Министерство образования Республики Беларусь

Учреждение образования

“БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ”

Факультет информационных технологий и управления

Кафедра интеллектуальных информационных технологий

Отчет по лабораторной работе №5

по курсу “Средства и методы защиты информации в интеллектуальных системах”

|  |  |
| --- | --- |
| Выполнил:  студент гр. 221702 | Юргилевич Е. В. |
| Проверил: | Крищенович В.А. |

Минск 2024

Лабораторная работа №5

АСИММЕТРИЧНОЕ ШИФРОВАНИЕ И ЭЛЕКТРОННАЯ

ЦИФРОВАЯ ПОДПИСЬ

**Задание:** разработать программное обеспечение, реализующее функции генерации секретного и открытого ключей, шифрования и цифровой подписи для алгоритма RSA. Обмен входными и выходными данными должен осуществляться через файлы:

– открытого ключа;

– секретного ключа;

– исходного сообщения;

– зашифрованного сообщения.

Для повышения скорости шифрования использовать метод

последовательного возведения в квадрат и умножения.

Выполнить тестирование разработанного программного обеспечения на 10 наборах тестовых данных.

Длина чисел p и q должна быть не менее 1024 бит.

**Реализация:**

package org.example.yurhilevich;  
  
import java.io.\*;  
import java.math.BigInteger;  
import java.security.SecureRandom;  
import java.util.ArrayList;  
import java.util.List;  
import java.util.Scanner;  
  
public class RSAMethod {  
 private BigInteger p, q, n, fiN, d, m, c, s;  
 private BigInteger e = BigInteger.*valueOf*(65537);  
  
 public void generatePrimeNumbers() {  
 int bitLength = 1024;  
 SecureRandom random = new SecureRandom();  
 this.p = BigInteger.*probablePrime*(bitLength, random);  
 this.q = BigInteger.*probablePrime*(bitLength, random);  
  
 while (this.p.equals(this.q)) {  
 this.q = BigInteger.*probablePrime*(bitLength, random);  
 }  
  
 System.*out*.println("Первое сгенерированное простое число: " + this.p);  
 System.*out*.println("Второе сгенерированное простое число: " + this.q);  
 System.*out*.println("Длина первого числа в битах: " + this.p.bitLength());  
 System.*out*.println("Длина второго числа в битах: " + this.q.bitLength());  
 }  
  
 public void interimCalculations() {  
 n = this.p.multiply(this.q);  
 BigInteger pMinusOne, qMinusOne;  
 pMinusOne = this.p.subtract(BigInteger.*ONE*);  
 qMinusOne = this.q.subtract(BigInteger.*ONE*);  
  
 this.fiN = pMinusOne.multiply(qMinusOne);  
 this.d = this.e.modInverse(this.fiN);  
  
// setValueM();  
 }  
  
 public void writeInFile(String filename, List<BigInteger> list) {  
 try (BufferedWriter writer = new BufferedWriter(new FileWriter(filename))) {  
 for (BigInteger var : list) {  
 writer.write(var.toString());  
 writer.newLine();  
 }  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 }  
  
 public List<BigInteger> readFromFile(String filename) {  
 try (BufferedReader reader = new BufferedReader(new FileReader(filename))) {  
 List<BigInteger> list = new ArrayList<>();  
 String line;  
 while ((line = reader.readLine()) != null) {  
 list.add(new BigInteger(line));  
 }  
 return list;  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 return null;  
 }  
  
 private void writeDataInFile() {  
 List<BigInteger> list1 = new ArrayList<>();  
 list1.add(this.e);  
 list1.add(this.n);  
 writeInFile("open\_key.txt", list1);  
 List<BigInteger> list2 = new ArrayList<>();  
 list2.add(this.d);  
 list2.add(this.n);  
 writeInFile("secret\_key.txt", list2);  
 list1.clear();  
 list1.add(this.m);  
 writeInFile("source\_message.txt", list1);  
 }  
  
