RSA algorithm

import java.math.BigInteger;

import java.util.Scanner;

public class RSAAlgorithm {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

// Input prime numbers p and q

System.out.print("Enter prime number p: ");

BigInteger p = scanner.nextBigInteger();

System.out.print("Enter prime number q: ");

BigInteger q = scanner.nextBigInteger();

// Compute n = p \* q

BigInteger n = p.multiply(q);

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

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

System.out.println("phi is " + phi);

// Input the public exponent e

System.out.print("Enter public exponent e (1 < e < phi(n) and gcd(e, phi(n)) = 1): ");

BigInteger e = scanner.nextBigInteger();

// Ensure e is valid

if (e.compareTo(BigInteger.ONE) <= 0 || e.compareTo(phi) >= 0 || !e.gcd(phi).equals(BigInteger.ONE)) {

System.out.println("Invalid value of e. It must satisfy 1 < e < phi(n) and gcd(e, phi(n)) = 1.");

return;

}

// Compute d such that (d \* e) % phi(n) = 1 (d is the multiplicative inverse of e mod phi(n))

BigInteger d = e.modInverse(phi);

// Input the message M

System.out.print("Enter the message integer M (0 <= M < n): ");

BigInteger M = scanner.nextBigInteger();

// Ensure the message is valid

if (M.compareTo(BigInteger.ZERO) < 0 || M.compareTo(n) >= 0) {

System.out.println("Invalid value of M. It must satisfy 0 <= M < n.");

return;

}

// Compute ciphertext C = M^e mod n

BigInteger C = M.modPow(e, n);

// Display the private key d and the ciphertext C

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

System.out.println("Ciphertext C: " + C);

System.out.println("Message M after decryption: " + C.modPow(d, n));

scanner.close();

}

}

**Sample Output**

Enter prime number p: 5

Enter prime number q: 11

phi is 40

Enter public exponent e (1 < e < phi(n) and gcd(e, phi(n)) = 1): 7

Enter the message integer M (0 <= M < n): 2

Private key d: 23

Ciphertext C: 18

Message M after decryption: 2

Enter prime number p: 17

Enter prime number q: 11

phi is 160

Enter public exponent e (1 < e < phi(n) and gcd(e, phi(n)) = 1): 7

Enter the message integer M (0 <= M < n): 2

Private key d: 23

Ciphertext C: 128

Message M after decryption: 2

**Explanation:**

1. **Input**:
   * Prime numbers p and q are input by the user.
   * The public exponent e is also input by the user. The program checks that e satisfies the necessary conditions (1 < e < phi(n) and gcd (e, phi(n)) = 1).
2. **Calculation**:
   * Computes n=p×q
   * Computes ϕ(n)=(p−1) ×(q−1)
   * Validates the user-provided e.
   * Computes the private exponent d, which is the modular multiplicative inverse of e modulo ϕ(n)
   * The user inputs the message integer M, and the program checks that M satisfies 0 <= M < n.
3. **Encryption**:
   * Computes the cipher text C as Me mod n
4. **Output**:
   * The private key d and the cipher text C are printed.