Crypto_lab1_Report

Task1 vigenere-encrypt

题目给定了加密的程序，由程序中我们可以得知，加密的秘钥为15-30位，且加密的方式也为乘法，而且字符集基本涵盖所有可见字符。但仍然可以根据字符出现的频率猜测秘钥。注意到{gY字符在文中经常出现，所以先对这三个字符进行查找。查找完后计算出现位置的间隔的最大公因数，发现29满足条件，则秘钥的长度大概率为29。然后对字符进行猜测，刚才的三个字符由于多次出现，所以猜测其为"the",之后再带回算法中进行验证，

```
root@LAPTOP-78LSF82F:/mnt/d/ctf/summer_course/crypto_lab1/vigenere-encrypt# python3 decode.py
[61, 612, 1105, 1279, 1656, 2091]
[91, 71, 57] [84, 72, 69] [90, 67, 43]
```

得到了密钥长度之后我们就可以对密钥进行破解，即寻找某处的最可能的密钥字符。尝试所有可能的密钥值，并检查解密后的字符是否是有效字符（小写字母或空格）。选择有效字符出现次数最多的那个密钥值，即可得到可能的密钥。 得到了秘钥之后，将秘钥带回原秘文中（即需写一个函数实现解密过程），最终就能得到flag

```
By the mid-nineteenth century, the term "icebox" had entered the American language, but ice was still only beginning to
affect the diet of ordinary citizens in the United States. The ice trade grew with the growth of cities. Ice was used in
 hotels, taverns, and hospitals, and by some forward-looking city dealers in fresh meat, fresh fish, and butter.After th
e Civil War (1861-1865), as ice was used to refrigerate freight cars, it also came into household use. Even before 1880,
 half the ice sold in New York, Philadelphia, and Baltimore, and one-third of that sold in Boston and Chicago, went to f
amilies for their own use. This had become possible because a new household convenience, the icebox, a precursor of the
modern refrigerator, had been invented.
Making an efficient icebox was not as easy as we might now suppose. In the early nineteenth century, the knowledge of th
e physics of heat, which was essential to a science of refrigeration, was rudimentary. The commonsense notion that the b
est icebox was one that prevented the ice from melting was of course mistaken, for it was the melting of the ice that pe
rformed the cooling.
Nevertheless, early efforts to economize ice included wrapping the ice in blankets, which kept the ice from doing its jo
b. Not until near the end of the nineteenth century did inventors achieve the delicate balance of insulation and circula
tion 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 twe
nty miles outside the city of Washington, for which the village of Georgetown was the market center. When he used an ice
box 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.
One advantage of his icebox, Moore explained, was that farmers would no longer have to travel to market at night in orde
r to keep their produce cool.
Perhaps the most obvious way artistic creation reflects how people live is by mirroring the environment - the materials
and technologies available to a culture. Stone, wood, tree bark, clay, and sand are generally available materials. In ad
```

