CTF-Crypto-学习笔记

原创

小龙在山东 于 2020-08-22 18:57:58 发布 5677 收藏 18

分类专栏： 安全实践

本文链接：https://blog.csdn.net/lilongsy/article/details/108171916

最常见的字符编码规范

ASCII

为了在计算机中表示字符，在设计编码的时候用**1个字节**也就是8bit位数来编码英文字符集

- 拉丁字母及标点符号
- 阿拉伯数字
- 一些控制字符

  这就是 ASCII（American Standard Code for InformationInterchange, 美国信息交换标准代码）

二进制	十进制	十六进制	缩写	解释	二进制	十进制	十六进制	字符	二进制	十进制	十六进制	字符	二进制	十进制	十六进制	字符
0000 0000	0	0	NUL	空字符(Null)	0010 0000	32	20	空格	0100 0001	65	41	A	0110 0001	97	61	a
0000 0001	1	1	SOH	标题开始	0010 0001	33	21	!	0100 0010	66	42	B	0110 0010	98	62	b
0000 0010	2	2	STX	正文开始	0010 0010	34	22	"	0100 0011	67	43	C	0110 0011	99	63	c
0000 0011	3	3	ETX	正文结束	0010 0011	35	23	#	0100 0100	68	44	D	0110 0100	100	64	d
0000 0100	4	4	EOT	传输结束	0010 0100	36	24	$	0100 0101	69	45	E	0110 0101	101	65	e
0000 0101	5	5	ENQ	请求	0010 0101	37	25	%	0100 0110	70	46	F	0110 0110	102	66	f
0000 0110	6	6	ACK	收到通知	0010 0110	38	26	&	0100 0111	71	47	G	0110 0111	103	67	g
0000 0111	7	7	BEL	响铃	0010 0111	39	27	'	0100 1000	72	48	H	0110 1000	104	68	h
0000 1000	8	8	BS	退格	0010 1000	40	28	(	0100 1001	73	49	I	0110 1001	105	69	i
0000 1001	9	9	HT	水平制表符	0010 1001	41	29	)	0100 1010	74	4A	J	0110 1010	106	6A	j
0000 1010	10	0A	LF	换行键	0010 1010	42	2A	*	0100 1011	75	4B	K	0110 1011	107	6B	k
0000 1011	11	0B	VT	垂直制表符	0010 1011	43	2B	+	0100 1100	76	4C	L	0110 1100	108	6C	l
0000 1100	12	0C	FF	换页键	0010 1100	44	2C	,	0100 1101	77	4D	M	0110 1101	109	6D	m
0000 1101	13	0D	CR	回车键	0010 1101	45	2D	-	0100 1110	78	4E	N	0110 1110	110	6E	n
0000 1110	14	0E	SO	不用切换	0010 1110	46	2E	.	0100 1111	79	4F	O	0110 1111	111	6F	o
0000 1111	15	0F	SI	启用切换	0010 1111	47	2F	/	0101 0000	80	50	P	0111 0000	112	70	p
0001 0000	16	10	DLE	数据链路转义	0011 0000	48	30	0	0101 0001	81	51	Q	0111 0001	113	71	q
0001 0001	17	11	DC1	设备控制1	0011 0001	49	31	1	0101 0010	82	52	R	0111 0010	114	72	r
0001 0010	18	12	DC2	设备控制2	0011 0010	50	32	2	0101 0011	83	53	S	0111 0011	115	73	s
0001 0011	19	13	DC3	设备控制3	0011 0011	51	33	3	0101 0100	84	54	T	0111 0100	116	74	t
0001 0100	20	14	DC4	设备控制4	0011 0100	52	34	4	0101 0101	85	55	U	0111 0101	117	75	u
0001 0101	21	15	NAK	拒绝接收	0011 0101	53	35	5	0101 0110	86	56	V	0111 0110	118	76	v
0001 0110	22	16	SYN	同步空闲	0011 0110	54	36	6	0101 0111	87	57	W	0111 0111	119	77	w
0001 0111	23	17	ETB	传输块结束	0011 0111	55	37	7	0101 1000	88	58	X	0111 1000	120	78	x
0001 1000	24	18	CAN	取消	0011 1000	56	38	8	0101 1001	89	59	Y	0111 1001	121	79	y
0001 1001	25	19	EM	介质中断	0011 1001	57	39	9	0101 1010	90	5A	Z	0111 1010	122	7A	z
0001 1010	26	1A	SUB	替补	0011 1010	58	3A	:	0101 1011	91	5B	[	0111 1011	123	7B	{
0001 1011	27	1B	ESC	溢出	0011 1011	59	3B	;	0101 1100	92	5C	\	0111 1100	124	7C	|
0001 1100	28	1C	FS	文件分割符	0011 1100	60	3C	<	0101 1101	93	5D	]	0111 1101	125	7D	}
0001 1101	29	1D	GS	分组符	0011 1101	61	3D	=	0101 1110	94	5E	^	0111 1110	126	7E	~
0001 1110	30	1E	RS	记录分离符	0011 1110	62	3E	>	0101 1111	95	5F	_				
0001 1111	31	1F	US	单元分隔符	0011 1111	63	3F	?	0110 0000	96	60	`				
0111 1111	127	7F	DEL	删除	0100 0000	64	40	@								

Python中，使用 `chr( )` 函数可得 `ASCII` 对应的字符，使用 `ord( )` 函数可得字符的 ASCII码。

Unicode 字符集

Unicode（万国码、统一码、国际码）用于编码世界
它是一本很厚的字典，为每一个字符规定了用来表示、存储的数字。
ASCII转Unicode：flag→ `flag`
中文转Unicode：你好→ `\u4f60\u597d`

进制转换

可见字符: `flag{congratulations!_you_got_it}`
字符串转字符：

```
s = "flag{congratulations!_you_got_it}"
s.encode()
# b'flag{congratulations!_you_got_it}'
```

可见字符的十六进制表示：
需要安装 `pycryptodome` ：

```
pip install pycryptodome
```

```
from Crypto.Util.number import *
s = "flag{congratulations!_you_got_it}"
hex(bytes_to_long(s.encode()))
```

可见字符的十进制表示：

```
from Crypto.Util.number import *
s = "flag{congratulations!_you_got_it}"
bytes_to_long(s.encode())
# 1185981498822494712295531163894582059771011824873819126601040329637687377064460 5
```

可见字符的二进制表示：

```
from Crypto.Util.number import *
s = "flag{congratulations!_you_got_it}"
print(bin(bytes_to_long(s.encode())))
# 0b1100110011011000110000101100111011110110110001101101111011011100110011101110010011000010111010001110101011010001100001011101000110100101101111011011100111001100100001010111110111100101101111011101010101111101100111011011101110100010111110110100101110100011111 01
```

URL（百分号）编码

统一资源定位符（Uniform Resource Locator, URL）是 Internet上 标准的资源地址，可视为网络上的门牌。
如果 URL 中出现了**拉丁字母、阿拉伯数字、.-_~外的符号**，则必须 使用百分号编码，所以 URL 编码也称为百分号编码。
如你在浏览器地址栏上看到：`https://www.baidu.com/s?wd=春节`
实际查询的网址是这样：`https://www.baidu.com/s?wd=%E6%98%A5%E8%8A%82`

```
import urllib.parse
s = "春节"
print(urllib.parse.quote(s))
# %E6%98%A5%E8%8A%82
print(urllib.parse.unquote("%E4%BD%A0%E5%A5%BD"))
# 你好
```

Base系列

Base64 / Base32 / Base16

这边的 Base 可理解为「基数」：
Base64 基数为 64，即 `6` 个比特位
Base32 基数为 32，即 5 个比特位
Base16 基数为 16，即 4 个比特位
ASCII 基数为 256，即 8 个比特位（不过最高位一定是 0）

Base64

Base64 索引表

数值	字符	数值	字符	数值	字符	数值	字符
0	A	16	Q	32	g	48	w
1	B	17	R	33	h	49	x
2	C	18	S	34	i	50	y
3	D	19	T	35	j	51	z
4	E	20	U	36	k	52	0
5	F	21	V	37	l	53	1
6	G	22	W	38	m	54	2
7	H	23	X	39	n	55	3
8	I	24	Y	40	o	56	4
9	J	25	Z	41	p	57	5
10	K	26	a	42	q	58	6
11	L	27	b	43	r	59	7
12	M	28	c	44	s	60	8
13	N	29	d	45	t	61	9
14	O	30	e	46	u	62	+
15	P	31	f	47	v	63	/

我们知道，ASCII 的一个字符对应 8 比特，所以3个字 符对应4个Base64 位。
例如:fff→二进制表示为01100110 0110011001100110
Base64也就是分为 011001 100110 011001 100110
对应的数值为 25 38 25 38
找到对应的Base64编码表 Z m Z m
不足 3 的倍数怎么办，直接在末尾填充全 0 。
填充几个字节就再最后加上几个 =
所以base64只可能有一个=号或者两个=号

Base系列表现形式

Ascii字符串:`flag{congratulations_you_got_it}`
Base64:

```
import base64
s = b"flag{congratulations_you_got_it}"
print(base64.b64encode(s))
# ZmxhZ3tjb25ncmF0dWxhdGlvbnNfeW91X2dvdF9pdH0=
```

Base32:

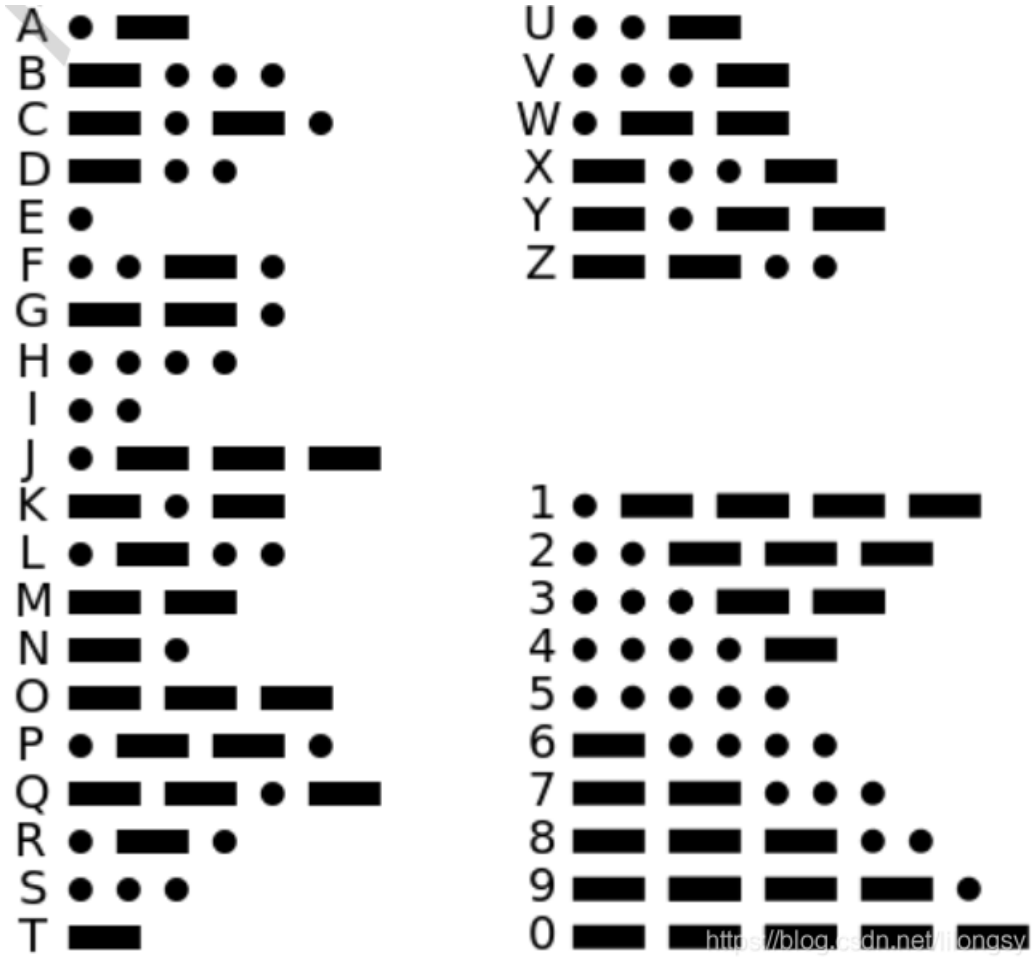
```
import base64
s = b"flag{congratulations_you_got_it}"
print(base64.b32encode(s))
# MZWGCZ33MNXW4Z3SMF2HK3DBORUW63TTL54W65K7M5XXIX3JOR6Q====
```

Base16:

```
import base64
s = b"flag{congratulations_you_got_it}"
print(base64.b16encode(s))
# 666C61677B636F6E67726174756C6174696F6E735F796F755F676F745F69747D
```

特征：
=号
base16的表现形式就是16进制的ascii字符串表示
建议遇到base系列的多尝试几种base系列的解码

摩斯编码



由**点**、**划**、**空格**组成。

- 字符内部的停顿：0
- 字符之间的停顿：000
- 单词之间的停顿：0000000
- 因无线电而生。
- 在线解密：
- http://ctf.ssleye.com/morse.html

JSFuck/BrainFuck/ook/aaencode

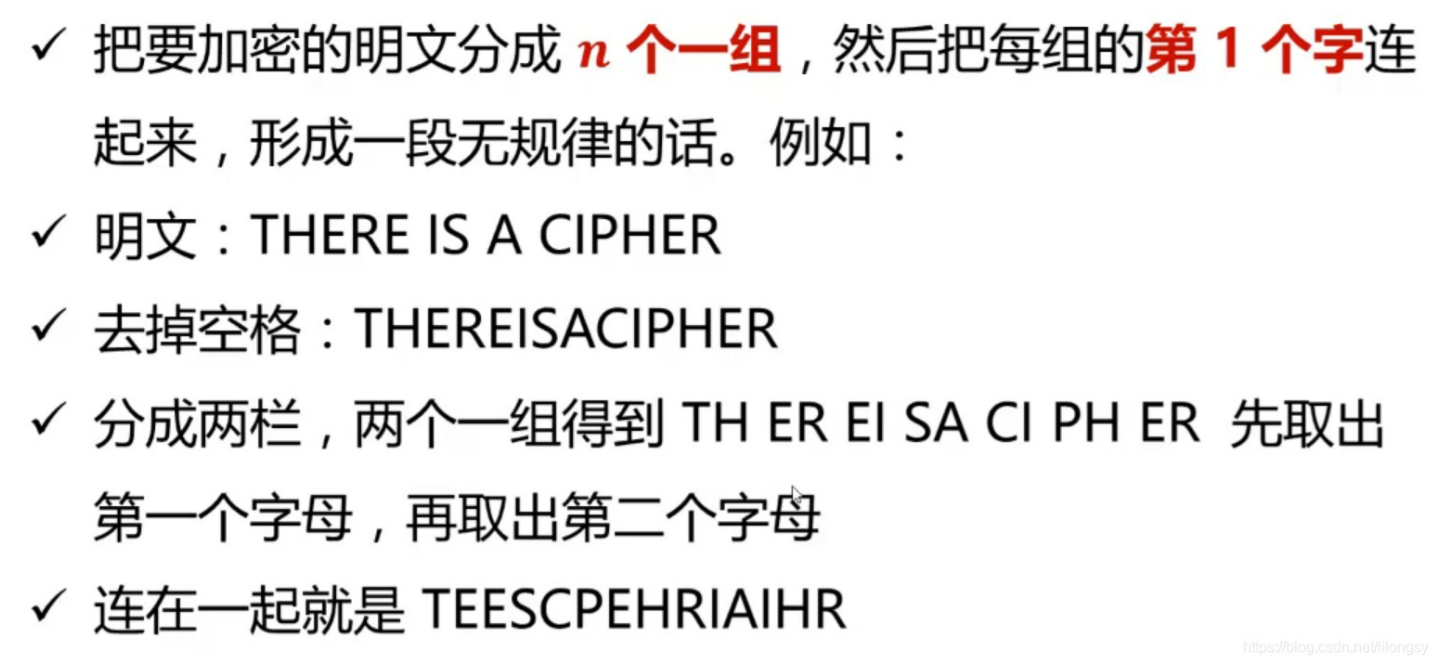
JSFuck:只用 6 个字符 !+ 来编写 JavaScript
BrainFuck:只有[->+空格
Ook！:表现形式就是Ook
Ook！:还有一种表现形式为!.？
aaencode:就是颜文字来编写JavaScript 程序， 表现形式就是颜文字。

