Cryptography GMI26H

Dalarna University,

Autumn 2022

Advanced Encyption Standard (AES)

Deadline: 27 October, 2022, 23:59 CET time

Student name: Atichoke Nantarat

Student ID: H20atina

University email address:H20atina@du.se

This document is a Jupyter notebook which will be used during the whole course. Students will run one lab every week as follows:

Week 3 of the course Lab 1

Week 4 of the course Lab 2

Week 5 of the course Lab 3

Week 6 of the course Lab 4, 5, and 6

Cryptography Lab 1

1. Key:

In the file **testKey**, a test key is provided. testKey is in plain text but represents bytes. Since AES-256 uses a key that is 256 bit long it corresponds to a key that is 256/8 bytes long. That is, the key is 32 bytes long, where each byte is encoded as two characters in hexadecimals. That is, the 256-bit key is represented as a string of length 64 hexamdecimal characters. We want the key to be transformed into a list of hexadecimal integers.

Method: getKey(filename)

Create the method that does the following:

Method name: getKey()

Input: String filename

Output: List of integers

Hint: The following Python methods may help you to build the getKey():

- hex()
- open()
- file.read()
- string.decode()
- map()
- ord()

```
def getKey(testKey):
    o = []
    mylist = []
    with open(testKey) as f:
        content = f.readline()
        for i in range(0,len(content),2):
            x = i
            y = (content[x:x+2])
            o.append(y)
        for i in o:
            xInt = int(i, base = 16)
            mylist.append(xInt)
        return mylist
```

The following code can be used to test your method implementation. The list after "Output:" is what the "print(key)" command should print out.

```
key = getKey("testKey")
print(key)
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 2
5, 26, 27, 28, 29, 30, 31]
```

Output:

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20,
21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31]
```

2. Block

In the file **testBlock**, a test block, which is a plain text but represents bytes, is provided. Since all versions of AES operate on blocks with the fix size of 128 bits, it corresponds to a block that is 128/8 bytes long. That is, the block is 16 bytes long, where each byte is encoded as two characters in hexadecimal. The 128-bit block is thus represented as a string of length 32.

In reality blocks have the format of a 4x4 matrix. In Python, matrices are typically lists. The first 4 values can be represented as the first column of the matrix, the next 4 as the second column, and so on. You can decide your format of the block yourself.

Method: getBlock(filename)

Create method that does the following:

Method name: getBlock()

Input: String filename

Output: List (or other format) of integers

Hint: The following Python methods may help you to build the method

hex()

open()

file.read()

string.decode()

map()

ord()

https://en.wikipedia.org/wiki/Advanced_Encryption_Standard#High-level_description_of_the_%20algorithm

```python
def getBlock(testBlock):
    o = []
    mylist = []
    with open(testBlock) as f:
        content = f.readline()
        for i in range(0,len(content),2):
            x = i
            y = (content[x:x+2])
            o.append(y)
        for i in o:
            #Transforming the elements int o integers to be able to Hex() them.
            xInt = int(i, base = 16)
            #Hex integers
            # hexInt = hex(xInt)
            #Transforming them into integers
            # output = int(hexInt,base = 16)
            mylist.append(xInt)
        return mylist
```

```python
block = getBlock('testBlock')
print(block)
```

```
[0, 17, 34, 51, 68, 85, 102, 119, 136, 153, 170, 187, 204, 221, 238, 255]
```

Output:

```
[0, 17, 34, 51, 68, 85, 102, 119, 136, 153, 170, 187, 204, 221, 238, 255]
```

3. sbox, sboxInv and rcon

From the following Wikipedia articles:

- https://en.wikipedia.org/wiki/Rijndael_S-box
- https://en.wikipedia.org/wiki/Rijndael_key_schedule

Create three Python code blocks, where each code block contains one of the three lists described in their respective Wikipedia articles.

```
def getBlock_2(testBlock):
    # write your Python code for Block here:
    o = []
    mylist = []
    with open(testBlock,'r') as f:
        txt = ''.join(map(str,f))
        tst = txt.replace('\n','')
        while tst:
            o.append(tst[:2])
            tst = tst[2:]
        for x in o:
            # print(x)
            xInt = int(x, base = 16)
            output = int(hex(xInt), base = 16)
            # print(f"Hexing {x} || got: {hex(xInt)} || int: {int(hex(xInt), base = 16)}")
            mylist.append(output)
        return mylist
        # print(txt)
```

```
bloack = getBlock_2('AES-Sbox.txt')
for i in range(0,255,16):
    x = i
    print(bloack[x:x+16])
```

```
[99, 124, 119, 123, 242, 107, 111, 197, 48, 1, 103, 43, 254, 215, 171, 118]
[202, 130, 201, 125, 250, 89, 71, 240, 173, 212, 162, 175, 156, 164, 114, 192]
[183, 253, 147, 38, 54, 63, 247, 204, 52, 165, 229, 241, 113, 216, 49, 21]
[4, 199, 35, 195, 24, 150, 5, 154, 7, 18, 128, 226, 235, 39, 178, 117]
[9, 131, 44, 26, 27, 110, 90, 160, 82, 59, 214, 179, 41, 227, 47, 132]
[83, 209, 0, 237, 32, 252, 177, 91, 106, 203, 190, 57, 74, 76, 88, 207]
[208, 239, 170, 251, 67, 77, 51, 133, 69, 249, 2, 127, 80, 60, 159, 168]
[81, 163, 64, 143, 146, 157, 56, 245, 188, 182, 218, 33, 16, 255, 243, 210]
[205, 12, 19, 236, 95, 151, 68, 23, 196, 167, 126, 61, 100, 93, 25, 115]
[96, 129, 79, 220, 34, 42, 144, 136, 70, 238, 184, 20, 222, 94, 11, 219]
[224, 50, 58, 10, 73, 6, 36, 92, 194, 211, 172, 98, 145, 149, 228, 121]
[231, 200, 55, 109, 141, 213, 78, 169, 108, 86, 244, 234, 101, 122, 174, 8]
[186, 120, 37, 46, 28, 166, 180, 198, 232, 221, 116, 31, 75, 189, 139, 138]
[112, 62, 181, 102, 72, 3, 246, 14, 97, 53, 87, 185, 134, 193, 29, 158]
[225, 248, 152, 17, 105, 217, 142, 148, 155, 30, 135, 233, 206, 85, 40, 223]
[140, 161, 137, 13, 191, 230, 66, 104, 65, 153, 45, 15, 176, 84, 187, 22]
```

