**СОДЕРЖАНИЕ**

[ВВЕДЕНИЕ 3](#_Toc69950578)

[1. РАЗРАБОТКА ЭСКИЗНОГО И ТЕХНИЧЕСКОГО ПРОЕКТОВ ПРОГРАММЫ «STEGOIMAGE» 4](#_Toc69950579)

[1.1. Назначение и область применения программы «StegoImage» 4](#_Toc69950580)

[1.2. Технические характеристики 4](#_Toc69950581)

[1.2.1. Постановка задачи 4](#_Toc69950582)

[1.2.2. Описание работы метода НЗБ 7](#_Toc69950583)

[1.2.3. Организация входных и выходных данных 8](#_Toc69950584)

[1.2.4. Выбор средств разработки программы 8](#_Toc69950585)

[2. РАЗРАБОТКА РАБОЧЕГО ПРОЕКТА ПРОГРАММЫ ДЛЯ СОКРЫТИЯ ИНФОРМАЦИИ В ГРАФИЧЕСКИХ ФАЙЛАХ 9](#_Toc69950586)

[2.1. Конструирование пользовательского интерфейса 9](#_Toc69950587)

[2.1.1. Визуальное проектирование программы 9](#_Toc69950588)

[2.2. Описание кода программы 10](#_Toc69950589)

[2.3. Спецификация программы 18](#_Toc69950590)

[2.4. Описание программы и рекомендации к использованию 18](#_Toc69950591)

[2.5. Тестирование программы 18](#_Toc69950592)

[ЗАКЛЮЧЕНИЕ 24](#_Toc69950593)

[СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ 25](#_Toc69950594)

[ПРИЛОЖЕНИЕ 26](#_Toc69950595)

# **ВВЕДЕНИЕ**

С развитием глобальных компьютерных сетей появилась возможность передавать моментально передавать электронные файлы: документы, архивы, графические файлы и так далее. Однако появилась проблемы с незаконным копированием и распространением. Особенно это обострилось с появлением глобальной сети Интернет: любой мог присвоить себе тот или иной документ, файл, статью. Прямое указание автора в документах, графических рисунках не всегда спасало от плагиата – данные просто редактировались. Необходимо было скрывать информацию об авторе внутри самого файла – сделать недоступным для редактирования простым пользователем.

Одними из способов сокрытия информации об авторстве являются цифровые водяные знаки. Разработка ЦВЗ – одно из перспективных направлений развития стеганографии. Помимо ЦВЗ, существует и другие весьма полезные алгоритмы стеганографии. Методы цифровой стеганографии также используются для скрытой связи, скрытой аннотации.

Целью курсовой работой изучение стенографических методов сокрытия информации в графических. Исходя из цели курсовой можно выделить следующие задачи курсовой работы:

1. Изучение информационных источников по теме изучения.
2. Выделение основных методов.
3. Провести анализ метода сокрытия информации.
4. Спроектировать приложение с функциями стеганографии.
5. Протестировать работу программы.

Курсовая работа состоит из введения, двух глав, заключение, списка и программы к курсовой.

В первой работе описывается область применения конечного программы, методы стеганографии в графических файлах.

Во второй главе рассматриваются функции и методы программы, рекомендация к использованию. Так же описан процесс тестирования.

# **РАЗРАБОТКА ЭСКИЗНОГО И ТЕХНИЧЕСКОГО ПРОЕКТОВ ПРОГРАММЫ «STEGOIMAGE»**

## **Назначение и область применения программы «StegoImage»**

Разрабатываемое программа «StegoImage» может использоваться как учебная модель для изучения стеганографии с использованием графических контейнеров: особенности скрытия и его уязвимости. Программа наглядно может демонстрировать как переполнение контейнера, так его грамотное использования.

Вне учебных в целях программа может сохранять необходимую для пользователя информацию, например, авторство.

## **Технические характеристики**

### **Постановка задачи**

Целью курсовой работы является изучение методов скрытия информации. При этом одной из главных задач является разработка программы по одному из методов – НЗБ (наименьше значащий бит), или LSB- метод.

Стоит выделить и определить термины используемы в цифровой стеганографии.

