

Ex.No.4
10/08/24

IMPLEMENTATION OF RSA

AIM:

To simulate the working of RSA in Virtual lab environment and to implement the same in Java/Python

THEORY:

One way function:

A one-way function is a function that is easy to compute in one direction, yet difficult to reverse. In RSA, the multiplication of two large prime numbers p and q to generate n is a one-way function.

Key generation:

Key generation is the process of generating keys in cryptography. A key is used to encrypt and decrypt whatever data is being encrypted/decrypted.

$$\phi(n) = (p-1) \times (q-1)$$

RSA Encryption:

Encryption is performed by raising the plaintext message M to the power of e (the public key exponent) and then taking the modulus of n . The ciphertext C is computed as:

$$C = M^e \bmod n$$

RSA Decryption:

Decryption is the reverse process of encryption. The ciphertext C is raised to the power of d (the private key exponent) and then taken modulus n . The decrypted message M is computed as:

$$M = C^d \bmod n$$

ALGORITHM:

Key generation:

1. Select two large prime numbers, p and q .

2. Multiply these numbers to find $n = p \times q$, where n is called the modulus for encryption and decryption.
3. Choose a number e less than n , such that n is relatively prime to $(p - 1) \times (q - 1)$. It means that e and $(p - 1) \times (q - 1)$ have no common factor except 1. Choose " e " such that $1 < e < \phi(n)$, e is prime to $\phi(n)$ i.e $\gcd(e, n) = 1$,
4. Repeat step 3 until the condition is satisfied.
5. To determine the private key, we use the following formula to calculate the d such that:

$$D * e \bmod \phi(n) = 1$$

6. The private key is (d, n) .
7. A ciphertext message c is decrypted using private key (d, n) .
8. To calculate plain text m from the ciphertext c following formula is used to get plain text m .

$$M = C^d \bmod n$$

RSA Encryption:

1. Get the Plaintext Input from the user.
2. Use the receiver's public key to encrypt the message, using the formula

$$C = M^e \bmod n$$

where, e is a random long integer, n is a product of two large primes and M is the message to be encrypted.

3. Return the Ciphertext.

RSA Decryption:


1. Get the Ciphertext from the user.
2. Use the receiver's secret key to decrypt the message using the formula

$$M = C^d \bmod n$$

3. Where d is the inverse of e with respect to $\phi(n)$ n is a product of two large primes and C is the message to be decrypted.

Screen Shots of simulation in Virtual labs

Encryption and Decryption


Public-Key Cryptosystems (PKCSv1.5)
★★★★☆
Rate Me
Report a Bug

Plaintext (string):

Hello

encrypt

Ciphertext (hex):

```
377f4d4f5cd2fa12750a4841cdccab5bff51b7df472a550426c5cc9788f4a587
9a9a78c8e891a5ee7ddb6c945c0eadaaaf08c783f750a28ded964eb3077f61e
873426ffa4d8d07208370e2a57df56801c9ce842ae7a4515e4eedfabb114416
cd020496a19b2d18a358049315fe967629e2d5d7f85e371914721e676383c4a5
```

decrypt

Decrypted Plaintext (string):

Hello

Status:

Decryption Time: 8ms

RSA private key

1024 bit 1024 bit (e=3) 512 bit 512 bit (e=3) Generate bits = 512

Modulus (hex):

```
a5261939975948bb7a58dfef5ff54e65f0498f917f5a09288810b8975871e99
af3b5d4d057b0fc07535f97444504fa35169d461d0d30cf0192e307727c06
5168c788771c561a9400fb49175e9e6aa4e23fe11af69e9412dd23b0cb6684c4
c2429bce139e848ab26d0829073351f4acd36074eafd036a5eb83359d2a698d3
```

Public exponent (hex, F4=0x10001):

10001

Key Generation:

Public exponent (hex, F4=0x10001):

10001

Private exponent (hex):

```
8e9912fd6d3645894e8d38cb58c0db81ff516cf4c7e5a14c7f1eddb1459d2cded
4d8d293cf97aee6aefb861859c8b6a3d1dfe710463e1f9ddc72048c09751971c
4a580aa51eb523357a3c48d31cfad1d4a165066ed92d4748fb6571211da5cb1
4bc11b6e2df7c1a559e6d5ac1cd5c94703a22891464fba23d0d965086277a161
```

