НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ОДЕСЬКА ПОЛІТЕХНІКА»
МІНІСТЕРСТВА ОСВІТИ І НАУКИ УКРАЇНИ
Кафедра комп’ютерних інтелектуальних систем та мереж

ПИНАРГОЗЮ Еврім Джема

КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

ВЕБ-СЕРВІС ДЛЯ ВБУДОВУВАННЯ ЦИФРОВИХ ВОДЯНИХ ЗНАКІВ У РАСТРОВІ ЗОБРАЖЕННЯ

Спеціальність 123 – Комп’ютерна інженерія
Спеціалізація – Комп’ютерні системи та мережі

Керівник: Защолкін Костянтин Вячеславович,
доктор технічних наук, професор

Одеса – 2022

З А В Д А Н Н Я

НА ДИПЛОМНИЙ ПРОЕКТ (РОБОТУ) СТУДЕНТУ

Пинаргозю Еврім Джема
(прізвище, ім'я, по батькові)

1.Тема проекту (роботи) Веб-сервіс для вбудовування цифрових водяних знаків у растрові зображення

керівник проекту (роботи) Защолкін К.В.
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом ректора від " 06 " 06.2022 № 187-в

2. Строк подання студентом проекту (роботи) 14.06.2022

3. Вихідні дані до проекту (роботи) Виконати розробку веб-сервісу вбудовування цифрових водяних знаків у растрові зображення.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

1. Аналітичний огляд методів цифрової стеганографії
2. Постановка та аналіз завдання кваліфікаційної роботи
3. Розробка веб-сервісу вбудовування цифрових водяних знаків

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)

1. Структура системи стеганографічного захисту даних
2. Вхідні та вихідні потоки даних
3. Діаграма використання веб-сервісу
4. Зв'язки модулів веб-сервісу
5. Діаграма послідовності для процесу вбудовування даних
6. Діаграма послідовності для процесу витягання даних
7. Схема алгоритму функціонування модуля вбудовування даних
8. Діаграма класів серверної частини веб-сервісу

6. Консультанти розділів проекту (роботи)

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

7. Дата видачі завдання 01.03.2022

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проекту (роботи)	Строк виконання етапів проекту (роботи)	Примітка
1	Аналітичний огляд систем та методів	15.03.22	виконано
2	стеганографічного захисту даних		
3	Специфікація вимог до веб-сервісу,	25.03.22	виконано
4	що проектується		
5	Проектування веб-сервісу вбудовування	15.04.22	виконано
6	цифрових водяних знаків		
7	Програмна реалізація веб-сервісу	10.05.22	виконано
8	вбудовування цифрових водяних знаків		
9	Оформлення пояснювальної записки	07.06.22	виконано
10			

Студент _____ Пинаргозю Е.Д.
(підпис) (прізвище та ініціали)

Керівник проекту (роботи)_____ Защолкін К.В.
(підпис) (прізвище та ініціали)

Відомість кваліфікаційної роботи бакалавра

№ рядка	Найменування	Кільк.	Примітка
1	Пояснювальна записка	60	
2	Структура системи стеганографічного захисту даних	1	
3	Вхідні та вихідні потоки даних	1	
4	Принцип використання зображення в якості контейнера	1	
5	Діаграма використання веб-сервісу	1	
6	Зв’язки модулів веб-сервісу	1	
7	Діаграма послідовності для процесу вбудовування даних	1	
8	Діаграма послідовності для процесу витягання даних	1	
9	Діаграма комунікацій веб-сервісу	1	
10	Схема алгоритму функціонування модуля вбудовування даних	1	
11	Діаграма класів серверної частини веб-сервісу	1	
12			
13			
14			
15			
16			
17			
18			
19			
20			
21			
22			
23			
24			
25			
26			
27			
28			

					АМДР.АМ183.3418
Зм.	*Лист*	*№ докум.*	*Підпис*	*Дата*	
Розробив		Пинаргозю Е.Д.			Веб-сервіс для вбудовування цифрових водяних знаків у растрові зображення Відомість кваліфікаційної роботи
Перевірив		Защолкін К.В.			
Реценз.					
Н. Контр.					
Затвердив					

Літ.	*Лист*	*Листів*

Одеська політехніка
ІКС, Каф. КІСМ

АНОТАЦІЯ

Пинаргозю Е.Д. Веб-сервіс для вбудовування цифрових водяних знаків у растрові зображення – кваліфікаційна робота бакалавра. Одеса, 2022: 60 с., 8 рис., 1 табл., 1 додаток, 5 джерел.

Метою роботи є розробка веб-сервісу для вбудовування цифрових водяних знаків у растрові зображення.

У роботі виконано аналіз існуючих програмних систем, які здійснюють захист інформації шляхом використання стеганографічного підходу. Також проведено аналітичний огляд розроблених алгоритмів стеганографічного приховування даних. За результатами аналізу було обрано для реалізації ефективний метод стеганографічного приховування даних в просторову область растрових зображень. На основі обраного методу стеганографічного приховування даних побудовано алгоритми функціонування програмного забезпечення відповідного веб-сервісу. Виконано проектування та програмну реалізацію веб-сервісу для забезпечення стеганографічного захисту даних. Розроблений веб-сервіс дозволяє здійснити приховування секретних даних в просторову область растрових зображень та непомітно передавати ці дані по відкритих каналах передачі. На приймаючій стороні за допомогою розробленого веб-сервісу забезпечено діставання прихованих даних при використанні відповідного стеганографічного ключа.

ЦИФРОВІ ВОДЯНІ ЗНАКИ, ЦИФРОВА СТЕГАНОГРАФІЯ, РАСТРОВІ ЗОБРАЖЕННЯ, ЗАХИСТ ІНФОРМАЦІЇ

ANNOTATION

Pinargozyu E.D. Web service for digital watermarking of bitmap images – qualification work of bachelor. Odessa, 2022: 60 pp., 8 fig., 1 table, 1 supplement, 5 sources.

The goal of the work is to develop a web service for embedding digital watermarks in bitmap images.

In the qualifying work, an analysis of existing software systems that protect information by using the steganographic approach is carried out. An analytical review of the developed algorithms for steganographic data hiding was also carried out. Based on the results of the analysis, an effective method of steganographic data hiding in the spatial domain of raster images was chosen for implementation. Based on the chosen method of steganographic data hiding, algorithms for the functioning of the software of the corresponding web resource are constructed. The design and software implementation of a web resource to ensure steganographic data protection has been completed. The developed web resource allows hiding secret data in the spatial area of raster images and transmitting this data imperceptibly through open transmission channels. On the receiving side, using the developed web resource, the extraction of hidden data is ensured using the appropriate steganographic key.

DIGITAL WATERMARKS, DIGITAL STEGANOGRAPHY, RASTER IMAGES, INFORMATION SECURITY

ЗМІСТ

ВСТУП

Захист інформації є одним з ключових процесів сучасної обробки, зберігання та передачі даних у просторі. Захист інформації криптографічним шляхом, тобто шляхом шифрування секретних даних є дуже розвинутим напрямом інформаційної безпеки. Але разом з перевагами цей напрям має недоліки, що полягають у відкритості факту наявності секретних даних. На противагу криптографічному підходу, конкуруючий стеганографічний підхід приховує факт наявності секретних даних. Таким чином, розробка програмного забезпечення, яке реалізує захист інформації шляхом використання стегнаоографічного підходу є актуальною.

Мета кваліфікаційної роботи полягає у створенні веб-сервісу для вбудовування цифрових водяних знаків у растрові зображення.

Для досягнення поставленої мети в кваліфікаційній роботі необхідно вирішити наступні задачі:

– провести аналіз методів стеганографічного захисту даних, зокрема методів приховування даних у просторовій області растрових зображень;

– виконати аналіз аналогів веб-сервісу для вбудовування цифрових водяних знаків у растрові зображення, що розробляється в кваліфікаційній роботі та порівняти очікувані характеристики веб-ресурсу з характеристиками аналогів;

– обрати платформу та середовище розробки, а також мову програмування, з використанням яких проводитиметься розробка веб-сервісу, і виконати обґрунтування вибору;

– розробити алгоритми та формальні описи функціонування модулів веб-сервісу для вбудовування цифрових водяних знаків у растрові зображення;

– виконати розробку зазначеного веб-сервісу;

– провести тестування отриманого веб-сервісу за результатами якого довести досягнення мети роботи.

1 АНАЛІТИЧНИЙ ОГЛЯД ПІДХОДІВ ДО ВБУДОВУВАННЯ ЦИФРОВИХ ВОДЯНИХ ЗНАКІВ У РАСТРОВІ ЗОБРАЖЕННЯ

1.1 Основні поняття цифрової стеганографії

Стеганографія це напрямок теорії захисту інформації, в межах якого секретне повідомлення вбудовується в цифровий об’єкт не змінюючи при цьому якості сприйняття цього об’єкту. Зазвичай контейнерами для вбудовування секретної інформації є мультимедійні об'єкти такі як растрові зображення, оцифроване відео та аудіо.

Основними напрямками використання цифрової стеганографії є наступні галузі:

- вбудовування секретної інформації для здійснення її прихованої передачі за відкритими каналами передачі даних;
- вбудовування цифрових водяних знаків в мультимедійні стего контейнери;
- вбудовування ідентифікаційних номерів у файли з мультимедійним контентом;
- вбудовування заголовків у різні цифрові об’єкти.

Основна сфера застосування цифрових водяних знаків це захист цифрового контенту від копіювання, а також його несанкціонованого використання. Вельми ефективним заходом захисту мультимедійного контенту є вбудовування до відповідного файлу невидимих знаків, які можуть бути проявлені при наявності секретного стего-ключа. Тобто за замовчуванням цифровий водяний знак візуально не відображається, більш

того, сторонній спостерігач навіть не знає факту існування цифрового водяного знаку в інформаційному об’єкті. За наявності стего-ключа цифровий водяний знак можна витягнути з цифрового об’єкта та проявити його існування.

1.2 Стеганографічні системи

Як і для криптографічних систем захисту інформації, безпека стегосистем описується й оцінюється їх стійкістю (стеганографічною стійкістю чи просто стегостійкістю). Під стійкістю різноманітних стегосистем розуміється їх здатність приховувати від кваліфікованого порушника факт прихованої передачі повідомлень, здатність протидіяти намаганням знищення, спотворення, видалення прихованого повідомлення, а також здатність підтвердити чи спростувати справжність передаваємої приховано інформації.

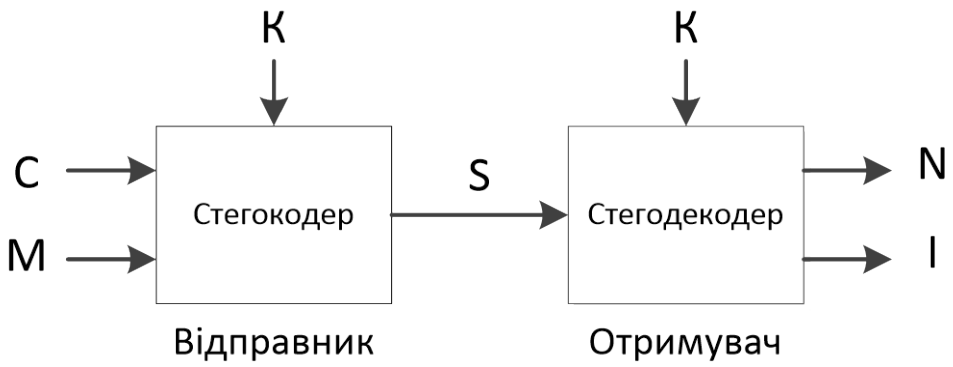

Рисунок 1.1 - Спрощена модель стегосистеми

В криптографічних системах (рис. 1.1) приховується зміст конфіденційного повідомлення, в той час як в стеганографії додатково

приховується факт існування такого повідомлення. Саме цьому визначення стійкості та злому цих систем відрізняються. В криптографії система захисту інформації є стійкою, якщо маючи перехоплену крипто програму, порушник не може читати повідомлення що в ній знаходиться. Тож можна неформально визначити, що стегосистема є стійкою, якщо порушник спостерігаючи інформаційний обмін між відправником та отримувачем , не здатен помітити, що під прикриттям контейнерів передаються приховані повідомлення, і тим більше читати їх.

