**ICS Assignment No: 1**

**S-DES**

**Program Code:**

**import** java.awt.datatransfer.StringSelection;

**import** java.io.DataInputStream;

**public** **class** SDES {

**public** **int** K1,K2;

**public** **static** **final** **int** ***P10***[] = { 3, 5, 2, 7, 4, 10, 1, 9, 8, 6};

**public** **static** **final** **int** ***P10max*** = 10;

//10 to 8 bit permutation

**public** **static** **final** **int** ***P8***[] = { 6, 3, 7, 4, 8, 5, 10, 9};

**public** **static** **final** **int** ***P8max*** = 10;

**public** **static** **final** **int** ***P4***[] = { 2, 4, 3, 1};

**public** **static** **final** **int** ***P4max*** = 4;

//Performing initial permutaion

**public** **static** **final** **int** ***IP***[] = { 2, 6, 3, 1, 4, 8, 5, 7};

**public** **static** **final** **int** ***IPmax*** = 8;

//performing final permuttaion(Inverse of initial permutation)

**public** **static** **final** **int** ***IPI***[] = { 4, 1, 3, 5, 7, 2, 8, 6};

**public** **static** **final** **int** ***IPImax*** = 8;

//expansion/permutation operation:Used in F1 Function

**public** **static** **final** **int** ***EP***[] = { 4, 1, 2, 3, 2, 3, 4, 1};

**public** **static** **final** **int** ***EPmax*** = 4;

**public** **static** **final** **int** ***S0***[][] = {{ 1, 0, 3, 2},{ 3, 2, 1, 0},{ 0, 2, 1,

3},{ 3, 1, 3, 2}};

**public** **static** **final** **int** ***S1***[][] = {{ 0, 1, 2, 3},{ 2, 0, 1, 3},{ 3, 0, 1,

2},{ 2, 1, 0, 3}};

/\*

\* Generic Function to perform all permutations

\*/

**public** **static** **int** permute(**int** x, **int** p[], **int** pmax) {

**int** y=0;

//System.out.println(Integer.toBinaryString(x));

/\*

String binary=Integer.toBinaryString(x);

StringBuffer br=new StringBuffer(new String(new char[10]).replace("\0", "0"));

\*/

**for**(**int** i=0;i<p.length;i++) {

// br.setCharAt(i, binary.charAt(p[i]-1));

y=y<<1;

y=y|(x>>(pmax-p[i]))&1;

}

//System.out.println("Final :"+Integer.toBinaryString(y));

**return** y;

}

**public** **static** **int** F( **int** R, **int** K)

{

**int** t = *permute*( R, ***EP***, ***EPmax***) ^ K;

**int** t0 = (t >> 4) & 0xF;

**int** t1 = t & 0xF;

t0 = ***S0***[ ((t0 & 0x8) >> 2) | (t0 & 1) ][ (t0 >> 1) & 0x3 ];

t1 = ***S1***[ ((t1 & 0x8) >> 2) | (t1 & 1) ][ (t1 >> 1) & 0x3 ];

t = *permute*( (t0 << 2) | t1, ***P4***, ***P4max***);

**return** t;

}

**public** **static** **int** fK( **int** m, **int** K)

{

**int** L = (m >> 4) & 0xF;

**int** R = m & 0xF;

**return** ((L ^ *F*(R,K)) << 4) | R;

}

**public** **static** **int** SW( **int** x)

{

**return** ((x & 0xF) << 4) | ((x >> 4) & 0xF);

}

/\*

\* Ciphertext = IP-1 (fK2 (SW (fk1 (IP (plaintext))))

\*/

**public** **byte** encrypt( **int** m)

{

System.***out***.println("\nEncryption Process Starts........\n\n");

m = *permute*( m, ***IP***, ***IPmax***);

System.***out***.print("\nAfter Initial Permutation(IP) : ");

*printData*( m, 8);

m = *fK*( m, K1);

System.***out***.print("\nbefore Swap : ");

*printData*( m, 8);

m = *SW*( m);

System.***out***.print("\nAfter Swap : ");

*printData*( m, 8);

m = *fK*( m, K2);

System.***out***.print("\nbefore IP inverse : ");

