Walchand Institute of Technology, Solapur
Department of Computer Science and Engineering

CERTIFICATE

Certified that Mr/Ms Sushant Magdum Roll No.41 of **BE CSE – 1** class has completed satisfactorily the experiments in the Subject **Information & Cyber Security** during the year **2021-2022**.

Date : 30 /06/2022

Mrs. Rashmi Dixit — Subject-In-Charge

Dr. S A Halkude — Principal

INDEX

ASSIGNMENT : 1

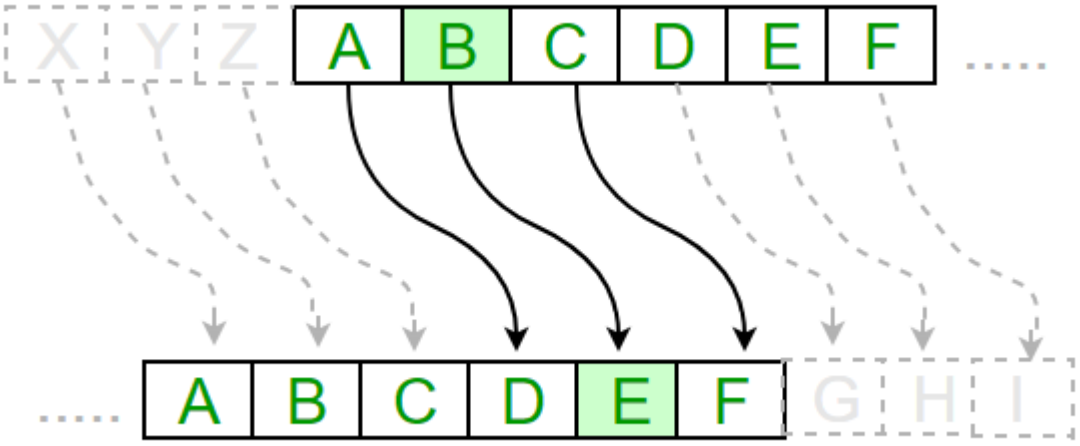
AIM : Caesar Cipher

THEORY :

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. The method is apparently named after Julius Caesar, who apparently used it to communicate with his officials.
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.

(Encryption Phase with shift n) :$E_n(x) = (x+n) \bmod 26$

(Decryption Phase with shift n) :$D_n(x) = (x-n) \bmod 26$



ALGORITHM :

Algorithm for Caesar Cipher:
Input:

1. A String of lower case letters, called Text.
2. An Integer between 0-25 denoting the required shift.

Procedure:

- Traverse the given text one character at a time .
- For each character, transform the given character as per the rule, depending on whether we're encrypting or decrypting the text.
- Return the new string generated.

Program that receives a Text (string) and Shift value(integer) and returns the encrypted text.

PROGRAM :

```
L2I = dict(zip("ABCDEFGHIJKLMNOPQRSTUVWXYZ",range(26)))

I2L = dict(zip(range(26),"ABCDEFGHIJKLMNOPQRSTUVWXYZ"))


#key = 3

#plaintext = "DEFEND THE EAST WALL OF THE CASTLE"

plaintext=input()

key=int(input())

# encipher

ciphertext = ""

for c in plaintext.upper():

   if c.isalpha(): ciphertext += I2L[ (L2I[c] + key)%26 ]

   else: ciphertext += c

# decipher

plaintext2 = ""

for c in ciphertext.upper():

   if c.isalpha(): plaintext2 += I2L[ (L2I[c] - key)%26 ]
```

```
    else: plaintext2 += c

print ("The plaintext is : ",plaintext)

print ("Encryption : ",ciphertext)

print ("Decryption : ",plaintext2)
```

OUTPUT :

```
Walchand Institute of Technology
3
The plaintext is :  Walchand Institute of Technology
Encryption :  ZDOFKDQG LQVWLWXWH RI WHFKQRORJB
Decryption :  WALCHAND INSTITUTE OF TECHNOLOGY
>>>
```

CONCLUSION :

Advantages :

- One of the easiest methods to use in cryptography and can provide minimum security to the information
- Use of only a short key in the entire process
- One of the best methods to use if the system cannot use any complicated coding techniques
- Requires few computing resources

Disadvantages :

- Simple structure usage
- Can only provide minimum security to the information
- Frequency of the letter pattern provides a big clue in deciphering the entire message

ASSIGNMENT : 2

AIM : Monoalphabetic Cipher.

THEORY :

Monoalphabetic cipher is a substitution cipher in which for a given key, the cipher alphabet for each plain alphabet is fixed throughout the encryption process. For example, if 'A' is encrypted as 'D', for any number of occurrence in that plaintext, 'A' will always get encrypted to 'D'.
All of the substitution ciphers we have discussed earlier in this chapter are monoalphabetic;

Monoalphabetic cipher is one where each symbol in plain text is mapped to a fixed symbol in cipher text.

The relationship between a character in the plain text and the characters in the cipher text is one-to-one.
Each alphabetic character of plain text is mapped onto a unique alphabetic character of a cipher text.
A stream cipher is a monoalphabetic cipher if the value of key does not depend on the position of the plain text character in the plain text stream.

It includes additive, multiplicative, affine and monoalphabetic substitution cipher.
It is a simple substitution cipher.
Monoalphabetic Cipher is described as a substitution cipher in which the same fixed mappings from plain text to cipher letters across the entire text are used.
Monoalphabetic ciphers are not that strong as compared to polyalphabetic cipher.

ALGORITHM :

1. Create two char arrays, one for normal alphabets(say normalChar[]) and another is for encoding(say codedChar[]).
2. We will use two functions:
    - **stringEncryption:** We pass string(string with all characters in lower case) as a parameter. Initialize an empty string(say encryptedString). We use for loop and compare each character with normal char array, whenever the condition

is true, add the character with the corresponding index of codedChar to the encrypted string. In the case of special characters, we will add them directly to the string.

- **stringDescryption:** We pass the encrypted string as the parameter. Initialize an empty string(say descryptedString). In the same way, we run the for loop and add the character with the corresponding index of normalChar to the decrypted string. In the case of special characters, we will add them directly to the string.

PROGRAM :

```
import sys, random

LETTERS = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'

def main():

myMessage = input()

myKey = 'QWERTYUIOPASDFGHJKLZXCVBNM'

myMode = input()

checkValidKey(myKey)

   if myMode == 'encrypt':

      translated = encryptMessage(myKey, myMessage)

elifmyMode == 'decrypt':

      translated = decryptMessage(myKey, myMessage)

print('Using key %s' % (myKey))

print('The %sed message is:' % (myMode))

   print(translated)

def checkValidKey(key):

keyList = list(key)

lettersList = list(LETTERS)
```

```
keyList.sort()
lettersList.sort()
    if keyList != lettersList:
sys.exit('There is an error in the key or symbol set.')
def encryptMessage(key, message):
    return translateMessage(key, message, 'encrypt')
def decryptMessage(key, message):
    return translateMessage(key, message, 'decrypt')
def translateMessage(key, message, mode):
    translated = ''
charsA = LETTERS
charsB = key
    if mode == 'decrypt':
        # For decrypting, we can use the same code as encrypting. We
        # just need to swap where the key and LETTERS strings are used.
charsA, charsB = charsB, charsA
    # loop through each symbol in the message
    for symbol in message:
        if symbol.upper() in charsA:
            # encrypt/decrypt the symbol
symIndex = charsA.find(symbol.upper())
            if symbol.isupper():
                translated += charsB[symIndex].upper()
            else:
```

```
            translated += charsB[symIndex].lower()
        else:
            # symbol is not in LETTERS, just add it
            translated += symbol
    return translated
def getRandomKey():
    key = list(LETTERS)
random.shuffle(key)
    return ''.join(key)
main()
```

OUTPUT :

CONCLUSION :

Advantages :

- Brute force cryptanalysis is highly impossible.
- Easy to implement as compared to Playfair cipher.

Disadvantages :

- Complex to implement as compared with Ceasar cipher.
- More time consuming

ASSIGNMENT : 3

AIM : Playfair Cipher.

THEORY :

Playfair cipher is an encryption algorithm to encrypt or encode a message. It is the same as a traditional cipher. The only difference is that it encrypts a **digraph** (a pair of two letters) instead of a single letter.

It initially creates a key-table of 5*5 matrix. The matrix contains alphabets that act as the key for encryption of the plaintext. Note that any alphabet should not be repeated. Another point to note that there are 26 alphabets and we have only 25 blocks to put a letter inside it. Therefore, one letter is excess so, a letter will be omitted (usually J) from the matrix. Nevertheless, the plaintext contains J, then **J** is replaced by **I**. It means treat I and J as the same letter, accordingly.

Since Playfair cipher encrypts the message **digraph by digraph**. Therefore, the Playfair cipher is an example of a **digraph substitution cipher**.

Before moving ahead, let's understand the terminology used in this Playfair cipher.

Terminology :

- **Plaintext:** It is the original message that is to be encrypted. It is also known as a **message**.
- **Ciphertext:** It is an encrypted message.
- **Cipher:** It is an algorithm for transforming plaintext to ciphertext.
- **Key:** It is the key to encrypt or decrypt the plaintext. It is known only to the sender and receiver. It is filled character by character in the matrix that is called **key-**

table or **key-matrix**.

- **Encipher:** The process of converting plaintext into ciphertext is called **encipher**.
- **Decipher:** The process of removing ciphertext from plaintext is called **decipher**.
- **Cryptanalysis:** It is the study of the methods and principles of deciphering ciphertext without knowing the key.

ALGORITHM :

Playfair Cipher Encryption Rules :

1. First, split the plaintext into **digraphs** (pair of two letters). If the plaintext has the odd number of letters, append the letter **Z** at the end of the plaintext. It makes the plaintext of **even** For example, the plaintext **MANGO** has five letters. So, it is not possible to make a digraph. Since, we will append a letter **Z** at the end of the plaintext, i.e. **MANGOZ.**

2. After that, break the plaintext into **digraphs** (pair of two letters). If any letter appears twice (side by side), put **X** at the place of the second occurrence. Suppose, the plaintext is **COMMUNICATE** then its digraph becomes **CO MX MU NI CA TE**. Similarly, the digraph for the plaintext **JAZZ** will be **JA ZX ZX**, and for plaintext **GREET**, the digraph will be **GR EX ET.**

3. To determine the cipher (encryption) text, first, build a 5*5 key-matrix or key-table and filled it with the letters of alphabets, as directed below:

- Fill the first row (left to right) with the letters of the given keyword (**ATHENS**). If the keyword has duplicate letters (if any) avoid them. It means a letter will be considered only once. After that, fill the remaining letters in alphabetical order. Let's create a 5*5 key-matrix for the keyword **ATHENS**.

