|  |
| --- |
| **МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ**  **федеральное государственное бюджетное образовательное учреждение**  **высшего образования**  **«Южно-Российский государственный**  **политехнический университет (НПИ) имени М.И. Платова»**  ФАКУЛЬТЕТ "Информационных технологий и управления"  КАФЕДРА "Программное обеспечение вычислительной техники"  НАПРАВЛЕНИЕ "Программная инженерия"  **ОТЧЁТ**  **по лабораторным работам**  по дисциплине: ***Разработка и анализ алгоритмов***  Выполнили студенты 2 курса 5 группы \_Леднев Евгений Владимирович\_\_  Курочкин Дмитрий Сергеевич\_\_  Принял ст. преп. каф. ПОВТ Спиридонова Ирина Артуровна  должность, степень, звание Фамилия И.О.  «\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_ 2020 г. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  Подпись        **г. Новочеркасск 2020 г.** |

**Оглавление**

[**Введение** 3](#_Toc61811191)

[**Лабораторная работа №1** 3](#_Toc61811192)

[**Лабораторная работа №2** 28](#_Toc61811193)

[**Лабораторная работа №3** 21](#_Toc61811194)

[**Лабораторная работа №4** 35](#_Toc61811195)

[**Заключение** 43](#_Toc61811196)

[**Список использованных источников** 45](#_Toc61811197)

**ВВЕДЕНИЕ**

Цель дисциплины «Разработка и анализ алгоритмов» - ознакомление с фундаментальными алгоритмами обработки данных, а также с современными методами исследования алгоритмов и оценки их алгоритмической сложности.

Практические и лабораторные занятия посвящены разработке, анализу

программной реализации алгоритмов решения следующих задач:

Созданию блок-схемы, отвечающей всем критериям.

Рассмотрению классических методов сортировки данных

(сортировка «пузырьком», сортировка вставками, сортировка слиянием, быстрая сортировка и их модификации).

Рассмотрению примеров, прямых методов и способов построения алгоритмов решения вычислительных задач линейной алгебры: перемножения матриц и решения систем линейных алгебраических уравнений (СЛАУ), обращения матрицы (модификации метода Гаусса, метод Жордана).

Рассмотрению примеров, итерационных методов и

способов построения рекурсивных алгоритмов решения вычислительных задач математического анализа. В качестве наглядного примера выбраны квадратурные методы численного интегрирования.

## **Лабораторная работа №1**

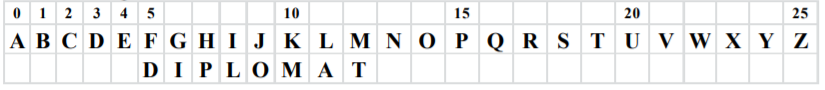
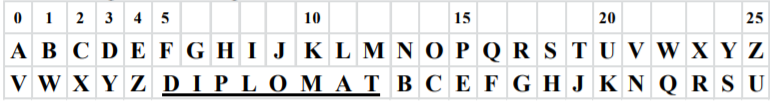
**Тема:** Криптография и криптологические системы.

**Выполнил:** Леднев Евгений Владимирович, ФИТУ 2-5.

**Задание:**Разработать программную реализацию системы шифрования Цезаря, аффинной системы подстановки Цезаря, шифра “двойной квадрат Уитстона”, шифра двойной перестановки.

**Шифр Цезаря**

Алгоритм шифра:

1. Пользователь вводит исходный текст и 2 ключа (ключ-сдвиг и ключевое слово).
2. Из ключевого слова убираем повторяющиеся символы.
3. Ключевое слово записывается под буквами алфавита, начиная с буквы, числовой код которой совпадает с ключом-сдвигом.
4. Далее после ключевого слова продолжаем заполнять ряд буквами алфавита, исключая буквы в ключевом слове.
5. Заменяем каждую букву исходного текста буквой, находящейся под ней.

Готово, текст успешно зашифрован.

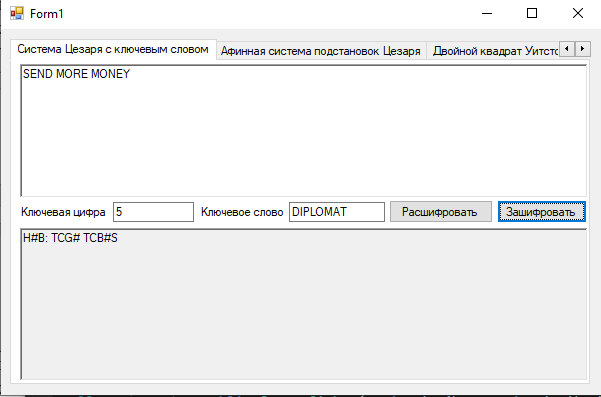


Рисунок 1. Программная реализация зашифровки шифра Цезаря

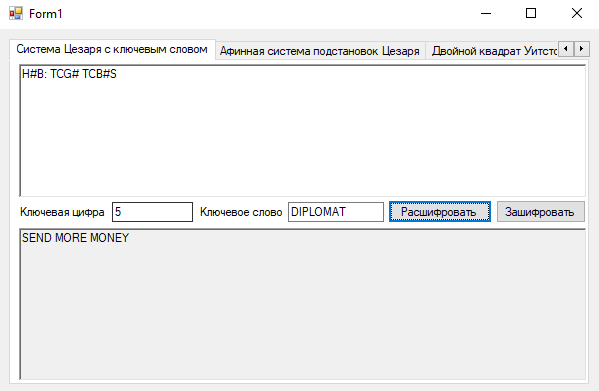
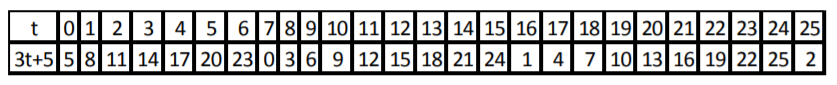
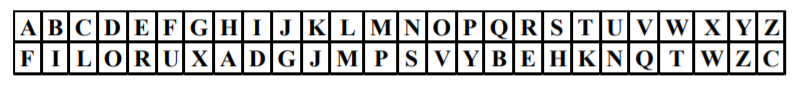


Рисунок 2. Программная реализация дешифровки шифра Цезаря

Как мы видим из примера, программа отлично работает и с русским и английским алфавитом, а так-же автоматически обрабатывает ключ-сдвиг, деля его по модулю на длину алфавита.

**Аффинный шифр Цезаря**

Алгоритм шифра:

1. Пользователь вводит исходный текст и 2 численных ключа, при этом первый ключ и количество символов в алфавите должны быть взаимно простыми.
2. Формируется массив, длиной равной длине алфавита. Где каждый элемент вычисляется по формуле, где a и b – ключи, m – длина алфавита, а t – индекс элемента.
3. Получаем набор чисел, которые в результате преобразовываем в буквы, согласно индексу.
4. Заменяем каждую букву исходного текста буквой, находящейся под ней.

Готово, текст успешно зашифрован.

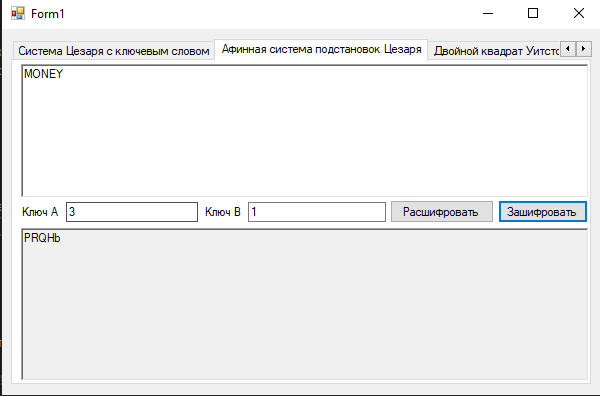


Рисунок 3. Программная реализация зашифровки Аффинного шифра.

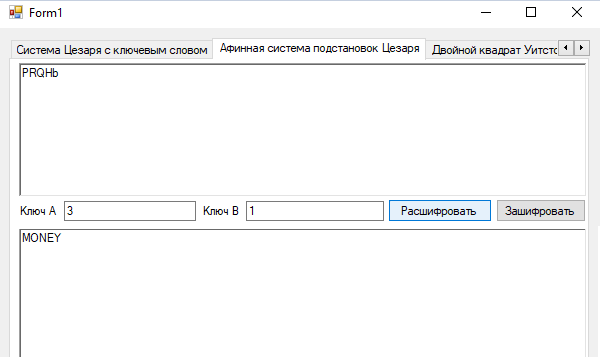


Рисунок 4. Программная реализация дешифровки Аффинного шифра.

