МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ государственное БЮДЖЕТНОЕ

образовательное учреждение

высшего образования

«НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

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

Кафедра защиты информации

**

**ОТЧЁТ**

**по лабораторной работе № 1**

**« Введение в ООП »**

**по дисциплине: «***Программирование***»**

Выполнил:Проверил:

Студент гр. «АБ-121», «АВТФ» *доцент кафедры ЗИ*

*Новиков Втюрин Александр Романович Архипова А. Б.*

«21» декабря 2022г«\_\_\_» \_\_\_\_\_\_ 2022 г.

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

(подпись) (подпись)

Новосибирск 2022

**Цели и задачи работы**: изучение основных принципов объектно-ориентированного программирования.

**Задание к работе**: Самостоятельно решить задачи в соответствии с индивидуальным вариантом на двух языках программирования высокого уровня (С++ и другой по выбору).

Реализовать криптографические протоколы Шамира, Диффи-Хеллмана, Эль-Гамаля, RSA с использованием классов для хранения данных.

Теория:

**Трёхэтапный протокол Шамира:**

Пусть абонент *А* должен передать абоненту *В* секретное сообщение *m*. *А* выбирает случайное большое простое число *p* и открыто передает его абоненту *В*. Затем абонент *А* выбирает два числа *cA* и *dA* такие, что

*cAdA* **mod** (*p*−1) = 1.

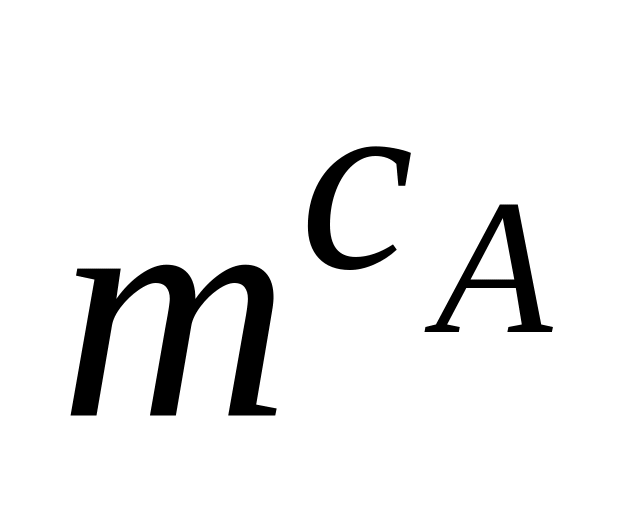
Эти числа *А* держит в секрете. Абонент *В* тоже выбирает два секретных числа *cB* и *dB* такие, что

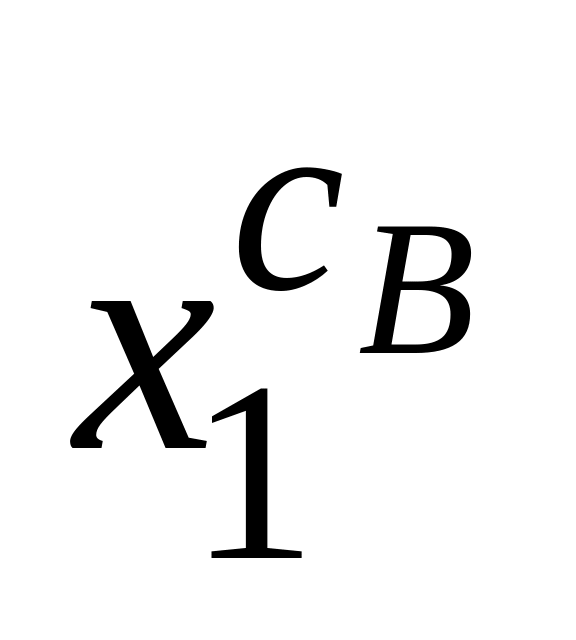
*cBdB* **mod** (*p*−1) = 1.

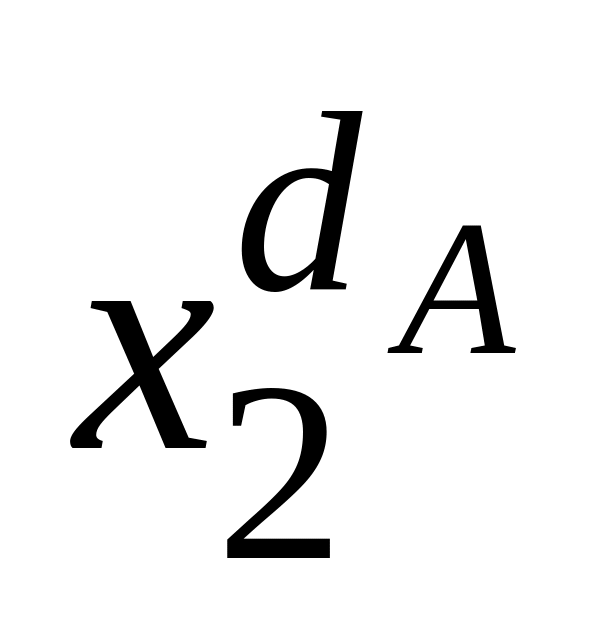
После этого *А* передает свое сообщение *m*, используя трехступенчатый протокол (см. ниже). При этом, если *m*<*p* (*m* рассматривается как число), то сообщение *m* передается целиком, иначе оно представляется в виде *m* = *m*1 *m*2 ... *mt* и каждая часть *mi* < *p* передается отдельно. Для кодирования каждой части *mi* лучше выбирать случайно новые пары (*cAdA*) и (*cBdB*); в противном случае надежность системы понижается.

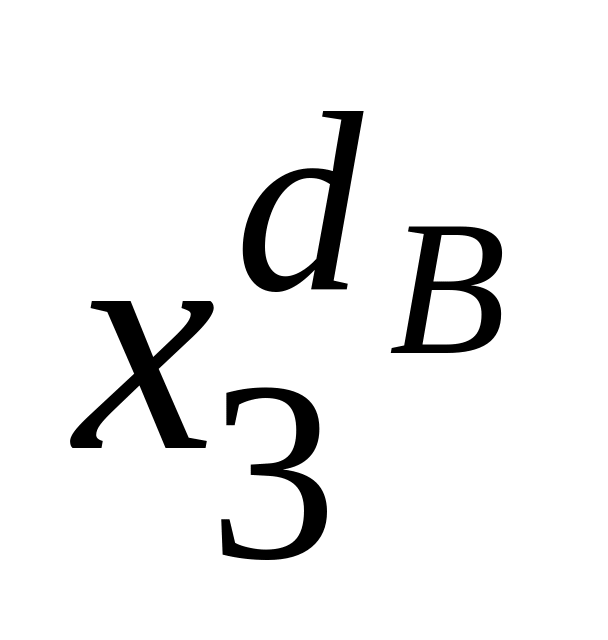
В настоящее время шифр Шамира используется, главным образом, для передачи чисел, например, секретных ключей, значения которых меньше *p*.