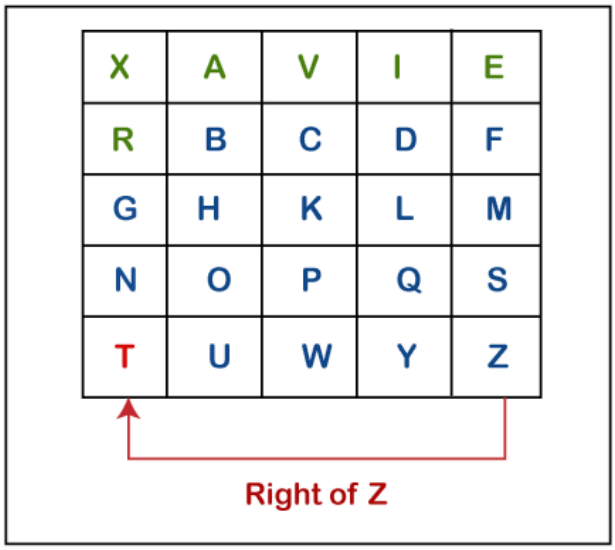
A	T	H	E	N
S	B	C	D	F
G	I/J	K	L	M
O	P	Q	R	U
V	W	X	Y	Z

Note that in the above matrix any letter is not repeated. The letters in the first row (in green color) represent the keyword and the remaining letters sets in alphabetical order.

4. There may be the following three conditions:

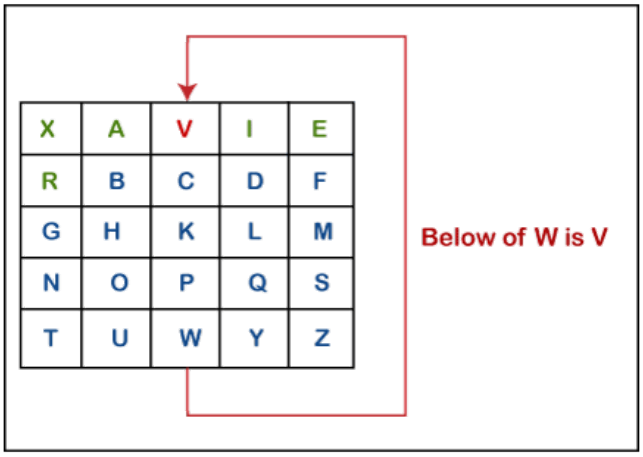
i) If a pair of letters (digraph) appears in the same row

In this case, replace each letter of the digraph with the letters immediately to their right. If there is no letter to the right, consider the first letter of the same row as the right letter. Suppose, **Z** is a letter whose right letter is required, in such case, **T** will be right to Z.



ii) If a pair of letters (digraph) appears in the same column

In this case, replace each letter of the digraph with the letters immediately below them. If there is no letter below, wrap around to the top of the same column. Suppose, **W** is a letter whose below letter is required, in such case, **V** will be below W.



iii) If a pair of letters (digraph) appears in a different row and different column

In this case, select a 3*3 matrix from a 5*5 matrix such that pair of letters appear in the 3*3 matrix. Since they occupy two opposite corners of a square within the matrix. The other corner will be a cipher for the given digraph.

In other words, we can also say that intersection of H and Y will be the cipher for the first letter and

Suppose, a digraph is **HY** and we have to find a cipher for it. We observe that both H and Y are placed in different rows and different columns. In such cases, we have to select a 3*3 matrix in such a way that both H and Y appear in the 3*3 matrix (highlighted with yellow color). Now, we will consider only the selected matrix to find the cipher.

X	A	V	I	E
R	B	C	D	F
G	H	K	L	M
N	O	P	Q	S
T	U	W	Y	Z

Now to find the cipher for HY, we will consider the diagonal **opposite** to HY, i.e. **LU.** Therefore, the cipher for **H** will be **L,** and the cipher for **Y** will be **U**.

X	A	V	I	E
R	B	C	D	F
G	H	K	L	M
N	O	P	Q	S
T	U	W	Y	Z

Playfair Cipher Decryption

The decryption procedure is the same as encryption but the steps are applied in **reverse** order. For decryption cipher is symmetric (move left along rows and up along columns). The receiver of the plain text has the same key and can create the same key-table that is used to decrypt the message.

Let's see an example of Playfair cipher.

Example of Playfair Cipher

Suppose, the plaintext is **COMMUNICATION** and the key that we will use to encipher the plaintext is **COMPUTER**. The key can be any word or phrase. Let's encipher the message **COMMUNICATION**.

1. First, split the plaintext into digraph (by rule 2) i.e. **CO MX MU NI CA TE**.

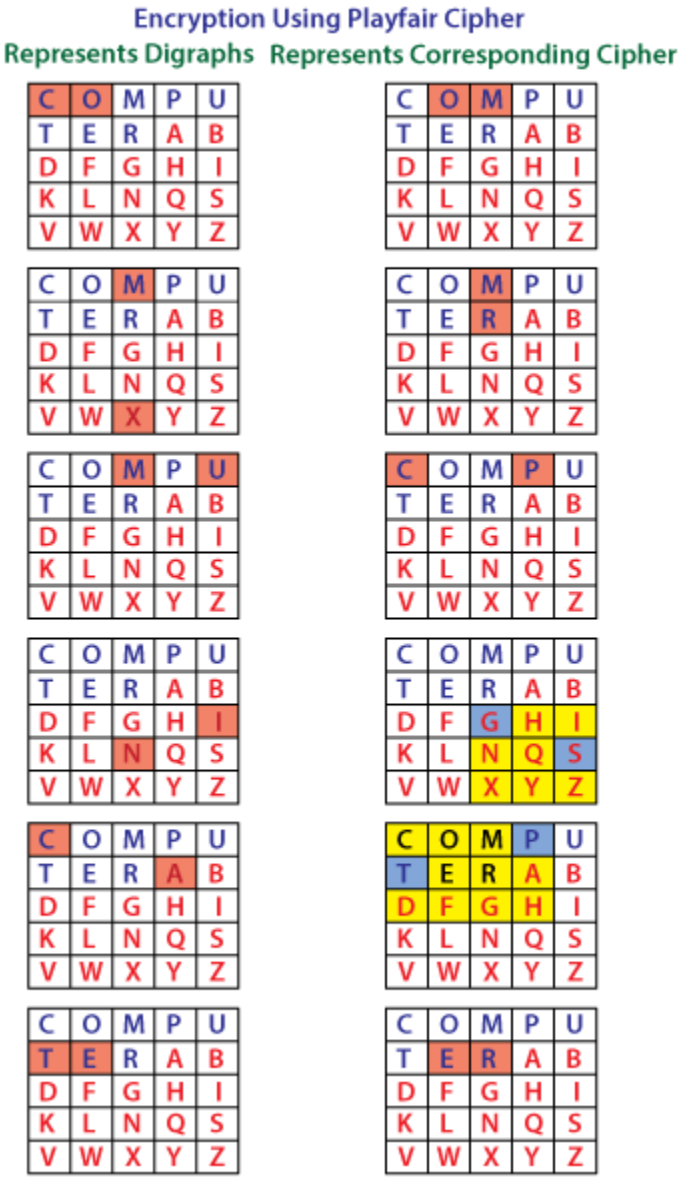
2. Construct a 5*5 key-matrix (by rule 3). In our case, the key is **COMPUTER**.

C	O	M	P	U
T	E	R	A	B
D	F	G	H	I
K	L	N	Q	S
V	W	X	Y	Z

3. Now, we will traverse in key-matrix pair by pair and find the corresponding encipher for the pair.

- The first digraph is **CO**. The pair appears in the same row. By using **Rule 4(i)** **CO** gets encipher into **OM**.
- The second digraph is **MX**. The pair appears in the same column. By using **Rule 4(ii) MX** gets encipher into **RM**.
- The third digraph is **MU**. The pair appears in the same row. By using **Rule 4(i)** **MU** gets encipher into **PC**.
- The fourth digraph is **NI**. The pair appears in different rows and different columns. By using **Rule 4(iii) NI** gets encipher into **SG**.
- The fifth digraph is **CA**. The pair appears in different rows and different columns. By using **Rule 4(iii) CA** gets encipher into **PT**.
- The sixth digraph is **TE**. The pair appears in the same row. By using **Rule 4(i)** **TE** gets encipher into **ER**.

Therefore, the plaintext **COMMUNICATE** gets encipher (encrypted) into **OMRMPCSGPTER.**



PROGRAM :

```
key=input("Enter key : ")

key=key.replace(" ", "")
```

```
key=key.upper()
def matrix(x,y,initial):
    return [[initial for i in range(x)] for j in range(y)]


result=list()
for c in key: #storing key
    if c not in result:
        if c=='J':
result.append('I')
        else:
result.append(c)
flag=0
for i in range(65,91): #storing other character
    if chr(i) not in result:
        if i==73 and chr(74) not in result:
result.append("I")
            flag=1
elif flag==0 and i==73 or i==74:
            pass
        else:
result.append(chr(i))
k=0
my_matrix=matrix(5,5,0) #initialize matrix
for i in range(0,5): #making matrix
```

```
    for j in range(0,5):
my_matrix[i][j]=result[k]
        k+=1


def locindex(c): #get location of each character
    loc=list()
    if c=='J':
        c='I'
    for i ,j in enumerate(my_matrix):
        for k,l in enumerate(j):
            if c==l:
loc.append(i)
loc.append(k)
                return loc


def encrypt():  #Encryption
    msg=str(input("ENTER MSG:"))
    msg=msg.upper()
    msg=msg.replace(" ", "")
i=0
    for s in range(0,len(msg)+1,2):
        if s<len(msg)-1:
            if msg[s]==msg[s+1]:
                msg=msg[:s+1]+'X'+msg[s+1:]
```

```
    if len(msg)%2!=0:
        msg=msg[:]+'X'
print("CIPHER TEXT:",end=' ')
    while i<len(msg):
        loc=list()
        loc=locindex(msg[i])
        loc1=list()
        loc1=locindex(msg[i+1])
        if loc[1]==loc1[1]:

print("{}{}".format(my_matrix[(loc[0]+1)%5][loc[1]],my_matrix[(loc1[0]+1)%5][loc1[1]]),end
=' ')
elif loc[0]==loc1[0]:

print("{}{}".format(my_matrix[loc[0]][(loc[1]+1)%5],my_matrix[loc1[0]][(loc1[1]+1)%5]),end
=' ')
        else:
            print("{}{}".format(my_matrix[loc[0]][loc1[1]],my_matrix[loc1[0]][loc[1]]),end=' ')
i=i+2


def decrypt():  #decryption
    msg=str(input("ENTER CIPHER TEXT:"))
    msg=msg.upper()
    msg=msg.replace(" ", "")
print("PLAIN TEXT:",end=' ')
i=0
```

```
    while i<len(msg):
        loc=list()
        loc=locindex(msg[i])
        loc1=list()
        loc1=locindex(msg[i+1])
        if loc[1]==loc1[1]:
            print("{}{}".format(my_matrix[(loc[0]-1)%5][loc[1]],my_matrix[(loc1[0]-
1)%5][loc1[1]]),end=' ')
elif loc[0]==loc1[0]:
            print("{}{}".format(my_matrix[loc[0]][(loc[1]-1)%5],my_matrix[loc1[0]][(loc1[1]-
1)%5]),end=' ')
        else:
            print("{}{}".format(my_matrix[loc[0]][loc1[1]],my_matrix[loc1[0]][loc[1]]),end=' ')
i=i+2


while(1):
    choice=int(input("\n 1.Encryption \n 2.Decryption: \n 3.EXIT\n"))
    if choice==1:
encrypt()
elif choice==2:
decrypt()
elif choice==3:
exit()
    else:
print("Choose correct choice")
```