**Двойной квадрат Уитстона**

Алгоритм шифра:

1. Пользователь вводит исходных текст и 2 любых ключа.
2. Из ключей убираются повторяющиеся символы и пробелы.
3. Пробелы в исходном тексте заменяются на “\_”.
4. Если длина текста нечетная, то в конец добавляется “\_”.
5. Создаются два квадрата, начинающиеся с своего ключа соответственно. Далее квадрат заполняется своим перемешанным алфавитом, который не содержит символы ключей.
6. Исходное сообщение разбивается на биграммы, где первая буква биграммы находится в левой таблице, а вторая - в правой. Затем используя двумерный массив находим новую биграмму с помощью прямоугольника.
7. Соединяем биграммы и заменяем “\_” на пробел.

Готово, текст успешно зашифрован.

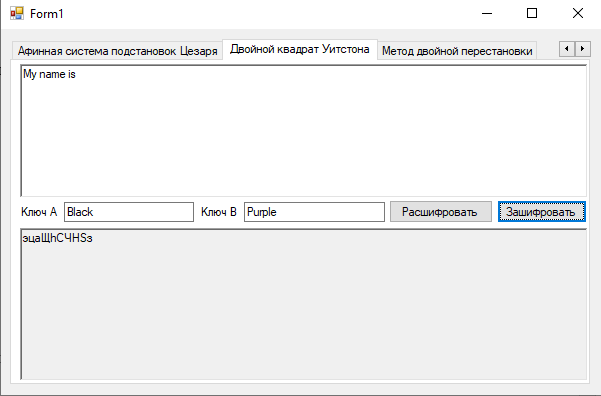


Рисунок 5. Программная реализация зашифровки “Двойной квадрат Уитстона”.

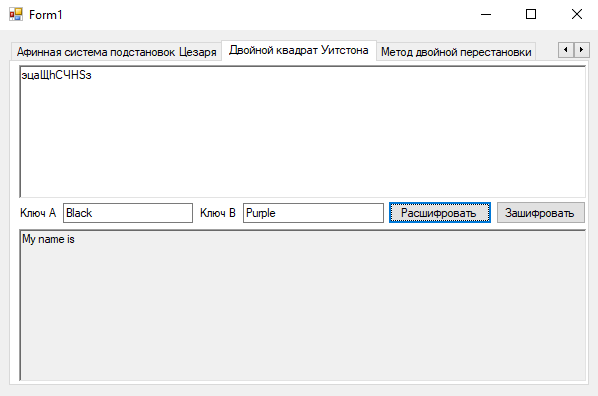


Рисунок 6. Программная реализация дешифровки “Двойной квадрат Уитстона”.

Программа отлично работает с английским, русским алфавитом. Умеет шифровать символы и перевидит “\_” в пробел.

**Шифр Двойной перестановки**

Алгоритм шифра:

1. Пользователь вводит исходный текст и два любых ключа (программа сама преобразовывает слова в числа”.
2. В исходном тексте пробелы заменяются на “\_”.
3. Текст помещается в таблицу, таблица будет сформирована, согласно ключам пользователя (длина 2 ключа = длина исходного текста / длину 1 ключа).
4. Меняем столбцы местами по порядку следования в алфавите букв.
5. Меняем строки местами по порядку следования чисел.
6. Извлекаем из таблицы текст.

Готово, текст успешно зашифрован

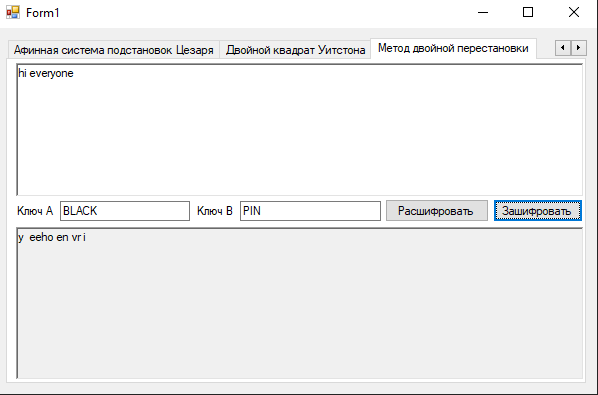


Рисунок 7. Программная реализация зашифровки шифра двойной перестановки.

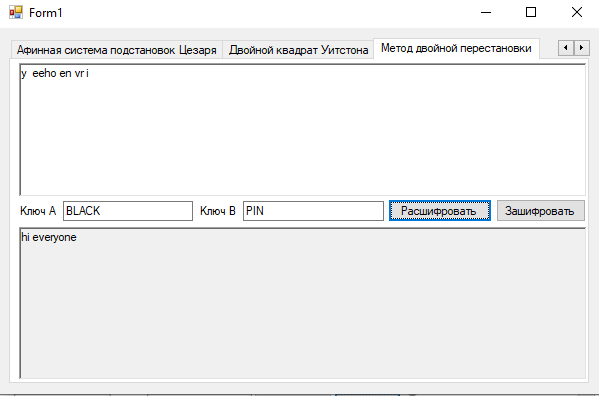


Рисунок 8. Программная реализация дешифровки шифра двойной перестановки.

**Листинг:**

**WindowsForms:**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace CryptoForms

{

static class Program

{

/// <summary>

/// Главная точка входа для приложения.

/// </summary>

[STAThread]

static void Main()

{

Application.EnableVisualStyles();

Application.SetCompatibleTextRenderingDefault(false);

Application.Run(new Form1());

}

}

}

**Шифр Цезаря:**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace Cryptography.Ciphers

{

class CaesarCipher

{

private string strEuABC = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";

private char[] EuABC;

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

string KeyWord;

private string text;

private Dictionary<char, char> EuEncoding = new Dictionary<char, char>();

private Dictionary<char, char> EuDecoding = new Dictionary<char, char>();

private char[] EuABCwithoutKeyWord;

private char[] EuABCwithCipher = new char[26];

/// <summary>

/// keyNum - ключевое число от 0 до 25, keyWord - ключевое слово,

/// text - текст, который нужно зашифровать или расшифровать.

/// </summary>

/// <param name="keyNum"></param>

/// <param name="keyWord"></param>

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

public CaesarCipher(string keyNum, string keyWord, string text)

{

EuABC = strEuABC.ToCharArray();

this.KeyNum = Convert.ToInt32(keyNum);

this.KeyWord = new string(keyWord.Distinct().ToArray()).Replace(" ", "").ToUpper(); ; // Удаляет пробелы и повторяющиеся символы

this.text = text;

foreach (char symbol in KeyWord)

{

strEuABC = strEuABC.Remove(strEuABC.IndexOf(symbol), 1);

}

char[] EuABCwithoutKeyWord = strEuABC.ToCharArray();

int k = 0; // auxiliary index

for (int i = KeyNum, j = 0; j < KeyWord.Length || i < EuABC.Length; j++, i++)

{

if (j < KeyWord.Length)

{

EuABCwithCipher[i] = KeyWord[j];

}

else

{

EuABCwithCipher[i] = EuABCwithoutKeyWord[k];

k++;

}

}

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

{

EuABCwithCipher[i] = EuABCwithoutKeyWord[k];

k++;

}

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

{

EuEncoding.Add(EuABC[i], EuABCwithCipher[i]);

EuDecoding.Add(EuABCwithCipher[i], EuABC[i]);

}

}

/// <summary>

/// Возвращает зашифрованную строку

/// Return encode string

/// </summary>

/// <returns></returns>

public string Encode()

{

string code = "";

foreach (char symbol in text)

{

code += EuEncoding[symbol].ToString();

}

return code;

}

/// <summary>

/// Возвращает расшифрованную строку

/// Return decode string

/// </summary>

/// <returns></returns>

public string Decode()

{

string code = "";

foreach (char symbol in text)

{

code += EuDecoding[symbol].ToString();

}

return code;

}

}

}

**Аффинная система подстановок Цезаря**

using System;

using System.Collections.Generic;

using System.Text;

namespace Cryptography.Ciphers

{

class AffineCaesarCipher

{

private string strEuABC = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";

private char[] EuABC;

private int a;

private int b;

private string text;

private Dictionary<char, char> EuEncoding = new Dictionary<char, char>();

private Dictionary<char, char> EuDecoding = new Dictionary<char, char>();

/// <summary>

/// В данном преобразовании буква, соответствующая число t,

/// заменяется на букву, соотвутствующую числовому значению (A\*t+B)mod m (m - длина алфавита)

/// A - первый ключ, В - второй ключ

/// A и B должны быть взаимно простыми

/// НОД (а, 26(длина алфавита)) = 1 !!!

