PRACTICAL-4

AIM:

RSA algorithm is used by Salim to transfer session key to Anarkali. He suspects that Akbar is performing man in middle attack he chose to use 1024 bit prime numbers. Hint: you may choose to use big integer in java

THEORY:

- The **RSA algorithm** is an asymmetric cryptography algorithm; this means that it uses a public key and a private key (i.e two different, mathematically linked keys).
- As their names suggest, a public key is shared publicly, while a private key is secret and must not be shared with anyone.
- The RSA algorithm is named after those who invented it in 1978: Ron Rivest, Adi Shamir, and Leonard Adleman.
- It is safe for exchange of data over internet.

ALGORITHM:

- 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

STEPS:

1. Generate RSA Modulus
2. Find derived number (e)
3. Derive Private key and Public Key
4. Encryption

NOTE: We will also use Euler Totient, Multiplicative inverse and some other algorithms to make the calculation.

ADVANTAGES:

- RSA algorithm is safe and secure for transmitting confidential data.
- Cracking RSA algorithm is very difficult as it involves complex mathematics.
- Sharing public key to users is easy.

DISADVANTAGES:

- It has slow data transfer rate due to large numbers involved.
- It requires third party to verify the reliability of public keys sometimes.
- It requires some complex calculations both for encryption and decryption, which sometimes takes time and can causes delay and on the receiver's side, calculations need to be done again.

PROGRAM CODE:

LANGUAGE OF CODE: Python

```
# CS 345 - CRNS

#PRACTICAL-4

#PERFORMED BY : PARTH N PATEL

# ID : 19DCS098


#LANGUAGE OF CODE: PYTHON


#IMPORTING math LIBRARY

import math




#TAKING INPUT FOR THE PRIME NUMBERS

print("PLEASE SELECT TWO PRIME NUMBERS: ")

p = int(input("ENTER THE FIRST PRIME NUMBER : "))

print()

q = int(input("ENTER THE SECOND PRIME NUMBER : "))
```

```
print()


#VALIDATING WHETHER THE NUMBERS ARE PRIME OR NOT

#FUNCTION TO VALIDATE IF NUMBER IS PRIME OR NOT


def validate_Prime(x):
    if(x==2):
        #2 IS THE ONLY EVEN NUMBER
        return True
    elif((x<2) or ((x%2)==0)):
        #NUMBERS LESS THAN 2 AND DIVISBLE BY ANOTHER NUMBER
        # ARE NOT PRIME NUMBERS
        return False
    elif(x>2):
        for i in range(2,x):
            if not(x%i):
                return False
    return True


#CECKING IF P AND Q ARE PRIME OR NOT
```

```
check_p = validate_Prime(p)

check_q = validate_Prime(q)



#LOOP TILL PRIME NUMBERS ARE ENTERED

while(((check_p==False)or(check_q==False))):

    p = int(input("Enter a prime number for p: "))

    q = int(input("Enter a prime number for q: "))

    check_p = validate_Prime(p)

    check_q = validate_Prime(q)



#MAIN LOGIC BUILDING FOR THE RSA



#FINDING THE RSA MODULUS

n = p * q

print("RSA MODULUS : ",n)

print()



#EULER'S TOITENT

r= (p-1)*(q-1)

print("EULER'S TOITENT : ",r)
```

```
print()


#CALCULATING THE GCD
def find_GCD(e,r):
    while(r!=0):
        e,r=r,e%r
    return e


#EUCLID'S ALGORITHM


def euclid_Algorithm(e,r):
    for i in range(1,r):
        while(e!=0):
            a,b=r//e,r%e
            if(b!=0):
                print("%d = %d*(%d) + %d"%(r,a,e,b))
            r=e
            e=b


#EXTENDED EUCLID'S ALGORITHM
```

```
def extended_Euclid_Algorithm(a,b):
    if(a%b==0):
        return(b,0,1)
    else:
        gcd,s,t = extended_Euclid_Algorithm(b,a%b)
        s = s-((a//b) * t)
        print("%d = %d*(%d) + (%d)*(%d)"%(gcd,a,t,s,b))
        return(gcd,t,s)


#MULTIPLICATIVE INVERSE
def multiplicative_Inverse(e,r):
    gcd,s,_=extended_Euclid_Algorithm(e,r)
    if(gcd!=1):
        return None
    else:
        if(s<0):
            print("s=%d. SINCE %d IS LESS THAN 0, s = s(modr), i.e., s=%d."%(s,s,s%r))
        elif(s>0):
            print("s=%d."%(s))
        return s%r
```

```
#CALCULATING THE VALUE OF e

#FINDS THE HIGHEST POSSIBLE VALUE OF 'e' BETWEEN 1 and 1000 THAT MAKES (e,r) COPRIME.

for i in range(1,1000):

    if(find_GCD(i,r)==1):

        e=i

print("THE VALUE OF e : ",e)

print()



#CALCULATION OF 'd', PRIVATE KEY, AND PUBLIC KEY.

print("EUCLID'S ALGORITHM:")

euclid_Algorithm(e,r)

print()

print("EUCLID'S EXTENDED ALGORITHM:")

d = multiplicative_Inverse(e,r)

print()

print("THE VALUE OF D : ",d)

print()

#PUBLIC KEY
```

```
public = (e,n)

#PRIVATE KEY

private = (d,n)



print("PRIVATE KEY : ",private)

print("PUBLIC KEY  : ",public)

print()




#ENCRYPTION

def encrypt(pub_key,n_text):

    e,n=pub_key

    x=[]

    m=0

    for i in n_text:

        if(i.isupper()):

            m = ord(i)-65

            c=(m**e)%n

            x.append(c)

        elif(i.islower()):
```

```
            m= ord(i)-97
            c=(m**e)%n
            x.append(c)
        elif(i.isspace()):
            spc=400
            x.append(400)
    return x



#Message
plain_Text = input("PLEASE ENTER THE MESSAGE TO BE ENCRYPTED : ")
print()
print("YOUR MESSAGE :",plain_Text)


cipher_Text=encrypt(public,plain_Text)
print()
print("THE ENCRYPTED CIPHER TEXT : ")
print()
```

```
print(cipher_Text)

print()



print("PARTH PATEL\n19DCS098")
```

