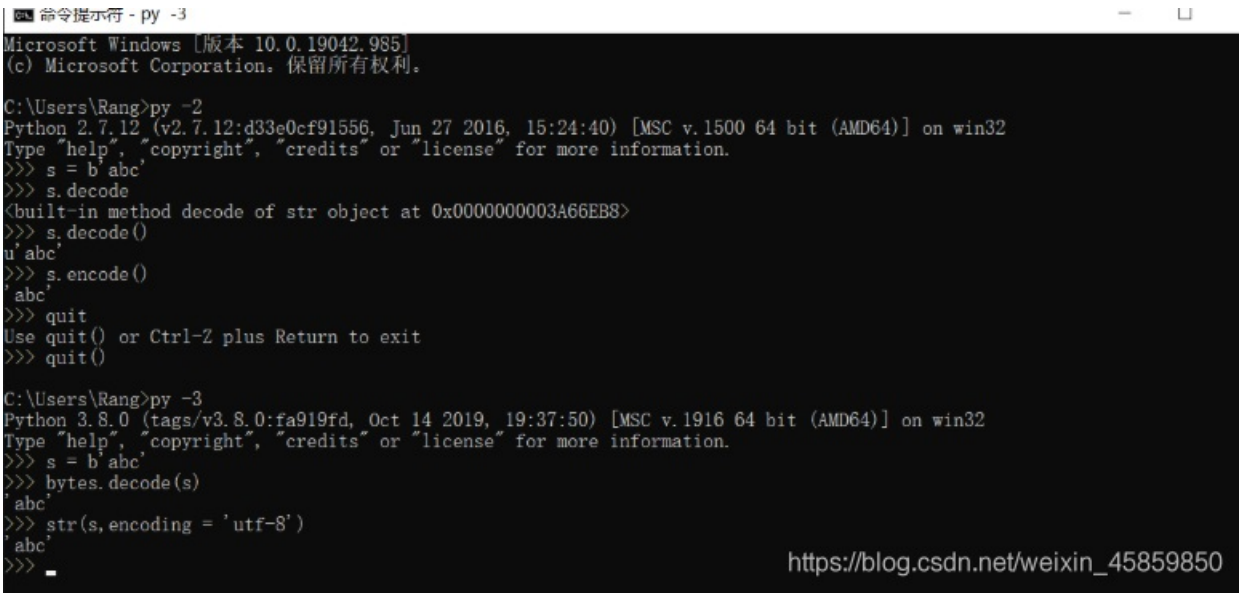


轮流进行base64、32、16直至出现{}即可

```python
# python2
import sys
import base64
path = sys.path[0]
with open (path + '\\flag_encode.txt','r') as f:

    for a in f:
        while 1:
            try :
                a=base64.b64decode(a).decode("utf-8")
            except:
                pass
            try:
                a=base64.b32decode(a).decode("utf-8")
            except:
                pass
            try:
                a=base64.b16decode(a).decode('utf-8')
            except:
                pass

            if "{" in a:
                print (a)
                break
```

```python
# python3
import sys
import base64
import libnum
path = sys.path[0]
with open (path + '\\flag_encode.txt','rb') as f:
    for a in f:
        a = libnum.n2s(int(a,16))
        a = bytes.decode(a)
        while 1:
            try :
                a=str(base64.b64decode(a),encoding="utf-8")

            except:
                pass
            try:
                a=str(base64.b32decode(a),encoding="utf-8")
            except:
                pass
            try:
                a=str(base64.b16decode(a),encoding="utf-8")
            except:
                pass
            if "{" in a:
                print (a)
                break
#afctf{U_5h0u1d_Us3_T00l5}
```

python2、python3byte和str间的转换



#######2018 AFCTF MyOwnCBC

AES加密过程详解：`https://www.cnblogs.com/luop/p/4334160.html`

encode:

```
#!/usr/bin/python2.7
# -*- coding: utf-8 -*-

from Crypto.Cipher import AES
from Crypto.Random import random
from Crypto.Util.number import long_to_bytes

def MyOwnCBC(key, plain):
 if len(key)!=32:
  return "error!"
 cipher_txt = b""
 cipher_arr = []
 cipher = AES.new(key, AES.MODE_ECB, "")
 plain = [plain[i:i+32] for i in range(0, len(plain), 32)]
 print plain
 cipher_arr.append(cipher.encrypt(plain[0]))    #初始密钥就是plain[0]
 cipher_txt += cipher_arr[0]
 for i in range(1, len(plain)):
  cipher = AES.new(cipher_arr[i-1], AES.MODE_ECB, "")
  cipher_arr.append(cipher.encrypt(plain[i]))
  cipher_txt += cipher_arr[i]
 return cipher_txt

key = random.getrandbits(256)
key = long_to_bytes(key)

s = ""
with open("flag.txt","r") as f:
 s = f.read()
 f.close()

with open("flag_cipher","wb") as f:
 f.write(MyOwnCBC(key, s))
 f.close()
```