***Протокол шифра Шамира*** имеет вид:

**Шаг 1**. *А* вычисляет число *x*1 =  **mod** *p* и передает его абоненту *В*.

**Шаг 2**. *B* вычисляет число *x*2 =  **mod** *p* и передает его абоненту *А*.

**Шаг 3**. *А* вычисляет число *x*3 =  **mod** *p* и передает его абоненту *В*.

**Шаг 4**. *В* вычисляет число *x*4 =  **mod** *p*.

**Криптопротокол Диффи-Хеллмана:**

Пусть p — большое простое число, g — примитивный элемент группы Zp, y=gx mod p, причём p, y и g известны заранее. Функцию y=gx mod p считаем однонаправленной, то есть вычисление функции при известном значении аргумента является лёгкой задачей, а её обращение (нахождение аргумента) при известном значении функции — трудной. (Обратную функцию x = logg y mod p называют функцией дискретного логарифма. В настоящий момент не существует быстрых способов вычисления такой функции для больших простых p.)  
  
Протокол обмена состоит из следующих действий.

1. Алиса выбирает случайное 2 <= a <= p-1

Alice {A = ga mod p} Bob

1. Боб выбирает случайное

Боб вычисляет сессионный ключ

Bob {B = gb mod p} Alice

1. Алиса вычисляет K = Ba mod p

Таким способом создан общий секретный сессионный ключ . За счёт случайного выбора значений  и  в новом сеансе будет получен новый сессионный ключ.

**Криптопротокол Эль Гамаля:**

Для того чтобы генерировать пару ключей (открытый ключ – секретный ключ), сначала выбирают некоторое большое простое число Р и большое целое число G, причем G < P. Числа Р и G могут быть распространены среди группы пользователей.

Затем выбирают случайное целое число X, причем X<P. Число X является секретным ключом и должно храниться в секрете.

Далее вычисляют Y = GX mod P. Число Y является открытым ключом.

Для того чтобы зашифровать сообщение M, выбирают случайное целое число K, 1< K< P –1, такое, что числа К и (Р –1) являются взаимно простыми.

Затем вычисляют числа

a = GK mod P,

b = YK M mod P.

Пара чисел (a,b) является шифртекстом. Заметим, что длина шифртекста вдвое больше длины исходного открытого текста М.

Для того чтобы расшифровать шифртекст (a,b), вычисляют

M = mod P. (\*)

Поскольку

aX≡ GKXmod P,

≡YK≡ GKX≡ M (mod P),

то соотношение (\*) справедливо

**Криптопротокол RSA:**

Первым этапом любого асимметричного алгоритма является создание пары ключей – открытого и закрытого и распространение открытого ключа "по всему миру".

* 1. **Создание ключей**

Для алгоритма RSA этап создания ключей состоит из следующих операций:

1. Выбираются два очень больших простых числа  and .
2. Вычисляется их произведение , которое называется модулем.
3. Вычисляется значение функции Эйлера от числа :



1. Выбирается произвольное число  ( ), взаимно простое со значением функции .

Число  называется открытой экспонентой

1. С помощью алгоритма Евклида вычисляется число , которое удовлетворяет условию 
2. Пара  публикуется в качестве открытого ключа RSA.
3. Пара  играет роль закрытого ключа RSA и держится в секрете.
   1. **Шифрование и расшифрование**

Предположим, отправитель хочет послать получателю сообщение .

Сообщениями являются целые числа в интервале от 0 до , т.е . . На рисунке 1 представлена схема алгоритма RSA.

**Алгоритм Отправителя:**

1. Взять открытый ключ  получателя
2. Взять открытый текст 
3. Зашифровать сообщение с использованием открытого ключа получателя:



**Алгоритм Получателя:**

1. Принять зашифрованное сообщение 
2. Взять свой закрытый ключ 
3. Применить закрытый ключ для расшифрования сообщения:



С++

**Main.cpp:**

#include "Class\_for\_crypto\_algorithm.h"

using namespace std;

int ost(int a, int x, int p) {

int res = 1;

for (int i = 0; i < x; i++) {

res \*= a;

res %= p;

}

return res;

}

int main()

{

SourseText soursetext;

string num;

string stext;

cout << "Enter the source text: " << endl;

//cin.ignore();

getline(cin, stext);

system("cls");

cout << "Text:\n" << stext << endl;

soursetext.SetSourseElement(stext);

cout << "Transform text: " << endl;

soursetext.GetSourseText();

cout << endl;

system("pause");

while (true)

{

system("cls");

cout << "1.RSA \n" << "2.Diffie-Hellman \n" << "3.ElGamal \n" << "4.Shamir \n" << "5.Exit" << endl;

cout << "Choose an encryption algorithm:" << endl;

cin >> num;

if (num == "1")

{

EncryptRSA(soursetext.GetTextForEncrypt());

}

else if (num == "2")

{

EncryptDH(soursetext.GetTextForEncrypt());

}

else if (num == "3")

{

EncryptElGam(soursetext.GetTextForEncrypt());

}

else if (num == "4")

{

EncryptSH(soursetext.GetTextForEncrypt());

}

else if (num == "5")

{

system("cls");

return 0;

}

else {

system("cls");

cout << "Invalid number\n" << "Try again" << endl;

system("pause");

}

}

}

**Class\_for\_crypto\_algorithm.h:**

#pragma once

#include <ctime>

#include <iostream>

#include <vector>

#include <windows.h>

#include <string>

using namespace std;

class RSA

{

public:

int o\_key; //Открытый ключ

int Na;//p\*q

int Getq()

{

return q;

}

int Getp()

{

return p;

}

void Setq(const int& iq)

{

q = iq;

}

void Setp(const int& ip)

{

p = ip;

}

void GenOpenKey(int& n)

{

while (1)

{

o\_key = n;

if (o\_key % ((p - 1) \* (q - 1)) == 1)

{

o\_key = n;

break;

}

else {

n--;

}

}

}

void GenPrivateKey()// генерация закрытого ключа

{

int k = 1;

while (1)

{

k = k + ((p - 1) \* (q - 1));

if (k % o\_key == 0 && k / o\_key != 1)

{

p\_key = (k / o\_key);

break;

}

}

cout << "Private key: " << p\_key << endl;

}

void EncryptText(const vector <int>& stext)// шифровка текста

{

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

{

long int current, result;

current = stext[i] - 97;

result = 1;

for (long int j = 0; j < o\_key; j++)

{

result = result \* current;

result = result % Na;

}

encrypttext.push\_back(result);

}

}

void DecryptText()// дешифровка текста

{

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

{

long int current, result;

current = encrypttext[i];

result = 1;

for (long int j = 0; j < p\_key; j++)

{

result = result \* current;

result = result % Na;

}

decrypttext.push\_back(result + 97);

}

}

void GetEncryptText()

{

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

{

cout << encrypttext[i] << " ";

}

}

void GetDecryptText()

