

**SCHOOL OF INFORMATION TECHNOLOGY AND ENGINEERING**

**A Project Report**

**On**

**AUTHENTICATION BASED HYBRID CRYPTOSYSTEM**

*Submitted in partial fulfilment of the requirements for the degree of*

**Bachelor of Computer Applications**

*By*

*LINGAMOORTHY V (20BCA0060)*

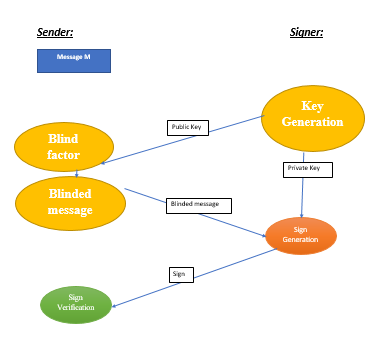
*GURUPRASATH A (20BCA0118*

**Under the guidance of**

Prof CHANDRASEGAR.T

**Signature of Guide:**

**PROPOSED METHODOLOGY AND ARCHITECTURE :**



**BLIND SIGN RSA:**

**HYBRID CRYPTOSYSTEM:**

**DESIGN AND MODULE DESCRIPTION:**

**BLIND SIGNATURE :**

* Blind Signature is a form of digital signature in which the content of a message is disguised (blinded) before it is signed.
* The resulting blind signature can be publicly verified against the original, un-blinded message in the manner of a regular digital signature.

Algorithm:

\* User have the Message (m).

\* Signer(Reciver) have the Private and Public key.

1 User makes a Sign Request to the Signer.

1. Signer shares his public key with the user.
2. User computes Blind factor.
3. User with the help of Blind factor computes Blind message.
4. User shares the Blind message to the Signer.
5. Signer generates the Sign.
6. User verifies the Sign.

**Public key and private key generation**

Private Key: The private key is one which is accessible only to the signer. It is used to generate the digital signature which is then attached to the message.

Public Key: The public key is made available to all those who receive the signed messages from the sender. It is used for verification of the received message.

* **ADVANTAGES OF BLIND SIGNATURE:**
* Authentication: The messages may often include information about the entity sending a message, that information may not be accurate. Digital signatures can be used to authenticate the source of messages. When ownership of a digital signature secret key is bound to a specific user, a valid signature shows that the message was sent by that user. The importance of high confidence in sender authenticity is especially obvious in a financial context.
* Integrity: The sender and receiver of a message may have a need for confidence that the message has not been altered during transmission. Although encryption hides the contents of a message, it may be possible to change an encrypted message without understanding it. However, if a message is digitally signed, any change in the message will invalidate the signature.

**Linear-PKC algorithm:**

**Key Generation:**

Input: p, q, R Output: k, c, L, b 1

* 1. Select three positive integers let’s say (p, q, R) such that R is a prime number and
  2. Defining the variables A, b, c as A=p\*q-R; b=R2A + p; c=RA + q
  3. Computation: b\*c-R= A (R3A 2 +p\*R + qR2 + 1)
  4. Defining: k = (b\*c – R)/A which implies k = R3A 2 +p\*R + qR2 + 1
  5. There is a unique L such that R\*L = 1(mod k) R\*k (By construction P does not divide n) that is GCD (P, k) = 1.

Thus the linear public and private keys are: Public key PU = (c, k) PR = L, b, k

**SIGN GENERATION**

* Compute Blind factor = r \* Q mod N = 5 (r is random Prime number)
* Compute Blind message: BF\*m mod N = 19 (m is message=8)
* Blinded message is being sent to the receiver.
* Sign generation = BM\*E\*D mod N = 20

**LINEAR-PKC:**

Linear RSA is a public key cryptography. PKC is a system where the sender, who send a message, to the receiver, by encrypting the message using the public key. This process is called encryption. And after the receiver receives the message from the sender the encrypted message is converted to original plain text using a private key. This process is called decryption. The public and private keys are generated by the algorithm. The main advantage of PKC over symmetric key cryptography is that PKC avoids the pre-distribution of the private key and the symmetric key cryptography has only one key to decrypt the message thus PKC has more advantageous than symmetric key cryptography.

