Lab 1

David Herel

This is my notebook for lab1 - communication security class.⚾

Exercise 0: make utilities

> This exercise is not strictly mandatory, but it will be useful for the rest of the lab.

> Write 6 functions `bin2txt`, `bin2hex`, `txt2bin`, `hex2bin`, `hex2txt`, `txt2hex`, that convert between the following representations:
>
> - hex: `"426f6f6d"` (more precisely, a string to be interepreted as hexadecimal)
> - text: `"Boom"`
> - binary: `b"Boom"` in Python

> Depending on the language, you may not have to distinguish between binary and text, for instance in C it is the same thing, however in Python one has type `str` whereas the other has type `bytes`.

> You are not expected to write any complex algorithm here, just delegate to the correct utility functions of your language if they are provided. In other words don't rewrite yet another routine to hand-parse hexadecimal.

Here is my implementations of these functions in python.


```
def bin2txt(my_input):
    ascii_string = "".join([chr(int(binary, 2)) for binary in my_input.split(" ")])
    return ascii_string

def bin2hex(my_input):
    ascii_string = "".join([hex(int(binary, 2)) for binary in my_input.split(" ")])
    return ascii_string

def txt2bin(my_input):
    return ' '.join(format(ord(x), 'b') for x in my_input)

def hex2bin(my_input):
    return "{0:08b}".format(int(my_input, 16))

def hex2bytes(my_input):
    return bytes.fromhex(my_input)

def hex2txt(my_input):
    return bytearray.fromhex(my_input).decode()

def txt2hex(my_input):
    return my_input.encode('utf-8').hex()

#XOR function
def encrypt_decrypt_xor(my_input, key):
    input_length = len(my_input)
    key_length = len(key)
    encoded = []

    for i in range(0, input_length):
        xor = ord(my_input[i]) ^ ord(key[i % key_length])
        encoded.append(xor)

    return bytes(encoded)

#XOR function
def encrypt_decrypt_xor_start(my_input, key, start, offset):
    input_length = len(my_input)
    key_length = len(key)
    encoded = []

    for i in range(1, input_length+1):
        if i%offset == start:
            xor = ord(my_input[i-1]) ^ ord(key[(i-1) % key_length])
        else:
            xor = ord(my_input[(i-1)])
```

```
        encoded.append(xor)

    return bytes(encoded)
```

Exercise 1: encrypt xor

> Write a function that encrypts a text with a xor key. The idea is simple: to obtain the first byte of the ciphertext, xor the first byte of the text and the first byte of the key. Continue in this fashion until the text is exhausted. If the key is shorter than the text, it should be recycled (start over from the beginning).

> For instance, xorring the text `everything remains raw` with the key `word up` should give you the following hexadecimal ciphertext: `121917165901181e01154452101d16061c1700071100`.

> What is the ciphertext of the hex-encoded text `the world is yours` against the key `illmatic`?

```
task1 = encrypt_decrypt_xor("the world is yours", "illmatic")
print(task1.hex())
```

```
1d04094d161b1b0f0d4c051e410d06161b1f
```

Ciphertext of the hex-encoded text is: `1d04094d161b1b0f0d4c051e410d06161b1f`.

We can confirm that answer is correct if we reverse the input like this:

```
#confirmation
task1_confirmation = encrypt_decrypt_xor(hex2txt("1d04094d161b1b0f0d4c051e410d06161b1f"), "illmatic")
print(task1_confirmation)
```

```
b'the world is yours'
```

We got right output: `the world is yours`. So we know our answer is correct.

Exercise 2: decrypt single-letter xor

> The following hex-encoded ciphertext was encoded against the single-letter key `$`, i.e. ASCII 36.
>
> `404b48484504404b48484504464d4848045d4b`
>
> Before decrypting, shortly explain what pattern(s) are present in this ciphertext due to the weak mode of encryption.

> Then, decrypt the text. What is the plaintext?

```
task2 = encrypt_decrypt_xor(hex2txt("404b48484504404b48484504464d4848045d4b"), "$")
print(task2)
```

```
b'dolla dolla bill yo'
```

The answer is `dolla dolla bill yo`.

