Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

КАФЕДРА ИНФОРМАТИКИ

Лабораторная работа № 3

**Компьютерная реализация алгоритмов криптографии с открытым ключом на примере RSA**

Выполнил студент гр. 953501

Трофимук Г.А

Проверил

Протько М.И.

Минск, 2022

# 1. Введение

Реализовать программные средства шифрования и дешифрования текстовых файлов при помощи алгоритма RSA.

**2. Теоретические сведения**

**Алгоритм RSA**

Алгоритм, разработанный Ривестом, Шамиром и Адлеманом, использует выражения с экспонентами. Данные шифруются блоками, каждый блок рассматривается как число, меньшее некоторого числа n. Шифрование и дешифрование имеют следующий вид для некоторого незашифрованного блока М и зашифрованного блока С.

С = Ме (mod n)

M = Cd (mod n) = (Me)d (mod n) = Med (mod n)

Как отправитель, так и получатель должны знать значение n. Отправитель знает значение е, получатель знает значение d. Таким образом, *открытый ключ* есть KU = {e, n} и *закрытый ключ* есть KR = {d, n}. При этом должны выполняться следующие условия:

1.           Возможность найти *е*, *d* и *n* такие, что *Med = M mod n* для всех    М < n .

2.           Относительная легкость вычисления Ме и Сd для всех значений   М < n.

3.           Невозможность определить d, зная е и n.

**Рассмотрим некоторые математические понятия, свойства и теоремы, которые позволят нам определить e, d и n.**

1.           Если (а · b) ≡ (a · c) mod n, то b ≡ c mod n,

если а и n взаимнопростые, т.е gcd (a, n) = 1.

2.           Обозначим Zp - все числа, взаимнопростые с p и меньшие p. Если p - простое, то Zp - это все остатки. Обозначим w-1 такое число, что w · w-1 ≡ 1 mod p.

Тогдаw Zp z: w · z ≡1 mod p

3.           Определим функцию Эйлера следующим образом: Φ(n) - число положительных чисел, меньших n и взаимнопростых с n. Если p - простое, то Φ(р) = p-1.

Если p и q - простые, то Φ(p · q) = (p-1) · (q-1).

4.           *Теорема Ферма*.

an-1 ≡1 mod n, если n - простое.

5.           *Теорема Эйлера*.

aΦ(n) ≡1 mod n для всех взаимнопростых a и n.

Теперь рассмотрим все элементы *алгоритма RSA*.

|  |  |
| --- | --- |
| p, q - два простых целых числа | -открыто, вычисляемо. |
| n = p · q | - закрыто, вычисляемо. |
| d, gcd (Φ(n), d) = 1; | - открыто, выбираемо. |
| 1 < d < Φ(n) |
| е ≡ d-1 mod Φ(n) | - закрыты, выбираемы. |

*Закрытый ключ* состоит из {d, n}, *открытый ключ* состоит из {e, n}. Предположим, что пользователь А опубликовал свой *открытый ключ*, и что пользователь В хочет послать пользователю А сообщение М. Тогда В вычисляет С = Ме (mod n) и передает С. При получении этого зашифрованного текста пользователь А дешифрует вычислением                   М = С d (mod n).

Суммируем *алгоритм RSA*:

**Создание ключей**

|  |
| --- |
| 1)    Выбрать простые р и q |
| 2)    Вычислить n = p · q |
| 3)    Выбрать d     gcd (Φ(n), d) = 1; 1 < d < Φ(n) |
| 4)    Вычислить е     е = d-1 mod Φ(n) |
| 5)    *Открытый ключ* KU = {e, n} |
| 6)    *Закрытый ключ* KR = {d, n} |

