# Crypto_Lab1

# Task 1

由源码可以知道，`key`是长度在15-30，每一项的值在1-97之间的一个列表，由于实际上我们的文章很长……,这个`key`一定是循环出现的。所以先找找`the`，即重复频率比较高的三个字母
发现`{gY`出现的最多，说明在几个间距的公约数里是`key`的长度，所以先求间距：

`<crypto_task_1>`

```
import math
f=open("cipher.txt",encoding='utf-8')
content=f.read()
f.close()
s=[]
for i in range(len(content)):
    if content[i:i+3]=='{gY':
        s.append(i)
dif=[]
for i in range(len(s)-1):
    dif.append(s[i+1]-s[i])
gcd=dif[0]
for i in range(1,len(dif)):
    gcd=math.gcd(gcd,dif[i])
print(gcd)
```

```
[Running] python -u "d:\MyRepository\slowist-notebook\docs\Coding\CTF\crypto-lec1lab1\crypto_task_1.py"
29

[Done] exited with code=0 in 0.096 seconds
```

因此恰好算出`key`的长度是`29`

由此，计算`the`对应的三个key值。我们已经知道`key`的长度是29,所以新建一个长度是29的空列表，并进行计算

```
maybe_the='{gY'
out=[text_list.index(i) for i in maybe_the] #encrypted data
index=[text_list.index(i) for i in 'the'] #original data
key = [(out[i]*pow(index[i],-1,97))%97 for i in range(3)]
print(out,index)
print(key)
```

这样我们就拿到了3位`key`值
接着理论上应该开始一一较对，但是我发现仅靠校对完全无法恢复原文，所以又做了一些尝试qwq

我们已经猜测`{gY`是`the`,那么继续找高频出现的三个字符的词；

```
def bubble_sort_descending(arr,lst,idx):
    n = len(arr)
```

```
    for i in range(n):
        for j in range(0, n-i-1):
            if arr[j] < arr[j+1]:
                arr[j], arr[j+1] = arr[j+1], arr[j]
                lst[j], lst[j+1] = lst[j+1], lst[j]
                idx[j], idx[j+1] = idx[j+1], idx[j]
for i in range(len(content)):
    if content[i:i+3] not in s_all:
        s_all.append(content[i:i+3])
        s_count.append(1)
        idx.append(i)
    else:
        s_count[s_all.index(content[i:i+3])]+=1
bubble_sort_descending(s_count,s_all,idx)
print(s_all[:10])
print(s_count[:10])
```

通过这个算法，我统计出了出现频率最高的前十位字符，至少说明他们有可能也是`the`，据此再用类似的方法进行计算。在数数量的时候，我们知道了也有一些词组的出现次数在4次以上，因此也可以把他们当成`the`的“备选项”，用和`{gY`类似方法：

```
lst_the=s_all[:10]
keylst=[0 for _ in range(29)]
for m in lst_the:
    if m==' {g' or m =="gY ":
        continue
    print(m)
    print((idx[s_all.index(m)])%29)
    s=[]
    for i in range(len(content)):
        if content[i:i+3]== m:
            s.append(i)
    dif=[]
    for i in range(len(s)-1):
        dif.append(s[i+1]-s[i])
    gcd=dif[0]
    for i in range(1,len(dif)):
        gcd=math.gcd(gcd,dif[i])
    if gcd%29!=0:
        break
    maybe_the= m
    out=[text_list.index(i) for i in maybe_the] #encrypted data
    index=[text_list.index(i) for i in 'the'] #original data
    # key = [(out[i]*pow(index[i],-1,97))%97 for i in range(3)]
    key = [(out[i]*pow(index[i],-1,97))%97 if out[i] != 0 and index[i] != 0 else 0
for i in range(3)]

    print(out,index)
    print(key)
    for i in range(3):
        keylst[(idx[s_all.index(m)])%29+i]=key[i]
print(keylst)
```

由此，我们得到了第一版简陋的`key`：

```
[0, 0, 0, 90, 67, 43, 0, 0, 0, 0, 0, 87, 76, 0, 0, 80, 13, 72, 0, 0, 92, 10, 0, 0,
0, 0, 0, 0, 0]
```