OUTPUT :

```
Enter key : 2

 1.Encryption
 2.Decryption:
 3.EXIT
1
ENTER MSG:Walchand Institute Of Technology
CIPHER TEXT: VB NA FC OC HO TP OY YP IK IQ H2 NS KM MI UY
 1.Encryption
 2.Decryption:
 3.EXIT
2
ENTER CIPHER TEXT:VB NA FC OC HO TP OY YP IK IQ H2 NS KM MI UY
PLAIN TEXT: WA LC HA ND IN ST IT UT EO FT EC HN OL OG YX
 1.Encryption
 2.Decryption:
 3.EXIT
3
>>>
```

CONCLUSION :

Advantages of Playfair Cipher :

- ✔ Diverse ciphertext if we scrutinize the Algorithm, we can notice at every Stage we are getting diverse ciphertext, thus more trouble to cryptanalyst.
- ✔ Brute force attack does not affect it.
- ✔ Cryptanalyze (the process of decoding cipher without knowing key) is not possible.
- ✔ Overcomes the limitation of simple Playfair square cipher.
- ✔ Easy to perform the substitution.

Limitations of Playfair Cipher :

The limitations of the Playfair cipher are as follows:

- ✔ Only 25 alphabets are supported.

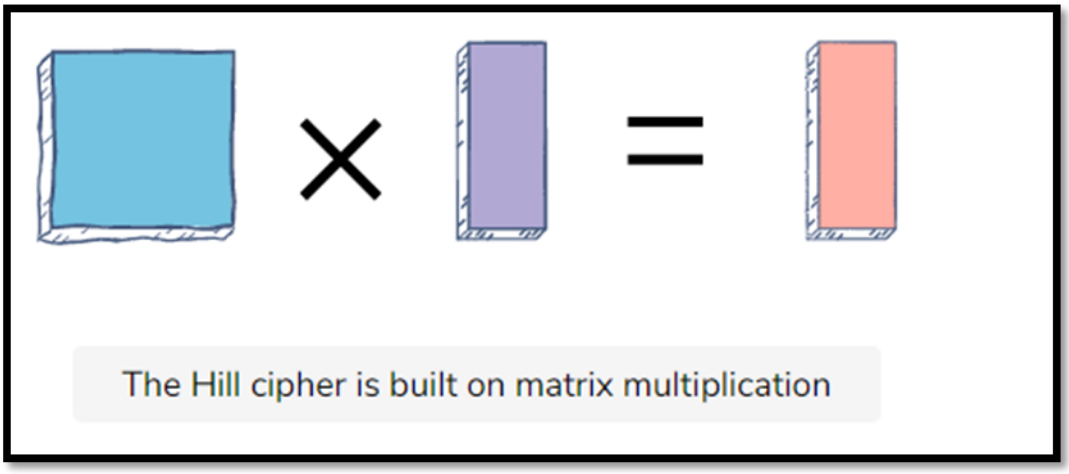
- It does not support numeric characters.
- Only either upper cases or lower cases are supported.
- The use of special characters (such as blank space, newline, punctuations, etc.) is prohibited.
- It does not support other languages, except English.
- Encryption of media files is also not supported.

ASSIGNMENT : 4

AIM : Hill Cipher for 2X2 Matrix.

THEORY :

The **Hill cipher** is a polygraphic substitution cipher built on concepts from Linear Algebra. The Hill cipher makes use of modulo arithmetic, matrix multiplication, and matrix inverses; hence, it is a more mathematical cipher than others. The Hill cipher is also a block cipher, so, theoretically, it can work on arbitrary sized blocks.



Since the Hill cipher is fairly complex, let's encrypt the text "CODE" and, later, decrypt the resulting ciphertext to understand how the Hill cipher works. To keep the example simple, we will use a straightforward substitution scheme where the letter A is mapped to 0, B is mapped to 1, etc. to stick to a 2x2 key matrix. The complexity of the Hill cipher increases with the size of the key matrix.

ALGORITHM :

Encryption :

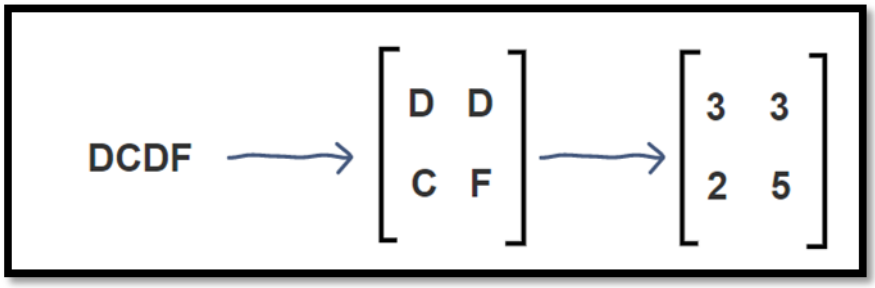
Encrypting with the Hill cipher is built on the following operation:

$$E(K, P) = (K*P) \bmod 26$$

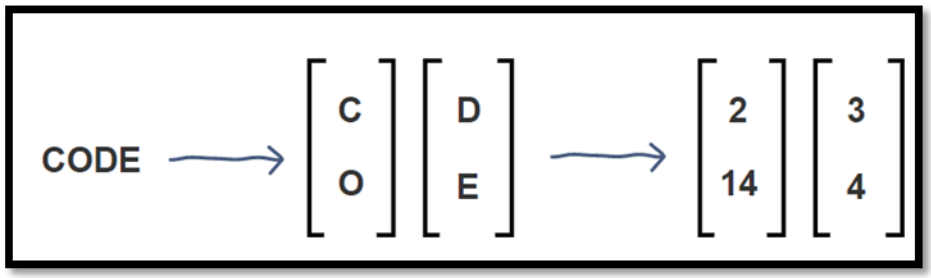
Where K is our key matrix and P is the plaintext in vector form. Matrix multiplying these two terms produces the encrypted ciphertext. Let's do so step by step:

1. Pick a keyword to encrypt your plaintext message. Let's work with the random

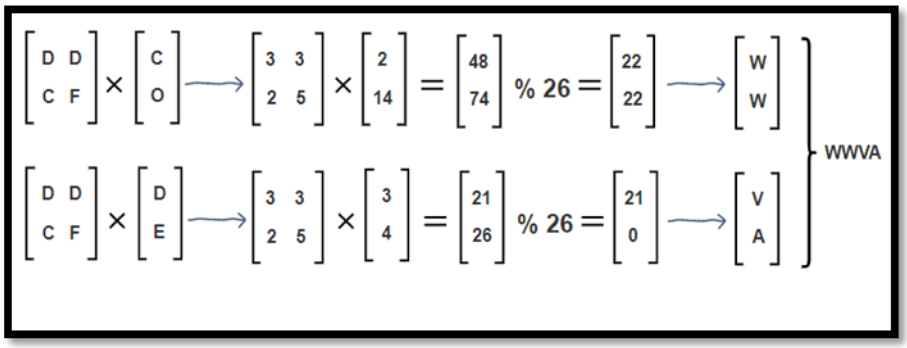
keyword “DCDF”. Convert this keyword to matrix form using your substitution scheme to convert it to a numerical 2x2 key matrix.



2. Next, we will convert our plaintext message to vector form. Since our key matrix is 2x2, the vector needs to be 2x1 for matrix multiplication to be possible. In our case, our message is four letters long so we can split it into blocks of two and then substitute to get our plaintext vectors.



3. Now, we can matrix multiply the key matrix with each 2x1 plaintext vector, take the moduli of the resulting 2x1 vectors by 26, and concatenate the results to get “WWVA”, the final ciphertext.



Decryption :

Decrypting with the Hill cipher is built on the following operation:

$$D(K, C) = (K^{-1} * C) \bmod 26$$

Where K is our key matrix and C is the ciphertext in vector form. Matrix multiplying the inverse of the key matrix with the ciphertext produces the decrypted plaintext. Let's do this step by step with our ciphertext, "WWVA":

1. First, we calculate the inverse of the key matrix. In doing so, we must keep the result between 0-25 using modulo 26. For this reason, the Extended Euclidean algorithm is used to find the modular multiplicative inverse of the key matrix determinant.

$$K^{-1} \% 26 = \begin{bmatrix} 3 & 3 \\ 2 & 5 \end{bmatrix}^{-1} \% 26 = \underbrace{((3x5) - (3x2))^{1}}_{\text{solved by Extended Euclidean Algorithm}} \times \begin{bmatrix} 5 & -3 \\ -2 & 3 \end{bmatrix} \% 26 = 3 \begin{bmatrix} 5 & 23 \\ 24 & 3 \end{bmatrix} = \begin{bmatrix} 15 & 69 \\ 72 & 9 \end{bmatrix} \% 26 = \begin{bmatrix} 15 & 17 \\ 20 & 9 \end{bmatrix}$$

2. Next, we will multiply 2x1 blocks of the ciphertext with the inverse of the key matrix to get our original plaintext message, “CODE,” back.

$$\begin{bmatrix} 15 & 17 \\ 20 & 9 \end{bmatrix} \times \begin{bmatrix} 22 \\ 22 \end{bmatrix} = \begin{bmatrix} 704 \\ 638 \end{bmatrix} \% 26 = \begin{bmatrix} 2 \\ 14 \end{bmatrix} \longrightarrow \begin{bmatrix} C \\ O \end{bmatrix}$$

$$\begin{bmatrix} 15 & 17 \\ 20 & 9 \end{bmatrix} \times \begin{bmatrix} 21 \\ 0 \end{bmatrix} = \begin{bmatrix} 315 \\ 420 \end{bmatrix} \% 26 = \begin{bmatrix} 3 \\ 4 \end{bmatrix} \longrightarrow \begin{bmatrix} D \\ E \end{bmatrix}$$

} CODE

PROGRAM :

```
import sys
import numpy as np
def cipher_encryption():
    msg = input("Enter message: ").upper()
    msg = msg.replace(" ", "")
    # if message length is odd number, append 0 at the end
len_chk = 0
    if len(msg) % 2 != 0:
        msg += "0"
len_chk = 1
    # msg to matrices
    row = 2
    col = int(len(msg)/2)
    msg2d = np.zeros((row, col), dtype=int)
    itr1 = 0
    itr2 = 0
    for i in range(len(msg)):
        if i % 2 == 0:
            msg2d[0][itr1] = int(ord(msg[i])-65)
            itr1 += 1
```

```
        else:
            msg2d[1][itr2] = int(ord(msg[i])-65)
            itr2 += 1
    # for
    key = input("Enter 4 letter Key String: ").upper()
    key = key.replace(" ", "")
    # key to 2x2
    key2d = np.zeros((2, 2), dtype=int)
    itr3 = 0
    for i in range(2):
        for j in range(2):
            key2d[i][j] = ord(key[itr3])-65
            itr3 += 1
    # checking validity of the key
    # finding determinant
    deter = key2d[0][0] * key2d[1][1] - key2d[0][1] * key2d[1][0]
    deter = deter % 26
    # finding multiplicative inverse
mul_inv = -1
    for i in range(26):
temp_inv = deter * i
        if temp_inv % 26 == 1:
mul_inv = i
            break
```

```
        else:
            continue
    # for
    if mul_inv == -1:
print("Invalid key")
sys.exit()
    # if
encryp_text = ""
itr_count = int(len(msg)/2)
    if len_chk == 0:
        for i in range(itr_count):
            temp1 = msg2d[0][i] * key2d[0][0] + msg2d[1][i] * key2d[0][1]
encryp_text += chr((temp1 % 26) + 65)
            temp2 = msg2d[0][i] * key2d[1][0] + msg2d[1][i] * key2d[1][1]
encryp_text += chr((temp2 % 26) + 65)
        # for
    else:
        for i in range(itr_count-1):
            temp1 = msg2d[0][i] * key2d[0][0] + msg2d[1][i] * key2d[0][1]
encryp_text += chr((temp1 % 26) + 65)
            temp2 = msg2d[0][i] * key2d[1][0] + msg2d[1][i] * key2d[1][1]
encryp_text += chr((temp2 % 26) + 65)
        # for
    # if else
```

```
print("Encrypted Text: {}".format(encryp_text))
def cipher_decryption():
    msg = input("Enter message: ").upper()
    msg = msg.replace(" ", "")
    # if message length is odd number, append 0 at the end
len_chk = 0
    if len(msg) % 2 != 0:
        msg += "0"
len_chk = 1
    # msg to matrices
    row = 2
    col = int(len(msg) / 2)
    msg2d = np.zeros((row, col), dtype=int)
    itr1 = 0
    itr2 = 0
    for i in range(len(msg)):
        if i % 2 == 0:
            msg2d[0][itr1] = int(ord(msg[i]) - 65)
            itr1 += 1
        else:
            msg2d[1][itr2] = int(ord(msg[i]) - 65)
            itr2 += 1
    # for
    key = input("Enter 4 letter Key String: ").upper()
```

```
    key = key.replace(" ", "")
    # key to 2x2
    key2d = np.zeros((2, 2), dtype=int)
    itr3 = 0
    for i in range(2):
        for j in range(2):
            key2d[i][j] = ord(key[itr3]) - 65
            itr3 += 1
    # for


    # finding determinant
    deter = key2d[0][0] * key2d[1][1] - key2d[0][1] * key2d[1][0]
    deter = deter % 26
    # finding multiplicative inverse
mul_inv = -1
    for i in range(26):
temp_inv = deter * i
        if temp_inv % 26 == 1:
mul_inv = i
            break
        else:
            continue
    # for
    # adjugate matrix
```

```
    # swapping
    key2d[0][0], key2d[1][1] = key2d[1][1], key2d[0][0]
    # changing signs
    key2d[0][1] *= -1
    key2d[1][0] *= -1

    key2d[0][1] = key2d[0][1] % 26
    key2d[1][0] = key2d[1][0] % 26
    # multiplying multiplicative inverse with adjugate matrix
    for i in range(2):
        for j in range(2):
            key2d[i][j] *= mul_inv
    # modulo
    for i in range(2):
        for j in range(2):
            key2d[i][j] = key2d[i][j] % 26
    # cipher to plain
decryp_text = ""
itr_count = int(len(msg) / 2)
    if len_chk == 0:
        for i in range(itr_count):
            temp1 = msg2d[0][i] * key2d[0][0] + msg2d[1][i] * key2d[0][1]
decryp_text += chr((temp1 % 26) + 65)
            temp2 = msg2d[0][i] * key2d[1][0] + msg2d[1][i] * key2d[1][1]
```

```
decryp_text += chr((temp2 % 26) + 65)
            # for
    else:
        for i in range(itr_count - 1):
            temp1 = msg2d[0][i] * key2d[0][0] + msg2d[1][i] * key2d[0][1]
decryp_text += chr((temp1 % 26) + 65)
            temp2 = msg2d[0][i] * key2d[1][0] + msg2d[1][i] * key2d[1][1]
decryp_text += chr((temp2 % 26) + 65)
            # for
    # if else
print("Decrypted Text: {}".format(decryp_text))
def main():
    choice = int(input("1. Encryption\n2. Decryption\nChoose(1,2): "))
    if choice == 1:
        print("---Encryption---")
cipher_encryption()
elif choice == 2:
        print("---Decryption---")
cipher_decryption()
    else:
print("Invalid Choice")
if __name__ == "__main__":
main()
```

OUTPUT :

```
1. Encryption
2. Decryption
Choose(1,2): 1
---Encryption---
Enter message: Roll Number 30
Enter 4 letter Key String: HILL
Encrypted Text: XDJIRZONIXQA
>>>
```

```
1. Encryption
2. Decryption
Choose(1,2): 2
---Decryption---
Enter message: Roll Number 30
Enter 4 letter Key String: HILL
Decrypted Text: FBXELFKJGFEO
>>>
```

CONCLUSION :

Thus we have learnt that the breaking of hill cipher is difficult and also learnt how to implement the hill cipher algorithm.

ASSIGNMENT : 5

AIM : Polyalphabetic Cipher (Vigenere Cipher).

THEORY :

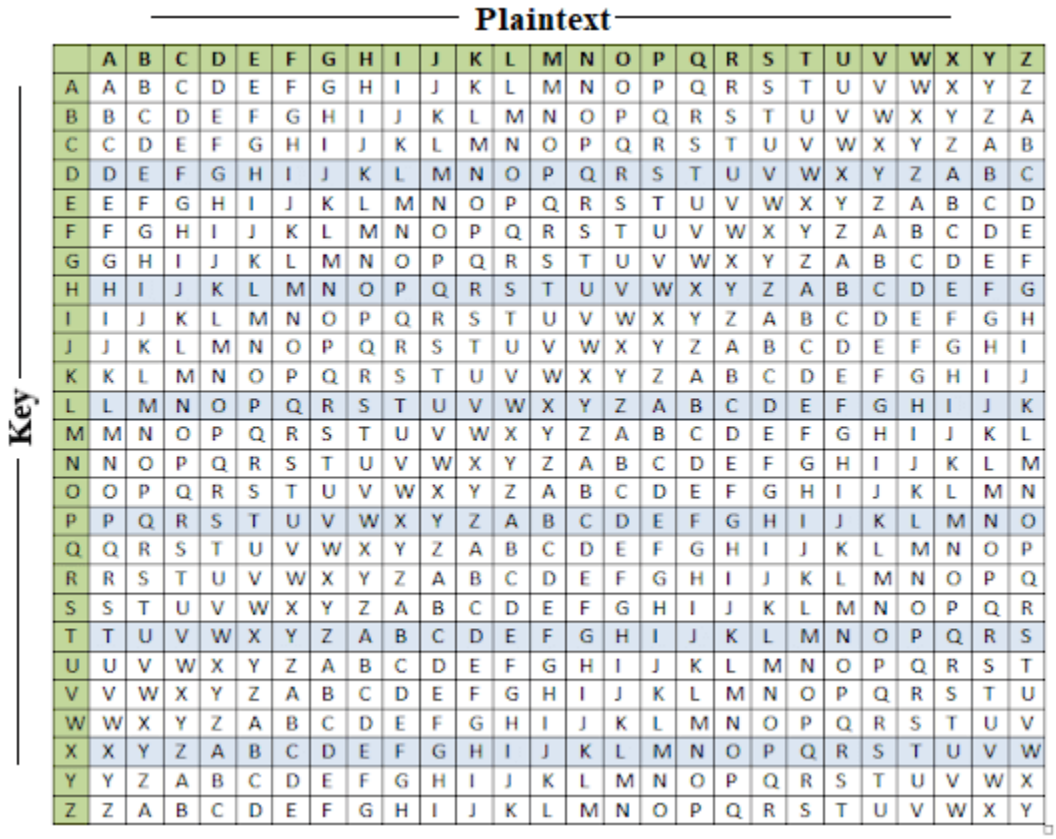
- Vigenere Cipher is a method of encrypting alphabetic text.
- It uses a simple form of polyalphabetic substitution.
- A polyalphabetic cipher is any cipher based on substitution, using multiple substitution alphabets.
- The encryption of the original text is done using the Vigenère square or Vigenère table.
- The table consists of the alphabets written out 26 times in different rows, each alphabet shifted cyclically to the left compared to the previous alphabet, corresponding to the 26 possible Caesar Ciphers.
- At different points in the encryption process, the cipher uses a different alphabet from one of the rows.
- The alphabet used at each point depends on a repeating keyword.
- The vigenere cipher is an algorithm that is used to encrypting and decrypting the text.
- The vigenere cipher is an algorithm of encrypting an alphabetic text that uses a series of interwoven caesar ciphers.
- It is based on a keyword's letters.
- It is an example of a polyalphabetic substitution cipher.
- This algorithm is easy to understand and implement.
- This algorithm was first described in 1553 by **Giovan Battista Bellaso**.
- It uses a Vigenere table or Vigenere square for encryption and decryption of the text.
- The vigenere table is also called the tabula recta.

ALGORITHM :

Two methods perform the vigenere cipher.

Method 1 :

When the vigenere table is given, the encryption and decryption are done using the vigenere table (26 * 26 matrix) in this method.



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
A	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
B	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A
C	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B
D	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C
E	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D
F	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E
G	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F
H	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G
I	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H
J	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I
K	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J
L	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K
M	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L
N	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M
O	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N
P	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
Q	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
R	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
S	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
T	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
U	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
V	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
W	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V
X	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
Y	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X
Z	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y

To generate a new key, the given key is repeated in a circular manner, as long as the length of the plain text does not equal to the new key.

J	A	V	A	T	P	O	I	N	T
B	E	S	T	B	E	S	T	B	E

Encryption

The first letter of the plaintext is combined with the first letter of the key. The column of plain text "J" and row of key "B" intersects the alphabet of "K" in the vigenere table, so the first letter of ciphertext is "K".

Similarly, the second letter of the plaintext is combined with the second letter of the key. The column of plain text "A" and row of key "E" intersects the alphabet of "E" in the vigenere table, so the second letter of ciphertext is "E".

This process continues continuously until the plaintext is finished.

Ciphertext = KENTUTGBOX

Decryption

Decryption is done by the row of keys in the vigenere table. First, select the row of the key letter, find the ciphertext letter's position in that row, and then select the column label of the corresponding ciphertext as the plaintext.