{

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

{

cout << decrypttext[i] << " ";

}

}

private:

int p\_key;//Закрытый ключ

int p, q;// Простые числа по выбору пользователя

vector <int> encrypttext;

vector <int> decrypttext;

};

class DH

{

public:

int p,g;

long int ya;

string Name;

int GetXkey()

{

return xa;

}

void SetYkey(const int t)

{

ya = 1;

for (int i = 0; i < (t - 1); i++)

{

ya = ya \* g % p;

}

cout << Name << "'s Public Key: " << ya << endl;

}

void SetXkey(const int t, const long int p\_key)

{

xa = 1;

for (int i = 0; i < (t - 1); i++)

{

xa = xa \* p\_key % p;

}

cout << Name << "'s Private Key: " << xa << endl;

}

private:

long int xa;

};

class SH

{

public:

int p;

vector <int> x1;

vector <int> x2;

SH(int a) {

p = a; Ca = 0; da = 0;

}

int GetCa() {

return Ca;

}

void SetCa(const int& a) {

Ca = a;

}

int Getda() {

return da;

}

void Setda(const int& a) {

da = a;

}

private:

int Ca, da;

};

class ElGam

{

public:

int p, g, y, k, decel;

pair <int, int> encrel;

vector <pair<int, int>> encrypttext;

vector <int> decrypttext;

ElGam(int a, int b, int d) {

p = a; g = b; y = 1; p\_key = d; decel = 1; k = 10;

}

int GetPKey()

{

return p\_key;

}

void GenKey(int a, int x, int m)

{

y = 1;

for (int i = 0; i < x; i++) {

y \*= a;

y %= m;

}

cout << "y = " << y << endl;

}

void GetEncryptText()

{

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

{

cout << encrypttext[i].first << " " << encrypttext[i].second << " ";

}

}

void GetDecryptText()

{

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

{

cout << decrypttext[i] << " ";

}

}

private:

int p\_key;//x

};

class EG {

public:

int p, g, k, Da, Db, e, r, m;

EG(int a, int b, int c) {

p = a; g = b; k = c, Ca = 0; Cb = 0; Da = 0; Db = 0; e = 0; r = 0; m = 0;

}

int GetCa() {

return Ca;

}

int GetCb() {

return Cb;

}

void SetCa(const int& a) {

Ca = a;

}

void SetCb(const int& b) {

Cb = b;

}

private:

int Ca, Cb;

};

class SourseText

{

public:

vector <int> soursetext;

void SetSourseElement(const string& stext)

{

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

int q = stext[i];

soursetext.push\_back(q);

}

}

void GetSourseText()

{

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

{

cout << soursetext[i] << " ";

}

}

vector <int> GetTextForEncrypt()

{

return soursetext;

}

};

void EncryptRSA(const vector <int>& stext);

void EncryptDH(const vector <int>& stext);

void EncryptElGam(const vector <int>& stext);

void EncryptSH(const vector <int>& stext);

int ost(int a, int x, int m);

**RSA.cpp:**

#pragma once

#include <ctime>

#include <iostream>

#include <vector>

#include <windows.h>

#include <string>

using namespace std;

class RSA

{

public:

int o\_key; //Открытый ключ

int Na;//p\*q

int Getq()

{

return q;

}

int Getp()

{

return p;

}

void Setq(const int& iq)

{

q = iq;

}

void Setp(const int& ip)

{

p = ip;

}

void GenOpenKey(int& n)

{

while (1)

{

o\_key = n;

if (o\_key % ((p - 1) \* (q - 1)) == 1)

{

o\_key = n;

break;

}

else {

n--;

}

}

}

void GenPrivateKey()// генерация закрытого ключа

{

int k = 1;

while (1)

{

k = k + ((p - 1) \* (q - 1));

if (k % o\_key == 0 && k / o\_key != 1)

{

p\_key = (k / o\_key);

break;

}

}

cout << "Private key: " << p\_key << endl;

}

void EncryptText(const vector <int>& stext)// шифровка текста

{

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

{

long int current, result;

current = stext[i] - 97;

result = 1;

for (long int j = 0; j < o\_key; j++)

{

result = result \* current;

result = result % Na;

}

encrypttext.push\_back(result);

}

}

void DecryptText()// дешифровка текста

{

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

{

long int current, result;

current = encrypttext[i];

result = 1;

for (long int j = 0; j < p\_key; j++)

{

result = result \* current;

result = result % Na;

}

decrypttext.push\_back(result + 97);

}

}

void GetEncryptText()

{

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

{

cout << encrypttext[i] << " ";

}

}

void GetDecryptText()

{

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

{

cout << decrypttext[i] << " ";

}

}

private:

int p\_key;//Закрытый ключ

int p, q;// Простые числа по выбору пользователя

vector <int> encrypttext;

vector <int> decrypttext;

};

class DH

{

public:

int p,g;

long int ya;

string Name;

int GetXkey()

{

return xa;

}

void SetYkey(const int t)

{

ya = 1;

for (int i = 0; i < (t - 1); i++)

{

ya = ya \* g % p;

}

cout << Name << "'s Public Key: " << ya << endl;

}

void SetXkey(const int t, const long int p\_key)

{

xa = 1;

for (int i = 0; i < (t - 1); i++)

{

xa = xa \* p\_key % p;

}

cout << Name << "'s Private Key: " << xa << endl;

}

private:

long int xa;

};

class SH

{

public:

int p;

vector <int> x1;

vector <int> x2;

SH(int a) {

p = a; Ca = 0; da = 0;

}

int GetCa() {

return Ca;

}

void SetCa(const int& a) {

Ca = a;

}

int Getda() {

return da;

}

void Setda(const int& a) {

da = a;

}

private:

int Ca, da;

};

class ElGam

{

public:

int p, g, y, k, decel;

pair <int, int> encrel;

vector <pair<int, int>> encrypttext;

vector <int> decrypttext;

ElGam(int a, int b, int d) {

p = a; g = b; y = 1; p\_key = d; decel = 1; k = 10;

}

int GetPKey()

{

return p\_key;

}

void GenKey(int a, int x, int m)

{

y = 1;

for (int i = 0; i < x; i++) {

y \*= a;

y %= m;

}

cout << "y = " << y << endl;

}

void GetEncryptText()

{

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

{

cout << encrypttext[i].first << " " << encrypttext[i].second << " ";

}

}

void GetDecryptText()

{

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

{

cout << decrypttext[i] << " ";

}

}

private:

int p\_key;//x

};

class EG {

public:

int p, g, k, Da, Db, e, r, m;

EG(int a, int b, int c) {

p = a; g = b; k = c, Ca = 0; Cb = 0; Da = 0; Db = 0; e = 0; r = 0; m = 0;

}

int GetCa() {

return Ca;

}

int GetCb() {

return Cb;

}

void SetCa(const int& a) {

Ca = a;

}

void SetCb(const int& b) {

Cb = b;

}

private:

int Ca, Cb;

};

class SourseText

{

public:

vector <int> soursetext;

void SetSourseElement(const string& stext)

{

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

int q = stext[i];

soursetext.push\_back(q);

}

}

void GetSourseText()

{

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

{

cout << soursetext[i] << " ";

}

}

vector <int> GetTextForEncrypt()

{

return soursetext;

}

};

void EncryptRSA(const vector <int>& stext);

void EncryptDH(const vector <int>& stext);

void EncryptElGam(const vector <int>& stext);

void EncryptSH(const vector <int>& stext);

int ost(int a, int x, int m);

**DH.cpp:**

#include "Class\_for\_crypto\_algorithm.h"

using namespace std;

void EncryptRSA(const vector <int>& stext)

{

RSA rsa;

system("cls");

cout << "=========================RSA" << endl;

int t;

cout << "Select the number p: ";

cin >> t;

rsa.Setp(t);

cout << "Select the number q: ";

cin >> t;

rsa.Setq(t);

rsa.Na = rsa.Getp() \* rsa.Getq();

srand(time(NULL));

int n = rand() % 10000 + 100;// Генерация экспоненты взаимнопростой с (p-1)\*(q-1)

rsa.GenOpenKey(n);

cout << "Open key: " << rsa.o\_key << endl;

rsa.GenPrivateKey();

rsa.EncryptText(stext);

cout << "Your encrypted text: " << endl;

rsa.GetEncryptText();

cout << endl;

rsa.DecryptText();

cout << "Your decrypted text: " << endl;

rsa.GetDecryptText();

system("pause");

}

**SH.cpp:**

#include "Class\_for\_crypto\_algorithm.h"

using namespace std;

void EncryptRSA(const vector <int>& stext)

{

RSA rsa;

system("cls");

cout << "=========================RSA" << endl;

int t;

cout << "Select the number p: ";

cin >> t;

rsa.Setp(t);

cout << "Select the number q: ";

cin >> t;

rsa.Setq(t);

rsa.Na = rsa.Getp() \* rsa.Getq();

srand(time(NULL));

int n = rand() % 10000 + 100;// Генерация экспоненты взаимнопростой с (p-1)\*(q-1)

rsa.GenOpenKey(n);

cout << "Open key: " << rsa.o\_key << endl;

rsa.GenPrivateKey();

rsa.EncryptText(stext);

cout << "Your encrypted text: " << endl;

rsa.GetEncryptText();

cout << endl;

rsa.DecryptText();

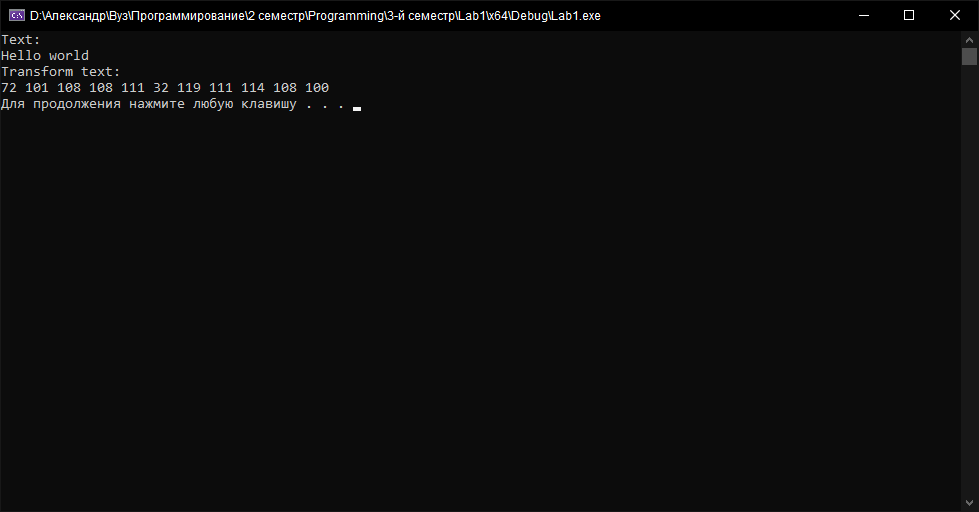
cout << "Your decrypted text: " << endl;

rsa.GetDecryptText();

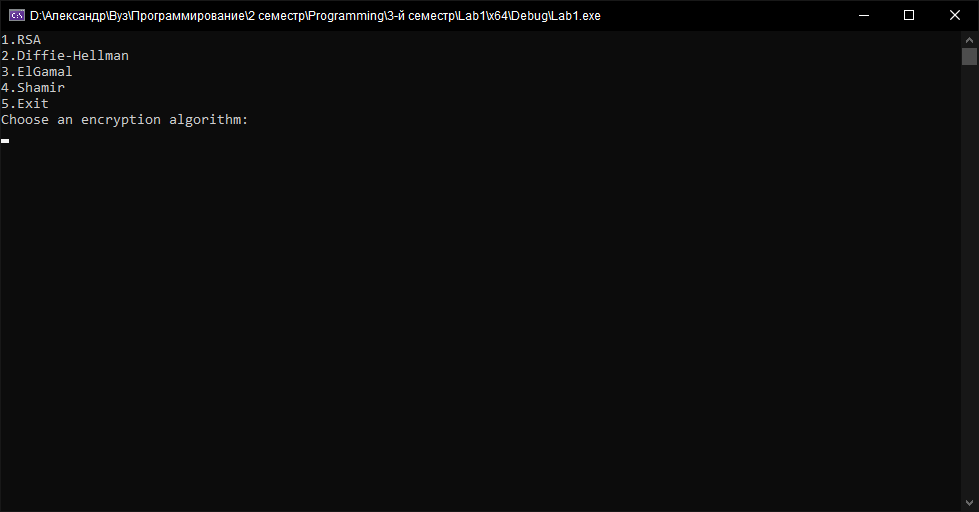
system("pause");

}

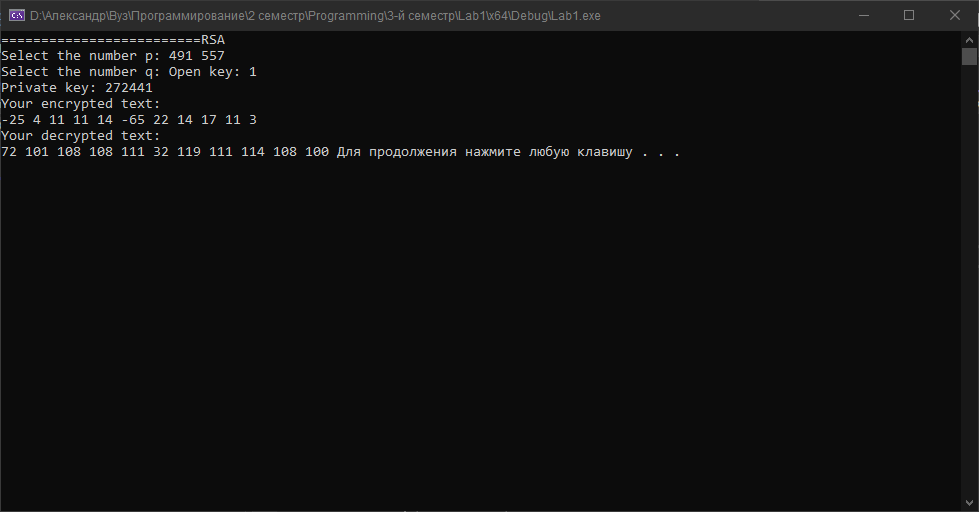
Ввод текста для шифровки



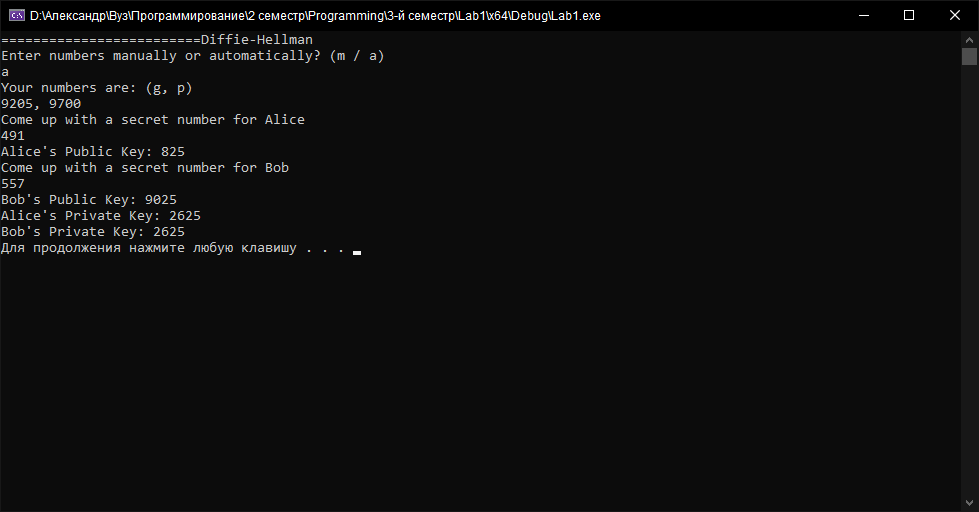
Меню выбора криптопротокола



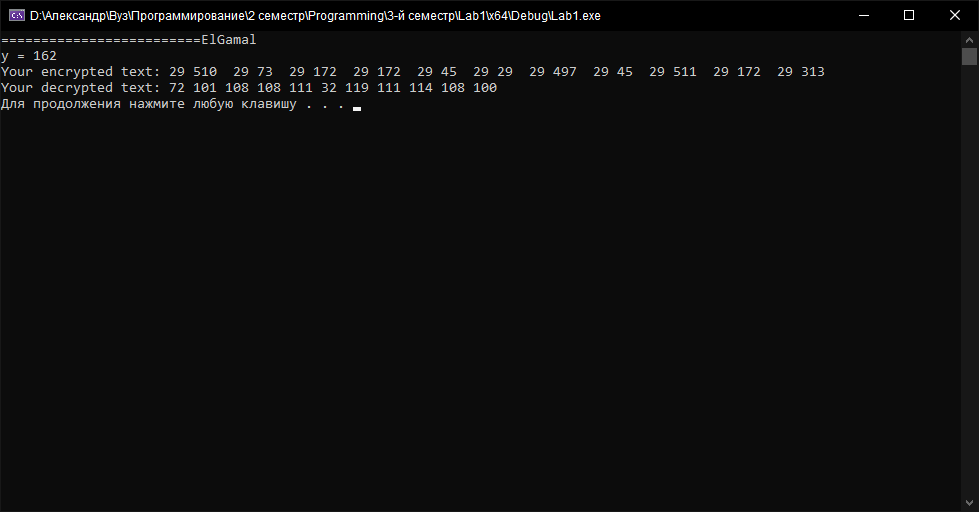
RSA



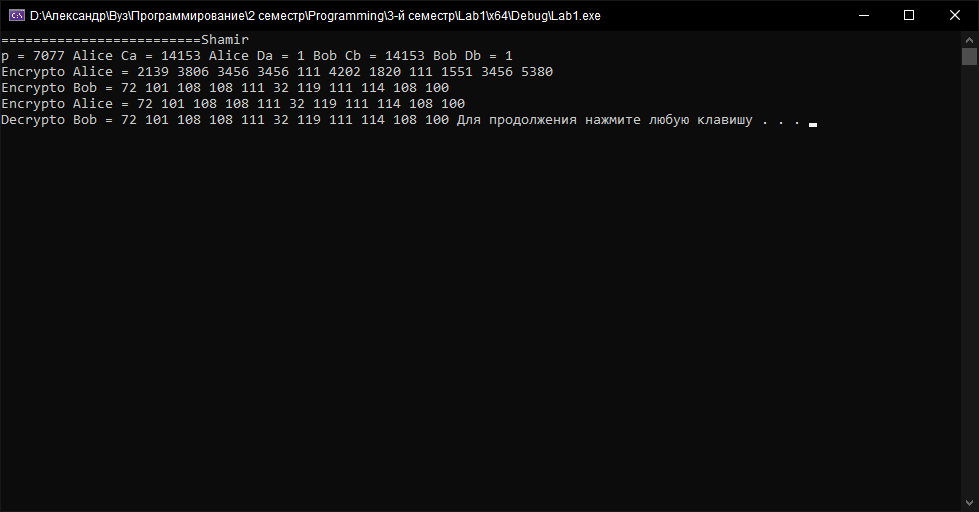
Диффи-Хеллман



Эль гамаль



Криптопротокол Шамира



**C#**

**Programm.cs:**

using System;

namespace Programm

{

class Programm

{

class RSA

{

public int o\_key;

public int Na;

public int Getq()

{

return q;

}

public int Getp()

{

return p;

}

public int Getpkey()

{

return p\_key;

}

public void Setq(int iq)

{

q = iq;

}

public void Setp(int ip)

{

p = ip;

}

public void GenOpenKey(int n)

{

while (true)

{

o\_key = n;

if (o\_key % ((p - 1) \* (q - 1)) == 1)

{

o\_key = n;

break;

}

else

{

n--;

}

}

}

public void GenPrivateKey()// генерация закрытого ключа

{

int k = 1;

while (true)

{

k = k + ((p - 1) \* (q - 1));

if (k % o\_key == 0 && k / o\_key != 1)

{

p\_key = (k / o\_key);

break;

}

}

Console.WriteLine("Private key: {0}\n", p\_key);

}

public void EncryptText(List<int> stext)// шифровка текста

{

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

{

int current;

int result;

current = stext[i] - 97;

result = 1;

for (int j = 0; j < o\_key; j++)

{

result = result \* current;

result = result % Na;

}

encrypttext.Add(result);

}

}

public void DecryptText()// дешифровка текста

{

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

{

int current, result;

current = encrypttext[i];

result = 1;

for (int j = 0; j < p\_key; j++)

{

result = result \* current;

result = result % Na;

}

decrypttext.Add(result + 97);

}

}

public void GetEncryptText()

{

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

{

Console.WriteLine("{0} ", encrypttext[i]);

}

}

public void GetDecryptText()

{

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

{

Console.WriteLine("{0} ", decrypttext[i]);

}

}

private List<int> encrypttext = new List<int>();

private List<int> decrypttext = new List<int>();

private int q;

private int p;

private int p\_key;

}

class DH

{

public int p, g, ya;

public string Name;

public int GetXKey()

{

return xa;

}

public void SetYKey(int t)

{

ya = 1;

for (int i = 0; i < (t - 1); i++)

{

ya = ya \* g % p;

}

Console.WriteLine("{0}`s Public Key: {1}\n", Name, ya);

}

public void SetXKey(int t, int p\_key)

{

xa = 1;

for (int i = 0; i < (t - 1); i++)

{

xa = xa \* p\_key % p;

}

Console.WriteLine("{0}`s Private Key: {1}\n", Name, ya);

}

private int xa;

}

class SH

{

public int p;

public List<int> x1 = new List<int>();

public List<int> x2 = new List<int>();

public SH(int a) { p = a; Ca = 0; da = 0; }

public int GetCa()

{

return Ca;

}

public void SetCa(int a)

{

Ca = a;

}

public int Getda()

{

return da;

}

public void Setda(int a)

{

da = a;

}

private int Ca, da;

}

class ElGam

{

public int p, g, y, k, decel;

public Tuple<int, int> encrel;

public List<Tuple<int, int>> encrypttext = new List<Tuple<int, int>>();

public List<int> decrypttext = new List<int>();

public ElGam(int a, int b, int d)

{

p = a; g = b; y = 1; p\_key = d; decel = 1; k = 10;

}

public int GetPKey()

{

return p\_key;

}

public void GenKey(int a, int x, int m)

{

y = 1;

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

{

y \*= a;

y %= m;

}

Console.WriteLine("y= {0}\n", y);

}

public void GetEncryptText()

{

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

{

Console.Write("{0} ", encrypttext[i]);

}

}

public void GetDecryptText()

{

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

{

Console.Write("{0} ", decrypttext[i]);

}

}

private int p\_key;

}

class SourseText

{

public List<int> soursetext = new List<int>();

public void SetSourseElement(string stext)

{

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

{

int q = stext[i];

soursetext.Add(q);

}

}

public void GetSourseText()

{

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

{

Console.Write(" {0}",soursetext[i]);

}

}

}

void EncryptRSA(ref List<int> text)

{

RSA rsa = new();

Random rnd = new Random();

Console.Clear();

Console.WriteLine("=========================RSA");

Console.Write("Select the number p: ");

rsa.Setp(Convert.ToInt32(Console.ReadLine()));

Console.Write("Select the number q: ");

rsa.Setq(Convert.ToInt32(Console.ReadLine()));

rsa.Na = rsa.Getp() \* rsa.Getq();

rsa.GenOpenKey(rnd.Next(100, 10000));

Console.WriteLine("Open key: {0}", rsa.o\_key);

rsa.GenPrivateKey();

rsa.EncryptText(text);

Console.WriteLine("Your encrypted text:");

rsa.GetEncryptText();

rsa.DecryptText();

Console.WriteLine("Your decrypted text:");

rsa.GetDecryptText();

Console.ReadKey();

}

void EncryptDH()

{

DH dha = new DH();

DH dhb = new DH();

Random rnd = new Random();

dha.Name = "Alice";

dhb.Name = "Bob";

while (true)

{

Console.Clear();

Console.WriteLine("=========================Diffie-Hellman");

string choice;

Console.WriteLine("Enter numbers manually or automatically? (m / a)");

choice = Console.ReadLine();

if (choice == "m")

{

Console.Write("Enter g: ");

dha.g = Convert.ToInt32(Console.ReadLine());

dhb.g = dha.g;

Console.Write("Enter p: ");

dha.p = Convert.ToInt32(Console.ReadLine());

dhb.p = dha.p;

Console.WriteLine("Your numbers are: {0} {1}", dha.g, dha.p);

break;

}

else if (choice == "a")

{

dha.g = rnd.Next(100, 10000);

dhb.g = dha.g;

dha.p = rnd.Next(100, 10000);

dhb.p = dha.p;

Console.WriteLine("Your numbers are: {0} {1}", dha.g, dha.p);

break;

}

else

{

Console.Clear();

Console.WriteLine("Incorrect data, try again");

Console.ReadKey();

continue;

}

}

Console.Write("Come up with a secret number for Alice: ");

int a = Convert.ToInt32(Console.ReadLine());

dha.SetYKey(a);

Console.Write("Come up with a secret number for Bob: ");

int b = Convert.ToInt32(Console.ReadLine());

dhb.SetYKey(b);

dha.SetXKey(a, dhb.ya);

dhb.SetXKey(b, dha.ya);

Console.ReadKey();

}

int ost (int a, int x, int p)

{

int res = 1;

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

{

res \*= a;

res %= p;

}

return res;

}

void EncryptElGam(ref List<int> text)

{

ElGam elgam = new ElGam(593, 123, 8);

Console.Clear();

Console.WriteLine("=========================ElGamal");

elgam.GenKey(elgam.g, elgam.GetPKey(), elgam.p);

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

{

int encelfist = 0, encelsecond = 0;

encelfist = ost(elgam.g, elgam.k, elgam.p);

encelsecond = ost(elgam.y, elgam.k, elgam.p);

encelsecond = (encelsecond \* text[i]) % elgam.p;

elgam.encrel = new Tuple<int, int> (encelfist, encelsecond);

elgam.encrypttext.Add(elgam.encrel);

}

for (int i = 0; i < elgam.encrypttext.Count; i++) //decrypt

{

elgam.decel = ost(elgam.encrypttext[i].Item1, elgam.p - 1 - elgam.GetPKey(), elgam.p);

elgam.decel = elgam.decel \* elgam.encrypttext[i].Item2 % elgam.p;

elgam.decrypttext.Add(elgam.decel);

}

Console.Write("Your encrypted text: ");

elgam.GetEncryptText();

Console.WriteLine();

Console.Write("Your decrypted text: ");

elgam.GetDecryptText();

Console.ReadKey();

}

int GenFirstKey(int p)

{

Random rnd = new Random();

int ca = rnd.Next(1000, (p - 1));

while (true)

{

if (ca > 0)

{

if (ca % (p - 1) == 1)

{

return ca;

}

else ca -= 1;

}

else ca += (p - 1);

}

}

int GenSecondKey(int ca, int p)

{

Random rnd = new Random();

int da = rnd.Next(1000, (p - 1));

while (true)

{

if (da > 0)