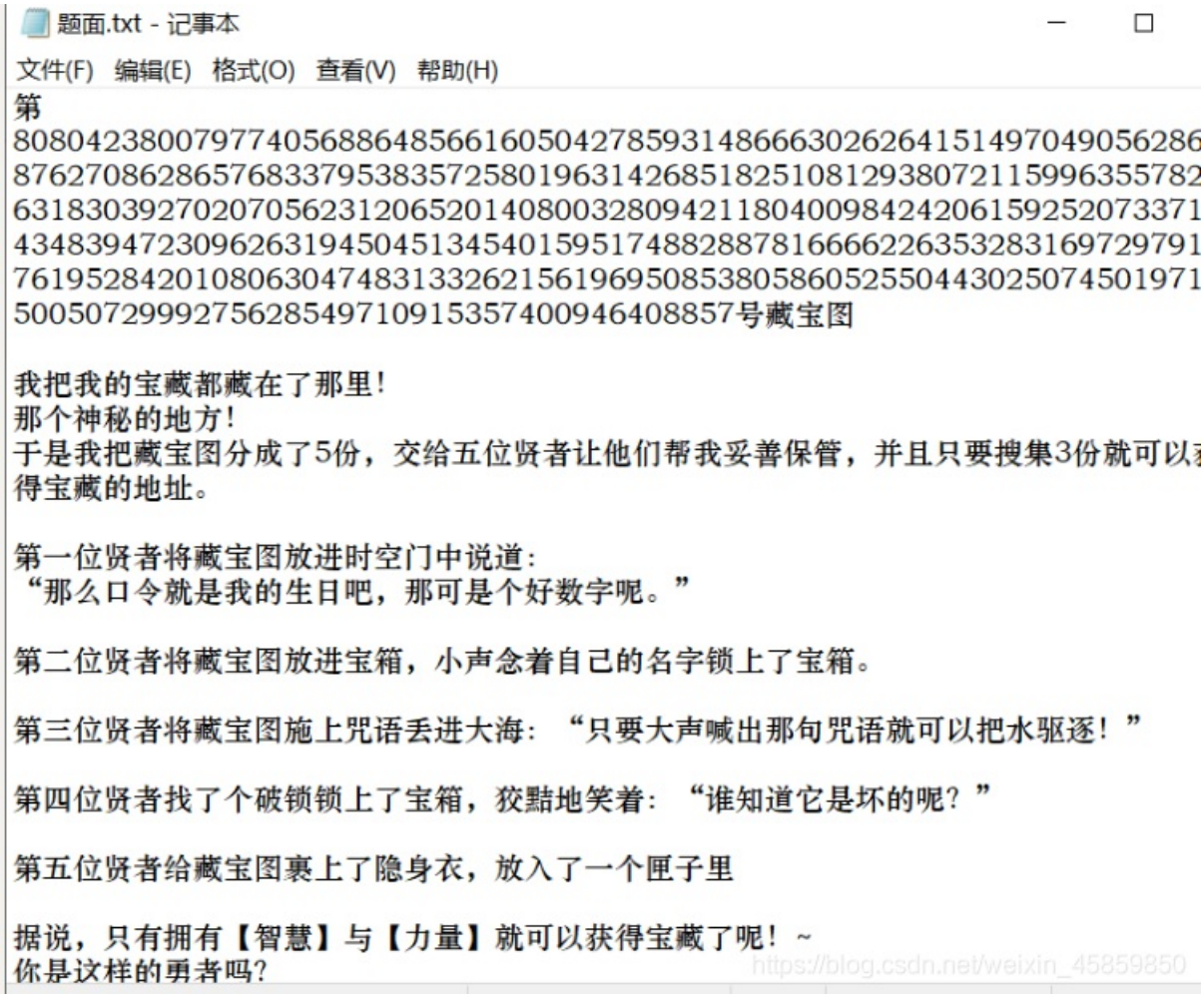
decode:

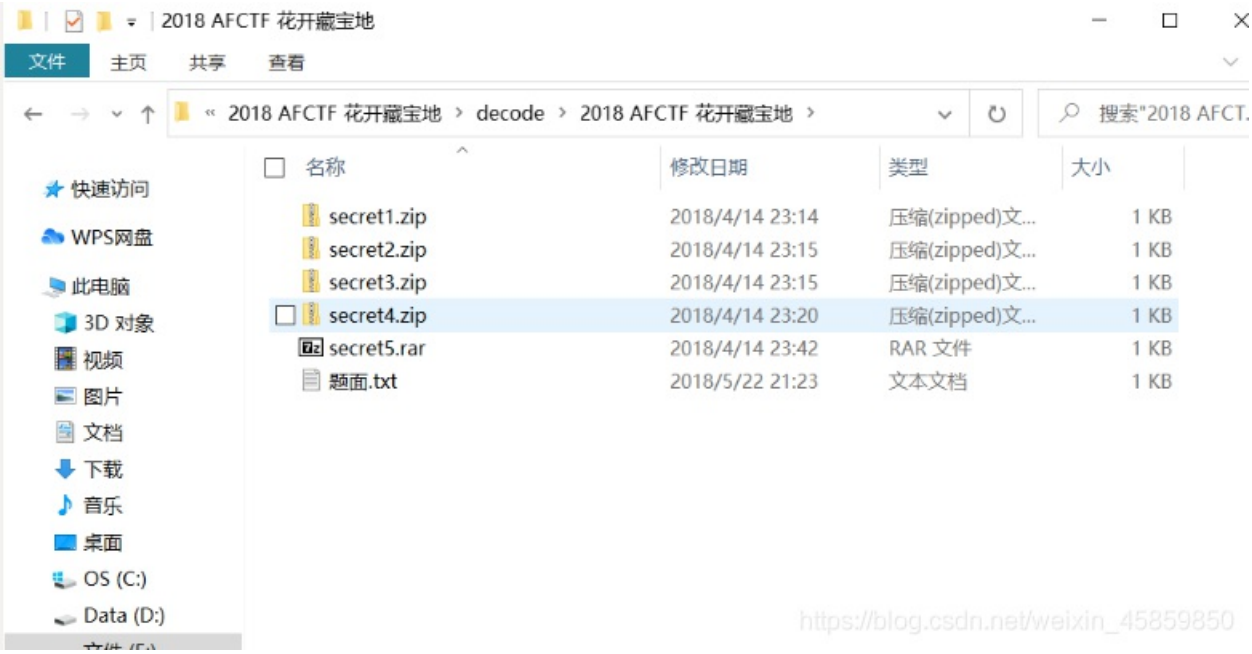
```
import Crypto.Cipher.AES
with open('flag_cipher','rb') as f:
    cipher=f.read()
    f.close()
key0=cipher[0:32]
def decrypt(key0,cipher):
    cipher=[cipher[i:i+32] for i in range(0,len(cipher),32)]
    tempkey=key0
    plain = b''
    for i in range(1,len(cipher)):
        aes=Crypto.Cipher.AES.new(tempkey,Crypto.Cipher.AES.MODE_ECB)
        plain+=aes.decrypt(cipher[i])
        tempkey=cipher[i]
    return plain
print(decrypt(key0,cipher))
```

2018 AFCTF 花开藏宝地 门限方案

参考https://shawroot.hatenablog.com/entry/2019/12/26/AFCTF2018/BUUCTF-Crypto%3A%E8%8A%B1%E5%BC%80%E8%97%8F%E5%AE%9D%E5%9C%B0：

