实验吧 密码学

### 变异凯撒

加密密文：afZ\_r9VYfScOeO\_UL^RWUc  
格式：flag{ }

找规律找规律，既然说是[凯撒](https://baike.baidu.com/item/%E6%81%BA%E6%92%92%E5%AF%86%E7%A0%81/4905284?fromtitle=%E5%87%AF%E6%92%92%E5%AF%86%E7%A0%81&fromid=1336345&fr=aladdin)那就看看怎么移动才能得到flag。

因为flag{前四个字符是确定的，所以对比下密文和flag{的ascii码。

IMG_256

发现规律是这样，然后写个python脚本

#实验吧密码学第一题变异凯撒  
m="afZ\_r9VYfScOeO\_UL^RWUc"  
j=5  
for i in m:  
 c=chr(ord(i)+j)  
 j=j+1  
 print(c,end="")

运行之后结果如下：

flag{Caesar\_variation}

自我感悟：要学会查找规律，凯撒就是简单的移位密码，flag开头所对应的ASCII码是不会变的，就从这个角度出发。

### 传统知识+古典密码



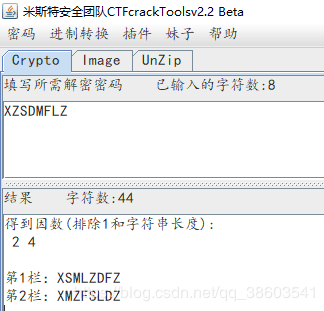
说实话，第一眼看到这个题，很懵，只能靠猜测来进行求解。

首先，我们根据天干地支表，把题中密文转换为数字：





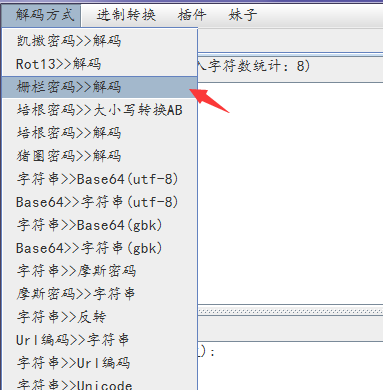
然后，使用密码解析工具CTFcrackTools进行，解码，经过几次测试之后，发现，栅栏密码好像挺靠谱的，嘿嘿，就它了



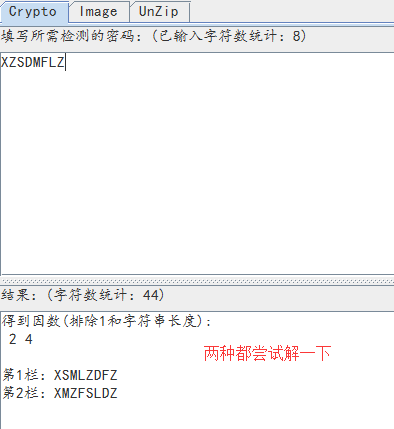
咦~解析出两个？？那就挨个试试吧……

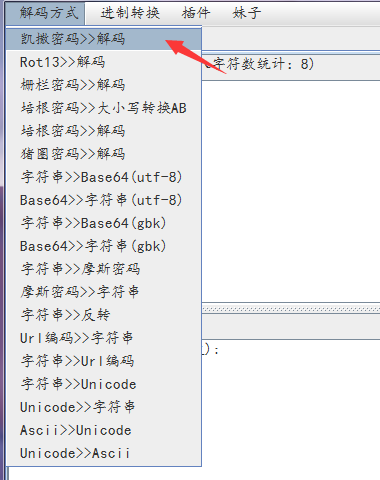
第2栏栅栏密码用凯撒密码解析出一下结果。。。

应为题目是“传统知识+古典密码”，而最常见的就是凯撒和栅栏两种。那么我们对他进行尝试。首先发现得到的字符串是8位，那么我们尝试栅栏解密



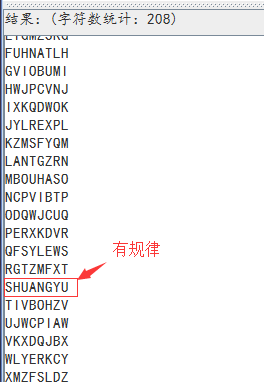
用工具解密后分出了两个字符串，那么我们对这两个字符串分别进行凯撒解密尝试一下







第一个字符串解密后并没有发现有规律的信息，那么对第二个字符串进行解密后发现有一个有规律的信息，他就是我们要找的答案





# ****四. try them all****

You have found a passwd file containing salted passwords. An unprotected configuration file has revealed a salt of 5948. The hashed password for the 'admin' user appears to be 81bdf501ef206ae7d3b92070196f7e98, try to brute force this password.

相关知识：[什么是salt](https://blog.csdn.net/yang1982_0907/article/details/43483205)，[怎么用python生成md5](http://outofmemory.cn/code-snippet/939/python-liangzhong-produce-md5-method)

# 密码学中的“盐值 Salt”

# 为什么要在密码里加点“盐”

盐（Salt）

在密码学中，是指通过在密码任意固定位置插入特定的字符串，让散列后的结果和使用原始密码的散列结果不相符，这种过程称之为“加盐”。

以上这句话是维基百科上对于 Salt 的定义，但是仅凭这句话还是很难理解什么叫 Salt，以及它究竟起到什么作用。

## 第一代密码

早期的软件系统或者互联网应用，数据库中设计用户表的时候，大致是这样的结构：

mysql> desc User;

+----------+--------------+------+-----+---------+-------+

| Field | Type | Null | Key | Default | Extra |

+----------+--------------+------+-----+---------+-------+

| UserName | varchar(50) | NO | | | |

| PassWord | varchar(150) | NO | | | |

+----------+--------------+------+-----+---------+-------+

数据存储形式如下：

mysql> select \* from User;

+----------+----------+

| UserName | PassWord |

+----------+----------+

| lichao | 123 |

| akasuna | 456 |

+----------+----------+

主要的关键字段就是这么两个，一个是登陆时的用户名，对应的一个密码，而且那个时候的用户名是明文存储的，如果你登陆时用户名是 123，那么数据库里存的就是 123。这种设计思路非常简单，但是缺陷也非常明显，数据库一旦泄露，那么所有用户名和密码都会泄露，后果非常严重。参见[《CSDN 详解 600 万用户密码泄露始末》](http://tech.qq.com/a/20111221/000485.htm)。

## 第二代密码

为了规避第一代密码设计的缺陷，聪明的人在数据库中不在存储明文密码，转而存储加密后的密码，典型的加密算法是 MD5 和 SHA1，其数据表大致是这样设计的：

mysql> desc User;

+----------+--------------+------+-----+---------+-------+

| Field | Type | Null | Key | Default | Extra |

+----------+--------------+------+-----+---------+-------+

| UserName | varchar(50) | NO | | | |

| PwdHash | char(32) | NO | | | |

+----------+--------------+------+-----+---------+-------+

数据存储形式如下：

mysql> select \* from User;

+----------+----------------------------------+

| UserName | PwdHash |

+----------+----------------------------------+

| lichao | 202cb962ac59075b964b07152d234b70 |

| akasuna | 250cf8b51c773f3f8dc8b4be867a9a02 |

+----------+----------------------------------+

假如你设置的密码是 123，那么数据库中存储的就是 202cb962ac59075b964b07152d234b70 或 40bd001563085fc35165329ea1ff5c5ecbdbbeef。当用户登陆的时候，会把用户输入的密码执行 MD5（或者 SHA1）后再和数据库就行对比，判断用户身份是否合法，这种加密算法称为****散列****。

严格地说，这种算法不能算是加密，因为理论上来说，它不能被解密。所以即使数据库丢失了，但是由于数据库里的密码都是密文，根本无法判断用户的原始密码，所以后果也不算太严重。