其他形式的编码表现形式

Quoted-printable:你好→=E4=BD=A0=E5=A5=BD
UUencode:flag{test}→*9FQA9WMT97-T?0
XXencode:flag{test}→8NalVNrhoNLBoTE++
在线解密:http://web.chacuo.net/charsetuuencode

古典密码学

栅栏密码

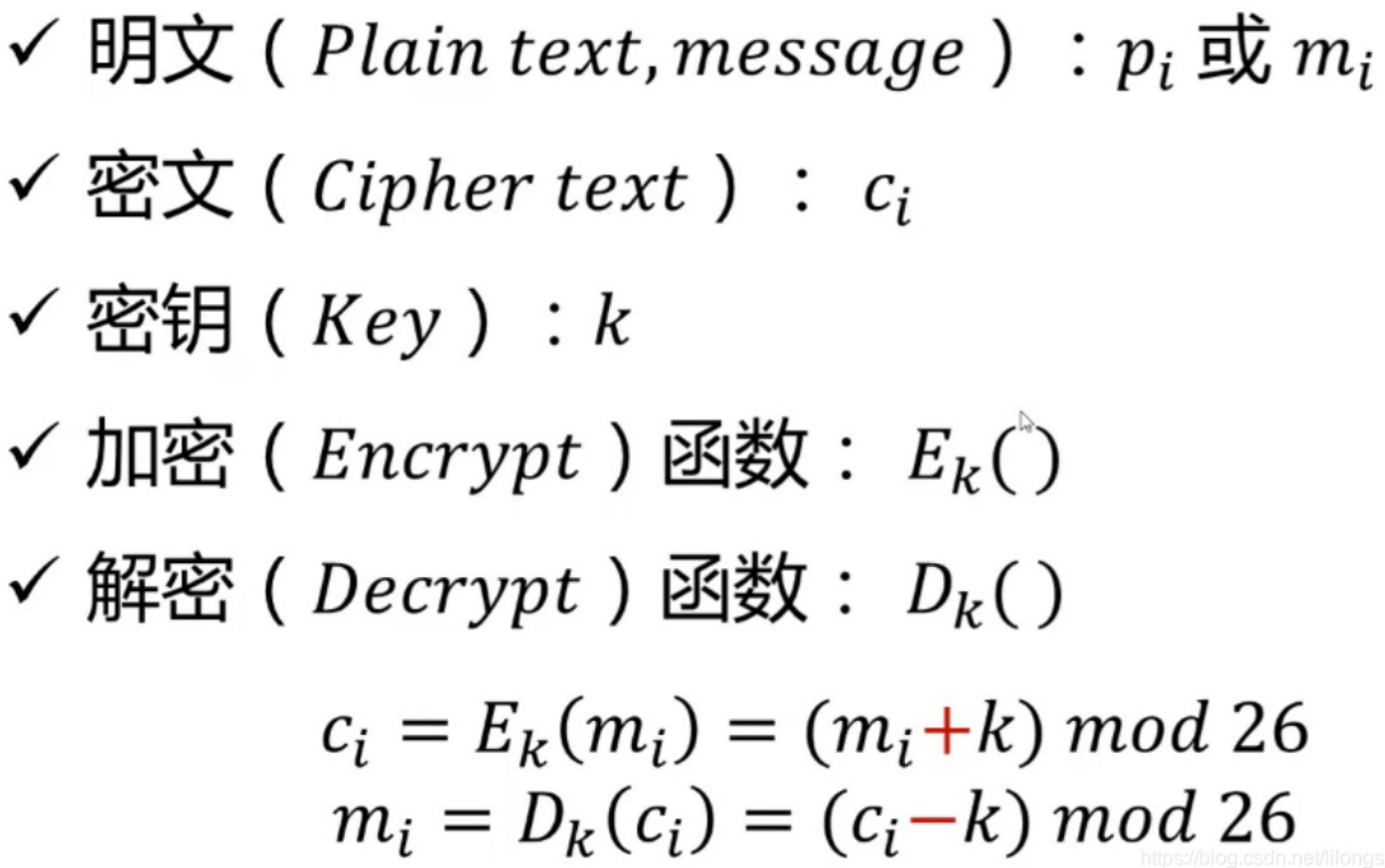


举例

篱笆墙的影子

```
felhaagv{ewtehtehfilnakgw}
```

凯撒密码

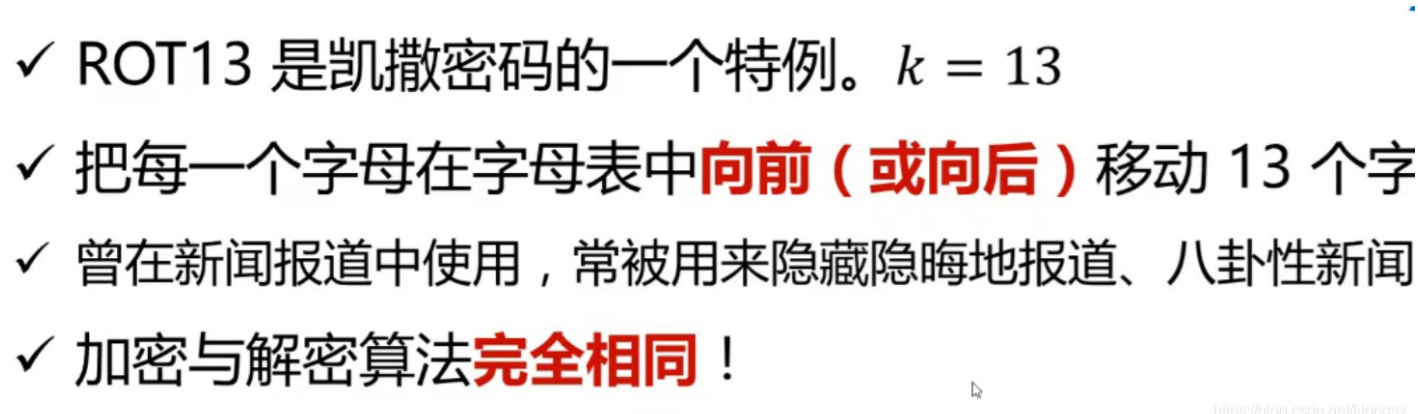


举例

凯撒？替换？呵呵！

```
MTHJ{CUBCGXGUGXWREXIPOYAOEYFIGXWRXCHTKHFCOHCFDUCGTXZOHIXOEOWMEHZO}
```

ROT13



ASCII 移位密码

✓ 与凯撒密码类似，还可以在**整个 ASCII 表中移位。**

ASCII Table

Dec	Hex	Oct	Char	Dec	Hex	Oct	Char	Dec	Hex	Oct	Char	Dec	Hex	Oct	Char
0	0	0		32	20	40	[space]	64	40	100	@	96	60	140	`
1	1	1		33	21	41	!	65	41	101	A	97	61	141	a
2	2	2		34	22	42	"	66	42	102	B	98	62	142	b
3	3	3		35	23	43	#	67	43	103	C	99	63	143	c
4	4	4		36	24	44	$	68	44	104	D	100	64	144	d
5	5	5		37	25	45	%	69	45	105	E	101	65	145	e
6	6	6		38	26	46	&	70	46	106	F	102	66	146	f
7	7	7		39	27	47	'	71	47	107	G	103	67	147	g
8	8	10		40	28	50	(	72	48	110	H	104	68	150	h
9	9	11		41	29	51	)	73	49	111	I	105	69	151	i
10	A	12		42	2A	52	*	74	4A	112	J	106	6A	152	j
11	B	13		43	2B	53	+	75	4B	113	K	107	6B	153	k
12	C	14		44	2C	54	,	76	4C	114	L	108	6C	154	l
13	D	15		45	2D	55	-	77	4D	115	M	109	6D	155	m
14	E	16		46	2E	56	.	78	4E	116	N	110	6E	156	n
15	F	17		47	2F	57	/	79	4F	117	O	111	6F	157	o
16	10	20		48	30	60	0	80	50	120	P	112	70	160	p
17	11	21		49	31	61	1	81	51	121	Q	113	71	161	q
18	12	22		50	32	62	2	82	52	122	R	114	72	162	r
19	13	23		51	33	63	3	83	53	123	S	115	73	163	s
20	14	24		52	34	64	4	84	54	124	T	116	74	164	t
21	15	25		53	35	65	5	85	55	125	U	117	75	165	u
22	16	26		54	36	66	6	86	56	126	V	118	76	166	v
23	17	27		55	37	67	7	87	57	127	W	119	77	167	w
24	18	30		56	38	70	8	88	58	130	X	120	78	170	x
25	19	31		57	39	71	9	89	59	131	Y	121	79	171	y
26	1A	32		58	3A	72	:	90	5A	132	Z	122	7A	172	z
27	1B	33		59	3B	73	;	91	5B	133	[	123	7B	173	{
28	1C	34		60	3C	74	<	92	5C	134	\	124	7C	174	|
29	1D	35		61	3D	75	=	93	5D	135	]	125	7D	175	}
30	1E	36		62	3E	76	>	94	5E	136	^	126	7E	176	~
31	1F	37		63	3F	77	?	95	5F	137	_	127	7F	177	

举例

```
'''
# -*- coding:utf-8 -*-
import A,SALT
from itertools import *

def encrypt(m, a, si):
    c=""
    for i in range(len(m)):
        c+=hex(((ord(m[i])) * a + ord(next(si))) % 128)[2:].zfill(2)
    return c
if __name__ == "__main__":
    m = 'flag{******************************}'
    a = A
    salt = SALT
    assert(len(salt)==3)
    assert(salt.isalpha())
    si = cycle(salt.lower())
    print("明文内容为：")
    print(m)
    print("加密后的密文为：")
    c=encrypt(m, a, si)
    print(c)
    #加密后的密文为：
    #177401504b0125272c122743171e2c250a602e3a7c206e014a012703273a3c0160173a73753d
'''
for a in range(128):
    for b in range(128):
        if (ord("f")*a+b)% 128==0x17:
            if (ord("g")*a+b)% 128==0x50:
                print(a,b)

a=57
salt=chr(97)+chr(104)+chr(104)
for b in range(128):
    if (ord("l")*a+b)% 128==0x74:
        print(b)
for b in range(128):
    if (ord("a")*a+b)% 128==0x01:
        print(b)
c=0x177401504b0125272c122743171e2c250a602e3a7c206e014a012703273a3c0160173a73753d
from Crypto.Util.number import *
c_bytes=long_to_bytes(c)
print(c_bytes)
from itertools import *
si=cycle(salt)
import gmpy2
a1=gmpy2.invert(a,128)
for i in c_bytes:
    print(chr(((i-ord(next(si)))*a1)%128),end="")
```

乘法密码

$$c_i = E_k(m_i) = (m_i \times k) \bmod 26$$
$$m_i = D_k(c_i) = (c_i \times k^{-1}) \bmod 26$$

✓ 模逆运算，满足：$k \times k^{-1} \equiv 1 \ (mod\ 26)$

✓ 例如 加密：m_i =4,$k = 3$,则 c_i=(4 ×3)=12 $mod\ 26 = 12$

✓ 解密：$m_i' = c_i \times \boldsymbol{k^{-1}} (mod\ 26) = 12 \times 9\ mod\ 26 = 4 =$

$\because 3 \times 9 = 27,\ 27\ mod\ 26 = 1$
$\therefore$ 当$k = 3$时，$k^{-1} = 9$

仿射密码

✓ 加法、乘法的组合：

$$c_i = E_k(m_i) = (a \cdot m_i + b) \bmod 26$$
$$m_i = D_k(c_i) = a^{-1} \cdot (c_i - b) \bmod 26$$

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
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25

以加密函数E(x) = (5x + 8) (mod 26)为例，以字母表26个字母作为编码系

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
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25

明文	A	F	F	I	N	E	C	I	P	H	E
x	0	5	5	8	13	4	2	8	15	7	4
y = 5x+8	8	33	33	48	73	28	18	48	83	43	28
y mod 26	8	7	7	22	21	2	18	22	5	17	2
密文	I	H	H	W	V	C	S	W	F	R	C

对应的加密结果为：IHHWVCSWFRCP

埃特巴什码

✓ 密文表是明文表的**逆转**：

- 明文：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：Z Y X W V U T S R Q P O N M L K J I H G F E D C B A

✓ 实例：

- 明文：the quick brown fox jumps over the lazy dog
- 密文：gsv jfrxp yildm ulc qfnkh levi gsv ozab wlt

单表替换密码的安全性

上面说的几种都是单表代换密码。

在单表替换加密中，所有的加密方式几乎都有一个共性，那就是**明密文一一对应**。

密钥空间小，直接爆破。

密钥空间大，尝试统计分析找到攻击点！

词频分析:https://quipqiup.com/

https://www.qqxiuzi.cn/daohang.htm

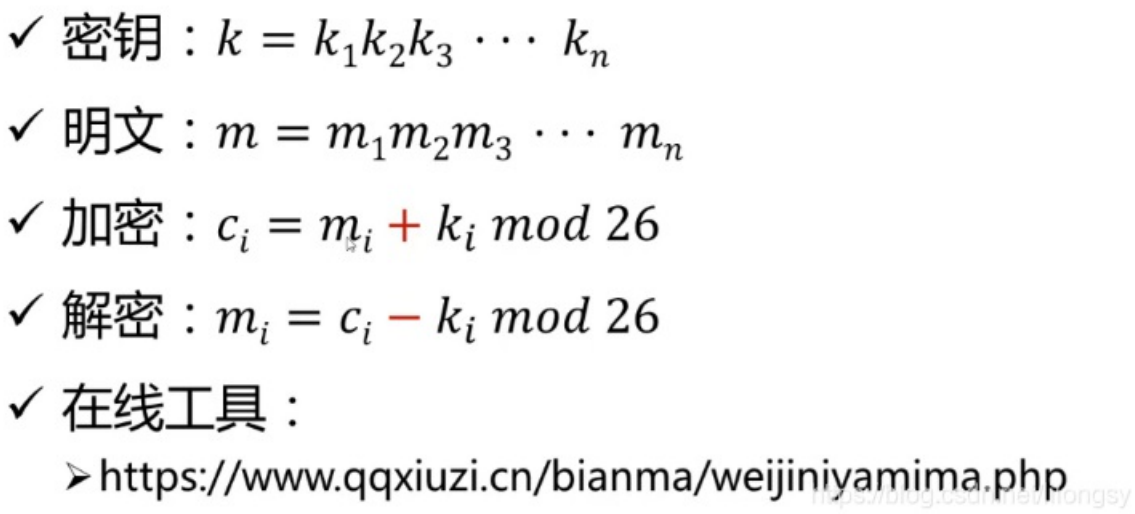
维吉尼亚密码 Vigenère cipher

✓ 加密过程：

- 密钥：encryption
- 明文：public key distribution
- 由于密钥字比明文短，所以要**重复书写密钥**以得与明文等长的密钥

密钥	e	n	c	r	y	p	t	i	o	n	e	n	c	r	y	p	t	i	o	r
明文	p	u	b	l	i	c	k	e	y	d	i	s	t	r	i	b	u	t	i	c
密文	T	H	D	C	G	R	D	M	M	Q	M	F	V	R	G	Q	N	B	W	I