 public void RSAEncryption() {  
 List<BigInteger> list = getOpenKey();  
 List<BigInteger> list1 = getSourceMessage();  
 BigInteger m = list1.get(0);  
 BigInteger e = list.get(0);  
 BigInteger n = list.get(1);  
 BigInteger c = this.modularExponential(m, e, n);  
 System.*out*.println("Зашифрованное сообщение: " + c);  
 List<BigInteger> list2 = new ArrayList<>();  
 list2.add(c);  
 writeInFile("encryption\_message.txt", list2);  
 }  
  
 public void encryption() {  
 generatePrimeNumbers();  
 interimCalculations();  
 writeDataInFile();  
 }  
  
 public BigInteger decryption() {  
 List<BigInteger> list = getSecretKey();  
 List<BigInteger> list1 = getEncryptionMessage();  
 BigInteger c = list1.get(0);  
 BigInteger d = list.get(0);  
 BigInteger n = list.get(1);  
 return modularExponential(c, d, n);  
 }  
  
 public BigInteger signatureDecryption() {  
 List<BigInteger> list = getOpenKey();  
 List<BigInteger> list1 = getEncryptionMessage();  
 BigInteger s = list1.get(0);  
 BigInteger e = list.get(0);  
 BigInteger n = list.get(1);  
 return modularExponential(s, e, n);  
 }  
  
 public void createDigitalSignature() {  
 List<BigInteger> list = getSecretKey();  
 List<BigInteger> list1 = getSourceMessage();  
 BigInteger m = list1.get(0);  
 BigInteger d = list.get(0);  
 BigInteger n = list.get(1);  
 BigInteger s = this.modularExponential(m, d, n);  
 System.*out*.println("Зашифрованное сообщение: " + s);  
 List<BigInteger> list2 = new ArrayList<>();  
 list2.add(s);  
 writeInFile("encryption\_message.txt", list2);  
 }  
  
 public void digitalSignature() {  
 generatePrimeNumbers();  
 interimCalculations();  
 writeDataInFile();  
 }  
  
 public void setValueM() {  
 System.*out*.print("Введите число M: ");  
 Scanner scanner = new Scanner(System.*in*);  
 this.m = scanner.nextBigInteger();  
 }  
  
 public void setValueM(int message){  
 BigInteger bigint = BigInteger.*valueOf*(message);  
 this.m = bigint;  
 }  
  
 private BigInteger modularExponential(BigInteger base, BigInteger exp, BigInteger mod) {  
 BigInteger result = BigInteger.*ONE*;  
 BigInteger baseTWO = BigInteger.*valueOf*(2);  
 while (exp.compareTo(BigInteger.*ZERO*) > 0) {  
 if (exp.mod(baseTWO).equals(BigInteger.*ONE*)) {  
 result = (result.multiply(base)).mod(mod);  
 }  
 exp = exp.divide(baseTWO);  
 base = (base.multiply(base)).mod(mod);  
 }  
 return result;  
 }  
  
 public List<BigInteger> getEncryptionMessage() {  
 System.*out*.println("Получение зашифрованного сообщения");  
 List<BigInteger> list = readFromFile("encryption\_message.txt");  
 System.*out*.println("Зашифрованный текст: " + list.get(0));  
 return list;  
 }  
  
 public List<BigInteger> getOpenKey() {  
 System.*out*.println("Получение открытых ключей");  
 List<BigInteger> list = readFromFile("open\_key.txt");  
 System.*out*.println("Исходный ключ (e): " + list.get(0));  
 System.*out*.println("Исходное ключ (n): " + list.get(1));  
 return list;  
 }  
  
 public List<BigInteger> getSecretKey() {  
 System.*out*.println("Получение закрытых ключей");  
 List<BigInteger> list = readFromFile("secret\_key.txt");  
 System.*out*.println("Секретный ключ (d): " + list.get(0));  
 System.*out*.println("Секретный ключ (n): " + list.get(1));  
 return list;  
 }  
  
 public List<BigInteger> getSourceMessage() {  
 System.*out*.println("Получение исходного сообщения");  
 List<BigInteger> list = readFromFile("source\_message.txt");  
 System.*out*.println("Исходный текст: " + "\u001B[33m" +list.get(0) + "\u001B[0m");  
 return list;  
 }  
}

