Міністерство освіти і науки України

Національний університет «Запорізька політехніка»

кафедра програмних засобів

ЗВІТ

з лабораторної роботи № 3

з дисципліни «Безпека та захист програм і даних» на тему:

«СТЕНОГРАФІЧНИЙ ЗАХИСТ»

Виконав:

ст. гр. КНТ-113сп Іван ЩЕДРОВСЬКИЙ

Прийняв:

доцент Тетяна ЗАЙКО

2025

1 Мета роботи

Ознайомитись методами стенографічного захисту інформації

2 Завдання до лабораторної роботи

2.1 Реалізувати програму для LSB. В якості контейнера застосувати файл формату *.bmp. В якості повідомлення використати своє “Прізвище Ім’я По-батькові”

3 Відповіді на контрольні питання

Що таке стеганографічний метод захисту інформації?

Стенографія – це наука про приховання однієї інформації серед іншої. Це метод організації зв’язку, що власне ховає саму наявність зв’язку

Які недоліки стеганографічного захисту?

Стенографічна інформацію легко пошкоджується. Наприклад, якщо картинка буде стиснута після застосування LSB, бо дані вже не можна буде розшифрувати

Для стенографії потрібно багато початкових даних в яких можна заховати інші дані. Якщо, наприклад, у нас маленька картинка, то заховати в ній щось буде досить складно

Стенографія не підходить для великих обсягів даних, бо шанс їх пошкодження при передачі більший, та потрібно багато інформації щоб їх приховати

Чим відрізняється стеганографія від криптографії?

В криптографії ворог точно може визначити, чи є передане повідомлення зашифрованим текстом, а в стеганографії повідомлення вбудовується таким чином, щоб не можливо було запідозрити існування вбудованого повідомлення

В чому сутність метода LSB?

Сутність метода LSB при роботі з картинками в тому, що колір окремого пікселя представляється декількома бітами. Наприклад, RGB має 24 біти, або ж 32 якщо з alpha каналом. Тобто, кожен канал має 8 біт

В бінарних числах позиція біту по різному впливає на кінцевий результат. Наприклад, у нас є число 10001000_2 або 136_{10}. Якщо ми змінимо найбільш правий нуль, або ж перший, то значення числа зміниться на 1, тобто 10001001_2 це 13710. Але, якщо змінити останній біт, або ж найлівіший, то значення зміниться дуже сильно, 00001000_2 це 8_{10}.

Найправіший біт, або ж перший є найменш значущим. Це і є LSB, або ж Least Significant Bit. А найлівіший біт, або ж останній(або перший, дивлячись як рахувати) є найбільш значущим, тобто Most Significant Bit

Суть метода LSB для картинок полягає в тому, що змінюючи один найменш значущий біт в кольорі майже не змінює цей колір. Це дуже складно відрізнити на око та також досить складно відрізнити навіть комп’ютером без початкового зображення, якщо картинка, звичайно, має достатню кількість кольорів

Що таке цифрові водяні знаки?

Цифровий водяний знак – це технологія захисту авторських прав мультимедійних файлів

Зазвичай ця інформація являє собою текст або логотип, який ідентифікує автора. Зазвичай цифрові водяні знаки невидимі, але на зображеннях або відео можуть бути видимими

Як використовуються цифрові водяні знаки?

Наприклад, компанія може вбудовувати цифрові водяні знаки в відео коли надає його в приватному вигляді невеликій групі людей щоб мати змогу в майбутньому зрозуміти від кого був витік

Використовується як засіб захисту документів з фотографіями — паспортів, водійських посвідчень, кредитних карток з фотографіями. Коментарі до цифрових фотографій з описовою інформацією — ще один приклад невидимих ЦВЗ

Що таке контейнер?

Контейнер – це будь-яка інформація, призначена для приховання таємних повідомлень

Що таке стегоконтейнер?

Це контейнер, який містить вбудовану інформацію

Що може виступати як контейнер?

Картинки, відео, аудіо, інші типи файли, навіть фізичні диски та пристрої

Вимоги, пропоновані до стего-повідомлень?

