Министерство образования Республики Беларусь Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИНФОРМАТИКИ

И РАДИОЭЛЕКТРОНИКИ

КАФЕДРА ИНФОРМАТИКИ

Отчёт по лабораторной работе №8

По теме «Стеганографические методы»

Выполнила: студентка гр. 053501 Шурко Т.А.

Проверил: ассистент кафедры информатики Лещенко Е. А.

Минск 2023

**СОДЕРЖАНИЕ**

[Введение 3](#_Toc151674677)

[1 Теоретические сведения 4](#_Toc151674678)

[1.1 Стеганография. Быстрое преобразование Фурье 4](#_Toc151674679)

[1.2 Блок-схема 4](#_Toc151674680)

[2 Результат выполнения задачи 6](#_Toc151674681)

[Заключение 8](#_Toc151674682)

[Приложение А 9](#_Toc151674683)

# ВВЕДЕНИЕ

2. Шифрование — обратимое преобразование информации в целях сокрытия от неавторизованных лиц, с предоставлением, в это же время, авторизованным пользователям доступа к ней. Главным образом, шифрование служит задачей соблюдения конфиденциальности передаваемой информации. Важной особенностью любого алгоритма шифрования является использование ключа, который утверждает выбор конкретного преобразования из совокупности возможных для данного алгоритма.
3. Пользователи являются авторизованными, если они обладают определенным аутентичным ключом. Вся сложность и, собственно, задача шифрования состоит в том, как именно реализован этот процесс.
4. В целом, шифрование состоит из двух составляющих — зашифрование и расшифрование.
5. С помощью шифрования обеспечиваются три состояния безопасности информации. Первой является конфиденциальность: шифрование используется для скрытия информации от неавторизованных пользователей при передаче или при хранении. Второе — целостность. Шифрование используется для предотвращения изменения информации при передаче или хранении. Последнее это идентифицируемость. Шифрование используется для аутентификации источника информации и предотвращения отказа отправителя информации от того факта, что данные были отправлены именно им.
6. Для выполнения лабораторной работы 8 необходимо реализовать программное сокрытие (извлечения) текстового сообщения в (из) JPEG изображения на основе метода сокрытия в частотной области с использованием быстрого преобразования Фурье (БПФ).

# 1 ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

## 1.1 Стеганография. Быстрое преобразование Фурье

Стеганография — способ передачи или хранения [информации](https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%86%D0%B8%D1%8F) с учётом сохранения в тайне самого факта такой передачи (хранения).

Цифровая стеганография — направление классической стеганографии, основанное на сокрытии или внедрении дополнительной информации в цифровые объекты, вызывая при этом некоторые искажения этих объектов. Но, как правило, данные объекты являются мультимедиа-объектами (изображения, видео, аудио, текстуры 3D-объектов) и внесение искажений, которые находятся ниже порога чувствительности среднестатистического человека, не приводит к заметным изменениям этих объектов. Кроме того, в оцифрованных объектах, изначально имеющих аналоговую природу, всегда присутствует шум квантования; далее, при воспроизведении этих объектов появляется дополнительный аналоговый шум и нелинейные искажения аппаратуры, все это способствует большей незаметности сокрытой информации.

Быстрое преобразование Фурье (БПФ, FFT) — [алгоритм](https://ru.wikipedia.org/wiki/%D0%90%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC) ускоренного вычисления [дискретного преобразования Фурье](https://ru.wikipedia.org/wiki/%D0%94%D0%B8%D1%81%D0%BA%D1%80%D0%B5%D1%82%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%B5%D0%BE%D0%B1%D1%80%D0%B0%D0%B7%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5_%D0%A4%D1%83%D1%80%D1%8C%D0%B5), позволяющий получить результат за время, меньшее чем n2 (требуемого для прямого, поформульного вычисления). Иногда под быстрым преобразованием Фурье понимается один из алгоритмов, называемый алгоритмом прореживания по частоте — времени, имеющий сложность O(n\*log(n)).

## 1.2 Блок-схема

На рисунке 1.1 изображена блок-схема аудио стеганографии с использованием быстрого преобразования Фурье.

