Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет Інфокомунікацій
(повна назва)
Кафедра Інфокомунікаційної інженерії імені В.В. Поповського
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

Рівень вищої освіти другий (магістерський)

Дослідження методів приховування інформації у стегосистемах
(тема)

Виконав:
студент 2 курсу, групи АМСЗІм-21-2
Ткачов К.В.
(прізвище, ініціали)

Спеціальність: 125 Кібербезпека
(код і повна назва спеціальності)
Тип програми: освітньо-наукова
(освітньо-професійна або освітньо-наукова)
Освітня програма: Адміністративний менеджмент у сфері захисту інформації
(повна назва освітньої програми)
Керівник: доцент кафедри ІКІ ім. В.В. Поповського
Білокуров О.О.
(посада, прізвище, ініціали)

Допускається до захисту
Зав. кафедри _____ Лемешко О.В.
(підпис) (прізвище, ініціали)

2023 р.

Харківський національний університет радіоелектроніки

Факультет Інфокомунікацій
(повна назва)
Кафедра Інфокомунікаційної інженерії імені В.В. Поповського
(повна назва)
Рівень вищої освіти другий (магістерський)
Спеціальність 125 Кібербезпека
(код і повна назва)
Тип програми освітньо-наукова
(освітньо-професійна або освітньо-наукова)
Освітня програма Адміністративний менеджмент у сфері захисту інформації
(повна назва)

ЗАТВЕРДЖУЮ
Зав. кафедри _____
(підпис)
«____» _____ 2023 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

студенту Ткачову Кирилу Владленовичу
(прізвище, ім'я, по батькові)

1. Тема роботи: Дослідження методів приховування інформації у стегосистемах
затверджена наказом по університету від «23» березня 2023р. №292 Ст.
2. Термін подання студентом роботи до екзаменаційної комісії 15.05.2021р.
3. Вихідні дані до роботи: Існуючі стегосистеми, огляд основних стегоситем, їх переваг та недоліків, методи приховування інформації у стегосистемах, розгляд математичних моделей та алгоритмів, що використовуються в стеганографії.
4. Перелік питань, що потрібно опрацювати в роботі:
1) Загальна інформація про стегосистеми.
2) Теоретичний аналіз методів приховування інформації.
3) Дослідження методів приховування інформації та аналіз існуючих стегосистем.
4) Розробки власного методу приховування інформації у стегосистемах.

5. Перелік графічного матеріалу із зазначенням креслень, плакатів, комп'ютерних ілюстрацій: Демонстраційний матеріал у вигляді ppt-презентації.

6. Консультанти розділів роботи

Найменування розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		(підпис)	(дата)
Основна частина	доцент Білокуров Олексій Олександрович		

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1	Отримання завдання	15.02.2023	Виконано
2	Збір матеріалів для дослідження	01.03.2023	Виконано
3	Розробка 1 розділу	20.03.2023	Виконано
4	Розробка 2 розділу	07.04.2023	Виконано
5	Розробка 3 розділу	15.04.2023	Виконано
6	Розробка 4 розділу	22.04.2023	Виконано
7	Оформлення кваліфікаційної роботи	30.04.2023	Виконано

Дата видачі завдання 15 лютого 2023 року

Студент _____ Ткачов К.В.
(підпис) (прізвище, ініціали)

Керівник роботи _____ доцент Білокуров О.О.
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ

Пояснювальна записка: 79 с., 44 рис., 5 додатків, 15 джерел.

ІНФОРМАЦІЙНА БЕЗПЕКА, СТЕГОСИСТЕМИ, ПРИХОВУВАННЯ ІНФОРМАЦІЇ, МОДЕЛЮВАННЯ СТЕГОСИСТЕМИ.

Об’єкт дослідження – процес розробки методу приховування інформації у стегосистемах.

Предмет дослідження – характеристики та властивості методів приховування інформації у стегосистемах, їх стійкість до атак, швидкість та ефективність приховування інформації.

Мета роботи – аналіз існуючих методів приховування інформації у стегосистемах, постановка завдання на розробку власного методу приховування інформації у стегосистемах та їх порівняння з існуючими, визначення ефективності різних методів за різними параметрами, а також дослідження можливості застосування методів приховування інформації у стегосистемах для різних завдань. Основна мета полягає в розробці нових методів приховування інформації у стегосистемах та підвищенні їх ефективності.

Методи досліджень – математичне моделювання, спостереження, аналіз та порівняння.

Проблема захисту інформації від неправомірного доступу та передачі її по незахищених каналах зв'язку є актуальною в сучасному світі. Методи приховування інформації у стегосистемах є одним з ефективних способів захисту конфіденційної інформації. Дослідження цих методів та розробка нових, більш ефективних, має велике значення для забезпечення безпеки інформації в різних сферах, включаючи бізнес, державну та військову сфери, медіа та освіту. Тому актуальність питання полягає в необхідності розробки нових та покращення існуючих методів приховування інформації у стегосистемах.

ABSTRACT

The report contains: 79 p., 44 fig., 5 applications, 15 sources.

INFORMATION SECURITY, STEGOSYSTEMS, INFORMATION HIDING, STEGOSYSTEMS MODELING.

The object of the study is the development process by the method of hiding information in stegosystems.

The subject of the study is the characteristics and properties of information hiding methods in stegosystems, their resistance to attacks, the speed and efficiency of information hiding.

Purpose – to analyze the existing methods of hiding information in stegosystems, to set the task of developing an own method of hiding information in stegosystems and comparing them with existing ones, to determine the effectiveness of various methods according to various parameters, as well as to study the possibility of using methods of hiding information in stegosystems for various tasks. The main goal is to develop new methods of hiding information in stegosystems and increase their efficiency.

Research methods: mathematical modeling, observation, analysis, and comparison.

The problem of protecting information from unauthorized access and its transmission through unprotected communication channels is relevant in the modern world. Methods of hiding information in stegosystems are one of the effective ways to protect confidential information. Researching these methods and developing new, more effective ones is of great importance for ensuring information security in various fields, including business, government and military, media and education. Therefore, the relevance of the issue lies in the need to develop new and improve existing methods of hiding information in stegosystem.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ, УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ І ТЕРМІНІВ

AES – Advanced Encryption Standard
BPM – Binary Phase Manipulation
BPSK – Binary Phase-Shift Keying
DC – Direct Current
DES – Data Encryption Standard
DCT – Discrete Cosine Transformation
DWT – Discrete Wavelet Transform
EXIF – Exchangeable Image File Format
FIS – Fuzzy Inference System
IFS – Iterated Function System
LSB – Least Significant Bit
MLSB – Modified Least Significant Bit
MP3 – MPEG-1 Audio Layer 3
MSB – Most Significant Bit
MSE – Mean Squared Error
NIST – National Institute of Standards and Technology
PSNR – Peak Signal to Noise Ratio
QPM – Quadrature Phase Manipulation
QPSK – Quadrature Phase-Shift Keying
SVM – Support Vector Machines

ВСТУП

У сучасному світі, де інформаційні технології займають все більш значну роль у житті суспільства, проблема захисту конфіденційної інформації є особливо актуальною. Інформація є найважливішим активом, який дозволяє зберігати конкурентну перевагу, забезпечувати безпеку, здійснювати управління тощо. Проте, зростання кількості кіберзлочинів, а також широке використання електронних пристроїв та мережі Інтернет, призводять до того, що конфіденційні дані можуть бути доступні для неправомірних дій третіх осіб.

Один із ефективних способів захисту конфіденційної інформації полягає у використанні методів приховування інформації у стегосистемах. Ці методи дозволяють вбудовувати додаткову інформацію в носії інформації (наприклад, в зображення або відео) таким чином, що зовнішній вигляд та структура носія залишається незмінним. Застосування методів приховування інформації у стегосистемах є широко використовуваним у багатьох галузях, включаючи бізнес, наукові дослідження, архівну документацію, обмін електронними повідомленнями та забезпечення безпеки відомостей в державному секторі.

Тому, метою даної наукової роботи є дослідження методів приховування інформації у стегосистемах та розробка нового методу, який має бути більш ефективним та безпечним порівняно з існуючими методами. Результати дослідження можуть мати велике практичне значення у галузі захисту конфіденційної інформації.

1 ТЕОРЕТИЧНІ АСПЕКТИ СТЕГОСИСТЕМ ТА МЕТОДІВ ПРИХОВУВАННЯ ІНФОРМАЦІЇ

Тема приховування інформації є актуальною в сучасному світі, що пов’язано зі зростання важливості забезпечення конфіденційності, цілісності та доступності інформації у сучасному цифровому світі. Завдяки глобалізації та швидкому розвитку інформаційних технологій, обсяги обміну даними зростають, а значення надійного захисту інформації від несанкціонованого доступу стає все більш важливим.

Стегосистема – це система, яка використовується для приховування одного або декількох повідомлень в іншому повідомленні, таким чином, щоб перше повідомлення було непомітним для більшості користувачів. Стегосистеми можуть бути застосовані в багатьох галузях, включаючи криміналістику, науку про дані, медіа та комунікації.

Для приховування інформації у стегосистемах можуть бути використані різні методи. Основними методами є метод заміни бітів, метод розширення діапазону та метод частотного діапазону. Кожен з цих методів має свої переваги та недоліки, і вибір методу залежить від конкретних обставин.

Стегосистеми складаються з двох основних компонентів: стеганографічного алгоритму та носія.

Стеганографічний алгоритм визначає, як прихована інформація буде вбудована в носій. Він може використовувати різні методи та техніки, залежно від типу носія та вимог до безпеки. Деякі стеганографічні алгоритми прості, такі як заміна менш значущих бітів носія – LSB (Least Significant Bit), в той час як інші можуть бути значно складнішими, використовуючи трансформації області частот, кодування та модуляцію або адаптивні методи.

Носій – це об'єкт, в якому приховується інформація, і який використовується для її передачі. Він може бути цифровим файлом, таким як зображення, аудіо, відео або текстовий документ, або фізичним об'єктом, таким як друковане зображення або аудіозапис. Носій повинен мати достатній рівень випадковості та складності для забезпечення ефективного приховування інформації та захисту від виявлення.

Стегосистеми можуть мати різні цілі та застосування, включаючи забезпечення конфіденційності передачі інформації, захист авторських прав та цифрових водяних знаків, а також військові або розвідувальні операції. Основними вимогами до стегосистем є прихованість, стійкість та пропускна здатність.

Прихованість – це здатність системи уникнути виявлення прихованої інформації. Ефективна стегосистема повинна гарантувати, що прихована інформація не може бути легко виявлена або відрізнена від шуму або інших випадкових характеристик носія.

Стійкість – це здатність системи витримувати спроби виявлення, витягування або зміни прихованої інформації. Ефективна стегосистема повинна забезпечувати захист від різних видів атак, включаючи стегоаналіз, атаки на основі статистичних методів або спроби корупції або зміни носія.

Пропускна здатність – це кількість інформації, яку можна ефективно приховати в носії за одиницю часу. Висока пропускна здатність важлива для деяких застосувань, таких як передача великих обсягів даних або відео.

Крім цього, в стегосистемах можуть бути використані різні методи кодування та шифрування інформації, щоб забезпечити безпеку та надійність приховування даних. Наприклад, метод LSB (Least Significant Bit) використовується для приховування інформації у найменш значущих бітах вказівника на піксель у зображенні. Крім того, можуть бути використані алгоритми шифрування, такі як AES (Advanced Encryption Standard) та DES (Data Encryption Standard), для захисту вбудованої інформації від несанкціонованого доступу.

Насамперед, необхідно провести детальний теоретичний аналіз існуючих методів приховування інформації у стегосистемах, їх переваг та недоліків. Також важливо проаналізувати технології захисту від атак на стегосистеми, зокрема атак на зменшення ємності, зменшення якості зображення та розкриття вбудованої інформації.

Після проведення теоретичного аналізу можна визначити перспективні напрямки досліджень, в тому числі розробку нових методів приховування інформації у стегосистемах, які будуть більш ефективними за певними параметрами, такими як стійкість до атак, швидкодія, якість зображення тощо.

1.1 Основні поняття та класифікація стегосистем

Стегосистеми – це системи, які дозволяють приховувати інформацію в середині інших носіїв, таких як зображення, аудіо, відео та текстові документи, з метою безпечної передачі інформації. Основні поняття стегосистем включають:

– стеганографія – процес приховування інформації в носіях, таких як зображення, аудіо та відео, без зміни їх зовнішнього вигляду чи структури;

– стегоаналіз – процес виявлення наявності прихованої інформації в носіях за допомогою статистичного аналізу, машинного навчання або інших методів;

– ключ стеганографії – секретний параметр, який використовується для приховування та вилучення інформації в стегосистемі.

Основними поняттями у стеганографії є повідомлення та контейнер. Повідомлення – певна закрита інформація, яку необхідно приховати. Контейнер – множина відкритих даних, яка використовується для вбудовування закритої інформації. Контейнери бувають двох типів: потокові та фіксовані, що зображено на рисунку 1.1 [1]. Потокові контейнери – це послідовність бітів, яка постійно змінюється. Повідомлення вбудовуються у нього в реальному режимі часу, тому заздалегідь невідомо, чи вистачить розміру даного контейнера для передачі повідомлення повністю. Фіксовані ж контейнери мають фіксований розмір, тому є можливість оптимальний контейнер для передачі повідомлення. Розмір контейнера повинен, принаймні, у декілька разів перевищувати розмір повідомлення, і, чим більше дане співвідношення, тим надійніше приховане повідомлення.

Для підвищення рівня захищеності секретної інформації повідомлення можна попередньо зашифрувати стійким криптографічним алгоритмом.

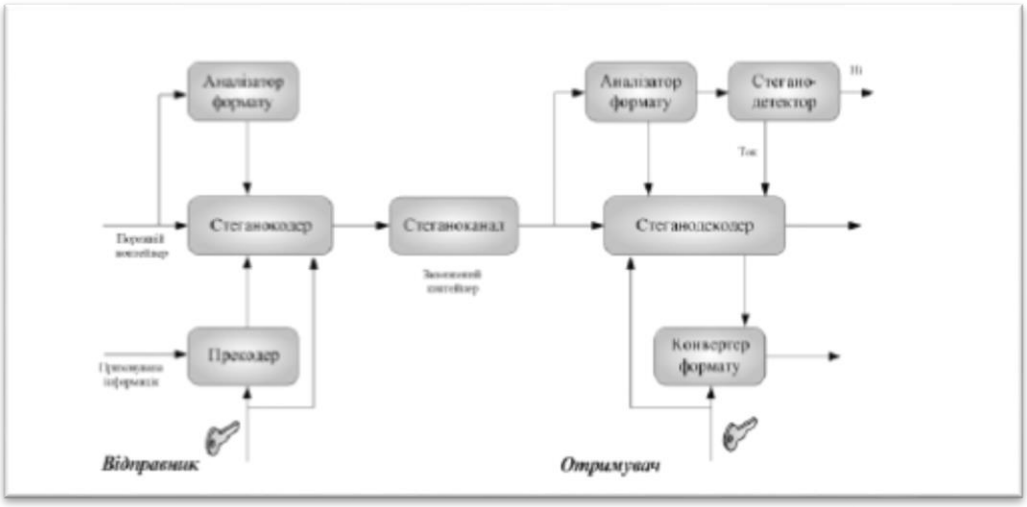

Рисунок 1.1 – Структурна схема типової стегосистеми

Класифікація стегосистем може бути здійснена за рядом критеріїв.

1) За типом носія:

– зображення (цифрові фотографії, графіка);

– аудіо (музика, голосові повідомлення);

– відео (фільми, анімація);

– текстові документи (формати тексту, файли .pdf).

2) За методом приховування:

– впровадження у менш значущі біти LSB;

– трансформація області частот DCT, DWT (Discrete Cosine Transformation, Discrete Wavelet Transform);

– алгоритми стиснення jpeg, mp3;

– використання кодування та модуляції BPSK (Binary Phase-Shift Keying), QPSK (Quadrature Phase-Shift Keying) ;

– адаптивні методи (приховування в областях з високим рівнем шуму або текстур).

3) За наявності ключа:

– симетричні стегосистеми (однакові ключі для приховування та вилучення інформації);

– асиметричні стегосистеми (різні ключі для приховування та вилучення інформації, подібно до схем асиметричного шифрування).

4) За рівнем обробки інформації:

– стегосистеми з обробкою на рівні біта (зміна окремих бітів в носії);

– стегосистеми з обробкою на рівні блоку (зміна цілого блоку даних в носії);

– стегосистеми з обробкою на рівні каналу (експлуатація каналів кольору, амплітуди чи фази).

Розглядаючи різноманітність стегосистем та їх класифікацій, можна краще зрозуміти основні принципи стеганографії, визначити потенційні слабкі місця та розробити нові та ефективні методи приховування інформації. Важливо звернути увагу на те, що кожна стегосистема має свої переваги та недоліки, а також може мати різні рівні ефективності та безпеки в залежності від контексту та застосування.

1.2 Методи приховування інформації

Розглянемо більш детально методи приховування інформації, які використовуються в стегосистемах.

Метод впровадження у менш значущі біти LSB є одним з найпростіших і найпопулярніших методів стеганографії. Він полягає в заміні менш значущих бітів носія (наприклад, зображення) на біти прихованої інформації. Цей метод простий та швидкий, але може бути вразливим до стегоаналізу, особливо якщо використовується на простих та однорідних носіях.

Трансформація області частот використовує перетворення носія з просторової області в частотну область, використовуючи, наприклад, дискретне косинусне перетворення DCT (Discrete Cosine Transformation) або дискретне перетворення вейвлета DWT (Discrete Wavelet Transform). Прихована інформація вбудовується в коефіцієнти частотної області, зазвичай з меншою енергією, для забезпечення мінімального впливу на якість носія. Цей метод може бути більш стійким до стеганалізу, але може вимагати більше обчислювальних ресурсів.

Алгоритми стиснення використовують властивості стиснення даних для приховування інформації. Наприклад, можна використовувати формат jpeg для зображень або формат MP3 (MPEG-1 Audio Layer 3) для аудіо файлів. Прихована інформація вбудовується під час процесу стиснення, використовуючи властивості

квантування або кодування Хаффмана, для забезпечення високої ступені прихованості.

Кодування та модуляція включають використання спеціальних сигналів або модуляції для кодування прихованої інформації. Наприклад, можна використовувати бінарну фазову маніпуляцію BPSK або квадратурну фазову маніпуляцію QPSK для цього. Кодовані сигнали потім вбудовуються в носій, зазвичай на рівні каналу (наприклад, канали кольору для зображень). Ці методи можуть забезпечити більш високу ступінь прихованості та захисту від стегоаналізу порівняно з LSB стеганографією.

