2021年9月绿城杯，CRYPTO的[warmup]加密算法

原创

沐一 · 林 于 2021-09-30 10:45:45 发布 49 收藏 1

分类专栏： CTF 密码学 文章标签： ctf

本文链接：https://blog.csdn.net/xiao__1bai/article/details/120561288

CTF 同时被 2 个专栏收录

167 篇文章 6 订阅

订阅专栏

密码学

51 篇文章 1 订阅

订阅专栏

2021年9月绿城杯，CRYPTO的[warmup]加密算法：

.

.

打开文件，发现是一个简单的加密表单元素`下标对应加密`：

```
from Crypto.Util.number import *
from flag import flag
assert flag[:5]=='flag{'

str1 = 'abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ'
def encode(plain_text, a, b, m):
        cipher_text = ''
        for i in plain_text:
                if i in str1:
                        addr = str1.find(i)
                        cipher_text += str1[(a*addr+b) % m]
                else:
                        cipher_text += i
        print(cipher_text)

encode(flag,37,23,52)
# cipher_text = 'aoxL{XaaHKP_tHgwpc_hN_ToXnnht}'
```

加密表单

明文对应表单进行下标对应，下标是加密关键。

不在表单内的直接获取即可

下标加密的一些参数

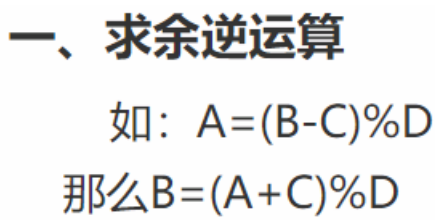
密文

.

.

(这里积累第一个经验)

因为下标加密是求余运算加密，所以上网找了一下求余运算的`逆运算`，结果发现负数的求余运算我解决不了。。。(哭~)



.

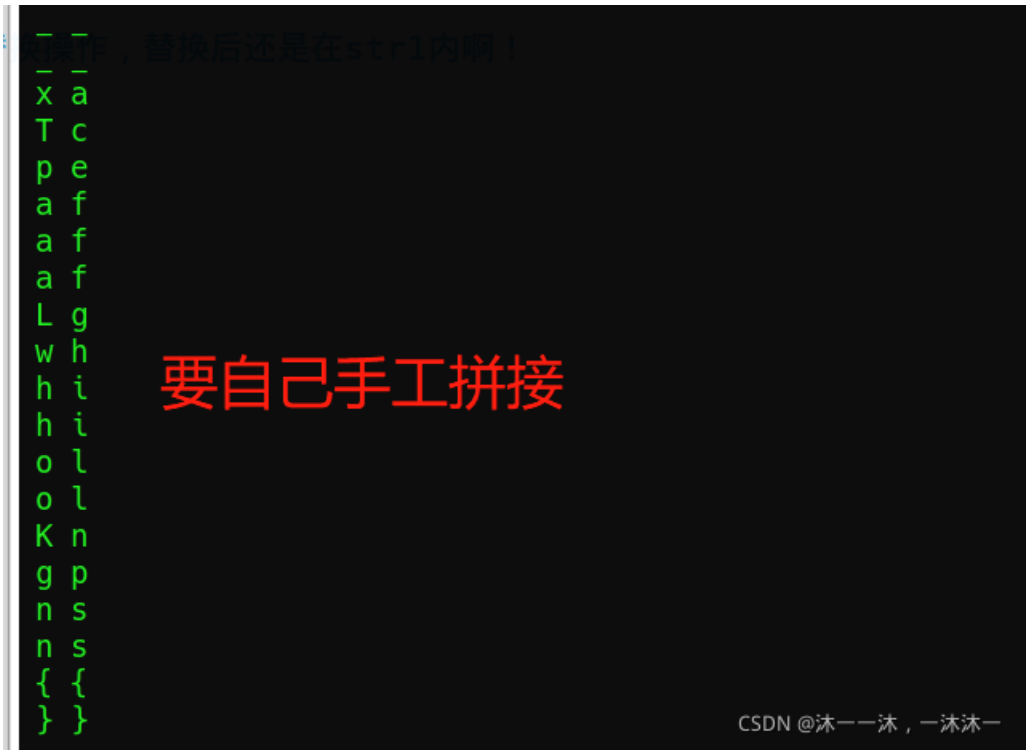
.

所以直接爆破算了，这是我第一次真正用爆破做题，所以比较生疏，先回顾一下一开始我犯的错误。

(这里积累第二个经验)

一开始我写的脚本是这样的，在`ASCII`的`32~127`中找到加密后与密文对应的字符，然后取`chr(i)`，然后错误就比较明显了，因为`for循环`会一直`向前`走，如果密文不按顺序摆放就会`跳过`，所以没法用`flag+=chr(i)`的方法来获取密文，结果就是得到明文字符----密文字符的对应，然后要自己一个个`拼接`：

```
mi_wen= 'aoxL{XaaHKP_tHgwpc_hN_ToXnnht}'
str1 = 'abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ' #长度52
c=0
flag=[]
cipher_text = ''
for i in range(32,127,1):
 if chr(i) in str1:   #flag在a~Z内，就获取下标并进行下标替换操作，替换后还是在str1内啊!
  addr = str1.find(chr(i))
  cipher_text = str1[(37*addr+23) % 52] #求余52
 else:     #flag不在a~Z内，加1后还是不在str1内
  cipher_text = chr(i) #flag在a~Z内，就不用加密直接用
 for a in mi_wen:
  if cipher_text==a:
   print(a,chr(i))
```



.

.

.

后来想了想，竟然`for循环`一直往前走，但是如果我换一下`顺序`，每个密文字符遍历一次`ASCII`字符，那不就密文也`往前走`了吗？虽然这样的算法复杂度大大增加，但是对电脑来说根本不值得一提好吧：

```python
mi_wen= 'aoxL{XaaHKP_tHgwpc_hN_ToXnnht}'
str1 = 'abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ' #长度52
c=0
flag=[]
cipher_text = ''
for a in mi_wen:
 for i in range(32,127):
  if chr(i) in str1:   #flag在a~Z内，就获取下标并进行下标替换操作，替换后还是在str1内啊！
   addr = str1.find(chr(i))
   cipher_text = str1[(37*addr+23) % 52] #求余52
  else:     #flag不在a~Z内，加1后还是不在str1内
   cipher_text = chr(i) #flag在a~Z内，就不用加密直接用
  if cipher_text==a:
   flag+=chr(i)
print(''.join(flag))
```

```
└─$ python 3.py
flag{AffInE_CIpheR_iS_clAssiC}
```

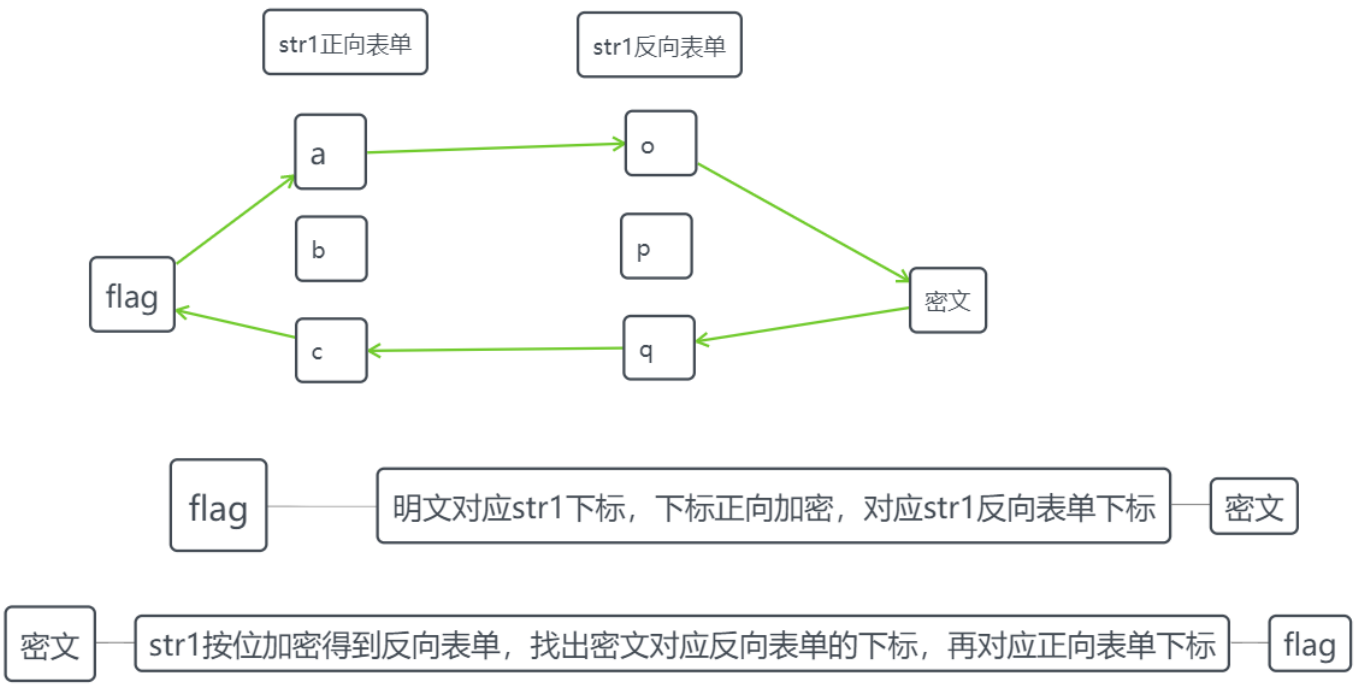
.

