МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ЗАПОРІЗЬКА ПОЛІТЕХНІКА»

кафедра програмних засобів

Звіт

з лабораторної роботи № 3

з дисципліни «Безпека програм та даних»

на тему:

«Стеганографічний захист»

Виконав

Ст. гр. КНТ-137 В.В. Козлов

Прийняв

Ст. викладач О.І. Качан

1. Мета роботи

Ознайомитися з методами стеганографічного захисту інформації.

1. Завдання на лабораторну роботу

Реалізувати метод LSB. В якості контейнера застосувати файл формата .bmp. В якості повідомлення викорістати своє «Прізвище Ім’я Побатькові».

1. Результати виконання роботи

Протягом виконання цієї лабораторної роботи було розроблено консольну програму на С#, яка надає можливість записувати і зчитувати інформацію у файли формата .bmp використовуючи LSB-метод (Least Significant Bits) приховування інформації.

Реалізовані методи запису та зчитування інформації було виділено в окремий файл LSBMethod.cs з класом LSBMethod.

* 1. Опис файлу LSBMethod.cs

Файл LSBMethod.cs складається з одного класу – LSBMethod. Клас LSBMethod – це клас, який реалізує зазначені методи запису та зчитування інформації.

Методи класу LSBMethod наведено у таблиці 3.1.

Таблиця 3.1 – Методи класу LSBMethod

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № | Назва методу | Призначення | Тип даних | Модифікатор доступу |
| 1 | WriteMessageToBMP | Метод, що записує інформацію | bool | public |
| 2 | ReadMessageFromBMP | Метод, що зчитує інформацію | string | public |

* 1. Лістинг файлу LSBMethod.cs

using System;

using System.Collections;

using System.Collections.Generic;

using System.IO;

namespace Lab3

{

class LSBMethod

{

static public bool WriteMessageToBMP(string pathToInput, string pathToOutput, string message, int bitsToReplace)

{

if (pathToInput == "" || pathToInput == null) return false;

if (pathToOutput == "" || pathToOutput == null) return false;

if (message == "" || message == null) return false;

if (bitsToReplace < 1 || bitsToReplace > 8) return false;

try

{

int imageWidth = 0, imageHeight = 0;

int imageOffset = 0, WidthOffset = 0;

using (BinaryReader reader = new BinaryReader(File.Open(pathToInput, FileMode.Open)))

{

reader.BaseStream.Seek(0x00, SeekOrigin.Begin);

string Signature = new string(reader.ReadChars(2));

if (Signature != "BM")

return false;

reader.BaseStream.Seek(0x0A, SeekOrigin.Begin);

imageOffset = reader.ReadInt32();

reader.BaseStream.Seek(0x12, SeekOrigin.Begin);

imageWidth = reader.ReadInt32();

reader.BaseStream.Seek(0x16, SeekOrigin.Begin);

imageHeight = reader.ReadInt32();

reader.BaseStream.Seek(0x1C, SeekOrigin.Begin);

int bitPerPixel = reader.ReadInt32();

if (bitPerPixel != 24)

return false;

if (bitsToReplace \* 3 \* imageWidth \* imageHeight < message.Length \* 8 + sizeof(int))

return false;

WidthOffset = 4 - ((3 \* imageWidth) % 4);

WidthOffset = WidthOffset == 4 ? 0 : WidthOffset;

}

if (File.Exists(pathToOutput))

File.Delete(pathToOutput);

File.Copy(pathToInput, pathToOutput);

List<bool> bitMessage = new List<bool>();

foreach (bool bit in new BitArray(BitConverter.GetBytes(message.Length)))

bitMessage.Add(bit);

foreach (char ch in message)

{

int i = 0;

foreach (bool bit in new BitArray(BitConverter.GetBytes(ch)))

{

if (i < 8)

bitMessage.Add(bit);

++i;

}

}

using (BinaryWriter writer = new BinaryWriter(File.Open(pathToOutput, FileMode.Open)))

{

using (BinaryReader reader = new BinaryReader(File.Open(pathToInput, FileMode.Open)))

{

byte[] PowerTwo = new byte[8] { 1, 2, 4, 8, 16, 32, 64, 128 };

int currBit = 0;

int bitMessageSize = bitMessage.Count;

writer.Seek(imageOffset, SeekOrigin.Begin);

reader.BaseStream.Seek(imageOffset, SeekOrigin.Begin);

for (int i = 0; i < imageHeight; ++i)

