УО «Белорусский государственный университет информатики и радиоэлектроники»

Кафедра ПОИТ

Отчет по лабораторной работе №2

по предмету

Теория Информации

Вариант 14

Выполнил:

Бетеня К.С.

Проверил:

Болтак С.В.

Группа 351001

Минск 2025

**Задание:**

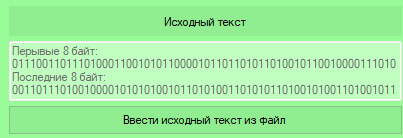
Реализовать систему потокового шифрования и дешифрования для файла с любым содержимым с помощью генератора ключевой последовательности на основе линейного сдвигового регистра с обратной связью LFSR1 (размерность регистра ***36***). Начальное состояние регистра ввести с клавиатуры. Поле для ввода состояния регистра должно игнорировать любые символы кроме 0 и 1. Вывести на экран сгенерированный ключ (последовательность из 0 и 1), исходный файл и зашифрованный файл в двоичном виде. Программа не должна быть написана в консольном режиме. Результат работы программы – зашифрованный/расшифрован- ный файл.

**Примитивный многочлен**

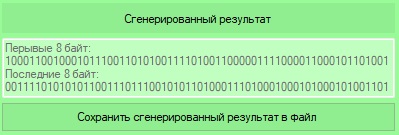
*x*36 + *x*11 + 1

**Работа с файлами**

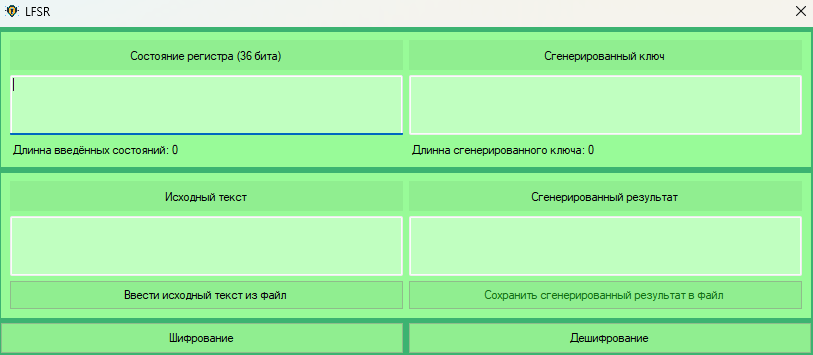
Взятие исходного файла:



Загрузка результата файла:



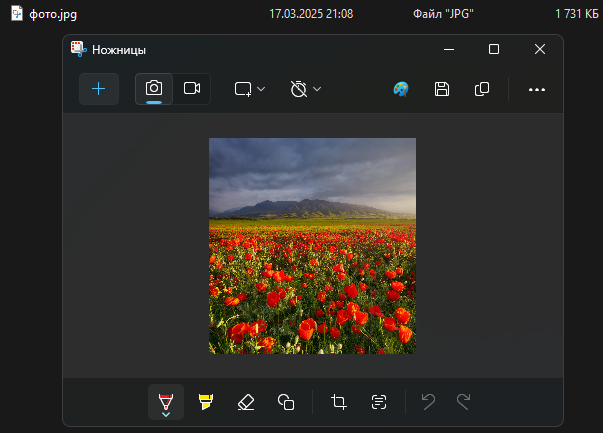
Общий внешний вид страницы:



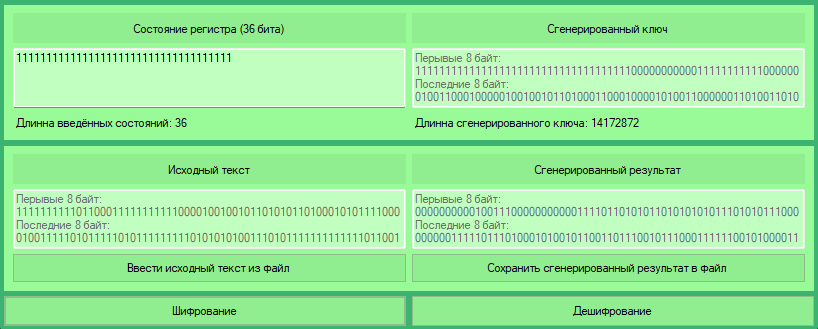
**Тесты разными файлами**

1. **Изображение**

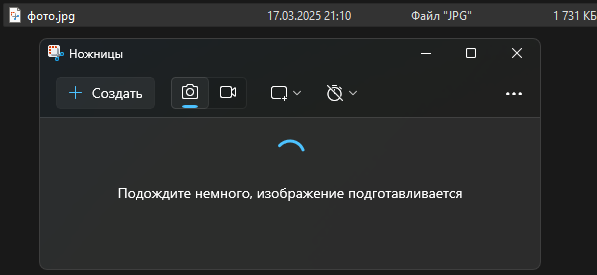
Файл до обработки:



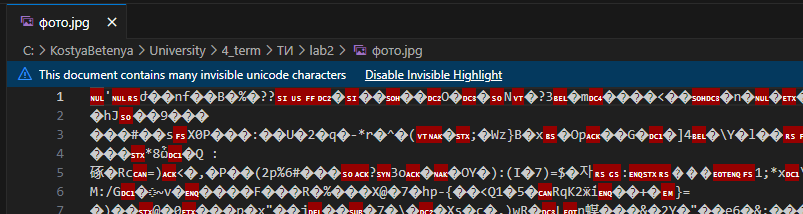
Шифрование:



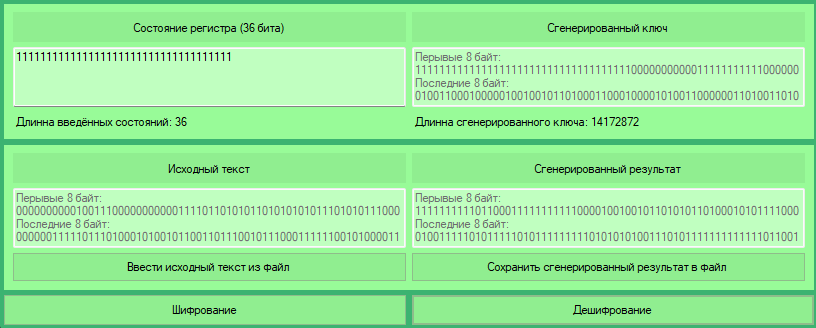
Файл после шифрования (файл не загрузился по причине неверного формата):



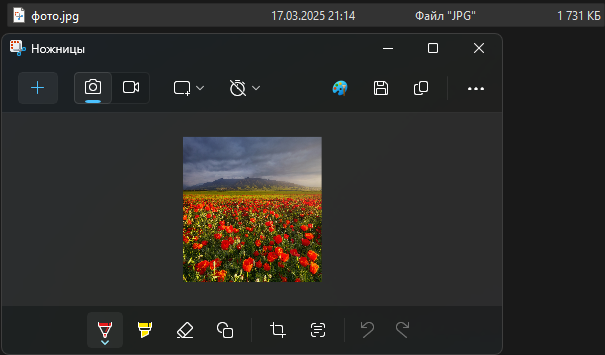
Биты после шифрования были переконвертированны в байты:



Дешифрование файла:

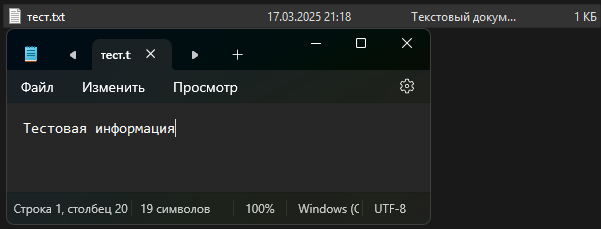


Файл после дешифрования (фотография восстановилась):

