

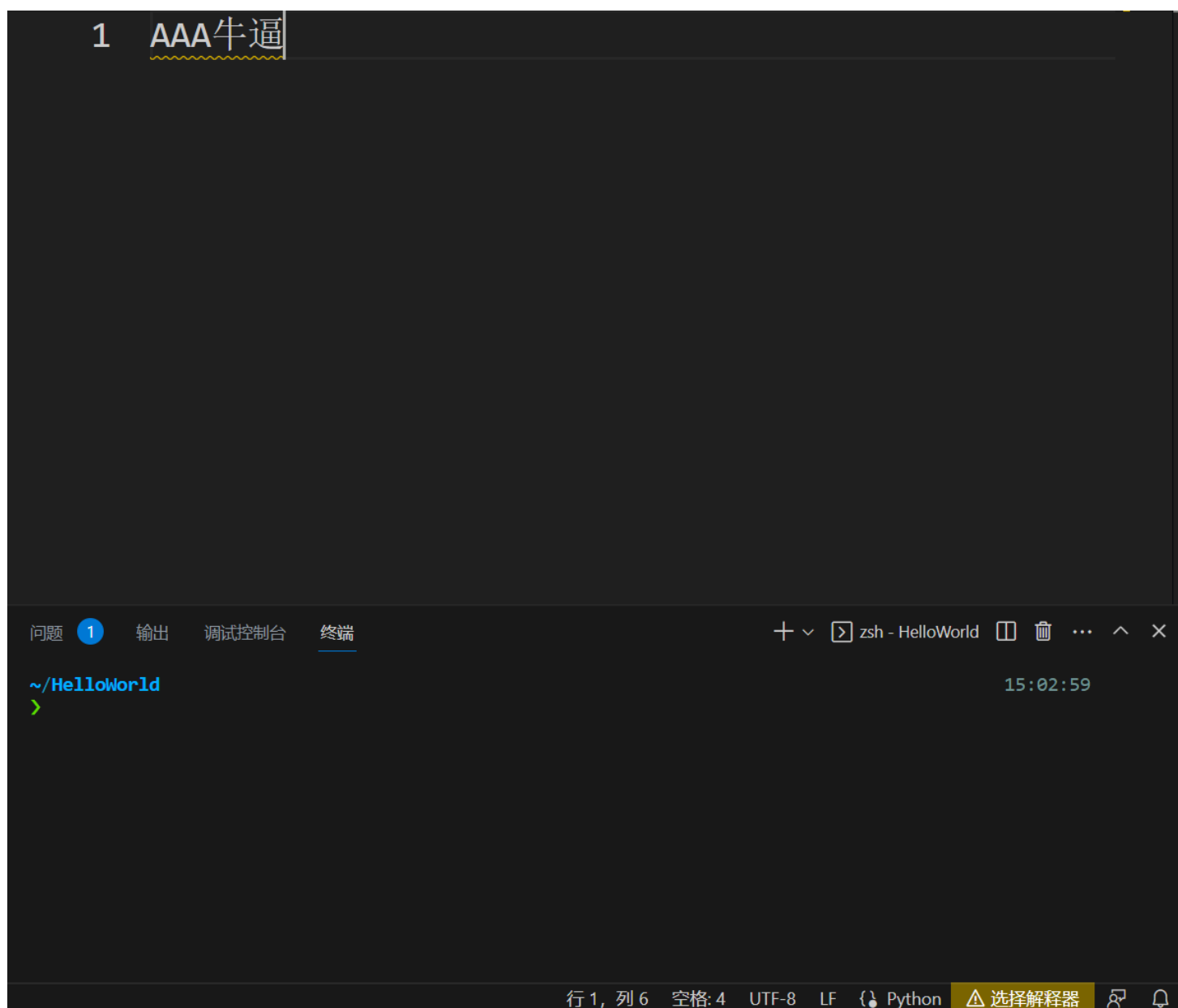
Misc Lab1

3220102732-周伟战

1. Task1

1.1.1复现6种编码乱码的情况

1) 用 GBK 解码 UTF-8 编码的文本



The image shows a Jupyter Notebook interface. The main area is a code cell with the text "1 AAA牛逼". The text "AAA牛逼" is underlined with a yellow wavy line. Below the code cell is a terminal window. The terminal window has a tab labeled "zsh - HelloWorld". The terminal output shows the prompt "~ / HelloWorld" and a green cursor. The terminal window also shows the time "15:02:59". At the bottom of the Jupyter Notebook interface, there is a status bar that says "行 1, 列 6 空格: 4 UTF-8 LF Python 选择解释器".

```
1 AAA牛逼
```