{

if ((ca \* da) % (p - 1) == 1)

{

return da;

}

else da -= 1;

}

else da += (p - 1);

}

}

void EncryptSH(ref List<int> text)

{

Console.Clear();

Console.WriteLine("=========================Shamir");

Random rnd = new Random();

int p = rnd.Next(1000, 10000);

SH ash = new SH(p);

SH bsh = new SH(p);

ash.SetCa(GenFirstKey(ash.p));

ash.Setda(GenSecondKey(ash.GetCa(), ash.p));

bsh.SetCa(GenFirstKey(bsh.p));

bsh.Setda(GenSecondKey(bsh.GetCa(), bsh.p));

Console.WriteLine("p = {0}, Alice Ca = {1}, Alice Da = {2}, Bob Cb = {3}, Bob Db = {4}", ash.p, ash.GetCa(), ash.Getda(), bsh.GetCa(), bsh.Getda());

Console.Write("Encrypto Alice = ");

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

{

int x1;

x1 = ost(text[i], ash.GetCa(), ash.p);

ash.x1.Add(x1);

Console.Write("{0} ",ash.x1[i]);

}

Console.WriteLine();

Console.Write("Encrypto Bob = ");

for (int i = 0; i < ash.x1.Count; i++)

{

int x1;

x1 = ost(ash.x1[i], bsh.GetCa(), bsh.p);

bsh.x1.Add(x1);

Console.Write("{0} ", bsh.x1[i]);

}

Console.WriteLine();

Console.Write("Encrypto Alice = ");

for (int i = 0; i < bsh.x1.Count; i++)

{

int x2;

x2 = ost(bsh.x1[i], ash.Getda(), ash.p);

ash.x2.Add(x2);

Console.Write("{0} ", ash.x2[i]);

}

Console.WriteLine();

Console.Write("Decrypto Bob = ");

for (int i = 0; i < ash.x2.Count; i++)

{

int x2;

x2 = ost(ash.x2[i], bsh.Getda(), bsh.p);

bsh.x2.Add(x2);

Console.Write("{0} ", bsh.x1[i]);

}

Console.ReadKey();

}

static int Main(string[] args)

{

var programm = new Programm();

SourseText soursetext = new();

string stext;

Console.WriteLine("Enter the source text: ");

stext = Console.ReadLine();

Console.Clear();

soursetext.SetSourseElement(stext);

Console.WriteLine("Your text: {0}", stext);

Console.Write("Transform text: ");

soursetext.GetSourseText();

Console.ReadKey();

while(true)

{

Console.Clear();

string num;

Console.WriteLine("1.RSA \n2.Diffie-Hellman \n3.ElGamal \n4.Shamir \n5.Exit");

num = Console.ReadLine();

switch(num)

{

case "1":

programm.EncryptRSA(ref soursetext.soursetext);

break;

case "2":

programm.EncryptDH();

break;

case "3":

programm.EncryptElGam(ref soursetext.soursetext);

break;

case "4":

programm.EncryptSH(ref soursetext.soursetext);

break;

case "5":

return 0;

default:

Console.Clear();

Console.WriteLine("Invalid number\n Try again");

Console.ReadKey();

break;