Під стего-повідомленням розуміється сама передача інформації через стегоканал. Якщо ж це вимоги саме до прихованого повідомлення – то їх майже немає, воно не повинно бути дуже великим, щоб його можна було зашифрувати, а також його повинно бути можна зашифрувати фізично

Вимогою до стего-повідомлень є те, що навіть якщо злоумисник дізнається про існування схованого повідомлення, це не повинно дозволити йому витягти подібні повідомлення в інших даних доти, поки ключ зберігається в таємниці

Які методи вбудовування повідомлень ви знаєте?

Є методи які працюють з самим цифровим сигналом, наприклад, LSB

Є методи коли відбувається накладення приховуваного зображення(або звуку, тексту) на оригінал. Це часто використовується для ЦВЗ

Також використовуються особливості форматів файлів, наприклад, запис в метадані або в різні ще не використовувані зарезервовані поля файлу

У LSB також є підвиди.

BlindHide (приховування наосліп) записує дані з верхнього лівого кута зображення до правого нижнього, піксель за пікселем

HideSeek (заховати-знайти). Цей алгоритм у псевдовипадковий спосіб розподіляє приховане повідомлення у контейнері. Для генерації випадкової послідовності використовує пароль

FilterFirst (попередня фільтрація). Виконує фільтрацію зображення-контейнера — пошук пікселів, у які записуватиметься прихована інформація (для яких зміна наймолодших розрядів буде найменш помітною для ока людини).

BattleSteg (стеганографія морської битви). Найскладніший і найдосконаліший алгоритм. Спочатку виконує фільтрацію зображення-контейнера, після чого прихована інформація записується у «найкращі місця» контейнера у псевдовипадковий спосіб (подібно, як у HideSeek)

Інші методи приховування інформації в графічних файлах орієнтовані на формати файлів з втратою, наприклад, JPEG

Що в даній програмі може бути використано як ключ?

Це може бути ключ для шифрування даних перед тим, як їх записувати в зображення/…

Це може бути функція, яка визначає позицію закодованих пікселів. Або ж якщо брати більш складні методи, то рядок який визначає позиції пікселів, можливо навіть без зміни зображення