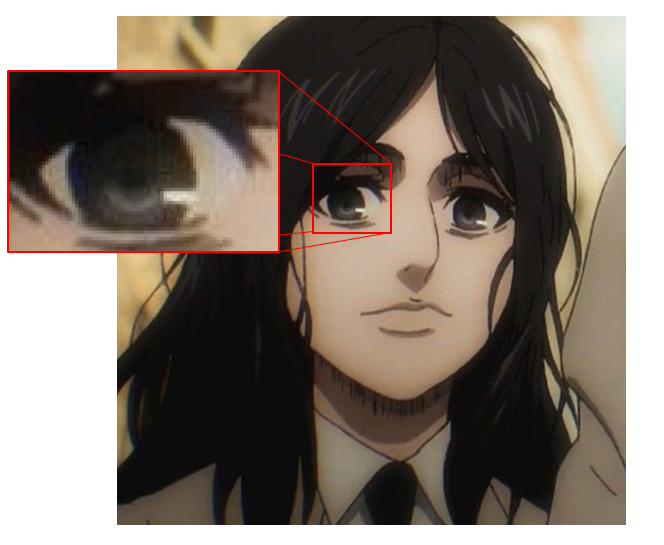
Стеганография (от греч. «тайнопись») – наука скрытой передачи информации путем сокрытия самого факта скрытой передачи. В цифровой стеганографии используется файлы, а точнее их контейнеры.

Под контейнером следует понимать последовательность бит, предназначенные для скрытия сообщения . Существует два вида контейнера:

* потоковый – неограниченная длина контейнера, при этом сложно определить начало сообщения;
* ограниченный – длина контейнера фиксирована, и его переполнение может привести к раскрытию факта передачи;

В стеганографии растровых изображений используется визуальный контейнер. Сообщение скрывается в таком контейнере при помощи изменения параметров яркости, RGB – каналов пикселя, палитры цветов.

Растровое изображение представляет собой набор упорядоченных пикселей размером m\*n (m – высота в пикселях, n - длина). На рисунке 1 предоставлен пример самого изображения и его увеличенного фрагмента – пиксели изображения.



*Рисунок 1 – Пиксели изображения*

Каждый пиксель характеризуется настройкой цветовой модели. Наиболее распространённой моделями являются RGB и ARGB. Обе модели имеют три цветовых канала – red, green и blue, за исключением второй модели – она имеет дополнительный канала – alpha, который отвечает за прозрачность каждого пикселя, в основном ARGB используется в изображениях PNG формата.

Структуру изображения можно представить следующим образом (рисунок 2): заголовок файла, заголовок изображения, палитра, и пикслели, разбитые на цветовые каналы.



*Рисунок 2 – Общая структура изображения*

Методы скрытия информации бывают следующие:

* НЗБ (наименьше значащий бит, LSB) – простой метод стеганографии, но при этом у этого метода высокая защита от стегоанализа, также этот метод используется и в звуковых файлах. Замена одного бита или нескольких (не более 3) не сказывается на изображении, то есть говорить об искажении файла не имеет смысла;
* Замена цветовой палитры – палитра, или число триад байт, описывает цвет пикселя. За палитрой идет массив байт, каждый из который представляется собой номер цвета в палитре.
* Сортировка палитры – цвета изображения выстраивается так, чтобы цвета с соседними номерами минимально отличались между собой. После в младшие биты заносятся биты сообщения. В отличии от НЗБ меняется не сам цвет, а его номер в палитре.
* Сокрытие в JPEG – это метод можно разделить на несколько дополнительных методов, но все они меняют не само изображение, а коэффициенты преобразования пикселей при сжатии.

### **Описание работы метода НЗБ**

Алгоритм делится на несколько этапов:

1. Представляем сообщение как массив битов.

2. Выбор изображения-контейнера подходящего размера. Иначе часть сообщения может быть утеряна.

3. Выбираем пиксель и разбиваем его на цветовые каналы.

