Федеральное государственное автономное

образовательное учреждение

высшего профессионального образования

«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

институт

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

кафедра

**ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

тема

\_\_\_\_\_\_\_\_\_\_\_\_Алгоритм шифрования RSA\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Преподаватель \_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_

подпись, дата инициалы, фамилия

Студент \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_

номер группы, зачетной книжки подпись, дата инициалы, фамилия

Оглавление

[Задание 3](#_Toc417025792)

[Описание алгоритма 4](#_Toc417025793)

[**1.** **Генерация открытого и закрытого ключей** 4](#_Toc417025794)

[**2.** **Шифрование** 4](#_Toc417025795)

[**3.** **Расшифровка** 4](#_Toc417025796)

[Блок-схема алгоритма 5](#_Toc417025797)

[Листинг программы, реализующей алгоритмы 6](#_Toc417025798)

[Класс RSACryptography 6](#_Toc417025799)

[Класс MessageManager 8](#_Toc417025800)

[Примеры работы программы 10](#_Toc417025801)

[Тестирование производительности 15](#_Toc417025802)

[Заключение 20](#_Toc417025803)

# Задание

1. Составить в виде блок-схемы алгоритм шифрования/дешифрования RSA, со следующими особенностями:

* объём исходного текста – любой (в разумных пределах);
* исходный текст может состоять из русских и английских букв, цифр, а также знаков препинания;
* исходный текст находится в кодировке ASCII;
* N состоит из 31 десятичного знака.
* Числа P и Q выбираются случайным образом, так, что , где P и Q – простые числа.
* исходный текст разбивается на K блоков, где K выбирается исходя из значения модуля N

1. Убедиться в правильности составления алгоритмов, а затем на языке C# составить программу, которая реализует данный алгоритм.
2. На ряде контрольных примеров (не менее 10) открытого текста проверить правильность работы алгоритмов шифрования и дешифрования (в качестве контрольного примера понимается текстовый файл в кодировке ASCII).
3. Оценить криптостойкость моего варианта алгоритма RSA, а также сделать оценку производительности, разработанной программы.

# Описание алгоритма

1. **Генерация открытого и закрытого ключей**
   1. Выбираются два различных случайных простых числа заданного размера.
   2. Вычисляется их произведение
   3. Вычисляется значение *функции Эйлера*
   4. Случайным образом выбирается число и взаимно простое с
   5. Вычисляется ***e***, такое что - обратный элемент в кольце целых чисел
2. **Шифрование**
   1. Сообщение ***M*** разбивается на на ***K***, блоков : так что размер каждого блока **.**
   2. Каждый из блоков шифруется по формуле
   3. Зашифрованное сообщениепередается.
3. **Расшифровка**
   1. Для каждого блока из полученного  вычисляется

* 1. Блоки объединяются в сообщение.
  2. - исходное сообщение

# Блок-схема алгоритма



# Листинг программы, реализующей алгоритмы

Программа разработана с использованием технологий ООП и инкапсуляцией алгоритмов шифратора в объекте класса RSACryptography. А класс MessageManager занимается конвертацией строк в массив байт и обратно, а такаже разбиением сообщений на блоки.

## Класс RSACryptography

using System;

using System.Collections.Generic;

using System.Numerics;

using System.Security.Cryptography;

namespace RSA\_Project

{

public class RsaCryptography

{

private static readonly Random Rnd = new Random((int) DateTime.Now.Ticks);

private static readonly RandomNumberGenerator Rng = RandomNumberGenerator.Create();

private readonly int[] \_fermatNumbers = {17, 257, 65537}; //Числа ферма

private BigInteger \_n; //P,Q -простые числа. N - их произведение

private BigInteger \_p; //P,Q -простые числа. N - их произведение

private BigInteger \_privateKey; //открытый и закрытый ключи

private BigInteger \_publicKey; //открытый и закрытый ключи

private BigInteger \_q; //P,Q -простые числа. N - их произведение

public RsaCryptography(BigInteger p, BigInteger q)