{

for (int j = 0; j < imageWidth; ++j)

{

for (int f = 0; f < 3; ++f)

{

byte color = reader.ReadByte();

if (currBit < bitMessageSize)

{

for (int k = bitsToReplace - 1; k >= 0; --k)

{

if (currBit >= bitMessageSize)

break;

if (bitMessage[currBit])

color = (byte)(color | PowerTwo[k]);

else

color = (byte)(color & (~PowerTwo[k]));

currBit++;

}

writer.Write(color);

}

else

writer.Seek(1, SeekOrigin.Current);

}

}

writer.Seek(WidthOffset, SeekOrigin.Current);

reader.BaseStream.Seek(WidthOffset, SeekOrigin.Current);

}

}

writer.Flush();

}

}

catch (Exception)

{

if (File.Exists(pathToOutput))

File.Delete(pathToOutput);

return false;

}

return true;

}

static public string ReadMessageFromBMP(string pathToInput, int bitsToReplace)

{

if (pathToInput == "" || pathToInput == null) return null;

if (bitsToReplace < 1 || bitsToReplace > 8) return null;

try

{

int imageWidth = 0, imageHeight = 0;

int imageOffset = 0, WidthOffset = 0;

using (BinaryReader reader = new BinaryReader(File.Open(pathToInput, FileMode.Open)))

{

reader.BaseStream.Seek(0x00, SeekOrigin.Begin);

string Signature = new string(reader.ReadChars(2));

if (Signature != "BM")

return null;

reader.BaseStream.Seek(0x0A, SeekOrigin.Begin);

imageOffset = reader.ReadInt32();

reader.BaseStream.Seek(0x12, SeekOrigin.Begin);

imageWidth = reader.ReadInt32();

reader.BaseStream.Seek(0x16, SeekOrigin.Begin);

imageHeight = reader.ReadInt32();

reader.BaseStream.Seek(0x1C, SeekOrigin.Begin);

int bitPerPixel = reader.ReadInt32();

if (bitPerPixel != 24)

return null;

WidthOffset = 4 - ((3 \* imageWidth) % 4);

WidthOffset = WidthOffset == 4 ? 0 : WidthOffset;

byte[] PowerTwo = new byte[8] { 1, 2, 4, 8, 16, 32, 64, 128 };

List<bool> bitMessage = new List<bool>();

reader.BaseStream.Seek(imageOffset, SeekOrigin.Begin);

for (int i = 0; i < imageHeight; ++i)

{

for (int j = 0; j < imageWidth; ++j)

{

for (int f = 0; f < 3; ++f)

{

byte color = reader.ReadByte();

for (int k = bitsToReplace - 1; k >= 0; --k)

{

if (color == (color | PowerTwo[k]))

bitMessage.Add(true);

else

bitMessage.Add(false);

}

}

}

reader.BaseStream.Seek(WidthOffset, SeekOrigin.Current);

}

BitArray messageSizeBitArray = new BitArray(32);

for (int i = 0; i < 32; ++i)

messageSizeBitArray[i] = bitMessage[i];

byte[] messageSizeByteArray = new byte[4];

messageSizeBitArray.CopyTo(messageSizeByteArray, 0);

int messageSize = BitConverter.ToInt32(messageSizeByteArray);

BitArray messageBitArray = new BitArray(bitMessage.Count - 32);

for (int i = 0; i < bitMessage.Count - 32; ++i)

messageBitArray[i] = bitMessage[i + 32];

byte[] messageByteArray = new byte[(bitMessage.Count - 32) / 8 + 1];

messageBitArray.CopyTo(messageByteArray, 0);

string message = "";

for (int i = 0; i < messageSize; ++i)

message += Convert.ToChar(messageByteArray[i]);

return message;

}

}

catch (Exception)

{

return null;

}

}

}

}

* 1. Інтерфейс користувача

Взаємодія користувача з програмою, реалізована завдяки консолі, шляхом послідовного виконання операцій, вибраних користувачем за допомогою текстового меню. Program

* + 1. Лістинг файлу Program.cs

using System;

namespace Lab3

{

class Program

{

static void Main(string[] args)

