Computer Network Security (ECE-543)

Project 2: Implement RSA Algorithm

Alan Palayil (A20447935)

Due Date: 4/30/2023

Contents

Project 2: Implement RSA Algorithm..1
Abstract:..3
Introduction:..3
Background:..3
Implementation:..4
Results:..4
Figure 1: Commands Used..4
Figure 2: Primecheck Test Cases..4
Figure 3: Primegen Test Cases..4
Figure 4: Keygen Test Cases..5
Figure 5: Menu..5
Figure 6: Encrypt Test Cases..5
Figure 7: Decrypt Test Cases..5
Conclusion:..6
Appendix:..6
RSA_Main.java..6
User_Interface.java..10
Prime_Numbers.java..12
RSA_Algorithm.java..13

Abstract:

The objective of this project is to investigate the implementation of the RSA encryption algorithm within a computer program application. The program encompasses fundamental functions associated with the generation of public and private key pairs, encryption, and decryption of values, as well as the generation and verification of prime numbers. This exploration aims to provide insights into the practical application of the RSA algorithm and its core components.

Introduction:

This project delves into the programming and implementation of the RSA encryption algorithm. The program will incorporate two sets of functions: one related to prime numbers and another related to RSA encryption. The prime number-related functions will generate prime numbers of a desired length and determine if a given value is a prime or composite number. The RSA encryption-related functions will generate public and private key pairs, encrypt values using specific public keys and n values, and decrypt values using specific private keys and n values. Ultimately, these functions will be rigorously tested using a variety of test cases to ensure their accuracy and effectiveness.

Background:

The RSA algorithm employs a straightforward process for generating public and private key pairs. The initial step involves determining the public key. In RSA implementation, the public and private key pairs have a relationship such that $e*d=1 \ mod \ \phi(n)$, where "e" is the public key, "d" is the private key, and $\phi(n)=(p-1)*(q-1)$ with p and q being two prime numbers. When computing the public and private key pairs, the public key can be set to an arbitrary prime number – in this project, the commonly chosen value of $2^{16}+1$, or 65537. The private key is then calculated as $d = \frac{1}{e} \ mod \ \phi(n)$. Subsequently, the value "n" is computed such that $n=p*q$. Using this value "n" and either the public or private key, RSA encryption and decryption can be performed. The encryption process follows the equation $ciphertext=plaintext^e \ mod \ n$, while the decryption process follows the equation $plaintext=ciphertext^d \ mod \ n$. Since the formulas for encryption and decryption within RSA are identical, except for swapping the public and private keys along with the plaintext for ciphertext, the same hardware can be used for both processes.

A crucial aspect of the RSA algorithm is the utilization of prime numbers. Several methods exist for determining whether a number is prime, but this project employs Fermat's Theorem. According to Fermat's Theorem, a number is prime if, for all values of a such that $1<a<n-1$ where "n" is the number to be tested, $a^{n-1} \ \% \ n=1$. If any value of a is found such that $a^{n-1} \ \% \ n\neq1$, the tested value "n" is determined to be non-prime. For large values of "n", it is impractical to test every value between 1 and n-1, so random numbers in between are chosen for testing. As not all values are directly tested in the case of large numbers by the theorem, it only provides complete confirmation for numbers that are found to be non-prime. Unless all values are tested, numbers

deemed "prime" are only considered "prime" to a degree of certainty that directly relates to the number of values tested; more tests yield a higher degree of certainty.

Implementation:

The RSA algorithm program was implemented using four Java files. The first file, "RSA_Main.java," executes the RSA algorithm program when run and contains test cases for all functions. The second file, "User_Interface.java," holds the on-screen menu for the program and manages function calls. The third file, "Prime_Numbers.java," contains two methods related to testing and generating prime numbers: primecheck() and primegen(). The primecheck() method employs Fermat's Theorem with a certainty of 100 to determine whether the given value is prime or composite. The primegen() method generates a prime number with a specified bit length by creating a string of "0" with the desired length, setting the most significant bit to "1," and adding random values to the number until primecheck() confirms it as prime.

