CTF密码学总结（一）

原创

沐一 · 林 于 2021-11-01 21:23:03 发布 1694 收藏 19
分类专栏： 笔记 文章标签： ctf
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/xiao__1bai/article/details/121089114
版权


笔记 专栏收录该内容
18 篇文章 5 订阅
订阅专栏

目录

CTF 密码学总结

出人意料的flag：

指在题目中获取到了flag，但是这个flag可能长得不像flag，或者flag还要经过进一步的脑洞处理，而不是常规的解密处理。

题目类型总结：

题目描述暗示：

指题目给出的描述中有解题的大方向思路，以及对解题过程中出现的一些疑惑点的解释。

简单密码类型：

摩斯密码：

指解题中的密文涉及摩斯密码，摩斯密码的特征是以.-或01组成的，分隔符有空格或斜杠/。

url编码：

指解题中的密文涉及url编码，url编码的特征是使用 "%" 其后跟随两位的十六进制数来替换非 ASCII 字符。

传统base64解密：

指题目中密文是涉及base64加密，密文通常是4的倍数，基本元素是ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/和补充的'='

unicode解密：

指解题中密文涉及 unicode 解密，unicode现在已知的特征是ASCII（英文也是ASCII码）转unicode码时是 &# 前缀 +2 个 16 进制数，并用 ; 分隔。中文转unicode时是 \u + 4 个 16 进制数，中间没有间隔。

举例：1234 ----->1234 牛逼-------->\u725b\u903c

ASCII解密：

指解题中密文涉及ASCII码，需要转成字符。ASCII共128个可显示字符，范围是0~127或0~7F。不同通常每个ASCII码的间隔依据出题者来定。

栅栏密码：

指解题中密文涉及栅栏密码，因为栅栏密码有传统的矩阵型和 W 型，所以需要自己辨认。根据题目给的 key 分段来逐个尝试。

转轮机加密：

指解密中密文涉及转轮机加密，转轮机密文的特点是等长的分好组的乱序字母，原理是转齿轮把一个字母换成另一个来拼成一句话，所以会有多组密钥，但是只有一组密文。

格式举例，等长的分好组的字符串：

ZWAXJGDLUBVIQHKYPNTCRMOSFE

KPBELNACZDTRXMJQOYHGVSFUWI

复杂密码类型：

ECC加密：

是一种建立公开密钥加密的算法，基于椭圆曲线数学的椭圆曲线密码学。

密码学脚本类总结：

ASCII转字符脚本：

指解题中遇到长串ASCII码形式，需要转字符，但是一个个转太麻烦，又没有在线的长串ASCII码转字符网站。且给出的长串ASCII码的间隔依情况而定，如：/119/101/，这时需要自己根据对应间隔写出批量转换脚本。

源代码修改逻辑解密：

指解题中在有较完整源代码的情况下代码逻辑比较明朗，且可以逆向。这时需要充分利用有源代码的优势来在源代码中修改处逆向逻辑，不要自己从头到尾另写一份。

单独的密文类型（优先使用cipher工具）

多层传统加密混合：

Bugku的密码学的入门题/.-：（摩斯密码、url编码、出人意料的flag）

摩斯密码是以.-或01组成的，分隔符有空格或斜杠/，所以直接扔去摩斯密码在线解密即可。

网址：https://www.bejson.com/enc/morse/

答案差不多了，却被%u7b和%u7d卡住了，猜想是{ }的url编码，{的url编码是%7B，}的url编码是%7D

所以去掉u符号（这里u符号应该是什么十六进制之类的，我也不知道）

直接得到flag：

FLAG{D3FCBF17F9399504}

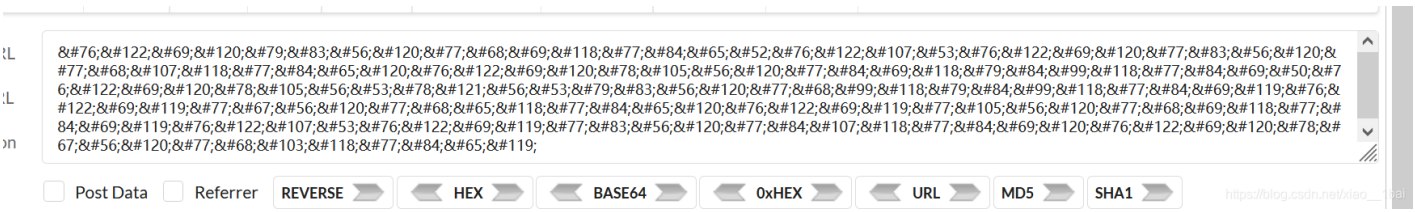
结果全部要小写：

flag{d3fcbf17f9399504}

攻防世界之混合编码：（base64解密、unicode解密、ASCII转字符脚本、传统base64解密、ASCII解密）

下载附件，打开，发现两个等号和19azA~Z的典型base64编码型，直接base64转换：

转了一堆&#出来，根据以前的做题经验，猜unicode或hex：



一开始用了十六进制来转，转了个四不像出来，后来发现转错了，unicode在线解码网址：

http://www.jsons.cn/unicode/

这三个数的看着像ASCII，因为题目暗示混合编码，直接转换看看：ASCCII表对照法：

0110 1101	0155	109	0x6D	m	小写字母m
0110 1110	0156	110	0x6E	n	小写字母n
0110 1111	0157	111	0x6F	o	小写字母o
0111 0000	0160	112	0x70	p	小写字母p
0111 0001	0161	113	0x71	q	小写字母q
0111 0010	0162	114	0x72	r	小写字母r
0111 0011	0163	115	0x73	s	小写字母s
0111 0100	0164	116	0x74	t	小写字母t
0111 0101	0165	117	0x75	u	小写字母u
0111 0110	0166	118	0x76	v	小写字母v
0111 0111	0167	119	0x77	w	小写字母w
0111 1000	0170	120	0x78	x	小写字母x
0111 1001	0171	121	0x79	y	小写字母y
0111 1010	0172	122	0x7A	z	小写字母z
0111 1011	0173	123	0x7B	{	开花括号
0111 1100	0174	124	0x7C	|	垂线
0111 1101	0175	125	0x7D	}	闭花括号

https://blog.csdn.net/xiao__1bai

ASCII转字符脚本法：（这里因为string.split切片后返回的是列表，所以可以直接用索引获取。）

```
import re
r="/119/101/108/99/111/109/101/116/111/97/116/116/97/99/107/97/110/100/100/101/102/101/110/99/101/119/
r=re.split("/",r)
#print(r)
flag=""
for i in range(1,len(r)):
flag+=chr(int(r[i]))
print(flag)
```

单层传统加密：

Bugku crypto之聪明的小羊：（题目描述暗示、栅栏密码）

好的，传统栅栏密码，下面是我以前的笔记：

所谓栅栏密码，就是把要加密的明文分成N个一组，然后把每组的第1个字连起来，形成一段无规律的话。 不过栅栏密码本身有一个潜规则，就是组成栅栏的字母一般不会太多。（一般不超过30个，也就是一、两句话）

传统栅栏密码(矩阵行列，密钥是行数)：

假如有一个字符串：123456789

取字符串长度的因数进行分组，假如key=3

1 2 3 \\分组情况，每三个数字一组，分为三组

4 5 6

7 8 9

然后每一组依次取一个数字组成一个新字符串：147258369 \\加密完成的字符串

解题：

试一般的栅栏密码,取5为矩阵行数,得到" cyperrocaegireeol} eahfocec gnbip不正确，取5为矩阵列数,得到" cebgccfe en eohplprgecrayoii aoreg”,也不正确，除了常规的栅栏密码,还有

