Crypto Lab 1

1. Task 1

这道题目是Vigenere密码的变式，这种密码是对凯撒密码的加强。区别在于：凯撒密码的偏移量是固定的，而Vigenere密码的偏移量周期是由密钥的长度决定的。这里的密钥实际上就是记录偏移量的数组，记数组长度为n，所以将密文等间隔分成n组后可以得到n组凯撒密码。

但这道题目与一般的Vigenere密码有所不同：字符范围不止是26个字母，而是97个字符；加密方式不是简单的加法后取模，而是乘法取模，但由于97是一个质数，所以乘法取模后的值仍旧是比较分立的，事实上处理差不多。

首先，需要求出密钥长度，这里给的密文文本量比较大，可以采用计算密文中相隔密钥长度一共重合了几次的方法来求，最长的那个很有可能就是正确的长度:

```
1 22
2 44
3 150
4 172
5 108
6 94
7 108
8 144
9 121
10 132
11 133
12 137
13 141
14 126
15 140
16 129
17 120
18 136
19 128
20 129
21 137
22 126
23 136
24 147
25 111
26 129
27 141
28 128
29 278
29
```

可见，以29为长度的密钥重合长度远远高于别的长度，因此29就是我们要找的密钥长度。

为了找到确定的密钥，我们很难像原本的Vigenere密码一样分析字母频率，人工破解也是一件工作量很大的事情。但是字符库中引入了很多不是字母的字符，而明文应当是以字母为主的，因此，可以像估计密钥长度一样，对使用碰撞的密钥恢复的明文进行字母计数，保留**字母出现最多**的那个情况，并组合成完整的密钥，以此完成解密。实际测试时一次成功，得到了完整的明文：

（略）

Nevertheless, early efforts to economize ice included wrapping the ice in blankets, which kept the ice from doing its job. Not until near the end of the nineteenth century did inventors achieve the delicate balance of insulation and circulation needed for an efficient icebox.

fLaG:AAA{i_like_T0ef1_v3ry_M3uh!!!}

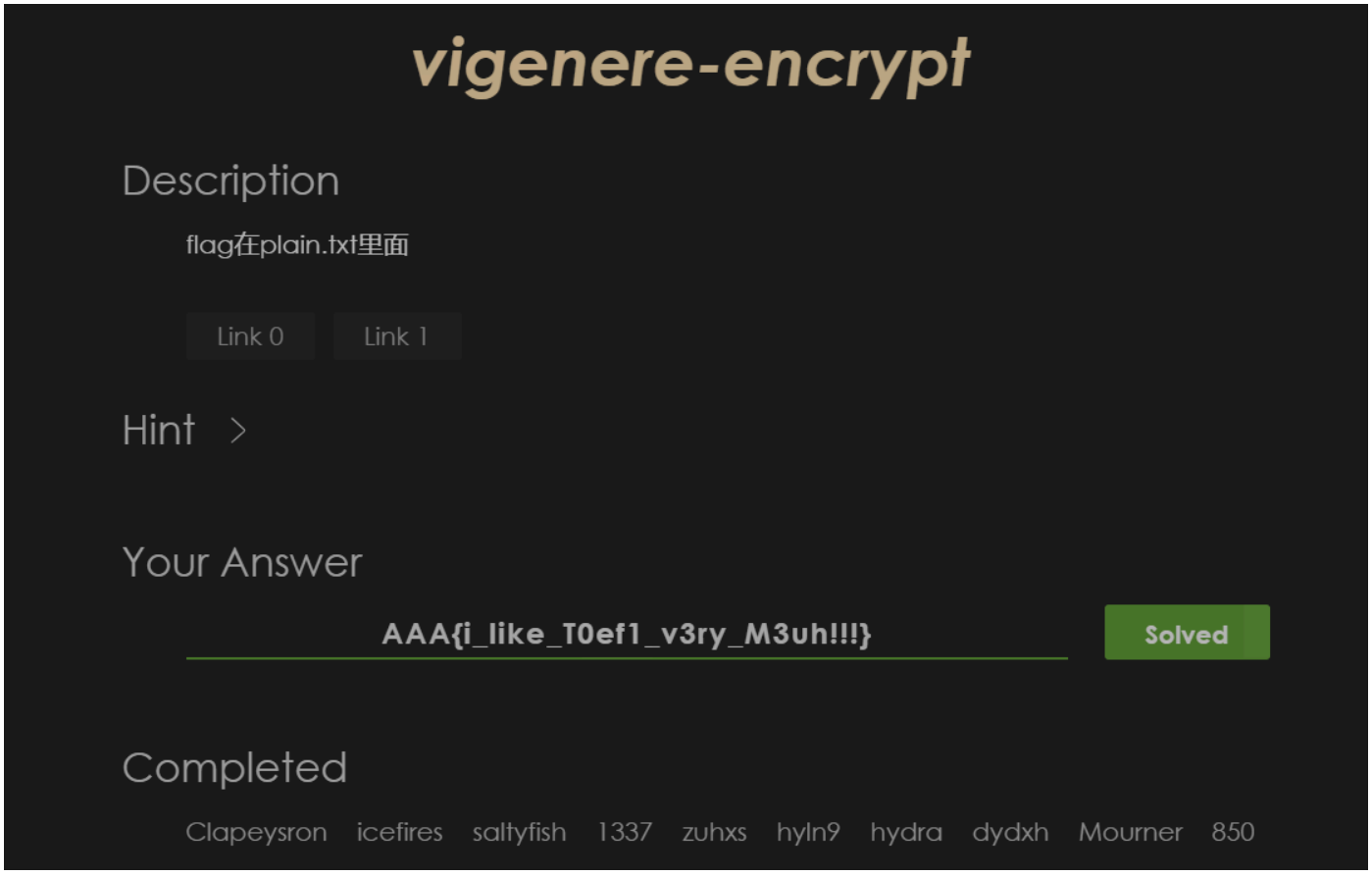
But as early as 1803, an ingenious Maryland farmer, Thomas Moore, had been on the right track. He owned a farm about twenty miles outside the city of Washington, for which the village of Georgetown was the market center. When he used an icebox of his own design to transport his butter to market, he found that customers would pass up the rapidly melting stuff in the tubs of his competitors to pay a premium price for his butter, still fresh and hard in neat, one-pound bricks.

