

保存名称

保存

61666374667B317327745F73305F333435797D

16进制转字符

字符转16进制

↕

复制结果

清空输入框

大写

小写

afctf{1s't_s0_345y} CSDN @dameow

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/>

📱 社会主义核心价值观编码器

社会主义核心价值观：富强民主文明和谐自由平等公正法治爱国敬业诚信友善！

HEBTUCTF{ejvovdasfjfvmrfsdemxj} CSDN @dameow

Ook!

这是Ook编码。解密工具：[Brainfuck/Ook! Obfuscation/Encoding \[splitbrain.org\]](https://splitbrain.org/Brainfuck/Ook!%20Obfuscation/Encoding)

All the hard work (like actually understanding how those languages work) was done by Daniel Lorch and his [Brainfuck interpreter in PHP](#)

```
f1ag{1c470f09af4c86b7}
```

Text to Ookl | Text to short Ookl | Ookl to Text
Text to Brainfuck | Brainfuck to Text

CSDN @dameow

栅栏密码

栅栏密码加密解密

```
fa{660cb679d7866ffa1g7d27e041cfd18ed}
```

每组字数

加密

解密

```
flag{76d6207ceb064719cdf7b8d6168fefda}
```

CSDN @dameow

栅栏中的base

```
4C4A5645455232524B3533544B544C4C4A5A5545324D434749564E48553344474A564548495A53524E595C
```

给了16进制，先16进制转字符串，得base32。

保存名称

保存

```
4C4A5645455232524B3533544B544C4C4A5A5545324D434749564E48553344474A564548495A53524E595944323D3D3D
```

16进制转字符

字符转16进制

↕

复制结果

清空输入框

大写

小写

```
LJVEER2RK53TKTLLJZUE2MCGIVNHU3DGVJEHIZSRNYYD2===
```

CSDN @dameow

base32解码得base64。

Base32编码解码

LJVEER2RK53TKTLLJZUE2MCGIVNHU3DGJVEHIZSRNYD2===

编码 解码 清空

ZjBGQWw5MkNhM0FEZzlfMHtfQn0=

CSDN @dameow

base64解码得栅栏密码。

Base64编码转换

ZjBGQWw5MkNhM0FEZzlfMHtfQn0=

清空 加密 解密 解密为UTF-8字节流

f0FA192Ca3ADg9_0{_B}

CSDN @dameow

栅栏密码解密，组数为5，得flag。



栅栏密码加密解密

f0FA192Ca3ADg9_0{_B}

每组字数 加密 解密

flag{0939_F2A_BACD0}

CSDN @dameow

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。



BASE64加密解密

请将要加密或解密的内容复制到以下区域

```
flag{S0_many_Bas3}
```

[BASE64加密](#) [BASE64解密](#)

CSDN @darneow

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

CSDN @dameow

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。



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

CSDN @dameow

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>

```

C:\Users\n0rland3r\Desktop\crc32-爆破\crc32-0.1>python3 crc32.py reverse 0x8E234AE0
4 bytes: {0x0e, 0xf5, 0x08, 0x69}
verification checksum: 0x8e234ae0 (OK)
alternative: 17LE1V (OK)
alternative: 53QD05 (OK)
alternative: 5Cm55e (OK)
alternative: 6cfFvu (OK)
alternative: 918GQb (OK)
alternative: BzTRNZ (OK)
alternative: EcSldq (OK)
alternative: HqVcs7 (OK)
alternative: JpB1Br (OK)
alternative: LuKbrT (OK)
alternative: TRy0cs (OK)
alternative: _5IbAC (OK)
alternative: bugku_ (OK)
alternative: dpn8Ey (OK)
alternative: mz9jRH (OK)
alternative: q5ekFH (OK)
alternative: w018vn (OK)
alternative: y00y00 (OK)
alternative: zRUFdx (OK)

C:\Users\n0rland3r\Desktop\crc32-爆破\crc32-0.1>python3 crc32.py reverse 0x8115A277
4 bytes: {0x76, 0xa2, 0x0b, 0xe1}
verification checksum: 0x8115a277 (OK)
alternative: 1x5NQ8 (OK)
alternative: 902a5H (OK)
alternative: G1qi4N (OK)

