

МИНОБРНАУКИ РОССИИ

федеральное государственное бюджетное образовательное учреждение высшего образования

«Санкт-Петербургский государственный технологический институт (технический университет)»

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| УГНС | | 09.00.00 | Информатика и вычислительная техника | | |
| Направление подготовки | | 09.03.01 | Информатика и вычислительная техника | | |
| Направленность (профиль) | |  | Системы автоматизированного проектирования | | |
| Форма обучения | |  | очная | | |
|  | |  |  | | |
| Факультет | |  | Информационных технологий и управления | | |
| Кафедра | |  | Систем автоматизированного проектирования и управления | | |
| Учебная дисциплина | |  | Программирование | | |
| Курс | II | | | Группа | 404 |

Отчёт по лабораторной № 2

Вариант № 2

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Исполнитель: |  |  |  |  |
| обучающийся группы 404 |  |  |  | Д. К. Азаров |
|  |  | (дата, подпись) |  |  |
|  |  |  |  |  |
| Проверили: |  |  |  | И. Г. Корниенко |
|  |  | (дата, подпись) |  | А. К. Федин |

### 1 Постановка задачи

Создать интерфейс ICipher, который определяет методы поддержки шифрования строк. В интерфейсе объявляются два метода Encode() и Decode(), которые используются для шифрования и дешифрования строк, соответственно. Реализовать классы шифрования ГОСТ 28147-89, Гаммирование.

### 2 Исходные данные

Строка и ключ.

### 3 Особые ситуации

Необходимо рассмотреть следующую особую ситуацию в шифровании по ГОСТ 28147-89:

1. Длина ключа должна быть 256 бит;
2. Кол-во бит в строке должно быть кратно 64.

Необходимо рассмотреть следующую особую ситуацию в шифровании по ГОСТ 28147-89:

1. Длина ключа не должна превышать размер текста.

### 4 Математические методы и алгоритмы решения задач

#### Шифрование по ГОСТ 28147-89:

#### **Зашифровывание:**

1. Открытые данные разбиваются на блоки по 64 бита.
2. Далее производится ввод первого блока в накопители N1 и N2. При этом биты открытой информации вводятся следующим образом: 1-й бит открытой информации — в 1-й разряд накопителя N1, ..., 32-й — в 32-й разряд накопителя N1, 33-й — в 1-й разряд накопителя N2 и так далее, пока 64-й бит открытой информации не будет введен в 32-й разряд накопителя N2.
3. В КЗУ вводится ключ длиной 256 бит способом, рассмотренным в статье.

#### **Производится зашифрование открытых данных в режиме простой замены (в 32 цикла):**

* 1. В первом цикле содержимое регистра N1 суммируется с заполнением X0 из КЗУ по модулю 232 в сумматоре СМ1.
  2. В блоке подстановки K производится замена 32 бит информации, поступившей из сумматора СМ1.
  3. В регистре сдвига R осуществляется циклический сдвиг на 11 в сторону старшего разряда.
  4. Информация с регистра сдвига R и накопителя N2 суммируется по модулю 2 в сумматоре СМ2.
  5. Старое заполнение накопителя N1 переписывается в накопитель N2.
  6. Результат с выхода сумматора СМ2 переписывается в накопитель N1.
  7. Первый цикл заканчивается.

1. Последующие циклы аналогичны первому, с тем лишь отличием, что во 2-м цикле вводится ключ X1, в 8-м — X7, в 9-м — X0 и так далее в том же порядке до 24 цикла. С 25 по 32 цикл ключ вводится в обратном порядке: X7 — в 25-м, X0 — в 32-м.
2. После 32-го цикла в N1 информация сохраняется, а вот результат с выхода сумматора СМ2 переписывается в N2.
3. Заполнение N1 и N2 и есть первый блок зашифрованных данных.
4. Следующие блоки зашифровываются аналогично.

#### **Расшифровывание:**

Расшифровывание осуществляется по тому же алгоритму, что и зашифровывание, только на вход накопителей N1 и N2 поступают разбитые на блоки по 64 бита зашифрованные данные. Важным отличием является еще и то, что в прямом порядке (с X0 по X7) ключ вводится только в первых 8 циклах РПЗ, в остальных — в обратном (с X7 по X0).А так, после прохождения 32 циклов в накопителях N1 и N2 содержатся блоки открытых данных.

Шифрование Гаммированием:

***Шифрование:***

1. Заполняется ключ до размера текста циклическим копированием введенного ключа;
2. Каждый символ текста складывается по модулю 2 с соответствующим символом ключа.