26 × 26 方阵
密钥循环使用
不可破译的密码



在线工具：https://www.qqxiuzi.cn/bianma/weijiniyamima.php

举例：其实很简单

在学习了凯撒大帝使用的神奇密码后，密码前辈们有创造出了更为奇异的加密方法。本题出题者喜欢用helloworld当密钥，密文如下：dlpcsegkshrij,请破解后提交。附录是一张似乎有用的表。答案为非常规形式。

希尔（Hill）密码：基于矩阵乘法

希尔密码（Hill）使用每个字母在字母表中的顺序作为其对应的数字，即 A=0，B=1，C=2 等，然后将明文转化为 n 维向量，跟一个 n × n 的矩阵相乘，再将得出的结果模 26。注意用作加密的矩阵（即密匙）在 $\mathbb{Z}_{26}^n$ 必须是可逆的，否则就不可能解码。只有矩阵的行列式和 26 互质，才是可逆的。下面举一个例子

明文:ACT

将明文化为矩阵。

$$\begin{bmatrix} 0 \\ 2 \\ 19 \end{bmatrix}$$

假设密钥为：

$$\begin{bmatrix} 6 & 24 & 1 \\ 13 & 16 & 10 \\ 20 & 17 & 15 \end{bmatrix}$$

加密过程为：

$$\begin{bmatrix} 6 & 24 & 1 \\ 13 & 16 & 10 \\ 20 & 17 & 15 \end{bmatrix} \begin{bmatrix} 0 \\ 2 \\ 19 \end{bmatrix} \equiv \begin{bmatrix} 67 \\ 222 \\ 319 \end{bmatrix} \equiv \begin{bmatrix} 15 \\ 14 \\ 7 \end{bmatrix} \mod 26$$

密文即为

密文：POH

在线工具：http://www.practicalcryptography.com/ciphers/hill-cipher/

培根（Bacon）密码

✓ 加密时，明文中的每个字母都会转换成一组五个英文字母。

A/a	aaaaa	H/h	aabbb	O/o	abbba	V/v	babab
B/b	aaaab	I/i	abaaa	P/p	abbbb	W/w	babba
C/c	aaaba	J/j	abaab	Q/q	baaaa	X/x	babbb
D/d	aaabb	K/k	ababa	R/r	baaab	Y/y	bbaaa
E/e	aabaa	L/l	ababb	S/s	baaba	Z/z	bbaab
F/f	aabab	M/m	abbaa	T/t	baabb		
G/g	aabba	N/n	abbab	U/u	babaa		

明文:

s u c c e s s

baaab baabb aaaba aaaba aabaa baaab baaab

✓ 另一种形式：

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

在线工具：http://rumkin.com/tools/cipher/baconian.php

举例

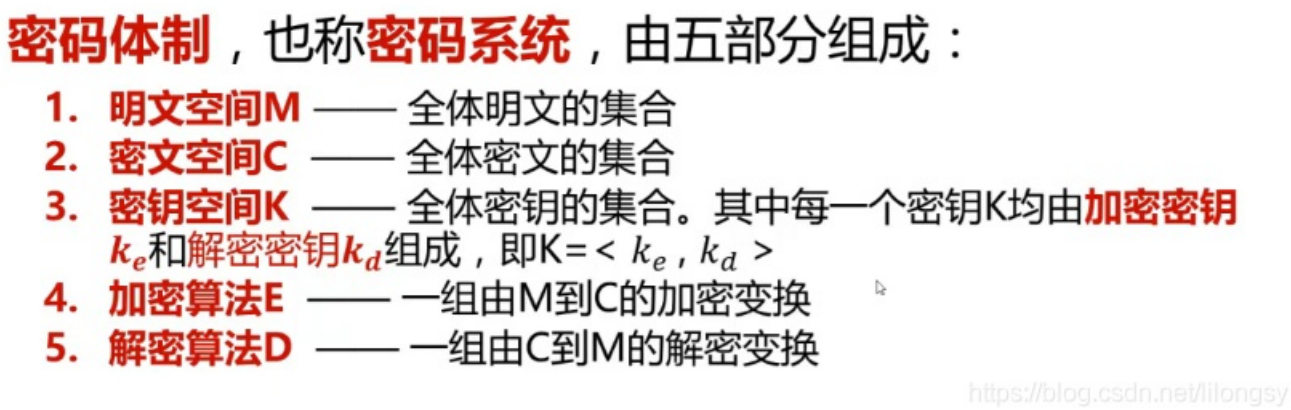
第一关：大帝
iodj{36g9i2777

第二关：滴滴滴
-... ----. ..--- -... .- -.-. ...-- ----. .- .- ..---

第三关：篱笆
a0dd}b6942c07

flag{36d9f2777b92bac39aa2ab206cd90d47}

现代密码

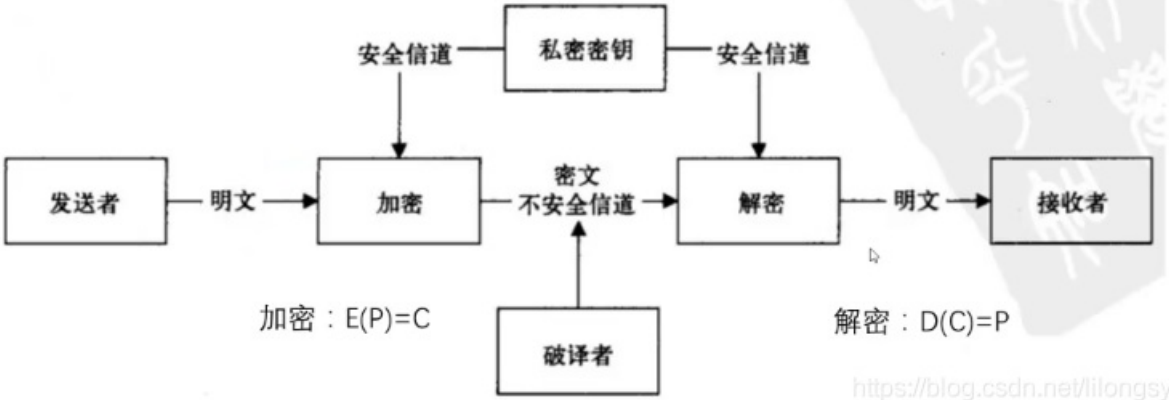


现代密码体制（系统）由五部分组成：

- 明文空间M
- 密文空间C
- 密钥空间K
- 加密算法E
- 解密算法D
  对称加密体制：加密密钥等于或约等于解密密钥，主要分为流加密和分组加密。
  流密码（Stream Cipher）：利用一个较短的种子密钥，通过密 钥流产生器，生成伪随机的密钥流，与明文逐比特异或，得到密文。
  根据异或运算的性质，解密算法也是异或运算。

公钥加密体制（双钥加密体制）:加密密钥（公开）≠ 解密密钥（保密） 加密算法 ≠ 解密算法。
安全性主要是基于数学上的计算困难问题，如大数分解RSA、离散对数、椭圆曲线上离散对数等。

加密模型



对称加密体制（单钥加密体制）

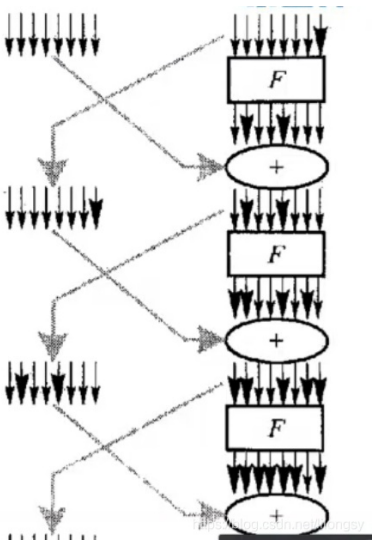
特点：
加密密钥 = 解密密钥
加密算法 ≈ 解密算法

- 速度快，效率高

主要分为**流密码**和**分组密码**

分组密码

- ✓ 首先将明文分组成多个**等长的块**（block），使用确定的算法和对称密钥对每组分别加密解密
- ✓ 两个**原则**：
  - ➢ **混淆（Confusion）**：密文的统计特性与密钥的取值之间的关系尽量复杂
  - ➢ **扩散（Diffusion）**：明文中的一个比特的变化影响密文中的多个比特，从而使明文的统计特征在密文中消失。会带来**雪崩效应。**

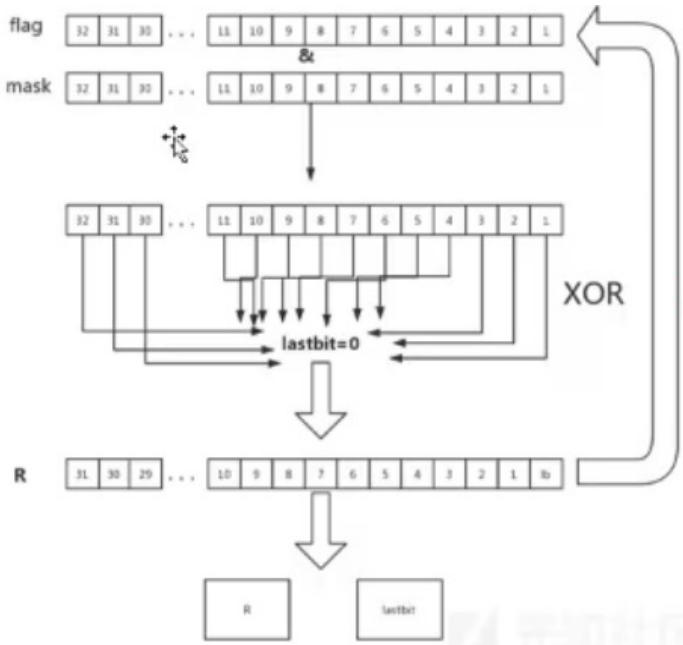


XOR

异性相吸

```
f1=open("key.txt","r")
key=f1.read()
print(key)
f2=open("密文.txt","rb")
c=f2.read()
print(c)
print(len(key))
print(len(c))
for i in range(len(key)):
    print(chr(ord(key[i])^c[i]),end="")

'''
asadsasdasdasdasdasdasdasdasdasdqwesqf
b'\x07\x1f\x00\x03\x08\x04\x12U\x03\x10TXK\\XJVSDR\x03D\x02XF\x06TG\x05VGWD\x12]J\x14\x1b'
38
38
flag{ea1bc0988992276b7f95b54a7435e89e}
'''
```

LFSR（线性反馈移位寄存器）



概括来说就是:将mask的二进制位为1的位置,对应种子flag的位置,让他们做亦或运算

例如:假设mask的二进制位中第2位和第12位为1,其余位置为0。那么就将flag的第2位和第12位的值亦或。然后将flag左移一位，最低位填充上刚才异或运算的结果，一直循环。

```
f=open("key","rb")
st=f.read()
out=""
for i in st:
    out+=bin(i)[2:].zfill(8)
#out="00001011100101011101010111100110110111001100110101010011110100001010000100111000000101101001 1"
key=out[18]+out[0:18]
print(key)
mask=0b1010011000100011100
flag=""
for x in range(19):
    tmp=0
    for i in range(19):
        if (mask>>i) &1:
            tmp^=int(key[18-i])
    flag=str(tmp)+flag
    key=key[18]+str(tmp)+key[1:18]
print(flag)
```

forecast

利用了python random的漏洞

flag.enc

```
499,123,495,149,245,118,133,441,273,140,421,391,268,493,269,413,273,386,115,107,94,459,134,481,271,172,361,287,9
2,235,501,477,169,416,501,477,470,421
```

```
import random

flag="************************************"

def test():
    f=open("test","wb")
    s=""
    for i in range(1000):
        s+=str(random.getrandbits(32))+"\n"
    f.write(s.encode())
    f.close()
test()
f=open("flag.enc","wb")
s=""
for i in flag:
    s+=str(ord(i)^random.getrandbits(9))+","
s=s[:-1]
f.write(s.encode())
f.close()
```

exp.py

```
c=[499,123,495,149,245,118,133,441,273,140,421,391,268,493,269,413,273,386,115,107,94,459,134,481,271,172,361,28
7,92,235,501,477,169,416,501,477,470,421]
from randcrack import RandCrack
rc=RandCrack()
f=open("test","r")
arr=[]
for i in range(1000):
    arr.append(int(f.readline().strip("\n")))
print(arr)
for i in arr[1000-624:]:
    rc.submit(i)

for i in c:
    print(chr((i^rc.predict_getrandbits(9))%128),end="")
```

randcrack.py

```
class RandCrack:

    def __init__(self):
        self.counter = 0
        self.mt = []
        self.state = False

    def submit(self, num):
        if self.state:
            raise ValueError("Already got enough bits")

        bits = self._to_bitarray(num)

        assert (all([x == 0 or x == 1 for x in bits]))
        self.counter += 1
        self.mt.append(self._harden_inverse(bits))
        if self.counter == 624:
            self._regen()
            self.state = True

    def _predict_32(self):
        if not self.state:
            raise ValueError("Didn't recieve enough bits to predict")

        if self.counter >= 624:
            self._regen()
        self.counter += 1

        return self._harden(self.mt[self.counter - 1])

    def predict_getrandbits(self, k):
        if not self.state:
            raise ValueError("Didn't recieve enough bits to predict")

        if k == 0:
            return 0
        words = (k - 1) // 32 + 1
        res = []
        for i in range(words):
            r = self._predict_32()
            if k < 32:
                r = [0] * (32 - k) + r[:k]
            res = r + res
```

```
            res = r + res
            k -= 32
        return self._to_int(res)

    def predict_randbelow(self, n):
        k = n.bit_length()
        r = self.predict_getrandbits(k)
        while r >= n:
            r = self.predict_getrandbits(k)
        return r

    def predict_randrange(self, start, stop=None, step=1, _int=int):
        # Adopted messy code from random.py module
        # In fact only changed _randbelow() method calls to predict_randbelow()
        istart = _int(start)
        if istart != start:
            raise ValueError("non-integer arg 1 for randrange()")
        if stop is None:
            if istart > 0:
                return self.predict_randbelow(istart)
            raise ValueError("empty range for randrange()")

        # stop argument supplied.
        istop = _int(stop)
        if istop != stop:
            raise ValueError("non-integer stop for randrange()")
        width = istop - istart
        if step == 1 and width > 0:
            return istart + self.predict_randbelow(width)
        if step == 1:
            raise ValueError("empty range for randrange() (%d,%d, %d)" % (istart, istop, width))

        # Non-unit step argument supplied.
        istep = _int(step)
        if istep != step:
            raise ValueError("non-integer step for randrange()")
        if istep > 0:
            n = (width + istep - 1) // istep
        elif istep < 0:
            n = (width + istep + 1) // istep
        else:
            raise ValueError("zero step for randrange()")

        if n <= 0:
            raise ValueError("empty range for randrange()")

        return istart + istep * self.predict_randbelow(n)

    def predict_randint(self, a, b):
        return self.predict_randrange(a, b + 1)

    def predict_choice(self, seq):
        try:
            i = self.predict_randbelow(len(seq))
        except ValueError:
            raise IndexError('Cannot choose from an empty sequence')
        return seq[i]

    def _to_bitarray(self, num):
        k = [int(x) for x in bin(num)[2:]]
```

