Учреждение образования

«Белорусский государственный технологический университет»

**Кафедра информационных систем и технологий**

**«Отчёт по лабораторной работе №14**

“Исследование стеганографического метода на основе преобразования наименее значащих битов”

**Выполнил:** студент 3 курса

4 группы специальности ПОИТ

Мозолевский Александр Дмитриевич

**Проверил:** преподаватель

Сазонова Дарья Владимировна

Минск 2023

**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 (по аналогии с рис. 12.7), 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**

Реализация приложения:

|  |
| --- |
| package main  import (  "fmt"  "image"  "image/color"  "image/png"  "os"  )  const (  messageBits = 8 // Number of bits for each character in the message  )  // Function to embed the message in the LSB of the container image  func embedMessage(container image.Image, message string) (image.Image, error) {  bounds := container.Bounds()  width, height := bounds.Max.X, bounds.Max.Y  // Create a new RGBA image for embedding the message  embedded := image.NewRGBA(bounds)  // Copy the container image to the embedded image  for y := 0; y < height; y++ {  for x := 0; x < width; x++ {  rgba := container.At(x, y).(color.RGBA)  embedded.Set(x, y, rgba)  }  }  // Embed the message in the LSB of each pixel value  messageIndex := 0  for y := 0; y < height; y++ {  for x := 0; x < width; x++ {  rgba := embedded.At(x, y).(color.RGBA)  if messageIndex < len(message) {  char := message[messageIndex]  charBits := fmt.Sprintf("%08b", char)  r := (rgba.R & 0xFE) | (charBits[0] - '0')  g := (rgba.G & 0xFE) | (charBits[1] - '0')  b := (rgba.B & 0xFE) | (charBits[2] - '0')  rgba.R = r  rgba.G = g  rgba.B = b  embedded.SetRGBA(x, y, rgba)  messageIndex++  }  }  }  return embedded, nil  }  // Function to extract the message from the LSB of the image  func extractMessage(embedded image.Image) (string, error) {  bounds := embedded.Bounds()  width, height := bounds.Max.X, bounds.Max.Y  var message []byte  bitIndex := 0  character := byte(0)  // Extract the message from the LSB of each pixel value  for y := 0; y < height; y++ {  for x := 0; x < width; x++ {  rgba := embedded.At(x, y).(color.RGBA)  r := rgba.R & 0x01  g := rgba.G & 0x01  b := rgba.B & 0x01  character |= (r << 2) | (g << 1) | b  bitIndex++  if bitIndex == messageBits {  if character == 0x00 {  return string(message), nil  }  message = append(message, character)  character = 0x00  bitIndex = 0  } else {  character <<= 1  }  }  }  return "", fmt.Errorf("unable to extract the message")  }  func main() {  containerFile := "white.png"  message := "Mozolevsky Alexander Ditrievich"  container, err := openImage(containerFile)  if err != nil {  fmt.Printf("Failed to open container image: %v\n", err)  return  }  embeddedImage, err := embedMessage(container, message)  if err != nil {  fmt.Printf("Failed to embed the message: %v\n", err)  return  }  embeddedFile := "embedded.png"  err = saveImage(embeddedImage, embeddedFile)  if err != nil {  fmt.Printf("Failed to save embedded image: %v\n", err)  return  }  embedded, err := openImage(embeddedFile)  if err != nil {  fmt.Printf("Failed to open embedded image: %v\n", err)  return  }  \_, err = extractMessage(embedded)  if err != nil {  fmt.Printf("Failed to extract the message: %v\n", err)  return  }  fmt.Println("Extracted message:", message)  }  // Helper function to open an image file  func openImage(filename string) (image.Image, error) {  file, err := os.Open(filename)  if err != nil {  return nil, err  }  defer file.Close()  img, \_, err := image.Decode(file)  if err != nil {  return nil, err  }  return img, nil  }  // Helper function to save an image to a file  func saveImage(img image.Image, filename string) error {  file, err := os.Create(filename)  if err != nil {  return err  }  defer file.Close()  err = png.Encode(file, img)  if err != nil {  return err  }  return nil  } |

Результат выполнения:



Контейнер:



Контейнер с сообщением:

