LAB MANUAL

Cryptography and network security

Experiment-1

Aim: To write a python program for Ceaser cipher encryption

Algorithm:

1. Define the encrypt_text with plaintext and shift n
2. Perform the operation along with loops and conditions
3. Return the cipher text as answer
4. Give the plain text and shift
5. Cipher text is achieved as result
6. Run the program

Program:

```
P="hello everyone"

lst1= []

plaintext= []

for i in range(97,123):

   lst1.append(chr(i))

print(lst1)

k=[]

for j in lst1:
    k.append(lst1.index(j))
print(k)
for i in P:
    if i in lst1:
        print(lst1.index(i))
        plaintext.append(lst1.index(i))
print(plaintext)
cipher=[x+1 for x in plaintext]
print(cipher)
for m in cipher:
    if m in k:
        print(lst1[m],end="")
```

Input:

Plaintext: hello everyone

Output:

```
Plaintext: hello everyone
Shift pattern:1
Cipher text: IFMMP FWFSZPOF
```

Result:

Thus the program for Ceaser cipher encryption is executed successfully

Experiment-2

Aim: To write a python program for Ceaser cipher decryption

Algorithm:

1. Define the decrypt text with ciphertext and shift n
2. Perform the operation along with loops and conditions
3. Give the statement to get encrypted message and key
4. Return the plaintext as answer
5. Give the plain text and shift
6. Plain text is achieved as result
7. Run the program

Program:

```
def decrypt():
    encrypted_message = input("Enter the message i.e to be decrypted: ").strip()
    letters="abcdefghijklmnopqrstuvwxyz"
    k = int(input("Enter the key to decrypt: "))
    decrypted_message = ""
    for ch in encrypted_message:
        if ch in letters:
            position = letters.find(ch)
            new_pos = (position - k) % 26
            new_char = letters[new_pos]
            decrypted_message += new_char
        else:
            decrypted_message += ch
    print("Your decrypted message is:\n")
    print(decrypted_message)
decrypt()
```

Input:

```
Enter the message to be decrypted: PHHW
Enter the key to decrypt :3
```

Output:

```
Your decrypted message is: MEET
```

Result:

Thus the program for Ceaser cipher decryption is executed successfully

Experiment-3

Aim: To write a python program for brute force Ceaser cipher.

Algorithm:

1.enter the encrypted message.

2. perform the operations using loops and conditions.

3.such that every possible key from 0 to 25 are used to decrypt

4.print the result as cipher text.

5. run the program.

Program:

```
message = 'RD SFRJ NX WFLMZ'
Letters = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
for key in range(len(Letters)):
  translated = ''
  for ch in message:
    if ch in Letters:
      num = Letters.find(ch)
```

```
        num = num - key
        if num < 0:
          num = num + len(Letters)
        translated = translated + Letters[num]
      else:
        translated = translated + ch
  print('Hacking key is %s: %s' % (key, translated))
```

Output:

```
Hacking key is 0: RD SFRJ NX WFLMZ
Hacking key is 1: QC REQI MW VEKLY
Hacking key is 2: PB QDPH LV UDJKX
Hacking key is 3: OA PCOG KU TCIJW
Hacking key is 4: NZ OBNF JT SBHIV
Hacking key is 5: MY NAME IS RAGHU
Hacking key is 6: LX MZLD HR QZFGT
Hacking key is 7: KW LYKC GQ PYEFS
Hacking key is 8: JV KXJB FP OXDER
Hacking key is 9: IU JWIA EO NWCDQ
Hacking key is 10: HT IVHZ DN MVBCP
Hacking key is 11: GS HUGY CM LUABO
Hacking key is 12: FR GTFX BL KTZAN
Hacking key is 13: EQ FSEW AK JSYZM
Hacking key is 14: DP ERDV ZJ IRXYL
Hacking key is 15: CO DQCU YI HQWXK
Hacking key is 16: BN CPBT XH GPVWJ
Hacking key is 17: AM BOAS WG FOUVI
Hacking key is 18: ZL ANZR VF ENTUH
Hacking key is 19: YK ZMYQ UE DMSTG
Hacking key is 20: XJ YLXP TD CLRSF
Hacking key is 21: WI XKWO SC BKQRE
Hacking key is 22: VH WJVN RB AJPQD
Hacking key is 23: UG VIUM QA ZIOPC
Hacking key is 24: TF UHTL PZ YHNOB
Hacking key is 25: SE TGSK OY XGMNA
```

Result:

Thus the program for brute force Ceaser cipher decryption is executed successfully

Experiment-4

Aim: To write a python program for Diffie hellman key exchange

Algorithm:

1. define the secret key
2. sends the other participants for public keys of number
3. also find the private key numbers
4. atlast find the secret keys for the participants
5. if the secret keys are same or equal means then success
6. if the secret keys are not same or unequal means then invalid
7. run the program.

Program:

```
q = 23

x = 9

print('The prime number is : ',q)

print('The primitive root of q is : ',x)

a = 4

print('The Private Key a for Ram is : ',a)
```

```
b = 3
print('The Private Key b for Preethi is : ',b)
s = int(pow(x,a,q))
t = int(pow(x,b,q))
ka = int(pow(t,a,q))
kb = int(pow(s,b,q))
print('Secret key for the Ram is : ',ka)
print('Secret Key for the Preethi is : ',kb)
```

output:

```
The prime number is:  23
The primitive root of q is:  9
The Private Key a for Ram is:  4
The Private Key b for Preethi is: 3
Secret key for the Ram is: 9
Secret Key for the Preethi is: 9
```

Result:

Thus the program for Diffie hellman key exchange is executed successfully

Experiment-5

Aim: To write a python program for play fair encryption.

Algorithm:

1.get the key and define the matrix

2.choose encryption

3.get the message

4. perform the operations with loops and conditions

5. print the result as cipher text

6. execute the program.

Program:

```
def gcd(a, b):
    if b == 0:
        return a
    else:
        return gcd(b, a % b)
def is_coprime(a, b):
    return gcd(a, b) == 1
def is_valid_affine(a, b):
    return is_coprime(a, 26) and b >= 0 and b < 26
def encrypt_affine(msg, a, b):
    ciphertext = ''
    for c in msg:
        if c.isalpha():
            idx = ord(c.upper()) - ord('A')
            idx = (a * idx + b) % 26
            ciphertext += chr(idx + ord('A'))
        else:
            ciphertext += c
    return ciphertext
msg =input("Enter the plain text: ")
a = 5
b = 7
if is_valid_affine(a, b):
    ciphertext = encrypt_affine(msg, a, b)
    print("plaintext:", msg)
    print("Ciphertext:", ciphertext)
else:
    print("Invalid values of a and/or b.")
```

Input:

```
Enter the plain text: meet
```

Output:

```
plaintext: meet
Ciphertext: PBBY
```

Result:

Thus the program for play fair encryption is executed successfully

Experiment-6

Aim: To write a python program for play fair decryption

Algorithm:

1.get the key and define the matrix

2.choose decryption

3.get the message

4. perform the operations with loops and conditions

5. print the result as plain text

6. execute the program.