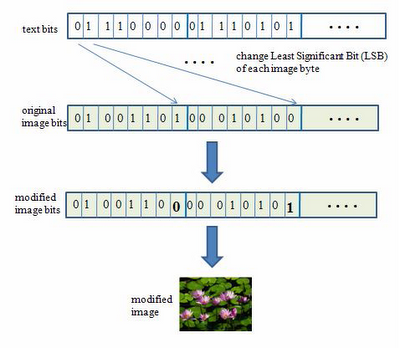
4. В необходимые цветовые каналы встраиваем нужное количество битов сообщения (по умолчанию 1 бит на канал). Биты встраиваются в младшие биты канала.

5. Собираем пиксель обратно.

6. Берем следующий писклей.

7. Если все биты сообщения были вложены, или все пиксели изображения были изменены, сохраняем стегоизображение.

На рисунке 3 представлена общая схема алгоритма. Первые 8 бит – красный канал, Вторые – зеленый. В схеме заменяется только один бит на канале, что гарантирует не возможность искажения изображения



*Рисунок 3 – Схема НЗБ метода*

### **Организация входных и выходных данных**

Входными данными являются: любое изображение форматов PNG, JPEG, BMP, сообщение в TXT файле, число бит (1,2, 4 и 8), список кодируемых каналов RGB – красный, зеленый, голубой.

Изображение используется как контейнер сообщения. Каналы и количество бит – настройка скрытия сообщения: в каких каналах кодируется информация и сколько бит налаживается на используемый канал.

### **Выбор средств разработки программы**

Для реализации программной модели использовать среда разработки Visual Studio 2019 с Framework 4.8.

Причинами такого выбора являются:

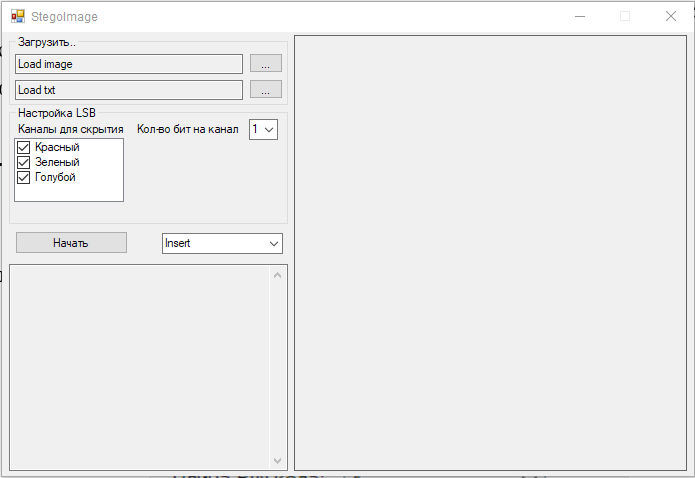
1. Простое построение окон для Windows приложений.
2. Поддержка кроссплатформенности.
3. Поддержка большого числа кодировок, что гарантирует точное отображение визуальной части программы в разных версиях Windows ОС.

Выбранный язык C# позволяет использовать быстрые и безопасные конструкции. При этом для это языка достаточно документации на официальном сайте Microsoft.

# **РАЗРАБОТКА РАБОЧЕГО ПРОЕКТА ПРОГРАММЫ ДЛЯ СОКРЫТИЯ ИНФОРМАЦИИ В ГРАФИЧЕСКИХ ФАЙЛАХ**

## **Конструирование пользовательского интерфейса**

Интерфейс достаточно прост в понимании. Ниже (рисунок 4) предоставлен сам интерфейс программы.



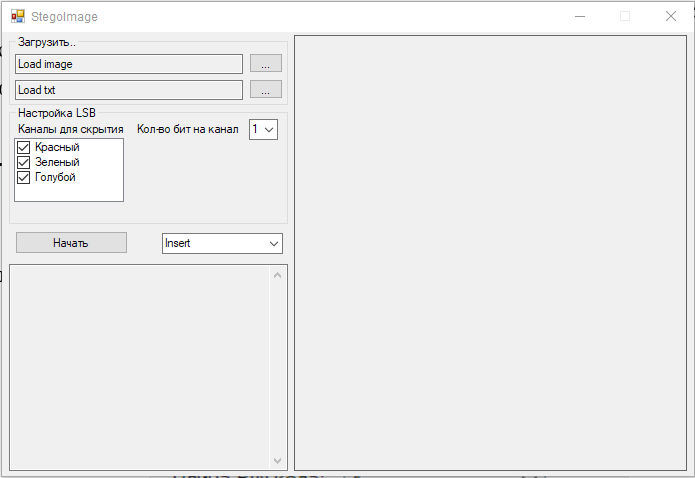
*Рисунок 4 - Внешний вид работающего приложения*

Форма разделена на несколько областей: область загрузки файлов, настройка алгоритма, и демонстрация данных (PictureBox и TextBox).