The fourth file, "RSA_Algorithm.java," contains functions for generating key pairs, encrypting values, and decrypting values. The keygen() method accepts two prime numbers, generates the "n" value, sets the public key to $2^{16}+1$, and calculates the private key based on the two prime numbers and "n." The encrypt() method takes in three values: "n," a public key, and the value to be encrypted. It then calculates the ciphertext using the RSA encryption equation. The decrypt() method also takes in three values: "n," a private key, and the value to be decrypted. It then computes the plaintext using the RSA decryption equation.

Results:

The terminal output shows the results of test cases for various functions of the RSA algorithm.

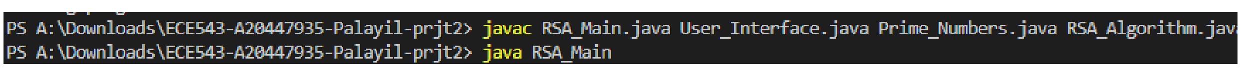

Figure 1: Commands Used

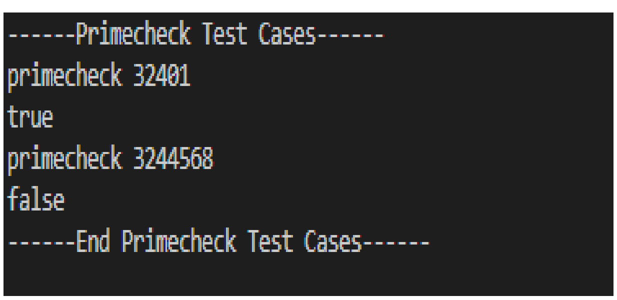

Figure 2: Primecheck Test Cases

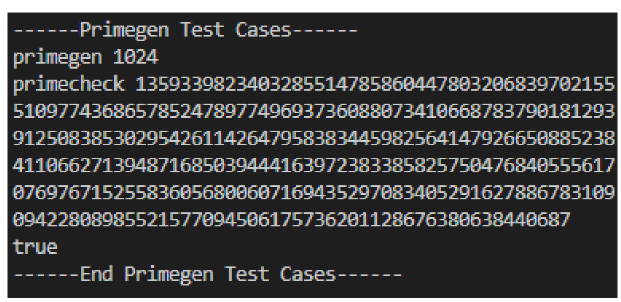

Figure 3: Primegen Test Cases

```
------Keygen Test Cases------
keygen 127 131
Public Key (n, e): (16637, 65537)
Private Key (n, d): (16637, 14453)
keygen 1019 1021
Public Key (n, e): (1040399, 65537)
Private Key (n, d): (1040399, 803633)
keygen 1093 1097
Public Key (n, e): (1199021, 65537)
Private Key (n, d): (1199021, 1031105)
keygen 433 499
Public Key (n, e): (216067, 65537)
Private Key (n, d): (216067, 22913)
keygen 1061 1063
Public Key (n, e): (1127843, 65537)
Private Key (n, d): (1127843, 715193)
keygen 1217 1201
Public Key (n, e): (1461617, 65537)
Private Key (n, d): (1461617, 566273)
keygen 313 337
Public Key (n, e): (105481, 65537)
Private Key (n, d): (105481, 70145)
keygen 419 463
Public Key (n, e): (193997, 65537)
Private Key (n, d): (193997, 159521)
------End Keygen Test Cases------
```

Figure 4: Keygen Test Cases

```
Choose an operation: [Enter the number]
1. Check Prime No.
2. Check Primegen no. of bits
3. Generate Key of Two Primes
4. Encrypt 'n' 'e' 'c'
5. Decrypt 'n' 'd' 'm'
6. Quit

Enter the operation you wish
```

