Experiment 1 - Playfair Cipher

```
pt = input("Enter plaintext : ").lower().replace("j", "i")
key = input("Enter key : ").lower().replace("j", "i")

m = []

for el in key:
    if el not in m:
        m.append(el)

for i in range(97, 123):
  if chr(i) not in m and chr(i)!='j' and len(m)!=25:
    m.append(chr(i))

matrix=[]
for i in range(0,len(m), 5):
    matrix.append(m[i:i+5])

pairs=[]
if len(pt)%2!=0:
  for i in range(0,len(pt)-1,2):
    if pt[i]!=pt[i+1]:
      pairs.append(pt[i:i+2])
    else:
      pairs.append(pt[i] + 'x')
      pairs.append(pt[i] + 'x')
  pairs.append(pt[-1] + 'x')
else:
  for i in range(0,len(pt),2):
    if pt[i]!=pt[i+1]:
      pairs.append(pt[i:i+2])
    else:
      pairs.append(pt[i] + 'x')
      pairs.append(pt[i] + 'x')

ct=[]
for p in pairs:
    ind1=[0,0]
    ind2=[0,0]
    for j in range(len(matrix)):
        if p[0] in matrix[j]:
            ind1=[j, matrix[j].index(p[0])]
```

```
        if p[1] in matrix[j]:
            ind2=[j, matrix[j].index(p[1])]
    if ind1[0] == ind2[0]:
        ct.append(matrix[ind1[0]][(ind1[1]+1)%5] +
matrix[ind2[0]][(ind2[1]+1)%5])
    elif ind1[1] == ind2[1]:
        ct.append(matrix[(ind1[0]+1)%5][ind1[1]] +
matrix[(ind2[0]+1)%5][ind2[1]])
    else:
        ct.append(matrix[ind1[0]][ind2[1]] + matrix[ind2[0]][ind1[1]])

print(''.join(ct))
```

```
PS C:\Users\Admin\OneDrive\Desktop\DJ\SEM6_Pracs\IS_final> py .\PlayfairCipher.py
Enter plaintext : instruments
Enter key : monarchy
gatlmzclrqxa
```

Experiment 2 - Vigenere Cipher

```
# len(key) != len(plainText), we repeat characters of key
pt = input("Enter plaintext : ").lower().replace(" ","")
key = input("Enter key : ").lower().replace(" ","")

tempKey = key
i=0
while(len(tempKey)!= len(pt)):
    tempKey+=key[i]
    i += 1
    i %= len(key)

o = ""
for i in range(len(pt)):
    n1 = ord(pt[i])- ord('a')
    n2 = ord(tempKey[i])- ord('a')
    sum = (n1 + n2) % 26
    o+=chr(sum + ord('a'))

print(o)
```

```
PS C:\Users\Admin\OneDrive\Desktop\DJ\SEM6_Pracs\IS_final> py .\VignereCipher.py
Enter plaintext : cryptography
Enter key : security
uvajkwzpstjs
```

Experiment 3 - Vernam Cipher

```python
# len(key) == len(plainText)
pt = input("Enter plaintext : ").lower().replace(" ","")
key = input("Enter key : ").lower().replace(" ","")
o = ""

if(len(key)!= len(pt)):
    print("Length of plain text and key should be of same length")
else:
    for i in range(len(pt)):
        b1 = bin(ord(pt[i]))
        b2 = bin(ord(key[i]))
        val = int(b1, 2) ^ int(b2, 2)
        o += chr(val)
    print(o)
```

```
PS C:\Users\Admin\OneDrive\Desktop\DJ\SEM6_Pracs\IS_final> py .\VernamCipher.py
Enter plaintext : cryptography
Enter key : security
Length of plain text and key should be of same length
PS C:\Users\Admin\OneDrive\Desktop\DJ\SEM6_Pracs\IS_final> py .\VernamCipher.py
Enter plaintext : qwertyu
Enter key : asdfghj
►♦8¶‼◄▼
```

Experiment 4 - Columnar Transposition Cipher

```
import math

def encryptMessage(msg):
    cipher = ""
    k_indx = 0
    msg_len = float(len(msg))
    msg_lst = list(msg)
    key_lst = sorted(list(key))
    col = len(key)
    row = int(math.ceil(msg_len / col))
    fill_null = int((row * col) - msg_len)
    msg_lst.extend('_' * fill_null)
    matrix = [msg_lst[i: i + col]  for i in range(0, len(msg_lst),
col)]

    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

pt = input("Enter plaintext : ").lower().replace(" ","")
key = input("Enter key : ").lower().replace(" ","")
ct = encryptMessage(pt)
print("Encrypted Message: {}".format(ct))
```

```
PS C:\Users\Admin\OneDrive\Desktop\DJ\SEM6_Pracs\IS_final> py .\ColumnarTranspositionCipher.py
Enter plaintext : abchsfre
Enter key : djwf
Encrypted Message: ashebfcr
```

