Міністерство освіти і науки України
Державний університет «Одеська політехніка»
Інститут інформаційної безпеки, радіоелектроніки та телекомунікацій
Кафедра кібербезпеки та програмного забезпечення

Надвоцький Олександр Юрійович,
студент групи РЗ-161

КВАЛІФІКАЦІЙНА РОБОТА МАГІСТРА

Удосконалення методу кількісної оцінки
захищеності стеганоповідомлення

Спеціальність:
125 Кібербезпека

Керівник:
Кобозєва Алла Анатоліївна,
д.т.н., проф.

Одеса – 2021

Міністерство освіти і науки України

Державний університет «Одеська політехніка»

Інститут інформаційної безпеки, радіоелектроніки та телекомунікацій

Кафедра кібербезпеки та програмного забезпечення

Рівень вищої освіти другий (магістерський)

Спеціальність 125 – Кібербезпека

Освітня програма – Кібербезпека

ЗАТВЕРДЖУЮ

Завідувач кафедри КБПЗ

_____

д.т.н.,проф. А.А.Кобозєва

«____» _____ 2021р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ

Надвоцькому Олександру Юрійовичу

1.Тема роботи: *Удосконалення методу кількісної оцінки захищеності стеганоповідомлення*.

керівник роботи *Кобозєва Алла Анатоліївна, д.т.н., проф.*, затверджені наказом ректора університету від «25» жовтня 2021р. № 372-в .

2.Зміст роботи*: дослідження проблеми кількісної оцінки захищеності стеганоповідомлення від атак проти вбудованого повідомлення, теоретичні основи удосконалення методу кількісної оцінки захищеності стеганоповідомлення, практична реалізація удосконаленого методу кількісної оцінки захищеності стеганоповідомлення.*

3. Перелік ілюстративного матеріалу: *слайди презентації*.

4. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		Завдання видав	Завдання прийняв

5. Дата видачі завдання «_____» _____ 20___ р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання	Примітка
1	*Дослідження існуючого методу кількісної оцінки захищеності стеганоповідомлення*	*01.09.2021*	*виконано*
2	*Удосконалення методу кількісної оцінки захищеності стеганоповідомлення. Модифікація оцінки відхилення власних векторів, впровадження нової умови вибору захищених власних векторів*	*20.09.2021*	*виконано*
3	*Реалізація програмного забезпечення для проведення обчислювального експерименту*	*05.10.2021*	*виконано*
4	*Розробка програмного продукту алгоритмічної реалізації методу кількісної оцінки захищеності стеганоповідомлення*	*12.10.2021*	*виконано*
5	*Тестування програмного продукту*	*20.10.2021*	*виконано*
6	*Підготовка тексту роботи*	*01.11.2021*	*виконано*
7	*Підготовка презентації та доповіді*	*14.11.2021*	*виконано*
8	*Попередній захист*	*26.11.2021*	*виконано*
9	*Нормоконтроль, рецензування*	*15.12.2021*	*виконано*
10	*Занесення роботи в електронний архів*	*18.12.2021*	*виконано*
11	*Допуск до захисту у зав. кафедри*	*19.12.2021*	*виконано*

Здобувач вищої освіти _____ *Надвоцький О.Ю.*

Керівник роботи _____ *Кобозєва А.А.*

ЗАВДАННЯ

на розробку розділу «Охорона праці»

Надвоцького Олександра Юрійовича

Інститут інформаційної безпеки, радіоелектроніки та телекомунікацій

Кафедра кібербезпеки та програмного забезпечення

Тема роботи: *Удосконалення методу кількісної оцінки захищеності стеганоповідомлення*

Зміст розділу:

1. Аналіз умов праці і вибір заходів і засобів захисту від небезпечних і шкідливих виробничих факторів.
2. Аналіз техногенних небезпек і вибір заходів і засобів забезпечення безпеки у надзвичайних ситуаціях.
3. Розрахунок величини опору штучного захисного заземлення.

Керівник роботи
д.т.н., проф. Кобозєва А.А.

_____
(підпис)

«____» _____ 2021 р.

Консультант з охорони праці та БНС
к.т.н., доцент Ярова І.А.

_____
(підпис)

«____» _____ 2021 р.

АНОТАЦІЯ

Кваліфікаційна робота на тему «Удосконалення методу кількісної оцінки захищеності стеганоповідомлення» на здобуття другого (магістерського) рівня вищої освіти за спеціальністю 125 – Кібербезпека, містить: 22 рисунки, 1 додаток, 33 літературних джерела за переліком посилань. Робота виконана на 93 сторінках загального тексту і 70 сторінках основного тексту.

Метою даної роботи є удосконалення методу кількісної оцінки захищеності стеганоповідомлення для забезпечення можливості вибору цифрового зображення-контейнера, що породжує стеганоповідомлення, нечутливе до збурних дій.

Теоретичними основами забезпечення досягнення поставленої мети є матричний аналіз, теорія збурень, обчислювальна лінійна алгебра.

Експериментальна частина роботи виконана із застосуванням спеціалізованого програмного забезпечення, створеного на основі системи технічних обчислень Python та MATLAB. Для програмної реалізації алгоритмів використані методи створення програмних систем та програмування на мовах високого рівня.

Результатом кваліфікаційної роботи є програмний продукт для кількісної оцінки захищеності зображень-контейнерів від певних збурних дій, які очікуються, що використовується для розв’язку задачі про вибір контейнера з множини наявних, який дасть можливість уникнути формування стеганоповідомлення, чутливого до збурних дій.

ЗАХИЩЕНІСТЬ СТЕГАНОПОВІДОМЛЕННЯ, СПЕКТРАЛЬНИЙ РОЗКЛАД МАТРИЦІ, ЦИФРОВЕ ЗОБРАЖЕННЯ, УМОВА ЗАХИЩЕНОСТІ ВЛАСНИХ ВЕКТОРІВ

ABSTRACT

Qualification work «Improvement of the method of quantitative assessment of security of the thigh message» for the second (master's) level of higher education in the specialty 125 – Cybersecurity, contains: 22 drawings, 1 appendix, 33 references at the list of references. The work is performed on 93 pages of general text and 70 pages of main text.

The aim of this work is to improve the method of quantifying the security of stegano message to provide the ability to choose a digital image-container that generates stegano message, insensitive to disturbances.

Theoretical bases for ensuring the achievement of this goal are matrix analysis, perturbation theory, computational linear algebra.

The experimental part of the work was performed using specialized software based on the system of technical calculations Python and MATLAB. Methods of creating software systems and programming in high-level languages were used for software implementation of algorithms.

The result of the qualification work is a software product for quantifying the protection of container images from certain perturbed actions, which is expected to be used to solve the problem of choosing a container from a variety of existing ones, which will avoid the formation of steppe messages sensitive to perturbations.

PROTECTION OF STEGANO MESSAGES, SPECTRAL DECOMPOSITION OF THE MATRIX, DIGITAL IMAGE, CONDITION OF PROTECTION OF EIGENVECTORS

ЗМІСТ

ВСТУП

Людське суспільство на поточному етапі його розвитку, ставить за свою мету побудову відкритого для всіх і спрямованого на розвиток інформаційного суспільства, в якому кожен міг би створювати і накопичувати інформацію та знання, мати до них вільний доступ, користуватися і обмінюватися ними, щоб надати можливість кожній людині повною мірою реалізувати свій потенціал, сприяючи суспільному і особистому розвиткові та підвищуючи якість життя.

Згідно з Конституцією України, кожному гарантується таємниця листування, телефонних розмов, телеграфної та іншої кореспонденції [1]. Однак забезпечення секретності того чи іншого повідомлення у сучасному світі інформаційних технологій — не тривіальна задача. Одним із способів забезпечення секретності передачі повідомлення є використання стеганографії, науки приховування секретних даних у звичайному, несекретному файлі чи повідомленні – контейнері, з ціллю уникнення самого факту виявлення секретної передачі. На сьогоднішній день в якості контейнерів найчастіше використовуються цифрові зображення.

Сучасні стеганографічні методи, за допомогою яких відбувається вбудова додаткової інформації (ДІ) в контейнер, результатом чого є стеганоповідомлення, повинні задовольняти багатьом вимогам, одною з яких є стійкість до атак проти вбудованого повідомлення, чи стійкість до збурень, в результаті яких відбувається збурення стеганоповідомлення, спотворення (знищення) вбудованої (ДІ). Найпоширенішою атакою проти вбудованого повідомлення є атака стиском, яка не привертає до себе уваги, але змінює (можливо значно) стеганоповідомлення. Таким чином, забезпечення стійкості до збурень (нечутливості стеганоповідомлення), зокрема до атаки стиском є актуальною задачею при створенні прихованого (стеганографічного) каналу зв’язку.

При обраному стеганографічному алгоритмі зниження чутливості стеганоповідомлення до збурень можливо досягти шляхом вибору контейнеру, який дасть можливість уникнути формування чутливого стеганоповідомлення.

Метою роботи є удосконалення методу кількісної оцінки захищеності стеганоповідомлення для забезпечення можливості вибору цифрового зображення-контейнера, що породжує стеганоповідомлення, нечутливе до збурень.

Для досягнення поставленої мети необхідно розв’язати наступні задачі:

– виконати детальний аналіз предметної області та дослідити існуючу реалізацію методу кількісної оцінки захищеності стеганоповідомлення;

– обрати формальні параметри цифрового зображення, їх кількісні характеристики, які можуть використовуватися як показники захищеності вбудованої в контейнер ДІ;

– розробити алгоритм удосконаленого методу кількісної оцінки захищеності стеганоповідомлення та його програмну реалізацію;

Об’єктом дослідження є процеси організації прихованого (стеганографічного) каналу зв’язку.

Предметом дослідження є нечутливі до збурень стеганоповідомлення, сформавані на основі ЦЗ-контейнера.

Теоретичними основами забезпечення досягнення поставленої мети є матричний аналіз, теорія збурень, обчислювальна лінійна алгебра. Практична частина роботи виконана із застосуванням спеціалізованого програмного забезпечення, створеного на основі системи технічних обчислень Python та MATLAB. Для програмної реалізації алгоритма використані методи створення програмних систем та програмування на мовах високого рівня.

Наукова новизна роботи полягає в удосконаленні методу кількісної оцінки захищеності стеганоповідомлення від збурень шляхом заміни формальних параметрів ЦЗ, збурення яких враховуються при обчисленні обсягу захищеної нформації.

Практична цінність кваліфікаційної роботи полягає в розробці програмного продукту для кількісної оцінки захищеності зображень-контейнерів від певних збурень, які очікуються, що використовується для розв’язку задачі про вибір контейнера з множини наявних, який дасть можливість уникнути формування стеганоповідомлення, чутливого до збурень.

Робота складається із вступу, чотирьох розділів, висновку, переліку посилань та додатків.

У вступі обґрунтовано актуальність дослідження, сформульована мета роботи та перераховані задачі, виконання яких необхідно для досягнення поставленої мети.

У першому розділі представлено загальний огляд задачі кількісної оцінки захищеності стеганоповідомлення. Розглянуто основні типові стеганоалгоритми, що становили частину експериментів, а також приведено обґрунтованість їх використання. Розглянуто методи, що використовуються для збурення цифрового зображення. Детально досліджено існуючий метод кількісної оцінки захищеності стеганоповідомлення.

У другому розділі представлені теоретичні основи удосконалення методу кількісної оцінки захищеності стеганоповідомлення. Запропоновано нову умову вибору захищених власних векторів, що враховує індивідуальні характеристики контейнера.

У третьому розділі представлена практична реалізація удосконаленого методу кількісної оцінки захищеності стеганоповідомлення. Детально описано вибір засобів реалізації програмного продукту. Представлена розробка модуля контролю виконання стадій Just. Для вирішення проблеми відсутності централізованого контролю над виконанням програмного коду, підтримки великих за своїм розміром проектів, легкого конфігурування та можливості логування інформації, що виводиться застосунком, була розроблено спеціальний модуль для забезпечення комплексного контролю над виконанням застосунку. Представлена розробка модулю єдиної області

знань та кешу CompCache. Впроваджена ідея альтернативного підходу до програмування предметної області, яка передбачає централізований та уніфікований процес опису зв’язків між усіма кроками алгоритму та безпосередньо процес виконання. Реалізовано функціонал збереження копій обчислень у тимчасовому сховищі операційної системи для забезпечення можливості виконання швидкої десеріалізації при рекурентних експериментах. Створена програмна реалізація модульного тестування застосунку. Розглянута реалізація системи зв’язку між інтерфейсом програми та модулем обчислень за допомогою рушія мови MATLAB Engine API. Розглянута інструкція з експлуатації програмного продукту «Wiagond».

В четвертому розділі проаналізовані умови праці і вибір заходів і засобів захисту від небезпечних і шкідливих виробничих факторів на робочому місці розробника програмного забезпечення. Проаналізовано можливі техногенні небезпеки і виконано вибір заходів і засобів забезпечення безпеки у надзвичайних ситуаціях. Виконано розрахунок основного технічного засобу захисту в системі електробезпеки в приміщеннях із обчислювальною технікою — захисного заземлення.

1 ДОСЛІДЖЕННЯ ЗАДАЧІ КІЛЬКІСНОЇ ОЦІНКИ ЗАХИЩЕНОСТІ СТЕГАНОПОВІДОМЛЕННЯ

1.1 Задача вибору контейнеру для організації стеганографічного каналу зв’язку

Стеганографія — це наука, яка вивчає способи й методи захисту конфіденційної інформації, основною задачею якої є приховування самого факту існування секретних даних при їхній передачі, зберіганні або обробці. Під прихованням існування інформації мається на увазі не тільки неможливість виявлення в перехопленому повідомленні наявності іншого (прихованого) повідомлення, але й взагалі неможливість виникнення будь-яких підозр на цей рахунок [2].

Розвиток стеганографії на сьогоднішній день відбувається як ніколи стрімко й багатогранно. Взагалі, стеганографія являє собою доволі об’ємну науку, котра включає у свій склад організацію прихованого каналу зв’язку, вбудовування заголовків, ідентифікаційних номерів та цифрових водяних знаків.

Порівняно новим розділом стеганографії є цифрова стеганографія, яка як наука існує лише декілька десятиліть, але уже являє собою дуже міцну область знань. Цифрова стеганографія — це розділ стеганографії, який займається питаннями реалізації стеганосистем з використанням комп’ютерної техніки, шляхом вбудови інформації, що приховується, в цифрові об’єкти [3].

У процесі стеганографування ДІ, яка є результатом попереднього кодування конфіденційного повідомлення, вбудовується в деякий об'єкт, або контейнер, у якості якого в роботі розглядається цифрове зображення (ЦЗ), результатом чого є стеганоповідомлення, яке передається по каналу загального користування або зберігається в такому виді.

Якими б різними не були напрями стеганографії, пропоновані ними вимоги багато в чому збігаються. Найбільш істотна відмінність постановки задачі прихованої передачі даних, яка розглядається в роботі, від, наприклад, постановки задачі вбудовування цифрового водяного знаку полягає в тому, що в першому випадку зловмисник повинен виявити приховане повідомлення, тоді як у другому випадку про його існування всі можуть знати.

Одними з основних вимог до стеганоперетворення є забезпечення надійності сприйняття стеганоповідомлення (збурення, що зазнає контейнер в результаті вбудови ДІ, повинні бути непомітними), а також стійкість до різного роду спотворень. Слово «непомітність» має на увазі обов'язкове включення людини в систему стеганографічної передачі даних. Людина тут може розглядатися як додатковий приймач даних, що пред'являє до системи передачі вимоги, що досить важко формалізувати.

Задача вбудовування й виділення повідомлень із іншої інформації виконує стеганосистема (рисунок 1.1), яка, як правило, має у своєму складі попередній кодер, тобто модуль, що призначений для перетворення приховуваного повідомлення (конфіденційної інформації) до виду, зручного для вбудовування в контейнер, а також кодер — модуль, призначення котрого полягає в здійсненні вкладення додаткової інформації в контейнер з урахуванням його особливостей.

Оскільки в якості контейнера розглядається ЦЗ, то попередній кодер у даному випадку найчастіше репрезентує ДІ як двовимірний/одновимірний масив біт.

Дані, що містять приховане повідомлення, можуть піддаватися навмисним атакам або випадковим перешкодам, зокрема, у каналі атаки. Існування цього факту є важливим твердженням, на базі якого будується основна ідея даної роботи, що пов'язана із прагненням знайти такий контейнер, котрий зможе найкраще протистояти таким атакам.

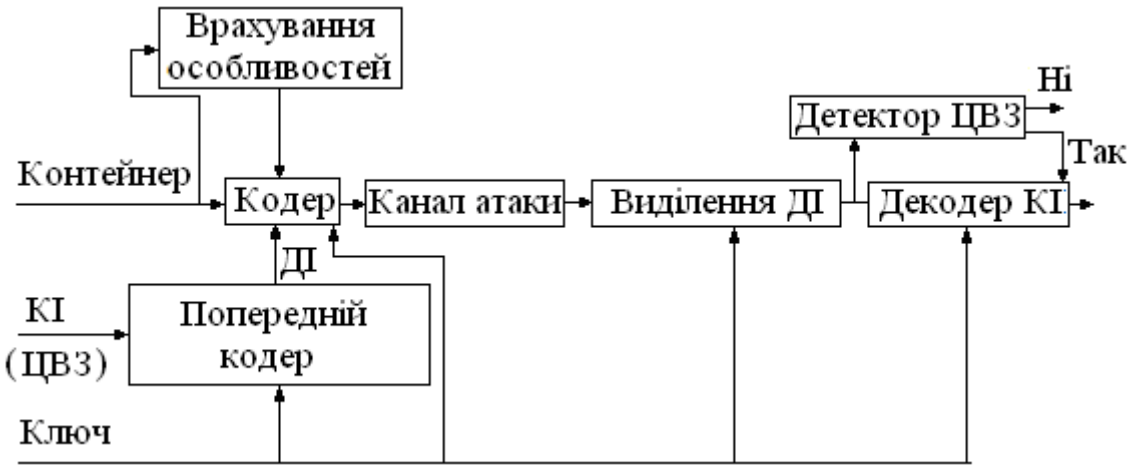

Рисунок 1.1 – Схема типової стеганосистеми

Контейнер може бути обраним, випадковим або нав’язаним. Нав’язаний контейнер — це такий контейнер, який нав’язується відправнику у випадку, коли зловмисник (порушник) підозрює про можливість прихованого обміну конфіденційною інформацією між сторонами. Випадкові ж контейнери широко і вільно використовуються у повсякденній діяльності, включаючи комп’ютерні технології, Інтернет і т.д. У більшості випадків на практиці використовуються саме випадкові контейнери [4]. Обраний контейнер залежить від повідомлення, що вбудовується, а в граничному випадку є його функцією, що може враховуватися стеганосистемою (рисунок 1.1 – модуль «Врахування особливостей»). Обраний контейнер є таким, що забезпечує певні вимоги до формованого стеганоповідомлення найкращим чином.

Існує принципова можливість вибору оригінального ЦЗ таким чином, щоб з урахуванням використовуваного стеганоалгоритму, обраного стеганошляху, секретного ключа це зображення можна було б розглядати як стеганоповідомлення, тобто як результат вбудови ДІ. Такий ідеальний контейнер цікавий не з точки зору того, що немає потреби виконувати певні математичні дії та кроки стеганоперетворення з ціллю зменшення обчислювальної складності, а с той точки зору, що надійність сприйняття у такому випадку буде достовірною подією, адже стеганоповідомлення є

оригінальним ЦЗ, а тому методи і інструменти стеганоаналізу тут будуть недієздатними.

Але задача вибору «ідеального» контейнера завжди була, є і, вірогідніше за все, буде залишатися однією із найбільш обчислювально складних задач. У загальному випадку, така задача зводиться до прямого перебору з аналізом властивостей всеможливих ЦЗ-контейнерів, розмір яких є дуже великим.

Процес вибору неідеального, але такого, що задовольняє певним вимогам чи забезпечує покращення тої чи іншої властивості стеганоповідомлення, контейнера у цілому являє важливий напрям стеганографії, який розглядається в даній роботі.

Нехай у відправника, який організовує прихований канал зв'язку, де передбаченими є атаки проти вбудованого повідомлення, є певна множина доступних оригінальних контейнерів, наприклад, альбом або певна колекція. Тоді в інтересах відправника знайти серед наявних можливих контейнерів такий, який забезпечить найменшу чутливість стегаоповідомлення до збурень.

1.2 Вибір стеганометоду для моделювання процесу стеганоперетворення в задачі про вибір контейнера

Підрозділ присвячено вибору стенографічного алгоритму, що використовувався в роботі для конкретизації стійкого стеганоперетворення.