AAA{i_like_T0ef1_v3ry_M3uh!!!}

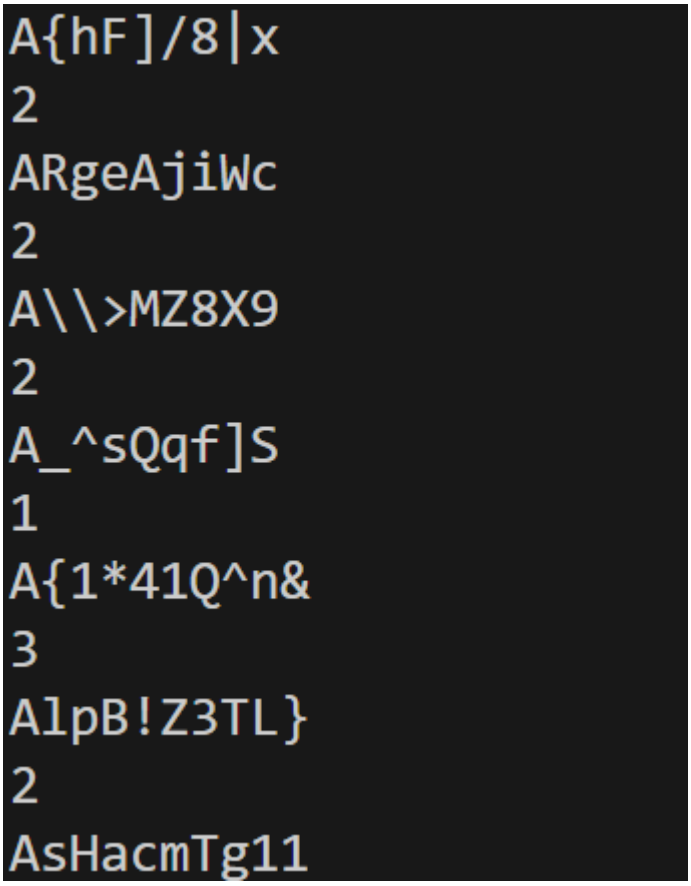
Task2 HSC.sage

前两小问见代码，运行结果如图

```
root@LAPTOP-78LSF82F:/mnt/d/ctf/summer_course/crypto_lab1/crypto_code# sage random.sage
[215  11  87]
[141  55   0]
[109 150 204]
AAA{85TPW#OwxwpGP#0*S1`\fp7e^}
[ 65 123  84  35 120  71  48  49 102 101]
[ 65  56  80  79 119  80  42  96 112  94]
[ 65  53  87 119 112  35  83  92  55 125]
RT (result of MT * FT):
 [121 184 141  59 245 246  83 139  43  88]
[196 199 116  64 169  75 118 157  62 211]
[143 107 248   5  18 255  48 109 226 177]
[ 65 123  84  35 120  71  48  49 102 101]
[ 65  56  80  79 119  80  42  96 112  94]
[ 65  53  87 119 112  35  83  92  55 125]
```

本题的代码中有最后呈现的秘文为两个矩阵MT和FT相乘的形式。而我们由flag的格式可知，第零列的三个数全为A，而且还知道两个{}的位置。当然，我们在这里需要将每一行分开枚举，得到的结果应该均在32-127之间（即可见字符）这样每次仅需要枚举三个数，时间复杂度为256^3，在我们的可接受范围之内。得到结果后与原矩阵的每一行进行比对即可获取有意义的flag（虽然也不知道有什么意义） 当然需要爆破的题目还是用c写好

一些（这个程序一秒钟即可得到结果）但sage代码也一并附上了。



AAA{sl1Hp*aB4c!1mZQT3^gTn1L&1}

Bonus

对于这道题，总共有四种加密的方法 Type 1 (t == 1): gcd(a, b) 这类请求计算的是加密后的 a 和 b 的最大公约数的加密值。我们可以通过多次尝试不同的 a 和 b，来验证加密函数的行为和最大公约数的加密结果。 Type 2 (t == 2): gcd(a, m) 这类请求直接将 a 与 m 进行 gcd 计算，并返回结果的加密值。m 是我们要破解的目标，如果我们选择的 a 恰好与 m 有公约数，gcd(a, m) 的结果会反映在返回值中。 如果 a 与 m 互质，返回的结果应该是 enc(1)。 Type 3 (t == 3): gcd(a, b, f=lambda x: gcd(x, m)) 这类请求在内部进行两次 gcd 计算。首先计算 gcd(a, b)，然后将结果与 m 进行 gcd 计算，并返回最终结果的加密值。 这种嵌套计算可以提供更多关于 m 的信息，特别是如果我们选择的 a 和 b 使得第一次 gcd 的结果与 m 有特定关系。 Type 4 (t == 4): gcd(a, m, f=lambda x: gcd(x, b)) 类似于 Type 3 的嵌套计算，但第二次 gcd 计算是与 b 进行。通过不同的 a 和 b，可以进一步探索 m 的特性。

破解思路

我们可以选择一些小的 a 值，通过 t == 2 请求计算 gcd(a, m) 的结果。如果 a 与 m 互质，结果应该是 enc(1)。如果 a 是 m 的一个因子，结果会是 enc(a)。 多次尝试即可帮助我们猜测 m 的部分因子。

通过 Type 3 和 Type 4 请求，利用嵌套的 gcd 计算， gcd(a, m, f=lambda x: gcd(x, b))，通过不同的 a 和 b，进一步推断 m 的值。