K	E	N	T	U	T	G	B	O	X
B	E	S	T	B	E	S	T	B	E

For example, in the row of the key is "B" and the ciphertext is "K" and this ciphertext letter appears in the column "J", that means the first plaintext letter is "J".

Next, in the row of the key is "E" and the ciphertext is "E" and this ciphertext letter appears in the column "A", that means the second plaintext letter is "A".

This process continues continuously until the ciphertext is finished.

Plaintext = JAVATPOINT

Method 2 :

When the vigenere table is not given, the encryption and decryption are done by Vigenar algebraically formula in this method (convert the letters (A-Z) into the numbers (0-25)).

Formula of encryption is,

$$E_i = (P_i + K_i) \bmod 26$$

Formula of decryption is,

$$D_i = (E_i - K_i) \bmod 26$$

If any case (D_i) value becomes negative (-ve), in this case, we will add 26 in the negative value.

Where,

E denotes the encryption.

D denotes the decryption.

P denotes the plaintext.

K denotes the key.

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25

PROGRAM :

```
def generateKey(string, key):
  key = list(key)
  if len(string) == len(key):
    return(key)
  else:
    for i in range(len(string) -len(key)):
key.append(key[i % len(key)])
return("" . join(key))


def encryption(string, key):
encrypt_text = []
```

```
  for i in range(len(string)):
    x = (ord(string[i]) +ord(key[i])) % 26
    x += ord('A')
encrypt_text.append(chr(x))
return("" . join(encrypt_text))

def decryption(encrypt_text, key):
orig_text = []
  for i in range(len(encrypt_text)):
    x = (ord(encrypt_text[i]) -ord(key[i]) + 26) % 26
    x += ord('A')
orig_text.append(chr(x))
return("" . join(orig_text))
if __name__ == "__main__":
  string = input("Enter the message: ")
  keyword = input("Enter the keyword: ")
  key = generateKey(string, keyword)
encrypt_text = encryption(string,key)
print("Encrypted message:", encrypt_text)
print("Decrypted message:", decryption(encrypt_text, key))
```

OUTPUT :

```
Enter the message: WALCHAND
Enter the keyword: HII
Encrypted message: DITJPIUL
Decrypted message: WALCHAND
>>>
```

CONCLUSION :

Thus we have learnt that what is Vigenere cipher, what are the methods to implement it.

ASSIGNMENT : 6

AIM :One-time pad (OTP)

THEORY :

One-time pad cipher is a type of Vignere cipher which includes the following features –

- It is an unbreakable cipher.
- The key is exactly same as the length of message which is encrypted.
- The key is made up of random symbols.
- As the name suggests, key is used one time only and never used again for any other message to be encrypted.

Due to this, encrypted message will be vulnerable to attack for a cryptanalyst. The key used for a one-time pad cipher is called **pad**, as it is printed on pads of paper.

Why is it Unbreakable?

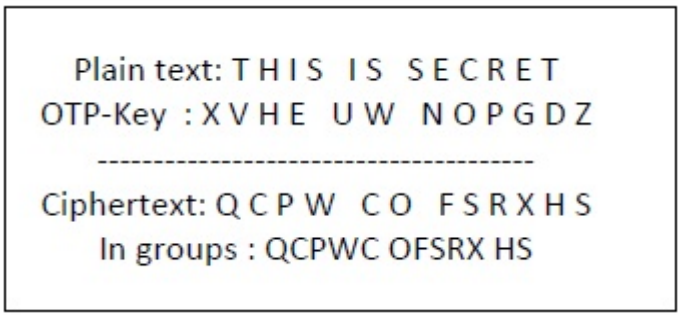
The key is unbreakable owing to the following features –

- The key is as long as the given message.
- The key is truly random and specially auto-generated.
- Key and plain text calculated as modulo 10/26/2.
- Each key should be used once and destroyed by both sender and receiver.

- There should be two copies of key: one with the sender and other with the receiver.

Encryption

To encrypt a letter, a user needs to write a key underneath the plaintext. The plaintext letter is placed on the top and the key letter on the left. The cross section achieved between two letters is the plain text. It is described in the example below –



Decryption

To decrypt a letter, user takes the key letter on the left and finds cipher text letter in that row. The plain text letter is placed at the top of the column where the user can find the cipher text letter.

ALGORITHM :

Encryption Process

- Here, firstly, we convert both our plain text and the key string into its binary form.
- The key can be either in string form or directly in binary form also.
- The bits of the key string is repeated again and again until its length becomes equal to that of the plain text.
- Perform XOR operation between the elements of the plain text with the respective elements of the ley string which hold the same positions.
- Therefore, the encryption on the plain text to convert it into ciphertext is performed as follows,

$$E(P_i, K_i) = P_i \ (XOR) \ K_i$$

Decryption Process :

The process of decrypting the ciphertext to convert it back into plain text is performed in the same way as the encryption process. Therefore, the formula for decryption of the

text under Vernam cipher is as follows,

$D(C_i, K_i) = C_i \text{ (XOR) } K_i$

PROGRAM :

```
import random
charset = "ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789"
def main():
    """Demo usage of functions."""
    vector = input()
    encrypted = encrypt(vector)
    decrypted = decrypt(encrypted[0], encrypted[1])
print("Test Vector: " + vector)
print("OTP: " + encrypted[0])
print("Encrypted: " + encrypted[1])
print("Decrypted: " + decrypted)
def encrypt(plaintext):
    """Encrypt plaintext value.
    Keyword arguments:
    plaintext -- the plaintext value to encrypt.
    """
otp = "".join(random.sample(charset, len(charset)))
    result = ""
    for c in plaintext.upper():
```

```
        if c not in otp:
            continue
        else:
            result += otp[charset.find(c)]
    return otp, result
def decrypt(otp, secret):
    """Decrypt secret value.
    Keyword arguments:
otp -- the one-time pad used when the secret value was encrypted.
    secret -- the value to be decrypted.
    """
    result = ""
    for c in secret.upper():
        if c not in otp:
            continue
        else:
            result += charset[otp.find(c)]
    return result
if __name__ == "__main__":
main()
```

OUTPUT :

```
Walchand
Test Vector: Walchand
OTP: 8PWMZTBON7LC02DE1SKJYV96UQX4RG5HIF3A
Encrypted: 98CWO82M
Decrypted: WALCHAND
>>>
```

CONCLUSION :

Advantages : Immune to most actacks.

Disadvantages : Key cannot be reused and needs very long key.

ASSIGNMENT : 7

AIM : Transposition techniques and rail cipher.

THEORY :

The transposition technique is a cryptographic technique that converts the plain text to cipher text by performing permutations on the plain text, i.e., changing each character of plain text for each round. It includes various techniques like the Rail Fence technique, Simple columnar transposition technique, simple columnar transposition technique with multiple rounds, Vernam cipher, and book Cipher to encrypt the plain text in a secure way.

Transposition Techniques

Below is the list of transposition techniques.

1. Rail-Fence Technique

Rail-Fence is the simple Transposition technique that involves writing plain text as a sequence of diagonals and then reading it row by row to produce the ciphertext.

2. Simple columnar transposition techniques

The simple columnar transposition technique can be categorized into two parts – Basic technique and multiple rounds.

Simples columnar transposition technique – basic technique. The simple columnar transposition technique simply arranges the plain text in a sequence of rows of a rectangle and reads it in a columnar manner.

How does this algorithm work?

Step 1: Write all the characters of plain text message row by row in a rectangle of predefined size.

Step 2: Read the message in a columnar manner, i.e. column by column.

Note: For reading the message, it needs not to be in the order of columns. It can happen in any random sequence.

Step 3: The resultant message is ciphertext.

3. Simple columnar transposition technique – Multiple rounds

Simple columnar transposition technique with multiple rounds is the same as basic; only the difference is that we iterate the process multiple times in multiple rounds.

Working of an algorithm

Step 1: Write all the characters of plain text message row by row in a rectangle of predefined size.

Step 2: Read the message in a columnar manner, i.e. column by column.

Note: For reading the message, it needs not to be in the order of columns. It can happen in any random sequence.

Step 3: The resultant message is ciphertext.

Step 4: Repeat the procedure from step 1 to step 3 many times as desired.

4. Vernam Cipher

A subset of Vernam cipher is called a one-time pad because it is implemented using a random set of nonrepeating characters as an input ciphertext.

Working of Algorithm

Step 1: Arrange all characters in the plain text as a number i.e. A = 0, B = 1, ….. Z = 25.
Step 2: Repeat the same procedure for all characters of the input ciphertext.
Step 3: Add each number corresponding to the plain text characters to the corresponding input ciphertext character number.
Step 4: If the sum of the number is greater than 25, subtract 26 from it.
Step 5: Translate each number of the sum into the corresponding characters.
Step 6: The output of step 5 will be a ciphertext.

In Vernam cipher, once the input ciphertext is used, it will never be used for any other message; hence it is suitable only for short messages.

ALGORITHM OF RAIL CIPHER :

Encryption :

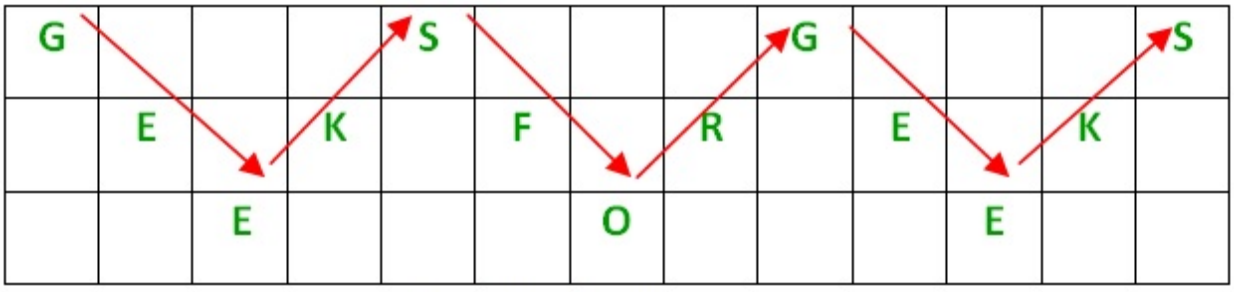
In a transposition cipher, the order of the alphabets is re-arranged to obtain the cipher-text.

In the rail fence cipher, the plain-text is written downwards and diagonally on successive rails of an imaginary fence.

When we reach the bottom rail, we traverse upwards moving diagonally, after reaching the top rail, the direction is changed again. Thus the alphabets of the message are written in a zig-zag manner.

After each alphabet has been written, the individual rows are combined to obtain the cipher-text.

For example, if the message is “GeeksforGeeks” and the number of rails = 3 then cipher is prepared as:



Decryption :

As we've seen earlier, the number of columns in rail fence cipher remains equal to the length of plain-text message. And the key corresponds to the number of rails.

Hence, rail matrix can be constructed accordingly. Once we've got the matrix we can figure-out the spots where texts should be placed (using the same way of moving diagonally up and down alternatively).