4 Текст програми

```
package lb3

import (
	"bufio"
	"fmt"
	"image"
	"image/color"
	"image/draw"
	"os"

	"golang.org/x/image/bmp"
)

const DEBUG = true

func readImage(path string) image.Image {
	inputFile, err := os.Open(path)

	if err != nil {
		panic(err)
	}

	defer func() {
		if err := inputFile.Close(); err != nil {
			panic(err)
		}
	}()
	reader := bufio.NewReader(inputFile)

	image, err := bmp.Decode(reader)

	if err != nil {
		panic(err)
	}

	return image
}

func writeImage(path string, outputImage image.Image) {
	outputFile, err := os.Create(path)

	if err != nil {
```

```
		panic(err)
	}

	defer func() {
		if err := outputFile.Close(); err != nil {
			panic(err)
		}
	}()

	writer := bufio.NewWriter(outputFile)
	err = bmp.Encode(writer, outputImage)

	if err != nil {
		panic(err)
	}
}

func encodeMessage(message string, inputImage image.Image) image.Image {
	bounds := inputImage.Bounds()
	rgba := image.NewRGBA(image.Rect(0, 0, bounds.Dx(), bounds.Dy()))
	draw.Draw(rgba, rgba.Bounds(), inputImage, bounds.Min, draw.Src)

	res := ""
	for _, r := range []byte(message) {
		res += fmt.Sprintf("%08b", r)
	}

	if len(res) > bounds.Size().X*bounds.Size().Y {
		panic("Image is small for this message")
	}

	x, y := bounds.Min.X, bounds.Min.Y
	for _, char := range res {
		one := uint8(0)
		if char == '1' {
			one = 1
		}

		cr, cg, cb, ca := rgba.At(x, y).RGBA()

		r := uint8(cr >> 8)
		g := uint8(cg >> 8)
		b := uint8(cb >> 8)
		a := uint8(ca >> 8)

		if r&1 != one {
			r ^= 1
		}

		if DEBUG {
			if one == 1 {
				r = 255
				g = 255
				b = 255
			} else {
				r = 0
				g = 0
				b = 0
			}
```

```
                }

                rgba.Set(x, y, color.RGBA{r, g, b, a})

                x += 5

                if x >= bounds.Max.X {
                        y += 10
                        x = bounds.Min.X
                }
        }

        return rgba
}

func decodeMessage(encodedImage image.Image, length int) string {
        bounds := encodedImage.Bounds()
        rgba := image.NewRGBA(image.Rect(0, 0, bounds.Dx(), bounds.Dy()))
        draw.Draw(rgba, rgba.Bounds(), encodedImage, bounds.Min, draw.Src)

        data := make([]byte, 0)
        var newByte byte = 0

        i := 7

        for y := bounds.Min.Y; y < bounds.Max.Y; y += 10 {
                for x := bounds.Min.X; x < bounds.Max.X; x += 5 {
                        cr, _, _, _ := rgba.At(x, y).RGBA()
                        r := uint8(cr >> 8)

                        if r&1 == 1 {
                                newByte = newByte | (1 << i)
                        }

                        i--
                        if i < 0 {
                                data = append(data, newByte)
                                i = 7
                                newByte = 0

                                if len(data) >= length {
                                        return string(data)
                                }
                        }
                }
        }

        return string(data)
}

func Run() {
        fmt.Println("Lb 3")

        // originalMessage := "Modern cryptography is heavily based on mathematical theory and computer science practice; cryptographic algorithms are designed around computational hardness assumptions, making such algorithms hard to break in actual practice by any adversary. While it is theoretically possible to break into a well-designed system, it is infeasible in actual practice to do so. Such schemes, if well designed, are therefore termed \"computationally secure\". Theoretical advances (e.g., improvements in integer factorization algorithms) and faster computing technology require these designs to be continually reevaluated and, if necessary, adapted.
```

```
Information-theoretically secure schemes that provably cannot be broken even with unlimited computing power,
such as the one-time pad, are much more difficult to use in practice than the best theoretically breakable but
computationally secure schemes."
        originalMessage := "Щедровський Іван Андрійович"

        fmt.Println("Message:", originalMessage)
        fmt.Println()

        fmt.Println("Read original image")
        originalImage := readImage("./lb3/input-cat.bmp")

        fmt.Println("Create encoced image")
        encodedImage := encodeMessage(originalMessage, originalImage)

        fmt.Println("Save encoded image")
        writeImage("./lb3/output-cat.bmp", encodedImage)

        fmt.Println("Read text from encoded image")
        decodedMessage := decodeMessage(encodedImage, len(originalMessage))

        fmt.Println()
        fmt.Println("Decoded:", decodedMessage)

        fmt.Println()
        fmt.Println("Is original == decoded?", originalMessage == decodedMessage)
}
```

5 Результати роботи програми

Програма написана на мові програмування Golang. Спочатку відбувається зчитування bmp картинки, після шифрування повідомлення в картинку, її збереження та розшифрування

Для роботи було обрано картинку, яка показана на рисунку 1

Рисунок 1 – Початкова картинка

Для шифрування даних було обрано використовувати тільки червоний канал, а також кожен п’ятий по вісі X і кожен десятий по вісі Y пікселі. Звичайно, для збільшення надійності алгоритму краще використовувати різні канали, або ж їх комбінації, а також використовувати не лінійні пікселі, а якусь спеціальну формулу. Але, в межах демонстрації такого застосунку більш ніж достатньо

Якщо взяти достатньо велике повідомлення та ввімкнути режим налагодження програм, то в результаті можна побачити сітку. В режимі налагодження RGB канали замінюються на білий або чорний колір в залежності від біту вхідного тексту. Приклад картинки показаний на рисунку 2

