ZanXie_CA1

January 30, 2024

0.1 Step 1 - Convert hex to base64

The string:

`49276d206b696c6c696e6720796f757220627261696e206c696b65206120706f69736f6e6f7573206d75736872`

Should produce:

`SSdtIGtpbGxpbmcgeW91ciBicmFpbiBsaWtlIGEgcG9pc29ub3VzIG11c2hyb29t`

So go ahead and make that happen. You'll need to use this code for the rest of the exercises.

0.1.1 Comment

Always operate on raw bytes, never on encoded strings. Only use hex and base64 for pretty-printing.

[1]:
```
import codecs
def hex_to_base64(hex_string):

  hex_decode = codecs.decode(hex_string, 'hex') # decode hex encryption to␣
 ↪binary form
  base64_encode = codecs.encode(hex_decode, 'base64') # encode binary form to␣
 ↪base64
  base64_string = base64_encode.decode() # decode binary form to string

  #print(hex_decode)
  #print(base64_encode)
  #print(base64_string)

  return base64_string

# send in hex_string
hex_string =␣
 ↪'49276d206b696c6c696e6720796f757220627261696e206c696b65206120706f69736f6e6f7573206d75736872
base64_string = hex_to_base64(hex_string)
print(base64_string)
```

```
SSdtIGtpbGxpbmcgeW91ciBicmFpbiBsaWtlIGEgcG9pc29ub3VzIG11c2hyb29t
```

0.2 Step 2 - Fixed XOR

Write a function that takes two equal-length buffers and produces their XOR combination.

If your function works properly, then when you feed it the string:

`1c0111001f010100061a024b53535009181c`

… after hex decoding, and when XOR'd (bitwise) against:

`686974207468652062756c6c277320657965`

… should produce:

`746865206b696420646f6e277420706c6179`

[2]:
```
def hex_to_int(s): # convert hex string to int
  return int(s,base=16)

def xor_string(str1, str2): # xor two hex string and return a hex string
  s1 = hex_to_int(str1)
  s2 = hex_to_int(str2)
  return hex(s1^s2)[2:] # slice the 0x at the front

str1 = '1c0111001f010100061a024b53535009181c'
str2 = '686974207468652062756c6c277320657965'

result = xor_string(str1, str2)
print(result)
```

```
746865206b696420646f6e277420706c6179
```

0.3 Step 3 - Single-byte XOR cipher

The hex encoded string:

`1b37373331363f78151b7f2b783431333d78397828372d363c78373e783a393b3736`

… has been XOR'd against a single character. Find the key (which is one byte) and decrypt the message. The message is a meaningful sentence in English!

You should write a code to find the key and decrypt the message. Don't do it manually!

0.3.1 Comment

There are several mini steps to achieve this! First, you need a strategy for searching in the key space. Second, you need a test/scoring mechanism to check whether the decrypted message is meaningful or not (i.e., detecting garbage vs. the correct output). You can read more about *"Caesar"* cipher to get some ideas and more background!

Description *A brief description of your approach. Don't just put the code. First explain what you did and WHY you did it!*

(your description) …

Reference: https://www.codementor.io/@arpitbhayani/deciphering-single-byte-xor-ciphertext-17mtwlzh30

