Author

Derek Xu

Challenge 1

Ceasar cipher. We can simply shift the ciphertext by 1-26

```
cipher = "^s<q□N}□{□;t~bp?vN}□{□;"
plaintext = ""

for i in range(26):
  plaintext = ""
  for char in cipher:
    plaintext += chr(ord(char) - i)
  print(plaintext, i)
```

This gives the following output (some characters cannot be displayed)

```
^s<q□N}□{□;t~bp?vN}□{□; 0
]r;p~M|□z~:s}ao>uM|□z~: 1
\q:o}L{□y}9r|`n=tL{□y}9 2
[p9n|Kz□x|8q{_m<sKz□x|8 3
Zo8m{Jy□w{7pz^l;rJy□w{7 4
Yn7lzIx□vz6oy]k:qIx□vz6 5
Xm6kyHw~uy5nx\j9pHw~uy5 6
Wl5jxGv}tx4mw[i8oGv}tx4 7
Vk4iwFu|sw3lvZh7nFu|sw3 8
Uj3hvEt{rv2kuYg6mEt{rv2 9
Ti2guDszqu1jtXf5lDszqu1 10
Sh1ftCrypt0isWe4kCrypt0 11
Rg0esBqxos/hrVd3jBqxos/ 12
Qf/drApwnr.gqUc2iApwnr. 13
Pe.cq@ovmq-fpTb1h@ovmq- 14
Od-bp?nulp,eoSa0g?nulp, 15
Nc,ao>mtko+dnR`/f>mtko+ 16
Mb+`n=lsjn*cmQ_.e=lsjn* 17
La*_m<krim)blP^-d<krim) 18
K`)^l;jqhl(akO],c;jqhl( 19
J_(]k:ipgk'`jN\+b:ipgk' 20
I^'\j9hofj&_iM[*a9hofj& 21
H]&[i8gnei%^hLZ)`8gnei% 22
G\%Zh7fmdh$]gKY(_7fmdh$ 23
F[$Yg6elcg#\fJX'^6elcg# 24
EZ#Xf5dkbf"[eIW&]5dkbf" 25
```

We look through by hand and find readable text "Sh1ftCrypt0isWe4kCrypt0"

Challenge 2

The text given looks roughly like real text with natural spacing. So this cipher was assumed to be encrypted with some sort of substitution cipher. Using Robert Lewand's frequency of the alphabet (etaoinshrdlcumwfgypbvkjxqz), we have a baseline for our mapping.

We first count the frequencies of letters in the ciphertext

```
frequencies = {}

for letter in cipher:
    if letter not in frequencies:
        frequencies[letter] = 1
    else:
        frequencies[letter] += 1
```

Then map the most to least frequent with the most to least frequent letters given by Lewand.

```
knownFrequencies = ['e', 't', 'a', 'o', 'i', 'n', 's', 'h', 'r', 'd', 'l', 'c',
'u', 'm', 'w', 'f', 'g', 'y', 'p', 'b', 'v', 'k', 'j', 'x', 'q', 'z']

# remove spaces in the frequencies
del frequencies[" "]

mapping = {}
for i in range(26):
    max_letter = max(frequencies, key=frequencies.get)
    mapping[max_letter] = knownFrequencies[i]
    del frequencies[max_letter]
```

```
// initial mapping
{
    "Q": "e",
    "J": "t",
    "C": "a",
    "W": "o",
    "I": "i",
    "Y": "n",
    "K": "s",
    "H": "h",
    "M": "r",
    "G": "d",
    "Z": "l",
    "N": "c",
    "T": "u",
    "X": "m",
    "O": "w",
    ",": "f",
    "P": "g",
    "R": "y",
    "B": "p",
    "S": "b",
    ".": "v",
    "D": "k",
    "U": "j",
    "\"": "x",
    "E": "q",
    "'": "z"
}
```