```
bloack = getBlock_2('AES-Sbox_inverse.txt')
print(bloack)
```

```
[82, 9, 106, 213, 48, 54, 165, 56, 191, 64, 163, 158, 129, 243, 215, 251, 124, 227, 57, 130,
155, 47, 255, 135, 52, 142, 67, 68, 196, 222, 233, 203, 84, 123, 148, 50, 166, 194, 35, 61,
238, 76, 149, 11, 66, 250, 195, 78, 8, 46, 161, 102, 40, 217, 36, 178, 118, 91, 162, 73, 10
9, 139, 209, 37, 114, 248, 246, 100, 134, 104, 152, 22, 212, 164, 92, 204, 93, 101, 182, 14
6, 108, 112, 72, 80, 253, 237, 185, 218, 94, 21, 70, 87, 167, 141, 157, 132, 144, 216, 171,
0, 140, 188, 211, 10, 247, 228, 88, 5, 184, 179, 69, 6, 208, 44, 30, 143, 202, 63, 15, 2, 19
3, 175, 189, 3, 1, 19, 138, 107, 58, 145, 17, 65, 79, 103, 220, 234, 151, 242, 207, 206, 24
0, 180, 230, 115, 150, 172, 116, 34, 231, 173, 53, 133, 226, 249, 55, 232, 28, 117, 223, 11
0, 71, 241, 26, 113, 29, 41, 197, 137, 111, 183, 98, 14, 170, 24, 190, 27, 252, 86, 62, 75,
198, 210, 121, 32, 154, 219, 192, 254, 120, 205, 90, 244, 31, 221, 168, 51, 136, 7, 199, 49,
177, 18, 16, 89, 39, 128, 236, 95, 96, 81, 127, 169, 25, 181, 74, 13, 45, 229, 122, 159, 14
```

```
7, 201, 156, 239, 160, 224, 59, 77, 174, 42, 245, 176, 200, 235, 187, 60, 131, 83, 153, 97,
```

```
x = "123456789"
mylist = []
for i in x:
    s = hex(int(i,base = 16))
    mylist.append(s)

print(mylist)
(hex(141))
```

```
['0x1', '0x2', '0x3', '0x4', '0x5', '0x6', '0x7', '0x8', '0x9']
'0x8d'
```

```
# write your Python code for Sbox, sboxInv and rcon here:
sbox = [99, 124, 119, 123, 242, 107, 111, 197, 48, 1, 103, 43, 254, 215, 171, 118, 202, 130,
sboxInv = [82, 9, 106, 213, 48, 54, 165, 56, 191, 64, 163, 158, 129, 243, 215, 251, 124, 227
# rcon = [0x8d, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08, 0x09, 0x0a, 0x0b, 0x0c, 0x0d
rcon = [0x8d, 0x01, 0x02, 0x04, 0x08, 0x10,0x20,0x40,0x80,0x1b,0x36,0x6c,0xd8,0xab,0x4d,0x9a
```

```
#Check that you can run the following commands to see if the lists map integers correctly:
print(sbox[0])
print(sboxInv[0])
print(rcon[0])
```

```
99
82
141
```

Output:

```
99 82 141
```

Cryptography Lab 2

1. shiftRows

shiftRows is a static block-operator. Meaning that it performs the same action every time it is called. It simply shifts each row to the left the same number of steps as the current row number. See: https://en.wikipedia.org/wiki/Advanced_Encryption_Standard#The_ShiftRows_step

Depending on how you have structured your block the implementation for this will be different. A list can be used to represent the matrix, where the first 4 elements corresponds to the first column, the next 4 elements corresponds to the second column, and so on. You can use either matrices, lists of lists or just a normal list.

Since this method is static you can either just make a temporary block and move the elements around as described or you can loop over each row and perform the shift depending on the current row number.

Create the following two methods: One called shiftRows(block) and one called shiftRowsInv(block). These methods are extremely similar but the reverse shift just moves elements to the right instead of left.

I got the code idea for shiftRows idea from Creel: https://www.youtube.com/watch?v=7uRK9iOk4uk&t=351s

```
# Add shiftRows() here:
def shiftRows(block):

    s0 = block[0::4]
    s1 = block[1::4]
    s2 = block[2::4]
    s3 = block[3::4]

    s1 = s1[1:] + s1[:1]
    s2 = s2[2:] + s2[:2]
    s3 = s3[3:] + s3[:3]

    # print(s0)
    # print(s1)
    # print(s2)
    # print(s3)

    x1 = [s0[0],s1[0],s2[0],s3[0]]
    x2 = [s0[1],s1[1],s2[1],s3[1]]
    x3 = [s0[2],s1[2],s2[2],s3[2]]
    x4 = [s0[3],s1[3],s2[3],s3[3]]
    x5 = x1+x2+x3+x4
    # print(x5)
    return x5
```

```
# Add shiftRowsInv() here:
def shiftRowsInv(block):
    s0 = block[0::4]
    s1 = block[1::4]
    s2 = block[2::4]
    s3 = block[3::4]

    s1 = s1[-1:] + s1[:-1]
    s2 = s2[-2:] + s2[:-2]
    s3 = s3[-3:] + s3[:-3]

    # print(s0)
    # print(s1)
    # print(s2)
    # print(s3)

    x1 = [s0[0],s1[0],s2[0],s3[0]]
    x2 = [s0[1],s1[1],s2[1],s3[1]]
    x3 = [s0[2],s1[2],s2[2],s3[2]]
    x4 = [s0[3],s1[3],s2[3],s3[3]]
    x5 = x1+x2+x3+x4
    return x5
```

```
#Test the shiftRows() using this code

block = getBlock('testBlock')

shiftedBlock = shiftRows(block)

print(shiftedBlock)
```

```
[0, 85, 170, 255, 68, 153, 238, 51, 136, 221, 34, 119, 204, 17, 102, 187]
```

Output:

[0, 85, 170, 255, 68, 153, 238, 51, 136, 221, 34, 119, 204, 17, 102, 187]