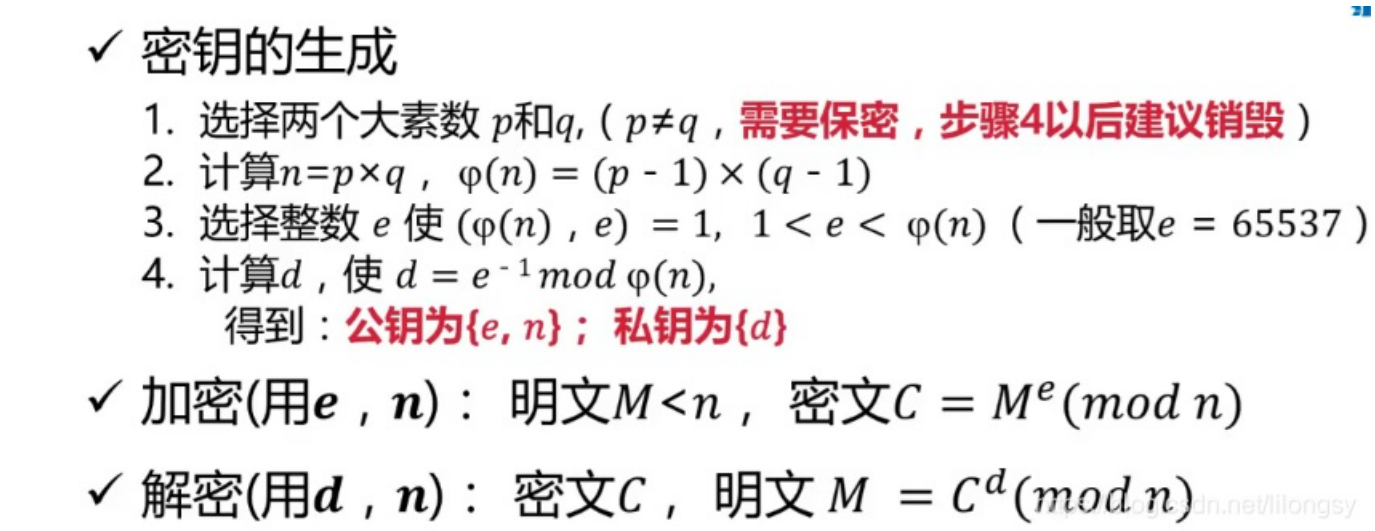
```
        return [0] * (32 - len(k)) + k

    def _to_int(self, bits):
        return int("".join(str(i) for i in bits), 2)

    def _or_nums(self, a, b):
        if len(a) < 32:
            a = [0] * (32 - len(a)) + a
        if len(b) < 32:
            b = [0] * (32 - len(b)) + b

        return [x[0] | x[1] for x in zip(a, b)]

    def _xor_nums(self, a, b):
        if len(a) < 32:
            a = [0] * (32 - len(a)) + a
        if len(b) < 32:
            b = [0] * (32 - len(b)) + b

        return [x[0] ^ x[1] for x in zip(a, b)]

    def _and_nums(self, a, b):
        if len(a) < 32:
            a = [0] * (32 - len(a)) + a
        if len(b) < 32:
            b = [0] * (32 - len(b)) + b

        return [x[0] & x[1] for x in zip(a, b)]

    def _decode_harden_midop(self, enc, and_arr, shift):

        NEW = 0
        XOR = 1
        OK = 2
        work = []
        for i in range(32):
            work.append((NEW, enc[i]))
        changed = True
        while changed:
            changed = False
            for i in range(32):
                status = work[i][0]
                data = work[i][1]
                if i >= 32 - shift and status == NEW:
                    work[i] = (OK, data)
                    changed = True
                elif i < 32 - shift and status == NEW:
                    if and_arr[i] == 0:
                        work[i] = (OK, data)
                        changed = True
                    else:
                        work[i] = (XOR, data)
                        changed = True
                elif status == XOR:
                    i_other = i + shift
                    if work[i_other][0] == OK:
                        work[i] = (OK, data ^ work[i_other][1])
                        changed = True

        return [x[1] for x in work]
```

```
        return [x[1] for x in work]

    def _harden(self, bits):
        bits = self._xor_nums(bits, bits[:-11])
        bits = self._xor_nums(bits, self._and_nums(bits[7:] + [0] * 7, self._to_bitarray(0x9d2c5680)))
        bits = self._xor_nums(bits, self._and_nums(bits[15:] + [0] * 15, self._to_bitarray(0xefc60000)))
        bits = self._xor_nums(bits, bits[:-18])
        return bits

    def _harden_inverse(self, bits):
        # inverse for: bits = _xor_nums(bits, bits[:-11])
        bits = self._xor_nums(bits, bits[:-18])
        # inverse for: bits = _xor_nums(bits, _and_nums(bits[15:] + [0] * 15 , _to_bitarray(0xefc60000)))
        bits = self._decode_harden_midop(bits, self._to_bitarray(0xefc60000), 15)
        # inverse for: bits = _xor_nums(bits, _and_nums(bits[7:] + [0] * 7 , _to_bitarray(0x9d2c5680)))
        bits = self._decode_harden_midop(bits, self._to_bitarray(0x9d2c5680), 7)
        # inverse for: bits = _xor_nums(bits, bits[:-11])
        bits = self._xor_nums(bits, [0] * 11 + bits[:11] + [0] * 10)
        bits = self._xor_nums(bits, bits[11:21])

        return bits

    def _regen(self):
        # C code translated from python sources
        N = 624
        M = 397
        MATRIX_A = 0x9908b0df
        LOWER_MASK = 0x7fffffff
        UPPER_MASK = 0x80000000
        mag01 = [self._to_bitarray(0), self._to_bitarray(MATRIX_A)]

        l_bits = self._to_bitarray(LOWER_MASK)
        u_bits = self._to_bitarray(UPPER_MASK)

        for kk in range(0, N - M):
            y = self._or_nums(self._and_nums(self.mt[kk], u_bits), self._and_nums(self.mt[kk + 1], l_bits))
            self.mt[kk] = self._xor_nums(self._xor_nums(self.mt[kk + M], y[:-1]), mag01[y[-1] & 1])

        for kk in range(N - M - 1, N - 1):
            y = self._or_nums(self._and_nums(self.mt[kk], u_bits), self._and_nums(self.mt[kk + 1], l_bits))
            self.mt[kk] = self._xor_nums(self._xor_nums(self.mt[kk + (M - N)], y[:-1]), mag01[y[-1] & 1])

        y = self._or_nums(self._and_nums(self.mt[N - 1], u_bits), self._and_nums(self.mt[0], l_bits))
        self.mt[N - 1] = self._xor_nums(self._xor_nums(self.mt[M - 1], y[:-1]), mag01[y[-1] & 1])

        self.counter = 0


if __name__ == "__main__":
    import random
    import time

    print("Testing random module cracker...")

    cracker = RandCrack()

    random.seed(time.time())

    for i in range(624):
        cracker.submit(random.randint(0, 4294967294))
```

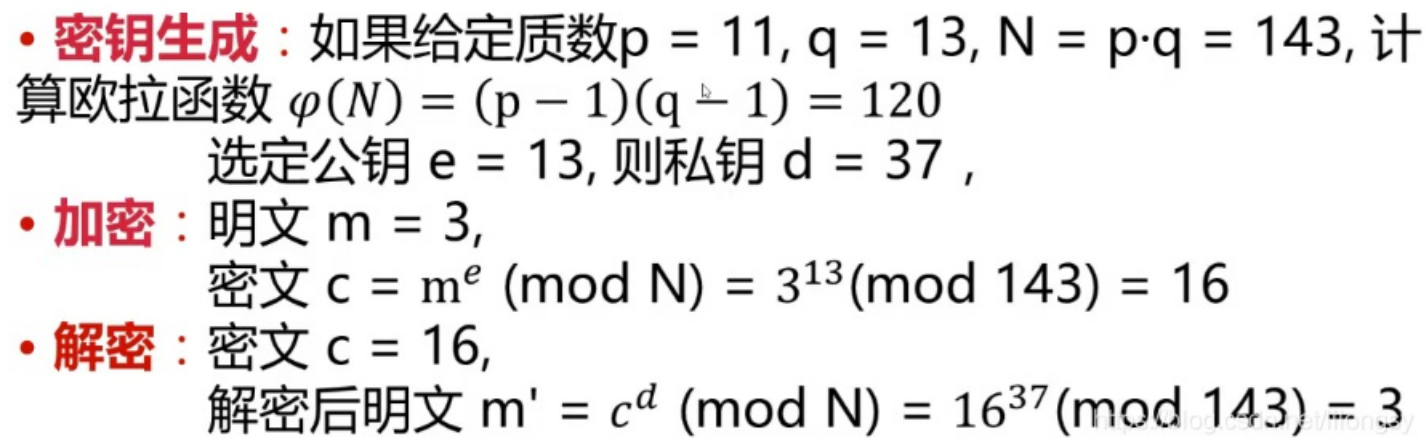
```
print("Guessing next 32000 random bits success rate: {}%"
      .format(sum([random.getrandbits(32) == cracker.predict_getrandbits(32) for x in range(1000)]) / 10))
```

RSA

密钥的生成



举例：



RSA安全性

理论上基于大数分解的困难性，即分解模数N的困难性
但随着计算机计算能力的发展，为了确保模数N不被分解，N需要取得越来越大。
工具：yafu
在线工具：factordb

RSA攻击套路

首先，我这边就不放冗长的百度百科的东西了，我概括一下我自己对RSA的看法。
RSA是一种算法，并且广泛应用于现代，用于保密通信。
RSA算法涉及三个参数,n,e,d，其中分为私钥和公钥，私钥是n,d，公钥是n,e
n是两个素数的乘积，一般这两个素数在RSA中用字母p,q表示
e是一个素数
d是e模 `varphi(n)` 的逆元，CTF的角度看就是，d是由e,p,q可以求解出的
一般CTF就是把我们想要获得的flag作为明文，RSA中表示为m。然后通过RSA加密，得到密文，RSA中表示为C。
加密过程 `c=m^e mod n`

```
c=pow(m,e,n)
```

解密过程`m=c^d mod n`

```
m=pow(c,d,n)
```

求解私钥d

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

一般来说，n，e是公开的，但是由于n一般是两个大素数的乘积，所以我们很难求解出d，所以RSA加密就是利用现代无法快速实现大素数的分解，所存在的一种安全的非对称加密。

基础RSA加密

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

# 明文m
msg = 'flag is :testflag'
# 转成16进制整数
hex_msg=int(msg.encode("hex"),16)
print(hex_msg)
# 素数p
p=getPrime(100)
# 素数q
q=getPrime(100)
# n是两个素数的乘积
n=p*q
# 公钥是n,e，e是一个素数。私钥是n,d
e=0x10001
#
phi=(p-1)*(q-1)
# d是e模 varphi(n) 的逆元，CTF的角度看就是，d是由e,p,q可以求解出的
d=gmpy2.invert(e,phi)
print("d=",hex(d))
# 密文c
c=pow(hex_msg,e,n)
print("e=",hex(e))
print("n=",hex(n))
print("c=",hex(c))
```

```
348528637939318975470908238077546712518l5
('d=', '0xe0cb53bac57348a423f43f2aed1bf0f503688c90a0cff19c1')
('e=', '0x10001')
('n=', '0xde23e693f412952e8e057270d0f14f56a0df99d598586e8edfL')
('c=', '0x644078982d648df7097dd388a29f67a8567de4c6ff196fbc6eL')
```

基础RSA解密

```
#!/usr/bin/env python
# -*- coding:utf-8 -*-
import binascii
import gmpy2
# 两个素数的乘积n
n=0x80b32f2ce68da974f25310a23144977d76732fa78fa29fdcbf
#这边我用yafu分解了n
p=780900790334269659443297956843
q=1034526559407993507734818408829
# 公钥e
e=0x10001
# 密文c
c=0x534280240c65bb1104ce3000bc8181363806e7173418d15762

phi=(p-1)*(q-1)
# 逆元d
d=gmpy2.invert(e,phi)
# 明文m
m=pow(c,d,n)
print(hex(m))
print(binascii.unhexlify(hex(m)[2:].strip("L")))
```

```
0x666c6167206973203a74657374666c6167
flag is :testflag
```

RSA因子p&q不当分解

p和q相差过大或过小，如上例。

利用条件

因为n=p*q
其中若p和q的值相差较小，或者较大，都会造成n更容易分解的结果
例如出题如下

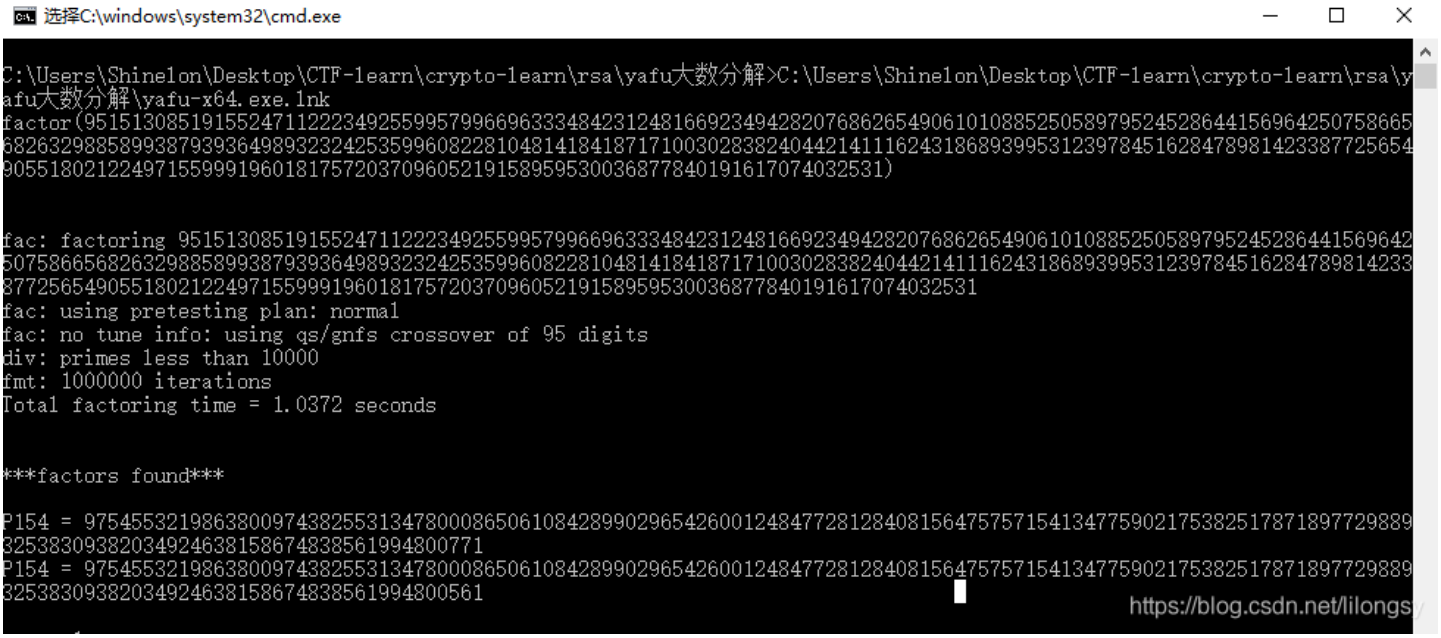
```
p=getPrime(512)
q=gmpy2.next_prime(p)
n=p*q
```

因为p和q十分接近，所以可以使用yafu直接分解

yafu分解
使用