```

CSDN @dameow

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 ',
'gsombC',
'mY3L6b',
'newctf',
'p7p1P7',
'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')
```

CSDN @dameow

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。

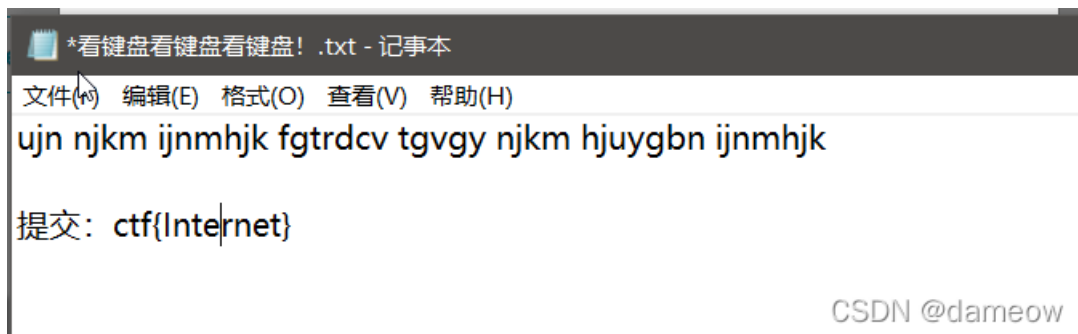
00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	对应文本
FF	D8	FF	E0	00	10	4A	46	49	46	00	01	01	01	00	48	ÿÿà..JFIF.....H
00	48	00	00	FF	DB	00	43	00	0E	0A	0B	0C	0B	09	0E	.H..ÿÿ.C.....
0C	0B	0C	10	0F	0E	11	15	23	17	15	13	13	15	2B	1F#.....+
21	66	6C	61	67	7B	43	72	63	72	63	72	63	72	63	5E	!flag{Crcrcrcrc
33	32	5F	42	4F	4F	4D	7D									32 BOOM}

CSDN @dameow

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。



CSDN @dameow

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。

RSA	
名称	修改日期
flag.enc	2018/4/15 15:53
public.key	2018/4/15 15:40

CSDN @dameow

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(79832181757332818552764610761349592984614744432279135328398999801627880283610900361281249973175805069916210179560
506497075132524902086881120372213626641879468491936860976686933630869673826972619938321951599146744807653301076026577949
579618331502776303983485566046485431039541708467141408260220098592761245010678592347501894176269580510459729633673468068
467144199744563731826362102608811033400887813754780282628099443490170016087838606998017490456601315802448567772411623826
281747245660954245413781519794295336197555688543537992197142258053220453757666537840276416475602759374950715283890232230
741542737319569819793988431443)

=====
fac: factoring 798321817573328185527646107613495929846147444322791353283989998016278802836109003612812499731758050699162
101795605064970751325249020868811203722136266418794684919368609766869336308696738269726199383219515991467448076533010760
265779495796183315027763039834855660464854310395417084671414082602200985927612450106785923475018941762695805104597296336
734680684671441997445637318263621026088110334008878137547802826280994434901700160878386069980174904566013158024485677724
116238262817472456609542454137815197942953361975556885435379921971422580532204537576665378402764164756027593749507152838
90232230741542737319569819793988431443
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 = 254783260649374192922001721363994977190818429145282283164559062116931183219713999360047291348411629741442462714
864396957860365881174246118819559509962196468073788222782856382615820991083394389495730341012151411561564087428438200480
668308638143623798857203950823184628500029016056897618763191511473527300909575569408421442998873946787436077669378280944
783364011594490358783068537162165483742734623865083073677131120730040113834189678949305540675824532489810220119228833744
427368480459206763413618712317871634414675330768900817218821793691687872877247696426653999925560521448458786001262839688
90273067575342061776244939
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。

$$d = \text{gmpy2.invert}(e, (q-1)*(p-1))$$

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 $\text{pow}(c, d, n)$ 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 =
254783260649374192922001721363994977190818429145282283164559062116931183219713999360047291348
< [ ] >

# 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
< [Progress bar] >

[+]N:
691316677109436623113422493782665795857921917893759942123087462879884062720557906429183155859
< [Progress bar] >

'''

```

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 ， $d = \text{gmpy2.invert}(e, \phi(n))$ ，但是欧拉函数 $\phi(n) = (q-1)*(p-1)$ ，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 $c = me \pmod{n}$ ，相当于 $m^3 + k*n = c$ ， 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 1509929362729692740720713536968392715648402189242304424408501797538190273763219595501
b'flag{4c466c3d094911b3a3ca3319b43fe792bef9e94a19c8f666d2ec6c890034d88ba}'

Process finished with exit code 0

```

CSDN @dameow

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 。

按照常规流程，先把 n 分解，得到两个大质数 qp 。

工具：factordb.com

The screenshot shows the factordb.com interface. The search bar contains the number 20499421483319837632829005665244953604816631094131482091599... and the 'Factorize!' button is clicked. The result table shows the number factored into three prime factors.

Result:		
status (2)	digits	number
FF	617 (show)	2049942148...07<617> = 1381495581...17<309> · 1483857187...71<309>

CSDN @dameow

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。


```
(kali@kali) [~/tmp]
└─$ openssl rsa -pubin -text -modulus -in pubkey1.pem
RSA Public-Key: (2048 bit)
Modulus:
 00:89:89:a3:98:98:84:56:b3:fe:f4:a6:ad:86:df:
 3c:99:57:7f:89:78:04:8d:e5:43:6b:ef:c3:0d:8d:
 8c:94:95:89:12:aa:52:6f:f3:33:b6:68:57:30:6e:
 bb:8d:e3:6c:2c:39:6a:84:ef:dc:5d:38:25:02:da:
 a1:a3:f3:b6:e9:75:02:d2:e3:1c:84:93:30:f5:b4:
 c9:52:57:a1:49:a9:7f:59:54:ea:f8:93:41:14:7a:
 dc:dd:4e:95:0f:ff:74:e3:0b:be:62:28:76:b4:2e:
 ea:c8:6d:f4:ad:97:15:d0:5b:56:04:aa:81:79:42:
 4c:7d:9a:c4:6b:d6:b5:f3:22:b2:b5:72:8b:a1:48:
 70:4a:25:a8:ef:cc:1e:7c:84:ea:7e:5c:e3:e0:17:
 03:f0:4f:94:a4:31:d9:95:4b:d7:ae:2c:7d:d6:e8:
 79:b3:5f:8a:2d:4a:5e:fb:e7:37:25:7b:f9:9b:d9:
 ee:66:b1:5a:ff:23:3f:c7:7b:55:8a:48:7d:a5:95:
 2f:be:2b:92:3d:a9:c5:eb:46:78:8c:05:03:36:b7:
 e3:6a:5e:d8:2d:5c:1b:2a:eb:0e:45:be:e4:05:cb:
 e7:24:81:db:25:68:aa:82:9e:ea:c8:7d:20:1a:5a:
 8f:f5:ee:6f:0b:e3:81:92:ab:28:39:63:5f:6c:66:
 42:17
Exponent: 2333 (0x91d)
Modulus=8989A398988456B3FEF4A6AD86DF3C99577F8978048DE5436BEFC30D8D8C94958912AA526FF333B66857306EBB8
DE36C2C396A84EFDCC5D382502DAA1A3F3B6E97502D2E31C849330F5B4C95257A149A97F5954EAF89341147ADCDD4E950FFF
74E30BBE622876B42EEAC86DF4AD9715D05B5604AA8179424C7D9AC46BD6B5F322B2B5728BA148704A25A8EFC1E7C84EA7
E5CE3E01703F04F94A431D9954BD7AE2C7DD6E879B35F8A2D4A5EFBE737257BF99BD9EE66B15AFF233FC77B558A487DA595
2FBE2B923DA9C5EB46788C050336B7E36A5ED82D5C1B2AEB0E45BEE405CBE72481DB2568AA829EEAC87D201A5A8FF5EE6F0
BE38192AB2839635F6C664217
writing RSA key
```

CSDN @dameow

密钥强度2048bit。

e = 2333, n =

1736252012414973605929160571783981408943126183397240817576650489487609127202119737448021

看看pubkey2.pem。

```
(kali@kali) [~/tmp]
└─$ openssl rsa -pubin -text -modulus -in pubkey2.pem
RSA Public-Key: (2048 bit)
Modulus:
 00:89:89:a3:98:98:84:56:b3:fe:f4:a6:ad:86:df:
 3c:99:57:7f:89:78:04:8d:e5:43:6b:ef:c3:0d:8d:
 8c:94:95:89:12:aa:52:6f:f3:33:b6:68:57:30:6e:
 bb:8d:e3:6c:2c:39:6a:84:ef:dc:5d:38:25:02:da:
 a1:a3:f3:b6:e9:75:02:d2:e3:1c:84:93:30:f5:b4:
 c9:52:57:a1:49:a9:7f:59:54:ea:f8:93:41:14:7a:
 dc:dd:4e:95:0f:ff:74:e3:0b:be:62:28:76:b4:2e:
 ea:c8:6d:f4:ad:97:15:d0:5b:56:04:aa:81:79:42:
 4c:7d:9a:c4:6b:d6:b5:f3:22:b2:b5:72:8b:a1:48:
 70:4a:25:a8:ef:cc:1e:7c:84:ea:7e:5c:e3:e0:17:
 03:f0:4f:94:a4:31:d9:95:4b:d7:ae:2c:7d:d6:e8:
 79:b3:5f:8a:2d:4a:5e:fb:e7:37:25:7b:f9:9b:d9:
 ee:66:b1:5a:ff:23:3f:c7:7b:55:8a:48:7d:a5:95:
 2f:be:2b:92:3d:a9:c5:eb:46:78:8c:05:03:36:b7:
 e3:6a:5e:d8:2d:5c:1b:2a:eb:0e:45:be:e4:05:cb:
 e7:24:81:db:25:68:aa:82:9e:ea:c8:7d:20:1a:5a:
 8f:f5:ee:6f:0b:e3:81:92:ab:28:39:63:5f:6c:66:
 42:17
Exponent: 23333 (0x5b25)
Modulus=8989A398988456B3FEF4A6AD86DF3C99577F8978048DE5436BEFC30D8D8C94958912AA526FF333B66857306EBB8DE36C2C396A84EFD
C5D382502DAA1A3F3B6E97502D2E31C849330F5B4C95257A149A97F5954EAF89341147ADCDD4E950FFF74E30BBE622876B42EEAC86DF4AD9715
D05B5604AA8179424C7D9AC46BD6B5F322B2B5728BA148704A25A8EFC1E7C84EA7E5CE3E01703F04F94A431D9954BD7AE2C7DD6E879B35F8A2
D4A5EFBE737257BF99BD9EE66B15AFF233FC77B558A487DA5952FBE2B923DA9C5EB46788C050336B7E36A5ED82D5C1B2AEB0E45BEE405CBE724
81DB2568AA829EEAC87D201A5A8FF5EE6F0BE38192AB2839635F6C664217
writing RSA key
```

CSDN @dameow

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。

```
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)
```

Traceback (most recent call last):
File "D:/Codes/python/d.py", line 6, in <module>
 c1 = libnum.s2n(c1)
File "D:\Applications\python38\lib\site-packages\libnum\strings.py", line 10, in s2n
 return int(s.encode("utf-8").hex(), 16)
AttributeError: 'bytes' object has no attribute 'encode'

CSDN @dameow

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

```
def s2n(s):  
    """  
    String to number.  
    """  
    if not len(s):  
        return 0  
    return int(s.encode("utf-8").hex(), 16)
```

CSDN @dameow

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
1175717716862997466131912906502093925960784385596461240751501561955133271730359493928426514842110
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=
1736252012414973605929160571783981408943126183397240817576650489487609127202119737448021558251
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)
```

```

1  from gmpy2 import invert
2
3  def gongmo(n, c1, c2, e1, e2):
4      def egcd(a, b): # 欧几里德
5          if b == 0:
6              return a, 0
7          else:
8              x, y = egcd(b, a % b)
9              return y, x - (a // b) * y
10
11     s = egcd(e1, e2)
12     s1 = s[0]
13     s2 = s[1]
14
15     # 求模反元素
16     if s1 < 0:
17         s1 = -s1
18         c1 = invert(c1, n)
19     elif s2 < 0:
20         s2 = -s2

```

gongmo() > if s1 < 0

RSA共模攻击 ×

D:\Applications\python38\python3.exe D:/Codes/python/RSA共模攻击.py
0x666c61677b34623062346338612d383266332d346438302d393032622d3865376135373036663866657d
CSDN @dameow

16进制转一下字符串得flag，大坑（横杠要去掉，这也太坑了）

保存名称

666c61677b34623062346338612d383266332d346438302d393032622d3865376135373036663866657d

flag{4b0b4c8a-82f3-4d80-902b-8e7a5706f8fe}

CSDN @dameow

【2021医疗行业CTF】base编码

R1kzRE1RWldHRTNET04yQ0dZWkRNTUpYR00zREtNWldHTTJES1JSVEdNWIRFTktHR01ZVEdOUIZJWtN

一次base6解码，一次base32解码，一次16进制转字符串。

看的出来吗

bE0veldtTDs7NzITe3hzbSFYSj5Sa2U6eyQ4NyVrI3FvWfU6QIs7QIVK

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。

Base58编码

在线base58编码、在线base58解码、base58编码、base58解码、base58check

```
iDMb6ZMTGMptmkhxw36mqkjCkyUHL3sSp4
```

模式

字符集

编码

解码

```
flag{JustUse3TimesEncode}
```

CSDN @dameow

键盘之争

```
cipher.txt - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)
ypau_kjg;"g;"ypau+
CSDN @dameow
```

肯定是改了键盘布局，我们常用的键盘布局是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"}
```

l / . l L ,
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",,

```

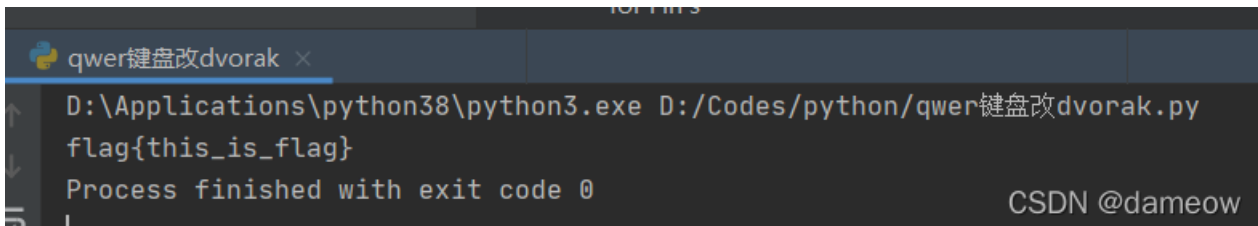
"W":r"<",
"v":r".",
"V":r">",
"z":r"/",
"Z":r"?",
r"!":r"!",
r"@":r"@ ",
r"#":r"#",
r"$":r"$",
r"%":r"% ",
r"^":r"^",
r"&":r"&",
r"*":r"*",
r"(":r"(",
r)":r")",
r"[":r"- ",
r"]":r"=",
r"{":r"_ ",
r"}":r"+"}
s=r'ypau_kjg;"g;"ypau+'

```

```

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_pnr4e_f0_fvzc1r}

移位13的凯撒密码，即rot13。

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。

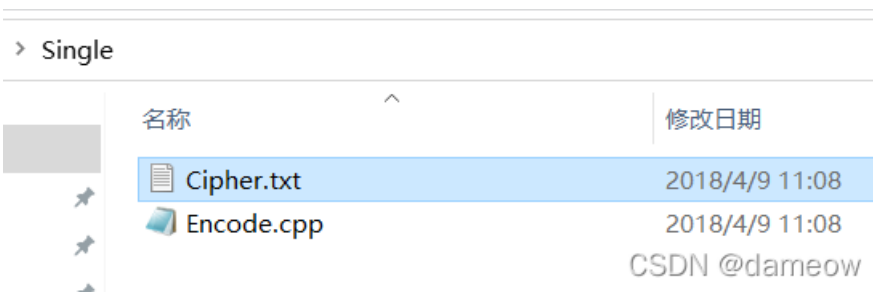
```
please input string to decode:
AABABABABBBAAAAAABBAABAABABBBAAAABBBABABAAAAAABAAAABAABAABAABBBAAAABBBABAAAABAABAAAAAABABBB
first decode method result is:
FLAGISYOUARESOGREAT
second decode method result is:
FMAGIJTPWASETTPGSEAUUV
```