```
#Test the shiftRowsInv() using this Code:

block = getBlock('testBlock')

shiftedBlock = shiftRows(block)

unShiftedBlock = shiftRowsInv(shiftedBlock)

print(unShiftedBlock)
```

```
[0, 17, 34, 51, 68, 85, 102, 119, 136, 153, 170, 187, 204, 221, 238, 255]
```

Output:

[0, 17, 34, 51, 68, 85, 102, 119, 136, 153, 170, 187, 204, 221, 238, 255]

2. mixColumn()

The mix of columns is a shuffle of each individual column in the block. This shuffling happens in a deterministic way depending on the values within the column. For consultation of the exact way of doing this, see the following Wikipedia article: https://en.wikipedia.org/wiki/Rijndael_mix_columns#MixColumns

You can either implement Galois multiplication and perform the mix directly. Or you use the pre-calculated lists found in the article. To do AES mixing of columns one only needs to multiply elements with 1, 2 or 3. And for the un-mixing 9, 11, 13 or 14.

Create method mixColumn() as follows:

This method should take one column perform the multiplications in accordance to the multiplication tables. Refer to the Wikipedia article.

Input: column

Output: column

Remember that in this context + means XOR. In Python, the hat-symbol (caret), ˆ, is used for XOR.

Also create the inverse method mixColumnInv().

MDS Matrix [d0] 2 3 1 1 [b0] [d1] 1 2 3 1 [b1] [d2] 1 1 2 3 [b2] [d3] 3 1 1 2 [b3]

Min matrix 0 68 136 204 17 85 153 221 34 102 170 238 51 119 187 255

[d0] = (2*68) ⊕ (3*85) ⊕ (1*102) ⊕ (1*119) OUTPUT: 34, 119, 0, 85, 102, 51, 68, 17, 170, 255, 136, 221,238, 187, 204, 153

```python
#https://gist.github.com/bonsaiviking/5571001
#used for the mixcolumn for both to mix and inverse mix.

def gmul(a, b):
    p = 0
    for c in range(8):
        if b & 1: # if * by 1
            p ^= a
        a <<= 1 #shift left
        if a & 0x100: #256
            a ^= 0x11b #283
        b >>= 1 #shift right
    return p
```

I discussed with a fellow student Mikael Olsson which helped me to understand galois multiplication, I later created the function which is seen below, however i noticed later on that that function had problems later on for mixcolumns and inverse, so I tried to fix it. I stumbled uppon the gmul() above which satisfied the condition for galois multiplication and needed no further coding on it.

```
# def gfMult(x,y):
#     if y == 1:
#         return x
#     tmp = (x << 1)
#     if y == 2:
#         return tmp
#     if y == 3:
#         return gfMult(x,2)^x
```

```
# Add mixColumn() here:
def mixColumn(column):
    ConstMatrix = [2,3,1,1,1,2,3,1,1,1,2,3,3,1,1,2]
    newList = []
    count = 0
    counter = 0
    for i in range(len(column)):
        # a = gfMult(column[count],ConstMatrix[counter]) ^ gfMult(column[count+1],ConstMatri
        a = gmul(column[count],ConstMatrix[counter]) ^ gmul(column[count+1],ConstMatrix[cour
        counter += 4
        newList.append(a)
    return newList
```

```
# Add mixColumnInv() here:
def mixColumnInv(column):
    ConstMatrix = [14,11,13,9,9,14,11,13,13,9,14,11,11,13,9,14]
    newList = []
    count = 0
    counter = 0
    for i in range(len(column)):
        a = gmul(column[count],ConstMatrix[counter]) ^ gmul(column[count+1],ConstMatrix[cour
        counter += 4
        newList.append(a)
    return newList
```

Next create method mixColumns() that calls the mixColumn(col) for each column in your block and then returns the mixed block.

```python
# Add mixColumns() here:
def mixColumns(block):
    newList = []
    start = 0
    end = 16
    step = 4
    for i in range(start,end,step):
        x = i
        column = (block[x:x+step])
        # print(f'Column: {column}')
        a = mixColumn(column)
        newList.extend(a)
    return newList
```

The corresponding inverse function should simply call the mixColumnInv() method for each column in your block and then return the mixed block.

```python
# Add mixColumnsInv() here:
def mixColumnsInv(block):
    newList = []
    step = 4
    for i in range(0,len(block),step):
        x = i
        column = (block[x:x+step])
        # print(f'Column: {column}')
        a = mixColumnInv(column)
        newList.extend(a)
    return newList
```

Use the following code to test your methods

```python
testBlock = getBlock('testBlock')

print(testBlock)

mixedBlock = mixColumns(testBlock)

print(mixedBlock)

unMixedBlock = mixColumnsInv(mixedBlock)

print(unMixedBlock)
```

```
[0, 17, 34, 51, 68, 85, 102, 119, 136, 153, 170, 187, 204, 221, 238, 255]
[34, 119, 0, 85, 102, 51, 68, 17, 170, 255, 136, 221, 238, 187, 204, 153]
[0, 17, 34, 51, 68, 85, 102, 119, 136, 153, 170, 187, 204, 221, 238, 255]
```

Output:

```
[0, 17, 34, 51, 68, 85, 102, 119, 136, 153, 170, 187, 204, 221, 238, 255]

[34, 119, 0, 85, 102, 51, 68, 17, 170, 255, 136, 221, 238, 187, 204, 153]

[0, 17, 34, 51, 68, 85, 102, 119, 136, 153, 170, 187, 204, 221, 238, 255]
```

3. subBytes

Create method subBytes() and apply the sBox to each element in the block and return it.

```python
# Add subBytes() here:
def subBytes(block):
    newList = []
    for i in range(len(block)):
        # block[i] = sbox[block[i]]
        x = block[i]
        i = sbox[x]
        newList.append(i)
    # print(newList)
    return newList
```

Create method subBytesInv() and apply sBoxInv() to each element in the block. And return it.