#### **Расшифровывание:**

Аналогично с шифрованием, но на вход поступает зашифрованный текст.

### 5 Форматы представления данных

Таблица 1 - Классы, используемые в программе

|  |  |  |  |
| --- | --- | --- | --- |
| **Название** | **Имя переменной** | **Тип** | **Описание** |
| Gost28147\_89 | - | - | Реализует шифрование по ГОСТ 28147-89 |
| replacementBlock | Byte[,] | Таблица замены |
|  | **Метод** | **Тип** | **Описание** |
|  | Encode | Byte[] | Кодирование |
|  | Decode | String | Декодирование |
|  | Coding | Byte[] | Алгоритм шифрования |
|  | SumFromMod\_2\_Degree\_32 | UInt32 | Сложение по модулю 2 |
|  | SubstitutionBlock | UInt32 | Замена блоков |
|  | ShiftingRegister | UInt32 | Сдвиг влево на 11 |
|  | SumFromMod\_2 | UInt32 | Сумма по модулю 2 |
| GammaXoring | - | - | Реализует шифрование Гаммированием |
| **Метод** | **Тип** | **Описание** |
| Encode | Byte[] | Кодирование |
| Decode | String | Декодирование |
| Coding | Byte[] | Алгоритм шифрования |
|  | FillKeyBytes | Byte[] | Заполнение ключа |

### 6 Структура программы

Таблица 2 - Интерфейсы, используемые в программе

|  |  |
| --- | --- |
| **Имя** | **Описание** |
| ICipher | Кодировка – Декодировка строк |

### 7 Блок-схем алгоритмов программы

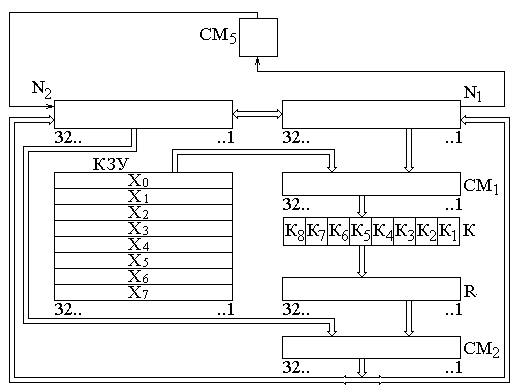


Рисунок 1 – Блок-схема алгоритма шифрования по ГОСТ 28147-89

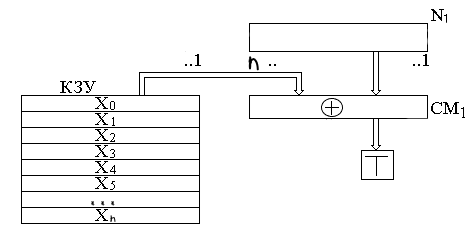


Рисунок 2 - Блок-схема алгоритма шифрования Гаммированием

### 8 Описание хода выполнения лабораторной работы

1. В ходе лабораторной работы было создано решение (Solution) в интегрированной среде разработки Microsoft Visual Studio C# WinForms 2019. В нем был создан проект.
2. В ходе лабораторной работы выявлена ошибка – пользователь мог ввести некорректную длину ключа, было принято решение добавлять ‘\0’ в конец переданной строки.
3. В ходе лабораторной работы выявлена ошибка – можно было передать некорректный размер зашифрованного текста. Было принято решение вызывать ошибку ‘throw’.
4. В ходе лабораторной работы выявлена ошибка – можно было передать некорректный размер ключа. Было принято решение вызывать ошибку ‘throw’.
5. В ходе лабораторной работы выявлена ошибка – использовалось сложение по модулю 2 int чисел, а не сложение по модулю 2 бит этих чисел. Было принято использовать оператор ‘^’ для сложения чисел.

### 9 Результаты работы программы

В результате вычислений программа выводит зашифрованный / расшифрованный текст.

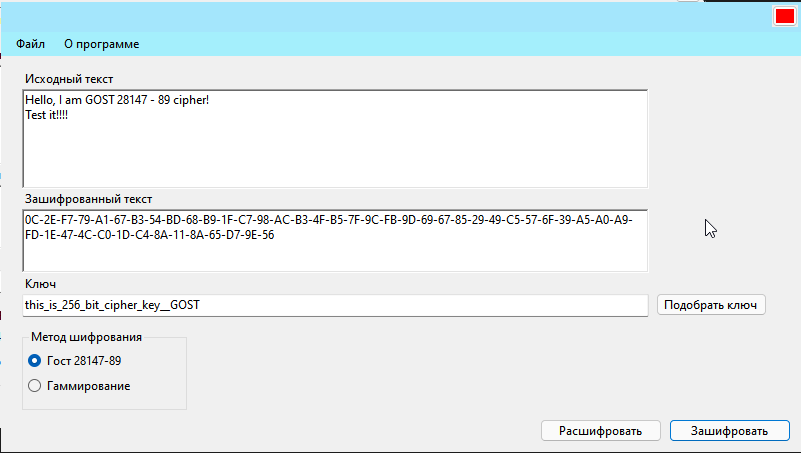


Рисунок 3 – Шифрование по ГОСТУ

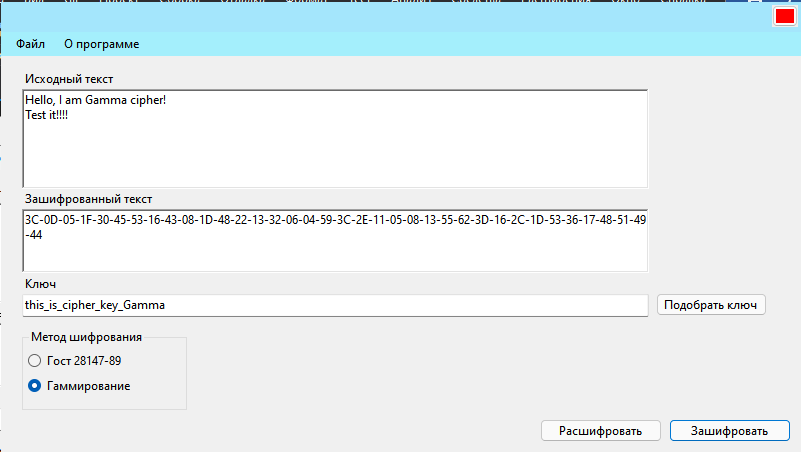


Рисунок 4 - Шифрование Гаммированием

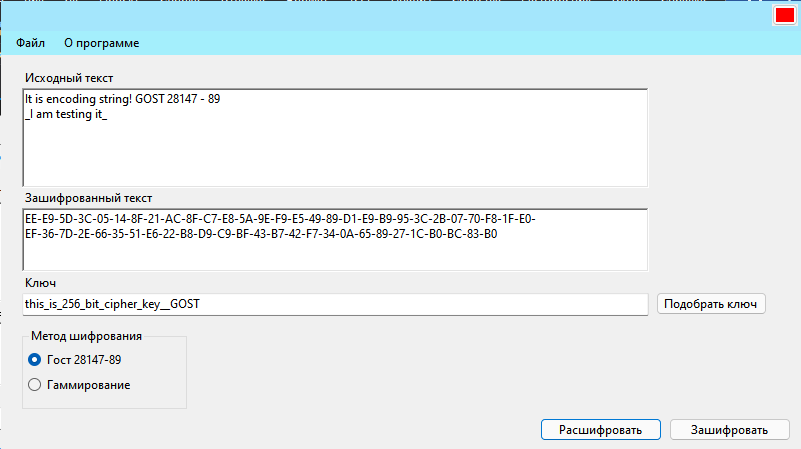


Рисунок 5 – Расшифровывание ГОСТ

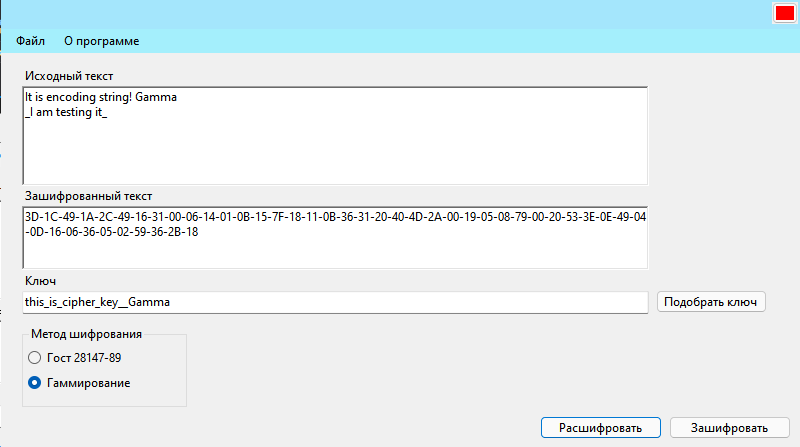


Рисунок 6 – Расшифровывание Гаммой

### 10 Исходный текст программы

[Начало ICipher.cs ---]

using System;

namespace Program

{

interface ICipher

{

public String Encode(String openData, String key);

public String Decode(String encryptedData, String key);

