Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования

«Пермский национальный исследовательский политехнический университет»

Электротехнический факультет

Кафедра «Информационные технологии и автоматизированные системы»

Направление 09.03.01 – «Информатика и вычислительная техника»

Дисциплина: «Защита информации»

Профиль: «Автоматизированные системы обработки информации и управления»

Семестр 7

ОТЧЕТ

по лабораторной работе №3

Тема: «Несимметричные алгоритмы шифрования»

Выполнил: студент группы АСУ-14-1б

Калмыков В. А. \_\_\_\_\_\_\_\_\_\_\_

Проверил: доцент кафедры ИТАС

Шереметьев В. Г. \_\_\_\_\_\_\_\_\_\_\_

Дата \_\_\_\_\_\_

Пермь, 2017

**ЦЕЛЬ РАБОТЫ**

Получить практические навыки по использованию несимметричных алгоритмов шифрования, на примере использования алгоритма Диффи-Хеллмана и Эль-Гамаля, а так же по применению электронных подписей на примере метода Эль-Гамаля.

**ЗАДАНИЕ**

**Вариант №14**. Создать электронную подпись текстового сообщения длиной не меньшей 256 символов, методом Эль-Гамаля, используя в качестве x и g простые числа с разрядностью не меньшей двенадцати и p и k не менее двадцати.

**ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ**

**Алгоритм Диффи-Хеллмана:**

Алгоритм назван по фамилиям его создателей Диффи (Diffie) и Хеллмана (Hellman).

Метод помогает обмениваться секретным ключом для симметричных криптосистем, но использует метод, очень похожий на асимметричный алгоритм RSA. Это не симметричный алгоритм, так как для шифрования и дешифрования используются различные ключи. Так же это не схема с открытым ключом, потому что ключи легко получаются один из другого, и ключ шифрования и ключ дешифрования должны храниться в секрете.

Определим круг его возможностей. Предположим, что двум абонентам необходимо провести конфиденциальную переписку, а в их распоряжении нет первоначально оговоренного секретного ключа. Однако, между ними существует канал, защищенный от модификации, то есть данные, передаваемые по нему, могут быть прослушаны, но не изменены (такие условия имеют место довольно часто). В этом случае две стороны могут создать одинаковый секретный ключ, ни разу не передав его по сети, по следующему алгоритму.

Предположим, что обоим абонентам известны некоторые два числа v и q. Они, впрочем, известны и всем остальным заинтересованным лицам. Например, они могут быть просто фиксированно «зашиты» в программное обеспечение. Далее один из партнеров P1 генерирует случайное или псевдослучайное простое число x и посылает другому участнику будущих обменов P2 значение A = qx mod n

По получении А партнер P2 генерирует случайное или псевдослучайное простое число у и посылает P2 вычисленное значение B = qy mod n

Партнер P1, получив В, вычисляет Kx = Bx mod n, а партнер P2 вычисляет Ky = Ay mod n. Алгоритм гарантирует, что числа Ky и Kx равны и могут быть использованы в качестве секретного ключа для шифрования. Ведь даже перехватив числа А и В, трудно вычислить Kx или Ky.

Например, по вычисленным Kx =Ky=K абоненты могут зашифровать сообщение M=123 по следующему алгоритму: к каждому символу сообщения M добавить K => сообщение С=234, при K=1. Соответственно алгоритмом расшифрования будет разность ключа K из каждого символа сообщения C.

Пример:

Пусть

n=3; q=5;

x=5; y=7;

тогда A = q^x mod n = 1, а B = q^y mod n = 2, то вычислив Kx = B^x mod n и Ky = A^y mod n получим Kx= Ky=1. Зашифруем приведенное выше сообщение M=123 по приведенному выше алгоритму => сообщение С=234, расшифровав сообщение C по обратному алгоритму получим сообщение M=123.

Необходимо еще раз отметить, что алгоритм Диффи-Хеллмана работает только на линиях связи, надежно защищенных от модификации. Если бы он был применим на любых открытых каналах, то давно снял бы проблему распространения ключей и, возможно, заменил собой всю асимметричную криптографию. Однако, в тех случаях, когда в канале возможна модификация данных, появляется очевидная возможность вклинивания в процесс генерации ключей «злоумышленника-посредника» по той же самой схеме, что и для асимметричной криптографии.