```
factor(*)
```

括号中为要分解的数



在线网站分解

http://factordb.com/

通过在此类网站上查询n，如果可以分解或者之前分解成功过，那么可以直接得到p和q

举例

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

flag = 'flag{**********************}'
m=bytes_to_long(flag.encode())
p=getPrime(512)
# 临近q
q=gmpy2.next_prime(p)
n=p*q
e=0x10001
phi=(p-1)*(q-1)
d=gmpy2.invert(e,phi)
c=pow(m,e,n)
print("e=",hex(e))
print("n=",hex(n))
print("c=",hex(c))

#e= 0x10001
#n= 0x96d562f7100cff53f608b6ba1552d58727fbf3fd163726fa65c5d6cf777bdabec3b549242897055caaea5c0a0bb6a30bab73a57b528265ae6045cae800c17978993bb6d4d81e4d995a6ffb92ee10ed606871037b0ddace3db81a1537cb6f16c322160fa36f5903eb1aa84f0ee46bfacde03a2cac18504df552648b407852040f
#c= 0x75c64aa529f2c6987653122d1d5e20d37bdb3c86049da0cac13b93279409a8a08c15ed09e3b0f4f2d854e64a167438ff36ef6aa0821542f31c9543db7d85e7aefde04f47e0403d8e8f74e83acb7b3a7b5b92fd9f12237d48ed865317ebd7888cd895f7959233dd3190965e7426a68d49bd98206712c168b144423351742f94f3
```

exp.py

```
e= 0x10001
n= 0x96d562f7100cff53f608b6ba1552d58727fbf3fd163726fa65c5d6cf777bdabec3b549242897055caaea5c0a0bb6a30bab73a57b528265ae6045cae800c17978993bb6d4d81e4d995a6ffb92ee10ed606871037b0ddace3db81a1537cb6f16c322160fa36f5903eb1aa84f0ee46bfacde03a2cac18504df552648b407852040f
c= 0x75c64aa529f2c6987653122d1d5e20d37bdb3c86049da0cac13b93279409a8a08c15ed09e3b0f4f2d854e64a167438ff36ef6aa0821542f31c9543db7d85e7aefde04f47e0403d8e8f74e83acb7b3a7b5b92fd9f12237d48ed865317ebd7888cd895f7959233dd3190965e7426a68d49bd98206712c168b144423351742f94f3

p = 10291691539956320954972581754251612994384170725799086560166488710492236835931157131382616562400233051981075764700669983013064614944432405231738947852601201
q = 10291691539956320954972581754251612994384170725799086560166488710492236835931157131382616562400233051981075764700669983013064614944432405231738947852600191
print(p*q==n)


import gmpy2
import binascii

phi=(p-1)*(q-1)
d=gmpy2.invert(e,phi)
m=pow(c,d,n)
print(binascii.unhexlify(hex(m)[2:].strip("L")))

'''
True
flag{a2d10a3211b415832791a6bc6031f9ab}
'''
```

公约数分解n

利用条件

当题目给的多对公钥n是公用了一个素数因子的时候，可以尝试公约数分解
出题一般如下

```
p1=getPrime(512)
p2=getPrime(512)
q=getPrime(512)
n1=p1*q
n2=p2*q
```

所以当题目给了多个n，并且发现n无法分解，可以尝试是否有公约数。

欧几里得辗转相除法
求公约数可以使用欧几里得辗转相除法，实现python脚本如下

```
def gcd(a, b):   #求最大公约数
    if a < b:
        a, b = b, a
    while b != 0:
        temp = a % b
        a = b
        b = temp
    return a
```

举例

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

flag = 'flag{******************}'
m=bytes_to_long(flag.encode())
p=getPrime(512)
q=getPrime(512)
p2=getPrime(512)
# n1和n2有公约数q
n1=p*q
n2=p2*q
e=0x10001
phi=(p-1)*(q-1)
d=gmpy2.invert(e,phi)
c=pow(m,e,n1)
print("e=",e)
print("n1=",n1)
print("n2=",n2)
print("c=",c)
'''
e= 65537
n1= 115205766471059781270815604837009314657798597158913292644119663583719125777824041586752032035243666254025203
786188035607022471569797983564319842982498755649057896881618966000477214120345088836045599502515494252992934633 3
594368501353986325871824992478369720878058647583102050676962929793602474912689354457607 47
n2= 139151110513318209083176529711756450516319765206948018486876906602717789940417835912041508342627710084011340
627446281100656028403764567565659197999643325780583386127242149024266190288969045344868291613275890848074094845 2
356361453676876006669353291506106851257523764026489457668775559606661136295848701518280 91
c= 377843620059121787212639737423041435962484038029046702055645881938881694646890523528871330378866971680827572 6
375558416654559821204839690842584386162417830984972069611672292764293618773342659322757998848955711369813343380 4
364722953380850950658119086544556004805582948121618951251321352586046056523556934859706
'''
```

exp.py

```
e= 65537
n1= 11520576647105978127081560483700931465779859715891329264411966358371912577782404158675203203524366625402520378618803560702247156979798356431984298249875564905789688161896600047721412034508883604559950251549425299293463335943685013539863258718249924783697208780586475831020506769629297936024749126893544576074 7
n2= 13915111051331820908317652971175645051631976520694801848687690660271778994041783591204150834262771008401134062744628110065602840376456756565919799964332578058338612724214902426619028896904534486829161327589084807409484523563614536768760066693532915061068512575237640264894576687755596066611362958487015182809 1
c= 37784362005912178721263973742304143596248403802904670205564588193888169464689052352887133037886697168082757263755584166545598212048396908425843861624178309849720696116722927642936187733426593227579988489557113698133433804364722953380850950658119086544556004805582948121618951251321352586046056523556934859706
import gmpy2
from Crypto.Util.number import *
q=gmpy2.gcd(n1,n2)
print(q)
p=n1//q
print(p*q==n1)

phi=(p-1)*(q-1)
d=gmpy2.invert(e,phi)
m=pow(c,d,n1)
print(long_to_bytes(m))

'''
1104368346620667438125999811855861489494403489409788719689700781877112537002957392473979449247462624726868288382918288824162034830932365309636119123835633 3
True
flag{a2d10a3211b415832791a6bc6031f9ab}
'''
```

模数分解

利用条件

已知e,d,n求p,q、

dp&dq泄露

利用条件

首先了解一下什么是dp、dq

```
dp=d%(p-1)
dq=d%(q-1)
```

这种参数是为了让解密的时候更快速产生的。

```
('p=', '0xf85d730bbf09033a75379e58a8465f8048b8516f8105ce2879ce774241305b6eb4ea506b61eb7e376d4fcd425c76e80cb748ebfaf3a852b5cf3119f028cc5971L')
('q=', '0xc1f34b4f826f91c5d68c5751c9af830bc770467a68699991be6e847c29c13170110ccd5e855710950abab2694b6ac730141152758acbeca0c5a51889cbe84d57L')
('dp=', '0xf7b885a246a59fa1b3fe88a2971cb1ee8b19c4a7f9c1a791b9845471320220803854a967a1a03820e297c0fc1aabc2e1c40228d50228766ebebc93c97577f511')
('dq=', '0x865fe807b8595067ff93d053cc269be6a75134a34e800b741cba39744501a31cffd31cdea6078267a0bd652aeaa39a49c73d9121fafdfa7e1131a764a12fdb95')
('c=', '0xae05e0c34e2ba4ca3536987cc2cfc3f1f7f53190164d0ac50b44832f0e7224c6fdeebd2c91e3991e7d179c26b1b997295dc9724925ba431f527fba212703a0d14a34ce133661ae0b6001ee326303d6ccdc27dbd94e0987fae25a84f197c1535bdac9094bfb3846b7ca696b2e5082bea7bff804da275772ca05dd51b185a4fc30L')
```

解题代码：

```
InvQ=gmpy2.invert(q,p)
mp=pow(c,dp,p)
mq=pow(c,dq,q)
m=(((mp-mq)*InvQ)%p)*q+mq
print '{:x}'.format(m).decode('hex')
```

举例

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

flag = 'flag{********************************}'
m=bytes_to_long(flag.encode())
p=getPrime(512)
q=getPrime(512)
n=p*q
e=getPrime(20)
phi=(p-1)*(q-1)
d=gmpy2.invert(e,phi)
dp=d%(p-1)
dq=d%(q-1)
c=pow(m,e,n)
print("n=",n)
print("p=",p)
print("q=",q)
print("dp=",dp)
print("dq=",dq)
print("c=",c)
'''
n= 11861895428761526969590487599548954851144494026762276709404773963449464635880930443397817349568767837040645870286053634278580375323470254621949466553586447465230186228459169067317676904363037289907588383460654517458634872099367807889565414913667602320696302121522612801475778338307678641335960486154916263674 7
p= 12244425890527037559457450758199625345224440148602500121947513405940248134857180032689144921510099766782376592176091852775339372003558472079284757727362571
q= 96875880787016259949674844763214315658568404114397889766140479022609351191706061285291846933042516632874184343251338149929451496703720826420157929849362 57
dp= 11661765341055037028213110243140762441891859947132976341835485861002039690412298548398000165827316261292160830963662695604031845480131372329938955617008307
dq= 53309721089622308195179253527537602471893425733693909836437545623822246592269498167946597155456301715573289750208735188799188891499736988869778988564761 83
c= 12374013610995935975987144573885493212459411904794697197903023715805777590324318665512805085245267803654557700689811822171219122398629103950576556025535606948328799420133553234348610960343249046259660281472602396406991408998707805300684186709474247145123999521260863968647240075574632249474210031448000004668
'''
```

exp.py

```
import gmpy2
import binascii
def decrypt(dp,dq,p,q,c):
    InvQ = gmpy2.invert(q,p)
    mp = pow(c,dp,p)
    mq = pow(c,dq,q)
    m=(((mp-mq)*InvQ)%p)*q+mq
    print (binascii.unhexlify(hex(m)[2:]))

n= 11861895428761526969590487599548954851144494026762276709404773963449464635880930443397817349568767837040645870286053634278580375323470254621949466553586447465230186228459169067317676904363037289907588383460654517458634872099367807889565414913667602320696302121522612801475778338307678641335960486154916263674 7
p= 12244425890527037559457450758199625345224440148602500121947513405940248134857180032689144921510099767782376592176091852775339372003558472079284757727362571
q= 96875880787016259949674844763214315658568404114397889766140479022609351191706061285291846933042516632874184343251338149929451496703720826420157929849362 57
dp= 11661765341055037028213110243140762441891859947132976341835485861002039690412298548398000165827316261292160830963662695604031845480131372329938955617008307
dq= 53309721089622308195179253527537602471893425733693909836437545623822246592269498167946597155456301715573289750208735188799188891499736988869778988564761 83
c= 12374013610995935975987144573885493212459411904794697197903023715805777590324318665512805085245267803654557700689811822171219122398629103950576556025535606948328799420133553234348610960343249046259660281472602396406991408998707805300684186709474247145123999521260863968647240075574632249474210031448000004668

decrypt(dp,dq,p,q,c)
# flag{a2d10a3211b415832791a6bc6031f9ab}
```

dp泄露

利用条件

假设题目给出公钥n,e以及dp

```
('dp=', '0x7f1344a0b8d2858492aaf88d692b32c23ef0d2745595bc5fe68de384b61c03e8fd054232f2986f8b279a0105b7bee85f74378c7f5f35c3fd505e214c0738e1d9')
('n=', '0x5eee1b4b4f17912274b7427d8dc0c274dc96baa72e43da36ff39d452ff6f2ef0dc6bf7eb9bdab899a6bb718c070687feff517fcf5377435c56c248ad88caddad6a9cefa0ca9182daffcc6e48451d481f37e6520be384bedb221465ec7c95e2434bf76568ef81e988039829a2db43572e2fe57e5be0dc5d94d45361e96e14bd65L')
('e=', '0x10001')
('c=', '0x510fd8c3f6e21dfc0764a352a2c7ff1e604e1681a3867480a070a480f722e2f4a63ca3d7a92b862955ab4be76cde43b51576a128fba49348af7a6e34b335cfdbda8e882925b20503762edf530d6cd765bfa951886e192b1e9aeed61c0ce50d55d11e343c78bb617d8a0adb7b4cf3b913ee85437191f1136e35b94078e68bee8dL')
```

给出密文要求解明文
我们可以通过n，e，dp求解私钥d。
公式推导参考简书
https://www.jianshu.com/p/74270dc7a14b
首先dp是

```
dp=d%(p-1)
```

以下推导过程如果有问题欢迎指正
现在我们可以知道的是

$$c \equiv m^e \bmod n$$
$$m \equiv c^d \bmod n$$
$$\varphi(n)=(p-1)*(q-1)$$
$$d*e \equiv 1 \bmod \varphi(n)$$
$$dp \equiv d \bmod (p-1)$$

由上式可以得到

$$dp*e \equiv d*e \bmod (p-1)$$

因此可以得到

$$d*e=k*(p-1)+dp*ed*e \equiv 1 \bmod \varphi(n)$$

我们将式1带入式2可以得到

$$k*(p-1)+dp*e \equiv 1 \bmod (p-1)*(q-1)$$

故此可以得到

$$k2*(p-1)*(q-1)+1=k1*(p-1)+dp*e$$

变换一下

$$(p-1)*[k2*(q-1)-k1]+1=dp*e$$

因为

$$dp<p-1$$

可以得到

$$e>k2*(q-1)-k1$$

我们假设

$$x=k2*(q-1)-k1$$

可以得到x的范围为

$$(0,e)$$

因此有

$$x*(p-1)+1=dp*e$$

那么我们可以遍历

$$x \in (0,e)$$

求出p-1，求的方法也很简单，遍历65537种可能，其中肯定有一个p可以被n整除那么求出p和q，即可利用

$$\varphi(n)=(p-1)*(q-1)d*e \equiv 1 \bmod \varphi(n)$$

推出

$$d \equiv 1*e-1 \bmod \varphi(n)$$

注：这里的-1为逆元，不是倒数的那个-1

公式的python实现

求解私钥d脚本如下

```
def getd(n,e,dp):
    for i in range(1,e):
        if (dp*e-1)%i == 0:
            if n%(((dp*e-1)/i)+1)==0:
                p=((dp*e-1)/i)+1
                q=n/(((dp*e-1)/i)+1)
                phi = (p-1)*(q-1)
                d = gmpy2.invert(e,phi)%phi
                return d
```