Взагалі виділяють два найбільш поширених способи приховування даних: в області початкового сигналу (в просторовій області ЦЗ) і в області перетворення (частотна область, область перетворення Уолша-Адамара, області різноманітних розкладань матриці ЦЗ-контейнера тощо) [5].

Критеріями для вибору стеганометоду, що можна використовувати для організації стеганоперетворення у наступних дослідженнях, стали: наявність стійкості методу до атак проти вбудованого повідомлення, загальна

поширеність і популярність, відсутність обмеженості на область застосування, наявність гнучких параметрів вбудовування інформації для управління ступенем внесення змін до контейнеру в результаті стеганоперетворення.

Основним в роботі, враховуючи її мету, є перший критерій, тому такі поширені на сьогоднішній день стеганометоди, як LSB, метод квантування та інші були відкинуті після попереднього аналізу.

Як такий, що задовольняє необхідним критеріям, було обрано метод Коха-Жао, що працює в частотній області контейнера. Суть методу полягає у зміні співвідношення між абсолютними значеннями коефіцієнтів дискретно-косинусного перетворення у середньочастотній області блоків зображення [6].

Відомо, що найменші візуальні спотворення в ЦЗ вноситимуться при модифікації у високочастотних коефіцієнтів, проте будь-яка атака проти вбудованого повідомлення, наприклад, стисненням, гарантовано змінить/зруйнує вбудоване повідомлення. Найменша ймовірність руйнування повідомлення спостерігатиметься при вбудовуванні ДІ завдяки збуренням низькочастотних коефіцієнтів, проте це з великою ймовірністю призведе до появи видимих спотворень зображення, що руйнує одну з основних вимог будь-якого стеганоперетворення — надійність візуального сприйняття стеганоповідомлення. Масив коефіцієнтів ДКП та їх поділ на область низькочастотних, середньочастотних та високочастотних на прикладі блоку 8×8 приведено на рисунку 1.2.

Як компроміс між вимогою стійкості стеганоперетворення до атак проти вбудованого повідомлення та вимогою збереження надійності сприйняття стеганоповідомлення для вбудови ДІ обирається область середньочастотних коефіцієнтів [7].

Згідно з методом Коха і Жао на початковому етапі матриця цифрового зображення розбивається на блоки розміром 8×8 пікселів. ДКП застосовується до кожного блоку, що використовується в процесі

стеганоперетворення, внаслідок чого отримують матриці 8×8 коефіцієнтів ДКП, які позначають $\Omega_b(\upsilon,\nu)$, де b - номер блоку контейнера C, а (υ,ν) — позиція коефіцієнта в цьому блоці. Такий блок при цьому призначений для приховування одного біта ДІ [7].

Під час організації секретного каналу абоненти повинні попередньо домовитись про два конкретні коефіцієнти ДКП з кожного блоку, які будуть використовуватись для приховування даних. Задамо дані коефіцієнти їх координатами в масивах коефіцієнтів ДКП: (υ_1,ν_1) і (υ_2,ν_2). Зазначені коефіцієнти повинні відповідати середнім частотам, що забезпечить збереження надійності сприйняття стеганоповідомлення зі значною ймовірністю, до того ж інформація не спотворюватиметься при JPEG-компресії з малим коефіцієнтом стиснення.

Безпосередньо процес приховування починається з випадкового вибору блоку C_b зображення, призначеного для кодування b-го біта повідомлення. Вбудовування інформації здійснюється таким чином: для передачі біта "0" прагнуть, щоб різниця абсолютних значень обраних коефіцієнтів ДКП перевищувала деяку позитивну величину, а для передачі біта "1" ця різниця робиться меншою порівняно з деякою негативною величиною:

$$\begin{cases} \left|\Omega_b(\upsilon_1,\nu_1)\right| - \left|\Omega_b(\upsilon_2,\nu_2)\right| > P, & \text{де } m_b = 0; \\ \left|\Omega_b(\upsilon_1,\nu_1)\right| - \left|\Omega_b(\upsilon_2,\nu_2)\right| > -P, & \text{де } m_b = 1; \end{cases} \qquad (1.1)$$

Таким чином, первинне зображення спотворюється за рахунок внесення змін до коефіцієнтів ДКП, якщо їх відносна величина не відповідає біту, що приховується. Чим більше значення P, тим стеганосистема, створена на основі даного методу, є більш стійкою до компресії, проте якість зображення значно погіршується.

Після відповідного внесення корекції значення коефіцієнтів, які повинні задовольняти нерівності (1.1), проводиться зворотне ДКП.

Для вилучення даних у декодері виконується аналогічна процедура вибору коефіцієнтів, а рішення про переданий біт приймається відповідно до наступного правила:

$$\begin{cases} m_b^* = 0, & \text{де } \left|\Omega_b^*(\upsilon_1, \nu_1)\right| > \left|\Omega_b(\upsilon_2, \nu_2)\right| \\ m_b^* = 1, & \text{де } \left|\Omega_b^*(\upsilon_1, \nu_1)\right| < \left|\Omega_b(\upsilon_2, \nu_2)\right| \end{cases}$$

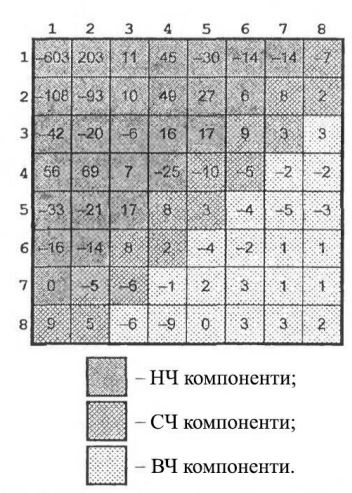

Рисунок 1.2 – Масив коефіцієнтів ДКП

Таким чином, для методу Коха і Жао забезпечується його стійкість до незначних атак проти вбудованого повідомлення, зокрема до атаки стиску з незначними коефіцієнтами стиску, значна ймовірність забезпечення надійності сприйняття, а також відсутність обмежень на область застосування. Наявність параметра P, що використовується при вбудові ДІ, дає можливість для керування властивостями відповідної алгоритмічної реалізації метода.

1.3 Вибір збурень для моделювання процесу атак проти вбудованого повідомлення

Головним критерієм вибору збурень для моделювання процесу атак проти вбудованого повідомлення є спрямованість на методи модифікації цифрових зображень, що не порушують надійність сприйняття

стеганоповідомлення, оскільки інакше буде розкритою наявність неавторизованого втручання, наявність атаки на прихований канал зв’язку, в чому порушник є незацікавленим. Наприклад, результат накладання на ЦЗ шуму «сіль-перець» при будь-якому параметрі шуму є очевидним (рисунок 1.3), накладання пуассонівського шуму в процесі стеганографічної атаки проти вбудованого повідомлення також є малоймовірним (рисунок 1.4), оскільки сприяє появі артефактів на зображенні.

а б

Рисунок 1.3 – Ілюстрація атаки проти вбудованого повідомлення: а – оригінальне ЦЗ; б – ЦЗ, що зазнало накладання шуму «сіль та перець»

а б

Рисунок 1.4 – Ілюстрація атаки проти вбудованого повідомлення: а – оригінальне ЦЗ; б – ЦЗ, що зазнало накладання пуассонівського шуму

Однією із найбільш частих атак проти вбудованого повідомлення є атака стиском. Це пов’язано з тим, що на сьогоднішній день, враховуючи величезні обсяги інформації, що зберігається, передається, обробляється, для цієї інформації використовуються формати з втратами. Таким чином, атака стиском стає такою, що не привертає уваги, але вносить зміни у зображення, що можуть привести до руйнування ДІ.

Найбільш поширеним форматом для збереження з втратами цифрових зображень є JPEG.

Непомітність результатів стиску ЦЗ є можливою завдяки особливостям системи сприйняття людини. Добре відомо, що зображення мають велику психовізуальну надлишковість; око людини подібне низькочастотному фільтру, що пропускає дрібні деталі. Особливо непомітні спотворення у високочастотній області зображень. Ці особливості людського зору і використовуються при розробці алгоритмів стиску зображень, серед яких JPEG.

Збереження ЦЗ в форматі JPEG є одною з атак, що використовується в роботі.

Серед атак проти вбудованого повідомлення поширеними залишаються атаки накладанням різноманітних шумів, але таким чином, щоб, по можливості, не порушувати надійність сприйняття ЦЗ. Це пов’язано з тим, що сам результат вбудови ДІ в стеганографії часто трактується як накладання шуму, зокрема, при застосуванні LSB-методу. Такі атаки є такими, що важко виявляються, особливо, коли параметри шуму обрані так, що величина збурення є незначною, але очевидно призводять до змін в стеганоповідомленні. При моделюванні атак проти вбудованого повідомлення в роботі використовувалися: гаусівський шум з нульовим математичним очікуванням та квадратичним відхиленням 0,001, 0,01 та 0,1; мультиплікативний шум, для якого дисперсія обиралася як 0,001, 0,01 та 0,1.

1.4 Метод кількісної оцінки захищеності стеганоповідомлення

Метод, що був покладений у основу роботи, приведено на рисунку 1.5 [8]. Основними значущими перевагами цього методу є його незалежність від конкретики атак проти вбудованого повідомлення, конкретики використовуваного при побудові прихованого каналу зв'язку стеганоалгоритму.

Крок 1. Побудувати нормальні СР: $A_j = U_j \Lambda_j U_j^T$, $\overline{A}_j = \overline{U}_j \overline{\Lambda}_j \overline{U}_j^T$, $j = \overline{1,k}$;

Крок 2. Для $j = 1,2,...k$:

а) Побудувати $\overline{VES_j}(i) = gap_{abs}(i, \overline{A}_j)$; $VES_j(i) = \dfrac{\overline{VES_j}(i)}{\left\|\overline{VES_j}\right\|}$, $i = \overline{1,n}$;

б) Побудувати $\overline{OTKLONENIYE_j}(i) = \sin \theta_i^{(j)}$, де $\theta_i^{(j)}$ — кут між $u_i(A_j)$ і $u_i(\overline{A}_j)$,

$OTKLONENIYE_j(i) = \dfrac{\overline{OTKLONENIYE_j}(i)}{\left\|\overline{OTKLONENIYE_j}\right\|}$, $i = \overline{1,n}$;

в) Побудувати вектор INF_j розподілу ДІ по ВВ СП:

$\overline{INF}_j(i) = VES_j(i) * OTKLONENIYE_j(i)$; $INF_j(i) = \dfrac{\overline{INF}_j(i)}{\sum_{i=1}^{n} \overline{INF}_j(i)} * 100\%$, $i = \overline{1,n}$;

г) Визначення ВЗ $\overline{\lambda}_{t_1}^{(j)}, ..., \overline{\lambda}_{t_p}^{(j)}$ СП $\overline{A}_j$ з достатньою абсолютною відокремленістю по відношенню до збурення E з використанням вектора $\overline{VES}_j$; визначення захищених ВВ;

д) Визначення обсягу захищеної інформації в СП $\overline{A}_j$: $OBYOM(j) = \sum_{i=1}^{p} INF_j(t_i)$;

Крок 3. Визначення СП з найбільшим обсягом ЗІ: $OBYOM(m) = \max_{1 \le j \le k} OBYOM(j)$;

A_m — шуканий контейнер

Рисунок 1.5 – Метод вибору контейнеру, що забезпечує найменшу чутливість СП

Чутливість стеганоповідомлення до збурень у випадку симетричної матриці визначається чутливістю збурених власних векторів (ВВ) матриці контейнера при стеганоперетворенні. Виходячи зі значень збурень власних векторів і абсолютних відокремленостей відповідних власних значень (ВЗ) можливо зробити якісні апріорні оцінки чутливості стеганоповідомлення до збурень.

Як правило, матриця ЦЗ-контейнера F не задовольняє властивості: $F = F^T$. Для отримання її симетричного виду можливо піти по наступному

шляху. Поставимо у відповідність довільній квадратній матриці F дві симетричні матриці A, B того ж розміру за наступним правилом:

$$F=\begin{pmatrix} a_{11} & a_{12} & a_{13} & \dots & a_{1n} \\ a_{21} & a_{22} & a_{23} & \cdots & a_{2n} \\ a_{31} & a_{32} & a_{33} & \cdots & a_{3n} \\ \dots & \dots & \dots & \dots & \dots \\ a_{n1} & a_{n2} & a_{n3} & \cdots & a_{nm} \end{pmatrix} \to A=\begin{pmatrix} a_{11} & a_{12} & a_{13} & \dots & a_{1n} \\ a_{12} & a_{22} & a_{23} & \cdots & a_{2n} \\ a_{13} & a_{23} & a_{33} & \cdots & a_{3n} \\ \dots & \dots & \dots & \dots & \dots \\ a_{1n} & a_{2n} & a_{3n} & \cdots & a_{nn} \end{pmatrix}, B=\begin{pmatrix} a_{11} & a_{21} & a_{31} & \dots & a_{n1} \\ a_{21} & a_{22} & a_{32} & \cdots & a_{n2} \\ a_{31} & a_{32} & a_{33} & \cdots & a_{n3} \\ \dots & \dots & \dots & \dots & \dots \\ a_{n1} & a_{n2} & a_{n3} & \cdots & a_{nn} \end{pmatrix}, \quad (1.2)$$

які розглядаються нижче як матриці контейнера.

При вбудові ДІ у вхідний контейнер це стеганоперетворення представляється у вигляді вбудови у верхній (нижній) трикутник матриці $A(B)$ з наступним симетричним відображенням результату відносно головної діагоналі $A(B)$. Нехай підсумком такої вбудови є симетричні матриці $\overline{A}$ і $\overline{B}$. При остаточному реальному формуванні матриці СП використовується верхній трикутник $\overline{A}$ і нижній трикутник матриці $\overline{B}$ [8].

Нехай E — матриця довільного збурення, якого зазнає контейнер (або СП). В загальному випадку $E \neq E^T$. Матриці E поставимо у відповідність дві симетричні матриці того ж розміру, використовуючи правило (1.2), розглядаючи матрицю, що відповідає верхньому (нижньому) трикутнику E як збурну для контейнера (СП), отриманого на основі $A(B)$, що дає принципову можливість матрицю довільного збурення й, як наслідок, матрицю СП також розглядати нижче як симетричні [8].

Нехай F — симетрична $n \times n$-матриця, елементи якої $f_{ij} \in R$, $i, j = \overline{1,n}$, з ВЗ $\lambda_i \in R$, $i = \overline{1,n}$, і ортонормованими ВВ u_i, $i = \overline{1,n}$, спектральне розкладання (СР) якої визначається відповідно до формули:

$$F = U \Lambda U^T$$

де $\Lambda = diag(\lambda_1, \dots, \lambda_n)$ — матриця ВЗ;

$U = [u_1, \dots, u_n]$ — матриця ВВ.

СР назвемо нормальним, якщо елементи матриці Λ задовольняють співвідношенню: $|\lambda_1| \geq \ldots \geq |\lambda_n|$, а ВВ u_i, $i = \overline{1,n}$, лексикографічно додатні [8].

На другому кроці алгоритму необхідно побудувати нормований вектор для кожного стеганоповідомлення, який виражає вагу власних значень, та який обчислюється шляхом знаходження абсолютної відокремленості кожного власного значення $\lambda_i \in R$, яка визначається відповідно до формули

$$gap_{abs}(i, \mathrm{F}) = \min_{i \neq j} \left\| \lambda_j \right| - \left| \lambda_i \right\|$$

Наступною дією алгоритму виконується обчислення відхилення між відповідними ВВ контейнеру та стеганоповідомлення обчисленням синусу кута між ними. Вектор відхилення також необхідно нормалізувати після виконання основних обчислень.

Чутливість ВВ u_i, який відповідає ВЗ λ_i, в межах матриці F визначається відповідно до співвідношень:

$$\frac{1}{2}\sin\theta_i \leq \frac{\left\|\Delta F\right\|_2}{\mathrm{gap}_{abs}(i, F)}, \qquad (1.3)$$

$$\frac{1}{2}\sin\theta_i \leq \frac{\left\|\Delta F\right\|_2}{\mathrm{gap}_{abs}(i, \bar{F})}, \qquad (1.4)$$

де $\bar{u}_i$ — нормований збурений ВВ;

θ_i — кут між u_i і $\bar{u}_i$.

Побудова вектору розподілу ДІ по ВВ стеганоповідомлення відбувається наступним чином: необхідно знайти результат множення відповідних елементів вектору ваги власних значень та вектору відхилення. Після виконання обчислень вектор нормалізується.

Для роботи методу необхідним є визначення ВЗ, що мають по відношенню до збурення E достатню абсолютну відокремленість, та захищених від збурення E ВВ. Згідно з [8], ВЗ λ_i має достатню абсолютну відокремленість стосовно збурення E, якщо

$$\|E\|_2 < \frac{gap_{abs}(i, \overline{A})}{2} \quad ;$$

а ВВ, які відповідають ВЗ із достатньою абсолютною відокремленістю стосовно збурення E, називаються захищеними від розглянутого збурення.

На завершальному кроці алгоритму відбувається відбір лише тих елементів відповідних векторів розподілу додаткової інформації по власним векторам стеганоповідомлення, що відповідають умові необхідної захищеності. Таким чином, для кожного з контейнерів буде поставлено у відповідність кількість захищеної інформації в ньому. Тому шуканим контейнером буде той, де така кількість захищеної інформації найбільша.

Метод порівняльної оцінки чутливості різних стеганоповідомлень до збурень демонструється при розв'язку задачі про вибір контейнера із заданої скінченної множини контейнерів для заданого секретного повідомлення з метою забезпечення найменшої чутливості одержуваного стеганоповідомлення. При обчисленні обсягу захищеної ДІ інформації, яка відповідно з [8] визначається як ДІ, результатом вбудови якої є збурення захищених ВВ, враховуються збурення власних векторів при стеганоперетворенні і абсолютні відокремленості відповідних ВЗ, розглянуті в якості вагових коефіцієнтів.

При детальному розгляді методу зрозуміло, що оцінку збурення ВВ при стегаоперетворенні треба уточнити (крок 2(б) (рисунок 1.5)), адже розгляд в якості такої оцінки синуса кута між відповідними ВВ ЦЗ-контейнера і стеганоповідомлення є прийнятним лише у випадку, коли сам кут є гострим. Але, як показує практика, в результаті стеганоперетворення, з урахуванням (1.3), (1.4), можливі ситуації, коли кут повороту ВВ буде тупим, але при цьому синус такого кута буде мати мале значення, хибно вказуючи на нечутливість такого ВВ. Врахування цієї ситуації очевидно дасть змогу покращити первісний метод. Крім цього, очевидним є подвійне врахування абсолютної відокремленості ВЗ, що відповідають захищеним ВВ: при виборі захищеного ВВ, а також в якості вагових коефіцієнтів при побудові вектору

розподілу ДІ (крок 2(в) (рисунок 1.5)), що також є джерелом удосконалення розглянутого методу.

Таким чином, в даному розділі представлений загальний огляд проблеми кількісної оцінки захищеності стеганоповідомлення від збурень, сформованого на основі ЦЗ-контейнера.

Розглянуто існуючий метод кількісної оцінки захищеності стеганоповідомлення, охарактеризовано його переваги, недоліки, виявлені шляхи удосконалення.

2 РОЗРОБКА ТЕОРЕТИЧНИХ ОСНОВ УДОСКОНАЛЕННЯ МЕТОДУ КІЛЬКІСНОЇ ОЦІНКИ ЗАХИЩЕНОСТІ СТЕГАНОПОВІДОМЛЕННЯ

2.1 Обґрунтування низької інформативності вагових коефіцієнтів при обчисленні вектору розподілу додаткової інформації

В попередньому підрозділі встановлено, що при побудові вектору розподілу ДІ практично двічі враховується абсолютна відокремленість ВЗ, що відповідають захищеним ВВ. При цьому власні значення симетричної матриці F є нечутливими [8] до збурень відповідно до формули:

$$\max_{1\le j\le n}\left|\lambda_j(F)-\lambda_j(F+\Delta F)\right|\le\left\|\Delta F\right\|_2,$$

де ΔF – матриця збурення, а тому їх абсолютна відокремленість $gap_{abs}(i,\mathrm{F})=\min_{i\ne j}\left|\left|\lambda_j\right|-\left|\lambda_i\right|\right|$ майже не змінюється при вбудові ДІ, а тому «не несе в собі» цієї інформації. Збурення ВЗ не залежать від розподілу ДІ по ВВ матриці контейнера, чого не можна сказати про ВВ, враховуючи співвідношення (1.3). Тому врахування абсолютної відокремленості ВЗ як вагового коефіцієнта (крок 2(в) (рисунок 1.5)) є не тільки недоцільним, але й шкідливим. Дійсно, $\left\|\Delta F\right\|_2$ — це верхня межа збурень ВЗ в результаті стеганоперетворення, але конкретні збурення, що зазнаються різними ВЗ в результаті вбудови ДІ є різними, залежать не від обсягу ДІ, а від величини модуля ВЗ (найбільше збурюються найбільші за модулем ВЗ), відмінності в збуреннях не залежать від особливостей стеганоперетворення, а врахування їх має негативні наслідки, є не тільки неінформативним, а шкідливим, оскільки не характеризує процес стеганоперетворення.