}

}

[--- Конец ICipher.cs]

[Начало Gost28147\_89.cs ---]

using System;

using System.Text;

namespace Program

{

public class Gost28147\_89 : ICipher

{

private readonly Byte[,] \_replacementBlock;

public Gost28147\_89()

{

\_replacementBlock = new Byte[8, 16] {

{0xF, 0xC, 0x2, 0xA, 0x6, 0x4, 0x5, 0x0, 0x7, 0x9, 0xE, 0xD, 0x1, 0xB, 0x8, 0x3},

{0xB, 0x6, 0x3, 0x4, 0xC, 0xF, 0xE, 0x2, 0x7, 0xD, 0x8, 0x0, 0x5, 0xA, 0x9, 0x1},

{0x1, 0xC, 0xB, 0x0, 0xF, 0xE, 0x6, 0x5, 0xA, 0xD, 0x4, 0x8, 0x9, 0x3, 0x7, 0x2},

{0x1, 0x5, 0xE, 0xC, 0xA, 0x7, 0x0, 0xD, 0x6, 0x2, 0xB, 0x4, 0x9, 0x3, 0xF, 0x8},

{0x0, 0xC, 0x8, 0x9, 0xD, 0x2, 0xA, 0xB, 0x7, 0x3, 0x6, 0x5, 0x4, 0xE, 0xF, 0x1},

{0x8, 0x0, 0xF, 0x3, 0x2, 0x5, 0xE, 0xB, 0x1, 0xA, 0x4, 0x7, 0xC, 0x9, 0xD, 0x6},

{0x3, 0x0, 0x6, 0xF, 0x1, 0xE, 0x9, 0x2, 0xD, 0x8, 0xC, 0x4, 0xB, 0xA, 0x5, 0x7},

{0x1, 0xA, 0x6, 0x8, 0xF, 0xB, 0x0, 0x4, 0xC, 0x3, 0x5, 0x9, 0x7, 0xD, 0x2, 0xE},

};

}

public String Encode(String openData, String key)

{

Byte[] keyBytes = Encoding.UTF8.GetBytes(key);

const int countBytesInKey = 32;

if (keyBytes.Length != countBytesInKey)

{

throw new KeyArgumentException("Пароль долже быть 256 битным");

}

const int countKeyStorage = 8;

UInt32[] keyStorage = new UInt32[countKeyStorage];

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

{

keyStorage[i] = BitConverter.ToUInt32(keyBytes, i \* 4);

}

Byte[] openDataByte = Encoding.UTF8.GetBytes(openData);

int differenceSize = openDataByte.Length - openData.Length;

if ((differenceSize > 0) || (openData.Length % 8 != 0))

{

int numberAappendCharacters = 8 - openDataByte.Length % 8;

Byte[] newOpenDataByte = new Byte[openDataByte.Length + numberAappendCharacters];

openDataByte.CopyTo(newOpenDataByte, 0);

Char[] nullChar = { '\0' };

Byte[] nullByte = Encoding.UTF8.GetBytes(nullChar);

while (numberAappendCharacters > 0)

{

newOpenDataByte[newOpenDataByte.Length - numberAappendCharacters] = nullByte[0];

numberAappendCharacters--;

}

openDataByte = newOpenDataByte;

}

Byte[] encryptedDataByte = Coding(openDataByte, keyStorage, true);

return BitConverter.ToString(encryptedDataByte, 0);

}

public String Decode(String encryptedData, String key)

{

Byte[] encryptedDataByte;

try

{

encryptedDataByte = SplitStringIntoBytes(encryptedData);

}

catch (Exception)

{

throw new EncryptedTextException("Введены некорректные данные");

}

if (encryptedDataByte.Length % 8 != 0)

{

throw new EncryptedTextException("Шифр должен быть кратен 8");

}

Byte[] keyBytes = Encoding.UTF8.GetBytes(key);

const int countBytesInKey = 32;

if (keyBytes.Length != countBytesInKey)

{

throw new KeyArgumentException("Пароль долже быть 256 битным");

}

const int countKeyStorage = 8;

UInt32[] keyStorage = new UInt32[countKeyStorage];

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

{

keyStorage[i] = BitConverter.ToUInt32(keyBytes, i \* 4);

}

Byte[] openDataByte = Coding(encryptedDataByte, keyStorage, false);

String openData = Encoding.UTF8.GetString(openDataByte);

return openData;

}

private Byte[] Coding(Byte[] data, UInt32[] keyStorage, bool isEncoding)

{

Byte[] encryptedDataByte = new Byte[data.Length];

const int sizeBlock = 8;

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

{

UInt32 leftBlock = BitConverter.ToUInt32(data, i);

UInt32 rightBlock = BitConverter.ToUInt32(data, i + sizeof(UInt32));

int countIteretion = 32;

UInt32 result = new UInt32();

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

{

result = SumFromMod\_2\_Degree\_32(leftBlock, keyStorage, j, isEncoding);

result = SubstitutionBlock(result);

result = ShiftingRegister(result);

result = SumFromMod\_2(result, rightBlock);

if (j != (countIteretion - 1))

{

rightBlock = leftBlock;

leftBlock = result;

}

}

rightBlock = result;

Byte[] leftBlockBytes = BitConverter.GetBytes(leftBlock);

Byte[] rightBlockBytes = BitConverter.GetBytes(rightBlock);

for (int j = 0; j < sizeBlock / 2; j++)

{

encryptedDataByte[i + j] = leftBlockBytes[j];

}

for (int j = 0; j < sizeBlock / 2; j++)

{

encryptedDataByte[i + j + sizeBlock / 2] = rightBlockBytes[j];

}

}

return encryptedDataByte;

}

private UInt32 SumFromMod\_2\_Degree\_32(UInt32 block, UInt32[] keyStorage, int numberIteration, bool isEncoding)

{

int countDirectCoding;

if (isEncoding == true)

{

countDirectCoding = 24;

}

else

{

countDirectCoding = 8;

}

if (numberIteration < countDirectCoding)

{

block = (block + keyStorage[numberIteration % keyStorage.Length]) % UInt32.MaxValue;

}

else

{

block = (block + keyStorage[(keyStorage.Length - 1) -

numberIteration % keyStorage.Length]) % UInt32.MaxValue;

}

return block;

}

private UInt32 SubstitutionBlock(UInt32 block)

{

Byte[] splitBlock = new Byte[8];

for (int i = 0; i < 8; i++)

{

splitBlock[i] = (Byte)block;

splitBlock[i] &= 0b0000\_1111;

block >>= 4;

}

for (int i = 7; i >= 0; i--)

{

block <<= 4;

block |= \_replacementBlock[i, splitBlock[i]];

}

return block;

}

private UInt32 ShiftingRegister(UInt32 block)

{

UInt32 safe = block;

safe >>= 21;

block <<= 11;

block |= safe;

return block;

}

private UInt32 SumFromMod\_2(UInt32 block, UInt32 rightBlock)

{

return block ^ rightBlock;

}

private Byte[] SplitStringIntoBytes(String data)

{

String[] splitData = new String[data.Length];

int byteIndex = 0;

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

{

if (data[i] == '-')

{

byteIndex++;

continue;

}

splitData[byteIndex] += data[i];

}

Byte[] dataByte = new Byte[++byteIndex];

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

{

dataByte[i] = byte.Parse(splitData[i], System.Globalization.NumberStyles.HexNumber);

}

return dataByte;

}

}

}

[--- Конец Gost28147\_89.cs]

[Начало GammaXoring.cs ---]

using System;

using System.Text;

namespace Program

{

public class GammaXoring : ICipher

{

public String Encode(String openData, String key)

{

if (key.Length == 0)

{

throw new KeyArgumentException("Пароль долже быть длинее 0 символов");

}

if (key.Length > openData.Length)

{

throw new KeyArgumentException("Пароль долже быть меньше длины введенного текста");

}

Byte[] openDataByte = Encoding.UTF8.GetBytes(openData);

Byte[] keyBytes = FillKeyBytes(openDataByte.Length, key);

Byte[] encryptedDataByte = Coding(openDataByte, keyBytes);

return BitConverter.ToString(encryptedDataByte, 0);

}

public String Decode(String encryptedData, String key)

{

if (key.Length == 0)

{

throw new KeyArgumentException("Пароль долже быть длинее 0");

}

Byte[] encryptedDataByte;

try

{

encryptedDataByte = SplitStringIntoBytes(encryptedData);