已知



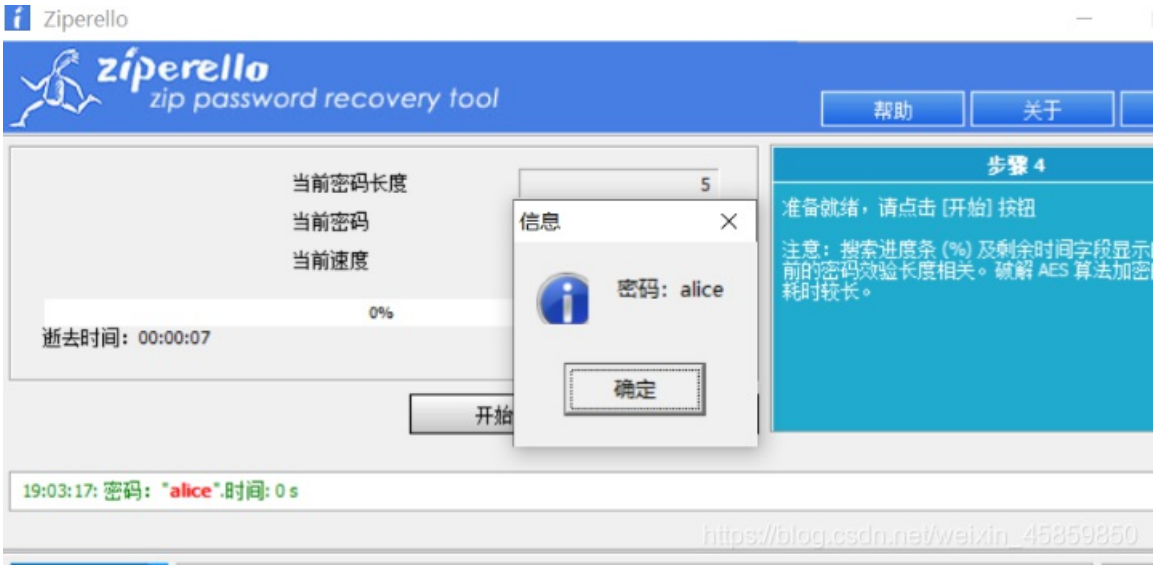


爆破第一个压缩包 8位纯数字得到

得到

```
x1 =
30534513391139521857379090350829623865914780227403179664301753901164880280876316290233564419564852537551894184843011449715008202513300003383508307654192753082955705152416106942349445166784823645233727186208534686936497698904718053216756079647006754991539077327120790153784721388247999732557527867291764841786875907715099904489109920613329633619047641316424099517707767148035273957253963135 9
m1 =
34705155962246314453966995009665816342564641143579769197370151372570157510081044617584942400000007585507043024050773273539341149386654057267962617274230136614650186267027244307097051194348586588749422948742050375045797426280205372209390512623534038026182859350845562166730994636170553066795748473192915187552748947844936119864831068470257462719932109292711113739833302969706847476282081341 3
```

第二个压缩包，采用小写爆破得到密码alice



`x2 =`

```
152012681270682340051690627924586232702552460810030322267827401771304907469802591861912921281833890613186317787813611372838066924894691892444503039545946728621696590087591246339208248647926966446848123290344911662916758039134817404720512465817867255277476717353439505243247568126193361558042940352204093381260402400739429050280526212446967632582771424597203000629197487733610187359662268583
```

`m2 =`

```
347051559622463144539669950096658163425646411435797691973701513725701575100810446175849424000000075855070430240507732735393411493866540572679626172742301366146501862670272443070970511943485865887494229487420503750457974262802053722093905126235340380261828593508455621667309946361705530667957484731929151875527489478449361198648310684702574627199321092927111137398333029697068474762820818553
```

第四个压缩包伪加密，直接解压得到

`x4 =`

```
100459779913520540098065407420629954816677926423356769524759072632219106155849450125185205557491138357760494272691949199099803239098119602186117878931534968435982565071570831032814288620974807498206233914826253433847572703407678712965098320122549759579566316372220959610814573945698083909575005303253205653244238542300266460559790606278310650849881421791081944960157781855164700773081375247
```

`m4 =`

```
347051559622463144539669950096658163425646411435797691973701513725701575100810446175849424000000075855070430240507732735393411493866540572679626172742301366146501862670272443070970511943485865887494229487420503750457974262802053722093905126235340380261828593508455621667309946361705530667957484731929151875527489478449361198648310684702574627199321092927111137398333029697068474762820820091
```

decode.py:

```
import libnum
a1 =1004597799135205400980654074206299548166779264233567695247590726322191061558494501251852055574911383577604942726919491990998032390981196021861178789315349684359825650715708310328142886209748074982062339148262534338475727034076787129650983201225497595795663163722209596108145739456980839095750053032532056532442385423002664605597906062783106508498814217910819449601577818551647007730813752478
d1 =34705155962246314453966995009665816342564641143579769197370151372570157510081044617584942400000007585507043024050773273539341149386654057267962617274230136614650186267027244307097051194348586588749422948742050375045797426280205372209390512623534038026182859350845562166730994636170553066795748473192915187552748947844936119864831068470257462719932109292711113739833302969706847476282082009178
a2 =30534513391139521857379090350829623865914780227403179664301753901164880280876316290233564419564852537551894184843011449715008202513300003383508307654192753082955705152416106942349445166784823645233727186208534686936497698904718053216756079647006754991539077327120790153784721388247999732557527867291764841786875907715099904489109920613329633619047641316424099517707767148035273957253963135978
d2 =34705155962246314453966995009665816342564641143579769197370151372570157510081044617584942400000007585507043024050773273539341149386654057267962617274230136614650186267027244307097051194348586588749422948742050375045797426280205372209390512623534038026182859350845562166730994636170553066795748473192915187552748947844936119864831068470257462719932109292711113739833302969706847476282081341378
a3 = 15201268127068234005169062792458623270255246081003032226782740177130490746980259186191292128183389061318631778781361137283806692489469189244450303954594672862169659008759124633920824864792696644684812329034491166291675803913481740472051246581786725527747671735343950524324756812619336155804294035220409338126040240073942905028052621244696763258277142459720300062919748773361018735966226858378
d3 =34705155962246314453966995009665816342564641143579769197370151372570157510081044617584942400000007585507043024050773273539341149386654057267962617274230136614650186267027244307097051194348586588749422948742050375045797426280205372209390512623534038026182859350845562166730994636170553066795748473192915187552748947844936119864831068470257462719932109292711113739833302969706847476282081855378
dd = d1*d2*d3
t1 = pow(dd//d1,d1-2,d1)
assert(t1*d2*d3%d1 == 1)
t2 = pow(dd//d2,d2-2,d2)
assert(t2*d1*d3%d2 == 1)
t3 = pow(dd//d3,d3-2,d3)
assert(t3*d2*d1%d3 == 1)
s = a1*t1*d2*d3+a2*t2*d1*d3+a3*t3*d1*d2
p = 8080423800797740568864856616050427859314866630262641514970490562862287627086286576833795383572580196314268518251081293807211599635578239631830392702070562312065201408003280942118040098424206159252073371024348394723096263194504513454015951748828878166662263532831697297918376195284201080630474831332621561969508538058605255044302507450197192500507299927562854971091535740094640885
s %= dd
# print(hex(s))
s %= p
print(libnum.n2s(int(s)))
```

