**Name : Akash Babu Misal**

**PRN: 21520005**

**Class:** Final Year (Computer Science and Engineering)

**Course Name: Cryptography and Network Security**  **Lab**

**Assignment No – 09**

**Aim**: Implementation of RSA algorithm.

**Theory**:

**RSA Algorithm Overview:**

The RSA algorithm is a widely used public-key cryptosystem that provides a secure method for encryption and digital signatures. It relies on the mathematical properties of large prime numbers and modular arithmetic. RSA consists of two main components: key generation and encryption/decryption. The key pair consists of a public key (used for encryption) and a private key (used for decryption).

**Key Generation:**

1. Select Two Large Prime Numbers (p and q): The first step in key generation is to choose two distinct large prime numbers, p and q. These primes are kept secret.
2. Calculate n: Compute n as the product of p and q, i.e., n = p \* q. This is the modulus used in both the public and private keys.
3. Calculate φ(n) (Euler's Totient Function): Compute φ(n), which represents the count of positive integers less than n that are coprime to n. For RSA, φ(n) = (p-1) \* (q-1).
4. Choose the Public Exponent (e): Select a small odd integer e, typically 65537 (2^16 + 1). e should be coprime to φ(n), which ensures that there exists a modular multiplicative inverse.
5. Calculate the Private Exponent (d): Compute d as the modular multiplicative inverse of e modulo φ(n). In other words, d \* e ≡ 1 (mod φ(n)). The private key consists of d and n.

**Encryption:**

To encrypt a plaintext message (M) using the recipient's public key (e, n), the sender performs the following steps:

* Represent the plaintext message as an integer M (typically using padding schemes).
* Calculate the ciphertext C as C ≡ M^e (mod n).

**Decryption:**

To decrypt the ciphertext (C) using the recipient's private key (d, n), the recipient performs the following steps:

* Compute the plaintext message M as M ≡ C^d (mod n).
* Extract and interpret the original message from the integer representation of M (typically using reverse padding schemes).

**Code:**

import java.math.BigInteger;

import java.security.SecureRandom;

public class RSAExample{

    private BigInteger privateKey;

    private BigInteger publicKey;

    private BigInteger modulus;

    public RSAExample(int bitLength) {

        SecureRandom random = new SecureRandom();

        BigInteger p = BigInteger.probablePrime(bitLength, random);

        BigInteger q = BigInteger.probablePrime(bitLength, random);

        modulus = p.multiply(q);

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

        publicKey = new BigInteger("65537"); // Common public exponent

        privateKey = publicKey.modInverse(phi);

        System.out.println("RSA Key Generation:");

        System.out.println("p (prime number 1): " + p);

        System.out.println("q (prime number 2): " + q);

        System.out.println("n (modulus): " + modulus);

        System.out.println("f(n) (Euler's totient function): " + phi);

        System.out.println("Public Key (e): " + publicKey);

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

    }

    public BigInteger encrypt(BigInteger message) {

        return message.modPow(publicKey, modulus);

    }

    public BigInteger decrypt(BigInteger encryptedMessage) {

        return encryptedMessage.modPow(privateKey, modulus);

    }

    public static void main(String[] args) {

        int bitLength = 512; // Adjust the bit length as needed

        RSAExample rsa = new RSAExample(bitLength);

        String message = "Hello, RSA!";

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

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

        System.out.println("Original Message (as BigInteger): " + originalMessage);

        BigInteger encryptedMessage = rsa.encrypt(originalMessage);

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

        BigInteger decryptedMessage = rsa.decrypt(encryptedMessage);

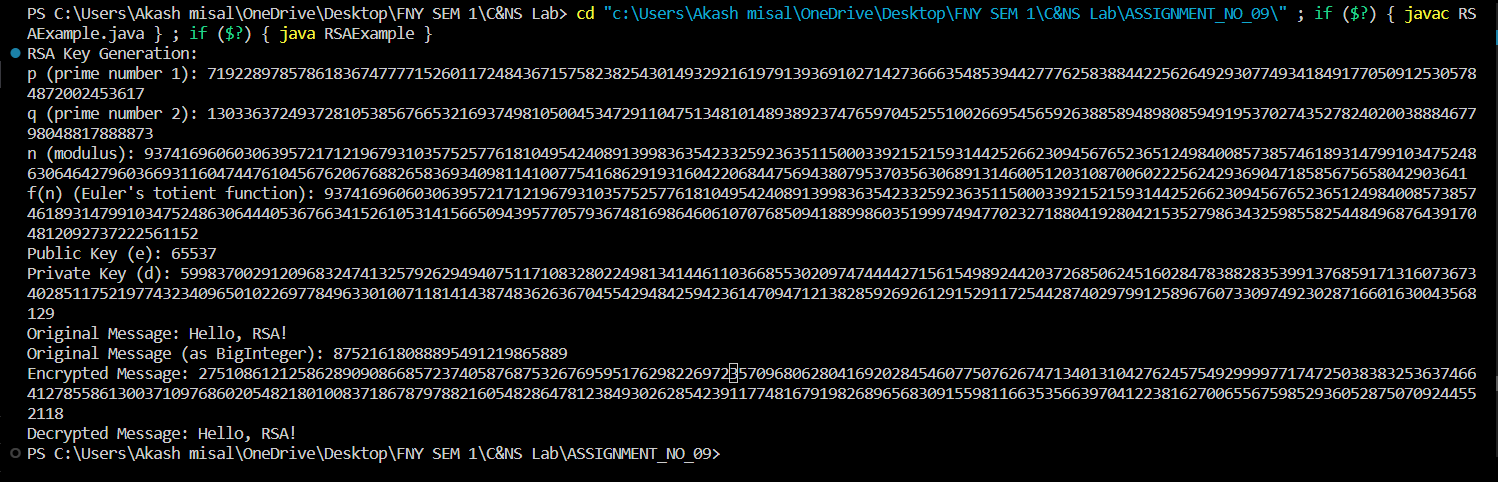
        String decryptedText = new String(decryptedMessage.toByteArray());

        System.out.println("Decrypted Message: " + decryptedText);

    }

}

**Output –**

****

**Conclusion:**

The RSA algorithm is a widely used public-key cryptosystem that enables secure data encryption and digital signatures. This experiment allows us to understand the fundamental principles of RSA, including key generation, encryption, and decryption. It highlights the importance of key security, prime number selection, and the mathematical relationships that underlie secure communication. RSA is crucial in securing data transmission over the internet, digital signatures, and other cryptographic applications.