**SIGN VERIFICATION**

* Sign verification t1= S\*Q mod N = 19 ( gives Blind message)
* t2 = t1\* r-1 mod N =16 (gives intermediate sign)
* t = t2\*E\*D mod N =8 (gives message)

If sign matches the message, then accept the sign.

**IMPLEMENTATION:**

**CODE :**

package com.mycompany.hybridcryptobs;

import java.io.\*;

import java.math.BigInteger;

import java.util.Random;

public class HybridcryptoBS

{

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

{

//\*\*\*\*KGS Start

long startTime1=System.currentTimeMillis();

long total1 = 0;

for (int i1 = 0; i1 < 10000000; i1++) total1 += i1;

BigInteger N,piN,d;

BigInteger one=new BigInteger("1");

BigInteger pi,qi,ei;

int bitLength = 1024; // INPUT bit length of primes P and Q;

Random rnd = new Random(); // create a random object

pi = BigInteger.probablePrime(bitLength, rnd);

qi = BigInteger.probablePrime(bitLength, rnd);

ei = BigInteger.probablePrime(bitLength, rnd); // Assigns probable Prime to p,q,e using bitLength and rnd

BigInteger M=new BigInteger("16"); //Original message

BigInteger a=new BigInteger("21");

BigInteger b=new BigInteger("12");

BigInteger M1=pi.multiply(qi).subtract(pi);

BigInteger E=M1.add(pi);

BigInteger D=M1.add(qi);

BigInteger N1=((E.multiply(D)).subtract(pi)).divide(M1);

BigInteger Q=pi.modInverse(N1);

//System.out.println("Q: " +Q);

long stopTime1 = System.currentTimeMillis();

long elapsedTime1 = stopTime1 - startTime1;

System.out.println("KGS time:");

System.out.println(elapsedTime1);

//Encryption

long startTime2 = System.currentTimeMillis();

long total2 = 0;

for (int i2 = 0; i2 < 10000000; i2++) total1 += i2;

//Input Message

BigInteger m1=new BigInteger("3");

BigInteger m2=new BigInteger("5");

//Encryption

BigInteger CT1=(m1.multiply(D)).mod(N1);

BigInteger CT2=(m2.multiply(D)).mod(N1);

//System.out.println("CT1: " +CT1);

//System.out.println("CT2: " +CT2);

long stopTime2 = System.currentTimeMillis();

long elapsedTime2 = stopTime2 - startTime2;

System.out.println("Encryption time:");

System.out.println(elapsedTime2);

//Encryption end

//Decryption

long startTime3 = System.currentTimeMillis(); long total3 = 0;

for (int i3 = 0; i3 < 10000000; i3++) total1 += i3;

//Decryption

BigInteger DT1=(Q.multiply(E).multiply(CT1)).mod(N1);

BigInteger DT2=(Q.multiply(E).multiply(CT2)).mod(N1);

System.out.println("DT1:"+DT1);

System.out.println("DT2:"+DT2);

N=pi.multiply(qi);

piN=(pi.subtract(one)).multiply(qi.subtract(one));

d=ei.modInverse(piN);

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

BigInteger CT;

BigInteger r=new BigInteger("5");

//Blind Factor r^e

BigInteger X;

X=(r.modPow(ei,N)).mod(N);

//Blinded Message

BigInteger XM;

XM=(X.multiply(M)).mod(N);

//System.out.println("Blinded Message XM:"+XM);

long stopTime3 = System.currentTimeMillis();

long elapsedTime3 = stopTime3 - startTime3;

System.out.println("Blinded Message time:");

System.out.println(elapsedTime3);

//Sign Generation

long startTime4 = System.currentTimeMillis();

long total4 = 0; for (int i4 = 0; i4 < 10000000; i4++) total1 += i4;

BigInteger s; s=XM.modPow(d,N);

//System.out.println("Signed value s:"+s);

long stopTime4 = System.currentTimeMillis();

long elapsedTime4 = stopTime4 - startTime4;

System.out.println("Sing Generation time:");

System.out.println(elapsedTime4);

long startTime5 = System.currentTimeMillis();

long total5 = 0; for (int i5 = 0; i5 < 10000000; i5++) total1 += i5;

//Sign verification

BigInteger t1,t; t1=(r.modInverse(N));