**Алгоритм Эль-Гамаля:**

Алгоритм Эль-Гамаля может использоваться для формирования электронной подписи или для шифрования данных. Он базируется на трудности вычисления дискретного логарифма. Для генерации пары ключей сначала берется простое число p и два случайных простых числа g и x, каждое из которых меньше p. Затем вычисляется:

y = g^x mod p

Общедоступными ключами являются y, g и p, а секретным ключом является х. Для подписи сообщения M выбирается случайное число k, которое является простым по отношению к p-1. После этого вычисляется a = gk mod p. Далее из уравнения M = (xa + kb) mod (p-1) находим b. Электронной подписью для сообщения M будет служить пара a и b. Случайное число k следует хранить в секрете. Для верификации подписи необходимо проверить равенство:

y^a \* a^b mod p = g^M mod p.

Пример:

Выберем p=11, g=2, а закрытый ключ x=8.

Вычислим y = g^x mod p = 3.

Открытым ключом являются p=11, g=2, y=3, чтобы подписать M=5 сначала выберем случайное число k=9. Вычисляем и с помощью расширенного алгоритма Эвклида находим:

M = (xa + kb) mod (p - 1)

5 = (8\*6 + 9 \* b) mod 10

Решение: b=3, а подпись представляет собой пару a=6 и b=3.

**Шифрование Эль-Гамаля:**

Модификация алгоритма позволяет шифровать сообщения. Для шифрования сообщения M сначала выбирается случайное число k, взаимно простое с p-1, затем вычисляются

a = g^k mod p

b = y^kM dom p

Пара a и b представляют собой зашифрованный текст. Следует заметить, что зашифрованный текст имеет размер в два раза больше исходного. Для дешифрования производится вычисление:

M = b/a^x mod p.

Пример:

Вернемся к предыдущему примеру:

a=6, b = y^kM mod p = 3.

Расшифруем: M = b/a^x mod p = 5.

**ХОД РАБОТЫ**

На рисунке 1 представлена главная форма программы.

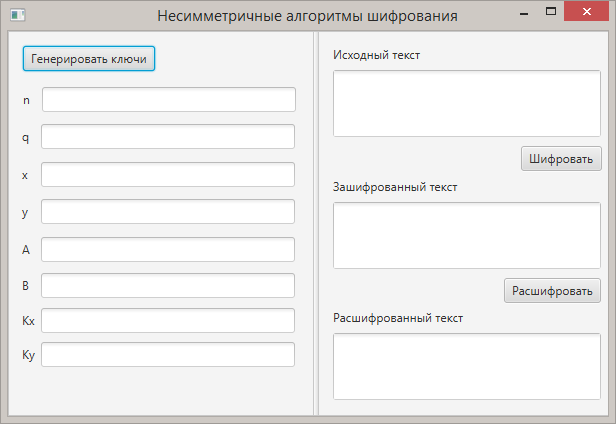


Рисунок 1 – Главная форма программы.

Пример работы программы представлен на рисунке 2.

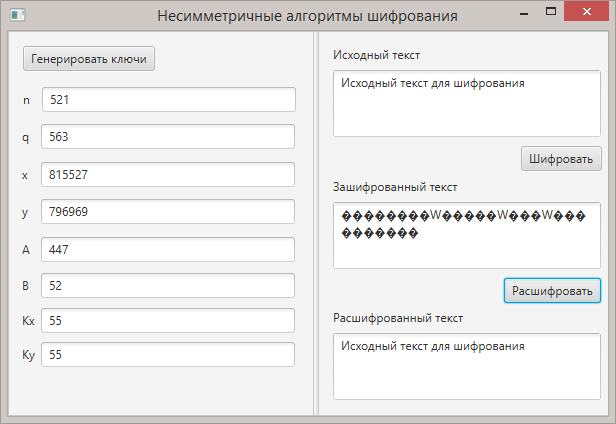


Рисунок 2 – Пример работы программы.