问题 1 输出 调试控制台 终端

zsh - HelloWorld

~/HelloWorld 15:02:59

行 1, 列 6 空格: 4 UTF-8 LF Python 选择解释器

1 AAA鏰€?

问题 3 输出 调试控制台 终端

+ zsh - HelloWorld

~/HelloWorld
>

15:02:59

行 1, 列 8 空格: 4 GBK LF Python 选择解释器

2) 用UTF-8 解码 GBK 编码的文本

CTF > Lab1 > Misc1.py

```
1 AAA牛逼
```

问题 1 输出 调试控制台 终端

~/HelloWorld 15:02:59

行 1, 列 6 空格: 4 GBK LF Python 选择解释器

CTF > Lab1 > Misc1.py

```
1 AAAt??
```

问题 3 输出 调试控制台 终端

~/HelloWorld 15:02:59

行 1, 列 7 空格: 4 UTF-8 LF Python 选择解释器

3) 用 latin-1 解码 UTF-8 编码的文本

Input

+

AAA牛逼

REC 51

Raw Bytes← LF

Output

AAAç•••é•¼

REC 91

2ms

ISO-8859-1 Latin 1 We...← LF

4) 用 latin-1 解码 GBK 编码的文本

Input

+

AAA牛逼

REC 51

Tt Simplified Chinese GBK ← LF

Output

AAAÅ£±Æ

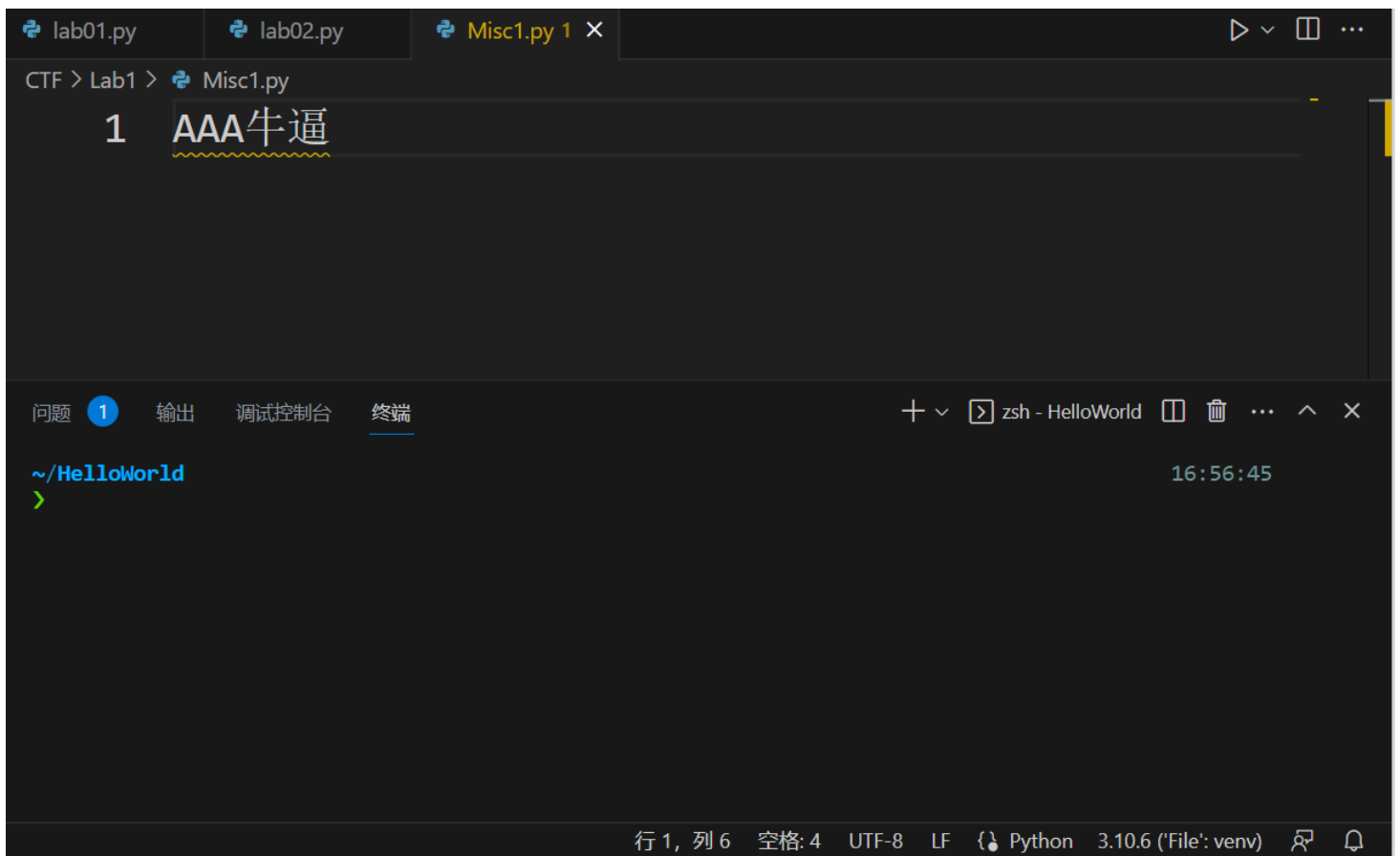
REC 71

2ms

Tt ISO-8859-1 Latin 1 We... ← LF

5) 先用 GBK 解码 UTF-8 编码的文本，再用 UTF-8 解码前面的结果

UTF-8的原码



通过GBK重新打开后解码



然后再保存后重新利用UTF-8打开

```
CTF > Lab1 > Misc1.py
1 AAA牛??
```

问题 3 输出 调试控制台 终端

~/HelloWorld 16:56:45

>

行 1, 列 7 空格: 4 UTF-8 LF Python 3.10.6 ('File': venv)

6) 先用 UTF-8 解码 GBK 编码的文本，再用 GBK 解码前面的结果

```
CTF > Lab1 > Misc1.py
1 AAA牛逼
```

问题 1 输出 调试控制台 终端

source /home/zwz123/HelloWorld/File/bin/activate
