Учреждение образования

«БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

**Исследование стеганографического метода на основе преобразования наименее значащих битов.**

Студент: Рубашек А. А.

ФИТ 3 курс 5 группа

Преподаватель:

Савельева Маргарита Геннадьевна

1. **Цель работы**

Изучение стеганографического метода встраивания/извлечения тайной информации с использованием электронного файла-контейнера на основе преобразования наименее значащих битов (НЗБ), приобретение практических навыков программной реализации данного метода.

1. **Задания**

Разработать собственное приложение, в котором должен быть реализован метод НЗБ. При этом:

* выбор файла-контейнера – по согласованию с преподавателем;
* реализовать два варианта осаждаемого/извлекаемого сообщения:
* собственные фамилия, имя и отчество;
* текстовая часть отчета по одной из выполненных лабораторных работ;
* реализовать два метода (на собственный выбор) размещения битового потока осаждаемого сообщения по содержимому контейнера;
* сформировать цветовые матрицы, отображающие каждый задействованный для осаждения уровень младших значащих битов контейнера;
* выполнить визуальный анализ (с привлечением коллег в качестве экспертов) стеганоконтейнеров с различным внутренним содержанием; сделать выводы на основе выполненного анализа.

3. Результаты выполнения работы оформить в виде отчета по установленным правилам.

1. **Ход работы**

В ходе работы для осаждения сообщения было разработано два метода: EmbedPixelPermutation и EmbedLSB. Первый метод реализует алгоритм псевдослучайной перестановки. Для извлечения сообщения из стеганоконтейнера была разработана функция ExtractPixelPermutation, код которой представлен. Код обоих методов представлен в Приложении 1. В качестве файла-контейнера был выбран следующий рисунок 3.1.



Рисунок 3.1 – Выбранный стеганоконтейнер

Входные данные и результат работы представлены на рисунках 3.2 – 3.3.

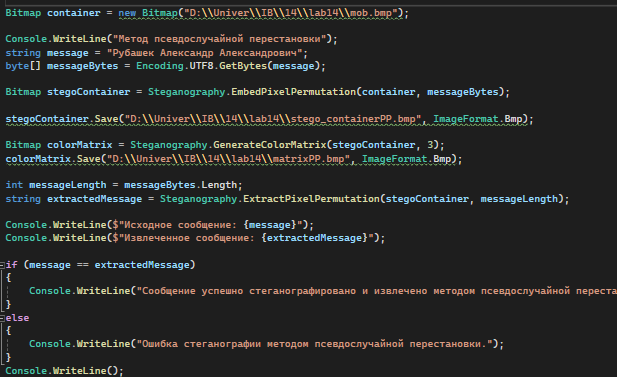


Рисунок 3.2 – Входные данные

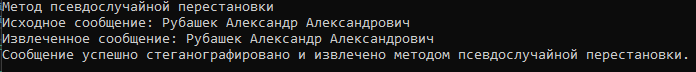


Рисунок 3.3 – Результат работы

Также, с помощью метода GenerateColorMatrix была сформирована цветовая матрица, отображающая каждый задействованный для осаждения уровень младших значащих битов контейнера. Сам метод и результат его работы представлены на рисунках 3.4 – 3.5.

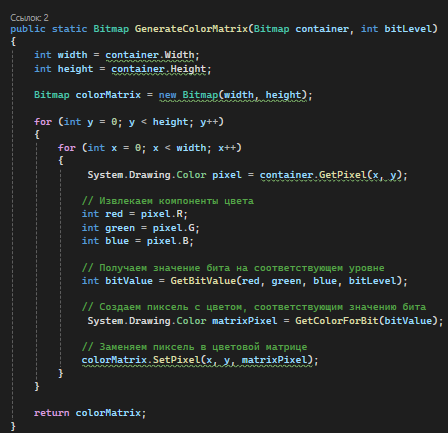


Рисунок 3.4 – Метод GenerateColorMatrix

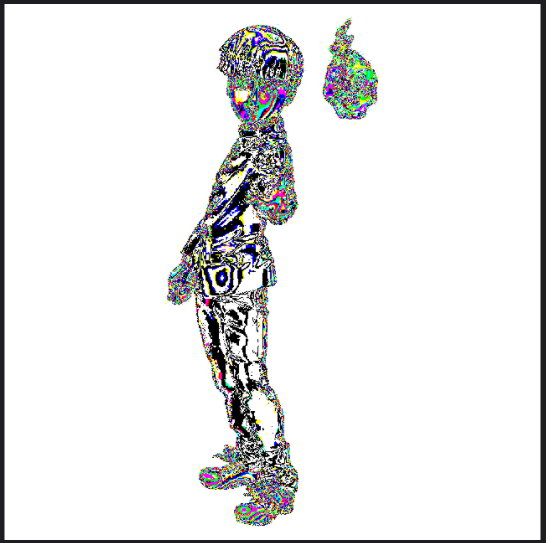


Рисунок 3.5 – Результат работы метода GenerateColorMatrix

Для реализации второго метода размещения битового потока осаждаемого сообщения по содержимому контейнера был выбран алгоритм LSB. Его реализация представлена в методе EmbedLSB в Приложении 2. Извлечения сообщения из стеганоконтейнера была разработана функция ExtractLSB, код которой представлен на рисунке в Приложении 2.