}

catch (Exception)

{

throw new EncryptedTextException("Введены некорректные данные");

}

Byte[] keyBytes = FillKeyBytes(encryptedDataByte.Length, key);

Byte[] openDataByte = Coding(encryptedDataByte, keyBytes);

String openData = Encoding.UTF8.GetString(openDataByte);

return openData;

}

private Byte[] Coding(Byte[] data, Byte[] key)

{

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

{

data[i] ^= key[i];

}

return data;

}

private Byte[] FillKeyBytes(int lengthKey, String key)

{

Byte[] keyBytes = Encoding.UTF8.GetBytes(key);

int differenceSize = lengthKey - keyBytes.Length;

if (differenceSize > 0)

{

Byte[] newKeyBytes = new Byte[keyBytes.Length + differenceSize];

keyBytes.CopyTo(newKeyBytes, 0);

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

{

newKeyBytes[keyBytes.Length + i] = keyBytes[i % keyBytes.Length];

}

keyBytes = newKeyBytes;

}

return keyBytes;

}

private Byte[] SplitStringIntoBytes(String data)

{

String[] splitData = new String[data.Length];

int byteIndex = 0;

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

{

if (data[i] == '-')

{

byteIndex++;

continue;

}

splitData[byteIndex] += data[i];

}

Byte[] dataByte = new Byte[++byteIndex];

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

{

dataByte[i] = byte.Parse(splitData[i], System.Globalization.NumberStyles.HexNumber);

}

return dataByte;

}

}

}

[--- Конец GammaXoring.cs]

[Начало Exceptions.cs ---]

using System;

namespace Program

{

class KeyArgumentException : ArgumentException

{

public KeyArgumentException(String massage) :

base(massage)

{

}

}

class OpenTextArgumentException : ArgumentException

{

public OpenTextArgumentException(String massage) :

base(massage)

{

}

}

class EncryptedTextException : ArgumentException

{

public EncryptedTextException(String massage) :

base(massage)

{

}

}

}

[--- Конец Exceptions.cs]

[Начало MainForm.cs ---]

using System;

using System.Drawing;

using System.Windows.Forms;

namespace Program

{

public partial class MainForm : Form

{

private static Random \_random;

Form \_form;

private bool \_isHeaderMoving = false;

private Point \_cursorPosition;

public MainForm()

{

InitializeComponent();

headerPictureBox.BackColor = ColorTranslator.FromHtml("#a4e6fc");

menuStrip.BackColor = ColorTranslator.FromHtml("#a4effc");

closeFormButton.BackColor = ColorTranslator.FromHtml("#FF0000");

gostRadioButton.Checked = true;

openFileDialog.Filter = "Text files(\*.txt)|\*.txt|All files(\*.\*)|\*.\*";

saveFileDialog.Filter = "Text files(\*.txt)|\*.txt|All files(\*.\*)|\*.\*";

\_random = new Random();

CreateAboutDialog();

this.Show();

if (Properties.Settings.Default.isShowAboutMenu)

{

\_form.ShowDialog();

}

}

private void CreateAboutDialog()

{

\_form = new Form();

\_form.Text = "О программе";

\_form.Size = new Size(550, 250);

\_form.StartPosition = FormStartPosition.CenterParent;

\_form.MaximizeBox = false;

\_form.MinimizeBox = false;

Label textAbout = new Label();

textAbout.AutoSize = true;

textAbout.Anchor = AnchorStyles.Top | AnchorStyles.Left;

textAbout.Left = 10;

textAbout.Top = 10;

textAbout.Text = "Лабораторная работа №2" + Environment.NewLine + Environment.NewLine +

"Создать интерфейс ICipher, который определяет методы поддержки шифрования строк. " + Environment.NewLine +

"В интерфейсе объявляются два метода Encode() и Decode(), которые используются для " + Environment.NewLine +

"шифрования и дешифрования строк, соответственно. Реализовать 2 класса реализующих " + Environment.NewLine +

"данный интерфейс. Алгоритмы: ГОСТ 28147\_89, Гаммирование." + Environment.NewLine + Environment.NewLine +

"Выполнил стедент группы 404, Азаров Даниил Константинович." + Environment.NewLine + Environment.NewLine +

"2022 год.";

CheckBox checkBox = new CheckBox();

checkBox.Location = new Point(295, 177);

checkBox.AutoSize = true;

checkBox.TextAlign = ContentAlignment.MiddleCenter;

checkBox.Text = "Больше не показывать";

checkBox.Checked = !Properties.Settings.Default.isShowAboutMenu;

