Homework 3 - Introduction to Ciphers

Kyle Killion

Collaborators - Randall Lisbona and Vinh Le

References -

- https://inventwithpython.com/chapter14.html
- https://en.wikipedia.org/wiki/Symmetric-key_algorithm
- https://github.com/pakesson/diy-ecb-penguin

Exercise 1 - Simple Substitution Cipher

In [2]:
```
alpha = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
ciphertext =  'DEFGHIJKLMNOPQRSTUVWXYZABC'

encodedPhrase = ''
phrase = 'Mayday! Mayday!'
for c in phrase:
    encodedPhrase = encodedPhrase + c
    print(encodedPhrase)
```

```
M
Ma
May
Mayd
Mayda
Mayday
Mayday!
Mayday!
Mayday! M
Mayday! Ma
Mayday! May
Mayday! Mayd
Mayday! Mayda
Mayday! Mayday
Mayday! Mayday!
```

1) Loops through all the characters in the phrase

In [419]:
```
aList = []
for c in phrase:
    if str.isalpha(c) == True:
        aList.append('x')
    else:
        aList.append(c)

print(''.join(aList))
```

```
xxxxxx! xxxxxx!
```

2) Alphabetic Characters Only

In [3]:
```
# Define a function that takes in the users phrase that he/she wants to encrypt/decrypt by shifting

def cry(phrase=None, shiftValue=None, decrypt=False):


    if phrase == None:
        phrase = input('What phrase do you want to encrypt? \n')

        cryptPhrase = ''

        if decrypt == True:
                shiftValue = -shiftValue
                print('Decrytping....')

        for char in phrase:

            if char.isalpha():
                digit = ord(char)

                if (digit > 64) and (digit < 91):
                    digit = chr(((digit - 64) + shiftValue) % 26 + 64)

                elif (digit > 96) and (digit < 123):
                    digit = chr(((digit - 96) + shiftValue) % 26 + 96)

                cryptPhrase += digit

            elif char.isdigit():
                digit = (int(char) + shiftValue) % 6

                cryptPhrase += digit

            else:
                cryptPhrase += char

        print('Message: ', cryptPhrase)

        return cryptPhrase, shiftValue

    else:

        cryptPhrase = ''

        if decrypt == True:
                shiftValue = -shiftValue
                print('Decrytping....')

        for char in phrase:

            if char.isalpha():
                digit = ord(char)

                if (digit > 64) and (digit < 91):
                    digit = chr(((digit - 64) + shiftValue) % 26 + 64)

                elif (digit > 96) and (digit < 123):
                    digit = chr(((digit - 96) + shiftValue) % 26 + 96)

                cryptPhrase += digit

            elif char.isdigit():
                digit = (int(char) + shiftValue) % 6

                cryptPhrase += digit

            else:
                cryptPhrase += char

        print('Message: ', cryptPhrase)

        return cryptPhrase, shiftValue


mess1 = cry('Mayday! Mayday!', shiftValue=4)
```

```
Message:  Qechec! Qechec!
```