Такий теоретичний висновок знайшов своє практичне підтвердження в роботі: експериментально встановлено, що на величину обсягу захищеної інформації впливає величина відхилення саме власних векторів, а не абсолютної відокремленості відповідних ВЗ. В ході експерименту відбувалося поступове зниження вкладу абсолютної відокремленості

власного значення, тобто вектору ваги під час побудови вектору розподілу додаткової інформації по власним векторам стеганоперетворення.

Таким чином, для обчислення обсягу ЗІ, при врахуванні збурення власного вектору при стеганоперетворенні, не є інформативним, є шкідливим врахування абсолютних відокремленостей відповідних власних значень, розглянутих в якості вагових коефіцієнтів.

2.2 Визначення відхилення власних векторів в результаті стеганоперетворення

В 1.4 зазначено, що оцінка збурення ВВ при стеганоперетворенні: $\overline{OTKLONENIYE_j}(i) = \sin\theta_i^{(j)}$, де $\theta_i^{(j)}$ — кут між $u_i(A_j)$ і $u_i(\overline{A_j})$, (крок 2(б) (рисунок 1.5)) не є адекватною з урахуванням можливості на практиці значного збурення ВВ, аж до тупого кута повороту. Взагалі на практиці можлива невизначена однозначно ситуація, коли два різни вектори, з різним кутом між собою, можуть мати одне й те саме значення функції синус, при цьому один з цих векторів є нечутливим до збурень, а інший навпаки.

Теоретичне існування такої прогалини у однозначності визначення величини відхилення власних векторів, перш за все, пояснюється властивостями функції синуса. Синус — це непарна періодична функція із найменшим додатним періодом 2π, функція буде мати однакову відповідь у випадку певного кута до $\frac{\pi}{2} - x$ та $\frac{\pi}{2} + x$.

Наявність подібної неоднозначності вводить низку неточностей до роботи первісного методу, адже шукана величина відхилення власних векторів при стеганоперетворенні з врахуванням збурень є критичною для кількісної оцінки захищеності стеганоповідомлення. Більшому куту повинно відповідати певне інше більше значення результуючої функції, аніж таке саме значення, що вже присвоюється певному іншому гострому куту.

Для усунення цього недоліку в роботі запропоновано використання в якості кількісної оцінки збурення ВВ евклідової норми вектору різниці між оригінальним і збуреним векторами (рисунок 2.1).

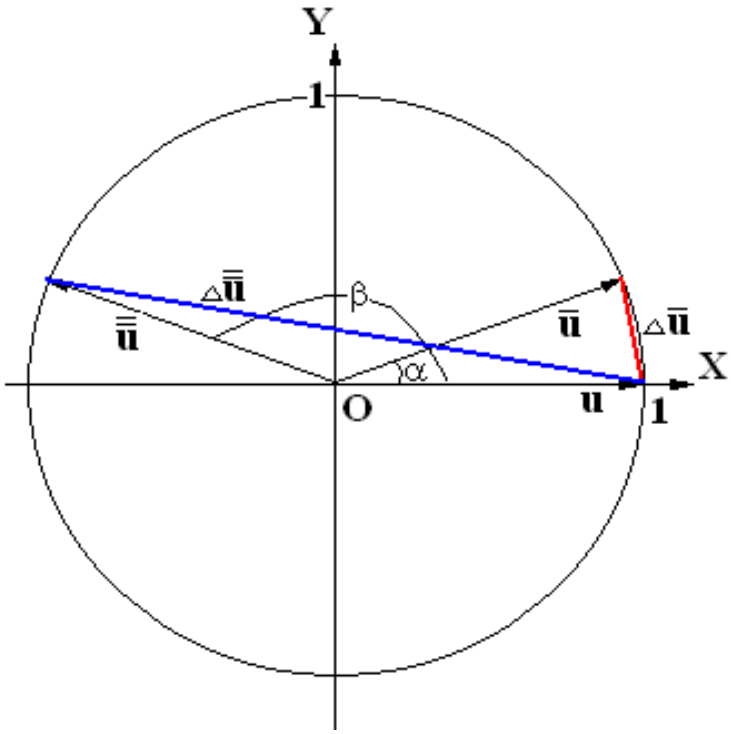

Рисунок 2.1 – Відмінність між збуреннями ВВ u при $\sin\alpha = \sin\beta$

При використанні норми вектору різниці між оригінальним і збуреним ВВ в якості кількісної оцінки збурення вектору, неоднозначності серед отримуваних значень не виникає, більшому збуренню вектору (більшому куту між поданим і збуреним векторами) відповідає більше значення норми вектору різниці і навпаки [33].

2.3 Умова захищеності власного вектору

Відповідно з [8] крок 2 г) передбачає необхідність визначення ВЗ з достатньою абсолютною відокремленістю по відношенню до збурення *E*, яке обирається у відповідності:

Крок 1. Для $j = 1, 2, \ldots, k$, де k — кількість контейнерів:

а) Побудувати $\overline{A_j} = E(A_j)$ де A_j — матриця j-го контейнера, $\overline{A}_j$ — матриця j-го контейнера після накладання деякого очікуваного збурення $E(A)$;

б) Знайти $N_j = \left\| A_j - \overline{A_j} \right\|$, де $\|v\|$ — евклідова норма $\|v\| = \sqrt{\sum_{k=1}^{N} |v_k|^2}$;

Крок 2. Знайти показник збурення $E = \frac{\sum_{i=1}^{k} N_i}{k}$

Але кожне із зображень є унікальним за своєю структурою. Використовуючи у якості кількісної оцінки збурної дії *E*, за якою обираються захищені власні вектори, усереднене значення з серії проведених попередньо експериментів, само собою зрозуміло, виконується прив'язка до характеристик тих зображень, на яких було виконано розрахунок. Це вводить додаткові умови щодо того, які саме зображення необхідно використовувати для обчислення розрахункового значення збурення, а також не враховує особливості безпосередньо тих контейнерів, що розглядаються, замінюючи важливу метрику лише усередненим значенням. Обчисливши середнє значення для певної кількості зображень, немає ніякої гарантії в тому, що це середнє значення однакового добре може бути застосовано для будь-яких інших зображень.

Замість цього до використання при удосконаленні методу [8] пропонується метрика, що не розраховується попередньо для певної вибірки зображень, а замість цього враховує особливості безпосередньо такого контейнера, об'єм захищеної інформації якого досліджується:

$$Capacity(j) = \begin{cases} \sum_{i=1}^{n} \overline{Distribution_j}(i), & якщо \quad \left\| \left| U_j(i) - \bar{U}_j(i) \right| \right\| < K \\ 0, & інакше \end{cases}$$

Для цього після вбудовування додаткової інформації отриманому стеганоповідомленню необхідно надати очікуване збурення, та для результуючої матриці виконати стандартний для інших станів розклад:

виділення симетричної матриці, нормальне спектральне розкладання та виділення власних векторів.

Тоді умовою виділення елементів відповідних векторів розподілу додаткової інформації по власним векторам стеганоповідомлення є умова того, чи норма вектору різниці між початковим власним вектором контейнера і власним вектором після накладання збурення на стеганоповідомлення менша, ніж деяке K, що є параметром удосконаленого методу і визначається експериментально.

Коефіцієнт K було встановлено цілком експериментально, його значення дорівнює 0,005. Використання саме такого K дало найбільш стабільні та очікувані результати відповідно до кількості захищеної інформації та кількості відновленої інформації, що продемонстровано на серії рисунків 2.2, 2.3, 2.4 та 2.5.

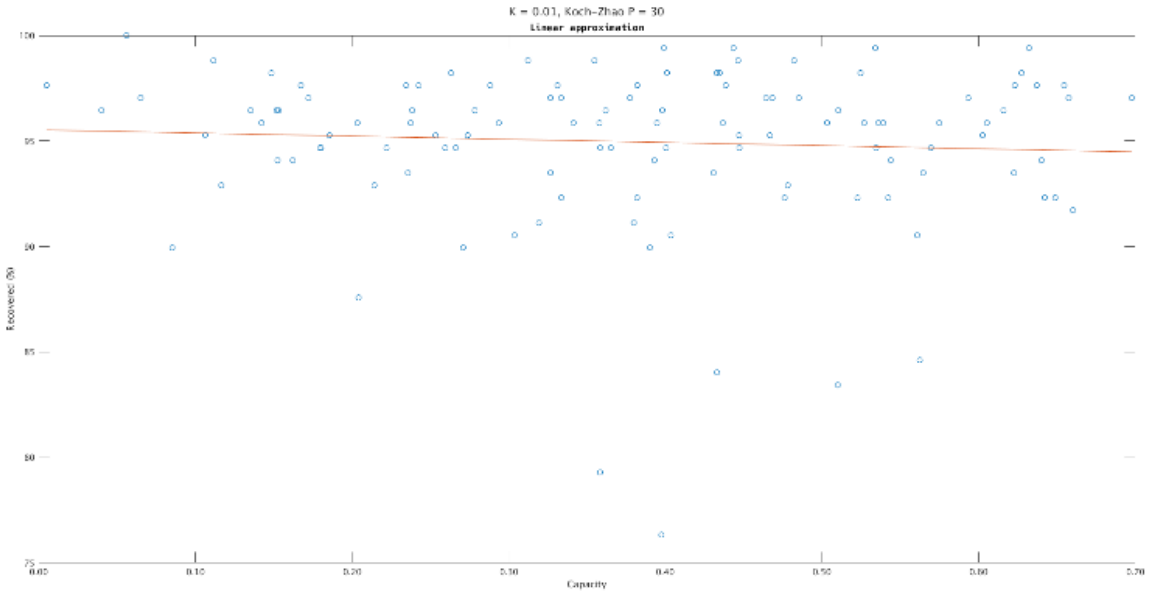

Рисунок 2.2 – Графік відповідності кількості захищеної інформації до відсотку вірно відновленої інформації при K = 0,01

Неочевидним плюсом такого удосконалення стала підтримка будь-якої серії модифікації, без необхідності в обчисленні усередненого значення

збурення для еталонної вибірки, що в реальних задачах комплексних вибірок може становити доволі складну обчислювальну задачу.

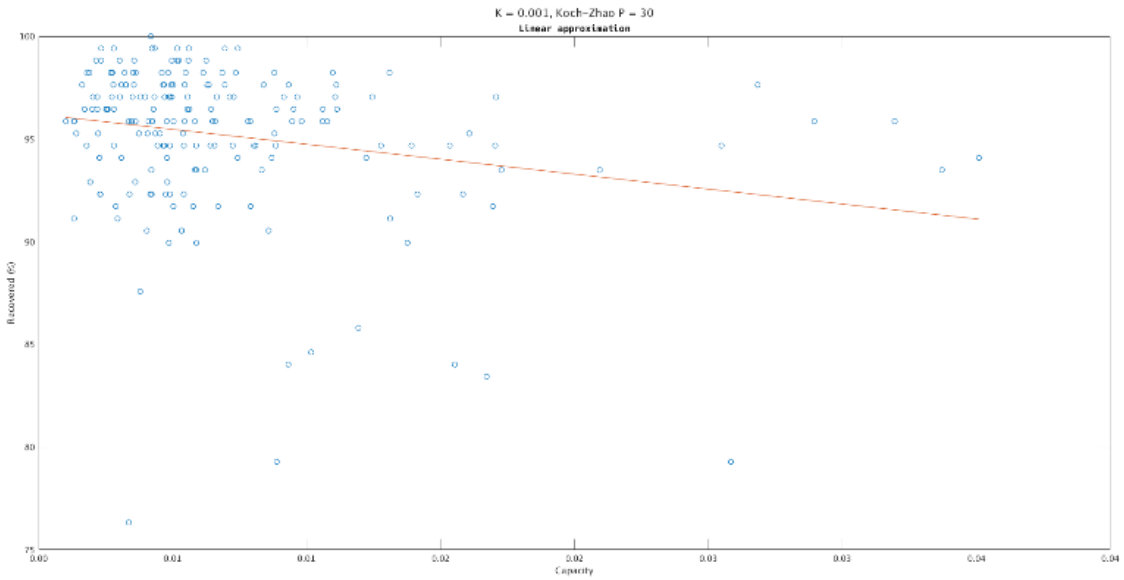

Рисунок 2.3 – Графік відповідності кількості захищеної інформації до відсотку вірно відновленої інформації при K = 0,001

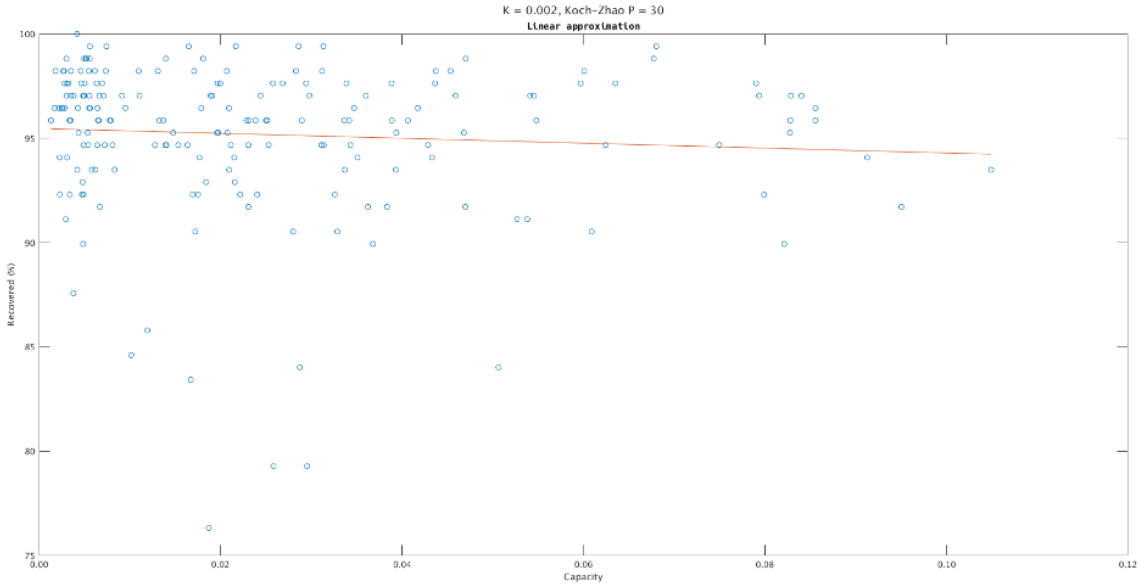

Рисунок 2.4 – Графік відповідності кількості захищеної інформації до відсотку вірно відновленої інформації при K = 0,002

2.4 Удосконалений метод кількісної оцінки захищеності стеганоповідомлення

Удосконалений метод кількісної оцінки захищеності стеганоповідомлення передбачає виконання наступних кроків:

Крок 1. Для $j = 1, 2, \ldots, k$, де k — кількість контейнерів:

а) Побудувати нормальні спектральні розкладання:

$$A_j = U_j \Lambda_j U_j^T,\ \Delta A_j = \Delta U_j \Delta \Lambda_j \Delta U_j^T,\ \bar{A}_j = \bar{U}_j \bar{\Lambda}_j \bar{U}_j^T,$$

де A_j — матриця j-го контейнера, ΔA_j — матриця j-го контейнера після вбудовування додаткової інформації, $\bar{A}_j$ — матриця j-го контейнера після вбудовування додаткової інформації та накладання деякого очікуваного збурення $E(A)$;

б) Побудувати

$$\overline{Deviation_j}(i) = \left\| \left| U_j(i) - \Delta U_j(i) \right| \right\|;\ Deviation_j(i) = \frac{\overline{Deviation_j}(i)}{\left\| \overline{Deviation_j} \right\|},$$

$i = 1, 2, \ldots, n$,

де n — кількість власних векторів j-го контейнера, $\|v\|$ — евклідова норма $\|v\| = \sqrt{\sum_{k=1}^{N} |v_k|^2}$;

в) Побудувати вектор розподілу додаткової інформації по власних векторах стеганоповідомлення:

$\overline{Distribution_j}(i) = Deviation_j(i)$;

$$Distribution_j(i) = \frac{\overline{Distribution_j(i)}}{\sum_{i=1}^{n} Distribution_j(i)} \cdot 100\%,\ i = 1, 2, \ldots, n,$$

де n — кількість власних векторів j-го контейнера;

г) Визначення об’єму захищеної інформації:

$$Capacity(j) = \begin{cases} \sum_{i=1}^{n} \overline{Distribution_j}(i), & якщо \quad \left\| \left| U_j(i) - \bar{U}_j(i) \right| \right\| < K \\ 0, & інакше \end{cases}.$$

Крок 2. Визначення контейнера з найбільшим обсягом захищеної інформації:

$$m = \arg\max_j Capacity(j);$$

Тоді A_m — шуканий контейнер.

Таким чином, основні кроки методу не змінилися за своєю структурою. У центрі уваги все ще стоїть нормальне спектральне розкладання, що виконується, згідно із оригінальним алгоритмом, для кожного контейнера та стеганоповідомлення, що розглядаються, а також, у зв’язку з введенням нової умови відбору захищених власних векторів, що враховує індивідуальні характеристики контейнера, й для стеганоповідомлення після накладання серії очікуваних модифікацій-збурень на останній.

Важливо також зауважити на тому, що перший крок алгоритму цілком і повністю полягається лише на контейнер, який розглядається, без суміжних зв’язків з іншими контейнерами, а тому може бути порахований у будь-якому порядку або навіть одночасно, використовуючи технології розпаралелювання або навіть кластеризації, що безумовно додає обчислювального потенціалу до алгоритму.

Через викладену попередньо неінформативність оцінки власних значень нормального спектрального розкладання в силу більш якісної прямої метрики, що працює безпосередньо із власними векторами, крок із обчисленням ваги було видалено.

Важливим удосконаленням методу, як вже було вказано у попередніх підрозділах, яке пов’язане з модифікацією обчислення відхилення власних векторів та вирішенням неоднозначності функції синус, стало введення евклідової норми вектору різниці між власними векторами у якості показника відхилення.

Розрахунок вектору розподілу додаткової інформації по власних векторах стеганоповідомлення зазнав змін у своєму складі, по-перше, через відсутність розрахунку кроку із обчисленням ваги.

Важливим удосконаленням методу, яке вже було описаний у серії минулих підрозділів, стало удосконалення процесу визначення об'єму захищеної інформації. Тепер нова умова вибору захищених власних векторів враховує індивідуальні характеристики контейнера та використовує для виділення захищених власних векторів власноруч уведену константу K.

Фінальним кроком методу все так само виступає процес зіставлення кожному контейнеру деякого дійсного числа, що показує кількість захищеної інформації в ньому. Тоді шуканим контейнером, згідно із удосконаленим методом кількісної оцінки захищеності стеганоповідомлення, стане такий контейнер, де кількість захищеної інформації є найбільшою.

Як вже було зазначено раніше, основне обчислювальне навантаження алгоритму насамперед пов'язане із знаходженням кількості захищеної інформації конкретного контейнеру. Однак, у центрі алгоритму стоїть спектральне нормальне розкладання, що у свою чергу, потребує виділення симетричної матриці, що саме по собі може зменшити затрати на її збереження (наприклад, зберігаючи лише будь-який трикутник, а не повну матрицю тощо), так само як і використання спектрального розкладання у противагу сингулярному розкладанню потребує лише двох матриць (матрицю власних векторів, за потребою, можна транспонувати). Крім того, матриця власних значень не тільки може зберігатися у вигляді вектору, а й зовсім не зберігатися, перш за все, через відмову від розрахунку ваги власних значень у силу більш інформативних характеристик власних векторів.

Тобто, таким чином, обчислювальна складність алгоритму є незначною, і може виконуватися на сучасному обладнанні одночасно для декількох зображень; за потребою, може обчислюватися для великої серії зображень (сотен і навіть тисяч), не накладаючи обмеження на занадто великий час виконання алгоритму. За бажанням, можна забезпечити більшу

долю обчислень із використанням відеокарти замість центрального процесору з ціллю збільшення потенціалу векторизації та розпаралелювання задач; усі виконувані операції обчислення евклідової норми, різниці між векторами тощо є гарно дослідженими та оптимізованими у складі скільки-небудь складних програмних середовищах.

2.5 Вплив захищеності інформації на якість відновлення секретного повідомлення

Важливість розуміння отриманих результатів удосконаленого методу являє собою одну з найважливіших частин застосування алгоритму.