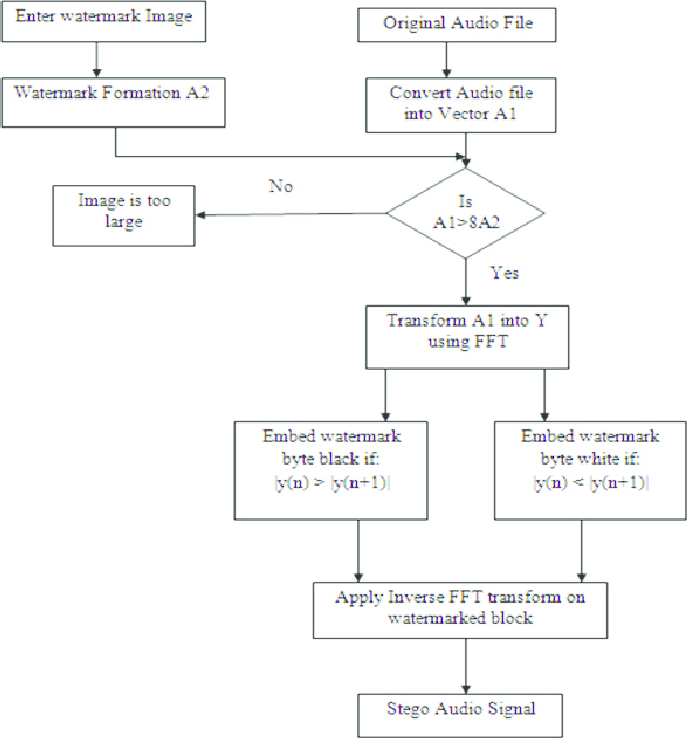


Рисунок 1.1 — Блок-схема стеганографии с использованием БПФ

# 2 РЕЗУЛЬТАТ ВЫПОЛНЕНИЯ ЗАДАЧИ

В результате выполнения задачи было реализовано программное сокрытие (извлечение) текстового сообщения в (из) JPEG изображения на основе метода сокрытия в частотной области с использованием быстрого преобразования Фурье (БПФ) на языке программирования С#. Текст, который необходимо скрыть в изображении, требуется преобразовать в строку, а также загрузить картинку, которая будет использоваться для преобразования. Текст исполняемой программы с использованием алгоритма БПФ в приложении А.

Для запуска программного продукта необходимо подключить классы Utilities, Steganography (Рисунок 2.1). Перед запуском программы необходимо создать файл с шифруемым текстом, а также загрузить картинку, которая будет использоваться для сокрытия сообщения. Метод «EncryptMessage» первым параметром принимает путь к картинке, а вторым строку с текстом для шифрования. Для расшифрования используется метод «DecryptMessage», в который необходимо передать только путь к картинке с зашифрованным текстом. Полученное значение и есть исходное сообщение.

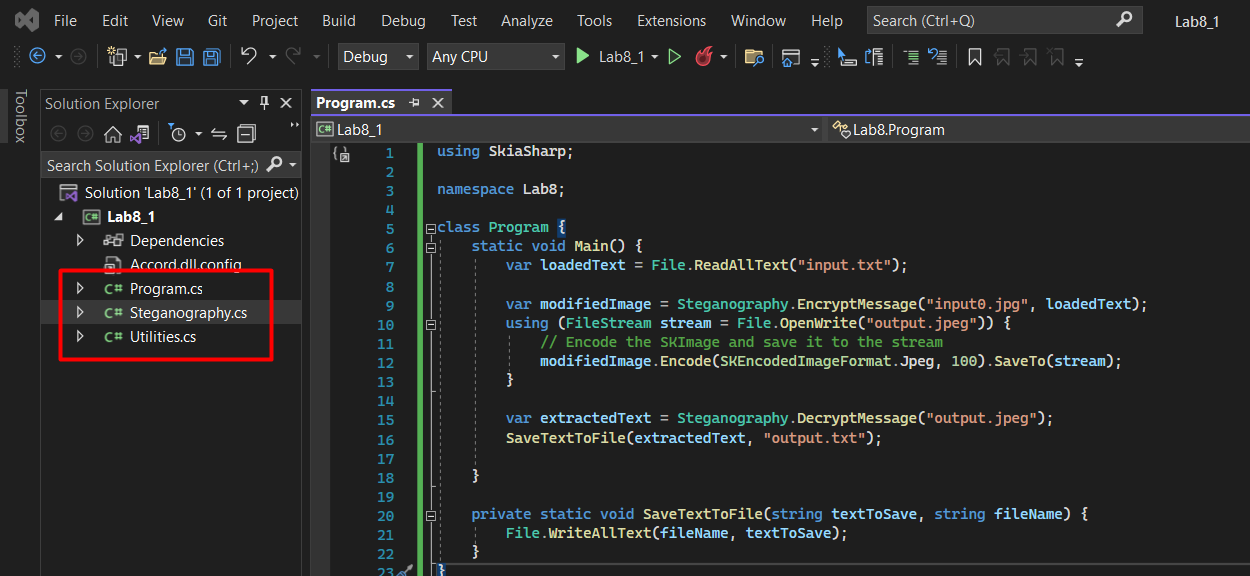


Рисунок 2.1 — Запускаемая программа

На рисунке 2.2 изображен результат работы программного продукта: справа изображена исходная картинка, а слева с зашифрованным в ней текстовым сообщением.