## 第三代密码

本来第二代密码设计方法已经很不错了，只要你密码设置得稍微复杂一点，就几乎没有被破解的可能性。但是如果你的密码设置得不够复杂，被破解出来的可能性还是比较大的。

好事者收集常用的密码，然后对他们执行 MD5 或者 SHA1，然后做成一个数据量非常庞大的数据字典，然后对泄露的数据库中的密码就行对比，如果你的原始密码很不幸的被包含在这个数据字典中，那么花不了多长时间就能把你的原始密码匹配出来。这个数据字典很容易收集，CSDN 泄露的那 600w 个密码，就是很好的原始素材。

于是，第三代密码设计方法诞生，用户表中多了一个字段：

mysql> desc User;

+----------+-------------+------+-----+---------+-------+

| Field | Type | Null | Key | Default | Extra |

+----------+-------------+------+-----+---------+-------+

| UserName | varchar(50) | NO | | | |

| Salt | char(50) | NO | | | |

| PwdHash | char(32) | NO | | | |

+----------+-------------+------+-----+---------+-------+

数据存储形式如下：

mysql> select \* from User;

+----------+----------------------------+----------------------------------+

| UserName | Salt | PwdHash |

+----------+----------------------------+----------------------------------+

| lichao | 1ck12b13k1jmjxrg1h0129h2lj | 6c22ef52be70e11b6f3bcf0f672c96ce |

| akasuna | 1h029kh2lj11jmjxrg13k1c12b | 7128f587d88d6686974d6ef57c193628 |

+----------+----------------------------+----------------------------------+

Salt 可以是任意字母、数字、或是字母或数字的组合，但必须是随机产生的，每个用户的 Salt 都不一样，用户注册的时候，数据库中存入的不是明文密码，也不是简单的对明文密码进行散列，而是 MD5( 明文密码 + Salt)，也就是说：

MD5('123' + '1ck12b13k1jmjxrg1h0129h2lj') = '6c22ef52be70e11b6f3bcf0f672c96ce'

MD5('456' + '1h029kh2lj11jmjxrg13k1c12b') = '7128f587d88d6686974d6ef57c193628'

当用户登陆的时候，同样用这种算法就行验证。

由于加了 Salt，即便数据库泄露了，但是由于密码都是加了 Salt 之后的散列，坏人们的数据字典已经无法直接匹配，明文密码被破解出来的概率也大大降低。

是不是加了 Salt 之后就绝对安全了呢？淡然没有！坏人们还是可以他们数据字典中的密码，加上我们泄露数据库中的 Salt，然后散列，然后再匹配。但是由于我们的 Salt 是随机产生的，假如我们的用户数据表中有 30w 条数据，数据字典中有 600w 条数据，坏人们如果想要完全覆盖的坏，他们加上 Salt 后再散列的数据字典数据量就应该是 300000\* 6000000 = 1800000000000，一万八千亿啊，干坏事的成本太高了吧。但是如果只是想破解某个用户的密码的话，只需为这 600w 条数据加上 Salt，然后散列匹配。可见 Salt 虽然大大提高了安全系数，但也并非绝对安全。

实际项目中，Salt 不一定要加在最前面或最后面，也可以插在中间嘛，也可以分开插入，也可以倒序，程序设计时可以灵活调整，都可以使破解的难度指数级增长。

PS，文中所谓第一、二、三代密码的称呼，是我自己 YY 的。

# python3中使用md5

使用hashlib

import hashlib

m2 = hashlib.md5()

m2.update(src) print m2.hexdigest()

MD5

首先来介绍一下什么是md5

MD5消息摘要算法（英语：MD5 Message-Digest Algorithm），一种被广泛使用的密码散列函数，可以产生出一个128位（16字节）的散列值（hash value），用于确保信息传输完整一致。

什么是摘要算法呢？摘要算法又称哈希算法、散列算法。它通过一个函数，把任意长度的数据转换为一个长度固定的数据串（通常用16进制的字符串表示）

详细查看百度百科md5

在python3中的用法

import hashlib

# 创建MD5对象，可以直接传入要加密的数据

m=hashlib.md5('123456'.encode(encoding='utf-8'))