//checkBox.Anchor = AnchorStyles.Left | AnchorStyles.Bottom;

Button ok = new Button();

ok.Text = "Ок";

ok.AutoSize = true;

ok.Location = new Point(450, 175);

//ok.Anchor = AnchorStyles.Bottom | AnchorStyles.Right;

ok.Click += (sender, e) =>

{

Properties.Settings.Default.isShowAboutMenu = !checkBox.Checked;

Properties.Settings.Default.Save();

\_form.Close();

};

Panel panel = new Panel();

panel.Dock = DockStyle.Fill;

panel.Controls.Add(textAbout);

panel.Controls.Add(checkBox);

panel.Controls.Add(ok);

\_form.Controls.Add(panel);

}

#region Encrypting text function

private void EncryptButton\_Click(object sender, EventArgs e)

{

ICipher cipher = GetCipher();

if(cipher == null)

{

return;

}

try

{

encryptedTextBox.Text = cipher.Encode(openTextBox.Text, keyTextBox.Text);

}

catch (Exception ex)

{

cipherTextBoxErrorProvider.SetError(keyTextBox, ex.Message);

return;

}

cipherTextBoxErrorProvider.Clear();

}

private void DecipherButton\_Click(object sender, EventArgs e)

{

ICipher cipher = GetCipher();

String encryptedData;

try

{

encryptedData = cipher.Decode(encryptedTextBox.Text, keyTextBox.Text);

}

catch (KeyArgumentException ex)

{

cipherTextBoxErrorProvider.SetError(keyTextBox, ex.Message);

return;

}

catch (EncryptedTextException ex)

{

cipherTextBoxErrorProvider.SetError(encryptedTextBox, ex.Message);

return;

}

openTextBox.Text = encryptedData;

cipherTextBoxErrorProvider.Clear();

}

private ICipher GetCipher()

{

if (gostRadioButton.Checked == true)

{

return new Gost28147\_89();

}

else if (gammaXoringRadioButton.Checked == true)

{

return new GammaXoring();

}

return null;

}

private void RandomInputButton\_Click(object sender, EventArgs e)

{

const int minASCII = 33;

const int maxASCII = 127;

int lenghtKey = 0;

if (gostRadioButton.Checked == true)

{

lenghtKey = 32;

}

else if (gammaXoringRadioButton.Checked == true)

{

lenghtKey = openTextBox.Text.Length;

}

keyTextBox.Text = "";

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

{

keyTextBox.Text += Convert.ToChar(\_random.Next(minASCII, maxASCII));

}

}

#endregion

#region Tool strip menu

private void InputOpenToolStripMenuItem\_Click(object sender, EventArgs e)

{

String text = InputData();

if (text != null)

{

openTextBox.Text = text;

}

}

private void InputEncryptToolStripMenuItem\_Click(object sender, EventArgs e)

{

String text = InputData();

if (text != null)

{

encryptedTextBox.Text = text;

}

}

private void InputKeyToolStripMenuItem\_Click(object sender, EventArgs e)

{

String text = InputData();

if (text != null)

{

keyTextBox.Text = text;

}

}

private void SaveOpenToolStripMenuItem\_Click(object sender, EventArgs e)

{

SaveData(openTextBox.Text);

}

private void SaveEncryptToolStripMenuItem\_Click(object sender, EventArgs e)

{

SaveData(encryptedTextBox.Text);

}

private void SaveKeyToolStripMenuItem1\_Click(object sender, EventArgs e)

{

SaveData(keyTextBox.Text);

}

private String InputData()

{

DialogResult result = openFileDialog.ShowDialog();

if (result == DialogResult.OK)

{

String text = "";

try

{

using System.IO.StreamReader sr = new System.IO.StreamReader(openFileDialog.OpenFile());

string inputText;

while ((inputText = sr.ReadLine()) != null)

{

text += inputText;

}

}

catch (System.IO.IOException)

{

text = null;

MessageBox.Show("Файл не может быть прочитан.", "Ошибка!");

}

return text;

}

return null;

}

private void SaveData(String text)

{

DialogResult result = saveFileDialog.ShowDialog();

if (result == DialogResult.OK)

{

try

{

using System.IO.StreamWriter sw = new System.IO.StreamWriter(saveFileDialog.OpenFile());

sw.WriteLine(text);

}

catch (System.IO.IOException)