由题目描述可知分两组：

fa{fe13f590

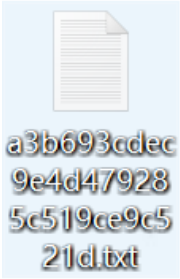
lg6d46d0d0}

那么答案很明显了，上一个下一个即可得flag：

flag{6fde4163df05d900}

攻防世界之转轮机加密：（转轮机加密、）

下载附件：



好了，记住了，以后这个内容格式的就是转轮机加密了：

```
1:  < ZWAXJGDLUBVIQHKYPNTCRMOSFE <
2:  < KPBELNACZDTRXMJQOYHGVSFUWI <
3:  < BDMAIZVRNSJUWFHTEQGYXPLOCK <
4:  < RPLNDVHGFCUKTEBSXQYIZMJWAO <
5:  < IHFRLABEUOTSGJVDKCPMNZQWXY <
6:  < AMKGHIWPNYCJBFZDRUSLOQXVET <
7:  < GWTHSPYBXIZULVKMRAFDCEONJQ <
8:  < NOZUTWDCVRJLXKISEFAPMYGHBQ <
9:  < XPLTDSRFHENYVUBMCQWAOIKZGJ <
10: < UDNAJFBOWTGVRSCZQKELMXYIHP <
11:  < MNBVCXZQWERTPOIUYALSKDJFHG <
12:  < LVNCMXZPQOWEIURYTASBKJDFHG <
13:  < JZQAWSXCDERFVBGTYHNUMKILOP <

密钥为：2,3,7,5,13,12,9,1,8,10,4,11,6
密文为：NFQKSEVOQOFNP
```

转轮机加密格式

https://blog.csdn.net/xiao_...baai

原理就是转齿轮把一个字母换成另一个，直接上一个修改后的大佬脚本：

```
rotor = [ #这里是要输入的转轮机原始字符串

"ZWAXJGDLUBVIQHKYPNTCRMOSFE", "KPBELNACZDTRXMJQOYHGVSFUWI",

"BDMAIZVRNSJUWFHTEQGYXPLOCK", "RPLNDVHGFCUKTEBSXQYIZMJWAO",

"IHFRLABEUOTSGJVDKCPMNZQWXY", "AMKGHIWPNYCJBFZDRUSLOQXVET",

"GWTHSPYBXIZULVKMRAFDCEONJQ", "NOZUTWDCVRJLXKISEFAPMYGHBQ",

"XPLTDSRFHENYVUBMCQWAOIKZGJ", "UDNAJFBOWTGVRSCZQKELMXYIHP",

"MNBVCXZQWERTPOIUYALSKDJFHG", "LVNCMXZPQOWEIURYTASBKJDFHG",

"JZQAWSXCDERFVBGTYHNUMKILOP"

]

cipher = "NFQKSEVOQOFNP" #这是要输入转轮机密文

key = [2,3,7,5,13,12,9,1,8,10,4,11,6] #这是要输入转轮机密钥

tmp_list=[]

for i in range(0, len(rotor)):
```

```
tmp=""
k = key[i] - 1
for j in range(0, len(rotor[k])):
if cipher[i] == rotor[k][j]:
if j == 0:
tmp=rotor[k]
break
else:
tmp=rotor[k][j:] + rotor[k][0:j]
break
tmp_list.append(tmp)
# print(tmp_list)
message_list = []
for i in range(0, len(tmp_list[i])):
tmp = ""
for j in range(0, len(tmp_list)):
tmp += tmp_list[j][i]
message_list.append(tmp)
print(message_list)
def spread_list(lst):
for item in lst:
if isinstance(item,(list,tuple)):
yield from spread_list(item)
else:
yield item
pass
if __name__ == '__main__':
for i in spread_list(message_list):
print("*"*25)
print(i) #在多个输出中查找有语义的字符串即为flag内容
```

复杂加密类型

ECC椭圆曲线加密：

攻防世界 之easy_ECC：（ECC加密）

下载附件，打开：

已知椭圆曲线加密Ep(a,b)参数为

p = 15424654874903

a = 16546484

b = 4548674875

G(6478678675,5636379357093)

私钥为

k = 546768

求公钥K(x,y)

https://blog.csdn.net/xiao__1bai

椭圆曲线ECC加密，没接触过，不懂原理，主要是找不到集成脚本，算了，浏览中发现一篇讲得透彻的博客：

https://blog.csdn.net/weixin_30951231/article/details/95919343

这里直接使用脚本：

```
#!/usr/bin/env python3
# -*- coding: UTF-8 -*-
# @Time :2020/9/28
# @Author :PeterJoin
import collections
import random
EllipticCurve = collections.namedtuple('EllipticCurve', 'name p a b g n h')
def banner():
print(("""%s

_____ ____ ____

| ____/ ___/ ___|

| _|| | | |
```

```
| |__| |__| |___
|_____\____\____|
%s%s
# Coded By PeterJoin -椭圆曲线加密（´・ω・）%s
""" % ('\033[91m', '\033[0m', '\033[93m', '\033[0m')))
curve = EllipticCurve(
'secp256k1',
# Field characteristic.
p=int(input('p=')),
# Curve coefficients.
a=int(input('a=')),
b=int(input('b=')),
# Base point.
g=(int(input('Gx=')),
int(input('Gy='))),
# Subgroup order.
n=int(input('k=')),
# Subgroup cofactor.
h=1,
)
# Modular arithmetic ########################################################
def inverse_mod(k, p):
"""Returns the inverse of k modulo p.
This function returns the only integer x such that (x * k) % p == 1.
k must be non-zero and p must be a prime.
"""
if k == 0:
raise ZeroDivisionError('division by zero')
if k < 0:
# k ** -1 = p - (-k) ** -1 (mod p)
return p - inverse_mod(-k, p)
```

```
# Extended Euclidean algorithm.
s, old_s = 0, 1
t, old_t = 1, 0
r, old_r = p, k
while r != 0:
quotient = old_r // r
old_r, r = r, old_r - quotient * r
old_s, s = s, old_s - quotient * s
old_t, t = t, old_t - quotient * t
gcd, x, y = old_r, old_s, old_t
assert gcd == 1
assert (k * x) % p == 1
return x % p
# Functions that work on curve points #############################################
def is_on_curve(point):
"""Returns True if the given point lies on the elliptic curve."""
if point is None:
# None represents the point at infinity.
return True
x, y = point
return (y * y - x * x * x - curve.a * x - curve.b) % curve.p == 0
def point_neg(point):
"""Returns -point."""
assert is_on_curve(point)
if point is None:
# -0 = 0
return None
x, y = point
result = (x, -y % curve.p)
assert is_on_curve(result)
return result
```

```
def point_add(point1, point2):
"""Returns the result of point1 + point2 according to the group law."""
assert is_on_curve(point1)
assert is_on_curve(point2)
if point1 is None:
# 0 + point2 = point2
return point2
if point2 is None:
# point1 + 0 = point1
return point1
x1, y1 = point1
x2, y2 = point2
if x1 == x2 and y1 != y2:
# point1 + (-point1) = 0
return None
if x1 == x2:
# This is the case point1 == point2.
m = (3 * x1 * x1 + curve.a) * inverse_mod(2 * y1, curve.p)
else:
# This is the case point1 != point2.
m = (y1 - y2) * inverse_mod(x1 - x2, curve.p)
x3 = m * m - x1 - x2
y3 = y1 + m * (x3 - x1)
result = (x3 % curve.p,
-y3 % curve.p)
assert is_on_curve(result)
return result
def scalar_mult(k, point):
"""Returns k * point computed using the double and point_add algorithm."""
assert is_on_curve(point)
if k < 0:
```

```
# k * point = -k * (-point)
return scalar_mult(-k, point_neg(point))
result = None
addend = point
while k:
if k & 1:
# Add.
result = point_add(result, addend)
# Double.
addend = point_add(addend, addend)
k >>= 1
assert is_on_curve(result)
return result
# Keypair generation and ECDHE ##################################################
def make_keypair():
"""Generates a random private-public key pair."""
private_key = curve.n
public_key = scalar_mult(private_key, curve.g)
return private_key, public_key
private_key, public_key = make_keypair()
print("private key:", hex(private_key))
print("public key: (0x{:x}, 0x{:x})".format(*public_key))
if __name__ == '__main__':
banner()
make_keypair()
```

运行之后照着输入题目附件给的参数即可：

解出的x+y就是flag了，原理太难了，以后有机会再接触。

python类型逻辑加密

pyc文件反编译：

攻防世界之easychallenge：（源代码修改逻辑解密、）

下载附件，是pyc文件，于是要反编译，一开始看资料说用uncompyle6，我也不知道为什么我的老是报错，后来又找了个在线反编译，可以支持的Python版本比较多：

https://tool.lu/pyc/

反编译代码：

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

# visit https://tool.lu/pyc/ for more information

import base64

def encode1(ans):

s = ""

for i in ans:

x = ord(i) ^ 36

x = x + 25

s += chr(x)

return s

def encode2(ans):

s = ""

for i in ans:

x = ord(i) + 36

x = x ^ 36

s += chr(x)

return s

def encode3(ans):
```

return base64.b32encode(ans)

flag = " "

print "Please Input your flag:"

flag = raw_input()

final = "UC7KOWVXWVNKNIC2XCXKHKK2W5NLBKNOUOSK3LNNVWW3E==="

if encode3(encode2(encode1(flag))) == final:

print "correct"

else:

print "wrong"

本来是自己一层层改的，怎么加密就怎么逆过来解密，后来发现又犯了以前的错误，有源码就要用源码啊！！！

直接在源码上修改即可：

#!/usr/bin/env python

visit https://tool.lu/pyc/ for more information

import base64

def decode1(ans):

s = ''

for i in ans:

x = ord(i) -25

x = x ^ 36

s += chr(x)

return s

def decode2(ans):

s = ''

for i in ans:

x = i ^ 36

x = x - 36

s += chr(x)

return s

def decode3(ans):

return base64.b32decode(ans)

final = 'UC7KOWVXWVNKNIC2XCXKHKK2W5NLBKNOUOSK3LNNVWW3E==='

print(decode1(decode2(decode3(final))))

传统密码类型解析

凯撒密码(24个字母):

特点:24个字母间的移动

在密码学中，恺撒密码（英语：Caesar cipher），或称恺撒加密、恺撒变换、变换加密，是一种最简单且最广为人知的加密技术。它是一种替换加密的技术，明文中的所有字母都在字母表上向后（或向前）按照一个固定数目进行偏移后被替换成密文。例如，当偏移量是3的时候，所有的字母A将被替换成D，B变成E，以此类推。

加密就是明文向后移动展现出对应的密文。

解密就是密文向前移动展现出对应的明文。

明文字母表	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
密文字母表	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

摩斯密码(只有01(无规则)或.-，空格或/做分隔符):

摩尔斯电码也被称作摩斯密码，是一种时通时断的信号代码，通过不同的排列顺序来表达不同的英文字母、数字和标点符号。它发明于1837年，是一种早期的数字化通信形式。不同于现代化的数字通讯，摩尔斯电码只使用零和一两种状态的二进制代码，它的代码包括五种：短促的点信号“·”，读“滴”（Di）保持一定时间的长信号“-”，读“嗒”（Da）表示点和划之间的停顿、每个词之间中等的停顿，以及句子之间长的停顿。

摩斯电码：（格式要求：可用空格或单斜杠/来分隔摩斯电码，但只可用一种，不可混用）

摩尔斯电码表

字符	电码符号	字符	电码符号	字符	电码符号
A	•—	N	—•	1	•————
B	—•••	O	———	2	••———
C	—•—•	P	•——•	3	•••——
D	—••	Q	——•—	4	••••—
E	•	R	•—•	5	•••••
F	••—•	S	•••	6	—••••
G	——•	T	—	7	——•••
H	••••	U	••—	8	———••
I	••	V	•••—	9	————•
J	•———	W	•——	0	—————
K	—•—	X	—••—	?	••——••
L	•—••	Y	—•——	/	—••—•
M	——	Z	——••	( )	—•——•—
				—	—••••—
				•	•—•—•—

云影密码(01248):

此密码运用了1248代码,因为本人才疏学法,问未发现有过使用的先例,因此暂归为原创密码。由于这个密码,我和片风云影初识,为了纪念,将其命名为“云影密码”。

有了1,2,4,8这四个简单的数字,你可以以加法表示出0~9任何个数字,例如0=28,7=124,9=18这样,再用1-26来表示A-Z,就可以用作密码了。为了不至于混乱,我个人引入了第五个数字0,来用作间隔,以遍免翻译错误,所以还可以称“01248密码”

注意(3个及以上数字时):

虽然是相加，但是可以在数字内不按顺序相加，如124可写成(12)4和1(24)结果分别是7和16，只要保证不大于26即可

题目(总不超过26)：

124010118011802120114018O4

第一步分割：

即124 、1、118、118、212、114、18、4

第二步基本翻译：

例如124可以表示7,也可以表示16(但不可能是34,因为不会超过26),所以可以放弃来翻译其他没有异议的,可得:124、a、s、s、w、o、18、d

第三步推测得出明文：

可以推测后面的18表示r,前面的为p最合适。

所以最后明文:

password(密码)

栅栏密码(分组数作密钥)：

所谓栅栏密码，就是把要加密的明文分成N个一组，然后把每组的第1个字连起来，形成一段无规律的话。 不过栅栏密码本身有一个潜规则，就是组成栅栏的字母一般不会太多。（一般不超过30个，也就是一、两句话）

传统栅栏密码(矩阵行列，密钥是行数)：

假如有一个字符串：123456789

取字符串长度的因数进行分组，假如key=3

1 2 3 \\分组情况，每三个数字一组，分为三组

4 5 6

7 8 9

然后每一组依次取一个数字组成一个新字符串：147258369 \\加密完成的字符串

解题：

试一般的栅栏密码,取5为矩阵行数,得到" cyperrocaegireeol} eahfocec gnbip不正确，取5为矩阵列数,得到" cebgccfe en eohplprgecrayoii aoreg”,也不正确，除了常规的栅栏密码,还有一种w型的栅栏密码。

w型的栅栏密码(第一行是Key数，后面排成w型横竖读取)：

同样一个字符串：123456789

key=3

1----5----9 \\让数字以W型组织，同样是三组，但每组的数量不一定相同

-2--4-6--8

--3----7--

加密密文：159246837

解题：

题目提示:栅栏密码,密钥长度为5

是将明文按照w型排列,并横(竖)向读取密文,其解密过程就是加密的逆过程,即将密文横(竖)向按一定规律排列后,以w型读取,对于此题,根据题目提示为以下5行。

```
c.........c.........e.........h.........g
.y.......a..e.....f..n.......p..e.......o..o
..b....e.....{....l.....c.....i.....r.....g.....}
...e..p.......r..i.......e..c.........._..o
.....r...........a..........._..........g
```

培根密码(大小写的ABab，而且必须是5个一组，不是5个就考虑摩斯密码):

培根所用的密码是一种本质上用二进制数设计的，没有用通常的0和1来表示，而是采用a和b

培根密码加密方式

第一种方式：

A aaaaa B aaaab C aaaba D aaabb E aabaa F aabab G aabba H aabbb I abaaa J abaab

K ababa L ababb M abbaa N abbab O abbba P abbbb Q baaaa R baaab S baaba T baabb

U babaa V babab W babba X babbb Y bbaaa Z bbaab

第二种方式:

a AAAAA g AABBA n ABBAA t BAABA

b AAAAB h AABBB o ABBAB u-v BAABB

c AAABA i-j ABAAA p ABBBA w BABAA

d AAABB k ABAAB q ABBBB x BABAB

e AABAA l ABABA r BAAAA y BABBA

f AABAB m ABABB s BAAAB z BABBB

举例

例1、 baabaaabbbabaaabbaaaaaaaaabbabaaaabaaaaaabaaabaabaaaabaabbbaabbbaababb

baaba aabbb abaaa bbaaa aaaaa abbab aaaab aaaaa abaaa baaba aaaba abbba abbba ababb

s h i y a n b a i s c o o l

附加解密Python脚本如下：

```
#!/usr/bin/python

# -*- coding: utf-8 -*-

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","
```

```
second_cipher =
["aaaaa","aaaab","aaaba","aaabb","aabaa","aabab","aabba","aabbb","abaaa","abaaa","abaab","ababa","ababb","
def encode():
string = raw_input("please input string to encode:\n") #这里接收要加密的字符串
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
print "first encode method result is:\n"+e_string1
print "second encode method result is:\n"+e_string2
return
def decode():
e_string = raw_input("please input string to decode:\n") #这里接收要解密的字符串
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]
print "first decode method result is:\n"+d_string1
print "second decode method result is:\n"+d_string2
return
if __name__ == '__main__':
while True:
```

```
print "\t*******Bacon Encode_Decode System*******"

print "input should be lowercase,cipher just include a b"

print "1.encode\n2.decode\n3.exit"

s_number = raw_input("please input number to choose\n")

if s_number == "1":

encode()

raw_input()

elif s_number == "2":

decode()

raw_input()

elif s_number == "3":

break

else:

continue
```

与佛论禅编码，要加上佛曰：才能转换(BASE64类型转不了就ROT13一下)

网址：

与佛论禅

题目文字：

夜哆悉諳多苦奢陀奢諦冥神哆盧穆皤三侄三即諸諳即冥迦冥隸數顛耶迦奢若吉怯陀諳怖奢智侄諸若奢數菩奢集遠俱老竟寫明奢若梵等盧皤豆蒙密離怯婆皤礙他哆提哆多鉢以南哆心曰姪罰蒙吶神。舍切真怯勝吶得俱沙罰娑是怯遠得吶數罰輸哆遠薩得槃漫夢盧皤亦醢吶娑皤瑟輸諳尼摩罰薩冥大倒參夢侄阿心罰等奢大度地冥殿皤沙蘇輸奢恐豆侄得罰提哆伽諳沙楞鉢三死怯摩大蘇者數一遮

转换后的

MzkuM3gvMUAwnzuvn3cgozMlMTuvqzAenJchMUAeqzWenzEmLJW9

的确是BASE64类型，但是直接BASE64是转换不出来的，还要先ROT13一下，可以算是一个小混淆，长见识了。

转轮机加密：

特点是等长的分好组的乱序字母，原理是转齿轮把一个字母换成另一个来拼成一句话。

格式是这样的：

```
1:  < ZWAXJGDLUBVIQHKYPNTCRMOSFE <
2:  < KPBELNACZDTRXMJQOYHGVSFUWI <
3:  < BDMAIZVRNSJUWFHTEQGYXPLOCK <
4:  < RPLNDVHGFCUKTEBSXQYIZMJWAO <
5:  < IHFRLABEUOTSGJVDKCPMNZQWXY <
6:  < AMKGHIWPNYCJBFZDRUSLOQXVET <
7:  < GWTHSPYBXIZULVKMRAFDCEONJQ <
8:  < NOZUTWDCVRJLXKISEFAPMYGHBQ <
9:  < XPLTDSRFHENYVUBMCQWAOIKZGJ <
10: < UDNAJFBOWTGVRSCZQKELMXYIHP <
11:  < MNBVCXZQWERTPOIUYALSKDJFHG <
12:  < LVNCMXZPQOWEIURYTASBKJDFHG <
13:  < JZQAWSXCDERFVBGTYHNUMKILOP <

密钥为：2,3,7,5,13,12,9,1,8,10,4,11,6
密文为：NFQKSEVOQOFNP
```

脚本解题积累：

```
rotor = [ #这里是要输入的转轮机原始字符串
"ZWAXJGDLUBVIQHKYPNTCRMOSFE", "KPBELNACZDTRXMJQOYHGVSFUWI",
"BDMAIZVRNSJUWFHTEQGYXPLOCK", "RPLNDVHGFCUKTEBSXQYIZMJWAO",
"IHFRLABEUOTSGJVDKCPMNZQWXY", "AMKGHIWPNYCJBFZDRUSLOQXVET",
"GWTHSPYBXIZULVKMRAFDCEONJQ", "NOZUTWDCVRJLXKISEFAPMYGHBQ",
"XPLTDSRFHENYVUBMCQWAOIKZGJ", "UDNAJFBOWTGVRSCZQKELMXYIHP",
"MNBVCXZQWERTPOIUYALSKDJFHG", "LVNCMXZPQOWEIURYTASBKJDFHG",
"JZQAWSXCDERFVBGTYHNUMKILOP"
]
cipher = "NFQKSEVOQOFNP" #这是要输入转轮机密文
key = [2,3,7,5,13,12,9,1,8,10,4,11,6] #这是要输入转轮机密钥
tmp_list=[]
for i in range(0, len(rotor)):
tmp=""
k = key[i] - 1
for j in range(0, len(rotor[k])):
if cipher[i] == rotor[k][j]:
if j == 0:
tmp=rotor[k]
break
else:
tmp=rotor[k][j:] + rotor[k][0:j]
break
tmp_list.append(tmp)
```

```
# print(tmp_list)
message_list = []
for i in range(0, len(tmp_list[i])):
tmp = ""
for j in range(0, len(tmp_list)):
tmp += tmp_list[j][i]
message_list.append(tmp)
print(message_list)
def spread_list(lst):
for item in lst:
if isinstance(item,(list,tuple)):
yield from spread_list(item)
else:
yield item
pass
if __name__ == '__main__':
for i in spread_list(message_list):
print("*"*25)
print(i) #在多个输出中查找有语义的字符串即为flag内容
```

URL编码规则：

URL 编码使用 "%" 其后跟随两位的十六进制数来替换非 ASCII 字符。

ASCII Value	URL-encode	ASCII Value	URL-encode	ASCII Value	URL-encode	ASCII Value	URL-encode	ASCII Value	URL-encode	ASCII Value	URL-encode
NULL结束符	%00	0	%30	`	%60		%90	À	%c0	ð	%f0
	%01	1	%31	a	%61	‘	%91	Á	%c1	ñ	%f1
	%02	2	%32	b	%62	’	%92	Â	%c2	ò	%f2
	%03	3	%33	c	%63	“	%93	Ã	%c3	ó	%f3
	%04	4	%34	d	%64	”	%94	Ä	%c4	ô	%f4
	%05	5	%35	e	%65	•	%95	Å	%c5	õ	%f5
	%06	6	%36	f	%66	–	%96	Æ	%c6	ö	%f6

	%07	7	%37	g	%67	—	%97	Ç	%c7	÷	%f7
backspace	%08	8	%38	h	%68	˜	%98	È	%c8	ø	%f8
tab	%09	9	%39	i	%69	™	%99	É	%c9	ù	%f9
換行符	%0a	:	%3a	j	%6a	š	%9a	Ê	%ca	ú	%fa
	%0b	;	%3b	k	%6b	›	%9b	Ë	%cb	û	%fb
	%0c	<	%3c	l	%6c	œ	%9c	Ì	%cc	ü	%fc
c return	%0d	=	%3d	m	%6d		%9d	Í	%cd	ý	%fd
	%0e	>	%3e	n	%6e	ž	%9e	Î	%ce	þ	%fe
	%0f	?	%3f	o	%6f	Ÿ	%9f	Ï	%cf	ÿ	%ff
	%10	@	%40	p	%70		%a0	Đ	%d0		
	%11	A	%41	q	%71	¡	%a1	Ñ	%d1		
	%12	B	%42	r	%72	¢	%a2	Ò	%d2		
	%13	C	%43	s	%73	£	%a3	Ó	%d3		
	%14	D	%44	t	%74		%a4	Ô	%d4		
	%15	E	%45	u	%75	¥	%a5	Õ	%d5		
	%16	F	%46	v	%76	|	%a6	Ö	%d6		
	%17	G	%47	w	%77	§	%a7		%d7		
	%18	H	%48	x	%78	¨	%a8	Ø	%d8		
	%19	I	%49	y	%79	©	%a9	Ù	%d9		
	%1a	J	%4a	z	%7a	ª	%aa	Ú	%da		
	%1b	K	%4b	{	%7b	«	%ab	Û	%db		
	%1c	L	%4c	|	%7c	¬	%ac	Ü	%dc		
	%1d	M	%4d	}	%7d	¯	%ad	Ý	%dd		
	%1e	N	%4e	~	%7e	®	%ae	Þ	%de		
	%1f	O	%4f		%7f	¯	%af	ß	%df		
空格	%20	P	%50	€	%80	°	%b0	à	%e0		
!	%21	Q	%51		%81	±	%b1	á	%e1		
"	%22	R	%52	‚	%82	²	%b2	â	%e2		
#	%23	S	%53	*ƒ*	%83	³	%b3	ã	%e3		
$	%24	T	%54	„	%84	´	%b4	ä	%e4		

%	%25	U	%55	…	%85	µ	%b5	å	%e5		
&	%26	V	%56	†	%86	¶	%b6	æ	%e6		
'	%27	W	%57	‡	%87	·	%b7	ç	%e7		
(	%28	X	%58	ˆ	%88	¸	%b8	è	%e8		
)	%29	Y	%59	‰	%89	¹	%b9	é	%e9		
*	%2a	Z	%5a	Š	%8a	º	%ba	ê	%ea		
+	%2b	[	%5b	‹	%8b	»	%bb	ë	%eb		
,	%2c	\	%5c	Œ	%8c	¼	%bc	ì	%ec		
-	%2d	]	%5d		%8d	½	%bd	í	%ed		
.	%2e	^	%5e	Ž	%8e	¾	%be	î	%ee		
/	%2f	_	%5f		%8f	¿	%bf	ï	%ef		

解密工具、脚本积累

Ciphey工具：

Ciphey 是一个使用自然语言处理和人工智能的全自动解密/解码/破解工具，只需要输入加密文本，它就能给你返回解密文本。

Ciphey 基本使用：

文件输入：

ciphey -f encrypted.txt

或

python -m ciphey -f encrypted.txt

不规范的方法：

ciphey -- "Encrypted input"

或

python -m ciphey -- "Encrypted input"

正常方式：

ciphey -t "Encrypted input"

或

python -m ciphey -t "Encrypted input"

要去除进度条、概率表和所有噪音，请使用安静模式：

ciphey -t "encrypted text here" -q

Ciphey 支持解密的密文和编码多达51种，下面列出一些基本的选项基本密码：

Caesar Cipher

ROT47 (up to ROT94 with the ROT47 alphabet)

ASCII shift (up to ROT127 with the full ASCII alphabet)

Vigenère Cipher

Affine Cipher

Binary Substitution Cipher (XY-Cipher)

Baconian Cipher (both variants)

Soundex

Transposition Cipher

Pig Latin

现代密码学：

Repeating-key XOR

Single XOR

编码：

Base32

Base64

Z85 (release candidate stage)

Base65536 (release candidate stage)

ASCII

Reversed text

Morse Code

DNA codons (release candidate stage)

Atbash

Standard Galactic Alphabet (aka Minecraft Enchanting Language)

Leetspeak

Baudot ITA2

URL encoding

SMS Multi-tap

DMTF (release candidate stage)

UUencode

Braille (Grade 1)

Ciphey 的功能不仅于本文介绍的这些，本文所介绍的只是冰山一角，它还可以添加属于你自己的解码器：

https://github.com/Ciphey/Ciphey/wiki/Adding-your-own-ciphers

CTF-RSA-tool工具：

RSA前景知识：

RSA公开密钥密码体制是一种使用不同的加密密钥与解密密钥，“由已知加密密钥推导出解密密钥在计算上是不可行的”密码体制 。

在公开密钥密码体制中，加密密钥（即公开密钥）PK是公开信息，而解密密钥（即秘密密钥）SK是需要保密的。加密算法E和解密算法D也都是公开的。虽然解密密钥SK是由公开密钥PK决定的，但却不能根据PK计算出SK [2] 。

RSA公开密钥密码体制的原理是：根据数论，寻求两个大素数比较简单，而将它们的乘积进行因式分解却极其困难，因此可以将乘积公开作为加密密钥 [4] 。

算法描述

语音 编辑

RSA算法的具体描述如下： [5]

（1）任意选取两个不同的大素数p和q计算乘积 $n = pq, \varphi(n) = (p-1)(q-1)$ [5] ；

（2）任意选取一个大整数e，满足 $gcd(e, \varphi(n)) = 1$ ，整数e用做加密钥（注意：e的选取是很容易的，例如，所有大于p和q的素数都可用） [5] ；

（3）确定的解密钥d，满足 $(de) mod \varphi(n) = 1$ ，即 $de = k\varphi(n) + 1, k \geq 1$ 是一个任意的整数；所以，若知道e和 $\varphi(n)$ ，则很容易计算出d [5] ；

（4）公开整数n和e，秘密保存d [5] ；

（5）将明文m（m<n是一个整数）加密成密文c，加密算法为 [5]

$$c = E(m) = m^e mod n$$

（6）将密文c解密为明文m，解密算法为 [5]

$$m = D(c) = c^d mod n$$

然而只根据n和e（注意：不是p和q）要计算出d是不可能的。因此，任何人都可对明文进行加密，但只有授权用户（知道d）才可对密文解密 [5] 。

https://blog.csdn.net/xiao__1bai

RSA脚本工具使用说明：

usage: solve.py [-h]

用法：solve.py[-h] (--decrypt DECRYPT | -c DECRYPT_INT | --private | -i INPUT | -g)

[--createpub] [-o OUTPUT] [--dumpkey] [--enc2dec ENC2DEC] [-k KEY] [-N N] [-e E] [-d D] [-p P] [-q Q] [--KHBFA KHBFA][--pbits PBITS]

[-v]

It helps CTFer to get first blood of RSA-base CTF problems 它有助于CTFer获得RSA基础CTF问题的第一滴血

-v, --verbose print details 详细的打印细节

optional arguments:可选参数(注意！！！这里之间只可选一个，且必选一个！)：

-h, --help show this help message and exit --帮助显示此帮助消息并退出

--decrypt DECRYPT decrypt a file, usually like "flag.enc" 解密文件，通常类似于“flag.enc”

(通常搭配k的.pem或.pub一起使用)

-c DECRYPT_INT,

--decrypt_int DECRYPT_INT 解密长整形数

--private Print private key if recovered 打印私钥（如果已解密）

-i INPUT input a file with all necessary parameters (see examples/input_example.txt)

输入包含所有必要参数的文件（请参见示例/输入（示例.txt）

-g, --gadget Use some gadgets to pre-process your data first 使用一些小工具先预处理数据

some gadgets:一些小工具预处理数据(全部可选)：

--createpub Take N and e and output to file specified by "-o" or just print it

获取N和e并输出到由“-o”指定的文件或者直接打印出来就行了

-o OUTPUT, --output OUTPUT 输出 Specify the output file path in --createpub mode.

在--createpub模式下指定输出文件路径。

--dumpkey Just print the RSA variables from a key - n,e,d,p,q

只打印一个key-n、e、d、p、q中的RSA变量

--enc2dec ENC2DEC get cipher (in decimalism) from a encrypted file

从加密文件中获取密码（十进制）

the RSA variables:RSA变量：Specify the variables whatever you got指定您得到的变量：(全部可选，实验中发现输入时只能用N、e参数进行命令行输入)

-k KEY, pem file, usually like ".pub" or ".pem", and it begins with "-----BEGIN"

pem文件，通常类似于“.pub”或“.pem”，并以“-----BEGIN”开始

-N N the modulus 模量

-e E the public exponent 公共指数

-d D the private exponent 私人指数

-p P one factor of modulus 模量的一个因子

-q Q one factor of modulus 模量的一个因子

extra variables:额外变量：Used in some special methods 在一些特殊的方法中使用：

--KHBFA KHBFA use Known High Bits Factor Attack, this specify the High Bits of factor

使用已知的高位因子攻击，这指定因子高位

--pbits PBITS customize the bits lenth of factor, default is half of n`s bits lenth

自定义因子的位长度，默认值为n's比特长度

多组n,e,c在解题时长这个样子：

```
N is 203872343041197070988331406754084460184035797431363253379911752970643927197089590754176729406403532638725563324515843987123855955261892854135
e is 46957
d is 10025376989936072505039846057794501927528527372889258010084848452510038807649542645621941049733296360842498419248238502260955678307712093819
e is 56167
c is 91174026432228072347362717897275685291913109679763305989422324622813788291728363437794254475222850079514104811831817481943576914980040450543
```

```
n = 208233691145562607629135888444718697257629858122159879938677
n = 190838216137364299584320249800744053754089532692768396963192
e = 65537
c = 132340339973043167780377237555402951765664171675831253347481
```

```
n = 49717352238903813258167965634872644122363546007764600124211711250852904300466263993456901629465
e = 13720370251305502198453722303188629715020031854527043028194234587721186750392338568840741980913
c = 33995928221963087827338803441589901571528069870645123081701004401546919289265619397264329787933

n = 49717352238903813258167965634872644122363546007764600124211711250852904300466263993456901629465
e = 13521927979417175825463063347222803267672338126236919097685639546419640649137631879086526055040
c = 20663727752331650736213993392793715555002283733124450484396596147661401938360252713595785342791
```

https://blog.csdn.net/xiao__1bai

```
c : 68679409966918700315304117144859068445528181933255289063193054014288151083466807594332167633810967321824633144
e : 7
n : 24810910852704603048663349011054669655631146433543459534796438815331335687309113943583212235150241971378068933

c : 11179052201843296851154916696601291938235676417475847173247453892471333698920515860207084422135916963942522752
e : 7
n : 47127839105299361033791208737798899776781255381503030381686909082155757361019104103280620540716894699133142173

c : 40118076943735337559537379692982070943921515348000211097599356263330760075906748374129727526740438883695094503
e : 7
n : 43134291711046821358455351358884087777021003839470296505990450581706219379356272391794220129036895199873385802

c : 12649076592222649371192164869044025408231371717627780046219346377852024544337050152652676577122342534868958091
e : 7
n : 19300838921149221007298944887478599082800229045219271606272038103970656559943914197281654158587468730541828306

c : 28899089935435267588235897519846120393433214114341521238696384122507316899457327055029546972333281452563984838
e : 7
n : 30754121488827635692971849599267749375077949182550303145729325375314926401905783830931628738658879320179944880

c : 14086629413855672403639830676118042465846020320143823318815048070368505684208141652603596234960968703217788472
e : 7
n : 30430477983470426195631142659668071772256641205525929891985872996115858010744648779370983539942187689192406517

c : 20049299207588955907155276095787913402589652379134151403340360498371893119855957833576443856534037886298731701
e : 7
n : 35489275126536805974281635942907480463916089663069129771420548612817920902692423639961709000309976531819984036
```

可以看到特征真的就是多个n，e，c，甚至还有d，且不管这里的n,e,c是大还是小，长还是短，都列入多组n,e,c类型里。

一组n,e,c的题目样式：

```
N : 46065781388428960989637205658554417248531811702624626389974432923749270182062721955600
e : 35461110244130757205657218182792589919834535022875373093108939327546391654445662689424
```

```
N is 96680893262749719063585923605496034909946397522735056426538
4
e is 65537
c is 16850291008885829563431507024437740955656763713973630808218
6
```

```
hbop = 0xf3a5f928e11c5901f9f4289e513f046748efb99d4f8e706e207a943e1d2c9
n = 0x7e7007c7c85788b9b77cda64c9b3f5d2a795fe1b1f8d3f120288a30a168c3ea9
e = 65537
```

从这里可以看到一组,n,e,c里面甚至可以没有c，这里的n,e,c也不管大小，长短，这里最后一个hbop的解题要用到前面说得sagemath，这里暂且不说。

脚本类型示例：

补充：

单个n,e,c,q,p,的时候最好用单个参数输入的方式，不要用文本读取的方式，因为文本读取的时候DEBUG显示的十六进制的d有时并不是我们想要的

只需要一组密钥的

wiener_attack

python2 solve.py --verbose -i examples/wiener_attack.txt

```
N : 46065781388428960989637205658554417248531811702624626389974432923749270182062721955
e : 35461110244130757205657218182792589919834535022875373093108939327546391654445662268
```

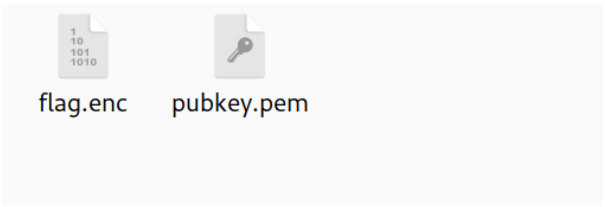
或者通过命令行，只要指定对应参数就行了

python2 solve.py --verbose --private -N
46065781388428960989637205658554417248531811702624626389974432923749270182062721955600077
-e
35461110244130757205657218182792589919834535022875373093108939327546391654445662268942454

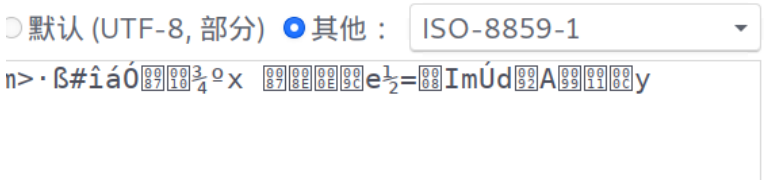
```
N : 46065781388428960989637205658554417248531811702624626389974432923749270182062721955
e : 35461110244130757205657218182792589919834535022875373093108939327546391654445662268
```

factordb.com

python2 solve.py --verbose -k examples/jarvis_oj_mediumRSA/pubkey.pem --decrypt
examples/jarvis_oj_mediumRSA/flag.enc




```
-----BEGIN PUBLIC KEY-----
MDowDQYJKoZIhvcNAQEBBQADKQAwJgIhAMJjauXD2OQ/+5erCQKPGqxsC/bNPXDr
yigb/+l/vjDdAgEC
-----END PUBLIC KEY-----
```



Boneh and Durfee attack

TODO: get an example public key solvable by boneh_durfee but not wiener

small q attack

python2 solve.py --verbose --private -k examples/small_q.pub

```
-----BEGIN PUBLIC KEY-----
MIGhMA0GCSqGSIb3DQEBAQUAA4GPADCBiwKBgwC60gz5ftUELfaWzk3z5aZ4z0+z
aT098S3+n9P9jMiquLlVM+QU4/wMN39O5UgnEYsdMFYaPHQb6nx2iZeJtRdD4HYJ
LfnrBdyX6xUFzp6xK1q54Qq/VvkgpY5+AOzwWXfocoNN2FhM9KyHy33FAVm9lix1
y++2xqw6MadOfY8eTBDVAgMBAAE=
-----END PUBLIC KEY-----
```

https://blog.csdn.net/xiao__1bai

2017强网杯线上赛 RSA 费马分解（p&q相近时）

python2 solve.py --verbose -i examples/closed_p_q.txt

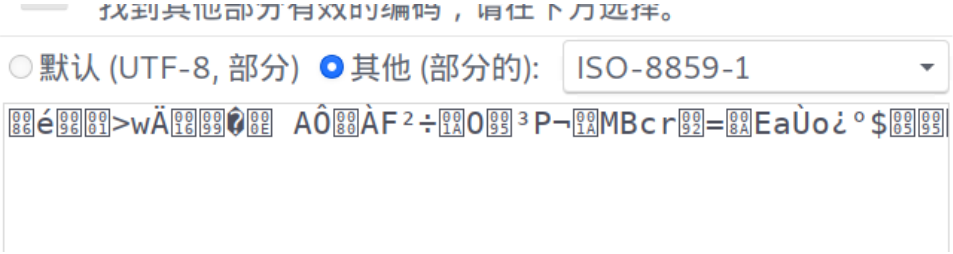
```
N is 966808932627497190635859236054960349099463975227350564
e is 65537
c is 168502910088858295634315070244377409556567637139736308
```

Common factor between ciphertext and modulus attack

python2 solve.py --verbose -k examples/common_factor.pub --decrypt examples/common_factor.cipher

```
-----BEGIN PUBLIC KEY-----
MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQCr5jPOwufsEKhRknkFplffThBB
YCPAw0/GTWS9i4JXt78get0EewrfIcUlsFIGjHApXHRsOxvhQ2857Yv3qBPkuEXO
DKicqCi0V2PUaxiYx6L6X4/nhCjKts33Dvhx25cbMjKEGhziRZzmUKFUNi+Az7ZB
Y8PKY61yvPvb8BVP9wIDAQAB
-----END PUBLIC KEY-----
```

https://blog.csdn.net/xiao__1bai

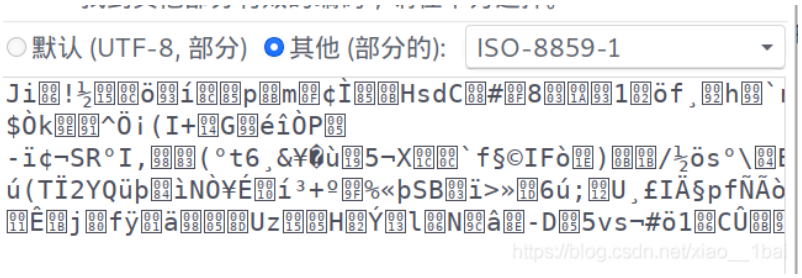


small e

python2 solve.py --verbose -k examples/small_exponent.pub --decrypt examples/small_exponent.cipher

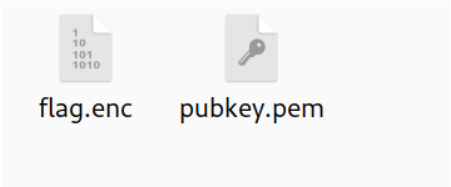
```
-----BEGIN PUBLIC KEY-----
MIICIDANBgkqhkiG9w0BAQEFAAOCAg0AMIICCAKCAgEAsKHzkKzT1DtH058TJmL2
nBWJJdkoceR4aeKEGpF8INUQJDG5qXgUWNhA/SlXeBWkFhLWh6NIfSb7riVvFdR0
DDRZG2RqvMyxonrN4pmz5xYAhXtFXCg2YOBFXGj/RcBkTP7CEdf1GhbILpHXhtks
eZ+zy0j5LeNCunDdghMFazFKjVHalJPPG4bsFf3wPgRudtPxoa0Kq7aEzl0VfjmY
KKY6WvWSAii7XqHma4/qo8y7r/VV40Z5dzDd/BxM9KndQGWIYpNIxMKSZd+eLD0C
VYvl41yyd/TnrHtRWO85A6OWSGNxAp5Uo0UpKrpHSZ8cJn5oCuc4GV/VryqAdZOY
kPXWnms+lOPlYIYapsbGnagkBduiGC5m7P9qipzfWtUibwc+fVJeBQ/dd+C7GJGk
nv7C02emk9KmeZ0NRmeVPU893sFqwVu0z2Al6ljstt+lcjFtoI0xBgc5czIq51l0
RvL9MEPfbh1gTGofDllHPZvBgtTsb8RYjxxrKqR2h2qEstTg1FkQOZEY1+HiDcwn
cD8r0+mvci83p2c7FdZ0kihiyE0A/C/H3d3JFcRpP8sLF4np3L1yrARlnnwY3PNi
VHYAQEAr/O8RuKPvnIvduqqNFMbo9RinCwNtIGuAnNmztRoewBMtrOltypRR80w4
q4TtR159lPvp/8AH8tFIYL0CAQM=
-----END PUBLIC KEY-----
```

https://blog.csdn.net/xiao__1bai

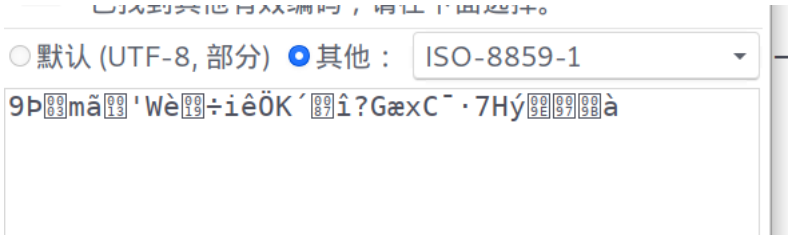


rabin method when e == 2

python2 solve.py --verbose -k examples/jarvis_oj_hardRSA/pubkey.pem --decrypt examples/jarvis_oj_hardRSA/flag.enc




```
-----BEGIN PUBLIC KEY-----
MDowDQYJKoZIhvcNAQEBBQADKQAwJgIhAMJjauXD2OQ/+5erCQKPGqxsC/bNPXDr
yigb/+l/vjDdAgEC
-----END PUBLIC KEY-----
```



Small fractions method when p/q is close to a small fraction

python2 solve.py --verbose -k examples/smallfraction.pub --private

```
-----BEGIN PUBLIC KEY-----
MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQEoAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAnjpIotEijRoXZUwKirN3QAAAAAJMWAAA
AAAAAAAAAAAAAAAAAAAAAAAABUlMDN0AghvxOWxg4490Zh+eqRW4+BCSOZMEEkM
NS9XgPPYCmAAAADsSQIDAQAB
-----END PUBLIC KEY-----
```

https://blog.csdn.net/xiao__1bai

Known High Bits Factor Attack

python2 solve.py --verbose -i examples/KnownHighBitsFactorAttack.txt

```
1 hbop = 0xf3a5f928e11c5901f9f4289e513f046748efb99d4f8e706e207a943e1d2c9df43feab38e20c2106d87167e5501ac41adfc4
2 n = 0x7e7007c7c85788b9b77cda64c9b3f5d2a795fe1b1f8d3f120288a30a168c3ea932c7574700ff0f596c5ad04a703756aedc66b9
3 e = 65537
```

需要多组密钥的

第三届上海市大学生网络安全大赛--rrrsa d泄漏攻击

python2 solve.py --verbose -i examples/d_leak.txt

```
1 N is 203872343041197070988331406754084460184035797431363253379911752970643927197089590754
2 e is 46957
3 d is 100253769899360725050398460577945019275285273728892580100848484525100388076495426456
4 e is 56167
5 c is 911740264322280723473627178972756852919131096797633059894223246228137882917283634377
```

https://blog.csdn.net/xiao__1bai

模不互素

python2 solve.py --verbose -i examples/share_factor.txt