### **Визуальное проектирование программы**

loadImage

imgBox



comboBoxmode

comboBox1

loadText

startButton

checkedListBox1

txtBox

contentBox

picBox

*Рисунок 5 - Визуальные и невизуальные компоненты главной формы программы*

*Таблица 1*

*Назначение компонентов главной формы программы*

| ***№ п/п*** | ***Компонент*** | ***Назначение*** |
| --- | --- | --- |
| 1 | MainForm | Главная форма приложения |
| 2 | imgBox | Путь к изображению |
| 3 | txtBox | Путь к текстовому файлу |
| 4 | loadImage | Загрузка изображения. Вызывает компонент openFileDialog |
| 5 | loadText | Загрузка файла. Вызывает компонент openFileDialog |
| 6 | comboBox1 | Выбор количества битов на канал |
| 7 | comboBoxmode | Выбор режима (Вставка сообщения/Раскрытие сообщения) |
| 8 | startButton | Запуск алгоритма |
| 9 | checkedListBox1 | Выбор каналов RGB |
| 10 | contentBox | Содержимое файла или изображения |

## **Описание кода программы**

При работе программы использованы следующие процедуры:

*Таблица 2*

*Используемые процедуры и функции в главной форме приложения*

| ***Процедура*** | ***Назначение*** |
| --- | --- |
| loadImage\_Click(object sender, EventArgs e) | Обработка события нажатия loadImage. Загрузка изображения. |
| loadText\_Click(object sender, EventArgs e) | Обработка события нажатия loadText. Загрузка файла. |
| ShowImage(Bitmap cur) | Отображение изображения на главной форме. |

| ShowTxt(string path) | Отображение текста на главной форме. |
| --- | --- |
| startButton\_Click(object sender, EventArgs e) | Запуск НЗБ метода. |
| SetValue(bool d) | Установка настройки LSB по умолчанию |
| StegoImage\_Load(object sender, EventArgs e) | Запуск главной формы |
| comboBoxmode\_SelectedIndexChanged(object sender, EventArgs e) | Смена режима. Обработка события. |

*Таблица 3*

*Используемые процедуры и функции класса LSB.cs*

| ***Процедура*** | ***Назначение*** |
| --- | --- |
| LSB(Bitmap image, string message, bool[] arr, int y) | Передача данных в текущий класс (режим кодирования) |
| LSB(Bitmap image, bool[] arr, int y) | Передача данных в текущий класс (режим декодирования) |
| GetBitsOfMessage(string message) | получение битового массива сообщения |
| insertTextToImage() | LSB - кодирование |
| ExtractMessage(Bitmap bmpCry) | Декодирования всего изображения |
| GetBitfromPixel(int currentPix, int i) | Получение бита из канала пикселя |
| SetPixel(bool currentBit, int pixel, int i) | Вставка бита в канал |

Рассмотрим код НЗБ более подробнее.

В режиме вставке в класс передаются следующие данные: булевый массив используемых каналов ( значение true – канал используется), изображение, сообщение и количество бит. Также вызывается метод перевода сообщения в битовое представление.

public LSB(Bitmap image, string message, bool[] arr, int y) //передача данных в текущий класс (режим кодирования)

{

this.bmp = new Bitmap(image);

numberOfSymbols = message.Length;

arr.CopyTo(cRGB, 0);

bitOnC = y;

GetBitsOfMessage(message);}

Сообщение переводится в формат битового массива:

public void GetBitsOfMessage(string message)

{ byte[] temp = Encoding.Default.GetBytes(message);

bits = new BitArray(temp); }

С главной формы уже вызывается уже метод кодирования insertTextToImage(). В ней каждый пиксель разбивается на каналы. После вызывается метод вставки бита. Как только на канал вставились необходимое количество бит на канал переходим к следующему. Как только все необходимые каналы пикселя были заполнены, пиксель формируется снова и сохраняться в изображении. Далее преступаем к следующему пикселю.

public Bitmap insertTextToImage() // LSB -кодировние

{

int R, G, B;

Bitmap bitmap = (Bitmap)bmp.Clone();//Клонируем изображение

int bitCount = 0; //счетчики битов сообщения и пикселей изображения

int allbit = 0;

bool getAllNeedPixel = false; // булевая переменная, необходимая для выхода из внешнего цикла

//С помощью циклов проходим по изображению (по пикселям)

for (int i = 0; i < bmp.Height; i++)

{

if (getAllNeedPixel) break; //Если сообщение уже закодированно, то прекращаем проход по изображению

for (int j = 0; j < bmp.Width; j++)

{

if (allbit != bits.Count)

{

Color pixel = bmp.GetPixel(j, i); // получем текущий пиксель по координате

R = pixel.R;

G = pixel.G;

B = pixel.B;

if (cRGB[0])

{

for (int l = 0; l < bitOnC; l++)

{

if (bitCount == bits.Count)

{

getAllNeedPixel = true;

break;

}

R = SetPixel(bits[bitCount], R, l);

bitCount++; // увеличиваем счетчик - получаем бит следующий сообщения

allbit++;

}

}

if (cRGB[1])

{

for (int l = 0; l < bitOnC; l++)

{

if (bitCount == bits.Count)

{

getAllNeedPixel = true;

break;

}

G = SetPixel(bits[bitCount], G, l);

bitCount++; // увеличиваем счетчик - получаем бит следующий сообщения

allbit++;

}

}

if (cRGB[2])

{

for (int l = 0; l < bitOnC; l++)

{

if (bitCount == bits.Count)

{

getAllNeedPixel = true;

break;

}

B = SetPixel(bits[bitCount], B, l);

bitCount++; // увеличиваем счетчик - получаем бит следующий сообщения

allbit++;

}

}

bitmap.SetPixel(j, i, Color.FromArgb(pixel.A, R, G, B));

}

else // выход из цикла

{

getAllNeedPixel = true;

break;

}

}

}

return bitmap; // возвращаем результат

}

Сама вставка проста – получаем индекс бита для замены, сам бит сообщения и битовый массив канала.

private int SetPixel(bool currentBit, int pixel, int i)

{

BitArray n = new BitArray(new int[] { pixel }); // битовое представление пикселя

// так как битовый массив хранит биты в дргугом порядке в отличии от привычного порядка

// Пример число 2 - 00010 (привычный порядок), в битовом массиве - реверс т.е. 01000

// последний бит - первый бит в массиве

if (currentBit)// если значение бита = 1

n[i] = true;

else

n[i] = false;

int[] y = new int[1]; //получаем значение пикселя в привычном виде (натурального числа)

n.CopyTo(y, 0);

return y[0];

}

Декодирование изображения начинается с получения данных скрытия: количество бит на канал, изображение и массив использованных каналов.

public LSB(Bitmap image, bool[] arr, int y)//передача данных в текущий класс (режим декодирования)

{

this.bmp = new Bitmap(image);

arr.CopyTo(cRGB, 0);

bitOnC = y;

}

Самое декодирование – разбиваем пиксель, вытягиваем из каждого необходимого канала нужное количество бит (используется метод GetBitfromPixel ). При этом декодируется все изображение.

public string ExtractMessage(Bitmap bmpCry) // Декодировка всего изображения

{

string result = string.Empty;

int bitCount = 0, bit; // счётчик и переменая для сохранение каждого бита отдельно (временная переменная)

List<bool> messBits = new List<bool>();//Список булевых значений, здесь будут храниться значения битов

for (int i = 0; i < bmp.Height; i++)

{

for (int j = 0; j < bmp.Width; j++)

{

Color pixel = bmp.GetPixel(j, i);// получение пикселя

if (cRGB[0])

for (int l = 0; l < bitOnC; l++)

{

bit = GetBitfromPixel(pixel.R, l);// вызов метода для получение последнего бита

messBits.Add(Convert.ToBoolean(bit));//запоминаем бит

bitCount++;

}

if (cRGB[1])

for (int l = 0; l < bitOnC; l++)

{

bit = GetBitfromPixel(pixel.G, l);// вызов метода для получение последнего бита

messBits.Add(Convert.ToBoolean(bit));//запоминаем бит

bitCount++;

}

if (cRGB[2])

for (int l = 0; l < bitOnC; l++)

{

bit = GetBitfromPixel(pixel.B, l);// вызов метода для получение последнего бита

messBits.Add(Convert.ToBoolean(bit));//запоминаем бит

bitCount++;}}}

BitArray n = new BitArray(messBits.ToArray()); // булевый список в битовый массив

byte[] n2 = new byte[n.Length]; //массив байтов

n.CopyTo(n2, 0); //перевод в байты

result = Encoding.Default.GetString(n2);// байты в строку

return result;

}

GetBitfromPixel – зеркальное отражение метода SetPixel, но на выходе получаем бит сообщения.

private int GetBitfromPixel(int currentPix, int i)

{

BitArray n = new BitArray(new int[] { currentPix });// битовое представление пикселя

// так как битовый массив хранит биты в дргугом порядке в отличии от привычного порядка

// Пример число 2 - 00010 (привычный порядок), в битовом массиве - реверс т.е. 01000

// последний бит - первый бит в массиве

return Convert.ToInt32(n[i]);

}

## **Спецификация программы**

Состав проекта

|  |  |
| --- | --- |
| *Наименование* | *Обозначение* |
| StegoImage.exe | Исполняемый файл приложения (Windows-программа). |

## **Описание программы и рекомендации к использованию**

Для быстродействия программы используется изображения небольшого формата. Также следует учитывать размер сообщения и изображения.

Загружаются два файла: изображение и txt-файл. В настройках определяем каналы, количество бит на канал.

Далее выбираем необходимый режим работы и запускаем:

* Insert – кодирование изображения;
* Extract – декодирование изображения;

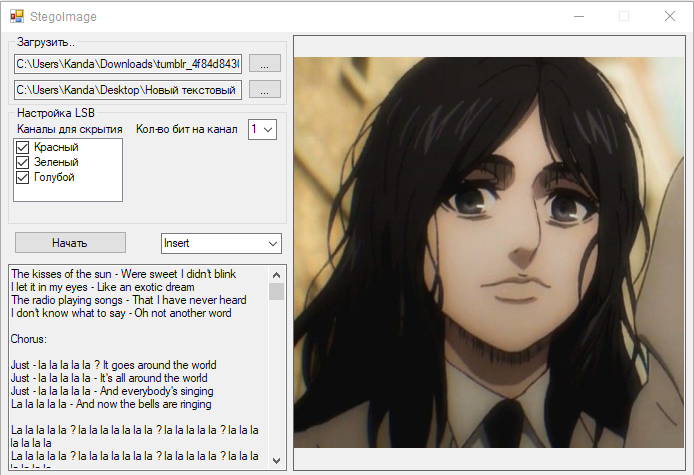
## **Тестирование программы**

Программа тестировалась на нескольких изображениях при разных значениях настройки метода.

Тестирование проводилось в три этапа:

1. Общая работоспособность;
2. Тестирование самого метода: искажение изображения, декодирования.
3. Оценка результатов тестирования.

Загружаем изображение (желательно цветное) и файла (рисунок 6). Начальные значения следующие: используем все каналы и кодируем по одному биту.



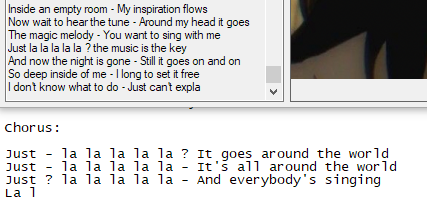
*Рисунок 6 – Начальная установка.*

Запускаем кодирование и ждем его завершения, изображение следует сохранить. В результате получим следующие: на рисунке 7 представлены два изображения (слева оригинал, справа стегоизображение).



