

**POLI VARDHINI REDDY**

CRYPTOGRAPHY USING FERMAT FACTORIZATION

# CODE:

import java.math.BigInteger;

import java.security.SecureRandom;

import java.util.Scanner;

public class Main {

private static final int BIT\_SIZE = 64;

public static void main(String[] args) {

// Generate keys

KeyPair keyPair = generateKeys();

Scanner sc = new Scanner(System.in);

System.out.println("Enter the message to be encrypted (integer or text):");

String input = sc.nextLine();

BigInteger message;

// Determine if the input is a number or text

try {

long msg = Long.parseLong(input);

message = BigInteger.valueOf(msg);

} catch (NumberFormatException e) {

message = stringToBigInteger(input);

}

// Encryption

BigInteger ciphertext = encrypt(message, keyPair.getPublicKey());

System.out.println("Encrypted message: " + ciphertext);

// Decryption

BigInteger decryptedMessage = decrypt(ciphertext, keyPair.getPrivateKey());

String output;

try {

// Try to treat as a long integer

output = decryptedMessage.toString();

long originalInput = Long.parseLong(input); // Original was a number

} catch (NumberFormatException e) {

output = bigIntegerToString(decryptedMessage); // Original was a string

}

System.out.println("Decrypted message: " + output);

}

private static BigInteger stringToBigInteger(String input) {

byte[] bytes = input.getBytes();

return new BigInteger(1, bytes);

}

private static String bigIntegerToString(BigInteger message) {

byte[] bytes = message.toByteArray();

return new String(bytes);

}

private static KeyPair generateKeys() {

SecureRandom random = new SecureRandom();

// Generate p and q

BigInteger p = generatePrimeNumber(BIT\_SIZE, random);

BigInteger q = generatePrimeNumber(BIT\_SIZE, random);

// Compute n = p \* q and phi(n) = (p-1)(q-1)

BigInteger n = p.multiply(q);

BigInteger phiN = p.subtract(BigInteger.ONE).multiply(q.subtract(BigInteger.ONE));

// Common choice for e

BigInteger e = BigInteger.valueOf(65537);

// Compute d as e^(-1) mod phi(n)

BigInteger d = e.modInverse(phiN);

// Public and private keys

PublicKey publicKey = new PublicKey(e, n);

PrivateKey privateKey = new PrivateKey(d, n);

return new KeyPair(publicKey, privateKey);

}

private static BigInteger encrypt(BigInteger message, PublicKey publicKey) {

return message.modPow(publicKey.getE(), publicKey.getN());

}

private static BigInteger decrypt(BigInteger ciphertext, PrivateKey privateKey) {

return ciphertext.modPow(privateKey.getD(), privateKey.getN());

}

private static BigInteger generatePrimeNumber(int bitSize, SecureRandom random) {

BigInteger num;

do {

num = new BigInteger(bitSize, 100, random);

} while (!num.isProbablePrime(100));

return num;

}

private static class KeyPair {

private final PublicKey publicKey;

private final PrivateKey privateKey;

public KeyPair(PublicKey publicKey, PrivateKey privateKey) {

this.publicKey = publicKey;

this.privateKey = privateKey;

}

public PublicKey getPublicKey() {

return publicKey;

}

public PrivateKey getPrivateKey() {

return privateKey;

}

}

private static class PublicKey {

private final BigInteger e;

private final BigInteger n;

public PublicKey(BigInteger e, BigInteger n) {

this.e = e;

this.n = n;

}

public BigInteger getE() {

return e;

}

public BigInteger getN() {

return n;

}

}

private static class PrivateKey {

private final BigInteger d;

private final BigInteger n;

public PrivateKey(BigInteger d, BigInteger n) {

this.d = d;

this.n = n;

}

public BigInteger getD() {

return d;