举例

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

flag = 'flag{****************************}'
m=bytes_to_long(flag.encode())
p=getPrime(512)
q=getPrime(512)
n=p*q
e=0x10001
phi=(p-1)*(q-1)
d=gmpy2.invert(e,phi)
dp=d%(p-1)
c=pow(m,e,n)
print("e=",e)
print("n=",n)
print("dp=",dp)
print("c=",c)
'''
e= 65537
n= 92828237009545230120118069089223514912716896165610308064504460604082624853938431962923949334864864436264408433256288068202484225961256898972743676389188622775737958995025654351705278456167733907608412584542016121478865285024454893448016017969632116442841446786562704173003893111037687107024681444848078613927
dp= 40822186823884545208438781607977449078424954470682480063797875283261148110471972689520395815427737818545596079446588344874672996755493986622563898368219 09
c= 14490533356014180733497751921539678090452759083476295303899740842911168177354564459129698267996626531829194777130342559029232479939335746334521688770129350369504991860912181278314347862204291618819118798920468232748953919317971699169539017795380789372322430009718018665278749598988666351269866046298409588225
'''
```

exp.py

```
import gmpy2
import binascii

def getd(n,e,dp):
    for i in range(1,e):
        if (dp*e-1)%i == 0:
            if n%(((dp*e-1)//i)+1)==0:
                p=((dp*e-1)//i)+1
                q=n//(((dp*e-1)//i)+1)
                phi = (p-1)*(q-1)
                d = gmpy2.invert(e,phi)%phi
                return d



e= 65537
n= 92828237009545230120118069089223514912716896165610308064504460604082624853938431962923949334864864436264408433256288068202484225961256898972743676389188622775737958995025654351705278456167733907608412584542016121478865285024454893448016017969632116442841446786562704173003893111037687107024681444848078613927
dp= 40822186823884545208438781607977449078424954470682480063797875283261148110471972689520395815427737818545596079446588344874672996755493986622563898368219 09
c= 14490533356014180733497751921539678090452759083476295303899740842911168177354564459129698267996626531829194777130342559029232479939335746334521688770129350369504991860912181278314347862204291618819118798920468232748953919317971699169539017795380789372322430009718018665278749598988666351269866046298409588225



d=getd(n,e,dp)
m=pow(c,d,n)
print (binascii.unhexlify(hex(m)[2:]))
# flag{a2d10a3211b415832791a6bc6031f9ab}
```

e与φ(n)不互素

利用条件

```
('n1=', '0xbf510b8e2b169fbce366eb15a4f6c71b370f02f2108c7feb482234a386185bce1a740fa6498e04edbdf2a639e320619d9f39d3e740ebaf578af0426bc3e851001a1d599108a08725347f6680a7f5581a32d91505023701872c3df723e8de9f201d3b17059bebff944b915045870d757eb6d6d009eb4561cc7e4b89968e4433a9L')
('e1=', '0x15d6439c6')
('c1=', '0x43e5cc4c99c3040aef2ccb0d4c45266f6b75cd7f9f1be105766689283f0886061c9cd52ac2b2b6c1b7d250c2079f354ca9b988db5556336201f3b5e489916b3b60b80c34bef8f608d7471fafaf14bee421b60630f42c5cc813356e786ff10e5efa334b8a73b7ea06afa6043f33be6a31010d306ba60516243add65c183da843aL')
('n2=', '0xba85d38d1bfc3fb281927c9246b5b771ac3344ca9fe1c2d9c793a886bffb5c84558f4a578cd5ba9e777a4e08f66d0cabe05b9aa2ae8d075778b5fbfff318a7f9b6f22e2eff6f79d8c1148941b3974f3e83a4a4f1520ad42336eddc572ec7ea04766eb798b2f1b1b52009b3eeea7741b2c55e3c7c11c5cf6a4e204c6b0d312f49L')
('e2=', '0x2c09848c6')
('c2=', '0x79ec6350649377f69b475eca83a7d9d5356a1d62e29933e9c8e2b19b4b23626a581037aba3be6d7f73d5bed049350e41c1ed4cdc3e10ee34ec576ef3449be2f7d930c759612e1c23c4db71d0e5185a80b548031e3857dd93eca4af017fcd25895fcc4e8a2b36c1dd36b8cd9cc9200e2879f025928fe346e2cfae5200e66de6ccL')
```

首先用公约数分解可以分解得到n1、n2的因子
但是发现e和φ(n)是不互为素数的，所以我们无法求出私钥d。

解题公式推导

```
gcd(e1,(p-1)*(q1-1))
gcd(e2,(p-1)*(q2-1))
```

得到结果为79858
也就是说，e和φ(n)不互素且具有公约数79858
公式推导过程参考博客
https://blog.csdn.net/chenzzhenguo/article/details/94339659

举例

exp.py

```
#!/usr/bin/env python
# -*- coding:utf-8 -*-
import gmpy2
import binascii


def gcd(a, b):
    if a < b:
        a, b = b, a
    while b != 0:
        temp = a % b
        a = b
        b = temp
    return a

n1=0xbf510b8e2b169fbce366eb15a4f6c71b370f02f2108c7feb482234a386185bce1a740fa6498e04edbdf2a639e320619d9f39d3e740ebaf578af0426bc3e851001a1d599108a08725347f6680a7f5581a32d91505023701872c3df723e8de9f201d3b17059bebff944b915045870d757eb6d6d009eb4561cc7e4b89968e4433a9
n2=0xba85d38d1bfc3fb281927c9246b5b771ac3344ca9fe1c2d9c793a886bffb5c84558f4a578cd5ba9e777a4e08f66d0cabe05b9aa2ae8d075778b5fbfff318a7f9b6f22e2eff6f79d8c1148941b3974f3e83a4a4f1520ad42336eddc572ec7ea04766eb798b2f1b1b52009b3eeea7741b2c55e3c7c11c5cf6a4e204c6b0d312f49

p=gcd(n1,n2)
q1=n1//p
q2=n2//p



c1=0x43e5cc4c99c3040aef2ccb0d4c45266f6b75cd7f9f1be105766689283f0886061c9cd52ac2b2b6c1b7d250c2079f354ca9b988db5556336201f3b5e489916b3b60b80c34bef8f608d7471fafaf14bee421b60630f42c5cc813356e786ff10e5efa334b8a73b7ea06afa6043f33be6a31010d306ba60516243add65c183da843a
c2=0x79ec6350649377f69b475eca83a7d9d5356a1d62e29933e9c8e2b19b4b23626a581037aba3be6d7f73d5bed049350e41c1ed4cdc3e10ee34ec576ef3449be2f7d930c759612e1c23c4db71d0e5185a80b548031e3857dd93eca4af017fcd25895fcc4e8a2b36c1dd36b8cd9cc9200e2879f025928fe346e2cfae5200e66de6cc
e1 =0x15d6439c6
e2 =0x2c09848c6

#print(gcd(e1,(p-1)*(q1-1)))
#print(gcd(e2,(p-1)*(q2-1)))


e1=e1//gcd(e1,(p-1)*(q1-1))
e2=e2//gcd(e2,(p-1)*(q2-1))


phi1=(p-1)*(q1-1);phi2=(p-1)*(q2-1)
d1=gmpy2.invert(e1,phi1)
d2=gmpy2.invert(e2,phi2)
f1=pow(c1,d1,n1)
f2=pow(c2,d2,n2)
```

```
def GCRT(mi, ai):
    curm, cura = mi[0], ai[0]
    for (m, a) in zip(mi[1:], ai[1:]):
        d = gmpy2.gcd(curm, m)
        c = a - cura
        K = c // d * gmpy2.invert(curm // d, m // d)
        cura += curm * K
        curm = curm * m // d
        cura %= curm
    return (cura % curm, curm)


f3,lcm = GCRT([n1,n2],[f1,f2])
n3=q1*q2
c3=f3%n3
phi3=(q1-1)*(q2-1)

d3=gmpy2.invert(39929,phi3)#39929是79858//gcd((q1-1)*(q2-1),79858) 因为新的e和φ(n)还是有公因数2
m3=pow(c3,d3,n3)

if gmpy2.iroot(m3,2)[1] == 1:
    flag=gmpy2.iroot(m3,2)[0]
    print(binascii.unhexlify(hex(flag)[2:].strip("L")))
```

公钥n由多个素数因子组成

利用条件

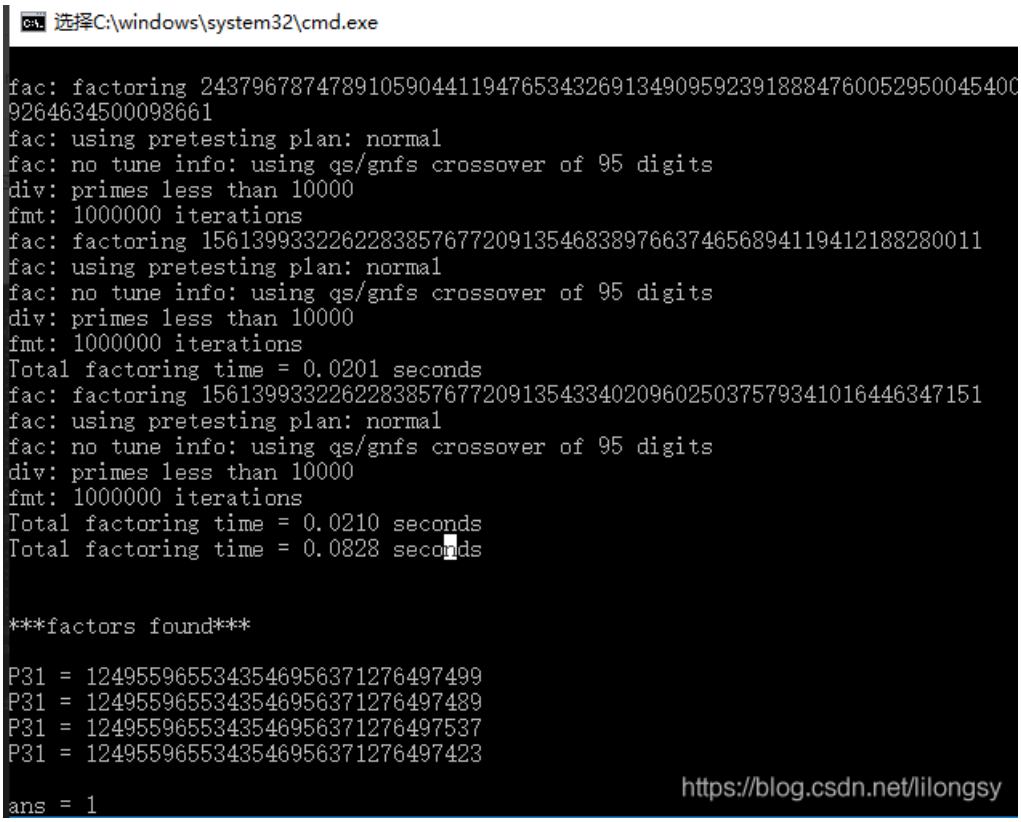
题目如下

```
('n=', '0xf1b234e8a03408df4868015d654dcb931f038ef4fc0be8658c9b951ee6c60d23689a1bfb151e74df0910fa1cf8a542282a65')
('e=', '0x10001')
('c=', '0x22fda6137013bac19754f78e8d9658498017f05a4b0814f2af97dc2c60fdc433d2949ea27b13337961ef3c4cf27452ad3c95')
```

因为这题的公钥n是由四个素数相乘得来的，
其中四个素数的值相差较小，或者较大，都会造成n更容易分解的结果
例如出题如下

```
p=getPrime(100)
q=gmpy2.next_prime(p)
r=gmpy2.next_prime(q)
s=gmpy2.next_prime(r)
n=p*q*r*s
```

因为p、q、r、s十分接近，所以可以使用yafu直接分解




```
import binascii
import gmpy2
p=1249559655343546956371276497499
q=1249559655343546956371276497489
r=1249559655343546956371276497537
s=1249559655343546956371276497423
e=0x10001
c=0x22fda6137013bac19754f78e8d9658498017f05a4b0814f2af97dc2c60fdc433d2949ea27b13337961ef3c4cf27452ad3c95
n=p*q*r*s

phi=(p-1)*(q-1)*(r-1)*(s-1)
d=gmpy2.invert(e,phi)
m=pow(c,d,n)
print(binascii.unhexlify(hex(m)[2:].strip("L")))
# flag is:testflag
```

举例

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

flag = 'flag{****************************}'
m=bytes_to_long(flag.encode())
p=getPrime(128)
q=gmpy2.next_prime(p)
r=getPrime(128)
n=p*q*r
e=0x10001
c=pow(m,e,n)
print("e=",e)
print("n=",n)
print("c=",c)
print("r=",r)
'''
e= 65537
n= 154579141850023961882727934818143548039010787336911317838571414210403431229044717875559431383208764123344395638704 23
c= 80365257771304431843934550470343339395940517417080511332601560169391034862388788918769069120262317684958183801584 85
r= 291772877613383069932935701252073023153
'''
```

exp.py

```
e= 65537
n= 154579141850023961882727934818143548039010787336911317838571414210403431229044717875559431383208764123344395638704 23
c= 80365257771304431843934550470343339395940517417080511332601560169391034862388788918769069120262317684958183801584 85
r= 291772877613383069932935701252073023153
import gmpy2
from Crypto.Util.number import *

pq=n//r
print(pq)
p = 230172262495731574542243822462679932863
q = 230172262495731574542243822462679932857
phi=(p-1)*(q-1)*(r-1)
d=gmpy2.invert(e,phi)
m=pow(c,d,n)
print(long_to_bytes(m))

'''
529792704224039599601695443416551449532187803887425978074923021113341077795 91
flag{a2d10a3211b415832791a6bc6031f9ab}
'''
```

小明文攻击

利用条件

明文过小，导致明文的e次方仍然小于n明文的三次方虽然比n大，但是大不了多少
小明文攻击是基于低加密指数的，主要分成两种情况。

明文过小，导致明文的e次方仍然小于n

```
('n=', '0xad03794ef170d81aad370dccb7b92af7d174c10e0ae9ddc99b7dc5f93af6c65b51cc9c40941b002c7633caf8cd50e1b73aa942c8488d46c0032064306de388151814982b6d35b4e2a62dd647f527b31b4f826c36848dc52999574a8694460e1b59b4e96bda1341d3ba5f991f0000a56004d47681ecfd37a5e64bd198617f8dadL')
('e=', '0x3')
('c=', '0x10652cdf6f422470ea251f77L')
```