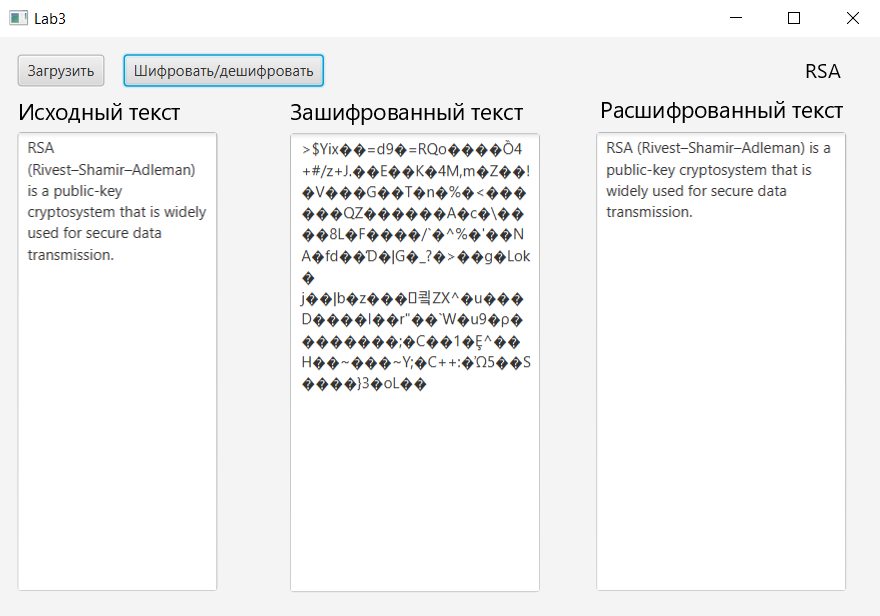
**Шифрование**

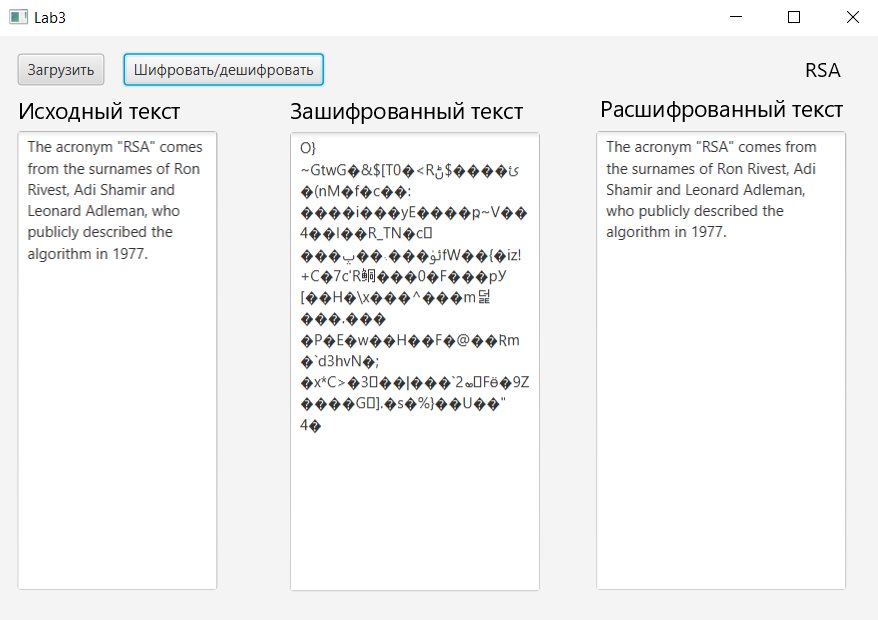
|  |
| --- |
| 1)    Незашифрованный текст: М < n |
| 2)    Зашифрованный текст: С = М е (mod n) |