3) Shift Sequence

Exercise 2 - Breaking a Simple Substitution Cipher

```
from nltk.corpus import words
import re

def bruteEng(mess=None):

    listWords = []
    theShift = []

    for key in range(1, 27):
        trial = cry(phrase= mess, shiftValue=key, decrypt=True)
        listWords.append(trial[0])
        theShift.append(trial[1])
        print('\n')

        try:
            count=0
            for word in listWords:
                word = word.split()[0].lower()
                word = re.split(r'[(|!]+', word)


                if word[0] in words.words():
                    print('================  English Phrase Located  ====================')
                    return listWords[count], theShift[count]

                else:
                    count += 1



        except Exception as e:
            print('oops', str(e))


bruteEng(mess='Ger csy avmxi gsa mr xlmvxiir pixxivw ? - Wii S Hsyfpi Csy')
```

```
Decrytping....
Message:  Fdq brx `ulwh fr` lq wkluwhhq ohwwhuv ? - Vhh R Grxeoh Brx


Decrytping....
Message:  Ecp aqw ytkvg eqy kp vjktvggp ngvvgtu ? - Ugg Q Fqwdng Aqw


Decrytping....
Message:  Dbo `pv xsjuf dpx jo uijsuffo mfuufst ? - Tff P Epvcmf @pv


Decrytping....
Message:  Can you write cow in thirteen letters ? - See O Double You


================  English Phrase Located  ====================

('Can you write cow in thirteen letters ? - See O Double You', -4)
```

```
this = open('ClassWall.txt')
with open('ClassWall.txt') as f:
    content = f.readlines()
content = [x.strip() for x in content]

for i in range(0,len(content),1):
    bruteEng(content[i])
```

```
Decrytping....
Message:  Mrgj `noy iuaxyk ayky Ve`nut


Decrytping....
Message:  Lqfi ymnx ht`wxj `xjx Udymts


Decrytping....
Message:  Kpeh xlmw gsyvwi ywiw Tcxlsr


Decrytping....
Message:  Jodg wklv frxuvh xvhv Sbwkrq


Decrytping....
Message:  Incf vjku eqwtug wugu Ravjqp


Decrytping....
Message:  Hmbe uijt dpvstf vtft Q`uipo


Decrytping....
Message:  Glad this course uses Python


===============  English Phrase Located  ====================
```

Exercise 3 - AES

Performance wise, the PNG format took the longest to conduct encryption over the chosen Pic

```
from PIL import Image
import time
import sys
import os
import timeit
from cryptography.hazmat.backends import default_backend
from cryptography.hazmat.primitives.ciphers import (Cipher, algorithms, modes)


"""
This is all a compilation of examples found in SO, Module Documentations, and collaboration. I went
ahead and created a class called 'fortress' to begin encompassing the AES Cipher KungFu

"""

class fortress():

    def __init__(self):

        # Initialize the sizes used for IV, Nonce, and Block
        iv_size = 16
        block_size = 16
        nonce_size = 16

        # Key must be one of 16/24/32 bytes in length.
        key = b"9876543210ZYXWVU"  # This is the biggy here ... need to keep this abreast (HARDCODED, weaksauce)

        # Randomize my IV and Nonce here upon Initialization
        iv = os.urandom(iv_size)
        nonce = os.urandom(nonce_size)
```

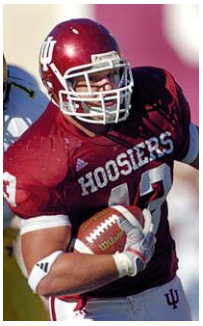
```
def check_args(self):

    """Function to implement to validate your stuff is setup correctly"""

    # Try Block to test modes and sizes are correct and you have file with paths
    try:
        if (len(sys.argv) != 4):
            print(sys.argv,len(sys.argv))
            raise Exception()

        elif (not os.path.isfile(sys.argv[1])):
            raise Exception("Input file must exist")

        elif (not sys.argv[3] in ['CBC', 'ECB','CTR']):
            raise Exception("Block cipher mode should be ECB, CBC, or CTR")

        return (sys.argv[1], sys.argv[2], sys.argv[3])

    except Exception as ex:
        print ("Usage:", sys.argv[0], \
            "full_path_to_input_image full_path_to_output_image ECB|CBC")
        if len(ex.args) > 0:
            print ("--" + str(ex))
        exit


def encryptFile(self, input_filename, output_filename, cipher_mode):
    """This encrypts your file and writes the encrypted file to the specified output"""

    #Get the located file and size
    input_image = Image.open(input_filename)
    filesize = os.path.getsize(input_filename)

    #Bring the image into type bytes
    image_string = input_image.tobytes()

    """
    Here we pad the image length to fill up unused portions of the data structure
    and encode with ascii
    http://www.asciitable.com/index/asciifull.gif to view an ascii table
    """

    image_padding_length = block_size - len(image_string) % block_size
    image_string += image_padding_length * "\n".encode('ascii')
```

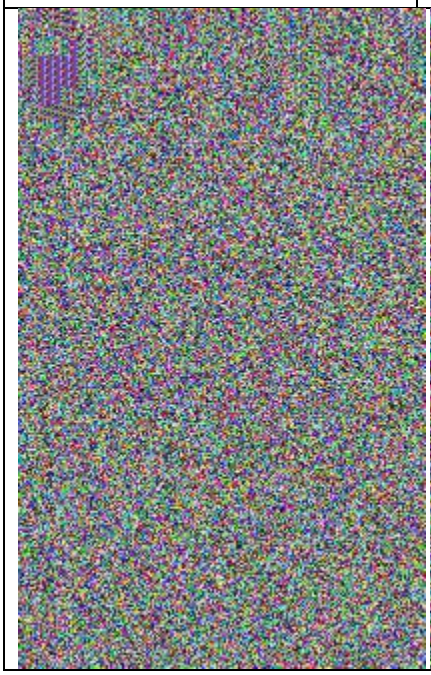
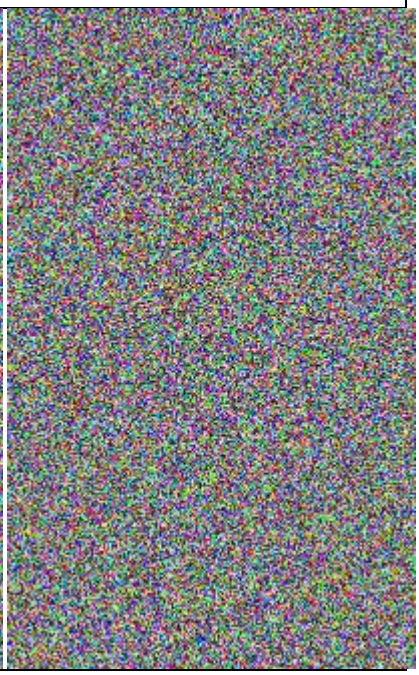
```
# Create the IF block to filter through modes and execute AES cipher algo (CBC, ECB, CTR)
backend = default_backend()
if cipher_mode == 'CBC':
    cipher = Cipher(algorithms.AES(key), modes.CBC(iv), backend=backend)

    """
    Under CBC mode, here we are Cipher Block Chaining and keystreaming from the previous
    ciphertext. On the initial block to encrypt PT, CBC utilizes the IV (Initialization Vector)
    to XOR PT and avalanches from there with the secret key.
    """

elif cipher_mode == 'ECB':
    cipher = Cipher(algorithms.AES(key), modes.ECB(), backend=backend)


    """
    ECB - Electronic Code Book utilizes the same secret key each time to encrypt. Depending on
    the situation, more often than not, you will want to use another mode to encrypt given that
    this mode can be repetitive and highly pattern forming. My image here had some minute patternistic
    sections that didn't resemble regular white noise-ish looking encryption.
    """

elif cipher_mode == 'CTR':
    cipher = Cipher(algorithms.AES(key), modes.CTR(nonce), backend=backend)

    """
    CTR - otherwise known as the In Counter mode, can operate under Stream Cipher architecture.
    Cool thing is, you can also decrypt at the same time (in parallel). This takes a Nonce (number one time - RANDOM)
    and is encrypted with the secret key. Once this is done, the encrypted Nonce is XOR'd with the PT. This
    is just another approach somewhat similar to IV in CBC mode. However, the Nonce is then incremented
    by 1 each time for the next subsequent encryption.
    """


#Print out File Description and encryption process
print("\n",input_filename,' ',filesize," bytes. Encrypted using AES in ",
     cipher_mode, " mode to \"", cipher_mode,'_',output_filename,"\"", sep ='' )

# Start the clock for reference
start_time = time.perf_counter()

# generate the encrypted image string
encryptor = cipher.encryptor()
encrypted = encryptor.update(image_string) + encryptor.finalize()
encryption_time = (time.perf_counter() - start_time)* 1000
bytes_per_msec = filesize/encryption_time
print('{:.4f} ms to encrypt = {:,.0f} bytes/msec'.format(encryption_time,bytes_per_msec))
```

```
        # create an image from the encrypted string, using the same mode (probably RGB)
        # and dimensions in pixels of original image.
        start_time = time.perf_counter()
        encrypted_img = Image.frombuffer(input_image.mode, input_image.size,
                                         encrypted, 'raw',input_image.mode, 0, 1)
        image_create_time = (time.perf_counter() - start_time)* 1000
        print('{:.4f} ms to create image'.format(image_create_time))


        # Finalize and Save encrypted files to output paths
        start_time = time.perf_counter()
        encrypted_img.save(cipher_mode+'_'+output_filename) #prefixed with the ciphermode used
        file_write_time = (time.perf_counter() - start_time)* 1000
        outname = cipher_mode + '_' + output_filename
        outfilesize = os.path.getsize(outname)
        print('{:.4f} ms to write {:,} byte image file'.format(file_write_time,outfilesize))


if __name__ == '__main__':

    # Initialize the class and check status (we init a lot of this in the class)
    letsDoThis = fortress()


    # Line up our list of info because I hate writting more lines than I have to
    listModes = ['ECB','CBC','CTR']
    listFileNames = ['KillionScoopScore','PicSix']
    formats = ['.bmp','.jpg','.png']

    # Loop through the number of formats and conduct each CipherMode for each format
    for mat in formats:
        print('\n')
        args = letsDoThis.check_args()

        for mode in listModes:
            file_in = listFileNames[0] + mat
            file_out = listFileNames[1] + mat

            %time letsDoThis.encryptFile(input_filename=file_in, output_filename=file_out, cipher_mode=mode)
```