/// text - текст, который нужно зашифровать или расшифровать.

/// </summary>

/// <param name="A"></param>

/// <param name="B"></param>

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

public AffineCaesarCipher(string A, string B, string text)

{

char[] EuABC = strEuABC.ToCharArray();

this.a = Convert.ToInt32(a);

this.b = Convert.ToInt32(b);

this.text = text;

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

{

EuEncoding.Add(EuABC[i], EuABC[(a \* i + b) % EuABC.Length]);

EuDecoding.Add(EuABC[(a \* i + b) % EuABC.Length], EuABC[i]);

}

}

/// <summary>

/// Возвращает зашифрованную строку

/// Return encode string by Affine Ceasar cipher

/// </summary>

/// <returns></returns>

public string Encode()

{

string code = "";

foreach (char symbol in text)

{

code += EuEncoding[symbol].ToString();

}

return code;

}

/// <summary>

/// Возвращает расшифрованную строку

/// Return decode string by Affine Ceasar cipher

/// </summary>

/// <returns></returns>

public string Decode()

{

string code = "";

foreach (char symbol in text)

{

code += EuDecoding[symbol].ToString();

}

return code;

}

}

}

**Двойной квадрат Уитстона:**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Cryptography.Ciphers

{

class WheatstoneDoubleSquare

{

//private static string Alphabet = "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyzАБВГДЕЁЖЗИЙКЛМНОПРСТУФХЦЧШЩЪЫЬЭЮЯабвгдеёжзийклмнопрстуфхцчшщъыьэюя0123456789.,?!\*/+-=\_()%;:#";//Набор сиволов

//Перемешанный алфавит для повышения качества шифра

private static string AlphabetA = "ъ=#VКsэGу6ЮлнzSОqhWЫxCФN.HдM8ИdcFjtQ+ITхЦв;awrтDУп:Бж5ЁХЛLчvёгE2Аu70аГ/3Aошeе)бПТЩЭЕZ9mXРЯpygкыi?%юй4Вl,сДьЪзН(ф-PBШOСb\_Ж!рoщUЬR\*JямnKЧЗ1YМцfkЙи";//Набор сиволов

private static string AlphabetB = "ъhRbБHOЪЬp;6zяАзуЕDwцэ(\_cGmЗж7PаФыоIчЫпП+К%\*N4vЙrЦl5KСТШjёшZiьЯ9НUДJУМQ=б)Ч0W:.?щРЩ/XЭХdCГAkкn1SЖ!юqV,Lфт-Bs3н82EрЛYMд#ВйFОеuИЮлftaхЁyмeгTxogсви";//Набор сиволов

private static int columns = 12;//Столбцы

private static int rows = 12;//Ряды

private char[,] FirstSquare = new char[rows, columns];//Первый квадрат Уитстона

private char[,] SecondSquare = new char[rows, columns];//Второй квадрат Уитстона

private string firstKey = "";//Первый ключ

private string secondKey = "";//Второй ключ

private string encrypt = "";//Зашифрованные данные

private string decrypt = "";//Дешифрованные данные

private string text = "";

/// <summary>

/// firstKey - первый ключ (Определяет перестановку столбцов)

/// secondKey - второй ключ (Определяет перестановку строк)

/// text - текст, который нужно зашифровать или расшифровать.

/// </summary>

/// <param name="firstKey"></param>

/// <param name="secondKey"></param>

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

public WheatstoneDoubleSquare(string firstKey, string secondKey, string text)

{

//Удаление повторяющихся символов из первого ключа

this.firstKey = new string(firstKey.Distinct().ToArray()).Replace(" ", "");

//Удаление повторяющихся символов из второго ключа

this.secondKey = new string(secondKey.Distinct().ToArray()).Replace(" ", "");

this.text = text;

}

private string GetKeyLess(string \_Key, string Alphabet\_)//Вернуть алфавит без указанного ключа

{

string ResidualAlphabet = Alphabet\_;

for (int i = 0; i < \_Key.Length; i++)//Алфавит без Ключевого слова

{

ResidualAlphabet = ResidualAlphabet.Replace(\_Key[i].ToString(), "");

}

return ResidualAlphabet;

}

//Первый квадрат

private void SetFirstSquare()

{

string KeyLess = GetKeyLess(firstKey, AlphabetA);

int index = 0, k = 0;

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

{

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

{

FirstSquare[i, j] = (index < firstKey.Length) ? firstKey[index++] : KeyLess[k++];

}

}

}

//Второй квадрат

private void SetSecondSquare()

{

string KeyLess = GetKeyLess(secondKey, AlphabetB);

int index = 0, k = 0;

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

{

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

{

SecondSquare[i, j] = (index < secondKey.Length) ? secondKey[index++] : KeyLess[k++];

}

}

}

//Создание двух квадратов и замена пробелов на подчеркивания

private void CipherDerivation()

{

SetFirstSquare();

SetSecondSquare();

text = text.Replace(" ", "\_");

if (text.Length % 2 == 1)

text += "\_";

}

/// <summary>

/// Возвращает засшифрованную строку

/// Return encode string

/// </summary>

/// <returns></returns>

public string Encoded()

{

CipherDerivation();

string Pair = "";

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

{

Pair += text[i];

if (i % 2 == 1)

{

encrypt += EncodeElementPairCipher(Pair);

Pair = "";

}

}

return encrypt;

}

/// <summary>

/// Возвращает расшифрованную строку

/// Return decode string

/// </summary>

/// <returns></returns>

public string Decoded()

{

SetFirstSquare();

SetSecondSquare();

string Pair = "";

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

{

Pair += text[i];

if (i % 2 == 1)

{

decrypt += DecodeElementPairCipher(Pair);

Pair = "";

}

}

return decrypt.Replace("\_", " ").Trim();

}

//Поиск позиции элемента в таблице

private void SearchIndexToArray(char[,] Square, char SearchChar, out int TableRows, out int TableColumns)

{

TableRows = -1;

TableColumns = -1;

int Rows = Square.GetUpperBound(0) + 1;//Ряды

int Columns = Square.GetUpperBound(1) + 1;//Колонки

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

{

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

{

if (Square[i, j] == SearchChar)

{

TableRows = i;

TableColumns = j;

}

}

}

}

//Замена пары символов на зашифрованную пару

private string EncodeElementPairCipher(string StrPair)

{

string Pair = "";

SearchIndexToArray(FirstSquare, StrPair[0], out int Rows\_0, out int Columns\_0);//Позиция 1 символа в 1 квадрате

SearchIndexToArray(SecondSquare, StrPair[1], out int Rows\_1, out int Columns\_1);//Позиция 2 символа в 2 квадрате

if (Rows\_0 == Rows\_1)//Если в 1 ряду

{

Pair += (Rows\_0 == -1 || Rows\_1 == -1) ? StrPair : SecondSquare[Rows\_0, Columns\_0].ToString() + FirstSquare[Rows\_1, Columns\_1].ToString();

}

else//Если в разных рядах

{

Pair += (Rows\_0 == -1 || Rows\_1 == -1) ? StrPair : SecondSquare[Rows\_0, Columns\_1].ToString() + FirstSquare[Rows\_1, Columns\_0].ToString();

}

return Pair;

}

//Замена пары символов на дешифрованную пару

private string DecodeElementPairCipher(string StrPair)

{

string Pair = "";

SearchIndexToArray(SecondSquare, StrPair[0], out int Rows\_0, out int Columns\_0);//Позиция 1 символа в 1 квадрате

SearchIndexToArray(FirstSquare, StrPair[1], out int Rows\_1, out int Columns\_1);//Позиция 2 символа в 2 квадрате

if (Rows\_0 == Rows\_1)//Если в 1 ряду

{

Pair += (Rows\_0 == -1 || Rows\_1 == -1) ? StrPair : FirstSquare[Rows\_0, Columns\_0].ToString() + SecondSquare[Rows\_1, Columns\_1].ToString();

}

else//Если в разных рядах

{

Pair += (Rows\_0 == -1 || Rows\_1 == -1) ? StrPair : FirstSquare[Rows\_0, Columns\_1].ToString() + SecondSquare[Rows\_1, Columns\_0].ToString();

}

return Pair;

}

}

}

**Шифр двойной перестановки**

using System;

using System.Collections.Generic;

using System.Text;

namespace Cryptography.Ciphers

{

class DoublePermutationOfCharacters

{

private string keyWord1;

private string keyWord2;

private string Text;

private char[,] TableEncode;

private char[,] TableDecode;

/// <summary>

/// keyWord1 - первый ключ (Определяет перестановку столбцов)

/// keyWord2 - второй ключ (Определяет перестановку строк)

/// text - текст, который нужно зашифровать или расшифровать.