Адаптивні стегосистеми враховують властивості носія, такі як рівень шуму, текстуру або контраст, для визначення оптимальних місць для вбудовування прихованої інформації. Ці методи зазвичай використовують машинне навчання або евристичні алгоритми для аналізу носія та адаптації стратегії приховування. Адаптивні стегосистеми можуть забезпечити високу прихованість та захист від стеганалізу, але можуть бути складнішими в реалізації та потребувати більше обчислювальних ресурсів.

Для створення більш ефективних та надійних стегосистем можна комбінувати декілька методів приховування інформації. Вибір конкретного методу залежить від вимог до безпеки, якості носія та обмежень на обчислювальні ресурси. Оптимальний метод також може залежати від специфіки застосування, таких як тип носія (зображення, аудіо, відео або текст), рівень стеганалітичного захисту та потреби у пропускній здатності. У процесі дослідження стегосистем важливо розглядати різні методи та їхні характеристики, щоб знайти найкращий підхід для конкретного застосування.

При використанні комбінацій методів приховування інформації можна досягти більшої стійкості до стеганалізу та більшої прихованості. Наприклад, можна використовувати адаптивний підхід з LSB стеганографією та трансформацією області частот для вбудовування інформації в різні частини носія, в залежності від його властивостей. Це може зробити виявлення прихованої інформації складнішим для атакуючого [3].

Крім того, можна використовувати шифрування для збільшення захисту прихованої інформації. Шифрування може забезпечити додатковий рівень захисту,

так як атакуючий, який виявить приховану інформацію, не зможе прочитати її без відповідного ключа. Це може бути особливо корисним для захисту конфіденційної або чутливої інформації.

У подальшому дослідженні стегосистем можна також розглянути розвиток нових методів приховування інформації, які використовують передові технології, такі як глибоке навчання, штучний інтелект та квантові комп'ютери. Це може відкрити нові можливості для створення більш стійких та ефективних стегосистем, які можуть відповідати вимогам сучасного світу в області безпеки інформації та захисту приватності.

1.3 Виявлення стеганографії та проблеми безпеки

Стегоаналіз – це процес виявлення наявності прихованої інформації в стегосистемах. Це важливий аспект безпеки, оскільки стеганографія може використовуватися для зловмисних цілей, таких як передача конфіденційної інформації, поширення вірусів або шпигунські операції. Стеганаліз може включати в себе візуальні, статистичні та машинне навчання для виявлення аномалій або змін, характерних для стеганографічних методів.

Візуальний аналіз полягає у перегляді носія з метою виявлення видимих ознак приховування інформації. Цей метод може бути ефективним для виявлення простих стеганографічних методів, але стає менш надійним для виявлення розроблених стегосистем, які використовують складні алгоритми та техніки.

Статистичний аналіз використовує математичні та статистичні методи для виявлення аномалій або змін у носії, які можуть вказувати на наявність прихованої інформації. Це може включати аналіз гістограм, кореляційний аналіз та аналіз спектральних характеристик. Статистичний аналіз може бути ефективним для виявлення більш розроблених стеганографічних методів, але може вимагати великої кількості обчислень та аналітичних ресурсів.

Машинне навчання дозволяє створювати моделі, які можуть виявляти стеганографію на основі навчання з використанням наборів даних із зразками стегосистем та чистих носіїв. Це може включати в себе методи класифікації, такі як опорні векторні машини SVM (Support Vector Machines), нейронні мережі, випадкові

ліси та ансамблі. Моделі машинного навчання можуть бути здатні виявляти складні стегосистеми, але їх ефективність залежить від якості навчальних даних та алгоритмів.

Стеганографія, хоча і може мати законні застосування, створює деякі проблеми безпеки через можливість зловмисного використання. Основні проблеми безпеки, пов'язані зі стеганографією, включають наступне.

1) Несанкціонована передача конфіденційної інформації: стеганографія може бути використана для передачі конфіденційної інформації без виявлення, що може призвести до витоку даних або порушення конфіденційності. Наприклад, корпоративні шпигуни можуть використовувати стеганографію для передачі таємниць компанії конкурентам або зловмисникам.

2) Розповсюдження забороненого або шкідливого вмісту: стеганографія може бути використана для приховування та розповсюдження незаконного вмісту, такого як дитяча порнографія, заборонені матеріали чи екстремістські ідеї. Це ускладнює роботу правоохоронних органів та органів цензури.

3) Шпигунство та кібершпигунство: стеганографія може бути використана у розвідувальних операціях або кібершпигунстві для передачі інформації або команд між зловмисниками, що ускладнює виявлення та протидію цим діям.

4) Розповсюдження вірусів та шкідливого програмного забезпечення: зловмисники можуть використовувати стеганографію для розповсюдження вірусів, троянських коней або іншого шкідливого програмного забезпечення, приховуючи його в медіафайлах або документах, які, здається, безпечні.

5) Ускладнення моніторингу комунікацій: стеганографія може використовуватися для маскування комунікацій між зловмисниками або терористами, що ускладнює моніторинг комунікацій правоохоронними органами та розвідувальними службами. Це створює додаткові виклики для національної безпеки та контртерористичних операцій.

6) Приховування атак на інформаційну безпеку: стеганографія може використовуватися для приховування атак на інформаційну безпеку, таких як атаки на середовище хмарних обчислень, незаконний доступ до баз даних або розповсюдження шкідливого коду в програмах. Виявлення таких атак стає складнішим, коли стеганографія використовується для приховування дій.

Враховуючи ці проблеми безпеки, важливо розробляти та вдосконалювати методи виявлення стеганографії, щоб забезпечити ефективний захист від її зловмисного використання. Зокрема, слід надавати пріоритет розвитку нових алгоритмів машинного навчання, що здатні виявляти приховані повідомлення в різних типах носіїв, а також розробці методів аналізу мережевого трафіку, які дозволяють ідентифікувати стеганографічні комунікації.

2 АНАЛІЗ ТА ПОРІВНЯННЯ ІСНУЮЧИХ СТЕГОСИСТЕМ

У цьому розділі буде проаналізовано та порівняно ряд існуючих стегосистем з метою визначення їх переваг та недоліків. Цей аналіз допоможе зрозуміти ключові аспекти різних підходів до стеганографії та сприятиме розробці нових, ефективніших методів приховування інформації.

2.1 Метод впровадження інформації у менш значущі біти

Метод впровадження у менш значущі біти LSB є одним з найпростіших і найпопулярніших методів стеганографії. Він полягає в заміні менш значущих бітів носія (наприклад, зображення) на біти прихованої інформації. Цей метод простий та швидкий, але може бути вразливим до стеганалізу, особливо якщо використовується на простих та однорідних носіях.

Молодший біт часто використовується для вставки даних у цифровий носій, де знаходяться біти даних повідомлення які вставлено, буде замінено на молодший біт [2]. Позиція молодшого біта може бути в останньому біті видно на рисунку 2.1.

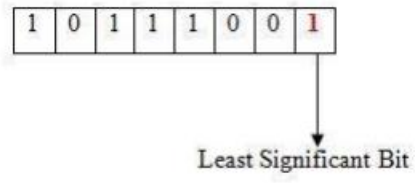

Рисунок 2.1 – Метод впровадження у менш значущі біти

Алгоритм стеганографічного кодування на основі LSB [4] був розроблений наступним чином.

1) Прочитаємо зображення та текстове повідомлення, яке має бути приховано у зображенні.
2) Перетворюємо кольорове зображення на сіре.
3) Перетворюємо текстове повідомлення у бінарний формат.

4) Обчислюємо LSB кожного пікселя зображення.

5) Заміняємо LSB зображення кожним бітом секретного повідомлення один за іншим.

6) Записуємо стегозображення.

Алгоритм декодування був розроблений наступним чином.

1) Зчитуємо стегозображення.

2) Обчислюємо LSB кожного пікселя стегозображення.

3) Отримуємо біти та перетворити кожні 8 бітів на символ.

Переваги:

– простота реалізації та швидкість операцій;

– можливість використання в різних носіях: зображення, аудіо, відео.

Недоліки:

– відносно низька стійкість до статистичного аналізу;

– втрата прихованої інформації при стисненні або обробці носія.

Також існує модифікований метод найменшого значущого біта – MLSB (Modified Least Significant Bit), розробка методу найменшого значущого біта. Цей метод використовується для створення методу, який є кращим за існуючий метод LSB. Багато досліджень використовують саме генератор псевдовипадкових чисел MLSB з алгоритмом множення з перенесенням, оскільки алгоритм множення з перенесенням має кращу продуктивність, і випадкові числа також створюються з більш високою швидкістю.

2.2 Стеганографія на основі трансформації області частот

Трансформація області частот використовує перетворення носія з просторової області в частотну область, використовуючи, наприклад, дискретне косинусне перетворення DCT або дискретне перетворення вейвлета DWT. Прихована інформація вбудовується в коефіцієнти частотної області, зазвичай з меншою енергією, для забезпечення мінімального впливу на якість носія. Цей метод може бути більш стійким до стегоаналізу, але може вимагати більше обчислювальних ресурсів.

Дискретне косинусне перетворення DCT [4] розділяє зображення на частини (або спектральні піддіапазони) різної важливості (стосовно візуальної якості

зображення). DCT перетворює сигнал або зображення з просторової області в частотний домен.

Загальне рівняння для двовимірного (N на M зображення) DCT таке визначається наступною формулою [4]:

$$C(u,v)=\alpha(u)\cdot\alpha(v)\cdot\sum_{x=0}^{N-1}\cdot\sum_{y=0}^{N-1}f(x,y)\cdot\cos\frac{\pi\cdot(2\cdot x+1)\cdot u}{2\cdot N}\cdot\cos\frac{\pi\cdot(2\cdot y+1)\cdot v}{2\cdot N}, \quad (2.1)$$

де $\alpha(u)=\sqrt{\frac{1}{N}}$ для u = 0;

$\alpha(u)=\sqrt{\frac{2}{N}}$ для u = 1,2,3,….,N-1;

C (u, v) – коефіцієнт DCT у рядку u та стовпці v матриці DCT.

Алгоритм кодування стеганографії на основі DCT був розроблений таким чином.

1) Зчитуємо зображення.

2) Пишемо секретне повідомлення та перетворюємо його у бінарний формат.

3) Зображення розбивається на блоки 8 x 8 пікселів.

4) Працюючи зліва направо, зверху вниз віднімiть 128 у кожному блоці пікселів.

5) DCT застосовується до кожного блоку.

6) Кожен блок стискається через таблицю квантування.

7) Обчислюємо LSB кожного коефіцієнта DC (Direct Current) та замінюємо кожним бітом секретного повідомлення.

8) Записуємо стегозображення.

Алгоритм декодування був розроблений таким чином.

1) Зчитуємо стегозображення.

2) Стегозображення розбивається на блоки 8 x 8 пікселів.

3) Працюючи зліва направо, зверху вниз віднімiть 128 у кожному блоці пікселів.

4) DCT застосовується до кожного блоку.

5) Кожен блок стискається через таблицю квантування.

6) Обчислюємо LSB кожного коефіцієнта DC.

Вейвлет-перетворення DWT описує багато роздільний процес декомпозиції з точки зору розширення зображення на набір вейвлет-базисних функцій. DWT має свою відмінну просторову частоту властивість локалізації. Застосування DWT у 2D зображеннях відповідає двовимірній обробці зображення фільтра в кожному вимірі. Вхідне зображення розділене на 4 неперекриваючих піддіапазони з різною роздільною здатністю за допомогою фільтрів, а саме LL1 (коефіцієнти апроксимації), LH1 (вертикальні деталі), HL1 (горизонтальні деталі) і HH1 (діагональні деталі). Піддіапазон (LL1) обробляється далі, щоб отримати наступний більш грубий масштаб вейвлет-коефіцієнту, поки не буде досягнуто кінцевого масштабу «N». Коли буде досягнуто «N», ми матимемо 3N+1 піддіапазони, що складаються піддіапазони із різною роздільною здатністю (LLN) і (LHX), (HLX) і (HHX), де «X» становить від 1 до «N». Загалом більша частина енергії зображення зберігається в цих піддіапазонах, детальніше можна побачити на рисунку 2.2.

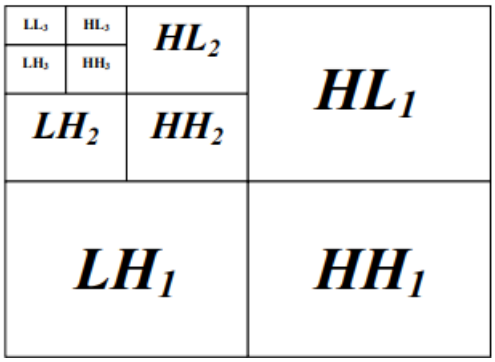

Рисунок 2.2 – Трифазне розкладання за допомогою трансформації вейвлета

Алгоритм кодування стеганографії на основі DWT був розроблений наступним чином [6].

1) Зчитуємо зображення обкладинки та текстове повідомлення, яке має бути приховано на зображенні обкладинки.

2) Перетворюємо текстове повідомлення в двійковий. Застосуйте 2D трансформацію Хаара до зображення обкладинки.

3) Отримаємо горизонтальний і вертикальний коефіцієнти фільтрації зображення обкладинки. Зображення обкладинки додається з бітами даних для коефіцієнтів DWT. Отримаємо стегозображення.

4) Обчислюємо середню квадратичну помилку MSE (Mean Squared Error), пікове співвідношення сигнал/шум – PSNR (Peak Signal to Noise Ratio) стегозображення.

Переваги:

– висока стійкість до статистичного аналізу;

– збереження прихованої інформації при стисненні або обробці носія.

Недоліки:

– складність реалізації та повільність операцій;

– обмежена кількість носіїв, у яких можна застосовувати метод.

2.3 Адаптивна стеганографія

Адаптивні стегосистеми враховують властивості носія, такі як рівень шуму, текстуру або контраст, для визначення оптимальних місць для вбудовування прихованої інформації. Ці методи зазвичай використовують машинне навчання або евристичні алгоритми для аналізу носія та адаптації стратегії приховування. Адаптивні стегосистеми можуть забезпечити високу прихованість та захист від стегоаналізу, але можуть бути складнішими в реалізації та потребувати більше обчислювальних ресурсів.

Приклад блок-схеми за адаптивної стеганографії виглядає наступним чином на рисунку 2.3 [7].

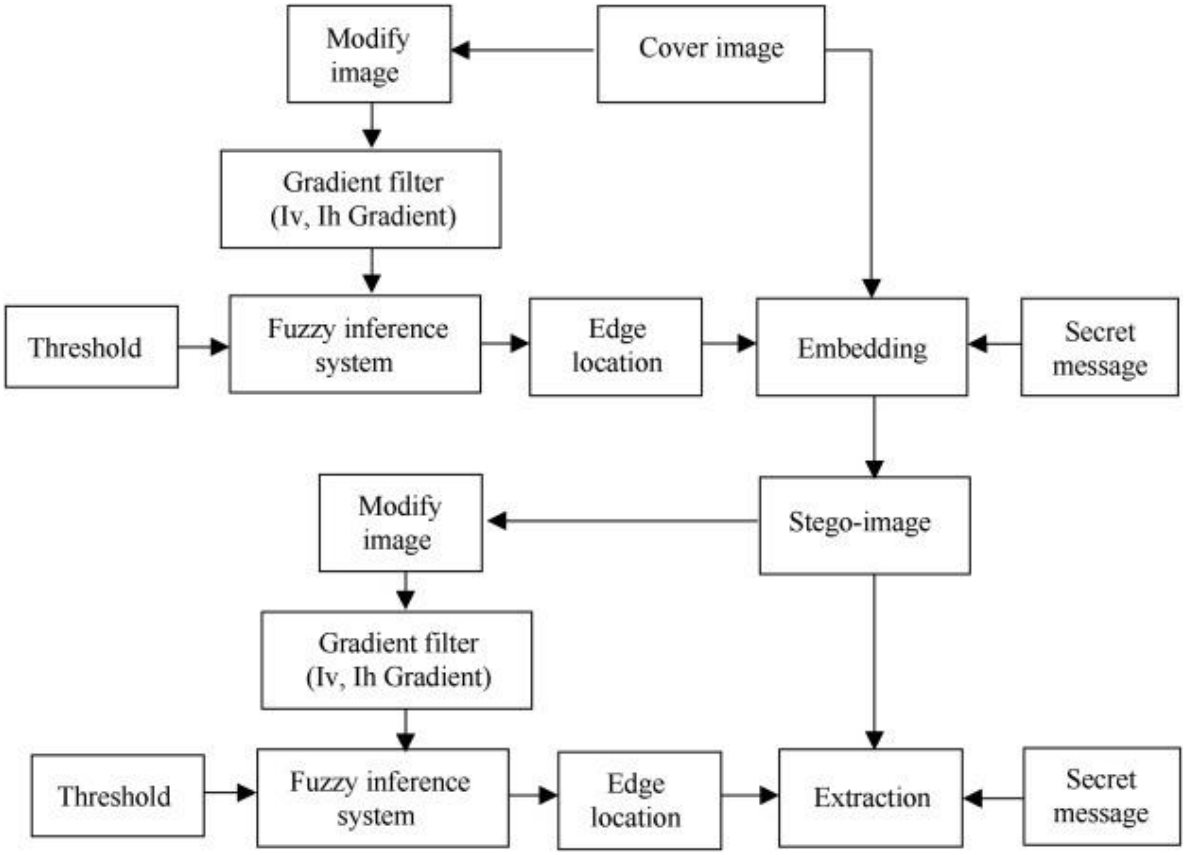

Рисунок 2.3 – Приклад блок-схема адаптивної стеганографії

Детальніше розглянемо сам процес, що зображено на рисунку 2.3.

1) Зображення обкладинки в градаціях сірого перетворюється на подвійну точність і масштабується до діапазону від 0 до 1.

2) Два найменших біта кожного пікселя тепер обнуляються перед визначенням країв зображення. Оскільки LSB кожного пікселя зображення не настільки важливі, тому лише MSB (Most Significant Bit) кожного пікселя зображення сприяють ідентифікації країв. Згенероване таким чином змінене зображення тепер передається через детектор нечітких країв як вхід.

3) Застосовуємо градієнтний фільтр, щоб знайти розриви в однорідній області. Виміряємо градієнт зображення вздовж горизонтального (Ih) і вертикального (Iv) положення.