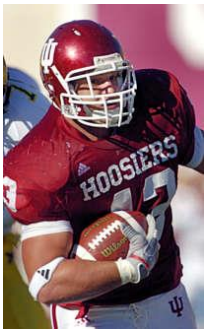
```
['C:\\Anaconda3\\Lib\\site-packages\\ipykernel\\__main__.py', '-f', 'C:\\Users\\hb13316\\AppData\\Roaming\\jupyter\\runtime\\ker
nel-0e02ae6d-4441-4e3d-a0cd-fb47c61aa640.json'] 3
Usage: C:\Anaconda3\Lib\site-packages\ipykernel\__main__.py full_path_to_input_image full_path_to_output_image ECB|CBC

KillionScoopScore.bmp 202014 bytes. Encrypted using AES in ECB mode to "ECB_PicSix.bmp"
0.3603 ms to encrypt = 560,657 bytes/msec
0.2664 ms to create image
3.7102 ms to write 202,014 byte image file
Wall time: 15.6 ms

KillionScoopScore.bmp 202014 bytes. Encrypted using AES in CBC mode to "CBC_PicSix.bmp"
0.6606 ms to encrypt = 305,813 bytes/msec
0.3299 ms to create image
3.1066 ms to write 202,014 byte image file
Wall time: 0 ns

KillionScoopScore.bmp 202014 bytes. Encrypted using AES in CTR mode to "CTR_PicSix.bmp"
0.5986 ms to encrypt = 337,476 bytes/msec
0.2926 ms to create image
5.0645 ms to write 202,014 byte image file
Wall time: 15.6 ms


['C:\\Anaconda3\\Lib\\site-packages\\ipykernel\\__main__.py', '-f', 'C:\\Users\\hb13316\\AppData\\Roaming\\jupyter\\runtime\\ker
nel-0e02ae6d-4441-4e3d-a0cd-fb47c61aa640.json'] 3
Usage: C:\Anaconda3\Lib\site-packages\ipykernel\__main__.py full_path_to_input_image full_path_to_output_image ECB|CBC

KillionScoopScore.jpg 15393 bytes. Encrypted using AES in ECB mode to "ECB_PicSix.jpg"
0.4227 ms to encrypt = 36,418 bytes/msec
0.3141 ms to create image
8.9398 ms to write 42,217 byte image file
Wall time: 23.6 ms

KillionScoopScore.jpg 15393 bytes. Encrypted using AES in CBC mode to "CBC_PicSix.jpg"
0.5301 ms to encrypt = 29,039 bytes/msec
0.2764 ms to create image
9.0630 ms to write 42,401 byte image file
Wall time: 9 ms

KillionScoopScore.jpg 15393 bytes. Encrypted using AES in CTR mode to "CTR_PicSix.jpg"
0.5547 ms to encrypt = 27,749 bytes/msec
0.3126 ms to create image
11.1868 ms to write 42,486 byte image file
Wall time: 15.6 ms


['C:\\Anaconda3\\Lib\\site-packages\\ipykernel\\__main__.py', '-f', 'C:\\Users\\hb13316\\AppData\\Roaming\\jupyter\\runtime\\ker
nel-0e02ae6d-4441-4e3d-a0cd-fb47c61aa640.json'] 3
Usage: C:\Anaconda3\Lib\site-packages\ipykernel\__main__.py full_path_to_input_image full_path_to_output_image ECB|CBC

KillionScoopScore.png 173661 bytes. Encrypted using AES in ECB mode to "ECB_PicSix.png"
0.4431 ms to encrypt = 391,939 bytes/msec
0.0720 ms to create image
34.2836 ms to write 242,683 byte image file
Wall time: 46.8 ms

KillionScoopScore.png 173661 bytes. Encrypted using AES in CBC mode to "CBC_PicSix.png"
0.6521 ms to encrypt = 266,306 bytes/msec
0.0681 ms to create image
38.9977 ms to write 270,011 byte image file
Wall time: 46.8 ms

KillionScoopScore.png 173661 bytes. Encrypted using AES in CTR mode to "CTR_PicSix.png"
0.5035 ms to encrypt = 344,894 bytes/msec
0.1594 ms to create image
78.0582 ms to write 270,012 byte image file
Wall time: 101 ms
```