但是这个`key`的可信度还是很低，看看能不能再猜出几个字母，于是尝试还原一下这段文本：

```
text_list=' !"#$%&\'()*+,-./0123456789:;<=>?
@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\\]^_`abcdefghijklmnopqrstuvwxyz{|}~\t\n'

def mod_inverse(a, m):
    # Find the modular inverse of a under modulo m
    for x in range(1, m):
        if (a * x) % m == 1:
            return x
    return 1

def decrypt(s, k):
    out = ''
    for i in range(len(s)):
        index = text_list.index(s[i])
        key_value = k[i % len(k)]
        if key_value == 0:
            key_value = 1
        inverse_key = mod_inverse(key_value, 97)
        index *= inverse_key
        index %= 97
        out += text_list[index]
    return out


key = [0, 0, 0, 90, 67, 43, 0, 0, 0, 0, 0, 87, 76, 0, 0, 80, 13, 72, 0, 0, 92, 10,
0, 0, 0, 0, 0, 0, 0]

cipher = open('cipher.txt', 'r').read()
plain = decrypt(cipher, key)
open('plain_decrypted.txt', 'w').write(plain)
```

```
ur the ),yPUjYbteeSYB o:|k'y^> %he f@XP gjzbbox] BB" [CP       ;o the w*)`5L
8 laS1[B^:L *-< T3e wF0 UQ5y  onlO /V^R|C.]X ,y af:@/f het dieH H[ vJA.]V+r citm(.a( jY the qUj?:7 K&V0}p. T\@ m?i
Gfade 1!V{ <BP3 0F  grrUW\ H[ Kiti3G, L.[ 6c\ :ped m{ \FhV  A, t{}i6(-L S]W Fyspif|[U' BY& by GH~6    jA;V+o|loo.eZK
"jG% de{*i6+ )C %|;   @ meF!{ Y!VJT fiZCF Gp7 *-<0}Z.Aff@X QBV  iviw ]Bx ~qsYFqLK65)* 0U 5Lt was kuV" 1j
y7nqVgerF!. Y!Vhyht W&!;N )k cz^G cam) ,aQH ktuse^`oQ 9nh( m5}c be:lX) f79w, h{*\ ?L[ .q7    yld m{ &0A 7trk,
xBja"7huk8EEf any pF,hj\tre, &UQ v|hQ&8EqI of !]FQ ;V!d iS K'+3?C c:i {hicF")* AVY} to
<~%d)hy xTq them? rkU DAe. BC5; L,A j76GMe pr0'mwoV peca6Gi G |h6 8T:peho?O hFUJt8ienWnF ?L[ .q7dG}, a ]
X)?[6Jtr o! he6 Xj9    |! Zefrm".DNh'MI haE /V6p B#47!, d.
v|+m~O
8 ef!U"j6p1 .q7dG} waU ZrQ BJ easO <; E[ L:Xs, now 0&"oH;t` In YBV :,~uI !Tcete){W\ "VY}uryk he6 qC[;z;o*e o: W\0 ,k%
sicZ H[ L[-Pt hFVch 8|' 0u;t8tiaw h' " Db:7!a  of ?.:`5`tfatiANF {"n A-WE+  ntaD7z  BV KommANuV(-[ #d<EGc thF! fJi }bst
LJiG<K    S
 T9  thF! "`iJt8ted YBV Rth %|T+ melfeZK ABJ of W`[6+: }.
<1p  n, :lX sh >4s t^n ~6d1B#, T* the e/) he
} pel
H6u:7 P37 ayolia"zpKiJtfthewnu;N [-AO^ }uforf0 fF VztnomLdi %.[ .]w}:led 8?0"o5"n the U"V R| *OV!p  ts, U]m?B bbpt
HCi %.[ Hy{O loinK ,f( oVp. NAY E(3)V ]71q the @Zy H[ }he SUUV?:[CP3 6}cturZ  mh jY3entA5u G.fBq47 ,@e d)8,hNhV
palaSJi <@ B#
}}E%ion |Zy "jMKulaHUH" p[h9  W *yr aa .:Y5Lhbnt LJiG<K3ZH1VQiOAA{m9[mdi8Fwef1>}L6Bwd^'3aS.)
Buf 0U iBM!y aZ f7=1L S] E9*enirs' "<68!and
<6u:Jp i8T+hs MrlX)' e
```

