

**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

1. 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.
2. 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.

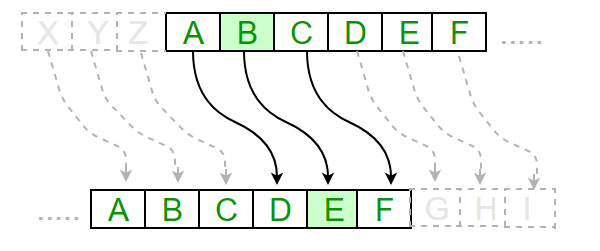
The encryption can be represented using modular arithmetic by first transforming the letters into numbers, according to the scheme, A = 0, B = 1,…, Z = 25. Encryption of a letter by a shift n can be described mathematically as.

| E\_n(x)=(x+n)mod\ 26 |
| --- |

(Encryption Phase with shift n)

| D\_n(x)=(x-n)mod\ 26 |
| --- |

(Decryption Phase with shift n)

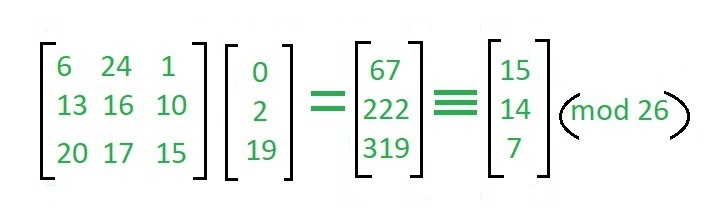
****

**HILL CIPHER**

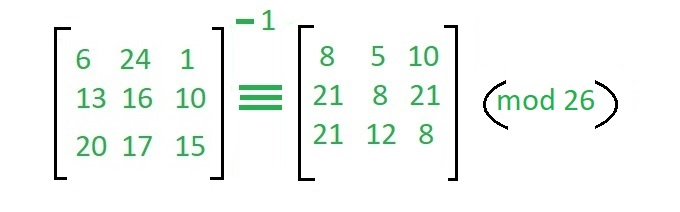
Hill cipher is a polygraphic substitution cipher based on linear algebra.Each letter is represented by a number modulo 26. Often the simple scheme A = 0, B = 1, …, Z = 25 is used, but this is not an essential feature of the cipher. To encrypt a message, each block of n letters (considered as an n-component 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**

****

**Decryption**

****

**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  # 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    @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]      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.