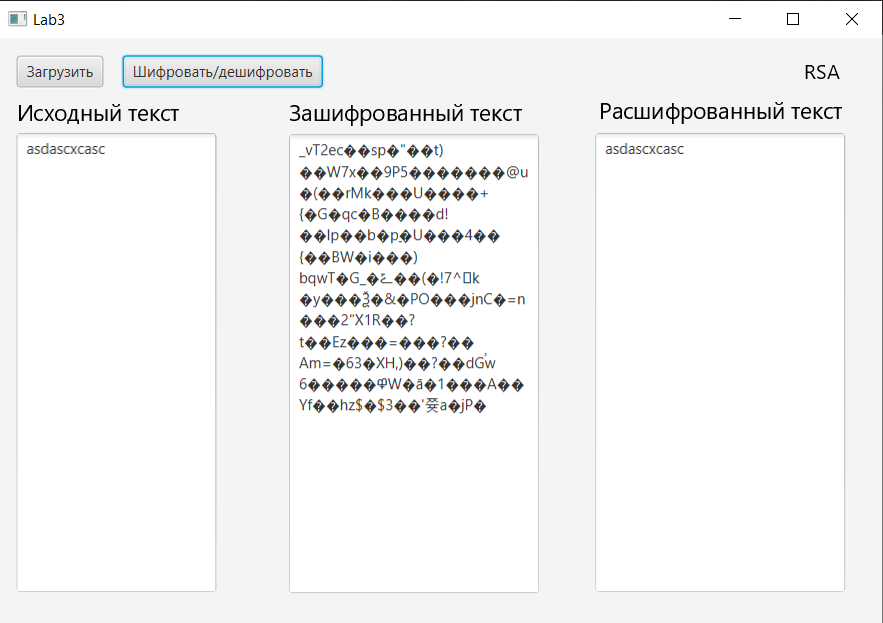
**Дешифрование**

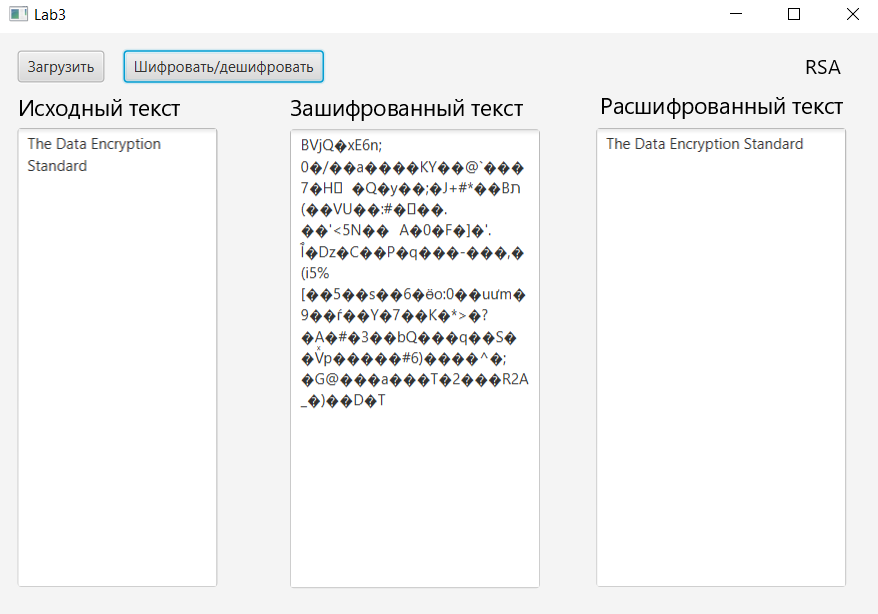
|  |
| --- |
| 1)    Зашифрованный текст: С |
| 2)    Незашифрованный текст: М = Сd (mod n) |
|  |

# 3. Результаты выполнения программы









# Выводы

RSA — криптографический алгоритм с открытым ключом, основывающийся на вычислительной сложности задачи факторизации больших целых чисел.

Криптосистема RSA стала первой системой, пригодной и для шифрования, и для цифровой подписи. Алгоритм используется в большом числе криптографических приложений, включая PGP, S/MIME, TLS/SSL, IPSEC/IKE и других.

Система RSA используется для защиты программного обеспечения и в схемах цифровой подписи.

Также она используется в открытой системе шифрования PGP и иных системах шифрования (к примеру, DarkCryptTC и формат xdc) в сочетании с симметричными алгоритмами.

