Міністерство освіти і науки України
Харківський національний університет ім. В. Н. Каразіна
Факультет комп'ютерних наук
Спеціальність 125 «Кібербезпека»
Освітня програма «Кібербезпека»

"Допущено до захисту"
В.о. завідувача кафедри БІСТ
Мелкозьорова О.М._____
"_____" червня 2024р.

Пояснювальна записка

до кваліфікаційної роботи бакалавра

на тему: «Аналіз та дослідження методів приховування інформації в цифрових зображеннях»

оцінка « » Керівник ст. викладач Шеханін К. Ю.

Голова ЕК

Лемешко О. В._____ Рецензент _____Лисицький К. Є.

Виконавець: студент групи КБ-41

_____Волотковський Д.С.

Харків – 2024

РЕФЕРАТ

Пояснювальна записка до дипломного проекту містить 63 сторінки, 34 рисунків, 3 таблиці, 2 додатки, 31 посилання на джерела, 8 лістингів.

Метою даної дипломної роботи є аналіз та дослідження методів стеганографії для приховування інформації в цифрових зображеннях, аналіз стійкості цих методів до компресії та виявлення прихованої інформації, а також дослідження особливостей реалізованих алгоритмів.

Предмет включає в себе вивчення різних методів стеганографії, таких як метод найменш значущих біт (LSB) та метод Куттера-Джордана-Боссена (KJB), аналіз їх ефективності та стійкості до стиснення.

Об'єктом роботи є цифрові зображення, в яких здійснюється приховування текстової інформації з використанням реалізованих стеганографічних методів.

У результаті проведених досліджень було реалізовано два методи стеганографії, перевірено їх ефективність у приховуванні інформації та стійкість до стиснення зображень. Метод LSB показав простоту реалізації, але низьку стійкість до компресії, тоді як метод Куттера-Джордана-Боссена виявився більш стійким до компресії, хоча й складнішим у реалізації. Було проведено тестування обох методів на різних зображеннях, що дозволило виявити їх переваги та недоліки.

Результати дослідження можуть бути використані для вибору оптимального методу стеганографії залежно від вимог до стійкості та обсягу приховуваної інформації.

Ключові слова: СТЕГАНОГРАФІЯ, LSB, МЕТОД КУТТЕРА-ДЖОРДАНА-БОССЕНА, ПРИХОВУВАННЯ ІНФОРМАЦІЇ, ЦИФРОВІ ЗОБРАЖЕННЯ, СТІЙКІСТЬ ДО КОМПРЕСІЇ, АЛГОРИТМИ.

ABSTRACT

The explanatory note for the diploma project consists of 63 pages, 34 figures, 3 tables, 2 appendices, 31 references, and 8 listings.

The purpose of this diploma work is to analyze and investigate steganography methods for concealing information in digital images, analyze the resilience of these methods to compression and detect hidden information, as well as to explore the features of implemented algorithms.

The subject includes studying various steganography methods, such as the Least Significant Bit (LSB) method and the Kutter-Jordan-Bossen (KJB) method, analyzing their effectiveness and resilience to compression.

The object of the work is digital images in which text information is concealed using implemented steganographic methods.

As a result of the conducted research, two steganography methods were implemented, their effectiveness in information hiding and resilience to image compression were tested. The LSB method demonstrated simplicity of implementation but low resilience to compression, while the Kutter-Jordan-Bossen method proved to be more resilient to compression, albeit more complex to implement. Testing of both methods was conducted on various images, which allowed identifying their advantages and disadvantages.

The research results can be used to choose the optimal steganography method depending on the requirements for resilience and the amount of hidden information.

Keywords: STEGANOGRAPHY, LSB, KUTTER-JORDAN-BOSSEN METHOD, INFORMATION HIDING, DIGITAL IMAGES, RESILIENCE TO COMPRESSION, ALGORITHMS.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

BMP - Bitmap Image File

DCT - Discrete Cosine Transform

DWT - Discrete Wavelet Transform

ДКП - Дискретне Косинусне Перетворення

EXIF - Exchangeable Image File Format

GIF - Graphics Interchange Format

JPEG - Joint Photographic Experts Group

KJB - Kutter-Jordan-Bossen

LSB - Least Significant Bit

LZW - Lempel–Ziv–Welch

НЗБ - Найменш Значущий Біт

НЧ - Низькочастотний

PNG - Portable Network Graphics

ПВП - Псевдовипадкова Послідовність

ПЗ - Програмне Забезпечення

PVD - Pixel-Value Differencing

RGB - Red Green Blue

SVG - Scalable Vector Graphics

TCP - Transmission Control Protocol

IP - Internet Protocol

TIFF - Tagged Image File Format

ВЧ - Високочастотний

WebP - Web Picture Format

YCbCr - luma - blue chroma - red chroma

ЗСЛ - Зорова Система Людини

ЦВЗ - Цифрові Водяні Знаки

ВСТУП

У сучасному світі цифрових технологій захист інформації набуває особливої актуальності. Інформація, що передається через цифрові канали, часто потребує високого рівня конфіденційності та безпеки. Одним з методів забезпечення цих вимог є стеганографія - наука про приховування інформації всередині інших даних, що дозволяє передавати секретні повідомлення без привертання уваги.

Стеганографія застосовується у різних сферах, починаючи від захисту конфіденційної інформації у військових і державних установах до забезпечення приватності у особистих комунікаціях. Цифрові зображення є одним з найпопулярніших носіїв для стеганографічного вбудовування завдяки їх розповсюдженню та легкості обміну через комунікаційні канали.

Проте, з поширенням методів приховування інформації, зростає й потреба у ефективних методах її виявлення та аналізу, відомих як стегоаналіз. Це особливо важливо для протидії зловмисникам, які можуть використовувати стеганографію для приховування шкідливих даних або незаконної інформації. Дослідження та розробка новітніх методів приховування інформації у цифрових зображеннях, а також вдосконалення поточних методів стегоаналізу, є надзвичайно важливими для підвищення рівня кібербезпеки. Це сприятиме збереженню конфіденційності даних та виявленню потенційних загроз.

Ця дипломна робота присвячена аналізу та дослідженню методів стеганографії у цифрових зображеннях. Вона має на меті проведення ґрунтовного аналізу та дослідження методів приховування інформації в цифрових зображеннях, що є одним з найпоширеніших способів стеганографічного приховування даних. Результати цього дослідження сприятимуть подальшому розвитку стеганографії та стегоаналізу, що, у свою чергу, вплине на забезпечення безпеки та приватності в цифровому просторі.

1.ТЕОРЕТИЧНІ ОСНОВИ СТЕГАНОГРАФІЇ

1.1 Стенографія: поняття, особливості, сутність

Стеганографія (від грецьких слів "steganos" - прихований та "grapho" - писати) - це наука про приховане передавання інформації з використанням невідомого каналу чи способу. На відміну від криптографії, де ворог може точно визначити, що передане повідомлення зашифроване, стеганографічні методи дозволяють вбудовувати секретні повідомлення у безпечні, незагрозливі передачі так, що навіть не можна запідозрити наявність прихованого послання. Стеганографія не замінює криптографію, а доповнює її, забезпечуючи додатковий рівень захисту шляхом приховування самого факту передачі повідомлення. [1]

Основна мета стеганографії полягає в тому, щоб переховати таємне повідомлення в іншому, незначному повідомленні таким чином, щоб не викликати підозри про наявність прихованих даних. Стеганографія передбачає використання невидимого каналу або способу передачі прихованої інформації, наприклад, вбудовування даних у цифрові мультимедійні файли (зображення, аудіо, відео). Ефективні стегосистеми повинні бути стійкими до спроб виявлення прихованих повідомлень та їх видалення. Методи стеганографії залежать від типу носія, в який вбудовуються приховані дані.

Щодо основної суті стеганографії, то вона полягає у тому щоб вбудувати секретне повідомлення (повідомлення-файл) у відкритий носій інформації (контейнер-файл) у такий спосіб, щоб різниця вхідного та вихідного носія інформації не була помітна для стороннього спостерігача. Ключем є секретний елемент, який визначає порядок внесення повідомлення в контейнер.

Процес стеганографії складається з двох основних етапів: вбудовування прихованого повідомлення в носій інформації (стегоконтейнер) та передача або зберігання стегоконтейнера. На приймальному кінці здійснюється зворотний процес – витягування прихованого повідомлення зі стегоконтейнера. [1]

Щодо оцінки ефективності стегосистеми можна виділити такі критерії: непомітність вбудованих даних (стегоконтейнер повинен бути візуально нерозрізненним від оригінального носія), висока місткість (здатність вбудовувати великі обсяги прихованих даних) та стійкість до атак (вбудовані дані повинні бути стійкими до спроб їх виявлення та видалення).

Клод Шеннон надав загальну теорію тайнопису, що стала основою стеганографії як науки. У сучасній комп'ютерній стеганографії існують два основні типи файлів:

- повідомлення-файл, який призначений для приховування
- контейнер-файл, який може використовуватися для приховування в ньому повідомлення

При цьому контейнери можуть бути двох типів: "порожній" контейнер, що не містить прихованої інформації, і "заповнений" контейнер, що містить приховану інформацію. Ключем є секретний елемент, який визначає порядок внесення повідомлення в контейнер.

На сьогодення, основні принципи комп'ютерної стеганографії [2-4] можна поділити на наступні: методи приховування повинні забезпечувати автентичність і цілісність файлу; противник повинен знати всі можливі методи стеганографії; Безпека методів базується на тому, що для збереження стеганографічних перетворень основних властивостей відкритого файлу застосовують внесення змін до секретного повідомлення; Безпека методів також забезпечується обчислювальною складністю витягу самого секретного повідомлення.

1.2 Застосування стеганографії в сучасних інформаційних системах

У зв'язку зі зростанням впливу глобальних комп'ютерних мереж значення стеганографії стає все більш актуальним. Результати аналізу інформаційних ресурсів показують, що стеганографічні системи активно використовуються для вирішення таких основних завдань:

- захист конфіденційної інформації від несанкціонованого доступу;

- обхід систем моніторингу та управління мережевими ресурсами;
- приховане впровадження програмного забезпечення;
- захист права на інтелектуальну власність.

Захист конфіденційної інформації від несанкціонованого доступу є однією з найбільш ефективних областей використання комп'ютерної стеганографії для розв'язання цієї проблеми. Наприклад, 1 секунда цифрового звуку з частотою дискретизації 44100 Гц і рівнем відліку 8 біт у стерео режимі може приховати близько 10 Кбайт інформації, замінюючи найменш значущі молодші розряди на приховане повідомлення. При цьому зміна значень відліків становить менше 1%, що практично не виявляється при прослуховуванні файлу більшістю користувачів [5]. Для реалізації захисту конфіденційної інформації від несанкціонованого доступу, як приклад можна навести деякі методи: Заміна найменш значущих бітів (LSB): Цей метод полягає в заміні найменш значущих бітів (LSB) пікселів зображення або інших даних-контейнерів бітами секретного повідомлення. Зміна LSB зазвичай непомітна для людського ока, але достатня для приховування значної кількості інформації.[7] Стеганографія на основі перетворень (DCT, DWT): Ці методи основані на математичних перетвореннях, таких як дискретне косинусне перетворення (DCT) або дискретне вейвлет–перетворення (DWT), для вбудовування секретного повідомлення в спектр контейнера.[8] Стеганографія на основі розріджених кодувань: Ці методи використовують розріджені кодування, для вбудовування секретного повідомлення в кодифіковані дані контейнера.[9]

Системи моніторингу та управління мережевими ресурсами використовують для протидії системам моніторингу та управління мережевими ресурсами в області діяльності промислового шпигунства дозволяє запобігти спробам контролю над інформаційним простором, в той момент коли дані проходять через сервери керування локальних і глобальних обчислювальних мереж. Для реалізації обходу систем моніторингу та управління мережевими ресурсами, як приклад можна навести деякі методи: Стеганографія в TCP/IP пакетах: Цей метод використовує стеганографічні методи для вбудовування

секретного повідомлення в заголовки або корисне навантаження TCP/IP пакетів. Стеганографія в голосових сигналах: Цей метод використовує стеганографічні методи для вбудовування секретного повідомлення в голосовий сигнал.

Ще одним важливим завданням стеганографії є приховування програмного забезпечення (ПЗ). Для того, щоб уникнути використання ПЗ незареєстрованими користувачам, його можна закамуфлювати під стандартні універсальні програмні продукти, наприклад, текстові редактори, або приховати у файлах мультимедіа, таких як звукові ефекти комп'ютерних ігор.

Захист авторських від піратства також є однією з важливих сфер застосовується стеганографія. Ця стеганографія передбачає те, що на комп'ютерні графічні зображення наносяться спеціальні мітки, котрі є невидимими для звичайного користувача, але можуть бути розпізнані за допомогою спеціального програмного забезпечення. Такі технології вже використовуються в комп'ютерних версіях деяких журналів. Цей напрям стеганографії застосовується не лише до обробки зображень, але і до аудіо та відеофайлів з метою захисту інтелектуальної власності. [6] Для реалізації захисту права на інтелектуальну власність, як приклад можна навести деякі методи: Цифрові водяні знаки (ЦВЗ): Це один з методів стеганографії, який використовується для того, щоб приховати спеціальну мітку у цифровий вміст, наприклад, у зображення, аудіо або відео, щоб ідентифікувати власника контенту або відстежувати його поширення. Наприклад, алгоритм DCT-Based Watermarking використовує перетворення косинусів для вбудовування водяного знака у зображення, а Spread Spectrum Watermarking розподіляє спектральні компоненти по всьому зображенню, зробляючи його стійким до різних атак. [25] Стеганографія з фрагментами зображень: Цей підхід використовує унікальні фрагменти зображень для того, щоб вбудувати секретне повідомлення в зображення. Це може бути корисним для захисту авторських прав на зображення.До прикладу такого методу можна віднести алгоритм Patchwork Steganography, який Розбиває зображення на невеликі фрагменти та використовує унікальні патерни цих фрагментів для вбудовування інформації.

Також можна сказати про алгоритм Pixel-Value Differencing (PVD), який бузється на оцінкі різниць між сусідніми пікселями у зображенні та використанні інформації для вбудовування прихованого повідомлення. [11] Стеганографія з аудіопозначками: Цей метод використовує аудіопозначки, які є унікальними для певного аудіофайлу, для вбудовування секретного повідомлення в цей аудіофайл. Це може бути корисно для захисту авторських прав на аудіозаписи. До прикладу такого методу можна віднести алгоритм Phase Coding, котрий Використовує фазову інформацію аудіосигналу для вбудовування стеганографічного повідомлення. [12]

1.3 Стеганографічні системи та їх властивості

Стеганографічна система, або стегосистема - це комплекс засобів та методів, що використовуються для створення прихованого каналу передачі інформації. При розробці стегосистеми необхідно враховувати деякі основні аспекти, а саме: безпека стегосистеми ґрунтується на принципі, що противник знає всі деталі алгоритму приховування, крім секретного ключа, який дозволяє виявити та витягнути приховане повідомлення.

Якщо противник навіть дізнається про наявність прихованого повідомлення, це не повинно дозволити йому отримати аналогічні повідомлення в інших даних, поки ключ залишається в секреті. Потенційний противник повинен бути позбавлений будь-яких технічних або інших переваг у виявленні або розкритті змісту таємних повідомлень. Узагальнена модель стегосистеми зображена на рисунку 1.1.

Рисунок 1.1 – Узагальнена модель стеганосистеми

Узагальнена модель стеганосистеми відображає складність процесу приховування інформації та складається з деяких елементів, розглянемо більш детально: Дані та повідомлення: Цей елемент відповідає за саму інформацію, яку слід приховати. Повідомлення може бути у будь-якому форматі, наприклад: текстовий файл, зображення, відео або аудіо; Контейнер: Це носій інформації, у який вбудовується таємне повідомлення. Контейнер може бути будь-якого типу, такого як зображення, аудіофайл, відео або навіть текстовий документ. Сутність контейнера полягає в тому, щоб приховане повідомлення було непомітним для користувачів; Порожній та заповнений контейнер: Порожній контейнер - це такий, що не містить прихованого повідомлення. Він може бути використаний як базовий елемент для приховування інформації. Заповнений контейнер містить вбудоване повідомлення і призначений для передачі отримувачеві; Вбудоване (приховане) повідомлення: Це саме таємне повідомлення, яке приховується в контейнері. Воно може бути вбудоване за допомогою різних стеганографічних методів, які забезпечують надійне приховування без помітної зміни в оригінальному контейнері; Стеганографічний канал: Це деяке середовище, яке може буде або відкритим, або захищеним, чрезе яке відбувається передача стеганоконтейнера від відправника до отримувача; Стеганоключ: Це такий секретний параметр, який використовується для приховування або вилучення секретної інформації з контейнера. Зазвичай їх використовують для захисту систем від несанкціонованого доступу та забезпечення конфіденційності прихованої інформації. Також, слід зазначити, що залежно від рівня безпеки може бути використано один або кілька стеганоключів. [13]

За аналогією із криптографією, тип ключа спричиняє існування двох типів стеганосистем, а саме системи з секретним ключем, системи з відкритим ключем.

Системи з секретним ключем: Ці системи використовують один секретний ключ, який повинен бути відомий як відправнику, так і одержувачу перед початком обміну повідомленнями. Основний принцип роботи виглядає так, що відправник використовує секретний ключ для вбудовування секретного повідомлення в контейнер, створюючи стеганоконтейнер. Одержувач, маючи

той самий секретний ключ, може вилучити секретне повідомлення з стеганоконтейнера. До переваги такої системи можна віднести те, що системи з секретним ключем зазвичай простіші у реалізації, ніж системи з відкритим ключем. При правильному збереженні секретного ключа системи з секретним ключем можуть бути дуже стійкими до атак. В противагу також існують деякі недоліки таких систем, а саме: секретний ключ повинен бути безпечно переданий відправнику одержувачу по захищеному каналу зв'язку. Це може бути проблемою, якщо відправник і одержувач не мають прямого зв'язку або не довіряють один одному. Якщо секретний ключ буде викрадений, зловмисник зможе читати всі повідомлення, які передаються через цю систему. [6]

Системи з відкритим ключем: Ці системи використовують два ключі: Відкритий ключ, який може бути відомий всім і використовується для вбудовування секретного повідомлення в контейнер. Секретний ключ, який відомий лише одержувачу і використовується для вилучення секретного повідомлення з стеганоконтейнера. Основний принцип роботи виглядає так, що відправник використовує відкритий ключ одержувача для вбудовування секретного повідомлення в контейнер. Одержувач використовує свій секретний ключ для вилучення секретного повідомлення з стеганоконтейнера. Основною перевагоє такої системи є те, що відкритий ключ можна вільно передавати по незахищеному каналу зв'язку. Також системи з відкритим ключем можуть використовуватися навіть при взаємній недовірі відправника і одержувача. Але як недоліки такої системи виділяють те, що: системи з відкритим ключем зазвичай складніші у реалізації, ніж системи з секретним ключем та менш стійкі до атак, ніж системи з секретним ключем.[6]

Для забезпечення ефективності стеганографічної системи вона повинна мати деякі ключові властивості [14-16]:

Непомітність (invisibility) – при аназілі стеганооб’єкта сторонньою особою приховане повідомлення не повинно бути помітним. Це повинно досягається шляхом зменшення візуальних відмінностей між вбудовуваним об'єктом та стеганооб'єктом.

Захищеність (security) – За допомогою стійких алгоритмів наше приховане повідомлення також повинно бути захищеним від зловмисників, котрі прагнуть отримати несанкціонований доступ до нашої інформації.

Стійкість (robustness) - стеганооб'єкт повинен зберігати приховану інформацію навіть після застосування певних перетворень, таких як стиснення, фільтрація, геометричні перетворення або навмисні спотворення. Існує доволі велика кількість алгоритмів котрі мають свої переваги та недоліки, тому забезпечити баланс стійкості доволі важко. Наприклад метод псевдовипадкової перестановки моє високу вразливість до модифікацій контейнера та має відносно велику кількість операцій для вбудовування і витягнення. Чи метод Куттера-Джордана-Боссена який відносно сильно спотворює зображення тому має низьку стійкість до стегоаналізу. Враховуючи важкість досягання максимальної стійності можна сказати, що ця властивість, мабуть, найважливіша, адже вона найбільше відображає сутність стеганографії.

Ємність (capacity) - система повинна забезпечувати достатню ємність для приховування необхідного обсягу інформації без істотного погіршення якості вбудовуваного об'єкта.

Більшість сучасних методів, які використовуються для приховування повідомлень у цифрових контейнерах, показують, що надійність системи знижується зі збільшенням обсягу вбудованих даних, що схематично зображено на рисунку 1.2. (за незмінного розміру контейнера). Це означає, що використання певного контейнера встановлює обмеження на обсяг вбудованих даних.

Рисунок 1.2 – Залежність надійності системи від обсягу вбудованих даних

2. ОСНОВИ СТЕГАНОГРАФІЇ ТА ЦИФРОВИХ ЗОБРАЖЕНЬ

2.1. Класифікація методів стеганографії

Більшість методів стеганографії ґрунтується на двох принципах. По-перше, файли, які не потребують абсолютної точності, можна трохи змінювати без втрати їхньої функціональності, хоча ці зміни не повинні сильно відрізняти контейнер від оригіналу. По-друге, людські органи чуття не можуть надійно розрізняти невеликі зміни в таких модифікованих файлах, і немає спеціальних інструментів для цього.

Існує кілька напрямків у комп'ютерній стеганографії, де контейнером є файл:

1) Пов'язаний з цифровою обробкою сигналів (цифрова комп'ютерна стеганографія). Цей напрямок є найбільш популярним і представлений кількома десятками програм (S-Tools, CryptArkan, Stegomagic, The Third Eye, Hermetic Stego, WeavWav, Gifshuffle, JSteg Shell, MP3Stego). В якості контейнерів використовується інформація аналогового характеру (піксельні дані у BMP, коефіцієнти DCT JPEG, фаза або амплітуда в аудіофайлах і т.д.). Приховування здійснюється за рахунок наявності похибки цифрування та особливостей сприйняття інформації органами чуття людини.
2) Пов'язаний з використанням особливостей форматів зберігання/передачі даних (форматна або структурна комп'ютерна стеганографія). Поширені структури зберігання та передачі цифрової інформації мають резервні поля або дозволяють інкапсулювати дані всередині себе без втрати функціональності (резервні байти в BMP, прозорі частики PNG, або метадані EXIF і т.д.).
3) Використовуючий текст як контейнер (цей напрямок можна віднести до комп'ютерної стеганографії з напруженням, оскільки текст сприймається людиною безпосередньо – з необов'язковим залученням комп'ютерної системи). Існують два основних підходи до вбудовування

інформації в текстовий контейнер. Перший передбачає використання семантичних особливостей мови (Tyrannosaurus lex, TextHide). Наприклад, у тексті виконується пошук слів, які мають певний набір синонімів. Потім, відповідно до прихованого повідомлення, виконується заміна знайдених слів на відповідні їм синоніми, що не спотворює смисловий зміст. Другий підхід полягає в генерації штучного тексту (Nicetext, Texto, Markov-Chain-Based).

Існуючі методи стеганографії можна класифікувати вибираючи той чи інший класифікаційний критерій (рис 2.1) [1, 17-22]:

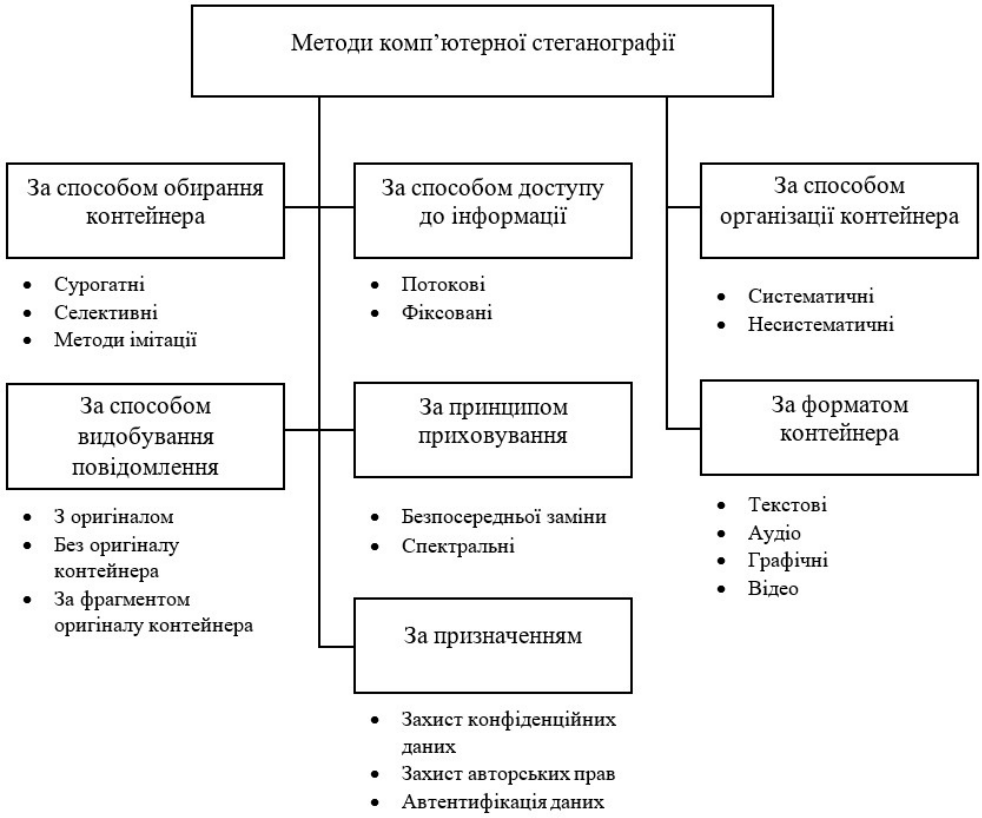


Рисунок 2.1 – Класифікація методів комп'ютерної стеганографії

Щодо способу обирання контейнеру, то загалом можна виділити сурогатні, селективні та імітаційні методи стеганографії [26].

Щодо сурогатних або безальтернативних методів, то однією з особливостей є те, що вони не надають вибору контейнера, тому для приховування нашого повідомлення використовується перший доступний контейнер (ерзац-контейнер). Зазвичає цей контейнер не є оптимальним

рішенням для приховування повідомлення, тому цей спосіб має свої недоліки. Наприклад може статися так, що розмір або якість контейнера не відповідатимуть потребам приховуваного повідомлення, що скоріш за все призведе до втрати частини інформації або погіршить якість носія, що дозволить розпізнати втручання в контейнер.

Селективні методи, навпаки, передбачають ретельний вибір контейнера для нашого прихованого повідомлення, таким чином, щоб повідомлення відповідало статистичним характеристикам шуму контейнера. Реалізація цього способу виконується шляхом відбракування альтернативних контейнерів, які генеруються у великій кількості. Таким чином знаходиться найбільш оптимальний контейнер для нашог повідомлення. Одним із різновидів селективного підходу є використання хеш-функцій. Хеш-функція обчислюється для кожного контейнера, а потім обирається той, чия хеш-функція збігається з хеш-функцією повідомлення. Тобто, можна сказати, що стеганограмою є обраний контейнер.

Імітаційні методи стеганографії йдуть ще далі, генеруючи контейнер самою системою. Цей метод пропонує найвищий рівень стійкості, але потребує значних ресурсів та складної реалізації. Існує декілька різновидів реалізації, наприклад, коли шум контейнера може імітуватися приховуваним повідомленням. Це здійснюється завдяки використанню процедур, які кодують приховане повідомлення під шум та зберігають модель початкового шуму. У крайньому випадку, на основі моделі шуму може створюватися ціле повідомлення. Прикладом може стати программа MandelSteg [21], яка в якості контейнера генерує фрактал Мандельброта, або ж апарат функцій імітації [22].

За способом доступу до приховуваної інформації можна виділити два основних типи контейнерів: потокові та фіксовані. Потоковий контейнер представляє собою деяку послідовність бітів, що безперервно змінюються. Таким чином кодеру наперед невідомо, чи достатньо розміру контейнера для передачі всього повідомлення. Інтервали між вбудованими бітами визначаються за допомогою генератора псевдовипадкової послідовності (ПВП) із рівномірним

розподілом інтервалів між відліками. Основною проблемою потокових контейнерів є необхідність забезпечення синхронізації для визначення початку та кінця послідовності вбудованих даних. Водночас, складність реалізувати синхронізацію є перевагою з точки зору забезпечення прихованості передачі, але нажаль на практиці стеганосистеми з потоковими контейнерами майже не представлені. На відміну від потокових контейнерів, фіксовані контейнери характеризуються тим, що їх розміри та заздалегідь відомими, що дозволяє виконувати вкладення даних в контейнер більш оптимальним чином. Також для цих контейнерів немає необхідності в організації синхронізації, як це потрібно для потокових контейнерів.

За способом організації контейнери можуть бути систематичними і несистематичними. У контейнерах з систематичною організацією можна чітко вказати місця, де знаходяться біти самого контейнера та біти шуму, котрі призначені для приховування нашої секретної інформації. Як приклад зображення у форматі BMP, де останні біти кольорових компонентів кожного пікселя можуть бути використані для вбудовування даних. Ці останні біти є чітко визначеним місцем для "шуму". Щодо несистематичної організації контейнера, то у такому випадку обробляється вміст всієї стеганограми, адже такий поділ, як у систематичній організації, просто неможливий.

За використовуваним принципом приховування методи комп’ютерної стеганографії поділяють на два основних класи: методи безпосередньої заміни і спектральні методи. Методи безпосередньої заміни використовують надлишкові дані у вихідному файлі, який слугує контейнером для приховування. Наприклад, для зображень це можуть бути найменш значущі біти кольорових компонентів пікселів, які не дуже впливають на візуальне сприйняття. Для аудіофайлів надлишковими є біти, що не використовуються для кодування звуку. Секретне повідомлення вбудовується шляхом заміни цих надлишкових елементів, які сприймаються як "шум" для людського сприйняття. Таким чином, на зображенні чи в аудіо здійснюються незначні зміни, однак візуально чи на слух вони залишаються практично ідентичними до оригіналу. Спектральні методи

засновані на використанні спектральних представлень елементів середовища, куди вбудовуються приховувані дані. Ці методи основані на математичних перетвореннях, таки як дискретне косинусне перетворення (ДКП), перетворення Фур'є, Карунена-Лоєва, Адамара, Хаара тощо.

За призначенням стеганометоди розрізняють на дві основні категорії: для прихованого передавання або збереження даних та для захисту авторських прав. Методи першої категорії використовуються для передачі або зберігання конфіденційної інформації, залишаючи її існування непомітним для сторонніх. Наприклад, метод найменш значущого біту (LSB) змінює найменш значущі біти пікселів зображення для приховування повідомлень. Методи другої категорії, використовуються для захисту авторських прав і підтвердження автентичності цифрових об'єктів, таких як зображення, відео або аудіо, шляхом приховування інформації, яка може бути виявлена за допомогою спеціальних алгоритмів. Найпоширенішим прикладом цієї категорії є цифрові водні знаки.

Враховуючи класифікацію методів комп’ютерної можна сформувати деякий алгоритм організації приховування нашого секретного повідомлення в деякий контейнер. (рис.2.2)

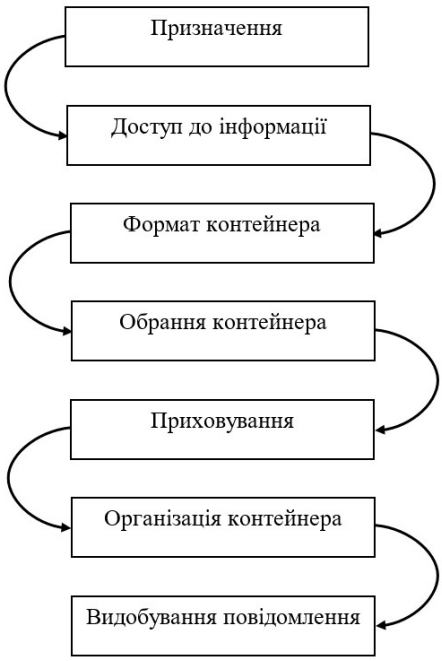


Рисунок 2.2 – Стеганографічний алгоритм приховування повідомлення

2.2. Особливості зорової системи людини (ЗСЛ).

Зорова система людини (ЗСЛ) є складним та багатоетапним процесом, що включає фізіологічні та психофізіологічні характеристики. Усі властивості ЗСЛ умовно поділяють на дві групи: низькорівневі («фізіологічні») та високорівневі («психофізіологічні») . Протягом тривалого часу дослідники зосереджувалися переважно на вивченні низькорівневих властивостей зору. Однак, останніми десятиліттями набуває популярності тенденція врахування також високорівневих характеристик ЗСЛ при розробці стеганографічних алгоритмів. [1]

До найважливіших низькорівневих властивостей, що впливають на помітність стороннього шуму в зображеннях, належить доволі таки велика кількість осболивостей.

Чутливість до зміни яскравості характеризується здатністю ЗСЛ розрізняти зміни в контрасті зображення. Як видно на рис. 2.3, для середнього діапазону зміни яскравості контраст залишається приблизно постійним, тоді як для малих і великих яскравостей значення порогу нерозрізненості зростає. Новітні дослідження суперечать класичній теорії, вказуючи на те, що при малих значеннях яскравості поріг нерозрізненості зменшується, тобто ЗСЛ стає більш чутливою до шуму в цьому діапазоні. [1]

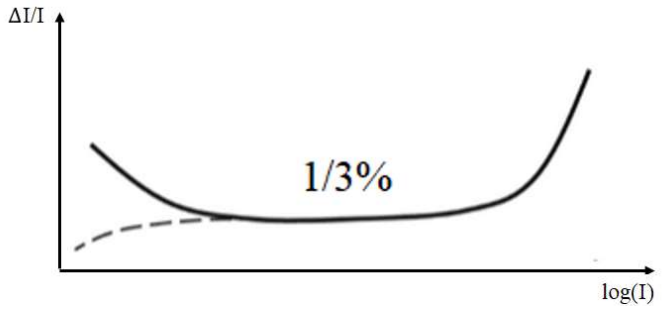


Рисунок 2.3 – Чутливість до зміни контрасту і поріг нерозрізненості ΔI

Частотна чутливість зорової системи людини проявляється в тому, що ми більш чутливі до низькочастотного шуму, ніж до високочастотного. Це пов'язано з нерівномірністю амплітудно-частотної характеристики зорової системи. Елементи зорової системи розщеплюють відеосигнал на деякі окремі складові,

які стимулюють нервові закінчення в оці через різні субканали. Ці елементи відрізняються за просторовими та частотними характеристиками, а також за їх просторовою орієнтацією, яка може бути горизонтальною, вертикальною або діагональною. При одночасному впливі на око двох складових із подібними характеристиками збуджуються одні й ті самі субканали, що призводить до ефекту маскування. Це означає, що адитивний шум є набагато помітнішим на НЧ ділянках зображення, ніж на ВЧ, де спостерігається маскування. Ефект маскування найбільш виражений, коли обидва сигнали мають однакову орієнтацію та місце розташування .

Ефект маскування у просторовій області можна пояснити через побудову ймовірнісних моделей зображення, де останнє представляється у вигляді марківського випадкового поля з розподілом імовірностей, описаним узагальненим ґаусівським законом. Частотна чутливість тісно пов'язана з яскравісною, і відомий вираз для визначення порогу маскування на основі яскравісної чутливості дозволяє знаходити метрику спотворення зображення, яка враховувала б властивості ЗСЛ. Такі математичні моделі ефективно застосовуються у стандарті JPEG, де квантуються коефіцієнти дискретного косинусного перетворення зображення. [1]

Високорівневі властивості ЗСЛ рідко враховуються при побудові стеганографічних алгоритмів. Вони є побічним продуктом, який проявляється вторинно: після обробки первинної інформації від ЗСЛ людський мозок надсилає команди щодо додаткового налаштування під зображення. До основних високорівневих властивостей належать:

- Чутливість до контрасту: Більш контрастні ділянки зображення та істотні перепади яскравості привертають більше уваги.
- Чутливість до розміру: Порівняно з меншими, більші за розміром ділянки зображення є більш помітними. Існує певний поріг насичення, після якого подальше збільшення розміру вже не відіграє ролі.
- Чутливість до форми: Довгі й тонкі об'єкти привертають значно більшу увагу, ніж закруглені та однорідні.

- Чутливість до кольорів: Деякі кольори, такі як червоний, є помітнішими за інші. Цей ефект посилюється, якщо фон відрізняється від кольорів фігур на ньому.
- Чутливість до місця розташування: Людина схильна розглядати центр зображення в першу чергу, а вже потім — периферію. Також уважніше розглядаються фігури переднього плану, ніж заднього.
- Чутливість до зовнішніх подразників: Рух очей спостерігача залежить від конкретної обстановки, отриманих перед переглядом або під час нього інструкцій, додаткової інформації тощо.

Розуміння цих характеристик є важливим для розробки методів приховування даних у цифрових зображеннях. Це дозволяє класифікувати їх на різні типи методів, такі як методи заміни в просторовій (часовій) області, методи приховування в частотній області, широкосмугові методи, статистичні (стохастичні) методи, методи спотворення та структурні методи.

2.3 Цифрові формати нерухомих зображень (BMP, JPEG, GIF, PNG).

BMP (Bitmap Image): BMP (або Bitmap) - це формат зображень, який відомий своєю безкомпресійною природою, що означає, що він зберігає всі деталі оригіналу, але при цьому може мати значний розмір файлу. BMP не використовує компресію, що дозволяє зберігати всі деталі оригінального зображення. Також слід зазначити, що BMP підтримується багатьма програмами та пристроями, що робить його універсальним для використання. Оскільки BMP не використовує компресію, розмір файлу може бути дуже великим, особливо для зображень високої роздільної здатності або з великою кількістю кольорів, що може бути не ефективним для зберігання інформації в цьому форматі. Я широко використовуються у сфері медицини для зберігання рентгенівських знімків.

JPEG (Joint Photographic Experts Group) - це популярний формат для зберігання фотографій та подібних зображень. Він використовує втратне стиснення, що дозволяє отримувати менші розміри файлів порівняно з форматами, які використовують безвтратне стиснення. JPEG ефективний для

стиснення зображень з реалістичними сценами та плавними колірними варіаціями. [24]

Щодо процесу стиснення формату JPEG, котрий описаний у першій частині стандарту JPEG, то він складається з нижче описаних етапів (рис 2.4):

У розглянутому описі використано сімейство колірних просторів YCbCr. Перетворення RGB в YCbCr робиться для того, щоб розділити зображення на складові яркості (Y) та кольоровості (Cb і Cr).

Зменшення просторової роздільної здатності (Subsampling): Цей етап є необов'язковим і може використовуватися для зменшення розміру файлу. При проріджуванні зменшується роздільна здатність компонентів Cb і Cr, що призводить до втрати деталей кольору. Блокування: Кожна компонента (Y, Cb і Cr) розбивається на блоки 8x8 пікселів. Зсув: Вибірки в кожному блоці зсуваються вниз на 128, припускаючи глибину 8 біт. Дискретне косинусне перетворення (ДКП): До кожного блоку застосовується ДКП. ДКП перетворює просторові частоти зображення в частотні домени. Квантування: Коефіцієнти ДКП квантуються, тобто округляються до набору дискретних рівнів. Це призводить до втрати даних, але також і до значного зменшення розміру файлу. Важливо зазначити, що людська зорова система більш імунна до втрати високочастотних компонентів, ніж до втрати низькочастотних компонентів. Це дозволяє квантуванню значно зменшити обсяг інформації в високочастотних компонентах. Після квантування, квантовані коефіцієнти ДКП перед кодуванням Хаффмана або арифметичним кодуванням перетворюються в послідовність за методом "зигзаг". Це робиться для того, щоб групувати подібні за значенням коефіцієнти, що покращує ефективність кодування. Ентропійне кодування: Квантовані коефіцієнти ДКП кодуються за допомогою кодування Хаффмана або арифметичного кодування.

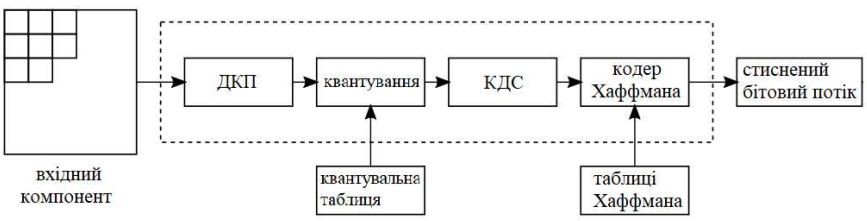


Рисунок 2.4 – Ланцюжок стиснення JPEG

Щодо переваг даного формату, то можна виділити те, що JPEG використовує алгоритм стиснення з втратами, який видаляє деякі дані зображення для зменшення його розміру. Це робить JPEG ідеальним форматом для зберігання та обміну фотографіями в Інтернеті, оскільки він дозволяє економити місце на диску та пропускну здатність. JPEG підтримується практично всіма програмами перегляду зображень та веб-браузерами, що робить його зручним форматом для обміну фотографіями з іншими. JPEG добре стискає зображення з плавними переходами кольорів і градієнтами, такі як фотографії людей, пейзажів та натюрмортів. JPEG пропонує широкий спектр параметрів стиснення, які дозволяють користувачам контролювати рівень якості та розмір файлу зображення. Крім того, JPEG підтримує різні перетворення, такі як обертання та зміна розміру, що робить його універсальним форматом для редагування фотографій.[23]

Напротивагу позитивним аспектам даного формату, також існують і відповідні недоліки. Алгоритм стиснення JPEG видаляє деякі дані зображення, що може призвести до втрати якості, особливо на ділянках з високим контрастом або дрібними деталями. Чим більше стискається зображення, тим більше втрачається якість. JPEG не підходить для стиснення зображень з різкими краями або текстом, оскільки стиснення може призвести до появи артефактів, таких як блочні шуми та кільця навколо країв. Ці артефакти можуть бути помітні на фотографіях з високим ступенем стиснення.

Формат GIF, подібно до формату JPEG, є доволі застарілим типом файлів і, як правило, пов'язаний з Інтернетом, а не з фотографією. GIF означає "Graphics Interchange Format" і використовує ту саму безвтратну компресію LZW (Lempel–Ziv–Welch — універсальний алгоритм стиснення даних без втрат), що і зображення у форматі TIFF. Ця технологія колись була предметом суперечок (через проблеми з патентами), але все ж таки стала прийнятим форматом. GIF за своєю природою є файлом з 8-бітною колірною гамою, що означає, що вони обмежені палітрою з 256 кольорів, які можна вибрати зі стандартної моделі кольорів RGB. Важливою рисою є те, що зображення в відтінках сірого за своєю

природою є палітрою 8-біт, тому зберігання їх у форматі GIF є досить ідеальним. Окрім підтримки прозорості, GIF також підтримує анімацію, обмежуючи кожний кадр до 256 попередньо вибраних кольорів. [23]

Хоча GIF не має таких втрат інформації, як JPEG, конвертація до 8-бітної колірної гами спотворює багато зображень, використовуючи дитер-фільтри для оптичного змішування або "дифузії" кольорів, схожих на полутонові крапки або пунктилізм. Це може радикально змінити зображення до гіршого або, у деяких випадках, бути використаним для створення цікавого ефекту. GIF може бути використаний для збереження чітких ліній у типографії та геометричних формах, хоча й ці речі краще підходять для векторних графічних файлів, таких як SVG. GIF не є ідеальним для сучасної фотографії чи зберігання зображень. У випадку дуже обмеженої кількості кольорів та малих розмірів, GIF-зображення може бути меншим за файли у форматі JPEG. Проте для більшості звичайних розмірів стиснення JPEG створюватиме менше зображення. Загалом, сам формат GIF досить застарілий і корисний лише для створення рухливих малюнків або іноді для створення приблизно прозорих ефектів. [23]

PNG (Portable Network Graphics) - це растровий формат зображень, розроблений для заміни GIF. Основна перевага PNG полягає в тому, що він використовує безвтратну компресію по алгоритму Deflate, що дозволяє легко відновлювати оригінальне зображення після стискання. На відміну від GIF, PNG підтримує 8- і 24-бітну кольорову гаму для більш точного відтворення кольорів. До основних переваг формату PNG можна віднести те, що він використовує безвтратне стиснення, що робить PNG ідеальним для зображень, де важлива чіткість та деталізація. Також від підтримує альфа-канал, який дозволяє зберігати прозорість зображення. Однак PNG має свої недоліки, зокрема, він може генерувати більші розміри файлів порівняно з іншими форматами, такими як WebP і JPEG. Наприклад, WebP може бути на 45% менше за PNG, а JPEG - на 28%. [23]

3. МЕТОДИ ПРИХОВУВАННЯ ДАНИХ У НЕРУХОМИХ ЗОБРАЖЕННЯХ

3.1Метод заміни молодших значущих біт

Метод заміни найменш значущого біта (НЗБ, LSB - Least Significant Bit) є найпоширенішим серед існуючих методів стеганографічного приховання інформації у просторовій області. [22]

Наймолодший значущий біт у кодовій комбінації градації яскравості пікселя зображення, який дорівнює кроку квантування, можна прирівняти до шуму квантування. За достатньої глибини дискретизації зображення (наприклад, у моделі RGB кожен канал має 256 рівнів квантування), зміни в цьому біті зазвичай непомітні для людського ока. Оскільки набір НЗБ зображення фактично є шумом, його можна використовувати для перенесення стеганографічних бітів.

У зображеннях у градаціях сірого, де кожен піксель кодується одним байтом, обсяг вбудованих даних може становити до 1/8 від загального обсягу контейнера. Наприклад, у зображенні розміром 512x512 пікселів (приблизно 256 КіБ) можна вбудувати близько 32 КіБ інформації через заміну НЗБ. Якщо змінювати не один, а два молодших біти в кожній комбінації яскравості, приховану пропускну здатність можна подвоїти, збільшуючи обсяг вбудованих даних до 64 КіБ, що також залишається практично непомітним для ока. [22]

Популярність методу заміни НЗБ пояснюється його простотою та здатністю приховувати великі обсяги інформації в невеликих файлах (створений прихований канал зв'язку має пропускну здатність від 12,5 до 25%, і при цьому пустий і заповнений контейнери майже не відрізнятимуться один від одного). Зазвичай цей метод використовується для растрових зображень у форматах без втрат (наприклад, GIF, BMP, PNG і т. д.). [22]. Головним недоліком методу НЗБ є його слабка стійкість до стеганалізу як пасивними, так і активними атаками, що призводить до високої чутливості до навіть найменших змін у контейнері. У певній мірі для послаблення цієї чутливості додатково може бути застосоване попереднє завадостійке кодування приховуваного повідомлення.

Основка суть методу стеганографії LSB полягає у тому, що біти коду секретного повідомлення встановлюють замість молодших бітів (останніх, на таб. 3.1) в байтах, які відповідають за кодування кольорів. Така незначна зміна колірної гами пікселя при заміні останнього біта палітри на «0» або «1» буде практично непомітною для людського ока.

Подання байтів, які відповідають за кодування кольору до внесення стеганограми (див. таб. 3.1).

Таблиця 3.1 – Подання байтів до внесення стеганограми

Channel	Bits								Decimal	Color
	1	2	3	4	5	6	7	8		
R	0	0	1	1	1	1	1	1	63	
G	1	1	1	0	1	1	0	1	237	
B	0	0	0	0	1	1	1	0	14	

Подання байтів, які відповідають за кодування кольору після внесення стеганограми в останні значущі біти (таб. 3.2).

Таблиця 3.2 – Подання байтів після внесення стеганограми

Channel	Bits								Decimal	Color
	1	2	3	4	5	6	7	8		
R	0	0	1	1	1	1	1	0	62	
G	1	1	1	0	1	1	0	0	236	
B	0	0	0	0	1	1	1	1	15	

Максимальна ємність 24-бітного BMP-файлу як стегоконтейнера при використанні останніх бітів усіх байтів, що кодують три кольори, становить одну восьму від обсягу файлу.

Наприклад, у файл зображення розміром 1280 на 720 пікселів можна вбудувати великий текст. Розрахунок: 1280 * 720 * 3 * 1/8 = 345600 байтів тексту. Якщо використовувати лише останні біти байтів, відповідальних за синій колір (Blue), ємність файлу знизиться втричі, але виявити стеганограму буде практично неможливо, оскільки людське око найгірше розрізняє відтінки синього.

3.2Метод псевдовипадкового інтервалу

У найпростішому випадку замінюються НЗБ всіх пікселів послідовно. Альтернативний підхід – метод псевдовипадкового інтервалу [22], де біти секретного повідомлення розподіляються випадковим чином по контейнеру. Відстань між вбудованими бітами визначається псевдовипадково. На відміну від послідовного методу, процес починається з певного пікселя, але далі біти розміщуються через випадкові інтервали.

Цей підхід особливо корисний, коли довжина секретного повідомлення набагато менша за кількість пікселів. Простіший варіант цього методу використовує функцію координат попереднього пікселя для визначення інтервалу між бітами. Перевагою цього методу є підвищена стійкість до статистичних атак через псевдовипадковий розподіл бітів повідомлення. Недоліком є необхідність передачі додаткових параметрів таких як мітка початку/кінця або ключ який відповідає за інтервал для вилучення.

3.3Метод квантування зображення

До методів приховання в просторовій області відноситься такий метод, як метод квантування зображення [31], заснований на міжпіксельній залежності, що може бути описаною через деяку функцію Θ. Якщо розглянути найпростіший випадк, то різницю можна обчислити різницю ε_i між суміжними пікселями c_i та c_{i+1} (або c_{i-1} та c_i), задавши її в якості параметра функції Θ:

$$\Delta_i = \Theta(c_i - c_{i+1}) \tag{3.1}$$

де Δ_i - дискретна апроксимація різниці сигналів $(c_i - c_{i+1})$

Оскільки Δ_i є цілим числом, а реальна різниця $(c_i - c_{i+1})$ дійсним, то виникають помилки квантування $\delta_i = \Delta_i - \varepsilon_i$. Для сильно корельованих сигналів ця помилка є близькою до нуля.

У даному методі приховування інформації здійснюється шляхом корегування різницевого сигналу Δ_i. Стеганоключ являє собою таблицю, в якій кожному можливому значенню Δ_i ставиться у відповідність визначений біт, наприклад (таб. 3.3):

Таблиця 3.3 – Приклад стеганоключа методу квантування

Δ_i	-4	-3	-2	-1	0	1	2	3	4
k_i	1	0	1	1	0	0	1	0	1

Для того, щоб приховати і-й біта повідомлення слід обчислити різницю дискретної апроксимація сигналів Δ_i. Якщо при цьому значення яскравості k_i відрізняється від секретного біта (який необхідно приховати), то у такому випадку значення Δ_i замінюється найближчим Δ_j для якого ця умова виконується. В результаті значення інтенсивностей пікселів, між якими обраховувалася різниця Δ_i корегується відповідним чином. Видобування секретної інформації відбувається відповідно до значення k_i^*, що відповідає різниці Δ_i^*

3.4Метод Куттера-Джордана-Боссена

Мартін Куттер (M.Kutter), Фредерік Джордан (F.Jordan) та Френк Боссен (F.Bossen) [26] запропонували алгоритм вбудовування до каналу синього кольору RGB-зображення, оскільки саме до нього є найменш чутливою ЗСЛ.

Розглянемо алгоритм передавання окремого біта секретної інформації у запропонованому методі. Нехай m_i – саме той біт, котрий ми хочемо вбудовувати; C={R, G, B} зображення-контейнер; p=(x,y) - псевдовипадково обраний піксель контейнера (канал синього кольору), до якого буде вбудовуватися біт m_i шляхом модифікації яскравості пікселя (3.2):

$$\lambda_{x,y} = 0.29890 \cdot R_{x,y} + 0.58662 \cdot G_{x,y} + 0.11448 \cdot B_{x,y} \qquad (3.2)$$

$$B'_{x,y} = \begin{cases} B_{x,y} - \nu \cdot \lambda_{x,y}, & \text{при } m_i = 0 \\ B_{x,y} + \nu \cdot \lambda_{x,y}, & \text{при } m_i = 1 \end{cases} = B_{x,y} + (2 \cdot m_i - 1) \cdot \nu \cdot \lambda_{x,y} \qquad (3.3)$$

де ν – константа, що визначає енергію вбудовуваного сигналу. Її величина залежить від призначення стеганосистеми. Чим більшою є ν, тим вище стійкість вбудованої інформації до спотворень, але й тим сильнішою буде відмінність заповненого контейнера від оригіналу.

Отримувач витягує біт «наосліп», тобто без доступу до первинного зображення. Це здійснюється шляхом прогнозування значення первинного,

немодифікованого пікселя на основі значень сусідніх пікселів. Для отримання такої оцінки пропонується використовувати значення кількох пікселів, розташованих у тому ж рядку та стовпці графічного контейнера. Наприклад, можна застосувати конфігурацію пікселів у формі «плюса» розміром 7x7 (рис.3.1). У такому випадку оцінка здійснюється за допомогою обчислення наступного виразу (3.4):

$$\hat{B}^*_{x,y} = \frac{1}{4*\sigma} \cdot \left[\sum_{i=-\sigma}^{+\sigma} B^*_{x+i,y} + \sum_{j=-\sigma}^{+\sigma} B^*_{x,y+j} - 2 \cdot B^*_{x,y}\right] \quad (3.4)$$

де σ – це кількість пікселів згори (знизу, ліворуч, праворуч) від оцінюваного пікселя (у випадку «плюса» 7x7 та $\sigma = 3$)

Під час видобування прихованого біта обчислюється різниця δ між поточним та прогнозованим значеннями інтенсивності пікселя. Знак цієї різниці визначає вбудований біт: якщо $\delta < 0$, то m_i = «0»; якщо $\delta > 0$, то m_i = «1». [26] Функції вбудовування та видобування в цьому методі є асиметричними, тобто функція видобування не є зворотною до функції вбудовування. Хоча автори зазначають, що правильність розпізнавання біта за описаною процедурою є високоймовірною, але не стовідсотково точною. [22] Для зниження ймовірності помилок пропонується багаторазово дублювати вбудовування кожного біта. Оскільки кожен біт повторюється N разів, одержувач може отримати N його оцінок. Остаточне значення секретного біта визначається після усереднення різниць між реальним (поточним) та прогнозованим значеннями інтенсивності пікселя в досліджуваному контейнері. Цей алгоритм є стійким до багатьох відомих атак, таких як низькочастотна фільтрація зображення, стиснення за алгоритмом JPEG, обрізання країв тощо.

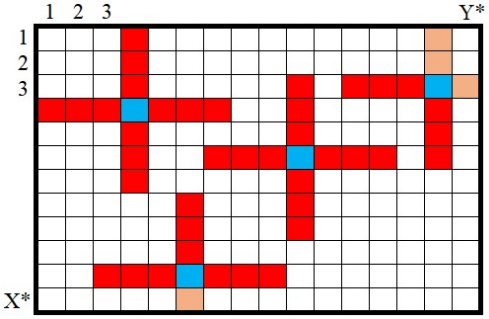


Рисунок 3.1 – Приклади оцінюваних пікселів і оцінювальних конфігурацій («плюсів» 7x7; $\sigma = 3$)

3.5Метод Коха – Жао

Метод Коха – Жао, розроблений Е. Кохом та Дж. Жао, є одним з найпоширеніших методів приховання конфіденційної інформації у частотній множині зображень. Цей метод базується на зміні величин коефіцієнтів дискретного косинусного перетворення (ДКП) у зображенні. Основна ідея полягає у відносній заміні величин цих коефіцієнтів для приховування даних, зберігаючи при цьому якість зображення.

На початковому етапі, зображення розбивається на блоки 8×8 пікселів, і для кожного блоку застосовується ДКП. Це призводить до формування матриць 8×8 коефіцієнтів ДКП (рис. 3.2). Кожен такий блок призначений для сховування одного біта даних. Існують два варіанти алгоритму: вибір двох або трьох коефіцієнтів ДКП. Для використання цих коефіцієнтів для сховування даних, абоненти повинні заздалегідь домовитися про конкретні координати коефіцієнтів ДКП для кожного блоку. Це забезпечить захист інформації в важливих областях сигналу та збереже дані при JPEG-компресії з малим ступенем стиску.Безпосередньо процес приховання починається з випадкового вибору блоку зображення, призначеного для кодування і-го біта повідомлення. Вбудовування інформації здійснюється наступним чином: для передачі біта "0" різниця абсолютних значень коефіцієнтів ДКП повинна перевищувати певну позитивну величину, а для передачі біта "1" ця різниця повинна бути меншою порівняно з певною негативною величиною. У разі невідповідності відносної величини коефіцієнтів приховуваному біту, вносяться корекції у значення коефіцієнтів. Після внесення необхідних змін у значення коефіцієнтів, які повинні задовольняти встановленим нерівностям, проводиться зворотне ДКП, що дозволяє отримати зображення з прихованою інформацією. Процес витягнення даних у декодері є аналогічним до процесу вбудовування. Виконується вибір тих самих коефіцієнтів, і визначення прихованого біта відбувається на основі порівняння різниці абсолютних значень обраних коефіцієнтів. Якщо різниця перевищує певну позитивну величину, переданий біт

визначається як "0". Якщо різниця менша певної негативної величини, переданий біт визначається як "1".

