Experiment 8

Aakriti Singh

20190802006

AIM: Write a C/C++/Python/JAVA program to implement RSA algorithm.

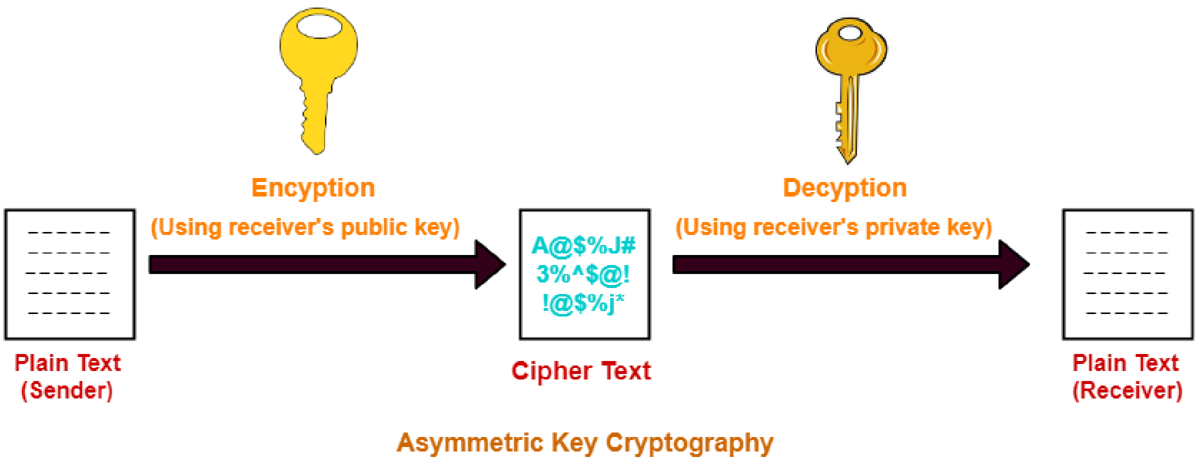
THEORY:

I. Introduction to RSA algorithm:

RSA algorithm is a public key encryption technique and is considered as the most secure way of encryption. It was invented by Rivest, Shamir and Adleman in year 1978 and hence name RSA algorithm.

The RSA algorithm holds the following features –

- 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


Encyption
(Using receiver's public key)
Decyption
(Using receiver's private key)
A@$%J#
3%^$@!
!@$%j*
Plain Text
(Sender)
Cipher Text
Plain Text
(Receiver)
Asymmetric Key Cryptography

II. Steps to implement RSA algorithm:

We require the following steps to work on RSA algorithm -

Step 1: Generate the RSA modulus

The initial procedure begins with selection of two prime numbers namely p and q, and then calculating their product N, as shown -

N = p*a

Here, let N be the specified large number.

Step 2: Derived Number (e)

Consider number e as a derived number which should be greater than 1 and less than (p-1) and (q-1). The primary condition will be that there should be no common factor of (p-1) and (0-1) except 1

Step 3: Public key

The specified pair of numbers n and e forms the RSA public key and it is made public.

Step 4: Private Key

Private Key d is calculated from the numbers p. q and e. The mathematical relationship between the numbers is as follows -

ed = 1 mod (p-1) (q-1)

The above formula is the basic formula for Extended Euclidean Algorithm, which takes p and q as the input parameters

Encryption Formula

Consider a sender who sends the plain text message to someone whose public key is **(n, e)**. To encrypt the plain text message in the given scenario, use the following syntax -

C = Pe mod n

Decryption Formula

The decryption process is very straightforward and includes analytics for calculation in a systematic approach. Considering receiver C has the private key d. the result modulus will be calculated as -

Plaintext = Cd mod n

Let-

- Public key of the receiver =le, n)
- Private key of the receiver = (d.n)

Then, RSA Algorithm works in the following steps -

Step-01:

At sender side,

- Sender represents the message to be sent as an integer between 0 and n-1.
- Sender encrypts the message using the public key of receiver
- It raises the plain text message "P" to the e^{th} power modulo n.
- This converts the message into cipher text C'.

$$\boxed{CP = P^e \bmod n}$$

Step-02:

- The cipher text 'C' is sent to the receiver over the communication channel

Step-03:

At receiver side,

- Receiver decrypts the cipher text using his private key.
- It raises the cipher text to the d" power modulo n.
- This converts the cipher text back into the plain text ‘P’.

$$\boxed{P = C^d \bmod n}$$

Steps to Generate Public Key And Private Key:

An individual can generate his public key and private key using the following steps-

Step-01:

Choose any two prime numbers p and q such that

- They are different
- They are very largo

Step-02:

Calculate 'n' and toilent function ∅(n) where –

- n = p x q
- ∅(n) = (p-1) x (q-1)

Step-03:

Choose any value of 'e' such that -

- 1<e< Ø(n)
- gcd (e, Ø (n)) = 1

Step-04:

Determined such that

$\boxed{ed = 1 \bmod Ø(n)}$

OR

$\boxed{d = \frac{1 + k\, Ø(n)}{e}}$

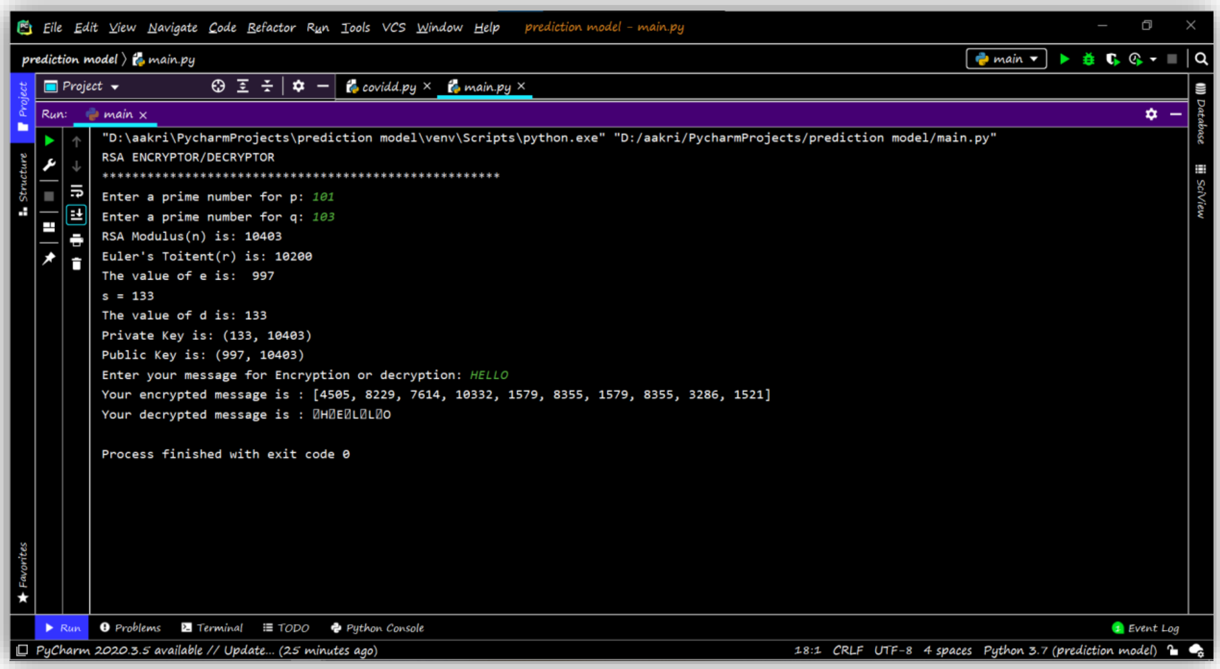
- We already know the value of 'e' and Ø(n)
- Choose the least positive integer value of k which gives the integer value of ‘d’ as a result.
- Use trial and error method.
- Start substituting different values of k from 0.