{

string pathToInput, pathToOutput, message;

string choice;

while (true)

{

Console.Write("Encrypt/decrypt(1,0): ");

choice = Console.ReadLine();

if (choice == "1")

{

Console.Write("Write path to input file: ");

pathToInput = Console.ReadLine();

Console.Write("Write path to output file: ");

pathToOutput = Console.ReadLine();

Console.Write("Write message: ");

message = Console.ReadLine();

if (LSBMethod.WriteMessageToBMP(pathToInput, pathToOutput, message, 4))

Console.WriteLine("Encoding successfull");

else

Console.WriteLine("Encoding failed");

}

else if (choice == "0")

{

Console.Write("Write path to input file: ");

pathToInput = Console.ReadLine();

message = LSBMethod.ReadMessageFromBMP(pathToInput, 4);

if (message == "" || message == null)

Console.WriteLine("Decoding failed");

else

{

Console.Write("Decoding message: ");

Console.WriteLine(message);

}

}

else if (choice == "end")

break;

Console.WriteLine();

}

}

}

}

* 1. Робота з програмою

Для роботи з програмою перш за все необхідно запустити виконуваний файл. Для запису інформації у графічний файл, необхідно вибрати режим Encoding (1). Далі необхідно ввести шлях до вхідного файлу, вихідного файлу та саму інформацію. В разі успішного виконання запису з’явиться повідомлення Encoding successfull, в іншому випадку (не знайдено файл, не можливо внести зміни до файлу чи невідповідність розмірів) з’явиться повідомлення Encoding failed.

Для зчитування інформації з графічного файлу , необхідно вибрати режим Decoding (0) і далі ввести шлях до вхідного файлу. У випадку успішного зчитування інформації з’явиться повідомлення Decoding message: X, де X – це повідомлення, в іншому випадку (не можливо зчитати дані з файлу) з’явиться повідомлення Decoding failed.

Для завершення роботи необхідно ввести end замість вибору режиму.

* + 1. **Запис та зчитування даних**

Нехай обране повідомлення «Valeriy Kozlov. Third laboratory work», кількість молодших біт для зміни – 4.

На рисунку 3.1 зображено результат роботи програми у консолі.

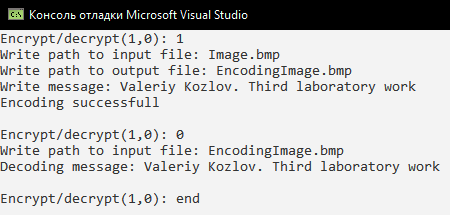


Рисунок 3.1 – Результат роботи з програмою з консолі

На рисунку 3.2 зображено порівняння вхідного (чистого) і вихідного (з записаною інформацією) зображення.

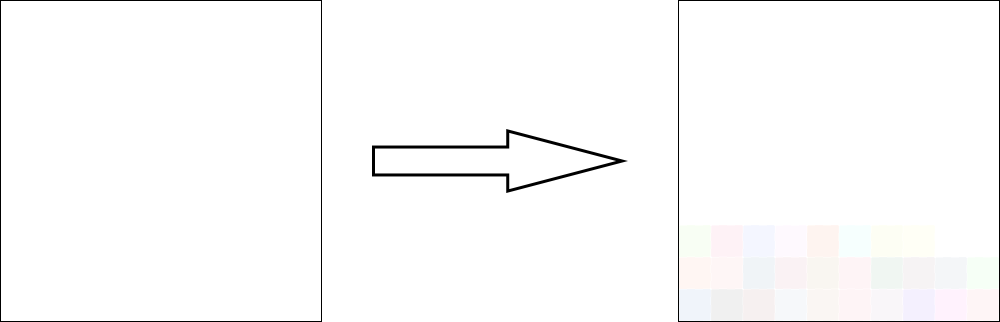


Рисунок 3.2 – Порівняння вхідного і вихідного зображення

Висновки

Протягом виконання цієї лабораторної роботи ознайомився з методами стеганографічного захисту інформації, а саме – LSB-методом (Least Significant Bits) приховування інформації.

Розробив консольну програму на С#, яка надає можливість записувати і зчитувати інформацію у файли формата .bmp використовуючи LSB-метод приховування інформації.