~/HelloWorld 20:16:50
> source /home/zwz123/HelloWorld/File/bin/activate

~/HelloWorld 20:16:50>HelloWorld 20:16:50

>

行 1, 列 6 空格: 4 GBK LF Python 3.10.6 ('File': venv)

再重新用UTF-8编码打开该GBK文件

CTF > Lab1 > Misc1.py

```
1 AAAt??
```

问题 3 输出 调试控制台 终端

```
source /home/zwz123/HelloWorld/File/bin/activate
~/HelloWorld 20:16:50
> source /home/zwz123/HelloWorld/File/bin/activate

~/HelloWorld 20:16:50
>
```

行 1, 列 7 空格: 4 UTF-8 LF Python 3.10.6 ('File': venv)

再进行重新编码保存，并用GBK重新打开得到AAA牛银斤拷

CTF > Lab1 > Misc1.py

```
1 AAA牛银斤拷
```

问题 1 输出 调试控制台 终端

```
source /home/zwz123/HelloWorld/File/bin/activate
~/HelloWorld 20:16:50
> source /home/zwz123/HelloWorld/File/bin/activate

~/HelloWorld 20:16:50
>
```

行 1, 列 8 空格: 4 GBK LF Python 3.10.6 ('File': venv)

1.2.1回答以下问题

Q:在自行研究了 GB 系列编码后，请阐述 GB 系列是如何实现三个版本兼容的？

(GB系列实现的是GB18030兼容的三个版本

1: GBK

2: GB2312

3: ASCII)

实现了GB2312对于ASCII码的兼容

(1) 首先是GB2312是可以叫做区位码，由两个字节组成，第一个字节是区，第二个字节是位。（分别是高低位，“高位字节”使用了0xA1-0xF7（把01-87区的区号加上0xA0），高位字节有部分为空区，“低位字节”使用了0xA1-0xFE（把01-94加上0xA0））他们分别选取了94个bit，那么GB2312就可以包含 $94 \times 94 = 8836$ 个字符。GB2312为了兼容ASCII码，把小于等于127的字符，表示为ASCII，单个字节，依旧以0开头，与原本的意义相同。但其他的字符都是两个字节，而且都以1开头，为了与ASCII码区分开来。

如：我叫ABC 这句话用GB2312码来表示：11001110 11010010（我） 10111101 11010000（叫） 01000001（A） 01000002（B） 01000003（C）这样就达到了兼容ASCII码的效果；

GBK兼容了GB2312和ASCII码

(2) GBK编码是微软在GB2312的编码规则基础上，利用GB2312未编完的空间，共收录21886个汉字和图形符号，其中汉字（包括部首和构件）21003个，图形符号883个。这样下来，GBK编码向下完全兼容GB2312编码，也等同于兼容ASCII码。

GB18030兼容GBK编码

(3) GB18030兼容GBK编码：采用变长多字节编码，每个字可以由1个、2个或4个字节组成。具有这样的特点，使得其可以包含161w的字符。1个字节的是保留了ASCII码的编码规则，而2个字节的是对于GBK编码的保留与拓展，而4字节的是用于包含所有Unicode编码下的UTF四字节区段。从而做到了GB18030兼容3个版本。

Bonus

Task2

破解Vigenere

Step1 爆破the length of key

代码在3220102732-周伟战-Misc基础-attachment.zip中的Vigenere-decrypt01.py中

```

#Vigenere-decrypt1 use for get the length of key
from random import randrange

text_list=' !"#$%&\'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz'
#key 是从text_list中随机选择15-30个的字符 生成的密钥
def TextRead(file):
    #将txt文件转化为字符串
    with open(file,"r") as f :
        TextContent = f.read()
    return TextContent

def Num(s):
    #已经是分组之后取出的一组字符串 , 计算单组中的重合指数
    count=[0]*97
    Num = 0
    for i in range(len(s)):
        index=text_list.index(s[i])
        count[index]+=1
    for i in range(97):
        if count[i]!=0:
            Num+=count[i]*(count[i]-1)
    Num/=len(s)*(len(s)-1)
    return Num

def Ic(s,k):
    #用来统计不同k值下的重合指数
    Ic=[]
    str=""
    for i in range(len(s)//k):#组数
        for x in range(i,len(s),k):
            str+=s[x]
        p = Num(str)
        Ic.append(p)
    return Ic

T=TextRead("vigenere.txt")

for k in range(15,31):
    p=0 #p用于改变存放每个k值下的Ic值总合
    for x in range(k):
        p+=Ic(T,k)[x]
    p/=k
    print (p)
#C[i]中存放的是Ic的均值

```

这个代码段是利用求重合指数来获得最接近0.038的那一组密钥长度.
 以下是获得的重合指数数据

0.03459784961078679
0.036255956720073736
0.03344268791214023
0.03609708271185117
0.03520982960248611
0.034196049891084775
0.03383462158072611
0.034262435691612865
0.03830053002394148
0.036043758485237894
0.03647059166610666
0.0347398741452819
0.03250182725535983
0.0338630034887915
0.03771683757058582
0.034791778064769595

很明显可以看到长度为29的时候是最贴近0.038，于是我们拿密钥长度为29爆破ng猜

Step2 利用length=29爆破密钥

总体思路是把这篇TOEFL按照key的长度分成29组，然后对每一组的字符串的字符出现频率与字母出现频率表进行比对，通过获得密钥&&验证密钥的反复比对，来获得准确的密钥。（分别是

[Vigenere-decrypt4.py](#)&&[Vigenere-verity.py](#)）

如果利用密钥验证出现 其他非26个字母的字符就很自然的知道是错误的

Vigenere-decrypt2.py进行分组

```
def TextRead(file):
    #将txt文件转化为字符串
    with open(file,"r") as f :
        TextContent = f.read()
    return TextContent

Text=TextRead('vigenere.txt')
Text=list(Text)
Str=''
i=29
for y in range(i,len(Text),29):
    Str+=Text[y]
print(Str)
```

以第二组字符串为例:

```
r oT,r}F:Fo }LqG} Eq qG:G ,G,ToFa+G9+pa}:p q},Ei.9+8Ga :TG+r*},G8T9E} T~9o EEEa}8 } ,V +} }  
TT~H }},:}EF9 G9~ oawaN}} r qEG,~oG99} G
```

然后以下是统计结果 (频率降序排列)

```
Character ' ' appears 0.49777777777777776  
Character '}' appears 0.07555555555555556  
Character 'G' appears 0.05333333333333334  
Character 'E' appears 0.03555555555555556  
Character '9' appears 0.03555555555555556  
Character 'T' appears 0.03111111111111111  
Character ',' appears 0.03111111111111111  
Character 'o' appears 0.02666666666666667  
Character 'a' appears 0.02666666666666667  
Character ':' appears 0.02222222222222223  
Character 'q' appears 0.02222222222222223  
Character '+' appears 0.02222222222222223  
Character 'r' appears 0.01777777777777778  
Character 'F' appears 0.01777777777777778  
Character '~' appears 0.01777777777777778  
Character '*' appears 0.01333333333333334  
Character '8' appears 0.01333333333333334  
Character 'p' appears 0.00888888888888889  
Character 'L' appears 0.0044444444444444444  
Character 'i' appears 0.0044444444444444444  
Character '.' appears 0.0044444444444444444  
Character 'V' appears 0.0044444444444444444  
Character 'H' appears 0.0044444444444444444  
Character 'w' appears 0.0044444444444444444  
Character 'N' appears 0.0044444444444444444
```

概率接近0.50的 ' ' 的可以忽略, 然后对于0.076的 '}' 开始利用e来尝试

字母频率出现降序排列 e t a o i n s

```
Text_List=' !"#$%&\'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz  
for key in range(97):  
    if (Text_List.index('e')*key)%97==Text_List.index('}'):   
        print(Text_List[key])
```

在Vigenere-decrypt4.py带入e的时候获得密钥是 '.' 于是利用这个密钥，以及 [Vigenere.verity.py](#) 中的频率降序字母表进行反复验证。

```
Text_List=' !"$%&\'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz'
def gcd(a,b):
    #用于计算最大公因数,默认a>=b
    if b==0:
        return a
    else:
        return gcd(b,a%b)

def inverse(a,n):
    #用来判断a的乘法逆元是否存在, 如果存在则输出
    b,y=0,0
    if gcd(n,a)==1:
        while y <=0:
            if (1-n*y) % a==0:
                b= (1-n*y)//a
                return b
            else:
                y-=1
    else:
        return 0

key=Text_List.index('.')
p=(Text_List.index('G')*inverse(key,97))%97
print(Text_List[p])
```

用 key='.' 这个密钥

验证 'G' 发现得到的是 'o'
验证 'E' 发现得到的是 'a'
验证 'g' 发现得到的是 'n'

很明显的发现符合上述的字母出现频率表的规律，于是得到key=2这一组的密钥为 '.'
同理对其他的28组密钥进行推测，当然每组都能推出来不太现实，大概推出到18-20组左右的时候，就能利用反推文章，利用常见的单词拼写错误，来反推key。

在比较长的推理过程中 我得到同理得到后面的X . 6 z c K 1 r P 1 l A y ~ X X X h o 2 a X X & D G X
' X

然后得到反推后的英文原文(篇幅原因，选了第一段):

uy thz mid-nin^;a/3j8 oent'i^> the =erm "ic^J5D 8Q" enP{[;d thz AmericaQ i?3S}Q^e, *G< ice NQs
still /&t(6W^inn.vX to ab(ect the (</- {(ord.vV+y ci5]zens in ;-/ Q:]?ed K2V0es. -+e ice tV[R/ S|W{
wiPY 0he gefwth of +4