4) Розробка системи нечіткого висновку FIS (Fuzzy Inference System) для виявлення меж. Вказуємо градієнти зображення Ih та Iv як вхідні дані FIS. Знаходимо ступінь гостроти μmn у позиції (m,n). Ми можемо змінити значення стандартного

відхилення функції належності, щоб налаштувати продуктивність детектора країв. Більше значення робить систему менш чутливою до країв зображення та зменшує інтенсивність виявлених країв.

5) Тепер ми отримуємо точне розташування країв, які використовуються для вставки секретних бітів у зображення обкладинки.

6) Вставляємо два біти секретного повідомлення в LSB ідентифікованого крайового пікселя зображення обкладинки. Тому MSB кожного пікселя стего-зображення та зображення обкладинки абсолютно однакові, що забезпечує точне отримання секретного повідомлення одержувачем.

7) Процес вилучення є протилежністю процесу вбудовування. Точне вилучення секретної інформації відбувається завдяки тому, що краї обкладинки та стегозображення однакові.

Переваги:

– висока стійкість до статистичного аналізу та атак;

– можливість оптимізації методу для конкретного носія.

Недоліки:

– складність реалізації та адаптації до різних типів носіїв;

– потенційне зниження ємності приховування через адаптацію.

Також можна провести аналіз та порівняння стегосистем за характеристиками.

1) Ємність приховування: ємність приховування відноситься до кількості інформації, яку можна приховати в носії без значного погіршення якості носія або збільшення його розміру. Різні методи стеганографії мають різну ємність приховування, яка залежить від типу носія та обраної техніки.

2) Стійкість до атак: стійкість до атак стосується здатності стегосистеми витримати спроби виявлення, вилучення або зміни прихованої інформації. Різні стегосистеми мають різні рівні стійкості до атак, що залежать від використовуваних методів приховування та стегоаналізу.

3) Візуальна якість та непомітність: візуальна якість та непомітність стегосистеми відносяться до ступеня видимості прихованої інформації та впливу приховування на якість носія. Ці характеристики зазвичай важливі для зображень, відео та аудіо носіїв, де зміни якості можуть привернути увагу до наявності прихованої інформації.

4) Складність реалізації та обчислювальна витрата: складність реалізації та обчислювальна витрата стосуються часу та ресурсів, необхідних для реалізації стегосистеми. Деякі стегосистеми можуть бути складнішими в реалізації та вимагати більше обчислювальної потужності, що може обмежувати їх практичне застосування.

2.4 Вбудовування даних на основі фрактальних алгоритмів

Стеганографія на основі фракталів – це інноваційний підхід, який використовує властивості фракталів для приховування інформації. Фрактали є геометричними структурами, які відрізняються самоподібністю на різних масштабах. Вони можуть бути використані для створення високоефективних і гнучких стегосистем.

Основна ідея стеганографії на основі фракталів полягає в тому, що візуально непомітні варіації в геометрії фрактала можуть бути використані для кодування прихованої інформації. Однією з переваг цього підходу є висока стійкість до шумів, оскільки малі зміни в структурі фрактала важко помітні.

Крім того, завдяки їхній самоподібності, фрактали можуть бути масштабовані без втрати якості, що може бути використано для створення стегосистем, здатних працювати з різними розмірами даних.

Більш того, через властивості самоподібності фракталів, стеганографія на основі фракталів може надати високу гнучкість та об'ємність приховування інформації. Це означає, що вона може бути адаптована до великого діапазону розмірів даних та медіа-контейнерів. На рисунку 2.4 можна побачити приклад фрактального зображення.

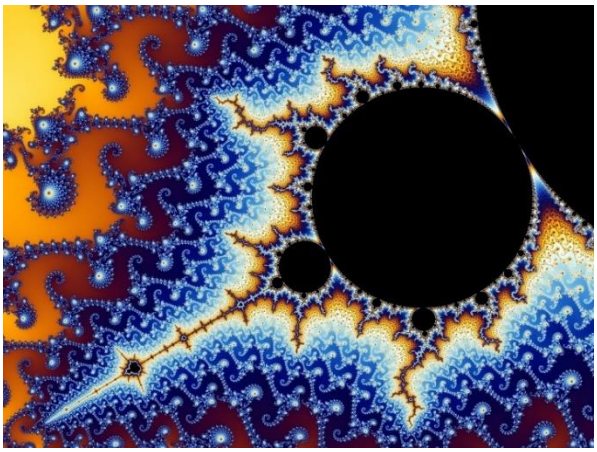

Рисунок 2.4 – Зображення фракталу

Однак, стеганографія на основі фракталів також має власні виклики. Одним із них є складність алгоритмів фрактального кодування, яка може призвести до більш високих вимог до обчислювальних ресурсів в порівнянні з більш простими методами, такими як LSB. Крім того, щоб ефективно використовувати цей метод, може знадобитися додаткове дослідження та розробка в області фрактального кодування та декодування.

Незважаючи на ці виклики, стеганографія на основі фракталів залишається перспективним напрямком, який може надати нові можливості для захисту інформації в цифровому світі. Її потенціал для високої стійкості до атак, великої об'ємністю приховування даних та адаптивності до різних медіа-контейнерів робить її цікавим вибором для подальших досліджень та розробок у галузі стеганографії.

Переваги стеганографії на основі фракталів.

1) Стійкість до атак: завдяки візуально непомітним варіаціям в геометрії фрактала, дані, вбудовані за допомогою фрактальної стеганографії, важко виявити та вилучити.

2) Висока об'ємність: фрактали мають властивість самоподібності, що дозволяє вбудовувати дані на різних масштабних рівнях, що забезпечує високу об'ємність приховування інформації.

3) Гнучкість: фрактальна стеганографія може бути адаптована до різних типів медіа-контейнерів, включаючи зображення, аудіо та відео.

Недоліки стеганографії на основі фракталів.

1) Складність алгоритмів: алгоритми фрактального кодування та декодування є досить складними, що може призвести до більших вимог до обчислювальних ресурсів в порівнянні з іншими методами стеганографії.

2) Потреба в додатковому дослідженні: оскільки фрактальна стеганографія є відносно новою областю, вона може вимагати додаткового дослідження і розробки, щоб досягти повного потенціалу.

3) Втрати при компресії: фрактальна стеганографія може бути вразлива до втрати даних при компресії, оскільки компресія може змінити структуру фрактала та вплинути на приховані дані.

З урахуванням проведеного аналізу та порівняння, можна зробити висновок, що кожна стегосистема має свої переваги та недоліки. Вибір конкретної стегосистеми

залежить від специфічних вимог та обмежень, таких як потреба в ємності приховування, стійкості до атак, візуальній якості та обчислювальних витратах.

Для отримання оптимальних результатів може бути корисним використовувати гібридні підходи, які поєднують кілька методів стеганографії, адаптуючи їх до конкретних вимог і обмежень. Це може включати комбінації адаптивної стеганографії з трансформаційними методами або інші техніки, які дозволяють досягти оптимального балансу між ємністю приховування, стійкістю до атак та іншими важливими характеристиками.

Для розробки нових, ефективніших стегосистем та покращення існуючих, важливо продовжувати дослідження та експерименти з різними методами приховування інформації, а також розробляти нові методи стегоаналізу для виявлення та протидії стеганографічним загрозам. Крім того, розробка та використання стандартів та рекомендацій для оцінки та порівняння стегосистем може сприяти визначенню оптимальних підходів та технологій для різних застосувань та сценаріїв.

3 СТЕГОАНАЛІЗ ТА ЙОГО ПРАКТИЧНЕ ЗАСТОСУВАННЯ

Стегоаналіз – це процес дослідження медіафайлів, таких як зображення, аудіо чи відео, з метою виявлення наявності прихованої інформації, яка передається за допомогою стегосистем. Стегоаналіз має на меті виявити наявність стеганографії та, за можливості, відновити приховану інформацію. Розглянемо детальніше основні аспекти стегоаналізу.

Методи стегоаналізу. Існують різні методи стегоаналізу, які можуть бути класифіковані наступним чином:

– візуальний аналіз: полягає у візуальному порівнянні оригінального та стеганографічного зображення з метою виявлення відмінностей, що вказують на наявність прихованої інформації;

– статистичний аналіз: включає вивчення статистичних характеристик медіафайлів, таких як гістограми, спектральний аналіз, кореляційні відносини тощо, для виявлення аномалій, які можуть свідчити про наявність стеганографії;

– машинне навчання та штучний інтелект: застосування алгоритмів машинного навчання та штучного інтелекту для аналізу та класифікації медіафайлів на основі їхніх характеристик і властивостей для виявлення стеганографічних активностей.

Проблеми та обмеження стегоаналізу: виявлення стеганографії може бути складним завданням, особливо в разі використання сучасних стегосистем, які намагаються мінімізувати відмінності між оригінальним та стеганографічним файлами. Обмеження стегоаналізу включають складність аналізу великих медіафайлів, обмеження доступу до оригінальних файлів для порівняння, а також можливі помилки у класифікації файлів, які можуть призвести до хибно позитивних чи хибно негативних результатів.

Застосування стегоаналізу. Стегоаналіз може бути використаний у різних сферах, включаючи:

– безпека корпоративних мереж: виявлення спроб передачі конфіденційної інформації через стеганографію, з метою запобігання витокам даних і забезпечення захисту корпоративних інтересів;

– боротьба з цифровою контрабандою: виявлення стеганографічних методів, які можуть бути використані для передачі незаконного контенту, такого як копірайтинг, порнографія, терористична інформація тощо.

– розвідка та контррозвідка: використання стегоаналізу для виявлення спроб передачі секретної інформації сторонніми державами або зловмисниками, з метою забезпечення національної безпеки та контролю за передачею інформації.

3.1 Приховування інформації у зображенні

Розглянемо один з прикладів приховування інформації, а саме передача секретного повідомлення у зображенні. За основу було взято зображення у якому уже наявне секретне повідомлення на рисунку 3.1 можна побачити зображення для дослідження.

Рисунок 3.1 – Зображення для аналізу

Першим кроком спробуємо отримати мета дані даного зображення. Це можливо за допомогою команди Kali Linux – «exiv2». На рисунку 3.2 показано команду та наявні опції.

```
┌──(vagrant㉿kali)-[~]
└─$ exiv2 -h
Usage: exiv2 [ options ] [ action ] file ...

Manipulate the Exif metadata of images.

Actions:
  ad | adjust   Adjust Exif timestamps by the given time. This action
                requires at least one of the -a, -Y, -O or -D options.
  pr | print    Print image metadata.
  rm | delete   Delete image metadata from the files.
  in | insert   Insert metadata from corresponding *.exv files.
                Use option -S to change the suffix of the input files.
  ex | extract  Extract metadata to *.exv, *.xmp and thumbnail image files.
  mv | rename   Rename files and/or set file timestamps according to the
                Exif create timestamp. The filename format can be set with
                -r format, timestamp options are controlled with -t and -T.
  mo | modify   Apply commands to modify (add, set, delete) the Exif and
                IPTC metadata of image files or set the JPEG comment.
                Requires option -c, -m or -M.
  fi | fixiso   Copy ISO setting from the Nikon Makernote to the regular
                Exif tag.
  fc | fixcom   Convert the UNICODE Exif user comment to UCS-2. Its current
                character encoding can be specified with the -n option.

Options:
   -h      Display this help and exit.
   -V      Show the program version and exit.
   -v      Be verbose during the program run.
   -q      Silence warnings and error messages during the program run (quiet).
   -Q lvl  Set log-level to d(ebug), i(nfo), w(arning), e(rror) or m(ute).
   -b      Show large binary values.
   -u      Show unknown tags.
   -g key  Only output info for this key (grep).
   -K key  Only output info for this key (exact match).
   -n enc  Charset to use to decode UNICODE Exif user comments.
   -k      Preserve file timestamps (keep).
   -t      Also set the file timestamp in 'rename' action (overrides -k).
   -T      Only set the file timestamp in 'rename' action, do not rename
```

Рисунок 3.2 – Виконання команди у консолі

З виводу команди «exiv2 pr hiding_cat.jpg», де параметр «pr» – print показує нам метадані заданого зображення hiding_cat.jpg, що показано на рисунку 3.3.

```
┌──(vagrant㉿kali)-[~/Downloads]
└─$ exiv2 pr hiding_cat.jpg
File name       : hiding_cat.jpg
File size       : 17363196 Bytes
MIME type       : image/jpeg
Image size      : 6000 x 7000
hiding_cat.jpg: No Exif data found in the file
```

Рисунок 3.3 – Виконання команди у консолі

Метадані зображення містять різні види інформації, такі як розмір файлу, роздільну здатність, формат файлу, дату створення та модифікації, параметри камери, автора зображення та інше. Проте, метадані самі по собі, як правило, не надають безпосередніх доказів приховування інформації в зображенні.

Втім, аналіз метаданих може вказувати на підозрілі аспекти зображення, які можуть стати причиною для проведення подальшого аналізу з метою виявлення стеганографії. Ось деякі можливі підказки, які можуть виявити метадані:

– нестандартні або відсутні метадані. Наявність нестандартних метаданих або їх відсутність може вказувати на те, що зображення було змінено або оброблено, і може містити приховану інформацію;

– невідповідність розмірів файлів. Значне збільшення розміру файлу, яке не пояснюється змінами в якості або роздільній здатності, може свідчити про наявність прихованих даних в зображенні;

– неспівпадіння дати створення та модифікації. Якщо дата модифікації зображення значно відрізняється від дати створення, це може вказувати на те, що зображення було змінено з метою вбудовування прихованих даних.

У нашому випадку можна побачити, що файл має великий розмір – «17363196 bytes».

Можемо зробити додаткову перевірку на градієнт зображення, чи не приховано на самому зображенні тексту, рисунок 3.4.

Рисунок 3.4 – Градієнт зображення

Більш докладний аналіз можна знайти у додатку А, де був проведений аналіз зображення з різними фільтрами на рисунку А.1, рисунку А.2, рисунку А.3.

Використовуючи функціонал Kali Linux за допомогою команди «steghide» будемо проводити аналіз далі.

На рисунку 3.5 зображено опис параметрів команди.

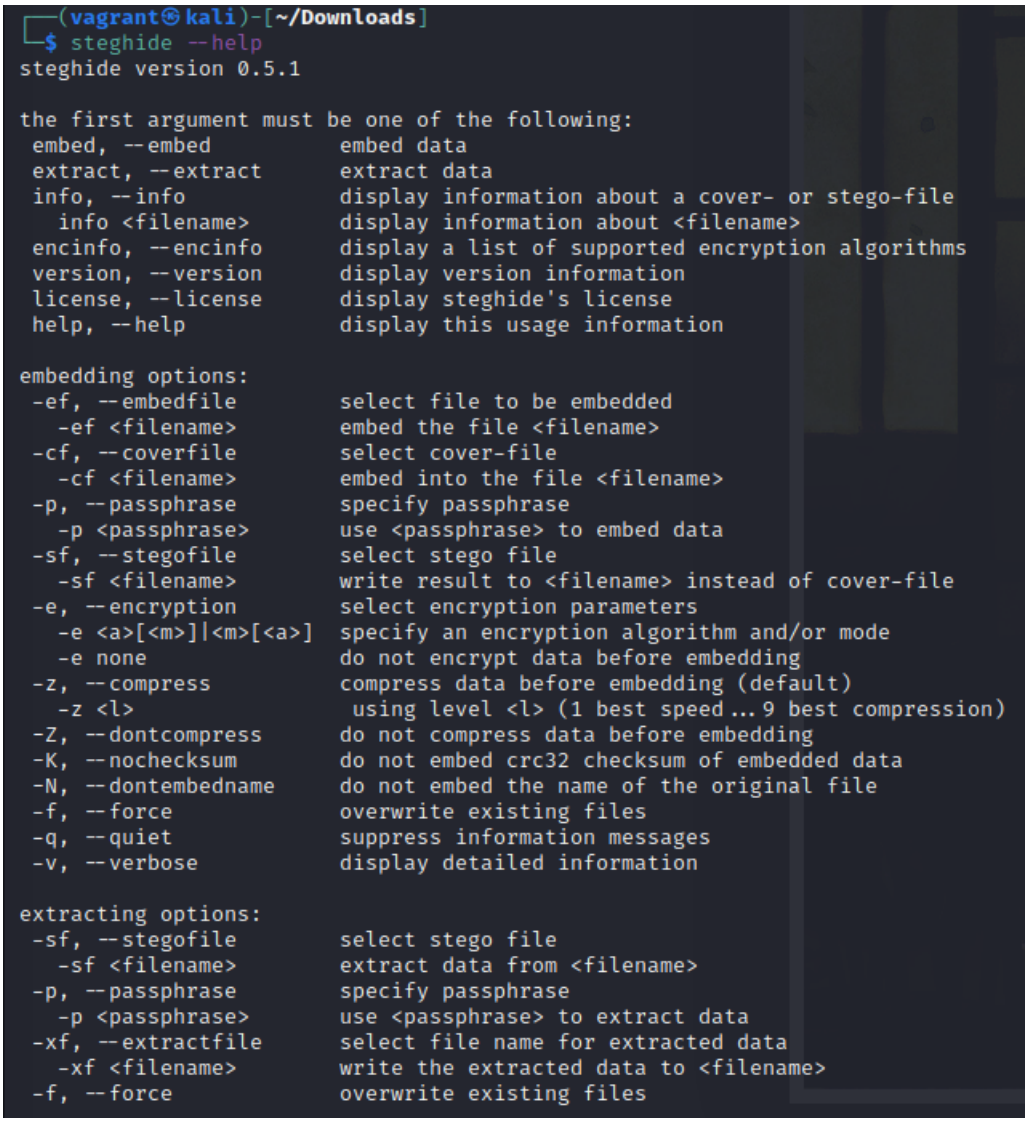

Рисунок 3.5 – Параметри команди у консолі

Використаємо дану команду у виді «steghide extract -sf Downloads/spy_cat.jpg» для нашого зображення наступним чином на рисунку 3.6, де параметр «-sf» вказує на файл.

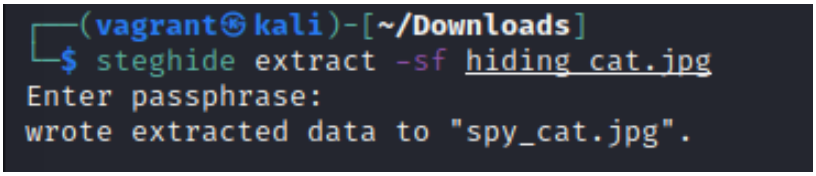

Рисунок 3.6 – Запуск команди у консолі

Після даної операції не вводивши ключа, вивід команди показав, що у першому зображенні було приховано ще одне зображення «spy_cat.jpg», його можна побачити на рисунок 3.7, але ми так і не отримали приховану фразу.