Program:

```
matrix = [['M', 'F', 'H', 'I', 'J', 'K'],
        ['U', 'N', 'O', 'P', 'Q', ' '],
        ['Z', 'V', 'W', 'X', 'Y', ' '],
        ['E', 'L', 'A', 'R', 'G', ' '],
        ['D', 'S', 'T', 'B', 'C', ' ']]
```

```
def playfair_encode(message):
    message = message.upper()
    message = message.replace('J', 'I')
    message = message.replace(' ', '')
    if len(message) % 2 != 0:
        message += 'X'
    ciphertext = ''
    for i in range(0, len(message), 2):
        a = message[i]
        b = message[i+1]
        a_row, a_col = 0, 0
        b_row, b_col = 0 , 0
        for row in range(len(matrix)):
            if a in matrix[row]:
                a_row = row
                a_col = matrix[row].index(a)
            if b in matrix[row]:
                b_row = row
                b_col = matrix[row].index(b)
        if a_row == b_row:
            ciphertext += matrix[a_row][(a_col+1)%6]
            ciphertext += matrix[b_row][(b_col+1)%6]
        elif a_col == b_col:
            ciphertext += matrix[(a_row+1)%5][a_col]
            ciphertext += matrix[(b_row+1)%5][b_col]
        else:
            ciphertext += matrix[a_row][b_col]
            ciphertext += matrix[b_row][a_col]
    return ciphertext
message = 'Must see you over Cadogan West. Coming at once.'
ciphertext = playfair_encode(message)
print(ciphertext)
```

output:

```
UZTBDLGZPNNWLGTGTU
ROVLDDHTQFJQLTHPODGIZ
```

Result:

Thus the program for play fair decryption is executed successfully

Experiment-7

Aim: To write a python program for monoalphabetic substitution.

Algorithm:

1. enter the plain text.
2. perform the operations with loops and conditions
3. return the result as cipher text
4. cipher text is archieved as a result
5. execute the program.

Program:

```
pt=str(input("ENTER THE PLAIN TEXT : "))
cipher=""
letter="abcdefghijklmnopqrstuvwxyz"
common=max(set(pt),key=pt.count)
print("COMMON LETTER : "+common)
if common in letter:
    com=letter.find(common)
```

```
key=com-6
#print("key = "+common+" - g = "+key)
if (key<0):
    key=26-key
for i in pt:
    if i in letter:
        pos=letter.find(i)
        new_pos=(pos+key)%26
        new_char=letter[new_pos]
        cipher+=new_char
print("CIPHER TEXT : "+cipher)
```

Input:

```
Enter the plain text: meet me after
```

Output:

```
Common letter: e
Cipher text: oggvogchvgt
```

Result:

Thus the program for monoalphabetic substitution is executed successfully.

Experiment-8

Aim: To write a python program for polyalphabetic substitution.

Algorithm:

1. enter the plain text.
2. perform the operations with loops and conditions
3. return the result as cipher text
4. cipher text is archieved as a result
5. execute the program.

Program:

```
alphabet = "abcdefghijklmnopqrstuvwxyz"
key = str(input("enter the key: "))
plaintext = str(input("enter the string: "))
ciphertext = ""
for i in range(len(plaintext)):
    index = alphabet.index(plaintext[i])
    key_index = i % len(key)
    key_char = key[key_index]
    key_alphabet_index = alphabet.index(key_char)
    cipher_index = (index + key_alphabet_index) % 26
    ciphertext += alphabet[cipher_index]
print("cipher text is: ",ciphertext)
```

Input:

```
enter the key: deceptive
enter the string: discovered
```

Output:

```
cipher text is:  gmugdommig
```

Result:

Thus the program for polyalphabetic substitution is executed successfully

Experiment-9

Aim: To write a python program for vernam cipher encryption.

Algorithm:

```
1.define function with plain text and key
2.check the lengths of plain text and key are same or not
3. perform the operations with loops and conditions
4. return the result as cipher text
5. cipher text is archieved as a result
6. execute the program.
```

Program:

```
def vernam(plain_text,key):
    plain_text=plain_text.replace(" ","")
    key=key.replace(" ","")
    plain_text=plain_text.lower()
    key=key.lower()
    if(len(plain_text)!=len(key)):
        print("Lengths are different")
    else:
        cipher_text=""
        for i in range(len(plain_text)):
            k1=ord(plain_text[i])-97
            k2=ord(key[i])-97
            s=chr((k1+k2)%26+97)
            cipher_text+=s
        print("Enrypted message is: ",cipher_text)
plain_text=input("Enter the message: ")
key=input("Enter the one time pad: ")
vernam(plain_text,key)
```

input:

```
Enter the message: attack
Enter the onetime pad: artery
```

Output:

```
Enrypted message is:  akmeti
```

Result:

Thus the program for vernam encryption is executed successfully

Experiment-10

Aim: To write a python program for vernam cipher decryption.

Algorithm:

```
1.define function with cipher text and key
2.check the lengths of ciphertext and key are same or not
3. perform the operations with loops and conditions
4. return the result as plain text
5. plain text is archieved as a result
6. execute the program.
```

Program:

```python
def vernam(cipher_text,key):
    cipher_text=cipher_text.lower()
    key=key.lower()
    cipher_text=cipher_text.replace(" ","")
    key=key.replace(" ","")
    plain_text=""
    for i in range(len(cipher_text)):
        k1=ord(cipher_text[i])-97
        k2=ord(key[i])-97
        s=chr((((k1-k2)+26)%26)+97)
        plain_text+=s
    print("Decrypted message is: ",plain_text)
plain_text=input("Enter the message to be decrypted: ")
key=input("Enter the one time pad: ")
vernam(plain_text,key)
```

input:

```
Enter the message to be decrypted: akmeti
Enter the onetime pad: artery
```

Output:

```
Decrypted message is:  attack
```

Result:

Thus the program for vernam decryption is executed successfully

Experiment-11

Aim: To write a python program for vigenere cipher encryption.

Algorithm:

1.get the key

2.choose encryption.

3. get the message.

4. perform the operations with loops and conditions

5. return the result as cipher text

6. cipher text is archieved as a result

7. execute the program.

Program:

```
import string
main=string.ascii_lowercase
def conversion(plain_text,key):
    index=0
    cipher_text=""
    plain_text=plain_text.lower()
    key=key.lower()
    for c in plain_text:
        if c in main:
            off=ord(key[index])-ord('a')
            encrypt_num=(ord(c)-ord('a')+off)%26
            encrypt=chr(encrypt_num+ord('a'))
            cipher_text+=encrypt
            index=(index+1)%len(key)
        else:
            cipher_text+=c
    print("plain text: ",plain_text)
    print("cipher text: ",cipher_text)
plain_text=input("Enter the message: ")
key=input("Enter the key: ")
conversion(plain_text,key)
```

input:

```
Enter the message: hello everyone
Enter the key: 4
```

Output:

```
plain text:  hello everyone
cipher text:  olssv lclyfvul
```

Result:

Thus the program for vigenere encryption is executed successfully

Experiment-12

Aim: To write a python program for vigenere cipher decryption.

Algorithm:

```
1.get the key
2.choose decryption.
3. get the message.
4. perform the operations with loops and conditions
5. return the result as plain text
6. plain text is archieved as a result
```

Program:

```
import string
main=string.ascii_lowercase
def conversion(cipher_text,key):
    index=0
    plain_text=""
    cipher_text=cipher_text.lower()
    key=key.lower()
    for c in cipher_text:
        if c in main:
            off=ord(key[index])-ord('a')
            positive_off=26-off
            decrypt=chr((ord(c)-
ord('a')+positive_off)%26+ord('a'))
            plain_text+=decrypt
            index=(index+1)%len(key)
        else:
            plain_text+=c
    print("cipher text: ",cipher_text)
    print("plain text (message): ",plain_text)
cipher_text=input("Enter the message to be decrypted:
")
key=input("Enter the key for decryption: ")
conversion(cipher_text,key)
```

input:

```
Enter the message to be decrypted: olssv lclyfvul
Enter the key for decryption: 4
```

Output:

```
cipher text:  olssv lclyfvul
plain text (message):  hello everyone
```

Result:

Thus the program for vigenere encryption is executed successfully

Experiment-13

Aim: To write a python program for affine cipher

Algorithm:

1.print the input statement

2.check the choice one conditions

3. print the alphabet

4. check the choice two conditions

5.run the program.

Program:

```
ct=str(input("ENTER THE PLAIN TEXT : "))
a=int(input("ENTER a : "))
b=int(input("ENTER b : "))
letter="abcdefghijklmnopqrstuvwxyz"
dec=""
for x in ct:
   en=0
   if x in letter:
      pos=letter.find(x)
```

```
        en=((a*pos)+b)%26
        dec+=letter[en]
print("CIPHER TEXT : "+dec)
```

input:

```
ENTER THE PLAIN TEXT : meet me after lunch
ENTER a : 46
ENTER b : 40
```

Output:

```
CIPHER TEXT : uqqeuqokeqqayocy
```

Result:

Thus the program for affine cipher is executed successfully

Experiment-14

Aim: To write a python program for additive cipher

Algorithm:

1declare the variables.

2.define the statements.

3.return the value

4.import the packets.

5. run the program

Program:

```
pt=str(input("ENTER THE PLAIN TEXT : "))
cipher=""
letter="abcdefghijklmnopqrstuvwxyz"
common=max(set(pt),key=pt.count)
print("COMMON LETTER : "+common)
if common in letter:
   com=letter.find(common)
key=com-6
```

```
if (key<0):
    key=26-key
for i in pt:
    if i in letter:
        pos=letter.find(i)
        new_pos=(pos+key)%26
        new_char=letter[new_pos]
        cipher+=new_char
print("CIPHER TEXT : "+cipher)
```

input:

```
ENTER THE PLAIN TEXT : meet
```

Output:

```
COMMON LETTER : e
CIPHER TEXT : oggv
```

Result:

Thus the program for additive cipher is executed successfully

Experiment-15

Aim: To write a python program for row column transposition encryption.

Algorithm:

1.define the function with plaintext ‘s’ and key

2.check the range of ‘i’ with key

3.perform the operations along with loops and conditions.

4.print the message matrix.

5.print the cipher text as result.

Program:

```
import math
def row(s,key):
    temp=[]
    for i in key:
        if i not in temp:
            temp.append(i)
    k=""
    for i in temp:
        k+=i
    print("The key used for encryption is: ",k)
```

```
b=math.ceil(len(s)/len(k))
if(b<len(k)):
   b=b+(len(k)-b)
arr=[['_' for i in range(len(k))]
    for j in range(b)]
i=0
j=0
for h in range(len(s)):
   arr[i][j]=s[h]
   j+=1
   if(j>len(k)-1):
      j=0
      i+=1
print("The message matrix is: ")
for i in arr:
   print(i)
cipher_text=""
kk=sorted(k)
for i in kk:
   h=k.index(i)
   for j in range(len(arr)):
```

```
            cipher_text+=arr[j][h]
    print("The cipher text is: ",cipher_text)
msg=input("Enter the message: ")
key=input("Enter the key in alphabets: ")
row(msg,key)
```

input:

```
Enter the message: welcome everyone
Enter the key in alphabets: daddy
```

Output:

```
The key used for encryption is:  day
The message matrix is:
['w', 'e', 'l']
['c', 'o', 'm']
['e', ' ', 'e']
['v', 'e', 'r']
['y', 'o', 'n']
['e', '_', '_']
The cipher text is:  eo eo_wcevyelmern_
```

Result:

Thus the program for row column encryption is executed successfully

Experiment-16

Aim: To write a python program for row column transposition decryption.

Algorithm:

1.define the function with plaintext ‘s’ and key

2.check the range of ‘i’ with key

3.perform the operations along with loops and conditions.

4.print the message matrix.

5.print the cipher text as result.

Program:

```
import math
def row(s,key):
    temp=[]
    for i in key:
        if i not in temp:
            temp.append(i)
    k=""
    for i in temp:
        k+=i
    print("The key used for encryption is: ",k)
    arr=[['' for i in range(len(k))]
         for j in range(int(len(s)/len(k)))]
    kk=sorted(k)
    d=0
    for i in kk:
        h=k.index(i)
        for j in range(len(k)):
            arr[j][h]=s[d]
            d+=1
    print("The message matrix is: ")
    for i in arr:
        print(i)
    plain_text=""
    for i in arr:
        for j in i:
            plain_text+=j
    print("The plain text is: ",plain_text)
msg=input("Enter the message to be decrypted: ")
key=input("Enter the key in alphabets: ")
row(msg,key)
```

input:

```
Enter the message to be decrypted: hello everyone
Enter the key in alphabets: daddy
```

Output:

```
The key used for encryption is:  day
The message matrix is:
['l', 'h', 'e']
['o', 'e', 'v']
[' ', 'l', 'e']
['', '', '']
The plain text is:  lheoev le
```

Result:

Thus the program for row column decryption is executed successfully

Experiment-17

Aim: To write a python program for rail fence encryption.

Algorithm:

```
1.define function with depth.
2.define function with string and depth
3.perform the operations along with loops and conditions.
4.give the plaintext and depth as input.
5.print the cipher text as result.
```

Program:

```
def sequence(n):
    arr=[]
    i=0
    while(i<n-1):
        arr.append(i)
        i+=1
    while(i>0):
        arr.append(i)
        i-=1
    return(arr)
```

```
def railfence(s,n):
    s=s.lower()
    L=sequence(n)
    print("The raw sequence of indices: ",L)
    temp=L
    while(len(s)>len(L)):
        L=L+temp
    for i in range(len(L)-len(s)):
        L.pop()
    print("The row indices of the characters in the given
string: ",L)
    print("Transformed message for encryption: ",s)
    num=0
    cipher_text=""
    while(num<n):
        for i in range(L.count(num)):
            cipher_text=cipher_text+s[L.index(num)]
            L[L.index(num)]=n
        num+=1
    print("The cipher text is: ",cipher_text)
plain_text=input("Enter the string to be encrypted: ")
```

```
n=int(input("Enter the number of rails: "))
railfence(plain_text,n)
```

input:

```
Enter the string to be encrypted: meet me after toga party
Enter the number of rails: 3
```

Output:

```
The raw sequence of indices:  [0, 1, 2, 1]
The row indices of the characters in the given string:
[0, 1, 2, 1, 0, 1, 2, 1, 0, 1, 2, 1, 0, 1, 2, 1, 0, 1, 2, 1, 0, 1, 2, 1]
Transformed message for encryption:  meet me after toga party
The cipher text is:  m argaetm fe oapryeett t
```

Result:

Thus the program for rail fence encryption is executed successfully

Experiment-18

Aim: To write a python program for rail fence decryption.