1.3 Приховування даних у растрових зображеннях

Найбільш поширеними стегоконтейнерами є зображення. В стегоалгоритмах досить часто використовуються ті ж перетворення, що й у сучасних алгоритмах стискання (наприклад, в JPEG). При цьому, очевидно, існує три можливості. Вбудовування інформації може відбуватися у вихідне зображення, або одночасно із стисканням зображення-контейнеру, або ж у вже стиснуте алгоритмом JPEG зображення.

В останні роки у зв’язку з інтенсивним розвитком мультимедійних технологій дуже гостро постало питання захисту авторських прав та інтелектуальної власності, що представлена у цифровому вигляді. Особливо актуальною ця проблема стала із розвитком загальнодоступних комп’ютерних мереж, в тому числі й мережі Інтернет. З урахуванням цього, наразі задачі захисту від копіювання та забезпечення аутентифікації розв’язуються, окрім заходів організаційно-юридичного характеру, з застосуванням технологій цифрових водяних знаків. Необхідно відмітити,

що найбільші досягнення стеганографії в минулому десятилітті були отримані саме в області розвитку цифрових водяних знаків. Ці досягнення викликані реакцією суспільства на актуальну проблему захисту авторських прав в умовах загальнодоступних комп'ютерних мереж.

Стеганографія використовується в деяких сучасних принтерах. При друці, окрім основного тексту, на кожну сторінку печатаються мікроточки, що надають інформацію про дату і час друку, фірму та серійний номер принтера тощо. Зроблено це за проханням державних структур аби було легше відслідковувати фальшивомонетників чи, наприклад, осіб, які надрукували листа із погрозами і т.д.

Однією з таких технологій є так звані «кільця Омрона», що наносяться на банкноти та деякі документи для протидії їх непрофесійній підробці. Називаються вони так по назві японської корпорації Omron, що має підтверджений Резервним банком Індії патент на систему виявлення промаркованих таким чином банкнот при скануванні та друці. Технічні дані цієї системи не розголошуються навіть компаніям, в чиє програмне забезпечення ця система вбудовується.

Загальний зміст «кілець Омрона» полягає в наступному: при емісії на купюрах друкуються кола (власне, «кільця Омрона») в деякій конфігурації, надалі при намаганні копіювання чи друку після сканування таких купюр на пристроях, що мають програмне забезпечення із вбудованою системою розпізнавання таких міток, друк або не відбувається, або частина зображення закривається чорним.

Вся ця система існує для протидії непрофесійному фальшивомонетництву. Наразі пропонується розробка або специфічних конфігурацій «кілець Омрона», які б наносилися лише на російські

купюри, або системи загалом, включаючи конкретні мітки, їх конфігурації, способи нанесення та програмні засоби виявлення.

1.4 Сфери застосувань стеганографії та технології цифрових водяних знаків

Аналіз тенденій розвитку стеганографії показує, що в найближчі роки інтерес до розвитку її методів буде збільшуватися все більше і більше. Передумови для цього вже сформувалися. Загальновідомо, що актуальність проблеми інформаційної безпеки постійно зростає і стимулює пошук нових методів захисту інформації. З іншого боку, бурхливий розвиток інформаційних технологій забезпечує можливість реалізації цих нових методів захисту інформації. І, звичайно, сильним каталізатором цього процесу є лавиноподібний розвиток Інтернету, в тому числі такі нерозв’язані контроверсійні проблеми Інтернету, як захист авторського права, захист прав на особисту таємницю, організація електронної торгівлі, протиправна діяльність хакерів, терористів тощо.

Досить характерною тенденцією в області захисту інформаціє є наразі широке впровадження криптографічних методів. Однак, на цьому шляху є ще багато нерозв’язаних питань, що пов’язані з руйнівною дією на крипто засоби таких складових електронної зброї як комп’ютерні віруси, логічні бомби, автономні репликативні програми і т.д. З іншого боку, проблема розповсюдження ключів при використанні криптографічних методів є також не до кінця завершеною. Об’єднання методів стеганографії та криптографії стало б гарним виходом із положення що склалося, так як в цьому випадку вдалося б позбутися слабких місць відомих методів

захисту інформації та розробити більш ефективні нові нетрадиційні методи забезпечення інформаційної безпеки.

Таким чином, наразі стеганографія стає основною для створення перспективних систем захисту інформації, оперативно-технічні характеристики яких визначаються новими інформаціонними технологіями. Сьогодні стеганографія дозволяє не лише успішно вирішувати основну задачу – приховано передавати інформацію, але й розв’язувати цілий ряд других актуальних завдань, в тому числі, завадостійкої аутентифікації, захисту від несанкціонованого копіювання, моніторингу інформації в мережах зв’язку, пошуку інформації в мультимедійних базах даних тощо.

1.5 Висновки

В даному розділі кваліфікаційної роботи виконано аналітичний огляд напрямів захисту інформації. Серед них виділено цифрову стеганографію. Визначено переваги та недоліки стенографічного підходу до захисту даних. Проаналізовано потенційні галузі застосування стеганографії. Відмічено один з прикладних напрямів теорії стеганографії – технологію цифрових водяних знаків, яка використовується для захису мультимедійного контенту.

2 ПОСТАНОВКА ТА АНАЛІЗ ЗАВДАННЯ КВАЛІФІКАЦІЙНОЇ РОБОТИ

2.1 Постановка завдання на розробку

2.1.1 Призначення та галузі використання розробки

В кваліфікаційній роботі необхідно розробити веб-сервіс стеганографічного вбудовування цифрових водяних знаків в просторову область растрового зображення. При цьому умовою функціонування веб-сервісу є те, що візуальні спотворення зображення мають бути непомітні для людського ока.

В даній кваліфікаційній роботі об’єктом програмної реалізації є процес вбудовування секретної текстової інформації у просторову область растрового зображення та обернений процес – виділення прихованої інформації із зображення. При цьому інформація, що вбудовується, являє собою цифровий водяний знак. Цей знак є непомітним маркуванням зображення, яке може бути проявлене при наявності відповідного стеганограічного ключа.

2.1.2 Мета та задачі розробки

Метою кваліфікаційної роботи виступає створення веб-сервісу вбудовування цифрових водяних знаків в растрові зображення, яке являє собою стеганографічне приховування даних в просторовій області растрового зображення. Використання веб-сервісу, що реалізовується в даній кваліфікаційній роботі, повинно забезпечувати наступні функціональності та можливості:

– вбудовування приховуваної інформації в растрове зображення без його візуальних спотворень;

– отримання прихованої інформації з заповненого раніше растрового зображення.

Для досягнення поставленої мети в кваліфікаційної роботі потрібно:

– вибрати мову програмування та середовище для розробки програмних модулів;
– вивчити теоретичні відомості представлення растрових зображень та їх обробки, в тому числі за допомогою обраної мови програмування;
– вибрати та вивчити стеганографічні методи, що дозволяють вбудовувати приховувані текстові повідомлення в просторову область растрових зображень;
– розробити алгоритми функціонування підмодулів вбудовування та виділення даних згідно обраних методів;
– виконати програмну реалізацію розроблених алгоритмів;
– протестувати розроблений веб-сервіс;
– оформити пояснювальну записку та графічні матеріали кваліфікаційної роботи згідно попередньо вирішених завдань.

2.1.3 Загальні вимоги до розробки

Розроблюваний в межах даної кваліфікаційної роботи веб-сервіс повинен надавати можливості вбудовування та виділення текстових повідомлень з растрових зображень наступними методами:

– метод заміни молодшого значущого біту з фіксованим кроком вбудовування біт;
– метод заміни молодшого значущого біту з псевдовипадковим кроком вбудовування біт;
– зональний стеганографічний метод.

Для роботи з розроблюваним веб-сервісом повинна забезпечуватися наступна послідовність дій. При вбудовуванні користувач має зробити наступне:

– обрати зображення-контейнер, в яке надалі буде вбудовано приховуване повідомлення;
– обрати один з методів для вбудовування;
– обрати стего-ключ (ключові параметри) вбудовування;
– ввести текст приховуваного цифрового водяного знаку в текстове поле або обрати відповідний текстовий файл;
– після вбудовування задати ім’я новоствореного зображення із прихованим цифровим водяним знаком.

Стего-ключ вбудовування при цьому являє собою наступну сукупність атрибутів:

– маркер;
– кольорова компонента;
– крок вбудовування для методу молодшого значущого біту з фіксованим кроком;
– ініціалізаційне число та межі інтервалу генерації кроку для методу молодшого значущого біту з псевдовипадковим кроком;
– або ініціалізаційне число та поріг для блочного методу вбудовування.

Аби отримати приховане повідомлення із заповненого зображення-контейнеру користувач має знати метод та ключові параметри, що використовувалися при вбудовуванні.

Розроблюваний веб-сервіс повинен дозволяти та забезпечувати виконання наступних дій:

– обрання зображення користувачем;
– введення ключових параметрів та тексту повідомлення;

– перевірку коректності вводу;
– вбудовування та виділення приховуваного повідомлення;
– відображення виконання дій на екрані.

2.1.4 Вхідні та вихідні данні веб-сервісу, що розробляється

На вхід веб-сервісу, що розробляється поступає растровий графічний стего-контейнер та цифровий водяний знак, який необхідно вбудувати в цей стего-контейнер.

Веб-сервіс, що розробляється в кваліфікаційній роботи отримує ці елементи вхідних даних, виконує вбудовування секретної інформації в стего-контейнер та повертає заповнений стего-контейнер у якості вихідних даних.

2.2 Аналіз методу заміни молодшого значущого біту

2.2.1 Аналіз загальних положень методу

Цифрові растрові зображення являються собою собою прямокутну матрицю пікселів. Піксель це елементарна одиниця зображення. Зазвичай в інформаційному плані пік сель складається з трьох цілих двійкових чисел деякої довжини, кожне з яких визначає кількісну характеристику певної кольорової компоненти.

Молодший значущий біт кожної колірної компоненти пікселю зображення несе в собі найменшу кількість інформації. Людина візуально не здатна зафіксувати зміну кольору пікселя, яка виникає через зміну молодшого значущого біта. Тому молодший значущий біт можна використовувати для вбудовування інформації шляхом заміни таких бітів бітами секретного повідомлення або цифрового водяного знаку.

Перевагами підходу. що полягає у використанні молодших значущих бітів в якості місця для зберігання розрядів секретного повідомлення є

простота та великий обсяг даних, який може бути прихований в графічних файлах.

Однак зазначений підхід має і неякі недоліки. Так цей метод заміни молодшого біту має відносно низьку стеганографічну стійкість до стеганографічних атак. Крім того, цей метод є чутливим до спотворень стего-контейнеру. Для зменшення такої чутливості використовується завадостійке кодування, а також дублювання даних, які вбудовуються в стего-контейнер.

Також зазначений метод, не завжди забезпечує секретність вбудовування інформації. Оскільки порушнику є відомим потенційне місце розміщення розрядів цифрового водяного знаку. Йому не відома послідовність зчитування цих розрядів та розділення їх на множину корисних та не корисних розрядів. Для подолання зазначеного недоліку цифровий водяний знак вбудовують не у всі пікселі растрового зображення, а лише певну їх підмножину. Ця підмножина визначається стего-ключем, який є відомим лише легітимному користувачу.

Відомо, що розподіл молодшого значущого біту зображення не є повніть випадковим. В силу цього для забезпечення секретності, цифровий водяний знак, що вбудовується в стего-контейнер не повиннен змінювати статистику розподілу.

2.2.2 Метод заміни молодшого значущого біту з фіксованим кроком

Метод заміни молодшого значущого біту з фіксованим кроком є різновидом методу молодшого значущого біту, в якому інтервал між пікселями, що використовуються для вбудовування цифрового водяного знаку є заздалегідь визначеним компонентом стего-ключа. Використання даного методу дозволяє боротися зазначеним вище недоліком первісного методу модифікації молодшого значущого біту.

При застосування цієї модифікації методу користувач обирає растровий-контейнер та фіксований інтервал між пікселями, в молодші розряди кольорових компонент яких будуть вбудовуватися біти цифрового водяного знаку. На основі цього, формується множина пікселів, молодші значущі біти яких будуть замінюватися на біти цифрового водяного знаку. Цифровий водяний знак, який задається користувачем, розбивається на множину біт теж на зазначеному етапі методу.

Приховування цифрового водяного знаку здійснюється шляхом вбудовування повідомлення в растровий контейнер. Для цього виконується зміна бітів з першої множини на біти з другої зазначеної множини. Окрім повідомлення в контейнер вбудовується й інформація для аутентифікації на стороні одержувача, а також допоміжні дані.

Для перевірки легітимності одержувача прихованого повідомлення в контейнер перед повідомленням вбудовується маркер, який є паролем, що встановив відправник. При намаганні отримати приховану в контейнері інформацію легітимним шляхом, за допомогою веб-сервісу, спочатку перевіряється відповідність маркеру введеному одержувачем тому, що знаходиться в контейнері. Без спів падання маркеру та пароля повідомлення не буде витягнуто з зображення-контейнеру.

Після маркеру в контейнер записується довжина таємного повідомлення, що дозволяє зчитувати лише необхідну інформацію.

2.2.3 Метод заміни молодшого значущого біту з псевдовипадковим кроком

Цей метод на відміну від попереднього відрізняється процедурою розділення пік селів на множини корисних та не корисних пікселів. Метод полягає у випадковому розподілі біт секретного повідомлення по контейнеру, в результаті чого відстань між двома вбудованими бітами визначається псевдовипадково. Таким чином даний метод співпадає із

попереднім щодо процедури вбудовування, та відрізняється щодо процедури вибору множини корисних пікселів.

В даному випадку відправник секретного повідомлення обирає не відстань між вбудованими бітами, а діапазон, в якому ця відстань може знаходитися. Таким чином, навіть відправник секретного повідомлення не може знати в яких саме пікселях зберігаються біти його повідомлення. Це робить ще важчим виділення секретного повідомлення без знання параметрів стего-ключа на етапі витягання цифрового водяного знаку з контейнера.

2.3 Зональний стеганографічний метод

Зональний метод має значні переваги порівняно з зазначеними вище базовими методами. Метод є стійким до спотворень контейнеру. Стійкість обумовлена розбиттям зображення на блоки, такої самої розмірності, як і блоки, що використовуються при стиску зображень.

В межах методу один біт цифрового водяного знаку вбудовується в блок пікселів, розміром вісім на вісім пікселів. В результаті цього, метод забезпечує менший ефективний обсяг для цифрового водяного знаку. Однак, при використанні зображень великих розмірів цей недолік не є критичним.

Даний метод дозволяє регулювати співвідношення між захищеністю від спотворень та візуальною непомітністю результатів вбудовування цифрового водяного знаку в зображення-контейнер.

Послідовність дій методу наступна. Зображення-контейнер розбивається на блоки розмірністю вісім на вісім пікселів. Кожен біт цифрового водяного знаку вбудовується в такий блок. Для цього для

кожного блоку створюється маска $\mu(x, y)$, яка має розмірність, що співпадає з розмірінстью блоку зображення. Зазначена маска заповнюється псевдовипадковими двійковими значеннями, що розміщуються в ній.

Пікселі кожного з блоків B в залежності від двійкових значень маски поділяються на дві підмножини B_1 та B_2. Біти, яким відповідає нульовий елемент маски відносяться до першої підмножини, біти, яким відповідає одиничні елементи відносяться до другої підмножини.

Для кожної з підмножин розраховуються середні значення яскравості λ_1 та λ_2. Вбудовування біту m_i цифрового водяного знаку в блок зображення виконується відповідно до співвідношення (2.1).

$$\begin{cases} \text{якщо } m_i = 1 \text{ то } \lambda_1 - \lambda_2 > E; \\ \text{якщо } m_i = 0 \text{ то } \lambda_1 - \lambda_2 < -E, \end{cases} \tag{2.1}$$

де E – значення порогу вбудовування, тобто задана різниця між вказаними середніми значеннями яскравостей підблоків.

Поріг задає ступінь стійкості стего-контейнеру до спотворень. Зі збільшенням значення порогу підвищується стійкість контейнеру, однак збільшується спотворення зображення.

У випадку, коли умова (2.1) не виконується, здійснюється модифікація значень однієї з підмножин пікселів (B_1 чи B_2), таким чином, щоб умова виконувалася. При цьому головним критерієм якості цієї модифікації є мінімізація спотворень результуючого зображення-контейнера. Кінцевою метою змін значень пік селів підмножини є приведення їх значень, що задовольняють вираз (2.1).

Для витягання біту m^*_i цифрового водяного знаку виконуються аналогічні кроки з тими, що виконувалися при вбудовуванні. При цьому: зображення поділяється на блоки; виконується формування масок для

кожного з блоків; виконується розділення кожного з блоків на підмножини B_1 та B_2 відповідно до маски; здійснюється розрахунок середніх значень яскравості підмножин B_1 та B_2 (λ^*_1 та λ^*_2).

Після зазначених дій, різниця між отриманими значеннями λ^*_1 та λ^*_2 дозволяє отримати значення прихованого розряду цифрового водяного знаку за правилом:

$$m_i^* = \begin{cases} 1 \text{ якщо } \lambda_1^* - \lambda_2^* > 0; \\ 0 \text{ якщо } \lambda_1^* - \lambda_2^* < 0; \\ ? \text{ якщо } \lambda_1^* - \lambda_2^* = 0, \end{cases} \quad (2.2)$$

де знак питання (?) означає невстановлене значення біту, яке виникає при такому спотворенні зображення, яке викликає нульову різницю між яскравістю підмножин B_1 та B_2.

2.4 Висновки

В даному розділі кваліфікаційної роботи виконано постановку завдання роботи. Сформульовано мету та задачі роботи, призначення розробки, описано вхідні та вихідні потоки даних розроблюваного веб-сервісу.

Також в даному розділі роботи проведено аналіз методів стеганографічного вбудовування цифрових водяних знаків у растрові зображення. Визначено переваги та недоліки кожного з проаналізованих методів, галузі їх використання та доцільність їх реалізації в межах розробки веб-ресурсу.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ ВЕБ-СЕРВІСУ ВБУДОВУВАННЯ ДАНИХ В РАСТРОВІ ЗОБРАЖЕННЯ

3.1 Модуль вбудовування даних

Модуль є складовою частиною розробленого програмного модулю стеганографічного приховування даних. Він забезпечує вбудовування приховуваної інформації в зображення трьома методами та функціонує відповідно до діаграми використання, наведеної на рис. 3.1.

3.1.1 Розробка інтерфейсу модуля вбудовування даних

У віконці форми даного модуля розміщено дві групи перемикачів типу RadioButton. Перша група містить три перемикачі із назвами Red, Green та Blue, якими користувач обирає кольорову компоненту для вбудовування. У другій групі розміщено ще три перемикачі, які дозволяють обрати необхідний алгоритм вбудовування: Manual step LSB, Random step LSB та Zonal method. При обрані певного алгоритму активується відповідна йому панель, яка містить необхідні для передачі вхідних параметрів елементи управління. Для першого методу передбачено єдине текстове поле, в яке заноситься необхідний крок вбудовування біт даних. Панель другого методу містить три текстових поля: для числа ініціалізації, нижньої межі інтервалу та верхньої межі інтервалу. Для третього методу необхідні два текстових поля, в перше з яких заноситься число ініціалізації для генерації маски, а в друге – константа порогу E.

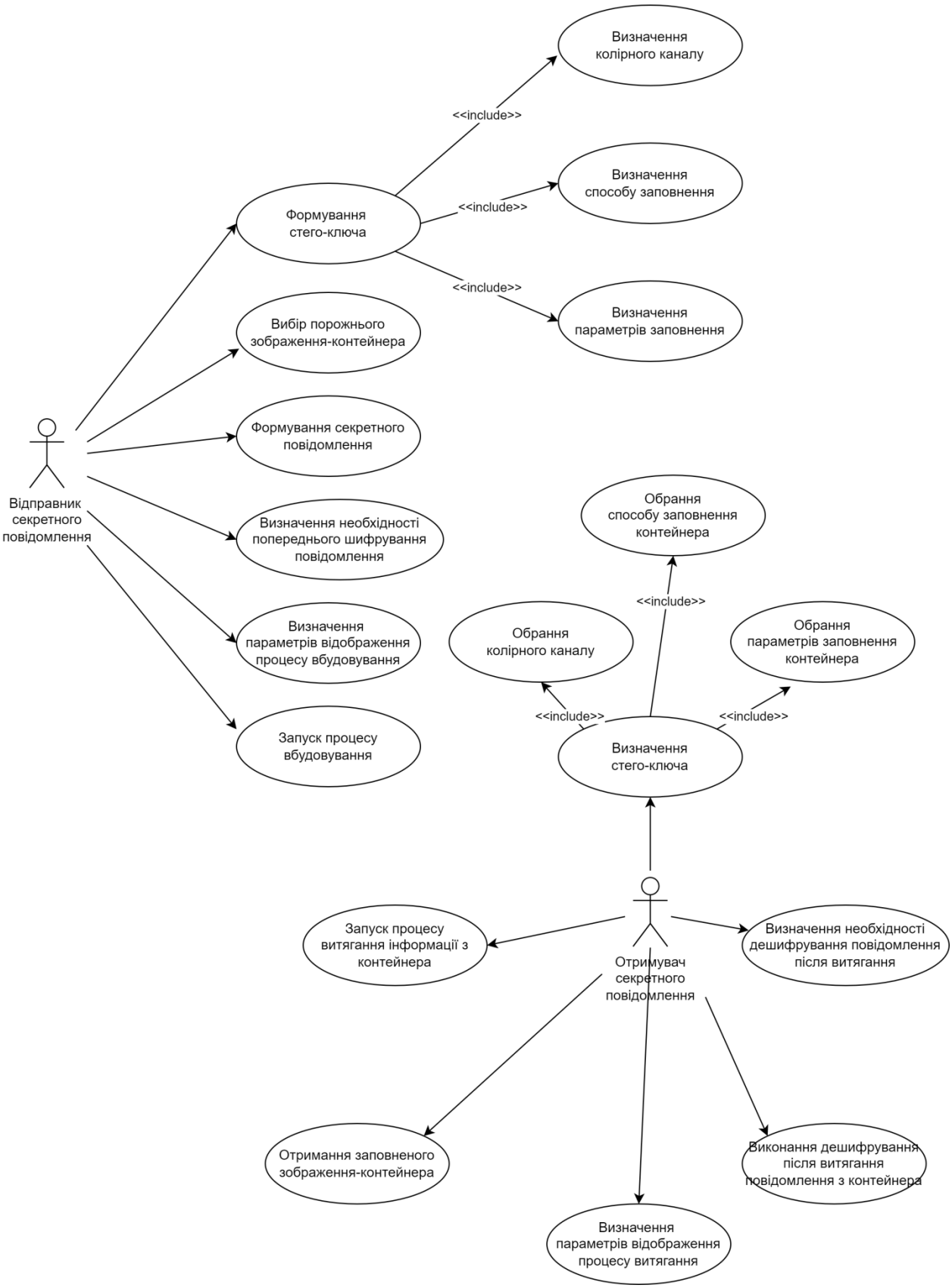

Рисунок 3.1 – Діаграма використання функціональносте розроблюваного веб-сервісу

Останнє поле на формі приймає текст повідомлення, яке має бути вбудовано, якщо буде натиснута відповідна кнопка. Загалом у віконці розміщується три кнопки:

- кнопка Load image;
- кнопка Hide text;
- кнопка Hide file.

Кнопка Load image викликає діалог завантаження растрового зображення-контейнеру з пам’яті комп’ютера до контейнеру на формі. Пошук зображення на жорсткому диску відводиться користувачу, який здатен обрати будь-яке задовольняюче його зображення у форматі bmp через провідник. Після того як зображення було обрано дві інші кнопки активуються. Всі дії виконуються програмним методом btnLoad_Click.