Рисунок 3.7 – Отримане нове зображення

Проведемо такий самий аналіз як із вихідним зображення, що можна знайти у додатку Б, а саме на рисунку Б.1, рисунку Б.2, рисунку Б.3, рисунку Б.4.

Та спробуємо застосувати нашу команду «steghide extract -sf» для нового отриманого зображення – «steghide extract -sf spy_cat.jpg» рисунок 3.8.

```
┌──(vagrant㉿kali)-[~/Downloads]
└─$ steghide extract -sf spy_cat.jpg
Enter passphrase:
wrote extracted data to "flag.txt".
```

Рисунок 3.8 – Застосування команди для нового зображення

На даному етапі ми вже отримали текстовий файл «flag.txt». Слід перевірити цей файл відкривши його у додатку. Результат показав, що все ж таки у другому файлі було приховано текст повідомлення, що знаходився у файлі «flag.txt». Отримане

повідомлення «ctf{fl465_w1ll_n07_h1d3_fr0m_y0u}» у відкритому файлі за допомогою командного рядку на рисунку 3.9.

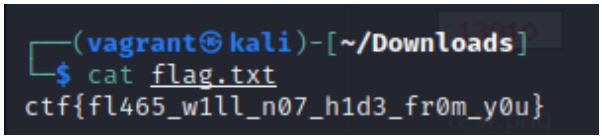

Рисунок 3.9 – Прихований файл з повідомленням

3.2 Розповсюдження спектру

У аудіо стеганографії одним із методів розповсюдження спектру – намагається поширювати секретну інформацію по частотному спектру аудіосигналу. Це схоже на LSB, яка поширює біти повідомлень випадковим чином по всьому звуковому аудіо файлу. Проте, на відміну від кодування LSB, метод розповсюдження спектру поширює секретну інформацію по частотному спектру звукового файлу за допомогою коду, який не залежить від вихідного сигналу [7]. Результатом є кінцевий сигнал, що займає смугу пропускання, яка перевищує те, що дійсно потрібно для передачі. Метод розповсюдження спектру сприяє поліпшенню продуктивності в певній області, порівняно з кодуванням LSB, оскільки він забезпечує помірну швидкість передачі даних і високий рівень надійності відносно методів видалення. Проте метод розповсюдження спектру має один основний недолік, який може вводити шум у звуковий файл.

Також є спосіб так названий «Приховування відлуння». Техніка приховування відлуння використовує секретну інформацію у звуковому файлі, вводячи відлуння в дискретний сигнал. Приховування відлуння має переваги:

– забезпечення високої швидкості передачі даних;
– вища надійність.

Лише один біт секретної інформації може бути закодований, якщо тільки вихідний сигнал отримав лише одне відлуння. Отже, перед початком процесу кодування оригінальний сигнал розбитий на блоки. Після завершення процесу кодування блоки об'єднуються разом, щоб створити остаточний сигнал [9].

Для успішного приховання даних, три параметри відлуння повинні бути різними:

– амплітуда;

– швидкість розпаду;

– зміщення (час затримки) від вихідного сигналу.

Усі три параметри встановлені нижче порогу слуху людини, тому відлуння не може бути легко виявлено. Крім того, зміщення змінюється, щоб представляти бінарне повідомлення для кодування. Одне значення зміщення являє собою двійкове значення, а значення другого зміщення являє собою двійковий нуль. Зміщення представлено на рисунку 3.10.

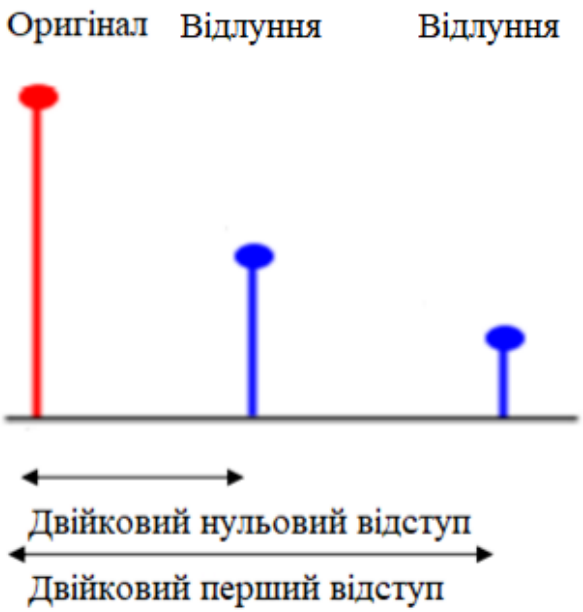

Рисунок 3.10 – Значення відступів відтворюють двійкові значення

Якщо вихідний сигнал був вироблений лише одним відлуння, кодування може містити лише один біт інформації. Тому початковий сигнал розбивається на блоки, перш ніж процес кодування починається. Після завершення процесу кодування, блоки об'єднуються разом, щоб створити остаточний сигнал [10]. Тепер ми пройдемо просту форму процесу приховування відлуння, використовуючи повідомлення "HEY". Для стислості сигнал буде повністю розділений на блоки, хоча за звичайних умов випадкова кількість зразків між кожною парою блоків повинна залишатись

невикористаною, щоб зменшити ймовірність виявлення. Перестановка остаточних блоків представлена на рисунку 3.11.

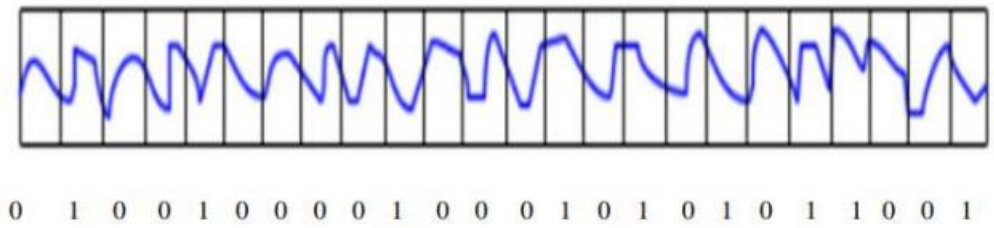

Рисунок 3.11 – Блоки переставляються для отримання остаточного сигналу

Спочатку сигнал ділиться на блоки, і кожному блоку присвоюється один чи нуль, що базується на секретному повідомленні. У цьому випадку повідомлення є двійковим еквівалентом "HEY". Використовуючи цю реалізацію процесу приховування відлуння, зазвичай може бути отриманий сигнал, який має досить помітний набір відлуння, що підвищує ризик виявлення. Друга реалізація процесу приховування відлуння вирішує цю проблему. Спочатку відлуння створюється з усього вихідного сигналу, використовуючи значення двосмугового зсуву нуля. Потім другий сигнал відлуння створюється з усього вихідного сигналу, використовуючи значення двосмугового зсуву. Щоб об'єднати два відлуння разом, щоб отримати остаточне кодування, використовуються два сигнали змішувача. Сигнали змішувачі мають значення як одиниці, так і нулі, в залежності від того, який біт потрібно кодувати в блоці. "Один" сигналу відлуння помножується на "один" сигнал змішувача, а "нульовий" сигнал відлуння помножується на "нульовий" сигнал змішувача. Потім два результати додаються разом, щоб отримати остаточний сигнал. Остаточний сигнал є менш різким, ніж той, який був отриманий за допомогою першої реалізації приховування відлуння. Це пояснюється тим, що два ефекти змішувача є доповненнями один до одного, і ці переходи використовуються в кожному сигналі. Ці дві характеристики сигналів змішувача забезпечують більш плавні переходи між відлуннями. Щоб витягти таємне повідомлення зі стего-сигналу, приймач повинен мати можливість розбити сигнал на той же блоковий порядок, який використовується під час процесу кодування.

4 РОЗРОБКА ТА АПРОБАЦІЯ НОВОГО МЕТОДУ ПРИХОВУВАННЯ ІНФОРМАЦІЇ У СТЕГОСИСТЕМАХ

При створенні програмного продукту були використані такі способи для програмуванні на мові Python, як Microsoft Visual Studio Code 2022.

Python дуже проста та повноцінна мова програмування, за допомогою якої можна отримати багато засобів для структурування і написання програм та рішень. Легко виправляти помилки та знайти потрібне рішення тієї чи іншої проблеми. Мова має усі потрібні бібліотеки для роботи з стеганографією.

На основі мови програмування було створено програмний код та легкий інтерфейс для взаємодії.

4.1 Опис запропонованого методу

Метою розробки власної стегосистеми є досягнення оптимального балансу між ємністю приховування, стійкістю до атак, візуальною якістю та обчислювальними витратами.

Як правило, будь-який метод стеганографії може бути схожий на вже існуючі методи в різних аспектах. Однак унікальність методу полягає в тому, як саме він комбінує різні елементи для досягнення кінцевої мети стеганографії.

Метод, заснований на використанні фрактальної геометрії для стеганографії, є відносно новим і не так широко поширеним, тому можна вважати його унікальним у тому сенсі, що він не є загальновідомим і часто використовуваним методом. Однак, можливо, він уже був використаний в якихось конкретних дослідженнях або додатках.

Метод стеганографії, заснований на використанні фрактальної геометрії, полягає в застосуванні фрактальних алгоритмів для приховування інформації в медіафайлах, таких як зображення, аудіо чи відео. Фрактальна геометрія – це галузь математики, що вивчає фрактали, самоподібні та складні структури, які часто зустрічаються в природі та мистецтві.

Стеганографія на основі фракталів використовує фрактальні алгоритми та властивості фракталів для вбудовування та відновлення прихованих повідомлень у

зображеннях. Фрактальні алгоритми мають характеристики самоподібності, випадковості та масштабності, які можуть бути використані для стеганографії.

Розглянемо кроки для реалізації стеганографічного методу на основі фракталів.

1) Кодування повідомлення: перетворюємо текстове повідомлення, яке потрібно приховати, у двійковий формат.

2) Генерація фрактального зображення: використовуємо алгоритм створення фрактального зображення, наприклад алгоритм Ляпунова або системи ітерованих функцій IFS (Iterated Function System), для генерації зображення, у якому буде вбудоване приховане повідомлення. Параметри фрактального алгоритму повинні бути відомі та незмінні, щоб можна було відтворити зображення при відновленні повідомлення.

3) Вбудовування повідомлення: використаємо фрактальні властивості зображення для вбудовування двійкового повідомлення в певні області або пікселі зображення. Наприклад, можна вбудовувати повідомлення у координати атракторів фракталу або у параметри системи ітерованих функцій IFS. Це може включати заміну окремих бітів у згаданих координатах або параметрах двійковими бітами повідомлення.

4) Збереження стеганографічного зображення: зберігаємо отримане фрактальне зображення з вбудованим прихованим повідомленням як кінцевий результат.

У свою чергу для відновлення прихованого повідомлення зі стегографічного фрактального зображення, розглянемо наступні кроки.

1) Генерація оригінального фрактального зображення: відтворюємо оригінальне фрактальне зображення, використовуючи ті самі параметри та алгоритм, які були використані під час вбудовування повідомлення.

2) Виявлення різниці: виявляємо різницю між оригінальним фрактальним зображенням та стеганографічним фрактальним зображенням. Це допоможе визначити області або пікселі, де було вбудоване приховане повідомлення.

3) Відновлення повідомлення: відновлюємо двійкове повідомлення, витягуючи відповідні біти з різниці між оригінальним та стеганографічним зображеннями. Використовуйте знання про спосіб вбудовування повідомлення

(наприклад, в яких координатах атракторів фракталу або параметрах IFS були внесені зміни) для відновлення правильного порядку бітів.

4) Декодування тексту: перетворюємо двійкове повідомлення назад у текстовий формат, щоб прочитати приховане повідомлення.

Один з можливих способів вбудовування повідомлення в координати фракталу – використовувати найменш значущі біти LSB координат, але цей метод може бути недостатньо стійким до атак. Іншим варіантом може бути використання характеристик фракталу, таких як збіжність або розбіжність точок, для кодування повідомлення.

Проте, варто врахувати, що такі методи можуть бути складні для реалізації та можуть вимагати значних обчислювальних ресурсів для генерації фракталів та відновлення прихованих повідомлень.

Використання характеристик фракталу, таких як збіжність або розбіжність точок, для кодування повідомлення вимагає обрання відповідного фрактального алгоритму та відповідного методу вбудовування. Один з можливих підходів – використання фрактального алгоритму Мандельброта або Жюліа для генерації зображення з відповідними збіжними та розбіжними точками.

Ось один із способів використання збіжності та розбіжності точок для кодування повідомлення.

1) Кодування повідомлення: перетворюємо текстове повідомлення, яке потрібно приховати, у двійковий формат.

2) Генеруємо фрактальне зображення: використовуємо фрактальний алгоритм Мандельброта для генерації зображення, у якому буде вбудоване приховане повідомлення. Вибираємо відповідні параметри для алгоритму, щоб отримати зображення з чітко видимими збіжними та розбіжними областями.

3) Вбудовуємо повідомлення: використовуємо збіжність та розбіжність точок для вбудовування двійкового повідомлення. Наприклад, можна змінювати кількість ітерацій для досягнення збіжності в збіжних точках або змінювати параметри алгоритму таким чином, щоб розбіжність точок відображала приховане повідомлення.

4) Та зберігаємо стеганографічне зображення: зберігаємо отриманий контейнер з нейтральним зображення для того щоб відкликати підозру, що ми щось приховуємо, та схованим у ньому фракталом з закодованим повідомленням.

Для відновлення прихованого повідомлення зі стеганографічного фрактального зображення, виконаємо такі кроки.

1) Генерація оригінального фрактального зображення: відтворюємо оригінальне фрактальне зображення, використовуючи ті самі параметри та алгоритм, які були використані під час вбудовування повідомлення.

2) Виявлення змін: аналізуємо стеганографічне фрактальне зображення та порівняйте його з оригінальним зображенням, щоб виявити зміни у збіжності або розбіжності точок, які відповідають вбудованому повідомленню.

3) Відновлення повідомлення: відновлення двійкового повідомлення, витягуючи відповідні біти з виявлених змін у збіжності або розбіжності точок.

4) Декодування тексту: перетворюємо двійкове повідомлення назад у текстовий формат, щоб прочитати приховане повідомлення.

4.2 Математичне обґрунтування

Для самого методу було обрано саме множину Мандельброта. Множина Мандельброта – обмежена та зв’язна множина на комплексній площині, межа якої утворює фрактал. Це множина комплексних чисел «c», де функція ітерацій не розходиться, якщо її ітерувати від значення z = 0, тобто, для якої послідовність $f_c(0)$ залишається обмеженою в абсолютному значенні [11].

Фрактал Мандельброта генерується на основі ітераційної функції за формулою:

$$z_{n+1} = z_n^2 + c, \tag{4.1}$$

де $z_0 = 0$;

c – комплексне число, n – номер ітерації.

Приклад такого фракталу можно побачити на рисунку 4.1.

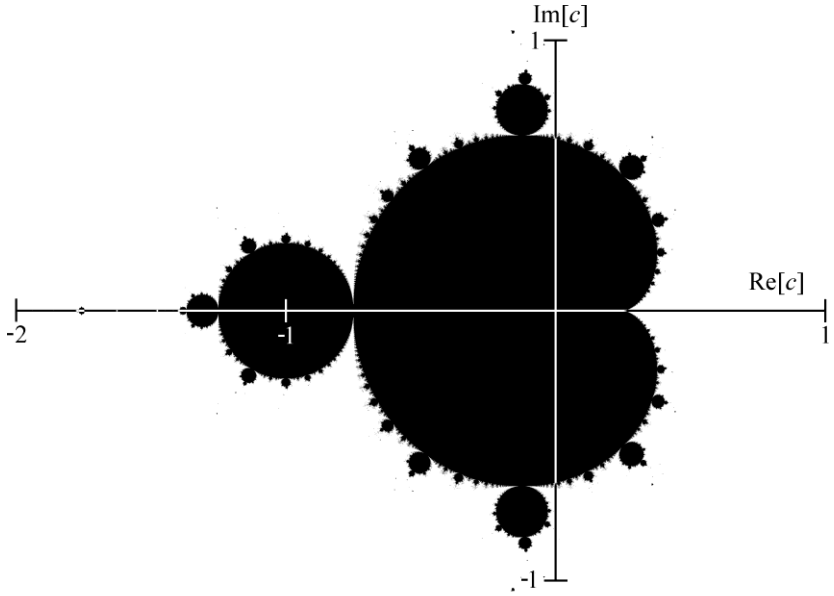

Рисунок 4.1 – Множина Мандельброта

Для встановлення приналежності точки «c» до фрактала Мандельброта, необхідно виконати ітераційну процедуру для кожного елемента комплексної площини. Точка визнається частиною фрактала, якщо відповідний їй ряд z_n демонструє збіжність. В іншому випадку, при розбіжності ряду, точка вважається не належною до фрактала.

Збіжність точок фракталу Мандельброта відповідає областям, де ітераційний процес призводить до збіжного ряду z_n. Розбіжність точок відповідає областям, де ітераційний процес призводить до розбіжного ряду.

Можна використовувати збіжність та розбіжність точок для вбудовування прихованого повідомлення, змінюючи кількість ітерацій або параметри алгоритму таким чином, щоб вони відповідали двійковому коду повідомлення.

Наприклад, можна використовувати кількість ітерацій, необхідних для досягнення збіжності, як параметр для вбудовування повідомлення. Якщо число ітерацій для збіжності точки кратне 2, можна вбудовувати біт «0», якщо число ітерацій для збіжності точки не кратне 2, можна вбудовувати біт «1». Аналогічно, можна змінювати параметри алгоритму таким чином, щоб розбіжність точок відображала приховане повідомлення.

Основне математичне обґрунтування полягає у тому, що ми використовуємо математичні властивості фракталів (зокрема, збіжність та розбіжність рядів) для

кодування та передачі інформації. Зміни у параметрах алгоритму або кількості ітерацій для досягнення збіжності дозволяють нам вбудовувати приховане повідомлення в зображення фрактала без значного впливу на його зовнішній вигляд.

Це обґрунтування базується на відомих математичних властивостях фракталів і може бути адаптовано для різних алгоритмів фракталів та методів вбудовування. Однак слід зазначити, що успіх методу залежить від вибору алгоритму, параметрів та деталей вбудовування повідомлення. Важливо знайти оптимальний баланс між збереженням візуальних характеристик фрактального зображення та забезпеченням достатньої кількості інформації для передачі прихованого повідомлення.

Ми вже розібрали як генерується фрактал Мандельброта, отже на основні цього ми будемо визначати певне ітераційне число «m», що необхідне для досягнення певного порогу збіжності і це число ітерацій можна використовувати для кодування повідомлення. Щоб вбудувати повідомлення, ми кодуємо його у двійковому форматі та модифікуємо значення «m» відповідно для кожного біту повідомлення.

Наприклад, якщо біт повідомлення дорівнює 0, ми можемо змінити «m» таким чином, щоб воно стало парним числом. Якщо біт повідомлення дорівнює «1», ми можемо змінити «m» таким чином, щоб воно стало непарним числом.

В процесі декодування ми відновлюємо двійкове представлення повідомлення, аналізуючи значення m для кожної точки «c». Якщо m парне, декодований біт дорівнює «0», якщо «m» непарне, декодований біт дорівнює 1. Після того, як ми відновили всі біти повідомлення, ми можемо перетворити їх назад у текстовий формат [12].

Щоб забезпечити надійність методу, ми використовуємо псевдовипадкове перемішування точок c, залежне від ключа. Це забезпечує, що лише знаючи ключ, можна відновити правильний порядок точок та декодувати повідомлення.

Для декодування повідомлення, вбудованого у зображення фракталу Мандельброта, вам потрібно знати наступні параметри.

1) Закодоване зображення – це зображення фракталу Мандельброта, у якому вже вбудоване повідомлення. Ви повинні мати доступ до цього зображення, щоб мати змогу декодувати повідомлення.

2) Довжина повідомлення – це кількість символів у повідомленні, яке було вбудоване у зображення. Знання довжини повідомлення допомагає зупинити процес декодування, коли всі біти повідомлення будуть відновлені.

3) Максимальна кількість ітерацій – це параметр, який визначає межу збіжності для фракталу Мандельброта. Цей параметр використовується під час генерації зображення фракталу, і його слід знати, щоб правильно відновити значення m для кожної точки.

4) Ключ – псевдовипадковий ключ, який використовується для перемішування точок фракталу перед вбудовуванням повідомлення. Знання ключа дозволяє відновити правильний порядок точок та декодувати повідомлення.

Для успішного декодування повідомлення нам потрібно знати закодоване зображення, довжину повідомлення, максимальну кількість ітерацій та ключ. З цією інформацією ми зможемо відновити вбудоване повідомлення зі зображення фракталу Мандельброта.

Отже, математичне обґрунтування для кодування та декодування полягає в тому, що значення «m» є унікальними для кожної точки фракталу, і ми можемо використовувати їх для передачі інформації. Значення m відповідає частоті збіжності чи розбіжності для кожної точки, і ця властивість використовується для вбудовування повідомлення. За допомогою ключа ми можемо гарантувати, що лише особи, які мають доступ до ключа, можуть правильно декодувати повідомлення.

4.3 Архітектура програмного забезпечення та його функціонал

У рамках даної роботи було розроблено відповідний програмний код для описаного методу та інтерфейс для легшої взаємодії з кодом.

Фрагмент коду програми можна побачити на рисунку 4.2 та код самого інтерфейсу користувача можна знайти у додатку В на рисунку В.1, рисунку В.2, рисунку В.3, рисунку В.4.

```python
import tkinter as tk
from tkinter import ttk
import subprocess
import imageio.v2 as imageio
import numpy as np
from PIL import Image, ImageTk

def on_encode_button_click():
    print("Encode Button clicked")
    message = entry_message.get()
    width = entry_width.get()
    height = entry_height.get()
    max_iter = entry_max_iter.get()
    key = entry_key.get()
    result = subprocess.run(["python3", "/home/vagrant/diplom/test_code.py", message, width, height, max_it
    output.insert(tk.END, result.stdout)

def on_decode_button_click():
    print("Decode Button clicked")
    encoded_image = entry_encoded_image.get()
    message_length = entry_message_length.get()
    key = entry_key_decoding.get()
    result = subprocess.run(["python3", "/home/vagrant/diplom/test_decode.py", encoded_image, message_lengt
    output_decoding.insert(tk.END, result.stdout)  # Use output_decoding instead of output

def open_fractal_image():
    image_path = "encoded_image.png"
    try:
        image = Image.open(image_path)
        image.thumbnail((500, 500))
        photo = ImageTk.PhotoImage(image)

        label_image.config(image=photo)
        label_image.image = photo
```

Рисунок 4.2 – Фрагмент коду інтерфейсу користувача

Так також фрагмент коду для введення даних через програмне забезпечення на рисунку 4.3.

```python
# Create labels and input Entry widgets for encoding parameters
label_message = tk.Label(encoding_tab, text="Message:")
label_message.pack()
entry_message = tk.Entry(encoding_tab)
entry_message.pack(pady=5)

label_width = tk.Label(encoding_tab, text="Width:")
label_width.pack()
entry_width = tk.Entry(encoding_tab)
entry_width.pack(pady=5)

label_height = tk.Label(encoding_tab, text="Height:")
label_height.pack()
entry_height = tk.Entry(encoding_tab)
entry_height.pack(pady=5)

label_max_iter = tk.Label(encoding_tab, text="Max Iterations:")
label_max_iter.pack()
entry_max_iter = tk.Entry(encoding_tab)
```

Рисунок 4.3 – Фрагмент коду для введення даних через інтерфейс користувача

Для легкої взаємодії з кодом було також написане відповідне програмне забезпечення, що зображено на рисунку 4.4 та усі вкладки у додатку В більш детально у повному обсязі на рисунку В.5, рисунку В.6, рисунку В.7, рисунку В.8.

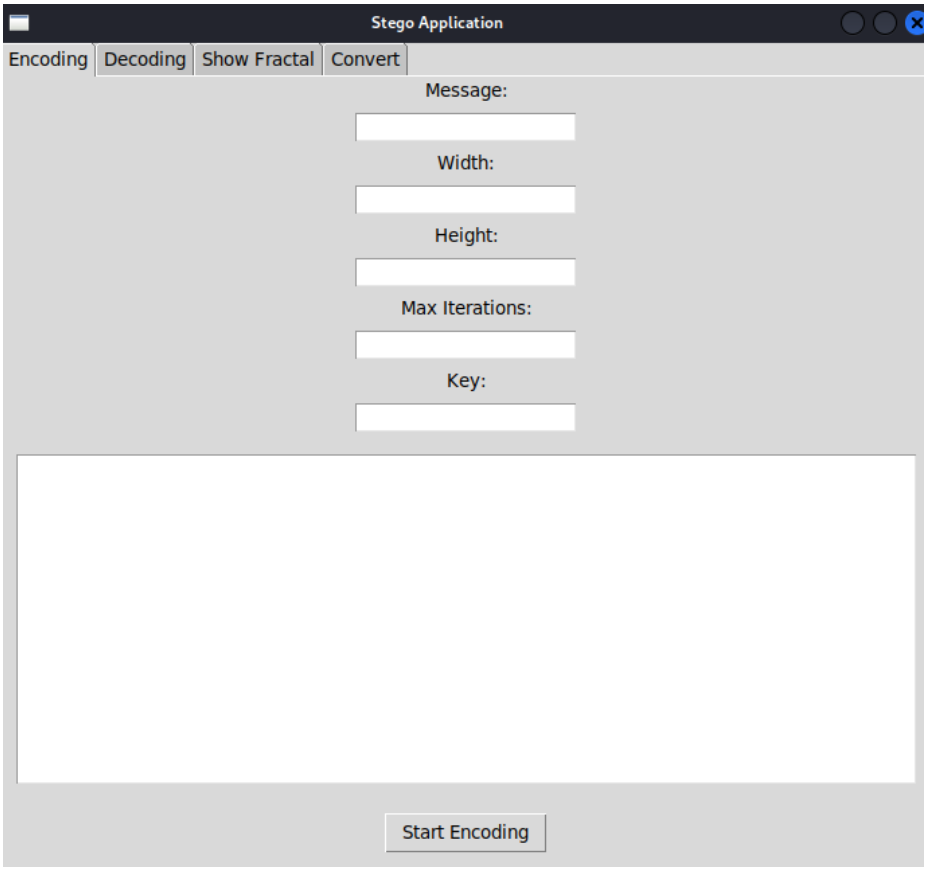

Рисунок 4.4 – Інтерфейс користувача програми «Stego Application»

Програмне забезпечення включає 4 основних вкладки.

1) «Encoding» – для кодування повідомлення у зображені.
2) «Decoding» – для декодування повідомлення з існуючим зображенням.
3) «Show fractal» – показати зображення згенерованого фракталу.
4) «Convert» – конвертувати файл «npy» у формат png.

Тепер розберемо, що саме відбувається у кожному з етапі кодування повідомлення у зображення фракталу та його ж декодування.

Програма приймає у себе вхідні данні, що задаються користувачем.

На рисунку 4.5 показані вхідні параметри для кодування та рисунку 4.6 для декодування відповідно.

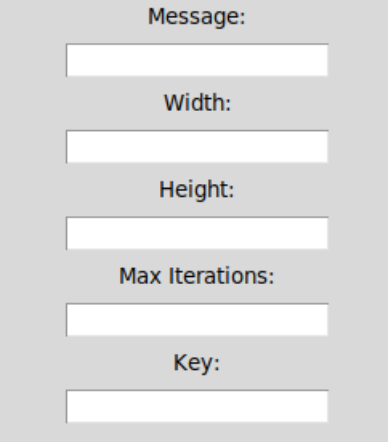

Рисунок 4.5 – Вхідні параметри для кодування

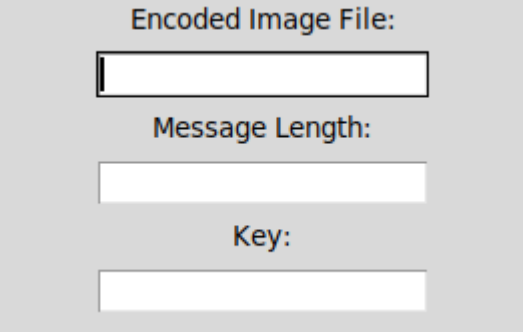

Рисунок 4.6 – Вхідні параметри для декодування

Важливий момент, коли при кодуванні ми передаємо повідомлення, то слід дізнатися його довжину, щоб передати потім другому користувачу для декодування.

Для цього в програмі є вікно для виведення інформації, трохи схоже на процес логування у системі .

На рисунку 4.7 можна побачити як саме це відбувається, коли ми вже запустили процес кодування.

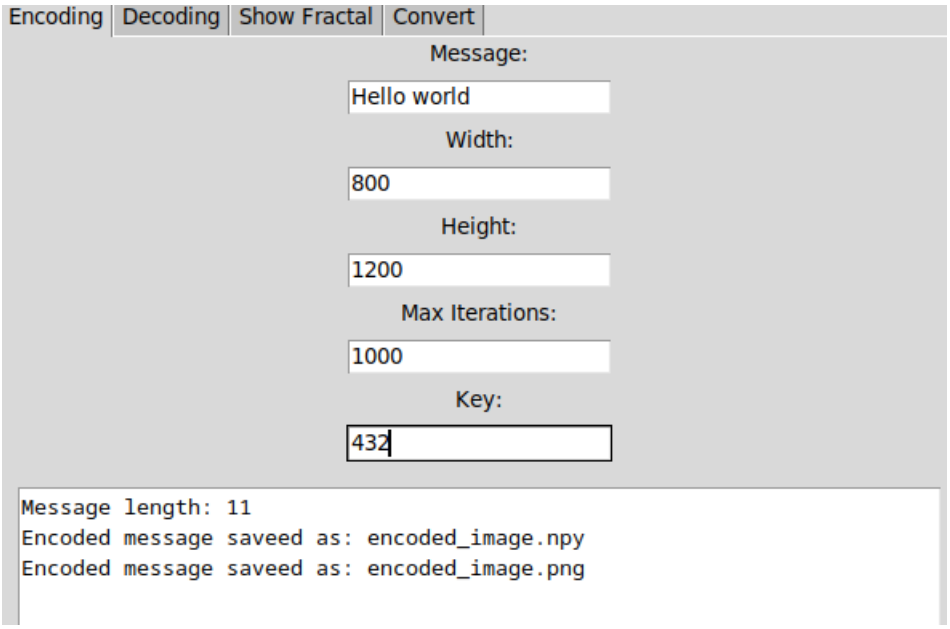

Рисунок 4.7 – Виведення інформації у вікно

Вікно відображає найголовнішу інформацію, а саме:

– кількість символів у повідомленні, що було передано через параметр Message;

– файли, що було збережено під час кодування повідомлення та їх назва.

Спочатку необхідно імпортувати потрібні бібліотеки, що знадобляться при роботі над дослідженням. Їх зображено на рисунку 4.8 для кодування та декодування, та на рисунку 4.9 для програми.

```
import sys
import numpy as np
import matplotlib.pyplot as plt
```

Рисунок 4.8 – Бібліотеки для кодування та декодування

```
import tkinter as tk
from tkinter import ttk
import subprocess
import imageio.v2 as imageio
import numpy as np
from PIL import Image, ImageTk
```

Рисунок 4.9 – Бібліотеки для програми

На рисунку 4.10 показана сама функція для фракталу Мандельброта.

```
def mandelbrot(c, max_iter):
    z = c
    for n in range(1, max_iter):
        z = z*z + c
        if abs(z) > 2:
            return n
    return max_iter
```

Рисунок 4.10 – Функція для множини Мандельброта

Параметр max_iter використовується у нашому коді для обмеження кількості ітерацій, які виконуються при обчисленні множини Мандельброта.

Також даний параметр контролює, скільки разів цей ітераційний процес виконується. Більша кількість ітерацій зазвичай веде до більш деталізованого зображення фракталу, але в той же час вимагає більше обчислювальних ресурсів.

У контексті стеганографії max_iter впливає на максимальну кількість інформації, яка може бути закодована в зображенні. Більше ітерацій може допомогти збільшити ємність для закодованої інформації, але може вплинути на видимість змін в зображенні, що може зробити стеганографічне повідомлення більш помітним.

Якщо після max_iter ітерацій значення z все ще не перевищує 2 (або точніше, abs(z) більше за 2), то вважається, що точка c належить множині Мандельброта.

У нашому коді параметр max_iter використовується для виклику функції mandelbrot (c, max_iter), яка повертає кількість ітерацій, які потрібно виконати, перш ніж значення z перевищить 2, або max_iter, якщо значення z не перевищує 2 після max_iter ітерацій.

Використання параметра max_iter дозволяє контролювати деталізацію зображення множини Мандельброта та, відповідно, час обчислень. Вище значення max_iter дасть більш детальне зображення множини Мандельброта, але також збільшить час обчислень. Нижче значення max_iter зменшить час обчислень, але зменшить деталізацію зображення множини Мандельброта.

Значення max_iter може варіюватися в залежності від того, наскільки детальним ми хочемо зробити зображення множини Мандельброта. Вибір великого значення

max_iter дозволить нам отримати більш детальні зображення, але це може збільшити час обчислень.

Значення max_iter може бути будь-яким цілим числом, яке ви вважаєте оптимальним для вашого випадку. Для початку можна використовувати max_iter від 100 до 1000. Ви можете експериментувати з різними значеннями і спостерігати, як змінюється зображення множини Мандельброта.

Менше значення max_iter:

– обчислюється швидше;

– менш деталізоване зображення (може втрачати деякі тонкі особливості множини).

Більше значення max_iter:

– обчислюється повільніше;

– більш деталізоване зображення (може відображати тонкі особливості множини).

Для зміни зображення фрактала множини Мандельброта в нашому коді можна змінювати декілька значень. Ось деякі з них.

1) «max_iter»: зміна значення max_iter (максимальна кількість ітерацій) впливає на деталізацію фрактала. Вище значення max_iter дозволить отримати більш детальний фрактал, але збільшить час обчислень. Зменшення max_iter призведе до менш детального зображення, але зменшить час обчислень.

2) Розмір зображення: зміна розміру зображення (ширина та висота) також вплине на відображення фрактала. Вищі розміри зображення дозволять побачити більше деталей, але збільшать час обчислень.

3) Межі координат: у нашому коді межі координат для множини Мандельброта задані за допомогою значення:

complex(x / img_size [1] * 3.5 - 2.5, y / img_size [0] * 2 - 1).

Зміна меж координат (наприклад, зміна множників або зсувів) дозволить вам змінювати видиму частину фрактала.

4) Колірна схема: зміна колірної схеми для зображення фрактала також може змінити його вигляд. У вашому коді колірна схема задається за допомогою аргументу

cmap у функції plt.imshow(). Замість «inferno» можна використати інші вбудовані колірні схеми Matplotlib, такі як «viridis», «plasma», «magma» або «cividis». Додаткові колірні схеми можна знайти в документації Matplotlib або створити власні.

Змінюючи ці параметри, можна отримати різні зображення фрактала множини Мандельброта. Приклад згенерованого Фракталу можна побачити на рисунку 4.11.

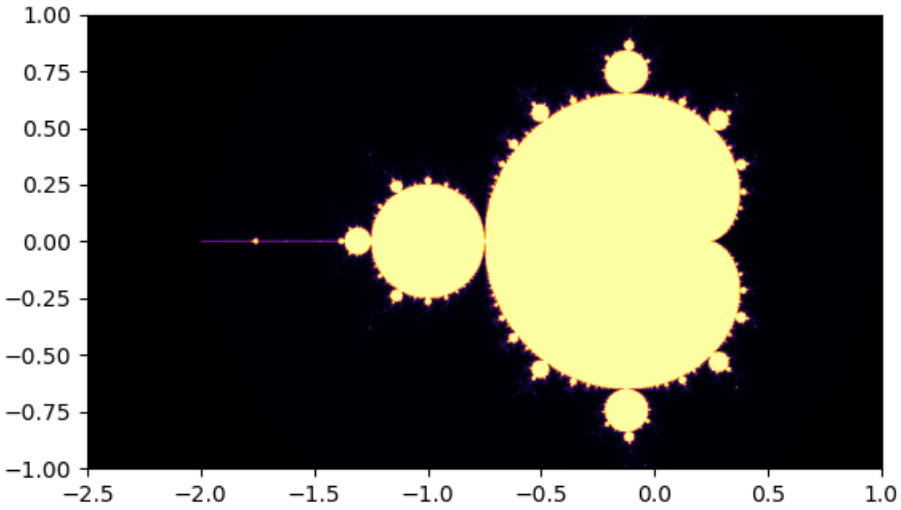

Рисунок 4.11 – Згенерований фрактал

Також у інтерфейсі користувача є додаткове вікно, для того, щоб одразу побачити фрактал, що було згенеровано у ході кодування. Рисунок 4.12 описує цей фунціонал відображання зображення фракталу.

