writeup

November 14, 2024

Field	Information
Author	Abdelaziz Nematallah
Date	2/11/2024
Desc	This notebook is to solve the second task of lab1

0.0.1 Problem Description: Task2. Find the Right Algorithm

- You are given the ciphertext cipher.crypt, and the two opposing sentences contained in plaintext1.txt and plaintext2.txt. Which of these two messages could actually be hidden here? It turns out that it might be both! Our task is to find an encryption scheme (Gen,Enc,Dec) and two keys k1 and k2 which fulfill the requirements:
  - Dec k1 (c) = p1 and
  - Dec k2 (c) = p2,
- i.e., the decryption of cipher.crypt results in the plaintext of plaintext1.txt if k1 is being used and in plaintext2.txt if k2 is being used. Submit both keys k1 and k2 in the form of k1.key and k2.key files containing the binary data. Also submit a small writeup (Writeup.md) including the algorithm you used and how you obtained the desired results. If you develop a script to aid your purposes, please submit it as well as cipher_keys.{py | cc | …}. For this task, it does not matter if you develop your own script or use a publicly available tool. However, in both cases make sure to guarantee access to the algorithm used, or else the task can not be assessed and no points can be awarded.

0.1 Solution Steps:

- Since we have Cipher text, and plaintext1, and plaintext2.
- if we used XOR algorithm, we can have such equations
  - C = p1 xor key1
  - C = p2 xor key2
- So now to get key1, all we need to do is to xor p1 with C, this should result in key1
- and applying the same using p2, should result in key2
- so lets try.

0.1.1 Algorithm Steps:

1. read all files (p1, p2, c) as binary
2. key1 = p1 xor c
3. key2 = p2 xor c
4. store key1 as key1.key

5. store key2 as key2.key

0.1.2 1. Read all files

[5]:
```
def readFiles():
    # this should return 3 items, p1,p2,c in binary format
    filePathes = ['./cipher.crypt', './plaintext1.txt', './plaintext2.txt']

    # we will store results here
    binaryContent = [] # c , p1 , p2

    for filePath in filePathes:
        with open(filePath, 'rb') as file :
            content = file.read()
            binaryContent.append(content)

    # now lets print it to be sure
    for content in binaryContent:
        # just to print it in binary format
        binaryString = ''.join(format(byte, '08b') for byte in content)
        print(binaryString)

    return binaryContent
```

2. get keys

[6]:
```
def evaluateKey(p:str, c:str):

    # Ensure both files have the same length for XOR operation
    if len(p) != len(c):
        raise ValueError("The two files must have the same length for XOR␣
↪operation.")

    # Perform XOR operation byte by byte
    key = bytes(a ^ b for a, b in zip(p, c))

    print(key)
    return key
```

0.1.3 main method

[10]:
```
def Task2():
    # read files
    files = readFiles()

    # separate the data
    c,p1,p2 = files[0],files[1],files[2]
```

```
    # evaluate key 1
    key1 = evaluateKey(p1, c)

    # evaluate key 2
    key2 = evaluateKey(p2, c)

    return key1, key2, c


Task2()
```

```
10001000111100000101111110100001011110010100000000101110010000100110110010010100
10110011111010000110100010010000110011110110000111101100111100110100000001111011
01010010101001001010111100100111100011001111111000110000101101011001011010010101
10001111101111001101111101000100000100110000010100100001011011111010010011100101
01010100011010000110010100100000011101110110111011100100110110001100100001000000
01101111011001100010000001100011011100100111100101110000011101000110111101100111
01110010011000010111000001101000011110010010000001101001011100110010000001100010
01100101011000010111010101110100011010010110011001110101011010000100001000010100
01000001011000110111010001110101011000010110110001101100011110010010000001100101
01101110011000110111001001111001011100000111010001101001011011100110011100100000
01110100011010000110100101101110011001110111001100100000011010010111001100100000
01101111011101100110010101110010011100100110000101110100011001010110010000001010
b'\xdc\x98:\x81\x0e/\\.\x08\xb4\xdc\x8eH\xf3\xbd\x18\x9c\x87/\x1c
\xc5\xdfO\xf5\xdeY\xc6\xb6\xf7\xea\xdd\xaa0zcT\x03\x85\xef'
b"\xc9\x93+\xd4\x18,B;L\xf1\xdd\x8b\x1a\xe9\xbf\x15\x85\x9d'[&\xcc\xc6I\xeb\x8d\
x10\xdc\xe5\xb5\xe0\xca\xba6adU\n\xc0\xef"
```