*printData*( m, 8);

m = *permute*( m, ***IPI***, ***IPImax***);

**return** (**byte**) m;

}

/\*

\* Plaintext = IP-1 (fK1 (SW (fk2 (IP (ciphertext)))))

\*/

**public** **byte** decrypt( **int** m)

{

System.***out***.println("\nDecryption Process Starts........\n\n");

*printData*( m, 8);

m = *permute*( m, ***IP***, ***IPmax***);

System.***out***.print("\nAfter Permutation : ");

*printData*( m, 8);

m = *fK*( m, K2);

System.***out***.print("\nbefore Swap : ");

*printData*( m, 8);

m = *SW*( m);

System.***out***.print("\nAfter Swap : ");

*printData*( m, 8);

m = *fK*( m, K1);

System.***out***.print("\nBefore Extraction Permutation : ");

*printData*( m, 4);

m = *permute*( m, ***IPI***, ***IPImax***);

System.***out***.print("\nAfter Extraction Permutation : ");

*printData*( m, 8);

**return** (**byte**) m;

}

**public** **static** **void** printData( **int** x, **int** n)

{

**int** mask = 1 << (n-1);

**while**( mask > 0)

{

System.***out***.print( ((x & mask) == 0) ? '0' : '1');

mask >>= 1;

}

}

/\*

\*

\* K1 = P8 (Shift\_1 (P10 (Key)))

K2 = P8 (Shift\_2 (shift\_1 (P10 (Key))))

\*/

**public** SDES( **int** K) //COnstructor generates key k1 & k2 used for encyption

{

K = *permute*( K, ***P10***, ***P10max***);

**int** t1 = (K >> 5) & 0x1F;

**int** t2 = K & 0x1F;

t1 = ((t1 & 0xF) << 1) | ((t1 & 0x10) >> 4);

t2 = ((t2 & 0xF) << 1) | ((t2 & 0x10) >> 4);

K1 = *permute*( (t1 << 5)| t2, ***P8***, ***P8max***);

t1 = ((t1 & 0x7) << 2) | ((t1 & 0x18) >> 3);

t2 = ((t2 & 0x7) << 2) | ((t2 & 0x18) >> 3);

K2 = *permute*( (t1 << 5)| t2, ***P8***, ***P8max***);

}

@SuppressWarnings("deprecation")

**public** **static** **void** main( String args[]) **throws** Exception

{

DataInputStream inp=**new** DataInputStream(System.***in***);

System.***out***.println("Enter the 10 Bit Key :"); //1011011010

**int** K = Integer.*parseInt*(inp.~~readLine~~(),2);

SDES A = **new** SDES( K);

System.***out***.println("Enter the 8 Bit message To be Encrypt : ");

**int** m = Integer.*parseInt*(inp.~~readLine~~(),2); //10110110

System.***out***.print("\nKey K1: ");

SDES.*printData*( A.K1, 8);

System.***out***.print("\nKey K2: ");

SDES.*printData*( A.K2, 8);

m = A.encrypt( m);

System.***out***.print("\nEncrypted Message: ");

SDES.*printData*( m, 8);

m = A.decrypt( m);

