МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РФ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ

ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«ВЯТСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

ИНСТИТУТ МАТЕМАТИКИ И ИНФОРМАЦИОННЫХ СИСТЕМ

ФАКУЛЬТЕТ КОМПЬЮТЕРНЫХ И ФИЗИКО-МАТЕМАТИЧЕСКИХ НАУК

КАФЕДРА ПРИКЛАДНОЙ МАТЕМАТИКИ И ИНФОРМАТИКИ

Отчёт по лабораторной работе №3 по дисциплине

“Методы защиты информации”

Выполнил студент группы ПМИб-4301-01-00 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/ Д. А. Смирнов /

Проверил преподаватель кафедры ПМИ   \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ / А.Г. Татаринова /

Киров 2020

# Цель

Реализовать программу, реализующую криптографические асимметричные шифры, основывающиеся на вычислительной сложности задачи факторизации больших целых чисел.

# Задача о рюкзаке

# Генерация ключей

Во-первых, выбирается некоторый сверхрастущий рюкзак – это закрытый ключ. Затем выбирается число m, такое, что m > (C1+C2+...+Cn). Далее выбирается число n, взаимно простое с m. На основании сверхрастущего рюкзака формируется обычный, не сверхрастущий рюкзак – это будет открытый ключ (O1, O2, ... , On) – по следующему правилу:

.

# Алгоритм шифрования

Открытый текст представляется в виде двоичной последовательности. Последовательность разбивается на блоки длины, не большей n. Каждый блок (x1, x2, ... , xn) заменяется на число X, вычисляемое по правилу

# Алгоритм дешифровки

Для дешифрования необходимо найти значение, обратное n по модулю m. Это делается с помощью расширенного алгоритм Евклида.

Затем каждое значение зашифрованного текста умножается на n-1 по модулю m и для получившегося значения решается рюкзачная задача (используется закрытый ключ – сверхрастущий рюкзак), в результате чего получается соответствующий блок расшифрованного текста.

# RSA

В алгоритме RSA также используются открытый и закрытый ключи. Оба генерируются на основе пары простых чисел p и q.

# ***Генерация ключей***

Открытый ключ представляет собой пару чисел n и e.

n = p \* q.

e – любое число, взаимно простое с (p-1)(q-1).

Закрытый ключ представляет собой пару чисел d и n.

d = e-1 mod n.

# Алгоритм шифрования

Для шифрования сообщения m оно разбивается на цифровые блоки, меньшие n (для двоичных данных выбирается самая большая степень числа 2, меньшая n). При необходимости последний блок дополняется нулями. Зашифрованное сообщение c будет состоять из блоков ci той же самой длины. Формула шифрования выглядит так

# Алгоритм дешифрования

Для расшифровки сообщения каждый блок mi расшифрованного текста вычисляется по формуле

# Выводы

В ходе лабораторной работы были изучены алгоритмы асимметричного шифрования данных RSA и алгоритм, использующий идею задачи о рюкзаке. На практике же используется алгоритм RSA.

# Листинг

Файл Program.cs, содержащий точку входа приложения и использование методов из классов ByKnapsack, RSA, RSANet, содержащихся соответственно в классах ByKnapsack.cs, RSA.cs, RSANet.cs:

using System;

using System.Collections.Generic;

using System.IO;

using System.Numerics;

using System.Security.Cryptography;

using System.Text;

using System.Linq;

namespace lab3

{

class Program

{

static int GetInt(string warning)

{

bool ka; int p;

do {

string s = Console.ReadLine();

ka = Int32.TryParse(s, out p);

if (ka == false || p < 1)

Console.WriteLine(warning);

} while (ka == false || p < 1);

return p;

}

static void Main(string[] args)

{

Console.WriteLine("Какой способ ассиметричного шифрования вы хотите использовать? " +

"Нажмите R, если вы хотите использовать самописный RSA, K, если вы хотите использовать алгоритм на основе задачи об упакове рюкзака, и N, если хотите использовать библиотечный RSA");

char ch;

do {

ch = Console.ReadKey().KeyChar;