Необхідно одразу прояснити, що приведений алгоритм не може у той чи іншій мірі однозначно передбачати з далекоглядною точністю показник того, як буде відновлена додаткова інформація після накладання на стеганоповідомлення очікуваних збурень. Такий факт є очевидним хоча б через те, що кожне із цифрових зображень є унікальним і дуже складним об’єктом для точного передбачення. Однак з точки зору введених удосконалень в метод, вдаючись до деталей, це навіть краще. Набагато цінніше буде розглянути та виділити закономірності, що будуть працювати на загальному рівні великої вибірки та різнотипності зображень, аніж виділити більш точні результати, що можуть бути з легкістю спростовані в результаті хоч яких-небудь досліджень нестандартних ситуацій, як це, скажімо, вийшло з неоднозначною ситуацією існування тупого кута між власними векторами контейнера та стеганоповідомлення, що фактично прирівнювався із допомогою функцій синуса до гострого кута.

Досліджувана величина захищеності стеганоповідомлення являє собою більш комплексну і неочевидну при першому розгляданні величину, що особливо може бути корисною під час порівняльних характеристиках між декількома контейнера, що недалеко від приведеного удосконаленого методу кількісної оцінки захищеності стеганоповідомлення.

Насамперед, необхідно виділити існування того факту, що обчислення кількості захищеної інформації для певного контейнера не дає ніякої ясності щодо того, яким саме буде відсоток відновленої інформації. Це, фактично кажучи, значить, що не можна із допомогою такого алгоритму дізнатися коли найшвидше можна досягнути, наприклад, 100% результату вилучення додаткової інформації із стеганоповідомлення. Як окремо взятий контейнер, так і параметри, за яких атакуюча сторона буде виконувати вбудовування (бінарний розподіл додаткової інформації тощо), мають у своєму складі занадто велику кількість унікальної інформації та свідчень, що не можуть бути алгоритмічно передбаченими, але на противагу цьому, кількість захищеної інформації може бути цінною при дещо іншому куті розгляданні її результаті.

Це твердження якісно підтверджується за допомогою рисунку 2.5, який показує, що кількість захищеної інформації не показує точно відновлений відсоток відновленої інформації. Беручи у якості зрізу невеликі значення захищеної інформації, скажімо 0,07, легко переконатися у тому, що для різних зображень можливий як 100% відсоток відновлення, так і аномально низький, близько 75%.

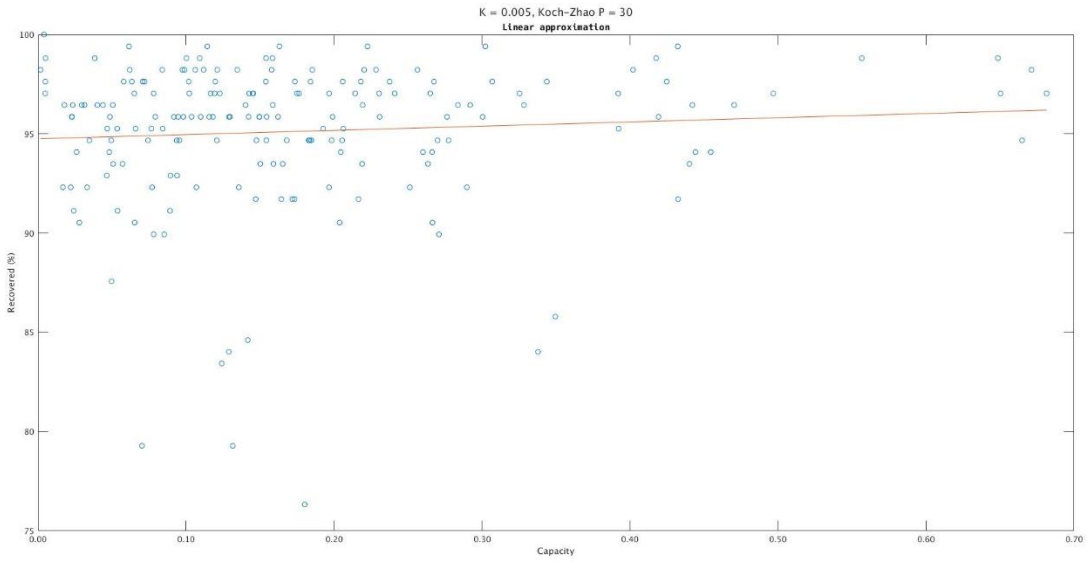

Рисунок 2.5 – Відношення кількості захищеної інформації до відсотку вірно відновленої інформації

Такі неоднозначні ситуації і є основними підводним камінням для будь-якого атакуючої сторони, що використовує стеганографічні алгоритми. Не можна зосереджувати свою увагу лише на найбільш позитивному результаті декодування додаткової інформації, адже це прийняття тільки такої стратегії може створити ситуацію, коли фактична кількість переданої інформації може навіть далеко не дорівнювати очікуваному.

Розрахованість на стабільність варіацій відновлення інформації — то, чому і присвячено поняття захищеності інформації. Саме з цією ціллю було відібрано критерій Вальда (або максиміну) як центральний під час застосування алгоритму, адже він дозволяє обрати такий результат, що є найбільш стабільним при найгіршому проявленні ситуації.

Рухаючись по осі абсцис та збільшенням зрізу кількості інформації, можна очевидно виділити підтвердження твердження, згідно із яким, чим більшою являється кількість захищеної інформації, тим більш стабільним є результат відновлення додаткової інформації із стеганоповідомлення. Беручи, наприклад, у якості зрізу кількості інформації, що розглядається, проміжок від 0,4 до 0,5, можна одразу помітити відсутність аномальних відхилень у результаті відновлення додаткової інформації; і хоча все ще можлива ситуація майже як 100% відновлення, так і менш результативного, близько 92%, тим не менш, розкид, що дорівнює 8%, набагато стабільніше оцінює ситуацію, аніж це було при невеликих значеннях захищеності інформації.

Тобто організаторам прихованого каналу зв’язку, що мають у себе на озброєнні серію певних контейнерів, дійсно може стати у нагоді представлений алгоритм, адже він надає можливість знайти серед вибірки контейнерів той, що дасть найменшу кількість помилок, ураховуючи найменш придатні умови (в першу чергу, особливості контейнеру), що нерідко є основною ціллю організаторів прихованого каналу зв’язку, - гарантувати у той чи іншій мірі те, що його стеганоповідомлення витримає серію очікуваних модифікацій та зможе надати можливість відновити прийнятну кількість інформації.

Більше того, використання у якості контейнеру такого контейнеру, що має найбільшу кількість захищеної інформації, приводить до того, що він все ще може дати бездоганний результат відновлення інформації, як і у випадку інших контейнерів з малим показником захищеності інформації, але що, безсумнівно, є більш важливим, так це існування того факту, що вибір такого контейнеру буде найменш помилковим. Мати гарантію того, що додаткова інформація, яка зазнала вбудовування, була вбудована у найстабільніший із можливих контейнерів — важливе свідчення, що безумовно стане у нагоді будь-якому спеціалісту або рядовому користувачеві стеганографічних алгоритмів. Вирішення такої проблеми являє собою важливе напрацювання, що дійсно має місце бути у реальних задачах вибору найбільш стабільного і перспективного контейнеру.

В даному розділі представлені теоретичні основи удосконалення методу кількісної оцінки захищеності стеганоповідомлення.

В результаті серії експериментів була доведена неінформативність використання обчислення вектору ваги із використанням власних значень.

Була виявлена проблема оцінювання збурення ВВ. Було запропоновано більш досконалу метрику, що полягає у вимірюванні евклідової норми різниці між власними векторами, котра не допускає експлуатації знайденої проблеми.

Впроваджено нову умову вибору захищених власних векторів, що враховує індивідуальні характеристики контейнера, замість попереднього обчислення над певною наперед визначеною, а отже небездоганною вибіркою зображень, що дозволяє враховувати особливості кожного ЦЗ-контейнера.

3 ПРАКТИЧНА РЕАЛІЗАЦІЯ УДОСКОНАЛЕННОГО МЕТОДУ КІЛЬКІСНОЇ ОЦІНКИ ЗАЩИЩЕНОСТІ СТЕГАНОПОВІДОМЛЕННЯ

3.1 Середовище застосування, обґрунтування потреб до обраних мов програмування

Перед початком розробки програмного забезпечення необхідно обрати середовище та мову програмування, адже саме з цим вибором і паралельно встановлюється вичерпний список підтримуваних платформ, а також загальна методика програмування та підхід до реалізації майбутнього застосунку.

Варто одразу прояснити, що за своєю структурою, викладений у 2 частині удосконалений метод кількісної оцінки захищеності стеганоповідомлення, принципово не полягається на езотеричні та специфічні математичні операції, підходи та досліджувану область; насамперед, потребується цілком стандартний набір операцій, що є влаштованим у більшості мов програмування, тому з великою вірогідністю можна стверджувати, що приведений удосконалений алгоритм кількісної оцінки захищеності стеганоповідомлення може бути імплементований із використанням майже більшості популярних мов програмування та архітектур, включаючи навіть вбудовуванні системи.

Мова вибору середовища застосування та мов програмування головною мірою складається з того, щоб залучити до використання найбільш підходящий, оптимізований та налаштований для цих цілей інструмент, що гарно себе зарекомендував та дозволяє одразу приступити до імплементації важливих функцій.

Важливо також обмовитися на тому, що та чи інша мова програмування, безумовно, має низку переваг використання перед іншими. Головне питання стоїть в тому, чи існує готовність платити за ті чи інші полегшення ціною інших. Не виникає ніяких сумнівів, що найбільшої

швидкості виконання можна досягнути використовуючи більш низкорівневі мови програмування з, наприклад, повністю ручним менеджментом пам’яті. Але використання такої мови програмування, як скажімо, C, у ситуації, коли кількість операцій з пам’яттю і близько не є такою, що критично впливає на продуктивність і у цілому не займає навіть 5% часу виконання, не є виправданою. Більш важливим буде звернутися до автоматичного управління пам’яттю у складі мов програмування високого рівня та отримати швидкість написання коду, гарантію від втрати пам’яті та низку інших високорівневих конструкцій. Саме на ці запитання необхідно перш за все бути готовим відповісти під час обґрунтування того чи іншого програмного середовища.

Треба підмітити, що до використання не примушується обрати один і тільки один інструмент. Якщо у такому є потреба, а витрати на зв’язок між різнотипними середовищами невеликі, можна розділити основні частини застосунку на незалежні частини, які будуть реалізовані із використанням різних технологій.

Почати вибір середовища необхідно із найпріоритетнішого напряму, а саме безпосереднього розрахунку алгоритму оцінки захищеності інформації. Розумніше всього буде обрати таке середовище, що вже має у своєму складі інструменти для роботи із цифровими зображеннями, матрицями та містить більшість елементарних операцій лінійної алгебри і не тільки (зокрема, мова йдеться про спектральне розкладання). Рішення не витрачати на власне написання такої базової та водночас складної теми має у своєму складі раціональну ланку, адже у більшості бібліотек та середовищах програмування такі операції є добре відомими та часто використовуваними, а тому добре налагодженими, протестованими та оптимізованими на максимальну продуктивність, наприклад, із використанням прийомів векторизації та спеціальних інструкцій центрального процесору. Вирішення такої задачі наодинці потребує велику кількість людино-годин, навіть не наближаюсь до самого проектування та програмування основних частин застосунку.

Крім того, вторинними критеріями вибору, орім бази готових до використання функцій, стали загальні прийоми, що покращують швидкість реалізації будь-яких проектів: мова програмування високого рівня, динамічна типізація, підтримка об’єктно-орієнтованого підходу до програмування (або альтернатива у вигляді прототипів, структурного розширення тощо), відносна швидкість написання коду та легкість синтаксичних конструкцій, наявність у складі елементарних інструментів мови колекцій у вигляді динамічних масивів, хеш-таблиць та об’єктів. Насамперед, деякі з особливостей, наприклад, динамічна типізація, не є суворим правилом, що потребує виконання, однак у цілому підвищує функціональні можливості застосунку, наприклад, з легкістю організовуючи рефлексію типів та серіалізацію даних, що є безумовним плюсом під час роботи.

Прикладами таких мов програмування і середовищ навколо них, що задовольняють викладеним критеріям є Scala [9], R [10], MATLAB [11], Julia [12] та Python [13]. У рамках поточної роботи вибір було зосереджено на MATLAB.

MATLAB — ультимативна платформа програмування та числових обчислень, яку використовують мільйони інженерів і вчених для аналізу даних, розробки алгоритмів і створення моделей. MATLAB розробляється комерційною компанією MathWorks, та являє собою величезний за змістом програмний продукт, що дозволяє здійснювати маніпуляції з матрицею, будувати графіки функцій і даних, реалізовувати алгоритми, створювати інтерфейси користувача та взаємодіяти з програмами, написаними на інших мовах програмування. У складі компонентів, що постачаються із MATLAB, є велика кількість додатків для вирішення найрізноманітніших задач: набір інструментів статистики та машинного навчання, набір інструментів глибокого навчання, набір інструментів текстової аналітики, набір інструментів обробки сигналів, набір фінансових інструментів і так далі і тому подібне [14]. Важко знайти ту популярну область досліджень, яка не була у той чи іншій мірі торкнута MATLAB, будь то машинне навчання,

символьна арифметика, лінійна алгебра або робота з математичними моделями.

MATLAB для вирішення задач користувачів постачається у складі «все в одному», надаючи доступ до редактору коду, налагоджувача, різноманітних інструментів формування звітів та дослідження математичних сутностей (наприклад, вейвлетів) тощо з одного місця, не потребуючи для роботи ніяких інше інструментів. MATLAB надає користувачам для використання, перш за все, власну мову програмування, однак за потребою можна написати власне розширення за допомогою мови програмування C++ (так названі MEX розширення [15]), або отримати доступ до середовища Java/.Net, так само як і запускати інші програми засобами операційної системи.

Мова програмування MATLAB являє собою багатопарадигмальну динамічну мову, що у цілому задовольняє усім потребам сучасності, має у своєму складі об’єктно орієнтовані можливості, фреймворк тестування, базову підтримку рефлексії через метаінформацію, а також обробку виключень [16]. Мова містить у своєму складі особливі типи даних, як структура, комірка та масив, а також особливий тип для роботи із посиланням — хендл [17].

Серед очевидних мінусів цього програмного середовища варто виділити пропрієтарність, що відображається як на впровадженні нових функцій, підтримці, так і у обов’язковій потребі у ліцензуванні — MATLAB не є безкоштовним програмним продуктом. Більшість з функцій являють собою закритий сирцевий код, доступ до якого неможливо отримати.

Із неочевидних плюсів є можливість використання середовища на більшості популярних платформ (Windows, GNU/Linux, macOS), можливість розпаралелювання на базі мультипроцесного або мультипоточного підходу, збереження робочих сесій, компіляція програм до складу автономних програм, набір інструментів побудови графіків та статистичних функцій апроксимації кривих тощо.

Необхідно також вказати те, що середовище MATLAB містить у своєму складі доволі потужний інструмент створення графічних застосунків, що дозволяє швидко та без особливих труднощів розробити базовий графічний застосунок, що буде містити у своєму складі популярні та необхідні для роботи віджети, наприклад, графіки, таблиці, кнопки та слайдери.

Однак необхідно зауважити, що MATLAB перш за все являє собою саме інженерний склад інструментів і ніколи не позиціонував себе у якості провідного розробника графічних інтерфейсів. Такі базові можливості зміни теми, налаштування стилів елементів, багатовіконного застосунку є або неможливим у певній мірі або дуже лімітованим, потребуючими шукати обхідні шляхи та винаходячи непотрібне.

Через можливість вільного зв’язку MATLAB з іншими мовами програмування, було прийнято рішення перекласти відповідальність за графічний інтерфейс на інше середовище та мову програмування, що більш повно охоплюють цю проблему.

Для вибору такого середовища необхідно перш за все звернути увагу на популярність та застосовність і функціональну складову, найчастіше представники таких груп містять ядро написане на одній мові програмування, що може бути викликане з великої кількості інших, використовуючи спеціалізовані прив’язки.

Серед таких фреймворків варто однозначно відділити ті, які лімітовані за своїм походженням від можливості бути використаним на тієї ж низці платформ, як і сам MATLAB. До переліку залучаються всі платформенно-орієнтовані фрейморки, як скажімо, Winforms, WPF та UWP, а також Cocoa — всі ці середовища націлені лише на певні операційні системи, у даному випадку Windows та macOS [18, 19].

Альтернативним рішенням, що може бути використаним, є заохочення веб-технологій для програмування нативних застосунків. Перш за все, мова йдеться про Electon — це фреймворк, що включає в себе Node.js для роботи з

back-end і бібліотеку рендерингу Chromium. Electron дозволяє створювати будь-які графічні застосунки з використанням браузерних технологій, логіка роботи яких визначається на JavaScript, HTML і CSS, а функціональність може бути розширена через систему доповнень [20].

Однак у програм, написаних із використання Electon, є також низка очевидних мінусів: навіть базові програми піддаються критиці за те, що вони несуть значні накладні витрати в порівнянні з нативними програмами зі схожою функціональністю. Програми, створені за допомогою Electron, можуть займати більше пам’яті та оперативної пам’яті та працювати повільніше, ніж подібні програми, створені за допомогою технологій, властивих операційній системі. Навіть крихітні за своєю функціональністю та кодовою базовою застосунки, зазвичай включають кілька процесів операційної системи для виконання. Як правило, має бути "основний" процес і кілька процесів "відтворення". Публікація ж таких застосунків, фактично зводиться до комплектування повноцінного Chromium та NodeJS, що потребує велику кількість місця на сховищі [21].

Замість цього, увагу було зосереджено на більш швидких, доступніших та легковісних програмних рішеннях, що гарно зарекомендували себе у світі графічних застосунків. Мова йдеться про такі широко відомі фрейморки, як Qt, Gtk, wxWidgets, FLTK тощо.

У рамках поточної роботи вибір було зупинено на Qt — це набір інструментів та віджетів для створення графічних інтерфейсів користувача, а також кросплатформних додатків, які працюють на різних програмних і апаратних платформах, таких як Linux, Windows, macOS, Android або вбудованих системах з незначними змінами або без змін у базовій кодовій базі, але залишаючись нативним додатком з нативними можливостями растеризації та швидкості [22].

Як вже було припущено до цього, Qt, в силу своєї популярності, може використовуватися не лише на мові програмування C++, а також може бути використаним у багатьох інших мовах програмування: Ada (QtAda), C#

(Qyoto/Kimono), Java (Qt Jambi), Qt Jambi, Node.js, Pascal, Perl, PHP (PHP-Qt), Ruby (QtRuby), та Python (PyQt, PySide) [23].

З приведеного списку мов програмування найбільший інтерес становила мова програмування Python, насамперед, через низку очевидних плюсів, що найкраще розкриваються при конкретній задачі розробки графічного застосунку: гнучкість, розширюваність, інтерпретованість, простота синтаксису.

Можна довго переліковувати переваги мови програмування Python, але цифри кажуть самі за себе: за щорічним дослідженням популярнішої системи питань і відповідей для професійних програмістів та ентузіастів Stackoverflow, Python знаходиться на 3 місці за популярністю, програючи лише очевидній трійці HTML/CSS/JS. Все це дає певні висновки, що Python — популярна, розвинута та затребуваною мова програмування, що дозволить швидко вирішити поставлені задачі щодо програмування графічного застосунку програмного продукту.

Важливо також виділити, що за бажанням спрямованного розповсюдження програмного продукту, існує можливість використовувати для цих цілей технологію контейнеризації, таких як Docker, для безболісного вирішення проблем інтеграції, інсталяції та взаємного використання декількох версій програмного забезпечення. Як MATLAB, так і Python можуть бути вільно використані у своїх проектах через те, що обидві редакції існують у складі офіційного репозиторію Docker [24, 25].

3.2 Реалізація модуля контролю виконання стадій Just

Програмування у складі середовища MATLAB, як правило, полягається у реалізації невеликих за своїх розміром скриптів, що виконують ту чи іншу задачу дослідження. Такий підхід є неймовірно зручним для швидкого старту та переключення між різними реалізаціями.

Однак труднощі підтримки такого підходу можуть стати неймовірно ускладнюючими під час роботи із проектами, великими за своїми розмірами. Такі проекти можуть розташовуватися у складі великої кількості сирцевих кодів, коли контроль їх виконання не може бути виконаний «як є», кожен із функціональних модулів може мати залежності до інших операцій, станів та контекстів, і це може стати важливим параметром під час проектування кодової бази.

У якості вирішення проблеми проектної структури застосунку було введено модуль контролю виконання стадій Just. Взагалі кажучи, класично ця проблема вирішується за допомогою так названих програмний засобів для автоматизації створення та тестування програмного забезпечення. Саме за такий тип інструментів у якості еталону і було обрано призначення модуля Just.