Figure 5: Menu

```
------Encrypt Test Cases------
encrypt 16637 11 20
Encrypted message: 12046
encrypt 1040399 7 99
Encrypted message: 579196
encrypt 1199021 5 70
Encrypted message: 871579
encrypt 216067 5 89
Encrypted message: 23901
encrypt 1127843 7 98
Encrypted message: 871444
encrypt 1461617 7 113
Encrypted message: 1411436
encrypt 105481 5 105
Encrypted message: 36549
encrypt 193997 5 85
Encrypted message: 147738
------End Encrypt Test Cases------
```

Figure 6: Encrypt Test Cases

```
------ decrypt test cases ------
decrypt 16637 14891 12046
Decrypted message: 20
decrypt 1040399 890023 16560
Decrypted message: 104
decrypt 1199021 478733 901767
Decrypted message: 71
decrypt 216067 172109 169487
Decrypted message: 101
decrypt 1127843 964903 539710
Decrypted message: 119
decrypt 1461617 1250743 93069
Decrypted message: 83
decrypt 105481 41933 78579
Decrypted message: 76
decrypt 193997 154493 1583
Decrypted message: 122
------ end decrypt test cases ------
```

Figure 7: Decrypt Test Cases

The primecheck function is tested with two inputs, 32401 and 3244568, and returns true and false respectively. The primegen function is tested with an input of 1024 and generates a prime number of length 1024 bits which is then checked using the primecheck function. The keygen function is tested with several pairs of prime numbers and outputs the public and private keys generated by the function. The encrypt function is tested with pairs of numbers (n, e) and a message (c) to encrypt and returns the encrypted message. The decrypt function is tested with pairs of numbers (n, d) and a message (m) to decrypt and returns the decrypted message. The user is prompted to choose an operation from the menu, which includes options to check prime numbers, generate prime numbers of a given length, generate keys from two prime numbers, encrypt a message, decrypt a message, or quit the program.

Conclusion:

In this project, we successfully implemented the RSA algorithm within a Java program. The program comprises functions such as primecheck(), primegen(), keygen(), encrypt(), and decrypt(). These methods enable the generation and verification of prime numbers, the creation of public and private key pairs, as well as the encryption and decryption of values. The project demonstrates the power of programming in executing and accelerating the complex and repetitive calculations typically required in encryption algorithms.

Utilizing the Java programming language, we developed and tested each RSA function, ensuring their correct and efficient operation. The test cases generated for each function confirmed their accuracy and effectiveness. Consequently, the implemented program serves as a reliable and efficient method for performing RSA encryption.

Appendix:

RSA_Main.java

```
/*
 * ECE-543 Project 2: Implement RSA Algorithm
 * Execute this file first.
 * After running the test cases for each method, the program's menu will display the available functions and
their syntax.
*/
import java.math.*;

public class RSA_Main {
  private static Prime_Numbers prime = new Prime_Numbers();
  private static RSA_Algorithm rsa = new RSA_Algorithm();
  private static User_Interface ui = new User_Interface();

  public static void main(String args[]) {
    runTestCases();
    ui.run();
  }

  public static void runTestCases() {
    testPrimecheck();
    testPrimegen();
    testKeygen();
    testEncrypt();
    testDecrypt();
  }

  public static void testPrimecheck() {
    System.out.println("------Primecheck Test Cases------");
```

```java
  BigInteger bigInt = new BigInteger("32401");
  System.out.println("primecheck " + bigInt);
  System.out.println(prime.primecheck(bigInt));
  bigInt = new BigInteger("3244568");
  System.out.println("primecheck " + bigInt);
  System.out.println(prime.primecheck(bigInt));
  //Possible additional primegen test cases here
  System.out.println("------End Primecheck Test Cases------\n");
  }

public static void testPrimegen() {
  System.out.println("------Primegen Test Cases------");
  int bits = 1024;
  System.out.println("primegen " + bits);
  BigInteger gen = prime.primegen(bits);
  System.out.println("primecheck " + gen);
  System.out.println(prime.primecheck(gen));
  //Possible additional primegen test cases here
  System.out.println("------End Primegen Test Cases------\n");
}

public static void testKeygen() {
  System.out.println("------Keygen Test Cases------");
  String primeOne = "127";
  String primeTwo = "131";
  System.out.println("keygen " + primeOne + " " + primeTwo);
  rsa.keygen(primeOne, primeTwo);
  // Additional keygen test cases here
  primeOne = "1019";
  primeTwo = "1021";
  System.out.println("keygen " + primeOne + " " + primeTwo);
  rsa.keygen(primeOne, primeTwo);
  primeOne = "1093";
  primeTwo = "1097";
  System.out.println("keygen " + primeOne + " " + primeTwo);
  rsa.keygen(primeOne, primeTwo);
  primeOne = "433";
  primeTwo = "499";
  System.out.println("keygen " + primeOne + " " + primeTwo);
  rsa.keygen(primeOne, primeTwo);
  primeOne = "1061";
  primeTwo = "1063";
  System.out.println("keygen " + primeOne + " " + primeTwo);
  rsa.keygen(primeOne, primeTwo);
  primeOne = "1217";
```

```java
  primeTwo = "1201";
  System.out.println("keygen " + primeOne + " " + primeTwo);
  rsa.keygen(primeOne, primeTwo);
  primeOne = "313";
  primeTwo = "337";
  System.out.println("keygen " + primeOne + " " + primeTwo);
  rsa.keygen(primeOne, primeTwo);
  primeOne = "419";
  primeTwo = "463";
  System.out.println("keygen " + primeOne + " " + primeTwo);
  rsa.keygen(primeOne, primeTwo);
  System.out.println("------End Keygen Test Cases------\n");
}

public static void testEncrypt() {
  System.out.println("------Encrypt Test Cases------");
  String n = "16637";
  String e = "11";
  String c = "20";
  System.out.println("encrypt " + n + " " + e + " " + c);
  rsa.encrypt(n, e, c);
  // Additional encrypt test cases here
  n = "1040399";
  e = "7";
  c = "99";
  System.out.println("encrypt " + n + " " + e + " " + c);
  rsa.encrypt(n, e, c);
  n = "1199021";
  e = "5";
  c = "70";
  System.out.println("encrypt " + n + " " + e + " " + c);
  rsa.encrypt(n, e, c);
  n = "216067";
  e = "5";
  c = "89";
  System.out.println("encrypt " + n + " " + e + " " + c);
  rsa.encrypt(n, e, c);
  n = "1127843";
  e = "7";
  c = "98";
  System.out.println("encrypt " + n + " " + e + " " + c);
  rsa.encrypt(n, e, c);
  n = "1461617";
  e = "7";
  c = "113";
```

```
  System.out.println("encrypt " + n + " " + e + " " + c);
  rsa.encrypt(n, e, c);
  n = "105481";
  e = "5";
  c = "105";
  System.out.println("encrypt " + n + " " + e + " " + c);
  rsa.encrypt(n, e, c);
  n = "193997";
  e = "5";
  c = "85";
  System.out.println("encrypt " + n + " " + e + " " + c);
  rsa.encrypt(n, e, c);
  System.out.println("------End Encrypt Test Cases------\n");
}

public static void testDecrypt() {
  System.out.println("------ decrypt test cases ------");
  String n = "16637";
  String d = "14891";
  String m = "12046";
  System.out.println("decrypt " + n + " " + d + " " + m);
  rsa.decrypt(n, d, m);
  // Additional decrypt test cases here
  n = "1040399";
  d = "890023";
  m = "16560";
  System.out.println("decrypt " + n + " " + d + " " + m);
  rsa.decrypt(n, d, m);
  n = "1199021";
  d = "478733";
  m = "901767";
  System.out.println("decrypt " + n + " " + d + " " + m);
  rsa.decrypt(n, d, m);
  n = "216067";
  d = "172109";
  m = "169487";
  System.out.println("decrypt " + n + " " + d + " " + m);
  rsa.decrypt(n, d, m);
  n = "1127843";
  d = "964903";
  m = "539710";
  System.out.println("decrypt " + n + " " + d + " " + m);
  rsa.decrypt(n, d, m);
  n = "1461617";
  d = "1250743";
```

```
    m = "93069";
    System.out.println("decrypt " + n + " " + d + " " + m);
    rsa.decrypt(n, d, m);
    n = "105481";
    d = "41933";
    m = "78579";
    System.out.println("decrypt " + n + " " + d + " " + m);
    rsa.decrypt(n, d, m);
    n = "193997";
    d = "154493";
    m = "1583";
    System.out.println("decrypt " + n + " " + d + " " + m);
    rsa.decrypt(n, d, m);
    System.out.println("------ end decrypt test cases ------\n");
  }
}
```