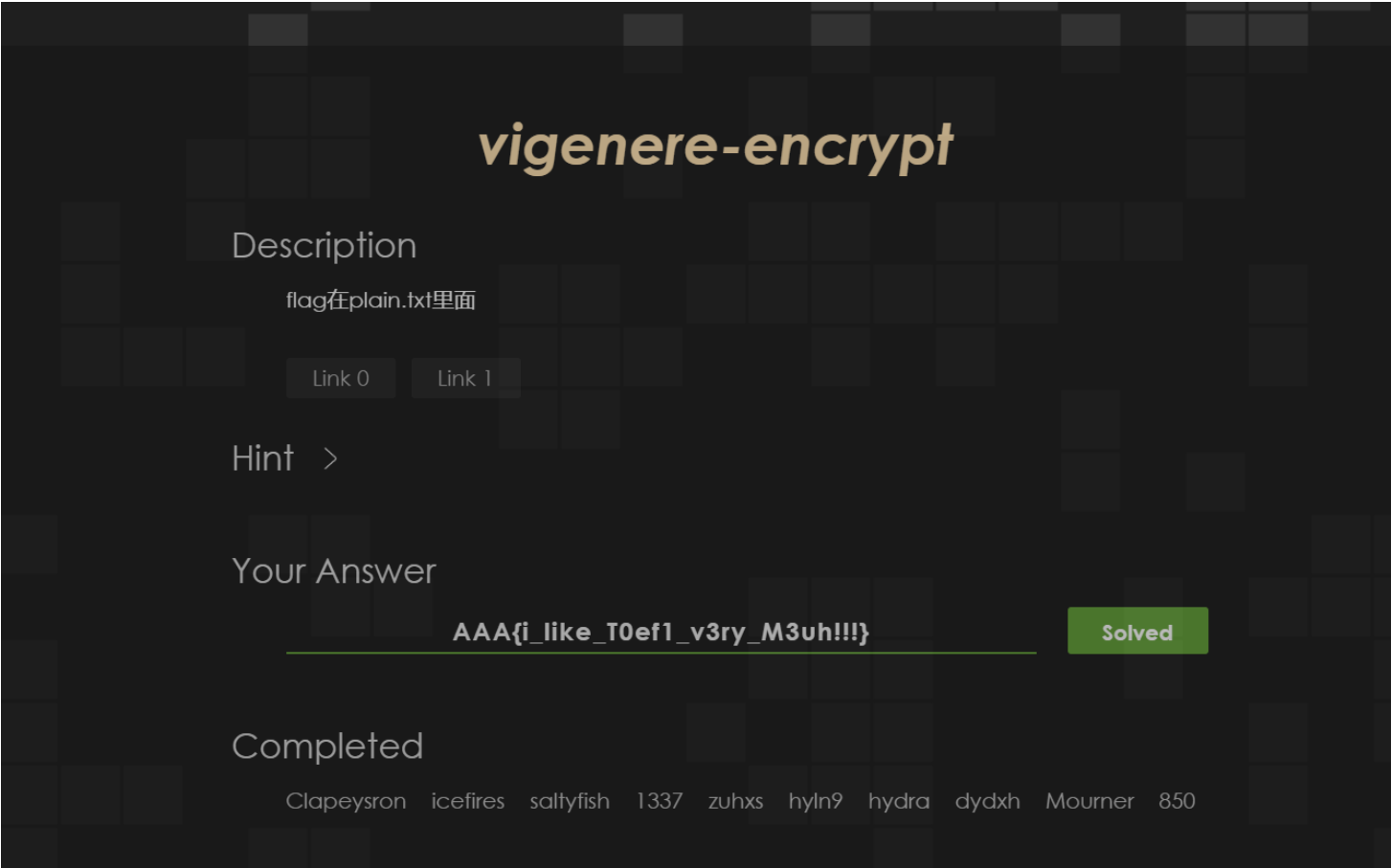
很明显的发现，已经有一些的单词是完整的了，如 still , ice ,像 AmericQ 就明显是 America

这种反推得到最后的密码为

key= ' S.6zck1r,P1IAy~\\p-h02aR4&DGX\'a' (注意转义符)