CSDN @dameow

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 jmraxzdeau. Lxd akmoqfa, Wab, Lxdanuej, Jdcqrx, Benmdc xd uxoarvenz afua. Ramo jmn zmen uxoa qxenru lxd atadc uxftay rmus. Oxda qxenru lxd oxda jxoqfejmrax 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 tinfmdmbfa uadtejau. Cxid ramo vmu reoa lxd qmrjvenz cxid uadtejau mny yatafxqenz akqfxeru iuimffc. Ux, rvan xdzmnehadu jxnnejru qmdrejeqmnrux 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 qxuebfax lxdomru. Er omc ba uxoarvenz fesa wmdzmoa werv uqajemf reoa lxd rmus-bmuay afaoanru (a.z. IJUB eJRL).

JRL zmoau xlrax rxiiiv xn omnc xrvad muuairu xl enlxdomrexn uaiiderc: idcarxzdmaxc. uraxx. benmdc


```

#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;
}

```

CSDN @dameow

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

quipqiup **beta3**

quipqiup is a fast and automated cryptogram solver by [Edwin Olson](#). It can solve simple substitution ciphers often found in newspapers, including puzzles like cryptoquips (in which word boundaries are preserved) and patristocrats (inwhi chwor dboun darie saren t).

Puzzle:

```

Oekay jxoqarerexrnu omc tmdc qxuuebfa lxdomru. Er omc ba uxoarvenz fesa wmdzmoa werv uqajenf reoa lxd rmus-bmuay afaoarru (a.z. IJUB eJRL).
JRL zmoau xlrav rxlrv xn omnc xrvad muqajru xl enlxdomrern uajiderc: jdoqrxzdmqvc, urazz, bermdc mmfmcueu, datadua anzanaadenz, oxbefa uajiderc mny xrvadu. Zzxy
ramou zanadmfcc vmta urdxnz useffu mny akqadeanja en mff rvaua euilau.
Iuimffc, ifmz eu uxoa urdenz xl dmyrxo ymrm xd rskr en uxoa lxdomr. Akmoqfa mljrl(Xv_I_lxiny_er_neja_rDc)

```

Clues: For example G=R QVW=THE

2.8

⊗ automatically selected statistics mode; you can override by using the drop down menu next to the solve button.

- 1.702 Capture the Flag (CTF) is a special kind of information security competitions. There are three common types of CTFs: Jeopardy, Attack-Defence and mixed. Jeopardy-style CTFs has a couple of questions (tasks) in range of categories. For example, Web, Forensic, Crypto, Binary or something else. Team can gain some points for every solved task. More points for more complicated tasks usually. The next task in chain can be opened only after some team solve previous task. Then the game time is over sum of points shows you a CTF winner. Famous example of such CTF is Defcon CTF quals. Well, attack-defence is another interesting kind of competitions. Here every team has own network(or only one host) with vulnerable services. Your team has time for patching your services and developing exploits usually. So, then organizers connects participants of competition and the wargame starts! You should protect own services for defence points and hack opponents for attack points. Historically this is a first type of CTFs, everybody knows about DEF CON CTF - something like a World Cup of all other competitions. Mixed competitions may vary possible formats. It may be something like wargame with special time for task-based elements (e.g. UCSE iCTF). CTF games often touch on many other aspects of information security: cryptography, stego, binary analysis, reverse engineering, mobile security and others. **Good teams generally** have strong skills and experience in all these issues. Usually, flag is some string of random data or text in some format. Example **afctf(Oh_U_found_it_nice_rTy)**
- 4.273 Wepkuma kha Ireg (WKT) os e spawer jond it ontimlekoins sawmoky wilpakokoins. Khama ema khmaa willin kypas it WKTs: Faipemdy, Ekkewj-Datarwa end lovad. Faipemdy-skyra WKTs hes e wiupra it quaskoins (kesjs) on menga it wekagimoas. Tim avelpra, Bac, Timansow, Wnykki, Conemy im silakhong arsa. Kael wen geon sila pionks tim azamy sirzad kesj. Lima pionks tim lima wilprowekad kesjs usuerry. Kha navk kesj on wheon wen ca ipanad inry etkam sila kael sirza pmazoius kesj. Khan kha geia koia os izam

CSDN @dameow

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
```

I

CSDN @darneow

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=0x70c9133e1647e95c3cb99bd998a9028b5bf492929725a9e8e6d2e277fa0f37205580b196e5f121a2e83bc80a8204c  
  
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_Jjb0 (OK)
6 bytes: 6FMTHd (OK)
6 bytes: 7ZC9Ri (OK)
6 bytes: 8IRdtj (OK)
6 bytes: DF90Ay (OK)
6 bytes: KIgNfn (OK)
6 bytes: SnUcwI (OK)
6 bytes: XDHs8q (OK)
6 bytes: ZxMma1 (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)
```

CSDN @dameow

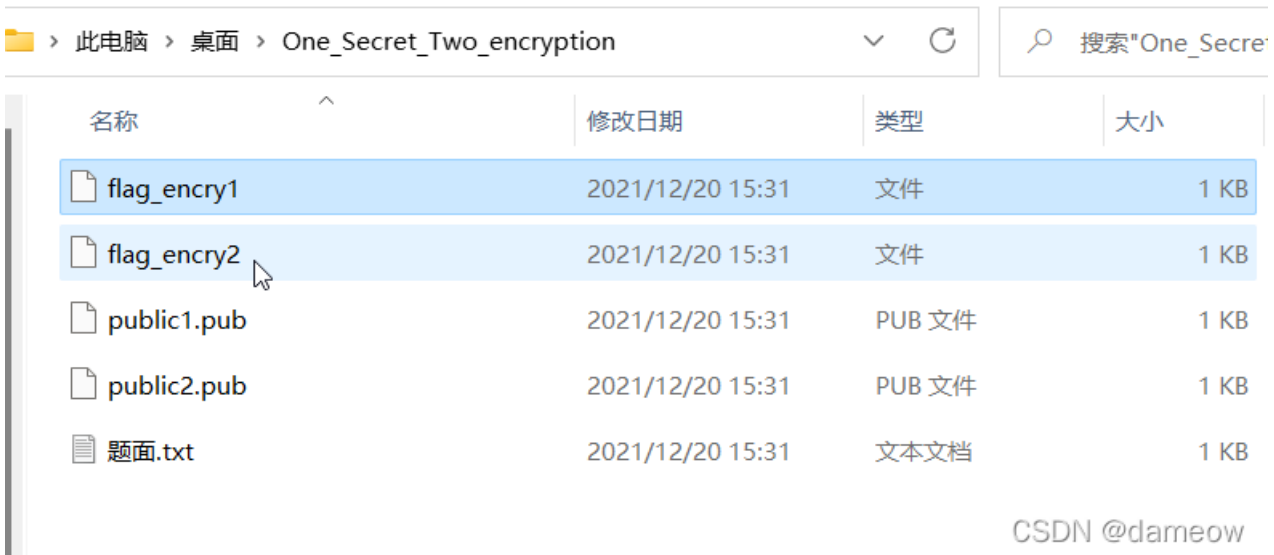
第一个解密得b323，这个脚本直接出字符串，不能额外16进制转字符串，比较好用。

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。



CSDN @dameow

```
题面.txt - 记事本
```

```
文件(F) 编辑(E) 格式(O) 视图(V) 帮助(H)
```

一份秘密发送给两个人不太好吧，那我各自加密一次好啦~~~

素数生成好慢呀

偷个懒也.....不会有问题的吧?

CSDN @dameow

直接对公钥进行提取e,n。

```
-----BEGIN PUBLIC KEY-----
MIICIDANBgkqhkiG9w0BAQFAAQAgOAMIICCAKCAQAma/gXML+bivU20mJu55PZ
SjNAE6SOPQ2WV5sYIA7ZLbJ6lshW8cf0hErNOTUIv+60+hXSMFd4wrv27+f6akPE
qeNL6LWjKqcnC9I03vbyYDZuLkfeOPwM9UHIuRUFU/L/DOCKjk0kHN5SMufg66y
0Ge4wLDi9f8sET4QMerAVF/HZ7acpYYCu8QoWn0Sy9KiVzKQmZKkaL+WcN2sblsA
61zjixv7ybMHDmcyMKHb5VbFpsqMW19roVLV51uY3S1rhTogmyGg19Q3k7hYW3ca
Jc7WLEbPD/OnlHMDLArNUYMyB9t0CdLNZZCHE6pbiMaNGS+rwGcqxHbWcPGCS0vZ
AoIBAAOzxRVobON2SXRSPv7GRHMzj/mpACiz3VrKtwKgm+MM1UrnvtCpPcqjqw6
4w7HJqf60/ccA710qY35jvvhOLD2II6rf8USZwkm27ygOuHA3SRS9XZypshUmRo
YyJtogGntK4R9R7wFw1HHJbIj5R3/HV/gYdT9XxKq7yICxFSVv3u1gVBjNeUzTn
5Mz01X9tCEjcs54eq90CGLEIetj24xTfsruKf/DDGNRA1et8fU4H49NMsqqHL1R
ZdKvy1SEBWo0rj/XQyez0Dsyre0jcaDwnayw1uiHGPrKXyC6ePAJ9S7IMOTKb1CF
FU9nj28KAsm/RBAV3PEMk3edPM=
-----END PUBLIC KEY-----
```