2018 AFCTF 一道有趣的题目

参考：https://blog.csdn.net/weixin_44110537/article/details/107986673

encode.py

```
#加密代码
def encrypt(plainText):
    space = 10
    cipherText = ""
    for i in range(len(plainText)):
        if i + space < len(plainText) - 1:
            cipherText += chr(ord(plainText[i]) ^ ord(plainText[i + space]))
        else:
            cipherText += chr(ord(plainText[i]) ^ ord(plainText[space]))
        if ord(plainText[i]) % 2 == 0:
            space += 1
        else:
            space -= 1
    return cipherText

# 密码
# 15120d1a0a0810010a031d3e31000d1d170d173b0d173b0c07060206
```

decode.py

```
def crackit():
    '''
    暴力求出每个明文字符的最后一位
    :return: 每个明文字符最后一位组成的字符串
    '''
    tag=b'\x15\x12\r\x1a\n\x08\x10\x01\n\x03\x1d>1\x00\r\x1d\x17\r\x17;\r\x17;\x0c\x07\x06\x02\x06'
    for x in range(268435456):
        temp=bin(x)[2:].zfill(28)
        space = 10
        flag=0
        for i in range(len(temp)):
            if i + space < len(temp) - 1:
                tempx= int(temp[i]) ^ int(temp[i + space])
            else:
                tempx= int(temp[i]) ^ int(temp[space])
            if tempx!=(tag[i]%2):
                flag=1
                break
            elif int(temp[i]) % 2 == 0:
                space += 1
            else:
                space -= 1
        if flag!=1:
            print(temp)

#temp ='1010011010010101111111101001'

def crackit_2():
    '''
    解出每个方程中的明文字符的序号.
    :return: 每个方程中的明文字符的序号组成的列表.
    '''
    tag=b'\x15\x12\r\x1a\n\x08\x10\x01\n\x03\x1d>1\x00\r\x1d\x17\r\x17;\r\x17;\x0c\x07\x06\x02\x06'
    temp ='1010011010010101111111101001'
    space = 10
    g=[]
    for i in range(len(temp)):
        if i + space < len(temp) - 1:
            tempx = int(temp[i]) ^ int(temp[i + space])
```

```
            tempx = int(temp[i]) ^ int(temp[i + space])
            g.append((i,i+space))
        else:
            tempx = int(temp[i]) ^ int(temp[space])
            g.append((i,space))
        if int(temp[i]) % 2 == 0:
            space += 1
        else:
            space -= 1
    return g
def solve(g,m):
    '''
    解明文方程组
    :param g: 明文方程组
    :param m :密文
    :return: 明文
    '''
    plain=[ord('a'),ord('f'),ord('c'),ord('t'),ord('f'),ord('{')]+[-1]*22#明文未知数
    while -1 in plain:
        for i in range(28):
            if plain[i] != -1:
                for j in range(len(g)):
                    if g[j][0] == i or g[j][1] == i:
                        if g[j][0] == i:
                            plain[g[j][1]] = m[j] ^ plain[i]
                        if g[j][1] == i:
                            plain[g[j][0]] = m[j] ^ plain[i]
    return plain

#[97, 102, 99, 116, 102, 123, 99, 114, 121, 112, 116, 97, 110, 97, 108, 121, 115, 105, 115, 95, 105, 115, 95, 104, 97, 114, 100, 125]

tag=b'\x15\x12\r\x1a\n\x08\x10\x01\n\x03\x1d>1\x00\r\x1d\x17\r\x17;\r\x17;\x0c\x07\x06\x02\x06'#密文
g=crackit_2()
print(g)
plain=solve(g,tag)
print(plain)
flag=''
for i in plain:
    flag+=chr(i)
print(flag)
```