```python
# Add subBytesInv() here:
def subBytesInv(block):
    newList = []
    for i in range(len(block)):
        x = block[i]
        i = sboxInv[x]
        # block[i] = sboxInv[block[i]]
        newList.append(i)
    # print(newList)
    return newList
```

Use the follwing code to test your methods:

```python
testBlock = getBlock('testBlock')

print(testBlock)

substitutedBlock = subBytes(testBlock)

print(substitutedBlock)

unSubstitutedBlock = subBytesInv(substitutedBlock)

print(unSubstitutedBlock)
```

```
[0, 17, 34, 51, 68, 85, 102, 119, 136, 153, 170, 187, 204, 221, 238, 255]
[99, 130, 147, 195, 27, 252, 51, 245, 196, 238, 172, 234, 75, 193, 40, 22]
[0, 17, 34, 51, 68, 85, 102, 119, 136, 153, 170, 187, 204, 221, 238, 255]
```

Output:

[0, 17, 34, 51, 68, 85, 102, 119, 136, 153, 170, 187, 204, 221, 238, 255]

[99, 130, 147, 195, 27, 252, 51, 245, 196, 238, 172, 234, 75, 193, 40, 22]

[0, 17, 34, 51, 68, 85, 102, 119, 136, 153, 170, 187, 204, 221, 238, 255]

Cryptography Lab 3

1. keyScheduleCore()

Read about this method in https://en.wikipedia.org/wiki/Rijndael_key_schedule#Key_schedule_core
or in NIST document paragraph 5.2 (Key Expansion)
Input: 4 byte word and Iteration Number

Output: 4 byte word

Explain what you do and why here in this cell (in English): For the keyScheduleCore function it takes 2 parameters word and Nk.

1. the elements from words gets shifted to the left.
2. shifted elements gets replaced by the elements from the sbox and added to the newList list.
3. the first element in word gets XOR with round constant which value is up to the Nk that is sent in.

```python
# Add keyScheduleCore() here:
def keyScheduleCore(word,Nk):
    newList = []
    # shift the elements to the left.
    s1 = word[1:] + word[:1]
    # print(s1)
    #subbytes
    for i in range(len(word)):
        x = s1[i]
        y = sbox[x]
        newList.append(y)
    #Rcon
    newList[0] ^= rcon[Nk]
    return newList
```

```
#Test your code
word = [1,2,3,4]

newWord = keyScheduleCore(word,1)

print(word)

print(newWord)
```

```
[1, 2, 3, 4]
[118, 123, 242, 124]
```

Output:

```
[1, 2, 3, 4]

[118, 123, 242, 124]
```

2. expandKey()

AES-operations are performed on blocks multiple times, in so-called rounds. Each round uses a specific round key. The round keys are created from the original key via an extension. The original key is 256 bit (32 bytes) but needs to be expanded to a 240 byte key. This extension is done using Rijndael key schedule, see https://en.wikipedia.org/wiki/Rijndael_key_schedule#Key_schedule_description.
Follow the algorithmic description in the Wikipedia page and create a method called expandKey().

Method name: expandKey(key)

Input: 256 bit key

Output: extended 240 byte key

Explain what you do and why here in this cell (in English): The expandKey takes the key and expands it to 240 bytes. The formular is from the FISP PUB 197 about Key expansion, "The Key Expansion generates a total of Nb (Nr + 1) words"(FIPS PUB 197, "Key Expansion" section.)

A 32 bytes key is sent in and divided into words. The function will expand and create new words from 1st and last word getting sent to the keyScheduleCore to later XOR with the 1st word which gives us a new word (let us call it word 4), the word 4 is then XOR with the 2nd word and gives us word 5, this repeats until there are no words for the lates word to XOR with, the latest newest word then gets sent to keyScheduleCore to be able to further create more words, all this happens until the key has been expended to 240 bytes.

```
#https://www.geeksforgeeks.org/python-convert-a-nested-list-into-a-flat-list/
#used to flatten a list.
def flatten(l):
    return [element for innerList in l for element in innerList]
```

```
def expandKey(key):
    keyList = []
    tempList =[]

    Nk = 8
    Nb = 4
    Nr = 14
    i = 0

    for i in range(0,32,4):
        x = i
        y = key[x:x+4]
        keyList.append(y)

    i = Nk

    while(i < Nb * (Nr+1)):
        round = i//Nk
        temp = keyList[i-1]
        if(i%Nk ==0):
            temp = keyScheduleCore(temp,round)
        elif(i%Nk==4):
            temp = subBytes(temp)

        for x in range(0,4):
            y = keyList[i-Nk][x]^temp[x]
            tempList.append(y)
        keyList.append(tempList)
        tempList = []
        i+=1

    return flatten(keyList)
```

```
#Test your code
key = getKey('testKey')
print(key)
expandedKey = expandKey(key)
print(expandedKey)
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 2
5, 26, 27, 28, 29, 30, 31]
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 2
5, 26, 27, 28, 29, 30, 31, 165, 115, 194, 159, 161, 118, 196, 152, 169, 127, 206, 147, 165,
114, 192, 156, 22, 81, 168, 205, 2, 68, 190, 218, 26, 93, 164, 193, 6, 64, 186, 222, 174, 13
5, 223, 240, 15, 241, 27, 104, 166, 142, 213, 251, 3, 252, 21, 103, 109, 225, 241, 72, 111,
165, 79, 146, 117, 248, 235, 83, 115, 184, 81, 141, 198, 86, 130, 127, 201, 167, 153, 23, 11
1, 41, 76, 236, 108, 213, 89, 139, 61, 226, 58, 117, 82, 71, 117, 231, 39, 191, 158, 180, 8
4, 7, 207, 57, 11, 220, 144, 95, 194, 123, 9, 72, 173, 82, 69, 164, 193, 135, 28, 47, 69, 24
5, 166, 96, 23, 178, 211, 135, 48, 13, 77, 51, 100, 10, 130, 10, 124, 207, 247, 28, 190, 18
0, 254, 84, 19, 230, 187, 240, 210, 97, 167, 223, 240, 26, 250, 254, 231, 168, 41, 121, 215,
165, 100, 74, 179, 175, 230, 64, 37, 65, 254, 113, 155, 245, 0, 37, 136, 19, 187, 213, 90, 1
14, 28, 10, 78, 90, 102, 153, 169, 242, 79, 224, 126, 87, 43, 170, 205, 248, 205, 234, 36, 2
52, 121, 204, 191, 9, 121, 233, 55, 26, 194, 60, 109, 104, 222, 54]
```

Output:

[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31]

[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 165, 115, 194, 159, 161, 118, 196, 152, 169, 127, 206, 147, 165, 114, 192, 156, 22, 81, 168, 205, 2, 68, 190, 218, 26, 93, 164, 193, 6, 64, 186, 222, 174, 135, 223, 240, 15, 241, 27, 104, 166, 142, 213, 251, 3, 252, 21, 103, 109, 225, 241, 72, 111, 165, 79, 146, 117, 248, 235, 83, 115, 184, 81, 141, 198, 86, 130, 127, 201, 167, 153, 23, 111, 41, 76, 236, 108, 213, 89, 139, 61, 226, 58, 117, 82, 71, 117, 231, 39, 191, 158, 180, 84, 7, 207, 57, 11, 220, 144, 95, 194, 123, 9, 72, 173, 82, 69, 164, 193, 135, 28, 47, 69, 245, 166, 96, 23, 178, 211, 135, 48, 13, 77, 51, 100, 10, 130, 10, 124, 207, 247, 28, 190, 180, 254, 84, 19, 230, 187, 240, 210, 97, 167, 223, 240, 26, 250, 254, 231, 168, 41, 121, 215, 165, 100, 74, 179, 175, 230, 64, 37, 65, 254, 113, 155, 245, 0, 37, 136, 19, 187, 213, 90, 114, 28, 10, 78, 90, 102, 153, 169, 242, 79, 224, 126, 87, 43, 170, 205, 248, 205, 234, 36, 252, 121, 204, 191, 9, 121, 233, 55, 26, 194, 60, 109, 104, 222, 54]

3. createRoundKey()

Method createRoundKey() samples the expanded key. In total there are 14 rounds in AES-256, and 14 round keys need to be extracted from the expanded key.

Method: createRoundKey(expandedKey, n)

Input: expandedKey (240 bytes) and round number n

Output: roundKey (16 bytes)

Explain what you do and why here in this cell (in English):

I divide the expandedKey into list, each list containing 16 elements which later gets added to roundList list which is returned out of the function.

n is for which round it is.

```
# createRoundKey(expandedKey, n) method here:
def createRoundKey(expandedKey,n):
    roundList = []
    count = -1
    for i in range(0,240,16):

        x = i
        y= expandedKey[x:x+16]
        # print(y)
        roundList.append(y)
        count +=1
    return roundList[n]
```

```
#Test your Code:
key = getKey('testKey')
expandedKey = expandKey(key)
roundKey0 = createRoundKey(expandedKey,0)
roundKey7 = createRoundKey(expandedKey,7)
roundKey14 = createRoundKey(expandedKey,14)
print(expandedKey)
print(roundKey0)
print(roundKey7)
print(roundKey14)
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 2
5, 26, 27, 28, 29, 30, 31, 165, 115, 194, 159, 161, 118, 196, 152, 169, 127, 206, 147, 165,
114, 192, 156, 22, 81, 168, 205, 2, 68, 190, 218, 26, 93, 164, 193, 6, 64, 186, 222, 174, 13
5, 223, 240, 15, 241, 27, 104, 166, 142, 213, 251, 3, 252, 21, 103, 109, 225, 241, 72, 111,
165, 79, 146, 117, 248, 235, 83, 115, 184, 81, 141, 198, 86, 130, 127, 201, 167, 153, 23, 11
1, 41, 76, 236, 108, 213, 89, 139, 61, 226, 58, 117, 82, 71, 117, 231, 39, 191, 158, 180, 8
4, 7, 207, 57, 11, 220, 144, 95, 194, 123, 9, 72, 173, 82, 69, 164, 193, 135, 28, 47, 69, 24
5, 166, 96, 23, 178, 211, 135, 48, 13, 77, 51, 100, 10, 130, 10, 124, 207, 247, 28, 190, 18
0, 254, 84, 19, 230, 187, 240, 210, 97, 167, 223, 240, 26, 250, 254, 231, 168, 41, 121, 215,
165, 100, 74, 179, 175, 230, 64, 37, 65, 254, 113, 155, 245, 0, 37, 136, 19, 187, 213, 90, 1
14, 28, 10, 78, 90, 102, 153, 169, 242, 79, 224, 126, 87, 43, 170, 205, 248, 205, 234, 36, 2
52, 121, 204, 191, 9, 121, 233, 55, 26, 194, 60, 109, 104, 222, 54]
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]
[61, 226, 58, 117, 82, 71, 117, 231, 39, 191, 158, 180, 84, 7, 207, 57]
[36, 252, 121, 204, 191, 9, 121, 233, 55, 26, 194, 60, 109, 104, 222, 54]
```

Output:

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31,
165, 115, 194, 159, 161, 118, 196, 152, 169, 127, 206, 147, 165, 114, 192, 156, 22, 81, 168, 205, 2, 68, 190,
218, 26, 93, 164, 193, 6, 64, 186, 222, 174, 135, 223, 240, 15, 241, 27, 104, 166, 142, 213, 251, 3, 252, 21,
103, 109, 225, 241, 72, 111, 165, 79, 146, 117, 248, 235, 83, 115, 184, 81, 141, 198, 86, 130, 127, 201, 167,
153, 23, 111, 41, 76, 236, 108, 213, 89, 139, 61, 226, 58, 117, 82, 71, 117, 231, 39, 191, 158, 180, 84, 7,
207, 57, 11, 220, 144, 95, 194, 123, 9, 72, 173, 82, 69, 164, 193, 135, 28, 47, 69, 245, 166, 96, 23, 178, 211,
135, 48, 13, 77, 51, 100, 10, 130, 10, 124, 207, 247, 28, 190, 180, 254, 84, 19, 230, 187, 240, 210, 97, 167,
223, 240, 26, 250, 254, 231, 168, 41, 121, 215, 165, 100, 74, 179, 175, 230, 64, 37, 65, 254, 113, 155, 245,
0, 37, 136, 19, 187, 213, 90, 114, 28, 10, 78, 90, 102, 153, 169, 242, 79, 224, 126, 87, 43, 170, 205, 248,
205, 234, 36, 252, 121, 204, 191, 9, 121, 233, 55, 26, 194, 60, 109, 104, 222, 54]

[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]

[61, 226, 58, 117, 82, 71, 117, 231, 39, 191, 158, 180, 84, 7, 207, 57]

[36, 252, 121, 204, 191, 9, 121, 233, 55, 26, 194, 60, 109, 104, 222, 54]
```

