­Министерство высшего и профессионального образования РФ

Государственное образовательное учреждение

высшего профессионального образования

Ульяновский государственный технический университет

Кафедра «Вычислительная техника»

Дисциплина "МиСЗКИ"

Лабораторная работа №2

Изучение работы блоковых шифров и способов их объединения.

Выполнил:

студент группы БЭВМд-41

Ключников Дмитрий

Проверил:

Мартынов А. И.

Ульяновск 2012

***Задание:***

1. Изучить блочные алгоритмы шифрования: алгоритм перестановки, алгоритм скремблеров, алгоритм замены по таблице, матричный метод преобразования и алгоритм Винжера.
2. Изучить режимы использования блочных шифров (ECB, CBC, CFB и OFB).
3. Изучить способы объединения блочных шифров (многократное шифрование, сеть Фейстела).
4. Реализовать систему в соответствии с вариантами

***Дополнительные требования к лабораторной работе:***

1. Функции шифрования/дешифрования + вспомогательные функции необходимые для осуществления процесса кодирования/декодирования помещаются в **отдельную** библиотеку dll (это необходимо для выполнения третьей лабораторной работы).
2. Пароль в зашифрованном виде записывается в закодированный файл. При попытке расшифровать его с другим паролем выводится сообщение об ошибке.
3. В процессе кодирования осуществляется **подсчет контрольной суммы**. При декодировании осуществляется проверка контрольной суммы. При несовпадении выдается сообщение об ошибке. Контрольная сумма считается по открытому тексту.
4. Должна иметься возможность отключения режима использования блочного шифра и работа программы в режиме простого ECB.
5. Паролем может быть любая последовательность символов (русских и английских, цифр, знаков препинания и т.д.).
6. Программа должна быть оформлена в виде удобной утилиты, позволяющей работать с любыми файлами.
7. Программа должна обеспечивать шифрование файлов произвольной длины.
8. Текст программы оформляется прилично (удобочитаемо, с описанием ВСЕХ функций, переменных и критических мест).
9. В процессе работы программа ОБЯЗАТЕЛЬНО выдает информацию о состоянии процесса кодирования/декодирования.
10. После завершения работы программы выдает информацию о скорости шифрования / дешифрования (символ /сек)
11. Интерфейс программы может быть произвольным, но удобным и понятным (разрешается использование библиотек VCL)
12. Среда разработки и язык программирования могут быть произвольными.

***Задание по варианту:***

Алгоритм: Перестановка

Режим использования блочного шифра: Сеть Фейстеля

Длина блока: 10 байт

Число подблоков: 3

**Перестановка**

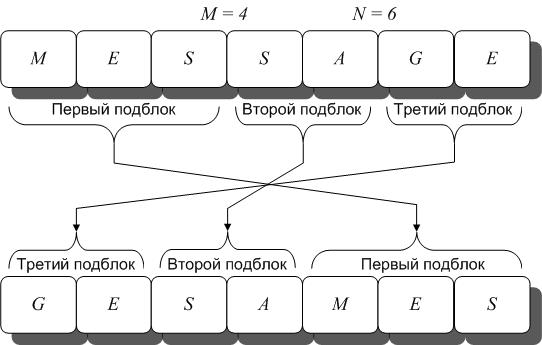
В алгоритме *перестановки* в каждом блоке меняется последовательность некоторых подблоков внутри блока, например байт или бит в слове, причем порядок перестановок определяется ключом.

Пусть имеется некоторое исходное сообщение «MESSAGE», которое необходимо закодировать (Рисунок II‑14). Это сообщение имеет длину в 7 байт (если используется ASCII код). Разобьем этот блок текста на три подблока: «MES», «SA» и «GE». Числа *M* и *N*, которые определяют границы подблоков, получены при помощи генератора случайных чисел и зависят от конкретного ключа. Для данного примера *M* = 4, *N* = 6.

Аналогично производится операция перестановки над группами бит.

*Примечание*: для блоков большой длины метод перестановок становится неэффективным, так как и длина подблоков в этом случае также велика. Поэтому шифрование должно осуществляться в несколько раундов, то есть перемешивание производится несколько раз над одним и тем же блоком, но с различными значениями коэффициентов M и N. При дешифровании необходимо воспроизвести эту последовательность псевдослучайных чисел в обратном порядке.

Рисунок ‑. Пример перестановки



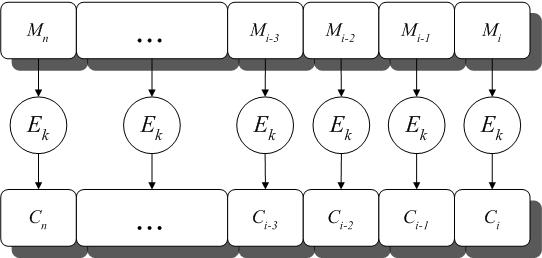
В данной реализации весь файл делится на блоки по 11 байт, каждый из которых шифруется методом перестановок. Каждый блок делится на 4 блока при помощи генератора случайных чисел, затравкой которого служит значение пароля.

**Режимы использования блочного шифра**

В режиме *электронной кодировочной книги (ECB)* каждый блок исходного текста шифруется блочным шифром независимо от других (Рисунок II‑14).



Режим электронной кодировочной книги (ECB)



Стойкость режима ECB равна стойкости самого шифра, однако структура шифрованного текста при этом не скрывается, так как каждый одинаковый блок исходного текста приводит к появлению одинакового блока шифрованного текста. Режим ECB допускает простое распараллеливание для увеличения скорости шифрования. Режим ECB соответствует режиму простой замены алгоритма ГОСТ 28147-89.

**Сеть Фейстеля**

Одним из наиболее распространенных способов задания блочных шифров является использование так называемых сетей Фейстеля. Сеть Фейстеля представляет собой общий метод преобразования произвольной функции (обычно называемой F-функцией) в перестановку на множестве блоков. Пусть X – блок текста (длина блока текста обязательно должна быть четной). Представим его в виде двух подблоков одинаковой длины X = {A,B}. Тогда одна итерация (или раунд) сети Фейстела определяется как



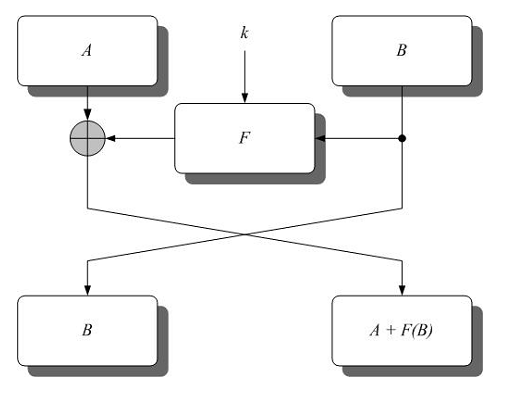
где X i +1 = {Ai −1 , Bi −1 }, || – операция конкатенации, а ⊕ – побитовое исключающее ИЛИ (Рисунок 2.20). Сеть Фейстела состоит из некоторого фиксированного числа итераций, определяемого соображениями стойкости разрабатываемого шифра, при этом на последней итерации перестановка местами половин блоков текста не производится, так как это не влияет на стойкость шифра.

Данная структура шифров обладает рядом достоинств, а именно:

• Процедуры шифрования и дешифрования совпадают, с тем исключением, что при дешифровании ключевая информация используется в обратном порядке.

• Для построения устройств шифрования можно использовать те же блоки в цепях шифрования и дешифрования.

Недостатком является то, что на каждой итерации изменяется только половина обрабатываемого текста, что приводит к необходимости увеличивать число итераций для достижения требуемой стойкости. В отношении выбора F-функции определенных стандартов не существует, однако, как правило, эта функция представляет собой последовательность зависящих от ключа нелинейных замен, перемешивающих перестановок и сдвигов.



**Код программы**

using System;

using System.Collections.Generic;

using System.IO;

using System.Linq;

using System.Reflection;

using System.Text;

using System.Threading;

namespace CommonLibrary

{

/// <summary>

/// Пользовательский аттрибут, применяемый к свойствам класса, которые являются настройками метода и вписываются в кодируемый файл