Exercise 3: hand crack single-letter xor

> The file `text1.hex` contains a hex-encoded ciphertext that was xor encoded with a single letter.

> Decrypt it. What is the first line?

```
hexdata_1 = "0f383e392c6d1f253420283e6d383d6d24236d3925286d3d212c2e28616d393f38286d2423292828294714283e6d046d2e2c392e256d3a3f282e266d2c2

for letter in (string.ascii_uppercase + string.ascii_lowercase):
    print(letter)
    task3 = encrypt_decrypt_xor(hex2txt(hexdata_1), letter)
    print(task3)
    #letter M
    #Busta Rhymes up in the place, true indeed
```

```
b'cTRU@\x01sIXLDR\x01TQ\x01HO\x01UID\x01QM@BD\r\x01USTD\x01HOEDDE+xDR\x01h\x01B@UBI\x01VSDBJ\x01@OE\x01UI@U\x06R\x01VNSE\x01NO\x01LX\x01RDDE+h\x06L\x01FT@S@OU
m
b'bUSTA\x00rHYMES\x00UP\x00IN\x00THE\x00PLACE\x0c\x00TRUE\x00INDEED*yES\x00i\x00CATCH\x00WRECK\x00AND\x00THAT\x07S\x00WORD\x00ON\x00MY\x00SEED*i\x07M\x00GUARA
```

```
n
b'aVPWB\x03qKZNFP\x03VS\x03JM\x03WKF\x03SOB@F\x0f\x03WQVF\x03JMGFFG)zFP\x03j\x03@BW@K\x03TQF@H\x03BMG\x03WKBW\x04P\x03TLQG\x03LM\x03NZ\x03PFFG)j\x04N\x03DVBQB

o
b'`WQVC\x02pJ[OGQ\x02WR\x02KL\x02VJG\x02RNCAG\x0e\x02VPWG\x02KLFGGF({GQ\x02k\x02ACVAJ\x02UPGAI\x02CLF\x02VJCV\x05Q\x02UMPF\x02ML\x02O[\x02QGGF(k\x05O\x02EWCPC
p
b'\x7fHNI\\\x1doUDPXN\x1dHM\x1dTS\x1dIUX\x1dMQ\\^X\x11\x1dIOHX\x1dTSYXXY7dXN\x1dt\x1d^\\I^U\x1dJOX^V\x1d\\SY\x1dIU\\I\x1aN\x1dJROY\x1dRS\x1dPD\x1dNXXY7t\x1aP\
q
b'~IOH]\x1cnTEQYO\x1cIL\x1cUR\x1cHTY\x1cLP]_Y\x10\x1cHNIY\x1cURXYYX6eYO\x1cu\x1c_]H_T\x1cKNY_W\x1c]RX\x1cHT]H\x1bO\x1cKSNX\x1cSR\x1cQE\x1cOYYX6u\x1bQ\x1c[I]N]