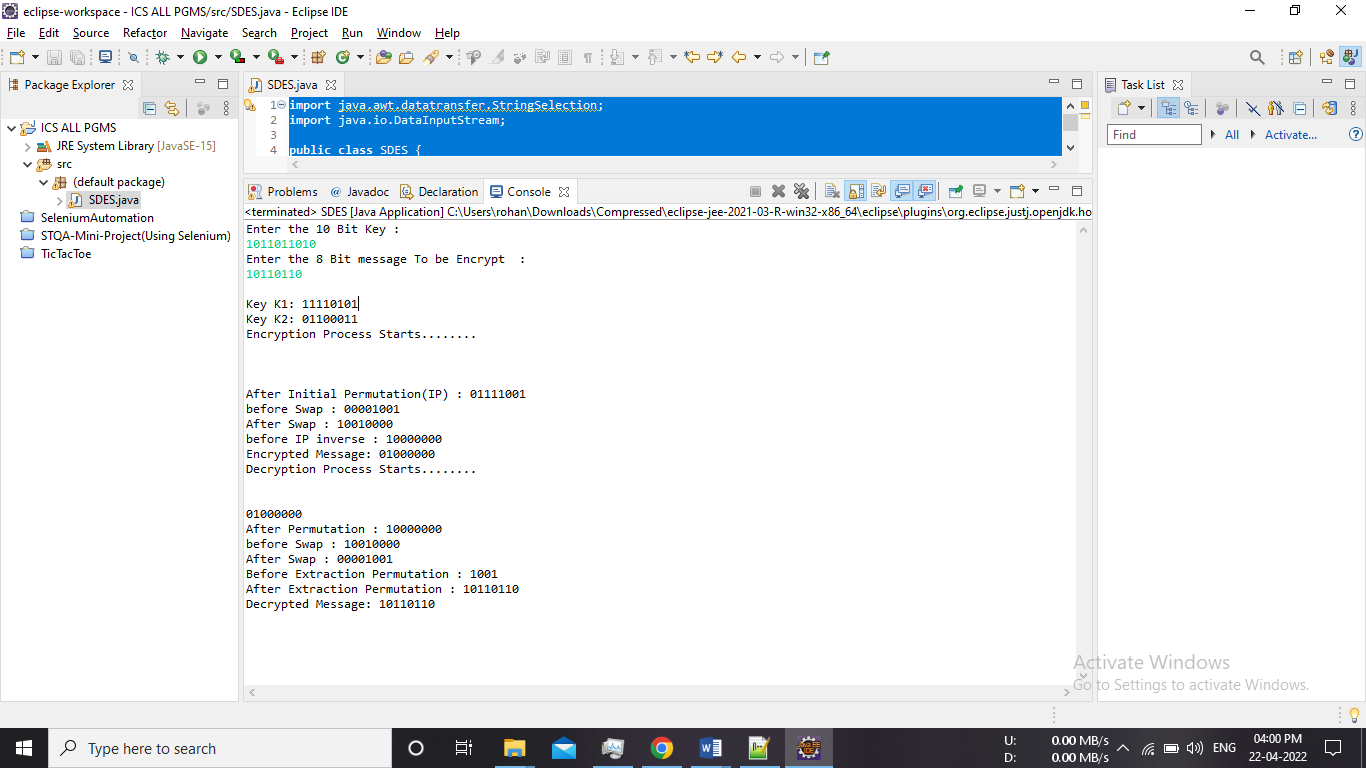
System.***out***.print("\nDecrypted Message: ");

SDES.*printData*( m, 8);

}

}

**Output:**



**ICS Assignment No: 2**

**AES**

**Program Code:**

**package** AES;

**import** java.io.UnsupportedEncodingException;

**import** java.security.MessageDigest;

**import** java.security.NoSuchAlgorithmException;

**import** java.util.Arrays;

**import** java.util.Base64;

**import** javax.crypto.Cipher;

**import** javax.crypto.spec.SecretKeySpec;

**public** **class** AES {

**private** **static** SecretKeySpec *secretKey*;

**private** **static** **byte**[] *key*;

**public** **static** **void** setKey(String myKey) {

MessageDigest sha = **null**;

**try** {

*key* = myKey.getBytes("UTF-8");

sha = MessageDigest.*getInstance*("SHA-1");

*key* = sha.digest(*key*);

*key* = Arrays.*copyOf*(*key*, 16);

*secretKey* = **new** SecretKeySpec(*key*, "AES");

} **catch** (NoSuchAlgorithmException e) {

e.printStackTrace();

} **catch** (UnsupportedEncodingException e) {

e.printStackTrace();

}

}

**public** **static** String encrypt(String strToEncrypt, String secret) {

**try** {

*setKey*(secret);

Cipher cipher = Cipher.*getInstance*("AES/ECB/PKCS5Padding");

cipher.init(Cipher.***ENCRYPT\_MODE***, *secretKey*);

**return**

Base64.*getEncoder*().encodeToString(cipher.doFinal(strToEncrypt.getBytes("UTF-8")));

} **catch** (Exception e) {

System.***out***.println("Error while encrypting: " + e.toString());

}

**return** **null**;

}

**public** **static** String decrypt(String strToDecrypt, String secret) {

**try** {

*setKey*(secret);