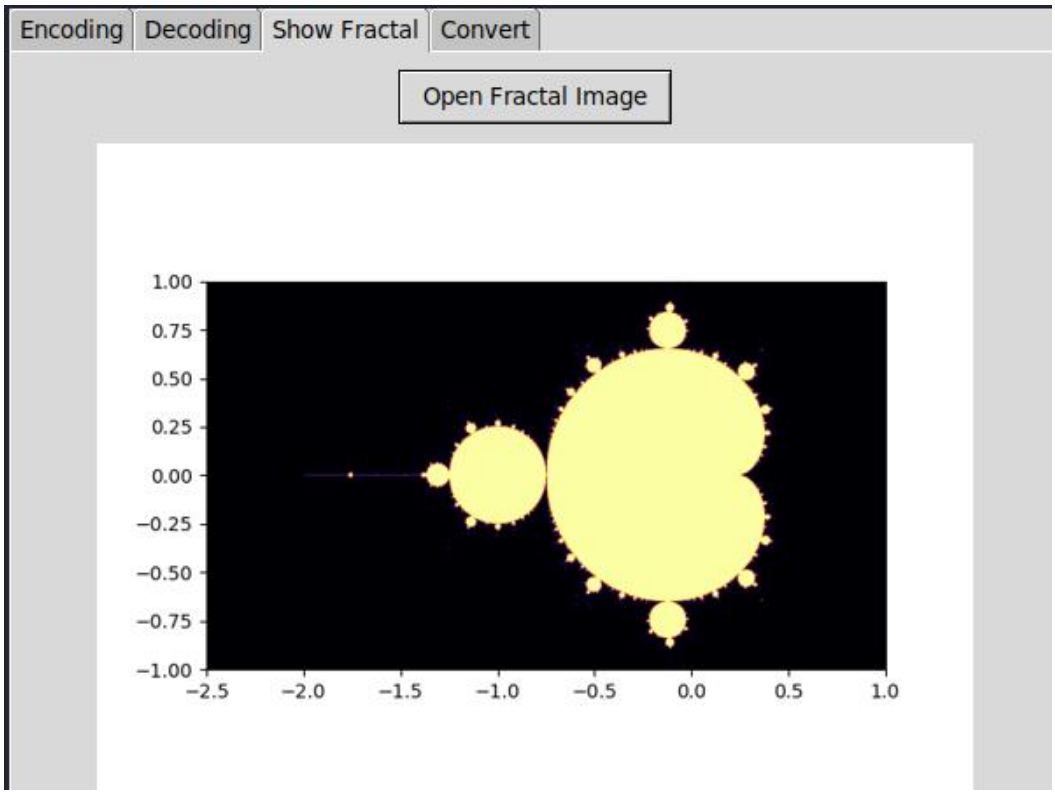

Рисунок 4.12 – Функція в інтерфейсі користувача для відображання згенерованого фракталу

Можна також побачити чутливу різницю при зміні параметру max_iter. Давайте згенеруємо два різних зображення з різною ітерацією.

На першому зображені фракталу буде використано значення 100 для параметру max_iter, на другом ж буде збільшено значення параметра до 1000 і відображено на наступних рисунках 4.13 та 4.14 відповідно.

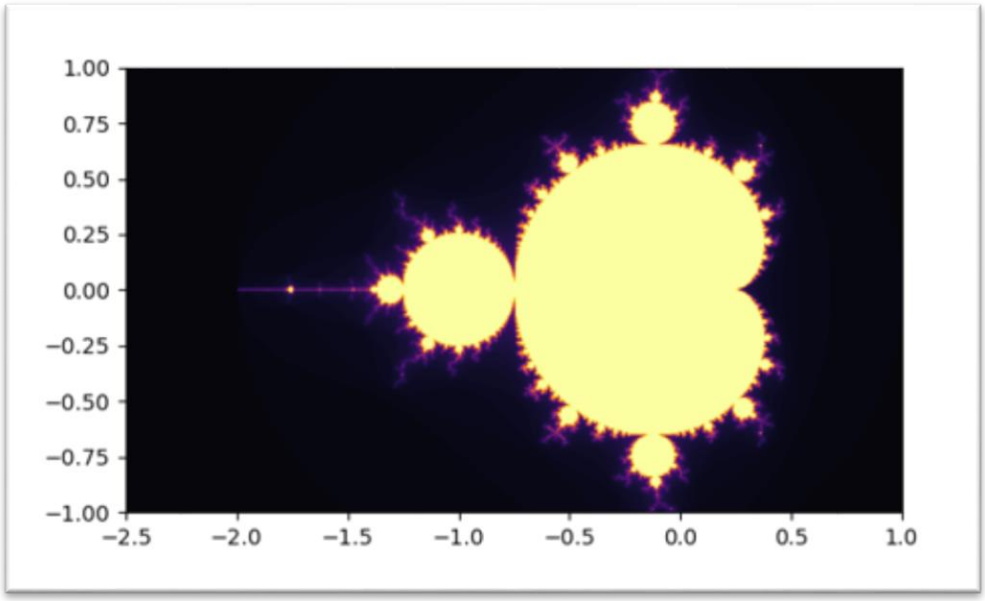

Рисунок 4.13 – Значення max_iter дорівнює 100

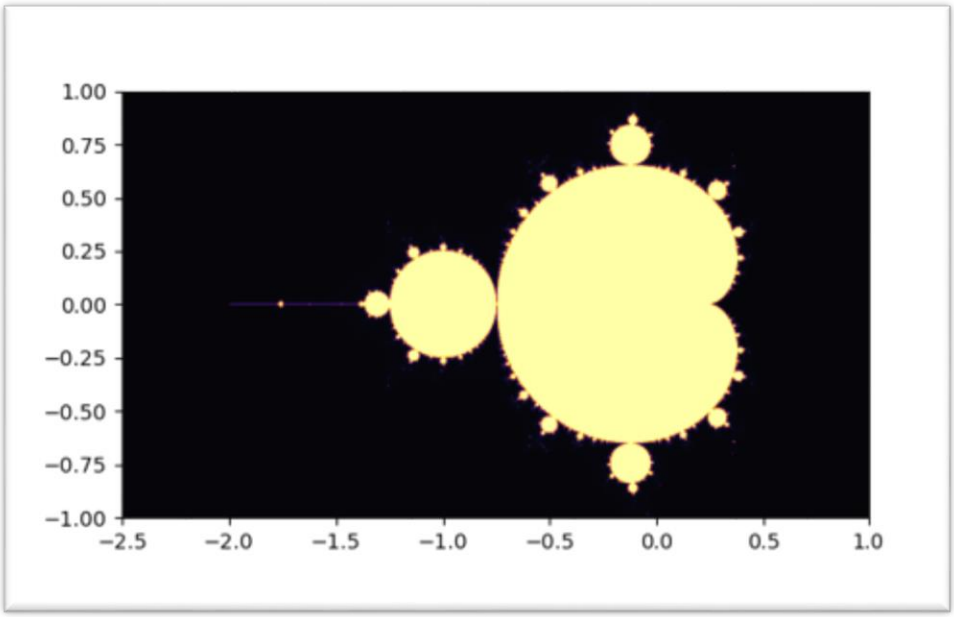

Рисунок 4.14 – Значення max_iter дорівнює 1000

Різниця на малюнках видна одразу. Значення max_iter впливає на кількість ітерацій, які виконуються для кожної точки при обчисленні множини Мандельброта. Це впливає на наступні аспекти:

– деталізація: збільшення max_iter призводить до більш детального зображення фракталу. Виходячи з певного значення, візуальні зміни можуть стати

непомітними для людського ока, але при використанні стеганографії це може допомогти зберегти більше інформації;

– час обчислення: збільшення max_iter збільшує час обчислення фрактала Мандельброта, оскільки потрібно виконати більше ітерацій для кожної точки. Це може стати проблемою, якщо потрібно швидко кодувати або декодувати повідомлення;

– стійкість до шуму: збільшення max_iter може збільшити стійкість кодованого повідомлення до шуму або втрати даних. Оскільки більше ітерацій дозволяє зберегти більше інформації, кодоване повідомлення може стати менш чутливим до помилок.

4.4 Кодування та декодування повідомлення

Функція encode_message, що показано на рисунку 4.15 приймає наступні параметри:

– message – текстове повідомлення, яке потрібно закодувати;

– img_size – розмір зображення фрактала (висота, ширина);

– max_iter – максимальна кількість ітерацій для визначення належності точки до множини Мандельброта;

– key – ключ, який використовується для генерації випадкової послідовності точок.

Спочатку функція перетворює повідомлення у бітову послідовність. Далі, вона створює порожній масив encoded_image розміром img_size. За допомогою заданого ключа генерується послідовність випадкових точок зображення.

Для кожної точки обчислюється значення фрактала Мандельброта, використовуючи функцію mandelbrot(c, max_iter). Якщо є ще біти повідомлення для кодування, значення фрактала коригується так, щоб його парність відповідала поточному біту повідомлення. Значення фрактала зберігається у масиві encoded_image.

Після обробки всіх точок, масив encoded_image містить закодоване повідомлення. Масив зберігається як файл .npy та зображення у форматі .png.

```python
def encode_message(message, img_size, max_iter, key):
    message_bits = ''.join(format(byte, '08b') for byte in message.encode('utf-8'))
    bit_index = 0
    encoded_image = np.zeros(img_size, dtype=np.int32)
    np.random.seed(key)
    points = [(y, x) for y in range(img_size[0]) for x in range(img_size[1])]
    np.random.shuffle(points)

    for y, x in points:
        c = complex(x / img_size[1] * 3.5 - 2.5, y / img_size[0] * 2 - 1)
        m = mandelbrot(c, max_iter)
        if bit_index < len(message_bits):
            if m % 2 != int(message_bits[bit_index]):
                m = m - m % 2 + int(message_bits[bit_index])
            bit_index += 1
        encoded_image[y, x] = m

    return encoded_image

message = "Hello world!"
print(f"Message length: {len(message)}")
img_size = (800, 1200)
max_iter = 2000
key = 42
```

Рисунок 4.15 – Функція encoded_message

Як було зазначено при відпрацюванні програми ми отримаємо зображення фракталу, що містить закодоване повідомлення та зображення у форматі .npy також з прихованим повідомленням, отримані файли можна побачити на рисунку 4.16.

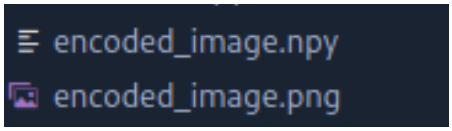

Рисунок 4.16 – Збережені файли, отримані при відпрацюванні функції

Важливо звернути увагу на те, що також застосовується при кодуванні додаткове перемішування точок з використанням np.random.shuffle(points) допомагає розподілити біти прихованого повідомлення випадковим чином по всьому зображенню. Це робить більш важким виявлення прихованого повідомлення, оскільки біти повідомлення не розташовані в послідовному порядку. Дане перемішування можна побачити у фрагменті коду на рисунку 4.17.

```
# Встановлюємо генератор випадкових чисел з заданим ключем
np.random.seed(key)
# Створюємо список точок зображення
points = [(y, x) for y in range(img_size[0]) for x in range(img_size[1])]
# Перемішуємо список точок з використанням випадкових чисел
np.random.shuffle(points)
```

Рисунок 4.17 – Перемішування точок у коді

Коли ми кодуємо повідомлення, ми замінюємо наприклад найменш значущий біт кожної точки зображення на біт з нашого повідомлення. Якщо б ми просто йшли по зображенню в ряд, починаючи з верхнього лівого кута і йдучи до нижнього правого, то це створило б візуальний шаблон, який може бути виявлений. Замість цього, ми перемішуємо точки перед тим, як розпочати кодування, щоб біти повідомлення були розподілені випадковим чином по всьому зображенню.

Ключ, який використовується для ініціалізації генератора випадкових чисел (np.random.seed(key)), потрібен для того, щоб забезпечити відтворюваність процесу перемішування, так що при декодуванні ми можемо відновити правильний порядок бітів.

Ми декодуємо повідомлення з файлу encoded_image.npy, а не з encoded_image.png, тому що формат .npy зберігає дані в більш точному вигляді, ніж формат зображення .png.

Файл .npy є бінарним форматом, призначеним для зберігання масивів NumPy. Він зберігає дані без втрати точності, оскільки він безпосередньо зберігає значення елементів масиву.

Формат зображень .png, з іншого боку, призначений для зображень і може стиснути дані для зменшення розміру файлу. В результаті може статися втрата точності даних або введення шуму в зображення через стиснення. Це може призвести до помилок при спробі декодувати повідомлення з зображення.

У нашому інтерфейсі для користувача наявна вкладка з відповідним функціоналом для конвертації файлу з формату npy у формат png, що зображено на рисунку 4.18.

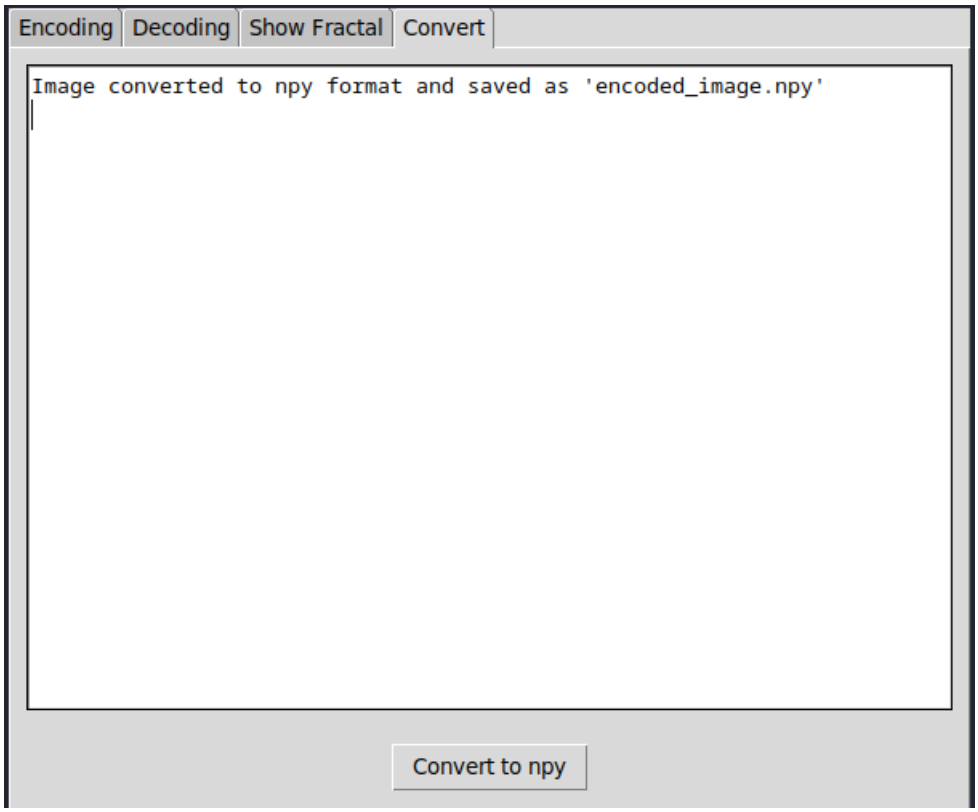

Рисунок 4.18 – Функціонал конвертації у інтерфейсі користувача

Але ще раз зауважу, що при декодуванні з файлу .png, можливі неточності у декодованому повідомленні через можливе стиснення та втрату точності в зображенні. Якщо результати декодування некоректні, спробуйте збільшити розмір зображення або зменшити рівень стиснення файлу .png.

Отже, для забезпечення точності декодування повідомлення, краще використовувати файл .npy, який зберігає дані без втрати точності, ніж файл .png, який може стиснути дані та втратити точність.

Сам процес декодування відбувається за допомогою наступної функції, що зображено на рисунку 4.19 та у додатку Г можна знайти код з коментарями: рисунок Г.1, рисунок Г.2, рисунок Г.3.

```python
import numpy as np

def decode_message(encoded_image, message_length, max_iter, key):
    message_bits = ''
    bit_index = 0
    np.random.seed(key)
    points = [(y, x) for y in range(encoded_image.shape[0]) for x in range(encoded_image.shape[1])]
    np.random.shuffle(points)

    for y, x in points:
        if bit_index < message_length * 8:
            m = encoded_image[y, x]
            message_bits += str(m % 2)
            bit_index += 1

    message = bytes(int(message_bits[i:i+8], 2) for i in range(0, len(message_bits), 8)).decode('utf-8', errors='ignore')
    return message[:message_length]

encoded_image = np.load("encoded_image.npy")
message_length = 13
max_iter = 1000
key = 42

decoded_message = decode_message(encoded_image, message_length, max_iter, key)
print(f"Decoded message: {decoded_message}")
```

Рисунок 4.19 – Функція decoded_message

Функція decode_message приймає наступні параметри:

– encoded_image – масив закодованого зображення, який потрібно декодувати;

– message_length – довжина повідомлення, яке потрібно декодувати;

– max_iter – максимальна кількість ітерацій для визначення належності точки до множини Мандельброта (не використовується у цьому випадку);

– key – ключ, який використовується для генерації випадкової послідовності точок. Сам процес розрахунку кількості точок можна побачити у вікні ще на процесі кодування, а вже для декодування цього значення потрібен ключ.

Спочатку функція ініціалізує порожню бітову стрічку message_bits. За допомогою заданого ключа генерується та відновлюється послідовність випадкових точок зображення.

Для кожної точки функція перевіряє парність значення фрактала Мандельброта, яке було збережено у масиві encoded_image. Парність додається до стрічки message_bits. Цей процес продовжується, доки не буде відновлено всі біти повідомлення.

Після того, як всі біти відновлено, функція конвертує бітову стрічку message_bits назад у текстове повідомлення, використовуючи кодування UTF-8.

Коли ж ми вже маємо усі потрібні параметри для декодування ми можемо скористатися програмою, та декодувати відповідно наше повідомлення, декодування повідомлення зображено на рисунку 4.20. У самій програмі ми передаємо параметри для декодування: закодоване зображення, довжину повідомлення та ключ.