Algorithm:

1.define function with depth.

2.define function with string and depth

3.perform the operations along with loops and conditions.

4.give the cipher text and depth as input.

5.print the plain text as result.

Program:

```
def sequence(n):
    arr=[]
    i=0
    while(i<n-1):
        arr.append(i)
        i+=1
    while(i>0):
        arr.append(i)
        i-=1
    return(arr)
```

```
def railfence(cipher_text,n):
    cipher_text=cipher_text.lower()
    L=sequence(n)
    print("The raw sequence of indices: ",L)
    temp=L
    while(len(cipher_text)>len(L)):
        L=L+temp
    for i in range(len(L)-len(cipher_text)):
        L.pop()
    temp1=sorted(L)
    print("The row indices of the characters in the cipher
string: ",L)
    print("The row indices of the characters in the plain
string: ",temp1)
    print("Transformed message for decryption:
",cipher_text)
    plain_text=""
    for i in L:
        k=temp1.index(i)
        temp1[k]=n
        plain_text+=cipher_text[k]
    print("The cipher text is: ",plain_text)
```

```
cipher_text=input("Enter the string to be decrypted: ")
n=int(input("Enter the number of rails: "))
railfence(cipher_text,n)
```

input:

```
Enter the string to be decrypted: math gr etefe teo aate
artpy
Enter the number of rails: 3
```

Output:

```
The raw sequence of indices:  [0, 1, 2, 1]
The row indices of the characters in the cipher string:
[0, 1, 2, 1, 0, 1, 2, 1, 0, 1, 2, 1, 0, 1, 2, 1, 0, 1, 2, 1, 0, 1,
2, 1, 0, 1, 2, 1]
The row indices of the characters in the plain string:  [0,
0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2,
2, 2, 2, 2, 2]
Transformed message for decryption:  math gr etefe teo
aate artpy
The cipher text is:  meet me after the toga party
```

Result:

Thus the program for rail fence decryption is executed successfully

Experiment-19

Aim: To write a python program for RSA algorithm.

Algorithm:

1.import the math packet

2. print the input values

3.perform the loop operations and condition.

4. print the private and public key.

5.print the result.

Program:

```
import math
p = int(input("Enter p: "))
q = int(input("Enter q: "))
n = p*q
print("n: ", n)
phi = (p-1)*(q-1)
print("phi: ",phi)
e = int(input("Enter e: "))
while(e<phi):
    if (math.gcd(e,phi)==1):
        break
    else:
        e+=1
print("e: ", e)
j = 0
while True:
    if ((j * e) % phi == 1):
        d = j
        break
    j += 1
print("d: ", d)
print(f'Public key: {e, n}')
print(f'Private key: {d, n}')
msg = int(input("Enter message: "))
print(f'Original message:{msg}')
C = pow(msg, e)
C = math.fmod(C, n)
print(f'Encrypted message: {C}')
M = pow(C, d)
M = math.fmod(M, n)
```

Output:

```
Enter p: 7
Enter q: 11
n:  77
phi:  60
Enter e: 7
e:  7
d:  43
Public key: (7, 77)
Private key: (43, 77)
Enter message: 9
Original message:9
Encrypted message: 37.0
Decrypted message: 38.0
```

Result:

Thus the program for RSA algorithm is executed successfully

Experiment-20

Aim: To write a python program for play fair algorithm.

Algorithm:

1. Get the key and define the matrix.
2. Choose encryption or decryption.
3. Get the message.
4. Perform the operations with loops and conditions.
5. Print the result.

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_ma
trix[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_ma
trix[loc1[0]][loc[1]]),end=' ')
        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")
```

output:

```
Enter key4
 1.Encryption
 2.Decryption:
 3.EXIT1
ENTER MSG:we together forever
CIPHER TEXT: UG YT HF SI GP IL PG UF SW
```

Result:

Thus the program for Play fair algorithm is executed successfully

Experiment-21

Aim: To write a python program for hill cipher.

Algorithm:

1.Let the order of the encryption key be N (as it is a square matrix).

2.Your text is divided into batches of length N and converted to numerical vectorsby a simple mapping starting with A=0 and so on.

3.The key is then multiplied with the newly created batch vector to obtain the encoded vector.

4.After each multiplication modular 36 calculations are performedon the vectors so as to bring the numbers between 0 and 36 and then mapped with their corresponding alpha numerics.

5.While decrypting, the decrypting key is found which is the inverse of the

6.encrypting key modular 36. The same process is repeated for decrypting to get the original message back.

Program:

```
import string

import numpy

def greatest_common_divisor(a: int, b: int) -> int:
 """

    >>> greatest_common_divisor(4, 8)
    4
    >>> greatest_common_divisor(8, 4)
    4
    >>> greatest_common_divisor(4, 7)
    1
    >>> greatest_common_divisor(0, 10)
    10
    """
    return b if a == 0 else greatest_common_divisor(b %
a, a)
class HillCipher:
    key_string = string.ascii_uppercase + string.digits
    # This cipher takes alphanumerics into account
    # i.e. a total of 36 characters
    # take x and return x % len(key_string)
    modulus = numpy.vectorize(lambda x: x % 36)
    to_int = numpy.vectorize(lambda x: round(x))
    def __init__(self, encrypt_key):
        """
        encrypt_key is an NxN numpy array
        """
        self.encrypt_key = self.modulus(encrypt_key)  #
mod36 calc's on the encrypt key
        self.check_determinant()  # validate the
determinant of the encryption key
        self.decrypt_key = None
        self.break_key = encrypt_key.shape[0]
    def replace_letters(self, letter: str) -> int:
        """
        >>> hill_cipher = HillCipher(numpy.array([[2, 5],
[1, 6]]))
        >>> hill_cipher.replace_letters('T')
        19
        >>> hill_cipher.replace_letters('0')
        26
        """
        return self.key_string.index(letter)


    def replace_digits(self, num: int) -> str:
        """
        >>> hill_cipher = HillCipher(numpy.array([[2, 5],
[1, 6]]))
        >>> hill_cipher.replace_digits(19)
        'T'
        >>> hill_cipher.replace_digits(26)
        '0'
        """
        return self.key_string[round(num)]

    def check_determinant(self) -> None:
        """
        >>> hill_cipher = HillCipher(numpy.array([[2, 5],
[1, 6]]))
        >>> hill_cipher.check_determinant()
        """
        det = round(numpy.linalg.det(self.encrypt_key))
        if det < 0:
            det = det % len(self.key_string)
        req_l = len(self.key_string)
        if greatest_common_divisor(det,
len(self.key_string)) != 1:
            raise ValueError(
                f"determinant modular {req_l} of encryption
key({det}) is not co prime w.r.t {req_l}.\nTry another
key."
```