	1	2	3	4	5	6	7	8
1	-603	203	11	45	-30	-14	-14	-7
2	-108	-93	10	49	27	6	8	2
3	-42	-20	-6	16	17	9	3	3
4	56	69	7	-25	-10	-5	-2	-2
5	-33	-21	17	8	3	-4	-5	-3
6	-16	-14	8	2	-4	-2	1	1
7	0	-5	-6	-1	2	3	1	1
8	9	5	-6	-8	0	3	3	2

Рисунок 3.2 – Приклад масиву коефіцієнтів ДКП

Метод Коха – Жао є досить стійким до компресії зображень, особливо при малих коефіцієнтах стиску JPEG. Однак, чим більше змін вносяться у коефіцієнти, тим більше погіршується якість зображення. Тому необхідно знаходити баланс між стійкістю прихованої інформації до компресії та збереженням якості зображення.

4. ПРАКТИЧНА РЕАЛІЗАЦІЯ СТЕГАНОГРАФІЧНИХ МЕТОДІВ

4.1 Вибір сфери розробки

Java була обрана як основна мова програмування для реалізації методів приховування інформації в цифрових зображеннях. Цей вибір обґрунтований низкою переваг, які надає Java для вирішення поставлених задач у порівнянні з іншими мовами програмування.

Перш за все, Java є об'єктно-орієнтованою мовою програмування, що дозволяє створювати модульний, легко підтримуваний та розширюваний код. Її об'єктна модель забезпечує інкапсуляцію даних, успадкування та поліморфізм, що спрощує впровадження різноманітних стеганографічних алгоритмів та методів цифрового водяного знаку. Водночас, мова Python, яка також є популярною для наукових розрахунків та обробки даних, не має такої жорсткої об'єктної моделі, що може ускладнити розробку складних систем.

Java є однією з найпопулярніших мов програмування як у минулому, так і на сьогоднішній день. Впродовж багатьох років вона стабільно займала лідируючі позиції в рейтингах використання серед розробників. Згідно із щорічними звітами організації (рис 4.1) TIOBE [27], яка відстежує популярність мов програмування, Java була найпопулярнішою мовою в світі з 2015 по 2019 рік (рис. 4.2). Хоча нині вона поступилася першим місцем мові Python, проте продовжує входити в топ-4 найпоширеніших мов програмування. Java широко використовується у корпоративному середовищі, веб-розробці, для створення мобільних додатків та ігор, а також активно застосовується в наукових дослідженнях та академічній сфері. [27]

Крім того, Java має багатий набір вбудованих бібліотек та інструментів для роботи з графікою та обробки зображень. Зокрема, бібліотека Java Advanced Imaging (JAI) або ImageIO надає потужні засоби для маніпулювання пікселями, фільтрації, перетворення та обробки зображень у різних форматах. Ця функціональність є критично важливою для реалізації методів приховування інформації шляхом модифікації цифрових зображень. Python також має

бібліотеки для роботи з зображеннями, такі як PIL та OpenCV, проте їх функціональність може бути обмеженою порівняно бібліотеками Java.

May 2024	May 2023	Change	Programming Language	Ratings	Change
1	1		Python	16.33%	+2.88%
2	2		C	9.98%	-3.37%
3	4	↑	C++	9.53%	-2.43%
4	3	↓	Java	8.69%	-3.53%
5	5		C#	6.49%	-0.94%
6	7	↑	JavaScript	3.01%	+0.57%
7	6	↓	Visual Basic	2.01%	-1.83%
8	12	⇈	Go	1.60%	+0.61%
9	9		SQL	1.44%	-0.03%
10	19	⇈	Fortran	1.24%	+0.46%

Рисунок 4.1 – Статистика популярності мов програмування [27]

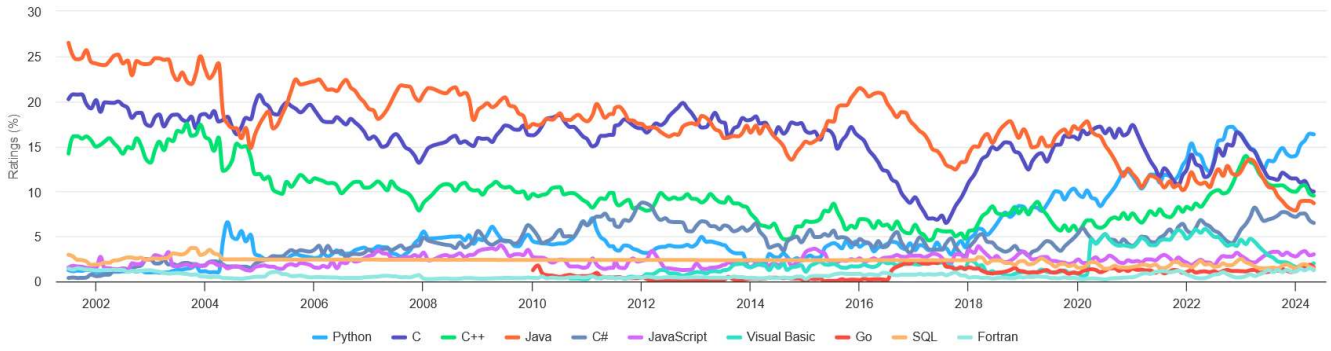


Рисунок 4.2 – Графічна статистика популярності мов програмування з 2002 по 2024 роки. [27]

Важливою перевагою Java є її кросплатформеність. Програми, написані на цій мові, можуть працювати на різних операційних системах, таких як Windows, Linux та macOS, завдяки використанню віртуальної машини Java (Java Virtual Machine, JVM). Це забезпечує широку сумісність та портативність розробленого програмного забезпечення.

Ще однією перевагою Java є наявність величезної екосистеми готових бібліотек та фреймворків, розроблених спільнотою. Завдяки системам управління залежностями, таким як Maven або Gradle, можна легко інтегрувати

ці сторонні бібліотеки у власний проект. Це значно прискорює розробку, дозволяючи використовувати перевірені та оптимізовані рішення для різноманітних задач. Python також має багату екосистему бібліотек, проте деякі з них можуть бути менш оптимізованими або документованими порівняно з Java. Крім того, Java має активну та постійно зростаючу спільноту розробників, що забезпечує постійну підтримку, оновлення та документацію. Це полегшує пошук рішень проблем, обмін досвідом та отримання допомоги під час розробки програмного забезпечення.

У своїй дипломній роботі я використовував стандартні бібліотеки Java для роботи з графікою та обробки зображень, зокрема javax.imageio.ImageIO та java.awt.image.BufferedImage. Ці бібліотеки були обрані через кілька ключових причин: По-перше, javax.imageio.ImageIO забезпечує зручний спосіб завантаження та збереження зображень у різних популярних форматах, таких як JPEG, PNG, BMP та GIF. Це дозволило легко інтегрувати обробку вхідних та вихідних даних у вигляді зображень у мою програму. По-друге, java.awt.image.BufferedImage надає можливість отримувати безпосередній доступ до даних пікселів зображення в оперативній пам'яті, що допомогає в реалізації алгоритмів приховування інформації, які вимагають маніпуляцій з окремими пікселями зображення на низькому рівні. Крім того, я використовував класи java.io.*, такі як BufferedWriter та FileWriter, для роботи з файловою системою та обробки вхідних/вихідних даних у текстовому форматі за необхідності. [28]

Також слід зазначити, що у процесі реалізації методів приховування інформації в цифрових зображеннях я використовував інтегроване середовище розробки (IDE) IntelliJ IDEA. Цей потужний інструмент був обраний через низку переваг, які він надає для програмування на Java. IntelliJ IDEA забезпечує інтелектуальне автодоповнення коду, рефакторинг, навігацію по коду та численні інші функції, що значно підвищують продуктивність розробника. IDE має вбудовані інструменти для налагодження, профілювання, тестування та розгортання додатків, що спрощує весь цикл розробки. Завдяки зручному

інтерфейсу, багатому функціоналу та підтримці сучасних технологій і фреймворків Java, IntelliJ IDEA стала оптимальним вибором для розробки мого програмного рішення (рис. 4.3).

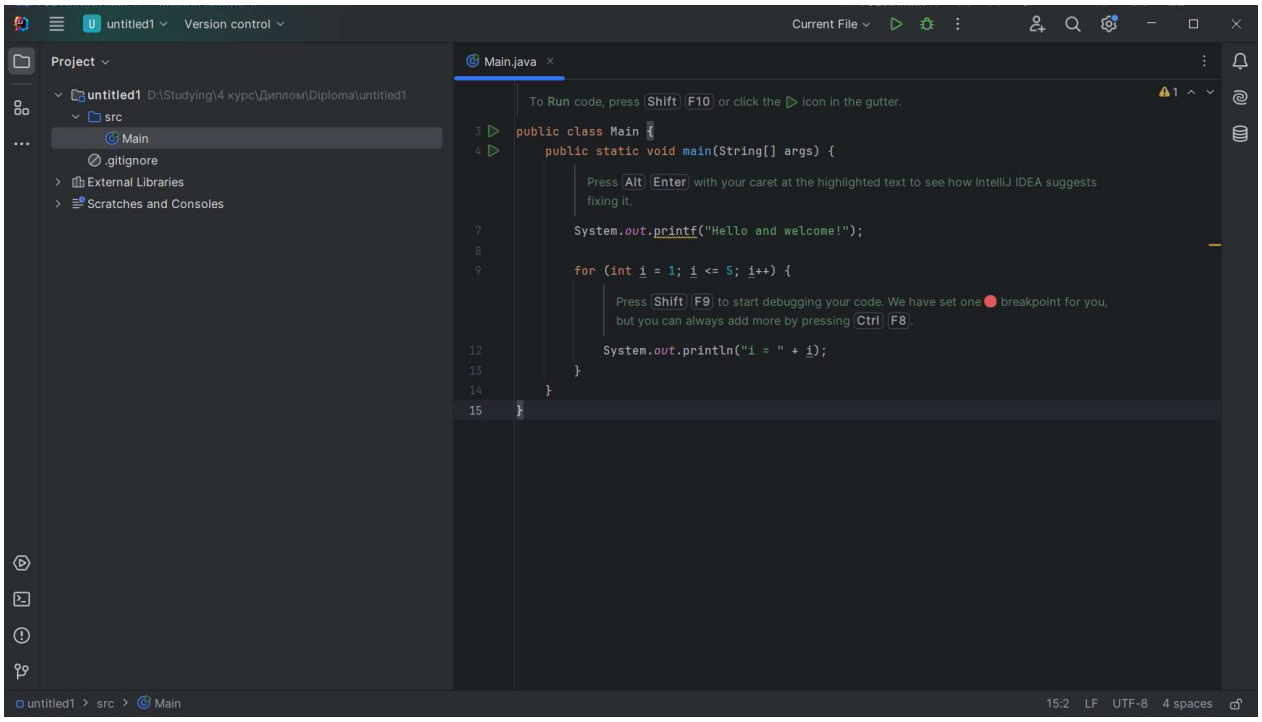


Рисунок 4.3 – Інтерфейс IntelliJ IDEA

4.2Реалізація методу найменш значущих біт

У цьому розділі ми розглянемо реалізацію методу найменш значущих біт (Least Significant Bit, LSB) для приховування інформації в цифрових зображеннях. Цей метод є одним з найпростіших і найпоширеніших методів стеганографії, що дозволяє приховувати текстові дані в пікселях зображення шляхом заміни найменш значущих бітів кожного байту.

Спершу розглянемо допоміжні функції, які забезпечують основні операції з обробки зображень та текстових даних.

saveTextToPath(String text, File file) – ця функція виконує зберігання тексту у вказаний файл. Вона приймає текст котрий слід зберегти та файл. Спочатку виконується перевірка, чи існує файл та у випадку відсутності файлу створюється новий. Після перевірки виконується саме запис текста у наданий файл.

getImageInUserSpace(BufferedImage image) – перетворює зображення у формат, зручний для маніпуляцій. Спочатку створюється новий об’єкт такого ж розміру як і вхідне зображення, але використовується тип BufferedImage.TYPE_3BYTE_BGR котрий являє собою зображення з 8-бітними колірними компонентами RGB, що відповідає колірній моделі BGR у стилі Windows з синім, зеленим і червоним кольорами, які зберігаються у 3 байтах. Використовуючи Graphics2D, вхідне зображення копіюється у нове зображення, яке потім повертається.

getBytesFromImage(BufferedImage image) – отримує масив байтів з зображення. Спочатку виконується конвертація зображення в об'єкт WritableRaster, який представляє растрові дані, а потім отримує з нього буфер даних DataBufferByte. Масив байтів з буферу повертається для подальшого використання.

getBytesFromInt(int integer) – перетворює ціле число на масив байтів. Спочатку виконується ByteBuffer.allocate(bytesForTextLengthData) де створюється новий буфер розміров bytesForTextLengthData (котре є константним і дорівнює 8). Після цього виконується putInt(integer) де передане у функцію число записується у створений буфер. Під кінець повертається масив байтів, який представляє собою оброблюваний буфер.

Далі, розглянемо основні функцій нашого методу LSB. Ці функції відповідають за безпосереднє приховування та витягування тексту в зображеннях.

Відповідно до нижче наведеного коду (лістинг 4.1) можна побачити алгоритм виконання дій для вбудовування тексту у забраження. Спочатку у цій функції виконується зчитування зображення та конвертація у формат придатний до маніпуляцій. Настпунним кроком виконується перетворення зображення у масив байтів, що представляють кожен піксель зображення. Текстове повідомлення та його довжину також перетворюємо у масив байтів. Далі виконується найголовніша дія, а саме кодування довжини тексту та самого зображення.

Кодування відбувається таким чином, що за допомогою функції encodeImage(byte[] image, byte[] addition, int offset) (лістинг 4.2) записується довжина тексту (перші 32 біти) в масив байтів зображення, починаючи з нульової позиції. Це дозволяє при розкодуванні визначити, скільки байтів займатиме прихований текст. Потім записується сам текст, починаючи з позиції bytesForTextLengthData * bitsInByte (тобто пропускаючи перші 32 (4 * 8) біти в котрих ми зберігаємо довжину повідомлення). Останнім кроком цієї функції є збереження зображення з прихованим текстом. Для цього створюється нове ім'я файлу, додаючи суфікс «_with_hidden_messageLSB.png» до оригінального імені файлу. Для цього видаляється розширення файлу (якщо воно є) і додається новий суфікс. Модифіковане зображення з прихованим текстом зберігається у новий файл з цим ім'ям.

Лістинг 4.1 – Реалізація функції кодування тексту у зображення методу LSB

```
      public void encode(String imagePath, String text) {
        BufferedImage imageInUserSpace = null;
        try {
            imageInUserSpace       =       getImageInUserSpace(ImageIO.read(new
File(imagePath)));

            byte imageInBytes[] = getBytesFromImage(imageInUserSpace);
            byte textInBytes[] = text.getBytes();
            byte textLengthInBytes[] = getBytesFromInt(textInBytes.length);

            encodeImage(imageInBytes, textLengthInBytes,  0);
            encodeImage(imageInBytes,                                  textInBytes,
bytesForTextLengthData*bitsInByte);
        }
        catch (Exception exception) {
            System.out.println("Couldn't  hide  text  in  image.  Error:  "  +
exception);
            return;
        }

        String fileName = imagePath;
        int position = fileName.lastIndexOf(".");
        if (position > 0) {
            fileName = fileName.substring(0, position);
        }

        String finalFileName = fileName + "_with_hidden_messageLSB.png";
        System.out.println("Successfully encoded text in: " + finalFileName);
        try {
            ImageIO.write(imageInUserSpace, "png", new File(finalFileName));
        } catch (IOException e) { throw new RuntimeException(e);}

    }
```

Розлянего найголовніший процес даного методу, а саме функція «encodeImage» (лістинг 4.2) котра є ключовою частиною методу LSB для приховування тексту в зображеннях. Спочатку перевіряється, чи достатньо місця в масиві байтів зображення для збереження додаткових даних, враховуючи початковий зсув. Далі виконується саме кодування байтів даних у байти зображення. Перший цикл проходить по кожному байту з даних, коли другий проходить по кожному біту в поточному байті даних, починаючи з найбільш значущого біта. «additionByte >>> bit» зсуває поточний байт на кількість позицій, визначених змінною bit, вправо. «& 0x1» виділяє найменш значущий біт результату зсуву. «image[offset] & 0xFE» (FE в бінарному вигляді = 11111110) очищає найменш значущий біт в байті зображення (застосовується маска 0xFE). «| b» вставляє новий біт в очищену позицію.

Лістинг 4.2 – функція «encodeImage» для приховування тексту методом LSB

```
      private static byte[] encodeImage(byte[] image, byte[] addition, int offset)
{
        if (addition.length + offset > image.length) {
            throw new IllegalArgumentException("Image file is not long enough to
store provided text");
        }
        for (int i = 0; i < addition.length; i++) {
            int additionByte = addition[i];
            for (int bit = bitsInByte-1; bit >= 0; --bit, offset++) {
                int b = (additionByte >>> bit) & 0x1;
                image[offset] = (byte)((image[offset] & 0xFE) | b);
            }
        }
        return image;
    }
```

Далі розглянемо процес декодування нашого текстового повідомлення з зображення. Функція «decode» (лістинг 4.3) зчитує зображення з файлу, витягує прихований текст методом LSB і зберігає цей текст у новий файл. Спочатку зображення зчитується з файлу, зазначеного шляхом imagePath, і конвертується у формат, придатний для маніпуляцій. Наступним кроком зображення перетворюється в масив байтів, щоб можна було здійснити декодування прихованих даних. Викликається функція «decodeImage» (лістинг 4.4), яка зчитує приховані дані з масиву байтів зображення і повертає їх у вигляді масиву байтів. Цей масив конвертується у текстовий рядок. Витягнутий текст

зберігається у новий файл з ім'ям «hidden_textLSB.txt». Виводиться повідомлення про успішне збереження тексту.

Функція decodeImage виконує саме дії щодо витягування прихованого тексту з зображення, використовуючи метод LSB (Least Significant Bit). Вона зчитує приховані біти з байтів зображення, щоб відновити оригінальний текст. Спочатку цикл збирає перші 32 біти, що представляють довжину прихованого тексту. Це виконується за допомогою наступних дій: «length = (length << 1) | (image[i] & 0x1)» виконує зсуває length на один біт вліво і додає найменш значущий біт з поточного байта зображення. Далі маючи довжину прихованого повідомлення можна прочитати саме повідомлення. Це виконається наступним чином: спочатку створюється масив байтів result, який буде містити відновлений текст, потім зовнишній цикл проходить по кожному байту в масиві result, в той час коли внутрішній цикл проходить по кожному біту в поточному байті тексту. result[b] = (byte)((result[b] << 1) | (image[offset] & 0x1)) зсуває поточний байт результату на один біт вліво і додає найменш значущий біт з поточного байта зображення. В кінці обробки функція повертає масив байтів result, що містить витягнутий текст.

Лістинг 4.3 – Функція decode для витягування прихованого тексту з зображення за допомогою методу LSB

```
      public String decode(String imagePath) {
        byte[] decodedHiddenText;
        try {
            BufferedImage imageInUserSpace = getImageInUserSpace(ImageIO.read(new
File(imagePath)));
            byte imageInBytes[] = getBytesFromImage(imageInUserSpace);
            decodedHiddenText = decodeImage(imageInBytes);
            String hiddenText = new String(decodedHiddenText);
            String outputFileName = "hidden_textLSB.txt";
            saveTextToPath(hiddenText, new File(outputFileName));
            System.out.println("Successfully   extracted   text   to:   "   +
outputFileName);
            return hiddenText;
        } catch (Exception exception) {
            System.out.println("No hidden message. Error: " + exception);
            return "";
        }
    }
```

Лістинг 4.4 – Функція decodeImage для відновлення текстових даних із зображення методом LSB

```
      private static byte[] decodeImage(byte[] image) {
              int length = 0;
```

```
            int offset  = bytesForTextLengthData * bitsInByte;

            for (int i = 0; i < offset; i++) {
                length = (length << 1) | (image[i] & 0x1);
            }

            byte[] result = new byte[length];

            for (int b=0; b<result.length; b++ ) {
                for (int i=0; i<bitsInByte; i++, offset++) {
                    result[b] = (byte)((result[b] << 1) | (image[offset] &
0x1));
                }
            }
            return result;
        }
```

4.3Реалізація методу Куттера-Джордана-Боссена

У цьому розділі ми розглянемо реалізацію методу Куттера-Джордона-Боссена (Kutter-Jordan-Bossen, KJB) для приховування інформації в цифрових зображеннях. Цей метод є одним із популярних методів стеганографії, який використовує модифікацію синьої компоненти кольору пікселів для вбудовування текстових даних. Метод KJB забезпечує високу стійкість до спотворень та шумів, що робить його ефективним для приховування інформації у зображеннях, що піддаються обробці та передачі через мережі. Нижче буде детально розглянуто принципи роботи цього методу, процеси кодування та декодування, а також наведено приклади реалізації на мові програмування Java.

Спершу розглянемо допоміжні функції, які забезпечують основні операції з обробки зображень та текстових даних.

getByteArrayLength(byte[] array) – ця функція використовується для отримання масиву байтів, що представляє довжину вхідного масиву байтів. Вона приймає масив байтів як вхідний параметр і повертає масив з 4 байтів (шляхом побітового зсуву та маскування), що містить довжину вхідного масиву.

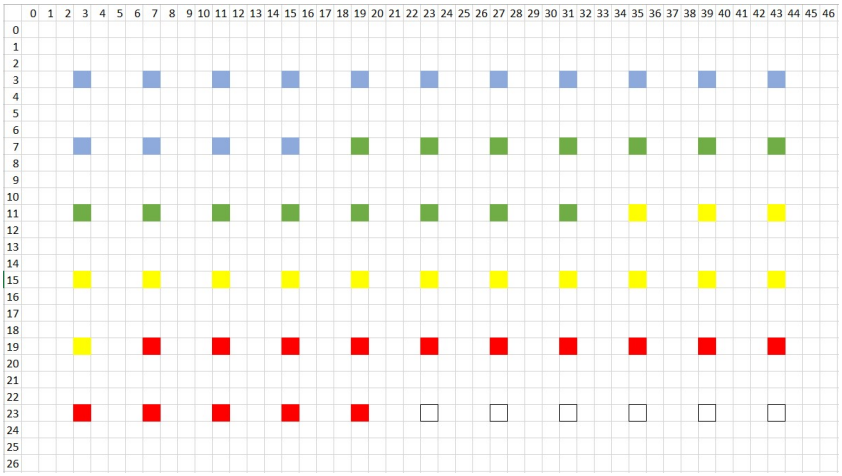
saveTextToPath(String text, File file) – Аналогічно до функції з методу LSB ця функція виконує зберігання тексту у вказаний файл. Вона приймає саме текст котрий слід зберегти та файл. Спочатку виконується перевірка, чи існує файл. У випадку відсутності файлу створюєтсья новий. Після перевірки виконується саме запис текста у наданий файл.

makeImageCopy(BufferedImage imageToCopy) – ця функція створює копію зображення для подальших маніпуляцій. Спочатку створюється новий об'єкт BufferedImage з такими ж розмірами і типом, як у вхідного зображення. Після цього вона використовує об'єкт Graphics для копіювання вхідного зображення у нове зображення та повернає отриманий результат.

Далі перейдемо до більш ключових функцій реалізації даного методу. Функція «encode» (лістинг 4.5) виконує приховування тексту у зображенні за методом Куттера-Джордона-Боссена (Kutter-Jordan-Bossen). Вона приймає текст, який потрібно сховати, та шлях до зображення. Функція повертає нове зображення з прихованим текстом. Спочатку виконується завантаження зображення та підготовка його до маніпуляцій. Також на початковому єтапі відбувається встановлення початкових координати для вбудовування тексту в пікселі (xPos = 3, yPos = 3). Далі за допомогою вище описаної функції «makeImageCopy» створюється копія зображення для запису даних.

Не менш важливим єтапом є перевірка, чи достатньо місця в зображенні для вбудовування тексту. Якщо місця недостатньо, функція показує виняток з відповідним повідомленням. Умову для перевірки можна пояснити наступним чином: «message.length» котре є значенням довжини повідомлення множимо на 8 через те, що повідомлення зберігається у байтах, а приховування даних відбувається саме у бітах, тому враховуючи цей момент слід врахувати, що довжина повідомлення при вбудовуванні буде подана саме у бітах. Після врахування цього виконуєтсья множення на «NUM_OF_REPEATS» котре відповідає за кількість повторів вбудовування. Це обумовленно саме принципом вбудовування бітів цього методу.

Зробуміти принцип допомагає нижче наведений приклад вбудовування (рис. 4.4), де кожен колір відповідає за один біт. У наведеному прикладі для вбудування 1 біта використовується 15 повторів («шансів»), що збільшує стійкість даного алгоритма, але зменшує кількість тексту який можна приховати в зображення. У даній реалізції цей параметр є змінним, тому ми можемо контролювати цей процес враховуючи наші потреби.



Рисноу 4.4 – Схематичний приклад вбудовування бітів методом Куттера-Джордана-Боссена (1 колів – вбудовує 1 біт інформації)

Після перевірки контейнера виконується саме вбудовування тексту в зображення за допомогою фукнції writeByte та аналогічно діям в методі LSB формується ім'я файлу для збереження зображення з прихованим текстом, додаючи суфікс "_with_hidden_messageKJB" та виконується зберігання нового зображення з прихованим текстом у файл з допомогою ImageIO.write.

Лістинг 4.5 – Функція encode метода Куттера-Джордана-Боссена

```
    public BufferedImage encode( String text, String imagePath ) throws Exception {
            BufferedImage bitmap = getImageInUserSpace(ImageIO.read(new File(imagePath)));
            byte[] message = prepareTextToEncode( text );

            xPos = yPos = 3;

            BufferedImage result = makeImageCopy( bitmap );


            if( ( message.length * 8 * NUM_OF_REPEATS ) > ( ( bitmap.getWidth() / 4 - 1 ) * ( bitmap.getHeight() / 4 - 1 ) ) ) {
                throw new Exception( " The image is too small for the given text." );
            }

            for( int i = 0; i < message.length; i++ ) {
                writeByte( result, message[ i ] );
            }

            String fileName = imagePath;
            int position = fileName.lastIndexOf(".");
            if (position > 0) {
                fileName = fileName.substring(0, position);
            }

            String finalFileName = fileName + "_with_hidden_messageKJB.png";
```

```
            System.out.println("Successfully encoded text in: " +
finalFileName);
            try {
                ImageIO.write(result, "png", new File(finalFileName));
            } catch (IOException e) { throw new RuntimeException(e);}

            return result;
        }
```

Якщо детальніше розглянути процес вбудовування, то він починається саме з функції writeByte. В цій функції виконується цикл, що дозволяє пройти через усі 8 бітів байта котний ми передаємо у функцію. Отримання окремого біта відбувається за допомогою використання побітового оператор зсуву вправо (>>>) для зсуву byteVal на j позицій вправо, а потім використовує побітовий оператор AND (&) з 1, щоб отримати значення конкретного біта. Наприклад, якщо j дорівнює 7, зсувається найбільш значущий біт на крайнє праве місце і виконується AND з 1, щоб отримати цей біт. Після отримання значення окремого біта (яке буде або 0, або 1), він передається до функції writeBit, яка відповідає за вбудовування цього біта в зображення.

Розглянемо функцію writeBit: спочатку виконується цикл котрий відповідає за повторне вбудовування біта кілька разів у різні пікселі (рис 4.4) для підвищення стійкості до шуму та інших впливів, що можуть зіпсувати дані. Після виконується переміщення позицій вбудовування, перевіряючи, чи не виходить поточна позиція за межі зображення. Якщо виходить, то позиція xPos повертається до початку нового рядка (xPos = 3), а позиція yPos збільшується на 4 пікселі вниз. (це також наглядно показано на рис 4.4). Далі виконується саме вбудовування біта в піксель за домогою функції writeIntoPixel, та переміщення позиції xPos, збільшуючи її на 4 пікселі вправо. Після кожного вбудовування, переходимо на наступних координат для шифрування. Як приклад припустимо, потрібно вбудувати біт 1. На першій ітерації: xPos = 3, yPos = 3, виконується вбудовування біта 1 у піксель (3, 3), після чого xPos збільшується на 4. На другій ітерації: xPos = 7, yPos = 3, виконується вбудовування біта 1 у піксель (7, 3), після чого xPos знову збільшується на 4. І так далі, доки цикл не завершиться. Якщо xPos перевищує ширину зображення, позиція xPos скидається до 3, а yPos збільшується на 4, переходячи до наступного рядка пікселів.

Функція writeIntoPixel (лістинг 4.6) є ключовою частиною методу Куттера-Джордона-Боссена для стеганографічного вбудовування інформації в зображення. Вона модифікує синю компоненту пікселя залежно від вбудовуваного біта і енергії, щоб приховати інформацію. Спочатку функція отримує поточний піксель з координатами (x, y) і зчитує його червону, зелену та синю компоненти. Використовується формула (0.29890 * red + 0.58662 * green + 0.11448 * blue) для обчислення яскравості пікселя. Це стандартна формула для перетворення кольорового зображення в градації сірого, яка враховує різну вагу червоної, зеленої та синьої компонент. Аналізуючи дану формулу можна зробити висновок, що зелений колір найбільш важливий для подання яскравості, коли синій найменш значущий та менш сприятливий людським оком. Далі виконується саме модифікація синьої компоненти залежно від значення біта, який потрібно вбудувати. Якщо біт дорівнює 1, синя компонента збільшується на величину, пропорційну енергії та яскравості пікселя. Якщо біт дорівнює 0, синя компонента зменшується. Збільшення або зменшення даної компоненти доволі незначні, але достатні для приховування інформації.

Також слід зауважити, що модифікована синя компонента обмежується в діапазоні від 0 до 255, щоб залишатися в межах допустимих значень для кольорових компонент. Але обмеження значень синьої компоненти пікселя в діапазоні від 0 до 255 може призвести до втрати інформації. Це відбувається через те, що модифікація синьої компоненти може виходити за межі допустимого діапазону, і в таких випадках значення компоненти коригується до найближчого допустимого значення (0 або 255). Після модифікації синьої компоненти створюється новий об'єкт Color з новими значеннями кольорових компонент, і цей модифікований піксель записується назад в зображення.

Лістинг 4.6 – Функція writeIntoPixel метода Куттера-Джордана-Боссена

```
    private static int writeIntoPixel( BufferedImage image, int x, int y, int
bit, double energy ) {

            Color pixel = new Color( image.getRGB( x, y ) );
            int red = pixel.getRed();
            int green = pixel.getGreen();
            int blue = pixel.getBlue();
```

```
                int pixelBrightness = ( int ) ( 0.29890 * red + 0.58662 * green +
0.11448 * blue );

                int modifiedBlueComponent;
                if( bit > 0 ) {
                    modifiedBlueComponent   =   (   int   )   (   blue   +   energy   *
pixelBrightness );
                } else {
                    modifiedBlueComponent   =   (   int   )   (   blue   -   energy   *
pixelBrightness );
                }

                if( modifiedBlueComponent > 255 ) {
                    modifiedBlueComponent = 255;
                }

                if( modifiedBlueComponent < 0 ) {
                    modifiedBlueComponent = 0;
                }

                Color pixelModified = new Color( red, green, modifiedBlueComponent
);
                image.setRGB( x, y, pixelModified.getRGB() );

                return modifiedBlueComponent;
            }
```

Далі за логічним змістом розглянемо процес декодування прихованого тексту з зображення за допомогою методу Куттера-Джордона-Боссена. Детальніше розглянемо одну із функцій декодування, а саме функцію decode (лістинг 4.7), яка реалізує цей процес. Спочатку цієї функції виконується зчитування зображення з вказаного шляху і конвертується у зручний для обробки формат за допомогою функції getImageInUserSpace. Після цього ініціалізуються початкові координати xPos та yPos для обробки пікселів (xPos = yPos = 3). Після цього за допомогою функції readByte витягуються перші 4 байти, які містять довжину прихованого тексту та конвертуються у ціле число msgLen, яке визначає довжину прихованого тексту. Наступним проком є перевірка на коректність довжини повідомлення. Якщо довжина неприпустима або не відповідає розміру зображення, виводиться відповідне повідомлення. Далі витягується прихований текст з зображення у байтовий масив msgBytes згідно з отриманою довжиною. Після отримання тексту виконується збереження отриманого тексту у файл hidden_textKJB.txt.

Лістинг 4.7 – Функція decode метода Куттера-Джордана-Боссена

```
    public String decode( String imagePath ) throws Exception {
            BufferedImage   bitmap   =   getImageInUserSpace(ImageIO.read(new
File(imagePath)));
```

```
            xPos = yPos = 3;

            byte lenByte0 = readByte( bitmap );
            byte lenByte1 = readByte( bitmap );
            byte lenByte2 = readByte( bitmap );
            byte lenByte3 = readByte( bitmap );

            int msgLen = ( ( lenByte0 & 0xff ) << 24 ) |
                    ( ( lenByte1 & 0xff ) << 16 ) |
                    ( ( lenByte2 & 0xff ) << 8 ) |
                    ( lenByte3 & 0xff );

            if( ( msgLen <= 0 ) || ( ( msgLen * 8 * NUM_OF_REPEATS ) > (
bitmap.getWidth() / 4 - 1 ) * ( bitmap.getHeight() / 4 - 1 ) ) ) {
                throw new Exception( "Error in the decoding process. Make sure
the image contains text." );
            }

            byte[] msgBytes = new byte[ msgLen ];
            for( int i1 = 0; i1 < msgLen; i1++ ) {
                msgBytes[ i1 ] = readByte( bitmap );
            }

            String msg = new String( msgBytes );
            String outputFileName = "hidden_textKJB.txt";
            saveTextToPath(msg, new File(outputFileName));
            System.out.println("Successfully   extracted   text   to:   "   +
outputFileName);
            return msg;
        }
```

Детальніше розглянемо процес зчитування прихованого повідомлення з нашого зображення. Як вище було згадано функція readByte відповідає за зчитування одного байта з зображення. Вона зчитує 8 бітів, складаючи їх у байт. Для реалізації цього виконуєтсья цикл в якому на кожній ітерації відбувається зсув значення byteVal на один біт вліво, щоб звільнити місце для нового біта з правого боку. Далі виконується процес зчитування одного біта з зображення за допомогою функції readBit(img) та за допомогою операції логічного OR додається значення зчитаного біта до byteVal.

Далі розглянемо функцію readBit котра призначена для зчитування одного біта з зображення, використовуючи ймовірнісну оцінку, засновану на значеннях пікселів. Спочатку створюється змінна bitEstimate, яка буде використовуватися для накопичення ймовірнісної оцінки значення біта. Далі за допомогою циклів аналогічно до процесу шифрування ми проходимо по потрібним нам пікселям та викликаємо функцію readFromPixel, яка буде зчитувати біт з пікселя та додавати його до значення накопичення ймовірності (bitEstimate). Після отримання деякого значення ймовірності ми ділимо його на NUM_OF_REPEATS для

отримання середнього значення за допомогою якого далі буде визначатися який біт ми вбудували (0 чи 1). Якщо на більшості ітерацій середне значення було більше 0.5, то було записано значення 1, в іншому випадку було записано значення 0.

Для більш наглядного прикладу уявимо, що ми намагалися записати біт зі значенням 1. Для цього ми будемо дублювати його вбудовування NUM_OF_REPEATS (наприклад 15) разів, але у деяких випадках, через перешкоди на ітераціях може записатися значення 0. Тоді послідовність може виглядати наступним чином (рис 4.5 показано синім кольором): 111001111111101. З даної посладовності видно, що помилка виникла на 4, 5 та 14 ітерації, але саме за допомогою ймовірнісної оцінки ми можемо сказати, що значення котре ми приховали буде саме 1, адже підсумовуючи значення одиниць, bitEstimate в кінцевій ітерації буде = 13. Після цього виконується знаходження середнього значення ймовірності, яке у наданому прикладі буде дорівнювати 13/15 = 0.8(6). Далі виконається умова, де 0.8(6) > 0.5 і як результат ми дізнаємося, що було вбудовано біт зі значенням 1.