这种情况直接对密文e次开方，即可得到明文

解题脚本

```
import binascii
import gmpy2
n=0xad03794ef170d81aad370dccb7b92af7d174c10e0ae9ddc99b7dc5f93af6c65b51cc9c40941b002c7633caf8cd50e1b73aa942c8488d46c0032064306de388151814982b6d35b4e2a62dd647f527b31b4f826c36848dc52999574a8694460e1b59b4e96bda1341d3ba5f991f0000a56004d47681ecfd37a5e64bd198617f8dad
e=0x3
c=0x10652cdf6f422470ea251f77

m=gmpy2.iroot(c, 3)[0]
print(binascii.unhexlify(hex(m)[2:].strip("L")))
```

明文的三次方虽然比n大，但是大不了多少

```
('n=', '0x9683f5f8073b6cd9df96ee4dbe6629c7965e1edd2854afa113d80c44f5dfcf030a18c1b2ff40575fe8e222230d7bb5b6dd8c419c9d4bca1a7e84440a2a87f691e2c0c76caaab61492db143a61132f584ba874a98363c23e93218ac83d1dd715db6711009ceda2a31820bbacaf1b6171bbaa68d1be76fe986e4b4c1b66d10af25L')
('e=', '0x3')
('c=', '0x8541ee560f77d8fe536d48eab425b0505e86178e6ffefa1b0c37ccbfc6cb5f9a7727baeb3916356d6fce3205cd4e586a1cc407703b3f709e2011d7b66eaaeea9e381e595b4d515c433682eb3906d9870fadbffd0695c0168aa26447f7a049c260456f51e937ce75b74e5c3c2bd7709b981898016a3a18f15ae99763ff40805aaL')
```

爆破即可，每次加上一个n

```
i = 0
while 1:
    res = iroot(c+i*n,3)
    if(res[1] == True):
        print res
        break
    print "i="+str(i)
    i = i+1
```

完整脚本：

```
import binascii
import gmpy2

n=0x9683f5f8073b6cd9df96ee4dbe6629c7965e1edd2854afa113d80c44f5dfcf030a18c1b2ff40575fe8e222230d7bb5b6dd8c419c9d4b
ca1a7e84440a2a87f691e2c0c76caaab61492db143a61132f584ba874a98363c23e93218ac83d1dd715db6711009ceda2a31820bbacaf1b6
171bbaa68d1be76fe986e4b4c1b66d10af25
e=0x3
c=0x8541ee560f77d8fe536d48eab425b0505e86178e6ffefa1b0c37ccbfc6cb5f9a7727baeb3916356d6fce3205cd4e586a1cc407703b3f
709e2011d7b66eaaeea9e381e595b4d515c433682eb3906d9870fadbffd0695c0168aa26447f7a049c260456f51e937ce75b74e5c3c2bd77
09b981898016a3a18f15ae99763ff40805aa

i = 0
while 1:
    res = gmpy2.iroot(c+i*n,3)
    if(res[1] == True):
        m=res[0]
        print(binascii.unhexlify(hex(m)[2:].strip("L")))
        break
    print "i="+str(i)
    i = i+1
```

举例

```
#n:  0x52d483c27cd806550fbe0e37a61af2e7cf5e0efb723dfc81174c918a27627779b21fa3c851e9e94188eaee3d5cd6f752406a43fbe
cb53e80836ff1e185d3ccd7782ea846c2e91a7b0808986666e0bdadbfb7bdd65670a589a4d2478e9adcafe97c6ee23614bcb2ecc23580f4d
2e3cc1ecfec25c50da4bc754dde6c8bfd8d1fc16956c74d8e9196046a01dc9f3024e11461c294f29d7421140732fedacac97b8fe50999117
d27943c953f18c4ff4f8c258d839764078d4b6ef6e8591e0ff5563b31a39e6374d0d41c8c46921c25e5904a817ef8e39e5c9b71225a83269
693e0b7e3218fc5e5a1e8412ba16e588b3d6ac536dce39fcdfce81eec79979ea6872793L
#e:  0x3
#c:0x10652cdfaa6b63f6d7bd1109da08181e500e5643f5b240a9024bfa84d5f2cac9310562978347bb232d63e7289283871efab83d84ff5
a7b64a94a79d34cfbd4ef121723ba1f663e514f83f6f01492b4e13e1bb4296d96ea5a353d3bf2edd2f449c03c4a3e995237985a596908adc
741f32365
so,how to get the message?
```

exp.py

```
e= 65537
n= 1545791418500239618827279348181435480390107873369113178385714142104034312290447178755594313832087641233443956
3870423
c= 8036525777130443184393455047034333939594051741708051133260156016939103486238878891876906912026231768495818380
158485
r= 291772877613383069932935701252073023153
import gmpy2
from Crypto.Util.number import *

pq=n//r
print(pq)
p = 230172262495731574542243822462679932863
q = 230172262495731574542243822462679932857
phi=(p-1)*(q-1)*(r-1)
d=gmpy2.invert(e,phi)
m=pow(c,d,n)
print(long_to_bytes(m))

'''
529792704224039599601695443416551449532187803887425978074923021113341077795 91
flag{a2d10a3211b415832791a6bc6031f9ab}
'''
```

低加密指数广播攻击

利用条件

如果选取的加密指数较低，并且使用了相同的加密指数给一个接受者的群发送相同的信息，那么可以进行广播攻击得到明文。
这个识别起来比较简单，一般来说都是给了三组加密的参数和明密文，其中题目很明确地能告诉你这三组的明文都是一样的，并且e都取了一个较小的数字。

举例

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

flag = 'flag{********************************}'
m=bytes_to_long(flag.encode())
def getN():
    p=getPrime(256)
    q=getPrime(256)
    n=p*q
    return n
e=0x3
n1=getN()
c1=pow(m,e,n1)
n2=getN()
c2=pow(m,e,n2)
n3=getN()
c3=pow(m,e,n3)
print("n=",[n1,n2,n3])
print("c=",[c1,c2,c3])

'''

n= [556700048622635325452467345435714707611807712227578948055523858455233252645918164156785842958722258768334590131046814078242085893170957047656269895311280 1, 698288736364535485654086703562944178154999970000636340394095137648551746050682203381281226317966387584373795343005043645329833610440430418396416270261286 3, 556962878052152609213675904777444880828958814512719121395659390233635317255990528260238371625850093502105639391797116430353994126839515995365276861430491 1]
c= [411252354488348083090340345251751682570307890267092640598477476202423598076780553894746691419395083548846960875677856969219096790173020769052530024768281 5, 418129247203627089759356870893141378788848862127403074113054943446797640547655513089072234631954939680018382730545201979611987304427913856683726481462709 8, 385743427078862136151734924727734783195302124141498443077058622042421411070099961578587946277158495274404935787575375420900643902808087062601234246714877 3]

'''
```

exp.py

```
import binascii,gmpy2

n= [556700048622635325452467345435714707611807712227578948055523858455233252645918164156785842958722258768334590
131046814078242085893170957047656269895311280l, 698288736364535485654086703562944178154999970000636340394095137 6
485517460506822033812812263179663875843737953430050436453298336104404304183964162702612863, 5569628780521526092 1
367590477744488082895881451271912139565939023363531725599052826023837162585009350210563939179711643035399412683 9
5159953652768614304911]
c= [411252354488348083090340345251751682570307890267092640598477476202423598076780553894746691419395083548846960
875677856969219096790173020769052530024768281 5, 418129247203627089759356870893141378788848862127403074113054943 4
467976405476555130890722346319549396800183827305452019796119873044279138566837264814627098, 385743427078862136 15
173492472773478319530212414149844307705862204242141107009961578587946277158495274404935787575375420900643902808
0870626012342467148773]
from functools import reduce
def CRT(mi, ai):
    assert(reduce(gmpy2.gcd,mi)==1)
    assert (isinstance(mi, list) and isinstance(ai, list))
    M = reduce(lambda x, y: x * y, mi)
    ai_ti_Mi = [a * (M // m) * gmpy2.invert(M // m, m) for (m, a) in zip(mi, ai)]
    return reduce(lambda x, y: x + y, ai_ti_Mi) % M
e=0x3
m=gmpy2.iroot(CRT(n, c), e)[0]
print(binascii.unhexlify(hex(m)[2:].strip("L")))

# flag{a2d10a3211b415832791a6bc6031f9ab}
```

低解密指数攻击

利用条件

主要利用的是私钥d很小，表现形式一般是e很大

举例

github上有开源的攻击代码https://github.com/pablocelayes/rsa-wiener-attack

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

m = bytes_to_long(flag)

p = getPrime(512)
q = getPrime(512)
N = p * q
phi = (p-1) * (q-1)
while True:
    d = getRandomNBitInteger(200)
    if GCD(d, phi) == 1:
        e = inverse(d, phi)
        break

c = pow(m, e, N)

print(c, e, N, sep='\n')

# 37625098109081701774571613785279343908814425141123915351527903477451570893536663171806089364574293449414561630
48531224706168619136666940438914234797256502057087717599209803375940331844370579186693936306196653821075861167 98
490379903151610356493899432565261678435766174691344131919509085829229022107913772200 66
# 46867417013414476511855705167486515292101865210840925173161828985833867821644239088991107524584028941183216735
11598631371996645860888168980237718163311138992081381435096431542042225705028751785121310946582344476789581737 23
77616723406116946259672358254060231210263961445286931270444042869857616609048537240249
# 86966590627372918010571457840724456774194080910694231109811773050866217415975647358784246153710824794652840306
38942872992377143134069934635464670839656420395727039388210504271492006005540154179474843724270718619294154618 56
66953574082803056612193004258064074902605834799171191314001030749992715155125694272289
```

exp.py

```
def rational_to_contfrac (x, y):
    '''
    Converts a rational x/y fraction into
    a list of partial quotients [a0, ..., an]
    '''
    a = x//y
    if a * y == x:
        return [a]
    else:
        pquotients = rational_to_contfrac(y, x - a * y)
        pquotients.insert(0, a)
        return pquotients
def convergents_from_contfrac(frac):
    '''
    computes the list of convergents
    using the list of partial quotients
    '''
    convs = [];
    for i in range(len(frac)):
        convs.append(contfrac_to_rational(frac[0:i]))
    return convs

def contfrac_to_rational (frac):
    '''Converts a finite continued fraction [a0, ..., an]
     to an x/y rational.
     '''
    if len(frac) == 0:
        return (0,1)
```

```
        return (0,1)
    elif len(frac) == 1:
        return (frac[0], 1)
    else:
        remainder = frac[1:len(frac)]
        (num, denom) = contfrac_to_rational(remainder)
        # fraction is now frac[0] + 1/(num/denom), which is
        # frac[0] + denom/num.
        return (frac[0] * num + denom, num)

def egcd(a,b):
    '''
    Extended Euclidean Algorithm
    returns x, y, gcd(a,b) such that ax + by = gcd(a,b)
    '''
    u, u1 = 1, 0
    v, v1 = 0, 1
    while b:
        q = a // b
        u, u1 = u1, u - q * u1
        v, v1 = v1, v - q * v1
        a, b = b, a - q * b
    return u, v, a

def gcd(a,b):
    '''
    2.8 times faster than egcd(a,b)[2]
    '''
    a,b=(b,a) if a<b else (a,b)
    while b:
        a,b=b,a%b
    return a

def modInverse(e,n):
    '''
    d such that de = 1 (mod n)
    e must be coprime to n
    this is assumed to be true
    '''
    return egcd(e,n)[0]%n

def totient(p,q):
    '''
    Calculates the totient of pq
    '''
    return (p-1)*(q-1)

def bitlength(x):
    '''
    Calculates the bitlength of x
    '''
    assert x >= 0
    n = 0
    while x > 0:
        n = n+1
        x = x>>1
    return n


def isqrt(n):
```

```
    '''
    Calculates the integer square root
    for arbitrary large nonnegative integers
    '''
    if n < 0:
        raise ValueError('square root not defined for negative numbers')

    if n == 0:
        return 0
    a, b = divmod(bitlength(n), 2)
    x = 2**(a+b)
    while True:
        y = (x + n//x)//2
        if y >= x:
            return x
        x = y


def is_perfect_square(n):
    '''
    If n is a perfect square it returns sqrt(n),

    otherwise returns -1
    '''
    h = n & 0xF; #last hexadecimal "digit"

    if h > 9:
        return -1 # return immediately in 6 cases out of 16.

    # Take advantage of Boolean short-circuit evaluation
    if ( h != 2 and h != 3 and h != 5 and h != 6 and h != 7 and h != 8 ):
        # take square root if you must
        t = isqrt(n)
        if t*t == n:
            return t
        else:
            return -1

    return -1

def hack_RSA(e,n):
    frac = rational_to_contfrac(e, n)
    convergents = convergents_from_contfrac(frac)

    for (k,d) in convergents:
        #check if d is actually the key
        if k!=0 and (e*d-1)%k == 0:
            phi = (e*d-1)//k
            s = n - phi + 1
            # check if the equation x^2 - s*x + n = 0
            # has integer roots
            discr = s*s - 4*n
            if(discr>=0):
                t = is_perfect_square(discr)
                if t!=-1 and (s+t)%2==0:
                    print("\nHacked!")
                    return d
from Crypto.Util.number import *
def main():
```

```
    c=376250981090817017745716137852793439088144251411239155151279034774515708935366631718060893645742934494145616304853122470616861913666694043891423479725650205708771759920980337594033184437057918669393630619665382107586116798490379903151610356493899432565261678435766174691344131919509085829229022107913772200660
    e=468674170134144765118557051674865152921018652108409251731618289858338678216442390889911075245840289411832167351159863137199664586088816898023771816331113899208138143509643154204222570502875178512131094658234447678958173723776167234061169462596723582540602312102639614452869312704440428698576166090485372402490
    n=869665906273729180105714578407244567741940809106942311098117730508662174159756473587842461537108247946528403063894287299237714313406993463546467083965642039572703938821050427149200600554015417947484372427071861929415461856669535740828030566121930042580640749026058347991711913140010307499927151551256946272890
    d=hack_RSA(e,n)
    print ("d=")
    print (d)
    m=pow(c,d,n)


    print(long_to_bytes(m))

if __name__ == '__main__':
    main()

'''
Hacked!
d=
14853131918303590550935457454515842994952729208404630087562330
flag{d3752538-90d0-c373-cfef-9247d3e16848}
'''
```