Experiment 5 - RSA Algorithm

```python
import math

def gcd(a, h):
    temp = 0
    while(1):
        temp = a % h
        if (temp == 0):
            return h
        a = h
        h = temp

p = 3
q = 7
n = p*q
e = 2
phi = (p-1)*(q-1)

while (e < phi):
    if(gcd(e, phi) == 1):
        break
    else:
        e = e+1

k = 2
d = (1 + (k*phi))/e
msg = 12.0
print("Message data = ", msg)

c = pow(msg, e)
c = math.fmod(c, n)
print("Encrypted data = ", c)

m = pow(c, d)
m = math.fmod(m, n)
print("Original Message Sent = ", m)
```

```
PS C:\Users\Admin\OneDrive\Desktop\DJ\SEM6_Pracs\IS_final> py '.\RSA Algorithm.py'
Message data =  12.0
Encrypted data =  3.0
Original Message Sent =  12.0
```

Experiment 6 - Diffie-Hellman

```
def prime_checker(p):
    if p < 1:
        return -1
    elif p > 1:
        if p == 2:
            return 1
        for i in range(2, p):
            if p % i == 0:
                return -1
            return 1


def primitive_check(g, p, L):
    for i in range(1, p):
        L.append(pow(g, i) % p)
    for i in range(1, p):
        if L.count(i) > 1:
            L.clear()
            return -1
        return 1

l = []
while 1:
    P = int(input("Enter P : "))
    if prime_checker(P) == -1:
        print("Number Is Not Prime, Please Enter Again!")
        continue
    break

while 1:
    G = int(input(f"Enter The Primitive Root Of {P} : "))
    if primitive_check(G, P, l) == -1:
        print(f"Number Is Not A Primitive Root Of {P}, Please Try
Again!")
        continue
    break

# Private Keys
x1, x2 = int(input("Enter The Private Key Of User 1 : ")), int(
    input("Enter The Private Key Of User 2 : "))
while 1:
    if x1 >= P or x2 >= P:
```

```
        print(f"Private Key Of Both The Users Should Be Less Than
{P}!")
        continue
    break

y1, y2 = pow(G, x1) % P, pow(G, x2) % P

k1, k2 = pow(y2, x1) % P, pow(y1, x2) % P

print(f"\nSecret Key For User 1 Is {k1}\nSecret Key For User 2 Is
{k2}\n")

if k1 == k2:
    print("Keys Have Been Exchanged Successfully")
else:
    print("Keys Have Not Been Exchanged Successfully")
```

```
PS C:\Users\Admin\OneDrive\Desktop\DJ\SEM6_Pracs\IS_final> py .\DiffieHellman.py
Enter P : 7
Enter The Primitive Root Of 7 : 9
Number Is Not A Primitive Root Of 7, Please Try Again!
Enter The Primitive Root Of 7 : 3
Enter The Private Key Of User 1 : 6
Enter The Private Key Of User 2 : 5

Secret Key For User 1 Is 1
Secret Key For User 2 Is 1

Keys Have Been Exchanged Successfully
```

Experiment 7 - MD5 Algorithm

```python
# importing the required libraries
import hashlib
# making a message
inputstring = "This is a message sent by a computer user."
# encoding the message using the library function
output = hashlib.md5(inputstring.encode())
# printing the hash function
print("Hash of the input string:")
print(output.hexdigest())
```

```
PS C:\Users\Admin\OneDrive\Desktop\DJ\SEM6_Pracs\IS_final> py '.\MD5 Algorithm.py'
Hash of the input string:
922547e866c89b8f677312df0ccec8ee
```

Experiment 8 - RSA Signature

```
def euclid(m, n):
    if n == 0:
        return m
    else:
        r = m % n
        return euclid(n, r)
# Program to find Multiplicative inverse
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)
# Enter two large prime numbers p and q
p = 823
q = 953
n = p * q
Pn = (p-1)*(q-1)
# Generate encryption key in range 1<e<Pn
key = []
for i in range(2, Pn):
    gcd = euclid(Pn, i)
    if gcd == 1:
        key.append(i)
# Select an encryption key from the above list
e = int(313)
# Obtain inverse of encryption key in Z_Pn
```

```
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 ")
```

```
PS C:\Users\Admin\OneDrive\Desktop\DJ\SEM6_Pracs\IS_final> py '.\RSA Signature.py'
decryption key is:  160009
As M = M1, Accept the\message sent by Alice
```