Після натискання кнопки Hide text, що змушує програмний модуль приховати у зображення повідомлення з текстового поля, виникає подія, на яку реагує програмний метод btnHide_Click. Перш за все він перевіряє наявність тексту в текстовому полі. Якщо він відсутній, то на екран виводиться повідомлення, яке попереджає про необхідність вводу тексту, інакше – виконання програми продовжується. Далі має місце перевірка маркеру на відповідність заданим вимогам (мінімум 4 символи). Якщо перевірка не пройдена, то на екран виводиться повідомлення з відповідним попередженням, інакше – виконання продовжується.

Наостанок даний метод обирає спосіб, яким буде проведено приховування, згідно вибору користувача. Можливий один з трьох варіантів:

- метод заміни молодшого значущого біту з фіксованим кроком вбудовування біт;
- метод заміни молодшого значущого біту з псевдовипадковим кроком вбудовування біт;
- зональний стеганографічний метод.

Задля вбудовування тексту з файлу існує кнопка Hide file, натискання якої викликає програмний метод btnFile_Click, сигнатура якого повністю відповідає сигнатурі btnHide_Click.

Цей метод так само перевіряє введений у відповідне текстове поле маркер, а після цього викликає діалог вибору текстового файлу. Після вибору потрібного файлу метод так само обирає один з трьох можливих способів приховування.

3.1.2 Реалізація вбудовування даних методом заміни молодшого значущого біту з фіксованим кроком

Якщо було обрано даний підхід, то викликається метод LSBCheckManual, який має два вхідні параметри: строка input, яка має бути прихована, та строка marker. Вихідних інформаційних параметрів даний метод не має.

Даний метод, головним чином, перевіряє чи підходять розміри обраного зображення для приховування даного текстового повідомлення разом зі службовою інформацією. В першу чергу, при виконанні перевірки, завантажене користувачем зображення заноситься з контейнера на формі до змінної bm класу Bitmap, після чого визначається його загальна кількість пікселів шляхом множення висоти зображення на ширину, значення яких отримуються завдяки таким властивостям Bitmap, як Height та Width. Отриманий результат заноситься до змінної ImageSizeInPixels цілочисельного типу int.

Далі з текстового поля на формі до змінної step типу int заноситься зазначена величина кроку приховування – числа, яке визначає на якій відстані одид від одного будуть знаходитися пікселі з прихованою інформацією. І наостанок визначаються необхідні кількості пікселів для приховування тексту повідомлення, його довжини та маркеру, що заносяться до змінних TextSizeInBits, LengthSizeInBits та MarkerSizeInBits

відповідно. Значення змінної LengthSizeInBits не залежить від повідомлення та складає 32 біти помножені на обраний крок. Значення змінних TextSizeInBits та MarkerSizeInBits визначається як добуток довжини строк input та marker, яка отримується завдяки використанню властивості Length, та обраного кроку вбудовування.

Після того як всі змінні отримали свої значення, вони використовуються для перевірки відповідності розмірів зображення-контейнеру потребам програмного модулю для приховування обраного повідомлення із зазначеними параметрами. Перевірка відбувається з використанням оператору if, що встановлює чи є значення змінної ImageSizeInPixels меншим за суму значень змінних TextSizeInBits, LengthSizeInBits та MarkerSizeInBits. Якщо результат даної перевірки позитивний, то зображення замале для приховування із заданими параметрами і на екран виводиться відповідне повідомлення за допомогою статичного методу Show() класу MessageBox, інакше – метод LSBCheckManual переходить до виконання своїх останніх дій, які зводяться до заповнення масиву manualStep, який має розмірність що співпадає з кількістю біт в повідомлені, цілочисельними значеннями кроку та виклику методу LSBHide з передаванням йому змінних bm, input, marker та manualStep в якості вхідних параметрів.

Метод LSBHide має 4 вхідних параметри: растрове зображення bm класу Bitmap, строку inputString типу string, строку marker, що також має тип string, та масив step, що містить значення типу int. Він перевіряє яку саме кольорову компоненту обрав користувач за допомогою операторів if, що визначають яка з кнопок типу RadioButon (rbtnRed, rbtnGreen чи rbtnBlue) натиснута, та викликає метод LSBHideRed, LSBHideGreen чи LSBHideBlue у відповідності до результатів перевірки. До методу передаються усі вхідні змінні методу LSBHide. Після відпрацювання

обраного методу приховування даних, управління повертається до методу LSBHide, який виводить заповнене інформацією зображення-контейнер на екран, розміщуючи його в контейнер класу PictureBox, та викликає метод ShowDialog() класу SaveFileDialog, який реалізує діалог збереження на жорсткий диск. Користувач сам обирає ім’я новоствореного зображення та місце його розташування на комп’ютері.

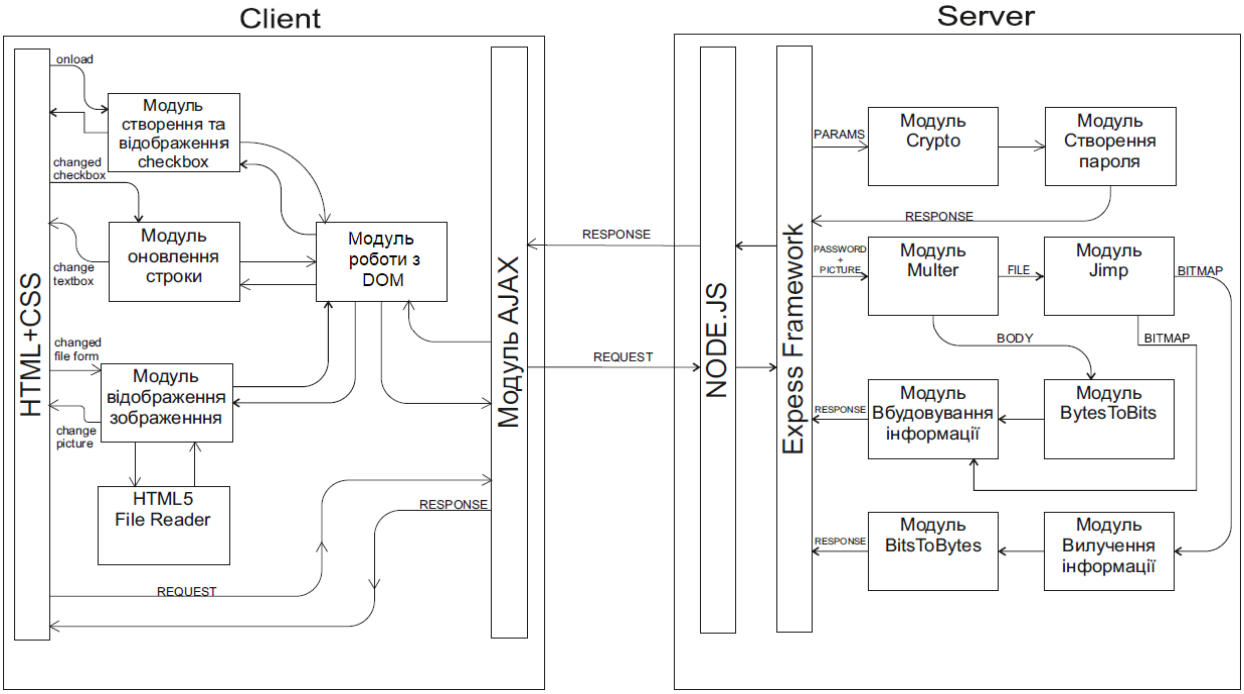

Рисунок 3.2 – Структура розроблюваного веб-сервісу
на стороні клієнта та сервера

Методи LSBHideRed, LSBHideGreen та LSBHideBlue повністю ідентичні, проте виконують свої функції відносно червоної, зеленої та синьої кольорової компоненти відповідно. Їх сигнатура співпадає із сигнатурою методу LSBHide. Ці методи виконують підготовчі дії для роботи методів LSBHideArrayRed, LSBHideArrayGreen та LSBHideArrayBlue відповідно, а саме створюють потрібні змінні та передають їх у вірному порядку. Серед створюваних змінних

найважливішими є messageArray, markerArray та lengthArray класу BitArray, що представляють текст повідомлення, маркер та довжину повідомлення у вигляді масивів біт. Перші два утворюються з використанням методу StringToBitArray власної розробки, який перетворює строку у масив біт. Опис принципів його роботи наведено в одному з наступних підрозділів. Масив lengthArray створюється стандартним конструктором класу BitArray.

Приховуване повідомлення складається з трьох частин: маркеру, довжини тексту, та власне тексту. Через це, методи LSBHideArray викликаються тричі – по разу для вбудовування кожного з масивів BitArray.

Методи LSBHideArrayRed, LSBHideArrayGreen та LSBHideArrayBlue приймають на вхід растрове зображення bm класу Bitmap, масив біт array класу BitArray, масив step, що містить значення типу int, а також три змінні типу int, що передаються по посиланню та потрібні для організації циклів обробки пікселів. Вихідних параметрів немає.

Власне, за один виклик метод приховує в передане змінною bm зображення масив біт array із зазначеним у step кроком. Для цього в ньому організовується 2 цикли. Зовнішній виконує контроль невиходу за межі зображення при обробці, в той час як внутрішній (вкладений) займається самою обробкою. У вкладеному циклі певний піксель зчитується до змінної col класу Color за допомогою методу GetPixel() класу Bitmap, який викликається на змінній bm та отримує на свій вхід координати необхідного пікселю. Далі до змінної b типу byte заноситься значення яскравості обраної кольорової компоненти з використанням відповідної властивості класу Color.

Для подальшого функціонування алгоритму необхідно знайти значення молодшого біту знайденої на попередньому кроці яскравості. Для

цього використовується одна з властивостей двійкового представлення чисел, згідно якої парне число має нульовий молодший біт, а непарне – одиничний. Тому наступним кроком є перевірка парності отриманого значення b шляхом перевірки остачі від його ділення на 2 з використанням оператору if. Для отримання певного біту з масиву array використовується стандартний метод Get() класу BitArray. Після отримання результату можливо 4 варіанти:

- молодший біт нульовий і має бути нульовим;
- молодший біт нульовий, а має бути одиничним;
- молодший біт одиничний, а має бути нульовим;
- молодший біт одиничний і має бути одиничним.

Як легко помітити, в першому та четвертому випадку, тобто в половині ситуацій, біт вже вбудовано в зображення без погіршення якості. В другому випадку до змінної b додають одиницю, а в третьому – віднімають. Після цього піксель з новими значеннями яскравості перезаписується з використанням методу SetPixel() класу Bitmap.

На цьому вбудовування методом заміни молодшого значущого біту з фіксованим кроком завершено.

3.1.3 Реалізація вбудовування даних методом заміни молодшого значущого біту з псевдовипадковим кроком

При використанні даного підходу першим викликається метод LSBCheckRandom. На його вхід приходять дві строкові змінні: input та marker. Вихідних змінних немає. Метод перевіряє чи достатньо велике зображення для приховування у ньому заданого повідомлення із зазначеним інтервалом величин кроків. Загальна кількість пікселів у зображенні, що представляється змінною bm класу Bitmap, рахується так само як і в попередньому підході – множенням довжини зображення на його ширину.