Из-за низкой скорости шифрования сообщения обычно шифруют с помощью более производительных симметричных алгоритмов со случайным сеансовым ключом (например, AES, IDEA, Serpent, Twofish), а с помощью RSA шифруют лишь этот ключ, таким образом реализуется гибридная криптосистема. Такой механизм имеет потенциальные уязвимости ввиду необходимости использовать криптографически стойкий генератор псевдослучайных чисел для формирования случайного сеансового ключа симметричного шифрования.

# КОД ПРОГРАММЫ

**Main.java:**

package com.example.lab3si;  
  
import javafx.application.Application;  
import javafx.fxml.FXMLLoader;  
import javafx.scene.Scene;  
import javafx.stage.Stage;  
import org.bouncycastle.jce.provider.BouncyCastleProvider;  
  
import java.io.IOException;  
import java.security.Security;  
  
public class Main extends Application {  
 @Override  
 public void start(Stage stage) throws IOException {  
 FXMLLoader fxmlLoader = new FXMLLoader(Main.class.getResource("view.fxml"));  
 Scene scene = new Scene(fxmlLoader.load());  
 stage.setTitle("Lab3");  
 stage.setScene(scene);  
 stage.show();  
 Controller controller = fxmlLoader.getController();  
 controller.setStage(stage);  
 Security.*addProvider*(new BouncyCastleProvider());  
 }  
  
 public static void main(String[] args) {  
 *launch*();  
 }  
}

**Controller.java:**

package com.example.lab3si;  
  
import javafx.event.ActionEvent;  
import javafx.fxml.FXML;  
import javafx.scene.control.TextArea;  
import javafx.stage.FileChooser;  
import javafx.stage.Stage;  
  
import javax.crypto.BadPaddingException;  
import javax.crypto.Cipher;  
import javax.crypto.IllegalBlockSizeException;  
import javax.crypto.NoSuchPaddingException;  
import java.io.File;  
import java.io.FileReader;  
import java.io.IOException;  
import java.nio.charset.StandardCharsets;  
import java.security.\*;  
  
public class Controller {  
 private Stage stage;  
 @FXML  
 private TextArea sourceTextArea;  
 @FXML  
 private TextArea encryptRSATextArea;  
 @FXML  
 private TextArea decryptRSATextArea;  
  
 public Stage getStage() {  
 return stage;  
 }  
  
 public void setStage(Stage stage) {  
 this.stage = stage;  
 }  
 @FXML  
 void Load(ActionEvent event) throws IOException {  
 final FileChooser fileChooser = new FileChooser();  
 fileChooser.setInitialDirectory(new File("C:\\Users\\User\\Desktop\\Tests\\Txts"));  
 File file = fileChooser.showOpenDialog(stage);  
 if (file != null) {  
 FileReader reader = new FileReader(file.getAbsolutePath());  
 StringBuilder result = new StringBuilder();  
 int character;  
 while((character=reader.read())!=-1)  
 {  
 result.append((char) character);  
 }  
 sourceTextArea.setText(result.toString());  
 }  
 }  
 @FXML  
 void EncryptDecrypt(ActionEvent event) throws NoSuchAlgorithmException, NoSuchPaddingException, InvalidKeyException, NoSuchProviderException, IllegalBlockSizeException, BadPaddingException {  
 Cipher cipher = Cipher.*getInstance*("RSA/ECB/PKCS1Padding", "BC");  
 String message = sourceTextArea.getText();  
  
 KeyPairGenerator kpg = KeyPairGenerator.*getInstance*("RSA");  
 kpg.initialize(2048);  
 KeyPair kp = kpg.generateKeyPair();  
 Key pub = kp.getPublic();  
 Key pvt = kp.getPrivate();  
  
 cipher.init(Cipher.*ENCRYPT\_MODE*, pub);  
 byte[] encryptedText = cipher.doFinal(message.getBytes());  
 encryptRSATextArea.setText(new String(encryptedText, StandardCharsets.*UTF\_8*));  
  
 cipher.init(Cipher.*DECRYPT\_MODE*, pvt);  
 byte[] decryptedText = cipher.doFinal(encryptedText);  
 decryptRSATextArea.setText(new String(decryptedText, StandardCharsets.*UTF\_8*));  
 }  
}