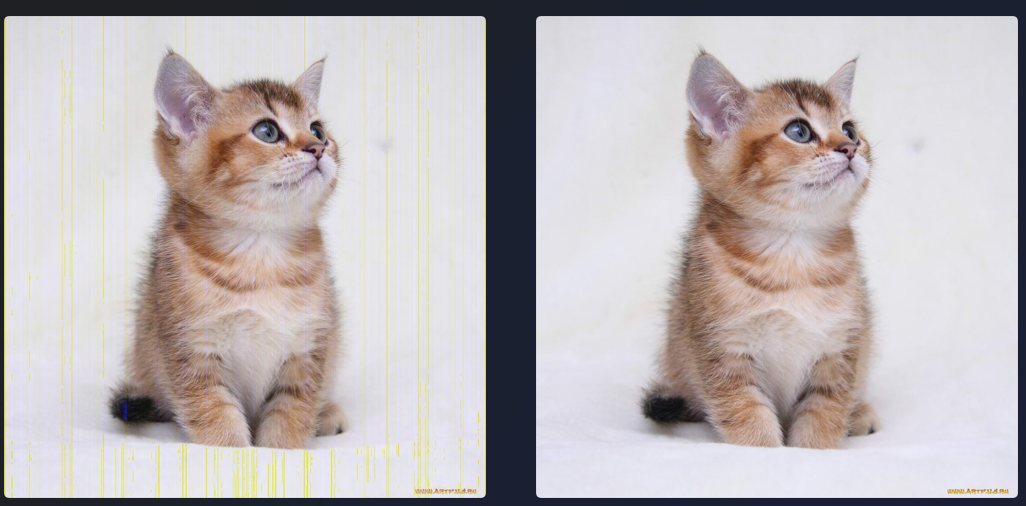


Рисунок 2.2 — Искаженное и исходное изображения

# ЗАКЛЮЧЕНИЕ

При выполнении лабораторной работы было изучено и реализовано программное сокрытие (извлечения) текстового сообщения в (из) JPEG изображения на основе метода сокрытия в частотной области с использованием быстрого преобразования Фурье (БПФ), рассмотрены и составлены блок-схемы, произведена их реализация на языке программирования C#, а также проверена работоспособность программного продукта.

Именно изображения чаще всего используются в качестве стегоконтейнеров.

1. Практическая значимость задачи защиты фотографий, картин, видео и прочей графической информации от незаконного копирования и распространения.
2. Большой информационный объём цифрового изображения, что позволяет скрывать цифровые водяные знаки (скрываемую информацию) большого объёма, либо делать больше [устойчивость](https://ru.wikipedia.org/wiki/%D0%A0%D0%BE%D0%B1%D0%B0%D1%81%D1%82%D0%BD%D0%BE%D1%81%D1%82%D1%8C) внедрения.
3. На момент встраивания цифровых водяных знаков известен конечный объём контейнера.
4. Нет ограничений на встраивание цифровых водяных знаков в режиме реального времени, как, например, в потоковом видео.
5. Многие изображения имеют области, имеющие шумовую структуру и хорошо подходящих для встраивания информации.
6. Криптоанализ таких систем начинается, обычно, с визуальной оценки, однако глаз не может различить незначительное изменение оттенка, вызванное записью информации в битовое представление цветов, что оставляет такой канал передачи информации вне подозрения.

# ПРИЛОЖЕНИЕ А

**(обязательное)**

**Код программы**

**Program.cs**

using SkiaSharp;

namespace Lab8;

class Program {

static void Main() {

var loadedText = File.ReadAllText("input.txt");

var modifiedImage = Steganography.EncryptMessage("input0.jpg", loadedText);

using (FileStream stream = File.OpenWrite("output.jpeg")) {

// Encode the SKImage and save it to the stream

modifiedImage.Encode(SKEncodedImageFormat.Jpeg, 100).SaveTo(stream);

}

var extractedText = Steganography.DecryptMessage("output.jpeg");

SaveTextToFile(extractedText, "output.txt");

}

private static void SaveTextToFile(string textToSave, string fileName) {

File.WriteAllText(fileName, textToSave);

}

}

**Steganography.cs**

using SkiaSharp;

using System.Numerics;

using System.Text;

using Accord.Math;

namespace Lab8;

public static class Steganography {

private static Complex[,]? \_data;

public static SKBitmap EncryptMessage(string imagePath, string message) {

using var image = SKBitmap.Decode(imagePath);

int messageLength = message.Length;

// Check message length

if (messageLength > (image.Width \* image.Height) / 8 - 1) {

throw new Exception("Слишком длинное сообщение для данного изображения.");

}

// Add end-marker to the text

message += '\0';