{

//Инициализация класса

\_p = p;

\_q = q;

\_n = \_p\*\_q;

BigInteger eulierFunction = (\_p - 1)\*(\_q - 1);

\_publicKey = \_fermatNumbers[Rnd.Next(0, \_fermatNumbers.Length)];

\_privateKey = BuildPrivateKey(eulierFunction);

}

public RsaCryptography(int log10N)

{

GenerateKeys(log10N);

}

public long ModuleLength

{

//Длина модуля в байтах

get

{

long l = \_n.ToByteArray().Length;

return l;

}

}

public List<byte[]> EncryptMessage(List<byte[]> p)

{

//Зашифровать строку

var result = new List<byte[]>();

foreach (var block in p)

{

result.Add(Encrypt(block));

}

return result;

}

public List<byte[]> DecryptMessage(List<byte[]> p)

{

//Разшифровать строку

var result = new List<byte[]>();

foreach (var block in p)

{

result.Add(Decrypt(block));

}

return result;

}

public byte[] Encrypt(byte[] message)

{

//Шифруем сообщение открытым ключом

var val = new BigInteger(message);

val = BigInteger.ModPow(val, \_publicKey, \_n);

return val.ToByteArray();

}

public byte[] Decrypt(byte[] message)

{

//Расшифруем cообщение открытм ключом

var val = new BigInteger(message);

val = BigInteger.ModPow(val, \_privateKey, \_n);

return val.ToByteArray();

}

private BigInteger BuildPrivateKey(BigInteger E)

{

//Вычисляем закрытый ключ, находя обратный по модулю элемент кольца

BigInteger x, y;

BigInteger g = GCD(\_publicKey, E, out x, out y);

return (x%E + E)%E;

}

private static BigInteger GCD(BigInteger a, BigInteger b, out BigInteger x, out BigInteger y)

{

//Расширеный Алгоритм Евклида

if (a.IsZero) //Находит НОД чисел A и B, и коэфициенты x,y уравнения Ax +By = НОД(A,B)

{

x = 0;

y = 1;

return b;

}

BigInteger x1, y1;

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

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

y = x1;

return d;

}

#region Свойства

public BigInteger Module

{

//Возвращает значения модуля(N)

get { return \_n; }

}

public BigInteger PrivateKey

{

//значение закрытого ключа

get { return \_privateKey; }

}

public BigInteger PublicKey

{

//значение откртыго ключа

get { return \_publicKey; }

}

#endregion

#region Методы для генерации случайного ключа

private static BigInteger Random(int bytes)

{

if (bytes == 0) return new BigInteger(0);

var buffer = new byte[bytes];

Rng.GetBytes(buffer);

buffer[bytes - 1] = (byte) ((buffer[bytes - 1] & 127) | 64); // старший байт

return new BigInteger(buffer);

}

private static bool IsPrimary(BigInteger x)

{

if (x < 2) return false; // отбрасываем отрицательные и единицу

int len = x.ToByteArray().Length;

int tests = 2\*len + 100; // Чем больше тестов тем меньше вероятность ошибиться

BigInteger y = x - 1;

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

{

BigInteger a = (Random((int) (len\*Rnd.NextDouble()))%y) + 1; // берём ненулевое

// проверяем выполнение малой теоремы Ферма

// если простое то

if (!(BigInteger.ModPow(a, y, x) - 1).IsZero) return false;

}

// признаём число простым

// хотя можем продолжать ошибаться

return true;

}

private BigInteger GeneratePrimary(int bytes)

{

BigInteger x = Random(bytes) | 1; // Простые являются нечётными

while (!IsPrimary(x)) x += 2; // Движемся вперёд пока не встретим простое

return x;

}

public void GenerateKeys(int log10N)

{

// 10^x==2^y

// x=y\*log10(2)

var bits = (int) Math.Ceiling(log10N/Math.Log10(2));

int bits1 = bits/2;

int bits2 = bits - bits1;

int bytes1 = (bits1 + 7)/8;

int bytes2 = (bits2 + 7)/8;