}

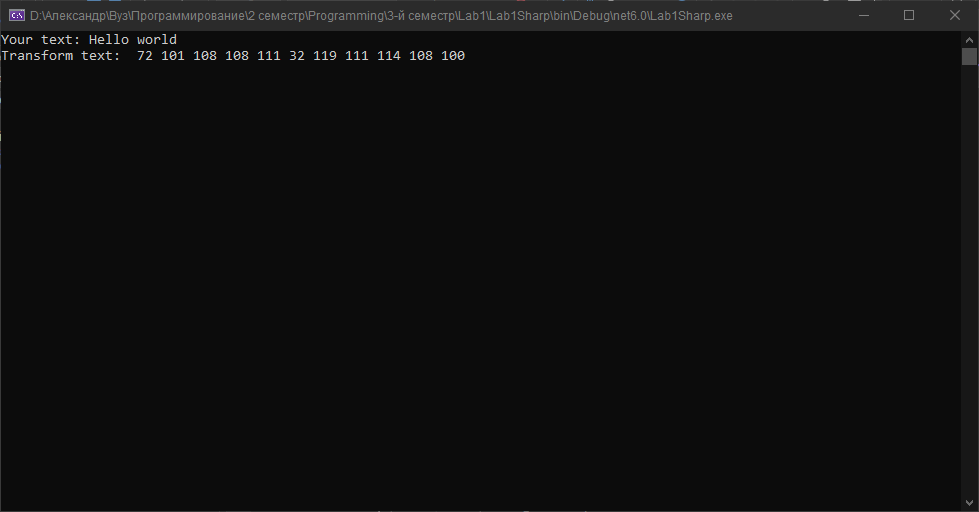
}

}

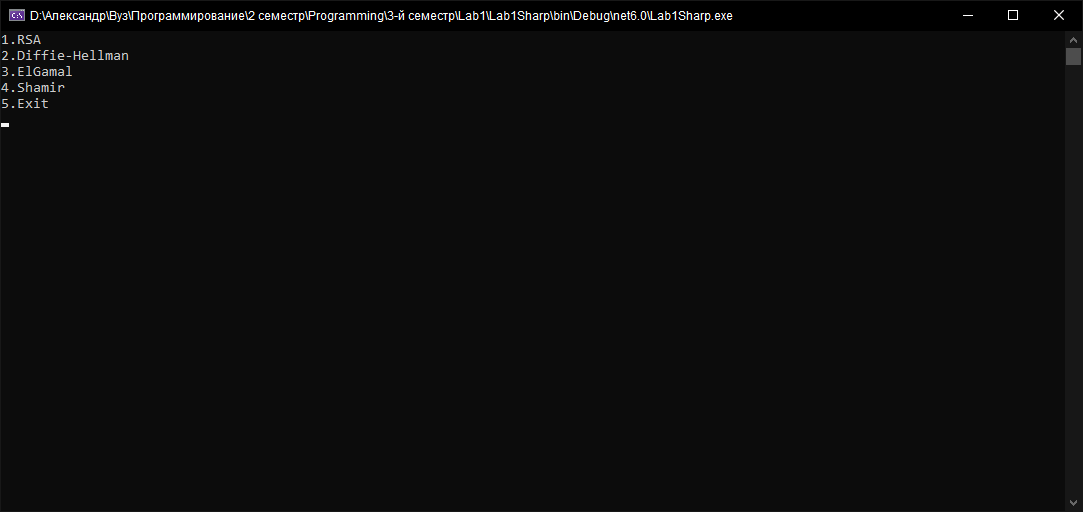
}

}

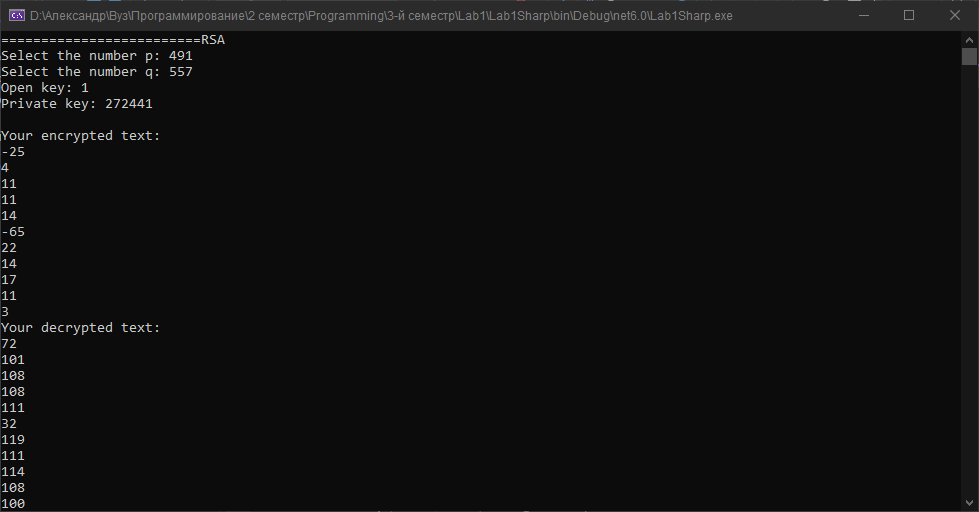
Ввод текста для шифровки

****

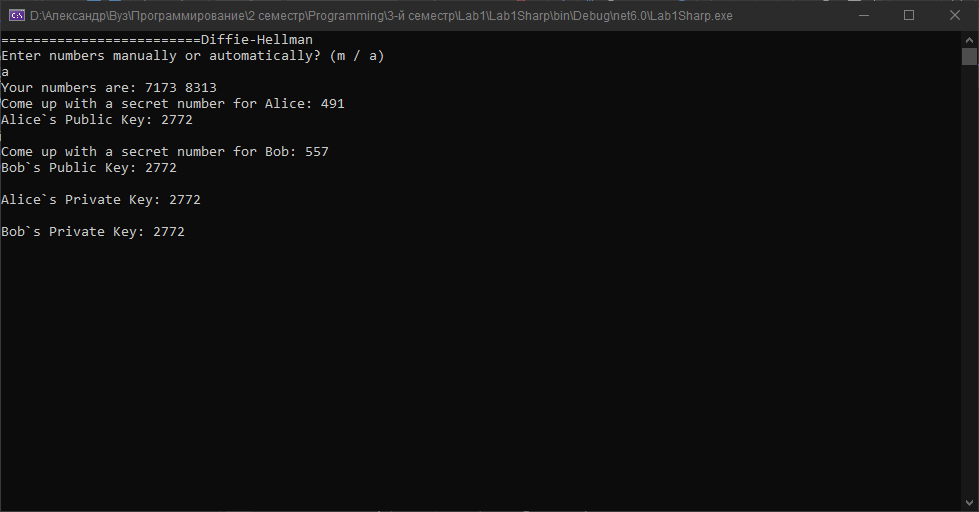
Меню выбора криптопротокола

****

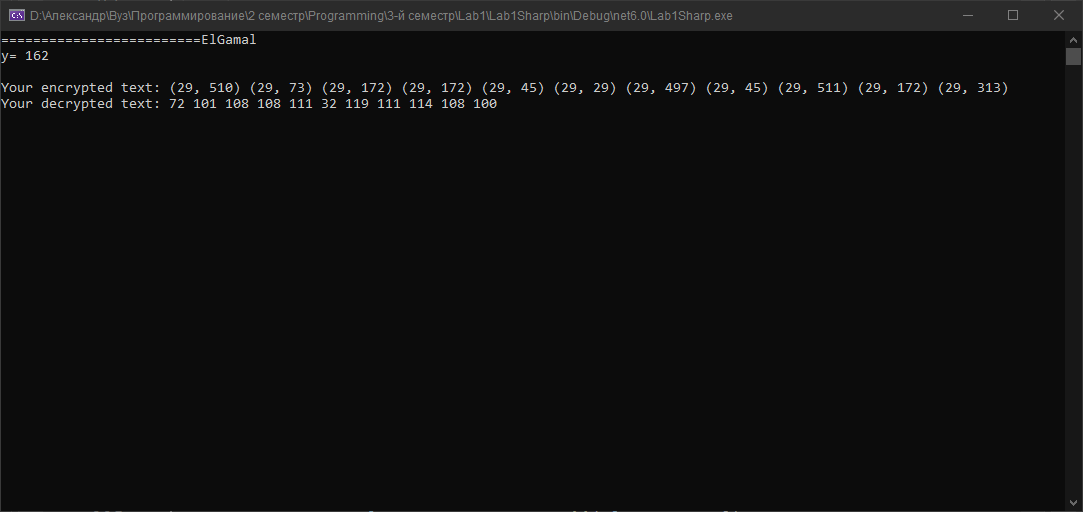
RSA

****

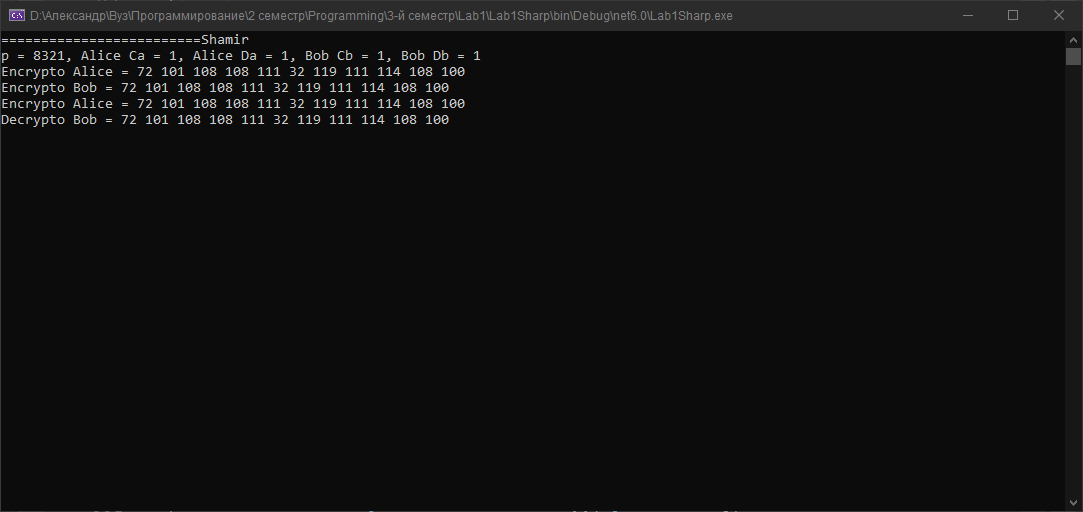
Диффи-Хеллман

****

Эль гамаль

****

Криптопротокол Шамира

****