// Text -> binary

byte[] binaryMessage = Encoding.UTF8.GetBytes(message);

// FFT

Complex[,] fftData = ApplyFFT(image);

int currentX = 0;

int currentY = 0;

int bitIndex = 0;

int messageIndex = 0;

while (messageIndex < binaryMessage.Length) {

Complex pixelValue = fftData[currentX, currentY];

byte bit = (byte)((binaryMessage[messageIndex] >> bitIndex) & 1);

pixelValue = Utilities.EmbedBit(pixelValue, bit);

fftData[currentX, currentY] = pixelValue;

// Обнуление индекса бита при достижение 8

if (++bitIndex == 8) {

// Console.WriteLine(binaryMessage[messageIndex]);

bitIndex = 0;

messageIndex++;

}

// Переход к следующему пикселю с учетом шага

currentX += 1;

if (currentX >= image.Width) {

currentX = 0;

currentY += 1;

}

if (currentY >= image.Height) {

break; // Изображение закончилось

}

}

// ~FFT

SKBitmap encodedImage = InverseFFT(fftData);

return encodedImage;

}

public static string DecryptMessage(string modifiedImagePath) {

// Get modified image bitmap

var encodedImage = SKBitmap.Decode(modifiedImagePath);

// Extract coefficients from image

Complex[,] fftData = ApplyFFT(encodedImage);

int currentX = 0;

int currentY = 0;

int bitIndex = 0;

byte binaryChar = byte.MinValue;

List<byte> binaryChars = new();

bool endFound = false;

while (currentY < encodedImage.Height && !endFound) {

Complex pixelValue = fftData[currentX, currentY];

byte bit = Utilities.ExtractBit(pixelValue);

if (bitIndex == 0) {

binaryChar = bit;

} else {

binaryChar |= (byte)(bit << bitIndex);

}

if (++bitIndex == 8) {

bitIndex = 0;

if (binaryChar != '\0') {

binaryChars.Add(binaryChar);

} else {

// Marker found

endFound = true;

}

binaryChar = byte.MinValue;

}

// Next pixel

currentX += 1;

if (currentX >= encodedImage.Width) {

currentX = 0;

currentY += 1;

}

}

// Text from bytes

var extractedMessage = Encoding.UTF8.GetString(binaryChars.ToArray());

return endFound ? extractedMessage : "ERROR\_EXTRACTING";

}

private static Complex[,] ApplyFFT(SKBitmap image) {

int width = image.Width;

int height = image.Height;

Complex[,] fftData = new Complex[width, height];

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

for (int y = 0; y < height; y++) {

SKColor pixel = image.GetPixel(x, y);

byte red = pixel.Red;

byte green = pixel.Green;

byte blue = pixel.Blue;

int combined = (red << 16) | (green << 8) | blue;

fftData[x, y] = new Complex(combined, 0);

}

}

FourierTransform.FFT2(fftData, FourierTransform.Direction.Forward);

if (\_data != null) {

Utilities.Copy(fftData, \_data);

}

return fftData;

}

private static SKBitmap InverseFFT(Complex[,] fftData) {

int width = fftData.GetLength(0);

int height = fftData.GetLength(1);

\_data = new Complex[width, height];

Utilities.Copy(\_data, fftData);

FourierTransform.FFT2(fftData, FourierTransform.Direction.Backward);

SKBitmap resultImage = new SKBitmap(width, height);

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

for (int y = 0; y < height; y++) {

Complex pixelValue = fftData[x, y];

int combined = (int)pixelValue.Real;

byte red = (byte)((combined >> 16) & 0xFF);

byte green = (byte)((combined >> 8) & 0xFF);

byte blue = (byte)(combined & 0xFF);

SKColor pixel = new SKColor(red, green, blue);

resultImage.SetPixel(x, y, pixel);

}

}

return resultImage;

}

}

**Utilities.cs**

using System.Numerics;

namespace Lab8;

public static class Utilities {

internal static Complex EmbedBit(Complex pixelValue, byte bit) {

double real = pixelValue.Real;

if (bit == 0) {

real = Math.Floor(real / 2) \* 2;

} else {

real = Math.Floor(real / 2) \* 2 + 1;

}

return new Complex(real, pixelValue.Imaginary);

}

internal static byte ExtractBit(Complex pixelValue) {

double real = pixelValue.Real;

// Извлекаем младший бит из реальной и мнимой частей

byte bitReal = (byte)((int)real & 1);

return bitReal;

}

public static void Copy(Complex[,] matrixTo, Complex[,] matrixFrom) {

for (int i = 0; i < matrixTo.GetLength(0); i++) {

for (int j = 0; j < matrixTo.GetLength(1); j++) {

matrixTo[i, j] = matrixFrom[i, j];

}

}

}

}