User_Interface.java

```
/*
 * User Interface for Project 2.
 * Execute the program using Main.java.
*/
import java.util.Scanner;
import java.math.*;

public class User_Interface {
    private boolean maintain = true;
    private static Prime_Numbers prime = new Prime_Numbers();
    private static RSA_Algorithm rsa = new RSA_Algorithm();
    public void run() {
        Scanner scanner = new Scanner(System.in);
        while (maintain) {
            System.out.println("Choose an operation: [Enter the number]");
            System.out.println("1. Check Prime No.");
            System.out.println("2. Check Primegen no. of bits");
            System.out.println("3. Generate Key of Two Primes");
            System.out.println("4. Encrypt 'n' 'e' 'c'");
            System.out.println("5. Decrypt 'n' 'd' 'm'");
            System.out.println("6. Quit");
            System.out.println("\nEnter the operation you wish");
            int choice = scanner.nextInt();
            switch (choice) {
                case 1:
                    String[] answer = scanner.nextLine().split(" ");
```

```
            try {
                BigInteger num = new BigInteger(answer[1]);
                System.out.println(prime.primecheck(num));
            }
            catch(Exception e)  {
                System.out.println("Missing values");
            }
            break;
        case 2:
            try {
                System.out.println("Enter the number of bits:");
                int bits = scanner.nextInt();
                BigInteger generatedPrime = prime.primegen(bits);
                System.out.println("Generated prime number: " + generatedPrime);
            }
            catch(Exception e)  {
                System.out.println("Missing values");
            }
            break;
        case 3:
            try {
                System.out.println("Enter two prime numbers (separated by space):");
                String primeOne = scanner.next();
                String primeTwo = scanner.next();
                rsa.keygen(primeOne, primeTwo);
            }
            catch(Exception e)  {
                System.out.println("Missing values");
            }
            break;
        case 4:
            try {
                System.out.println("Enter the values n, e, and c (separated by space):");
                String n = scanner.next();
                String e = scanner.next();
                String c = scanner.next();
                rsa.encrypt(n, e, c);
            }
            catch(Exception e)  {
                System.out.println("Missing values");
            }
            break;
        case 5:
            try {
                System.out.println("Enter the values n, d, and m (separated by space):");
```

```
                String n = scanner.next();
                String d = scanner.next();
                String m = scanner.next();
                rsa.decrypt(n, d, m);
            }
            catch(Exception e)  {
                System.out.println("Missing values");
            }
            break;
        case 6:
            maintain = false;
            System.out.println("Exiting program...");
            break;
        default:
            System.out.println("Invalid choice. Please try again.");
        }
    }
    scanner.close();
  }
}
```