*Рисунок 7 – Оригинал и результат кодирование всех каналов с одним битом.*

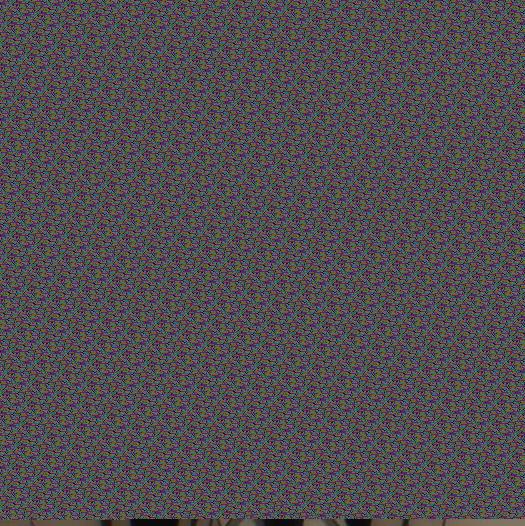
Входное и выходное изображение имеет расширение JPEG. Ощутимые искажения изображения отсутствуют. Однако это не говорит о его пустоте. Запустим обратный алгоритм и получим исходный текст (рисунок 8). Однако была утеряна часть информации – текст превышает размер контейнеров, по этому произошла потеря.



*Рисунок 8 – Отсутствует последний куплет.*

Протестируем теперь при следующих значений бит на канал: 2, 4 и 8 бит. На рисунке 9 представлена шкала искажения изображения: оригинал, 1 бит, 2 бита, 4 бит, 8 бит.

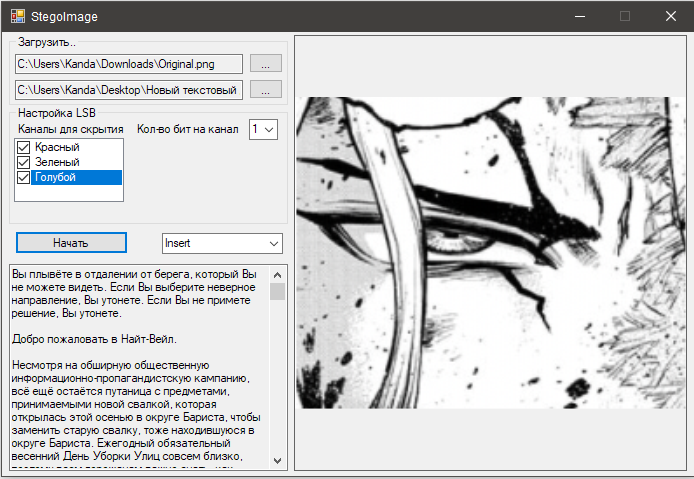




*Рисунок 9 –Шкала искажения.*

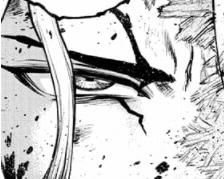
Первые три изображения не отличаются между собой. При 4 бит искажение ярко выражено – тени, оттенки цветов и переходы между ними видны, при этом сохранен образ. Последние изображение результат при изменение всех битов каждого канала. Значение при 4 и 8 бит говорит о переполнение контейнера данного изображения, что нарушает основной принцип стеганографии.

Теперь протестируем на черно-белом небольшом изображении и сократим сообщение (рисунок 10).



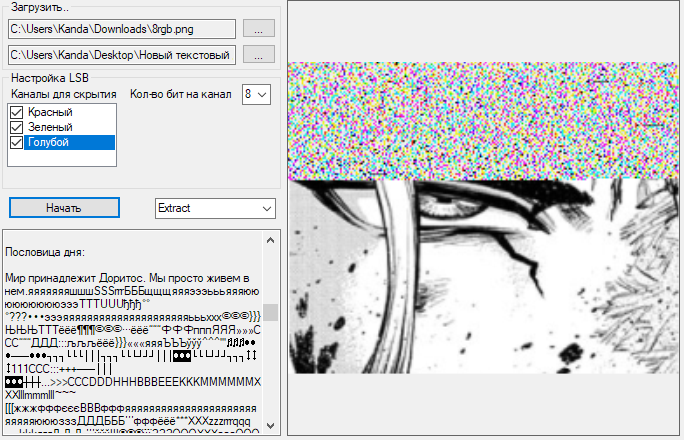
*Рисунок 10 – Новые данные.*

Тест всех каналов при 1, 2, 4, 8 бит. Получим следующую шкалу искажения на рисунке 11.