}

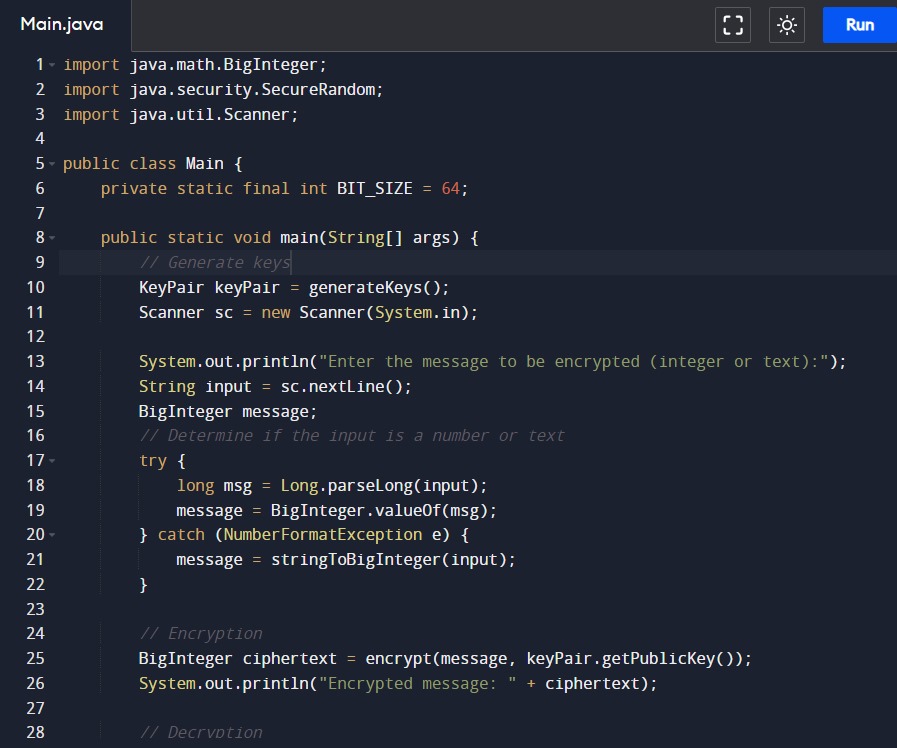
public BigInteger getN() {

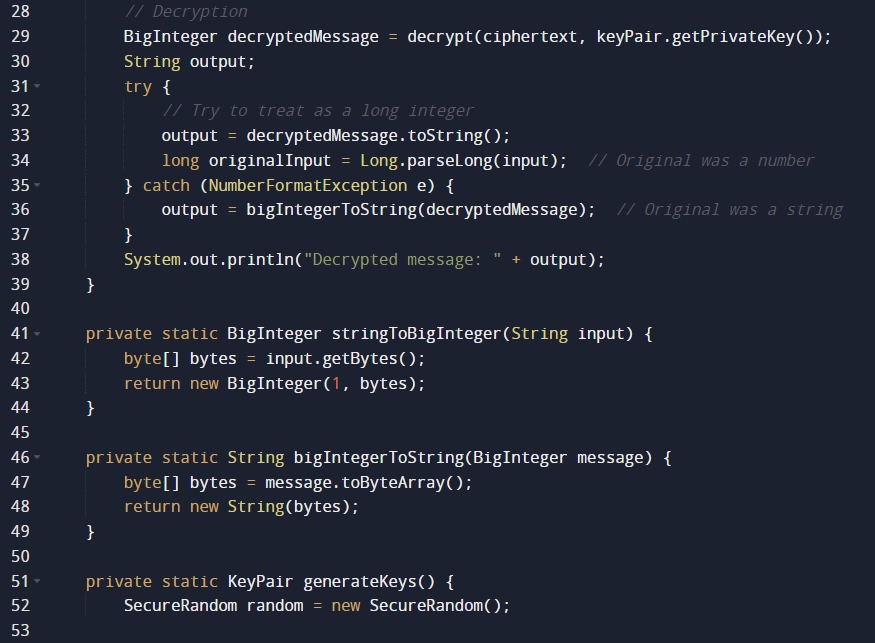
return n;

}

}

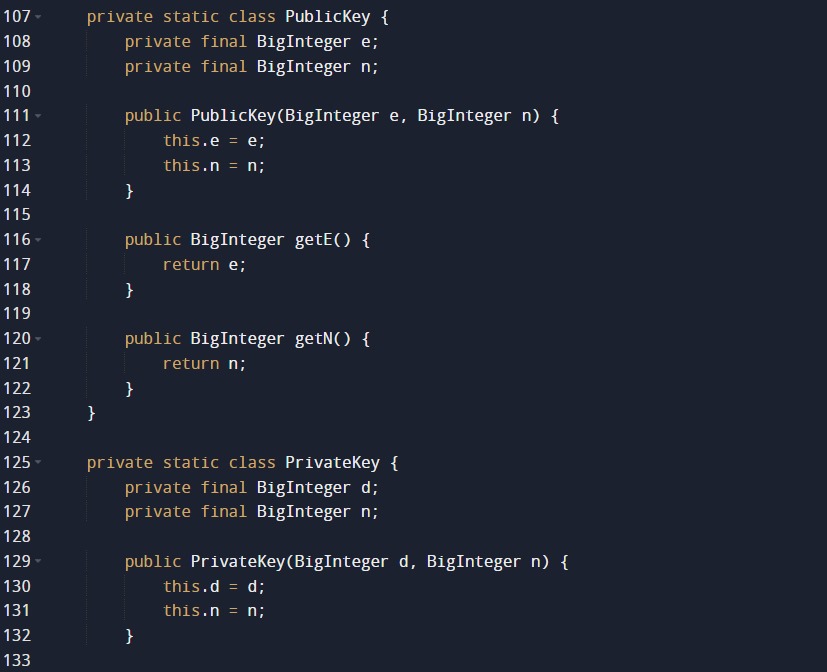
}

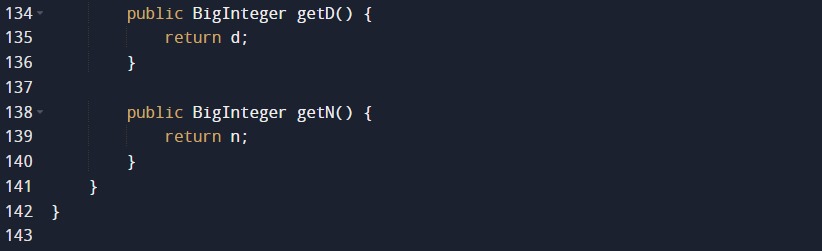












OUTPUT:

