**IT8761 – Security Laboratory**

**Reshma Ramesh Babu**

**312217104129**

**Exercise 6**

**Aim:** To implement the Rivest-Shamir-Adleman (RSA) Algorithm.

**Code:**

import java.math.BigInteger ;

import java.util.Random ;

import java.io.\*;

public class RSA

{

    // RSA 8 => primeSize 8

    int primeSize ;

    // prime numbers

    BigInteger p, q ;

    // N= pq

    BigInteger N ;

    //r = ( p - 1 ) \* ( q - 1 )

    BigInteger r ;

    BigInteger E, D ;

    public RSA( int primeSize )

    {

        this.primeSize = primeSize ;

        // Generate two distinct large prime numbers p and q.

        generatePrimeNumbers() ;

        // Generate Public and Private Keys.

        generatePublicPrivateKeys() ;

    }

    public void generatePrimeNumbers()

    {

        p = new BigInteger( primeSize, 10, new Random()) ;

        do

        {

            q = new BigInteger( primeSize, 10, new Random()) ;

        }

        while( q.compareTo( p ) == 0 ) ;

    }

    public void generatePublicPrivateKeys()

    {

        // N = p \* q

        N = p.multiply( q ) ;

        // r = ( p - 1 ) \* ( q - 1 )

        r = p.subtract( BigInteger.valueOf( 1 ) ) ;

        r = r.multiply( q.subtract( BigInteger.valueOf( 1 ) ) ) ;

        // Choose E, coprime to and less than r

        do

        {

            E = new BigInteger( 2 \* primeSize, new Random() ) ;

        }

        while( ( E.compareTo( r ) != -1 ) || ( E.gcd( r ).compareTo( BigInteger.valueOf( 1 ) ) != 0 ) ) ;

        // Compute D, the inverse of E mod r

        D = E.modInverse( r ) ;

    }

    public BigInteger[] encrypt( String message )

    {

        int i ;

        byte[] temp = new byte[1] ;

        byte[] digits = message.getBytes() ;

        BigInteger[] bigdigits = new BigInteger[digits.length] ;

        for( i = 0 ; i < bigdigits.length ; i++ )

        {

            temp[0] = digits[i] ;

            bigdigits[i] = new BigInteger( temp ) ;

        }

        BigInteger[] encrypted = new BigInteger[bigdigits.length] ;

        for( i = 0 ; i < bigdigits.length ; i++ )

            encrypted[i] = bigdigits[i].modPow( E, N ) ;

        return( encrypted ) ;

    }

    public String decrypt( BigInteger[] encrypted,BigInteger D,BigInteger N )

    {

        int i ;

        BigInteger[] decrypted = new BigInteger[encrypted.length] ;

        for( i = 0 ; i < decrypted.length ; i++ )

            decrypted[i] = encrypted[i].modPow( D, N ) ;

        char[] charArray = new char[decrypted.length] ;

        for( i = 0 ; i < charArray.length ; i++ )

            charArray[i] = (char) ( decrypted[i].intValue() ) ;

        return( new String( charArray ) ) ;

    }

    public BigInteger getp()

    {

        return( p ) ;

    }

    public BigInteger getq()

    {

        return( q ) ;

    }

    public BigInteger getr()

    {

        return( r ) ;

    }

    public BigInteger getN()

    {

        return( N ) ;

    }

    public BigInteger getE()

    {

        return( E ) ;

    }

    public BigInteger getD()

    {

        return( D ) ;

    }

    public static void main( String[] args ) throws IOException

    {

        int primeSize =8;

        // Generate Public and Private Keys

        RSA rsa = new RSA( primeSize ) ;

        System.out.println( "Key Size: " + primeSize );

        System.out.println( "" ) ;

        System.out.println( "Generated prime numbers p and q" );

        System.out.println( "p: " + rsa.getp().toString( 16 ).toUpperCase()  );

        System.out.println( "q: " + rsa.getq().toString( 16 ).toUpperCase()  );

        System.out.println( "" ) ;

        System.out.println( "The public key is the pair (N, E) which will be published." ) ;

        System.out.println( "N: " + rsa.getN().toString( 16 ).toUpperCase()  ) ;

        System.out.println( "E: " + rsa.getE().toString( 16 ).toUpperCase()  ) ;

        System.out.println( "" ) ;

        int ch;

        BigInteger[] ciphertext;

        // Get message (plaintext) from user

        System.out.println( "Please enter message (plaintext):" ) ;

        String plaintext = ( new BufferedReader( new InputStreamReader( System.in ) ) ).readLine() ;

        System.out.println( "" ) ;

        // Encrypt Message

        ciphertext = rsa.encrypt( plaintext ) ;

        do

        {

            System.out.println("MENU\n\t1.Encrypt\n\t2.Decrypt\n\t3.Exit\n");

            System.out.println("Enter Choice:");

            ch= Integer.parseInt(System.console().readLine());

            if(ch==1)

            {

                System.out.print( "Ciphertext: " ) ;

                for( int i = 0 ; i < ciphertext.length ; i++ )

                {

                    System.out.print( ciphertext[i].toString( 16 ).toUpperCase() ) ;

                    if( i != ciphertext.length - 1 )

                    System.out.print( " " ) ;

                }

                System.out.println( "" ) ;

            }

            else if(ch==2)

            {

                RSA rsa1 = new RSA(8);

                String recoveredPlaintext = rsa1.decrypt( ciphertext ,rsa.getD(),rsa.getN()) ;

                System.out.println( "Recovered plaintext: " + recoveredPlaintext  ) ;

            }

        }while(ch!=3);

    }

}

**Output:**

