bmzctf-crypto writeup（一） (持续更新)

原创

dameow 已于 2022-02-27 09:51:36 修改 120 收藏 1
分类专栏： CTF 文章标签： 安全
于 2022-01-07 19:27:17 首次发布

4进制

4进制转16进制，再转字符串。

2018 AFCTF Morse

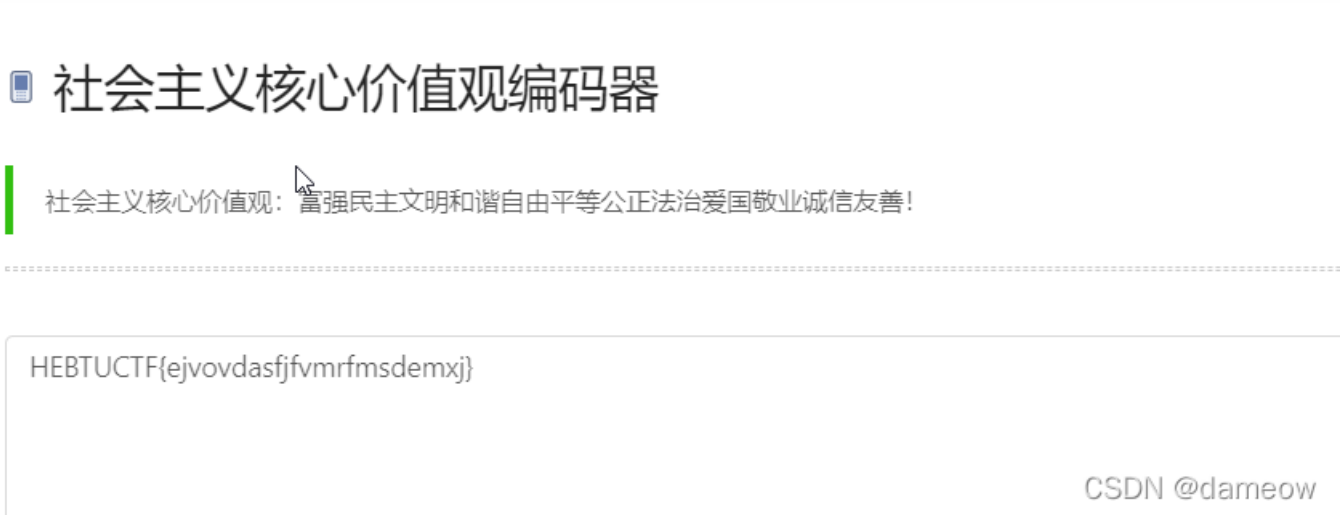
摩斯电码解密得16进制。

16进制转字符串得flag。

2018 HEBTUCTF 社会主义接班人

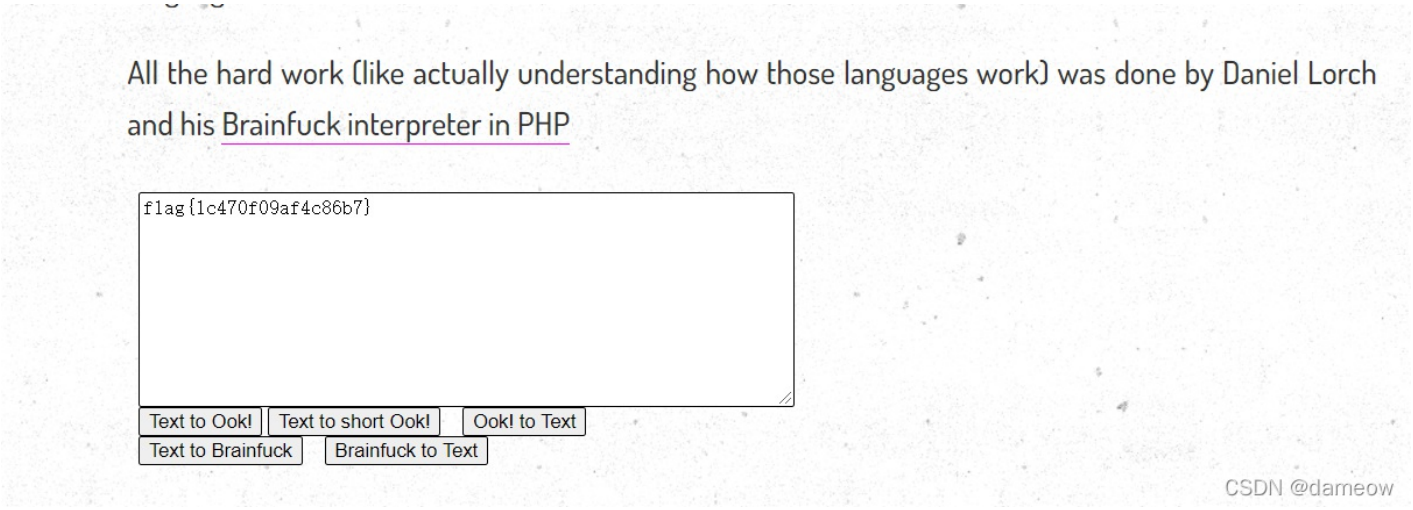
自由爱国自由平等自由文明平等自由平等平等自由和谐平等自由自由公正法治诚信民主公正平等公正友善自由法治公正公正友善敬业法治公正公正自由公正民主法治和谐公正公正公正诚信富强公正公正法治公正公正友善法治法治文明公正公正公正友善法治法治和谐公正自由公正平等公正诚信和谐法治爱国公正友善自由法治诚信和谐

这是社会主义编码，解码工具：https://atool.vip/corevalue/



Ook!

这是Ook编码。解密工具：Brainfuck/Ook! Obfuscation/Encoding [splitbrain.org]



栅栏密码

栅栏中的base

给了16进制，先16进制转字符串，得base32。

base32解码得base64。

base64解码得栅栏密码。

栅栏密码解密，组数为5，得flag。

easy_base

题目说影分身之术*40，也就是说，base加密了40次。

解密：BASE64加密解密 点上40次也不是很久。

2018 HEBTUCTF Sudoku&Viginere

做数独，做出来才能得到维吉尼亚的密码。

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

y	_	1	a	t	r	a	5	3
r	t	a	y	e	3	e	1	_
e	3	5	5	_	1	r	t	y
a	e	r	_	1	y	5	3	t
5	y	t	r	3	e	_	a	1
3	1	_	t	a	5	y	r	e
_	a	y	1	5	t	3	e	r
1	r	e	3	y	a	t	_	5
t	5	3	e	r	_	1	y	a

数独就是每个单元格所在行所在列，该字符都是唯一的。

做出来了但是不知道密码在哪里，看到维吉尼亚的数字，应该是每个分组指向一个字符，合起来就是密码。如45指向1，4行5列。以此类推....得到**15_1t_3a5y。**

这就是flag，但是要加上HEBTUCTF{}，题目没说明。

CRC32 BOOM!

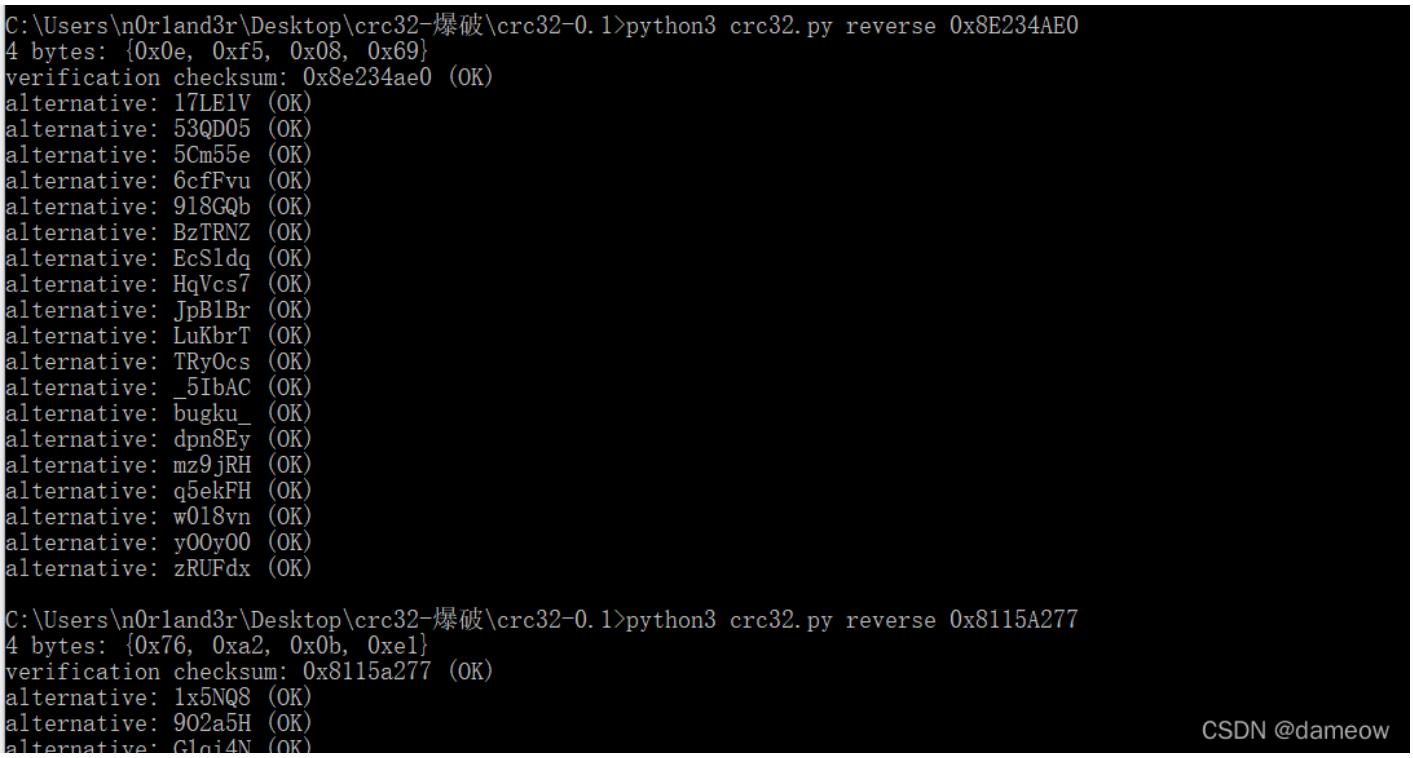
给了一个三个文件。

C:\Users\n0r1and3r\Desktop\CRC32_BOOM (1).zip\

名称	压缩后大小	修改时间	创建时间	访问时间	属性	加密	注释	CRC	算法
1.txt	18	2019-04-11 22:46	2019-03-24 17:17	2019-04-11 22:46	A	+		8E234AE0	ZipCrypto Store
2.txt	18	2019-04-11 22:45	2019-03-24 17:17	2019-04-11 22:45	A	+		8115A277	ZipCrypto Store
flag.jpg	80	2019-03-27 23:56	2019-03-24 17:21	2019-04-11 22:36	A	+		26834817	ZipCrypto Deflate