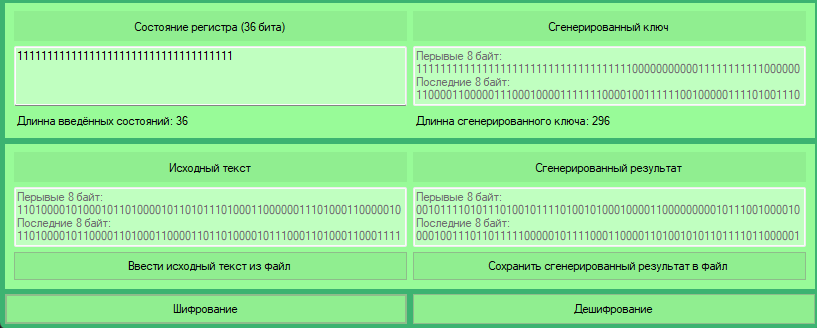


1. **Текстовый файл**

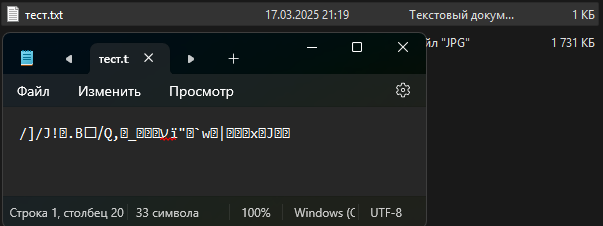
Файл до обработки:



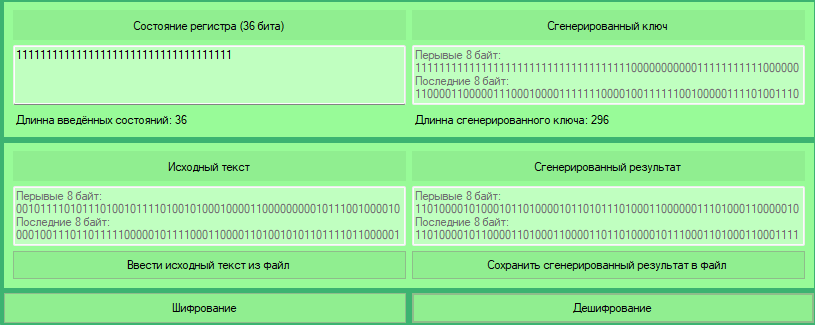
Шифрование:



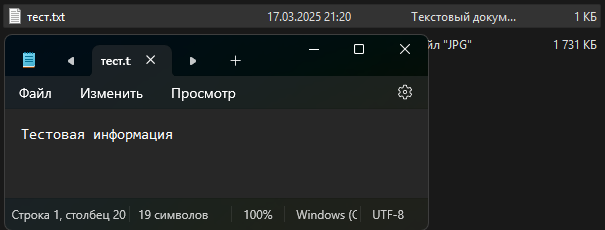
Файл после шифрования (файл содержит зашифрованные байты):



Дешифрование файла:



Файл после дешифрования (информация восстановилась):



**Тесты разными ключами**

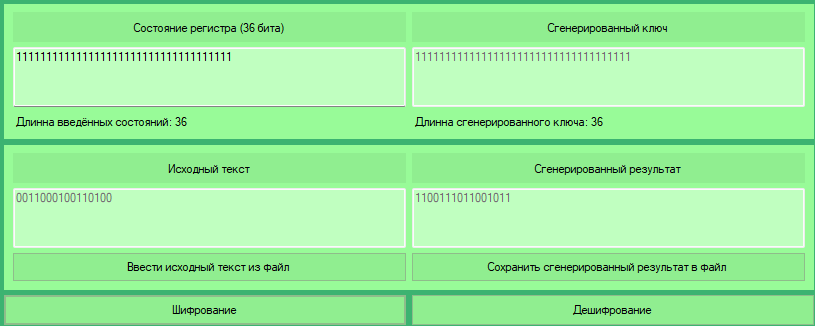
1. **Простой ключ, небольшой текст.**

Ключ: 111111111111111111111111111111111111

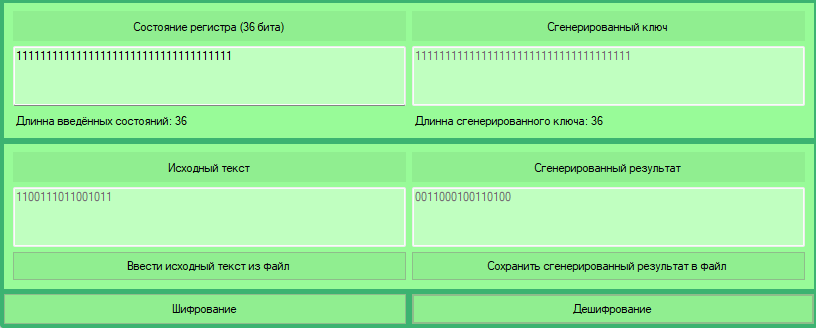
Исходный текст: 0011000100110100

Шифротекст: 1100111011001011

Шифрование:



Дешифрирование:



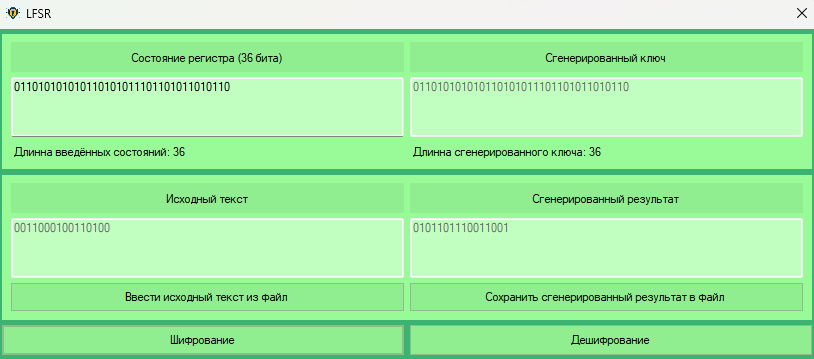
1. **Сложный ключ, небольшой текст.**

Ключ: 011010101010110101011101101011010110

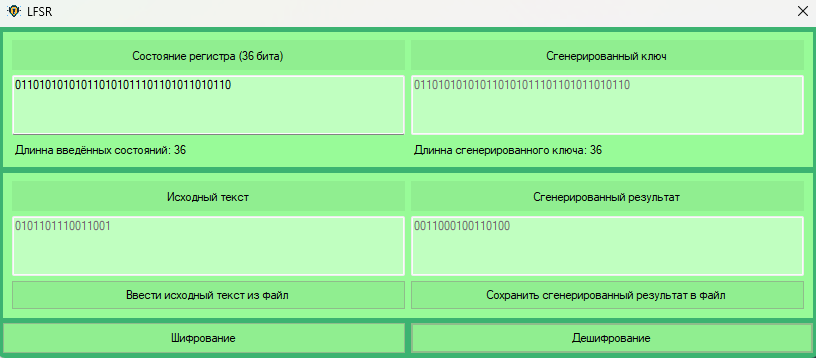
Исходный текст: 0011000100110100

Шифротекст: 0101101110011001

Шифрование:



Дешифрирование:



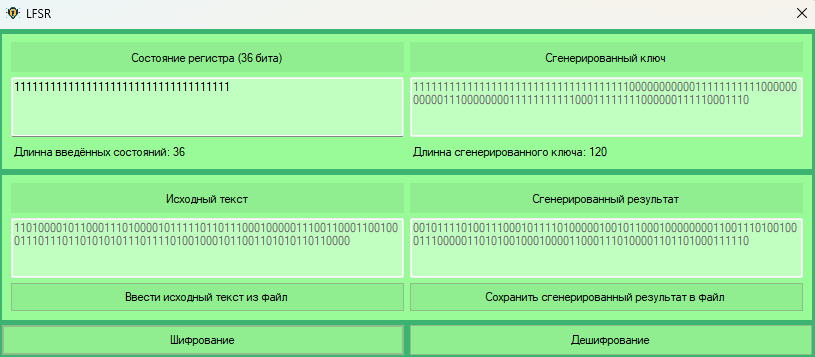
1. **Простой ключ, большой текст.**

Ключ: 111111111111111111111111111111111111

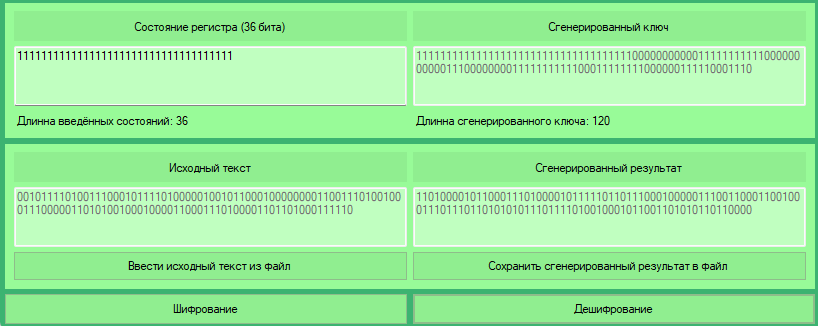
Исходный текст: 110100001011000111010000101111101101000010111011110100011000110011010001100010001101000010111110110100001011100100100000

Шифротекст: 001011110100111000101111010000010010010111101110100001001101111111100010101110111110000110101111110000011010100111010000

Шифрование:



Дешифрирование:



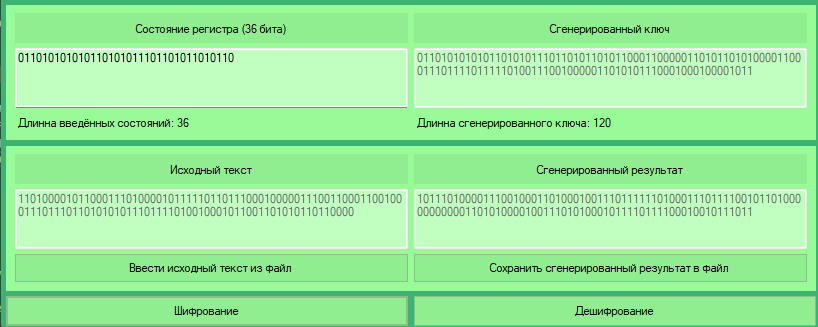
1. **Сложный ключ, большой текст.**

Ключ: 011010101010110101011101101011010110

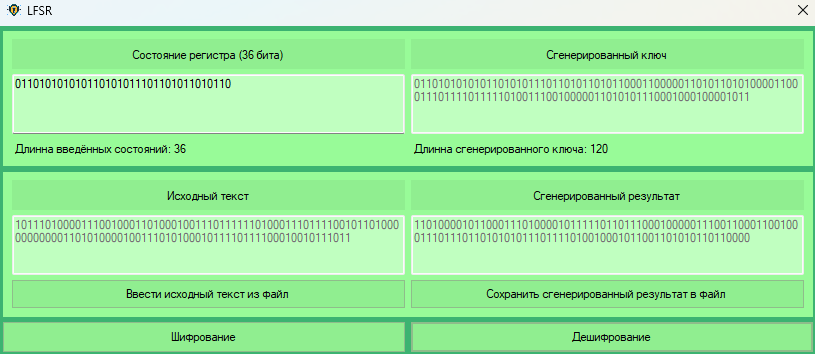
Исходный текст: 110100001011000111010000101111101101000010111011110100011000110011010001100010001101000010111110110100001011100100100000

Шифротекст: 101110100001110010001101000100111011010001110000101101010101000010010111010101001001101101111100100110110111111110011110

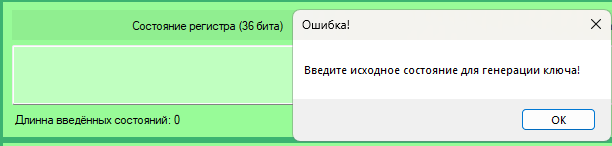
Шифрование:

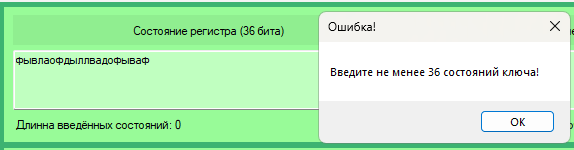


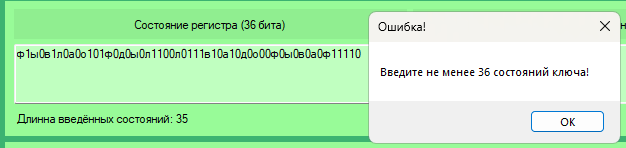
Дешифрирование:

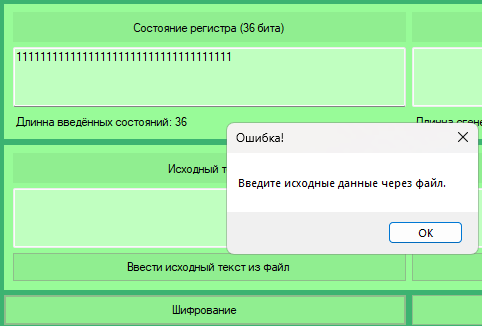
****

1. **Проверка на не валидных данных.**









P.S. Алгоритм генерации ключей для примеров приведён в Excel файле.

**Исходный код**

using System;

using System.IO;

using System.Linq;

using System.Security.Cryptography;

using System.Text;

using System.Web;

using System.Windows.Forms;

namespace ToI\_2

{

public partial class MainForm : Form

{

private char[] goodKeys = { '0', '1' };

private string PLAIN\_TEXT;

private string GENERATED\_KEY;

private string CIPER\_TEXT;

public MainForm()

{

InitializeComponent();

setCurrentKeyLength(0);

}

private void setCurrentKeyLength(int length)

{

keyLengthLabel.Text = $"Длинна введённых состояний: {length}";

}

private string getCurrentKey(string str)

{

string res = "";

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

if (goodKeys.Contains(str[i]))

res += str[i];

return res;

}

private (bool, string) getCurrentKeyString(string str)

{

if (str.Length == 0)

return (false, "Введите исходное состояние для генерации ключа!");

string res = getCurrentKey(str);

if (res.Length < 36)

return (false, "Введите не менее 36 состояний ключа!");

return (true, res);

}

private string getFirstBytes(string str, int bytes)

{

return str.Substring(0, bytes \* 8);

}

private string getLastBytes(string str, int bytes)

{

return str.Substring(str.Length - bytes \* 8, bytes \* 8);

}

private void setGeneratedText(TextBox textBox, string str)

{

const int bytesToSee = 8;

if (str.Length > bytesToSee \* 8 \* 2)

textBox.Text = $"Перывые 8 байт: {getFirstBytes(str, bytesToSee)}\n\nПоследние 8 байт: {getLastBytes(str, bytesToSee)}";

else

textBox.Text = str;

}

private char[] shiftKey(char[] key)

{

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

key[i] = key[i + 1];

return key;

}

private char xorNums(char bit1, char bit2)

{

int num1 = bit1 - '0';

int num2 = bit2 - '0';

int result = num1 ^ num2;

char xorResult = (char)(result + '0');

return xorResult;

}

private char getNewValue(int[] genPos, char[] key)

{

char first;

char second = key[genPos[0] - 1];

for (var i = 1; i < genPos.Length; i++)

{

first = second;

second = key[genPos[i] - 1];

second = xorNums(first, second);

}

return second;

}

private string generateByLFSR(string register, int length)

{

char[] genKey = register.ToArray();

char[] currentKey = register.ToArray();

int[] xorPositions = { 36, 11 }; // from 1 .. 36

StringBuilder bits = new StringBuilder();

for (var i = 0; i < currentKey.Length; i++)

{

bits.Append(currentKey[i]);

}

while (bits.Length < length)