```
        )
    def process_text(self, text: str) -> str:
        """
        >>> hill_cipher = HillCipher(numpy.array([[2, 5],
[1, 6]]))
        >>> hill_cipher.process_text('Testing Hill Cipher')
        'TESTINGHILLCIPHERR'
        >>> hill_cipher.process_text('hello')
        'HELLOO'
        """
        chars = [char for char in text.upper() if char in
self.key_string]
        last = chars[-1]
        while len(chars) % self.break_key != 0:
            chars.append(last)
        return "".join(chars)
   def encrypt(self, text: str) -> str:
        """
 >>> hill_cipher = HillCipher(numpy.array([[2, 5], [1,
6]]))
        >>> hill_cipher.encrypt('testing hill cipher')
        'WHXYJOLM9C6XT085LL'
        >>> hill_cipher.encrypt('hello')
        '85FF00'
        """
        text = self.process_text(text.upper())
        encrypted = ""
        for i in range(0, len(text) - self.break_key + 1,
self.break_key):
            batch = text[i : i + self.break_key]
            batch_vec = [self.replace_letters(char) for char
in batch]
            batch_vec = numpy.array([batch_vec]).T
            batch_encrypted =
self.modulus(self.encrypt_key.dot(batch_vec)).T.tolist()
[
                0
            ]
            encrypted_batch = "".join(
                self.replace_digits(num) for num in
batch_encrypted
            )
            encrypted += encrypted_batch
        return encrypted
```

```
    def make_decrypt_key(self):
        """
        >>> hill_cipher = HillCipher(numpy.array([[2, 5],
[1, 6]]))
        >>> hill_cipher.make_decrypt_key()
        array([[ 6., 25.],
               [ 5., 26.]])
        """
        det = round(numpy.linalg.det(self.encrypt_key))
        if det < 0:
            det = det % len(self.key_string)
        det_inv = None
        for i in range(len(self.key_string)):
            if (det * i) % len(self.key_string) == 1:
                det_inv = i
                break
        inv_key = (
            det_inv
            * numpy.linalg.det(self.encrypt_key)
            * numpy.linalg.inv(self.encrypt_key)
        )
        return self.to_int(self.modulus(inv_key))
    def decrypt(self, text: str) -> str:
        """
        >>> hill_cipher = HillCipher(numpy.array([[2, 5],
[1, 6]]))
        >>>
hill_cipher.decrypt('WHXYJOLM9C6XT085LL')
        'TESTINGHILLCIPHERR'
        >>> hill_cipher.decrypt('85FF00')
        'HELLOO'
        """
        self.decrypt_key = self.make_decrypt_key()
        text = self.process_text(text.upper())
        decrypted = ""
        for i in range(0, len(text) - self.break_key + 1,
self.break_key):
            batch = text[i : i + self.break_key]
            batch_vec = [self.replace_letters(char) for char
in batch]
            batch_vec = numpy.array([batch_vec]).T
            batch_decrypted =
self.modulus(self.decrypt_key.dot(batch_vec)).T.tolist()
[
                0
            ]
            decrypted_batch = "".join(
                self.replace_digits(num) for num in
batch_decrypted
            )
            decrypted += decrypted_batch
        return decrypted


def main():
    N = int(input("Enter the order of the encryption key:
"))
    hill_matrix = []
    print("Enter each row of the encryption key with
space separated integers")
    for i in range(N):
        row = [int(x) for x in input().split()]
        hill_matrix.append(row)
    hc = HillCipher(numpy.array(hill_matrix))
    print("Would you like to encrypt or decrypt some
text? (1 or 2)")
    option = input("\n1. Encrypt\n2. Decrypt\n")
    if option == "1":
        text_e = input("What text would you like to
encrypt?: ")
        print("Your encrypted text is:")
        print(hc.encrypt(text_e))
    elif option == "2":
        text_d = input("What text would you like to
decrypt?: ")
        print("Your decrypted text is:")
        print(hc.decrypt(text_d))
if __name__ == "__main__":
    import doctest
    doctest.testmod()
    main()
```

output:

```
Enter the order of the encryption key: 3
Enter each row of the encryption key with space
separated integers
0 2 19
8 21 0
19 4 3
```

```
Would you like to encrypt or decrypt some text? (1 or
2)
1. Encrypt
2. Decrypt
1
What text would you like to encrypt: pen
Your encrypted text is:
DYQ
```

Result:

Thus the program for hill cipher is executed successfully

Experiment-22

Aim: To write a python program for secure hash function-1.

Algorithm:

1. Import the hash library.
2. Get the string.
3. Give the condition for result.
4. Print the hexadecimal equivalent.
5. Print the output.
6. Run the program.

Program:

```
import hashlib
str = "Hello everyone"
result = hashlib.sha256(str.encode())
print("The hexadecimal equivalent of SHA256 is : ")
print(result.hexdigest())
print ("\r")
str = "Hello everyone"
result = hashlib.sha384(str.encode())
print("The hexadecimal equivalent of SHA384 is : ")
print(result.hexdigest())
print ("\r")
```

```
str = "Hello everyone"
result = hashlib.sha224(str.encode())
print("The hexadecimal equivalent of SHA224 is : ")
print(result.hexdigest())
print ("\r")
str ="Hello everyone"
result = hashlib.sha512(str.encode())
print("The hexadecimal equivalent of SHA512 is : ")
print(result.hexdigest())
print ("\r")
str = "Hello everyone"
result = hashlib.sha1(str.encode())
print("The hexadecimal equivalent of SHA1 is : ")
print(result.hexdigest())
```

output:

```
The hexadecimal equivalent of SHA256 is :
341d9445779b19f8ad7bfa93cf22acc2058af13407eafc0
106d675d1fe5bb2b9


The hexadecimal equivalent of SHA384 is :
```

```
e26f3a6db1e03363858321101331fa65952e4d140804d0
1632d7195e98972965ff2cc124fa48ec9ae732cd5afee83
7d1


The hexadecimal equivalent of SHA224 is :
c01eac968690ce5c1184d3b8d9f9ffe25e2e693e9a8a1ba
001f52def


The hexadecimal equivalent of SHA512 is :
79accf9b9877840fc74375259fb93b4cc12023b93d370a
b711a0424d5c5972102797d5f45f821db7bf13bd5bb7f7
cbf40dbffb53305dd83cf78f96b093b7380a
The hexadecimal equivalent of SHA1 is :
64aa4395c9ec959f8616c5bb40ec9b0587b9f80b
```

Result:

Thus the program for secure hash function-1 is executed successfully

Experiment-23

Aim: To write a python program for Data encryption standard (DES).

Algorithm:

1.define the functions.

2.check the loops and conditions.

3. perform the loops and conditions for encryption and decryption.

4.print the results as rounds

5. run the program.

Program:

```
def hex2bin(s):
        mp = {'0': "0000",
                '1': "0001",
                '2': "0010",
                '3': "0011",
                '4': "0100",
                '5': "0101",
                '6': "0110",
                '7': "0111",
                '8': "1000",
                '9': "1001",
                'A': "1010",
                'B': "1011",
                'C': "1100",
                'D': "1101",
                'E': "1110",
                'F': "1111"}
        bin = ""
        for i in range(len(s)):
                bin = bin + mp[s[i]]
        return bin


def bin2hex(s):
        mp = {"0000": '0',
                "0001": '1',
                "0010": '2',
                "0011": '3',
                "0100": '4',
                "0101": '5',
                "0110": '6',
                "0111": '7',
                "1000": '8',
                "1001": '9',
                "1010": 'A',
                "1011": 'B',
                "1100": 'C',
                "1101": 'D',
                "1110": 'E',
                "1111": 'F'}
        hex = ""
        for i in range(0, len(s), 4):
                ch = ""
                ch = ch + s[i]
                ch = ch + s[i + 1]
                ch = ch + s[i + 2]
                ch = ch + s[i + 3]
                hex = hex + mp[ch]

        return hex



def bin2dec(binary):
        binary1 = binary
        decimal, i, n = 0, 0, 0
        while(binary != 0):
                dec = binary % 10
                decimal = decimal + dec * pow(2, i)
                binary = binary//10
                i += 1
        return decimal



def dec2bin(num):
        res = bin(num).replace("0b", "")
        if(len(res) % 4 != 0):
                div = len(res) / 4
                div = int(div)
                counter = (4 * (div + 1)) - len(res)
                for i in range(0, counter):
                    res = '0' + res
        return res
```

