Shri Vile Parle Kelavani Mandal's
DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING
(Autonomous College Affiliated to the University of Mumbai)
NAAC Accredited with "A" Grade (CGPA : 3.18)

Experiment 1

Date of Performance : **Date of Submission:**

SAP Id: 60004190057 **Name : Junaid Altaf Girkar**

Div: A **Batch : A4**

Aim of Experiment

Design and Implement Encryption and Decryption Algorithm for

a. Caesar cipher cryptographic algorithm by considering letter [A..Z] and digits [0..9]. Create two functions Encrypt() and Decrypt(). Apply Brute Force Attack to reveal secret. Create Function BruteForce(). Demonstrate the use of these functions on any paragraph.
b. Affine Cipher. Your Program Must Input Image in Gray Scale. Choose keys according to Gray Scale Intensity level. Create two functions Encrypt() and Decrypt(). Make sure to have Multiplicative Inverse Exists for one of the Key in selected Key pair of Affine Cipher. (CO1)

Theory / Algorithm / Conceptual Description

CAESAR CIPHER

The Caesar Cipher technique is one of the earliest and simplest method of encryption technique. It's simply a type of substitution cipher, i.e., each letter of a given text is replaced by a letter some fixed number of positions down the alphabet. For example with a shift of 1, A would be replaced by B, B would become C, and so on.

Thus to cipher a given text we need an integer value, known as shift which indicates the number of position each letter of the text has been moved down.

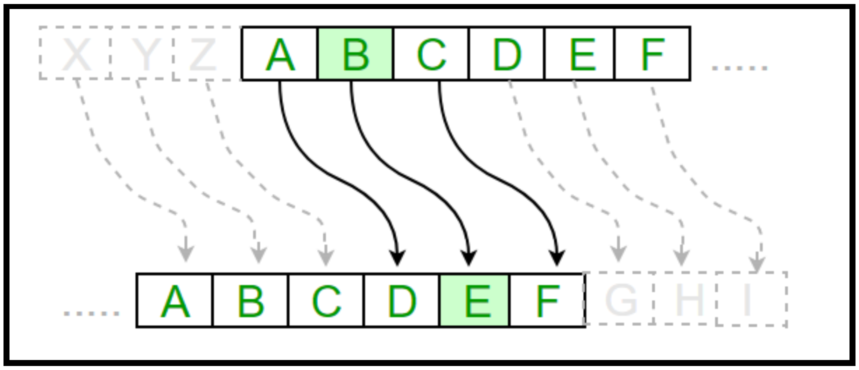
The encryption can be represented using modular arithmetic by first transforming the letters into numbers, according to the scheme, A = 0, B = 1,…, Z = 25. Encryption of a letter by a shift n can be described mathematically as.

```
E_n(x)=(x+n)mod\ 26
```

(Encryption Phase with shift n)

```
D_n(x)=(x-n)mod\ 26
```

(Decryption Phase with shift n)



HILL CIPHER

Hill cipher is a polygraphic substitution cipher based on linear algebra.Each letter is represented by a number modulo 26. Often the simple scheme A = 0, B = 1, …, Z = 25 is used, but this is not an essential feature of the cipher. To encrypt a message, each block of n letters (considered as an n-component vector) is multiplied by an invertible n × n matrix, against modulus 26. To decrypt the message, each block is multiplied by the inverse of the matrix used for encryption.

The matrix used for encryption is the cipher key, and it should be chosen randomly from the set of invertible n × n matrices

Encryption

$$\begin{bmatrix} 6 & 24 & 1 \\ 13 & 16 & 10 \\ 20 & 17 & 15 \end{bmatrix} \begin{bmatrix} 0 \\ 2 \\ 19 \end{bmatrix} = \begin{bmatrix} 67 \\ 222 \\ 319 \end{bmatrix} \equiv \begin{bmatrix} 15 \\ 14 \\ 7 \end{bmatrix} \pmod{26}$$

Decryption

$$\begin{bmatrix} 6 & 24 & 1 \\ 13 & 16 & 10 \\ 20 & 17 & 15 \end{bmatrix}^{-1} \equiv \begin{bmatrix} 8 & 5 & 10 \\ 21 & 8 & 21 \\ 21 & 12 & 8 \end{bmatrix} \pmod{26}$$

Program

A)

```
# Encryption and decryption of a message using a caesar cipher

def encrypt(message, key):

    encrypted_message = ""
    for letter in message:
        if letter.isupper():
            encrypted_message += chr((ord(letter) + key - 64) % 26 +
65)
        else:
            encrypted_message += chr((ord(letter) + key - 96) % 26 +
97)
    return encrypted_message

plain_text = "UkraineIsACountryInEasternEurope"
key = 5

print("PLain text: ", plain_text)
print("Key: ", key)
print("Cipher Text : " + encrypt(plain_text, key))

def decrypt(cipher_text, key):
    decrypted_message = ""
    for letter in cipher_text:
        if letter.isupper():
            decrypted_message += chr((ord(letter) + key - 65) % 26 +
65)
        else:
```

```
            decrypted_message += chr((ord(letter) + key - 97) % 26 +
97)
    return decrypted_message

def brute_force_decrypt(cipher_text):
    for i in range(26):
        print("Key: ", abs(25 - i))
        print("Decrypted Text: " + decrypt(cipher_text, i))

brute_force_decrypt(encrypt(plain_text, key))
```

