NAME:-PREKSHA ASHOK PATEL

SAPID:-60004210126

BRANCH:-COMPUTER ENGINEERING

DIV:-C2 ; BATCH:-1

INFORMATION SECURITY

Experiment No. 1 – Playfair cipher

Code:

```
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:
    if c not in result:
        if c=='J':
            result.append('I')
        else:
            result.append(c)
flag=0
for i in range(65,91):
    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)
for i in range(0,5):
    for j in range(0,5):
        my_matrix[i][j]=result[k]
        k+=1

def locindex(c):
    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():
    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[(loc
1[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():
    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: keyword

 1.Encryption
 2.Decryption:
 3.EXIT
1
ENTER MSG: hello world
CIPHER TEXT:  GY IZ SC OK CF BU
 1.Encryption
 2.Decryption:
 3.EXIT
2
ENTER CIPHER TEXT: gyizscokcfbu
PLAIN TEXT:  HE LX LO WO RL DX
 1.Encryption
 2.Decryption:
 3.EXIT
3
```

Experiment No. 2 – Vignere Cipher

Code:

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

def originalText(cipher_text, key):
    orig_text = []
    for i in range(len(cipher_text)):
        x = (ord(cipher_text[i]) -
             ord(key[i]) + 26) % 26
        x += ord('A')
        orig_text.append(chr(x))
    return("" . join(orig_text))

string = "hello"
keyword = "key"
key = generateKey(string, keyword)
cipher_text = cipherText(string,key)
print("Ciphertext :", cipher_text)
print("Original/Decrypted Text :",
originalText(cipher_text, key))
```

Output:

```
Ciphertext : DUVHE
Original/Decrypted Text : NKRRU
```

Experiment No. 3 – Vernam Cipher

Code:

```
def char_to_num(char):
    return ord(char) - ord('A')

def num_to_char(num):
    return chr(num + ord('A'))

def vernam_encrypt(plaintext, key):
    # Ensure the key length matches the plaintext length
    if len(key) != len(plaintext):
        raise ValueError("Key length must match plaintext length")

    ciphertext = ""
    for p, k in zip(plaintext, key):
        p_num = char_to_num(p)
        k_num = char_to_num(k)
        encrypted_num = (p_num + k_num) % 26
        ciphertext += num_to_char(encrypted_num)

    return ciphertext

def vernam_decrypt(ciphertext, key):
    # Ensure the key length matches the ciphertext length
    if len(key) != len(ciphertext):
        raise ValueError("Key length must match ciphertext length")

    plaintext = ""
    for c, k in zip(ciphertext, key):
        c_num = char_to_num(c)
        k_num = char_to_num(k)
        decrypted_num = (c_num - k_num) % 26
        plaintext += num_to_char(decrypted_num)

    return plaintext

def main():
    plaintext = input("Enter plaintext: ").upper()
    key = input("Enter key (must be the same length as plaintext): ").upper()

    ciphertext = vernam_encrypt(plaintext, key)
    print("Encrypted:", ciphertext)

    decrypted_text = vernam_decrypt(ciphertext, key)
    print("Decrypted:", decrypted_text)

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

Output:

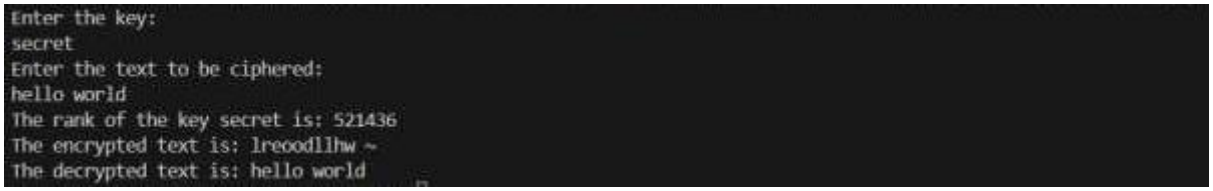
```
Enter plaintext: oak
Enter key (must be the same length as plaintext): son
Encrypted: GOX
Decrypted: OAK
```