因为1.txt和2.txt比较小，可以爆破。flag.jpg估计是用爆破出来的crc32再去爆破它。

爆破脚本：https://github.com/theonlypwner/crc32



这里爆破出来的是文件内容，并不是压缩包的密码。

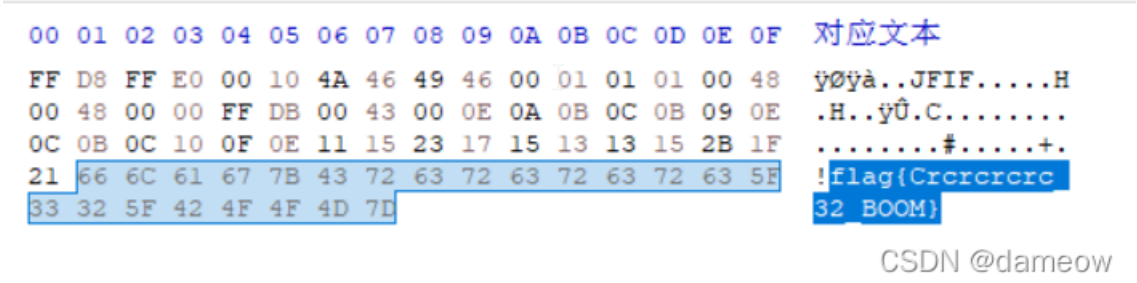
把两个文件爆破的结果进行组合，制作成爆破flag.jpg的字典。

```
dic = []
# 已t1开头，t2结尾的字典
for k in range((len(t1)+len(t2))):
for i in t1:
for j in t2:
dic.append(i+j)
# 已t1开头，t2结尾的字典
for k in range((len(t1)+len(t2))):
for i in t2:
for j in t1:
dic.append(i+j)
print(dic)

with open('dic.txt','w') as f:
for i in dic:
f.write(i+'\n')
```

```
'Rh5f6k',
'XBiGbJ',
'Z3AdV_',
'qsombC',
'mY3L6b',
'newctf',
'p7plP7',
't3mmQT',
'vBENeA'
]

dic = []
# 已t1开头，t2结尾的字典
for k in range((len(t1)+len(t2))):
    for i in t1:
        for j in t2:
            dic.append(i+j)
# 已t1开头，t2结尾的字典
for k in range((len(t1)+len(t2))):
    for i in t2:
        for j in t1:
            dic.append(i+j)
print(dic)

with open('dic.txt','w') as f:
    for i in dic:
        f.write(i+'\n')
```


然后使用archpr等工具进行字典爆破，或者ziperello。

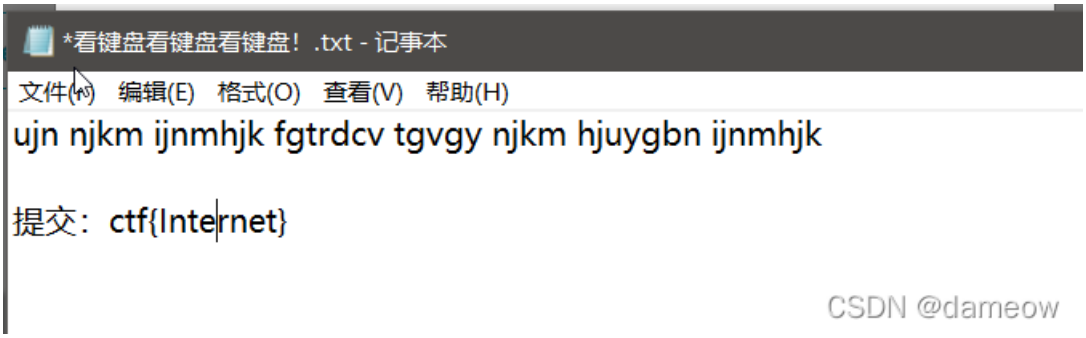
把解密出来的flag.jpg扔进16进制工具看到flag，这个图片估计是损坏的。



看键盘

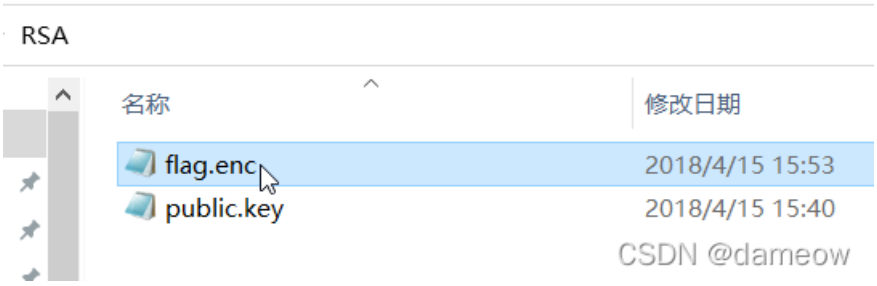
ujn njkm ijnmhjk fgtrdcv tgvgy njkm hjuygbn ijnmhjk

实际上，就是在键盘上，安装这个字母比划，每组就是划一个单词，第一个ujn是比划i，njkm是比划n，第三个是比划t，一次类推，最后是Internet。



可怜的RSA

给了两个文件，flag.enc是加密后的密文，public.key是公钥文件。



相当于我们知道了c，n，e。n和e可以从公钥文件中分离出来。

分离方法：yafu工具。

```
C:\Users\n0rland3r\Desktop\yafu-1.34>yafu-x64.exe "factor(@)" -batchfile test.txt


=== Starting work on batchfile expression ===
factor(798321817573328185527646107613495929846147444322791353283989998016278802836109003612812499731758050699162101795605064970751325249020868811203722136266418794684919368609766869336308696738269726199383219515991467448076533010760265779495796183315027763039834855660464854310395417084671414082602200985927612450106785923475018941762695805104597296336734680684671441997445637318263621026088110334008878137547802826280994434901700160878386069980174904566013158024485677724116238262817472456609542454137815197942953361975556885435379921971422580532204537576665378402764164756027593749507152838902322307415427373195698197939884314431
fac: factoring 798321817573328185527646107613495929846147444322791353283989998016278802836109003612812499731758050699162101795605064970751325249020868811203722136266418794684919368609766869336308696738269726199383219515991467448076533010760265779495796183315027763039834855660464854310395417084671414082602200985927612450106785923475018941762695805104597296336734680684671441997445637318263621026088110334008878137547802826280994434901700160878386069980174904566013158024485677724116238262817472456609542454137815197942953361975556885435379921971422580532204537576665378402764164756027593749507152838902322307415427373195698197939884314431443
fac: using pretesting plan: normal
fac: no tune info: using qs/gnfs crossover of 95 digits
div: primes less than 10000
fmt: 1000000 iterations
rho: x^2 + 3, starting 1000 iterations on C623
rho: x^2 + 2, starting 1000 iterations on C623
Total factoring time = 0.2037 seconds


***factors found***

P7 = 3133337
PRP617 = 254783260649374192922001721363994977190818429145282283164559062116931183219713999360047291348411629741442462714864396957860365881174246118819559509962196468073788222782856382615820991083394389495730341012151411561564087428438200480668308638143623798857203950823184628500029016056897618763191511473527300909575569408421442998873946787436077669378280944783364011594490358783068537162165483742734623865083073677131120730040113834189678949305540675824532489810220119228833744427368480459206763413618712317871634414675330768900817218821793691687872877247696426653999925560521448458786001262839688902730675753420617762449390 CSDN @dameow
```

即分离出来q，p，n=q*p。

接下来需要求私钥d。知道e，n就可以求d。

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

到这里就知道了n , e , d , q , p , c（密文c只是base64加密了而已）

本来这里pow（c, d, n）就可以出来了，但是c 和n 长度一样，明文空间m和密文空间c都必须满足0<c<n-1，所以这里需要对密文c进行分组来求，有点麻烦，而PKCS1_OAEP这个库可以padding，就直接用。

脚本：

```
import base64
from Crypto.Cipher import PKCS1_OAEP
from Crypto.PublicKey import RSA
import gmpy2

# 分离公钥文件，求n，e。也可以使用openssl：openssl rsa -pubin -text -modulus -in pubkey.pem
f = open('C:\\Users\\n0r1and3r\\Desktop\\RSA\\public.key','rb').read()
pub = RSA.importKey(f)
n = pub.n
e = pub.e

# 大素数分解，yafu工具
p = 3133337
q =
25478326064937419292200172136399497719081842914528228316455906211693118321971399936004729134841

# 得到私钥d
d = gmpy2.invert(e,(q-1)*(p-1))

key_info = RSA.construct((n,e,int(d),p,q))
key = RSA.importKey(key_info.exportKey())
key = PKCS1_OAEP.new(key)

# 读密文
f1 = open('C:\\Users\\n0r1and3r\\Desktop\\RSA\\flag.enc','rb').read()
C = base64.b64decode(f1)
flag = key.decrypt(C)
print(flag)
```

```
D:\Applications\python38\python3.exe D:/Codes/python/test.py
b'afctf{R54_|5_$0_B0rin9}'

Process finished with exit code 0
```

CSDN @dameow

small_RSA

给了一个加密脚本：

```
# -*- 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:
344246784248256132370323757453790755403533762276297110321055748005034935987304162433626178273

[+]N:
691316677109436623113422493782665795857921917893759942123087462879884062720557906429183155859

'''
```

有题目，可知三个参数，大素数n，密文c，加密指数e。仅凭这三个参数是不能直接求出明文m的，因为

$$m = cd \pmod n$$

不知道私钥d，d = gmpy2.invert(e，ϕ(n))，但是欧拉函数ϕ(n) = (q-1)*(p-1)，尝试去分解n，但是失败了，4096位加密目前是安全的。

切入点就在于e=3很小，可以低加密指数攻击。

根据加密函c =me mod n，相当于m3+k*n = c，k取整数。所以只需爆破这个k值即可。因为指数是3，比较低，可以在可计算范围爆破，如果一般e取65537等，就是不可计算范围了。

解密脚本：

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

n =
691316677109436623113422493782665795857921917893759942123087462879884062720557906429183155859
e = 3
flag = 0
c =
344246784248256132370323757453790755403533762276297110321055748005034935987304162433626178273
for i in range(200000000):
if gmpy2.iroot(c + n * i, 3)[1] == 1:
'''
这个函数是开n次方，gmpy2.iroot（x,n），x的n次方根。
返回值[0]是方根，[1]是布尔值。
如a = gmpy2.iroot(4,2)，则返回值a[0]=2,a[1]=TRUE。
'''
flag = gmpy2.iroot(c + n * i, 3)[0]
print(i, flag)
print(long_to_bytes(flag))
break
```

```
D:\Applications\python38\python3.exe D:/Codes/python/dd.py
0 150992936272969274072071353696839271564840218924230442440850179753819027376321959550
b'flag{4c466c3d094911ba3ca3319b43fe792bef9e94a19c8f666d2ec6c890034d88ba}'