Console.CursorLeft = 1;

if (ch == 'R')

{ Task2(); break; }

else if (ch == 'K')

{ Task1(); break; }

else if (ch == 'N')

{ Task3(); break; }

} while (true);

Console.ReadKey();

}

static void Task2()

{

// Запрашиваем пару чисел (p, q) ключ

Console.WriteLine("Введите значение простого числа p:");

int p = GetInt("Введите корректное положительное число.");

Console.WriteLine("Введите значение простого числа q:");

int q = GetInt("Введите корректное положительное число.");

// Генерируем закрытый и открытый ключи

Tuple<BigInteger, BigInteger, BigInteger> T1 = RSA.GenerateKeys(p, q);

Console.WriteLine("На основании пары чисел (p, q) были сгенерированы приватный и публичный ключи.");

Console.WriteLine("Приватный ключ (d, n):");

Console.WriteLine($"d = {T1.Item1}, n = {T1.Item2}");

Console.WriteLine("Публичный ключ (e, n):");

Console.WriteLine($"e = {T1.Item3}, n = {T1.Item2}");

// Запрашиваем путь к файлу с текстом, который необходимо зашифровать

Console.WriteLine("Укажите имя текстового файла с текстом, который необходимо зашифровать:");

string path;

string textBefore;

do {

try

{

path = Console.ReadLine();

textBefore = File.ReadAllText(path);

break;

}

catch

{

Console.WriteLine("Указанный вами файл не существует =(");

Console.WriteLine("Попробуйте снова");

}

} while (true);

// Шифруем и показываем текст с помощью публичного ключа:

List<BigInteger> cryptoText = RSA.Encrypt(T1.Item2, T1.Item3, textBefore);

Console.WriteLine("Зашифрованное сообщение:");

for (int i = 0; i < cryptoText.Count; i++)

Console.Write(cryptoText[i].ToString() + " ");

Console.WriteLine();

// Расшифровываем и показываем зашифрованный текст с помощью приватного ключа

string text = RSA.Decrypt(T1.Item1, T1.Item2, cryptoText);

Console.WriteLine("Расшифрованный текст:");

Console.WriteLine(text);

}

static void Task1()

{

// Запрашиваем приватный ключ

Console.WriteLine("Введите приватный ключ. Во-первых, укажите длину ключа:");

int size = GetInt("Введите корректное положительное число.");

List<int> privateKey = new List<int>(size);

for (int i = 0; i < size; i++)

{

Console.WriteLine("Введите " + i + "-е число приватного ключа");

privateKey.Add(GetInt("Введите корректное положительное число."));

}

// Проверяем публичный ключ на корректность

bool isOk = ByKnapsack.Check(privateKey);

if (!isOk)

{

Console.WriteLine("Введённый вами ключ не является сверхвозрастающей последовательностью =(");

Console.ReadKey();

return;

}

// Генерируем и показываем публичный ключ

Tuple<List<int>, int, int> T1 = ByKnapsack.GeneratePublicKey(privateKey);

List<int> publicKey = T1.Item1;

int m = T1.Item2, n = T1.Item3;

Console.WriteLine("На основании приватного ключа сгенерировался следующий публичный ключ:");

for (int i = 0; i < publicKey.Count; i++)

Console.Write(publicKey[i].ToString() + " ");

Console.WriteLine();

// Запрашиваем путь к файлу с текстом, который необходимо зашифровать

Console.WriteLine("Укажите имя текстового файла с текстом, который необходимо зашифровать:");

string path;

string textBefore;

do {

try

{

path = Console.ReadLine();

textBefore = File.ReadAllText(path);

break;

}

catch

{

Console.WriteLine("Указанный вами файл не существует =(");

Console.WriteLine("Попробуйте снова");

}

} while (true);

// Шифруем и показываем текст с помощью публичного ключа:

List<int> cryptoText = ByKnapsack.Encrypt(textBefore, publicKey);

Console.WriteLine("Зашифрованное сообщение:");

for (int i = 0; i < cryptoText.Count; i++)