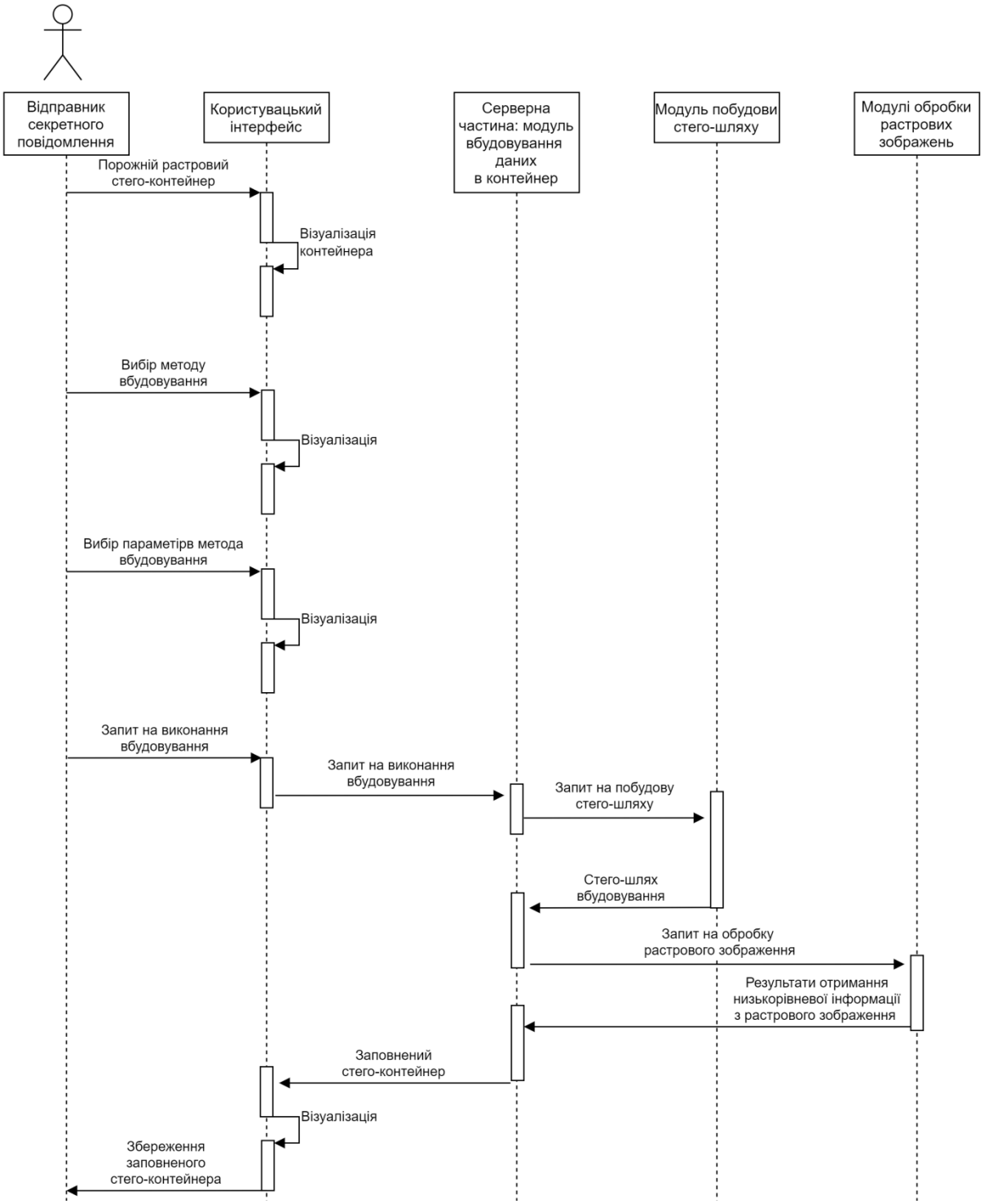

Рисунок 3.3 – Діаграма послідовності функціонування веб-сервісу в процесі вбудовування даних

Для визначення необхідної кількості пікселів використовується інша схема. Спершу створюється об’єкт класу Random із назвою r за допомогою

конструктору класу, якому передається вказане користувачем у відповідному текстовому полі ініціалізуюче значення. Далі значення верхньої та нижньої границі заносяться із відповідних текстових полів до змінних типу int із іменами to та from відповідно. Також заводиться змінна fullSize типу int, яка прирівнюється нулю. Надалі вона міститиме шукану величину.

Задля генерації масиву псевдовипадкових кроків randomStep, який необхідний для подальшого виклику методу LSBHide, використовується конструкція, що виділяє пам’ять під стільки елементів типу int, скількома бітами представлено маркер, довжину та текст повідомлення. Варто відзначити, що довжина повідомлення задається константною кількістю біт – 32. Далі у циклі усі виділені на попередньому кроці елементи заповнюються псевдовипадковими значеннями, що генеруються завдяки викликанню стандартного методу Next класу Random на змінній r з вхідними параметрами from та to, збільшеними на одиницю.

Передостанньою дією методу є перевірка достатності розмірів зображення, при негативному результаті якої на екран виводиться інформуючи про це повідомлення, а при позитивному відбувається передача управління методу LSBHide, на вхід якого передаються змінні bm, input, marker та randomStep. Логіка роботи даного методу не відрізняється від описаної раніше.

3.1.4 Реалізація вбудовування даних зональним стеганографічним методом

При використанні зонального підходу першим викликається метод ZonalCheck. На його вхід приходять дві строкові змінні: input та marker. Вихідних змінних немає. Як і при попередніх підходах, першим кроком є занесення зображення з контейнера на формі до змінної bm класу Bitmap та підрахунок максимальної кількості біт, яка може бути в ньому прихована.

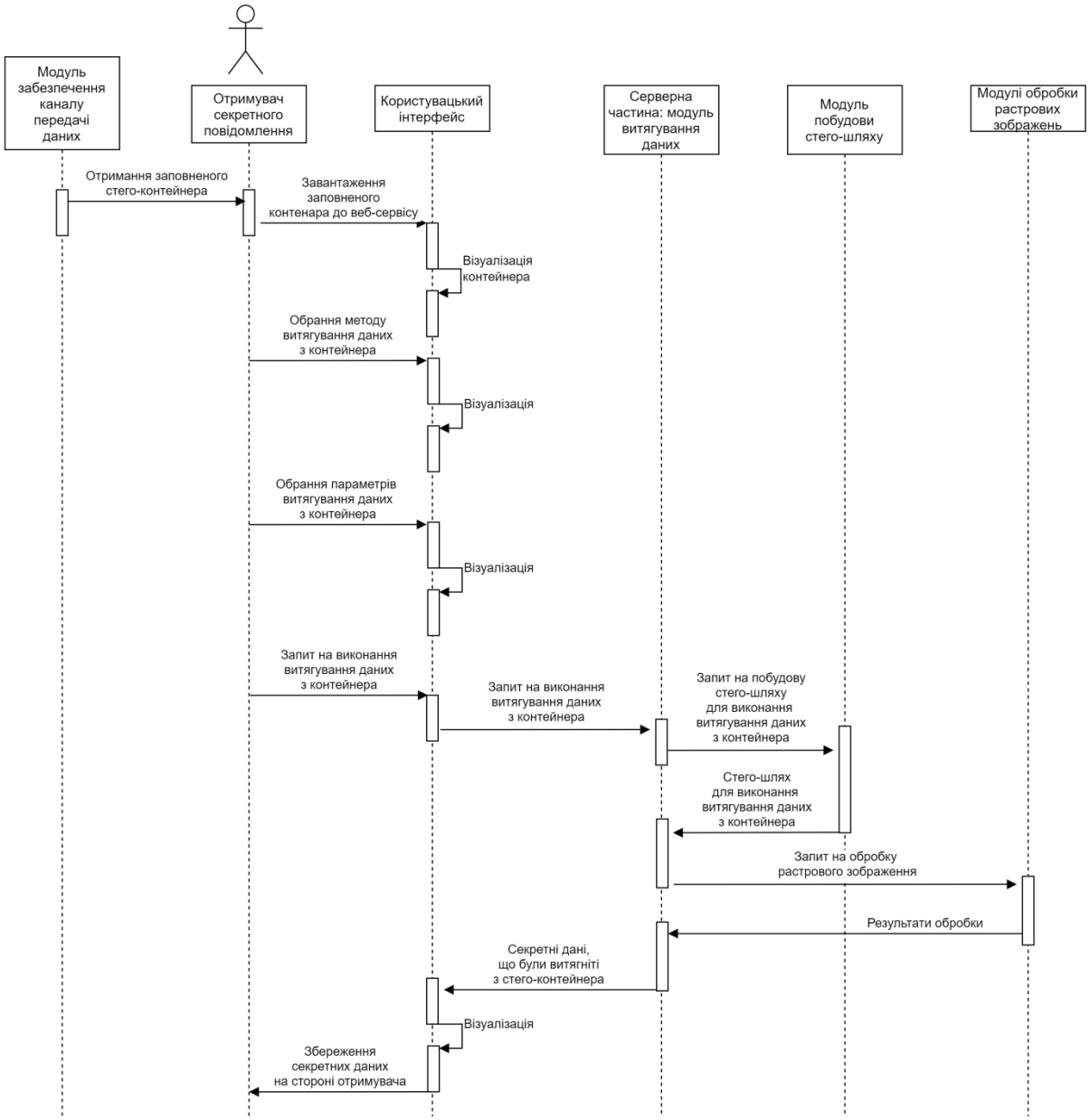

Рисунок 3.4 – Діаграма послідовності функціонування веб-сервісу в процесі витягання даних

Оскільки біти ховаються в блоки розмірністю 8 х 8, то рахується саме кількість таких блоків. Для цього від зображення спершу відтинаються невеликі смужки з правої сторони та знизу так, аби ширина та висота зображення стали кратні вісімці, а потім добуток приведених ширини та висоти ділиться на кількість пікселів в одному блоці. Отримана

величина записується до змінної ImageCapacity типу int. Фрагмент реалізації виглядає наступним чином:

```
int ImageCapacity = (bm.Height - (bm.Height % 8)) *
(bm.Width - (bm.Width % 8)) / 64;
```

Необхідні для приховування тексту та маркеру кількості блоків розраховуються як помножена на 32 довжина строки input та marker і заносяться до цілочисельних змінних TextSizeInBits та MarkerSizeInBits відповідно. Далі за допомогою оператору if виконується перевірка чи менше ImageCapacity за суму TextSizeInBits та MarkerSizeInBits, до яких додано ще 32 біти, які необхідні для приховування довжини повідомлення. Якщо так, то зображення не задовольняє потребам і на екран виводиться попереджувальне повідомлення, якщо ж ні – зображення підходить – управління передається методу ZonalHide з вхідними параметрами bm, input та marker.

Метод ZonalHide має 3 вхідних параметри: змінну bm класу Bitmap, що зберігає растрове зображення, а також дві строки типу string, що мають назви inputString та marker. Метод містить 3 оператори if, що визначають кнопка якої кольорової компоненти натиснута та викликають метод ZonalHideColor, якому передають 4 вхідні параметри. Перші три параметри є вхідними власне методу ZonalHide, а четвертим виступає строка "red", "green" чи "blue" в залежності від результатів перевірки. Після виконання дій методу приховування даних, управління повертається до методу ZonalHide і на екран виводиться заповнене інформацією зображення-контейнер та викликається метод ShowDialog() класу SaveFileDialog, який ініціює діалог збереження на жорсткий диск. За своїм призначенням даний

метод повністю співпадає з методом LSBHide, який використовується в описаному вище підході.

Метод ZonalHideColor має 4 вхідні параметри, три з яких повторюються з попереднього методу, а четвертий – змінна color, що має тип string. Метод умовно поділяється на дві частини, в першій з яких створюються та ініціалізуються необхідні змінні, а в другій – викликаються методи вбудовування даних.

Спершу створюється цілочисельна змінна block, якій присвоюється нульове значення. Далі заводиться інша цілочисельна змінна E, значення якої утворюється шляхом перетворення чисельної строки з текстового поля на формі, в яке користувач записує певне значення порогу, в значення типу int з використанням статичного методу Parse() класу Int32. Для створення та ініціалізації змінних messageArray та markerArray класу BitArray використовується, як і в розглянутому раніше алгоритмі приховування, метод власної розробки StringToBitArray. Для утворення першої змінної в якості вхідного параметру методу передається строкова змінна inputString, для другої – marker. Масив lengthArray утворюється стандартним конструктором, якому передається масив s значень типу int. Цей масив містить один елемент, значення якого відповідає кількості блоків, яка потрібна для приховування тексту повідомлення. Такий спосіб ініціалізації масиву обумовлений вимогами стандартного конструктору класу.

Після завершення підготовки змінних тричі викликається метод ZonalHideArray, який виконує вбудовування бітів масиву в блоки пікселів зображення. Він має 5 вхідних параметрів та один вихідний. Серед вхідних присутні: зображення-контейнер bm класу Bitmap, масив біт inputArray класу BitArray, що має бути вбудований, число startingBlock типу int, яке представляє номер блоку, з якого слід починати вбудовування, цілочисельне значення порогу E та строкове значення color, яке

представляє кольорову змінну, в яку слід вбудовувати дані. Вихідним параметром є змінна number типу int, яка вказує номер наступного вільного блоку.