Output

```
PLain text:  UkraineIsACountryInEasternEurope
Key:  5
Cipher Text : AqxgotkOyGIuatzxeOtKgyzkxtKaxuvk
Key:  25
Decrypted Text: AqxgotkOyGIuatzxeOtKgyzkxtKaxuvk
Key:  24
Decrypted Text: BryhpulPzHJvbuayfPuLhzalyuLbyvwl
Key:  23
Decrypted Text: CsziqvmQaIKwcvbzgQvMiabmzvMczwxm
Key:  22
Decrypted Text: DtajrwnRbJLxdwcahRwNjbcnawNdaxyn
Key:  21
Decrypted Text: EubksxoScKMyexdbiSxOkcdobxOebyzo
Key:  20
Decrypted Text: FvcltypTdLNzfyecjTyPldepcyPfczap
Key:  19
Decrypted Text: GwdmuzqUeMOagzfdkUzQmefqdzQgdabq
Key:  18
Decrypted Text: HxenvarVfNPbhagelVaRnfgreaRhebcr
Key:  17
Decrypted Text: IyfowbsWgOQcibhfmWbSoghsfbSifcds
Key:  16
Decrypted Text: JzgpxctXhPRdjcignXcTphitgcTjgdet
Key:  15
Decrypted Text: KahqyduYiQSekdjhoYdUqijuhdUkhefu
Key:  14
Decrypted Text: LbirzevZjRTflekipZeVrjkvieVlifgv
Key:  13
```

```
Decrypted Text: McjsafwAkSUgmfljqAfWsklwjfWmjghw
Key:  12
Decrypted Text: NdktbgxBlTVhngmkrBgXtlmxkgXnkhix
Key:  11
Decrypted Text: OeluchyCmUWiohnlsChYumnylhYolijy
Key:  10
Decrypted Text: PfmvdizDnVXjpiomtDiZvnozmiZpmjkz
Key:  9
Decrypted Text: QgnwejaEoWYkqjpnuEjAwopanjAqnkla
Key:  8
Decrypted Text: RhoxfkbFpXZlrkqovFkBxpqbokBrolmb
Key:  7
Decrypted Text: SipyglcGqYAmslrpwGlCyqrcplCspmnc
Key:  6
Decrypted Text: TjqzhmdHrZBntmsqxHmDzrsdqmDtqnod
Key:  5
Decrypted Text: UkraineIsACountryInEasternEurope
Key:  4
Decrypted Text: VlsbjofJtBDpvouszJoFbtufsoFvspqf
Key:  3
Decrypted Text: WmtckpgKuCEqwpvtaKpGcuvgtpGwtqrg
Key:  2
Decrypted Text: XnudlqhLvDFrxqwubLqHdvwhuqHxursh
Key:  1
Decrypted Text: YovemriMwEGsyrxvcMrIewxivrIyvsti
Key:  0
Decrypted Text: ZpwfnsjNxFHtzsywdNsJfxyjwsJzwtuj
```