[2021-红明谷]RSA attack 低加密指数

```python
from gmpy2 import *
from Crypto.Util.number import *
import sympy
import random
from secret import flag

p1 = getPrime(1024)
print(p1)
#p1=17207120109394529415429224063180973354515455963338675823406382405343883595851554335491124997117417264960625793685762754731176017451131698440976773898124787700580215579662358746177410495179712299526621733415873684830765554397032295033998848980167216005880542215381695002259064465024759550128019220550664993603 1

p2 = p1 - random(999,99999)
print(p2)
#p2=17207120109394529415429224063180973354515455963338675823406382405343883595851554335491124997117417264960625793685762754731176017451131698440976773898124787700580215579662358746177410495179712299526621733415873684830765554397032295033998848980167216005880542215381695002259064465024759550128019220550664990203 4

p_1=1
for i in range(1,p1+1):
    p_1*=i
p3 = sympy.nextPrime(p_1 % p2 )

p4 = p3 >> 50 << 50
#已知高位
p = p4
while(isPrime(P)!=1):
    P = p + random.randint(0,2**50)

Q = getPrime(1024)

e = 1+1+1
N = P * Q
print(N)
#N=28592245028568852124815768977111125874262599260058745599820769758676575163359612268623240652811172009403854869932602124987089815595007954065785558682294503755479266935877152343298248656222514238984548734114192436817346633473367019138600818158715715935132231386478333980631609437639665255977026081124468935510279104246449817606049991764744352123119281766258347177186790624246492739368005511017524914036614317783472537220720739454744527197507751921840839876863945184171493740832516867733853656800209669179467244407710022070593053034488226101034106881990117738617496520445046561073310892360430531295027470929927226907793 2

flag=bytes_to_long(flag)
c = pow(flag,e,N)
print(c)
#c=15839981826831548396886036749682663273035548220969819480071392201237477433920362840542848967952612687163860026284987497137578272157113399130705412843449686711908583139117413 3
```

decode.py

```
#python3
## -*- coding: utf-8 -*-#
import gmpy2
import libnum
e = 3
n = 28592245028568852124815768977111125874262599260058745599820769758676575163359612268623240652811172009403854869932602124987089815595007954065785558682294503755479266935877152343298248656222514238984548734114192436817346633473367019138600818158715715935132231386478333980631609437639665255977026081124468935510279104246449817606049991764744352123119281766258347177186790624246492739368005511017524914036614317783472537220720739454744527197507751921840839876863945184171493740832516867733853656800209669179467244407710022070593053034488226101034106881990117738617496520445046561073310892360430531295027470929927226907793
c = 15839981826831548396886036749682663273035548220969819480071392201237477433920362840542848967952612687163860026284987497137578272157113399130705412843449686711908583139117413
k = 0
while 1:
    result = gmpy2.iroot(c + k*n,e)   #(mpz(.....),True)
    if(result[1] == True):
        print(libnum.n2s(int(result[0])))
        break
    k = k + 1
```

经典与现代的碰撞

task.py

```
from Crypto.Util.number import *

flag="**********************************"
p = getPrime(512)
q = getPrime(512)
e = 65537
n = p*q
c = []
for i in flag:
 c.append(pow(ord(i),e,n))
with open("data","w")as f:
 f.write(str(c))
print(n)

# n = 10220415322299411177863287675935774343316569952991142862343016060614671518655306910275730352031532564773135359136446101566016638234758757655866406088970453408688969848078581361656788470639262532693598506817323102237068251776210759334584904939852187749733971661673780003933469336904158450224924787828308851401 3
#分解n、低加密指数
```

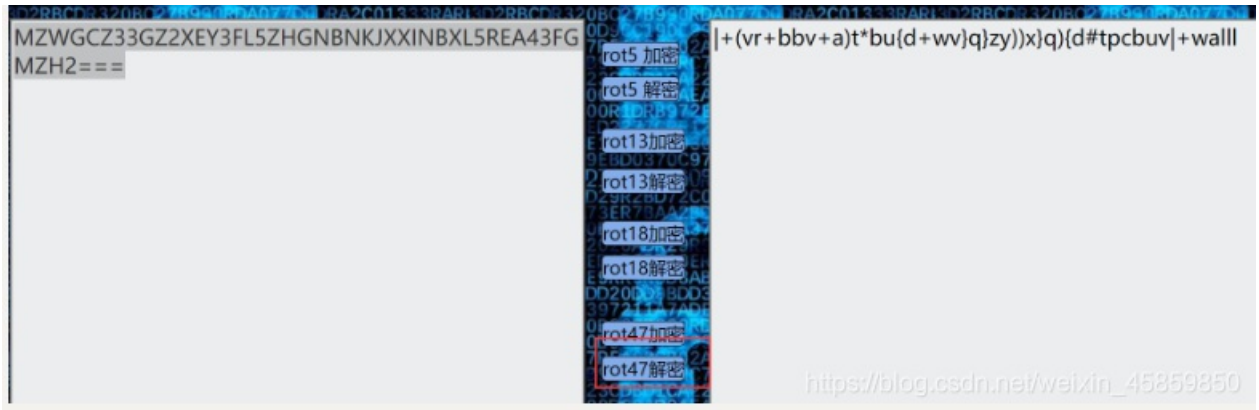
我们发现密文串被拆分，也就是每个密文m都很小，可以直接fuzz

```python
#我们知道对应得密文m很小
import os
import sys
import libnum
import gmpy2
path = sys.path[0]
with open(path + r'\data','r') as f:
    f = f.read()
    c = f.strip('[').strip(']').split(', ')
    #print(c)
    print(len(c))
e = 65537
n = 102204153222994111778632876759357743433165699529911428623430160606146715186553069102757303520315325647731353591364461015660166382347587576558664060889704534086889698480785813616567884706392625326935985068173231022370682517762107593345849049398521877497339716616737800039334693369041584502249247878283088514013
def fuzz_m(c_):
    for i in range(1,150):
        if c_ == pow(i,e,n):
            print(chr(i),end= '')
            break
x = 0
for i in c:
    fuzz_m(int(i))
#ROT:|+(vr+bbv+a)t*bu{d+wv}q}zy))x}q){d#tpcbuv|+walll
```