Just оперує поняттям стадій як центральним у середовищі виконання. Для забезпечення працездатності стадій, Just охоплює створення контексту виконання для полегшення простоти роботи із модульністю застосунку.

У складі контексту виконання Just варто виділити дві основні частини: модуль логування та модуль конфігурації MProps. Модуль логування присвячений вирішити одну з важливих проблем, що може виникнути під час програмування великих за своїм розміром проектів — відсутність уніфікованого інструменту виведення інформаційних повідомлень про роботу програми. Крім того, саме у середовищі MATLAB існують проблеми із легким форматуванням інформації, потребуючи явного перетворення даних.

Таким чином, модуль логуваннння бере на себе відповідальність за забезпечення якісного доступу до функцій виведення повідомлень до кінцевих потоків даних. Важливою складовою цього процесу є можливість фільтрації повідомлень за їх важливістю, точніше кажучи рівнем важливості: TRACE, INFO, WARNING, ERROR з відповідними позначками трасувальних, інформаційних, попереджувальних та помилкових

повідомлень. Наприклад, трасувальні повідомлення можуть бути корисними під час безпосередньої розробки та відлагодження роботи стадій, в той час як виводити лише повідомлення про помилки доцільно при стабільному релізі. Необхідно звернути увагу на тому, що модуль логування є доволі комплексним і дозволяє охопити виведення інформації не тільки до потоку виведення інформації консолі MATLAB, а й до файлів, примітивів синхронізації (за умови асинхронного виконання) та у пустоту (коли є потреба в повному ігноруванні інформації стадії, наприклад, при запуску однієї стадії з під іншої).

Крім того, модуль логування також має у своєму складі функціонал так званих стайлерів, що відповідають за форматування тексту, що відображається на кінцевому потоку. Дією стайлера за замовчуванням є автоматичне вирівнювання стрічок згідно з певним лімітом (наприклад, 80 символів), що дозволяє не піклуватися про можливе виведення значення змінної занадто довгої довжини. Крім того, наприклад, існує окрема реалізація стайлеру, що дозволяє враховуючи особливості консольного вікна MATLAB та виводити кольорові повідомлення, використання котрої приведено на рисунку 3.1

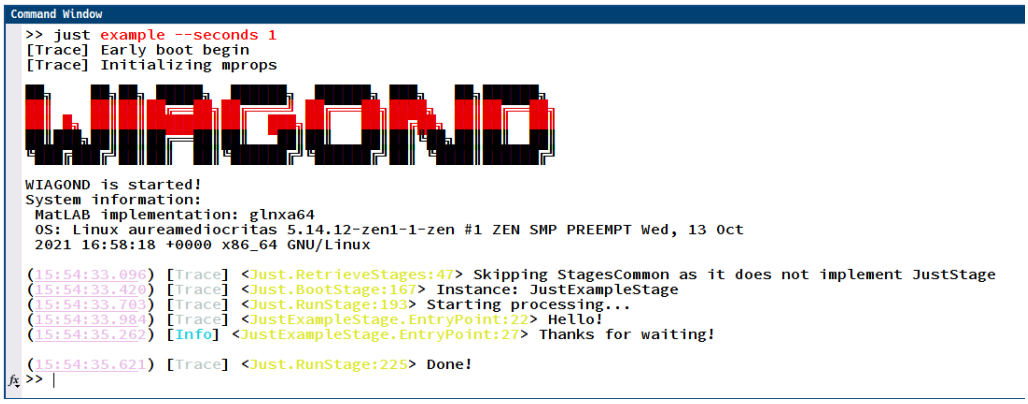

Рисунок 3.1 – Виведення кольорових повідомлень у складі консольного вікна MATLAB за допомогою стайлеру

Центральною ідеєю у всьому модулі логування є, безумовно, простота використання. Кінцевий розробник стадій не повинен піклуватися про те,

який саме стайлер використовується або чи було вимкнено логування інформаційних повідомлень, для нього все це не є важливою інформацією, над якою він повинен турбуватися.

Іншою важливою складовою контексту виконання є забезпечення функціоналу конфігураційних файлів. Рано чи пізно, будь-який із застосунків буде полягатися на велику кількість фактів та налаштувань, що не мають бути розподілені по великій кількості сирцевих кодів. З ціллю вирішення цієї проблеми, як правило, користуються конфігураційними файлами — файлами із спеціальним розширенням та форматуванням, що дозволяють централізовано визначити ключові або мінорні параметри застосунку.

Наприклад, доволі незручним є зберігання вже відомої константи K у алгоритмі кількісної оцінки захищеності стеганоповідомлення у безпосередньому місці його використання, а це декілька рівнів сирцевих файлів. Така практика носить назву hardcode, тобто полягається у вбудовуванні дані безпосередньо у вихідний код програми чи іншого виконуваного об’єкта, на відміну від отримання даних із зовнішніх джерел або їх генерування під час виконання. Така практика не є припустимою через складність модифікації як розробникам, так і кінцевим користувачам.

Набагато краще буде створити єдине місце конфігурації у складі конфігураційного файлу, що буде детермінувати усі гнучкі параметри та дозволятиме швидко їх змінювати, навіть без знань у області програмування. Таке розділення є класичним та загальноприйнятим у світі інформаційних технологій і програмного забезпечення і безумовно являє собою гарну практику програмування та розділення відомостей програмного застосунку на зовнішні та внутрішні.

У якості формату конфігураційного файлу було обрано модифіковану версію текстових файлів з розширенням .properties, які містять рядки у вигляді пар ключів і значень, до яких можна отримати доступ та відтворити на етапі виконання застосунку. Файли типу .properties є доволі популярними у середовищі програмування Java.

Саме за допомогою конфігураційного файлу вдалося виділити реалізацію за замовчуванням для серіалізації даних, визначити версію та ім’я застосунку, вказати стайлер та рівень логування за замовчуванням, визначити зерно ініціалізації генератору псевдовипадкових чисел, розмір блоку у стеганографічному методі Коха-Жао та багато інших гнучких параметрів, що можуть бути з легкістю налаштовані у одному текстовому файлі, без модифікації сирцевого коду застосунку.

Через велику кількість конфігураційних точок дотику сирцевого коду та конфігураційних файлів, також було розроблено відповідні механізми автоматичної ін’єкції значень конфігураційних полів за їх іменем. Це зменшує кількість можливих помилок, пов’язаних із неініціалізованим значенням змінної, а також покращує швидкість інтеграції застосунку з конфігураційними файлами. Реалізація такої можливості стала реальною насамперед через використання рефлексії динамічних типів, що у мові програмування MATLAB імплементовано із допомогою метакласів. Для автоматичного влучення значення конфігураційного параметру до складу поля певного об’єкту, достатньо позначити таке поле спеціальною інстанцією класу у якості значення за замовчуванням, а також викликати відповідний конструктор суперкласу. У цілому, таке використання конфігураційної складової в рази швидше за ручне управління конфігурацією.

Як вже було сказано раніше, у центрі уваги модуля Just стоїть об’єктно-орієнтований підхід, де кожна із стадій має у своєму складі чітку структуру, яка дозволяє добитися централізованості та уніфікованості стартової точки стадій. Прийоми абстрактного програмування інтерфейсів не допустить, наприклад, такої ситуації, коли одна із стадій не буде мати у своєму складі обов’язкового константного поля, що буде репрезентувати коротке ім’я стадії для її запуску.

Модульна сувора структура стадій, що підкріплюється об’єктно-орієнтованим підходом являє собою міцне ядро застосунку та розблоковує низку важливих впроваджень, що не є очевидними на перший погляд.

Наприклад, необхідною умовою для будь-якого файлу сирцевого коду, аби він був стадією виконання, є проста перевірка на імплементацію інтерфейсу стадії. Аналогічну ж операцію при використанні простих скриптів, ймовірніше за все, необхідно було б реалізовувати шляхом нав’язування особливої моделі іменування об’єктів, що накладає неочевидні складнощі на застосунок.

Можливість гнучкого налаштування параметрів стадій та всього виконання за допомогою конфігураційних файлів — не єдина можливість налаштувати поведінки стадії. У складі модуля контроля виконання Just також було реалізована можливість завдання аргументів командного рядку на кшталт класичних нативних програмних застосунків.

Такі параметри є найбільш гнучкими, та частіше всього, обов’язковими, що безпосередньо унікальні у рамках одного запуску. Виносити такі параметри хоча й можна до складу конфігураційного файлу, все одно краще було б сфокусувати увагу користувача на передачі таких важливих параметрів саме під час безпосереднього виклику.

Наприклад, під час безпосереднього запуску алгоритму оцінки захищеності стеганоповідомлення, тобто головної стадії застосунку, важливо також зазначити, яке саме очікуване збурення передбачається для протистояння та відповідного дослідження. Виносити такий важливий параметр до складу конфігураційного файлу не вірно з точки того, що такий параметр не є довготривалим, як, скажімо, розмір блоку у алгоритмі Коха-Жао, який де-факто дорівнює 8.

Вказувати аргументи командного рядку можливо із використанням короткої або довгої форми. Коротка форма містить лише один знак дефісу та перший символ параметру, в той час як довга форма містить два знаки дефісу та повну назву параметру. Безумовно, для спрощення процесу задання аргументів, їх порядок задання не має значення.

Кількість та характеристики відповідних аргументів командного рядку не потребує ручного задання на стадії програмування. Як правило, все, що

необхідно для декларування того, що певна стадія може брати на свій вхід певний аргумент, так це відповідним чином назвати один із параметрів конструктору стадії. За допомогою розумної рефлексії типів, модуль контрою виконання Just автоматично проаналізує конструктор стадії на предмет додаткових параметрів, вилучить їх, якщо вони були передані під час запуску стадії із командного рядку, та виконає запуск стадії з вже підставленими параметрами. Такий підхід неймовірно економить час програмування стадій та майже не залишає місця для помилок, пов’язаних із заданням аргументів командного рядку, адже для їх задання необхідно лише оголосити відповідний параметр у складі конструктору і це все.

3.3 Реалізація модуля єдиної області знань та кешу CompCache

Не менш важливим етапом імплементації практичної програмної частини роботи стало залучення особливого підходу до алгоритмічного викладення основної зони дії застосунку.

При стандартному підході до програмування великої за обсягом предметної області, що включає в себе декілька основних гілок та серію опціональних експериментальних гілок, існує велика ймовірність у накопиченні однотипного коду у складі декількох стадій, який важко підтримувати.

Маючи у своєму складі велику кількість кроків алгоритму, хотілося б бачити загальну картину обчислень та зв’язків конкретних кроків між собою. Під час класичного підходу стає проблемним задача обчислення лише потрібних кроків алгоритму, оминаючи, наприклад, інформаційні кроки порівняння старого та нового підходу до обчислення величини між двома власними векторами цифрового зображення. Необхідність ручного контролювання таких залежностей, конфліктів змінних та інших мінорних проблем сама по собі зменшує продуктивність роботи.

Не менш погіршуючим фактором стала потреба у постійному внесенні змін до сирцевого коду стадій із потребою у швидкому реагуванні обчислень на такі зміни. При класичному підході, як правило, необхідно із кожним запуском обчислювати купу розрахунків, що вже були попередньо обчислені, витрачаючи цінний час. Через саму характеристику предметної області та характеру сутностей, що досліджуються, навмисне зменшення вибірки зображень з ціллю економії часу обчислень, може погіршити результаті, що спостерігаються в результаті таких експериментів.

Таким чином, була очевидною потреба у створенні централізованого інструменту контролю виконання серії алгоритмічних перетворень над великою серією зображень, маючи ідею реального обчислення розрахунків у найбільш можливий оптимізований шлях.

Так було створено CompCache — модуль, що бере на себе відповідальність за розумне, наперед визначене та декларативно описане виконання основних обчислень над серією зображень, оголошуючи новий підхід до опису перетворень між тимчасовими кроками.

У центрі CompCache стоїть ідея того, що не має необхідності вручну виділяти схожі частини алгоритму, особливо коли мова йдеться про велику низку кроків, що можуть бути неоднорідно пов’язані між собою, для того, щоб використовувати таку область у декількох представленнях, що у цілому базуються на певному загальному зрізі від цієї області.

Модуль CompCache надає можливість описати всі операції, що можуть навіть теоретично виконуватися над певною сутністю, що досліджується. Фактично кажучи, це означає, що модуль, на відміну від класичного підходу, де для досягнення результатів очевидним є будування ланцюжку виконання відповідно до очікуваного результату, пропонує централізовано описати всю предметну область та всі можливі операції, що можуть мати місце бути.

Добре, але чи не виникатиме у такому випадку проблема того, що описуючи таку велику кількість відомостей, необхідною умовою для обчислення будь-якої з них, стане обчислення всіх відомих зв’язків та кроків?

Насамперед, ні. Центральною ідеєю у модулі CompCache стало те, що всі обчислення, кроки перетворень необхідно створювати із вказуванням того, від чого саме цей крок залежить. Така проста умова побудови дозволяє якісно оцінити зв’язки між етапами виконання та розглядати всю оголошену область лише під певним зрізом, що має цінність у конкретний момент часу.

Наприклад, описуючи у складі предметної області велику кількість додаткових кроків щодо побудови графіків, оцінки ефективності використання тієї чи іншої метрики, з’являється ситуація, коли все це обчислювати не є необхідним, потрібно лише слідкувати основному ходу алгоритму і обчислити відповідь у максимально простий и швидкий спосіб, не беручи до уваги введені додаткові кроки. При класичному підході це було б катастрофою, яка неминуче потребувала великої кількості змін до сирцевого коду застосунку, введення додаткових умов і тому подібного. При використанні CompCache — достатньо лише змінити селектор при побудові моделі предметної області, вказуючи, що саме являється кінцевим інтересом при цьому запуску.

І такий підхід дійсно спрощує програмування алгоритмів, кроки яких наперед нефіналізовані та навіть невідомі. Маючи лише єдину точку опису всієї предметної області, легко не допустити помилки при її проектуванні. Використовуючи просту методику опису кожного наступного кроку, де необхідно вказати необхідні залежності, ім’я та процедуру кроку, з’являється можливість побудувати повноцінний граф залежностей кроків, а це значить мати можливість візуалізувати та обходити такий граф, та в залежності від кінцевого селектору, обчислювати лише ті кроки, що необхідно для побудови кінцевого результату. Приклад візуалізації графу однієї з предметних областей із допомогою модуля CompCache приведено на рисунку 3.2.

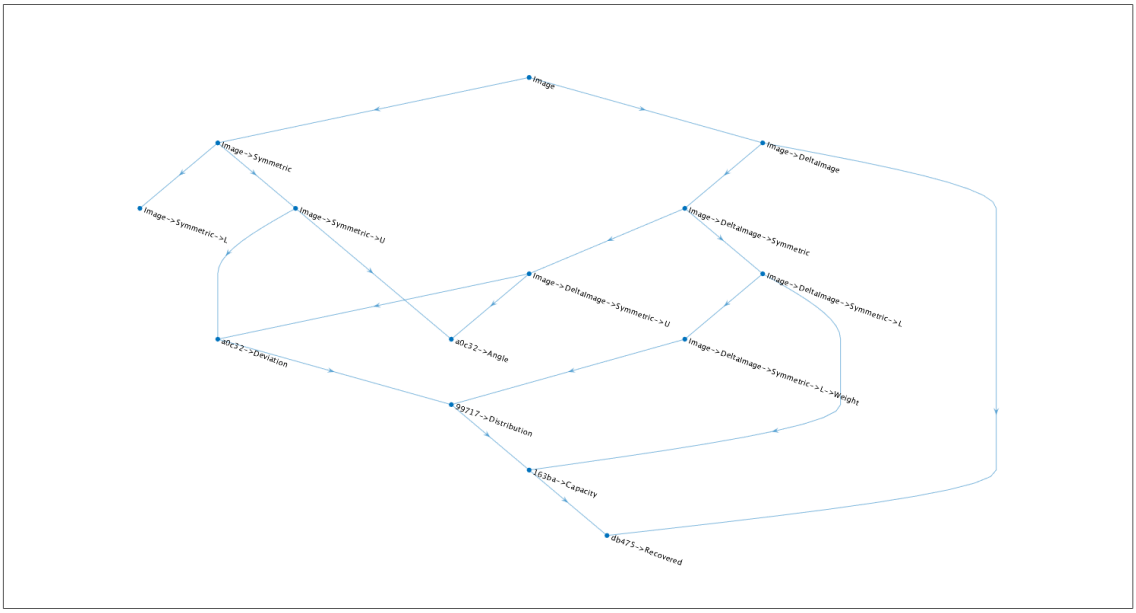

Рисунок 3.2 – Візуалізація графу предметної області, що побудований із допомогою модуля CompCache

Із неочевидних плюсів використання CompCache також варто виділити безпосередню орієнтованість на серію однотипних обчислень над певною серією сутностей. Про ручне використання циклів можна забути, кожна із процедур виконується індивідуально для кожного із елементів вибірки, створюючи легку для використання структуру, що містить у своєму складі всі залежні для цього елементу обчислення. Програмування кожної із процедур передбачається без турбування про те, що ця процедура повинна бути виконана для серії зображень, всі ці розбиття автоматично покриваються модулем CompCache. Крім того, такий централізований підхід до розгалуження роботи обчислень між елементами вибірки та процедурами дозволив за потребою організувати повноцінне розпаралелювання роботи програми за допомогою багатопотокового підходу, що дозволяє підвищити швидкість виконання у декілька разів, в залежності від кількості створених робітників та кінцевої потужності центрального процесору, на якому виконується застосунок.

Однак вирішення озвучених проблем не перекреслює існування іншої, що пов’язана із непотрібним обчисленням одних і тих же розрахунків у результаті внесення навіть мінімальних змін до складу сирцевого коду застосунку.

З метою вирішення цієї складної ситуації, модуль CompCache, як каже його назва Computation Cache, дозволяє створити і вдало використовувати в подальшому проміжний буфер із швидким доступом до нього з ціллю збереження копії часто використовуваних даних.

І хоча застосування такого кроку саме по собі не передбачає переосмислення вхідних даних застосунку, все ж з введенням такого потужного інструменту з’являється повноцінна можливість оголошувати потоки та конвеєри важких даних, не турбуючись про можливі проблеми обчислювального характеру, що можуть мати місце спостерігатися в результаті. Мова йдеться, наприклад, про повну відмову від необхідності поверхневого та передчасного передпроцессінгу та передобробки вибірки зображень, що полягає у зменшенні розміру зображень. Із введенням CompCache можна не виконувати цей крок вручну, і що більш важливо — контролювати, аби будь-які зміни в вибірці зображень своєчасно відображались, адже тепер завантаження таких даних набагато швидше, аніж обчислення.

У якості сховища за замовчуванням використовується так назване тимчасове сховище операційної системи. Наприклад, у операційній системі GNU/Linux таке сховище являє собою особливу файлову систему, що розташовується у оперативній пам’яті комп’ютера, а тому зчитування даних з такої області сховища є дуже швидким процесом.

І весь цей процес, як вже було попередньо сказано, повністю безшовний для кінцевого користувача застосунку. Якщо всі обчислення вже перебувають у стані обчислених попередньо, тобто існують у складі кешу, модуль CompCache зможе одразу їх завантажити до робочої області, користуючись методами десеріалізації, вбудованими у MATLAB. Однак це

не єдина умова того, як може скластися ситуація: модуль CompCache дозволяє без усіляких проблем знаходити із допомогою елементарних операцій над множинами модифікації до первинної вибірки, дозволяючи вносити модифікації до існуючої моделі кешу. Така можливість стала у нагоді при розрахунку великої кількості зображень (близько 5 тисяч) та необхідності майже мінімальних змін до складу вибірки (наприклад, змінити лише одне зображення). У цьому випадку модуль CompCache, користуючись ключовою інформацію щодо елементів вибірки (за замовчуванням, це час модифікації файлу), відслідкує модифікацію первинного кроку перетворень, виконає повну серію обчислень лише для одного з зображень та користуючись транзакціями тимчасового сховища, виконає модифікацію останнього на відповідність новим потребам.

Таким чином, середній час виконання застосунку при важливих експериментальних запусках кількісно зменшився, надаючи можливість без усіляких труднощів вносити мінімальні зміни до складу застосунку, не переймаючись про можливі затримки в обчислені результатів. Система кешу CompCache дозволяє без проблем видалити із складу сховища кешу будь-який із елементів-кроків, ставляючи за потребу обчислення лише втраченого елементу.

І хоча, можливо, у цілому, виконати програмування застосунку можливо було б і без використання модуля єдиної області знань та кешу CompCache, таке рішення безумовно б коштувало великої кількості часу та сил, створюючи потенційне місце виникнення неточностей, затримок та навіть помилок, що очевидно не є бажаними під час роботи. Створений модуль може бути використаний у додаткових або навіть несуміжних дослідженнях, не потребуючи модифікації кодової бази — модуль CompCache не орієнтований на лише певний алгоритм та може стати у нагоді при вирішення будь-якої складної алгоритмічної предметної області.