В першу чергу створюються цілочисельні змінні number та index. Першій присвоюється значення startingBlock, а другій – нуль. Далі організовується загальний зовнішній цикл по змінній number з одиничним кроком, який закінчує свою дію після вбудовування кожного з бітів вхідного масиву inputArray. Всередині циклу першою дією є отримання координат блоку з номером number за допомогою методу власної розробки GetBlock, який приймає на вхід змінну типу Bitmap та змінну типу int, які відповідають зображенню та номеру блоку в зображенні, а на вихід видає двомірний масив значень типу Coords, який також розроблено власноруч. Масив записується до змінної block. Далі заводиться двомірний масив mask значень типу int, який отримується в результаті виклику ще одного методу власної розробки GetMask, якому передаються номер блоку та ініціалізуюче число з відповідного текстового поля на формі. Після цього у дворанговому циклі підраховується кількість одиниць в новоутвореній масці та на основі цього створюється два масиви значень типу Coords (array1 та array0), в перший заносяться координати тих пікселів блоку, яким в масці відповідають одиниці, а у другий – ті, яким відповідають нулі. Це виконується з використанням дворангового циклу по змінним i та j, які змінюють свої значення від 0 до 7, в змінних k та l зберігається кількість оброблених одиниць та нулів маски відповідно. Тіло даного циклу виглядає наступним чином:

```
if (mask[i][j] == 1) array1[k++] = block[i][j];
else array0[l++] = block[i][j];
```

Останньою дією у загальному циклі є виклик методу BrightnessCheck, якому передаються наступні дані: bm,

inputArray.Get(index++), array0, array1, color та E. Другий параметр представляє собою біт, який має бути вбудовано в даний блок. Після закінчення роботи цього циклу змінна number передається на вихід.

Метод BrightnessCheck приймає на свій вхід такі шість параметрів: растрове зображення bm, бульове значення b класу Boolean, два масиви array0 та array1 значень класу Coords, строкове значення color, та цілочисельне значення E. На вихід не передає нічого. Даний метод відповідає за те аби переданий біт було вбудовано.

Спершу створюються дві змінні типу int, що мають назви br1 та br2, які ініціалізуються викликом методу GetBrightness, що повертає їм значення середньої яскравості пікселів масиву, що йому передається, по кольоровій компоненті, яка також передається в якості вхідного параметру. Перша змінна зберігає в собі середню яскравість «нульового» масиву, а друга – «одиничного». Далі оператором if перевіряється вхідне бульове значення.

При значенні true організовується цикл while, який діє доки різниця яскравостей не буде більшою за значення змінної E. Якщо умова не виконується, то відбувається вхід до тіла циклу, яке складається з чотирьох дій. Перші дві – це виклики методу BrightnessChange, який в першому випадку збільшує середню яскравість пікселів «нульового» масиву на одиницю, а у другому – на одиницю зменшує середню яскравість «одиничного» масиву. Останні дві дії тіла циклу – отримання нових значень середньої яскравості масивів.

При значенні false також організовується цикл while, але він діє доки різниця середніх яскравостей не буде меншою за значення змінної E з протилежним знаком. Тіло циклу має аналогічну будову, проте в цьому випадку яскравість «нульового» масиву зменшується на одиницю, а «одиничного» – збільшується.

Метод BrightnessChange приймає чотири вхідні параметри: bm класу Bitmap, масив array значень класу Coords, строкове значення color типу string та цілочисельне значення value типу int. Вихідних параметрів немає. Метод складається з трьох ідентичних частин, кожна з яких обробляє певну кольорову компоненту. Потрібна кольорова компонента визначається за допомогою оператора if. В тілі оператору організовується цикл for, який виконується стільки разів, скільки в масиві присутньо елементів. На кожній ітерації змінна c класу Color отримує наступний піксель з масиву array внаслідок виклику методу GetPixel класу Bitmap. Далі виконується перевірка допустимості нового значення яскравості кольорової компоненти пікселю i, при її проходженні, піксель зі зміненою на вказану величину value яскравістю перезаписується.

3.2 Модуль виділення даних

Даний модуль є другою складовою частиною розробленого програмного модулю стеганографічного приховування даних. Він забезпечує виділення прихованої трьома методами інформації із зображення.

3.2.1 Розробка інтерфейсу модуля виділення даних

Інтерфейс даного під модулю відрізняється зовсім незначним чином. Це зумовлено тим, що обидва підмодулі працюють з одними алгоритмами, а отже мають ідентичні елементи управління для вводу вхідних параметрів. Головною відмінністю є наявність кнопки Extract text замість двох кнопок для приховування у підмодулі вбудовування даних. А також текстове поле, яке розміщено на тому ж місці, що й поле для вводу тексту

приховуваного повідомлення у підмодулі вбудовування, служить для виводу виділеної з контейнеру інформації.

При натисканні кнопки Extract text викликається метод btnExtract_Click, що має два вхідні параметри: sender класу object та e класу EventArgs. Вихідних параметрів немає. Першою дією методу є зчитування маркеру з текстового поля на формі до змінної marker типу string. Він має містити не менше чотирьох символів, що перевіряється за допомогою оператору if. Якщо умова не виконується, то на екран виводиться відповідне повідомлення і виділення даних треба починати спочатку. При задоволенні даної умови метод заносить зображення з форми до змінної bm класу Bitmap. Далі операторами if визначається обраний метод виділення прихованих даних, якому передається управління та 2 параметри: bm та marker.

3.2.2 Реалізація виділення даних методом заміни молодшого значущого біту з фіксованим кроком

При виборі цього підходу першим викликається метод LSBExtractManual. Даний метод отримує на вхід змінну bm класу Bitmap та marker типу string. В першу чергу створюється змінна step типу int, якій присвоюється обране користувачем значення кроку, що перетворюється зі строкового типу завдяки методу Parse класу Int32. Далі створюється масив цілочисельних значень manualStep, який заповнюється значеннями step. Наостанок викликається метод LSBExtract, якому передаються змінні bm, marker та manualStep.

Метод LSBExtract має 3 вхідних параметри та не має вихідних. Вхідними є: змінна bm класу Bitmap, змінна marker типу string та масив значень типу int із назвою step. У методі створюється змінна messageArray класу BitArray, в яку повертає значення обраний метод виділення даних з певної кольорової компоненти: LSBExtractRed, LSBExtractGreen чи LSBExtractBlue.

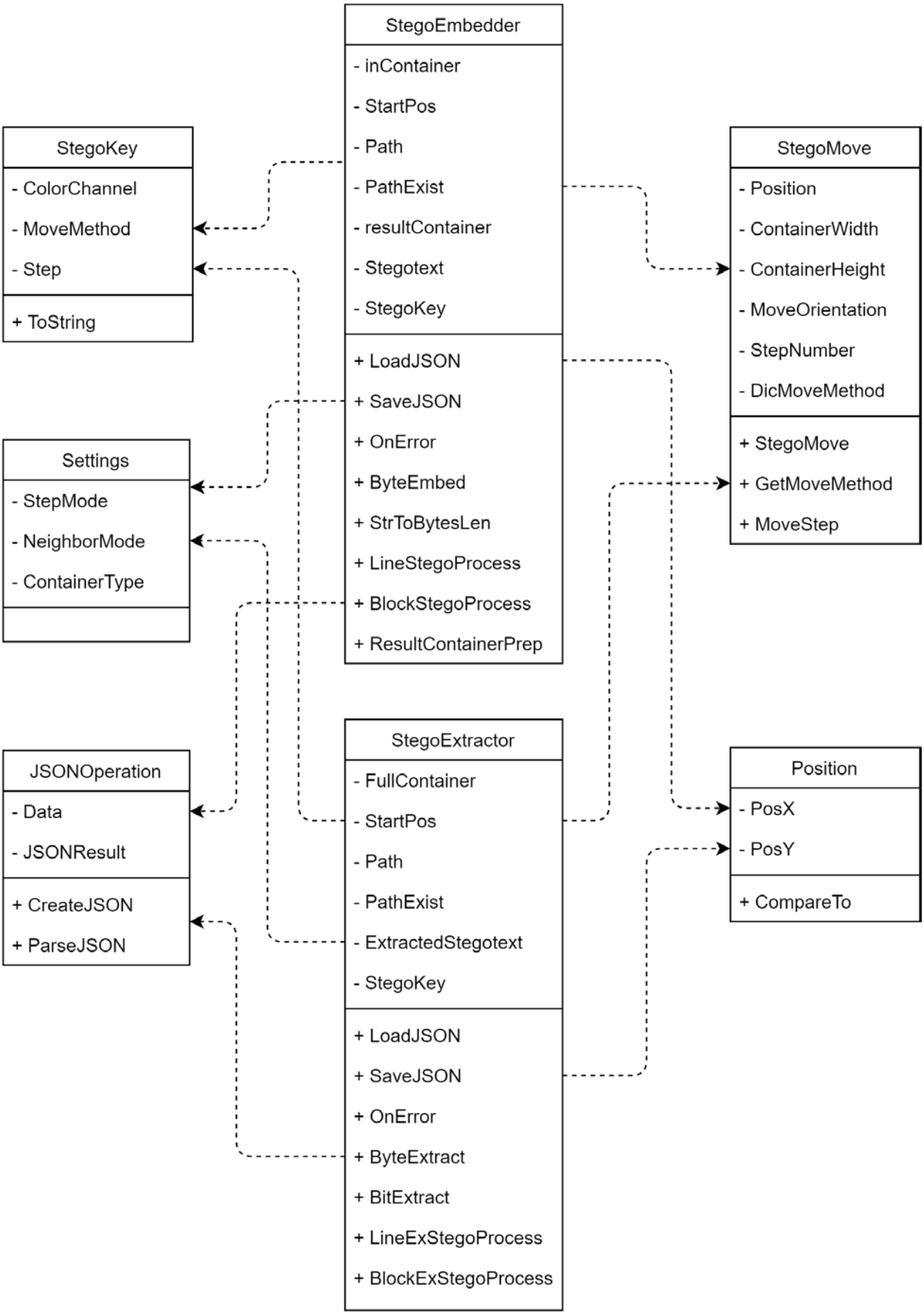

Рисунок 3.5 – Діаграма класів головного модуля веб-сервісу

Певний метод обирається згідно з вибором користувача, який перевіряється операторами if. Після отримання результату від методу викликається метод GetMessage із параметром messageArray.

Метод LSBExtractRed є одним з трьох методів виділення даних методом що розглядається. Він виділяє дані з червоної компоненти. Методи LSBExtractGreen та LSBExtractBlue виконують ті ж самі функції але для зеленої та синьої компонент відповідно. Даний метод отримує на свій вхід ті ж дані, що й попередній, а повертає змінну типу BitArray. Окрім створення необхідних для обробки зображення змінних i, j та t типу int, створюється й змінна markerArray класу BitArray з використанням стандартного конструктору. Для заповнення масиву викликається метод LSBExtractArrayRed.

Далі виконується перевірка вказаного користувачем маркеру на відповідність отриманому в змінній markerArray, який спершу перетворюється в змінну строкового типу методом власної розробки BitArrayToString. Якщо маркери не співпадають, то на екран виводиться відповідне повідомленням силами методу Show класу MessageBox та робота модуля завершується. Якщо ж маркери співпали, то робота методу продовжується створенням бітового масиву lengthArray потужністю в 32 значення стандартним конструктором. Для його заповнення використовується той самий метод LSBExtractArrayRed. Надалі отримані дані перетворюються в значення типу int завдяки розробленому методу BitArrayToInt, а воно, в свою чергу, використовується для створення масиву messageArray класу BitArray. Наостанок масив заповнюється ще одним викликом методу LSBExtractArrayRed, а далі отримане в змінній messageArray значення передається на вихід.

Метод LSBExtractArrayRed приймає на вхід наступні змінні: bm класу Bitmap, array класу BitArray, масив step значень int, та цілочисельні i,

j та t по посиланню. Вихідних параметрів немає. Він відповідає за власне отримання значень молодших значущих бітів певної кольорової компоненти певного пікселю.