Console.Write(cryptoText[i].ToString() + " ");

Console.WriteLine();

// Расшифровываем и показываем зашифрованный текст с помощью приватного ключа

string text = ByKnapsack.Decrypt(cryptoText, privateKey, n, m);

Console.WriteLine("Расшифрованный текст:");

Console.WriteLine(text);

}

static void Task3()

{

using (RSACryptoServiceProvider rsa\_csp = new RSACryptoServiceProvider(2048))

{

Console.WriteLine("Хотите использовать уже имеющиеся приватный и публичный ключи (Y)? " +

"Нажмите (N), если хотите сгенерировать новые. В этом случае они сохранятся в файл xmlstring.txt.");

char ch;

do {

ch = Console.ReadKey().KeyChar;

Console.CursorLeft = 1;

if (ch == 'Y')

{

// Запрашиваем путь к файлу с ключами

Console.WriteLine("Укажите имя текстового файла с ключами:");

string path;

string xmlString;

do {

try

{

path = Console.ReadLine();

xmlString = File.ReadAllText(path);

break;

}

catch

{

Console.WriteLine("Указанный вами файл не существует =(");

Console.WriteLine("Попробуйте снова");

}

} while (true);

rsa\_csp.FromXmlString(xmlString);

break;

}

else if(ch == 'N')

{

string xmlString = rsa\_csp.ToXmlString(true);

File.WriteAllText("xmlstring.txt", xmlString);

break;

}

} while (true);

// Запрашиваем путь к файлу с текстом, который необходимо зашифровать

Console.WriteLine("Укажите имя текстового файла с текстом, который необходимо зашифровать:");

string path2;

string textBefore;

do {

try

{

path2 = Console.ReadLine();

textBefore = File.ReadAllText(path2);

break;

}

catch

{

Console.WriteLine("Указанный вами файл не существует =(");

Console.WriteLine("Попробуйте снова");

}

} while (true);

// Зашифровываем текст блоками по 224 байта в соответствии с открытым ключом

List<byte> encryptedData = new List<byte>();

for (int i = 0; i < textBefore.Length; i += RSANet.blockSize / 2)

{

string curStr;

if (i + RSANet.blockSize / 2 > textBefore.Length)

curStr = textBefore.Substring(i);

else

curStr = textBefore.Substring(i, RSANet.blockSize / 2);

byte[] curByteBlock = RSANet.StringToBytes(curStr); // ByteConverter.GetBytes(curStr);

byte[] curEncryptedData = RSANet.Encrypt(curByteBlock, rsa\_csp.ExportParameters(false), false);

encryptedData.InsertRange(encryptedData.Count, curEncryptedData);

}

// Выводим зашифрованное сообщение на экран

Console.WriteLine("Зашифрованное сообщение:");

for (int i = 0; i < encryptedData.Count; i++)

Console.Write(encryptedData[i] + " ");

Console.WriteLine();

// Расшифровываем текст в соответствии с закрытым ключом

StringBuilder decryptedData = new StringBuilder();

int blockSize = 256;

for (int i = 0; i < encryptedData.Count; i += blockSize)

{

byte[] curByteBlock = encryptedData.Skip(i).Take(blockSize).ToArray();

byte[] curDecryptedData = RSANet.Decrypt(curByteBlock, rsa\_csp.ExportParameters(true), false);

decryptedData.Append(RSANet.BytesToString(curDecryptedData)); // Encoding.Un.GetString(curDecryptedData));

}

// Выводим расшифрованное сообщение на экран

Console.WriteLine("Расшифрованный текст:");

Console.WriteLine(decryptedData);

}

}

}

}

Файл ByKnapsack.cs, содержащий реализацию алгоритма асимметричного шифрования на основе задачи о рюкзаке:

using System;

using System.Collections.Generic;

using System.IO;

using System.Numerics;

using System.Security.Cryptography;

using System.Text;

using System.Linq;

namespace lab3

{

class Program

{

static int GetInt(string warning)