3.4 Модульне тестування застосунку

Під час побудови великих за своїм обсягом проектів, не менш важливим етапом дослідження працездатності, окрім саме тестового запуску, є також впевненість в тому, що всі програмні частини функціонують вірно та за своєю задумкою. Якщо у випадку невеликих за розміром скриптів, їх область використання, в силу своєї некомплексності, при виникненні неточностей або помилок, однозначно видає себе, про існування деяких проблем у великих проектах можна дізнатися лише при специфічній умові, що не завжди може бути досліджена через особливість умови або затратність часу.

З цією ціллю, як правило використовують модульне тестування — це процес розробки програмного забезпечення, в якому найменші частини програми, які можна перевірити, так звані блоки, окремо та незалежно перевіряються на процес належної роботи. Основна мета модульного тестування — виділити написаний код для перевірки та визначити, чи працює він належним чином.

Модульне тестування, як правило, виконується за допомогою коду для іншого коду, що дозволяє розробити умови перевірки для кожної, навіть мінорної зміни, що здавалося б, не може в ніякої степені пошкодити вже існуючі компоненти.

Важливість покриття функціональності застосунку тестами складає собою важливий процес розробки будь-якого, скільки-небудь важливого проекту та дозволяє безболісно розширювати та підтримувати велику кодову базу різних за розміром та застосуванням проектів.

У роботі, що розглядається, для організації модульного тестування було створено відповідну стадію у складі модуля контролю виконання стадій Just, а також використано фреймворк unittest у складі MATLAB. Останній являє собою потужний інструмент, що дозволяє створювати тести за

допомогою скриптів, функцій та класів, надаючи можливість доступу до інтерфейсів дослідження швидкодії, діагностики, кваліфікаторів, прив’язок та плагінів, дозволяючи покривати тестування скільки-небудь складні умови.

Особливої уваги заслуговує частина функціоналу, що присвячена створенню макету об'єкта, котра дозволяє ізолювати частину системи для тестування, імітуючи поведінку залежностей. Важливість використання такої практики пов’язано безпосередньо із тим, що час модульного тестування часто стоїть потреба в тестуванні частини цілісної системи, ізольованої від залежностей. Щоб перевірити частину системи, до використання рекомендується використовувати фіктивні об’єкти для заміни залежностей. Макетний об’єкт реалізує принаймні частину того ж інтерфейсу, що й виробничий об’єкт, але часто простіше, швидше, більш передбачувано чи більш керованим способом.

3.5 Реалізація зв’язку між інтерфейсом програми та модулем обчислень за допомогою рушія мови MATLAB Engine API

Під час вибору у якості ядра обчислення середовища і мови програмування MATLAB, а у якості графічного застосунку мови програмування Python разом із бібліотекою Qt, постало важливе питання щодо способу комунікації між цими повністю несумісними програмними середовищами.

На щастя, у складі такого комплексного середовища, як MATLAB, існує спеціальний набір інтерфейсів та відповідний рушій, що дозволяє виконувати код MATLAB із інших середовищ. Таким чином, надається можливість додаткам, що працюють за межами MATLAB, працювати з даними MATLAB через MATLAB-нейтральний інтерфейс.

Рушій мови MATLAB Engine API підтримує об’єктно-орієнтоване програмування та асинхронне виконання. Цей API дозволяє програмам,

виконаним на інших мовах програмування, запускати MATLAB, виконувати функції MATLAB з аргументами та обмінюватися даними між програмами.

Офіційно виділяють дві ревізії рушія: версія MATLAB Engine API для мови C++ [27] та версія MATLAB Engine API для мови Python [26]. У цілому, версію для мови C++ можна адаптувати під більшість інтерпретованих мов програмування через нативні прив'язки виконані шляхом розширюваних бібліотек.

Механізм MATLAB Engine API у складі пакету для мови програмування Python надає функції для виклику MATLAB, а класи масивів надають функції для створення масивів MATLAB як об'єктів Python. За допомогою пакету стає можливим створювати масиви MATLAB у Python, викликаючи відповідні конструктори типу масиву (наприклад, matlab.double для створення масиву типу double). Масиви MATLAB можуть бути вхідними аргументами для функцій MATLAB, викликаних за допомогою двигуна. Схематично представлення комунікації між цими двома середовищами показано на рисунку 3.3.

Така інтеграція між двома несхожими за своєю структурою середовищами, дозволила легко обмінюватися повідомленнями між важливими частинами застосунку у майже нативний спосіб: всі операції серіалізації та приведення типів виконуються прозоро для кінцевого розробника.

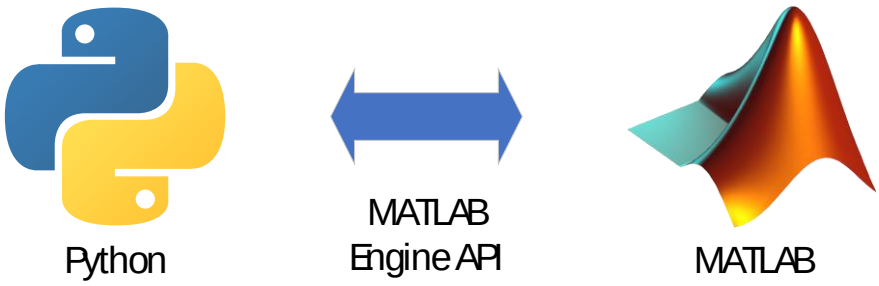

Рисунок 3.3 – Схема інтеграції Python з MATLAB шляхом використання MATLAB Engine API

Неочевидним плюсом використання рушія MATLAB Engine API також стало те, що користувачеві, за бажанням, надається можливість використовувати вже готову інстанцію MATLAB для подальшого налагодження або контролю над виконанням. Безумовно, цей крок не є обов'язковим; за замовчуванням, створюється повністю ефемерна сесія MATLAB, цикл життя котрої лімітований циклом життя застосунку, однак така неочевидна можливість все ще носить корисну функцію.

3.6 Інструкція з експлуатації програмного продукту «Wiagond»

Для розв'язання задачі кількісної оцінки захищеності стеганоповідомлення було розроблено програмний продукт, що носить назву «Wiagond». Інтерфейс програмного застосунку представлено на рисунку 3.4.

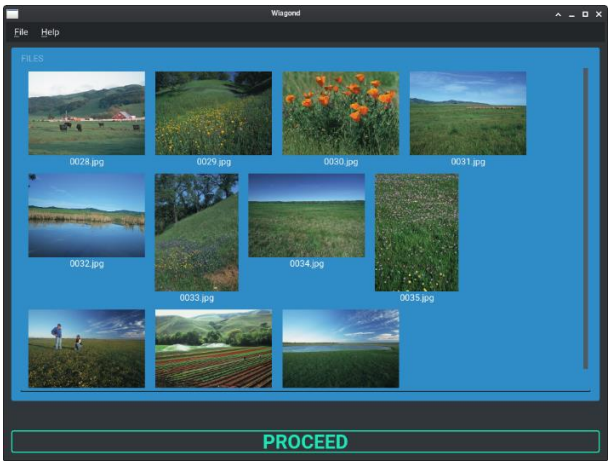

Рисунок 3.4 – Інтерфейс програмного продукту «Wiagond»

Основна ідея спілкування кінцевого користувача з програмним застосунком полягає в забезпеченні максимальної простоти взаємодії з програмним продуктом — таким чином, утиліта буде зрозумілою для користування як судовим експертам, так рядовим користувачам. Крім того, у випадку необхідності дослідження великої кількості зображень, із великою кількістю повторень, важливо не насити графічний інтерфейс програмного продукту зайвою кількістю інформаційних вікон і/або віджетами.

Роботу із застосунком можна умовно поділити на наступні частини:

— вітання та ініціалізація підключення до рушія MATLAB Engine API;

— вибір зображень для оцінки;

— вибір очікуваних збурень над контейнером;

— отримання результатів обчислення.

Вітання із застосунком виконується виведенням простого інформаційного вікна в тривіальною кнопкою-стрілкою підтвердженням, що показано на рисунку 3.5.

Після цього з’являється вікно ініціалізації підключення до рушія MATLAB Engine API. Як вже було сказано попередньо, у застосунку імплементована можливість використання вже існуючої сесії MATLAB, але також існує можливість створення повністю незалежної інстанції. Після вибору необхідної опції, кнопка-стрілка розблоковується та надає можливість переходити до наступного кроку. Це діалогове вікно приведено на рисунку 3.6.

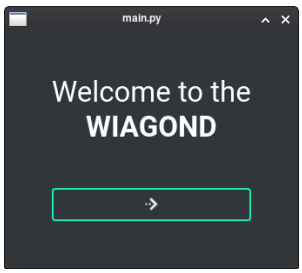

Рисунок 3.5 – Діалогове вікно вітання застосунку із користувачем

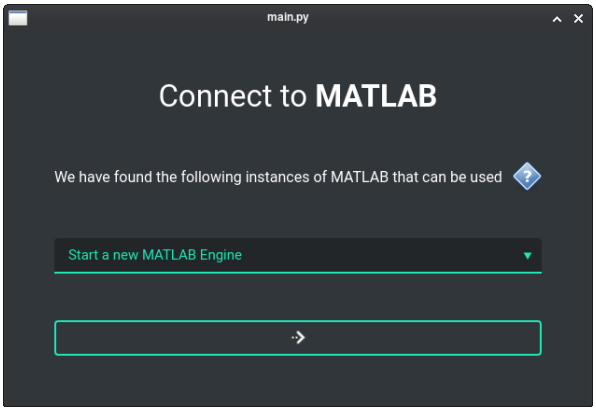

Рисунок 3.6 – Діалогове вікно ініціалізації сесії MATLAB

У випадку успішного з’єднання із рушієм мови MATLAB, виводиться тривіальне інформаційне вікно, ціль котрого — проінформувати користувача про те, що всі етапі підготовки були успішно пройдені. Вигляд такого діалогового вікна приведено на рисунку 3.7.

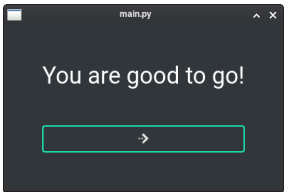

Рисунок 3.7 – Інформаційне вікно про успішність з’єднання з рушієм MATLAB

Наступним кроком увазі користувача виводиться головне вікно програми, в якому необхідно безпосередньо обрати ті зображення, що виступають у якості потенційних контейнерів та потребують кількісної оцінки захищеності інформації. Для цього можна скористатися контекстним меню «File», де необхідно буде обрати пункт «Add Files», що продемонстровано на рисунку 3.8, або ж скористатися більш інтуїтивно зрозумілим шляхом — за допомогою файлового менеджеру виконати процес перетаскування зображень у робочу область програмного застосунку, що продемонстровано на рисунку 3.9.

Також у складі контекстного меню є можливість вибору відомостей про програмний продукт, за допомогою секції «Help» та пункту «About», де виводиться інформація про назву, версію, походження та відкриту ліцензію програмного продукту.

Якщо вибір користувача щодо потенційних контейнерів було зроблено, йому необхідно натиснути кнопку «PROCEED» для переходу до наступного етапу.

На наступному етапі, користувачеві необхідно обрати серію модифікацій, проти яких стеганоповідомлення повинно вистояти. Увазі користувача представлено вибір трьох, найбільш популярних збурень, а саме

стистення JPEG, накладання шуму Гауса та мультиплікативного шуму, однак, як вже було визначено попередньо, алгоритм принципово не фокусується лише на них, представлені алгоритми модифікації, як правило, не псують візуальну складову зображення, у цілому алгоритм оперує модифікаціями як певним математичним явищем збурення, не прив'язуючись лише до певних з них. Діалогове вікно вибору очікуваних модифікацій приведено на рисунку 3.10.

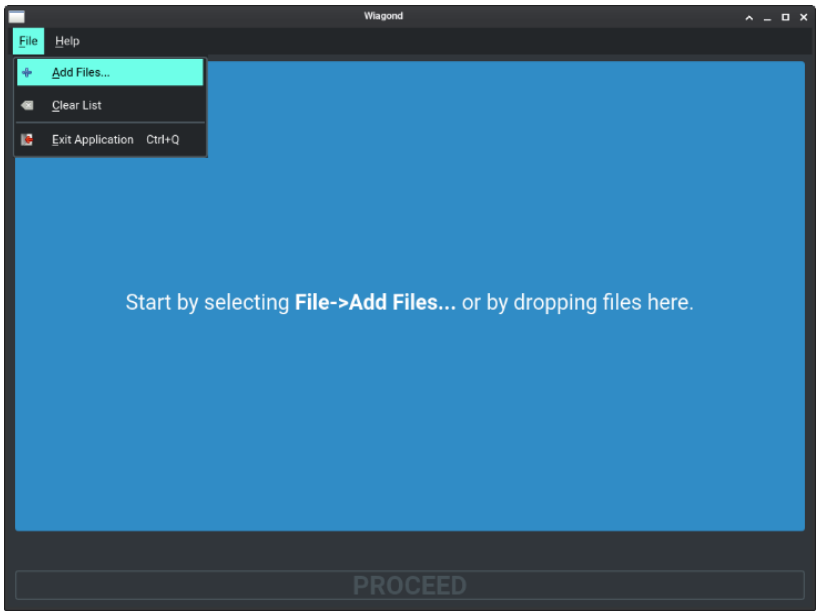

Рисунок 3.8 – Задіяння функціоналу додавання файлів за допомогою контекстного меню застосунку

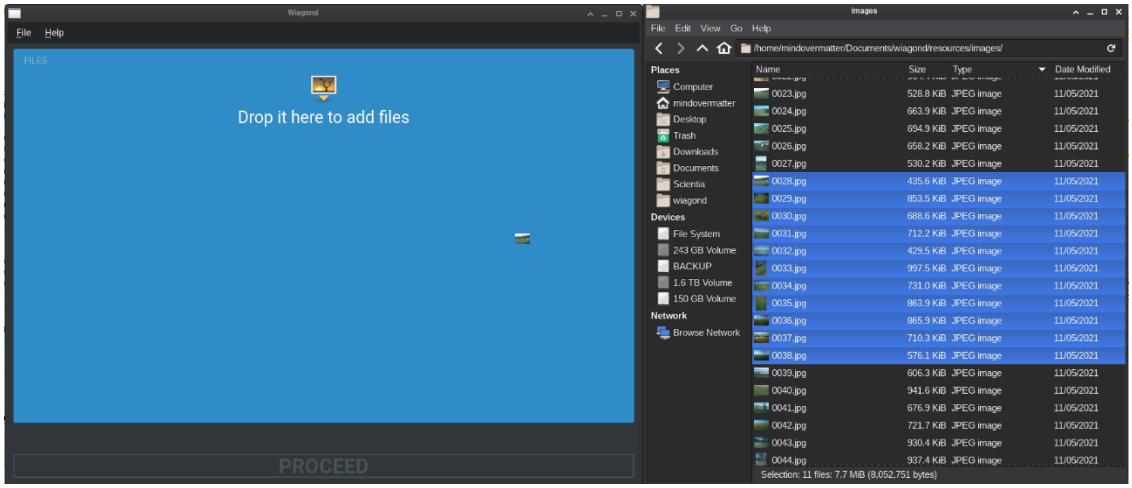

Рисунок 3.9 – Задіяння функціоналу додавання файлів за допомогою перетягування файлів із області файлового менеджеру

До речі, користувач може обрати всеможливі комбінації цих збурень, наприклад, виконати стиснення за допомогою JPEG декілька разів підряд, тобто перед користувачем постає величезна кількість комбінацій очікуваних збурень.

Якщо користувач обрав необхідні для нього модифікації, за допомогою кнопки «RUN!» виконується перехід до наступного етапу виконання.

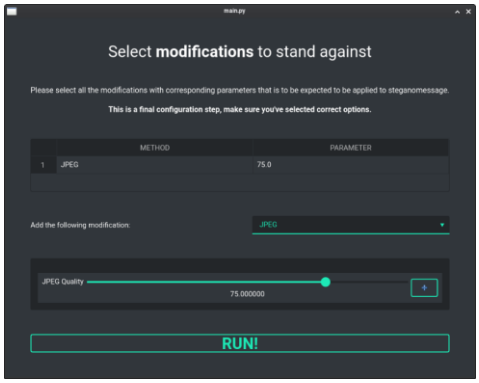

Рисунок 3.10 – Діалогове вікно вибору очікуваних модифікацій

Після цього наступає етап обчислення кількості захищеності інформації, тривалість котрого, насамперед, визначається обчислювальною потужністю комп’ютера, на якому виконується застосунок. Звісно, такий етап неможливо пропустити, він не потребує ніякої інтеграції з користувачем, але являє собою важливу частину процесу. Діалогове вікно процесу виконання обчислень приведено на рисунку 3.11.

Рисунок 3.11 – Діалогове вікно процесу виконання обчислень

Нарешті, фінальним етапом роботи застосунку, є виведення діалогового вікна з результатами знаходження кількості захищеності інформації для кожного з поданих на вхід застосунку контейнерами. У результуючій таблиці користувач може обирати рядки таблиці та тим самим бачити відображення того чи іншого зображення у віджеті зліва для легкості результатів. Особливим рядком являється переможець з найбільшою кількістю захищеності, такий рядок виділений особливим шрифтом для очевидної наглядності, що приведено на рисунку 3.12.

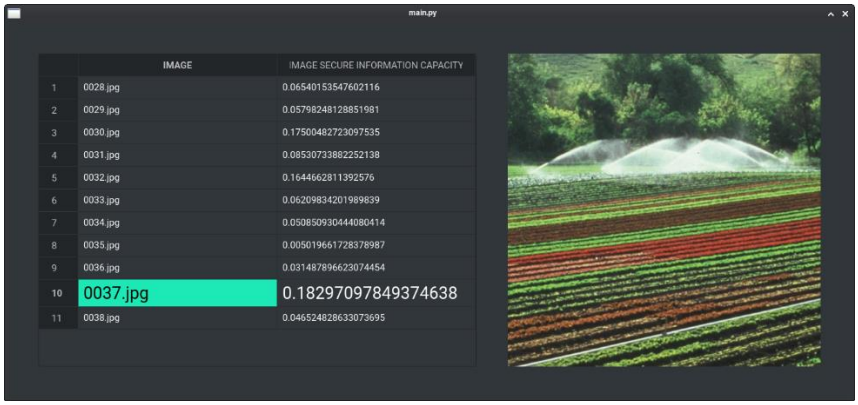

Рисунок 3.12 – Діалогове вікно результатів обчислень

Код програми міститься в додатку А.

В даному розділі представлена практична реалізація методу кількісної оцінки захищеності стеганоповідомлення.

Детально описано вибір засобів реалізації програмного продукту. У якості мови програмування для графічного застосунку було обрано мову Python з бібліотекою-біндингом PySide, а у якості мови програмування, що використовується для створення ядра обчислення кількісної оцінки захищеності стеганоповідомлення, після детального обзору популярних програмних середовищ, було обрано мову програмування MATLAB. MATLAB — це серйозний програмний продукт для інженерів та дослідників, що розробляється комерційною компанією MathWorks та дозволяє здійснювати маніпуляції з матрицями, будувати графіки функцій і даних,

реалізовувати алгоритми, створювати інтерфейси користувача та взаємодіяти з програмами, написаними на інших мовах програмування.

Представлена розробка модуля контролю виконання стадій Just.

Для вирішення проблеми відсутності централізованого контролю над виконанням програмного коду, підтримки великих за своїм розміром проектів, легкого конфігурування та можливості логування інформації, що виводиться застосунком, була розроблено спеціальний модуль для забезпечення комплексного контролю над виконанням застосунку. У центрі модуля зосереджено поняття стадій, що реалізуються із використанням об’єктно-орієнтованого підходу. Надання функціональної можливості доступу до системи логування та системи конфігурації вдалося можливим із введенням контексту виконання. Реалізовано функціонал передачі параметрів командного рядку до стадії шляхом автоматичного аналізу конструктору відповідних класів за допомогою рефлексії та метакласів.

Представлена розробка модулю єдиної області знань та кешу CompCache.

Впроваджена ідея альтернативного підходу до програмування предметної області, яка передбачає централізований та уніфікований процес опису зв’язків між усіма кроками алгоритму та безпосередньо процес виконання, який полягає у виборі бажаного кінцевого результату, згідно з яким з усієї області знань, за допомогою теорії графів, обираються лише ті кроки, котрі необхідні для забезпечення результату. Реалізовано функціонал збереження копій обчислень у тимчасовому сховищі операційної системи для забезпечення можливості виконання швидкої десеріалізації при рекурентних експериментах.

Створена програмна реалізація модульного тестування застосунку.