/// </summary>

/// <param name="keyWord1"></param>

/// <param name="keyWord2"></param>

/// <param name="Text"></param>

public DoublePermutationOfCharacters(string keyWord1, string keyWord2, string Text)

{

this.keyWord1 = keyWord1.ToUpper();

this.keyWord2 = keyWord2.ToUpper();

this.Text = Text.ToUpper();

}

/// <summary>

/// Возвращает зашифрованную строку

/// Return encode string

/// </summary>

/// <returns></returns>

public string Encoded()

{

string Code = "";

Encoding();

for (int i = 1; i < keyWord1.Length + 1; ++i)

{

for (int j = 1; j < keyWord2.Length + 1; ++j) // keyWord 1 = СКАНЕР

{ // keyWord 2 = 4123

Code += TableEncode[j, i].ToString();

}

}

return Code;

}

/// <summary>

/// Возвращает расшифрованную строку

/// Return decoded string

/// </summary>

/// <returns></returns>

public string Decoded()

{

string Code = "";

Decoding();

for (int i = 1; i < keyWord2.Length + 1; ++i)

{

for (int j = 1; j < keyWord1.Length + 1; ++j)

{

Code += TableDecode[j, i].ToString();

}

}

return Code;

}

private void Encoding()

{

TableEncode = new char[keyWord2.Length + 1, keyWord1.Length + 1];

int n = 0;

TableEncode[0, 0] = ' ';

for (int i = 1; i < keyWord2.Length + 1; ++i)

{

TableEncode[i, 0] = keyWord2[i - 1];

for (int j = 1; j < keyWord1.Length + 1; ++j) // keyWord 1 = СКАНЕР

{ // keyWord 2 = 4123

TableEncode[0, j] = keyWord1[j - 1];

if (n == Text.Length)

TableEncode[i, j] = ' ';

TableEncode[i, j] = Text[n];

n++;

}

}

// permutation of columns

for (int i = 1; i < keyWord1.Length; ++i)

{

for (int j = i + 1; j < keyWord1.Length + 1; ++j)

{

if (TableEncode[0, i] > TableEncode[0, j])

for (int k = 0; k < keyWord2.Length + 1; ++k)

Swap(ref TableEncode[k, i], ref TableEncode[k, j]);

}

}

// permutation of lines

for (int i = 1; i < keyWord2.Length; ++i)

{

for (int j = i + 1; j < keyWord2.Length + 1; ++j)

{

if (TableEncode[i, 0] > TableEncode[j, 0])

for (int k = 0; k < keyWord1.Length + 1; ++k)

Swap(ref TableEncode[i, k], ref TableEncode[j, k]);

}

}

}

private void Decoding()

{

TableDecode = new char[keyWord1.Length + 1, keyWord2.Length + 1];

char[] SortedkeyWord1 = SortString(keyWord1).ToCharArray();

char[] SortedkeyWord2 = SortString(keyWord2).ToCharArray();

int n = 0;

TableDecode[0, 0] = ' ';

for (int i = 1; i < SortedkeyWord1.Length + 1; ++i)

{

TableDecode[i, 0] = SortedkeyWord1[i - 1];

for (int j = 1; j < SortedkeyWord2.Length + 1; ++j) // keyWord 1 = СКАНЕР

{ // keyWord 2 = 4123

TableDecode[0, j] = SortedkeyWord2[j - 1];

TableDecode[i, j] = Text[n];

n++;

}

}

// permutation of columns

for (int i = 0; i < keyWord2.Length; ++i)

{

for (int j = i; j < keyWord2.Length; ++j)

{

if (keyWord2[i] == SortedkeyWord2[j])

{

for (int k = 0; k < SortedkeyWord1.Length + 1; ++k)

Swap(ref TableDecode[k, i + 1], ref TableDecode[k, j + 1]);

Swap(ref SortedkeyWord2[i], ref SortedkeyWord2[j]);

}

}

}

// permutation of lines

for (int i = 0; i < keyWord1.Length; ++i)

{

for (int j = i; j < keyWord1.Length; ++j)

{

if (keyWord1[i] == SortedkeyWord1[j])

{

for (int k = 0; k < SortedkeyWord2.Length + 1; ++k)

Swap(ref TableDecode[i + 1, k], ref TableDecode[j + 1, k]);

Swap(ref SortedkeyWord1[i], ref SortedkeyWord1[j]);

}

}

}

}

private void Swap(ref char A, ref char B)

{

char ch = A;

A = B;

B = ch;

}

private string SortString(string Text)

{

char[] text = Text.ToCharArray();

char temp;

for (int i = 0; i < text.Length - 1; i++)

{

for (int j = i + 1; j < text.Length; j++)

{

if (text[i] > text[j])

{

temp = text[i];

text[i] = text[j];

text[j] = temp;

}

}

}

return new string(text);

}

}

}

## **Лабораторная работа №2**

**Тема:** криптографические протоколы.

**Цель работы:** освоение и программная реализация работы криптосистем, приобретение навыков их использования на практике для решения задач шифрования, дешифрования данных и обмена ключами.

**Выполнил:** Леднев Евгений Владимирович, ФИТУ 2-5.

**Задание:**Используя полученные данные, выполнить программную реализацию криптографического протокола Шамира для передачи сообщения М от абонента А к абоненту В.

**Программа работы.**

Этап 1. Генерация ключей абонентов А и В.

Этап 2. Преобразование сообщения в числовой эквивалент.

Этап 3. Используем трехпроходный алгоритм Шамира.

Этап 4. Преобразование полученного значения к формату текста.

***Вариант 10:*** *P = 34231; m = CSS.*

**Алгоритм протокола:**

1. абонент А зашифровывает открытое сообщение своим ключом и передает это зашифрованное сообщение абоненту В;
2. абонент В зашифровывает полученное сообщение своим ключом и передает это (дважды зашифрованное) сообщение обратно абоненту А;
3. абонент А расшифровывает сообщение от абонента В своим ключом, тем самым снимая свое шифрование, и возвращает зашифрованное только ключом абонента В сообщение абоненту В;
4. абонент В расшифровывает сообщение своим ключом и тем самым восстанавливает исходное открытое сообщение.

**Программная реализация**

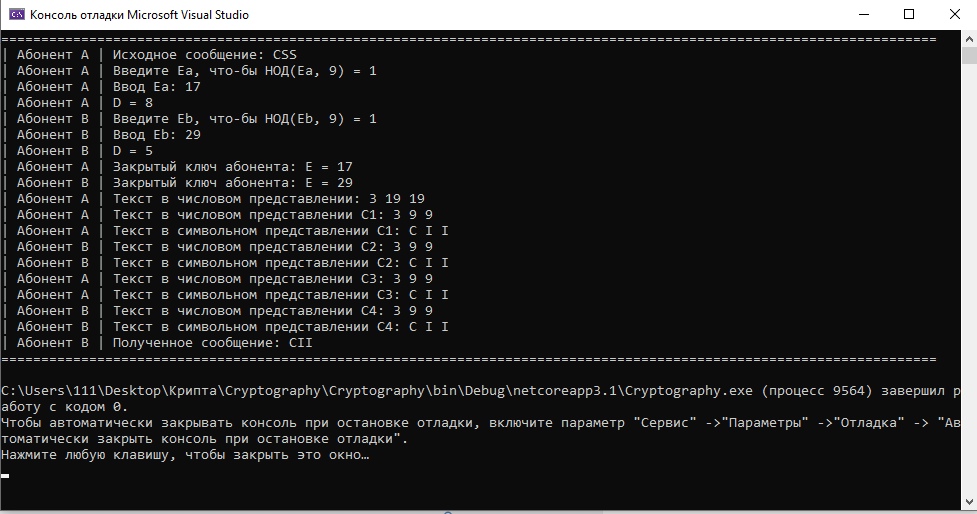


Рисунок 1. Реализация протокола Шамира

**Листинг:**

using System;

using System.Collections.Generic;

using System.Numerics;

using System.Text;