Process finished with exit code 0
```


这里解得i=0，即k=0的情况。

山东省大学生网络技术大赛-baby

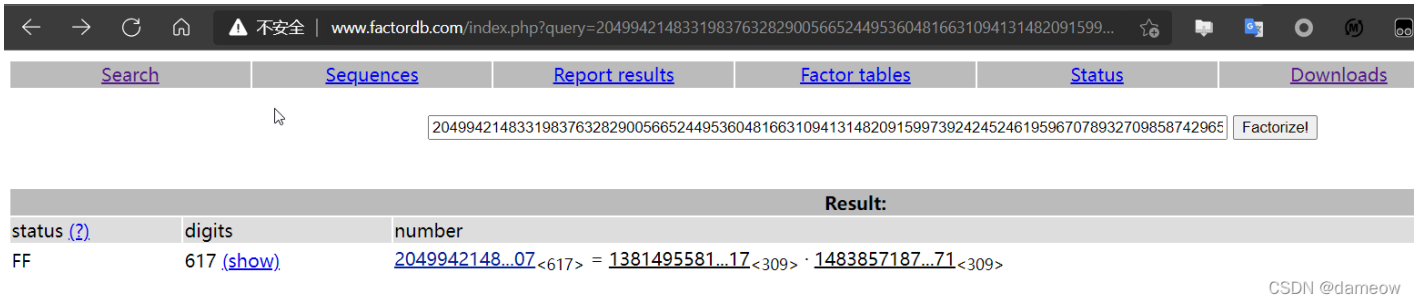
和上面一题small_RSA 是同一题，略过。

easy_rsa

给出了大素数n，密文c，加密指数e。

按照常规历流程，先把n分解，得到两个大质数qp。

工具：factordb.com



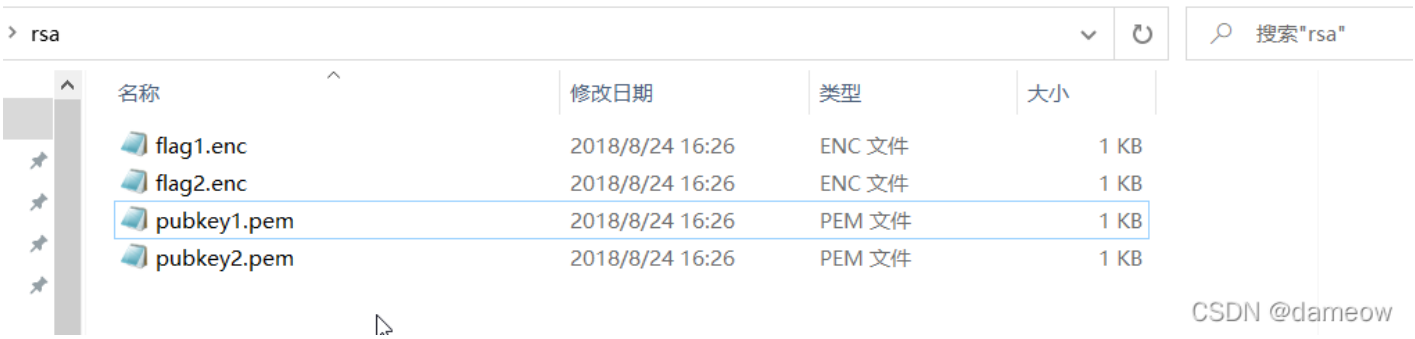
有了qp就好办了。脚本：

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

flag{Eeeasy_RSA!}
```

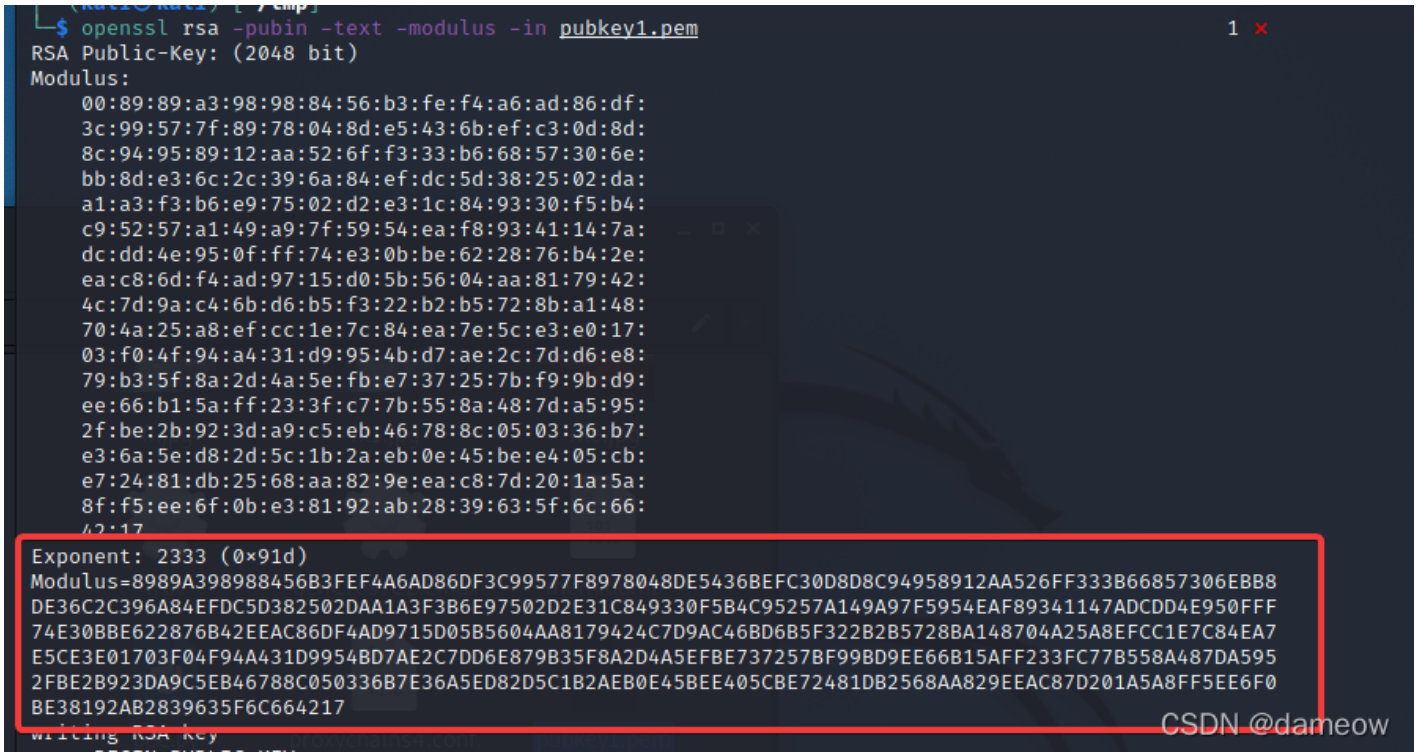
rsass

爆破压缩包，密码为password。

里面是四个文件：



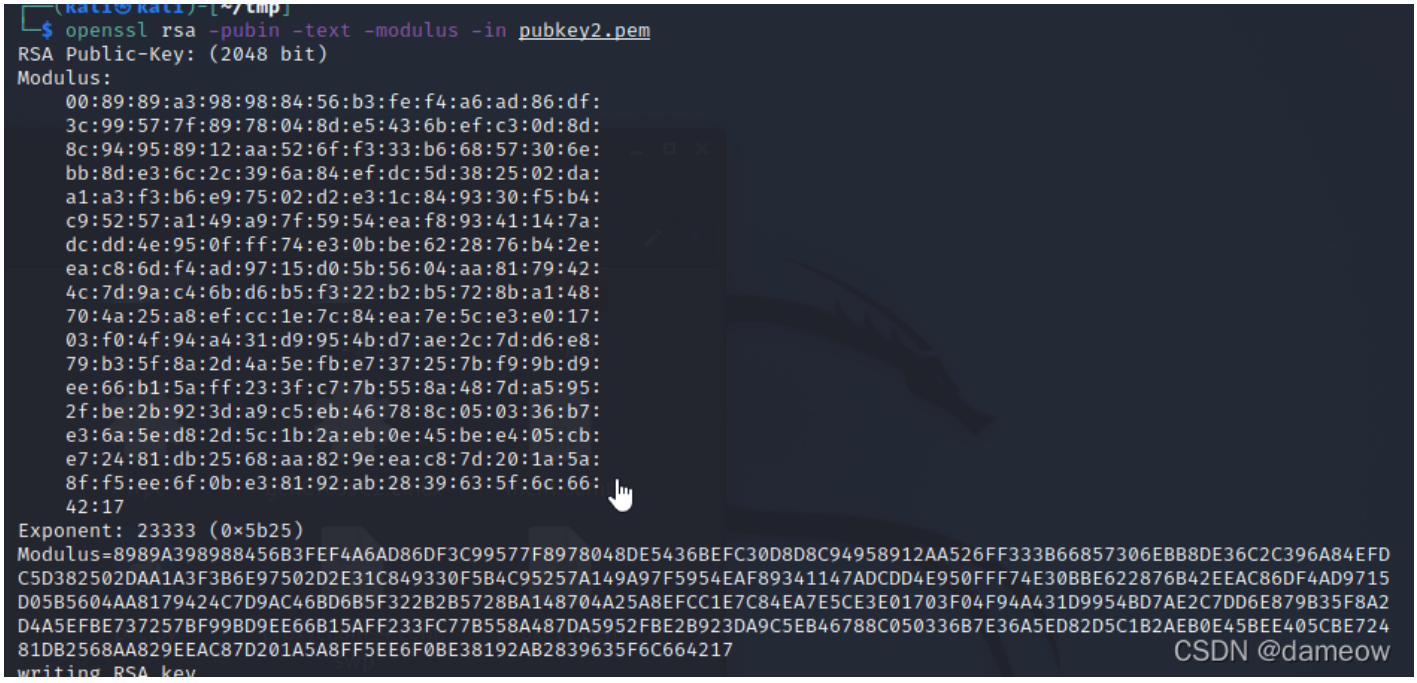
先从pubkey1.pem分离n，e试试看。



密钥强度2048bit。

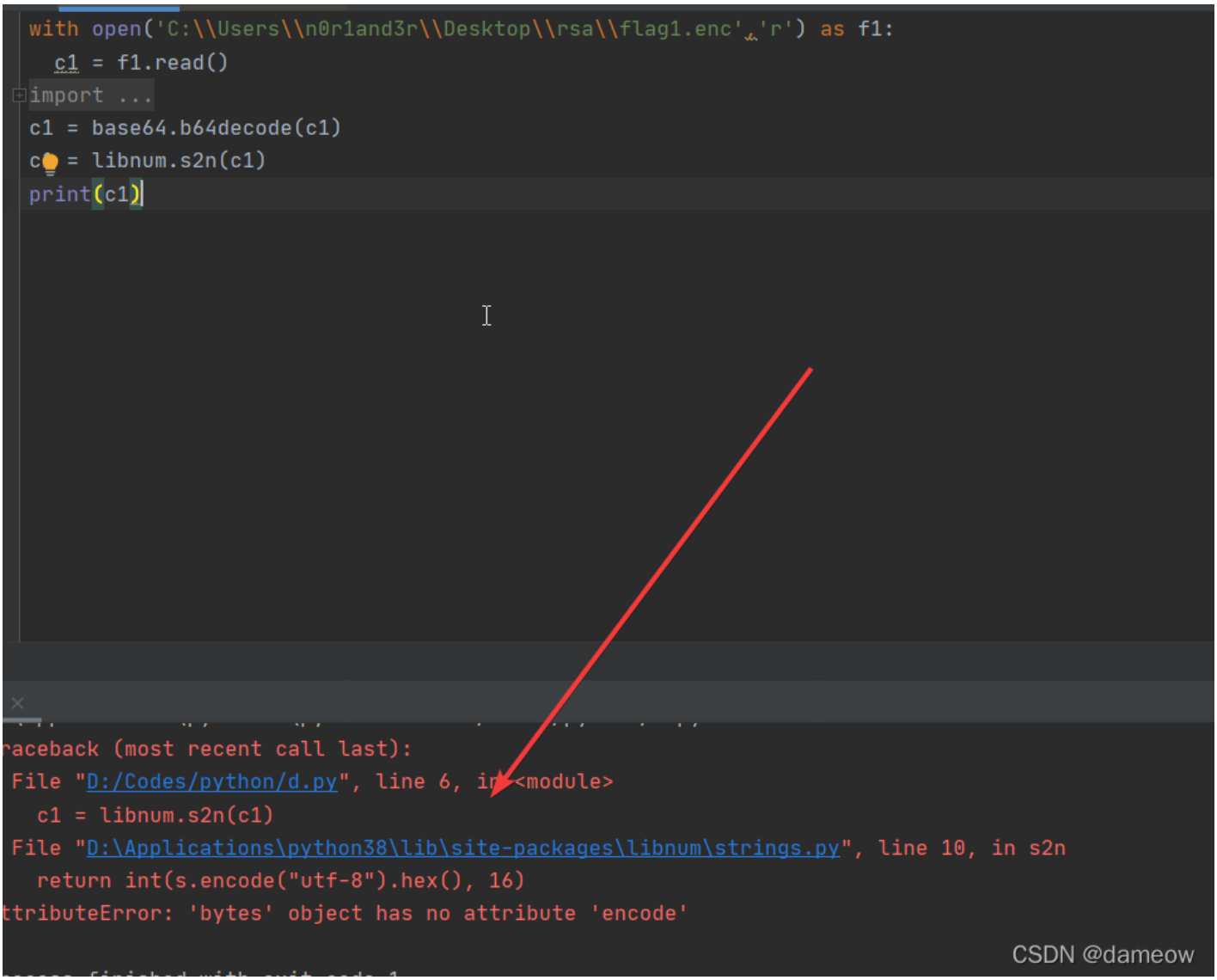
e = 2333，n = 17362520124149736059291605717839814089431261833972408175766504894876091272021197374480217