[3]:
```
import codecs
from collections import Counter
from typing import List, Tuple

def single_byte_xor(text: bytes, key) -> bytes:
    return bytes([b ^ key for b in text])

occurance_english = { # estimated frequency of words appear in scentence
    'a': 8.2389258,    'b': 1.5051398,    'c': 2.8065007,    'd': 4.2904556,
    'e': 12.813865,    'f': 2.2476217,    'g': 2.0327458,    'h': 6.1476691,
    'i': 6.1476691,    'j': 0.1543474,    'k': 0.7787989,    'l': 4.0604477,
    'm': 2.4271893,    'n': 6.8084376,    'o': 7.5731132,    'p': 1.9459884,
    'q': 0.0958366,    'r': 6.0397268,    's': 6.3827211,    't': 9.1357551,
    'u': 2.7822893,    'v': 0.9866131,    'w': 2.3807842,    'x': 0.1513210,
    'y': 1.9913847,    'z': 0.0746517
}

dist_english = list(occurance_english.values())

def compute_fitting_quotient(text: bytes) -> float:
    counter = Counter(text) # count the number of times the corresponding␣
 ↪letter/symbol/number appears in the text.
    dist_text = [
        (counter.get(ord(ch), 0) * 100) / len(text) # claculate the frequency␣
 ↪of each letter in the text.
        for ch in occurance_english
    ]
    # return the absolute avg of letter frequency between the current text and␣
 ↪the given occurance_english
    return sum([abs(a - b) for a, b in zip(dist_english, dist_text)]) /␣
 ↪len(dist_text)

def decipher(text: bytes) -> Tuple[bytes, int]:
    """The function deciphers an encrypted text using Single Byte XOR and␣
 ↪returns
    the original plain text message and the encryption key.
    """
    #text = codecs.decode(text, 'hex') # decode hex string to binary form
    original_text, encryption_key, min_fq = None, None, None
    for k in range(256): # loop all possibilities
        # we generate the plain text using encryption key `k`
        _text = single_byte_xor(text, k)
        # we compute the fitting quotient for this decrypted plain text
        _fq = compute_fitting_quotient(_text)

        # if the fitting quotient of this generated plain text is lesser
        # than the minimum seen till now `min_fq` we update.
        if min_fq is None or _fq < min_fq:
            encryption_key, original_text, min_fq = k, _text, _fq

    # return the text and key that has the minimum fitting quotient
    return original_text, encryption_key, min_fq

hex_string =␣
 ↪'1b37373331363f78151b7f2b783431333d78397828372d363c78373e783a393b3736'
hex_byte = codecs.decode(hex_string, 'hex')
text,key,_ = decipher(hex_byte)
print('encrypted text:', text.decode())
print('xor_key:', key)
```

```
encrypted text: Cooking MC's like a pound of bacon
xor_key: 88
```

0.4 Step 4 - Detect single-character XOR

One of the 60-character strings in this file has been encrypted by single-character XOR (each line is one string).

Find it.

0.4.1 Comment

You should use your code in Step 3 to test each line. One line should output a meaningful message. Remeber that you don't know the key either but you can find it for each line (if any).

Description *A brief description of your approach. Don't just put the code. First explain what you did and WHY you did it!*

(your description) ...

```
[4]: from google.colab import drive # mount google drive to google colab
drive.mount('/content/drive')

file = open('/content/drive/Shareddrives/ECEcourse/ECE209AS/CA1/data/04.
 ↪txt','r')
content=file.readlines()
file.close()
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call
drive.mount("/content/drive", force_remount=True).
```

[5]:
```
import pandas as pd
collect = pd.DataFrame(columns=['string','xor_key','min_fq']) # initialize a␣
 ↪dataframe

for hex_string in content:
  hex_string = codecs.decode(hex_string.strip('\n'), 'hex') # convert␣
 ↪hex_string to hex_byte
  text,key,min_fq = decipher(hex_string) # collect decrypted text, key, and␣
 ↪min_fq
  collect = pd.concat([pd.DataFrame([[text,key,min_fq]], columns=collect.
 ↪columns), collect], ignore_index=True) # pad them into a dataframe

idx_min = collect['min_fq'].idxmin() # look for the index where min_fq is the␣
 ↪minimum among all strings
print('the location of the encrypted text:', idx_min)
print('encrypted text:', collect.loc[idx_min][0].decode().strip('\n'))
print('xor_key:', collect.loc[idx_min][1])
```

```
the location of the encrypted text: 156
encrypted text: Now that the party is jumping
xor_key: 53
```

0.5 Step 5 - Implement repeating-key XOR

Here is the opening stanza of an important work of the English language:

```
Burning 'em, if you ain't quick and nimble
I go crazy when I hear a cymbal
```

Encrypt it, under the key "ICE", using repeating-key XOR.

In repeating-key XOR, you'll sequentially apply each byte of the key; the first byte of plaintext will be XOR'd against I, the next C, the next E, then I again for the 4th byte, and so on.