{

bool ka; int p;

do {

string s = Console.ReadLine();

ka = Int32.TryParse(s, out p);

if (ka == false || p < 1)

Console.WriteLine(warning);

} while (ka == false || p < 1);

return p;

}

static void Main(string[] args)

{

Console.WriteLine("Какой способ ассиметричного шифрования вы хотите использовать? " +

"Нажмите R, если вы хотите использовать самописный RSA, K, если вы хотите использовать алгоритм на основе задачи об упакове рюкзака, и N, если хотите использовать библиотечный RSA");

char ch;

do {

ch = Console.ReadKey().KeyChar;

Console.CursorLeft = 1;

if (ch == 'R')

{ Task2(); break; }

else if (ch == 'K')

{ Task1(); break; }

else if (ch == 'N')

{ Task3(); break; }

} while (true);

Console.ReadKey();

}

static void Task2()

{

// Запрашиваем пару чисел (p, q) ключ

Console.WriteLine("Введите значение простого числа p:");

int p = GetInt("Введите корректное положительное число.");

Console.WriteLine("Введите значение простого числа q:");

int q = GetInt("Введите корректное положительное число.");

// Генерируем закрытый и открытый ключи

Tuple<BigInteger, BigInteger, BigInteger> T1 = RSA.GenerateKeys(p, q);

Console.WriteLine("На основании пары чисел (p, q) были сгенерированы приватный и публичный ключи.");

Console.WriteLine("Приватный ключ (d, n):");

Console.WriteLine($"d = {T1.Item1}, n = {T1.Item2}");

Console.WriteLine("Публичный ключ (e, n):");

Console.WriteLine($"e = {T1.Item3}, n = {T1.Item2}");

// Запрашиваем путь к файлу с текстом, который необходимо зашифровать

Console.WriteLine("Укажите имя текстового файла с текстом, который необходимо зашифровать:");

string path;

string textBefore;

do {

try

{

path = Console.ReadLine();

textBefore = File.ReadAllText(path);

break;

}

catch

{

Console.WriteLine("Указанный вами файл не существует =(");

Console.WriteLine("Попробуйте снова");

}

} while (true);

// Шифруем и показываем текст с помощью публичного ключа:

List<BigInteger> cryptoText = RSA.Encrypt(T1.Item2, T1.Item3, textBefore);

Console.WriteLine("Зашифрованное сообщение:");

for (int i = 0; i < cryptoText.Count; i++)

Console.Write(cryptoText[i].ToString() + " ");

Console.WriteLine();

// Расшифровываем и показываем зашифрованный текст с помощью приватного ключа

string text = RSA.Decrypt(T1.Item1, T1.Item2, cryptoText);

Console.WriteLine("Расшифрованный текст:");

Console.WriteLine(text);

}

static void Task1()

{

// Запрашиваем приватный ключ

Console.WriteLine("Введите приватный ключ. Во-первых, укажите длину ключа:");

int size = GetInt("Введите корректное положительное число.");

List<int> privateKey = new List<int>(size);

for (int i = 0; i < size; i++)

{

Console.WriteLine("Введите " + i + "-е число приватного ключа");

privateKey.Add(GetInt("Введите корректное положительное число."));

}

// Проверяем публичный ключ на корректность

bool isOk = ByKnapsack.Check(privateKey);

if (!isOk)

{

Console.WriteLine("Введённый вами ключ не является сверхвозрастающей последовательностью =(");

Console.ReadKey();

return;

}

// Генерируем и показываем публичный ключ

Tuple<List<int>, int, int> T1 = ByKnapsack.GeneratePublicKey(privateKey);

List<int> publicKey = T1.Item1;

int m = T1.Item2, n = T1.Item3;

Console.WriteLine("На основании приватного ключа сгенерировался следующий публичный ключ:");

for (int i = 0; i < publicKey.Count; i++)

Console.Write(publicKey[i].ToString() + " ");

Console.WriteLine();