{

genKey = shiftKey(genKey);

genKey[genKey.Length - 1] = getNewValue(xorPositions, genKey);

bits.Append(genKey[genKey.Length - 1]);

}

return bits.ToString();

}

private void keyTextBox\_TextChanged(object sender, EventArgs e)

{

setCurrentKeyLength(getCurrentKey(keyTextBox.Text).Length);

}

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

{

//string str;

//bool isCorrect;

//(isCorrect, str) = getCurrentKeyString(keyTextBox.Text);

//if (!isCorrect)

//{

// MessageBox.Show(str, "Ошибка!");

// return;

//}

// generate

//string key = generateByLFSR(str.Substring(0, 36));

//setGeneratedKey(key);

}

private string encryptionAlghoritm(string plain, string key)

{

int test = key.Length;

StringBuilder bits = new StringBuilder();

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

{

char p = plain[i];

char k = key[i];

bits.Append(xorNums(p, k));

}

return bits.ToString();

}

private void ENCRIPTION()

{

/\* Key part \*/

string key;

bool isCorrect;

(isCorrect, key) = getCurrentKeyString(keyTextBox.Text);

if (!isCorrect)

{

MessageBox.Show(key, "Ошибка!");

return;

}

/\* Plain text part \*/

if (PLAIN\_TEXT == null || PLAIN\_TEXT.Length == 0)

{

MessageBox.Show("Введите исходные данные через файл.", "Ошибка!");

return;

}

/\* Generate key part \*/

GENERATED\_KEY = generateByLFSR(key.Substring(0, 36), PLAIN\_TEXT.Length);

setGeneratedText(genKeyTextBox, GENERATED\_KEY);

/\* Cipher part \*/

CIPER\_TEXT = encryptionAlghoritm(PLAIN\_TEXT, GENERATED\_KEY);

setGeneratedText(cypherTextBox, CIPER\_TEXT);

/\* Enables \*/

inFileButton.Enabled = true;

genKeySizeLabel.Text = $"Длинна сгенерированного ключа: {GENERATED\_KEY.Length}";

}

private void button1\_Click(object sender, EventArgs e)

{

ENCRIPTION();

}

private (bool, string) ReadFileAsBits(string filePath)

{

try

{

StringBuilder bits = new StringBuilder();

using (FileStream fs = new FileStream(filePath, FileMode.Open, FileAccess.Read))

{

int byteRead;

while ((byteRead = fs.ReadByte()) != -1)

{

bits.Append(Convert.ToString(byteRead, 2).PadLeft(8, '0'));

}

}

if (bits.Length > 0)

return (true, bits.ToString());

return (false, null);

}

catch

{

return (false, null);

}

}

private (bool, string) WriteFileAsBits(string filePath)

{

string bits = CIPER\_TEXT;

try

{

using (FileStream fs = new FileStream(filePath, FileMode.Create, FileAccess.Write))

{

for (int i = 0; i < bits.Length; i += 8)

{

string byteBits = bits.Substring(i, Math.Min(8, bits.Length - i));

byte byteToWrite = Convert.ToByte(byteBits, 2);

fs.WriteByte(byteToWrite);

}

}

return (true, null);

}

catch

{

return (false, null);

}

}

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

{

if (openFileDialog.ShowDialog() == DialogResult.OK)

{

bool status;

string result;

(status, result) = ReadFileAsBits(openFileDialog.FileName);

if (status) {

PLAIN\_TEXT = result;

setGeneratedText(plainTextBox, PLAIN\_TEXT);

return;

}

MessageBox.Show("Данные в файле не являются корректными.", "Ошибка!");

}

}

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

{

if (saveFileDialog.ShowDialog() == DialogResult.OK)

{

string str;

bool status;

(status, str) = WriteFileAsBits(saveFileDialog.FileName);

MessageBox.Show(status ? "Данные успешно записаны в файл." : "Ошибка при записи в файл.", status ? "Успех" : "Ошибка");

}

}

private void button2\_Click(object sender, EventArgs e)

{

ENCRIPTION();

}

}

}