Cipher cipher = Cipher.*getInstance*("AES/ECB/PKCS5PADDING");

cipher.init(Cipher.***DECRYPT\_MODE***, *secretKey*);

**return** **new**

String(cipher.doFinal(Base64.*getDecoder*().decode(strToDecrypt)));

} **catch** (Exception e) {

System.***out***.println("Error while decrypting: " + e.toString());

}

**return** **null**;

}

**public** **static** **void** main(String[] args) {

**final** String secretKey = "SPPUUniversity";

String originalString = "www.SPPUuniv.edu";

String encryptedString = AES.*encrypt*(originalString, secretKey);

String decryptedString = AES.*decrypt*(encryptedString, secretKey);

System.***out***.println("URL Encryption Using AES Algorithm\n------------");

System.***out***.println("Original URL : " + originalString);

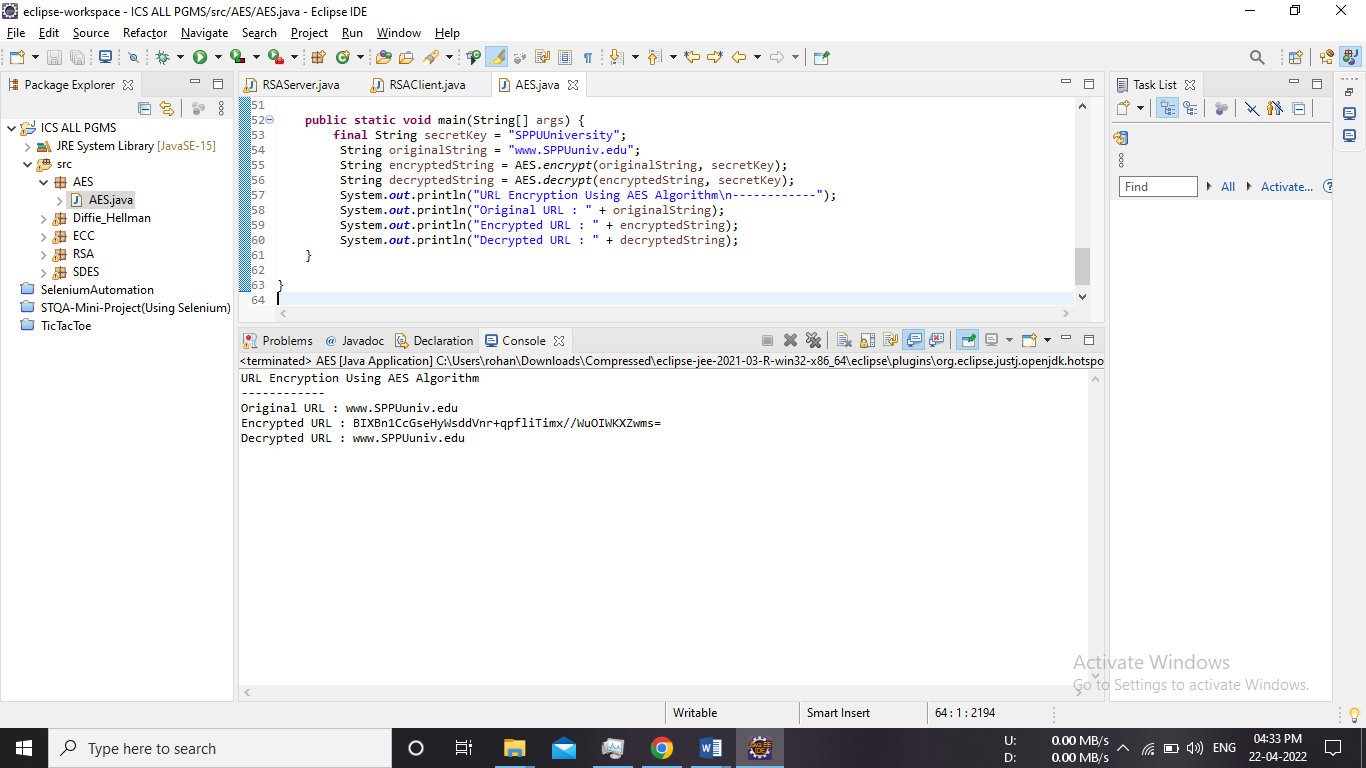
System.***out***.println("Encrypted URL : " + encryptedString);