4. addRoundKey()

addRoundKey() XORs each element of a roundkey with a block.
See: https://en.wikipedia.org/wiki/Advanced_Encryption_Standard#The_AddRoundKey_step

Method: addRoundKey(block,roundKey)

Input: block (16 bytes) and roundKey (16 bytes)

Output: block (16 bytes)

Explain what you do and why here in this cell (in English): The function takes 2 parameters block and roundKey. The elements in the block gets XOR with the round key and is later added to the State which in this case is block.

```python
# addRoundKey(block,roundKey) method here:
def addRoundKey(block,roundKey):
    tempList = []
    for i in range(len(block)):
        y = block[i]^roundKey[i]
        tempList.append(y)
    return tempList
```

```
#Test your Code:

key = getKey('testKey')
testBlock = getBlock('testBlock')
# print(len(testBlock))
expandedKey = expandKey(key)
roundKey0 = createRoundKey(expandedKey,0)
addedRoundKeyToBlock = addRoundKey(testBlock,roundKey0)
print(addedRoundKeyToBlock)
```

```
[0, 16, 32, 48, 64, 80, 96, 112, 128, 144, 160, 176, 192, 208, 224, 240]
```

Output:

[0, 16, 32, 48, 64, 80, 96, 112, 128, 144, 160, 176, 192, 208, 224, 240]

Cryptography Lab 4, 5, and 6

A large text file tWotW.txt is now available on Learn. Your task in Lab 6 is to read this file,Split it into blocks, Encrypt the blocks, and produce an encrypted version of the file. Then, read the encrypted file block by block and decrypt it. Upload the encrypted version of tWOW.txt to Learn, as well.
Labs 1-6 must be merged in one JNB file and that file must be compressed with the corresponding .pdf file and encryptedtWOW.txt (the encrypted version of tWOW.txt), that is, your final submission includes three files compressed into one .rar or .zip file:

1. AES labs 1-6 in one .ipynb file,
2. the .ipynb file converted into .pdf, and
3. encryptedtWOW.txt by you.

Add brief explanations regarding how you built each method, and why each method is needed (in English).

Sometimes it is good to define helper methods in addition to the main methods before the actual methods. You may add helper methods before the main methods in the same Python cell.

Lab 4

Encryption

By now all essential parts of AES has been implemented. What is left to do is to call all the methods in the correct order to encrypt a block. We are going to follow the description provided here:

https://en.wikipedia.org/wiki/Advanced_Encryption_Standard#High-level_description_of_the_algorithm

More specifically, the encryption method should perform the following steps:

encrypt(block, key)

Method name: encrypt(blocks, key)

Input: block and key

Output: encrypted block

Explain what you do and why here in this cell (in English):

1. I copy all elements from block into state list
2. I create expandedKey from key to be able to add round keys later on for state.
3. A round key is later created for the 0-th round. addRoundKey is later added to state.
4. A loop is later created for the 1-st to 13-th round, elements in the state is replaced with elements from sbox with the method subBytes. the state's elements are later shifted to left then columns in state are mixed. Lastly a round key is added to the state.
5. for the 14-th round state elements are replaced with sbox elements then shifted and lastly a round key is added.

Legend for CIPHER (ENCRYPT) (round number r = 0 to 10, 12 or 14): input: cipher input start: state at start of round[r] s_box: state after SubBytes() s_row: state after ShiftRows() m_col: state after MixColumns() k_sch: key schedule value for round[r] output: cipher output

Legend for INVERSE CIPHER (DECRYPT) (round number r = 0 to 10, 12 or 14): iinput: inverse cipher input istart: state at start of round[r] is_box: state after InvSubBytes() is_row: state after InvShiftRows() ik_sch: key schedule value for round[r] ik_add: state after AddRoundKey() ioutput: inverse cipher output

input: cipher input start: state at start of round[r] s_box: state after SubBytes() s_row: state after ShiftRows() m_col: state after MixColumns() k_sch: key schedule value for round[r] output: cipher output

```
def encrypt(block,key):
    state = []
    state.extend(block)

    # #**Initialize:**
    expandedKey = expandKey(key)
    #**0-th round:**
    roundKey = createRoundKey(expandedKey,0)
    state = addRoundKey(state,roundKey)
    #**1-st to 13:th round:**
    for i in range(1,14):
        state = subBytes(state)
        state = shiftRows(state)
        state = mixColumns(state)
        roundKey = createRoundKey(expandedKey,i)
        state = addRoundKey(state,roundKey)

    # #**14:th round:**
    state = subBytes(state)
    state = shiftRows(state)
    roundKey = createRoundKey(expandedKey,14)
    state = addRoundKey(state,roundKey)
    #**Return:** block
    return state
```

```
#test your Code:
key = getKey('testKey')
block = getBlock('testBlock')
print(f'block:{block}')
print(f'key:{key}')
encryptedBlock = encrypt(block,key)
print(f'EncryptedBlock: {encryptedBlock}')
```

```
block:[0, 17, 34, 51, 68, 85, 102, 119, 136, 153, 170, 187, 204, 221, 238, 255]
key:[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 2
4, 25, 26, 27, 28, 29, 30, 31]
EncryptedBlock: [142, 162, 183, 202, 81, 103, 69, 191, 234, 252, 73, 144, 75, 73, 96, 137]
```

Output:

```
[0, 17, 34, 51, 68, 85, 102, 119, 136, 153, 170, 187, 204, 221, 238, 255]

[142, 162, 183, 202, 81, 103, 69, 191, 234, 252, 73, 144, 75, 73, 96, 137]
```

Lab 5

Decryption

Again, see: https://en.wikipedia.org/wiki/Advanced_Encryption_Standard#High-level_description_of_the_algorithm

decrypt(block, key)

Input: block and key

Output: decrypted block

```
# decrypt(block, key) method here:
def decrypt(block,key):
    state = []
    state.extend(block)
    expandedKey = expandKey(key)

    #**14-th round:**
    roundKey = createRoundKey(expandedKey,14)
    state = addRoundKey(state, roundKey)
    state = shiftRowsInv(state)
    state = subBytesInv(state)

    #**13-th to 1:st round:**
    for i in range(1,14):
        x = 14-i
        roundKey = createRoundKey(expandedKey,x)
        state = addRoundKey(state, roundKey)
        state = mixColumnsInv(state)
        state = shiftRowsInv(state)
        state = subBytesInv(state)

    #**0:th round:**
    roundKey = createRoundKey(expandedKey,0)
    state = addRoundKey(state, roundKey)
    return state
    #**Return:** block
```

Explain what you do and why here in this cell (in English): It works like encrypt function just in reverse

1. a round key is created, state is later shifted and lastly state elements are replaced with sboxInv elements.
2. for the 13-th to 1st round: round key is added, the columns in state are mixed, elements in state is later shifted to the right and then lastly replaced with sboxInv elements.
3. 0-th round round key is added.

```python
#Test your Code:

key = getKey('testKey')
block = getBlock('testBlock')
print(f'block:{block}')

encryptedBlock = encrypt(block,key)
print(f'Encrypted:{encryptedBlock}')
decryptedBlock = decrypt(encryptedBlock,key)
print(f'Decrypted:{decryptedBlock}')
```

```
block:[0, 17, 34, 51, 68, 85, 102, 119, 136, 153, 170, 187, 204, 221, 238, 255]
Encrypted:[142, 162, 183, 202, 81, 103, 69, 191, 234, 252, 73, 144, 75, 73, 96, 137]
Decrypted:[0, 17, 34, 51, 68, 85, 102, 119, 136, 153, 170, 187, 204, 221, 238, 255]
```

Output:

```
[0, 17, 34, 51, 68, 85, 102, 119, 136, 153, 170, 187, 204, 221, 238, 255]

[142, 162, 183, 202, 81, 103, 69, 191, 234, 252, 73, 144, 75, 73, 96, 137]

[0, 17, 34, 51, 68, 85, 102, 119, 136, 153, 170, 187, 204, 221, 238, 255]
```

More specifically, the decryption method should perform the following steps:

Lab 6

By now you should have implemented the methods encrypt() and decrypt(). Both take one block and one key as input and output a block. In the last lab you will apply your implementation to a larger file, analyse your results and prepare for your examination on the project.

A large text file tWotW.txt is now available on Learn. Your task is to read this file, split it into blocks,encrypt the blocks, and produce an encrypted version of the file. Read the encrypted file block by block and decrypt it.

```
def readFile(book):
    o = [] * 32
    mylist = []
    tempList = []
    #Read the file.
    with open(book,mode = 'r', encoding='utf-8') as f:
        content = f.read()
        o.extend(content)
    #Split the file into blocks of 16.
    for i in range(0,len(o),16):
        x = i
        y = (o[x:x+16])
        for j in y:
            tempList.append(ord(j))
        mylist.append(tempList)
        tempList = []

    mylist[-1] = [100,46,10,32,32,32,32,32,32,32,32,32,32,32,32,32]
    return mylist
```

```
#convert int to str
def convertToStr(l):
    res = int(''.join(map(str,l)))
    return res
```

```
#https://www.geeksforgeeks.org/python-convert-list-characters-string/
def convert(s):
    msg = ''
    for x in s:
        msg += x
    return msg
```

```
import time
encryptTime = time.time()
listOfBlocks = readFile('tWotW.txt')
# print(f'List of block:{listOfBlocks}')
print(len(listOfBlocks))
print(listOfBlocks[0])
```

```
21501
[84, 104, 101, 32, 87, 97, 114, 32, 111, 102, 32, 116, 104, 101, 32, 87]
```

```
listOfEncryptedBlocks=[]
key = getKey('testKey')
for i in listOfBlocks:
    encryptedBlock = encrypt(i,key)
    listOfEncryptedBlocks.append(encryptedBlock)
```

```
for i in range(len(listOfEncryptedBlocks)):
    x = listOfEncryptedBlocks[i]
# print(convertToStr(x))
```

```
with open('tWotWEncrypted',mode = 'w+', encoding='utf-8') as f:
    content = f.read()
    output = ' '.join(str(i)for i in flatten(listOfEncryptedBlocks))
    f.write(output)
endTimer = time.time()
```

```
print(f'Time to read>encrypt>save: {endTimer-encryptTime}')
```

```
Time to read>encrypt>save: 29.57697319984436
```

```
decryptTime = time.time()
blocksToDecrypt =[]
tp = []
step = 16
with open('tWotWEncrypted',mode = 'r', encoding='utf-8') as f:
    content = f.readline()
    y = (content.split(' '))
    tp.extend(y)
for i in tp:
    blocksToDecrypt.append(int(i))
tp = []
for i in range(0,len(blocksToDecrypt),step):
    x = i
    y = blocksToDecrypt[x:x+step]
    tp.append(y)
```

```
listOfDecryptedBlocks=[]
for i in tp:
    # print(i)
    decryptedBlock = decrypt(i,key)
    # print(decryptedBlock)
    listOfDecryptedBlocks.append(decryptedBlock)
tempList = [listOfDecryptedBlocks[-1][0],listOfDecryptedBlocks[-1][1],listOfDecryptedBlocks[
listOfDecryptedBlocks[-1] = tempList
tempList = []
```

```
# for i in range(4):
#     print(f'\nListOfBlocks: {listOfBlocks[i]}')
#     print(f'ListOfEncryptedBlocks:{listOfEncryptedBlocks[i]}')
#     print(f'ListOfDecrypted:{listOfDecryptedBlocks[i]}\n')
```

Convert function to convert the list that contains a list of chr(s) that later is saved as a string to save as a text file.

This function dencrypts the numbers to characters.

```
listOfChar = []
for items in listOfDecryptedBlocks:
    for i in items:
        toChr = chr(i)
        listOfChar.extend(toChr)
```

```
convertedChr = convert(listOfChar)
```

Method below saves the contents as a text file.

```
with open('tWotWDecrypted.txt',mode = 'w+', encoding='utf-8') as f:
    content = f.read()
    f.write(convertedChr)
endTimer2 = time.time()
```

```
print(f'Time to read>decrypt>store: {endTimer2-decryptTime}')
```

```
Time to read>decrypt>store: 32.57912993431091
```

Key

Question 1: What is the encryption key are you using (in hex format)?

Question 1 answer: The encryption key I use are from testKey: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31

Question 2: What is the avalanche effect in cryptography? If one bit is changed in a key, what should happen to the ciphertext? This is a theoretical question, no need to demonstrate this in practice.