System.out.println("r inverse: "+t1);

t=((r.modInverse(N)).multiply(s)).modPow(ei,N);

System.out.println("True sign t:"+t);

long stopTime5 = System.currentTimeMillis();

long elapsedTime5 = stopTime5 - startTime5;

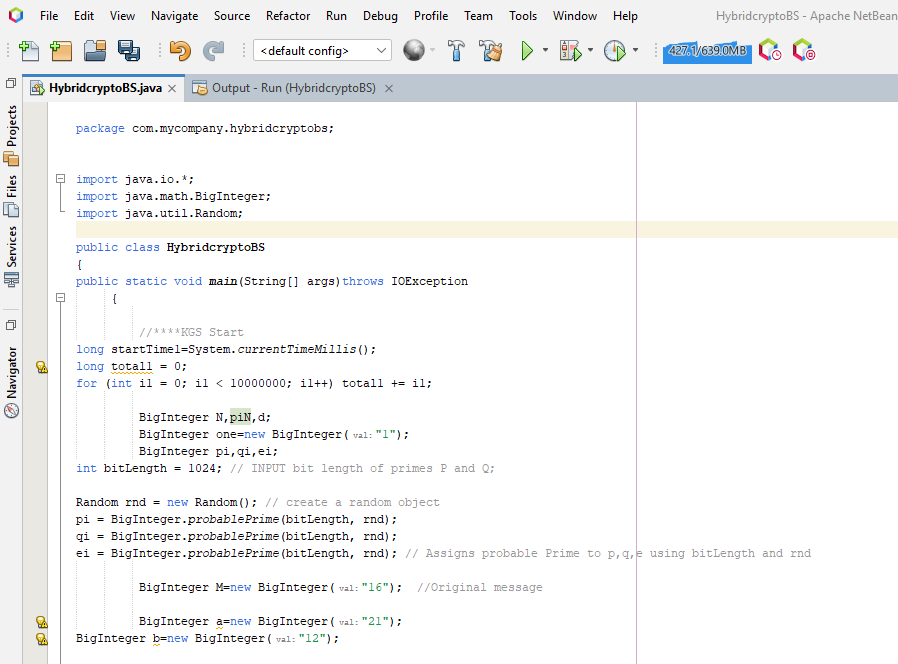
System.out.println("Blind verification time:");

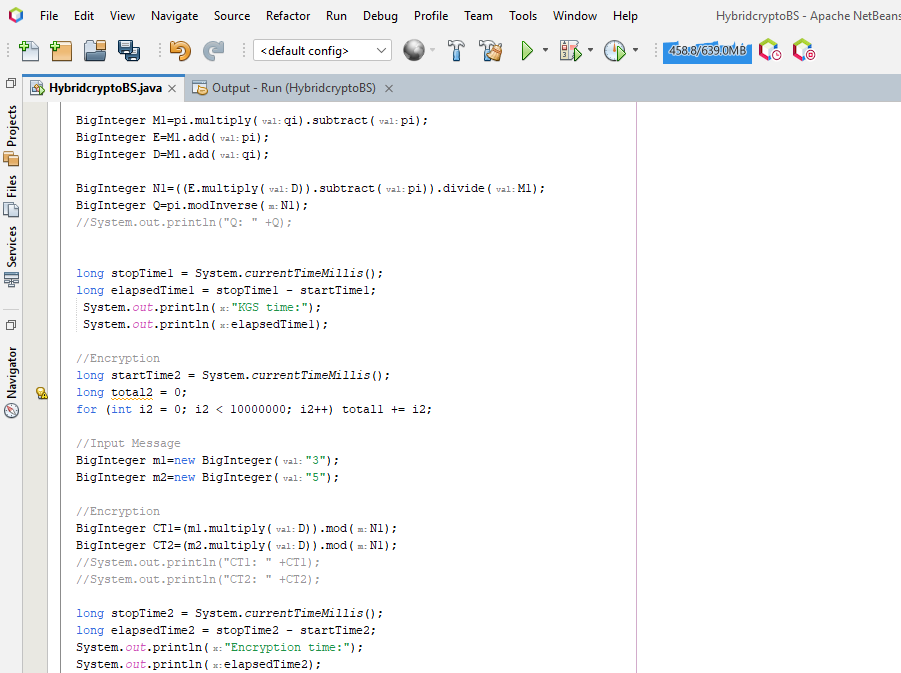
System.out.println(elapsedTime5);