# m=hashlib.md5(b'123456') 与上面等价

print(m)

print(m.hexdigest()) # 转化为16进制打印md5值

这里传入的字符串前加以个b将其转为二进制，或者声明为utf-8,否则回报错误TypeError: Unicode-objects must be encoded before hashing

输出结果为

<md5 HASH object @ 0x000001C67C71C8A0>

e10adc3949ba59abbe56e057f20f883e

也可以使用MD5对象的update()方法传入要加密的数据

import hashlib

str='This is a string.'

m=hashlib.md5()

m.update(str.encode(encoding='utf-8'))

print(m.hexdigest())

结果为：

13562b471182311b6eea8d241103e8f0

如果要被加密的数据太长，可以分段update,结果是一样的

import hashlib

str='This is a string.'

m=hashlib.md5()

m.update('This i'.encode('utf-8'))

m.update('s a string.'.encode('utf-8'))

print(m.hexdigest())

结果为：

13562b471182311b6eea8d241103e8f0

原文链接：<https://blog.csdn.net/API_demo/article/details/81878459>

这道题给你了一串加密后的字符和一串salt值，我们的思路是先解密再去盐

既然是md5解密（匹配），而且题目提示是暴力破解，那么我们有两种思路1.上网找md5的字典跑 2.写个python脚本（最多六位，多的话跑太久就不能做题了）

下面贴下实验吧里WP的解密脚本

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

\_\_author\_\_ = 'Administrator'

#from ultrapower.fd

import itertools as its

import md5

#暴力破解

def uncipher(maxlenth,salt,ciphertext\_s,str\_letter):

ciphertext\_s=ciphertext\_s

salt = salt

maxlenth=int(maxlenth)

str\_letter=str\_letter

ciphertext=''

for i in range(1, maxlenth+1):

# 迭代生成明文(例如abc,repeat=2 结果为（a,a)(a,b)(a,c)(b,b)(b,a)(b,c)(c,c)(c,a)(c,b)

r = its.product(str\_letter, repeat=i)

for j in r:

plaintext = "".join(j) #连接成字符串

plaintext = "%s%s" % (plaintext, salt) #把盐加到明文的后面 每次生成的最终明文

print plaintext #打印明文

# 开始解密，方法是，每个明文进来，加密成密文，然后密文与密文做对比

md501 = md5.new()

md501.update(plaintext)

ciphertext = md501.hexdigest()

# 对比密文确认明文

if ciphertext == ciphertext\_s: #如果密文一致 退出2层循环

break

if ciphertext == ciphertext\_s: #如果密文一致，退出1层循环，打印结果

print "task finished(plain,cipher)"

print "%s:%s" % (plaintext, ciphertext) #打印结果

break

#开始执行主函数

#str\_letter="abcdefghijklmnopqrstuvwxyz0123456789"

#str\_letter="abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789"

#str\_letter="abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ"

str\_letter="abcdefghijklmnopqrstuvwxyz" #明文的字符范围

maxlenth=6 #明文的最大长度,一般不会超过6位，否则短时间很难暴力破解

salt='5948' #加盐 #如果不加盐，为空就是正常的md5解密

ciphertext\_s='81bdf501ef206ae7d3b92070196f7e98' #密文

uncipher(maxlenth,salt,ciphertext\_s,str\_letter) #开始解密

原文链接：<https://blog.csdn.net/taozijun/article/details/80948549>

其实就是迭代生成字符串，加盐后md5加密再与题目中的密文比较，比较成功就输出原文。

最后跑出的答案是sniper5948，去掉盐值就是sniper

### trivial分值：20

An unlocked terminal is displaying the following:  
  
Encryption complete, ENC(???,T0pS3cre7key) = Bot kmws mikferuigmzf rmfrxrwqe abs perudsf! Nvm kda ut ab8bv\_w4ue0\_ab8v\_DDU  
  
You poke around and find this interesting file.