**ПРИЛОЖЕНИЕ А**

**Листинг класса AsymmetricEncryption.java**

package Models;

import java.math.BigInteger;

import java.util.Random;

public class AsymmetricEncryption {

private static final int CAP = 20;

private BigInteger n;

private BigInteger q;

private BigInteger x;

private BigInteger y;

private BigInteger kx;

private BigInteger ky;

private BigInteger A;

private BigInteger B;

public AsymmetricEncryption() {

Random rnd = new Random();

this.n = BigInteger.probablePrime(AsymmetricEncryption.CAP / 2, rnd);

this.q = BigInteger.probablePrime(AsymmetricEncryption.CAP / 2, rnd);

this.x = BigInteger.probablePrime(AsymmetricEncryption.CAP, rnd);

this.y = BigInteger.probablePrime(AsymmetricEncryption.CAP, rnd);

this.A = this.q.pow(this.x.intValue()).mod(this.n);

this.B = this.q.pow(this.y.intValue()).mod(this.n);

this.kx = this.B.pow(this.x.intValue()).mod(this.n);

this.ky = this.A.pow(this.y.intValue()).mod(this.n);

}

public BigInteger getY() {

return y;

}

public BigInteger getN() {

return n;

}

public BigInteger getQ() {

return q;

}

public BigInteger getA() {

return A;

}

public BigInteger getB() {

return B;

}

public BigInteger getKx() {

return kx;

}

public BigInteger getKy() {

return ky;

}

public BigInteger getX() {

return x;

}

public byte encrypt(byte part) {

return (byte) (part + this.kx.byteValue());

}

public byte decrypt(byte part) {

return (byte) (part - this.ky.byteValue());

}

}

**ПРИЛОЖЕНИЕ Б**

**Листинг класса SampleController.java**

package Forms;

import Models.AsymmetricEncryption;

import javafx.fxml.FXML;

import javafx.scene.control.TextArea;

import javafx.scene.control.TextField;

public class SampleController {

@FXML

private TextField txtN;

@FXML

private TextField txtQ;

@FXML

private TextField txtX;

@FXML

private TextField txtY;

@FXML

private TextField txtA;

@FXML

private TextField txtB;

@FXML

private TextField txtKx;

@FXML

private TextField txtKy;

@FXML

private TextArea txtText;

@FXML

private TextArea txtEncryptText;

@FXML

private TextArea txtDecryptText;

private AsymmetricEncryption ae;

private byte[] encryptedMessage;

@FXML

public void initialize() {

}

@FXML

private void handleBtnKeygen() {

this.ae = new AsymmetricEncryption();

this.txtN.setText(this.ae.getN().toString());

this.txtQ.setText(this.ae.getQ().toString());

this.txtX.setText(this.ae.getX().toString());

this.txtY.setText(this.ae.getY().toString());

this.txtA.setText(this.ae.getA().toString());

this.txtB.setText(this.ae.getB().toString());

this.txtKx.setText(this.ae.getKx().toString());

this.txtKy.setText(this.ae.getKy().toString());

}

@FXML

private void handleBtnEncrypt() {

String text = txtText.getText();

byte[] txtBytes = text.getBytes();

this.encryptedMessage = new byte[txtBytes.length];

for(int i = 0; i < txtBytes.length; i++) {

encryptedMessage[i] = ae.encrypt(txtBytes[i]);

}

this.txtEncryptText.setText(new String(this.encryptedMessage));

}

@FXML

private void handleBtnDecrypt() {

byte[] decryptedMessage = new byte[this.encryptedMessage.length];

for(int i = 0; i < this.encryptedMessage.length; i++) {

decryptedMessage[i] = this.ae.decrypt(this.encryptedMessage[i]);

}

this.txtDecryptText.setText(new String(decryptedMessage));

}

}

**ПРИЛОЖЕНИЕ В**

**Листинг класса Main.java**

import javafx.application.Application;

import javafx.fxml.FXMLLoader;

import javafx.scene.Parent;

import javafx.scene.Scene;

import javafx.stage.Stage;

public class Main extends Application {

@Override

public void start(Stage primaryStage) throws Exception{

Parent root = FXMLLoader.load(getClass().getResource("/Forms/sample.fxml"));

primaryStage.setTitle("Несимметричные алгоритмы шифрования");

primaryStage.setScene(new Scene(root));

primaryStage.show();

}

public static void main(String[] args) {

launch(args);

}

}