Then, we fill the cipher-text row wise. After filling it, we traverse the matrix in zig-zag manner to obtain the original text.

PROGRAM :

```
def encryptRailFence(text, key):

    # create the matrix to cipher

    # plain text key = rows ,

    # length(text) = columns

    # filling the rail matrix

    # to distinguish filled

    # spaces from blank ones

    rail = [['\n' for i in range(len(text))]
```

```
                    for j in range(key)]

# to find the direction

dir_down = False

row, col = 0, 0

for i in range(len(text)):



        # check the direction of flow

        # reverse the direction if we've just

        # filled the top or bottom rail

        if (row == 0) or (row == key - 1):

                dir_down = not dir_down

        # fill the corresponding alphabet

        rail[row][col] = text[i]

        col += 1

        # find the next row using
```

```
            # direction flag
            if dir_down:
                row += 1
            else:
                row -= 1
        # now we can construct the cipher
        # using the rail matrix
        result = []
        for i in range(key):
            for j in range(len(text)):
                if rail[i][j] != '\n':
                    result.append(rail[i][j])
        return("" . join(result))
# This function receives cipher-text
# and key and returns the original
```

```
# text after decryption

def decryptRailFence(cipher, key):



        # create the matrix to cipher

        # plain text key = rows ,

        # length(text) = columns

        # filling the rail matrix to

        # distinguish filled spaces

        # from blank ones

        rail = [['\n' for i in range(len(cipher))]

        for j in range(key)]

        # to find the direction

        dir_down = None

        row, col = 0, 0

        # mark the places with '*'
```

```
for i in range(len(cipher)):

        if row == 0:

                dir_down = True

        if row == key - 1:

                dir_down = False

        # place the marker

        rail[row][col] = '*'

        col += 1

        # find the next row

        # using direction flag

        if dir_down:

                row += 1

        else:

                row -= 1

# now we can construct the
```

```
# fill the rail matrix

index = 0

for i in range(key):

        for j in range(len(cipher)):

                if ((rail[i][j] == '*') and

                (index <len(cipher))):

                        rail[i][j] = cipher[index]

                        index += 1

# now read the matrix in

# zig-zag manner to construct

# the resultant text

result = []

row, col = 0, 0

for i in range(len(cipher)):

        # check the direction of flow
```

```
        if row == 0:
            dir_down = True
        if row == key-1:
            dir_down = False
        # place the marker
        if (rail[row][col] != '*'):
            result.append(rail[row][col])
            col += 1
        # find the next row using
        # direction flag
        if dir_down:
            row += 1
        else:
            row -= 1
    return("".join(result))
```

```
# Driver code

if __name__ == "__main__":

    message = input('Enter Message : \n')

    key = int(input('Enter key \n'))

    ciphertext = encryptRailFence(message, key)

print('Encrypted Cipher text \n' ,ciphertext)

        # Now decryption of the

        # same cipher-text

print('Decrypted message \n' ,decryptRailFence(ciphertext ,key))
```

OUTPUT :

```
Enter Message :
Walchand
Enter key
4
Encrypted Cipher text
 Wnaadlhc
Decrypted message
 Walchand
>>>
```

CONCLUSION :

Advantages : Easy and fast.

Disadvantages : The security of the code isdependent on the fact that a cryptanalyst does not know the method ofencryption and also it is not strong.

ASSIGNMENT : 8

AIM :RSA algorithm.

THOERY :

RSA or **Rivest–Shamir–Adleman** is an algorithm employed by modern computers to encrypt and decrypt messages. It is an asymmetric cryptographic algorithm. Asymmetric means that there are two different keys. This is also called public-key cryptography because one among the keys are often given to anyone. The other is the **private key** which is kept private. The algorithm is predicated on the very fact that finding the factors of an outsized number is difficult: when the factors are prime numbers, the matter is named prime factorization. It is also a key pair (public and personal key) generator.

Algorithm :

The RSA algorithm holds the following features –

RSA algorithm is a popular exponentiation in a finite field over integers including prime

numbers.

The integers used by this method are sufficiently large making it difficult to solve.

There are two sets of keys in this algorithm: private key and public key.

You will have to go through the following steps to work on RSA algorithm −

Step 1: Generate the RSA modulus

The initial procedure begins with selection of two prime numbers namely p and q, and then calculating their product N, as shown −

$$N=p*q$$

Here, let N be the specified large number.

Step 2: Derived Number (e)

Consider number e as a derived number which should be greater than 1 and less than (p -1) and (q-1). The primary condition will be that there should be no common factor of (p-1) and (q-1) except 1

Step 3: Public key

The specified pair of numbers n and e forms the RSA public key and it is made public.

Step 4: Private Key

Private Key d is calculated from the numbers p, q and e. The mathematical relationship between the numbers is as follows −

$$ed = 1 \bmod (p-1)(q-1)$$

The above formula is the basic formula for Extended Euclidean Algorithm, which takes p and q as the input parameters.

Encryption Formula :

Consider a sender who sends the plain text message to someone whose public key is (n,e). To encrypt the plain text message in the given scenario, use the following syntax −

$$C = Pe \bmod n$$

Decryption Formula :

The decryption process is very straightforward and includes analytics for calculation in

a systematic approach. Considering receiver C has the private key d, the result modulus will be calculated as –

Plaintext = Cd mod n

PROGRAM :

```
packagersa;
importjava.math.*;
importjava.util.*;

classrsa {
publicstaticvoidmain(String args[])
  {
int  n, z, d = 0, e, i;

      // The number to be encrypted and decrypted

         //author : @Tanish Banthia

double c;
BigIntegermsgback;

      Scanner sc= newScanner(System.in);    //System.in is a standard input stream
System.out.print("Enter a prime number p : ");
int p= sc.nextInt();
System.out.print("Enter a prime number q : ");
int q= sc.nextInt();
System.out.print("Enter a message msg :");
intmsg = sc.nextInt();

      n = p * q;
      z = (p - 1) * (q - 1);
System.out.println("the value of z = " + z);

for (e = 2; e < z; e++) {

         // e is for public key exponent
if (gcd(e, z) == 1) {
break;
         }
      }
System.out.println("the value of e = " + e);
for (i = 0; i<= 9; i++) {
```

```
int x = 1 + (i * z);

        // d is for private key exponent
if (x % e == 0) {
            d = x / e;
break;
        }
    }
System.out.println("the value of d = " + d);
    c = (Math.pow(msg, e)) % n;
System.out.println("Encrypted message is : " + c);

    // converting int value of n to BigInteger
BigInteger N = BigInteger.valueOf(n);

    // converting float value of c to BigInteger
BigInteger C = BigDecimal.valueOf(c).toBigInteger();
msgback = (C.pow(d)).mod(N);
System.out.println("Decrypted message is : + msgback);
  }
```

OUTPUT :

```
Enter a prime number p : 5
Enter a prime number q : 7
Enter a message msg : 13
the value of z = 24
the value of e = 5
the value of d = 5
Encrypted message is : 13.0
Decrypted message is : 13
```

```
Enter a prime number p : 3
Enter a prime number q : 5
Enter a message msg :12
the value of z = 8
the value of e = 3
the value of d = 3
Encrypted message is : 3.0
Decrypted message is : 12
```

CONCLUSION :

Advantage :

- RSA is stronger than any other symmetric key algorithm.
- RSA has overcome the weakness of symmetric algorithm i.e. authenticity and confidentiality.

Disadvantage :

- RSA has too much computation.

ASSIGNMENT : 9

AIM : Diffee Hellman Key exchange Encryption Decryption Algorithm.

THOERY :

Elliptic Curve Cryptography (ECC) is an approach to public-key cryptography, based on the algebraic structure of elliptic curves over finite fields. ECC requires a smaller key as compared to non-ECC cryptography to provide equivalent security (a 256-bit ECC security has an equivalent security attained by 3072-bit RSA cryptography).
For a better understanding of Elliptic Curve Cryptography, it is very important to understand the basics of Elliptic Curve. An elliptic curve is a planar algebraic curve defined by an equation of the form

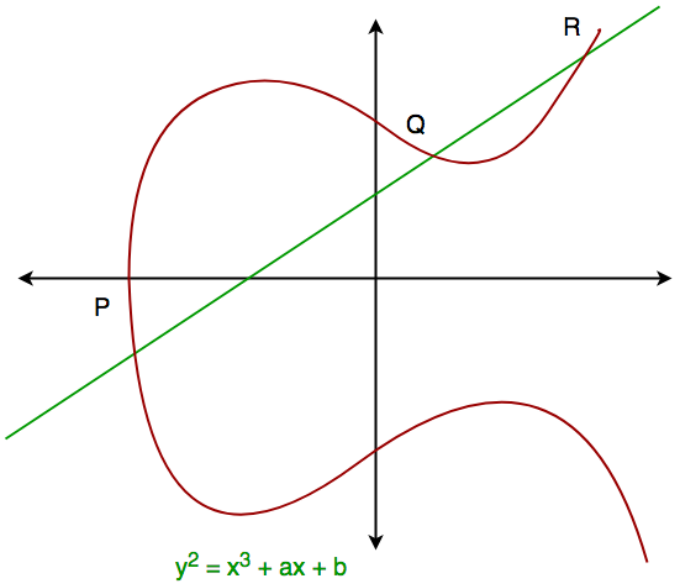
$$y^2 = x^3 + ax + b$$

Where 'a' is the co-efficient of x and 'b' is the constant of the equation

The curve is non-singular; that is its graph has no cusps or self-intersections (when

the characteristic of the Co-efficient field is equal to 2 or 3).

In general, an elliptic curve looks like as shown below. Elliptic curves could intersect almost 3 points when a straight line is drawn intersecting the curve. As we can see the elliptic curve is symmetric about the x-axis, this property plays a key role in the algorithm.



This approach uses six tuple {P, a, b, G, n, h}

P = Field that the curve is define over

G = Generator point

a, b = Values define the curve

h = Co- factor

n = Prime order of G

Diffie-Hellman algorithm

The Diffie-Hellman algorithm is being used to establish a shared secret that can be used for secret communications while exchanging data over a public network using the elliptic curve to generate points and get the secret key using the parameters.

- For the sake of simplicity and practical implementation of the algorithm, we will consider only 4 variables one prime P and G (a primitive root of P) and two private values a and b.
- P and G are both publicly available numbers. Users (say Alice and Bob) pick private values a and b and they generate a key and exchange it publicly, the opposite person received the key and from that generates a secret key after which they have the same secret key to encrypt.

Step by Step Explanation

Alice	Bob
Public Keys available = P, G	Public Keys available = P, G

Alice	Bob
Private Key Selected = a	Private Key Selected = b
Key generated : $x = G^a \bmod P$	Key Generated : $y = G^b \bmod P$

Exchange of generated keys takes place	
Key received = y	key received = x
Generated Secret Key = $k_a = y^a \bmod P$	Generated Secret Key = $k_b = x^b \bmod P$
Algebraically it can be shown that $k_a = k_b$	
Users now have a symmetric secret key to encrypt	

PROGRAM :

```
from random import randint


if __name__ == '__main__':


    # Both the persons will be agreed upon the

    # public keys G and P

    # A prime number P is taken
```

```
print("Enter P :")
    P = int(input())

    # A primitve root for P, G is taken
print("Enter G :")
    G = int(input())



print('The Value of P is :%d'%(P))
print('The Value of G is :%d'%(G))

    # Alice will choose the private key a
print("Enter Private key for Alice :")
    a = int(input())
print('The Private Key a for Alice is :%d'%(a))

    # gets the generated key
    x = int(pow(G,a,P))

    # Bob will choose the private key b
print("Enter Private Key for Bob :")
    b = int(input())
print('The Private Key b for Bob is :%d'%(b))
    # gets the generated key
```

```
    y = int(pow(G,b,P))

    # Secret key for Alice

    ka = int(pow(y,a,P))

    # Secret key for Bob

    kb = int(pow(x,b,P))

print('Secret key for the Alice is : %d'%(ka))

print('Secret Key for the Bob is : %d'%(kb))
```

OUTPUT :

```
Enter P :
23
Enter G :
9
The Value of P is :23
The Value of G is :9
Enter Private key for Alice :
3
The Private Key a for Alice is :3
Enter Private Key for Bob :
4
The Private Key b for Bob is :4
Secret key for the Alice is : 9
Secret Key for the Bob is : 9
>>>
```

```
Enter P :
23
Enter G :
5
The Value of P is :23
The Value of G is :5
Enter Private key for Alice :
3
The Private Key a for Alice is :3
Enter Private Key for Bob :
4
The Private Key b for Bob is :4
Secret key for the Alice is : 18
Secret Key for the Bob is : 18
>>>
```

CONCLUSION :

Advantages :

- The sender and receiver don't need any prior knowledge of each other.
- Once the keys are exchanged, the communication of data can be done through an insecure channel.
- The sharing of the secret key is safe.

Disadvantages :

- The algorithm can not be sued for any asymmetric key exchange.
- Similarly, it can not be used for signing digital signatures.
- Since it doesn't authenticate any party in the transmission, the Diffie Hellman key exchange is susceptible to a man-in-the-middle attack.

ASSIGNMENT : 10

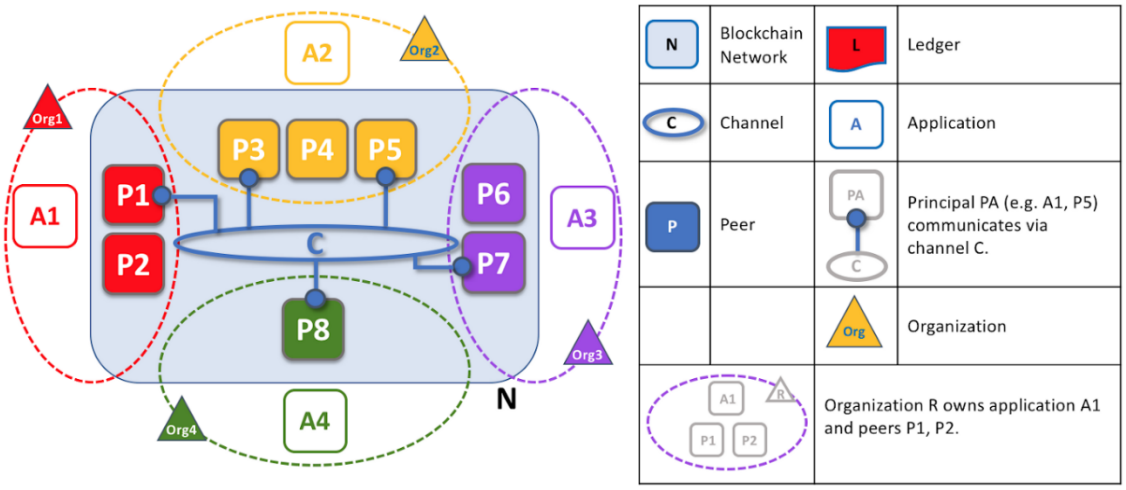
AIM : Blockchain network - Set up basic blockchain network using hyper ledger fabric.

THEORY :

Hyperledger Fabric is a blockchain platform for distributed ledger solutions underpinned by a modular architecture delivering high degrees of confidentiality, resiliency, flexibility, and scalability. It is designed to support pluggable implementations of different components and accommodate the complexity and intricacies that exists across the economic ecosystem.

Components of Hyperledger Fabric Network:

1. Ledger : A ledger consists of two distinct, though related, parts — a "blockchain" and the "state database", also known as "world state". Unlike other ledgers, blockchains are immutable — that is, once a block has been added to the chain, it cannot be changed.
2. Membership Service Provider : The Membership Service Provider (MSP) refers to an abstract component of the system that provides credentials to clients, and peers for them to participate in a Hyperledger Fabric network.
3. Smart Contract : A smart contract is code — invoked by a client application external to the blockchain network — that manages access and modifications to a set of key-value pairs in the World State.
4. Peers : A network entity that maintains a ledger and runs chaincode containers in order to perform read/write operations to the ledger. Peers are owned and maintained by members.
5. Ordering Service : A defined collective of nodes that orders transactions into a block. The ordering service exists independent of the peer processes and orders transactions on a first-come-first-serve basis for all channel's on the network.
6. Channel : A channel is a private blockchain overlay which allows for data isolation and confidentiality. A channel-specific ledger is shared across the peers in the channel, and transacting parties must be properly authenticated to a channel in order to interact with it.
7. Certificate Authority : Hyperledger Fabric CA is the default Certificate Authority component, which issues PKI-based certificates to network member organizations and their users.
8. Organizations : Also known as "members", organizations are invited to join the blockchain network by a blockchain service provider. An organization is joined to a network by adding its Membership Service Provider to the network.



The following are prerequisites for installing the required development tools:

- Operating Systems: Ubuntu Linux 14.04 / 16.04 LTS (both 64-bit), or Mac OS 10.12
- Docker Engine: Version 17.03 or higher
- Docker-Compose: Version 1.8 or higher
- Node: 8.9 or higher (Note: version 9 is not supported)
- npm: v5.x
- git: 2.9.x or higher
- Python: 2.7.x
- A code editor of your choice, we recommend VSCode.

PROGRAM / OUTPUT :

Steps :

1) Create a directory — mkdirmultichain_network :

cd multichain_network

curl -sSLhttp://bit.ly/2ysbOFE | bash -s 1.2.0

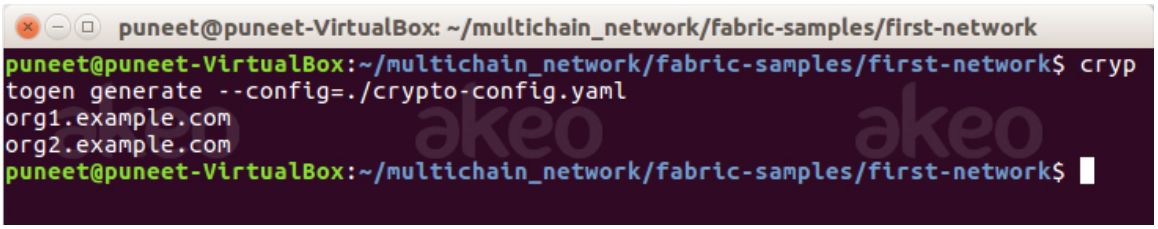
export PATH=<path to download location>/multichain_network/fabric-samples/first-network/bin:$PATH

2) Generate Certificates :

cd first-network

export FABRIC_CFG_PATH=$PWD

cryptogen generate — config=./crypto-config.yaml



In first-network folder run this command — mkdir -p tmp/composer/org1

awk 'NF {sub(/\r/, ""); printf "%s\\n",$0;}' ./crypto-config/peerOrganizations/org1.example.com/peers/peer0.org1.example.com/tls/ca.crt > ./tmp/composer/org1/ca-org1.txt

awk 'NF {sub(/\r/, ""); printf "%s\\n",$0;}' ./crypto-config/ordererOrganizations/example.com/orderers/orderer.example.com/tls/ca.crt > ./tmp/composer/ca-orderer.txt

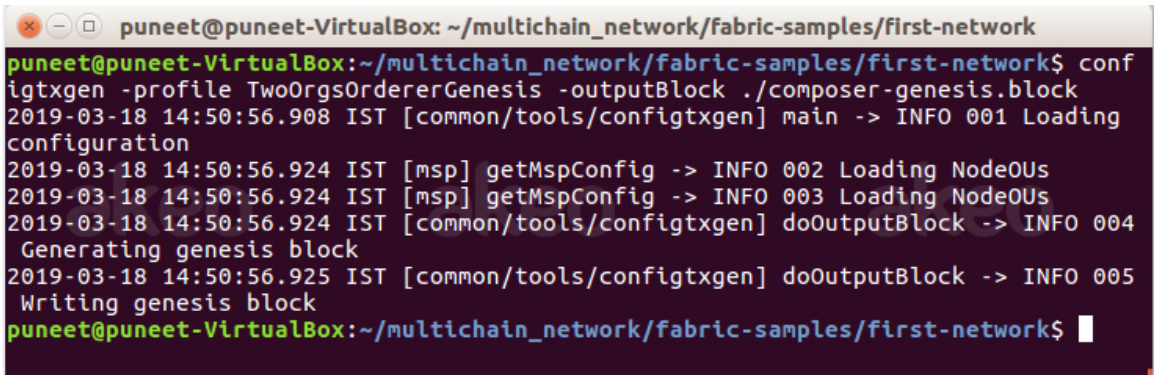
export ORG1=./crypto-config/peerOrganizations/org1.example.com/users/Admin@org1.example.com/msp

cp -p $ORG1/signcerts/A*.pem ./tmp/composer/org1

cp -p $ORG1/keystore/*_sk ./tmp/composer/org1

3) Create genesis block and channeltx :

configtxgen -profile TwoOrgsOrdererGenesis -outputBlock ./composer-genesis.block



configtxgen -profile TwoOrgsChannel -outputCreateChannelTx ./composer-channel.tx -channelIDcomposerchannel

```
puneet@puneet-VirtualBox: ~/multichain_network/fabric-samples/first-network
puneet@puneet-VirtualBox:~/multichain_network/fabric-samples/first-network$ conf
igtxgen -profile TwoOrgsChannel -outputCreateChannelTx ./composer-channel.tx -ch
annelID composerchannel
2019-03-18 15:17:36.592 IST [common/tools/configtxgen] main -> INFO 001 Loading
configuration
2019-03-18 15:17:36.612 IST [common/tools/configtxgen] doOutputChannelCreateTx -
> INFO 002 Generating new channel configtx
2019-03-18 15:17:36.614 IST [msp] getMspConfig -> INFO 003 Loading NodeOUs
2019-03-18 15:17:36.616 IST [msp] getMspConfig -> INFO 004 Loading NodeOUs
2019-03-18 15:17:36.656 IST [common/tools/configtxgen] doOutputChannelCreateTx -
> INFO 005 Writing new channel tx
puneet@puneet-VirtualBox:~/multichain_network/fabric-samples/first-network$
```

4) Update CA keys in docker composer file :

Open project in code editor.