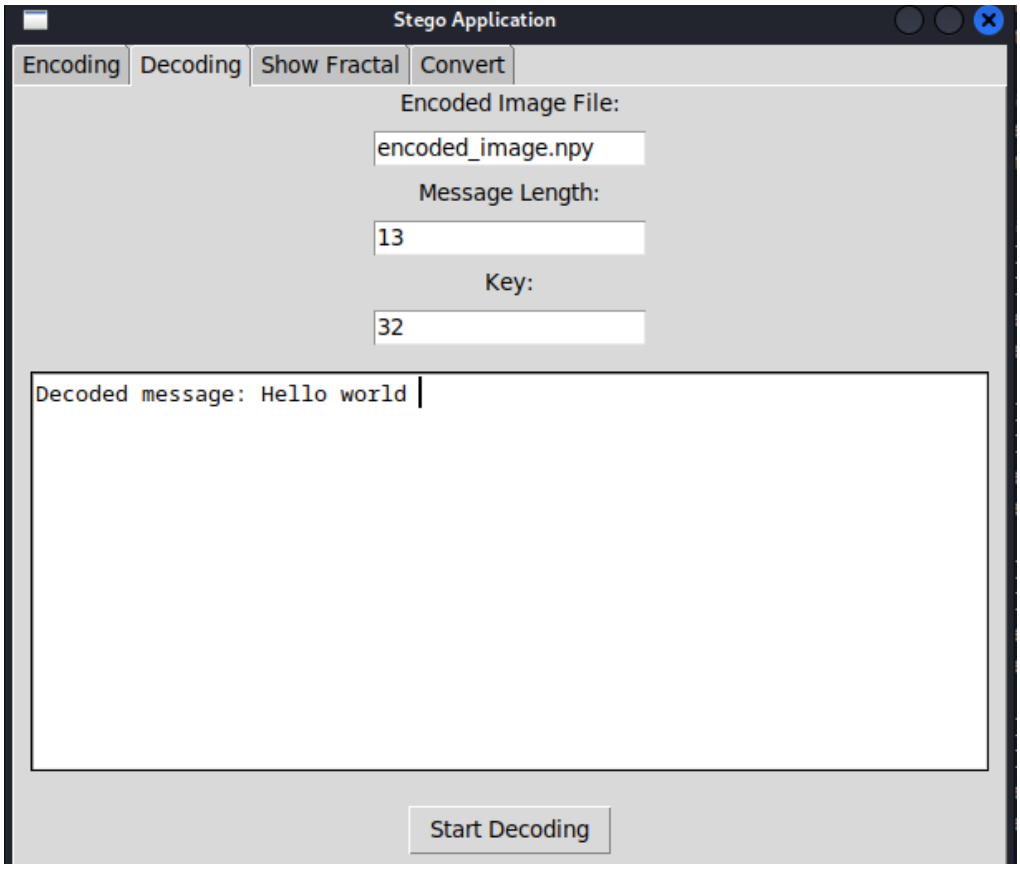

Рисунок 4.20 – Функціонал виведення повідомлення при декодуванні

Для додаткового забезпечення прихованості зображення для його передачі, можна використовувати різні додаткові методи, включаючи, наприклад, обмін послідовності каналів кольору, зміну розмірів зображення, застосування невеликих геометричних перетворень та інших такі як LSB.

Завершеною стратегією є комбінація кількох методів приховування та захисту даних. Наприклад, ми можемо застосувати стеганографію для вбудовування повідомлення в зображення, потім застосувати метод LSB для нашого зображення, а потім передати його через безпечний канал зв'язку. Це забезпечить декілька рівнів захисту для нашого секретного повідомлення.

4.5 Оцінка ефективності проаналізованого методу та його стегоаналіз

Метод, який ми проаналізували, використовує множину Мандельброта для стегографічного приховування текстових повідомлень в зображенні фракталу. Щоб оцінити ефективність цього методу, ми розглянемо такі критерії оцінки, як стійкість до атак, обсяг приховування даних та якість зображень.

Стійкість до атак. Стійкість до атак вимірюється тим, наскільки важко виявити приховане повідомлення або отримати доступ до нього. У цьому методі, повідомлення кодується в множині Мандельброта, змінюючи ітерації пікселів, які належать до множини. Це забезпечує деякий рівень стійкості, оскільки атакуючому буде важко знайти закономірності в структурі множини Мандельброта. Тим не менше, метод заснований на використанні ключа для генерування послідовності точок, які використовуються для кодування та декодування повідомлення. Якщо атакуючий знає алгоритм та ключ, він може отримати доступ до прихованого повідомлення.

Обсяг приховування даних. Обсяг приховування даних вимірюється кількістю інформації, яку можна вбудувати в зображення без погіршення якості зображення. У цьому методі, обсяг приховування даних обмежений кількістю точок множини Мандельброта. Збільшення кількості ітерацій (max_iter) може збільшити обсяг приховування даних, але також може знизити якість зображення.

Якість зображень. Якість зображень вимірюється візуальною якістю отриманих зображень, порівняно з оригінальним зображенням фракталу. У цьому методі, якість зображень залежить від параметра max_iter. Збільшення кількості ітерацій може збільшити деталізацію зображення фракталу, але також може призвести до збільшення шуму, що може знизити візуальну якість зображення. Зменшення кількості ітерацій може знизити деталізацію зображення фракталу, але забезпечити менше візуальних спотворень.

Давайте розглянемо такі параметри як мета дані для нашого отриманого зображення. Для цього використаємо вже наведену вище команду «exiv2 pr encoded_image.png», де параметр «pr» – print показує нам метадані заданого зображення encoded_image.png, що показано на рисунку 4.21.

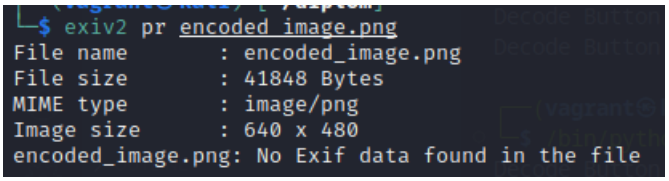

Рисунок 4.21 – Результат команди exiv2 для encoded_image.png

За допомогою команди «exiv2 pr encoded_image.png», ми змогли отримати інформацію про файл encoded_image.png.

Виведена інформація включає в себе:

– розмір файлу: 41848 байтів;

– MIME-тип: image/png;

– розміри зображення: 640x480 пікселів.

Також варто зазначити, що в файлі не виявлено Exif-даних. Exif (Exchangeable Image File Format) – це стандарт метаданих для зображень, який використовується для зберігання додаткової інформації про зображення, такої як параметри камери, GPS-координати та інше. Відсутність Exif-даних у файлі encoded_image.png означає, що додаткова інформація про зображення відсутня або не підтримується форматом PNG.

З результатів виконання команди «exiv2 pr encoded_image.png», можна зробити наступні висновки.

1) Зображення має розміри 640x480 пікселів, що може бути досить великим для деяких застосувань, але загалом він вважається досить компактним.

2) Зображення має розмір файлу 41848 байтів, що свідчить про те, що воно займає невеликий обсяг пам'яті. Це може бути важливим, якщо вам потрібно передавати зображення через мережу або зберігати його на обмеженому просторі для зберігання.

3) Файл зображення має формат PNG, який підтримує без втрат стиснення. Це означає, що якість зображення не постраждала від стиснення, і воно може зберігати достатньо деталей для декодування повідомлення.

Також слід звернути уваги на час виконання кодування при зміні параметрів при кодуванні.

Знайти розрахунку можна у таблиці 4.1, де ми будемо змінювати параметр max_iter, та дивитися як це впливає на час кодування.

Таблиця 4.1 – Час на виконання при зміні параметру max_iter

Час виконання	Max_iter
3,73 сек.	100 ітерацій
7,33 сек.	300 ітерацій
13,63 сек.	600 ітерацій
19,2 сек.	1000 ітерацій

Зміна параметру max_iter впливає і на час декодування. У таблиці 4.2 зображено як саме цей влив відбувається і який час використовується.

Таблиця 4.2 – Час на декодування при зміні параметру max_iter

Час декодування	Max_iter
1 сек.	100 ітерацій
1,21 сек.	300 ітерацій
1,21 сек.	600 ітерацій
1,23 сек.	1000 ітерацій

Бачимо, що процес декодування при різному значенні параметру max_iter ніяк не впливає на час, та є достатньо незмінним.

Повний стегоаніліз для даного методу можна побачити у додатку Д на рисунку Д.1, рисунку Д.2, рисунку Д.3, рисунку Д.4, рисунку Д.5.

Слід звернути увагу на те що, коли отримаємо список точок зображення ми додатково перемішуємо точки за допомогою функції мови програмування, а саме random shuffle, що видно на рисунку 4.22.

```
# Створюємо список точок зображення
points = [(y, x) for y in range(img_size[0]) for x in range(img_size[1])]
# Перемішуємо список точок з використанням випадкових чисел
np.random.shuffle(points)
```

Рисунок 4.22 – Перемішування точок за допомогою функції бібліотеки python

Але використання випадкових чисел за допомогою звичайної функції мови програмування може бути ненадійним, тому застосуємо лінійний конгруентний

генератор (змішаний) для перемішування точок, який є широко використовуваним методом для псевдовипадкових чисел. Формула для лінійного конгруентного генератора [13]:

$$x_i = (a \cdot x_i - 1 + c) \cdot (\mathrm{mod} M), \quad (4.2)$$

де a, c – константи, які використовуються;

M – модуль, який використовується для обмеження виводу.

На мові програмування Python генерація буде виглядати наступним чином, рисунок 4.23.

```
import matplotlib.pyplot as plt

# Лінійний конгруентний генератор (змішаний)
# xi = (a*xi-1 + c) mod M.

N = 1000
a = 7
c = 7
m = 55
x = [20]

for i in range(N):
    x.append((a*x[i] + c) % m)

plt.plot(range(N+1), x, 'ks')
plt.axis([0, 100, 0, 60])
plt.show()
```

Рисунок 4.23 – Лінійно конгруентний генератор

Ми використовуємо наступні параметри:

– N = 1000 : задаємо кількість чисел для генерації;

– a = 7, c = 7, m = 55, x(1) = 20: встановлюємо параметри для лінійного конгруентного генератора. «a, c» – константи, m – модуль, який використовується для обмеження виводу, x(1) – встановлюємо початкове значення для генерації та axis[0, 100, 0, 60] – встановлюємо область графіку. Отримані згенеровані точки можна побачити на рисунку 4.24.

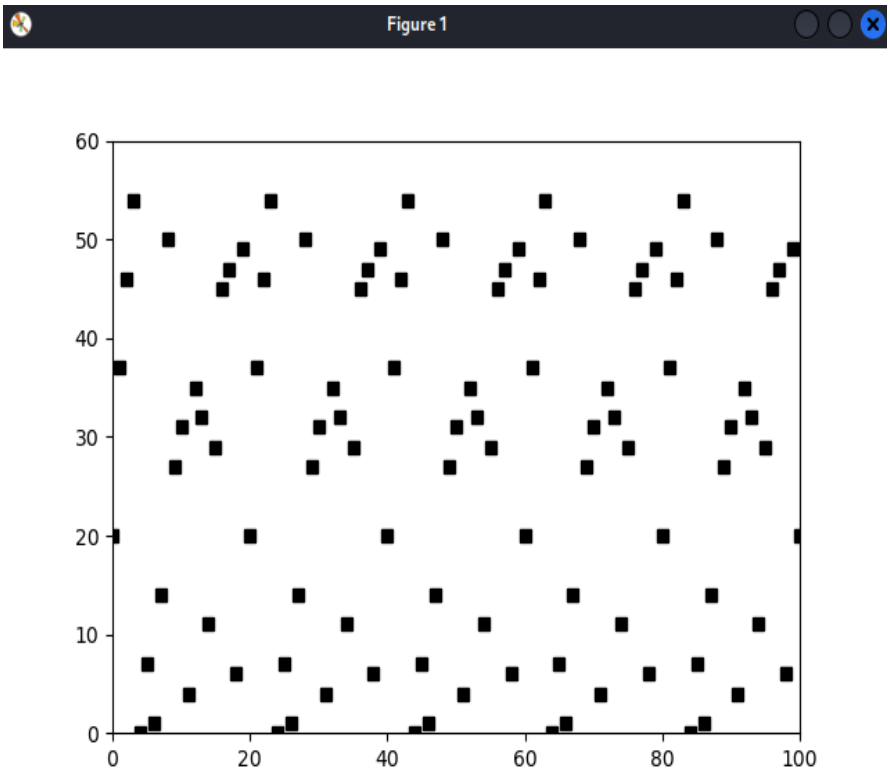

Рисунок 4.24 – Згенеровані точки за допомогою лінійного конгруентного генератора

Та на основі отриманих точок можемо побудувати гістограму послідовності, що зображено на рисунку 4.25, де побачимо наскільки великий період повторення та рівномірність розподілу точок.

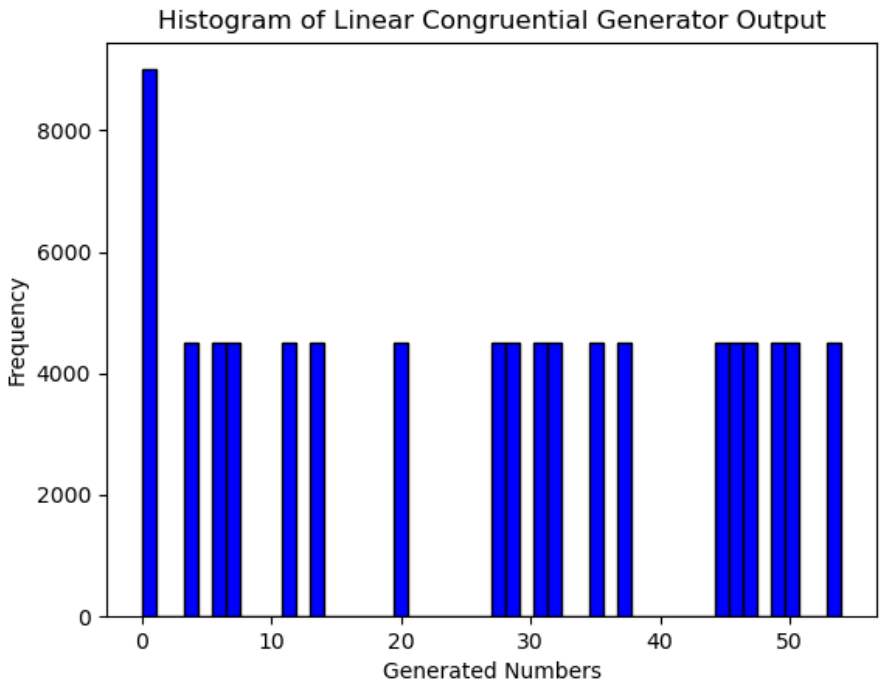

Рисунок 4.25 – Гістограма періоду повторення та рівномірності розподілу точок

Даний метод можемо використати і у нашому коді для наших точок наступним чином, рисунок 4.26. Перемішуємо точки за допомогою лінійного конгруентного методу для додаткової безпеки.

Перемішування точок фракталу може змінити його візуальні характеристики та дати нам можливість створювати різноманітні зображення. По суті, фрактал – це геометрична структура, яка має властивість самоподібності, тобто її частини мають структуру, подібну до цілої структури.

```python
def encode_message(message, img_size, max_iter, key):
    message_bits = ''.join(format(byte, '08b') for byte in message.encode('utf-8'))
    bit_index = 0
    encoded_image = np.zeros(img_size, dtype=np.int32)

    N = img_size[0] * img_size[1]  # Number of points
    a = 7
    c = 7
    m = 55
    x = [key]  # Initialize with the key

    for i in range(N):
        x.append((a*x[i] + c) % m)

    shuffle_mapping = sorted(range(N), key=lambda i: x[i])

    points = [(y, x) for y in range(img_size[0]) for x in range(img_size[1])]

    points = [points[i] for i in shuffle_mapping]

    mandelbrot_points = 0

    for y, x in points:
        c = complex(x / img_size[1] * 3.5 - 2.5, y / img_size[0] * 2 - 1)
        m = mandelbrot(c, max_iter)
        if m == max_iter:
            mandelbrot_points += 1
        if bit_index < len(message_bits):
            if m % 2 != int(message_bits[bit_index]):
                m = m - m % 2 + int(message_bits[bit_index])
            bit_index += 1
        encoded_image[y, x] = m

    return encoded_image, mandelbrot_pointsS
```

Рисунок 4.25 – Перемішування точок за допомогою лінійного конгруентного методу у коді

З гістограми на рисунку 4.25 робимо висновок, що даний генератор має не великий період повторення, тобто не забезпечує належного рівномірного розподілення, тому слід використовувати генератори цього ж типу, але з більшим

періодом повторення, наприклад мінімальний стандартний генератор випадкових чисел С. Парка та К. Міллера [15].

Різниця зі звичайним лінійним конгруентним генератором та генератор випадкових чисел С. Парка та К. Міллера у тому, що він не має додаткового зміщення і має наступні параметри:

– a = 16807;
– m = 2147483647;
– c = 0.

Його реалізацію у виді коду можна побачити на рисунку 4.26 та гістограму для даного генератора на рисунку 4.27 відповідно.

```python
import matplotlib.pyplot as plt
import numpy as np

N = 3*3000
a = 16807
m = 2147483647
x = [20]

for i in range(N):
    x.append((a*x[i]) % m)

plt.figure(1)
plt.plot(range(N+1), x, 'ks')
plt.axis([0, 100, 0, 60])

plt.figure(2)
plt.hist(x, bins='auto', color='g', alpha=0.7) # 'auto' дозволяє автоматично вибрати кількість бінів
plt.show()
```

Рисунок 4.26 – Генератор С. Парка та К .Міллера у виді коду

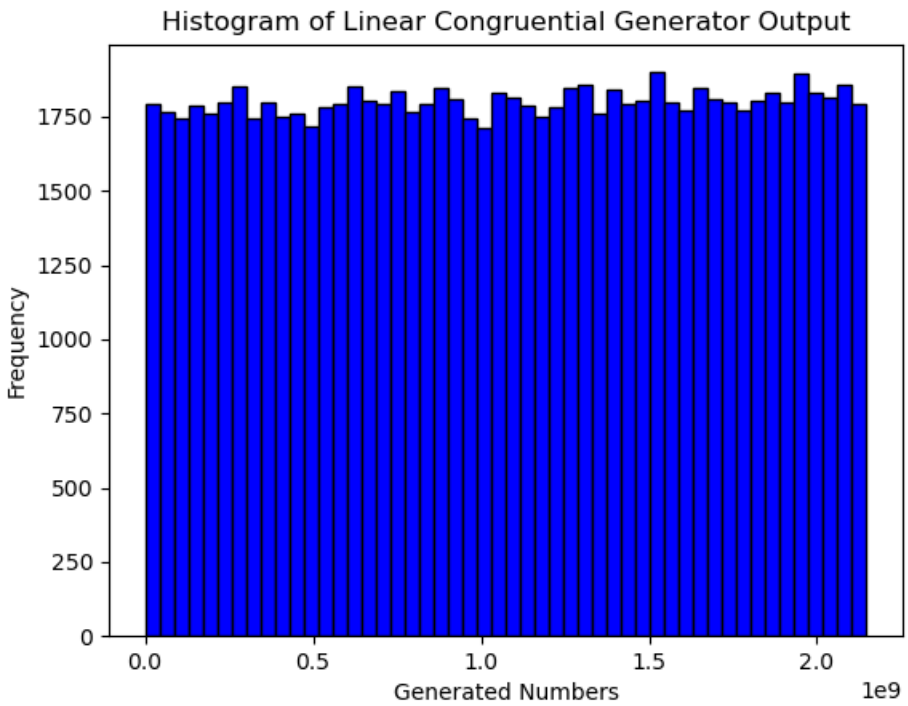

Рисунок 4.27 – Гістограма періоду повторення та рівномірності розподілу точок

З гістограми видно, що ми отримали більший період повторення, це забезпечить добре рівномірне розподілення.