PROGRAM:

```
print("RSA ENCRYPTOR/DECRYPTOR")
print("*****************************************************")
```

```python
# Input Prime Number
p = int(input("Enter a prime number for p: "))
q = int(input("Enter a prime number for q: "))


# Check if Input's are Prime
def prime_check(a):
    if a == 2:
        return True
    elif (a < 2) or (a % 2 == 0):
        return False
    elif a > 2:
        for i in range(2, a):
            if not (a % i):
                return False
    return True


check_p = prime_check(p)
check_q = prime_check(q)
while check_p is False or check_q is False:
    p = int(input("Enter a prime number for p: "))
    q = int(input("Enter a prime number for q: "))
    check_p = prime_check(p)
    check_q = prime_check(q)

# RSA Modulus
n = p * q
print("RSA Modulus(n) is:", n)

# Euler's Toitent
r = (p - 1) * (q - 1)
print("Euler's Toitent(r) is:", r)


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


# Euclid's Algorithm
def eugcd(e, r):
    for i in range(1, r):
        while e != 0:
            a, b = r // e, r % e
            r = e
            e = b


# Extended Euclidean Algorithm
def ExtendedEuclideanAlgorithm(a, b):
    if a % b == 0:
        return b, 0, 1
    else:
        gcd, s, t = ExtendedEuclideanAlgorithm(b, a % b)
        s = s - ((a // b) * t)
        return gcd, t, s


# Multiplicative Inverse
def MultiplicativeInverse(e, r):
    gcd, s, _ = ExtendedEuclideanAlgorithm(e, r)
    if gcd != 1:
        return None
    else:
        if s < 0:
            print("s =", s, ". Since", s, "is less than 0, s = (modr), i.e., s =",
s % r)
        elif s > 0:
            print("s =", s,)
        return s % r


# Value Calculation
for i in range(1, 1000):
    if GCD(i, r) == 1:
        e = i
print("The value of e is: ", e)

# print("EUCLID'S ALGORITHM:")
eugcd(e, r)
d = MultiplicativeInverse(e, r)
print("The value of d is:", d)
public = (e, n)
private = (d, n)
print("Private Key is:", private)
print("Public Key is:", public)


# Encryption
def encrypt(pub_key, encmsg):
    e, n = pub_key
    x = []
    m = 0
    for letter in encmsg:
        m = ord(letter)
        c = (m ** e) % n
        x.append(c)
        if letter.isupper():
            m = ord(letter) - 65
            c = (m**e) % n
            x.append(c)
        elif letter.islower():
            m = ord(letter) - 97
            c = (m**e) % n
            x.append(c)
        elif letter.isspace():
            c = 400
            x.append(400)
    return x


# Decryption
def decrypt(priv_key, msg):
    d, n = priv_key
    txt = msg.split(',')
    x = ''
    m = 0
    for number in txt:
        if number == '400':
            x += ' '
        else:
            m = (int(number)**d) % n
            m += 65
            c = chr(m)
                        x += c
    return x


# Driver Code
message = input("Enter your message for Encryption or decryption: ")
enc_msg = encrypt(public, message)
print("Your encrypted message is :", enc_msg)
dec_msg = str(enc_msg)[1:len(str(enc_msg))-1]
print("Your decrypted message is :", decrypt(private, dec_msg))
```

OUTPUT:



```
"D:\aakri\PycharmProjects\prediction model\venv\Scripts\python.exe" "D:/aakri/PycharmProjects/prediction model/main.py"
RSA ENCRYPTOR/DECRYPTOR
*****************************************************
Enter a prime number for p: 101
Enter a prime number for q: 103
RSA Modulus(n) is: 10403
Euler's Toitent(r) is: 10200
The value of e is:  997
s = 133
The value of d is: 133
Private Key is: (133, 10403)
Public Key is: (997, 10403)
Enter your message for Encryption or decryption: HELLO
Your encrypted message is : [4505, 8229, 7614, 10332, 1579, 8355, 1579, 8355, 3286, 1521]
Your decrypted message is : ▯H▯E▯L▯L▯O

Process finished with exit code 0
```

File Edit View Navigate Code Refactor Run Tools VCS Window Help
prediction model – main.py
prediction model 〉 main.py
Project
covidd.py
main.py
Run: main
Run Problems Terminal TODO Python Console Event Log
PyCharm 2020.3.5 available // Update... (25 minutes ago)
18:1 CRLF UTF-8 4 spaces Python 3.7 (prediction model)

RESULT: RSA algorithm for cryptography is implemented.