// Запрашиваем путь к файлу с текстом, который необходимо зашифровать

Console.WriteLine("Укажите имя текстового файла с текстом, который необходимо зашифровать:");

string path;

string textBefore;

do {

try

{

path = Console.ReadLine();

textBefore = File.ReadAllText(path);

break;

}

catch

{

Console.WriteLine("Указанный вами файл не существует =(");

Console.WriteLine("Попробуйте снова");

}

} while (true);

// Шифруем и показываем текст с помощью публичного ключа:

List<int> cryptoText = ByKnapsack.Encrypt(textBefore, publicKey);

Console.WriteLine("Зашифрованное сообщение:");

for (int i = 0; i < cryptoText.Count; i++)

Console.Write(cryptoText[i].ToString() + " ");

Console.WriteLine();

// Расшифровываем и показываем зашифрованный текст с помощью приватного ключа

string text = ByKnapsack.Decrypt(cryptoText, privateKey, n, m);

Console.WriteLine("Расшифрованный текст:");

Console.WriteLine(text);

}

static void Task3()

{

using (RSACryptoServiceProvider rsa\_csp = new RSACryptoServiceProvider(2048))

{

Console.WriteLine("Хотите использовать уже имеющиеся приватный и публичный ключи (Y)? " +

"Нажмите (N), если хотите сгенерировать новые. В этом случае они сохранятся в файл xmlstring.txt.");

char ch;

do {

ch = Console.ReadKey().KeyChar;

Console.CursorLeft = 1;

if (ch == 'Y')

{

// Запрашиваем путь к файлу с ключами

Console.WriteLine("Укажите имя текстового файла с ключами:");

string path;

string xmlString;

do {

try

{

path = Console.ReadLine();

xmlString = File.ReadAllText(path);

break;

}

catch

{

Console.WriteLine("Указанный вами файл не существует =(");

Console.WriteLine("Попробуйте снова");

}

} while (true);

rsa\_csp.FromXmlString(xmlString);

break;

}

else if(ch == 'N')

{

string xmlString = rsa\_csp.ToXmlString(true);

File.WriteAllText("xmlstring.txt", xmlString);

break;

}

} while (true);

// Запрашиваем путь к файлу с текстом, который необходимо зашифровать

Console.WriteLine("Укажите имя текстового файла с текстом, который необходимо зашифровать:");

string path2;

string textBefore;

do {

try

{

path2 = Console.ReadLine();

textBefore = File.ReadAllText(path2);

break;

}

catch

{

Console.WriteLine("Указанный вами файл не существует =(");

Console.WriteLine("Попробуйте снова");

}

} while (true);

// Зашифровываем текст блоками по 224 байта в соответствии с открытым ключом

List<byte> encryptedData = new List<byte>();

for (int i = 0; i < textBefore.Length; i += RSANet.blockSize / 2)

{

string curStr;

if (i + RSANet.blockSize / 2 > textBefore.Length)

curStr = textBefore.Substring(i);

else

curStr = textBefore.Substring(i, RSANet.blockSize / 2);

byte[] curByteBlock = RSANet.StringToBytes(curStr); // ByteConverter.GetBytes(curStr);

byte[] curEncryptedData = RSANet.Encrypt(curByteBlock, rsa\_csp.ExportParameters(false), false);

encryptedData.InsertRange(encryptedData.Count, curEncryptedData);

}

// Выводим зашифрованное сообщение на экран

Console.WriteLine("Зашифрованное сообщение:");

for (int i = 0; i < encryptedData.Count; i++)

Console.Write(encryptedData[i] + " ");

Console.WriteLine();

// Расшифровываем текст в соответствии с закрытым ключом

StringBuilder decryptedData = new StringBuilder();

int blockSize = 256;

for (int i = 0; i < encryptedData.Count; i += blockSize)

{

byte[] curByteBlock = encryptedData.Skip(i).Take(blockSize).ToArray();

byte[] curDecryptedData = RSANet.Decrypt(curByteBlock, rsa\_csp.ExportParameters(true), false);

decryptedData.Append(RSANet.BytesToString(curDecryptedData)); // Encoding.Un.GetString(curDecryptedData));

}

// Выводим расшифрованное сообщение на экран

Console.WriteLine("Расшифрованный текст:");

Console.WriteLine(decryptedData);

}

}

}

}

Файл RSA.cs, содержащий реализацию алгоритм RSA:

using System;

using System.Collections.Generic;

using System.Text;

using System.Numerics;

using System.Linq;

namespace lab3

{

static class RSA

{

static int encodingSize = 16;

/// <summary>

/// Расширенный Алгоритм Евклида.