System.***out***.println("Decrypted URL : " + decryptedString);

}

}

**Output:**



**ICS Assignment No: 3**

**Diffie-Hellman**

**Program Code:**

**DiffiehellmanClient.java:**

**import** java.io.DataInputStream;

**import** java.io.DataOutputStream;

**import** java.io.IOException;

**import** java.math.BigInteger;

**import** java.net.Socket;

**import** java.net.UnknownHostException;

**import** java.util.Scanner;

**public** **class** DiffiehellmanClient {

**private** **static** Socket *soc*;

**private** **static** Scanner *sc*;

**public** **static** **void** main(String[] args) **throws** UnknownHostException, IOException {

**int** p,g;//Two large prime numbers(public keys)

**int** a;//selected private key of user A

//Socket connection code

*soc* = **new** Socket("127.0.0.1",8088);

System.***out***.println("Connected to "+*soc*.getRemoteSocketAddress());

DataInputStream dis=**new** DataInputStream(*soc*.getInputStream());

DataOutputStream dos=**new** DataOutputStream(*soc*.getOutputStream());

*sc* = **new** Scanner(System.***in***);

//Take 2 prime numbers as input

System.***out***.println("Enter first prime Number(P):");

p=*sc*.nextInt();

System.***out***.println("Enter second prime number (G):");

g=*sc*.nextInt();

//send public keys(2 large prime nos) to another user B

dos.writeInt(p);

dos.writeInt(g);

//Select random private key of user A

a=(**int**)Math.*random*()+1;

System.***out***.println("Private Key (A): "+a);

BigInteger x=(BigInteger.*valueOf*(g).pow(a)).mod(BigInteger.*valueOf*(p)); //x=(g^a)mod(p)

System.***out***.println("Generated key (X)= "+x.intValue());

dos.writeInt(x.intValue());

**int** y=dis.readInt(); //received public key from B

System.***out***.println("Received data from server (y) = "+y);

BigInteger secretKey=(BigInteger.*valueOf*(y).pow(a)).mod(BigInteger.*valueOf*(p));

System.***out***.println(" Secret Key: "+secretKey.intValue());

//if secret keys at User A & B are same then communication can be performed.

*soc*.close();

}

}

**DiffiehellmanServer.java:**

**import** java.io.DataInputStream;

**import** java.io.DataOutputStream;

**import** java.io.IOException;

**import** java.math.BigInteger;

**import** java.net.ServerSocket;

**import** java.net.Socket;

**public** **class** DiffieHellmanServer {

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

//Socket connection code (Server)

ServerSocket server=**new** ServerSocket(8088);

System.***out***.println("waitinng for connection on port "+server.getLocalPort());

Socket soc=server.accept();

System.***out***.println("Accepted connection from "+soc.getRemoteSocketAddress());

DataInputStream dis=**new** DataInputStream(soc.getInputStream());

DataOutputStream dos=**new** DataOutputStream(soc.getOutputStream());

//Read data sent by User A

**int** p=dis.readInt();

**int** g=dis.readInt();

**int** x=dis.readInt(); //received public key of A

System.***out***.println("Received Values: \nP = "+p+" \nG = "+g+"\nPublic Key of A (X) = "+x);

//Generate private key b

**int** b=(**int**)Math.*random*()+1;

System.***out***.println("Private Ket of B (b) = "+b);

//y=(g^b)mod(p)

BigInteger y=(BigInteger.*valueOf*(g).pow(b)).mod(BigInteger.*valueOf*(p));

System.***out***.println("Public key of B(y) = "+y);

dos.writeInt(y.intValue());

BigInteger secretKey=(BigInteger.*valueOf*(x).pow(b)).mod(BigInteger.*valueOf*(p));

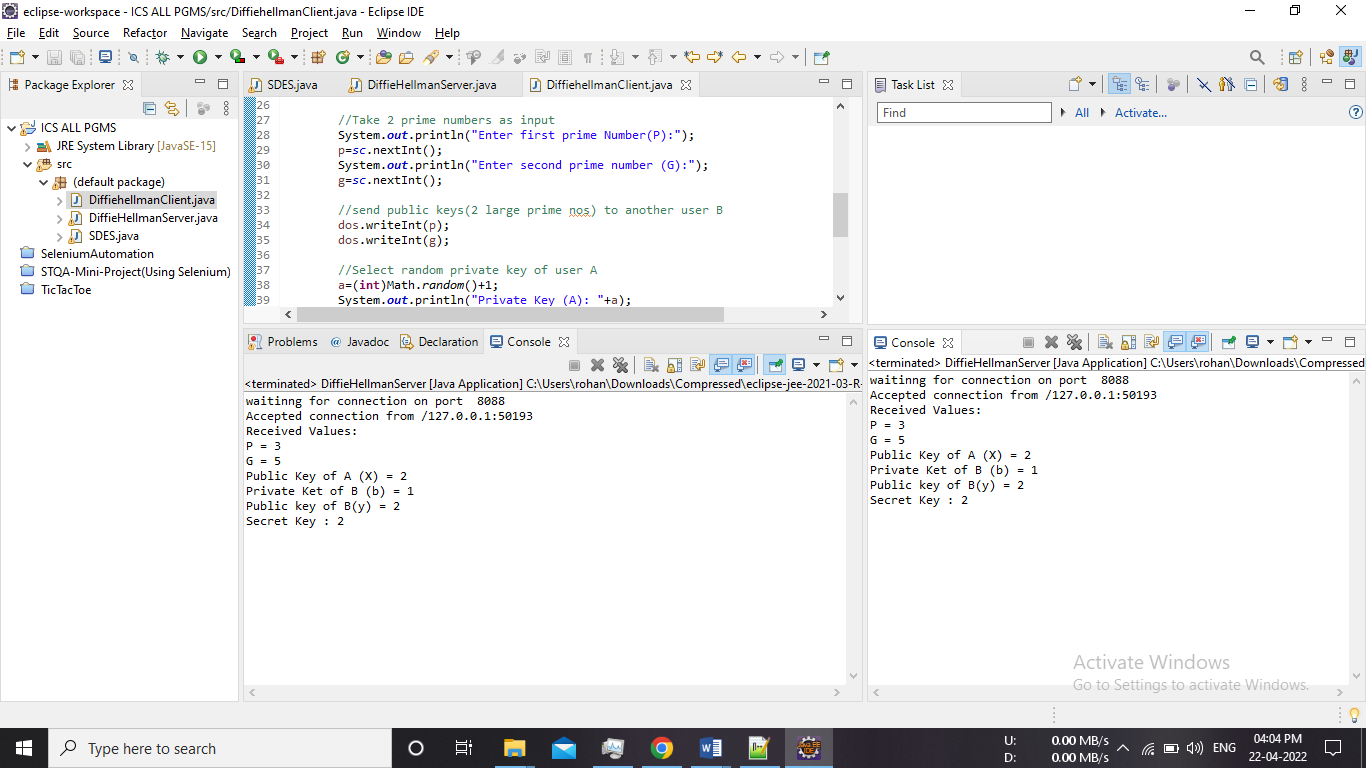
System.***out***.println("Secret Key : "+secretKey.intValue());

//If Secret key at A & B are same then communication can be performed

}

}

**Output:**



**ICS Assignment No: 4**

**RSA**

**Program Code:**

**RSAClient.java:**

**import** java.io.DataInputStream;

**import** java.io.DataOutputStream;

**import** java.io.IOException;

**import** java.math.BigInteger;

**import** java.net.Socket;

**import** java.net.UnknownHostException;

**import** java.util.Scanner;

**public** **class** RSAClient {

**public** **static** **final** String ***SERVER***="127.0.0.1";

**public** **static** **final** **int** ***PORT***=8088;

**static** **long** gcd(**long** phiOfN, **long** e) {

**long** temp;

**while** (**true**)

{

temp = phiOfN%e;

**if** (temp == 0)

**return** e;

phiOfN = e;

e = temp;

}

}

**public** **static** **long** getEncyptionkey(**long** phiOfN) {

**long** e=2;

**for**(e=2;e<phiOfN;e++) {

**if**(*gcd*(phiOfN,e)==1) {

//e=13;

**return** e;

}

}

//e=13;