\_p = GeneratePrimary(bytes1);

\_q = GeneratePrimary(bytes2);

\_n = \_p\*\_q;

BigInteger eulierFunction = (\_p - 1)\*(\_q - 1);

\_publicKey = \_fermatNumbers[Rnd.Next(0, \_fermatNumbers.Length)];

\_privateKey = BuildPrivateKey(eulierFunction);

}

#endregion

}

}

## Класс MessageManager

using System;

using System.Collections.Generic;

using System.Text;

namespace RSA\_Project

{

public static class MessageManager

{

//Класс работающий с кодировками и разбиением текста

public static String PartitionToString(List<byte[]> p)

{

//Переобразует разбиение в строку

String result = String.Empty;

foreach (var block in p)

{

result += GetString(block);

}

return result;

}

public static List<byte[]> PartitionOfString(String s, long length)

{

//Разбиваем строку на блоки(байт-массивы)

var result = new List<byte[]>();

byte[] bytes = GetBytes(s);

var block = new List<byte>();

int count = 0;

foreach (byte x in bytes)

{

count++;

if (count < length)

{

block.Add(x);

}

else

{

result.Add(block.ToArray());

block.Clear();

block.Add(x);

count = 1;

}

}

if (block.Count > 0) result.Add(block.ToArray());

return result;

}

public static byte[] GetBytes(string str)

{

byte[] utf8bytes = Encoding.Unicode.GetBytes(str);

byte[] win1251Bytes = Encoding.Convert(

Encoding.Unicode, Encoding.GetEncoding("windows-1251"), utf8bytes);

//byte[] bytes = Encoding.ASCII.GetBytes(str);

return win1251Bytes;

}

public static string GetString(byte[] bytes)

{

byte[] unicodeBytes = Encoding.Convert(

Encoding.GetEncoding("windows-1251"), Encoding.Unicode, bytes);

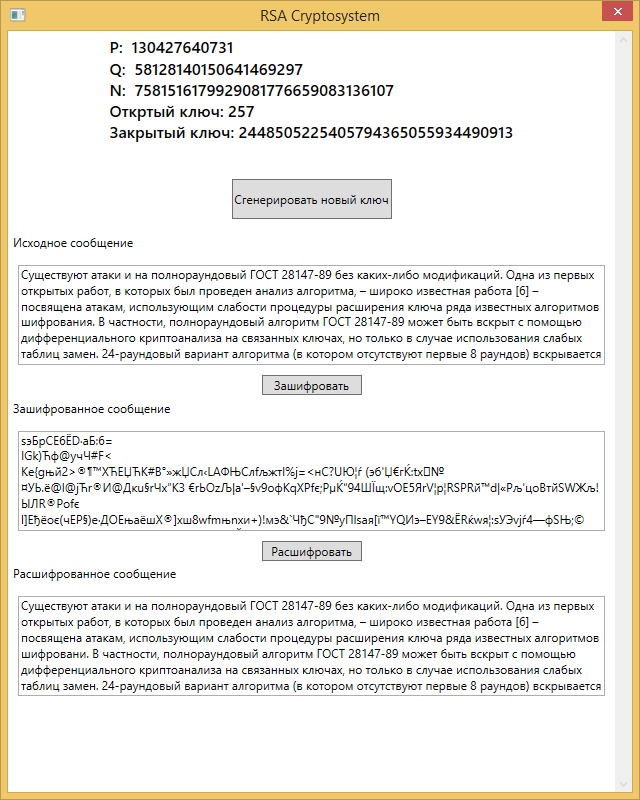
return Encoding.Unicode.GetString(unicodeBytes);