```
def permute(k, arr, n):
    permutation = ""
    for i in range(0, n):
        permutation = permutation + k[arr[i] - 1]
    return permutation


def shift_left(k, nth_shifts):
    s = ""
    for i in range(nth_shifts):
        for j in range(1, len(k)):
            s = s + k[j]
        s = s + k[0]
        k = s
        s = ""
    return k
```

```
def xor(a, b):
    ans = ""
    for i in range(len(a)):
        if a[i] == b[i]:
            ans = ans + "0"
        else:
            ans = ans + "1"
    return ans

initial_perm = [58, 50, 42, 34, 26, 18, 10, 2,
                60, 52, 44, 36, 28, 20, 12, 4,
                62, 54, 46, 38, 30, 22, 14, 6,
                64, 56, 48, 40, 32, 24, 16, 8,
                57, 49, 41, 33, 25, 17, 9, 1,
                59, 51, 43, 35, 27, 19, 11, 3,
                61, 53, 45, 37, 29, 21, 13, 5,
                63, 55, 47, 39, 31, 23, 15, 7]

exp_d = [32, 1, 2, 3, 4, 5, 4, 5,
        6, 7, 8, 9, 8, 9, 10, 11,
        12, 13, 12, 13, 14, 15, 16, 17,
          16, 17, 18, 19, 20, 21, 20, 21,
          22, 23, 24, 25, 24, 25, 26, 27,
          28, 29, 28, 29, 30, 31, 32, 1]

per = [16, 7, 20, 21,
     29, 12, 28, 17,
     1, 15, 23, 26,
     5, 18, 31, 10,
     2, 8, 24, 14,
     32, 27, 3, 9,
     19, 13, 30, 6,
     22, 11, 4, 25]

sbox = [[[14, 4, 13, 1, 2, 15, 11, 8, 3, 10, 6, 12, 5, 9, 0,
7],
          [0, 15, 7, 4, 14, 2, 13, 1, 10, 6, 12, 11, 9, 5, 3,
8],
          [4, 1, 14, 8, 13, 6, 2, 11, 15, 12, 9, 7, 3, 10, 5,
0],
          [15, 12, 8, 2, 4, 9, 1, 7, 5, 11, 3, 14, 10, 0, 6,
13]],
        [[15, 1, 8, 14, 6, 11, 3, 4, 9, 7, 2, 13, 12, 0, 5,
10],
        [3, 13, 4, 7, 15, 2, 8, 14, 12, 0, 1, 10, 6, 9, 11,
5],
        [0, 14, 7, 11, 10, 4, 13, 1, 5, 8, 12, 6, 9, 3, 2,
15],
        [13, 8, 10, 1, 3, 15, 4, 2, 11, 6, 7, 12, 0, 5, 14,
9]],

        [[10, 0, 9, 14, 6, 3, 15, 5, 1, 13, 12, 7, 11, 4, 2,
8],
        [13, 7, 0, 9, 3, 4, 6, 10, 2, 8, 5, 14, 12, 11, 15,
1],
        [13, 6, 4, 9, 8, 15, 3, 0, 11, 1, 2, 12, 5, 10, 14,
7],
        [1, 10, 13, 0, 6, 9, 8, 7, 4, 15, 14, 3, 11, 5, 2,
12]],

        [[7, 13, 14, 3, 0, 6, 9, 10, 1, 2, 8, 5, 11, 12, 4,
15],
        [13, 8, 11, 5, 6, 15, 0, 3, 4, 7, 2, 12, 1, 10, 14,
9],
        [10, 6, 9, 0, 12, 11, 7, 13, 15, 1, 3, 14, 5, 2, 8,
4],
        [3, 15, 0, 6, 10, 1, 13, 8, 9, 4, 5, 11, 12, 7, 2,
14]],

        [[2, 12, 4, 1, 7, 10, 11, 6, 8, 5, 3, 15, 13, 0, 14,
9],
        [14, 11, 2, 12, 4, 7, 13, 1, 5, 0, 15, 10, 3, 9, 8,
6],
        [4, 2, 1, 11, 10, 13, 7, 8, 15, 9, 12, 5, 6, 3, 0,
14],
        [11, 8, 12, 7, 1, 14, 2, 13, 6, 15, 0, 9, 10, 4, 5,
3]],

        [[12, 1, 10, 15, 9, 2, 6, 8, 0, 13, 3, 4, 14, 7, 5,
11],
        [10, 15, 4, 2, 7, 12, 9, 5, 6, 1, 13, 14, 0, 11, 3,
8],
        [9, 14, 15, 5, 2, 8, 12, 3, 7, 0, 4, 10, 1, 13, 11,
6],
        [4, 3, 2, 12, 9, 5, 15, 10, 11, 14, 1, 7, 6, 0, 8,
13]],

        [[4, 11, 2, 14, 15, 0, 8, 13, 3, 12, 9, 7, 5, 10, 6,
1],
          [13, 0, 11, 7, 4, 9, 1, 10, 14, 3, 5, 12, 2, 15, 8,
6],
          [1, 4, 11, 13, 12, 3, 7, 14, 10, 15, 6, 8, 0, 5, 9,
2],
          [6, 11, 13, 8, 1, 4, 10, 7, 9, 5, 0, 15, 14, 2, 3,
12]],

          [[13, 2, 8, 4, 6, 15, 11, 1, 10, 9, 3, 14, 5, 0, 12,
7],
          [1, 15, 13, 8, 10, 3, 7, 4, 12, 5, 6, 11, 0, 14, 9,
2],
          [7, 11, 4, 1, 9, 12, 14, 2, 0, 6, 10, 13, 15, 3, 5,
8],
          [2, 1, 14, 7, 4, 10, 8, 13, 15, 12, 9, 0, 3, 5, 6,
11]]]

final_perm = [40, 8, 48, 16, 56, 24, 64, 32,
              39, 7, 47, 15, 55, 23, 63, 31,
              38, 6, 46, 14, 54, 22, 62, 30,
              37, 5, 45, 13, 53, 21, 61, 29,
              36, 4, 44, 12, 52, 20, 60, 28,
              35, 3, 43, 11, 51, 19, 59, 27,
              34, 2, 42, 10, 50, 18, 58, 26,
                33, 1, 41, 9, 49, 17, 57, 25]


def encrypt(pt, rkb, rk):
        pt = hex2bin(pt)
        pt = permute(pt, initial_perm, 64)
        print("After initial permutation", bin2hex(pt))

        left = pt[0:32]
        right = pt[32:64]
        for i in range(0, 16):