Program

B)

```
 from scipy import misc
import imageio
import numpy as np
import matplotlib.pyplot as plt
import os.path
import pickle
from numpy.linalg import inv, det
import sys
import scipy.misc
```

```python
#   IMAGE SECTION
def read_image(image_path):
    """ Read an image and return a one hot vector of the image"""
    img = imageio.imread(image_path)
    reshape_value = 1

    for i in img.shape:
        reshape_value *= i

    return img.reshape((1, reshape_value)), img.shape


def show_image(image):
    """ Show a single image"""
    plt.imshow(image)
    plt.show()


def show_images(a, b):
    """ Show two images side by side"""
    plot_image = np.concatenate((a, b), axis=1)
    plt.imshow(plot_image)
    plt.show()

# HILL CLIMB SECTION

class HillClimb:
    def __init__(self, data, file_name, key_path=None):

        self.data = data

        # Computet the chunk
        self.chunk = self.computer_chunk()

        if key_path:
            # Load the key if it exist in the current dir
            self._key = pickle.load(open( key_path, "rb" ))
            print('Usigng the args -k ' + key_path)
        else:
```

```
            file_name = file_name + '.key'

            if os.path.isfile(file_name):
                # Load the key if it exist in the current dir
                self._key = pickle.load(open( file_name, "rb" ))
                print('Usigng the ' + file_name)
            else:
                # Generate a random key
                self._key = np.random.random_integers(0, 100,
(self.chunk, self.chunk))

                # If determinat is equal to zero regenrate another
key
                if det(self._key) == 0:
                    self._key = np.random.random_integers(0, 100,
(self.chunk, self.chunk))

                # Save the key in a pickle
                pickle.dump( self._key, open( file_name, "wb" ) )

        print(self._key.dtype)
        print(self._key.shape)
        print(self._key)

        # Get the inverse of the key
        self.reversed_key = np.matrix(self._key).I.A

        print(self.reversed_key.dtype)
        print(self.reversed_key.shape)
        print(self.reversed_key)

    def computer_chunk(self):
        max_chunk = 100
        data_shape = self.data.shape[1]
        print(data_shape)

        for i in range(max_chunk, 0, -1):
            if data_shape % i == 0:
                return i
```

```python
@property
def key(self):
    return self._key

def encode(self, data):
    """ Encode function """
    crypted = []
    chunk = self.chunk
    key = self._key

    for i in range(0, len(data), chunk):

        temp = list(np.dot(key, data[i:i + chunk]))
        crypted.append(temp)

    crypted = (np.array(crypted)).reshape((1, len(data)))
    return crypted[0]


def decode(self, data):
    """ Decode function """
    uncrypted = []
    chunk = self.chunk
    reversed_key = self.reversed_key

    for i in range(0, len(data), chunk):
        temp = list(np.dot(reversed_key, data[i:i + chunk]))
        uncrypted.append(temp)

    uncrypted = (np.array(uncrypted)).reshape((1, len(data)))

    return uncrypted[0]
```

```python
import pickle
from numpy.linalg import inv, det
import sys
import scipy.misc
from HillClimb import HillClimb
from HillClimb import *
import imageio

def transform(np_array, shape):
    return np_array.reshape(shape).astype('uint8')


if __name__ == '__main__':
    if len(sys.argv) > 1:
        image_file_name = sys.argv[1]
    else:
        raise Exception('Missing image file name')


    img, original_shape = read_image(image_file_name)
    hill = HillClimb(data=img, file_name=image_file_name)

    ### Testing zone
    print(img.shape)


    # ------------------------- Encoding -------------------------

    # Get the encdoed vector image
    encoded_image_vector = hill.encode(img[0])

    # Reshape to the original shape of the image
    encoded_image = encoded_image_vector.reshape(original_shape)

    # Show the decoded image
    # show_image(encoded_image.astype('uint8'))

    # Setup the encdoed file name to be used when saving the encdoed
image
```

```
    img_name = image_file_name.split('.')[0]
    img_extension = image_file_name.split('.')[1]
    encoded_img_name = '{0}-encoded.{1}'.format(img_name,
img_extension)


     # Convert to uint8
    encoded_image = encoded_image.astype('uint8')

    # Save the image
    imageio.imsave(encoded_img_name, encoded_image)

    # Save the image as a pickle model
    pickle.dump(encoded_image_vector, open( encoded_img_name + '.pk',
"wb" ))


    # # ------------------------- Decoding -------------------------


    img_vector = pickle.load(open(encoded_img_name + '.pk', 'rb'))

    # Get the decoded vector image
    decoded_image_vector = hill.decode(img_vector)

    # Reshape to the original shape of the image
    decoded_image = decoded_image_vector.reshape(original_shape)

    decoded_img_name = '{0}-decoded.{1}'.format(img_name,
img_extension)

    # Save the image
    imageio.imsave(decoded_img_name, decoded_image)
```

Output

```
 722775
Usigng the Hello.jpg.key
int32
(75, 75)
[[  2  91  29 ...  12  27  46]
 [  8  57  87 ...  75  84  63]
 [ 50  18  84 ...   2  25  86]
 ...
 [ 77  91  41 ... 100  75  22]
 [ 17  54  53 ...  80  69  62]
 [ 91  66  16 ...  35  28  83]]
float64
(75, 75)
[[-0.00260477  0.00198561  0.00334513 ...  0.00091603  0.00228508
   0.00486007]
 [ 0.00694458 -0.00346724  0.00111209 ...  0.00046972 -0.00580259
  -0.00113174]
 [ 0.01007937 -0.01406634 -0.00100486 ... -0.00049209 -0.01050542
   0.0045447 ]
 ...
 [-0.00092501  0.0058364   0.00134957 ...  0.00548127  0.00062253
   0.00192339]
 [-0.00492509  0.00583449 -0.00210589 ... -0.00322735  0.00416567
  -0.00583711]
 [-0.00516961  0.0075935   0.00729953 ... -0.00127429  0.00345992
  -0.0035384 ]]
(1, 722775)
```

CONCLUSION

With the increasing amount of data being generated, it is very important that confidential information does not get leaked and is read by the intended recipient.We learnt about the different encryption techniques and different ciphers. We then wrote a python program which implemented Caesar Cipher and Hill Cipher.