Для організації функціонування модульного тестування було створено відповідну стадію у складі модуля контролю виконання стадій Just, а також використано фреймворк unittest у складі MATLAB. Описано одну з ситуацію ізоляції процесу тестування з метою перевірки працездатності частини

системи за допомогою фіктивних об’єктів-макетів для заміни реалізації залежностей до більш передбачуваної і керованої.

Розглянута реалізація системи зв’язку між інтерфейсом програми та модулем обчислень за допомогою рушія мови MATLAB Engine API.

Механізм MATLAB Engine API у складі пакету matlab для мови програмування Python надає функції для виклику MATLAB. За допомогою цього пакету стає можливим створити сесію інтерпретатору, створювати та використовувати об’єкти мови програмування MATLAB та Python як взаємозамінні поняття. Рушій MATLAB Engine API вміє прозоро для користувача пакету виконувати необхідні приведення типів даних та серіалізацію.

Розглянута інструкція з експлуатації програмного продукту «Wiagond».

Основною ідеєю, з якою будувався графічний застосунок програмного продукту, стала інтуїтивна зрозумілість, мінімалістичність та простота. Результати оцінки захищеності контейнерів подаються із використанням табличного представлення.

4 ОХОРОНА ПРАЦІ І БЕЗПЕКА В НАДЗВИЧАЙНИХ СИТУАЦІЯХ

Аналіз умов праці і вибір заходів і засобів захисту від небезпечних і шкідливих виробничих факторів

Людина та її здоров’я — найбільша цінність держави, яка докладає великих зусиль, створюючи умови безпечної життєдіяльності людини як у середовищі мешкання, так і в середовищі праці. Державна політика в галузі охорони праці ґрунтується на головному принципі, згідно з яким життя і здоров'я працівників мають пріоритет над будь-якими результатами виробничої діяльності.

Стаття 43 Конституції України закріплює, що кожен має право на належні, безпечні і здорові умови праці [1]. Згідно з статтею 153 Кодексу законів про працю України, на всіх підприємствах, в установах, організаціях повинні бути безпечні і нешкідливі умови праці, а забезпечення таких умов покладається на власника [28]. Нарешті, Закон України «Про охорону праці» чітко визначає, що умови праці на робочому місці, безпека технологічних процесів, машин, механізмів, устаткування та інших засобів виробництва, стан засобів колективного та індивідуального захисту, що використовуються працівником, а також санітарно-побутові умови повинні відповідати вимогам законодавства [29].

Головними елементами робочого місця розробника програмного забезпечення є письмовий стіл, крісло та персональний комп’ютер, допоміжними ж елементами є шафи, полиці, периферійні пристрої.

У робочому приміщенні на розробника програмного забезпечення можуть негативно діяти наступні фізичні шкідливі виробничі фактори:

– підвищена і знижена температура, вологість і рухливість повітря;
– рівень шуму, який перевищує допустимі норми;
– підвищений рівень електромагнітних полів;
– надмірна запиленість повітря;
– небезпека ураження електричним струмом;

– нерівномірність розподілу яскравості у полі зору.

До психофізіологічних небезпечних та шкідливих виробничих факторів, що впливають на розробника програмного забезпечення протягом його робочої зміни можна віднести наступні:

– нервово-емоційні перевантаження;
– перенапруження аналізаторів;
– монотонність праці;
– необхідність обробки великого обсягу інформації у одиницю часу.

Організація робочого місця передбачає:

– коректне розміщення робочого місця у приміщенні;
– раціональну компоновку обладнання на робочих місцях;
– вибір ергономічно обґрунтованого робочого положення, виробничих меблів з урахуванням антропометричних характеристик людини;
– урахування характеру та особливостей трудової діяльності.

НПАОП 0.00-1.31-99 регламентує вимоги до організації робочого місця користувача персонального комп’ютера. Бажано розмістити робочі місця з моніторами рядами, при цьому відносно вікон вони повинні розміщуватися так, щоб природне світло падало збоку, переважно зліва. Це дасть змогу виключити дзеркальне відбиття на моніторі джерел природного світла та потрапляння таких в поле зору розробників програмного забезпечення [30].

Площа, виділена для одного робочого місця з ПК, повинна складати не менше 6 $м^2$, а об'єм – не менше 20 $м^3$. При розміщенні робочих місць необхідно дотримуватись таких вимог:

– робочі місця з ПК розміщуються на відстані не менше 1 м від стін зі світловими прорізами;
– відстань між бічними поверхнями ПК має бути не меншою за 1,2 м;
– відстань між тильною поверхнею одного ПК та екраном іншого не повинна бути меншою 2,5 м;
– прохід між рядами робочих місць має бути не меншим 1 м.

Встановлено, що електромагнітні поля негативно впливають на організм людини. Основним джерелом таких випромінювань для розробників програмного забезпечення, що використовують у своїй роботі автоматизовані інформаційні системи на основі персональних комп'ютерів, є монітори.

Персональний комп’ютер є джерелами таких випромінювань як:

– утрафіолетове випромінювання в межах довжини хвиль 200-400 нм;
– видиме випромінювання в межах довжини хвиль 400-700 нм;
– ближнє інфрачервононе випромінювання в межах довжини хвиль 700-1050 нм;
– електромагнітне випромінювання радіочастотного діапазону ультрависокими та надвисокими частотами з діапазоном 300-3000 МГц та 3-30 ГГц відповідно;

Ультрафіолетове випромінювання може бути корисним в невеликих дозах, але у великих кількостях призводить до головного болю, дерматиту шкіри, різі в очах. Інфрачервоне випромінювання призводить до перегріву тканин людини, головного болю та ознак запаморочення.

У роботі розробника програмного забезпечення має місце бути небезпека високого рівня напруженості електромагнітного поля. На відстані 5-10 см від екрана і корпусу персонального комп’ютера рівень напруженості може сягати 140 В/м, що значно перевищує допустимі значення державних санітарних правил і норм роботи з персональними комп’ютерами [31].

Для попередження експлуатації небезпечної техніки всі дисплеї повинні відповідати вимогам безпеки (наприклад, міжнародний стандарт MRP II або група стандартів TCO: TCO’92, TCO’95, TCO’99, TCO’01, TCO’03, TCO’04, TCO’06, TCO’5.1 тощо).

Негативний вплив електромагнітного випромінювання можна зменшити шляхом використання екрануючих матеріалів та засобів індивідуального захисту. Екранування електромагнітних полів насамперед ґрунтується на двох основних фізичних властивостях – відбиванні і поглинанні електромагнітних хвиль при переході з одного стану в інший.

Обидва ці явища знижують енергію електромагнітної хвилі, що проходить крізь екран. Найчастіше в якості центрального матеріалу захисного екрану використовується певний провідник. Такі екрануючі матеріали містять в своєму складі металеві волокна (мідні, срібні чи стальні). Такі волокна утворюють сітку, що і служить захисним екраном.

Таким чином, електромагнітна хвиля під час взаємодії з екраном частково відбивається від його поверхні, частково проникає в стінку екрана та зазнає там відповідного поглинання, декілька разів відбивається від його стінок і частково проникає в екрануючу область. При цьому всі перераховані вище процеси, безумовно, супроводжуються втратами енергії електромагнітної хвилі, а це значить, що вони ослаблюють її дію.

Серед індивідуальних засобів захисту від електромагнітних полів виділяють такі, як екрануючий одяг, екрануючі головні убори, сорочки та легінси. Для виготовлення всіх моделей екрануючого одягу використовуються високоякісні металовмісні екрануючі тканини, що ослаблюють дію електромагнітних полів.

Аналіз техногенних небезпек і вибір заходів і засобів забезпечення безпеки у надзвичайних ситуаціях

Техногенна безпека — відсутність ризику виникнення аварій та/або катастроф на потенційно небезпечних об’єктах, а також у суб’єктів господарювання, що можуть створити реальну загрозу їх виникнення. Техногенна безпека характеризує стан захисту населення і територій від надзвичайних ситуацій техногенного характеру.

Аварією називають вихід з ладу машин, механізмів, пристроїв, комунікацій, споруд внаслідок порушення технології виробництва, правил експлуатації, правил безпеки, помилок, які допущені при проектуванні, будівництві, а також внаслідок стихійних лих.

Види аварій, які зустрічаються найчастіше:

– аварії з витоком сильнодіючих отруйних речовин (аміаку, хлору, сірчаної та азотної кислот, чадного газу, сірчаного газу та інших речовин);
– аварії з викидом радіоактивних речовин у навколишнє середовище;
– пожежі та вибухи;
– аварії, пов'язані з використанням транспортних засобів.

Згідно з статистичними даними, найбільш імовірною техногенною небезпекою для приміщень, де працюють розробники програмного забезпечення, є пожежі.

Основними причинами виникнення пожеж у приміщеннях розробників ПЗ варто виділити:

– незадовільний стан електротехнічних пристроїв та порушення правил їх монтажу та експлуатації;
– технологічний пил, що за своєю структурою є струмопровідним і накопичується в такій кількості, що він може осідати на проводах, проникати всередину машин, апаратів тощо;
– експлуатація пошкоджених розеток, роз’ємів та форм-факторів;
– перевантаження електромережі шляхом підключення зайвої побутової техніки (мікрохвильові печі, обігрівачі тощо), або ігнорування параметрів максимально допустимого струму електричних приборів (трійників, мережевих фільтрів тощо);
– пошкодження системи заземлення електротехнічних пристроїв;
– порушення правил влаштування та експлуатації приладів опалення.

Підвищення надійності і безпечності електроустановок реалізується шляхом виконання Правил улаштування електроустановок (ПУЕ), що визначають будову, принципи улаштування, особливі вимоги до окремих систем, їх елементів, вузлів і комунікацій електроустановок. Усі робочі місця з персональними комп’ютерами та пов’язаними периферійними пристроями, а також принтери, серверне обладнання і побутові прилади необхідно забезпечити відповідним обладнанням стабілізаторів напруги, пристроями

захисного відключення та обмежувачами імпульсних перенапруг, кнопковими постами контролю. Безумовно, до використання також необхідно залучити захисне заземлення, занулення та системи захисного вимкнення, разом із комплексом пристроїв для захисту від атмосферної електрики, а також огороджувальні електрозахисні засоби.

З метою забезпечення вимог пожежної безпеки, в усіх приміщеннях розміщують первинні засоби пожежогасіння, серед яких найбільш зручними для використання в умовах офісу є вогнегасники. Допускається використовувати порошкові, вуглекислотні та аерозольні вогнегасники. Враховуючи те, що в приміщеннях багато апаратури, приладів та документів, щоб запобігти їх псуванню при гасінні, краще користуватись вуглекислотними вогнегасниками.

На 20 м² площі підлоги в офісних приміщеннях з оргтехнікою, слід передбачати по одному вуглекислотному вогнегаснику з величиною заряду вогнегасної речовини 3 кг і більше.

Вогнегасники слід розташовувати наступним чином: вони мають бути захищені від впливу прямих сонячних променів, теплових потоків, механічних впливів і інших несприятливих факторів (вібрація, агресивне середовище, підвищена вологість та інші). Крім того, вогнегасники повинні бути добре видимими і легкодоступними в разі пожежі та не повинні перешкоджати евакуації людей. Купувати вогнегасники необхідно лише в спеціалізованих організаціях, що ліцензовані на такий вид діяльності і продукція яких сертифікована в Україні. Для гасіння пожежі в початковій стадії доречно мати ще кошму. Пожежні покривала повинні бути розміром не менше ніж 1х1 м.

Усі працівники повинні бути освідомленні у тому, як користуватися первинними засобами пожежогасіння, а раз на півроку повинні проводитись практичні заняття, на яких персонал поновлює знання та відпрацьовує навички на випадок пожежі.

Розрахунок величини опору штучного захисного заземлення

Основним технічним засобом захисту в системі електробезпеки в приміщеннях із обчислювальною технікою являється захисне заземлення, яке забезпечує захист людей від ураження електричним струмом при дотиканні до металевих неструмопровідних частин, які можуть опинитися під напругою, як правило, внаслідок пошкодження ізоляції.

Вихідні дані:

– Напруга електроустановки 220 В;
– Потужність джерела живлення мережі – більше 100 кВА;
– Розміщення електродів — "в ряд";
– Вертикальний заземлювач: труба, діаметр d = 0,02 м, довжина l = 3м;
– Відношення відстані між електродами A до їх довжини l, $A{:}l$ = 1:3;
– Глибина траншеї $t_0 = 1$ м;
– Горизонтальний електрод – смуга сталева, ширина b = 0,08 м

Нормативне значення опору заземлювального пристрою, згідно із характеристикою напруги електроустановки до 1000 В та потужності трансформатора більше 100 кВА, R_d = 4 Ом.

Для другої кліматичної зони, в якої знаходиться Одеська область, розрахунковий опір ґрунту:

$$\rho_{розрах} = \rho_{пит} \cdot \varphi = 20 \cdot 1,45 = 29 \text{ Ом} \cdot \text{м}$$

Заглиблення заземлення:

$$t = t_0 + \frac{l}{2 \cdot l} = 1 + \frac{3}{2 \cdot 3} = 1,5 \text{ м}$$

Опір захисного заземлення з одиночним вертикальним заземлювачем:

$$R_d = \frac{\rho_{розрах} \cdot \left(\ln \frac{2 \cdot l}{d} + \frac{1}{2} \cdot \ln \frac{4 \cdot t + 1}{4 \cdot t - 1} \right)}{2\pi \cdot l} = \frac{29 \cdot \left(\ln \frac{6}{0.02} + \frac{1}{2} \cdot \ln \frac{7}{5} \right)}{18,85} = 8,78 \text{ Ом}$$

Таким чином, фактичний опір захисного заземлення з одиночним вертикальним заземлювачем не забезпечує надійного захисту персоналу від ураження електричним струмом при короткому замиканні на корпус

електроустановки. Тому існує необхідність модернізувати існуючу систему захисного заземлення, для чого необхідно виконати наступні розрахунки [32].

Число вертикальних заземлювачів:

Приймаючи $\eta_в = 1,\ n = \frac{R_d}{R_Д} \cdot \eta_в = \frac{8,78}{4} = 2,195$ шт.

Для $n \approx 2,\ \frac{A}{l} = 3,$ при розміщенні заземлення "в ряд", $\eta_в = 0,94$

Тоді $n = \frac{8,78}{4} \cdot 0,94 = 2,06$ шт.

Приймаємо n = 3 шт.

Довжина горизонтальної смуги при розміщенні заземлення "в ряд":

$$L = \frac{A}{l} \cdot l \cdot (n-1) = 3 \cdot 3 \cdot (3-1) = 18 \text{ м.}$$

Для горизонтальної смуги 18 м кліматичний коефіцієнт $\Psi_1 = 1,25$.

Діаметр горизонтального електрода:

$$d_1 = \frac{1}{2} \cdot b = \frac{1}{2} \cdot 0.08 = 0,04 \text{ м.}$$

Опір горизонтального електрода:

$$R_в = \frac{\rho_{розрах}}{2\pi \cdot L} \cdot \ln \frac{L^2}{d_1 \cdot t} = \frac{29}{2\pi \cdot 18} \cdot \ln \frac{18^2}{0,04 \cdot 1,5} = 2,2 \text{ Ом}$$

Для $\frac{A}{l} = 3, n = 3$ і розміщенні заземлення "в ряд", $\eta_г = 0,96$

Загальний розрахунковий опор заземлювального пристрою:

$$R_{заг} = \frac{R_в \cdot R_d}{R_в \cdot \eta_в + R_d \cdot \eta_г \cdot n} = \frac{2,20 \cdot 8,78}{2,20 \cdot 0,94 + 8,78 \cdot 0,96 \cdot 3} = 0,7 \text{ Ом, що} < R_d = 4 \text{ Ом.}$$

Загальний опір захисного заземлення $R_{заг}$, що складається з 3 вертикальних заземлювачів діаметром 0,02 м, довжиною 3 м, розташованих «в ряд», з'єднаних горизонтальним електродом у вигляді смуги шириною 0,08 м, довжиною 18 м, дорівнює 0,7 Ом, забезпечує надійний захист персоналу від ураження електричним струмом при замиканні на корпус електроустановки.

ВИСНОВКИ

В результаті кваліфікаційної роботи вирішено важливу науково-практичну задачу удосконалення методу кількісної оцінки захищеності стеганоповідомлення для забезпечення можливості вибору цифрового зображення-контейнера, що породжує стеганоповідомлення, нечутливе до збурень. Запропонований удосконалений метод не залежить від використовуваного стеганографічного алгоритму та конкретики застосовуваних збурень.

Отримані наступні результати:

1) Запропоновані теоретичні основи удосконалення методу кількісної оцінки захищеності стеганоповідомлення:

2) Теоретично обґрунтовано та практично підтверджено неінформативність та шкідливість урахування при розрахунку обсягу захищеної в стеганоповідомленні додаткової інформації абсолютних відокремленостей власних значень матриці цифрового зображення, що привело до усунення власних значень з множини параметрів, збурення яких враховується при кількісній оцінці захищеності стеганоповідомлення;

3) Запропонована нова кількісна оцінка збурення власного вектору матриці цифрового зображення в результаті стеганоперетворення, яка дає змогу для відокремлення чутливого власного вектору від нечутливого, адекватно відображає збурення вектору незалежно від величини кута повороту, що дало можливість для удосконалення формули для розрахунку обсягу захищеної додаткової інформації;

4) Запропонована нова умова вибору захищених власних векторів, що враховує індивідуальні характеристики контейнера, замість попереднього обчислення над певною наперед визначеною, а отже небездоганною вибіркою зображень, що дозволяє враховувати особливості кожного ЦЗ-контейнера.

5) Розроблено та реалізовано програмний продукт для оцінки захищеності контейнерів на протидію певному збуренню, яке очікується. Під час розробки: обґрунтовано вибір засобів реалізації програмного продукту (програмні середовища MATLAB, Python); розроблено модуль контролю виконання стадій Just, спеціальний модуль для забезпечення комплексного контролю над виконанням застосунку; розроблено модуль єдиної області знань та кешу CompCache; впроваджена ідея альтернативного підходу до програмування предметної області, яка передбачає централізований та уніфікований процес опису зв'язків між усіма кроками алгоритму та безпосередньо процес виконання, що за допомогою теорії графів обирає для обчислення лише потрібні для результату кроки; реалізовано функціонал збереження копій обчислень у тимчасовому сховищі операційної системи для забезпечення можливості виконання швидкої десеріалізації при рекурентних експериментах; створена програмна реалізація модульного тестування застосунку з підтримкою фіктивних залежностей модулів; розглянута реалізація системи зв'язку між інтерфейсом програми та модулем обчислень за допомогою рушія мови MATLAB Engine API; розроблена інструкція з експлуатації програмного продукту «Wiagond».

Результати роботи знайшли своє відображення в [33].

ПЕРЕЛІК ПОСИЛАНЬ

1. Конституція України : Верховна Рада України; Конституція України, Конституція, Закон від 28.06.1996 № 254к/96-ВР. URL: https://zakon.rada.gov.ua/laws/show/254к/96-вр.
2. What is steganography and how does it differ from cryptography?. *Comparitech - Tech researched, compared and rated*. URL: https://www.comparitech.com/blog/information-security/what-is-steganography.
3. Сучасні стеганографічні методи захисту інформації. *ResearchGate | Find and share research*. URL: https://www.researchgate.net/publication/311663916_SUCASNI_STEGANOGRAFICNI_METODI_ZAHISTU_INFORMACII.
4. Структурна схема стегосистеми. Поняття та види контейнерів. *StudFiles. Файловий архів студентів*. URL: https://studfile.net/preview/9650052/page:4.
5. Стеганография в цифровых изображениях. *Материал из Википедии — свободной энциклопедии*. URL: http://ru.wikipedia.org/wiki/Стеганография_ в_цифровых_изображениях.
6. Алгоритм поиска в изображениях скрытых данных, встроенных методом Коха–Жао. *Библиотека материалов по теме выпускной работы Крахмаль Марии Вячеславовны*. URL: http://masters.donntu.org/2019/fknt/krakhmal/library/article10.htm.
7. Конахович Г.Ф., Пузыренко А.Ю. Компьютерная стеганография. Теория и практика : монографія. Київ : МК-Пресс, 2006. 288 с.
8. Кобозева, А.А., Нариманова Е.В. Оценка чувствительности стегосообщения к возмущающим воздействиям. *Системні дослідження та інформаційні технології*. 2008. №3. С. 52-65.
9. Documentation. *Scala Programming Language*. URL: https://docs.scala-lang.org.