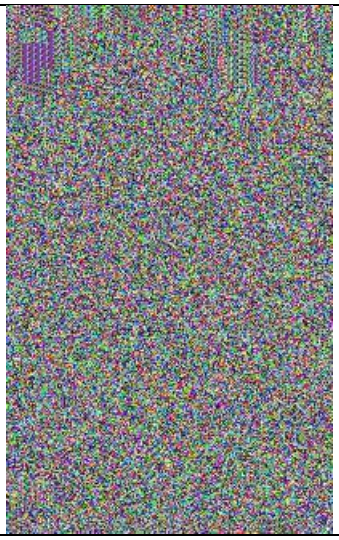
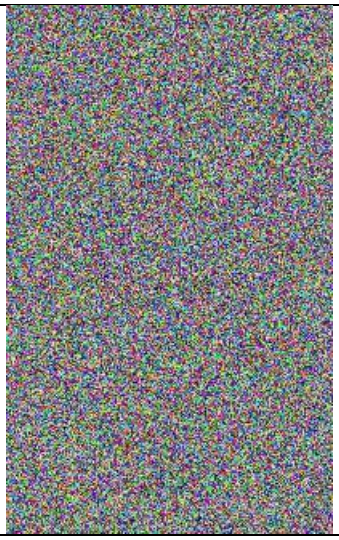
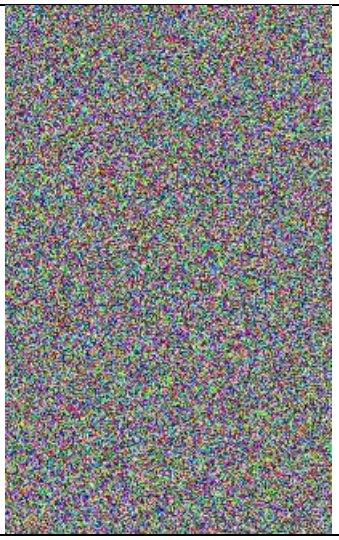
BMP Format



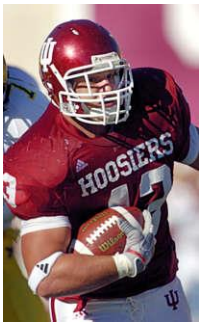
ECB	CBC	CTR
		

JPG Format



ECB	CBC	CTR
		

PNG Format



ECB	CBC	CTR