（略）

其中得到的密钥是：[78, 7, 75, 83, 42, 88, 40, 84, 95, 40, 71, 50, 12, 32, 76, 57, 15, 31, 70, 27, 3, 33, 34, 81, 62, 5, 26, 14, 3]

注意：这里的密钥不一定是加密用的那个，密文回到明文的过程可以理解为一种再加密，负负得正。

将flag输入校巴得到验证：



2. Task 2

2.1. Sagemath用法

随机赋值，使其可逆，并求出逆矩阵：

```
encrypted_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'
MT = matrix(Zmod(256), [[77, 72, 35], [72, 211, 43], [17, 117, 75]])
MT_inverse = MT.inverse()
print(MT_inverse)
```

```
[218  47 199]
[ 43  92  33]
[ 61  95 255]
```

随便写了flag，以此生成了FT，计算出对应的RT；用RT和MT计算得到新的FT。通过比对，两者一致。

In [21]:
```
MT = matrix(Zmod(256), [[77, 72, 35], [72, 211, 43], [17, 117, 75]])
assert MT.is_invertible()
flag = "AAA{jasdasasdhasjasvhsahfbask}"
FT = matrix(Zmod(256), 3, 10)
for i in range(3):
    for j in range(10):
        FT[i, j] = ord(flag[i + j * 3])
print(FT)
RT = MT * FT
FT=MT.inverse()*RT
print(FT)
```

```
[ 65 123 115 115 100 115 115 115 102 115]
[ 65 106 100  97 104 106 118  97  98 107]
[ 65  97  97 115  97  97 104 104  97 125]
[ 65 123 115 115 100 115 115 115 102 115]
[ 65 106 100  97 104 106 118  97  98 107]
[ 65  97  97 115  97  97 104 104  97 125]
```

In [ ]:

2.2. HSC

这道题目是希尔密码的爆破，字符码值取值范围是0-255，明文是可打印字符串（ASCII码在32到126之间），给出了长度为30的flag的前四个字符和最后一个字符：AAA{??????????????????????????}，加密矩阵是一个3*3的矩阵，也给出了加密后的字符串：

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'

2.2.1. 思路

最朴素的想法是，遍历密钥矩阵的每一个可能值，判断是否是可逆矩阵，如果是的话计算得到FT判断是否符合原字符串的结构（AAA{??????????????????????????}），但这种办法的尝试次数是256^9，即2^72次，是不现实的，因此需要转换思路。

首先，我们可以直接找密钥的逆矩阵，而不需要多做一步求逆的操作；其次，根据线性代数的知识，MT.invRT的结果的第一行由MT.inv的第一行和RT的全部来决定，也就是说，**每一行的判断可以相互独立**，每行的尝试次数是2^24次，三行也就是3x2^24次，具有可行性。