It should come out to:

```
0b3637272a2b2e63622c2e69692a23693a2a3c6324202d623d63343c2a26226324272765272
a282b2f20430a652e2c652a3124333a653e2b2027630c692b20283165286326302e27282f
```

[6]:
```
import codecs
text = "Burning 'em, if you ain't quick and nimble\nI go crazy when I hear a␣
 ↪cymbal"
key = 'ICE'
encode_hex =␣
 ↪'0b3637272a2b2e63622c2e69692a23693a2a3c6324202d623d63343c2a26226324272765272a282b2f20430a65

def repeating_key_xor(text, key): # take two bytes input
  len_key = len(key)
  encoded = []
  for i in range(0, len(text)):
      encoded.append(text[i] ^ key[i % len_key]) # iterating byte in key to␣
 ↪encode text
  return bytes(encoded)

text_byte = codecs.encode(text) # covert the string to type:bytes
key_byte = codecs.encode(key)

text_encode = repeating_key_xor(text_byte,key_byte)
if (text_encode.hex()==encode_hex): # check if decryption correct
  print('decode success')
  print(text_encode.hex())
```

```
decode success
0b3637272a2b2e63622c2e69692a23693a2a3c6324202d623d63343c2a26226324272765272a282b
2f20430a652e2c652a3124333a653e2b2027630c692b20283165286326302e27282f
```

0.6 Step 6 (Main Step) - Break repeating-key XOR

There's a file here. It's been base64'd after being encrypted with repeating-key XOR.

Decrypt it.

Here's how:

- Let KEYSIZE be the guessed length of the key; try values from 2 to (say) 40.
- Write a function to compute the edit distance/Hamming distance between two strings. The Hamming distance is just the number of differing bits. The distance between: `"this is a test"` and `"wokka wokka!!!"` is 37. Make sure your code agrees before you proceed.
- For each KEYSIZE, take the first KEYSIZE worth of bytes, and the second KEYSIZE worth of bytes, and find the edit distance between them. Normalize this result by dividing by KEYSIZE.
- The KEYSIZE with the smallest normalized edit distance is probably the key. You could proceed perhaps with the smallest 2-3 KEYSIZE values. Or take 4 KEYSIZE blocks instead of 2 and average the distances.
- Now that you probably know the KEYSIZE: break the ciphertext into blocks of KEYSIZE length.
- Now transpose the blocks: make a block that is the first byte of every block, and a block that is the second byte of every block, and so on.
- Solve each block as if it was single-character XOR. You already have code to do this. For each block, the single-byte XOR key that produces the best looking histogram is the repeating-key XOR key byte for that block. Put them together and you have the key.

Description *A brief description of your approach. Don't just put the code. First explain what you did and WHY you did it!*

(your description) ...

Reference (1) https://www.educative.io/answers/how-to-break-a-repeating-key-xor-encryption(2) https://arpitbhayani.me/blogs/decipher-repeated-key-xor/

[7]:
```
from google.colab import drive # mount google drive to google colab
drive.mount('/content/drive')

import base64
file = open('/content/drive/Shareddrives/ECEcourse/ECE209AS/CA1/data/06.
 ↪txt','r')
content=file.read()
content=base64.b64decode(content) # convert base64 to byte
print(type(content))
print(len(content))
file.close()
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call
drive.mount("/content/drive", force_remount=True).
<class 'bytes'>
2876
```

[8]:
```
def hamming_distance(text1, text2): # take in two bytes representation
  dist = 0
  for byte1, byte2 in zip(text1, text2):
      dist += bin(byte1 ^ byte2).count('1')
  return dist

str1 = "this is a test"
str2 = "wokka wokka!!!"
str1_byte = str1.encode('utf-8') # encode to byte
str2_byte = str2.encode('utf-8') # encode to byte
print('hamming distance:',hamming_distance(str1_byte, str2_byte)) # should be 37
```

```
hamming distance: 37
```

[9]:
```
from itertools import combinations
# Search for KEYSIZE
def find_key_size(ciphertext, key_range):
    distances = []
    for key_size in key_range:
        blocks = [ciphertext[i:i+key_size] for i in range(0, len(ciphertext),␣
 ↪key_size)][:4] # a list of blocks based on keysize, limit to 4 blocks
        pairs = combinations(blocks, 2) # make combination of blocks
        normalized_distance = sum(hamming_distance(pair[0], pair[1]) for pair␣
 ↪in pairs) / key_size
```

```
        distances.append((key_size, normalized_distance))
    # Sort by normalized distance and return the key size with the smallest␣
  ↪distance
    return min(distances, key=lambda x: x[1])[0]

KEY_RANGE = range(2,40)
KEY_SIZE = find_key_size(content, KEY_RANGE)
print('the keysize:',KEY_SIZE)
```

```
the keysize: 29
```