.

.

（这里积累第三个经验）

这里补充一些别人的做法，他直接把码表加密，之后按位找。是一种比较新颖的方法，是对正向反向下标搞操作，值得研究，我感觉他的逻辑应该是这样的：



.

还来有一天发现了是仿射密码：

仿射密码

仿射密码也是一个单表替换密码，字母表中的每个字母相应的值使用一个简单的数学函数映射到对应的数值，再把对应数值转换成字母。

仿射密码的加密函数是 $e(x) = ax + b \pmod{m}$，其中：

- a和m互质。
- m是字母的数目。

解密函数是$d(x) = a^{-1}(x - b) \pmod{m}$，其中 a^-1 是 a 在集合 Z 上的逆元，满足 a * a^-1 ≡ 1 mod 26

求集合Z上某个数的逆元：

```
gmpy2.invert(x, Z)
```



```
mi_wen= 'aoxL{XaaHKP_tHgwpc_hN_ToXnnht}'
str1 = 'abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ' #长度52
flag=''
def encode(plain_text, a, b, m):
 cipher_text = ''
 for i in plain_text:
  if i in str1:
   addr = str1.find(i)
   cipher_text += str1[(a*addr+b) % m]
  else:
   cipher_text += i
 return cipher_text

reverse=encode(str1,37,23,52)

for i in mi_wen:  #对应密文下标
 if i in str1:
  addr=reverse.find(i) #对应反向str1下标
  flag+=str1[addr]  #获取正向明文
 else:
  flag+=i
print(flag)
```

```
└─$ python 2.py
flag{AffInE_CIpheR_iS_clAssiC}
```

.

.

.

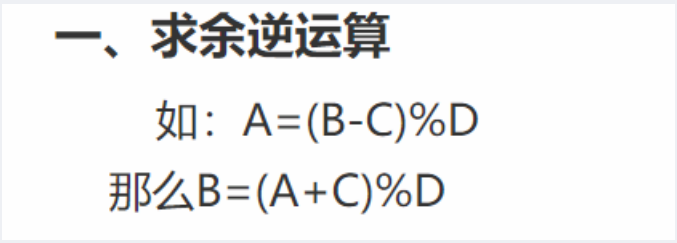
最后还差原本的负数求余逆运算了，这个暂时找不到资料，先放着。~

.

.