得到Flag是AAA{i_like_T0ef1_v3ry_M3uh!!!}



3.Task3

OSINT

3.1 TonyCrane.jpg

利用exiftool解析TonyCrane.jpg

```
D:\ZJU\【CS】\Capture The Flag\lab1基础\misc>exiftool TonyCrane.jpg
ExifTool Version Number      : 12.64
File Name                    : TonyCrane.jpg
Directory                   : .
File Size                    : 3.1 MB
File Modification Date/Time   : 2023:07:19 20:32:02+08:00
File Access Date/Time        : 2023:07:20 17:40:53+08:00
File Creation Date/Time      : 2023:07:19 20:32:01+08:00
File Permissions              : -rw-rw-rw-
File Type                    : JPEG
File Type Extension          : jpg
MIME Type                    : image/jpeg
Exif Byte Order               : Big-endian (Motorola, MM)
Resolution Unit               : inches
Make                         : Xiaomi
Camera Model Name             : M2102K1C
Modify Date                   : 2023:02:09 21:28:31
Orientation                   : Horizontal (normal)
X Cb Cr Positioning           : Centered
ISO                           : 550
Exposure Program              : Not Defined
F Number                      : 2.0
Exposure Time                 : 1/13
Sensing Method                : Unknown (0)
Sub Sec Time Digitized        : 122
Offset Time Original          : +09:00
Sub Sec Time Original         : 122
Offset Time                   : +09:00
Sub Sec Time                  : 122
Focal Length                  : 7.6 mm
Flash                         : Off, Did not fire
Light Source                   : D65
Metering Mode                 : Center-weighted average
Scene Capture Type            : Standard
Interoperability Index        : R98 - DCF basic file (sRGB)
Interoperability Version      : 0100
Focal Length In 35mm Format    : 0 mm
Max Aperture Value            : 1.9
Create Date                   : 2023:02:09 21:28:31
Exposure Compensation         : 0
Exif Image Height             : 1800
White Balance                  : Auto
Date/Time Original            : 2023:02:09 21:28:31
Brightness Value              : 0
Exif Image Width              : 4000
Exposure Mode                  : Auto
Aperture Value                : 1.9
Components Configuration      : Y, Cb, Cr, -
Color Space                    : sRGB
Scene Type                    : Unknown (0)
Shutter Speed Value           : 1/12
Exif Version                   : 0220
Flashpix Version              : 0100
GPS Latitude Ref               : North

GPS Longitude Ref              : East
GPS Altitude Ref              : Above Sea Level
GPS Time Stamp                 : 12:28:33
GPS Processing Method          : GPS
GPS Date Stamp                 : 2023:02:09
Y Resolution                   : 72
X Resolution                   : 72
Thumbnail Offset               : 8898
Thumbnail Length               : 14058
Compression                    : JPEG (old-style)
XMP Toolkit                    : Adobe XMP Core 5.1.0-jc003
XMP Meta                       : <?xml version="1.0" encoding='UTF-8' standalone='yes' ?><subimage offset="25819" length="25819" paddingx="76" paddingy="1626" width="512" height="116" rotation="-1" /><lenswater
ark offset="56810" length="3099" width="510" height="114" paddingx="78" paddingy="58" isLTR="true" />
Image Width                    : 4000
Image Height                   : 1800
Encoding Process               : Baseline DCT, Huffman coding
Bits Per Sample                : 8
Color Components               : 3
Y Cb Cr Sub Sampling           : YCbCr4:2:0 (2 2)
Aperture                       : 2.0
Image Size                     : 4000x1800
Megapixels                     : 7.2
Shutter Speed                  : 1/13
Create Date                    : 2023:02:09 21:28:31.122
Date/Time Original             : 2023:02:09 21:28:31.122+09:00
Modify Date                    : 2023:02:09 21:28:31.122+09:00
Thumbnail Image                : (Binary data 14058 bytes, use -b option to extract)
GPS Altitude                   : 63.4 m Above Sea Level
GPS Date/Time                  : 2023:02:09 12:28:33Z
GPS Latitude                   : 35 deg 42' 11.67" N
GPS Longitude                  : 139 deg 45' 8.62" E
Focal Length                   : 7.6 mm
GPS Position                   : 35 deg 42' 11.67" N, 139 deg 45' 8.62" E
Light Value                    : 3.2
```

- 拍摄这张图片时所在位置的高度为多少？（以海平面为基准，尽可能精确）
- GPS Altitude : 63.4 m Above Sea Level
- 拍摄这张图片的时间是什么时候？（尽可能精确）
- Create Date : 2023:02:09 21:28:31