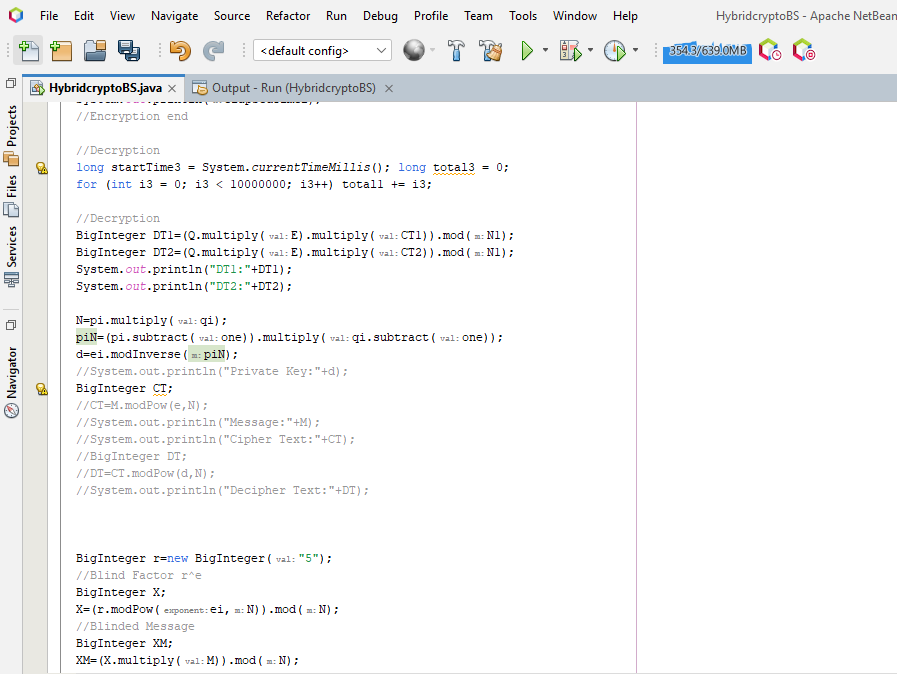
}

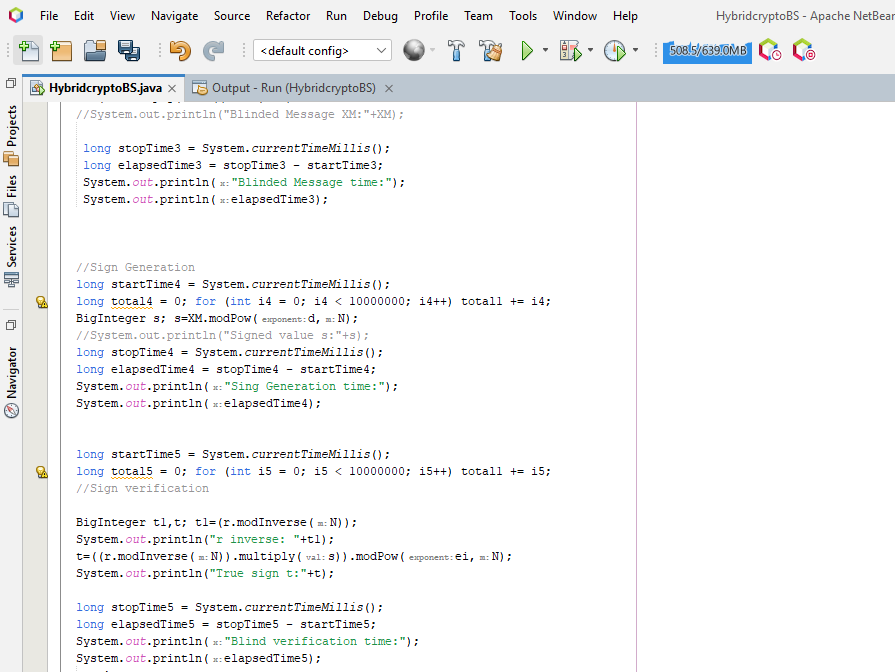
}

**SCREENSHOT :**

****





****

**OUTPUT :**