未上锁的终端机显示以下内容:

加密完成，ENC(??，T0pS3cre7key) = Bot kmws mikferuigmzf rmfrxrwqe abs perudsf!Nvm kda ut ab8bv\_w4ue0\_ab8v\_DDU

你随便看看，就会发现这个有趣的文件。

题目代码如下：

#!/usr/bin/env python  
import sys  
  
alphaL = "abcdefghijklnmopqrstuvwxyz"  
alphaU = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"  
num = "0123456789"  
keychars = num+alphaL+alphaU  
  
if len(sys.argv) != 3:  
 print ("Usage: %s SECRET\_KEY PLAINTEXT"%(sys.argv[0]))  
 sys.exit()  
  
key = sys.argv[1]  
if not key.isalnum():  
 print ("Your key is invalid, it may only be alphanumeric characters")  
 sys.exit()  
  
plaintext = sys.argv[2]  
  
ciphertext = ""  
for i in range(len(plaintext)):  
 rotate\_amount = keychars.index(key[i%len(key)])  
 if plaintext[i] in alphaL:  
 enc\_char = ord('a') + (ord(plaintext[i])-ord('a')+rotate\_amount)%26  
 elif plaintext[i] in alphaU:  
 enc\_char = ord('A') + (ord(plaintext[i])-ord('A')+rotate\_amount)%26  
 elif plaintext[i] in num:  
 enc\_char = ord('0') + (ord(plaintext[i])-ord('0')+rotate\_amount)%10  
 else:  
 enc\_char = ord(plaintext[i])  
 ciphertext = ciphertext + chr(enc\_char)  
  
print ("Encryption complete, ENC(%s,%s) = %s"%(plaintext,key,ciphertext))

从代码我们可以了解到，这是一个for i in range(len(plaintext))开始的加密算法（前面的语句是要求执行这个加密文件前面放3段语句，第一段是文件路径名，这个不用你放，第二是key，第三是原文）这道题是个有密匙的对称加密算法，做这道题首先我们要知道个知识点：若A=(B-C)%D  则B=(A+C)%D 具体推导见百度

比如假设都是正整数

A=(B-C)%D

则 B - C = D\*n + A 其中 A < D

移项 B = A+C + D\*n

当B<D时，两边对D取摸，

B = B%D = ( A+C + D\*n )%D = (A+C)%D

那么加密算法就可以看成 y = A + （x - A + B）% d    转换下就是 y - A = (x - A + B)%d

把x - A和y - A看成一个整体，是不是很熟悉！ 那易得解密算法就是x - A = (y - A + B)%d

换一下形势就是x= (y - A + B)%d + A

给一个别人的python解密小脚本

alphaL = "abcdefghijklnmopqrstuvqxyz"  
alphaU = "ABCDEFGHIJKLMNOPQRSTUVQXYZ"  
num = "0123456789"  
keychars = num+alphaL+alphaU  
key = 'T0pS3cre7key'  
ciphertext = 'Bot kmws mikferuigmzf rmfrxrwqe abs perudsf! Nvm kda ut ab8bv\_w4ue0\_ab8v\_DDU'  
  
s = ""  
for i in range(len(ciphertext)):  
 rotate\_amount = keychars.index(key[i%len(key)])  
 if ciphertext[i] in alphaL:  
 enc\_char = ord('a') + (ord(ciphertext[i]) - ord('a') - rotate\_amount) % 26  
 elif ciphertext[i] in alphaU:  
 enc\_char = ord('A') + (ord(ciphertext[i]) - ord('A') - rotate\_amount) % 26  
 elif ciphertext[i] in num:  
 enc\_char = ord('0') + (ord(ciphertext[i]) - ord('0') - rotate\_amount) % 10  
 else:  
 enc\_char = ord(ciphertext[i])  
 s = s + chr(enc\_char)  
print(s)