r
b'}JLK^\x1fmWFRZL\x1fJO\x1fVQ\x1fKWZ\x1fOS^\\Z\x13\x1fKMJZ\x1fVQ[ZZ[5fZL\x1fv\x1f\\^K\\W\x1fHMZ\\T\x1f^Q[\x1fKW^K\x18L\x1fHPM[\x1fPQ\x1fRF\x1fLZZ[5v\x18R\x1fX
s
b'|KMJ_\x1elVGS[M\x1eKN\x1eWP\x1eJV[\x1eNR_][\x12\x1eJLK[\x1eWPZ[[Z4g[M\x1ew\x1e]_J]V\x1eIL[]U\x1e_PZ\x1eJV_J\x19M\x1eIQLZ\x1eQP\x1eSG\x1eM[[Z4w\x19S\x1eYK_L_
t
b'{LJMX\x19kQ@T\\J\x19LI\x19PW\x19MQ\\\x19IUXZ\\\x15\x19MKL\\\x19PW]\\\\]3`\\J\x19p\x19ZXMZQ\x19NK\\ZR\x19XW]\x19MQXM\x1eJ\x19NVK]\x19VW\x19T@\x19J\\\\]3p\x1e
u
b'zMKLY\x18jPAU]K\x18MH\x18QV\x18LP]\x18HTY[]\x14\x18LJM]\x18QV\\]]\\2a]K\x18q\x18[YL[P\x18OJ][S\x18YV\\\x18LPYL\x1fK\x18OWJ\\\x18WV\x18UA\x18K]]\\2q\x1fU\x18
v
b'yNHOZ\x1biSBV^H\x1bNK\x1bRU\x1bOS^\x1bKWZX^\x17\x1bOIN^\x1bRU_^^_1b^H\x1br\x1bXZOXS\x1bLI^XP\x1bZU_\x1bOSZO\x1cH\x1bLTI_\x1bTU\x1bVB\x1bH^^_1r\x1cV\x1b\\NZI
w
b'xOIN[\x1ahRCW_I\x1aOJ\x1aST\x1aNR_\x1aJV[Y_\x16\x1aNHO_\x1aST^__^0c_I\x1as\x1aY[NYR\x1aMH_YQ\x1a[T^\x1aNR[N\x1dI\x1aMUH^\x1aUT\x1aWC\x1aI__^0s\x1dW\x1a]O[H[
x
b'w@FAT\x15g]LXPF\x15@E\x15\\[\x15A]P\x15EYTVP\x19\x15AG@P\x15\\[QPPQ?lPF\x15|\x15VTAV]\x15BGPV^\x15T[Q\x15A]TA\x12F\x15BZGQ\x15Z[\x15XL\x15FPPQ?|\x12X\x15R@T
y
b'vAG@U\x14f\\MYQG\x14AD\x14]Z\x14@\\Q\x14DXUWQ\x18\x14@FAQ\x14]ZPQQP>mQG\x14}\x14WU@W\\\x14CFQW_\x14UZP\x14@\\U@\x13G\x14C[FP\x14[Z\x14YM\x14GQQP>}\x13Y\x14S
z
b'uBDCV\x17e_NZRD\x17BG\x17^Y\x17C_R\x17G[VTR\x1b\x17CEBR\x17^YSRRS=nRD\x17~\x17TVCT_\x17@ERT\\\x17VYS\x17C_VC\x10D\x17@XES\x17XY\x17ZN\x17DRRS=~\x10Z\x17PBVE
```