/// </summary>

public class ParametrCryptAttribute : Attribute

{ }

public enum CryptResult

{

/// <summary>

/// Усшнешно

/// </summary>

Success,

/// <summary>

/// Несовпадение значений параметров записанных в файл с заданными

/// </summary>

MismatchValueParameters,

/// <summary>

/// Несовпадение контрольной суммы

/// </summary>

MismatchСhecksum

}

public abstract class CrypterBlocks

{

public delegate void ComplitedHandler(CryptResult cryptResult, string password);

public delegate void ComplitedStringHandler(string result);

public delegate void ComplitedBytesHandler(byte[] result);

public event ComplitedStringHandler EncryptStringComplitedEvent;

public event ComplitedStringHandler DecryptStringComplitedEvent;

public event ComplitedBytesHandler EncryptBytesComplitedEvent;

public event ComplitedBytesHandler DecryptBytesComplitedEvent;

public event ComplitedHandler DecryptComplitedEvent;

public event ComplitedHandler EncryptComplitedEvent;

protected delegate CryptResult CryptFunctionDelegate(string inputFile, string outputFile, bool reverse);

protected delegate void CryptStringFunctionDelegate(ref string resstr);

protected delegate void CryptBytesFunctionDelegate(byte[] data);

public int MaxValueProcess { get; protected set; }

public int CurrentValueProcess { get; protected set; }

/// <summary>

/// Пароль - ключ

/// </summary>

public string Password { get; protected set; }

/// <summary>

/// Количество подблоков

/// </summary>

[ParametrCryptAttribute]

public byte SubBlocks { get; protected set; }

/// <summary>

/// Длина блока

/// </summary>

[ParametrCryptAttribute]

public byte BlockLenth { get; protected set; }

protected byte[] keys;

public CrypterBlocks(byte BlockLenth, byte subBlocks, string pass)

{

this.BlockLenth = BlockLenth;

this.SubBlocks = subBlocks;

this.Password = pass;

GenKeys(pass);

}

/// <summary>

/// Генерация длины блоков псевдослучайным генератором, зависимым от входного ключа-пароля

/// </summary>

/// <param name="key"></param>

protected abstract void GenKeys(string key);

/// <summary>

/// Основная функция изменения блока данных по данному методу

/// </summary>

/// <param name="isEncrypt">если true - шифрация, false - дешифрация</param>

/// <param name="buffer">массив данных</param>

/// <returns></returns>

protected abstract byte[] CryptBlock(bool isEncrypt, byte[] buffer);

/// <summary>

/// Общая функция преобразования входного массива байтов

/// </summary>

/// <param name="inputFile">путь входного файла</param>

/// <param name="outputFile">путь выходного файла</param>

/// <param name="isEncrypt">если true - шифрация, false - дешифрация</param>

/// <param name="key">ключ</param>

protected virtual CryptResult Crypt(string inputFile, string outputFile, bool isEncrypt)

{

var inputstream = File.OpenRead(inputFile);

var sr = new BinaryReader(inputstream);

FileStream outputstream = null;

BinaryWriter wr = null;

this.CurrentValueProcess = 0;

this.MaxValueProcess = (int)(inputstream.Length);

byte crc = 0xFF;

byte crcDec = 0;

long crcPosition = 0;

if (isEncrypt)

{

outputstream = File.OpenWrite(outputFile);

wr = new BinaryWriter(outputstream);

var enc = MaHash8v64.GetHashCode(this.Password);

// var bytesPass = Encoding.Unicode.GetBytes(enc);

wr.Write(enc);

var properties = this.GetType().GetProperties().Where(a => a.GetCustomAttributes(true).OfType<ParametrCryptAttribute>().Count() != 0).ToArray();

foreach (var p in properties)

wr.Write((byte)p.GetValue(this, null));

crcPosition = outputstream.Position;

outputstream.Seek(1, SeekOrigin.Current);

}

else

{

var hashPass = sr.ReadUInt32();

var properties = this.GetType().GetProperties().Where(a => a.GetCustomAttributes(true).OfType<ParametrCryptAttribute>().Count() != 0).ToArray();

var paramsValue = new byte[properties.Length];

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

paramsValue[i] = sr.ReadByte();

if (hashPass != MaHash8v64.GetHashCode(this.Password))

{

inputstream.Close();

return CryptResult.MismatchValueParameters;

}

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

if (paramsValue[i] != (byte)properties[i].GetValue(this, null))

{

inputstream.Close();

return CryptResult.MismatchValueParameters;

}

crcDec = sr.ReadByte();

}

if (wr == null)

{

outputstream = File.OpenWrite(outputFile);

wr = new BinaryWriter(outputstream);

}

while (true)

