**Name - Gaurang A Raorane Roll No - 49**

**Class - D15A Batch - C**

**EXPERIMENT 6**

**Aim:-**  Write a program to implement and analyze RSA cryptosystem.

**Theory :-**

**RSA Encryption and Decryption Theory:**

* Public-Key Cryptography:
  + RSA is a widely used public-key cryptography algorithm.
  + In public-key cryptography, two different keys are used: a public key for encryption and a private key for decryption.
  + The public key is openly available, while the private key is kept secret.
* Key Pair Generation:
  + The RSA key pair consists of a public key and a private key.
  + The keys are generated using a mathematical algorithm.
  + The public key consists of two parts: the modulus (N) and the public exponent (e).
  + The private key consists of the modulus (N) and the private exponent (d).
  + The security of RSA relies on the difficulty of factoring a large composite number, N, into its prime factors.
* Encryption Process:
  + To send an encrypted message to a recipient, the sender obtains the recipient's public key (N, e).
  + The sender converts the plaintext message into a numerical value M.
  + The sender calculates the ciphertext C using the recipient's public key: C = M^e mod N.
  + The ciphertext C is sent to the recipient.
* Decryption Process:
  + The recipient uses their private key (N, d) to decrypt the ciphertext.
  + The recipient computes the original numerical value M using the private key: M = C^d mod N.
  + The recipient then converts the numerical value M back into plaintext.
* Security:
  + RSA encryption is considered secure because it relies on the difficulty of factoring large numbers.
  + Breaking RSA encryption by factoring the modulus N into its prime factors is computationally infeasible for sufficiently large key sizes.
  + Longer key lengths provide stronger security but require more computational resources for encryption and decryption.
* Applications:
  + RSA is widely used for securing data transmission and authentication in various applications, including secure email, SSL/TLS for web security, digital signatures, and secure communication in IoT devices.
  + It is a fundamental building block of modern cryptography and is used in combination with other cryptographic techniques for robust security.
* Key Management:
  + The security of RSA relies on the protection of private keys.
  + Proper key management practices, including key storage and backup, are crucial to prevent unauthorized access and ensure the long-term security of encrypted data.

**Input:-**

import java.math.BigInteger;

import java.security.SecureRandom;

public class lab6 {

public static void main(String[] args) {

// Step 1: Generate RSA key pair

RSAKeyPair keyPair = generateRSAKeyPair(1024); // Change the key size as needed

// Step 2: Encrypt and decrypt a message

String originalMessage = "Hello, RSA!";

System.out.println("Original message: " + originalMessage);

BigInteger encryptedMessage = encrypt(originalMessage, keyPair.getPublicKey());

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

String decryptedMessage = decrypt(encryptedMessage, keyPair.getPrivateKey());

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

}

// Step 1: Generate RSA key pair

public static RSAKeyPair generateRSAKeyPair(int keySize) {

SecureRandom random = new SecureRandom();

BigInteger p = BigInteger.probablePrime(keySize / 2, random);

BigInteger q = BigInteger.probablePrime(keySize / 2, random);

BigInteger n = p.multiply(q);

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

BigInteger e = BigInteger.valueOf(65537); // Common choice for the public exponent

BigInteger d = e.modInverse(phi);

RSAPublicKey publicKey = new RSAPublicKey(n, e);

RSAPrivateKey privateKey = new RSAPrivateKey(n, d);

return new RSAKeyPair(publicKey, privateKey);

}

// Step 2: Encrypt a message

public static BigInteger encrypt(String message, RSAPublicKey publicKey) {

BigInteger plaintext = new BigInteger(message.getBytes());

return plaintext.modPow(publicKey.getExponent(), publicKey.getModulus());

}

// Step 2: Decrypt a message

public static String decrypt(BigInteger ciphertext, RSAPrivateKey privateKey) {

BigInteger plaintext = ciphertext.modPow(privateKey.getExponent(), privateKey.getModulus());

return new String(plaintext.toByteArray());

}

}

class RSAKeyPair {

private RSAPublicKey publicKey;

private RSAPrivateKey privateKey;

public RSAKeyPair(RSAPublicKey publicKey, RSAPrivateKey privateKey) {

this.publicKey = publicKey;

this.privateKey = privateKey;

}

public RSAPublicKey getPublicKey() {

return publicKey;

}

public RSAPrivateKey getPrivateKey() {

return privateKey;

}

}

class RSAPublicKey {

private BigInteger modulus;

private BigInteger exponent;

public RSAPublicKey(BigInteger modulus, BigInteger exponent) {

this.modulus = modulus;

this.exponent = exponent;

}

public BigInteger getModulus() {

return modulus;

}

public BigInteger getExponent() {

return exponent;

}

}

class RSAPrivateKey {

private BigInteger modulus;

private BigInteger exponent;

public RSAPrivateKey(BigInteger modulus, BigInteger exponent) {

this.modulus = modulus;

this.exponent = exponent;

}

public BigInteger getModulus() {

return modulus;

}

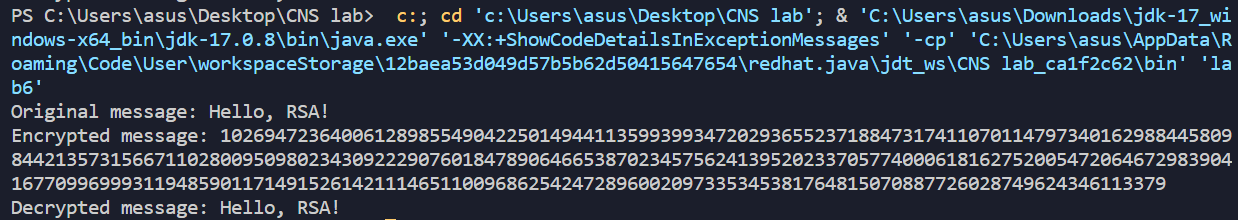
public BigInteger getExponent() {

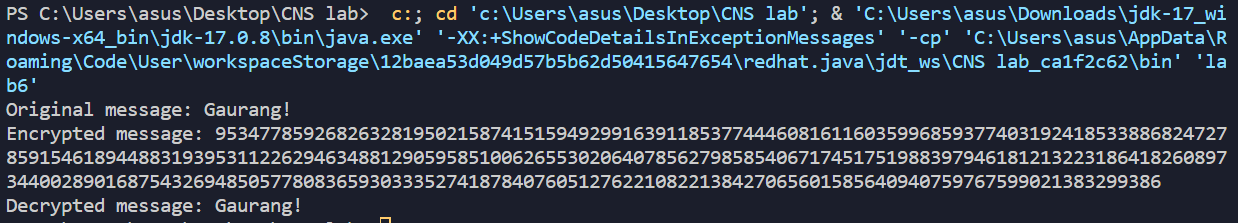
return exponent;

}

}

**Output:-**





Conclusion :-   
Thus RSA system is well understood and implemented.