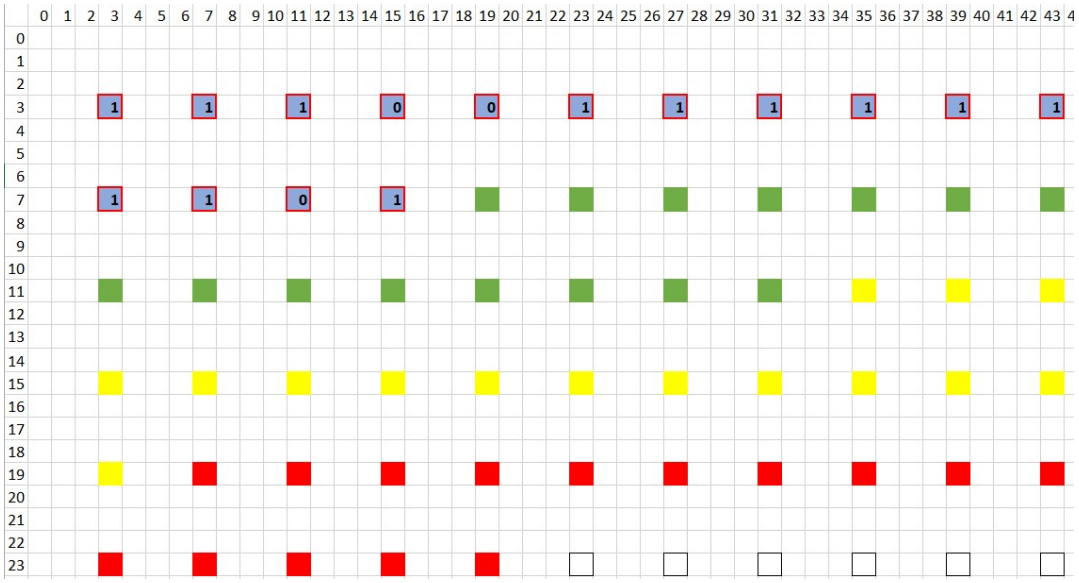


Рисунок 4.5 – Приклад заповнення одного біта методом Куттера-Джордана-Боссена

Найголовнішою функцією для декодування повідомлення є саме функція readFromPixel котра визначає значення біта на основі синьої компоненти пікселя

та його оточуючих пікселів. Спочатку створюємо змінну estimate для накопичення значень синьої компоненти оточуючих пікселів. Далі виконуємо чотири цикли котрі проходять по пікселях, розташованих навколо центрального пікселя (x, y) на відстані від 1 до 3 пікселів у кожному напрямку (вправо, вліво, вгору і вниз) та отримуємо значення синьої компоненти, що додається до значення estimate (рис. 4.6). Цей процес допомагає оцінити контекст і виявити незначні зміни в синій компоненті, які могли бути внесені для приховування інформації. Далі маючи деяке значення (estimate) накопичення синьої компоненти оточуючих пікселів знаходимо її середнє значення. Для цього ділимо це значення на 12 (4 напрямки і у кожній по 3 пікселі. 4*3 = 12). Далі ми зчитуємо значення сильної компоненти з центрального пікселя та порівнюємо з середнім значенням estimate. Якщо значення центрального пісеня (котре є значенням однієї з інтерацій у функції readBit) більше ніж estimate, то центральний біт = 1, у іншому випадку = 0.

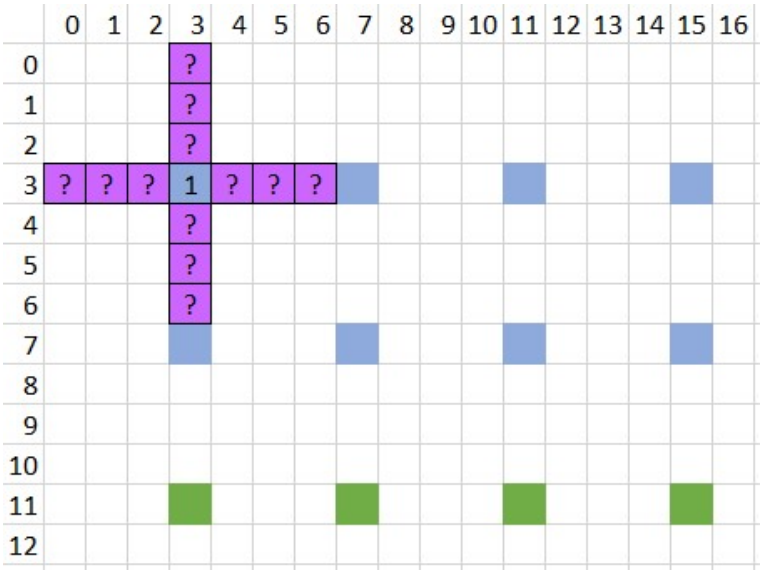


Рисунок 4.6 – Приклад знаходження значення накопичення синьої компоненти оточуючих пікселів методу Куттера-Джордана-Боссена

4.4Перевірка роботи методів приховування інформації

У цьому розділі ми проведемо аналіз роботи двох методів стеганографії: методу найменш значущих біт (Least Significant Bit, LSB) та методу Куттера-Джордана-Боссена. Ми перевіримо їхню здатність приховувати та відновлювати

інформацію без втрат, а також їх стійкість до стиснення зображень. Метод LSB, як попредньо вже було сказано в роботі, є одним з найпростіших і найпоширеніших методів стеганографії, який дозволяє приховувати текстові дані в пікселях зображення шляхом заміни найменш значущих бітів кожного байту. Метод Куттера-Джордана-Боссена використовує більш складні алгоритми для зміни компонент пікселів з метою забезпечення кращої стійкості до змін у зображенні.

Перш за все, ми продемонструємо правильність роботи обох методів на прикладах, де вони приховують і успішно відновлюють інформацію з незміненого зображення. Після цього ми перевіримо роботу методів після стиснення зображення. Очікується, що метод LSB зазнає значних втрат інформації, тоді як метод Куттера-Джордана-Боссена, завдяки своїй стійкості до шуму, покаже кращі результати.

Використане зображення має розмір 1920x1080 пікселів (рис 4.7). Текстове повідомлення для перевірки працездатності має вигляд «Volotkovskiy Dmitriy. This is LSB realisation!» та «Volotkovskiy Dmitriy. This is KJB realisation!» відповідно до кожного методу.

Загальні характеристики вбудованого тексту :

- Довжина тексту: 46 символів
- Кількість байт: 46 байтів
- Кількість біт: 368 бітів

Рисунок 4.7 – Використане зображення у роботі

Розрахунок приблизної кількості можливо вбудованих біт у зображення розміром 1920x1080 пікселів для методу LSB:

1) Зображення розміром 1920x1080 пікселів містить 1920×1080=2,073,600 пікселів.
2) Кожен піксель має три кольорових канали (R, G, B).
3) Метод LSB може вбудовувати до 1 біта інформації в кожен кольоровий канал кожного пікселя. (У нашому випадку вбудовування відбувається тільки у синій канал тому розрахуємо це окремо): 2,073,600 пікселів * 3 канали на піксель *1 біт = 6,220,800 біт, тобто 777,600 байт. Для одного каналу на піксель: 2,073,600 пікселів * 1 канали на піксель *1 біт = 2,073,600 біт, тобто 259,200 байт. Якщо врахувати що перші 32 біти призначені для довжили повідомлення, то фактичного тексту можна вбудувати 259,200 – 4 = 259,196 байт

Розрахунок приблизної кількості можливо вбудованих біт у зображення розміром 1920x1080 пікселів для методу Куттера-Джордана-Боссена (Враховуючи значення енергії = 0.25 та кількість повторів вбудовування = 15):

1) Кількість доступних пікселів для вбудовування даних: (1920 / 4 - 1) * (1080 / 4 - 1) = 479 * 269 = 128 851 пікселів
2) Кожен піксель містить 1 біт даних, повторений 15 разів: 128 851 піксель * 1 біт / 15 повторів = 8 590 біт
3) Перші 32 біти використовуються для зберігання довжини тексту: 8 590 біт - 32 біти для довжини = 8 558 біт, тобто 1 069 байт фактичного тексту.

Аналізуючи принципи алгоритмів та виконані підрахунки можна сказати, що метод LSB дозволяє приховати більшу кількість інформації в зображенні, оскільки він використовує всі пікселі для вбудовування даних. Метод Куттера-Джордана-Боссена в свою чергу має меншу значно меншу ємність порівняно з LSB.

Враховуючи розмір використаних тестових повідомлень (46 символів, що дорівнює 368 бітам) та розрахункову кількість інформації, яку можна приховати

у зображенні розміром 1920x1080 пікселів для кожного з методів, можна зробити наступний висновок, що обидва методи теоретично здатні успішно приховати повідомлення довжиною 368 бітів у зазначеному зображенні.

Далі (рис.4.8 – 4.12) показано результати та виводи при виконанні методу LSB. З наведених результатів можна зробити висновок, що при звичайному вбудовуванні та видобування повідомлення алгоритм працює коректно.

```
public static void main(String[] args) {
    LSBMethod lsb = new LSBMethod();
    lsb.encode( imagePath: "public/image.png",  text: "Volotkovskiy Dmitriy. This is LSB realisation!");
    lsb.decode( imagePath: "public/image_with_hidden_messageLSB.png");
```

Рисунок 4.8 – Ночне зобжалення виклику функцій та передання у них переметрів

```
Successfully encoded text in: public/image_with_hidden_messageLSB.png
Successfully extracted text to: hidden_textLSB.txt

Process finished with exit code 0
```

Рисунок 4.9 – Результати виконання в консолі

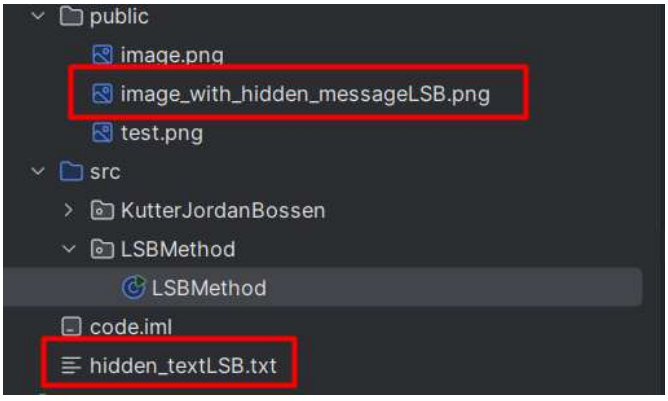


Рисунок 4.10 – Результати створення файлів методу LSB (encode та decode)

Рисунок 4.11 – Результати вилучення зашифрованого повідомлення методом LSB

Рисунок 4.12 – Зображення з прихованим текстом

Спробуємо стиснути зображення з прихованим текстом за допомогою сайту «compresspng.com»[29]. Результати (рис. 4.13) дешефрування в консолі показують помилку котра вказує що немає ніякого прихованого зображення. Це пов’язано з тим, що стиснення зображеня може змінювати або повністю видаляти приховані дані, вбудовані у найменш значущі біти пікселів. Так як пікселі котрі зберігають довжину повідомлення теж змінюються, то найбільш вірогідно, що алгоритм не зміг коректно прочитати ні довжину повідомлення, ні сам зашифрований текст. Порівняти розбір стиснутого та оригінального зображення можна на рис. 4.14.

Рисунок 4.13 – Результати дешефрування стисненого зображення

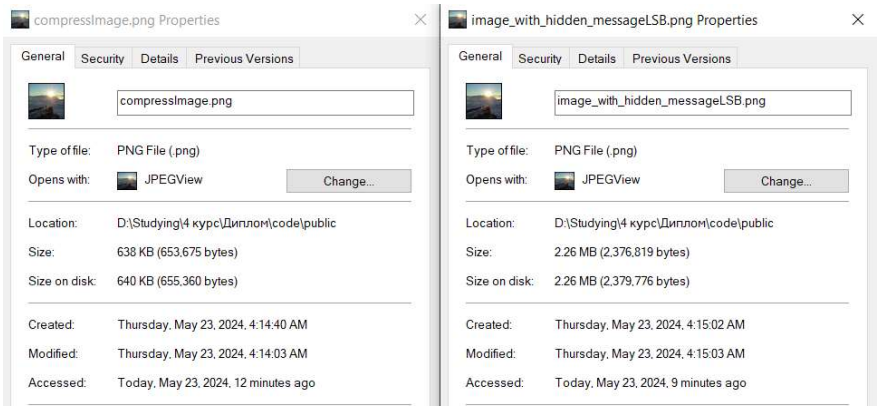


Рисунок 4.14 – Порівняння розмірів зображення (оригіналу та стиснутого)

Наступним короком виконаємо аналогічні дії для методу Куттера-Джордана-Боссена. На вказаних (рис 4.15 – 4.17) малюнках показано які параметри приймають функції encode та decode, показано успішні результати виконання в консолі та створення відповідних файлів, котрі відповідають саме за результати алгоритму.

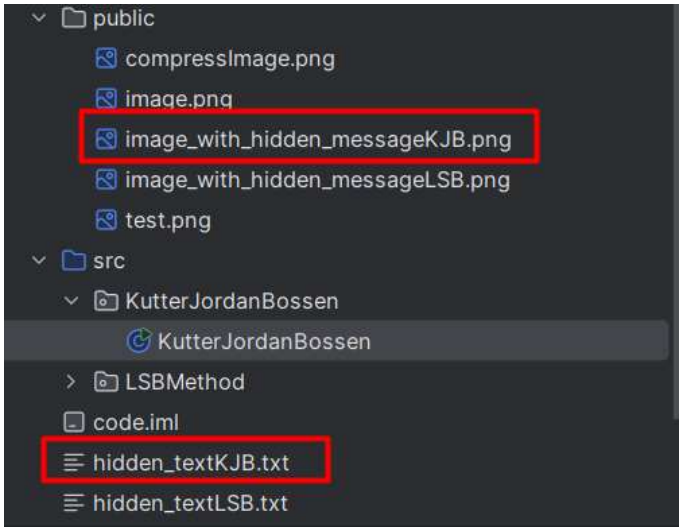


Рисунок 4.15 – Створення результатів виконання алгоритму KJB

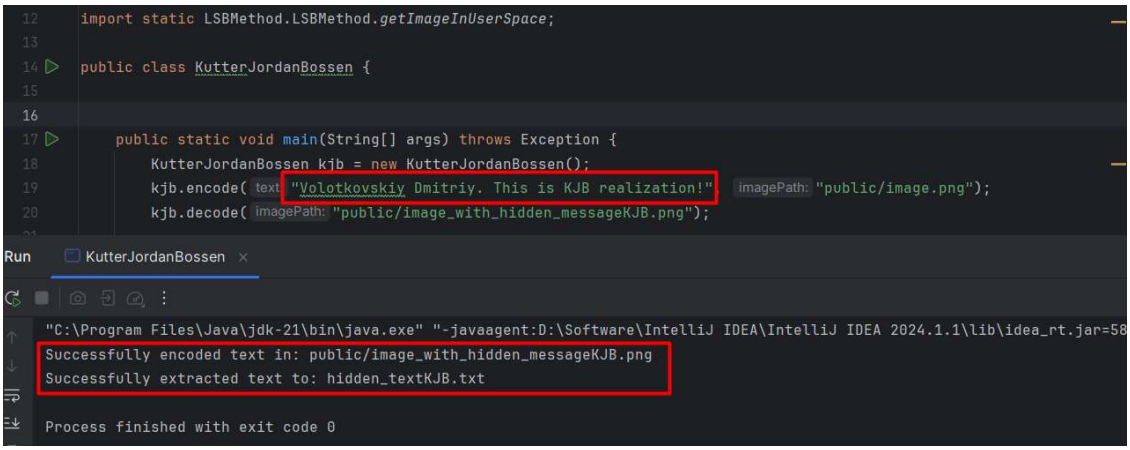


Рисунок 4.16 – Наочне зображення параметрів та виводу консолі роботи методу KJB

Рисунок 4.17 – Результати дешефрування повідомлення методом KJB

Далі аналогічно до попереднього методу виконаємо стиснення та перевіремо декодування зображення. З нижче наведених прикладів (рис. 4.17 – 4.18) видно, що метод Куттера-Джордана-Боссена дістав повідомлення навіть зі стисненого зображення, що показує стійкість даного алгоритму. Слід враховувати, що для цього дослідження були обрані параметри енергії = 0.25 та повторів вбудовування = 15. Можливо, це не самі отпимільні параметри котрі можна використати для коректного дешефрування повідомлення саме для цього стиснення та саме цього зображення, але для наочності та демонстрації працездатності методу були обрані саме ці параметри.

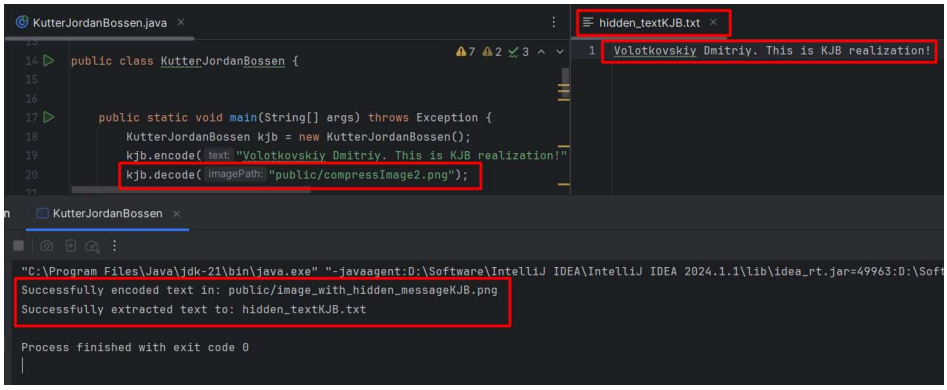


Рисунок 4.17 – Результати алгоритму для стистеного забраження

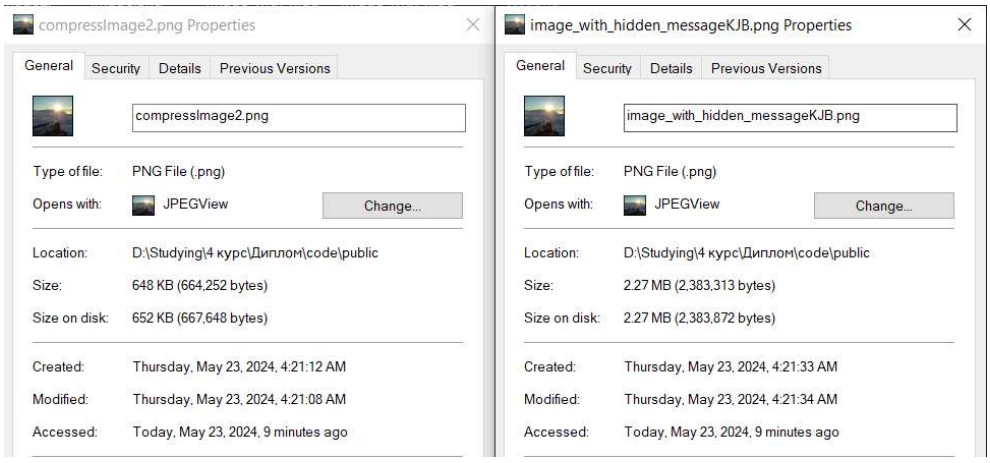


Рисунок 4.18 – Порівняння розмірів зображення (оригіналу та стиснутого)

Аналізуючи метод Куттера-Джордана-Боссена можна також зробити висновок, що цей метод має деякі недоліки які обумовлені деякими особливостями даного методу. Наприклад, цей метод не буде працювати у випадках, коли зображення складається з однорідного синього каналу з

максимальним (255) або близьким до максимального значенням синього кольору. Можливість змінити значення цього каналу буде обмежена. Також метод не буде працювати, якщо у верхній частині, де повинна зберігатися довжина повідомлення, буде максимально синя однорідна частина. Це обумовленно особливіслю реалізації функції вбудовування. Нижче (лістинг 4.8) наведена ця «проблемна» частина коду. У наведеному коді, якщо значення синього каналу після додавання або віднімання енергії виходить за межі допустимого діапазону (0-255), воно обмежується цими значеннями. Це призводить до втрати точності вбудовування даних, оскільки такі зміни не можуть бути коректно закодовані. Як наслідок, інформація може бути втрачена або спотворена.

Лістинг 4.8 – Частина коду функції writeIntoPixel

```
if (modifiedBlueComponent > 255) {
    modifiedBlueComponent = 255;
}

if (modifiedBlueComponent < 0) {
    modifiedBlueComponent = 0;
}
```

Для тестування випадку, коли метод Куттера-Джордана-Боссена не працює з зображенням, яке має максимально синій колір, можна скористатися програмним забезпеченням Windows. Одним з найзручніших інструментів для цього є програма Paint, яка дозволяє швидко створити зображення з необхідними параметрами (рис.4.20). Розмір зображення та текстового повідомлення залишається аналогічним попереднім дослідам цього методу.

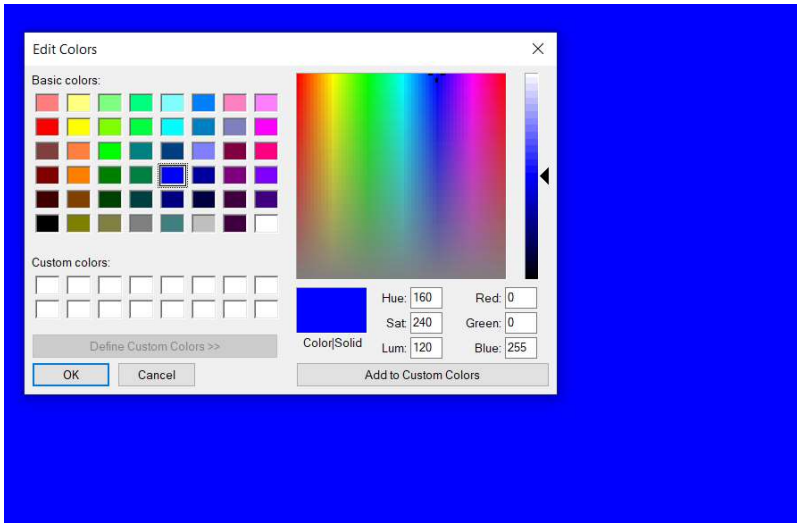


Рисунок 4.20 – Створення зображення для тестування методу KJB

Тестування такого зображення виконувалось з різнипи параметрами енергії та повторів, але результат дешефрування залишався незмінним (рис. 4.21). В усіх випадках дешефрування було перервано помилкою. Також слід зазначили, що при збільшенні параметрів до недопустимого значення (у таких випадках порушується суть стеганографії і вбудовування повідомлення стає помітним та спотворює зображення), а саме 100 повторів та 50 енергії, результати не змінилися (рис 4.22).

Рисунок 4.21 – Дешефрування максимально синього зображення

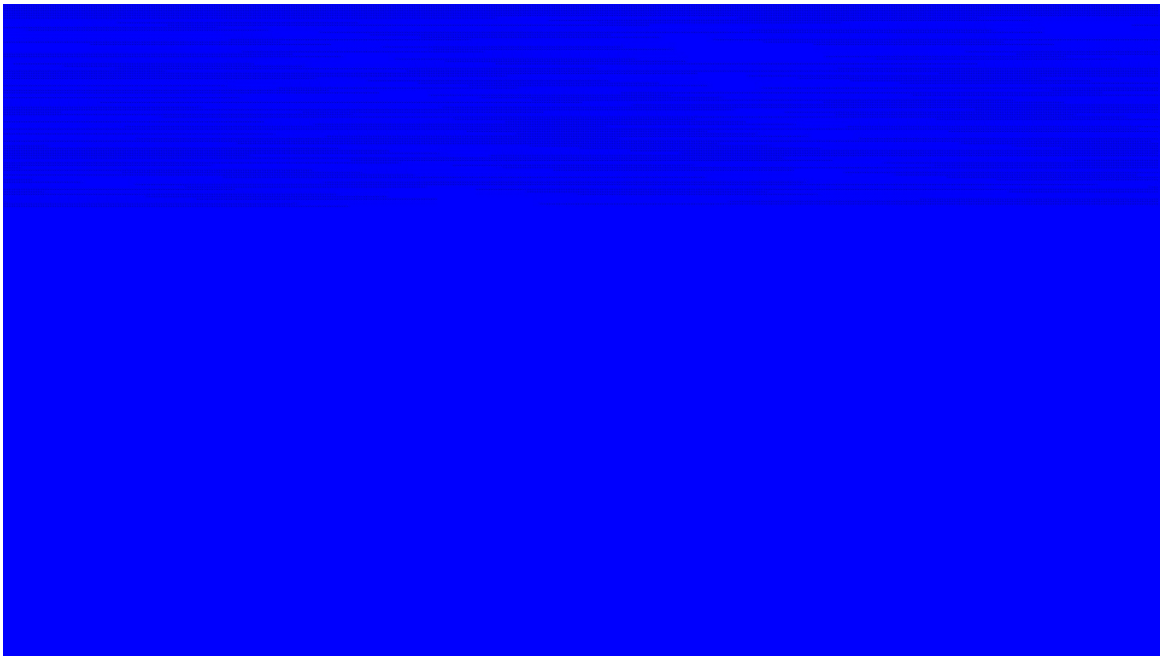


Рисунок 4.22 – Вбудовування повідомлення в зображення зі значенням синього 255

Наступним тестом даного методу є перевірка ефективності та стійкості методу Куттера-Джордана-Боссена при вбудовуванні та зчитуванні прихованої інформації з зображення, яке містить значний рівень шуму. Мета цього тесту - дослідити, як присутність шуму в зображенні-контейнері впливає на здатність методу коректно вбудовувати та витягувати приховану інформацію. Також ми плануємо визначити, які параметри або налаштування методу можуть покращити результати при роботі з такими "зашумленими" зображеннями.

Для виконання цього тесту ми створимо доволі складні умови, використавши в якості зображення-контейнера зображення, яке фактично складається лише з шуму (рис. 4.23). Характеристики цього тестового зображення трохи відрізняються від попередніх експериментів. Воно має розмір 512x512 пікселів, проте такого розміру цілком достатньо для наших цілей. Також зменшимо повідомлення до «This is KJB realization!»

На першому етапі ми спробуємо безпосередньо вбудувати приховане зображення в зашумлене зображення-контейнер. Потім ми вставимо це модифіковане зашумлене зображення як частину іншого зображення та проаналізуємо отриманий результат. Це дозволить оцінити, як наявність сторонніх даних поруч із зашумленим регіоном впливає на процес витягування прихованої інформації.

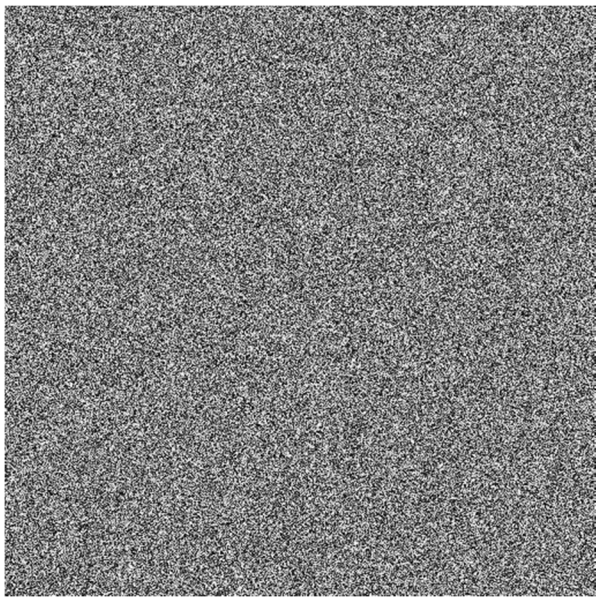


Рисунок 4.23 – Зображення для тесту на приховування повідомлення в шум

Спочатку спробуємо використати стандартні параметри енергії та кількості повторів, які застосовувалися в наших попередніх дослідженнях, а саме 0.25 та 15 відповідно. У такому випадку консоль видала повідомлення про успішне вбудовування прихованих даних. Однак при спробі витягнути ці приховані дані наш алгоритм виявився нездатним розпізнати вбудований текст.

Для покращення результатів ми збільшили кількість повторів до 50, зберігши параметр енергії 0.25, і перевірили результат у цьому випадку. Ця зміна дозволила методу спробувати декодувати вбудоване повідомлення, проте результат отриманого повідомлення значно відрізнявся від початкового вбудованого тексту (рис. 4.24).

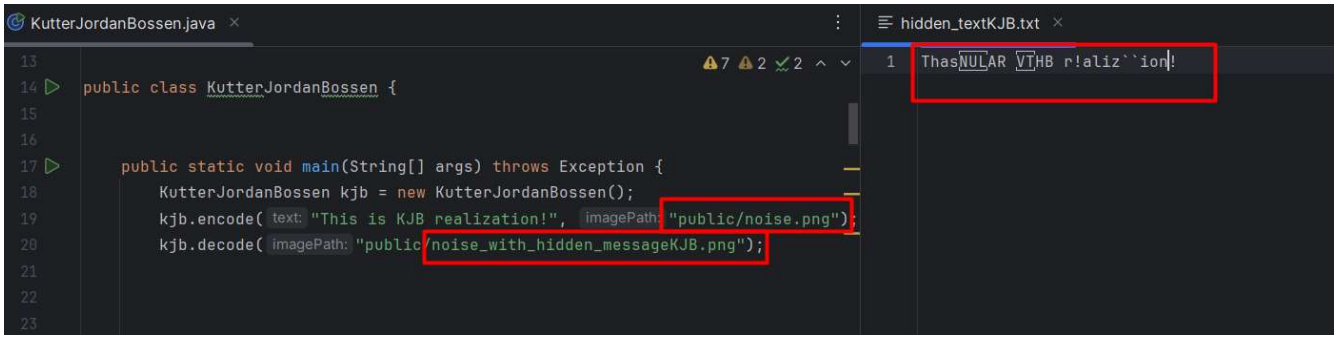


Рисунок 4.24 – Результати видобування тексту з шуму

Після невдалої спроби отримати задовільні результати лише зі збільшенням кількості повторів, також збільшимо значення параметра енергії. Замість стандартного значення 0.25, встановимо енергію на рівні 5, залишивши кількість повторів 50. З такими змінами параметрів нам вдалося успішно декодувати приховане повідомлення з зашумленого зображення-контейнера (рис. 4.25). Витягнутий текст повністю збігався з вихідним повідомленням, вбудованим на початку експерименту. Цей результат демонструє, що збільшення значення енергії в поєднанні з високою кількістю повторів може значно підвищити стійкість методу Куттера-Джордана-Боссена до впливу шуму в зображенні під час вбудовування та витягування прихованих даних.

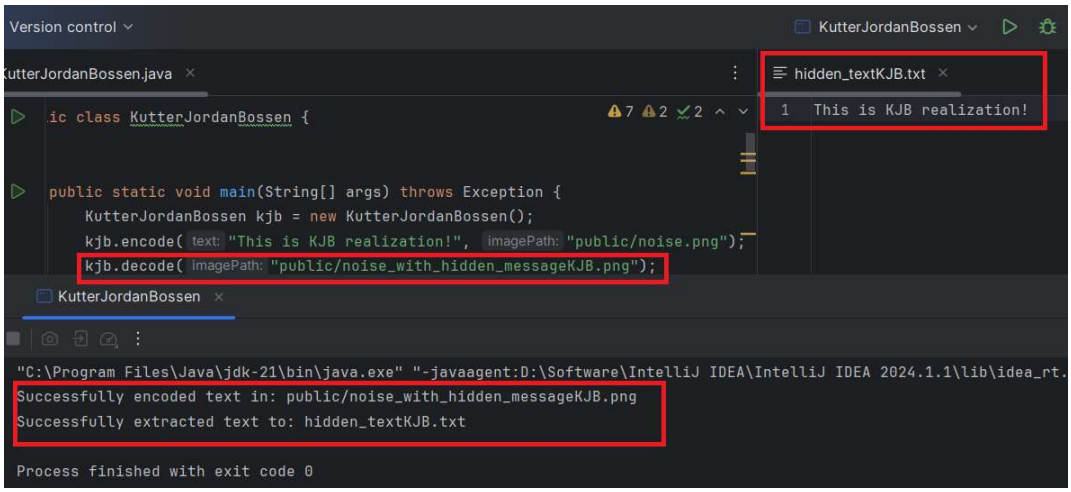


Рисунок 4.25 – Результати видобування тексту з шуму

Також виконаємо ще один додатковий текст, при вбудовуванні частини шуму як частину іншого зображення та проаналізуємо отриманий результат. Використане зображення для тесту має розмір 1920x1080 та зображено на рис 4.26. Також збільшимо текстове повідомлення для наочності даного тесту. Встановимо стандартні параметри енергії та кількості повторів, які застосовувалися в наших попередніх дослідженнях, а саме 0.25 та 15 відповідно.

Рисунок 4.26 – Зображення із вбудованою частоною шумового зображення

Результати даного тесту показують (рис. 4.27), що на початку, коли приховані дані записуються у незашумлену частину зображення-контейнера, процес дешифрування виконується коректно і витягнутий текст збігається з вихідним. Однак, коли вбудовування повідомлення починає відбуватись у

зашумленій частині зображення, процес дешифрування демонструє невірний результат.

Оригінальний текст для вбудовування виглядає наступним чином: «This is KJB realization!This is KJB realization!This is KJB realization!This is KJB realization!This is KJB realization!This is KJB realization!This is KJB realization!This is KJB realization!»

В той час, як результат дешефрування «This is KJB realization!This is KJB realization!This is KJB realization!This is KJB real zathon!This is IJB r�alYzatAon!Phis �s �B real!zaton!Ehispis IJB ealqzat`on!This!is jJB realazaton!».

Як видно, частина витягнутого тексту була спотворена у порівнянні з вихідним повідомленням. Це відбувалось через вплив шуму в області вбудовування прихованих даних. Проте, як було продемонстровано в попередньому експерименті, збільшення значень параметрів енергії та кількості повторів дозволяє успішно дешифрувати вбудоване повідомлення навіть з зашумлених ділянок зображення. Високі значення цих параметрів підвищують стійкість методу до впливу шуму, але як було показано на прикладі вбудовування повідомлення у синє зображення, назадто великі значення значно спотворюєть зображення та можуть не дати потрібного результату.

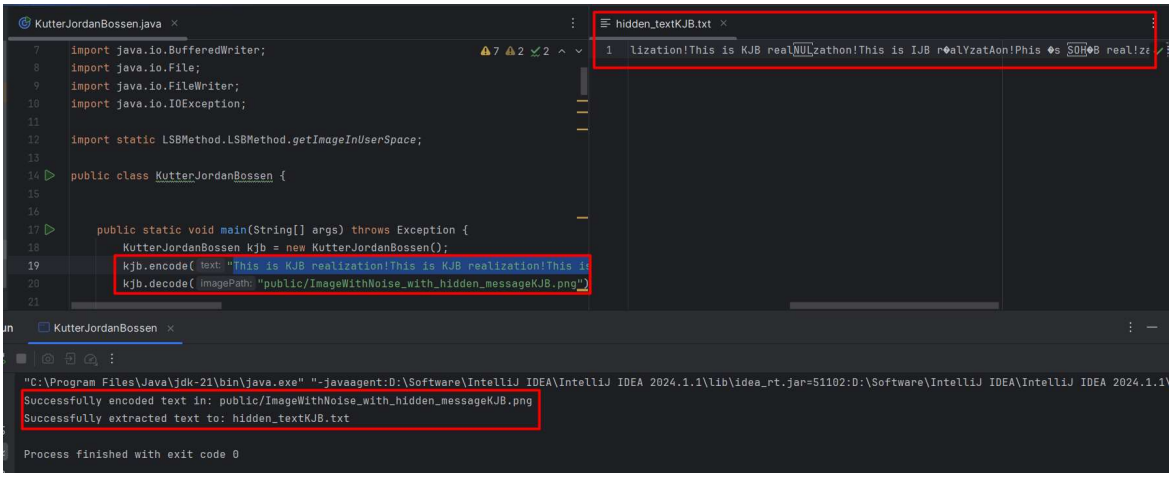


Рисунок 4.27 – Результати дешифрування повідомлення із зображення модифіковане шумовю частино.

ВИСНОВОК

У світі, де цифрові технології відіграють надзвичайно важливу роль, захист конфіденційної інформації стає все більш актуальним питанням. Стеганографія, мистецтво приховування секретних даних у звичайних об'єктах, набуває нового значення в епоху цифрових комунікацій. У рамках дипломної роботи було проведено ґрунтовний аналіз та дослідження методів приховування інформації в цифрових зображеннях, що є одним з найпоширеніших способів стеганографічного приховування даних.

Основна увага була зосереджена на двох провідних методах: методі найменш значущих біт (LSB) та методі Куттера-Джордана-Боссена (KJB). Метою дослідження був ретельний аналіз стійкості цих методів до компресії зображень, виявлення прихованої інформації, а також оцінка їх загальної ефективності та продуктивності.

Метод LSB, один з найпростіших підходів у стеганографії, базується на заміні найменш значущих бітів кожного пікселя зображення-контейнера на біти приховуваного повідомлення. Він вирізняється високою швидкістю та простотою реалізації, проте демонструє низьку стійкість до компресії зображень, що призводить до значних втрат прихованої інформації після стиснення.

Натомість, метод KJB використовує більш складні алгоритми для модифікації компонент пікселів зображення з метою приховування даних. Цей підхід забезпечує значно вищу стійкість прихованої інформації до стиснення та інших спотворень зображень, однак його реалізація є складнішою порівняно з методом LSB.

Результати ретельного тестування обох методів на різноманітних цифрових зображеннях дозволили виявити їхні переваги та недоліки. Зокрема, метод LSB виявився ефективним для приховування великих обсягів даних у зображеннях, проте, при значному стисненні зображень цей метод ставав неефективним, призводячи до суттєвих втрат прихованої інформації.

Натомість, метод KJB продемонстрував високу ефективність та стійкість до компресії при використанні підвищених значень параметрів енергії та кількості повторів. Це дозволяло успішно декодувати приховане повідомлення навіть із зашумлених зображень-контейнерів, на відміну від методу LSB, який зазнавав значних втрат інформації в таких умовах. Водночас, надмірно високі значення цих параметрів могли спричинити спотворення якості зображення.

Загалом, результати ґрунтовного дослідження методів приховування інформації в цифрових зображеннях демонструють, що метод KJB є більш надійним вибором для використання у випадках, де критично важливими є стійкість до стиснення та інших спотворень зображень.

Отримані висновки відкривають перспективи для подальших досліджень, спрямованих на оптимізацію існуючих алгоритмів стеганографії з метою підвищення їх стійкості, ефективності та продуктивності. Розробка нових гібридних методів, що поєднують переваги різних підходів, також є перспективним напрямком. Крім того, важливим є дослідження методів захисту прихованої інформації від стегоаналізу, що дозволить підвищити безпеку та конфіденційність систем стеганографії в цифровому світі.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Конахович Г.Ф., Пузиренко А.Ю. Компьютерная стеганография. Теория и практика. – К.: МК-Пресс, 2006. [Електронний ресурс]. – Режим доступу: https://studfile.net/preview/7379018/
2. Венбо Мао Современная криптография: теория и практика Modern Cryptography: Theory and Practice. – М.: «Вильямс», 2005. – С. 768.89
3. Воробьев В.И., Грибунин В.Г. Теория и практика вейвлет-преобразования. СПб: ВУС, 2009. – 325 с.
4. Нильс Фергюсон, Брюс Шнайер Практическая криптография : Practical Cryptography: Designing and Implementing Secure Cryptographic Systems. – М.: «Диалектика», 2012. – С. 432.
5. Ростовцев А.Г. , Михайлова Н.В. Методи криптоаналізу класичних шифрів. – К.: «Наука», 2012. – С. 142.
6. Саломан А. Криптографія з відкритим ключем. – К.: «Наука», 2013. – 342 с.
7. Fridrich, J., Goljan, M., & Du, R. (2001). Detecting LSB steganography in color and grayscale images. IEEE Multimedia, 8(4), 22-28.
8. Provos, N., Honeyman, P., & Brubaker, C. (2003). Hide and seek: an introduction to steganography. IEEE Security & Privacy, 1(3), 32-44.
9. Li, B., & Wang, M. (2006). An adaptive image steganography technique based on simple matrix encoding. Journal of Information Hiding and Multimedia Signal Processing, 1(2), 167-174.
10. Cox, I. J., Miller, M. L., Bloom, J. A., Fridrich, J., & Kalker, T. (2007). Digital watermarking and steganography. Morgan Kaufmann.
11. Fridrich, J., & Kodovsky, J. (2012). Rich models for steganalysis of digital images. IEEE Transactions on Information Forensics and Security, 7(3), 868-882.
12. Johnson, N. F., & Jajodia, S. (1998). Steganalysis of images created using current steganography software. Proceedings of the 1998 IEEE workshop on information assurance and security, 66-73.

13. О. В. Генне, ТОВ "Конфідент" журнал "Захист інформації. Конфідент", №3, 2000.
14. Столлингс В. Криптография и защита сетей: теория и практика. М: Вильямс. 2001. Пер. с англ. – 235 с.
15. Чмора А.Л. Сучасна прикладна криптографія. 2-е вид., Стер. – М.: Геліос АРВ, 2012. – 256 с.
16. Шнайер, Брюс. Прикладная криптография. Протоколы, алгоритмы, исходные тексты на языке Си – М.: Издательство ТРИУМФ, 2002 – 816 с.
17. Грибунин В.Г.,Оков И.Н.,Туринцев И.В. Цифровая стеганография.М.:Солон Пресс, 2002.
18. Хорошко В.О., Азаров О.Д., Шелест М.Є., Яремчук Ю.Є. Основи комп'ютерної стеганографії : Навч. посіб. для студентів і аспірантів. — Вінниця: ВДТУ, 2003.
19. Генне О.В. Основные положения стеганографии.// Защита информации. №3, 2000.
20. N.F. Johnson, S. Jajodia, Steganalysis: The Investigation of Hidden Information, IEEE Information Technology Conference, Syracuse, New York, USA, Sept. 1st-3rd. 1998.
21. Abbas Cheddad , Joan Condell, Kevin Curran, Paul Mc Kevitt : Digital image steganography: Survey and analysis of current methods, School of Computing and Intelligent Systems, Faculty of Computing and Engineering, University of Ulster at Magee, Londonderry, Northern Ireland, UK.
22. Барсуков В.С. «Компьютерная стеганография вчера, сегодня, завтра. Технологии информационной безопасности 21 века» / В.С. Барсуков, А.П. Романцов ; – М : «Специальная Техника», 2007. [Електронний ресурс]. – Режим доступу: http://www.bnti.ru/showart.asp?aid=330&lvl=04.03.07.
23. What's the Difference Between JPG, PNG, and GIF [Електронний ресурс]. – Режим доступу: https://www.howtogeek.com/30941/whats-the-difference-between-jpg-png-and-gif/

24. ITU-T Recommendation T.81. Information technology – Digital com-pression and coding of continuous-tone still images – Requirements andguidelines; 1992
25. Ю. Яремчук, В. Карпінець УДК: 621.391.7 Використання Цифрових Водяних Знаків Для Захисту Авторського Права В Зображеннях, вип. 2 (13), 2006 р.
26. Кузнецов О. О., Євсеєв С. П., Король О. Г. «СТЕГАНОГРАФІЯ» Навчальний посібник, Харків. Вид. ХНЕУ, 2011
27. TIOBE Index for May 2024 [Електронний ресурс]. – Режим доступу: https://www.tiobe.com/tiobe-index/
28. Java Documentation [Електронний ресурс]. – Режим доступу: https://docs.oracle.com/en/java/
29. Compresspng [Електронний ресурс]. – Режим доступу: https://compresspng.com/
30. S. Moller. A. Pfitzmann, I. Stirand. Computer Based Steganography: How It Works And Why Therefore Any Restriction On Cryptography Are Nonsense. At Best Information Hiding: First International Workshop "InfoHiding'96". Springer as Lecture Notes in Computing Science. 1996. Vol. 1174, pp. 7-21.
31. K. Matsui, K. Tanaka, Fideo-steganography: How To secret Embed A Signature In A Picture // IMA intellectual properly project proceeding, vol. 1, No 1, 1994, pp. 187-205.

ДОДАТОК А

Лістинг А – Реалізація методу LSB

```
package KutterJordanBossen;


import javax.imageio.ImageIO;
import java.awt.*;
import java.awt.image.BufferedImage;
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileWriter;
import java.io.IOException;

import static LSBMethod.LSBMethod.getImageInUserSpace;

public class KutterJordanBossen {

    public static void main(String[] args) throws Exception {
        KutterJordanBossen kjb = new KutterJordanBossen();
        kjb.encode("Volotkovskiy   Dmitriy.   This   is   KJB   realization!",
"public/ImageWithNoise.png");
        kjb.decode("public/image_with_hidden_messageKJB.png");

    }


    private final static int NUM_OF_REPEATS = 15;

    /* The current position of an image point which has information being written
into it */
    private int xPos, yPos;


    public BufferedImage encode( String text, String imagePath ) throws Exception
{
        BufferedImage     bitmap     =     getImageInUserSpace(ImageIO.read(new
File(imagePath)));
        byte[] message = prepareTextToEncode( text );

        xPos = yPos = 3;

        BufferedImage result = makeImageCopy( bitmap );

        /* Checking if the image is big enough for the text */
        if( ( message.length * 8 * NUM_OF_REPEATS ) > ( ( bitmap.getWidth() / 4 -
1 ) * ( bitmap.getHeight() / 4 - 1 ) ) ) {
            throw new Exception( "The image is too small for the given text." );
        }
        /* Encoding */
        for( int i = 0; i < message.length; i++ ) {
            writeByte( result, message[ i ] );
        }

        String fileName = imagePath;
        int position = fileName.lastIndexOf(".");
        if (position > 0) {
            fileName = fileName.substring(0, position);
        }

        String finalFileName = fileName + "_with_hidden_messageKJB.png";
```

```
        System.out.println("Successfully encoded text in: " + finalFileName);
        try {
            ImageIO.write(result, "png", new File(finalFileName));
        } catch (IOException e) { throw new RuntimeException(e);}

        return result;
    }


    public String decode( String imagePath ) throws Exception {
        BufferedImage     bitmap     =     getImageInUserSpace(ImageIO.read(new
File(imagePath)));
        xPos = yPos = 3;

        byte lenByte0 = readByte( bitmap );
        byte lenByte1 = readByte( bitmap );
        byte lenByte2 = readByte( bitmap );
        byte lenByte3 = readByte( bitmap );

        int msgLen = ( ( lenByte0 & 0xff ) << 24 ) |
                ( ( lenByte1 & 0xff ) << 16 ) |
                ( ( lenByte2 & 0xff ) << 8 ) |
                ( lenByte3 & 0xff );

        if( ( msgLen <= 0 ) || ( ( msgLen * 8 * NUM_OF_REPEATS ) > (
bitmap.getWidth() / 4 - 1 ) * ( bitmap.getHeight() / 4 - 1 ) ) ) {
            throw new Exception( "Error in the decoding process. Make sure the
image contains text." );
        }

        byte[] msgBytes = new byte[ msgLen ];
        for( int i1 = 0; i1 < msgLen; i1++ ) {
            msgBytes[ i1 ] = readByte( bitmap );
        }

        String msg = new String( msgBytes );
        String outputFileName = "hidden_textKJB.txt";
        saveTextToPath(msg, new File(outputFileName));
        System.out.println("Successfully extracted text to: " + outputFileName);
        return msg;
    }


    byte[] prepareTextToEncode( String text ) {
        /* Converting text to byte array */
        byte[] msgBytes = text.getBytes();

        /* Getting length of a byte array */
        byte[] lenBytes = getByteArrayLength( msgBytes );

        /* Preparing information to insert */
        byte[] message = new byte[ msgBytes.length + 4 ];
        System.arraycopy( lenBytes, 0, message, 0, lenBytes.length );
        System.arraycopy( msgBytes, 0, message, lenBytes.length, msgBytes.length
);

        return message;
    }

    private void writeByte( BufferedImage img, byte byteVal ) {
        /* Loop through 8 bits of byteVal byte */
        for( int j = 7; j >= 0; j-- ) {
            int bitVal = ( byteVal >>> j ) & 1;
            writeBit( img, bitVal );
```

```
        }
    }


    private byte readByte( BufferedImage img ) {
        byte byteVal = 0;
        /* Getting a byte from 8 bits */
        for( int i = 0; i < 8; i++ ) {
            /* Left shift founded bits and add a bit to the right */
            byteVal = ( byte ) ( ( byteVal << 1 ) | ( readBit( img ) & 1 ) );
        }

        return byteVal;
    }

    private void writeBit( BufferedImage img, int bit ) {

        /* Writing a bit for NUM_OF_REPEATS times */
        for( int i1 = 0; i1 < NUM_OF_REPEATS; i1++ ) {
            if( xPos + 4 > img.getWidth() ) {
                xPos = 3;
                yPos += 4;
            }

            writeIntoPixel( img, xPos, yPos, bit, 0.25);
            xPos += 4;
        }
    }


    private int readBit( BufferedImage img ) {

        /* Probabilistic estimate of an information bit */
        float bitEstimate = 0;
        for( int i1 = 0; i1 < NUM_OF_REPEATS; i1++ ) {

            if( xPos + 4 > img.getWidth() ) {
                xPos = 3;
                yPos += 4;
            }
            bitEstimate += readFromPixel( img, xPos, yPos );
            xPos += 4;
        }
        bitEstimate /= NUM_OF_REPEATS;

        /* if more than half of NUM_OF_REPEATS read bits were 1s, so consider 1
was encoded */
        if( bitEstimate > 0.5 ) {
            return 1;
        } else {
            return 0;
        }
    }

    private static int writeIntoPixel( BufferedImage image, int x, int y, int bit,
double energy ) {

        Color pixel = new Color( image.getRGB( x, y ) );
        int red = pixel.getRed();
        int green = pixel.getGreen();
        int blue = pixel.getBlue();

        int pixelBrightness = ( int ) ( 0.29890 * red + 0.58662 * green + 0.11448
* blue );
```

```
        /* Variable blue component */
        int modifiedBlueComponent;
        if( bit > 0 ) {
            modifiedBlueComponent = ( int ) ( blue + energy * pixelBrightness );
        } else {
            modifiedBlueComponent = ( int ) ( blue - energy * pixelBrightness );
        }

        if( modifiedBlueComponent > 255 ) {
            modifiedBlueComponent = 255;
        }

        if( modifiedBlueComponent < 0 ) {
            modifiedBlueComponent = 0;
        }

        Color pixelModified = new Color( red, green, modifiedBlueComponent );
        image.setRGB( x, y, pixelModified.getRGB() );

        return modifiedBlueComponent;
    }

    private int readFromPixel( BufferedImage image, int x, int y ) {

        /* Summing up all the blue components of surrounding points */
        int estimate = 0;

        for( int i1 = 1; i1 <= 3; i1++ ) {

            Color pixel = new Color( image.getRGB( x + i1, y ) );
            estimate += pixel.getBlue();
        }

        for( int i1 = 1; i1 <= 3; i1++ ) {
            Color pixel = new Color( image.getRGB( x - i1, y ) );
            estimate += pixel.getBlue();
        }

        for( int i1 = 1; i1 <= 3; i1++ ) {
            Color pixel = new Color( image.getRGB( x, y + i1 ) );
            estimate += pixel.getBlue();
        }

        for( int i1 = 1; i1 <= 3; i1++ ) {
            Color pixel = new Color( image.getRGB( x, y - i1 ) );
            estimate += pixel.getBlue();
        }

        /* Average */
        estimate /= 12;

        Color pixel = new Color( image.getRGB( x, y ) );
        int blue = pixel.getBlue();
        if( blue > estimate ) {
            return 1;
        } else {
            return 0;
        }
    }


    public static byte[] getByteArrayLength( byte[] array ) {
```

```
        byte[] lenBytes = new byte[ 4 ];

        lenBytes[ 0 ] = ( byte ) ( ( array.length >>> 24 ) & 0xFF );
        lenBytes[ 1 ] = ( byte ) ( ( array.length >>> 16 ) & 0xFF );
        lenBytes[ 2 ] = ( byte ) ( ( array.length >>> 8 ) & 0xFF );
        lenBytes[ 3 ] = ( byte ) ( array.length & 0xFF );

        return lenBytes;
    }

    private static void saveTextToPath(String text, File file) {
        try {
            if (file.exists() == false) {
                file.createNewFile( );
            }
            FileWriter fileWriter = new FileWriter(file.getAbsoluteFile());
            BufferedWriter bufferedWriter = new BufferedWriter(fileWriter);
            bufferedWriter.write(text);
            bufferedWriter.close();
        } catch (Exception exception) {
            System.out.println("Couldn't write text to file: " + exception);
        }
    }

    BufferedImage makeImageCopy( BufferedImage imageToCopy ) {
        BufferedImage result = new BufferedImage( imageToCopy.getWidth(),
imageToCopy.getHeight(), imageToCopy.getType() );
        Graphics g = result.getGraphics();
        g.drawImage( imageToCopy, 0, 0, null );
        return result;
    }
}
```

ДОБАТОК Б

Лістинг Б – Реалізація методу KJB

```
package LSBMethod;


import javax.imageio.ImageIO;
import java.awt.*;
import java.awt.image.BufferedImage;
import java.awt.image.DataBufferByte;
import java.awt.image.WritableRaster;
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileWriter;
import java.io.IOException;
import java.nio.ByteBuffer;


public class LSBMethod {
    private static int bytesForTextLengthData = 4;
    private static int bitsInByte = 8;

    public static void main(String[] args) {
        LSBMethod lsb = new LSBMethod();
        lsb.encode("public/image.png", "Volotkovskiy Dmitriy. This is LSB
realisation!");
        lsb.decode("public/image_with_hidden_messageLSB.png");
    }

    // Encode
    public void encode(String imagePath, String text) {
        BufferedImage imageInUserSpace = null;
        try {
            imageInUserSpace = getImageInUserSpace(ImageIO.read(new
File(imagePath)));

            byte imageInBytes[] = getBytesFromImage(imageInUserSpace);
            byte textInBytes[] = text.getBytes();
            byte textLengthInBytes[] = getBytesFromInt(textInBytes.length);

            encodeImage(imageInBytes, textLengthInBytes,  0);
            encodeImage(imageInBytes, textInBytes,
bytesForTextLengthData*bitsInByte);
        }
        catch (Exception exception) {
            System.out.println("Couldn't hide text in image. Error: " +
exception);
            return;
        }

        String fileName = imagePath;
        int position = fileName.lastIndexOf(".");
        if (position > 0) {
            fileName = fileName.substring(0, position);
        }

        String finalFileName = fileName + "_with_hidden_messageLSB.png";
        System.out.println("Successfully encoded text in: " + finalFileName);
        try {
            ImageIO.write(imageInUserSpace, "png", new File(finalFileName));
        } catch (IOException e) { throw new RuntimeException(e);}
```

```
    }

    private static byte[] encodeImage(byte[] image, byte[] addition, int offset)
{
        if (addition.length + offset > image.length) {
            throw new IllegalArgumentException("Image file is not long enough to
store provided text");
        }
        for (int i = 0; i < addition.length; i++) {
            int additionByte = addition[i];
            for (int bit = bitsInByte-1; bit >= 0; --bit, offset++) {
                int b = (additionByte >>> bit) & 0x1;
                image[offset] = (byte)((image[offset] & 0xFE) | b);
            }
        }
        return image;
    }

    // Decode
    public String decode(String imagePath) {
        byte[] decodedHiddenText;
        try {
            BufferedImage imageInUserSpace =
getImageInUserSpace(ImageIO.read(new File(imagePath)));
            byte imageInBytes[] = getBytesFromImage(imageInUserSpace);
            decodedHiddenText = decodeImage(imageInBytes);
            String hiddenText = new String(decodedHiddenText);
            String outputFileName = "hidden_textLSB.txt";
            saveTextToPath(hiddenText, new File(outputFileName));
            System.out.println("Successfully extracted text to: " +
outputFileName);
            return hiddenText;
        } catch (Exception exception) {
            System.out.println("No hidden message. Error: " + exception);
            return "";
        }
    }

    private static byte[] decodeImage(byte[] image) {
        int length = 0;
        int offset  = bytesForTextLengthData * bitsInByte;

        for (int i = 0; i < offset; i++) {
            length = (length << 1) | (image[i] & 0x1);
        }

        byte[] result = new byte[length];

        for (int b=0; b<result.length; b++ ) {
            for (int i=0; i<bitsInByte; i++, offset++) {
                result[b] = (byte)((result[b] << 1) | (image[offset] & 0x1));
            }
        }
        return result;
    }

    private static void saveTextToPath(String text, File file) {
        try {
            if (file.exists() == false) {
                file.createNewFile( );
            }
            FileWriter fileWriter = new FileWriter(file.getAbsoluteFile());
            BufferedWriter bufferedWriter = new BufferedWriter(fileWriter);
```

```
            bufferedWriter.write(text);
            bufferedWriter.close();
        } catch (Exception exception) {
            System.out.println("Couldn't write text to file: " + exception);
        }
    }


    // Helpers
    public static BufferedImage getImageInUserSpace(BufferedImage image) {
        BufferedImage imageInUserSpace  = new BufferedImage(image.getWidth(),
image.getHeight(), BufferedImage.TYPE_3BYTE_BGR);
        Graphics2D graphics = imageInUserSpace.createGraphics();
        graphics.drawRenderedImage(image, null);
        graphics.dispose();
        return imageInUserSpace;
    }

    private static byte[] getBytesFromImage(BufferedImage image) {
        WritableRaster raster = image.getRaster();
        DataBufferByte buffer = (DataBufferByte)raster.getDataBuffer();
        return buffer.getData();
    }

    private static byte[] getBytesFromInt(int integer) {
        return
ByteBuffer.allocate(bytesForTextLengthData).putInt(integer).array();
    }
}
```