МИНОБРНАУКИ РОССИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«ВОРОНЕЖСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

(ФГБОУ ВО «ВГУ»)

Факультет прикладной математики, информатики и механики

Кафедра ERP-систем и бизнес-процессов

**О Т Ч Е Т**

о выполнении преддипломной практики

На тему «Алгоритм Эль-Гамаля. Лабораторная работа»

Зав. кафедрой Йорг Беккер

Обучающийся В. А. Ковун

Руководитель к.т.н., доц. Б. Н. Воронков

Воронеж **–** 2018

*Общие сведения*

В обучении будущих специалистов в любой области немаловажную роль играет использование наглядных средств обучения, эксперимент (в том числе вычислительный) и проверка полученных знаний на практике. В настоящее время в образовании успешно используются обучающие программы, лучшие из которых, как правило, сочетают в себе все эти три фактора. В рамках ВКР магистра была разработана обучающая программа, проектировавшаяся в первую очередь исходя из всех вышеуказанных принципов. Программа написана на языке C#, работает на операционных системах семейства Windows с установленным фреймворком Microsoft .NET Framework 4.5 или новее. Работоспособность проверена на Windows 7 x32, Windows 7 x64, Windows 8 x64 и Windows 10 x64. Объём программы – 245 КБ.

*Функциональное назначение*

El-Gamal\_Tutor – обучающая программа, созданная для использования в лабораторных работах по дисциплинам, связанным с изучением методов и средств защиты информации.

*Логическая структура*

Программа имеет две части: обучающую и дополнительную. В обучающей части пользователь знакомится с криптосистемой Эль-Гамаля, математическими методами, на которых она базируется, изучает основы криптоанализа предлагаемой системы, а так же выполняет задания и проходит тесты с получением оценки по итогам их выполнения. Дополнительная часть предоставляет образец функционирующей асимметричной криптосистемы, включающий в себя генерацию ключей, зашифрование и расшифрование. Также программа предоставляет пользователю набор вычислительного функционала в области теории чисел, относящегося к криптосистеме Эль-Гамаля. Обеспечивается хорошее быстродействие длинной арифметики для чисел до ста десятичных разрядов длиной.

Ниже приведено описание основных классов El-Gamal\_Tutor.

Класс CryptoMath содержит математические методы для осуществления криптографических преобразований, таких, как нахождение функции Эйлера, генерация случайных простых чисел заданной длины, нахождение обратного числа по модулю и т.д.

Класс PollardsRho содержит функции вычисления дискретного логарифма ро-методом Полларда.

Класс BabyStepGiantStep содержит функции вычисления дискретного логарифма с помощью алгоритма Гельфонда-Шенкса, также известного как алгоритм больших и малых шагов.

Класс SilverPohligHellman содержит функции вычисления дискретного логарифма с помощью алгоритма Сильвера-Полига-Хеллмана.

Класс BigIntegerSqrt содержит метод расширения класса BigInteger для извлечения квадратного корня из больших чисел методом Ньютона.

Класс MainMenuForm содержит форму главного окна приложения, а также точку входа программы.

Графический интерфейс пользователя реализован посредством нескольких форм, часть из которых относится к обучающей части приложения, а другая часть – к дополнительной.

Класс CryptoMath:

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

using System.Numerics;

namespace ElgamalTutor

{

class CryptoMath

{

public static Random rand = new Random();

public static int MILLER\_RABIN\_ROUNDS = 4000;

public const long DIGITS = 20;

public static BigInteger phi(BigInteger n)

{

BigInteger result = n;

BigInteger i = new BigInteger(2);

for (i = 2; i \* i <= n; ++i)

if (n % i == 0)

{

while (n % i == 0)

n /= i;

result -= result / i;

}

if (n > 1)

result -= result / n;

return result;

}

public static bool isPrime(BigInteger n)

{

if (n == 2 || n == 3 || n == 5 || n == 7)

return true;

if ((n % 60) % 5 == 0)

return false;

if ((n % 60) % 3 == 0)

return false;

if ((n % 60) % 2 == 0)

return false;

//тест Миллера-Рабина

//сначала представим m-1 в виде m-1 = 2^s \* t

BigInteger m = n-1;

BigInteger s = 0;

BigInteger t = 1;

while(m/2 > 0 && m%2 == 0)

{

m /= 2;

s++;

}

t = m;

m = n;

for (BigInteger i = 0; i < MILLER\_RABIN\_ROUNDS; i++)

{

BigInteger a = rand.Next()%(m-3) + 3;

BigInteger x = BigInteger.ModPow(a,t,m);

if (x == 1 || x == m-1) continue;

for (BigInteger j = 0; j < s-1; j++)

{

x = BigInteger.ModPow(x, 2, m);

if (x == m-1) break;

if (x == 1) return false;

}

if (x==m-1) continue;

return false;

}

return true;

}

public static BigInteger calculatePrimitiveRoot(BigInteger p)

{

List<BigInteger> fact = new List<BigInteger>();

BigInteger phi = p-1, n = phi;

for (BigInteger i=2; i\*i<=n; ++i)

if (n % i == 0) {

fact.Add(i);

while (n % i == 0)

n /= i;

}

if (n > 1)

fact.Add(n);

for (BigInteger res=2; res<=p; ++res)

{

bool ok = true;

for (int i=0; i<fact.Count && ok; ++i)

ok &= BigInteger.ModPow(res, phi / fact[i], p) != 1;

if (ok) return res;

}

return -1;

}

public static BigInteger calculateFakePrimitiveRoot(BigInteger p)

{

List<BigInteger> fact = new List<BigInteger>();

BigInteger phi = p - 1, n = phi;

for (BigInteger i = 2; i \* i \* i \* i <= n; ++i)

if (n % i == 0)

{

fact.Add(i);

while (n % i == 0)

n /= i;

}

if (n > 1)

fact.Add(n);

for (BigInteger res = 2; res <= p; ++res)

{

bool ok = true;

for (int i = 0; i < fact.Count && ok; ++i)

ok &= BigInteger.ModPow(res, phi / fact[i], p) != 1;

if (ok) return res;

}

return -1;

}

public static BigInteger genSimpleRand(long decDigits/\*=DIGITS\*/)

{

BigInteger a = new BigInteger(0);

BigInteger pow = 1;

while (!isPrime(a))

{

a = 0;

pow = 1;

for (long i = 0; i < decDigits; i++)

{

BigInteger rnd = (rand.Next(8)+1) \* pow;

a += rnd;

pow \*= 10;

}

}

return a;

}

public static BigInteger genRand(long decDigits/\* = DIGITS\*/)

{

BigInteger a = new BigInteger(0);

BigInteger pow = 1;

a = 0;

pow = 1;

for (long i = 0; i < decDigits; i++)

{

BigInteger rnd = rand.Next(10) \* pow;

a += rnd;

pow \*= 10;

}

return a;

}

}

}