We can see that letter that text was encoded with a letter `M`.

The first line of the text is: `Busta Rhymes up in the place, true indeed`.

Exercise 4: automate cracking single-letter xor

> Solve the previous exercise, but instead of searching for the correct key/plaintext with your eyes, make the computer do it. In other words, you should have a function that, given a single-letter xor encoded ciphertext, will return you the single-byte key (and, if you want, the plaintext).

> You could devise a scoring function that checks, for a given decryption, if it seems like English. Then just iterate through all possible keys and return the key whose decryption gets the best score.

```
english_letter_freq = {'a': 8.167, 'b': 1.492, 'c': 2.782, 'd': 4.253, 'e': 12.702, 'f': 2.228,
        'g': 2.015, 'h': 6.094, 'i': 6.966, 'j': 0.153, 'k': 0.772, 'l': 4.025,
        'm': 2.406, 'n': 6.749, 'o': 7.507, 'p': 1.929, 'q': 0.095, 'r': 5.987,
        's': 6.327, 't': 9.056, 'u': 2.758, 'v': 0.978, 'w': 2.360, 'x': 0.150,
        'y': 1.974, 'z': 0.074}


#returns score of message (how similar it is to freq analysis of words)
def get_frequency(message):
    my_letter_freq = {'a': 0, 'b': 0, 'c': 0, 'd': 0, 'e': 0, 'f': 0,
        'g': 0, 'h': 0, 'i': 0, 'j': 0, 'k': 0, 'l': 0,
        'm': 0, 'n': 0, 'o': 0, 'p': 0, 'q': 0, 'r': 0,
        's': 0, 't': 0, 'u': 0, 'v': 0, 'w': 0, 'x': 0,
        'y': 0, 'z': 0}

    letters = 'abcdefghijklmnopqrstuvwxyz'
    len_of_text = 1
    for letter in message:
        if letter in letters:
            my_letter_freq[letter] += 1
            len_of_text += 1

    #normalize it
    my_letter_freq = {k: v / len_of_text for k, v in my_letter_freq.items()}

    score = 0.0
    #count differences
    for letter in letters:
        score += abs(english_letter_freq[letter] - my_letter_freq[letter])

    return score

def automatic_xor(message, start, offset):
    best_score = 99999
    best_message = ""
    key = ""

    for letter in (string.printable):
```

```
        #xor message with key(letter)
        if (offset == 1):
            task3 = encrypt_decrypt_xor(message, letter)
        else:
            task3 = encrypt_decrypt_xor_start(message, letter, start, offset)
        #from bytes to text
        task3_text = task3.decode()
        #get score of message
        score = get_frequency(task3_text)

        #if it is lower than best one, update it
        if(score < best_score):
            best_score = score
            best_message = task3_text
            key = letter
    return best_message, key, best_score


#run automatic xor
answer, key, score = automatic_xor(hex2txt(hexdata_1), 1, 1)
print(answer)
```

```
Whenever I travel the world I landcruise
If you choose to around you get bruised
Now I got you gassed on super unleaded fuels
Give me room, give me some space yo excuse

You now rocking with the best
Busta Rhymes coming through from the Flip Mode Squad
Boy Scout's who I be
Straight to your dome, we coming straight to your dome
Bringing all new ruckus to all you rap
Boy Scout's who I be, Flip Mode is the squ-id-ad
Busta Rhymes break it down like this

Yo which stole my flow
Eenie, meenie miney mo
Throw them type of right out my window
Blast your hit you with a direct blow
Bo! Coming through like G.I. Joe
Star Wars moving ill like Han Solo
Make you bounce around like this was calypso
Always shine cause I got the Hi-Pro Glow
You think that you can hide you think you can lay low
Roll up on your like Hawaii 5-0
Macked out with my dreads and my Kangol
Forget the Moet just pass the Cisco
Yo! Take a trip down to Mexico

Come back with that that might make you psycho
Maximum frequencies through your stereo
Sorry this is it but homeboy I got to go
```

Now I have automated the method. Firstly I wanted to seperate line by words and check if word exists. The sentence with most correct words is the best one. But I used nltk corpus and it took so much time. So I decided to go for more simple option: frequency analysis.

Exercise 5: crack multiple-letter xor with given key length

> he file `text2.hex` contains a hex-encoded ciphertext that was xor encoded against a multiple-letter key -- just like Ex 1.

> Crack it. You are given the following indication: the key contain 10 characters.

> Notice that by a simple manipulation of the ciphertext, the 10-letter encryption is nothing more than a collection of 10 single-letter encryptions -- which you can already crack thanks to Ex 4.