Метод складається з основного зовнішнього циклу, що контролює невихід змінної j, по якій ітерується внутрішній цикл, за межі допустимих значень. Тіло внутрішнього циклу складається з декількох простих дій. Перша – отримання пікселю методом GetPixel класу Bitmap та його запис у змінну col класу Color. Друга – занесення значення червоної кольорової компоненти до змінної b типу byte. Третя – перевірка оператором if залишку від ділення змінної b на 2, тобто перевіряється значення молодшого біту даної змінної. Четверта і остання дія – додавання отриманого значення до масиву array стандартним методом Set класу BitArray.

3.2.3 Реалізація виділення даних методом заміни молодшого значущого біту з псевдовипадковим кроком

Єдиною різницею в реалізації даного підходу отримання даних є використання іншого підготовчого методу. При обрані цього способу першим методом викликається LSBExtractRandom замість LSBExtractManual. Даний метод має аналогічну сигнатуру та кінцеву ціль – створення масиву кроків.

Спершу створюється змінна r класу Random, яка ініціалізується стандартним конструктором класу. Конструктору передається цілочисельне значення ініціалізаційного числа, яке зчитується з форми та перетворюється в потрібний тип методом Parse класу Int32. Таким же чином записуються значення з відповідних текстових полів форми до змінних from та to типу int. Масив randomStep містить максимально можливу кількість кроків для повного заповнення зображення-контейнеру. Він заповнюється в циклі викликом методу Next з вхідними параметрами

from та to на змінній r для кожного елементу. Після цього викликається метод LSBExtract з вхідними параметрами bm, marker та randomStep. Подальші операції визначаються даним методом як і в попередньому підході.

3.2.4 Реалізація виділення даних зональним стеганографічним методом

Першим методом, що використовується за даного підходу, є ZonalExtract. Він, як і аналогічні йому попередні, приймає на свій вхід змінні bm класу Bitmap та marker типу string. Вихідних параметрів немає. Метод визначає вказану користувачем кольорову компоненту, та передає її назву та змінні bm і marker методу ZonalExtractColor, який, в свою чергу, є вхідним параметром методу GetMessage. Даний фрагмент реалізації для червоної компоненти виглядає наступним чином:

```
GetMessage(ZonalExtractColor(bm, marker, "red"));
```

Метод ZonalExtractColor приймає на свій вхід ті ж параметри що й попередній, а також строкову змінну color, яка визначає кольорову компоненту, з якої виділяються необхідні дані, а повертає змінну класу BitArray. В ньому створюються змінні lengthArray, markerArray та messageArray класу BitArray, яким присвоюється значення null, а також block та amount типу int, першій з яких присвоюється нульове значення, а другій – довжину маркеру помножену на 32.

Для зчитування маркеру методу ZonalExtractArray передаються змінні bm, block, amount та color, а значення, що метод повертає, записується до змінної markerArray. Далі оператором if перевіряється відповідність уведеного на формі маркеру отриманому та перетвореному на строку методом BitArrayToString. При негативному результаті на екран

виводиться відповідне повідомлення, при позитивному – зчитування продовжується.

Перш ніж зчитати довжину повідомлення, значення змінної block збільшується на значення змінної amount, якій після цього присвоюється значення 32. Масив lengthArray заповнюється методом ZonalExtractArray, який отримує змінні з оновленими значеннями. Після цього змінна block знову збільшується на amount. Зчитування тексту повідомлення в змінну messageArray виконується все тим самим ZonalExtractArray, який отримує ті ж змінні окрім amount, замість якої передається приведене методом BitArrayToInt до типу int значення lengthArray. Завершується робота методу поверненням змінної messageArray в якості вихідного параметру.

Метод ZonalExtractArray приймає чотири вхідні параметри: змінну bm класу Bitmap, змінні startingBlock та amount типу int та змінну color типу string. Вихідним параметром є змінна класу BitArray.

Створюється змінна number типу int, що ініціалізується значенням змінної startingBlock та index – нульовим значенням. Також створюється змінна ba класу BitArray, яка ініціалізується стандартним конструктором з параметром amount. Далі організовується цикл на amount ітерацій. В його тілі виконуються всі дії по отриманню біту з певного блоку пікселів. Спершу сторюється двовимірний масив block значень типу Coords, який ініціалізується методом GetBlock з параметрами bm та number. Далі створюється двовимірний масив mask, що ініціалізується методом GetMask. Після цього у дворанговому циклі підраховується кількість одиниць у масці. На основі отриманого значення утворюються масиви array1 та array0 значень класу Coords. Далі організовується ще один дворанговий цикл, в якому відповідні масиви заповнюються координатами відповідних пікселів яким відповідають одиничні та нульові значення маски. Потім заводяться дві змінні типу int із назвами br1 та br2, які

отримують значення від методу GetBrightness із параметрами bm, array0 та color для першої та bm, array1 та color для другої змінної. Наостанок ітерації до масиву ba додається бульове значення true якщо різниця значень br1 та br2 більша за нуль або false в протилежному напрямку. Якщо змінні мають однакове значення, то значення біту встановити неможливо. Після відпрацювання всіх ітерацій масив ba повертається в якості результату.

3.3 Розробка допоміжних програмних методів

При реалізації програмних підмодулів вбудовування та виділення даних виникла потреба у виконанні ідентичних дій на обох сторонах прихованого каналу зв’язку. Вони реалізовані у вигляді допоміжних методів, опис яких наведено нижче.

Метод GetBrightness приймає на свій вхід 3 вхідні параметри: зображення, масив координат пікселів та кольорову компоненту. В якості вихідного параметру він видає розраховану середню яскравість. Після того як потрібний колір встановлено, організовується цикл, який виконується допоки усі пікселі масиву не будуть оброблені. В тілі циклу спершу зчитується наступний піксель з масиву, а потім яскравість потрібної кольорової компоненти шумується до зовнішньої змінної, якій попередньо було надано нульове значення. Після отримання загальної суми яскравості пікселів та закінчення роботи циклу, зазначена змінна ділиться на кількість пікселів у масиві. Таким чином отримується середня яскравість, яка передається вихідним параметром.

Метод GetBlock приймає як вхідні параметри зображення та номер блоку, а видає двовимірний масив координат пікселів цього блоку.

Координати містяться в об’єктах класу Coords. Для реалізації використано два дворангових цикли, перший з яких створює масив, а другий заповнює його значеннями координат. Загалом реалізація досить тривіальна і не потребує більшої деталізації.

Метод GetMask приймає два вхідні параметри: number типу int та constant типу int. Повертає він двовимірний масив значень типу int, який грає роль маски. Спершу створюється двовимірний масив 8 х 8 з нульовими значеннями, а потім він заповнюється псевдовипадково генерованими нулями та одиницями з використанням об’єкту r класу Random, який ініціалізується стандартним конструктором класу, що отримує на вхід суму значень змінних constant та number в процесі вбудовування даних.

Метод GetMessage має єдиний вхідний параметр – змінну messageArray класу BitArray, яка містить біти повідомлення, які перетворюються методом на строку та виводяться у спеціальне текстове поле на формі. В першу чергу виконується перевірка того, щоб масив не був порожнім. Якщо вона пройдена, то далі перевіряється наявність в ньому елементів та кратність їх кількості 32. При подоланні і другої перевірки створюється змінна result типу string, значення якій надає метод BitArrayToString, що отримує змінну messageArray на вхід. А далі вона виводиться у відповідне текстове поле на формі завдяки його властивості Text, якому присвоюється текст зі змінної. Якщо якась із перевірок не була пройдена, то на екран виводиться повідомлення про спотворення повідомлення.

Метод StringToBitArray перетворює строкову змінну str, яка йому передається у масив біт класу BitArray, який повертається запитувачу. В тілі методу в першу чергу виконується генерація об’єкту encoding класу UTF32Encoding стандартним конструктором без вхідних параметрів. Далі

вхідні строкові дані перетворюються в масив байт b за допомогою стандартного методу класу UTF32Encoding, що має назву GetBytes() та викликається на новоствореному об’єкті даного класу. І наостанок масив байтів перетворюється на масив бітів ba класу BitArray стандартним конструктором даного класу при використанні функцій перетворення системи числення.

Метод BitArrayToString перетворює змінну ba класу BitArray, що передається на його вхід, у строку, яка є кінцевим результатом роботи методу. Метод є протилежністю описаному вище. В першу чергу виконується генерація об’єкту encoding класу UTF32Encoding стандартним конструктором без вхідних параметрів. Далі створюється масив b значень типу byte, який має потужність у 8 раз нижчу за потужність вхідного масиву. Далі дана змінна заповнюється стандартним методом CopyTo класу BitArray, що викликається на змінній ba. В якості параметрів йому передаються змінна b в якості масиву, який потрібно заповнити, та нуль, як індекс елементу, з якого слід почати. Останньою дією є перетворення масиву байт на строку і запис її до змінної str методом GetString викликаним на об’єкті encoding в процесі вбудовування цифрового водяного знаку до зображення.

Метод BitArrayToInt перетворює ba класу BitArray, що передається на його вхід, у змінну типу int. Оскільки метод потрібен лише для перетворення 32 біт в одне цілочисельне значення, то при кількості біт більшій за 32 генерується виключення типу ArgumentException. Далі створюється масив array значень типу int на один елемент, який заповнюється як в попередньому методі. Якщо значення утвореного елементу більше за нуль, то воно повертається в якості результату, інакше знав обчислюється нове значення, яке перевіряється на те, чи є воно більшим за нуль.

3.4 Тестування розроблених модулів веб-сервісу

Розроблений веб-сервіс має інтуїтивний інтерфейс та дозволяє користувачу з легкістю вбудовувати повідомлення в растрові зображення на будь-якій мові та з використанням будь-яких символів. Для перевірки даного твердження в рамках дипломної роботи проводиться тестування створеного програмного забезпечення.

3.4.1 Тестування модуля приховування

Для того щоб приховати текстове повідомлення потрібно мати зображення-контейнер та вигадати ключові параметри. Для тестування позитивного сценарію візьмемо достатнє за розмірами зображення в форматі bmp.

Перш за все запускається розроблений модуль. Все що можна зробити, це натиснути на кнопку Load image. Після її натискання виникає діалог вибору растрового зображення. Коли зображення обрано та натиснута кнопка Ок, діалог закривається, а зображення у зменшеному вигляді з’являється у верхньому лівому кутку.

Настає черга введення ключових параметрів. Послідовність заповнення полів немає значення, проте одним з перших слід обирається алгоритм вбудовування, адже без цього потрібні елементи управління не будуть доступні.

Наприклад, ставиться крапка поруч з зональним методом (Zonal method), що розблоковує нижню панель з двома текстовими полями. До верхнього поля, в якому має бути число ініціалізації, заноситься будь-яке ціле число. Ввести щось окрім цифр не вийде, адже дане поле, як і усі інші числові поля, захищені від некоректного вводу. До нижнього поля, поля

константи межі E, яка відповідає за стійкість заповненого контейнеру до спотворень, заноситься будь-яке число в інтервалі [-255;255]. Чим більше за модулем число, тим більшою стійкістю до спотворень буде наділений контейнер, проте й його відхилення від початкового вигляду будуть кардинальнішими.

Тепер заповнюється стандартна для всіх алгоритмів частина параметрів. В поле маркеру вводиться будь-яка цифро-літерна послідовність, що містить не менше чотирьох символів. На черзі обрання однієї з трьох кольорових компонент. Вона обирається так само як і алгоритм приховування – за допомогою кліку на відповідному перемикачу типу RadioButton.

Останнім підготовчим кроком є підготовка тексту приховуваного повідомлення. Якщо має приховуватися текст з файлу, то такий файл спершу створюється чи перевіряється та зберігається десь у доступній директорії. Потім натискається кнопка Hide file і на екрані з’являється, схожий на діалог обрання зображення, діалог вибору текстового файлу. Після цього програма переходить до фази власне вбудовування.

При роботі з текстом у текстовому полі, то повідомлення заноситься туди та натискається кнопка Hide text, що ініціює відпрацювання методів вбудовування. З якого б джерела не брався текст, після закінчення вбудовування заповнене зображення-контейнер з’являється у верхньому правому кутку. Якщо параметри були обрані вірно, то неозброєним оком різницю між правим та лівим зображенням помітити неможливо.