{

var buffer = sr.ReadBytes(this.BlockLenth);

if (!isEncrypt)

crc = GetCRC(crc, buffer);

if (buffer.Length == BlockLenth)

buffer = CryptBlock(isEncrypt, buffer);

if (isEncrypt)

crc = GetCRC(crc, buffer);

if (buffer.Length != BlockLenth)

{

this.CurrentValueProcess = this.MaxValueProcess - 1;

wr.Write(buffer);

break;

}

wr.Write(buffer);

this.CurrentValueProcess += 10;

}

if (isEncrypt)

{

outputstream.Seek(crcPosition, SeekOrigin.Begin);

wr.Write(crc);

}

else

{

if (crc != crcDec)

{

inputstream.Close();

outputstream.Close();

return CryptResult.MismatchСhecksum;

}

}

inputstream.Close();

outputstream.Close();

return CryptResult.Success;

}

private byte GetCRC(byte crc, byte[] buffer)

{

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

{

crc ^= buffer[j];

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

crc = (byte)((crc & 0x80) != 0 ? (crc << 1) ^ 0x31 : crc << 1);

}

return crc;

}

protected CryptStringFunctionDelegate GetCryptStringDelegate(bool isEncrypt)

{

return new CryptStringFunctionDelegate(delegate(ref string strr)

{

strr = Encrypt(strr);

});

}

protected CryptBytesFunctionDelegate GetCryptBytesDelegate(bool isEncrypt)

{

return new CryptBytesFunctionDelegate(delegate(byte[] data)

{

Encrypt(data);

});

}

public virtual void EncryptAsync(string inputFile, string outputFile)

{

IAsyncResult res = null;

var syncCurrent = SynchronizationContext.Current;

var cfDelegate = new CryptFunctionDelegate(this.Crypt);

res = cfDelegate.BeginInvoke(inputFile, outputFile, true, delegate

{

syncCurrent.Send(delegate

{

var cryprres = cfDelegate.EndInvoke(res);

if (EncryptComplitedEvent != null)

EncryptComplitedEvent(cryprres, Password);

}, null);

}, null);

}

public virtual void DecryptAsync(string inputFile, string outputFile)

{

IAsyncResult res = null;

var cfDelegate = new CryptFunctionDelegate(this.Crypt);

var syncCurrent = SynchronizationContext.Current;

res = cfDelegate.BeginInvoke(inputFile, outputFile, false, delegate

{

syncCurrent.Send(delegate

{

var cryprres = cfDelegate.EndInvoke(res);

if (DecryptComplitedEvent != null)

DecryptComplitedEvent(cryprres, Password);

}, null);

}, null);

}

public virtual string Encrypt(string str)

{

var data = Encoding.Unicode.GetBytes(str);

var res = CryptBlock(true, data);

return Encoding.Unicode.GetString(res);

}

public virtual void EncryptAsync(string str)

{

IAsyncResult res = null;

var resstr = str + "";

var deleg = GetCryptStringDelegate(true);

var syncCurrent = SynchronizationContext.Current;

res = deleg.BeginInvoke(ref resstr, delegate

{

syncCurrent.Send(delegate

{

deleg.EndInvoke(ref resstr, res);

if (EncryptStringComplitedEvent != null)

EncryptStringComplitedEvent(resstr);

}, null);

}, null);

}

public virtual string Decrypt(string str)

{

var data = Encoding.Unicode.GetBytes(str);

var res = CryptBlock(true, data);

return Encoding.Unicode.GetString(res);

}

public virtual void DecryptAsync(string str)

{

IAsyncResult res = null;

var resstr = str + "";

var deleg = GetCryptStringDelegate(false);

var syncCurrent = SynchronizationContext.Current;

res = deleg.BeginInvoke(ref resstr, delegate

{

syncCurrent.Send(delegate

{

deleg.EndInvoke(ref resstr, res);

if (DecryptStringComplitedEvent != null)

DecryptStringComplitedEvent(resstr);

}, null);

}, null);

}

public virtual byte[] Encrypt(byte[] data)

{

var clone = (byte[])data.Clone();

return CryptBlock(true, clone);

}

public virtual void EncryptAsync(byte[] data)