得到一串ROT:...一串古典密码加密得密文（。。。凯撒）

提示ROT：`http://www.ab126.com/goju/11044.html`

经过rot47解码得到一串base编码的密文



再经过base32解码`http://ctf.ssleye.com/base64.html`得到flag:flag{6urce_rs4-Rot47_b@se32}

LRX

encode.py

```
import random
from sympy import nextprime
from Crypto.Util.number import *
flag="flag{**********************}"

def lcg(seed,params):
    (m,c,n)=params
    x = seed % n
    while True:
        x = (m * x + c) % n
        yield x   #看做return，再把它看做一个是生成器（generator）的一部分（带yield的函数才是真正的迭代器）

seed = random.randint(0,0xffffffff)
m = random.randint(0,0xffffffff)
c = random.randint(0,0xffffffff)
n = random.randint(0,0xffffffff)

(m,c,n) = (378397371,3163496981,4161661778)
key_stream = lcg(seed,(m,c,n))
for _ in range(6):
 print(next(key_stream))

secret =  next(key_stream)   #secret <=> key_stream[5] 、secret = (m * s[5] + c) % n

e = nextprime(secret)
p = getPrime(1024)
q = getPrime(1024)
phi = (p-1)*(q-1)

flag = bytes_to_long(flag)
flag ^= flag >> 10
print(phi)
print(p*q)
print(pow(flag,e,p*q))
```

```
# 3836384749
# 3395903484
# 2071233297
# 784139606
# 799255125
# 2029002258
# 138473353161275365387328088979890225395562490707072733065694090514000405147526768858024871233031398404153383783
70664720292963480194299667734532323504964017655553559984444757198026939289607642274379099943475039015068954828
566502337327598451567876130270838934244307870535472010589025001553484742930483432387442315084794635809090529554859
45360876061209366163955351349288487102780740553143348832936381821651257280772957044640156231980014817952229918
931651204203358799284605762262337020714573809585791528977045651712794260807489896480884644220973424042723101009539
45589685217309069447321591032088089589262038377350212816 80
# 138473353161275365387328088979890225395562490707072733065694090514000405147526768858024871233031398404153383783
70664720292963480194299667734532323504964017655553559984444757198026939289607642274379099943475039015068954828
566502337327598451567876130270838934244307870535472010589025001553484742930483432387442551834089172903643768641265
01942700054046050465652804684411818548657566226701330198458317786992889875521396375303321864836657044354971487190
50630810527553055429554352188761180209727717548976044389855294367209866854904072882062399766236087628233945854
15785374519997635420976880700398260198479277069282174283567
# 752340318957926992995883802332174509383161095517712025293530186753025733451572690746986922653015139201977710016
33276546392476860159014399932656015147061890686088759280824402644387595088803364892586956316147074857567203533315
72282438961744475814819985647986235237127112688458798443715231546818031849775965656330485887950774061767496680
501407209426116561035464728967965450960645963675248557630604467297046809534669875544667319838467864836146400639949
66089733850750466337459945157007776415335066777491193665351450255144586138337816631313829594048515042468965881
0316274747025200570548377567156398446497776465393005751004
```