**Описание основных функций программы:   
1. Генерация простых чисел (функция generatePrimeNumbers)**

Эта функция генерирует случайное простое число размером менее 1024 бит с использованием библиотеки java.math.BigInteger. В RSA алгоритме для генерации ключей используются два таких простых числа — p и q.

**2. Промежуточные вычисления (функция interimCalculations)**

Этот метод вычисляет произведение чисел a и b, а также коэффициенты для вычисления значения функции Эйлера φ(n), значение φ(n), а так же находит обратный элемент d в RSA.

**3. Генерация ключей RSA (функция Encryption)**

1. Генерируются два больших простых числа p и q.
2. Вычисляется модуль n=p\*q, который используется как часть открытого и закрытого ключа.
3. Вычисляется функция Эйлера φ(n)=(p−1)×(q−1).
4. Выбирается значение e=65537 — это распространённое значение для открытого ключа, так как оно маленькое и взаимно простое с φ(n).
5. Вычисляется значение d— закрытая экспонента, обратная к e по модулю φ(n).
6. Ключи записываются в файлы: открытый ключ — в open\_key.txt, закрытый ключ — в secret\_key.txt.
7. Сообщения записывается в файл: исходное — source\_message.txt, зашифрованное — encryption\_message.txt.

**4. Функция возведения в степень по модулю (функция modularExponential)**

Эта функция реализует метод последовательного возведения в квадрат и умножения для быстрого вычисления выражений вида baseexp mod *mod*. Этот метод эффективен для операций шифрования и дешифрования, предотвращая переполнение.

**5. Шифрование RSA (функция RSAEncryption)**

1. Из файла открытого ключа извлекаются значения n и e.
2. Сообщение шифруется с помощью c=me mod n .
3. Результат записывается в файл encryption\_message.txt.

**6. Дешифрование RSA (функция RSADecryption)**

1. Из файла закрытого ключа извлекаются значения n и d.
2. Зашифрованное сообщение считывается из файла.
3. Сообщение дешифруется с помощью output=cd mod  n.
4. Результат возвращается обратно в виде числа типа BigInteger.

**7. Цифровая подпись (функция createDigitalSignature)**

1. Сообщение представляется в виде целого числа
2. Цифровая подпись создается с использованием закрытого ключа s=md mod  n.
3. Подпись записывается в файл encryption\_message.txt.

**8. Проверка цифровой подписи (функция signatureDecryption)**

1. Из файла открытого ключа извлекаются значения n и e.
2. Подпись считывается из файла.
3. Проверяется подпись с использованием output=se mod  n, и сравнивается с исходным сообщением.
4. Результат возвращается обратно в виде числа типа BigInteger.

**9. Тестирование на 10 наборах данных (функция test\_rsa)**

Эта функция тестирует весь процесс на 10 тестовых сообщениях. Каждый тест включает следующие шаги:

1. Задание исходного сообщения:
   * Произвольно создаётся целое число от 0-1000000.
2. Генерация ключей:
   * Вызывается функция encryption которая записывает ключи в файлы, а так же исходное число.
3. Шифрование сообщения:
   * Вызывается функция RSAEncryption, которая шифрует тестовое сообщение и выводит зашифрованное сообщение в консоль.
4. Дешифрование сообщения:
   * Дешифруется сообщение с помощью функции RSADecryption, и выводится расшифрованное сообщение.
5. Создание цифровой подписи:
   * Для сообщения создается цифровая подпись с использованием закрытого ключа, которая выводится в консоль.
6. Проверка подписи:
   * Проверяется, валидна ли подпись, что также выводится в консоль.