> What is the key, what is the first line?

```
hexdata_2 = "10521501114d092b74233c0006170418496e2425361b1f1f00030422786d3507151716191c22316d370050040c1e0429316d23141c176f21006e362c310

from itertools import product
from string import ascii_lowercase
```

```
#multiple xor -does not work
def automatic_multiple_xor(message, length):
    answer = message
    key_final = ""
    for i in range(1, length+1):
        if i == length:
            answer, key, score = automatic_xor(answer, 0, length)
            key_final += key
        else:
            answer, key, score = automatic_xor(answer, i, length)
            key_final += key
    return answer, key_final, score

answer, key, score = automatic_multiple_xor(hex2txt(hexdata_2), 10)
print(answer)
print("With key: " + str(key))
```

```
Et non ici c'est
Saint-Denis, Saint-Denis, Fon-fonky fresh
Saint-Denis, Saint-Denis, Fon-fonky fresh
Dans l'arene, le supreme, la creme, la cerise sur le gateau
Tu connais le deal gros, pas besoin que j'en fasse trop
C'est moi la voix qui fout ta te-ci dans tous ses etats
Tu kiff, tu kiff pas, Nicoumouk viendra a toi
Voila pourquoi j'ai pas le droit
J'lache pas le 9.3., j'file droit
Avec un funk bestial, Seine-Saint-Denis Style!
Seine-Saint-Denis Style, Seine-Saint-Denis Style
Seine-Saint-Denis Style, baby
Seine-Saint-Denis Style!
Fous donc ton gilet par balle
A base de popopopop, mec pour le Hip-Hop je developpe
La Seine-Saint-Denis, c'est de la bombe baby
Et si t'as le pedigree ca se reconnait au debit!
Seine-Saint-Denis Style!
Fous donc ton gilet par balle
A base de popopopop, mec pour le Hip-Hop je developpe
La Seine-Saint-Denis, c'est de la bombe baby
Et si t'as le pedigree ca se reconnait au debit!
Hey ca se reconnait au debit baby
C'est la generation Fonky-Tacchinni
Ah, pas de chichis, pas de tie-pi ici
Si tu derapes on te dessus
A'ight, Seine-Saint-Denis style
C'est de la bombe Baby!

With key: SupremeNTM
```

Done :) The correct key is: `SupremeNTM`

The message is: `C'est le nouveau, phenomenal, freestyle du visage pale...`

Exercise 6: crack multiple-letter xor with unknown key length

> Decrypt `text3.hex` , except this time you don't know the keylength. Even better if you can make your code find out the keylength before trying to decrypt.

> What is the key, what is the first line?

```
hexdata_3 = "022250733d4e347f32263f13214a153d263440733d542e3d313b6d7a6b57153f2a275173274e632d313d38412213413c63385c367d0f6d5517252c403f5

def find_best_len(message):
    lowest = 99999
    opt_len = -1
    for i in range(1, 20):
        answer, key, score = automatic_multiple_xor(message, i)
        if score < lowest:
            opt_len = i
            lowest = score
    return opt_len

opt_len = find_best_len(hex2txt(hexdata_3))
print(opt_len)

answer, key, score = automatic_multiple_xor(hex2txt(hexdata_3), opt_len)
print("Answer is: ")
```

```
print(answer)
print("With the key: " + str(key) + " of length: " + str(opt_len))
```

```
How many of y'all got Criminal Minded?
You, you, you; y'all don't be blinded
Me, I got no jewels on my neck
Why? I don't need 'em, I got your respect
KRS-One, twenty years I rock
I do it for JMJ and Scott La Rock
This hip-hop and we's a nation
Don't you wanna hear more KRS-One on your radio station?
Instead of broadcasting how we smoke them trees
On the radio we need to hear more local emcees
Where you at? C'mon, where you at?
This is the difference between emceeing and rap
Rappers spit rhymes that are mostly illegal
Emcees spit rhymes to uplift their people
Peace, Love, Unity, havin' fun
These are the lyrics of KRS-One

And now for my next number I'd like to return to the...
Classic
Uh, uh, - timeless
Live straight classic
Classic
-ive, straight classic
Timeless
I'd like to return to the classic
Kanye West
I'm Rakim, the fiend of a microphone

With the key: CL4SS!C_TIM3L35S of length: 16
```

I wrote code which determines length of the key. The key with length `16` is `CL4SS!C_TIM3L35S`.
And the answer is `And now for my next number...`.

Bonus: when you have finished all exercises

> A careless user of cryptography has reused a classic timeless key to encrypt the file `secret.zip`, which reveals the way to an important philosophical work.

> According to this masterpiece, what comes brand new?

I did not know what to do, but after random trials and errors I focused on a lyrics from previous task. There are Classic - timeless. Which is also in the hint of this task. So I put there answer from previous task `CL4SS!C_TIM3L35S` and Voila. Got into the file:

The content of file is: `PMbELEUfmIA`.