[10]:

```
# break the encoded text
def break_repeating_key_xor(ciphertext, key_size):
  blocks = [ciphertext[i:i+key_size] for i in range(0,␣
  ↪len(ciphertext)-key_size, key_size)] # break text into blocks of keysize␣
  ↪length, 99 items x 29 bytes
  transpose_blocks = [bytes(b[i] for b in blocks) for i in range(key_size)] #␣
  ↪transpose blocks

  key = [decipher(block)[1] for block in transpose_blocks] # find the key that␣
  ↪contains 29 letter, each in type int
  key_byte = [bytes([item]) for item in key] # convert int to byte
  key_string = ''.join(item.decode() for item in key_byte) # convert byte to␣
  ↪string

  decrypted_text = repeating_key_xor(ciphertext,key_string.encode('utf-8'))
  return decrypted_text.decode(), key_string # output two strings

msg, key = break_repeating_key_xor(content,KEY_SIZE)

print('The key:', key)
print('The message:', msg)
```

```
The key: Terminator X: Bring the noise
The message: I'm back and I'm ringin' the bell
A rockin' on the mike while the fly girls yell
In ecstasy in the back of me
Well that's my DJ Deshay cuttin' all them Z's
Hittin' hard and the girlies goin' crazy
Vanilla's on the mike, man I'm not lazy.

I'm lettin' my drug kick in
It controls my mouth and I begin
To just let it flow, let my concepts go
My posse's to the side yellin', Go Vanilla Go!

Smooth 'cause that's the way I will be
And if you don't give a damn, then
```

Why you starin' at me
So get off 'cause I control the stage
There's no dissin' allowed
I'm in my own phase
The girlies sa y they love me and that is ok
And I can dance better than any kid n' play

Stage 2 -- Yea the one ya' wanna listen to
It's off my head so let the beat play through
So I can funk it up and make it sound good
1-2-3 Yo -- Knock on some wood
For good luck, I like my rhymes atrocious
Supercalafragilisticexpialidocious
I'm an effect and that you can bet
I can take a fly girl and make her wet.

I'm like Samson -- Samson to Delilah
There's no denyin', You can try to hang
But you'll keep tryin' to get my style
Over and over, practice makes perfect
But not if you're a loafer.

You'll get nowhere, no place, no time, no girls
Soon -- Oh my God, homebody, you probably eat
Spaghetti with a spoon! Come on and say it!

VIP. Vanilla Ice yep, yep, I'm comin' hard like a rhino
Intoxicating so you stagger like a wino
So punks stop trying and girl stop cryin'
Vanilla Ice is sellin' and you people are buyin'
'Cause why the freaks are jockin' like Crazy Glue
Movin' and groovin' trying to sing along
All through the ghetto groovin' this here song
Now you're amazed by the VIP posse.

Steppin' so hard like a German Nazi
Startled by the bases hittin' ground
There's no trippin' on mine, I'm just gettin' down
Sparkamatic, I'm hangin' tight like a fanatic
You trapped me once and I thought that
You might have it
So step down and lend me your ear
'89 in my time! You, '90 is my year.

You're weakenin' fast, YO! and I can tell it
Your body's gettin' hot, so, so I can smell it
So don't be mad and don't be sad
'Cause the lyrics belong to ICE, You can call me Dad

You're pitchin' a fit, so step back and endure
Let the witch doctor, Ice, do the dance to cure
So come up close and don't be square
You wanna battle me -- Anytime, anywhere

You thought that I was weak, Boy, you're dead wrong
So come on, everybody and sing this song

Say -- Play that funky music Say, go white boy, go white boy go
play that funky music Go white boy, go white boy, go
Lay down and boogie and play that funky music till you die.

Play that funky music Come on, Come on, let me hear
Play that funky music white boy you say it, say it
Play that funky music A little louder now
Play that funky music, white boy Come on, Come on, Come on
Play that funky music