3.2 yyy's real OSINT

获得可能是正确的拍照时间

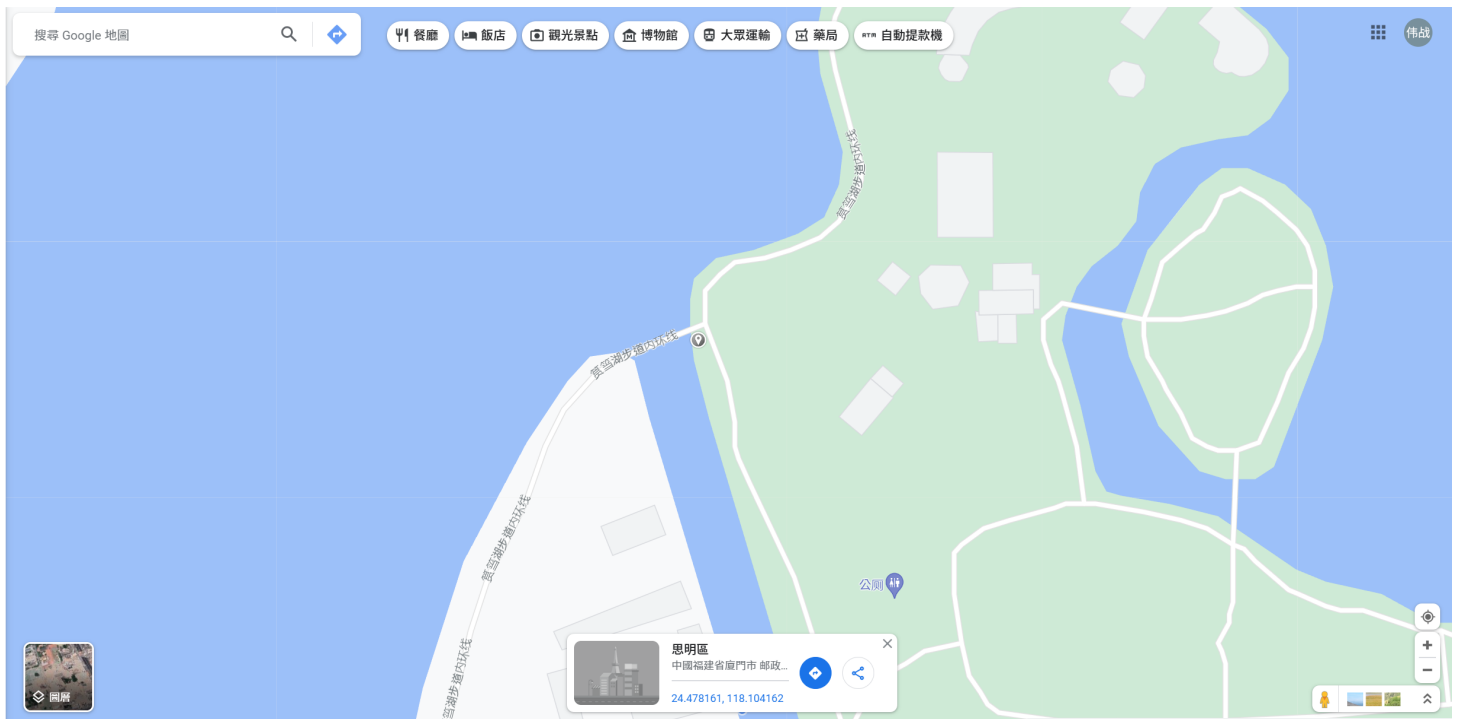
Modify Date : 2022:02:16 16:32:29

其他的GPS Altitude, 经纬度等具体信息都被unknown掉了, 不像第一题, 能够直接exiftool获得具体信息。

所以选择百度识图, 将图片中比较具有特点的建筑进行搜索, 得到了一张图



发现是厦门的南湖公园, 以及地标性建筑, 厦门海上明珠
然后利用小桥, 靠湖等要素, 判断得到大致的位置如下图

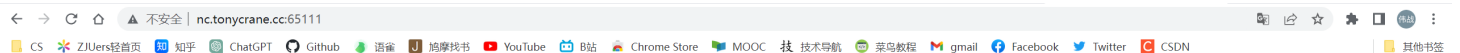


得到经度: 24.478161

得到纬度: 118.104162

加上利用exiftool得到的时间

Modify Date : 2022:02:16 16:32:29



OSINT Result

Latitude (abs. error<=0.006):

24.478161

Longitude (abs. error<=0.006):

118.104162

Time in the day(hours only, 0-24):

16

Month (1-12):

2

Check it! (5chances/10mins)



AAA{tH3_kINg_0F_OP3N_THE_8OX}



进行尝试，获得了flag

AAA{tH3_kINg_0F_OP3N_THE_8OX}

Over