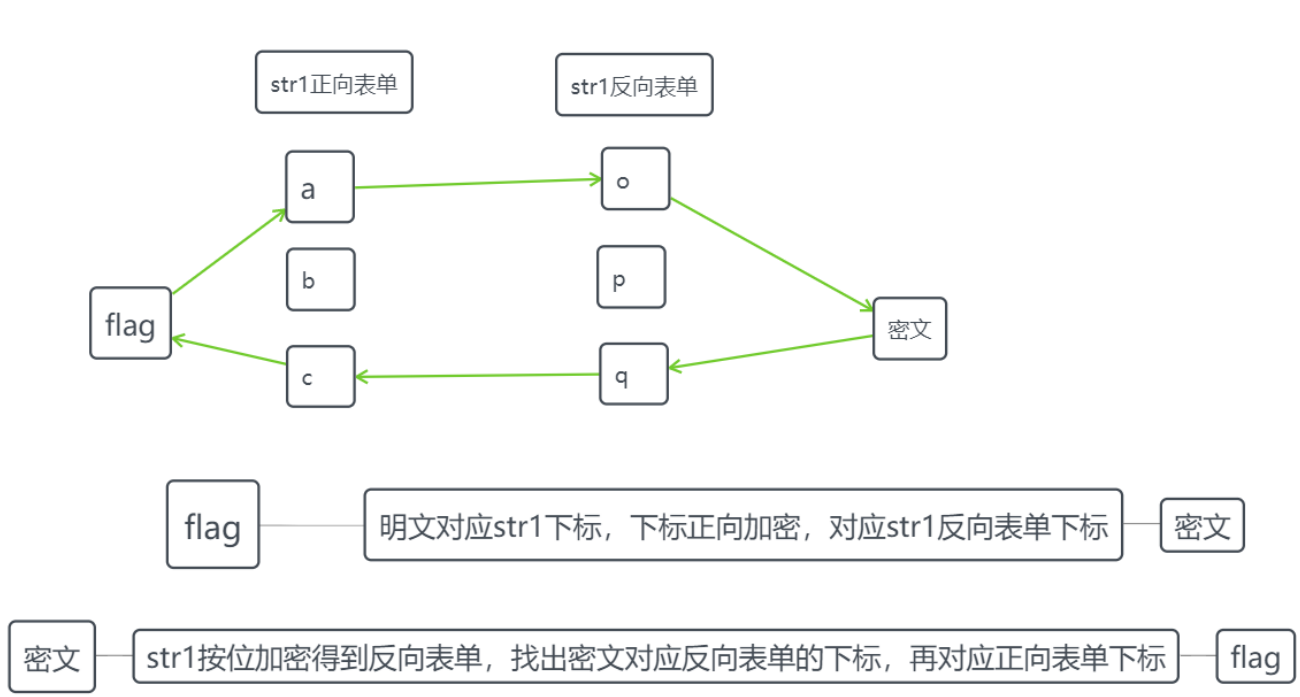
总结：

1： (这里积累第一个经验)
因为下标加密是求余运算加密，所以上网找了一下求余运算的 逆运算 ，结果发现负数的求余运算我解决不了。。。(哭~)



2：
(这里积累第二个经验)
一开始我写的脚本是这样的，在 ASCII 的 32~127 中找到加密后与密文对应的字符，然后取 chr(i) ，然后错误就比较明显了，因为 for 循环 会一直 向前 走，如果密文不按顺序摆放就会 跳过 ，所以没法用 flag+=chr(i) 的方法来获取密文，结果就是得到明文字符----密文字符的对应，然后要自己一个个 拼接 ：
.
后来想了想，竟然 for循环 一直往前走，但是如果我换一下 顺序 ，每个密文字符遍历一次 ASCII 字符，那不就密文也 往前走 了吗？虽然这样的算法复杂度大大增加，但是对电脑来说根本不值得一提好吧：

3:
（这里积累第三个经验）
这里补充一些别人的做法，他直接把 码表加密 ，之后按位找。是一种比较新颖的方法，是对正向反向下标搞操作，值得研究，我感觉他的逻辑应该是这样的：



.
还来有一天发现了是仿射密码：

仿射密码

仿射密码也是一个单表替换密码，字母表中的每个字母相应的值使用一个简单的数学函数映射到对应的数值，再把对应数值转换成字母。
仿射密码的加密函数是 $e(x) = ax + b \pmod m$，其中：

- a和m互质。
- m是字母的数目。

解密函数是 $d(x) = a^{-1}(x - b) \pmod m$，其中 a^-1 是 a 在集合 Z 上的逆元，满足 a * a^-1 ≡ 1 mod 26
求集合Z上某个数的逆元：

```
gmpy2.invert(x, Z)
```


解毕！敬礼！