/// </summary>

/// <param name="a"></param>

/// <param name="b"></param>

/// <param name="x"></param>

/// <param name="y"></param>

/// <returns></returns>

static public BigInteger GCDEx(BigInteger a, BigInteger b, out BigInteger x, out BigInteger y)

{

if (a == 0)

{

x = 0; y = 1;

return b;

}

BigInteger x1, y1;

BigInteger d = GCDEx(b % a, a, out x1, out y1);

x = y1 - (b / a) \* x1;

y = x1;

return d;

}

/// <summary>

/// Возвращает закрытый и открытый ключи (d, n, e) в зависимости от двух простых чисел p и q.

/// </summary>

/// <param name="p"></param>

/// <param name="q"></param>

/// <returns></returns>

static public Tuple<BigInteger, BigInteger, BigInteger> GenerateKeys(BigInteger p, BigInteger q)

{

BigInteger n = p \* q;

//BigInteger e = 13;

//BigInteger e = 79;

BigInteger e = 65537;

BigInteger d, sup;

BigInteger m = (p - 1) \* (q - 1);

BigInteger g = GCDEx(e, m, out d, out sup);

if (g != 1)

throw new Exception("");

d = (d % m + m) % m;

return new Tuple<BigInteger, BigInteger, BigInteger>(d, n, e);

}

/// <summary>

/// Возвращает разрядность числа в двоичной СС.

/// </summary>

/// <param name="a"></param>

/// <returns></returns>

static private int GetNumberSize(BigInteger a)

{

return (int)(BigInteger.Log10(a) \* Math.Log(10) / Math.Log(2));

}

/// <summary>

/// Переводит сообщение s в последовательность бит.

/// </summary>

/// <param name="s"></param>

/// <returns></returns>

static private List<bool> ToBinary(string s)

{

List<bool> binaryText = new List<bool>(s.Length \* encodingSize);

for (int i = 0; i < s.Length; i++)

{

int a = s[i];

for (int j = 0; j < encodingSize; j++)

{

binaryText.Add((a % 2) == 1);

a >>= 1;

}

}

return binaryText;

}

/// <summary>

/// Переводит 64-разрядное число в последовательность бит.

/// </summary>

/// <param name="a"></param>

/// <returns></returns>

static public List<bool> ToBinary(BigInteger a, int size)

{

List<bool> binaryText = new List<bool>(size);

for (int i = 0; i < size; i++)

{

binaryText.Add((a % 2) == 1);

a >>= 1;

}

return binaryText;

}

/// <summary>

/// Переводит последовательность бит в 64-разрядное целое число.

/// При этом элементы в начале последовательности считаются старшими разрядами.

/// </summary>

/// <param name="b"></param>

/// <returns></returns>

static public BigInteger BinaryToInt64(List<bool> b)

{

BigInteger result = 0;

for (int i = 0; i < b.Count; i++)

{

result <<= 1;

result += b[i] ? 1 : 0;

}

return result;

}

static public BigInteger ModularExp(BigInteger a, BigInteger e, BigInteger m)

{

BigInteger result = 1;

List<bool> bin = ToBinary(e, 64);

for (int i = bin.Count - 1; i >= 0; i--)

{

result \*= result;

result %= m;

if (bin[i])

result \*= a;

result %= m;

}

return result;

}

/// <summary>

/// Зашифровывает сообщение text в соответствии с открытым ключом (n, e).