Одночасно із появою зображення у віконці, з’являється діалог його збереження у постійній пам’яті. Після надання зображенню імені, обрання директорії розташування та натискання кнопки Ок модуль завершує свою роботу.

3.4.2 Тестування модуля виділення

Для того щоб виділити текстове повідомлення з растрового зображення потрібно мати відповідний заповнений контейнер та ключові параметри, з якими повідомлення було вбудоване. Для перевірки даного модуля повідомлення виділяється із зображення з попереднього пункту.

Після запуску програми першою справою є завантаження зображення. Для цього слугує така сама як і в приховувачі кнопка Load image. В діалозі обирається заповнений контейнер та натискається кнопка Ок. Далі обирається той самий зональний метод, а в активоване поле заноситься те ж саме число ініціалізації.

Далі так само обирається кольорова компонента та вводиться маркер. Якщо все зроблено вірно, то при натисканні кнопки Extract text, в текстовому полі поруч із кнопкою з’являється приховане повідомлення.

3.5 Висновки

В даному розділі кваліфікаційної роботи виконано проектування та розробку на рівні програмного коду веб-сервісу для вбудовування цифрових водяних знаків у растрові зображення. Для цього було розроблену структуру веб-сервісу, діаграму класів його головного модуля, діаграми використання та послідовності, а також виконано розробку програмного коду веб-сервісу.

ВИСНОВКИ

В кваліфікаційній роботі виконано аналітичний огляд напрямів захисту інформації. Серед них виділено цифрову стеганографію. Визначено переваги та недоліки стенографічного підходу до захисту даних. Проаналізовано потенційні галузі застосування стеганографії. Відмічено один з прикладних напрямів теорії стеганографії – технологію цифрових водяних знаків, яка використовується для захису мультимедійного контенту.

Проведено аналіз методів стеганографічного захисту даних, зокрема методів приховування даних у просторовій області растрових зображень. Виконано аналіз аналогів веб-сервісу для вбудовування цифрових водяних знаків у растрові зображення, що розробляється в кваліфікаційній роботі та порівняти очікувані характеристики веб-ресурсу з характеристиками аналогів. Обрано платформу та середовище розробки, а також мову програмування, з використанням яких проводитиметься розробка веб-сервісу, і виконати обґрунтування вибору. Розроблено алгоритми та формальні описи функціонування модулів веб-сервісу для вбудовування цифрових водяних знаків у растрові зображення. Виконано розробку зазначеного веб-сервісу. Проведено тестування отриманого веб-сервісу за результатами якого довести досягнення мети роботи.

ПЕРЕЛІК ПОСИЛАНЬ

1. Shih F. Digital Watermarking and Steganography: Fundamentals and Techniques. 2nd ed. USA, Boca Raton: CRC Press, 2017. 420 p.

2. Cox, I., Miller, M., Bloom, J., Fridrich, J. Digital Watermarking and Steganography. Amsterdam: Morgan Kaufmann Publishers, 2018. 594 p.

3. Fridrich J.: Steganography in Digital Media. New York: Cambridge University Press, 2010. 438 p.

4. Конахович Г.Ф. Компьютерная стеганография. Теория и практика. МК-Пресс, 2006. 288 с.

5. Хорошко В.А. Введение в компьютерную стеганографию. Киев: Національний Авіаційний Університет, 2002. 152 с.

ДОДАТОК А

Програма розробленого веб-сервісу

```
const jimp = require("jimp");
const fs = require("fs");
const rndGenerator = require("random-seed").create();
const BITS_PER_CHAR = require("./config").config.BITS_PER_CHAR;
const BITS_FOR_MESSAGE_LENGTH =
require("./config").config.BITS_FOR_MESSAGE_LENGTH;

function LSBemb(imageFilename, message, key, em) {
   const sourcePath = __dirname + "/../uploads/" + imageFilename;

   jimp.read(sourcePath, function (err, image) {
      if (err) throw err;

      if(sizeEstimate(image, message, key) === false){
         em.emit("imageTooSmall");
         return;
      }

      const messageBinArr = stringToBin(message); //(length in bin format +
message) -> bin format
      const amountOfBitsInMessage = messageBinArr.length;

      let n = 0;
      const colorObj = {};
```

```
        if(key.mode === "sequential") {
            outer:
            for (var i = 0; i < image.bitmap.width; i++)
                for (var j = 0; j < image.bitmap.height; j++) {

                if(amountOfBitsInMessage === n){
                    break outer;
                }

                    var color = image.getPixelColor(i, j);
                    colorObj.r = jimp.intToRGBA(color).r;
                    colorObj.g = jimp.intToRGBA(color).g;
                    colorObj.b = jimp.intToRGBA(color).b;

                    var workChannelValue = jimp.intToRGBA(color)[key.colorChannel];
                    var resultWorkChannelValue = embed(workChannelValue,
messageBinArr[n]);
                    colorObj[key.colorChannel] = resultWorkChannelValue;

                    var newColor = jimp.rgbaToInt(colorObj.r, colorObj.g, colorObj.b,
255);

                    image.setPixelColor(newColor, i, j);

                    n++;
                }
        }
        else if(key.mode === "fixed"){
            let count = key.fixedIntervalAmount;
```

```
        outerFixed:
        for (var i = 0; i < image.bitmap.width; i++)
            for (var j = 0; j < image.bitmap.height; j++) {

                if(amountOfBitsInMessage === n){
                    break outerFixed;
                }

                if (count !== 0) {
                    count--;
                }
                else {
                    count = key.fixedIntervalAmount;

                    var color = image.getPixelColor(i, j);
                    colorObj.r = jimp.intToRGBA(color).r;
                    colorObj.g = jimp.intToRGBA(color).g;
                    colorObj.b = jimp.intToRGBA(color).b;

                    var workChannelValue =
jimp.intToRGBA(color)[key.colorChannel];
                    var resultWorkChannelValue = embed(workChannelValue,
messageBinArr[n]);
                    colorObj[key.colorChannel] = resultWorkChannelValue;

                    var newColor = jimp.rgbaToInt(colorObj.r, colorObj.g, colorObj.b,
255);
```

```
                image.setPixelColor(newColor, i, j);

                n++;
            }
        }
    }
    else if(key.mode === "random"){
        rndGenerator.seed(key.randomintervalSeed);

        let count = rndGenerator.intBetween(+key.randomintervalMin,
+key.randomintervalMax);

        outerRandom:
        for (var i = 0; i < image.bitmap.width; i++)
            for (var j = 0; j < image.bitmap.height; j++) {

                if(amountOfBitsInMessage === n){
                    break outerRandom;
                }

                if (count !== 0) {
                    count--;
                }
                else {
                    count = rndGenerator.intBetween(+key.randomintervalMin,
+key.randomintervalMax);

                    var color = image.getPixelColor(i, j);
                    colorObj.r = jimp.intToRGBA(color).r;
```

```
                colorObj.g = jimp.intToRGBA(color).g;
                colorObj.b = jimp.intToRGBA(color).b;

                var workChannelValue =
jimp.intToRGBA(color)[key.colorChannel];
                var resultWorkChannelValue = embed(workChannelValue,
messageBinArr[n]);
                colorObj[key.colorChannel] = resultWorkChannelValue;

                var newColor = jimp.rgbaToInt(colorObj.r, colorObj.g, colorObj.b,
255);

                image.setPixelColor(newColor, i, j);

                n++;
            }
        }

    }
    const dirResultImages = __dirname + "/../result-images/";
    fs.mkdir(dirResultImages, function(err){
        image.write(dirResultImages + imageFilename, function(err){
            em.emit("embedReady", imageFilename);
        });
    });

  });
}
```

```
const embeddingMethods = {
   LSBemb: LSBemb
};

module.exports = embeddingMethods;



function sizeEstimate(image, message, key){

   const bitsAmount = message.length * BITS_PER_CHAR +
BITS_FOR_MESSAGE_LENGTH;
   const pixelsAmount = image.bitmap.width * image.bitmap.height;
   let pixelsAmountNeed = 0;

   if(key.mode === "sequential"){
      pixelsAmountNeed = bitsAmount;
   }
   else if(key.mode === "fixed"){
      pixelsAmountNeed = bitsAmount * (+key.fixedIntervalAmount + 1);
   }
   else if(key.mode === "random"){
      rndGenerator.seed(key.randomintervalSeed);
      let rndSum = 0;
      for (var i = 0; i < bitsAmount; i++) {
         rndSum += rndGenerator.intBetween(+key.randomintervalMin,
+key.randomintervalMax);
      }
      pixelsAmountNeed = bitsAmount + rndSum;
   }
```

```
    return pixelsAmount > pixelsAmountNeed; //if pixelsAmount > bitsAmount
return true else return false
}

function stringToBin(message){
    let resultBinArr = [];

    const messageLength = message.length * BITS_PER_CHAR;
    const binMessageLength = messageLength.toString(2);
    let binMessageLengthArr = binMessageLength.split("");
    const diffLength = BITS_FOR_MESSAGE_LENGTH -
binMessageLengthArr.length;
    if(diffLength > 0){
        binMessageLengthArr =
(Array(diffLength).fill(0)).concat(binMessageLengthArr); //[1010] -> [0000..00
1010]
    }
    else{
        throw "Message too large for parameter BITS_FOR_MESSAGE_LENGTH";
    }

    resultBinArr = resultBinArr.concat(binMessageLengthArr);

    for (var i = 0; i < message.length; i++) {
        var char = message.charCodeAt(i);
        var binChar = char.toString(2);
        var binCharArr = binChar.split("");
```

```
        var diff = BITS_PER_CHAR - binCharArr.length;

        if (diff !== 0) {
            binCharArr = (Array(diff).fill(0)).concat(binCharArr); //[1,0,1,0] -> [0,0,0,
1,0,1,0]
        }
        resultBinArr = resultBinArr.concat(binCharArr);
    }

    return resultBinArr;
}


//embed stegoBit in sourceByte on Low Position
function embed(sourceByte, stegoBit){

    if ((sourceByte & 1) == stegoBit){ //if low bit in sourceByte and stegoBit are
equal
        return sourceByte;
    }

    return sourceByte ^ 1; //invert low bit
}

const jimp = require("jimp");
const rndGenerator = require("random-seed").create();
const BITS_PER_CHAR = require("./config").config.BITS_PER_CHAR;
const                    BITS_FOR_MESSAGE_LENGTH                    =
require("./config").config.BITS_FOR_MESSAGE_LENGTH;
```

```
function LSBextr(imageFilename, key, em, wsHandler) {
    const sourcePath = __dirname + "/../uploads/" + imageFilename;

    const messageBinArr = [];

    let progressValue = 0;

    jimp.read(sourcePath, function (err, image) {
        if (err) throw err;

        const activePixelsAmount = image.bitmap.width * image.bitmap.height -
BITS_FOR_MESSAGE_LENGTH;

        const binMessageLengthArr = [];
        let messageLengthReady = false;

        if(key.mode === "sequential") {
            let n = 0;
            outer:
                for (var i = 0; i < image.bitmap.width; i++)
                    for (var j = 0; j < image.bitmap.height; j++) {
                        var color = image.getPixelColor(i, j);
                        var workChannelValue =
jimp.intToRGBA(color)[key.colorChannel];

                        var lowBit = workChannelValue & 1;

                        if (messageLengthReady == false) {
```

```
                binMessageLengthArr.push(lowBit);
                if (binMessageLengthArr.length ==
BITS_FOR_MESSAGE_LENGTH) {
                    var binLength = binMessageLengthArr.join("");
                    var messageLength = parseInt(binLength, 2);

                    if(messageLength > activePixelsAmount){
                        em.emit("incorrectMessageLength");
                        return;
                    }

                    messageLengthReady = true;
                }
            }
            else {
                messageBinArr.push(lowBit);

                n++;

                var newProgressValue = Math.round(n / messageLength * 100);
                if (wsHandler != null) {
                    if (newProgressValue > progressValue) {
                        progressValue = newProgressValue;
                        wsHandler.send(progressValue);
                    }
                }
```