参考：`https://blog.csdn.net/weixin_45883223/article/details/112427962`

已知m,c,n

可以根据m、c、n得到secret(e的上一个素数)，也就是得到了e。

又知道e,phi 可以得到d

此时求得的flag，需要经过异或flag>>10后得到最后的flag

```
# 138473353161275365387328088979890225395562490707072733065694090514000405147526768858024871233031398404153383783
70664720292963480194299667734532323504964017655553559984444757198026939289607642274379099943475039015068954828
566502337327598451567876130270838934244307870535472010589025001553484742930483432387442315084794635809090529554859
45360876061209366163955351349288487102780740553143348832936381821651257280772957044640156231980014817952229918
931651204203358799284605762262337020714573809585791528977045651712794260807489896480884644220973424042723101009539
4558968521730906944732159103208808958926203837735021281680
# 138473353161275365387328088979890225395562490707072733065694090514000405147526768858024871233031398404153383783
70664720292963480194299667734532323504964017655553559984444757198026939289607642274379099943475039015068954828
566502337327598451567876130270838934244307870535472010589025001553484742930483432387442551834089172903643768641265
01942700054046050465652804684411818548657566226701330198458317786992889875521396375303321864836657044354971487190
50630810527553055429554352188761180209727717548976044389855294367209866854904072882062399766236087628233945854
15785374519997635420976880700398260198479277069282174283567
# 752340318957926992995883802332174509383161095517712025293530186753025733451572690746986922653015139201977710016
33276546392476860159014399932656015147061890686088759280824402644387595088803364892586956316147074857567203533315
72282438961744475814819985647986235237127112688458798443715231546818031849775965656330485887950774061767496680
501407209426116561035464728967965450960645963675248557630604467297046809534669875544667319838467864836146400639949
66089733850750466337459945157007776415335066777491193665351450255144586138337816631313829594048515042468965881
0316274747025200570548377567156398446497776465393005751004

from libnum import *
from gmpy2 import *
from sympy import nextprime
from binascii import *
```

```
from gmpy2 import *
s = [3836384749,3395903484,2071233297,784139606,799255125,2029002258]

m = 378397371
c = 3163496981
n = 4161661778
secret = (m * s[5] + c) % n         #secret = (m * s[5] + c) % n
e = nextprime(secret)
phi = 13847335316127536538732808897989022539556249070707273306569409051400040514752676885802487123303139840415338378370664720292963480194299667734532323504964017655553559984444757198026939289607642274379099943475039015068954828566502337327598451567876130270838934244307870535472010589025001553484742930483432387442315084794635809090529554859453608760612093661639553513492884871027807405531433488329363818216512572807729570446401562319800148179522299189316512042033587992846057622623370207145738095857915289770456517127942608074898964808846442209734240427231010095394558968521730906944732159103208808958926203837735021281680
cipher = 75234031895792699299588380233217450938316109551771202529353018675302573345157269074698692265301513920197771001633276546392476860159014399326560151470618906860887592808244026443875950888033648925869563161470748575672035333157228243896174447581481998564798623523712711268845879844371523154681803184977596565633048588795077406176749668050140720942611656103546472896796545096064596367524855763060446729704680953466987554466731983846786483614640063994966089733850750466337459945157007776415335066777491193665351450255144586138337816631313829594048515042468965881031627474702520057054837756715639844649777646539300575100 4
n = 13847335316127536538732808897989022539556249070707273306569409051400040514752676885802487123303139840415338378370664720292963480194299667734532323504964017655553559984444757198026939289607642274379099943475039015068954828566502337327598451567876130270838934244307870535472010589025001553484742930483432387442551834089172903643768641265019427000540460504656528046844118185486575662267013301984583177869928898755213963753033218648366570443549714871905063081052755305542955435218876118020972771754897604438985529436720986685490407288206239976623608762823394585415785374519997635420976880700398260198479277069282174283567
d = invert(e, phi)
flag_ = pow(cipher, d, n)
flag = ''
for i in range(len(bin(flag_)[2:])):
    j = i
    k = int(bin(flag)[2:][i])
    while j >= 10:
        j -= 10       #flag = flag ^ flag<<10
        k ^= int(bin(flag_)[2:][j])
    flag += str(k)
print(n2s(int(flag_, 2)))
#flag{You_Know_LCG_and_RSA_and_xor}
```

01765555355998444475719802693928960764227437909994347503901506895482856650233732759845156787613027083893424430787053547201058902500155348474729304834323874423150847946358090905295548594536087606120936616395535134928848710278074055314334883293638182165125728077295704464015623198001481795222991893165120420335879928460576226233702071457380958579152897704565171279426080748989648088464422097342404272310100953945589685217309069447321591032088089589262038377350212816801
cipher =
752340318957926992995883802332174509383161095517712025293530186753025733451572690746986922653015139201977710016332765463924768601590143999326560151470618906860887592808244026443875950888033648925869563161470748575672035333157228243896174447581481998564798623523712711268845879844371523154681803184977596565633048588795077406176749668050140720942611656103546472896796545096064596367524855763060446729704680953466987554466731983846786483614640063994966089733850750466337459945157007776415335066777491193665351450255144586138337816631313829594048515042468965881031627474702520057054837756715639844649777646539300575100 4
n =
138473353161275365387328088979890225395562490707072733065694090514000405147526768858024871233031398404153383783706647202929634801942996677345323235049640176555535599844447571980269392896076422743790999434750390150689548285665023373275984515678761302708389342443078705354720105890250015534847429304834323874425518340891729036437686412650194270005404605046565280468441181854865756622670133019845831778699288987552139637530332186483665704435497148719050630810527553055429554352188761180209727717548976044389855294367209866854904072882062399766236087628233945854157853745199976354209768807003982601984792770692821742835 67
d = invert(e, phi)
flag_ = pow(cipher, d, n)
flag = ‘’
for i in range(len(bin(flag_)[2:])):
j = i
k = int(bin(flag)[2:][i])
while j >= 10:
j -= 10 #flag = flag ^ flag<<10
k ^= int(bin(flag_)[2:][j])
flag += str(k)
print(n2s(int(flag_, 2)))
#flag{You_Know_LCG_and_RSA_and_xor}