{

IAsyncResult res = null;

var deleg = GetCryptBytesDelegate(true);

var clone = (byte[])data.Clone();

var syncCurrent = SynchronizationContext.Current;

res = deleg.BeginInvoke(clone, delegate

{

syncCurrent.Send(delegate

{

deleg.EndInvoke(res);

if (EncryptBytesComplitedEvent != null)

EncryptBytesComplitedEvent(clone);

}, null);

}, null);

}

public virtual byte[] Decrypt(byte[] data)

{

var clone = (byte[])data.Clone();

return CryptBlock(false, clone);

}

public virtual void DecryptAsync(byte[] data)

{

IAsyncResult res = null;

var deleg = GetCryptBytesDelegate(false);

var clone = (byte[])data.Clone();

var syncCurrent = SynchronizationContext.Current;

res = deleg.BeginInvoke(clone, delegate

{

syncCurrent.Send(delegate

{

deleg.EndInvoke(res);

if (DecryptBytesComplitedEvent != null)

DecryptBytesComplitedEvent(clone);

}, null);

}, null);

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace CommonLibrary

{

public class ECB : CrypterBlocks

{

public ECB(byte BlockLenth, byte subBlocks, string pass)

: base(BlockLenth, subBlocks, pass)

{

inds = new int[this.SubBlocks];

keysDecrypt = new byte[this.SubBlocks];

for (int i = 0; i < this.SubBlocks; i++)

inds[i] = -1;

var gen = new CongruentialGenerator(MaHash8v64.GetHashCode(this.Password));

for (int i = 0; i < this.SubBlocks - 1; i++)

{

byte newValue;

do

{

newValue = (byte)gen.Next(0, this.SubBlocks - 1);

} while (inds[newValue] != -1 || (newValue == 0 && i == 0));

inds[newValue] = i;

}

for (int i = 0; i < this.SubBlocks; i++)

{

if (inds[i] == -1)

inds[i] = this.SubBlocks - 1;

keysDecrypt[i] = keys[inds[i]];

}

}

private int[] inds;

private byte[] keysDecrypt;

protected override byte[] CryptBlock(bool isEncrypt, byte[] buffer)

{

var newblock = new byte[BlockLenth];

int ind = 0;

for (int i = 0; i < this.SubBlocks; i++)

{

if (isEncrypt)

for (int k = 0; k < keys[inds[i]]; k++)

newblock[ind++] = buffer[keys.Take(inds[i]).Sum(a => a) + k];

else

for (int k = 0; k < keysDecrypt[inds[i]]; k++)

newblock[ind++] = buffer[keysDecrypt.Take(inds[i]).Sum(a => a) + k];

}

return newblock;

}

protected override void GenKeys(string key)

{

var gen = new CongruentialGenerator(MaHash8v64.GetHashCode(key));

keys = new byte[this.SubBlocks];

for (var i = 0; i < keys.Length - 1; i++)

keys[i] = (byte)gen.Next(1, this.BlockLenth - keys.Sum(a => a) - (this.SubBlocks - i - 1));

keys[keys.Length - 1] = (byte)(this.BlockLenth - keys.Sum(a => a));

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Reflection;

using System.Text;

using System.IO;

using System.ComponentModel;

namespace CommonLibrary

{

/// <summary>

/// Сеть Фейстеля - методов построения блочных шифров

/// </summary>

public class FeistelNetwork : CrypterBlocks

{

/// <summary>

/// Количество иттераций

/// </summary>

[ParametrCryptAttribute]

public byte Rounds { get; private set; }

/// <summary>

/// Создание экземпляра

/// </summary>

/// <param name="BlockLenth">Длина блока</param>

/// <param name="rounds">Количество иттераций</param>

public FeistelNetwork(byte BlockLenth, byte subBlocks, string pass, byte rounds)

: base(BlockLenth, subBlocks, pass)

{

this.Rounds = rounds;

}

protected override void GenKeys(string key)

{

var gen = new CongruentialGenerator(MaHash8v64.GetHashCode(key));

keys = new byte[this.SubBlocks];

for (var i = 0; i < keys.Length - 1; i++)

keys[i] = (byte)gen.Next(1, this.BlockLenth / 2 - keys.Sum(a => a) - (this.SubBlocks - i - 1));

keys[keys.Length - 1] = (byte)(this.BlockLenth / 2 - keys.Sum(a => a));

}

/// <summary>

/// Функция преобразования данных, сеть Фейстеля, изменненной левой ^ правой части и присваивание к левой части