共模攻击

利用条件

识别：若干次加密，e不同，n相同，m相同。就可以在不分解n和求d的前提下，解出明文m。

推导过程：

首先，两个加密指数互质：

$gcd(e1,e2)=1$

即存在s1、s2使得：

$s1+*e1+s2*e2=1$

又因为：

$c1\equiv m^{e1} \bmod n$

$c2\equiv m \bmod n$

代入化简可得：

$c1^{s1} * c2^{s2} \equiv m \bmod n$

即可求出明文

举例

```
{626656572072690726599724135833158541709572614634198975553801712298136074281349840153359475708879653634194165969
1259323065631249,773}
{626656572072690726599724135833158541709572614634198975553801712298136074281349840153359475708879653634194165969
1259323065631249,839}

message1=345352059272344393545115154524502586423238887172168232640891502434980406204197670236472866068291239690396819398111315531115373 49
message2=567281802681629334407011933253662961945716357003630529686905353229310537969079338601906575446529286776952173641417080323830953 5
```

exp.py

```
import sys
import binascii
sys.setrecursionlimit(1000000)
def egcd(a, b):
    if a == 0:
      return (b, 0, 1)
    else:
      g, y, x = egcd(b % a, a)
      return (g, x - (b // a) * y, y)
def modinv(a, m):
    g, x, y = egcd(a, m)
    if g != 1:
      raise Exception('modular inverse does not exist')
    else:
      return x % m

c1=345352059272344393545115154524502586423238887172168232640891502434980406204197670236472866068291239690396819398113155311153734 9
n=626656572072690726599724135833158541709572614634198975553801712298136074281349840153359475708879653634194165969125932306563124 9
e1=773
c2=567281802681629334407011933253662961945716357003630529686905353229310537969079338601906575446529286776952173641417080323830953 5
e2=839

s = egcd(e1, e2)
s1 = s[1]
s2 = s[2]

if s1<0:
   s1 = - s1
   c1 = modinv(c1, n)
elif s2<0:
   s2 = - s2
   c2 = modinv(c2, n)
m=(pow(c1,s1,n)*pow(c2,s2,n)) % n
print(m)
#print (binascii.unhexlify(hex(m)[2:].strip("L")))
flag=[102,108,97,103,123,119,104,101,110,119,101,116,104,105,110,107,105,116,105,115,112,111,115,115,105,98,108,101,125]
r = ""
for i in flag:
    r += chr(i)
print(r)

'''
102108971031231191041011101191011161041051101071051161051151121111151151059810810112 5
flag{whenwethinkitispossible}
'''
```

参考：https://xz.aliyun.com/t/6459

国密算法

- **SM2 / SM3 / SM4**
  - SM2为非对称加密，基于ECC。该算法已公开。由于该算法基于ECC，故其签名速度与秘钥生成速度都快于RSA。
  - SM3 消息摘要。可以用MD5作为对比理解。该算法已公开。
  - SM4 无线局域网标准的分组数据算法。对称加密，密钥长度和分组长度均为128位。

SM2（对标国际RSA算法)

- SM2椭圆曲线公钥密码算法：
  - SM2椭圆曲线公钥密码算法是我国自主设计的公钥密码算法，
  - 包括SM2-1椭圆曲线数字签名算法
  - SM2-2椭圆曲线密钥交换协议
  - SM2-3椭圆曲线公钥加密算法

- 公钥加密算法：
  - SM2算法与RSA算法不同的是，SM2算法是基于椭圆曲线上点群离散对数难题

SM2椭圆曲线公钥密码算法

- 非对称密码算法：
  - 使用公钥进行加密而使用私钥进行解密的一类密码算法，已知公钥求私钥在计算上不可行。

- SM2椭圆曲线公钥密码体系：
  - 密钥 key
  - 私钥 private key
  - 公钥 public key
  - 秘密密钥 secret key
  - 消息 message
  - 明文 plaintext
  - 密文 ciphertext
  - 加密过程 encipherment
  - 解密过程 decipherment
  - 杂凑函数 hash function
  - 杂凑值 hash value
  - 密钥派生函数 key derivation function

SM2椭圆曲线公钥密码算法

- 加密算法
  - ✓ 设需要发送的消息为比特串M，klen为M的比特长度。

- ✓ 为了对明文M进行加密，作为加密者的用户A应实现以下运算步骤：
  - A1：用随机数发生器产生随机数k∈[1,n-1]；
  - A2：计算椭圆曲线点C1=[k]G=(x1,y1)，，将C1的数据类型转换为比特串；
  - A3：计算椭圆曲线点S=[h]PB，若S是无穷远点，则报错并退出；
  - A4：计算椭圆曲线点[k]PB=(x2,y2)，将坐标x2、y2 的数据类型转换为比特串
  - A5：计算t=KDF(x2 ‖ y2, klen)，若t为全0比特串，则返回A1；
  - A6：计算C2 = M ⊕ t；
  - A7：计算C3 = Hash(x2 ‖ M ‖ y2)；
  - A8：输出密文C = C1 ‖ C2 ‖ C3。

➢ 解密算法

- ✓ 设klen为密文中C2的比特长度。
- ✓ 为了对密文C=C1 ‖ C2 ‖ C3 进行解密，作为解密者的用户B应实现以下运算步骤：
  - B1：从C中取出比特串C1，将C1的数据类型转换为椭圆曲线上的点，验证C1是否满足椭圆曲线方程，若不满足则报错并退出；
  - B2：计算椭圆曲线点S=[h]C1，若S是无穷远点，则报错并退出；
  - B3：计算[dB]C1=(x2,y2)，将坐标x2、y2的数据类型转换为比特串；
  - B4：计算t=KDF(x2 ‖ y2, klen)，若t为全0比特串，则报错并退出；
  - B5：从C中取出比特串C2，计算M' = C2 ⊕ t；
  - B6：计算u = Hash(x2 ‖ M' ‖ y2)，从C中取出比特串C3，若u != C3，则报错并退出；

➢ 加密算法

- ✓ 设需要发送的消息为比特串M，klen为M的比特长度。
- ✓ 为了对明文M进行加密，作为加密者的用户A应实现以下运算步骤：
  - A1：用随机数发生器产生随机数k∈[1,n-1]；
  - A2：计算椭圆曲线点C1=[k]G=(x1,y1)，，将C1的数据类型转换为比特串；
  - A3：计算椭圆曲线点S=[h]PB，若S是无穷远点，则报错并退出；
  - A4：计算椭圆曲线点[k]PB=(x2,y2)，将坐标x2、y2 的数据类型转换为比特串
  - A5：计算t=KDF(x2 ‖ y2, klen)，若t为全0比特串，则返回A1；
  - A6：计算C2 = M ⊕ t；
  - A7：计算C3 = Hash(x2 ‖ M ‖ y2)；
  - A8：输出密文C = C1 ‖ C2 ‖ C3。

- ➢ **SM3算法:哈希算法（散列算法，杂凑算法）**，任意长度的数据经过SM3算法后会生成长度固定为256bit的摘要。SM3算法的逆运算在数学上是不可实现的，即通过256bit的摘要无法反推出原数据的内容，因此在信息安全领域内常用SM3算法对信息的完整性进行度量

- ➢ **场景：**适用于商用密码应用中的**数字签名和验证，消息认证码的生成与验证**以及**随机数的生成**，可满足多种密码应用的安全需求。

- SM3算法主要包括数据填充、数据扩展和迭代压缩三个过程。

- 实现SM3算法所需常量:
  - IV：初始值，用于确定压缩函数寄存器的初态
  - T_j ：常量，随j的变化取不同的值

4.1 初始值

IV=7380166f 4914b2b9 172442d7 da8a0600 a96f30bc 163138aa e38dee4d b0fb0e4e

4.2 常量

$$T_j = \begin{cases} 79cc4519 & 0 \le j \le 15 \\ 7a879d8a & 16 \le j \le 63 \end{cases}$$

SM3杂凑算法

- 实现SM3算法所需函数:
  - FF_j / CG_j ：布尔函数，随j的变化去取不同的表达式
  - P_0/P_1：置换函数

4.3 布尔函数

$$FF_j(X,Y,Z) = \begin{cases} X \oplus Y \oplus Z & 0 \le j \le 15 \\ (X \wedge Y) \vee (X \wedge Z) \vee (Y \wedge Z) & 16 \le j \le 63 \end{cases}$$

$$GG_j(X,Y,Z) = \begin{cases} X \oplus Y \oplus Z & 0 \le j \le 15 \\ (X \wedge Y) \vee (\neg X \wedge Z) & 16 \le j \le 63 \end{cases}$$

式中X,Y,Z 为字。

4.4 置换函数

$$P_0(X) = X \oplus (X \lll 9) \oplus (X \lll 17)$$

$$P_1(X) = X \oplus (X \lll 15) \oplus (X \lll 23)$$

式中X为字。

SM3杂凑算法

- 迭代压缩:
  - 压缩函数

令A,B,C,D,E,F,G,H为字寄存器,$SS1,SS2,TT1,TT2$为中间变量,压缩函数$V^{i+1} = CF(V^{(i)}, B^{(i)}), 0 \le i \le n-1$。计算过程描述如下：

$ABCDEFGH \leftarrow V^{(i)}$

FOR j=0 TO 63

$SS1 \leftarrow ((A \lll 12) + E + (T_j \lll j)) \lll 7$

$SS2 \leftarrow SS1 \oplus (A \lll 12)$

$TT1 \leftarrow FF_j(A,B,C) + D + SS2 + W'_j$

$TT2 \leftarrow GG_j(E,F,G) + H + SS1 + W_j$

$D \leftarrow C$

$C \leftarrow B \lll 9$

$B \leftarrow A$

$A \leftarrow TT1$

$H \leftarrow G$

$G \leftarrow F \lll 19$

$F \leftarrow E$

$E \leftarrow P_0(TT2)$

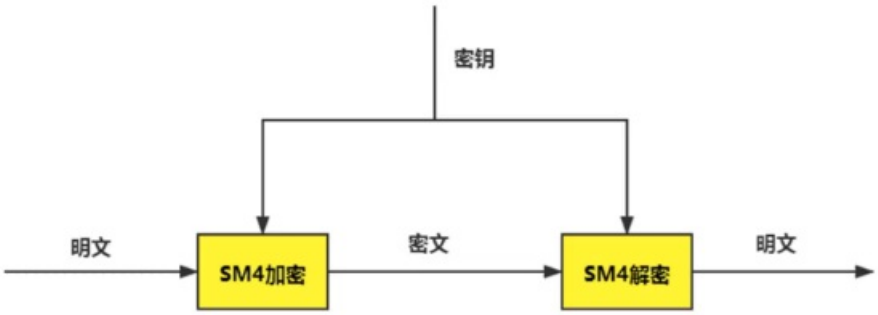
ENDFOR

$V^{(i+1)} \leftarrow ABCDEFGH \oplus V^{(i)}$

其中，字的存储为大端(big-endian)格式。

SM4分组密码算法

- 对称加密



- 使用某一密钥加密后的密文只能用该密钥解密出明文，故而称为对称加密

SM4分组密码算法

- 固定参数CK

固定参数 $CK_i(i=0,1,\cdots,31)$具体值为：

00070E15，1C232A31，383F464D，545B6269，
70777E85，8C939AA1，A8AFB6BD，C4CBD2D9，
E0E7EEF5，FC030A11，181F262D，343B4249，
50575E65，6C737A81，888F969D，A4ABB2B9，
C0C7CED5，DCE3EAF1，F8FF060D，141B2229，
30373E45，4C535A61，686F767D，848B9299，
A0A7AEB5，BCC3CAD1，D8DFE6ED，F4FB0209，
10171E25，2C333A41，484F565D，646B7279。

- 系统参数FK

b) 系统参数 FK 的取值为：

$FK_0=(A3B1BAC6), FK_1=(56AA3350), FK_2=(677D9197), FK_3=(B27022DC)$；

SM4分组密码算法

- Sbox

表 1 Sbox 数据

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	D6	90	E9	FE	CC	E1	3D	B7	16	B6	14	C2	28	FB	2C	05
1	2B	67	9A	76	2A	BE	04	C3	AA	44	13	26	49	86	06	99
2	9C	42	50	F4	91	EF	98	7A	33	54	0B	43	ED	CF	AC	62
3	E4	B3	1C	A9	C9	08	E8	95	80	DF	94	FA	75	8F	3F	A6
4	47	07	A7	FC	F3	73	17	BA	83	59	3C	19	E6	85	4F	A8
5	68	6B	81	B2	71	64	DA	8B	F8	EB	0F	4B	70	56	9D	35
6	1E	24	0E	5E	63	58	D1	A2	25	22	7C	3B	01	21	78	87
7	D4	00	46	57	9F	D3	27	52	4C	36	02	E7	A0	C4	C8	9E
8	EA	BF	8A	D2	40	C7	38	B5	A3	F7	F2	CE	F9	61	15	A1
9	E0	AE	5D	A4	9B	34	1A	55	AD	93	32	30	F5	8C	B1	E3

表 1（续）

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
A	1D	F6	E2	2E	82	66	CA	60	C0	29	23	AB	0D	53	4E	6F

- 例如“EF”经过第E行和第F列的Sbox(EF)=84

B	D5	DB	37	45	DE	FD	8E	2F	03	FF	6A	72	6D	6C	5B	51
C	8D	1B	AF	92	BB	DD	BC	7F	11	D9	5C	41	1F	10	5A	D8
D	0A	C1	31	88	A5	CD	7B	BD	2D	74	D0	12	B8	E5	B4	B0
E	89	69	97	4A	0C	96	77	7E	65	B9	F1	09	C5	6E	C6	84
F	18	F0	7D	EC	3A	DC	4D	20	79	EE	5F	3E	D7	CB	39	48

➢ 例如“EF”经过S盒的值为第E行和第F列的值，Sbox(EF)=84

实操

RSA是一种算法，并且广泛应用于现代，用于保密通信。
RSA算法涉及三个参数,`n,e,d`，其中分为私钥和公钥，私钥是`n,d`，公钥是`n,e`
n是两个素数的乘积，一般这两个素数在RSA中用字母p,q表示
e是一个素数
d是e模 `varphi(n)` 的逆元，CTF的角度看就是，d是由`e,p,q`可以求解出的
一般CTF就是把我们想要获得的flag作为明文，RSA中表示为m。然后通过RSA加密，得到**密文**，RSA中表示为C。

加密过程 `c=m^e mod n`

```
c=pow(m,e,n)
```

解密过程 `m=c^d mod n`

```
m=pow(c,d,n)
```

求解私钥d

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

一般来说，n，e是公开的，但是由于n一般是两个大素数的乘积，所以我们很难求解出d，所以RSA加密就是利用现代无法快速实现大素数的分解，所存在的一种安全的非对称加密。

基础RSA加密脚本

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

msg = 'flag is :testflag'
hex_msg=int(msg.encode("hex"),16)
print(hex_msg)
p=getPrime(100)
q=getPrime(100)
n=p*q
e=0x10001
phi=(p-1)*(q-1)
d=gmpy2.invert(e,phi)
print("d=",hex(d))
c=pow(hex_msg,e,n)
print("e=",hex(e))
print("n=",hex(n))
print("c=",hex(c))
```

```
348528637939318975470908238077546712518l5
('d=', '0x66e16452adf1d19c8cbad6544d0e519a3072fb25cfcec1e041')
('e=', '0x10001')
('n=', '0xb072fbfb159a37ff21a09106517fe9eddfb70fa7a199d50913L')
('c=', '0x35499823e8e3c9e27909d5c46da61f29b95883ad4c017241d0L')
```

基础RSA解密脚本

```
#!/usr/bin/env python
# -*- coding:utf-8 -*-
import binascii
import gmpy2
n=0x80b32f2ce68da974f25310a23144977d76732fa78fa29fdcbf
#这边我用yafu分解了n
p=78090079033426965944329795684 3
q=10345265594079935077348184088 29
e=0x10001
c=0x534280240c65bb1104ce3000bc8181363806e7173418d15762

phi=(p-1)*(q-1)
d=gmpy2.invert(e,phi)
m=pow(c,d,n)
print(hex(m))
print(binascii.unhexlify(hex(m)[2:].strip("L")))
```

```
0x666c6167206973203a74657374666c6167
flag is :testflag
```