10. What is R? *The R Project for Statistical Computing*. URL: https://www.r-project.org/about.html.
11. Overview. *MathWorks - Makers of MATLAB and Simulink*. URL: https://www.mathworks.com/products/matlab.html.
12. Julia 1.7 Documentation. *The Julia Programming Language*. URL: https://docs.julialang.org/en/v1.
13. Python is powerful... and fast; plays well with others; runs everywhere; s friendly & easy to learn; is Open. *The official home of the Python Programming Language*. URL: https://www.python.org/about.
14. Access MATLAB Add-On Toolboxes. *MathWorks - Makers of MATLAB and Simulink*. URL: https://www.mathworks.com/help/thingspeak/matlab-toolbox-access.html.
15. Mex. Build MEX function or engine application. *MathWorks - Makers of MATLAB and Simulink*. URL: https://www.mathworks.com/help/matlab/ref/mex.html.
16. Language Fundamentals. *MathWorks - Makers of MATLAB and Simulink* : веб-сайт. URL: https://www.mathworks.com/help/matlab/language-fundamentals.html.
17. Handle Classes. *MathWorks - Makers of MATLAB and Simulink* : веб-сайт. URL: https://www.mathworks.com/help/matlab/handle-classes.html.
18. WPF, UWP, WinUI, MAUI, Windows App SDK. *Habr*. URL: https://habr.com/ru/post/566352.
19. What Is Cocoa?. *Apple Developer*. URL: https://developer.apple.com/library/archive/documentation/Cocoa/Conceptual/CocoaFundamentals/WhatIsCocoa/WhatIsCocoa.html.
20. Introduction. *Electron documentation*. URL: https://www.electronjs.org/docs/latest.
21. Why Electron is a Necessary Evil. *Federico Terzi. A Software Engineering Journey*. URL: https://federicoterzi.com/blog/why-electron-is-a-necessary-evil.

22. About Qt. *The Qt wiki* : веб-сайт. URL: https://wiki.qt.io/About_Qt.
23. Language Bindings. *The Qt wiki*. URL: https://wiki.qt.io/ Language_Bindings.
24. Python - Official Image | Docker Hub. *Docker Hub Container Image Library. App Containerization*. URL: https://hub.docker.com/_/python.
25. mathworks/matlab - Docker Image. *Docker Hub Container Image Library. App Containerization*. URL: https://hub.docker.com/r/mathworks/matlab.
26. Get Started with MATLAB Engine API for Python. *MathWorks - Makers of MATLAB and Simulink*. URL: https://www.mathworks.com/help/matlab/ matlab_external/get-started-with-matlab-engine-for-python.html.
27. MATLAB Engine API for C++. *MathWorks - Makers of MATLAB and Simulink* : веб-сайт. URL: https://www.mathworks.com/help/matlab/calling-matlab-engine-from-cpp-programs.html.
28. Кодекс законів про працю України : Кодекс України; Закон, Кодекс від 10.12.1971 № 322-VIII. URL: https://zakon.rada.gov.ua/laws/show/322-08.
29. Про охорону праці : Закон України від 14.10.1992 № 2694-XII. URL: https://zakon.rada.gov.ua/laws/show/2694-12.
30. Про затвердження Правил охорони праці під час експлуатації електронно-обчислювальних машин (НПАОП 0.00-1.31-99) : Мінпраці України, Комнаглядохоронпраці; Наказ, Правила від 10.02.1999 № 21. URL: https://zakon.rada.gov.ua/laws/show/z0382-99.
31. Державні санітарні правила і норми роботи з візуальними дисплейними терміналами електронно-обчислювальних машин : МОЗ України; Правила від 10.12.1998 № 7. URL: https://zakon.rada.gov.ua/rada/show/v0007282-98.
32. Методичні вказівки до лабораторної роботи «Дослідження ефективності захисного заземлення» для студентів всіх спеціальностей / укл. О.Т. Озернюк. Одеса : ОНПУ, 2006. 16 с.
33. Кобозєва А.А., Кирилюк В.О., Надвоцький О.Ю., Сокальський С.О. Удосконалення методу кількісної оцінки захищеності

стеганоповідомлення. *Інформатика та математичні методи в моделюванні*. 2021. Т11, №3-4.

ДОДАТОК А

Лістинг програмного коду «Wiagond»

```
classdef CompCache < handle & JustLoggable
    properties (Constant)
        Sep  = "->";
        Root = string(DataHash("Root", 'SHA-256'));
    end
    properties
        Graph;
        Storage;
        Target;
        ObjectIdentifier;
        Roots = [];
        Hashes = "";
        Context;
        InsertionWeight;
    end

    methods
        function obj = CompCache(context, storage)
            arguments
                context
                storage {mustBeA(storage, "CompCacheStorage")}
            end
            obj = obj@JustLoggable(context);

            obj.Graph = digraph();
            obj.Storage = storage;
            obj.Context = context;
            obj.InsertionWeight = 1;
        end

        function obj = For(obj, target)
            obj.Target = target;
        end

        function obj = WithKey(obj, id)
            arguments
                obj
                id {mustBeA(id, "function_handle")}
            end
            obj.ObjectIdentifier = id;
        end

        function hash = Hash(obj, nodes)
            if length(nodes) == 1
                hash = nodes(1);
            else
                hash = DataHash(join(sort(nodes), ""), 'hex',
'MD2');
                obj.Hashes = vertcat(obj.Hashes, hash);
```

```
            end
        end

        function expr = TrimHashes(obj, expr)
            parts = split(expr, CompCache.Sep);
            expr = join(...
                arrayfun(@(x) if_(ismember(x, obj.Hashes), ...
                    @() extractBefore(x, 5 + 1), x), parts), ...
                CompCache.Sep);
        end


        function lastPart = ExtractLastPart(~, expr)
            lastPart = split(expr, CompCache.Sep);
            lastPart = lastPart(end);
        end

        function AddNodes(obj, nodes, child, proc)
            if isempty(nodes)
                newnode = child;
            else
                newnode = obj.Hash(nodes) + CompCache.Sep +
child;
            end
            obj.Graph = addnode(obj.Graph, ...
                table(newnode, {proc}, obj.InsertionWeight, ...
                VariableNames=["Name", "Proc", "Weight"]));
            obj.InsertionWeight = obj.InsertionWeight + 1;

            for node = nodes'
                obj.Graph = addedge(obj.Graph, node, newnode);
            end
        end

        function filteredNodes = NodesWhere(obj, nameExpressions)
            longGraphNodes = string(obj.Graph.Nodes.Name);
            shortGraphNodes = arrayfun(@obj.ExtractLastPart,
longGraphNodes);

            filteredNodes = [];
            for name = nameExpressions
                if contains(name, CompCache.Sep)
                    nodesSatisfying = ismember(longGraphNodes,
name);
                else
                    nodesSatisfying = ismember(shortGraphNodes,
name);
                end
                filteredNodes = vertcat(filteredNodes,
longGraphNodes(nodesSatisfying));
            end
        end
```

```
        function obj = Chain(obj, opts)
            arguments
                obj
                opts.forRoot = []
                opts.class {mustBeNonzeroLengthText}
                opts.proc {mustBeA(opts.proc, "function_handle")}
                opts.nocollision = false
            end

            if isempty(opts.forRoot)
                eachClass = @(c) obj.AddNodes([], c, opts.proc);
                obj.Roots = horzcat(obj.Roots, opts.class);
            else
                nodes = obj.NodesWhere(opts.forRoot);
                if isempty(nodes)
                    error("forRoot lookup failed.");
                end

                if opts.nocollision
                    eachClass = @(c) arrayfun(@(n)
obj.AddNodes(n, c, opts.proc), nodes);
                else
                    eachClass = @(c) obj.AddNodes(nodes, c,
opts.proc);
                end
            end

            arrayfun(eachClass, opts.class);
        end

        function [ops, mask, actual] =
CollectTransactionOperations(obj, actual, relevant)
            ops = struct(Created=[], Removed=[], Modified=[]);
            mask = struct(Created=[], Removed=[], Modified=[]);

            keepers = {actual, relevant};
            keeperNames = {string({actual.name}),
string({relevant.name})};
            args = flip(perms(keeperNames));
            for it = zip(1:2, ["Created", "Removed"])
                [i, member] = it{:};
                [~, indx] = setdiff(args{:, i});
                mask.(member) = indx;
            end

            commonMembersName = intersect(keeperNames{:});
            slicesIndxes = cellfun(...
                @(kn) ismember(kn, commonMembersName), ...
                keeperNames, "UniformOutput", false ...
            );
            slices = cellfun(...
                @(k, ki) k(ki), keepers, slicesIndxes,
"UniformOutput", false ...
```

```
            );

            if nargout(obj.ObjectIdentifier) ~= 1
                meta = cell(1, nargout(obj.ObjectIdentifier) -
1);
                [actualKeys, meta{1:length(meta)}] = ...
                    obj.ObjectIdentifier(slices{1});
                if mod(length(meta), 2) ~= 0
                    obj.Log.Warn("Key function {!} incorrectly
sets metainfo", ...
                            obj.ObjectIdentifier);
                end
                for i = 1:2:length(meta)
                    metaName = meta{i};
                    metaInfo = meta{i+1};
                    actual.(metaName) = metaInfo;
                end
                slices{1} = actual(slicesIndxes{1});
            else
                actualKeys = obj.ObjectIdentifier(slices{1});
            end
            relevantKeys = obj.ObjectIdentifier(slices{2});
            mask.Modified = actualKeys ~= relevantKeys;
            ops.Modified = slices{1}(mask.Modified);

            for it = ["Created", "Removed"]
                ops.(it) = actual(mask.(it));
            end
        end

        function dirs = Dir(~, d)
            dirs = dir(d);
            dirs(ismember({dirs.name}, {'..', '.'})) = [];
        end

        function name = FilterStructName(~, name)
            name = "F_" + strrep(name, "->", "_");
        end

        function Satisfy(obj, element, actualMeta, runner,
debugSlice)
            obj.Log.Trace("Running {!} chain", element);
            proc =
table2cell(obj.Graph.Nodes(obj.Graph.Nodes.Name == element,
"Proc"));

            element = obj.FilterStructName(element);
            targetComp = obj.Storage.Retrieve(element);
            if isempty(targetComp)
                obj.Log.Trace("'{!}' chain metadata information
is empty", element);
                targetComp = struct(...
                    name='', folder='', date='', ...
```

```
                    bytes=0, isdir=false, datenum=0 ...
                );
                targetComp = targetComp([]);
            elseif ~isempty(debugSlice)
                targetComp = targetComp(debugSlice);
            end

            proc = repmat(proc, length(actualMeta), 1);
            [actualMeta.Proc] = proc{:};
            [ops, mask, actualChainMeta] =
obj.CollectTransactionOperations(...
                actualMeta, targetComp);
            if isempty(targetComp)
                targetComp = actualChainMeta([]);
                targetComp(1).(element) = [];
                targetComp = targetComp([]);
            end
            if ~all(cellfun(@(f) isempty(ops.(f)),
fieldnames(ops)))
                obj.Log.Trace("Pending chain operations\n{}",
ops);

                for field = ["Created", "Modified"]
                    if isempty(ops.(field))
                        continue;
                    end
                    t = runner.Compute(element, mask.(field),
ops.(field));
                    [ops.(field).(element)] = t{:};
                    targetComp = obj.Storage.Transact(...
                        element, targetComp,
CompCacheTransaction.Add, ops.(field));
                end
                affected = [mask.Created; mask.Modified;
mask.Removed];
                targetMask = ones(length(affected), 1,
"logical");
                targetMask(affected) = false;
                if any(targetMask)
                    runner.Assign(element, targetMask,
actualChainMeta(targetMask));
                end

                if ~isempty(ops.Removed)
                    targetComp = obj.Storage.Transact(...
                        element, targetComp,
CompCacheTransaction.Remove, ops.Removed);
                end
            else
                obj.Log.Trace("Chain {!} is up to date!",
element);
                runner.Assign(element, ones(length(targetComp),
1, "logical"), targetComp);
```

```
                return;
            end

            if isempty(debugSlice)
                obj.Storage.Transact(element, targetComp,
CompCacheTransaction.Finalize);
            end
        end

        function sorted = SortByWeight(~, graph, items)
            if isempty(items)
                sorted = [];
                return;
            end

            weights = arrayfun(@(p) ...
                table(graph.Nodes(graph.Nodes.Name == p,
:).Weight, p, ...
                    VariableNames=["Weight", "Name"]), ...
                items, "UniformOutput", false);
            weights = sortrows(vertcat(weights{:}));
            sorted = weights.Name;
        end

        function res = Build(obj, opts)
            arguments
                obj
                opts.selector {mustBeNonzeroLengthText}
                opts.debugSlice = []
            end
            if any(cellfun(@isempty, {obj.Storage, obj.Target,
obj.ObjectIdentifier}))
                error("CompCache is not initialized enough to be
built.");
            end

            graph = addnode(obj.Graph, ...
                table(CompCache.Root, {@disp}, 0, ...
                VariableNames=["Name", "Proc", "Weight"]));
            for r = obj.Roots
                graph = addedge(graph, CompCache.Root, r);
            end
            paths = arrayfun(@(n) flatten(allpaths(graph,
CompCache.Root, n)), ...
                obj.NodesWhere(opts.selector), "UniformOutput",
false);
            paths = unique(flatten(paths));

            paths = obj.SortByWeight(graph, paths);
            graph = subgraph(graph, paths);

            if ~isstruct(obj.Target)
                actualMeta = obj.Dir(obj.Target);
```

```
            else
                actualMeta = obj.Target;
            end
            if ~isempty(opts.debugSlice)
                slice = opts.debugSlice;
                opts.debugSlice = 1:length(actualMeta);
                opts.debugSlice = opts.debugSlice <= slice;
                actualMeta = actualMeta(opts.debugSlice);
            end
            obj.Log.Trace("Actual metadata information\n{}",
actualMeta);

            satisfied = CompCache.Root;
            runner = CompCacheRunner(obj.Context, actualMeta);
            for it = string(bfsearch(graph, CompCache.Root)')
                if it == CompCache.Root
                    continue;
                end
                if ismember(it, satisfied)
                    continue;
                end

                satisfied(end+1) = it;

                deps = allpaths(graph, CompCache.Root, it);
                deps = string([deps{:}]);
                deps = unique(deps, "stable");

                deps = deps(~ismember(deps, satisfied));
                deps = obj.SortByWeight(graph, deps);
                for d = deps'
                    obj.Satisfy(d, actualMeta, runner,
opts.debugSlice);
                end
                obj.Satisfy(it, actualMeta, runner,
opts.debugSlice);
            end

            res = runner.Workspace;
        end

        function Info(obj)
            g = obj.Graph;

            obj.Log.Info("{}", g.Nodes);
            g.Nodes.Name = arrayfun(@obj.TrimHashes,
string(g.Nodes.Name));

            plot(g);
        end
    end
end
```

```
classdef JustExecuteStage < JustLoggable & JustStage &
StagesCommon & JustInjectProps
   properties (Constant)
        Name = "execute";
        Description = [...
            "runs execution stage. Usage:", ...
            "just execute <args> ", ...
            "-i <file/file-list> for inputs", ...
            "-m <JSON array of arrays [Mod, param]> for
modifications, where Mod can be " + ...
            "the following: JPEG, GaussianNoise, SpeckleNoise"
        ];
        ForceSerialExecution = true;
    end

    properties
        Context;
        KochZhaoDiff = JustInjectableProp("main-stage.koch-
zhao.diff");
        K = JustInjectableProp("main-stage.K");
        SteganoImpl;
        Message;
        Method;
        ImmodImpl;
        SpectralIndex = JustInjectableProp("main-
stage.image.spectral-index");
        SpectralTriangle;

        Target;
        Modifications;
    end

    methods
        function obj = JustExecuteStage( ...
                context, inputs, modifications)
            obj = obj@JustLoggable(context);
            obj = obj@JustStage();
            obj = obj@StagesCommon(context);
            obj = obj@JustInjectProps(context);

            obj.Context = context;
            obj.ImmodImpl = ImageModifications([], [], []);

            obj.Target = obj.FileListToDirStruct(inputs);
            obj.Modifications = jsondecode(modifications);

            obj.Log.Info("Building stegano toolchain");
obj.SteganoPreconfigure(SteganoMethodFactoryRetriever.KochZhao);

            if obj.SpectralIndex == obj.ATriangle
                obj.SpectralTriangle = @tril;
            else
                obj.SpectralTriangle = @triu;
```

```
            end
        end

        function SteganoPreconfigure(obj, method)
            factory = SteganoMethodFactoryRetriever.For(method,
obj.Context);

            obj.Log.Info("Stegano method {!}", method);
            [ok, messageLength] =
factory.CheckTarget(obj.ImageSize);
            if ~ok
                error(...
                    "Target image size cannot be used by %s. " +
...
                    "Please check 'main-stage.image.crop-to'
property.", ...
                    method ...
                );
            end
            random = Random(obj.Context);
            obj.Message = random.Logical(size=[1,
messageLength]);
            key = factory.BuildKey(...
                targetSize=obj.ImageSize, message=obj.Message,
...
                diff=obj.KochZhaoDiff ...
            );

            obj.SteganoImpl =
factory.Create(obj.Context).SetKey(key);
            obj.Log.Info("Stegano Transformation implementation
{!}", ...
                class(obj.SteganoImpl));
        end

        function proc = MakeSpectralProc(~, prop)
            adjust = 1 - strlength("_" + prop);
            trimmer = @(p) extractBefore(p, strlength(p) +
adjust);
            function U = ueig(env, f)
                [U, ~] = eig(double(env.(trimmer(f))));
                mask = U(1, :) < 0;
                U(:, mask) = -U(:, mask);
            end
            function L = leig(env, f)
                [~, L] = eig(double(env.(trimmer(f))));
            end
            switch prop
                case "U"
                    proc = @ueig;
                case "L"
                    proc = @leig;
            end
```

```
        end

        function deviation = DeltaImageDeviation(~, env, ~)
            deviation = vecnorm(abs(env.F_Image_Symmetric_U - ...
                env.F_Image_DeltaImage_Symmetric_U));
            deviation = deviation / norm(deviation);
        end

        function distribution = Distribution(~, env, ~)
            distribution = env.(FieldUtils.FieldWhere(env,
"DeltaDeviation"));
            distribution = distribution ./ sum(distribution) *
100;
        end

        function capacity = Capacity(obj, env, ~)
            deviation = env.(FieldUtils.FieldWhere(env,
"EDeviation"));
            distributionMask = deviation < obj.K;

            distribution = env.(FieldUtils.FieldWhere(env,
"Distribution"));
            capacity = sum(distribution(distributionMask));
        end

        function result = EntryPoint(obj)
            obj.Log.Info("Execution stage is running");

            cache = obj.GetImagesChain(obj.Context, "exec",
obj.Component, obj.ImageSize, obj.Target) ...
                .Chain(forRoot="Image", class="Symmetric",
proc=@obj.Symmetric);

            cache = cache ...
                .Chain(forRoot="Image", class="DeltaImage",
proc=@obj.SteganoTransform) ...
                .Chain(forRoot="DeltaImage", class="Symmetric",
proc=@obj.Symmetric) ...
                .Chain(forRoot="Image", class="EImage",
proc=@obj.DoModifications) ...
                .Chain(forRoot="EImage", class="Symmetric",
proc=@obj.Symmetric);

            for s = ["U", "L"]
                cache = cache.Chain(...
                    forRoot="Symmetric", nocollision=true, ...
                    class=s, proc=obj.MakeSpectralProc(s));
            end

            cache = cache.Chain(...
                forRoot=[...
                    "Image->Symmetric->U" ...
                    "Image->DeltaImage->Symmetric->U",
```

```
                ], ...
                class="DeltaDeviation",
proc=@obj.DeltaImageDeviation ...
            );
            cache = cache.Chain(...
                forRoot=[...
                    "Image->Symmetric->U" ...
                    "Image->EImage->Symmetric->U",
                ], ...
                class="EDeviation", proc=@obj.EImageDeviation ...
            );
            cache = cache.Chain( ...
                forRoot="DeltaDeviation", ...
                class="Distribution", proc=@obj.Distribution ...
            );
            cache = cache.Chain( ...
                forRoot=["Distribution", "EDeviation"], ...
                class="Capacity", proc=@obj.Capacity ...
            );

            result = cache.Build(selector="Capacity",
debugSlice=length(obj.Target));
            capacityFieldName = FieldUtils.FieldWhere(result,
"Capacity");
            for f = string(fieldnames(result))'
                if ~ismember(f, ["name", "folder",
capacityFieldName])
                    result = rmfield(result, f);
                end
            end

            paths = cellfun(...
                @(f, n) fullfile(f, n), {result.folder},
{result.name}, ...
                "UniformOutput", false ...
            );
            [result.path] = paths{:};

            result = rmfield(result, "name");
            result = rmfield(result, "folder");
            capacity = {result.(capacityFieldName)};
            result = rmfield(result, capacityFieldName);
            [result.capacity] = capacity{:};

            result = string(jsonencode(result));
        end
    end
end
```