还原之后看效果，发现文章依旧是乱码，根本不可以继续猜下去。

设想：假设能多用几个特征词，从而确定 `key` ,这个状况可能可以一定程度的解决，可惜我想不出来qwq

# Task 2

复现：

```
sage: MT=Matrix(Zmod(256),[[49,47,49],[31,22,25],[18,4,35]])
sage: IT=MT.inverse()
sage: IT
[ 78 111   9]
[205 233  22]
[112  26 205]
sage: flag="AAA{TESTFLAGTESTFLAGTESTFLAGT}"
sage: for i in range(3):
....:     for j in range(10):
....:         FT[i,j]=ord(flag[i+j*3])
....: RT=MT*FT
sage: RT
[209  44 181  18 162 122 142 134 203 240]
[206 218  27 185  53 156  45 177  91   6]
[121 101 184  17  85 100  42 162 255 101]
sage: FT
[ 65 123  83  76  84  84  65  69  70  71]
[ 65  84  84  65  69  70  71  83  76  84]
[ 65  69  70  71  83  76  84  84  65 125]
sage: FT_2=IT*RT
sage: FT_2
[ 65 123  83  76  84  84  65  69  70  71]
[ 65  84  84  65  69  70  71  83  76  84]
[ 65  69  70  71  83  76  84  84  65 125]
sage:
```

```
sage: MT=Matrix(Zmod(256),[[49,47,49],[31,22,25],[18,4,35]])
sage: IT=MT.inverse()
sage: IT
[ 78 111   9]
[205 233  22]
[112  26 205]
sage: flag="AAA{TESTFLAGTESTFLAGTESTFLAGT}"
sage: for i in range(3):
....:     for j in range(10):
....:         FT[i,j]=ord(flag[i+j*3])
....: RT=MT*FT
sage: RT
[209  44 181  18 162 122 142 134 203 240]
[206 218  27 185  53 156  45 177  91   6]
[121 101 184  17  85 100  42 162 255 101]
sage: FT
[ 65 123  83  76  84  84  65  69  70  71]
[ 65  84  84  65  69  70  71  83  76  84]
[ 65  69  70  71  83  76  84  84  65 125]
sage: FT_2=IT*RT
sage: FT_2
[ 65 123  83  76  84  84  65  69  70  71]
[ 65  84  84  65  69  70  71  83  76  84]
[ 65  69  70  71  83  76  84  84  65 125]
sage:
```

一些解题思路：

由所给的代码，从result反推，知道 `RT` 是一个可以推断出来的矩阵：

```
result =
b'\xfc\xf2\x1dE\xf7\xd8\xf7\x1e\xed\xccQ\x8b9:z\xb5\xc7\xca\xea\xcd\xb4b\xdd\xcb\xf2
\x939\x0b\xec\xf2'
RT = [[0] * 10 for _ in range(3)]

index = 0
for i in range(10):
    for j in range(3):
        RT[j][i] = result[index]
        index += 1

# 打印恢复的 RT
for row in RT:
    print(row)
```

得到的RT：

```
[252, 69, 247, 204, 57, 181, 234, 98, 242, 11]
[242, 247, 30, 81, 58, 199, 205, 221, 147, 236]
```

```
[29, 216, 237, 139, 122, 202, 180, 203, 57, 242]
```

假设我能知道密钥`MT`,我的明文解密方法如下:

```
from sage.all import *

MT = matrix(Zmod(256), [
    [a11, a12, a13],
    [a21, a22, a23],
    [a31, a32, a33]
])
MT_inv = MT.inverse()
result = b'...'
RT = matrix(Zmod(256), 3, 10)
for i in range(10):
    for j in range(3):
        RT[j, i] = result[i * 3 + j]
FT = MT_inv * RT
flag = ''
for j in range(10):
    for i in range(3):
        flag += chr(FT[i, j])

print(flag)
```

现在知道了flag的前四个字符和最后一个字符，由于`FT = MT_inv * RT`即`MT*FT=RT`。根据算法中:

```
for i in range(3):
        for j in range(10):
                FT[i, j] = ord(flag[i + j * 3]) #FT储存的是ord值
```

知道`FT`的一维数组的第一项都是65的倍数，$(k = 65) \Rightarrow$

$$\begin{cases} a_{11} + a_{12} + a_{13} = 252 \\ a_{21} + a_{22} + a_{23} = 242 \\ a_{31} + a_{32} + a_{33} = 29 \end{cases}$$

这样得到了三行和，遍历的范围就缩小了;

```
from sage.all import *
cnt=0
for a11 in range(252):
    for a12 in range(252):
        a13 = (252 - a11 - a12)
        for a21 in range(242):
            for a22 in range(242):
                a23 = (242 - a21 - a22)
                for a31 in  range(29):
                    for a32 in  range(29):
                        a33 = (29 - a31 - a32)
                        # Create the matrix
                        MT = Matrix(Zmod(256), [[a11, a12, a13], [a21, a22, a23],
```

```
 [a31, a32, a33]])
                        #...解密算法
```

但是这里的优化应该还不够多，所以这个程序还是没有运行结果qwq
一些奇怪的思路：我觉得可能需要用到可逆这个条件，但是似乎想不出来有哪条性质可以继续限制矩阵的值，如果分别判断是否可逆的话，计算量其实也没有减小qwq