/// </summary>

/// <param name="data">массив данных</param>

/// <param name="lk">преобразованный массив левой части</param>

/// <param name="indL">индекс начала</param>

private void XORl(ref byte[] data, byte[] lk, int indL)

{

byte temp;

for (var i = 0; i < BlockLenth / 2; i++)

{

temp = data[i + indL + BlockLenth / 2];

data[i + indL + BlockLenth / 2] = data[i + indL];

data[i + indL] = (byte)(temp ^ lk[i]);

}

}

/// <summary>

/// Функция преобразования данных, сеть Фейстеля, изменненной левой ^ правой части и присваивание к правой части

/// </summary>

/// <param name="data">массив данных</param>

/// <param name="lk">преобразованный массив левой части</param>

/// <param name="indL">индекс начала</param>

private void XORr(ref byte[] data, byte[] lk, int indL)

{

for (var i = 0; i < BlockLenth / 2; i++)

data[i + indL + BlockLenth / 2] = (byte)(data[i + indL + BlockLenth / 2] ^ lk[i]);

}

/// <summary>

/// Функция изменения блока данных - перестановка

/// </summary>

/// <param name="data">массив данных</param>

/// <param name="keys"массив ключей></param>

/// <param name="ind">индекс начала</param>

/// <returns></returns>

private byte[] F(byte[] data, int ind)

{

var clone = new byte[BlockLenth / 2];

for (var i = 0; i < BlockLenth / 2 - keys[0]; i++)

clone[i] = data[ind + i + keys[0]];

for (var i = 0; i < keys[0]; i++)

clone[i + BlockLenth / 2 - keys[0]] = data[i + ind];

return clone;

}

/// <summary>

/// Основная функция изменения блока данных по данному методу

/// </summary>

/// <param name="isEncrypt">если true - шифрация, false - дешифрация</param>

/// <param name="buffer">массив данных</param>

/// <returns></returns>

protected override byte[] CryptBlock(bool isEncrypt, byte[] buffer)

{

if (buffer.Length == this.BlockLenth)

{

var round = isEncrypt ? this.Rounds : 1;

for (byte k = 0; k < this.Rounds; k++)

{

if (k < this.Rounds - 1)

XORl(ref buffer, F(buffer, 0), 0);

else

XORr(ref buffer, F(buffer, 0), 0);

round += isEncrypt ? -1 : 1;

}

}

else

{

}

return buffer;

}

}

}

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Windows.Forms;

using CommonLibrary;

using System.IO;

namespace Lab2

{

public partial class MainForm : Form

{

private CrypterBlocks crypter;

private Timer timer;

private DateTime oldTime;

public MainForm()

{

InitializeComponent();

timer = new Timer { Interval = 10 };

timer.Tick += new EventHandler(timer\_Tick);

rb.Minimum = 0;

rb.Maximum = 1000;

}

public MainForm(string[] args)

: this()

{

passTB.Text = args[0];

encryptRB.Checked = args[1] == "1";

decryptRB.Checked = args[1] == "2";

inputBrowseTB.Text = args[2];

outputBrowseTB.Text = args[3];

roundTB.Text = 3 + "";

lenthBlockTB.Text = 10 + "";

subblockTB.Text = 3 + "";

}

private void timer\_Tick(object sender, EventArgs e)

{

if (crypter.MaxValueProcess == 0)

return;

var diff = DateTime.Now - oldTime;

SpeedLabel.Text = string.Format("Скорость: {0:N} симв/сек", (crypter.CurrentValueProcess / diff.TotalSeconds));

rb.Value = (int)((crypter.CurrentValueProcess / (double)crypter.MaxValueProcess) \* 1000);

}

private void onlyDigit\_KeyPress(object sender, KeyPressEventArgs e)

{

if (Char.IsDigit(e.KeyChar)) return;

if (e.KeyChar != (char)Keys.Back)

e.Handled = true;

}

private void onlyByte\_TextChange(object sender, EventArgs e)

{

var textbox = sender as TextBox;

byte res;

if (!byte.TryParse(textbox.Text, out res) && textbox.Text != "")

{

MessageBox.Show("Значение должно находиться в диапазоне от 0 до 254.");

textbox.Text = "";

return;

}

}

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

{

var openFileDialog = new OpenFileDialog();