看看pubkey2.pem。

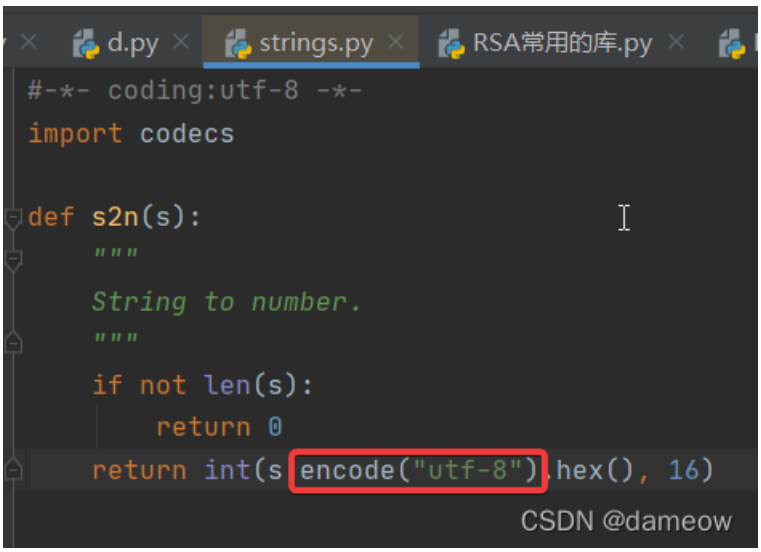


两个模数n是一样的。所以，应该是考察共模攻击。

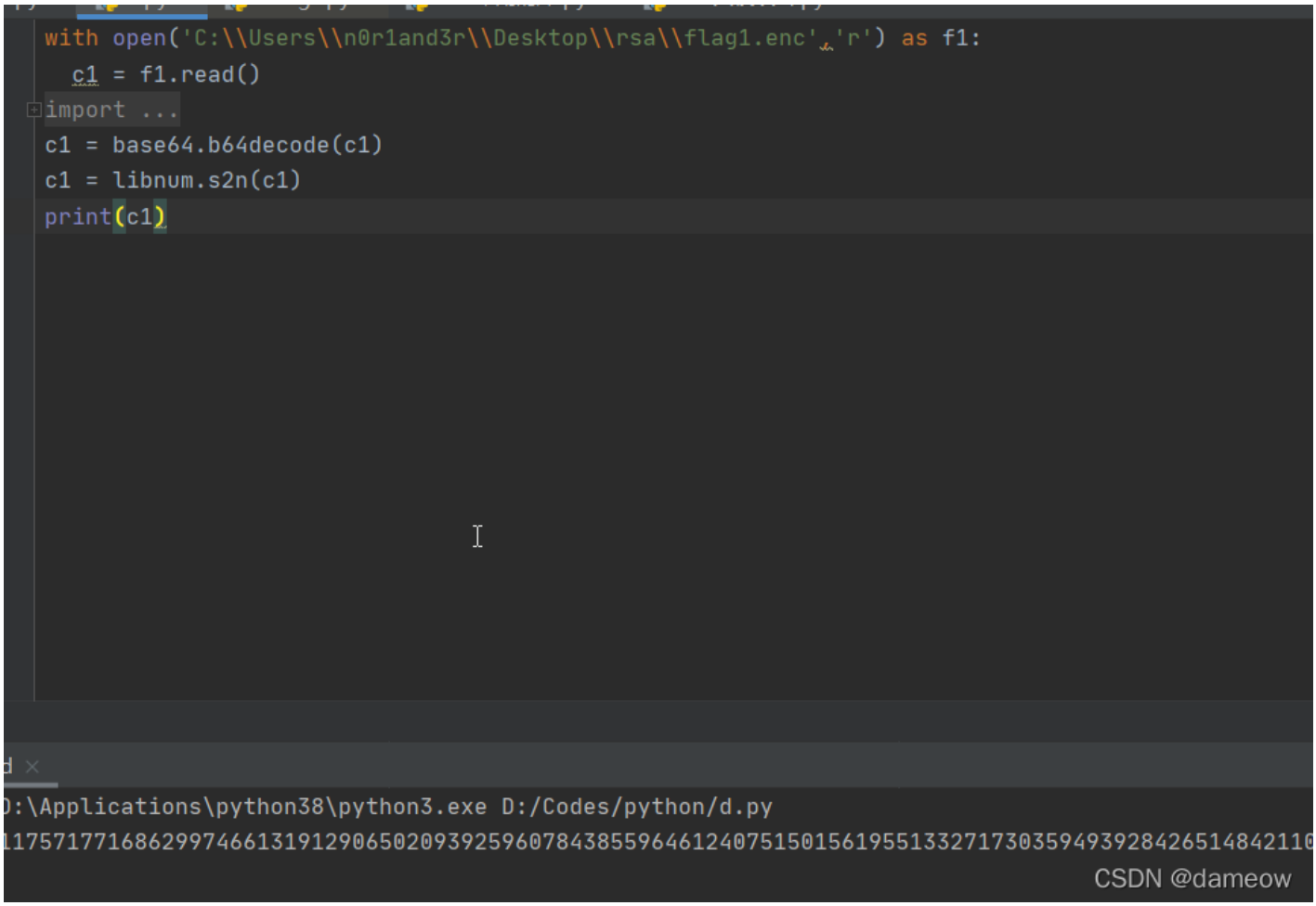
先把两个密文读取出来。这里有个小挫折。



直接读取报错了，点击这个strings.py文件，把encode('utf-8')删掉。

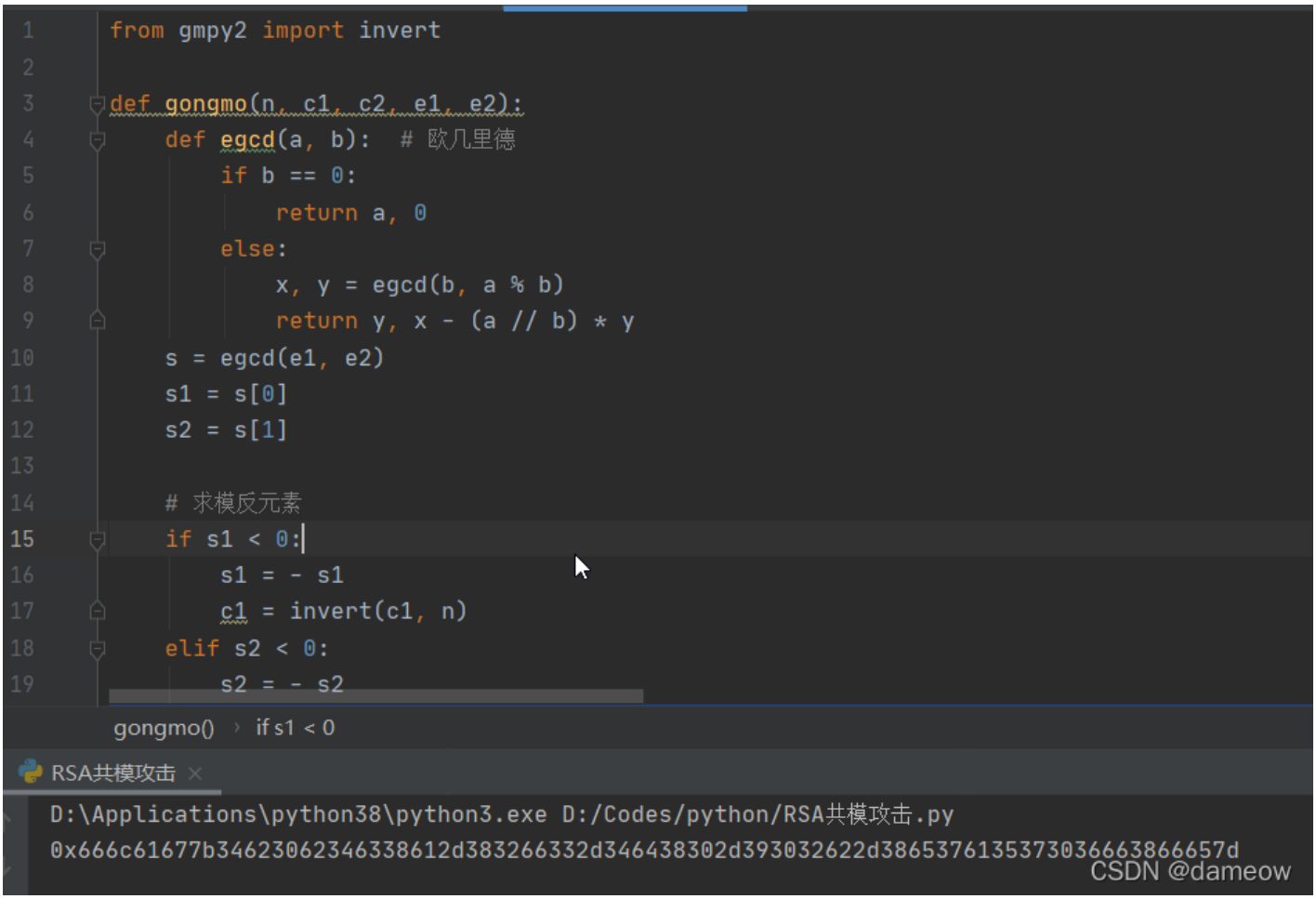


然后可以正常读取了。


with open('C:\\Users\\n0r1and3r\\Desktop\\rsa\\flag1.enc','r') as f1:
c1 = f1.read()
import ...
c1 = base64.b64decode(c1)
c1 = libnum.s2n(c1)
print(c1)
D:\Applications\python38\python3.exe D:/Codes/python/d.py
CSDN @dameow

共模攻击脚本：

```
from gmpy2 import invert

def gongmo(n, c1, c2, e1, e2):
def egcd(a, b): # 欧几里德
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
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=
173625201241497360592916057178398140894312618339724081757665048948760912720211973744802155825
e1= 2333
e2= 23333

with open('C:\\Users\\n0r1and3r\\Desktop\\rsa\\flag1.enc','r') as f1:
c1 = f1.read()
import base64
import libnum
c1 = base64.b64decode(c1)
c1 = libnum.s2n(c1)

with open('C:\\Users\\n0r1and3r\\Desktop\\rsa\\flag2.enc','r') as f1:
c2 = f1.read()
c2 = base64.b64decode(c2)
c2 = libnum.s2n(c2)

flag = gongmo(n, c1, c2, e1, e2)
flag = hex(flag)
print(flag)
```



