Coursework 1 - Exersice 1

October 8, 2023

Alice wants to send a couple of secret messages to Bob. To achieve this, they both agreed on OTP key which they will use for encryption and decryption. While one of the messages were being sent you managed to obtain both the plaintext message m_1 and the corresponding ciphertext c_1.

a) Can you compute the OTP key from m_1 and c_1, when:

$$m_1 = \text{LIFEISLIKEABOXOFCHOCOLATES}$$

$$c_1 = \text{CXGDXNIPWXYXTONWQTCVCFXKCY}$$

If it is possible, describe the process of how to achieve the key.

b) Alice and Bob continue to use the same OTP key for multiple messages. Please recover the new message m_2 using all previously known information.

$$c_2 = \text{PDVMTQBYWGMSBYZKMAIPWFIXCZ}$$

We first define the functions:

[1]:
```python
plaintext = "LIFEISLIKEABOXOFCHOCOLATES"
ciphertext = "CXGDXNIPWXYXTONWQTCVCFXKCY"
ciphertext2 = "PDVMTQBYWGMSBYZKMAIPWFIXCZ"

def str2bin(s):
    return ''.join(format(ord(i), '08b') for i in s)

def bin2str(b):
    return ''.join(chr(int(b[i:i+8], 2)) for i in range(0, len(b), 8))

def xor(a, b):
    return ''.join(str(int(a) ^ int(b)) for a,b in zip(a, b))
```

And now proceed to convert the strings into binary elements. We do this by assuming we are working with ASCII characters, this is of great importance as working with different encodings, we would end up with completly different results

[2]:
```python
bin_plaintext = str2bin(plaintext)
bin_ciphertext = str2bin(ciphertext)
print(f"Binary length: {len(bin_plaintext)}")
```

```
Binary length: 208
```

The *XOR* operation is a bidirectional operation, meaning that:

$$c = k \oplus t \leftrightarrow k = c \oplus$$

With this in mind we just need to compute the *XOR* operation between the two binary values computed in the previous step

[3]:
```python
# XOR the plaintext and ciphertext to get the key
bin_key = xor(bin_ciphertext, bin_plaintext)
```

Now we can go ahead and get the key back as a string:

[4]:
```python
# Convert the key to ASCII
key = bin2str(bin_key)
print(f"Bin key: {bin_key}\nHex Key: {hex(int(bin_key, 2))}")
```

```
Bin key: 00001111000100010000000100000001000100010001110100000101000110010001110
00001110100011000000110100001101100010111000000010001000100010010000111000000110
000010101000011000000101000011001000111110000011000001010
Hex Key: 0xf110101111d05191c1d181a1b170111121c0c150c0a191f060a
```

We check that the reverse operations are still valid:

[5]:
```python
if(xor(bin_ciphertext, bin_key) == bin_plaintext and xor(bin_plaintext,␣
 ↪bin_key) == bin_ciphertext):
    print("Key is correct")
```

```
Key is correct
```

For this next section it is important to understand the concept of the OTP: One Time Pad.

What this means is that the key used for one encryption can only be used once, so using it to decrypt a second message would not be valid

If we still wanted to deciphre the message we would have to do the same operations as before: 1. Convert the ciphertext/key to binary 2. Compute the XOR operation 3. Decode the binary result to ascii(assuming ascii is being used)

The result would be the following(insist that this does not make sense in a practical way, since the key used is not valid and we would need the correct one)

[6]:
```python
# Decrypt the second ciphertext
bin_ciphertext2 = str2bin(ciphertext2)
bin_plaintext2 = xor(bin_ciphertext2, bin_key)
plaintext2 = bin2str(bin_plaintext2)
print(plaintext2)
```

```
_UWLELG@KZUIYN[Z_]EE[LPGEP
```

[8]:
```python
hodei = "01000011 01101111 01101101 01100101 01101101 01100101 00100000␣
↪01101100 01101111 01110011 00100000 01101000 01110101 01100101 01110110␣
↪01101111 01110011"
hodei = hodei.split(" ")
hodei = [int(i, 2) for i in hodei]
hodei = [chr(i) for i in hodei]
hodei = "".join(hodei)
print(hodei)
```

```
Comeme los huevos
```

[9]:
```python
x = [1,2,3,4,5]
print([i+1 for i in x])
```

```
[2, 3, 4, 5, 6]
```