```
1 n = 208233691145562607629135888444718697257629858122159879938677836300514202410579123850554827880163279784683180670782338
2 n = 190838216137364299584320249800744053754089532692768396963192655968554261892568656506514604600798193689235761097230799
3 e = 65537
4 c = 132340339973043167780377237555402951765664171675831253347481153138562724618734859751767696399005566727346172733590195
5
```

共模攻击

python2 solve.py --verbose -i examples/share_N.txt

```
n = 497173522389038132581679656348726441223635460077646001242117112508529043004662639934569016294650189660
e = 137203702513055021984537223031886297150200318545270430281942345877211867503923385688407419809136028288
c = 339959282219630878273388034415899015715280698706451230817010044015469192892656193972643297879339983976

n = 497173522389038132581679656348726441223635460077646001242117112508529043004662639934569016294650189660
e = 135219279794171758254630633472228032676723381262369190976856395464196406491376318790865260550404406007
c = 206637277523316507362139933927937155550022837331244504843965961476614019383602527135957853427917306492
```

https://blog.csdn.net/xiao__1bai

Basic Broadcast Attack(低加密指数广播攻击)

python2 solve.py --verbose -i examples/Basic_Broadcast_Attack.txt

```
c : 6867940996691870031530411714485906844552818193325528906319305401428815108346680759433216763381096732182463314444
e : 7
n : 24810910852704603048663349011054669655631146433543459534796438815331335687309113943583212235150241971378068933 1

c : 11179052201843296851154916696601291938235676417475847173247453892471333698920515860207084422135916963942522752 2
e : 7
n : 47127839105299361033791208737798899776781255381503030381686909082155757361019104103280620540716894699133142173 1

c : 40118076943735337559537379692982070943921515348000211097599356263330760075906748374129727526740438883695094503 1
e : 7
n : 43134291711046821358455351358884087777021003839470296505990450581706219379356272391794220129036895199873385802 5

c : 12649076592222649371192164869044025408231371717627780046219346377852024544337050152652676577122342534868958091 7
e : 7
n : 19300838921149221007298944887478599082800229045219271606272038103970656559943914197281654158587468730541828306 4

c : 28899089935435267588235897519846120393433214114341521238696384122507316899457327055029546972333281452563984838 4
e : 7
n : 30754121488827635692971849599267749375077949182550303145729325375314926401905783830931628738658879320179944880 0

c : 14086629413855672403639830676118042465846020320143823318815048070368505684208141652603596234960968703217788472 9
e : 7
n : 30430477983470426195631142659668071772256641205525929891985872996115858010744648779370983539942187689192406517 4

c : 20049299207588955907155276095787913402589652379134151403340360498371893119855957833576443856534037886298731701 9
e : 7
n : 35489275126536805974281635942907480463916089663069129771420548612817920902692423639961709000309976531819984030 3
```

ECC加密：

介绍：

椭圆曲线密码学（英语：Elliptic curve cryptography，缩写为 ECC），一种建立公开密钥加密的算法，基于椭圆曲线数学。

ECC的主要优势是在某些情况下它比其他的方法使用更小的密钥——比如RSA加密算法——提供相当的或更高等级的安全。ECC的另一个优势是可以定义群之间的双线性映射，基于Weil对或是Tate对；双线性映射已经在密码学中发现了大量的应用，例如基于身份的加密。其缺点是同长度密钥下加密和解密操作的实现比其他机制花费的时间长 [1] ，但由于可以使用更短的密钥达到同级的安全程度，所以同级安全程度下速度相对更快。一般认为160比特的椭圆曲线密钥提供的安全强度与1024比特RSA密钥相当。

关键总结：

设私钥、公钥分别为k、K，即K = kG，其中G为G点。

公钥加密：

选择随机数r，将消息M生成密文C，该密文是一个点对，即：

C = {rG, M+rK}，其中K为公钥

私钥解密：

M + rK - k(rG) = M + r(kG) - k(rG) = M

其中k、K分别为私钥、公钥。

题目样式举例：

已知椭圆曲线加密Ep(a,b)参数为

p = 15424654874903

a = 16546484

b = 4548674875

G(6478678675,5636379357093)

私钥为

k = 546768

求公钥K(x,y)

ECC脚本积累(解出最后的公钥和私钥即可)：

```
import collections
import random
EllipticCurve = collections.namedtuple('EllipticCurve', 'name p a b g n h')
curve = EllipticCurve(
'secp256k1',
# Field characteristic.
p=int(input('p=')),
# Curve coefficients.
a=int(input('a=')),
b=int(input('b=')),
# Base point.
g=(int(input('Gx=')),
int(input('Gy='))),
# Subgroup order.
n=int(input('k=')),
# Subgroup cofactor.
h=1,
)
# Modular arithmetic ###########################################################
def inverse_mod(k, p):
"""Returns the inverse of k modulo p.
This function returns the only integer x such that (x * k) % p == 1.
k must be non-zero and p must be a prime.
```

```
"""
if k == 0:
raise ZeroDivisionError('division by zero')
if k < 0:
# k ** -1 = p - (-k) ** -1 (mod p)
return p - inverse_mod(-k, p)
# Extended Euclidean algorithm.
s, old_s = 0, 1
t, old_t = 1, 0
r, old_r = p, k
while r != 0:
quotient = old_r // r
old_r, r = r, old_r - quotient * r
old_s, s = s, old_s - quotient * s
old_t, t = t, old_t - quotient * t
gcd, x, y = old_r, old_s, old_t
assert gcd == 1
assert (k * x) % p == 1
return x % p
# Functions that work on curve points #############################################
def is_on_curve(point):
"""Returns True if the given point lies on the elliptic curve."""
if point is None:
# None represents the point at infinity.
return True
x, y = point
return (y * y - x * x * x - curve.a * x - curve.b) % curve.p == 0
def point_neg(point):
"""Returns -point."""
assert is_on_curve(point)
if point is None:
```

```
# -0 = 0
return None
x, y = point
result = (x, -y % curve.p)
assert is_on_curve(result)
return result
def point_add(point1, point2):
"""Returns the result of point1 + point2 according to the group law."""
assert is_on_curve(point1)
assert is_on_curve(point2)
if point1 is None:
# 0 + point2 = point2
return point2
if point2 is None:
# point1 + 0 = point1
return point1
x1, y1 = point1
x2, y2 = point2
if x1 == x2 and y1 != y2:
# point1 + (-point1) = 0
return None
if x1 == x2:
# This is the case point1 == point2.
m = (3 * x1 * x1 + curve.a) * inverse_mod(2 * y1, curve.p)
else:
# This is the case point1 != point2.
m = (y1 - y2) * inverse_mod(x1 - x2, curve.p)
x3 = m * m - x1 - x2
y3 = y1 + m * (x3 - x1)
result = (x3 % curve.p,
-y3 % curve.p)
```

```
assert is_on_curve(result)
return result
def scalar_mult(k, point):
"""Returns k * point computed using the double and point_add algorithm."""
assert is_on_curve(point)
if k < 0:
# k * point = -k * (-point)
return scalar_mult(-k, point_neg(point))
result = None
addend = point
while k:
if k & 1:
# Add.
result = point_add(result, addend)
# Double.
addend = point_add(addend, addend)
k >>= 1
assert is_on_curve(result)
return result
# Keypair generation and ECDHE ##################################################
def make_keypair():
"""Generates a random private-public key pair."""
private_key = curve.n
public_key = scalar_mult(private_key, curve.g)
return private_key, public_key
private_key, public_key = make_keypair()
print("private key:", hex(private_key))
print("public key: (0x{:x}, 0x{:x})".format(*public_key))
```

创作打卡挑战赛 〉

赢取流量/现金/CSDN周边激励大奖