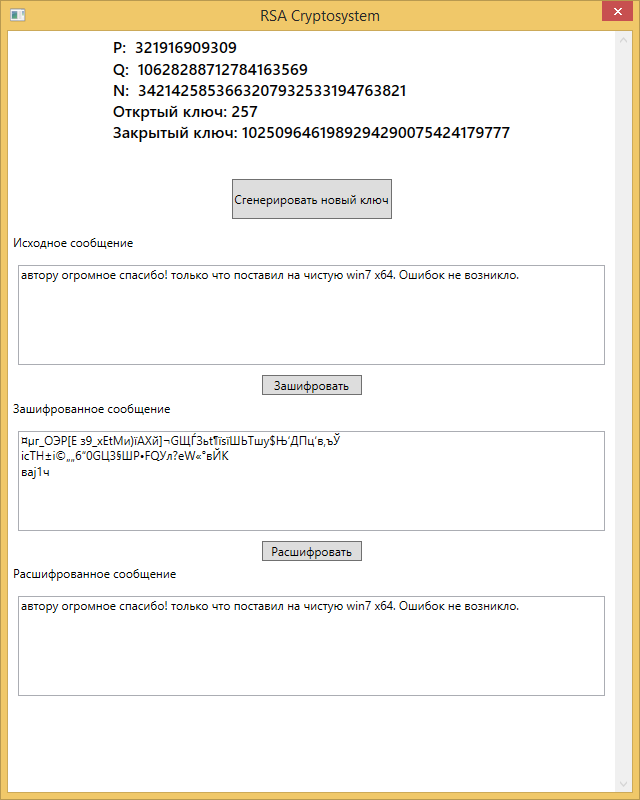
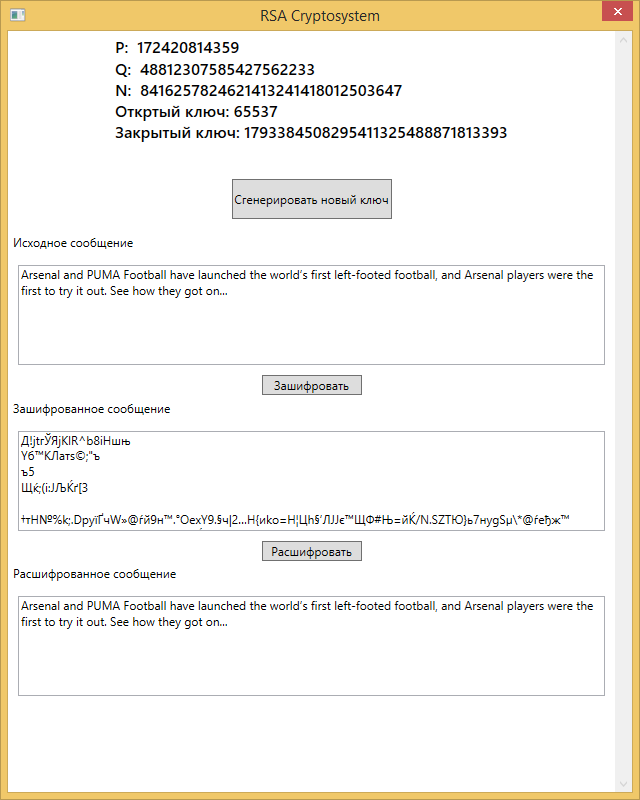
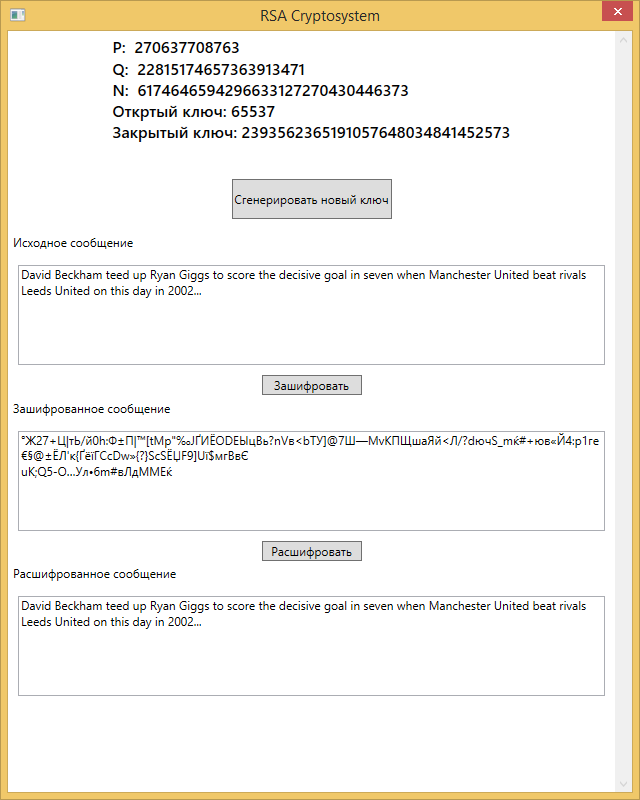
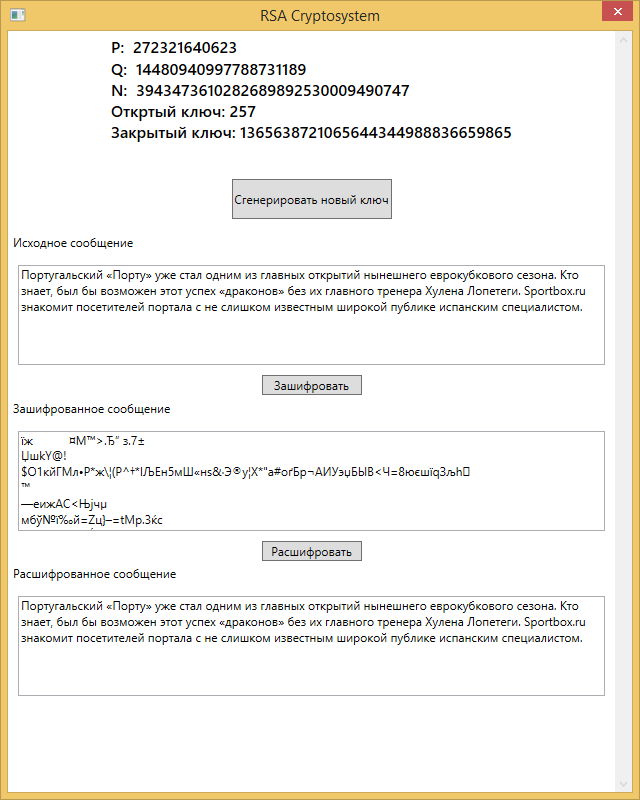
}