Prime_Numbers.java

```
/*
 * Prime Number Check for Project 2.
 * Execute the program using Main.java.
*/
import java.math.BigInteger;
import java.util.Random;
public class Prime_Numbers {
  final BigInteger One = new BigInteger("1");
  public boolean primecheck(BigInteger check) {
    if (check.compareTo(One) <= 0)
        return false;
    if (check.compareTo(new BigInteger("3")) <= 0)
        return true;
    if (check.mod(new BigInteger("2")).equals(BigInteger.ZERO))
        return false;
    // Compute s and t such that check-1 = 2^s * t
    BigInteger t = check.subtract(BigInteger.ONE);
    int s = 0;
    while (t.mod(new BigInteger("2")).equals(BigInteger.ZERO)) {
        t = t.divide(new BigInteger("2"));
        s++;
    }
```

```
  // Repeat k times
  for (int k = 0; k < 100; k++) {
    BigInteger a = getRandomBase(check.subtract(BigInteger.ONE));
    BigInteger v = a.modPow(t, check);
    if (!v.equals(BigInteger.ONE)) {
      int i = 0;
      while (!v.equals(check.subtract(BigInteger.ONE))) {
        if (++i == s || v.modPow(new BigInteger("2"), check).equals(BigInteger.ONE))
          return false;
        v = v.modPow(new BigInteger("2"), check);
      }
    }
  }
  return true;
 }
 public BigInteger primegen(int bits) {
  Random rand = new Random();
  BigInteger prime = BigInteger.ZERO;
  while (!prime.isProbablePrime(100)) {
    BigInteger maxVal = BigInteger.ONE.shiftLeft(bits).subtract(BigInteger.ONE);
    prime = new BigInteger(bits, rand);
    prime = prime.or(BigInteger.ONE.shiftLeft(bits - 1));
    prime = prime.or(BigInteger.ONE);
    while (prime.compareTo(maxVal) > 0) {
      prime = new BigInteger(bits, rand);
      prime = prime.or(BigInteger.ONE.shiftLeft(bits - 1));
      prime = prime.or(BigInteger.ONE);
    }
   }
  return prime;
 }
 private BigInteger getRandomBase(BigInteger n) {
  Random rand = new Random();
  BigInteger result = new BigInteger(n.bitLength(), rand);
  while (result.compareTo(n) >= 0) {
    result = new BigInteger(n.bitLength(), rand);
  }
  return result;
 }
}
```

RSA_Algorithm.java

```
/*
 * RSA Algorithm for Project 2.
```

```
 * Execute the program using Main.java.
*/
import java.math.BigInteger;
public class RSA_Algorithm {
  final BigInteger One = new BigInteger("1");
  public void keygen(String primeOne, String primeTwo) {
    BigInteger p = new BigInteger(primeOne);
    BigInteger q = new BigInteger(primeTwo);
    BigInteger n = p.multiply(q);
    //Calculate f(n)
    BigInteger phi = p.subtract(One).multiply(q.subtract(One));
    //Public Key Value
    BigInteger e = new BigInteger("65537");
    //Private Key Value
    BigInteger d = e.modInverse(phi);
    System.out.println("Public Key (n, e): (" + n + ", " + e + ")");
    System.out.println("Private Key (n, d): (" + n + ", " + d + ")");
  }
  public String encrypt(String nIn, String eIn, String cIn) {
    BigInteger n = new BigInteger(nIn);
    BigInteger e = new BigInteger(eIn);
    BigInteger c = new BigInteger(cIn);
    BigInteger cipher = c.pow(e.intValue()).mod(n);
    System.out.println("Encrypted message: " + cipher);
    return cipher.toString();
  }
  public String decrypt(String nIn, String dIn, String mIn) {
    BigInteger n = new BigInteger(nIn);
    BigInteger d = new BigInteger(dIn);
    BigInteger m = new BigInteger(mIn);
    BigInteger plaintext = m.pow(d.intValue()).mod(n);
    System.out.println("Decrypted message: " + plaintext);
    return plaintext.toString();
  }
}
```