16进制转一下字符串得flag，大坑（横杠要去掉，这也太坑了）

【2021医疗行业CTF】base编码

R1kzRE1RWldHRTNET04yQ0dZWkRNTUpYR00zREtNWldHTTJES1JSVEdNWlRFTktHR01ZVEdOUlZJWTN

一次base6解码，一次base32解码，一次16进制转字符串。

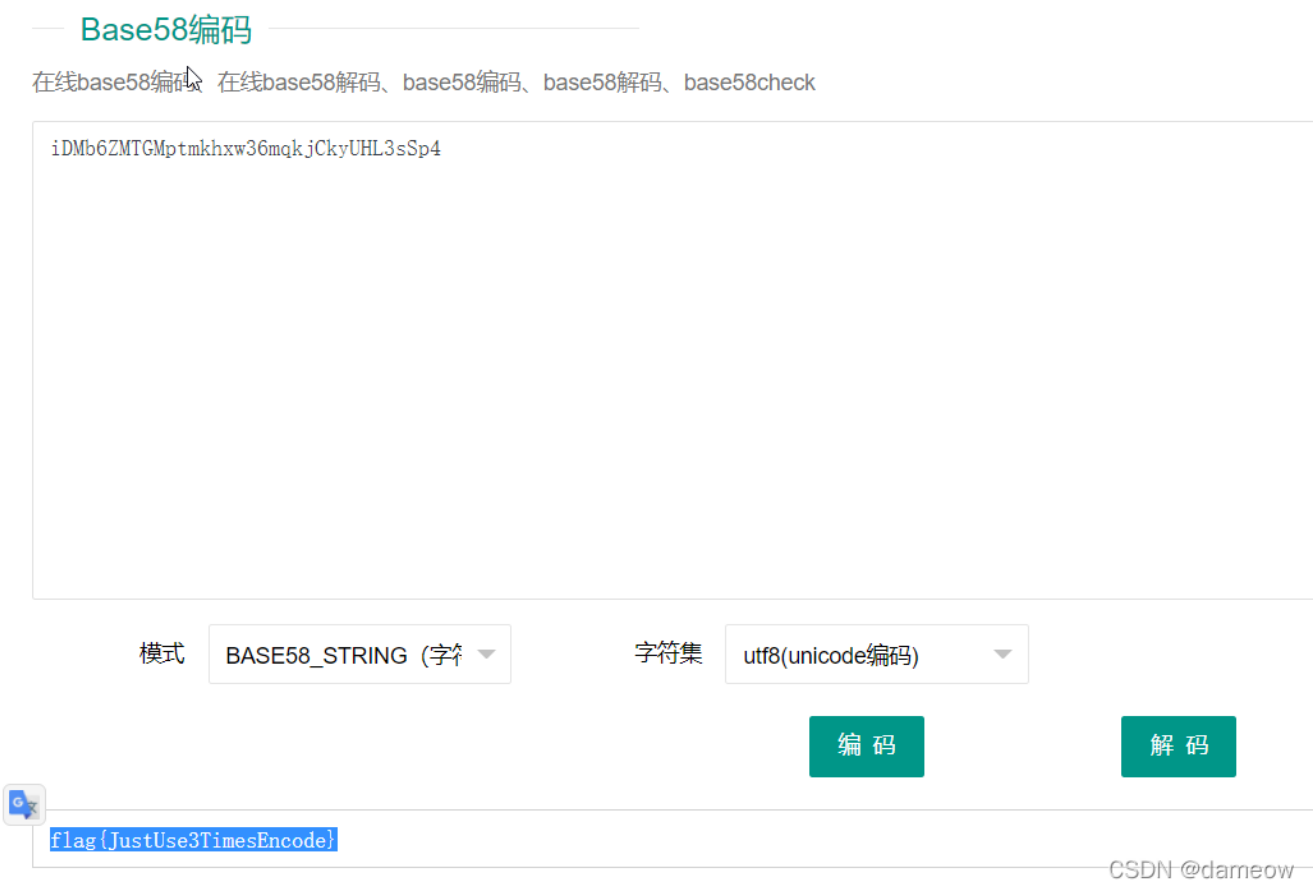
看的出来吗

bE0veldtTDs7NzlTe3hzbSFYSj5Sa2U6eyQ4NyVrl3FvWFU6Qls7QlVK

密钥是589164。

其实就是先base58编码，再base91编码，再base64编码。

反过来解码就行。



键盘之争

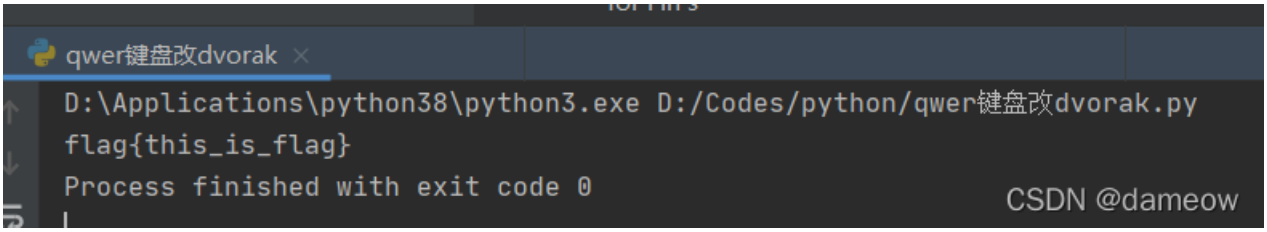
肯定是改了键盘布局，我们常用的键盘布局时qwer布局，还有dvorak布局，colema布局等。

qwer布局改dvorak布局脚本：

```
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='')
```



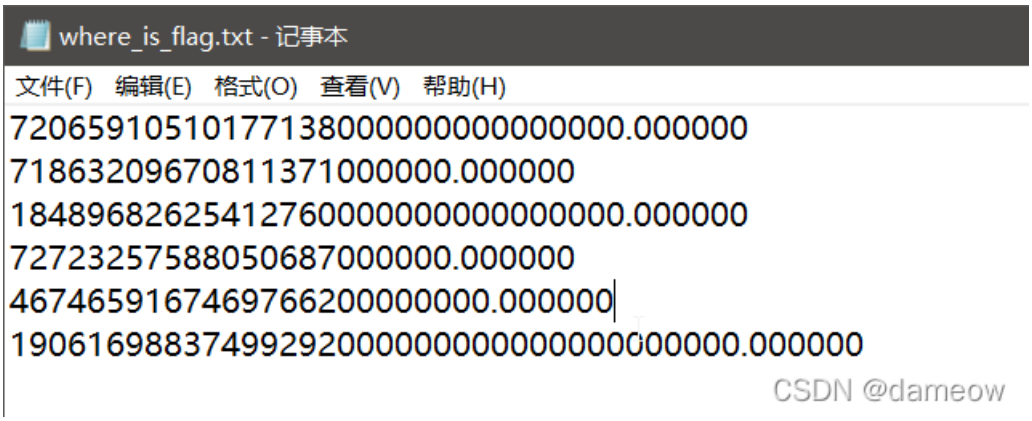
把this_is_flag改为它的MD5值再提交。

2018 HEBTUCTF Simple Caesar!

UROGHPGS{f1zcyr_pnrf4e_f0_fvzc1r}

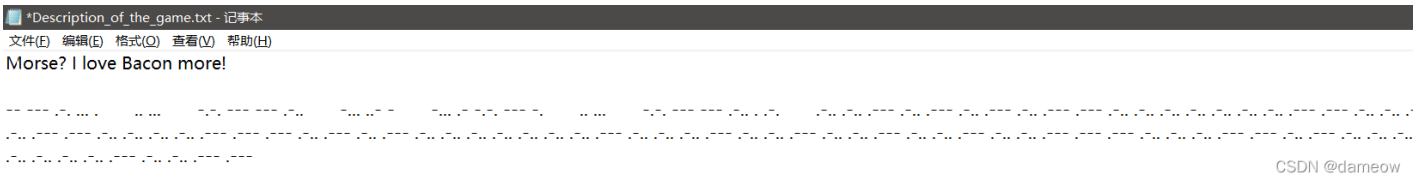
移位13的凯撒密码，即rot13。

2018 AFCTF MagicNum

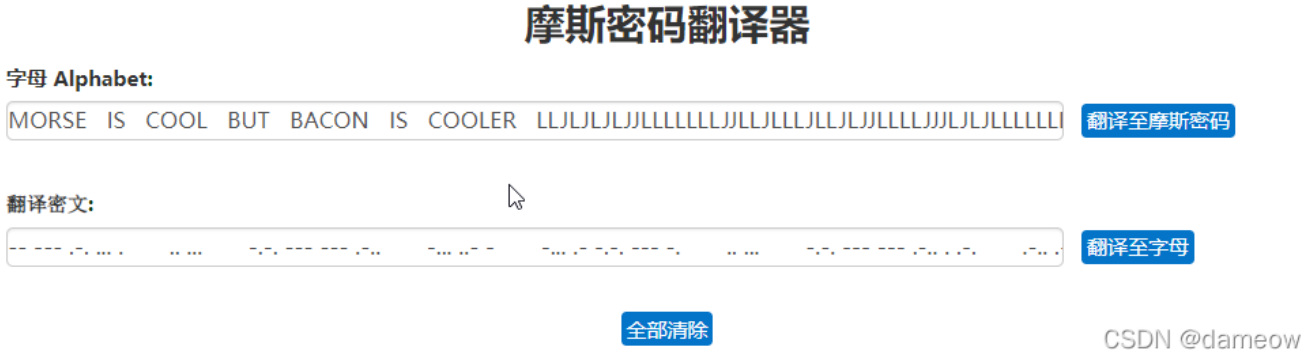


这

2018 HEBTUCTF easy_crypto



第一层是摩斯电码，第二层是baconian培根密码。



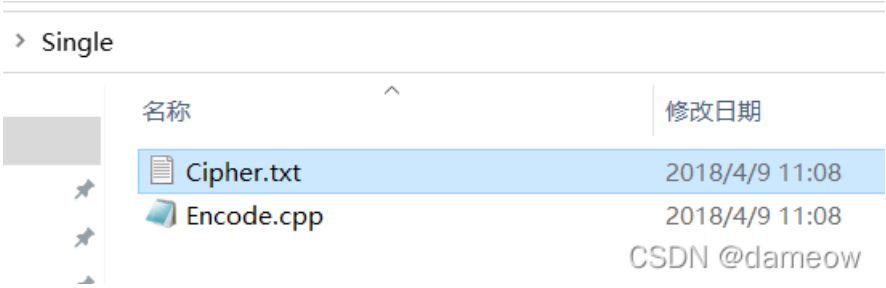
明显后面那串是培根密码，把L和J换乘A和B去解密培根，或者反过来。

```
please input string to decode:
AABABABABBAAAAAAABBAABAAABAABABBAAAABBBABABAAAAAAABAAABAABAABAABAABBBAAABBABAAABAABAAAAAAABAABB
first decode method result is:
FLAGISYOUARESOGREAT
second decode method result is:
FMAGIJTPWASETPGSEAUV
```


flag格式：HEBTUCTF{YOUARESOGREAT}

2018 AFCTF Single

两个文件，一个加密源码，一个加密后的密文。



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.

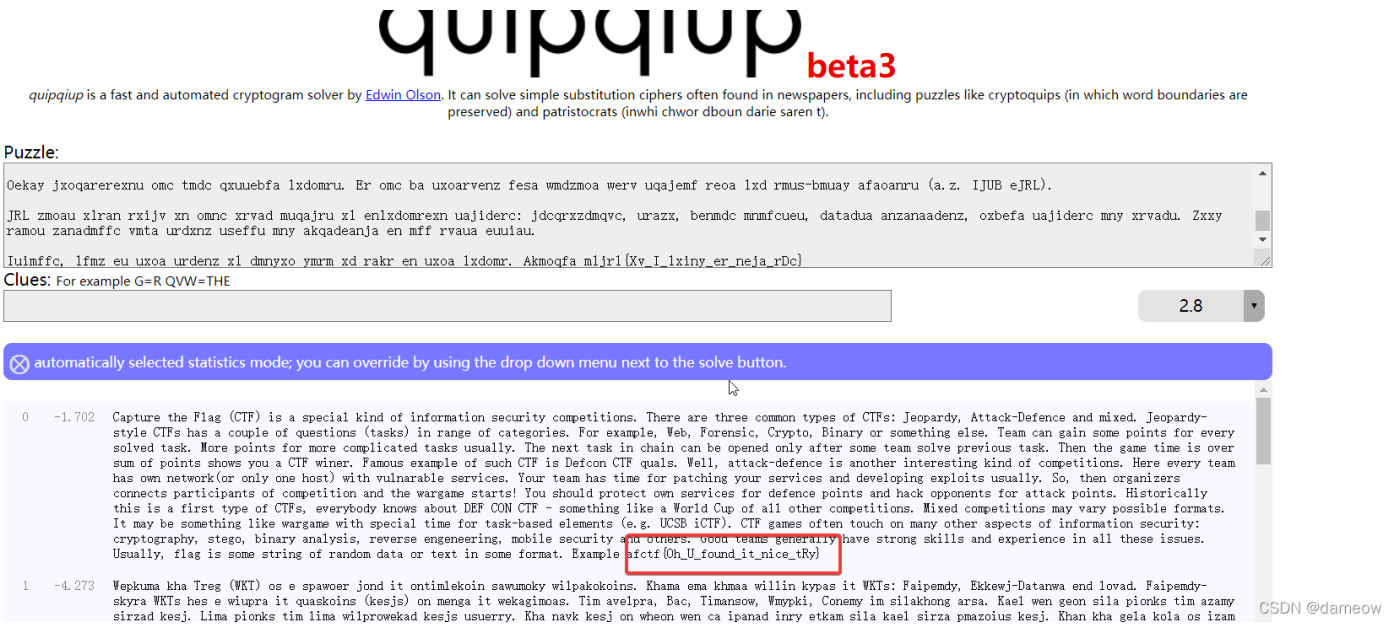
Waff, mrrmjs-yalanja eu mnxrvad enradaurenz seny xl jxoqarerexnu. Vada atadc ramo vmu xwn narwxds(xd xnfc xna vxur) werv tifnmdmbfa uadtejau. Cxid ramo vmu reoa lxd qmrjvenz cxid uadtejau mny yatafxqenz akqfxeru iuimffc. Ux, rvan xdzmnehadu jxnnajru qmdrejeqmnru xl jxoqarerexn mny rva wmdzmoa urmdru! 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 rxiiv xn omnc xrvad mugairu xl enlxdomrexn uajiderc: idcqrxzdmqvc, urazx, benmdc

```cpp
#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;
}
```


看到密文，首先应该想到词频分析。quipqiup - cryptoquip and cryptogram solver



也不用管它是什么加密了。

2018 AFCTF 你能看出这是什么加密么

```
p=0x928fb6aa9d813b6c3270131818a7c54edb18e3806942b88670106c1821e0326364194a8c49392849432b37632f0abe3f3c52e909b939c91c50e41a7b8cd00c67d6743b4f

q=0xec301417ccdffa679a8dcc4027dd0d75baf9d441625ed8930472165717f4732884c33f25d4ee6a6c9ae6c44aedad039b0b72cf42cab7f80d32b74061

e=0x10001

c=0x70c9133e1647e95c3cb99bd998a9028b5bf492929725a9e8e6d2e277fa0f37205580b196e5f121a2e83bc80a8204c99f5036a07c8cf6f96c420369b4161d2654a7eccbdaf583204b645e137b3bd15c5ce865298416fd5831cba0d947113ed5be5426b708b89451934d11f9aed9085b48b729449e461ff0863552149b965e22b6
```


这一看就是RSA，但是我看出来了，它并没有直接给我flag。。。。

好吧，还是要解的。

给了p，q，e，c。

解密脚本：

```
p=0x928fb6aa9d813b6c3270131818a7c54edb18e3806942b88670106c1821e0326364194a8c49392849432b37632f0
q=0xec301417ccdffa679a8dcc4027dd0d75baf9d441625ed8930472165717f4732884c33f25d4ee6a6c9ae6c44aedad0
e=0x10001
c=0x70c9133e1647e95c3cb99bd998a9028b5bf492929725a9e8e6d2e277fa0f37205580b196e5f121a2e83bc80a8204

p = int(p)
q = int(q)
e = int(e)
c = int(c)
fn = (q-1)*(p-1)
n = q*p
import gmpy2
d = gmpy2.invert(e,fn)
m = pow(c,d,n)

from Crypto.Util.number import *
m = long_to_bytes(m)
print(m)
```

easy_CRC

1:C4E68E3A

2:08FF301C

3:F6FBA587

4:B85C2F9A

5:F1B73D20

6:C6F1D9D7

flag=1+2+3+4+5+6

Each serial number consists of 4 characters

CRC32的爆破，flag就是解密后123456串起来，提示了明文是4个字符，直接爆破。

```
C:\Users\n0r1and3r\Desktop\crc32-master>python3 crc32.py reverse 0xC4E68E3A
4 bytes: b323 {0x62, 0x33, 0x32, 0x33}
verification checksum: 0xc4e68e3a (OK)
5 bytes: Jat9w (OK)
6 bytes: 1_JjbO (OK)
6 bytes: 6FMTHd (OK)
6 bytes: 7ZC9Ri (OK)
6 bytes: 8IRdtj (OK)
6 bytes: DF9OAy (OK)
6 bytes: KIgNfn (OK)
6 bytes: SnUcwI (OK)
6 bytes: XDHs8q (OK)
6 bytes: ZxMmal (OK)
6 bytes: bPLyKN (OK)
6 bytes: e9w6d5 (OK)
6 bytes: eIKGae (OK)
6 bytes: g8cdUp (OK)
6 bytes: guNY8x (OK)
6 bytes: n246BA (OK)
6 bytes: safZLL (OK)
6 bytes: txadfg (OK)
```

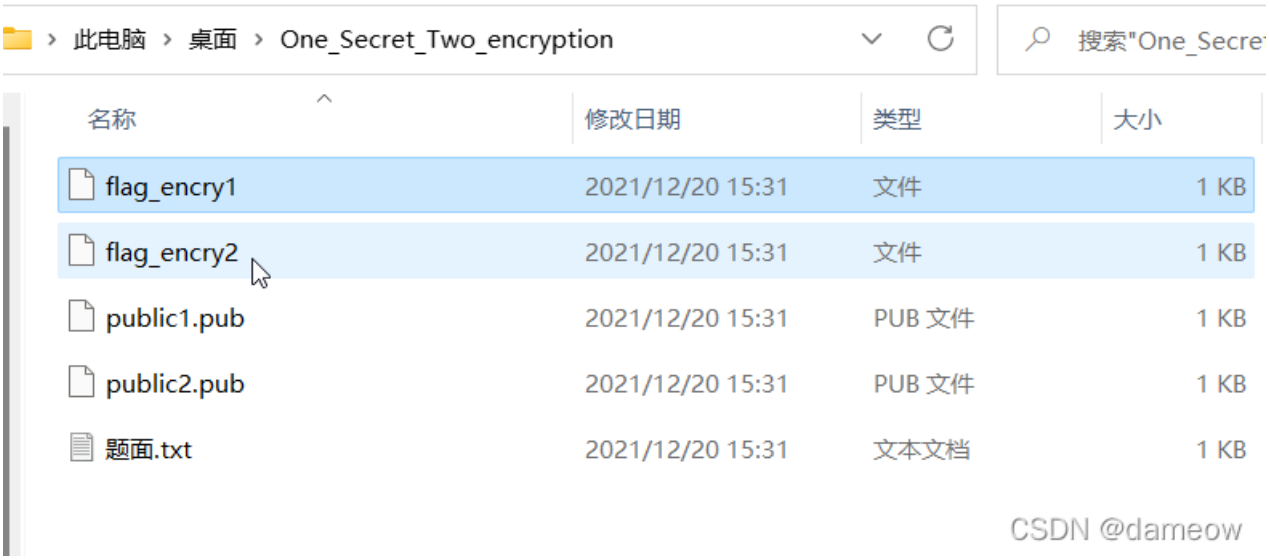

第一个解密得b323，这个脚本直接出字符串，不能额外16进制转字符串，比较好用。

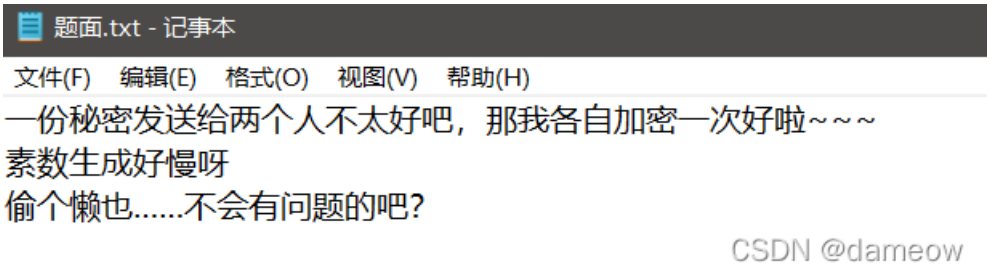
按照这个方法，直接破解6个CRC，得到b3231e94f045e76e593266f4，即为flag。