{

MessageBox.Show("Не удалось сохранить данные.", "Ошибка!");

}

}

}

private void AboutProgramToolStripMenuItem\_Click(object sender, EventArgs e)

{

\_form.ShowDialog();

}

#endregion

#region Form's header functional

private void HeaderPictureBox\_MouseDown(object sender, MouseEventArgs e)

{

if (e.Button == MouseButtons.Left)

{

\_isHeaderMoving = true;

\_cursorPosition = new Point(e.X, e.Y);

}

else

{

\_isHeaderMoving = false;

}

}

private void HeaderPictureBox\_MouseMove(object sender, MouseEventArgs e)

{

if (\_isHeaderMoving)

{

Point cursorMoveTo;

cursorMoveTo = this.PointToScreen(new Point(e.X, e.Y));

cursorMoveTo.Offset(-\_cursorPosition.X, -\_cursorPosition.Y);

this.Location = cursorMoveTo;

}

}

private void HeaderPictureBox\_MouseUp(object sender, MouseEventArgs e)

{

\_isHeaderMoving = false;

}

private void CloseFormButton\_Click(object sender, EventArgs e)

{

Application.Exit();

}

#endregion

}

}

[--- Конец MainForm.cs]

[Начало UnitTest.cs ---]

using System;

using Microsoft.VisualStudio.TestTools.UnitTesting;

using Program;

namespace CipherTest

{

[TestClass]

public class UnitTest

{

[TestMethod]

public void GostEncodingTest()

{

// Arrange

String openText = "It is encoding string! GOST 28147 - 89!";

String expectedEncriptedText = "24-53-CE-08-44-AB-EE-A0-EB-ED-FF-75-59-0D-A0-B8-8F-C3-45-43-E4-" +

"16-1B-22-46-6D-37-98-05-9E-AA-A0-02-46-A3-26-CC-7A-61-44";

String key = "this\_is\_256\_bits\_key\_for\_GOST!!!";

// Act

Gost28147\_89 gost28147\_89 = new Gost28147\_89();

String resultEncriptedText = gost28147\_89.Encode(openText, key);

// Assert

Assert.AreEqual(expectedEncriptedText, resultEncriptedText);

}

[TestMethod]

public void GostDecodingTest()

{

// Arrange

String expectedOpenText = "It is decoding string! GOST 28147 - 89!\0";

String encriptedText = "74-9F-06-5F-67-B6-AF-5C-EB-ED-FF-75-59-0D-A0-B8-8F-" +

"C3-45-43-E4-16-1B-22-46-6D-37-98-05-9E-AA-A0-02-46-A3-26-CC-7A-61-44";

String key = "this\_is\_256\_bits\_key\_for\_GOST!!!";

// Act

Gost28147\_89 gost28147\_89 = new Gost28147\_89();

String resultDecriptedText = gost28147\_89.Decode(encriptedText, key);

// Assert

Assert.AreEqual(expectedOpenText, resultDecriptedText);

}

[TestMethod]

public void GammaXoringEncodingTest()

{

// Arrange

String openText = "It is dencoding string! GammaXoring!";

String expectedEncriptedText = "1F-3F-56-5A-2A-16-1E-38-29-09-33-56-2F-26-5A" +

"-46-3E-05-15-50-40-15-41-1C-7B-26-25-2D-40-30-47-29-11-0B-5A-7C";

String key = "VKv3Y6z]Gj\\2FH=fMqg9.r`<<GH@!h([xe=]";

// Act

GammaXoring gammaXoring = new GammaXoring();

String resultEncriptedText = gammaXoring.Encode(openText, key);

// Assert

Assert.AreEqual(expectedEncriptedText, resultEncriptedText);

}

[TestMethod]

public void GammaXoringDecodingTest()

{

// Arrange

String expectedOpenText = "It is decoding string! GammaXoring!";

String encriptedText = "2D-1A-01-24-03-4A-17-38-31-23-50-1E-24-1D-4D-" +

"32-51-30-46-1E-0D-03-54-1C-5C-42-16-42-3D-41-12-55-0A-09-00";

String key = "dn!Mpjs]RL4wJzmA%B/pj\"t[=/{#e.`<";

// Act

GammaXoring gammaXoring = new GammaXoring();

String resultDecriptedText = gammaXoring.Decode(encriptedText, key);

// Assert

Assert.AreEqual(expectedOpenText, resultDecriptedText);

}

}

}

[--- Конец UnitTest.cs]