Входные данные, результат работы и цветовая матрица представлены на рисунках 3.6 – 3.8.

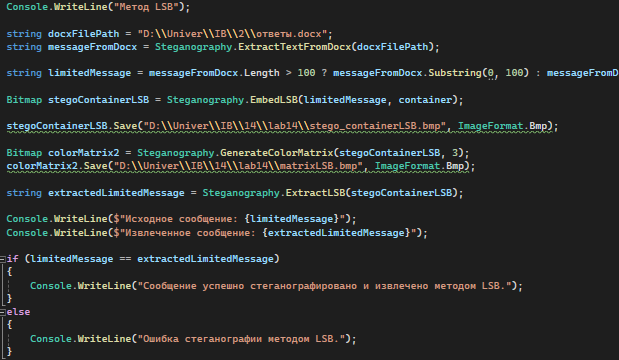


Рисунок 3.6 – Входные данные

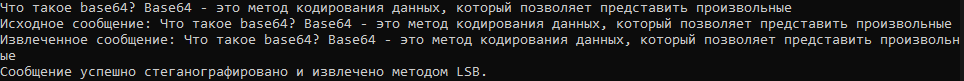


Рисунок 3.7 – Результат работы

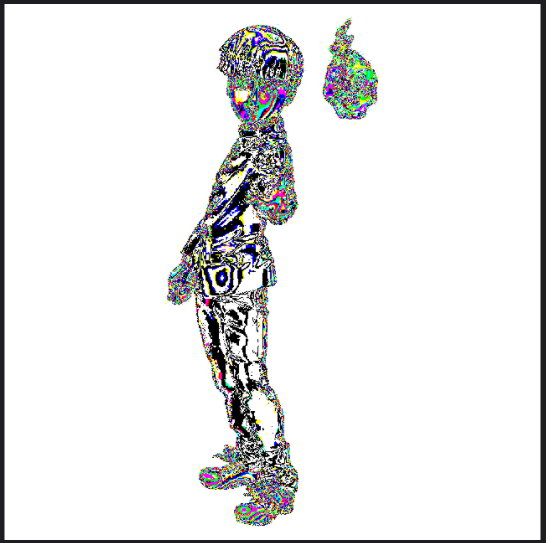


Рисунок 3.8 – Цветовая матрица

На рисунке 3.9 приведена пара из оригинального изображения и изображения полученного в ходе работы метода EmbedPixelPermutation.



Рисунок 3.9 ­– Исходное и модифицированное изображение

Исходя из визуального анализа отличий не видно.

На рисунке 3.10 приведена пара из оригинального изображения и изображения полученного в ходе работы метода EmbedLSB



Рисунок 3.10 ­– Исходное и модифицированное изображение

Изменений на «глаз» так-же не обнаружено

**Вывод**

В ходе лабораторной работы были изучены стеганографические методы встраивания/извлечения тайной информации с использованием электронного файла-контейнера на основе преобразования наименее значащих битов (НЗБ), приобретение практических навыков программной реализации данного метода.

Также было разработано авторское приложение в соответствии с целью лабораторной работы.

**Приложение 1**

Методы EmbedPixelPermutation и ExtractPixelPermutation

public static Bitmap EmbedPixelPermutation(Bitmap container, byte[] message)

{

int width = container.Width;

int height = container.Height;

Bitmap stegoContainer = new Bitmap(container);

int messageIndex = 0;

int bitIndex = 0;

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

{

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

{

if (messageIndex < message.Length)

{

System.Drawing.Color pixel = container.GetPixel(x, y);

int red = pixel.R;

int green = pixel.G;

int blue = pixel.B;

// Получаем текущий байт сообщения и его бит

byte currentByte = message[messageIndex];

int currentBit = (currentByte >> (7 - bitIndex)) & 0x01;

// Заменяем младший бит компоненты цвета пикселя на бит сообщения

red = (red & 0xFE) | currentBit;

// Создаем новый пиксель с измененной компонентой цвета

System.Drawing.Color stegoPixel = System.Drawing.Color.FromArgb(red, green, blue);

// Заменяем пиксель в стегоконтейнере

stegoContainer.SetPixel(x, y, stegoPixel);

bitIndex++;

// Если все 8 бит текущего байта сообщения были внедрены, переходим к следующему байту

if (bitIndex >= 8)

{

bitIndex = 0;

messageIndex++;

}

}

}

}

return stegoContainer;

}

public static string ExtractPixelPermutation(Bitmap stegoContainer, int messageLength)

{

int width = stegoContainer.Width;

int height = stegoContainer.Height;

List<byte> messageBytes = new List<byte>();

int bitIndex = 0;

byte currentByte = 0;

