Федеральное агентство связи

Ордена Трудового Красного Знамени федеральное государственное бюджетное образовательное учреждение высшего образования

«Московский технический университет связи и информатики»

Кафедра «Информационная безопасность»

Отчёт по лабораторной работе №2

по дисциплине

«Методы и средства защиты компьютерной информации»

Изучение алгоритмов асимметричного шифрования

Выполнил:

студент группы БВТ1301

Редько Е.Ю.

Проверил:

ассистент кафедры ИБ

Барков В.В.

Москва 2016

## Цель работы

Изучить принципы работы алгоритмов асимметричного шифрования на примере реализации одного из алгоритмов.

## Задание

Используя один из объектно-ориентированных языков программирования реализовать алгоритм асимметричного шифрования, указанный в индивидуальном варианте.

Проект реализовать в виде библиотеки классов. При реализации использовать интерфейсы. Алгоритм должен быть универсальным с точки зрения данных для шифрования (строка, файл, сокет).

Для созданного проекта необходимо написать модульные тесты, показывающие корректность работы алгоритма.

## Описание алгоритма RSA

В алгоритме RSA используется пара из открытого и секретного ключей, выбираемых следующим образом:

1. Выбрать простые числа p и q
2. Вычислить n = p \* q
3. Вычислить m = (p - 1) \* (q - 1)
4. Выбрать число d взаимно простое с m
5. Выбрать число e так, чтобы e \* d = 1 (mod m)

Числа e и d, поставленные в пару с n, являются ключами RSA. Шифруемые данные необходимо разбить на блоки — числа от 0 до n-1. Шифрование и дешифровка каждого блока производятся следующим образом:

* Шифрование: b = ae (mod n)
* Дешифровка: a = bd (mod n)

Следует также отметить, что ключи e и d равноправны, то есть сообщение можно шифровать как ключом e, так и ключом d, при этом расшифровка должна быть произведена с помощью другого ключа.

## Схема алгоритма шифрования RSA

D:\downloads\Untitled Diagram.png

**Код программы**

IRSA.cs (интерфейс):

namespace RedkoLib

{

public interface IRSA

{

void dataToRsa(Stream sourceData, Stream returnData, RSAKey publicKey);

void rsaToData(Stream sourceData, Stream returnData, RSAKey privateKey);

}

}

RSA.cs:

namespace RedkoLib

{

public struct RSAKey

{

public double first;

public double n;

public RSAKey(double first1, double n1)

{

first = first1;

n = n1;

}

}

public class cypherRSAclass : IRSA

{

public void dataToRsa(Stream sourceData, Stream returnData, RSAKey publicKey)

{

RSAClass bufferRSA = new RSAClass();

int sourceByte;

while ((sourceByte = sourceData.ReadByte()) != -1)

{

returnData.WriteByte(bufferRSA.RSADataCrypt((byte)sourceByte, publicKey));

}

}

public void rsaToData(Stream sourceData, Stream returnData, RSAKey privateKey)

{

RSAClass bufferRSA = new RSAClass();

int sourceByte;

while ((sourceByte = sourceData.ReadByte()) != -1)

{

returnData.WriteByte(bufferRSA.RSADataDecrypt((byte)sourceByte, privateKey));

}

}

}

public class RSAClass

{

public byte RSADataCrypt(byte sourceByte, RSAKey publicKey)

{

BigInteger result = BigInteger.Pow(sourceByte, (int)publicKey.first) % (BigInteger)publicKey.n;

return (byte)result;

}

public byte RSADataDecrypt(byte sourceByte, RSAKey privateKey)

{

BigInteger result = BigInteger.Pow(sourceByte, (int)privateKey.first) % (BigInteger)privateKey.n;

return (byte)result;

}

}

}

TRSA.cs (тест):

namespace RedkoTest

{

[TestClass]

public class TRSA

{

//Проверяет, получаем ли мы строку, эквивалентную исходной, после зашифровки и расшифровки

//По сути, полное тестирование, но в случае успешного прохождения теста EncryptedEqualeStandard

//автоматически становится проверкой только расшифровки

[TestMethod]

public void RSASourceEqualeResult()

{

RedkoLib.cypherRSAclass bufferRSA = new RedkoLib.cypherRSAclass();

RedkoLib.RSAKey publicKey = new RedkoLib.RSAKey(17, 249);

RedkoLib.RSAKey privateKey = new RedkoLib.RSAKey(29, 249);

Stream encryptedDataStream = new MemoryStream();

Stream decryptedDataStream = new MemoryStream();

string sourceDataString = "test";

MemoryStream sourceDataStream = new MemoryStream(Encoding.Default.GetBytes(sourceDataString));

bufferRSA.dataToRsa(sourceDataStream, encryptedDataStream, publicKey);

encryptedDataStream.Position = 0;

bufferRSA.rsaToData(encryptedDataStream, decryptedDataStream, privateKey);

StreamReader reader = new StreamReader(decryptedDataStream);

decryptedDataStream.Position = 0;

string decryptedDataString = reader.ReadToEnd();

Assert.AreEqual(sourceDataString, decryptedDataString);

}

//Проверка соответствия зашифрованной строки той, что должна была получиться при правильной реализации алгоритма

[TestMethod]

public void RSAEncryptedEqualeStandard()

{

RedkoLib.cypherRSAclass bufferRSA = new RedkoLib.cypherRSAclass();

RedkoLib.RSAKey publicKey = new RedkoLib.RSAKey(17, 249);

RedkoLib.RSAKey privateKey = new RedkoLib.RSAKey(29, 249);

Stream encryptedDataStream = new MemoryStream();

string sourceDataString = "test";

MemoryStream sourceDataStream = new MemoryStream(Encoding.Default.GetBytes(sourceDataString));

bufferRSA.dataToRsa(sourceDataStream, encryptedDataStream, publicKey);

encryptedDataStream.Position = 0;

StreamReader reader = new StreamReader(encryptedDataStream);

string encryptedDataString = reader.ReadToEnd();

Assert.AreEqual(encryptedDataString, "\_ [\_");

}

}

}