We have to change CA keys in “docker-compose-e2e-template.yaml” file, therefore navigate to this file in vscode.

Under services section we have two certificate authorities named “ca0” and “ca1”.

Steps of Starting Hyperledger Fabric :

a)Open docker-compose-base.yaml file which is present in the bin folder and introduce following changes.

Change orderer volume binding to -

../composer-genesis.block:/var/hyperledger/orderer/orderer.genesis.block

```
10  orderer.example.com:
11    container_name: orderer.example.com
12    image: hyperledger/fabric-orderer:$IMAGE_TAG
13    environment:
14      - ORDERER_GENERAL_LOGLEVEL=INFO
15      - ORDERER_GENERAL_LISTENADDRESS=0.0.0.0
16      - ORDERER_GENERAL_GENESISMETHOD=file
17      - ORDERER_GENERAL_GENESISFILE=/var/hyperledger/orderer/orderer.genesis.block
18      - ORDERER_GENERAL_LOCALMSPID=OrdererMSP
19      - ORDERER_GENERAL_LOCALMSPDIR=/var/hyperledger/orderer/msp
20      # enabled TLS
21      - ORDERER_GENERAL_TLS_ENABLED=true
22      - ORDERER_GENERAL_TLS_PRIVATEKEY=/var/hyperledger/orderer/tls/server.key
23      - ORDERER_GENERAL_TLS_CERTIFICATE=/var/hyperledger/orderer/tls/server.crt
24      - ORDERER_GENERAL_TLS_ROOTCAS=[/var/hyperledger/orderer/tls/ca.crt]
25    working_dir: /opt/gopath/src/github.com/hyperledger/fabric
26    command: orderer
27    volumes:
28    - ../composer-genesis.block:/var/hyperledger/orderer/orderer.genesis.block
29    - ../crypto-config/ordererOrganizations/example.com/orderers/orderer.example.com/msp:/var/hyperledger/orderer/msp
30    - ../crypto-config/ordererOrganizations/example.com/orderers/orderer.example.com/tls/:/var/hyperledger/orderer/tls
31    - orderer.example.com:/var/hyperledger/production/orderer
32    ports:
33      - 7050:7050
```

Change peer volume binding to (for all 4 peers)-

- ../:/etc/configtx

```
peer0.org1.example.com:
  extends:
    file: peer-base.yaml
    service: peer-base
  environment:
    - CORE_PEER_ID=peer0.org1.example.com
    - CORE_PEER_ADDRESS=peer0.org1.example.com:7051
    - CORE_PEER_GOSSIP_BOOTSTRAP=peer1.org1.example.com:7051
    - CORE_PEER_GOSSIP_EXTERNALENDPOINT=peer0.org1.example.com:7051
    - CORE_PEER_LOCALMSPID=Org1MSP
  volumes:
      - ../:/etc/configtx
      - /var/run/:/host/var/run/
      - ../crypto-config/peerOrganizations/org1.example.com/peers/peer0.org1.example.com/msp:/etc/hyperledger/fabric/msp
      - ../crypto-config/peerOrganizations/org1.example.com/peers/peer0.org1.example.com/tls:/etc/hyperledger/fabric/tls
      - peer0.org1.example.com:/var/hyperledger/production
  ports:
    - 7051:7051
    - 7053:7053
```

b) To start fabric, run the following command -

docker-compose -f docker-compose-cli.yaml -f docker-compose-couch.yaml up -d 2>&1

The output of the above command is shown below -

```
puneet@puneet-VirtualBox:~/multichain_network/fabric-samples/first-network$ docker-compose -f docker-compose-cli.yaml -f docker-compose-couch.yaml up -
d 2>&1
Creating orderer.example.com ...
Creating couchdb1 ...
Creating couchdb3 ...
Creating couchdb2 ...
Creating orderer.example.com
Creating couchdb0 ...
Creating couchdb2
Creating couchdb1
Creating couchdb3
Creating couchdb0 ... done
Creating peer0.org1.example.com ...
Creating couchdb3 ... done
Creating peer0.org2.example.com ...
Creating peer1.org1.example.com ...
Creating peer1.org2.example.com ...
Creating peer1.org1.example.com
Creating peer0.org2.example.com
Creating peer1.org2.example.com ... done
Creating cli ...
Creating cli ... done
```

c) After completing all these steps , you can run command -

docker ps -a

This will list all running containers regarding our network setup, in our case it will list 10 containers.

```
CONTAINER ID        IMAGE                               COMMAND                  CREATED             STATUS
    PORTS                                       NAMES
badfe5ea8a92        hyperledger/fabric-peer:latest      "peer node start"        4 hours ago         Up 3 hours
                                                peer1.org1.example.com
50d9fa8f5dfb        hyperledger/fabric-peer:latest      "peer node start"        4 hours ago         Up 3 hours
                                                peer0.org2.example.com
ee4b501b7d00        hyperledger/fabric-peer:latest      "peer node start"        4 hours ago         Up 3 hours
                                                peer0.org1.example.com
32f0f4426a22        hyperledger/fabric-peer:latest      "peer node start"        4 hours ago         Up 3 hours
                                                peer1.org2.example.com
ec8f9df17258        hyperledger/fabric-couchdb          "tini -- /docker-ent…"   4 hours ago         Up 4 hours
    4369/tcp, 9100/tcp, 0.0.0.0:8984->5984/tcp  couchdb3
2b4be64efcfe        hyperledger/fabric-couchdb          "tini -- /docker-ent…"   4 hours ago         Up 4 hours
    4369/tcp, 9100/tcp, 0.0.0.0:7984->5984/tcp  couchdb2
f922625a027e        hyperledger/fabric-couchdb          "tini -- /docker-ent…"   4 hours ago         Up 4 hours
    4369/tcp, 9100/tcp, 0.0.0.0:5984->5984/tcp  couchdb0
8de729e174b0        hyperledger/fabric-couchdb          "tini -- /docker-ent…"   4 hours ago         Up 4 hours
    4369/tcp, 9100/tcp, 0.0.0.0:6984->5984/tcp  couchdb1
18830d7ccf5e        hyperledger/fabric-tools:latest     "/bin/bash"              11 hours ago        Up 11 hours
                                                cli
c7ef15868cd5        hyperledger/fabric-orderer:latest   "orderer"                11 hours ago        Up 11 hours
    0.0.0.0:7050->7050/tcp                      orderer.example.com
```

Proposed network setup is complete, our network have -

- One orderer
- Two Organizations
- Four peers (two peers on each organization)
- Couchdb for all peers

d) Creating channel :

docker exec peer0.org1.example.com peer channel create -o orderer.example.com:7050 -c composerchannel -f /etc/configtx/composer-channel.tx — tls true — cafile /etc/configtx/crypto-config/ordererOrganizations/example.com/orderers/orderer.example.com/msp/tlscacerts/tlsca.example.com-cert.pem

e) Joining first peer where channel is created —

docker exec -e

“CORE_PEER_MSPCONFIGPATH=/etc/configtx/tmp/composer/org1/Admin@org1.example.com/msp” peer0.org1.example.com peer channel join -b composer-genesis.block

f) For other peers, we have to first fetch the config of block from orderer :

For Fetching -

docker exec -e

“CORE_PEER_MSPCONFIGPATH=/etc/hyperledger/msp/users/Admin@org1.example.com/msp”

peer1.org1.example.com peer channel fetch config -o

orderer.example.com:7050 -c TwoOrgsChannel

For joining channel –

docker exec -e

“CORE_PEER_MSPCONFIGPATH=/etc/hyperledger/msp/users/Admin@org1.example.com/msp”

peer1.org1.example.com peer channel join -b composerchannel_config.block

CONCLUSION :

We learnt about installing Hyperledger Fabric development tools and Hyperledger Composer using Linux. We also successfully deployed a Hyperledger Fabric network having one orderer, two organizations and two peers in each organization.

ASSIGNMENT : 11

AIM : Cyber Crime.

THEORY :

Cyber Crime

Cyber crime or computer-oriented crime is a crime that includes a computer and a network. The computer may have been used in the execution of a crime or it may be the target.

Cyber crime is the use of a computer as a weapon for committing crimes such as committing fraud, identities theft or breaching privacy. Cyber crime, especially through the Internet, has grown in importance as the computer has become central to every field like commerce, entertainment and government. Cyber crime may endanger a person or a nation's security and financial health.

Cyber crime encloses a wide range of activities but these can generally be divided into two categories:

Crimes that aim computer networks or devices. These types of crimes involves different threats (like virus, bugs etc.) and denial-of-service (DoS) attacks.

Crimes that use computer networks to commit other criminal activities. These types of crimes include cyber stalking, financial fraud or identity theft.

Classification of Cyber Crime:

a) Cyber Terrorism:

Cyber terrorism is the use of the computer and internet to perform violent acts that result in loss of life. This may include different type of activities either by software or hardware for threatening life of citizens.

In general, Cyber terrorism can be defined as an act of terrorism committed through the use of cyberspace or computer resources.

b) Cyber Extortion:

Cyber extortion occurs when a website, e-mail server or computer system is subjected to or threatened with repeated denial of service or other attacks by malicious hackers. These hackers demand huge money in return for assurance to stop the attacks and to offer protection.

c) Cyber Warfare:

Cyber warfare is the use or targeting in a battle space or warfare context of computers, online control systems and networks. It involves both offensive and defensive

operations concerning to the threat of cyber attacks, espionage and sabotage.

d) Internet Fraud:

Internet fraud is a type of fraud or deceit which makes use of the Internet and could include hiding of information or providing incorrect information for the purpose of deceiving victims for money or property. Internet fraud is not considered a single, distinctive crime but covers a range of illegal and illicit actions that are committed in cyberspace.

e) Cyber Stalking:

This is a kind of online harassment wherein the victim is subjected to a barrage of online messages and emails. In this case, these stalkers know their victims and instead of offline stalking, they use the Internet to stalk. However, if they notice that cyber stalking is not having the desired effect, they begin offline stalking along with cyber stalking to make the victims' lives more miserable.

Prevention of Cyber Crime:

Below are some points by means of which we can prevent cyber crime:

1) Use strong password:

Maintain different password and username combinations for each account and resist the temptation to write them down. Weak passwords can be easily cracked using certain attacking methods like Brute force attack, Rainbow table attack etc.

2) Use trusted antivirus in devices:

Always use trustworthy and highly advanced antivirus software in mobile and personal computers. This leads to the prevention of different virus attack on devices.

3) Keep social media private:

Always keep your social media accounts data privacy only to your friends. Also make sure only to make friend who are known to you.

4) Keep your device software updated:

Whenever your get the updates of the system software update it at the same time because sometimes the previous version can be easily attacked.

CONCLUSION :

Thus we have learnt that what is cyber crime, classification of cyber crime and prevention of cyber crime.