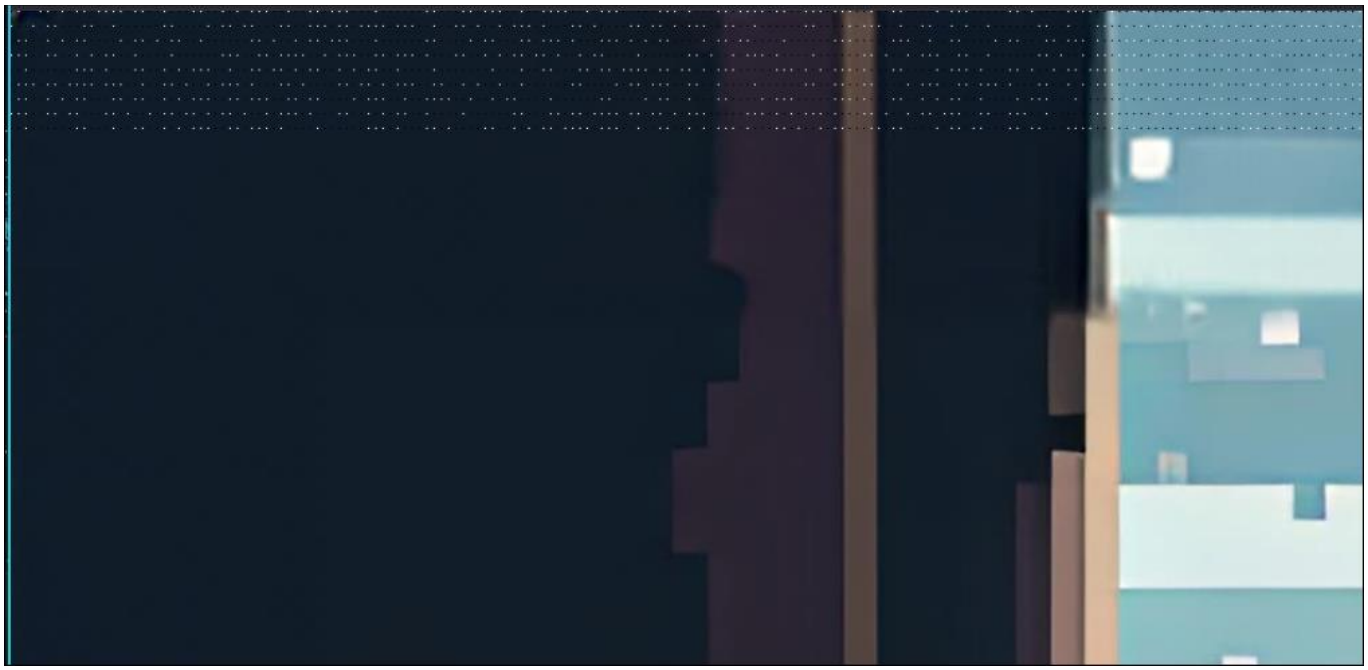

Рисунок 2 – Режим налагодження з великим повідомленням

Приклад роботи програми з повідомленням “Прізвище Ім’я По-батькові” показаний на рисунку 3. Вихідне зображення показане на рисунку 4. Вихідне зображення в режимі налагодження, дуже збільшене, щоб показати дані, показане на рисунку 5.

```
C:\home\university-4\security>go run .
Lb 3
Message: Щедровський Іван Андрійович

Read original image
Create encoced image
Save encoded image
Read text from encoded image

Decoded: Щедровський Іван Андрійович

Is original == decoded? true
```