解析

详细信息

密钥类型	RSA
密钥强度	2048
PN(e)	1666626632960368239001159408047765991270250042206244157447171188195657302933019501932101777999510001235736338E
PN(n)	4850297138162223468826481623082440249579136876798312652735204698 6896139690086325452209766991703084540823908347425707182478042020 6092949357164207467942856516840587711068151810566730178565351769 7684490982375078989886040451115082120928982588380914609273008153 9779079505324986054862258839736431415160240583153605729887446071 3411025448942151602693724916349398268133662872603348912470565721 7768229058487155865265080427488028921879608338898933540825564889 0121661813461772766398283463763621689342088224672956737618769658 6457316452933688525057735776731425658101947413065141210089783960 6491189424373959244023695669653213498329

CSDN @dameow

两个公钥得到如下结果：

e1 =
1666626632960368239001159408047765991270250042206244157447171188195657302933019501932101

n1 =
4850297138162223468826481623082440249579136876798312652735204698689613969008632545220976

e2 = 65537

n2 =
2367536768672000959668181171787295271898789288397672997134843418932405959946739637368044

意外解：直接对n1或者n2进行因数分解，可以直接分解出pq。

Search Sequences Report results Factor tables Status Download

485029713816222346882648162308244024957913687679831265273520469868961396900863254522097669917C Factorize!

Result:

status (2)	digits	number
FF	616 (show)	4850297138...29 <616> = 2780972247...43 <308> · 1744101237...03 <309>

More information

ECM

CSDN @dameow

得到：

q1 =
2780972247225275648823657238494934989120864309011734950999441704798974648457613039220678

p1 =
1744101237616313375207991798085981279141849719788117967224142152398741140483478306092558

解密脚本:

```
e1 =  
1666626632960368239001159408047765991270250042206244157447171188195657302933019501932101777991  
n1 =  
4850297138162223468826481623082440249579136876798312652735204698689613969008632545220976699171  
p1 =  
2780972247225275648823657238494934989120864309011734950999441704798974648457613039220678187574  
q1 =  
174410123761631337520799179808598127914184971978811796722414215239874114048347830609255805203  
  
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))
```

```
test123 x  
D:\Applications\Python37\python3.exe D:/Codes/python/test123.py  
b'openssl is widely used\r\nflag is afctf{You_Know_0p3u55I}'  
  
Process finished with exit code 0  
CSDN @dameow
```

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。

```
1 n2 = 236753676867200095966818117178729527189878928839767299713484341893240595994673963
2 n1 = 485029713816222346882648162308244024957913687679831265273520469868961396900863254
3
4 import gmpy2
5 print(gmpy2.gcd(n1,n2))
6
7
8
```

```
test123 x
D:\Applications\Python37\python3.exe D:/Codes/python/test123.py
1744101237616313375207991798085981279141849719788117967224142152398741140483478306092558
Process finished with exit code 0 CSDN @dameow
```

得到公约数

1744101237616313375207991798085981279141849719788117967224142152398741140483478306092558

由于 $n = q \cdot p$ ，假设公约数是 q ，则 $p = n$ 除以 q 。

```
1 n2 = 2367536768672000959668181171787295271898789288397672997134843418932405959946739637368044420319861797856771490573443003520
2 n1 = 4850297138162223468826481623082440249579136876798312652735204698689613969008632545220976699170308454082390834742570718247
3
4 import gmpy2
5 q = gmpy2.gcd(n1,n2)
6 p = n1/q
7 print("{:.0f}".format(p))_# 为了不以科学计数法输出，
8
9
10
```

```
test123 x
pplications\Python37\python3.exe D:/Codes/python/test123.py
97224722527550201037496077148307955488717729176656302963273628450659298669450412040367485653693110465736340970173107538480795260
Process finished with exit code 0 CSDN @dameow
```

得到 e, n, q, p ，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 =
(7639292249012390500776473424251443633180062193602104651134586229378821197667054582737889
2775041668183790046863142587579780448282786422609917541471782589482330329915927720297407
#enc =
< [REDACTED] >
7579294603035135817234501131256282324240132424272000903035801073847222917306092792610406
< [REDACTED] >

```

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] =
7639292249012390500776473424251443633180062193602104651134586229378821197667054582737889
< [REDACTED] >

pubkey[1] =
2775041668183790046863142587579780448282786422609917541471782589482330329915927720297407
< [REDACTED] >

```

看了wp，实在是明白为什么脚本中pubkey[1]可以充当模数n。

解密脚本：


```

pub0 =
763929224901239050077647342425144363318006219360210465113458622937882119766705458273788947718911994090187932
947694326848868575500906975653763043489419853300731695950407389203582333794360159494405111004208058198680365
172060235321945875374635278292661761319773173838900295933943437761251440819434675631450032782188481758975272
071015244168751016874668742536151359822585793537758876078350987062972237271763834743266722655055439868971805
843593929839097963319788083763
pub1 =
277504166818379004686314258757978044828278642260991754147178258948233032991592772029740709036302768682487556
42608884903122768656427165401563920443482151217

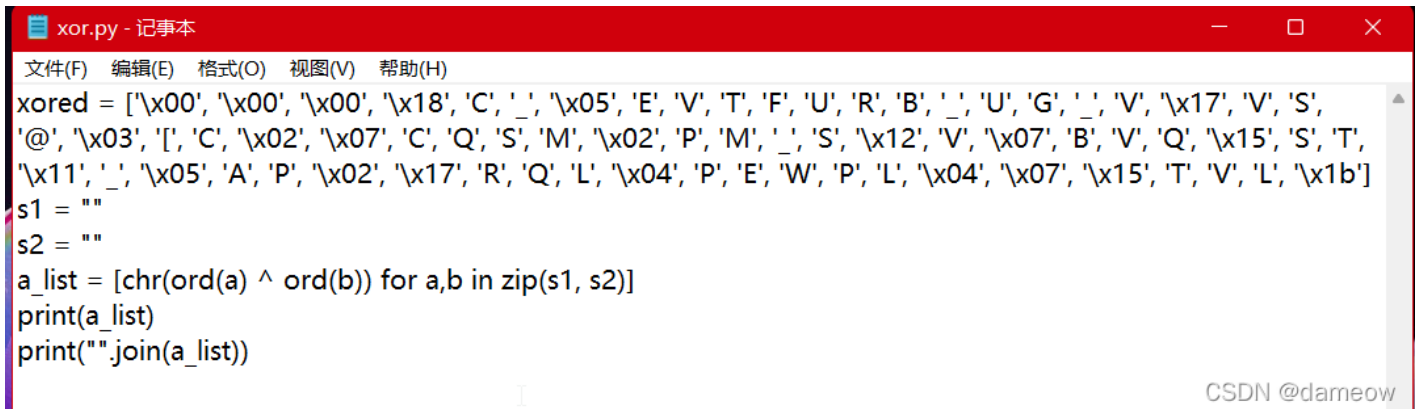
c =
757929460303513581723450113125628232424013242427200090303580107384722291730609279261040687645169812195623204
703163944876732347508817035786365343509674664060932552503532932862656214304965612424126357664997482053380016
143266694632722641803682614616028827317552186468712272083679559663506776114651994044181776573216038201843446
577674937220918121281724718747468524615619338182204477274665469012674671572816652727493363475044347868281266
402200944434844895572022657946878192777613333604552514920338413785986815307782070816714605242901479066092158
6958660759949399826568136755816563468739111123761288318276379025018708883521

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



```

xor.py - 记事本
文件(F) 编辑(E) 格式(O) 视图(V) 帮助(H)
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 = ""
a_list = [chr(ord(a) ^ ord(b)) for a,b in zip(s1, s2)]
print(a_list)
print("".join(a_list))

```

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;
}

```

CSDN @dameow

不用看加密程序，直接爆破，工具：[Vigenere Solver - www.guballa.de](http://www.guballa.de)

Result

Clear text [\[hide\]](#)

Clear text using key "csuwangjiang":

```
Assent to Laws for establishing Judiciary Powers.  
  
He has made Judges dependent on his Will alone for the tenure of  
their offices, and the amount and payment of their salaries.  
  
flag is afctf{Whooooooo_U_Gotcha!}  
  
He has erected a multitude of New Offices, and sent hither swarms  
of Officers to harass our people and eat out their substance.
```

Details [\[hide\]](#)

Key	"csuwangjiang"
Key length	12
Cipher text length	6558
<i>Ratio</i> (cipher_len:key_len)	546.50
<i>Difficulty</i>	easy
<i>Clear text score (fitness)</i>	90.30

CSDN @dameow

```
afctf{Whooooooo_U_Gotcha!}
```

2018 AFCTF 你听过一次一密么？

Problem.txt - 记事本

文件(F) 编辑(E) 格式(O) 视图(V) 帮助(H)

```
25030206463d3d393131555f7f1d061d4052111a19544e2e5d  
0f020606150f203f307f5c0a7f24070747130e16545000035d  
1203075429152a7020365c167f390f1013170b1006481e1314  
0f4610170e1e2235787f7853372c0f065752111b15454e0e09  
081543000e1e6f3f3a3348533a270d064a02111a1b5f4e0a18  
0909075412132e247436425332281a1c561f04071d520f0b11  
4116111b101e2170203011113a69001b475206011552050219  
041006064612297020375453342c17545a01451811411a470e  
021311114a5b0335207f7c167f22001b44520c15544801125d  
06140611460c26243c7f5c167f3d015446010053005907145d  
0f05110d160f263f3a7f4210372c03111313090415481d49
```

CSDN @dameow

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 encrypted with the same key
```

```
c1 = '25030206463d3d393131555f7f1d061d4052111a19544e2e5d'
```

```
c2 = '0f020606150f203f307f5c0a7f24070747130e16545000035d'
```

```
c3 = '1203075429152a7020365c167f390f1013170b1006481e1314'
```

```
c4 = '0f4610170e1e2235787f7853372c0f065752111b15454e0e09'
```

```
c5 = '081543000e1e6f3f3a3348533a270d064a02111a1b5f4e0a18'
```

```
c6 = '0909075412132e247436425332281a1c561f04071d520f0b11'
```

```
c7 = '4116111b101e2170203011113a69001b475206011552050219'
```

```
c8 = '041006064612297020375453342c17545a01451811411a470e'
```

```
c9 = '021311114a5b0335207f7c167f22001b44520c15544801125d'
```

```
c10 = '06140611460c26243c7f5c167f3d015446010053005907145d'
```

```
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 know 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

©2019 安全喵