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

{

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

{

System.Drawing.Color pixel = stegoContainer.GetPixel(x, y);

// Извлекаем младший бит компоненты цвета пикселя

int redBit = pixel.R & 0x01;

int greenBit = pixel.G & 0x01;

int blueBit = pixel.B & 0x01;

// Комбинируем младшие биты в байт сообщения

currentByte = (byte)((currentByte << 1) | redBit);

bitIndex++;

// Если извлечены все 8 бит текущего байта, добавляем его в сообщение

if (bitIndex >= 8)

{

messageBytes.Add(currentByte);

currentByte = 0;

bitIndex = 0;

}

// Если извлечено достаточное количество бит для сообщения, завершаем извлечение

if (messageBytes.Count >= messageLength)

{

break;

}

}

if (messageBytes.Count >= messageLength)

{

break;

}

}

// Преобразуем массив байт в строку с использованием кодировки UTF-8

string message = Encoding.UTF8.GetString(messageBytes.ToArray());

return message;

}

**Приложение 2**

Методы EmbedLSB и ExtractLSB

public static Bitmap EmbedLSB(string message, Bitmap bmp)

{

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

int messageLength = messageBytes.Length;

// Check if the image can hold the message

if ((messageLength + 2) \* 8 > bmp.Width \* bmp.Height)

{

throw new ArgumentException("Message is too large to be hidden in the given image.");

}

// Embed the length of the message as the first two bytes

byte[] lengthBytes = BitConverter.GetBytes((short)messageLength);

byte[] fullMessageBytes = new byte[messageBytes.Length + lengthBytes.Length];

Array.Copy(lengthBytes, fullMessageBytes, lengthBytes.Length);

Array.Copy(messageBytes, 0, fullMessageBytes, lengthBytes.Length, messageBytes.Length);

Bitmap stegoImage = new Bitmap(bmp);

int byteIndex = 0, bitIndex = 0;

for (int y = 0; y < stegoImage.Height; y++)

{

for (int x = 0; x < stegoImage.Width; x++)

{

if (byteIndex >= fullMessageBytes.Length)

{

return stegoImage;

}

System.Drawing.Color pixel = stegoImage.GetPixel(x, y);

byte r = pixel.R, g = pixel.G, b = pixel.B;

// Вставка бита в младший бит синего канала

b = (byte)((b & 0xFE) | ((fullMessageBytes[byteIndex] >> bitIndex) & 1));

Console.WriteLine($"Embedding bit: {(fullMessageBytes[byteIndex] >> bitIndex) & 1} at pixel ({x}, {y})");

bitIndex++;

if (bitIndex == 8)

{

bitIndex = 0;

byteIndex++;

}

System.Drawing.Color newPixel = System.Drawing.Color.FromArgb(r, g, b);

stegoImage.SetPixel(x, y, newPixel);

}

}

return stegoImage;

}

public static string ExtractLSB(Bitmap bmp)

{

int byteIndex = 0, bitIndex = 0;

byte[] lengthBytes = new byte[2];

// Extract the length of the message first

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

{

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

{

if (byteIndex < lengthBytes.Length \* 8)

{

System.Drawing.Color pixel = bmp.GetPixel(x, y);

byte b = pixel.B;

// Extract the least significant bit of the blue channel

lengthBytes[byteIndex / 8] |= (byte)((b & 1) << (byteIndex % 8));

Console.WriteLine($"Extracting length bit: {(b & 1)} from pixel ({x}, {y})");

byteIndex++;

if (byteIndex >= lengthBytes.Length \* 8)

{

break;

}

}

else

{

break;

}

}

if (byteIndex >= lengthBytes.Length \* 8)

{

break; // Exit the outer loop once the length is extracted

}

}

int messageLength = BitConverter.ToInt16(lengthBytes, 0);

byte[] messageBytes = new byte[messageLength];

byteIndex = 0;

bitIndex = 0;

int extractedBytes = 0; // Variable to track the number of extracted bytes

// Extract the message using the obtained length

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

{

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

{

if (extractedBytes < messageLength) // Check if all message bytes are extracted

{

// Skip processing until the length of the message is reached

if (byteIndex < lengthBytes.Length)

{

bitIndex++;

byteIndex += bitIndex / 8;

bitIndex %= 8;

continue;

}

System.Drawing.Color pixel = bmp.GetPixel(x, y);

byte b = pixel.B;

// Extract the least significant bit of the blue channel

messageBytes[extractedBytes] |= (byte)((b & 1) << bitIndex);

Console.WriteLine($"Extracting message bit: {(b & 1)} from pixel ({x}, {y})");

Console.WriteLine($"{Encoding.UTF8.GetString(messageBytes)}");

bitIndex++;

if (bitIndex == 8)

{

bitIndex = 0;

extractedBytes++;

}

}

else

{

return Encoding.UTF8.GetString(messageBytes); // Return the extracted message

}

}

}

return Encoding.UTF8.GetString(messageBytes); // Return the extracted message

}