In [3]:

```
#Hill Cipher
# Python3 code to implement Hill Cipher

keyMatrix = [[0] * 3 for i in range(3)]

# Generate vector for the message
messageVector = [[0] for i in range(3)]

# Generate vector for the cipher
cipherMatrix = [[0] for i in range(3)]

# Following function generates the
# key matrix for the key string
def getKeyMatrix(key):
    k = 0
    for i in range(3):
        for j in range(3):
            keyMatrix[i][j] = ord(key[k]) % 65
            k += 1

# Following function encrypts the message
def encrypt(messageVector):
    for i in range(3):
        for j in range(1):
            cipherMatrix[i][j] = 0
            for x in range(3):
                cipherMatrix[i][j] += (keyMatrix[i][x] *
                                       messageVector[x][j])
            cipherMatrix[i][j] = cipherMatrix[i][j] % 26

def HillCipher(message, key):

    # Get key matrix from the key string
    getKeyMatrix(key)

    # Generate vector for the message
    for i in range(3):
        messageVector[i][0] = ord(message[i]) % 65

    # Following function generates
    # the encrypted vector
    encrypt(messageVector)

    # Generate the encrypted text
    # from the encrypted vector
    CipherText = []
    for i in range(3):
        CipherText.append(chr(cipherMatrix[i][0] + 65))

    # Finally print the ciphertext
    print("Ciphertext: ", "".join(CipherText))

# Driver Code
def main():

    # Get the message to
    # be encrypted
    message = "MAY"

    # Get the key
    key = "GYBNQKURP"

    HillCipher(message, key)

if __name__ == "__main__":
    main()
```

```
Ciphertext:  SGC
```

In [5]:

```python
#Columnar transposition

import math

key = "HACK"

# Encryption
def encryptMessage(msg):
    cipher = ""

    # track key indices
    k_indx = 0

    msg_len = float(len(msg))
    msg_lst = list(msg)
    key_lst = sorted(list(key))

    # calculate column of the matrix
    col = len(key)

    # calculate maximum row of the matrix
    row = int(math.ceil(msg_len / col))

    # add the padding character '_' in empty
    # the empty cell of the matix
    fill_null = int((row * col) - msg_len)
    msg_lst.extend('_' * fill_null)

    # create Matrix and insert message and
    # padding characters row-wise
    matrix = [msg_lst[i: i + col]
              for i in range(0, len(msg_lst), col)]

    # read matrix column-wise using key
    for _ in range(col):
        curr_idx = key.index(key_lst[k_indx])
        cipher += ''.join([row[curr_idx]
                           for row in matrix])
        k_indx += 1

    return cipher

# Decryption
def decryptMessage(cipher):
    msg = ""

    # track key indices
    k_indx = 0

    # track msg indices
    msg_indx = 0
    msg_len = float(len(cipher))
    msg_lst = list(cipher)

    # calculate column of the matrix
    col = len(key)

    # calculate maximum row of the matrix
    row = int(math.ceil(msg_len / col))

    # convert key into list and sort
    # alphabetically so we can access
    # each character by its alphabetical position.
    key_lst = sorted(list(key))

    # create an empty matrix to
    # store deciphered message
    dec_cipher = []
```

```
    for _ in range(row):
        dec_cipher += [[None] * col]

    # Arrange the matrix column wise according
    # to permutation order by adding into new matrix
    for _ in range(col):
        curr_idx = key.index(key_lst[k_indx])

        for j in range(row):
            dec_cipher[j][curr_idx] = msg_lst[msg_indx]
            msg_indx += 1
        k_indx += 1

    # convert decrypted msg matrix into a string
    try:
        msg = ''.join(sum(dec_cipher, []))
    except TypeError:
        raise TypeError("This program cannot",
                        "handle repeating words.")

    null_count = msg.count('_')

    if null_count > 0:
        return msg[: -null_count]

    return msg

# Driver Code
msg = "Geeks for Geeks"

cipher = encryptMessage(msg)
print("Encrypted Message: {}".
               format(cipher))

print("Decryped Message: {}".
       format(decryptMessage(cipher)))
```

```
Encrypted Message: e  kefGsGsrekoe_
Decryped Message: Geeks for Geeks
```

In [ ]:

```
# Play Fair Cipher
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]]),e
nd=' ')
        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]]),e
nd=' ')
        i=i+2

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

```
Enter keyMayuri

 1.Encryption
 2.Decryption:
 3.EXIT1
ENTER MSG:You are nice
CIPHER TEXT: AP RY EL TF DI
 1.Encryption
 2.Decryption:
 3.EXIT2
ENTER CIPHER TEXT:APRYELTFDI
PLAIN TEXT: YO UA RE NI CE
 1.Encryption
 2.Decryption:
 3.EXIT3

 1.Encryption
 2.Decryption:
 3.EXIT3

 1.Encryption
 2.Decryption:
 3.EXIT4
Choose correct choice

 1.Encryption
 2.Decryption:
 3.EXIT0
Choose correct choice
```

In [ ]:

```python
#Diffie-Hellman key exchange

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
    P = 23

    # A primitive root for P, G is taken
    G = 9


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

    # Alice will choose the private key a
    a = 4
    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
    b = 3
    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))
```

```python
    print('Secret key for the Alice is : %d'%(ka))
    print('Secret Key for the Bob is : %d'%(kb))
```

In [ ]:

```python
#RSA algorithm

def rsa_algo(p: int,q: int, msg: str):
    # n = pq
    n = p * q
    # z = (p-1)(q-1)
    z = (p-1)*(q-1)

    # e -> gcd(e,z)==1      ; 1 < e < z
    # d -> ed = 1(mod z)        ; 1 < d < z
    e = find_e(z)
    d = find_d(e, z)

    # Convert Plain Text -> Cypher Text
    cypher_text = ''
    # C = (P ^ e) % n
    for ch in msg:
        # convert the Character to ascii (ord)
        ch = ord(ch)
        # encrypt the char and add to cypher text
        # convert the calculated value to Characters(chr)
        cypher_text += chr((ch ** e) % n)

    # Convert Plain Text -> Cypher Text
    plain_text = ''
    # P = (C ^ d) % n
    for ch in cypher_text:
        # convert it to ascii
        ch = ord(ch)
        # decrypt the char and add to plain text
        # convert the calculated value to Characters(chr)
        plain_text += chr((ch ** d) % n)

    return cypher_text, plain_text

def find_e(z: int):
    # e -> gcd(e,z)==1      ; 1 < e < z
    e = 2
    while e < z:
        # check if this is the required `e` value
        if gcd(e, z)==1:
            return e
        # else : increment and continue
        e += 1

def find_d(e: int, z: int):
    # d -> ed = 1(mod z)        ; 1 < d < z
    d = 2
    while d < z:
        # check if this is the required `d` value
        if ((d*e) % z)==1:
            return d
        # else : increment and continue
        d += 1

def gcd(x: int, y: int):
    # GCD by Euclidean method
    small,large = (x,y) if x<y else (y,x)

    while small != 0:
        temp = large % small
        large = small
        small = temp

    return large
```

```python
#main
if __name__ == "__main__":
    p,q = map(int, input().split())
    msg = input()

    cypher_text, plain_text = rsa_algo(p, q, msg)

    print("Encrypted (Cypher text) : ", cypher_text)
    print("Decrypted (Plain text) : ", plain_text)
```