// lab 2

namespace Cryptography.Ciphers

{

static class Shamir\_protocol

{

static private string Alphabet = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";

static private long[] numericConversion;

static private string stringConversion;

static public void extendedGCD(long a, long b, out long x, out long y, out long d)//Расширенный алгоритм Евклида

{

long q, r, x1, x2, y1, y2;

if (b == 0)

{

d = a;

x = 1;

y = 0;

return;

}

x2 = 1; y2 = 0;

x1 = 0; y1 = 1;

while (b > 0)

{

q = a / b; r = a - q \* b;

x = x2 - q \* x1; y = y2 - q \* y1;

a = b; b = r;

x2 = x1; x1 = x; y2 = y1; y1 = y;

}

d = a; x = x2; y = y2;

}

static public long GCD(long A, long B)//Поиск НОД | Алгоритм Евклида

{

while (B != 0)

(A, B) = (B, A % B);

return A;

}

static public long[] TextToNumberEncrypt(string Text)//Текст в набор чисел

{

numericConversion = new long[Text.Length];

for (int i = 0; i < Text.Length; i++)//Числовой вид сообщения

{

for (int j = 0; j < Alphabet.Length; j++)

{

if (Text[i] == Alphabet[j])

{

numericConversion[i] = j + 1; break;

}

}

}

return numericConversion;

}

static public string NumberEncryptToText(long[] NumberEncrypt)//Набор чисел в текст

{

stringConversion = "";

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

{

if (NumberEncrypt[i] == 0) stringConversion += " ";

else stringConversion += Alphabet[(int)(NumberEncrypt[i] - 1)];

}

return stringConversion;

}

static public string NumberEncryptToTextWithSpace(long[] NumberEncrypt)//Набор чисел в текст

{

stringConversion = "";

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

{

if (NumberEncrypt[i] == 0) stringConversion += " ";

else stringConversion += Alphabet[(int)(NumberEncrypt[i] - 1)] + " ";

}

return stringConversion;

}

public static BigInteger reSquaring(BigInteger m, BigInteger e, BigInteger n)//c=m^e(mod n)

{

BigInteger E = e;

long i;

for (i = 1; E != 1; i++)//Проверка количества элементов

E = E / 2;

BigInteger[] bynaryN = new BigInteger[i];

E = e;

for (long j = 0; j < i; j++)//Степень в бинарном представлении

{

bynaryN[j] = E % 2;

E = E / 2;

}

BigInteger b = m % n;

for (long j = 1; j < i; j++)//Возведение в степень

{

b = BigInteger.ModPow(b, 2, n);

if (bynaryN[j] == 1)

m = (m \* b) % n;

}

return m;

}

public static string NumberEncryptToString(long[] NE)

{

string str = "";

foreach (var item in NE)

{

str += item + " ";

}

return str;

}

}

static class A\_Abonent

{

static private long[] NumberEncrypt;

static private long[] FirstNumberEncrypt;

static private long[] SecondNumberEncrypt;

static private long E;

static private long P;

static private long D;

static public long[] GetNumberEncrypt()

{

return NumberEncrypt;

}

static public bool isNormalE(long e)

{

if (Shamir\_protocol.GCD(e, P - 1) == 1)

{

E = e;

return true;

}

else return false;

}

static public void SetP(long p)

{

P = p;

}

static public void SetE()

{

do

{

A\_Abonent.PrintLog("Ввод Еa: ", false);

E = Convert.ToInt64(Console.ReadLine());

}

while (!A\_Abonent.isNormalE(E));

//A\_Abonent.PrintLog($"E = {E} удовлетворяет условию НОД(Е, Р-1)!", true);

}

static public void SetD()

{

Shamir\_protocol.extendedGCD(E, P - 1, out long x, out long y, out long d);

D = x + (P - 1);

PrintLog($"D = {D}", true);

}

static public void PrintE()

{

PrintLog($"Закрытый ключ абонента: E = {E}", true);

}

static public void PrintNumberEncrypt()

{

PrintLog($"Текст в числовом представлении: ", false);

foreach (var item in NumberEncrypt)

Console.Write($"{item} ");

Console.WriteLine();

}

static public void TextToNumberEncrypt(string Text)

{

NumberEncrypt = Shamir\_protocol.TextToNumberEncrypt(Text);

}

static public void PrintLog(string str, bool endl)

{

if (endl)

Console.WriteLine("| Абонент A | " + str);

else

Console.Write("| Абонент A | " + str);

}

static public long[] FirstStepE(long[] C)

{

FirstNumberEncrypt = new long[C.Length];

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

{

FirstNumberEncrypt[i] = (long)(Shamir\_protocol.reSquaring(C[i], E, P));//c=m^e(mod n)

}

return FirstNumberEncrypt;

}

static public long[] SecondStepD(long[] C)

{

SecondNumberEncrypt = new long[C.Length];

for (long i = 0; i < C.Length; i++)

{

SecondNumberEncrypt[i] = (long)(Shamir\_protocol.reSquaring(C[i], D, P));//c=m^e(mod n)

}

return SecondNumberEncrypt;

}

}

static class B\_Abonent

{

static private long[] NumberEncrypt;

static private long[] FirstNumberEncrypt;

static private long[] SecondNumberEncrypt;

static private long E;

static private long P;

static private long D;

static public long[] GetNumberEncrypt()

{

return NumberEncrypt;

}

static public bool isNormalE(long e)

{

if (Shamir\_protocol.GCD(e, P - 1) == 1)

{

E = e;

return true;

}

else return false;

}

static public void SetP(long p)

{

P = p;

}

static public void SetE()

{

do

{

B\_Abonent.PrintLog("Ввод Еb: ", false);

E = Convert.ToInt64(Console.ReadLine());

}

while (!B\_Abonent.isNormalE(E));

//B\_Abonent.PrintLog($"E = {E} удовлетворяет условию НОД(Е, Р-1)!",true);

}

static public void SetD()

{

Shamir\_protocol.extendedGCD(E, P - 1, out long x, out long y, out long d);

D = x + (P - 1);

PrintLog($"D = {D}", true);

}

static public void PrintE()

{

PrintLog($"Закрытый ключ абонента: E = {E}", true);

}

static public void PrintNumberEncrypt()

{

PrintLog($"Текст в числовом представлении: ", false);

foreach (var item in NumberEncrypt)

Console.Write($"{item} ");

Console.WriteLine();

}

static public void TextToNumberEncrypt(string Text)

{

NumberEncrypt = Shamir\_protocol.TextToNumberEncrypt(Text);

}

static public void PrintLog(string str, bool endl)

{

if (endl)

Console.WriteLine("| Абонент B | " + str);

else

Console.Write("| Абонент B | " + str);

}

static public long[] FirstStepE(long[] C)

{

FirstNumberEncrypt = new long[C.Length];

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

{

FirstNumberEncrypt[i] = (long)(Shamir\_protocol.reSquaring(C[i], E, P));//c=m^e(mod n)

}

return FirstNumberEncrypt;

}

static public long[] SecondStepD(long[] C)

{

SecondNumberEncrypt = new long[C.Length];

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

{

SecondNumberEncrypt[i] = (long)(Shamir\_protocol.reSquaring(C[i], D, P));//c=m^e(mod n)

}

return SecondNumberEncrypt;

}

}

}

**Вывод:** освоил и программно реализовал работы криптосистем, приобрел навыки их использования на практике для решения задач шифрования, дешифрования данных и обмена ключами.

## **Лабораторная работа №3**

**Тема:** Криптографическая система RSA.

**Выполнил:** Леднев Евгений, ФИТУ 2-5.

**Задание:**Используя полученные данные, выполнить процедуры шифрования шифрования и дешифрования в рамках RSA.

**Протокол RSA**

***Программа работы.***

*Этап 1. Генерация ключей*

*Этап 2. Преобразование открытого текста в числовой эквивалент*

*Этап 3. Шифрование*

*Этап 4. Преобразование шифртекста в символьное представление*

*Этап 5. Преобразование символьного представления шифртекста в числовой эквивалент*

*Этап 6. Дешифрование*

*Этап 7. Восстановление символьного представления открытого текста.*

**Вариант 8:** P = 55799; Q = 65141; Text = GEPARD.

***Алгоритм:***

Сообщение хочет получить абонент B от абонента A.

Процесс инициирует абонент B!