结合已有的信息，明文的结构应当如下，其中?是未确定部分，通过对字符的筛选（可打印字符和已知部分），可以完成绝大部分的筛选工作。

```
[65, ?, ?, ?, ?, ?, ?, ?, ?, ?]
[65, ?, ?, ?, ?, ?, ?, ?, ?, ?]
[65, ?, ?, ?, ?, ?, ?, ?, ?, 125]
```

2.2.2. 实现

```
result =
b'\xfc\xf2\x1dE\xf7\xd8\xf7\x1e\xed\xccQ\x8b9:z\xb5\xc7\xca\xea\xcd\xb4b\xdd\xcb\xf
2\x939\x0b\xec\xf2'
RT = matrix(Zmod(256), 3, 10)
for p in range(3):
    for q in range(10):
        RT[p, q] = result[p + q * 3]
for a in range(256):
    print(f'Line:{a}')
    for b in range(256):
        for c in range(256):
            MT_inv = matrix(Zmod(256), [[a, b, c], [0,0,0], [0, 0, 0]])
            FT_decrypted = MT_inv * RT
            flag_decrypted = ''.join(chr(int(x)) for x in FT_decrypted.list())
            if all(32 <= ord(char) <= 126 or ord(char)==0 for char in
flag_decrypted) and flag_decrypted[0] == 'A' and flag_decrypted[1] == '{':
                print(f"找到可能的解密结果: {flag_decrypted}", a, b, c)
print("程序执行完成。")
```

这是第一行的代码示例，第二行和第三行的逻辑也与此类似。在运行过程中，为了加快运算速度，我采取了多线程的做法，将任务分到了8个进程进行，最后得到了以下运行情况：

```
Line:122
Line:123
Line:124
找到可能的解密结果: A{1*41Q^n& 124 177 43
Line:125
Line:126
Line:127
程序执行完成。
```

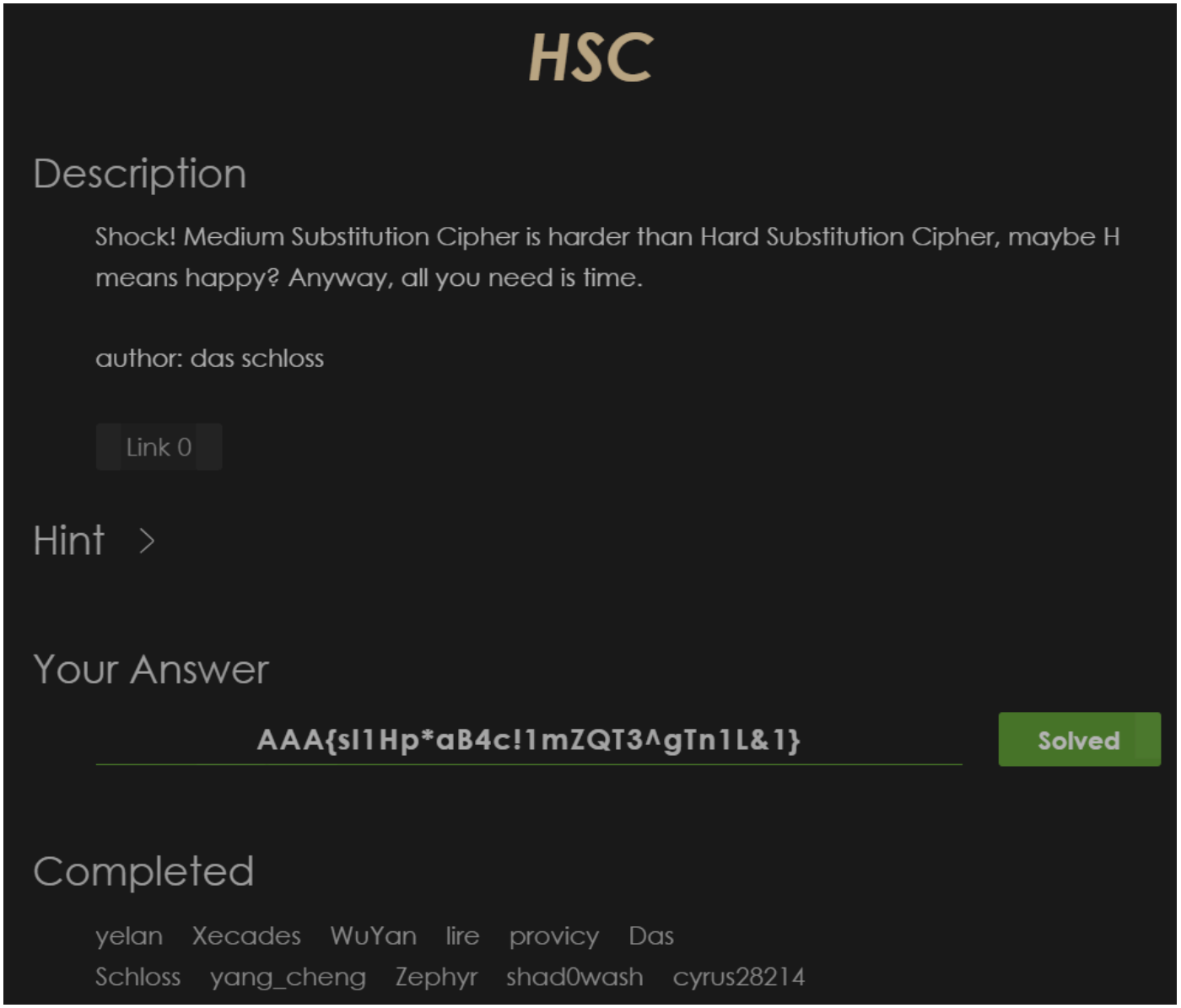
```
Line:156
Line:157
找到可能的解密结果: A{1*41Q^n&AsHacmTg11 157 38 157
Line:158
Line:159
```

```
Line:136
Line:137
找到可能的解密结果: A{1*41Q^n&AsHacmTg11AlpB!Z3TL} 137 33 143
Line:138
Line:139
Line:140
```