                right_expanded = permute(right, exp_d, 48)
                xor_x = xor(right_expanded, rkb[i])
                sbox_str = ""
                for j in range(0, 8):
                        row = bin2dec(int(xor_x[j * 6] + xor_x[j *
6 + 5]))
                        col = bin2dec(
                                int(xor_x[j * 6 + 1] + xor_x[j * 6 + 2]
+ xor_x[j * 6 + 3] + xor_x[j * 6 + 4]))
                        val = sbox[j][row][col]
                sbox_str = sbox_str + dec2bin(val)
            sbox_str = permute(sbox_str, per, 32)
            result = xor(left, sbox_str)
            left = result
            if(i != 15):
                left, right = right, left
            print("Round ", i + 1, " ", bin2hex(left),
                " ", bin2hex(right), " ", rk[i])
        combine = left + right
        cipher_text = permute(combine, final_perm, 64)
        return cipher_text
pt = "123456ABCD132536"
key = "AABB09182736CCDD"
key = hex2bin(key)
keyp = [57, 49, 41, 33, 25, 17, 9,
            1, 58, 50, 42, 34, 26, 18,
            10, 2, 59, 51, 43, 35, 27,
            19, 11, 3, 60, 52, 44, 36,
            63, 55, 47, 39, 31, 23, 15,
            7, 62, 54, 46, 38, 30, 22,
            14, 6, 61, 53, 45, 37, 29,
            21, 13, 5, 28, 20, 12, 4]
key = permute(key, keyp, 56)
shift_table = [1, 1, 2, 2,
               2, 2, 2, 2,
               1, 2, 2, 2,
               2, 2, 2, 1]
key_comp = [14, 17, 11, 24, 1, 5,
            3, 28, 15, 6, 21, 10,
            23, 19, 12, 4, 26, 8,
            16, 7, 27, 20, 13, 2,
            41, 52, 31, 37, 47, 55,
            30, 40, 51, 45, 33, 48,
            44, 49, 39, 56, 34, 53,
            46, 42, 50, 36, 29, 32]
left = key[0:28]
right = key[28:56]
rkb = []
rk = []
for i in range(0, 16):
    left = shift_left(left, shift_table[i])
    right = shift_left(right, shift_table[i])
    combine_str = left + right
    round_key = permute(combine_str, key_comp, 48)
    rkb.append(round_key)
    rk.append(bin2hex(round_key))
print("Encryption")
cipher_text = bin2hex(encrypt(pt, rkb, rk))
print("Cipher Text : ", cipher_text)
print("Decryption")
rkb_rev = rkb[::-1]
rk_rev = rk[::-1]
text = bin2hex(encrypt(cipher_text, rkb_rev, rk_rev))
print("Plain Text : ", text)
```

output:

```
Encryption
After initial permutation 14A7D67818CA18AD
Round  1   18CA18AD   5A78E394   194CD072DE8C
Round  2   5A78E394   4A1210F6   4568581ABCCE
Round  3   4A1210F6   B8089591   06EDA4ACF5B5
Round  4   B8089591   236779C2   DA2D032B6EE3
Round  5   236779C2   A15A4B87   69A629FEC913
```

Round 6 A15A4B87 2E8F9C65 C1948E87475E

Round 7 2E8F9C65 A9FC20A3 708AD2DDB3C0

Round 8 A9FC20A3 308BEE97 34F822F0C66D

Round 9 308BEE97 10AF9D37 84BB4473DCCC

Round 10 10AF9D37 6CA6CB20 02765708B5BF

Round 11 6CA6CB20 FF3C485F 6D5560AF7CA5

Round 12 FF3C485F 22A5963B C2C1E96A4BF3

Round 13 22A5963B 387CCDAA 99C31397C91F

Round 14 387CCDAA BD2DD2AB
251B8BC717D0

Round 15 BD2DD2AB CF26B472
3330C5D9A36D

Round 16 19BA9212 CF26B472 181C5D75C66D

Cipher Text : C0B7A8D05F3A829C

Decryption

After initial permutation 19BA9212CF26B472

Round 1 CF26B472 BD2DD2AB 181C5D75C66D

Round 2 BD2DD2AB 387CCDAA
3330C5D9A36D

Round 3 387CCDAA 22A5963B 251B8BC717D0

Round 4 22A5963B FF3C485F 99C31397C91F

Round 5 FF3C485F 6CA6CB20 C2C1E96A4BF3

```
Round  6   6CA6CB20   10AF9D37   6D5560AF7CA5
Round  7   10AF9D37   308BEE97   02765708B5BF
Round  8   308BEE97   A9FC20A3   84BB4473DCCC
Round  9   A9FC20A3   2E8F9C65   34F822F0C66D
Round  10   2E8F9C65   A15A4B87
708AD2DDB3C0
Round  11   A15A4B87   236779C2   C1948E87475E
Round  12   236779C2   B8089591   69A629FEC913
Round  13   B8089591   4A1210F6   DA2D032B6EE3
Round  14   4A1210F6   5A78E394   06EDA4ACF5B5
Round  15   5A78E394   18CA18AD   4568581ABCCE
Round  16   14A7D678   18CA18AD
194CD072DE8C
Plain Text :  123456ABCD132536
```

Result:

Thus the program for Data encryption standard(DES) is executed successfully

Experiment-24

Aim: To write a python program for hill cipher encryption.

Algorithm:

1. Import the string
2. Define the key and function.
3. Check the loop conditions.
4. Check the range of the k
5. Print the matrix.
6. Encrypt the message
7. Print the result.

Program:

```
import string
main=string.ascii_lowercase
def generate_key(n,s):
    s=s.replace(" ","")
    s=s.lower()
    key_matrix=['' for i in range(n)]
    i=0;j=0
    for c in s:
        if c in main:
            key_matrix[i]+=c
            j+=1
            if(j>n-1):
                i+=1
                j=0
    print("The key matrix "+"("+str(n)+'x'+str(n)+") is:")
    print(key_matrix)
    key_num_matrix=[]
    for i in key_matrix:
        sub_array=[]
        for j in range(n):
            sub_array.append(ord(i[j])-ord('a'))
        key_num_matrix.append(sub_array)
    for i in key_num_matrix:
        print(i)
    return(key_num_matrix)
def message_matrix(s,n):
    s=s.replace(" ","")
    s=s.lower()
    final_matrix=[]
    if(len(s)%n!=0):
        while(len(s)%n!=0):
            s=s+'z'
    print("Converted plain_text for encryption: ",s)
    for k in range(len(s)//n):
        message_matrix=[]
        for i in range(n):
            sub=[]
            for j in range(1):
                sub.append(ord(s[i+(n*k)])-ord('a'))
            message_matrix.append(sub)
        final_matrix.append(message_matrix)
    print("The column matrices of plain text in numbers 
are:  ")
    for i in final_matrix:
        print(i)
    return(final_matrix)
def getCofactor(mat, temp, p, q, n):
    i = 0
    j = 0
    for row in range(n):
        for col in range(n):
            if (row != p and col != q) :
                temp[i][j] = mat[row][col]
                j += 1
                if (j == n - 1):
                    j = 0
                    i += 1
```