}

}

# Примеры работы программы





# Тестирование производительности

Windows 8.1 Профессиональная

Процессор: Intel Core Duo E8600 @ 3.33GHz 2.67GHz

Установленная память: 8,00 ГБ

Тип системы: 64-разрядная операционная система, процессор x64

Время измеряется в миллисекундах (1s == 1000ms)

N – количество десятичных разрядов

TestMethod1 – измерение среднего времени генерации ключа

TestMethod2 – измерение среднего времени генерации одного простого числа

TestMethod3 – измерение среднего времени одной итерации проверки числа на простоту

using System;

using System.Numerics;

using Microsoft.VisualStudio.TestTools.UnitTesting;

using RSA\_Project;

namespace ZI\_Lab\_Lab\_3\_1.UnitTest

{

[TestClass]

public class UnitTest2

{

[TestMethod]

public void TestMethod1()

{

const int count = 20;

Console.WriteLine(@"# N GenerateKeys");

for (int log10N = 20; log10N < 50; log10N++)

{

var rsa = new RsaCryptography(log10N);

DateTime t = DateTime.Now;

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

rsa.GenerateKeys(log10N);

var ts = new TimeSpan(DateTime.Now.Ticks - t.Ticks);

Console.WriteLine(@"{0} {1}", log10N, ts.TotalMilliseconds/count);

}

}

[TestMethod]

public void TestMethod2()

{

const int count = 20;