**return** e;

}

**public** **static** **long** getDecryptionKey(**long** phiOfN, **long** e) {

**double** num=0.0;

**for**(**int** i=0;i<100;i++) {

num=(i\*phiOfN+1)/e;

**if**(num%1==0)

**break**;

}

**return** (**long**)num;

}

**public** **static** **void** main(String[] args) **throws** UnknownHostException, IOException {

System.***out***.println("==============Client Side====================");

Socket soc=**new** Socket(***SERVER***, ***PORT***);

System.***out***.println("Connected to "+soc.getRemoteSocketAddress());

DataInputStream dis=**new** DataInputStream(soc.getInputStream());

DataOutputStream dos=**new** DataOutputStream(soc.getOutputStream());

Scanner sc=**new** Scanner(System.***in***);

System.***out***.print("Enter Prime number (P):");

**long** p=sc.nextLong();

System.***out***.print("Enter Prime number (Q):");

**long** q=sc.nextLong();

**long** n=p\*q;

**long** phiOfN=(p-1)\*(q-1);

**int** e=(**int**)*getEncyptionkey*(phiOfN);

//long d=getDecryptionKey(phiOfN, e);

System.***out***.println("P = "+p+"\nQ = "+q);

System.***out***.println("N = "+n);

System.***out***.println("Phi(n) = "+phiOfN);

System.***out***.println("Public key (e) = "+e);

//System.out.println("Private Key (d) = "+d);

System.***out***.println("Enter message(number) to be encrypted");

**long** PT=sc.nextLong();

System.***out***.println("Plain Text : "+PT);

BigInteger CT=(BigInteger.*valueOf*(PT).pow(e)).mod(BigInteger.*valueOf*(n));

System.***out***.println("Cipher Text: "+CT.intValue());

//sending data to server for Decryption of CT

//dos.writeLong(d);

dos.writeLong(phiOfN);

dos.writeLong(n);

dos.writeLong(e);

dos.writeInt(CT.intValue());

soc.close();

}

//Extra Code if you want to check if entered no is prime number or not

**public** **static** **boolean** isPrime(**long** num) {

**if**(num==1||num==2)

**return** **true**;

**for**(**int** i=2;i<Math.*sqrt*(num);i++) {

**if**(num%i==0)

**return** **false**;

}

**return** **true**;

}

}

**RSAServer.java:**

**import** java.io.DataInputStream;

**import** java.io.DataOutputStream;

**import** java.io.IOException;

**import** java.math.BigInteger;

**import** java.net.ServerSocket;

**import** java.net.Socket;

**public** **class** RSAServer {

**public** **static** **long** getDecryptionKey(**long** phiOfN, **long** e) {

**double** num=0.0;

**for**(**int** i=1;i<100;i++) {

num=(**double**)(i\*phiOfN+1)/e; //Note:typecasting is imp here..Otherwise it will typecast to long value and below if condition will be true for first run

**if**(Math.*floor*(num)==num) //num%1==0

{

System.***out***.println("i = "+i+" num = "+num);

**break**;

}

}

**return** (**long**)num;

}

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

System.***out***.println("==============Server Side====================");

ServerSocket server=**new** ServerSocket(8088);

System.***out***.println("waitinng for connection on port "+server.getLocalPort());

Socket soc=server.accept();

System.***out***.println("Accepted connection from "+soc.getRemoteSocketAddress());

DataInputStream dis=**new** DataInputStream(soc.getInputStream());

DataOutputStream dos=**new** DataOutputStream(soc.getOutputStream());

**long** phiOfN=dis.readLong();

**long** n=dis.readLong();

**long** e=dis.readLong();

**int** CT=dis.readInt();

System.***out***.println("N = "+n);

System.***out***.println("Public key (e) = "+e);

System.***out***.println("phi(n) = "+phiOfN);

System.***out***.println("Cipher text : "+CT);

**long** d=*getDecryptionKey*(phiOfN, e);

System.***out***.println("Private key (D): "+d);

BigInteger PT=(BigInteger.*valueOf*(CT).pow((**int**)d)).mod(BigInteger.*valueOf*(n));

System.***out***.println("Decypted Plain text: "+PT.intValue());

soc.close();

}

}

**Output:**