实际上原理就像哈希碰撞，用可打印字符一一加密，看密文是否和这个次密文一样，一样就得出明文。

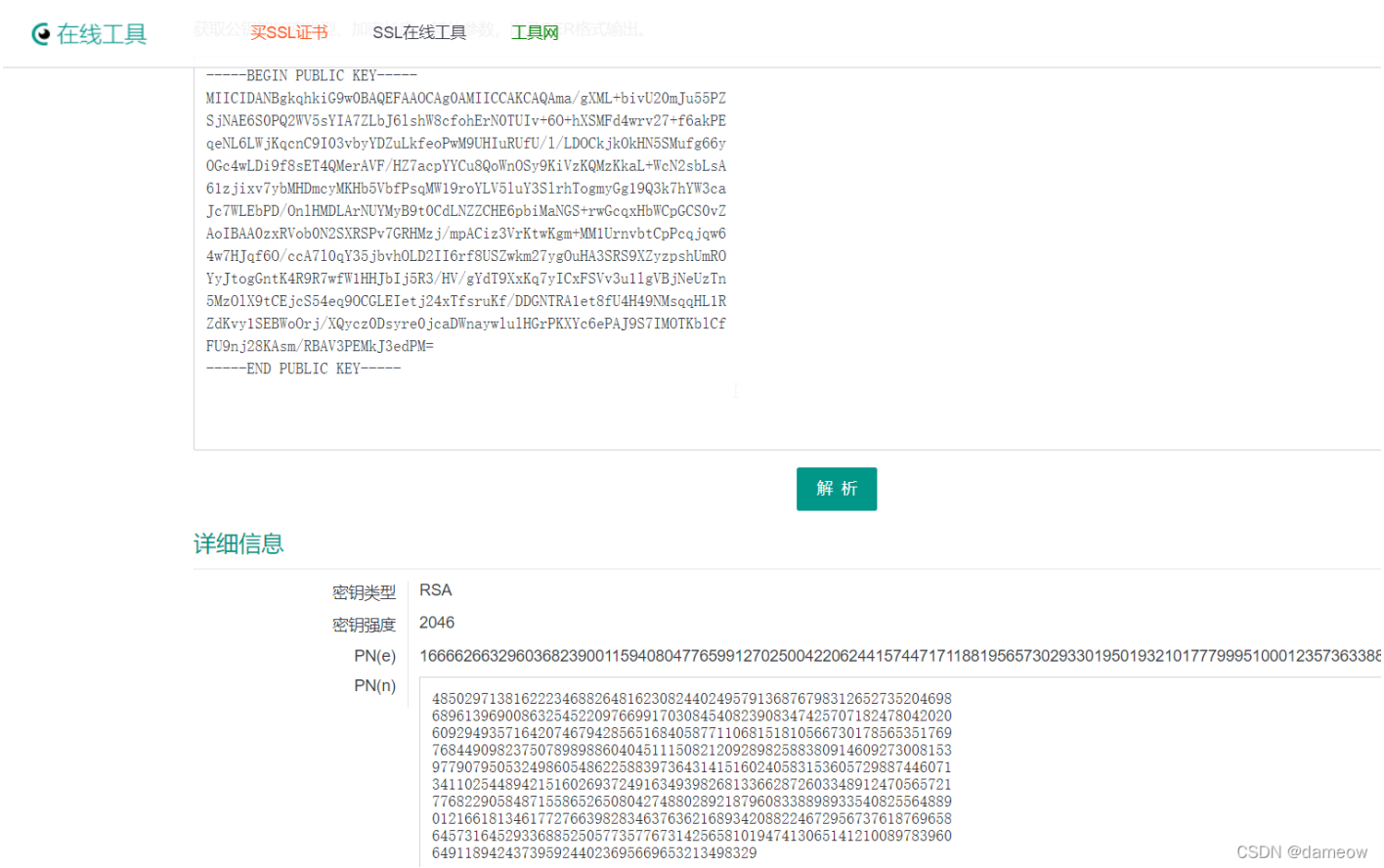
2018 AFCTF One Secret, Two encryption

两个公钥，两个密文。





直接对公钥进行提取e,n。



两个公钥得到如下结果：

e1 =

166662663296036823900115940804776599127025004220624415744717118819565730293301950193210

n1 =

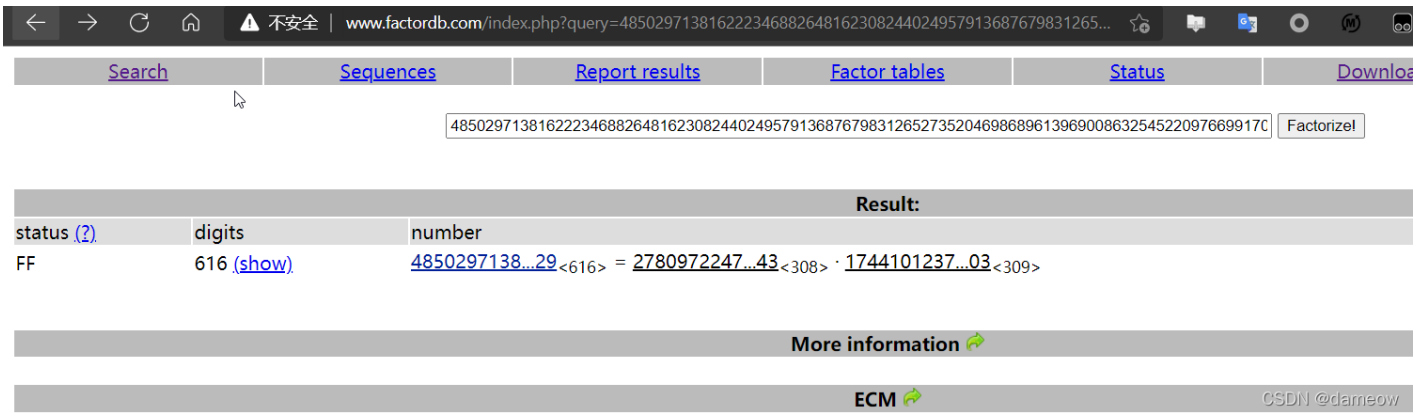
485029713816222346882648162308244024957913687679831265273520469868961396900863254522097

e2 = 65537

n2 =

236753676867200095966818117178729527189878928839767299713484341893240595994673963736804

意外解：直接对n1或者n2进行因数分解，可以直接分解出pq。



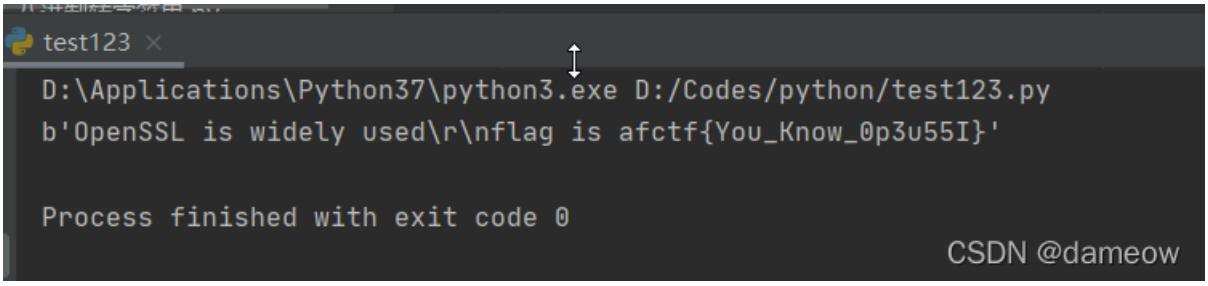
得到：

q1 =
27809722472252756488236572384949349891208643090117349509994417047989746484576130392206782

p1 =
17441012376163133752079917980859812791418497197881179672241421523987411404834783060925582

解密脚本：

```
e1 =
16666266329603682390011594080477659912702500422062441574471711881956573029330195019321017779995
n1 =
48502971381622234688264816230824402495791368767983126527352046986896139690086325452209766991706
p1 =
27809722472252756488236572384949349891208643090117349509994417047989746484576130392206781875746
q1 =
17441012376163133752079917980859812791418497197881179672241421523987411404834783060925580520317

import gmpy2
d1 = gmpy2.invert(e1,((q1-1)*(p1-1)))
import rsa
key1 = rsa.PrivateKey(n1,e1,int(d1),p1,q1) # 生成rsa私钥，用这个私钥来解密密文文件flag.enc
with open('C:\\Users\\n0r1and3r\\Desktop\\One_Secret_Two_encryption\\flag_encry1','rb') as f1:
c1 = f1.read()
print(rsa.decrypt(c1,key1))
```



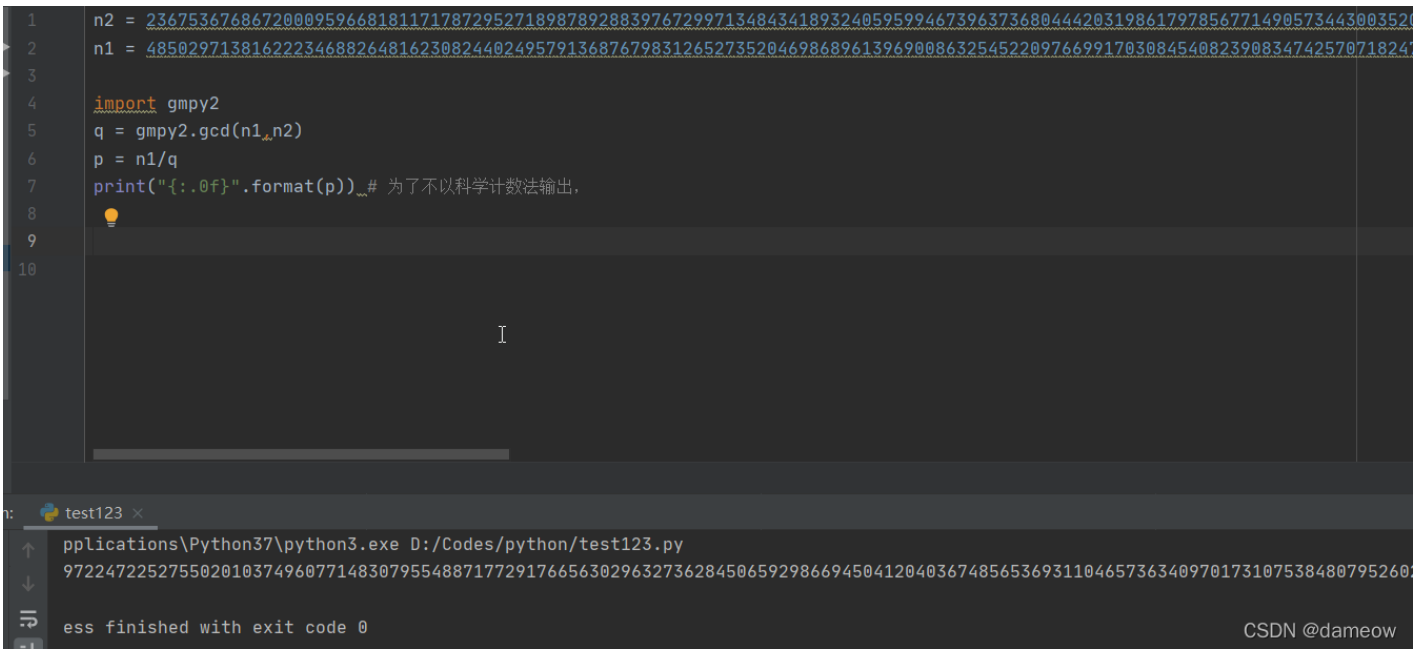
还有另外一种解法，可能这才是考察的方法。

求n1和n2的公约数，因为题目偷懒了，估计使用的同一个素数。

得到公约数

1744101237616313375207991798085981279141849719788117967224142152398741140483478306092558

由于n = q*p，假设公约数是q，则p = n除以q。



得到e1,n1,q1,p1，则解密和上面的解密脚本一致。

Crypto_easy_crypto

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

def keygen(nbit):
while True:
p, q, r = [getPrime(nbit) for _ in range(3)] # 生成三个素数qpr
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]) # enc = m^e mod n
return enc

nbit = 512
pubkey, _ = keygen(nbit)
print('pubkey =', pubkey)

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

这是一个RSA的加密过程。输出给了三个信息：p,q,c。

#pubkey =
(763929224901239050077647342425144363318006219360210465113458622937882119766705458273788
27750416681837900468631425875797804482827864226099175414717825894823303299159277202974 07
#enc =

75792946030351358172345011312562823242401324242720009030358010738472229173060927926104 06

从enc = pow(bytes_to_long(msg.encode('utf-8')), 0x10001, pubkey[0] * pubkey[1]) 这条加密代码，可以得到，msg是flag，e = 0x10001=65537，

pubkey[0] =
763929224901239050077647342425144363318006219360210465113458622937882119766705458273788 9

pubkey[1] =
27750416681837900468631425875797804482827864226099175414717825894823303299159277202974 07