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

{

return;

}

inputBrowseTB.Text = openFileDialog.FileName;

}

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

{

var saveFileDialog = new SaveFileDialog();

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

{

return;

}

outputBrowseTB.Text = saveFileDialog.FileName;

}

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

{

onlyByte\_TextChange(sender, e);

if (subblockTB.Text == "")

return;

if (lenthBlockTB.Text == "")

{

MessageBox.Show("Задайте сначала длину блока.");

subblockTB.Text = "";

return;

}

if (int.Parse(subblockTB.Text) <= int.Parse(lenthBlockTB.Text)) return;

MessageBox.Show("Количество подблоков не может быть больше поливины длины блока.");

subblockTB.Text = "";

}

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

{

if (passTB.Text == "")

{

MessageBox.Show("Задайте пароль.");

return;

}

if (subblockTB.Text == "")

{

MessageBox.Show("Задайте количество подблоков.");

return;

}

if (lenthBlockTB.Text == "")

{

MessageBox.Show("Задайте длину блока.");

return;

}

if (lenthBlockTB.Text == "")

{

MessageBox.Show("Задайте колличество иттераций.");

return;

}

if (inputBrowseTB.Text == "")

{

MessageBox.Show("Задайте входной файл.");

return;

}

if (!File.Exists(inputBrowseTB.Text))

{

MessageBox.Show("Входного файла не существует.");

return;

}

if (outputBrowseTB.Text == "")

{

MessageBox.Show("Задайте выходной файл.");

return;

}

var lenthBlock = byte.Parse(lenthBlockTB.Text);

var round = byte.Parse(roundTB.Text);

var subblocks = byte.Parse(subblockTB.Text);

if (ecbRB.Checked)

crypter = new ECB(lenthBlock, subblocks, passTB.Text);

else

crypter = new FeistelNetwork(lenthBlock, subblocks, passTB.Text, round);

timer.Stop();

timer.Start();

groupBox1.Enabled = false;

groupBox2.Enabled = false;

groupBox3.Enabled = false;

groupBox4.Enabled = false;

groupBox5.Enabled = false;

groupBox6.Enabled = false;

oldTime = DateTime.Now;

crypter.DecryptComplitedEvent += crypter\_DecryptComplitedEvent;

crypter.EncryptComplitedEvent += crypter\_DecryptComplitedEvent;

if (encryptRB.Checked)

crypter.EncryptAsync(inputBrowseTB.Text, outputBrowseTB.Text);

else if (decryptRB.Checked)

crypter.DecryptAsync(inputBrowseTB.Text, outputBrowseTB.Text);

}

void crypter\_DecryptComplitedEvent(CryptResult cryptResult, string password)

{

if (cryptResult == CryptResult.MismatchValueParameters)

MessageBox.Show("Заданные параметры дешифрации не соответствуют параметрам при шифровании", "Ошибка");

timer.Stop();

rb.Maximum = 1000;

rb.Minimum = 0;

rb.Value = rb.Maximum;

if (cryptResult == CryptResult.Success)

SpeedLabel.Text = string.Format("Время выполнения: {0} секунд, {1:N} символов, средняя скорость: {2:N} симв/сек",

(DateTime.Now - oldTime).TotalSeconds, crypter.MaxValueProcess, crypter.MaxValueProcess / (DateTime.Now - oldTime).TotalSeconds);

else if (cryptResult == CryptResult.MismatchValueParameters)

SpeedLabel.Text = "Ошибка. Заданные параметры дешифрации не соответствуют параметрам при шифровании!";

else if (cryptResult == CryptResult.MismatchСhecksum)

SpeedLabel.Text = "Ошибка. Контрольная сумма не совпадает!";

groupBox1.Enabled = true;

groupBox2.Enabled = true;

groupBox3.Enabled = true;

groupBox4.Enabled = true;

groupBox5.Enabled = true;

groupBox6.Enabled = true;

}

private void ecbRB\_CheckedChanged(object sender, EventArgs e)

{

label2.Enabled = !ecbRB.Checked;

roundTB.Enabled = !ecbRB.Checked;

}

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

{

if (!File.Exists((sender as TextBox).Text))

{

MessageBox.Show("По данному пути файла не существует. Задайте другой путь.");

(sender as TextBox).Text = "";

return;

}

}

}

}