With this baseline, the mapping was tweaked slowly to reveal more english words until eventually, the following mapping made the password readable.

```
mapping = {
    'Q': 'e',
    'J': 't',
    'C': 'a',
    'W': 'o',
    'I': 's',
    'Y': 'r',
    'K': 'n',
    'H': 'h',
    'M': 'i',
    'G': 'd',
    'Z': 'l',
    'N': 'u',
    'T': 'u',
    'X': 'm',
    'O': 'p',
    ',': ',',
    'P': 'g',
    'R': 'y',
    'B': 'c',
    'S': 'm',
    '.': '.',
    'D': 'k',
    'U': 'b',
    '"': 'x',
    'E': 'v',
    "'": 'z'
}


for letter in cipher:
    if letter in mapping:
        print(mapping[letter], end="")
    else:
        print(letter, end="")
```

```
xthat strongest passmord ever devised is thispassmordisverusecure?xxoh - ues,x
said harru, mho mas obviouslu supposed to agree.
```

We can infer the answer then to be "THISPASSWORDISVERYSECURE"

Challenge 3

We are told that the file is an encrypted zip file. Zip files will tend to have the same local file header as shown at this link: https://en.wikipedia.org/wiki/ZIP_(file_format)#File_headers with a value of 0x04034b50. Since this is an encrypted file, we can make a guess that the file was encoded with either AES or XOR encryption algorithms. Since XOR is easier to solve for, we try that first.

XOR encryption has the flaw that any XOR operation of either the plaintext, ciphertext, or key will result in revealing the plain version of one of those components. Even just XORing a small segment of the ciphertext with the plaintext can reveal the key used to encrypt. SO we can exploit the fact that XOR is not random and patterns can arise so long as we have a guess of the plaintext or key.

A dummy zip file is first created in order to get data of zip file headers. These first bytes of the zip file and cipher are XORed together to reveal a possible key.

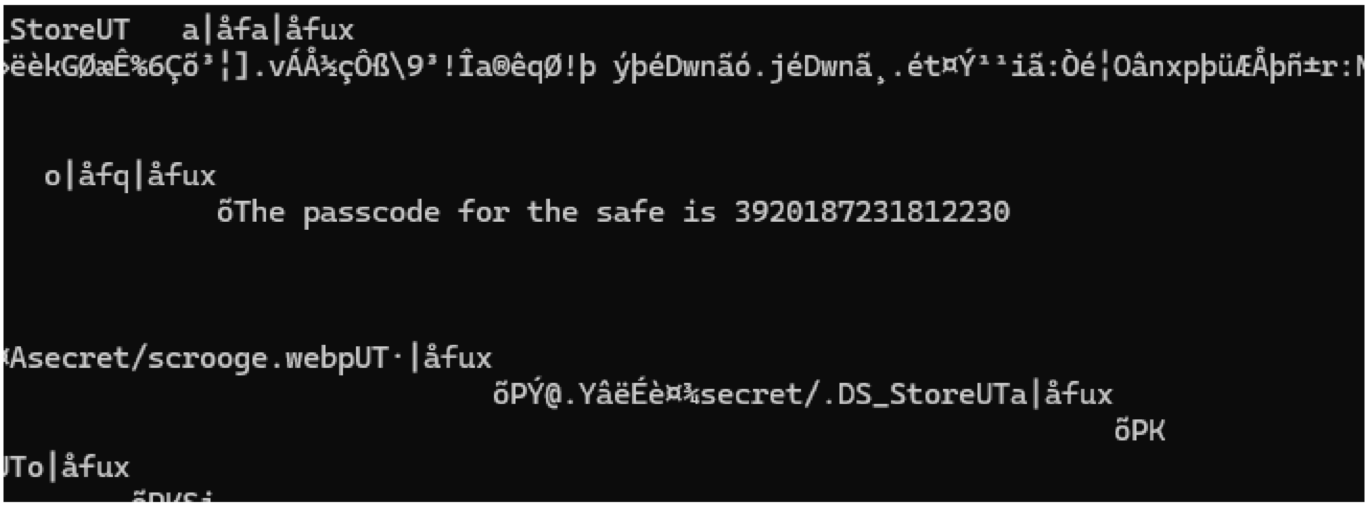
```
xor = ""

with open("challenge3", "rb") as cFile, open("test.txt.zip", "rb") as zip_file:
    # Read bytes from both files
    cText = cFile.read()
    zText = zip_file.read()

    # XOR corresponding bytes

    xor = bytes([a ^ b for a, b in zip(cText, zText)])
```

```
// segment of output
b"LPCER0DPCEC\x00'SCER0LPC"
b"LPCER0DPCEC\x00'SCER0LPCE"
b"LPCER0DPCEC\x00'SCER0LPCER"
b"LPCER0DPCEC\x00'SCER0LPCER0"
b"LPCER0DPCEC\x00'SCER0LPCER0L"
b"LPCER0DPCEC\x00'SCER0LPCER0LP"
b"LPCER0DPCEC\x00'SCER0LPCER0LPL"
b"LPCER0DPCEC\x00'SCER0LPCER0LPLE"
b"LPCER0DPCEC\x00'SCER0LPCER0LPLEn"
```

```
b"LPCER0DPCEC\x00'SCER0LPCER0LPLEn0"
b"LPCER0DPCEC\x00'SCER0LPCER0LPLEn0K"
b"LPCER0DPCEC\x00'SCER0LPCER0LPLEn0KP"
b"LPCER0DPCEC\x00'SCER0LPCER0LPLEn0KPS"
b"LPCER0DPCEC\x00'SCER0LPCER0LPLEn0KPSC"
```

Knowing the first 4 bytes are the same between the dummy zip and the cipher, we can assume that the start of the key is LPCE. Then this key is XORed with the ciphertext to reveal mostly nothing. Parts of the key are then added on and trial and error gives a key of LPCER0 showing in plaintext the passcode.

```
plaintext = ""
key = "LPCER0"
for pos in range(len(ciphertext)):
    plaintext += chr(ciphertext[pos] ^ ord(key[pos % len(key)]))
```



```
StoreUT   a|åfa|åfux
ëèkGØæÊ%6Çõ³¦].vÁÅ¾çÔß\9³!Îa®êqØ!þ ýþéDwnãó.jéDwnã¸.ét¤Ý¹¹iã:Òé¦Oânxpþü ÆÅþñ±r:

  o|åfq|åfux
          õThe passcode for the safe is 3920187231812230



Asecret/scrooge.webpUT·|åfux
                           õPÝ@.YâëÉè¤%secret/.DS_StoreUTa|åfux
                                                              õPK
To|åfux
```

Challenge 4

Using the context clue that the photo was taken with Olympus Deltis VC-1100, we look online to see what type of image format the camera supports. Looking on https://www.digitalkameramuseum.de/en/cameras/item/olympus-deltis-vc-1100, we see that the camera stores images as j6i files and also supports conversions to TIFF and BMP. The image resolution is also 768x576

Using this, we assume that the image was encrypted using AES encryption which encrypts in blocks and thus shows patterns despite being encrypted.

We first generate a BMP file with the same resolution of the camera. I did this by using photoshop to create a PNG file, and then using an online converter to a BMP file.