P (hex):

```
d090ce58a92c75233a6486cb0a9209bf3583b64f540c76f5294bb97d285eed33
aec220bde14b2417951178ac152ceab6da7090905b478195498b352048f15e7d
```

Q (hex):

```
cab575dc652bb66df15a0359609d51d1db184750c00c6698b90ef3465c996551
03edb0d54c56aec0ce3c4d22592338092a126a0cc49f65a4a30d22b411e58f
```

D mod (P-1) (hex):

```
1a24bca8e273df2f0e47c199bbf678604e7df7215480c77c8db39f49b000ce2c
f7500038acff5433b7d582a01f1826e6f4d42e1c57f5e1fef7b12aabc59fd25
```

D mod (Q-1) (hex):

```
3d06982efbbe47339e1f6d36b1216b8a741d410b0c662f54f7118b27b9a4ec9d
914337eb39841d8666f3034408cf94f5b62f11c402fc994fe15a05493150d9fd
```

1/Q mod P (hex):

```
3a3e731acd8960b7ff9eb81a7ff93bd1cf74cbd56987db58b4594fb09c09084
db1734c8143f98b602b981aaa9243ca28deb69b5b280ee8dcee0fd2625e53250
```

Coding

```
import java.util.Scanner;
import java.math.BigInteger;
public class RSA {
    public int eulerTotient(int n) {
```

```

int result = n;
for (int p = 2; p * p <= n; p++) {
    if (n % p == 0) {
        while (n % p == 0) {
            n /= p;
        }
        result -= result / p;
    }
}
if (n > 1) {
    result -= result / n;
}
return result;
}

public BigInteger modInverse(BigInteger e, BigInteger phi) {
    return e.modInverse(phi);
}

public BigInteger encrypt(BigInteger M, BigInteger e, BigInteger n) {
    return M.modPow(e, n);
}

public BigInteger decrypt(BigInteger C, BigInteger d, BigInteger n) {
    return C.modPow(d, n);
}

public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    RSA rsa = new RSA();
    System.out.println("Enter prime number p:");
    int p = sc.nextInt();
    System.out.println("Enter prime number q:");
    int q = sc.nextInt();
    int n = p * q;
    int phi = rsa.eulerTotient(p) * rsa.eulerTotient(q);
    BigInteger phiBig = BigInteger.valueOf(phi);
    System.out.println("Enter public exponent e (should be coprime with  $\phi(n)$ ):");
    int e = sc.nextInt();
    BigInteger eBig = BigInteger.valueOf(e);
    BigInteger d = rsa.modInverse(eBig, phiBig);
    System.out.println("Public Key: (e = " + e + ", n = " + n + ")");
}

```

```

System.out.println("Private Key: (d = " + d + ", n = " + n + ")");
System.out.println("Enter the message M (as an integer):");
BigInteger M = sc.nextBigInteger();
BigInteger C = rsa.encrypt(M, eBig, BigInteger.valueOf(n));
System.out.println("Encrypted message C = " + C);
BigInteger decryptedMessage = rsa.decrypt(C, d, BigInteger.valueOf(n));
System.out.println("Decrypted message M = " + decryptedMessage);
}
}

```

SCREEN SHOTS:

User Defined Values

```

Enter prime number p:
13
Enter prime number q:
17
Enter public exponent e (should be coprime with  $\phi(n)$ ):
11

```

Generating Keys:

```

Public Key: (e = 11, n = 221)
Private Key: (d = 35, n = 221)

```

Encrypting and Decrypting Messages:

```

Enter the message M (as an integer):
19
Encrypted message C = 76
Decrypted message M = 19

```

Final Output:

```

C:\Users\gokul\.jdk\openjdk-22.0.2\bin\java.exe "-javaagent
Enter prime number p:
13
Enter prime number q:
17
Enter public exponent e (should be coprime with  $\phi(n)$ ):
11
Public Key: (e = 11, n = 221)
Private Key: (d = 35, n = 221)
Enter the message M (as an integer):
19
Encrypted message C = 76
Decrypted message M = 19

Process finished with exit code 0

```

RESULT:

Thus, the stimulation of RSA in Virtual lab environment and to implement the same in Python has been done successfully.

Evaluation

Parameter	Max Marks	Marks Obtained
Uniqueness of the Code	40	
Completion of experiment on time	5	
Documentation	20	
Simulation in Vlabs	10	
Total	75	
Signature of the faculty with Date		