**У абонента B:**

* Введем p = 55799;
* Введем q = 65141;
* Вычислим n из выражения по формуле n = p \* q = 3651877153;
* Вычислим Ф(n) по формуле Ф(n) = (p-1)\*(q-1). Ф(n) = 3651755908;
* Выбираем ключ e, соответствующий условию алгоритма. e = 17;
* Сгенерируем ключ d, соответствующий e^-1 (mod Ф(n)). d = 3436946737;

**У абонента A:**

* Формируем алфавит;
* Преобразовываем текст в числовое представление: 6 4 15 0 17 3
* Выполняем шифрование по формуле m^e(mod n): 208840581 2572360572 2300890357 0 2233374000 129140163
* Приведение к символьному представлению: AAARPADUB AAAIINCLDC AAAHLRAXYZ A AAAHFZHNSA AAAKWPNTJ Передаем зашифрованное сообщение Абоненту B;

**У абонента B:**

* Преобразование шифра в числовой эквивалент: 08840581 2572360572 2300890357 0 2233374000 129140163 Расшифровка сообщения: 6 4 15 0 17 3
* Расшифрованное текстовое сообщение: GEPARD

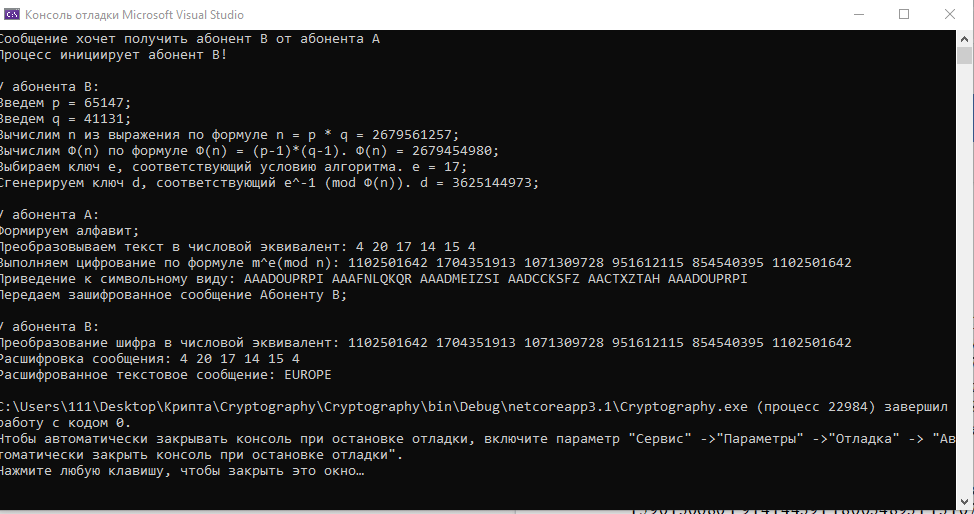


Рисунок. Программная реализация RSA.

**Листинг:**

using System;

using System.Collections.Generic;

using System.Numerics;

using System.Text;

namespace Cryptography.Ciphers

{

public class RSA

{

private long p;// p простое число

private long q;// q простое число

private long e;// e открытая экспонента

private long d;// d секретная экспонента

private long n;// n

private long Fi;// Ф(n)

private string alphabet = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";

//private string alphabet = " ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyzАБВГДЕЁЖЗИЙКЛМНОПРСТУФХЦЧШЩЪЫЬЭЮЯабвгдеёжзийклмнопрстуфхцчшщъыьэюя0123456789.,?!\*/+-=\_()%;:#";//Набор сиволов

private int[] numericConversion;//Перевод текста в числовую запись

private long[] encrypt;//Числовой шифр

private string[] alphabetEncrypt;//Зашифрованное сообщение в символьном представлении

private long[] decrypt;//Числовой шифр до расшифровки

private long[] alphabetDecrypt;//Дешифрованное сообщение в численном представлении

private string OutDecrypt;//Дешифрованное сообщение

public void getPublicKey(out long e, out long n)//Получить открытый ключ

{

e = this.e;

n = this.n;

}

public void getPrivateKey(out long d, out long n)//Получить секретный ключ

{

d = this.d;

n = this.n;

}

public RSA(long e, long p, long q)//Генерация ключей

{

try

{

this.p = (IsPrimeNumber(p)) ? p : throw new Exception("Error: p должно быть простым");

this.q = (IsPrimeNumber(q)) ? q : throw new Exception("Error: q должно быть простым");

this.n = p \* q;// n модуль

this.Fi = (p - 1) \* (q - 1);// Фи(n)

this.e = ((1 < e && e < Fi) && (GCD(e, Fi) == 1)) ? e : throw new Exception("Error: Должно выполняться 2 условия:\n 1 - (1 < e < Ф(n))\n 2 - e и Ф(n) - взаимно простые числа!");

extendedGCD(e, Fi, out long x, out long y, out long d);

this.d = x + Fi;

}

catch (Exception ex)

{

Console.WriteLine(ex.Message);

}

}

private static bool IsPrimeNumber(long n)//Проверка на простоту

{

bool result = true;

if (n > 1)

{

for (var i = 2u; i < n; i++)

{

if (n % i == 0)

{

result = false;

break;

}

}

}

else

{

result = false;

}

return result;

}

private static long GCD(long A, long B)//Поиск НОД | Алгоритм Евклида

{

while (B != 0)

(A, B) = (B, A % B);

return A;

}

private static void extendedGCD(long a, long b, out long x, out long y, out long d)//Расширенный алгоритм Евклида

{

long q, r, x1, x2, y1, y2;

if (b == 0)

{

d = a;

x = 1;

y = 0;

return;

}

x2 = 1; y2 = 0;

x1 = 0; y1 = 1;

while (b > 0)

{

q = a / b; r = a - q \* b;

x = x2 - q \* x1; y = y2 - q \* y1;

a = b; b = r;

x2 = x1; x1 = x; y2 = y1; y1 = y;

}

d = a; x = x2; y = y2;

}

public void CipherLog()//Лог работы программы

{

//(e,n) --> Открытый ключ

//(d,n) --> Закрытый ключ

Console.WriteLine($"Сообщение хочет получить абонент B от абонента A");

Console.WriteLine($"Процесс инициирует абонент B!\n");

Console.WriteLine($"У абонента B:");

Console.WriteLine($"Введем p = {p};");

Console.WriteLine($"Введем q = {q};");

Console.WriteLine($"Вычислим n из выражения по формуле n = p \* q = {n};");

Console.WriteLine($"Вычислим Ф(n) по формуле Ф(n) = (p-1)\*(q-1). Ф(n) = {Fi};");

Console.WriteLine($"Выбираем ключ e, соответствующий условию алгоритма. e = {e};");

Console.WriteLine($"Сгенерируем ключ d, соответствующий e^-1 (mod Ф(n)). d = {d};\n");

Console.WriteLine($"У абонента A:");

Console.WriteLine($"Формируем алфавит;");

Console.Write($"Преобразовываем текст в числовой эквивалент: ");

foreach (var item in numericConversion) Console.Write(item + " ");

Console.Write($"\nВыполняем цифрование по формуле m^e(mod n): ");

foreach (var item in encrypt) Console.Write(item + " ");

Console.Write($"\nПриведение к символьному виду: ");

foreach (var item in alphabetEncrypt) Console.Write(item + " ");

Console.WriteLine("\nПередаем зашифрованное сообщение Абоненту B;");

Console.WriteLine("\nУ абонента B:");

Console.Write("Преобразование шифра в числовой эквивалент: ");

foreach (var item in alphabetDecrypt) Console.Write(item + " ");

Console.Write("\nРасшифровка сообщения: ");

foreach (var item in decrypt) Console.Write(item + " ");

Console.Write("\nРасшифрованное текстовое сообщение: ");

foreach (var item in OutDecrypt) Console.Write(item);

Console.WriteLine();

}

public static BigInteger reSquaring(BigInteger m, BigInteger e, BigInteger n)//c=m^e(mod n)

{

BigInteger E = e;

int i;

for (i = 1; E != 1; i++)//Проверка количества элементов

E = E / 2;

BigInteger[] bynaryN = new BigInteger[i];

E = e;

for (int j = 0; j < i; j++)//Степень в бинарном представлении

{

bynaryN[j] = E % 2;

E = E / 2;

}

BigInteger b = m % n;

for (int j = 1; j < i; j++)//Возведение в степень

{

b = BigInteger.ModPow(b, 2, n);

if (bynaryN[j] == 1)

m = (m \* b) % n;

}

return m;

}

//=========================================================================================================== Шифрование

private string symbolicRepresentationToString(long C)//Приведение к символьному типу

{

string returnStr = "";

int numberLength = C.ToString().Length;//Количество цифр в числе

long index;

long numberC = C;

for (int i = 1; i <= numberLength; i++)

{

index = (numberC / (long)Math.Pow(alphabet.Length, numberLength - i));

numberC -= (index \* (long)Math.Pow(alphabet.Length, numberLength - i));

returnStr += alphabet[(int)index];

}

return returnStr;

}

public string[] Encrypt(string m, long e, long n)//Зашифровать

{

numericConversion = new int[m.Length];

for (int i = 0; i < m.Length; i++)//Числовой вид сообщения

{

for (int j = 0; j < alphabet.Length; j++)

{

if (m[i] == alphabet[j])

{

numericConversion[i] = j; break;

}

}

}

encrypt = new long[numericConversion.Length];//Шифротекст

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

{

encrypt[i] = (long)reSquaring(numericConversion[i], e, n);

}

alphabetEncrypt = new string[encrypt.Length];//Шифротекст в символьном представлении

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

{

alphabetEncrypt[i] = symbolicRepresentationToString(encrypt[i]);

}

return alphabetEncrypt;

}

//=========================================================================================================== Дешифрование

private long antiSymbolicRepresentationToString(string Str)//Перевод символов в числовой эквивалент

{

long StrLength = Str.Length;//Количество букв

long C = 0;//В числовой эквивалент

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

{

for (int j = 0; j < alphabet.Length; j++)

{

if (Str[i] == alphabet[j])

{

C += j \* (long)Math.Pow(alphabet.Length, StrLength - i - 1); break;

}

}

}

return C;//Вернуть число

}

public string Decrypt(string[] m, long d, long n)//Дешифровать

{

alphabetDecrypt = new long[m.Length];//Шифротекст в символьном представлении

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

{

alphabetDecrypt[i] = antiSymbolicRepresentationToString(m[i]);

}

decrypt = new long[alphabetDecrypt.Length];//Шифротекст в число

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

{

decrypt[i] = (long)reSquaring(alphabetDecrypt[i], d, n);

}

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

{

OutDecrypt += alphabet[(int)decrypt[i]].ToString();

}

return OutDecrypt;

}

}

}

## **Лабораторная работа №4**

**Тема:** криптографические прикладные программные интерфейсы.

**Выполнил:** Леднев Евгений, ФИТУ 2-5.

**Цель работы:** освоение и программная реализация работы асимметричных криптосистем, приобретение навыков их использования на практике для решения задач подписания сообщений ЭЦП и проверки подлинности подписей.

**Задание:** используя полученные данные, выполнить программную реализацию в рамках схемы Эль-Гамаль для подписания сообщения М и проверки подписи.

**Программа работы.**

Этап 1. Генерация открытого и закрытого ключей.

Этап 2. Преобразование сообщения в числовой эквивалент.

Этап 3. Подписание сообщения (генерация цифровой подписи).

Этап 4. Проверка цифровой подписи.

***Вариант 10:*** *P = 35149; m=DNS.*

Алгоритм протокола:

1. генерация ключей.

Выберем значения g, x из условия: g <P, x <P. Пусть g = 17, x = 13.

Вычислим значение у: 𝑦= 𝑔𝑥 𝑚𝑜𝑑 𝑃= 57 𝑚𝑜𝑑 23=17.

(вычисление производится по методу повторного возведения в квадрат).

Открытый ключ: (y,g,P) = (13855, 17, 35149); закрытый ключ х = 13.

Отправляем открытый ключ абоненту В.

2. преобразование сообщения в числовой эквивалент.

В числовом представлении m = “DNS” = 4 14 19.

3. генерация ЭЦП.

Из условия НОД (k, P – 1) = 1 выберем значение k. Пусть k = 5.

Вычислим 1 часть цифровой подписи: 𝑎= 𝑔𝑘 𝑚𝑜𝑑 𝑃= 55 𝑚𝑜𝑑 23=20.

Найдем значение 2 части цифровой подписи: 𝑏=(𝑚−𝑥∗𝑎)𝑘−1 𝑚𝑜𝑑 (𝑃−1)=(7−7∗20)∗ 5−1 𝑚𝑜𝑑 22=13.

Цифровая подпись: (a, b) = (20, 13).

Отправляем ЭЦП абоненту В.

4. проверка цифровой подписи (у абонента В).

В выражении 𝑦𝑎∗ 𝑎𝑏 𝑚𝑜𝑑 𝑃= 𝑔𝑚 𝑚𝑜𝑑 𝑃 вычислим раздельно левую и правую части. (𝑦𝑎∗ 𝑎𝑏) 𝑚𝑜𝑑 𝑃= (1720∗ 2013) 𝑚𝑜𝑑 23=17.

Для вычисления используем метод цепочек.

𝑔𝑚 𝑚𝑜𝑑 𝑃= 57 𝑚𝑜𝑑 23=17.

**Программная реализация**

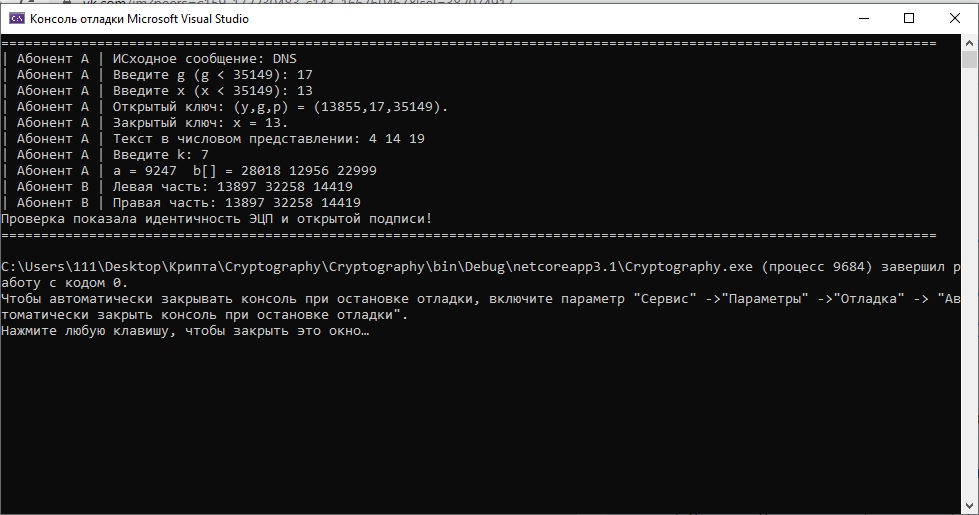


Рисунок 1. Программная реализация в рамках схемы Эль-Гамаль

**Листинг:**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Numerics;

using System.Text;

using System.Threading.Tasks;

namespace Cryptography.Ciphers

{

static class ELGAmal

{

static private string Alphabet = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";

static private long[] numericConversion;

static public bool IsPrimeNumber(long n)//Проверка на простоту

{

bool result = true;

if (n > 1)

{

for (var i = 2u; i < n; i++)

{

if (n % i == 0)

{

result = false;

break;

}

}

}

else

{

result = false;

}

return result;

}

static public BigInteger reSquaring(BigInteger m, BigInteger e, BigInteger n)//c=m^e(mod n)

{

BigInteger E = e;

long i;

for (i = 1; E != 1; i++)//Проверка количества элементов

E = E / 2;

BigInteger[] bynaryN = new BigInteger[i];

E = e;

for (long j = 0; j < i; j++)//Степень в бинарном представлении

{

bynaryN[j] = E % 2;

E = E / 2;

}

BigInteger b = m % n;

for (long j = 1; j < i; j++)//Возведение в степень

{

b = BigInteger.ModPow(b, 2, n);

if (bynaryN[j] == 1)

m = (m \* b) % n;

}

return m;

}

static public long Mod(long a, long b)

{

if (a < 0)

{

a \*= -1;

a = a % b;

a = b - a;

a = a % b;

return a;

}

else

{

return a % b;

}

}

static public long[] TextToNumberEncrypt(string Text)//Текст в набор чисел

{

numericConversion = new long[Text.Length];

for (int i = 0; i < Text.Length; i++)//Числовой вид сообщения

{

for (int j = 0; j < Alphabet.Length; j++)

{

if (Text[i] == Alphabet[j])

{

numericConversion[i] = j + 1; break;

}

}

}

return numericConversion;

}

static public void PrintNumberEncrypt(long[] NumberEncrypt, bool Abonent)

{

if (Abonent)