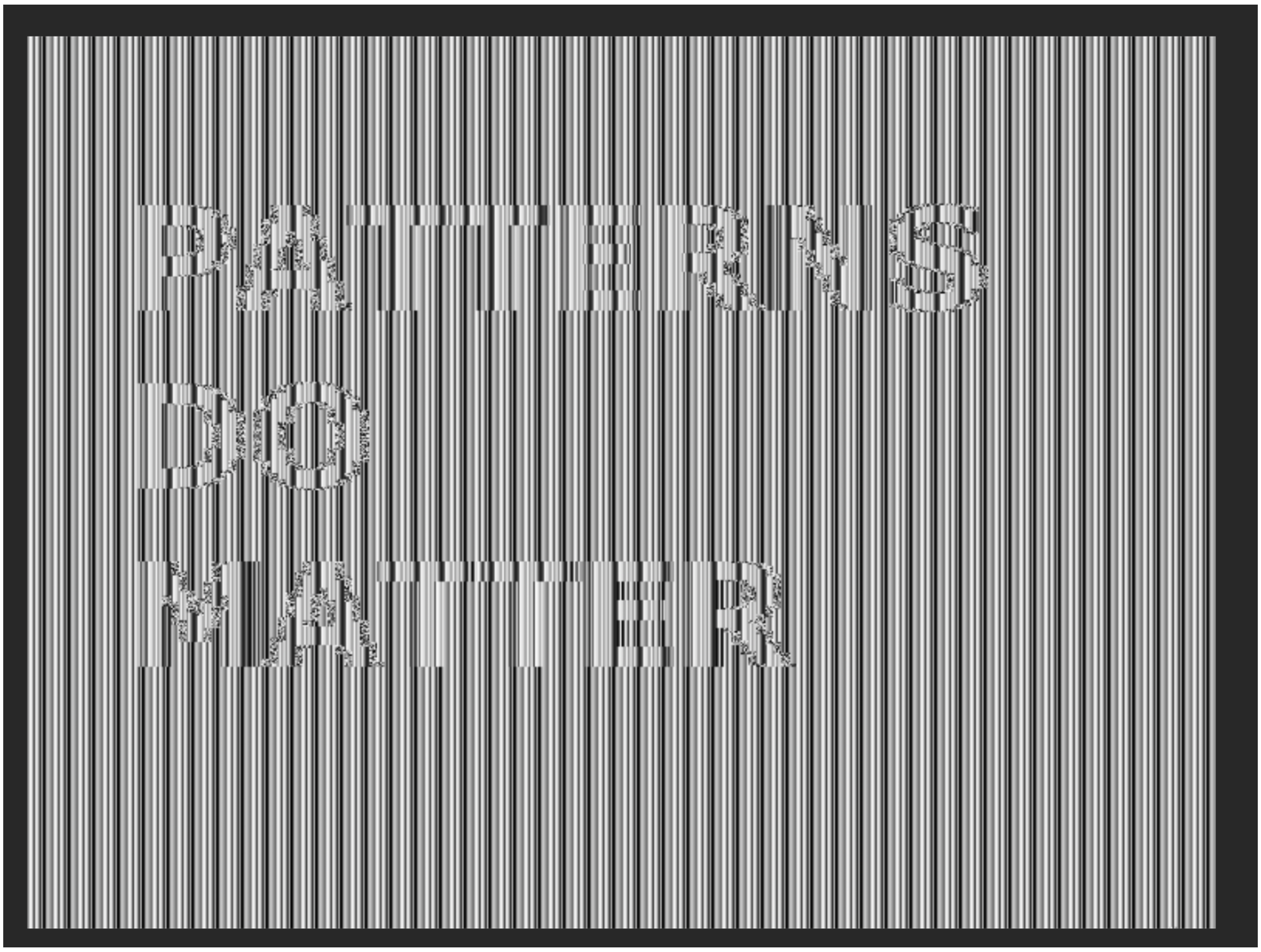
Take the BMP file and extract its header, copy the bytes after the header length from the encrypted image and then append the plaintext header to the encrypted data.

- The head command needed some trial and error to get the correct number of lines to get the header but not too much of the sample image.

Viewing the new .bmp file, we can find the cipher text.

```
// zsh commands run
head -n 1 sample.bmp > header
stat -f%z header # gives 104
dd bs=1 skip=104 if=challenge4 of=content
cat header content > challenge4_image.bmp
```

The resulting image shows vertically flipped text of "PATTERNS DO MATTER"


PATTERNS
DO
MATTER