根据得到的结果使用Python进行处理，得到：

```
[65, 123, 49, 42, 52, 49, 81, 94, 110, 38]
[65, 115, 72, 97, 99, 109, 84, 103, 49, 49]
[65, 108, 112, 66, 33, 90, 51, 84, 76, 125]
b'AAA{sl1Hp*aB4c!1mZQT3^gTn1L&1}'
```

AAA{sl1Hp*aB4c!1mZQT3^gTn1L&1}即为预期flag，并通过校巴完成验证：



3. Bonus

PoW部分就是一个sha256验证，简单写个代码穷举一下即可，范围是四位的字母加数字。

该程序自定义了一个gcd函数，以递归的方式求最大公因数，然后默认用一个模幂运算加密，并且在后面提供了四种方式来调用这个函数。

这里的a和b都被要求是大于0，对于gcd(a,m)这个函数，a不能为0时很难直接得到m。虽然a是m的一个因数时，gcd(a,m)=a，以及gcd(m-1,m)=gcd(m+1,m)=1，但是m在此处是一个非常大的数，即便a一直取质数去爆破质因数，效率估计也是很低下的。

理论上来说，线性时间复杂度的算法不太适合于这道题，除非系数特别小，例如通过某一组同余式来使用中国剩余定理去缩小范围。也就是说，在gcd(a,m)这个式子中使用一定数量的质数来构造同余式。然而这个式子只能返回gcd的值，而不是余数的值，似乎没办法这么做。

回头看gcd函数，这个函数的写法不怎么高明，写成递归的效率没有循环高，而且a、b的特定取值还会可能因为递归深度过大而导致程序崩溃。m是一个很大的值，a的特定取值似乎有可能会让gcd(a,m)崩溃。为了了解递归崩溃的具体上限，我上网查了一下，这个深度默认是一个定值——1000。

要么是出题人保证做出这道题目的方法不会导致程序崩溃，要么是这道题需要用到这个递归上限，我猜测是后者。

递归的次数和辗转相除的次数有关，也就是和同余式的余数有关，这么说来用同余式来完成这道题是有可能的。通过判断递归次数就能初步猜测余数取值，为了凑够1000次，可以用gcd(a,b,f=lambdax:gcd(x,m)))来凑，这个式子先算x=gcd(a,b)，再算gcd(x,m)，我们可以用前一个式子来凑1000次中的绝大多数部分，用后一个式子来达到递归临界值。

查询OI Wiki可知，使用斐波那契数列相邻两项可以使辗转相除达到最糟糕情况，而且也能很方便地计算递归次数，很适合做第一个式子，用其算出的x来进行第二个式子的运算，从而构造同余式。足够的同余式理论上能大幅减少爆破时间。

综上，借助Python递归上限来构造同余式组来爆破成功的可能性比较大。