Можна також провести статистичний тест послідовностей, що ми отримуємо під час перемішування. Це можливо зробити за допомогою відомого стандарту NIST. Набір тестів NIST – це статистичний пакет, що складається з 15 тестів, розроблених для перевірки випадковість (довільної довжини) двійкових послідовностей, створених апаратним або програмним забезпеченням криптографічні генератори випадкових чи псевдовипадкових чисел. Ці тести зосереджені на різноманітних типи не випадковості, які можуть існувати в послідовності [14]. Деякі тести можна розкласти на безліч субтестів.

Наявні тести:

– частотний (монобітний) тест;
– перевірка частоти в блоці;
– тест Runs;
– тести для найдовших одиниць у блоці;
– ранговий тест двійкової матриці;

- дискретне перетворення Фур'є (спектральний) тест;
- тест на відповідність шаблону, що не перекривається;
- тест на відповідність шаблонів, що перекриваються;
- «Універсальний статистичний» тест Маурера;
- тест лінійної складності;
- серійний тест;
- приблизний тест на ентропію;
- тест кумулятивних сум;
- тест на випадкові екскурсії;
- варіант тесту «Випадкові екскурсії».

Отже, проведемо деякі з тестів над нашою послідовністю.

Частотний тест.

Тест фокусується на співвідношенні нулів і одиниць у всій послідовності. Мета цього тесту полягає в тому, щоб визначити, чи кількість одиниць і нулів у послідовності приблизно однакова можна очікувати для справді випадкової послідовності. Тест оцінює близькість частки одиниць до 0.5, тобто кількість одиниць і нулів у послідовності має бути приблизно однаковою. Усі наступні тести залежать від проходження цього тесту, тому при усіх підрахунках слід бути дуже уважним та не допускати помилок, оскільки значення будуть використовуватися у формулах інших тестів.

Наприклад візьмемо, що ε – послідовність бітів, що було згенеровано за допомогою лінійного конгруентного генератора, а саме ми візьмемо перші 10 перемішених точок, дорівнює 1011010101. Для цього випадку n = 10, n – довжина бітового рядка. Також нам знадобиться S_{obs} – абсолютне значення суми x_i , поділеній на на квадратний корінь із довжини послідовності.

Для нашого заданого ε ми будемо обчислювати S_n наступним чином за формулою:

$$S_n = 1 + (-1) + 1 + 1 + (-1) + 1 + (-1) + 1 + (-1) + 1 = 2. \qquad (4.3)$$

Обчислюємо також тестову статистику S_{obs} за формулою:

$$S_{obs} = \frac{|S_n|}{\sqrt{n}}. \quad (4.4)$$

Підставляємо наші значення у формулу (4.4) та отримуємо наступне:

$$S_{obs} = \frac{|2|}{\sqrt{10}} = 0.632455532. \quad (4.5)$$

Та вже знаючи тестову статистику S_{obs} з формули (4.4) можемо розрахувати P-value, де erfc вже відома нам функція, формула:

$$P-value = erfc\left(\frac{S_{obs}}{\sqrt{2}}\right), \quad (4.6)$$

де erfc – додаткова функція помилки.

У нашому випадку підставляючи значення у формулу (4.6), отримаємо:

$$P-value = erfc\left(\frac{0.632455532}{\sqrt{2}}\right) = 0.527089. \quad (4.7)$$

Якщо обчислене P-значення менше за 0.01, то можна зробити висновок, що послідовність невипадкова. В іншому випадку можемо зробити висновок, що послідовність є випадковою. Маємо те, що 0.527809 > 0.01, отже послідовність випадкова.

Тест на найдовший запуск блоку.

Тест зосереджується на найдовшій серії одиниць у M-бітних блоках. Метою цього тесту є визначити, чи відповідає довжина найдовшого ряду одиниць у перевіреній послідовності довжина найдовшого циклу з тих, які можна було б очікувати у випадковій послідовності. Зауважу, що порушення в очікуваній тривалості найдовшого ряду одиниць означає, що в очікуваному також є

нерегулярність довжини найдовшого ряду нулів. Тому потрібен лише тест для одиниць. Проведемо тест за наступними кроками.

1) Розділяємо послідовність на M-розрядні блоки.

2) Заносимо у таблицю частоти v_i найдовших циклів одиниць у кожному блоці за категоріями, де кожна клітинка містить кількість прогонів одиниць заданої довжини. Для значень M, маємо наступну таблицю 4.3 з показниками.

Таблиця 4.3 – Показники при різних M-розрядних блоках

v_i	M = 8	M = 128	M = 10^4
v_0	≤ 1	≤ 4	≤ 10
v_1	2	5	11
v_2	3	6	12
v_3	≥ 4	7	13
v_4		8	14
v_5		≥ 9	15
v_6			≥ 16

Тепер розрахуємо X_{obs} – найбільш очікувана довжина в межах M-бітових блоків.

Використаємо наступну формулу для цих розрахунків:

$$x^2(obs) = \sum_{i=0}^{K} \frac{(v_i - N \cdot \pi_i)^2}{N \cdot \pi_i}, \tag{4.8}$$

де π_i – теоретична ймовірність;

значення K та N визначаються значенням M згідно таблиці 4.4.

Таблиця 4.4 – Значення K та N при відповідному M

M	K	N
8	3	16
128	5	49
10^4	6	75

Підставивши значення у формулу (4.8), отримаємо наступне:

$$x^2(\text{obs}) = \frac{(4 - 16 \cdot 0.2148)^2}{16 \cdot 0.2148} + \frac{(9 - 16 \cdot 0.3672)^2}{16 \cdot 0.3672} + \frac{(3 - 16 \cdot 0.2305)^2}{16 \cdot 0.2305} + \quad (4.9)$$
$$+ \frac{(0 - 16 \cdot 0.1875)^2}{16 \cdot 0.1875} = 4.882605.$$

Та також розрахуємо P-value за формулою:

$$\text{P-value} = \text{igamc}\left(\frac{K}{2}, \frac{x^2(\text{obs})}{2}\right). \quad (4.10)$$

Підставляємо наші значення у формулу (4.10) та отримаємо:

$$\text{P-value} = \text{igamc}\left(\frac{3}{2}, \frac{4.882605}{2}\right) = 0.180598.$$

Якщо обчислене P-значення менше за 0.01, то можна зробити висновок, що послідовність невипадкова. В іншому випадку робимо висновок, що послідовність є випадковою.

Оскільки, отримане нами P-value більше 0.01 (P-value дорівнює 0.180609), висновок полягає в тому, що послідовність випадкова. Зауважу, що великі значення $x^2(\text{obs})$ вказують на те, що перевірена послідовність містить кластери одиниць.

Серійний тест.

Цей тест зосереджується на частоті всіх можливих m-бітових шаблонів, що перекриваються в усій послідовності. Мета цього тесту полягає в тому, щоб визначити, чи кількість входжень 2m, m-біт шаблонів, що перекриваються, приблизно такі ж, як і очікувалося б для випадкової послідовності. Випадкові послідовності мають рівномірність; тобто кожен m-бітовий шаблон має такий же шанс появи, як і будь-який інший m-бітовий шаблон. Зауважте, що для m = 1 серійний тест еквівалентний частотному тесту з першого нашого тесту. Отже перейдемо до нашого тесту на серійність.

1) Формуємо розширену послідовність ε′: розширити послідовність, додавши перші m-1 бітів до кінця послідовності для різних значень n. Наприклад, задано n = 10 і ε = 0011011101. Якщо m = 3, то ε′ = 001101110100. Якщо m = 2, тоді ε′ = 00110111010. Якщо m = 1, то ε′ = вихідна послідовність 0011011101.

2) Визначаємо частоту всіх можливих m-бітових блоків, що перекриваються, усі можливі перекриття (m-1) – бітові блоки та всі можливі (m-2) – бітові блоки, що перекриваються. Нехай v_{i_1} позначає частоту m-біт в $i_1 \dots i_m$ бітовий шаблон; нехай $v_{i_1 \dots i_m}$ позначає частоту (m-1) – бітового шаблону $v_{i_1 \dots i_{m-1}}$; і нехай $v_{i \ \dots i_{m-2}}$позначає частоту (m-2) – бітового шаблону $i_1 \dots i_{m-2}$. Для прикладу в цьому розділі, коли m = 3, тоді (m-1) = 2 і (m-2) = 1. Частота всіх 3-розрядні блоки: $v_{000} = 0$, $v_{001} = 1$, $v_{010} = 1$, $v_{011} = 2$, $v_{100} = 1$, $v_{101} = 2$, $v_{110} = 2$, $v_{111} = 0$. Частота всіх можливих (m-1) – бітових блоків: $v_{00} = 1$, $v_{01} = 3$, $v_{10} = 3$, $v_{11} = 3$. Частота всіх (m-2) – бітових блоків: $v_0 = 4$, $v_1 = 6$.

3) Проводимо розрахунок за наступними формулами:

$$\psi_m^2 = \frac{2^m}{n} \cdot \sum_{i_1 \dots i_m} (v_{i_1 \dots i_m} - \frac{n}{m^2})^2 = \frac{2^m}{n} \cdot \sum_{i_1 \dots i_m} v_{i_1 \dots i_m}^2 - n, \qquad (4.11)$$

$$\psi_{m-1}^2 = \frac{2^{m-1}}{n} \cdot \sum_{i_1 \dots i_{m-1}} (v_{i_1 \dots i_{m-1}} - \frac{n}{2^{m-1}})^2 = \frac{2^{m-1}}{n} \cdot \sum_{i_1 \dots i_{m-1}} v_{i_1 \dots i_{m-1}}^2 - n, \qquad (4.12)$$

$$\psi_{m-2}^2 = \frac{2^{m-2}}{n} \cdot \sum_{i_1 \dots i_{m-2}} (v_{i_1 \dots i_{m-2}} - \frac{n}{2^{m-2}})^2 = \frac{2^{m-2}}{n} \cdot \sum_{i_1 \dots i_{m-2}} v_{i_1 \dots i_{m-2}}^2 - n. \qquad (4.13)$$

4) Далі розраховуємо, використовуючи формули:

$$\nabla\psi_m^2 = \psi_m^2 - \psi_{m-1}^2, \tag{4.14}$$

$$\nabla^2\psi_m^2 = \psi_m^2 - 2\psi_{m-1}^2 + \psi_{m-2}^2. \tag{4.15}$$

Та отримаємо результат розрахунків підставивши значення у формулу (4.14) та формулу (4.15):

$$\nabla\psi_m^2 = \psi_m^2 - \psi_{m-1}^2 = \psi_3^2 - \psi_2^2 = 2.8 - 1.2 = 1.6,$$

$$\nabla^2\psi_m^2 = \psi_m^2 - 2\psi_{m-1}^2 + \psi_{m-2}^2 = \psi_3^2 - 2\psi_2^2 + \psi_1^2 = 2.8 - 2(1.2) + 0.4 = 0.8.$$

5) Розраховуємо P-value для обох випадків:

$$\text{P-value1} = \text{igamc}\left(2^{m-2}, \nabla\psi_m^2\right) \text{ and } \text{P-value2} = \text{igamc}\left(2^{m-3}, \nabla^2\psi_m^2\right). \tag{4.16}$$

Маємо формули для отримання P-value1 та P-value2, отже проведемо розрахунок за формулою (4.17):

$$\text{P-value1} = \text{igamc}\left(2, \frac{1.6}{2}\right) = 0.9057,$$

$$\text{P-value2} = \text{igamc}\left(1, \frac{0.8}{2}\right) = 0.8805.$$

Оскільки P-value, отримане вище, становить більше 0.01 (P-value1 дорівнює 0.9057 і P-value2 дорівнює 0.8805), висновок полягає в тому, що послідовність є випадковою.

ВИСНОВКИ

У ході магістерської роботи було розглянуто та проаналізовано метод стеганографії на основі фракталів Мандельброта. Метод полягає у кодуванні текстових повідомлень у фрактальних зображеннях, а також у декодуванні цих повідомлень зі зображень.

За допомогою реалізованого коду було продемонстровано, що метод є працездатним і забезпечує можливість кодування та декодування повідомлень з фрактальних зображень. Розглянуто параметри, які впливають на вигляд фрактала, такі як кількість ітерацій (max_iter).

За результатами аналізу ефективності методу, було встановлено, що стеганографічні системи, що використовують фрактал Мандельброта, мають декілька переваг.

1) Відсутність стеганографічних артефактів на зображеннях, що ускладнює виявлення прихованих повідомлень.

2) Висока стійкість зображень до стиснення та редагування, завдяки використанню формату PNG.

3) Простота реалізації методу та можливість додаткової оптимізації параметрів фрактала для підвищення стійкості до атак.

Проте, метод також має певні обмеження.

1) Обмежена кількість даних, які можуть бути закодовані в зображеннях фіксованого розміру.

2) Необхідність зберігання зображень у форматі, який підтримує без втрат стиснення, що може збільшити розмір файлу.

Загалом, результати магістерської роботи показують, що стеганографічний метод на основі фракталів Мандельброта є перспективним для застосування у різних галузях, включаючи захист інформації, передачу конфіденційних даних та безпеку комунікацій. У майбутньому можна розглянути дослідження додаткових параметрів фракталів, що можуть впливати на стеганографічні характеристики, та розробити алгоритми адаптивного кодування, які залежать від структури фрактала. Також можна

провести дослідження стійкості розробленого методу до стеганографічних атак та розробити алгоритми виявлення прихованої інформації у фрактальних зображеннях.

Важливим напрямком подальшого дослідження є оптимізація обчислювальної складності алгоритмів кодування та декодування, а також розробка методів забезпечення високої стійкості до атак на стеганографічні системи. Наприклад, можна розробити гібридні методи, що використовують криптографічні алгоритми та стеганографічні техніки для підвищення безпеки передачі даних.

Також важливо розглянути можливість інтеграції стеганографічного методу на основі фракталів Мандельброта з іншими методами обробки зображень та мультимедійних даних, щоб розширити сфери застосування та досягти більшої гнучкості рішень.

У результаті, проведена магістерська робота дозволила розглянути та проаналізувати стеганографічний метод на основі фракталів Мандельброта, показати його переваги та обмеження, а також визначити напрямки подальших досліджень для підвищення ефективності та безпеки даного методу.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Дослідження стегосистем. URL: https://cyberleninka.ua/article/n/analiz-steganografichnih-metodiv-prihovuvannya-informatsiynih-potokiv-u-konteyneri-riznih-formativ/viewer. (дата звернення: 04.04.2023).
2. Naveen K. Nishchal. Digital techniques of data and image encryption: Digital techniques of data and image encryption. *IOP Publishing Ltd*. 2019. P.8 – 10.
3. Tkachov K.V. Ivestigation of methods for obtaining the information in stegosystems. *International Journal of Applied Engineering*. 2023.
4. Dr. Ekta Walia, Payal Jain, Navdeep. An Analysis of LSB & DCT based Steganography: An Analysis of LSB & DCT based Steganography. *Global Journal of Computer science & technology*. 2010. Vol. 10, No 1. P. 5 – 8.
5. Ajit Danti, Preethi Acharya. Randomized Embedding Scheme Based on DCT: «Randomized Embedding Scheme Based on DCT Coefficients for Image Steganography». *IJCA Special Issue on «Recent Trends in Image Processing and Pattern Recognition» RTIPPR*. India, 2010. P. 97 – 103.
6. Po-Yueh Chen and Hung-Ju Lin. A DWT Based Approach for Image Steganography. *International Journal of Applied Science and Engineering*. 2006. No 1. P. 275 – 290.
7. Information hiding with adaptive steganography. URL: https://www.sciencedirect.com/science/article/pii/S2214914718300965#sec3 (дата звернення: 04.04.2023).
8. Weinstein C.J. Roundoff noise in floating point fast Fourier transform computation. *IEEE Trans. Audio Elekroacoust*. 1969. No 2. P. 209 – 211.
9. Кошкіна Н.В. Стеганоаналіз на основі атаки контрольного введення. *Журнал «Проблемы управления и информатики»*. 2014. №6. С. 137–144.
10. Сергієнко І.В., Задирака В.К., Бабич М.Д., Березовський А.І., Бесараб П.Н., Людвиченко В.А. «Компьютерные технологии решения задач прикладной и вычислительной математики с заданными значениями характеристик качества». 2006. №5. С. 33-41.

11. Множина Мандельброта. URL: https://uk.wikipedia.org/wiki/Множина_Мандельброта (дата звернення: 04.04.2023).

12. Ткачов К.В. Методи приховування інформації у стегосистемах. *Актуальні питання розвитку галузей науки:* матеріали міжнар. наук.-практ. конф., м. Чернігів, 12 трав. 2023. Чернігів, 2023. С. 130–131.

13. Лінійний конгруентний метод. URL: https://uk.wikipedia.org/wiki/Лінійний_конгруентний_метод# (дата звернення: 04.04.2023).

14. A static test suite for random and pseudorandom number generators for cryptographic applications. URL: https://nvlpubs.nist.gov/nistpubs/legacy/sp/nistspecialpublication800-22r1a.pdf (дата звернення: 20.04.2023).

15. Random Number Generators: Good ones are hard to find. URL: https://www.firstpr.com.au/dsp/rand31/p1192-park.pdf (дата звернення: 22.04.2023).