[10]:
```
(b'\xdc\x98:\x81\x0e/\\.\x08\xb4\xdc\x8eH\xf3\xbd\x18\x9c\x87/\x1c
\xc5\xdfO\xf5\xdeY\xc6\xb6\xf7\xea\xdd\xaa0zcT\x03\x85\xef',
 b"\xc9\x93+\xd4\x18,B;L\xf1\xdd\x8b\x1a\xe9\xbf\x15\x85\x9d'[&\xcc\xc6I\xeb\x8d
\x10\xdc\xe5\xb5\xe0\xca\xba6adU\n\xc0\xef",
 b"\x88\xf0_\xa1y@.Bl\x94\xb3\xe8h\x90\xcfa\xec\xf3@{R\xa4\xaf'\x8c\xfe0\xb5\x96
\x95\x8f\xbc\xdfD\x13\x05!o\xa4\xe5")
```

0.1.4 asses the result

- in order to make sure that we have reached the correct result, we need to encrypt the plain text with the keys, if we got the same result, then we are correct
- another way is to decrypt the encrypted cipher with keys, if we got the same plaintext, then we are correct

[19]:
```
# lets try the decryption method easier.
# we can just use evaluate key, but instead of passing p1, we should pass the␣
 ↪key

keys = Task2()
```

```
k1, k2, c = keys[0],keys[1],keys[2]

print('\nplain text1')
p1 = evaluateKey(k1, c) #
print('plain text2')
p2 = evaluateKey(k2, c)

# lets store them and finish this task ;)
for index, key in enumerate(keys):
    if key == c: # skip cipher
        break
    with open(f'key{index + 1}.key', 'wb') as file:
        file.write(key)
```

```
10001000111100000101111110100001011110010100000000101110010000100110110010010100
10110011111010000110100010010000110011111011000011110110011111001101000000001111011
01010010101001001010111100100111100011001111111000110000101101011001011010010101
10001111101111001101111101000100000100110000010100100001011011111010010011100101
01010100011010000110010100100000011101110110111101110010011011000110010000100000
01101111011001100010000001100011011100100111100101110000011101000110111101100111
01110010011000010111000001101000011110010010000001101001011100110010000001100010
01100101011000010111010101110100011010010110011001110101011011000010000100001010
01000001011000110111010001110101011000010110110001101100011110010010000001100101
01101110011000110111001001111001011100000111010001101001011011100110011100100000
01110100011010000110100101101110011001110111001100100000011010010111001100100000
01101111011101100110010101110010011100100110000101110100011001010110010000001010
b'\xdc\x98:\x81\x0e/\\.\x08\xb4\xdc\x8eH\xf3\xbd\x18\x9c\x87/\x1c
\xc5\xdf0\xf5\xdeY\xc6\xb6\xf7\xea\xdd\xaa0zcT\x03\x85\xef'
b"\xc9\x93+\xd4\x18,B;L\xf1\xdd\x8b\x1a\xe9\xbf\x15\x85\x9d'[&\xcc\xc6I\xeb\x8d\
x10\xdc\xe5\xb5\xe0\xca\xba6adU\n\xc0\xef"

plain text1
b'The world of cryptography is beautiful!\n'
plain text2
b'Actually encrypting things is overrated\n'
```

0.2 Why XOR?

- I thought about it, because having 2 plaintexts resulting in the same cipher with different keys will be a game of bit manipulation, thats can be done using XOR operations, also, most of the used ciphers, and explained ciphers are too complext to be able to generate exactly the same result (cipher text) from two different keys, so it was my first blick thought, and when I applied my analysis, I found the solution, and it would work!

[14]:
```
### DONE :)
```