```
def determinantOfMatrix(mat, n):
    D = 0
    if (n == 1):
        return mat[0][0]
    temp = [[0 for x in range(n)]
             for y in range(n)]
    sign = 1
    for f in range(n):
        getCofactor(mat, temp, 0, f, n)
        D += (sign * mat[0][f] *
            determinantOfMatrix(temp, n - 1))
        sign = -sign
    return D
def isInvertible(mat, n):
    if (determinantOfMatrix(mat, n) != 0):
        return True
    else:
        return False
def multiply_and_convert(key,message):
    res_num = [[0 for x in range(len(message[0]))] for y
in range(len(key))]
```

```
    for i in range(len(key)):
        for j in range(len(message[0])):
            for k in range(len(message)):
                res_num[i][j]+=key[i][k] * message[k][j]
    res_alpha = [['' for x in range(len(message[0]))] for y
in range(len(key))]
    for i in range(len(key)):
        for j in range(len(message[0])):
            res_alpha[i][j]+=chr((res_num[i][j]%26)+97)
    return(res_alpha)
n=int(input("What will be the order of square matrix:
"))
s=input("Enter the key: ")
key=generate_key(n,s)
if (isInvertible(key, len(key))):
    print("Yes it is invertable and can be decrypted")
else:
    print("No it is not invertable and cannot be
decrypted")
plain_text=input("Enter the message: ")
message=message_matrix(plain_text,n)
final_message=''
```

```
for i in message:
    sub=multiply_and_convert(key,i)
    for j in sub:
        for k in j:
            final_message+=k
print("plain message: ",plain_text)
print("final encrypted message: ",final_message)
```

output:

```
What will be the order of square matrix: 3
Enter the key: gybnqkurp
The key matrix (3x3) is:
['gyb', 'nqk', 'urp']
[6, 24, 1]
[13, 16, 10]
[20, 17, 15]
Yes it is invertable and can be decrypted
Enter the message: act
Converted plain_text for encryption:  act
The column matrices of plain text in numbers are:
[[0], [2], [19]]
plain message:  act
```

```
final encrypted message:  qrt
```

Result:

Thus the program for hill cipher encryption is executed successfully

Experiment-25

Aim: To write a python program for hill cipher decryption.

Algorithm:

1. Import the string
2. Define the key and function.
3. Check the loop conditions.
4. Check the range of the k
5. Print the matrix.
6. decrypt the message
7. Print the result.

```
import string
import numpy as np

main=string.ascii_lowercase

def generate_key(n,s):
    s=s.replace(" ","")
    s=s.lower()

    key_matrix=['' for i in range(n)]
    i=0;j=0
    for c in s:
        if c in main:
            key_matrix[i]+=c
            j+=1
            if(j>n-1):
                i+=1
            j=0
    print("The key matrix "+"("+str(n)+'x'+str(n)+")
is:")
    print(key_matrix)

    key_num_matrix=[]
    for i in key_matrix:
        sub_array=[]
        for j in range(n):
            sub_array.append(ord(i[j])-ord('a'))
        key_num_matrix.append(sub_array)

    for i in key_num_matrix:
        print(i)
    return(key_num_matrix)


def modInverse(a, m) :
    a = a % m;
    for x in range(1, m) :
        if ((a * x) % m == 1) :
            return x
    return 1

def method(a, m) :
    if(a>0):
        return (a%m)
    else:
        k=(abs(a)//m)+1
    return method(a+k*m,m)


def message_matrix(s,n):
    s=s.replace(" ","")
    s=s.lower()
    final_matrix=[]
    if(len(s)%n!=0):
        # may be negative also
        for i in range(abs(len(s)%n)):
            # z is the bogus word
            s=s+'z'
    print("Converted cipher_text for decryption: ",s)
    for k in range(len(s)//n):
        message_matrix=[]
        for i in range(n):
            sub=[]
            for j in range(1):
                sub.append(ord(s[i+(n*k)])-ord('a'))
            message_matrix.append(sub)
        final_matrix.append(message_matrix)
    print("The column matrices of plain text in
numbers are:  ")
    for i in final_matrix:
        print(i)
    return(final_matrix)
```

```
def multiply_and_convert(key,message):

    # multiplying matrices
    # resultant must have:
    # rows = numbers of rows in message matrix
    # columns = number of columns in key matrix
    res_num = [[0 for x in range(len(message[0]))]
for y in range(len(key))]

    for i in range(len(key)):
        for j in range(len(message[0])):
            for k in range(len(message)):
                # resulted number matrix
                res_num[i][j]+=key[i][k] * message[k][j]

    res_alpha = [['' for x in range(len(message[0]))]
for y in range(len(key))]
    # getting the alphabets from the numbers
    # according to the logic of hill ciipher
    for i in range(len(key)):
        for j in range(len(message[0])):
            # resultant alphabet matrix
res_alpha[i][j]+=chr((res_num[i][j]%26)+97)

    return(res_alpha)
```

```
n=int(input("What will be the order of square
matrix: "))
s=input("Enter the key: ")
key_matrix=generate_key(n,s)
A = np.array(key_matrix)
det=np.linalg.det(A)
adjoint=det*np.linalg.inv(A)

if(det!=0):
   convert_det=modInverse(int(det),26)
   adjoint=adjoint.tolist()
   print("Adjoint Matrix before modulo26
operation: ")
   for i in adjoint:
      print(i)
   print(convert_det)

   # applying modulo 26 to all elements in adjoint
matrix
   for i in range(len(adjoint)):
      for j in range(len(adjoint[i])):
         adjoint[i][j]=round(adjoint[i][j])
         adjoint[i][j]=method(adjoint[i][j],26)
   print("Adjoint Matrix after modulo26 operation:
")
   for i in adjoint:
      print(i)
```

```
    # modulo is applied to inverse of determinant
and
    # multiplied to all elements in the adjoint matrix
    # to form inverse matrix
    adjoint=np.array(adjoint)
    inverse=convert_det*adjoint

    inverse=inverse.tolist()
    for i in range(len(inverse)):
        for j in range(len(inverse[i])):
            inverse[i][j]=inverse[i][j]%26

    print("Inverse matrix after applying modulo26
operation: ")
    for i in inverse:
        print(i)

    cipher_text=input("Enter the cipher text: ")
    message=message_matrix(cipher_text,n)
    plain_text=''
    for i in message:
        sub=multiply_and_convert(inverse,i)
        for j in sub:
            for k in j:
                plain_text+=k

    print("plain message: ",plain_text)
```

```
        else:
            print("Matrix cannot be inverted")
```

output:

```
What will be the order of square matrix: 3
Enter the key: gybnqkurp
The key matrix (3x3) is:
['gyb', 'nqk', 'urp']
[6, 24, 1]
[13, 16, 10]
[20, 17, 15]
Adjoint Matrix before modulo26 operation:
[70.00000000000003, -343.0000000000002,
224.00000000000014]
[4.999999999999991, 70.00000000000006, -
47.00000000000036]
[-99.00000000000003, 378.00000000000017, -
216.00000000000009]
25
Adjoint Matrix after modulo26 operation:
[18, 21, 16]
[5, 18, 5]
[5, 14, 18]
```

```
Inverse matrix after applying modulo26 operation:
[8, 5, 10]
[21, 8, 21]
[21, 12, 8]
Enter the cipher text: qrt
Converted cipher_text for decryption:  qrt
The column matrices of plain text in numbers are:
[[16], [17], [19]]
plain message:  act
```

Result:

Thus the program for hill cipher decryption is executed successfully