Рисунок 3 – Результат виконання програми

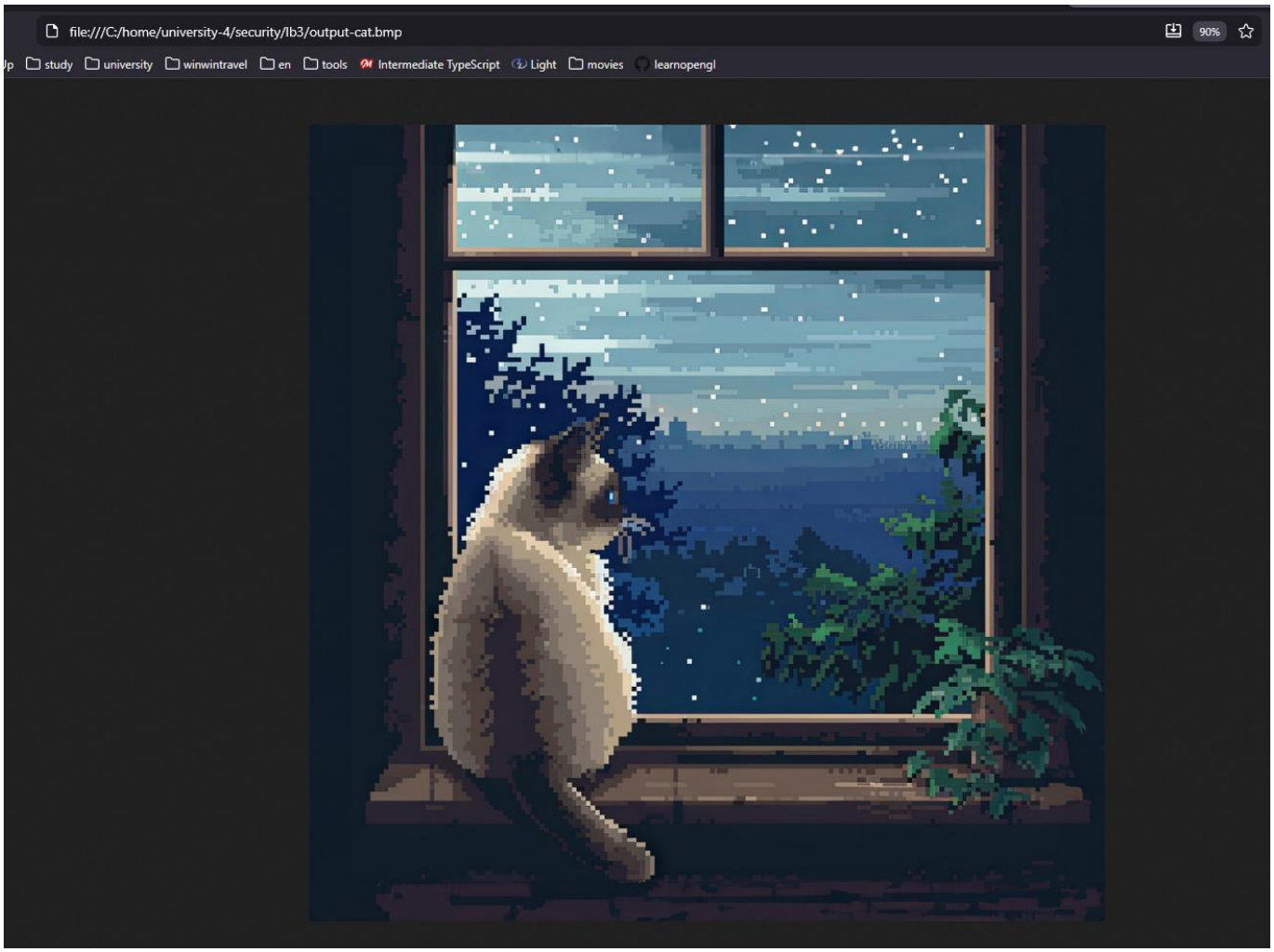

Рисунок 4 – Вихідне зображення з зашифрованою інформацією

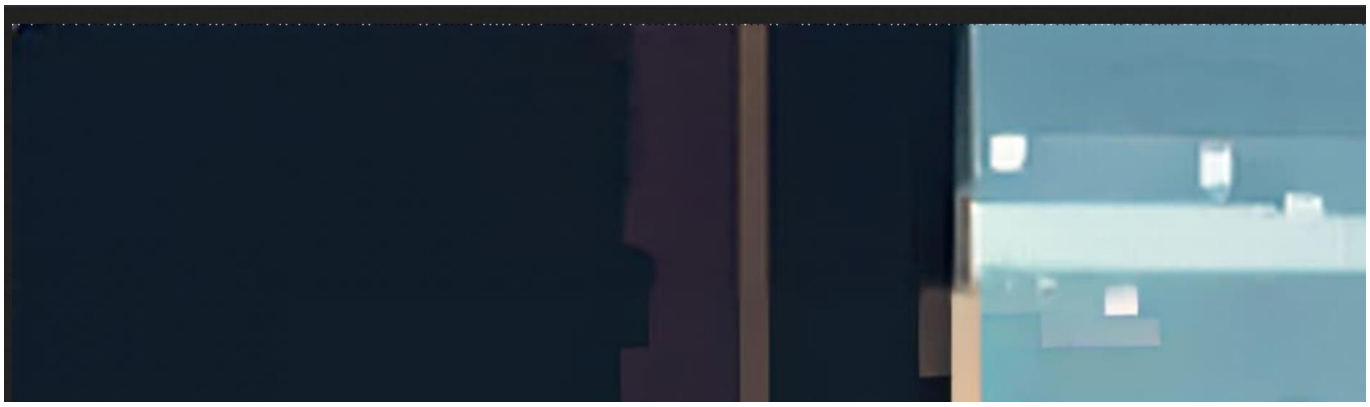

Рисунок 5 – Режим налагодження

6 Висновки

Я ознайомився з методами стенографічного захисту інформації