Experiment No. 4 – Columnar Cipher

Code:

```
import java.util.*;

class ColumnCipher {
    StringBuilder sb = new StringBuilder();

    String rank(String key) {
        sb.setLength(0);
        char[] ch = key.toLowerCase().toCharArray();
        Arrays.sort(ch);
        for (int i = 0; i < key.length(); i++) {
            for (int j = 0; j < key.length(); j++) {
                if (key.toLowerCase().charAt(i) == ch[j]) {
                    ch[j] = '~';
                    sb.append(j + 1);
                    break;
                }
            }
        }
        return sb.toString();
    }

    ArrayList<ArrayList<Character>> encryptMatrix(String text, int n) {
        ArrayList<ArrayList<Character>> arrayList = new ArrayList<>();
        ArrayList<Character> array;
        int row = (int) Math.ceil((double) text.length() / n);
        for (int i = 0; i < row; i++) {
            array = new ArrayList<>();
            for (int j = 0; j < n; j++) {
                if (i * n + j < text.length()) {
                    array.add(text.charAt(i * n + j));
                }
            }
            arrayList.add(array);
        }
        while (arrayList.get(arrayList.size() - 1).size() < n) {
            arrayList.get(arrayList.size() - 1).add('~');
        }
        return arrayList;
    }

    String encrypt(String rank, String text) {
        Map<Integer, Integer> map = new HashMap<>();
        ArrayList<ArrayList<Character>> arrayList =
encryptMatrix(text.toLowerCase(), rank.length());
        sb.setLength(0);
```

Output:


Enter the key:
secret
Enter the text to be ciphered:
hello world
The rank of the key secret is: 521436
The encrypted text is: lreoodllhw ~
The decrypted text is: hello world

Experiment No. 5 – RSA

Code:

```
import random as r
import numpy as np

def modInverse(e, phin):
    for d in range(1, phin):
        if (((e % phin) * (d % phin)) % phin == 1):
            return d
    return -1

def prime(a):
    count = 0
    for i in range(2, int(a/2)):
        if a % i == 0:
            count += 1
    if count == 0:
        return True
    else:
        return False

cond = 1
count = 0
while cond:
    p = r.randint(2, 255)
    if prime(p):
        break

cond = 1
while cond:
    q = r.randint(2, 255)
    if prime(q) and q != p:
        break

n = (p * q)
phin = (p - 1) * (q - 1)

cond = 1
while cond:
    e = r.randint(2, phin)
    if prime(e) and e != p and e != q:
        break

d = modInverse(e, phin)
while d == -1:
    cond = 1
    temp = e
```

```
    while cond:
        e = r.randint(2, phin)
        if prime(e) and e != p and e != q and e != temp:
            break
        d = modInverse(e, phin)

print(f"p - {p}, q - {q}, e - {e}, d - {d}")

publicKey = (e, n)
privateKey = (d, n)
print(f"Public key - {publicKey}")
print(f"Private key - {privateKey}")
```

Output:

```
p - 179, q - 131, e - 1621, d - 10221
Public key - (1621, 23449)
Private key - (10221, 23449)
```

Experiment No. 6 – Diffie Hellman

Code:

Alice.py:-

```
import socket
import random as r

def alice():
    host = socket.gethostname()
    port = 5000
    s = socket.socket()
    s.bind((host, port))
    s.listen(2)
    conn, address = s.accept()
    print("Connection from: " + str(address))

    p = int(input("Enter p = "))
    g = int(input("Enter g = "))
    conn.send(str(p).encode('ascii'))
    conn.send(str(g).encode('ascii'))

    a = r.randint(3, 1000)
    Xa = int(pow(g, a, p))
    print("Xa computed = ", Xa)
    conn.send(str(Xa).encode('ascii'))

    Xb = int(conn.recv(1024).decode('ascii'))
    print("Xb from Bob = ", Xb)

    Ak = int(pow(Xb, a, p))
    print('Secret key for Alice is = %d' % (Ak))
    conn.close()

alice()
```

bob.py :

```
import socket
import random as r

def bob():
    host = socket.gethostname()
    port = 5000
```

```
    s = socket.socket()
    s.connect((host, port))

    p = int(s.recv(1024).decode('ascii'))
    print("p = ", p)
    g = int(s.recv(1024).decode('ascii'))
    print("g = ", g)
    Xa = int(s.recv(1024).decode('ascii'))
    print("Xa from Man in the middle = ", Xa)

    b = r.randint(3, 1000)
    Xb = int(pow(g, b, p))
    print("Xb computed = ", Xb)
    s.send(str(Xb).encode('ascii'))

    Bk = int(pow(Xa, b, p))
    print('Secret key for Bob is = %d' % (Bk))
    s.close()

bob()
```

Output:

```
Connection from: ('192.168.29.66', 58415)
Enter p = 23
Enter g = 9
Xa computed =  18
Xb from Bob =  13
Secret key for Alice is = 16
```

```
p =  23
g =  9
Xa from Man in the middle =  18
Xb computed =  13
Secret key for Bob is = 16
```

Experiment No. 7 – MD5

Code:

```
from hashlib import sha256

def hash(x):
    ans = sha256(x.encode("utf8")).hexdigest()
    return ans

def hash_value(h):
    h1 = []
    if len(h) % 2 == 0:
        for i in range(0, len(h), 2):
            text = h[i] + h[i + 1]
            h1.append(hash(text))
    else:
        for i in range(0, len(h) - 1, 2):
            text = h[i] + h[i + 1]
            h1.append(hash(text))
        h1.append(h[len(h) - 1])
    return h1

para = input("Enter para (use '.' to seperate lines): ")
l = para.split('.')
count = len(l)

if count % 8 != 0:
    temp = int(count / 8)
    for i in range(0, (temp + 1) * 8 - count):
        l.append(l[count - 1])

h = list(map(hash, l))
length = len(h)

while length > 1:
    h = hash_value(h)
    length = len(h)

print("\n\nHash - ", h[0])
```

Output:

```
Enter para (use '.' to seperate lines): hello.how.are.you.today.is.a.cloudy.day


Hash -  8b4af9abfd1d5ed929b3922a2b6bb218202edff0dc6459da97f4895b937826b7
```

Experiment No. 8 – RSA signature

Code:

```
def euclid(m, n):

    if n == 0:
        return m
    else:
        r = m % n
        return euclid(n, r)

def exteuclid(a, b):

    r1 = a
    r2 = b
    s1 = int(1)
    s2 = int(0)
    t1 = int(0)
    t2 = int(1)

    while r2 > 0:

        q = r1//r2
        r = r1-q * r2
        r1 = r2
        r2 = r
        s = s1-q * s2
        s1 = s2
        s2 = s
        t = t1-q * t2
        t1 = t2
        t2 = t

    if t1 < 0:
        t1 = t1 % a

    return (r1, t1)

p = 823
q = 953
n = p * q
Pn = (p-1)*(q-1)
```

```
key = []

for i in range(2, Pn):

    gcd = euclid(Pn, i)

    if gcd == 1:
        key.append(i)

e = int(313)

r, d = exteuclid(Pn, e)
if r == 1:
    d = int(d)
    print("decryption key is: ", d)

else:
    print("Multiplicative inverse for\
    the given encryption key does not \
    exist. Choose a different encryption key ")


# Enter the message to be sent
M = 19070

# Signature is created by Alice
S = (M**d) % n

# Alice sends M and S both to Bob
# Bob generates message M1 using the
# signature S, Alice's public key e
# and product n.
M1 = (S**e) % n

# If M = M1 only then Bob accepts
# the message sent by Alice.

if M == M1:
    print("As M = M1, Accept the\
    message sent by Alice")
else:
    print("As M not equal to M1,\
    Do not accept the message\
    sent by Alice ")
```

Output:

```
decryption key is:  160009
As M = M1, Accept the   message sent by Alice
```