A\_Abonent.PrintLog($"Текст в числовом представлении: ", false);

else

B\_Abonent.PrintLog($"Текст в числовом представлении: ", false);

foreach (var item in NumberEncrypt)

Console.Write($"{item} ");

Console.WriteLine();

}

static public long GCD(long A, long B)//Поиск НОД | Алгоритм Евклида

{

while (B != 0)

(A, B) = (B, A % B);

return A;

}

static public void extendedGCD(long a, long b, out long x, out long y, out long d)//Расширенный алгоритм Евклида

{

long q, r, x1, x2, y1, y2;

if (b == 0)

{

d = a;

x = 1;

y = 0;

return;

}

x2 = 1; y2 = 0;

x1 = 0; y1 = 1;

while (b > 0)

{

q = a / b; r = a - q \* b;

x = x2 - q \* x1; y = y2 - q \* y1;

a = b; b = r;

x2 = x1; x1 = x; y2 = y1; y1 = y;

}

d = a; x = x2; y = y2;

}

}

static class Abonent1

{

static private long g;

static private long p;

static private long x;

static private long y;

static private long k;

static public long a;

static public long[] b;

static public long KeyAbonentY;

static public long KeyAbonentG;

static public long KeyAbonentP;

static public void EnterG()

{

do

{

PrintLog($"Введите g (g < {p}): ", false);

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

}

while (!(ELGAmal.IsPrimeNumber(g) && (g < p)));

}

static public void EnterX()

{

do

{

PrintLog($"Введите x (x < {p}): ", false);

x = Convert.ToInt32(Console.ReadLine());

}

while (!(ELGAmal.IsPrimeNumber(x) && (x < p)));

}

static public void SetP(int P)

{

p = P;

}

static public void CalculateY()

{

y = (long)ELGAmal.reSquaring(g, x, p);

}

static public void PrintOpenKeys()

{

PrintLog($"Открытый ключ: (y,g,p) = ({y},{g},{p}).", true);

}

static public void PrintPrivateKeys()

{

PrintLog($"Закрытый ключ: х = {x}.", true);

}

static public void SendOpenKeys()

{

Abonent2.KeyAbonentY = y;

Abonent2.KeyAbonentG = g;

Abonent2.KeyAbonentP = p;

}

static public void PrintLog(string str, bool endl)

{

if (endl)

Console.WriteLine("| Абонент A | " + str);

else

Console.Write("| Абонент A | " + str);

}

static public bool isNormalE(long e)

{

return (ELGAmal.GCD(k, p - 1) == 1) ? true : false;

}

static public void EnterK()

{

do

{

A\_Abonent.PrintLog("Введите k: ", false);

k = Convert.ToInt64(Console.ReadLine());

}

while (!A\_Abonent.isNormalE(k));

}

static public void FindFirstPart()

{

a = (long)ELGAmal.reSquaring(g, k, p);

PrintLog($"а = {a}", false);

}

static public void PrintB(long[] b)

{

Console.Write("\tb[] = ");

foreach (var item in b)

{

Console.Write(item + " ");

}

Console.WriteLine();

}

static public void FindSecondPart()

{

b = new long[Program.NumberEncrypt.Length];

ELGAmal.extendedGCD(k, p - 1, out long t1, out long t2, out long d);

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

{

long firstPart = t1 + p - 1;

long secondPart = ELGAmal.Mod(Program.NumberEncrypt[i] + 1 - x \* a, p - 1);

//Console.WriteLine(firstPart + " " + secondPart);

b[i] = ELGAmal.Mod(firstPart \* secondPart, p - 1);

//Console.WriteLine(b[i]);

}

PrintB(b);

}

static public void SignatureGeneration()

{

EnterK();

FindFirstPart();

FindSecondPart();

}

}

static class Abonent2

{

static private long g;

static private long p;

static private long x;

static private long y;

static private long k;

static public long KeyAbonentY;

static public long KeyAbonentG;

static public long KeyAbonentP;

static public void EnterG()

{

do

{

PrintLog($"Введите g (g < {p}): ", false);

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

}

while (!(ELGAmal.IsPrimeNumber(g) && (g < p)));

}

static public void EnterX()

{

do

{

PrintLog($"Введите x (x < {p}): ", false);

x = Convert.ToInt32(Console.ReadLine());

}

while (!(ELGAmal.IsPrimeNumber(x) && (x < p)));

}

static public void SetP(int P)

{

p = P;

}

static public void CalculateY()

{

y = (long)ELGAmal.reSquaring(g, x, p);

}

static public void PrintOpenKeys()

{

PrintLog($"Открытый ключ: (y,g,p) = ({y},{g},{p}).", true);

}

static public void PrintPrivateKeys()

{

PrintLog($"Закрытый ключ: х = {x}.", true);

}

static public void SendOpenKeys()

{

Abonent1.KeyAbonentY = y;

Abonent1.KeyAbonentG = g;

Abonent1.KeyAbonentP = p;

}

static public void PrintLog(string str, bool endl)

{

if (endl)

Console.WriteLine("| Абонент B | " + str);

else

Console.Write("| Абонент B | " + str);

}

static public bool isNormalE(long e)

{

return (ELGAmal.GCD(k, p - 1) == 1) ? true : false;

}

static public void SignatureGeneration()

{

do

{

A\_Abonent.PrintLog("Введите k: ", false);

k = Convert.ToInt64(Console.ReadLine());

}

while (!A\_Abonent.isNormalE(k));

}

static public void verificationOfDigitalSignature(long a, long[] b, long[] NumberEncrypt)

{

long[] LeftPart = new long[b.Length];//Левая часть

long[] RightPart = new long[b.Length];//Правая часть

PrintLog("Левая часть: ", false);

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

{

RightPart[i] = (long)ELGAmal.reSquaring(KeyAbonentG, NumberEncrypt[i], KeyAbonentP);

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

}

Console.WriteLine();

PrintLog("Правая часть: ", false);

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

{

long temp = (long)(ELGAmal.reSquaring(KeyAbonentY, a, KeyAbonentP) \* ELGAmal.reSquaring(a, b[i], KeyAbonentP));

LeftPart[i] = ELGAmal.Mod(temp, KeyAbonentP);

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

}

Console.WriteLine("\nПроверка показала идентичность ЭЦП и открытой подписи!");

}

}

}

**Заключение**

В результате выполнения лабораторной работы №1 были освоены и произведены программные реализации следующих симметричных (логических – замены и перестановки) методов шифрования: «Метод Цезаря с ключевым словом», «Метод Цезаря с аффинной формулой расчета заменяющего символа», метод «Двойной квадрат Уитстона», метод двойной перестановки с ключевым словом и цифровым ключом.

В результате выполнения лабораторной работы №2 была освоена и произведена программная реализация в рамках схемы RSA для шифрования и дешифрования сообщений.

В результате выполнения лабораторной работы №3 была освоена и произведена программная реализация в рамках схемы Эль-Гамаль для шифрования и дешифрования сообщений.

В результате выполнения лабораторной работы №4 была освоена и произведена программная реализация криптографического протокола Шамира для передачи сообщения М от абонента А абоненту В.

Полученные в ходе выполнения лабораторных работ программные продукты работают корректно и выполняют шифрование и дешифрование сообщений верно.

**Список использованных источников**

1. Зайцев Р.Г., Спиридонова И.А.. Защита информации и основы прикладной криптографии [Электронный ресурс]:учебное пособие. - Новочеркасск: ЮРГПУ (НПИ), 2016. - 80 с. – Режим доступа: http://lib.npi-tu.ru
2. Аграновский А. В., Хади Р. А.. Практическая криптография: алгоритмы и их программирование [Электронный ресурс]:. - Москва: СОЛОН-ПРЕСС, 2009. - 256 с. – Режим доступа: http://biblioclub.ru
3. Шаньгин В. Ф.. Защита информации в компьютерных системах и сетях [Электронный ресурс]:. - Москва: ДМК Пресс, 2012. - 592 с. – Режим доступа: http://e.lanbook.com
4. Голиков А. М.. Защита информации в инфокоммуникационных системах и сетях [Электронный ресурс]:учебное пособие. - Томск: Томский государственный университет систем управления и радиоэлектроники, 2015. - 284 с. – Режим доступа: <http://biblioclub.ru>

Спиридонова И.А.. Основы прикладной криптографии [Электронный ресурс]: методические указания к практическим занятиям, лабораторным работам и самостоятельной работе студента. - Новочеркасск: ЮРГПУ (НПИ), 2016. - 27 с. – Режим доступа: http://lib.npi-tu.ru