/// </summary>

/// <param name="n"></param>

/// <param name="e"></param>

/// <param name="text"></param>

/// <returns></returns>

static public List<BigInteger> Encrypt(BigInteger n, BigInteger e, string text)

{

int blockSize = GetNumberSize(n) - 2;

List<bool> binaryText = ToBinary(text);

while (binaryText.Count % blockSize != 0)

binaryText.Add(false);

List<BigInteger> result = new List<BigInteger>(text.Length);

for (int i = 0; i < binaryText.Count; i += blockSize)

{

List<bool> curBlock = binaryText.Skip(i).Take(blockSize).ToList();

BigInteger mi = BinaryToInt64(curBlock);

BigInteger ci = ModularExp(mi, e, n);

result.Add(ci);

}

return result;

}

/// <summary>

/// Расшифровывает сообщение text в соответствии с приватным ключом d и значением n публичного ключа.

/// </summary>

/// <param name="d"></param>

/// <param name="n"></param>

/// <param name="text"></param>

/// <returns></returns>

static public string Decrypt(BigInteger d, BigInteger n, List<BigInteger> text)

{

int blockSize = GetNumberSize(n) - 2;

List<bool> binaryText = new List<bool>(text.Count \* blockSize);

for (int i = 0; i < text.Count; i++)

{

BigInteger ci = text[i];

BigInteger mi = ModularExp(ci, d, n);

List<bool> bin = ToBinary(mi, blockSize); bin.Reverse();

binaryText.InsertRange(binaryText.Count, bin);

}

StringBuilder result = new StringBuilder(binaryText.Count / encodingSize);

for (int i = 0; i < binaryText.Count; i += encodingSize)

{

List<bool> curBlock = binaryText.Skip(i).Take(encodingSize).ToList();

curBlock.Reverse();

char curCh = (char)BinaryToInt64(curBlock);

result.Append(curCh);

}

return result.ToString();

}

}

}

Файл RSANet.cs, содержащий функции, использующие готовую реализацию алгоритма RSA из пространства имён System.Security.Cryptography:

using System;

using System.Collections.Generic;

using System.Security.Cryptography;

using System.Text;

namespace lab3

{

static class RSANet

{

static int encodingSize = 2;

static public int blockSize = 224;

public static string BytesToString(byte[] data)

{

if (data.Length % encodingSize != 0)

throw new Exception("Длина массива data не кратна размеру одного символа");

StringBuilder result = new StringBuilder(data.Length / encodingSize);

for (int i = 0; i < data.Length; i += encodingSize)

{

char ch = (char)((data[i] << 8) + data[i + 1]);

result.Append(ch);

}

return result.ToString();

}

public static byte[] StringToBytes(string data)

{

byte[] result = new byte[data.Length \* encodingSize];

for (int i = 0; i < data.Length; i++)

{

result[i \* encodingSize] = (byte)(data[i] >> 8);

result[i \* encodingSize + 1] = (byte)(data[i] ^ (data[i] >> 8 << 8));

}

return result;

}

public static byte[] Encrypt(byte[] DataToEncrypt, RSAParameters RSAKeyInfo, bool DoOAEPPadding)

{

try

{

byte[] encryptedData;

using (RSACryptoServiceProvider RSA = new RSACryptoServiceProvider())

{

RSA.ImportParameters(RSAKeyInfo);

encryptedData = RSA.Encrypt(DataToEncrypt, DoOAEPPadding);

}

return encryptedData;

}

catch (CryptographicException e)

{

Console.WriteLine(e.Message);

return null;

}

}

public static byte[] Decrypt(byte[] DataToDecrypt, RSAParameters RSAKeyInfo, bool DoOAEPPadding)

{

try

{

byte[] decryptedData;

using (RSACryptoServiceProvider RSA = new RSACryptoServiceProvider())

{

RSA.ImportParameters(RSAKeyInfo);

decryptedData = RSA.Decrypt(DataToDecrypt, DoOAEPPadding);

}

return decryptedData;

}

catch (CryptographicException e)

{

Console.WriteLine(e.ToString());

return null;

}

}

}

}