Question 2 answer:

The output of the ciphertext gets changed significantly(wikipedia, 2022. Avalance effect.https://en.wikipedia.org/wiki/Avalanche_effect)

Confusion and diffusion

See: https://en.wikipedia.org/wiki/Confusion_and_diffusion

Question 3: What is confusion and how is it achieved in AES?

Question 3 answer:

"These two properties make cryptanalysis difficult for cryptanalysts" (The Security Buddy, 2022, What are confusion and diffusion in cryptography?. https://www.youtube.com/watch?v=ojnyzas0HcI).

Confusion is to increase the difficulty in finding the key from a ciphertext. It is achieved by hiding the relationship between the ciphertext and the key (Shannon, C. E. (October 1949). "Communication Theory of Secrecy Systems*". Bell System Technical Journal. 28 (4): 656–715. doi:10.1002/j.1538-7305.1949.tb00928.x). According to Wikipedia "confusion is provided by substitution boxes"(Wikipedia, 2022. "Confusion and diffusion". https://en.wikipedia.org/wiki/Confusion_and_diffusion).

Question 4: What is diffusion and how is it achieved in AES?

Question 4 answer:

According to Wikipedia it is achieved by changiing one bit of the ciphertext and then the other half of the plaintext bits are to change "This is equivalent to the expectation that encryption schemes exhibit an avalanche effect" (Wikipedia, 2022, "Confusion and diffusion". https://en.wikipedia.org/wiki/Confusion_and_diffusion)

Mode of operation

See: https://en.wikipedia.org/wiki/Block_cipher_mode_of_operation#Common_modes

Question 5: You are free to apply any mode of operation described in the article above. ECB is the easiest to implement, but it has some weaknesses. What are these weaknesses?

Question 5 answer:

It lacks "diffusion" and according to Wikipedia is not recommended for use in cryptographic protocols.

Question 6: A stronger mode from a security point-of-view is CBC. It is not much harder to implement but much stronger. What are the advantages and disadvantages of CBC compared to ECB?

Question 6 answer:

According to Wikipedia "Its main drawbacks are that encryption is sequential (i.e., it cannot be parallelized), and that the message must be padded to a multiple of the cipher block size" and according to Educative.io it is not tolerant of block losses.(Educative.io, 2022, "Advantages and disadvantages of using the CBC mode". https://www.educative.io/answers/what-is-cbc).

The advantage of CBC in the initialization vector it adds random factor to each block "which means same blocks in different positions will have different ciphers "(Educative.io, 2022, "Advantages and disadvantages of using the CBC mode". https://www.educative.io/answers/what-is-cbc).

Question 7: What mode of operation are you using? Describe how it works.

Question 7 answer:

The Advanced Encryption Standard (AES-256). It encrypts and decrypts data with a key of 128, 192, 256 and 128 repectivly. To chipher the data:

1. The data gets copied into state.
2. The key which has the size of 64 bytes are expanded into 256 bytes.
3. A round key is added to the State by using XOR for the 0th round.
4. Later a loop is created to replace elements in state with sBox elements, then the elements gets shifted to the left, then the columns gets mixed and lastly a round key is added. This loop runs from the 1st to 13th round of the round key.
5. the 14th round the last round, the elements in state once more gets replaced by sbox elements, which later gets shifted to the left and at last a round key is added.

The decyrption is quite similar, it is the cipher tranformation however in reverse order, it start at round 14th and then loop round 13th-1th then 0-th round is applied. However instead of replacing elements with sbox elements it is replaced by sbox inverse elements, and instead of shifting elements to the left they are shifted to the right (FIPS 197, 2001, "Announcing the advanced encryption standard (AES)". FIPS PUB 197).

Performance

Question 8: How many blocks does the file tWotW.txt consist of?

Question 8 answer: 21501 blocks.

Question 9: How long does it take to read the file, encrypt it and store (write to disk) the encrypted file? You can import and use the time class in Python for this. Use any key you wish.

Question 9 answer: To read the book > encrypt > save: 31.042130947113037s

Question 10: How long does it take to read the encrypted file, decrypt it and store the plain text file?

Question 10 answer: To read the encrypted file > decrypt it > save: 31.462691068649292s

Question 11: Do the encryption and decryption times differ? If so, why?

Question 11 answer: Yes they differ, 1st mainly of my code 2nd becaue of the method to cipher and decipher it, however the difference is not much it is only about 00.04s difference.

Question 12: Give suggestions regarding how you could improve the performance of your code.

Question 12 answer: I could have made the readBook function better without needing to add 13 more elements to the last list, but I ran out of time and had to make sure that the list could be compatible with the codes and functions to encrypt/decrypt.

As question 13 mentions, I could have compressed the data that I would encrypt to improve the time of encryption.

I also noticed that with the tWotW.txt being in a zip file I would not need to use ord() to transform the letters into integers.

Question 13: Try to compress (.zip or other compression) both the plain text file and the encrypted file. Are there any differences in the size of these files? If so why? Is it better to compress first and encrypt later or encrypt first and compress later? Motivate your answer!

Question 13 answer:

The compressed file sizes are smaller compared to the plaintext, as I tried to read the the tWotW.txt file from ZIP it made me relaize that it is much faster to read and the contents in the txt files are all translated to integers which could have made my code more efficient if I had gone with the approach of compressing it before encrypting it. As Crypto book stated "If the text message has been compressed before encryption, then recognition is more difficult. And if the message is some more general type of data, such as a numerical file, and this has been compressed, the problem becomes even more difficult to automate."(Stallings, 2019, Cryptography and Network Security Principles and Pratice 8th edition ,"BRUTE-FORCE ATTACK" section, page 89.)

Notes:

During your presentation we will look at your code together and we will ask questions regarding your implementation. You must be able to explain how all parts of your code works. Feel free to add comments to the code and text cells to help your explanation.

If you get help from others you **MUST** comment this in your code. You should comment if you have discussed your code or collaborated with fellow students and comment this for every method and give credit to the right person. The same goes if you have looked at code from the internet. Provide links.

Upload the encrypted version encryptedtWOW.txt of the original file tWOW.txt to Learn, as well. Make sure that the decrypted file is identical to the original file tWOW.txt.