OUTPUT:

OUTPUT FOR BIG PRIME NUMBERS:

```
C:\0_SEM_6\1_CS_345_CRYPTOGRAPHY_AND_NETWORK_SECURITY\3_CODES>python -u "c:\(
CURITY\3_CODES\CRNS_Practical_4.py"
PLEASE SELECT TWO PRIME NUMBERS:
ENTER THE FIRST PRIME NUMBER : 6029

ENTER THE SECOND PRIME NUMBER : 9973

RSA MODULUS :  60127217

EULER'S TOITENT :  60111216

THE VALUE OF e :  997

EUCLID'S ALGORITHM:
60111216 = 60292*(997) + 92
997 = 10*(92) + 77
92 = 1*(77) + 15
77 = 5*(15) + 2
15 = 7*(2) + 1

EUCLID'S EXTENDED ALGORITHM:
1 = 15*(1) + (-7)*(2)
1 = 77*(-7) + (36)*(15)
1 = 92*(36) + (-43)*(77)
1 = 997*(-43) + (466)*(92)
1 = 60111216*(466) + (-28096115)*(997)
1 = 997*(-28096115) + (466)*(60111216)
s=-28096115. SINCE -28096115 IS LESS THAN 0, s = s(modr), i.e., s=32015101.
```

```
THE VALUE OF D :  32015101

PRIVATE KEY :  (32015101, 60127217)
PUBLIC KEY  :  (997, 60127217)

PLEASE ENTER THE MESSAGE TO BE ENCRYPTED : HELLO ANARKALI I AM SALIM

YOUR MESSAGE : HELLO ANARKALI I AM SALIM

THE ENCRYPTED CIPHER TEXT :

[10988007, 15673550, 13428330, 13428330, 51206376, 400, 0, 41511227, 0, 22705414, 37489707, 0, 13428330, 44472872, 400,
44472872, 400, 0, 40831172, 400, 7305973, 0, 13428330, 44472872, 40831172]

PARTH PATEL
19DCS098
```

OUTPUT FOR SMALL PRIME NUMBERS:

```
C:\0_SEM_6\1_CS_345_CRYPTOGRAPHY_AND_NETWORK_SECURITY\3_CODES>p
CURITY\3_CODES\CRNS_Practical_4.py"
PLEASE SELECT TWO PRIME NUMBERS:
ENTER THE FIRST PRIME NUMBER : 29

ENTER THE SECOND PRIME NUMBER : 53

RSA MODULUS :  1537

EULER'S TOITENT :  1456

THE VALUE OF e :  999

EUCLID'S ALGORITHM:
1456 = 1*(999) + 457
999 = 2*(457) + 85
457 = 5*(85) + 32
85 = 2*(32) + 21
32 = 1*(21) + 11
21 = 1*(11) + 10
11 = 1*(10) + 1

EUCLID'S EXTENDED ALGORITHM:
1 = 11*(1) + (-1)*(10)
1 = 21*(-1) + (2)*(11)
1 = 32*(2) + (-3)*(21)
1 = 85*(-3) + (8)*(32)
1 = 457*(8) + (-43)*(85)
1 = 999*(-43) + (94)*(457)
1 = 1456*(94) + (-137)*(999)
1 = 999*(-137) + (94)*(1456)
s=-137. SINCE -137 IS LESS THAN 0, s = s(modr), i.e., s=1319.
```

```
THE VALUE OF D :  1319

PRIVATE KEY :  (1319, 1537)
PUBLIC KEY  :  (999, 1537)

PLEASE ENTER THE MESSAGE TO BE ENCRYPTED : HELLO ANARKALI I AM SALIM

YOUR MESSAGE : HELLO ANARKALI I AM SALIM

THE ENCRYPTED CIPHER TEXT :

[1524, 96, 537, 537, 300, 400, 0, 122, 0, 1230, 1210, 0, 537, 31, 400, 31, 400, 0, 1380, 400, 101, 0, 537, 31, 1380]

PARTH PATEL
19DCS098
```

CONCLUSION:

By performing the above practical, I learned the basic concept of RSA algorithm, why it is extensively used and how the encryption process works.