*Рисунок 11 – Искажения черного-белого изображения при всех каналах.*

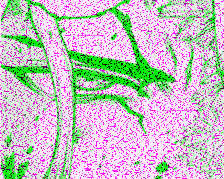
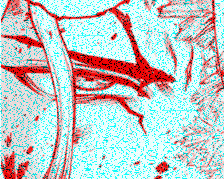
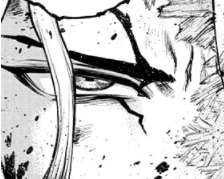
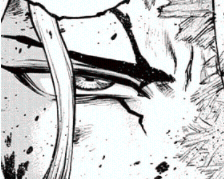
Первые три изображения не отличаются, при 4 битах заметна серость и падение качества не связанное с сжатие JPEG (расширение картинки – PNG). что подтверждает теорию о трех битах: для заметной передачи информации методом НЗБ следует использовать контейнер менее 3 бит.

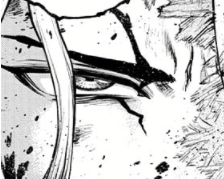


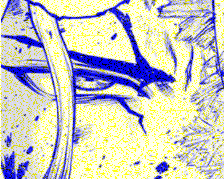
*Рисунок 12 – Искажения черного-белого изображения при всех каналах.*

При 8 бит шум покрывает часть изображение, сообщение заняло эту часть. Это подтверждает декодирование (рисунок 12). После сообщения идут нечитабельные символы.

Следующий тест – 2, 4 и 8 бит, каждый канал отдельно.







*Рисунок 13 – Тест каналов*

На рисунке 13 сверху вниз предоставлены «шифрованные» RGB каналы (красный, зеленый, и голубой). При четырех бита видны оттенки, если сравнивать с результатами 2 бит. Последний столбец – ярко выраженное искажения канала;

Скрытие в каналах является модификацией классического НЗБ-метода, что осложняет раскрытие сообщения.

В результате теста программы мы убедились в следующем:

* Программа работоспособна;
* От настроек программы, размера текстового файла, и изображения зависит полнота сообщения конечного;
* Скрывать следует в 1 – 2 младших битах;
* Чем тоньше настройка при скрытии, тем выше защита от стегоанализа – сложность подбора начальных данных;

# ЗАКЛЮЧЕНИЕ

Целью курсовой работы являлось изучение стеганографических алгоритмов скрытия информации в изображении.. Программная модель выполняет основную функцию – скрытие информации. Цель была достигнута.

При написании курсовой работы были выполнены следующие задачи:

1. Изучили информационные источники по теме изучения.
2. Выделили основные методы.
3. Провели анализ метода сокрытия информации.
4. Спроектировали программу с функцией стеганографии.
5. Протестировали работу программы.

Данная программа имеет возможность развития: добавление дополнительных функций для защиты (пароль и шифрование, алгоритмы сжатия).

Практическая значимость данной программы – учебная модель для изучения метода НЗБ.

# СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

1. J. Fridrich, R. Du, and L. Meng, “Steganalysis of LSB Encoding in Color Images”, ICME 2000, New York City.
2. J. Friedrich, G. Miroslav, R. Du. Reliable Detection of LSB Steganography in Color and Grayscale Images. Binghampton, New York: SUNY, 2001.
3. В.Г. Грибунин, И. Н. Оков, И. В. Туринцев. Цифровая стеганография. – М., Солон-пресс. 2002.

4. Национальная электронная библиотека им. Н. Э. Баумана – «Стеганография изображения методом LSB»- (http://surl.li/rlhl)

5. Habr.com – «Прячем текст в Bitmap»- (https://habr.com/ru/post/115673/)

6. Habr.com – «Скрывать не скрывая. Еще раз о LSB-стеганографии, хи-квадрате и… сингулярности?»- (https://habr.com/ru/post/422593/)

7. Spy-soft.net – «Цифровая стеганография: Программы и другие способы реализации»- (https://spy-soft.net/cifrovaya-steganografiya-sposoby-realizacii/)

# ПРИЛОЖЕНИЕ

**Структура диска:**