Console.WriteLine(@"# N GeneratePrimary");

for (int log10N = 20; log10N < 50; log10N++)

{

var bits = (int) Math.Ceiling(log10N/Math.Log10(2));

int bytes = (bits + 7)/8;

DateTime t = DateTime.Now;

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

RsaCryptography.GeneratePrimary(bytes);

var ts = new TimeSpan(DateTime.Now.Ticks - t.Ticks);

Console.WriteLine(@"{0} {1}", log10N, ts.TotalMilliseconds/count);

}

}

[TestMethod]

public void TestMethod3()

{

const int count = 20;

Console.WriteLine(@"# N IsPrimary");

for (int log10N = 20; log10N < 50; log10N++)

{

var bits = (int) Math.Ceiling(log10N/Math.Log10(2));

int bytes = (bits + 7)/8;

DateTime t = DateTime.Now;

int total = 0;

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

{

BigInteger x = RsaCryptography.Random(bytes) | 1; // Простые являются нечётными

total += RsaCryptography.NumberOfTests(x);

RsaCryptography.IsPrimary(x);

}

var ts = new TimeSpan(DateTime.Now.Ticks - t.Ticks);

Console.WriteLine(@"{0} {1}", log10N, ts.TotalMilliseconds/ count);

}

}

}

}

**# N GenerateKeys**

**20 5.150305**

**21 4.30026**

**22 4.80027**

**23 4.250275**

**24 4.600245**

**25 5.850355**

**26 5.80035**

**27 6.00035**

**28 6.400365**

**29 7.55044**

**30 8.500505**

**31 8.100455**

**32 8.400485**

**33 7.95048**

**34 8.20046**

**35 9.35053**

**36 15.45089**

**37 8.850495**

**38 9.05053**

**39 15.00085**

**40 16.10093**

**41 14.65083**

**42 15.50089**

**43 17.801035**

**44 22.301275**

**45 29.251675**

**46 35.702015**

**47 24.85142**

**48 35.202035**

**49 23.35135**

**# N GeneratePrimary**

**20 9.90058**

**21 9.500555**

**22 9.50054**

**23 9.600575**

**24 21.901255**

**25 12.450735**

**26 19.701115**

**27 15.850925**

**28 18.401065**

**29 26.55155**

**30 38.60221**

**31 37.402175**

**32 31.45183**

**33 33.70193**

**34 37.80221**

**35 22.55129**

**36 26.80155**

**37 23.55137**

**38 25.001445**

**39 46.00267**

**40 44.352575**

**41 45.25263**

**42 44.05253**

**43 41.452395**

**44 38.80226**

**45 45.302605**

**46 58.25338**

**47 68.35397**

**48 62.553595**

**49 67.90393**

**# N IsPrimary**

**20 1.450075**

**21 0.400005**

**22 0.850055**

**23 0.15004**

**24 0.05**

**25 0.09999**

**26 0.449995**

**27 0.60006**

**28 0.049985**

**29 0.15001**

**30 0.10005**

**31 0.199995**

**32 0.09999**

**33 0.15001**

**34 0.950045**

**35 0.15004**

**36 0.150005**

**37 0.199995**

**38 1.15009**

**39 0.199995**

**40 3.100185**

**41 0.25**

**42 1.750135**

**43 0.3**

**44 0.19999**

**45 0.25003**

**46 0.35002**

**47 0.249995**

**48 0.25003**

**49 4.95028**

# Заключение

1. В ходе работы были изучены следующие дополнительные навыки:

* Обобщеный алгоритм Евклида
* Вероятностный тест на простоту Миллера-Рабина
* Работ с большими числами в .NET
* Реализация оконного интерфейса с помощью технологии WPF
* Базовые знания теории чисел такие как:
  + Кольцо целых чисел
  + Функция Эйлера
  + Теорема Ферма
  + НОД
  + Сравнение по модулю

1. Алгоритм RSA является криптостойким и это свойство зависит напрямую от длины модуля N. Чем длиннее число, тем дольше происходит процесс его факторизации.