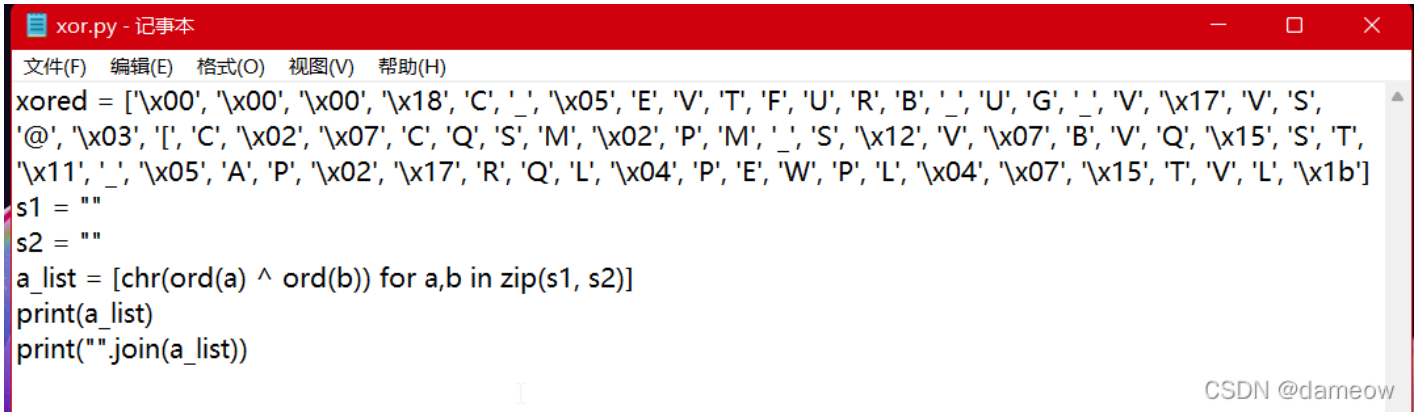
看了wp，实在是不明白为什么脚本中pubkey[1]可以充当模数n。

解密脚本：

```
pub0 =
763929224901239050077647342425144363318006219360210465113458622937882119766705458273788947718911994090187932
947694326848868575500906975653763043489419853300731695950407389203582333794360159494405111004208058198680365
172060235321945875374635278292661761319773173838900295933943437761251440819434675631450032782188481758975272
071015244168751016874668742536151359822585793537758876078350987062972237271763834743266722655055439868971805
843593929839097963319788083763
pub1 =
277504166818379004686314258757978044828278642260991754147178258948233032991592772029740709036302768682487556
426088849031227686564271654015639204434821512 17

c =
757929460303513581723450113125628232424013242427200090303580107384722291730609279261040687645169812195623204
703163944876732347508817035786365343509674664060932552503532932862656214304965612424126357664997482053380016
143266694632722641803682614616028827317552186468712272083679559663506776114651994044181776573216038201843446
577674937220918121281724718747468524615619338182204477274665469012674671572816652727493363475044347868281266
402200944434844895572022657946878192777613333604552514920338413785986815307782070816714605242901479066092158
695866075994939982656813675581656346873911112376128831827637902501870888352 1
```

```
e = 65537
import gmpy2
d = gmpy2.invert(e,pub1-1)
m = pow(c,d,pub1)

import libnum
flag = libnum.n2s(int(m))
print(flag)
```

Crypto_xor



题目提示了['\x00', '\x00', '\x00'] at start of xored is the best hint you get

xored是异或后的结果，开头三个/x00，只能是ctf与ctf的异或，相同字母异或得到0，下一个/x18，用/x18和{的异或结果是c，所以flag是与ctfctfctf.....的异或结果得到xored。

则flag可以是xored和ctfctfctf.....的结果。

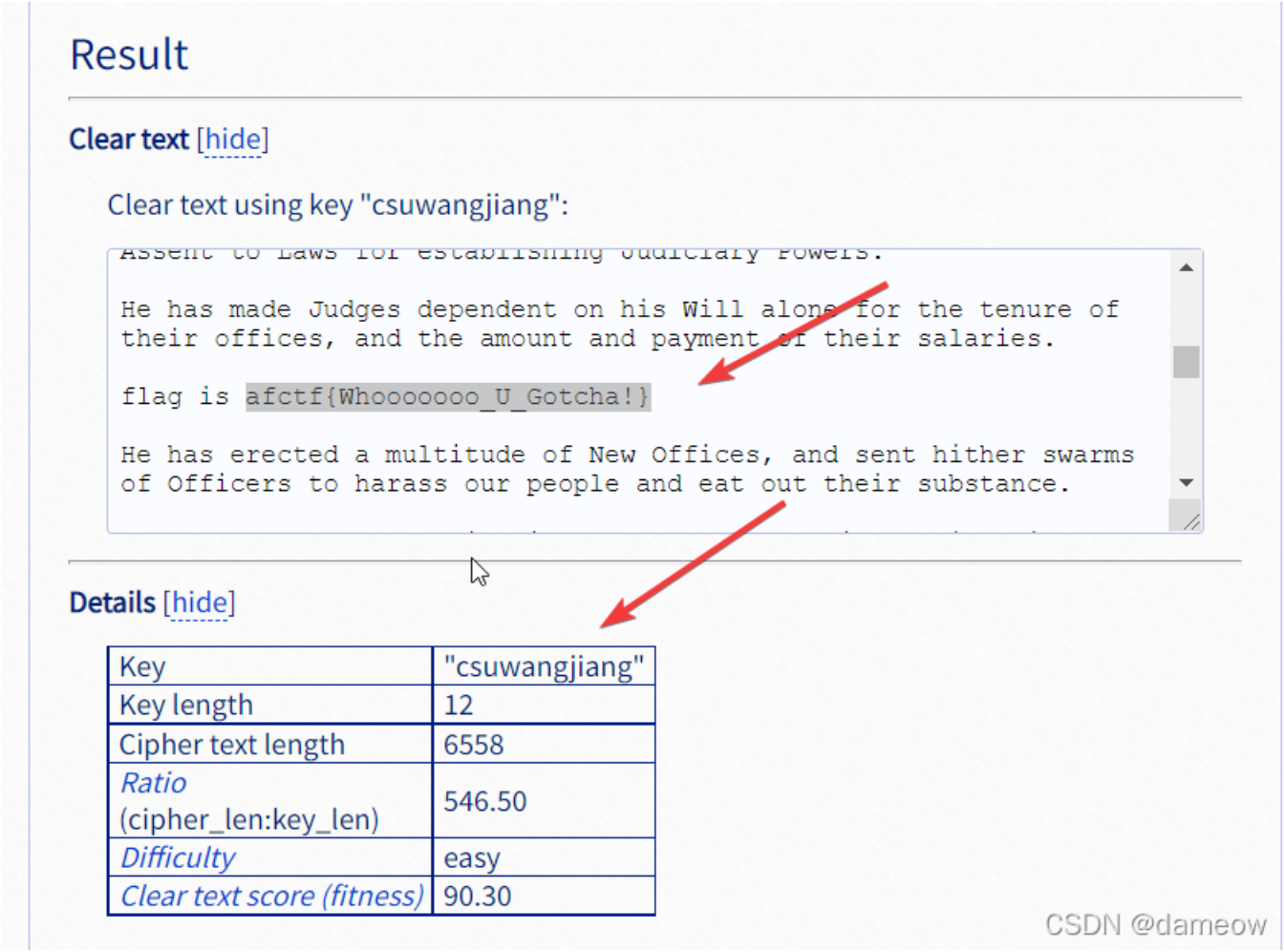
脚本：

```
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']
print(len(ls))
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}
```

2018 AFCTF Vigenère

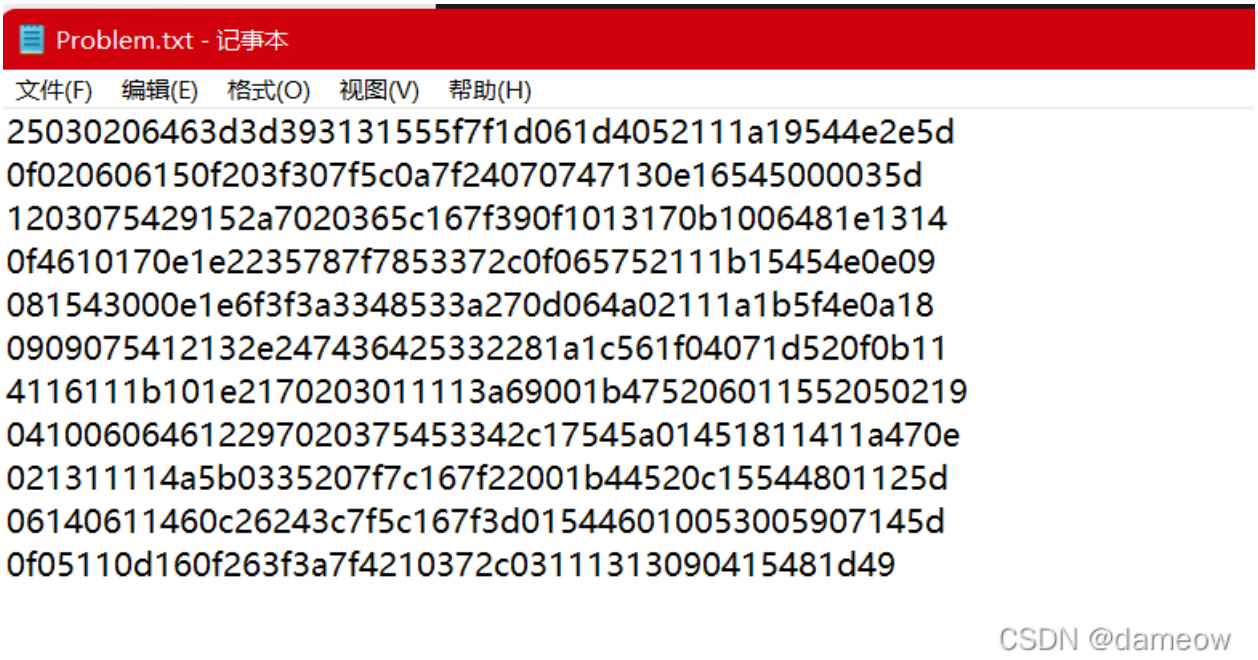
```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

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;
}
```


不用看加密程序，直接爆破，工具：Vigenere Solver - www.guballa.de



afctf{Whooooooo_U_Gotcha!}

2018 AFCTF 你听过一次一密么？



不会做，直接抄脚本：

```
#!/usr/bin/python
## OTP - Recovering the private key from a set of messages that were encrypted w/ the same private key (Many time 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 = '0410060646122970203754533 42c17545a01451811411a470e'
c9 = '021311114a5b0335207f7c167f22001b44520c15544801125d'
c10 = '06140611460c26243c7f5c167f3d0154460100530059071 45d'
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):
# print current_index,ciphertext
counter = collections.Counter()
# for each other ciphertext
for index, ciphertext2 in enumerate(ciphers):
# print index,ciphertext2
if current_index != index: # don't xor a ciphertext with itself
# print ciphertext.decode('hex'),ciphertext2.decode('hex')
for indexOfChar, char in enumerate(strxor(ciphertext.decode('hex'), ciphertext2.decode('hex'))): # Xor the two ciphertexts
# print indexOfChar,char
# If a character in the xored result is a alphanumeric character, it means there was probably a space character 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 index
```

```
counter[indexOfChar] += 1 # Increment the counter at this index
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 was 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 complete 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)
```


微信搜一搜
安全喵